Оглавление

Посвящается Сандре, поддерживавшей меня все это время.

Твоя доброта, неослабевающая поддержка
и профессиональные навыки продолжают менять мою жизнь

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте **www.dmkpress.com**, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу **dmkpress@gmail.com**.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие

Каждые несколько лет появляется революционно новый язык, обещающий изменить подходы к разработке программного обеспечения. Однако такие обещания редко сбываются. Язык Kotlin – совсем другое дело. С момента создания в 2011 году он медленно и почти незаметно прокрался в базы кода по всему миру. Разработчики, долго использовавшие Java и неоднократно сталкивавшиеся с недостатками этого языка, смогли добавить вкрапления кода на Kotlin и благодаря этому уменьшили размер и увеличили мощность своего кода.

Получив некоторую известность как предпочтительный язык для разработки на Android, Kotlin достиг достаточно высокой степени зрелости, поэтому такая книга крайне необходима. Наполненная множеством полезных советов, «Kotlin. Сборник рецептов» начинает с самого начала. Кен последовательно показывает, как установить Kotlin и настроить его для использования в проекте. Как вызывать код на Kotlin из Java, выполнять его в браузере и в виде отдельного приложения. Но книга быстро движется вперед, описывая приемы решения задач программирования, с которыми каждый день сталкиваются разработчики и архитекторы.

Тестированию кода на Kotlin в книге выделен отдельный раздел, однако вы увидите, что идея тестирования пронизывает книгу от начала до конца. Тесты в книге служат практическими примерами использования языка и позволят вам точнее приспособить рецепты к своим потребностям.

Книга предлагает простую и действенную помощь, которая поможет вам продвинуться по пути освоения Kotlin. Это важное практическое руководство по Kotlin, которое должно лежать на рабочем столе каждого разработчика (реальном или виртуальном).

Дон Гриффитс (Dawn Griffiths) и Дэвид Гриффитс (David Griffiths),
авторы книги «Head First Kotlin»[1]
6 октября 2019 г.

[1] *Дон Гриффитс, Дэвид Гриффитс*. Head First. Kotlin. Питер, 2020. – *Прим. перев.*

Вступление

Добро пожаловать в «Kotlin. Сборник рецептов»! Общая цель книги – не только рассказать и показать синтаксис и семантику Kotlin, но также объяснить, когда и почему следует использовать ту или иную особенность. Книга не стремится охватить все детали синтаксиса Kotlin и представить исчерпывающий список библиотек, но содержит множество практических рецептов решения основных задач и должна быть понятна даже читателям, лишь поверхностно знакомым с Kotlin.

Компания JetBrains активно продвигает в сообщество Kotlin идею мультиплатформенной, нативной разработки и разработки в окружении JavaScript. Но после долгих размышлений было принято решение *не* включать в книгу рецепты, демонстрирующие эти направления, потому что все они или находятся в стадии бета-тестирования, или имеют низкую скорость внедрения. Как результат книга сосредоточена исключительно на Kotlin для JVM.

Код всех примеров можно найти в репозитории GitHub по адресу: https://github.com/kousen/kotlin-cookbook. Он включает оболочку Gradle (с файлом сборки, написанным на Kotlin DSL, разумеется), и все тесты в нем выполняются успешно.

Все примеры кода в книге были скомпилированы и протестированы с обеими доступными версиями Java с долгосрочной поддержкой, а именно с Java 8 и Java 11. Несмотря на то что технически для Java 8 истек срок службы, эта версия по-прежнему широко используется в отрасли, поэтому я решил протестировать примеры кода с ней тоже. На момент написания этой книги текущей была версия Kotlin 1.3.50 и шла работа над версией 1.3.60. Весь код работает с обеими версиями, и репозиторий GitHub будет обновляться для поддержки самой последней версии Kotlin.

Кому адресована эта книга

Эта книга написана для разработчиков, уже знакомых с основами объектно-ориентированного программирования, особенно на Java или другом языке, основанном на JVM. Знание Java пригодится, но не требуется.

Данная книга, как и любые другие книги рецептов, в большей степени ориентирована на описание приемов и идиом Kotlin, чем на исчерпывающее описание языка. Это позволяет без оглядки использовать всю широту возможностей языка в любом рецепте, но ограничивает пространство для описания основ этих возможностей. Каждая глава включает лишь краткое изложение основных методов, поэтому если вы имеете лишь смутное представление о том, как создавать коллекции, работать с массивами или проектировать классы, то не волнуйтесь. Подробное введение в язык вы найдете в справочном онлайн-руководстве (https://kotlinlang.org/docs/reference), и в книге часто упоминаются примеры и обсуждения, имеющиеся в нем.

Кроме того, в книге часто описываются реализации функций из библиотек Kotlin, чтобы показать, как разработчики языка работают с ним на практике, и рассказать, почему что-то делается именно так. Однако наличия у читателя предварительных знаний о реализации не ожидается, и вы можете пропускать эти детали.

Структура книги

Эта книга организована в виде сборника рецептов. Каждый рецепт самодостаточен и независим, но многие из них ссылаются на другие рецепты в книге. В общем и целом их можно читать в любом порядке. Тем не менее главы организованы следующим образом:

- глава 1 описывает процесс установки и запуска Kotlin, включая использование оболочки REPL, работу с инструментами сборки, такими как Maven и Gradle, и использование собственного генератора изображений в Graal;
- глава 2 описывает некоторые фундаментальные возможности и особенности Kotlin, такие как типы с поддержкой `null`, перегрузка операторов и преобразование типов, а затем переходит к исследованию некоторых малоизвестных вопросов, в том числе работы с операторами поразрядного сдвига и с функцией расширения `to` в классе `Pair`;
- глава 3 основное внимание уделяет объектно-ориентированным возможностям языка, которые могут показаться неожиданными или необычными разработчикам на других языках, таким как использование ключевого слова `const`, поддержка свойств, отложенная инициализация и ужасный класс `Nothing`, который гарантированно запутает любого разработчика на Java;
- глава 4 содержит лишь несколько рецептов, демонстрирующих возможности функционального программирования, которые требуют отдельного объяснения. Идеи функционального программирования рассматриваются на протяжении всей книги, особенно в рецептах, использующих коллекции, последовательности и сопрограммы, но в эту главу включено несколько приемов, которые могут показаться необычными и интересными;
- глава 5 охватывает массивы и коллекции, представляя в основном неочевидные методы работы с ними, такие как уничтожение коллекций, сортировка по нескольким свойствам, построение окна для коллекции и создание прогрессий;
- глава 6 описывает, как в Kotlin поддерживается «ленивая» обработка последовательностей элементов, по аналогии с обработкой потоков в Java. Рецепты в этой главе демонстрируют создание последовательностей, получение данных из них и работу с бесконечными последовательностями;
- глава 7 рассматривает еще одну тему, уникальную для Kotlin: функции, выполняющие блок кода в контексте объекта. Такие функции, как `let`, `apply` и `also`, играют очень важную роль в Kotlin, и эта глава рассказывает, почему их следует использовать, и показывает, как это делать;

- глава 8 обсуждает удобную возможность делегирования. Делегирование позволяет использовать композицию вместо наследования, и в стандартной библиотеке самого Kotlin имеется несколько делегатов, таких как `lazy`, `observable` и `vetoable`;
- глава 9 охватывает важную тему тестирования, делая основной упор на Junit 5. Эта текущая версия JUnit прекрасно поддерживает Kotlin и может использоваться для тестирования обычных приложений на Kotlin и кода на Kotlin в приложениях Spring Framework. В этой главе обсуждается несколько подходов, упрощающих написание и выполнение тестов;
- глава 10 включает пару рецептов управления ресурсами. Они демонстрируют приемы работы с файлами, а также использование функции `use`, которая широко применяется в нескольких контекстах;
- глава 11 рассматривает темы, которые трудно отнести к какой-то конкретной категории, в том числе: как получить текущую версию Kotlin, как заставить оператор `when` быть исчерпывающим, даже если он не возвращает значения, и как использовать функцию `replace` с регулярными выражениями. Здесь также обсуждается функция `TODO`, класс `Random` и некоторые способы интеграции с механизмом исключений в Java;
- глава 12 затрагивает вопросы работы с фреймворками Spring Framework и Spring Boot, очень дружелюбными, по отношению к Kotlin. Здесь приводятся рецепты, показывающие, как использовать классы Kotlin в роли управляемых bean-компонентов, как реализовать хранение данных с помощью JPA и как внедрять зависимости;
- глава 13 посвящена сопрограммам, одной из самых популярных особенностей Kotlin, и основам параллельного и конкурентного программирования на этом языке. Рецепты охватывают такие основы, как построители и диспетчеры, а также приемы отмены сопрограмм, их отладки и использования собственного пула потоков Java для их выполнения.

Главы, да и сами рецепты не обязательно читать в каком-либо определенном порядке. Они действительно дополняют друг друга, и каждый рецепт заканчивается ссылками на другие, но вы можете начать читать с любого места в книге. Основная цель глав – объединить похожие рецепты, и они построены так, что вы можете перепрыгивать между рецептами в произвольном порядке, чтобы решить свою задачу, стоящую перед вами в данный момент.

Специальное примечание для разработчиков на Android: в настоящее время Kotlin объявлен предпочтительным языком для разработки на Android, но это гораздо более широкий и универсальный язык программирования. Его можно использовать везде, где применяется Java, и даже шире. В этой книге нет специального раздела, посвященного исключительно Android, и приемы программирования на Kotlin для Android обсуждаются повсюду. Есть несколько конкретных рецептов, связанных с Android, таких как отмена сопрограмм, которые основаны на том факте, что библиотеки Android широко используют Kotlin, но в целом возможности языка, описанные в этой книге, можно использовать где угодно. Есть надежда, что охват языка в более общем плане поможет разработчикам для Android найти приемы, которые пригодятся им в любых других программных проектах.

Соглашения

В этой книге используются следующие соглашения по оформлению:

Курсив

Используется для обозначения новых терминов, адресов URL и электронной почты, имен файлов и расширений.

`Моноширинный шрифт`

Применяется для оформления листингов программ и программных элементов внутри обычного текста, таких как имена переменных и функций, баз данных, типов данных, переменных окружения, инструкций и ключевых слов.

`Моноширинный жирный`

Обозначает команды или другой текст, который должен вводиться пользователем.

`Моноширинный курсив`

Обозначает текст, который должен замещаться фактическими значениями, вводимыми пользователем или определяемыми из контекста.

Так выделяются советы и предложения.

Так обозначаются советы, предложения и примечания общего характера.

Так обозначаются предупреждения и предостережения.

Использование программного кода примеров

Вспомогательные материалы (примеры кода, упражнения и т. д.) доступны для загрузки по адресу: https://github.com/kousen/kotlin-cookbook.

Данная книга призвана оказать вам помощь в решении ваших задач. В общем случае все примеры кода из этой книги вы можете использовать в своих программах и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения примеров из этой книги вам необходимо получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цити-

руя данную книгу или примеры из нее, получение разрешения не требуется. Но при включении существенных объемов программного кода примеров из этой книги в вашу документацию вам необходимо будет получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: «Kotlin Cookbook by Ken Kousen (O'Reilly). Copyright 2020 Ken Kousen, 978-1-492-04667-7».

За получением разрешения на использование значительных объемов программного кода примеров из этой книги обращайтесь по адресу *permissions@oreilly.com*.

O'Reilly Online Learning

Вот уже более 40 лет *O'Reilly Media* (http://oreilly.com/) предоставляет технологии и бизнес-обучение, знания и опыт, помогающие компаниям добиться успеха.

Наше уникальное сообщество экспертов и новаторов делится своими знаниями и опытом через книги, статьи, конференции, а также нашу платформу онлайн-обучения. Платформа онлайн-обучения O'Reilly Online Learning предлагает доступ к очным курсам, углубленным учебным планам, интерактивным средам программирования и обширной коллекции текстовых и видеоматериалов от O'Reilly и более 200 других издателей. За дополнительной информацией обращайтесь по адресу: http://oreilly.com.

Как с нами связаться

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (США или Канада)
707-829-0515 (международный и местный)
707-829-0104 (факс)

На сайте издательства имеется веб-страница этой книги, где можно найти список опечаток в тексте, примеры кода и дополнительную информацию. Страница доступна по адресу: https://oreil.ly/kotlin-cookbook.

Свои пожелания и вопросы технического характера отправляйте по адресу: *bookquestions@oreilly.com*.

Дополнительную информацию о книгах, обучающие курсы, конференции и новости вы найдете на веб-сайте издательства: http://www.oreilly.com.

Ищите нас в Facebook: http://facebook.com/oreilly.

Следуйте за нами в Твиттере: http://twitter.com/oreillymedia.

Смотрите нас на YouTube: http://www.youtube.com/oreillymedia.

Благодарности

На конференции Google I/O в 2017 году компания объявила, что Kotlin будет поддерживаемым языком разработки для Android. Позднее в том же году Gradle Inc. – компания, создавшая инструмент сборки Gradle, – объявила, что будет поддерживать предметно-ориентированный язык (DSL) Gradle для сборки. Оба этих события подтолкнули меня начать исследовать этот язык, и я рад, что сделал это.

Последние несколько лет я регулярно проводил презентации и семинары по Kotlin. Хотя основы языка легко изучить и применить на практике, я был впечатлен его глубиной и тем, насколько быстро он перенимает современные идеи разработки из других языков, таких как Groovy или Scala. Kotlin – это синтез многих передовых идей программирования, и я многому научился, углубившись в исследования перед написанием данной книги.

В процессе изучения я познакомился со многими активными разработчиками Kotlin, включая Дона Гриффитса (Dawn Griffiths) и Дэвида Гриффитса (Dave Griffiths), написавших выдающиеся книги «Head First Android Development»[2] и «Head First Kotlin»[3]; они даже согласились написать предисловие к этой книге. Хади Харрири (Hadi Harriri), технический евангелист JetBrains, регулярно проводит презентации о Kotlin. Его выступления всегда вдохновляют меня уделять время языку, и он оказался настолько любезен, что согласился взять на себя труд технического рецензирования для этой книги. Я очень благодарен им.

Билл Флай (Bill Fly) тоже принял участие в рецензировании книги. Я общался с ним на платформе O'Reilly Learning Platform бесчисленное множество раз, и он всегда подавал интересные идеи (и задавал сложные вопросы). Мой хороший друг Джим Хармон (Jim Harmon) помог мне освоить Android много лет назад и всегда был готов ответить на мои вопросы и рассказать о том, как Kotlin используется на практике. Марк Мейнард (Mark Maynard) – активный разработчик, который помог мне понять, как Kotlin взаимодействует с фреймворком Spring Framework, и я очень благодарен ему за это. Наконец, неподражаемый Венкат Субраманиам (Venkat Subramaniam), написавший свою собственную книгу о Kotlin (озаглавленную «Programming Kotlin» и такую же отменную, как и все остальные его книги), любезно согласился выделить время в своем плотном графике, чтобы помочь мне с моей книгой. Я был рад познакомиться со всеми моими техническими рецензентами, и меня впечатляет, сколько времени и сил они потратили на улучшение книги, которую вы сейчас видите.

Я также хочу поблагодарить многих из моих коллег-докладчиков по туру NFJS, в том числе Нейта Шутту (Nate Schutta), Майкла Кардуччи (Michael Carducci), Мэтта Стайна (Matt Stine), Брайана Слеттена (Brian Sletten), Марка Ричардса (Mark Richards), Пратика Пателя (Pratik Patel), Нила Форда (Neal Ford), Крейга

[2] *Дон Гриффитс и Дэвид Гриффитс.* Head First. Программирование для Android. Питер, 2016. ISBN: 978-5-496-02171-5. – *Прим. перев.*

[3] *Дон Гриффитс и Дэвид Гриффитс.* Head First. Kotlin. Питер, 2020. ISBN: 978-5-4461-1335-4. – *Прим. перев.*

Уоллса (Craig Walls), Раджу Ганди (Raju Gandhi), Джонатана Джонсона (Jonathan Johnson) и Дэна Инохоса (Dan «the Man» Hinojosa), за их постоянные внимание и поддержку. Я наверняка пропустил кого-то в этом перечислении, и если это действительно так, то уверяю вас, что это было сделано не намеренно.

Написание книг и преподавание на учебных курсах (моя основная работа) – это не коллективная работа. Приятно иметь друзей и коллег, на внимание и советы которых я могу рассчитывать.

В создании этой книги приняли участие многие сотрудники O'Reilly Media. Очень непросто перечислить их всех, поэтому я особо упомяну Зана МакКуэйда (Zan McQuade), которого часто ставил в неловкое положение из-за своего нерегулярного графика и моего противоречивого характера. Спасибо тебе за терпение, понимание и упорный труд над этой книгой.

Наконец, я хочу выразить всю свою любовь моей жене Джинджер (Ginger) и моему сыну Ксандеру (Xander). Без поддержки моей семьи я не стал бы тем, кем являюсь сегодня, и этот факт становится для меня все очевиднее с каждым годом. Я никогда не смогу выразить, насколько вы оба дороги мне.

Глава 1

Установка и запуск Kotlin

Рецепты в этой главе помогут вам начать работу с компилятором Kotlin из командной строки и в интегрированной среде разработки (Integrated Development Environment, IDE).

1.1. Запуск Kotlin без локального компилятора

Задача

Опробовать Kotlin без установки на локальный компьютер, например на Chromebook, не поддерживающем такую возможность.

Решение

Использовать Kotlin Playground (https://play.kotlinlang.org/) – онлайн-песочницу для исследования Kotlin.

Обсуждение

Kotlin Playground предлагает простую возможность поэкспериментировать с Kotlin, исследовать его возможности или просто опробовать Kotlin в системах, где отсутствует компилятор этого языка. Эта онлайн-песочница дает доступ к последней версии компилятора, а также к веб-редактору, позволяющему писать и выполнять свой код.

На рис. 1.1 показан скриншот страницы Kotlin Playground в браузере.

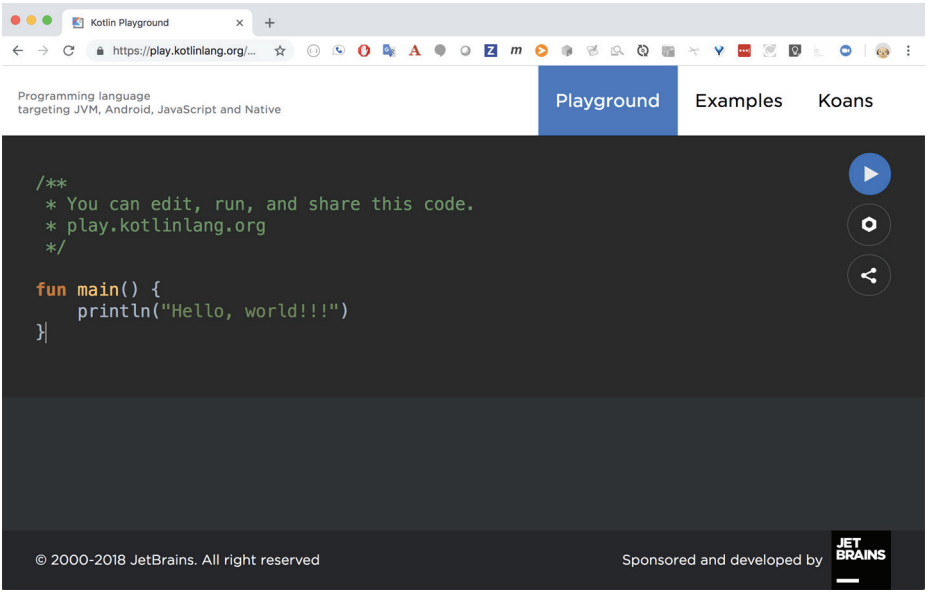

Рис. 1.1. Домашняя страница Kotlin Playground

Просто введите свой код и щелкните на кнопке Run (Запустить), чтобы выполнить его. Кнопка Settings (Настройки), со значком шестеренки, позволяет изменить версию Kotlin, выбрать платформу для запуска (JVM, JS, Canvas или JUnit) или добавить аргументы для программы.

Начиная с версии Kotlin 1.3 функцию main можно объявлять без параметров.

В разделе Examples (Примеры) вы найдете множество примеров программ, организованных по темам, которые можно выполнять прямо в браузере. На рис. 1.2 показана страница с примером программы «Hello world».

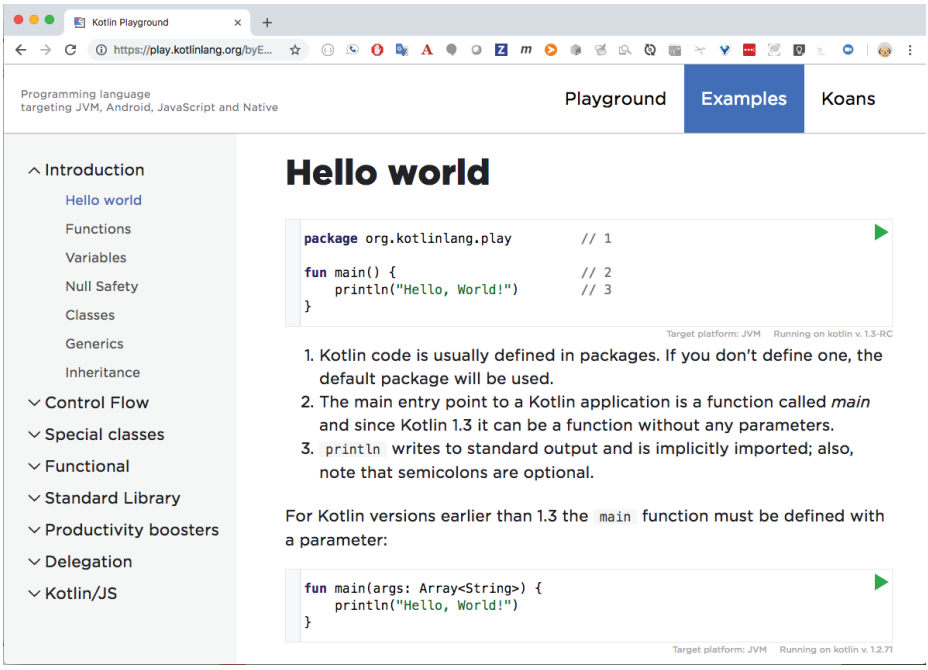

Рис. 1.2. Примеры в Kotlin Playground

В специальном разделе Koans (Задачи) вы найдете серию упражнений, которые помогут вам ближе познакомиться с языком. Упражнения доступны не только в интернете – если вы используете IntelliJ IDEA или Android Studio, то доступ к упражнениям можно получить с помощью плагина EduTools.

1.2. Установка Kotlin на локальный компьютер

Задача

Получить возможность запускать код на Kotlin из командной строки на локальном компьютере.

Решение

Установить компилятор вручную из GitHub или воспользоваться диспетчером пакетов операционной системы.

Обсуждение

На странице http://kotlinlang.org/docs/tutorials/command-line.html описываются возможные варианты установки компилятора командной строки. Один из вариантов – загрузить ZIP-файл с программой установки для своей операционной системы. На этой странице имеется ссылка на репозиторий GitHub (*https://oreil.ly/AXqXM*) с текущими версиями Kotlin. Там вы найдете файлы ZIP с пакетами для Linux, macOS, Windows и с исходным кодом. Просто разархивируйте пакет и добавьте путь к его подкаталогу *bin* в переменную окружения `PATH`.

Установить компилятор вручную несложно, но некоторые разработчики предпочитают использовать диспетчер пакетов. *Диспетчер пакетов* автоматизирует процесс установки, а некоторые из них даже позволяют поддерживать несколько версий компилятора.

SDKMAN!, Scoop и другие диспетчеры пакетов

SDKMAN! (https://sdkman.io/) – одна из самых популярных программ установки пакетов. Первоначально она разрабатывалась для командных оболочек Unix, но уже есть планы создать версии для других платформ.

Установка Kotlin с помощью SDKMAN! начинается с загрузки и установки этой программы с использованием `curl`:

```
> curl -s https://get.sdkman.io | bash
```

После установки следует выполнить команду `sdk`, чтобы установить любой из поддерживаемых продуктов, в число которых входит и Kotlin:

```
> sdk install kotlin
```

По умолчанию устанавливается последняя версия в каталог *~/.sdkman/candidates/kotlin* вместе со ссылкой `current`, указывающей на выбранную версию.

Узнать, какие версии доступны, можно с помощью команды `list`:

```
> sdk list kotlin
```

По умолчанию команда `install` выбирает последнюю версию, но при желании ее можно заставить установить конкретную версию:

```
> sdk use kotlin 1.3.50
```

Эта команда установит версию Kotlin 1.3.50.

IntelliJ IDEA и Android Studio могут использовать и свою собственную, и загруженную версию.

В числе других диспетчеров пакетов, поддерживающих Kotlin, можно назвать: Homebrew (http://brew.sh/), MacPorts (https://www.macports.org/) и Snapcraft (https://snapcraft.io/).

В Windows можно использовать Scoop (https://scoop.sh/). Scoop играет в Windows ту же роль, что другие диспетчеры пакетов в системах, отличных от Windows. Scoop требует наличия в системе PowerShell 5 или выше и .NET Framework 4.5 или выше. Инструкции по установке можно найти на сайте Scoop.

После установки Scoop можно установить текущую версию Kotlin:

```
> scoop install kotlin
```

Эта команда установит сценарии *kotlin.bat*, *kotlinc.bat*, *kotlin-js.bat* и *kotlin-jvm.bat* и добавит путь к ним в переменную окружения PATH.

Этого вполне достаточно, но если вы решите поэкспериментировать, попробуйте экспериментальную программу установки `kotlin-native`, которая также устанавливает собственный компилятор для Windows. При этом еще будет установлен LLVM-компилятор для Kotlin, среда выполнения и инструмент генерации низкоуровневого кода с использованием комплекта инструментов LLVM.

Независимо от способа установки Kotlin, убедиться в его доступности и работоспособности можно с помощью простой команды `kotlin -version`. Вот типичный вывод этой команды:

```
> kotlin -version
Kotlin version 1.3.50-release-112 (JRE 13+33)
```

Смотри также

Рецепт 1.3, где обсуждается, как использовать Kotlin из командной строки после установки.

1.3. Компиляция и выполнение кода на Kotlin из командной строки

Задача

Скомпилировать и выполнить код на Kotlin из командной строки.

Решение

Использовать команды `kotlinc-jvm` и `kotlin`.

Обсуждение

Kotlin SDK для JVM включает команду `kotlinc-jvm` вызова компилятора Kotlin и команду `kotlin` выполнения кода на Kotlin. Они используются подобно командам `javac` и `java` в Java.

Дистрибутив Kotlin включает сценарий `kotlinc-js` для компиляции в JavaScript. В этой книге предполагается использование версии для JVM. Базовый сценарий `kotlinc` – это псевдоним для `kotlinc-jvm`.

Рассмотрим для примера простейшую программу «Hello, Kotlin!». Создайте файл с именем *hello.kt* и добавьте в него код из примера 1.1.

Пример 1.1. hello.kt

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

Команда `kotlinc` скомпилирует этот файл, а команда `kotlin` выполнит полученный файл класса, как показано в примере 1.2.

Пример 1.2. Компиляция и выполнение файла с кодом на Kotlin

```
> kotlinc-jvm hello.kt     ❶

> ls
hello.kt HelloKt.class     ❷

> kotlin HelloKt
Hello, Kotlin!
```

❶ Компиляция исходного кода

❷ Выполнение получившегося файла класса

Компилятор создаст файл *HelloKt.class* с байт-кодом, который можно выполнить в виртуальной машине Java. Kotlin не генерирует исходный код на Java – это не транспилятор. Он генерирует байт-код, который может интерпретироваться виртуальной машиной JVM.

Скомпилированный класс получает имя, соответствующее имени файла, но с первой заглавной буквой, и в конец имени добавляется окончание *Kt*. Этим поведением можно управлять с помощью аннотаций.

Чтобы создать автономный файл JAR, который можно запустить командой `java`, добавьте аргумент `-include-runtime`, как показано в примере 1.3.

Пример 1.3. Включение среды выполнения Kotlin

```
> kotlinc-jvm hello.kt -include-runtime -d hello.jar
```

Эта команда создаст файл *hello.jar*, который можно запустить командой `java`:

```
> java -jar hello.jar
Hello, Kotlin!
```

Без флага `-include-runtime` компилятор создаст JAR-файл, которому необходимо, чтобы среда выполнения Kotlin находилась в пути к классам (classpath).

Команда `kotlinc` без аргументов запускает интерактивную оболочку Kotlin REPL, которая обсуждается в рецепте 1.4.

Смотри также

Рецепт 1.4 демонстрирует, как использовать интерактивную оболочку Kotlin REPL (Read-Eval-Print-Loop – прочитать, вычислить, вывести и повторить). Рецепт 1.5 демонстрирует выполнение сценариев на Kotlin из командной строки.

1.4. Использование Kotlin REPL

Задача

Выполнить код на Kotlin в интерактивной оболочке.

Решение

Запустить Kotlin REPL командой `kotlinc` без аргументов.

Обсуждение

Kotlin включает интерактивную оболочку для работы с компилятором, которая называется REPL и запускается командой `kotlinc` без аргументов. После запуска в оболочке REPL можно вводить произвольные команды Kotlin и сразу же получать результаты.

Оболочка Kotlin REPL также доступна в Android Studio и IntelliJ IDEA в виде пункта меню Tools → Kotlin → Kotlin REPL (Инструменты → Kotlin → Kotlin REPL).

После запуска команды `kotlinc` вы оказываетесь в интерактивной оболочке. В примере 1.4 показан пример сеанса работы в этой оболочке.

Пример 1.4. Использование оболочки Kotlin REPL

```
> kotlinc
Welcome to Kotlin version 1.3.50 (JRE 11.0.4+11)
Type :help for help, :quit for quit
>>> println(“Hello, World!”)
Hello, World!
>>> var name = “Dolly”
>>> println(“Hello, $name!”)
Hello, Dolly!

>>> :help
Available commands:
:help                 show this help
:quit                 exit the interpreter
:dump                 bytecode dump classes to terminal
:load <file>          load script from specified file
>>> :quit
```

Оболочка REPL позволяет легко и быстро вычислять выражения Kotlin без запуска IDE. Используйте ее в случаях, когда нежелательно создавать проект или другую коллекцию файлов в IDE и нужно лишь быстро проверить какую-то идею, помочь другому разработчику либо если IDE вообще отсутствует на компьютере.

1.5. Запуск сценария на Kotlin

Задача

Написать и выполнить сценарий на Kotlin.

Решение

Сохранить код в файле с расширением *.kts* и использовать команду `kotlinc` с параметром `-script`, чтобы запустить его.

Обсуждение

Команда `kotlinc` поддерживает несколько параметров командной строки. Один из них позволяет использовать эту команду как интерпретатор для выполнения сценариев на Kotlin. Сценарий – это текстовый файл с исходным кодом на Kotlin и с расширением *.kts*.

Для демонстрации в примере 1.5 приводится сценарий в файле *southpole.kts*, отображающий текущее время на Южном полюсе и какое время используется – зимнее или летнее. Сценарий использует пакет *java.time*, добавленный в Java 8.

Пример 1.5. southpole.kts

```
import java.time.*

val instant = Instant.now()
val southPole = instant.atZone(ZoneId.of("Antarctica/South_Pole"))
val dst = southPole.zone.rules.isDaylightSavings(instant)
println("It is ${southPole.toLocalTime()} (UTC${southPole.offset}) at the South Pole")
println("The South Pole ${if (dst) "is" else "is not"} on Daylight Savings Time")
```

Запустить этот сценарий можно командой `kotlinc` с параметром `-script`:

```
> kotlinc -script southpole.kts
It is 10:42:56.056729 (UTC+13:00) at the South Pole
The South Pole is on Daylight Savings Time
```

Сценарии содержат код, который обычно помещается в стандартный метод `main` класса. То есть Kotlin можно использовать как язык сценариев для JVM.

1.6. Сборка автономного приложения с помощью GraalVM

Задача

Создать приложение, которое можно запускать из командной строки без применения любых дополнительных зависимостей.

Решение

Использовать компилятор GraalVM и инструмент `native-image`.

Обсуждение

GraalVM (https://www.graalvm.org/) – это высокопроизводительная виртуальная машина с универсальной средой выполнения для запуска приложений, написанных на различных языках. Вы можете написать приложение на любом языке, основанном на JVM, таком как Java или Kotlin, и интегрировать его с программным кодом на JavaScript, Ruby, Python, R и других языках.

Одной из замечательных особенностей GraalVM является возможность использовать ее для создания автономного выполняемого файла. Этот рецепт демонстрирует простой способ использования инструмента `native-image` из GraalVM для создания двоичных файлов из исходного кода на Kotlin.

Получить дистрибутив GraalVM можно по адресу: https://oreil.ly/UcmZD. Для разработки текущего рецепта я установил бесплатную версию Community Edition с помощью диспетчера пакетов SDKMAN!:

```
> sdk install java 19.2.0.1-grl
> java -version
openjdk version «1.8.0_222»
OpenJDK Runtime Environment (build 1.8.0_222-20190711112007.graal.jdk8u-src...
OpenJDK 64-Bit GraalVM CE 19.2.0.1 (build 25.222-b08-jvmci-19.2-b02, mixed mode)

> gu install native-image
// эта команда установит компонент native-image
```

Возьмем за основу Kotlin-версию программы «Hello, World!», изображенную на рис. 1.1, и воспроизведем ее:

```
fun main() {
    println(“Hello, World!”)
}
```

Как отмечалось в рецепте 1.3, этот сценарий легко скомпилировать с помощью `kotlinc-jvm`, получить файл *HelloKt.class*, а затем запустить его командой `kotlin`:

```
> kotlinc-jvm hello.kt // сгенерирует HelloKt.class
> kotlin HelloKt
Hello, World!
```

Но, чтобы получить автономный выполняемый файл, сначала следует скомпилировать сценарий с параметром `-include-runtime` и получить файл *hello.jar*:

```
> kotlinc-jvm hello.kt -include-runtime -d hello.jar
```

А потом с помощью инструмента `native-image`, входящего в состав GraalVM, сгенерировать из него выполняемый файл, как показано в примере 1.6.

Пример 1.6. Создание автономного выполняемого файла с помощью GraalVM

```
> native-image -jar hello.jar
```

Вот выдержка из документации: «Для компиляции *native-image* использует набор инструментов, установленный локально, поэтому в системе должны быть установлены пакеты *glibc-devel*, *zlib-devel* (с файлами заголовков для библиотеки C и zlib) и gcc» .

В процессе работы эта команда выведет следующие строки:

```
> native-image -jar hello.jar
Build on Server(pid: 61247, port: 49590)*
[hello:61247]    classlist:   1,497.63 ms
[hello:61247]        (cap):   2,225.47 ms
[hello:61247]        setup:   3,451.98 ms
[hello:61247]   (typeflow):   2,163.16 ms
[hello:61247]    (objects):   1,793.53 ms
[hello:61247]   (features):     215.90 ms
[hello:61247]     analysis:   4,247.68 ms
[hello:61247]     (clinit):     107.96 ms
[hello:61247]     universe:     399.58 ms
[hello:61247]      (parse):     329.84 ms
[hello:61247]     (inline):     753.12 ms
[hello:61247]    (compile):   3,426.14 ms
[hello:61247]      compile:   4,807.54 ms
[hello:61247]        image:     306.96 ms
[hello:61247]        write:     180.22 ms
[hello:61247]      [total]:  15,246.88 ms
```

В результате будет создан файл *hello*, который можно запустить из командной строки. В Mac или другой Unix-подобной системе для этого достаточно выполнить команду:

```
> ./hello
Hello, World!
```

Теперь мы знаем три способа запуска сценариев на Kotlin:

- скомпилировать его командой `kotlinc-jvm` и затем запустить командой `kotlin`;
- скомпилировать JAR-файл с включением в него среды выполнения и затем выполнить командой `java`;
- скомпилировать командой `kotlinc`, создать двоичный файл с помощью GraalVM и затем выполнить как самую обычную команду.

Размеры файлов, получаемых в этих трех случаях, сильно различаются. Размер скомпилированного файла *HelloKt.class* с байт-кодом составляет около 700 байт. Размер файла *hello.jar* с включенной средой выполнения составляет около 1,2 Мбайт. Автономный двоичный файл получается еще больше – около 2,1 Мбайт. Однако разница в скорости выполнения огромна даже для такого крошечного сценария, как показано в примере 1.7.

Пример 1.7. Хронометраж выполнения сценария hello

```
> time kotlin HelloKt
Hello, World!
kotlin HelloKt 0.13s user 0.05s system 112% cpu 0.157 total

~/Documents/kotlin
> time java -jar hello.jar
Hello, World!
java -jar hello.jar 0.08s user 0.02s system 99% cpu 0.106 total
```

```
~/Documents/kotlin
> time ./hello
Hello, World!
./hello 0.00s user 0.00s system 59% cpu 0.008 total
```

Результаты весьма показательны. JAR-файл выполняется несколько быстрее, чем запуск класса командой `kotlin`, но двоичный файл выполняется на порядок быстрее. В этом примере для его выполнения потребовалось всего около 8 миллисекунд.

Если вы пользуетесь Gradle, то можете использовать плагин поддержки GraalVM с названием `gradle-graal`. Он добавит в вашу систему сборки задачу `gradle-graal` (кроме всего прочего). Подробности смотрите на домашней странице (https://oreil.ly/3eY3Y) плагина.

1.7. Добавление в Gradle плагина поддержки Kotlin (синтаксис Groovy)

Задача

Добавить в систему сборки Gradle плагин поддержки Kotlin, используя синтаксис предметно-ориентированного языка (Domain-Specific Language, DSL) Groovy.

Решение

Добавить в файл сборки зависимость Kotlin и плагин, используя теги Groovy DSL.

Обсуждение

В этом рецепте используется язык Groovy DSL для Gradle. В следующем рецепте будет показано, как использовать Kotlin DSL для Gradle.

Инструмент сборки Gradle (https://gradle.org/) поддерживает компиляцию исходного кода на Kotlin в байт-код JVM с помощью плагина, предлагаемого компанией JetBrains. Плагин `kotlin-gradle-plugin` зарегистрирован в репозитории плагинов Gradle и может быть добавлен в сценарий сборки Gradle, как показано в примере 1.8. Этот код нужно добавить в файл *build.gradle* в корне проекта.

Пример 1-8. Добавление плагина поддержки Kotlin с помощью блока plugins (Groovy DSL)

```
plugins {
    id «org.jetbrains.kotlin.jvm» version «1.3.50»
}
```

Значение `version` представляет одновременно версию плагина и Kotlin. Gradle все еще поддерживает старый синтаксис добавления плагинов, как показано в примере 1.9.

Пример 1.9. Старый синтаксис добавления плагина поддержки Kotlin (Groovy DSL)

```
buildscript {
    repositories {
        mavenCentral()
    }

    dependencies {
        classpath 'org.jetbrains.kotlin:kotlin-gradle-plugin:1.3.50'
    }
}

plugins {
    id «org.jetbrains.kotlin.jvm» version «1.3.50»
}
```

В обоих примерах используется синтаксис Groovy DSL для файлов сборки Gradle, который поддерживает строки в одинарных и в двойных кавычках. В языке Groovy, как и в Kotlin, строки в двойных кавычках поддерживают интерполяцию, но здесь этого не требуется, поэтому двойные кавычки можно заменить одинарными.

Блок `plugins`, в отличие от блока `repositories`, не требует указывать местоположение плагина. Это верно для любого плагина Gradle, зарегистрированного в репозитории плагинов Gradle. Блок `plugins` также автоматически «применяет» плагин, поэтому при его использовании оператор `apply` не требуется.

Файл *settings.gradle* рекомендуется, но не требуется. Он обрабатывается на этапе инициализации, когда Gradle определяет, какие файлы сборки в проекте необходимо проанализировать. В случае сборки сразу нескольких проектов файл настроек показывает, какие подкаталоги в корневом каталоге также являются каталогами проектов. Gradle позволяет использовать общие настройки и зависимости для подпроектов, сделать один подпроект зависимым от другого и даже выполнять сборку подпроектов параллельно. За дополнительной информацией о сборках с несколькими проектами обращайтесь к руководству пользователя Gradle (https://oreil.ly/mwGJW).

Исходный код на Kotlin можно смешивать с исходным кодом на Java в одной папке или хранить их отдельно, в разных папках, например *src/main/java* и *src/main/kotlin*.

Проекты для Android

Плагин поддержки Kotlin для Android работает немного иначе. Проекты для Android – это сборки Gradle с несколькими подпроектами, поэтому они обычно имеют два файла *build.gradle*: один в корневом каталоге и один в подкаталоге с именем по умолчанию *app*. В примере 1.10 показан типичный файл *build.gradle* верхнего уровня, содержащий только информацию о плагине Kotlin.

Пример 1.10. Использование Kotlin в проектах для Android (Groovy DSL)

```
buildscript {
    ext.kotlin_version = '1.3.50'
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.0'
        classpath «org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version»
    }
}
// ... другие задачи, не связанные с плагином ...
```

Затем, выражаясь языком Gradle, плагин *применяется*, как показано в примере 1.11 с типичным файлом *build.gradle* в каталоге *app*.

Пример 1.11. Применение плагина Kotlin

```
apply plugin: 'com.android.application'

apply plugin: 'kotlin-android'                ❶

apply plugin: 'kotlin-android-extensions'     ❷

android {
    // ... android information ...
}

dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin_version" ❸

    // ... other unrelated dependencies ...
}
```

❶ Применяет плагин Kotlin для Android

❷ Применяет расширения для плагина Kotlin для Android

❸ Определение зависимости от стандартной библиотеки, можно использовать JDK 8 или JDK 7

Плагин Kotlin для Android объявляется в разделе `buildscript` и затем применяется в этом файле. Плагин умеет компилировать код на Kotlin внутри Android-приложения. В состав загружаемого плагина также входит пакет расширений для Android, которые упрощают доступ к виджетам Android по их идентификаторам.

Плагин поддержки Kotlin может генерировать байт-код для JDK 7 или JDK 8. Измените значение `jdk` в указанной зависимости, чтобы выбрать предпочитаемую версию.

На момент написания этой книги нельзя было выбрать Kotlin DSL при создании проекта для Android. Конечно, можно создать файлы сборки вручную и использовать в них Kotlin DSL, но этот подход редко применяется на практике. Kotlin DSL будет доступен в версии Android Studio 4.0, которая также будет включать полную поддержку файлов KTS и «живых шаблонов» Kotlin.

Смотри также

Тот же процесс с использованием Kotlin DSL, кроме раздела для Android, показан в рецепте 1.8.

1.8. Добавление в Gradle плагина поддержки Kotlin (синтаксис Kotlin)

Задача

Добавить в систему сборки Gradle плагин поддержки Kotlin, используя синтаксис Kotlin DSL.

Решение

Добавить в файл сборки зависимость Kotlin и плагин, используя теги Kotlin DSL.

Обсуждение

В этом рецепте применяется язык Kotlin DSL для Gradle. В предыдущем рецепте показано, как использовать Groovy DSL для Gradle.

Версия Gradle 5.0 и выше включает новый язык Kotlin DSL для настройки файлов сборки. В них также доступен плагин `kotlin-gradle-plugin`, зарегистрированный в репозитории плагинов Gradle, который можно добавить в сценарий сборки Gradle, как показано в примере 1.12. В качестве альтернативы можно использовать старый синтаксис `buildscript` (см. пример 1.13). Этот код нужно добавить в файл *build.gradle.kts* в корне проекта.

Пример 1-12. Добавление плагина поддержки Kotlin с помощью блока plugins (Kotlin DSL)

```
plugins {
    kotlin(«jvm») version «1.3.50»
}
```

Пример 1.13. Старый синтаксис добавления плагина поддержки Kotlin (Kotlin DSL)

```
buildscript {
    repositories {
        mavenCentral()
    }

    dependencies {
        classpath(kotlin(«gradle-plugin», version = «1.3.50»))
    }
}

plugins {
    kotlin(“jvm”)
}
```

Блок `plugins`, в отличие от блока `repositories`, не требует указывать местоположение плагина. Это верно для любого плагина Gradle, зарегистрированного в репозитории плагинов Gradle. Блок `plugins` также автоматически «применяет» плагин, поэтому при его использовании оператор `apply` не требуется.

По умолчанию файлам сборки на Kotlin DSL в Gradle даются имена *settings.gradle.kts* и *build.gradle.kts*.

Как можно заметить, самые большие отличия от Groovy DSL заключаются в следующем:

- все строки должны заключаться в двойные кавычки;
- в Kotlin DSL необходимо использовать круглые скобки;
- присваивание в Kotlin определяется знаком равенства (=), а не двоеточием (:).

Файл *settings.gradle.kts* рекомендуется, но не требуется. Он обрабатывается на этапе инициализации, когда Gradle определяет, какие файлы сборки в проекте необходимо проанализировать. В случае сборки сразу нескольких проектов файл настроек показывает, какие подкаталоги в корневом каталоге также являются каталогами проектов. Gradle позволяет использовать общие настройки и зависимости для подпроектов, сделать один подпроект зависимым от другого и даже выполнять сборку подпроектов параллельно. За дополнительной информацией о сборках с несколькими проектами обращайтесь к руководству пользователя Gradle (https://oreil.ly/XG4EN).

Исходный код на Kotlin можно смешивать с исходным кодом на Java в папках *src/main/java* и *src/main/kotlin*, или можно добавить свои собственные исходные файлы, используя свойство `sourceSets` в Gradle. За подробностями обращайтесь к документации Gradle (https://oreil.ly/XG4EN).

Смотри также

Тот же процесс с применением Groovy DSL показан в рецепте 1.7. Там же вы найдете дополнительные сведения о проектах для Android, где в настоящее время Kotlin DSL недоступен для выбора при создании проектов Android.

1.9. Сборка проектов на Kotlin с помощью Gradle

Задача

Собрать проект с кодом на Kotlin, используя Gradle.

Решение

Добавить зависимость от Kotlin JDK на этапе компиляции, помимо плагина Kotlin.

Обсуждение

Примеры в рецептах 1.7 и 1.8 показывают, как добавить плагин поддержки Kotlin для Gradle. Этот рецепт демонстрирует добавление новых возможностей в файл сборки для обработки любого кода на Kotlin в проекте.

Чтобы скомпилировать код Kotlin с помощью Gradle, нужно добавить еще один элемент в блок `dependencies`, как показано в примере 1.14.

Пример 1.14. Файл build.gradle.kts на Kotlin DSL для простого проекта

```
plugins {
    `java-library`                  ❶
    kotlin("jvm") version "1.3.50" ❷
}

repositories {
    jcenter()
}

dependencies {
    implementation(kotlin("stdlib")) ❸
}
```

❶ Добавляет задачи из плагина Java Library

❷ Добавляет плагин Kotlin в Gradle

❸ Добавляет стандартную библиотеку Kotlin в проект

Плагин `java-library` определяет задачи для простых проектов JVM, такие как `build`, `compileJava`, `compileTestJava`, `javadoc`, `jar` и другие.

Раздел `plugins` должен следовать первым, но остальные блоки верхнего уровня (`repositories`, `dependencies` и т. д.) могут располагаться в любом порядке.

Блок `dependencies` добавляет стандартную библиотеку Kotlin на этапе компиляции (чтобы добиться того же эффекта в старой версии Gradle, можно использовать конфигурацию `compile` вместо `implementation`). Блок `repositories` указывает, что зависимость Kotlin будет загружена из `jcenter` – общедоступного репозитория Artifactory Bintray.

Если теперь выполнить команду `gradle build --dry-run`, она перечислит задачи, доступные для выполнения, но не выполнит их:

```
> gradle build -m

:compileKotlin SKIPPED
:compileJava SKIPPED
:processResources SKIPPED
:classes SKIPPED
:inspectClassesForKotlinIC SKIPPED
:jar SKIPPED
:assemble SKIPPED
:compileTestKotlin SKIPPED
:compileTestJava SKIPPED
```

```
:processTestResources SKIPPED
:testClasses SKIPPED
:test SKIPPED
:check SKIPPED
:build SKIPPED

BUILD SUCCESSFUL in 0s
```

Плагин Kotlin добавляет задачи `compileKotlin`, `inspectClassesForKotlinIC` и `compileTestKotlin`.

Собрать проект можно той же командой, опустив параметр `-m`, который является синонимом параметра `--dry-run`.

1.10. Использование Maven с Kotlin

Задача

Скомпилировать код на Kotlin с помощью инструмента сборки Maven.

Решение

Использовать плагин поддержки Kotlin для Maven и добавить стандартную библиотеку в зависимости.

Обсуждение

Основные сведения о Maven можно найти на веб-странице с документацией (https://oreil.ly/LLy3h).

В документации рекомендуется сначала указать версию Kotlin в Maven-файле *pom.xml*, как показано ниже:

```
<properties>
    <kotlin.version>1.3.50</kotlin.version>
</properties>
```

Затем добавить в зависимости стандартную библиотеку Kotlin, как показано в примере 1.15.

Пример 1.15. Добавление стандартной библиотеки Kotlin в зависимости

```
<dependencies>
    <dependency>
        <groupId>org.jetbrains.kotlin</groupId>
        <artifactId>kotlin-stdlib</artifactId>
        <version>${kotlin.version}</version>
    </dependency>
</dependencies>
```

Так же как в Gradle, можно указать `kotlin-stdlib-jdk7` или `kotlin-stdlib-jdk8`, чтобы использовать функции расширения для Java 1.7 или 1.8 соответственно.

Кроме того, можно использовать артефакты `kotlin-reflect` (поддержка рефлексии) и `kotlin-test` с `kotlin-test-junit` (поддержка тестирования).

Чтобы скомпилировать исходный код на Kotlin, нужно сообщить Maven, в каких каталогах он находится, как показано в примере 1.16.

Пример 1.16. Определение каталогов с исходным кодом на Kotlin

```
<build>
    <sourceDirectory>${project.basedir}/src/main/kotlin</sourceDirectory>
    <testSourceDirectory>${project.basedir}/src/test/kotlin</testSourceDirectory>
</build>
```

И использовать плагин Kotlin для компиляции и тестирования (пример 1.17).

Пример 1.17. Использование плагина Kotlin

```
<build>
    <plugins>
        <plugin>
            <artifactId>kotlin-maven-plugin</artifactId>
            <groupId>org.jetbrains.kotlin</groupId>
            <version>${kotlin.version}</version>

            <executions>
                <execution>
                    <id>compile</id>
                    <goals><goal>compile</goal></goals>
                </execution>

                <execution>
                    <id>test-compile</id>
                    <goals><goal>test-compile</goal></goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Если проект содержит исходный код на Kotlin и на Java, первым должен компилироваться код на Kotlin. То есть плагин `kotlin-maven-plugin` должен запускаться перед `maven-compiler-plugin`. В упоминавшейся выше документации показано, как это организовать с помощью параметров конфигурации в файле *pom.xml*.