1 Знакомство с Kotlin

В этой главе мы рассмотрим основы синтаксиса Kotlin. Ведь прежде, чем переходить к реализации паттернов проектирования, необходимо получить представление о самом этом языке.

Кроме того, мы кратко обсудим проблемы, решить которые призваны паттерны проектирования, и выясним, почему их следует использовать в Kotlin. Это обсуждение будет полезно для тех читателей, которые не очень хорошо знакомы с концепцией паттернов проектирования. Однако даже опытные инженеры смогут почерпнуть из этой дискуссии пользу.

Важно отметить, что в этой главе будут описаны далеко не все аспекты языка. Ее цель — познакомить вас с фундаментальными понятиями и идиомами. В следующих главах будет рассказано о дополнительных функциях языка по мере того, как они будут становиться актуальными в свете обсуждаемых паттернов проектирования.

Технические требования

Чтобы следовать инструкциям, приведенным в этой главе, вам понадобится следующее:

- IntelliJ IDEA Community Edition (https://www.jetbrains.com/idea/download/);
- OpenJDK версии 19 или выше (https://jdk.java.net/19/).

Файлы с кодом примеров главы доступны по адресу https://github.com/PacktPublishing/Kotlin-Design-Patterns-and-Best-Practices_Third-Edition/tree/main/Chapter01.

ПРИМЕЧАНИЕ

Нет необходимости записывать в файл простые фрагменты кода. Вы можете изучать язык онлайн с помощью таких платформ, как https://play.kotlinlang.org/, или воспользоваться REPL и интерактивной оболочкой, установив Kotlin и выполнив команду kotlinc.

Основной синтаксис и возможности языка Kotlin

Если вы уже работали с Java, C#, Scala или другими подобными языками программирования, то синтаксис Kotlin покажется вам знакомым. Это вполне естественно, учитывая, что Kotlin ориентирован на тех пользователей, которые уже освоили другие языки. Помимо решения реальных проблем с помощью таких функций, как улучшенная типобезопасность, Kotlin устраняет недостатки, присущие другим языкам, в частности печально известную проблему с *ошибкой обращения к null-ссылке* (Null Pointer Exception, NPE) в Java или отсутствие функций верхнего уровня. В данном языке сохраняется практический подход, который последовательно реализуется во всех его аспектах.

Одно из самых больших преимуществ Kotlin заключается в его способности органично сочетаться с Java. В одном проекте можно использовать как классы Java, так и классы Kotlin, а также свободно применять любую библиотеку Java. Однако стоит отметить, что такая совместимость сопряжена с трудностями, которые проявляются, например, при работе с типами, допускающими значение `null`. Поэтому, несмотря на всю простоту интеграции, устранение этих нюансов потребует некоторых усилий.

В целом Kotlin помогает достигать следующих целей.

- *Прагматичность* — упрощает выполнение обычных задач.
- *Удобочитаемость* — позволяет найти баланс между краткостью и понятностью кода.
- *Простота повторного использования* — помогает адаптировать код к различным сценариям.
- *Безопасность* — препятствует использованию практик написания кода, чреватых возникновением ошибок.
- *Совместимость* — позволяет применять существующие библиотеки и фреймворки, уже популярные среди Java-разработчиков, в частности Spring Framework, Hibernate и jOOQ.

В этой главе мы поговорим о том, как с помощью Kotlin можно достичь всех этих целей.

Kotlin — мультипарадигменный язык

Основные парадигмы программирования — процедурная, объектно-ориентированная и функциональная.

Язык Kotlin, опирающийся на прагматичный подход, учитывает их все. Он не заставляет вас придерживаться какой-то одной парадигмы, как это делают некоторые другие языки.

Kotlin поддерживает классы и наследование, свойственные объектно-ориентированной парадигме, а также функции высшего порядка, присущие функциональному программированию. Однако Kotlin не принуждает вас инкапсулировать все внутри классов. При желании вы можете структурировать свой код, используя исключительно процедуры и структуры. Изучая примеры, приведенные в этой книге, вы увидите, как сочетание различных парадигм позволяет решать те или иные проблемы.

Структура кода, написанного на Kotlin

Когда вы начинаете программировать на Kotlin, вашим первым шагом обычно является создание нового файла, который, как правило, имеет расширение `.kt`.

В отличие от Java, Kotlin не предусматривает строгой зависимости между именем файла и именем класса. Вы можете добавить несколько публичных классов в один файл с кодом на языке Kotlin, однако в целом рекомендуется объединять в файле только логически связанные классы. Помните, что файл при этом не должен оказаться чрезмерно длинным или трудночитаемым. Кроме того, упаковка нескольких публичных классов в один файл может усложнить поиск конкретной функциональности, так как в обзоре проекта могут отображаться не все доступные классы.

Соглашения об именовании

Если ваш файл состоит из одного класса, то рекомендуется выбрать для него имя, совпадающее с названием этого класса.

Если ваш файл содержит несколько классов, то имя файла должно описывать их общую цель или тему. В соответствии с правилами написания кода на Kotlin при именовании файлов рекомендуется использовать *UpperCamelCase* (https://kotlinlang.org/docs/coding-conventions.html).

Главный файл проекта Kotlin обычно имеет название `Main.kt`. Он, как правило, служит точкой входа в приложение.

Пакеты

Пакет в Kotlin — это коллекция файлов и классов, которые имеют общую цель или область применения. Пакеты предоставляют удобный способ организации классов и функций под единым пространством имен и часто хранятся в одной папке. Эта концепция широко применяется в Kotlin, а также во многих других языках программирования.

Для объявления пакета, к которому относится файл, используется ключевое слово `package`:

```
package me.soshin
```

При работе с сочетанием файлов, написанных на языках Java и Kotlin, важно убедиться в том, что файлы Kotlin соответствуют правилам именования пакетов Java, которые приведены на сайте https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html. Язык Kotlin позволяет гибко организовывать пакеты в каталогах и файлах, однако в целях обеспечения согласованности и совместимости настоятельно рекомендуется придерживаться правил именования пакетов Java.

Кроме того, многие интегрированные среды разработки (IDE) отображают предупреждения или предложения поместить файл в правильный каталог в соответствии с объявлением пакета. Эта функция может быть особенно полезна для обеспечения правильной организации при одновременной работе с файлами Kotlin и Java.

Если ваш проект состоит исключительно из файлов Kotlin, то вы можете не указывать префиксы общих пакетов в структуре папок. Например, если все ваши проекты Kotlin находятся в пакете `me.soshin`, а конкретная часть вашего приложения связана с ипотечными кредитами, то вы можете поместить свои файлы непосредственно в папку `/mortgages` вместо того, чтобы использовать вложенную структуру `me/soshin/mortgages`, как требуется в проектах Java (табл. 1.1).

Таблица 1.1. В случае с файлом Main.kt в явном объявлении пакета нет необходимости

Java	Kotlin
me └── soshin └── mortgages └── Main.java	mortgages └── Main.kt

ПРИМЕЧАНИЕ

С этого момента многоточие будет использоваться для обозначения опущенных фрагментов кода. Однако вы всегда можете получить доступ к полным версиям примеров кода, перейдя по соответствующей ссылке на GitHub.

Комментарии

Далее в книге части кода будут документироваться с помощью комментариев Kotlin. Как и во многих других языках программирования, для создания однострочных комментариев в Kotlin используются символы `//`, а для многострочных — `/* */`.

Комментарии позволяют предоставить дополнительную информацию коллегам-разработчикам и вам самим в будущем. Учитывая это, напишем первую программу на Kotlin и посмотрим, как в ней применяются основные принципы этого языка.

ПРИМЕЧАНИЕ

Примеры на Java приведены исключительно для ознакомления, а не для того, чтобы доказать превосходство Kotlin.

Hello Kotlin

Каждый учебник по языку программирования обычно содержит описание программы `Hello World`, и в данной книге я буду придерживаться этой традиции.

Начнем изучение принципов работы Kotlin с того, что поместим в файл `Main.kt` следующий код и запустим его:

```
fun main() {
    println("Hello Kotlin")
}
```

Когда вы запустите этот код, например нажав кнопку Run (Выполнить) в среде IntelliJ IDEA, вы увидите следующий результат:

```
> Hello Kotlin
```

Этот код имеет несколько заметных отличий от эквивалентного кода на языке Java, предназначенного для достижения аналогичного результата:

```
class Main {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

Далее мы рассмотрим эти отличительные особенности более подробно.

ПРИМЕЧАНИЕ

Многие примеры в этой книге предполагают, что приведенный код находится внутри основной функции. Если вы не видите сигнатуры функции main, значит, код, скорее всего, является частью основной функции. Кроме того, вы можете запускать примеры в scratch-файле IntelliJ, что является весьма удобным способом выполнения фрагментов кода.

Отсутствие класса-обертки

В таких языках, как Java, C#, Scala и многие другие, сделать функцию исполняемой можно, только инкапсулировав ее в класс.

Однако в Kotlin появилось понятие функций уровня пакета. Если функция не требует доступа к свойствам класса, то оборачивать ее в класс нет необходимости. Ее можно определить непосредственно на уровне пакета.

В последующих главах мы более подробно рассмотрим функции уровня пакета, в том числе способы использования и преимущества.

Отсутствие аргументов

Для настройки приложений командной строки аргументы предоставляются в виде массива строк. В Java функция `main()` должна была принимать этот массив аргументов, по крайней мере до выхода версии Java 21:

```
public static void main(String[] args) { ... }
```

Однако в Kotlin этот массив аргументов является необязательным. Вы можете определить функцию `main()` вместе с ним:

```
fun main(args: Array<String>) { ... }
```

А можете сделать это без него:

```
fun main() { ... }
```

Такая гибкость Kotlin позволяет создать функцию `main()`, не требующую никаких аргументов.

Отсутствие модификатора static

В некоторых языках ключевое слово `static` используется для обозначения функций внутри класса, которые можно вызывать, не создавая экземпляр этого класса. Классический пример — функция `main()`.

Kotlin же предоставляет гораздо больше возможностей. Если функция не зависит от внутреннего состояния или свойств класса, то вы можете определить ее за пределами любого класса. Начиная с версии Kotlin 1.0 и по крайней мере до выхода версии Kotlin 2.0 ключевое слово `static` к функциям не применялось.

Стоит отметить, что, хотя такой подход и повышает удобство повторного использования кода и его модульность, у него есть недостатки. Например, если вы отделите функцию от логически связанного с ней класса, то процесс отслеживания структуры кода или поддержания инкапсуляции может усложниться.

Более лаконичная функция print

В качестве альтернативы громоздкому методу `System.out.println()` языка Java, который отправляет строку в стандартный поток вывода, Kotlin предлагает удобный псевдоним `println()`, который функционирует точно так же, как и `System.out.println()`, позволяя отправлять строки в стандартный поток вывода более лаконичным образом. Важно отметить, что функция `println()` является встроенной (inline), то есть в сгенерированном коде заменяется своим содержимым (`System.out.println()`). Благодаря такому поведению она и называется *псевдонимом* (alias).

Отсутствие точек с запятой

Во многих языках, в частности в Java, каждый оператор или выражение должны завершаться точкой с запятой, например:

```
System.out.println("Semicolon =>");
```

Однако Kotlin использует прагматичный подход. Во время компиляции язык автоматически определяет, где должна находиться точка с запятой, избавляя вас от необходимости явно ее указывать, например:

```
println("No semicolons! =>")
```

В коде Kotlin вам чаще всего не придется использовать точки с запятой, поскольку в этом языке, делающем акцент на прагматизме и лаконичности, они считаются необязательными. Благодаря подобной гибкости вы можете устранять беспорядок в коде, сосредотачиваясь на его существенных аспектах, и повышать его читабельность и удобство сопровождения.

Однако, как и в случае со многими другими преимуществами, у подобной гибкости могут быть и отрицательные стороны. Да, она упрощает процесс написания кода, особенно для опытных разработчиков, однако очень важно принимать во внимание его удобочитаемость как для вас самих, так и для других людей, которые могут работать с вашим кодом. Гибкость Kotlin позволяет создавать лаконичный код, и тем не менее при его написании следует придерживаться лучших практик и стандартов. Например, несмотря на то, что точка с запятой не является обязательной, ее использование в определенных ситуациях может повысить удобочитаемость кода, особенно при наличии нескольких инструкций[1] в одной строке или в объявлениях перечислений.

Таким образом, язык Kotlin обеспечивает гибкость, позволяющую писать код максимально эффективно, однако при этом очень важно делать его четким и понятным, придерживаясь лучших отраслевых практик.

---

[1] В языке Kotlin слово statement означает не оператор, а инструкцию. — *Примеч. ред.*

Система типов Kotlin

Ранее мы говорили о том, что язык Kotlin является типобезопасным. Теперь рассмотрим его систему типов и сравним ее с тем, что предлагает Java.

Основные типы

В некоторых языках существует различие между примитивными типами и объектами. Например, в Java для примитивных значений используется тип `int`, а для объектов — `Integer`. Первый тип более эффективен с точки зрения использования памяти, а второй более выразителен благодаря поддержке значений `null` и дополнительных методов.

Однако в Kotlin нет такого различия между примитивами и объектами, как в Java. С точки зрения разработчика, все типы Kotlin рассматриваются одинаково, и в отличие от Java в Kotlin вы обычно не имеете дела с примитивами напрямую. При написании кода на Java вам часто приходится задумываться о том, с чем вы работаете: с примитивами или объектами.

Тем не менее данное различие не означает, что Kotlin в этом отношении менее эффективен, чем Java. Компилятор Kotlin оптимизирует типы в фоновом режиме, гарантируя сохранение высокой производительности. Поэтому, несмотря на то что в Kotlin вы можете не работать непосредственно с примитивами, вам не стоит беспокоиться об этом с точки зрения эффективности.

Большинство типов Kotlin имеют имена, схожие с их аналогами в Java. Исключениями являются тип `Int` (в Java `Integer`) и `Unit` (в Java `void`).

Перечисление всех типов заняло бы слишком много места, так что приведу лишь несколько примеров (табл. 1.2).

Таблица 1.2. Типы Kotlin

Семейство типов	Примеры типов	Примеры значений
Числа	`Int, Long, Double`	`42, 6_000_000L, 3.14`
Строки	`String`	`"C-3 PO"`
Логические значения	`Boolean`	`true, false`
Символы	`Char`	`z, \n, \u263A`

Вывод типов

Объявим первую переменную Kotlin, позаимствовав строку из примера `Hello Kotlin`:

```
fun main() {
    var greeting = "Hello Kotlin"
    println(greeting)
}
```