

Привет, Ксандер, это тебе посвящается. Сюрприз!

Содержание

Предисловие	9
Вступление	11
Глава 1. Основные понятия	18
1.1. Лямбда-выражения	19
1.2. Ссылки на методы	22
1.3. Ссылки на конструкторы	26
1.4. Функциональные интерфейсы	30
1.5. Методы по умолчанию в интерфейсах	33
1.6. Статические методы в интерфейсах	36
Глава 2. Пакет java.util.function	39
2.1. Потребители	39
2.2. Поставщики	42
2.3. Предикаты	44
2.4. Функции	48
Глава 3. Потоки	51
3.1. Создание потоков	51
3.2. Оборнутые потоки	55
3.3. Операции редукции	57
3.4. Проверка правильности сортировки с помощью редукции	65
3.5. Отладка потоков с помощью reek	66
3.6. Преобразование строк в потоки и наоборот	69
3.7. Подсчет элементов	72
3.8. Сводные статистики	73
3.9. Нахождение первого элемента в потоке	76
3.10. Методы anyMatch, allMatch и noneMatch	81
3.11. Методы flatMap и map	83
3.12. Конкатенация потоков	85
3.13. Ленивые потоки	89

Глава 4. Компараторы и коллекторы	91
4.1. Сортировка с помощью компаратора.....	91
4.2. Преобразование потока в коллекцию	95
4.3. Добавление линейной коллекции в отображение	97
4.4. Сортировка отображений.....	99
4.5. Разбиение и группировка.....	102
4.6. Подчиненные коллекторы.....	104
4.7. Нахождение минимального и максимального значений.....	106
4.8. Создание неизменяемых коллекций	109
4.9. Реализация интерфейса Collector	111
 Глава 5. Применение потоков, лямбда-выражений и ссылок на методы	115
5.1. Класс java.util.Objects	115
5.2. Лямбда-выражения и эффективная финальность.....	117
5.3. Потоки случайных чисел	120
5.4. Методы по умолчанию интерфейса Map.....	122
5.5. Конфликт между методами по умолчанию.....	126
5.6. Обход коллекций и отображений	128
5.7. Протоколирование с помощью Supplier	130
5.8. Композиция замыканий.....	132
5.9. Применение вынесенного метода для обработки исключений	135
5.10. Контролируемые исключения и лямбда-выражения	138
5.11. Использование универсальной обертки исключений.....	141
 Глава 6. Тип Optional	143
6.1. Создание Optional	143
6.2. Извлечение значения из Optional.....	146
6.3. Optional в методах чтения и установки.....	149
6.4. Методы flatMap и map класса Optional	150
6.5. Отображение объектов Optional	154
 Глава 7. Файловый ввод-вывод	157
7.1. Обработка файлов	158
7.2. Получение файлов в виде потока.....	160
7.3. Обход файловой системы	161
7.4. Поиск в файловой системе	163

Глава 8. Пакет java.time	165
8.1. Основные классы для работы с датами и временем	166
8.2. Создание даты и времени на основе существующих экземпляров	169
8.3. Корректоры и запросы	173
8.4. Преобразование java.util.Date в java.time.LocalDate	178
8.5. Разбор и форматирование	182
8.6. Нахождение часовых поясов с необычным смещением	185
8.7. Нахождение названий регионов по смещению	187
8.8. Время между событиями	189
 Глава 9. Параллелизм и конкурентность	192
9.1. Преобразование последовательного потока в параллельный	193
9.2. Когда распараллеливание помогает	196
9.3. Изменение размера пула	201
9.4. Интерфейс Future	203
9.5. Завершение CompletableFuture	206
9.6. Координация нескольких CompletableFuture, часть 1	210
9.7. Координация нескольких CompletableFuture, часть 2	215
 Глава 10. Нововведения в Java 9	222
10.1. Модули в проекте Jigsaw	223
10.2. Закрытые методы в интерфейсах	227
10.3. Создание неизменяемых коллекций	229
10.4. Интерфейс Stream: ofNullable, iterate, takeWhile и dropWhile	233
10.5. Подчиненные коллекторы: filtering и flatMapping	236
10.6. Класс Optional: методы stream, or, ifPresentOrElse	240
10.7. Диапазоны дат	243
 Приложение А. Универсальные типы и Java 8	246
Общие сведения	246
Что знает каждый	246
Чего некоторые разработчики не осознают	249
Метатипы и PECS	250
Примеры из Java 8 API	255
Резюме	262
 Предметный указатель	263
Об авторе	272
Об иллюстрации на обложке	273

Предисловие

Несомненно, новые возможности, появившиеся в Java 8, а особенно лямбда-выражения и Streams API, – гигантский шаг вперед. Вот уже несколько лет я пользуюсь версией Java 8 и рассказываю о новшествах разработчикам на конференциях, семинарах и в своем блоге. И мне совершенно ясно, что хотя лямбда-выражения и потоки приносят в Java функциональный стиль программирования (а заодно позволяют органично использовать возможности распараллеливания), не это привлекает разработчиков, впервые начинающих работать с ними, а то, насколько проще с их помощью становится решать некоторые типы задач и как эти идиомы повышают производительность труда программиста.

Мне как разработчику, лектору и писателю особенно интересно не просто рассказывать об эволюции языка Java, а демонстрировать, как эта эволюция упрощает нашу жизнь – как вновь появившиеся средства позволяют проще решать не только известные, но и совсем новые задачи. И в работе Кена меня подкупает именно это – стремление научить новому, не мусоля детали, которые вам уже известны или не нужны, а сосредоточившись на тех аспектах технологии, которые ценны для программистов-практиков.

Впервые я познакомился с подходом Кена, когда он презентовал свою книгу «Making Java Groovy» на конференции JavaOne. В то время наша команда как раз пыталась написать понятные и полезные тесты, и одним из рассматриваемых вариантов было применение Groovy. Будучи ветераном программирования на Java, я не очень хотел изучать совсем новый язык только для того, чтобы написать тесты, тем более что мне казалось, что уж, как писать тесты, я знаю. Но, послушав, как Кен рассказывает о Groovy для Java-программистов, я узнал много нового и полезного, не отвлекаясь на вещи, которые и так хорошо понимал. И понял, что если подойти к изучению материала правильно, то не придется продираться сквозь дебри языка только для того, чтобы узнать то небольшое, что мне действительно необходимо. Я немедленно купил его книгу.

И эта книга о рецептах программирования на современном Java написана с тех же позиций – нам, опытным программистам, нет нужды изучать все возможности Java 8 и 9 так, будто это какой-то новый для нас язык, да и времени на это не хватит. Что нам нужно, так что быстро понять, что появилось интересного, и увидеть примеры реального кода, которые можно было бы применить в своих программах. Именно так и построена книга. На примерах рецептов решения повседневных задач с использованием новых средств Java 8 и 9 автор знакомит нас с языковыми нововведениями самым естественным для нас способом, так чтобы мы могли обогатить свой арсенал.

Прочитав раздел об операторах редукции, я действительно «врубился» в функциональный стиль программирования, не пытаюсь перепрограммировать собственные мозги. Рассмотренные возможности Java 9 – это именно то, что полезно нам, разработчикам, и (пока) не очень хорошо известно. Это прекрасный способ познакомиться с последней версией Java быстро и эффективно. Любой пишущий на Java программист найдет в этой книге что-то такое, что позволит повысить свой уровень.

*Триша Джи,
Java Champion и Java Developer Advocate
в компании JetBrains,
июль 2017 г.*

Вступление

СОВРЕМЕННЫЙ JAVA

Иногда трудно поверить, что язык, имеющий 20-летнюю историю обратной совместимости, мог так радикально измениться. До выхода версии Java SE 8 в марте 2014-го¹ Java, несмотря на все успехи в качестве языка программирования серверов, пользовался репутацией «COBOL'a XXI века». Стабильный, вездесущий, ориентированный на производительность. Изменения происходили медленно, если вообще происходили. Компании не особенно торопились переходить на новые версии, когда они наконец появлялись.

Все изменилось с выходом Java SE 8. В эту версию был включен «проект лямбда», главное нововведение, которое приносило идеи функционального программирования в то, что многие считали самым распространенным в мире языком объектно-ориентированного программирования. Лямбда-выражения, ссылки на методы и потоки принципиально изменили идиомы языка, и с тех пор разработчики стремятся не отстать от времени.

В этой книге я не пытаюсь судить, хорошо ли то, что случилось, или плохо, и что можно было бы сделать по-другому. Принят другой подход: «вот что мы имеем и вот как этим можно воспользоваться во благо». Поэтому книга построена как сборник рецептов. Она о том, какую задачу требуется решить и как этому могут помочь новые средства Java.

Но нельзя не сказать о том, что у новой модели программирования немало достоинств, нужно к ним только привыкнуть. Функциональный код часто оказывается проще писать и читать. Функциональный подход поощряет неизменяемость, благодаря чему конкурентный код становится чище и с большей вероятностью правильнее. Когда язык Java только создавался, мы еще могли полагаться на закон Мура, гласящий, что быстродействие процессоров удваивается примерно каждые 18 месяцев. Но в наши дни повышение производительности обусловлено тем фактом, что даже большинство современных смартфонов оснащено несколькими процессорами.

Поскольку разработчики Java всегда уделяли первостепенное внимание обратной совместимости, многие компании и программисты перешли на Java SE 8, не дав себе труда освоить новые идиомы. Платформа-то в любом случае стала более эффективной, так что перейти на нее стоило, даже если забыть о том, что корпорация Oracle формально объявила апрель 2015-го датой кончины Java 7.

¹ Да, с тех пор прошло уже больше трех лет, просто не верится.

Понадобилось два года, но сейчас большинство Java-разработчиков использует Java 8 JDK, и пришло время разобраться в том, что же это означает и какие последствия несет для будущих проектов. Книга, которую вы держите в руках, поможет в этом.

Кому стоит прочитать эту книгу?

Приведенные в книге рецепты рассчитаны на читателя, который хорошо знаком с версиями Java, предшествующими Java SE 8. Экспертом быть необязательно, и о некоторых старых концепциях мы напомним, но книга определенно не является руководством по Java или объектно-ориентированному программированию для начинающих. Если вы уже писали проекты на Java и знакомы со стандартной библиотекой, то все нормально.

В книге охвачено почти все, что есть в Java SE 8, а одна глава целиком посвящена нововведениям в Java 9. Если вам интересно, как новые функциональные идиомы, добавленные в язык, изменяют подход к написанию кода, то книга поможет разобраться в этом на конкретных примерах.

Java повсеместно используется на стороне сервера и располагает богатой поддержкой в виде библиотек и инструментов с открытым исходным кодом. Каркасы Spring Framework и Hibernate относятся к числу наиболее популярных, и оба уже требуют или будут требовать в ближайшем будущем как минимум Java 8. Если вы планируете работать в этой экосистеме, то эта книга для вас.

О СТРУКТУРЕ КНИГИ

Книга представляет собой набор рецептов, но вряд ли возможно изложить рецепты, относящиеся к лямбда-выражениям, ссылкам на методы и потокам, так чтобы они не ссылались друг на друга. На самом деле в первых шести главах обсуждаются взаимосвязанные концепции, хотя читать их можно в любом порядке.

В главе 1 «Основные понятия» рассмотрены лямбда-выражения и ссылки на методы, а также новые возможности интерфейсов: методы по умолчанию и статические методы. Здесь же определен термин «функциональный интерфейс» и объяснено, почему он так важен для понимания лямбда-выражений.

В главе 2 «Пакет `java.util.function`» представлен новый пакет `java.util.function`, добавленный в Java 8. Интерфейсы, входящие в этот пакет, распределены по четырем категориям (потребители, поставщики, предикаты и функции) и используются во всей стандартной библиотеке.

В главе 3 «Потоки» вводятся понятие потока и та абстракция, которая позволяет использовать потоки для преобразования и фильтрации данных вместо итеративной обработки. В рецептах из этой главы рассматриваются связанные с потоками концепции «отображения», «фильтрации» и «редукции», которые ведут к идеям параллелизма и конкурентности, составляющим содержание главы 9.

В главе 4 «Компараторы и коллекторы» рассматриваются сортировка потоковых данных и преобразование их в коллекции. Сюда же включены операции разбиения и группировки, которые обычно считаются операциями базы данных, но представлены в виде простых библиотечных вызовов.

Глава 5 «Применение потоков, лямбда-выражений и ссылок на методы» – сборная солянка. Идея в том, что раз вы уже знаете, как использовать лямбда-выражения, ссылки на методы и потоки, то не худо бы посмотреть, как их комбинирование позволяет решать интересные задачи. Также рассматриваются отложенное (ленивое) выполнение, композиция замыканий и набившая оскомину тема обработки исключений.

В главе 6 «Тип `Optional`» обсуждается одно из самых спорных добавлений в язык – тип `Optional`. Описано, как предполагается использовать этот тип и как создавать экземпляры этого типа и получать хранящиеся в них значения. Мы также вернемся к функциональным операциям `map` и `flat-map` в применении к объектам типа `Optional` и обсудим, чем они отличаются от аналогичных операций для потоков.

В главе 7 «Файловый ввод-вывод» мы перейдем к практическому вопросу о потоках ввода-вывода (в отличие от функциональных потоков) и тем добавлениям в стандартную библиотеку, которые привносят новые функциональные идеи в работу с файлами и каталогами.

В главе 8 «Пакет `java.time`» описаны основы нового API для работы с датами и временем и рассказано, как он (наконец-то) заменяет унаследованные классы `Date` и `Calendar`. Новый API основан на библиотеке `Joda-Time`, за которой стоят многие человеко-годы разработки и использования и которая теперь переписана в виде пакета `java.time`. Откровенно говоря, даже если бы это было единственное добавление Java 8, оно уже оправдало бы переход на эту версию.

Глава 9 «Параллелизм и конкурентность» посвящена одному из неявных обещаний потоковой модели: что мы можем одним вызовом метода превратить последовательный поток в параллельный и воспользоваться всеми имеющимися процессорами. Конкурентность – обширная тема, но в этой главе описаны те добавления в библиотеку Java, которые упрощают экспериментирование и позволяют оценить, стоит ли игра свеч.

В главе 10 «Нововведения в Java 9» рассматриваются многие изменения, включенные в версию Java 9, выход которой был намечен на 21 сентября 2017 г. Детали проекта `Jigsaw` заслуживают отдельной книги¹, но основные элементы понятны и описаны в этой главе. Здесь же рассмотрены закрытые методы в интерфейсах, новые методы потоков, коллекторов и типа `Optional`, а также вопрос о создании потока дат².

¹ И такая книга есть: *Кишори Шаран. Java 9. Полный обзор нововведений*. М.: ДМК, 2017. – *Прим. перев.*

² Да, я тоже хотел бы, чтобы версии Java 9 была посвящена глава 9, но изменять логичный порядок глав ради этой, не столь существенной симметрии было бы неправильно. Достаточно и этой сноски.

Приложение А «Универсальные типы и Java 8» посвящено механизмам универсальности в Java. Хотя универсальные типы как технология были включены еще в версию 1.5, большинство разработчиков изучило только тот минимум, который необходим для работы с ними. Но даже беглое знакомство с официальной документацией по Java 8 и 9 убеждает, что эти дни давно миновали. Цель этого приложения – показать, как читать и интерпретировать API, чтобы разобраться в ставших гораздо более сложными сигнатурах методов.

Главы и, конечно, сами рецепты необязательно читать в каком-то определенном порядке. Да, они дополняют друг друга, и в конце каждого рецепта даются ссылки на другие рецепты, но начинать можно с любого места. Деление на главы позволило собрать схожие рецепты в одном месте, но предполагается, что вы будете перескакивать из одного места в другое, чтобы найти решение стоящей в данный момент задачи.

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ И СОГЛАШЕНИЯ, ПРИНЯТЫЕ В КНИГЕ

В книге используются следующие типографские соглашения:

Курсив

Используется для смыслового выделения важных положений, новых терминов, имен команд и утилит, а также имен и расширений файлов и каталогов.

Моноширинный шрифт

Используется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

Моноширинный полужирный шрифт

Используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений.

Моноширинный курсив

Используется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.



Такая пиктограмма обозначает совет или рекомендацию.



Такая пиктограмма обозначает указание или примечание общего характера.



Эта пиктограмма обозначает предупреждение или предостережение.

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 прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы 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.

СКАЧИВАНИЕ ИСХОДНОГО КОДА ПРИМЕРОВ

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии 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. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

БЛАГОДАРНОСТИ

Эта книга стала неожиданным результатом моего разговора с Джем Циммерманном в июле 2015 года. Я принимал (и до сих пор принимаю) участие в конференциях No Fluff, Just Stuff, а в тот год с несколькими докладами по Java 8 выступал Венкат Субраманиам. Джей сказал мне, что Венкат решил снизить активность в будущем году, и поинтересовался, не хотел бы я сменить его в новом сезоне, стартующем в начале 2016 года. Я программировал на Java с середины 1990-х годов и в любом случае собирался изучить новые API, поэтому согласился.

С тех пор я вот уже два года провожу презентации по новым функциональным возможностям Java. Осенью 2016-го я закончил свою последнюю книгу¹ и, поскольку собирался писать еще один сборник рецептов для того же самого издательства, сгоряча решил, что проект будет легким.

Известный писатель-фантаст Нил Гейман как-то сказал, что ему казалось, будто после завершения «Американских богов» он знает, как писать романы. Но один приятель поправил его, заметив, что теперь он знает, как писать *этот* роман. Сейчас я понимаю, что он имел в виду. Первоначально предполагалось, что книга будет содержать 25–30 рецептов и насчитывать примерно 150 страниц. В итоге рецептов оказалось больше 70 и занимают они почти 300 страниц, но благодаря увеличению охвата материала и количества деталей книга получилась куда более ценной, чем я ожидал.

Конечно, все это стало возможным, потому что я был не один. Уже упомянутый Венкат Субраманиам очень помог мне и докладами, и своими книгами, и частными беседами. Он также любезно согласился выступить в роли технического редактора, так что если остались какие-то ошибки, то только по его вине. (Шучу, конечно, все ошибки мои, но не говорите ему, что я это признал.)

Я также весьма ценю помощь, которую мне часто оказывал Тим Йейтс (Tim Yates), один из лучших известных мне кодировщиков. Я знаю его по работе в сообществе Groovy, но его интересы гораздо шире, о чем красноречиво свидетельствует его рейтинг на сайте Stack Overflow. Род Хилтон, с которым я повстречался, когда делал доклад о Java 8 на одной из конференций NFJS, также любезно предложил отрецензировать рукопись. Рекомендации того и другого были просто бесценны.

Мне повезло работать с отличными редакторами и персоналом издательства O'Reilly, где вышли две мои книги, с дюжины видеокурсов и много онлайн-уроков, размещенных на разработанной издательством платформе Safari. Брайан Фостер неизменно поддерживал меня и к тому же обладает фантастической способностью устранять бюрократические препоны. Я познакомился с ним во время работы над предыдущей книгой, и, хотя не он был редактором

¹ «Gradle Recipes for Android», также вышедшую в O'Reilly Media и посвященную применению инструмента сборки Gradle к проектам для Android.

этой, его помощь и дружеское участие на протяжении всего процесса были очень важны для меня.

Мой редактор Джефф Блейел выказал полное понимание, видя, как книга растет в объеме, и обеспечил всю организационную структуру, необходимую для продолжения работы. Я очень доволен нашей совместной работой и надеюсь продолжить ее в будущем.

Хочу выразить признательность другим докладчикам на конференциях NFJS: Нейту Шутта (Nate Schutta), Майклу Кардуччи (Michael Carducci), Мэту Стайну (Matt Stine), Брайану Слеттену (Brian Sletten), Марку Ричардсу (Mark Richards), Пратику Пателю (Pratik Patel), Нилу Форду (Neal Ford), Крейгу Уоллсу (Craig Walls), Раджу Ганди (Raju Gandhi), Кирку Кноерншильду (Kirk Knoernschild), Дэну «the Man» Инойоза (Dan «the Man» Hinojosa) и Джанелл Клейн (Janelle Klein) – за постоянную целеустремленность и поддержку. Написание книг и преподавание на курсах (моя повседневная деятельность) – работа, которую выполняешь в одиночестве, и тем важнее иметь друзей и единомышленников, с которыми можно поделиться своими мыслями, получить совет и вместе поразвлечься.

И наконец, выражаю бесконечную любовь своей жене Джинджер и сыну Ксандеру. Без поддержки и сердечного отношения семьи я не был бы тем, кем стал, и с каждым годом это становится для меня все более очевидно. Не могу выразить словами, как много вы значите для меня.

Глава 1

Основные понятия

Самое важное изменение в Java 8 – включение в язык ряда концепций функционального программирования: лямбда-выражений, ссылок на методы и потоков.

Если вы еще не пользовались функциональными средствами, то, вероятно, будете удивлены тем, как сильно ваш код станет отличаться от того, что вы писали в прошлом. Java 8 знаменует самое сильное изменение языка за всю его историю. Иногда складывается впечатление, что изучаешь совсем новый язык.

Возникает вопрос: а зачем это нужно? К чему такие радикальные изменения в языке, которому уже исполнилось двадцать лет и который планирует обеспечивать обратную совместимость? Зачем переходить на функциональную парадигму в языке, который считается одним из самых успешных когда-либо созданных объектно-ориентированных языков?

Все дело в том, что мир разработки программного обеспечения меняется, и если язык хочет остаться успешным, то должен адаптироваться к новому. В середине 1990-х годов, когда Java еще сверкала новизной, действовал закон Мура¹. Надо было подождать каких-то два года, чтобы быстродействие вашего компьютера возросло в два раза.

Но сегодня рост быстродействия оборудования зависит не от плотности упаковки на кристалле. Ныне даже телефоны оборудованы несколькими процессорными ядрами, а значит, программы нужно писать с учетом того, что они будут исполняться в многопроцессорной среде. Функциональное программирование с его вниманием к «чистым» функциям (которые возвращают одинаковый результат при одних и тех же входных данных и не имеют побочных эффектов) и неизменяемости упрощает создание программ, допускающих распараллеливание. Если отсутствует разделяемое изменяемое состояние и программу можно разложить на несколько простых функций, то ее поведение проще понять и предсказать.

¹ Сформулированный Гордоном Муром, одним из основателей компании Fairchild Semiconductor, этот закон основан на том наблюдении, что количество транзисторов в интегральной схеме удваивается примерно каждые 18 месяцев. Детали см. в статье Википедии о законе Мура.

Но это не книга о Haskell, Erlang, Frege или еще каком-то функциональном языке программирования. Это книга о Java и о тех изменениях, которые позволили включить функциональные концепции в язык, остающийся в основе своей объектно-ориентированным.

Теперь Java поддерживает лямбда-выражения, т. е., по существу, методы, которые рассматриваются как полноправные объекты. В языке появились также ссылки на методы, позволяющие использовать существующий метод там, где ожидается лямбда-выражение. Чтобы в полной мере ощутить преимущества лямбда-выражений и ссылок на методы, в язык добавлена также потоковая модель, которая порождает элементы и передает их по конвейеру преобразований и фильтров, не изменяя источник.

В рецептах из этой главы описывается базовый синтаксис лямбда-выражений, ссылок на методы и функциональных интерфейсов, а также поддержка статических методов и методов по умолчанию в интерфейсах. Потоки подробно обсуждаются в главе 3.

1.1. ЛЯМБДА-ВЫРАЖЕНИЯ

Проблема

Вы хотите использовать в своем коде лямбда-выражения.

Решение

Воспользуйтесь одной из синтаксических разновидностей лямбда-выражений и присвойте результат ссылке, имеющей тип функционального интерфейса.

Обсуждение

Функциональный интерфейс – это интерфейс, имеющий единственный абстрактный метод. Класс реализует любой интерфейс, предоставляя реализации всех его методов. Это может быть класс верхнего уровня, внутренний класс и даже анонимный внутренний класс.

Рассмотрим, к примеру, интерфейс `Runnable`, существующий со времен версии Java 1.0. В нем имеется единственный абстрактный метод `run`, который не принимает аргументов и возвращает `void`. Конструктор класса `Thread` принимает экземпляр `Runnable` в качестве аргумента, в примере 1.1 показана реализация анонимного внутреннего класса.

Пример 1.1 ❖ Анонимный внутренний класс, реализующий интерфейс `Runnable`

```
public class RunnableDemo {
    public static void main(String[] args) {
        new Thread(new Runnable() { ❶
            @Override
            public void run() {
                System.out.println(
```

```

        "внутри Runnable в анонимном внутреннем классе");
    }
    }).start();
}
}

```

❶ Анонимный внутренний класс

Синтаксически анонимный внутренний класс начинается словом `new`, за которым следуют имя интерфейса `Runnable` и скобки, означающие, что определяется класс без явно указанного имени, который реализует интерфейс. Код внутри фигурных скобок – это переопределенный метод `run`, который просто выводит строку на консоль.

В примере 1.2 показано, как то же самое реализуется с помощью лямбда-выражения.

Пример 1.2 ❖ Использование лямбда-выражения в конструкторе `Thread`

```

new Thread(() -> System.out.println(
    "внутри конструктора Thread с использованием лямбды")).start();

```

Синтаксически здесь используется стрелка, отделяющая аргументы (в данном случае аргументов нет, так что мы видим только пустую пару скобок) от тела. В этом примере тело содержит всего одну строку, поэтому фигурные скобки не нужны. Такая конструкция называется лямбда-выражением. Вычисленное значение выражения автоматически возвращается. В данном случае `println` возвращает `void`, поэтому и выражение имеет тип `void`, что соответствует сигнатуре метода `run`.

Типы аргументов и возвращаемого значения лямбда-выражения должны соответствовать сигнатуре единственного абстрактного метода интерфейса. Это называется *совместимостью* с сигнатурой метода. Таким образом, лямбда-выражение является реализацией метода интерфейса и может быть при желании присвоено ссылке, имеющей тип интерфейса.

Для демонстрации в примере 1.2 показано присваивание лямбда-выражения переменной.

Пример 1.3 ❖ Присваивание лямбда-выражения переменной

```

Runnable r = () -> System.out.println(
    "лямбда-выражение, реализующее метод run");
new Thread(r).start();

```



В библиотеке Java нет класса с именем `Lambda`. Лямбда-выражения можно присваивать только ссылкам типа функционального интерфейса.

Присвоить лямбда-выражение переменной типа функционального интерфейса – все равно, что сказать, что это лямбда-выражение является реализацией его единственного абстрактного метода. Мы можем рассматривать лямбда-выражение как тело анонимного внутреннего класса, реализующего интерфейс. Именно поэтому лямбда-выражение должно быть совместимо

с абстрактным методом, т. е. типы его аргументов и возвращаемого значения должны соответствовать сигнатуре метода. Отметим, однако, что имя реализуемого метода несущественно. Оно вообще не фигурирует в синтаксисе лямбда-выражения.

Это очень простой пример, поскольку метод `gui` не принимает аргументов и возвращает `void`. Рассмотрим вместо этого функциональный интерфейс `java.io FilenameFilter`, который также является частью стандартной библиотеки Java, начиная с версии 1.0. Аргументом метода `File.list` должен быть экземпляр интерфейса `FilenameFilter`, а сам метод возвращает список имен файлов, удовлетворяющих условию фильтрации.

Согласно документации Java, интерфейс `FilenameFilter` содержит единственный абстрактный метод `accept` с такой сигнатурой:

```
boolean accept(File dir, String name)
```

Аргумент `dir` – каталог, в котором находится файл, а аргумент `name` – имя файла.

В примере 1.4 интерфейс `FilenameFilter` реализован с помощью анонимного внутреннего класса, так что возвращаются только файлы, содержащие исходный код на Java.

Пример 1.4 ❖ Реализация `FilenameFilter` с помощью анонимного внутреннего класса

```
File directory = new File("./src/main/java");

String[] names = directory.list(new FilenameFilter() { ❶
    @Override
    public boolean accept(File dir, String name) {
        return name.endsWith(".java");
    }
});
System.out.println(Arrays.asList(names));
```

❶ Анонимный внутренний класс

Здесь метод `accept` возвращает `true`, если имя файла заканчивается строкой `.java`, и `false` в противном случае.

В примере 1.5 приведена версия с лямбда-выражением.

Пример 1.5 ❖ Лямбда-выражение, реализующее интерфейс `FilenameFilter`

```
File directory = new File("./src/main/java");

String[] names = directory.list((dir, name) -> name.endsWith(".java")); ❶
System.out.println(Arrays.asList(names));
}
```

❶ Лямбда-выражение

Этот код намного проще. На этот раз в скобках указаны аргументы, но без типов. Компилятор знает, что метод `list` принимает аргумент типа `FilenameFilter`, и, следовательно, ему известна сигнатура единственного абстрактного

метода `ассепт`. А раз так, то он знает, что `ассепт` принимает аргументы типа `File` и `String`, так что совместимое лямбда-выражение должно принимать аргументы таких же типов. Метод `ассепт` возвращает значение типа `boolean`, значение такого же типа должно возвращать выражение справа от стрелки.

При желании можно задать типы данных явно, как показано в примере 1.6.

Пример 1.6 ❖ Лямбда-выражение с явно заданными типами данных

```
File directory = new File("./src/main/java");

String[] names = directory.list((File dir, String name) -> { ❶
    name.endsWith(".java");
});
```

❶ Явные типы данных

Наконец, если реализация лямбда-выражения занимает несколько строчек, то необходимо использовать фигурные скобки и включать явное предложение `return`, как показано в примере 1.7.

Пример 1.7 ❖ Блочное лямбда-выражение

```
File directory = new File("./src/main/java");

String[] names = directory.list((File dir, String name) -> { ❶
    return name.endsWith(".java");
});
System.out.println(Arrays.asList(names));
```

❶ Блочный синтаксис

Такое лямбда-выражение называется *блочным*. В данном случае тело содержит всего одну строчку, но благодаря наличию фигурных скобок строчек могло бы быть несколько. Ключевое слово `return` в этом варианте обязательно.

Лямбда-выражение никогда не существует само по себе. Всегда имеется *контекст*, который определяет, объекту какого функционального интерфейса присваивается выражение. Лямбда-выражение может быть аргументом метода, значением, возвращаемым методом, или значением, присваиваемым ссылке. В любом случае, соответствующий объект должен иметь тип функционального интерфейса.

1.2. Ссылки на методы

Проблема

Требуется использовать ссылку на метод, чтобы получить доступ к существующему методу и рассматривать его как лямбда-выражение.

Решение

Воспользуйтесь нотацией с двойным двоеточием, чтобы отделить имя ссылки или класса от имени метода.

Обсуждение

Если лямбда-выражение – это, по существу, способ обращаться с методом, как с объектом, то ссылка на метод – способ обращаться с существующим методом, как с лямбда-выражением.

Например, метод `forEach` интерфейса `Iterable` принимает в качестве аргумента объект типа `Consumer`. В примере 1.8 показано, что `Consumer` можно реализовать как с помощью лямбда-выражения, так и с помощью ссылки на метод.

Пример 1.8 ❖ Использование ссылки на метод для доступа к `println`

```
Stream.of(3, 1, 4, 1, 5, 9)
    .forEach(x -> System.out.println(x));    ❶
Stream.of(3, 1, 4, 1, 5, 9)
    .forEach(System.out::println);          ❷
Consumer<Integer> printer = System.out::println; ❸
Stream.of(3, 1, 4, 1, 5, 9)
    .forEach(printer);
```

- ❶ С помощью лямбда-выражения
- ❷ С помощью ссылки на метод
- ❸ Присваивание ссылки на метод переменной типа функционального интерфейса

Нотация с двойным двоеточием дает ссылку на метод `println` объекта `System.out`, т. е. экземпляра типа `PrintStream`. В конце ссылки на метод скобки не ставятся. В примере выше все элементы потока выводятся в стандартный вывод¹.



Если написанное вами лямбда-выражение состоит из одной строки, в которой вызывается метод, попробуйте заменить его эквивалентной ссылкой на метод.

Ссылка на метод обладает двумя (не очень существенными) преимуществами, по сравнению с лямбда-выражением. Во-первых, она немного короче, а во-вторых, часто включает имя класса, содержащего метод. То и другое упрощает чтение кода.

Ссылки на методы применимы и к статическим методам, как показано в примере 1.9.

Пример 1.9 ❖ Ссылка на статический метод

```
Stream.generate(Math::random)    ❶
    .limit(10)
    .forEach(System.out::println); ❷
```

- ❶ Статический метод
- ❷ Метод экземпляра

¹ Довольно трудно обсуждать лямбда-выражения и ссылки на методы, не затрагивая потоков, которым будет посвящена отдельная глава. Пока скажем лишь, что поток порождает последовательность элементов, но нигде не хранит их и не модифицирует источник.

Метод `generate` интерфейса `Stream` принимает в качестве аргумента экземпляр функционального интерфейса `Supplier`, единственный абстрактный метод которого не имеет аргументов и порождает один результат. Метод `random` класса `Math` совместим с этой сигнатурой, потому что тоже не имеет аргументов и возвращает одно псевдослучайное число типа `double` с равномерным распределением в интервале от 0 до 1. Выражение `Math::random` ссылается на этот метод в качестве реализации интерфейса `Supplier`.

Поскольку метод `Stream.generate` порождает бесконечный поток, мы используем метод `limit` – он оставляет только 10 значений, которые выводятся в стандартный вывод с помощью ссылки на метод `System.out::println`, играющей роль реализации `Consumer`.

Синтаксис

Есть три синтаксических варианта ссылки на метод, один из которых может сбить с толку:

`object::instanceMethod`

Ссылка на метод с помощью имеющейся ссылки на объект, например `System.out::println`.

`Class::staticMethod`

Ссылка на статический метод, например `Math::max`.

`Class::instanceMethod`

Вызов метода экземпляра от имени ссылки на объект, предоставляемой контекстом, например `String::length`.

Именно последний пример приводит в замешательство, поскольку Java-разработчики привыкли, что от имени класса вызываются только статические методы. Напомним, что лямбда-выражения и ссылки на методы никогда не обитают в вакууме – всегда существует контекст. В случае ссылки на объект контекст определяет аргументы метода. Если говорить о печати, то эквивалентным лямбда-выражением (в контексте оно показано в примере 1.8) будет:

```
// эквивалентно System.out::println
x -> System.out.println(x)
```

Контекст предоставляет значение `x`, которое используется в качестве аргумента метода. Для статического метода `max` ситуация аналогична:

```
// эквивалентно Math::max
(x,y) -> Math.max(x,y)
```

Теперь контекст предоставляет два аргумента, и лямбда-выражение возвращает больший из них.

Синтаксическая конструкция «метод экземпляра через имя класса» интерпретируется иначе. Эквивалентное лямбда-выражение выглядит так:

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru