

Кёрк Скотт

JAVA



ДЛЯ
СТУДЕНТА

bhv®

+ CD



Кёрк Скотт

JAVA

ДЛЯ СТУДЕНТА

Санкт-Петербург

«БХВ-Петербург»

2007

УДК 681.3.068+800.92java
ББК 32.973.26-018.1
С44

Скотт К.

С44 Java для студента. — СПб.: БХВ-Петербург, 2007. — 448 с.: ил. + CD-ROM

ISBN 978-5-94157-968-6

Книга написана на базе курса лекций, читаемых автором на протяжении многих лет в США, России и Казахстане. В краткой и доступной форме описаны основные особенности объектно-ориентированного программирования на языке Java, иллюстрируемые многочисленными примерами. Первая часть книги знакомит читателей с основным синтаксисом языка и используемыми обозначениями. Вторая часть посвящена программированию графического интерфейса пользователя. Изложение материала сопровождается упражнениями и заданиями к каждой главе. Промежуточные задания представляют собой написание программы для древних восточных игр, а итоговое — графической версии программы, моделирующей простой микропроцессор. Исходные тексты программ и упражнения располагаются на компакт-диске и сайте <http://math.uaa.alaska.edu/~afkas/BHV>.

Для начинающих программистов и студентов

УДК 681.3.068+800.92java
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Перевод с английского	<i>Андрея Резникова</i>
Редактор	<i>Анна Кузьмина</i>
Компьютерная верстка	<i>Ольги Сергеенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Игоря Цырульникова</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.02.07.

Формат 60×90^{1/16}. Печать офсетная. Усл. печ. л. 28.

Тираж 3000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-94157-968-6

© Скотт К., 2007

© Оформление, издательство "БХВ-Петербург", 2007

Оглавление

Введение.....	9
Благодарности	10
Глава 1. Основы языка Java и объектно-ориентированного программирования	11
1.1. Java	11
Выводы.....	14
1.2. Объектно-ориентированное программирование (ООП).....	15
Выводы.....	18
1.3. Язык UML.....	18
Выводы.....	21
1.4. Загрузка и установка Java и TextPad	22
Выводы.....	24
1.5. Практические советы.....	24
Выводы.....	27
Глава 2. Начало программирования	28
2.1. Пример первой программы.....	28
2.2. Структура кода и результат вывода программы на экран.....	30
2.3. Документация Java API	33
2.4. Создание и использование объектов	37
2.5. Ошибки	40
Глава 3. Типы данных, операторы и переменные	44
3.1. Типы данных, объявление переменных и инициализация	44
3.2. Присваивание и отношения между числовыми типами	47

3.3. Присваивание и простые арифметические операции	49
3.4. Класс <i>Math</i>	52
3.5. Класс <i>String</i>	55
3.6. Класс <i>MyTerminalIO</i>	60

Глава 4. Как написать и использовать коды для классов62

4.1. Переменные экземпляра, конструкторы и методы	62
4.2. Копирование и использование ссылок на объект	71
4.3. Ссылка на <i>null</i>	75

Глава 5. Апплеты и графика82

5.1. Web-страницы и просмотр апплетов.....	82
5.2. Информация о геометрических и графических классах.....	83
5.3. Пример простого апплета	86
5.4. Черчение и заполнение геометрических форм.....	89
5.5. Цвета	90
5.6. Текст	93
5.7. Размещение апплетов на Web-страницах	95
5.8. Web-документация Java.....	97

Глава 6. Оператор *if* — условное выполнение.....102

6.1. Синтаксис оператора <i>if</i> и условия вычисления	102
6.2. Множественные альтернативы и вложенный оператор <i>if</i>	105
6.3. Логические операторы, сложные выражения и булевы переменные.....	108
6.4. Сравнение объектов.....	115
6.5. Последний пример, символьный тип и некоторые типичные ошибки.....	118

Глава 7. Циклы123

7.1. Циклы <i>while</i> и <i>do</i>	123
7.2. Цикл <i>for</i>	126
7.3. Область действия	128
7.4. Примеры	131
7.5. Вложенные циклы.....	135
7.6. Графические примеры	137

Глава 8. Методы, параметры и переменные147

8.1. Передача параметров простых типов.....	147
8.2. Передача объектных ссылок в качестве параметров	152

8.3. Методы доступа, модифицирующие методы и побочные эффекты	157
8.4. Статические методы	160
8.5. Статические переменные	162
8.6. Полная классификация переменных в Java	166
8.7. Еще о синтаксисе	170

Глава 9. Массивы, класс Vector и класс ArrayLists.....173

9.1. Объявление и использование массивов	173
9.2. Копирование массивов, передача массивов в качестве параметров и возвращение массивов методами.....	178
9.3. Примеры массивов.....	181
9.4. Структура коллекций в языке Java	187
9.5. Синтаксис коллекций	189
9.6. Примеры коллекций	193

Глава 10. Файл ввода/вывода.....199

10.1. Вывод на экран и чтение из командной строки	199
10.2. Символы и Unicode.....	202
10.3. Обработка исключений	207
10.4. Запись символов в файл	210
10.5. Чтение символов из файла	212
10.6. Запись строк в файл.....	217
10.7. Файл ввода/вывода с произвольным доступом	218

Глава 11. Рекурсия222

11.1. Описание рекурсии	222
11.2. Составление и выполнение рекурсивного кода.....	225
11.3. Особенности рекурсии	230
11.4. Рекурсия и циклы, рекурсия с двумя параметрами.....	232

Глава 12. Подклассы и наследование237

12.1. Что такое наследование?	237
12.2. Написание кода для подклассов	241
12.3. Доступ к переменным экземпляра и переопределение методов...245	
12.4. Ключевое слово <i>super</i> , методы и конструкторы	249
12.5. Ссылки на классы и подклассы в программах и методах	252

Глава 13. Абстрактность, полиморфизм и интерфейсы.....258

13.1. Абстрактность в конструировании классов.....	258
--	-----

13.2. Полиморфизм и динамическая диспетчеризация.....	263
13.3. Множественное наследование и интерфейсы	268
Глава 14. Методы <i>toString()</i>, <i>equals()</i> и <i>clone()</i>	273
14.1. Метод <i>toString()</i>	273
14.2. Метод <i>equals()</i>	276
14.3. Клонирование.....	278
14.4. Реализация метода <i>clone()</i>	284
14.5. Важное использование клонирования.....	289
Глава 15. Проекты игровых программ	290
15.1. Исторический обзор	290
15.2. Wari	293
15.3. Togiz Kumalak	295
15.4. Варианты реализации	298
Глава 16. Простые схемы UML с помощью программы Visio.....	300
16.1. Схемы статической структуры для классов и объектов	300
16.2. Ассоциации, агрегации и композиции	304
16.3. Наследование, интерфейсы и внутренние классы.....	307
16.4. Схемы последовательностей.....	311
Глава 17. Контейнеры, рисование и диалоговые окна.....	313
17.1. Контейнеры и компоненты	313
17.2. Рисование	317
17.3. Диалоговые окна и строковый вывод на панель	320
Echo1	320
Echo2	322
Echo3	325
Глава 18. Внутренние классы, блоки прослушивания и классы-адаптеры	329
18.1. Внутренние классы, блоки прослушивания и интегрированные внутренние поля.....	329
Echo4	330
Echo5	332
Echo6	336
18.2. Панели, классы-адаптеры и анонимные внутренние классы	339
Echo7	339

Echo8	345
Echo9	348
Глава 19. Объекты, реагирующие на щелчок мыши, апплеты и jar-файл.....	355
19.1. Объекты, реагирующие на щелчок мыши	355
ClickCup	355
ClickDot.....	360
ClickHand	364
19.2. Преобразование приложения в апплет и размещение его на Web-странице	367
19.3. Преобразование приложения в самовыполняющийся jar-файл....	371
Глава 20. Меню, множественные фреймы, сериализация и текстовые поля.....	373
20.1. Ввод с клавиатуры, меню и множественные фреймы	373
ClickKey	373
ClickMenu.....	375
ClickMany.....	379
20.2. Сериализация, выборщики файлов, текстовые поля и полосы прокрутки.....	381
ClickSave	381
ClickChooser.....	385
ClickTextScroll.....	387
Глава 21. Задание для итогового проекта.....	390
21.1. Регистры и память: общие сведения	390
21.2. Примеры программ.....	393
MiscButton1	393
MiscButton2.....	394
MiscButton3	394
MiscRegister	394
MiscText	396
MiscLabel.....	397
21.3. Фокус	397
MiscFocus	397
21.4. Новые примеры программ	404
MiscAction.....	404
MiscMulti.....	404
MiscMemory	405
MiscRegAndMem	406

21.5. Потоки	407
MiscThread	407
Глава 22. Компьютер с минимальным набором команд	412
22.1. Использование MISC	412
22.2. Структура MISC	416
22.3. Машинный класс	418
22.4. Машинные команды	420
Команда <i>MOVE</i>	421
Команда <i>ADD</i>	422
Команда <i>SUB</i>	422
Команда <i>JUMP</i>	423
22.5. Программы машинного языка	423
22.6. Машинная арифметика	426
22.7. Подробное описание проекта	430
Схема зависимостей	433
Приложение. Описание компакт-диска	435
Предметный указатель	443

Введение

Уважаемые читатели!

Вашему вниманию предлагается вводный курс в язык программирования Java, в котором в краткой и доступной форме (в отличие от иных изданий, ориентированных на профессиональных программистов или требующих от читателей определенного уровня подготовки) описаны основные особенности объектно-ориентированного программирования, иллюстрируемые многочисленными примерами. Изложение материала сопровождается упражнениями к каждой главе, вместе с кодами программ, где это необходимо, а также кодами для примеров; весь этот материал вы сможете найти на прилагаемом компакт-диске, а также в Интернете по адресу: <http://math.uaa.alaska.edu/~afkas/BHV>.

В первых главах, как правило, полные листинги приведены в самом тексте книги, но даже в этих случаях понять разбираемые примеры будет значительно проще, если загрузить, составить и попробовать в работе коды, представленные на компакт-диске или сайте. В последующих главах примеры становятся настолько громоздкими и сложными, что нам представлялось нецелесообразным помещать листинги полностью в тексте книги, поэтому необходимо будет взять их с компакт-диска или загрузить с указанного сайта. На этом же сайте будут выкладываться и соответствующие исправления и дополнения.

Благодарности

Пользуясь случаем, я хотел бы выразить благодарность тем людям, которые помогли осуществить настоящее издание.

Я выражаю благодарность программе "Фулбрайт" за поддержку. Первые 15 глав книги были написаны с использованием грантов этой программы, во время моей исследовательской и преподавательской работы в России и Казахстане.

Я также хочу выразить благодарность переводчику книги, А. Резникову. Без его заинтересованного отношения к работе, мой проект не дошел бы до стадии опубликования.

Я хочу поблагодарить мой университет, Университет штата Аляски, Анкоридж, за финансовую поддержку работы над *главами 16—22* издания.

Я хочу также поблагодарить моих студентов, К. Хаванского и Д. Коробова, за помощь в подготовке материалов заданий для диска и Web-сайта.

Я выражаю благодарность всем сотрудникам редакции издательства "БХВ-Петербург", которые принимали участие в работе над текстом книги.

И последнее: вся ответственность за любые недочеты, которые могут быть в книге, лежит исключительно на мне. При этом я надеюсь, что издание будет полезно всем читателям, желающим научиться программировать на языке Java.

*Керк Скотт
Анкоридж, январь 2007*

ГЛАВА 1



Основы языка Java и объектно-ориентированного программирования

1.1. Java

Каковы характеристики языка Java, отличающие его от других языков программирования? Какими особенностями обладает этот язык? Что позволяет ему быть полезным в тех ситуациях, когда другие языки не подходят?

С технической точки зрения, Java обладает одним главным свойством, которым объясняются все остальные особенности данного языка: Java — это язык, который работает на виртуальной машине. В принципе можно создать архитектуру, которая будет непосредственно выполнять код Java, однако в стандартной среде, такой как Windows или UNIX, установка Java на компьютере означает, что установлена виртуальная машина. Когда программы работают, считается, что они функционируют на виртуальной машине. В других языках, таких как C или C++, в результате компиляции мы получаем выполняемый файл, который содержит машинные команды для той архитектуры, для которой был создан компилятор. Для языка Java этот процесс также называется *компиляцией*, но он не является компиляцией в буквальном смысле слова. Когда компилируется код для Java, то результатом будет байт-код, т. е. код, который работает в виртуальной Java-машине.

Язык программирования такого типа является просто еще одним слоем программного обеспечения и не зависит от основных характеристик архитектуры компьютера. В итоге, по крайней мере в теории, Java является потенциально "универсальным" языком программирования. Версии Java существуют для различных операционных систем, механизм их действия отличается друг от друга. Однако результат работы кода Java, запущенного в разных операционных системах, будет один и тот же. Таким образом, программы Java должны быть свободно переносимы между ними. Если программа написана в одной среде, то нет необходимости производить какие-либо изменения в исходном коде, чтобы скомпилировать или напрямую запустить ее в другой среде. В идеале, это можно выразить следующим образом: "Написал один раз, работает везде". Однако на практике тонкие различия в осуществлении механизма Java приводят к проблемам, которые заключаются во фразе: "Написал один раз, находишь ошибки везде".

Как и во всех системных и программных продуктах, в Java существуют компромиссы. Одной из движущих сил развития языка такого типа является Интернет. Если существует "универсальный" язык программирования, то тогда программы могут быть установлены и будут работать на машинах пользователей независимо от характеристик платформы. С помощью Java создаются *апплеты* — программы, которые могут быть загружены из Интернета. Не имеет значения, из-под какой операционной системы происходит подключение к Интернету: если в ней установлен Java, то она будет работать с апплетами, которые доступны во Всемирной сети. Большим компромиссом в данном случае является то, что программы, написанные на Java, работают значительно медленнее, чем программы, которые скомпилированы для конкретной архитектуры. В среднем код Java будет работать в 10 раз медленнее. Но по мере того как процессоры становятся все быстрее, а Интернет становится средой распространения программ, недостаток в скорости, за счет которого достигается универсальность, выглядит все более приемлемым компромиссом.

Есть еще одна особенность языка Java, обусловленная желанием загружать готовые программы. Некоторые языки, такие как C и C++, дают программисту возможность написать код, который

напрямую обращается к адресам в памяти компьютера. Это сделано путем использования указателей. Этого нет в Java. Может показаться странным, что в современном языке невозможно работать с указателями. Можно написать код для связанных структур на Java, но нельзя написать код с использованием адресов памяти. Программисты знают, что программы, которые напрямую имеют доступ к памяти, могут, преднамеренно или случайно, навредить системе. По этой причине прямой доступ к памяти нежелателен в Java. Когда пользователи запускают программы, они хотят быть уверенными, что эти программы не будут разрушать их системы из-за прямого доступа к памяти. Таким образом, Java поддерживает и универсальность, и безопасность программ.

Цель нашей книги — познакомить читателя с основами программирования на языке Java. Java — это объектно-ориентированный язык программирования. Он смоделирован для того, чтобы поддерживать событийно-действующее программирование. Другими словами, он может быть использован для создания графического пользовательского интерфейса, который поддерживает действия по типу "укажи и щелкни" (point and click). Такие приложения могут быть самостоятельными программами или апплетами, загружаемыми из Интернета. Java обладает основными синтаксическими особенностями структурированных языков программирования, а именно последовательным выполнением блоков кода, условным выполнением (if-операторы) и повторяющимся выполнением (циклы). Эти элементы будут рассмотрены в следующих главах, так же как и описание объектно-ориентированного механизма, который поддерживает более сложные виды программирования.

Эксперты расходятся во мнениях относительно технических достоинств Java, однако необходимо сразу отметить, что, с точки зрения изучающих, язык Java обладает рядом недостатков. Java не рассчитан для обучения программированию. Он предназначен для построения сложных приложений. В дополнение ко многим свойствам в самом языке, он имеет библиотеки готовых компонентов кода, которые могут использовать опытные программисты, вместо самостоятельного написания компонентов с нуля. Ни полностью синтаксис, ни библиотеки невозможно изучить за ко-

роткий срок. Многое из того, что изучается на первом этапе, может показаться не слишком полезным. Необходимо рассмотреть основы объектно-ориентированного программирования прежде, чем станет возможным создание интересных приложений, включающих интерактивные графические свойства.

Как только вы познакомитесь с этими основами, то убедитесь, что Java прекрасно соответствует своим целям. Вы сможете создавать графические приложения, которые просто невозможно написать на таких языках программирования, как C. Кроме того, код, написанный на Java, может быть значительно проще, чем код, который написан на C++, чтобы выполнить то же самое. Java с самого начала был создан как объектно-ориентированный язык. В этом его преимущество перед языком C++, который является объектно-ориентированной эволюцией языка C. В заключение нужно сказать, что язык C# был создан для той же самой цели, что и язык Java, поэтому C# является его прямым конкурентом, который по своим характеристикам может быть не хуже или даже способен превосходить Java в некоторых аспектах.

Выводы

1. Язык Java интерпретируется на виртуальной Java-машине.
2. Исходный код и код заданий Java легко переносятся с компьютера на компьютер.
3. Существует возможность загружать код Java из Интернета, несмотря на то, что программы будут не так быстро работать.
4. Java не использует доступ напрямую к памяти для обеспечения безопасности загруженной программы.
5. Java — это объектно-ориентированный язык, имеющий свойства, необходимые для того, чтобы написать графические приложения с управлением событиями.
6. Java предназначен для программирования, но не обучения языку. Он обладает многими свойствами и содержит много библиотек.
7. В Java можно создавать такие программы, которые невозможно или трудно написать на других языках.

1.2. Объектно-ориентированное программирование (ООП)

Языки программирования и парадигмы развиваются с течением времени. Так, появилось новое аппаратное обеспечение, которое способно поддерживать новые парадигмы программирования; возникла необходимость решать новые проблемы, которые, в свою очередь, требуют новых возможностей программирования; при разработке крупных или сложных проектов выявились недостатки в существующих системах. Объектно-ориентированное программирование — одна из новых парадигм программирования. Его развитие и особенности заключаются в следующих трех свойствах: это принцип программирования, который не смогли бы поддержать примитивные машины; программирование графического пользовательского интерфейса — это одна из областей, где его применение очевидно; и поскольку ООП решает, по крайней мере, некоторые проблемы, связанные со структурным программированием — его непосредственным предшественником, оно в значительной мере вытеснило структурное программирование в развитии сложных систем программного обеспечения.

Java — это объектно-ориентированный язык. Есть много объектно-ориентированных языков, и каждый осуществляет этот принцип программирования по-своему. Полное объяснение ООП не входит в круг задач настоящего пособия. Невозможно даже дать полное объяснение ООП в рамках языка Java. Однако желательно начать с некоторых общих идей, относящихся к ООП, и определения терминов и основных понятий. Более полное понимание возникнет тогда, когда применение этих идей в языке Java будет изучено и когда к основным понятиям будут добавлены более сложные понятия и их применение.

ООП имеет два фундаментальных термина, которые будут повторяться снова и снова. Эти термины "класс" и "объект". *Класс* — это модель для объектов. Можно также сказать, что класс — это шаблон или образец для объекта. Точнее, класс — это часть компьютерного кода, которая дает полную спецификацию для объектов. Определение класса — это часть программного исходного кода. Во время работы программы можно создать объект соглас-

но спецификациям данного класса. *Объект* может быть рассмотрен как экземпляр класса. Программа может использовать более одного класса, и более одного экземпляра классов может быть создано во время работы программы. Работа программы состоит из манипуляций такими объектами.

В ООП включены два дополнительных понятия. Компьютерные программы содержат последовательность инструкций, которые действуют вместе и могут выполнять конкретные задания. В объектно-ориентированном программировании они называются методами. *Методы* — это часть определения класса. Компьютерные программы также включают объявления переменных, таких как числа, последовательность символов и т. д., с которыми производятся манипуляции. *Переменные* — это символьные имена, которые используются для указания на сохраненные количества или значения в программе. Различные виды переменных могут быть использованы в различных частях программы. Переменные, связанные с классами и объектами, являются *переменными экземпляра* (или *экземплярными переменными*), и так же, как и методы, они являются частью определения класса.

Хотя оба этих понятия приводятся в определении класса, методы и переменные экземпляра рассматриваются в ООП по-разному. В программе может быть более одного экземпляра более чем одного класса. Следующее отличие между методами и переменными экземпляра заключается в том, что нет необходимости для каждого экземпляра класса иметь свой код метода. Методы определяются в классе и доступны всем объектам данного класса, существующим в программе в данное время. Таким образом, методы для всех объектов класса являются абсолютно одинаковыми. Однако, когда создаются экземпляры класса, они получают свои уникальные копии переменных экземпляра. Каждый объект класса получает тот же самый набор переменных, но значения переменных для каждого объекта могут быть различны.

Итак, каковы взаимоотношения между методами и переменными экземпляра? Фундаментальным аспектом ООП является инкапсуляция. Каждый объект имеет свои копии экземплярных переменных. Значения этих переменных могут быть доступны или изменены во время работы программы только одним способом: вызо-

вом метода, целью которого является доступ к переменным экземпляра или их изменение. Это обеспечивает строгую структуру программы. Невозможно в каких-либо местах программного кода изменить значение переменных экземпляра так, как это удобно в данный момент. Изменения могут быть произведены только в рамках инкапсуляции. Свободный доступ к значению переменных считался одним из недостатков структурного программирования. Эта проблема решена в объектно-ориентированном программировании.

На рис. 1.1 представлена простая графическая схема объекта, показывающая переменные экземпляра, которые он содержит, и методы, которые он использует.



Рис. 1.1

Простой способ понять значение этого рисунка — это представить объект в виде яичницы-глазуньи. Как только объект создан, программа имеет доступ к желтку только через белок. Важно понять, что программисту, пользующемуся объектами, нужно знать, через какие методы доступны переменные. Однако нет необходимости знать, как этот код работает. Хотя наличие некоторых переменных экземпляра может быть очевидно, программисту не нужно знать, как они применяются, и не нужно знать все переменные, чтобы использовать объект. В этом смысле объект можно представить в виде "черного ящика". Программист может производить с ним определенные операции с конкретным ожидаемым результатом, но каким образом достигается данный результат, мы не видим. Методы предоставляют собой интерфейс для доступа к переменным экземпляра и иным значениям, находящимся внутри объекта.

Выводы

1. ООП полагается на современные возможности аппаратного обеспечения, решает проблемы, которые не могли быть решены предшествующими концепциями программирования, и преодолевает некоторые недостатки предыдущих концепций программирования.
2. Java осуществляет концепцию ООП; основные термины и идеи ООП полезны при объяснении языка Java.
3. Классы и объекты представляют собой два основных понятия ООП.
4. Методы и переменные — это два дополнительных ключевых понятия ООП.
5. Когда создаются объекты класса, они используют его методы совместно, но каждый объект получает свои копии переменных экземпляра.
6. Объект инкапсулирует переменные. Значения переменных могут быть доступны только при условии, если существует метод в классе, который позволяет это сделать.
7. Программист, использующий объект, имеет доступ к нему только через методы и ему не нужно знать, как методы работают и какие переменные экземпляра они используют.

1.3. Язык UML

В разных источниках аббревиатуру UML расшифровывают по-разному: "unified modeling language" (единый язык моделирования) или "universal modeling language" (универсальный язык моделирования). Как бы то ни было, это набор условных обозначений для графического изображения объектно-ориентированных программ. Можно нарисовать такие вещи, как классы и их методы, экземпляры объекта, объекты классов, отношения между классами и т. д. В нашей книге мы не будем давать полное описание UML. Однако при описании Java в некоторых случаях графическое изображение идеи или отношения очень помогает. В таких случаях мы будем использовать соответствующие символы UML.

Кроме собственных графических символов, в UML имеется и свой словарь, который используется при представлении графически-ориентированных компонентов. Ранее в этой главе уже упоминались термины *"класс"* и *"объект"*. Эти термины используются в таких же значениях и в рамках UML. Также ранее были объяснены термины *"переменная экземпляра"* и *"метод"*. В UML соответствующими терминами являются *"атрибут"* и *"операция"*. Существует более чем один объектно-ориентированный язык программирования, однако язык UML является нейтральным. Его терминологию можно использовать для описания любой объектно-ориентированной программы, созданной на любом из таких языков.

Поскольку в данной книге рассматривается только язык программирования Java, то мы будем продолжать использовать термины *"переменная экземпляра"* и *"метод"*. Следует также отметить, что в официальной документации языка Java термин *"поле"* (field) используется в значении *"переменная экземпляра"*. Поскольку данный термин выбран разработчиками языка, то, вероятно, он является более правильным. Однако этот термин может сбивать с толку, т. к. он имеет и иные значения в программировании. Поэтому мы будем использовать более описательный термин *"переменная экземпляра"*.

Для иллюстрации употребления UML необходим пример. Такой пример можно рассмотреть и обсудить с его помощью идеи объектно-ориентированного программирования, даже не касаясь языка Java. Пример, который мы будем использовать на протяжении всей книги, основан на следующем. Предположим, что мы будем хранить семена в чашке. На первый взгляд, это может показаться не слишком практичным. Обычно мы представляем себе чашку как сосуд, содержащий какую-либо жидкость, да и не совсем понятно, как можно написать интересные программы, используя чашки и семена. Но любой вводный пример неизбежно будет не слишком практичным. Пример с чашками и семенами приводится здесь потому, что в дальнейшем он используется в качестве компонента игровой программы.

Классы и объекты изображаются с помощью прямоугольников. Отношения указываются с помощью надписей, линий, стрелок и

иных символов. В своей простейшей форме класс может быть изображен прямоугольником, содержащим имя класса. Имя пишется жирным шрифтом с большой буквы, как на рис. 1.2.

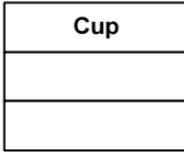


Рис. 1.2

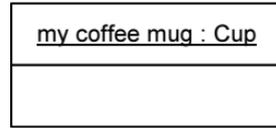


Рис. 1.3

Любой объект данного класса будет иметь собственное имя. Предположим, экземпляр известен как "моя кофейная кружка". Он также представлен прямоугольником. Имя не пишется с большой буквы. За ним ставится двоеточие и название класса, экземпляром которого он является. Все это подчеркивается (рис. 1.3).

Отношения между классами и объектами графически могут быть изображены в виде стрелок. В UML различные виды отношений указываются с помощью различных стрелок, так что форма стрелки имеет значение. Стрелка направлена от объекта к классу (рис. 1.4).

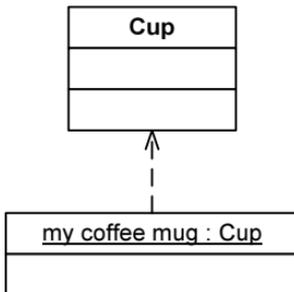


Рис. 1.4

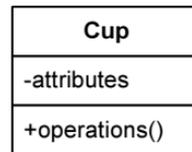


Рис. 1.5

Более подробная информация о классе дается с помощью указания атрибутов и операций на рисунке. В целом, изображение такого класса принимает следующий вид (рис. 1.5).

Пусть наш класс имеет одну переменную экземпляра, `seedCount`, и два метода: `setSeedCount()` и `getSeedCount()`. Пусть переменная будет целым числом, и когда создается новый экземпляр класса, то пусть она инициализируется со значением 0. Метод `setSeed()` присписывает значение экземпляру объекта, а метод `getSeed()` возвращает значение, приданное переменной `seedCount` в данный момент, каким бы оно ни было. Эта информация показана более подробно в следующем изображении класса на рис. 1.6.

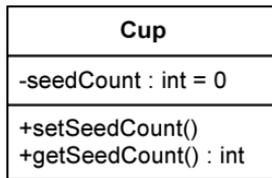


Рис. 1.6

Можно дать еще более подробную информацию, например информацию о параметрах метода, диапазоне значений переменной и иные сведения. Такие схемы используются при документировании систем программного обеспечения, программ или их компонентов. Уровень детализации должен быть рассчитан так, чтобы быть полезным для того, для кого создается графическое изображение, будь то дизайнер, программист или пользователь.

Выводы

1. UML используется для графического изображения объектно-ориентированных программ.
2. UML имеет свой словарь для наименования компонентов классов.
3. Язык Java имеет собственный, параллельный словарь для обозначения компонентов классов.
4. Пример с семенами и чашками используется для иллюстрации.
5. В UML классы и объекты изображаются в виде прямоугольников, а отношения указываются с помощью стрелок и иных символов и надписей.

6. Простейшая схема класса представляет собой прямоугольник, содержащий название объекта, написанное с заглавной буквы жирным шрифтом.
7. Простейшая схема объекта представляет собой прямоугольник, содержащий название объекта, за которым через двоеточие ставится название класса; все это набирается жирным шрифтом.
8. Отношения между объектом и его классом показываются с помощью пунктирной стрелки, направленной от объекта к классу.
9. Более полное изображение класса включает атрибуты и операции класса, а также, возможно, и более подробную информацию.
10. Уровень детализации рисунков может быть разным.

1.4. Загрузка и установка Java и TextPad

Для того чтобы программировать на языке Java, необходимо иметь компилятор. Вы можете бесплатно загрузить компилятор фирмы Sun Microsystems с Web-сайта этой фирмы. Редактор TextPad может быть загружен бесплатно для ознакомления и последующей покупки, при необходимости. Этот редактор рассчитан на работу с Java-программами фирмы Sun. Если сначала вы установили Java, а затем TextPad, следуя инструкции по инсталлированию, то в меню редактора должны появиться активные опции для компиляции и выполнения Java-приложений и апплетов.

Java находится в Интернете по данному адресу:

<http://java.sun.com/javase/downloads/index.jsp>

Часть страницы с желательным компилятором выглядит так, как показано на рис. 1.7.

Когда вы щелкните мышью по ссылке **Download**, то сначала откроется страница с лицензионным соглашением. После этого вам

будет представлен список различных версий программ для разных типов компьютеров. Первой дается стандартная версия для компьютеров с Windows. Вам потребуется установка программы в режиме автономной работы (offline). Это значит, что программа будет скачана на ваш компьютер, а затем вам нужно два раза щелкнуть мышью по ее пиктограмме, чтобы установить программу на вашем компьютере. Если вы работаете с каким-либо иным компьютером, то для вас может подойти одна из других ссылок. Вы можете изучить сайт компании Sun в поисках дополнительной информации, такой как документация к языку или обучающие программы. Здесь мы не можем дать описание всего, что находится на сайте.

JDK 5.0 Update 9

» Download

The Java SE Development Kit (JDK) includes:
the Java Runtime Environment (JRE)
command-line development tools, such as compilers
and debuggers, that are necessary or useful for
developing applets and applications

» [More info about Java SE](#)

[Installation Instructions](#) | [ReadMe](#) | [ReleaseNotes](#) | [Sun License](#) | [Third Party Licenses](#)

Рис. 1.7

Редактор TextPad находится в Интернете по адресу:

<http://www.textpad.com/download/index.html>

Следуйте инструкциям, данным на этой странице. Если возникнут проблемы, можно следовать этим ссылкам: **Support**, **FAQ**, **java**.

Вы найдете ответы на любые вопросы, которые касаются возможных проблем при работе редактора TextPad с Java. Например, если вы загрузили и установили TextPad и Java не в том порядке, здесь вы найдете инструкцию, как сделать так, чтобы они заработали вместе без переустановки. Также имеются ответы и на другие вопросы. Если инструкции вам непонятны или вы столкну-

лись с неразрешимыми проблемами, то, возможно, вам следует использовать интегрированный пакет, который можно скачать из Интернета или купить. Данная книга не содержит полных сведений по установке и работе с редактором TextPad. Если вы используете последнюю версию языка Java, то сможете сделать все, что описано в книге, независимо от типа редактора.

Выводы

1. Как Java, так TextPad и можно скачать из Интернета.
2. Если их установить в указанном порядке, то TextPad будет содержать инструменты для компилирования и выполнения программ на Java.

1.5. Практические советы

Существует несколько деталей, касающихся наименования файлов программ Java, которые следует запомнить. Они сводятся к следующему:

- Файлы программ на Java должны сохраняться с расширением `java`.

Мы еще не ознакомились с кодом для Java-программ. Тем не менее в начале каждой программы будет появляться строка такого вида:

```
public class Name_of_program
```

Имя файла, с которым программа сохранена, должно точно совпадать с названием, которое появляется в первой строке внутри программы. Первая буква названия программы пишется заглавной, и первая буква названия файла также должна быть заглавной. Другими словами, для того чтобы быть работающей, программа должна быть сохранена с таким именем:

```
Name_of_program.java
```

- В языке Java, в отличие от Microsoft Windows, всегда имеет значение, употребляете ли вы заглавную (прописную) или строчную букву. Это может привести к проблемам. Если вы

сохранили файл в Windows с именем, записанным со строчной буквы, а затем попытаетесь изменить название файла, чтобы использовать заглавную букву, то в этом случае Windows может не позволить это сделать, сообщая вам: "Файл с этим именем уже существует". Если это произошло, сохраните файл под временным названием, удалите копию с неправильным именем и переименуйте временный файл, используя заглавную букву.

Если вы используете редактор TextPad, то вам необходимо иметь в виду следующее.

В меню **Tools** редактора TextPad вы найдете 3 опции: **Compile Java**, **Run Java Application** и **Run Java Applet**. Сейчас мы рассмотрим первые две опции: компиляцию и запуск программы. Работу апплетов мы рассмотрим позже.

В меню **Tools** редактора TextPad вы также увидите опцию **Run**. Мы упоминаем о ней только потому, что вы наверняка по ошибке попытаетесь хотя бы раз ее выбрать. Используя эту опцию для того, чтобы запустить программу Java, вы либо откроете диалоговое окно, спрашивающее, какое приложение необходимо запустить, либо получите сообщение об ошибке. Вы должны быть уверены, что запускаете программу с помощью опции **Run Java Application**.

Чтобы проверить компилирование и запуск программ Java в TextPad, необходимо сделать следующее: загружаете программы MyTerminalIO.java и FirstProg.java и сохраняете их в одном месте — на дискете, жестком диске, в какой-либо папке и т. д. на вашем компьютере. Вы можете также ввести имя программы, используя TextPad. Далее приведен код для справки. Сохраните его в файле с именем FirstProg.java.

```
public class FirstProg
{
    public static void main(String[] args)
    {
        MyTerminalIO myterminal = new MyTerminalIO();
```

```
myterminal.println("Hello, World!");  
}  
}
```

Затем выберите в меню опцию **Compile Java**. На это может уйти несколько секунд. Успешная компиляция просто закончится, показывая на экране файл программы. В противном случае на экране появится сообщение об ошибке. Приведенный пример не содержит ошибок, поэтому, если возникла ошибка, проверьте, находится ли копия программы `MyTerminalIO.java` в том же месте, правильно ли вы набрали код, а также сохранили ли вы файл под правильным именем, а затем попытайтесь снова скомпилировать программу.

После успешной компиляции вы можете выбрать в меню **Run Java Application**. Это приведет к открытию окна с результатом работы программы.

Вы можете случайно сделать ошибку при входе в программу или же сделать ошибку намеренно, пытаясь посмотреть, что делает компилятор. В редакторе появится список ошибок с номерами строк, а также достаточно непонятные объяснения. Если установленная программа функционирует нормально, то у вас должна быть возможность перескочить прямо в строку программы, в которой находится ошибка, щелкнув мышью на сообщении об ошибке. Если эта функция программы не работает, то просто запомните номер строки, перейдите в файл кода, выбрав из списка в левом верхнем меню программы `TextPad`, и затем вручную найдите нужную строку. Номер текущей строки показан внизу окна редактора.

Если ваш компьютер соединен с принтером, то вы можете напечатать программу исходных файлов прямо из редактора `TextPad` через выбор пункта **File** в меню.

Для того чтобы распечатать результаты работы приложения, необходимо еще один шаг: дело в том, что окно с выводом программы не содержит меню. Поэтому нужно выделить и скопировать в буфер обмена содержимое окна, затем открыть новый документ в `TextPad` и вставить содержимое буфера. Теперь вы можете распечатать файл, используя опции меню как обычно.

Выводы

1. Имена файлов Java-программ должны совпадать с названиями классов и иметь расширение java. Java различает прописные и строчные буквы.
2. Меню TextPad включает опции для компилирования и запуска программ на Java.
3. Имеется возможность распечатать как код, так и результат программы.

ГЛАВА 2



Начало программирования

2.1. Пример первой программы

Цель данного раздела — дать первоначальные пояснения к первому примеру программы, приведенному в конце предыдущей главы. Напомним эту программу:

```
public class FirstProg
{
    public static void main(String[] args)
    {
        MyTerminalIO myterminal = new MyTerminalIO();
        myterminal.println("Hello, World!");
    }
}
```

Далее приводится пошаговое описание синтаксиса программы вместе с некоторыми пояснениями того, как и почему код написан именно таким образом. На первой стадии самое главное — познакомиться с синтаксисом, чтобы вы могли использовать его при написании собственных программ. Пояснения даются потому, что для большинства людей легче понять что-либо с опорой на пояснения. Однако, если некоторые из комментариев покажутся вам непонятными или недостаточными, не волнуйтесь: в последующих главах вы найдете более подробные пояснения.

1. Первая строка кода показывает синтаксис для объявления класса:

```
public class FirstProg
```

Все программы, написанные пользователями, являются классами. Класс `public` означает, что он доступен для общего пользования. Название класса — `FirstProg`.

2. Вторую строку кода довольно легко объяснить, и это же касается последней строки кода. Все определения класса заключаются в фигурные скобки `{}`.
3. В третьей строке кода указывается метод.

```
public static void main(String[] args)
```

Название этого метода — `main()`. Самостоятельная программа должна иметь этот метод. Когда вы просите систему Java выполнить программу, она смотрит внутрь класса программы и вызывает работу данного метода. Ключевое слово `public` означает, что этот метод свободно вызывается системой. Ключевое слово `static` означает, что метод является общим методом, а не таким, который используется только с данным объектом. Эти понятия будут более подробно объяснены позже. Ключевое слово `void` показывает, что выполнение этого метода не вызывает возвращения каких-либо значений.

Каждое имя метода сопровождается соответствующей парой круглых скобок. Когда в книге будет идти речь о методах, они всегда будут указываться именем со скобками, например `main()`. Подобное написание указывает на то, что имя `main` — название метода, а не чего-то еще. Метод может содержать запись в скобках. Это параметры для аргументов командной строки, в случае если программа использовала их. Взятая для примера программа не использует их, но объявление в скобках все же необходимо. Что обозначает такое объявление, будет объяснено позже.

4. Четвертую и седьмую строки кода довольно легко объяснить. Все определения метода должны быть заключены в соответствующий набор фигурных скобок `{}`.