Кёрк Скотт

JAVA для студента

+ CD

bhv®

Кёрк Скотт

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

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

Оглавление

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.

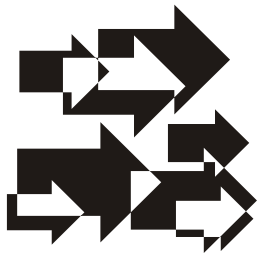
Я хочу также поблагодарить моих студентов, К. Хаванского и Д. Коробова, за помощь в подготовке материалов заданий для диска и Web-сайта.

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 согласс-

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.

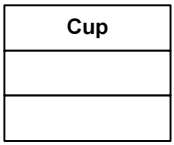


Рис. 1.2

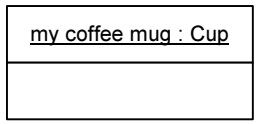


Рис. 1.3

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).

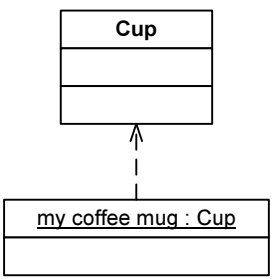


Рис. 1.4

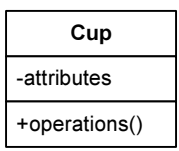


Рис. 1.5

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()` приписывает 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.

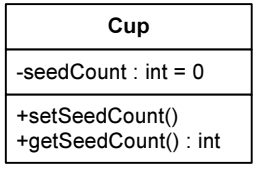


Рис. 1.6

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.

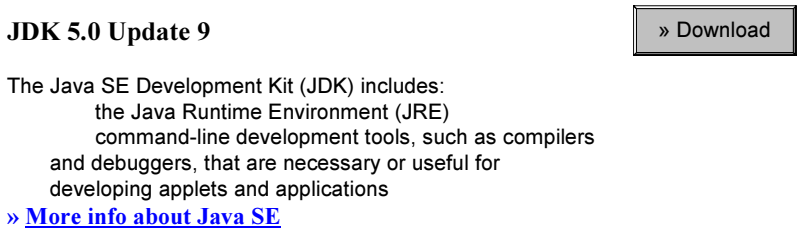


Рис. 1.7

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.

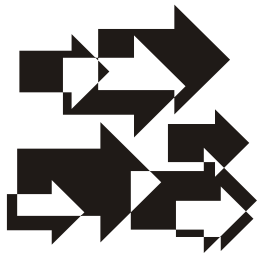
Если ваш компьютер соединен с принтером, то вы можете напечатать программу исходных файлов прямо из редактора TextPad через выбор пункта **File** в меню.

Для того чтобы распечатать результаты работы приложения, необходим 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 `{}`.

5. В пятой строке происходит создание объекта. Классом объекта является `MyTerminalIO`. Название объекта — `myterminal`. Ключевое слово `new` находится непосредственно перед вызовом конструктора, представляющего собой имя класса, за которым следуют круглые скобки.

   ```
   MyTerminalIO myterminal = new MyTerminalIO();
   ```

   Этот объект, `myterminal`, является экземпляром класса `MyTerminalIO` и может использовать метод, который осуществляет вывод на консоль. Вы увидите позже, что этот класс также поддерживает ввод с клавиатуры.

6. Шестая строка кода выполняет реальную работу программы. Этот пример показывает, как можно выполнить полезную работу, используя объекты. Имя объекта — `myterminal`. Затем следует точка, после которой — имя метода. Метод `println()` вызывается для объекта с параметром "Hello, World!".

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

   Если вам удалось скомпилировать и запустить программу, рассмотренную в *главе 1*, вы знаете, что она делает. Она выводит сообщение "Hello, World!" на экран. Это результат вызова метода `println()`. Последовательность символов в двойных кавычках называется *строкой*, и эта последовательность может быть выведена на экран. Возможность сделать это является отправной точкой для изучения того, что можно сделать с помощью языка Java.

2.2. Структура кода и результат вывода программы на экран

Комментарии представляют собой пояснения, которые программист может включить в программу. Они не являются частью кода и не влияют на работу программы. В любой строке, где появляются символы `//`, чтобы ни следовало за этой парой символов, рассматривается как комментарий. Другими словами, компилятор не будет пытаться компилировать этот фрагмент кода, и он не будет выполняться, когда программа заработает. Аналогичным

образом, если что-либо появляется между символами `/*` и `*/`, то это будет рассматриваться как комментарий. Вот версия первой программы с комментариями:

```
/*  Это моя первая программа.  */
public class FirstProg
{
  public static void main(String[] args)
  {
    // Здесь создается объект.
    MyTerminalIO myterminal = new MyTerminalIO();
    // Здесь вызывается метод к объекту.
    myterminal.println("Hello, World!");
  }
}
```

При написании программы полезно выделять каждый блок программы отступом фигурных скобок так, чтобы было ясно, где начинается и где заканчивается каждый блок кода. Редактор TextPad сделает это для вас автоматически, и все приведенные примеры будут показаны в этой форме. Обратите внимание на то, что внутри блоков кода могут быть еще и вложенные блоки, и каждый из них пишется с отступом.

Приведем краткое описание некоторых аспектов вывода на экран с использованием метода `println()` и связанных с ним методов.

1. Метод `println()` выводит строку на экран и осуществляет переход на новую строку или возврат каретки. Вы также можете использовать метод `print()`, который просто печатает строку без возврата каретки. Полный вызов метода имеет следующий вид:

   ```
   myterminal.print("Some String");
   ```

2. Вы можете печатать арифметические величины, так же как и строки символов. Следующий вызов метода напечатает число 3:

   ```
   myterminal.println(3);
   ```

   В результате следующего вызова на экране также появится 3, но имеются существенные различия в значении этих двух вы-

зовов. Если в предыдущем примере был показан вывод числа, то в следующем — это вывод строки, содержащей символ 3:

```
myterminal.println("3");
```

3. Вы можете вывести на экран более одного параметра за один раз. Для этого используется знак +. Печатая строки, если вы хотите получить пробелы между словами, вы должны их указать. Если вы сделаете так:

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

   то увидите:

   ```
   Hello,World!
   ```

   Однако вы можете также сделать:

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

   или

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

   и тогда вы увидите:

   ```
   Hello, World!
   ```

4. Знак +, когда вы выводите на экран числа, имеет другое значение. Если вы набрали:

   ```
   myterminal.println(3 + 4);
   ```

   то увидите:

   ```
   7
   ```

5. Вы также можете комбинировать параметры в кавычках с числовым значением без кавычек, используя знак +. В таких случаях система автоматически конвертирует числовое значение в строку и осуществляет конкатенацию. Если вы сделаете так:

   ```
   myterminal.println("Hello" + 7)
   ```

   то вы увидите следующее:

   ```
   Hello7
   ```

   Более полное объяснение способов употребления знака + будет дано позже. Пока вы должны просто уметь пользоваться им правильно со строками и числами.

6. Хотя точка имеет специальное синтаксическое значение в разных местах кода Java, в тех случаях, где допускаются простые параметры или переменные, например при выводе на экран, она имеет свое привычное значение — точка десятичной дроби. Таким образом, если в вашей программе есть строка кода:

   ```
   myterminal.println(3.4);
   ```

   то вы увидите такое значение:

   ```
   3.4
   ```

7. В Java символ `\` используется для вывода на экран escape-последовательности. Это означает, что любой символ, следующий непосредственно за обратным слэшем, рассматривается как выводимый на экран символ. Это позволяет распечатывать символы, которые в противном случае были бы двусмысленными или синтаксически неверными в параметре печати. Например, если вы хотите напечатать двойные кавычки, то можете сделать следующее:

   ```
   myterminal.println("\"");
   ```

   Если вы хотите напечатать сам символ `\`, то можете указать строку кода:

   ```
   myterminal.println("\\");
   ```

   Символ `\` может быть также использован в некоторых командах для вывода строк на экран. Так, `\n` напечатает новую строку. Таким образом, если вы зададите строку кода:

   ```
   myterminal.println("Hello,\nWorld!");
   ```

   то увидите:

   ```
   Hello,
   World!
   ```

2.3. Документация Java API

На рис. 2.1 приводится отрывок из документации Java API (Java Application Programming Interface) для имеющегося в системе класса `Point`. Полную документацию Java API можно найти в Интернете по адресу: **http://java.sun.com/j2se/1.5.0/docs/api/**. Это лучший доступный справочник в тех ситуациях, когда у вас есть

вопросы по программированию на языке Java. После рисунка дано объяснение класса `Point` и некоторых положений документации.

java.awt

Class Point

```
java.lang.Object
  └ java.awt.geom.Point2D
      └ java.awt.Point
```

All Implemented Interfaces:
Serializable, Cloneable

```
public class Point
extends Point2D
implements Serializable
```

A point representing a location in (x, y) coordinate space, specified in integer precision.

Since:
JDK1.0

See Also:
Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class java.awt.geom.Point2D
Point2D.Double, Point2D.Float

Field Summary

int	**x** The *x* coordinate.
int	**y** The *y* coordinate.

Рис. 2.1. *(См. продолжение)*

Constructor Summary

Point() Constructs and initializes a point at the origin (0, 0) of the coordinate space.
Point(int x, int y) Constructs and initializes a point at the specified (*x*, *y*) location in the coordinate space.
Point(Point p) Constructs and initializes a point with the same location as the specified Point object.

Method Summary

boolean	**equals**(Object obj) Determines whether or not two points are equal.
Point	**getLocation**() Returns the location of this point.
double	**getX**() Returns the X coordinate of the point in double precision.
double	**getY**() Returns the Y coordinate of the point in double precision.
void	**move**(int x, int y) Moves this point to the specified location in the (*x*, *y*) coordinate plane.
void	**setLocation**(double x, double y) Sets the location of this point to the specified double coordinates.
void	**setLocation**(int x, int y) Changes the point to have the specified location.
void	**setLocation**(Point p) Sets the location of the point to the specified location.
String	**toString**() Returns a string representation of this point and its location in the (*x*, *y*) coordinate space.
void	**translate**(int dx, int dy) Translates this point, at location (*x*, *y*), by dx along the *x* axis and dy along the *y* axis so that it now represents the point (x + dx, y + dy).

Рис. 2.1. *Окончание*

Геометрически точка имеет некоторое расположение в пространстве. Класс `Point` в Java содержит информацию, необходимую для представления точки в двумерном пространстве. Для объектов этого класса переменными экземпляра будут координаты *x* и *y*. В Java координата *x* изменяется слева направо, координата *y* — сверху вниз. Позднее вы сможете графически изобразить точку на экране. Пока же экземпляры этого класса не будут иметь графического представления. Однако можно сконструировать экземпляры данного класса и совершать манипуляции ими посредством методов.

Информация в начале документации касается имени класса и его расположения. Классы, предоставляемые системой, находятся в пакетах. Если вы посмотрите информацию в начале, то увидите следующее: `Java.awt.Point`. Сокращение awt означает abstract windowing toolkit. Это имя пакета в Java, который включает классы, связанные с созданием графических интерфейсов. `Point` является одним из этих классов. Если в программе используется один из классов, предоставляемых системой, то подобная строка будет стоять в шапке кода:

```
import java.awt.Point;
```

Идея заключается в том, что данная строка делает класс доступным для использования в программе. Это будет показано в приводимых примерах программ. Можно импортировать весь пакет программ одновременно. Тогда нужно будет использовать знак `*`:

```
import java.awt.*;
```

Следующим разделом документации, который представляет для нас интерес, является раздел ***Field Summary***. То, что мы в *главе 1* называли переменными экземпляра, в документации называется *fields*. Другими словами, в документации говорится об объекте класса `Point`, имеющем 2 переменные экземпляра: *x*-координату и *y*-координату. Эти переменные имеют тип `int`, который означает, что указанные координаты могут принимать любые значения целых чисел.

Затем, в документации идет раздел ***Constructor Summary***. Конструкторы представляют собой специальные разделы кода, которые используются для создания экземпляров классов. Конструкторы

имеют форму, напоминающую форму методов — у них есть имя, за которым следуют скобки, содержащие или не содержащие параметры. Однако они не являются методами, и имя конструктора является таким же, как имя класса, к которому он принадлежит. Как видите, один класс может иметь более одного конструктора. Система различает их, т. к. у них разные списки параметров. В документации показаны типы параметров. Пока мы ограничимся примерами с параметром типа `int`.

Последней частью документации является ***Method Summary***. Здесь приводятся имена методов, их параметры и типы. Могут существовать разные методы с одним именем, которые различаются тем, что они имеют разный список параметров. Только через эти методы программа может влиять на переменные экземпляра — координаты *x* и *y* — объекта `Point`, который был создан. Как вы видите, `int` и `double` являются двумя различными числовыми типами. Значения типов будут описаны в следующей главе. Пока же все примеры будут ограничены использованием типа `int`, т. е. типа, принимающего целые числовые значения.

2.4. Создание и использование объектов

Написание кода для ваших собственных классов будет рассмотрено позже. Вы уже создали и использовали объект класса `MyTerminalIO`. Сейчас у нас есть возможность понять и правильно написать строки кода и маленькие логически завершенные программы, которые создают объекты классов, имеющихся в системе, и затем использовать их.

После того как класс `Point` был импортирован в программу, следующие две строки кода позволят вам:

- объявить имя, которое может быть использовано для объекта `Point`;
- сконструировать объект и придать ему имя, используя знак `=`.

```
Point mypoint;
mypoint = new Point(10, 20);
```

Таков синтаксис для вызова конструктора и передачи ему параметров:

```
new Point(10, 20);
```

Для того чтобы вызвать конструктор, необходимо использовать ключевое слово `new`. В показанном вызове координаты *x* и *y* инициализируются значениями 10 и 20 соответственно.

```
Point mypoint = new Point(10, 20);
```

Когда вы создаете объект и даете ему имя, это служит своего рода рукояткой для объекта. Имя `mypoint` является такой рукояткой. С ее помощью можно ссылаться на объект и манипулировать позже им в программе. Чтобы пояснить эту мысль, рассмотрим пример, в котором создается объект, но ему не придается никакого имени:

```
new Point(10, 20);
```

Синтаксически здесь все правильно. Эта строка не вызовет ошибки компилирования или запуска программы. Однако данный объект не может быть доступен в программе, т. к. у него нет имени.

Ранее мы говорили о приписывании объекту имени. Более точный термин, который будет использоваться в дальнейшем, — это *ссылка на объект*. Вызов конструктора возвращает безымянную ссылку на объект. Когда вы объявляете имя, например `mypoint`, и приписываете ему безымянную ссылку, то это имя и становится ссылкой на объект.

Представление о названных и безымянных ссылках можно проиллюстрировать еще одним примером. Рассмотрим следующую синтаксически правильную строку кода:

```
myterminal.println(new Point(10, 20));
```

Этот вызов работает, т. к. код, содержащий вложенные скобки, выполняется изнутри наружу. Сначала создается новая точка, и когда ее создание завершено, безымянная ссылка к этому объекту служит параметром для метода `println()`. Этот пример показывает, что вызов конструктора вызывает появление объекта, даже если он остается безымянным. Обратите внимание на то, что, несмотря на свою компактность, коды, в которых один вызов вы-

полняется внутри другого, или коды с безымянными ссылками, могут быть достаточно трудными.

Как только объект был создан и назван, к нему могут быть применены методы. Класс `Point` имеет метод `translate()`, который перемещает расположение точки на плоскости путем прибавления или вычитания значений к ее координатам *x* и *y*. Рассмотрим строку кода, которая иллюстрирует вызов метода к объекту:

```
mypoint.translate(30, 40);
```

Вызов имеет следующий вид: ссылка на объект, точка, имя метода и список параметров. Именно точка указывает на вызов метода к объекту. Важно отметить синтаксическую разницу между вызовом конструктора и вызовом метода. Вызов метода не использует ключевое слово `new`.

Результатом выполнения этой строки кода будет перемещение точки, которая выше была построена с координатами *x* и *y*, равными 10 и 20, в новое место с координатой *x*, равной 10 + 30, или 40, и координатой *y*, равной 20 + 40, или 60. Строка кода, которая выполняет вызов метода, не принимает вид приписывания. Значения приписываются переменным экземпляра в результате вызова, но работа по созданию приписываний скрыта внутри кода метода. Класс `Point` является черным ящиком, поскольку не обязательно видеть код метода для того, чтобы правильно применить этот метод.

Вот модель, по которой производится вызов метода и которая будет повторяться снова и снова, с изменениями:

```
object.method(parameters);
```

На этом этапе мы также можем вывести объект на экран. Методы вывода на экран будут принимать ссылки объекта в качестве параметров. Если данный объект существует, то следующая запись будет работающей строкой кода:

```
myterminal.println(mypoint);
```

Мы еще не программируем графику, поэтому на экране вы увидите:

```
Java.awt.Point[x=10,y=20]
```

Система просто говорит вам, какой это тип обращения и каковы текущие значения переменных экземпляра.

Теперь можем написать пример программы, которая включает все описанные ранее свойства.

```
import java.awt.Point;
/*  Это пример второй программы.  */
public class SecondProg
{
  public static void main(String[] args)
  {
    MyTerminalIO myterminal = new MyTerminalIO();
    Point mypoint = new Point(10, 20);
    myterminal.println(mypoint);
    mypoint.translate(30, 40);
    myterminal.println(mypoint);
  }
}
```

2.5. Ошибки

Ошибки программирования можно классифицировать различными способами. Один из них — это классифицировать ошибки в зависимости от того, когда они обнаруживаются: до выполнения программы, во время ее выполнения или после выполнения. Иным способом, связанным с первым, будет деление ошибок на синтаксические и логические. Синтаксические ошибки — это ошибки в использовании правил языка программирования. Логические ошибки — это ошибки в понимании проблемы и в предложении путей ее решения. Далее приводится общая информация об этих категориях ошибок.

- *Когда обнаруживается ошибка*: во время составления программы.

  Тип ошибки: синтаксические.

  Пояснение. Компилятор будет пытаться обнаружить синтаксические ошибки. Программа, которая содержит такие ошибки, не может быть выполнена.

- *Когда обнаруживается ошибка*: во время выполнения программы.

  Тип ошибки: синтаксические или логические.

  Пояснение. Система также может обнаружить ошибки во время выполнения и прекратить работу программы с сообщением об ошибке.

- *Когда обнаруживается ошибка*: после выполнения программы.

  Тип ошибки: логические.

  Пояснение. Программа, которая выполняется до конца, может выдать явно неверные результаты. Программа также может выдать неверные результаты, которые не будут столь очевидными. Задача программиста — тестировать программы и выверять результаты их работы. Компилятор и система не могут этого сделать.

Три самые распространенные, простые и наиболее часто досаждающие начинающим программистам синтаксические ошибки в Java — следующие:

- *отсутствие точки с запятой в конце строки кода*. Это будет раздражать вас, т. к. сообщение об ошибке будет говорить вам, что ошибка имеется в соседней строке, а не в строке, где пропущена точка с запятой;

- *отсутствие вторых скобок*. Это тоже будет мешать работе, потому что компилятор не будет распознавать строку, в которой произошла ошибка. Если вы будете располагать строки с отступом, то шансы сделать такую ошибку уменьшаются, а шансы заметить такую ошибку, в случае если вы ее допустили, увеличиваются;

- тот факт, что *Java различает прописные и строчные буквы*, также может быть причиной ошибок. Отказ от использования прописной буквы там, где это необходимо, или указание ее там, где она не нужна, может привести к проблемам, но человеку исключительно трудно заметить такие незначительные различия в коде.

  Ошибки в использовании прописной буквы могут быть особенно чреваты последствиями. В большинстве случаев они

происходят по невнимательности к деталям или в силу опечаток. Иногда они приводят к синтаксическим ошибкам, которые могут быть выявлены компилятором или которые будут выявлены во время работы программы. Однако они могут также привести к логическим ошибкам, которые станут очевидными только во время работы программы или когда проверяется результат программы. Иными словами, такие ошибки могут вызывать проблемы, которые, кажется, стирают грань между синтаксическими и логическими ошибками. По этой причине, а также потому, что трудно обнаружить пропущенную или ошибочно употребленную прописную букву, такие ошибки трудно обнаружить.

В языке, который использует ссылки на объекты, имеется еще один источник ошибок. Это происходит при попытке использовать ссылки на объект, который еще не был создан. Рассмотрим следующий фрагмент кода, содержащего подобную ошибку:

```
...
Point mypoint;
myterminal.println(mypoint);
...
```

В этом примере объявляется ссылка, но объект не создается. Ссылка ни к чему не отсылает. В коде эта ссылка передается методу `println()` в качестве параметра. Если запустить программу, то компилятор выдаст сообщение об ошибке, указывая, что объект `mypoint` не был инициализирован. Теперь рассмотрим следующий пример:

```
...
Point mypoint;
mypoint.translate(30, 40);
...
```

На этот раз вызывается метод по ссылке, которая не отсылает к объекту. Компилятор обнаружит и эту ошибку и сообщит вам, что `mypoint` не был инициализирован.

Компилятор Java и среда, в которой выполняется программа, стараются обнаружить ошибки во время выполнения и выводят на

экран сообщения об ошибках. Однако они не могут обнаружить все ошибки, и иногда сообщения об ошибках могут быть достаточно непонятными. По мере того как все более сложные понятия и синтаксис становятся доступными, а программы усложняются, ошибки могут становиться более трудноуловимыми, и программист должен быть осторожным во время программирования и внимательным при выявлении ошибок в программах. Ссылки на объект являются принципиально важной частью программирования на Java. Их правильное использование и ошибки, вытекающие из их неверного использования, будут более подробно рассмотрены в следующих главах.