ЧАСТЬ I

Введение в паттерны проектирования Java EE

1 Краткий обзор паттернов проектирования

В этой главе:

- обзор паттернов проектирования;
- краткая история паттернов проектирования и пояснение, чем они так важны;
- использование паттернов проектирования на практике;
- история и эволюция Java Enterprise Edition;
- появление корпоративных паттернов;
- как эти паттерны проектирования развиваются в корпоративном окружении;
- как и почему паттерны становятся антипаттернами.

Цель этой книги — наведение мостов между традиционной реализацией паттернов проектирования в среде Java SE и их реализацией в Java EE.

Если вы новичок в паттернах проектирования, эта книга поможет вам быстро войти в курс дела, так как каждая глава знакомит с паттерном проектирования в простой и понятной манере, с множеством примеров работающего кода.

Если же вы уже хорошо знакомы с паттернами проектирования и их реализацией, но не с их реализацией в среде Java EE, то эта книга идеальна для вас. Каждая глава соединяет традиционную реализацию и новую, зачастую более простую реализацию в Java EE.

Если вы эксперт в языке Java, это издание послужит вам надежным справочником по реализациям большинства распространенных паттернов проектирования на платформах Java SE и Java EE.

Эта книга сосредоточена на наиболее распространенных паттернах проектирования платформы Java EE и наглядно показывает, как они реализованы во вселенной Java EE. Каждая глава вводит новый паттерн, объясняя его суть и обсуждая приносимую им пользу. Затем в ней демонстрируется реализация этого паттерна в Java SE и дается детальное описание того, как он работает. В издании наглядно показывается, как паттерн в данный момент реализован в Java EE, и обсуждается наиболее распространенное его применение, его преимущества и подводные камни. Все объяснения сопровождаются подробными примерами кода, каждый из которых может быть скачан с прилагаемой к книге веб-страницы. В конце любой главы вы найдете заключительное обсуждение и краткое резюме, подытоживающее

все прочитанное вами ранее. А еще для вас там есть несколько интересных и иногда весьма непростых упражнений — для проверки понимания охваченных этой главой паттернов.

Что такое паттерн проектирования

> Паттерны проектирования — это «описания обменивающихся информацией объектов и классов, которые адаптированы для решения общей проблемы проектирования в конкретных условиях».
>
> *«Банда четырех»*

Паттерны проектирования предлагают решения распространенных проблем проектирования приложений. В объектно-ориентированном программировании паттерны проектирования обычно нацелены на решение задач, связанных с созданием и взаимодействием объектов, а не крупномасштабных задач, стоящих перед общей архитектурой программного обеспечения. Они обеспечивают обобщенные решения в виде шаблонов, которые могут быть применены к практическим задачам.

Обычно паттерны проектирования наглядно представляют в виде диаграмм классов, демонстрирующих поведение классов и отношения между ними. Типичная диаграмма классов показана на рис. 1.1.

Здесь продемонстрированы отношения наследования между тремя классами. Подклассы `CheckingAccount` и `SavingsAccount` наследуют от их абстрактного родительского класса `BankAccount`.

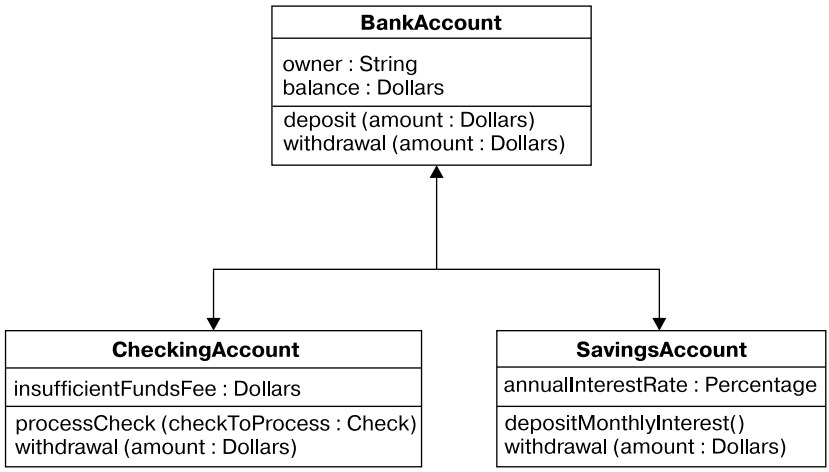

Рис. 1.1. Диаграмма классов, иллюстрирующая наследование

За такой диаграммой следует реализация на языке Java, демонстрирующая простейшую реализацию. Пример паттерна «Одиночка» (Singleton), который будет описан в дальнейших главах, показан на рис. 1.2.

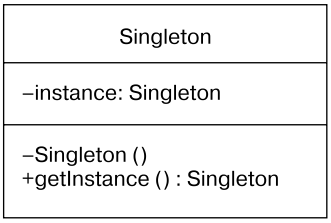

Рис 1.2. Диаграмма классов паттерна «Одиночка»

А вот пример его простейшей реализации.

```
public enum MySingletonEnum {
    INSTANCE;
    public void doSomethingInteresting(){}
}
```

Как были изобретены паттерны проектирования и почему они нам нужны

Паттерны проектирования стали злободневным вопросом с тех пор, как знаменитая «Банда четырех» (GoF, состоящая из Эриха Гаммы, Ричарда Хелма, Ральфа Джонсона и Джона Влиссидеса) написала книгу «Приемы объектно-ориентированного проектирования. Паттерны проектирования», наконец-то предложив разработчикам со всего мира испытанные и проверенные решения для наиболее распространенных проблем программотехники. Эта важнейшая книга описывает различные методики разработки, их подводные камни и предоставляет 23 паттерна проектирования. Паттерны разделены на три категории: порождающие, структурные и паттерны поведения.

Но почему? Почему мы неожиданно осознали, что нам так нужны паттерны проектирования?

Решение не было таким уж внезапным. Объектно-ориентированное программирование появилось в 1980-х годах, и вскоре за ним последовали несколько основанных на этой новой идее языков программирования. Smalltalk, C++ и Objective C — часть из того небольшого количества объектно-ориентированных языков, которые до сих пор остались широко распространены. Впрочем, они принесли и свои собственные проблемы, и, в отличие от эволюции процедурного программирования, в этот раз перемены были слишком быстрыми, чтобы можно было увидеть, что было работоспособным, а что — нет.

Хотя паттерны проектирования решили много проблем (например, спагетти-код, то есть плохо структурированные программы), возникавших у специалистов по программному обеспечению с процедурными языками программирования, такими как C и COBOL, объектно-ориентированные языки внесли свой набор спорных вопросов. C++ быстро развивался и из-за своей сложности увел многих разработчиков на минные поля программных ошибок, таких как утечки памяти, неудачное проектирование объектов, небезопасное использование памяти и неудобный в сопровождении унаследованный код.

Однако большинство задач, с которыми сталкиваются разработчики, строятся по одним и тем же паттернам, и вполне разумно предположить, что кто-то когда-то уже нашел для них решение. В те времена, когда появилось объектно-ориентированное программирование, в доинтернетовском мире, было непросто обмениваться опытом с широкими массами народа. Поэтому прошло некоторое время, прежде чем GoF создали набор паттернов для известных повторяющихся проблем.

Паттерны в реальном мире

Паттерны проектирования — бесконечно полезные и проверенные решения для задач, с которыми вы неизбежно встретитесь. Они не только передают многолетние совокупные знания и опыт, но и предлагают полезный справочник для общения между разработчиками и проливают свет на многие проблемы.

Тем не менее паттерны проектирования не волшебная палочка, они не предоставляют, подобно фреймворкам или наборам утилит, готовой для использования реализации. Излишнее — только потому, что это модно, или потому, что вы хотите произвести впечатление на начальника, — использование паттернов проектирования может вылиться в переусложненную систему, которая не решит никаких задач, но взамен продемонстрирует наличие ошибок, неумелый дизайн, низкую производительность и проблемы с сопровождением. Большинство паттернов может решить проблемы дизайна, обеспечить надежные решения известных задач и позволит разработчикам общаться общими идиомами через пропасть между разными языками. Паттерны, вообще говоря, следует использовать лишь тогда, когда существует вероятность возникновения проблем.

Паттерны проектирования были изначально разделены на три группы.

- **Порождающие паттерны** управляют созданием и инициализацией объекта, а также выбором класса. Примерами паттернов из этой группы могут быть «Одиночка» (см. главу 4) и «Фабрика» (см. главу 6).
- **Паттерны поведения** управляют связью, обменом сообщениями и взаимодействием между объектами. Примером паттерна из этой группы может быть «Наблюдатель» (см. главу 11).
- **Структурные паттерны** упорядочивают отношения между классами и объектами, обеспечивая критерии соединения и совместного использования связанных объектов для достижения желаемого поведения. Хороший пример паттерна из этой группы — «Декоратор» (см. главу 7).

Паттерны проектирования предлагают разработчикам своеобразный общий справочник. Разработчики могут использовать их, чтобы общаться гораздо проще, не изобретая колесо для каждой задачи. Хотите показать вашему приятелю, как вы собираетесь добавлять динамическое поведение во время исполнения? Довольно пошаговых рисунков и недоразумений! Все просто и ясно: вам достаточно произнести несколько слов: «Давай использовать для этой задачи паттерн “Декоратор”!» Ваш друг тотчас же поймет, о чем вы говорите, не нужны никакие дополнительные разъяснения. Если вы уже знаете, что такое паттерн, и используете его в правильном контексте, вы явно встали на путь создания надежного и легкосопровождаемого приложения.

РЕКОМЕНДУЕМОЕ ЧТЕНИЕ

Мы настоятельно рекомендуем вам прочитать «Приемы объектно-ориентированного проектирования. Паттерны проектирования» Эриха Гаммы, Ричарда Хелма, Ральфа Джонсона и Джона Влиссидеса или «Паттерны проектирования» Эрика Фримена, Элизабет Фримен, Кэтти Сьерра и Берта Бейтса[1] (Head First Design Patterns). Обе книги — прекрасные дополнения к той, которую вы сейчас читаете, и бесценные руководства по изучению паттернов проектирования.

Основы паттернов проектирования

Одним из ключевых нюансов, относящихся к паттернам проектирования, является то, что злоупотребление ими или излишнее их использование может стать источником проблем. Как только некоторые разработчики выучивают новые паттерны, им страшно хочется использовать их где только можно. Однако подобное поведение часто приводит к тому, что их проекты разбухают от одиночек или заключены в оболочку фасадов или неоправданно сложных декораторов. Паттерны проектирования — ответ на все вопросы, так что, если нет проблемы или хотя бы шанса, что проблема появится, реализовывать паттерн смысла нет. Для примера: использование паттерна «Декоратор» только из-за незначительной вероятности будущего изменения поведения объекта приносит сложность разработки сейчас и кошмар при сопровождении в будущем.

Корпоративные паттерны

Язык Java 1.0 быстро стал популярным после своего выхода в начале 1996 года. Выбор момента был идеален для представления нового языка, который бы устранил сложность управления памятью, синтаксиса и работы с указателями языков C/C++. Java предлагал постепенную кривую обучения, позволявшую многим разработчикам быстро его усвоить и начать программировать на нем. Однако было еще нечто, ускорившее перемены, — апплеты. Апплет — это маленькое приложение, запускаемое на сайте в отдельном процессе браузера и добавляющее сайту функциональность, невозможную при использовании только HTML и CSS. Примером может служить интерактивный график или передача потокового видео.

С быстрым ростом Интернета статические веб-страницы вскоре стали устаревшими и скучными. Пользователям хотелось получать от веб-серфинга больше информации и делать это быстрее. И в этот момент появились апплеты, которые предлагали невероятные интерактивность, эффекты и экшен для статичной тогда Всемирной паутины.

Вскоре танцующий Дюк (символ языка Java) стал общей тенденцией для современных сайтов. Однако в Интернете ничего не остается неизменным надолго. Пользователи хотели большего, а апплеты потерпели полный крах в попытках приспособиться к этим требованиям и не сумели сохранить свою популярность.

[1] *Фримен Э., Фримен Э., Сьерра К., Бейтс Б.* Паттерны проектирования. — СПб.: Питер, 2011. — 656 с.

Тем не менее апплеты были той движущей силой, которая стояла за быстрым приспособлением и популярностью платформы Java. Сегодня (на момент написания этой книги) Java все еще один из двух самых популярных языков программирования в мире: согласно индексу TIOBE, Java находится на втором месте по популярности после C[1] (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html).

От языка Java к корпоративной платформе Java

Вслед за выпуском стандартной версии Java (Java Standard Edition) IBM представила в 1997 году корпоративную модель Java-компонентов (Enterprise JavaBeans, EJB), которая была в 1999 году принята Sun и составила часть корпоративной платформы Java (Enterprise Java Platform, J2EE) версии 1.2. В 1998 году, перед выпуском J2EE[2], Sun выпустила профессиональную версию Java, получившую название JPE. Однако только после выхода EJB поставщики и разработчики заинтересовались возможностью использования корпоративной платформы Java. А с выходом J2EE 1.3 в 2001 году Java становится ключевым игроком мира корпоративных приложений, еще более закрепив свои позиции выпуском J2EE 1.4 в 2003 году.

Версия 1.4 была одной из важнейших вех в истории Java. Она широко применялась и удерживала свою популярность долгие годы, несмотря на выпуск новых версий. Поставщики и корпорации не торопились переходить на новые версии, хотя причины жаловаться на J2EE 1.4 были у многих. Использование ее можно было сравнить с поездкой по магазинам на монстр-траке вместо семейного седана. Несомненно, она была производительной, но попросту слишком сложной и слишком раздутой от XML-файлов, и как фреймворки, так и контейнеры были весьма тяжеловесными.

Все же J2EE стала наиболее популярной корпоративной платформой разработки. У нее был набор возможностей, делавший ее отличным выбором для корпоративной разработки.

- **Переносимость** — JVM позволяла Java-коду запускаться в любой операционной системе. Программисты могли вести разработку в Windows, тестировать в Linux, а вводить в промышленную эксплуатацию в UNIX-системе.
- **Безопасность** — J2EE предлагала свою собственную ролевую модель безопасности.
- **Транзакции** — J2EE предлагала встроенные транзакции.
- **Возможность языка платформы J2SE** — J2SE предлагала простой синтаксис, сборку мусора и великолепные объектно-ориентированные возможности программирования.

---

[1] Согласно вышедшему в июне 2015 года очередному ежегодному выпуску индекса TIOBE, Java укрепил свои позиции и потеснил C с первого места. — *Примеч. пер.*

[2] До версии 5 Java EE обычно называлась J2EE. С этого места мы будем использовать термин J2EE для версий, предшествовавших Java EE 5.

Однако J2EE была несовершенной. Достаточно скоро сложное устройство платформы с ее интенсивным использованием XML-конфигураций создало идеальную среду для возникновения проблем.

Появление корпоративных паттернов Java

Сложные модели программирования J2EE вскоре поставили многие проекты в затруднительное положение. Созданные с помощью J2EE-технологий приложения имели склонность содержать непомерное количество «канализационного» кода вроде кода поиска JNDI[1], файлов XML-конфигураций и блоков `try/catch`, запрашивающих и освобождающих ресурсы JDBC[2]. Написание и обслуживание такого кода на практике становилось причиной серьезного расхода ресурсов и было источником множества программных ошибок и проблем с производительностью. Компонентная модель EJB была призвана снизить сложность при реализации бизнес-логики, но не преуспела в этом. Модель оказалась слишком сложной и зачастую используемой чрезмерно.

Спустя всего несколько лет после первого выпуска платформы Дипак Алур, Джон Друки и Дэн Малкс выступили на конференции JavaOne в 2000 году с докладом «Макетирующие паттерны для платформы J2EE», в котором представили несколько паттернов, нацеленных на решение распространенных проблем, встречающихся при дизайне приложений под J2EE. Этот доклад стал книгой. На следующий год они опубликовали книгу под названием *Core J2EE Patterns: Best Practices and Design Strategies*[3]. В дополнение к 15 уже широко известным паттернам авторы представили еще шесть новых паттернов проектирования для J2EE. Новые паттерны включали Context Object и Application Controller для слоя представления, Application Service и Business Object — для слоя бизнес-логики, Domain Store и Web Service Broker — для слоя интеграции.

Некоторые из этих паттернов развились из «классических» паттернов GoF, в то время как остальные были новыми, направленными на борьбу с изъянами J2EE. В последующие годы было выпущено несколько проектов и фреймворков, таких как Apache Camel, облегчающих жизнь корпоративных разработчиков. Некоторые даже под руководством Рода Джонсона[4] поступили смелее, отойдя от J2EE и выпустив фреймворк Spring. Spring быстро обрел популярность, что оказало свое влияние на масштабные перемены в новой модели программирования в Java EE. На сегодняшний день большинство из этих паттернов используются и остаются вполне пригодными. Впрочем, некоторые устарели и более не нужны благодаря упрощенной модели программирования Java EE.

---

1 Java Naming and Directory Interface — стандартный программный интерфейс к корпоративной службе каталогов. — *Примеч. пер.*

2 Java Database Connectivity — интерфейс Java для доступа к базам данных. — *Примеч. пер.*

3 *Алур Дипак, Крупи Джон, Малкс Дэн.* Образцы J2EE. Лучшие решения и стратегии проектирования. — М.: Лори, 2013. — 375 с.

4 Род Джонсон (@springrod) — известный австралийский специалист по компьютерной технике, создавший фреймворк Spring, сооснователь компании SpringSource. http://en.wikipedia.org/wiki/Rod_Johnson_(programmer).