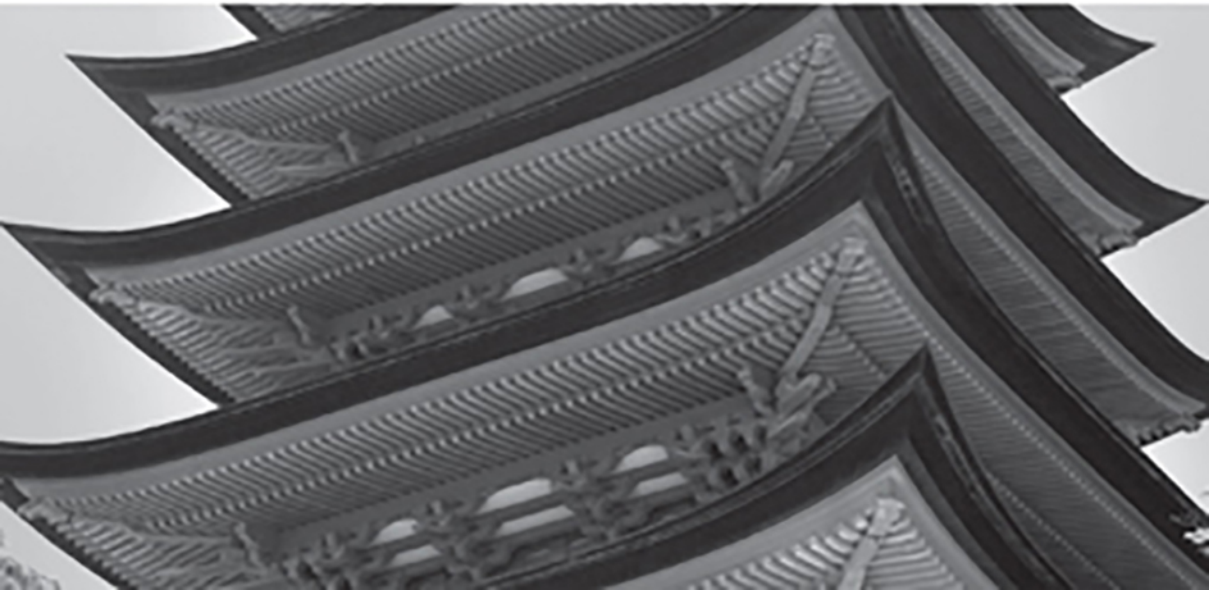
Анил Хемраджани

Предисловие
Скотта У. Амблера и Рода Джонсона

# Гибкая разработка приложений на Java™

## с помощью Spring, Hibernate и Eclipse



# Гибкая разработка приложений на Java™

## с помощью Spring, Hibernate и Eclipse

Анил Хемраджани


Москва • Санкт-Петербург • Киев
2008

ББК 32.973.26-018.2.75
Х37
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С.Н. Тригуб*
Перевод с английского и редакция *В.А. Коваленко*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com
115419, Москва, а/я 783; 03150, Киев, а/я 152

**Хемраджани**, Анил.

Х37 Гибкая разработка приложений на Java с помощью Spring, Hibernate и Eclipse. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2008. — 352 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1375-3 (рус.)

В этой книге основное внимание уделено процессу разработки приложений и в меньшей степени инфраструктуре. Другими словами, больше внимания уделено технологиям разработки приложений, таким как Spring, Hibernate и Eclipse, а не программным продуктам, таким как серверы приложений или базы данных. Все, что представлено в этой книге, опробовано в реальных приложениях, которые успешно работают (некоторые в кластеризуемой среде сервера приложений). Одна из задач этой книги заключается в краткости и конкретности, поэтому автор решил практически полностью сосредоточиться на разработке хорошо масштабируемого приложения. В данной книге, кроме технологий Spring, Hibernate и Eclipse, также описаны альтернативные и конкурирующие технологии.

**ББК 32.973.26-018.2.75**



ISBN 978-5-8459-1375-3 (рус.)
ISBN 0-672-32896-8 (англ.)

# Оглавление

# Содержание

# Предисловие

Настоящая задача предисловия — убедить людей купить книгу. Как я полагаю, сейчас вы имеете три выбора. Во-первых, вы можете сэкономить время и прекратить читать это предисловие прямо сейчас, просто поверив мне на слово, что покупка этой книги является хорошим вложением ваших тяжело заработанных денег. Во-вторых, можете не поверить мне на слово, поскольку недоверие к консультантам вполне законно, и продолжить читать это предисловие, где я мог бы написать что-нибудь резонирующее с вашими интересами и, таким образом, мотивировать вас купить книгу. В-третьих, вы можете отказаться от приобретения книги, что будет, вероятно, плохой идеей, поскольку подавляющее большинство программистов Java, которых я знаю, могли бы действительно извлечь пользу из этого материала.

Последняя вещь, которая, на мой взгляд, нужна сообществу разработчиков Java, — это очередная книга, описывающая некоторые "действительно замечательные" технологии Java, поскольку этим темам посвящены уже сотни книг, тысячи статей в журналах и десятки тысяч Web-страниц. К счастью, это не относится к данной книге. Это одна из тех редких книг, которые содержат знания, используемые разработчиками в практической деятельности. Да, здесь рассматриваются основные принципы таких технологий, как Spring, Hibernate, Ant и других, необходимых для того, чтобы преуспеть сегодня. Однако важнее всего то, что здесь описаны гибкие (Agile) методы программирования, адаптированные к экстремальному программированию (XP) и гибкому моделированию, позволяющим преуспеть в современной разработке программного обеспечения. Большинство разработчиков Java слышали об XP, и многие переняли некоторые из его методов, такие как Test-Driven Design (TDD), рефакторинг и даже парное программирование. Это хорошо для начала, но вовсе не достаточно. В этой книге Анил суммирует годы своего опыта практической работы. Данная книга отличается от других, авторы которых зачастую имеют лишь теоретическое видение проблем; но как мы все знаем, теория и практика — это зачастую две совершенно разные вещи.

Когда Анил впервые обратился ко мне с предложением стать техническим рецензентом этой книги, меня поразила простота и эффективность описания подходов моделирования в проектах Java. Фактически, вы можете пролистать книгу прямо сейчас и сразу изучить некоторые из этих моделей. Полагаю, вы обратите внимание на то, что его схемы очень похожи на те, которые вы разрабатываете для своих настоящих проектов, в отличие от представленных в большинстве современных книг по моделированию. Вы обратите также внимание на то, как Анил описывает перенос этих простых моделей в зачастую достаточно сложный код, подобный тому, который вы пишете ежедневно. Но самое большое преимущество данной книги, на мой взгляд, это отражение реального практического опыта высоко профессиональной разработки.

Книга описывает также множество общих задач, которые мы выполняем в ходе разработки программного обеспечения, включая входной контроль, проверку модулей, объектное и реляционное сопоставление, интеграцию системы и рефакторинг. Большинство книг отстаивает тот или иной взгляд, но здесь внимательно рассматриваются оба (объективный и субъективный). Уделите несколько минут просмотру остальной части этой книги, и, полагаю, вы поймете, что я имею в виду.

**— Скотт У. Амблер**
Технический директор, Agile Modeling

Категоризировать эту книгу нелегко. Позвольте мне объяснить, почему она так необычна и почему заслуживает вашего внимания.

Легко категоризировать те книги, которыми наша область деятельности изобилует. Зачастую это книги о некоем вполне определенном продукте или API. Одни из них хороши; другие плохи. Вы можете выбирать их по обложке, шрифту, издателю, квалификации автора, однако вы уже сделали гораздо более важный выбор: *вы знаете, какую книгу ищете.* Хорошая книга поможет вам эффективнее работать в определенной области, но она вряд ли изменит ваш способ работы.

Книги, которые нелегко категоризировать, более редки. Они намного ближе к их автору и, потенциально, к вам.

Анил Хемраджани обладает обширным опытом архитектора и разработчика, поэтому его книга изобилует примерами успешного практического проектирования. Подобно всем лучшим книгам, эта обладает эффективной связью между автором и читателем, что вполне объяснимо, ведь Анил обладает столь естественным стилем изложения, что читать его книгу доставляет удовольствие.

Эта книга имеет широкий контекст. Здесь затронуты темы, которые редко обсуждаются вместе, а надо бы. Процесс, который мы используем для разработки программного обеспечения, неразрывно связан со способом, которым мы структурируем наш код, и инструментами, которые мы используем для написания этого кода. Ни один из профессиональных разработчиков не работает обособленно, многие решения, которые они принимают, являются совместными, однако в большинстве книг даже не предпринимается попытка раскрыть общую картину, что весьма важно для получения положительных результатов.

Чтобы продуктивно разрабатывать программное обеспечение Java сегодня, необходимо понять ключевые концепции, такие как сопоставление O/R и Dependency Injection; необходимо также понять, как и почему используются такие методы, как проверка модуля и автоматизированное создание; и, что не менее существенно для работы, необходимы наилучшие инструменты, такие как среды выполнения и IDE. Вам также нужно разобраться в некоторых из проблем, чтобы избегать их, поскольку такие знания иногда не менее важны, чем технологии.

Анил выполнил замечательную работу, объединив все это в одной книге, коротко и ясно описав те области, которую многим непонятны. Здесь рассматривается много основ, но никогда не теряется из виду цель — помочь читателям успешно завершать проекты. На мой взгляд, эта книга как хорошая карта, она ясно демонстрирует вам путь к успешному завершению разработки проекта Java. При этом вы можете дополнять ее более подробными картами специфических областей, но всегда следовать основному направлению, которое она предоставляет.

Я рекомендую эту книгу любому, кто сейчас занимается Java практически. Она может полностью изменить способ, которым вы работаете.

**— Род Джонсон**
CEO, Interface21
Основатель Spring Framework

# Об авторе

**Анил Хемраджани** работает с технологиями Java с 1995 года как разработчик, предприниматель, автор и преподаватель. Он основатель успешной корпорации Isavix Corporation, а также такого ресурса, как `isavix.net` (ныне `DeveloperHub.com`), — знаменитого сетевого сообщества разработчиков, которое выросло до более чем 100 000 зарегистрированных участников. Анил обладает двадцатилетним опытом в области информационных технологий, а также сотрудничества с несколькими сотнями крупных компаний. Он опубликовал множество статей в известных журналах и выступал на конференциях и семинарах по всему миру. Анил был удостоен наград за выдающиеся заслуги в расширении сообщества Java от Sun Microsystems, за лучшее клиентское решение Java для BackOnline от JavaOne, за лучший сетевой клиент-серверный продукт на Java, а также был номинирован на Смитсоновскую премию за лучшее сетевое хранилище файлов для службы Web-сайта. Его последняя работа в этой области — Web-сайт `visualpatterns.com`.

*Посвящается моей любящей и заботливой жене, которая всегда терпеливо поддерживала меня (и мои начинания). Спасибо за заботу обо всем, в то время когда я был занят такими проектами, как эта книга. Без тебя эта книга была бы невозможна! Благодарю также моих дорогих детей.*

# Благодарности

Я написал эту книгу, но это не было бы возможно без помощи многих блестящих людей, которых я имею удовольствие знать. Я хочу поименно поблагодарить каждого из них!

- Клиф Берг (Cliff Berg). В первую очередь, я хочу поблагодарить Берга, моего давнего друга и коллегу, который вдохновил меня на написание этой книги и поддержал мои идеи.
- Скотт У. Амблер (Scott W. Ambler). Благодарю за участие в создании предисловия к этой книге, а также за просмотр каждого ее элемента, глава за главой. Скотт стал одним из моих кумиров в этой области, благодаря своим работам по Agile Modeling и Agile Data, так что я до сих пор поражаюсь, как он согласился стать соавтором предисловия, с учетом его чрезвычайной занятости. Я также хочу благодарить Скотта за Web-сайты `www.agilemodeling.com` и `www.agiledata.org`; они были неоценимы для меня. Скотт, спасибо!
- Род Джонсон (Rod Johnson). Впервые встретив Рода, я еще понятия не имел, что буду писать эту книгу или просить его писать предисловие. Я знал только то, что наши взгляды совпадали, и я уважал его работы. С учетом популярности Spring Framework в наши дни, я удивлен, что Род находит время поспать, поэтому его предисловие для моей книги особенно ценно для меня (спасибо!). Я также хочу поблагодарить Рода за советы по системе Spring Framework, с которой мне так нравится работать.
- Анил Сингх (Anil Singh). Я не могу выразить свою благодарить Анилу за его помощь; долгие часы обсуждения содержимого моих глав до ночи по телефону (иногда до 3 часов) неоценимы, спасибо за все! Я особенно ценю доступность Анила почти в любой момент, когда мне нужно было обсудить книгу. Однако все, что я могу предоставить взамен, — это дружеское *спасибо*!
- Дэн Шеллман (Dan Shellman). Спасибо за *оперативные*, но подробные и *честные* отзывы, которые помогли сделать эту книгу существенно лучше. Спасибо за терпение, в ходе создания этой книги я послал Дэну больше сотни сообщений по электронной почте. Я особенно ценю ваши советы, а также телефонные диалоги в уик-энды и даже семейные праздники! Дэн мой давний друг и коллега, и я надеюсь, что он останется таковым на долгие годы.
- Хареш Лала (Haresh Lala). Спасибо за поддержку во всем и всегда! Благодарю за проверку всего моего кода (дважды!), а также чтение моих глав начиная с самых ранних, довольно сырых, версий. Однако больше всего благодарю за помощь в то время, когда вы были очень заняты переменами в своей жизни.
- Хернандо Вера (Hernando Vera). Что бы я делал без проницательных суждений Хернандо, его отточенных идей и хорошо продуманных интерпретаций технологий? Хернандо был одним из моих близких друзей на протяжении почти десятилетия. Когда я в чем-нибудь сомневаюсь, знаю, что могу обратиться к нему за советом о разработке, архитектуре, процессе и многом другом. Я не встречал никого, кто обладал бы столь полным набором достоинств.
- Мартин Реммелзваал (Martin Remmelzwaal). С Мартином я встретился относительно недавно, но уже считаю его близким другом. Благодарю за просмотр мо-

их ранних версий глав. Особенно хочу поблагодарить за ответы на мои бесконечные вопросы по электронной почте о перспективах различных технологий и методологий, а также по темам, связанным с ними. Я надеюсь сотрудничать с Мартином и в будущем.

- Группа Spring Framework. Для начала, вы просто великолепные парни! А теперь поименно. Я хочу поблагодарить Алефа Андерсена (Alef Arendsen) за его обзор глав 6 и 7, а также помощь во всем, что относится к Spring; обзоры Алефа существенно улучшили главы 6 и 7. Я хочу также поблагодарить Юргена Хойлера (Juergen Hoeller) за его помощь в декларативном управлении транзакциями и недавнем обсуждении интерфейсов. С учетом загруженности группы Spring (помощь клиентам, ночные бдения над средой выполнения и подготовка нескольких выпусков одновременно), мне остается только сказать, *спасибо, парни!*
- Мадху Сиддалингаях (Madhu Siddalingaiah). Благодарю за руководство в создании иллюстраций этой книги (и других вопросах публикации), а также за помощь по главе 8.
- Дейв Берман (Dave Berman). Благодарю за содержательный обзор главы 2 и обсуждение различных вопросов и методов Agile, что помогло мне сделать аспекты Agile/XP в этой книге более полными и содержательными.
- Джеф Нильсен (Jeff Nielsen). Спасибо за советы по схемам для глав 2 и 3, а также комикс по XP и AMDD, приведенный в этой книге. Вы помогли мне устранить несколько существенных ошибок в самый последний момент, спасибо, Джеф!
- Рамананд Сингх (Ramanand Singh). Спасибо за первоначальные обсуждения среды Spring, а также советы по главе 6.
- Персонал Pearson. Я хочу поблагодарить персонал компании Pearson (Сонглин, Менди, Ким, Марк, Барбара и многих других) за создание этой книги; особая благодарность Дженни, за ее участие с начала и до конца. Я также хочу поблагодарить Бориса за его глубокий и откровенный обзор этой книги. Без его советов эта книга не была бы тем, чем она является.
- Благодарю моего друга Питера за интеллектуальную поддержку, которая помогла мне выразить в этой книге уникальные взгляды. Благодарю также Энди и Мисси за их шутки, которые помогли мне немного расслабиться, когда я в этом нуждался (особенно в период непрерывной работы над этой книгой, зачастую по 14–15 часов в день).
- Благодарю кафе Greenberry и персонал чайной за обеспечение удобной рабочей обстановки, быстрый Интернет и великолепный кофе с закусками; все они позволили мне работать там в течение многих часов над этой книгой.
- И наконец, но не в последнюю очередь, благодарю тех многих специалистов нашей области, творческая работа которых легла в основу этой книги. Я хотел бы поблагодарить этих людей за косвенную помощь, они опубликовали некоторые удивительные и неоценимые концепции. Я имею в виду Мартина Фаулера (Martin Fowler), Кента Бека (Kent Beck), Эриха Гамма (Eric Gamma), Варда Каннингема (Ward Cunningham) и др.

# Введение

Я начал работать с технологиями Java еще начиная с 1995 года, вскоре после выпуска *комплекта разработки Java* (Java Development Kit – JDK) версии 1.0. До этого я много лет программировал на языках C и C++. Я был на самом деле восхищен такими возможностями Java, как межплатформенная переносимость, простой синтаксис (более простой, чем у языка C++, например), ориентированность на объекты, безопасность, богатство API и т.д.

На протяжении 20-летней карьеры мне пришлось изучить многое. Поэтому мне всегда нравилась простота; когда я встречаю сложность, начинаю сомневаться, является ли решение правильным. Именно это как раз я и почувствовал в 2000 году, когда язык Java 2 Enterprise Edition (J2EE) начал становиться господствующим. Обратите внимание, с этого момента я буду называть версию J2EE как JEE, поскольку цифра "2" была недавно ликвидирована из имени компанией Sun Microsystems.

Мой интерес к Java снизился в результате того, что я заметил ненужную сложность в JEE, обусловленную слоями абстракции. Я начал полагать, что компания Sun Microsystems (разработчик Java) предназначила Java и JEE для создания наиболее сложных корпоративных приложений, несколько проигнорировав относительно менее сложные приложения малого и среднего размера. Кроме того, я с разочарованием наблюдал потерю у людей здравого смысла, поскольку не раз натыкался на проекты, в которых среда Enterprise JavaBeans (EJB) использовалось вовсе не для распределенной обработки, например для локальной регистрации. По этому поводу в 2000 году я написал короткую статью для журнала JavaWorld.com (`http://www.javaworld.com/javaworld/jw-10-2000/jw-1006-soapbox.html`) под названием "Do You Really Need Enterprise JavaBeans?" (Действительно ли вам нужна Enterprise JavaBeans?). (Примерно пять лет спустя мы заметили переработку спецификации EJB 3.0, она была сделана проще, чтобы облегчить разработку.) Это подводит нас к содержанию этой книги и причине, по которой я ее написал.

Я был недавно приглашен компанией Fortune 100 в качестве консультанта при создании корпоративного Web-приложения, работающего в кластеризуемой среде. Рассматривая альтернативу стандартной модели JEE/EJB, я исследовал сетевые ресурсы и беседовал с компетентными людьми. В результате я остановился на решении, которое подразумевало использование Web-среды Spring MVC, отказоустойчивой объектно-реляционной (Object-Relational – OR) среды Hibernate, среды IDE Eclipse, среды проверки JUnit, утилиты Ant, нескольких библиотек дескрипторов и других продуктов. (Более подробная информация об этих продуктах приведена в книге далее, наряду с моими объяснениями по поводу выбора этих технологий.)

Я люблю работать с системами Spring и Hibernate, главным образом, потому, что они позволяют работать со *старыми простыми объектами Java* (Plain-Old Java Object — POJO) и избегать некоторых из проблем работы с EJB. Кроме того, работая с IDE Eclipse, я получил хороший опыт. Мне бы хотелось еще поговорить о том, как хорош этот продукт, именно поэтому я посвящаю данной теме целую главу. По моему мнению, такие продукты, как упомянутые выше, вдыхают новую жизнь в язык Java, когда он подвергается опасности потери популярности в связи с появлением таких продуктов, как .NET от Microsoft, LAMP (Linux, Apache, MySQL и PHP или Python/PERL), а также Ruby on Rails.

Хотя в этой книге и описаны технологии Spring, Hibernate и Eclipse, основная моя задача заключается в том, чтобы снабдить вас полным решением с технической точки зрения и с точки зрения процесса. С технической точки зрения я предоставляю сквозное решение (с использованием ряда инструментальных средств) для полной реализации типового Web-приложения с фоновым управлением транзакциями и подходящего для кластеризуемой среды. С точки зрения процесса, я недавно перешел с использования процесса Rational Unified Process (RUP) на процесс, состоящий из критериев моделей Agile Model Driven Development (AMDD; `agilemodeling.com`) и Extreme Programming (XP; `extremeprogramming.org`). В этой книге вы найдете описание концепций, а также такие артефакты, как статьи пользователей, планы выпуска, карточки CRC и многое другое. Идея заключается в том, чтобы предоставить вам исчерпывающее решение для быстрой разработки и развертывания приложений Java.

Еще одно, дополнительное, замечание по поводу моей квалификации. Я обладаю почти 20-летним опытом разработки с использованием таких фундаментальных технологий, как C/C++, Java, корпоративные реляционные базы данных, серверы приложений, Unix, Microsoft Windows и т.д. Однако в моей работе был приблизительно пятилетний перерыв, с 1996 по 1998 год, поскольку я основал собственную компанию и программировал в это время мало. Позже я продал компанию, чтобы вернуться к разработке. Даже при том, что в этой компании я был руководителем высшего ранга и имел несколько подчиненных, работавших на меня, я находил возможность разговаривать буквально с сотнями разработчиков и обсуждать с ними технологические новшества. Кроме этой компании, я также основал сетевое сообщество для разработчиков Java, которое разрослось до более чем 100 000 участников (я очень горжусь этим). Я надеюсь, что мой опыт этих предприятий добавит уникальную привлекательность этой книге. Надеюсь, что вы найдете эту книгу полезной и вам понравится читать ее!

# Для кого написана эта книга

Эта книга подразумевает, что вы имеете некоторые практические навыки работы с Java и реляционными базами данных (включая SQL), а также опыт работы с командной строкой.

Эта книга предназначена также для следующих категорий читателей.

- Разработчики и архитекторы программного обеспечения. Они могут почерпнуть из этой книги знания о процессе разработки программного обеспечения высокого уровня, проектировании приложений, а также о глубокой и полной проверке кода Java и связанных с ними файлов на предмет полной функциональности на примере типичного корпоративного Web-приложения.

- Технические руководители и менеджеры. Руководители, обладающие опытом программирования, предпочтительно в Java или подобном языке, могут получить углубленное представление о создании приложений с использованием ряда технологий Java. Эти знания окажутся весьма полезны на этапе планирования проекта или при решении персоналом технических проблем (когда сроки поджимают; возможно, только для моральной поддержки). В качестве альтернативы технические менеджеры могут углубиться в определенную главу (например, главу 5, "Применение Hibernate для постоянных объектов"), чтобы понять, как та или иная технология работает и вписывается в общую картину.

Кроме того, из этой книги вы можете получить некоторое представление об альтернативах JEE, которые можно использовать для создания надежных приложений корпоративного класса. Если вы не знакомы с технологиями Agile Modeling и Extreme Programming или подбираете более легкий процесс разработки программного обеспечения, то этой книги могло бы оказаться достаточно, чтобы получить полное представление о разработке программных приложений.

# Задачи этой книги

При написании этой книги решались следующие задачи.

- Гибкая разработка. Приоритетная задача этой книги заключается в том, чтобы продемонстрировать вам быстрые способы разработки приложений Java. Это достигается за счет комбинации нескольких аспектов: продуманного и минимизированного процесса разработки программного обеспечения, простого проекта (это снижает необходимость использования схем проектирования или слоев абстракции), применения вспомогательных технологий (типа Spring и Hibernate), работы с POJO вместо дистанционных объектов, а главное, манипуляций классическими технологиями реализации с открытым исходным кодом, когда это возможно. Короче говоря, идея заключается в том, чтобы сделать приложения Java по возможности более простыми и быстрыми в разработке.
- Полное решение. Вторая задача этой книги заключается в том, чтобы снабдить вас полным решением с технической точки зрения и с точки зрения процесса. По завершении изучения этой книги, вы должны быть способны создавать любые приложения, не только технически, но и с использованием процесса, описанного в этой книге. Кроме того, когда я не могу описать какую-либо технологию достаточно глубоко, предоставляю ссылки на ресурсы (Web-сайты) для дальнейшего ее исследования. Замечательной особенностью технологий, рассматриваемых в этой книге, является то, что вы можете получить полную систему, от пользовательского интерфейса до встроенной базы данных, наряду с возможностью планирования работ (благодаря среде Spring Framework), в одном единственном саморазворачивающемся файле архива Web-приложения (`.war`)! Но вы можете в любой момент заменить упомянутые здесь технологии некоторыми другими, по своему выбору (например, использовать базу данных Oracle вместо HSQLDB). В результате, вы будете иметь полное решение, как с технической точки зрения, так и с точки зрения процесса!
- Использование решений исключительно за счет реализации с открытым исходным кодом не является задачей этой книги. Хоть я и старался использовать

в этой книге среды, инструменты и продукты только с открытым исходным кодом, проповедь реализации решений исключительно с открытым исходным кодом также не является задачей этой книги. Например, вы вполне можете отойти от требований производителя Java по переносимости и заменить один из описанных здесь продуктов каким-либо коммерческим. Однако реализация с открытым исходным кодом прошла очень долгий путь, и я просто поражен тем, насколько надежными оказались эти технологии и насколько хорошо они себя зарекомендовали. Например, такие технологии, как SDK Eclipse и Hibernate, на мой взгляд, значительно лучше, чем некоторые из их коммерческих дубликатов. Точно так же для корпоративного решения вы можете использовать все упомянутые в этой книге технологии и надеяться, что все остальные сработают так, как указано в их рекламе. Фактически я недавно реализовал корпоративное решение для большой компании, используя такие инструменты, как Spring Framework, Hibernate, Eclipse, JUnit, Ant и другие, упомянутые в этой книге! Но мы также использовали и коммерческие продукты, такие как WebLogic Server от BEA и сервер баз данных Oracle. Упомянутая выше компания (и несколько других, известных мне) основывает свои корпоративные решения на реализации технологий с открытым исходным кодом.

- Быстрое чтение. Эта книга преднамеренно имеет объем меньше, чем у типичной 600-страничной книги по Java. Это было сделано для того, чтобы позволить вам прочитать книгу быстро и сразу начать использовать описанные в ней решения в реальном мире. В свете сказанного выше я постарался сделать содержимое этой книги как можно более рациональным и близким к сути дела. Одним из недостатков написания весьма "нетолстой" книги является то, что я вынужден был зачастую принимать жесткие решения о предоставляемом материале; однако я постарался все же включить побольше материала, связанного с процессом и технологией, в котором вы будете нуждаться при быстрой разработке приложений Java (как уже упоминалось в предыдущем абзаце, для предоставления полного решения).
- Простота. Каждый раз, когда это возможно, я предпочитаю изложить более простой подход выполнения задачи. Например, примеры рассматриваемых в этой книге приложений используют минимальные слои абстракции для достижения цели. Под слоями абстракции я подразумеваю чрезмерное использование схем проектирования, интерфейсов и дробления приложений. Каждый из этих элементов имеет огромный смысл, но их уместное использование — это хорошая практика, и я ей, как правило, следую, когда разрабатываю программное обеспечение. Кроме того, я полагаю, что простота должна также распространяться на проектирование, в котором я предпочитаю использовать UML, когда это необходимо, но в более простых случаях я предпочитаю использовать для диаграмм такие инструментальные средства, как OpenOffice.org, PowerPoint или Visio, а не что-нибудь столь сложное, как Rational Rose.
- Советы и приемы. Как вы могли уже заметить, при работе с инструментальными средствами и технологиями советы и приемы позволяют не только эффективнее использовать программный продукт, но и проще решать некоторые задачи. Для некоторых технологий, описываемых в этой книге, я привожу советы и описания приемов. Но и приложения сами содержат некоторые вспомога-

тельные средства, такие как контрольные списки и списки особенно полезных инструментов.

- Альтернативы. В этой книге (хоть и без подробностей) я попытаюсь предоставить альтернативы стандартным решениям. Я понимаю, что не существует одного решения для всех случаев. Например, в качестве IDE вы могли бы использовать NetBeans от Sun Microsystems или IntelliJ от JetBrains, а не Eclipse. Это вполне ожидаемо и полностью понято. Организация этой книги учитывает такой случай, и вы вполне можете извлечь пользу из остальной части книги, заменив технологию, описанную здесь, на технологию по вашему выбору (например, JDO вместо Hibernate).

# Что не описано

Эта книга подразумевает, что вы имеете практические знания в области Java и относительно хорошее понимание JEE. Здесь также в значительной степени подразумевается, что вы имеете приемлемое понимание процессов разработки программного обеспечения, реляционных баз данных, многоуровневых архитектур и т.д. С учетом этого, я сразу перехожу к вопросам, связанным с созданием нашего типового приложения. Кроме того, я считаю, что вам доступны соответствующие Web-сайты (и дополнительные средства) и не дублирую здесь эту информацию, рискуя выпустить устаревшую книгу.

С другой стороны, эта книга подразумевает, что вы не имеете никаких практических знаний об основных описанных здесь технологиях, таких как Spring Framework, Hibernate, Eclipse и т.д. С учетом сказанного выше, данная книга содержит изложение основ этих технологий и их совместной работы; эта книга делает также следующий шаг и обеспечивает вас кратким вводным курсом по некоторым из дополнительных возможностей, предоставляемых этими технологиями. Все, что не упомянуто здесь, выходит за рамки этой книги, поскольку существует множество других книг, посвященных большинству технологий, упомянутых в данной книге.

# Что описано (технологии и процесс)

В этой книге основное внимание уделено разработке и в меньшей мере инфраструктуре. Другими словами, я уделил больше внимания технологиям разработки приложения, таким как Spring, Hibernate и Eclipse, а не программным продуктам, таким как серверы приложений (например, JBoss) или базы данных (например, MySQL). Кроме того, я предпочитаю сначала получить функциональные возможности реализуемого приложения, а затем осуществить его оптимизацию и рефакторинг. Все, что представлено в этой книге, опробовано в реальных приложениях, которые успешно работают (некоторые в кластеризуемой среде сервера приложений), поэтому мне бы не хотелось, чтобы у вас создалось впечатление, что мы игнорируем инфраструктуру в целом. Одна из задач этой книги заключается в краткости и конкретности, поэтому я решил практически полностью сосредоточиться на разработке хорошо масштабируемого приложения.

С учетом преимуществ переносимости на операционные системы (Operating System — OS), в теории, когда ваше приложение будет готово и развернуто, вы можете перевести его на более надежный сервер Web-приложения в комбинации с базой дан-

ных. Например, сначала вы могли бы использовать относительно маломощные продукты, описанные в этой книге (Tomcat от Apache и HSQLDB), а затем перейти на комбинацию сервера JBoss Application Server и базы данных MySQL или наращивать вычислительные возможности далее, до комбинации серверов BEA WebLogic Server и Oracle, например. В этом красота Java; т.е. не только переносимость на другую OS, но и возможность смены производителя.

Еще одно замечание об основных технологиях, описанных в этой книге, а именно Spring, Hibernate и Eclipse. Хотя именно эти технологии я использовал в данной книге, я также упомянул альтернативные и конкурирующие технологии, которые советую вам также рассмотреть. Например, если вы решили использовать JDO вместо Hibernate, вполне можете использовать содержимое всех остальных глав, за исключением, возможно, посвященных Hibernate.

# Как организована эта книга

Эта книга организована так, чтобы каждая последующая глава основывалась на материале предыдущей. Кроме того, поскольку главы логически разделены, вы можете непосредственно перейти к определенной главе (например, главе 6, "Обзор среды Spring Framework") и прочитать только ее содержимое. Кроме того, вы можете пропустить некоторую главу, если использование описанной в ней технологии вас не интересует (например, если, вместо Eclipse, вы используете NetBeans, главу 8, "Феномен Eclipse!", можно пропустить). Глава 1, "Введение в разработку приложений на Java", содержит краткий обзор и предварительное описание технологий и процессов, используемых в этой книге. Глава 2, "Простое приложение: сетевая система учета рабочего времени", посвящена, в основном, определению бизнес-требований для нашего типового приложения; но она содержит также краткий обзор методологий AMDD и XP. В главе 3, "Архитектура и модель проекта на базе XP и AMDD", рассматривается проект нашего типового приложения. В главе 4, "Установка среды: JDK, Ant и JUnit", описана установка среды.

Теперь мы быстро переходим в мир кода Java и рассматриваем в главе 5, "Применение Hibernate для постоянных объектов", программирование с использованием среды Hibernate. Главы 6, "Обзор среды Spring Framework", и 7, "Среда Spring Web MVC Framework", посвящены среде Spring Framework. Главы 7, "Среда Spring Web MVC Framework", и 8, "Феномен Eclipse!", — это то, что я называю "oro!", поскольку здесь все собирается вместе и вы понимаете, почему мы прошли предыдущие главы именно так, как это было сделано. Вы узнаете, что происходит внутри, а следовательно, получите представление об основе таких технологий, как Spring, Hibernate, Ant и JUnit. Далее мы рассмотрим некоторые дополнительные концепции и закончим книгу приложениями, содержащими полезные сведения.

Еще одно замечание по поводу использования при разработке командной строки, а не GUI (например, использования SDK Eclipse). В первых главах командная строка используется преднамеренно для того, чтобы вы могли получить некоторое представление об основных правилах работы этих инструментальных средств Java. Зато впоследствии, когда будете использовать эти инструменты (например, Ant и JUnit) в IDE типа Eclipse, вы будете точно знать, что происходит за кулисами. Это становится особенно важным, если IDE не удовлетворяет вашим потребностям.

# О коде для этой книги

Эта книга о программировании в Java, поэтому она содержит большое количество кода, относящегося к примерам приложений. Данное полнофункциональное приложение со всем исходным кодом и связанными файлами может быть разгружено с Web-сайта издателя (`http://www.samspublishing.com/content/images/0672328968/downloads/0672328968_AgileJavaDev_code.zip`). Получив код в исходной электронной форме, вместо подвергшегося многократной правке и напечатанного в книге, можно избежать возможных опечаток.

Код в книге рассматривается в виде фрагментов полного кода. Кроме того, для этого кода используются два соглашения:

- В некоторых местах, когда строка кода слишком длинная (т.е. больше чем 82 символа), для удобочитаемости она разделяется на две строки символом ↵.
- Код представлен моноширинным шрифтом.

**Примечание**

Весь код книги доступен для загрузки на Web-сайте издателя. Для удобства доступа к файлам примеров этой книги и исходному коду, а также к любым возможным обновлениям или исправлениям, зарегистрируйтесь по адресу `www.samspublishing.com/register`.

# Об иллюстрациях для этой книги

Как вполне справедливо подмечено, лучше один раз увидеть, чем сто раз услышать, поэтому в этой книге вы найдете множество схем и снимков экрана. Хотя я и привел в этой книге несколько диаграмм UML, разрабатывая приложение, я предпочитаю более простые схемы в свободной форме. Я не трачу много времени на формальные схемы, поскольку большинство из них становится не нужно, после того как они выполнили свою задачу, а кроме того, в большинстве проектов такие схемы даже на это не способны. По моему глубокому убеждению, код и база данных в окончательном виде значительно важнее других артефактов.

Таким образом, я полагаю, что схемы должны удовлетворять следующим требованиям:

- Схемы нужны только тогда, когда они эффективны.
- Схемы должны быть понятны (используйте только понятные слова, применяйте легенды).
- Схемы должны быть просты и действительно отражать суть дела.
- Придерживайтесь стандартной формы записи (например, UML), если таково требование вашей организации или если схему предстоит передать другому человеку или группе для создания объектного кода или просто потому, что вы предпочитаете использовать стандартную форму записи.

Еще одно замечание об иллюстрациях в этой книге. В некоторых главах я несколько раз использовал иллюстрации из предыдущих глав. Это было сделано по следующим причинам.

- Чтобы задать контекст главы или раздела, который предстоит обсудить.
- Мне не хотелось беспокоить вас требованиями вернуться к предыдущей главе, чтобы посмотреть приведенную там иллюстрацию.

# Вставки с личным мнением

Текст книги содержит многочисленные вставки, озаглавленные "Личное мнение", – именно таково содержимое этих вставок. Я постарался отделить этот субъективный материал от объективного материала книги. Однако надеюсь, что вам моя точка зрения, выраженная в этих разделах, будет полезна, поскольку в них содержатся обобщения моего личного опыта как: инженера по программному обеспечению, консультанта, преподавателя, автора, создателя сетевого сообщества и даже руководителя компании, занимающейся технологиями IT.

# Комикс

В начале каждой главы вы увидите иллюстрации, составляющие вымышленную историю о восьминедельном проекте с использованием AMDD. Четыре главных действующих лица этой истории, включая клиента (Сьюзен), руководителя проекта (Рон) и двух программистов (Стив и Радж), также вымышленные. Цель этих иллюстраций проста: придать книге немного юмора. Кроме того, свободный стиль этих иллюстраций – следствие моего опыта написания книг для детей (моих первых читателей). Предупреждаю, юмор немного банальный, но я надеюсь, что вы его оцените.

Если стиль этих иллюстраций вам понравился и вы хотите посмотреть их в цвете, посетите, пожалуйста, Web-сайт `visualpatterns.com`.

# Рекомендуемые ресурсы

Здесь описано множество технологий, однако, с учетом характера этой книги, о каждой технологии будет предоставляться только та информация, которая достаточна для практической работы. Но каждая из описанных здесь технологий сама по себе заслуживает отдельной книги, и такие книги действительно имеются (во всяком случае по большинству технологий).

Поэтому ниже приведены Web-сайты, посвященные основным технологиям, описанным в этой книге. Каждый из них предоставляет дополнительную информацию (а в некоторых случаях и форумы для обсуждения) по соответствующей технологии:

- Agile Modeling `http://www.agilemodeling.com`
- Ant `http://ant.apache.org/`
- Eclipse SDK `http://eclipse.org`
- Extreme Programming `http://extremeprogramming.org`
- Hibernate Framework `http://hibernate.org`
- Механизм баз данных HSQLDB `http://hsqldb.org/`
- JUnit `http://junit.org`
- Spring Framework `http://springframework.org`
- Visual Patterns `http://visualpatterns.com`

В конце каждой главы я буду предоставлять ссылки на соответствующие ресурсы, так что к концу этой книги у вас будет хорошая коллекция ресурсов для дальнейшего изучения!

# Соглашения, принятые в книге

При оформлении книги использованы соглашения, общепринятые в компьютерной литературе.

- Новые термины в тексте выделяются *курсивом*. Чтобы обратить внимание читателя на отдельные фрагменты текста, также применяется *курсив*.
- Текст программ, функций, переменных, URL Web-страниц и другой код представлен моноширинным шрифтом.
- Все, что придется вводить с клавиатуры, выделено **полужирным моноширинным** шрифтом.
- Знакоместо в описаниях синтаксиса выделено *курсивом*. Это указывает на необходимость заменить знакоместо фактическим именем переменной, параметром или другим элементом, который должен находиться на этом месте `BINDSIZE=(`*`максимальная ширина колонки`*`)*(`*`номер колонки`*`)`.
- Пункты меню и названия диалоговых окон представлены следующим образом: Menu Option (Пункт меню).

# От издательства

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что бы еще вы хотели увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать авторам.

Мы ждем ваших комментариев. Вы можете прислать письмо по электронной почте или просто посетить наш Web-сервер, оставив на нем свои замечания, — одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более подходящими для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш e-mail. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию следующих книг. Наши координаты:

E-mail: `info@williamspublishing.com`

WWW: `http://www.williamspublishing.com`

# I

# Краткий обзор

# 1

# Введение в разработку приложений на Java


**Сьюзен:** Я получила от руководства одобрение на создание первого выпуска этого приложения.

**Рон:** Хорошая новость; я предложу моим программистам, Стиву и Раджу, начать этот проект на следующей неделе; они предпочитают работать вместе.

*Комплект разработчика Java* (Java Development Kit – JDK) версии 1.0, выпущенный в январе 1996 года, имел довольно простой *интерфейс прикладных программ* (Application Programming Interface – API). За последние годы Java развилась в полнофункциональную платформу. Комплект JDK версии 1.5, по сравнению с версией 1.0, имеет множество новых средств, таких как Java Collections Framework, API регистрации, автоупаковку, встраивание и многое другое. Хотя большинство из этих средств весьма полезно, разработка в Java стала более сложной, особенно после появления платформы Java Platform Enterprise Edition (JEE). Платформа JEE представила такую концепцию, как Enterprise JavaBeans (EJB), которая была призвана упростить переносимость и распределенные вычисления на корпоративном уровне, но вместо этого создала ненужные сложности для 80% приложений. Нет ничего удивительного в том, что сейчас многие считают технологию Java/JEE тяжелой и громоздкой. И хоть это не далеко от истины, давайте посмотрим, не удастся ли нам в данной книге изменить такое мнение.

За прошедшие несколько лет многие системы реализации с открытым исходным кодом оказались способны решить некоторые из проблем, созданных JEE. В этой книге описаны некоторые из этих систем (например, Spring и Hibernate), а также инструментальные средства реализации с открытым исходным кодом (такие, как Ant и Eclipse), предоставляющие исчерпывающие, эффективные и изящные решения, которые можно рассматривать как дополнение или как полную альтернативу JEE, в зависимости от того, как вы применяете эти технологии для ваших конкретных потребностей. Кроме того, усовершенствованные процессы разработки программного обеспечения, такие как *экстремальное программирование* (Extreme Programming – XP) и *разработка методом гибкого моделирования* (Agile Model Driven Development – AMDD), способны существенно ускорить реализацию проекта.

Разработка программного обеспечения – это вопрос персонала, процесса и технологий (причем, вероятно, в таком порядке очередности). Люди являются и разработчиками, и заказчиками создаваемого программного обеспечения. В этой книге я опишу остальные две составляющие: технологии и процесс. Вы узнаете, как, используя инструментальные средства и технологии, быстро разработать приложения Java, от уровня клиента до уровня данных. Кроме того, вы оцените большинство преимуществ, следующих из использования этих инструментальных средств и технологий, например, простоты и скорости разработки.

Прежде чем мы начнем, если вы не читали предисловие, я рекомендовал бы, по крайней мере, просмотреть его, поскольку там представлены основные задачи и способ организации этой книги, а также указаны некоторые причины, по которым я написал эту книгу.

# Что рассматривается в этой главе

Эта глава содержит краткий обзор ключевых технологий и процесса разработки программного обеспечения, который мы используем в этой книге. В данной главе вы получите представление о следующем.

- Технологии времени выполнения и инструментальные средства разработки, используемые в этой книге для создания примера приложения.
- Процесс разработки программного обеспечения, используемый при создании примера приложения.
- Как организована эта книга.

# Технологии, используемые в этой книге

Эта книга объединяет различные технологии с открытым исходным кодом, представленные в табл. 1.1. Они были выбраны для реализации исчерпывающего решения при создании корпоративных приложений на базе Java. Я также предоставил альтернативные технологии с открытым исходным кодом и, в некоторых случаях, коммерческие технологии, если вы не хотите реализовать сквозную систему, используя технологии, описанные в этой книге. Как я упоминал в предисловии, эта книга организована так, чтобы вы могли либо подряд читать ее, либо сразу переходить к определенным главам, пропуская некоторые главы, если какая-либо из рассматриваемых здесь технологий вами не используется (например, Hibernate). Хотя данная книга посвящена технологиям с открытым исходным кодом, это вовсе не означает, что я их фанатик. На самом деле я интенсивно работаю и с коммерческими продуктами, такими как сервер WebLogic от BEA, сервер баз данных Oracle и т.д. Однако и эти технологии вполне надежны. Их вполне достаточно для развертывания готового корпоративного приложения Java, причем они не будут вам ничего стоить!

**Таблица 1.1. Технологии, рассматриваемые в этой книге**

| Выбранная технология | Категория | Бесплатная альтернатива | Коммерческая альтернатива |
|---|---|---|---|
| Spring Framework (`springframework.org`) | Контейнер Inversion of Control (IoC), Web Framework | HiveMind и Pico для контейнера IoC; Struts, JavaServer Faces, Tapestry и другие для Web Framework | Применимых нет |
| Hibernate (`hibernate.org`) | Persistence Framework | EJB, JDO, iBatis | TopLink для Oracle |
| SDK Eclipse (`eclipse.org`) | IDE | NetBeans, jEdit и некоторые другие | IntelliJ то JetBrain, WebSphere Studio Application Developer от IBM |
| Ant (`ant.apache.org`) | Управление конфигурацией | make, gnumake, nmake, jam, cruise control, maven | Microsoft nmake, MKS make |
| Junit (`junit.org`) | Проверка | TestNG, Fit | Mercury LoadRunner |
| HSQLDB (`hsqldb.org`) | База данных Java 100% | MySQL, PostgreSQL, One$DB | Oracle, Microsoft, Sybase и др. |
| Apache Tomcat (`tomcat.apache.org`) | Контейнер HTTP Server/Servlet | Jetty и некоторые другие | BEA WebLogic, IBM Websphere, Caucho Resin и др. |
| Mozilla Firefox (`mozilla.com`) | Web Browser | Microsoft Internet Explorer, Opera | Применимых нет |

*Окончание табл. 1.1*

| Выбранная технология | Категория | Бесплатная альтернатива | Коммерческая альтернатива |
|---|---|---|---|
| OpenOffice.org (`openoffice.org`) | Office Suite (используется в этой книге для схем в свободной форме) | Koffice (для Linux KDE) | Microsoft Office, StarOffice, EasyOffice |

Как я упомянул в предисловии, основное внимание этой книги уделено разработке, а не инфраструктуре. Однако, как вы, несомненно, знаете, Java — это независимость не только от операционной системы, но и от производителя программного продукта; например, при развертывании собственного примера приложения вы вполне можете заменить систему Tomcat на что-нибудь другое типа IBM WebSphere. Хотя на деле это может оказаться не так просто, как на словах, но такая замена вполне возможна.

> **Отдельное, самодостаточное приложение в едином файле WAR**
> Я бы хотел подчеркнуть некоторые интересные моменты и возможности технологий, обсуждаемых в этой книге. Представьте себе возможность получить готовое корпоративное приложение со встроенной базой данных (в данном случае — HSQLDB), встроенным планированием задач (благодаря Spring Framework), управлением транзакциями на корпоративном уровнем и некоторыми другими услугами внутри одного единственного автономного, легко устанавливаемого файла Web-архива (`.war`)! (Web ARchive — WAR.)

Следующие два раздела содержат краткое описание технологий и инструментов, их задач и причин, по которым я их выбрал.

## Технологии времени выполнения

Этот раздел содержит краткое описание технологий времени выполнения, которые используются для запуска приложения после его развертывания. Они отличаются от инструментальных средств разработки, используемых для создания приложения. На рис. 1.1 содержится визуальное представление взаимодействия этих технологий, обеспечивающего полное решение времени выполнения.

### Комплект разработки Java Platform, Standard Edition (Java SE)

Мы используем версию JDK (например, 1.5), доступную на Web-сайте `java.sun.com`. Обратите внимание на то, что описанные в этой книге технологии (например, Hibernate и Spring) прекрасно будут работать и с JDK версии 1.4.

### Среда Spring Framework (`springframework.org`)

Среда Spring Framework, содержащая большое количество классов и пакетов, была разработана как модульная среда, которая может быть поэтапно или частично введена в проект, т.е. использованы будут только необходимые средства (например, среда выполнения Web). Среда Spring дополняет систему Java/JEE, предоставляя контейнер *инверсии управления* (Inversion of Control — IoC), обсуждаемый далее в книге, среду выполнения Web, слой абстракции управления транзакциями, вспомогательные классы JDBC, API планирования задач, возможности электронной почты и многое другое.

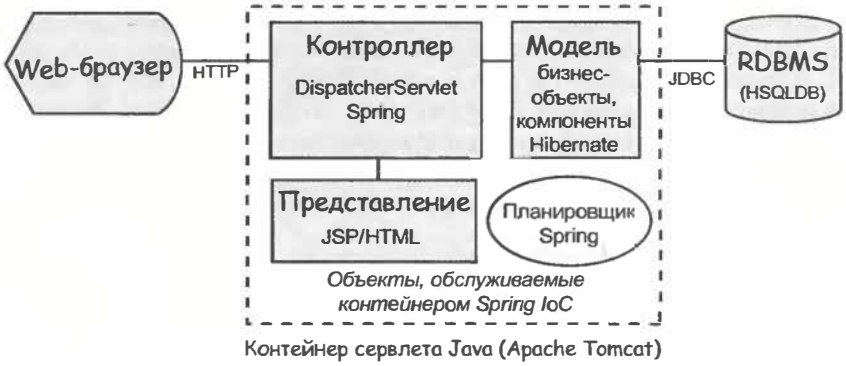



**Рис. 1.1.** Взаимодействие технологий времени выполнения, рассматриваемых в этой книге

Среда Spring появилась в 2002 году и с тех пор завоевала популярность и поддержку в обществе, включая таких коммерческих производителей, как BEA Systems.

На момент написания этой книги среда Spring была лидером в области контейнеров IoC; однако и ее среда выполнения Web также удивительно популярна. Spring я выбрал в качестве среды выполнения Web потому, что мне нужны также и другие ее возможности, такие как IoC, управление транзакциями, электронная почта, планирование и т.д.

Среда выполнения Web Spring MVC не имеет аналогов в области отказоустойчивости и гибкости. Я был приятно удивлен большим количеством статей по этой среде, а несколько посвященных ей книг (на момент написания данной книги) и большое количество запросов на `google.com` (для слов *spring* и *mvc*) свидетельствуют, что система Spring MVC является второй по использованию после Struts (`http://www.bejug.org/confluenceBeJUG/display/BeJUG/2005/07/05/Polls+results`).

### Среда Hibernate (`hibernate.org`)

Среда Hibernate — это *объектно-реляционная* (Object-to-Relational — OR) отказоустойчивая среда выполнения Java. Она вполне может подойти для разработчиков Java средней квалификации, а не только экспертов по технологиям OR. Сейчас Hibernate — возможно, наиболее широко используемая среда выполнения OR в мире разработчиков Java. Она также является хорошей альтернативой Entity Beans, что, вероятно, послужило одной из причин того, что EJB 3 имеет теперь много методов из Hibernate (JDO и Toplink). Исходя из этого, мое решение использовать Hibernate было даже проще, чем выбор среды выполнения Web.

### HSQLDB (`hsqldb.org`)

Это облегченная, но вполне полнофункциональная *система управления реляционными базами данных* (Relational Database Management System — RDBMS), на 100% написанная в Java. База HSQLDB удовлетворяет подмножеству стандартов ANSI-92 SQL и обладает драйвером JDBC для взаимодействия программ Java с базами данных. В течение последних нескольких лет популярность HSQLDB устойчиво возрастала.

Я решил использовать HSQLDB потому, что она проста в применении и установке, а также потому, что основное внимание этой книги уделено разработке, а не инфраструктуре. В недавнем проекте мы использовали базу данных Oracle; но на начальных этапах разработки я использовал HSQLDB, пока нашу базу данных Oracle устанавливали (очень медленно, благодаря корпоративной бюрократии). Если вы используете стопроцентный SQL ANSI, то теоретически, в процессе разработки, вы могли бы переключаться между локальной и корпоративной базами данных.

### Apache Tomcat (`tomcat.apache.org`)

Tomcat — возможно, наиболее популярный Java-ориентированный Web-сервер и контейнер сервлетов. Это относительно облегченный контейнер сервлетов, популярность которого за последние несколько лет существенно возросла. Я выбрал этот продукт потому, что с ним уже знакомо достаточно много разработчиков, так что решение это вполне очевидно. Подобно системе HSQLDB, которая может быть заменена на более надежную базу данных (типа MySQL или Oracle), сервер Tomcat также может быть заменен на более надежный Web-сервер или сервер приложений, такой как WebLogic от BEA.

## Инструменты разработки

Ниже описаны инструментальные средства разработки, используемые для создания нашего примера приложения.

### SDK Eclipse(`eclipse.org`)

Eclipse — один из наилучших инструментов Java, появившихся за последние годы. На мой взгляд, это продлило жизнь Java в качестве доминирующей технологии. Eclipse посвящена глава 8, "Феномен Eclipse!", которая содержит информацию о его базовой IDE и огромном количестве доступных дополнений для нее.

SDK Eclipse — это *интегрированная среда разработки* (Integrated Development Environment — IDE) с открытым исходным кодом, созданным IBM. Сама по себе Eclipse — это только платформа; однако возможность разрабатывать дополнения для нее делает Eclipse весьма мощным инструментом. Дело в том, что основные компании-производители перестроили или перепаковали свои программные продукты в качестве дополнений Eclipse. С учетом того, что Eclipse — реализация с открытым исходным кодом, наличие большого количества доступных дополнений и все возрастающая поддержка среди крупных производителей делают Eclipse, некоторым образом, полным победителем в области IDE Java.

Базовые инструментальные средства Eclipse Java включают средства форматирования исходного кода, его создания, отладки и интеграции с Ant. Но для Eclipse доступны буквально сотни бесплатных и коммерческих дополнений — от инструментальных средств создания схем UML до баз данных. Если необходимы некоторые функциональные возможности, то вы, с высокой долей вероятности, найдете дополнение для этого! Более подробная информация об Eclipse приведена в последующих главах, а сейчас можно посмотреть снимок экрана SDK Eclipse на Mac OS X (рис. 1.2), на Windows XP (рис. 1.3) и на Linux (рис. 1.4).

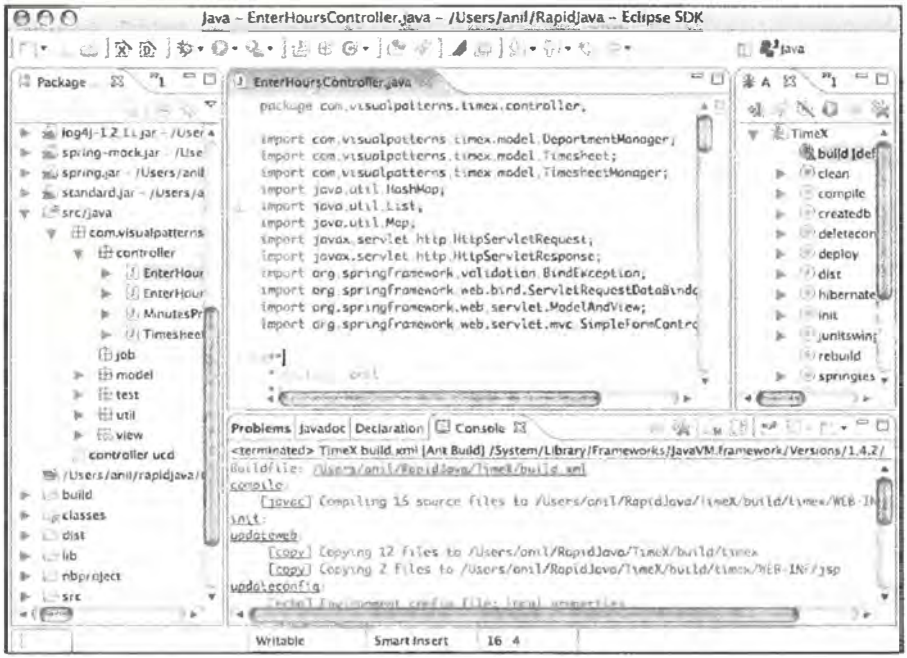



**Рис. 1.2.** SDK Eclipse 3.1 на Mac OS X

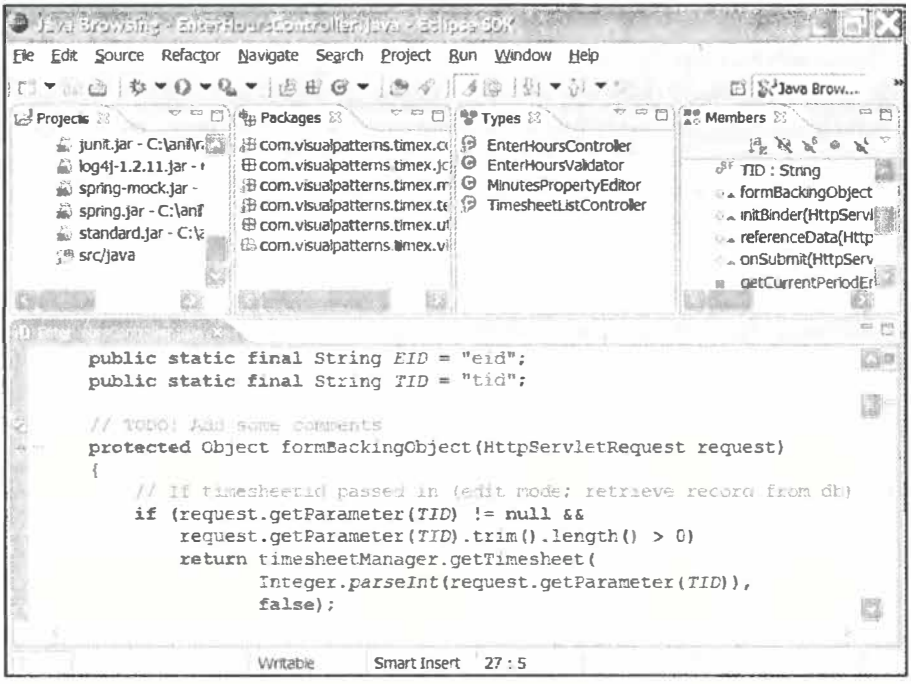



**Рис. 1.3.** SDK Eclipse 3.1 на Windows XP

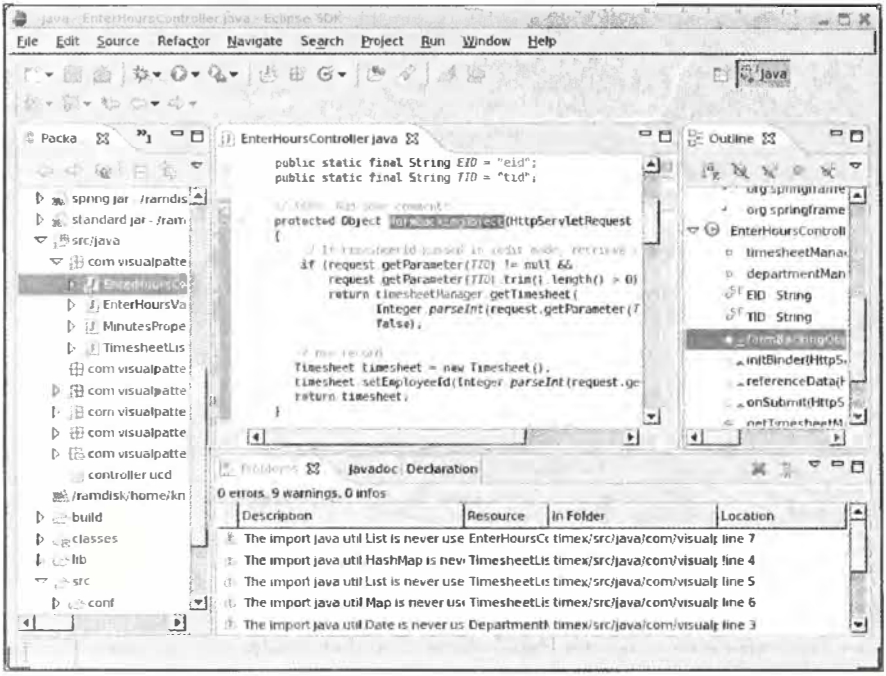



**Рис. 1.4.** SDK Eclipse 3.1 на Linux

### Ant (`ant.apache.org`)

Любой, кто работал с Java в последнее время, безусловно, слышал или работал с таким инструментом, как Ant. Ныне применение Ant — это общепринятый способ создания (и развертывания) программ Java. Хотя Ant во многом подобен утилитам разработки Unix, он предоставляет несколько дополнительных преимуществ.

Более подробная информация об Ant приведена в последующих главах, а сейчас рассмотрим следующий пример Ant в файле `build.xml`.

```
<?xml version="1.0"?>

<project name="HelloTest" default="printmessage">
    <target name="printmessage">
        <echo message="Hello world!"/>
    </target>
</project>
```

Более подробная информация об Ant приведена в главе 4, "Установка среды: JDK, Ant и JUnit", где вам будет продемонстрирована мощь инструмента Ant и объяснено, почему он фактически стал основным средством разработки в сообществе Java.

### JUnit (`junit.org`)

Фактически JUnit — это модульная среда проверки, используемая разработчиками Java ныне. Я никогда не был фанатиком создания и предварительной проверки модулей, однако недавно я пришел к мнению, что это весьма полезно. Далее в этой книге

я объясню, как проводить проверку готового изделия. На рис. 1.5 приведен снимок экрана GUI инструмента JUnit, чтобы дать вам общее представление о проверке кода вашего модуля. Однако степень интеграции JUnit с SDK Eclipse вы сможете оценить только в главе 8, "Феномен Eclipse!".

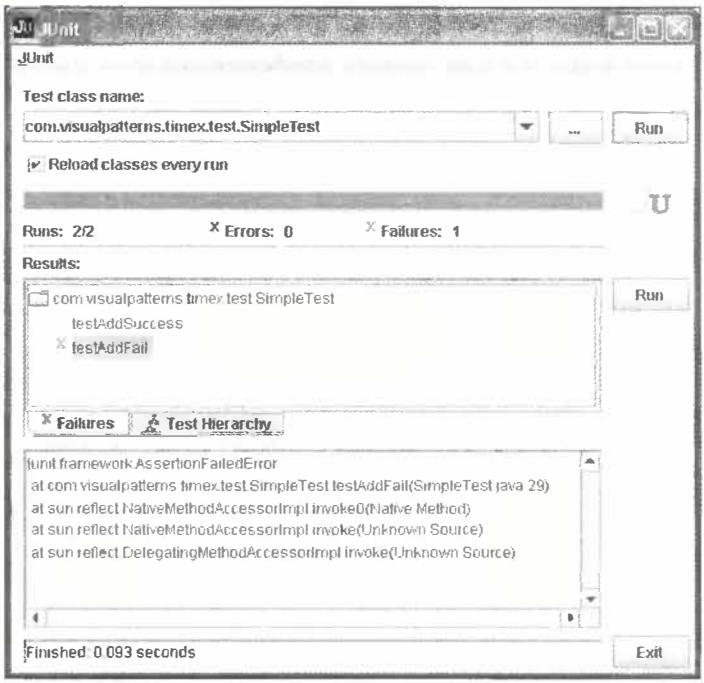



**Рис. 1.5.** Пускатель Swing JUnit на Windows XP

### Firefox (`mozilla.com`)

Вы могли бы задаться вопросом, почему Web-браузер здесь указан явно? Поверьте, я выбрал этот браузер вовсе не из религиозного фанатизма. Просто Firefox имеет множество достоинств, помогающих при разработке Web-приложений, и мы воспользуемся некоторыми из них в главе 9, "Регистрация, отладка, мониторинг и профилирование".

Особенно очаровательным в Firefox является большое количество полезных дополнений, доступных для него. На момент написания этой книги по адресу `https://addons.mozilla.org/website` было доступно 1 091 дополнение (там есть также инструментальные средства и расширения)!

### OpenOffice.org (`openoffice.org`)

Это комплект эффективных инструментальных средств с открытым исходным кодом, вполне способный конкурировать даже с Microsoft Office. Сначала я не стал включать его в табл. 1.1, сомневался в его необходимости для этой главы. Впервые я наткнулся на него пару лет назад, когда это был еще совсем сырой продукт и, на мой взгляд, никоим образом не был эквивалентен Microsoft Office практически ни в каком отношении. Фактически OpenOffice.org способен также читать и записывать файлы Microsoft Office.

Я выбрал OpenOffice.org потому, что купил новый портативный компьютер и хотел проверить новую (и, безусловно, бесплатную) версию OpenOffice.org перед тем, как выложить пару сотен долларов за Microsoft Office. Я был так поражен последней версией, что переделал рукопись этой книги с использованием комплекта офисных инструментов OpenOffice.org.

# Методы разработки программного обеспечения, используемые в этой книге

Я всегда полагал, что для успеха проекта необходимо позаботиться о его участниках, простоте (проекта, инструментальных средств, документации), здравом смысле, а также основных приемах разработки программного обеспечения. Кроме того, мне не нравится заново изобретать колесо, поэтому, найдя хорошее решение, я стараюсь использовать его снова. Одно из таких решений, которое я использовал в недавнем прошлом, — это комбинация полезных советов и приемов, рекомендуемых при разработке методом гибкого моделирования (`http://agilemodeling.com`) и экстремального программирования (`http://extremeprogramming.org`), или, для краткости, AMDD и XP соответственно.

Прежде чем перейти на AMDD и XP, я использовал для проектов *рациональный унифицированный процесс* (Rational Unified Process — RUP). Однако, на мой взгляд, RUP немного сложен при работе с артефактами. Мне нравится комбинация AMDD и XP потому, что оба метода удобны и дополняют друг друга; XP ориентирован на полный жизненный цикл, а AMDD — на моделирование (пользовательского интерфейса, например). Больше об этих двух методах вы узнаете в следующей главе, наряду с моим мнением о том, почему мне нравятся принципы и действия гибкого моделирования.

**Личное мнение: десятилетие Java и многое другое**

Вынужден признать, что этот раздел скорее обо мне, чем о Java, так что вы можете пропустить его, если мое личное мнение вам не интересно.

Когда я оглядываюсь на годы, потраченные на работу с Java, чувствую себя стариком. На самом деле я начал работать с Java еще в 1995 году, когда акроним JDK еще даже не установился. Однако истоки Java фактически относятся к 1990 году (чтобы узнать больше об истории Java, посетите Web-сайт `wikipedia.org/wiki/Java_program-ming_language`).

За эти десять с небольшим лет работы с Java я встречал некоторых из первых создателей Java, таких как Джеймс Гослинг (James Gosling), Артур Ван Хофф (Arthur Van Hoff), Джонатан Пейн (Jonathan Payne) и Сами Шайо (Sami Shaio) (имена некоторых из них все еще можно встретить в исходном коде JDK). Я также имел честь быть пятым человеком в компании WebLogic, Inc. (пару лет, прежде чем компания BEA приобрела ее), но я не захотел переметнуться (да, знаю, и все еще укоряю себя за эту упущенную возможность). Я также обучил более тысячи студентов Java и Web-технологиям на моих собственных курсах. Я опубликовал более 25 статей о Java и даже основал (и продал) две компании, специализирующиеся на Java-ориентированных решениях. В одной из этих компаний мой персонал (и я) разрабатывал корпоративные решения Java для многих компаний. Фактически, мы ознакомили с Java несколько больших компаний. (Я лично особенно горжусь тем, что предоставил решения Java компании Fortune 50!) Кроме того, я участвовал в пяти конференциях JavaOne и дважды был удостоен награды за программное обеспечение резервного копирования, написанное исключительно на Java. И наконец, я участвую в нескольких всемирно известных группах пользователей и конференциях.

Зачем я вам все это сообщил? Ну, во-первых, чтобы похвастаться. Во-вторых, чтобы подчеркнуть уникальность этой книги. И в-третьих, потому, что даже после десяти лет Java все

еще считается доминирующей технологией! В этой книге я познакомлю вас с новыми технологиями, предоставляющими Java совершенно новый путь выживания. Теперь я убежден, что Java сохранит передовые позиции, по крайней мере, еще несколько лет.
Короче говоря, если вы разработчик Java, вам для работы нужны изящные и надежные технологии! Я надеюсь доказать это вам в данной книге. Наслаждайтесь!

# Резюме

В этой главе мы кратко рассмотрели следующее.

- Технологии времени выполнения и инструменты разработки, используемые в этой книге для создания примера типового приложения.
- Процесс разработки программного обеспечения, используемый для создания примера типового приложения.
- Организацию этой книги.

Короче говоря, я предоставил вам краткий обзор инструментов, которые мы используем в этой книге для создания примера типового приложения, наряду с процессом разработки программного обеспечения, которому мы будем следовать. В следующих главах мы будем использовать эти технологии при разработке реального приложения — системы учета.

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, содержится на следующих Web-сайтах:

- Agile Modeling `http://www.agilemodeling.com`
- Ant `http://ant.apache.org/`
- Apache Tomcat `http://tomcat.apache.org`
- SDK Eclipse `http://eclipse.org/`
- Hibernate `http://hibernate.org`
- HSQLDB `http://hsqldb.org`
- Продукты Java с открытым исходным кодом `http://java-source.net/`
- JUnit `http://junit.org`
- OpenOffice.org `http://www.openoffice.org/`
- Spring Framework `http://springframework.org`
- Visual Patterns `http://visualpatterns.com`

# 2

# Простое приложение: сетевая система учета рабочего времени

Выпуск 1, Неделя 1, Итерация 0

**Радж:** В настоящее время мы находимся в фазе исследования проекта, который будет содержать модель домена, эскизы пользовательского интерфейса и т.д. Затем мы со Стивом сделаем набросок архитектуры, что позволит нам перейти к итерации 1. Первый выпуск, который мы фактически развернем на предприятии, будет включать итерации 1 и 2, а впоследствии мы будем разворачивать итерации каждые 2 недели. Обратите внимание, каждая итерация будет иметь вполне работоспособный код, так что мы сможем решить, следует ли ее устанавливать в фактически рабочей среде.

Одна из наших основных задач — не усложнять все чрезмерно и иметь только достаточный минимум документации, поэтому мы не будем тратить недели времени на полную разработку требований и архитектуры. Кроме того, основой документации системы послужит наш код, поскольку мы всегда сможем получить документацию из него.

В реальном мире новый проект программного обеспечения инициализируется обычно потому, что у клиента возникает некоторая потребность, проблема или необходимость оптимизации процесса. Эта потребность может быть как внутренней, так и внешней (например, необходимость взаимодействовать с внешней системой партнера или предоставить потребителям возможность доступа к своим ресурсам). После выявления проблемы или потребностей обычно осуществляется некоторая форма процедуры начала проекта, позволяющая лучше определить требования к нему.

Как я упомянул в предыдущей главе, в этой книге предпринимается попытка следовать реальной последовательности реализации проекта. Предположим, что у некоего вымышленного клиента возникла задача, решение которой требует создания типового приложения, например, под названием Time Expression (Выражение времени).

Для примера, используемого в этой книге, я рассмотрел несколько типов приложений; в результате я выбрал простую систему учета рабочего времени, поскольку, на мой взгляд, специфика этого приложения будет знакома большинству читателей. Например, если среди читателей есть сотрудники или консультанты с почасовой оплатой, то вы можете получить эту систему учета по сети (и даже проверить ее по сети).

Наше типовое приложение, Time Expression, будет иметь *пользовательский интерфейс* (User Interface — UI), также некоторые возможности по фоновой обработке. Пользовательский интерфейс будет Web-ориентированным, содержащим экраны, позволяющие вводить количество рабочих часов, утверждать табель, передавать отчет руководству и т.д. Фоновая обработка будет включать еженедельное (планируемое) пакетное задание (выполняемое автоматически) по отправке электронной почты с напоминанием.

# Что рассматривается в этой главе

В этой книге основное внимание уделено технологии и в меньшей мере процессу. Но эта глава содержит краткий обзор процесса *гибкой* (agile) разработки программного обеспечения, который вы можете легко применить в своем проекте. Обычно я создаю контрольный список, не более одной страницы, из примерно 10 пунктов, чтобы не забыть задачи, решаемые в процессе разработки. (*Примечание:* такой контрольный список я включил в приложение Д, "Типовой контрольный список процесса разработки".) Этот список гарантирует минимизацию и эффективность процесса разработки проекта и в то же время позволит сосредоточиться на удовлетворении требований заказчика.

В этой главе мы сделаем следующее.

- Установим, рассмотрев бизнес-требования, что будет делать наше типовое приложение.
- Установим простую методологию программного обеспечения на основании *экстремального программирования* (Extreme Programming — XP) и *разработки методом гибкого моделирования* (Agile Modeling Driven Development — AMDD).
- Разработаем некоторые из высокоуровневых артефактов, таких как доменная модель, прототипы UI, архитектура и т.д.
- Создадим упрощенный план выпуска на основании записей пользователя.

**Примечание**

Следует уяснить, что большинство представленных в этой главе артефактов (планы выпусков и итераций, например) является не более чем демонстрацией. Но эта глава очень важна, поскольку мы реализуем здесь часть функциональных возможностей (например, экраны Enter Hours (Ввод часов) и Timesheet List (Список учета рабочего времени). Подробные сведения, такие как дата и смета, пока еще можно игнорировать.

В этой главе подразумевается также, что вы имеете элементарное понятие о процессе разработки программного обеспечения и о связанных с ним концепциях (например, о прецедентах). Но если моих кратких объяснений различных концепций в этой главе недостаточно, для более подробных объяснений рекомендую посетить Web-сайт `www.agilemodeling.com`. На самом деле этот Web-сайт полон информацией, вполне уместной для данной главы. Кроме того, посетите Web-сайт `extremeprogramming.org`, он содержит подробную информацию о методологии XP.

# Бизнес-требования

Как я уже упомянул, нашему вымышленному клиенту требуется простая сетевая система записи и подтверждения учета рабочего времени. Давайте рассмотрим эти требования немного подробнее.

Прежде чем приступать к работе, всегда имеет смысл установить, *как* будет решена бизнес-проблема, для *кого* предназначено это решение и *почему* наше решение так важно для клиента. Кроме того, очень важно уяснить, *какое* решение необходимо клиенту и *каков* контекст проекта.

Предположим, что вымышленная организация хочет получить систему учета рабочего времени, чтобы почасово контролировать работу своего персонала. Для начала можно определить набор задач следующим образом.

**Формулировка задачи**

В настоящее время наши сотрудники передают свои еженедельные отчеты о количестве рабочих часов на бумаге, что подразумевает наличие ручной работы, а следовательно ошибок. Нам необходимо автоматизированное решение, которое позволит сотрудникам передавать отчеты о количестве рабочих часов в формате электронного табеля учета рабочего времени, а администрации утверждать их и оплачивать. Кроме того, мы хотели бы иметь автоматические уведомления об изменении состояния табеля учета, а также еженедельные напоминания о необходимости передать и утвердить табель сотрудника.

Исходя из нашей общей формулировки задачи, мы можем разбить ее на следующий набор возможностей или бизнес-требований; в мире *унифицированного языка моделирования* (Unified Modeling Language — UML) этот процесс можно рассматривать как часть *анализа прецедента* (use case analysis).

- Сотрудники с почасовой оплатой труда должны быть способны зарегистрироваться на Web-приложении (только раз или каждую неделю) и ввести свои часы работы на протяжении каждого дня данной недели. Наряду с количеством часов, сотрудник должен указать, какой из отделов должен их оплачивать.
- Сотрудники должны быть обязаны передавать свои табели учета рабочего времени каждую неделю.
- Менеджер должен быть уведомлен об успешной передаче табелей учета рабочего времени. Затем менеджер должен одобрить (утвердить) или не одобрить табель.

- После того как табель учета одобрен (или не одобрен), сотруднику возвращается уведомление об изменении состояния его табеля. Если табель утвержден, в бухгалтерию передается сообщение электронной почты о необходимости начисления зарплаты для данного сотрудника.
- Всем пользователям приложения Time Expression должен быть доступен один или несколько типов отчетов.
- Тем сотрудникам, которые не передали свой табель учета рабочего времени, по электронной почте передается еженедельное напоминание. Другое напоминание по электронной почте передается менеджерам, которые должны утверждать задержавшийся табель учета сотрудника.

Теперь, когда имеются некоторые из основных бизнес-требований, можем продолжать наш процесс разработки программного обеспечения.

# Методология разработки программного обеспечения

Каждый проект, маленький или большой, должен иметь некую базовую структуру или процесс разработки (методологию). Это может быть простой контрольный список на одной странице или более формализованный процесс. Полное отсутствие описания процесса – это обычно плохо, но слишком большое – еще хуже. Баланс зависит от потребностей клиента и размера проекта, однако на практике я пришел к выводу, что описание процесса должно быть "минимально достаточным" для документирования требований, а не иметь вид фолианта, способного потрясти клиента и разработчиков обилием документов и процедур. В этой главе я предоставлю базовый процесс разработки программного обеспечения на основании XP и AMDD.

## Краткий обзор XP и AMDD

XP и AMDD предоставляют некоторый набор фундаментальных правил для создания приложений программного обеспечения быстро и эффективно. Методы XP и AMDD дополняют друг друга, поскольку XP предоставляет дисциплинированный подход разработки полного жизненного цикла программного обеспечения, который гарантирует удовлетворение требований заказчика. AMDD, напротив, не только обеспечивает эффективные действия по моделированию и документации, но и предоставляет богатство дополнительных полезных возможностей, которыми можно воспользоваться в каждом рабочем проекте программного обеспечения.

**Личное мнение: почему гибкое моделирование и экстремальное программирование**
В последнее время я стал горячим поклонником значений, действий и принципов технологии Agile Modeling (AM; `agilemodeling.com`) (см. контрольные списки в приложении Д, "Типовой контрольный список процесса разработки"). Аналогично мой интерес к технологии Extreme Programming (XP; `extremeprogramming.org`) также вырос. Кроме того, в пользу этих методологий свидетельствует и тот простой факт, что я не слышал ничего плохого в их адрес от тех разработчиков, которые работали с ними. Я говорю о ряде весьма интеллектуальных людей, которые либо работают с этими двумя методами, либо осуществляют их поддержку. AM — это эффективное моделирование и документирование с использованием простых инструментальных средств. XP — это, напротив, обеспечение полного жизненного цикла разработки. Оба они очень хорошо совмещаются с моими представлениями о разработке программного обеспече-

ния, а мои представления основаны на более чем 20-летнем опыте работы в области IT примерно на дюжине очень больших и не очень больших предприятий.

До AM и XP я работал с технологией RUP (Rational Unified Process — *рациональный унифицированный процесс*), а перед этим с некоторыми специальными и самодельными методологиями. Даже при том, что некоторое время мне нравилось использовать RUP, со временем я нашел эту технологию немного тяжеловесной, когда дело доходило до документирования требований, разработки архитектуры и проектирования; по крайней мере, я заметил это у большинства организаций, использующих RUP. С другой стороны, методы AM и XP весьма гибки!

Чтобы приобрести навыки работы с XP, я потратил не много времени; в основном, это было связано с моим внутренним сопротивлением изменениям или неправильным представлениям. Однако когда все уже известно, чувствуешь себя уверенно! Причем настолько, что просто удивляешься, почему обучение длилось столь долго. Серьезно, эти две методологии становятся все более популярными потому, что разработчики чувствуют себя естественно, работая с ними.

Еще одной причиной популярности XP и AMDD является то, что их методы работают не линейно. Дело в том, что в реальном мире тоже не все работает линейно. Кроме того, этот подход лучше облегчает изменение, чем жесткие линейные методологии, в которых все требования следует предварительно тщательно фиксировать, и клиенты просто опускают руки, если им требуется внести слишком много изменений. Изменения неизбежны, так что иметь с ними дело придется. Поверите, у меня это занимало больше всего времени, поскольку я работал в жестком линейном режиме на протяжении многих лет, прежде чем обнаружил XP. Первоначально термин *гибкий* (Agile) относился к широкому диапазону методологий разработки программного обеспечения. В 2001 году термин *гибкий* был коллективно одобрен наравне с такими методами, как *экстремальное программирование* (Extreme Programming), SCRUM, DSDM, *адаптивная разработка программного обеспечения* (Adaptive Software Development), Crystal, Feature-Driven Development, *прагматичное программирование* (Pragmatic Programming) и другие (см. манифест и историю на `agilemanifesto.org`). Web-сайт `agilemanifesto.org` предлагает эти темы к рассмотрению и выступает в роли универсального Web-сайта для пополнения знаний об этих методах.

Технология AMDD более специфична для разработки. Я выбрал AMDD для этой книги потому, что, в дополнение к фактам, принципы AM совпадают с мой собственной точкой зрения, поскольку AMDD предлагает не только достаточно удобные артефакты моделирования (такие, например, как архитектурные схемы в свободной форме). Возможно, следующие слова, взятые с Web-сайта, лучше всего подведут итог моих взглядов на создание артефактов: "Ваша задача заключается в том, чтобы выработать общепринятые положения, а не писать подробную документацию". Можно ли выразиться яснее?

В XP вы найдете большинство концепций, используемых в этой главе и книге, включая такие концепции, как *пользовательские истории* (user stories), карточки CRC, *разработка с предварительной проверкой* (test-first design), планирование *выпусков* (release) и *итераций* (iteration), а также многое другое. Я уже указал причины, по которым они нравятся разработчикам. Однако клиенты также любят этот стиль работы, поскольку оба метода ориентированы на клиента и требуют активного участия коллектива разработчиков. Безусловно, это требует большего времени общения с клиентами, но они как раз любят постоянно контролировать процесс; в результате меньше вероятности того, что с проектом что-то пойдет не так, как надо, ведь между клиентом и разработчиками существует постоянная связь.

Постоянное общение с клиентом помогает также разработчикам не забывать о конечной цели и осознать предметную область лучше и быстрее. Следовательно, понимая проблемы или потребности клиента и оставаясь в курсе дела, легче следовать жизненному циклу разработки программного обеспечения. При наличии свободного доступа к клиенту решить эту задачу намного проще.

Позвольте мне подвести итог, почему так важно ознакомить вас с полным процессом использования XP. Ваша группа ежедневно создает и проверяет модули программного обеспечения; часто интегрирует их в приложение (возможно, даже постоянно, как только код модуля будет проверен); готовая рабочая версия приложения развертывается каждые две не-

дели (итерация); а клиент старается участвовать в проверках каждые две недели или даже использовать только что добавленные функциональные возможности (полный выпуск происходит каждые два месяца).
В целом, победа — это заслуга всех участников проекта.

Хотя я имею некоторые фундаментальные принципы, которые использую во всех проектах, обычно раздумываю, что применить к каждому конкретному проекту и как настроить процесс разработки программного обеспечения на основании потребностей текущего проекта и клиента. Например, в нашем типовом проекте приложения для этой книги, Time Expression, я использовал некоторые из методов, описанных далее и представленных на рис. 2.1 (более подробная информация по этой теме приведена по адресу `agilemodeling.com/essays/agileModelingXPLifecycle.htm`). Обратите внимание на то, что, хоть я и избрал путь XP и AMDD, существует несколько областей, где я уточняю эти методы (например, в начале проекта), чтобы приспособить их для нужд приложения Time Expression.

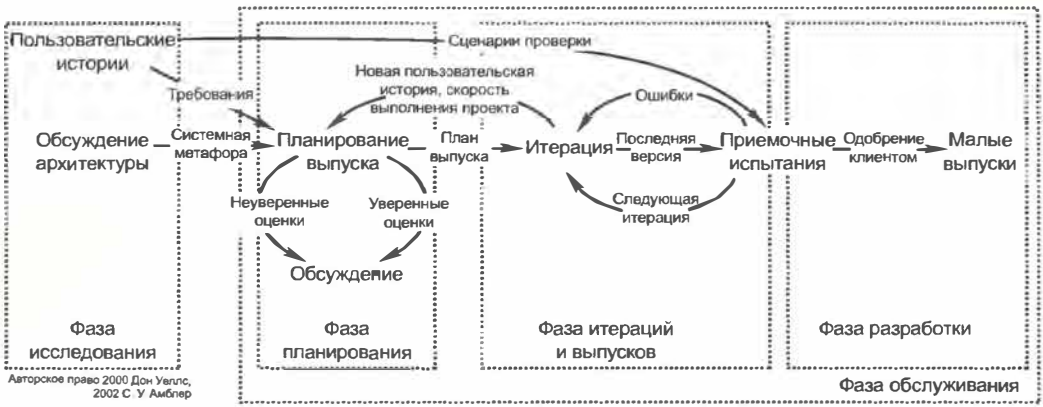



**Рис. 2.1.** Жизненный цикл проекта XP (источник: `agilemodeling.com/essays/agileModelingXPLifecycle.htm`); автор оригинала схемы Дон Уеллс (`http://extremeprogramming.org`)

## Фаза исследования

Обычно фаза исследования включает комбинацию исследовательских действий, которые помогают вам лучше понять потребности клиента и, следовательно, получить в результате приложение, которое будет им удовлетворять. Ниже приведены некоторые примеры действий, которые могли бы происходить на этой фазе проекта.

- Доменная модель. Помогает определять главные бизнес-концепции (объекты) и отношения между ними.
- Прототипы пользовательского интерфейса и последовательности экранов. Это первоначальные наброски экранов, позволяющие выяснить, как клиент представляет себе внешний вид приложения.
- Пользовательские истории. Несколько пользовательских историй дают начало проекту и составляют первый выпуск или версию приложения. Пользовательские истории — это записанные в кратком виде предложения, пожелания

и объяснения клиента по поводу того, что должно делать приложение. Обратите внимание на то, что количество первоначально собираемых пользовательских историй будет зависеть от проекта, но их должно быть достаточно для того, чтобы создать хороший и работоспособный выпуск.

- Определение контекста. Контекст проекта следует определить заранее, чтобы вы знали, для каких потребностей он создается и чем можно пренебречь. Это также проясняет ожидания клиента.
- Анализ. При анализе может использоваться комбинация *общей доски объявлений* (whiteboard), неформальные схемы архитектуры[1], глоссарий и т.д.

## Фаза планирования

Каждый из нас под планированием понимает что-то свое. Для меня планирование – это, по крайней мере, следующее.

- План выпуска. Это, по существу, план следующего выпуска (версии) системы, и он может быть легко создан при помощи или программы электронных таблиц, или программы подготовки текстов, или даже таблиц HTML. Сюда, сгруппированные в несколько итераций, заносятся все пользовательские истории, которые будут включены в следующий выпуск системы. Выпуски обычно имеют продолжительность где-то от одного до трех месяцев; как правило, оптимальным является срок в два месяца.
- План итераций. Создается план итерации перед каждой итерацией. Сюда включают пользовательские истории и замечания клиента, подлежащие реализации в следующей итерации. Итерации обычно имеют продолжительность – от одной до трех недель; как правило, оптимальным является срок в две недели.
- Определение стандартов (для кода, баз данных, процесса). Перед началом любой разработки имеет смысл стандартизировать соглашения по оформлению кода, об именовании баз данных и процессах (компиляции, интеграции, развертывания) и т.д.

**Активное участие заинтересованных лиц**

Согласно Web-сайту `agilemodeling.com`, "активное участие заинтересованных лиц — это залог распространения практики On-Site Customer (заказчик рядом) *экстремального программирования* (eXtreme Programming — XP), которая подразумевает возможность беспрепятственного доступа к людям, как правило заказчикам или их уполномоченным представителям, способность предоставлять информацию, имеющую отношение к создаваемой системе, а также принимать адекватные и своевременные решения в соответствии с текущими задачами и приоритетами".

С учетом сказанного выше, я рекомендовал бы всегда осуществлять планирование выпусков и итераций совместно с клиентами и разработчиками. Помните, успешные проекты — это обычно те проекты, в которых активно участвует клиент (отсюда и фраза "Активное участие заинтересованных лиц").

Более подробная информация об активном участии заинтересованных лиц приведена на Web-сайте AM: `agilemodeling.com/essays/activeStakeholderParticipation.htm`.

[1] Пример общей доски объявлений с неформальной схемой архитектуры приведен на рис. 2.12. – *Примеч. ред.*

## Фаза итераций и выпусков (инкрементное создание программного обеспечения)

*Итерационная разработка* (iterative development) — это термин, который вам, вероятно, знаком. Однако концепция итерационной разработки и состав каждой итерации зависят от конкретных обстоятельств и методологий.

Для меня итерационная разработка означает, что каждая итерация включает проект, создание кода, утверждение заказчиком и развертывание "работоспособного" кода. Этот код может быть развернут в рабочей системе или, если вы работаете в большой корпорации и развертывание в рабочей системе нереально, можно установить код на проверочной системе, где он будет одобрен клиентом, что позволит вам таким образом перейти к следующей итерации. В итоге, каждая итерация могла бы включать следующие действия.

- Задачи разработки, оценки разработчиков и план следующей итерации.
- Диалог из вопросов и ответов между разработчиками и клиентом.
- Проектирование. Карточки CRC, схемы UML и т.д.
- Создание кода, предварительная проверка, рефакторинг кода, базы данных, архитектуры, а также последующая оптимизация.
- *Входной контроль пользователя* (User acceptance testing — UAT).
- Развертывание итерации для работы (или UAT); этот этап называют также *малым выпуском* (small release).

Выполняя итерации таким образом, вы можете последовательно создавать последующие выпуски приложения. Например, мы могли бы выделить для данного проекта три месяца и разделить их на двухнедельные итерации, получив порядка шести итераций.

Результатом каждой итерации должен быть поставляемый проект; другими словами, малые выпуски должны содержать работоспособный код, даже если это только часть всей системы.

## Контекст проекта

Контекст проекта может быть определен в многих формах. Иногда это просто абзац на две строки, написанный клиентом, а иногда — структурные схемы использования. Многие организации идут дальше и заключают с группой разработчиков *договор о предоставлении услуг* (Service Level Agreement — SLA). Кроме того, чтобы определить контекст, можно провести обсуждения функциональных и нефункциональных требований.

В прошлом, для описания того, что входит в контекст приложения и что в него не входит, я использовал таблицу. Подобная таблица контекста для приложения Time Expression приведена в табл. 2.1.

**Таблица 2.1. Пример таблицы контекста**

| Контекст | Функциональные возможности |
|---|---|
| Включено | Приложение Time Expression позволяет вводить, утверждать и оплачивать рабочее время сотрудников |

*Окончание табл. 2.1*

| Контекст | Функциональные возможности |
|---|---|
| Исключено | Приложение Time Expression не будет рассчитывать удержание из зарплаты, например федеральные и местные налоги, а также медицинские издержки |
| Исключено | Приложение Time Expression не будет учитывать отпуск и отсутствие по болезни |

## Обслуживание

Здесь же следует обсудить обслуживание приложения. Эта фаза могла бы включать обучение пользователей, а также, при необходимости, небольшие усовершенствования и исправление ошибок (документируется в форме пользовательских историй). Возможно, клиент захочет получить другой главный выпуск, тогда придется снова вернуться к фазе исследования, описанной ранее в этой главе.

# Применение XP и AMDD к нашему примеру приложения

Теперь, когда имеется общее представление о процессе разработки программного обеспечения, применим его к нашему типовому приложению.

## Доменная модель

Приступим к доменной модели, которая, по существу, содержит все объекты данных и их отношения, но никаких атрибутов. Это поможет определить некоторые исходные элементы домена и их отношения. Как правило, *доменная модель* (domain model) — это схема, полученная в результате консультаций со специалистами в проблемной области и людьми, обладающими бизнес-знаниями, например квалифицированными пользователями и аналитиками.

На рис. 2.2 представлена доменная модель приложения Time Expression. Как можно заметить, наша доменная модель довольно проста, поскольку содержит минимальное количество объектов, достаточное на настоящий момент для того, чтобы перейти к проектированию. Повторюсь, сейчас нам нужна очень простая доменная модель, которой хватит для начала.

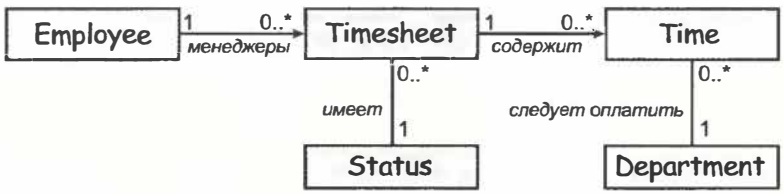



**Рис. 2.2.** Доменная модель приложения Time Expression

## Прототип пользовательского интерфейса

Теперь, когда есть довольно хорошее представление о возможностях приложения Time Expression, необходимых нашему клиенту, мы можем быстро набросать некоторые из прототипов экранов, которые позволят клиенту проверить работоспособность Web-приложения.

При первоначальной разработке прототипов экранов закладывается внешний вид пользовательского интерфейса, который позволяет людям комфортно работать с создаваемым приложением. Это также хороший способ избежать большинства косметических изменений внешнего вида (шрифтов, цветов), а также возможность продумать согласованный внешний вид, который можно реализовать, используя *каскадные таблицы стилей* (Cascading Style Sheets – CSS). Кроме того, прототипы можно использовать для моделирования бизнес-процессов и впоследствии применять их для реализации пользовательских историй (описанных далее в этой главе).

На рис. 2.3–2.10 демонстрируется, как будут выглядеть различные экраны приложения Time Expression. Мы начнем разрабатывать это приложение в следующей главе; затем, в последующих главах, мы получим нашу устанавливаемую систему и начнем разрабатывать некоторый код!


**Рис. 2.3.** Экран Sign-In

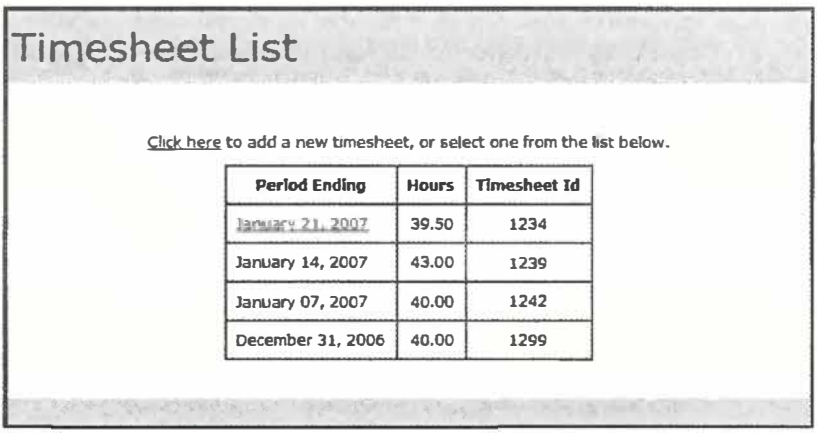



**Рис. 2.4.** Экран Timesheet List

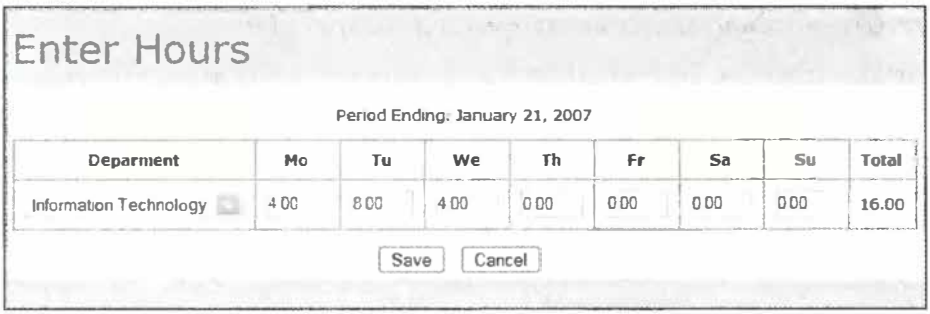



**Рис. 2.5.** Экран Enter Hours

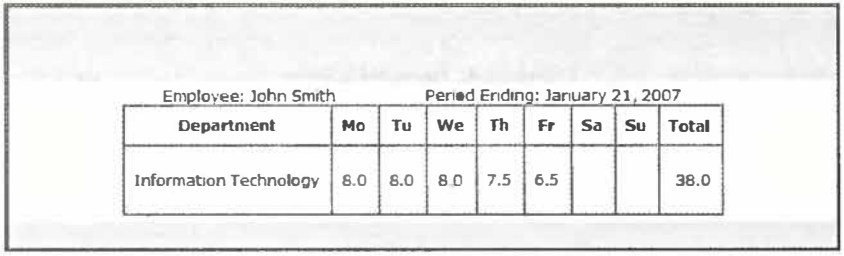



**Рис. 2.6.** Экран Print Hours

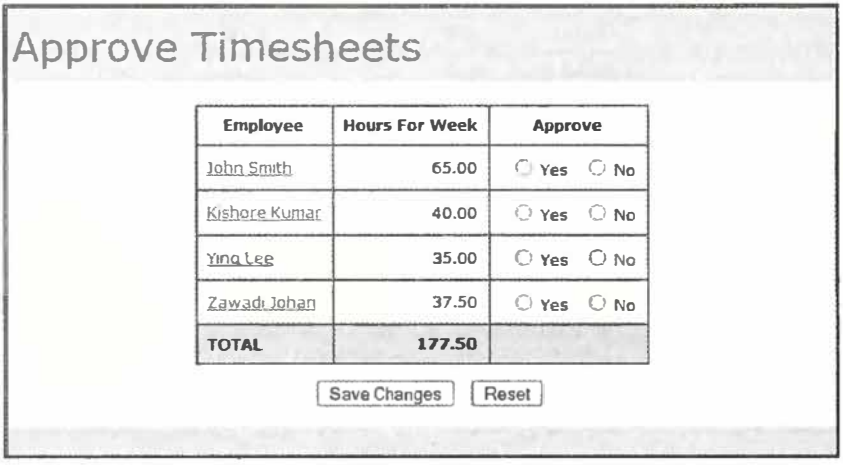



**Рис. 2.7.** Экран Approve Timesheets

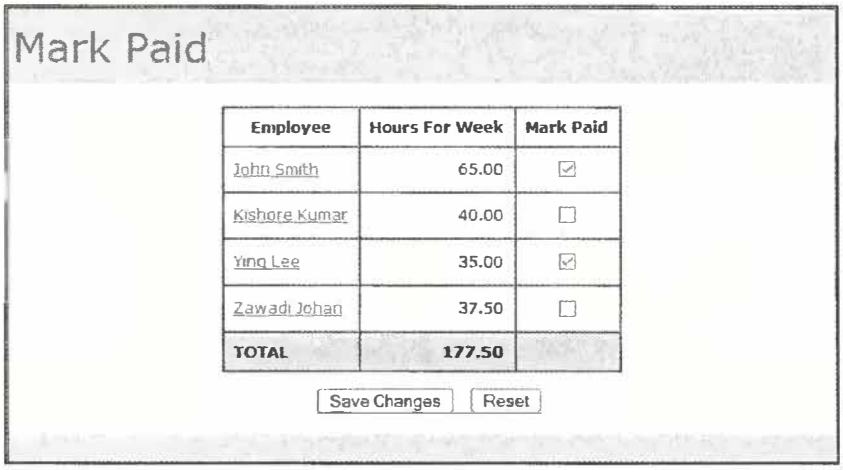



**Рис. 2.8.** Экран Mark Paid


**Рис. 2.9.** Экран Report: Staff Hours

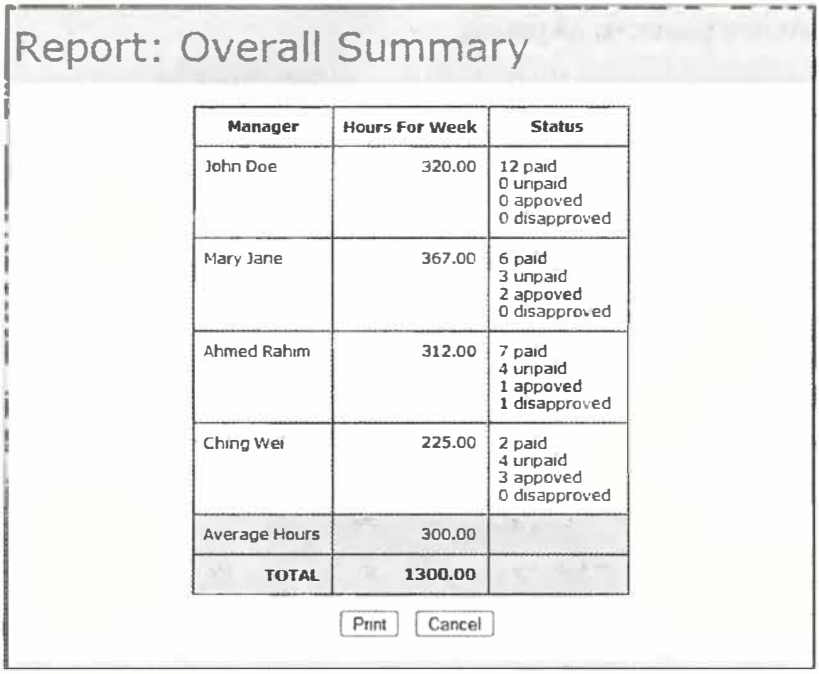



**Рис. 2.10.** Экран Report: Overall Summary

Теперь давайте сохраним эти экраны как простые файлы HTML (а не JSP), чтобы их можно было просматривать в браузере локально, без необходимости запускать каждый раз Web-сервер Java или редактор JSP/HTML. При работе с клиентом имеет смысл максимально сохранять наглядность технологий, пока вы не готовы к разработке, что позволит обеспечить простоту и избежать любых "технические трудностей", а следовательно, сосредоточиться на бизнес-требованиях.

**Учет ожиданий клиента при макетировании пользовательского интерфейса**

Хотя прототип UI обеспечивает существенные преимущества, на этом этапе работы очень важно также учитывать ожидания клиента. Например, я сталкивался с двумя проблемами. Во-первых, когда клиенты видят прототипы экранов, они обычно полагают, что приложение уже разработано и почти готово к развертыванию. Однако, как известно разработчикам, внешний вид экранов — это только начало разработки приложения. Во-вторых, если клиент слишком придирчив к шрифтам, размещению кнопок и другим эстетическим элементам, макетирование UI может выйти из-под контроля. Однако клиент всегда прав, потому что он обычно оплачивает приложение. Поэтому вы должны попытаться найти баланс между здравым смыслом и ожиданиями клиента.

Безусловно, весьма эффективным способом решения первой проблемы является рисование эскизов UI на бумаге, вместо макетирования. Это упрощает дело, поскольку эскизы не создают иллюзий у заказчика, а кроме того, бумагу можно приколоть на общую доску объявлений.

## Последовательность экранов

*Последовательность экранов* (storyboard), называемая также блок-схемой UI или картой Web-сайта, является отображением последовательности перемещения между различными экранами. На рис. 2.11 представлена последовательность экранов для нашего типового приложения. Как вы можете предположить исходя из нашей последовательности экранов, после того как пользователь зарегистрируется в системе, он попадает на начальный экран, предназначенный для ввода роли пользователя. Пользователи, соответствующие роли сотрудника и менеджера, имеют дополнительные функциональные возможности, обратиться к которым можно из начальных экранов.


**Рис. 2.11.** Последовательность экранов (известная также как блок-схема UI)

## Пользовательские истории

С учетом бизнес-требований и прототипов UI, определенных для приложения Time Expression ранее в этой главе, можем теперь зафиксировать набор пользовательских историй. Как объяснялось ранее в этой книге, я решил использовать XP/AMDD; следовательно, мы будем использовать термин *пользовательская история* (user story), а не *прецедент* (use case). Хотя они преследуют подобные цели, пользовательские истории обычно короче прецедентов — от одного до трех предложений каждая. Остальные детали могут быть обсуждены между разработчиком и клиентом, когда разработчик начнет работать над данной пользовательской историей в запланированной итерации; еще раз напомню об активном участии заинтересованных лиц.

Существует множество форматов прецедентов. Я знаю три: формальный, краткий и случайный. Случайный формат, вероятно, ближе всего к пользовательской истории, поскольку он короток и неофициален. Формальные прецеденты могут занимать одну или две страницы, где для каждого прецедента указаны такие разделы, как требования, предварительные условия, последующие условия, успешный/простой путь, ошибочный/альтернативный путь и др. Причина, по которой я описал здесь прецеденты и их форматы, заключается в том, что некоторые организации сначала определяют идеальные прецеденты (или даже бизнес-требования с заявлением "будет"), а затем разделяют каждый прецедент на одну или несколько (как правило, несколько) пользовательских историй, используемых как руководство для одно- или трехдневных задач разработки.

В этой книге мы используем пользовательские истории, причем определим не только требования, но и названия наших классов Java (используя имя истории/дескриптор),

а также создадим план приемочных испытаний. В табл. 2.2 представлены пользовательские истории с приоритетами и начальными оценками. Обратите внимание на то, что пользовательские истории, которые могли бы встретиться в реальном мире, могли быть немного более подробными, чем те, которые я привел в табл. 2.2; однако, с учетом простоты приложения Time Expression, эта работа не затруднит нас.

Есть очень хорошая книга о пользовательских историях *User Stories Applied: For Agile Software Development*, автор Майк Кон (Mike Cohn) (серия *Signature* издательство Addison-Wesley, 2004).

**Таблица 2.2. Пользовательские истории, приоритеты и оценки нашего приложения**

| № | Имя истории (дескриптор) | Описание истории | Приоритет | Баллы (оценка) |
|---|---|---|---|---|
| 1 | Enter Hours (Ввод часов) | Пользователь может вводить рабочие часы и сохранять эти данные | 1 | 2 |
| 2 | Timesheet List (Список табеля) | Сотрудник может просматривать список введенных ранее табелей учета рабочего времени и щелкать на тех, которые можно модифицировать | 1 | 1 |
| 3 | Sign In (Регистрация) | Пользователь может зарегистрироваться в системе, используя допустимый идентификатор сотрудника и пароль | 2 | 1 |
| 4 | Sign Out (Отмена регистрации) | Пользователь может выйти из системы, чтобы закончить текущий сеанс | 2 | 1 |
| 5 | Reminder Email: Employee (Напоминание по электронной почте: Сотрудник) | Тем сотрудникам, которые не передали свой табель, каждую пятницу в 2 часа пополудни по электронной почте передается напоминание | 2 | 1 |
| 6 | Print Timesheet (Печать табеля) | Сотрудник может отпечатать табель, используя форматирование в браузере, и автоматически просмотреть его в диалоговом окне печати | 3 | 1 |
| 7 | Report: My Hours (Отчет: Мои часы) | Сотрудник может затребовать отчет My Hours (Мои часы), чтобы просмотреть или отпечатать еженедельный отчет о своем рабочем времени | 3 | 1 |
| 8 | Submit Timesheet (Передача табеля) | После ввода количества часов пользователь может передать табель по электронной почте менеджеру | 3 | 1 |
| 9 | Report: Staff Hours (Отчет: Часы персонала) | Менеджер может затребовать отчет Staff Report (Часы персонала), чтобы просмотреть отчеты сотрудников за данную неделю | 4 | 1 |

*Окончание табл. 2.2*

| № | Имя истории (дескриптор) | Описание истории | Приоритет | Баллы (оценка) |
|---|---|---|---|---|
| 10 | Report: Overall Summary (Отчет: Общий результат) | Исполнитель может затребовать отчет Overall Summary (Общий результат), чтобы просмотреть отчеты всех менеджеров компании за данную неделю | 4 | 2 |
| 11 | Timesheet Approval (Утверждение табеля) | Менеджер может одобрить или не одобрить табель; по электронной почте уведомление отправляется сотруднику и в бухгалтерию | 5 | 1 |
| 12 | Timesheet Payment (Оплата табеля) | Бухгалтерия может оповестить сотрудника об оплате | 5 | 1 |
| 13 | Reminder Email: Manager (Напоминание по электронной почте: Менеджер) | Тем менеджерам, которые должны утверждать задержавшиеся табели, в 4 часа пополудни по электронной почте передается напоминание | 5 | 1 |
| | | | Всего баллов: | 15 |

Оценки, приведенные в табл. 2.2, — это начальные оценки или *прикидка* (sizing). В начале итерации, когда разработчик приступит к реализации пользовательской истории, он уточнит оценки, поскольку пользовательская история может быть разделена на задачи разработки, время выполнения которых точнее соответствует необходимому для всех пользовательских историй. Иногда мне встречались реальные проекты, где не было ни разделения пользовательских историй на задачи, ни итеративного планирования. Причина этого заключалась в том, что иногда изменяются приоритеты пользовательских историй или сами пользовательские истории, что сводило на нет предварительное планирование. Вместо этого они выбирали по две-три статьи в день для новой итерации и оценивали только их.

*Баллы* (points) (последний столбец табл. 2.2) — некоторая единица измерения, зависимая от конкретного проекта. Например, один балл может равняться одному обычному рабочему дню, одному идеальному рабочему дню, одной неделе или другому периоду, как договорятся клиент и разработчики. Количество баллов может измеряться и используется затем для оценки клиента.

*Идеальный день* (ideal day) разработки — это хороший способ оценки проекта, поскольку в реальности происходят как запланированные, так и не запланированные события (например, совещания, проблемы с компьютерами, отпуск по болезни и т.д.). Различие между обычным рабочим днем и идеальным называется *коэффициентом загрузки* (load factor).

Коэффициент загрузки, обычно 2–4, — значение, на которое вы умножаете свою начальную оценку на придуманный идеальный день. Например, я предпочитаю использовать коэффициент загрузки 3 для тех проектов, где бизнес-требования и технологии мне хорошо понятны. (*Примечание:* Если вы полагаете, что имеется некото-

рый риск, например придется познакомиться с новой технологией, то можно использовать большее число, 4 или 5.) Таким образом, если мне известно, что задача по разработке займет у меня приблизительно 8 часов непрерывной работы с полным сосредоточением, можем умножать это на 3 (т.е. 8 * 3). В результате получим 24 идеальных часа или 3 идеальных дня (с учетом 8-часового рабочего дня).

В этой книге мы разработаем экраны Enter Hours и Timesheet List (элементы 1 и 2 в табл. 2.2; оба с приоритетом 1). Так, например, на разработку этих двух экранов могло бы уйти 9 фактических дней (т.е. 3 идеальных дня с коэффициентом загрузки 3).

## План выпуска (и итераций)

После того как клиент (конечный пользователь и/или бизнес-аналитик) определил пользовательские истории, клиент и менеджер рабочего проекта (и/или разработчик) могут составить план следующего выпуска или версии приложения.

*План выпуска* (release plan) — это, по существу, список различных элементов проекта и даты выпуска каждого из них. В этой книге мы подразумеваем, что имеется только один выпуск — например, v1.0.

Выпуски обычно занимают немного времени, приблизительно от 1 до 3 месяцев. Каждый выпуск учитывает очередной набор пользовательских историй и ошибок, обнаруженных клиентом в прошлом выпуске. Затем каждый выпуск разделяется на итерации.

Продолжительность итераций составляет от 1 до 3 недель. Каждая итерация содержит список пользовательских историй, выбранных из набора пользовательских историй для данного выпуска, которые клиент считает нужным реализовать в данной итерации (наряду с устранением ошибок, обнаруженных ранее в ходе приемочных испытаний).

На основании пользовательских историй, которые мы определили ранее в этой главе, составим начальный план выпуска, представленный в табл. 2.3, чтобы инкрементно (поэтапно) создать выпуск v1.0 нашего приложения.

**Таблица 2.3. План выпуска**

| Итерация | Возможности | Дата выпуска |
|---|---|---|
| 0 | Установка среды (JDK, Ant, JUnit) и подключение к демонстрационной базе данных Hibernate | 23-Дек-06 |
| 1 | Малый выпуск. Все пользовательские истории с приоритетом 1 | 12-Янв-07 |
| 2 | Малый выпуск. Все пользовательские истории с приоритетом 2 | 26-Янв-07 |
| 3 | Малый выпуск. Все пользовательские истории с приоритетом 3 | 09-Фев-07 |
| 4 | Малый выпуск. Все пользовательские истории с приоритетом 4 | 23-Фев-07 |

Наш план выпуска не учитывает пользовательских историй с приоритетом 5, потому что для них предназначен второй выпуск нашего приложения. Безусловно, это полностью вымышленный пример, и большинство рассмотренных до сих пор элементов планов итераций и выпусков имеет исключительно демонстрационный характер. В реальном мире вам придется создавать существенно более подробные планы.

После того как план выпуска, содержащий различные итерации, определен для следующего выпуска программного обеспечения, разработчик может начинать работать над первой итерацией.

Перед каждой итерацией клиент и разработчики собираются на совещании, чтобы спланировать последующие итерации. Клиент выбирает пользовательские истории, которые будут разработаны на следующей итерации. Разработчики разбивают каждую пользовательскую историю на отдельные задачи разработки, необходимые для реализации пользовательской истории. Это делается не только для того, чтобы поставить конкретную задачу перед каждым разработчиком, но для того, чтобы более точно оценить общие усилия по разработке данной пользовательской истории и, в свою очередь, уточнить следующие итерации.

Если общая оценка, или баллы, (для реализации всех выбранных пользовательских историй для следующей итерации) превышает общее количество баллов, реализованных в предыдущей итерации (называемое в экстремальном программировании *скоростью проекта* (Project Velocity)), клиент должен выбрать, какие пользовательские истории следует реализовать (или какие дефекты устранить) при следующей итерации или следующем выпуске. Но если в следующей итерации осталось место, то при необходимости клиент может выбрать дополнительные пользовательские истории для реализации или дефекты для устранения.

В этой книге я пропустил процедуру составления типичного плана итераций, поскольку эта книга, в основном, о самой разработке, а не о процессе. Кроме того, форматы плана итерации (и выпуска) также могут варьироваться, поэтому я решил привести план выпуска (см. табл. 2.3) и сгруппировать наши пользовательские истории по приоритетам, вместо того чтобы разделять их на отдельные планы итераций.

## Глоссарий

Теперь настало время определить глоссарий для приложения Time Expression.

*Глоссарий* (glossary) – это, по существу, набор общих терминов, или словарь, который принимается всеми участниками проекта. Этот список может включать бизнес-термины (например, Timesheet (табель учета рабочего времени) и Approved (одобрение)) или технические термины (например, Entity (объект)), используемые при обсуждении логической модели данных. Очевидное преимущество глоссария – это согласованность в терминологии и определениях каждого термина, что позволяет избежать недоразумений. (В настоящее время имеется достаточно много не всем понятных терминов и сокращений, которые желательно согласовывать.)

- Accounting (бухгалтерия). Бухгалтерский отдел или служба.
- Approved (утвержден, одобрен). Состояние табеля учета рабочего времени, когда менеджер одобряет *переданный* (submitted) табель.
- Employee (сотрудник). Человек, работающий по почасовой системе оплаты и отчитывающийся перед *менеджером* (manager).
- Executive (управляющий). Высокопоставленный сотрудник компании, такой как CEO, CFO или COO.
- Hour (час). Полный учетный рабочий час, который может быть введен сотрудником в табель учета рабочего времени и подлежит оплате.
- Manager (менеджер). Непосредственный начальник *сотрудника* (employee).

- Paid (оплачен). Состояние табеля учета рабочего времени, когда бухгалтерский отдел выдал наличные.
- Period Ending Date (конечный срок). Это последний день каждой недели (в данном случае – воскресенье).
- Pending (задержка). Состояние табеля учета рабочего времени, когда пользователь не передал его.
- Submitted (переданный). Состояние табеля учета рабочего времени, когда пользователь уже передал его. После этого табель "блокируется" от дальнейших изменений сотрудником.
- Week (неделя). 40-часовая рабочая неделя с понедельника по пятницу.

## Архитектура на общей доске объявлений

На настоящий момент имеется достаточно информации, чтобы составить неофициальную схему архитектуры. На рис. 2.12 представлена высокоуровневая схема архитектуры приложения Time Expression (на столь необходимом инструменте, как общая доска объявлений). Составляя эту схему, клиент и разработчики могут договариваться об основных технологиях (например, Java, база данных, Web-сервер, сервер приложений), которые будут использованы для создания приложения Time Expression.

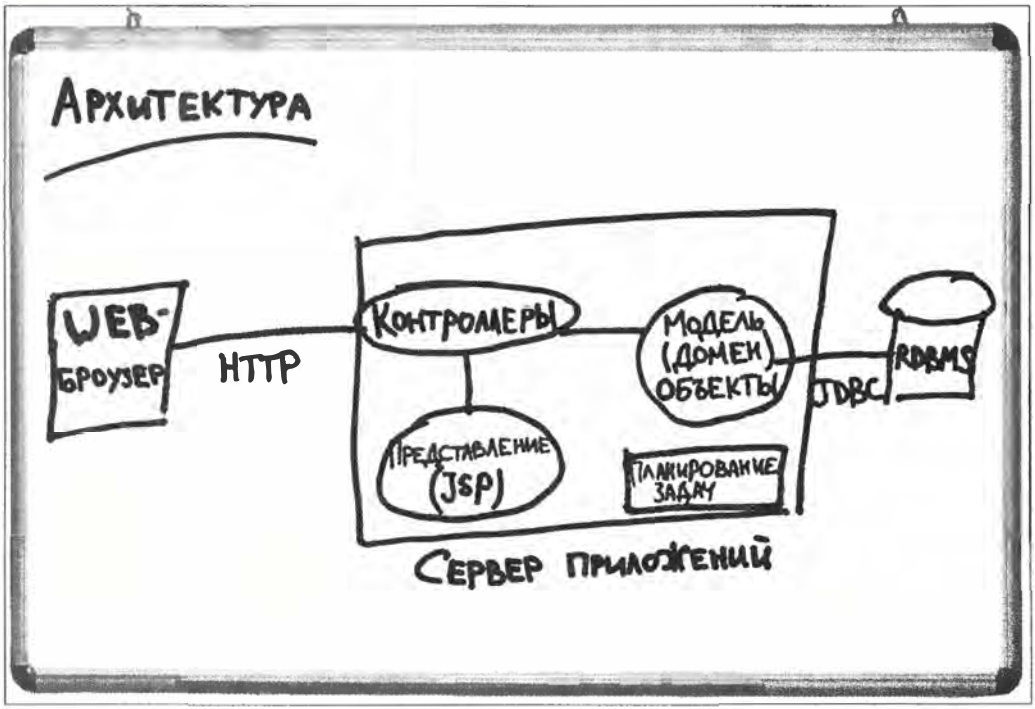



**Рис. 2.12.** Неофициальная высокоуровневая архитектура на общей доске объявлений

В следующей главе используется более подробная электронная версия этой схемы архитектуры, которая поможет перейти на следующий уровень проектирования приложения. Лично я предпочитаю преобразовывать схемы с общей доски объявлений или карточек CRC в электронный формат не только для повышения удобочитаемости; это позволяет также при помощи цифровой камеры и сканера разместить такие артефакты, как рисунки и карточки CRC.

# Замечание о программном обеспечении вики

Когда несколько лет назад я впервые услышал термин вики (wiki), или викивики (wikiwiki), то проигнорировал его, полагая, что это не стоит моего внимания (возможно, из-за странного названия). Однако популярность этого программного обеспечения распространялась лавинообразно, и в результате оно стало чрезвычайно эффективным инструментом, используемым для сотрудничества и управления знаниями в проекте.

Согласно Уорду Каннингему (Ward Cunningham), изобретателю программного обеспечения вики, "это самая простая сетевая база данных, из способных работать". Вики – программное обеспечение для всеобщего сотрудничества, которое, по существу, установлено на Web-сервере и позволяет редактировать Web-страницы, использующие простой текст (с различными параметрами форматирования). Хотя эта концепция может показаться слишком простой, на самом деле она оказалась очень мощной.

В проекте программного обеспечения, Web-сайт вики выступает в роли центрального хранилища для содержимого, которое участники группы программного обеспечения могут совместно добавлять или изменять. Это очень хорошо соответствует стилю работы в итерационном режиме. Например, документацию интерактивной справочной системы создаваемого программного обеспечения можно накапливать, используя вики итерационно и с приращением, в течение некоторого времени (скажем, на протяжении двухмесячного выпуска). Фактически большинство вставок с личным мнением, которые вы видите в этой книге, было накоплено при помощи моего вики-сайта в течение почти года. Это позволило быстро обращаться к доступным для редактирования Web-страницам, когда у меня появлялась подходящая идея.

Существуют буквально сотни процессоров вики (программное обеспечение), разработанных практически на всех известных языках программирования. Одни процессоры вики работают с текстовыми базами данных, другие – с реляционными. Некоторые процессоры вики обеспечивают надежную аутентификацию и авторизацию, в то время как другие предназначены для персонального пользования. Вашей самой большой проблемой будет, вероятно, выбор используемой системы.

# Резюме

В этой главе мы сделали следующее.

- С учетом задач нашего типового приложения, были рассмотрены некоторые из бизнес-требований.
- Определена простая методология разработки программного обеспечения на основании XP и AMDD.

- Разработано несколько высокоуровневых артефактов, такие как доменная модель, прототипы UI, общая архитектура и т.д.
- Создан простой план выпусков на основании наших пользовательских историй.

Теперь пришло время приступить к проектным работам.

# Рекомендуемые ресурсы

Дополнительную информацию о темах, обсуждавшихся в этой главе, содержат следующие Web-сайты:

- Agile Data `http://www.agiledata.org`
- Agile Manifesto `http://agilemanifesto.org`
- Agile Modeling `http://www.agilemodeling.com`
- Экстремальное программирование `http://extremeprogramming.org/`
- Статьи о PmWiki и процессоре вики на базе PHP `http://visualpatterns.com/resources.jsp`
- Сайт вики `http://wiki.org/`

Если XP не для вас, то вы могли бы предпочесть *гибкий унифицированный процесс* (Agile Unified Process), облегченную версию RUP, `http://www.ambysoft.com/unifiedprocess/agileUP.html`.

# II

# Создание простого приложения

# 3

# Архитектура и модель проекта на базе XP и AMDD

Выпуск 1, Неделя 2, Итерация 1

**Радж:** Стив, между нами, разработав прототипы UI, доменную модель, карточки CRC и т.д., я полагаю, мы сделали достаточно для итерации 1, как вы думаете?

**Стив:** Да, давайте не усложнять это приложение. Необходимо переходить к программированию, так что мы можем закончить наши начальные проектные работы. Между прочим, сначала необходимо определить способы быстрого программирования и стандарты интеграции, так что мы находимся на том же месте.

В этой главе мы наконец перейдем к технологическим вопросам, так что теперь начинается самое интересное.

В настоящей среде итерационной разработки всю архитектуру и проблемы проекта не обязательно следует решать в первую очередь. *Рефакторинг* (refactoring), улучшение кода без нарушения его функциональных возможностей, играет существенную роль в поэтапном проектировании, поскольку при реальном программировании постоянно приходится что-то менять. Кроме того, в то время как контекст проекта может быть определен заранее, требования пользователя могут продолжать развиваться от итерации к итерации и в конце существенно отличаться от начальных. Разрешение проблем с требованиями заключается в постоянном взаимодействии всех заинтересованных лиц проекта, а также способности обсудить возникающие вопросы.

Хотя некоторые работы могут быть выполнены заранее, например пользовательские истории, высокоуровневая архитектура, прототипы пользовательского интерфейса, доменная модель, стандарты и так далее, другие проблемы проекта могут проявиться на этапах соответствующих итераций. Кроме того, как будет продемонстрировано в главах 5, "Применение Hibernate для постоянных объектов", и 7, "Среда Spring Web MVC Framework", создание предварительных проверок может также помочь в проектировании классов, поэтому не следует рассчитывать на решение всех проблем сразу.

Хотя, безусловно, некоторые задачи проекта следует решить заранее, на так называемой нулевой итерации (возможно, когда вы попробуете продемонстрировать эффективность и работоспособность выбранных технологий, от пользовательского интерфейса до баз данных, например).

**Примечание**

**Из-за дополнительного времени, необходимого на проектирование и работы по установке среды (включая доменную модель, определение бизнес-объектов, соглашение об именовании Java, процесс, сценарии построения, сбор группы и так далее), на итерациях 1 и 2, возможно, удастся реализовать меньшее количество пользовательских историй.**

В этой главе я надеюсь снабдить вас сквозным подходом моделирования с применением основных принципов *разработки методом гибкого моделирования* (Agile Model Driven Development – AMDD; `agilemodel-ing.com`) и *экстремального программирования* (Extreme Programming – XP; `extremeprogramming.org`).

# Что рассматривается в этой главе

В этой главе мы закончим архитектуру и выработаем задачи для нашего типового приложения Time Expression.

- Разработаем схемы архитектуры в свободной форме
- Исследуем объекты с использованием карточек CRC
- Соберем артефакты, мне нравится обращаться к карте выполнения приложения
- Разработаем схемы класса и пакета для приложения Time Expression
- Установим структуру нашего рабочего каталога и рассмотрим некоторые типовые имена файлов (создадим их в последующих главах)
- Рассмотрим шаги, которые предстоит предпринять в следующих главах для сквозной разработки наших экранов

- Создадим список дополнительных концепций, которые мы должны будем рассмотреть по мере развития нашего типового приложения: обработка исключений, планирование работ, управление транзакциями, регистрация и т.д.

# Выбор подхода проектирования и артефактов

В предыдущей главе мы рассмотрели ориентированный на XP подход к определению бизнес-требований и работе с клиентом. В этой главе разработаем некоторую минимальную архитектуру и проект приложения Time Expression, создаваемого с использованием таких популярных технологий, как Hibernate, Spring Framework, SDK Eclipse, и многих других связанных с ними инструментов, таких как Ant, JUnit и т.д.

Если вы слышали миф о том, что программисты XP обходятся без проекта и документирования, надеюсь, что к концу данной главы это заблуждение будет полностью рассеяно, поскольку нет ничего более далекого от правды. Позвольте мне коротко объяснить, что я имею в виду. Обратите внимание на рис. 3.1, демонстрирующий некоторые из возможных артефактов, которые вы можете создать на уровне выпуска или итерации. Артефакты уровня выпуска создаются на протяжении выпуска, артефакты уровня итераций – на протяжении каждой итерации. Не все они обязательны для каждого проекта, поэтому мы можем использовать только необходимые. Однако здесь, в отличие от главы 2, "Простое приложение: сетевая система учета рабочего времени", я решил продемонстрировать как можно больше из них на практическом примере приложения Time Expression. В конце этой главы я представлю вам другую схему, которая свяжет вместе все артефакты (однако не стоит смотреть ее прямо сейчас, поскольку слишком подробная схема только запутает вас в данный момент).


**Рис. 3.1.** Выбор артефактов в стиле XP/AMDD для уровней выпусков и итераций

По крайней мере, то, что вы узнаете в этой главе, даст вам одну точку зрения. Этот процесс может либо подходить, либо не подходить для вас. Тем не менее некоторые элементы этих методологий популярны у разработчиков, и я видел много успешных проектов в результате их применения. Кроме того, создаваемые в этой главе артефакты необходимы для остальной части этой книги. Как можно заметить на рис. 3.1, в этой главе придется создать несколько артефактов, так что давайте приступим. Но прежде чем сделать то, я хочу ознакомить вас с обеими точками зрения практических пользователей XP.

Руководитель проекта, работающий в компании Fortune 50, недавно сказал мне следующее: "Когда мы начинаем итерацию, первый день обычно тратится на обзор статей и разделение их на задачи. Упражнение по разделению на задачи – это настоящий сеанс проектирования. По моим наблюдениям, порядка 20% времени разра-

ботчиков, в течение итерации, тратится на проект. Если сложить все время всех разработчиков по всем итерациям, получится весьма большое количество, которое верно разоблачит миф об отсутствии проектирования".

Чтобы ознакомить вас с другой точкой зрения на стиль работы XP, рассмотрим утверждение старшего архитектора известной компании решений IT, которая развернула более дюжины успешных проектов, используя методы XP и AMDD: "В проекте XP существует также другой уровень проектирования, ежедневный. Рефакторинг – это тоже проектирование. Хотя итеративное проектирование – это важный шаг, проектные работы после того, как код уже написан, позволяют сделать хороший проект отличным".

Различие в подходах XP вызвано тем, что архитектура и проектирование осуществляются на протяжении всего цикла выпуска приложения, а не только заранее. Другими словами, приложение продолжает развиваться от итерации к итерации. Преимущество этого подхода заключается в том, что проектирование на самом деле применимо ко всему процессу создания, а не только в течение трех-шести месяцев разработки, когда требования могут измениться (что, безусловно, вполне вероятно в нашем быстро изменяющемся современном мире).

# Схема архитектуры в свободной форме

На рис. 3.2 представлена высокоуровневая архитектура нашего типового приложения. Обратите внимание на то, что она была преобразована в электронную форму из версии для общей доски объявлений, приведенной в конце предыдущей главы. Преобразование в электронный формат – это персональное предпочтение; вы можете без проблем сделать цифровую фотографию изображения с общей доски объявлений, но лично я предпочитаю более читабельную форму схем.

Архитектура довольно проста. Здесь имеется наша стандартная трехуровневая Web-архитектура, включающая уровень клиента (Web-браузер), промежуточный уровень (сервер приложений) и уровень данных (база данных).

Кроме того, здесь используется стандартная схема проекта *модель–представление–контроллер* (Model–View–Controller – MVC), которую вы находите в большинстве современных, ориентированных на Java, Web-систем. *Контроллер* (controller) – входной

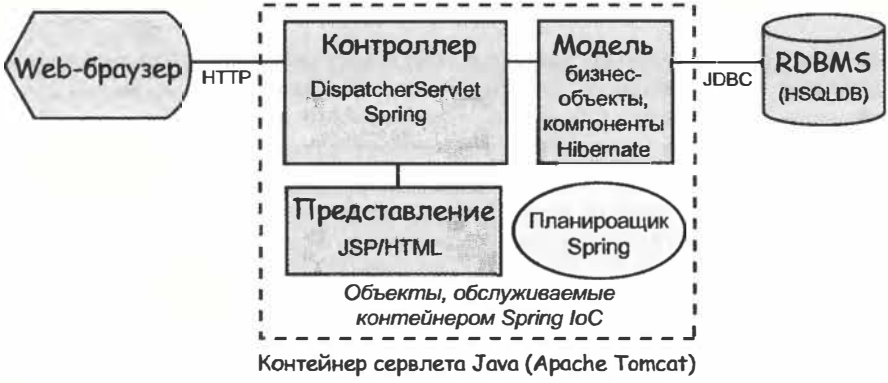



**Рис. 3.2.** Высокоуровневая архитектура приложения Time Expression

пункт запроса HTTP/Web; он контролирует модель и представление. *Модель* (model) имеет дело с данными, которые получены контроллером, и передает их уровню представления для визуализации соответствующим способом. В данном случае *представление* (view) будет написано с использованием JavaServer Pages (JSP).

Однако интересной нашу архитектуру делает вовсе не используемая схема MVC, а, скорее, то, что находится на промежуточном уровне, а именно Spring Framework и Hibernate – две технологии, которые будут рассматриваться подробно далее в книге. Технология Hibernate, как будет продемонстрировано, обеспечивает простоту и отказоустойчивость базы данных, поскольку можно ссылаться на таблицы и записи базы данных как на простые старые объекты Java (POJO). Технология Spring Framework (`springframework.org`) предоставляет множество преимуществ, особенно когда для среды выполнения Web вы используете MVC Spring, поскольку код получается проще (по сравнению с чем-нибудь вроде Struts). Еще одна замечательная возможность Spring Framework – это внутренняя поддержка планирования работ, вместо зависимости от внешней службы планирования, такой как CRON или Планировщик Windows (Windows Scheduler). Безусловно, основной возможностью, предоставляемой технологией Spring Framework, являются функциональные возможности *инверсии управления* (Inversion of Control – IoC), рассматриваемые в последующих главах.

# От пользовательских историй к разработке

*Пользовательские истории* (user stories) мы рассматривали в главе 2, "Простое приложение: сетевая система учета рабочего времени". Для краткости, мы не будем разрабатывать в этой книге каждую пользовательскую историю по отдельности. Однако пользовательские истории, которые я выбрал, дадут вам полное представление о создании экранов в виде форм (form screen) и не в виде форм (no-form screen). Кроме того, мы рассмотрим дополнительные возможности, такие как реализация защиты приложения с использованием перехватчиков, передача электронной почты и планирование работ, которые упоминаются в двух или более пользовательских историях, приведенных в главе 2.

В остальной части этой главы я буду приводить примеры на основании, по крайней мере, двух первых пользовательских историй, *Enter Hours* и *Timesheet List*, упомянутых в главе 2.

# Исследование классов с использованием карточек CRC

На рис. 3.3 демонстрируется доменная модель, которую мы установили в главе 2, "Простое приложение: сетевая система учета рабочего времени". Доменная модель позволяет нам исследовать домен или бизнес-объекты. Пользовательские истории позволят нам выявить Web-ориентированные классы контроллера пользовательского интерфейса. Итак, давайте рассмотрим и придумаем объекты для пользовательской истории Timesheet List, а затем увидим, как именно работают карточки CRC.

На рис. 3.4 представлен прототип UI Timesheet List (Список табеля учета рабочего времени) из главы 2. Как я упомянул, мы уже знаем, что наш пользовательский интерфейс будет ориентирован на Web и использует парадигму MVC. Поэтому давайте займемся определением наших исходных классов с точки зрения MVC.

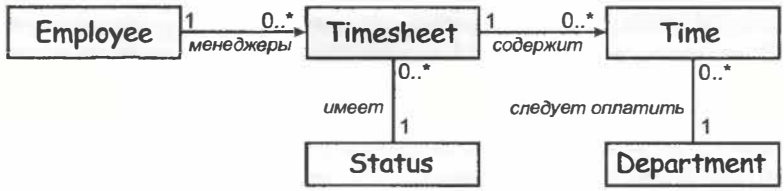



**Рис. 3.3.** Доменная модель приложения Time Expression

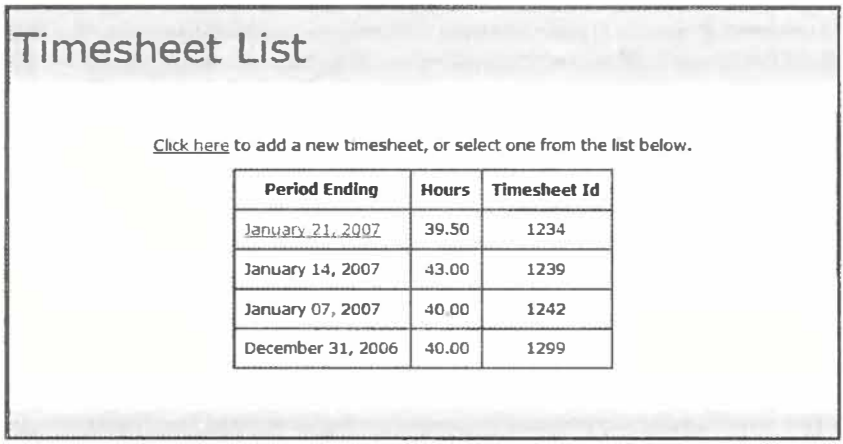



**Рис. 3.4.** Экран Timesheet List

Для части модели MVC, из нашей доменной модели, мы уже знаем некоторые имена объектов приложения Time Expression. Для части контроллера известна пользовательская история (в данном случае, *Timesheet List*) из главы 2, "Простое приложение: сетевая система учета рабочего времени". С учетом этого, мы можем теперь продолжать проектирование классов, используя карточки CRC.

В случае, если вас интересует, карточки CRC соотносятся с классами, их ответственностью и взаимодействием. Табл. 3.1 демонстрирует компоновку типичной карточки CRC, наряду с некоторыми объяснениями трех компонентов, которые вы видите там. Обратите внимание, хотя я представил электронную версию, карточки CRC вполне могут быть сделаны из картона 3×5 дюймов, которые впоследствии (при необходимости) преобразуются в схему класса.

Карточки CRC представляют собой неофициальную объектно-ориентированную методику выявления взаимодействий между классами. Мне нравятся карточки CRC потому, что они применяются при неофициальной встрече с разработчиками или пользователями для определения объектов (и компьютер для них ненужен). Кроме того, карточки CRC применяются для разработки формальной схемы класса, если это необходимо (нечто подобное мы проделаем далее в этой главе).

В табл. 3.2–3.4 демонстрируются некоторые типичные элементы карточки CRC для фактических классов, которые мы разработаем далее в этой книге, реализуя требования для экрана Timesheet List.

**Таблица 3.1. Типичное расположение карточки CRC**

| Имя класса (отсутствует) | |
|---|---|
| Обязанности (обязанности этого класса, такие как бизнес-методы, обработка исключений, методы защиты, атрибуты и переменные) | Сотрудники (другие классы, необходимые для обеспечения полного решения высокоуровневых требований) |

**Таблица 3.2. Пример карточки CRC для класса `Timesheet`**

| `Timesheet` | |
|---|---|
| Знает о конечном сроке, о времени, о коде отдела | |

**Таблица 3.3. Пример карточки CRC для класса `TimesheetManager`**

| `TimesheetManager` | |
|---|---|
| Выбирает табель(и) учета из базы данных. Сохраняет табель в базе данных | `Timesheet` |

**Таблица 3.4. Пример карточки CRC для класса `TimesheetListController`**

| `TimesheetListController` | |
|---|---|
| Контроллер (в MVC) для отображения списка табелей учета | `TimesheetManager` |

Мы только что рассмотрели некоторые основы применения карточек CRC. Теперь настало время перейти к следующему шагу.

# Карта потока приложения (самодельный артефакт)

В прошлых проектах я использовал таблицу, подобную табл. 3.5. Этот самодельный формат — не стандартный, просто я его придумал. Я называю это *картой потока приложения* (application flow map), поскольку она демонстрирует, как будет функционировать пользовательский интерфейс (или поток) от начала до конца. Эта методика удобна также при отображении пользовательских историй для *представления* ("V" в MVC), которые соотносятся с *контроллером* и, наконец, с *моделью* объектов.

**Таблица 3.5. Типовая карта потока приложения**

| Дескриптор истории | Представление | Класс контроллера | Сотрудники | Закрепленные таблицы |
|---|---|---|---|---|
| Timesheet List | `timesheetlist` | `TimeSheetList-Controller` | `TimesheetManager` | Timesheet |
| Enter Hours | `enterhours` | `EnterHours-Controller` | `TimesheetManager` | Timesheet Department |

### Методика дополнения

Сравнив эту карту потока приложения со схемами класса или карточками CRC, вы обнаружите, что эта карта дополняет карточки CRC и схемы класса. Карточки CRC содержат, кроме всего прочего, обязанности каждого класса, чего недостает карте потока приложения. Схемы класса, напротив, представляют отношения, количество элементов, режим (методы), атрибуты, а возможно, и что-то другое.

Собирая вместе информацию о классах в текстовом и табличном форматах, мы сможем также найти имена классов (в больших системах это проблема), а также легко отсортировать их, используя приложение электронных таблиц или утилиты командной строки.

### Расширение карты потока приложения столбцами CRUD

Эта таблица может также быть изменена для применения пользовательских историй, не относящихся к UI, таких как напоминания по электронной почте (Reminder Email: Employee). Например, столбцы представления и класса контроллера могут быть заменены на один столбец, по имени Job (Работа).

Кроме того, вы можете дополнить эту таблицу, разделив столбец *Закрепленные таблицы* на четыре отдельных столбца CRUD (create, read, update, delete – создать, читать, изменить, удалить). Это позволит продемонстрировать не только то, какие из таблиц закреплены, но и как они закреплены за различными классами-сотрудниками. Добавляя столбцы CRUD, вы, по существу, обеспечиваете сквозной поток пользовательской истории (от представления до базы данных и назад) в одной строке нашей таблицы.

## Схема UML для класса

Теперь рассмотрим элементарную схему класса. На мой взгляд, это необязательный шаг (см. вставку о схемах UML), поскольку наши карточки CRC и карта потока приложения предоставляют достаточно информации, чтобы перейти к созданию кода. Но схемы класса могут быть очень удобны, когда применяются правильно.

На рис. 3.5 представлена типичная схема (упрощенная) для классов, определенных на настоящий момент.

**Личное мнение: схемы UML**

За последние годы я использовал несколько типов схем UML, включая упомянутые схемы класса, схемы пакета (мои любимые), а также менее популярные, но не менее замеченные схемы размещения.

Кроме того, мне не очень нравятся популярные ныне циклограммы. Эта схема мне не нравится потому, что, на мой взгляд, она сложна и громоздка. Но я же первый скажу вам, что им нет альтернативны в некоторых случаях (по крайней мере, еще не во всех, однако я провожу сейчас некоторые исследования и надеюсь в конечном счете найти лучшие способы для схем и моделей). Более подробная информация по этой теме приведена на Web-сайте `visualpatterns.com`.

Тем не менее при необходимости я использую схемы UML, поскольку они придают проекту значимость, когда используются а правильном месте и в прааильное время. Фактически, я думаю, что схемы UML наиболее полезны при проектировании инструментальных средств обратной разработки для документирования уже созданных систем (возможно, в ходе передачи системы).

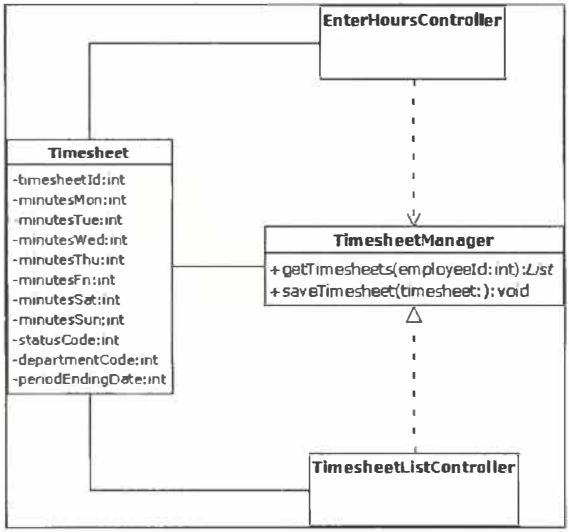



**Рис. 3.5.** Типичная схема классов для приложения Time Expression

Надеюсь, у вас не создалось впечатления, что я имею предубеждение против схем UML, поскольку это не соответствует действительности, тем более что я работал с ними на протяжении многих лет, а их популярность среди многих людей практически сделала UML стандартом. (Фактически, именно поэтому я основываю все свои исследования на работе, которая уже была проделана ранее, а не пытаюсь изобрести колесо заново.)

Схемы UML мне не нравятся, в основном, потому, что они очень быстро становятся слишком сложными, особенно в больших проектах. Еще одна проблема схем UML, на мой взгляд, заключается в том, что им требуются специальные инструментальные средства, которые из-за необходимости приобретения лицензии на программное обеспечение могут быть дороговаты для организации. Кроме того, обучение работе с некоторыми из этих инструментов может потребоаать длительного времени (общеизвестный пример — Rational Rose), что приводит к дополнительным затратам в организации.

Кроме того, более простые инструментальные средства, такие как OpenOffice.org, Microsoft PowerPoint, Microsoft Visio и другие подобные инструменты, позволяют соединять некоторые формы (например, прямоугольники) с использованием соединителей, которые, по существу, являются прямыми или изогнутыми линиями, остающимися прикрепленными к этим объектам при их перемещении. Это очень удобная возможность при создании блок-схем. Я очень часто использую соединители, это будет заметно на многих схемах в этой книге; фактически, почти все они были разработаны с использованием OpenOffice.org!

Я также стараюсь следовать рекомендациям Agile Modeling, таким как моделирование с созданием достаточного количестаа артефактов. Кроме того, я модифицирую их только тогда, когда это действительно необходимо, поскольку большинство артефактоа может быть отброшено после того, как они выполнят свою задачу. После реализации проекта в коде, вы уже будете иметь документацию, в виде кода. Как я упомянул ранее, код может быть преобразован обратно в схему класса (так называемое *обратное проектирование* (reverse engineering)).

Что делает идею документирования относительно тяжелой? Невероятно, но факт, я не могу припомнить ни одного практического проекта программного обеспечения, где документация аелась бы с начала до конца и соответствовала конечному продукту. Дело обстоит так потому, что мы живем в быстро изменяющемся мире иногда с абсолютно нереалистичными

сроками создания программного обеспечения, что крайне затрудняет адекватность документации.

Таким образом, используйте схемы UML, когда необходимо, но не стесняйтесь использовать упрощенные, однако не менее эффективные схемы в свободной форме. Позвольте мне закончить той же самой сентенцией, взятой с Web-сайта `agilemodeling.com`, которую я уже приводил в главе 2: "Ваша задача заключается в том, чтобы выработать общепринятые положения, а не писать подробную документацию".

# Схема UML пакета

В нашем примере приложения, Time Expression, используем для имени пакета префикс `com.visualpatterns.timex`. Если вы уже работали с Java, то, вероятно, знаете, что первая часть имени пакета обычно соответствует имени домена организации, только используется в обратном направлении.

Например, `com.visualpatterns` – это `visualpatterns.com` наоборот (мой Web-сайт). Часть `timex` в имени нашего пакета происходит от названия нашего типового приложения. Остальное – суффиксы для имен нашего пакета, представленного на рис. 3.6, содержащего упрощенную схему UML пакета.

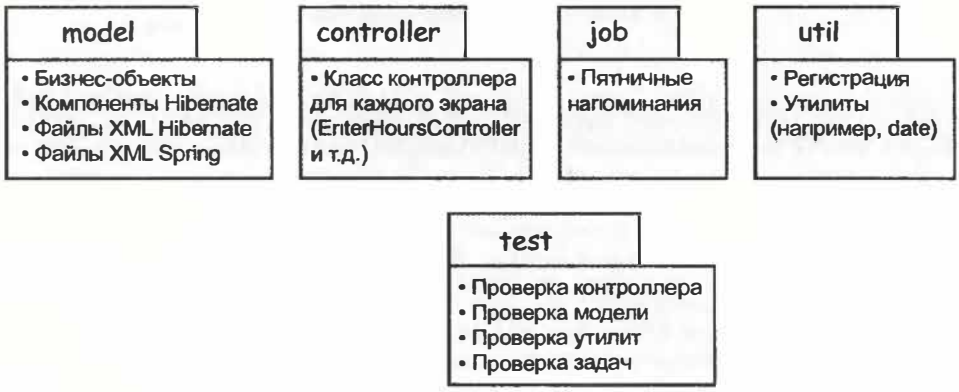



**Рис. 3.6.** Схема UML пакета для приложения Time Expression

**Примечание**

Я выбрал очень простые имена пакетов Java, чтобы они соответствовали нашей схеме MVC проекта. Например, мы могли назвать наш пакет модели (`model`) как-то похоже на домен, но я предпочитаю подбирать имена, подобные, например, именам пакета в архитектуре или на карточках. Таким образом, любой, кто будет просматривать мой код, сможет легко проследить его организацию. Так, полностью определенным именем пакета `model` будет `com.visualpatterns.timex.model`.

Как вы могли уже догадаться, пакет `controller` будет содержать классы, связанные контроллером. Пакет `job` будет содержать нашу задачу напоминания по электронной почте. Пакет `util` содержит общий и/или вспомогательный код.

И наконец, но не в последнюю очередь, пакет `test` будет содержать наш код проверки модуля. Хотя я решил поместить наши классы проверки в отдельный пакет, большинство разработчиков предпочитают хранить проверочные классы в том же

каталоге, что и код реализации, который они проверяют. Это вопрос личных предпочтений, но, по моему мнению, отдельный пакет (каталог) для классов проверки позволяет упростить работу и не загромождать каталоги пакетов реализации.

# Структура каталога

На рис. 3.7 представлена структура каталога, которую мы используем для нашего типового приложения. Это должно выглядеть довольно простым, понятным и знакомым; наиболее известными каталогами здесь являются `src`, `build`, `lib` и `dist`. К этому рисунку придется вернуться и в последующих главах (например, в главах 4, "Установка среды: JDK, Ant и JUnit", 5, "Применение Hibernate для постоянных объектов", и 7, "Среда Spring Web MVC Framework"), когда указанные здесь каталоги будут обсуждаться более подробно. А сейчас рис. 3.7 предоставляет лишь краткое описание всех основных каталогов.

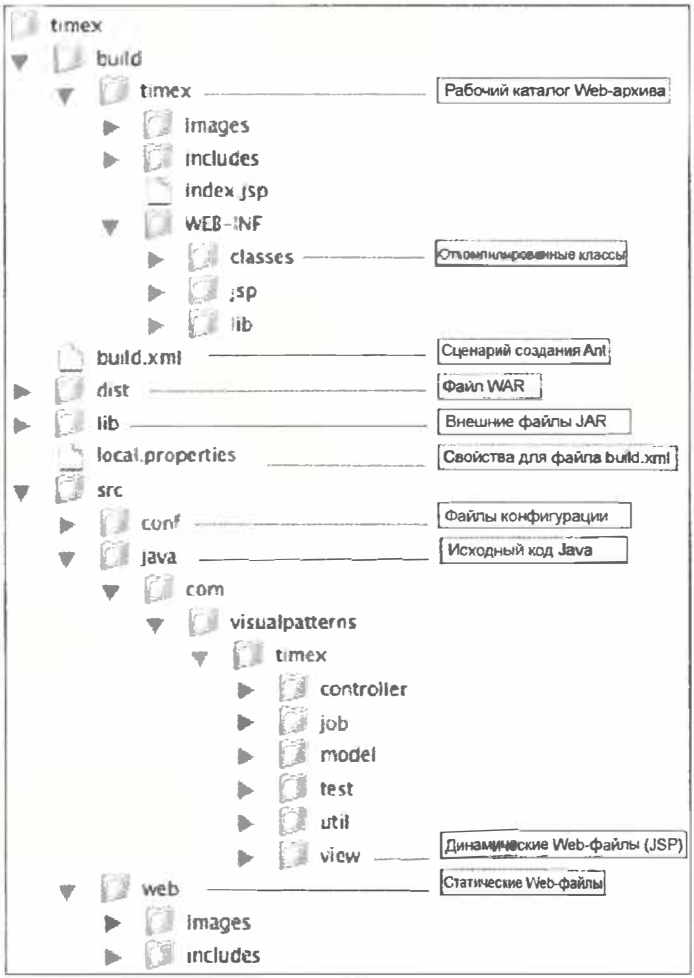



**Рис. 3.7.** Структура рабочего каталога для приложения Time Expression

# Типовые имена файлов

С учетом нашей структуры каталога, представленной на рис. 3.7, можем теперь придумывать имена для файлов некоторых типовых классов, которые мы обсуждали в этой главе. Например, для экрана Timesheet List, приведенного ранее в этой главе, нам, вероятнее всего, понадобятся следующие файлы в каталоге `timex/src/java/com/visualpatterns/timex/`:

- `controller/TimesheetListController.java`
- `model/Timesheet.java`
- `model/TimesheetManager.java`
- `test/TimesheetListControllerTest.java`
- `test/TimesheetManagerTest.java`
- `view/timesheetlist.jsp`

# Этапы разработки

С учетом изученного в этой главе до сих пор, мы можем определить шаги, которые обязательно следует сделать, чтобы разработать код для нашей первой пользовательской истории, от Web UI до базы данных и обратно. Для реализации первой пользовательской истории, вероятнее всего, придется выполнить следующие задачи.

- Установить рабочую среду, включая JDK, Ant и JUnit (см. главу 4, "Установка среды: JDK, Ant и JUnit")
- Создать и проверить классы реализации для пакета модели (использование Hibernate описано в главе 5, "Применение Hibernate для постоянных объектов")
- Создать и проверить классы реализации для пакета контроллера (использование Spring Framework описано в главе 7, "Среда Spring Web MVC Framework")

# Приемочные испытания

Для подробной проверки требований могут служить *приемочные испытания* (acceptance test), как это принято для большинства проектов в стиле Agile. Например, это список операций, которые пользователь может выполнять на данном экране. Идея об использовании приемочных испытаний для требований вполне возможна, поскольку это позволяет уточнить, соответствует ли наше приложение ожиданиям клиента. В данном случае используем их только для проверки наших модулей; однако в реальном мире использование приемочных испытаний для детализации требований становится все более популярным.

Следующие разделы представляют собой список приемочных испытаний и всего остального, что мы реализуем для разрабатываемых пользовательских историй. В реальном мире подобные типы приемочных испытаний были бы предоставлены клиентом.

## Регистрация

- *Идентификатор сотрудника* (employee id) не должен превышать 6 символов. Пароль должен быть от 8 до 10 символов.
- Зарегистрироваться могут только пользователи, обладающие соответствующими правами.

### Список табеля учета

- Пользователю доступен только его табель.

### Ввод часов

- Часы должны быть представлены в виде числовых данных.
- Ежедневное количество не может превышать 16 часов, еженедельное – 96 часов.
- Вместе с часами должен быть указан отдел.
- Часы могут быть введены только с двумя десятичными знаками.
- Сотрудники могут просматривать и редактировать только свои табели учета рабочего времени.

## Другие соображения

Как я упомянул ранее, для успеха необходим только достаточный минимум архитектуры и проекта. Хотя в этой главе мы выполнили вполне приемлемый объем работ по разработке архитектуры и проекта, в этом направлении осталось сделать еще не мало, а именно:

- защита приложения. Это будет описано в главах 7, "Среда Spring Web MVC Framework", и 10, "Кроме основ";
- управление транзакциями. Это будет описано в главе 5, "Применение Hibernate для постоянных объектов". Мы увидим, как программно реализуется управление транзакциями с использованием Hibernate;
- обработка исключений. В главе 10, "Кроме основ", мы рассмотрим обрабатываемые и необрабатываемые исключения, а также ознакомимся с критериями, когда использовать одни, а когда – другие;
- другие возможности. Возможности, необходимые приложению Time Expression, такие как планирование работ и передача сообщений электронной почты, будут описаны в главе 10, "Кроме основ". В последующих главах обсуждаются также другие темы, включая регистрацию, библиотеки дескрипторов и т.д.

**Большое предварительное проектирование или рефакторинг**

Согласно Мартину Фаулеру (Martin Fowler) (`refactoring.com`), "*рефакторинг* (refactoring)[1] является дисциплинарной методикой перестройки структуры тела существующего кода за счет изменения внутренней структуры без изменения его внешнего поведения". Многие разработчики занимаются рефакторингом кода на протяжении лет, но формальное назаание дал этому Мартин Фаулер (и я доволен, что он это сделал). Приступая к созданию кода приложения, вы неизбежно уясните, что лучше делать это, предварительно обдумав. Сюда, например, может относиться удаление избыточного кода или очистка кода. Таким образом, я большой сторонник того, чтобы рефакторинг был всегда доступен не только для кода, но и для проекта базы данных, архитектуры, документации, сценариев интеграции и т.д. Это также облегчает процесс предаарительного проектирования приложения.

Например, недаано на Web-сайте `agiledata.org` я нашел эссе, которое резюмирует мое собстаенное мнение по этой теме; там, в частности, сказано, что "разработчики Agile переключаются вперед и назад между такими задачами, как моделирование данных, моделиро-

[1] Или реорганизация, если дословно. – *Примеч. ред.*

вание объектов, рефакторинг, сопоставление, реализация и настройка производительности". Возьмите, например, эту книгу. Здесь, по существу, я разрабатываю типовое приложение с самого начала, а вместе с ним и книгу. Хот я и осуществил некоторое предварительное планирование, 100% ответов на все вопросы я не имел. Тем не менее это меня не особенно волновало, поскольку в последующих главах я мог переделать архитектуру, проект, код или процесс, используемый для приложения Time Expression. Таким образом, не стоит тратить слишком много времени, пытаясь обдумать каждый возможный случай, когда что-то может пойти не так, как надо, поскольку практически все вопросы можно решить по ходу дела. Короче говоря, вы должны предварительно разработать некоторую исходную архитектуру и дизайн, но при этом иметь в виду, что если появляется возможность что-нибудь улучшить, например упростить или очистить код, и если еще не слишком поздно (в день приемочных испытаний или размещения), то вы вполне можете приняться за переделку!

# Резюме

В этой главе мы рассмотрели обширный материал и достигли следующих целей, поставленных в начале этой главы.

- Разработать схему архитектуры в свободной форме
- Исследовать объекты, используя карточки CRC
- Собрать артефакты; мне нравится для этого использовать карту потока приложения
- Разработать схемы классов и пакетов для приложения Time Expression
- Определить рабочий каталог, структуру и имена некоторых типовых файлов (мы создадим их в последующих главах)
- Определить шаги, которые мы предпримем в следующий главах для сквозной разработки наших экранов
- Обзор списка дополнительных концепций, подлежащих рассмотрению по мере развития нашего типового приложения

В начале главы я обещал вам схему, демонстрирующую текущее положение дел и созданные на настоящий момент артефакты. (Вы думали, я вас обманул?) Эта схема представлена на рис. 3.8. Она со всей очевидностью демонстрирует, что XP имеет артефакты на разных уровнях: концептуальном, физическом и даже реализации. Обратите внимание на то, что линии на рис. 3.8 однонаправленные; это связано с тем, что мы разработали эти артефакты в предыдущей и этой главе. Однако в реальном мире они были бы двунаправленными в связи с наличием обратной связи.

Один раздел полностью посвящен теме артефактов и документации. Помните, что база данных и код – это наиболее важные артефакты из всех! Я не могу подчеркивать это более выразительно. Другие артефакты, которые мы обсуждали в этой книге, не всегда будут необходимы (в зависимости от ваших потребностей). Кроме того, большинство этих необязательных артефактов может быть отброшено после того, как они выполнят свою задачу, поскольку именно так поступают большинство людей, сдав приложение клиенту. Но база данных может пережить все программы, написанные для нее, поэтому она по праву считается наиболее важным компонентом системы.

Раз уж речь зашла о базе данных и коде, значит, пришло время устанавливать нашу среду разработки и используемые инструменты, такие как Ant и JUnit, чтобы в следующих главах начать программировать фактически!

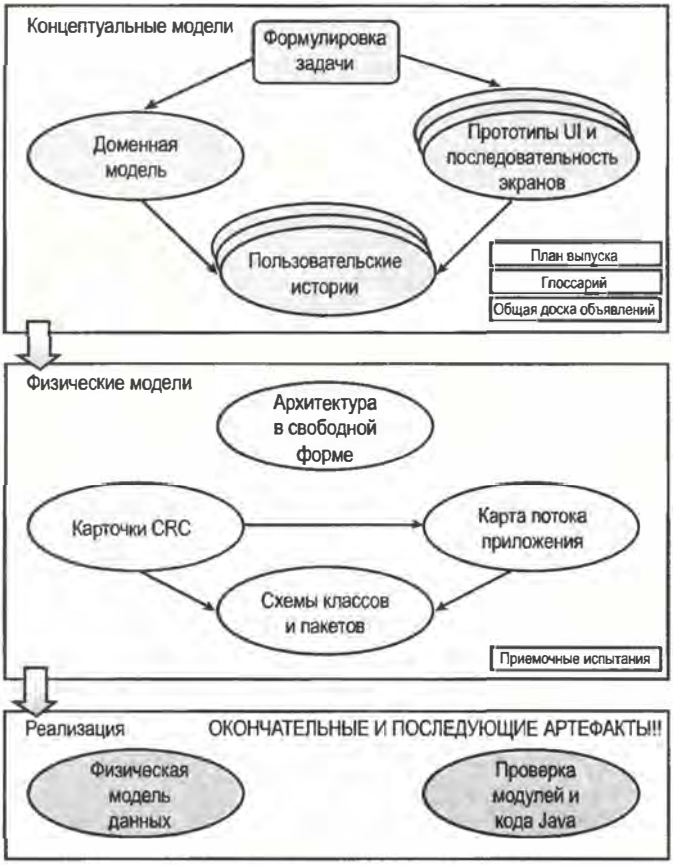



**Рис. 3.8.** Концептуальные и физические артефакты, а также артефакты реализации для приложения Time Expression

# Рекомендуемые ресурсы

Приведенные ниже Web-сайты содержат дополнительную информацию по темам, обсуждаемым в этой главе:

- Agile Model Driven Development `http://www.agilemodeling.com`
- Agile Data `http://www.agiledata.org/`
- Экстремальное программирование `http://extremeprogramming.org`
- Карточки CRC `http://c2.com/doc/oopsla89/paper.html`.

# 4

# Установка среды: JDK, Ant и JUnit


**Стив:** Привет, Рон. Дело движется. Мы пытаемся развернуть приложение, чтобы оно прошло приемочные испытания. Радж устраняет небольшие проблемы с сетью, которые мешают нам подключиться к серверу. В любом случае мы должны быть готовы к нашей встрече по поводу следующей итерации к понедельнику будущей недели.

В этой главе мы получим набор установленных инструментальных средств, которые позволят нам начать разрабатывать, создавать, проверять и развертывать код Java.

Одна из задач этой книги – быть быстрой в прочтении, а не избыточным справочником, полным устаревшей информации. Следовательно, я не собираюсь приводить инструкции по установке для различных инструментальных средств (например, Ant), описанных в этой главе. Вместо этого, я адресую вас к соответствующим Web-сайтам, потому что все затронутые здесь инструментальные средства снабжены вполне достаточными и современными инструкциями по установке.

# Что рассматривается в этой главе

В этой главе мы начинаем установку нашей среды разработки, а именно следующих основных инструментальных средств, обязательных для разработки Java в остальной части этой книги.

- Комплект разработки Java Development Kit (JDK) платформы Java Standard Edition (JSE). Поскольку данная книга о технологиях Java, это программное обеспечение необходимо для работы в первую очередь.
- Ant. Это, фактически, утилита для создания и развертывания приложений на базе Java.
- JUnit. Это просто среда проверки модулей, а также стандартный ныне способ проверки кода Java.
- Заставить все это работать вместе. И наконец, мы соберем все эти три технологии и опробуем их на примере простого модуля.

**Примечание**
Полный код примеров, используемых в этой главе, находится внутри файла zip, доступного на Web-сайте книги[1].

В последующих главах мы добавим к этой среде другие продукты, такие как базы данных, Web-серверы, IDE, библиотеки дескрипторов и др.

# Стандартная платформа и комплект разработки Java (JDK)

Поскольку мы занимаемся разработкой на Java, имеет смысл обзавестись необходимыми для этого инструментальными средствами (например, компилятором). Если на вашей машине установлена устаревшая версия JDK, не подходящая для JUnit и Ant, то вам следует загрузить его последнюю версию с Web-сайта `java.sun.com` и установить на своей машине так, чтобы команды типа `java` могли быть выполнены без указания пути.

После загрузки и установки Java, вы должны быть способны ввести команду **`java -verbion`**, чтобы проверить установку и удостовериться в соответствии версии JDK, как показано ниже.

---

[1] `http://www.samspublishing.com/content/images/0672328968/downloads/0672328968_AgileJavaDev_code.zip`. – *Примеч. ред.*

```
C:\anil\rapidjava\timex>java -version
java version "1.5.0_06"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05)
Java HotSpot(TM) Client VM (build 1.5.0_06-b05, mixed mode, sharing)
```

# Структура каталога

Давайте вернемся к структуре каталога, описанной в предыдущей главе. Структура каталога представлена на рис. 4.1. Важно рассмотреть ее снова, прежде чем мы перейдем к обсуждению Ant. Поэтому обсудим здесь некоторые из наиболее важных подкаталогов.

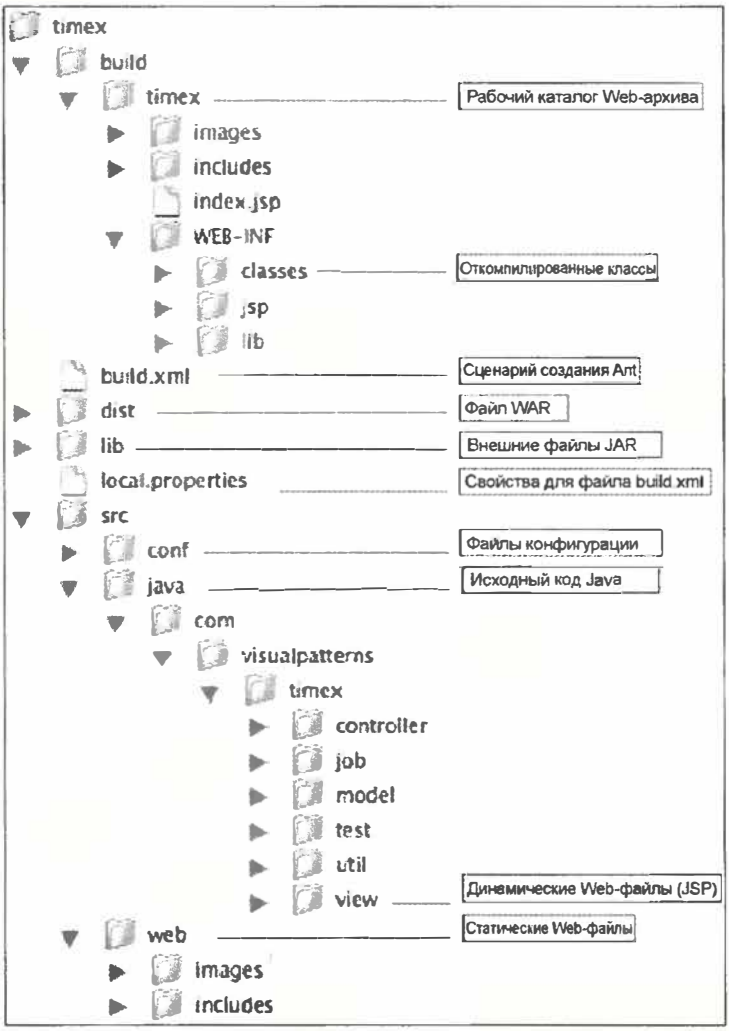



**Рис. 4.1.** Структура рабочего каталога для приложения Time Expression

- Каталог `src` будет содержать все разработанные нами файлы Java, HTML/Javascript, XML и другие исходные файлы.
- Каталог `build` будет содержать наши результаты построения (например, откомпилированные файлы, копии Web-файлов, библиотечных файлов и т.д.).
- Каталог `lib` будет содержать все внешние файлы JAR, необходимые для запуска нашего приложения.
- Каталог `dist` будет содержать наш Web-архив (файл `.war`) со всеми файлами, связанными с Web, включая откомпилированные файлы `.class`, библиотечные файлы `.jar` и др.

# Ant

Я вовсе не преувеличивал, когда утверждал, что Ant (`ant.apache.org`) является ныне, вероятно, одним из важнейших и наиболее популярных инструментов в мире Java! Следовательно, овладение этим инструментом – залог быстрой разработки Java. Поэтому нет ничего удивительного в том, что я рассматриваю этот инструмент непосредственно после JDK, поскольку считаю Ant наиболее важным инструментом и устанавливаю его сразу после Java.

Теперь вы, вероятно, понимаете важность роли Ant в разработке Java. В этой книге мы будем использовать его весьма интенсивно! Например, мы используем его для создания нашего приложения, развертывания и запуска разных программ Java, создания базы данных, выполнения проверок и т.д. Ant был первоначально разработан Джеймсом Дунканом Давидсоном (James Duncan Davidson) из отдела Open Source Program компании Sun Microsystems. Ant представляет собой независимый от платформы инструмент, который устраняет множество проблем и сложностей, присущих таким инструментальным средствам, как Unix make. Вместо того чтобы использовать команды оболочки, специфические для операционной системы, при определении различных задач Ant использует файлы XML. Ant – высоко расширяемый инструмент, главным образом, из-за наличия на рынке огромного количества встроенных и внешних дополнений (с открытым исходным кодом и коммерческих), которые делают его настолько мощным. Кроме того, вы можете легко написать свои собственные специальные расширения.

С учетом того, что Ant сам разработан на Java, он обладает переносимостью и, согласно Web-сайту Ant, был проверен на разных системах, включая Unix, Microsoft Windows, Mac OS X и др. Web-сайт `ant.apache.org` предоставляет вполне достаточную (и современную) информацию о том, как получить и установить Ant на вашей системе; если он еще не установлен, то сейчас самое время сделать это.

Когда Ant будет установлен успешно, вы должны быть способны выполнить команду `ant` без указания полного пути. Иными словами, команда `ant` должна быть указана в вашей переменной окружения `path`, поскольку в остальной части нашей книги мы будем обращаться к ней без полного пути. Например, если вы введете в командной строке **`ant -version`**, то результат должен быть похож на рис. 4.2.


**Рис. 4.2.** Проверка установки ant при помощи команды `ant -version`

## Создание простого файла Ant

По умолчанию Ant ожидает, что файл `build.xml` расположен в текущем каталоге, если вы не сопровождаете команду `ant` какими-либо аргументами. Давайте вспомним упоминавшийся ранее в этой книге файл `build.xml`, содержащий следующий крошечный сценарий Ant. Он предоставляет целевой объект, который выполняет команду echo:

```
<?xml version="1.0"?>

<project name="HelloTest" default="printmessage">
    <target name="printmessage">
        <echo message="Hello, world!"/>
    </target>
</project>
```

Например, если мы сохраним этот код XML в файле по имени `build.xml`, расположим его в том же каталоге, что и Ant, а затем введем команду `ant`, то результат будет выглядеть следующим образом:

```
> ant
Buildfile: build.xml
printmessage:
[echo] Hello, world!
BUILD SUCCESSFUL
Total time: 0 seconds
```

## Создание исчерпывающего файла Ant

Примеры "Hello, world" хороши, но давайте перейдем к созданию исчерпывающего сценария Ant для нашего типового приложения.

> **Примечание**
> Загружаемый код этой книги демонстрирует полный сценарий для нашего типового приложения, включая файл `build.xml` и файл `local.properties`, используемый файлом `build.xml` для загрузки некоторых внешних свойств. Оба эти файла следует поместить в каталог верхнего уровня, `timex/`. Обратите внимание на то, что использование файла `local.properties` демонстрирует удобный способ управления конфигурацией в различных случаях (разработка, проверка, опробование, работа) за счет применения разных файлов свойств.

### Концепция Ant

Прежде чем поэтапно исследовать наш файл `build.xml`, сделаем обзор некоторых фундаментальных концепций Ant.

К ключевым концепциям Ant относятся проект, свойства, целевые объекты, задачи и элементы. *Свойства* (property) – это переменные, которые вы можете устанавливать для сеанса Ant. *Целевые объекты* (target) содержат блоки кода XML, которые запускаются на выполнение в форме задач. *Задачи* (task) – это фактические действия, такие как встроенные задачи `javac`. Задачи в свою очередь могут содержать *элементы* (element), например, `dirset` или `fileset`.

### Поэтапное исследование

Теперь сделаем обзор ключевых объектов нашего файла `build.xml`, однако сначала рассмотрим графическое представление этого файла, представленное на рис. 4.3. Вы можете также вернуться к рис. 4.1, прежде чем начать это исследование, поскольку сценарий построения Ant жестко привязан к данной структуре рабочего каталога.

Первый элемент XML, который должен присутствовать в файле Ant, – это элемент `project`, как показано ниже:

```
<project name="timex" basedir="." default="build">
```

Следующие строки кода, по существу, устанавливают внутренние переменные (свойства) для сценария. Большинство этих свойств связано с различными используемыми входными и выходными каталогами, как показано в приведенном ниже отрывке кода (обратите внимание та то, как мы можем использовать внутренние переменные, заключенные в фигурные скобки и предваренные знаком доллара; например, `${dist.dir}`):

```
<property name="appname" value="timex" />
<property name="lib.dir" value="lib" />
<property name="war.dir" value="build/timex" />
<property name="war.file" value="${dist.dir}/${appname}.war" />
<property name="webinf.dir" value="${war.dir}/WEB-INF" />
```

После установки свойств сценарий устанавливает путь к классу, который используется другими задачами в файле. *Путь к классу* (classpath), по существу, включает два набора файлов: все файлы `.jar` в нашем каталоге `lib/` и файлы классов в каталоге `build/timex/WEB-INF/classes/`, как показано ниже:

```
<path id="master-classpath"
          description="Master CLASSPATH for this script">
    <fileset dir="${lib.dir}">
        <include name="*.jar" />
    </fileset>
    <pathelement location="build/timex/WEB-INF/classes/" />
</path>
```

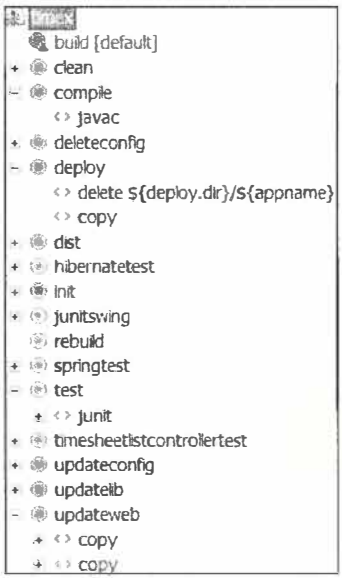



**Рис. 4.3.** Иерархическое представление нашего файла Ant `build.xml`

Следующий в нашем сценарии целевой объект, `init`, гарантирует, что внутри каталога `build/` будут созданы некоторые выходные каталоги, используемые для других задач (для этого используется атрибут `depends` других целевых объектов):

```
<target name="init" description="Setup for build script">
    <mkdir dir="${class.dir}" />
    <mkdir dir="${libs.dir}" />
    <mkdir dir="${jsp.dir}" />
</target>
```

Наши целевые объекты `updateweb`, `updatelib`, `deleteconfig` и `updateconfig` просто копируют или удаляют файлы, связанные с Web и библиотеками в каталоге назначения.

Следующий интересный целевой объект, `compile`, компилирует файлы `.java` каталога `src/java/` в файлы `.class` каталога `build/timex/WEB-INF/classes/`, как показано ниже:

```
<target name="compile"
              escription="Compiles .java files to WAR directory">
    <javac srcdir="${src.dir}" destdir="${class.dir}" debug="true"
        failonerror="true" classpathref="master-classpath" />
</target>
```

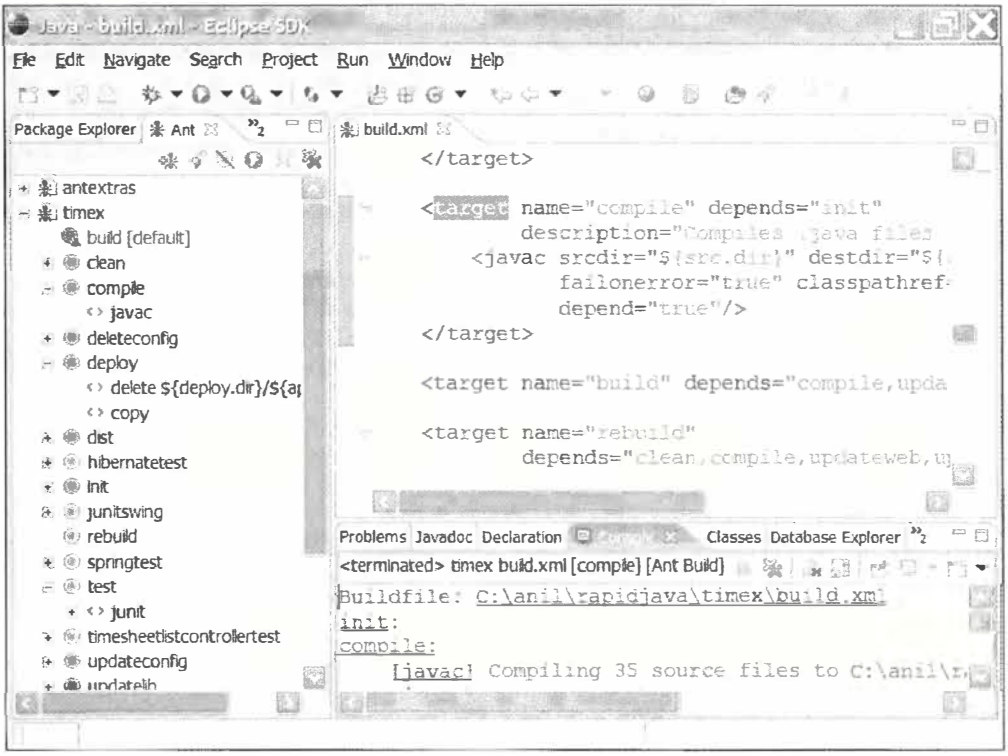



**Рис. 4.4.** Вид Ant внутри Eclipse

Несколько задач рассмотрим также в главе 10, "Кроме основ". Напомню, поскольку Ant посвящены целые книги, я, как вы уже догадались, не буду описывать эту технологию особенно глубоко. Идея этой книги заключается в быстром чтении. Если вы желаете исследовать Ant далее, то можете найти вполне достаточно информации в сети и в печати (то же относится ко всем остальным технологиям, рассматриваемым в этой книге).

# JUnit

Среда JUnit, первоначально описанная в книге *Gang of Four, Design Patterns* Эрихом Гамма (Erich Gamma) и Кентом Беком (являющимся также автором экстремального программирования), представляет собой проверочную среду выполнения Java с открытым исходным кодом, общепринятую для проверки модулей кода Java. Ее можно разгрузить с Web-сайта `junit.org`; этот Web-сайт предоставляет не только инструкцию по установке, но и статьи по проверке модулей и множество советов по разработке предварительных проверок.

*Проектирование методом проверки* (Test-Driven Design, TDD) – это термин, введенный Кентом Беком для описания подхода, позволяющего улучшить проект, упростить код (меньше операторов `print` и `debug`, а также сценариев проверки) и повысить его

эффективность. Поскольку в этой книге мы будем использовать этот подход, создавая предварительные проверки, JUnit имеет смысл рассмотреть непосредственно после разделов по JDK и Ant.

## Автономные пускатели JUnit

Инструкции по установке JUnit, как обычно, находятся на Web-сайте этого продукта, `junit.org`. *Пускатель* (runner) проверок JUnit (главный класс Java) поставляется двух видов: текстовая версия и графическая. Графическая версия доступна в двух вариантах: на базе Java Swing (рекомендуемая) и устаревшая на базе AWT.

После того как JUnit установлен правильно, вы должны быть способны ввести приведенную ниже команду (в каталоге, где установлен JUnit; например, `C:\junit3.8.1`) и запустить версию Swing пользовательского интерфейса JUnit, показанного на рис. 4.5:

```
java -cp junit.jar junit.swingui.TestRunner
```

## JUnit в SDK Eclipse

В следующих главах мы продолжим использовать автономный пускатель проверок JUnit для работы с JUnit, как описано здесь. Но когда далее в книге я представлю Eclipse, мы перейдем к использованию JUnit внутри Eclipse (как показано на рис. 4.6), что позволяет осуществлять отладку и проверку JUnit значительно удобнее. Но иногда возникает необходимость запуска на сервере *пакетной проверки* (batch test) с использованием задачи Ant `junit` или проверки отдельного класса вне IDE при помощи одного из встроенных пускателей JUnit.

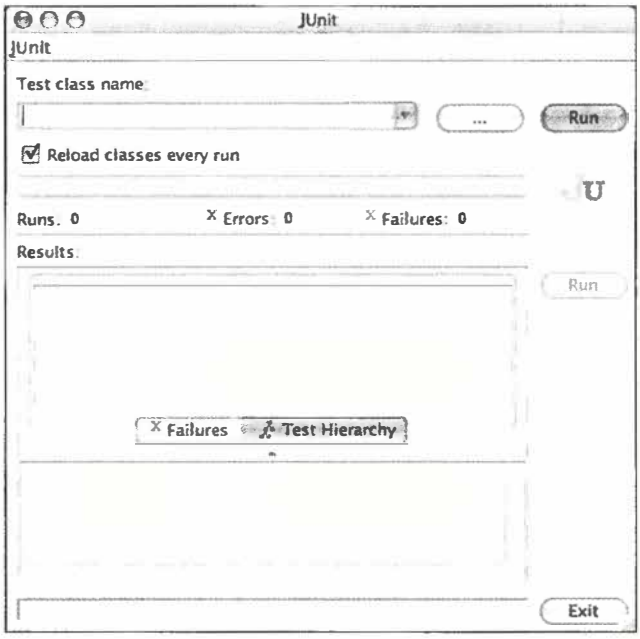



**Рис. 4.5.** Пускатель проверок JUnit на базе Java Swing

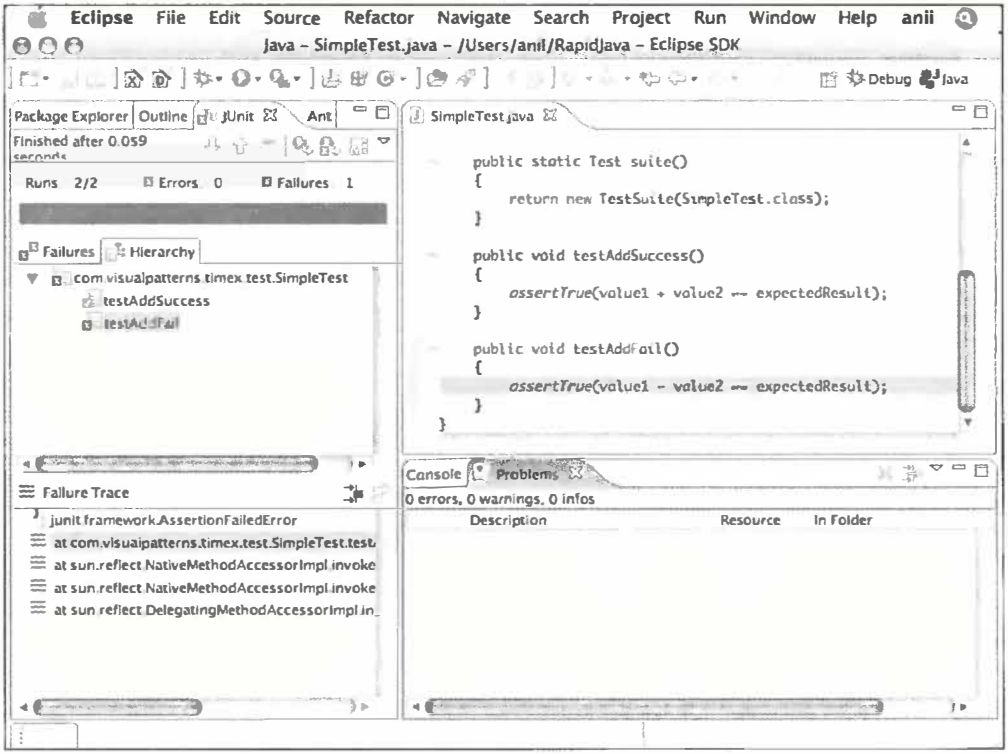



**Рис. 4.6.** Вид JUnit внутри Eclipse

# Как заставить все это работать вместе

Теперь, когда JDK, Ant и JUnit установлены правильно, можем написать пример проверки JUnit и опробовать ее на практике.

Независимо от используемой разновидности JUnit, можем либо передать ему имя проверяемого класса[2], либо ввести его в UI пускателя. Например, если хотим записать очень простой случай проверки, например для проверки факта 2 + 3 = 5, должны будем предпринять следующее:

- разработать проверочный класс JUnit, например `SimpleTest.java`;
- запустить класс в JUnit, используя один из его пускателей.

## Файл `SimpleTest.java`

Файлы кода этой книги (доступные на Web-сайте) содержат файл `SimpleTest.java` полностью. Этот код довольно прост. Имеются два испытательных метода: `testAddSuccess` и `testAddFail`, как показано далее:

```
public void testAddSuccess()
{
```

[2] Автор, вероятно, имел в виду командную строку. — *Примеч. ред.*

```
    assertTrue(value1 + value2 == expectedResult);
}

public void testAddFail()
{
    assertTrue(value1 - value2 == expectedResult);
}
```

Метод `testAddSuccess` засвидетельствует успех, а метод `testAddFail` будет свидетельствовать о неудаче (потому что 2 минус 3 не равно 5). Успех или неудача определены методом JUnit `assert`, который в случае негативного результата проверки передает исключение.

## Методы JUnit `assert`

В предыдущем примере был продемонстрирован метод JUnit `assertTrue`; JUnit предоставляет также несколько других разновидностей методов `assert`, приведенных ниже:

- `assertEquals`
- `assertFalse`
- `assertNotNull`
- `assertNotSame`
- `assertNull`
- `assertSame`
- `assertTrue`

## Запуск примера SimpleTest

Чтобы опробовать код примера SimpleTest, необходимо создать файл `SimpleTest.java`, скомпилировать его, а затем попытаться запустить. Поэтому давайте создадим файл `SimpleTest.java` в нашим каталоге `src/java/com/visualpatterns/timex/test/`. Затем, чтобы скомпилировать исходный код нашего проверочного модуля, просто введем команду **ant** в каталоге `timex/`.

Теперь давайте попробуем запустить наш пример SimpleTest (из каталога, уровнем выше `timex/`) при помощи проверочного пускателя JUnit, как показано здесь:

```
C:\anil\rapidjava\timex>java
↳-cp \junit3.8.1\junit.jar;build/timex/WEB-INF/classes
↳junit.textui.TestRunner com.visualpatterns.timex.test.SimpleTest
```

Мы должны увидеть нечто подобное тому, что представлено на рис. 4.7.

Мы могли бы также запустить класс `SimpleTest.class` в пускателе JUnit Swing, как показано ниже (результат выполнения этой команды показан на рис. 4.8).

```
java -cp \junit3.8.1\junit.jar;build/timex/WEB-INF/classes
↳junit.swingui.TestRunner
com.visualpatterns.timex.test.SimpleTest
```

На самом деле стандартное сообщение об ошибке JUnit `junit.framework.AssertionFailedError`, которое вы видите в обоих случаях, — это хороший знак, поскольку наш метод проверки `testAddFail` потерпел неудачу.

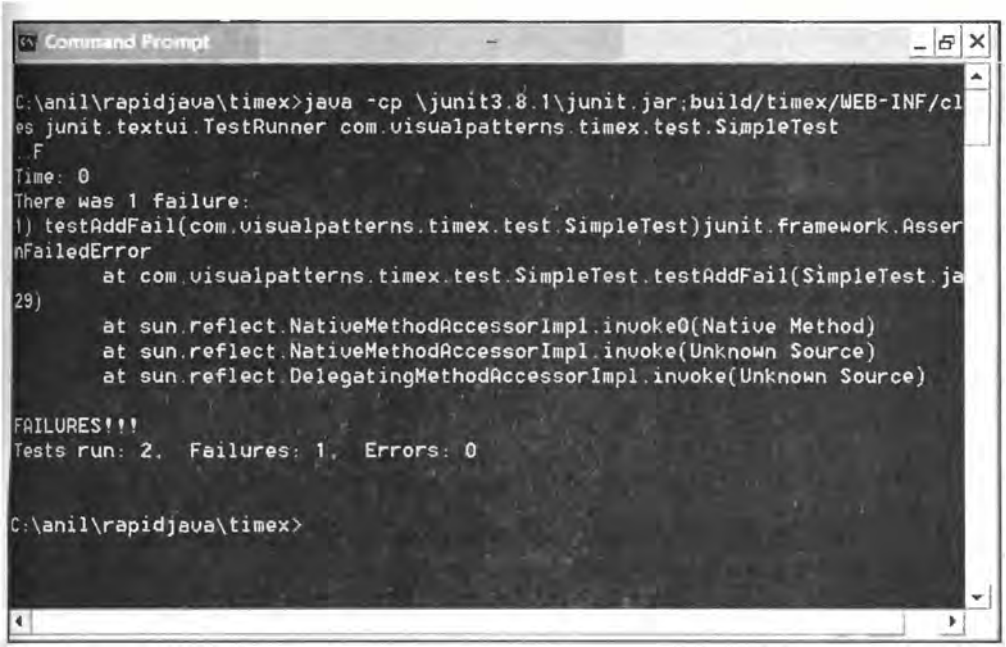



**Рис. 4.7.** Запуск примера SimpleTest в текстовом пускателе

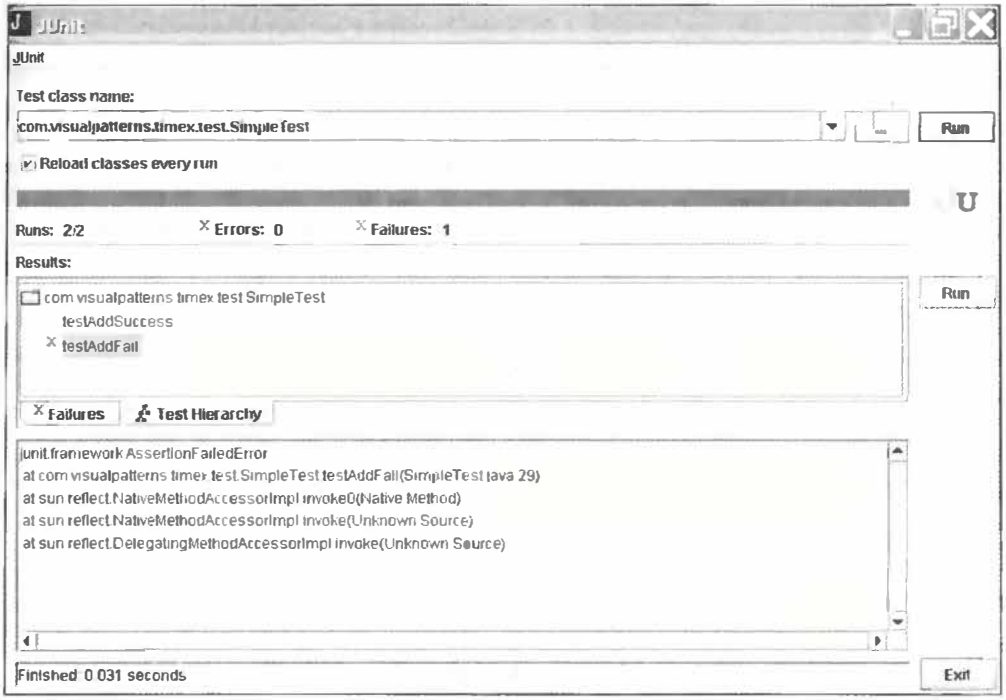



**Рис. 4.8.** Запуск приложения SimpleTest в пускателе JUnit на базе Swing

### Запуск проверки JUnit в пакете

Существует еще один способ запуска JUnit, в качестве задачи Ant. Впоследствии мы рассмотрим, как это сделать.

Сначала скопируем файл `junit.jar` из каталога установки JUnit в каталог *<место-ant>*`/lib`; например, на операционной системе Microsoft Windows введите в командной строке что-нибудь вроде **`copy \junit3.8.1\junit.jar\apache-ant-1.6.5\lib`**. Это позволит нам использовать задачу Ant `junit`.

Затем необходимо скопировать тот же файл `junit.jar` в наш каталог `timex/lib`; это также поможет в компиляции с использованием Ant.

Теперь вернемся к нашему файлу `build.xml`. Этот файл содержит целевой объект по имени `test`, который использует задачу `junit`, как показано в следующем отрывке:

```
<target name="test" depends="compile">
    <junit printsummary="true" showoutput="yes" filtertrace="false">
            <classpath refid="master-classpath"/>
        <batchtest fork="yes">
                <formatter type="plain"/>
            <fileset dir="${class.dir}">
```

Если ввести команду **`ant test`**, то результаты пакетного теста будут помещены в файл по имени `TEST-com.visualpatterns.timex.test.SimpleTest.txt`, расположенный в текущем каталоге. Его отрывок приведен ниже:

```
Testsuite: com.visualpatterns.timex.test.SimpleTest
Tests run: 2, Failures: 1, Errors: 0, Time elapsed: 0.021 sec

Testcase: testAddSuccess took 0.004 sec
Testcase: testAddFail took 0.003 sec
FAILED
```

Это просто замечательно, поскольку вы можете применить несколько методов проверки внутри каждого подкласса примера TestCase. Кроме того, вы можете включать индивидуальные комплекты внутрь других комплектов (без ограничений). Например, можно создать класс по имени `AllTests`, который вызывает комплекты всех других проверочных классов в пакете проверки.

## Разработка с предварительной проверкой и рефакторинг

*Разработка методом проверки* (Test-Driven Development – TDD) вынесла концепцию *разработки с предварительной проверкой* (test first design) на первый план. Этот подход имеет несколько преимуществ, а следовательно, мы будем писать в этой книге *предварительные проверки* (tests first) всякий раз, когда это можно.

Хотя написание предварительной проверки занимает не много времени, зачастую, когда сроки поджимают, вы задаетесь вопросом, а стоит ли его тратить на проверки. Но я нахожу этот способ программирования удобным (если владеешь им хорошо), в частности, потому, что он помогает мне обдумывать конструкцию классов. Кроме того, если вы не располагаете достаточным временем на проверку модулей и устранение дефектов, обнаруженных в ходе проверок функций и проверок пользователя, обнаружите, что этот стиль работы способен существенно сэкономить время.

Запись предварительных проверок обеспечивает несколько преимуществ. Например, это гарантирует, что вы пишете только функциональный код, который будет фактически использоваться (данное утверждение основано на предположении, что вы пишете код для удовлетворения проверок модулей, которые в свою очередь написаны на основании приемочных испытаний, определенных ранее на базе бизнес-требований). Далее, если ваш код прошел проверку модуля и приемочные испытания, эту часть кода можно считать законченной. Кроме того, это поможет вам лучше спроектировать классы, поскольку при записи предварительной проверки вы заранее определяете, как будут фактически использоваться ваши классы и методы. И наконец, предварительная проверка может помочь вам в повторном использовании, поскольку можно быстро проверить по модулям JUnit код, *подвергшийся рефакторингу* (refactored), и удостовериться, что он работает, как и исходная версия (с учетом отсутствия или незначительных изменений внешнего интерфейса, как определено на `refactoring.com`).

Хотя проверка модуля – это только часть корпоративной проверки, разработчики должны выполнять ее всегда. К другим проверкам относятся: проверка функций, *входной контроль пользователя* (User Acceptance Testing – UAT), проверка интеграции системы (называемая также проверкой интерфейса), проверка в предельных режимах и проверка производительности, а также многие другие.

Мы используем JUnit для реализации наших приемочных испытаний. Ниже приведены примеры файлов, демонстрирующих наше соглашение об именовании для проверочных классов:

- `test/TimesheetListControllerTest.java`
- `test/TimesheetManagerTest.java`
- `test/ReminderEmailTest.java`

Я решил хранить наши проверочные классы JUnit в отдельном проверочном пакете (т.е. `com.visualpatterns.timex.test`), поскольку, на мой взгляд, это упрощает проектирование. Но я также видел, что другие разработчики хранят проверочные классы JUnit в том же каталоге, что и проверяемый код. В данном случае, это был бы наш пакет контроллера `TimesheetListControllerTest.java`.

**Личное мнение: установка среды заранее все же необходима**

После почти двух десятилетий разработки программного обеспечения, я поражен, как изменились некоторые из фундаментальных концепций. Несмотря на существенное изменение технологий, основные концепции, такие как установка среды проектирования, разработка, отладка и так далее, остаются неизменными. Одно, я уяснил за последние годы, — это то, что установка среды почти всегда подразумевает немного большее, чем ожидают люди или предусматривает план. Поэтому установка среды заранее жизненно необходима. На основании моего личного опыта, я предпочитаю рекомендовать своим клиентам относительно установки среды следующее. Во-первых, сначала получите минимально достаточную, но полностью функциональную среду (структура каталогов и сценарии создания). Это должно быть согласовано со всеми участниками группы разработчиков программного обеспечения. Во-вторых, получите простую работоспособную демонстрационную версию (например, пользовательский интерфейс для обращения к базе данных). В-третьих, с учетом того, что установка среды продлится дольше, чем ожидалось, заранее зарезервируйте для нее достаточно времени. Это одна из причин, почему многие проекты Agile используют нулевую итерацию или нулевой цикл (Джим Хайсмит (Jim Highsmith)), поскольку демонстрация и установка среды не являются чем-нибудь материальным, с точки зрения клиента; они нужны, скорее, разработчикам для работы и проверок.

# 5

# Применение Hibernate для постоянных объектов

Выпуск 1, Неделя 5, Итерация 2

**Радж:** Когда вы хотите встретиться со Сьюзен по поводу экранов, над которыми мы сейчас работаем?

**Стив:** Давайте позвоним ей прямо сейчас, поскольку она зарубила наш проект отказоустойчивой базы данных. В конце концов, она должна быть активным участником проекта, ведь ей за это платят.

На протяжении нескольких первых лет моей карьеры по разработке программного обеспечения я пришел к выводу, что информация (данные) – это основное достояние организации; однако многие разработчики имеют тенденцию терять из виду данный факт. К сожалению, эту тенденцию подхватили и современные инструментальные средства.

Мы часто используем словосочетание "информационная технология" ("information technology", или просто IT), но что оно означает? По моему мнению, это вполне очевидно – технология для управления информацией. Данные – это основа того, что мы делаем в нашей отрасли, поскольку постоянно перемещаем данные из точки А в точку Б. Не имеет значения, сколько систем проходят при этом данные, появляясь на одном конце (точка А; например, UI) и обычно отображаясь на другом (точка Б; например, отчет). Кроме того, данные и их структура обычно переживают приложения, созданные для их обслуживания; следовательно, это, вероятно, наиболее важный компонент всей архитектуры приложения программного обеспечения.

С учетом моего акцента на данных и базах данных, я опишу в этой книге систему Hibernate перед другими, не менее важными продуктами, такими как Spring Framework и SDK Eclipse.

# Что рассматривается в этой главе

В этой главе мы разработаем классы, необходимые для реализации функциональных возможностей пяти первых пользовательских историй нашего типового приложения Time Expression. Итак, сделаем следующее.

- Выясним, что такое объектно-реляционная технология и какие преимущества она предоставляет.
- Установим HSQLDB – ориентированную на Java облегченную реляционную базу данных.
- Разработаем нашу базу данных.
- Напишем сценарий на *языке определения данных* (Data Definition Language – DDL), который позволит создать таблицы базы данных, а также используем Ant и проверим некоторые типичные вопросы.
- Установим Hibernate, рассмотрим его фундаментальные концепции и начнем работать с ним.
- Рассмотрим простой, а затем немного более сложный пример (наряду с соответствующим классом) использования Hibernate для таблиц `Department` и `Timesheet` приложения Time Expression.
- Обсудим дополнительные возможности Hibernate, используемые при необходимости.

**Примечание**
Полный код примеров, используемых в этой главе, находится внутри файла zip, доступного на Web-сайте книги.

В этой главе предстоит рассмотреть много материала, так что давайте начинать.

# Краткий обзор объектно-реляционного связывания (ORM)

Не секрет, что ныне реляционные базы данных являются наиболее распространенным типом баз данных в большинстве организаций (по сравнению с другими, такими, например, как объектно-ориентированные, иерархические, сетевые). Названия таких продуктов, как Oracle, Microsoft SQL Server, MySQL, IBM DB2 и Sybase, стали общеизвестными.

Среди машинных языков, в свою очередь, нормой стало *объектно-ориентированное* программирование (Object-Oriented – OO). Наиболее обсуждаемыми темами среди разработчиков ныне являются такие языки, как Java, C#, C++, и даже языки объектно-ориентированных сценариев.

При написании большинства приложений, использующих реляционные базы данных и языки OO, приходится создавать код, связывающий реляционную модель с моделью OO. Этот код может оказаться весьма громоздким (из-за встроенных или хранимых процедур SQL) и дополнительных технологий, таких как Entity Beans EJB. Поскольку большинство из нас предпочитает использовать реляционные базы данных и OO, *объектно-реляционное связывание* (Object-Relational Mapping – ORM) стало естественным выбором для работы с объектами POJO (Plain Old Java Objects – простые старые объекты Java), особенно если вы не нуждаетесь в распределенном и защищенном выполнении Entity Beans EJB (что также позволяет связать атрибуты объектов с полями реляционных баз данных).

Хотя вы все еще должны соотносить реляционную модель с моделью OO, связывание обычно осуществляется вне языка программирования, например в файлах XML. Кроме того, если связывание осуществлено для некоторого класса, то экземпляры этого класса вы можете использовать в своих приложениях как объект POJO. Например, вы можете использовать для данного объекта метод `save`, и базовая среда выполнения ORM сохранит данные самостоятельно, не утруждая вас необходимостью писать утомительные операторы `INSERT` и `UPDATE`. Hibernate – это одна из таких сред выполнения ORM, а с учетом ее популярности в мире Java сегодня, мы используем для приложения Time Expression именно ее. В конце этой главы, в разделе "Рекомендуемые ресурсы", перечислены некоторые другие ORM, такие как JDO, iBATIS, Java и Apache ObJectRelationalBridge.

Hibernate поддерживает также стандарт EJB 3.0, поэтому, при необходимости перехода на EJB 3.0, это будет сделать просто (фактически, стандарт EJB 3.0 основан на большинстве концепций и методов Hibernate). EJB 3.0, как вы, возможно, уже знаете, призван упростить работу с технологиями EJB прежних выпусков; например, EJB 3.0 предоставляет удобные API, подобные тем, которые предоставляет Hibernate. Но если вы не нуждаетесь в большинстве услуг, предоставляемых технологией EJB, можете использовать базовую технологию Hibernate отдельно (без такого объемного контейнера EJB, как сервер приложений).

Однако прежде чем перейти к Hibernate, давайте рассмотрим некоторые фундаментальные концепции, общепринятые в технологиях ORM. Впоследствии рассмотрим несколько примеров кода Hibernate Java и файлов XML. После того как приобретете навыки программирования с использованием среды выполнения ORM, вы вряд ли вернетесь к старым способам работы с реляционными базами данных.

**Примечание**
В этой главе мы рассмотрим примеры отношений с разных точек зрения, а именно схемы, код и связи. Например, рис. 5.1 и 5.2 демонстрируют случаи отношений один к многим.

## Идентификатор объекта

*Идентификатор объекта* (object identity или просто object id) – это то, что уникально определяет объект (т.е. запись в базе данных). Это обычно соответствует первичному ключу таблицы базы данных.

## Каскад

Под *каскадом* (cascade) подразумевается действие, осуществляемое над объектом и распространяемое на объекты, связанные с ним. Например, если бы в базе данных мы хотели обеспечить целостность данных таблиц, связанных родительско-дочерними отношениями, то нам бы пришлось удалять записи из дочерней таблицы всякий раз, когда удаляется связанная запись из родительской таблицы, чтобы в базе данных не оставалось никаких *записей-сирот* (orphan record). Аналогично, когда вы читаете родительскую запись, весьма вероятно, придется прочитать и все ее дочерние записи.

Каскадными могут быть все четыре операции CRUD, т.е. *создание*, *чтение*, *модификация* и *удаление* (Create, Read, Update, Delete – CRUD). Каскадные операции зачастую реализуют с использованием *триггеров* (trigger) базы данных.

## Связывание

Прежде чем мы сможем начать работу с объектами, которые сохраняют и возвращают данные из реляционной базы, мы должны создать *связь* (mapping) между таблицами базы данных и классами Java (обычно в файле XML). Файл связи обычно содержит свойства, которые фактически связывают атрибут (переменную) класса со столбцом в базе данных. Если эти концепции вам пока непонятны, не волнуйтесь, впоследствии все объяснится.

Существует несколько разных способов связывания, например горизонтальное, вертикальное и объединяющее. При *вертикальном связывании* (vertical mapping) все классы в иерархии (абстрактные или конкретные) привязываются к разным таблицам. Например, если имеются конкретные классы по имени `Dog` и `Cat`, оба происходящие от абстрактного класса `Animal`, то мы должны были бы иметь в базе данных три таблицы – по одной для каждого класса. При *горизонтальном связывании* (horizontal mapping) к таблице привязан каждый конкретный класс. При *объединяющем связывании* (union mapping) к единой таблице привязывается несколько классов (возможно, принадлежащих той же иерархии).

Хотя вертикальное связывание гибче, оно несколько сложнее, поскольку для извлечения всех данных требуется несколько таблиц. Следовательно, мы используем горизонтальное связывание, поскольку оно проще в проектировании и может обеспечить большую производительность, особенно в простом приложении. Для большей определенности заметим, что наш подход будет подразумевать одну таблицу на один класс.

### Объекты в оперативной памяти или постоянные объекты

Когда мы работаем с технологиями ORM, существует различие между объектами базы данных, располагаемыми в оперативной памяти, и *постоянными объектами* (persisted object). Если объект в базе данных не существует или значение его атрибута совпадает со значением соответствующего столбца в базе данных, то этот объект считается находящимся *в оперативной памяти* (in-memory). Например, Hibernate различает такие состояния объекта, как постоянный, отсоединенный, временный (каждый из них рассматривается далее в этой главе).

Еще одно очевидное различие заключается в том, что удаление объекта из памяти (например, удаление его из коллекции Java) вовсе не обязательно означает физическое удаление записи из базы данных (если, конечно, мы не задали для коллекции в Hibernate режим автоматического выполнения каскадных операций при удалении родительских записей).

## Проект простой базы данных

Теперь, когда рассмотрены некоторые из объектно-реляционных концепций, пришло время установить базу данных, чтобы приблизиться на один шаг ближе к созданию пользовательского интерфейса приложения с помощью Spring Web MVC Framework.

Как я неоднократно упоминал прежде, основное внимание этой книги уделяется, скорее, разработке, а не инфраструктуре. С учетом независимости Java от производителя программного продукта (например, операционной системы, Web-сервера, сервера приложений, базы данных), теоретически, должно быть относительно несложно разработать приложение, используя один продукт, а разворачивать его на другом. В свете этого, я выбрал самые простые (а следовательно, и наименее тяжеловесные) продукты. Реляционная база данных HSQLDB является одним из таких продуктов (обсуждается далее в этой главе); ее мы и используем для приложения Time Expression.

### Денормализация

Прежде чем рассматривать базу HSQLDB, давайте вернемся к нашей доменной модели, приведенной в главе 3, "Архитектура и модель проекта на базе XP и AMDD"; она представлена на рис. 5.1.

С учетом простоты нашего типового приложения Time Expression и его доменной модели, мы сможем создать *физическую модель базы данных* (Physical Database Model – PDM), известную также как *объектно-реляционная* (Entity-Relationship – ER) схема, которая содержит объекты, идентичные нашей доменной модели, но с указанием

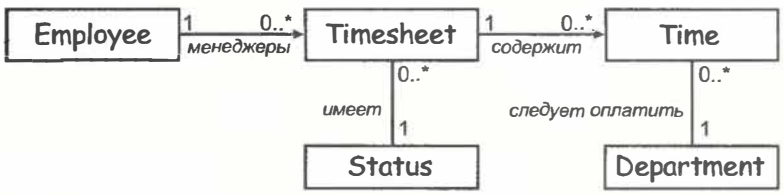



**Рис. 5.1.** Доменная модель приложения Time Expression

столбцов, типов данных и других условий базы данных. Однако давайте немного сократим ее для повышения эффективности и облегчения разработки.

На рис. 5.2 представлена физическая модель PDM, немного сокращенная относительно нашей доменной модели, но с указанием типов данных (например, `varchar`). Денормализация связана только с таблицами `Timesheet` и `Time`.

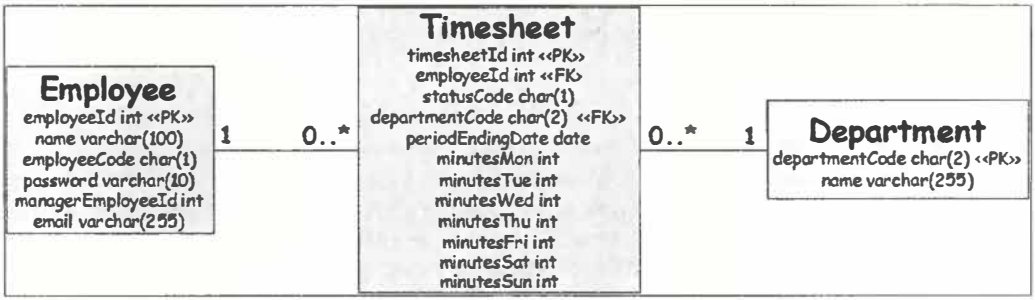



**Рис. 5.2.** Физическая модель базы данных приложения Time Expression

## Соглашение об именовании

Возможно, вы обратите внимание на то, что для имен столбцов и таблиц мы используем *соглашение об именовании* (naming convention), подобное применяемому в Java. Это упрощает нашу задачу, поскольку позволит использовать те же имена во всех артефактах, связанных с приложением Time Expression, а также обеспечить их сквозную непротиворечивость. Другими словами, одинаковые имена будут использоваться в пользовательских историях (дескриптор/имя), для классов контроллера, в модели объектов (домена), для постоянных объектов Hibernate, в коде Java и, наконец, для таблиц и столбцов базы данных (см. рис. 5.2).

Этот подход к именованию упрощает нашу задачу двумя способами. Во-первых, мы не должны заботиться о соглашениях об именовании для каждого слоя, и, во-вторых, это уменьшает объем подробностей, необходимых для определения в наших файлах связывания классов Hibernate, поскольку мы не должны определять соответствующие имена столбцов для каждого связываемого свойства (см. далее в этой главе).

Однако в реальном случае вам вряд ли удастся изменить имена столбцов и таблиц базы данных, поскольку группа разработки базы данных может иметь собственный набор стандартов по именованию. В таком случае, для определения имени столбца базы данных, удобно использовать атрибут столбца Hibernate. Для обеспечения целостности имен я все же настоятельно рекомендую придерживаться стандартов именования вашей организации.

Обратите внимание на то, что для объектов базы данных (таких, как таблицы и счетчики) я предпочитаю имена, начинающиеся с прописной буквы, в то время как для столбцов – имена, начинающиеся со строчного символа.

## Оговорки проекта базы данных

Ниже приведены некоторые оговорки и пояснения модели PDM, представленной на рис. 5.2.

### Неиспользуемые столбцы

Объединив объекты `Timesheet` и `Time` в одну физическую таблицу, можно потратить впустую объем базы данных из-за неиспользуемых столбцов. Например, существует высокая вероятность того, что столбцы `MinutesSat` и `MinutesSun` будут использоваться крайне редко (если сотрудники этой компании не работают сверхурочно или все выходные). Однако преимущества более простого проекта и высокой эффективности, возможно, перевесят недостатки незначительной расточительности объема базы.

### Тип `Int` или `Float`

В таблице `Timesheet` для хранения дробных значений рабочих часов (например, `30` минут или `0.5` часа) мы используем столбец `Minutes<Day>`, а не `Hours<Day>`, который должен был бы иметь тип данных `float`. Это связано с тем, что я хочу продемонстрировать вам возможность использования среды Spring Web MVC Framework (см. главу 7, "Среда Spring Web MVC Framework") для автоматического переноса данных между UI и базой данных. Кроме того, тип `int`, как обычно, занимает в физической памяти существенно меньше пространства, чем тип `float` (2 байт против 4 байт).

### Пароль

В таблице `Employee` имеется столбец `Password`. Как правило, в больших организациях можно использовать нечто вроде централизованной службы аутентификации на базе *упрощенного протокола доступа к каталогу* (Lightweight Directory Access Protocol — LDAP). Но это хорошо сработает и для нашего небольшого (к тому же демонстрационного) приложения Time Expression.

## Сценарий DDL

Теперь, когда модель PDM имеется (см. рис. 5.2), мы можем перейти на следующий уровень (вниз) и написать сценарий DDL, который будет использован для фактического создания базы данных. Наш сценарий DDL будет встроен внутрь одного из наших сценариев Ant (а именно, `timexhsqldb.xml`). Имена таблицы, имена столбцов и типы данных в сценарии DDL соответствуют модели PDM на рис. 5.2.

Наш файл DDL, прежде всего, содержит операторы `CREATE TABLE`. Однако я хотел бы обратить ваше внимание на пару дополнительных элементов.

Во-первых, столбец первичного ключа таблицы `Timesheet` имеет тип данных `identity`, как показано в следующем отрывке кода:

```
CREATE TABLE Timesheet
(
    timesheetId IDENTITY NOT NULL,
```

Как вы, возможно, уже знаете, тип `identity` обеспечивает столбцу базы данных автоматический инкремент (и непосредственно поддерживается Hibernate). У баз данных, которые не поддерживают тип `identity`, мы можем использовать вместо него тип `sequence`.

Во-вторых, здесь присутствуют некоторые проверки данных; это предназначено для использования нашим набором проверок JUnit, рассматриваемых далее в этой главе. Ради простоты, я не создавал никаких первичных или внешних ключей, поскольку так обычно происходит с реальным приложением. Кроме того, основная задача этой главы — продемонстрировать возможности Hibernate, а не проект базы данных.

# Как HSQLDB и Hibernate вписываются в нашу архитектуру

Прежде чем углубиться в вопросы HSQLDB и Hibernate, имеет смысл вспомнить нашу схему архитектуры, которую мы разрабатывали ранее в этой книге. Она представлена на рис. 5.3; обратите внимание, как HSQLDB и Hibernate вписываются в общую картину (справа вверху).

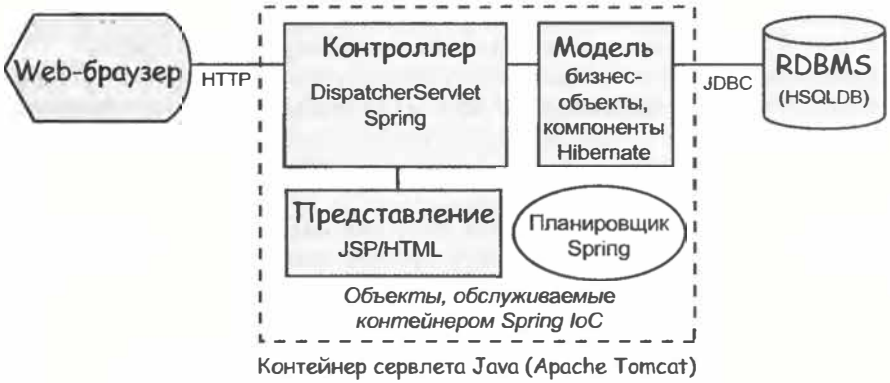



**Рис. 5.3.** Высокоуровневая архитектура приложения Time Expression

В последующих главах, когда мы приступим к разработке кода, связанного с задачами планирования и Web, нам понадобятся классы и база данных, которую мы создадим в этой главе.

# База данных HSQLDB

HSQLDB – это облегченный механизм баз данных Java, который появился в 2001 году. Однако он был жестко связан с проектом Hypersonic SQL Project Томаса Мюллера (Thomas Mueller), который фактически был закончен позже 2001 года. Короче говоря, этот продукт не так уж и нов.

Система HSQLDB предоставляет достаточно большой объем средств, совместимых со стандартом SQL ANSI-92 (и много дополнений для более современных стандартов SQL), причем существенно больший, чем понадобится в этой книге. Кроме того, поддерживается большинство средств, определенных стандартом JDBC 2, и некоторые средства JDBC 3. Популярность механизма HSQLDB существенно возросла за несколько последних лет, и это, вероятно, обусловлено реализацией с открытым исходным кодом и рекламой таких связанных с Java продуктов, как JBoss, OpenOffice.org, JIRA от Atlassian, и многих других.

На момент написания этой книги проект HSQLDB занимал место выше пятидесятого из больше чем 100 000 проектов `SourceForge.net`. HSQLDB находится по адресу `http://hsqldb.org`. Там имеется вполне достаточная инструкция по установке и настройке. В этой книге я использую версию 1.8.x.

## Сервер HSQLDB и соответствующие задачи Ant

Теперь необходимо запустить сервер HSQLDB и создать базу данных, используя наш файл DDL. Однако сначала скопируем файл `hsqldb.jar` из каталога установки HSQLDB в наш каталог `lib/`; например, на моей системе Microsoft Windows XP я ввел следующее:

```
copy \hsqldb\lib\hsqldb.jar \anil\rapidjava\timex\lib\
```

Для запуска сервера и создания базы данных мы используем наш сценарий Ant `timexhsqldb.xml`. Этот файл находится в каталоге на уровень выше нашего типового приложения (в моем случае, это каталог `C:\anil\rapidjava\timex`).

С учетом правильной установки и настройки сервера HSQLDB, мы можем теперь ввести команду **`ant -f timexhsqldb.xml starthsql`** (см. ниже), которая запустит сервер HSQLDB.

```
C:\anil\rapidjava\timex>ant -f timexhsqldb.xml starthsql
```

Чтобы запустить сценарий DDL для создания нашей базы данных на сервере HSQLDB, в другом командном окне следует ввести команду **`ant -f timexhsqldb.xml execddl`**, как показано ниже:

```
C:\anil\rapidjava\timex>ant -f timexhsqldb.xml execddl
```

Прежде чем переходить дальше, рассмотрим часть файла `timexhsqldb.xml`.

Следующие свойства связаны с HSQLDB, т.е. `hfile` указывает на локальный набор файлов в каталоге `timex/data/`, свойство `halias` – это псевдоним, который мы используем в наших клиентских приложениях для подключения к серверу HSQLDB, а свойство `hport` – это номер порта, который будет прослушивать сервер HSQLDB:

```
<property name="hfile" value="-database.0 data/timexdb"/>
<property name="halias" value="timex"/>
<property name="hport" value="9005"/>
```

Далее целевой объект Ant `starthsql` запускает сервер HSQLDB, используя встроенную задачу Ant `java`, как показано далее:

```
<java fork="true "
    classname="${hclass}" classpath="${hjar}"
    args="${hfile} -dbname.0 ${halias} -port ${hport}" />
```

Целевой объект Ant `execddl` использует встроенную задачу `sql` для выполнения сценария SQL DDL, как показано далее:

```
<sql classpath="${hjar}"
    driver="org.hsqldb.jdbcDriver"
    url="jdbc:hsqldb:hsql://localhost:${hport}/${halias} "
    userid="sa" password=""
    print="yes" >
```

## Диспетчер базы данных HSQLDB и SqlTool

В документации HSQLDB указаны два инструментальных средства, связанных с этой системой: диспетчер базы данных HSQL (GUI) и SqlTool (для командной строки). Это прекрасные инструменты для работы с нашей базой данных. Кроме того,

в нашем файле timexhsqldb.xml вы найдете две очень удобные задачи Ant, hsqldm и sqltool, которые применяются для запуска этих двух инструментов. Например, для запуска диспетчера базы данных HSQL, введите в командной строке следующее:

```
ant -f timexhsqldb.xml hsqldm
```

После того как диспетчер базы данных появится на экране, мы сможем изменять некоторые параметры и работать непосредственно с нашей базой данных в режиме GUI (подразумевается, что сервер HSQLDB выполняется в другом окне).

## Режимы HSQLDB: постоянный и оперативной памяти

Уведомив читателя о наличии разных режимов работы HSQLDB (таких, как локальный или сервер в оперативной памяти либо постоянный), мы используем сервер в постоянном режиме. Например, мы могли бы использовать те же файлы базы данных HSQLDB (находящиеся в каталоге timex/data/) следующим образом:

```
jdbc:hsqldb:file:${catalina.base}/webapps/timex/WEB-INF/data/timexdb
```

Эта возможность будет использована в следующем разделе при размещении HSQLDB в файле архива.

## Размещение HSQLDB в развертываемом файле архива

Дополнительным преимуществом HSQLDB является ее достаточно маленький размер, вполне приемлемый для размещения в оперативной памяти полностью. Например, мы могли бы развернуть наше типовое приложение вместе с его HSQLDB в том же файле Web-архива (WAR), что, по существу, сделает файл WAR полностью самодостаточной системой, не нуждающейся во внешней базе данных!

**Личное мнение: данные — самое ценное достояние клиента!**

За многие годы разработки программного обеспечения я все еще замечаю, что некоторые люди упускают из виду ключевой пункт *информационных технологий* (Information Technology — IT). На мой взгляд, IT означает технологию для управления информацией или данными, которые хранятся в базе данных.

Данные — это ценное достояние клиента, а следовательно, использовать их может несколько поколений разных программ. Вот почему доменная модель, физическая модель данных и рефакторинг базы данных существенно более важные аспекты разработки программного обеспечения, чем, например, мощность инструментальных средств или дополнительные уровни ненужных абстракций в вашем коде.

Когда разрабатываете базу данных, важно учитывать, что она может использоваться несколькими приложениями, а не только единственным, хорошо разработанным, объектно-ориентированным, многоуровневым приложением. Например, к базе данных, для создания клиентских отчетов, могут также обращаться средства составления отчетов. Поэтому структура базы данных не должна никоим образом зависеть от конкретного приложения. Кроме того, даже первоначальное приложение, разработанное для базы данных, может за несколько лет устареть, но сама база данных будет продолжать работать еще очень долго.

Более подробная информация о методах рефакторинга базы данных содержится на Web-сайте agiledata.org. Вы можете также посетить Web-сайт domaindrivendesign.org, который дополняет данный Web-сайт. Например, приведенную ниже строку я нашел в статье Эрика Еванса (Eric Evans) на этом Web-сайте: "сложность, которой мы должны заняться, — это сложность самого домена, а не технической архитектуры, не пользовательского интерфейса и даже не специфических особенностей".

Подведем итог, данные — это достояние клиента, поэтому сосредоточьтесь на создании правильной доменной модели и структуры базы данных, используя комбинацию предварительного проектирования и рефакторинга базы данных по мере необходимости.

# Работа с Hibernate

Hibernate совсем недавно завоевал популярность в мире разработки приложений баз данных Java. Хотя такие продукты, как Toplink и другие, находятся на рынке много лет, Hibernate имеет реализацию с открытым исходным кодом (а, следовательно, бесплатен). Это стабильный, зрелый, хорошо зарекомендовавший себя и относительно простой для изучения продукт; таковы, вероятно, некоторые из причин его популярности. Hibernate был выпущен несколько лет назад, но недавно был приобретен группой JBoss. (Но этот продукт остается доступным и как автономный проект с открытым исходным кодом.) Отказоустойчивая среда выполнения Hibernate может работать с реляционными базами данных, используя Java. Это справедливо также при использовании JDBC или Entity Beans.

## Ненужность DAO или DTO

Дополнительные работы по связыванию стоят того, поскольку наш код станет надежнее и проще, а кроме того, отпадет необходимость в *объектах доступа к данным* (Data Access Object – DAO), которые обычно используются для собственно сохранения. Нам также не будут нужны *объекты передачи данных* (Data Transfer Object – DTO), которые используются для инкапсуляции бизнес-данных и перемещения их между слоями приложения.

## Поддерживаемые базы данных

На момент написания этой книги Hibernate поддержал следующие базы данных (для поддержки других баз данных сообщество прилагает усилия):

- DB2
- HSQLDB
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- SAP DB
- Sybase
- TimesTen

**Примечание**
Поддержку базы данных в Hibernate осуществляют классы диалекта SQL, такие как `org.hibernate.dialect.HSQLDialect`, `org.hibernate.dialect.OracleDialect`, `org.hibernate.dialect.MySQLDialect` и т.д.

## Hibernate и EJB 3.x

Здесь стоит упомянуть еще об одном, участники группы Hibernate/JBoss входили в состав группы экспертов EJB 3.0, помогавшей упростить спецификацию EJB. Поэтому нет ничего удивительного в том, что последняя версия Hibernate поддерживает спецификацию EJB 3.0. Но в этой книге мы не будем рассматривать стандарт EJB 3.0, поскольку она посвящена более простым средам выполнения (с открытым исходным кодом), а не сложным спецификациям, требующим, к тому же, установки коммерческих серверов приложений.

## Простая проверка установки Hibernate

Прежде чем погрузиться в концепции и терминологию Hibernate, давайте рассмотрим простую программу Hibernate и связанную с ней настройку. В следующем разделе описаны шаги, которые необходимо предпринять для запуска нашей первой проверочной программы, SimpleTest. Но сначала давайте еще раз обратим внимание на структуру каталога, созданную в главе 3, "Архитектура и модель проекта на базе XP и AMDD"

Структура рабочего каталога приложения Time Expression представлена на рис. 5.4. Рассмотреть ее снова важно, потому что в этой главе мы создадим несколько файлов и будем обращаться к ним, используя имена с относительным путем. Например, `model/Department.java` означает файл `Department.java` в каталоге `timex/src/java/com/visualpatterns/timex/model/`.

### Файлы XML Hibernate и соответствующие файлы Java

Мы создадим три типа файлов Hibernate (обсуждаемых далее), файл конфигурации Hibernate, соответствующие классы Java, а также файлы таблиц связывания в том же каталоге. Это рекомендуемая практика, в документации Hibernate и примерах. Для файлов связывания Hibernate обычно используется следующее соглашение об именовании: к имени класса Java добавляется суффикс `.hbm.xml`, например `Timesheet.hbm.xml`.

### Файл конфигурации Hibernate (`hibernate.cfg.xml`)

Сначала мы создадим в каталоге `timex/src/java/com/visualpatterns/timex/model/` файл по имени `hibernate.cfg.xml`. Этот файл будет содержать определение приложения SessionFactory (обсуждается далее в этой главе) и ссылку на наш первый файл связывания, `Department.hbm.xml`. Давайте рассмотрим некоторые из наиболее интересных строк кода этого файла.

Следующие строки демонстрируют настройку, связанную с HSQLDB (как в файле `timexhsqldb.xml`, приведенном ранее):

```
<property name="connection.driver_class">
    org.hsqldb.jdbcDriver
</property>
<property name="connection.url">
    jdbc:hsqldb:hsql://localhost:9005/timex
</property>
<property name="connection.username">sa</property>
```

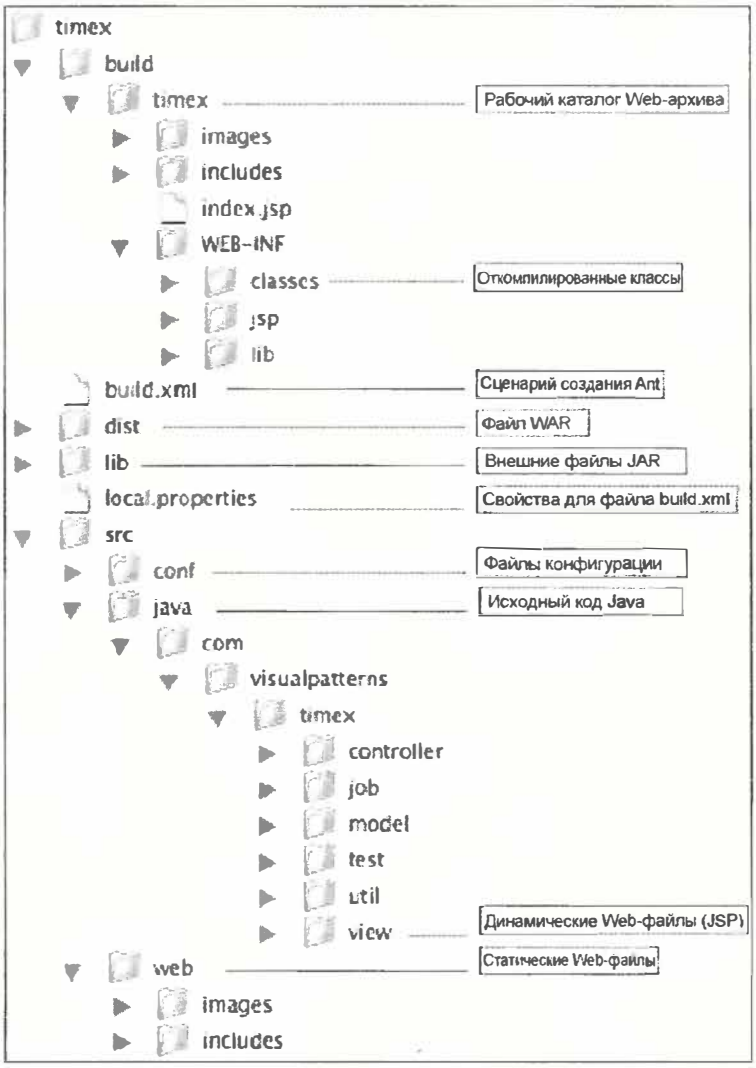



**Рис. 5.4.** Структура рабочего каталога приложения Time Expression

Приведенные ниже строки файла `hibernate.cfg.xml` демонстрируют ссылки на файлы связывания, которые мы создадим в этой главе:

```
<mapping resource="Department.hbm.xml"/>
<mapping resource="Employee.hbm.xml"/>
<mapping resource="Timesheet.hbm.xml"/>
```

Используя законченный файл `hibernate.cfg.xml`, мы сможем создать Hibernate SessionFactory (обсуждается далее в этой главе).

### Файл связывания (Department.hbm.xml)

Мы создадим наш первый файл связывания, Department.hbm.xml, в каталоге timex/src/java/com/visualpatterns/timex/model/.

Начнем с самого простого, а именно с таблицы Department, поскольку это одна из самых простых таблиц. Кроме того, ее предстоит использовать в более сложном примере далее в этой главе. Давайте рассмотрим файл Department.hbm.xml немного подробнее.

Следующая строка кода свяжет наш класс Java с таблицей базы данных:

```
<class name="com.visualpatterns.timex.model.Department"
↳table="Department">
```

Следующая строка кода установит идентификатор departmentCode в качестве идентификатора объекта (как было описано ранее) и в качестве первичного ключа базы данных, а также свяжет их вместе:

```
<id name="departmentCode" column="departmentCode">
```

Значение generator class="assigned", показанное ниже, уведомляет Hibernate о том, что установку значения этого объекта мы осуществим в классе Java и Hibernate не должен предпринимать никаких специальных действий, подобных получению следующего значения последовательности из столбца автоинкрементна (как, например, у типа данных HSQLDB identity):

```
<generator class="assigned"/>
```

Эти строки связывают остальные поля таблицы Department, т.е. столбец name со свойством name в файле класса Department.java (обсуждается далее):

```
<property name="name" column="name"/>
```

### Код Java

Мы создадим два класса Java, один по имени com.visualpatterns.timex.model.Department и второй com.visualpatterns.timex.test.HibernateTest.

#### *Файл Department.java*

Файл Department.java (в каталоге src/java/com/visualpatterns/timex/model) содержит простой класс JavaBean, который предоставляет методы чтения (get или getter) и записи (set или setter) для значения двух следующих переменных:

```
String departmentCode;
String name;
```

#### *Файл HibernateTest.java*

Теперь мы напишем некоторый простой код, выполняющий две задачи: проверку установки Hibernate, а также демонстрирующий пример использования Hibernate. Давайте рассмотрим наш файл HibernateTest.java (в каталоге src/java/com/visualpatterns/timex/test) последовательно.

Несколько первых строк кода демонстрируют, как мы получаем класс Hibernate SessionFactory и извлекаем одну запись из таблицы Department со значением "IT" столбца departmentCode:

```
SessionFactory SessionFactory = new Configuration().configure()
    .buildSessionFactory();
Session session = SessionFactory.getCurrentSession();
Transaction tx = session.beginTransaction();
Department department;
department = (Department) session.get(Department.class, "IT");
System.out.println("Name for IT = " + department.getName());
```

Следующие строки демонстрируют, как получить и обработать объект `java.util.List` таблицы `Department`.

```
List departmentList = session.createQuery("from Department").list();
for (int i = 0; i < departmentList.size(); i++)
{
    department = (Department) departmentList.get(i);
    System.out.println("Row " + (i + 1) + "> " + department.getName()
       + " (" + department.getDepartmentCode() + ")");
}
```

Последняя рассматриваемая строка кода закрывает класс `SessionFactory`.

```
SessionFactory.close();
```

Обратите внимание на то, что файл `HibernateTest.java` представляет простой пример использования Hibernate, когда весь код сосредоточивается в одном (главном) методе. Далее в этой главе мы рассмотрим лучший способ создания класса `SessionFactory` и последующего получения его объекта `Session`.

Теперь давайте попробуем запустить проверку, используя наш файл Ant `build.xml`, представленный в главе 4, "Установка среды: JDK, Ant и JUnit". Наш целевой объект Ant, `hibernatetest`, имеет следующий вид:

```
<target name="hibernatetest" depends="build">
    <java fork="true" classpathref="master-classpath"
        classname="com.visualpatterns.timex.test.HibernateTest"/>
</target>
```

Чтобы выполнить проверку, необходимо предпринять следующее.

- Сменить (`cd`) каталог на расположенный выше (`timex/`).
- Ввести команду **`ant hibernatetest`**, как показано на рис. 5.5.

Обратите внимание на сообщения об ошибках, например `package org.hibernate does not exist`. Это означает, что пришло время загрузить и установить Hibernate в нашей системе!

## Установка Hibernate

Hibernate можно найти по адресу `http://hibernate.org`. Теперь, следуя предоставляемой на этом сайте инструкции, загрузим и установим Hibernate в рекомендуемый (или заданный по умолчанию) каталог.

После установки Hibernate, скопируем все рекомендуемые документацией Hibernate библиотеки (например, `hibernate3.jar` и `antlr.jar`) в каталог `rapidjava/lib`.

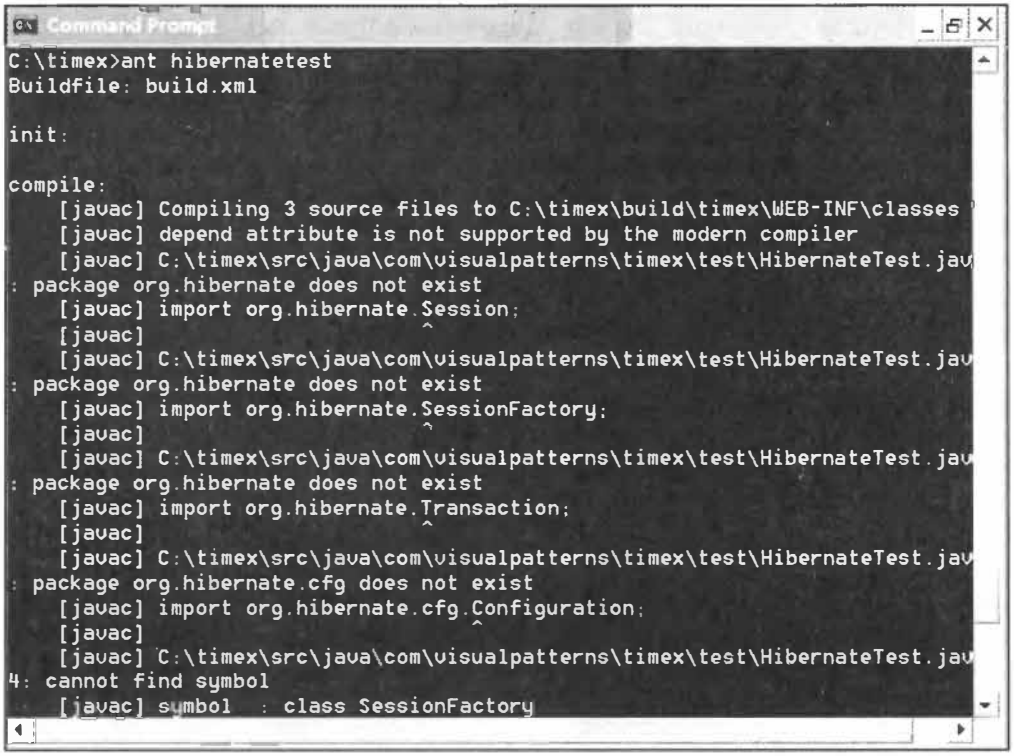



**Рис. 5.5.** Ошибка Ant вызвана отсутствием указания файла `lib/hibernate3.jar` в пути к классу

Обратите внимание на то, что необходимо также скопировать файлы `ehcache-1.1.jar` и `antlr-2.7.6rc1.jar` (ныне не упоминающийся в документации Hibernate). Таким образом, в моем каталоге `timex/lib/` должны находиться следующие файлы:

- `antlr-2.7.6rc1.jar`
- `asm-attrs.jar`
- `asm.jar`
- `cglib-2.1.3.jar`
- `commons-collections-2.1.1.jar`
- `commons-logging-1.0.4.jar`
- `dom4j-1.6.1.jar`
- `ehcache-1.1.jar`
- `hibernate3.jar`
- `jta.jar`
- `log4j-1.2.11.jar`

Прежде чем переходить к проверке, необходимо временно изменить файл `hibernate.cfg.xml`. Поскольку реализован пока только файл `Department.hbm.`

xml, необходимо временно удалить следующие строки кода из нашего файла hibernate.cfg.xml:

```
<mapping resource="Timesheet.hbm.xml"/>
<mapping resource="Employee.hbm.xml"/>
```

И наконец, мы можем повторно выполнить команду **ant hibernatetest**. Если мы запустим Ant, как показано ранее на рис. 5.5, то на сей раз ее выполнение будет успешным, как показано на рис. 5.6!

Теперь мы можем вернуть в файл hibernate.cfg.xml следующие две строки кода:

```
<mapping resource="Timesheet.hbm.xml"/>
<mapping resource="Employee.hbm.xml"/>
```

Обратите внимание на предупреждающие сообщения log4j (рис. 5.6). Мы могли бы игнорировать их, поскольку они безопасны. Однако мы пойдем дальше и создадим минимальный файл log4j.properties (доступный в файле архива zip кода этой книги) в каталоге timex/src/conf. Более подробная информация о регистрации приведена в главе 9, "Регистрация, отладка, мониторинг и профилирование".

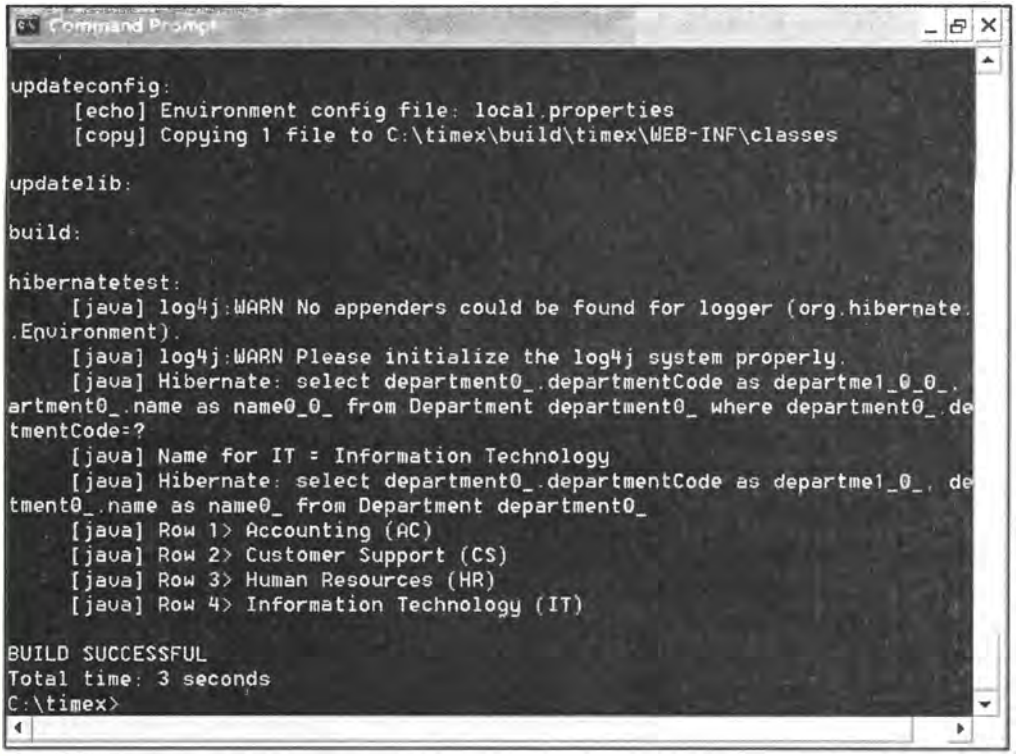



**Рис. 5.6.** Вывод класса HibernateTest

## Основы Hibernate

Теперь, после краткого обзора кода Java и файлов XML, связанных с Hibernate, рассмотрим некоторые из основных высокоуровневых концепций, прежде чем переходить к изучению более сложного кода Hibernate для приложения Time Expression.

### Диалект

Hibernate предоставляет классы диалекта (`dialect`) для различных поддерживаемых баз данных, упоминавшихся ранее. По существу, это позволяет гарантировать правильное и оптимизированное использование SQL для конкретной базы данных. Например, для базы данных HSQLDB мы используем класс `org.hibernate.dialect.HSQLDialect`.

### Объекты `SessionFactory`, `Session` и `Transaction`

Объект `SessionFactory`, как вы могли, вероятно, догадаться, — это коллекция объектов `Session`. Каждый объект `SessionFactory` связан с определенной базой данных. Объект `Session`, по существу, — это оболочка для соединения JDBC, а также источник объектов `Transaction`. Объекты `Transaction` — это оболочка для собственно транзакций, как правило, транзакций JDBC.

### Встроенный буфер подключений

Побочным, но весьма полезным эффектом применения Hibernate является предоставляемый им встроенный буфер подключений к базе данных (следовательно, для нас меньше проблем). Буферизация подключений, как вы, возможно, знаете, используется для создания специального буфера открытых подключений к базе данных (см. свойство `connection.pool_size` в нашем файле `hibernate.cfg.xml`). Используя буфер подключений, мы можем достигать большей эффективности использования базы данных за счет многократного использования уже существующих открытых подключений. Кроме того, мы получаем выигрыш по производительности, поскольку повторное использование открытых подключений позволяет избежать задержек на открытие и закрытие подключений к базе данных.

### Работа с записями базы данных (как с объектами Java)

Некоторые из методов, доступных в Hibernate, позволяют работать с записями базы данных, как с объектами. Наиболее известны такие методы, как `save`, `load`, `get`, `update`, `merge`, `saveOrUpdate`, `delete` и `createQuery` (некоторые из них описаны далее в этой главе).

Еще один примечательный интерфейс, который следует упомянуть, — это `org.hibernate.Query`. Он возвращается при вызове метода Hibernate `Session.createQuery` в нашем файле `HibernateTest.java`. Класс `Query` применяется для получения группы записей в форме объекта `java.util.Collection` (например, Hibernate предоставляет такие элементы связывания, как `array` (массив), `set` (набор), `bag` (мешок) и т.д.).

Последний интерфейс, который стоит упомянуть здесь, — это `org.hibernate.Criteria`. Он может применяться при обращении к базе данных объектно-ориентированным способом, как альтернатива классу `Query` (который является ориентированным на HQL).

Примеры большинства этих интерфейсов и методов мы рассмотрим далее в этой главе.

### Состояния объекта

Hibernate различает три состояния экземпляров объекта: *постоянный* (persistent), *отсоединенный* (detached) и *временный* (transient). Постоянные объекты – это объекты, которые в настоящее время связаны с сеансом Hibernate; как только сеанс будет закрыт (или объект отключен), объект становится отсоединенным. Hibernate гарантирует, что объекты Java в постоянном состоянии (для активного сеанса) соответствуют некой записи (записям) в базе данных. Временные объекты – это объекты, которые не связаны (и, вероятно, никогда не были связаны) с сеансом Hibernate, а также не имеют идентификатора объекта.

### Типы данных

Hibernate поддерживает большее количество типов данных Java, SQL и собственно Hibernate, чем вам, вероятно, понадобится для типичного приложения. Кроме того, Hibernate может автоматически преобразовывать один тип данных в другой, используя тип данных определенного свойства, предоставленного в файле связывания объекта/класса.

Вот лишь частичный список типов данных, поддерживаемых Hibernate: `integer`, `long`, `short`, `float`, `double`, `character`, `byte`, `boolean`, `yes_no`, `true_false`, `string`, `date`, `time`, `timestamp`, `calendar`, `calendar_date`, `big_decimal`, `big_integer`, `locale`, `timezone`, `currency`, `class`, `binary`, `text`, `serializable`, `clob` и `blob`.

### Язык запросов Hibernate (HQL)

*Язык запросов Hibernate* (Hibernate Query Language – HQL) – надежный, похожий на SQL язык запросов, который, к тому же, не чувствителен к регистру. HQL обеспечивает большинство возможностей, определенных стандартом ANSI SQL, и даже больше, поскольку он полностью объектно-ориентирован и поддерживает такие концепции ОО, как наследование, полиморфизм и многие другие. Ниже приведены некоторые из основных директив и средств, поддерживаемых языком HQL. Некоторые примеры их применения будут приведены далее в этой главе.

- `SELECT`, `UPDATE`, `DELETE`, `INSERT`, `FROM`, `WHERE`, `GROUP BY`, `ORDER BY`.
- Объединения (внутренние, внешние).
- Подчиненные запросы.
- Итоговые функции (например, `sum` и `count`).
- Выражения и функции (математические, строковые, дата, внутренние функции и т.д.).

Кроме того, Hibernate предоставляет методы, которые позволяют использовать базовый язык SQL, в тех редких случаях, когда языка HQL недостаточно (см. главу 10, "Кроме основ").

Простые примеры применения языка HQL будут приведены в этой главе.

### Уникальный идентификатор объекта (<id>)

Hibernate требует, чтобы связываемые классы идентифицировали первичный ключ таблицы при помощи элемента `<id>`. Например, приведенный ниже отрывок кода из

нашего файла Department.hbm.xml демонстрирует, как литерал departmentCode определяется в качестве первичного ключа (для связи с таблицей Department):

```
<id name="departmentCode" column="departmentCode">
    <generator class="assigned"/>
</id>
```

Обратите внимание на класс generator типа "assigned" в данном отрывке кода; это значит, что приложение будет предоставлять значение для этого свойства-идентификатора при любых операциях базы данных с этим объектом.

Hibernate предоставляет несколько методов создания уникальных идентификаторов для вставляемых записей, включая increment, identity, sequence, hilo, seqhilo, uuid, guid, native, assigned, select и foreign. Документация Hibernate предоставляет вполне достаточное описание каждого из них. Для наших примеров мы будем использовать генераторы assigned и identity.

### Принудительные транзакции Hibernate

Согласно документации Hibernate, посвященной работе с этим API, "транзакции никогда не бывают необязательны, все взаимодействия с базой данных должны осуществляться внутри транзакции, независимо от того, читаете вы данные или пишете". Таким образом, вызовы приведенных ниже типов вы найдете в любом коде, связанном с Hibernate:

- session.beginTransaction()
- session.getTransaction().commit()
- session.getTransaction().rollback()

### Файл HibernateUtil.java

Для установки объекта SessionFactory и обеспечения доступа к нему (при помощи метода доступа) справочная документация Hibernate рекомендует использовать вспомогательный класс (по имени HibernateUtil, например).

Пример файла класса HibernateUtil.java находится в каталоге timex\src\java\com\visualpatterns\timex\test. Фактически, этот вспомогательный класс содержит лишь несколько строк кода. Первые строки кода создают объект SessionFactory:

```
SessionFactory = new Configuration().configure()
                .buildSessionFactory();
```

Еще один интересный код в данном классе — это соответствующий метод получения значения, возвращающий объект SessionFactory, как показано далее:

```
public static SessionFactory getSessionFactory()
{
    return SessionFactory;
}
```

Затем мы можем получить объект Session, как демонстрирует следующий фрагмент кода, который выбирает список из объекта базы данных Department:

```
List departmentList=null;
Session session =
↳HibernateUtil.getSessionFactory().getCurrentSession();
```

```
session.beginTransaction();
departmentList = session.createQuery("from Department ORDER BY
↪name").list();
session.getTransaction().commit();
```

### Дальнейшее изучение

Здесь мы тоже рассмотрели лишь высокоуровневые концепции Hibernate. Более подробная информация об обсуждаемых здесь концепциях содержится в документации Hibernate (ее можно найти на Web-сайте Hibernate); однако поскольку мы будем использовать большинство этих концепций в наших примерах, я предоставлю дополнительные объяснения и примеры кода.

Теперь, рассмотрев основы, приступим к реальной работе — реализуем полнофункциональный пример, использующий некоторые из множества средств Hibernate.

## Создание файла `TimesheetManager.java` с использованием Hibernate

Обдумав бизнес-требования к приложению Time Expression, причем совместно с прототипами экранов, созданных в главе 2, "Простое приложение: сетевая система учета рабочего времени", и моделью базы данных, представленной на рис. 5.2, можно прийти к выводу, что таблица `Timesheet` — это основа нашего приложения (Time Expression). Поэтому я решил использовать данную таблицу в качестве примера этой главы. Дело в том, что, согласно нашим требованиям, для этой таблицы базы данных должны осуществляться операции чтения и записи. Например, большинство экранов приложения Time Expression либо модифицирует данные этой таблицы, либо извлекает данные из нее. Исходя из структуры класс/пакет, разработанной в главе 3, "Архитектура и модель проекта на базе XP и AMDD", и необходимых файлов конфигурации Hibernate, для нашего примера потребуются следующие файлы:

- `test/TimesheetManagerTest.java`
- `model/TimesheetManager.java`
- `model/Timesheet.java`
- `model/Timesheet.hbm.xml`

В следующих разделах каждый из этих файлов рассматривается более подробно.

### Файл `TimesheetManagerTest.java`

Давайте начнем создавать небольшой проверочный класс JUnit. Не забывайте писать предварительные проверки всякий раз, когда это возможно. В главе 4, "Установка среды: JDK, Ant и JUnit", мы уже рассмотрели причины и преимущества применения *предварительной проверки*, поэтому я не буду повторять здесь ту же информацию.

Единственный класс, необходимый для проверки модуля, — `TimesheetManager`, поскольку `Timesheet.java` — это JavaBean, а следовательно, он не имеет никакой *реальной* логики в своих методах (только методы чтения и записи).

Если мы проанализируем следующие два экрана из главы, 2, "Простое приложение: сетевая система учета рабочего времени", то сможем придумать набор функциональных возможностей, необходимых нашему классу `TimesheetManager`:

- Timesheet List. Список записей таблицы `Timesheet` для конкретного идентификатора сотрудника `employeeId`.
- Enter Hours. Возможность добавить и модифицировать одну запись таблицы `Timesheet`.

Давайте рассмотрим пример необходимых функциональных возможностей и то, как мы могли бы реализовать их. Известно, что для экрана Timesheet List (представленного на рис. 5.7) необходим список записей таблицы `Timesheet`; этот список будет выбран из базы данных с использованием идентификатора сотрудника `employeeId` (введенного им при регистрации), причем за текущий платежный период. Следовательно, теперь я могу вообразить метод класса `TimesheetManager` с примерно такой сигнатурой: `getTimesheet(int employeeId, Date periodEndingDate)`. Таким образом, мы можем легко написать проверочный метод, скажем `testGetByEmployeeIdAndPeriod()`.

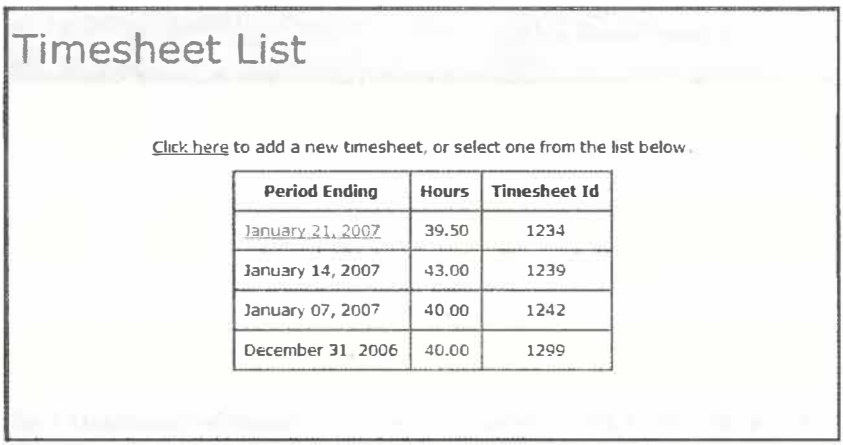



**Рис. 5.7.** Прототип экрана Timesheet List (из главы 2)

Архивный файл кода этой книги содержит законченный комплект проверки класса JUnit `TimesheetManagerTest.java`, необходимый для проверки всех методов нашего класса `TimesheetManager`.

**Примечание**

Обратите внимание на то, что я не создаю проверочный класс полностью и сразу. Как я упомянул ранее, проверка модулей и программирование осуществляются одновременно. Поэтому, написав небольшой фрагмент проверочного кода (возможно, несколько строк в отдельном методе), напишите небольшой фрагмент кода реализации, откомпилируйте и опробуйте его, а затем повторяйте эти шаги до тех пор, пока метод не будет полностью реализован. Идея заключается в том, чтобы писать маленькие методы, которые можно будет относительно легко проверить. Эта методика позволяет нам также писать минимально необходимый объем кода, достаточный для удовлетворения требований наших пользователей (не более и не менее).

Давайте изучим часть проверочного кода для этого класса; мы не будем рассматривать этот файл полностью, поскольку нам пока нужна только выборка и сохранение

отдельных объектов Timesheet, так что давайте исследуем методы, связанные с этими операциями.

testGetByEmployeeId()

Начнем с метода testGetByEmployeeId(). Несколько первых строк этого кода гарантируют, что предварительно мы получим список java.util.List объектов Timesheet:

```
List timesheetList = timesheetManager.getTimesheets();
assertNotNull(timesheetList);
assertTrue(timesheetList.size() > 0);
```

Удостоверившись, что мы имеем, по крайней мере, один объект Timesheet, можем выбрать соответствующую запись, используя идентификатор employeeId, находящийся в первом объекте Timesheet, как показано далее:

```
int employeeId=((Timesheet)timesheetList.get(0)).getEmployeeId();
timesheetList = timesheetManager.getTimesheets(employeeId);
assertNotNull(timesheetList);
```

Теперь можем просто проверить каждый объект Timesheet в списке и удостовериться, что эти записи имеют требуемый идентификатор employeeId, как продемонстрировано ниже:

```
Timesheet timesheet;
for (int i=0; i < timesheetList.size(); i++)
{
    timesheet = (Timesheet)timesheetList.get(i);
    assertEquals(employeeId, timesheet.getEmployeeId());
    System.out.println(">>>> Department name = "
                + timesheet.getDepartment().getName());
}
```

### Метод testSaveSingle()

Давайте рассмотрим в проверочном комплекте TimesheetManagerTest.java, еще одну проверку, метод testSaveSingle. Первая половина этого метода устанавливает подлежащий сохранению объект Timesheet; однако следующие строки кода заслуживают внимания:

```
timesheetManager.saveTimesheet(timesheet);
Timesheet timesheet2 = timesheetManager.getTimesheet(EMPLOYEE_ID,
    periodEndingDate);
assertEquals(timesheet2.getEmployeeId(), timesheet.getEmployeeId());
assertEquals(timesheet2.getStatusCode(), "P");
```

По существу, мы сохраняем объект Timesheet, выбираем его из базы данных снова, а затем сравниваем атрибуты этих двух объектов при помощи метода assertEquals.

### Файл TimesheetManager.java

Затем рассмотрим наш, так сказать, бутербродный класс (bread-and-butter class). Этот класс будем использовать в главе 7, "Среда Spring Web MVC Framework", когда будем реализовывать наши пользовательские интерфейсы (например, Timesheet List и Enter Hours).

Ключевые методы, которые мы рассмотрим здесь,— это getTimesheets, getTimesheet и saveTimesheet.

Начнем с метода TimesheetManager.getTimesheets(int employeeId). Ключевые строки кода, по существу, получают список java.util.List объектов Timesheet из базы данных, используя метод Hibernate Session.createQuery, как показано далее:

```
timesheetList = session.createQuery(
    "from Timesheet" + " where employeeId = ?").setInteger(0,
        employeeId).list();
```

Следующий метод, TimesheetManager.getTimesheet(int employeeId, Date periodEndingDate), немного отличается от только что рассмотренного метода getTimesheets(int employeeId); основное различие заключается в использовании метода Hibernate uniqueResult, который очень удобен для возвращения из запроса только одного объекта. Приведенный ниже код демонстрирует соответствующие строки нашего метода getTimesheet:

```
timesheet = (Timesheet) session.createQuery(
    "from Timesheet" + " where employeeId = ?"
    + " and periodEndingDate = ?").setInteger(0,
    employeeId).setDate(1, periodEndingDate).uniqueResult();
```

И последний метод класса TimesheetManager, который мы рассмотрим здесь,— это saveTimesheet(Timesheet timesheet). Данный метод достаточно прост, а единственный, заслуживающий внимания код— это метод Hibernate session.saveOrUpdate, который, в зависимости от того, существует ли запись в базе данных, осуществляет операцию INSERT или UPDATE:

```
session.saveOrUpdate(timesheet)
```

### Файл Timesheet.java (и Timesheet.hbm.xml)

Прежде чем мы сможем успешно откомпилировать и применить файл TimesheetManager.java, необходимо создать файлы, с которыми он связан, а именно: файл Timesheet.java и файл связывания Timesheet.hbm.xml (оба доступны среди файлов кода этой книги). Эти файлы невелики; код Java— простой JavaBean, а файл XML просто связывает свойства Bean с соответствующим столбцом базы данных.

## Файлы Employee.* и DepartmentManager.java

К другим файлам, предоставленным в этой книге, относятся приведенные ниже. Они будут нужны для реализации наших первых пяти пользовательских историй (см. стр. 55).

Сейчас мы создадим эти файлы в нашем каталоге src/java/com/visualpatterns/timex/model:

- DepartmentManager.java
- Employee.hbm.xml
- Employee.java
- EmployeeManager.java

## Файлы, подлежащие указанию в CLASSPATH

Различные файлы Hibernate, такие как `hibernate.cfg.xml`, и файлы связывания, например `Department.hbm.xml`, должны быть указаны в переменной `CLASSPATH`; в результате наш сценарий Ant, `build.xml`, автоматически скопирует эти файлы в каталог `timex/build/timex/WEB-INF/classe` при построении.

## Запуск проверочного комплекта при помощи Ant

Теперь можем запустить наш описанный ранее проверочный комплект (`TimesheetManagerTest`). Но прежде чем мы сможем запустить проверочный комплект, необходимо запустить базу данных HSQLDB в режиме сервера. Мы можем сделать это вручную, как показано далее:

```
java -cp /hsqldb/lib/hsqldb.jar org.hsqldb.Server
↳-database.0 data\time xdb -dbname.0 timex -port 9005
```

> **Примечание**
> Я подразумеваю, что база HSQLDB установлена в соответствующий корневой каталог (т.е. `/hsqldb`); если у вас она установлена иначе, измените команду `java` так, чтобы она соответствовала вашей системе.

Либо можем запустить сервер, используя наш вспомогательный сценарий Ant следующим образом:

```
ant -f timexhsqldb.xml starthsql
```

После успешного запуска сервера HSQLDB, при наличии всех наших файлов в соответствующих каталогах, мы сможем проверить наши новые классы, вводя в командной строке команду **`ant rebuild test`** (из каталога верхнего уровня `timex/`). Результат выполнения этой команды представлен на рис. 5.8.

## Удаление записей

Мы рассмотрели чтение и запись в базе данных. Однако осталось без внимания удаление записи — неизбежная потребность в любом приложении CRUD. Удаление записи не было указано среди задач приложения Time Expression. Не описанными ранее элементами здесь являются метод `Session.delete`, который удаляет запись из базы данных (объект), и метод `Session.load`, который выбирает запись из базы данных, как показано ниже:

```
session.delete(session.load(Timesheet.class, new
↳Integer(timesheetId)));
```

В качестве альтернативы, код удаления можно написать с использованием метода `Query.executeUpdate()`, полезного при обработке нескольких записей, как показано далее:

```
int updated = session.createQuery("DELETE from Timesheet"
                                  + " where timesheetId = ?")
                     .setInteger(0, timesheetId)
                     .executeUpdate();
```


**Рис. 5.8.** Запуск проверочного комплекта JUnit при помощи Ant

## Запросы и критерии

До сих пор, для выборки записи из базы данных, мы использовали интерфейс Hibernate `Query` (при помощи метода `Session.createQuery`). Но существует немного более динамичный, а возможно, и более простой способ выборки записей — интерфейс Hibernate `Criteria`. Это обеспечит более объектно-ориентированный подход, который может снизить количество ошибок, поскольку он способен осуществлять проверку типов, а также позволяет избежать возможных синтаксических ошибок и исключений, присущих HQL. Этот метод проще, поскольку разработчик использует более объектно-ориентированный подход, т.е., скорее, объекты, а не простые текстовые запросы. Таким образом, разработчик имеет больше контроля над соответствием типов данных, следовательно, уменьшается количество ошибок, особенно таких, как `QueryExceptions`. Но это же может стать и проблемой, если синтаксис запроса слишком сложен.

Интерфейс `Criteria` доступен при помощи метода `Session.createCriteria`, как показано в следующем отрывке кода:

```
timesheetList = session.createCriteria(Timesheet.class)
                       .add(Restrictions.eq("employeeId", employeeId))
                       .list();
```

Кроме того, Hibernate предоставляет несколько классов в пакете `org.hibernate.criterion`, которые, работая с интерфейсом `Criteria`, предоставляют надежные объекты, реализующие функциональные возможности запроса. К этим классам относятся: `Restrictions`, `Order`, `Junction`, `Distinct` и некоторые другие.

### Обработка исключений

Большинство связанных с базой данных *исключений* (exception), передаваемых при использовании API Hibernate, определено внутри класса `org.hibernate.HibernateException`; более подробная информация по этой теме приведена в справочной документации Hibernate. Тем не менее для обработки исключений базы данных справочная документация Hibernate рекомендует следующую стратегию:

"Если объект `Session` передает исключение (включая любое исключение SQLException), вы должны немедленно осуществить откат транзакции базы данных, вызвать метод `Session.close()` и удалить экземпляр объект `Session`. Некоторые методы объекта `Session` способны перевести сеанс в некорректное состояние. Ни одно из исключений, переданных Hibernate, не может рассматриваться как восстанавливаемое. Чтобы гарантировать закрытие сеанса, ближе к концу блока расположите вызов метода `close()`".

Мы, безусловно, будем следовать этому совету. Однако, как вы могли заметить в нашем коде, мы повторно передаем все перехваченные исключения. Вообще-то, это хорошая идея передавать исключения в стек вызовов, чтобы методы верхнего уровня смогли определить, как их обработать. Более подробно обработку исключений обсудим в главе 10, "Кроме основ".

Я полагаю, что проблема этого фрагмента кода может заключаться в том, что разработчик никогда не знает, какое исключение будет передано фактически. Поэтому все, что мы делаем в блоке обработчика, — это откат транзакции. Чтобы гарантировать правильное обнаружение и обработку исключения, необходим некоторый механизм регистрации или повторной передачи исключения внешней вызывающей стороне (в противном случае мы никогда не узнаем подробностей отказа, за исключением того, что табель учета рабочего времени не был сохранен).

## Другие возможности Hibernate

До сих пор мы рассматривали базовые возможности Hibernate. Теперь обратим внимание на более передовые концепции.

### Ассоциации

Физический проект базы данных, связывание и классы Java для приложения Time Expression — это все довольно просто. Чтобы проект остался простым и эффективным, мы, по существу, используем стратегию связывания "один класс на одну таблицу". Однако при получении имени отдела мы можем использовать связь типа "многие к одному", чтобы выбрать соответствующие записи таблицы `Department`. Эти функциональные возможности понадобятся нам на тех экранах приложения Time Expression, которые отображают полное имя отдела (`Department.name`), а не только его код (`Department.departmentCode`).

Давайте поэтапно рассмотрим код Java и соответствующий код связывания XML.

Во-первых, код постоянного атрибута JavaBean находится в файле `Department.java`; его отрывок приведен ниже:

```
private Department department;

public Department getDepartment()
{
    return department;
}
public void setDepartment(Department department)
{
    this.department = department;
}
```

Во-вторых, связывание по принципу "многие к одному" осуществляется в нашем файле `Timesheet.hbm.xml`:

```
<many-to-one name="department" column="departmentCode"
    class="com.visualpatterns.timex.model.Department"
    lazy="false" not-found="ignore" cascade="none"
    insert="false" update="false"/>
```

И наконец, код получения названия отдела находится в нашем файле `TimesheetManagerTest.java`:

```
System.out.println(">>>> Department name = " +
timesheet.getDepartment().getName());
```

## Блокировка объектов (параллельное управление)

*Блокировка* (locking) базы данных может осуществляться любым приложением базы данных, а не только приложением на основании технологии ORM. Существует две общепринятые стратегии, когда дело касается обновления записей базы данных: пессимистическая блокировка и оптимистическая.

Оптимистическая блокировка обеспечивает большую масштабируемость, чем пессимистическая. Однако пессимистическая блокировка лучше подходит для ситуаций, когда существует высокая вероятность одновременных модификаций тех же данных несколькими источниками (например, пользователями), следовательно, есть возможность "порчи" данных. Давайте рассмотрим краткое описание каждой из этих двух стратегий блокировки.

*Пессимистическая блокировка* (pessimistic locking) осуществляется тогда, когда запись следует зарезервировать для монопольного использования. При этом может блокироваться как отдельная запись базы данных, так и вся таблица. Hibernate обеспечивает пессимистическую блокировку (при использовании основной базы данных, а не записи в оперативной памяти) при помощи одного из следующих методов:

- `Session.get`
- `Session.load`
- `Session.lock`
- `Session.refresh`
- `Query.setLockMode`

Хотя каждый из методов получает разные параметры, для всех них есть один общий параметр — класс LockMode, который обеспечивает различные режимы блокировки, такие как NONE, READ, UPGRADE, UPGRADE_NOWAIT и WRITE. Например, чтобы получить запись Timesheet для модификации, мы могли использовать следующий код (подразумевается, что основная база данных поддерживает блокировку):

```
public Timesheet getTimesheetWithLock(int timesheetId)
{
    Session session =
        HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    Timesheet timesheet = (Timesheet)session.get(Timesheet.class,
        new Integer(timesheetId), LockMode.UPGRADE);
    session.getTransaction().commit();
    session.close();0

   return timesheet;
}
```

*Оптимистическая блокировка* (optimistic locking) означает, что вы будете не блокировать данную запись или таблицу базы данных, а проверять некоторым способом свойство или столбец (например, столбец timestamp), чтобы удостовериться в том, что данные не изменились с момента их чтения. Hibernate обеспечивает это за счет использования свойства version, которое может быть проверено либо самим приложением, либо автоматически Hibernate. Например, следующий фрагмент кода, взятый из справочной документации Hibernate, демонстрирует, как приложение может самостоятельно сравнить переменную oldVersion с текущей версией при помощи метода получения значения (например, getVersion):

```
// foo - экземпляр, загруженный на предыдущем сеансе
session = factory.openSession();
int oldVersion = foo.getVersion();
session.load( foo, foo.getKey() );
if ( oldVersion!=foo.getVersion ) throw new
StaleObjectStateException();
foo.setProperty("bar");
session.flush();
session.connection().commit();
session.close();
```

Исключение StaleObjectStateException, представленное в предыдущем примере, принадлежит пакету org.hibernate.

## И еще о Hibernate

Хотя в этой главе мы рассмотрели много материала, о Hibernate осталось сказать еще очень много. Однако, как я упомянул ранее, Hibernate посвящены отдельные книги, и мы не можем рассмотреть в одной главе эту технологию полностью. Однако я привел здесь достаточно сведений для создания приемлемо сложных приложений, использующих Java, Hibernate и реляционные базы данных.

К другим дополнительным возможностям Hibernate, не описанным здесь, но способным пригодиться, относятся следующие.

- Расширенное связывание (например, двунаправленные связи, троичные связи, сортируемые коллекции, компонентное связывание, наследование связей и т.д.)
- Расширенный HQL
- Аннотации (и XDoclet)
- Фильтры
- Утилита Hibernate SchemaExport
- Наследование связей
- Перехватчики
- Объекты блокировки
- Стратегии повышения производительности (например, стратегия выборки, кэш второго уровня)
- Прокручиваемая итерация и разбиение на страницы
- Управление транзакциями (расширенные возможности)
- Другие области, такие как использование хранимых процедур, базового SQL и т.д.

# Резюме

В этой главе мы разрабатывали классы, необходимые для реализации функциональных возможностей пяти первых пользовательских историй (см. стр. 55). Кроме того, эти классы будут использованы в коде, который мы разработаем в главе 7, "Среда Spring Web MVC Framework", и при планировании работ в главе 10, "Кроме основ".

В этой главе мы сделали следующее.

- Рассмотрели, что такое технология объектно-реляционного связывания и каковы ее преимущества
- Установили базу HSQLDB
- Разработали нашу база данных
- Использовали сценарий DDL для создания таблицы базы данных и некоторых проверок данных
- Установили среду Hibernate, рассмотрели ее фундаментальные концепции и начали работу с ней
- Разработали простой, а затем немного более сложный пример (наряду с соответствующим классом проверочного комплекта) использования Hibernate для приложения Time Expression, а также таблиц `Department` и `Timesheet`
- Обсудили дополнительные темы по Hibernate, а также средства для их самостоятельного изучения

Но мы еще отнюдь не закончили с Hibernate! Например, мы используем некоторые из классов, созданных в этой главе для нашего Web-приложения, в следующей главе. Кроме того, я продемонстрирую, как мы сможем использовать дополнение Eclipse для создания файлов связывания Hibernate (глава 8, "Феномен Eclipse!").

Теперь мы готовы углубиться в следующие две главы, где познакомимся с миром пользовательских интерфейсов, использования Web-среды выполнения MVC Spring для разработки нашего Web UI.

В главе 8 мы начнем работу с SDK Eclipse, а также убедимся, сколько времени может сэкономить IDE. До сих пор я преднамеренно использовал командную строку, поскольку уверен, что изучение основных принципов с использованием ручного способа ввода поможет вам лучше понять, как все это работает. Приобретенные навыки весьма пригодятся, если вам придется вернуться к командной строке в случае отсутствия у IDE некоторых функциональных возможностей или наличия неких ошибок.

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, содержится на следующих Web-сайтах:

- Agile Data `http://www.agiledata.org`
- Agile Modeling `http://www.agilemodeling.com`
- Apache ObJectRelationalBridge (OJB) `http://db.apache.org/ojb/`
- Введение в параллельное управление `http://www.agiledata.org/essays/concurrencyControl.html`
- Cocobase `http://www.thoughtinc.com/`
- Рефакторинг баз данных `http://www.agiledata.org/essays/databaseRefactoring.html`
- Domain-Driven Design `http://domaindrivendesign.org/`
- Форум Hibernate `http://forum.hibernate.org/`
- Hibernate `http://hibernate.org/`
- HSQLDB `http://hsqldb.org/`
- iBATIS Java `http://ibatis.apache.org`
- JDO и EJB 3.0 `http://java.sun.com`
- JORM `http://jorm.objectweb.org/`
- Группа Object Data Management Group `http://www.odmg.org/`
- Пример ORM `http://www.simpleorm.org/`
- Проект Castor `http://www.castor.org/`
- Cocobase `http://www.thoughtinc.com/`

# 6

# Обзор среды Spring Framework

Выпуск 1, Неделя 5, Итерация 2

**Стив:** Сьюзен, мы работаем над итерацией 2 и имеем пару вопросов по поводу этих экранов. Например, сколько десятичных знаков мы должны отображать в этом поле?

**Сьюзен:** Давайте остановимся на двух десятичных знаках. Мне также хотелось бы переместить верхний раздел в середину, а этот раскрывающийся список содержит неправильные значения. Но в целом все выглядит хорошо. Меня радует прогресс, который я наблюдаю в разработке данного приложения каждые две недели. Это внушает уверенность в том, что мы не только реализуем необходимые возможности, но и устраним вероятные проблемы.

Когда несколько лет назад корпорация Sun Microsystems представила спецификацию Enterprise JavaBean (EJB) 1.0, я был так возбужден, что даже написал относительно длинную статью для журнала JavaWorld.com (`http://www.javaworld.com/javaworld/jw-04-1999/jw-04-middleware.html`), по существу, подводя итог этой спецификации в одной статье. Статья получилась хорошо, и я гордился своей работой.

Годом позже я написал, также для журнала JavaWorld.com, более короткую статью "Do You Really Need Enterprise JavaBeans?" (Действительно ли вам нужна Enterprise JavaBeans?) (`http://www.javaworld.com/javaworld/jw-10-2000/jw-1006-soapbox.html`).

Предпосылками написания этой статьи были многие проекты, в которых я видел неправильное использование EJB. Например, я видел нераспределенное использование этой распределенной технологии. Я даже видел простые Web-приложения, использующие EJB, когда они могли бы легко использовать *простые старые объекты Java* (Plain Old Java Objects — POJO) в хорошо проработанном приложении с простым разделением уровней (представление и бизнес-уровень, например).

Кроме того, та роль, которую я играл в своей компании, обязывала меня общаться с многими разработчиками и быть в курсе дел ведущих компаний, проектов и разработок. На протяжении этого трех- или четырехлетнего периода я выслушал также личные мнения, вероятно, сотен разработчиков относительно EJB (и других компонентов JEE). Наиболее популярная тема, которую затрагивает большинство собеседников, — это использование компонентов сеанса только без фиксации состояния, иногда речь идет об использовании смешанных объектных компонентов и компонентов, управляемых сообщениями; но я не могу припомнить никого, кто осуждал бы использование компонентов сеанса с фиксацией состояния.

Так в чем же суть этой истории? Она очень проста. По моему мнению, компания Sun определила стандарт, который, вероятно, более применим для сложных приложений и менее применим для приложений, которым не требуется большинство средств, предоставляемых JEE. Кроме того, EJB используется в проектах, где, вероятно, понадобится контейнер EJB, который впоследствии может перемещаться на дорогой сервер приложений, если так решит ваша организация, или возможен переход на коммерческий продукт вместо продукта с открытым исходным кодом, таким как сервер приложений JBoss Application Server (`jboss.com`) или Apache Geronimo (`geronimo.apache.org`).

Когда не так давно я познакомился со средой Spring Framework, меня поразило то, что я мог работать с объектами POJO и одновременно иметь доступ к большинству таких возможностей, как JEE/EJB. Например, среда Spring обеспечивает защищенное декларативное управление транзакциями.

Я полагаю вполне справедливым посвятить среде Spring Framework две полные главы и несколько глав частично, поскольку эта тема достаточно важна. Однако, как вы вскоре увидите, среда Spring Framework не является бескомпромиссным подходом, т.е. вы можете сами решать, какие из модулей применимы именно для вас. В этой книге мы используем две основные возможности среды Spring: ее среда выполнения Web и, безусловно, службы *инверсии управления* (Inversion of Control — IoC), называемые также службами *внедрения зависимости* (dependency injection). Кроме того, мы используем среду Spring для планирования работ и передачи электронной почты. Мы не бу-

дем использовать Spring AOP непосредственно, а только косвенно, при использовании среды Spring Web MVC Framework.

**Примечание**
С этого момента я буду называть среду Spring Framework просто Spring. Кроме того, я буду использовать термины *внедрение зависимости* (dependency injection) и *IoC* как синонимы.

# Что рассматривается в этой главе

Основное внимание в этой главе уделяется высокоуровневым концепциям среды Spring Framework и некоторым другим концепциям, связанным с ней (например, внедрению зависимости). Среду Spring Web MVC Framework мы рассмотрим в следующей главе, а другие возможности, типа планирования задач и передачи электронной почты, опишем в последующих главах. В этой главе мы сделаем следующее.

- Добьемся полного понимания, что представляет собой среда Spring Framework, каковы ее фундаментальные концепции, как она организована и каковы преимущества ее использования.
- Изучим основные концепции Spring, такие как внедрение зависимости, компоненты и фабрики компонентов, контекст приложения, редакторы свойства и многое другое.
- Рассмотрим, с точки зрения развертывания и разработки, как упакована среда Spring Framework.

# Что такое Spring

Когда меня спрашивают, что такое Spring, я задумываюсь о том, как ответить на этот вопрос одним предложением, но сделать это не просто, и вы вскоре увидите, почему. Поэтому позвольте мне сначала привести отрывок из справочной документации Spring Framework Reference Documentation (`springframework.org`), поскольку он хорошо описывает эту среду выполнения:

"... Spring предоставляет облегченное решение по созданию готовых корпоративных приложений, при сохранении возможности декларативного управления транзакциями, дистанционного доступа к вашей логике, использования RMI или Web-служб, средств отправки по почте и различных возможностей обработки данных в базе данных. Spring предоставляет среду выполнения MVC, сквозные способы интегрирования AOP в ваше программное обеспечение и хорошо структурированную иерархию исключений, включающую автоматическое связывание с собственной иерархий исключений.

Потенциально, Spring может стать универсальным пунктом для всех ваших корпоративных приложений, однако Spring является модульной средой, позволяя вам использовать свои части без необходимости вводить остальные ..."

Давайте рассмотрим аспект "модульности" Spring немного подробнее. На рис. 6.1 (также взятом непосредственно из справочной документации Spring Framework Reference Documentation), вероятно, наилучшим образом демонстрируются различные возможности Spring, а также наглядно представлено, почему так трудно описать всю среду Spring Framework в одном предложении.

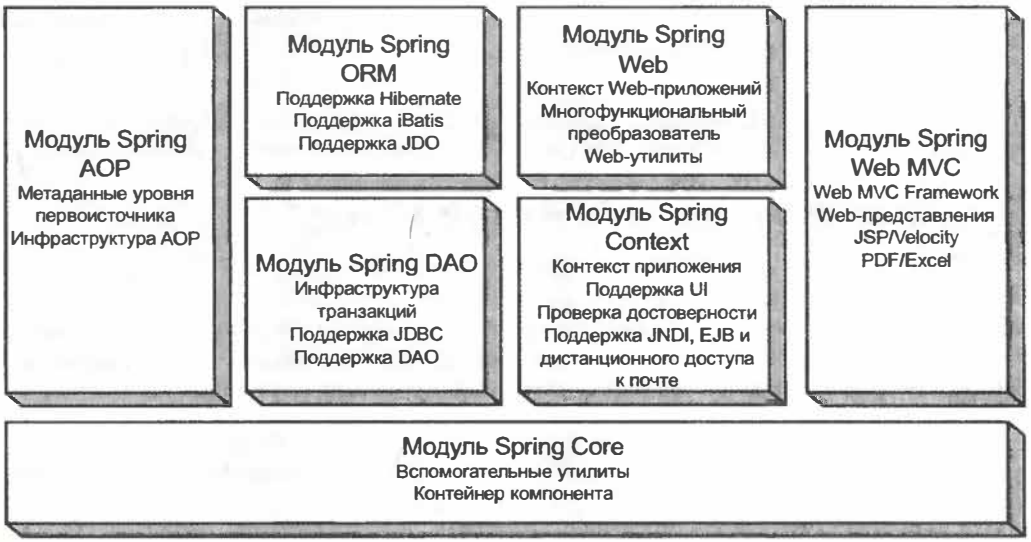



**Рис. 6.1.** Структура среды Spring Framework (оригинал взят непосредственно из справочной документации Spring Framework Reference Documentation, расположенной по адресу `springframework.org`)

В этой книге я использую Spring Framework версии 2.0 RC1.

# Упаковка Spring для разработки

*Модули* Spring, представленные на рис. 6.1, являются, по существу, концептуальными группами функциональных возможностей, предоставляемых обширным списком пакетов Java Spring и основных классов.

На рис. 6.2 представлены различные высокоуровневые пакеты Spring. (*Примечание:* серым цветом помечены те пакеты, которые мы используем для нашего типового приложения, что еще раз подчеркивает отнюдь не бескомпромиссный подход Spring.)

> **Удивлены размером Spring Framework?**
> Среда Spring Framework содержит порядка 130 серий пакетов Java и более 1 200 классов Java. Однако давайте не впадать в панику по поводу такого количества; тому есть ряд причин.
> Во-первых, вы можете использовать только часть возможностей, например IoC Spring, игнорируя, по существу, большинство доступных API. Во-вторых, хотя имеется большое количество классов и документов Java, большинство из них предназначено для внутреннего использования самой средой выполнения. В-третьих, Spring позволяет выбирать, какие модули использовать, а какие игнорировать. Например, для нашего типового приложения, Time Expression, мы используем лишь незначительное их количество; это еще одно доказательство высокой модульности среды Spring.

# Упаковка Spring для развертывания

Модульный аспект Spring распространяется также и на развертывание, поскольку вы можете развертывать ваше приложение лишь с некоторыми необходимыми фай-

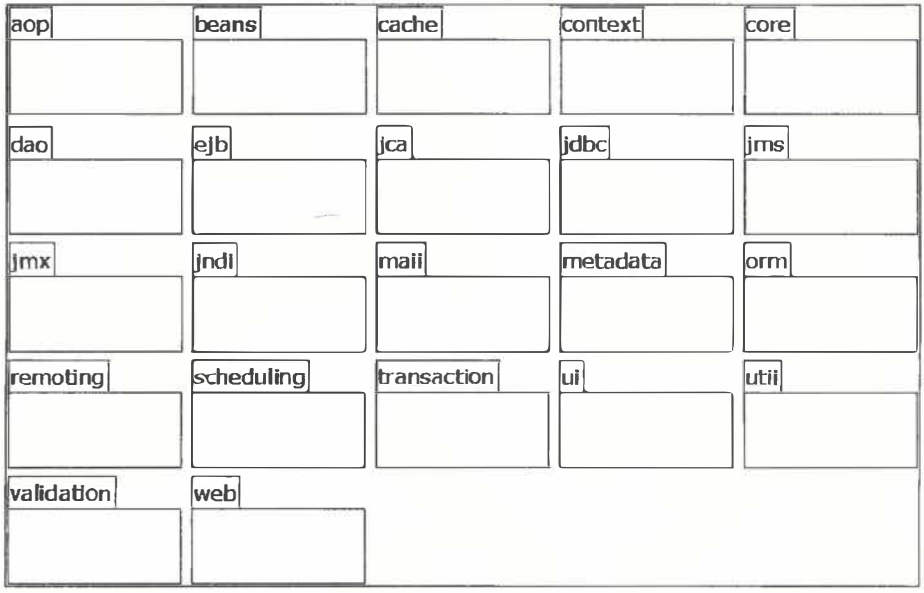



**Рис. 6.2.** Высокоуровневые пакеты Java Spring от `org.springframework`

лами JAR среды Spring (например, `spring-jdbc.jar`). Табл. 6.1 демонстрирует, как среда Spring обеспечивает пакеты, используя различные файлы JAR. Обратите внимание на то, что файл `spring.jar` существенно больше других; дело в том, что это полный пакет. Я предпочитаю включать в свои приложения полный файл `spring.jar`, поскольку размер в 1,7 Мбайт не считается ныне очень большим, особенно для серверных приложений.

**Таблица 6.1. Полный и отдельные файлы JAR среды Spring**

| Файл JAR | Размер (Кбайт) |
|---|---|
| `spring.jar` | 1 731 |
| `modules/spring-beans.jar` | 258 |
| `modules/spring-aop.jar` | 224 |
| `modules/spring-jdbc.jar` | 210 |
| `modules/spring-webmvc.jar` | 208 |
| `modules/spring-support.jar` | 189 |
| `modules/spring-remoting.jar` | 166 |
| `modules/spring-web.jar` | 138 |
| `modules/spring-core.jar` | 129 |
| `modules/spring-context.jar` | 111 |
| `modules/spring-dao.jar` | 103 |
| `extmodules/spring-hibernate3.jar` | 102 |

*Окончание табл. 6.1*

| Файл JAR | Размер (Кбайт) |
|---|---|
| extmodules/spring-portlet.jar | 97 |
| extmodules/spring-hibernate2.jar | 85 |
| extmodules/spring-mock.jar | 69 |
| extmodules/spring-jdo.jar | 61 |
| extmodules/spring-toplink.jar | 55 |
| extmodules/spring-ojb.jar | 27 |
| aspects/spring-aspects.jar | 10 |

# Обзор модулей Spring

Теперь, поверхностно изучив среду Spring в целом, давайте коротко рассмотрим описание каждого модуля. Начнем с двух наиболее важных модулей, а затем рассмотрим остальные в алфавитном порядке. Здесь мы затронем лишь некоторые из модулей, в очередной раз подчеркнув, насколько обширна эта среда выполнения.

## Модуль Spring Core

Базовый модуль — это, по существу, ядро среды Spring. Оно предоставляет такие фундаментальные средства, как внедрение зависимости (рассматривается далее в этой главе) и управление компонентами. К высокоуровневым пакетам, входящим в состав этого модуля, относятся `org.springframework.beans`, `org.springframework.core` и `org.springframework.util`.

## Модуль Spring Context

Модуль контекста — вероятно, второй по важности модуль Spring, после модуля Spring Core. Он содержит такие ключевые классы, как `ApplicationContext` (рассматривается далее в этой главе) и `WebApplicationContext`, которые мы используем для экрана Time Expression (загружаются из нашего файла `timex-servlet.xml`, рассматриваемого в следующей главе). Кроме того, мы используем пакеты `org.springframework.mail` (для передачи электронной почты) и `org.springframework.validation` (для проверки достоверности полей Web UI), которые также считаются частью этого модуля.

Другие пакеты, прежде всего, используемые для дистанционных/распределенных функций (EJB и JNDI, например), выходят за пределы этой книги.

## Модуль Spring AOP

Мы не будем писать код AOP, используя среду Spring непосредственно. Но косвенно мы применим AOP-ориентированные средства Spring, такие как *перехватчики* (interceptor), для нашего Web-приложения и декларативного управления транзакциями. Фактически, справочная документация среды Spring разделяет ту же точку зрения, т.е. если встроенных функциональных возможностей, предоставляемых средой Spring, достаточно для ваших потребностей, вам не стоит применять Spring AOP (например, для реализации специальных аспектов).

**Личное мнение: проект не годится для AOP**

Некоторым образом аспект-ориентированное программирование (Aspect-Oriented Programming — AOP) — это целый новый мир, даже несмотря на то, что оно дополняет OO. Подобно многим другим креативным нововведениям, концепцию AOP выдвинула компания Xerox PARC. Джордж Кикзалес (Gregor Kiczales), автор концепции AOP, вместе со своей группой разработал также первый язык AOP, AspectJ, который, вероятно, все еще остается наиболее популярным языком среды выполнения AOP или, по крайней мере, наиболее богатым возможностями и комплектами инструментов, доступными на сегодня.

По существу, AOP предоставляет более простой способ разбиения приложения на отдельные модули (называемый также *разделением интересов* (separation of concerns)). Каждый слой приложения (например, слой бизнес-объекта) сосредоточивается на своих основных функциях и не содержит функций, пересекающихся с другими слоями. Регистрация, защита, управление транзакциями и проверка — это наиболее распространенные примеры интересов, которые могут быть легко отделены от слоя бизнес-объекта, например, за счет использования перехватчиков. Как правило, в реализации языка AOP эти типы интересов (такие, как регистрация, защита, транзакции) стараются инкапсулировать за счет введения новой конструкции, называемой аспектом. *Аспект* (aspect) может изменять поведение базового кода (часть программы, не относящаяся к аспекту) за счет применения *уведомлений* (advice), дополнительных режимов, в определенных *точках привязки* (join point) (пункта структуры или исполняемого кода программы), называемых также *pointcut* (логическое описание набора точек привязки).

Хотя AOP предоставляет некоторые вполне очевидные преимущества и многообещающие пути разбиения программ на модули (с использованием разделения интересов), по моему личному мнению, большинство программных проектов не годится для AOP; следовательно, этот подход не станет господствующей тенденцией на протяжении, по крайней мере, еще нескольких лет. Кроме того, я считаю, что фундаментальной проблемой AOP является ее базовая терминология. Безусловно, привыкание к таким терминам, как *интерес* (concern), *уведомление* (advice), *точка привязки* (jointpoint, pointcut) и *аспект* (aspect), займет у разработчиков некоторое время. Понимаю, это звучит довольно глупо, но я всегда полагал, что люди должны чувствовать себя комфортно с основами новой технологии или нового подхода, перед тем как переходить на следующий, более сложный уровень. Повторюсь, это мое личное мнение, поскольку, как технология, AOP вполне готово и может прекрасно использоваться в ваших проектах, если будет применено правильно.

## Модуль Spring DAO

Поскольку для отказоустойчивой базы данных мы уже используем Hibernate, этот модуль нам не нужен. Но можно задействовать и этот модуль, например, если необходимо работать с JDBC, но нет желания возиться с блоками `try-catch-finally`, открытием/закрытием подключений и т.д. Кроме того, среда Spring идет на один шаг далее, предоставляя стройную иерархию исключений, которая способна преобразовать исключения, определенные производителем, в более понятные исключения времени выполнения, которые могут быть либо обработаны на слое вашего приложения, либо проигнорированы и обработаны в других местах (так устраняется необходимость в блоках кода `try/catch/finally`).

**Среда Spring и отказоустойчивость**

Следует уяснить, что среда Spring предоставляет все функции отказоустойчивости. Другими словами, вы могли бы использовать Spring и без таких продуктов ORM, как Hibernate, получая все преимущества механизма отказоустойчивости, причем существенно проще, чем у JDBC (хоть это и не совсем то, что мы обсуждали в главе 5, "Применение Hibernate для постоянных объектов").

Кроме того, вы получаете надежный механизм обработки исключений и все связанные с этим преимущества. Мы используем среду ORM потому, что так мы можем работать с записями базы данных, как с объектами Java.

## Модуль Spring ORM

Модуль Spring для *объектно-реляционного связывания* (Object-Relational Map – ORM) обеспечивает поддержку для интеграции с популярными продуктами ORM, используемыми разработчиками Java, такими как Hibernate, JDO, Oracle TopLink, Apache OJB и iBATIS SQL Maps.

Одними из преимуществ использования Spring ORM являются такие возможности, как легкость проверки (за счет внедрения зависимости), общие исключения данных, постоянное управление ресурсами (например, SessionFactory Hibernate), интегрированное управление транзакциями корпоративного класса и т.д.

## Модули Spring Web и Spring Web MVC

Среди всех упомянутых здесь модулей Spring, модули Web – это именно те модули, с которыми мы будем работать большую часть времени в этой книге. Однако работать мы будем только с несколькими из классов и пакетов. Например, нам не понадобятся пакеты, которые поддерживают стандарт JavaServer Faces (JSF) от Sun Microsystem и т.д.

В приложении Time Expression мы применим такие классы этих пакетов, как `SimpleUrlHandlerMapping`, `InternalResourceViewResolver`, `SimpleFormController`, `Validator` и многие другие. Эти и другие классы более подробно рассматриваются в следующей главе.

**Примечание**

Существует множество других пакетов Java, которые не упомянуты здесь отнюдь не из-за их малой значимости, просто их описание выходит за рамки этой книги.

Здесь стоит упомянуть еще пару классов, `org.springframework.scheduling.quartz.CronTriggerBean` и `org.springframework.mail.javamail.JavaMailSenderImpl`. Мы будем использовать их в главе 10, "Кроме основ", для удовлетворения требования по автоматической отправке электронной почты, описанного в главе 2, "Простое приложение: сетевая система учета рабочего времени". Кроме того, мы будем использовать классы, расположенные в пакете `org.springframework.mock.web` при проверке кода модуля с использованием ложных Web-объектов. (Более подробная информация о ложных Web-объектах приведена по адресу `mockobjects.com`.)

# Когда среда Spring вступает в нашу архитектуру

Прежде чем перейти к обсуждению различных концепций и преимуществ среды Spring, имеет смысл вспомнить нашу архитектурную схему (из главы 3, "Архитектура и модель проекта на базе XP и AMDD"), чтобы увидеть, где мы будем использовать среду Spring. Схема представлена на рис. 6.3. Обратите внимание на то, что мы будем использовать Web-контроллеры Spring (подробно обсуждаются в главе 7, "Среда Spring Web MVC Framework"), планирование задач, отправку электронной почты (глава 10, "Кроме основ") и т.д. В ядре нашего приложения будем использовать службы Spring IoC.

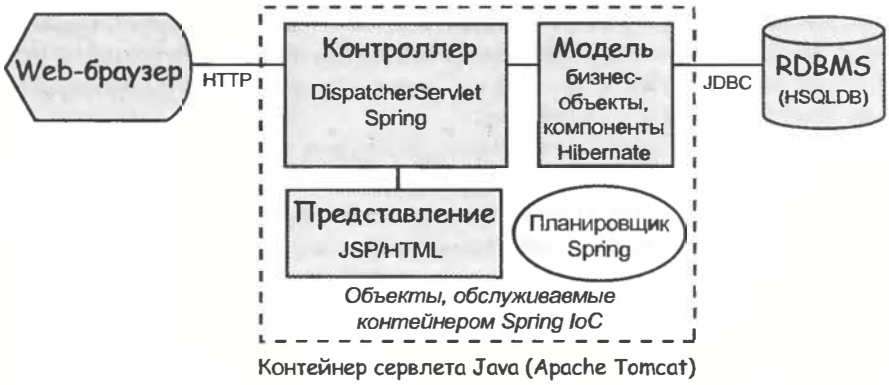



**Рис. 6.3.** Высокоуровневая архитектура приложения Time Expression

# Преимущества использования Spring

Теперь вы должны иметь довольно хорошее представление о том, что представляет собой среда Spring, как она логически организована (в виде модулей) и каковы типы предоставляемых ею функциональных возможностей. Чтобы преимущества стали более очевидны, ниже приведен их список.

- POJO. Мне больше всего нравится то, что среда Spring позволяет разрабатывать прикладные программы корпоративного класса, использующие объекты POJO. Преимущество использования только объектов POJO заключается в том, что вы не нуждаетесь в таких контейнерах EJB, как серверы приложений, если ваше приложение не требует всех возможностей, которые такие средства обеспечивают. Используя Spring, вы имеете возможность применять только защищенный контейнер сервлетов, такой как Tomcat или некий коммерческий продукт.
- Модульность. Как уже упоминалось, среда Spring организована в виде модулей. Несмотря на то что количество пакетов и классов существенно, вам следует позаботиться только о необходимых (см. раздел "Удивлены размером Spring Framework?"). Следовательно, введение Spring в существующий или новый проект может быть осуществлено на разовом или модульном основании.
- Возможность дополнения. Мне нравится тот факт, что среда Spring не изобретает колесо заново; она позволяет частично дополнить существующие возможности. Например, она дополняет среды ORM, JEE, Quartz и таймеры JDK, другие технологии представления и т.д.
- Проверка. Проверка приложений, написанных с использованием среды Spring, довольно проста, поскольку зависимый от системы код перемещается в эту среду выполнения (в отличие от поисковых таблиц JNDI, встраиваемых в код, например). Кроме того, используя объекты POJO в стиле JavaBean, становится проще применять внедрение зависимости для проверки данных (возможно, при помощи файла XML в качестве источника проверяемых данных). Кроме того, ложные классы Spring могут помочь вам смоделировать такие классы, как объекты запросов HTTP. Это особенно важно, поскольку введение зависимости работает с методами чтения и записи значений. Следовательно, достаточно

просто ввести проверяемые данные в ваши объекты, и модуль проверит ваш код, используя такой продукт, как JUnit. Сюда относится также проверка разрабатываемых Web-компонентов при помощи Web-среды MVC, как будет продемонстрировано в следующей главе.

- Одноэлементные классы. Среда Spring устраняет необходимость в поддержке собственных одноэлементных классов. Вместо этого, вы пишете класс как нормальный объект POJO (без необходимости в статических переменных/методах), а среда Spring гарантирует, что вы всегда получаете доступ к тому же самому объекту, если не переопределите класс по умолчанию как многоэлементный.
- Web-среда. Web-среда Spring – это хорошо проработанная среда выполнения MVC, которая предоставляет прекрасную альтернативу таким Web-средам, как Struts или другим, слишком сложным или менее популярным. Она позволяет разрабатывать экраны без формы, экраны с простыми формами, экраны, подобные мастерам, и многие другие. Это позволяет также связывать поля формы HTML непосредственно с бизнес-объектами, причем без необходимости писать специальные классы, которые дополняют классы Web-среды. Среда Web MVC Spring способна также работать с системами JavaServer Pages (JSP), Velocity, JavaServer Faces (JSF) и др. Кроме того, проект Spring Web Flow применим для разработки таких Web-приложений, которые требуют управления состоянием на протяжении нескольких запросов HTTP (как, например, у Web-сайта заказа авиабилетов).
- Четкая иерархия исключений. Среда Spring предоставляет API для преобразования исключений, определенных технологией (например, передаваемых JDBC, Hibernate или JDO), в независимые исключения (например, `org.springframework.dao.PessimisticLockingFailureException`).
- Классы управления транзакциями. Среда Spring предоставляет четкий интерфейс управления транзакциями, который может распространяться как на локальные транзакции (с использованием, например, единой базы данных), так и на глобальные (например, с использованием JTA). Среда Spring обеспечивает также интеграцию с определенными серверами приложений, например с сервером WebLogic Server от BEA и WebSphere от IBM. Управление транзакциями Spring применяется программно или декларативно. Мы будем использовать декларативную версию в главе 10, "Кроме основ".
- Облегченный контейнер. Контейнеры IoC считаются относительно облегченными, особенно по сравнению с контейнерами EJB, например. Это может оказаться выгодным при разработке и развертывании прикладных программ на компьютерах с ограниченными ресурсами памяти и процессора.

Здесь мы рассмотрели лишь некоторые из преимуществ, в частности, те, которые применимы к приложению Time Expression. Но если вы решите использовать среду Spring более интенсивно, то, вероятно, создадите и собственный список преимуществ.

# Фундаментальные концепции Spring

На настоящий момент мы рассмотрели достаточно много вводного материала о среде Spring. Но этот материал относится, в основном, к организации и возможно-

стям Spring. Нам все еще предстоит рассмотреть некоторые фундаментальные концепции, лежащие в основе Spring. Давайте сделаем это сейчас, перед тем как перейти к коду Spring, связанному с приложением Time Expression (в следующей главе).

## Схема внедрения зависимости (и контейнеры IoC)

В 2004 году Мартин Фаулер опубликовал статью (`http://www.martinfowler.com/articles/injection.html`), в которой обсуждались контейнеры *инверсии управления* (Inversion of Control – IoC). Мартин Фаулер и некоторые сотрудники компании PicoContainer, включая Рода Джонсона (автора Spring Framework), совместно определили схему внедрения зависимости. *Внедрение зависимости* (dependency injection) – это стиль IoC; другой стиль подразумевает использование технологии шаблонов и обратного вызова. В обоих случаях вы передаете управление чему-то еще (отсюда и термин инверсия управления). Мы будем, конечно, использовать стиль внедрения зависимости.

Так что же такое внедрение зависимости? Давайте рассмотрим эти два слова по отдельности. Сначала *зависимости*; это, в основном, подразумевает связь между двумя классами. Например, для выполнения некой задачи класс A мог бы нуждаться в классе B. Другими словами, класс A зависит от класса B. Теперь рассмотрим *внедрение*. Это означает, что класс B *внедряется* в класс A как в контейнер IoC. Подобное внедрение может осуществляться как в ходе передачи параметров для конструктора, так и после конструктора, с использованием методов установки значений (setter). (Данные способы более подробно описаны далее в этой главе.)

На рис. 6.4 представлена связь между двумя классами, которые будем использовать в нашем демонстрационном приложении Time Expression (следующая глава). Связь, или *ассоциация* (association), представленная на этой схеме, точно соответствует зависимости, которую мы отобразим в файле контекста приложения Spring. Другими словами, класс `TimesheetListController` зависит от класса `TimesheetManager`; следовательно, мы будем использовать среду Spring для того, чтобы автоматически создать один экземпляр класса `TimesheetManager` и *внедрить* (связать) его при помощи метода `TimesheetListController.setTimesheetManager`.


**Рис. 6.4.** Пример связи, используемой для установления зависимости между контейнерными классами при использовании среды Spring

Этот пример не может, конечно, полностью убедить в преимуществах использования схематического подхода при проектировании прикладных программ. Однако после того как начнете применять этот подход в работе, вероятно, не захотите больше использовать операторы new, которые не только норовят запутать ваш код Java существенно больше, чем внедрение зависимости, но и связывают ваши ассоциативные классы немного жестче.

## Два стиля внедрения

Существует два основных стиля внедрения зависимости: первый — при помощи аргументов, передаваемых конструктору при создании объекта, и второй — при помощи методов установки значения класса в стиле JavaBean уже после того, как объект был создан.

Среда Spring приветствует использование внедрения зависимости на базе методов установки значения (в справочной документации Spring), поскольку передача слишком большого количества аргументов в конструктор может вызвать затруднения. С другой стороны, Мартин Фаулер утверждает (по адресу `http://www.martinfowler.com/articles/injection.html`), что если вы имеете конструктор со слишком большим количеством аргументов и ваш объект оказывается чересчур загроможден, то имеет смысл рассмотреть возможность разделения класса.

Хотя оба подхода имеют свои "за" и "против", мы будем полагаться на рекомендации документации среды Spring и, соответственно, использовать для приложения Time Expression внедрение на основе методов установки значения. Однако лично я предпочитаю стиль внедрения JavaBean, подразумевающий применение методов получения и установки значений. По некоторым причинам это кажется мне немного более естественным при программировании в Java, но вы можете не согласиться с этим. Чтобы не создавать лишних сложностей, группа Pico Container предпочитает внедрение на основе конструктора и обосновывает ряд вполне справедливых причин для этого на своем Web-сайте (`http://www.picocontainer.org/Why+Constructor+Injection`).

В любом случае имеет смысл хорошо знать оба стиля, поддерживаемых Spring. Мы рассмотрим их вскоре.

> **Примечание**
> Фаулер предлагает также третий тип, *интерфейсное внедрение* (interface injection), который не рассматривается здесь, но описан по адресу `http://www.martinfowler.com/articles/injection.html`.

## Компоненты, интерфейсы BeanFactory и ApplicationContext

Интерфейс `org.springframework.beans.factory.BeanFactory` — это, фактически, контейнер IoC! Это интерфейс, который, по существу, управляет конфигурацией приложения за счет создания экземпляров и манипулирования компонентами, определенными в коде или, как в данном случае, в файле XML. Компонентом может быть объект любого типа, но обычно это классы в стиле JavaBean, т.е. классы с методами получения и установки значений.

Интерфейс `org.springframework.context.ApplicationContext`, по существу, дополняет интерфейс `BeanFactory` и добавляет такие средства, как *связи ресурсов* (resource bundle), интеграция с AOP Spring, обработка *ресурсов сообщений* (message resource), *распространение сообщений о событиях* (event broadcasting) и многое другое. Кроме того, интерфейс `WebApplicationContext` дополняет интерфейс `ApplicationContext` средствами, специфическими для Web-приложений. Хотя вы можете использовать реализацию интерфейса `BeanFactory` (например, `XmlBeanFactory`), для серверных приложений рекомендуется использовать интерфейс `ApplicationContext`.

Архивный файл кода этой книги содержит файл springtest-applicationcontext.xml — типичный пример файла контекста приложения Spring. Этот пример демонстрирует внедрение как на основании методов установки значения, с использованием элемента <property>, так и на основе конструктора, с использованием элемента <constructor-arg>. В этом примере обратите внимание на то, что компонент springtestmessage зависит от компонента stringmessage, как показано далее:

```
<bean id="springtestmessage"
    class="com.visualpatterns.timex.test.SpringTestMessage"
    lazy-init="false" init-method="printMessage">
    <property name="message" ref="stringmessage" />
</bean>
```

Обратите внимание на то, что здесь имеется также метод инициализации, использующий атрибут init-method; это прекрасный способ вызова методов инициализации и завершения в объектах. Эти отношения (или ассоциация) устанавливаются при помощи элемента <property>, а внедрение — при помощи метода SpringTestMessage.setMessage, как показано далее:

```
public void setMessage(String message)
{
    this.message = message;
}
```

Наш класс SpringTest.java (также находящийся в архивном файле кода этой книги) демонстрирует, насколько просто загрузить законченный файл контекста приложения и немедленно начать использовать различные компоненты, со всеми сопутствующими классами, автоматически связанными контейнером Spring IoC, как продемонстрировано здесь:

```
public static void main(String args[]) throws Exception
{
    FileSystemXmlApplicationContext factory =
        new FileSystemXmlApplicationContext(
            "src/conf/springtest-applicationcontext.xml");

    SpringTestMessage stm = (SpringTestMessage) factory
        .getBean("springtestmessage");
}
```

> **Примечание**
> В этом примере я выбрал класс FileSystemXmlApplicationContext ради простоты. Однако, как правило, лучше использовать класс ClassPathXmlApplicationContext, поскольку он осуществляет поиск файлов определения контекста не только в путях к классу, но и в пределах встроенных файлов JAR.

Прежде чем мы перейдем к следующему разделу, давайте рассмотрим приведенный ниже список дополнительных замечаний о компонентах.

- Компоненты в среде Spring могут быть созданы при помощи конструктора (как вы обычно используете оператор new) или при помощи статических методов другого класса.

- Каждый компонент должен иметь уникальный идентификатор (например, `springtestmessage`).
- Компоненты бывают двух типов, одноэлементные (Singleton) и прототипы (Prototype), т.е. неодноэлементные. По умолчанию все компоненты одноэлементные, но вы можете сделать их прототипами, определив, например, атрибут `singleton="false"`. Для приложения Time Expression будем использовать только одноэлементные компоненты, поскольку среда Spring не может управлять жизненным циклом прототипного компонента после того, как он был создан и передан объекту (внедрен), от которого зависит. Кроме того, определяя атрибут `singleton="false"` для данного компонента, вы инструктируете среду Spring создавать новый экземпляр компонента каждый раз, когда он нужен. И наконец, практически в 100% случаев вполне достаточно использования одноэлементных компонентов.
- Метод установки и конструктор позволяют вводить значения и как отдельные объекты, и как элементы коллекции (например, `List`, `Set`, `Map` и `Properties`).

Вот это и все, что мы скажем о классах `BeanFactory` и `ApplicationContext` в этой главе. После того как перейдем к Spring Web MVC Framework (в следующей главе), нам не придется волноваться о создании этих объектов вручную.

Но если вас заинтересуют подробности их работы, можете ознакомиться со справочной документацией Spring (`springframework.org`). Например, вам может понадобиться глубже изучить аргументы конструктора, проверку зависимости, автоматическое связывание взаимозависимых компонентов, программное взаимодействие с `BeanFactory` при введении значений `null`, внедрение на основе методов и многое другое. Примеры применения некоторых из этих элементов будут также приведены в последующих главах этой книги.

Файл `spring-beans.xsd` (или его прежняя версия `spring-beans.dtd`) предоставляет схему XML для файла контекста приложения среды Spring Framework. Просмотр этих файлов, расположенных в программных каталогах среды Spring, мог бы дать вам некоторое дополнительное представление о различных элементах и свойствах, которые можно определить в файле контекста приложения.

### Редакторы свойств

Среда Spring затрудняет использование интерфейса `java.beans.PropertyEditor`, позволяя регистрацию специальных редакторов свойств, которые преобразуют объект в читаемый текст и наоборот. Например, в следующей главе мы будем использовать эту возможность для преобразования количества часов, введенного пользователем на Web-странице Enter Hours.

## Дополнительные проекты Spring

Как только решишь, что знаешь о среде Spring достаточно, оказывается, что существует что-то новое. До сих пор мы изучили лишь модули и пакеты, являющиеся частью базового комплекта среды Spring. Однако существуют также другие, дополнительные проекты (перечисленные ниже), выпущенные группой Spring и доступные на Web-сайте `springframework.org`:

- Acegi Security System for Spring. Это проект, предоставляющий полный комплект служб обеспечения безопасности для Spring Framework (например, защиту HTTP, защиту экземпляра объекта, защиту библиотеки дескрипторов, кодирование паролей и т.д.).
- Spring BeanDoc. Это инструмент, облегчающий документирование и схематизацию приложений Spring и файлов контекста приложения. Очень полезный, простой и гибкий инструмент.
- Spring IDE for Eclipse. Это графический пользовательский интерфейс для файлов конфигурации, используемых Spring-ориентированными приложениями. Мы познакомимся с ним в главе 8, "Феномен Eclipse!".
- Spring Rich Client. Лично я не работал с этим проектом, поэтому процитирую Web-сайт Spring, который утверждает, что это "прекрасная альтернатива для разработчиков, нуждающихся в платформе для быстрого создания приложений Swing".
- Spring Web Flow. Этот проект основан на концепции, близкой к Spring Web MVC Framework. По существу, он предоставляет похожие на мастера экранные функциональные возможности для реализации различных аспектов коммерческой деятельности. В последнее время этому проекту уделяется много внимания, и его стоит обязательно опробовать, если вам необходим данный тип функциональных возможностей.

# Резюме

В этой главе мы сделали следующее.

- Получили четкое представление о том, что представляет собой среда Spring Framework, каковы ее фундаментальные концепции, как она организована и каковы многочисленные преимущества использования этой среды.
- Изучили такие фундаментальные концепции среды Spring, как внедрение зависимости, компоненты и фабрики компонентов, контекст приложения, редактор свойства и т.д.
- Выяснили, как упакована среда Spring Framework, с точки зрения разработки и развертывания.

В этой главе мы изучили достаточно много материала по основным принципам Spring Framework. В следующей главе мы подробнее рассмотрим модули Spring, связанные с Web. В последующих главах мы также обратим внимание на следующие средства Spring:

- декларативное управление транзакциями;
- планирование заданий;
- передача электронной почты.

Теперь пришло время разработать несколько экранов приложения Time Expression, использующих среду Spring Web MVC Framework!

И последнее замечание, прежде чем двигаться дальше: как вы могли заметить, здесь довольно часто упоминается справочная документация среды Spring. Это весьма

ценный документ. Но если вы не можете найти нужную информацию там, попробуете поискать ее в документации Spring Framework API JavaDocs, которая дополняет основную справочную документацию.

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, содержится на следующих Web-сайтах:

- Система безопасности Acegi для Spring `http://acegisecurity.org/`
- Apache Geronimo `http://geronimo.apache.org`
- Альянс AOP (стандарты Java/JEE AOP) `http://aopalliance.sourceforge.net/`
- Внедрение зависимости `http://www.martinfowler.com/articles/injection.html`
- SDK Eclipse `http://www.eclipse.org/aspectj/`
- Excalibur `http://excalibur.apache.org/`
- HiveMind `http://jakarta.apache.org/hivemind/`
- Сервер приложений JBoss `http://www.jboss.com`
- Ложные объекты `http://mockobjects.com`
- PicoContainer `http://www.picocontainer.org/`
- Software Practices Lab `http://www.cs.ubc.ca/labs/spl/`
- Форумы обсуждения Spring `http://forum.springframework.org/`
- IDE Spring `http://springide.org/project`
- Среда Spring `http://springframework.org`

# 7

# Среда Spring Web MVC Framework


**Стив:** Радж, разве жизнь не прекрасна? Пользователи приняли итерацию 2, Spring есть, листья распускаются, а Eclipse займемся позже на этой неделе.

**Радж:** Звучит, Стив, прекрасно. Между прочим, эта интегрированная среда разработки, о которой я читал, кажется многообещающей. Кроме того, для нее есть сотни бесплатных и коммерческих дополнений. Не могу дождаться, хочется ее опробовать!

В предыдущей главе был сделан лишь краткий обзор среды Spring Framework. Мы обсудили, что представляет собой среда Spring, как она организована и какие модули содержит. Кроме того, было упомянуто, что среда Spring не обязывает к бескомпромиссности при попытке решить, должны ли вы использовать среду Spring. Другими словами, на основании ваших потребностей вы можете решить, какие модули среды Spring Framework использовать (наряду со всеми зависимостями). В этой главе будет продемонстрировано, как использовать среду Spring Web MVC Framework (модуль) и создать приложение Time Expression, наше типовое Web-приложение. Обратите внимание на то, что с этого момента я буду упоминать среду Spring Web MVC Framework как Spring MVC.

# Что рассматривается в этой главе

В этой главе мы предпримем следующее.

- Выясним преимущества использования Spring MVC
- Рассмотрим среду Spring Web MVC Framework более подробно
- Используя Spring MVC, создадим три экрана для приложения Time Expression: контроллер без формы, два контроллера с формами и перехватчик HTTP Spring.

**Примечание**
Полный код примеров, используемых в этой главе, находится внутри файла zip, доступного на Web-сайте книги.

Это захватывающая глава, поэтому не буду больше тратить время впустую, надоедая вам вводным материалом. Давайте приведем пружину[1] в действие!

# Преимущества Spring Web MVC Framework

Среда Spring Web MVC Framework надежна, гибка и весьма удобна для быстрой разработки Web-приложений, использующих схему проектирования MVC. Преимущества, извлекаемые из использования этого модуля Spring, подобны тем, которые предоставляет остальная часть среды Spring Framework. Давайте рассмотрим некоторые из них. В этой главе я продемонстрирую следующие преимущества.

- Простота проверки. Это наиболее общее свойство, присущее всем классам Spring. Тот факт, что большинство классов Spring разработано как JavaBeans, позволяет задействовать проверку данных при помощи методов установки значений этих классов. Среда Spring предоставляет также ложные классы, предназначенные для моделирования объектов Java HTTP (например, `HttpServletRequest`), которые существенно упрощают проверку модулей слоя.
- Непосредственная привязка к бизнес-объектам. Spring MVC не требует, чтобы ваши бизнес-классы дополнили какие-либо специальные классы; это позволяет повторно использовать бизнес-объекты, привязывая их непосредственно к полям форм HTML. Фактически, ваши контроллерные классы — это и все, что не-

[1] Игра слов, дословно spring — пружина. — *Примеч. ред.*

обходимо для применения классов Spring (или реализации интерфейса контроллера Spring).

- Простота разделения ролей. Spring MVC прекрасно разделяет роли, которые выполняют различные компоненты, составляющие эту Web-среду. Например, когда начнем обсуждение таких концепций, как контроллеры, командные объекты и системы проверки достоверности, вы поймете, что каждый компонент играет свою роль, отличную от других.
- Адаптивные контроллеры. Если ваше приложение не требует форм HTML, можете использовать более простую версию контроллера Spring, который не нуждается во всех дополнительных компонентах, обязательных для контроллеров форм. Фактически, среда Spring предоставляет несколько типов контроллеров, каждый из которых предназначен для собственной цели. Например, существуют контроллеры без форм, простые контроллеры форм, контроллеры форм, подобные мастеру, представления без контроллеров и даже упакованные контроллеры, которые позволяют вам создавать представления без вашего собственного специального контроллера.
- Простая, но очень мощная библиотека дескрипторов. Библиотека дескрипторов Spring невелика, проста, но очень мощна. Например, для аргументов дескриптора `<spring:bind>` среда Spring использует язык выражений JSP.
- Поток Web. Этот модуль является дополнительным проектом и не входит в базовый комплект ядра Spring. Он встраивается поверх Spring MVC и облегчает создание подобных мастерам Web-приложений, контролирующих несколько запросов HTTP (например, сетевая корзина покупателя).
- Технологии представления Web-среды. Хотя в качестве технологии представления мы используем JSP, среда Spring поддерживает и другие технологии представления, такие как Apache Velocity (`jakarta.apache.org/velocity/`) и FreeMarker (`freemarker.org`). Это мощнейшая концепция, поскольку переход с JSP на Velocity – это вопрос конфигурации. Кроме того, среда Spring обеспечивает интеграцию с Apache Struts (`struts.apache.org`), Apache Tapestry (`jakarta.apache.org/tapestry`) и WebWork от OpenSymphony (`opensymphony.com/webwork/`).
- Облегченная среда. Как я упомянул в предыдущей главе, среда Spring позволяет создавать готовые корпоративные приложения, используя объекты POJO. Установка среды оказывается проще и дешевле, поскольку вы можете разрабатывать и развертывать ваши приложения, используя облегченный контейнер сервлета.

# Концепции Spring Web MVC

Мир Java видел много Web-сред разработки на основе схем MVC, неожиданно появившихся за последние несколько лет (некоторые из них перечислены в конце этой главы). Схема проекта *модель-представление-контроллер* (Model-View-Controller – MVC) была первоначально задумана в компании XEROX PARC примерно в 1978-79 годах, она развивалась и впоследствии была реализована в виде библиотеки классов Smalltalk-80 (также на XEROX PARC). Эта относительно простая концепция подразумевает полное разделение представления и данных, как я сейчас объясню вкратце.

Сначала рассмотрим нашу схему архитектуры, созданную в книге ранее и представленную здесь на рис. 7.1.

Как можно заметить, все запросы HTTP, поступающие от Web-браузера, обрабатываются контроллерами. *Контроллер* (controller), как и следует из его названия, контролирует модель и представление, облегчая обмен данными между ними. Ключевое преимущество этого подхода заключается в том, что модель может заботиться только о данных и не иметь дела с представлением. Представление, с другой стороны, может не заботиться о модели и бизнес-логике, просто визуализируя передаваемые ему данные (как Web-страницы, в данном случае). Схема MVC позволяет также изменять представление без необходимости менять модель.

Давайте рассмотрим некоторые из фундаментальных концепций Spring MVC. В первую очередь изучим концепции, связанные с программированием Java, а затем рассмотрим конфигурацию, необходимую для выполнения этих работ.

## Концепции Spring MVC для Java

На рис. 7.1 отображена высокоуровневая архитектура приложения Time Expression. Теперь давайте немного подробнее рассмотрим компоненты Spring MVC. На рис. 7.2 демонстрируется полная последовательность экранов типового приложения Time Expression. Эта схема демонстрирует большинство концепций, которые мы обсудим сейчас.


**Рис. 7.1.** Высокоуровневая архитектура приложения Time Expression

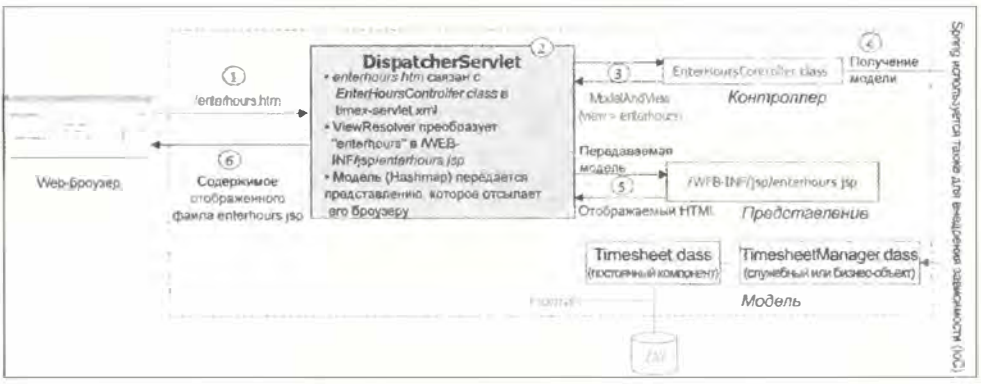



**Рис. 7.2.** Полная последовательность действий экрана Enter Hours с учетом использования Spring и Hibernate

### Контроллер

Среда Spring предоставляет множество типов контроллеров. Это, может быть, и хорошо, и плохо. Хорошо то, что вы имеете набор контроллеров, из которых можно выбирать подходящие, плохо то, что первое время выбрать подходящие трудно. Наилучший способ выбора подходящего типа контроллера, вероятно, заключается в знании необходимых функциональных возможностей. Например, содержат ли ваши экраны форму? Нужны ли функциональные возможности, подобные мастеру? Возможно, необходима только переадресация на страницу JSP, и никакой контроллер не нужен вообще? Примерно таковы вопросы, которые стоит задать себе, чтобы сузить круг вариантов при выборе.

На рис. 7.3 представлена схема классов некоторых из более интересных контроллеров, которые являются частью Spring MVC. Табл. 7.1 содержит краткие описания интерфейсов и классов, представленных на рис. 7.3. (*Примечание:* предоставленные в этой таблице описания взяты непосредственно из документации Spring Framework Javadocs.) Как правило, я предпочитаю использовать контроллеры `SimpleFormController` `UrlFilenameViewController` и `AbstractController`. Примеры их применения будут приведены далее в этой главе.

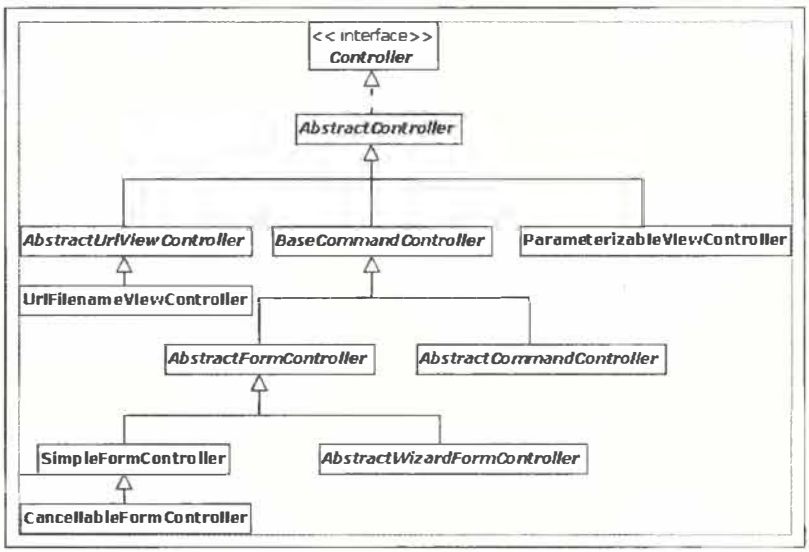



**Рис. 7.3.** Схема класса, демонстрирующая некоторые из контроллеров Spring

**Таблица 7.1. Описание некоторых контроллеров Spring**

| Контроллер | Описание (взято непосредственно из Spring Javadocs) |
|---|---|
| `AbstractCommandController` | Абстрактный базовый класс для контроллеров специальных команд |
| `AbstractController` | Суперкласс, предназначенный для реализаций контроллера, использующего схему разработки Template Method |

*Окончание табл. 7.1.*

| Контроллер | Описание (взято непосредственно из Spring Javadocs) |
|---|---|
| `AbstractFormController` | Контроллер форм, который автоматически заполняет форму компонента из запроса |
| `AbstractUrlViewController` | Абстрактный базовый класс для контроллеров, которые возвращают имя представления на основании URL |
| `AbstractWizardFormController` | Контроллер форм для типичных последовательностей в стиле мастера |
| `BaseCommandController` | Реализация контроллера, которая при получении запроса создает объект (командный объект) и пытается заполнить этот объект параметрами запроса |
| `CancellableFormController` | Расширение контроллера `SimpleFormController`, который обеспечивает "отмену" обработки формы |
| `Controller` | Базовый интерфейс контроллера, представляющий компонент, который получает `HttpServletRequest` и `HttpServletResponse`, подобно `HttpServlet`, но может участвовать в рабочем потоке MVC |
| `ParameterizableViewController` | Обычный контроллер, который всегда возвращает именованное представление |
| `SimpleFormController` | Конкретная реализация контроллера `FormController`, которая предоставляет конфигурируемую форму, форму успеха и цепь `onSubmit` для соответствующего переопределения |
| `UrlFilenameViewController` | Контроллер, который преобразует виртуальное имя файла, расположенное в конце URL, в имя представления и возвращает это представление |

### Модель и представление

Большинство методов, имеющих отношение к контроллерам, возвращают объект класса `org.springframework.web.servlet.ModelAndView`. Этот объект содержит модель (как объект `java.util.Map`) и имя представления, а также позволяет методу возвратить их оба в одном значении. Примеры того, как это дается, мы увидим далее в этой главе, когда дойдем до создания экранов приложения Time Expression.

### Командный объект (поддержка формы)

Среда Spring использует понятие *командного объекта* (command object), который, по существу, является классом в стиле JavaBean и заполняется данными из полей фор-

мы HTML. Это тот же объект, который передается нашему классу validator (обсуждаемому впоследствии) для проверки правильности данных и, если проверка пройдена, методу onSubmit (контроллера, связанного с классом) для обработки корректных данных. С учетом того, что командный объект — это просто класс в стиле JavaBean, можем использовать наши бизнес-объекты для связывания данных непосредственно, а не создавать специальные классы только для связывания данных. Я продемонстрирую это преимущество далее в этой главе.

### Класс Validator

Класс Spring validator является необязательным классом, который может быть использован для проверки данных формы, предназначенных для определенной команды (формы) контроллера. Класс validator — это конкретный класс, который реализует интерфейс org.springframework.validation.Validator. Для этого интерфейса обязателен метод validate, который, как упоминалось ранее, передает объекты command и Errors, применяемые для возвращения сообщения об ошибке. Пример класса validator я приведу далее в этой главе. Еще один весьма популярный класс проверки правильности, org.springframework.validation.ValidationUtils, предоставляет весьма удобные методы для защиты от пустых полей.

### Библиотека дескрипторов Spring (spring:bind)

Подключаемая библиотека дескрипторов Spring проста, но все же очень мощна. Как правило, она применяется в файлах JSP при помощи дескриптора <spring:bind>, который по существу связывает поля формы HTML с объектом command. Кроме того, в пределах JSP она обеспечивает доступ к специальным переменным, таким как ${status.value}, ${status.expression} и ${status.errorMessages}; их мы рассмотрим позже.

## Концепции конфигурации Spring MVC

В этом разделе рассмотрим ряд фундаментальных концепций, связанных с настройкой Spring Web MVC Framework.

### Класс DispatcherServlet

Класс DispatcherServlet (часть пакета org.springframework.web.servlet) — это точка входа в мир Spring Web MVC, как отображено на рис. 7.2. По существу, он посылает запросы на контроллеры. Если вы работали с Web-приложениями Java прежде, то не будете удивлены, обнаружив конфигурацию этого класса в файле web.xml, как показано в следующем фрагменте файла web.xml приложения Time Expression:

```
<servlet-class>
    org.springframework.web.servlet.DispatcherServlet
</servlet-class>
```

Более подробная информация о классе DispatcherServlet приведена далее в этой главе.

### Связывание обработчиков

Вы можете связать обработчики с поступающими запросами HTTP в файле контекста приложения Spring. *Обработчик* (handler) — это, как правило, контроллер, ко-

торый связан с частичным или полным URL поступающего запроса. Связанный обработчик может также содержать необязательные *перехватчики* (interceptor), вызываемые до и после обработчика. Это мощная концепция. Я продемонстрирую ее пример далее в этой главе, когда будем использовать такой Web-перехватчик при аутентификации и закрытии нашего сеанса Hibernate для данного запроса HTTP.

Следующий отрывок кода, взятый из нашего файла `timex-servlet.xml`, демонстрирует, как обработчик может быть связан с частичным URL:

```
<bean id="urlMap"

class="org.springframework.web.servlet.handler.SimpleUrlHandlerMappin">
    <property name="urlMap">
        <props>
            <prop key="/signin.htm">signInController</prop>
            <prop key="/signout.htm">signOutController</prop>
        </props>
    </property>
</bean>
```

### Преобразователи представления

Среда Spring использует концепцию *преобразователей представления* (view resolver), которые преобразуют имена представлений в фактические представления (например, `enterhours` в `enterhours.jsp`). Для преобразования имен наших представлений будем использовать класс Spring `InternalResourceViewResolver`. (Это рассматривается в следующем разделе.)

# Настройка Spring для Time Expression

Теперь, когда я снабдил вас фундаментальными концепциями Spring MVC, давайте применим их при создании экранов приложения Time Expression. Для запуска среды Spring необходима пара компонентов; см. рис. 7.1, представляющий созданную ранее высокоуровневую архитектуру приложения Time Expression. Как можно заметить, необходим контейнер сервлета, в пределах которого может выполняться среда Spring для нашего Web-приложения. Итак, начнем с установки контейнера сервлета, а затем загрузим и установим среду Spring Framework.

## Установка контейнера сервлета (Apache Tomcat)

В качестве контейнера сервлета для приложения Time Expression я решил использовать Apache Tomcat (`http://tomcat.apache.org/`). Но вы можете использовать любой другой продукт, который считаете нужным; это может быть продукт только для контейнера-сервлета, например Tomcat, или полнофункциональный сервер приложений, такой как JBoss Application Server, BEA WebLogic или IBM Websphere.

> **Примечание**
> Если вы следовали примерам этой книги, вероятно, помните файл `timex/local.properties`, использовавшийся нашим файлом Ant `build.xml` (оба файла содержатся в архивном файле кода этой книги). Обратите внимание на свойство `deploy.dir` в файле `timex/local.properties`; вго можно откорректировать так, чтобы оно указывало на ваш каталог, где установлен контейнер сврвлета. Например, в моем случае свойство `deploy.dir` установлено следующим образом:

```
deploy.dir=/apache-tomcat-5.5.15/webapps
```

Теперь мы можем ввести и выполнить команду **ant deploy** в командной строке, используя наш файл `build.xml`.

В результате выполнения этой команды ant, будет создан и развернут в соответствующем каталоге (указанном свойством `deploy.dir`) новый файл Web-архива `timex.war`.

**Быстрое развертывание файлов WAR и проверка в стиле HTTP Mock**

В 2001 году я написал статью "How Many Times Do You Restart Your Server During Development?" (Сколько раз вы перезагружаете ваш сервер в течение разработки?) (`http://www.javaworld.com/javaworld/jw-04-2001/jw-0406-soapbox.html`). Хотя различные контейнеры сервлетов и серверов приложений требуют разное количество перезагрузок, частые перезагрузки сервера при каждом изменении вашего приложения не только надоедают, но и отнимают много времени. Большинство из этих случаев имеет отношение к пути загружаемых классов Java, но расстройства от этого не меньше.

Если ваш сервер не обеспечивает успешной переустановки файлов `.war` (без перезагрузки), вы могли бы сменить стиль программирования и проверки. Хорошей альтернативой (обсуждается в этой главе) является использование ложных классов Spring для моделирования запросов HTTP и использования при проверке кода модулей JUnit, вместо их установки на сервере Web-приложений.

Недавно я случайно наткнулся на возможность Apache Tomcat, позволяющую избежать перезагрузок при развертывании нашего приложения. Ее можно задействовать, установив приведенные ниже атрибуты в файле `conf/context.xml`, расположенном в каталоге Tomcat.

```
<Context antiJARLocking= "true" antiResourceLocking="true">
```

Документация по этим атрибутам находится по адресу `http://tomcat.apache.org/tomcat-5.5-doc/config/context.html#Standard%20Implementation`.

В качестве альтернативы мы могли бы использовать задачу Tomcat Ant `deploy`, но я хотел сделать наш файл `build.xml` подходящим для большинства Web-серверов. Однако документация по этим задачам находится на Web-сайте `tomcat.apache.org`.

## Установка Spring Framework

Теперь вы должны иметь полное представление о том, что среда Spring может сделать для вас. Итак, пришло время загрузить среду Spring, установить ее и начать использовать! Загрузить Spring Framework можно по адресу `http://springframework.org`. Для загрузки и установки мы будем теперь следовать инструкции, предоставляемой на этом Web-сайте. Ниже приведены шаги, которые нам предстоит сделать для установки Spring в нашей системе. Здесь можно также установить в каталог `timex/lib/` некоторые файлы JAR, обеспечивающие дополнительные функциональные возможности Spring. (В главе 10, "Кроме основ", мы добавим в этот каталог файл `quartz.jar` от OpenSymphony.)

- Spring. Скопируйте файл `spring.jar` в каталог `timex/lib/` приложения Time Expression, на основании структуры каталога, созданной в главе 3, "Архитектура и модель проекта на базе XP и AMDD", и представленной здесь на рис. 7.4.
- JSTL. Нам необходимо также получить *стандартную библиотеку дескрипторов страниц JavaServer* (JavaServer Pages Standard Tag Library — JSTL), являющуюся частью библиотеки дескрипторов проекта Jakarta (она доступна для загрузки по адресу `http://jakarta.apache.org/taglibs/`). После загрузки этого пакета скопируйте файлы `jstl.jar` и `standard.jar` в каталог `timex/lib/`. При-

менение библиотеки JSTL позволяет избежать (или, по крайней мере, существенно ограничить) встраивания кода скриптлетов (scriptlet) в наши файлы JSP. Например, библиотека JSTL предоставляет дескрипторы для итераций и циклов (например, `<forEach>`), условные дескрипторы (например, `<if>`), дескрипторы форматирования (например, `fmt:formatDate`) и некоторые другие. Примеры большинства этих дескрипторов вы увидите в данной главе.

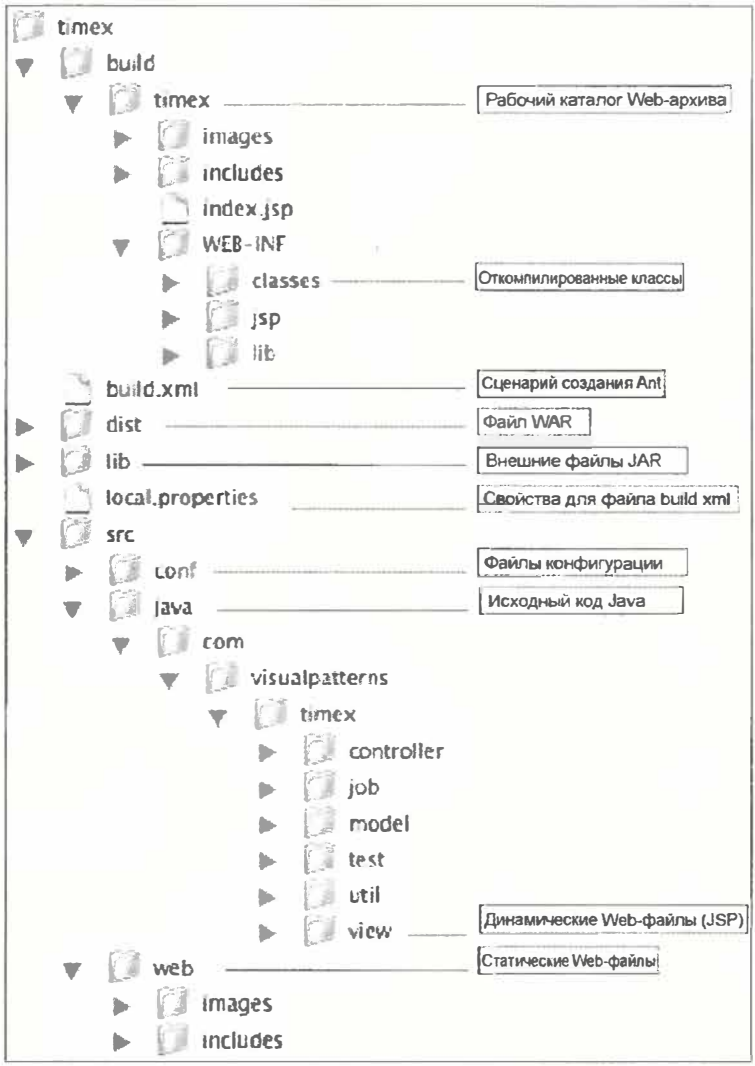



**Рис. 7.4.** Структура рабочего каталога для приложения Time Expression

## Запуск SpringTest

Кстати, эти три файла, которые мы обсуждали в предыдущей главе, теперь могут быть созданы в каталогах, указанных ниже, и мы могли бы выполнить команду `ant springtest` (в нашем корневом каталоге `timex/`), чтобы проверить, можем ли мы использовать среду Spring в нашем коде. Законченный код этих файлов находится в архивном файле кода этой книги.

- `src/conf/springtest-applicationcontext.xml`
- `src/java/com/visualpatterns/timex/test/SpringTest.java`
- `src/java/com/visualpatterns/timex/test/SpringTestMessage.java`

## Настройка Spring MVC

Теперь, когда имеется контейнер сервлета и установлено программное обеспечение Spring, необходимо настроить Spring MVC так, чтобы мы могли начать разработку и развертывание приложения Time Expression.

### Настройка класса `DispatcherServlet` в файле `web.xml`

В первую очередь необходимо организовать поступление всех запросов HTTP (которые соответствуют некоторой схеме) и перенаправление их сервером Tomcat на Spring MVC. Приведенный ниже отрывок файла `web.xml` демонстрирует, как мы можем настроить все запросы, заканчивающиеся расширением `.htm`, так, чтобы их обрабатывал класс Spring `org.springframework.web.servlet.DispatcherServlet`:

```
<servlet>
    <servlet-name>timex</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>timex</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Впоследствии мы рассмотрим, как запросы с расширением `.jsp` обрабатываются классом Spring `DispatcherServlet`.

> **Примечание**
> Наш файл контекста приложения Spring, `timex-servlet.xml`, будет автоматически найден и загружен средой Spring.
> Этот файл хранится в каталоге `timex/src/conf`, но автоматически копируется в каталог `timex/build/timex/WEB-INF/` нашим файлом Ant `build.xml`, когда он создает или развертывает используемые целевые объекты.

### Создание файла XML контекста приложения (`timex-servlet.xml`)

Теперь необходимо создать наш файл XML контекста приложения, `timex-servlet.xml`. До конца этой главы мы рассмотрим различные элементы этого файла. Вы заметите, что этот файл быстро становится основной частью работы в Spring MVC. Приведенный ниже отрывок файла `timex-servlet.xml` демонстрирует, как

мы настраиваем *преобразователь* (resolver) представления Spring, чтобы преобразовать логические имена представлений в физические файлы представлений (JSP):

```
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResol
ver">
    <property name="viewClass">
        <value>org.springframework.web.servlet.view.JstlView</value>
    </property>
    <property name="prefix" >
        <value>/WEB-INF/jsp/</value>
    </property>
    <property name="suffix" >
        <value>.jsp</value>
    </property>
</bean>
```

**Примечание**

Сохраняя наши файлы JSP в каталоге `build/timex/WEB-INF/jsp/`, мы, по существу, *скрываем* их, чтобы нельзя было обратиться к ним непосредственно из Web-браузера, используя их фактические имена (т.е. с этими файлами связаны только представления с окончанием `.htm`). Чтобы получить непосредственный доступ к файлам `.jsp`, их следует поместить в каталог на несколько уровней выше, `build/timex/`, где расположен также файл приветствия `index.jsp`. Сокрытие файлов — это защитная мера. Приложение Г, "Защита Web-приложений", содержит дополнительные правила безопасности.

# Разработка пользовательских интерфейсов приложения Time Expression при помощи Spring

Теперь, когда Tomcat и Spring установлены и настроены, мы можем предпринять действия, необходимые для разработки наших типовых экранов. Давайте рассмотрим два экрана приложения Time Expression, которые разработаем в этой главе. Один из них содержит форму HTML, а другой — нет.

## Экран Timesheet List

На рис. 7.5 представлен экран Timesheet List, который не содержит формы (т.е. здесь нет никаких полей, в которые пользователь может вводить данные; это единственный экран, являющийся только представлением). С точки зрения программирования контроллера, это самый простой экран, который вы можете разработать с использованием Spring MVC (его код мы рассмотрим вскоре).

## Экран Enter Hours

На рис. 7.6 демонстрируется экран Enter Hours, который является формой (т.е. он содержит поля, которые пользователь может заполнять). Этот экран немного сложнее, чем экран Timesheet List, поскольку мы должны связать поля формы HTML с нашим кодом Java, осуществить проверку правильности введенных данных, отобразить сообщения об ошибках и т.д.

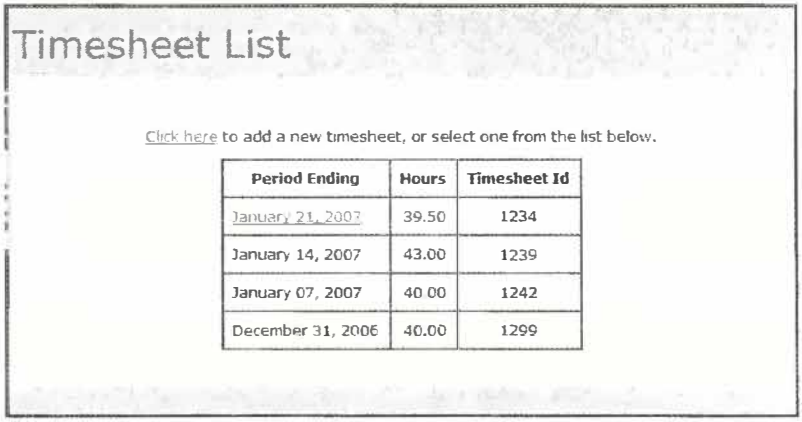



**Рис. 7.5.** Web-страница **Timesheet List** приложения Time Expression (имя представления: `timesheetlist`)

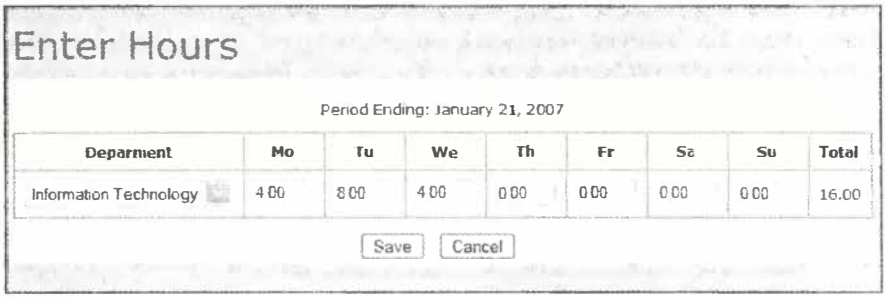



**Рис. 7.6.** Web-страница **Enter Hours** приложения Time Expression (имя представления: `enterhours`)

## Файлы Java

Теперь у нас имеется достаточно информации, чтобы придумать имена для файлов наших классов Java и файлов JSP (представлений). В табл. 7.2 содержится список соответствий классов представлений, контроллеров и сотрудников (модели), необходимых для этих двух экранов, представленных на рис. 7.5 и 7.6. Вы могли бы припомнить, что эту схему мы разработали еще в главе 3, "Архитектура и модель проекта на базе XP и AMDD"(см. табл. 3.5).

**Таблица 7.2. Типовая карта потока приложения (из главы 3, "Архитектура и модель проекта на базе XP и AMDD")**

| Дескриптор истории | Представление | Класс контроллера | Сотрудники | Закрепленные таблицы |
|---|---|---|---|---|
| Timesheet List | `timesheetlist` | `TimeSheetList-Controller` | `TimesheetManager` | Timesheet |
| Enter Hours | `enterhours` | `EnterHours-Controller` | `TimesheetManager` | Timesheet Department |

Обратите внимание на то, что упомянутые здесь классы сотрудников (они были разработаны в главе 5, "Применение Hibernate для постоянных объектов") необходимы для разработки классов представлений и контроллеров.

На рис. 7.7 представлена упрощенная схема взаимосвязи классов контроллера и модели.

Если вы разрабатывали Web-приложения в Java прежде, то могли бы подвергнуть сомнению размещение файлов `.jsp` в той же структуре каталога, что и классы Java (т.е. `java/com/visualpatterns/timex/`). Это мое личное предпочтение, поскольку мне нравится видеть мои файлы MVC, сгруппированными вместе в том же каталоге.

Давайте рассмотрим поэтапно, как разработать экраны Timesheet List и Enter Hours. Создание экрана Sign In мы рассмотрим позже, поскольку это частный случай из-за необходимости аутентификации (регистрации).

# Каскадная таблица стилей (CSS)

Еще один обсуждаемый здесь файл, кроме файлов Java и JSP, – это расположенный в каталоге `src/web/includes` файл `timex.css`, использующий *каскадную таблицу стилей* (Cascading Style Sheet – CSS). Таблицы CSS обеспечивают однозначность для всех наших пользовательских интерфейсов; кроме того, это позволяет уменьшить размер нашего кода JSP/HTML, поскольку нам не понадобится так много кода форматирования для наших файлов представления (JSP).

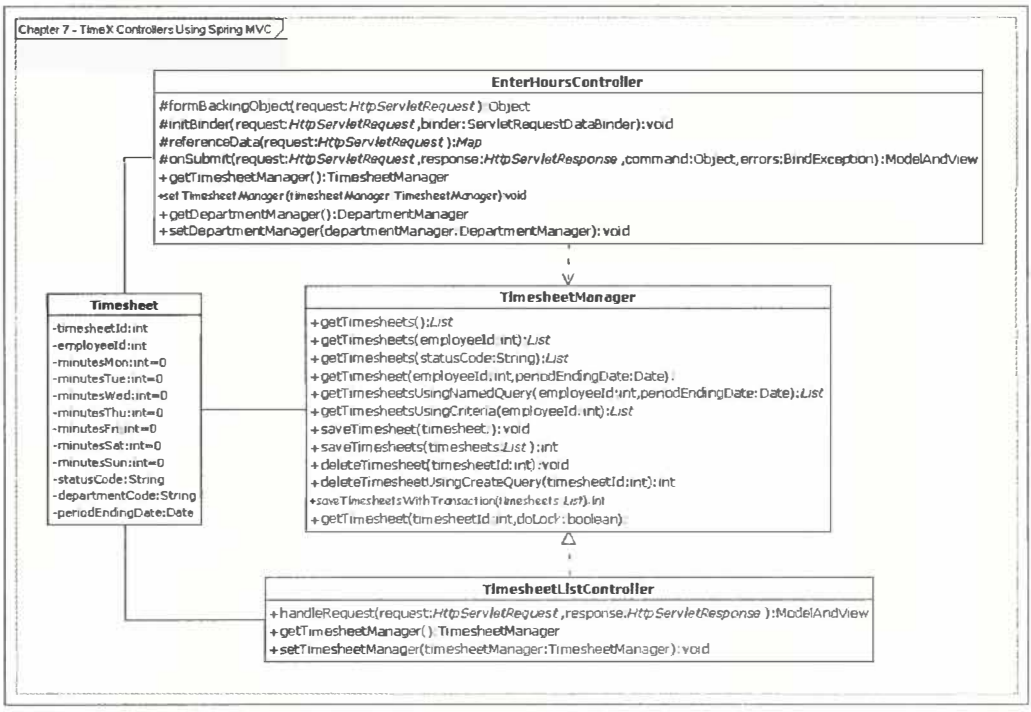



**Рис. 7.7.** Схема класса, демонстрирующая взаимоотношения между классами контроллера и модели приложения Time Expression

```
        <value>timesheetlist</value>
    </property>
</bean>
```

Обратите внимание на атрибуты `ref`. Как вы могли бы предположить, это ссылки на другие компоненты, определенные в нашем контексте приложения, как демонстрирует следующий отрывок файла XML:

```
<bean id="timesheetManager"
    class="com.visualpatterns.timex.model.TimesheetManager"/>
<bean id="applicationSecurityManager"
    class="com.visualpatterns.timex.util.ApplicationSecurityManager"/>
```

Класс `TimesheetManager` мы уже разработали в главе 5, "Применение Hibernate для постоянных объектов", а класс `ApplicationSecurityManager` разработаем далее в этой главе.

Это и все, что необходимо для настройки экрана Timesheet List. Теперь необходимо написать код контроллера и представления, ссылки на который находятся здесь. Давайте сделаем это позже.

## Поэтапное программирование

Экран Timesheet List относительно прост и будет разработан с использованием контроллера Spring самого простого типа. Поскольку он не содержит никаких полей формы, нам не понадобятся такие классы, как `Validator` и `Command`. Говоря проще, если мы посмотрим на это с точки зрения схемы разработки MVC, то для данного экрана понадобятся следующие файлы:

- модель — `TimesheetManager.java` и `Timesheet.java`
- представление — `timesheetlist.jsp`
- контроллер — `TimesheetController.java`

Файлы модели мы уже разработали в предыдущей главе, поэтому нам осталось создать только файлы контроллера и представления. Итак, давайте рассмотрим фрагменты нашего законченного кода.

Начнем с проверочного кода модуля для нашего класса контроллера.

### Создание предварительной проверки при помощи ложных объектов

Несколько следующих фрагментов кода из нашего файла `TimesheetListControllerTest.java` демонстрируют, как мы можем проверить модуль класса контроллера. Мы сделаем это в каталоге `timex/src/java/com/visualpatterns/timex/controller`.

Мы начнем с создания экземпляра класса `org.springframework.mock.web.MockHttpServletRequest`, чтобы смоделировать реальный запрос HTTP. Этот класс не только предоставляет преимущества проверки кода нашего модуля, но и уменьшает также вероятность переустановки приложения, а следовательно, перезагрузки контейнера сервлета (например, Tomcat) при каждый проверке.

```
mockHttpServletRequest = new MockHttpServletRequest("GET",
↳"/timesheetlist.htm");
```

Затем создадим ряд объектов проверки зависимости и *внедрим* их, как это автоматически будет делать среда Spring во время выполнения:

```
Employee employee = new Employee();
employee.setEmployeeId(EMPLOYEE_ID);
applicationSecurityManager.setEmployee(mockHttpServletRequest,
    employee);

// внедрить объекты, как это обычно делает Spring
timesheetListController = new TimesheetListController();
timesheetListController.setTimesheetManager(timesheetManager);
timesheetListController
    .setApplicationSecurityManager(applicationSecurityManager);
```

В этом проверочном коде мы привели наш собственный класс `TimesheetManager` ради простоты. Однако в реальном приложении, для создания экземпляра ваших классов, вы могли бы использовать классы Spring `FileSystemXmlApplicationContext` или `ClassPathXmlApplicationContext`. Таким образом, вы получаете не только экземпляр класса, но и объекты, зависимые от него, причем загруженные и внедренные средой Spring.

Теперь мы можем завершать нашу проверку, проверив только что полученный объект `java.util.List`. Наша проверка гарантирует, что список не пуст и что он содержит объекты `Timesheet` для сотрудников, записи которых затребованы:

```
ModelAndView modelAndView = timesheetListController.handleRequest(
    mockHttpServletRequest, null);

assertNotNull(modelAndView);
assertNotNull(modelAndView.getModel());

List timesheets = (List) modelAndView.getModel().get(
    TimesheetListController.MAP_KEY);
assertNotNull(timesheets);

Timesheet timesheet;
for (int i = 0; i < timesheets.size(); i++)
{
    timesheet = (Timesheet) timesheets.get(i);
    assertEquals(EMPLOYEE_ID, timesheet.getEmployeeId());
    System.out.println(timesheet.getTimesheetId() + " passed!");
}
```

Это и все о проверке модуля класса; теперь давайте рассмотрим реальный класс `TimesheetListController`.

> **Создание проверки модуля и собственно кода одновременно**
> Архивный файл кода этой книги содержит полный код нашего класса `TimesheetListControllerTest.java`, который предназначен для проверки JUnit контроллера `TimesheetController.java`. Как я уже упоминал ранее, разработку проверки модуля и работу над фактическим кодом лучше осуществлять одновременно. Например, я одновременно написал классы `TimesheetListControllerTest.java` и `TimesheetListController.java`; т.е. я создавал небольшой фрагмент кода, компилировал его, а затем проверял и снова повторял эти действия, пока мой класс контроллера не приобрел все функциональные возможности, в которых я нуждался. Очевидное преимущество этого подхода заключалось в том, что код моего модуля был проверен к моменту его завершения!

Кроме того, наш класс контроллера будет теперь содержать только необходимый код, не более и не менее.

Еще одним вполне очевидным преимуществом является то, что я получаю программный блок (подобный абзацу). Однако предварительная проверка кода модуля позволяет мне двигаться дальше. Обратите внимание на то, о чем я упомянул здесь, — это мой личный стиль работы, и вы вполне можете отвергнуть предварительную проверку (если вы так привыкли).

Я также хочу подчеркнуть, что везде необходим баланс. Хотя я предпочитаю предварительную проверку, иногда создание кода проверки модуля оказывается сложнее, чем создание фактического кода. В конце концов, вы пишете код Java для проверки другого кода Java, который поднимает вполне резонный вопрос: следует ли проверять проверочный код? Безусловно, я шучу, однако, по-моему, нужно не забывать о здравом смысле при разработке проверки модуля. И наконец, проверка модуля осуществляется лучше, если вы пишете небольшие методы, проверять которые относительно легко.

### Код контроллера

Теперь пришло время рассмотреть код нашего класса контроллера, `TimesheetListController.java`, для экрана **Timesheet List**. Мы создадим его в каталоге `timex/src/java/com/visualpatterns/timex/controller`.

Напомним, что мы реализуем интерфейс `org.springframework.web.servlet.mvc.Controller`; вероятно, самый простой тип контроллера, который можно разработать с использованием Spring.

```
public class TimesheetListController implements Controller
```

Следующее, на что следует обратить внимание, — это метод `handleRequest`. Это единственный метод, который мы должны реализовать, чтобы удовлетворить требования интерфейса контроллера.

```
public ModelAndView handleRequest(HttpServletRequest request,
                                  HttpServletResponse response)
```

Метод `handleRequest` возвращает объект `ModelAndView`, который содержит имя представления и данные модели (в данном случае список `java.util.List`). Имя представления преобразует объект `JstlView`, который мы определили в файле `timex-servlet.xml` ранее в этой главе.

```
return new ModelAndView(VIEW_NAME,
                        MAP_KEY,
                        timesheetManager.getTimesheets(employeeId));
```

Существует еще несколько вариантов того, как вы можете создать класс `ModelAndView`, это демонстрирует следующий список (см. подробности в документации Spring Framework API Javadocs):

- `ModelAndView()`
- `ModelAndView(String viewName)`
- `ModelAndView(String viewName, Map model)`
- `ModelAndView(String viewName, String modelName, Object modelObject)`
- `ModelAndView(View view)`
- `ModelAndView(View view, Map model)`
- `ModelAndView(View view, String modelName, Object modelObject)`

### Код представления и JSP

Мы уже создали прототип этого экрана в главе 2, "Простое приложение: сетевая система учета рабочего времени", поэтому нам необходимо добавить теперь некоторый код в соответствующие файлы представления (`.jsp`). Архивный файл кода этой книги содержит первоначальную, `timesheetlist.html` (прототип, статический HTML), и конечную, `timesheetlist.jsp` (динамический, JSP), версии этого файла.

Давайте рассмотрим наш файл `timesheetlist.jsp` немного ближе. Для начала создадим его в каталоге `timex/src/java/com/visualpatterns/timex/view`. Теперь давайте рассмотрим некоторый код JSP.

Приведенный ниже отрывок файла `timesheetlist.jsp` демонстрирует динамический код, используемый для заполнения таблицы HTML экрана Timesheet List; это сделано в цикле, созданном при помощи дескриптора JSTL `forEach`. В пределах каждого цикла мы создаем ряды и столбцы таблицы HTML (и также форматируем часы), используя базовую библиотеку JSTL.

```
<c:forEach items="${timesheets}" var="timesheet">
    <tr>
        <td align="center"><a
        href='enterhours.htm?eid=<c:out
        value="${timesheet.employeeId}"/>&amp;tid=<c:out
        value="${timesheet.timesheetId}"/>'><fmt:formatDate
        value="${timesheet.periodEndingDate}" type="date"
```

Теперь давайте рассмотрим другую интересную часть кода нашего файла представления `timesheetlist.jsp`:

```
<c:if test="${not empty message}">
    <font color="green"><c:out value="${message}"/></font>
    <c:set var="message" value="" scope="session"/>
</c:if>
```

Весь этот код проверяют все сообщения, хранимые в атрибуте сеанса `message`. Как будет указано далее в этой главе, эти сообщения посылает контроллер экрана Enter Hours при успешном сохранении в методе `onSubmit`.

Сейчас мы рассмотрели лишь то, как настроить и создать код экрана Timesheet List. Теперь пришло время рассмотреть более сложные возможности Spring MVC.

# Экран Enter Hours: пример контроллера формы

Рассмотренный только что пример экрана Timesheet List продемонстрировал разработку простого контроллера без формы. Теперь давайте рассмотрим более сложный пример – экран Enter Hours, представленный на рис. 7.6.

Как можно заметить на рис. 7.6, экран Enter Hours позволяет пользователям вводить свои рабочие часы и выбирать отдел, который эти часы должен оплатить (при помощи раскрывающегося списка). Эти функциональные возможности потребуют, чтобы мы получили список названий отделов, связали поля формы HTML с объектом Java, проверили введенные в форму данные и отобразили на экране сообщение об успехе или ошибке.

## Поэтапная настройка

Ниже приведена последовательность действий, необходимая для настройки экрана Enter Hours в нашем файле `timex-servlet.xml`. Для краткости, я не буду повторять объяснения тех же этапов, которые мы рассмотрели ранее на примере экрана Timesheet List.

### Связывание обработчиков

Следующая строка кода обеспечивает связывание представления Enter Hours с классом контроллера:

```
<prop key="/enterhours.htm">enterHoursController</prop>
```

### Определение контроллера и связанных классов

Конфигурация для контроллера экрана Enter Hours немного сложнее, чем для контроллера экрана Timesheet List, так что давайте рассмотрим ее подробнее. В первую очередь обратите внимание на то, что здесь имеется два класса модели и один класс, связанный с защитой (вспомогательный); они необходимы для работы экрана Enter Hours и настроены следующим образом:

```
<property name="timesheetManager">
    <ref bean="timesheetManager" />
</property>
<property name="departmentManager">
    <ref bean="departmentManager" />
</property>
<property name="applicationSecurityManager">
    <ref bean="applicationSecurityManager" />
</property>
```

Приведенные ниже строки кода настраивают командный класс для контроллера `EnterHoursController`:

```
<property name="commandClass">
    <value>com.visualpatterns.timex.model.Timesheet</value>
</property>
```

Остальная часть конфигурации этого контроллера специфична для Spring. Обратите, например, внимание на свойство `validator`, которое является необязательным, но мы будем использовать его для проверки достоверности данных, введенных на экране. Имя `formView` — это название фактической формы представления, а `successView` — это представление, к которому среда Spring должна переадресовать пользователя после успешной подачи формы. Свойство `sessionForm` позволяет нам сохранить тот же экземпляр командного объекта на протяжении сеанса, а не создавать его каждый раз.

```
<property name="formView">
    <value>enterhours</value>
</property>
<property name="successView">
    <value>redirect:timesheetlist.htm</value>
</property>
```

```
<property name="validator">
    <ref bean="enterHoursValidator" />
</property>
```

### Класс ResourceBundle

Еще один элемент конфигурации, который мы должны рассмотреть, связан со строками внутренних и внешних сообщений, а также предназначен и для других целей, как показано далее:

```
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMess
ageSource">
    <property name="basenames">
        <list>
            <value>messages</value>
        </list>
    </property>
</bean>
```

Класс Spring `ResourceBundleMessageSource` полагается на класс JDK `java.util.ResourceBundle`; мы будем использовать его для отображения сообщений о состоянии и об ошибках в файле `messages.properties` (располагающемся в каталоге `timex/src/conf`), который содержит следующие сообщения:

```
typeMismatch.int=Invalid number specified in a numeric field
error.enterhours.missingdepartment=Please select a department
error.login.invalid=Invalid employee id or password
message.enterhours.savesuccess=Timesheet saved successfully
```

В качестве альтернативы среда Spring предоставляет также класс по имени `ReloadableResourceBundleMessageSource`, который применяется для периодической перезагрузки свойств за счет установки параметра `cacheSeconds`. Это может оказаться весьма удобным, когда при разработке приходится часто изменять файл сообщений.

## Поэтапное программирование

Для создания нашего контроллера формы необходим следующий код Java, связанный со Spring. К концу примера экрана Enter Hours мы должны получить следующие файлы (в нашем каталоге `timex/src`):

- `conf/messages.properties`
- `java/com/visualpatterns/timex/controller/EnterHoursController.java`
- `java/com/visualpatterns/timex/controller/EnterHoursValidator.java`
- `java/com/visualpatterns/timex/controller/MinutesPropertyEditor.java`
- `java/com/visualpatterns/timex/view/enterhours.jsp`

### Код контроллера

Давайте приступим к разработке контроллера. Для начала обратите внимание на то, что вместо реализации интерфейса `org.springframework.web.servlet.mvc.Controller`, как мы сделали для контроллера `TimesheetListController`, здесь мы дополним конкретный класс Spring `org.springframework.web.servlet.mvc.SimpleFormController`.

```
public class EnterHoursController extends SimpleFormController
```

Класс `SimpleFormController` реализует интерфейс `Controller`, а также часть иерархии различных абстрактных классов, связанных с контроллером (как мы видели на рис. 7.3). Он также может автоматически переадресовывать пользователя к заданному по умолчанию представлению, в случае ошибки, или к другому представлению (или тому же), в случае успеха. Для этого используются свойства `successView` и `formView`, значения которых устанавливаются в нашем файле `timex-servlet.xml` для компонента Spring `enterHoursController`, как мы видели ранее.

Рассмотрим методы обработки формы, связанные со Spring. Но прежде чем рассмотреть каждый метод, обратим внимание на порядок, в котором эти методы вызываются. На рис. 7.8 представлены три блока. Первый — это, по существу, тот экран, который пользователь видит сначала, когда вводит данные. Второй блок — это экран, который пользователь видит, отправив форму с недопустимыми полями (т.е. когда проверка допустимости не пройдена). Третий, и последний, блок демонстрирует методы, вызываемые при успехе проверки. Теперь рассмотрим код каждого из этих методов.

Первый обсуждаемый здесь метод, `formBackingObject`, возвращает командный объект, который используется для содержания данных, введенных пользователем в поля формы HTML. Обратите внимание на то, что мы выбираем существующую запись `Timesheet` из базы данных, если в контроллер переданы параметры, указывающие, что это операция редактирования, а не добавления, в результате которой создается новый командный объект.

```
protected Object formBackingObject(HttpServletRequest request)
{
    if (request.getParameter(TID) != null
        && request.getParameter(TID).trim().length() > 0)
            return timesheetManager.getTimesheet(Integer.
↪parseInt(request
                .getParameter(TID)), false);

    Timesheet timesheet = new Timesheet();
    Employee employee = (Employee) applicationSecurityManager
        .getEmployee(request);
    timesheet.setEmployeeId(employee.getEmployeeId());
    timesheet.setStatusCode("P");
    timesheet.setPeriodEndingDate(DateUtil.getCurrentPeriodEndingD
↪ate());
    return timesheet;
}
```

**Непосредственная привязка к объекту домена (бизнес-объекту)**
Очевидное преимущество Spring MVC — это способность привязывать поля формы непосредственно к объекту домена (например, `Timesheet`)! Это одно из отличий среды Spring от многих других Web-сред.

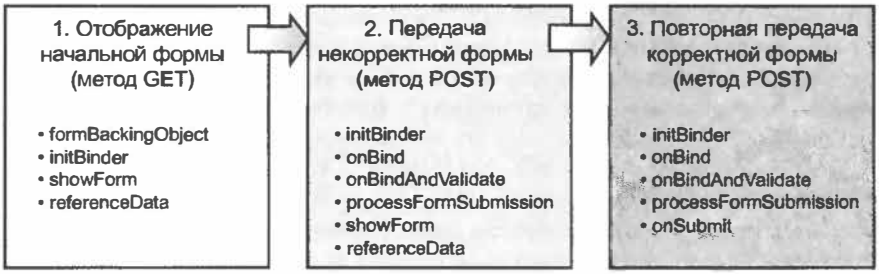



**Рис. 7.8.** Жизненный цикл `EnterHoursController`

Затем следует метод `initBinder`, который предоставляет место для регистрации обсуждаемых вскоре специальных редакторов свойств, как показано далее:

```
binder.registerCustomEditor(int.class, new MinutesPropertyEditor());
```

Метод `referenceData` — это хорошее место для возвращения данных форм, предназначенных только для чтения, например для раскрывающихся списков, как это сделано для списков отделов на экране Enter Hours:

```
model.put("departments", departmentManager.getDepartments());
```

И наконец, но не в последнюю очередь, рассмотрим один из наиболее важных методов нашего класса контроллера, метод `onSubmit`, показанный далее. Как мы видели на рис. 7.8, этот метод вызывается только после того, как все проверки правильности прошли успешно:

```
protected ModelAndView onSubmit(
        HttpServletRequest request,
        HttpServletResponse response,
        Object command,
        BindException errors)
{
    Timesheet timesheet = (Timesheet) command;
    timesheetManager.saveTimesheet(timesheet);
    request.getSession().setAttribute(
        "message",
            getMessageSourceAccessor().getMessage(
                "message.enterhours.savesuccess"));
    return new ModelAndView(getSuccessView());
}
```

Обратите также внимание на следующий код в методе `onSubmit`, который возвращает сообщение об успехе при помощи сеанса HTTP. Это сообщение извлечено из файла `messages.properties` (с использованием ключа `message.enterhours.savesuccess`) и отображено на экране Timesheet List.

Вот и все о классе контроллера. Теперь давайте рассмотрим другие классы, используемые этим контроллером.

### Специальный редактор свойств

Как я упомянул ранее в этой главе, среда Spring затрудняет использование редакторов свойств в стиле JavaBean (т.е. `java.beans.PropertyEditorSupport`).

Мы напишем специальный редактор свойств, класс `MinutesPropertyEditor`, чтобы преобразовать введенные на экране часы в минуты, поскольку именно так разработана наша база данных. Код для этого класса должен быть довольно прост, поскольку он лишь выполняет преобразование минут в часы и наоборот (т.е. умножает или делит на 60).

### Проверка правильности

Наш пример проверки правильности также довольно прост. Главный код действительно находится в методе `validate` этого класса, как показано в следующем отрывке кода:

```
Timesheet timesheet = (Timesheet)command;
if (timesheet.getDepartmentCode() == null ||
    timesheet.getDepartmentCode().trim().length() < 1)
    errors.reject("error.enterhours.missingdepartment");
```

Переменная `error`, показанная здесь, имеет тип `org.springframework.validation.Errors`, который предоставляет несколько методов `reject`. Пример, который я привел здесь, полезен для демонстрации глобальных сообщений на полный экран; я предпочитаю использовать этот метод, а не методы, специфические для полей. Например, один из специфических для полей методов `reject` имел бы следующую сигнатуру: `rejectValue(String field, String errorCode)`.

На рис. 7.8 вы также могли бы заметить метод `onBindAndValidate`. Этот метод имеет следующую сигнатуру:

```
onBindAndValidate(HttpServletRequest request,
                  Object command,
                  BindException errors)
```

Этот метод среда Spring вызывает автоматически, после того как был вызван объект `Validator`. Это прекрасное место для дополнительных проверок правильности, например проверки правильности на основании параметров, представленных запросом HTTP, или проверки правильности базы данных при помощи одного из внедренных классов модели, возможно, для выявления двойных записей в базе данных.

### Код представления и JSP

Теперь, когда мы закончили рассмотрение классов Java для экрана Enter Hours, можем перейти к соответствующему коду представления, расположенному в нашем файле `enterhours.jsp`. Мы рассмотрим здесь несколько отрывков.

Первый интересный блок кода, предназначенный для отображения сообщений об ошибках, находится в нашем классе `EnterHoursValidator`, как показано далее:

```
<spring:bind path="command.*">
    <c:if test="${not empty status.errorMessages}">
        <c:forEach var="error" items="${status.errorMessages}" >
            <font color="red"><c:out value="${error} "
escapeXml="false"/></font><br/>
        </c:forEach>
    </c:if>
</spring:bind>
```

Поскольку это первый раз, когда встречается дескриптор `spring:bind`, позвольте мне сказать о нем несколько слов.

Ключевой класс, лежащий в основе библиотечного дескриптора Spring `bind`, — это класс `org.springframework.web.servlet.support.BindStatus`. Данный дескриптор позволяет связывать поля формы HTML с командным объектом (в данном случае, `Timesheet`). Но он предоставляет также доступ к специальной переменной `status` (*состояние*). Объект `status` содержит несколько перечисленных ниже атрибутов, которые применяются в коде JSP:

- `status.value`. Значение данного атрибута в командном объекте
- `status.expression`. Имя данного атрибута в командном объекте
- `status.error`. Логический флаг, указывающий на наличие ошибки
- `status.errorMessage`. Сообщение об ошибке, специфическое для поля
- `status.errorMessages`. Сообщения об ошибке, глобальные для данного представления
- `status.displayValue`. Строковое значение, подходящее для отображения с использованием метода `toString`

Теперь рассмотрим, как осуществляется привязка полей. Приведенный ниже код демонстрирует, как переменная JSP/HTML `departmentCode` связывается с соответствующей переменной в нашем командном объекте (т.е. `Timesheet.departmentCode`).

```
<spring:bind path="command.departmentCode">
```

Это действительно вся часть кода `enterhours.jsp`, которую я не объяснил. Дело в том, что мы уже рассматривали ранее в этой главе подобный код для экрана Timesheet List (например, код цикла с использованием дескриптора JSTL `forEach`). Мне очень жаль, что я не могу рассказать вам больше о библиотеке дескрипторов привязки Spring, но, как я упомянул ранее, эта библиотека довольно проста, но то, что вы можете сделать с ее помощью, просто невероятно.

## Привязка к специальным командным объектам (небизнес-объектам)

Одним из ключевых преимуществ MVC Spring является способность связывать поля формы HTML непосредственно с вашим объектом домена. В среде Spring эти объекты называются *командными* (command), возможно, из-за схемы проектирования "Command", которая подразумевает инкапсуляцию запроса в объекте. Еще один способ описания концепции командных объектов заключается в том, что его можно рассматривать как *объект формы* (form object), поскольку он способен содержать все значения, введенные в форму HTML. Однако поскольку мы можем связывать наши поля формы HTML непосредственно с нашими бизнес-объектами или хранить в этом объекте другие данные, термин *командный* подходит больше.

Для экранов приложения Time Expression мы используем непосредственное связывание с нашим объектом домена `Timesheet`. Однако вы всегда имеете возможность создать специальный класс `Command`, который мог бы, например, дополнять или содержать класс `Timesheet` и предоставлять некоторые дополнительные методы. Например, я работал над проектом, где должен был транслировать и дезассемблировать

объект `java.util.Date`, поскольку форма HTML имела отдельные раскрывающиеся списки для месяца, даты и года. В этом случае я использовал такие методы, как `assembleDate` и `disassembleDate`, в специальном командном классе.

Существует два способа использования специального командного класса. Например, мы могли сделать нечто вроде следующего:

```
public class TimesheetCommand extends Timesheet
```

Вы можете также осуществить привязку непосредственно при помощи методов установки и получения значения (setter/getter) нашего бизнес-объекта, но это дополнительные методы, они не всегда необходимы. Кроме того, чтобы создать специальный командный класс, вы должны были бы определить его в файле `timex-servlet.xml`, а также организовать создание и возвращение объекта этого типа в методе `formBackingObject`. Другой подход заключается в использовании класса `TimesheetCommand`, содержащего ссылку на объект `Timesheet`. Например, этот класс мог бы иметь следующий конструктор:

```
public TimesheetCommand(Timesheet timesheet) {...}
```

Применяя этот подход, вы связали бы поля формы HTML с объектом `Timesheet`, используя форму записи, подобную следующей:

```
command.timesheet.minutesMon
```

Одна из проблем, с которой вы столкнетесь, используя этот подход, связана с проверкой JavaScript, поскольку JavaScript запутывается в точках, содержащихся в именах полей HTML. Например, имя `command.timesheet.minutesMon` преобразуется в имя поля ввода текста HTML `timesheet.minutesMon`, если для заполнения мы используем форму `${status.expression}`.

## Файл `DateUtil.java`

Еще один интересный файл, `DateUtil.java`, предоставляет некоторые вспомогательные методы. Например, наш класс `enterHoursController` использует один из этих методов в своем методе `formBackingObject`:

```
timesheet.setPeriodEndingDate(DateUtil.getCurrentPeriodEndingDate());
```

## Директивы библиотеки дескрипторов JSP

Единственное, что я не объяснил до сих пор подробно, — это следующие строки кода, которые вы могли бы заметить в наших файлах JSP:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"
↳ %>
```

Это директивы, которые необходимы перед использованием библиотеки дескрипторов JSP. Более подробная информация об этой и других возможностях JSP приведена на Web-сайте `java.sun.com`.

# Представления без контроллеров

Иногда бывают ситуации, когда контроллер не нужен или его не хочется писать. Предположим, что мы хотим реализовать экран справки для приложения Time Expression. Мы хотим, чтобы этот экран справки был доступен как /help.htm, а его реальный файл (help.jsp) был скрыт в /WEBINF/jsp. В данном случае мы сначала определили бы компонент urlFilenameViewController в файле timexservlet.xml следующим образом:

```
<bean id="urlFilenameController"
class="org.springframework.web.servlet.mvc.UrlFilenameViewController"
↳/>
```

Теперь мы можем ссылаться на компонент urlFilenameController при связывании обработчика (например, компонент urlMap в файле timex-servlet.xml):

```
<prop key="/help.htm">urlFilenameController</prop>
```

# Перехватчики обработчиков Spring

До сих пор мы разрабатывали наши экраны Timesheet List и Enter Hours, не заботясь об аутентификации. Однако одним из основных требований, выдвинутых в главе 2, "Простое приложение: сетевая система учета рабочего времени", было то, что сотрудники могут просматривать только собственные табели учета рабочего времени. Таким образом, нам понадобятся функциональные возможности экранов Sign In (Регистрация) и Sign Out (Отмена регистрации).

Среда Spring поддерживает концепцию перехватчиков для разработки Web-приложений; она позволяет перехватить запросы HTTP. Мы будем использовать эту возможность для организации аутентификации в приложении Time Expression.

Для реализации регистрации и ее отмены нам понадобится создать в каталоге src/java/com/visualpatterns/timex следующие файлы:

- controller/HttpRequestInterceptor.java
- controller/SignInController.java
- controller/SignInValidator.java
- controller/SignOutController.java
- util/ApplicationSecurityManager.java
- util/DateUtil.java
- view/signin.jsp

## Аутентификация приложения Time Expression

Аутентификация в приложении Time Expression производится для всех требующих аутентификации запросов HTTP и осуществляется при их поступлении в наш класс перехватчика, HttpRequestInterceptor.java. Приведенный ниже отрывок кода демонстрирует, как может быть обработан перехваченный запрос:

```
public class HttpRequestInterceptor extends HandlerInterceptorAdapter
{
    private ApplicationSecurityManager applicationSecurityManager;
```

```
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler)
            throws Exception
    {
        Employee employee =
            (Employee)applicationSecurityManager.getEmployee(req
↳uest);
        if (employee == null)
        {
            response.sendRedirect(this.signInPage);
            return false;
        }

        return true;
```

Обратите внимание на применение здесь класса `ApplicationSecurityManager` (несколько ссылок на него встречалось ранее в этой главе). Законченный код этого класса должен быть довольно прост, по существу, он предоставляет методы для установки, получения и удаления атрибутов сеанса HTTP, именованного пользователем (одним из наших объектов домена типа `Employee`), как демонстрирует следующий отрывок кода, который устанавливает этот атрибут:

```
public static final String USER = "user";
public void setEmployee(HttpServletRequest request, Object employee)
{
    request.getSession(true).setAttribute(USER, employee);
}
```

Класс `SignInController` проверяет *регистрационное имя* (login), а также устанавливает объект домена `Employee`, используя метод `ApplicationSecurityManager.setEmployee`, как показано далее:

```
Employee formEmployee = (Employee) command;
Employee dbEmployee = (Employee) command;
if ((dbEmployee = employeeManager.getEmployee(formEmployee
        .getEmployeeId())) == null)
    errors.reject("error.login.invalid");
else
    applicationSecurityManager.setEmployee(request, dbEmployee);
```

Наш класс `SignOutController` отменяет регистрацию пользователя, удаляя атрибут `Employee` из сеанса, как показано далее:

```
applicationSecurityManager.removeEmployee(request);
```

**Примечание**

Наше приложение использует небольшой файл `index.jsp` как файл приветствия; он помещается в наш каталог `src/web` и переадресует запрос по URL `signin.htm`, как показано далее:

```
<c:redirect url="signin.htm"/>
```

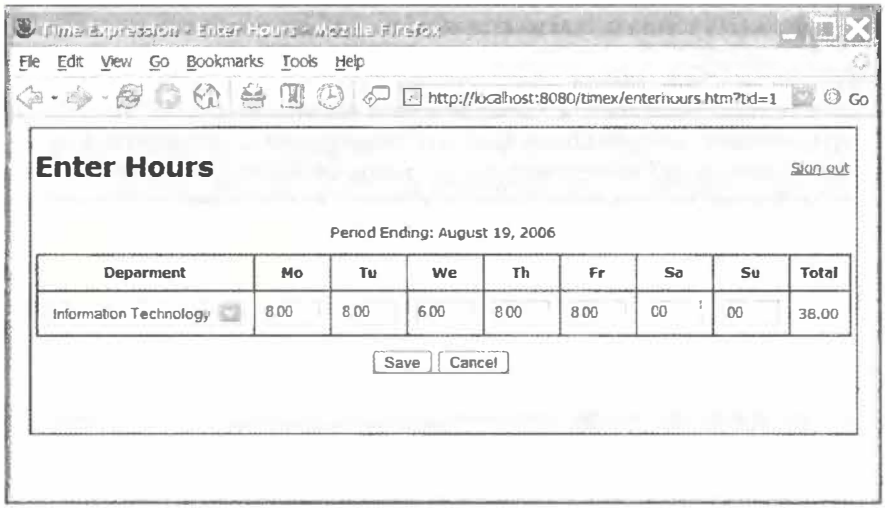



**Рис. 7.11.** Экран Enter Hours

**Личное мнение: проектирование и программирование с использованием интерфейсов**

Справочная документация Spring и много статей о Web рекомендуют проектирование и программирование с использованием интерфейсов. Среда Spring поддерживает компоненты и на основе интерфейсов, и на основе классов.

Как вы могли бы заметить, при проектировании классов приложения Time Expression я не использовал интерфейсы Java. Это связано с моей философией, когда и где использовать интерфейсы.

Позвольте мне для начала сказать, что мне очень нравится программировать с использованием интерфейсов Java! Однако, подобно многим другим людям, перешедшим на систему EJB несколько лет назад, я вижу многих людей, переходящих на интерфейсы. Поэтому я выскажу свои мысли по этому вопросу; вы можете соглашаться с ними или не соглашаться. Для начала я хотел бы привести вам небольшую статью на тему проектирования и программирования с использованием интерфейсов.

Я использовал интерфейсы начиная с 1996 года и сохранил любовь к этой концепции до сих пор. В 1997 году я разработал программное обеспечение для резервирования по имени BackOnline (`http://visualpatterns.com/backonline/`), исключительно на Java. Этот продукт, упомянутый в нескольких известных журналах, был удостоен премии Best Client на JavaOne; он был даже номинирован Скоттом Мак-Нили (Scott McNealy) (CEO, Sun Microsystems) на премию Computerworld-Smithsonian. Продукт BackOnline имеет архитектуру клиент/сервер; сервер многопотоковый и многопользовательский, получает файлы и сохраняет их, используя класс реализации интерфейса по имени `DataStore`. Интерфейс `DataStore` имеет методы, подобные FTP, такие как `get`, `put`, `open`, `close` и так далее; их было бы трудно реализовать в конкретных классах. Программное обеспечение BackOnline (которое больше не продается) поставлялось укомплектованным двумя стандартными классами реализации интерфейса `DataStore`, а именно: `DataStoreFileSystem` и `DataStoreJDBC` (полностью определенные имена классов реализации были определены в файле конфигурации и динамически загружались во время выполнения). Класс `DataStoreFileSystem`, по существу, использовал пакет `java.io` для сохранения файлов в локальной файловой системе. Класс `DataStoreJDBC` использовал JDBC для хранения содержимого файла в качестве *двоичных объектов большого размера* (Binary Large Object — BLOB) в реляционной базе данных.

Для интерфейса `DataStore` предоставлялась документация Javadoc и дополнительная техническая документация, поэтому сотрудничающие с моей компанией *провайдеры услуг Интернета* (Internet Service Provider — ISP) и производители программных продуктов, ориентирующихся на OEM (Original Equipment Manufacturer — изготовитель комплектного оборудования), в случае необходимости могли создавать собственные специальные реализации. Например, ISP мог бы воспользоваться такими средствами базовой операционной системы, как дополнительные права для доступа к файлам.

Выбор интерфейсов для приложения BackOnline был вполне очевиден. Кроме того, я много раз убеждался, что интерфейсы хорошо подходят для низкоуровневых API, таких как я использовал для приложения BackOnline, а также для таких сред, как JDK или Spring Framework (например, `java.util.Collections` или `java.sql.Connection`). Кроме того, интерфейсы незаменимы, если существует вероятность изменения основной реализации (например, службы регистрации, аутентификации и функциональных возможностей, специфичных для операционной системы). Безусловно, с дистанционными технологиями (например, EJB) вы не имеете другого выбора, кроме использования интерфейсов. Для бизнес-приложений, в отличие от меньших проектов, я полагаю, необходима реализация только объектов домена (бизнес-объектов) или сервисных объектов (таких, как класс `TimesheetManager` приложения Time Expression). Кроме того, нет смысла иметь интерфейсы для объектов домена (таких, например, как `Timesheet.java`).

Создание отдельного файла интерфейса для каждого класса реализации приводит, по моему мнению, к слишком большому количеству ненужной дополнительной информации. В больших проектах это может создать слишком большое количество дополнительных файлов `.java` (и `.class`), причем без особого значения. С другой стороны, иногда использование интерфейсов имеет смысл. Например, в главе 2, "Простое приложение: сетевая система учета рабочего времени", мы обсуждали многопользовательские типы (роли) для приложения Time Expression, такие как `Employee`, `Manager` и `Executive`. Они легко могли быть созданы как конкретные классы, реализующие интерфейс по имени `Person` или `Role`. С другой стороны, с учетом общего поведения этих объектов, абстрактный класс также имел бы не много смысла, поскольку общие методы могли быть унаследованы из базового (родительского) класса (например, класса `Person`).

Таким образом, с учетом реальных возможностей, вы должны использовать интерфейсы, однако не стоит делать это только потому, что так проповедует некая книга или статья. Кроме того, вы не должны считать ошибкой отказ от использования интерфейсов для любого и каждого конкретного класса, который вы создаете. Уделяйте больше внимания проектированию вашего приложения, например, четкости разделения слоев, хорошему проекту базы данных, простоте кода, корректному использованию схем архитектуры и т.д. Надеюсь, я не испугал вас интерфейсами, поскольку мое намерение не таково. Моя точка зрения: все возможности следует использовать с умом и в соответствующих обстоятельствах.

# Новые библиотеки дескрипторов Spring Framework 2.0

На момент написания этой книги группа Spring готовилась к выпуску дополнительных библиотек дескрипторов, чтобы еще более упростить работу Spring с JSP. Однако проектирование этих новых библиотек дескрипторов все еще не закончено, поэтому я не смог изучить их в достаточной степени.

# Несколько слов о технологиях Spring Web Flow и API Portlet

Вас могли бы заинтересовать две дополнительные технологии пользовательских интерфейсов Spring, если вам необходимы предоставляемые ими средства.

### Технология Spring Web Flow

Эта технология основана на концепции, подобной Spring Web MVC Framework. По существу, она предоставляет экранные функциональные возможности, подобные мастеру, но предназначенные для реализации коммерческой деятельности. Хорошим примером подобных приложений являются сетевые магазины, такие как `amazon.com`, которым требуется, чтобы вы прошли несколько экранов, прежде чем транзакция завершится. Этот тип функциональных возможностей требует управления состоянием сеанса, что позволяет вернуться к предыдущему экрану, не теряя ранее введенную информацию. Это похоже на облегченный тип функциональных возможностей Web Flow. Однако наше приложение Time Expression не требует таких возможностей и не является хорошим примером для технологии Spring Web Flow.

Хотя технология Web Flow не рассматривается в этой книге, с учетом ее контекста, я настоятельно рекомендую вам серьезно изучить эту технологию, если ваши требования соответствуют данному типу функциональных возможностей.

### Технология Spring Portlet API

Данная технология — это новое дополнение Spring 2.0. Оно реализует спецификацию JSR-168 Portlet Specification, определенную на Web-сайте сообщества Java (Java Community Process — JCP) (`http://www.jcp.org/en/jsr/detail?id=168`). Согласно этой спецификации, Portlet API применяется как "портал компьютерной адресации, агрегации, персонализации, представления и защиты". Еще один взгляд на это заключается в том, что *портлет* (portlet) является частью *портального* (portal) Web-сайта, который способен содержать несколько портлетов. Портлеты отличаются от *сервлетов* (servlet), они не переадресовывают и не перенаправляют запросы от браузера или к нему; вместо этого, ими управляет контейнер портлетов.

Если вас заинтересовали эти API, посетите Web-сайт JCP. Кроме того, вас может заинтересовать Apache Pluto, рекомендуемая реализация для Portlet API.

## Резюме

В этой главе мы сделали следующее.

- Рассмотрели преимущества применения Spring MVC
- Подробно изучили Spring Web MVC Framework
- Создали три экрана приложения Time Expression с использованием Spring MVC: один без формы и два с формами

Мы рассмотрели достаточно много материала в этой главе, но это не все о среде Spring. В следующих главах мы рассмотрим дополнительные аспекты среды Spring Framework, включая:

- дополнение IDE Spring для Eclipse;
- планирование задач;
- передачу электронной почты.

Тем не менее, если вы хотите изучить среду Spring подробнее, обратите внимание на пример JPetstore и справочную документацию. Оба входят в комплект поставки программного обеспечения Spring. В следующей главе рассмотрим систему Eclipse, которая полностью изменит стиль работы в этой книге! Другими словами, перейдем от программирования в командной строке к интегрированной среде разработки (IDE), которая упрощает настройку, программирование, проверку и отладку модулей. Короче говоря, это гибкая разработка Java!

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, и конкурирующим технологиям содержится на следующих Web-сайтах:

- Apache Jakarta Tapestry `http://jakarta.apache.org/tapestry/`
- Apache Jakarta Turbine `http://jakarta.apache.org/turbine/`
- Apache Struts `http://struts.apache.org/`
- Apache Tapestry `http://jakarta.apache.org/tapestry/`
- Apache Tomcat `http://tomcat.apache.org/`
- Атрибут конфигурации `antiJARLocking` и `antiResourceLocking` Apache Tomcat `http://tomcat.apache.org/tomcat-5.5-doc/config/context.html#Standard%20Implementation`
- Apache Tomcat и задачи `http://tomcat.apache.org/tomcat-5.0-doc/catalina/docs/api/org/apache/catalina/ant/package-summary.html`
- Apache Velocity `http://jakarta.apache.org/velocity/`
- FreeMarker `http://www.freemarker.org/`
- Вопросы JavaServer `http://java.sun.com/j2ee/javaserverfaces/`
- Контейнер сервлета Jetty `http://jetty.mortbay.org/jetty/`
- Mock Objects `http://mockobjects.com`
- OpenSymphony WebWork `http://www.opensymphony.com/webwork/`
- Форум обсуждения Spring `http://forum.springframework.org/`
- Spring Framework `http://springframework.org`
- Исходный MVC `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`

# 8

# Феномен Eclipse!

Выпуск 1, Неделя 6, Итерация 3

**Сьюзен:** *С днем рождения, Стив. Каково это почувствовать себя на год старше?*

**Стив:** *О, это великолепное чувство, и не только, потому что это мой день рождения, а также и потому, что мы развернули наше рабочее приложение. Радж решил проблему управления транзакциями. Кроме того, система Eclipse, которой я ждал так долго, поступила сегодня вечером. Вы, парни, должны опробовать ее, я полагаю, она вам понравится.* (c) Visual Patterns, Inc.

До сих пор в книге мы работали с командной строкой (при запуске Ant и JUnit, например). Работу в командной строке можно в некотором роде считать противоположностью *гибкой разработки* (agile development). Однако я сторонник того, что сначала необходимо уяснить фундаментальные принципы, другими словами, необходимо изучить то, что происходит за кулисами.

Кроме того, большинство разработчиков предпочитают именно командную строку, причем работают в ней весьма эффективно. Поэтому я хотел и вас познакомить сначала с этим подходом. Теперь пришло время отбросить командную строку и перейти к SDK Eclipse!

SDK Eclipse – это весьма полезная и высоко эффективная *интегрированная среда разработки* (Integrated Development Environment – IDE) для программирования в Java, и не только, как будет продемонстрировано вскоре. Комплект SDK Eclipse весьма надежен сам по себе, но что на самом деле делает его столь мощным средством, так это то, что он является также платформой с однозначными стандартами для разработки дополнений и небольших приложений, запускаемых внутри Eclipse. Для Eclipse доступны сотни дополнений. Например, на момент написания этой книги запрос на `google.com` для слов "eclipse" и "plug-in" возвращал миллионы ссылок!

Если вы уже используете SDK Eclipse или использовали другие продукты типа браузера Mozilla Firefox (`http://mozilla.com`), то оцените мощь дополнений. По моему мнению, парадигма дополнений переводит реализацию с открытым исходным кодом на новый уровень, прежде всего, потому, что такие организации, как Eclipse Foundation, могут не тратить своих усилий на решение проблем платформы. Но сообщество в целом улучшает и расширяет платформу (с использованием дополнений, например).

В этой главе мы подробно рассмотрим SDK Eclipse, а также различные дополнения, полезные при разработке в Java.

Это особенная глава, поскольку для ее написания мне пришлось весьма глубоко изучить Eclipse; причем моя точка зрения на Eclipse при этом изменилась. Я поделюсь с вами информацией об истории Eclipse Foundation, платформе Eclipse, огромном количестве дополнений и чрезвычайной активности сообщества Eclipse. Я надеюсь, что к концу этой главы вы скажете об Eclipse только одно – *это феноменально!*

# Что рассматривается в этой главе

В этой главе мы подробнее рассмотрим систему Eclipse и ее применение для нашего типового приложения, Time Expression. К времени, когда эта глава будет закончена, мы соберем все технологии, описанные в этой книге до сих пор, под эгидой Eclipse, используя ряд дополнений. В этой главе мы учим следующее.

- Организация Eclipse Foundation
- Платформа и проекты Eclipse
- Концепции SDK Eclipse
- Установка Eclipse
- Настройка Eclipse для приложения Time Expression
- Дополнение Java Development Tools (JDT)

- Проект Eclipse Web Tools Platform Project (для поддержки Tomcat, редактирования JSP и другой поддержки JEE)
- Применение Eclipse для приложения Time Expression (запуск Tomcat и подключение к HSQLDB, например)
- Поддержка группы Eclipse, использующей CVS
- Каталоги бесплатных и коммерческих дополнений
- Советы и приемы Eclipse
- Результаты 30-минутного сравнения IntelliJ и NetBeans

**Почему SDK, а не IDE**

Под *комплектом разработчика программного обеспечения* (Software Development Kit — SDK) обычно подразумевают *интерфейс прикладных программ* (Application Programming Interface — API), а *интегрированная среда разработки* (Integrated Development Environment — IDE) обычно означает систему программирования и отладки. В документации IDE Eclipse упоминается как SDK Eclipse (возможно, потому, что это также и API для разрабатываемых дополнений). Однако лучше употреблять термин IDE Eclipse, как это делают многие.

# Организация Eclipse Foundation

Позвольте мне начать с цитирования некоторого материала, взятого непосредственно с Web-сайта Eclipse (`eclipse.org`), поскольку это даст вам некоторое представление об организации Eclipse Foundation, ее истории, а главное, о ее сотрудниках.

"Eclipse – это сообщество реализации с открытым исходным кодом, чьи проекты ориентированы на поддержку независимых от производителей открытых платформ разработки и сред выполнения приложений, предназначенных для создания программного обеспечения. Eclipse Foundation – это некоммерческая корпорация, сформированная для создания, развития, содействия и поддержки платформы Eclipse Platform, а также для обеспечения сообщества реализации с открытым исходным кодом и сообществ дополнительных продуктов, возможностей и услуг."

Web-сайт продолжает:

"Первоначально, в ноябре 2001 года, консорциум Board of Stewards (`eclipse.org`) было сформировано такими индустриальными лидерами, как Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft и Webgain. К концу 2003 года этот консорциум насчитывал свыше 80 участников."

Изначально система Eclipse была разработана компанией Object Technology International (OTI), которая впоследствии была приобретена корпорацией IBM. Некоторое время спустя, корпорация IBM пожертвовала технологию Eclipse (ценой приблизительно 40 миллионов долларов) сообществу реализации с открытым исходным кодом, а также мобилизовала упомянутые выше корпорации на совместную разработку высоко интегрированных продуктов для этой платформы (в виде дополнений).

Организация Eclipse Foundation подобна организации Apache Foundation, которая предоставляет инструментальные средства с открытым исходным кодом. Но между ними есть одно существенное различие, инструменты Eclipse имеют, в основном, графический характер, в отличие от инструментов Apache, которые, как правило, ориентированы на текст, например серверы, API и инструменты (Ant и Tomcat, например). Я нахожу это полезным новаторством. Производители инструментов Java начинают наконец осознавать это (см. раздел "Личное мнение: инструменты разра-

ботки GUI, сражение началось!" далее в этой главе). Я не понимаю, почему для этого потребовалось так много времени, для других языков программирования инструменты разработки и отладки, обладающие GUI, были созданы несколько десятилетий назад.

**Личное мнение: Java против Microsoft**

Эта глава не о соревновании или сражении Java и Microsoft, но я должен немного отвлечься на это. К корпорации Microsoft я испытываю и любовь, и ненависть, поскольку, с одной стороны, я ненавижу способы, которыми они пытались пускать под откос Java несколько лет назад, с другой стороны, мне нравится большинство программных продуктов, которые они производят, например Microsoft Windows Media Center Edition. Сказав это, позвольте мне также заявить, что Eclipse — это, на мой взгляд, первый инструмент сообщества Java, способный соперничать с таким программным обеспечением, как Visual Studio от Microsoft. Это то, что мне нравится в Eclipse, по сравнению с другими продуктами Java.

Так что же делает Eclipse таким мощным (и не только) инструментом разработки GUI? Другие продукты, такие как IntelliJ от JetBrain и NetBeans от Sun Microsystem, тоже хороши. Однако что на самом деле делает Eclipse феноменально популярным, так это огромное количество довольно сложных дополнений, доступных для этой платформы, причем как коммерческих, так и с открытым исходным кодом. Благодаря наличию сотен бесплатных и коммерческих дополнений, сообщество Eclipse стремительно растет. Вот лишь один из подобных примеров, сайт `myeclipseide.com` предоставляет ежегодную подписку и обеспечивает место под солнцем любому дополнению, которое может понадобиться вам для корпоративной программной разработки. Кроме того, консорциум Eclipse насчитывал в 2003 году больше 80 участников и быстро пополнился такими крупными производителями, как Borland, Rational Software, Red Hat, SUSE и TogetherSoft. Фактически, в начале 2005 года корпорация Borland объявила, что ее флагманский программный продукт разработки Java, JBuilder, был основан на платформе Eclipse.

Два факта, которые могли бы заинтересовать вас, связаны с Вардом Каннингемом и Эрихом Гамма. Вард, основатель Вики, соавтор карточек CRC и человек, внесший существенный вклад в экстремальное программирование, оставил группу Patterns and Practices Team в корпорации Microsoft и присоединился к организации Eclipse Foundation. Эрих — соавтор популярной книги *Design Patterns* и участник проверки среды JUnit. Он один из ключевых специалистов, включенных в проект JDT Project. Интересно знать, что фонд Eclipse способен принимать на работу таких весьма уважаемых, инновационных лидеров нашей индустрии.

Короче говоря, сообщество Eclipse росло очень быстро, однако в будущем, на мой взгляд, его ожидает рост в геометрической прогрессии!

# Платформа и проекты Eclipse

Платформа Eclipse – это межплатформенная реализация с открытым исходным кодом и расширяемым встроенным IDE, использующая язык Java. По существу, платформа Eclipse – это среда, предоставляющая набор служб, которыми другие дополнения могут пользоваться (как показано на рис. 8.1). Каждое дополнение разработано для той же платформы, которая объединяет их в набор высокоинтегрированных инструментов.

**Eclipse: объединенная панель инструментов**

Платформа Eclipse, объединенная с большим количеством высокоинтегрированных дополнений, по существу, служит объединенной программной средой разработки. Например, существуют дополнения для схем UML, программирования, отладки, управления базами данных, проверки модулей, управления сервером приложений, документирования и многого другого. Таким образом, Eclipse можно рассматривать как панель инструментов, аналогичную набору инструментов плотника, содержащему множество типов инструментов, необходимых для выполнения всех работ.

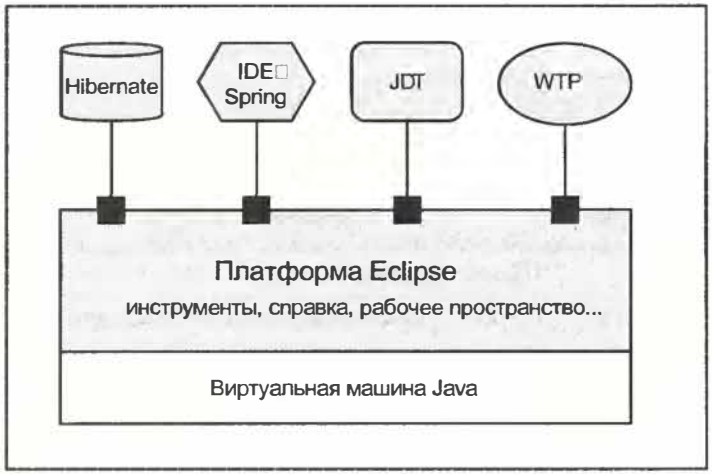



**Рис. 8.1.** Платформа Eclipse и некоторые типовые дополнения (например, JDT)

Среда Eclipse поставляется в комплекте с *инструментами разработки Java* (Java Development Tools – JDT), которые представляют для нас большой интерес, учитывая характер этой книги. Однако Eclipse не останавливается на Java; доступны дополнения для разных языков, включая HTML, C/C++, COBOL и Eiffel. Кроме того, вы найдете дополнения стороннего производителя для других языков, таких как PERL, PHP и Ruby on Rails.

Вы можете также разрабатывать собственное специальное дополнение для расширения Eclipse. Хотя я лично не делал этого, другие, имеющие такой опыт, утверждают, что это не так уж и трудно, поскольку вы разрабатываете дополнения на языке Java. Кроме того, об Eclipse, как правило, говорят в контексте инструментов разработки, а дополнения разрабатывают для функциональных возможностей, отличных от программирования, например для контроля содержимого и других целей.

### Задачи платформы Eclipse

Давайте рассмотрим задачи платформы Eclipse, поставленные организацией Eclipse Foundation. Знать их, пожалуй, стоит, поскольку это поможет связать все вместе, когда мы перейдем в этой главе к рассмотрению обширных функциональных возможностей, предоставляемых Eclipse. Вот некоторые из задач Eclipse.

- Обеспечить надежную платформу для высокоинтегрированных инструментов разработки приложений
- Обеспечить просмотр или редактирование содержимого любого типа (например, код Java, JSP, XML, C/C++, документы Word и т.д.).
- Привлечь большое сообщество разработчиков, которые создадут дополнения для этой платформы

### Проекты Eclipse

Чтобы дать вам лучшее представление о росте области действия общества Eclipse Foundation, я привожу табл. 8.1, демонстрирующую состояние проектов Eclipse, нахо-

дящихся в процессе реализации, на момент написания этой книги. Обратите внимание на то, что за каждым проектом наблюдает *комитет по управлению проектом* (Project Management Committee – PMC). Каждый проект может быть разделен на подчиненные проекты со своим лидером во главе. Кроме того, каждый подчиненный проект, в свою очередь, может состоять из одного или нескольких компонентов.

**Таблица 8.1. Проекты Eclipse (организованные по темам) на момент написания этой книги**

| Тема | Проект (проекты) |
|---|---|
| Разработка приложений | Среда управления жизненным циклом приложения |
| | Инфраструктура разработки, управляемая моделью |
| | Среда связи Eclipse |
| | Проект Buckminister Component Assembly |
| | Платформа SDK Eclipse |
| | Отказоустойчивые инструменты разработки API (DaliORM) EJB 3.0/Java |
| | Ориентированный на задачи UI (проект Mylar) |
| | Инструменты параллельной разработки |
| | Проект Business Intelligence and Reporting Tools (BIRT) |
| | Проект Voice Tools |
| | Проект Eclipse Web Tools Platform |
| Редакторы | Среда графических редакторов (Graphical Editor Framework – GEF) |
| | Визуальный редактор (Visual Editor – VE) |
| | Проект Eclipse Web Tools Platform (HTML, JSP) |
| Моделирование | Порождающий преобразователь модели |
| | Среда графического моделирования |
| | Среда моделирования EMF |
| | UML2 – реализация UML 2.0 на базе EMF |
| Производительность и проверка | Проект Eclipse Test and Performance Tools Platform |
| Языки программирования | COBOL |
| | Проект AspectJ Development Tools |
| | C/C++ IDE |
| | Проект Photran (Fortran) |
| | Инструменты разработчика Java (Java Development Tools – JDT) |
| | Проект Eclipse Web Tools Platform (Java/JSP) |

# Концепции SDK Eclipse

Когда люди впервые используют среду Eclipse, они зачастую находят ее немного сложной, из-за непонимания концепций. Со мной было то же самое. Однако я быстро пришел к выводу, что мне достаточно уяснить только три из пяти фундаментальных концепций Eclipse, чтобы немедленно овладеть этим инструментом. Итак, давайте рассмотрим их здесь. Обратите внимание на то, что я преднамеренно привел здесь лишь краткие описания, поскольку мы рассмотрим эти концепции, когда практически применим Eclipse для приложения Time Expression.

**Оговорка: снимки экранов Eclipse в этой главе**

Почти все снимки экранов SDK Eclipse приведены в этой главе с разрешением 800×600 ради удобочитаемости. Однако большинство разработчиков предпочитают более высокое разрешение, например 1024×768. Лично я использую разрешение 1600×1050. Следовательно, снимки экранов здесь не совсем точно соответствуют реальным экранам Eclipse, поскольку я не имел возможности показать, как можно было бы *фактически* использовать Eclipse, т.е. с большим количеством редакторов и представлений, находящихся на экране одновременно.

## Рабочее пространство

Прежде чем обсудить, насколько применим термин *рабочее пространство* к Eclipse, сначала рассмотрим эту концепцию.

В самом простом случае *рабочее пространство* (workspace) — это каталог для ваших проектов. По существу, это верховный (родительский) каталог, в котором находятся ваши файлы, связанные с проектом (например, файлы `.java`). У себя я использовал каталог `C:\anil\rapidjava`, как будет продемонстрировано в главе далее. Таким образом, все, что находится в этом каталоге (например, каталог `timex/` и каталоги внутри него), считается частью того же самого рабочего пространства.

## Инструменты, компоновки, редакторы и представления

Инструментальные средства Eclipse — это первое, что мы замечаем, когда запускаем Eclipse. Другими словами, это, по существу, платформа Eclipse, плюс некоторые простые функциональные возможности, такие, например, как управление проектом. Фактическая работа, такая, например, как просмотр и редактирование, осуществляется дополнениями (например, JDT).

К *инструментам* (workbench) относится набор соответствующих редакторов и представлений; этот специфический для конкретной задачи набор называется *перспективой* (perspective) или *компоновкой*. Рис. 8.2 демонстрирует, как пользователь может переключаться между компоновками, используя единственный щелчок мышью или комбинацию клавиш (например, <Ctrl+F8>).

### Компоновки

*Компоновка* (perspective) — это набор представлений и редакторов, расположенных в том порядке, какой вам нравится. Инструменты могут располагаться в нескольких компоновках, но в одно и то же время может быть видима только одна из них; кстати, мы можем быстро переключаться между компоновками, щелкнув на кнопке или используя комбинацию клавиш <Ctrl+F8> (<COMMAND+F8> на Mac OS X). Компоновки предоставляют весьма удобный способ упорядочить по вашему выбору представления

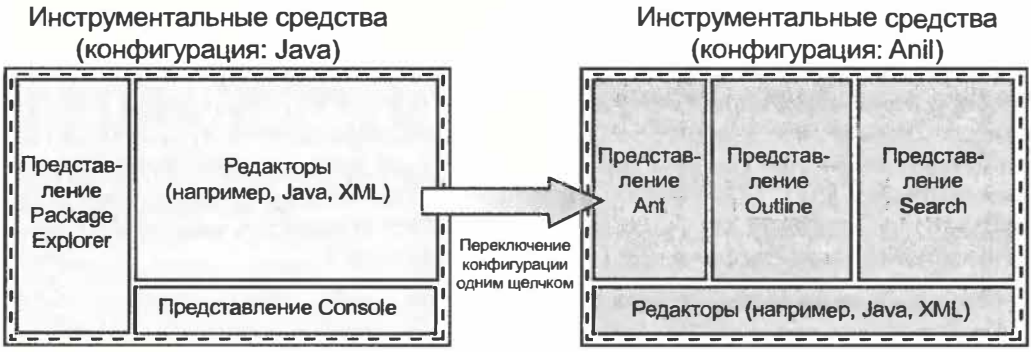



**Рис. 8.2.** Схематическое представление SDK Eclipse

и редакторы для выполнения определенных задач. Например, в компоновке Debug (Отладка) желательно иметь представление Breakpoints (Контрольные точки), тогда как в представлении Java Browsing (Просмотр кода Java) это не нужно. Обратите внимание на то, что представления можно размещать и как фиксированные окна (плиткой или каскадом), и как свободные.

Существует несколько предопределенных компоновок, например Java, Java Browsing, Debug и др. Кроме того, мы можем создавать собственные компоновки. Например, я предпочитаю создавать собственные компоновки, изменяя существующие (например, Java), добавляя и удаляя представления, а затем сохраняю полученные компоновки под именем Anil (я понимаю, это не очень оригинально, однако такое имя позволяет избежать конфликта с именами компоновок в пакетах). На рис. 8.3 представлена моя собственная компоновка Eclipse, а на рис. 8.4 — предопределенная компоновка Debug.

### Редакторы и представления

Редакторы, как вы, вероятно, догадались, позволяют нам открывать, редактировать, сохранять и закрывать файлы. К концу этой главы мы будем иметь редакторы для обычного текста, файлов `.java`, `.jsp`, `.xml` и др.

Представления дополняют редакторы, предоставляя доступную только для чтения информацию, обычно, о файле, редактируемом в редакторе. Например, представление Outline (Иерархическое представление) предоставляет список методов редактируемого файла Java. Однако для файла XML вы увидите различные элементы XML и атрибуты редактируемого файла.

Платформа Eclipse, в комбинации с JDT, содержит пакеты с различными представлениями, такими как Ant, Console, Breakpoints, Package Explorer и т.д. Другие пакеты содержат компоновки Bookmarks (Закладки), Properties (Свойства), Tasks (Задачи), Problems (Проблемы), Progress (Прогресс), Call Hierarchy (Иерархия вызовов) и многие другие. Несколько редакторов и представлений могут быть сгруппированы вместе при помощи вкладок. Например, рис. 8.4 демонстрирует представления Console (Консоль) и Task (Задачи) с вкладками, группирующими их в верхней части экрана.

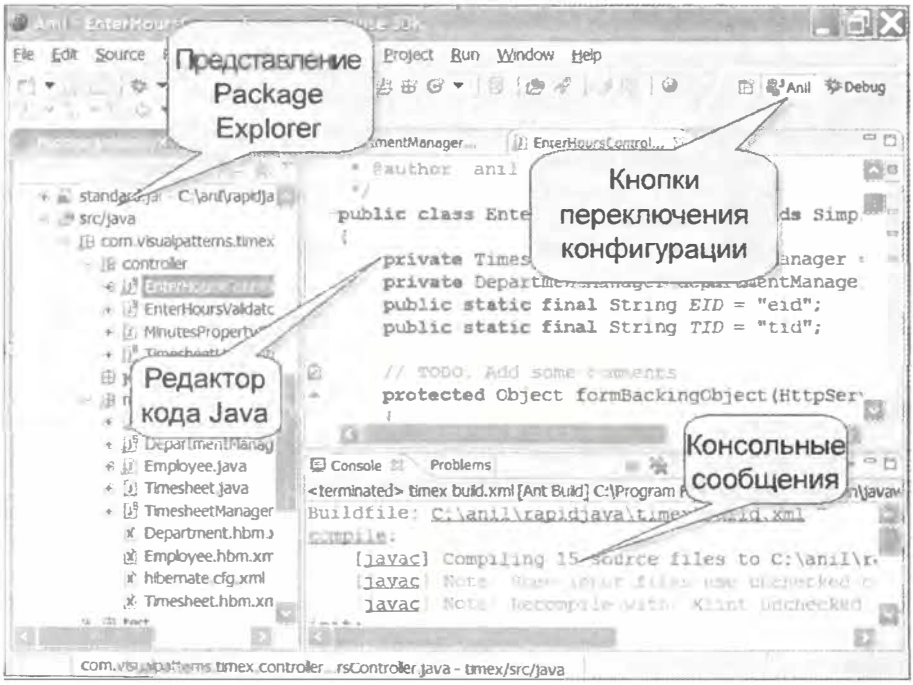



**Рис. 8.3.** Собственная компоновка (Anil)

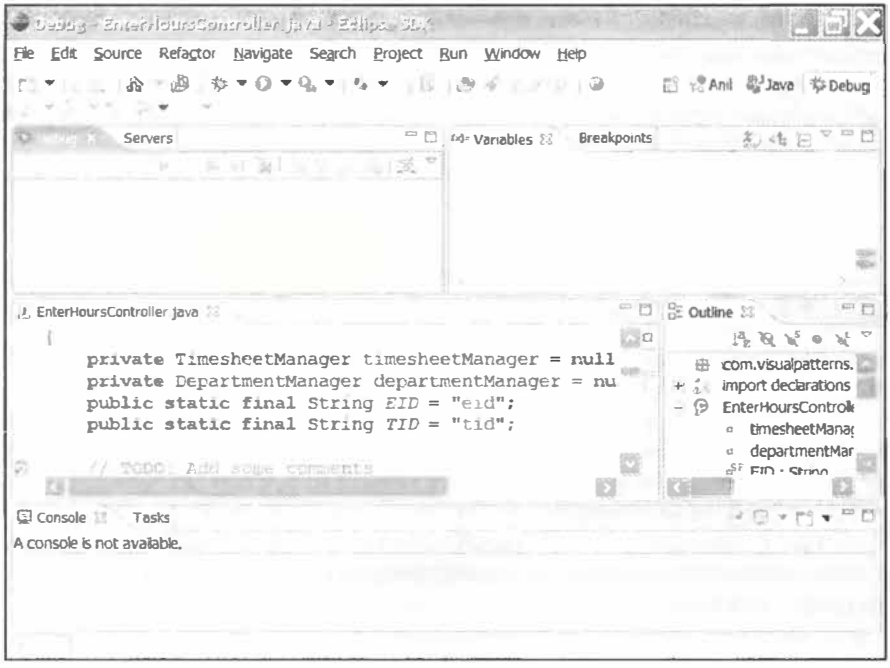


**Рис. 8.4.** Предопределенная компоновка (Debug)

### Проект

*Проект* (project) – это набор ваших файлов, которыми вы манипулируете, например файлы `.java`, `.xml` и т.д. Кроме того, Eclipse обеспечивает возможность работать с несколькими проектами разных типов (например, простой код Java и Web-приложение) в том же рабочем пространстве, каждый в конфигурации, специфической для проекта, а не в стандартной инструментальной конфигурации. Поскольку вскоре мы будем настраивать среду Eclipse для приложения Time Expression, организуем редактирование всех его файлов в едином пространстве, обеспечив перемещение между файлами и каталогами при помощи представлений Package Explorer или Navigator (Навигатор). Другими словами, мы можем работать со всеми нашими файлами `.java`, `.xml`, `.properties`, `.jsp` и другими в едином месте.

### Дополнения

*Дополнения* (plug-in) – это приложения (программы), которые могут быть легко установлены на платформе Eclipse и использованы как инструменты. Например, JDT является дополнением. Большинство людей полагают, что дополнения – это небольшие или простые программы. Однако с Eclipse дело обстоит не так, как вы со всей определенностью увидите далее в этой главе, когда мы дойдем до обсуждения JDT, а также установим и используем набор дополнений Eclipse Web Tools Platform (WTP).

### Мастера

Когда начнем работать с Eclipse, обратите внимание на изобилие мастеров, которыми он обладает! Рис. 8.5 демонстрирует часть списка мастеров, который можно увидеть, когда мы выбираем в меню File (Файл) пункты New (Новый) и Other (Другой) или когда используем специфическую для операционной системы комбинацию клавиш, например <Ctrl+N> на Microsoft Windows. Ко времени завершения этой главы нам будет доступно приблизительно 100 мастеров, включая дополнения для настройки Tomcat, конфигурирования Hibernate и связывания файлов, работы с файлами контекста приложения Spring, управления базой HSQLDB и многими другими, не применимыми к приложению Time Expression, например создания EJB, Web-служб и много другого!

## Установка Eclipse

Теперь, когда вы имеете некоторую базовую информацию об организации Eclipse Foundation, платформе Eclipse и ее архитектуре, займемся Eclipse на практике.

Прежде чем запустить среду Eclipse, совместимую с JRE (версии 1.3 или позже), ее следует установить на вашей системе (см. вставку "Среда выполнения Java (Java Runtime Environment – JRE), необходимая для Eclipse"). Если JRE уже установлен на вашей системе или вы установили его, читая главу 4, "Установка среды: JDK, Ant и JUnit", то все в порядке. Чтобы удостовериться в правильности установки VM (виртуальной машины) Java соответствующей версии, выполните в командной строке команду `java -version`.


**Рис. 8.5.** Мастера Eclipse

**Среда выполнения Java (Java Runtime Environment — JRE), необходимая для Eclipse**
Согласно разделу Frequently Asked Questions (FAQ, наиболее часто задаваемые вопросы) Web-сайта Eclipse (`eclipse.org`), если среда Eclipse не запускается на вашей системе, то "наиболее вероятной причиной является невозможность найти JRE. На вашем компьютере должна быть установлена *среда выполнения Java* (Java Runtime Environment — JRE). Для Eclipse нужна версия 1.3 или 1.4 среды Java 2 Standard Edition JRE. Среда JRE не входит в комплект SDK Eclipse". Более подробная информация по этой теме приведена на Web-сайте Eclipse.

Прежде чем мы перейдем к установке, еще раз просмотрим структуру каталога (рис. 8.6), созданную в главе 3, "Архитектура и модель проекта на базе XP и AMDD", поскольку она понадобится при работе с Eclipse.

Теперь пришло время запустить Eclipse! Ниже приведена простая инструкция, которая позволит нам установить и запустить Eclipse за несколько минут (с учетом того, что среда JRE установлена правильно и есть подключение к Интернету, достаточно быстрое для загрузки Eclipse).

Давайте загрузим самую последнюю версию SDK Eclipse с `http://www.eclipse.org/`. В этой главе используется версия 3.1.2 (самая последняя из доступных на момент написания этой книги). Обратите внимание на то, что предлагаемый SDK Eclipse включает указание платформы, например, типа Microsoft Windows, Linux, Solaris, AIX,

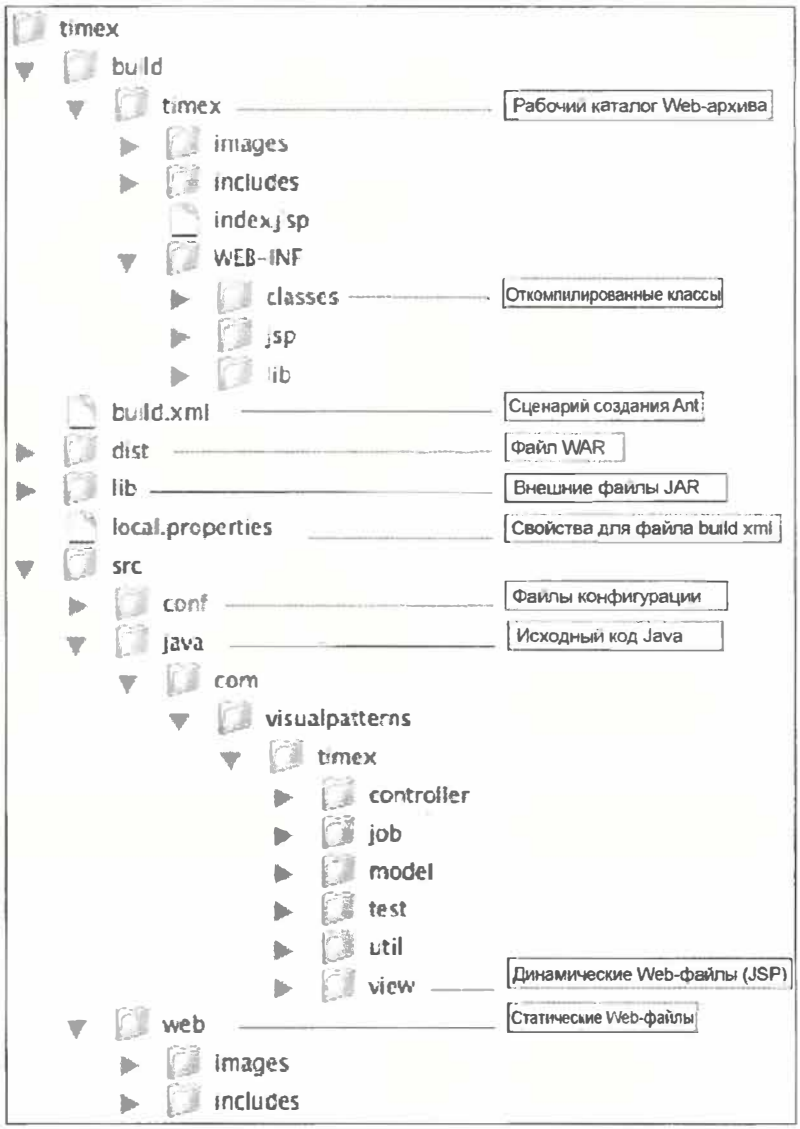



**Рис. 8.6.** Структура рабочего каталога для приложения Time Expression

HP-UX и Mac OS X. Сначала Web-сайт может показаться вам немного запутанным или слишком сложным, поскольку на нем представлено много проектов и сайтов-зеркал; лично я обычно начинаю с раздела Downloads (Загрузка), к которому можно перейти непосредственно с начальной страницы.

После распаковки архива SDK Eclipse (файла `.zip` или `.tar.gz`) запустите приложение Eclipse из каталога `/eclipse`. (В зависимости от вашей платформы, этот исполняемый файл будет называться `eclipse.exe`, `eclipse.app` или просто `eclipse`).

Непосредственно после выполнения команды `eclipse` появится запрос Select a workspace (Выбор рабочего пространства), подобный представленному на рис. 8.7. Обратите внимание на то, что я ввел собственный рабочий каталог, `C:\anil\rapidjava`, это на один уровень выше фактического каталога `timex`, представленного на рис. 8.6. После указания соответствующего каталога щелкните на кнопке OK, чтобы продолжить.

Затем мы увидим экран приветствия, представленный на рис. 8.8. Закройте его, щелкнув на значке "x" рядом с вкладкой Welcome (Добро пожаловать) или на пиктограмме Workbench (Инструменты) справа вверху.

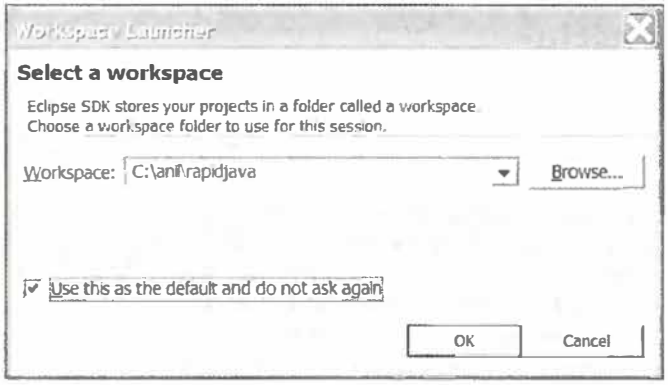



**Рис. 8.7.** Запрос Select a workspace

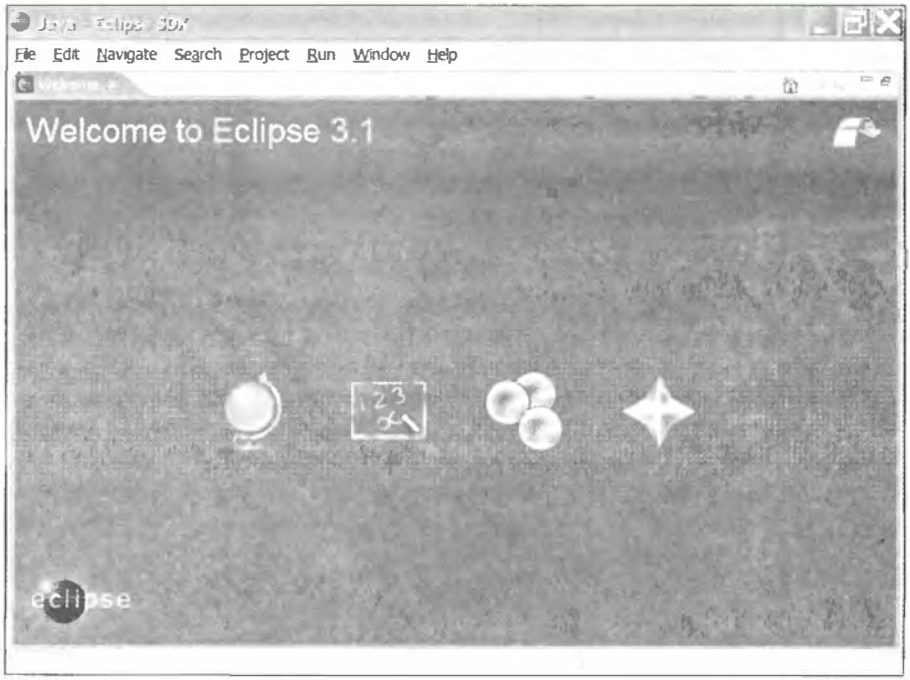



**Рис. 8.8.** Экран приветствия Eclipse

**Написана ли среда Eclipse на языке Java**

При запуске Eclipse сразу обращаешь внимание на высокую скорость пользовательского интерфейса данного продукта (в отличие от первого поколения инструментов, ориентированных на Java). Интерфейс данного приложения столь быстр, что у вас могли бы даже возникнуть сомнения, на самом ли деле это написано на языке Java. Короткий ответ: и да, и нет. Комплект SDK Eclipse использует ориентированный на Java *стандартный комплект инструментов управления* (Standard Widget Toolkit — SWT), который в свою очередь использует элементы управления GUI базовой операционной системы.

# Настройка Eclipse для приложения Time Expression

До сих пор я знакомил вас лишь с фундаментальными концепциями Eclipse, достаточными для перехода к его установке и настройки (в целях приложения Time Expression). Теперь мы можем создать проект Eclipse, используя наш файл Ant `build.xml`. Для этого в меню File (Файл) можно воспользоваться пунктами New (Новый), Project (Проект). На рис. 8.9 представлен экран мастера New Project (Новый проект). Выберите тип проекта Java Project from an Existing Ant Buildfile (Проект Java из существующего файла Ant), а затем щелкните на кнопке Next (Далее).

На рис. 8.10 демонстрируется следующий экран, где мы фактически указываем наш файл Ant `build.xml`. Для нас это ключевой момент, поскольку после этого шага мы можем начинать работать со всем нашим существующим исходным кодом в пределах Eclipse; нет больше командной строки! Теперь мы можем щелкнуть на кнопке Finish (Конец).


**Рис. 8.9.** Экран выбора типа нового проекта


**Рис. 8.10.** Экран New Java Project мастера Create a Java Project from an Ant Buildfile

На рис. 8.11 представлен первый экран, который мы должны увидеть непосредственно после создания нового проекта по имени `timex`. Однако, как вы могли бы заметить, здесь есть несколько проблем, а именно сотня, как демонстрирует представление Problems (Проблемы). Это связано с путями, которые я выбрал при установке для исходных и результирующих каталогов приложения Time Expression.

Я мог бы привести здесь безукоризненный практический пример, в котором используется стандартная структура каталога Eclipse; однако я преднамеренно продемонстрировал проблемы, поскольку это хорошая возможность для вас или вашей организации, если в ней уже имеется стандартный путь для установки структур каталога. Так или иначе, мы можем легко устранить эту проблему, как объясняется далее.

Давайте развернем дерево иерархии каталогов и файлов проекта `timex` в представлении Package Explorer (слева на экране). Найдите в структуре каталог `src` и щелкните на нем правой кнопкой мыши, чтобы отобразить контекстное меню. Выберите пункты Build Path (Путь построения), Remove from Build Path (Удалить из пути построения), как показано на рис. 8.12.

После того как мы это сделаем, каталог `src` вновь появится в списке, но уже с другой пиктограммой. Обратите внимание на то, что фактически мы не удалили каталог, а только убрали его из пути построения. Теперь мы развернем каталог `src`, найдем внутри него каталог `java`, щелкнем на нем правой кнопкой мыши и в появившемся контекстном меню выберем пункты Build Path (Путь построения), Use as Source Folder (Использовать как исходную папку), как показано на рис. 8.13.

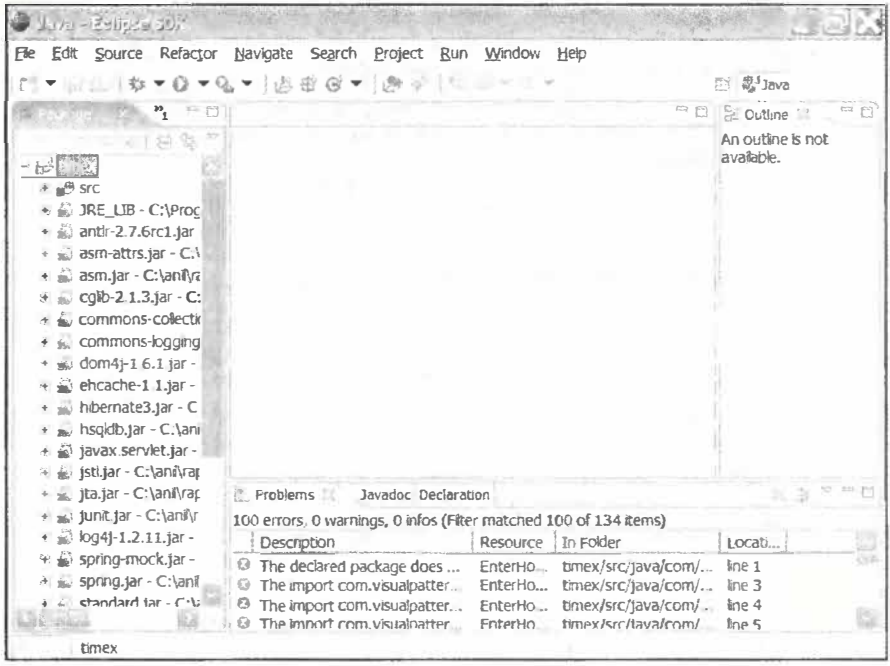



**Рис. 8.11.** Только что созданный проект `timex`

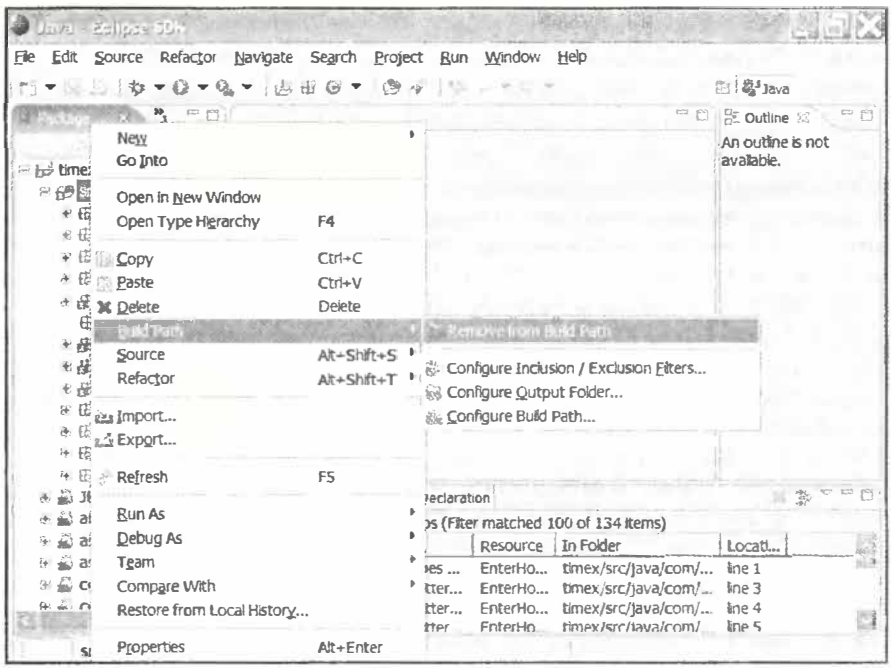



**Рис. 8.12.** Удаление каталога `src` из пути построения

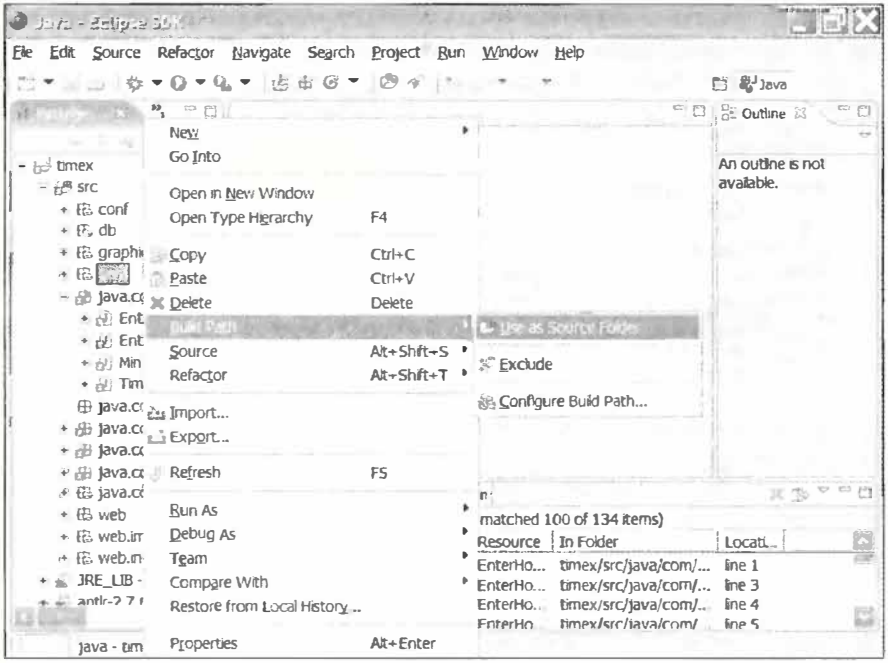



**Рис. 8.13.** Пункт меню Use as Source Folder

Мы уже почти закончили. Мы позаботились об исходном каталоге; теперь необходимо откорректировать папку для вывода наших откомпилированных файлов (`.class`), чтобы мы могли использовать наш файл Ant `build.xml` для компиляции классов, а также для встроенных возможностей компилятора (см. вставку "Мгновенная компиляция при сохранении!").

**Мгновенная компиляция при сохранении!**

Если вы уже занимались программированием, то, вероятно, привыкли к методике проб и ошибок, т.е. вы пишете некоторый код, компилируете его, исправляете ошибки компиляции, повторяете компиляцию, устраняете следующие ошибки компиляции, повторяете компиляцию, запускаете результат, находите ошибки времени выполнения программы. Хорошо, вы имеете представление.

Eclipse поставляется со встроенным компилятором, предназначенным для компиляции файлов `.java` в файлы `.class`, при каждом сохранении ваших файлов. Это, возможно, одно из наиболее важных, хотя и наименее рекламируемых преимуществ Eclipse (на мой взгляд). Компиляция происходит настолько быстро, что вы даже не замечаете, что она произошла. Просто замечательно, не так ли?

Когда я нажимаю комбинацию клавиш <Ctrl+S>, чтобы сохранить файл `.java`, немедленно получаю файл `.class`, созданный автоматически в мгновение ока (так сказать). Безусловно, компиляция имеет место только тогда, когда исходный код не имеет ошибок (предупреждения допускаются).

Чтобы изменить заданную по умолчанию папку вывода, щелкните в представлении Package Explorer правой кнопкой мыши на каталоге `src/java` и в появившемся контекстном меню выберите пункты Build Path (Путь построения), Configure Output Folder (Настроить папку вывода), как показано на рис. 8.14.

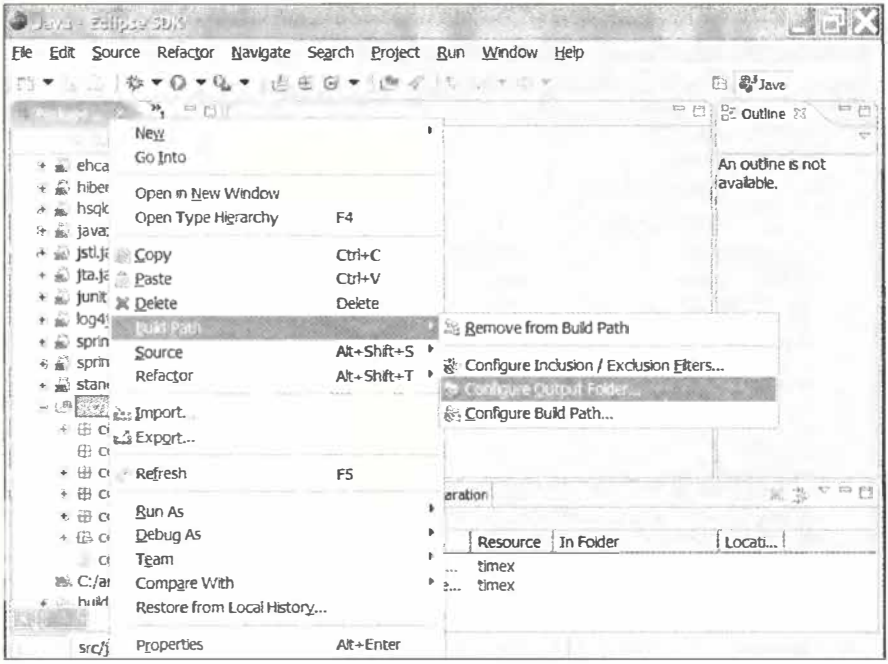



**Рис. 8.14.** Настройка папки вывода

Теперь мы должны увидеть экран, подобный представленному на рис. 8.15. Как можно заметить, здесь мы меняем заданную по умолчанию папку вывода на `build/timex/WEB-INF/classes`. Это относится к рабочему пространству, которое мы определили ранее (см. рис. 8.7).

Теперь мы на самом деле готовы использовать Eclipse в нашей работе! Однако давайте побеспокоимся еще о нескольких действиях по очистке. Они необязательны, но мы не хотим оставлять за собой никакого мусора, поэтому задержимся еще.

Когда мы сменили исходную папку на `src/java`, Eclipse приготовилась располагать откомпилированные файлы `.java` в своей папке вывода, заданной по умолчанию (ею является `timex/bin/`). Мы можем удалять эту папку вручную каждый раз, но лучше давайте настроим Eclipse так, чтобы она не создавала этот каталог снова. Для этого

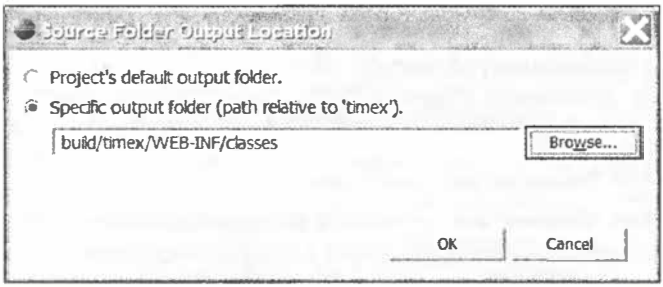



**Рис. 8.15.** Указание положения папки вывода

щелкните в представлении Package Explorer правой кнопкой мыши на любом элементе (файл, каталог, или имя проекта) и в появившемся контекстном меню выберите пункты Build Path (Путь построения), Configure Build Path (Настроить путь построения). На рис. 8.16 представлен экран свойств проекта нашего приложения, Time Expression. Обратите внимание на то, что заданная по умолчанию папка вывода заменена на `timex/build/timex/WEB-INF/classes` (низ экрана). Это та же папка вывода, которую мы выбрали для нашего каталога `src/java`. Это тоже необязательный шаг, поскольку мы работаем только с одним исходным каталогом, папка по умолчанию и папка вывода исходного каталога те же самые. После этого мы можем щелкнуть правой кнопкой мыши в представлении Package Explorer на каталоге `bin/` и спокойно удалить его.

Вы также могли бы заметить в представлении Package Explorer элемент каталога с именем вроде `C:/anil/rapidjava/timex/build/timex/WEB-INF/classes`, содержащий индикатор `(unknown)`, т.е. неизвестный, в конце всплывающей подсказки, появляющейся при наведении курсора мыши на элемент. Мы можем спокойно удалить этот элемент, щелкнув на нем правой кнопкой мыши и выбрав в появившемся контекстном меню пункты Build Path (Путь построения), Remove from Build Path (Удалить из пути построения).

Теперь среда Eclipse полностью установлена и готова к разработке приложения Time Expression!

Между прочим, я опробовал эту настройку на моем Mac OS X и Linux, и она сработала корректно. На рис. 8.17 представлены инструменты Eclipse на Mac OS X, с открытым проектом Time Expression. Рис. 8.18 демонстрирует Eclipse на Linux (с использованием загрузочного CD Knoppix Linux). Впечатляет, не так ли? Причем мы только начали, осталось гораздо больше.

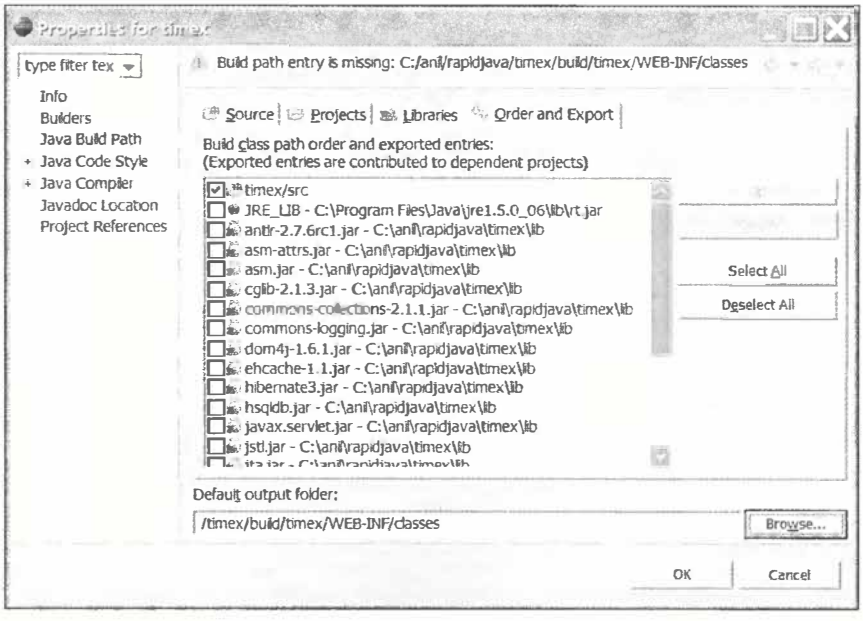



**Рис. 8.16.** Окно свойств проекта

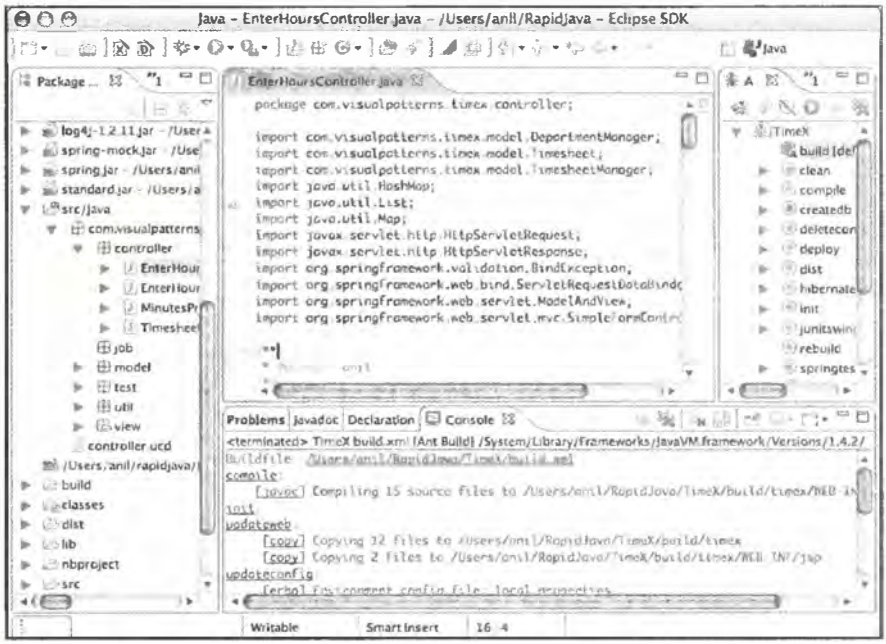



**Рис. 8.17.** Среда Eclipse с проектом `timex` на Mac OS X

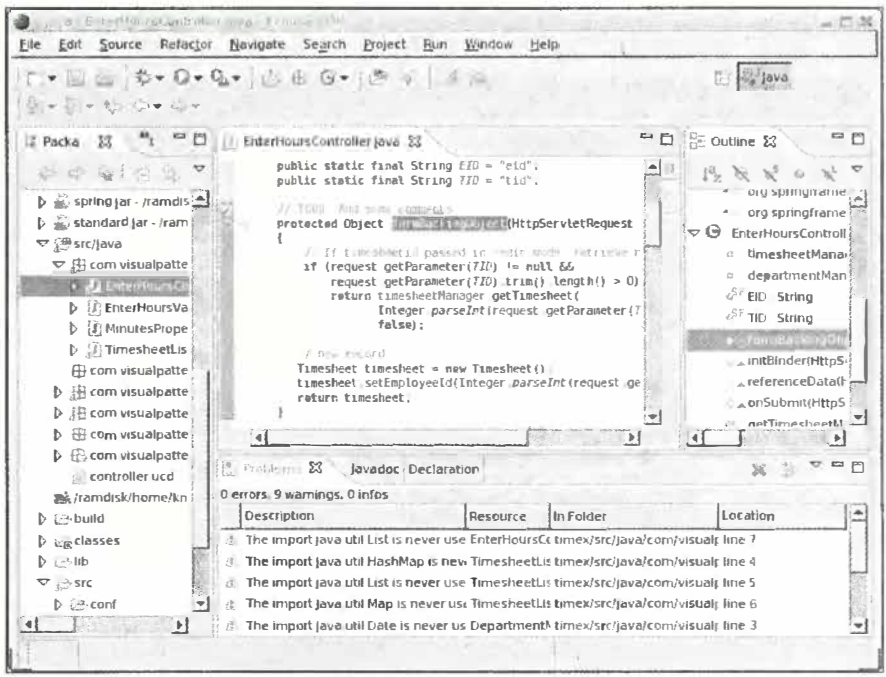



**Рис. 8.18.** Среда Eclipse с проектом `timex` на Linux (загрузочный CD Knoppix)

# Возможности инструментов разработки Java (JDT)

Дополнение JDT построено на платформе Eclipse, для придания ей функциональных возможностей, связанных с Java. JDT — это базовая часть SDK Eclipse. Платформа Eclipse, объединенная с дополнением JDT, предоставляет так много возможностей, что для их описания хватило бы целой книги. Следовательно, ради здравого смысла, ниже я приведу лишь краткий перечень некоторых из них. Более подробная информация о работе этих средств содержится в интерактивной справочной системе Eclipse (справочная система Eclipse рассматривается далее в этой главе).

- Управление проектами Java и такими файлами, как `.java`, `.class`, `.jar`, а также файлами Javadocs.
- Редактирование исходного кода Java, с выделением цветом ключевых слов и синтаксиса, интеллектуальное редактирование кода, быстрое исправление ошибок, импорт организации и многое другое.
- Многообразие представлений Java. Рис. 8.19 демонстрирует пример компоновки Java Browsing (Просмотр Java), позволяющей просматривать содержимое пакетов Java, типы данных (классы, интерфейсы), их переменные и члены. Это также прекрасное средство просмотра других файлов `.jar` (например, `spring.jar`). К представлениям Java относятся также JUnit, Ant, Package Explorer и многие другие.

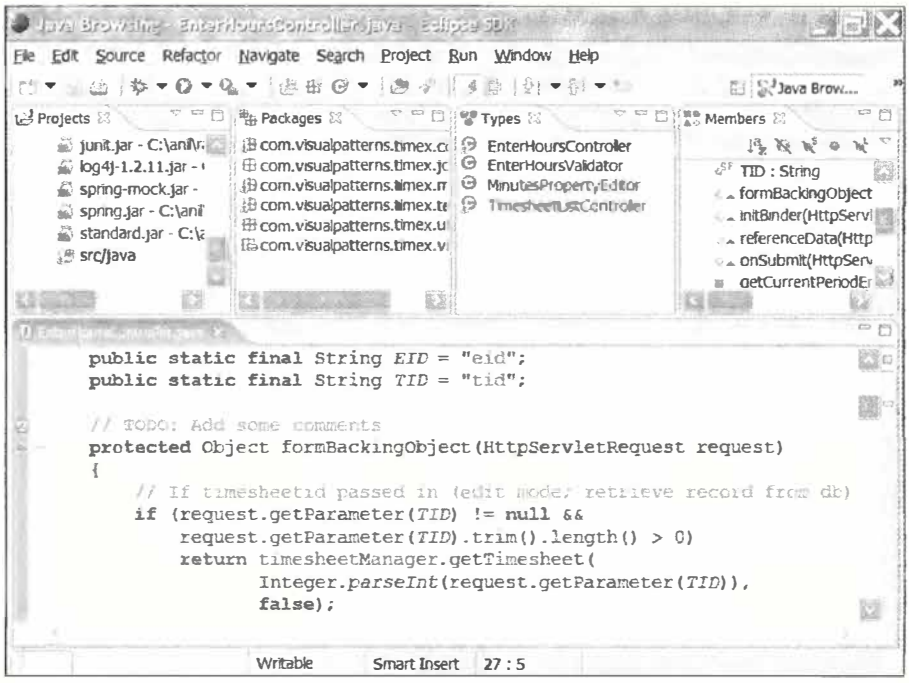



**Рис. 8.19.** Компоновка Java Browsing

- Каждый раз, когда мы сохраняем файл `.java`, он немедленно компилируется в файл `.class` (см. вставку "Мгновенная компиляция при сохранении!").
- Быстрое интеллектуальное исправление ошибок (<Ctrl+1>) и помощник содержимого (<Ctrl+Пробел>). Опробуйте при программировании их везде и всюду, поскольку эти клавиши делают очень много из того, что перечислено здесь. Помощник выполняет различные действия на основании положения курсора, т.е. ведет себя интеллектуально, предлагая на выбор список адекватных возможностей. Например, комбинация клавиш <Ctrl+Пробел> может дописать за вас строку кода, предоставив раскрывающийся список с возможными вариантами, показать сигнатуру вызываемого метода и многое другое. Повторюсь, существует слишком много возможностей, чтобы перечислять их здесь, но интерактивная справочная система Eclipse предоставляет подробную информацию об этих клавишах.
- Создание кода с использованием упакованных или специальных шаблонов (для фрагментов кода), заключение кода в блок `try/catch`, автоматический импорт и многое другое. Рис. 8.20 демонстрирует контекстное меню Source (Исходный код). Создание методов установки и получения значений (setter/getter).
- JDT Eclipse предоставляет так много параметров форматирования кода, что сначала просто теряешься. Если бы мне пришлось делать предположение о количестве доступных параметров настройки форматирования кода, я, вероятно, говорил бы приблизительно о сотне или более. Рис. 8.21 демонстрирует, что я

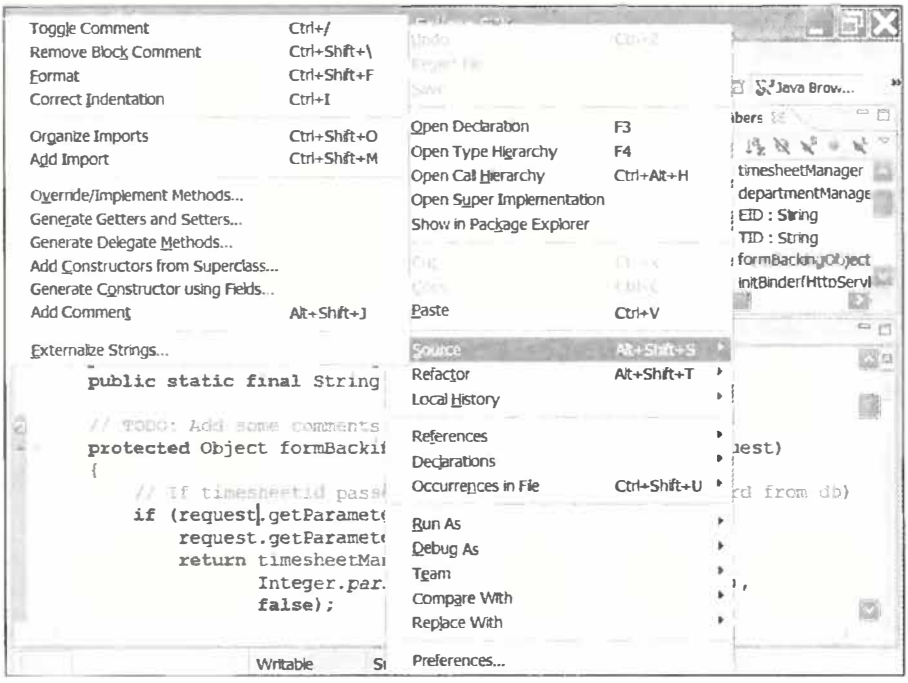



**Рис. 8.20.** Контекстное меню Source

имею в виду. Обратите внимание на количество параметров первой вкладки, теперь умножьте его на восемь вкладок, представленных на этом экране, и вы получите представление об общем количестве параметров.

- Предупреждения компилятора и сообщения об ошибках отображаются в левой и правой части окна редактора. Ошибки выделяются красным цветом, а предупреждения — желтым. Рис. 8.22 демонстрирует сообщение об ошибке, которую я создал намеренно. Маркер в левом поле — это позиция ошибки на уровне страницы; индикатор в правом поле — это ее позиция в файле. Обратите также внимание на то, что предупреждения и сообщения об ошибке отображаются в представлении Problems.
- Дополнение JDT предоставляет все средства отладки, которые только можно пожелать, например контрольные точки, просмотр значений переменных, отслеживание выражений и многое другое. Eclipse поддерживает также возможность *быстрого обмена* (hotswap), если ваш JRE также поддерживает ее; она позволяет изменять код в отладчике и немедленно перезагружать его, без необходимости перезагрузки сеанса отладки (весьма замечательная возможность!). Более подробно отладку мы рассмотрим в следующей главе.
- Дополнение JDT обеспечивает бесконфликтное переименование методов и классов, а также ссылок на них по всему проекту. Например, если мы переименовываем класс, Eclipse будет пытаться изменить все ссылки на него в коде Java и файлах XML.

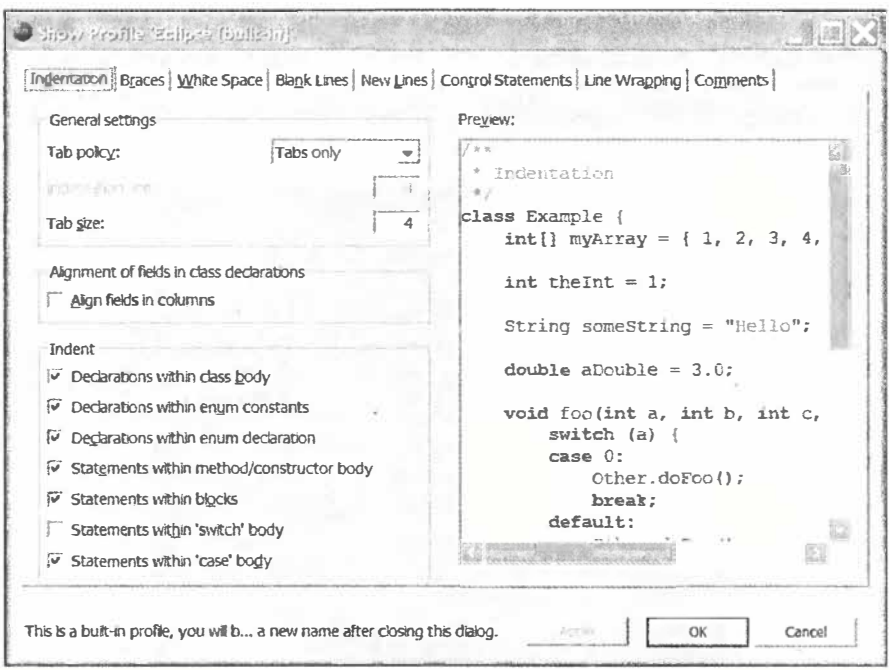



**Рис. 8.21.** Параметры форматирования кода Java

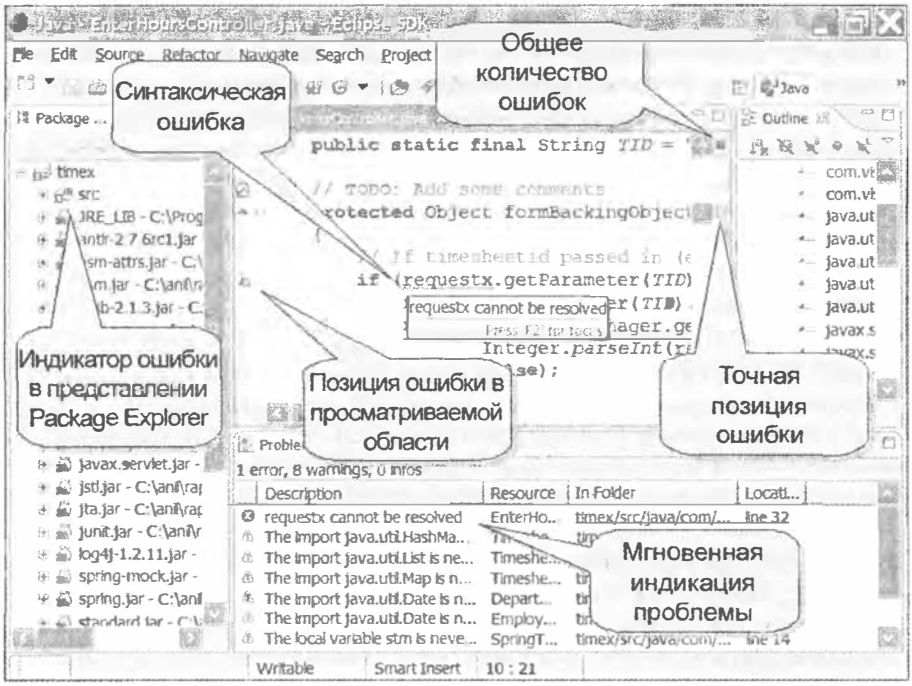



**Рис. 8.22.** Сообщения компилятора об ошибках и предупреждения

- Мощнейший поиск на основании сигнатур используемых методов Java и другие возможности, поддерживаемые JDT. Рис. 8.23 демонстрирует диалоговое окно Search (Поиск). Обратите внимание на то, что мы можем искать в файле, коде Java и даже в дополнениях. Кроме того, существуют и другие параметры поиска, а также повторного поиска.

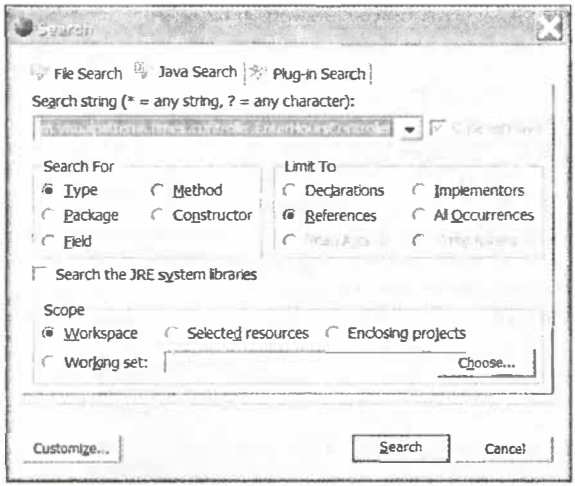



**Рис. 8.23.** Диалоговое окно Search

- Дополнение JDT не только обеспечивает выделение синтаксиса цветом и отладку, но и предоставляет такие возможности, как способность добавлять комментарии со словом TODO (мы можем выбирать и другие слова) и сделать так, чтобы они появлялись в представлении Tasks (Задачи). Рис. 8.24 демонстрирует один из элементов TODO, находящийся во всех файлах исходного кода.

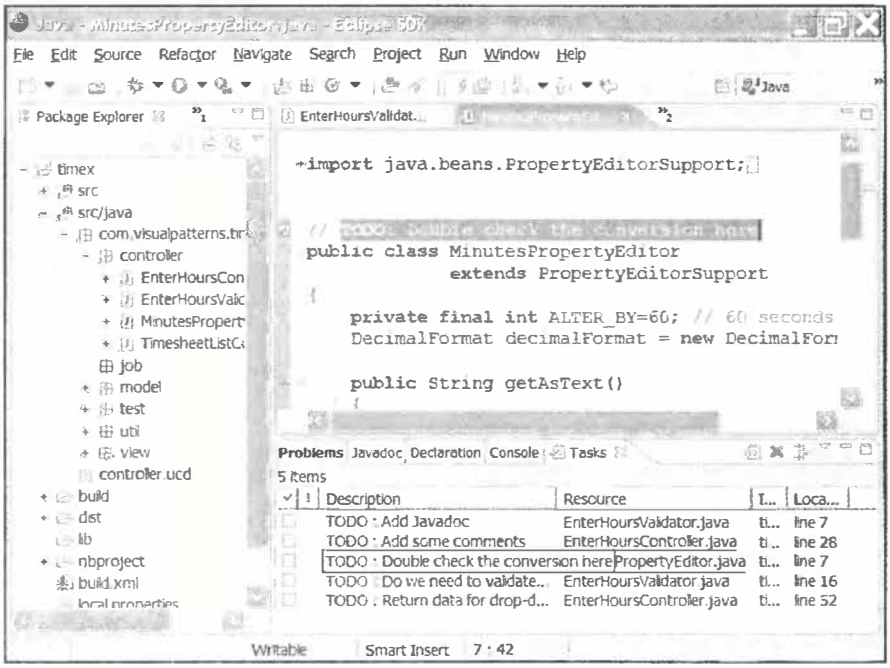



**Рис. 8.24.** Комментарий TODO и представление Tasks

Как я упоминал ранее, Eclipse предоставляет мастера множества типов. Мастера, специфические для JDT, позволяют создавать классы, пакеты, аннотации, простые файлы и многое другое. Каждый из этих мастеров мощен по-своему. Например, рис. 8.25 демонстрирует мастер Class Wizard. Обратите внимание на то, что здесь мы можем указать практически все, что необходимо для создания каркаса класса, принадлежащего некоторому пакету, дополнить другой класс или реализовать некий интерфейс, содержащий метод `main`, наследуя конструкторы из базового класса и многое другое. Если бы Eclipse мог еще писать код бизнес-логики, нам вообще нечего было бы делать. Дополнение JDT поставляется укомплектованным дополнением JUnit. Дополнение JUnit предоставляет мастера и представления, необходимые для создания и проверки классов. Мастер JUnit представлен на рис. 8.26. Представление JUnit мы будем использовать далее в этой главе.

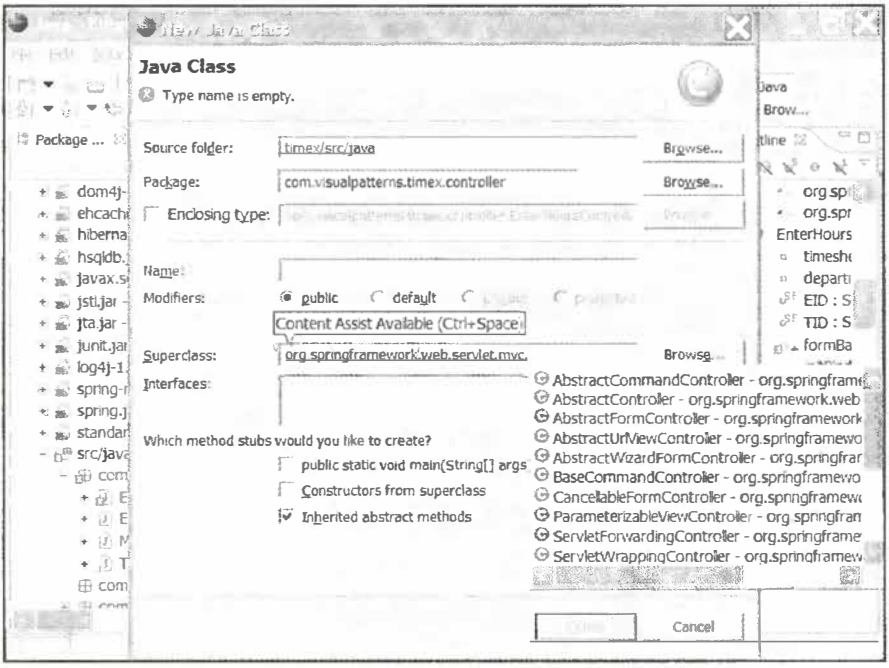



**Рис. 8.25.** Мастер New Java Class

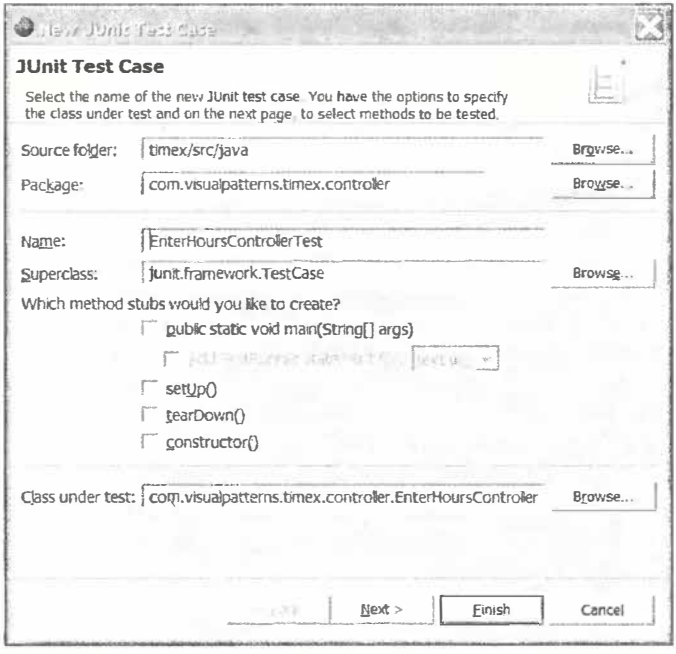



**Рис. 8.26.** Мастер New JUnit Test Case

Как вы уже догадались, Ant – это неотъемлемая часть Eclipse. Рис. 8.27 демонстрирует представление Ant, используемое для выполнения задач Ant, а также редактор Ant, применяемый для редактирования кода Ant и создания файлов XML (обратите внимание на контекстную справку, комбинация клавиш <Ctrl+Пробел>, для элементов и атрибутов XML). Кроме того, представление Console демонстрирует вывод команд Ant, причем щелчок мышью на имени файла позволяет немедленно перейти к сути ошибки!

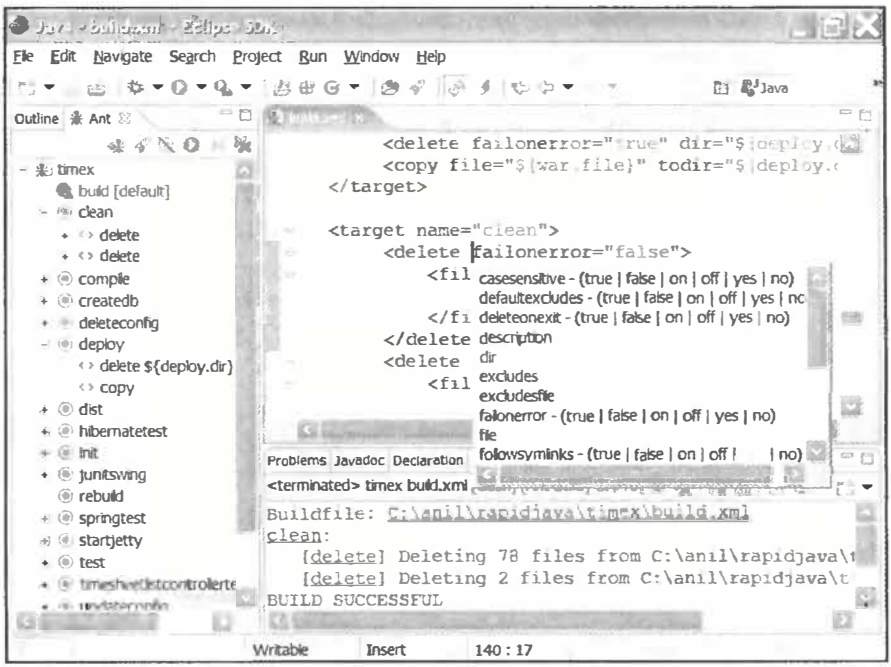



**Рис. 8.27.** Редактор и представление Ant

Средства экспорта Eclipse поддерживают множество типов файлов. Мне нравится тот факт, что любые экспортируемые файлы любых дополнений находятся в одном месте. Используемый здесь термин "экспорт" не до конца соответствует сути, поскольку при помощи этого средства мы можем создавать даже файлы JAR. Рис. 8.28 демонстрирует список типов файлов, доступных для экспорта (в текущей конфигурации).

Компонент Java Scrapbook очень удобен для экспериментов с отдельными строками кода Java, как показано на рис. 8.29. Вы можете создать новый альбом *вырезок* (scrapbook) Java при помощи мастера (меню File (Файл), пункты New (Новый) и Other (Другой)), а далее найти в древовидном представлении Java, Java Run/Debug, Scrapbook Page. Затем необходимую строку кода можно выполнить, выделив ее и выбрав пункт Execute (Выполнить) в контекстном меню (щелчок правой кнопкой мыши) или меню Run (Пуск) либо нажав комбинацию клавиш <Ctrl+U> на Microsoft Windows.

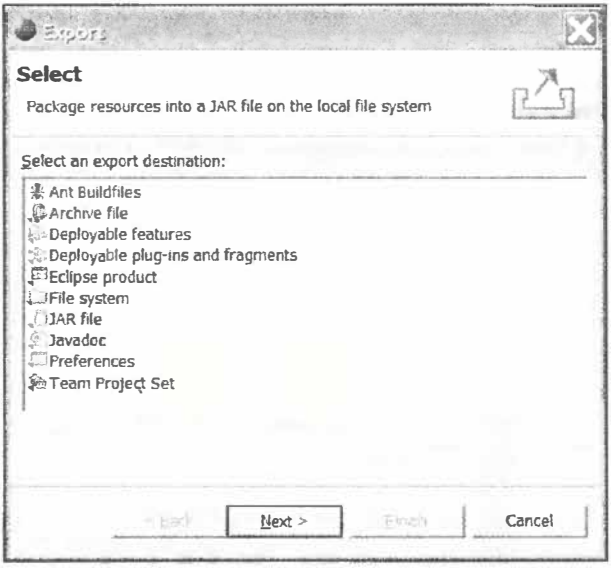



**Рис. 8.28.** Возможности экспорта

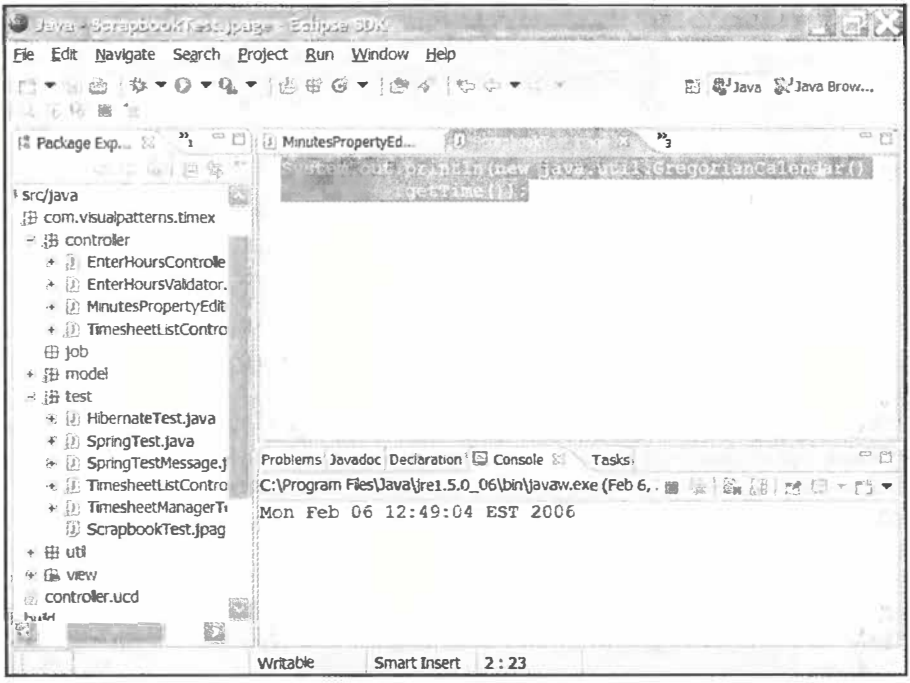



**Рис. 8.29.** Страница Java Scrapbook

Дополнение JDT предоставляет обширные возможности рефакторинга кода, хотя я должен признать, что это мое слабое место, поскольку я не очень интенсивно использовал эту возможность. Согласно интерактивной справочной системе Eclipse, "инструменты рефакторинга обеспечивают множество трансформаций, описанных в книге Мартина Фаулера (Martin Fowler) *Refactoring: Improving the Design of Existing Code*, издательство Addison Wesley 1999 год, включая Extract Method, Inline Local Variable и т.д.". Концепция, лежащая в основе рефакторинга, подразумевает улучшение кода без изменения функциональных возможностей, которые он предоставляет. Возможно, это лишь мое личное мнение, но, на мой взгляд, идеи рефакторинга скорее усложняют разработку программного обеспечения, хотя большинство параметров рефакторинга в Eclipse довольно понятны и просты в использовании.

На рис. 8.30 представлены пункты меню Refactor. Например, мы можем *извлечь* (создать) класс интерфейса, используя конкретные методы конкретного класса. Другие параметры позволяют извлечь метод из родительского класса, переименовать пакет, класс, метод или встраиваемые переменные, добавить исключения и многое другое.

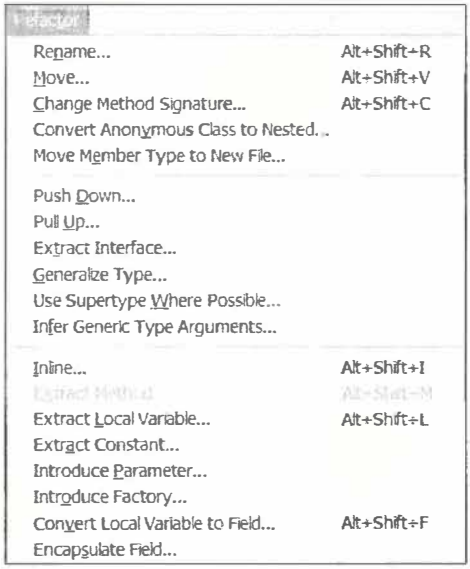



**Рис. 8.30.** Меню Refactor

Мы только просмотрели некоторые из множества средств, предоставляемых комбинацией платформы Eclipse и дополнения JDT, но, как говорится, *это еще не все!*

На Web-сайте `eclipse.org` и в других местах есть так много информации об Eclipse, что ее хватило бы на несколько книг (а готовится еще больше). Тем не менее, к концу этой главы я постараюсь изложить не мало. Как я упоминал ранее, мощь Eclipse обусловливают дополнения, доступные для этой платформы. До сих пор мы обсуждали только одно дополнение, JDT. Ранее я также упомянул, что доступны сотни дополнений: маленькие и простые, большие и сложные. Одним из производителей дополнений является корпорация Borland; однако список компаний, разрабатывающих дополнения для Eclipse, слишком длинный, включая саму организацию Eclipse Foundation, как будет продемонстрировано в следующем разделе.

# Установка дополнения Eclipse Web Tools Platform (WTP)

Если вы подумали, что дополнение JDT перегружено возможностями, то вы будете поражены разнообразием возможностей, предоставляемых дополнением WTP (Eclipse Web Tools Platform — *платформа инструментальных средств Eclipse для Web*). Согласно Web-сайту `eclipse.org`, "проект The Eclipse Web Tools Platform (WTP) дополняет платформу Eclipse инструментальными средствами для разработки Web-приложений J2EE. Проект WTP включает следующие инструменты: редакторы исходного кода для HTML, JavaScript, CSS, JSP, SQL, XML, DTD, XSD и WSDL; графические редакторы для XSD и WSDL; средства проектирования J2EE; компиляторы и модели; навигатор J2EE; мастер и средство просмотра Web-служб; инструменты проверки WS-I Test Tools; средства доступа к базе данных; а также инструментальные средства создания запросов и модели". Все это так и работает, как рекламируется. Однако на самом деле возможностей даже больше, поскольку краткое описание на `eclipse.org` не упоминает поддержку для управления различными Web-серверами и серверами приложений, которые мы используем для запуска и остановки сервера Tomcat далее в этой главе.

Для установки дополнения WTP следуйте инструкции, предоставленной Web-сайтом `http://www.eclipse.org`. Например, я установил его, выбрав в меню Help (Справка) пункты Software Updates (Обновление программного обеспечения), Find (Найти) и Install (Установить). На рис. 8.31 показано, как я фактически добавил WTP новый дистанционный сайт (кнопка New Remote Site), `http://download.eclipse.`

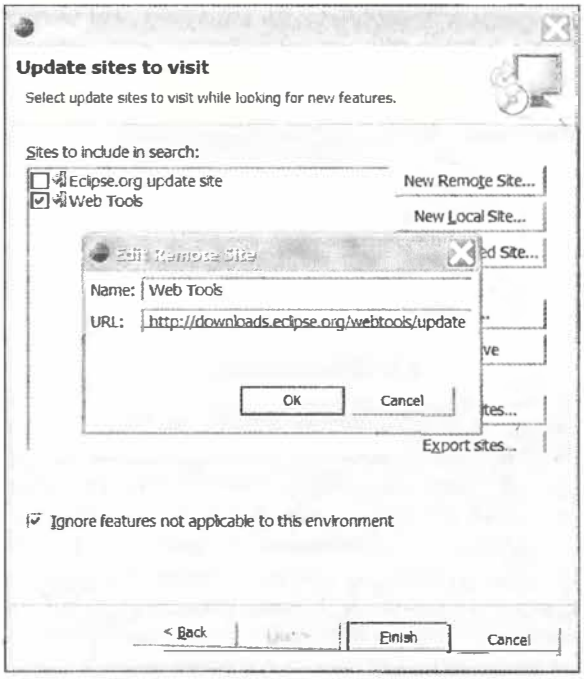



**Рис. 8.31.** Обновление программного обеспечения Eclipse (здесь устанавливается WTP)

org/webtools/updates/, и получил оттуда все дополнения, используя возможность обновления программного обеспечения Eclipse. Сам процесс прошел очень гладко; однако, в связи с большим объемом загружаемой информации, установка заняла некоторое время. После завершения установки WTP следует предложение перезапустить Eclipse. Кроме того, после установки WTP, Eclipse может сначала загружаться немного дольше, возможно, потому, что для проверки и загрузки установленных дополнений требуется некоторое время.

После успешной установки WTP мы можем выбрать компоновку J2EE (меню Window (Окно), пункт Open Perspective (Открыть компоновку)).

Дополнение WTP позволяет нам работать как со статическими (файлы HTML), так и с динамическими (файлы EAR) проектами. Здесь вы найдете редакторы для таких языков разметки, как HTML и XML, а также редакторы для кода JSP и JavaScript. Вы получаете в свое распоряжение те средства, которые вы, вероятно, и ожидали от современной версии Eclipse, включая помощник исходного кода, шаблоны и многое другое. Кстати, дополнение WTP предлагает не только редактор содержимого J2EE. Оно предоставляет также мощнейшие инструментальные средства для данных и серверов, как мы увидим далее в этой главе, когда дело дойдет до управления базой HSQLDB и сервером Tomcat.

# Применение Eclipse для приложения Time Expression

До сих пор мы применяли разнообразные инструменты и технологии из командной строки, поэтому было бы неплохо получить те же или лучшие функциональные возможности, с учетом парадигмы панели инструментов, которую я использую для описания Eclipse в этой главе.

Вам будет приятно узнать, что доступны дополнения для каждой из этих технологий. Приложение JDT уже укомплектовано представлениями для Ant и JUnit, так что нам осталось получить дополнения для других технологий. Сделать это относительно просто, можно воспользоваться возможностью обновления программного обеспечения Eclipse или загрузить дополнение как файл архива (например, .zip и .tar.gz) и установить его. Давайте теперь коротко рассмотрим дополнения, применимые для приложения Time Expression.

## Предустановленные дополнения JDT

Как я упомянул ранее, в комплект JDT входит несколько дополнений, например, редактор кода JSP, представление Ant и дополнения JUnit. Кроме того, мы рассмотрели пару дополнений Ant и JUnit, связанных со снимками экранов на рис. 8.26 и 8.27.

На рис. 8.32 демонстрируется вид файла enterhours.jsp нашего экрана Enter Hours, в процессе редактирования. Обратите внимание на контекстную справку (вызываемую комбинацией клавиш <Ctrl+Пробел>) для библиотеки дескрипторов spring:bind и интеграцию с представлением Outline (Иерархическая структура), отображающим элементы и атрибуты JSP.

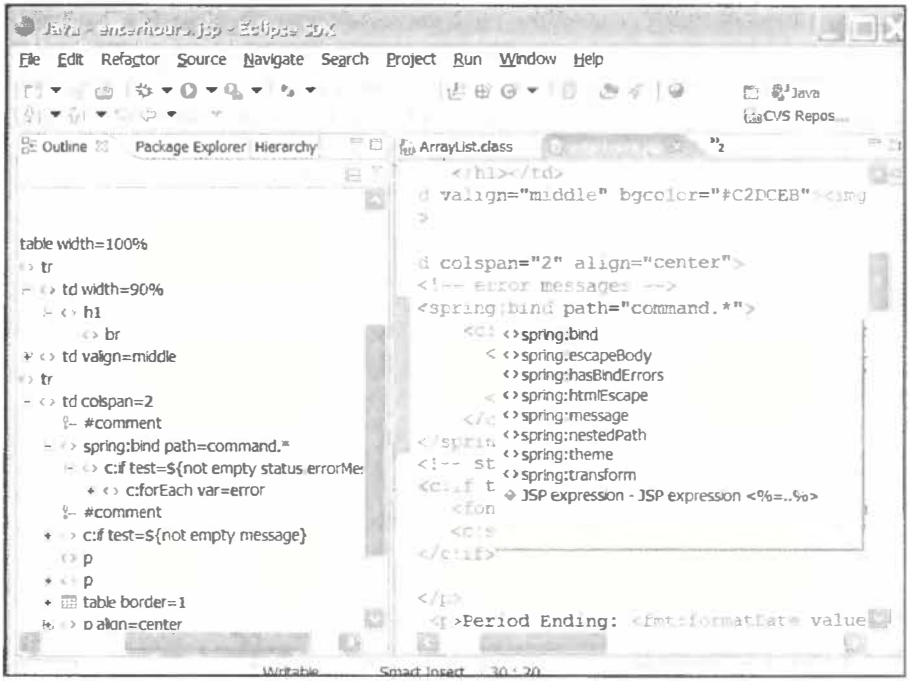



**Рис. 8.32.** Редактор JSP

Я использую Ant внутри Eclipse только тогда, когда хочу скопировать файлы конфигурации, откомпилировать и установить код или выполнить другие специфические задачи, которые я определил в нашем файле `build.xml`. Однако для компиляции отдельных файлов `.class` я полагаюсь на компилятор JDT, поскольку он весьма быстр и прост (т.е. осуществляет автоматическую компиляцию при каждом сохранении файлов `.java`). Чтобы применить наш файл `build.xml` внутри Eclipse, необходимо добавить его в представление Ant. Для этого выберите в меню Window (Окно) пункты Show View (Показать представление) и Ant. В представлении Ant задействуйте параметр Add Buildfiles и выберите файл `build.xml` приложения Time Expression, как показано на рис. 8.33.

Что касается JUnit, его интегрированную поддержку в JDT я использую довольно интенсивно. Но сначала давайте добавим представление JUnit, для этого выберите в меню Window (Окно) пункты Show View (Показать представление), Other (Другое) (ищите в списке представлений пункт JUnit). Рис. 8.34 демонстрирует, как мы можем запустить класс `TimesheetManagerTest`, выбрав в меню Run (Пуск) пункты Run As (Выполнить как), JUnit Test (Проверка JUnit) (это можно сделать также в меню Eclipse, контекстном меню исходного кода или представления Package Explorer). Результаты выполнения пробного запуска отображаются в представлении JUnit и выводе команд в представлении Console.

Хотя здесь мы обсуждаем отдельный проверочный класс JUnit, реальные проекты обычно содержат большое количество проверочных классов JUnit (их могут быть десятки и сотни).

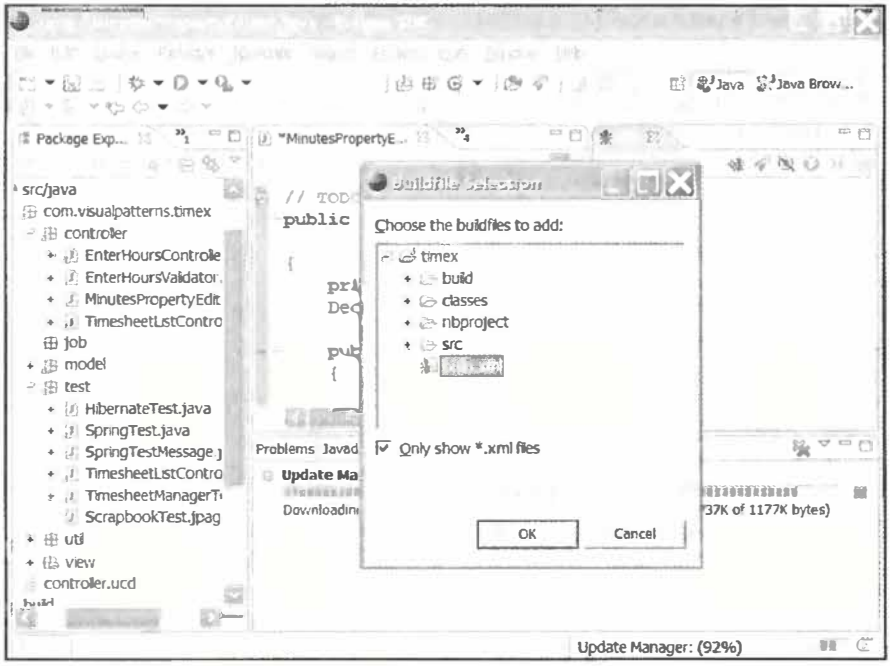



**Рис. 8.33.** Добавление файла `build.xml` приложения Time Expression

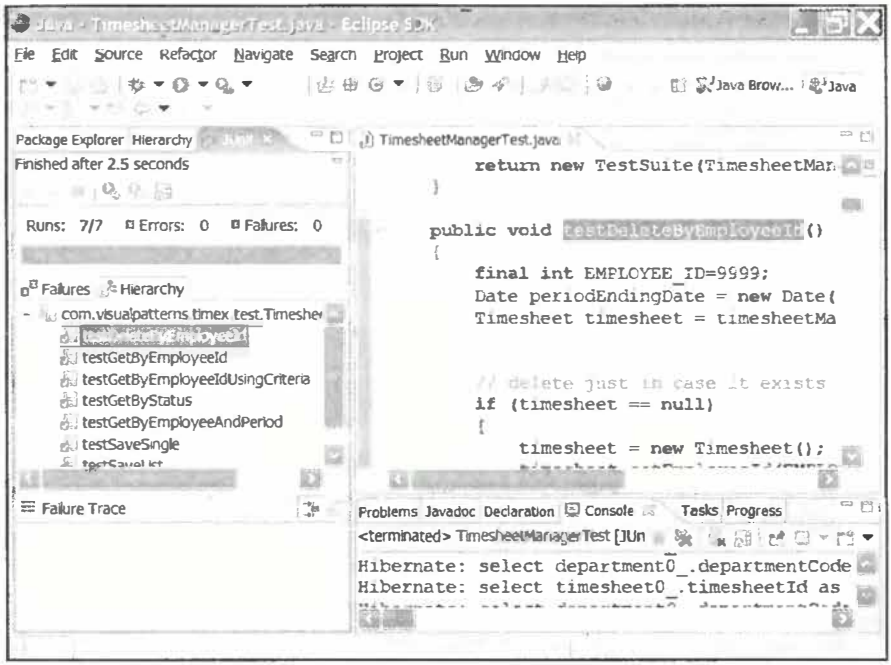



**Рис. 8.34.** Пример запуска проверки JUnit

## Дополнение Data (для HSQLDB)

Теперь обеспечим возможность работы с базой данных HSQLDB внутри графического интерфейса Eclipse. Вначале следует удостовериться, что сервер HSQLDB запущен, как описано в главе 5, "Применение Hibernate для постоянных объектов". Затем добавим представление Database Explorer (Проводник баз данных) дополнения Data. В представлении Database Explorer щелкните правой кнопкой мыши на пиктограмме Connections (Соединения) или на соответствующей кнопке панели инструментов представления. Рис. 8.35 демонстрирует мои параметры подключения. Я не мог найти диспетчер для базы данных HSQLDB, но я обошел эту проблему, используя DB2 UDB V8.1, и все, кажется, сработало нормально. Для HSQLDB существует несколько специальных дополнений, но я хотел использовать только стандартные дополнения, предоставляемые WTP. Рис. 8.36 демонстрирует возможности, связанные с дополнением Data. Например, я могу просмотреть объекты базы данных, осуществить обращение к таблице `Employee` и даже создать сценарий DDL!

## Дополнение Servers (для Tomcat)

Далее обеспечим возможность работы с сервером Tomcat внутри графического интерфейса Eclipse. Сначала добавим представление Servers (Серверы). Теперь можем щелкать правой кнопкой мыши в представлении Servers (или открыть мастер Server в списке мастеров, доступном при выборе в меню File (Файл) пунктов New (Новый), Other (Другой)). На следующем экране выберем для установки сервер Tomcat версии 5.5; если вы используете другой сервер, то можете выбирать здесь

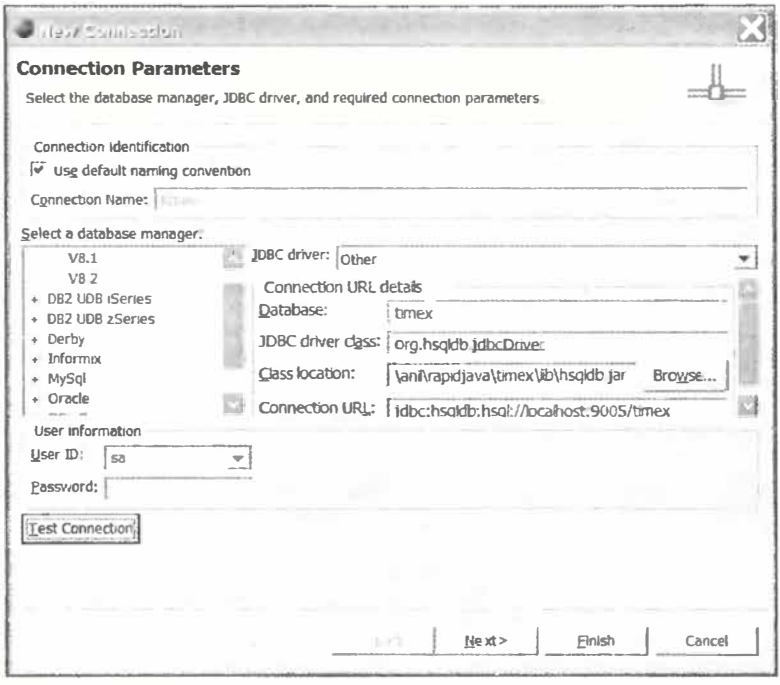



**Рис. 8.35.** Настройка HSQLDB для приложения Database Explorer

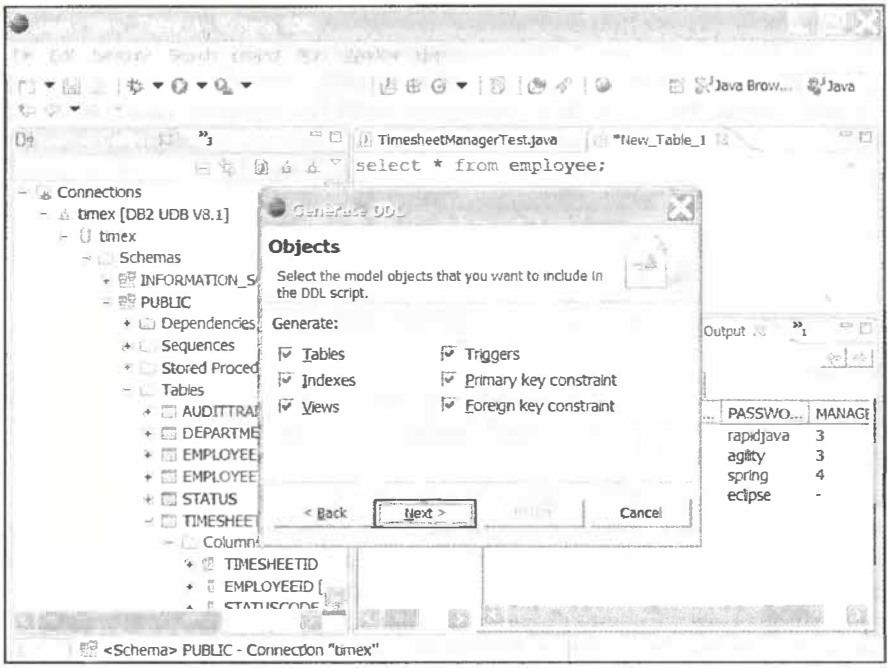



**Рис. 8.36.** Разнообразие средств, предоставляемых дополнением Data

и его (например, BEA, IBM или JBoss). На экране, представленном на рис. 8.37 (часть мастера New Server), я заметил небольшое предупреждающее сообщение, указывающее на то, что сервер Tomcat требует JDK (а не JRE). Поэтому я добавил новый JDK при помощи кнопки Installed JREs (Установленные JRE) и, после возврата к экрану мастера, выделил его в раскрывающемся списке JRE: (да, JRE и JDK могут ввести в заблуждение).

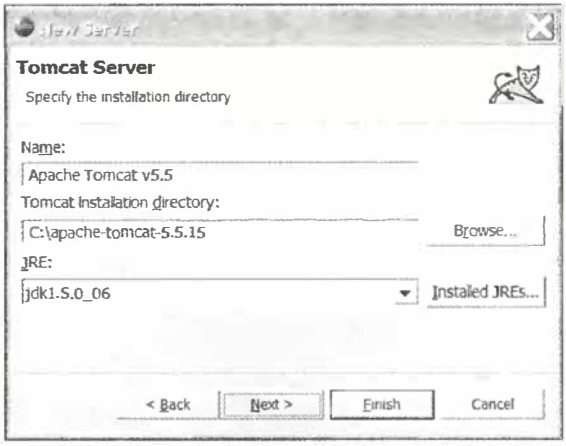



**Рис. 8.37.** Экран мастера добавления нового сервера (при добавлении сервера Tomcat)

После добавления сервера Tomcat необходимо настроить записи. (Сейчас это очевидно, но чтобы выяснить это, мне понадобилось не мало времени.)

На рис. 8.38 представлен редактор конфигурации Server Overview (да, редактор; Eclipse может иметь любое количество редакторов содержимого, причем не только для текста, но и для изображений, а также других типов содержимого). Затем сбросьте в параметрах рабочего пространства флажок Run Modules Directly (Запускать модули непосредственно), поскольку мы хотим использовать существующую конфигурацию, чтобы наша среда Ant могла продолжать функционировать. Обратите также внимание на вкладку Modules (Модули) в левой нижней части экрана (рис. 8.38); она применяется для настройки внешних модулей, Web-архивов (файлы `.war`) и рабочих каталогов Web-приложений (например, наш каталог `build/timex`).

> **Примечание**
> При разработке рабочие каталоги удобнее, чем файл `.war` (или UAT, например), поскольку вы можете быстро проверять изменения, внесенные в такие файлы, как JSP, HTML, и рисунки, без необходимости повторно развертывать все приложение (файл `.war`).

Теперь мы готовы запустить сервер Tomcat. Обратите внимание на представление Servers, расположенное в нижней части рис. 8.38. Именно здесь мы можем останавливать и запускать сервер Tomcat. Сейчас отображается состояние Started (Запущен), а все изменения, которые мы внесем в конфигурацию, отобразятся синхронно. В этом представлении мы можем запустить, остановить и контролировать сервер Tomcat. Мы можем также запустить сервер Tomcat в режиме отладки, позволяющем отлаживать серверные приложения (например, использовать контрольные точки, проверять

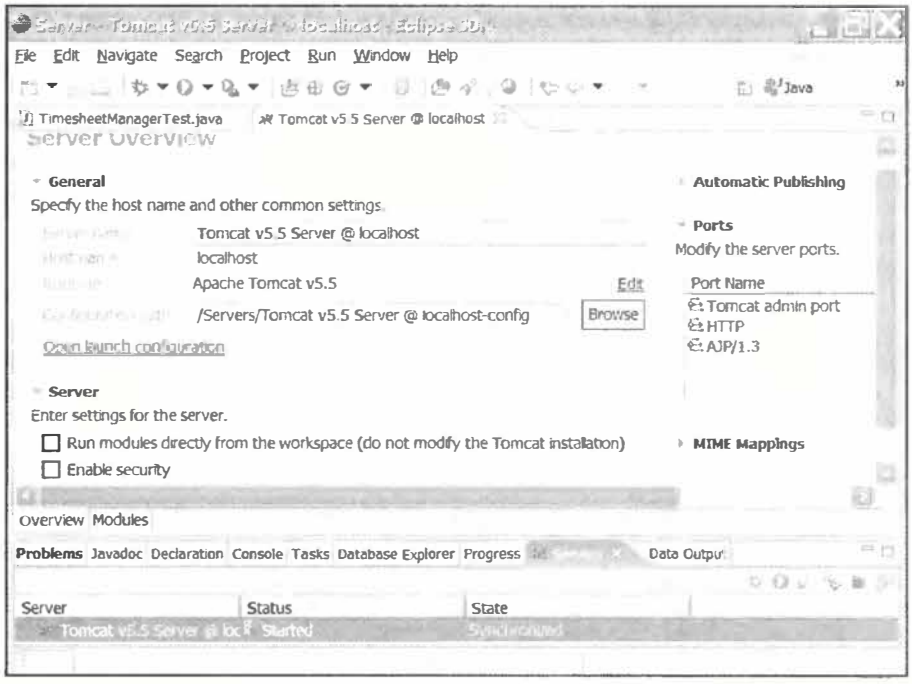



**Рис. 8.38.** Редактор конфигурации сервера Tomcat

содержимое переменных и так далее; более подробно отладка рассматривается в главе 9, "Регистрация, отладка, мониторинг и профилирование").

## Дополнение Hibernate

Теперь, когда дополнения от `eclipse.org` установлены и настроены, имеет смысл двигаться дальше, а именно: загрузить и установить некоторые дополнения стороннего производителя. Они могут быть установлены тем же способом, что и дополнение WTP, т.е. выбрав в меню Help (Справка) пункты Software Updates (Обновление программного обеспечения), Find (Найти) и Install (Установить) или (если вы защищены брандмауэром, например) просто распаковав соответствующий архив в каталог `/eclipse` или `/eclipse/plugins/`, в зависимости от того, как архив упакован производителем.

Hibernate предоставляет дополнение, которое облегчает создание файлов конфигурации XML и файлов связывания. Это очень удобная возможность, поскольку связывание является одной из наиболее утомительных задач при работе с технологиями ORM.

Дополнения Hibernate можно получить как в результате непосредственной загрузки архива, так и в результате обновления программного обеспечения Eclipse (я предпочитаю последнее). Чтобы получить это дополнение, следуйте инструкции, приведенной на сайте Hibernate (`http://hibernate.org/`). Во время написания этой книги я без проблем загрузил последнюю версию дополнения по адресу `http://download.jboss.org/jbosside/updates/development/`. После установки мне понадобилось перезапустить Eclipse. На рис. 8.39 демонстрируются неко-

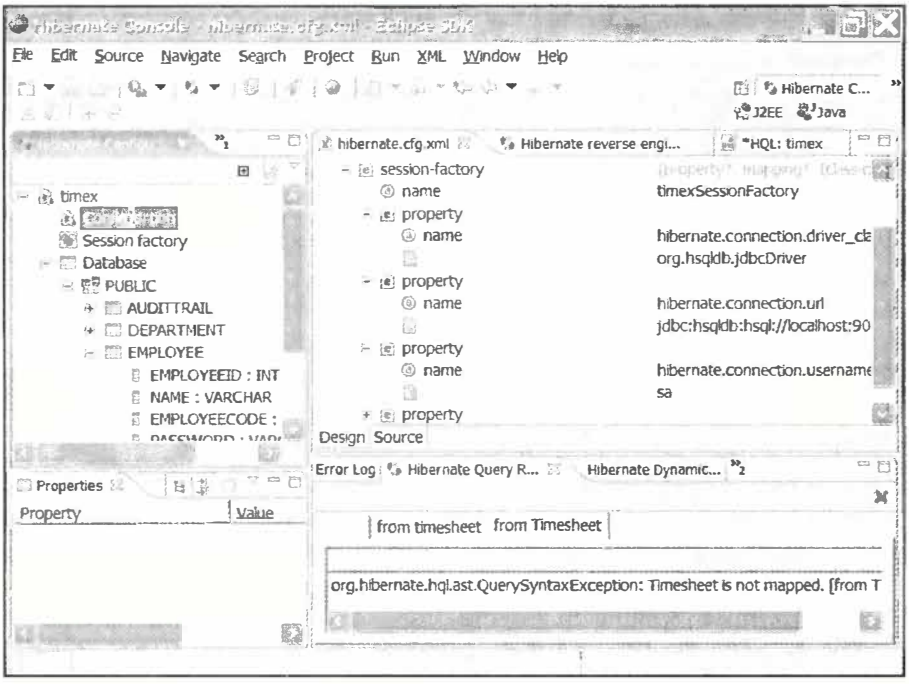



**Рис. 8.39.** Представления дополнения Hibernate

торые представления Hibernate (организованные в виде конфигурации Hibernate Console (Консоль Hibernate)). Эти представления позволяют нам создать файлы `hibernate.cfg.xml`, файлы связывания, проверить выполнение сценариев HQL и многое другое.

## Дополнение Spring IDE

И наконец, но не в последнюю очередь, мы получим дополнение Spring, известное как Spring IDE (это подчиненный проект Spring Framework). Одна из фундаментальных проблем работы со средой Spring заключается в сложности использования файлов XML для конфигурации. Дело в том, что файл XML быстро разрастается и может стать сложной задачей для редактирования и поддержки, даже если вы разделили его на несколько файлов XML контекста приложения (как рекомендует документация Spring).

Согласно Web-сайту `http://springide.org`, это дополнение следует загружать только при обновлении программного обеспечения Eclipse. Недавно я попытался установить его непосредственно (разархивировав файл архива), и это повредило мою систему, так что остерегайтесь! Чтобы получить это дополнение, строго следуйте инструкции Web-сайта Spring IDE. На момент написания этой книги мне удалось загрузить и установить это дополнение (по адресу `http://springide.org/updatesite/`) за несколько минут. Как обычно, чтобы это новое дополнение вступило в действие, я должен был перезапустить Eclipse.

> **Примечание**
> Дополнению Spring IDE необходим проект Web Standard Tools (WST); к счастью, мы уже установили его.

После того как это дополнение будет успешно установлено, щелкните в представлении Package Explorer правой кнопкой мыши на проекте `timex` и в появившемся контекстном меню выберите пункт Add Spring Project Nature (Добавить характер проекта Spring), как показано на рис. 8.40. Ниже приведена цитата, взятая непосредственно из справочной документации Eclipse:

"*Характер* (nature) проекта позволяет дополнению получить проект определенного вида. Например, инструменты разработки Java (JDT) используют "характер Java" для придания проектам поведения, характерного для языка Java. Характер проекта определяется дополнениями и обычно добавляется и удаляется для каждого проекта "индивидуально", когда пользователь выполняет некоторое действие, определенное дополнением."

После того как проекту `timex` присвоен характер Spring, необходимо сделать следующее.

- Добавьте представление Spring Beans (Компоненты Spring).
- Используйте это представление для добавления нашего файла контекста приложения `timex-servlet.xml` (разработанного в главе 7, "Среда Spring Web MVC Framework"). Для этого достаточно щелкнуть правой кнопкой мыши на имени проекта (оно расположено в верхней части данного представления).
- Щелкните правой кнопкой мыши на элементе XML в представлении Spring Beans и в появившемся контекстном меню выберите пункт Show Graph (Показать схему); результат представлен на рис. 8.41.

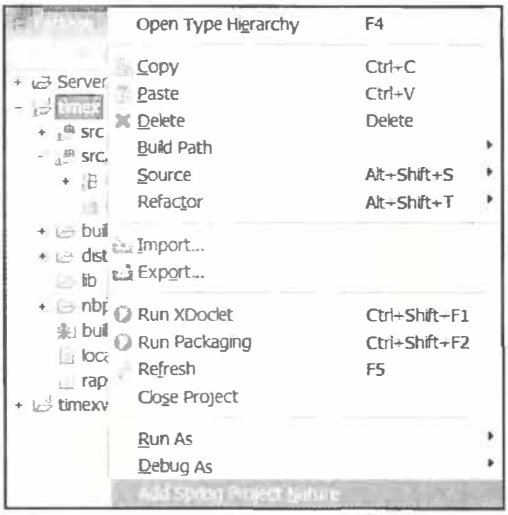



**Рис. 8.40.** Пункт меню Add Spring Project Nature

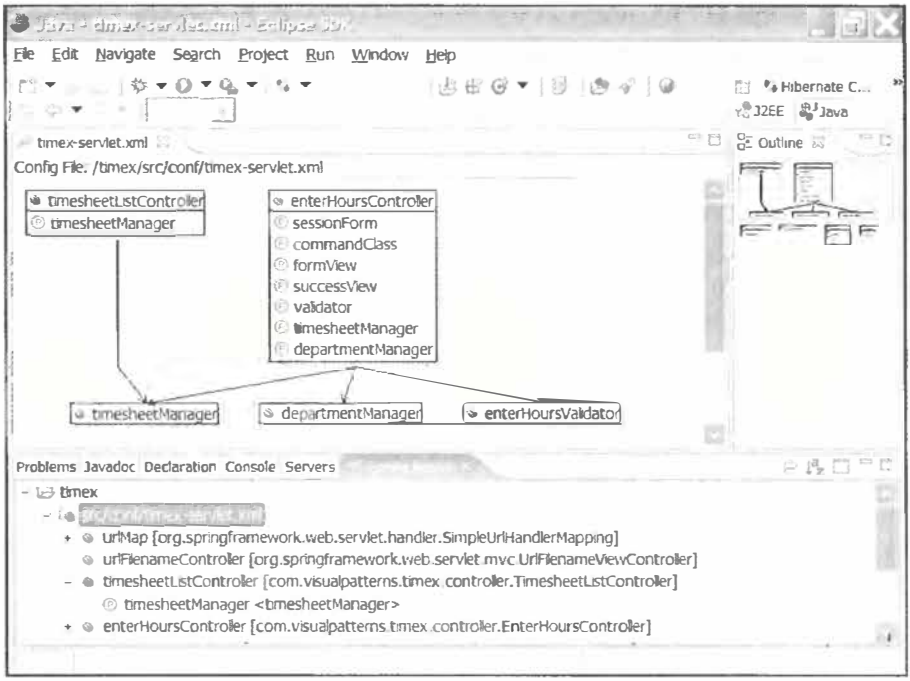



**Рис. 8.41.** Схема компонентов Spring

Теперь, для перемещения по нашему файлу XML, мы можем использовать и схему, и древовидное представление! Но побеждает использование текстового редактора. Кроме того, представление Spring IDE жестко интегрировано с другими представлениями Eclipse, таким как представление Problems, отображающим ошибки при проверке допустимости файлов контекста приложения Spring, связанных с IDE Spring. Обратите также внимание на представление Outline (Иерархическая структура) на рис. 8.41; оно демонстрирует упрощенную версию схемы, на тот случай, если ваши схемы станут слишком большими и сложными. Дополнение Spring IDE укомплектовано интерактивной справочной системой, к которой можно обратиться, выбрав в меню Eclipse Help (Справка) пункт Help Contents (Содержимое справки).

### Другие популярные дополнения WTP

В этом разделе мы рассмотрели те дополнения, которые будут применены к приложению Time Expression. Однако проект WTP предоставляет дополнения, редакторы, представления и мастера для множества других аспектов разработки приложения JEE, поддержки проектов EJB, проектов клиентских приложений, Web-служб, соединителей и многого другого. Все они без проблем доступны в конфигурации JEE и соответствующих представлениях.

## Еще об Eclipse? Да, дополнений в изобилии!

Как я упомянул в начале этой главы, при написании этой книги я сделал запрос на `google.com` для слов "eclipse" и "plug-in", получив в результате миллионы ссылок! Хоть это и не свидетельствует о точном количестве дополнений для Eclipse, но должно дать вам представление о том, насколько активно сообщество Eclipse.

### Проекты `eclipse.org`

Новичкам имеет смысл искать новые дополнения в первоисточниках, как мы делали с дополнениями WTP. Кроме того, напомню о списке проектов Eclipse, который я привел в табл. 8.1 ранее.

### Каталоги дополнений

На Web-сайте `eclipse.org` доступно множество дополнений для разных проектов Eclipse, которые я упомянул ранее в этой главе. Кроме того, ниже приведены два каталога дополнений Eclipse, которые стоит посетить, поскольку они предоставляют большое количество дополнений, как бесплатных, так и коммерческих:

- `http://www.eclipseplugincentral.com/`
- `http://eclipse-plugins.2y.net/eclipse/`

Например, на Web-сайте `eclipseplugincentral.com` приведены сотни дополнений, организованных в следующие категории: Application Management (Управление приложением), Application Server (Сервер приложений), Code Management (Управление кодом), Database (База данных), Deployment (Развертывание), Documentation (Документирование), Editor (Редактор), Entertainment (Развлечение), Graphics (Графика), IDE, J2EE Development Platform (Платформа разработки), J2ME,

Languages (Языки), Modeling (Моделирование), Network (Сеть), Other (Другое), Profiling (Профилирование), Rich Client Applications (Улучшенные клиентские приложения), SCM, Source Code Analyzer (Анализатор исходного кода), Team Development (Групповая разработка), Testing (Проверка), Tools (Инструменты), UI, UML, Web, Web Services (Web-службы) и XML.

Сайт `plugins.2y.net` предоставляет даже больше категорий. Вот лишь некоторые из них: Ant, AspectJ, Bug Tracker (Отслеживание ошибок), Business Process Tools (Инструменты для коммерческой деятельности), Code Generation (Создание кода), Code Generation/Modeling (Создание кода и моделирование), Code mngt (Код mngt), Com, CORBA/IDL, Database (База данных), Database Persistence (Отказоустойчивость баз данных), Decompiler (Декомпилятор), Deployment (Развертывание), Distribution Package (Распространение пакетов), Documentation (Документация), Entertainment (Развлечение), J2EE development platform (Платформа разработки J2EE), Languages (Языки), LDAP, Logging (Регистрация), Misc, Mobile/PDA, Modelling (Моделирование), Network (Сеть), Obsolete (Устаревшие), Patterns (Схемы), Profiling (Профилирование), Project management (Управление проектом), Report (Отчет), Rich Client (Улучшенный клиент), RSS, SAP, SCM, Source Code Analyzer (Анализатор исходного кода), Source Code Formatter (Форматирование исходного кода), Team (Группы), Testing (Проверка), Tomcat, Tools (Инструменты), Tutorial (Руководства), UI, UI components (Компоненты пользовательского интерфейса), UML, Web, Web Service (Web-служба) и XML.

## Web-сайт `MyEclipseIDE.com`

Как вы уже, вероятно, заметили, в этой книге я не упоминаю многих коммерческих продуктов. Это вовсе не потому, что я фанатик реализации с открытым исходным кодом, на самом деле я широко использую коммерческие продукты в своей работе. Однако продукты с открытым исходным кодом легко доступны всем для загрузки (и бесплатны), а следовательно, при желании вы сможете опробовать примеры этой книги, не разоряясь на коммерческие продукты. Но я должен упомянуть один уникальный продукт, доступный на Web-сайте `http://myeclipseide.com`. Его стоит посетить, особенно если вам необходимо универсальное решение для дополнений по разработке Java, здесь всем есть место под солнцем (так сказать).

Web-сайт `MyEclipseIDE.com` предоставляет такие категории, как modeling and code generation (UML) (Моделирование и создание объектного кода), web development tools (struts, spring, jsf, hibernate, DB, tapestry) (Инструменты разработки Web), productivity wizards (Проекты Web/EJB) (Мастера производительности), application server integration (JBoss, WebLogic, Websphere и т.д.) (Интеграция сервера приложений), packaging and installation (installer и т.д.) (Упаковка и установка) и т.д.

Web-сайт `MyEclipseIDE.com` не следует считать обязательным, особенно когда такой проект Eclipse, как Web Technologies Project (WTP), предоставляет и так немало возможностей. Кроме того, вы вполне можете найти и другие дополнения, например дополнения Hibernate от `http://hibernate.org` и Spring IDE от `http://springide.org`. Но если вы хотите остановиться там, где сможете найти множество тематически организованных дополнений на одном месте (за низкую цену), причем с поддержкой, то сайт `myeclipseide.com` может вам пригодиться.

### Web-сайт google.com

Безусловно, Web-сайт google.com — это самый большой каталог в мире на настоящий момент. На google.com вы можете найти практически все, что пожелаете. Например, я без проблем нашел EclipseUML (http://eclipseuml.com) при поиске слов "eclipse" и "uml".

## Поддержка группы Eclipse

Инструменты Eclipse поставляются со встроенным клиентом CVS (Concurrent Versions System — система управления параллельными версиями). Если вы работали с CVS прежде, то, вероятно, знакомы с серверами CVS и различными командами CVS. Среда Eclipse предоставляет две конфигурации для работы в группах: CVS Repository Exploring (Обзор хранилища CVS) и Team Synchronization (Синхронизация группы). Эти конфигурации содержат логическое объединение ряда представлений, таких как CVS Repositories (Хранилища CVS) и Synchronize (Синхронизация). Например, на основании инструкций CVS для Spring Framework я смог подключиться к хранилищу CVS среды Spring Framework и просмотреть его каталог CVS внутри Eclipse (как показано на рис. 8.42)!

Возможность синхронизации позволяет нам синхронизировать наш локальный каталог с сервером CVS. Например, я могу создать новый проект и отметить весь каталог CVS Spring, используя мастер синхронизации Eclipse. Впоследствии, когда я ре-

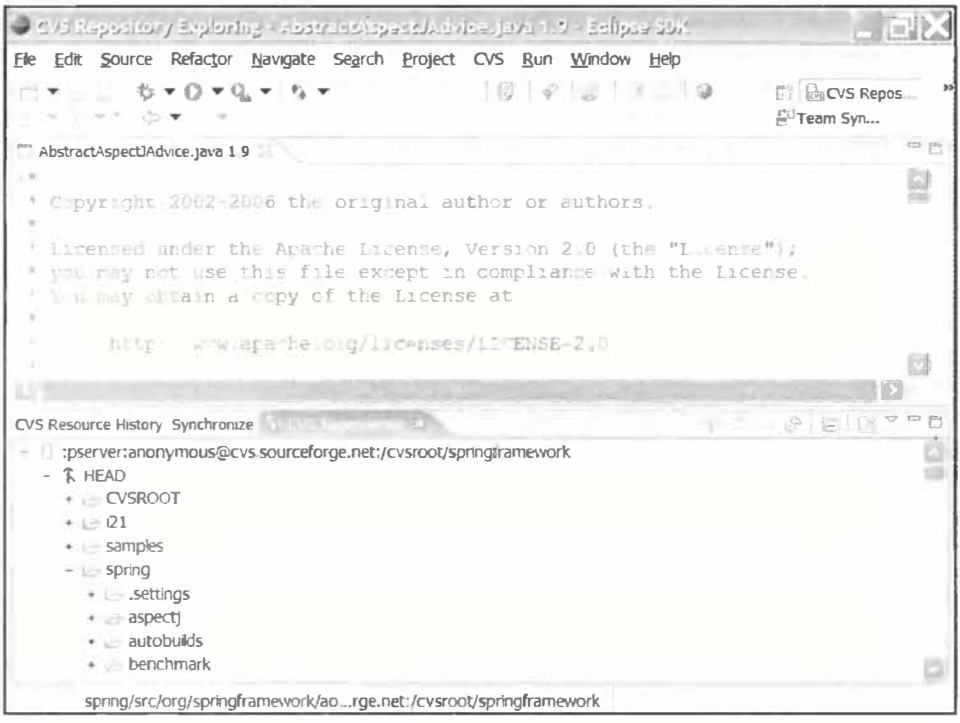



**Рис. 8.42.** Просмотр хранилища CVS

дактировал выбранный файл .java Spring, я был способен щелкнуть правой кнопкой мыши и получить контекстное меню Team (Группа), представленное на рис. 8.43. Обратите внимание на разнообразие доступных здесь параметров CVS, например Update (Обновления), Commit (Подтверждение) и т.д.


**Рис. 8.43.** Контекстное меню Team

# Справочная система Eclipse

Среда Eclipse обладает справочной системой на базе HTML, обеспечивающей все возможности для поиска. Среда Eclipse поставляется укомплектованной также справкой для инструментальных средств Workbench, JDT, JEE и многих других. Фактически, если вы следовали примерам этой главы, то, вероятно, уже видели документацию Spring (как показано на рис. 8.44), Hibernate, Web-инструментов и т.д. Кроме того, Eclipse предоставляет контрольные списки (Cheat Sheets), которые сопровождают вас на протяжении всего процесса. Например, рис. 8.45 демонстрирует список с инструкцией о том, как создать простое приложение Java от начала до конца. Вы можете также использовать Eclipse для поддержки контекстно-зависимой документации Javadocs, как показано на рис. 8.46. Для доступа к этой справке достаточно навести курсор мыши на метод, и появится справка Javadoc для данного метода. Впоследствии, если мы захотим перейти к этому окну, достаточно будет нажать клавишу <F2>. Чтобы задействовать эту возможность, вам придется присоединить исходный код API (как я объясняю в разделе "Советы и приемы Eclipse" далее в этой главе).

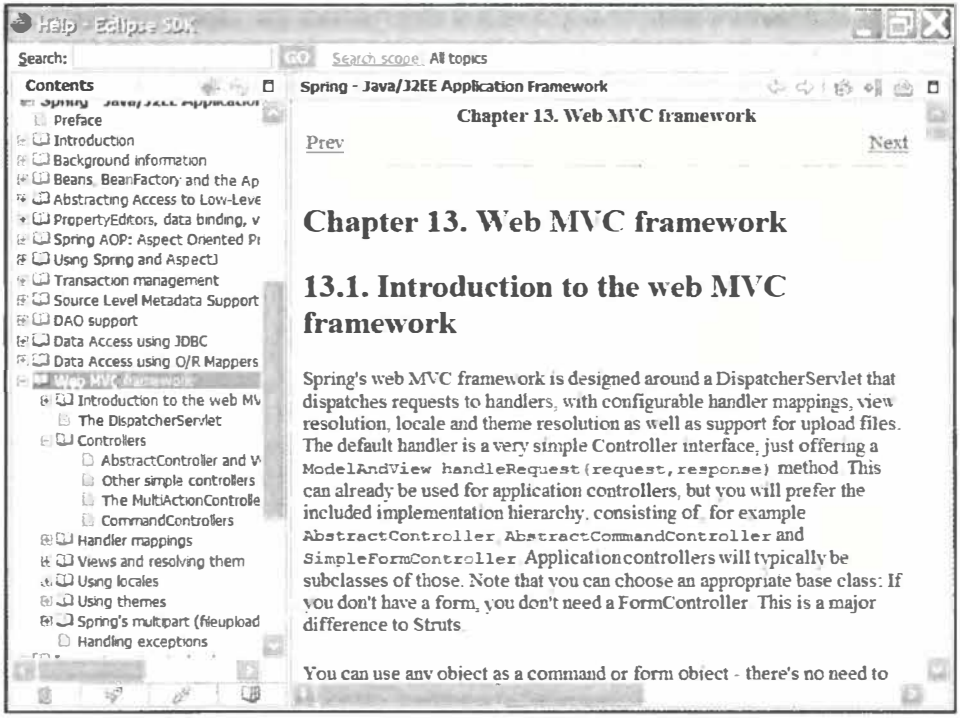



**Рис. 8.44.** Справка Eclipse

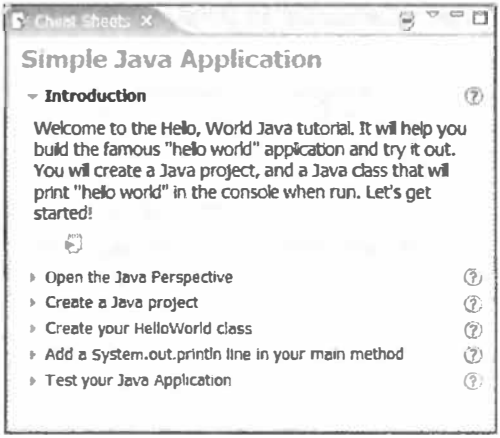



**Рис. 8.45.** Окно Cheat Sheets

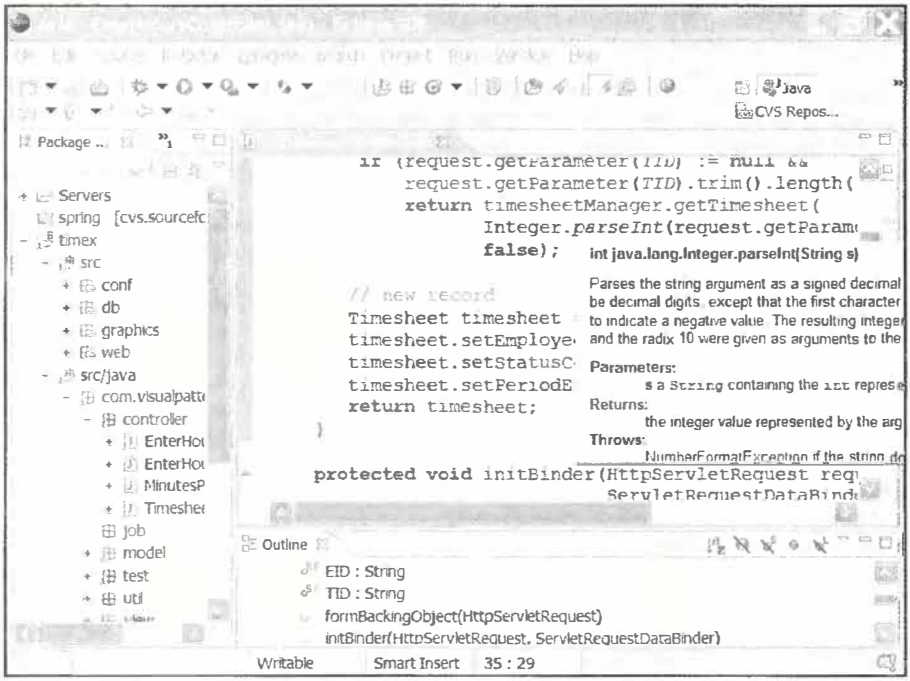



**Рис. 8.46.** Контекстно-зависимая документация Javadocs

**Личное мнение: инструменты разработки GUI, сражение началось!**

В мае 2001 года я написал статью для журнала JavaWorld.com под названием "Does Sun Understand GUI Design?" (Разбирается ли Sun в проектировании GUI?). В первую очередь, эта статья была о сложности Java Swing, заумности API и низкой производительности (некоторые из этих недостатков остались до сих пор). Однако я не обошел также вниманием корневые каталоги Unix от Sun Microsystem, которые, по моему мнению, отражают состояние инструментальных средств Java до появления SDK Eclipse, NetBeans и таких коммерческих продуктов, как IntelliJ IDEA от JetBrain.

Прежде чем я оскорблю пользователей Unix, позвольте мне сказать, что начиная с 1990 года я работал над больше чем шестью разновидностями операционных систем Unix/Linux (Solaris, AIX, Irix, AT&T System V, SCO Unix, Red Hat и др.). Фактически, я могу смело заявить, что почти в 100% проектов, над которыми я работал лично, в некоторой форме использовались серверы Unix (прежде всего, Solaris). Я также использовал инструментальные средства Unix на Windows (например, MKS Tools и Cygwin). Некоторые из моих коллег иногда критикуют меня за выражения, используемые мной в редакторе Vi и сценариях оболочки Unix, которые кажутся им головоломкой, когда они пытаются разобрать их. Другими словами, мне нравятся гибкие и мощные инструменты командной строки Unix, сложные регулярные выражения, сценарии оболочки и другие средства, которые предоставляет Unix (особенно для повторяемых задач, которые могут быть повторно выполнены при помощи сценариев). Однако в недавних 80-х годах я работал также с платформами GUI, такими как Microsoft Windows, начиная с версии 2.1 (известной также, как Windows/286 и Windows/386), Mac OS X и X-Windows. Таким образом, я не избегаю инструментов GUI, поскольку при правильном использовании они способны сэкономить много времени, предоставляя единую среду, причем Eclipse — совершенный пример этого. Инструментальные средства GUI также имеют некоторые ограничения, которые время от времени проявляются, вот почему в первых главах этой книги я использовал командную строку.

С учетом моих смешанных чувств к командной строке и GUI, я всегда считал, что корпорация Sun ничего не понимает в GUI и инструментальных средствах GUI, даже при том, что группа Swing была сформирована с участием некоторых инженеров из Apple и Nextstep. Корпорация Microsoft, с другой стороны, всегда оказывалась на один шаг впереди, когда дело доходило до инструментов GUI, возможно, потому, что они буквально вкладывают капитал в GUI (миллиарды долларов), их операционная система и комплект приложений ориентированы на GUI (в отличие от компаний, ориентированных на Unix/Linux, в основе которых лежит интерфейс на базе командной строки).

Хочу представить вам еще одну точку зрения относительно непонимания корпорацией Sun проблем GUI, поэтому потерпите меня еще немного. Несколько лет назад я имел пятиминутный диалог с Джеймсом Гослингом на конференции JavaOne. Он сидел рядом со мной в пресс-центре, когда мы проверяли свою электронную почту (там был ряд компьютеров, доступных для всех, кто хотел проверить электронную почту). Естественно, я должен был что-то сказать ему, раз уж представилась такая редкая возможность. Я пожаловался ему, что корпорации Sun Microsystems стоило бы заняться созданием инструментальных средств GUI (оболочки) для наиболее широко используемых утилит Unix на Solaris, таких как ls, cron, find, grep и многих др. Как я упоминал, инструменты GUI были одним из ключевых факторов популярности Microsoft Windows NT (ныне Windows 200*x*). Его ответ свелся к тому, что клиенты Sun "предпочитают командную строку" (Хммм ..., ну что можно ответить на такой комментарий, причем исходящий от отца Java?).

Прежде чем появился язык Java, я использовал для работы Unix, но я также написал немало кода под Microsoft Windows, используя Microsoft Visual C++, а еще раньше Borland C++. Я не мог понять, почему мы, имея вполне стабильные продукты и прекрасные отладчики, перешли затем на менее надежные технологии, такие как Java. Тем не менее, чтобы закончить эту длинную историю, скажу, что теперь это уже не так. Такие продукты, как Eclipse, NetBeans и IntelliJ, прошли длинный путь совершенствования.

В завершение позвольте мне пояснить, почему я так доволен тем, что организация Eclipse Foundation сосредоточилась на инструментальных средствах GUI. И наконец, если бы организации, ориентирующиеся на Apache, тоже занялись инструментами GUI и сообща писали дополнения, только вообразите возможности!

Феномен Eclipse — это то, чего я не видел уже очень давно и что очень близко тому, что я предвидел более десятилетия назад, когда открыл для себя язык Java и программирование для Web. Платформа Eclipse выравнивает счет в игре, предоставляя надежную платформу GUI, а также целый набор услуг и позволяя таким образом производителям инструментальных средств сосредоточиться на вспомогательных возможностях, а их количество бесконечно. До меня дошли слухи, что имя Eclipse имеет скрытый смысл "the Eclipse of the Sun" (дословно, с намеком: затмение солнца); так что, это атака на Sun Microsystems? Или Eclipse конкурирует с Microsoft Visual Studio? Чтобы сделать вопрос интереснее, упомяну системы Swing и NetBeans, которые существуют давно и ныне снова завоевывают популярность!

Безусловно, IntelliJ от JetBrain — это тоже фактор ...

Короче говоря, сражение IDE только что началось!

# Советы и приемы Eclipse

Существует так много советов и приемов для платформы Eclipse, что перечислить их здесь все просто невозможно! Этот список начался бы от закладок (bookmark) и перетаскивания (drag-and-drop), на Microsoft Windows, до быстрого обмена (hotswap) при отладке и эффективности в работе (working effectively). Я настоятельно рекомендую читать справочную информацию (Help) Eclipse, используя пункт меню Help Contents для доступа к различным советам и приемам, которые хорошо задокументированы.

## Комбинации клавиш

Следующий раздел приводит примеры комбинаций клавиш, используемых на Microsoft Windows. Если вы используете Eclipse на платформе Unix/Linux или Mac OS

X, эти комбинации могут несколько отличаться. Например, на Mac OS X, для вызова диалогового окна мастера нового проекта — это <COMMAND+N>, а на Microsoft Windows — это <Ctrl+N>. Обратите также внимание на то, что функциональные возможности, предоставляемые этими комбинациями клавиш, легко доступны в панели главного меню и в контекстных меню.

Ранее я уже упоминал две комбинации клавиш и напомню их здесь — это <Ctrl+Пробел> и <Ctrl+1>. Они срабатывают почти везде и всюду, предоставляя интеллектуальную, контекстно-зависимую справку (например, при автоматическом объявлении переменных). Используйте их как можно чаще (но в правильном контексте, конечно), поскольку они не только избавят от чрезмерного ввода кода вручную, но и сделают ваш код почти безошибочным!

К наиболее популярным комбинациям клавиш, на мой взгляд, относятся следующие:

- <Ctrl+M> — для минимизации и развертывания окон редакторов и представлений.
- <Ctrl+N> — для создания чего-нибудь нового при помощи одного из 100 мастеров, доступных в Eclipse.
- <Ctrl+Shift+Пробел> — предоставляет подсказки в параметрах метода.
- <Ctrl+Shift+M> — вставка отсутствующего импорта.
- <Ctrl+K> — повторить последний поиск, а <Shift+Ctrl+K> — поиск назад.
- <Ctrl+Shift+X> — преобразовывает выделенный текст в верхний регистр, а <Ctrl+Shift+Y> — в нижний.
- <Ctrl+Shift+F> — форматирует существующий файл (на основании предпочтений).
- <Ctrl +/> — комментирует одну строку кода.
- <Ctrl+F6>, <Ctrl+F7> и <Ctrl+F8> — позволяют циклически перебирать редакторы, представления и компоновки соответственно (<Ctrl+Shift+F6>, <Ctrl+Shift+F7>, <Ctrl+Shift+F8> — то же, но в противоположном направлении).
- <Tab> и <Shift+Tab> — увеличение и уменьшение отступа блока кода соответственно.
- <F3> — открывает объявление выделенного элемента.
- <F5> — обновление представления.

И наконец, но не в последнюю очередь, мать всех комбинаций клавиш — <Ctrl+Shift+L>, предназначенная для вызова справки или пунктов меню Key Assist. Этот компонент предоставляет список комбинаций клавиш, которые могут быть выполнены непосредственно из списка, представленного на рис. 8.47.

## Предпочтения

Eclipse — это высоко адаптивный инструмент. На рис. 8.48 представлен снимок экрана диалогового окна Preferences (Предпочтения), доступного при выборе в меню Windows инструментальных средств Eclipse пункта Preferences. Настраиваемых элементов конфигурации очень много. Фактически, именно это, вероятно, послужило причиной наличия параметра Filter (Фильтр) вверху окна Preferences; он позволяет ограничить набор доступных для выбора параметров.

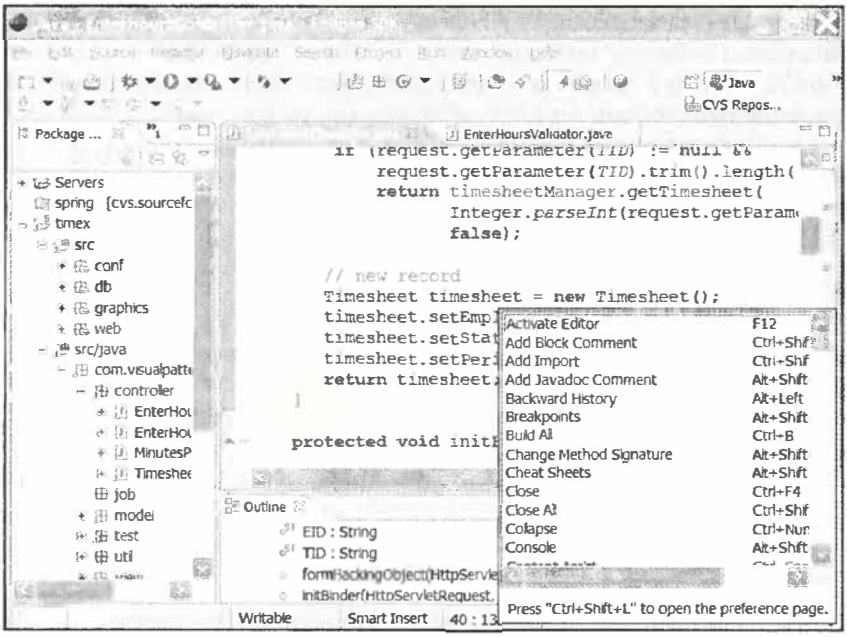



**Рис. 8.47.** Меню Key Assist

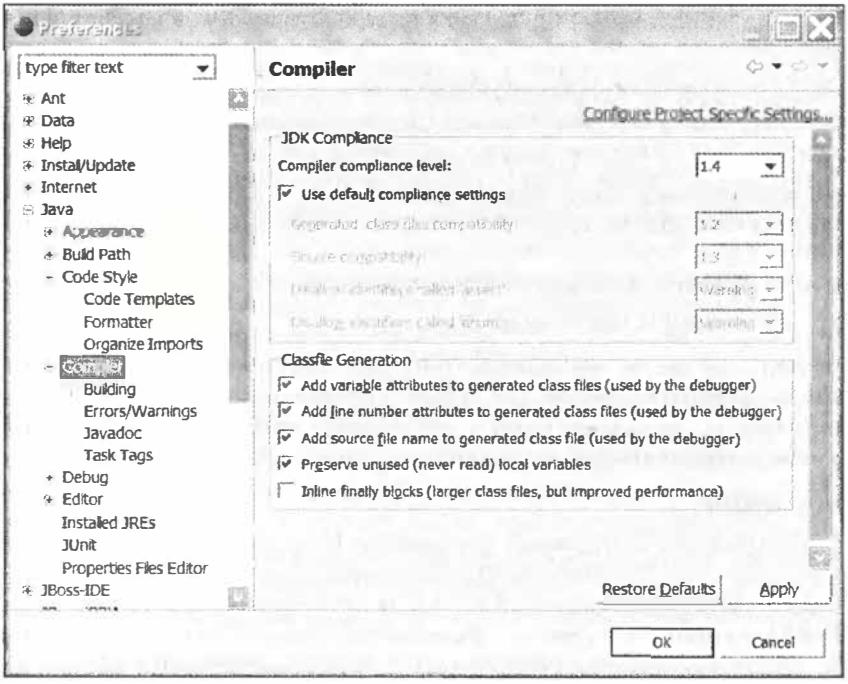



**Рис. 8.48.** Меню окна Preferences

Большинство представлений в SDK Eclipse подлежит перенастройке, при помощи их собственных панелей инструментов, причем некоторые предоставляют больше параметров настройки, другие меньше. Например, представление Package Explorer позволяет изменить компоновку и файлы фильтра для отображения (например, если вы хотите отфильтровать файлы JAR из списка или просмотреть иерархическое представление пакетов).

## Закладки

*Закладки* (bookmark) присутствуют практически в любом текстовом редакторе, используемом для разработки кода. Eclipse не является исключением в этом отношении. Пункт Add Bookmark (Добавить закладку) меню Edit (Правка) позволяет добавить закладку. Представление Bookmarks (Закладки) позволяет просматривать и перемещаться по закладкам, как показано на рис. 8.49.

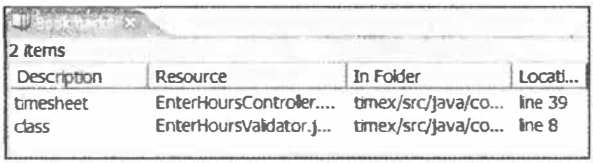



**Рис. 8.49.** Представление Bookmarks

## Запуск внешних инструментов и Web-браузера

С помощью пункта External Tools (Внешние инструменты) меню Run (Пуск) вы можете легко запускать внешние программы. Кроме того, использовав пункт Web Browser (Web-браузер) меню Window (Окно), вы можете также вызывать Web-браузер по вашему выбору. Например, рис. 8.50 демонстрирует диалоговое окно Run; обратите внимание на различные типы программ, которые мы можем запускать, и различные вкладки, предназначенные для настройки запускаемой программы (параметров командной строки, например).

## Локальная история

Локальная история (Local History) позволяет нам сравнивать и восстанавливать многие элементы истории файла. К ней можно обратиться в ходе редактирования файла, используя контекстное меню (щелчок правой кнопкой мыши).

## Восстановление конфигурации

Если вы случайно измените предопределенную конфигурацию окон, а затем захотите вернуться к исходной конфигурации, то всегда сможете сделать это, выбрав в меню Window (Окно) пункт Reset Perspective (Восстановить конфигурацию).

## Копирование элементов

Когда мы копируем строку кода из одного файла в другой, JDT автоматически копирует и вставляет в результирующий файл исходного кода и импортируемые ресурсы, необходимые для этого кода.

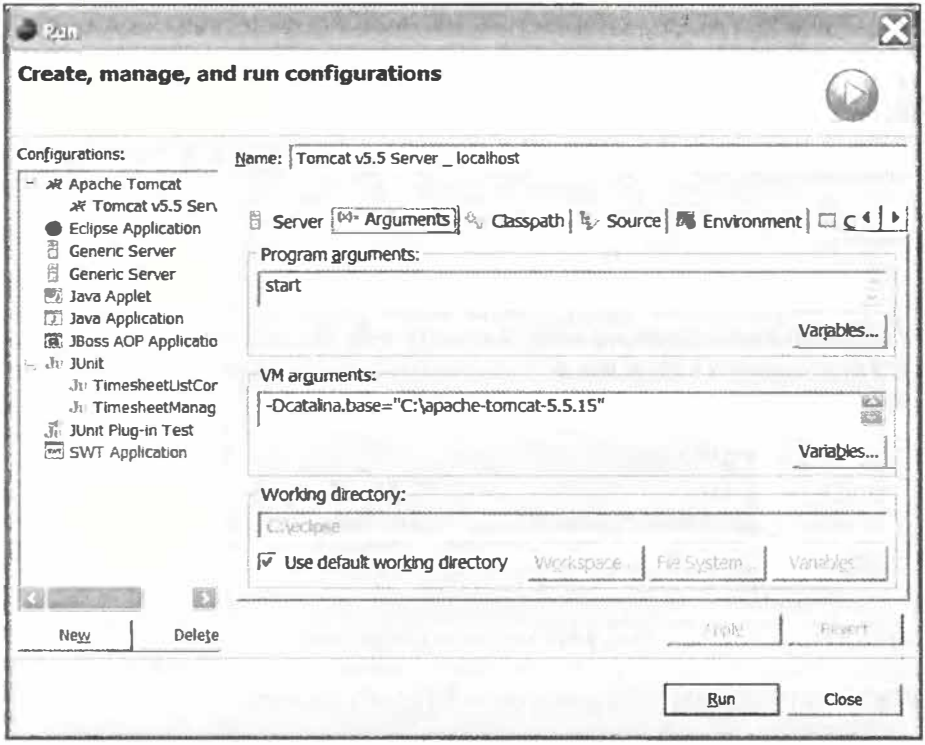



**Рис. 8.50.** Диалоговое окно Run

## Мастер очистки

Если вы полагаете, что ваши откомпилированные файлы `.class` не синхронизируются Eclipse, или вы видите красные сообщения об ошибке неожиданного содержания, попробуйте запустить мастер Clean (Очистка), выбрав в меню Project (Проект) пункт Clean (Очистка).

## Преобразование разделителей

Если вы работаете с файлами одновременно на платформе Microsoft Windows и Unix/Linux, то вам, вероятно, понадобится осуществлять преобразование разделителей строк. Это можно легко сделать, используя в меню File (Файл) пункт Convert File Delimiters To (Преобразовать разделители файла в).

## Параметры запуска Eclipse/JVM

Заслуживает внимания еще одна вещь, без которой Eclipse прекрасно обходится в большинстве случаев. Но если вы столкнетесь с ошибками из-за нехватки памяти (в очень больших проектах), то можете задать дополнительные параметры для Eclipse и виртуальной машины (VM) Java, которая запускает Eclipse (например, чтобы увеличить размер распределяемой памяти). Ниже приведен пример команды запуска

Eclipse на Microsoft Windows, отдаваемой в командной строке (более подробная информация о параметрах приведена в справочной документации):

```
\eclipse\eclipse.exe -vm "c:\Program Files\Java\jre1.5.0_06\
↳bin\javaw.exe". -vmargs -Xmx512m
```

За дополнительными параметрами обратитесь к документации Eclipse; например, параметр `-refresh` позволяет осуществить глобальное обновление рабочего пространства при запуске.

## Просмотр исходного кода стороннего производителя

Вы можете просматривать (и отлаживать) исходный код файлов JAR, присоединив их к файлу исходного кода или каталогу. Один из способов присоединения к исходному коду — это дважды щелкнуть в представлении Package Explorer на файле `.class` (в файле JAR), а затем щелкнуть на кнопке Attach Source Code (Присоединить исходный код). На рис. 8.51 приведен пример, демонстрирующий, как я смог просмотреть код JDK после присоединения библиотеки `lib` JRE в представлении Package Explorer к файлу `C:/Program Files/Java/jdk1.5.0_06/src.zip`.

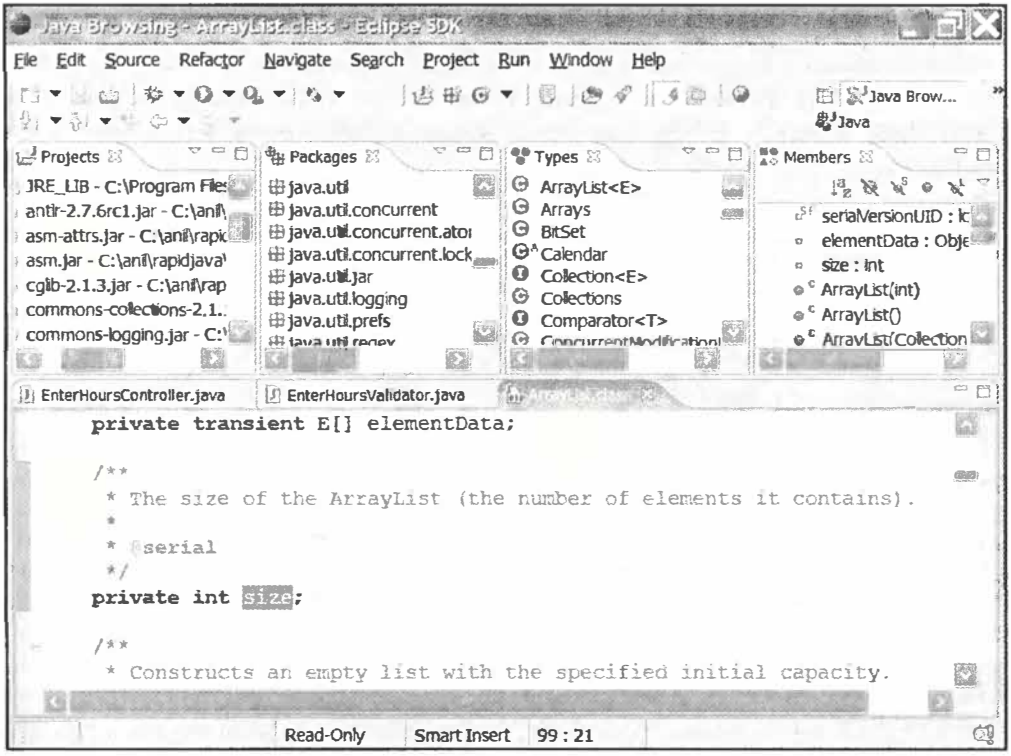



**Рис. 8.51.** Просмотр исходного кода стороннего производителя

### Скрытые файлы Eclipse

Если вы испытываете некоторые загадочные затруднения с вашим проектом Eclipse (лично я с таким никогда не сталкивался, между прочим), то вы могли бы попытаться удалить проект из каталога Eclipse и снова воссоздать его. Кроме того, в каталоге проекта Eclipse создает несколько скрытых файлов и каталогов, например `.classpath`, `.project` и `.metadata`. Вы можете удалить их, в случае если вы намереваетесь удалить, а затем воссоздать ваш проект.

## Удаление Eclipse

Если вы когда-либо решите удалить Eclipse, то вам достаточно будет просто удалить каталог `/eclipse`. (Здесь нет никакой специальной программы удаления, по крайней мере в версии 3.1.2, которая использовалась в этой главе.) Если вы используете стандартное рабочее пространство (т.е. не создавали собственное рабочее пространство при запуске Eclipse), то могли бы резервно скопировать ваши файлы проекта перед удалением каталога `/eclipse`, поскольку файлы проектов, принадлежащих стандартному рабочему пространству, также хранятся в каталоге `/eclipse`.

## Сравнение IntelliJ и NetBeans

К двум другим весьма популярным IDE языка Java относятся IntelliJ от JetBrain и NetBeans от Sun Microsystem. Я решил установить их и поработать с каждым приблизительно 30 минут, чтобы выяснить, насколько быстро я смогу привыкнуть к их IDE, по сравнению с Eclipse.

> **Примечание**
> Оговорка: позвольте мне заявить открыто, что мое мнение здесь не беспристрастно по отношению к Eclipse, с учетом всего, что я упомянул в этой главе до сих пор. Я не уверен, насколько значителен этот раздел и насколько справедливо судить о продукте за 30 минут знакомства. Однако я надеялся получить некоторые первые впечатления, а также увидеть, удастся ли мне воссоздавать проект Time Expression в другом IDE.

### Система IntelliJ 5.0

Сначала я решил опробовать IntelliJ 5.0 от JetBrain, поскольку слышал от многих, что это замечательный продукт. Загрузка IntelliJ прошла очень быстро, а установка очень гладко. Но с запуском дело обстояло не так. Система IntelliJ почему-то хотела работать как сервер, и мой брандмауэр обнаружил это. Мне понадобилось время, чтобы выяснить, как установить проект. Установка Tomcat также не была дружественной. Даже установку JDK пришлось провести вручную. Интерфейс IntelliJ, в целом, простым мне не показался, и он медленнее, чем у Eclipse. На рис. 8.52 представлен снимок экрана, на котором файлы приложения Time Expression загружены в IntelliJ. Мне понравилась возможность разделения окон, которая позволяет вам просматривать тот же файл в нескольких окнах. Это похоже на возможность разделения окна в Microsoft Word, но существенно мощнее. Напомню, что я пристрастен к Eclipse, а также то, что сообщаю вам лишь первые впечатления.


**Рис. 8.52.** Интерфейс IntelliJ 5.0 от JetBrain

## Система NetBeans 5.0

Затем я решил установить систему NetBeans 5.0 от Sun Microsystem. Установка прошла очень просто. Мне удалось также создать проект Time Expression из существующего файла `build.xml`, точно так же как в Eclipse. Интеграция с сервером Tomcat также оказалась очень полной и простой (фактически, немного лучше, чем у Eclipse). В комплекте имелась даже версия сервера Tomcat, но я добавил к NetBeans прежний сервер, что позволило мне указать на существующую установку. С учетом этого, мне удалось запустить сервер Tomcat непосредственно после настройки, и файл `timex.war` был развернут в NetBeans (рис. 8.53). Мне также удалось немедленно проверить мой код JUnit в единой конфигурации.

Я должен сказать, что IDE NetBeans также сделан очень хорошо, мне удалось осуществить загрузку, установку, компиляцию, проверку модуля и развертывание приложения за 30 минут, причем без единого обращения к интерактивной справочной системе! В целом, было забавно использовать этот изящный IDE; возможно, потому, что все срабатывало сразу.

Несмотря на то что система NetBeans была установлена быстрее и ее IDE очень хорошо проработан, для Eclipse доступно множество дополнений, в результате чего первенство достается Eclipse, а NetBeans остается вторым. На момент написания этой книги, как я упомянул ранее, в одном из каталогов Eclipse содержалось более 1 075 дополнений, в то время как сайт NetBeans насчитывал лишь 46 дополнений. Кроме того, когда я осуществлял на `google.com` поиск для слов “netbeans” и “plugin” (или

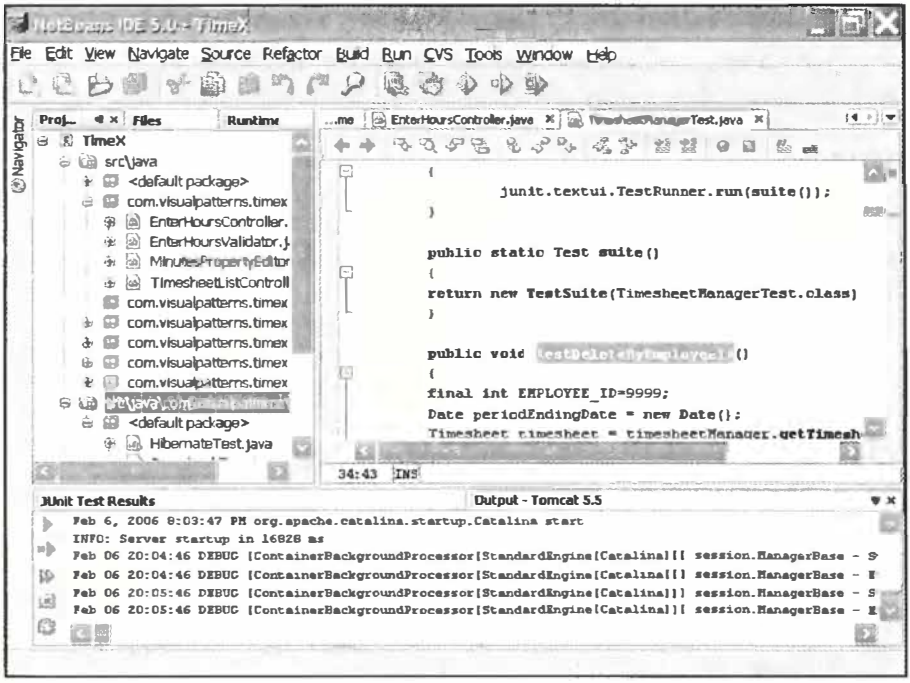



**Рис. 8.53.** Интерфейс NetBeans 5.0 от Sun Microsystem

"plug-in"), получил меньше 500 000 ссылок; а для Eclipse, как я упомянул в начале этой главы, я получил миллионы ссылок!

## Время запуска

Я заметил, что запуск SDK Eclipse происходит быстрее, но я хотел получить некие реальные экспериментальные числа, а не формальное "дольше-быстрее", поэтому я уделил каждому IDE несколько минут и засек время запуска и загрузки проекта Time Expression. Я перезагружал компьютер, чтобы ни один из IDE не использовал файлы, кэшируемые в памяти (это создает одинаковые условия для всех). Время запуска каждого IDE приведено ниже.

- SDK Eclipse – 19 секунд.
- IntelliJ – 1 минута 5 секунд.
- NetBeans – 42 секунды.

Кстати, я использовал компьютер Intel Pentium M с процессором 750 (1,86 ГГц, 2 Мбайт кэш, 533 МГц FSB), с одним гигабайтом разделяемой памяти (DDR2, 533 МГц), с операционной системой Microsoft Windows XP Media Center Edition 2005. Кроме того, время запуска, приведенное для Eclipse, включает загрузку всех дополнений, описанных в этой главе (без некоторых или всех дополнений, время запуска, вероятно, окажется меньше). Напомню, что мои представления небеспристрастны; однако от этих чисел не спрятаться. Фактически, я перезагрузил мою машину три

раза, по различным причинам, чтобы гарантировать каждому IDE равные условия при проверке. Вы можете провести подобную проверку IDE, если не доверяете моему мнению.

# Резюме

В этой главе мы изучили следующее.

- Организация Eclipse Foundation
- Платформа и проекты Eclipse
- Концепции SDK Eclipse
- Установка Eclipse
- Настройка Eclipse для приложения Time Expression
- Дополнение Java Development Tools (JDT)
- Проект Eclipse Web Tools Platform Project (для поддержки Tomcat, редактирования JSP и другой поддержки JEE)
- Применение Eclipse для приложения Time Expression (запуск Tomcat и подключение к HSQLDB, например)
- Поддержка группы Eclipse, использующей CVS
- Каталоги бесплатных и коммерческих дополнений
- Советы и приемы Eclipse
- Результаты 30-минутного сравнения IntelliJ и NetBeans

В этой главе мы рассмотрели много материала об Eclipse. Однако мы отнюдь не покончили с Eclipse. В следующей главе я покажу вам, как манипулировать средствами отладки Eclipse. Однако даже после этого нам останется изучить обширный материал о SDK Eclipse и дополнениях стороннего производителя. Например, мы не будем рассматривать такие темы, как проект UML2, или даже, что более важно, среду визуального редактора (Visual Editor), которая, согласно Web-сайту `eclipse.org`, является "независимой от производителя, открытой платформой разработки, предоставляющей среду для создания компиляторов GUI, а также инструментами для Swing/JFC и SWT/RCP с расширяемыми реализациями". Системы Swing и SWT, как известно, позволяют разрабатывать сложные приложения GUI. Замечательным в этой главе является то, что здесь продемонстрирована мощь Eclipse для разработки Web-приложений. Можете себе представить, сколько великолепных дополнений существует для разработки различных приложений. В сообществе Eclipse происходит так много событий, что почти невозможно представить количество типов продуктов, которые появятся в этом быстро развивающемся сообществе.

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, и конкурирующим технологиям содержится на следующих Web-сайтах:

- Сообщество Eclipse `http://www.eclipse.org/community/`
- Каталоги Eclipse `http://www.eclipseplugincentral.com/`, `http://eclipse-plugins.2y.net/eclipse/`
- Сайт MyEclipseIDE.com `http://www.myeclipseide.com/`
- Технические статьи об Eclipse `http://www.eclipse.org/articles/`
- Краткий обзор платформы Eclipse (официальный документ) `http://www.eclipse.org/whitepapers/eclipse-overview.pdf`
- Блог об уходе Варда Каннингема из Microsoft `http://blogs.msdn.com/edjez/archive/2005/10/17/so_long_Ward.aspx`
- Статья: "Eclipse and HSQLDB: Embedding a Relational Database Server into Eclipse, Part 1" `http://www-128.ibm.com/developerworks/opensource/library/os-echsql/?ca=lnxw961HSQLDB`
- Конференция EclipseCon `http://www.eclipsecon.org/`

Web-сайты альтернативных технологий:

- NetBeans от Sun Microsystem `http://www.netbeans.org/`
- IntellIJ от JetBrain `http://www.jetbrains.com/`
- Текстовый редактор программиста jEdit `http://www.jedit.org/`

# III

# Дополнительные возможности

# 9

# Регистрация, отладка, мониторинг и профилирование


**Рон:** Эй, парни, как дела? Я вижу вы весьма заняты, поэтому буду краток. Не работайте так напряженно, у вас, парни, обычное 40-часовое расписание, так что находите время на жизнь вне работы.

**Радж:** Стив занимается рефакторингом базы данных и связанного с ней кода, поскольку он нашел более простые и эффективные методы обеспечения отказоустойчивости. Я отлаживаю занудный дефект, который, полагаю, почти нашел. Наша скорость на этой итерации могла бы быть выше, но мы столкнулись с дефектом в API стороннего производителя. Однако мы могли бы попросить Сьюзен отложить одну или две пользовательские истории либо несколько задержать выпуск.

*Регистрация* (logging) – это весьма важный вопрос, которому иногда не уделяют заслуженного внимания. Регистрация применяется для отладки, поиска неисправностей, трассировки и заполнения контрольного журнала. Средства регистрации располагаются в диапазоне от простых операторов `print` до сложных баз данных и дистанционной регистрации. Поиск хорошего решения для регистрации – это ключевой вопрос, обеспечивающий такие возможности, как управление предоставляемым уровнем доступа (например, к базе данных) и возможности загрузки (например, файлов).

*Отладка* (debugging), с другой стороны, необходима при поиске ошибок в системе, нравится нам это или нет. Ошибки – это дефекты программного обеспечения. Дефекты могут проявляться по ряду причин, например, из-за логических ошибок в коде, неправильных требований, проблем данных, несоответствия внешнему интерфейсу (например, службе передачи сообщений), неправильной версии зависимых элементов (например, файлов JAR стороннего производителя), несовместимых комбинаций аппаратных средств и программного обеспечения и т.д. Принцип отладки заключается в том, чтобы найти источник ошибки и устранить его! Однако найти некоторые дефекты может оказаться сложнее, чем другие. В то время как на обнаружение и устранение одних может уйти несколько минут, другие могут занять дни, в зависимости от сложности системы. Самыми сложными, на мой взгляд, являются те ошибки, которые трудно воспроизвести или имеющие неустойчивый характер. (Они сводят меня с ума!) Дефекты различаются также по серьезности; одни незначительны (например, низкая эстетика UI), а другие – серьезны (например, обращение к неверной ячейке памяти). Некоторые ошибки могут привести к огромным финансовым потерям организации, в результате некорректной реализации функциональных возможностей. Короче говоря, дефект – это то, что приводит к сбою или не позволяет удовлетворить требования клиента. Таким образом, когда дефекты проявляются, их необходимо находить и устранять.

Отладка – это обычно процесс нахождения и устранения дефектов, хотя ее можно иногда использовать для анализа кода и обеспечения правильной работы логики. Поиск и устранение дефектов – вероятно, одна из наиболее трудных и сложных задач при разработчике программного обеспечения. Однако отладка чужого кода еще труднее. В проекте могут существовать и иные дефекты, например, связанные с нефункциональностью требований (производительностью, масштабируемостью). *Мониторинг* (monitoring) и *профилирование* (profiling) приложения могут помочь обнаружить узкие места, связанные с повышенным потреблением памяти, использованием процессора, сбором мусора и т.д. Платформа Java Platform Standard Edition (JSE) 5.0 обеспечивает дистанционный мониторинг и управление, наряду с хорошей консолью GUI, которая графически отображает информацию о контролируемом приложении.

В этой главе я надеюсь представить вам некий справочник по регистрации, отладке, мониторингу и немного по профилированию.

# Что рассматривается в этой главе

В этой главе мы рассмотрим следующую информацию о регистрации и методах отладки.

- Обзор концепций регистрации
- Регистрация при помощи пакета Jakarta Commons Logging (JCL)

- Отладка приложений Java с использованием Eclipse
- Отладка пользовательских Web-интерфейсов с использованием Firefox
- Управление и мониторинг JMX
- Профайлеры Java
- Советы и приемы

К концу этой главы вы должны будете иметь достаточно информации, чтобы эффективно применять регистрацию и методы поиска неисправностей в программах Java и пользовательских Web-интерфейсах, работая над своими проектами программного обеспечения.

# Обзор регистрации

Ранее я упомянул некоторые случаи применения регистрации, поэтому начну здесь с их краткого обзора. Итак, регистрация применяется для следующего.

- *Журнал аудита* (audit log). Это, вероятно, наиболее важный компонент регистрации, поскольку он, как правило, используется для ведения записей (аудита). Например, организация могла бы потребовать, чтобы все защищаемые операции регистрировались (например, в соответствии с правилами Sarbanes-Oxley и Basel II, принятыми в Соединенных Штатах).
- *Трассировка* (tracing). Подразумевает создание информации о том, что приложение делает в данный момент. Результат трассировки обычно передается на устройство `system.out` или в файл (запись в файл предпочтительнее). Трассировка применяется для проверки производительности, отслеживания работы приложения и т.д.
- *Сообщение об ошибках* (error reporting). Использование регистрации для сообщений об ошибках и исключений — это общепринятое дело; обычно их передают на устройство `System.err`, хотя использование устройства `System.err` для протоколирования сообщений об ошибках и не наилучшая практика. Регистрация может оказаться весьма эффективной в ходе поиска причины проблемы.

Результат регистрации может быть также сохранен в файл, для последующего анализа, в отличие от отладчиков на базе GUI (обсуждаемых далее в этой главе), которые требуют ручного вмешательства. Кроме того, отладчик GUI не всегда применим, например, при поиске проблем на сервере с *безликими* (faceless) приложениями (т.е. приложениями без пользовательского интерфейса). Регистрация воспроизводима, поскольку приложение может быть запущено повторно, чтобы повторить регистрацию. Здесь я перечислил некоторые из преимуществ, а ниже приведены некоторые недостатки регистрации.

- Операторы регистрации могут усложнить код и добавить работы разработчику (по добавлению/удалению операторов `print`). Большинство разработчиков, поместив операторы регистрации в код, забывают их удалить, а иногда вывод сообщений вообще недопустим или ненужен.
- Регистрация несколько увеличивает исполняемый код приложения.

Теперь, после краткого обзора основ регистрации, рассмотрим, как мы можем реализовать регистрацию в Java.

# Проект Jakarta Commons Logging (со средствами регистрации Log4j и JDK)

Большинство разработчиков все еще используют операторы `println` для трассировки и отладки в программах Java. Однако среда регистрации является хорошей альтернативой этому подходу, поскольку обладает преимуществом по производительности и позволяет нам выборочно включать и отключать сообщения, создаваемые при регистрации; кроме того, она предоставляет различные параметры форматирования и возможность передавать сообщения на несколько устройств, причем без необходимости добавлять дополнительный код в приложение.

Если вы работали с Java на протяжении некоторого времени, то, вероятно, слышали о системе Log4j от Apache (`logging.apache.org`) и средствах регистрации, встроенных в комплект разработчика Java Platform Standard Edition Development Kit (JDK версии 1.4 или более; `java.sun.com`). Но о чем вы, вероятно, не слышали, так это об API проекта Jakarta Commons Logging (JCL).

Чтобы не пересказывать описание, приведенное на Web-сайте проекта Commons Logging (`jakarta.apache.org/commons/logging/`), я процитирую его часть здесь, поскольку написано на самом деле хорошо:

"Пакет Logging – это очень удобный мост между различными реализациями регистрации ..., использование общей регистрации позволяет приложению сменить реализацию регистрации без перекомпиляции кода. Обратите внимание, что общая регистрация не пытается инициализировать или устранить основную реализацию регистрации, которая используется во время выполнения; это ответственность приложения. Однако большинство популярных реализаций регистрации автоматически инициализирует себя; в данном случае приложение может и не содержать никакого кода, специфичного для используемой реализации регистрации".

Как вы могли предположить, прочитав это описание, JCL поддерживает log4j и регистрацию JDK, и мы используем это здесь, поскольку это позволяет избежать использования при программировании конкретной реализации регистрации. Кроме того, JCL автоматически выясняет, какая реализация регистрации используется внутри, т.е. log4j или JDK.

Систему JCL можно загрузить по адресу `http://jakarta.apache.org/commons/logging/`. Единственный файл из этого пакета, в котором мы технически нуждаемся, – это файл `commons-logging.jar`, который должен быть помещен в ваш путь к классу. Кроме того, нам понадобится JDK 1.4 (или выше) либо файлы, связанные с log4j, а также файлы конфигурации. Здесь я продемонстрирую регистрацию на базе JDK 1.4, поскольку она встроена в JDK. Более подробная информация об установке log4j приведена на Web-сайте `http://logging.apache.org`.

## Как работает JCL

JCL – это просто API. Два ключевых компонента – это `org.apache.commons.logging.LogFactory` (конкретный класс фабрики) и экземпляр возвращаемого им объекта, который реализует интерфейс `org.apache.commons.logging.Log`. Но что на самом деле замечательно в JCL, так это то, что настройки требуется самый минимум. В большинстве случаев, если файл `commons-logging.jar` (поставляемый в комплекте JCL) указан в пути к классу, система JCL настроит себя сама! Кроме того,

стандартный класс LogFactory обеспечивает автоматический процесс розыска базовых API регистрации. Сначала осуществляется попытка найти файл log4j.jar в пути к классу; если его там нет, осуществляется попытка вернуться к системе регистрации JDK 1.4 (или более поздней версии). Но если мы используем старую виртуальную машину Java (JVM), общая регистрация вернется к стандартной, встроенной оболочке регистрации.

## Разработка с использованием JCL

После создания экземпляра интерфейса org.apache.commons.logging.Log мы сможем вызывать один из следующих методов:

- fatal(*Сообщение объекта*)
- error(*Сообщение объекта*)
- warn(*Сообщение объекта*)
- info(*Сообщение объекта*)
- debug(*Сообщение объекта*)
- trace(*Сообщение объекта*)

Эти методы содержат также дубликаты, имеющие второй параметр типа java.lang.Throwable. Например, log.fatal(*Сообщение объекта*, Throwable).

Позвольте мне продемонстрировать простой пример использования JCL. Приведенный ниже код демонстрирует простейшую программу. Я смог запустить эту программу на выполнение только при наличии указания файла commons-logging.jar в пути к классу:

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class CommonsLoggingTest
{
    private static Log log =
                LogFactory.getLog(CommonsLoggingTest.class);
    public static void main(String[] args)
    {
        log.fatal("This is a FATAL message.");
        log.error("This is an ERROR message.");
        log.warn("This is a WARN message.");
        log.info("This is an INFO message.");
        log.debug("This is a DEBUG message.");
    }
}
```

## Уровни сообщений регистрации

Обратите внимание на установку уровня регистрации для типов сообщений, которые мы хотим выводить. Как мы видели в предыдущем примере CommonsLoggingTest, существуют различные методы для необходимого нам уровня регистрации (например, fatal). Информация о назначение, формате и пороге (threshold) сообщений, созданных вашей программой, может быть затем доставлена согласно файлу log4j log4j.properties или файлу JDK logging.properties (см. детали на соответствующих сайтах).

Методы регистрации следует использовать корректно. Например, если мы хотим выводить только отладочные сообщения, то можем использовать метод `.debug` вместо метода `.info` или другого. Таким образом, когда в наборе файлов свойств пороговый уровень вывода установлен в значение `WARN` (например, в `lo4j`), наши сообщения отладки не будут выводиться на консоль. Аналогично используйте метод `fatal` для серьезных ошибок, поскольку мы не хотели бы отображать сообщения отладки, когда приложение настроено для работы, а не для отладки.

Уровень регистрации приложения — это проблема проектирования. Другими словами, вы должны решить, хотите ли вы, чтобы ваше приложение было "шумным" или "тихим"; т.е. будет ли оно сообщать только о серьезных ошибках или изрыгать наружу огромные объемы сообщений трассировки (иногда это меня просто ошеломляет, потому я предпочитаю тихую регистрацию). Ниже приведены уровни регистрации для log4j и JDK:

- log4j. `DEBUG`, `INFO`, `WARN`, `ERROR` и `FATAL` (устанавливается в файле `log4j.properties`).
- Регистрация JDK. `FINEST`, `FINER`, `FINE`, `CONFIG`, `INFO`, `WARNING` и `SEVERE` (устанавливается в файле `logging.properties` в каталоге JRE `lib/` или передается в качестве параметра `-Djava.util.logging.config.file` в JVM).

Приведенные ниже строки демонстрируют вывод класса `CommonsLoggingTest`:

```
Feb 14, 2006 4:19:55 PM CommonsLoggingTest main
SEVERE: This is a FATAL message.
Feb 14, 2006 4:19:55 PM CommonsLoggingTest main
SEVERE: This is an ERROR message.
Feb 14, 2006 4:19:55 PM CommonsLoggingTest main
WARNING: This is a WARN message.
```

Причина, по которой вы не видите сообщений, созданных методами `debug` и `info`, заключается в том, что в моем файле `C:\Program Files\Java\jre1.5.0_06\lib\logging.properties` я установил уровень регистрации, исправив следующую строку так:

```
java.util.logging.ConsoleHandler.level = WARNING
```

Я решил быстро проверить log4J, используя компонент Eclipse Scrapbook, обсуждаемый в главе 8, "Феномен Eclipse!" (кстати, это весьма полезная возможность).

Сначала я скопировал файлы `log4j-1.2.11.jar` и `log4.properties` в мой путь к классу. Вот и все, что необходимо для переключения регистрации JDK на log4j при использовании общей регистрации. Затем я применил следующее однострочное выражение на странице Scrapbook:

```
org.apache.commons.logging.LogFactory.getLog(CommonsLoggingTest.cla
↳ss)
    .fatal("This is a FATAL message.");
```

Обратите внимание на то, что я все еще использую API общей регистрации, а не вызовы log4j или регистрацию JDK, что позволяет мне отделить мой код от основной среды регистрации. Вывод страницы Scrapbook выглядел примерно так:

```
2006-02-17 10:25:50,654 FATAL [CommonsLoggingTest] - This is a FATAL
↳message.
```

Мой файл log4j.properties (расположенный в пути к классу) выглядел следующим образом (*примечание:* я использую только stdout, а не logfile):

```
log4j.rootLogger=WARN, stdout
# log4j.rootLogger=WARN, stdout, logfile
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
log4j.appender.logfile=org.apache.log4j.FileAppender
log4j.appender.logfile.File=timex.log
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n
```

## Простая регистрация в классе TimesheetListController

Давайте опробуем регистрацию на примере приложения Time Expression. Теперь немного изменим код метода TimesheetListControllerTest.handleRequest следующим образом:

```
List timesheets = timesheetManager.getTimesheets(employeeId);
Log log = LogFactory.getLog(TimesheetListController.class);
log.info("Returning " + timesheets.size() + " rows.");
return new ModelAndView(VIEW_NAME, MAP_KEY, timesheets);
```

После этого настроим упомянутый ранее файл log4j.properties следующим образом:

```
log4j.rootLogger=INFO, stdout
```

Результат вызова метода log.info должен выглядеть примерно так:

```
2006-02-17 10:52:15,648 INFO [com.visualpatterns.timex.contr
↳oller.TimesheetListController] - Returning 4 rows.
```

## Замечание о форматах

Обе системы регистрации, JDK и log4j, предоставляют средства форматирования выводимых сообщений в несколько форматов, от простого текста и XML (например, java.util.logging.XMLFormatter) до специализированной компоновки на базе шаблонов (например, org.apache.log4j.PatternLayout).

Мы можем также создавать специальные средства форматирования. Эту ценнейшую возможность исследуем далее, поскольку она позволяет задать вывод по своему устранению и настроить новый способ форматирования в соответствующих файлах конфигурации, т.е. без изменения кода.

## Использование регистрации в Spring и Hibernate

Если вы столкнетесь с проблемами в Hibernate или Spring, то можете поднять уровень регистрации (т.е. до DEBUG для log4j и до FINEST для регистрации JDK). Это, вероятно, приведет к увеличению количества сообщений, однако окажется весьма полезным, особенно в Hibernate, поскольку продемонстрирует специфический для базы данных код SQL (скрытый за сценой).

# Отладка приложений Java с использованием Eclipse

Отладчики, обладающие GUI, – это превосходные инструменты для обнаружения и устранения ошибок, поскольку, кроме всего прочего, они позволяют нам просматривать код и исследовать переменные. Еще одна возможность отладчиков, которую зачастую упускают, – это исследование организации кода при его просмотре, что может помочь вам улучшить стабильность кода, а также найти вероятные причины проблем.

Инструменты Eclipse Java Development Tools (JDT) поставляются укомплектованными отладчиком, обеспечивающим надежную отладку кода Java (он на самом деле внушителен).

Интегрированная среда разработки (IDE) Java действительно прошла долгий путь совершенствования: ее система отладки стабильна, существует возможность удаленной отладки. Здесь есть все средства, типичные для отладчиков GUI, а также некоторые новые, такие как Eclipse Hotswap, позволяющие исправлять код на лету (объясняется позже).

Свою первую статью о Café от Symantec (предшественнике Visual Café) я написал в августе 1996 года (для журнала Dr. Dobb's Journal). Тогда я проработал с Visual Café и JBuilder от Borland не долгое время, но их IDE и отладчики показались мне не очень удобными и не достаточно стабильными. (Что озадачивает меня больше всего, так это то, что я использовал отладчики, которые позволяли просматривать код в стиле почти 15–20-летней давности на таких языках, как C/C++, и мне не понятно, почему для отладчиков Java потребовался столь длинный срок, чтобы заняться этим вопросом.)

Отладчик JDT Eclipse весьма многообещающ, учитывая его набор возможностей, надежность и стабильность. Итак, рассмотрим некоторые из его возможностей. Если вы не читали главу 8, "Феномен Eclipse!", рекомендую вам сделать это до чтения данного раздела, поскольку я буду подразумевать, что вы имеете некоторое представление о платформе Eclipse и JDT. Сначала давайте рассмотрим, как мы начинаем процесс отладки в Eclipse. Eclipse позволяет нам отлаживать различные типы программ Java. Рис. 9.1 демонстрирует некоторые из доступных нам параметров. Теперь рассмотрим различные концепции и средства отладчика JDT.

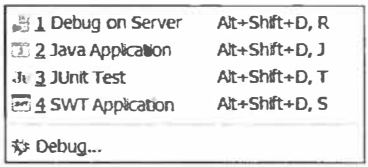



**Рис. 9.1.** Контекстное меню отладки Eclipse

## Концепции и возможности отладки JDT

Отладчик JDT прекрасно позволяет обнаруживать, диагностировать и устранять ошибки. Мы можем отлаживать программы, выполняющиеся локально или дистанционно (если дистанционная отладка поддерживается дистанционной JVM). Клиент отладки запускается внутри Eclipse, в то время как дистанционный сервер отладчика

(с учетом, что вы осуществляете дистанционную отладку) выполняется внутри той же JVM, на которой выполняется ваша серверная программа. Здесь я продемонстрирую локальную отладку, однако скажу несколько слов и о дистанционной отладке, которая подобна локальной отладке, после того как вы настроите дистанционный сервер (это относительно просто). Но сначала сделаем обзор некоторых из фундаментальных концепций отладчика JDT.

Ниже приведен перечень возможностей, предоставляемых средствами отладки JDT. Большинство этих возможностей доступно в контекстном меню (для редактируемого исходного кода), в меню Eclipse Run, в представлениях Debug и даже с использованием комбинаций клавиш. Поэтому я буду рассматривать только сами возможности, не уточняя, где именно в Eclipse к ним можно обратиться, поскольку для этого существует несколько способов.

### Конфигурация и представления отладки

Для начала Eclipse предоставляет конфигурацию Debug (Отладка), которая очень удобно организует набор представлений. Кроме того, когда мы начинаем отлаживать программу, Eclipse автоматически включает эту конфигурацию. С ней связаны такие представления, как Debug (Отладка), Expressions (Выражения), Variables (Переменные), Breakpoints (Контрольные точки), Console (Консоль) и некоторые другие.

### Контрольные точки

*Контрольные точки* (breakpoint) – это локации в вашей программе, где мы хотим приостановить выполнение программы, чтобы исследовать состояние переменных или изменить код. Контрольные точки могут быть установлены на определенных строках кода (на Windows XP я предпочитаю использовать для этого комбинацию клавиш <Ctrl+Shift+B>, но вы можете дважды щелкнуть на строке мышью или щелкнуть правой кнопкой мыши слева от нее в окне редактора). Это и все, что необходимо для установки контрольной точки. Но иногда нужно сделать так, чтобы контрольная точка срабатывала только при некоторых условиях, поэтому давайте рассмотрим и это.

После того как контрольная точка установлена, мы можем задать *количество срабатываний* (hit count), чтобы программа приостанавливалась на этой контрольной точке только после того, как эта строка кода выполнится некоторое количество раз. (Для этого используется контекстное меню контрольной точки.) Мы можем также задать условия срабатывания контрольной точки (используя свойства контрольной точки). Например, мы могли бы захотеть, чтобы программа остановилась на контрольной точке только тогда, когда переменная имеет некоторое значение (как показано на рис. 9.2). Полагаю, я должен упомянуть, что лично я почти никогда не использую количество срабатываний и условия срабатывания контрольных точек.

И последний тип контрольных точек, которые вы можете устанавливать, – это контрольные точки для перехвата исключения, как показано на рис. 9.3. Это выполняется с использованием небольшой кнопки "J!" на панели инструментов в представлении Breakpoints.

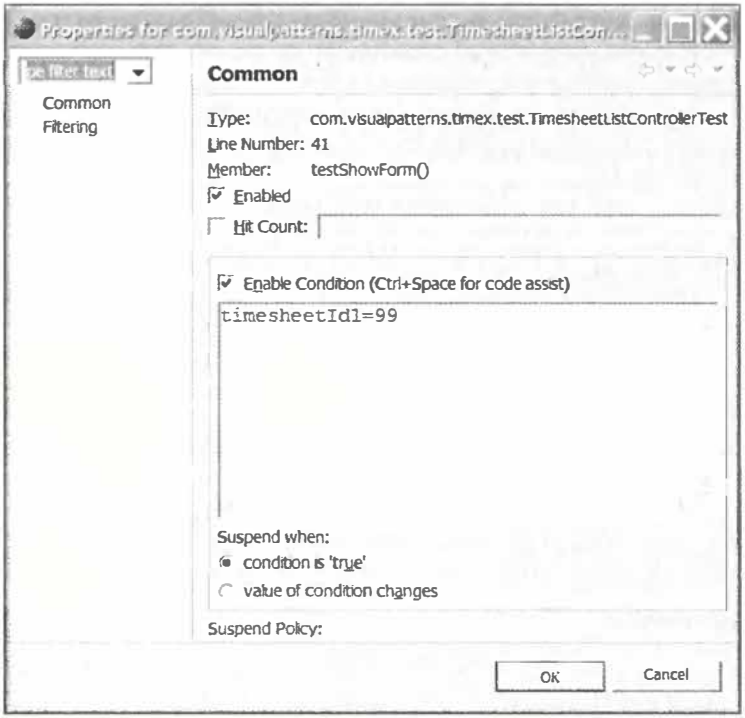



**Рис. 9.2.** Установка условий для контрольных точек


**Рис. 9.3.** Контрольные точки для исключений Java

### Перемещение по коду

После установки нашего набора контрольных точек и запуска программы (например, с использованием пункта Debug меню Run) мы можем начать отлаживать наш код, перемещаясь по нему. Когда встречается контрольная точка, т.е. программа приостанавливается на некоторой строке кода, мы можем либо войти внутрь вызываемого метода, либо переступить через него, либо возвратиться из текущего метода обратно в вызывающий метод (т.е. на один уровень в стеке вызовов). Мы можем даже использовать фильтры шагов (в меню Windows (Окна) пункты Preferences (Предпочтения), Java, Debug (Отладка)), чтобы входить в методы выборочно.

Вместо того чтобы проходить каждую отдельную строку кода, мы можем установить несколько контрольных точек и продолжать выполнение программы, позволяя ей работать до следующей контрольной точки (пропустив все шаги между двумя контрольными точками).

### Переменные

Проверка переменных — это один из наиболее важных аспектов отладки. После того как ваша программа встречает контрольную точку и приостанавливается, вы можете просматривать значения всех переменных, которые находятся в области видимости (например, локальные переменные отлаживаемого метода или класса). Вы можете также отслеживать выражения; например, с использованием представления Watch (Отслеживание), в качестве выражения может быть добавлено нечто вроде `"periodEndingDate != null"`; это выражение возвратит значение `true`, если значение переменной `periodEndingDate` не равно нулю.

Очень мощной возможностью для работы с переменными является также способность изменять их значения. Изменение значения конкретной переменной в определенной строке кода позволяет на лету изменять весь ход сеанса отладки. Эта возможность дополняет возможность Hotswap (обсуждается вскоре), позволяющую изменять на лету код и продолжать отладку.

И последняя, заслуживающая внимания возможность — это способность отображать значение определенной переменной, выделив ее в исходном коде и выбрав в контекстном меню (щелчок правой кнопкой) пункт Display (Отобразить).

### Hotswap — исправление кода на лету!

Это настолько замечательная и мощная возможность, что просто невероятно.

Предположим, мы остановились на строке кода, отмеченной контрольной точкой, и, сразу обнаружив причину ошибки, хотим устранить ее. Собственно, это именно то, что мы и должны сделать! Непосредственно после того, как мы сохраняем файл (нажав комбинацию клавиш <Ctrl+S>, например), Eclipse сразу перекомпилирует ваш класс и перезагрузит метод в том же сеансе отладки без необходимости его перезапуска! Ну не замечательно ли это?

Но что смущает меня в этой возможности, так это ее практическая мгновенность, отладчик автоматически позиционирует вашу текущую строку на начало метода, а следовательно, по существу, происходит перезапуск сеанса отладки, но лишь от начала метода, не для всего стека вызовов.

Обратите внимание на то, что для возможности Hotswap необходим JRE версии 1.4 или позднее.

### Дистанционная отладка

Eclipse позволяет отлаживать дистанционные программы, если дистанционный сервер поддерживает такую возможность. Например, я запустил сервер Tomcat вне Eclipse, на Windows XP, при помощи файла `catalina.bat`, содержащего следующие команды (обратите внимание на две переменные среды окружения, которые должны быть установлены заранее):

```
set JPDA_ADDRESS=8000
set JPDA_TRANSPORT=dt_socket
catalina jpda start
```

Кстати, JPDA (сокращение от Java Platform Debugger Architecture) — это *архитектура отладчика платформы Java* (архитектура для дистанционной отладки; `java.sun.com`). После запуска сервера Tomcat, я смог подключиться к нему, использовав пункты Debug (Отладка), Remote Java Application (Дистанционное приложение Java) меню Run (Пуск) или пункт New (Новый) контекстного меню.

### Другое

В представлениях, связанных с отладкой, для просмотра доступно множество разных параметров. На панелях инструментов этих представлений вы найдете несколько параметров, которые могли бы приятно удивить вас. Одна из возможностей — это *стек копий* (copy stack), доступный в контекстном меню представления Debug; это может оказаться полезным при разных обстоятельствах, например при анализе стека электронной почты. Данная возможность, по существу, копирует стек вызовов в буфер обмена как открытый текст, см. пример фрагмента ниже.

```
TimesheetListControllerTest.testShowForm() line: 45
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line:
↳not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: not available
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: not
↳available
Method.invoke(Object, Object...) line: not available
TimesheetListControllerTest(TestCase).runTest() line: 154
TimesheetListControllerTest(TestCase).runBare() line: 127
TestResult$1.protect() line: 106
TestResult.runProtected(Test, Protectable) line: 124
TestResult.run(TestCase) line: 109
TimesheetListControllerTest(TestCase).run(TestResult) line: 118
TestSuite.runTest(Test, TestResult) line: 208
TestSuite.run(TestResult) line: 203
RemoteTestRunner.runTests(String[], String) line: 478
RemoteTestRunner.run() line: 344
    RemoteTestRunner.main(String[]) line: 196
```

## Отладка пользовательских Web-интерфейсов с использованием Firefox

На рынке имеется множество инструментальных средств отладки приложений Java. Однако отладка клиентских приложений JavaScript не так проста. Браузер Firefox от Mozilla несколько облегчает эту часть. Например, я использовал два описанных ниже высокотехнологичных расширения (доступны на `addons.mozilla.org`).

## Отладчик JavaScript

Это расширение (кодовое название Venkman) чрезвычайно полезно для отладки кода JavaScript (в Web-ориентированных пользовательских интерфейсах). Оно предоставляет большинство средств, которые вы ожидали бы от отладчика GUI, например навигация по коду, проверка значений переменных и т.д. На рис. 9.4 представлен снимок экрана этого отладчика.

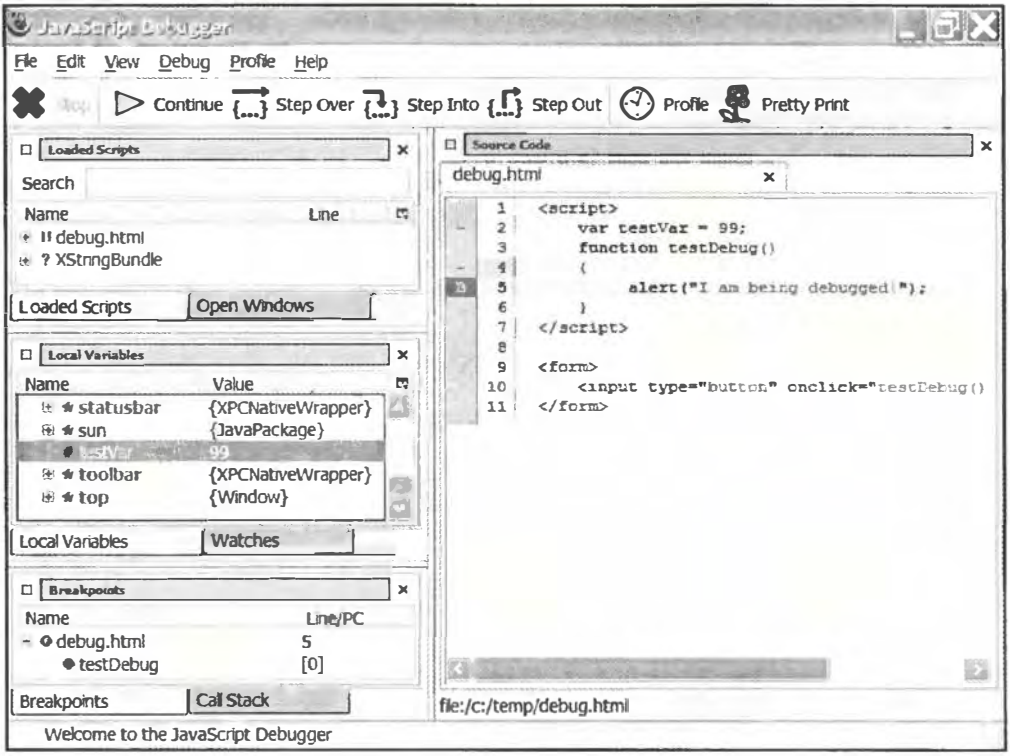



**Рис. 9.4.** Расширение отладчика JavaScript от Firefox

## Расширение Web Developer

Расширение Web Developer от Firefox загружается в комплекте с 50 или более утилитами. Это, вероятно, одно из наиболее хорошо проработанных расширений. Ничего удивительного, что оно удостоилось оценки в пять звездочек из пяти возможных на сайте расширений Firefox. Я, главным образом, использую меню Form (Форма) и Resize (Изменение размеров) только потому, что другие мне пока не пригодились, а к ним относятся средства для работы с CSS, проверки достоверности ссылок, просмотра комментариев, отображения информации о каждом отдельном элементе на странице и много другого!

## Другие расширения от Firefox

Хотя я продемонстрировал здесь лишь пару расширений от Firefox, их существует значительно больше. Например, на момент написания этой книги на Web-сайте Firefox имелось больше 1 000 расширений. Рис. 9.5 демонстрирует еще один пример такого расширения, по имени Tamper Data. Это расширение позволяет изменять (вмешаться в (tamper)) заголовок и запросы POST конкретной страницы. Это может прекрасно подойти для проверки защиты, например. Чтобы ознакомиться с другими расширениями, посетите Web-сайт `addons.mozilla.org`.

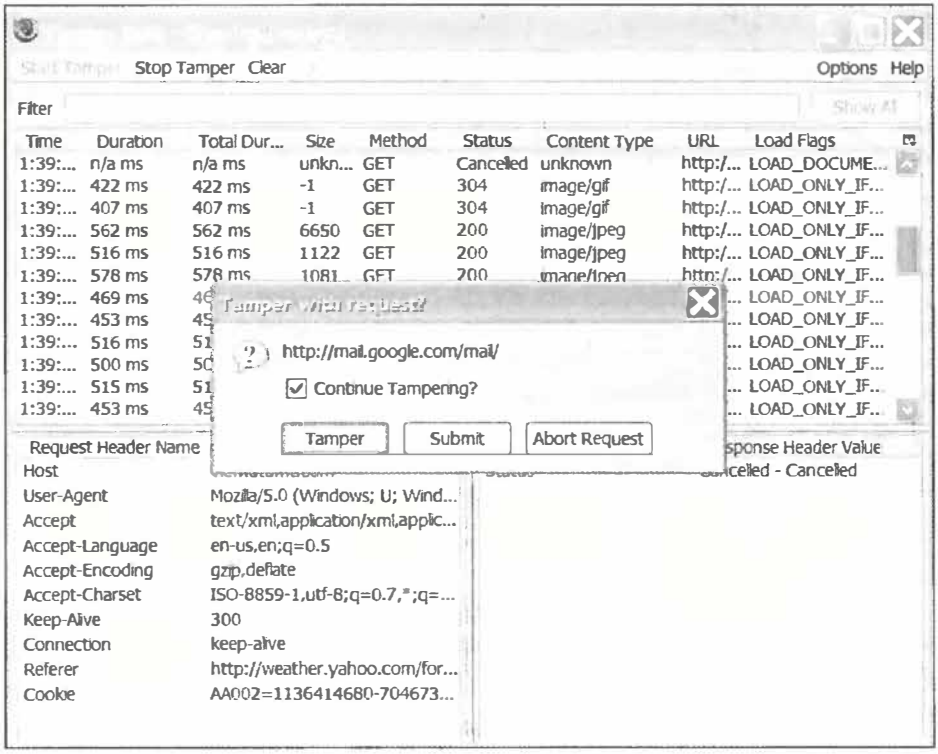



**Рис. 9.5.** Расширение Tamper Data от Firefox

## Консоль JavaScript

Еще одно, заслуживающее внимания расширение Firefox – это консоль JavaScript (доступная в меню Tools), При помощи которой я обнаружил несколько ошибок в пользовательских интерфейсах. Она позволяет даже вычислять результат однострочных выражений JavaScript. Например, я использую меню Resize для изменения размера моего экрана на 800×600, чтобы удостовериться в пригодности моего UI для использования в этом разрешении. Меню Form отображает информацию о полях формы и многом другом.

# Сквозная отладка класса TimesheetManagerTest (от браузера до базы данных)

Как я упоминал ранее, отладка время от времени может проходить болезненно. Было бы очень хорошо, если бы мы могли просто сообщать компьютеру причину проблемы и делать так, чтобы он устранял ошибку сам. Поскольку сейчас это невозможно, нам приходится изыскивать способы облегчения этого процесса, и Eclipse хорошо справляется с этой задачей.

Способность полностью отладить мой серверный код (например, Web и базы данных) – вероятно, моя любимая возможность, связанная с отладкой в Eclipse. Как мы обсуждали в главе 8, "Феномен Eclipse!", дополнений для Eclipse имеется в изобилии. Одно из них – дополнение Data; является частью инструментальной платформы Web Tools Platform (WTP) проекта Eclipse. В комбинации со встроенными средствами серверной отладки получается очень мощная концепция! Давайте рассмотрим это на примере.

На рис. 9.6 представлен снимок экрана Eclipse. Здесь я просматриваю код файла `TimesheetManagerTest.java`, проверяя различные переменные, просматривая вывод в представлении Console, просматривая результаты проверки в представлении JUnit и, вы готовы?, отслеживая изменение данных в базе (вручную обновив их) прямо перед моими глазами (по мере просмотра метода `timesheetManager. deleteTimesheet`) при помощи представления Data Output (Вывод данных). Это просто замечательно!

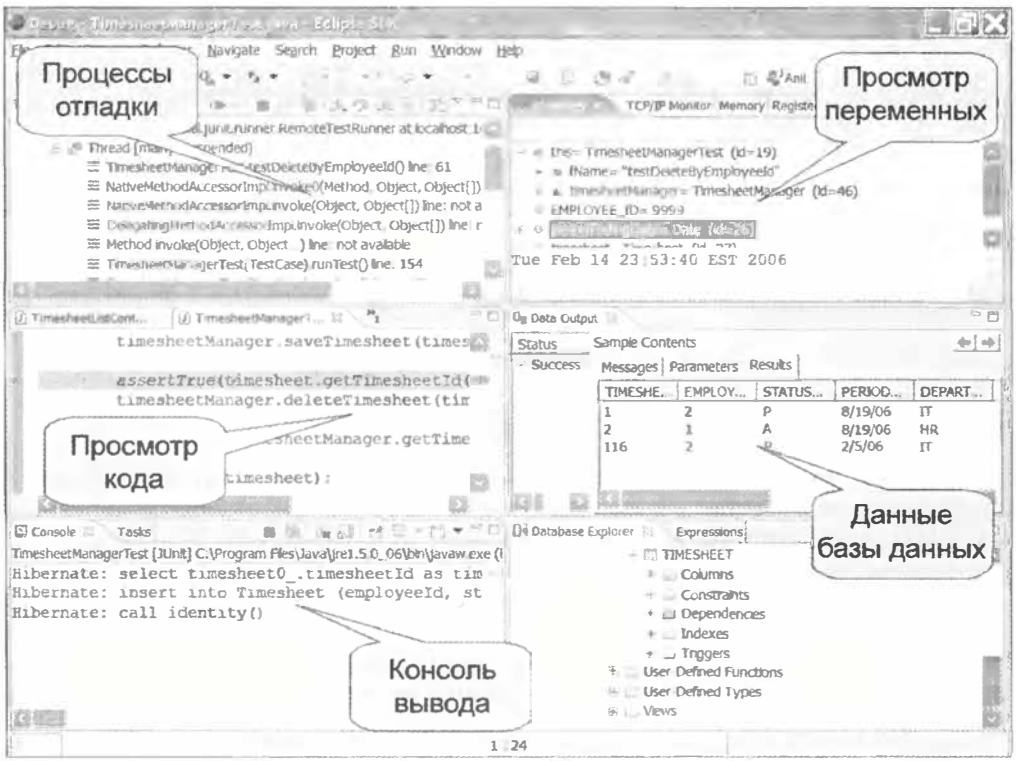



**Рис. 9.6.** Консолидированная отладка приложения Time Expression в Eclipse

На клиентской стороне я предпочитаю использовать некоторые из расширений, которые упомянул ранее, например отладчик JavaScript, консоль JavaScript и Web Developer. Например, на рис. 9.7 приведен снимок экрана нашей формы Enter Hours с именами полей, их размерами и другой информацией. Это расширение помогло мне найти ошибки усечения полей, т.е. мои поля ввода текста оказались слишком маленькими.

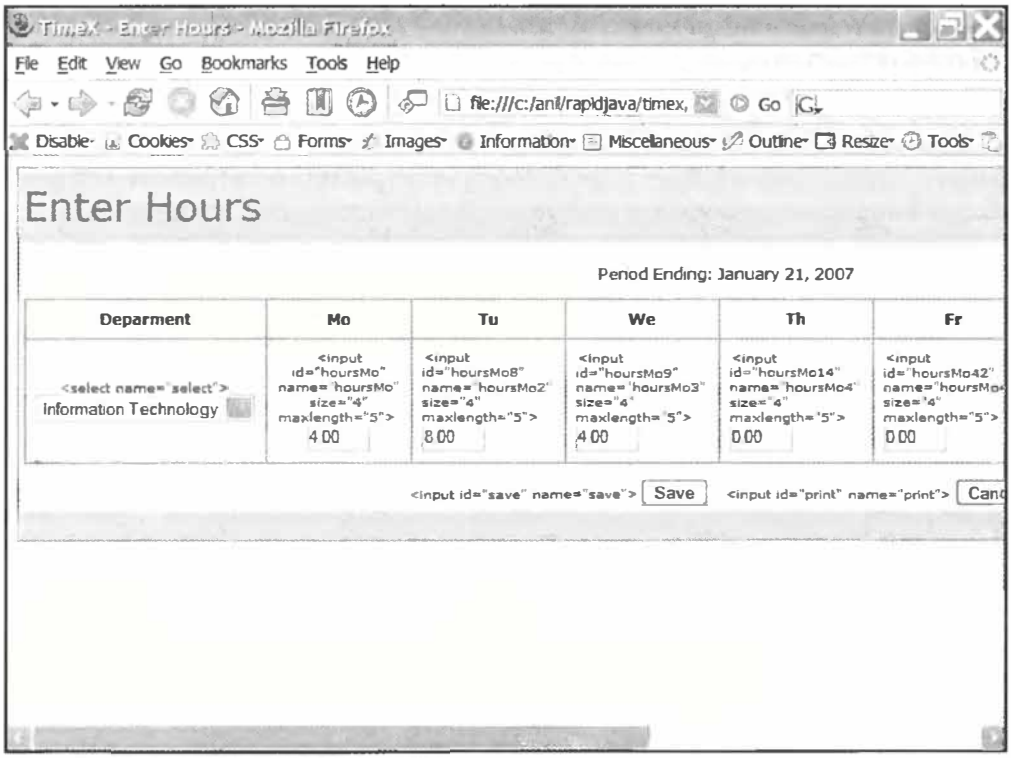



**Рис. 9.7.** Расширение Web Developer от Firefox

# Управление и мониторинг JMX

Платформа Java Platform Standard Edition (JSE) 5.0 обеспечивает дистанционный мониторинг и управление, а также консоль для контроля приложения, выполняющегося с использованием JSE версии 5.0 или более поздней версии. Эти инструменты применяются для просмотра ресурсов, используемых приложениями Java. Например, это может помочь в обнаружении проблем памяти, загрузки класса и сбора мусора, контроля уровня регистрации JDK и управления компонентами приложения Managed Beans (MBeans). Я решил контролировать сервер Tomcat с развернутым на нем приложением Time Expression. Сначала я установил переменную среды окружения `CATALINA_OPTS` следующим образом:

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote
```

После установки этой переменной среды окружения, я запустил сервер Tomcat из командной строки. Однако можно было бы запустить сервер Tomcat и изнутри Eclipse. После запуска сервера Tomcat я запустил предоставляемую JDK утилиту JConsole следующим образом:

```
c:\program files\java\jdk1.5.0_06\bin\jconsole
```

Рис. 9.8 и 9.9 демонстрируют мониторинг локального экземпляра сервера Tomcat при помощи консоли JConsole. Более подробная информация об управлении и мониторинге на базе JMX содержится на Web-сайте `java.sun.com`.

В следующей главе мы разработаем наш собственный компонент JMX (с помощью среды Spring Framework) и будем контролировать его, используя приложение JConsole.

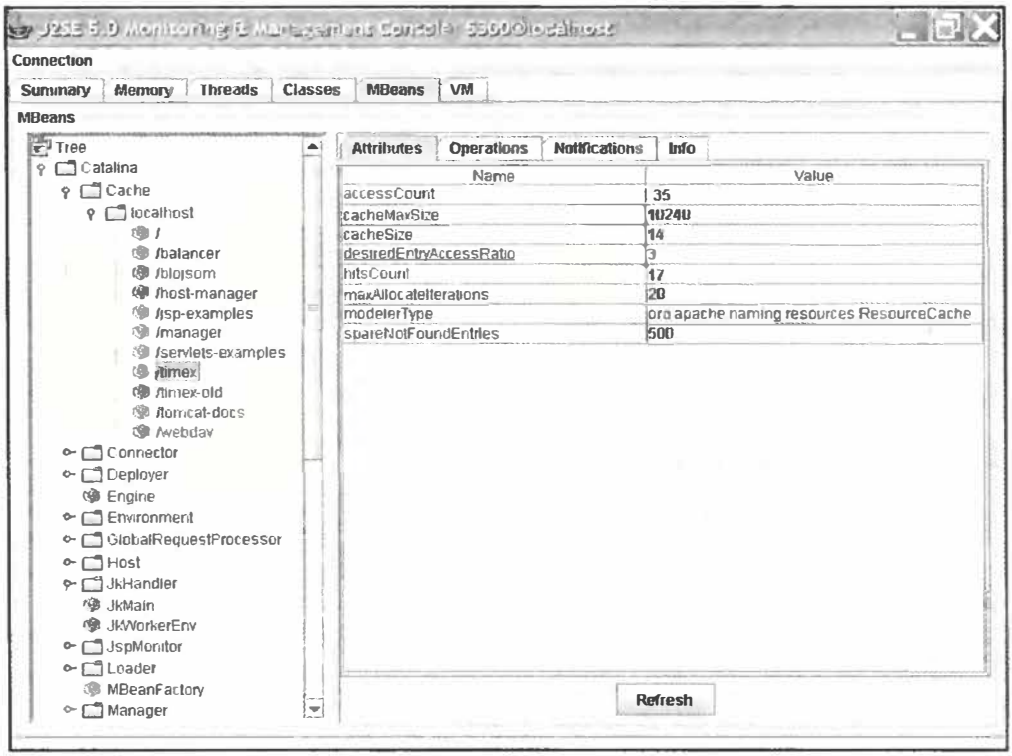



**Рис. 9.8.** Приложение JConsole JSE 5.0 (мониторинг локального сервера Tomcat от Apache и файла `timex.war`)

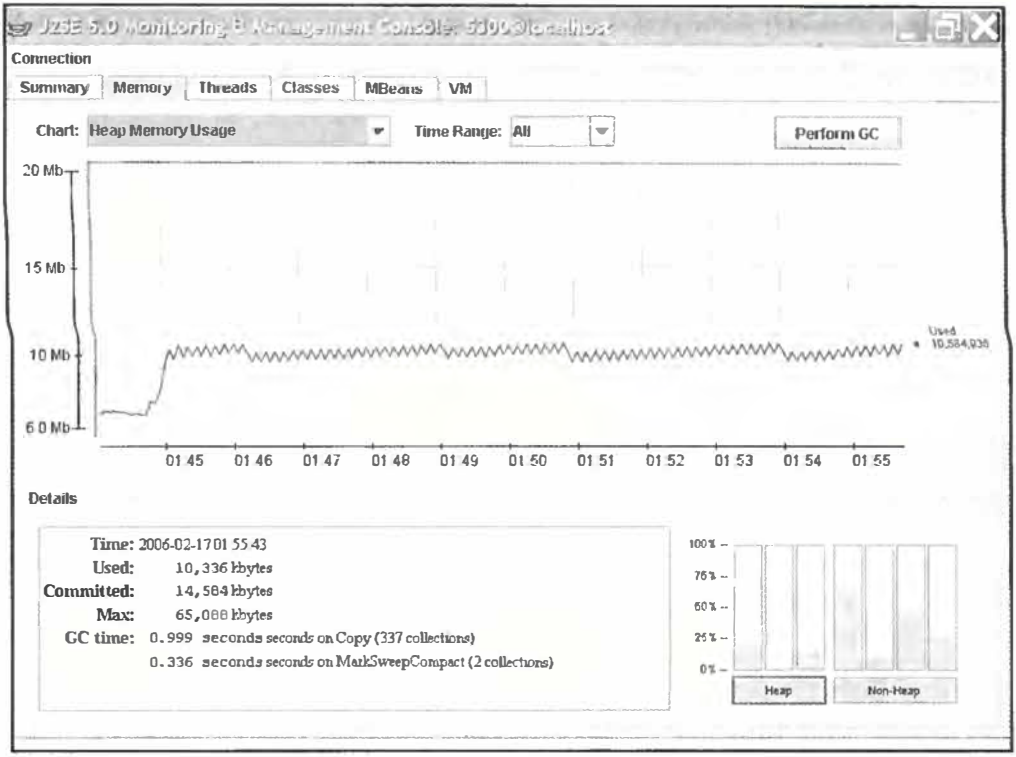



**Рис. 9.9.** Приложение JConsole JSE 5.0 (мониторинг локального сервера Tomcat от Apache)

# Профайлеры Java

*Профайлеры* (profiler) Java существуют практически столько же, сколько и Java. Кроме всего другого, профайлеры позволяют анализировать использование распределяемой памяти, ее утечку, использование процессора, трассировку объектов и методов, выявлять узкие места производительности и многое другое. Ныне доступны разнообразные профайлеры, как с открытым исходным кодом, так и коммерческие (например, профайлер YourKit на `yourkit.com` и комплект Quest JProbe Suite). Одни из них выполняются как автономные программы Java, другие могут быть развернуты в контейнер сервлета или как дополнения Eclipse.

Мониторинг JMX, который я упоминал ранее, довольно надежен; однако он требует JSE версии 5.0 и не способен обеспечить представления, специфического для приложения, а также необходимого профилирования. Так, если вам необходим профайлер с открытым исходным кодом, посетите Web-сайт `http://www.manageability.org/blog/stuff/open-source-profilers-for-java/view/`, содержащий перечень профайлеров Java с открытым исходным кодом.

# Советы по отладке

Ниже приведены советы по отладке. Некоторые из них более очевидны, другие менее, но я надеюсь, что вы найдете среди них полезные для себя.

- Если вы никак не можете найти причину ошибки, отложите решение проблемы на позже, даже на следующее утро.
- Всегда пытайтесь воспроизвести проблему. Например, я зачастую прошу своих клиентов предоставить мне снимок данных, которые они использовали при проверке системы, чтобы я мог воссоздавать это в среде разработки. Этот снимок (выборка) может быть легко получен с использованием инструментальных средств базы данных, которые позволяют вам импортировать и экспортировать данные (одним из таких инструментов является Aqua Data Studio; `aquafold.com`). Просите также, чтобы клиент воссоздал проблему самостоятельно, т.е. повторить действие, при котором произошла проблема, и состояние, в котором находилось приложение.
- Привлеките вторую пару глаз для рассмотрения вашей проблемы. Нередки случаи, когда вашу неразрешимую проблему некто посторонний решает за пару секунд, поскольку он обладает свежим взглядом на нее, в то время как вы, устав просто, не видите вполне очевидного решения (об этом еще говорят "не видеть леса за деревьями"). Если вы используете экстремальное программирование, то это одна из дополнительных льгот, которые вам предоставляет групповое программирование (детали на `extremeprogramming.org`).
- Используйте методику отсечения, т.е. исключайте из проверки код, который, по вашему мнению, работает правильно, и анализируйте тот, который сомнителен. Если возможно, физически отделите код, который работает, от кода, который содержит проблему. Отладка небольших фрагментов кода проходит значительно проще, чем у сложных систем.
- Выясните, не является ли причиной проблем то, от чего ваш код зависит. Например, не изменились ли внешние API или интерфейс? Нет ли ошибок в данных? Корректна ли версия сервера приложений и базы данных? Не кроется ли проблема в аппаратных средствах?
- Программируйте внимательно, проверяя допустимость вводимых значений. Хорошие примеры реализации этого утверждения представлены в J2SE 1.4 (см. детали по адресу `java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html`). Еще одна возможность – это проверка допустимости значений вручную или передача для недопустимых значений исключения `java.lang.IllegalArgumentException`.
- Для вывода стека без фактической передачи исключения, используйте в вашем коде `new Exception().printStackTrace()`. Это может быть полезно, если вы хотите знать, где именно нечто происходит.
- Пытайтесь всегда создавать в своих классах простые методы `toString()`. Это может оказать существенную помощь при отображении объектов.

**Личное мнение: методы предотвращения ошибок**

Ошибки — это обычная часть нашей жизни, как разработчиков программного обеспечения. Однако у меня есть несколько рекомендаций, которые помогут вам снизить вероятность их возникновения.

Для начала, не стоит распечатывать *собственный* код и просматривать его на бумаге без компьютера. Возможно, это один из самых эффективных подходов, но требует немалого терпения. Некоторые организации идут на шаг дальше, делая разбор программы. Экстремальное программирование рассматривает это как групповое программирование.

Еще один хороший способ просмотра вашего кода — это использовать отладчик GUI. Вы можете использовать отладчик не только для отладки, но и для проверки работы вашего кода.

Однако, на мой взгляд, один из наилучших способов снижения вероятности ошибок — это разработка программного обеспечения небольшими фрагментами. Другими словами, для каждой двухнедельной итерации должны предъявляться требования, как минимум, в форме низкоуровневых и высокоуровневых требований. Для высокоуровневых требований могли подойти исходные эскизы UI (для приложений пользовательского интерфейса), модель домена, пользовательские истории и т.д. Как правило, они создаются в начале выпуска. В начале каждой итерации вы получаете подробные требования в форме приемочных испытаний. С учетом этого подхода, будет существенно меньше того, что может пойти не так. Кроме того, когда вы применяете предварительную проверку (вспомните вставку "Создание проверки модуля и собственно кода одновременно" из главы 7, "Среда Spring Web MVC Framework") совместно с рефакторингом тогда, когда это необходимо, ваш код, в конце концов, получится более надежным.

Итерационная разработка, подразумевающая использование коротких и фиксированных циклов (например, по две недели) наравне с активным участием всех заинтересованных лиц, способна снизить давление на вас при срочной разработке программного обеспечения, поскольку клиенты постоянно видят прогресс в выполнении проекта и, вероятно, будут гибче, когда вы не уложитесь в сроки. Отсутствие давления — это тоже предпосылка к созданию более стабильного кода, поскольку вы, так сказать, не прижаты к стене.

Поиск ошибок — это не забава. Иногда мне это кажется похожим на поиск реального жука[1] в большой, загроможденной комнате.

Осуществляя реальную итерационную разработку (от требований до развертывания) при небольших итерациях (в идеале по две недели), вы, вероятно, сумеете предотвратить ошибки и полюбите программирование еще больше!

# Резюме

В этой главе мы рассмотрели следующее.

- Обзор концепций регистрации
- Регистрация при помощи пакета Jakarta Commons Logging (JCL)
- Отладка приложений Java с использованием Eclipse
- Отладка пользовательских Web-интерфейсов с использованием Firefox
- Управление и мониторинг JMX
- Профайлеры Java
- Советы и приемы

В этой книге мы уже сделали немало. Однако осталось еще много дополнительных возможностей, которые предстоит рассмотреть в следующей главе. Итак, давайте двигаться дальше.

---

[1] Игра слов, bug — и компьютерная ошибка, и подслушивающее устройство, и жук. — *Примеч. ред.*

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, содержится на следующих Web-сайтах:

- Профайлеры Java с открытым исходным кодом `http://www.manageability.org/blog/stuff/open-source-profilers-for-java/view/`
- SDK Eclipse `http://www.eclipse.org/`
- Экстремальное программирование `http://extremeprogramming.org/`
- Расширения Firefox `https://addons.mozilla.org/`
- Проект Jakarta Commons Logging (JCL) `http://jakarta.apache.org/commons/logging/`
- Регистрация JDK `http://java.sun.com`
- Мониторинг и управление JMX `http://java.sun.com`
- Log4J `http://logging.apache.org/`
- Simple Logging Facade for Java (SLF4J) `http://slf4j.org/`
- Отладка, проверка и верификация программного обеспечения (статья) `http://www.research.ibm.com/journal/sj/411/hailpern.html`
- TPTP Eclipse `http://www.eclipse.org/tptp/home/downloads/quicktour/v41/quick_tour.html`
- Профайлер Java YourKit `http://www.yourkit.com/`

# 10

# Кроме основ

Выпуск 1, Неделя 8, Итерация 4

**Радж:** Минди, этот центр данных внушителен. Просто удивительно, что наше со Стивом приложение будет выполняться на одном из этих серверов.

**Минди:** Так и есть. Я слышала, что ваше приложение создано с учетом отказоустойчивой архитектуры, обладает защитой, надежно и т.д. Теперь, если обслуживающий персонал обеспечит 99-процентную непрерывность работы аппаратных средств и Сети, Сьюзен и ее группа будут довольны.

Мы рассмотрели довольно много фундаментального материала, а теперь пришло время двинуться дальше основ. Это последняя глава с примерами кода; следующая глава продемонстрирует некоторые идеи о том, что имеет смысл прочитать еще. Поэтому давайте рассмотрим несколько новых передовых средств, чтобы завершить ими основную часть этой книги.

# Что рассматривается в этой главе

В этой главе мы рассмотрим некоторые из средств, которые немного сложнее основ технологий, рассмотренных ранее в этой книге. Я решил описывать эти технологии в том же порядке, в котором я представил их в прежних главах. В этой главе мы рассмотрим следующий материал:

- Новые возможности Java
- Дополнительные встроенные и внешние задачи Ant
- Специальные комплекты JUnit
- Дополнительные возможности Hibernate
- Другие возможности среды Spring Framework
- Интеграция Hibernate и Spring
- Библиотека дескрипторов Displaytag и создание специальных библиотек дескрипторов
- Пример рефакторинга в нашем приложении
- Другие вопросы, такие как управление транзакциями, безопасность, обработка исключений, кластеризация и т.д.
- Простой пример AJaX

**Примечание**
Полный код примеров, используемых в этой главе, находится внутри файла zip, доступного на Web-сайте книги.

Обратите внимание на то, что в этой главе приведены очень краткие описания рассматриваемых средств. Я решил, что лучше привести примеры кода, а не детальные объяснения. Кроме того, большинство упомянутых здесь возможностей подробно описано в соответствующей справочной документации, ссылки на которую я привожу.

# Новые возможности Java

Начнем с комплекта разработчика Java Development Kit (JDK) платформы Java Standard Edition (JSE), поскольку именно с этого мы начали нашу практическую работу в главе 4, "Установка среды: JDK, Ant и JUnit". Комплект JDK 1.5 (известный также как JSE 5.0) предоставил несколько новых средств расширения языка Java. Давайте рассмотрим некоторые из этих средств; но если вы уже знакомы с ними, то этот раздел можете пропустить.

**Совместимость с прежней версией JDK 1.4 для приложения Time Expression**
Я преднамеренно не использовал ни одну из этих новых возможностей в нашем типовом приложении Time Expression, поскольку хотел, чтобы оно было совместимо с прежней вер-

сией JDK 1.4, на случай, если в вашей организации не принята версия JDK 1.5. Обратите также внимание на то, что для обеспечения совместимости с JDK 1.4 я использовал в меню Eclipse Window (Окно) пункты Preferences (Предпочтения), Java, Compiler (Компилятор). Это весьма удобная возможность, которую я рекомендую вам исследовать, если вы еще не знакомы с ней.

Полный код обсуждаемых далее возможностей находится в файле `DemoNewJavaFeatures.java`, расположенном в архиве кода этой книги. Здесь я представлю лишь краткие описания, поскольку примеры просты и достаточно хорошо документированы на Web-сайте `java.sun.com`.

## Статический импорт

С появлением JSE 5.0 стало возможным использовать статические члены непосредственно. Например, нечто вроде `Integer.MAX_VALUE` можно использовать следующим образом:

```
import static java.lang.Integer.*;
System.out.println(MAX_VALUE);
```

## Встраивание

В предыдущих выпусках мы могли вставить в класс любой тип объекта из среды Collections, но затем мы должны были преобразовать такие объекты. Эти операции были несколько небезопасны, поскольку компилятор не способен проверить допустимость типа. Теперь мы можем избежать этого, сделав нечто, подобное тому, что приведено ниже. Обратите внимание, как я могу использовать полученный метод, без всякой необходимости в преобразовании:

```
ArrayList<String> arrayList = new ArrayList<String>();
arrayList.add("Testing");
System.out.println(arrayList.get(0));
```

Как можно заметить, мы просто добавляем объекты типа `String` в список `ArrayList`. Например, приведенный ниже код привел бы к ошибке во время компиляции:

```
arrayList.add(new Integer(1));
```

## Расширенный цикл for

Цикл `for` был значительно упрощен в JSE 5.0. Для перебора коллекции класса, вместо старого стиля, `for (int i=0; i < c.size; i++)`, мы могли бы использовать следующий:

```
public static void demoForLoop(Collection<Integer> c)
{
    // новый стиль использования цикла for
    for (Integer i : c)
        System.out.println(i);
}
```

Такая форма не только уменьшает объем кода, но и снижает вероятность возникновения ошибок, вызванных недопустимыми значениями индексных переменных (например, переменной `i` в этом примере).

## Автоупаковка

Это великолепная возможность, на мой взгляд, ее следует поддерживать в Java и далее. Как вы, возможно, знаете, мы не можем помещать данные простых типов (например, `int`) в коллекции классов. Обладая этой новой возможностью, мы можем добавить число в список `ArrayList`, как показано далее, причем без необходимости в конструкции вроде `new Integer(1)`:

```
ArrayList<Integer>list = new ArrayList<Integer>();
list.add(1);
```

## Перечисления

Приведенный далее простой пример не лучшим образом демонстрирует мощь *перечислений* (enum) в Java:

```
enum BookName { RAPID, JAVA, DEVELOPMENT };
for (BookName bookName : BookName.values())
    System.out.println(bookName);
```

Приведенное ниже описание, взятое непосредственно с Web-сайта `java.sun.com`, лучше освещает этот вопрос:

"Перечисления в языке программирования Java гораздо мощнее их аналогов в других языках, где они немногим больше, чем прославленные целые числа. Объявление нового перечисления определяет вполне развитый класс (дублирующий тип перечисления)..., он позволяет добавлять произвольные методы и поля к типу перечисления (enum), а также реализовать произвольные интерфейсы и т.д. Типы перечислений предоставляют высококачественные реализации всех объектных методов. Они поддерживают сравнение и сериализацию, а последовательная форма позволяет противостоять произвольным изменениям в типе перечисления."

## Аргументы переменной длины

*Аргументы переменной длины* (vararg) позволяют передать в методы переменные аргументы. Прежде мы должны были использовать для этого нечто вроде объектного массива. Теперь для этого мы можем использовать многоточие (`...`), как показано далее:

```
public static void demoVarargs(Object... args)
MessageFormat.format(
        "I''m working on {0}"
    + "  on {1}"
    + "  at {2} hours.", args);
```

Код, который вызывает наш метод `demoVarargs`, выглядит следующим образом:

```
demoVarargs("Rapid Java Development", new Date(), 1800 );
```

Кроме того, такие классы JDK, как `MessageFormat`, также способны получать переменные аргументы, как показано в этом примере.

## Другие возможности

Существует множество других средств (например, аннотации), а также расширения API и JVM, которые я не рассматривал здесь, поскольку подробности и руково-

дства по ним могут быть легко найдены на Web-сайте `java.sun.com`. Кроме того, мои простые примеры подтверждают справедливость этого подхода. Как можно заметить на рис. 10.1, Java — это огромная платформа; ничего удивительного, что по ней и сопутствующим темам написаны сотни книг.

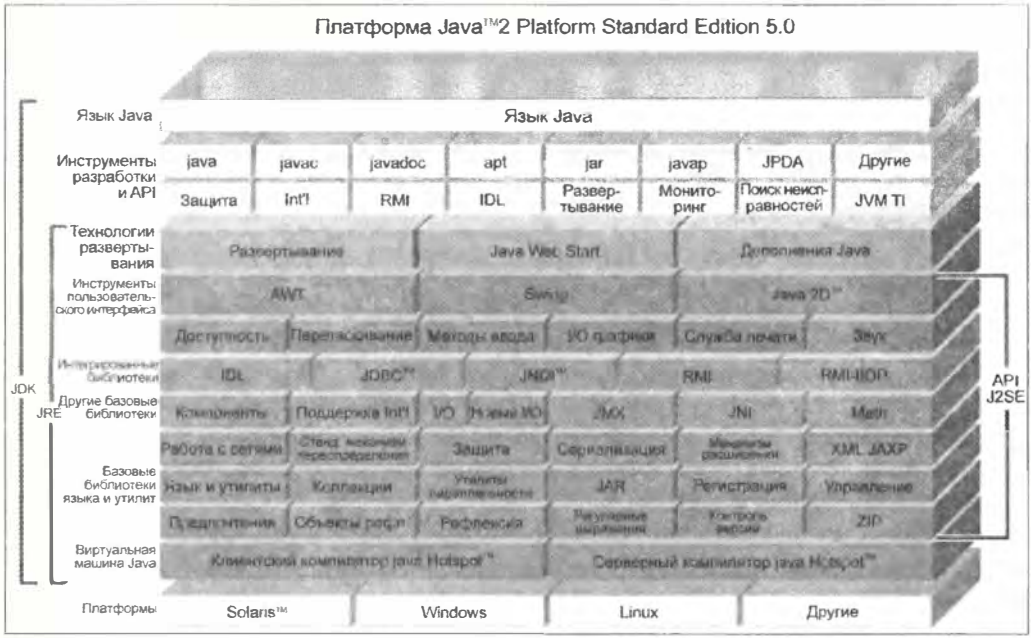



**Рис. 10.1.** Платформа Java Platform Standard Edition 5.0 (источник: `java.sun.com`)

На момент написания этой книги был анонсирован новый релиз J2SE 6, который предоставляет новые, расширенные средства защиты, интегрированные Web-службы, расширенную поддержку JMX, улучшенный GUI и многое другое. Более подробная информация по этой теме приведена на Web-сайте `java.sun.com`.

# Задачи Ant

На протяжении этой книги мы не раз использовали Ant, даже внутри Eclipse. Хотя мы уже рассмотрели некоторые весьма полезные и надежные средства Ant, давайте исследуем еще несколько задач Ant, использованных в нашем файле `antextras.xml`. Некоторые из них — это встроенные задачи, другие считаются внешними задачами, требующими размещения в каталог `<ant-home>/lib` одного или нескольких файлов JAR (библиотек). Под каталогом `ant-home` подразумевали каталог установок Ant; например, `C:\apache-ant-1.6.5`.

Как вы заметите на примере некоторых из этих задач, Ant — это один из красивейших инструментов, с учетом его простоты и огромной мощности. Это намного больше, чем инструмент построения, как я продемонстрирую далее.

Более подробная информация об этих и других задачах, не упомянутых здесь, приведена на Web-сайт `ant.apache.org`. Этот сайт предоставляет также информацию о создании своих собственных специальных задач Ant.

## Задача CVS

Как и подразумевает имя этой задачи, она позволяет нам работать с CVS. Она требует наличия исполняемого файла CVS в вашем пути. Например, я загрузил TortoiseCVS с Web-сайта `tortoisecvs.org` и был способен использовать файл `cvs.exe`, расположенный в каталоге этой программы (на системах Linux, Unix и Mac OS X вы могли бы иметь заранее установленный CVS). Приведенный ниже код XML демонстрирует, как работает эта задача; я смог загрузить в среду выполнения модуль Spring, используя Ant:

```
<cvs cvsRoot=":pserver:anonymous@cvs.sourceforge.net:/cvsroot/spr
↳ingframework"
    package="spring"
    dest="/temp"
/>
```

## Задача Exec

Эта задача позволяет нам запускать внешние программы; например, следующий код выполняет команду `date`:

```
<exec command="date"/>
```

## Задача Get

Задача `Get` позволяет нам выбрать файл, используя метод HTTP `GET`, как продемонстрировано ниже:

```
<get src="http://visualpatterns.com/comics/funny.gif"
    dest="funny.gif "
    verbose="true" />
```

## Задача Sleep

Как можно предположить, эта задача позволяет нам приостанавливать работу на определенный период времени, как, например, здесь, на 2 секунды:

```
<sleep seconds="2"/>
```

## Задача FTP

Это очень мощная задача, поскольку позволяет использовать FTP непосредственно из файла XML Ant. Например, при работе над этой книгой я использовал ее для резервного копирования моих файлов, связанных с книгой и приложением Time Expression, на сервер FTP, при автоматическом использовании компонента Microsoft Windows Scheduled Tasks!

```
<ftp server="mirrors.kernel.org"
    action="get"
    remotedir="/gnu/chess "
```

```
    userid="anonymous"
    password="guest@guest.com"
    verbose="yes "
    binary="yes" >
    <fileset file="README.gnuchess"/>
</ftp>
```

Задача FTP — это внешняя задача, она требует наличия в каталоге <ant-home>/lib следующих файлов:

- Jakarta Commons Net (http://jakarta.apache.org/commons/net/;commons-net<version>.jar)
- Jakarta-ORO (http://jakarta.apache.org/oro/;jakarta-oro<version>.jar)

### Задача Mail

Эта задача позволяет осуществлять передачу по почте с использованием протокола SMTP. Используя элемент <fileset>, мы можем также присоединять к посланиям файлы. Это весьма мощная комбинация, поскольку мы можем использовать ее для передачи сообщений электронной почты сразу после их создания (это особенно полезно в автоматизированной непрерывно работающей системе). Ниже приведен простой пример применения этой задачи:

```
<mail tolist="friend@somehost.com"
    subject="Hello!"
    from="me@myhost.com"
    mailhost="myhost.com"
    user="myuserid"
    password="mypassword"/>
```

Эта возможность требует наличия в каталоге <ant-home>/lib файлов mail.jar (часть API JavaMail) и activation.jar (часть JavaBeans Activation Framework). Оба они находятся на Web-сайте java.sun.com.

### Задач в изобилии!

Я упомянул только некоторые из задач Ant, которые, я полагаю, будут вам полезны. Существуют буквально сотни разнообразных задач Ant, от специализированных до общих, таких как Gzip, Tar и многие другие. Чтобы выяснить подробности об этих и некоторых других задачах, а также о создании собственных специальных задач, посетите Web-сайт ant.apache.org.

## Возможности JUnit

До сих пор в книге использовались одиночные проверочные комплекты, которые применялись для проверки только тех методов, имена которых начинались со слова "test". Мы рассмотрели также запуск всех проверочных классов в пакете из специфического каталога, при помощи Ant. Теперь рассмотрим, как создавать специальные проверочные комплекты, а также многократного использовать *фиксирующий* (fixture) код.

## Специальные проверочные комплекты

Система JUnit позволяет нам создать свои собственные проверочные комплекты. Кроме того, комплекты могут содержать другие комплекты, что делает эту концепцию весьма мощной, несмотря на простоту JUnit. Приведенный ниже отрывок кода из нашего файла `AllTests.java` (в каталоге `timex/`) демонстрирует создание специальных комплектов.

```
TestSuite modelTestSuite = new TestSuite("Model Tests");
modelTestSuite.addTestSuite(TimesheetManagerTest.class);
TestSuite controllerTestSuite = new TestSuite("Controller Tests");
controllerTestSuite.addTestSuite(TimesheetListControllerTest.class);
TestSuite fullSuite = new TestSuite("All Tests");
fullSuite.addTest(modelTestSuite);
fullSuite.addTest(controllerTestSuite);
return fullSuite;
```

## Проверка фиксирующего кода

Когда мы пишем классы проверки модулей, имеет смысл переместить фиксирующий код в родительский класс, который, в свою очередь, может расширять класс `junit.framework.TestCase`. *Фиксирующий код* (fixture) — это набор объектов (например, проверочные данные), который может быть использован при одной или нескольких проверках. Преимущество фиксирующего кода заключается в том, что он помогает избежать наличия избыточного кода. Например, в прошлых проектах я перемещал в родительский класс код подключения к JDBC, код загрузки контекста приложения Spring и различный другой общий код. Более подробная информация о фиксирующем коде приведена по адресу `junit.sourceforge.net/doc/cookbook/cookbook.htm`. Для нашего примера приложения, фиксирующий код был перемещен в конструктор родительского класса по имени `TimexTestCase`, как показано далее:

```
protected TimexTestCase()
{
    FileSystemResource res =
        new FileSystemResource("src/conf/timex2-servlet.xml");
    springFactory = new XmlBeanFactory(res);

    departmentManager = (DepartmentManager)
        springFactory.getBean("departmentManagerProxy");
    employeeManager = (EmployeeManager)
        springFactory.getBean("employeeManagerProxy");
    timesheetManager = (TimesheetManager)
        springFactory.getBean("timesheetManagerProxy");
    applicationSecurityManager = (ApplicationSecurityManager)
        springFactory.getBean("applicationSecurityManager");
}
```

Обратите внимание на то, что я расположил фиксирующий код в конструкторе. Но вы могли бы также расположить его в методе JUnit `setUp`. В качестве альтернативы, если вы используете JDK версии 1.5 или более, можете применять аннотации для инициализации вашего фиксирующего кода (более подробная информация об этой возможности приведена на Web-сайте `junit.org`).

Теперь такие классы, как TimesheetManagerTest, могут расширить наш новый класс TimexTestCase, вместо непосредственного наследования класса TestCase, как показано в этом примере:

```
public class TimesheetManagerTest extends TimexTestCase
```

При наличии общего фиксирующего кода в родительском классе TimexTestCase, все производные от него классы смогут получить к нему непосредственный доступ и без проблем использовать объекты (благодаря внедрению зависимости Spring). Это позволяет в производных классах сосредоточиться на реализации проверки модулей, чтобы передать их на приемочные испытания пользователю, а не тратить время на создание фиксирующего кода для каждого случая проверки. Более подробная информация о проверке модулей, фиксирующем коде, проверочных комплектах и по сопутствующим темам приведена на Web-сайте junit.org.

# Возможности Hibernate

Хотя в главе 5, "Применение Hibernate для постоянных объектов", мы рассмотрели довольно много основ Hibernate, эта технология существенно обширнее, и не удивительно, что ей посвящено столько книг. Однако пару средств, которые вам могли бы пригодиться, имеет смысл обсудить здесь.

## Базовые запросы SQL

Большинство реляционных баз данных совместимо со стандартом ANSI SQL; однако некоторые идут дальше, предоставляя свои собственные специальные расширения, которые могли бы не поддерживаться HQL. Чтобы воспользоваться таким средством, как базовые запросы SQL, мы можем использовать метод Hibernate Session.createSQLQuery, как демонстрирует следующий код, использующий встроенную функцию HSQLDB datediff:

```
String sql = "select datediff('dd', NOW, ?) AS daysleft"
        + " from timesheet";
Integer valueObject = (Integer)session.createSQLQuery(sql)
        .addScalar("daysleft", Hibernate.INTEGER)
        .setDate(0, DateUtil.getCurrentPeriodEndingDate())
        .uniqueResult();
if (valueObject != null)
    daysLeft = valueObject.intValue();
```

## Перехватчики

*Перехватчики* (interceptor), как вы могли бы догадаться, перехватывают запросы. Вероятно, приведенное далее описание, взятое из справочной документации Hibernate, лучше описывает использование перехватчиков Hibernate: "Интерфейс Interceptor поддерживает обратные вызовы для сеанса приложения, позволяя приложению просматривать и манипулировать свойствами постоянного объекта прежде чем он будет сохранен, модифицирован, удален или загружен. Он может использоваться также для отслеживания информации аудита".

Для применения перехватчика Hibernate мы должны либо реализовать все методы, определенные в интерфейсе `org.hibernate.Interceptor`, либо, в качестве альтернативы, создать класс, производный от конкретного класса `org.hibernate.EmptyInterceptor`, и переопределить только те методы, которые необходимы. Это показано в следующем фрагменте кода.

```
public class AuditInterceptor extends EmptyInterceptor
{
    public void afterTransactionCompletion(Transaction tx)
```

После того как класс перехватчика будет создан, мы сможем запустить его либо для данного сеанса, используя метод `Session.openSession(Interceptor interceptor)`, либо на уровне Configuration, как показано далее:

```
SessionFactory = new Configuration()
        .setInterceptor(new AuditInterceptor())
        .configure().buildSessionFactory();
```

# Возможности Spring Framework

Поскольку мы посвятили среде Spring Framework две главы, у вас мог бы возникнуть вопрос, что еще можно сказать о Spring. Ответ прост: хотя такие технологии, как Hibernate, весьма надежны, они сосредоточены в одном направлении (объектно-реляционное связывание, например), однако среда Spring Framework настолько обширна и перегружена различными средствами, что ее трудно рассмотреть даже в небольшой книге. Итак, давайте рассмотрим еще несколько примеров использования среды Spring.

## Планирование работ

Вполне обычным требованием для многих приложений является планирование работ, необходимое, например, для запуска, для пакетной обработки, передачи данных подчиненным приложениям или создания в конце дня отчета о загруженности процессора. Чтобы облегчить планирование работ, большинство компаний используют такие планировщики, как Unix/Linux CRON, Unicenter AutoSys от Computer Associate, Scheduled Tasks от Microsoft и т.д.

Система Quartz от Open Symphony (`opensymphony.com`) позволяет планировать работы в программах Java, а среда Spring Framework обеспечивает поддержку для этого продукта (и таймеров JDK). Функции API Quartz намного надежнее таймеров JDK, поскольку предоставляют мощные средства планирования, такие как обсуждаемые далее выражения типа `CRON`.

Вернувшись к главе 2, "Простое приложение: сетевая система учета рабочего времени", содержащей бизнес-требования нашего типового приложения Time Expression, можно вспомнить, что по пятницам, в 2 часа пополудни, необходимо пересылать по электронной почте напоминание. Наши два файла, `ReminderEmail.java` и `timex-servlet.xml`, совместно обеспечивают эти функциональные возможности. Файл `ReminderEmail.java` обсуждается в следующем разделе. Однако давайте рассмотрим отрывок кода из файла `timex-servlet.xml`, относящийся к планированию задачи.

```
<!-- Spring job scheduling -->
<bean id="reminderEmailJobDetail"
    class=
    "org.springframework.scheduling.quartz.MethodInvokingJob
↳DetailFactoryBean">
    <property name="targetObject" ref="reminderEmail" />
    <property name="targetMethod" value="sendMail" />
</bean>
<bean id="reminderEmailJobTrigger"
    class="org.springframework.scheduling.quartz.CronTriggerBean">
    <property name="jobDetail" ref="reminderEmailJobDetail" />h
    <property name="cronExpression" value="0 0 14 ? * 6" />
</bean>
<bean
    class="org.springframework.scheduling.quartz.SchedulerFa
↳ctoryBean">
    <property name="triggers">
        <list>
            <ref bean="reminderEmailJobTrigger" />
        </list>
    </property>
</bean>
```

Обратите внимание на значение `0 0 14 ? * 6`; это выражение в стиле CRON, которое указывает, что задача должна быть выполнена в 14:00 (или 2 часа пополудни) в пятницу (шестой день недели). Подробная документация об API Quartz приведена на сайте `quartz.sourceforge.net`.

Еще один интересный элемент конфигурации – это способ, с помощью которого мы сообщаем среде Spring, какой метод определенного целевого объекта следует вызвать:

```
<property name="targetObject" ref="reminderEmail" />
<property name="targetMethod" value="sendMail" />
```

Для этой возможности требуется доступ к файлу `quartz.jar` от Open Symphony на Web-сайте `opensymphony.com`. В случае приложения Time Expression, его понадобилось бы установить в каталог `timex/lib`.

Одна из проблем, с которой мы сталкиваемся при планировании выполнения работ на Web-сервере или сервере приложений, заключается в возможной *кластеризации* (clustered) сервера, т.е. наличии нескольких экземпляров сервера. (Кластеризация обсуждается далее.) Как вы могли бы предположить, в кластеризуемой системе задача будет запущена на всех экземплярах (серверах). Чтобы обойти эту проблему, в прошлом я использовал методику базы данных, позволяющую управлять параллельным выполнением задач. Эта методика подразумевает использование таблицы `ObjectLock` со столбцом `isLocked`. Столбец `isLocked` применяется для имитации блокировки задачи; впоследствии его значение может быть проверено, что позволит выяснить, выполняется ли уже задача (т.е. установлена блокировка или нет).

## Поддержка электронной почты

Как я упоминал ранее, среди наших бизнес-требований есть такое, согласно которому тем сотрудникам, которые не передали еженедельный табель учета рабочего времени, следует послать по электронной почте напоминание. Чтобы реализовать эту

возможность, мы используем простые API электронной почты, предоставляемые средой Spring Framework, а не нечто вроде непосредственно JavaMail. Обратите внимание на то, что для поддержки электронной почты среда Spring использует JavaMail.

Эти функциональные возможности обеспечивают два наши файла, `ReminderEmail.java` и `timex-servlet.xml`. Ниже приведен отрывок файла `timex-servlet.xml`:

```
<bean id="mailSender"
    class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="host" value="acme.com" />
    <property name="username" value="myuserid" />
    <property name="password" value="mypassword" />
</bean>

<bean id="reminderEmailMessage"
    class="org.springframework.mail.SimpleMailMessage" >
    <property name="from" value="me@me.com" />
    <property name="subject" value="Reminder: Submit Timesheet" />
    <property name="text"
        value="Please don't forget to submit your timesheet.
↳Thank you!" />
</bean>

<bean id="reminderEmail"
    class="com.visualpatterns.timex.job.ReminderEmail" >
    <property name="employeeManager">
        <ref bean="employeeManager" />
    </property>
    <property name="mailSender">
        <ref bean="mailSender" />
    </property>
    <property name="message">
        <ref bean="reminderEmailMessage" />
    </property>
</bean>
```

Приведенный ниже метод из файла `ReminderEmail.java` содержит фактический код, демонстрирующий, как из базы данных получается список рассылки (с использованием одного из наших классов модели из главы 5, "Применение Hibernate для постоянных объектов") для передачи электронной почты соответствующим сотрудникам:

```
public void sendMail()
{
    List list = employeeManager.getHourlyEmployees();
    if (list == null || list.size()< 1)
        return;
    String emailAddresses[] = new String[list.size()];
    Employee employee;
    for (int i=0; i < list.size(); i++)
    {
        employee = (Employee)list.get(i);
        emailAddresses[i] = employee.getEmail();
    }
    message.setTo(emailAddresses);
    SimpleMailMessage threadSafeMailMessage =
        new SimpleMailMessage(message);
    mailSender.send(threadSafeMailMessage);
}
```

Эта возможность требует наличия файлов `mail.jar` (часть API JavaMail) и `activation.jar` (часть среды JavaBeans Activation Framework) в каталоге `timex/lib`. Оба они находятся на Web-сайте `java.sun.com`.

## Поддержка JMX

Мы рассмотрели технологию *расширений управления Java* (Java Management Extensions – JMX) в предыдущей главе. Хотя эта технология, в основном, используется для управления серверами, сейчас, на мой взгляд, ее можно легко применить для контроля деловых аспектов приложения. Например, мы могли бы контролировать количество пользователей, зарегистрировавшихся в системе на протяжении данного дня, количество счетов, обработанных системой учета, количество обращений обработанных системой страхования и т.д. Это может использоваться не только для контроля состояния приложения, но и для обеспечения определенного уровня комфорта работы.

Поддержка JMX средой Spring позволяет нам автоматически регистрировать простые объекты JavaBean. В данном случае это реализовано при помощи двух файлов: `timex-servlet.xml` и простого класса JavaBean `TimexJmxBean.java`, который отслеживает количество пользователей, зарегистрированных в приложении Time Expression, а также количество выбранных записей табеля учета рабочего времени.

Ниже приведен отрывок кода из файла `timex-servlet.xml`:

```
<bean id="timexJmxBean"
    class="com.visualpatterns.timex.util.TimexJmxBean" />
<bean id="exporter"
    class="org.springframework.jmx.export.MBeanExporter" >
    <property name="registrationBehaviorName"
        value="REGISTRATION_IGNORE_EXISTING" />
    <property name="beans">
        <map>
            <entry key="Time Expression:name=timex-stats "
            value-ref="timexJmxBean" />
        </map>
    </property>
</bean>
```

Следующий отрывок кода демонстрирует выдержку из файла `TimexJmxBean.java`:

```
public class TimexJmxBean
{
    private static int signInCount;
    private static int timesheetsFetched;
    public int getSignInCount()
    {
        return signInCount;
```

На рис. 10.2 представлен наш компонент JMX Bean в приложении JConsole (связываемый с JSE 5.0). Это прекрасный способ контроля состояния приложения, выполняющегося на дистанционном сервере, непосредственно на вашем рабочем столе! На мой взгляд, это великолепная возможность для разработчиков, в отличие от громоздких, хоть и намного больше надежных инструментальных средств, таких как OpenView от Hewlett-Packard или Unicenter от Computer Associate, которые предназначены для использования крупными организациями.

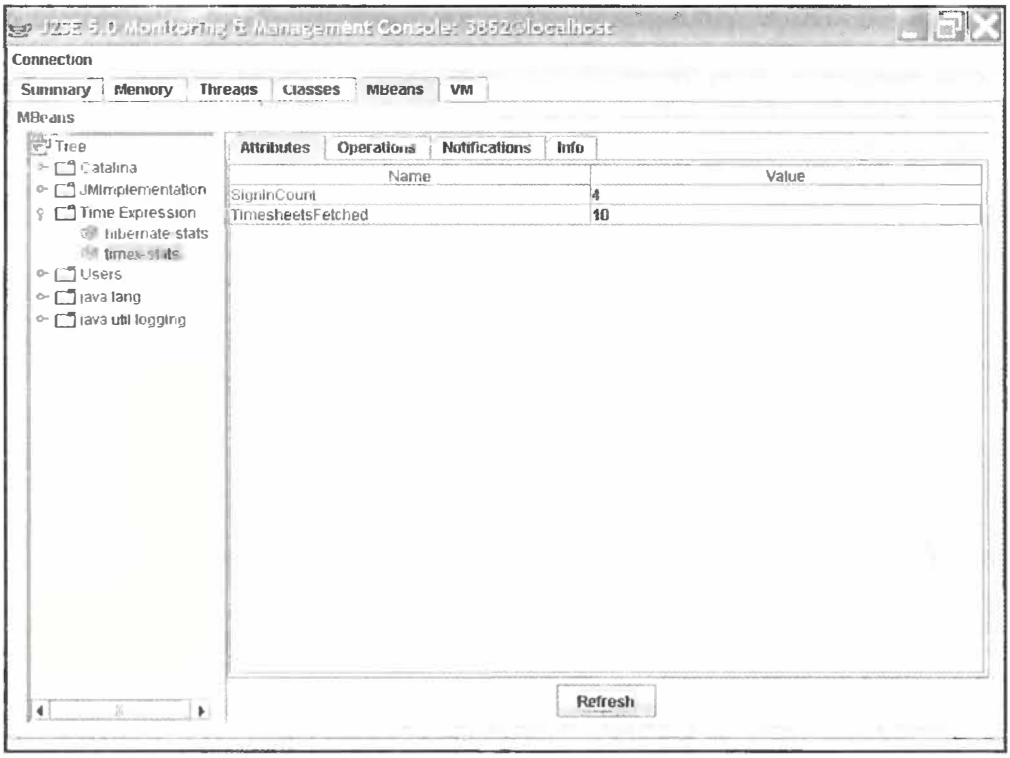



**Рис. 10.2.** Приложение JConsole с загруженным `TimexJmxBean`

## Еще о Spring

Напомню, что Spring столь обширная среда и настолько перегруженная различными средствами, что я могу описывать здесь только те из них, которые имеют отношение к нашему примеру приложения.

### Новые библиотеки дескрипторов

На момент написания этой книги наиболее современной версией Spring Framework была M4 (или RC1). Эта версия представляет несколько новых библиотек дескрипторов, облегчающих работу в JSP. К ним относятся: `form:form`, `form:input`, `form:password`, `form:hidden`, `form:select`, `form:option`, `form:radiobutton`, `form:checkbox`, `form:textarea` и `form:errors`.

Упомянутые библиотеки дескрипторов все еще нестабильны, и группа Spring Framework попросила меня не рассматривать их подробно, пока они не будут окончательно отлажены. Последняя документация и файлы доступны для загрузки на Web-сайте `springframework.org`.

### Поддержка Web-служб, JMS, JTA, EJB, DAO, RMI и JDBC

Среда Spring обеспечивает также поддержку для протоколов дистанционного доступа, таких как дистанционный вызов метода Java (Remote Method Invocation – RMI),

Web-службы (с использованием JAX-RPC), архитектура подключений Java (Java Connector Architecture – JCA), объекты доступа к данным (Data Access Object – DAO), поддержка различных средств объектно-реляционного связывания (Object-Relational Mapping – ORM), продукты, отличные от Hibernate (например, JDO и iBATIS), а также подключения баз данных Java (JDBC Java Database Connectivity), EJB, службы сообщений Java (Java Database Connectivity – JMS), API транзакцией Java (Java Transaction API – JTA) и т.д.

### Классы запуска

Возможно, вам знакома концепция *классов запуска* (startup class) или *сервлетов* (servlet) на Web-серверах и серверах приложений. Среда Spring не поддерживает концепцию классов запуска явно. Но ее легко реализовать, используя атрибут зависимости для элемента компонента. Большинство объектов автоматически создается в правильном порядке, поскольку среда Spring распознает зависимости. Но если вы имеете независимые классы, такие как класс `HibernateUtil`, атрибут зависимости может оказаться весьма полезным, чтобы гарантировать создание их экземпляров заранее, чтобы они были готовы, когда в них возникнет необходимость.

### Другое

Еще одна весьма популярная возможность среды Spring – это способность загрузить внешний файл свойств, как показано в этом примере:

```
<bean id="placeholderConfig"
        class= "org.springframework.beans.factory.config.PropertyPla
ceholderConfigurer">
    <property name="location" value="WEB-INF/classes/pas-servlet.prop
erties"/>
</bean>
```

После того как файл свойств загружен, он применяется для замены значений атрибута, связанного с компонентом, как показано далее:

```
<property name="url" value="${db.url}" />
```

# Гармония Spring и Hibernate

Среда Spring Framework обеспечивает первоклассную интеграцию с Hibernate. К некоторым из преимуществ интеграции этих двух технологий относится легкость проверки, однозначность данных исключений (например, `DataIntegrityViolationException`) и еще одно, ключевое преимущество для наших конкретных потребностей – декларативное управление транзакциями в облегченных контейнерах! (Более подробная информация о преимуществах среды Spring и интеграции с Hibernate содержится в справочной документации Spring на Web-сайте `springframework.org`.)

Теперь давайте рассмотрим, как мы можем использовать Spring для обеспечения декларативного управления транзакциями, переложив таким образом бремя на контейнер Spring и сосредоточившись на бизнес-логике (подробно декларативное управление транзакциями обсуждается позже).

Мы также увидим, как декларативное управление транзакциями сократит сроки создания кода некоторых из наших классов пакета `model` почти наполовину!

## Настройка управления транзакциями в среде Spring

До сих пор мы рассматривали приложение Time Expression в каталоге `timex/`. Однако файл архива кода этой книги содержит также подвергшуюся рефакторингу версию этого приложения в каталоге `timex2/`. Этот код демонстрирует интеграцию Spring и Hibernate, так что давайте проанализируем некоторые из файлов в этом каталоге.

Рефакторинг кода Java обсуждается в приложении Б, "Рефакторинг типового приложения". Здесь перечислим изменения, которые мы сделали в нашем файле `timex-servlet.xml`, переименованном теперь в `timex2-servlet.xml` (расположен в каталоге `timex2/`).

### Перенастройка нашего типового приложения

На рис. 10.3 демонстрируется, что наши классы контроллера (в каталоге `timex/`) используют непосредственно управляющие классы; обратите, например, внимание, как наш класс `EnterHoursController` непосредственно использует класс `DepartmentManager`. Этот прямой подход продемонстрирован далее в следующей конфигурации XML на базе нашего исходного файла `timex-servlet.xml`:

```
<bean name="enterHoursController"
    class="com.visualpatterns.timex.controller.EnterHoursController">
    <property name="departmentManager">
        <ref bean="departmentManager" />
    </property>
```

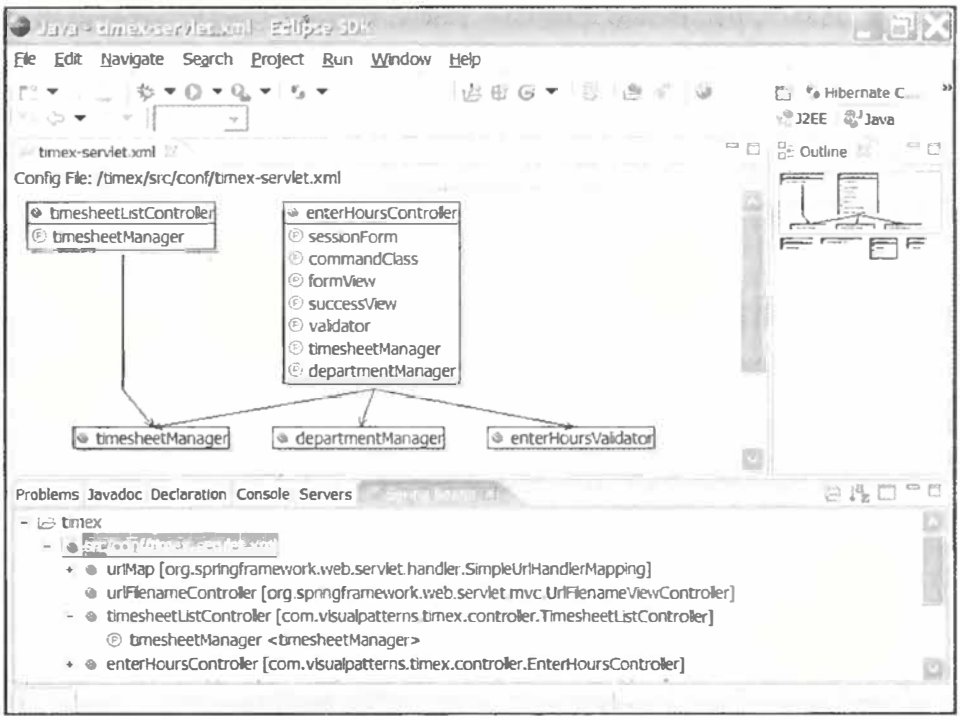



**Рис. 10.3.** Графическое представление `timex-servlet.xml`

Используя управляющие классы непосредственно, мы должны реализовать наше собственное управление транзакциями; другими словами, программное управление транзакциями.

Теперь давайте рассмотрим, как мы будем работать с нашими управляющими классами *косвенно* (indirectly), т.е. при помощи класса прокси-компонента транзакции.

На рис. 10.4 изображена структура IDE Spring для контроллера `EnterHoursController` и связанных с ним компонентов, как они выглядят в нашем подвергшемся рефакторингу файле `timex2-servlet.xml`. Обратите внимание, как класс `EnterHoursController` теперь проходит прокси-компонент по имени `departmentManagerProxy` (класс Spring `TransactionProxyFactoryBean`). Этот прокси-компонент предоставляет нам два главных преимущества.

- Управление сеансами Hibernate, чтобы мы не заботились о закрытии сеанса Hibernate вручную.
- Декларативное управление транзакциями внутри облегченных контейнеров с возможностью опускаться на уровень отдельных транзакций базы данных или подниматься до уровня *глобальных API транзакций Java* (Java Transaction API — JTA). Среда Spring предоставляет даже специальные классы поддержки управления транзакциями (например, `WebLogicJtaTransactionManager`) для таких продуктов, как *открытый менеджер транзакции Java* (Java Open Transaction Manager — JOTM; `jotm.objectweb.org`) от ObjectWeb, сервер приложений WebLogic от BEA (`bea.com`) и сервер приложений WebSphere от IBM (`ibm.com`).

Теперь рассмотрим код, лежащий позади схемы на рис. 10.4.

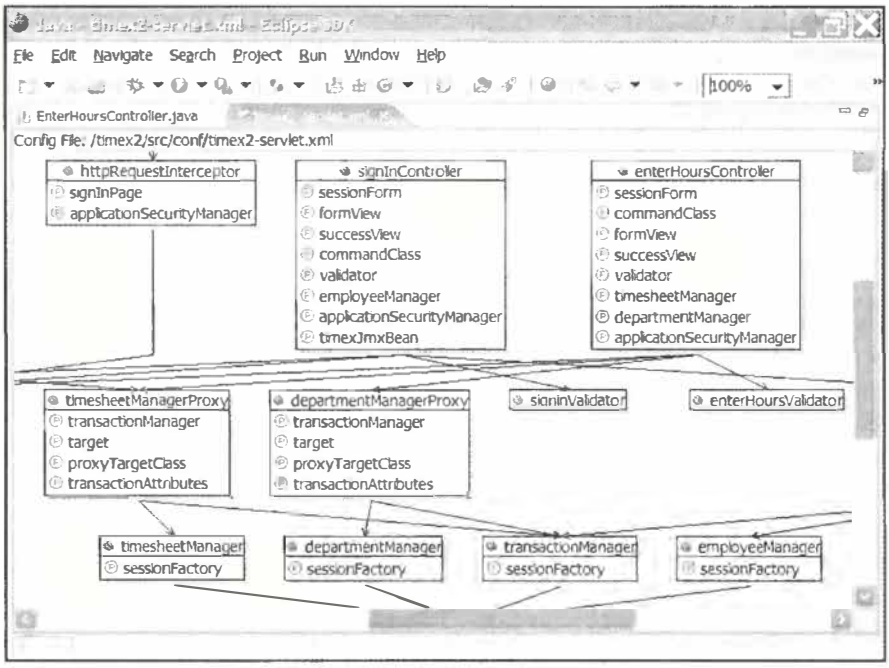



**Рис. 10.4.** Графическое представление `timex2-servlet.xml`

## Код Java проще и короче!

Одно из ключевых преимуществ *декларативного управления транзакциями* (declarative transaction management), как я упомянул ранее, — это то, что бремя управления транзакциями возлагается на контейнер. Это позволяет нам также уменьшить объем кода и сосредоточиться, в основном, на бизнес-логике, а не на низкоуровневом программировании, таком как управление транзакциями. Эта возможность была доступна в Enterprise JavaBeans, но, как я упомянул ранее, используя среду Spring, мы получаем те же самые средства корпоративного управления транзакциями в облегченных контейнерах типа Apache Tomcat.

Приведенный ниже отрывок исходного кода демонстрирует наш метод `saveTimesheet(Timesheet timesheet)` из класса `TimesheetManager` прежнего приложения Time Expression (расположенного в каталоге `timex/`). Здесь используется программное управление транзакциями Hibernate.

```
Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
session.beginTransaction();
try
{
    session.saveOrUpdate(timesheet);
    session.getTransaction().commit();
}
catch (HibernateException e)
{
    session.getTransaction().rollback();
    throw e;
}
```

**Примечание**

В приложении Б, "Рефакторинг типового приложения", приведен измененный код, обсуждаемый в этом разделе.

При помощи Spring тот же самый код может быть уменьшен до одной строки, как показано ниже (отрывок из класса `TimesheetManagerImpl1`, расположенного в каталоге `springhibernate/`):

```
this.SessionFactory.getCurrentSession().merge(timesheet);
```

В качестве альтернативы, мы можем дополнить вспомогательный класс Spring `HibernateDaoSupport`, который позволяет нам уменьшить объем кода Java еще больше, удалив методы `getSessionFactory` и `setSessionFactory`. Кроме того, исключения Hibernate автоматически преобразуются в четкую иерархию исключений данных.

Класс `HibernateDaoSupport` предоставляет метод `getHibernateTemplate`, который поддерживает методы, находящиеся обычно в интерфейсе Hibernate `Session`, как показано далее:

```
getHibernateTemplate().merge(timesheet);
```

Наши три управляющих класса, `DepartmentManager`, `EmployeeManager` и `TimesheetManager` (в каталоге `timex2/`), прошли рефакторинг так, чтобы дополнить класс `HibernateDaoSupport`. Теперь все эти классы вместе имеют меньше 126

строк кода! В табл. 10.1 сравнивается количество строк кода в этих трех файлах классов, использующих оба типа управления транзакциями.

**Таблица 10.1. Количество строк кода при программном и декларативном управлении транзакциями**

| Файл | Программное | Декларативное |
|---|---|---|
| `DepartmentManager.java` | 39 | 22 |
| `EmployeeManager.java` | 66 | 36 |
| `TimesheetManager.java` | 166 | 87 |
| Всего: | 271 | 145 |

Если только три простых класса способны настолько сократить объем кода, можете представить себе, насколько строк можно уменьшить код в типичном реальном корпоративном приложении Java с намного большим количеством классов этой категории (т.е. на сервисном уровне).

Хотя класс `HibernateDaoSupport` обеспечивает некоторые преимущества, ему также присущи некоторые незначительные недостатки, которые следует упомянуть.

- Он слишком сильно связывает Hibernate и Spring; если среда Hibernate подверглась обновлению, которое не совместимо с данной версией Spring, нам придется подождать, пока появится среда Spring Framework, поддерживающая это обновление.
- Поскольку Java поддерживает только одиночное наследование, после того, как мы унаследуем наш класс от класса `HibernateDaoSupport`, возможность наследования (расширения, дополнения) от другого класса будет утеряна; безусловно, мы могли бы расширить наш собственный специальный класс, который, в свою очередь, мог бы дополнить класс `HibernateDaoSupport`, чтобы обойти это ограничение.

Однако описанные ранее преимущества компенсируют эти незначительные недостатки.

## Проверка модуля интегрированного кода

Теперь, когда используется новый стиль управления транзакциями в каталоге `timex2/`, связанном с подвергшимся рефакторингу кодом, мы также должны по-другому использовать эти классы в наших проверках модуля. Например, чтобы использовать один из наших управляющих классов, мы должны теперь загружать прокси-компонент вместо класса `TimexTestCase`, как представлено в следующем отрывке кода:

```
FileSystemResource res =
    new FileSystemResource("src/conf/timex2-servlet.xml");
springFactory = new XmlBeanFactory(res);
departmentManager =
    (DepartmentManager)springFactory.getBean("departmentMana
↪gerProxy");
```

### Подход на базе интерфейсов

В архивном файле кода этой книги, в каталоге springhibernate/, имеется другой проект (имена файлов приведены в приложении A, "Загружаемый код этой книги"). Этот проект демонстрирует следующее.

- Как осуществлять настройку и программирование, используя интерфейсы и классы реализации.
- Как передавать исключение org.springframework.dao.DataIntegrityViolationException, чтобы убедиться в четкости работы исключений данных Spring, а также в том, как Web-среда Spring может переадресовывать пользователя к представлению (в данном случае, dberror.jsp) для отображения сообщения данного исключения.

Обратите внимание на следующее: поскольку здесь используется интерфейс-ориентированный подход, а не класс-ориентированный, как в приложении Time Expression, мы не должны использовать атрибут proxyTargetClass, как в нашем файле timex2-servlet.xml.

```
<bean id="departmentManagerProxy"
    class=
       "org.springframework.transaction.interceptor.TransactionProxy
↳FactoryBean">
    <property name="proxyTargetClass" value="true"/>
```

Рис. 10.5 демонстрирует графическое представление файла контекста приложения Spring springhibernate-servlet.xml, используемого в этом демонстрационном приложении. Обратите внимание на то, что атрибут proxyTargetClass не используется в прокси-классах.

> **Примечание**
> Здесь заслуживает внимания еще одна возможность Eclipse. Обратите внимание, как Eclipse позволяет нам работать с несколькими проектами в том же рабочем пространстве (на рис. 10.5, *слева*); в данном случае мы работаем с тремя нашими проектами: springhibernate, timex и timex2.

## Библиотеки дескрипторов JSP

Мы уже использовали несколько библиотек дескрипторов в этой книге, прежде всего, в главе 7, "Среда Spring Web MVC Framework", где мы обсуждали среду Web Spring MVC Framework. К этим библиотекам дескрипторов относятся spring:bind, JSTL и др. Одна из моих любимых библиотек дескрипторов — Displaytag, так что давайте рассмотрим ее первой.

### Библиотека Displaytag

Библиотека Displaytag, согласно ее Web-сайту displaytag.sourceforge.net, "является комплектом специальных дескрипторов с открытым исходным кодом, которые обеспечивают высокоуровневые схемы Web-презентации, способные работать в модели MVC. Библиотека предоставляет существенный объем функциональных возможностей, оставаясь в то же время простой в использовании".

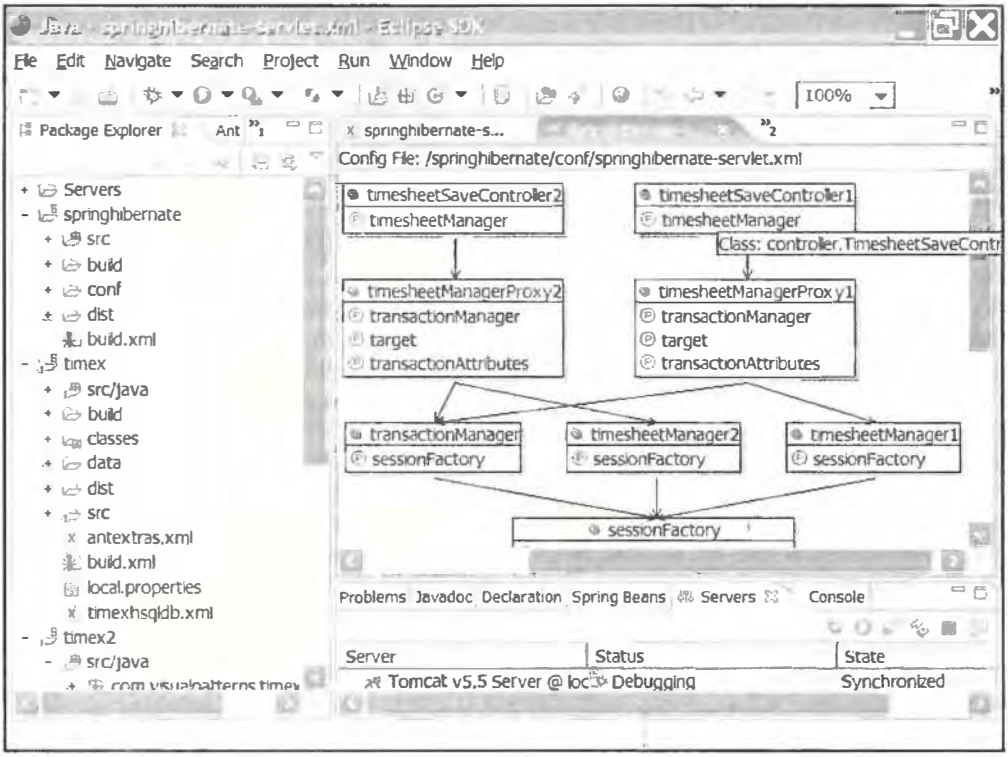



**Рис. 10.5.** Графическое представление `springhibernate-servlet.xml`

Я использовал эту библиотеку дескрипторов прежде и люблю ее за простоту и отказоустойчивость. Например, из менее чем 20 строк кода JSP мы могли бы создать сложную таблицу HTML, используя данные из базы данных через JDBC. Мы используем это для преобразования таблицы HTML на нашем экране Timesheet List, который мы первоначально создали с использованием комбинации дескрипторов JSTL `c:forEach` и `c:out`. Ключевое преимущество для нашего типового приложения — это возможность сортировать список, а также осуществлять экспорт в файлы формата PDF, CSV и другие, при необходимости.

Следующий отрывок кода из нашего подвергшегося рефакторингу файла `timesheetlist.jsp` (находится в каталоге `timex2/`) демонстрирует, как работает библиотека `Displaytag`:

```
<%@ taglib prefix="display" uri="http://displaytag.sf.net/el" %>
<display:table name="timesheets" id="timesheet" defaultsort="1"
        requestURI="timesheetlist.htm"
        cellpadding="5" cellspacing="0"
        export="false" class="tableborder">
<display:column property="department.name" sortable="true"
        title="Department"/>
</display:column>
```

Эту библиотеку дескрипторов можно загрузить по адресу displaytag.sourceforge.net, наряду с документацией по дополнительным возможностям Displaytag. Как объясняется на этом Web-сайте, она имеет несколько зависимостей. Для ее установки и применения в нашем примере приложения следуйте инструкции Web-сайта.

## Создание специальных библиотек дескрипторов

Хотя в сети доступно множество различных библиотек дескрипторов, иногда приходится создавать специальные библиотеки дескрипторов, специфические для вашего проекта. В прошлом мне пришлось создать специальную библиотеку дескрипторов, чтобы скрывать некоторые кнопки на экране, в зависимости от уровня прав пользователя.

Наш файл PayPeriodCheckTag.java является примером полнофункциональной специальной библиотеки дескрипторов, используемой в приложении Time Expression. Эта библиотека дескрипторов – просто пример применения библиотеки дескрипторов. Данная конкретная библиотека дескрипторов проверяет текущую дату относительно начальной и конечной дат отчетного периода. Если текущая дата указана вне этого диапазона дат, кнопка Save (Сохранить) на экране не отображается, чтобы пользователь не мог ошибочно модифицировать текущий табель учета рабочего времени.

Приведенный ниже фрагмент из файла PayPeriodCheckTag.java демонстрирует, как мы дополняем класс javax.servlet.jsp.tagext.TagSupport и предоставляем специальную библиотеку дескрипторов.

```
public class PayPeriodCheckTag
    extends TagSupport
{
    private Date checkDate;
    public int doStartTag() throws JspException
    {
        boolean includeText = (DateUtil.isInCurrentPayPeriod(check
↳Date));
        if (includeText)
            return TagSupport.EVAL_BODY_INCLUDE;

        return TagSupport.SKIP_BODY;
    }
```

Следующий отрывок кода демонстрирует выдержку из *описания библиотеки дескрипторов* (Tag Library Descriptor – TLD), файл timex.tld.

```
<shortname>timex</shortname>
<tag>
    <name>periodcheck</name>
    <tagclass>com.visualpatterns.timex.util.PayPeriodCheckTag</ta
↳gclass>
    <attribute>
        <name>checkDate</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
```

Этот отрывок кода из нашего файла `enterhours.jsp` демонстрирует использование дескриптора `timex:periodcheck`, применяемого для отображения или сокрытия кнопки Save.

```
<%@ taglib prefix="timex" uri="/WEB-INF/timex.tld"%>
<timex:periodcheck checkDate="${command.periodEndingDate}">
    <input name="save" type="submit" value="Save">
</timex:periodcheck>
```

# Рефакторинг

На протяжении этой книги я столь часто затрагивал рефакторинг, что может сложиться впечатление, что я его проповедую. Начиная с главы 5, "Применение Hibernate для постоянных объектов", рефакторингу подверглось множество кода.

Как я упоминал ранее, рефакторинг — это не новая концепция; относительно новым является сам термин, который был введен Мартином Фаулером (`refactoring.com`). *Рефакторинг* (refactoring)[1] — это процесс улучшения внутренней реализации кода без воздействия (или с минимальным воздействием) на внешний интерфейс. Это то, что разработчики делают ежедневно и делали уже на протяжении многих лет; сейчас мы только получили общий термин для обозначения этой практики. В приложении Б, "Рефакторинг типового приложения", демонстрируется часть кода, который подвергался рефакторингу в главах 5, "Применение Hibernate для постоянных объектов", 7, "Среда Spring Web MVC Framework", и этой главе. Это отражает реальное положение дел при разработке! Другими словами, со временем мы переделываем код, поскольку обнаруживаем лучшие способы его реализации.

## Примеры рефакторинга в нашем типовом приложении

Ниже приведено несколько примеров рефакторинга кода Java и JSP в приложении Time Expression. Позвольте еще раз подчеркнуть, что рефакторинг не сродни ядерной физике, и это не технология, которая перевернет мир, как некоторым могло бы показаться, а простой и очевидный способ улучшения кода, таков сам характер рефакторинга.

> **Примечание**
> Приложение Б, "Рефакторинг типового приложения", демонстрирует большинство связанных с рефакторингом изменений кода, упомянутых здесь.

### Примеры рефакторинга Java

Ниже приведено несколько примеров рефакторинга кода Java, связанных с приложением Time Expression.

- Переход от программного управления транзакциями к декларативному. Как мы обсуждали ранее, для существенного рефакторинга наших трех управляющих классов обязательна интеграция Hibernate со Spring.
- Суперкласс извлечения. Web-сайт `refactoring.com` описывает "суперкласс извлечения" ("Extract Superclass") как тип рефакторинга, определяя его следующим образом: "Вы имеете два класса с подобными возможностями. Создай-

[1] Или реорганизация. — *Примеч. ред.*

те суперкласс и переместите общие возможности в него". Мы сделали это только для нашего проверочного класса JUnit по имени `TimexTestCase` (обсуждался ранее).

- Перемещение метода. Я переместил метод `getCurrentPeriodEndingDate` из класса `TimesheetListController` в класс `DateUtil`, поскольку этот метод был нужен мне в нескольких местах, и его имело смысл переместить в общий вспомогательный класс. На `refactoring.com` это называется *перемещение метода* (Move Method).
- Перемещение класса. Первоначально я поместил класс `ReminderEmail` в пакет `util`; однако впоследствии я переместил его в пакет `job`, поскольку это более подходящее место для данного класса.
- Удаление неиспользуемых переменных. Благодаря предупреждениям Eclipse о неиспользуемых элементах Java, которые мы получали в главе 8, "Феномен Eclipse!", я обнаружил наличие некоторых неиспользуемых переменных и операторов импорта. Я удалил большинство из них без воздействия на внешний интерфейс соответствующих классов или методов.

### Рефакторинг JSP

Приведенный ниже список описывает некоторые из элементов рефакторинга, связанных с JSP.

- Перемещение сообщения об ошибках для JSP в общий подключаемый файл. Поскольку мне необходимо дублировать представление с сообщениями о состоянии и ошибках, я перенес их в общий файл `includemessages.jsp` и включил его в необходимые файлы (например, `enterhours.jsp timesheetlist.Jsp` и `signin.jsp`).
- Переход на `displaytag`. После обнаружения лучшего способа отображения и сортировки таблицы HTML, я перенес весь код на библиотеку `Displaytag` и отказался от использования для этого JSTL. Это демонстрирует также ключевое преимущество схемы проектирования MVC, т.е. мы смогли изменить представление без воздействия на модель.

## Рефакторинг беспощаден, но ... экономит код

Возможно, вы слышали выражение *рефакторинг беспощаден* (refactor mercilessly); это один из принципов экстремального программирования (`extremeprogramming.org`). Я полагаю, что с этим можно согласиться, поэтому предостерег бы вас по поводу необходимости сохранять резервные копии вашего кода либо в хранилище исходного кода, типа CVS, либо в отдельном каталоге. Это позволит вернуться к первоначальному рабочему коду, если рефакторинг не сработает так, как предполагалось. Например, работая над этой книгой, я слишком сильно подверг рефакторингу некоторый код, пытаясь внести в него нечто новое. Но поскольку это не сработало так, как я планировал, мне удалось быстро вернуться к сохраненной ранее версии кода. Возможность Eclipse Local History (обсуждалась в главе 8, "Феномен Eclipse!") может также использоваться для восстановления предыдущей версии кода.

### Сетевые каталоги рефакторинга (`refactoring.com` и `agiledata.org`)

Рефакторинг возможен не только для кода, он может быть также применен на уровне базы данных. Более подробная информация о рефакторинге кода приведена на сайте `refactoring.com`; там имеется великолепный каталог методов рефакторинга (почти 100, на момент написания этой книги).

Для изучения методов рефакторинга базы данных посетите сайт `agiledata.org`. Там есть длинный список общих методов рефакторинга для баз данных (почти 70, на момент написания этой книги). Вы могли бы заметить, что уже используете большинство этих методов рефакторинга.

### Замечание о рефакторинге в Eclipse

Как упоминалось в главе 8, Eclipse предоставляет несколько способов рефакторинга кода Java.

Например, для демонстрации интеграции Spring и Hibernate (в каталоге `springhibernate/`), как упоминалось ранее в главе, я использовал пункт Extract Interface (Извлечение интерфейса) меню рефакторинга Eclipse, чтобы автоматически создать файл интерфейса и получить конкретные классы, реализующие этот интерфейс.

## Другие вопросы

В этой книге и, в частности, в этой главе мы рассмотрели большой объем базовой информации, но описать все способы разработки надежных и безопасных приложений в одной книге практически невозможно. Однако следующие разделы содержат краткий обзор дополнительных элементов, которые следует учесть при проектировании и разработке реальных приложений. Обратите внимание на то, что большинство этих аспектов я рассматриваю здесь на концептуальном уровне.

### Управление транзакциями

*Транзакция* (transaction) — это серия операций, которые должны либо завершиться успехом *все*, либо система должна остаться в исходном состоянии. Самый простой пример для транзакции — это две таблицы в той же базе данных: одна — родительская, вторая — дочерняя. Если мы удалим записи из родительской таблицы, должны будем также удалить записи из дочерней. Но если в дочерней таблице при удалении произойдет сбой, нам придется осуществить *откат* (roll back) транзакции и восстановить базу данных в ее первоначальном состоянии. Хоть это только один пример операции с базой данных, та же концепция применяется для всех баз данных.

Обеспечение транзакционной целостности — это жизненно важная часть любого корпоративного приложения. Как правило, управление транзакциями осуществляется двумя способами: программно или декларативно. В то время как программное управление транзакциями предоставляет разработчику управление в коде, декларативное управление транзакциями позволяет нам управлять точками демаркации транзакции и уровнем изоляции, манипулируя файлами XML (файлами конфигурации). Кроме того, в то время как программное управление транзакциями статично по характеру (поскольку осуществляется в коде), декларативное управление транзакциями

по характеру более динамично, поскольку оно может быть отложено до времени развертывания.

Ныне декларативное управление транзакциями играет большую роль в корпоративной распределенной системе, поскольку управление транзакциями больше не применяется только для одиночных баз данных (локальная транзакция), ведь организации стремятся хранить свою информацию в нескольких базах данных (глобальная транзакция).

**Примечание**

Мы уже видели примеры программного управления транзакциями с использованием Hibernate в главе 5, "Применение Hibernate для постоянных объектов", и декларативное управление транзакциями с использованием среды Spring ранее в этой главе. Файл архива кода этой книги содержит примеры обоих типов управления транзакциями.

Код программного управления транзакциями и файлы конфигурации находятся в каталоге `timex/`, а код декларативного управления транзакциями с использованием Spring — в каталоге `timex2/`.

Декларативное управление транзакциями смещает бремя управления транзакциями на контейнер (например, контейнер EJB) и позволяет разработчику сосредоточиться на бизнес-логике. Кроме того, это может сделать код немного надежнее, поскольку код управления транзакциями не смешивается с кодом бизнес-логики. Как мы обсуждали ранее в этой главе, среда Spring Framework обеспечивает поддержку для декларативного управления транзакциями, не требуя, чтобы ваши приложения выполнялись в контейнере EJB; другими словами, ваше приложение может выполняться внутри контейнера сервлета, такого как Apache Tomcat. Чтобы понять, как работает декларативное управление транзакциями, давайте рассмотрим некоторые связанные с ним концепции.

Корпоративные транзакции обычно соответствуют свойствам ACID (Atomicity, Consistency, Isolation и Durability — *неделимость*, *последовательность*, *изоляция* и *устойчивость*). Транзакции ACID гарантируют, что либо *все* операции пройдут успешно, либо *все* останется так, как было до начала транзакции, т.е. *все* или *ничего*. Неделимость гарантирует, что либо все операции в транзакции окажутся успешны, либо система будет восстановлена в ее первоначальном состоянии. Последовательность гарантирует, что все преобразования данных транзакций из одного состояния в другое будут происходить последовательно. Изоляция означает, что транзакции изолированы от других транзакций, пока они не завершены. Устойчивость означает, что после того, как все транзакции будут завершены успешно, изменения разрешаются.

Ниже приведены некоторые термины, относящиеся к корпоративному управлению транзакциями.

- *Демаркация* (demarcation). Это способ маркировки границ для данной транзакции. В распределенной системе вы могли бы иметь одни транзакции внутри других (это называется вложенными транзакциями).
- *Уровень изоляции транзакций* (transaction isolation level). Это уровень отделения одной транзакции от других. Другими словами, это определяет, какую часть работы одной транзакции может видеть другая транзакция. Хотя различные технологии используют различные термины для уровней изоляции, среда Spring Framework содержит следующие уровни: `ISOLATION_READ_COMMITTED`, `ISOLATION_READ_UNCOMMITTED`, `ISOLATION_REPEATABLE_READ` и

ISOLATION_SERIALIZABLE. Обратите внимание, ISOLATION_READ_UNCOMMITTED — самая низкая форма изоляции, поэтому она допускает прямое чтение и запись, однако она обеспечивает и наиболее высокую производительность. Уровень ISOLATION_SERIALIZABLE, напротив, является самой высокой и самой надежной формой, соответственно, производительность из-за дополнительных гарантий страдает.

- *Распространение транзакции* (transaction propagation). Это подразумевает, должна ли группа кода выполняться внутри ее собственной транзакции или участвовать в существующей транзакции, которая могла бы уже начаться до достижения этого фрагмента кода. Например, среда Spring поддерживает следующие типы распространения транзакции: PROPAGATION_MANDATORY, PROPAGATION_NESTED, PROPAGATION_NEVER, PROPAGATION_NOT_SUPPORTED, PROPAGATION_REQUIRED, PROPAGATION_REQUIRES_NEW и PROPAGATION_SUPPORTS.

Напомню, что декларативное управление транзакциями в EJB поддерживается довольно давно; однако среда Spring Framework позволяет получить такое управление транзакциями на корпоративным уровне даже в контейнере сервлета и не вынуждает нас использовать контейнер EJB. Так, если мы хотим повысить надежность кода приложения Time Expression, могли бы перейти на использование декларативного управления транзакциями Spring и удалить весь программный код управления транзакциями Hibernate. Более подробная информация о коде декларативного управления транзакциями Spring приведена на Web-сайте springframework.org.

## Защита приложения

Это обширная тема сама по себе, которая достойна нескольких книг. Однако давайте сделаем краткий обзор некоторых концепций защиты, в контексте нашего типового приложения, к которым относились бы аутентификация, авторизация и шифрование (рис. 10.6).

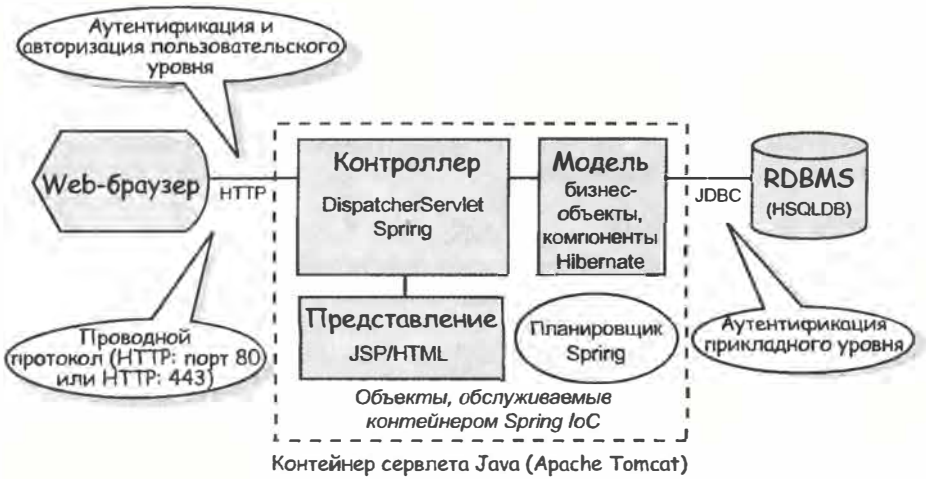



**Рис. 10.6.** Концепции защиты приложения (аутентификация, авторизация, шифрование)

### Аутентификация

Сюда относится аутентификация пользовательского уровня и аутентификация прикладного уровня. *Аутентификация* (authentication) — это процесс опознавания, который гарантирует, что пользователь является тем, за кого себя выдает. В приложении Time Expression аутентификация пользовательского уровня — это просто экран регистрации, где пользователь обозначает себя (например, вводит идентификатор и пароль). Аутентификация прикладного уровня — это проверка удостоверения приложения, обязательного для подключения к базе данных. В реальных проектах это может также подразумевать соединение с сервером LDAP и сервером службы сообщения Java (Java Message Service — JMS), например.

### Авторизация

*Авторизация* (authorization) контролирует доступ к средствам на основании роли пользователя. Например, в главе 2, "Простое приложение: сетевая система учета рабочего времени", мы определили различные типы ролей в наших пользовательских историях, например Employee (сотрудник), Manager (менеджер), Executive (управляющий) и Accounting (бухгалтер). Каждый из обладателей этих ролей имел бы различную степень доступа; например, сотрудники не могут начислять себе зарплату.

### Шифрование

В контексте нашего типового приложения *шифрование* (encryption) могло бы осуществляться в двух местах: проводной протокол и файлы конфигурации. В качестве *проводного протокола* (wire protocol), как правило, используется протокол HTTP (порт 443), обычно его вполне достаточно для защиты данных, передаваемых по проводам. Даже если эти данные будут перехвачены, они не будут расшифрованы. Шифрование может также осуществляться в файлах конфигурации. Если вы не хотите хранить пароли прикладного уровня (например, идентификатор и пароль пользователя базы данных) в виде открытого текста, то для шифрования строк в ваших файлах конфигурации вы могли бы использовать такой алгоритм, как SHA или MD5. Между прочим, это можно сделать, используя *криптографическое расширение Java* (Java Cryptography Extension — JCE). Более подробная информация по этой теме приведена на Web-сайте `java.sun.com`.

### Другое

Как я упоминал ранее, защита — это обширная тема, которой посвящено множество книг. Но в приложении Г, "Защита Web-приложений", я предоставил краткий список средств защиты Web-приложения.

## Обработка исключений

Исключения бывают двух разновидностей: *непроверяемые* (unchecked) и *проверяемые* (checked). Непроверяемые исключения (например, `java.lang.NullPointerException`) не обязательны к обработке кодом, в то время как проверяемые исключения требуют обязательной обработки в коде или повторной передачи в результате вызова оператора `throw`. Существуют также ошибки (например, `OutOfMemoryError`), поправить которые обычно довольно трудно.

Принять решение о том, расположить ли весь код обработки исключений приложения вместе или распределить его, довольно трудно. Недавно я прочитал следующее в руководстве по Java от Sun (на `http://java.sun.com/docs/books/tutorial/`):

"Если клиент ожидает, что приложение сможет справиться с причиной исключения, делайте это исключение проверяемым. Если клиент ничего не сможет сделать, чтобы ликвидировать причину исключения, делайте его непроверяемым".

Еще одной причиной перехвата и повторной передачи исключения (возможно, с переупаковкой в новое исключение) является организация некой централизованной системы регистрации и оповещения (например, по электронной почте или пейджеру).

### Гипотетическая схема обработки исключений

С учетом приведенных выше причин, можем заметить, что подход к обработке исключений приложения мог бы зависеть от свойств приложения. Рис. 10.7 демонстрирует, как наше типовое приложение могло бы использовать обработку исключения.


**Рис. 10.7.** Обработка исключений в приложении Time Expression

- `com.visualpatterns.timex.model`. Любые исключения, которые передаются в пакете `model` (например, ошибки базы данных), могут быть переупакованы как класс `ModelException` и повторно переданы как непроверяемое исключение. Точно так же пакет `job` мог бы использовать класс `JobException`, а пакет `util` – класс `UtilException`. При желании мы могли бы определить даже более подробные исключения, например `IllegalTimesheetHoursException`, но я обычно использую один класс исключения на один пакет (это мое личное предпочтение).
- `com.visualpatterns.timex.controller`. Любое исключение, переданное в пакете `controller`, может быть проверено; в пользовательском интерфейсе отображено сообщение об ошибке. Здесь, например, могла бы осуществляться проверка правильности ввода данных в поля UI, а также анализ связанных с этим исключений (например, `Integer.parseInt`).

И последнее замечание об исключениях. Во многих проектах исключения скрывают, используя код следующего типа; как правило, это не рекомендуемый подход, но иногда он допустим, если исключение на самом деле можно игнорировать:

```
try { // некий код, который приводит к исключению ... }
catch (Exception e) { e.printStackTrace(); }
```

### Обработка исключений с использованием среды Web Spring MVC Framework

Среда Web Spring MVC Framework предоставляет удобную и практичную схему обработки исключений. А именно, среда Spring допускает настройку имен классов исключений для отображения; это позволяет обработать непредвиденные исключения весьма элегантно, отображая соответственно оформленную Web-страницу.

Ниже приведен отрывок кода из нашего файла `springhibernate-servlet.xml` (из каталога `springhibernate/`), демонстрирующий, как исключение `org.springframework.dao.DataIntegrityViolationException` может быть сопоставлено с нашим файлом представления `dberror.jsp` (обсуждавшимся ранее в этой главе).

```
<bean id="exceptionResolver"

class="org.springframework.web.servlet.handler.SimpleMappingException
↳Resolver">
    <property name="exceptionMappings">
        <props>
        <prop key="org.springframework.dao.DataIntegrityViolationEx
↳ception">
        dberror
```

Приведенный ниже отрывок кода нашего файла представления `dberror.jsp` демонстрирует, как среда Spring передает объект исключения, используя атрибут сеанса запроса по имени `exception`, и позволяет отобразить его стек трассировки:

```
<%
    Exception ex = (Exception)request.getAttribute("exception");
    if (ex != null)
        ex.printStackTrace(new java.io.PrintWriter(out));
%>
```

# Кластеризация

Согласно сайту `webopidia.com`, *кластеризация* (clustering) определена как "два или несколько компьютеров, совместно работающих как один компьютер. Кластеризация используется для параллельной обработки, распределения нагрузки и повышения отказоустойчивости".

Кластеризация может быть применена ко всему, от серверов приложений до баз данных и сетевых устройств. В мире Java кластеризация обычно используется для повышения стабильности и надежности корпоративных приложений. Например, с точки зрения масштабируемости это может означать применение сеансов HTTP и безотказной архитектуры компонентов.

Этой теме посвящено множество статей, однако позвольте мне предоставить вам некий минимум принципов кластеризации приложений Java.

- Осуществляйте сериализацию объектов, безотказную архитектуру которых мы хотим обеспечить (например, объектов пользователя, созданных после завершения сеанса регистрации). Кластеризация может быть осуществлена только для базовых типов Java и объектов типа `Serializable`.
- Не используйте для хранения информации статические переменные, поскольку они не реплицируются и будут применимы только на одном экземпляре серве-

ра. Например, используя одноэлементный компонент на одном сервере, мы не получим той же информации на другом сервере. Это, возможно, одна из причин, почему среда Spring Framework не приветствует использование объектов `Singleton` и рекомендует позволить среде Spring обрабатывать их самой.

- Ограничьте количество данных в компонентах, безотказность которых следует обеспечить. Это может снизить производительность сервера приложений, например, потому, что потребуется реплицировать эту информацию для нескольких экземпляров.
- Кластеризация серверов приложений (и Web-серверов) осуществляется в оперативной памяти для постоянных данных. Например, ваши данные. которые являются постоянными в базе данных, но не в памяти, не будут реплицированы сервером приложений (Web-сервером).
- И вообще, боритесь при проектировании за простоту тех классов, которые вы хотите сделать кластеризуемыми.

# Многопоточность

В мире платформы Java Enterprise Edition (JEE) *многопоточность* (multithreading) обычно не приветствуется, поскольку сервер приложений – это единственная программа, которая, как предполагается, имеет контроль над потоками внутри JVM. Это гарантирует стабильную среду сервера приложений. Но иногда в приложении JEE вам может понадобиться многопоточность. Изолируйте многопоточность в одном классе, где мы сможем лучше управлять потоками, чем, например, в нескольких классах, реализующих собственные потоки. Независимо от того, как многопоточность спроектирована и используется в приложении, стоит подумать и о сопутствующих проблемах, таких как синхронизация, блокировка и др.

В JSE 5.0 представлен пакет `java.util.concurrent`, разработанный ранее Дугом Ли (Doug Lea). *Примечание:* для J2SE (до версии 5.0) используйте обратно совместимую версию, которая находится по адресу `http://gee.cs.oswego.edu/dl/`. Это самый корректный способ использования многопоточности в ваших корпоративных приложениях.

# Замечания о приложениях GUI Java

В этой книге мы не рассматривали создание клиентских приложений вообще, поскольку наше приложение имеет пользовательский Web-интерфейс. В настоящее время ведется скрытая война между Swing для Java и SWT (Standard Widget Toolkit – стандартный комплект инструментальных средств управления) от IBM. Swing – это комплект инструментальных средств, ориентирующийся на GUI, для JSE, а SWT – это графическая библиотека, доступная на платформе Eclipse. Обе технологии не зависят от платформы.

SWT – это тонкая оболочка вокруг элементов управления GUI базовой операционной системы, имеющая высокоуровневый интерфейс JFace. Swing, напротив, имеет собственный визуальный интерфейс, внешний вид которого подстраивается под внешний вид базовой операционной системы. Обе эти технологии имеют свои преимущества и недостатки. Например, если вы читаете в сети статьи по этой теме или просматриваете сообщения в различных сетевых форумах, то, вероятно, слышали,

что технология Swing слишком перегружена и работает медленнее, чем SWT. С другой стороны, выбор SWT подразумевает платформу Eclipse, а это, конечно, заслуживает внимания. По моему убеждению, если вы разрабатываете приложение GUI только для Microsoft Windows, выбор SWT будет наилучшим. Но если ваше приложение предназначено для продажи или передачи внешним клиентам, Swing может быть лучшим выбором. Напомню, это мое личное мнение. Вам стоит исследовать эти технологии далее, если у вас есть потребность в данном типе функциональных возможностей.

Еще один фактор, который следует учитывать для приложений GUI, – это технология Java Web Start от Sun, которая позволяет запускать приложения одним щелчком, независимо от Web-браузера. Приложение также может быть запущено при помощи ярлыков на рабочем столе, что делает запуск Web-приложения похожим на запуск обычного приложения. Само приложение кэшируется локально, поэтому если нет никаких обновлений, оно сработает немедленно. В противном случае, если приложение было обновлено, новая версия может быть загружена из сети (кстати, это стало возможным благодаря спецификации *сетевого протокола запуска Java* (Java's Network Launching Protocol – JNLP)). Более подробная информация о SWT приведена на Web-сайте `eclipse.org`, а о технологиях Java Swing и Web Start – на Web-сайте `java.sun.com`.

# Среда управления конфигурацией

Вы могли бы заметить следующие строки кода в нашем файле Ant `build.xml`:

```
<property name="env" value="local"/>
<property file="local.properties"/>
```

Хотя файл свойств `local.properties` задан по умолчанию, он может быть легко заменен другим файлом, например `test.properties`. Для этого используется команда вроде следующей (в командной строке):

```
ant -Denv=test
```

Это позволяет использовать разные файлы для развертывания в различных средах. Например, рис. 10.8 демонстрирует, как мы могли бы обеспечить ту же архитектуру для разных сред, от разработки до рабочей, используя индивидуальные имена хостов и серверов, Web-сервера и сервера баз данных в каждой среде.

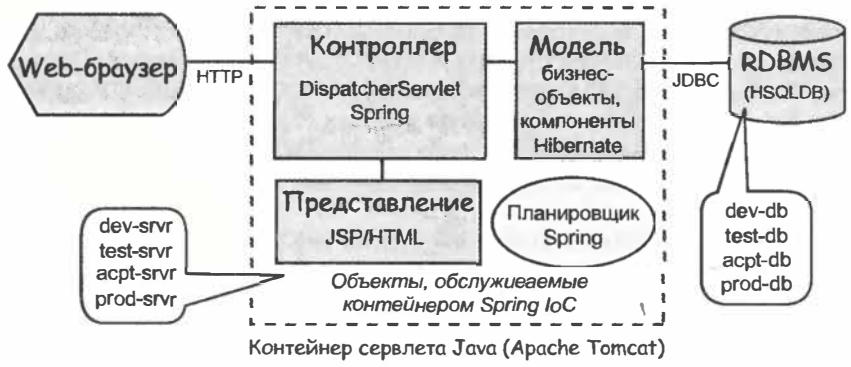



**Рис. 10.8.** Наше типовое приложение в различных средах

Использование внешних файлов, вместо внедрения всего необходимого в файл Ant типа `build`, предоставляет нам преимущество хранения пароля и другой информации в отдельном, возможно меньшем и более простом, файле свойств, в отличие от большого файла `build.xml`. Например, во многих компаниях я нередко встречал следующие среды.

- *Local* (локальная). Индивидуальная разработка на персональном компьютере.
- *Dev* (разработки). Интегрированная групповая разработка на сервере.
- *Test* (проверки). Используется для проверки функций (называется также *гарантией качества* (Quality Assurance – QA)).
- *Входной контроль пользователя* (User-Acceptance Testing – UAT), называемый также *областью постановки* (staging area), используется для проверки пользователями.
- *Production* (рабочая). Рабочее состояние приложения.

Для небольших проектов может оказаться вполне достаточно одной или двух из приведенных выше сред. Например, если в проекте используются меньшие итерации (например, двухнедельные циклы) или небольшие группы (по два–четыре разработчика), для проверки функций применяется интегрированная среда разработки. В качестве альтернативы, среды UAT и Test могли быть объединены в одну. Поэтому выбирайте те среды, которые лучше подходят вашим потребностям. Короче говоря, наш файл Ant может приспосабливаться к различным средам при наличии различных файлов свойств и использовании приведенного ранее параметра `-Denv`.

# Асинхронный JavaScript и XML

*Асинхронный JavaScript и XML* (Asynchronous JavaScript and XML – AJaX) – это Web-технология, которая позволяет нам сделать страницы интерактивными. AJaX – это мощнейшая концепция по двум причинам. Во-первых, она предоставляет пользователю более удобные условия, поскольку не нужно перезагружать всю страницу. Во-вторых, она перекладывает часть бремени на клиента, поскольку браузер запрашивает только подмножество элементов экрана (например, модифицируемые биржевые цены или оповещения), вместо того чтобы перезагружать всю страницу.

В архивном файле кода этой книги я привел чрезвычайно простой пример (файлы `rapidjava-ajax.html` и `rapidjava-ajax.jsp`). Хоть этот пример, безусловно, и не демонстрирует всю мощь технологии AJaX, он должен дать вам представление о том, чего мы можем достичь с использованием этой технологии. По существу, AJaX использует объект JavaScript `XMLHttpRequest` в комбинации с дескриптором HTML `<div>`, что обеспечивает динамическое обновление только части страницы. Для ознакомления с вводной информацией по AJaX весьма хорош Web-сайт `http://en.wikipedia.org/wiki/AJAX`. Однако по этой теме доступно множество книг и статей в сети; воспользуйтесь таким поисковым сервером, как Google, и получите множество ссылок по этой теме.

Опробуйте также DWR (Direct Web Remoting – прямой дистанционный доступ к Web); это среда с открытым исходным кодом, доступная по адресу `http://getahead.ltd.uk/dwr/`. Согласно этому Web-сайту, "DWR – это упрощенный AJaX для Java". Далее Web-сайт поясняет, что DWR "обеспечивает способ вызова

кода Java на сервере непосредственно из кода JavaScript в браузере". Обратите внимание на то, что я не работал с DWR лично. И последнее замечание об AJaX. То, что продемонстрировано в моих примерах, чрезвычайно примитивно. Мощность AJaX проявляется тогда, когда небольшая часть Web-интерфейса может осуществлять запросы к отдельным службам из коллекции служб на другом конце, другими словами, *архитектура, ориентированная на службы*, (Service-Oriented Architecture – SOA) – это очень мощная концепция!

# Документация и комментарии

С учетом акцентов, которые я расставил в коде, (а также физической структуры базы данных и данных) очень важно добавить к нашему коду документацию! Одни разработчики должны быть способны понять код других разработчиков, поэтому комментарии и документация Javadoc для сложного кода – большая помощь. Всегда думайте о других; представьте себе их мнение, когда они будут работать с вашим кодом, и постарайтесь упростить их работу. Однако нужно также знать меру в количестве документации, которую вы предоставляете; я предпочитаю минимум документирования в моем коде, только достаточное количество информации о данных, классах, методах, переменных и наиболее важной, а также сложной логике.

В свете сказанного выше, обратите внимание на то, что весь наш код имеет комментарии. Обратите также внимание на то, что комментариев в коде минимум, их не слишком много и не слишком мало. При наличии этих комментариев я могу выполнить команду JDK `javadoc` для моего кода Java, чтобы создать исчерпывающую документацию по коду для нашего типового приложения.

# Вся система в одном файле WAR!

И наконец, обратите внимание на то, что приложение Time Expression представляет собой развертываемые файлы архива Web-приложения (`timex.war` и `timex2.war`), содержащие систему Time Expression полностью (как показано на рис. 10.9). Напомню, что эти файлы WAR могут быть созданы с использованием нашего сценария Ant `build.xml` или возможностей экспорта Eclipse (см. главу 8, "Феномен Eclipse!").

Замечательным является тот факт, что этот файл WAR содержит весь наш исходный код Java и откомпилированные классы, статические и динамические Web-файлы, зависимости (библиотеки сторонних производителей), средства планирования задач и даже реляционную базу данных!

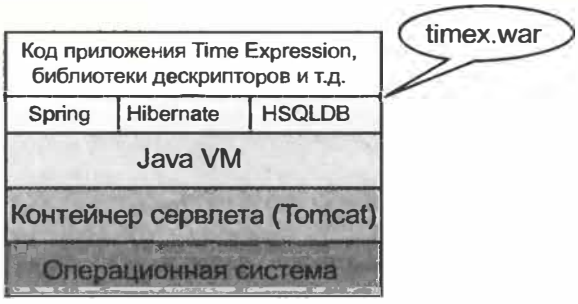



**Рис. 10.9.** Файл `timex.war` – полностью самодостаточная система

Точка зрения, которую я пытаюсь пропагандировать, — это мощнейшие возможности Java. Существует очень немного языков, которые обладают такими возможностями.

# Резюме

В этой главе мы рассмотрели следующие темы.

- Новые возможности Java
- Дополнительные встроенные и внешние задачи Ant
- Специальные комплекты JUnit
- Дополнительные возможности Hibernate
- Другие возможности среды Spring Framework
- Интеграция Hibernate и Spring
- Библиотека дескрипторов Displaytag и создание специальных библиотек дескрипторов
- Пример рефакторинга в нашем приложении
- Другие вопросы, такие как управление транзакциями, безопасность, обработка исключений, кластеризация и т.д.
- Простой пример AJaX

Ну хорошо, книга почти закончена! В следующей главе я ознакомлю вас с некоторыми руководствами, которые имеет смысл изучить впоследствии.

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, содержится на следующих Web-сайтах:

- Agile Data (рефакторинг базы данных и т.д.) `http://agiledata.org/`
- Apache Ant `http://ant.apache.org/`
- Библиотека дескрипторов `Displaytag` `http://displaytag.sourceforge.net/`
- Фиксирующий код JUnit `http://junit.sourceforge.net/doc/cookbook/cookbook.htm`
- Hibernate `http://hibernate.org/`
- Проверочная среда JUnit `http://junit.org`
- Web-сайт по рефакторингу Мартина Фаулера `http://refactoring.com/`
- Среда Spring `http://springframework.org/`
- Web-сайт Java Sun `http://java.sun.com/`

# 11

# Что дальше

Неделя 9 — первый выпуск готов!

**Сьюзен:** Итак, парни, я хочу поздравить вас, работа закончена успешно. Это, вероятно, одно из лучших приложений, над которыми я работала. Мне понравился используемый вами способ разработки программного обеспечения: приятен тот факт, что я принимала участие во всем процессе, результат итерации каждые 2 недели также был хорошо заметен! Не могу дождаться, когда представлю это приложение перед пользователями, я думаю, что им оно понравится! Между прочим, на следующей неделе мы с Роном встречаемся, чтобы обсудить второй выпуск, поэтому ожидайте услышать на совещании пожелания пользователя. Перед вами, парни, снова великая задача!

На настоящий момент мы рассмотрели существенный объем материала, вполне достаточный для создания завершенного, надежного и корпоративного приложения Java. Однако по технологии Java доступны сотни книг, прежде всего, потому, что существует большое количество технологий, связанных с Java, а также потому, что многие желают обучиться работе с Java. С учетом этого факта, фактически невозможно изучить все, так что давайте рассмотрим потенциальные области, которые вы могли бы исследовать далее.

# Что рассматривается в этой главе

В этой главе я представлю вам некоторые рекомендации по темам, которые имеет смысл исследовать далее.

- Завершение приложения Time Expression
- Разработка программного обеспечения на базе XP и AMDD
- Платформа Java
- Ant
- JUnit
- Hibernate
- Среда Spring Framework
- SDK Eclipse
- Регистрация, отладка, мониторинг и профилирование
- Получение помощи

# Завершение приложения Time Expression

Если вы читали эту книгу с начала до конца, то, безусловно, заметили, что мы реализовали только часть функциональных возможностей, определенных в наших бизнес-требованиях, изложенных в главе 2, "Простое приложение: сетевая система учета рабочего времени". Пользовательские истории для реализации я выбрал исходя из двух причин.

- Возможность дистанционно вводить часы показалась мне хорошим началом, как нечто реальное, наглядное и реализуемое за пару первых итераций.
- Эти пользовательские истории выглядели привлекательно с точки зрения демонстрации создания экрана как содержащего форму, так и не содержащего ее.

Наилучший способ изучения различных технологий, включая описанные в этой книге, заключается в работе с кодом, поэтому давайте завершим здесь остаток приложения. Мы закончили первые пять пользовательских историй из табл. 2.2, приведенной в главе 2, "Простое приложение: сетевая система учета рабочего времени". Остальные пользовательские истории вы могли бы попробовать реализовать самостоятельно. Если вы решитесь на это и у вас возникнут творческие идеи, я с удовольствием узнал бы о них. (Моя контактная информация доступна на сайте `visualpatterns.com`.)

# Разработка программного обеспечения на базе XP и AMDD

В главах 2, "Простое приложение: сетевая система учета рабочего времени", и 3, "Архитектура и модель проекта на базе XP и AMDD", мы обсуждали методы экстремального программирования (XP) и разработки методом гибкого моделирования (AMDD). Если этот стиль работы подходит для вас, вы могли бы использовать его в ваших проектах. Более подробная информация по этой теме приведена на сайтах `agilemodeling.com` и `extremeprogramming.org`. Я также привел контрольные списки этих методов в разделе приложений.

Еще один Web-сайт по этой теме, который я упомянул ранее, это `refactoring.com`, который содержит обширный каталог методов рефакторинга кода. Кроме того, сайт `agiledata.org` содержит список методов работы с базой данных, включая рефакторинг базы данных и гибкое моделирование данных. Это весьма перспективная область.

# Платформа Java

Технология Java постоянно развивается, вы все чаще видите на Web-сайте корпорации Sun Microsystems (`java.sun.com`) новые выпуски. Я рассмотрел лишь некоторые из наиболее новых средств, однако с учетом того, что Java является обширной платформой, здесь слишком много тем для изучения. Но вы вполне могли бы потратить некоторое время на изучение относительно новых средств и API языка Java.

# Ant

Как я упоминал в прежних главах, Ant — это замечательный инструмент, когда дело доходит до загрузки встроенных задач. Однако существует большое количество внешних задач, доступных в Сети, и вы можете также создавать свои собственные. Я настоятельно рекомендую вам изучить Ant подробнее. Более подробная информация по этой теме приведена на сайте `http://ant.apache.org/`.

Также изучите *непрерывную интеграцию*[1] (continuous integration), сайт `martinfowler.com`. Эта концепция быстро завоевывает популярность, а такие инструментальные средства, как CruiseControl (`cruisecontrol.sourceforge.net`), облегчают ее реализацию.

Весьма известен еще один продукт от Apache — Maven (`maven.apache.org`). Эта среда, автоматизирующая и упрощающая процесс построения, завоевывает в последнее время широкую популярность.

# JUnit

Среда JUnit, с одной стороны, довольно проста, я предоставил вам более чем достаточную информацию для проверки модуля. С другой стороны, на Web-сайте `junit.org` доступно множество других инструментов и утилит, применимых для

[1] Практически одновременно с данной, выходит книга *Непрерывная интеграция* (авторы П. Дюваль, С. Матиас и Э. Гловер) по этой теме. Рекомендую. — *Примеч. ред.*

проверки таких разнообразных элементов, как ложные объекты, JSystem, JUnit PDF Report, и многого другого.

Я также рекомендовал бы вам изучить подходы проектирования методом проверки (TDD), которые подразумевают создание предварительной проверки. Этот метод работы не так прост, но, овладев им, вы, вероятно, захотите использовать только его. Поэтому уделите ему также некоторое время, поскольку сначала может показаться, что вы тратите больше времени, создавая проверки; но если вы учтете то количество времени, которое придется потратить на проверку кода вашего модуля и устранение дефектов, обнаруженных в ходе испытаний или уже пользователями, то поймете, почему этот подход может фактически сэкономить ваше время. Кроме того, эта методика лучше всего работает, если вы пишете маленькие порции кода (например, небольшие методы и классы, в отличие от больших и сложных).

# Hibernate

Мы рассмотрели ту часть базовых функциональных возможностей Hibernate, которой достаточно, чтобы помочь вам создать сложное приложение. Однако, как я неоднократно упоминал ранее, этой теме посвящены целые книги. Кроме того, для устаревших и сложных приложений вы, вероятно, захотите использовать более сложные методы, предоставляемые Hibernate. Например, вам, вероятно, стоит глубже изучить следующие средства Hibernate:

- связывание компонентов;
- двунаправленные ассоциации;
- связывание наследования;
- повышение производительности;
- хранимые процедуры;
- кэширование.

Более подробная информация об этих и других средствах Hibernate приведена на Web-сайте `hibernate.org`.

# Среда Spring Framework

Несмотря на то что среде Spring Framework в этой книге посвящены главы 6, "Обзор среды Spring Framework", 7, "Среда Spring Web MVC Framework", и часть главы 10, "Кроме основ", я, вероятно, рассмотрел только половину. Среда Spring намного больше. Вам, вероятно, стоит глубже изучить следующие средства Spring:

- средства аспект-ориентированного программирования (Aspect-Oriented Programming – AOP);
- поддержка дистанционного доступа для следующих технологий: дистанционный вызов методов Java (Java Remote Method Invocation – RMI), Web-службы (с использованием JAX-RPC), архитектура соединителя Java (Java Connector Architecture – JCA), Enterprise JavaBean (EJB) и служба сообщений Java (Java Message Service – JMS);

- поддержка баз данных, включая поддержку объектов доступа к данным (Data Access Object – DAO), поддержку ORM (для Hibernate, JDO, iBATIS), а также поддержку подключений к базам данных Java (Java Database Connectivity – JDBC);
- поддержка транзакции для API транзакций Java (Java Transaction API – JTA);
- среда Spring Portlet MVC (поддержка для JSR-168 Portlet API);
- интеграция среды Spring Web MVC Framework с технологией композиции сайтов, такой как Tiles от Apache (`struts.apache.org/struts-tiles/`), или технологией художественного оформления, такой как SiteMesh от OpenSymphony (`opensymphony.com`);
- подчиненные проекты Spring (включая Web Flow, защиту Acegi, BeanDoc и поддержку Rich Client);
- новые библиотеки дескрипторов, представляемые Spring;
- другое (поддержка Spring для JEE, например JMS, EJB и т.д.).

# SDK Eclipse

Как я упомянул в главе 8, "Феномен Eclipse!", платформа Eclipse разрастается в потрясающем темпе! Поэтому в этой области очень много тем для изучения. Например, новые или расширенные дополнения от разных производителей появляются буквально каждую неделю. Не ленитесь почаще посещать Web-сайт `eclipse.org`, а также различные каталоги дополнений, если вы ищете определенные дополнения.

Что касается ядра платформы SDK Eclipse (включая входящие в комплект инструментальные средства разработки Java), позвольте порекомендовать темы для дальнейшего исследования:

- стили оформления кода. Эта возможность Eclipse предоставляет обширный набор параметров для настройки способа, которым вы предпочитаете оформлять код. Эти параметры доступны при выборе в меню Windows (Окна) пункта Preferences (Предпочтения);
- советы и приемы. Просматривайте интерактивную справочную систему различных дополнений (например, JDT). Вы найдете длинный список советов и приемов, которые могут сэкономить вам время и расширить опыт пользователя при работе в Eclipse;
- поддержка рефакторинга. Рефакторинг в Eclipse уже весьма надежен, но, вероятно, продолжит улучшаться. Кроме того, эти методы рефакторинга основаны на некоторых концепциях, представленных на сайте `refactoring.com`, который имеет обширный каталог методов рефакторинга, так что Eclipse, вероятно, обогатится поддержкой более новых методов рефакторинга.

Создание дополнений Eclipse – это не ракетостроение, так что вы можете сами опробовать эту возможность. Собственные дополнения предназначаются не только для технических потребностей или потребностей разработки. Лично мне известен проект, связанный с компьютеризацией юриспруденции, который был разработан с использованием стандартного комплекта инструментов управления (Standard Widget Toolkit – SWT) и развернут как дополнение Eclipse.

# Регистрация, отладка, мониторинг и профилирование

Мы рассмотрели методы регистрации, отладки, мониторинга и профилирования в главе 9, "Регистрация, отладка, мониторинг и профилирование". Однако я просто царапнул поверхность материала, который пришлось изучить мне.

Регистрация, по минимуму, может применяться для простой трассировки и отладки или для контрольного журнала системы защиты, располагающегося локально. Создание своих собственных специальных расширений регистрации, использующих регистрацию JDK или log4j, относительно просто, так что вы можете писать классы регистрации для различных типов задач. Регистрация может также осуществляться локально или дистанционно, что открывает некоторые интересные возможности.

Отладка – это своего рода искусство, и не удивительно, что разработчики предпочитают подходить к ней различными способами. В то время как некоторые разработчики отлаживают код, используя операторы `print`, другие наслаждаются отладчиком GUI. Если вам нравится отладка GUI, исследуйте отладчик Eclipse далее. Мы рассмотрели много базового материала в главе 8, "Феномен Eclipse!", что вполне достаточно для самостоятельной отладки кода в Eclipse. Однако когда вы начнете работать с отладчиком Eclipse (если еще не работаете), то, вероятно, найдете индивидуальные способы отладки с использованием выражений отслеживания, условных контрольных точек и т.д.

Мы рассмотрели чрезвычайно простые методы мониторинга в главах 9, "Регистрация, отладка, мониторинг и профилирование", и 10, "Кроме основ". Технология расширений управления Java (Java Management Extensions – JMX) – это целый собственный мир, заслуживающий глубокого изучения. Более подробная информация об этой технологии приведена на сайте `java.sun.com`. Профилирование приложений Java я затронул в главе 9, "Регистрация, отладка, мониторинг и профилирование". Профилирование требует более глубокого исследования, если ваше приложение нуждается в настройке производительности. Например, IDE NetBeans от Sun имеет замечательный профайлер, а SDK Eclipse обладает профайлером в виде дополнения. Как я упомянул в главе 9, "Регистрация, отладка, мониторинг и профилирование", множество профайлеров Java с открытым исходным кодом также заслуживают внимания.

# Получение помощи

При работе с любой технологией вы неизменно столкнетесь с техническими трудностями. Это, безусловно, расстраивает, но любая помощь, которую вы можете получить, приветствуется при поиске ошибок и решении проблем. Ниже приведено несколько рекомендаций относительно того, как получить помощь по технологиям, которые мы рассмотрели.

## Обсуждения в сетевых форумах

Лично мне обсуждения в сетевых форумах по среде Spring Framework (например, `forum.springframework.org`) оказали существенную помощь, когда я завяз с практическими примерами и нуждался в помощи. Фактически, на вопросы в этих форумах зачастую отвечают ведущие разработчики Spring. На мои вопросы отвечал даже лично Род Джонсон (основатель Spring)!

Форумы обсуждения Hibernate (`forum.hibernate.org`) тоже весьма полезны, причем я обнаружил, что время от времени вы можете получить краткие ответы от основателей Hibernate. (Мне жаль, что это не произошло со мной.) Однако есть много других пользователей Hibernate, желающих помочь вам.

Кроме того, для Eclipse есть открытые группы новостей, доступные при помощи Web-сайта `eclipse.org`.

### Документация Java и исходный код

Хотя и Spring, и Hibernate снабжены хорошей справочной документацией, иногда необходима дополнительная информация об этих средах. В конце предыдущей главы я подчеркнул, что документирование Javadoc очень важно, поскольку код — это один из артефактов, который всегда под рукой; с точки зрения тех, кто с ним обычно работает. Среды Spring Framework и Hibernate придерживаются этой логики, следовательно, вы найдете, что их документация Javadocs обширнее, чем вы могли ожидать. Если вы не можете найти информацию в справочной документации, попробуйте поискать ее в документации Javadocs для этих API.

Еще один дополнительный источник документации, хотя не оптимальный, — это просмотр исходного кода технологий с открытым исходным кодом, таких как Spring и Hibernate. Для большинства технологий с открытым исходным кодом исходный код либо вводится в комплект поставки, либо предоставляется для загрузки отдельно. Он может также сопровождать инструментальные средства, такие как Eclipse, что позволяет вам обратиться к папке исходного кода, когда вы отлаживаете ваш код. Отладка — не единственная причина, по которой вы иногда могли бы захотеть обратиться к их исходному коду в Eclipse; вы могли бы захотеть просмотреть исходный код этих API, чтобы понять, как они работают внутри.

## Краткое замечание об инструментах "качества" кода

Еще одна область, в которой лично я не работал интенсивно, но, возможно, начну изучать, — это так называемые *инструменты качества кода* (code quality tool) или *статические инструменты анализа кода* (static code analysis). Эти инструменты помогают проверить качество вашего кода: например, соответствие кода стандартам, придельному коду, покрытие кода проверками модуля и т.д. К некоторым из инструментов этой области относятся следующие:

- Checkstyle (`checkstyle.sourceforge.net`)
- Clover (`www.cenqua.com`)
- Cobertura (`cobertura.sourceforge.net`)
- PMD (`pmd.sourceforge.net`)

## Резюме

В этой главе мы рассмотрели следующие темы, подлежащие дальнейшему изучению.

- Завершение приложения Time Expression
- Разработка программного обеспечения на базе XP и AMDD

- Платформа Java
- Ant
- JUnit
- Hibernate
- Среда Spring Framework
- SDK Eclipse
- Регистрация, отладка, мониторинг и профилирование
- Получение помощи

Книга практически закончена! Не забудьте ознакомиться с разделом приложений, который содержит примеры рефакторинга кода, контрольные списки, перечни замечательных инструментов и т.д.!

# Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, содержится на следующих Web-сайтах:

- Agile Data `http://www.agiledata.org/`
- Agile Modeling `http://www.agilemodeling.com`
- Ant `http://ant.apache.org/`
- Apache Tiles `http://struts.apache.org/struts-tiles/`
- Checkstyle `checkstyle.sourceforge.net`
- Clover `www.cenqua.com`
- Непрерывная интеграция `http://www.martinfowler.com/articles/continuousIntegration.html`
- Cobertura `cobertura.sourceforge.net`
- SDK Eclipse `http://eclipse.org`
- Экстремальное программирование `http://extremeprogramming.org`
- Hibernate Framework `http://hibernate.org`
- Форумы обсуждения Hibernate `http://forum.hibernate.org/`
- Механизм баз данных HSQLDB `http://hsqldb.org/`
- Концентратор для ресурсов Spring Framework `http://www.springhub.com/`
- Технология Java `http://java.sun.com`
- JUnit `http://junit.org`
- Maven `http://maven.apache.org/`
- OpenSymphony Sitemes `http://www.opensymphony.com/sitemesh/`
- PMD `http://pmd.sourceforge.net/`
- Spring Framework `http://springframework.org`
- Форумы обсуждения Spring `http://forum.springframework.org/`
- NetBeans IDE `http://netbeans.org`
- Визуальные схемы `http://visualpatterns.com`

# 12

# Некоторые соображения

Неделя 10 — последующий выпуск 1. Совещание

**Сьюзен:** Добро пожаловать всем в этот учебный класс. Полагаю, вы все найдете, что это приложение хорошо разработано и обладает необходимыми возможностями, не более, не менее.

Итак, позвольте мне начать с краткого обзора ... 

Я надеюсь, что эта книга понравилась вам и оказалась полезной. С учетом небольшого размера, я не мог рассмотреть здесь все, что хотел. Однако мое намерение заключалось в том, чтобы предоставить вам достаточно информации для создания полнофункционального приложения наряду с хорошим знанием дополнительных возможностей.

**1 400 часов за 4 месяца!**

На эту книгу ушло много крови и пота (моего и тех, кто мне помогал). Вас мог бы заинтересовать тот факт, что я писал эту книгу приблизительно 1 400 часов в течение 4 месяцев! (Никакого преувеличения. Я трудился в среднем по 14–15 часов в день без выходных.) Это намного больше, чем при нормальной работе в течение 9 месяцев. Это было сумасшедшее расписание, но мы сделали это, чтобы выпустить книгу к конференции JavaOne 2006 (Центр Moscone, Сан-Франциско, Калифорния).

# Мои ближайшие планы на будущее

Данная книга — и счастливое, и грустное событие в моей жизни, поскольку, вероятно, это моя последняя существенная публикация о Java после десятилетия описания этой замечательной технологии (28 статей и 2 главы в книге Java). Хотя я планирую продолжать консультировать по технологиям, близким к Java, все же основное внимание буду уделять методам гибкого моделирования. Я также надеюсь на сотрудничество в специальных проектах с другими специалистами, особенно с некоторыми творческими людьми, которых я имею удовольствие знать.

В ближайшее время я планирую исследовать визуальные способы улучшения использования методологий программного обеспечения и различных схематических методов. Я убежден, что существуют лучшие способы моделирования, чем UML. Я также убежден, что существуют визуальные способы обучения разработчиков программного обеспечения, основам или некой структуре знаний, которая сделает менеджеров и управляющих счастливыми и позволит заниматься тем, что нам нравится больше всего, кодом! Я надеюсь представить этот сбалансированный подход в ближайшем будущем.

**Примечание**

В ближайшее время я планирую создать глобальное виртуальное сообщество, и этот процесс уже начался. Нашей задачей будет исследование улучшения методов моделирования и разработки уникальных визуальных процессов (более подробная информация по этой теме приведена в приложении И, "Исследование визуальных шаблонов"). Если вы хотите присоединиться к этому виртуальному сообществу, свяжитесь со мной через `visual-patterns.com`.

# Будущие гибких методов технологии Java

Ныне я охватил методы экстремального программирования (XP) и гибкого моделирования (AM) (как вы, вероятно, заметили в этой книге). После многих лет использования больших предварительных требований (Big Requirements Up Front — BRUF) и большого предварительного проектирования (Big Design Up Front — BDUF) я наслаждаюсь этим новым и естественным стилем работы, вероятно, потому, что он помогает мне больше не чувствовать себя виноватым за недостаточное документирование или срыв сроков выполнения проекта!

Что касается Java, это процветающая и все еще доминирующая технология! Я определенно предвижу эту тенденцию, по крайней мере, еще на несколько лет, поскольку Java выполняется на всем, от мобильного телефона и электронных устройств на рабочем столе до малых и больших серверов (с длинным списком поддерживаемых операционных систем). Помните, Java – это платформа, а не просто язык!

# Всего наилучшего!

И наконец, я надеюсь, что вы находите мои личные мнения терпимыми, изложенный в этой книге материал достаточно ценным, а продемонстрированные методы применимыми. Если эти темы вас заинтересовали, посетите мой сайт `visualpatterns.com`. Еще раз, от всего сердца благодарю вас за чтение этой книги. Всего наилучшего!

– Анил Хемраджани
2006

# IV

# Приложения

# A

# Загружаемый код этой книги

Этот раздел содержит частичный список файлов, находящихся в загружаемом файле архива кода этой книги, `bookcode.zip`, доступном на Web-сайте книги. В книге неоднократно встречаются ссылки на эти файлы (например `.java` и `.xml`).

> **Примечание**
> Файл `bookcode.zip` содержит внутри три каталога проектов: `timex/`, `timex2/` и `springhibernate/`; они подробно описаны в этом приложении.

Остальная часть этого приложения рассматривает список файлов, находящихся в файле `bookcode.zip`. В корневом каталоге, `rapidjava`, вы найдете другие каталоги, а также файл `rapidjava/README.txt`, который предоставляет дальнейшие инструкции по трем Web-приложениям.

> **Примечание**
> Полный код примеров, описанных в этой книге, доступен на Web-сайте издательства Sams. Для удобства доступа к странице книги, зарегистрируйтесь по адресу `www.samspublishing.com/register`, введите ISBN этой книги (без дефисов), а затем щелкните на кнопке SUBMIT. Когда отобразится заголовок книги, щелкните на нем, чтобы перейти к странице, где вы сможете разгрузить код.

## Совместно используемый каталог библиотек сторонних производителей

В этом приложении описаны файлы JAR стороннего производителя, используемые тремя Web-приложениями. Они загружаются сценариями Ant `build.xml` для каждого из этих трех проектов и связываются с их соответствующим развертываемым файлом WAR.

> **Примечание**
> Обычно я располагаю каталог `lib/` внутри каталога определенного проекта (например, `rapidjava/timex/lib/`); но чтобы уменьшить размер файла `bookcode.zip`, содержащего три проекта, я совместно использую каталог `lib/` для них, поскольку для каждого из этих проектов обязателен практически тот же набор внешних файлов JAR.

```
rapidjava/lib/activation.jar
rapidjava/lib/antlr-2.7.6rc1.jar
rapidjava/lib/asm-attrs.jar
rapidjava/lib/asm.jar
rapidjava/lib/cglib-2.1.3.jar
```

```
rapidjava/lib/commons-beanutils-1.7.0.jar
rapidjava/lib/commons-collections-2.1.1.jar
rapidjava/lib/commons-lang-2.1.jar
rapidjava/lib/commons-logging-1.0.4.jar
rapidjava/lib/displaytag-1.1.jar
rapidjava/lib/displaytag-export-poi-1.1.jar
rapidjava/lib/dom4j-1.6.1.jar
rapidjava/lib/ehcache-1.1.jar
rapidjava/lib/hibernate3.jar
rapidjava/lib/hsqldb.jar
rapidjava/lib/itext-1.3.jar
rapidjava/lib/javax.servlet.jar
rapidjava/lib/jstl.jar
rapidjava/lib/jta.jar
rapidjava/lib/junit.jar
rapidjava/lib/log4j-1.2.11.jar
rapidjava/lib/mail.jar
rapidjava/lib/quartz-1.5.1.jar
rapidjava/lib/spring-hibernate3.jar
rapidjava/lib/spring-mock.jar
rapidjava/lib/spring.jar
rapidjava/lib/standard.jar
```

# Каталог примера приложения (`rapidjava/timex/`)

Приведенные ниже файлы относятся к демонстрационному приложению Time Expression, представленному в главе 2, "Простое приложение: сетевая система учета рабочего времени". Эти файлы находится в каталоге `rapidjava/timex/`.

## Файлы Ant

```
antextras.xml
build.xml
local.properties
timexhsqldb.xml
```

## Файлы базы данных HSQLDB

```
data/timexdb.lck
data/timexdb.log
data/timexdb.properties
data/timexdb.script
```

## Файлы конфигурации и Java

```
src/conf/log4j.properties
src/conf/messages.properties
src/conf/springtest-applicationcontext.xml
src/conf/timex-servlet.xml
src/conf/timex.tld
src/conf/web.xml
src/java/com/visualpatterns/timex/controller/EnterHoursControl
↳ler.java
```

```
src/java/com/visualpatterns/timex/controller/EnterHoursValidator.java
src/java/com/visualpatterns/timex/controller/HttpRequestIntercep
↳tor.java
src/java/com/visualpatterns/timex/controller/MinutesPropertyEdit
↳or.java
src/java/com/visualpatterns/timex/controller/SignInController.java
src/java/com/visualpatterns/timex/controller/SignInValidator.java
src/java/com/visualpatterns/timex/controller/SignOutController.java
src/java/com/visualpatterns/timex/controller/TimesheetListControl
↳ler.java
src/java/com/visualpatterns/timex/job/ReminderEmail.java
src/java/com/visualpatterns/timex/model/AuditInterceptor.java
src/java/com/visualpatterns/timex/model/Department.hbm.xml
src/java/com/visualpatterns/timex/model/Department.java
src/java/com/visualpatterns/timex/model/DepartmentManager.java
src/java/com/visualpatterns/timex/model/Employee.hbm.xml
src/java/com/visualpatterns/timex/model/Employee.java
src/java/com/visualpatterns/timex/model/EmployeeManager.java
src/java/com/visualpatterns/timex/model/hibernate.cfg.xml
src/java/com/visualpatterns/timex/model/Timesheet.hbm.xml
src/java/com/visualpatterns/timex/model/Timesheet.java
src/java/com/visualpatterns/timex/model/TimesheetManager.java
src/java/com/visualpatterns/timex/test/AllTests.java
src/java/com/visualpatterns/timex/test/DemoNewJavaFeatures.java
src/java/com/visualpatterns/timex/test/HibernateTest.java
src/java/com/visualpatterns/timex/test/SimpleTest.java
src/java/com/visualpatterns/timex/test/SpringTest.java
src/java/com/visualpatterns/timex/test/SpringTestMessage.java
src/java/com/visualpatterns/timex/test/TimesheetListControl
↳lerTest.java
src/java/com/visualpatterns/timex/test/TimesheetManagerExtras.java
src/java/com/visualpatterns/timex/test/TimesheetManagerTest.java
src/java/com/visualpatterns/timex/util/ApplicationSecurityMan
↳ager.java
src/java/com/visualpatterns/timex/util/DateUtil.java
src/java/com/visualpatterns/timex/util/HibernateUtil.java
src/java/com/visualpatterns/timex/util/PayPeriodCheckTag.java
src/java/com/visualpatterns/timex/util/TimexJmxBean.java
src/java/com/visualpatterns/timex/view/enterhours.jsp
src/java/com/visualpatterns/timex/view/signin.jsp
src/java/com/visualpatterns/timex/view/timesheetlist.jsp
src/web/index.jsp
src/web/rapidjava-ajax.html
src/web/rapidjava-ajax.jsp
src/web/includes/timex.css
```

# Каталог примера приложения после рефакторинга (`rapidjava/timex2/`)

Следующие файлы связаны с материалом главы 10, "Кроме основ". Эти файлы находится в каталоге `rapidjava/timex2/`.

В этом списке и списках, расположенных ниже, приведены только те файлы, которые были изменены в результате рефакторинга исходного примера приложения (более подробная информация по этой теме приведена в приложении Б,

"Рефакторинг типового приложения"); другие файлы каталога `timex2/` подобны файлам каталога `timex/`.

```
antextras.xml
build.xml
local.properties
timexhsqldb.xml
src/conf/timex2-servlet.xml
src/java/com/visualpatterns/timex/model/DepartmentManager.java
src/java/com/visualpatterns/timex/model/EmployeeManager.java
src/java/com/visualpatterns/timex/model/hibernate.cfg.xml
src/java/com/visualpatterns/timex/model/TimesheetManager.java
src/java/com/visualpatterns/timex/test/TimesheetManagerTest.java
src/java/com/visualpatterns/timex/view/dberror.jsp
src/java/com/visualpatterns/timex/view/enterhours.jsp
src/java/com/visualpatterns/timex/view/includemessages.jsp
src/java/com/visualpatterns/timex/view/signin.jsp
src/java/com/visualpatterns/timex/view/timesheetlist.jsp
```

# Каталог примера интеграции Spring и Hibernate (`rapidjava/springhibernate/`)

Следующие файлы связаны с разделом, посвященным интеграции Spring и Hibernate, главы 10, "Кроме основ". Они демонстрируют два стиля использования поддержки Spring функций API для Hibernate (например, поддержка Spring декларативного управления транзакциями).

```
conf/springhibernate-servlet.xml
conf/web.xml
build.xml
src/controller/TimesheetSaveController.java
src/model/Timesheet.hbm.xml
src/model/Timesheet.java
src/model/TimesheetManager.java
src/model/TimesheetManagerImpl1.java
src/model/TimesheetManagerImpl2.java
src/view/dberror.jsp
src/view/timesheetsave.jsp
```

# Б

# Рефакторинг типового приложения

Ниже приведен код конфигурации и отрывки сценариев, демонстрирующие несколько простых примеров рефакторинга исходного приложения этой книги, по мере внесения изменений.

**Примечание**
Архивный файл кода книги содержит два каталога проектов нашего типового приложения: `timex/` и `timex2/`. Последний содержит обсуждаемый далее код после рефакторинга. Поскольку вы имеете обе версии кода, можете сравнить их и убедиться, как рефакторинг работает в реальном мире. Однако не забывайте, что сначала необходимо получить полностью работоспособный код, а затем оптимизировать его. Такова натура рефакторинга, непрерывное проектирование и перепроектирование, улучшение кода по мере необходимости, постоянные попытки совершенствования. Более подробная информация о концепции и типах рефакторинга приведена на сайте `refactoring.com`.

## Файл `SignInController.java`: мониторинг JMX

Следующие строки кода были добавлены для мониторинга JMX:

```
import com.visualpatterns.timex.util.TimexJmxBean;
    private TimexJmxBean timexJmxBean;
...
    timexJmxBean.setSignInCount(timexJmxBean.getSignInCount() + 1);
...
    public TimexJmxBean getTimexJmxBean()
    {
        return timexJmxBean;
    }
public void setTimexJmxBean(TimexJmxBean timexJmxBean)
    {
        this.timexJmxBean = timexJmxBean;
    }
```

# Файл TimesheetListController.java: мониторинг JMX

Следующие строки кода были добавлены для мониторинга JMX:

```
import com.visualpatterns.timex.util.TimexJmxBean;
...
private TimexJmxBean timexJmxBean;
...
    timexJmxBean.setTimesheetsFetched(timexJmxBean.getTimeshee
↳tsFetched()
        + timesheets.size());
...
public TimexJmxBean getTimexJmxBean()
{
    return timexJmxBean;
}

public void setTimexJmxBean(TimexJmxBean timexJmxBean)
{
    this.timexJmxBean = timexJmxBean;
}
```

# Управляющие классы: интеграция Spring и Hibernate

В главе 10, "Кроме основ", мы обсуждали интеграцию Hibernate и Spring. Это позволяет значительно уменьшить количество строк кода в нашем управляющем классе. Код после рефакторинга находится в каталоге `rapidjava/timex2/`.

Например, ниже приведен первоначальный метод `saveTimesheet` нашего класса `TimesheetManager` (расположен в каталоге `timex/`):

```
public void saveTimesheet(Timesheet timesheet)
{
    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session.beginTransaction();
    try
    {
        session.saveOrUpdate(timesheet);
        session.getTransaction().commit();
    }
    catch (HibernateException e)
{
        session.getTransaction().rollback();
        throw e;
    }
}
```

Весь этот код был уменьшен до одной строки! Новый метод `saveTimesheet` находится в каталоге `timex2/` и выглядит следующим образом:

```
public void saveTimesheet(Timesheet timesheet)
{
    getHibernateTemplate().merge(timesheet);
}
```

## Файл `timesheetlist.jsp`: переход на подключаемый файл и библиотеку `Displaytag`

Чтобы централизовать код JSP, который отображает сообщения об ошибке и состоянии, мы переместили его в файл `includemessage.jsp`:

```
<%@ taglib prefix="display" uri="http://displaytag.sf.net/el" %>
...
<%@ include file="/WEB-INF/jsp/includemessages.jsp" %>
...
```

Кроме того, чтобы получить возможность сортировки в пользовательском Web-интерфейсе, мы перешли с библиотеки JSTL `C:forEach` на библиотеку `Displaytag`. Это не только демонстрирует рефакторинг, но и преимущества шаблона проектирования модель-представление-контроллер (Model-View-Controller — MVC), поскольку код представления изменился без всякого воздействия на код модели:

```
<display:table name="timesheets" id="timesheet" defaultsort="1"
    requestURI="timesheetlist.htm"
    cellpadding="5" cellspacing="0"
    export="false" class="tableborder">
<display:column sortable="true" title="Period Ending"
    href="enterhours.htm"
    sortProperty="periodEndingDate "
    paramId="tid" paramProperty="timesheetId"
    class="tdcenter">
    <fmt:formatDate value="${timesheet.periodEndingDate}"
    type="date" pattern="MM/dd/yyyy"/>
  </display:column>
  <display:column sortable="true" title="Hours "
    sortProperty="totalMinutes" class="tdright">
    <fmt:formatNumber value="${timesheet.totalMinutes / 60.0} "
    pattern="0.00"/>
  </display:column>
...
```

## Файл `enterhours.jsp`: переход на подключаемый файл и библиотеку дескрипторов `timex`

Для проверки текущего времени добавим библиотеку дескрипторов `timex`. Кроме того, переместим код сообщения об ошибках и состоянии в файл `includemessage.jsp`.

```
<%@ taglib prefix="timex" uri="/WEB-INF/timex.tld" %>
<timex:periodcheck checkDate="${command.periodEndingDate}" >
<input name="save" type="submit" value="Save">
</timex:periodcheck>
...
<%@ include file="/WEB-INF/jsp/includemessages.jsp" %>
```

## Классы *Test и TimexTestCase

Большинство наших прежних классов *Test выглядит наподобие следующего фрагмента кода из нашего исходного класса TimesheetManagerTest:

```
public class TimesheetManagerTest extends TestCase
{
    TimesheetManager timesheetManager = new TimesheetManager();
    public static void main(String args[])
```

Теперь, вместо того чтобы расширять непосредственно класс JUnit TestCase, мы дополним класс TimexTestCase (находится в каталоге timex2/), как показано далее:

```
public class TimesheetManagerTest extends TimexTestCase
{
    public static void main(String args[])
```

Преимущество этого подхода заключается в возможности переместить весь фиксирующий код в родительский класс (например, TimexTestCase), так что производные проверочные подклассы могут сосредоточиваться на проверках модулей, а не фиксирующего кода (более подробная информация о фиксирующем коде приведена на сайте junit.org).

## Файл DateUtil.java: новый метод

Добавленный новый метод для библиотеки дескрипторов timex.

```
public static boolean isInCurrentPayPeriod(Date checkDate)
{
    Date weekStartDate = getDateWithZeroTime(getCurrentPeriodStar
↳tingDate());
    Date weekEndDate = getDateWithMaxTime(getCurrentPeriodEn
↳dingDate());

    return (!checkDate.before(weekStartDate) && !checkDate
        .after(weekEndDate));
}
```

## Файл timex.css: новые стили

Добавлено несколько новых стилей.

```
thead { background-color: #D0D6EA;}
.tableborder { border: thin; }
.tdright { text-align: right; }
.tdcenter { text-align: center;}
.even { background-color: #F1F8FE}
```

# Файл `timexhsqldb.xml`: исправление дефекта данных

Исправлены пароли в операторе SQL `INSERTS`, поскольку обнаружился дефект в проверочных данных (не в коде). В результате приемочных испытаний кода главы 3, "Архитектура и модель проекта на базе XP и AMDD", оказалось, что "пароль должен содержать от 8 до 10 символов". Первоначальные пароли имели меньше 8 символов, таким образом, пользователь не может зарегистрироваться.

```
INSERT INTO Employee (employeeId, name, employeeCode,
                      password, email, managerEmployeeId)
    VALUES (2, 'Ajay Kumar', 'H', 'visualpatterns',
↳'akumar@acme.com', 3);
INSERT INTO Employee (employeeId, name, employeeCode,
                      password, email, managerEmployeeId)
    VALUES (3, 'Teresa Walker', 'M', 'agilestuff',
↳'twalker@acme.com', 4);
INSERT INTO Employee (employeeId, name, employeeCode,
                      password, email)
    VALUES (4, 'Tom Brady', 'E', 'superbowl', 'tbrady@acme.com');
```

Следует заметить, что дефекты не всегда связаны с кодом; они могут быть также вызваны некорректными данными.

# B

# Соглашения для кода Java

Ниже перечислены некоторые принципы, соответствующие соглашению для языка программирования Java, предписанному на Web-сайте Sun `java.sun.com`, которые я использую в своей работе. (*Примечание:* корпорация Sun рекомендует существенно больше соглашений, поэтому посетите этот Web-сайт.)

- Все файлы исходного кода должны иметь вначале комментарий Javadoc.
- Первой строкой в файле исходного кода является оператор `package`, сопровождаемый операторами `import`.
- Имена пакетов должны начинаться с имени верховного домена, причем написанного со строчной буквы (например, `com.` или `edu.`).
- Имена классов и интерфейсов должны быть именами существительными и использовать смешанный регистр, т.е. прописные буквы для разделения слов (например, `EmployeeHours`).
- Все файлы классов должны иметь следующие элементы, причем в порядке, представленном ниже:
  - документация Javadoc для класса;
  - список переменных в следующем порядке: статические переменные (`static`), переменные экземпляра (открытые (`public`), защищенные (`protected`), без указания степени доступа, закрытые (`private`));
  - список методов в следующем порядке: сначала конструкторы, а затем методы (методы должны быть сгруппированы по функциональным возможностям, а не по области видимости).
- Имена методов должны быть глаголами и использовать смешанный регистр, т.е. прописные буквы для разделения слов, за исключением первого символа; он должен быть в нижнем регистре (например, `getHoursWorked`).
- Имена переменных должны быть глаголами и использовать смешанный регистр, т.е. прописные буквы для разделения слов, за исключением первого символа; он должен быть в нижнем регистре (например, `hoursWorked`). Односимвольных переменных (например, `i`, `j` или `k`) следует избегать, они могут использоваться только для временных переменных (например, в операторе `for`).
- Старайтесь делать все переменные класса закрытыми, т.е. доступными только при помощи соответствующих методов.

- Имена констант должны быть полностью набраны в верхнем регистре, при разделении слов символами подчеркивания (например, `MAX_WORK_HOURS`).
- Старайтесь использовать числовые значения как константы (например, `int MAX_WORK_HOURS=24;`).
- Старайтесь инициализировать локальные переменные там, где они объявлены.
- Избегайте строк более 80 символов.
- Каждая строка должна содержать только один оператор.
- Фигурные скобки для операторов `if-else`, `for`, `while`, `do` и `switch` должны использоваться всегда.

# Г

# Защита Web-приложений

Ниже приведено несколько правил защиты Web-приложения. Более подробная информация по этой теме приведена на Web-сайте `owasp.org`.

- Проверьте правильность ввода браузера (параметры, специальные символы, встроенный SQL) на стороне сервера, а не только на стороне клиента (т.е. JavaScript). Если вы работаете непосредственно с JDBC, старайтесь использовать оператор `java.sql.PreparedStatement`, а не `java.sql.Statement`.
- Не используйте в Web-коде оболочки (`Runtime.exec`); это практически открытое приглашение для хакеров.
- Не храните важные данные где попало (базы данных, файлы и т.д.). Если вам абсолютно необходимо хранить эту информацию, храните ее в зашифрованном виде.
- Не предоставляйте непосредственного доступа ни к каким системным ресурсам, например файлам, базам данных, классам или программам. Выключите просмотр каталога на всех Web-серверах. Не используйте реальные имена файлов и каталогов (например, скрывайте файлы JSP под WEB-INF).
- Для важных данных, таких как имя пользователя, пароль, финансовые данные, информация о здоровье и секретная правительственная информация, используйте протокол HTTPS, а не HTTP.
- Обязательно требуйте от пользователя введения идентификатора и пароля (пароль должен содержать, как минимум, от шести до восьми символов, включая специальные).
- Скрытые поля HTML не так уж надежно скрыты; любой желающий может просмотреть код HTML в браузере, так что учитывайте это.
- Отключайте учетную запись (временно или постоянно) после трех неудачных попыток входа.
- Не храните пароли в виде открытого текста (например, идентификатор и пароль приложения в файле конфигурации).
- Регистрируйте всю или только подозрительную деятельность.
- Используйте промышленные стандарты, хорошо проверенные протоколы безопасности, а не собственные, самодельные решения.

- Метод POST скрывает важные данные надежнее метода GET (с учетом панели адресов браузера и журнала доступа). Рекомендация: проверяйте защиту при помощи расширения Firefox Tamper Data.
- Устраивайте демонстрации исходного кода. Ваши коллеги могли бы заметить нечто, что вы упустили.
- Опасайтесь *межсайтового скриптинга* (Cross-Site Scripting – XSS); хакер может использовать эту методику для похищения персональной информации ваших пользователей.

И наконец, но не в последнюю очередь, будьте параноиком! На самом деле, существует множество людей, пытающихся подобрать пароли, вскрыть сайт и так далее; всегда оставайтесь бдительным! Думайте, как хакер, подразумевая, что хакер знает не меньше вас, а то и больше, поэтому регулярно проверяйте защиту. Помните, вы не можете полностью избегать угроз безопасности; однако вы можете контролировать и пресекать их. Важнее всего то, что существуют автоматизированные агенты поисковых серверов, ищущие лазейки в защите. Когда брешь в защите обнаруживается, в дело вступает уже человеческий фактор; результат может быть убийственным.

# Д

# Типовой контрольный список процесса разработки

Ниже приведен пример простого контрольного списка процесса разработки. Более подробная информация по этой теме приведена в главах 2, "Простое приложение: сетевая система учета рабочего времени", и 3, "Архитектура и модель проекта на базе XP и AMDD", а также на Web-сайтах `extremeprogramming.org` и `agilemodeling.com`.

## Начало проекта

- Неофициальные деловые обсуждения задач и проблем.
- Запуск проекта.
- Формулировка проблемных задач (например, идеальные прецеденты или утверждения).

## Фаза исследования

- Исследование концепций домена (разработка доменной модели).
- Разработка базовых прототипов и последовательности экранов (для приложений с пользовательским интерфейсом).
- Определение контекста (что включать, что отложить на следующий выпуск).
- Определение пользовательских историй для следующего выпуска.
- Создание неофициальной общей доски объявлений для архитектуры и т.д.

## Планирование

- Разработка плана следующего выпуска (версии) системы.
- Определение глоссария общих терминов.
- Разработка плана для следующей итерации.
- Определение системного соглашения (именование, проверка и интеграция программы, а также многое другое).

## Последовательное создание программного обеспечения в итерациях

- Разрабатывайте программное обеспечение последовательно, используя двухнедельные итерации; причем итерацию 0 (или цикл 0) используйте для установки среды и проверки концепции.
- Перед началом каждой итерации созывайте совещание, чтобы выбрать те пользовательские истории, которые будут реализованы.
- Снабдите разработчиков пожеланиями, на основании выбранных статей для итерации.
- Пользователи осуществляют приемочные испытания, как оговорено в требованиях; разработчики реализуют их как проверки модулей.
- Обеспечьте пользователям доступ к разработчикам во время проектирования и разработки.

Развертывайте готовый, работоспособный код каждые две недели, после того как он пройдет приемочные испытания пользователя.

# E

# Значение, действия и принципы гибкого моделирования

Исходная информация находится на странице: `www.agilemodeling.com`

**Значение**

Связь, простота, обратная связь, мужество и смирение.

| Практики | Принципы |
|---|---|
| ОСНОВНЫЕ ПРАКТИКИ: | ОСНОВНЫЕ ПРИНЦИПЫ: |
| Активное участие заинтересованных лиц | Модель должна иметь цель |
| Моделируйте с участием посторонних | Расширяйте участие заинтересованных лиц |
| Применяйте корректные артефакты | Путешествуйте налегке |
| Последовательно переходите к следующему артефакту | Множество моделей |
| Докажите это в коде | Быстрая обратная связь |
| Используйте самые простые инструменты | Помните о простоте |
| Моделируйте в малых приращениях | Окружающие изменения |
| Единый источник информации | Инкрементные изменения |
| Коллективная собственность | Качество работы |
| Создавайте по нескольку моделей параллельно | Ваша первая цель — программное обеспечение |
| Создавайте простое содержимое | Ваша вторая цель — обеспечить следующее усилие |
| Описывайте модели просто | |
| Обсуждайте модель публично | |
| ДОПОЛНИТЕЛЬНЫЕ ПРАКТИКИ: | ДОПОЛНИТЕЛЬНЫЕ ПРИНЦИПЫ: |
| Применяйте стандарты моделирования | Содержимое важнее внешнего вида |

*Окончание таблицы*

| Практики | Принципы |
|---|---|
| Примените шаблоны своевременно | Открытое и честное общение |
| Откажитесь от временных моделей | |
| Формализуйте соглашения модели | |
| Модифицируйте только при необходимости | |
| ДЕЙСТВИТЕЛЬНО ХОРОШИЕ ИДЕИ: | |
| Рефакторинг | |
| Разработка с предварительной проверкой | |

# Ж

# Контрольный список экстремального программирования

Исходная информация содержится на странице: `extremeprogramming.org`

## Обзор

- Клиент перечисляет возможности, которыми должно обладать программное обеспечение.
- Программисты разделяют возможности на отдельные задачи и оценивают количество труда, необходимое для выполнения каждой задачи.
- Клиент выбирает наиболее важные задачи, которые могут быть выполнены к следующему выпуску.
- Программисты выбирают задачи и работают парами.
- Программисты создают проверки модулей.
- Программисты добавляют средства для проверки модуля.
- Программисты исправляют средства и проверяют их по мере необходимости.
- Программисты интегрируют код.
- Программисты готовят версию выпуска.
- Клиент проводит приемочные испытания.
- Версия принимается в работу.
- Программисты уточняют свои оценки трудозатрат на основании реального объема работ, выполненного в ходе цикла выпуска.

**Правила и действия**

| Планирование | Программирование |
|---|---|
| Запись пользовательских историй | Клиент всегда доступен |
| Создание расписания в ходе планирования выпуска | Код следует писать в соответствии с принятыми стандартами |
| Использование частых кратковременных выпусков | Программируйте с предварительной проверкой модуля |
| Контроль скорости выполнения проекта | Программируйте парами |
| Проект разделен на итерации | Интегрируется код только одной пары за раз |
| Итерация начинается с планирования | Интегрируйте чаще |
| Перемещайте людей по кругу | Используйте коллективную собственность на код |
| Установочное совещание в начале каждого дня | Оставьте оптимизацию до последнего момента |
| Устраняйте задержки при их возникновении | Никаких сверхурочных |
| **Проектирование** | **Проверка** |
| Простота | Весь код должен иметь проверки модулей |
| Выберите метафору системы | Весь код должен пройти все проверки модулей, прежде чем он может быть выпущен |
| Для сеансов проектирования используйте карточки CRC | При обнаружении ошибки создаются новые проверки |
| Коллективные решения снижают риск | Приемочные испытания выполняются часто, и результат публикуется |
| Не добавляйте функциональные возможности слишком рано | |
| Осуществляйте рефакторинг при любой возможности | |

# 3

# Замечательные инструменты

Увсех нас есть любимые инструменты и утилиты, которые нам нравится использовать для *ускорения* работы. Ниже перечислены те из них, которые я либо использовал лично, либо мне их рекомендовали друзья и коллеги. Все они были бесплатны (на момент написания этой книги); если нет, то это указано здесь специально. Безусловно, все продукты, описанные в этой книге, (например, Eclipse) также заслуживают звания замечательных инструментов.

Подобно всему бесплатному программному обеспечению, вы используете приведенные ниже на свой страх и риск!

## Инструменты, не зависящие от платформы

- 7-Zip. Файловый архиватор с весьма высоким коэффициентом сжатия — `http://www.7-zip.org/`.
- Aqua Data Studio (Java). Нетороплив, зато хорош для импорта и экспорта данных — `http://aquafold.com/`.
- ArgoUML. Инструмент разработки UML — `http://argouml.tigris.org/`.
- CruiseControl. Среда для процесса последовательного построения — `http://cruisecontrol.sourceforge.net`.
- CVS. Управление конфигурацией исходного кода — `http://www.nongnu.org/cvs/`.
- DbVisualizer. Инструмент баз данных — `http://www.minq.se`.
- FreeMind. Программное обеспечение связывания — `http://freemind.sourceforge.net/`.
- Gaim. Клиент многопротокольного *мгновенного обмена сообщениями* (Instant Messaging — IM) — `http://gaim.sourceforge.net/`.
- Программа манипулирования изображениями — `http://gimp.org/`.
- JAR Class Finder. Утилита, дополнение Eclipse, для поиска файлов JAR, содержащих определенный класс — `http://www.alphaworks.ibm.com/tech/jarclassfinder`.
- J. Текстовый редактор (Java) — `http://armedbear-j.sourceforge.net/`.

- KDiff3. Утилита для сравнения и объединения двух или трех текстовых исходных файлов или каталогов — `http://kdiff3.sourceforge.net/`.
- Дополнения Mozilla Firefox. Сотни полезных дополнений для браузера Firefox — `https://addons.mozilla.org/extensions/`.
- Netbeans. Полнофункциональная интегрированная среда разработки (IDE) — `http://www.netbeans.org/`.
- Nvu. Web-система авторизации — `http://nvu.com/`.
- OpenOffice. Полный офисный комплект приложений, сравнимый с Microsoft Office — `http://www.openoffice.org/`.
- Poseidon (выпуск сообщества). Инструмент разработки UML — `http://gentleware.com`.
- SQuirreL SQL Client. Управление базами данных, совместимыми с JDBC — `http://squirrel-sql.sourceforge.net/`.
- Sun Java Studio Enterprise. IDE с интегрированным инструментом UML — `http://developers.sun.com/prodtech/javatools/jsenterprise/`.
- TkCVS (включает TkDiff). Графический интерфейс для CVS и систем управления конфигурацией промежуточных версий — `http://www.twobarleycorns.net/tkcvs.html`.
- Vim. Текстовый редактор с GUI, подражающий 'Vi', для Unix — `http://www.vim.org/`.
- vnc2swf. Инструмент разработки экранов — `http://www.unixuser.org/%7Eeuske/vnc2swf/`.
- XEmacs. Текстовый редактор и система разработки прикладных программ — `http://www.xemacs.org/`.
- Элементы управления Yahoo — `http://widgets.yahoo.com/`.

# Инструменты для Microsoft Windows

- AppRocket. Утилита клавиатуры (для Windows) — `http://www.candylabs.com/approcket/` (пробная версия, но слишком замечательная, чтобы не упомянуть ее здесь).
- ReplaceEm. Программа поиска и замены текста для Windows. (Вероятно, самая простая, быстро устанавливаемая и наиболее интуитивно понятная программа данного типа, с которой я сталкивался.) — `http://www.orbit.org/replace/`.
- MWSnap. Снимки (копирование) изображений выделенных частей экрана — `http://www.mirekw.com/winfreeware/mwsnap.html`.
- Textpad. Мощный, универсальный редактор для текстовых файлов — `http://www.textpad.com/`.
- FileZilla. Клиент и сервер FTP — `http://filezilla.sourceforge.net/`.
- PrimoPDF. Преобразование в формат PDF из любого приложения (через печать) — `http://www.primopdf.com/`.
- Cygwin. Окружение для Windows, похожее на Linux — `http://www. cygwin.com/`.

- Sysinternals. Расширенная система утилит для Windows — `http://www.sysinternals.com/`.
- WinMerge. Визуальный инструмент сравнения и слияния текстовых файлов — `http://winmerge.sourceforge.net/`.
- Whiteboard Photo (ограниченная версия). Программное обеспечение захвата изображений — `http://www.polyvision.com/`.
- ExamDiff. Визуальный инструмент сравнения файлов — `http://www.prestosoft.com/ps.asp?page=edp_examdiff`.
- TortoiseCVS. Клиент CVS, интегрированный с проводником Windows — `http://www.tortoisecvs.org/`.

# Инструменты для Mac OS X

- Vim. Vi IMproved — улучшенный Vi для Mac OSX — `http://macvim.org/OSX/`.
- Adium. Приложение мгновенного обмена сообщениями; способно подключиться к AIM, MSN, Jabber, Yahoo и т.д. — `http://www.adiumx.com/`.
- Grab It. Часть компиляции для Mac OS X.
- Quicksilver (для Mac OS X) — `http://quicksilver.blacktree.com/`.

# Инструменты для Linux (KDE)

Трудно назвать другую платформу, обладающую таким большим количеством замечательных утилит и инструментов. Кроме того, некоторые из инструментов, не зависящих от платформы, рекомендованных в этом разделе выше, также подходят для Linux. Однако есть еще несколько средств, рекомендованных мне коллегами.

Некоторые утилиты *среды разработки K* (K Development Environment — KDE), которые имеет смысл опробовать, — `Konsole`, `Klipper`, `KEdit`, `KPrinter`, `Kate`, `Kompare`, `KFind`, `KSnapshot`, `KRuler`, `K3B`, `KAlarm`, `KTimer`, `KInfoCenter`, `KWiFiManager` и `KSayIt`.

Pollix Live CD (на базе Knoppix; загружается непосредственно с CD; установка не нужна!) — Pollix загружается с такими инструментальными средствами программирования, как Eclipse, NetBeans, Python, и другими — `http://moe.tnc.edu.tw/%7Ekendrew/pollix/`.

Другие утилиты Linux, включая `locate`, `expect` и `wish`.

# И

# Исследование визуальных шаблонов

Множество организаций, больших и маленьких, нуждаются в программном обеспечении, созданном специально для их бизнеса, поскольку имеющееся в наличии программное обеспечение зачастую не удовлетворяет их потребностям. Эти специальные приложения создаются на дому или дистанционно либо в комбинации обоих подходов. В этом приложении обсуждаются некоторые из актуальных проблем нашей отрасли при разработке программного обеспечения на заказ. Научно-исследовательский проект, который я недавно начал на моем Web-сайте, `visualpatterns.com`, посвящен решению некоторых из этих проблем.

## Проблема

Поскольку в основе бизнеса большинства этих организаций лежит программное обеспечение, они в конце концов оказываются перед проблемами, связанными с созданием сложного специального программного обеспечения.

## Прошлое: детская самодеятельность

В прошлом для большинства проектов разработки программного обеспечения применялся последовательный подход, т.е. перед переходом к следующей фазе цикла разработки программного обеспечения (например, требования, архитектура, проектирование, программирование, проверка) полностью завершалась предыдущая фаза. Кроме того, большинство этих проектов требовало множества времени и усилий на предварительные действия, подобные гаданию на хрустальном шаре. Чтобы справиться с этой проблемой, большинство организаций использует не лучший подход, они пытаются создать большой объем документации (как правило, предварительно), а затем поддерживать ее актуальность на протяжении цикла разработки проекта, что практически никогда не удается. Результаты этого метода работы по количеству успешных, неудачных и спорных проектов отражены на рис. И.1.


**Рис. И.1.** История завершения проектов (источник: `standishgroup.com`)

Результаты наблюдений группы Standish Group демонстрируют, что чем выше цена проекта, тем выше риск его провала, как свидетельствуют цифры на рис. И.2.

Влияние на успешность проекта продолжительности проектирования и размера группы

| Размер проекта | Сотрудники | Время (мес.) | Процент успешности |
|---|---|---|---|
| Меньше 750 тыс. долл. | 6 | 6 | 55% |
| От 750 тыс. до 1,5 млн. долл. | 12 | 9 | 33% |
| От 1,5 до 3 млн. долл. | 25 | 12 | 25% |
| От 3 до 6 млн. долл. | 40 | 18 | 15% |
| От 6 до 10 млн. долл. | +250 | +24 | 8% |
| Свыше 10 млн. долл. | +500 | +36 | 0% |

*Чем меньше группа и короче продолжительность проекта, тем больше вероятность успеха. Безусловно, это вовсе не означает, что сжатие сроков и уменьшение ресурсов большого проекта сделают его успешным. Из этого также не следует, что большие проекты с большими группами не могут быть успешными. Группа Standish Group полагает, что любой проект может быть успешным, если учтены все ключевые факторы.*

**Рис. И.2.** Успешность в зависимости от размера проекта (источник: `standishgroup.com`)

Кроме того, в большинство приложений встраивают средства, которые либо никогда не используются, либо используются редко или только иногда (как показано на рис. И.3); это приводит к удорожанию программного обеспечения и пустой трате усилий.


**Рис. И.3.** Использование возможностей в развернутых приложениях

# Будущее: гибкие методы

Согласно данным группы Standish Group, десятью наиболее важными факторами, необходимыми для успеха проекта, являются представленные на рис. И.4.

**Примечание**

Большинство статистических данных, приведенных в этом разделе, взяты из документов международной группы Standish Group International, Inc. Хотя они приведены только из одного источника, но вполне соответствуют тому, что я извлек из своего 20-летнего опыта в области информационных технологий.

Большинство проблем, обозначенных ранее в этом приложении, может быть решено при помощи нового, на самом деле итерационного стиля проектирования.

В 2001 году 17 методологов собрались вместе, чтобы объединить свои методики под одной крышей. Они совместно определили термин *Agile* (`agilemanifesto.org`, `agilealliance.com`). Замечательным в этом событии было то, что эти методологи согласились на общий набор принципов (см. статью на `martinfowler.com/articles/agileStory.html`).

Десять ключевых факторов

| | |
|---|---|
| Поддержка исполнителя | 18 |
| Участие пользователей | 16 |
| Опытный руководитель проекта | 14 |
| Очевидность бизнес-целей | 12 |
| Минимизация контекста | 10 |
| Стандартная инфраструктура | 8 |
| Простота требований заказчика | 6 |
| Формализация методологии | 6 |
| Реализм ожиданий | 5 |
| Другое | 5 |

*Каждый фактор был оценен, согласно его влиянию на успех проекта. Чем выше значение, тем менее рискован проект.*

**Рис. И.4.** Десять ключевых факторов успеха проекта (источник: `standishgroup.com`)

Среди основных значений гибких (Agile) методов были следующее.

- Основное внимание клиенту. Короче говоря, удовлетворите клиента. Разрабатывайте только то, что он хочет, не более и не менее.
- Охват изменений. В современном скоротечном мире изменения неизбежны. Пользователи должны быть способны изменить систему (они за это платят). Так что лучше принять этот факт и смириться.
- Итерационная разработка. Разрабатывайте программное обеспечение малыми порциями (например, двухмесячный главный выпуск с двухнедельными итерациями, в результате которых получается работоспособный код). Инкрементное программирование с постоянными дополнениями (при помощи рефакторинга) имеет неоспоримое преимущество перед созданием программного обеспечения методом больших требований и большого предварительного проектирования (BRUF/BDUF).
- Мотивируйте людей. Создавайте проект вокруг активного персонала; предоставьте им среду и пространство, в котором им будет удобно выполнить работу.
- Коммуникация. Это залог успеха, а следовательно, пользователям, разработчикам, испытателям и другим участникам проекта должна быть обеспечена хорошая связь. Кроме того, личное общение предпочтительнее заочного (электронная почта, например).

- Мера прогресса. Используйте в качестве меры прогресса работоспособное программное обеспечение (код, база данных), а не документацию и планы проектирования.
- Поддержка темпа разработки. Участники проекта (пользователи, разработчики, испытатели и т.д.) должны быть способны поддерживать постоянный темп разработки программного обеспечения (см. итерационную разработку выше).
- Лаконичная элегантность. Предпочитайте актуальные бизнес-возможности, а не заумные или сложные технические решения.
- Постоянная эффективность. Опыт свидетельствует, залог успеха — это хороший проект, полезный совет, шаблоны, контрольные списки, глоссарий и т.д.

# Мои перспективы

По моему мнению, основанному на упомянутых ранее значениях Agile и двадцатилетнем опыте в области IT, наиболее серьезными проблемами разработки программного обеспечения являются завышенные требования, большой проект и терминология. Эти проблемы обсуждаются далее.

## BRUF и BDUF

В сообществе Agile *большие предварительные требования* (Big Requirements Up Front) и *большое предварительное проектирование* (Big Design Up Front) зачастую упоминаются как сокращения BRUF и BDUF соответственно. Принцип получения всех требований заранее (каскадный метод) и их неизменности, на мой взгляд, безумен, это архаичный стиль разработки программного обеспечения. Предварительно достаточно выдвинуть лишь некоторые из требований, например на уровне выпуска. Остальные требования (более детальные) могут быть получены в начале каждой итерации или после приемочных испытаний.

Подобно BRUF, попытки предварительно завершить все проектирование архитектуры неблагоразумны, поскольку впоследствии, когда разработчики приступят к программированию, вы неизбежно обнаружите лучшие способы построения системы. Некий проект архитектуры на нулевом цикле — это хорошая идея, однако его обсуждение должно занимать не более нескольких минут или часов, в зависимости от сложности системы, однако отнюдь не дни или недели (Джим Хайсмит). Кроме того, поддержание актуальности документации BRUF и BDUF на протяжении цикла разработки программного обеспечения — это трудная задача для любой организации, причем практически всегда невыполнимая.

## Терминология

Для меня это особенно больная тема, поскольку, на мой взгляд, многие технически грамотные люди просто стыдятся признать терминологию проблемой. Когда вы имеете изобилие сокращений (акронимов), избыточных (хранилище, постоянный) или неоднозначных терминов (пессимистическая блокировка, неизменность), это не только приводит к проблемам общения технического персонала, но и снижает эффективность переговоров с пользователями.

# Присоединяться ли к сообществу

В ближайшее время я буду создавать глобальное виртуальное сообщество из людей в разных частях мира, и этот процесс уже начался. Наша цель будет заключаться в исследовании наилучших методов моделирования и уникальных визуальных методов технологий, призванных решить некоторые из проблем, затронутых в этом приложении.

Если решились присоединиться к этому виртуальному сообщества, свяжитесь со мной через `visualpatterns.com`.