20 Совместимость с Java

На протяжении этой книги вы изучали фундаментальные основы языка программирования Kotlin, и мы надеемся, что у вас появилось желание использовать Kotlin для улучшения существующих Java-проектов. С чего начать?

Ранее вы уже видели, что Kotlin компилируется в байт-код Java. Это означает, что Kotlin *совместим* с Java. То есть он может работать вместе с этим кодом.

Это одна из самых главных возможностей языка программирования Kotlin. Полная совместимость с Java означает, что файлы Kotlin и файлы Java могут существовать и выполняться в одном проекте. Можно вызывать методы Java из Kotlin, а функции Kotlin — из Java. Это означает, что можно использовать существующие Java-библиотеки в Kotlin, включая фреймворк Android.

Полная совместимость с Java подразумевает, что можно постепенно переводить свой код из Java в Kotlin. Если у вас нет возможности переписать проект на языке Kotlin, рассмотрите возможность разработки будущих проектов на Kotlin. Возможно, вы захотите перевести только те Java-файлы, которые получат наибольшее преимущество от перехода на Kotlin. В этом случае рассмотрите возможность перевода предметных объектов или модульных тестов.

Эта глава расскажет о том, как совмещаются файлы Java и Kotlin, а также о чем стоит помнить при написании кода, чтобы он был совместимым.

Совместимость с классом Java

В этой главе вы создадите в IntelliJ новый проект с именем Interop. Interop будет содержать два файла: Hero.kt (файл с кодом на Kotlin, который представляет героя из игры NyetHack) и Jhava.java (класс Java, который представляет монстра из другого измерения). Создайте эти два файла.

В этой главе вы напишете код на двух языках — Kotlin и Java. Даже если у вас нет опыта в Java, не бойтесь, потому что примеры на Java будут вам понятны, учитывая уже имеющийся опыт работы с Kotlin.

Начнем с объявления класса `Jhava` и его метода с именем `utterGreeting`, который возвращает `String`.

Листинг 20.1. Объявление класса и метода в Java (Jhava.java)

```
public class Jhava {
    public String utterGreeting() {
        return "BLARGH";
    }
}
```

Теперь создадим функцию `main` в Hero.kt. Внутри нее объявите переменную монстра `val adversary`, экземпляр `Jhava`.

Листинг 20.2. Объявление функции main и переменной adversary (Hero.kt)

```
fun main(args: Array<String>) {
    val adversary = Jhava()
}
```

Свершилось! Написав эти строки на Kotlin, вы создали объект Java и сократили дистанцию между двумя языками. Как видите, вызвать код на Java из Kotlin очень просто.

Но нам есть что еще показать. Для проверки выведите приветственный рык монстра `Jhava`.

Листинг 20.3. Вызов метода Java из Kotlin (Hero.kt)

```
fun main(args: Array<String>) {
    val adversary = Jhava()
    println(adversary.utterGreeting())
}
```

Вы создали объект Java, вызвали его метод и сделали все это из Kotlin. Запустите Hero.kt. Вы увидите, как монстр приветствует героя (`BLARGH`!) в консоли.

Kotlin создавался с прицелом на бесшовную совместимость с Java. Также он включает ряд усовершенствований, недоступных в Java. Придется ли вам отказаться от всех усовершенствований, организуя взаимодействия между этими языками? Конечно нет! Зная о различиях между этими двумя языками и ис-

пользуя аннотации, доступные на обеих сторонах, вы сможете взять все лучшее, что предлагает Kotlin.

Совместимость и null

Добавим еще один метод в `Jhava` с именем `determineFriendshiplevel` (уровень дружелюбности). `determineFriendshiplevel` должен возвращать значение типа `String`, но, так как монстру чуждо понятие дружелюбия, он возвращает null.

Листинг 20.4. Возвращение null из метода Java (Jhava.java)

```
public class Jhava {
    public String utterGreeting() {
        return "BLARGH";
    }

    public String determineFriendshipLevel() {
        return null;
    }
}
```

Вызовите этот новый метод из Hero.kt, сохранив значение дружелюбности монстра в `val`. Выведите это значение в консоль, но так как громкий рык монстра выводился прописными буквами, уровень дружелюбности мы выведем строчными буквами.

Листинг 20.5. Вывод уровня дружелюбности (Hero.kt)

```
fun main(args: Array<String>) {
    val adversary = Jhava()
    println(adversary.utterGreeting())

    val friendshipLevel = adversary.determineFriendshipLevel()
    println(friendshipLevel.toLowerCase())
}
```

Запустите Hero.kt. Хотя компилятор и не сообщил об ошибках, после запуска программа тут же завершится с ошибкой:

```
    Exception in thread "main"
    java.lang.IllegalStateException: friendshipLevel must not be null
```

В главе 6 мы уже говорили о том, что в Java все объекты могут быть null. Вызывая Java-метод, такой как `determineFriendshiplevel`, мы видим, что он возвра-

щает `String`, но это не значит, что возвращаемое значение будет соответствовать правилам языка Kotlin в отношении null.

Так как все объекты в Java могут быть null, то безопаснее предположить, что все значения имеют тип с поддержкой null, если явно не указано обратное. Следование этому правилу обеспечит безопасность, но код получится более многословным, так как вам придется обрабатывать каждый возможный null для каждой переменной Java, на которую вы ссылаетесь.

Удерживая клавишу Ctrl (Command), щелкните левой кнопкой мыши на `determineFriendshiplevel` в Hero.kt. IntelliJ сообщит, что метод возвращает значение `String!`. Восклицательный знак говорит о том, что возвращаемое значение может быть `String` или `String?`. Компилятор Kotlin не знает, какое значение будет получено из Java, — строка или null.

Этот неопределенный возвращаемый тип называется *платформенным типом*. Платформенные типы не имеют синтаксического представления: их можно видеть только в IDE или другой документации.

К счастью, программисты на Java могут писать дружелюбный к Kotlin код, явно сообщая о поддержке null с помощью соответствующих аннотаций. Объявим явно, что `determineFriendshiplevel` может возвращать значение null, добавив аннотацию `@Nullable` в заголовок метода.

Листинг 20.6. Указание, что возвращаемое значение может быть null (Jhava.java)

```
public class Jhava {
    public String utterGreeting() {
        return "BLARGH";
    }

    @Nullable
    public String determineFriendshipLevel() {
        return null;
    }
}
```

(Вам надо будет импортировать `org.jetbrains.annotations.Nullable`, который предложит вам IntelliJ.)

`@Nullable` предупреждает пользователя этого API, что метод может (а не *обязан*) вернуть null. Компилятор Kotlin понимает значение этой аннотации. Вернитесь в Hero.kt и обратите внимание, что теперь IntelliJ предупреждает вас о прямом вызове `toLowerCase` для `String?`.

Замените прямой вызов безопасным.

Листинг 20.7. Обработка null с помощью оператора безопасного вызова (Hero.kt)

```
fun main(args: Array<String>) {
    val adversary = Jhava()
    println(adversary.utterGreeting())

    val friendshipLevel = adversary.determineFriendshipLevel()
    println(friendshipLevel?.toLowerCase())
}
```

Запустите Hero.kt. Теперь значение `null` будет выведено в консоль.

Возможно, вы захотите представить отсутствие дружелюбия каким-то другим значением, отличным от null. Используйте оператор `?:` (объединения по null), чтобы вернуть значение по умолчанию в тех случаях, когда `friendshipLevel` null.

Листинг 20.8. Подстановка значения по умолчанию с помощью оператора ?: (Hero.kt)

```
fun main(args: Array<String>) {
    val adversary = Jhava()
    println(adversary.utterGreeting())

    val friendshipLevel = adversary.determineFriendshipLevel()
    println(friendshipLevel?.toLowerCase() ?: "It's complicated.")
}
```

Запустив Hero.kt, вы увидите сообщение `It's complicated`.

Вы использовали `@Nullable`, чтобы сообщить, что метод может вернуть null. Сообщить, что значение точно не будет null, можно с помощью аннотации `@NotNull`. Эта замечательная аннотация избавляет пользователя API от беспокойства о том, что возвращаемое значение может быть null. Приветствие монстра `Jhava` не должно быть null, поэтому добавьте аннотацию `@NotNull` в заголовок метода `utterGreeting`.

Листинг 20.9. Указание, что возвращаемое значение не будет null (Jhava.java)

```
public class Jhava {

    @NotNull
    public String utterGreeting() {
        return "BLARGH";
    }
```

```
    @Nullable
    public String determineFriendshipLevel() {
        return null;
    }
}
```

(Вам снова понадобится импортировать аннотации.)

Аннотации `@Nullable` и `@NotNull` можно использовать для уточнения контекста возвращаемых значений, параметров и даже полей.

Kotlin обеспечивает разные инструменты для обработки null-значений, включая возможность запрещать обычным типам принимать значение null. Самый распространенный источник ошибок с null-значениями в Kotlin — это взаимодействия с кодом на Java, так что будьте аккуратны, вызывая код Java из Kotllin.

Соответствие типов

Типы в Kotlin и Java часто прямо соответствуют друг другу. `String` в Kotlin также остается `String`, когда компилируется в Java. Это значит, что `String`, возвращаемый методом Java, можно использовать в Kotlin, как если бы это значение было получено непосредственно в Kotlin.

Однако есть типы, не имеющие прямых аналогов. Например, базовые типы. Как уже обсуждалось в разделе главы 2 «Для любопытных: простые типы Java в Kotlin», Java представляет базовые типы данных через примитивы. Примитивы в Java — это не объекты, однако в Kotlin все типы — это объекты, включая базовые типы. Несмотря на это, компилятор Kotlin может отображать простые типы Java в наиболее простые типы в Kotlin.

Чтобы увидеть, как происходит такое преобразование, добавьте целочисленное значение `hitPoints` в `Jhava`. Целое число представлено как объект типа `Int` в Kotlin и как примитив `int` в Java.

Листинг 20.10. Объявление int в java (Jhava.java)

```
public class Jhava {

    public int hitPoints = 52489112;

    @NotNull
    public String utterGreeting() {
```

```
        return "BLARGH";
    }

    @Nullable
    public String determineFriendshipLevel() {
        return null;
    }
}
```

Теперь получим ссылку на `hitPoints` в Hero.kt.

Листинг 20.11. Обращение к полю Java из Kotlin (Hero.kt)

```
fun main(args: Array<String>) {
    val adversary = Jhava()
    println(adversary.utterGreeting())

    val friendshipLevel = adversary.determineFriendshipLevel()\
    println(friendshipLevel?.toLowerCase() ?: "It's complicated.")

    val adversaryHitPoints: Int = adversary.hitPoints
}
```

Несмотря на то что `hitPoints` объявлен в `Jhava` как `int`, у вас не возникло проблем при обращении к нему как к `Int`. (Мы не используем автоматическое определение типов в примере только для того, чтобы показать как происходит преобразование типов. Явные объявления типов не нужны для интеграции языков: объявление `val adversaryHitPoints = adversary.hitPoints` ничуть не хуже.)

Теперь, когда у вас есть ссылка на целое число, вы можете вызывать функции с ним. Вызовите функцию для `adversaryHitPoints` и выведите результат.

Листинг 20.12. Обращение к полю Java через Kotlin (Hero.kt)

```
fun main(args: Array<String>) {
    ...
    val adversaryHitPoints: Int = adversary.hitPoints
    println(adversaryHitPoints.dec())
}
```

Запустите Hero.kt для вывода очков здоровья противника, уменьшенных на 1.

В Java нельзя вызывать методы для простых типов. В Kotlin целое число `adversaryHitPoints` — это объект типа `Int`, и вы можете вызывать его функции.