Содержание

ГЛАВА 1

Введение

Эта книга разработана с тем, чтобы помочь вам максимально эффективно использовать возможности языка программирования Java и его основных библиотек `java.lang`, `java.util` и `java.io`, а также подпакетов наподобие `java.util.concurrent` и `java.util.function`. Прочие библиотеки рассматриваются эпизодически.

Эта книга состоит из 90 разделов, каждый из которых посвящен одному правилу. В этих правилах собран опыт, который лучшие, наиболее опытные программисты считают весьма полезным. Эти разделы сгруппированы в одиннадцать глав, каждая из которых охватывает один из аспектов проектирования программного обеспечения. Книга не предназначена для чтения от корки до корки: каждый раздел более или менее самодостаточен. Разделы снабжены перекрестными ссылками, позволяющими пройти собственный путь через книгу.

Со времени публикации предыдущего издания книги к платформе Java было добавлено немало новых функциональных возможностей. Большинство разделов этой книги тем или иным способом используют эти возможности. В приведенной ниже таблице показано, где именно в первую очередь освещены те или иные ключевые особенности языка.

Функциональная возможность	**Разделы**	**Версия Java**
Лямбда-выражения	7.1–7.3	8
Потоки	7.4–7.7	8
Использование класса Optional	8.7	8
Методы по умолчанию в интерфейсах	4.7	8
try-с-ресурсами	2.9	7
@SafeVarargs	5.7	7
Модули	4.1	9

Большинство разделов проиллюстрированы примерами программ. Ключевой особенностью этой книги является то, что она содержит примеры кода,

иллюстрирующие многие проектные шаблоны и идиомы. Там, где это уместно, представлены перекрестные ссылки на стандартный справочник в этой области [12].

Многие разделы содержат один или несколько примеров программ, иллюстрирующих некоторые практики, которых следует избегать. Такие примеры, иногда известные как "*антишаблоны*" (антипаттерны), ясно указываются с помощью комментариев наподобие

```
// Никогда этого не делайте!
```

В каждом случае поясняется, почему этот пример плох, и предлагается альтернативный подход.

Эта книга не для начинающих: предполагается, что вы уже знакомы с Java. Если это не так, обратитесь к одной из множества книг для новичков, таких как *Java Precisely* Питера Сестофта (Peter Sestoft) [41]. Хотя данная книга должна быть доступна любому обладающему рабочим знанием языка, она должна дать пищу для размышлений даже самым "продвинутым" программистам.

Большинство правил этой книги вытекают из нескольких основополагающих принципов. Ясность и простота имеют первостепенное значение. Пользователь компонента никогда не должен удивляться его поведению. Компоненты должны быть как можно меньшими, но не меньше, чем нужно. (В этой книге термин *компонент* относится к любому повторно используемому элементу программы, от отдельных методов до сложных каркасов, состоящих из нескольких пакетов.) Код должен повторно использоваться, а не копироваться. Зависимости между компонентами должны быть сведены к минимуму. Ошибки должны обнаруживаться немедленно после того, как они сделаны, в идеале — во время компиляции.

Хотя правила из этой книги и не применяются постоянно во время работы, в подавляющем большинстве случаев они характеризуют лучшие практики программирования. Вы не должны следовать этим правилам рабски и без размышлений, но нарушать их следует лишь изредка и по уважительной причине. Обучение искусству программирования, как и большинству других дисциплин, состоит из, во-первых, обучения правилам и, во-вторых, обучения, когда эти правила нарушать.

По большей части эта книга не посвящена вопросам производительности. Речь идет о написании ясных, правильных, полезных, надежных, гибких и легких в обслуживании и поддержке программ. Если вы можете написать такую программу, то добиться требуемой производительности — обычно относительно простой вопрос (раздел 9.11). В некоторых разделах обсуждаются проблемы производительности и в некоторых из них приводятся цифры, эту производительность характеризующие. Эти цифры, которые обычно дополнены словами

“на моей машине”, следует рассматривать как в лучшем случае очень приблизительные.

Для тех, кому это важно: у меня старенькая домашняя машина с четырехъядерным 3.5 ГГц процессором Intel Core i7-4770K с 16 Гбайтами DDR3-1866 CL9 RAM, работающая с Azul’s Zulu 9.0.0.15-версией OpenJDK, с операционной системой Microsoft Windows 7 Professional SP1 (64-bit).

При обсуждении возможностей языка программирования Java и его библиотек иногда необходимо сослаться на определенные версии. Для удобства в этой книге используются краткие названия вместо официальных. В приведенной далее таблице показано соответствие официальных названий используемым в книге кратким названиям.

Официальное название выпуска	**Краткое название**
JDK 1.0.x	Java 1.0
JDK 1.1.x	Java 1.1
Java 2 Platform, Standard Edition, v1.2	Java 2
Java 2 Platform, Standard Edition, v1.3	Java 3
Java 2 Platform, Standard Edition, v1.4	Java 4
Java 2 Platform, Standard Edition, v5.0	Java 5
Java Platform, Standard Edition 6	Java 6
Java Platform, Standard Edition 7	Java 7
Java Platform, Standard Edition 8	Java 8
Java Platform, Standard Edition 9	Java 9

Примеры в книге достаточно полные, но удобочитаемость предпочтительнее полноты. В них используются классы из пакетов `java.util` и `java.io`. Для компиляции примеров, возможно, придется добавить одно или несколько объявлений импорта или иной подобный шаблон. Веб-сайт книги по адресу `http://joshbloch.com/effectivejava` содержит расширенную версию каждого примера, который можно скомпилировать и запустить.

По большей части в этой книге используются технические термины, определенные в книге *The Java Language Specification, Java SE 8 Edition* [25]. Несколько терминов заслуживают отдельного упоминания. Язык поддерживает четыре разновидности типов: *интерфейсы* (включая *аннотации*), *классы* (включая *перечисления*), *массивы* и *примитивы*. Первые три называются *ссылочными типами*. Экземпляры класса и массивы являются *объектами*; примитивные значения таковыми не являются. *Члены* класса включают его *поля*, *методы*, *классы-члены* и *интерфейсы-члены*. *Сигнатура* метода состоит из его имени и типов его формальных параметров; сигнатура *не* содержит тип возвращаемого методом значения.

В этой книге используется несколько терминов, отличных от используемых в упомянутой книге. Здесь термин *наследование* (inheritance) используется как синоним для создания подклассов (subclassing). Вместо использования термина *наследования* для интерфейсов книга просто говорит, что класс *реализует* интерфейс или что один интерфейс *расширяет* другой. Чтобы описать уровень доступа, применяемый при отсутствии явного указания, в книге используется традиционный доступ *закрытый на уровне пакета* (package-private) вместо технически правильного *доступа пакета* (package access) [25, 6.6.1].

Также здесь используется несколько технических терминов, которые не определены в упомянутой книге. Термин *экспортируемый API*, или просто *API*, ссылается на классы, интерфейсы, конструкторы, члены и сериализованные формы, с помощью которых программист обращается к классу, интерфейсу или пакету. (Аббревиатуре “API”, означающей *интерфейс прикладного программирования*, отдается предпочтение перед термином *интерфейс*, чтобы избежать путаницы с конструкцией языка с этим названием.) Программист, который пишет программу, которая использует API, именуется *пользователем* API. Класс, реализация которого использует API, является *клиентом* API.

Классы, интерфейсы, конструкторы, члены и сериализованные формы вместе называются *элементами* API. Экспортированный API состоит из элементов API, которые доступны за пределами пакета, в котором определен API. Это те элементы API, которые может использовать любой клиент и которые автор API обязуется поддерживать. Не случайно они также являются элементами, для которых утилита Javadoc генерирует документацию в режиме работы по умолчанию. Грубо говоря, экспортируемый API пакета состоит из открытых и защищенных членов и конструкторов каждого открытого класса или интерфейса в пакете.

В Java 9 в платформу была добавлена *модульная система* (module system). Если библиотека использует модульную систему, ее экспортированный API представляет собой объединение экспортированных API всех пакетов, экспортируемых объявлением модуля библиотеки.