Оглавление

Предисловие

Впервые оказавшись в JetBrains весной 2010 года, я был абсолютно уверен, что мир не нужен еще один язык программирования общего назначения. Я полагал, что существующие JVM-языки достаточно хороши, да и кто в здравом уме станет создавать новый язык? Примерно после часа обсуждения проблем разработки крупномасштабных программных продуктов мое мнение изменилось, и я набросал на доске первые идеи, которые позже стали частью языка Kotlin. Вскоре после этого я присоединился к JetBrains для проектирования языка и работы над компилятором.

Сегодня, спустя шесть лет, мы приближаемся к выпуску второй версии. Сейчас в команде работает более 30 человек, а у языка появились тысячи активных пользователей, и у нас осталось еще множество потрясающих идей, которые с трудом укладываются в моей голове. Но не волнуйтесь: прежде чем стать частью языка, эти идеи подвергнутся тщательной проверке. Мы хотим, чтобы в будущем описание языка Kotlin по-прежнему могло уместиться в одну не слишком большую книгу.

Изучение языка программирования – это захватывающее и часто очень полезное занятие. Если это ваш первый язык, с ним вы откроете целый новый мир программирования. Если нет, он заставит вас по-другому думать о знакомых вещах и глубже осознать их на более высоком уровне абстракции. Эта книга предназначена в основном для последней категории читателей, уже знакомых с Java.

Проектирование языка с нуля – это сама по себе сложная задача, но обеспечение взаимодействия с другим языком – это совсем другая история, полная злых огров и мрачных подземелий. (Если не верите мне – спросите Бьярне Страуструпа (Bjarne Stroustrup), создателя C++.) Совместимость с Java (то есть возможность смешивать код на Java и Kotlin и вызывать один из другого) стала одним из краеугольных камней Kotlin, и эта книга уделяет большое внимание данному аспекту. Взаимодействие с Java очень важно для постепенного внедрения Kotlin в существующие проекты на Java. Даже создавая проект с нуля, важно учитывать, как язык вписывается в общую картину платформы со всем многообразием библиотек, написанных на Java.

В данный момент, когда я пишу эти строки, разрабатываются две новые платформы: Kotlin уже запускается на виртуальных машинах JavaScript, обеспечивая поддержку разработки всех уровней веб-приложения, и скоро его можно будет компилировать прямо в машинный код и запускать без виртуальной машины. Хотя эта книга ориентирована на JVM, многое из того, что вы узнаете, может быть использовано в других средах выполнения.

Авторы присоединились к команде Kotlin с самых первых дней, поэтому они хорошо знакомы с языком и его внутренним устройством. Благодаря опыту выступления на конференциях и проведению семинаров и курсов о Kotlin авторы смогли сформировать хорошие объяснения, предвосхищающие самые распространенные вопросы и возможные подводные камни. Книга объясняет высокоуровневые понятия языка во всех необходимых подробностях.

Надеюсь, вы хорошо проведете время с книгой, изучая наш язык. Как я часто говорю в сообщениях нашего сообщества: «Хорошего Kotlin!»

Андрей Бреслав,
ведущий разработчик языка Kotlin в JetBrains

Вступление

Идея создания Kotlin зародилась в JetBrains в 2010 году. К тому времени компания уже была признанным производителем инструментов разработки для множества языков, включая Java, C#, JavaScript, Python, Ruby и PHP. IntelliJ IDEA – наш флагманский продукт, интегрированная среда разработки (IDE) для Java – также включала плагины для Groovy и Scala.

Опыт разработки инструментария для такого разнообразного набора языков позволил нам получить уникальное понимание процесса проектирования языков целом и взглянуть на него под другим углом. И все же интегрированные среды разработки на платформе IntelliJ, включая IntelliJ IDEA, по-прежнему разрабатывались на Java. Мы немного завидовали нашим коллегам из команды .NET, которые вели разработку на C# – современном, мощном, быстро развивающемся языке. Но у нас не было языка, который мы могли бы использовать вместо Java.

Какими должны быть требования к такому языку? Первое и самое очевидное – статическая типизация. Мы не знаем другого способа разрабатывать проекты с миллионами строк кода на протяжении многих лет, не сходя при этом с ума. Второе – полная совместимость с существующим кодом на Java. Этот код является чрезвычайно ценным активом компании JetBrains, и мы не могли себе позволить потерять или обесценить его из-за проблем совместимости. В-третьих, мы не хотели идти на компромиссы с точки зрения качества инструментария. Производительность разработчиков – самая главная ценность компании JetBrains, а для её достижения нужен хороший инструментарий. Наконец, нам был нужен язык, простой в изучении и применении.

Когда мы видим в своей компании неудовлетворенную потребность, мы знаем, что есть и другие компании, оказавшиеся в подобной ситуации, и что наше решение найдет много пользователей за пределами JetBrains. Учитывая это, мы решили начать проект разработки нового языка: Kotlin. Как это часто бывает, проект занял больше времени, чем мы ожидали, и Kotlin 1.0 вышел больше чем через пять лет после того, как в репозитории появился первый фрагмент его реализации; но теперь мы уверены, что язык нашел своих пользователей и будет использоваться в дальнейшем.

Язык Kotlin назван в честь острова Котлин неподалеку от Санкт-Петербурга в России, где живет большинство разработчиков Kotlin. Выбрав для названия языка имя острова, мы последовали прецеденту, созданному языками Java и Ceylon, но решили найти что-то ближе к нашему дому.

По мере приближения к выпуску первой версии языка мы поняли, что нам не помешала бы книга про Kotlin, написанная людьми, которые участвовали в принятии проектных решений и могут уверенно объяснить, почему Kotlin устроен так, а не иначе. Данная книга – результат совместных усилий этих людей, и мы надеемся, что она поможет вам узнать и понять язык Kotlin. Удачи вам, и программируйте с удовольствием!

Благодарности

Прежде всего мы хотим поблагодарить Сергея Дмитриева и Максима Шафирова за веру в идею нового языка и решение вложить средства JetBrains в его разработку. Без них не было бы ни языка, ни этой книги.

Мы хотели бы особо поблагодарить Андрея Бреслава – главного виновника, что язык спроектирован так, что писать про него одно удовольствие (впрочем, как и программировать на нем). Несмотря на занятость управлением постоянно растущей командой Kotlin, Андрей смог сделать много полезных замечаний, что мы очень высоко ценим. Более того, вы можете быть уверены, что книга получила одобрение от ведущего разработчика языка в виде предисловия, которое он любезно написал.

Мы благодарны команде издательства Manning, которая помогла нам написать книгу и сделать ее легко читаемой и хорошо структурированной. В частности, мы хотим поблагодарить редактора-консультанта Дэна Махари (Dan Maharry), который всегда стремился найти время для обсуждения, несмотря на наш напряженный график, а также Майкла Стивенса (Michael Stephens), Хелен Стергиус (Helen Stergius), Кевина Салливана (Kevin Sullivan), Тиффани Тейлор (Tiffany Taylor), Элизабет Мартин (Elizabeth Martin) и Марию Тюдор (Marija Tudor). Отзывы наших технических редакторов Брента Уотсона (Brent Watson) и Игоря Войды (Igor Wojda) оказались просто бесценны, так же как замечания рецензентов, читавших рукопись в процессе работы: Алессандро Кампеи (Alessandro Campeis), Амита Ламба (Amit Lamba), Анджело Косты (Angelo Costa), Бориса Василе (Boris Vasile), Брендана Грейнджера (Brendan Grainger), Кальвина Фернандеса (Calvin Fernandes), Кристофера Бейли (Christopher Bailey), Кристофера Борца (Christopher Bortz), Конора Редмонда (Conor Redmond), Дилана Скотта (Dylan Scott), Филипа Правика (Filip Pravica), Джейсона Ли (Jason Lee), Джастина Ли (Justin Lee), Кевина Орра (Kevin Orr), Николаса Франкеля (Nicolas Frankel), Павла Гайды (Paweł Gajda), Рональда Тишлера (Ronald Tischliar) и Тима Лаверса (Tim Laver).

Также благодарим всех, кто отправил свои предложения в рамках MEAP (Manning Early Access Program – программы раннего доступа Manning) на форуме книги; ваши комментарии помогли улучшить текст книги.

Мы благодарны всем участникам команды Kotlin, которым приходилось выслушивать ежедневные заявления вроде: «Еще один раздел закончен!» – на протяжении всего периода написания этой книги. Мы хотим поблагодарить наших коллег, которые помогли составить план книги и оставляли отзывы о её ранних вариантах, особенно Илью Рыженкова, Хади Харири (Hadi Hariri), Михаила Глухих и Илью Горбунова. Мы также хотим поблагодарить друзей, которые не только поддерживали нас, но и читали текст книги и оставляли отзывы о ней (иногда на горнолыжных курортах во время отпуска): Льва Серебрякова, Павла Николаева и Алису Афонину.

Наконец, мы хотели бы поблагодарить наши семьи и котов за то, что они делают этот мир лучше.

Об этой книге

Книга «*Kotlin в действии*» расскажет о языке Kotlin и как писать на нем приложения для виртуальной машины Java и Android. Она начинается с обзора основных особенностей языка Kotlin, постепенно раскрывая наиболее отличительные аспекты, такие как поддержка создания высокоуровневых абстракций и предметно-ориентированных языков (Domain-Specific Languages, DSL). Книга уделяет большое внимание интеграции Kotlin с существующими проектами на языке Java и поможет вам внедрить Kotlin в текущую рабочую среду.

Книга описывает версию языка Kotlin 1.0. Версия Kotlin 1.1 разрабатывалась параллельно с написанием книги, и, когда это было возможно, мы упоминали об изменениях в версии 1.1. Но поскольку на момент написания книги новая версия еще не была готова, мы не могли полностью охватить все нововведения. За более подробной информацией о новых возможностях и изменениях обращайтесь к документации по адресу: `https://kotlinlang.org`.

Кому адресована эта книга

Книга «*Kotlin в действии*» адресована в первую очередь разработчикам с опытом программирования на языке Java. Kotlin во многом основан на понятиях и приёмах языка Java, и с помощью этой книги вы быстро освоите его, используя имеющиеся знания. Если вы только начали изучать Java или владеете другими языками программирования, такими как C# или JavaScript, вам может понадобиться обратиться к другим источникам информации, чтобы понять наиболее сложные аспекты взаимодействия Kotlin с JVM, но вы все равно сможете изучить Kotlin, читая эту книгу. Мы описываем язык Kotlin в целом, не привязываясь к конкретной предметной области, поэтому книга должна быть одинаково полезна и для разработчиков серверных приложений, и для разработчиков на Android, и для всех, кто создает проекты для JVM.

Как организована эта книга

Книга делится на две части. *Часть 1* объясняет, как начать использовать Kotlin вместе с существующими библиотеками и API:

- *глава 1* рассказывает о ключевых целях, ценностях и областях применения языка и показывает различные способы запуска кода на Kotlin;
- *глава 2* демонстрирует важные элементы любой программы на языке Kotlin, включая управляющие структуры, переменные и функции;

- *в главе 3* подробно рассматриваются объявления функций в Kotlin, а также вводятся понятия функций-расширений (extension functions) и свойств-расширений (extension properties);
- *глава 4* посвящена объявлению классов и знакомит с понятиями классов данных (data classes) и объектов-компаньонов (companion objects);
- *глава 5* знакомит с лямбда-выражениями и демонстрирует ряд примеров их использования в стандартной библиотеке Kotlin;
- *глава 6* описывает систему типов Kotlin, обращая особое внимание на работу с типами, допускающими значения null, и с коллекциями.

Часть 2 научит вас создавать собственные API и абстракции на языке Kotlin и охватывает некоторые более продвинутые особенности языка:

- *глава 7* рассказывает о соглашениях, придающих особый смысл методам и свойствам с определенными именами, и вводит понятие делегируемых свойств (delegated properties);
- *глава 8* показывает, как объявлять функции высшего порядка – функции, принимающие другие функции или возвращающие их. Здесь также вводится понятие встраиваемых функций (inline functions);
- *глава 9* глубоко погружается в тему обобщенных типов (generics), начиная с базового синтаксиса и переходя к более продвинутым темам, таким как овеществляемые типовые параметры (reified type parameters) и вариантность;
- *глава 10* посвящена использованию аннотаций и механизма рефлексии (reflection) и организована вокруг JKid – простой библиотеки сериализации в формат JSON, которая интенсивно использует эти понятия;
- *глава 11* вводит понятие предметно-ориентированного языка (DSL), описывает инструменты Kotlin для их создания и демонстрирует множество примеров DSL.

В книге есть три приложения. Приложение A объясняет, как выполнять сборку проектов на Kotlin с помощью Gradle, Maven и Ant. Приложение B фокусируется на документирующих комментариях и создании документации с описанием API модулей. Приложение C является руководством по экосистеме Kotlin и поиску актуальной информации в Интернете.

Книгу лучше читать последовательно, от начала до конца, но также можно обращаться к отдельным главам, посвященным интересующим вам конкретным темам, и переходить по перекрестным ссылкам для уточнения незнакомых понятий.

Соглашения об оформлении программного кода и загружаемые ресурсы

В книге приняты следующие соглашения:

- *Курсивом* обозначаются новые термины.
- `Моноширинным шрифтом` выделены фрагменты кода, имена классов и функций и другие идентификаторы.
- Примеры кода сопровождаются многочисленными примечаниями, подчеркивающими важные понятия.

Многие листинги кода в книге показаны вместе с его выводом. В таких случаях строки кода, производящие вывод, начинаются с префикса >>>, как показано ниже:

```
>>> println("Hello World")
Hello World
```

Некоторые примеры являются полноценными программами, тогда как другие – лишь фрагменты, демонстрирующие определенные понятия и могущие содержать сокращения (обозначенные как ...) или синтаксические ошибки (описанные в тексте книги или самих примерах). Выполняемые примеры можно загрузить в виде zip-архива на сайте издательства www.manning.com/books/kotlin-in-action. Примеры из книги также выгружены в онлайн-окружение http://try.kotlinlang.org, где вы сможете опробовать любой пример, сделав всего несколько щелчков в окне браузера.

Авторы онлайн

Покупка книги «*Kotlin в действии*» дает право свободного доступа к закрытому веб-форуму издательства Manning Publication, где можно высказать свои замечания о книге, задать технические вопросы и получить помощь от авторов и других пользователей. Чтобы получить доступ к форуму, перейдите по ссылке www.manning.com/books/kotlin-in-action. На этой странице описывается, как попасть на форум после регистрации, какая помощь доступна и какие правила поведения действуют на форуме.

Издательство Manning обязуется предоставить читателям площадку для содержательного диалога не только между читателями, но и между читателями и авторами. Но авторы не обязаны выделять определённое количество времени для участия, т. к. их вклад в работу форума является добровольным (и неоплачиваемым). Мы предлагаем читателям задавать авторам действительно непростые вопросы, чтобы их интерес не угасал!

Прочие онлайн-ресурсы

Kotlin имеет активное сообщество, поэтому имеющие вопросы или желающие пообщаться с другими пользователями Kotlin могут воспользоваться следующими ресурсами:

- официальный форум Kotlin – https://discuss.kotlinlang.org;
- чат Slack – http://kotlinlang.slack.com (вы можете получить приглашение по ссылке http://kotlinslackin.herokuapp.com/);
- вопросы и ответы с тегом Kotlin на Stack Overflow – http://stackoverflow.com/questions/tagged/kotlin;
- Kotlin на форуме Reddit – http://www.reddit.com/r/Kotlin.

Об авторах

Дмитрий Жемеров работает в компании JetBrains с 2003 года и принимал участие в разработке многих продуктов, в том числе IntelliJ IDEA, PyCharm и WebStorm. Был одним из первых участников команды Kotlin, создал начальную версию генератора байт-кода JVM из кода Kotlin и сделал много презентаций о языке Kotlin на различных встречах по всему миру. Сейчас возглавляет команду, работающую над плагином Kotlin IntelliJ IDEA.

Светлана Исакова вошла в состав команды Kotlin в 2011 году. Работала над механизмом вывода типов (type inference) и подсистемой компилятора по разрешению перегруженных имен. Сейчас она – технический евангелист, рассказывает о Kotlin на конференциях и разрабатывает онлайн-курс по языку Kotlin.

Об изображении на обложке

Иллюстрация на обложке книги «*Kotlin в действии*» называется «Одежда русской женщины на Валдае в 1764 году». Город Валдай находится в Новгородской области, между Москвой и Санкт-Петербургом. Иллюстрация взята из работы Томаса Джефферис (Thomas Jefferys) «Коллекция платьев разных народов, древних и современных», опубликованной в Лондоне между 1757 и 1772 г.. На титульной странице говорится, что это – раскрашенная вручную гравюра, обработанная гуммиарабиком для повышения яркости. Томаса Джефферис (1719–1771) называли географом короля Георга III. Он был английским картографом и ведущим поставщиком карт своего времени. Он гравировал и печатал карты для правительства и других официальных органов, выпускал широкий спектр коммерческих карт и атласов, особенно Северной Америки. Работа картографом пробудила интерес к местным традиционным нарядам в землях, которые он исследовал и картографировал; эти наряды великолепно представлены в четырехтомном сборнике.

Очарование дальними странами и путешествия для удовольствия были относительно новым явлением в восемнадцатом веке, и такие коллекции, как эта, были популярны и показывали туристам и любителям книг о путешествиях жителей других стран. Разнообразие рисунков в книгах Джефферис красноречиво говорит об уникальности и индивидуальности народов мира много веков назад. С тех пор стиль одежды сильно изменился, и разнообразие, характеризующее различные области и страны, исчезло. Сейчас часто трудно отличить даже жителей одного континента от дру-

гого. Возможно, с оптимистической точки зрения, мы обменяли культурное и визуальное разнообразие на более разнообразную частную жизнь или более разнообразную и интересную интеллектуальную и техническую деятельность.

В наше время, когда трудно отличить одну компьютерную книгу от другой, издательство Manning с инициативой и находчивостью наделяет книги обложками, изображающими богатое разнообразие жизненного уклада народов многовековой давности, давая новую жизнь рисункам Джефферис.

Часть 1

Введение в Kotlin

Цель этой части книги – помочь начать продуктивно писать код на языке Kotlin, используя существующие API. Глава 1 познакомит вас с языком Kotlin в общих чертах. В главах 2–4 вы узнаете, как в Kotlin реализованы основные понятия языка Java – операторы, функции, классы и типы, – и как Kotlin обогащает их, делая программирование более приятным. Вы сможете положиться на имеющиеся знания языка Java, а также на вспомогательные инструменты, входящие в состав интегрированной среды разработки, и конвертер кода на Java в код на Kotlin, чтобы быстро начать писать код. В главе 5 вы узнаете, как лямбда-выражения помогают эффективно решать некоторые из распространенных задач программирования, такие как работа с коллекциями. Наконец, в главе 6 вы познакомитесь с одной из ключевых особенностей Kotlin – поддержкой операций со значениями `null`.

Глава 1

Kotlin: что это и зачем

В этой главе:

- общий обзор языка Kotlin;
- основные особенности;
- возможности разработки для Android и серверных приложений;
- отличие Kotlin от других языков;
- написание и выполнение кода на языке Kotlin.

Что же такое Kotlin? Это новый язык программирования для платформы Java. Kotlin – лаконичный, безопасный и прагматичный язык, совместимый с Java. Его можно использовать практически везде, где применяется Java: для разработки серверных приложений, приложений для Android и многого другого. Kotlin прекрасно работает со всеми существующими библиотекам и фреймворками, написанными на Java, не уступая последнему в производительности. В этой главе мы подробно рассмотрим основные черты языка Kotlin.

1.1. Знакомство с Kotlin

Начнем с небольшого примера для демонстрации языка Kotlin. В этом примере определяется класс `Person`, создается коллекция его экземпляров, выполняется поиск самого старого и выводится результат. Даже в этом маленьком фрагменте кода можно заметить множество интересных особенностей языка Kotlin; мы выделили некоторые из них, чтобы вам проще было отыскать их в книге в будущем. Код пояснен довольно кратко, но не беспокойтесь, если что-то останется непонятным. Позже мы подробно всё обсудим.

Желающие опробовать этот пример могут воспользоваться онлайн-полигоном по адресу: `http://try.kotl.in`. Введите пример, щелкните на кнопке **Run** (Запустить), и код будет выполнен.

Листинг 1.1. Первое знакомство с Kotlin

```
data class Person(val name: String,                    <— Класс «данных»
                  val age: Int? = null)                <— Тип, допускающий значение null (Int?);
                                                          значение параметра по умолчанию

fun main(args: Array<String>) {                        <— Функция верхнего уровня
  val persons = listOf(Person("Alice"),
                       Person("Bob", age = 29))        <— Именованный аргумент

  val oldest = persons.maxBy { it.age ?: 0 }           <— Лямбда-выражение; оператор «Элвис»
  println("The oldest is: $oldest")                    <— Строка-шаблон
}

// The oldest is: Person(name=Bon, age=29)             <— Часть вывода автоматически сгенерирована
                                                          методом toString
```

Здесь объявляется простой класс данных с двумя свойствами: `name` и `age`. Свойству `age` по умолчанию присваивается значение `null` (если оно не задано). При создании списка людей возраст Алисы не указывается, поэтому он принимает значение `null`. Затем, чтобы отыскать самого старого человека в списке, вызывается функция `maxBy`. Лямбда-выражение, которое передается функции, принимает один параметр с именем `it` по умолчанию. *Оператор «Элвис»* (`?:`) возвращает ноль, если возраст имеет значение `null`. Поскольку возраст Алисы не указан, оператор «Элвис» заменит его нулем, поэтому Боб получит приз как самый старый человек.

Вам понравилось? Читайте дальше, чтобы узнать больше и стать экспертом в языке Kotlin. Мы надеемся, что скоро вы увидите такой код в своих проектах, а не только в этой книге.

1.2. Основные черты языка Kotlin

Возможно, у вас уже есть некоторое представление о языке Kotlin. Давайте подробнее рассмотрим его ключевые особенности. Для начала определим типы приложений, которые можно создавать с его помощью.

1.2.1. Целевые платформы: серверные приложения, Android и везде, где запускается Java

Основная цель языка Kotlin – предложить более компактную, производительную и безопасную альтернативу языку Java, пригодную для использования везде, где сегодня применяется Java. Java – чрезвычайно популярный язык, который используется в самых разных окружениях, начиная от смарт-карт (технология Java Card) до крупнейших вычислительных центров таких компаний, как Google, Twitter и LinkedIn. В большинстве таких окружений применение Kotlin способно помочь разработчикам достигать своих целей меньшим объемом кода и избегая многих неприятностей.

Наиболее типичные области применения Kotlin:

- разработка кода, работающего на стороне сервера (как правило, серверной части веб-приложений);
- создание приложений, работающих на устройствах Android.

Но Kotlin работает также в других областях. Например, код на Kotlin можно выполнять на устройствах с iOS, используя технологию Intel Multi-OS Engine (https://software.intel.com/en-us/multi-os-engine). На Kotlin можно писать и настольные приложения, используя его совместно с TornadoFX (https://github.com/edvin/tornadofx) и JavaFX[1] .

Помимо Java, код на Kotlin можно скомпилировать в код на JavaScript и выполнять его в браузере. Но на момент написания этой книги поддержка JavaScript находилась в стадии исследования и прототипирования, поэтому она осталась за рамками данной книги. В будущих версиях языка также рассматривается возможность поддержки других платформ.

Как видите, область применения Kotlin достаточно обширна. Kotlin не ограничивается одной предметной областью или одним типом проблем, с которыми сегодня сталкиваются разработчики программного обеспечения. Вместо этого он предлагает всестороннее повышение продуктивности при решении любых задач, возникающих в процессе разработки. Он также предоставляет отличный уровень интеграции с библиотеками, созданными для поддержки определенных предметных областей или парадигм программирования. Давайте рассмотрим основные качества Kotlin как языка программирования.

1.2.2. Статическая типизация

Так же, как Java, Kotlin – *статически типизированный* язык программирования. Это означает, что тип каждого выражения в программе известен во время компиляции, и компилятор может проверить, что методы и поля, к которым вы обращаетесь, действительно существуют в используемых объектах.

Этим Kotlin отличается от *динамически типизированных* (dynamically typed) языков программирования на платформе JVM, таких как Groovy и JRuby. Такие языки позволяют определять переменные и функции, способные хранить или возвращать данные любого типа, а ссылки на поля и методы определяются во время выполнения. Это позволяет писать более компактный код и дает большую гибкость в создании структур данных. Но в языках с динамической типизацией есть свои недостатки: например, опечатки в именах нельзя обнаружить во время компиляции, что влечет появление ошибок во время выполнения.

---

[1] «JavaFX: Getting Started with JavaFX», Oracle, http://mng.bz/500y.

С другой стороны, в отличие от Java, Kotlin не требует явно указывать тип каждой переменной. В большинстве случаев тип переменной может быть определен автоматически. Вот самый простой пример:

```
val x = 1
```

Вы объявляете переменную, но поскольку она инициализируется целочисленным значением, Kotlin автоматически определит её тип как `Int`. Способность компилятора определять типы из контекста называется *выведением типа* (type inference).

Ниже перечислены некоторые преимущества статической типизации:

- *Производительность* – вызов методов происходит быстрее, поскольку во время выполнения не нужно выяснять, какой метод должен быть вызван.
- *Надежность* – корректность программы проверяется компилятором, поэтому вероятность ошибок во время выполнения меньше.
- *Удобство сопровождения* – работать с незнакомым кодом проще, потому что сразу видно, с какими объектами код работает.
- *Поддержка инструментов* – статическая типизация позволяет увереннее выполнять рефакторинг, обеспечивает точное автодополнение кода и поддержку других возможностей IDE.

Благодаря поддержке выведения типов в Kotlin исчезает излишняя избыточность статически типизированного кода, поскольку больше не нужно объявлять типы явно.

Система типов в Kotlin поддерживает много знакомых понятий. Классы, интерфейсы и обобщенные типы работают практически так же, как в Java, так что большую часть своих знаний Java вы с успехом сможете применить на Kotlin. Однако есть кое-что новое.

Наиболее важным нововведением в Kotlin является поддержка типов, допускающих значения `null` (nullable types), которая позволяет писать более надежные программы за счет выявления потенциальных ошибок обращения к пустому указателю на этапе компиляции. Мы ещё вернемся к типам, допускающим значение `null`, далее в этой главе и подробно обсудим их в главе 6.

Другим новшеством в системе типов Kotlin является поддержка *функциональных типов* (function types). Чтобы понять, о чем идет речь, обратимся к основным идеям функционального программирования и посмотрим, как они поддерживаются в Kotlin.

1.2.3. Функциональное и объектно-ориентированное программирование

Как Java-разработчик вы, без сомнения, знакомы с основными понятиями объектно-ориентированного программирования, но функциональное

программирование может оказаться для вас в новинку. Ниже перечислены ключевые понятия функционального программирования:

- *Функции как полноценные объекты* – с функциями (элементами поведения) можно работать как со значениями. Их можно хранить в переменных, передавать в аргументах или возвращать из других функций.
- *Неизменяемость* – программные объекты никогда не изменяются, что гарантирует неизменность их состояния после создания.
- *Отсутствие побочных эффектов* – функции всегда возвращают один и тот же результат для тех же аргументов, не изменяют состояние других объектов и не взаимодействуют с окружающим миром.

Какие преимущества дает функциональный стиль? Во-первых, *лаконичность*. Функциональный код может быть более элегантным и компактным, по сравнению с императивными аналогами, потому что возможность работать с функциями как со значениями дает возможность создавать более мощные абстракции, позволяющие избегать дублирования в коде.

Представьте, что у вас есть два похожих фрагмента кода, решающих аналогичную задачу (например, поиск элемента в коллекции), которые отличаются в деталях (способом проверки критериев поиска). Вы легко сможете перенести общую логику в функцию, передавая отличающиеся части в виде аргументов. Эти аргументы сами будут функциями, но вы сможете описать их, используя лаконичный синтаксис анонимных функций, называемых *лямбда-выражениями*:

```
fun findAlice() = findPerson { it.name == "Alice" }
fun findBob() = findPerson { it.name == "Bob" }
```

Функция findPerson() описывает общую логику поиска

Блок кода в фигурных скобках задает свойства искомого элемента

Второе преимущество функциональной парадигмы – *безопасное многопоточное программирование*. Одним из основных источников ошибок в многопоточных программах является модификация одних и тех же данных из нескольких потоков без надлежащей синхронизации. Используя неизменяемые структуры данных и чистые функции, можно не опасаться никаких изменений и не надо придумывать сложных схем синхронизации.

Наконец, функциональное программирование *облегчает тестирование*. Функции без побочных эффектов можно проверять по отдельности, без необходимости писать много кода для настройки окружения.

В целом функциональную парадигму можно использовать в любом языке программирования, включая Java, и многие ее аспекты считаются хорошим стилем программирования. Но не все языки поддерживают соответствующий синтаксис и библиотеки, упрощающие применение этого стиля; например, такая поддержка отсутствовала в Java до версии Java 8.

Язык Kotlin изначально обладает богатым арсеналом возможностей для поддержки функционального программирования. К ним относятся:

- *функциональные типы,* позволяющие функциям принимать или возвращать другие функции;
- *лямбда-выражения,* упрощающие передачу фрагментов кода;
- *классы данных,* предоставляющие емкий синтаксис для создания неизменяемых объектов-значений;
- обширный набор средств в стандартной библиотеке для работы с объектами и коллекциями в функциональном стиле.

Kotlin позволяет программировать в функциональном стиле, но не требует этого. Когда нужно, вы можете работать с изменяемыми данными и писать функции с побочными эффектами без всяких затруднений. Работать с фреймворками, основанными на иерархиях классов и интерфейсах, так же легко, как на языке Java. В программном коде на Kotlin вы можете совмещать объектно-ориентированный и функциональный подходы, используя для каждой решаемой проблемы наиболее подходящий инструмент.

1.2.4. Бесплатный язык с открытым исходным кодом

Язык Kotlin, включая компилятор, библиотеки и все связанные с ними инструменты, – это проект с открытым исходным кодом, который может свободно применяться для любых целей. Он доступен на условиях лицензии Apache 2, разработка ведется открыто в GitHub (`http://github.com/jetbrains/kotlin`), и любой добровольный вклад приветствуется. Также на выбор есть три IDE с открытым исходным кодом, поддерживающих разработку приложений на Kotlin: IntelliJ IDEA Community Edition, Android Studio и Eclipse. (Конечно же, поддержка Kotlin имеется также в IntelliJ IDEA Ultimate.)

Теперь, когда вы получили некоторое представление о Kotlin, пришла пора узнать, как использовать его преимущества для конкретных практических приложений.

1.3. Приложения на Kotlin

Как мы уже упоминали ранее, основные области применения Kotlin – это создание серверной части приложений и разработка для Android. Рассмотрим эти области по очереди и разберемся, почему Kotlin так хорошо для них подходят.

1.3.1. Kotlin на сервере

Серверное программирование – довольно широкое понятие. Оно охватывает все следующие типы приложений и многое другое:

- веб-приложения, возвращающие браузеру страницы HTML;
- серверные части мобильных приложений, открывающие доступ к своему JSON API по протоколу HTTP;
- микрослужбы, взаимодействующие с другими микрослужбами посредством RPC.

Разработчики много лет создавали эти типы приложений на Java и накопили обширный набор фреймворков и технологий, облегчающих их создание. Подобные приложения, как правило, не разрабатываются изолированно и не пишутся с нуля. Почти всегда есть готовая система, которую нужно расширять, улучшать или заменять, и новый код должен интегрироваться с существующими частями системы, которые могли быть написаны много лет назад.

Большим преимуществом Kotlin в подобной среде является его взаимодействие с существующим Java-кодом. Kotlin отлично подходит и для создания новых компонентов, и для переноса кода существующей службы на Kotlin. Вы не столкнетесь с затруднениями, когда коду на Kotlin понадобится унаследовать Java-классы или отметить специальными аннотациями методы и поля класса. Преимущество же заключается в том, что код системы станет более компактным, надежным и простым в обслуживании.

В то же время Kotlin предлагает ряд новых приемов для создания таких систем. Например, его поддержка шаблона «Строитель» (Builder) позволяет создавать произвольные графы объектов, используя очень лаконичный синтаксис, не отказываясь при этом от полного набора абстракций и инструментов повторного использования кода в языке.

Один из простейших примеров использования этой особенности – библиотека создания разметки HTML: лаконичное и полностью типобезопасное решение, которое может полностью заменить сторонний язык шаблонов. Например:

```
fun renderPersonList(persons: Collection<Person>) =
  createHTML().table {
    for (person in persons) {
      tr {
        td { +person.name }
        td { +person.age }
      }
    }
  }
}
```

Обычный цикл

Функции, выполняющие отображение в теги HTML

Вы легко сможете объединить функции, выводящие теги HTML, с обычными конструкциями языка Kotlin. Вам больше не нужно изучать отдельный язык шаблонов со своим синтаксисом только затем, чтобы использовать цикл при создании HTML-страницы.

Другой пример использования ясности Kotlin и лаконичности предметно-ориентированных языков – фреймворки хранения данных. Например, фреймворк Exposed (https://github.com/jetbrains/exposed) поддерживает простой и понятный предметный язык для описания структуры базы данных SQL и выполнения запросов прямо из кода на Kotlin с полноценной проверкой типов. Вот маленький пример, показывающий возможности такого подхода:

```
object CountryTable : IdTable() {
  val name = varchar("name", 250).uniqueIndex()
  val iso = varchar("iso", 2).uniqueIndex()
}

class Country(id: EntityID) : Entity(id) {
  var name: String by CountryTable.name
  var iso: String by CountryTable.iso
}

val russia = Country.find {
  CountryTable.iso.eq("ru")
}.first()

println(russia.name)
```

Описание таблицы в базе данных

Определение класса, соответствующего сущности в базе данных

Вы можете выполнять запросы к базе данных на чистом Kotlin

Мы рассмотрим эти методы более подробно в разделе 7.5 и в главе 11.

1.3.2. Kotlin в Android

Типичное мобильное приложение значительно отличается от типичного корпоративного приложения. Оно гораздо меньше по объему, не так сильно зависит от интеграции с существующими кодовыми базами и, как правило, должно быть разработано за короткий срок, при этом поддерживая надежную работу на различных устройствах. Kotlin также хорошо справляется с проектами такого типа.

Языковые особенности Kotlin в сочетании со специальным плагином для компилятора делают разработку для Android приятным и продуктивным занятием. Часто встречающиеся задачи программирования, такие как добавление обработчиков событий в элементы управления или связывание элементов интерфейса с полями, можно решить гораздо меньшим объемом кода, а иногда и совсем без кода (компилятор сгенерирует его за вас). Библиотека Anko (https://github.com/kotlin/anko), тоже разработанная командой Kotlin, сделает вашу работу ещё приятнее за счет Kotlin-совместимых адаптеров для многих стандартных Android API.

Ниже продемонстрирован простой пример использования Anko, чтобы вы почувствовали, что значит разработка для Android на языке Kotlin. Вы

можете скопировать этот код в класс-наследник `Activity` и получить готовое Android-приложение!

```
verticalLayout {
  val name = editText()
  button("Say Hello") {
    onClick { toast("Hello, ${name.text}!") }
  }
}
```

Создание простого текстового поля

По щелчку на кнопке вывести всплывающее сообщение с содержимым текстового поля

Лаконичный синтаксис подключения обработчика событий и отображения всплывающего сообщения

Другое большое преимущество Kotlin – повышенная надежность приложений. Если у вас есть какой-либо опыт разработки приложений для Android, вы наверняка знакомы с сообщением «К сожалению, процесс остановлен». Это диалоговое окно появляется, когда приложение встречается с необработанным исключением, обычно `NullPointerException`. Система типов в Kotlin, с её точным контролем значений `null`, значительно снижает риск исключений из-за обращения к пустому указателю. Большую часть кода, который в Java приводит к исключению `NullPointerException`, в языке Kotlin попросту не удастся скомпилировать, что гарантирует исправление ошибки до того, как приложение попадет к пользователям.

В то же время, поскольку Kotlin полностью совместим с Java 6, его использование не создает каких-либо новых опасений в области совместимости. Вы сможете воспользоваться всеми новыми возможностями языка, а пользователи по-прежнему смогут запускать ваше приложение на устройствах даже с устаревшей версией Android.

С точки зрения производительности, применение Kotlin также не влечет за собой каких-либо недостатков. Код, сгенерированный компилятором Kotlin, выполняется так же эффективно, как обычный Java-код. Стандартная библиотека Kotlin довольно небольшая, поэтому вы не заметите существенного увеличения размеров скомпилированного приложения. А при использовании лямбда-выражений многие функции из стандартной библиотеки Kotlin просто будут встраивать их в место вызова. Встраивание лямбда-выражений гарантирует, что не будет создано никаких новых объектов, а приложение не будет работать медленнее из-за дополнительных пауз сборщика мусора (GC).

Познакомившись с преимуществами Котлин, давайте теперь рассмотрим философию Kotlin – основные черты, которые отличают Kotlin от других современных языков для платформы JVM.

1.4. Философия Kotlin

Говоря о Kotlin, мы подчеркиваем, что это прагматичный, лаконичный, безопасный язык, совместимый с Java. Что мы подразумеваем под каждым из этих понятий? Давайте рассмотрим по порядку.

1.4.1. Прагматичность

Под *прагматичностью* мы понимаем одну простую вещь: Kotlin является практичным языком, предназначенным для решения реальных задач. Он спроектирован с учетом многолетнего опыта создания крупномасштабных систем, а его характеристики выбирались исходя из задач, которые чаще всего приходится решать разработчикам. Более того, разработчики в компании JetBrains и в сообществе несколько лет использовали ранние версии Kotlin, и полученная от них обратная связь помогла сформировать итоговую версию языка. Это дает нам основания заявлять, что Kotlin действительно помогает решать проблемы в реальных проектах.

Kotlin не является исследовательским языком. Мы не пытаемся создать ультрасовременный язык программирования и исследовать различные инновационные идеи. Вместо этого, когда это возможно, мы полагаемся на особенности и готовые решения в других языках программирования, оказавшиеся успешными. Это уменьшает сложность языка и упрощает его изучение за счет того, что многие понятия уже знакомы.

Кроме того, Kotlin не требует применения какого-то конкретного стиля программирования или парадигмы. Приступая к изучению языка, вы сможете использовать стиль и методы, знакомые вам по работе с Java. Позже вы постепенно откроете более мощные возможности Kotlin и научитесь применять их в своем коде, делая его более лаконичным и идиоматичным.

Другой аспект прагматизма Kotlin касается инструментария. Хорошая среда разработки так же важна для программиста, как и хороший язык; следовательно, поддержка Kotlin в IDE не является чем-то малозначительным. Плагин для IntelliJ IDEA с самого начала разрабатывался параллельно с компилятором, а свойства языка всегда рассматривались через призму инструментария.

Поддержка в IDE также играет важную роль в изучении возможностей Kotlin. В большинстве случаев инструменты автоматически обнаруживают шаблонный код, который можно заменить более лаконичными конструкциями, и предлагают его исправить. Изучая особенности языка в исправлениях, предлагаемых инструментами, вы сможете научиться применять их в своем собственном коде.

1.4.2. Лаконичность

Известно, что разработчики тратят больше времени на чтение существующего кода, чем на создание нового. Представьте, что вы в составе команды участвуете в разработке большого проекта, и вам нужно добавить новую функцию или исправить ошибку. Каковы ваши первые шаги? Вы найдете область кода, которую нужно изменить, и только потом сделаете исправление. Вы должны прочесть много кода, чтобы выяснить, что нужно сделать. Этот код мог быть недавно написан вашими коллегами, кем-то,

кто уже не работает на проекте, или вами, но давным-давно. Только поняв, как работает окружающий код, вы сможете внести необходимые изменения.

Чем проще и лаконичнее код, тем быстрее вы поймете, что он делает. Конечно, хороший дизайн и выразительные имена играют свою роль. Но выбор языка и лаконичность также имеют большое значение. Язык является *лаконичным*, если его синтаксис явно выражает намерения кода, не загромождая его вспомогательными конструкциями с деталями реализации.

Создавая Kotlin, мы старались организовать синтаксис так, чтобы весь код нес определенный смысл, а не писался просто ради удовлетворения требований к структуре кода. Большинство операций, стандартных для Java, таких как определение методов чтения/записи для свойств и присваивание параметров конструктора полям объекта, реализовано в Kotlin неявно и не захламляет исходного кода.

Другой причиной ненужной избыточности кода является необходимость описания типовых задач, таких как поиск элемента в коллекции. Как во многих современных языках, в Kotlin есть богатая стандартная библиотека, позволяющая заменить эти длинные, повторяющиеся участки кода вызовами библиотечных функций. Поддержка лямбда-выражений в Kotlin позволяет передавать небольшие блоки кода в библиотечные функции и инкапсулировать всю общую логику в библиотеке, оставляя в коде только уникальную логику для решения конкретных задач.

В то же время Kotlin не пытается минимизировать количество символов в исходном коде. Например, несмотря на то что Kotlin поддерживает перегрузку операторов, пользователи не могут определять собственных операторов. Поэтому разработчики библиотек не смогут заменять имена методов загадочными последовательностями знаков препинания. Как правило, слова читать легче, чем знаки препинания, и их проще искать в документации.

Более лаконичный код требует меньше времени для написания и, что особенно важно, меньше времени для чтения. Это повышает продуктивность и позволяет разрабатывать программы значительно быстрее.

1.4.3. Безопасность

Называя язык *безопасным*, мы обычно подразумеваем, что его дизайн предотвращает появление определенных видов ошибок в программах. Конечно, это качество не абсолютно: ни один язык не защитит от всех возможных ошибок. Кроме того, за предотвращение ошибок, как правило, приходится платить. Вы должны сообщить компилятору больше информации о планируемых действиях программы, чтобы компилятор мог проверить, что код действительно соответствует этим действиям. Вследствие

этого всегда возникает компромисс между приемлемым уровнем безопасности и потерей продуктивности из-за необходимости добавления дополнительных аннотаций.

В Kotlin мы попытались достичь более высокого уровня безопасности, чем в Java, с минимальными накладными расходами. Выполнение кода в JVM уже дает многие гарантии безопасности, например: защита памяти избавляет от переполнения буфера и других проблем, связанных с некорректным использованием динамически выделяемой памяти. Будучи статически типизированным языком для JVM, Kotlin также обеспечивает безопасность типов в приложении. Это обходится дешевле, чем в Java: вам не нужно явно объявлять типы всех сущностей, поскольку во многих случаях компилятор может вывести тип автоматически.

Но Kotlin идёт ещё дальше: теперь ещё больше ошибок может быть предотвращено во время компиляции, а не во время выполнения. Важнее всего, что Kotlin пытается избавить программу от исключений `NullPointerException`. Система типов в Kotlin отслеживает значения, которые могут или и не могут принимать значение `null`, и запрещает операции, которые могут привести к возникновению `NullPointerException` во время выполнения. Дополнительные затраты при этом минимальны: чтобы указать, что значение может принимать значение `null`, требуется только один символ – вопросительный знак в конце:

```
val s: String? = null          <— Может содержать значение null
val s2: String = ""            <— Не может содержать значения null
```

Кроме того, Kotlin поддерживает множество удобных способов обработки значений, которые могут содержать `null`. Это очень помогает в устранении сбоев приложений.

Другой тип исключений, которого помогает избежать Kotlin, – это `ClassCastException`. Оно возникает во время приведения типа без предварительной проверки возможности такой операции. В Java разработчики часто не делают такую проверку, потому что имя типа приходится повторять в выражении проверки и в коде приведения типа. Однако в Kotlin проверка типа и приведение к нему объединены в одну операцию: после проверки можно обращаться к членам этого типа без дополнительного явного приведения. Поэтому нет причин не делать проверку и нет никаких шансов сделать ошибку. Вот как это работает:

```
if (value is String)           <— Проверка типа
  println(value.toUpperCase())          <— Вызов метода типа
```

1.4.4. Совместимость

В отношении совместимости часто первым возникает вопрос: «Смогу ли я использовать существующие библиотеки?» Kotlin дает однозначный

ответ: «Да, безусловно». Независимо от того, какой тип API предлагает библиотека, вы сможете работать с ними напрямую из Kotlin. Вы сможете вызывать Java-методы, наследовать Java-классы и реализовывать интерфейсы, использовать Java-аннотации в Kotlin-классах и т. д.

В отличие от некоторых других языков для JVM, Kotlin идет еще дальше по пути совместимости, позволяя также легко вызывать код Kotlin из Java. Для этого не требуется никаких трюков: классы и методы на Kotlin можно вызывать как обычные классы и методы на Java. Это дает максимальную гибкость при смешивании кода Java с кодом Kotlin на любом этапе вашего проекта. Приступая к внедрению Kotlin в свой Java-проект, попробуйте преобразовать какой-нибудь один класс из Java в Kotlin с помощью конвертера, и остальной код будет продолжать компилироваться и работать без каких-либо изменений. Этот прием работает независимо от назначения преобразованного класса.

Ещё одна область, где уделяется большое внимание совместимости, – максимальное использование существующих библиотек Java. Например, в Kotlin нет своей библиотеки коллекций. Он полностью полагается на классы стандартной библиотеки Java, расширяя их дополнительными функциями для большего удобства использования в Kotlin. (Мы рассмотрим этот механизм более подробно в разделе 3.3.) Это означает, что вам никогда не придется обертывать или конвертировать объекты при использовании Java API из Kotlin или наоборот. Все богатство возможностей предоставляется языком Kotlin без дополнительных накладных расходов во время выполнения.

Инструментарий Kotlin также обеспечивает полную поддержку многоязычных проектов. Можно скомпилировать произвольную смесь исходных файлов на Java и Kotlin с любыми зависимостями друг от друга. Поддержка IDE также распространяется на оба языка, что позволяет:

- свободно перемещаться между исходными файлами на Java и Kotlin;
- отлаживать смешанные проекты, перемещаясь по коду, написанному на разных языках;
- выполнять рефакторинг Java-методов, получая нужные изменения в коде на Kotlin, и наоборот.

Надеемся, что теперь мы убедили вас дать шанс языку Kotlin. Итак, как начать пользоваться им? В следующем разделе мы рассмотрим процесс компиляции и выполнения кода на Kotlin из командной строки и с помощью различных инструментов.

1.5. Инструментарий Kotlin

Как и Java, Kotlin – компилируемый язык. То есть, прежде чем запустить код на Kotlin, его нужно скомпилировать. Давайте обсудим процесс компи-

ляции, а затем рассмотрим различные инструменты, которые позаботятся о нём за вас. Более подробную информацию о настройке вашего окружения вы найдете в разделе «Tutorials» на сайте Kotlin (https://kotlinlang.org/docs/tutorials).

1.5.1. Компиляция кода на Kotlin

Исходный код на Kotlin обычно хранится в файлах с расширением *.kt*. Компилятор Kotlin анализирует исходный код и генерирует файлы *.class* так же, как и компилятор Java. Затем сгенерированные файлы *.class* упаковываются и выполняются с использованием процедуры, стандартной для данного типа приложения. В простейшем случае скомпилировать код из командной строки можно с помощью команды `kotlinc`, а запустить – командой `java`:

```
kotlinc <исходный файл или каталог> -include-runtime -d <имя jar-файла>
java -jar <имя jar-файла>
```

На рис. 1.1 приводится упрощенная схема процесса сборки в Kotlin.

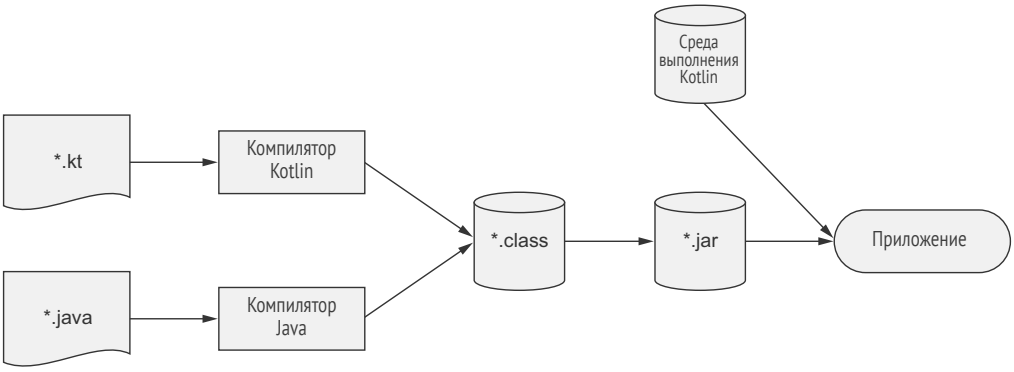

Рис. 1.1. Процесс сборки в Kotlin

Код, скомпилированный с помощью компилятора Kotlin, зависит от *библиотеки среды выполнения Kotlin*. Она содержит определения собственных классов стандартной библиотеки Kotlin, а также расширения, которые Kotlin добавляет в стандартный Java API. Библиотека среды выполнения должна распространяться вместе с приложением.

На практике для компиляции кода обычно используются специализированные системы сборки, такие как Maven, Gradle или Ant. Kotlin совместим со всеми ними, и мы обсудим эту тему в приложении А. Все эти системы сборки поддерживают многоязычные проекты, включающие код на Kotlin и Java в общую базу кода. Кроме того, Maven и Gradle сами позаботятся о подключении библиотеки времени выполнения Kotlin как зависимости вашего приложения.

1.5.2. Плагин для Intellij IDEA и Android Studio

Плагин с поддержкой Kotlin для IntelliJ IDEA разрабатывался параллельно с языком и является наиболее полной средой разработки для языка Kotlin. Это зрелая и стабильная среда, обладающая полным набором инструментов для разработки на Kotlin. Плагин Kotlin вошел в состав IntelliJ IDEA, начиная с версии 15, поэтому дополнительная установка не потребуется. Вы можете использовать бесплатную среду разработки IntelliJ IDEA Community Edition с открытым исходным кодом, или IntelliJ IDEA Ultimate. Выберите **Kotlin** в диалоге **New Project** (Новый проект) и можете начинать разработку.

Если вы используете Android Studio, можете установить плагин Kotlin с помощью диспетчера плагинов. В диалоговом окне **Settings** (Настройки) выберите вкладку **Plugins** (Плагины), затем щелкните на кнопке **Install JetBrains Plugin** (Установить плагин JetBrains) и выберите из списка **Kotlin**.

1.5.3. Интерактивная оболочка

Для быстрого опробования небольших фрагментов кода на Kotlin можно использовать интерактивную оболочку (так называемый цикл *REPL* – Read Eval Print Loop: чтение ввода, выполнение, вывод результата, повтор). В REPL можно вводить код на Kotlin строку за строкой и сразу же видеть результаты выполнения. Чтобы запустить REPL, выполните команду `kotlinc` без аргументов или воспользуйтесь соответствующим пунктом меню в плагине IntelliJ IDEA.

1.5.4. Плагин для Eclipse

Пользователи Eclipse также имеют возможность программировать на Kotlin в своей IDE. Плагин Kotlin для Eclipse поддерживает основную функциональность, такую как навигация и автодополнение кода. Плагин доступен в Eclipse Marketplace. Чтобы установить его, выберите пункт меню **Help > Eclipse Marketplace** (Справка > Eclipse Marketplace) и найдите **Kotlin** в списке.

1.5.5. Онлайн-полигон

Самый простой способ попробовать Kotlin в действии не требует никаких дополнительных установок и настроек. По адресу `http://try.kotl.in` вы найдете онлайн-полигон, где сможете писать, компилировать и запускать небольшие программы на Kotlin. На полигоне представлены примеры кода, демонстрирующие возможности Kotlin (включая все примеры из этой книги), а также ряд упражнений для изучения Kotlin в интерактивном режиме.

1.5.6. Конвертер кода из Java в Kotlin

Освоение нового языка никогда не бывает легким. К счастью, мы создали утилиту, которая поможет быстрее изучить и овладеть им, опираясь на знание языка Java. Это – автоматизированный конвертер кода на Java в код на Kotlin.

В начале изучения конвертер поможет вам запомнить точный синтаксис Kotlin. Напишите фрагмент кода на Java, вставьте его в файл с исходным кодом на Kotlin, и конвертер автоматически предложит перевести этот фрагмент на язык Kotlin. Результат не всегда будет идиоматичным, но это будет рабочий код, с которым вы сможете продвинуться ближе к решению своей задачи.

Конвертер также удобно использовать для внедрения Kotlin в существующий Java-проект. Новый класс можно сразу определить на языке Kotlin. Но если понадобится внести значительные изменения в существующий класс и у вас появится желание использовать Kotlin, конвертер придет вам на выручку. Сначала переведите определение класса на язык Kotlin, а затем сделайте необходимые изменения, используя все преимущества современного языка.

Использовать конвертер в IntelliJ IDEA очень просто. Достаточно просто скопировать фрагмент кода на Java и вставить в файл на Kotlin или выбрать пункт меню **Code** → **Convert Java File to Kotlin** (Код → Преобразовать файл Java в Kotlin), чтобы перевести весь файл на язык Kotlin. Конвертер также доступен в Eclipse и на онлайн-полигоне.

1.6. Резюме

- Kotlin – статически типизированный язык, поддерживающий автоматический вывод типов, что позволяет гарантировать корректность и производительность, сохраняя при этом исходный код лаконичным.
- Kotlin поддерживает как объектно-ориентированный, так и функциональный стиль программирования, позволяя создавать высокоуровневые абстракции с помощью функций, являющихся полноценными объектами, и упрощая разработку и тестирование многопоточных приложений благодаря поддержке неизменяемых значений.
- Язык хорошо подходит для разработки серверных частей приложений, поддерживает все существующие Java-фреймворки и предоставляет новые инструменты для решения типичных задач, таких как создание разметки HTML и операции с хранимыми данными.
- Благодаря компактной среде выполнения, специальной поддержке Android API в компиляторе и богатой библиотеке Kotlin-функций для

решения основных задач Kotlin отлично подходит для разработки под Android.

- Это бесплатный язык с открытым исходным кодом, поддерживаемый основными IDE и системами сборки.
- Kotlin – прагматичный, безопасный, лаконичный и совместимый язык, уделяющий большое внимание возможности использования проверенных решений для популярных задач, предотвращающий распространенные ошибки (такие как исключение `NullPointerException`), позволяющий писать компактный, легко читаемый код и обеспечивающий бесшовную интеграцию с Java.