



Содержание

От издательства	15
Предисловие	16
Вступление	18
Об авторе	20
Глава 1. Основы	21
1.1. Интерпретатор Scala	21
1.2. Объявление значений и переменных	23
1.3. Часто используемые типы	25
1.4. Арифметика и перегрузка операторов	26
1.5. Вызов функций и методов	28
1.6. Метод apply	29
1.7. Scaladoc	30
Упражнения	33
Глава 2. Управляющие структуры и функции	34
2.1. Условные выражения	35
2.2. Завершение инструкций	37
2.3. Блочные выражения и присвоение	38
2.4. Ввод и вывод	39
2.5. Циклы	40
2.6. Расширенные циклы for и for-генераторы	41
2.7. Функции	43
2.8. Аргументы по умолчанию и именованные аргументы L1	44
2.9. Переменное количество аргументов L1	45
2.10. Процедуры	46

2.11. Ленивые значения L1	47
2.12. Исключения	48
Упражнения	50
Глава 3. Работа с массивами	52
3.1. Массивы фиксированной длины	53
3.2. Массивы переменной длины: буферы	53
3.3. Обход массивов и буферов	54
3.4. Преобразование массивов	56
3.5. Типичные алгоритмы	57
3.6. Расшифровываем Scaladoc	59
3.7. Многомерные массивы	60
3.8. Взаимодействие с Java	61
Упражнения	62
Глава 4. Ассоциативные массивы и кортежи	64
4.1. Конструирование отображений	65
4.2. Доступ к значениям в ассоциативных массивах	66
4.3. Изменение значений в ассоциативных массивах	66
4.4. Обход элементов ассоциативных массивов	67
4.5. Сортированные ассоциативные массивы	68
4.6. Взаимодействие с Java	69
4.7. Кортежи	69
4.8. Функция zip	71
Упражнения	71
Глава 5. Классы	73
5.1. Простые классы и методы без параметров	74
5.2. Свойства с методами доступа	75
5.3. Свойства только с методами чтения	77
5.4. Приватные поля объектов	79
5.5. Свойства компонентов L1	80
5.6. Дополнительные конструкторы	81
5.7. Главный конструктор	82
5.8. Вложенные классы L1	85
Упражнения	88
Глава 6. Объекты	90
6.1. Объекты-одиночки	90

6.2. Объекты-компаньоны	91
6.3. Объекты, расширяющие классы или трейты	92
6.4. Метод apply	93
6.5. Объект, представляющий приложение	94
6.6. Перечисления	95
Упражнения	97

Глава 7. Пакеты и импортирование

7.1. Пакеты	99
7.2. Правила видимости	100
7.3. Объявления цепочек пакетов	102
7.4. Объявления в начале файла	102
7.5. Объекты пакетов	103
7.6. Видимость внутри пакетов	104
7.7. Импортирование	104
7.8. Импортирование возможно в любом месте	105
7.9. Переименование и сокрытие членов	106
7.10. Неявный импорт	107
Упражнения	107

Глава 8. Наследование

8.1. Наследование классов	109
8.2. Переопределение методов	110
8.3. Проверка и приведение типов	111
8.4. Защищенные поля и методы	112
8.5. Создание суперклассов	112
8.6. Переопределение полей	114
8.7. Анонимные подклассы	115
8.8. Абстрактные классы	116
8.9. Абстрактные поля	116
8.10. Порядок создания и опережающие определения L3	117
8.11. Иерархия наследования в Scala	119
8.12. Равенство объектов L1	121
Упражнения	122

Глава 9. Файлы и регулярные выражения

9.1. Чтение строк	125
9.2. Чтение символов	125
9.3. Чтение лексем и чисел	126

9.4. Чтение из URL и других источников.....	127
9.5. Чтение двоичных файлов	127
9.6. Запись в текстовые файлы.....	127
9.7. Обход каталогов	128
9.8. Сериализация.....	129
9.9. Управление процессами A2	130
9.10. Регулярные выражения	132
9.11. Группы в регулярных выражениях	133
Упражнения.....	134
Глава 10. Трейты	136
10.1. Почему не поддерживается множественное наследование?	137
10.2. Трейты как интерфейсы	139
10.3. Трейты с конкретными реализациями	140
10.4. Объекты с трейтами.....	141
10.5. Многоуровневые трейты	142
10.6. Переопределение абстрактных методов в трейтах.....	143
10.7. Трейты с богатыми интерфейсами	144
10.8. Конкретные поля в трейтах	145
10.9. Абстрактные поля в трейтах	146
10.10. Порядок конструирования трейтов.....	147
10.11. Инициализация полей трейтов.....	149
10.12. Трейты, наследующие классы	151
10.13. Собственные типы L2	152
10.14. За кулисами	153
Упражнения.....	155
Глава 11. Операторы	158
11.1. Идентификаторы	159
11.2. Инфиксные операторы.....	160
11.3. Унарные операторы	161
11.4. Операторы присвоения.....	161
11.5. Приоритет.....	162
11.6. Ассоциативность	163
11.7. Методы apply и update	163
11.8. Экстракторы L2	164
11.9. Экстракторы с одним аргументом или без аргументов L2	166

11.10. Метод <code>unapplySeq</code> L2	167
Упражнения	168
Глава 12. Функции высшего порядка	170
12.1. Функции как значения	171
12.2. Анонимные функции	172
12.3. Функции с функциональными параметрами	173
12.4. Вывод типов	174
12.5. Полезные функции высшего порядка	175
12.6. Замыкания	176
12.7. Преобразование функций в SAM	177
12.8. Карринг	178
12.9. Абстракция управляющих конструкций	180
12.10. Выражение <code>return</code>	181
Упражнения	182
Глава 13. Коллекции	184
13.1. Основные трейты коллекций	185
13.2. Изменяемые и неизменяемые коллекции	186
13.3. Последовательности	188
13.4. Списки	189
13.5. Изменяемые списки	190
13.6. Множества	191
13.7. Операторы добавления и удаления элементов	193
13.8. Общие методы	195
13.9. Функции <code>map</code> и <code>flatMap</code>	198
13.10. Функции <code>reduce</code> , <code>fold</code> и <code>scan</code> A3	199
13.11. Функция <code>zip</code>	203
13.12. Итераторы	204
13.13. Потoki A3	205
13.14. Ленивые представления	207
13.15. Взаимодействие с коллекциями Java	208
13.16. Потокобезопасные коллекции	210
13.17. Параллельные коллекции	211
Упражнения	213
Глава 14. Сопоставление с образцом и case-классы	215
14.1. Лучше, чем <code>switch</code>	216

14.2. Ограничители	217
14.3. Переменные в образцах	218
14.4. Сопоставление с типами	219
14.5. Сопоставление с массивами, списками и кортежами....	219
14.6. Экстракторы	220
14.7. Образцы в объявлениях переменных	221
14.8. Образцы в выражениях for	222
14.9. Case-классы	223
14.10. Метод сору и именованные параметры	224
14.11. Инфиксная нотация в предложениях case	224
14.12. Сопоставление с вложенными структурами	226
14.13. Так ли необходимы case-классы?	227
14.14. Запечатанные классы.....	228
14.15. Имитация перечислений	229
14.16. Тип Option	229
14.17. Частично определенные функции L2	231
Упражнения	231
Глава 15. Аннотации	234
15.1. Что такое аннотации?	235
15.2. Что можно аннотировать?	236
15.3. Аргументы аннотаций	237
15.4. Реализация аннотаций.....	238
15.5. Аннотации для элементов Java	239
15.5.1. Модификаторы Java	239
15.5.2. Интерфейсы-маркеры	240
15.5.3. Контролируемые исключения	241
15.5.4. Списки аргументов переменной длины.....	241
15.5.5. Компоненты JavaBeans	242
15.6. Аннотации для оптимизации	242
15.6.1. Хвостовая рекурсия	243
15.6.2. Создание таблиц переходов и встраивание	245
15.6.3. Игнорируемые методы	245
15.6.4. Специализация простых типов	247
15.7. Аннотации ошибок и предупреждений	248
Упражнения	249
Глава 16. Обработка XML	251
16.1. Литералы XML	252
16.2. Узлы XML	252

16.3. Атрибуты элементов	254
16.4. Встроенные выражения	255
16.5. Выражения в атрибутах.....	257
16.6. Необычные типы узлов.....	258
16.7. XPath-подобные выражения.....	259
16.8. Сопоставление с образцом.....	260
16.9. Модификация элементов и атрибутов.....	262
16.10. Трансформация XML	263
16.11. Загрузка и сохранение	263
16.12. Пространства имен	266
Упражнения.....	267
Глава 17. Параметризованные типы	269
17.1. Обобщенные классы	270
17.2. Обобщенные функции.....	270
17.3. Границы изменения типов.....	271
17.4. Границы представления	273
17.5. Границы контекста	273
17.6. Границы контекста Manifest	274
17.7. Множественные границы	274
17.8. Ограничение типов L3	275
17.9. Вариантность.....	277
17.10. Ко- и контравариантные позиции	279
17.11. Объекты не могут быть обобщенными.....	281
17.12. Подстановочный символ	282
Упражнения.....	283
Глава 18. Дополнительные типы	285
18.1. Типы-одиночки.....	286
18.2. Проекция типов	288
18.3. Цепочки	289
18.4. Псевдонимы типов.....	290
18.5. Структурные типы	291
18.6. Составные типы	291
18.7. Инфиксные типы	293
18.8. Экзистенциальные типы	293
18.9. Система типов языка Scala	295
18.10. Собственные типы	296
18.11. Внедрение зависимостей	297

18.12. Абстрактные типы L3	300
18.13. Родовой полиморфизм L3	302
18.14. Типы высшего порядка L3	305
Упражнения	309

Глава 19. Парсинг

19.1. Грамматики	312
19.2. Комбинирование операций парсера	314
19.3. Преобразование результатов парсинга	316
19.4. Отбрасывание лексем	318
19.5. Создание деревьев синтаксического анализа	318
19.6. Уход от левой рекурсии	319
19.7. Дополнительные комбинаторы	321
19.8. Уход от возвратов	324
19.9. Packrat-парсеры	325
19.10. Что такое парсеры?	326
19.11. Парсеры на основе регулярных выражений	327
19.12. Парсеры на основе лексем	328
19.13. Обработка ошибок	330
Упражнения	331

Глава 20. Акторы

20.1. Создание и запуск акторов	334
20.2. Отправка сообщений	335
20.3. Прием сообщений	336
20.4. Отправка сообщений другим акторам	338
20.5. Каналы	339
20.6. Синхронные сообщения и Futures	340
20.7. Совместное использование потоков выполнения	342
20.8. Жизненный цикл акторов	345
20.9. Связывание акторов	346
20.10. Проектирование приложений с применением акторов	347
Упражнения	349

Глава 21. Неявные параметры и преобразования ...

21.1. Неявные преобразования	352
21.2. Использование неявных преобразований для расширения существующих библиотек	353

21.3. Импорт неявных преобразований	353
21.4. Правила неявных преобразований	355
21.5. Неявные параметры	356
21.6. Неявные преобразования с неявными параметрами	358
21.7. Границы контекста	359
21.8. Неявный параметр подтверждения	360
21.9. Аннотация @implicitNotFound	362
21.10. Тайна CanBuildFrom	363
Упражнения	365
Глава 22. Ограниченные продолжения	367
22.1. Сохранение и вызов продолжений	368
22.2. Вычисления с «дырками»	370
22.3. Управление выполнением в блоках reset и shift	370
22.4. Значение выражения reset	372
22.5. Типы выражений reset и shift	372
22.6. CPS-аннотации	374
22.7. Преобразование рекурсии в итерации	375
22.8. Устранение инверсии управления	378
22.9. CPS-трансформация	382
22.10. Трансформирование вложенных контекстов управления	385
Упражнения	387
Предметный указатель	389



От издательства

При подготовке данной книги к печати, ее рукописи были переданы для обсуждения членам сообщества ru-scala (<http://ru-scala.livejournal.com/>), принявшим самое деятельное участие в ее улучшении.

Особую благодарность издательство выражает Руслану Шевченко, Виталию Морариану, Андрею Легкому и Ивану Федорову. Спасибо вам, друзья! Вашу помощь переоценить невозможно!



Предисловие

Когда я встретил Кея Хорстманна (Cay Horstmann) несколько лет тому назад, он сказал, что необходимо написать хорошую вводную книгу, описывающую язык Scala. Как раз перед этим вышла моя собственная книга, поэтому я, разумеется, спросил его, что в ней не так. Он ответил, что книга замечательная, но слишком большая – его студентам просто не хватает терпения прочитать все восемьсот страниц книги «Programming in Scala». Мне не оставалось ничего иного, как признать его правоту. И он вознамерился исправить ситуацию, написав книгу «Scala для нетерпеливых».

Я очень рад, что его книга наконец вышла, потому что она полностью соответствует своему названию. Она представляет собой весьма практичное введение в язык программирования Scala, описывает, в частности, чем этот язык отличается от Java, как преодолевать некоторые типичные проблемы, возникающие при его изучении, и как писать хороший программный код на языке Scala.

Scala – чрезвычайно выразительный и гибкий язык программирования. Он позволяет разработчикам библиотек использовать весьма сложные, высокоуровневые абстракции, чтобы пользователи этих библиотек, в свою очередь, могли легко и просто выражать свои мысли. В зависимости от того, с каким кодом вы столкнетесь, он может казаться очень простым или очень сложным.

Год назад я попытался дать некоторые разъяснения, определив ряд уровней для языка Scala и его стандартной библиотеки. Всего было выделено по три уровня для прикладных программистов и для создателей библиотек. Начальные уровни были просты в изучении, и их было вполне достаточно, чтобы начать писать программы. Знания, получаемые на средних уровнях, позволяют писать более выразительные и более функциональные программы, а библиотеки более гибкие в использовании. Освоив высшие уровни, программисты становятся экспертами, способными решать специализированные задачи. В то время я писал:

Я надеюсь, что это поможет начинающим решить, в каком порядке изучать темы, а учителям и авторам книг подскажет, в каком порядке представлять материал.

Книга Кея стала первой, где эта идея была воплощена в жизнь. Каждая глава помечена значком, обозначающим ее уровень, который сообщает читателю, насколько простой или сложной она является и на кого ориентирована – на разработчиков библиотек или прикладных программистов.

Как можно догадаться, первые главы представляют собой быстрое введение в основные возможности языка Scala. Но книга не останавливается на этом. Она также охватывает множество концепций «среднего» уровня и, наконец, доходит до описания весьма сложных тем, которые обычно не рассматриваются во вводных книгах, таких как создание парсер-комбинаторов или использование ограниченных продолжений. Метки, обозначающие уровень, могут служить руководством при выборе глав для чтения. И Кею с успехом удалось просто и доходчиво рассказать даже о самых сложных понятиях.

Мне настолько понравилась идея книги «Scala для нетерпеливых», что я предложил Кею и его редактору Грегу Доенчу (Greg Doench) выложить первую часть книги в свободный доступ на веб-сайте Typesafe¹. Они любезно согласились с моим предложением, за что я очень благодарен им. Теперь любой желающий может быстро обратиться к самому лучшему, на мой взгляд, компактному введению в язык Scala.

Мартин Одерски (Martin Odersky)
Январь 2012

¹ <http://typesafe.com/resources/free-books>. – Прим. перев.



Вступление

Развитие языков Java и C++ существенно замедлилось, и программисты, стремящиеся использовать самые современные технологии, обратили свои взоры на другие языки. Scala – весьма привлекательный выбор. Я считаю, что это самый интересный вариант для программистов, стремящихся вырваться за рамки Java или C++. Scala имеет выразительный синтаксис, который выглядит весьма свежо после приевшихся шаблонов Java. Программы на этом языке выполняются под управлением виртуальной машины Java, что открывает доступ к огромному количеству библиотек и инструментов. Он поддерживает функциональный стиль программирования, не отказываясь при этом от объектно-ориентированного стиля, давая возможность осваивать новые парадигмы постепенно. Интерпретатор дает возможность быстро опробовать свои идеи, что превращает изучение Scala в весьма увлекательное занятие. Наконец, язык Scala является статически типизированным языком, что дает компилятору возможность находить ошибки, а вам не тратить время на их поиск в работающей программе.

Я написал эту книгу для *нетерпеливых* читателей, желающих приступить к программированию на языке Scala немедленно. Я полагаю, что вы знакомы с Java, C# или C++, и потому не буду утруждать себя объяснением, что такое переменные, циклы или классы. Я не буду терпеливо перечислять все особенности языка, я не буду читать лекции о превосходстве одной парадигмы над другой, и я не заставлю вас продирааться сквозь длинные искусственные примеры. Вместо этого вы будете получать необходимую информацию небольшими порциями, чтобы ее можно было быстро прочитать и вернуться к ней при необходимости.

Scala – сложный язык, но вы сможете эффективно использовать его, даже не зная всех его тонкостей. Мартин Одерски (Martin Odersky), создатель языка Scala, определил уровни владения языком для прикладных программистов и разработчиков библиотек, как показано в табл. П.1.

Таблица П. 1. Уровни владения языком Scala

Прикладные программисты	Разработчики библиотек	Общий уровень владения языком
Начальный A1		Начальный
Переходный A2	Простой L1	Переходный
Эксперт A3	Сложный L2	Сложный
	Эксперт L3	Эксперт

Каждая глава (а иногда и отдельные разделы) помечены специальным значком, обозначающим уровень владения языком, необходимым для ее чтения. Главы следуют по возрастанию уровня сложности **A1**, **L1**, **A2**, **L2**, **A3**, **L3**. Даже если вы не планируете создавать собственные библиотеки, знание инструментов Scala, которыми пользуются разработчики библиотек, поможет вам эффективнее использовать чужие библиотеки.

Надеюсь, вам понравится изучать язык Scala с помощью этой книги. Если вы обнаружите ошибки или у вас появятся предложения по улучшению, посетите сайт <http://horstmann.com/scala> и оставьте свой комментарий. На этом сайте вы также найдете ссылку на файл архива, содержащего все примеры программного кода из этой книги.

Я очень благодарен Дмитрию Кирсанову (Dmitry Kirsanov) и Алине Кирсановой (Alina Kirsanova), превратившим мою рукопись в формате XHTML в замечательную книгу, позволив мне сконцентрироваться на содержимом и не отвлекаться на оформление. Любой автор скажет, насколько это здорово!

Книгу рецензировали: Адриан Кумиски (Adrian Cumiskey), Майк Дэвис (Mike Davis), Роб Диккенс (Rob Dickens), Даниэль Собрал (Daniel Sobral), Крейг Татарин (Craig Tataryn), Дэвид Уоленд (David Walend) и Уильям Уилер (William Wheeler). Спасибо вам за ваши комментарии и предложения!

Наконец, как всегда, хочу выразить признательность моему редактору Грегу Доенчу (Greg Doench) за то, что подал идею написать эту книгу, и за его поддержку в процессе работы.

*Кей Хорстманн (Cay Horstmann)
Сан-Франциско, 2012*



Об авторе

Кей Хорстманн (Cay S. Horstmann) – основной автор книги «Core Java™, Volumes I & II, Eighth Edition» (Sun Microsystems Press, 2008)¹, а также десятков других книг для профессиональных программистов и студентов факультетов информатики. Он является профессором информатики университета в Сан-Хосе и обладателем звания Java Champion.

¹ Кей С. Хорстманн, Г. Корнелл. Java 2. Библиотека профессионала. Основы. – Т. 1. – Вильямс, 2008. – ISBN: 978-5-8459-1378-4; Кей С. Хорстманн, Г. Корнелл. Java 2. Библиотека профессионала. Тонкости программирования. – Т. 2. – Вильямс, 2008. – ISBN: 978-5-8459-1482-8. – *Прим. перев.*



Глава 1. Основы

Темы, рассматриваемые в этой главе **A1**

- 1.1. Интерпретатор Scala.
- 1.2. Объявление значений и переменных.
- 1.3. Часто используемые типы.
- 1.4. Арифметика и перегрузка операторов.
- 1.5. Вызов функций и методов.
- 1.6. Метод `apply`.
- 1.7. Scaladoc.
- Упражнения.

В этой главе вы узнаете, как использовать язык Scala в качестве мощного карманного калькулятора, выполняя арифметические операции над числами в интерактивном режиме. Попутно здесь будет представлено множество важных понятий и идиом языка Scala. Вы также узнаете, как просматривать документацию Scaladoc.

В этом введении рассматриваются следующие основные темы:

- использование интерпретатора Scala;
- определение переменных с помощью объявлений `var` и `val`;
- числовые типы;
- использование операторов и функций;
- навигация по документации Scaladoc.

1.1. Интерпретатор Scala

Чтобы приступить к работе с интерпретатором Scala, необходимо:

- установить язык Scala;
- добавить путь к каталогу `scala/bin` в переменную окружения `PATH`;
- открыть окно терминала в своей операционной системе;
- ввести команду `scala` и нажать клавишу **Enter**.

Совет. Не любите пользоваться командной оболочкой? Другие способы запуска интерпретатора описываются на странице <http://horstmann.com/scala/install>.

Вводите следующие команды, завершая ввод нажатием клавиши **Enter**. Каждый раз интерпретатор будет выводить ответ. Например, если ввести $8 * 5 + 2$ (как показано ниже), интерпретатор выведет ответ 42.

```
scala> 8 * 5 + 2
res0: Int = 42
```

Результату назначается имя `res0`. Его можно использовать в последующих вычислениях:

```
scala> 0.5 * res0
res1: Double = 21.0
scala> "Hello, " + res0
res2: java.lang.String = Hello, 42
```

Как видите, интерпретатор также отображает тип результата – `Int`, `Double` и `java.lang.String` в примерах выше.

Вы можете вызывать методы. При некоторых способах запуска есть возможность использовать клавишу **Tab** для автоматического дополнения имен методов. Попробуйте ввести `res2.to` и нажать клавишу **Tab**. Если интерпретатор предложит варианты выбора, такие как:

```
toCharArray toLowerCase toString toUpperCase
```

это означает, что функция автоматического дополнения работает. Введите `U` и снова нажмите клавишу табуляции. Теперь вы должны получить единственный вариант:

```
res2.toUpperCase
```

Нажмите клавишу **Enter**, и на экране появится ответ. (Если функция автоматического дополнения не работает, придется ввести имя метода вручную.)

Попробуйте также понажимать клавиши со стрелками \uparrow и \downarrow . В большинстве реализаций вы увидите прежде выполнявшиеся команды и сможете редактировать их. С помощью клавиш \leftarrow , \rightarrow и **Del** измените последнюю команду на

```
res2.toLowerCase
```

Как видите, интерпретатор Scala способен прочитать выражение, вычислить его, вывести результат и прочитать следующее выражение. Такой порядок действий называется *цикл чтения–вычисления–вывода* (read-eval-print loop, REPL).

Строго говоря, программа `scala` *не является* интерпретатором. За кулисами она быстро компилирует введенные вами команды в байт-код и выполняет его в виртуальной машине Java. По этой причине большинство программистов на Scala предпочитают называть этот цикл «the REPL».

Совет. Цикл REPL – ваш друг и помощник. Немедленная обратная связь поощряет эксперименты, и вы будете чувствовать себя увереннее, имея возможность немедленно получать результаты.

При этом очень хорошо иметь постоянно открытое окно редактора, чтобы можно было копировать в него удачные фрагменты кода для использования в будущем. Кроме того, экспериментируя с более сложными примерами, их можно сначала компоновать в редакторе, а затем копировать в окно REPL.

1.2. Объявление значений и переменных

Вместо имен `res0`, `res1` и т. д. можно определить собственные имена:

```
scala> val answer = 8 * 5 + 2
answer: Int = 42
```

Их можно использовать в последующих выражениях:

```
scala> 0.5 * answer
res3: Double = 21.0
```

Значение, объявленное с помощью `val`, в действительности является константой – ее значение нельзя изменить:

```
scala> answer = 0
<console>:6: error: reassignment to val
```

Чтобы объявить переменную, значение которой может изменяться, следует использовать ключевое слово `var`:

```
var counter = 0
counter = 1 // ОК, переменные могут изменяться
```

В языке `Scala` предпочтительнее использовать `val`, если в дальнейшем не предполагается изменять значение. Самое удивительное для программистов на `Java` или `C++`, что в большинстве программ не требуется много переменных `var`.

Обратите внимание на отсутствие необходимости явно объявлять тип значения или переменной. Тип автоматически определяется из типа инициализирующего выражения. (Объявление значения или переменной без инициализации является ошибкой.)

Однако при необходимости тип можно объявить явно. Например:

```
val greeting: String = null
val greeting: Any = "Hello"
```

Примечание. В языке `Scala` тип переменной или функции всегда указывается после имени этой переменной или функции. Это упрощает чтение объявлений сложных типов.

Так как мне часто приходится переключаться между языками `Scala` и `Java`, я замечаю, что мои пальцы автоматически набирают такие `Java`-объявления, как `String greeting`, поэтому мне приходится исправлять их на `greeting: String`. Это немного раздражает, но когда я работаю со сложными программами на языке `Scala`, мне нравится, что не нужно расшифровывать объявления в стиле языка `C`.

Примечание. Возможно, кто-то уже заметил отсутствие точек с запятой после объявлений переменных и инструкций присваивания. В языке `Scala` точки с запятой необходимы, только если в одной строке присутствует несколько инструкций.

В одном объявлении можно объявить сразу несколько значений или переменных:

```
val xmax, ymax = 100 // xmax и ymax получат значение 100
var greeting, message: String = null
// обе переменные, greeting и message, - строки со значением null
```

1.3. Часто используемые типы

Вы уже видели некоторые типы данных из языка Scala, такие как `Int` и `Double`. Как и в языке Java, в Scala имеются семь числовых типов: `Byte`, `Char`, `Short`, `Int`, `Long`, `Float` и `Double`, — и один логический тип `Boolean`. Однако, в отличие от Java, все эти типы являются *классами*. В Scala нет никакой разницы между простыми типами и классами. Вы можете вызывать методы чисел, например:

```
1.toString() // Вернет строку "1"
```

или еще интереснее:

```
1.to(10) // Вернет Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

(Класс `Range` будет рассматриваться в главе 13, а пока просто считайте его коллекцией чисел.)

В языке Scala нет необходимости использовать типы-обертки (*wrapper types*). Эту работу берет на себя компилятор, преобразуя простые типы в обертки и обратно. Например, создав массив чисел типа `Int`, в виртуальной машине вы получите массив `int[]`.

Как было показано в разделе 1.1 «Интерпретатор Scala», в случае со строками Scala опирается на класс `java.lang.String`. Однако расширяет этот класс более чем сотней дополнительных операций в классе `StringOps`. Например, метод `intersect` возвращает символы, общие для двух строк:

```
"Hello".intersect("World") // Вернет "lo"
```

В этом примере объект `"Hello"` типа `java.lang.String` неявно преобразуется в объект типа `StringOps`, и затем вызывается метод `intersect` класса `StringOps`.

Поэтому не забудьте заглянуть в описание класса `StringOps`, когда будете пользоваться документацией для Scala (см. раздел 1.7 «Scaladoc»).

Существуют также аналогичные классы `RichInt`, `RichDouble`, `RichChar` и т. д. Каждый из них имеет небольшой набор удобных методов для расширения на своих бедных родственников — `Int`, `Double` или `Char`. Метод `to`, представленный выше, в действительности является методом класса `RichInt`. В выражении

```
1.to(10)
```

значение типа `Int` сначала преобразуется в объект типа `RichInt`, а затем вызывается метод `to` этого объекта.

Наконец, существуют классы `BigInt` и `BigDecimal` для вычислений с произвольной (но конечной) точностью. Они опираются на классы `java.math.BigInteger` и `java.math.BigDecimal`, но, как будет показано в следующем разделе, они намного удобнее, потому что допускают использование с обычными математическими операторами.

Примечание. Для преобразований между числовыми типами в Scala используются методы, а не операция приведения типа. Например, `99.44.toInt` вернет `99`, а `99.toChar` – `'c'`. Разумеется, как и в языке Java, метод `toString` преобразует любой объект в строку.

Преобразовать строку цифр в число можно с помощью методов `toInt` и `toDouble`. Например, `"99.44".toDouble` вернет `99.44`.

1.4. Арифметика и перегрузка операторов

Арифметические операторы в языке Scala действуют так же, как в Java или в C++:

```
val answer = 8 * 5 + 2
```

Операторы `+` `-` `*` `/` `%` выполняют свою обычную работу, как и поразрядные операторы `&` `|` `^` `>>` `<<`. Есть лишь один необычный аспект: эти операторы в действительности являются методами. Например,

```
a + b
```

это сокращенная форма записи

```
a.+(b)
```

Здесь `+` является именем метода. В языке Scala отсутствует глупое предубеждение против неалфавитно-цифровых символов в именах методов. Вы можете объявлять методы, содержащие почти любые символы в именах. Например, класс `BigInt` определяет метод с именем `/%`, который возвращает частное и остаток от деления.

В общем случае следующая форма записи

```
a method b
```

является сокращением от

```
a.method(b)
```

где `method` – это метод с двумя параметрами (один неявный и один – явный). Например, вместо

```
1.to(10)
```

можно записать

```
1 to 10
```

Используйте любую форму, которая будет проще для восприятия. Начинающие программисты на Scala по привычке придерживаются синтаксиса языка Java, и это хорошо. Разумеется, даже самые прожженные Java-программисты предпочтут использовать `a + b` вместо `a.+(b)`.

Между языком Scala, с одной стороны, и Java или C++ – с другой, существует одна весьма заметная разница. В языке Scala отсутствуют операторы `++` и `--`. Вместо них используются выражения `+=1` и `--=1`:

```
counter+=1 // Увеличит значение переменной counter, так как в Scala нет ++
```

Некоторые спрашивают, существуют ли какие-либо глубинные причины, объясняющие отсутствие оператора `++` в языке Scala. (Обратите внимание, что нельзя просто реализовать метод с именем `++`. Поскольку класс `Int` является неизменяемым, такой метод не сможет изменить целочисленное значение.) Разработчики языка Scala решили не вводить еще одно специальное правило, только чтобы сэкономить на нажатиях клавиш.

При работе с объектами типа `BigInt` и `BigDecimal` можно использовать обычные математические операторы:

```
val x: BigInt = 1234567890
x * x * x // Вернет 1881676371789154860897069000
```

Так намного лучше, чем в Java, где вы были бы вынуждены вызывать `x.multiply(x).multiply(x)`.

Примечание. В Java не поддерживается возможность перегрузки операторов, и разработчики Java утверждают, что это – благо, потому что препятствует появлению совершенно сумасшедших операторов, таких как `!@$&*`, которые могут сделать программу совершенно нечитаемой. Конечно, это глупо – программу точно так же можно сделать нечитаемой, выбирая сумасшедшие имена методов, такие как `qxuwz`. Язык Scala позволяет определять операторы, полагаясь на ваше благоразумие при использовании этой возможности.

1.5. Вызов функций и методов

В дополнение к методам в языке Scala поддерживаются функции. В Scala имеются такие математические функции, как `min` или `pow`, пользоваться которыми намного проще, чем в Java, – для этого не требуется вызывать *статические методы* класса.

```
sqrt(2)    // Вернет 1.4142135623730951
pow(2, 4)  // Вернет 16.0
min(3, Pi) // Вернет 3.0
```

Математические функции определены в пакете `scala.math`. Их можно импортировать инструкцией

```
import scala.math._ // Символ _ в Scala – "групповой" символ, аналог * в Java
```

Примечание. При использовании пакета, имя которого начинается с префикса `scala.`, этот префикс можно опустить. Например, инструкция `import math._` эквивалентна инструкции `import scala.math._`, а вызов `math.sqrt(2)` эквивалентен вызову `scala.math.sqrt(2)`.

Инструкция `import` подробнее будет обсуждаться в главе 7. А пока просто используйте инструкцию `import packageName._`, когда вам потребуется импортировать какой-либо пакет.

В языке Scala отсутствуют статические методы, но в нем есть похожая особенность – *объекты-одиночки* (singleton objects), которые подробно будут рассматриваться в главе 6. Часто классы имеют *объекты-компаньоны* (companion object), чьи методы играют роль статических методов в Java. Например, объект-компаньон `BigInt` для класса `BigInt` имеет метод `probablePrime`, который генерирует случайное простое число с заданным количеством битов:

```
BigInt.probablePrime(100, scala.util.Random)
```

Попробуйте выполнить эту команду в REPL – вы получите число вида 1039447980491200275486540240713. Обратите внимание, что вызов `BigInt.probablePrime` напоминает вызов статического метода в Java.

Примечание. Здесь `Random` – это объект-одиночка, генератор случайных чисел, определенный в пакете `scala.util`. Это одна из ситуаций, когда объекты-одиночки оказываются предпочтительнее классов. В Java часто встречается ошибка, когда для каждого случайного числа создается новый объект `java.util.Random`.

Вызовы методов без аргументов в языке Scala часто записываются без использования круглых скобок. Например, в API класса `StringOps` имеется метод `distinct` без `()`, возвращающий уникальные символы в строке. Его можно вызвать как

```
"Hello".distinct
```

Скобки обычно отсутствуют у методов без параметров, которые не изменяют объект. Подробнее об этом будет рассказываться в главе 5.

1.6. Метод apply

В языке принято использовать синтаксис, напоминающий вызовы функций. Например, если `s` – это строка, тогда выражение `s(i)` вернет i -й символ строки. (В C++ та же самая операция выполняется как `s[i]`; в Java – как `s.charAt(i)`.) Попробуйте выполнить в REPL следующую команду:

```
"Hello"(4) // Вернет 'o'
```

Ее можно считать перегруженной формой оператора `()`. Однако в действительности она реализована как метод с именем `apply`. Например, в описании класса `StringOps` можно найти метод

```
def apply(n: Int): Char
```

То есть выражение `"Hello"(4)` фактически является краткой формой записи

```
"Hello".apply(4)
```

Заглянув в описание объекта-компаньона `BigInt`, можно увидеть методы `apply`, позволяющие преобразовывать строки или числа в объекты `BigInt`. Например, вызов

```
BigInt("1234567890")
```

является краткой формой записи

```
BigInt.apply("1234567890")
```

и возвращает новый объект `BigInt`, при этом *нет необходимости использовать ключевое слово `new`*. Например:

```
BigInt("1234567890") * BigInt("112358111321")
```

Использование метода `apply` объекта-компаньона является в языке Scala типичной идиомой конструирования объектов. Например, вызов `Array(1, 4, 9, 16)` вернет массив, созданный методом `apply` объекта-компаньона `Array`.

1.7. Scaladoc

Для исследования Java API программисты на Java пользуются системой генерации документации Javadoc. В Scala есть аналогичный инструмент – Scaladoc (рис. 1.1).

Навигация по документации в Scaladoc немного сложнее, чем в Javadoc. Классы в языке Scala обычно имеют намного больше вспомогательных методов, чем Java-классы. Некоторые методы используют пока неизвестные вам возможности. Наконец, описания некоторых особенностей рассказывают о том, как они реализованы, а не как ими пользоваться. (Разработчики языка Scala продолжают работать над улучшением внешнего вида Scaladoc, поэтому в будущем этот инструмент, возможно, станет более доступным для начинающих программистов.)

Ниже приводятся несколько советов, касающихся навигации в Scaladoc, для тех, кто приступает к изучению языка.

Существует возможность работать с электронной документацией на сайте www.scala-lang.org/api, но лучше загрузить копию на странице www.scala-lang.org/downloads#api и установить ее локально.

В отличие от Javadoc, где список классов приводится в алфавитном порядке, классы в Scaladoc отсортированы по именам пакетов.