Глава 3

Базовые компоненты Java

В этой главе...

- Поговорим на языке Java
- Ваша первая программа на Java
- Как работает ваша первая программа
- Не говорите “Без комментариев...”

“Все мысли, которые имеют
огромные последствия, всегда просты”.
Лев Толстой

Этот принцип относится практически ко всем областям человеческой жизни, в том числе к компьютерному программированию. Поэтому в данной главе применяется многоуровневый подход. Сначала вы ознакомитесь с некоторыми компонентами программы Java, а затем, узнав множество подробностей, увидите, что они подчиняются простым принципам.

Поговорим на языке Java

Попытайтесь представить себе весь русский язык одновременно. Что вы видите? Слова, слова, слова... Огромное количество слов! Рассматривая русский язык “под микроскопом”, вы отчетливо видите каждое слово. Однако отойдите на шаг назад, и вы увидите еще кое-что, кроме слов:

- ✓ грамматика языка;
- ✓ имена, слова и устойчивые группы слов.

Первая категория (грамматика) содержит правила наподобие “подлежащее должно быть того же рода и числа, что и сказуемое”. Вторая категория содержит названия конкретных и абстрактных объектов. Например, четыре буквы “стол” обозначают любой предмет с четырьмя ножками, на котором что-нибудь лежит или может лежать, а шесть букв “Берлин” обозначают конкретный город, столицу Германии.

Язык программирования Java обладает многими свойствами естественного языка. В нем есть слова, грамматика, часто используемые имена, стилистические идиомы и ряд других компонентов естественных языков.

Грамматика и общие имена

Люди в Sun Microsystems, которые создавали Java, считали, что язык должен состоять из двух частей. Как и естественный язык, он должен иметь грамматику и набор слов.

- **Грамматика Java** определяется спецификацией языка, т.е. официальным документом, в котором написано, как используются компоненты языка. В принципе, вы можете почитать спецификацию Java, но для вас она будет бесполезной, потому что она создавалась главным образом для разработчиков компиляторов. Вам лучше подойдут учебники по Java. Из них вы узнаете все нужные вам правила, такие как "Каждой открывающей скобке должна соответствовать одна закрывающая" или "Чтобы умножить два числа, поместите между ними звездочку".

  Наряду с термином *грамматика* часто можно встретить термин *синтаксис*. Строго говоря, синтаксис — это раздел грамматики, но этот вопрос чисто академический. Даже профессиональные программисты не очень беспокоятся о том, какая разница между грамматикой и синтаксисом.

- **Классы API** (Application Programming Interface — интерфейс программирования приложений). Библиотека стандартных классов API, входящая в состав каждой виртуальной машины Java, содержит тысячи инструментов, позволяющих выполнять практически любые операции, которые может выполнять компьютер. Например, библиотека API содержит класс `Math`, который предоставляет большое количество математических функций, таких как синус, косинус, арксинус, арккосинус и т.п. Другой пример — стандартный класс `JFrame`, отображающий на экране окно, в которое можно поместить элементы интерфейса (кнопки, флажки, списки и т.п.).

  Полный официальный справочник по классам API можно найти на сайте `http://java.sun.com/javase/reference/api.jsp`, но учитывайте, что на момент публикации этой книги адрес может измениться. В таком случае откройте сайт Oracle и найдите ссылку на справочник по библиотечным классам.

Первая часть Java — грамматика языка — сравнительно небольшая. Это характерная особенность Java, довольно приятная. В других языках программирования количество правил в несколько раз больше, а в некоторых языках — в десятки раз.

Вторая часть Java — набор стандартных классов API — выглядит устрашающе большой. Стандартная библиотека содержит 5 тысяч инструментов, причем их количество продолжает увеличиваться с каждой новой версией JDK. Однако не пугайтесь, вам не нужно помнить все классы API. Даже профессиональные программисты помнят не более десятка наиболее "ходовых" классов, а с остальными знакомятся по необходимости, открыв справочник по классам. Вы также будете помнить только то, что часто используете.

Ни один человек в мире не знает все о Java API. Если вы будете разрабатывать программы с графическим интерфейсом пользователя для настольных компьютеров, вы будете часто использовать класс `JFrame` и, соответственно, помнить его структуру. Если же вы редко будете писать программы,

открывающие окна, но в некоторый момент вам понадобится окно, вы почитаете описание класса `JFrame` и узнаете, как это делается. Если программисту Java (даже самому крутому профессионалу) запретить пользоваться справочником по стандартным классам, он не сможет написать на Java почти ничего.

Каждый раз, когда вы пишете программу на Java, даже самую маленькую, вы создаете классы, которые обладают теми же правами, что и стандартные классы Java API. Интерфейс API — это набор классов и других инструментов, созданных программистами — участниками официальных проектов JCP (Java Community Process — процесс сообщества Java) и OpenJDK Project (Открытый проект пакета инструментов Java). Вы пока что, как я понимаю, не участвуете в JCP или OpenJDK, но, когда прочитаете эту книгу, вполне можете предложить свои услуги и получить часть работы.

Если вас интересует деятельность JCP, посетите сайт `http://jcp.org`. С проектом OpenJDK можно ознакомиться на сайте `http://openjdk.java.net`.

Ребята из JCP не пытаются держать в секрете свою работу над официальной библиотекой Java API. При желании вы также можете ознакомиться с исходными кодами официальных стандартных библиотек. При установке Java на ваш компьютер инсталляционная программа поместила на ваш диск архивный файл `src.zip`. В нем находятся все исходные коды Java API. Можете распаковать его с помощью любой программы архивирования.

Слова в программе Java

Принято считать, что в исходном коде Java используются слова двух категорий: *идентификаторы* и *ключевые слова*. Однако без дополнительных разъяснений такая классификация может ввести в заблуждение. Поэтому лучше считать, что в Java используются слова трех категорий: ключевые слова, идентификаторы API и пользовательские идентификаторы. *Пользовательские идентификаторы* — это слова, придумываемые программистом во время написания программы, а *идентификаторы API* — это имена переменных и классов, используемые в стандартных библиотечных классах Java.

Отличие между этими тремя категориями приблизительно такое же, как между словами естественного языка. В предложении “Сергей купил тетрадь” слово “купил” играет ту же роль, что и ключевое слово Java. Независимо от того, кто и в каком месте его использует, оно всегда означает приблизительно одно и то же — акт обмена денег на продукт. Слово “тетрадь” похоже на идентификатор API тем, что оно везде обозначает предмет, состоящий из нескольких сшитых листов. Конечно, в естественном языке из этого правила есть много исключений. Некоторые слова, называемые “синонимами”, могут обозначать разные предметы, но мы не будем обращать на это внимания, потому что язык программирования более строг, чем естественный. В нем нет синонимов, и каждое ключевое слово всегда означает одно и то же.

В упомянутом выше предложении слово “Сергей” похоже на пользовательский идентификатор Java, потому что оно является именем конкретного объекта и в разных местах может обозначать разные предметы. В русском языке слова “Витя”, “Коля”, “Сидоров” и подобные им не имеют предустановленного, общего для всех значения. Значение такого слова зависит от того, где оно используется, ведь, как вы понимаете, есть много Сергеев, и это не один и тот же человек.

05ch03.indd 51 19.03.2012 11:10:07

Теперь рассмотрим предложение "Юлий Цезарь завоевал Египет". На первый взгляд, идентификатор "Юлий Цезарь" здесь такой же, как в предыдущем примере — он обозначает человека. Однако его значение не зависит от того, кто и где его использует. Оно всегда означает римского императора, причем каждому читателю понятно, какого именно императора. Если бы русский язык был языком программирования, имя "Юлий Цезарь" было бы идентификатором API.

Итак, резюмируем вышесказанное: в исходном коде Java используются следующие слова.

- **Ключевые слова.** Это слова, имеющие специальное значение и определенные в грамматике Java. Значение ключевого слова всегда одно и то же в любой программе. Примеры ключевых слов — `if`, `else`, `while` и т.п.

  Количество и написание всех ключевых слов определено в спецификации Java, утверждаемой комитетом JCP. Никакой придуманный вами идентификатор не должен совпасть ни с одним ключевым словом Java.

  Полный список ключевых слов Java можно увидеть в шпаргалке, прилагаемой к данной книге.
- **Идентификаторы.** Идентификатор — это имя чего-либо (класса, объекта, метода, события и т.д.). В спецификации языка идентификаторы не определены (определено лишь общее понятие "идентификатор"), поэтому в разных программах они могут иметь разные значения.
- **Пользовательские идентификаторы.** Создавая программу на Java, вы придумываете новые имена для классов и других сущностей, встречающихся в исходном коде. Предположим, вы назвали что-то именем `Prime`, а программист в соседнем отделе случайно, не сговариваясь с вами, тоже назвал что-то именем `Prime`. Что произойдет в результате этого? Ничего! Слово `Prime` не имеет в Java никакого предустановленного значения. Кроме того, ваши программы не контактируют друг с другом. Поэтому имена `Prime` в них никак не затрагивают друг друга. Впрочем, иногда имена пересекаются. Такая ситуация называется *конфликтом имен*. Это может произойти, например, если вы и ваш коллега работаете над одним проектом и случайно назовете два разных класса одним и тем же именем.
- **Идентификаторы API.** Участники проекта JCP создали классы Java API, в которых используется более 40 тысяч имен. Классы API поставляются в дистрибутиве JDK, поэтому все эти имена доступны для вас. Можете использовать их в своей программе. Примеры идентификаторов API — `String`, `Integer`, `JWindow`, `JButton`, `JTextField` и т.п.

Строго говоря, значения идентификаторов API не увековечены в камне или бронзе. Вы можете использовать любой идентификатор API (например, `JButton`) в своем коде не по назначению. Конечно, он будет обозначать вашу сущность, а не ту, для обозначения которой он предназначен в API. Программа будет работать правильно, но не делайте так, потому что сами запутаетесь и собьете с толку другого программиста, который будет работать с вашей программой, потому что он привык к тому, что `JButton` — это класс кнопки. Кроме того, вы не сможете использовать в своей

05ch03.indd 52 19.03.2012 11:10:07

программе стандартную кнопку `JButton`. Программисты Sun Microsystems, JCP и OpenJDK проделали огромную работу, чтобы предоставить вам простую, удобную кнопку, а вы презрительно ее отбросите. Впрочем, если вам понадобится кнопка, вы и без моих увещеваний не будете называть именем `JButton` другую сущность в своей программе.

Ваша первая программа на Java

Когда вы впервые посмотрите на программу Java, написанную другим человеком, вы испытаете легкое головокружение. Вы с ужасом осознаете, что ничего не понимаете в этом коде. За свою жизнь я написал сотни программ на Java, но до сих пор, читая чужой исходный код, чувствую себя неуверенно.

Чтобы понимать некоторый код, нужно знать назначение каждого идентификатора, структуру алгоритма, компоновку интерфейса и многое другое. Увидеть все это с первого взгляда невозможно даже для опытного программиста. Чтобы понять код, он также вынужден рассматривать его очень долго, может быть, даже много дней. Не верьте человеку, который утверждает, что без труда понимает любой код. Даже опытные программисты подходят к новому проекту медленно и аккуратно.

Поэтому начнем с простейшей программы, приведенной в листинге 3.1. В этом коде я проиллюстрирую вам несколько важных концепций, которые мы рассмотрим в следующем разделе. Это такие концепции, как классы, методы, инструкции Java и структура программы.

Листинг 3.1. Исходный код простейшей программы на Java

```
class Displayer {
  public static void main(String args[]) {
    System.out.println("Мне нравится Java!");
  }
}
```

Эта программа всего лишь выводит на консоль фразу `Мне нравится Java!`. Запустите программу в интегрированной среде разработки Eclipse, и вы увидите эту фразу в окне консоли Eclipse (рис. 3.1). Такой результат может показаться обескураживающим. Вы прочитали уже несколько глав, и все это лишь для того, чтобы вывести на экран простую фразу. Однако это только начало. Через некоторое время (не такое уж долгое) ваша программа Java будет делать все, чего вы от нее захотите.

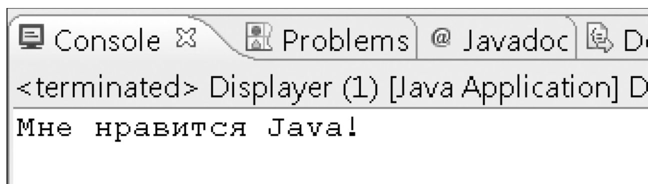

Рис. 3.1. Результат выполнения программы в Eclipse

Подробные инструкции по работе с Eclipse приведены в приложении Б.

05ch03.indd 53 19.03.2012 11:10:07

Как работает ваша первая программа

В этом разделе приведено описание и объяснение программы, показанной в листинге 3.1.

Классы

Поскольку Java является объектно-ориентированным языком программирования, ваша главная задача — определить классы и создать объекты (т.е. экземпляры классов). Если вы этого еще не знаете, почитайте еще раз главу 1.

В особые дни, когда я становлюсь сентиментальным, я рассказываю людям о том, что в Java концепция объектно-ориентированного программирования реализована более чисто и полно, чем во многих других так называемых "объектно-ориентированных" языках. Я говорю это потому, что в Java вы не сможете ничего сделать, пока не создадите соответствующий класс. Это настолько сильно войдет у вас в привычку, что, когда ваша супруга попросит вас сходить в магазин за хлебом, вы ответите: "Извини, дорогая, но ты должна вложить эту просьбу в класс".

В программе Java есть только классы и ничего кроме классов. Буквально все (данные, методы, идеи, структура пользовательского интерфейса, ошибки, гениальные решения и досадные промахи) находится в классах. Для каждого класса нужно придумать имя. Программа, показанная в листинге 3.1, состоит из одного класса, который я назвал `Displayer`. Поэтому исходный код класса начинается словами `class Displayer` (рис. 3.2).

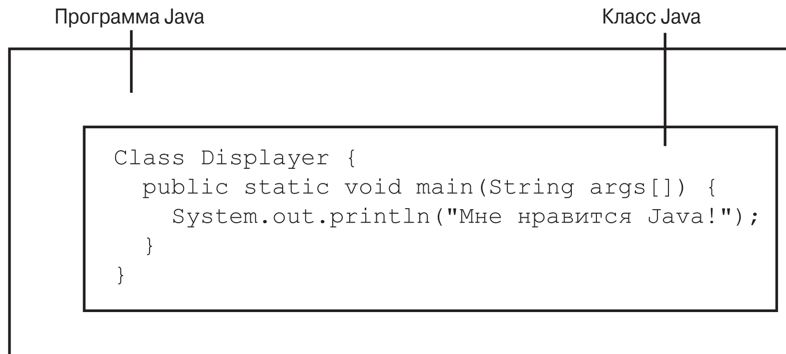

Рис. 3.2. Эта программа состоит из одного класса

Первое слово, `class`, — это ключевое слово Java. Независимо от того, кто пишет программу и что она будет делать, слово `class` всегда используется одинаково: оно обозначает определение класса и находится перед его именем. Слово `Displayer` — это пользовательский идентификатор, служащий именем класса. Я придумал это имя, когда создавал класс.

Исходный код Java чувствителен к регистру букв. Это означает, что `class` и `Class` являются разными словами. Если заменить первое слово вторым, программа не будет компилироваться, пока вы не исправите ошибку.

> К регистру букв чувствительны как ключевые слова, так и идентификаторы. Вы можете назвать класс `displayer`, но, если в другом месте вы сошлетесь на него как на `Displayer`, компилятор сообщит, что не нашел этого класса. Согласно принятым в сообществе Java соглашениям, имена классов принято начинать с большой буквы, а имена объектов (т.е. экземпляров классов) — с малой. Если вы нарушите это правило, компилятор не отметит имя как ошибочное. Ему все равно, однако ваши коллеги, глядя на код, будут удивляться, почему вы используете такие странные имена, и вы не найдете, что им ответить.

Если назвать класс, например, `DogAndPony`, программа Eclipse поместит его в файл `DogAndPony.java`. Компилятор создаст байтовый код класса и поместит его в файл `DogAndPony.class`.

Методы

Предположим, вы работаете механиком в гараже. Ваш босс, который всегда спешит, имеет привычку отдавать приказы скороговоркой: “Нужно починитьГенератор в вон-том старомСеромФорде”. Мысленно вы составляете список задач: “Загнать машину на яму, открыть капот, отвинтить крепление, залезть в яму, снять генератор, открыть корпус генератора и т.д.”. Во всем этом можно выделить три компонента.

- ✓ **Имя задания, которое вы должны выполнить.** В данном случае имя задания — `починитьГенератор`.
- ✓ **Список задач, которые для этого нужно решить.** Задачи решаются строго в той последовательности, в которой они указаны в списке. Например, нельзя сначала отвинтить крепление генератора, а затем открыть капот.
- ✓ **Сердитый босс, который дал вам задание.** Ваш босс произнес имя задания `починитьГенератор`. Этого достаточно, потому что вы знаете, как это делается, т.е. имеете упомянутый выше список задач.

В этом сценарии уместно будет ввести понятие *метод*. Вы знаете, как починить генератор. На языке программирования это означает, что у вас есть метод починки генератора. Ваш босс посредством словесного приказа приводит этот метод в действие, а вы выполняете все операции, заложенные в метод.

Если вы согласны с таким пониманием термина “метод”, значит, готовы к пониманию того, что такое метод в языках программирования. В Java метод — это список инструкций, которые должен выполнить компьютер. У каждого метода есть персональное имя, которое в отличие от человеческих имен (Сергей, Лена и т.п.) должно быть уникальным в своем классе. Однако в отличие от вашего босса программист не приказывает голосом, а пишет конструкцию языка, которая называется *вызов метода*. Это означает следующее: чтобы компьютер выполнил список инструкций, заложенных в методе, программа должна “вызвать метод”.

Я никогда не писал программу для робота, ремонтирующего генератор, но если бы мне пришлось делать это, я назвал бы метод `починитьГенератор`. Список инструкций в теле метода `починитьГенератор` выглядел бы приблизительно так, как код в листинге 3.2.

Листинг 3.2. Определение метода

```
void починитьГенератор(Машина машина) {
  загнатьМашинуНаЯму(машина, яма);
  поднятьКапот(капот);
  отвинтитьГенератор(гайка, гайка, гайка, гайка);
  ...
}
```

Не рассматривайте листинги 3.2 и 3.3 слишком придирчиво. Это воображаемый код. Компьютер не может поднять капот или отвинтить гайку. Единственное достоинство этого кода в том, что он выглядит точно так же, как реальный код Java. В принципе, это синтаксически правильный код Java, но он никогда не будет выполнен (сейчас уже есть роботы, умеющие отвинчивать гайки, но в них не используется код, приведенный в листинге 3.2).

Итак, у нас есть метод Java, который нужно запустить на выполнение. Для этого в другом месте кода Java необходимо вставить инструкцию вызова метода `починитьГенератор`. Инструкция вызова метода представляет собой имя метода, после которого в скобках приведен список параметров. В данном случае нужно сообщить методу, в какой машине нужно отремонтировать генератор. Для этого укажем в списке единственный параметр — имя машины (листинг 3.3).

Листинг 3.3. Вызов метода

```
починитьГенератор(старыйСерыйФорд);
```

Теперь, когда вы знаете, что такое метод и как он работает, мы можем глубже “копнуть” терминологию.

- ✓ **Определение метода.** В русском языке слово “определение” имеет два разных значения: во-первых, вычисление, выяснение, обнаружение чего-либо, и, во-вторых, задание, постулирование, указание на то, что это такое. Возможно, вы помните, как на уроке геометрии в школе учили “определение треугольника” и “определение окружности”. В математике и программировании слово “определение” всегда используется только во втором смысле. В данном случае определение метода — это список инструкций, т.е. вся информация о том, что компьютер должен сделать.

  В других книгах часто используется термин *объявление метода*. Это синоним, т.е. то же самое, что определение метода.
- ✓ **Заголовок метода.** Первая строка в определении, в которой указано имя метода и приведен список принимаемых параметров.
- ✓ **Тело метода.** Содержит список инструкций.
- ✓ **Вызов метода.** Инструкция, расположенная в другом месте программы и запускающая метод на выполнение (см. листинг 3.3).

05ch03.indd 56 19.03.2012 11:10:07

ПОМНИ!

Определение метода указывает компьютеру, что он должен сделать при вызове метода. Инструкция вызова метода приказывает компьютеру сделать это. Определение не инициирует выполнение метода, оно лишь содержит информацию о том, что должно быть сделано. Определение и вызов метода — это две разные части программы. В принципе, вызов можно разместить в теле этого же метода. Такой метод будет называться *рекурсивным*. Однако вам пока что не придется использовать такую хитрую конструкцию.

ПОМНИ!

В тексте статей, книг и описаний, посвященных программированию, принято при упоминании метода ставить после его имени пару скобок, чтобы читатель с первого взгляда видел, что это метод, а не какая-либо другая сущность языка, например `починитьГенератор()`.

Главный метод программы

На рис. 3.3 приведена копия кода, показанного в листинге 3.1. Код содержит определение метода `main` (данное имя приведено в заголовке метода). Пока что не беспокойтесь о назначении ключевых слов `public`, `static`, `void`, `String` и др. Я объясню их в следующих главах.

Программа Java

Заголовок главного метода

Класс Java

```
Class Displayer {
  public static void main(String args[]) {
    System.out.println("Мне нравится Java!");
  }
}
```

Тело главного метода

Рис. 3.3. Главный метод

Как и любой метод Java, главный метод — это рецепт.

```
Как приготовить пирог {
  разогреть печь;
  раскатать тесто;
  приготовить начинку;
  ...
}
```

В нашем случае рецепт выглядит так.

```
Как выполнить главный метод {
  вывести на консоль фразу "Мне нравится Java!";
}
```


05ch03.indd 57 19.03.2012 11:10:07

Ключевое слово `main` (главный) играет в Java специальную роль. В частности, в коде программы нельзя написать инструкцию вызова метода `main()`. Имя `main` обозначает метод, который автоматически вызывается операционной системой при запуске программы. Следовательно, работа любой программы начинается с выполнения тела метода `main()`.

Взгляните еще раз на рис. 3.1. Когда вы запускаете программу `Displayer`, операционная система автоматически запускает метод `main()` и выполняет расположенные в нем инструкции. В программе `Displayer` метод `main()` содержит только одну инструкцию — приказ компьютеру вывести на консоль фразу `Мне нравится Java!` (см. рис. 3.1).

Все инструкции любой программы расположены только в методах. Компьютер делает только то, что ему приказывают инструкции, а инструкция выполняется только после вызова метода, в котором она расположена. Следовательно, любой метод, который нужно выполнить, должен быть вызван в коде программы. Но это не относится к методу `main()`, который вызывается операционной системой автоматически при запуске программы.

Практически в каждом языке программирования есть некоторый аналог метода Java. Если вы работали с другими языками, вам должны быть знакомы такие понятия, как "процедура", "функция", "подпрограмма", "оператор `PERFORM`", "оператор `call`". Все это фактически методы и вызовы методов, но в других языках они называются по-другому. Однако в любом языке программирования метод (или как он там называется) обладает именем и содержит список инструкций.

Как приказать компьютеру выполнить нужную операцию

Очень просто: написать инструкцию и вставить ее в тело метода. В листинге 3.1 есть только одна инструкция, заключенная в рамку на рис. 3.4. Она приказывает компьютеру вывести на консоль фразу `Мне нравится Java!`.

```
Class Displayer {
  public static void main(String args[]) {
    System.out.println("Мне нравится Java!");
  }
}
```

Инструкция, представляющая собой вызов метода System.out.println()

Рис. 3.4. Инструкция Java

Конечно, в Java есть много разных видов инструкций. Один из них — вызов метода. В листинге 3.3 показано, как выглядит вызов метода, а на рис. 3.4 содержится следующая инструкция вызова метода:

```
System.out.println("Мне нравится Java");
```


05ch03.indd 58 19.03.2012 11:10:07

Выполняя эту инструкцию, компьютер вызывает стандартный библиотечный метод System.out.println(), который выводит на консоль заданную фразу.

В Java имена могут содержать точки. Что они означают, мы рассмотрим в главе 9.

На рис. 3.5 проиллюстрирован вызов метода System.out.println(). Фактически в программе Displayer используются два метода. Вот как они работают.

- ✓ **Программа содержит определение метода main().** Я сам написал метод main(). Он вызывается операционной системой автоматически при запуске программы.
- ✓ **Программа содержит вызов метода System.out.println().** Это единственная инструкция метода main(). Иными словами, вызов метода System.out.println() — это единственное, что делает метод main().

  Определение метода System.out.println() находится в стандартной библиотеке Java API, устанавливаемой при инсталляции на компьютер пакета JDK или JRE.

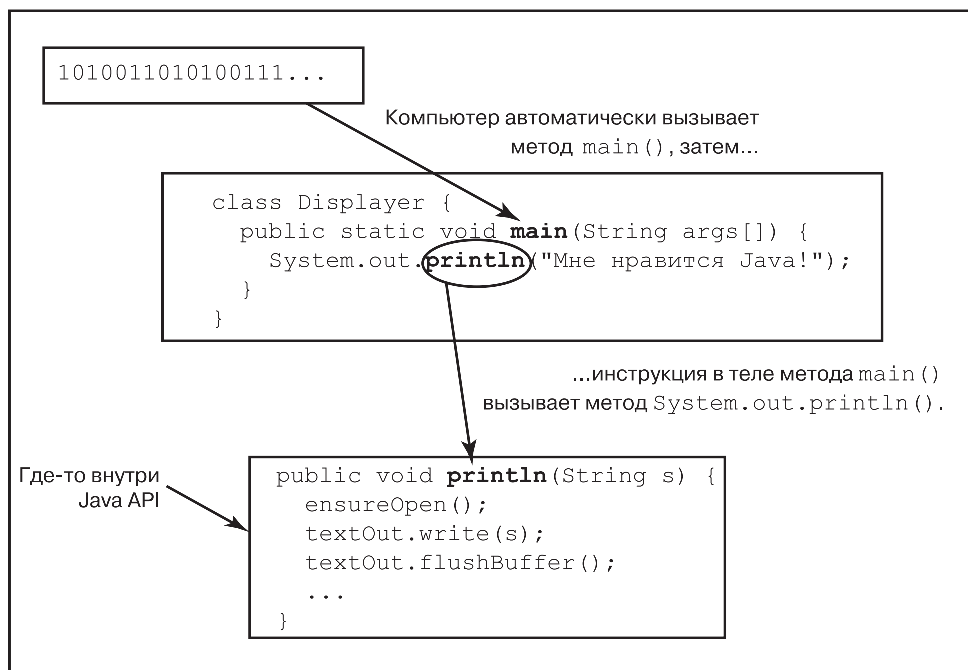

Рис. 3.5. Вызов метода System.out.println()

Если вы любопытны, можете посмотреть определение метода System.out.println() в библиотеке Java API. Это всего лишь еще один метод, написанный на языке Java. Однако пользы от ознакомления с кодами

стандартных библиотечных методов мало, вернее, совсем никакой. Лучше сосредоточьте внимание на применении метода. Вы должны знать, что он делает и какие параметры ему нужно передать.

Каждая инструкция Java заканчивается точкой с запятой (например, заключенная в рамку инструкция на рис. 3.4). Другие строки на рис. 3.4 не заканчиваются точками с запятыми, потому что это не инструкции. Например, заголовок метода не является инструкцией, потому что он не приказывает компьютеру сделать что-либо, а всего лишь информирует его о том, как называется метод и какие параметры ему нужно передать. Инструкции можно размещать по строкам произвольно — по нескольку инструкций в одной строке или одну инструкцию в нескольких строках. Границами между инструкциями служат точки с запятыми, а разрывы строк никак не влияют на них. Тем не менее в сообществе Java принято размещать по одной инструкции в каждой строке, чтобы код было легче читать.

Каждая инструкция Java должна заканчиваться точкой с запятой.

Фигурные скобки

Вспомните, как много лет назад школьный учитель убеждал вас в том, что во всем важен порядок и что нужно правильно организовать, разложить по полочкам свои дела, идеи, мысли, планы и знания. Фактически программа Java — это и есть хорошо структурированная, тщательно организованная информация, разложенная по полочкам. В листинге 3.1 программа начинается с заголовка класса `Displayer`, сообщающего о том, что дальше расположено определение класса. В класс вложен метод `main()`. Его заголовок сообщает, как он называется и какие параметры ему нужно передать, а также о том, что непосредственно после него начинается тело метода.

Вы должны строго придерживаться этой структуры. Нельзя написать заголовок метода внутри его тела или определение метода за пределами определения класса. Программа должна иметь древовидную структуру, в которой одни элементы вложены в другие. Инструкции вложены в метод. Методы вложены в класс. Классы вложены в программу.

Однако программа представляет собой текст, т.е. последовательность символов. Как же отобразить в ней древовидную структуру, не прибегая к помощи рисунка? Это делается с помощью следующих средств.

- ✓ **Фигурные скобки.** В коде Java они обозначают элементы программы. Например, тело метода (как и тело класса) должно быть заключено в фигурные скобки.
- ✓ **Отступы.** В принципе, программу Java можно записать в произвольной форме, например ввести все подряд, без пробелов и перевода строк. Синтаксически это будет правильная программа. Она будет работать точно так же, как хорошо структурированная. Но ее будет тяжело читать визуально. Чтобы облегчить понимание структуры программы, необходимо использовать отступы.

05ch03.indd 60 19.03.2012 11:10:07

В древовидной, иерархической структуре программы Java высший элемент — класс. Тело класса заключено в фигурные скобки. Следовательно, после заголовка класса должна располагаться открывающая фигурная скобка, а после тела класса должна быть закрывающая фигурная скобка (листинг. 3.4).

Листинг 3.4. Тело класса заключено в фигурные скобки

```
class Displayer {
  public static void main(String args[]) {
    System.out.println("Мне нравится Java!");
  }
}
```

Следующий, более низкий уровень иерархии — метод. Как и класс, он начинается с заголовка, и его тело должно быть заключено в фигурные скобки (листинг 3.5). Чтобы было лучше видно, что метод вложен в класс, код метода введен с отступом.

Листинг 3.5. Тело метода заключено в фигурные скобки

```
class Displayer {
  public static void main(String args[]) {
    System.out.println("Мне нравится Java!");
  }
}
```

Самый нижний элемент иерархической структуры программы — инструкция. Чтобы ее было лучше видно, она введена с самым большим отступом (см. рис. 3.5).

Никогда не забывайте о древовидной иерархической структуре программы Java.

Если ошибочно разместить фигурную скобку не там, где нужно, или опустить ее, программа, скорее всего, не будет работать. Компилятор сгенерирует сообщение о синтаксической ошибке. Но может быть и хуже: программа будет работать, но не так, как нужно, компилятор не сообщит об ошибке, вы ее не заметите, и она обнаружится только у заказчика, что повлечет для вас большие неприятности.

Если написать код без отступов, информирующих о структуре программы, она будет работать правильно, компилятор не будет возражать, но вам и тем более другому программисту будет тяжело понять, как работает программа.

Можете представить себе структуру программы как древовидную иерархическую структуру оглавления книги: книга разделена на части, в которых размещены главы, в которых, в свою очередь, размещены разделы (рис. 3.6). Существует также еще более причудливое представление структуры программы (рис. 3.7).

Многие среды разработки, в том числе Eclipse, создают отступы автоматически. Вы можете отредактировать величину отступов и правила их расстановки или вообще отключить их создание.

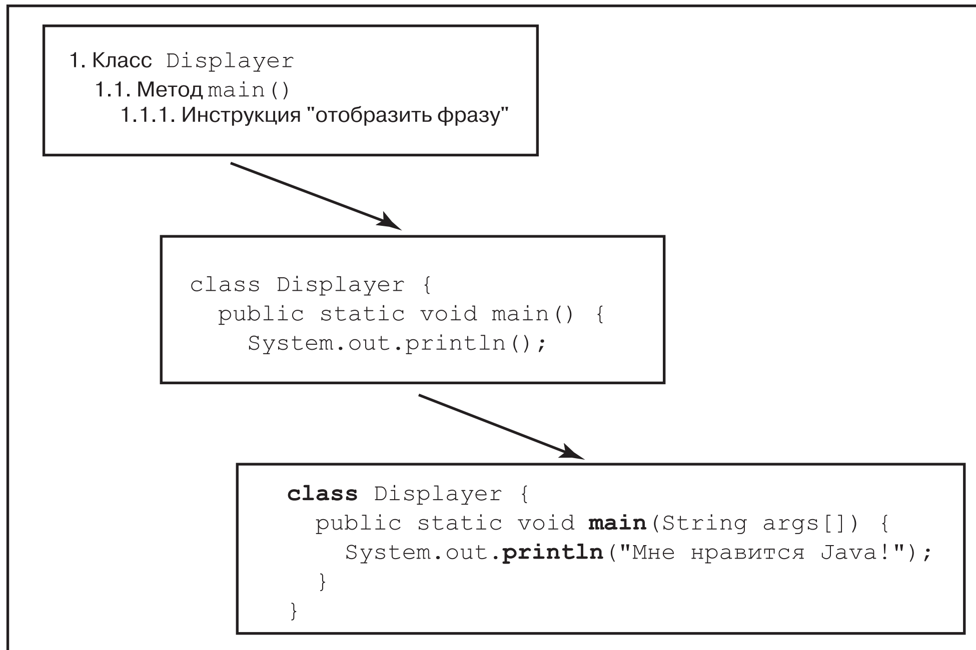

Рис. 3.6. Представление иерархической структуры программы

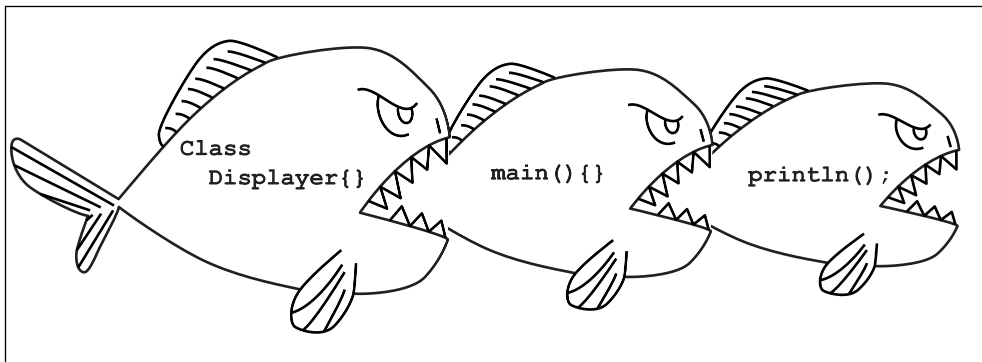

Рис. 3.7. Класс больше, чем метод; метод больше, чем инструкция

Не говорите "Без комментариев..."

Люди собрались вокруг костра, чтобы послушать древнюю легенду о талантливой программистке, ленившейся вставлять комментарии. Чтобы не стать объектом судебных преследований, назовем ее Джейн Про. Много лет Джейн работала над "Святым Граалем" компьютерных технологий — программой, которая могла бы самостоятельно думать. Если бы эту программу удалось сделать, она могла бы без участия и помощи человека выполнять нужную работу, изучать меняющуюся ситуацию и даже саму себя совершенствовать. День за днем долгими бессонными ночами Джейн продвигалась к

05ch03.indd 62 19.03.2012 11:10:07

цели, и уже близок был момент, когда она рассчитывала увидеть в работе программы искру самостоятельной мысли.

Но в один прекрасный день произошло следующее событие. Когда она наконец-то почти высекла долгожданную искру, в ее почтовый ящик (традиционный, не электронный) пришло бумажное письмо от ее страховой компании по поводу ее здоровья. Не пугайтесь, в письме было не сообщение об ужасном заболевании, а всего лишь рутинная просьба позвонить в офис компании для уточнения некоторых формальностей. В стандартной форме страховой компании был пункт о ее дате рождения (как будто он мог измениться со времени заключения договора о социальном страховании). В последний раз она, будучи в состоянии глубокой задумчивости (Святой Грааль требует полной самоотдачи), в графе "Год рождения" написала "2011". Несовершенная программа (ей было очень далеко до Святого Грааля), установленная на компьютере компании, автоматически подсчитала сумму очередного страхового платежа, и он оказался отрицательным. Другая несовершенная программа, перечисляющая деньги на счет Джейн, не поняла, что нужно делать в таком случае, и все застопорилось в ожидании вмешательства человека.

Джейн набрала телефонный номер компании. После 20 минут блуждания по голосовому меню она наконец услышала человеческий голос. После 20 минут переговоров менеджер компании сообщила: "Мне очень жаль, но для решения этого вопроса вы должны позвонить по другому номеру". Звонок по другому номеру привел к тому же результату, но фраза немного отличалась от прежней: "Мне очень жаль, но оператор дал вам неправильный номер, вы должны обратиться к нему еще раз".

Через несколько месяцев, после 800 часов телефонных переговоров, ее ухо опухло, но она все же добилась обещания, что компания исправит графу "Год рождения" в ее форме. Воодушевленная таким успехом, она немедленно вернулась к своему Святому Граалю, но — о ужас! — вдруг увидела, что не может вспомнить, что делают и зачем нужны все эти буковки и закорючки, которые она сама же написала в огромном количестве.

Она смотрела на свою работу, все больше и больше казавшуюся ей сном, который она не может вспомнить утром после пробуждения. Код ничего не означал для нее. Она написала тысячи строк кода, и ни в одной строке не было комментария, объясняющего, как работает программа. Она не оставила ни единого намека, который помог бы ей вспомнить, о чем она думала, когда писала эти строки. В полной прострации Джейн выключила компьютер. Ей пришлось оставить работу над проектом.

Добавление комментариев в код

В листинге 3.6 показана версия программы, приведенной в листинге 3.1, но с комментариями. Кроме ключевых слов, идентификаторов и знаков пунктуации, предназначенных для компилятора, код теперь содержит текст, предназначенный для чтения человеком.

Листинг 3.6. Код программы с комментариями

```
/*
 * Листинг 3.6 книги "Java для чайников, 5-е издание"
 *
 * Copyright 2011, Издательство Диалектика.
 * Все права защищены.
*/
```

```
/**
 * Текст класса Displayer, который отображает заданный
 * фрагмент текста на консоли.
 *
 * @author  Barry Burd
 * @version 1.0 10/24/11
 * @see     java.lang.System
 */
class Displayer {
 /**
  * Метод main(), содержащий выполняющийся код.
  *
  * @param args (см. главу 11)
  */
  public static void main(String args[]) {
    System.out.println("Мне нравится Java!");
  } //Конец метода main().
}
```

Комментарий — это специальный раздел в исходном коде, предназначенный для того, чтобы облегчить человеку понимание программы, и считающийся частью документации программы.

В языке Java существуют комментарии трех видов.

- **Традиционные комментарии.** Первые шесть строк в листинге 3.6 представляют собой один традиционный комментарий. Комментарий должен начинаться символами `/*` и заканчиваться символами `*/`. Вся информация между этими символами предназначена только для человека, компилятор игнорирует ее. Часто комментарии такого типа называются *многострочными* (в отличие от однострочных, о которых говорится далее).

  Компиляторы рассматривались в главе 2.

  Строки 2–4 листинга 3.6 начинаются с дополнительных звездочек. Они не обязательные, их принято добавлять для украшения комментария. Лично я не вижу, стал ли комментарий красивее, но, раз все программисты на Java так делают, то лучше не быть белой вороной.

- **Однострочные комментарии.** Пример однострочного комментария — текст `//Конец метода main()` в листинге 3.6. В программе на Java все содержимое строки после символов `//` является комментарием, т.е. компилятор игнорирует этот текст.

- **Комментарии Javadoc.** Начинается с косой черты и двух звездочек (`/**`). В листинге 3.6 есть два комментария Javadoc — над заголовком класса `Displayer` и в теле этого класса.

  Комментарий Javadoc — это специальная разновидность традиционного комментария, но предназначенного для чтения людьми, которые не увидят код Java. "Что за чепуха?" — скажете вы. — "Как можно читать комментарий листинга 3.6, не видя его?"

05ch03.indd 64 19.03.2012 11:10:08

Тем не менее можно. Специальная утилита `javadoc.exe` преобразует комментарии Javadoc в стандартную документацию класса. На рис. 3.8 показана веб-страница, автоматически сгенерированная на основе листинга 3.6. Таким способом задокументированы все библиотечные классы, что легко можно увидеть, читая справочник по классам Java.

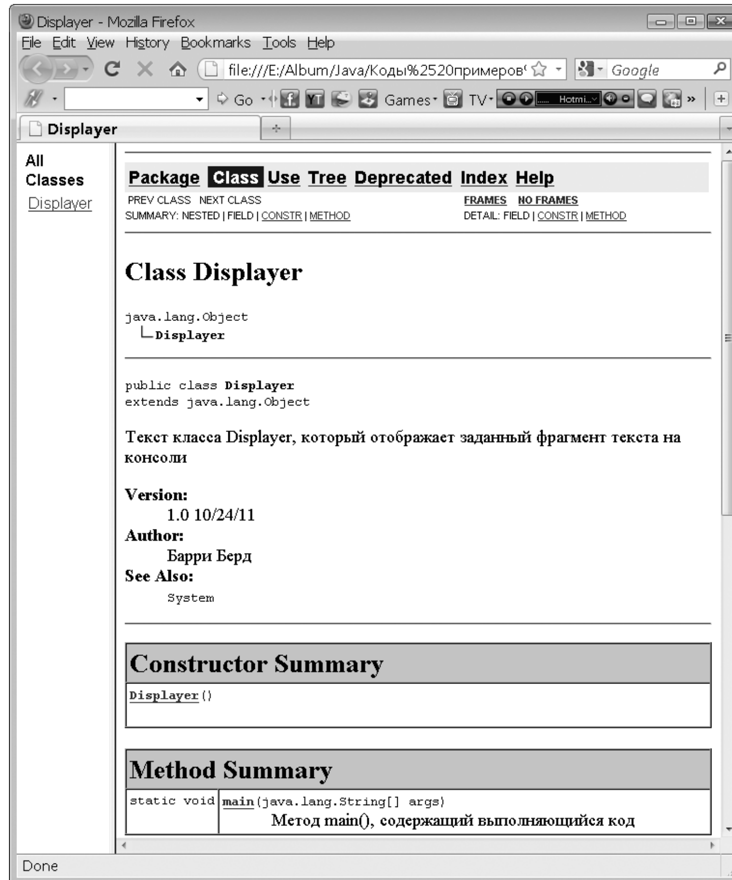

Рис. 3.8. Документация класса, приведенного в листинге 3.6; сгенерирована автоматически на основе комментариев Javadoc

Комментарии Javadoc очень полезны по следующим причинам.

- ✓ Часто единственный человек, читавший код Java, — это программист, писавший этот код. Все остальные применяют этот код, а для этого им нужно знать, что он делает. Как он работает, их не интересует. В комментариях Javadoc подробно описано, что делает данный код и как его следует использовать.
- ✓ Другие люди не смотрят в ваш код и не вносят в него изменений. Поэтому им лучше вообще не иметь к нему доступа, чтобы не внести в него ошибок.

- ✓ Некоторые люди могут не анализировать подробности работы кода. Все, что им нужно, есть в справочнике, автоматически сгенерированном утилитой `javadoc.exe`.
- ✓ Программист не создает две отдельные сущности: код Java — в одном месте и документацию к нему — в другом. Вместо этого он в одном и том же месте создает код и добавляет в него комментарии Javadoc. Он лучше всего знает код непосредственно во время его создания.
- ✓ Генерация веб-страницы на основе комментариев Javadoc выполняется автоматически. Поэтому вся документация продуцируется в одном и том же формате. Независимо от того, кто создавал код и каковы его личные предпочтения, вы увидите документацию в формате, аналогичном показанному на рис. 3.8. Этот формат документации хорошо знаком всем разработчикам приложений Java.

Чтобы сгенерировать веб-страницу документации на основе комментариев Javadoc, выберите в меню Eclipse команду Project⇨Generate Javadoc (Проект⇨Сгенерировать Javadoc). В открывшемся окне щелкните на кнопке Configure (Сконфигурировать). Найдите и выберите файл `javadoc.exe` (обычно он находится в каталоге `jdk` или `jre`). Задайте место расположения документации (по умолчанию предлагается каталог `doc`) и щелкните на кнопке Finish (Готово).

Не будьте слишком строги к старине Барри

В течение многих лет я учу студентов добавлять комментарии в код и убеждаю их не лениться делать это. В то же время многие годы я создаю учебные примеры кодов (как в листинге 3.1) без комментариев. Почему?

Три волшебных слова: “Знайте свою аудиторию”. Когда вы пишете сложный код для решения реальных задач, ваша аудитория — программисты, менеджеры компаний, разрабатывающих программное обеспечение, и люди, которым зачем-то нужно понять, что и как вы сделали. Когда же я пишу примеры для книги, моя аудитория — это вы, начинающие программисты. Вместо того чтобы читать комментарии, вам лучше внимательно смотреть на инструкции Java. Более подробным комментарием служит текст книги.

Кроме того, я и сам немного ленив.

Использование комментариев для экспериментов с кодом

Если вы присутствовали при разговоре программистов, значит, вам довольно часто приходилось слышать специфические словечки *закомментировать* и *раскомментировать*. Когда вы пишете программу и что-то работает не так, как нужно, часто бывает полезно для эксперимента удалить часть кода, а затем вернуть его на место. Это поможет сравнить, как код работает с данным фрагментом и без него. Конечно, вам может не понравиться то, что код делает без фрагмента, поэтому удалять фрагмент кода навсегда нежелательно. Нужно иметь возможность вернуть его на место точно в том же виде, в каком он был до удаления. Проще всего это сделать, добавив две косые черты (`//`). Этим вы закомментируете код строки, т.е. превратите его в комментарий. Удалив две косые черты, вы раскомментируете код.

05ch03.indd 66 19.03.2012 11:10:08

Например, инструкция

```
System.out.println("Мне нравится Java!");
```

превращается в комментарий

```
// System.out.println("Мне нравится Java!");
```

Компилятор Java игнорирует комментарий, поэтому закомментированный код ведет себя так, как будто его нет. В то же время его легко вернуть на место.

Традиционные (многострочные) комментарии не очень полезны для этой цели. Главная проблема состоит в том, что один многострочный комментарий может попасть внутрь другого. Предположим, вы хотите закомментировать следующие инструкции.

```
System.out.println("Родители!");
System.out.println("Ссорьтесь");
/*
 * Фраза умышленно размещена в четырех строках
 */
System.out.println("очень");
System.out.println("осторожно!");
```

Если вы окружите данный фрагмент символами многострочных комментариев, то получите следующий код.

```
/*
System.out.println("Родители!");
System.out.println("Ссорьтесь");
/*
 * Фраза умышленно размещена в четырех строках
 */
System.out.println("очень");
System.out.println("осторожно!");
*/
```

Первое вхождение символов `*/` завершает многострочный комментарий. В результате две последние инструкции оказываются не закомментированными, и, что хуже всего, символы `*/` сбивают с толку компилятор. Он не может понять, что нужно делать, и останавливается, сгенерировав сообщение об ошибке. Следовательно, чтобы закомментировать данный фрагмент, нужно раскомментировать внутренний фрагмент, а это настолько неудобно (ведь его нужно будет найти и опять закомментировать), что лучше отказаться от данного приема.

Большинство современных интегрированных сред разработки (в том числе Eclipse) предоставляют средства автоматического комментирования крупных блоков кода. Здесь я не описываю их только для того, чтобы не делать книгу слишком толстой.

05ch03.indd 67 19.03.2012 11:10:08

05ch03.indd 68 19.03.2012 11:10:08