Java 9. Полный обзор нововведений

Для быстрого ознакомления и миграции

Кишори Шаран

ДМК Пресс
издательство

УДК 004.438Java
ББК 32.973.26-018.1
Ш25

Ш25 Кишори Шаран
Java 9. Полный обзор нововведений. Для быстрого ознакомления и миграции. / пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2018. – 544 с.: ил.

ISBN 978-5-97060-575-2

В книге рассмотрены все нововведения в Java 9. Приведена разработка приложений с использованием системы модулей. Описаны измененная модульная структура образа среды выполнения JDK/JRE, новые фабричные методы для создания коллекций, уведомления процессора об активном ожидании с целью оптимизации потребления ресурсов, новый API платформенно-зависимого рабочего стола, API реактивных потоков и др. Уделено внимание несовместимым изменениям в Java 9. Также рассказано об оболочке JShell, предназначенной для быстрого создания прототипа.

Издание предназначено опытным Java-разработчикам, которым необходима миграция от Java 7 и 8 к Java 9.

УДК 004.438Java
ББК 32.973.26-018.1

ISBN 978-1-4842-2591-2 (англ.)
ISBN 978-5-97060-575-2 (рус.)

Оглавление

Глава 1

Введение

Краткое содержание главы:

- что включено в JDK 9;
- как читать эту книгу;
- требования к системе;
- как установить NetBeans;
- как скачать и использовать исходный код примеров.

Введение в JDK 9

JDK – девятая основная версия Java Development Kit, выпуск которой намечен на конец июля 2017 года. В нее входит несколько важных новых возможностей. В этой книге описаны те возможности, которые полезны разработчикам Java-приложений в их повседневной работе. Полный перечень всех изменений опубликован по адресу `http://openjdk.java.net/projects/jdk9/`.

Одна из самых важных и долгожданных возможностей JDK 9 – *система модулей*, разрабатывавшаяся в рамках проекта с кодовым названием *Jigsaw*. Проектировщики JDK пытались включить систему модулей в платформу Java более 10 лет. Планировалось, что она станет частью предыдущих версий JDK, но каждый раз что-то срывалось. Из-за включения Jigsaw даже несколько раз переносились сроки выпуска JDK 9. Но наконец-то Jigsaw вышел, и скоро вы сможете увидеть его в действии.

Что же такого трудного в Jigsaw, что на доведение до ума понадобилось много лет? Основная цель Jigsaw – предоставить программистам методику разработки приложений в терминах компонентов, именуемых *модулями*. Модуль может экспортировать свой открытый API и инкапсулировать внутренние детали. Модуль также может объявить зависимости от других модулей, и они будут проверяться на этапе инициализации, что позволит впоследствии избежать сюрпризов из-за отсутствия типов. Сам JDK представлен в виде набора взаимодействующих модулей, так что среда выполнения оказывается масштабируемой. Теперь, если в приложении используется только подмножество JDK, то можно создать образ среды выполнения, который будет содержать только необходимые модули JDK и модули приложения. Во всем этом нет ничего удивительного. Главная же пробле-

ма, преследовавшая проектировщиков JDK, заключалась в обеспечении обратной совместимости и порядка перехода на систему модулей. Язык Java существует уже больше 20 лет. Для любого крупного нововведения, в т. ч. системы модулей, необходимо обеспечить преемственность. В результате нескольких итераций и уточнений, основанных на замечаниях сообщества, цель наконец-то достигнута! Я посвятил 9 глав (со 2 по 10), т. е. свыше 40% книги описанию системы модулей.

Еще одно важное новшество в JDK 9 – оболочка JShell, реализующая цикл чтения-выполнения-печати (Read-Eval-Print Loop, REPL) для Java. JShell – одновременно командная утилита и API, который позволяет выполнить фрагмент кода и сразу получить результат. До появления JShell для получения результата необходимо было написать полную программу, откомпилировать и выполнить ее. JShell – инструмент, которым вы будете пользоваться постоянно, удивляясь, как столько лет обходились без него. JShell – бесценное подспорье для начинающих, поскольку поможет им быстро выучить язык, не вдаваясь в такие детали структуры программы, как модули и пакеты. Обсуждению JShell посвящена глава 11.

В JDK 9 есть еще несколько нововведений, облегчающих жизнь разработчика, в том числе: API реактивных потоков, методы фабрики коллекций, инкубаторный клиентский API протокола HTTP/2, API навигации по стеку, API платформенного протоколирования и унифицированного протоколирования JVM. Все эти темы подробно обсуждаются в книге.

Как читать эту книгу

Книга состоит из 20 глав. Главы со второй по десятую посвящены одной теме – системе модулей. Их рекомендуется читать последовательно. Конечно, можно перенести существующее приложение на JDK 9, вообще ничего не зная о модулях. Но если вам хочется получить преимущества, которые дает JDK 9, то необходимо изучить, что такое модули, и начать пользоваться ими при разработке новых и переносе старых приложений. Главы с 11 по 20 можно читать в любом порядке.

Глава 2 «Система модулей» содержит краткую историю проблем, с которыми сталкивались разработчики в JDK версии 8 и более ранних. Здесь же дается введение в систему модулей и объясняется, как она решает проблемы инкапсуляции API, конфигурирования и масштабируемости. Вводится много новых терминов, относящихся к системе модулей. Даже если вы их не понимаете, продолжайте читать. Более полные пояснения будут даны в последующих главах.

В *главе 3 «Создаем свой первый модуль»* вы увидите модули в действии. Будет показано, как объявить модуль, написать его код, откомпилировать и выполнить класс, являющийся частью модуля. Но в детали объявления модулей мы пока не вдаемся.

Глава 4 «Зависимости модулей» представляет собой интенсивный практический курс по системе модулей. Она опирается на две предыдущие главы и содержит более полное изложение вопроса о зависимостях модулей и экспорте пакетов из модуля. Показано, как объявляются модули разных типов, как смешивать модульный и немодульный код в одном приложении и еще несколько возможностей системы модулей. Материал, изложенный в этой главе, должен понимать любой серьезный Java-разработчик, планирующий использовать JDK 9.

В *главе 5 «Реализация служб»* показано, как с помощью простых директив системы модулей реализовать службы в JDK 9. Здесь же объясняется, как службы реализовывались раньше.

В *главе 6 «Упаковка модулей»* описываются различные форматы упаковки модулей и объясняется, когда какой формат лучше использовать. Вплоть до версии JDK 8 существовал только один способ упаковки Java-приложений – JAR-файлы. В этой главе показано, как упаковывать модули в файл формата JMOD.

В *главе 7 «Создание пользовательских образов среды выполнения»* описано, как создавать образы среды выполнения во внутреннем формате JIMAGE с помощью программы `jlink`.

В *главе 8 «Несовместимые изменения в JDK 9»* описываются изменения, которые могут привести к тому, что приложения, перенесенные на JDK 9, перестанут работать. Здесь же даются рекомендации, как исправить код, и сообщается о предлагаемых альтернативах, если таковые существуют.

В *главе 9 «Нарушение инкапсуляции модуля»* объясняется, как можно обойти главный постулат системы модулей – инкапсуляцию. JDK 9 предлагает несколько нестандартных параметров командной строки, которые позволяют получить доступ к инкапсулированному и вообще-то недоступному извне коду модуля. Это может понадобиться при переносе унаследованных приложений, ранее имевших доступ к внутренним API JDK или сторонних библиотек, которые после перехода на JDK 9 оказались инкапсулированы в модулях.

В *главе 10 «API модулей»* речь пойдет о том, как обращаться к системе модулей из программы с помощью API модулей (Module API). Все, что можно сделать посредством объявлений модулей и параметров командной строки, доступно и через API модулей.

Глава 11 «Оболочка Java» посвящена программе JShell и JShell API. Сюда входит печать сообщений, объявление классов и использование существующих модулей внутри `jshell`. Описывается, как конфигурировать `jshell` и как использовать JShell API для выполнения фрагментов кода.

В *главе 12 «Изменения API процессов»* описываются новые возможности API процессов (Process API) в JDK 9. Полного описания API процессов в том виде, в каком он существует со времен JDK 1.0, вы здесь не найдете.

В *главе 13 «Изменения API коллекций»* описываются новые возможности API коллекций (Collection API) в JDK 9, в т. ч. статические фабричные методы создания списков, множеств и отображений.

Глава 14 «Клиентский API HTTP/2» посвящена клиентскому API новой версии протокола, HTTP/2. Это часть инкубаторного, т. е. еще не окончательно стандартизованного, модуля. В версии JDK 10 он либо будет стандартизован и станет полноправной частью платформы Java SE, либо будет исключен.

В *главе 15 «Модифицированный тип Deprecated»* описываются изменения в типе аннотаций `Deprecated`, порядок использования аннотации `@SuppressWarnings` в JDK 9 и сканирование кода для поиска нерекомендованных API с помощью программы `jdeprscan`.

В *главе 16 «Навигация по стеку»* рассматривается API навигации по стеку (Stack-Walking API), появившийся в версии JDK 9. Его можно использовать для обхода стека и нахождения ссылок на вызывающие классы.

В *главе 7 «Реактивные потоки»* речь идет об API реактивных потоков (Reactive Streams API) – реализации спецификации реактивных объектов в Java.

В *главе 18 «Изменения API потоков»* описываются новые методы, добавленные в API потоков (Streams API). Рассматриваются также новые коллекторы в API потоков.

В *главе 19 «Протоколирование на уровне платформы и JVM»* описывается новый API платформенного протоколирования, позволяющий отправлять сообщения из платформенных классов пользовательскому диспетчеру протоколирования. Здесь показано, как использовать для этой цели библиотеку Log4j. В этой же главе показано, как обращаться к журналам JVM с помощью параметра командной строки и как настроить сообщения JVM перед записью в журнал.

В *главе 20 «Прочие изменения в JDK 9»* собраны все остальные изменения, представляющие интерес для разработчиков приложений и не нашедшие отражения в предыдущих главах. В каждом разделе рассматривается одна тема. Можете прочитать только то, что вам интересно, а остальное пропустить. К вопросам, обсуждаемым в этой главе, относятся использование знака подчеркивания в качестве ключевого слова, улучшенные блоки `try` с ресурсами, закрытые методы в интерфейсах, потоки объектов типа `Optional`, использование фильтров в процессе десериализации объектов, новые методы в API ввода-вывода и др.

Требования к системе

Чтобы выполнять описанные в книге программы и писать свои собственные, необходимо установить на свой компьютер JDK 9.

На момент написания книги JDK 9 еще разрабатывался. Выпуск производственной версии запланирован на конец июля 2017 года. Текущая сборка называется *Early Access* Build 157. Скачать самую свежую версию можно со страницы по адресу `https://jdk9.java.net/download/`. Там же приведены инструкции по установке JDK 9 для разных операционных систем.

Чтобы проверить версию установленного JDK, выполните команду:

```
c:\> java -version
```

```
java version "9-ea"
Java(TM) SE Runtime Environment (build 9-ea+157)
Java HotSpot(TM) 64-Bit Server VM (build 9-ea+157, mixed mode)
```

Обратите внимание на строку `9-ea`. Она означает, что установлена версия JDK 9 Early Access. В окончательной версии будет указан только номер 9.

Возможно, что при выполнении команды будет выдана ошибка вида:

```
c:\> java -version
```

```
'java' is not recognized as an internal or external command,
operable program or batch file.
```

Она означает, что каталог, содержащий команду `java`, не включен в переменную среды `PATH`. Команда `java` находится в каталоге `JDK_HOME\bin`, где `JDK_HOME` – каталог, в который установлен JDK 9. На моей системе Windows это каталог `C:\java9`. Для добавления каталога `JDK_HOME\bin` в переменную среды `PATH` в каждой операционной системе есть свои методы. Можно также указывать полный путь к команде, например: `C:\java9\bin\java` в Windows. Ниже для печати сведений о версии указан полный путь к команде `java`:

```
c:\> C:\java9\bin\java -version
```

```
java version "9-ea"
Java(TM) SE Runtime Environment (build 9-ea+157)
Java HotSpot(TM) 64-Bit Server VM (build 9-ea+157, mixed mode)
```

Установка NetBeans IDE

Для компиляции и выполнения программы из книги понадобится интегрированная среда разработки (IDE), например, NetBeans или Eclipse. На момент написания книги ни одна IDE не поддерживала систему модулей в полном объеме. В настоящее время NetBeans поддерживает создание одного модуля на проект. Следовательно, если вы хотите создать в NetBeans три Java-модуля, то нужно будет создать три проекта. При этом один модуль может ссылаться на другие, пользуясь зависимостями проекта. Актуальную информацию о поддержке модулей в NetBeans можно найти на вики-странице `http://wiki.netbeans.org/JDK9Support`.

Скачать самую свежую сборку NetBeans с поддержкой JDK 9 можно по ссылке `http://bits.netbeans.org/download/trunk/nightly/latest/`.

На этой странице можно скачать файлы разных типов. Самый маленький «весит» меньше 100 МБ и содержит Java SE. Он годится во всей книге, кроме главы 14, где рассматривается веб-приложение. Для выполнения примеров из главы 14 понадобится версия NetBeans для Java EE.

NetBeans работает поверх JDK версий 8 и 9. Я рекомендую устанавливать JDK 9 раньше NetBeans. Если на машине установлено несколько версий JDK, то NetBeans по умолчанию выбирает JDK 8. Чтобы использовался JDK 9, его установочный каталог нужно указать явно. Для случая, когда NetBeans устанавливался поверх JDK 8, в главе 3 имеются пошаговые инструкции, как использовать JDK 9 в качестве платформы Java в NetBeans. Там, где нужно, я буду приводить снимки окон NetBeans, чтобы помочь вам в создании программ и настройке IDE для их компиляции и выполнения.

Скачивание исходного кода

Исходный код примеров можно скачать по адресу `http://www.apress.com/us/book/9781484225912`. Это ZIP-файл, в котором корневой каталог называется `Java9Revealed`. Пользователям Windows я рекомендую распаковать архив в каталог `C:\`. В результате будет создан каталог `C:\Java9Revealed`, и в нем будет находиться весь исходный код.

Во всех примерах предполагается, что исходный код находится в каталоге `C:\Java9Revealed`. Если вы работаете не в Windows или скопировали исходный код не в каталог `C:\`, то в коде нужно будет заменить путь к каталогу с учетом платформенных соглашений о записи пути в файловой системе. Так, в Linux вместо `C:\Java9Revealed` нужно будет указать путь вида `/usr/ks/Java9Revealed`.

Каждый модуль в исходном коде представляет собой отдельный проект NetBeans и размещается в отдельном каталоге. Иногда в примерах используется несколько модулей, тогда будет и несколько каталогов. Так, в главе 5 создается пять модулей: `com.jdojo.prime`, `com.jdojo.prime.generic`, `com.jdojo.prime.faster`, `com.jdojo.prime.probable` и `com.jdojo.prime.client`. Соответственно в исходном коде имеется пять каталогов:

- `Java9Revealed/com.jdojo.prime`
- `Java9Revealed/com.jdojo.prime.generic`
- `Java9Revealed/com.jdojo.prime.faster`
- `Java9Revealed/com.jdojo.prime.probable`
- `Java9Revealed/com.jdojo.prime.client`

Каждый каталог с проектом NetBeans содержит подкаталоги `build`, `dist` и `src`. В подкаталоге `src` находится исходный код модуля, в `build` – его откомпилированный код, в `dist` – JAR-файл модуля. Примеры в этой книге ссылаются на подкаталоги `build` и `dist`. Если примеры работают неправильно, то, возможно, все наладится, если открыть проект в NetBeans, выполнить команду **Clean and Build** (Очистить и собрать), которая очищает подкаталоги `build` и `dist`, перекомпилирует проект в каталоге `build` и заново создает JAR-файл в каталоге `dist`. Рассмотрим следующую команду:

```
C:\Java9Revealed> java --module-path com.jdojo.intro\build\classes
 --module com.jdojo.intro/com.jdojo.intro.Welcome
```

Здесь предполагается, что `C:\Java9Revealed` – текущий каталог, т. е. относительному пути `com.jdojo.intro\build\classes` соответствует полный путь `C:\Java9Revealed\com.jdojo.intro\build\classes`. Встретив подобные команды в книге, заменяйте относительный путь в соответствии с тем, какой каталог является текущим и где находится исходный код.

Есть также возможность выполнять классы прямо в NetBeans, что гораздо проще, чем из командной строки. Некоторые примеры строятся в книге пошагово, и я показываю, что получается на каждом шаге. В таких случаях в исходном коде приводится окончательный вариант программы, а чтобы воспроизвести результаты на каждом шаге, вам придется отредактировать код.

Очень может статься, что формат файла откомпилированного класса немного изменится к тому моменту, когда вы будете их использовать. Я настоятельно рекомендую открывать каждый проект в NetBeans IDE и выполнять команду **Clean and Build**, прежде чем пытаться их запустить.

Глава 2

Система модулей

Краткое содержание главы:

- как производилось написание, упаковка и развертывание кода на Java до выхода JDK 9 и какие при этом возникали проблемы;
- что такое модуль в JDK 9;
- как объявляются модули и их зависимости;
- как упаковываются модули;
- что такое путь к модулю;
- что такое видимые модули;
- как напечатать список видимых модулей;
- как напечатать описание модуля.

Жизнь до Java 9

До выхода JDK 9 разработка Java-приложения состояла из следующих шагов.

- Написать код на Java, состоящий из различных типов: классов, интерфейсов, перечислений, аннотаций и т. п.
- Распределить типы по пакетам. Каждый тип принадлежит некоторому пакету – явно или неявно. Пакет – это логический контейнер типов, по существу он является пространством имен для находящихся в нем типов. В пакете могут быть открытые и закрытые типы, а также типы, которые, хотя и объявлены открытыми, на самом деле содержат внутренние реализации.
- Упаковать откомпилированный код типов в один или несколько JAR-файлов. Код, принадлежащий одном пакету, может размещаться в нескольких JAR-файлах.
- В приложении могут использоваться библиотеки, которые также поставляются в виде одного или нескольких JAR-файлов.
- Произвести развертывание приложения, т. е. копирование всех JAR-файлов, прикладных и библиотечных, в каталоги, перечисленные в пути к классам.

На рис. 2.1 показана типичная организация кода, упакованного в один JAR-файл. Представлены только пакеты и типы, а прочие артефакты, в частности файл `manifest.mf` и файлы ресурсов, опущены.

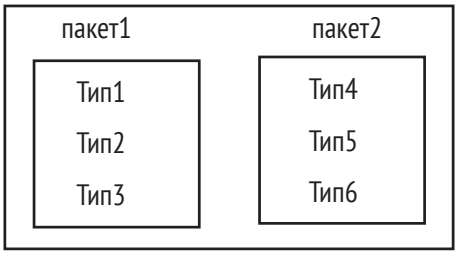

Рис. 2.1. Организация кода в JAR-файле

Свыше 20 лет сообщество Java разрабатывало код именно так. Но эта жизнь не была такой безоблачной, как хотелось бы! Перечислим проблемы, которые возникали при таком способе организации и выполнения кода.

- Пакет – это всего лишь контейнер типов, он не определяет границы доступности. Открытый тип, находящийся в некотором пакете, доступен всем остальным пакетам, не существует никакого способа ограничить глобальную видимость открытых типов.
- Все пакеты, кроме тех, чьи имена начинаются с java и javax, открыты для расширения. Если в вашем JAR-файле имеются типы с пакетным уровнем доступа, то они будут видны в других JAR-файлах, где определены типы, принадлежащие пакету с таким же именем, как ваш.
- Среда выполнения Java видит плоский набор пакетов, загруженных из списка JAR-файлов. Нет никакого способа узнать, существуют ли разные экземпляры одного и того же типа в разных JAR-файлах. Среда выполнения загрузит тип из того файла, который указан в пути к классам первым.
- Если какой-нибудь JAR-файл не включен в путь к классам, то среда выполнения выдаст сообщение об отсутствующих типах, причем это произойдет не на этапе компиляции, а на этапе выполнения, когда программа попытается обратиться к такому типу.
- На этапе инициализации приложение никак не может узнать, что некоторые используемые типы отсутствуют. Кроме того, можно включить неправильную версию JAR-файла, что закончится ошибкой на этапе выполнения.

Эти проблемы встречаются так часто и пользуются такой печальной известностью, что получили в сообществе Java специальное название – *ад JAR*. Я попадал в этот ад несколько раз на протяжении своей карьеры! Термину «ад JAR» даже посвящен специальный раздел статьи в википедии – см. https://en.wikipedia.org/wiki/Java_Classloader#JAR_hell.

Упаковка JDK и JRE тоже составляет проблему. Они поставлялись в виде гигантских монолитов, что увеличивает время загрузки, время инициализации и объем потребляемой памяти. Из-за монолитного JRE Java-приложения невозможно использовать на устройствах с небольшим объемом памяти. Если Java-приложение развертывается в облаке, то приходится платить за потребляемую память. Чаще всего монолитный JRE занимает больше памяти, чем необходимо, а это значит, что вы платите лишнее за предоставление облачных служб. Компактные профили, появившиеся в Java 8, – шаг в направлении уменьшения размера JRE, а, значит, и памяти, потребляемой средой выполнения. Это достигается путем упаковки

подмножества JRE в специальный образ среды выполнения, называемый *компактным профилем*.

> **Совет.** В ознакомительных версиях JDK 9 было три модуля: `java.compact1`, `java.compact2` и `java.compact3`, соответствующих трем компактным профилям JDK 8. Впоследствии от них отказались, потому что модули в JDK дают полный контроль над списком модулей, включаемых в пользовательский JRE.

Проблемы, мучившие JDK/JRE, до появления JDK 9 можно отнести к трем категориям:

- ненадежная конфигурация;
- слабая инкапсуляция;
- монолитная структура JDK/JRE.

На рис. 2.2 показано, как среда исполнения Java видит JAR-файлы на пути к классам и как к коду, находящемуся в одном JAR-файле, можно обратиться из других JAR-файлов без каких-либо ограничений, кроме заданных в объявлениях типов в терминах контроля доступа.

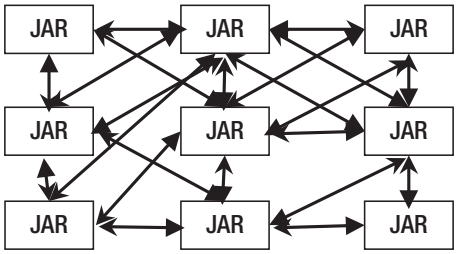

Рис. 2.2. JAR-файлы на пути к классам: загрузка средой выполнения и доступ

В Java 9 эти проблемы решаются посредством нового способа разработки, упаковки и развертывания приложений. Приложение в Java 9 состоит из небольших взаимодействующих компонентов, называемых *модулями*. JDK/JRE в Java 9 тоже организованы в виде набора модулей. В этой главе мы познакомимся с модулями в общих чертах, а в последующих рассмотрим их подробно.

Новая система модулей

В Java 9 введен новый компонент программы – *модуль*. Теперь приложение можно рассматривать как набор взаимодействующих модулей с четко определенными границами и зависимостями. При разработке системы модулей ставились следующие цели:

- надежная конфигурация;
- строгая инкапсуляция;
- модульная организация JDK/JRE.

Надежная конфигурация решает проблему провоцирующего ошибки механизма пути к классам, который используется для поиска типов. Модуль обязан явно объявить зависимости от других модулей. Система модулей проверяет зависимос-

ти на всех этапах разработки приложения: компиляции, компоновки и выполнения. Если один модуль объявил зависимость от другого, а этот другой модуль отсутствует на этапе инициализации приложения, то JVM обнаружит отсутствие зависимости и откажется запускать приложение. До Java 9 ошибка возникла бы на этапе выполнения (а не инициализации) – при попытке воспользоваться отсутствующими типами.

Строгая инкапсуляция решает проблему беспрепятственного доступа к открытым типам через границы JAR-файлов на пути к классам. Модуль должен явно объявить, какие из находящихся в нем открытых типов доступны другим модулям, ко всем остальным открытым типам обратиться из других модулей невозможно. В Java 9 слово `public` еще не означает, что тип доступен из любой части программы. Система модулей уточняет контроль доступности.

> **Совет.** В Java 9 надежная конфигурация достигается за счет того, что модуль явно объявляет зависимости, а система проверяет их на всех этапах разработки. Для обеспечения строгой инкапсуляции модуль объявляет, в каких пакетах могут находиться открытые типы, доступные другим модулям.

JDK 9 переписан и вместо монолитной структуры представляет собой набор так называемых *платформенных модулей*. Введен факультативный *этап компоновки*, который может находиться между этапами компиляции и выполнения. На этом этапе используется компоновщик `jlink`, входящий в состав JDK 9, который создает пользовательский образ среды исполнения, включающий только модули, используемые в приложении. В результате оптимизируется размер среды исполнения.

Что такое модуль?

Модуль – это именованная коллекция кода и данных. Он может содержать Java-код и платформенный код. Java-код организован в виде набора пакетов, содержащих типы: классы, интерфейсы, перечисления и аннотации. В состав данных могут входить ресурсы, например, графические и конфигурационные файлы.

С точки зрения Java-кода, модуль предстает в виде набора, содержащего нуль или более пакетов. На рис. 2.3 показано три модуля: `policy`, `claim` и `utility`, причем `policy` содержит два пакета, `claim` – один пакет, а `utility` – ни одного.

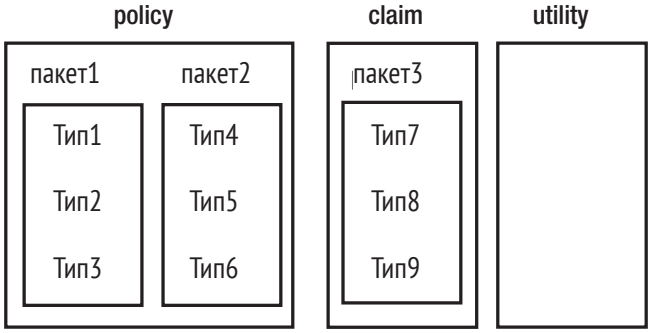

Рис. 2.3. Типы, пакеты и модули

Модуль – не просто контейнер пакетов. Помимо имени, определение модуля содержит:

- список *требуемых* модулей (от которых зависит данный);
- список *экспортируемых* пакетов (открытый API), которые могут использоваться другими модулями;
- список *раскрываемых* (с полным API, открытым и закрытым) пакетов, доступных другим модулям для рефлексии;
- список *используемых* (обнаруженных и загруженных с помощью класса `java.util.ServiceLoader`) служб;
- список предоставляемых реализаций служб.

Работа с модулем сводится к работе с одним или несколькими вышеперечисленными аспектами.

В спецификации Java SE 9 платформа представлена в виде набора *платформенных модулей*. Конкретная реализация Java SE 9 может содержать все или некоторые платформенные модули, что обеспечивает масштабируемость среды исполнения Java. Имена стандартных модулей начинаются словом `java`, например: `java.base`, `java.sql`, `java.xml`, `java.logging`. API стандартных платформенных модулей официально поддерживаются и предназначены для разработчиков.

Нестандартные платформенные модули являются частью JDK, но не включены в спецификацию платформы Java SE 9. Имена таких модулей начинаются словом `jdk`, например: `jdk.charsets`, `jdk.compiler`, `jdk.jlink`, `jdk.policytool`, `jdk.zipfs`. API, определенные в JDK-зависимых модулях, не предназначены для обычных разработчиков, а используются самим JDK и разработчиками библиотек, которые не могут добраться до нужной функциональности с помощью Java SE API. API, определенные в этих модулях, могут не поддерживаться или изменяться без предупреждения.

JavaFX не является частью спецификации платформы Java SE 9, но соответствующие модули устанавливаются вместе с JDK/JRE. Имена таких модулей начинаются словом `javafx`, например: `javafx.base`, `javafx.controls`, `javafx.fxml`, `javafx.graphics`, `javafx.web`.

Модуль `java.base`, являющийся частью платформы Java SE 9, можно назвать «первородным». Он не зависит ни от каких модулей. Система модулей знает только о `java.base`, а все остальные модули обнаруживает по определенным в модулях зависимостям. Модуль `java.base` экспортирует пакеты Java SE: `java.lang`, `java.io`, `java.math`, `java.text`, `java.time`, `java.util` и т. д.

Зависимости модулей

Вплоть до версии JDK 8 открытые типы, определенные в одном пакете, были доступны другим пакетам без ограничений, т. е. пакеты не контролировали доступность находящихся в них типов. Система модулей в JDK 9 обеспечивает более точный контроль доступности типов.

Доступность – это двустороннее соглашение между используемым и использующим модулем. Модуль может явно сделать свои открытые типы доступными другим модулям, а модуль, использующий типы, определенные в другом модуле, явно объявляет о зависимости от него. Все неэкспортируемые пакеты являются частной собственностью модуля и не могут использоваться вне него.