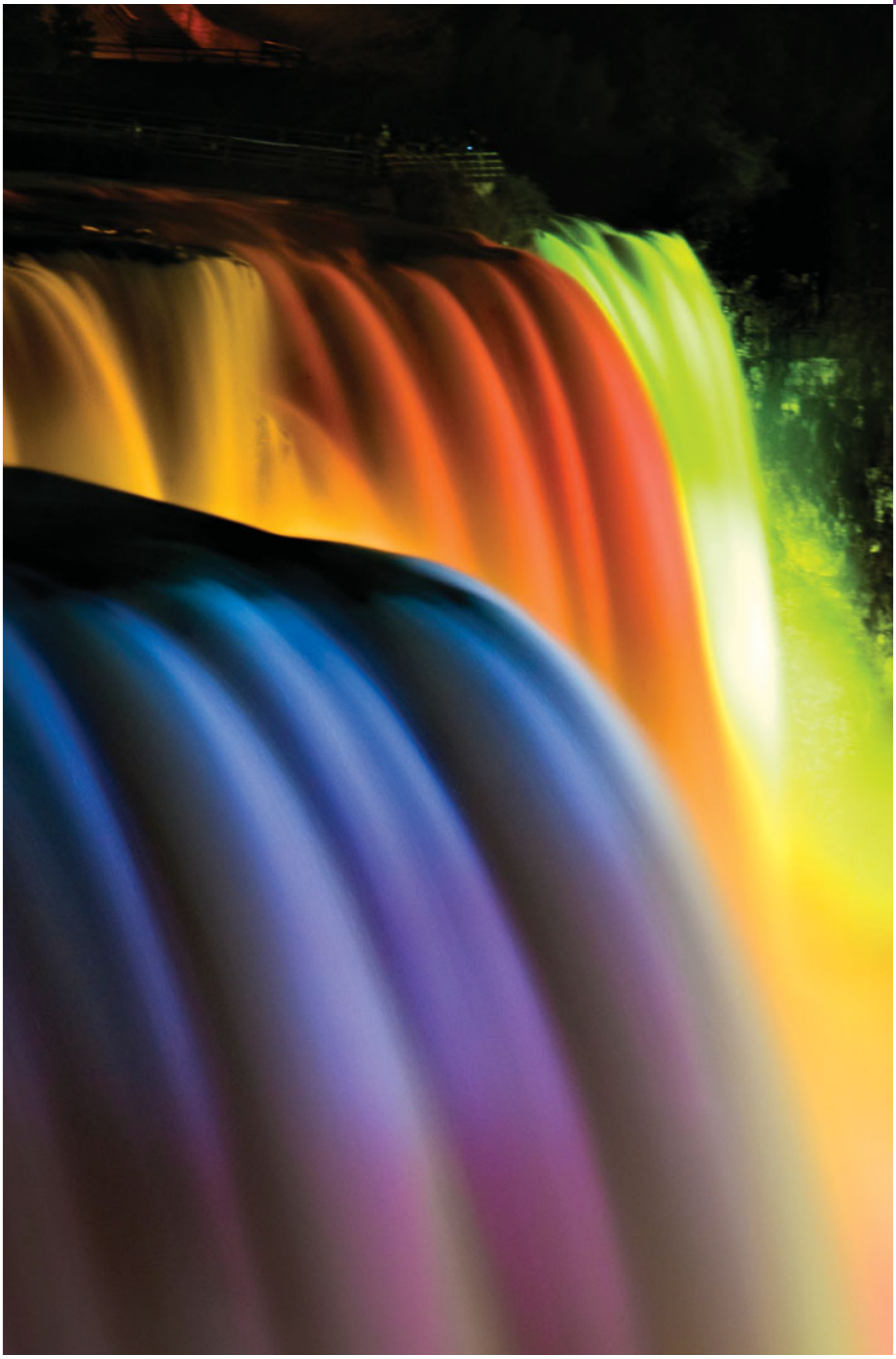
Introduction to Computers and Java

1



Man is still the most extraordinary computer of all.
—John F. Kennedy

Good design is good business.
—Thomas J. Watson, Founder of IBM

How wonderful it is that nobody need wait a single moment before starting to improve the world.
—Anne Frank

Objectives

In this chapter you'll learn:

- Exciting recent developments in the computer field.
- Computer hardware, software and networking basics.
- The data hierarchy.
- The different types of programming languages.
- Basic object-technology concepts.
- The importance of the Internet and the web.
- A typical Java program-development environment.
- To test-drive a Java application.
- Some key recent software technologies.
- How computers can help you make a difference.

Outline

1.1 Introduction

Welcome to Java—the world's most widely used computer programming language. You're already familiar with the powerful tasks computers perform. Using this textbook, you'll write instructions commanding computers to perform those kinds of tasks. *Software* (i.e., the instructions you write) controls *hardware* (i.e., computers).

You'll learn *object-oriented programming*—today's key programming methodology. You'll create and work with many *software objects* in this text.

Java is the preferred language for meeting many organizations' enterprise programming needs. Java has also become the language of choice for implementing Internet-based applications and software for devices that communicate over a network.

In use today are more than a billion general-purpose computers and billions more Java-enabled cell phones, smartphones and handheld devices (such as tablet computers). According to a study by eMarketer, the number of mobile Internet users will reach approximately 134 million by 2013.[1] Other studies have projected smartphone sales to surpass personal computer sales in 2011[2] and tablet sales to account for over 20% of all personal computer sales by 2015.[3] By 2014, the smartphone applications market is expected to exceed $40 billion,[4] which is creating significant opportunities for programming mobile applications.

Java Editions: SE, EE and ME

Java has evolved so rapidly that this ninth edition of *Java How to Program*—based on **Java Standard Edition 6 (Java SE 6)** with optional modules on the new features of **Java SE 7**—was published just 14 years after the first edition. Java is used in such a broad spectrum of applications that it has two other editions. The **Java Enterprise Edition (Java EE)** is geared toward developing large-scale, distributed networking applications and web-based applications. In the past, most computer applications ran on "standalone" computers (comput-

[1] www.circleid.com/posts/mobile_internet_users_to_reach_134_million_by_2013/.

[2] www.pcworld.com/article/171380/more_smartphones_than_desktop_pcs_by_2011.html.

[3] www.forrester.com/ER/Press/Release/0,1769,1340,00.html.

[4] *Inc.*, December 2010/January 2011, pages 116–123.

ers that were not networked together). Today's applications can be written with the aim of communicating among the world's computers via the Internet and the web. Later in this book we discuss how to build such web-based applications with Java.

The **Java Micro Edition (Java ME)** is geared toward developing applications for small, memory-constrained devices, such as BlackBerry smartphones. Google's Android operating system—used on numerous smartphones, tablets (small, lightweight mobile computers with touch screens), e-readers and other devices—uses a customized version of Java not based on Java ME.

Computing in Industry and Research

These are exciting times in the computer field. Many of the most influential and successful businesses of the last two decades are technology companies, including Apple, IBM, Hewlett Packard, Dell, Intel, Motorola, Cisco, Microsoft, Google, Amazon, Facebook, Twitter, Groupon, Foursquare, Yahoo!, eBay and many more—these are major employers of people who study computer science, information systems or related disciplines. At the time of this writing, Apple was the second most valuable company in the world and *the* most valuable technology company.[5] Computers are also used extensively in academic and industrial research. Figure 1.1 provides just a few examples of exciting ways in which computers are used in research and industry.

Name	Description
Internet	The Internet—a global network of computers—was made possible by the *convergence of computing and communications*. It has its roots in the 1960s, when research funding was supplied by the U.S. Department of Defense. Originally designed to connect the main computer systems of about a dozen universities and research organizations, the Internet today is accessible by billions of computers and computer-controlled devices worldwide. Computers break lengthy transmissions into packets at the sending end, route the packets to their intended receivers and ensure that those packets are received in sequence and without error at the receiving end. According to a study by Forrester Research, the average U.S. online consumer now spends as much time online as watching television (`forrester.com/rb/Research/understanding_changing_needs_of_us_online_consumer/q/id/57861/t/2`).
Human Genome Project	The Human Genome Project was founded to identify and analyze the 20,000+ genes in human DNA . The project used computer programs to analyze complex genetic data, determine the sequences of the billions of chemical base pairs that make up human DNA and store the information in databases which have been made available to researchers in many fields. This research has led to tremendous innovation and growth in the biotechnology industry.

Fig. 1.1 | A few uses for computers. (Part 1 of 3.)

[5] `www.zdnet.com/blog/apple/apple-becomes-worlds-second-most-valuable-company/9047`.

Name	Description
World Community Grid	World Community Grid (`www.worldcommunitygrid.org`) is a nonprofit computing grid. People worldwide donate their unused computer processing power by installing a free secure software program that allows the World Community Grid to harness the excess power when the computers are idle. The computing power is used in place of supercomputers to conduct scientific research projects that are making a difference, including developing affordable solar energy, providing clean water to the developing world, fighting cancer, curing muscular dystrophy, finding influenza antiviral drugs, growing more nutritious rice for regions fighting hunger and more.
Medical imaging	X-ray computed tomography (CT) scans, also called CAT (computerized axial tomography) scans, take X-rays of the body from hundreds of different angles. Computers are used to adjust the intensity of the X-ray, optimizing the scan for each type of tissue, then to combine all of the information to create a 3D image.
GPS	Global Positioning System (GPS) devices use a network of satellites to retrieve location-based information. Multiple satellites send time-stamped signals to the device GPS device, which calculates the distance to each satellite based on the time the signal left the satellite and the time the signal was received. The location of each satellite and the distance to each are used to determine the exact location of the device. Based on your location, GPS devices can provide step-by-step directions, help you easily find nearby businesses (restaurants, gas stations, etc.) and points of interest, or help you find your friends.
Microsoft's SYNC®	Many Ford cars now feature Microsoft's SYNC technology, providing speech-synthesis (for reading text messages to you) and speech-recognition capabilities that allow you to use voice commands to browse music, request traffic alerts and more.
AMBER™ Alert	The AMBER (America's Missing: Broadcast Emergency Response) Alert System is used to find abducted children. Law enforcement notifies TV and radio broadcasters and state transportation officials, who then broadcast alerts on TV, radio, computerized highway signs, the Internet and wireless devices. AMBER Alert recently partnered with Facebook. Facebook users can "Like" AMBER Alert pages by location to receive alerts in their news feeds.
Robots	Robots are computerized machines that can perform tasks (including physical tasks), respond to stimuli and more. They can be used for day-to-day tasks (e.g., iRobot's Roomba vacuum), entertainment (such as robotic pets), military combat, space and deep sea exploration, manufacturing and more. In 2004, NASA's remote-controlled Mars rover—which used Java technology—explored the surface to learn about the history of water on the planet.

Fig. 1.1 | A few uses for computers. (Part 2 of 3.)

Name	Description
One Laptop Per Child (OLPC)	One Laptop Per Child (OLPC) is providing low-power, inexpensive, Internet-enabled laptops to poor children worldwide—enabling learning and reducing the digital divide (`one.laptop.org`). By providing these educational resources, OLPC is increasing the opportunities for poor children to learn and make a difference in their communities.
Game programming	The computer game business is larger than the first-run movie business. The most sophisticated video games can cost as much as $100 million to develop. Activision's *Call of Duty 2: Modern Warfare*, released in November 2009, earned $310 million in just one day in North America and the U.K. (`news.cnet.com/8301-13772_3-10396593-52.html?tag=mncol;txt`)! Online *social gaming*, which enables users worldwide to compete with one another, is growing rapidly. Zynga—creator of popular online games such as *Farmville* and *Mafia Wars*—was founded in 2007 and already has over 215 million monthly users. To accommodate the growth in traffic, Zynga is adding nearly 1,000 servers each week (`techcrunch.com/2010/09/22/zynga-moves-1-petabyte-of-data-daily-adds-1000-servers-a-week/`)! Video game consoles are also becoming increasingly sophisticated. The Wii Remote uses an *accelerometer* (to detect tilt and acceleration) and a sensor that determines where the device is pointing, allowing the device to respond to motion. By gesturing with the Wii Remote in hand, you can control the video game on the screen. With Microsoft's Kinect for Xbox 360, you—the player—become the controller. Kinect uses a camera, depth sensor and sophisticated software to follow your body movement, allowing you to control the game (`en.wikipedia.org/wiki/Kinect`). Kinect games include dancing, exercising, playing sports, training virtual animals and more.
Internet TV	Internet TV set-top boxes (such as Apple TV and Google TV) give you access to content—such as games, news, movies, television shows and more—allowing you to access an enormous amount of content on demand; you no longer need to rely on cable or satellite television providers to get content.

Fig. 1.1 | A few uses for computers. (Part 3 of 3.)

1.2 Computers: Hardware and Software

A computer is a device that can perform computations and make logical decisions phenomenally faster than human beings can. Many of today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime. *Supercomputers* are already performing *thousands of trillions (quadrillions)* of instructions per second! To put that in perspective, a quadrillion-instruction-per-second computer can perform in one second more than 100,000 calculations *for every person on the planet!* And—these "upper limits" are growing quickly!

Computers process data under the control of sets of instructions called **computer programs**. These programs guide the computer through orderly sets of actions specified by people called computer **programmers**. The programs that run on a computer are referred to as **software**. In this book, you'll learn today's key programming methodology that's enhancing programmer productivity, thereby reducing software-development costs—*object-oriented programming*.

A computer consists of various devices referred to as **hardware** (e.g., the keyboard, screen, mouse, hard disks, memory, DVDs and processing units). Computing costs are *dropping dramatically*, owing to rapid developments in hardware and software technologies. Computers that might have filled large rooms and cost millions of dollars decades ago are now inscribed on silicon chips smaller than a fingernail, costing perhaps a few dollars each. Ironically, silicon is one of the most abundant materials—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that more than a billion general-purpose computers are in use worldwide, and this is expected to *double* in the next few years.

Computer chips (*microprocessors*) control countless devices. These **embedded systems** include anti-lock brakes in cars, navigation systems, smart home appliances, home security systems, cell phones and smartphones, robots, intelligent traffic intersections, collision avoidance systems, video game controllers and more. The vast majority of the microprocessors produced each year are embedded in devices other than general-purpose computers.[6]

Moore's Law

Every year, you probably expect to pay at least a little more for most products and services. The opposite has been the case in the computer and communications fields, especially with regard to the costs of hardware supporting these technologies. For many decades, hardware costs have fallen rapidly. Every year or two, the capacities of computers have approximately *doubled* without any increase in price. This remarkable observation often is called **Moore's Law**, named for the person who identified the trend, Gordon Moore, cofounder of Intel—a leading manufacturer of the processors in today's computers and embedded systems. Moore's Law and related observations are especially true in relation to the amount of memory that computers have for programs, the amount of secondary storage (such as disk storage) they have to hold programs and data over longer periods of time, and their processor speeds—the speeds at which computers execute their programs (i.e., do their work). Similar growth has occurred in the communications field, in which costs have plummeted as enormous demand for communications bandwidth (i.e., information-carrying capacity) has attracted intense competition. We know of no other fields in which technology improves so quickly and costs fall so rapidly. Such phenomenal improvement is truly fostering the *Information Revolution*.

1.3 Data Hierarchy

Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from bits to characters to fields, and so on. Figure 1.2 illustrates a portion of the data hierarchy. Figure 1.3 summarizes the data hierarchy's levels.

[6] `www.eetimes.com/electronics-blogs/industrial-control-designline-blog/4027479/Real-men-program-in-C?pageNumber=1`.

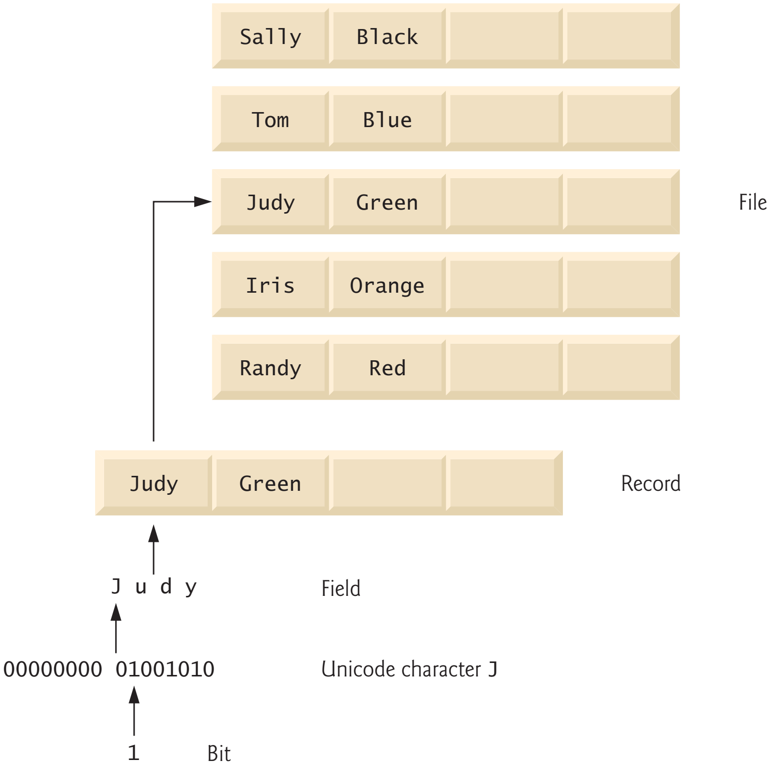


Fig. 1.2 | Data hierarchy.

Level	Description
Bits	The smallest data item in a computer can assume the value 0 or the value 1. Such a data item is called a **bit** (short for "binary digit"—a digit that can assume one of two values). It's remarkable that the impressive functions performed by computers involve only the simplest manipulations of 0s and 1s—*examining a bit's value*, *setting a bit's value* and *reversing a bit's value* (from 1 to 0 or from 0 to 1).
Characters	It's tedious for people to work with data in the low-level form of bits. Instead, they prefer to work with *decimal digits* (0–9), *letters* (A–Z and a–z), and *special symbols* (e.g., $, @, %, &, *, ( ), –, +, ", :, ? and /). Digits, letters and special symbols are known as **characters**. The computer's **character set** is the set of all the characters used to write programs and represent data items. Computers process only 1s and 0s, so a computer's character set represents every character as a pattern of 1s and 0s. Java uses **Unicode**® characters that are composed of two **bytes**, each composed of eight bits. Unicode contains characters for many of the world's languages. See Appendix L for more information on Unicode. See Appendix B for more information on the **ASCII (American Standard Code for Information Interchange)** character set—the popular subset of Unicode that represents uppercase and lowercase letters, digits and some common special characters.

Fig. 1.3 | Levels of the data hierarchy. (Part 1 of 2.)

Level	Description
Fields	Just as characters are composed of bits, **fields** are composed of characters or bytes. A field is a group of characters or bytes that conveys meaning. For example, a field consisting of uppercase and lowercase letters can be used to represent a person's name, and a field consisting of decimal digits could represent a person's age.
Records	Several related fields can be used to compose a **record** (implemented as a `class` in Java). In a payroll system, for example, the record for an employee might consist of the following fields (possible types for these fields are shown in parentheses): • Employee identification number (a whole number) • Name (a string of characters) • Address (a string of characters) • Hourly pay rate (a number with a decimal point) • Year-to-date earnings (a number with a decimal point) • Amount of taxes withheld (a number with a decimal point) Thus, a record is a group of related fields. In the preceding example, all the fields belong to the same employee. A company might have many employees and a payroll record for each one.
Files	A **file** is a group of related records. [*Note:* More generally, a file contains arbitrary data in arbitrary formats. In some operating systems, a file is viewed simply as a *sequence of bytes*—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer.] It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.

Fig. 1.3 | Levels of the data hierarchy. (Part 2 of 2.)

1.4 Computer Organization

Regardless of differences in physical appearance, computers can be envisioned as divided into various **logical units** or sections (Fig. 1.4).

Logical unit	Description
Input unit	This "receiving" section obtains information (data and computer programs) from **input devices** and places it at the disposal of the other units for processing. Most information is entered into computers through keyboards, touch screens and mouse devices. Other forms of input include speaking to your computer, scanning images and barcodes, reading from secondary storage devices (like hard drives, DVD drives, Blu-ray Disc™ drives and USB flash drives—also called "thumb drives" or "memory sticks"), receiving video from a webcam and having your computer receive information from the Internet (such as when you download videos from YouTube™ or e-books from Amazon). Newer forms of input include reading position data from a GPS device, and motion and orientation information from an accelerometer in a smartphone or game controller.

Fig. 1.4 | Logical units of a computer. (Part 1 of 2.)

Logical unit	Description
Output unit	This "shipping" section takes information that the computer has processed and places it on various **output devices** to make it available for use outside the computer. Most information that's output from computers today is displayed on screens, printed on paper, played as audio or video on portable media players (such as Apple's popular iPods) and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as robots and "intelligent" appliances.
Memory unit	This rapid-access, relatively low-capacity "warehouse" section retains information that has been entered through the input unit, making it immediately available for processing when needed. The memory unit also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is *volatile*—it's typically lost when the computer's power is turned off. The memory unit is often called either **memory** or **primary memory**. Typical main memories on desktop and notebook computers contain between 1 GB and 8 GB (GB stands for gigabytes; a gigabyte is approximately one billion bytes).
Arithmetic and logic unit (ALU)	This "manufacturing" section performs *calculations*, such as addition, subtraction, multiplication and division. It also contains the *decision* mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they're equal. In today's systems, the ALU is usually implemented as part of the next logical unit, the CPU.
Central processing unit (CPU)	This "administrative" section coordinates and supervises the operation of the other sections. The CPU tells the input unit when information should be read into the memory unit, tells the ALU when information from the memory unit should be used in calculations and tells the output unit when to send information from the memory unit to certain output devices. Many of today's computers have multiple CPUs and, hence, can perform many operations simultaneously. A **multi-core processor** implements multiple processors on a single integrated-circuit chip—a *dual-core processor* has two CPUs and a *quad-core processor* has four CPUs. Today's desktop computers have processors that can execute billions of instructions per second.
Secondary storage unit	This is the long-term, high-capacity "warehousing" section. Programs or data not actively being used by the other units normally are placed on secondary storage devices (e.g., your *hard drive*) until they're again needed, possibly hours, days, months or even years later. Information on secondary storage devices is *persistent*—it's preserved even when the computer's power is turned off. Secondary storage information takes much longer to access than information in primary memory, but the cost per unit of secondary storage is much less than that of primary memory. Examples of secondary storage devices include CD drives, DVD drives and flash drives, some of which can hold up to 128 GB. Typical hard drives on desktop and notebook computers can hold up to 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes).

Fig. 1.4 | Logical units of a computer. (Part 2 of 2.)

1.5 Machine Languages, Assembly Languages and High-Level Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps. Hundreds of such languages are in use today. These may be divided into three general types:

1. Machine languages
2. Assembly languages
3. High-level languages

Any computer can directly understand only its own **machine language**, defined by its hardware design. Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. Machine languages are *machine dependent* (a particular machine language can be used on only one type of computer). Such languages are cumbersome for humans. For example, here's a section of an early machine-language program that adds overtime pay to base pay and stores the result in gross pay:

```
+1300042774
+1400593419
+1200274027
```

Programming in machine language was simply too slow and tedious for most programmers. Instead of using the strings of numbers that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations. These abbreviations formed the basis of **assembly languages**. *Translator programs* called **assemblers** were developed to convert early assembly-language programs to machine language at computer speeds. The following section of an assembly-language program also adds overtime pay to base pay and stores the result in gross pay:

```
load    basepay
add     overpay
store   grosspay
```

Although such code is clearer to humans, it's incomprehensible to computers until translated to machine language.

Computer usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks. To speed the programming process, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks. Translator programs called **compilers** convert high-level language programs into machine language. High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical notations. A payroll program written in a high-level language might contain a *single* statement such as

```
grossPay = basePay + overTimePay
```

From the programmer's standpoint, high-level languages are preferable to machine and assembly languages. Java is by far the most widely used high-level programming language.

Compiling a large high-level language program into machine language can take a considerable amount of computer time. *Interpreter* programs were developed to execute high-level language programs directly (without the delay of compilation), although slower than compiled programs run. We'll say more about how interpreters work in Section 1.9, where you'll learn that Java uses a clever performance-tuned mixture of compilation and interpretation to ultimately run programs. Exercises 7.35–7.37 (in the Special Section: Building Your Own Computer) guide you through the process of building an interpreter program.

1.6 Introduction to Object Technology

Building software quickly, correctly and economically remains an elusive goal at a time when demands for new and more powerful software are soaring. *Objects*, or more precisely—as we'll see in Chapter 3—the *classes* objects come from, are essentially *reusable* software components. There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc. Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating). Software developers are discovering that using a modular, object-oriented design and implementation approach can make software-development groups much more productive than was possible with earlier popular techniques like "structured programming"—object-oriented programs are often easier to understand, correct and modify.

The Automobile as an Object

To help you understand objects and their contents, let's begin with a simple analogy. Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal*. What must happen before you can do this? Well, before you can drive a car, someone has to *design* it. A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house. These drawings include the design for an accelerator pedal. The pedal *hides* from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel "hides" the mechanisms that turn the car. This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.

Just as you cannot cook meals in the kitchen of a blueprint, you cannot drive a car's engineering drawings. Before you can drive a car, it must be *built* from the engineering drawings that describe it. A completed car has an *actual* accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must *press* the pedal to accelerate the car.

Methods and Classes

Let's use our car example to introduce some key object-oriented programming concepts. Performing a task in a program requires a **method**. The method houses the program statements that actually perform its tasks. The method hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster. In Java, we create a program unit called a **class** to house the set of methods that perform the class's tasks. For example, a class that represents a bank account might contain one method to *deposit* money to an account, another to *withdraw* money from an account and a third to *inquire* what the account's current balance is. A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

Instantiation

Just as someone has to *build a car* from its engineering drawings before you can actually drive a car, you must *build an object* of a class before a program can perform the tasks that the class's methods define. The process of doing this is called *instantiation*. An object is then referred to as an **instance** of its class.

Reuse

Just as a car's engineering drawings can be *reused* many times to build many cars, you can *reuse* a class many times to build many objects. Reuse of existing classes when building new classes and programs saves time and effort. Reuse also helps you build more reliable and effective systems, because existing classes and components often have gone through extensive *testing*, *debugging* and *performance* tuning. Just as the notion of *interchangeable parts* was crucial to the Industrial Revolution, reusable classes are crucial to the software revolution that has been spurred by object technology.

Software Engineering Observation 1.1

Use a building-block approach to creating your programs. Avoid reinventing the wheel—use existing pieces wherever possible. This software reuse *is a key benefit of object-oriented programming.*

Messages and Methods Calls

When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster. Similarly, you *send messages to an object*. Each message is implemented as a **method call** that tells a method of the object to perform its task. For example, a program might call a particular bank account object's *deposit* method to increase the account's balance.

Attributes and Instance Variables

A car, besides having capabilities to accomplish tasks, also has *attributes*, such as its color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading). Like its capabilities, the car's attributes are represented as part of its design in its engineering diagrams (which, for example, include an odometer and a fuel gauge). As you drive an actual car, these attributes are carried along with the car. Every car maintains its *own* attributes. For example, each car knows how much gas is in its own gas tank, but *not* how much is in the tanks of *other* cars.

An object, similarly, has attributes that it carries along as it's used in a program. These attributes are specified as part of the object's class. For example, a bank account object has a *balance attribute* that represents the amount of money in the account. Each bank account object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank. Attributes are specified by the class's **instance variables**.

Encapsulation

Classes **encapsulate** (i.e., wrap) attributes and methods into objects—an object's attributes and methods are intimately related. Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details are *hidden* within the objects themselves. This **information hiding**, as we'll see, is crucial to good software engineering.

Inheritance

A new class of objects can be created quickly and conveniently by **inheritance**—the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own. In our car analogy, an object of class "convertible" certainly *is an* object of the more *general* class "automobile," but more *specifically*, the roof can be raised or lowered.

Object-Oriented Analysis and Design (OOAD)

Soon you'll be writing programs in Java. How will you create the **code** (i.e., the program instructions) for your programs? Perhaps, like many programmers, you'll simply turn on your computer and start typing. This approach may work for small programs (like the ones we present in the early chapters of the book), but what if you were asked to create a software system to control thousands of automated teller machines for a major bank? Or suppose you were asked to work on a team of 1,000 software developers building the next U.S. air traffic control system? For projects so large and complex, you should not simply sit down and start writing programs.

To create the best solutions, you should follow a detailed **analysis** process for determining your project's **requirements** (i.e., defining *what* the system is supposed to do) and developing a **design** that satisfies them (i.e., deciding *how* the system should do it). Ideally, you'd go through this process and carefully review the design (and have your design reviewed by other software professionals) before writing any code. If this process involves analyzing and designing your system from an object-oriented point of view, it's called an **object-oriented analysis and design (OOAD) process**. Languages like Java are object oriented. Programming in such a language, called **object-oriented programming (OOP)**, allows you to implement an object-oriented design as a working system.

The UML (Unified Modeling Language)

Although many different OOAD processes exist, a single graphical language for communicating the results of *any* OOAD process has come into wide use. This language, known as the Unified Modeling Language (UML), is now the most widely used graphical scheme for modeling object-oriented systems. We present our first UML diagrams in Chapters 3 and 4, then use them in our deeper treatment of object-oriented programming through Chapter 11. In our *optional* ATM Software Engineering Case Study in Chapters 12–13 we present a simple subset of the UML's features as we guide you through an object-oriented design experience.

1.7 Operating Systems

Operating systems are software systems that make using computers more convenient for users, application developers and system administrators. Operating systems provide services that allow each application to execute safely, efficiently and *concurrently* (i.e., in parallel) with other applications. The software that contains the core components of the operating system is called the **kernel**. Popular desktop operating systems include Linux, Windows 7 and Mac OS X. Popular mobile operating systems used in smartphones and tablets include Google's Android, BlackBerry OS and Apple's iOS (for its iPhone, iPad and iPod Touch devices).

Windows—A Proprietary Operating System

In the mid-1980s, Microsoft developed the **Windows operating system**, consisting of a graphical user interface built on top of DOS—an enormously popular personal-computer operating system of the time that users interacted with by typing commands. Windows borrowed from many concepts (such as icons, menus and windows) popularized by early Apple Macintosh operating systems and originally developed by Xerox PARC. Windows 7 is Microsoft's latest operating system—its features include enhancements to the user interface, faster startup times, further refinement of security features, touch-screen and multi-touch support, and more. Windows is a *proprietary* operating system—it's controlled by one company exclusively. Windows is by far the world's most widely used operating system.

Linux—An Open-Source Operating System

The Linux operating system is perhaps the greatest success of the *open-source* movement. **Open-source software** is a software development style that departs from the *proprietary* development that dominated software's early years. With open-source development, individuals and companies contribute their efforts in developing, maintaining and evolving software in exchange for the right to use that software for their own purposes, typically at no charge. Open-source code is often scrutinized by a much larger audience than proprietary software, so errors often get removed faster. Open source also encourages more innovation. Sun open sourced its implementation of the Java Development Kit and many of its related Java technologies.

Some organizations in the open-source community are the Eclipse Foundation (the Eclipse Integrated Development Environment helps Java programmers conveniently develop software), the Mozilla Foundation (creators of the Firefox web browser), the Apache Software Foundation (creators of the Apache web server used to develop web-based applications) and SourceForge (which provides the tools for managing open source projects—it has over 260,000 of them under development). Rapid improvements to computing and communications, decreasing costs and open-source software have made it much easier and more economical to create a software-based business now than just a few decades ago. A great example is Facebook, which was launched from a college dorm room and built with open-source software.[7]

The **Linux** kernel is the core of the most popular open-source, freely distributed, full-featured operating system. It's developed by a loosely organized team of volunteers, and is popular in servers, personal computers and embedded systems. Unlike that of proprietary operating systems like Microsoft's Windows and Apple's Mac OS X, Linux source code (the program code) is available to the public for examination and modification and is free to download and install. As a result, users of the operating system benefit from a community of developers actively debugging and improving the kernel, an absence of licensing

[7] `developers.facebook.com/opensource/`.

fees and restrictions, and the ability to completely customize the operating system to meet specific needs.

In 1991, Linus Torvalds, a 21-year-old student at the University of Helsinki, Finland, began developing the Linux kernel as a hobby. (The name Linux is derived from "Linus" and "UNIX"—an operating system developed by Bell Labs in 1969.) Torvalds wished to improve upon the design of Minix, an educational operating system created by Professor Andrew Tanenbaum of the Vrije Universiteit in Amsterdam. The Minix source code was publicly available to allow professors to demonstrate basic operating-system implementation concepts to their students.

Torvalds released the first version of Linux in 1991. The favorable response led to the creation of a community that has continued to develop and support Linux. Developers downloaded, tested, and modified the Linux code, submitting bug fixes and feedback to Torvalds, who reviewed them and applied the improvements to the code.

The 1994 release of Linux included many features commonly found in a mature operating system, making Linux a viable alternative to UNIX. Enterprise systems companies such as IBM and Oracle became increasingly interested in Linux as it continued to stabilize and spread to new platforms.

A variety of issues—such as Microsoft's market power, the small number of user-friendly Linux applications and the diversity of Linux distributions, such as Red Hat Linux, Ubuntu Linux and many others—have prevented widespread Linux use on desktop computers. But Linux has become extremely popular on servers and in embedded systems, such as Google's Android-based smartphones.

Android

Android—the fastest growing mobile and smartphone operating system—is based on the Linux kernel and Java. Experienced Java programmers can quickly dive into Android development. One benefit of developing Android apps is the openness of the platform. The operating system is open source and free.

The Android operating system was developed by Android, Inc., which was acquired by Google in 2005. In 2007, the Open Handset Alliance™—a consortium of 34 companies initially and 79 by 2010—was formed to continue developing Android. As of December 2010, more than 300,000 Android smartphones were being activated each day![8] Android smartphones are now outselling iPhones.[9] The Android operating system is used in numerous smartphones (such as the Motorola Droid, HTC EVO™ 4G, Samsung Vibrant™ and many more), e-reader devices (such as the Barnes and Noble Nook™), tablet computers (such as the Dell Streak, the Samsung Galaxy Tab and more), in-store touch-screen kiosks, cars, robots and multimedia players.

Android smartphones include the functionality of a mobile phone, Internet client (for web browsing and Internet communication), MP3 player, gaming console, digital camera and more, wrapped into handheld devices with full-color *multitouch screens*—these allow you to control the device with *gestures* involving one touch or multiple simultaneous touches. You can download apps directly onto your Android device through Android Market and other app marketplaces. As of December 2010, there were over 200,000 apps in Google's Android Market.

[8] www.pcmag.com/article2/0,2817,2374076,00.asp.

[9] mashable.com/2010/08/02/android-outselling-iphone-2/.

Android App-Development Chapters on the Companion Website

Because of the tremendous interest in Android-based devices and apps, we've included on the book's companion website a three-chapter introduction to Android app development from our new book, *Android for Programmers: An App-Driven Approach*. After you learn Java, you'll find it straightforward to begin developing and running Android apps. You can place your apps on the online Android Market (`www.market.android.com`) and if they're successful, you may even be able to launch a business. Just remember—Facebook, Microsoft and Dell were all launched from dorm rooms.

1.8 Programming Languages

In this section, we provide brief comments on several popular programming languages (Fig. 1.5). In the next section we introduce Java.

Programming language	Description
Fortran	Fortran (FORmula TRANslator) was developed by IBM Corporation in the mid-1950s to be used for scientific and engineering applications that require complex mathematical computations. It's still widely used and its latest versions are object oriented.
COBOL	COBOL (COmmon Business Oriented Language) was developed in the late 1950s by computer manufacturers, the U.S. government and industrial computer users based on a language developed by Grace Hopper, a career U.S. Navy officer and computer scientist. COBOL is still widely used for commercial applications that require precise and efficient manipulation of large amounts of data. Its latest version supports object-oriented programming.
Pascal	Research in the 1960s resulted in *structured programming*—a disciplined approach to writing programs that are clearer, easier to test and debug and easier to modify than large programs produced with previous techniques. One of the more tangible results of this research was the development of Pascal by Professor Niklaus Wirth in 1971. It was designed for teaching structured programming and was popular in college courses for several decades.
Ada	Ada, based on Pascal, was developed under the sponsorship of the U.S. Department of Defense (DOD) during the 1970s and early 1980s. The DOD wanted a single language that would fill most of its needs. The Pascal-based language was named after Lady Ada Lovelace, daughter of the poet Lord Byron. She's credited with writing the world's first computer program in the early 1800s (for the Analytical Engine mechanical computing device designed by Charles Babbage). Its latest version supports object-oriented programming.
Basic	Basic was developed in the 1960s at Dartmouth College to introduce novices to programming. Many of its latest versions are object oriented.
C	C was implemented in 1972 by Dennis Ritchie at Bell Laboratories. It initially became widely known as the UNIX operating system's development language. Today, most of the code for general-purpose operating systems is written in C or C++.

Fig. 1.5 | Other programming languages. (Part 1 of 2.)

Programming language	Description
C++	C++, an extension of C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ provides a number of features that "spruce up" the C language, but more important, it provides capabilities for object-oriented programming.
Objective-C	Objective-C is an object-oriented language based on C. It was developed in the early 1980s and later acquired by Next, which in turn was acquired by Apple. It has become the key programming language for the Mac OS X operating system and all iOS-powered devices (such as iPods, iPhones and iPads).
Visual Basic	Microsoft's Visual Basic language was introduced in the early 1990s to simplify the development of Microsoft Windows applications. Its latest versions support object-oriented programming.
Visual C#	Microsoft's three primary object-oriented programming languages are Visual Basic, Visual C++ (based on C++) and C# (based on C++ and Java, and developed for integrating the Internet and the web into computer applications).
PHP	PHP is an object-oriented, "open-source" (see Section 1.7) "scripting" language supported by a community of users and developers and is used by numerous websites including Wikipedia and Facebook. PHP is platform independent—implementations exist for all major UNIX, Linux, Mac and Windows operating systems. PHP also supports many databases, including MySQL.
Python	Python, another object-oriented scripting language, was released publicly in 1991. Developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam (CWI), Python draws heavily from Modula-3—a systems programming language. Python is "extensible"—it can be extended through classes and programming interfaces.
JavaScript	JavaScript is the most widely used scripting language. It's primarily used to add programmability to web pages—for example, animations and interactivity with the user. It's provided with all major web browsers.
Ruby on Rails	Ruby—created in the mid-1990s by Yukihiro Matsumoto—is an open-source, object-oriented programming language with a simple syntax that's similar to Python. Ruby on Rails combines the scripting language Ruby with the Rails web application framework developed by 37Signals. Their book, *Getting Real* (`gettingreal.37signals.com/toc.php`), is a must read for web developers. Many Ruby on Rails developers have reported productivity gains over other languages when developing database-intensive web applications. Ruby on Rails was used to build Twitter's user interface.
Scala	Scala (`www.scala-lang.org/node/273`)—short for "scalable language"—was designed by Martin Odersky, a professor at École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland. Released in 2003, Scala uses both the object-oriented and functional programming paradigms and is designed to integrate with Java. Programming in Scala can reduce the amount of code in your applications significantly. Twitter and Foursquare use Scala.

Fig. 1.5 | Other programming languages. (Part 2 of 2.)

1.9 Java and a Typical Java Development Environment

The microprocessor revolution's most important contribution to date is that it made possible the development of personal computers. Microprocessors are having a profound impact in intelligent consumer-electronic devices. Recognizing this, Sun Microsystems in 1991 funded an internal corporate research project led by James Gosling, which resulted in a C++-based object-oriented programming language Sun called Java.

A key goal of Java is to be able to write programs that will run on a great variety of computer systems and computer-control devices. This is sometimes called "write once, run anywhere."

The web exploded in popularity in 1993, and Sun saw the potential of using Java to add *dynamic content*, such as interactivity and animations, to web pages. Java garnered the attention of the business community because of the phenomenal interest in the web. Java is now used to develop large-scale enterprise applications, to enhance the functionality of web servers (the computers that provide the content we see in our web browsers), to provide applications for consumer devices (e.g., cell phones, smartphones, television set-top boxes and more) and for many other purposes. Sun Microsystems was acquired by Oracle in 2009. At the JavaOne 2010 conference, Oracle announced that 97% of enterprise desktops, three billion handsets, and 80 million television devices run Java. There are currently over 9 million Java developers, up from 4.5 million in 2005.[10] Java is now the most widely used software development language in the world.

Java Class Libraries

You can create each class and method you need to form your Java programs. However, most Java programmers take advantage of the rich collections of existing classes and methods in the **Java class libraries**, which are also known as the **Java APIs (Application Programming Interfaces)**.

Performance Tip 1.1

Using Java API classes and methods instead of writing your own versions can improve program performance, because they're carefully written to perform efficiently. This also shortens program development time.

Portability Tip 1.1

Although it's easier to write portable programs (i.e., programs that can run on many different types of computers) in Java than in most other programming languages, differences between compilers, JVMs and computers can make portability difficult to achieve. Simply writing programs in Java does not *guarantee portability.*

We now explain the commonly used steps in creating and executing a Java application using a Java development environment (illustrated in Figs. 1.6–1.10). Java programs normally go through five phases—edit, compile, load, verify and execute. We discuss these phases in the context of the Java SE Development Kit (JDK). You can download the most up-to-date JDK and its documentation from www.oracle.com/technetwork/java/

[10] jaxenter.com/how-many-java-developers-are-there-10462.html.

javase/downloads/index.html. *Read the Before You Begin section of this book to ensure that you set up your computer properly to compile and execute Java programs.* You may also want to visit Oracle's New to Java Center at:

```
www.oracle.com/technetwork/topics/newtojava/overview/index.html
```

[*Note:* This website provides installation instructions for Windows, Linux and Mac OS X. If you aren't using one of these operating systems, refer to the documentation for your system's Java environment or ask your instructor how to accomplish these tasks based on your computer's operating system. If you encounter a problem with this link or any others referenced in this book, please check www.deitel.com/books/jhtp9/ for errata and please notify us by e-mail at deitel@deitel.com.]

Phase 1: Creating a Program

Phase 1 consists of editing a file with an *editor program*, normally known simply as an *editor* (Fig. 1.6). You type a Java program (typically referred to as **source code**) using the editor, make any necessary corrections and save the program on a secondary storage device, such as your hard drive. A file name ending with the **.java extension** indicates that the file contains Java source code.

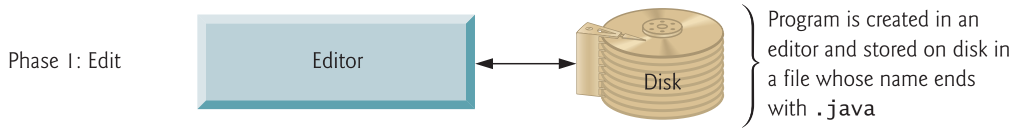

Phase 1: Edit
Editor
Disk
Program is created in an editor and stored on disk in a file whose name ends with .java

Fig. 1.6 | Typical Java development environment—editing phase.

Two editors widely used on Linux systems are vi and emacs. On Windows, Notepad will suffice. Many freeware and shareware editors are also available online, including EditPlus (www.editplus.com), TextPad (www.textpad.com) and jEdit (www.jedit.org).

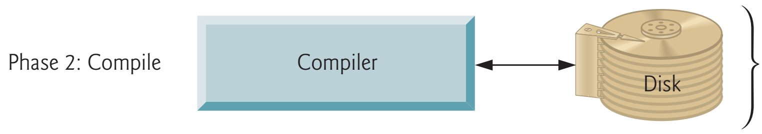
For organizations that develop substantial information systems, **integrated development environments (IDEs)** are available from many major software suppliers. IDEs provide tools that support the software development process, including editors for writing and editing programs and debuggers for locating **logic errors**—errors that cause programs to execute incorrectly. Popular IDEs include Eclipse (www.eclipse.org) and NetBeans (www.netbeans.org).

Phase 2: Compiling a Java Program into Bytecodes

In Phase 2, you use the command **javac** (the **Java compiler**) to **compile** a program (Fig. 1.7). For example, to compile a program called Welcome.java, you'd type

```
javac Welcome.java
```

in the command window of your system (i.e., the **Command Prompt** in Windows, the *shell prompt* in Linux or the Terminal application in Mac OS X). If the program compiles, the compiler produces a **.class** file called Welcome.class that contains the compiled version of the program.



Compiler creates bytecodes and stores them on disk in a file whose name ends with `.class`

Fig. 1.7 | Typical Java development environment—compilation phase.

The Java compiler translates Java source code into **bytecodes** that represent the tasks to execute in the execution phase (Phase 5). Bytecodes are executed by the **Java Virtual Machine** (**JVM**)—a part of the JDK and the foundation of the Java platform. A **virtual machine** (**VM**) is a software application that simulates a computer but hides the underlying operating system and hardware from the programs that interact with it. If the same VM is implemented on many computer platforms, applications that it executes can be used on all those platforms. The JVM is one of the most widely used virtual machines. Microsoft's .NET uses a similar virtual-machine architecture.

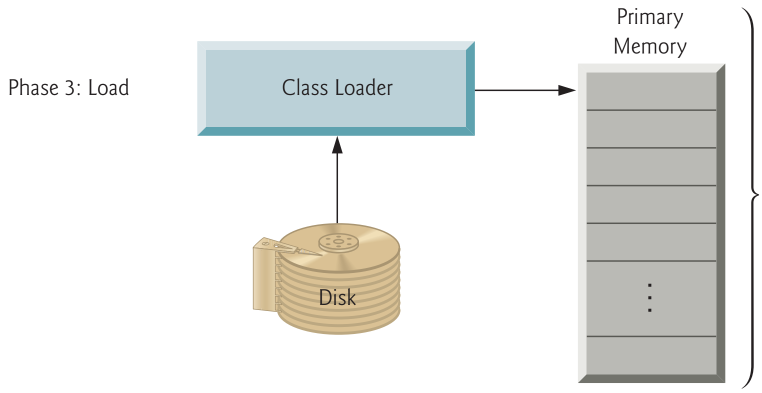
Unlike machine language, which is dependent on specific computer hardware, bytecodes are platform independent—they do not depend on a particular hardware platform. So, Java's bytecodes are **portable**—without recompiling the source code, the same bytecodes can execute on any platform containing a JVM that understands the version of Java in which the bytecodes were compiled. The JVM is invoked by the **`java`** command. For example, to execute a Java application called `Welcome`, you'd type the command

```
    java Welcome
```

in a command window to invoke the JVM, which would then initiate the steps necessary to execute the application. This begins Phase 3.

Phase 3: Loading a Program into Memory

In Phase 3, the JVM places the program in memory to execute it—this is known as **loading** (Fig. 1.8).The JVM's **class loader** takes the `.class` files containing the program's bytecodes and transfers them to primary memory. The class loader also loads any of the `.class` files provided by Java that your program uses. The `.class` files can be loaded from a disk on your system or over a network (e.g., your local college or company network, or the Internet).



Class loader reads `.class` files containing bytecodes from disk and puts those bytecodes in memory

Fig. 1.8 | Typical Java development environment—loading phase.

Phase 4: Bytecode Verification

In Phase 4, as the classes are loaded, the **bytecode verifier** examines their bytecodes to ensure that they're valid and do not violate Java's security restrictions (Fig. 1.9). Java enforces strong security to make sure that Java programs arriving over the network do not damage your files or your system (as computer viruses and worms might).

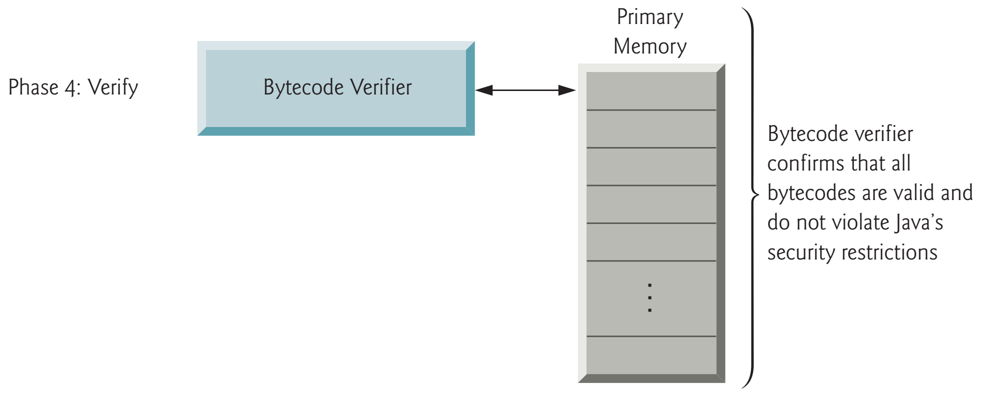

Phase 4: Verify
Bytecode Verifier
Primary Memory
⋮
Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions

Fig. 1.9 | Typical Java development environment—verification phase.

Phase 5: Execution

In Phase 5, the JVM **executes** the program's bytecodes, thus performing the actions specified by the program (Fig. 1.10). In early Java versions, the JVM was simply an interpreter for Java bytecodes. This caused most Java programs to execute slowly, because the JVM would interpret and execute one bytecode at a time. Some modern computer architectures can execute several instructions in parallel. Today's JVMs typically execute bytecodes using a combination of interpretation and so-called **just-in-time (JIT) compilation**. In this process, the JVM analyzes the bytecodes as they're interpreted, searching for *hot spots*—parts of the bytecodes that execute frequently. For these parts, a **just-in-time** (JIT) **compiler**—known as the **Java HotSpot compiler**—translates the bytecodes into the underlying computer's machine language. When the JVM encounters these compiled parts again,

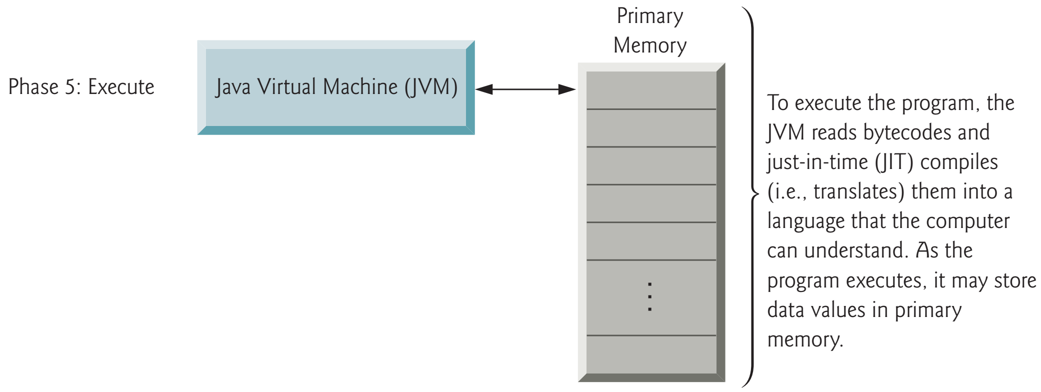

Phase 5: Execute
Java Virtual Machine (JVM)
Primary Memory
⋮
To execute the program, the JVM reads bytecodes and just-in-time (JIT) compiles (i.e., translates) them into a language that the computer can understand. As the program executes, it may store data values in primary memory.

Fig. 1.10 | Typical Java development environment—execution phase.

the faster machine-language code executes. Thus Java programs actually go through *two* compilation phases—one in which source code is translated into bytecodes (for portability across JVMs on different computer platforms) and a second in which, during execution, the bytecodes are translated into machine language for the actual computer on which the program executes.

Problems That May Occur at Execution Time

Programs might not work on the first try. Each of the preceding phases can fail because of various errors that we'll discuss throughout this book. For example, an executing program might try to divide by zero (an illegal operation for whole-number arithmetic in Java). This would cause the Java program to display an error message. If this occurred, you'd have to return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections fixed the problem(s). [*Note:* Most programs in Java input or output data. When we say that a program displays a message, we normally mean that it displays that message on your computer's screen. Messages and other data may be output to other devices, such as disks and hardcopy printers, or even to a network for transmission to other computers.]

Common Programming Error 1.1

Errors such as division by zero occur as a program runs, so they're called ***runtime errors*** *or* ***execution-time errors****.* ***Fatal runtime errors*** *cause programs to terminate immediately without having successfully performed their jobs.* ***Nonfatal runtime errors*** *allow programs to run to completion, often producing incorrect results.*

1.10 Test-Driving a Java Application

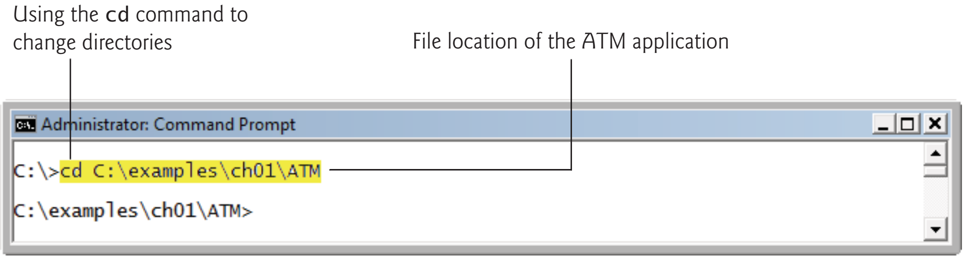
In this section, you'll run and interact with your first Java application. You'll begin by running an ATM application that simulates the transactions that take place when you use an ATM machine (e.g., withdrawing money, making deposits and checking your account balances). You'll learn how to build this application in the *optional*, object-oriented case study included in Chapters 12–13. For the purpose of this section, we assume you're running Microsoft Windows.[11]

In the following steps, you'll run the application and perform various transactions. The elements and functionality you see here are typical of what you'll learn to program in this book. [*Note:* We use fonts to distinguish between features you see on a screen (e.g., the **Command Prompt**) and elements that are not directly related to a screen. Our convention is to emphasize screen features like titles and menus (e.g., the **File** menu) in a semibold **sans-serif Helvetica** font and to emphasize nonscreen elements, such as file names or input (e.g., `ProgramName.java`) in a `sans-serif Lucida` font. As you've already noticed, the defining occurrence of each key term in the text is set in **bold maroon**. In the figures in this section, we highlight in yellow the user input required by each step and point out sig-

[11] At `www.deitel.com/books/jhtp9/`, we provide a Linux version of this test-drive. We also provide links to videos that help you get started with several popular integrated development environments (IDEs), including Java SE Development Kit 6 for Windows, Eclipse SDK for Windows, NetBeans, jGRASP, DrJava, BlueJ and TestPad Text Editor for Windows.

nificant parts of the application. To make these features more visible, we've changed the background color of the **Command Prompt** windows to white and the foreground color to black.] This is a simple text-only version. Later in the book, you'll learn the techniques to rework this using GUI (graphical user interface) techniques.

1. ***Checking your setup.*** Read the Before You Begin section of the book to confirm that you've set up Java properly on your computer and that you've copied the book's examples to your hard drive.
2. ***Locating the completed application.*** Open a **Command Prompt** window. This can be done by selecting **Start > All Programs > Accessories > Command Prompt**. Change to the ATM application directory by typing `cd C:\examples\ch01\ATM`, then press *Enter* (Fig. 1.11). The command `cd` is used to change directories.

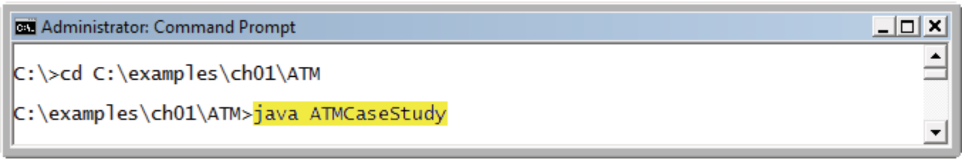


Using the cd command to change directories
File location of the ATM application
Administrator: Command Prompt

```
C:\>cd C:\examples\ch01\ATM

C:\examples\ch01\ATM>
```

Fig. 1.11 | Opening a **Command Prompt** and changing directories.

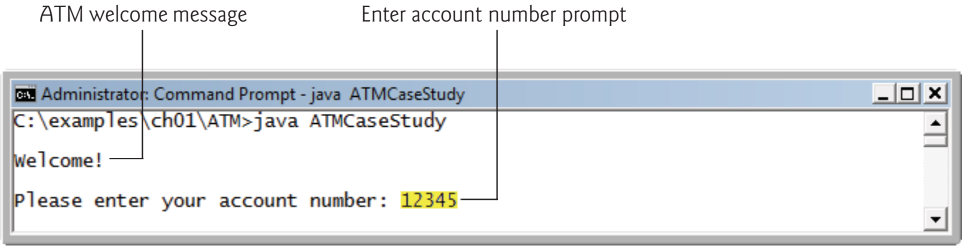
3. ***Running the ATM application.*** Type the command `java ATMCaseStudy` and press *Enter* (Fig. 1.12). Recall that the `java` command, followed by the name of the application's `.class` file (in this case, `ATMCaseStudy`), executes the application. Specifying the `.class` extension when using the `java` command results in an error. [*Note:* Java commands are case sensitive. It's important to type the name of this application with a capital A, T and M in "ATM," a capital C in "Case" and a capital S in "Study." Otherwise, the application will not execute.] If you receive the error message, "`Exception in thread "main" java.lang.NoClassDefFoundError: ATMCaseStudy`," your system has a `CLASSPATH` problem. Please refer to the Before You Begin section of the book for instructions to help you fix this problem.



```
C:\>cd C:\examples\ch01\ATM

C:\examples\ch01\ATM>java ATMCaseStudy
```

Fig. 1.12 | Using the `java` command to execute the ATM application.

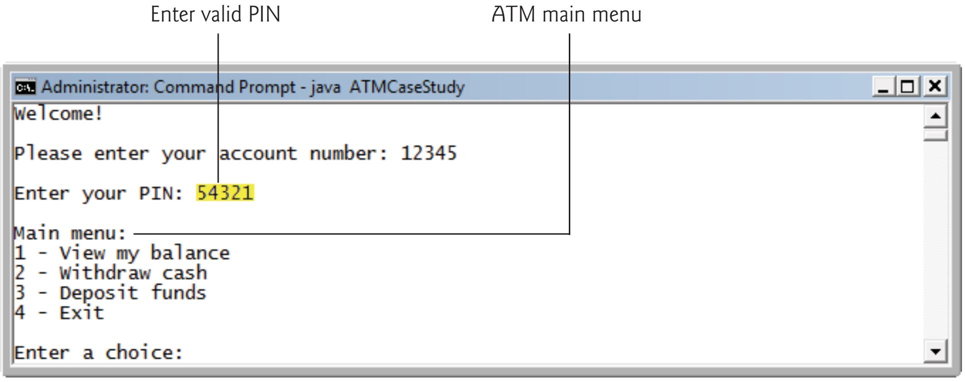
4. ***Entering an account number.*** When the application first executes, it displays a `"Welcome!"` greeting and prompts you for an account number. Type `12345` at the `"Please enter your account number:"` prompt (Fig. 1.13) and press *Enter*.


ATM welcome message
Enter account number prompt
Administrator: Command Prompt - java ATMCaseStudy

```
C:\examples\ch01\ATM>java ATMCaseStudy

Welcome!

Please enter your account number: 12345
```

Fig. 1.13 | Prompting the user for an account number.

5. ***Entering a PIN.*** Once a valid account number is entered, the application displays the prompt `"Enter your PIN:"`. Type `"54321"` as your valid PIN (Personal Identification Number) and press *Enter*. The ATM main menu containing a list of options will be displayed (Fig. 1.14). We'll show how you can enter a PIN privately using a `JPasswordField` in Chapter 14.

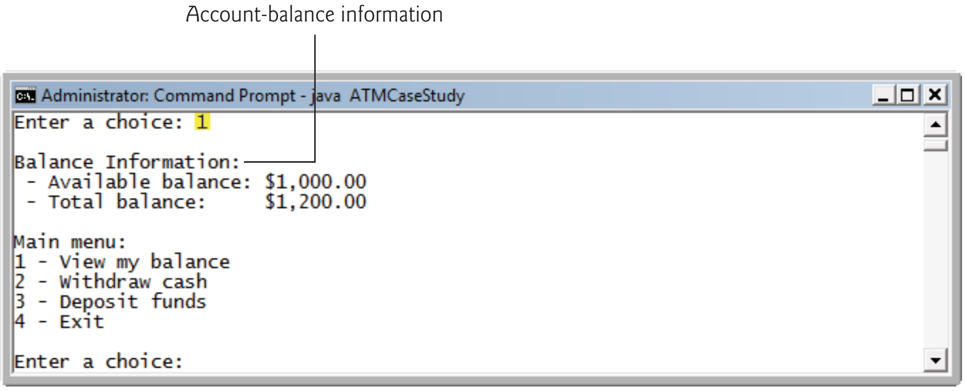


```
Welcome!

Please enter your account number: 12345

Enter your PIN: 54321

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice:
```

Fig. 1.14 | Entering a valid PIN number and displaying the ATM application's main menu.

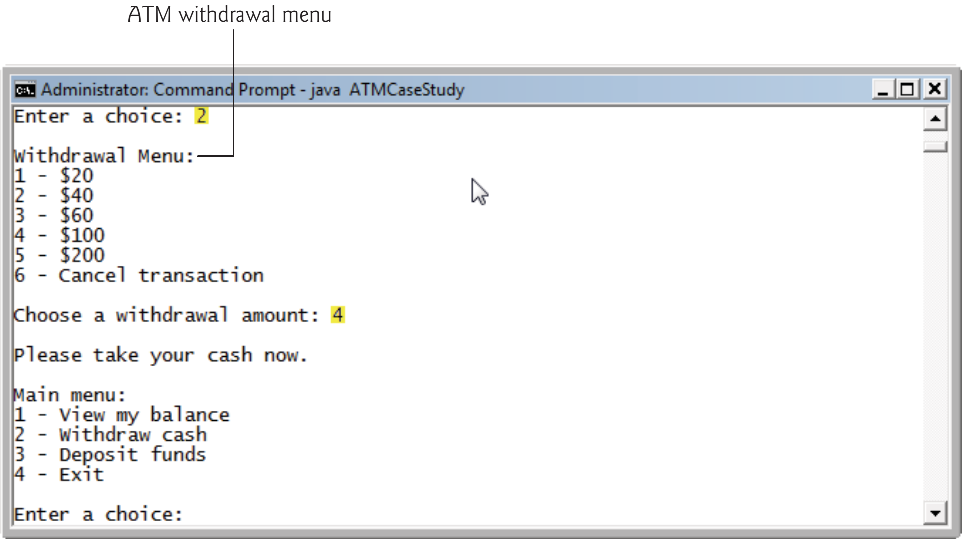
6. ***Viewing the account balance.*** Select option `1`, `"View my balance"`, from the ATM menu (Fig. 1.15). The application then displays two numbers—the `Available balance` (`$1000.00`) and the `Total balance` (`$1200.00`). The available balance is the maximum amount of money in your account which is available for withdrawal at a given time. In some cases, certain funds, such as recent deposits, are not immediately available for the user to withdraw, so the available balance may be less than the total balance, as it is here. After the account-balance information is shown, the application's main menu is displayed again.

7. ***Withdrawing money from the account.*** Select option `2`, `"Withdraw cash"`, from the application menu. You're then presented (Fig. 1.16) with a list of dollar amounts (e.g., `20`, `40`, `60`, `100` and `200`). You're also given the option to cancel the transaction and return to the main menu. Withdraw $100 by selecting option `4`. The application displays `"Please take your cash now."` and returns to the main menu. [*Note:* Unfortunately, this application only *simulates* the behavior of a real ATM and thus does not actually dispense money.]


Account-balance information
Administrator: Command Prompt - java ATMCaseStudy

```
Enter a choice: 1

Balance Information:
 - Available balance:  $1,000.00
 - Total balance:      $1,200.00

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice:
```

Fig. 1.15 | ATM application displaying user account-balance information.



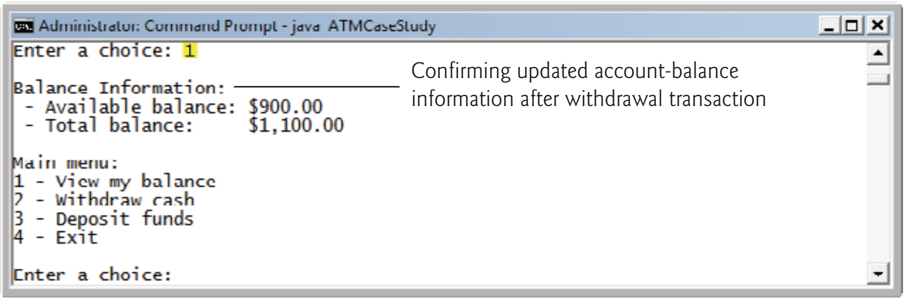
```
Enter a choice: 2

Withdrawal Menu:
1 - $20
2 - $40
3 - $60
4 - $100
5 - $200
6 - Cancel transaction

Choose a withdrawal amount: 4

Please take your cash now.

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice:
```

Fig. 1.16 | Withdrawing money from the account and returning to the main menu.

8. *Confirming that the account information has been updated.* From the main menu, select option `1` again to view your current account balance (Fig. 1.17). Both the available balance and the total balance have been updated to reflect your withdrawal transaction.

9. *Ending the transaction.* To end your current ATM session, select option `4`, `"Exit"`, from the main menu (Fig. 1.18). The ATM will exit the system and display a goodbye message to the user. The application will then return to its original prompt, asking for the next user's account number.

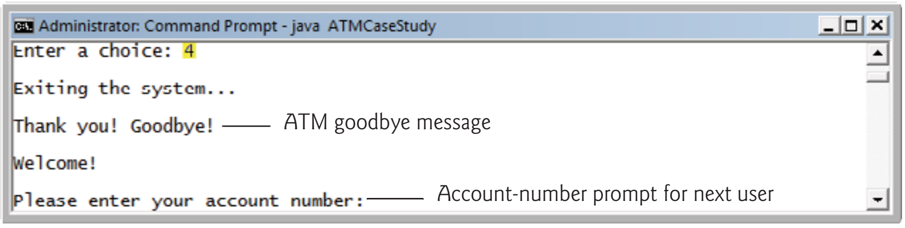
10. *Exiting the ATM and closing the* **Command Prompt** *window.* Most applications provide an option to exit and return to the **Command Prompt** directory from which the application was run. A real ATM does not provide a user with the option to turn off the ATM. Rather, when a user has completed all desired transactions and chosen



```
Enter a choice: 1

Balance Information:
 - Available balance:  $900.00
 - Total balance:      $1,100.00

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice:
```

Confirming updated account-balance information after withdrawal transaction

Fig. 1.17 | Checking the new balance.



```
Enter a choice: 4

Exiting the system...

Thank you! Goodbye!

Welcome!

Please enter your account number:
```

ATM goodbye message

Account-number prompt for next user

Fig. 1.18 | Ending an ATM transaction session.

the menu option to exit, the ATM resets itself and displays a prompt for the next user's account number. As Fig. 1.18 illustrates, the ATM application here behaves similarly. Choosing the menu option to exit ends only the current user's ATM session, not the entire ATM application. To actually exit the ATM application, click the close (x) button in the upper-right corner of the **Command Prompt** window. Closing the window causes the running application to terminate.

1.11 Web 2.0: Going Social

The web literally exploded in the mid-to-late 1990s, but the "dot com" economic bust brought hard times in the early 2000s. The resurgence that began in 2004 or so has been named **Web 2.0**. Google is widely regarded as the signature company of Web 2.0. Some other companies with "Web 2.0 characteristics" are YouTube (video sharing), FaceBook (social networking), Twitter (microblogging), Groupon (social commerce), Foursquare (mobile check-in), Salesforce (business software offered as online services), Craigslist (free classified listings), Flickr (photo sharing), Second Life (a virtual world), Skype (Internet telephony) and Wikipedia (a free online encyclopedia).

Google

In 1996, Stanford computer science Ph.D. candidates Larry Page and Sergey Brin began collaborating on a new search engine. In 1997, they changed the name to Google—a play on the mathematical term *googol*, a quantity represented by the number "one" followed by 100 "zeros" (or 10^{100})—a staggeringly large number. Google's ability to return extremely accurate search results quickly helped it become the most widely used search engine and one of the most popular websites in the world.

Google continues to be an innovator in search technologies. For example, Google Goggles is a fascinating mobile app (available on Android and iPhone) that allows you to perform a Google search using a photo rather than entering text. You simply take pictures of a landmarks, books (covers or barcodes), logos, art or wine bottle labels, and Google Goggles scans the photo and returns search results. You can also take a picture of text (for example, a restaurant menu or a sign) and Google Goggles will translate it for you.

Web Services and Mashups

We include in this book a substantial treatment of web services (Chapter 31) and introduce the applications-development methodology of *mashups* in which you can rapidly develop powerful and intriguing applications by combining (often free) complementary web services and other forms of information feeds (Fig. 1.19). One of the first mashups was `www.housingmaps.com`, which quickly combines the real estate listings provided by `www.craigslist.org` with the mapping capabilities of *Google Maps* to offer maps that show the locations of apartments for rent in a given area.

Web services source	How they're used
Google Maps	Mapping services
Facebook	Social networking
Foursquare	Mobile check-in
LinkedIn	Social networking for business
YouTube	Video search
Twitter	Microblogging
Groupon	Social commerce
Netflix	Movie rentals
eBay	Internet auctions
Wikipedia	Collaborative encyclopedia
PayPal	Payments
Last.fm	Internet radio
Amazon eCommerce	Shopping for books and more
Salesforce.com	Customer Relationship Management (CRM)
Skype	Internet telephony
Microsoft Bing	Search
Flickr	Photo sharing
Zillow	Real estate pricing
Yahoo Search	Search
WeatherBug	Weather

Fig. 1.19 | Some popular web services (`www.programmableweb.com/apis/directory/1?sort=mashups`).

Ajax

Ajax is one of the premier Web 2.0 software technologies. Ajax helps Internet-based applications perform like desktop applications—a difficult task, given that such applications suffer transmission delays as data is shuttled back and forth between your computer and server computers on the Internet. Using Ajax, applications like Google Maps have achieved excellent performance and approach the look-and-feel of desktop applications. Although we don't discuss "raw" Ajax programming (which is quite complex) in this text, we do show in Chapter 30 how to build Ajax-enabled applications using JavaServer Faces (JSF) Ajax-enabled components.

Social Applications

Over the last several years, there's been a tremendous increase in the number of social applications on the web. Even though the computer industry is mature, these sites were still able to become phenomenally successful in a relatively short period of time. Figure 1.20 discusses a few of the social applications that are making an impact.

Company	Description
Facebook	Facebook was launched from a Harvard dorm room in 2004 by classmates Mark Zuckerberg, Chris Hughes, Dustin Moskovitz and Eduardo Saverin and is already worth an estimated $70 billion. By January 2011, Facebook was the most active site on the Internet with more than 600 million users—nearly 9% of the Earth's population—who spend 700 billion minutes on Facebook per month (`www.time.com/time/specials/packages/article/0,28804,2036683_2037183,00.html`). At its current growth rate (about 5% per month), Facebook will reach one billion users in 2012, out of the two billion Internet users! The activity on the site makes it extremely attractive for application developers. Each day, over 20 million applications are installed by Facebook users (`www.facebook.com/press/info.php?statistics`).
Twitter	Twitter was founded in 2006 by Jack Dorsey, Evan Williams and Isaac "Biz" Stone—all from the podcast company, Odeo. Twitter has revolutionized *microblogging*. Users post tweets—messages of up to 140 characters long. Approximately 95 million tweets are posted per day (`twitter.com/about`). You can follow the tweets of friends, celebrities, businesses, government representatives (including the U.S. President, who has 6.3 million followers), etc., or you can follow tweets by subject to track news, trends and more. At the time of this writing, Lady Gaga had the most followers (over 7.7 million). Twitter has become the point of origin for many breaking news stories worldwide.
Groupon	Groupon, a *social commerce* site, was launched by Andrew Mason in 2008. By January 2011, the company was valued around $15 billion, making it the fastest growing company ever! It's now available in hundreds of markets worldwide. Groupon offers one daily deal in each market for restaurants, retailers, services, attractions and more. Deals are activated only after a minimum number of people sign up to buy the product or service. If you sign up for a deal and it has yet to meet the minimum, you might be inclined to tell others about the deal by email, Facebook, Twitter, etc. If the deal does not meet the minimum sales, it's cancelled. One of the most successful national Groupon deals to date was a certificate for $50 worth of merchandise from a major apparel company for $25. Over 440,000 vouchers were sold in one day.

Fig. 1.20 | Social applications. (Part 1 of 2.)

Company	Description
Foursquare	Foursquare—launched in 2009 by Dennis Crowley and Naveen Selvadurai—is a mobile *check-in* application that allows you to notify your friends of your whereabouts. You can download the app to your smartphone and link it to your Facebook and Twitter accounts so your friends can follow you from multiple platforms. If you do not have a smartphone, you can check in by text message. Foursquare uses GPS to determine your exact location. Businesses use Foursquare to send offers to users in the area. Launched in March 2009, Foursquare already has over 5 million users worldwide.
Skype	Skype is a software product that allows you to make mostly free voice and video calls over the Internet using a technology called *VoIP (Voice over IP*; IP stands for "Internet Protocol"*)*. Skype was founded in 2003 by Niklas Zennström and Dane Janus Friis. Just two years later, the company was sold to eBay for $2.6 billion.
YouTube	YouTube is a video-sharing site that was founded in 2005. Within one year, the company was purchased by Google for $1.65 billion. YouTube now accounts for 10% of all Internet traffic (`www.webpronews.com/topnews/2010/04/16/facebook-and-youtube-get-the-most-business-internet-traffic`). Within one week of the release of Apple's iPhone 3GS—the first iPhone model to offer video—mobile uploads to YouTube grew 400% (`www.hypebot.com/hypebot/2009/06/youtube-reports-1700-jump-in-mobile-video.html`).

Fig. 1.20 | Social applications. (Part 2 of 2.)

1.12 Software Technologies

Figure 1.21 lists a number of buzzwords that you'll hear in the software development community. We've created Resource Centers on most of these topics, with more on the way.

Technology	Description
Agile software development	**Agile software development** is a set of methodologies that try to get software implemented faster and using fewer resources than previous methodologies. Check out the Agile Alliance (`www.agilealliance.org`) and the Agile Manifesto (`www.agilemanifesto.org`).
Refactoring	**Refactoring** involves reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality. It's widely employed with agile development methodologies. Many IDEs contain built-in *refactoring tools* to do major portions of the reworking automatically.
Design patterns	**Design patterns** are proven architectures for constructing flexible and maintainable object-oriented software. The field of design patterns tries to enumerate those recurring patterns, encouraging software designers to *reuse* them to develop better-quality software using less time, money and effort. We discuss Java design patterns in Appendix Q.

Fig. 1.21 | Software technologies. (Part 1 of 2.)

Technology	Description
LAMP	MySQL is an open-source database management system. PHP is the most popular open-source server-side "scripting" language for developing web applications. **LAMP** is an acronym for the open-source technologies that many developers use to build web applications—it stands for Linux, Apache, MySQL and PHP (or Perl or Python—two other scripting languages).
Software as a Service (SaaS)	Software has generally been viewed as a product; most software still is offered this way. To run an application, you buy it from a software vendor. You then install it on your computer and run it as needed. As new versions appear, you upgrade the software, often at considerable expense. This process can be cumbersome for organizations with tens of thousands of systems that must be maintained on a diverse array of computer equipment. With **Software as a Service (SaaS)**, the software runs on servers elsewhere on the Internet. When that server is updated, all clients worldwide see the new capabilities—no local installation is needed. You access the service through a browser. Browsers are quite portable, so you can run the same applications on a wide variety of computers from anywhere in the world. Salesforce.com, Google, and Microsoft's Office Live and Windows Live all offer SaaS.
Platform as a Service (PaaS)	**Platform as a Service (PaaS)** provides a computing platform for developing and running applications as a service over the web, rather than installing the tools on your computer. PaaS providers include Google App Engine, Amazon EC2, Bungee Labs and more.
Cloud computing	SaaS and PaaS are examples of **cloud computing** in which software, platforms and infrastructure (e.g., processing power and storage) are hosted on demand over the Internet. This provides users with flexibility, scalability and cost savings. For example, consider a company's data storage needs which can fluctuate significantly over the course of a year. Rather than investing in large-scale storage hardware—which can be costly to purchase, maintain and secure, and would most likely not be used to capacity at all times—the company could purchase cloud-based services (such as Amazon S3, Google Storage, Microsoft Windows Azure™, Nirvanix™ and others) dynamically as needed.
Software Development Kit (SDK)	**Software Development Kits (SDKs)** include the tools and documentation developers use to program applications. For example, you'll use the Java Development Kit (JDK) to build and run Java applications.

Fig. 1.21 | Software technologies. (Part 2 of 2.)

Figure 1.22 describes software product release categories.

Version	Description
Alpha	*Alpha* software is the earliest release of a software product that's still under active development. Alpha versions are often buggy, incomplete and unstable, and are released to a relatively small number of developers for testing new features, getting early feedback, etc.

Fig. 1.22 | Software product release terminology. (Part 1 of 2.)

Version	Description
Beta	*Beta* versions are released to a larger number of developers later in the development process after most major bugs have been fixed and new features are nearly complete. Beta software is more stable, but still subject to change.
Release candidates	*Release candidates* are generally *feature complete* and (supposedly) bug free, and ready for use by the community, which provides a diverse testing environment—the software is used on different systems, with varying constraints and for a variety of purposes. Any bugs that appear are corrected and eventually the final product is released to the general public. Software companies often distribute incremental updates over the Internet.
Continuous beta	Software that's developed using this approach generally does not have version numbers (for example, Google search or Gmail). The software, which is hosted in the cloud (not installed on your computer), is constantly evolving so that users always have the latest version.

Fig. 1.22 | Software product release terminology. (Part 2 of 2.)

1.13 Keeping Up-to-Date with Information Technologies

Figure 1.23 lists key technical and business publications that will help you stay up-to-date with the latest news and trends and technology. You can also find a growing list of Internet- and web-related Resource Centers at `www.deitel.com/ResourceCenters.html`.

Publication	URL
Bloomberg BusinessWeek	`www.businessweek.com`
CNET	`news.cnet.com`
Computer World	`www.computerworld.com`
Engadget	`www.engadget.com`
eWeek	`www.eweek.com`
Fast Company	`www.fastcompany.com/`
Fortune	`money.cnn.com/magazines/fortune/`
InfoWorld	`www.infoworld.com`
Mashable	`mashable.com`
PCWorld	`www.pcworld.com`
SD Times	`www.sdtimes.com`
Slashdot	`slashdot.org/`
Smarter Technology	`www.smartertechnology.com`
Technology Review	`technologyreview.com`
Techcrunch	`techcrunch.com`
Wired	`www.wired.com`

Fig. 1.23 | Technical and business publications.

1.14 Wrap-Up

In this chapter we discussed computer hardware, software, programming languages and operating systems. We overviewed a typical Java program development environment and you test-drove a Java application. We introduced the basics of object technology. You learned about some of the exciting recent developments in the computer field. We also discussed some key software development terminology.

In Chapter 2, you'll create your first Java applications. You'll see how programs display messages on the screen and obtain information from the user at the keyboard for processing. You'll use Java's primitive data types and arithmetic operators in calculations and use Java's equality and relational operators to write simple decision-making statements.

Self-Review Exercises

1.1 Fill in the blanks in each of the following statements:

a) The company that popularized personal computing was ________.

b) The computer that made personal computing legitimate in business and industry was the ________.

c) Computers process data under the control of sets of instructions called ________.

d) The key logical units of the computer are the ________, ________, ________, ________, ________ and ________.

e) The three types of languages discussed in the chapter are ________, ________ and ________.

f) The programs that translate high-level language programs into machine language are called ________.

g) ________ is a smartphone operating system based on the Linux kernel and Java.

h) ________ software is generally feature complete and (supposedly) bug free, and ready for use by the community.

i) The Wii Remote, as well as many smartphones, use a(n) ________ which allows the device to respond to motion.

1.2 Fill in the blanks in each of the following sentences about the Java environment:

a) The ________ command from the JDK executes a Java application.

b) The ________ command from the JDK compiles a Java program.

c) A Java program file must end with the ________ file extension.

d) When a Java program is compiled, the file produced by the compiler ends with the ________ file extension.

e) The file produced by the Java compiler contains ________ that are executed by the Java Virtual Machine.

1.3 Fill in the blanks in each of the following statements (based on Section 1.6):

a) Objects have the property of ________—although objects may know how to communicate with one another across well-defined interfaces, they normally are not allowed to know how other objects are implemented.

b) Java programmers concentrate on creating ________, which contain fields and the set of methods that manipulate those fields and provide services to clients.

c) The process of analyzing and designing a system from an object-oriented point of view is called ________.

d) With ________, new classes of objects are derived by absorbing characteristics of existing classes, then adding unique characteristics of their own.

e) ________ is a graphical language that allows people who design software systems to use an industry-standard notation to represent them.

f) The size, shape, color and weight of an object are considered ________ of the object's class.

Answers to Self-Review Exercises

1.1 a) Apple. b) IBM Personal Computer. c) programs. d) input unit, output unit, memory unit, central processing unit, arithmetic and logic unit, secondary storage unit. e) machine languages, assembly languages, high-level languages. f) compilers. g) Android. h) Release candidate. i) accelerometer.

1.2 a) `java`. b) `javac`. c) `.java`. d) `.class`. e) bytecodes.

1.3 a) information hiding. b) classes. c) object-oriented analysis and design (OOAD). d) inheritance. e) The Unified Modeling Language (UML). f) attributes.

Exercises

1.4 Fill in the blanks in each of the following statements:

a) The logical unit of the computer that receives information from outside the computer for use by the computer is the ________.

b) The process of instructing the computer to solve a problem is called ________.

c) ________ is a type of computer language that uses English-like abbreviations for machine-language instructions.

d) ________ is a logical unit of the computer that sends information which has already been processed by the computer to various devices so that it may be used outside the computer.

e) ________ and ________ are logical units of the computer that retain information.

f) ________ is a logical unit of the computer that performs calculations.

g) ________ is a logical unit of the computer that makes logical decisions.

h) ________ languages are most convenient to the programmer for writing programs quickly and easily.

i) The only language a computer can directly understand is that computer's ________.

j) ________ is a logical unit of the computer that coordinates the activities of all the other logical units.

1.5 Fill in the blanks in each of the following statements:

a) ________ is now used to develop large-scale enterprise applications, to enhance the functionality of web servers, to provide applications for consumer devices and for many other purposes.

b) ________ initially became widely known as the development language of the Unix operating system.

c) The Web 2.0 company ________ is the fastest growing company ever.

d) The ________ programming language was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories.

1.6 Fill in the blanks in each of the following statements:

a) Java programs normally go through five phases—________, ________, ________, ________ and ________.

b) A(n) ________ provides many tools that support the software development process, such as editors for writing and editing programs, debuggers for locating logic errors in programs, and many other features.

c) The command `java` invokes the ________, which executes Java programs.
d) A(n) ________ is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with it.
e) The ________ takes the `.class` files containing the program's bytecodes and transfers them to primary memory.
f) The ________ examines bytecodes to ensure that they're valid.

1.7 Explain the two compilation phases of Java programs.

1.8 You're probably wearing on your wrist one of the world's most common types of objects—a watch. Discuss how each of the following terms and concepts applies to the notion of a watch: object, attributes, behaviors, class, inheritance (consider, for example, an alarm clock), abstraction, modeling, messages, encapsulation, interface and information hiding.

Making a Difference

Throughout the book we've included Making a Difference exercises in which you'll be asked to work on problems that really matter to individuals, communities, countries and the world. For more information about worldwide organizations working to make a difference, and for related programming project ideas, visit our Making a Difference Resource Center at `www.deitel.com/makingadifference`.

1.9 *(Test Drive: Carbon Footprint Calculator)* Some scientists believe that carbon emissions, especially from the burning of fossil fuels, contribute significantly to global warming and that this can be combatted if individuals take steps to limit their use of carbon-based fuels. Organizations and individuals are increasingly concerned about their "carbon footprints." Websites such as TerraPass

```
www.terrapass.com/carbon-footprint-calculator/
```

and Carbon Footprint

```
www.carbonfootprint.com/calculator.aspx
```

provide carbon footprint calculators. Test-drive these calculators to determine your carbon footprint. Exercises in later chapters will ask you to program your own carbon footprint calculator. To prepare for this, research the formulas for calculating carbon footprints.

1.10 *(Test Drive: Body Mass Index Calculator)* By recent estimates, two-thirds of the people in the United States are overweight and about half of those are obese. This causes significant increases in illnesses such as diabetes and heart disease. To determine whether a person is overweight or obese, you can use a measure called the body mass index (BMI). The United States Department of Health and Human Services provides a BMI calculator at `www.nhlbisupport.com/bmi/`. Use it to calculate your own BMI. An exercise in Chapter 2 will ask you to program your own BMI calculator. To prepare for this, research the formulas for calculating BMI.

1.11 *(Attributes of Hybrid Vehicles)* In this chapter you learned the basics of classes. Now you'll begin "fleshing out" aspects of a class called "Hybrid Vehicle." Hybrid vehicles are becoming increasingly popular, because they often get much better mileage than purely gasoline-powered vehicles. Browse the web and study the features of four or five of today's popular hybrid cars, then list as many of their hybrid-related attributes as you can. For example, common attributes include city-miles-per-gallon and highway-miles-per-gallon. Also list the attributes of the batteries (type, weight, etc.).

1.12 *(Gender Neutrality)* Many people want to eliminate sexism in all forms of communication. You've been asked to create a program that can process a paragraph of text and replace gender-specific words with gender-neutral ones. Assuming that you've been given a list of gender-specific words and their gender-neutral replacements (e.g., replace "wife" by "spouse," "man" by "person," "daughter" by "child" and so on), explain the procedure you'd use to read through a paragraph of text and manually perform these replacements. How might your procedure generate a strange term like “woperchild,” which is actually listed in the Urban Dictionary (www.urbandictionary.com)? In Chapter 4, you’ll learn that a more formal term for “procedure” is “algorithm,” and that an algorithm specifies the steps to be performed and the order in which to perform them.

1.13 *(Privacy)* Some online email services save all email correspondence for some period of time. Suppose a disgruntled employee of one of these online email services were to post all of the email correspondences for millions of people, including yours, on the Internet. Discuss the issues.

1.14 *(Programmer Responsibility and Liability)* As a programmer in industry, you may develop software that could affect people’s health or even their lives. Suppose a software bug in one of your programs were to cause a cancer patient to receive an excessive dose during radiation therapy and that the person is either severely injured or dies. Discuss the issues.

1.15 *(2010 “Flash Crash”)* An example of the consequences of our excessive dependency on computers was the so-called “flash crash” which occurred on May 6, 2010, when the U.S. stock market fell precipitously in a matter of minutes, wiping out trillions of dollars of investments, and then recovered within minutes. Use the Internet to investigate the causes of this crash and discuss the issues it raises.

Making a Difference Resources

The *Microsoft Image Cup* is a global competition in which students use technology to try to solve some of the world’s most difficult problems, such as environmental sustainability, ending hunger, emergency response, literacy, combating HIV/AIDS and more. Visit www.imaginecup.com/about for more information about the competition and to learn about the projects developed by previous winners. You can also find several project ideas submitted by worldwide charitable organizations at www.imaginecup.com/students/imagine-cup-solve-this. For additional ideas for programming projects that can make a difference, search the web for “making a difference” and visit the following websites:

www.un.org/millenniumgoals

The United Nations Millennium Project seeks solutions to major worldwide issues such as environmental sustainability, gender equality, child and maternal health, universal education and more.

www.ibm.com/smarterplanet/

The IBM® Smarter Planet website discusses how IBM is using technology to solve issues related to business, cloud computing, education, sustainability and more.

www.gatesfoundation.org/Pages/home.aspx

The Bill and Melinda Gates Foundation provides grants to organizations that work to alleviate hunger, poverty and disease in developing countries. In the United States, the foundation focusses on improving public education, particularly for people with few resources.

www.nethope.org/

NetHope is a collaboration of humanitarian organizations worldwide working to solve technology problems such as connectivity, emergency response and more.

www.rainforestfoundation.org/home

The Rainforest Foundation works to preserve rainforests and to protect the rights of the indigenous people who call the rainforests home. The site includes a list of things you can do to help.

www.undp.org/

The United Nations Development Programme (UNDP) seeks solutions to global challenges such as crisis prevention and recovery, energy and the environment, democratic governance and more.

www.unido.org

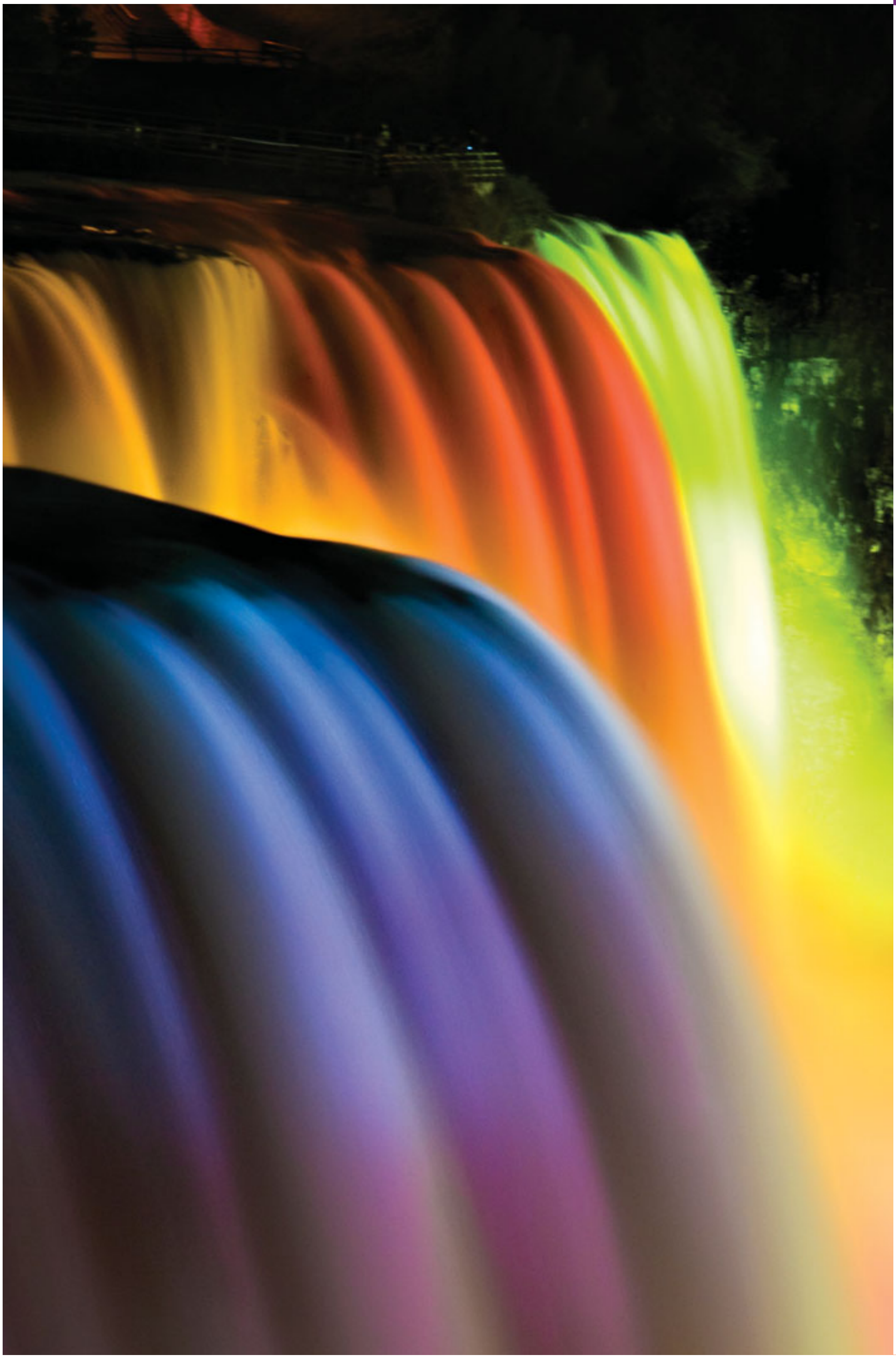
The United Nations Industrial Development Organization (UNIDO) seeks to reduce poverty, give developing countries the opportunity to participate in global trade, and promote energy efficiency and sustainability.

`www.usaid.gov/`
USAID promotes global democracy, health, economic growth, conflict prevention, humanitarian aid and more.

`www.toyota.com/ideas-for-good/`
Toyota's Ideas for Good website describes several Toyota technologies that are making a difference—including their Advanced Parking Guidance System, Hybrid Synergy Drive®, Solar Powered Ventilation System, T.H.U.M.S. (Total Human Model for Safety) and Touch Tracer Display. You can participate in the Ideas for Good challenge by submitting a short essay or video describing how these technologies can be used for other good purposes.

Introduction to Java Applications

2



What's in a name?
That which we call a rose
By any other name would
smell as sweet.
—William Shakespeare

When faced with a decision,
I always ask, "What would be
the most fun?"
—Peggy Walker

The chief merit of language
is clearness.
—Galen

One person can make a
difference and every person
should try.
—John F. Kennedy

Objectives

In this chapter you'll learn:

- To write simple Java applications.
- To use input and output statements.
- Java's primitive types.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

Outline

2.1 Introduction

This chapter introduces Java application programming. We begin with examples of programs that display messages on the screen. We then present a program that obtains two numbers from a user, calculates their sum and displays the result. You'll learn how to instruct the computer to perform arithmetic calculations and save their results for later use. The last example demonstrates how to make decisions. The application compares numbers, then displays messages that show the comparison results.

This chapter uses tools from the JDK to compile and run programs. We've also posted Dive Into® videos at `www.deitel.com/books/jhtp9/` to help you get started with the popular Eclipse and NetBeans integrated development environments.

2.2 Your First Program in Java: Printing a Line of Text

A Java **application** is a computer program that executes when you use the **`java` command** to launch the Java Virtual Machine (JVM). Later in this section we'll discuss how to compile and run a Java application. First we consider a simple application that displays a line of text. Figure 2.1 shows the program followed by a box that displays its output. The program includes line numbers. We've added these for instructional purposes—they're *not* part of a Java program. This example illustrates several important Java features. We'll see that line 9 does the real work—displaying the phrase `Welcome to Java Programming!` on the screen.

```
 1  // Fig. 2.1: Welcome1.java
 2  // Text-printing program.
 3
 4  public class Welcome1
 5  {
 6     // main method begins execution of Java application
 7     public static void main( String[] args )
 8     {
 9        System.out.println( "Welcome to Java Programming!" );
10     } // end method main
11  } // end class Welcome1
```

```
Welcome to Java Programming!
```

Fig. 2.1 | Text-printing program.

Commenting Your Programs

We insert **comments** to **document programs** and improve their readability. The Java compiler ignores comments, so they do *not* cause the computer to perform any action when the program is run.

By convention, we begin every program with a comment indicating the figure number and file name. The comment in line 1

```
// Fig. 2.1: Welcome1.java
```

begins with **//**, indicating that it is an **end-of-line comment**—it terminates at the end of the line on which the // appears. An end-of-line comment need not begin a line; it also can begin in the middle of a line and continue until the end (as in lines 10 and 11). Line 2

```
// Text-printing program.
```

is a comment that describes the purpose of the program.

Java also has **traditional comments**, which can be spread over several lines as in

```
/* This is a traditional comment. It
   can be split over multiple lines */
```

These begin and end with delimiters, **/*** and ***/**. The compiler ignores all text between the delimiters. Java incorporated traditional comments and end-of-line comments from the C and C++ programming languages, respectively. In this book, we use only // comments.

Java provides comments of a third type, **Javadoc comments**. These are delimited by **/**** and ***/**. The compiler ignores all text between the delimiters. Javadoc comments enable you to embed program documentation directly in your programs. Such comments are the preferred Java documenting format in industry. The **javadoc utility program** (part of the Java SE Development Kit) reads Javadoc comments and uses them to prepare your program's documentation in HTML format. We demonstrate Javadoc comments and the javadoc utility in Appendix M, Creating Documentation with javadoc.

Common Programming Error 2.1

Forgetting one of the delimiters of a traditional or Javadoc comment is a syntax error. A ***syntax error*** *occurs when the compiler encounters code that violates Java's language rules (i.e., its syntax). These rules are similar to a natural language's grammar rules specifying sentence structure. Syntax errors are also called* ***compiler errors, compile-time errors*** *or* ***compilation errors****, because the compiler detects them during the compilation phase. The compiler responds by issuing an error message and preventing your program from compiling.*

Good Programming Practice 2.1

Some organizations require that every program begin with a comment that states the purpose of the program and the author, date and time when the program was last modified.

Using Blank Lines

Line 3 is a blank line. Blank lines, space characters and tabs make programs easier to read. Together, they're known as **white space** (or whitespace). The compiler ignores white space.

Good Programming Practice 2.2

Use blank lines and spaces to enhance program readability.

Declaring a Class

Line 4

```
    public class Welcome1
```

begins a **class declaration** for class `Welcome1`. Every Java program consists of at least one class that you (the programmer) define. The **`class` keyword** introduces a class declaration and is immediately followed by the **class name** (`Welcome1`). **Keywords** (sometimes called **reserved words**) are reserved for use by Java and are always spelled with all lowercase letters. The complete list of keywords is shown in Appendix C.

Class Names and Identifiers

By convention, class names begin with a capital letter and capitalize the first letter of each word they include (e.g., `SampleClassName`). A class name is an **identifier**—a series of characters consisting of letters, digits, underscores (_) and dollar signs ($) that does not begin with a digit and does not contain spaces. Some valid identifiers are `Welcome1`, `$value`, `_value`, `m_inputField1` and `button7`. The name `7button` is not a valid identifier because it begins with a digit, and the name `input field` is not a valid identifier because it contains a space. Normally, an identifier that does not begin with a capital letter is not a class name. Java is **case sensitive**—uppercase and lowercase letters are distinct—so `value` and `Value` are different (but both valid) identifiers.

In Chapters 2–7, every class we define begins with the **`public`** keyword. For now, we simply require this keyword. For our application, the file name is `Welcome1.java`. You'll learn more about `public` and non-`public` classes in Chapter 8.

Common Programming Error 2.2

A `public` class must be placed in a file that has the same name as the class (in terms of both spelling and capitalization) plus the `.java` extension; otherwise, a compilation error occurs. For example, `public` class `Welcome` must be placed in a file named `Welcome.java`.

A **left brace** (as in line 5), **`{`**, begins the **body** of every class declaration. A corresponding **right brace** (at line 11), **`}`**, must end each class declaration. Lines 6–10 are indented.

Error-Prevention Tip 2.1

When you type an opening left brace, `{`, immediately type the closing right brace, `}`, then reposition the cursor between the braces and indent to begin typing the body. This practice helps prevent errors due to missing braces. Many IDEs insert the braces for you.

Common Programming Error 2.3

It's a syntax error if braces do not occur in matching pairs.

Good Programming Practice 2.3

Indent the entire body of each class declaration one "level" between the left brace and the right brace that delimit the body of the class. We recommend using three spaces to form a level of indent. This format emphasizes the class declaration's structure and makes it easier to read.

Good Programming Practice 2.4

Many IDEs insert indentation for you in all the right places. The Tab *key may also be used to indent code, but tab stops vary among text editors. Most IDEs allow you to configure tabs such that a specified number of spaces is inserted each time you press the* Tab *key.*

Declaring a Method

Line 6

```
    // main method begins execution of Java application
```

is an end-of-line comment indicating the purpose of lines 7–10 of the program. Line 7

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

is the starting point of every Java application. The **parentheses** after the identifier `main` indicate that it's a program building block called a **method**. Java class declarations normally contain one or more methods. For a Java application, one of the methods *must* be called `main` and must be defined as shown in line 7; otherwise, the Java Virtual Machine (JVM) will not execute the application. Methods perform tasks and can return information when they complete their tasks. Keyword **`void`** indicates that this method will *not* return any information. Later, we'll see how a method can return information. For now, simply mimic `main`'s first line in your Java applications. In line 7, the `String[] args` in parentheses is a required part of the method `main`'s declaration—we discuss this in Chapter 7.

The left brace in line 8 begins the **body of the method declaration**. A corresponding right brace must end it (line 10). Line 9 in the method body is indented between the braces.

Good Programming Practice 2.5

Indent the entire body of each method declaration one "level" between the braces that define the body of the method. This makes the structure of the method stand out and makes the method declaration easier to read.

Performing Output with `System.out.println`

Line 9

```
    System.out.println( "Welcome to Java Programming!" );
```

instructs the computer to perform an action—namely, to print the **string** of characters contained between the double quotation marks (but not the quotation marks themselves). A string is sometimes called a **character string** or a **string literal**. White-space characters in strings are *not* ignored by the compiler. Strings cannot span multiple lines of code, but as you'll see later, this does not restrict you from using long strings in your code.

The **`System.out`** object is known as the **standard output object**. It allows a Java applications to display information in the **command window** from which it executes. In recent versions of Microsoft Windows, the command window is the **Command Prompt**. In UNIX/Linux/Mac OS X, the command window is called a **terminal window** or a **shell**. Many programmers call it simply the **command line**.

Method **`System.out.println`** displays (or prints) a line of text in the command window. The string in the parentheses in line 9 is the **argument** to the method. When `System.out.println` completes its task, it positions the output cursor (the location where the next character will be displayed) at the beginning of the next line in the command window. This is similar to what happens when you press the *Enter* key while typing in a text editor—the cursor appears at the beginning of the next line in the document.

The entire line 9, including `System.out.println`, the argument `"Welcome to Java Programming!"` in the parentheses and the **semicolon** (**;**), is called a **statement**. A method typically contains one or more statements that perform its task. Most statements end with a semicolon. When the statement in line 9 executes, it displays `Welcome to Java Programming!` in the command window.

Error-Prevention Tip 2.2

When learning how to program, sometimes it's helpful to "break" a working program so you can familiarize yourself with the compiler's syntax-error messages. These messages do not always state the exact problem in the code. When you encounter an error message, it will give you an idea of what caused the error. [Try removing a semicolon or brace from the program of Fig. 2.1, then recompile the program to see the error messages generated by the omission.]

Error-Prevention Tip 2.3

When the compiler reports a syntax error, it may not be on the line that the error message indicates. First, check the line for which the error was reported. If you don't find an error on that line,, check several preceding lines.

Using End-of-Line Comments on Right Braces for Readability

We include an end-of-line comment after a closing brace that ends a method declaration and after a closing brace that ends a class declaration. For example, line 10

```
   } // end method main
```

indicates the closing brace of method `main`, and line 11

```
} // end class Welcome1
```

indicates the closing brace of class `Welcome1`. Each comment indicates the method or class that the right brace terminates.

Compiling and Executing Your First Java Application

We're now ready to compile and execute our program. We assume you're using the Java Development Kit's command-line tools, not an IDE. Our Java Resource Centers at `www.deitel.com/ResourceCenters.html` provide links to tutorials that help you get started with several popular Java development tools, including NetBeans™, Eclipse™ and others. We've also posted NetBeans and Eclipse videos at `www.deitel.com/books/jhtp9/` to help you get started using these popular IDEs.

To prepare to compile the program, open a command window and change to the directory where the program is stored. Many operating systems use the command `cd` to change directories. On Windows, for example,

```
cd c:\examples\ch02\fig02_01
```

changes to the `fig02_01` directory. On UNIX/Linux/Max OS X, the command

```
cd ~/examples/ch02/fig02_01
```

changes to the `fig02_01` directory.

To compile the program, type

```
    javac Welcome1.java
```

If the program contains no syntax errors, this command creates a new file called `Welcome1.class` (known as the **class file** for `Welcome1`) containing the platform-independent Java bytecodes that represent our application. When we use the `java` command to execute the application on a given platform, the JVM will translate these bytecodes into instructions that are understood by the underlying operating system and hardware.

Error-Prevention Tip 2.4

When attempting to compile a program, if you receive a message such as "`bad command or filename`," "`javac: command not found`" or "`'javac' is not recognized as an internal or external command, operable program or batch file`," then your Java software installation was not completed properly. If you're using the JDK, this indicates that the system's `PATH` environment variable was not set properly. Please carefully review the installation instructions in the Before You Begin section of this book. On some systems, after correcting the `PATH`, you may need to reboot your computer or open a new command window for these settings to take effect.

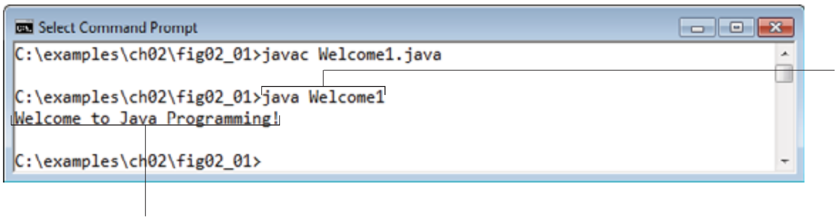
Error-Prevention Tip 2.5

Each syntax-error message contains the file name and line number where the error occurred. For example, `Welcome1.java:6` indicates that an error occurred at line 6 in `Welcome1.java`. The rest of the message provides information about the syntax error.

Error-Prevention Tip 2.6

The compiler error message "`class Welcome1 is public, should be declared in a file named Welcome1.java`" indicates that the file name does not match the name of the public class in the file or that you typed the class name incorrectly when compiling the class.

Figure 2.2 shows the program of Fig. 2.1 executing in a Microsoft® Windows® 7 **Command Prompt** window. To execute the program, type `java Welcome1`. This command launches the JVM, which loads the `.class` file for class `Welcome1`. The command omits the `.class` file-name extension; otherwise, the JVM will not execute the program. The JVM calls method `main`. Next, the statement at line 9 of `main` displays `"Welcome to Java Programming!"` [*Note:* Many environments show command prompts with black backgrounds and white text. We adjusted these settings in our environment to make our screen captures more readable.]



```
C:\examples\ch02\fig02_01>javac Welcome1.java

C:\examples\ch02\fig02_01>java Welcome1
Welcome to Java Programming!

C:\examples\ch02\fig02_01>
```

You type this command to execute the application

The program outputs to the screen `Welcome to Java Programming!`

Fig. 2.2 | Executing `Welcome1` from the **Command Prompt**.

Error-Prevention Tip 2.7

When attempting to run a Java program, if you receive a message such as "`Exception in thread "main" java.lang.NoClassDefFoundError: Welcome1`," your `CLASSPATH` environment variable has not been set properly. Please carefully review the installation instructions in the Before You Begin section of this book. On some systems, you may need to reboot your computer or open a new command window after configuring the `CLASSPATH`.

2.3 Modifying Your First Java Program

In this section, we modify the example in Fig. 2.1 to print text on one line by using multiple statements and to print text on several lines by using a single statement.

Displaying a Single Line of Text with Multiple Statements

`Welcome to Java Programming!` can be displayed several ways. Class `Welcome2`, shown in Fig. 2.3, uses two statements (lines 9–10) to produce the output shown in Fig. 2.1. [*Note:* From this point forward, we highlight the new and key features in each code listing, as we've done for lines 9–10.]

```
 1  // Fig. 2.3: Welcome2.java
 2  // Printing a line of text with multiple statements.
 3
 4  public class Welcome2
 5  {
 6     // main method begins execution of Java application
 7     public static void main( String[] args )
 8     {
 9        System.out.print( "Welcome to " );
10        System.out.println( "Java Programming!" );
11     } // end method main
12  } // end class Welcome2
```

```
Welcome to Java Programming!
```

Fig. 2.3 | Printing a line of text with multiple statements.

The program is similar to Fig. 2.1, so we discuss only the changes here. Line 2

```
    // Printing a line of text with multiple statements.
```

is an end-of-line comment stating the purpose of the program. Line 4 begins the `Welcome2` class declaration. Lines 9–10 of method `main`

```
    System.out.print( "Welcome to " );
    System.out.println( "Java Programming!" );
```

display one line of text. The first statement uses `System.out`'s method `print` to display a string. Each `print` or `println` statement resumes displaying characters from where the last `print` or `println` statement stopped displaying characters. Unlike `println`, after displaying its argument, `print` does *not* position the output cursor at the beginning of the next line in the command window—the next character the program displays will appear *immediately after* the last character that `print` displays. Thus, line 10 positions the first character in its argument (the letter “J”) immediately after the last character that line 9 displays (the *space character* before the string’s closing double-quote character).

Displaying Multiple Lines of Text with a Single Statement

A single statement can display multiple lines by using **newline characters**, which indicate to System.out’s print and println methods when to position the output cursor at the beginning of the next line in the command window. Like blank lines, space characters and tab characters, newline characters are white-space characters. The program in Fig. 2.4 outputs four lines of text, using newline characters to determine when to begin each new line. Most of the program is identical to those in Fig. 2.1 and Fig. 2.3.

```java
 1  // Fig. 2.4: Welcome3.java
 2  // Printing multiple lines of text with a single statement.
 3
 4  public class Welcome3
 5  {
 6     // main method begins execution of Java application
 7     public static void main( String[] args )
 8     {
 9        System.out.println( "Welcome\nto\nJava\nProgramming!" );
10     } // end method main
11  } // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

Line 2

```java
// Printing multiple lines of text with a single statement.
```

is a comment stating the program’s purpose. Line 4 begins the Welcome3 class declaration. Line 9

```java
System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

displays four separate lines of text in the command window. Normally, the characters in a string are displayed *exactly* as they appear in the double quotes. Note, however, that the paired characters \ and n (repeated three times in the statement) do not appear on the screen. The **backslash** (\) is an **escape character**. which has special meaning to System.out’s print and println methods. When a backslash appears in a string, Java combines it with the next character to form an **escape sequence**. The escape sequence \n represents the newline character. When a newline character appears in a string being output with System.out, the newline character causes the screen’s output cursor to move to the beginning of the next line in the command window.

Figure 2.5 lists several common escape sequences and describes how they affect the display of characters in the command window. For the complete list of escape sequences, visit java.sun.com/docs/books/jls/third_edition/html/lexical.html#3.10.6.

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do *not* advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`.

Fig. 2.5 | Some common escape sequences.

2.4 Displaying Text with `printf`

The **`System.out.printf`** method (`f` means “formatted”) displays formatted data. Figure 2.6 uses this method to output the strings `"Welcome to"` and `"Java Programming!"`. Lines 9–10

```
     System.out.printf( "%s\n%s\n",
        "Welcome to", "Java Programming!" );
```

call method `System.out.printf` to display the program’s output. The method call specifies three arguments. When a method requires multiple arguments, they’re placed in a **comma-separated list**.

Good Programming Practice 2.6

Place a space after each comma (`,`) in an argument list to make programs more readable.

```
 1  // Fig. 2.6: Welcome4.java
 2  // Displaying multiple lines with method System.out.printf.
 3
 4  public class Welcome4
 5  {
 6     // main method begins execution of Java application
 7     public static void main( String[] args )
 8     {
 9        System.out.printf( "%s\n%s\n",
10           "Welcome to", "Java Programming!" );
11     } // end method main
12  } // end class Welcome4
```

```
Welcome to
Java Programming!
```

Fig. 2.6 | Displaying multiple lines with method `System.out.printf`.

Lines 9–10 represent only *one* statement. Java allows large statements to be split over many lines. We indent line 10 to indicate that it's a *continuation* of line 9.

Common Programming Error 2.4

Splitting a statement in the middle of an identifier or a string is a syntax error.

Method `printf`'s first argument is a **format string** that may consist of **fixed text** and **format specifiers**. Fixed text is output by `printf` just as it would be by `print` or `println`. Each format specifier is a placeholder for a value and specifies the type of data to output. Format specifiers also may include optional formatting information.

Format specifiers begin with a percent sign (%) followed by a character that represents the data type. For example, the format specifier **`%s`** is a placeholder for a string. The format string in line 9 specifies that `printf` should output two strings, each followed by a newline character. At the first format specifier's position, `printf` substitutes the value of the first argument after the format string. At each subsequent format specifier's position, `printf` substitutes the value of the next argument. So this example substitutes `"Welcome to"` for the first `%s` and `"Java Programming!"` for the second `%s`. The output shows that two lines of text are displayed.

We introduce various formatting features as they're needed in our examples. Appendix G presents the details of formatting output with `printf`.

2.5 Another Application: Adding Integers

Our next application reads (or inputs) two **integers** (whole numbers, such as –22, 7, 0 and 1024) typed by a user at the keyboard, computes their sum and displays it. This program must keep track of the numbers supplied by the user for the calculation later in the program. Programs remember numbers and other data in the computer's memory and access that data through program elements called **variables**. The program of Fig. 2.7 demonstrates these concepts. In the sample output, we use bold text to identify the user's input (i.e., **45** and **72**).

```java
// Fig. 2.7: Addition.java
// Addition program that displays the sum of two numbers.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
   // main method begins execution of Java application
   public static void main( String[] args )
   {
      // create a Scanner to obtain input from the command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

```

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 1 of 2.)

```
17         System.out.print( "Enter first integer: " ); // prompt
18         number1 = input.nextInt(); // read first number from user
19
20         System.out.print( "Enter second integer: " ); // prompt
21         number2 = input.nextInt(); // read second number from user
22
23         sum = number1 + number2; // add numbers, then store total in sum
24
25         System.out.printf( "Sum is %d\n", sum ); // display sum
26      } // end method main
27   } // end class Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 2 of 2.)

`Import` Declarations

Lines 1–2

```
    // Fig. 2.7: Addition.java
    // Addition program that displays the sum of two numbers.
```

state the figure number, file name and purpose of the program.

A great strength of Java is its rich set of predefined classes that you can *reuse* rather than "reinventing the wheel." These classes are grouped into **packages**—named groups of related classes—and are collectively referred to as the **Java class library**, or the **Java Application Programming Interface** (**Java API**). Line 3

```
    import java.util.Scanner; // program uses class Scanner
```

is an **`import` declaration** that helps the compiler locate a class that's used in this program. It indicates that this example uses Java's predefined `Scanner` class (discussed shortly) from package **`java.util`**.

Common Programming Error 2.5

All `import` declarations must appear before the first class declaration in the file. Placing an `import` declaration inside or after a class declaration is a syntax error.

Error-Prevention Tip 2.8

Forgetting to include an `import` declaration for a class used in your program typically results in a compilation error containing a message such as "`cannot find symbol`." When this occurs, check that you provided the proper `import` declarations and that the names in them are correct, including proper capitalization.

Declaring Class `Addition`

Line 5

```
    public class Addition
```

begins the declaration of class `Addition`. The file name for this `public` class must be `Addition.java`. Remember that the body of each class declaration starts with an opening left brace (line 6) and ends with a closing right brace (line 27).

The application begins execution with the `main` method (lines 8–26). The left brace (line 9) marks the beginning of method `main`'s body, and the corresponding right brace (line 26) marks its end. Method `main` is indented one level in the body of class `Addition`, and the code in the body of `main` is indented another level for readability.

Declaring and Creating a `Scanner` *to Obtain User Input from the Keyboard*

A **variable** is a location in the computer's memory where a value can be stored for use later in a program. All Java variables *must* be declared with a **name** and a **type** *before* they can be used. A variable's name enables the program to access the value of the variable in memory. A variable's name can be any valid identifier. A variable's type specifies what kind of information is stored at that location in memory. Like other statements, declaration statements end with a semicolon (`;`).

Line 11

```
      Scanner input = new Scanner( System.in );
```

is a **variable declaration statement** that specifies the name (`input`) and type (`Scanner`) of a variable that's used in this program. A **`Scanner`** enables a program to read data (e.g., numbers and strings) for use in a program. The data can come from many sources, such as the user at the keyboard or a file on disk. Before using a `Scanner`, you must create it and specify the source of the data.

The `=` in line 11 indicates that `Scanner` variable `input` should be **initialized** (i.e., prepared for use in the program) in its declaration with the result of the expression to the right of the equals sign—`new Scanner(System.in)`. This expression uses the **`new`** keyword to create a `Scanner` object that reads characters typed by the user at the keyboard. The **standard input object**, **`System.in`**, enables applications to read bytes of information typed by the user. The `Scanner` translates these bytes into types (like `int`s) that can be used in a program.

Declaring Variables to Store Integers

The variable declaration statements in lines 13–15

```
      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2
```

declare that variables `number1`, `number2` and `sum` hold data of type **`int`**—they can hold integer values (whole numbers such as `72`, `-1127` and `0`). These variables are not yet initialized. The range of values for an `int` is –2,147,483,648 to +2,147,483,647. [*Note:* Actual `int` values may not contain commas.]

Other types of data include **`float`** and **`double`**, for holding real numbers, and **`char`**, for holding character data. Real numbers contain decimal points, such as `3.4`, `0.0` and `-11.19`. Variables of type `char` represent individual characters, such as an uppercase letter (e.g., `A`), a digit (e.g., `7`), a special character (e.g., `*` or `%`) or an escape sequence (e.g., the newline character, `\n`). The types `int`, `float`, `double` and `char` are called **primitive types**.Primitive-type names are keywords and must appear in all lowercase letters. Appendix D summarizes the characteristics of the eight primitive types (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`).

Several variables of the same type may be declared in a single declaration with the variable names separated by commas (i.e., a comma-separated list of variable names). For example, lines 13–15 can also be written as:

```
int number1, // first number to add
    number2, // second number to add
    sum; // sum of number1 and number2
```

Good Programming Practice 2.7

Declare each variable on a separate line. This format allows a descriptive comment to be inserted next to each declaration.

Good Programming Practice 2.8

Choosing meaningful variable names helps a program to be self-documenting (i.e., one can understand the program simply by reading it rather than by reading manuals or viewing an excessive number of comments).

Good Programming Practice 2.9

By convention, variable-name identifiers begin with a lowercase letter, and every word in the name after the first word begins with a capital letter. For example, variable-name identifier `firstNumber` *starts its second word,* `Number`*, with a capital* `N`*.*

Prompting the User for Input

Line 17

```
System.out.print( "Enter first integer: " ); // prompt
```

uses `System.out.print` to display the message `"Enter first integer: "`. This message is called a **prompt** because it directs the user to take a specific action. We use method `print` here rather than `println` so that the user's input appears on the same line as the prompt. Recall from Section 2.2 that identifiers starting with capital letters typically represent class names. So, `System` is a class. Class `System` is part of package **`java.lang`**. Notice that class `System` is not imported with an `import` declaration at the beginning of the program.

Software Engineering Observation 2.1

By default, package `java.lang` *is imported in every Java program; thus, classes in* `java.lang` *are the only ones in the Java API that do not require an* `import` *declaration.*

Obtaining an `int` as Input from the User

Line 18

```
number1 = input.nextInt(); // read first number from user
```

uses `Scanner` object `input`'s `nextInt` method to obtain an integer from the user at the keyboard. At this point the program waits for the user to type the number and press the *Enter* key to submit the number to the program.

Our program assumes that the user enters a valid integer value. If not, a runtime logic error will occur and the program will terminate. Chapter 11, Exception Handling: A Deeper Look, discusses how to make your programs more robust by enabling them to handle such errors. This is also known as making your program *fault tolerant*.

In line 18, we place the result of the call to method `nextInt` (an `int` value) in variable `number1` by using the **assignment operator**, **`=`**. The statement is read as "`number1` gets the value of `input.nextInt()`." Operator `=` is called a **binary operator**, because it has two **operands**—`number1` and the result of the method call `input.nextInt()`. This statement is called an assignment statement, because it assigns a value to a variable. Everything to the *right* of the assignment operator, `=`, is always evaluated *before* the assignment is performed.

Good Programming Practice 2.10

Placing spaces on either side of a binary operator makes the program more readable.

Prompting for and Inputting a Second `int`

Line 20

```
      System.out.print( "Enter second integer: " ); // prompt
```

prompts the user to input the second integer. Line 21

```
      number2 = input.nextInt(); // read second number from user
```

reads the second integer and assigns it to variable `number2`.

Using Variables in a Calculation

Line 23

```
      sum = number1 + number2; // add numbers then store total in sum
```

is an assignment statement that calculates the sum of the variables `number1` and `number2` then assigns the result to variable `sum` by using the assignment operator, `=`. The statement is read as "`sum` *gets* the value of `number1 + number2`." In general, calculations are performed in assignment statements. When the program encounters the addition operation, it performs the calculation using the values stored in the variables `number1` and `number2`. In the preceding statement, the addition operator is a *binary operator*—its *two* operands are the variables `number1` and `number2`. Portions of statements that contain calculations are called **expressions**. In fact, an expression is any portion of a statement that has a *value* associated with it. For example, the value of the expression `number1 + number2` is the *sum* of the numbers. Similarly, the value of the expression `input.nextInt()` is the integer typed by the user.

Displaying the Result of the Calculation

After the calculation has been performed, line 25

```
      System.out.printf( "Sum is %d\n", sum ); // display sum
```

uses method `System.out.printf` to display the `sum`. The format specifier **`%d`** is a placeholder for an `int` value (in this case the value of `sum`)—the letter `d` stands for "decimal integer." The remaining characters in the format string are all fixed text. So, method `printf` displays `"Sum is "`, followed by the value of `sum` (in the position of the `%d` format specifier) and a newline.

Calculations can also be performed *inside* `printf` statements. We could have combined the statements at lines 23 and 25 into the statement

```
      System.out.printf( "Sum is %d\n", ( number1 + number2 ) );
```

The parentheses around the expression `number1 + number2` are not required—they're included to emphasize that the value of the *entire* expression is output in the position of the `%d` format specifier.

Java API Documentation

For each new Java API class we use, we indicate the package in which it's located. This information helps you locate descriptions of each package and class in the Java API documentation. A web-based version of this documentation can be found at

```
      download.oracle.com/javase/6/docs/api/
```

You can download it from

```
      www.oracle.com/technetwork/java/javase/downloads/index.html
```

Appendix E shows how to use this documentation.

2.6 Memory Concepts

Variable names such as `number1`, `number2` and `sum` actually correspond to locations in the computer's memory. Every variable has a **name**, a **type**, a **size** (in bytes) and a **value**.

In the addition program of Fig. 2.7, when the following statement (line 18) executes:

```
      number1 = input.nextInt(); // read first number from user
```

the number typed by the user is placed into a memory location corresponding to the name `number1`. Suppose that the user enters `45`. The computer places that integer value into location `number1` (Fig. 2.8), replacing the previous value (if any) in that location. The previous value is lost.

Fig. 2.8 | Memory location showing the name and value of variable `number1`.

When the statement (line 21)

```
      number2 = input.nextInt(); // read second number from user
```

executes, suppose that the user enters `72`. The computer places that integer value into location `number2`. The memory now appears as shown in Fig. 2.9.

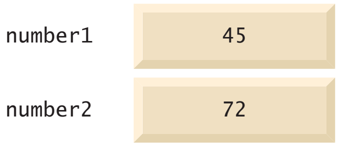


Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

After the program of Fig. 2.7 obtains values for `number1` and `number2`, it adds the values and places the total into variable `sum`. The statement (line 23)

```
        sum = number1 + number2; // add numbers, then store total in sum
```

performs the addition, then replaces any previous value in `sum`. After `sum` has been calculated, memory appears as shown in Fig. 2.10. The values of `number1` and `number2` appear exactly as they did before they were used in the calculation of `sum`. These values were used, but not destroyed, as the computer performed the calculation. When a value is read from a memory location, the process is nondestructive.

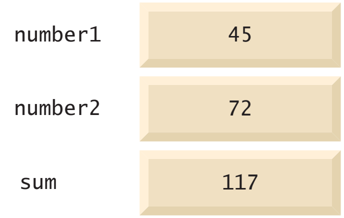


Fig. 2.10 | Memory locations after storing the sum of `number1` and `number2`.

2.7 Arithmetic

Most programs perform arithmetic calculations. The **arithmetic operators** are summarized in Fig. 2.11. Note the use of various special symbols not used in algebra. The **asterisk** (`*`) indicates multiplication, and the percent sign (**`%`**) is the **remainder operator**, which we'll discuss shortly. The arithmetic operators in Fig. 2.11 are *binary* operators, because each operates on *two* operands. For example, the expression `f + 7` contains the binary operator `+` and the two operands `f` and `7`.

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.11 | Arithmetic operators.

Integer division yields an integer quotient. For example, the expression `7 / 4` evaluates to `1`, and the expression `17 / 5` evaluates to `3`. Any fractional part in integer division is simply *discarded* (i.e., *truncated*)—no rounding occurs. Java provides the remainder operator, `%`, which yields the remainder after division. The expression `x % y` yields the remainder after `x` is divided by `y`. Thus, `7 % 4` yields `3`, and `17 % 5` yields `2`. This operator is most commonly used with integer operands but can also be used with other arithmetic types. In this chapter's exercises and in later chapters, we consider several interesting applications of the remainder operator, such as determining whether one number is a multiple of another.

Arithmetic Expressions in Straight-Line Form

Arithmetic expressions in Java must be written in **straight-line form** to facilitate entering programs into the computer. Thus, expressions such as "a divided by b" must be written as `a / b`, so that all constants, variables and operators appear in a straight line. The following algebraic notation is generally not acceptable to compilers:

$$\frac{a}{b}$$

Parentheses for Grouping Subexpressions

Parentheses are used to group terms in Java expressions in the same manner as in algebraic expressions. For example, to multiply `a` times the quantity `b + c`, we write

```
a * ( b + c )
```

If an expression contains **nested parentheses**, such as

```
( ( a + b ) * c )
```

the expression in the innermost set of parentheses (`a + b` in this case) is evaluated first.

Rules of Operator Precedence

Java applies the operators in arithmetic expressions in a precise sequence determined by the **rules of operator precedence**, which are generally the same as those followed in algebra:

1. Multiplication, division and remainder operations are applied first. If an expression contains several such operations, they're applied from left to right. Multiplication, division and remainder operators have the same level of precedence.
2. Addition and subtraction operations are applied next. If an expression contains several such operations, the operators are applied from left to right. Addition and subtraction operators have the same level of precedence.

These rules enable Java to apply operators in the correct order.[1] When we say that operators are applied from left to right, we're referring to their **associativity**. Some operators associate from right to left. Figure 2.12 summarizes these rules of operator precedence. A complete precedence chart is included in Appendix A.

Operator(s)	Operation(s)	Order of evaluation (precedence)

`*`
`/`
`%`
	Multiplication
	Division
	Remainder
		Evaluated first. If there are several operators of this type, they're evaluated from left to right.

`+` `-`	Addition Subtraction	Evaluated next. If there are several operators of this type, they're evaluated from left to right.
`=`	Assignment	Evaluated last.

Fig. 2.12 | Precedence of arithmetic operators.

[1] We use simple examples to explain the order of evaluation of expressions. Subtle issues occur in the more complex expressions you'll encounter later in the book. For more information on order of evaluation, see Chapter 15 of *The Java™ Language Specification* (`java.sun.com/docs/books/jls/`).

Sample Algebraic and Java Expressions

Now let's consider several expressions in light of the rules of operator precedence. Each example lists an algebraic expression and its Java equivalent. The following is an example of an arithmetic mean (average) of five terms:

Algebra: $m = \dfrac{a + b + c + d + e}{5}$

Java:
```
m = ( a + b + c + d + e ) / 5;
```

The parentheses are required because division has higher precedence than addition. The entire quantity `(a + b + c + d + e)` is to be divided by `5`. If the parentheses are erroneously omitted, we obtain `a + b + c + d + e / 5`, which evaluates as

$$a + b + c + d + \frac{e}{5}$$

Here's an example of the equation of a straight line:

Algebra: $y = mx + b$

Java:
```
y = m * x + b;
```

No parentheses are required. The multiplication operator is applied first because multiplication has a higher precedence than addition. The assignment occurs last because it has a lower precedence than multiplication or addition.

The following example contains remainder (%), multiplication, division, addition and subtraction operations:

Algebra: $z = pr\%q + w/x - y$

Java:
```
z  =  p  *  r  %  q  +  w  /  x  -  y;
   6     1     2     4     3     5
```

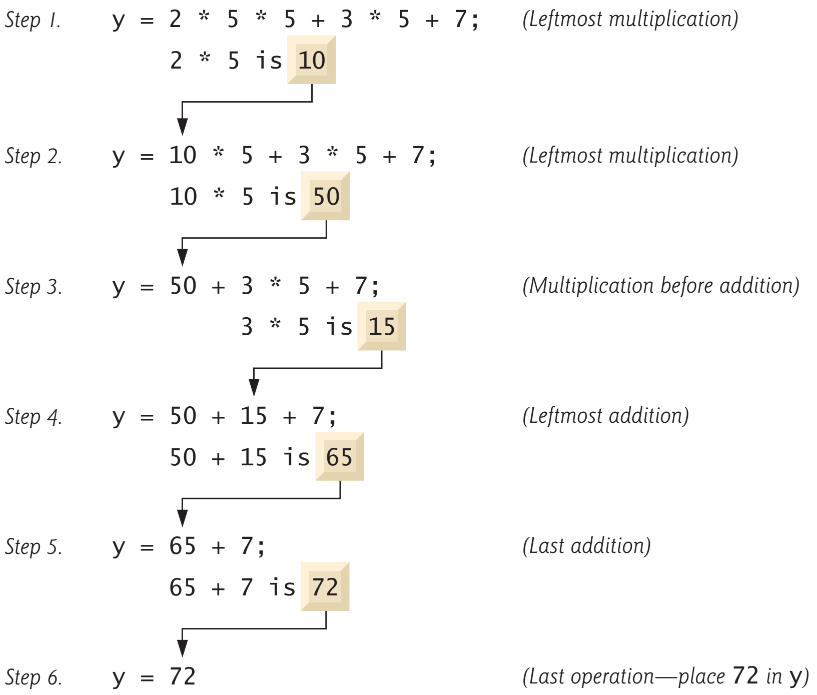
The circled numbers under the statement indicate the order in which Java applies the operators. The `*`, `%` and `/` operations are evaluated first in left-to-right order (i.e., they associate from left to right), because they have higher precedence than `+` and `-`. The `+` and `-` operations are evaluated next. These operations are also applied from left to right. The assignment (`=`) operaton is evaluated last.

Evaluation of a Second-Degree Polynomial

To develop a better understanding of the rules of operator precedence, consider the evaluation of an assignment expression that includes a second-degree polynomial $ax^2 + bx + c$:

```
y  =  a  *  x  *  x  +  b  *  x  +  c;
   6     1     2     4     3     5
```

The multiplication operations are evaluated first in left-to-right order (i.e., they associate from left to right), because they have higher precedence than addition. (Java has no arithmetic operator for exponentiation in Java, so x^2 is represented as `x * x`. Section 5.4 shows an alternative for performing exponentiation.) The addition operations are evaluated next from left to right. Suppose that `a`, `b`, `c` and `x` are initialized (given values) as follows: `a = 2`, `b = 3`, `c = 7` and `x = 5`. Figure 2.13 illustrates the order in which the operators are applied.



```
Step 1.    y = 2 * 5 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
               2 * 5 is 10

Step 2.    y = 10 * 5 + 3 * 5 + 7;         (Leftmost multiplication)
               10 * 5 is 50

Step 3.    y = 50 + 3 * 5 + 7;             (Multiplication before addition)
                    3 * 5 is 15

Step 4.    y = 50 + 15 + 7;                (Leftmost addition)
               50 + 15 is 65

Step 5.    y = 65 + 7;                     (Last addition)
               65 + 7 is 72

Step 6.    y = 72                          (Last operation—place 72 in y)
```

Fig. 2.13 | Order in which a second-degree polynomial is evaluated.

You can use **redundant parentheses** (unnecessary parentheses) to make an expression clearer. For example, the preceding statement might be parenthesized as follows:

```
        y = ( a * x * x ) + ( b * x ) + c;
```

2.8 Decision Making: Equality and Relational Operators

A **condition** is an expression that can be `true` or `false`. This section introduces Java's **`if` selection statement**, which allows a program to make a **decision** based on a condition's value. For example, the condition "grade is greater than or equal to 60" determines whether a student passed a test. If the condition in an `if` statement is true, the body of the `if` statement executes. If the condition is false, the body does not execute. We'll see an example shortly.

Conditions in `if` statements can be formed by using the **equality operators** (`==` and `!=`) and **relational operators** (`>`, `<`, `>=` and `<=`) summarized in Fig. 2.14. Both equality operators have the same level of precedence, which is *lower* than that of the relational operators. The equality operators associate from left to right. The relational operators all have the same level of precedence and also associate from left to right.

Figure 2.15 uses six `if` statements to compare two integers input by the user. If the condition in any of these `if` statements is true, the statement associated with that `if` statement executes; otherwise, the statement is skipped. We use a `Scanner` to input the integers from the user and store them in variables `number1` and `number2`. The program compares the numbers and displays the results of the comparisons that are true.

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	`==`	`x == y`	`x` is equal to `y`
≠	`!=`	`x != y`	`x` is not equal to `y`
Relational operators			
>	`>`	`x > y`	`x` is greater than `y`
<	`<`	`x < y`	`x` is less than `y`
≥	`>=`	`x >= y`	`x` is greater than or equal to `y`
≤	`<=`	`x <= y`	`x` is less than or equal to `y`

Fig. 2.14 | Equality and relational operators.

```java
// Fig. 2.15: Comparison.java
// Compare integers using if statements, relational operators
// and equality operators.
import java.util.Scanner; // program uses class Scanner

public class Comparison
{
   // main method begins execution of Java application
   public static void main( String[] args )
   {
      // create Scanner to obtain input from command line
      Scanner input = new Scanner( System.in );

      int number1; // first number to compare
      int number2; // second number to compare

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      if ( number1 == number2 )
         System.out.printf( "%d == %d\n", number1, number2 );

      if ( number1 != number2 )
         System.out.printf( "%d != %d\n", number1, number2 );

      if ( number1 < number2 )
         System.out.printf( "%d < %d\n", number1, number2 );

```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 1 of 2.)

```
32          if ( number1 > number2 )
33             System.out.printf( "%d > %d\n", number1, number2 );
34
35          if ( number1 <= number2 )
36             System.out.printf( "%d <= %d\n", number1, number2 );
37
38          if ( number1 >= number2 )
39             System.out.printf( "%d >= %d\n", number1, number2 );
40       } // end method main
41    } // end class Comparison
```

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 2 of 2.)

The declaration of class `Comparison` begins at line 6

```
      public class Comparison
```

The class's `main` method (lines 9–40) begins the execution of the program. Line 12

```
      Scanner input = new Scanner( System.in );
```

declares `Scanner` variable `input` and assigns it a `Scanner` that inputs data from the standard input (i.e., the keyboard).

Lines 14–15

```
      int number1; // first number to compare
      int number2; // second number to compare
```

declare the `int` variables used to store the values input from the user.

Lines 17–18

```
      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user
```

prompt the user to enter the first integer and input the value, respectively. The input value is stored in variable `number1`.

Lines 20–21

```
    System.out.print( "Enter second integer: " ); // prompt
    number2 = input.nextInt(); // read second number from user
```

prompt the user to enter the second integer and input the value, respectively. The input value is stored in variable `number2`.

Lines 23–24

```
    if ( number1 == number2 )
       System.out.printf( "%d == %d\n", number1, number2 );
```

compare the values of `number1` and `number2` to determine whether they're equal. An `if` statement always begins with keyword `if`, followed by a condition in parentheses. An `if` statement expects one statement in its body, but may contain multiple statements if they're enclosed in a set of braces (`{}`). The indentation of the body statement shown here is not required, but it improves the program's readability by emphasizing that the statement in line 24 *is part of* the `if` statement that begins at line 23. Line 24 executes only if the numbers stored in variables `number1` and `number2` are equal (i.e., the condition is true). The `if` statements in lines 26–27, 29–30, 32–33, 35–36 and 38–39 compare `number1` and `number2` using the operators `!=`, `<`, `>`, `<=` and `>=`, respectively. If the condition in one or more of the `if` statements is true, the corresponding body statement executes.

Common Programming Error 2.6

Confusing the equality operator, `==`, with the assignment operator, `=`, can cause a logic error or a syntax error. The equality operator should be read as "is equal to" and the assignment operator as "gets" or "gets the value of." To avoid confusion, some people read the equality operator as "double equals" or "equals equals."

Good Programming Practice 2.11

Placing only one statement per line in a program enhances program readability.

There's no semicolon (`;`) at the end of the first line of each `if` statement. Such a semicolon would result in a logic error at execution time. For example,

```
    if ( number1 == number2 ); // logic error
       System.out.printf( "%d == %d\n", number1, number2 );
```

would actually be interpreted by Java as

```
    if ( number1 == number2 )
       ; // empty statement

    System.out.printf( "%d == %d\n", number1, number2 );
```

where the semicolon on the line by itself—called the **empty statement**—is the statement to execute if the condition in the `if` statement is true. When the empty statement executes, no task is performed. The program then continues with the output statement, which always executes, regardless of whether the condition is true or false, because the output statement is not part of the `if` statement.

Common Programming Error 2.7

Placing a semicolon immediately after the right parenthesis of the condition in an `if` statement is normally a logic error.

Note the use of white space in Fig. 2.15. Recall that the compiler normally ignores white space. So, statements may be split over several lines and may be spaced according to your preferences without affecting a program's meaning. It's incorrect to split identifiers and strings. Ideally, statements should be kept small, but this is not always possible.

Error-Prevention Tip 2.9

A lengthy statement can be spread over several lines. If a single statement must be split across lines, choose breaking points that make sense, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines until the end of the statement.

Figure 2.16 shows the operators discussed so far in decreasing order of precedence. All but the assignment operator, `=`, associate from left to right. The assignment operator, `=`, associates from right to left, so an expression like `x = y = 0` is evaluated as if it had been written as `x = (y = 0)`, which first assigns the value `0` to variable `y`, then assigns the result of that assignment, `0`, to `x`.

Good Programming Practice 2.12

When writing expressions containing many operators, refer to the operator precedence chart (Appendix A) . Confirm that the operations in the expression are performed in the order you expect. If, in a complex expression, you're uncertain about the order of evaluation, use parentheses to force the order, exactly as you'd do in algebraic expressions.

Operators	Associativity	Type
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`=`	right to left	assignment

Fig. 2.16 | Precedence and associativity of operators discussed.

2.9 Wrap-Up

In this chapter, you learned many important features of Java, including displaying data on the screen in a **Command Prompt**, inputting data from the keyboard, performing calculations and making decisions. The applications presented here introduced you to basic programming concepts. As you'll see in Chapter 3, Java applications typically contain just a few lines of code in method `main`—these statements normally create the objects that perform the work of the application. In Chapter 3, you'll learn how to implement your own classes and use objects of those classes in applications.

Summary

Section 2.2 Your First Program in Java: Printing a Line of Text

- A Java application (p. 38) executes when you use the `java` command to launch the JVM.

- Comments (p. 39) document programs and improve their readability. The compiler ignores them.
- A comment that begins with `//` is an end-of-line comment—it terminates at the end of the line on which it appears.
- Traditional comments (p. 39) can be spread over several lines and are delimited by `/*` and `*/`.
- Javadoc comments (p. 39), delimited by `/**` and `*/`, enable you to embed program documentation in your code. The `javadoc` utility program generates HTML pages based on these comments.
- A syntax error (p. 39; also called a compiler error, compile-time error or compilation error) occurs when the compiler encounters code that violates Java's language rules. It's similar to a grammar error in a natural language.
- Blank lines, space characters and tab characters are known as white space (p. 39). White space makes programs easier to read and is ignored by the compiler.
- Keywords (p. 40) are reserved for use by Java and are always spelled with all lowercase letters.
- Keyword `class` (p. 40) introduces a class declaration.
- By convention, all class names in Java begin with a capital letter and capitalize the first letter of each word they include (e.g., `SampleClassName`).
- A Java class name is an identifier—a series of characters consisting of letters, digits, underscores (`_`) and dollar signs (`$`) that does not begin with a digit and does not contain spaces.
- Java is case sensitive (p. 40)—that is, uppercase and lowercase letters are distinct.
- The body of every class declaration (p. 40) is delimited by braces, `{` and `}`.
- A `public` (p. 40) class declaration must be saved in a file with the same name as the class followed by the "`.java`" file-name extension.
- Method `main` (p. 41) is the starting point of every Java application and must begin with

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

  otherwise, the JVM will not execute the application.
- Methods perform tasks and return information when they complete them. Keyword `void` (p. 41) indicates that a method will perform a task but return no information.
- Statements instruct the computer to perform actions.
- A string (p. 41) in double quotes is sometimes called a character string or a string literal.
- The standard output object (`System.out`; p. 41) displays characters in the command window.
- Method `System.out.println` (p. 41) displays its argument (p. 41) in the command window followed by a newline character to position the output cursor to the beginning of the next line.
- You compile a program with the command `javac`. If the program contains no syntax errors, a class file (p. 43) containing the Java bytecodes that represent the application is created. These bytecodes are interpreted by the JVM when you execute the program.
- To run an application, type `java` (p. 38) followed by the name of the class that contains `main`.

Section 2.3 Modifying Your First Java Program

- `System.out.print` (p. 44) displays its argument and positions the output cursor immediately after the last character displayed.
- A backslash (`\`) in a string is an escape character (p. 45). Java combines it with the next character to form an escape sequence (p. 45). The escape sequence `\n` (p. 45) represents the newline character.

Section 2.4 Displaying Text with `printf`

- `System.out.printf` method (p. 46; `f` means "formatted") displays formatted data.

- Method printf's first argument is a format string (p. 47) containing fixed text and/or format specifiers. Each format specifier (p. 47) indicates the type of data to output and is a placeholder for a corresponding argument that appears after the format string.
- Format specifiers begin with a percent sign (%) and are followed by a character that represents the data type. The format specifier %s (p. 47) is a placeholder for a string.

Section 2.5 Another Application: Adding Integers

- An import declaration (p. 48) helps the compiler locate a class that's used in a program.
- Java's rich set of predefined classes are grouped into packages (p. 48)—named groups of classes. These are referred to as the Java class library (p. 48), or the Java Application Programming Interface (Java API).
- A variable (p. 49) is a location in the computer's memory where a value can be stored for use later in a program. All variables must be declared with a name and a type before they can be used.
- A variable's name enables the program to access the variable's value in memory.
- A Scanner (package java.util; p. 49) enables a program to read data that the program will use. Before a Scanner can be used, the program must create it and specify the source of the data.
- Variables should be initialized (p. 49) to prepare them for use in a program.
- The expression new Scanner(System.in) creates a Scanner that reads from the standard input object (System.in; p. 49)—normally the keyboard.
- Data type int (p. 49) is used to declare variables that will hold integer values. The range of values for an int is –2,147,483,648 to +2,147,483,647.
- Types float and double (p. 49) specify real numbers with decimal points, such as 3.4 and -11.19.
- Variables of type char (p. 49) represent individual characters, such as an uppercase letter (e.g., A), a digit (e.g., 7), a special character (e.g., * or %) or an escape sequence (e.g., newline, \n).
- Types such as int, float, double and char are primitive types (p. 49). Primitive-type names are keywords; thus, they must appear in all lowercase letters.
- A prompt (p. 50) directs the user to take a specific action.
- Scanner method nextInt obtains an integer for use in a program.
- The assignment operator, = (p. 51), enables the program to give a value to a variable. It's called a binary operator (p. 51) because it has two operands.
- Portions of statements that have values are called expressions (p. 51).
- The format specifier %d (p. 51) is a placeholder for an int value.

Section 2.6 Memory Concepts

- Variable names (p. 52) correspond to locations in the computer's memory. Every variable has a name, a type, a size and a value.
- A value that's placed in a memory location replaces the location's previous value, which is lost.

Section 2.7 Arithmetic

- The arithmetic operators (p. 53) are + (addition), - (subtraction), * (multiplication), / (division) and % (remainder).
- Integer division (p. 53) yields an integer quotient.
- The remainder operator, % (p. 53), yields the remainder after division.
- Arithmetic expressions must be written in straight-line form (p. 54).
- If an expression contains nested parentheses (p. 54), the innermost set is evaluated first.

- Java applies the operators in arithmetic expressions in a precise sequence determined by the rules of operator precedence (p. 54).
- When we say that operators are applied from left to right, we're referring to their associativity (p. 54). Some operators associate from right to left.
- Redundant parentheses (p. 56) can make an expression clearer.

Section 2.8 Decision Making: Equality and Relational Operators

- The `if` statement (p. 56) makes a decision based on a condition's value (true or false).
- Conditions in `if` statements can be formed by using the equality (`==` and `!=`) and relational (`>`, `<`, `>=` and `<=`) operators (p. 56).
- An `if` statement begins with keyword `if` followed by a condition in parentheses and expects one statement in its body.
- The empty statement (p. 59) is a statement that does not perform a task.

Self-Review Exercises

2.1 Fill in the blanks in each of the following statements:

a) A(n) ________ begins the body of every method, and a(n) ________ ends the body of every method.

b) The ________ statement is used to make decisions.

c) ________ begins an end-of-line comment.

d) ________, ________ and ________ are called white space.

e) ________ are reserved for use by Java.

f) Java applications begin execution at method ________.

g) Methods ________, ________ and ________ display information in a command window.

2.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Comments cause the computer to print the text after the `//` on the screen when the program executes.

b) All variables must be given a type when they're declared.

c) Java considers the variables `number` and `NuMbEr` to be identical.

d) The remainder operator (`%`) can be used only with integer operands.

e) The arithmetic operators `*`, `/`, `%`, `+` and `-` all have the same level of precedence.

2.3 Write statements to accomplish each of the following tasks:

a) Declare variables `c`, `thisIsAVariable`, `q76354` and `number` to be of type `int`.

b) Prompt the user to enter an integer.

c) Input an integer and assign the result to `int` variable `value`. Assume `Scanner` variable `input` can be used to read a value from the keyboard.

d) Print `"This is a Java program"` on one line in the command window. Use method `System.out.println`.

e) Print `"This is a Java program"` on two lines in the command window. The first line should end with `Java`. Use method `System.out.println`.

f) Print `"This is a Java program"` on two lines in the command window. The first line should end with `Java`. Use method `System.out.printf` and two `%s` format specifiers.

g) If the variable `number` is not equal to `7`, display `"The variable number is not equal to 7"`.

2.4 Identify and correct the errors in each of the following statements:

a)
```
if ( c < 7 );
   System.out.println( "c is less than 7" );
```

b)
```
if ( c => 7 )
   System.out.println( "c is equal to or greater than 7" );
```

2.5 Write declarations, statements or comments that accomplish each of the following tasks:

a) State that a program will calculate the product of three integers.
b) Create a `Scanner` called `input` that reads values from the standard input.
c) Declare the variables `x`, `y`, `z` and `result` to be of type `int`.
d) Prompt the user to enter the first integer.
e) Read the first integer from the user and store it in the variable `x`.
f) Prompt the user to enter the second integer.
g) Read the second integer from the user and store it in the variable `y`.
h) Prompt the user to enter the third integer.
i) Read the third integer from the user and store it in the variable `z`.
j) Compute the product of the three integers contained in variables `x`, `y` and `z`, and assign the result to the variable `result`.
k) Display the message `"Product is"` followed by the value of the variable `result`.

2.6 Using the statements you wrote in Exercise 2.5, write a complete program that calculates and prints the product of three integers.

Answers to Self-Review Exercises

2.1 a) left brace (`{`), right brace (`}`). b) `if`. c) `//`. d) Space characters, newlines and tabs. e) Keywords. f) `main`. g) `System.out.print`, `System.out.println` and `System.out.printf`.

2.2 a) False. Comments do not cause any action to be performed when the program executes. They're used to document programs and improve their readability.
b) True.
c) False. Java is case sensitive, so these variables are distinct.
d) False. The remainder operator can also be used with noninteger operands in Java.
e) False. The operators `*`, `/` and `%` are higher precedence than operators `+` and `-`.

2.3 a)

```
int c, thisIsAVariable, q76354, number;
```

or

```
int c;
int thisIsAVariable;
int q76354;
int number;
```

b) `System.out.print( "Enter an integer: " );`
c) `value = input.nextInt();`
d) `System.out.println( "This is a Java program" );`
e) `System.out.println( "This is a Java\nprogram" );`
f) `System.out.printf( "%s\n%s\n", "This is a Java", "program" );`
g)

```
if ( number != 7 )
   System.out.println( "The variable number is not equal to 7" );
```

2.4 a) Error: Semicolon after the right parenthesis of the condition `( c < 7 )` in the `if`. Correction: Remove the semicolon after the right parenthesis. [*Note:* As a result, the output statement will execute regardless of whether the condition in the `if` is true.]
b) Error: The relational operator `=>` is incorrect. Correction: Change `=>` to `>=`.

2.5 a) `// Calculate the product of three integers`
b) `Scanner input = new Scanner( System.in );`
c) `int x, y, z, result;`
or

```
int x;
int y;
int z;
int result;
```

d) `System.out.print( "Enter first integer: " );`

e) `x = input.nextInt();`

f) `System.out.print( "Enter second integer: " );`

g) `y = input.nextInt();`

h) `System.out.print( "Enter third integer: " );`

i) `z = input.nextInt();`

j) `result = x * y * z;`

k) `System.out.printf( "Product is %d\n", result );`

2.6 The solution to Self-Review Exercise 2.6 is as follows:

```java
// Ex. 2.6: Product.java
// Calculate the product of three integers.
import java.util.Scanner; // program uses Scanner

public class Product
{
   public static void main( String[] args )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int x; // first number input by user
      int y; // second number input by user
      int z; // third number input by user
      int result; // product of numbers

      System.out.print( "Enter first integer: " ); // prompt for input
      x = input.nextInt(); // read first integer

      System.out.print( "Enter second integer: " ); // prompt for input
      y = input.nextInt(); // read second integer

      System.out.print( "Enter third integer: " ); // prompt for input
      z = input.nextInt(); // read third integer

      result = x * y * z; // calculate product of numbers

      System.out.printf( "Product is %d\n", result );
   } // end method main
} // end class Product
```

```
Enter first integer: 10
Enter second integer: 20
Enter third integer: 30
Product is 6000
```

Exercises

2.7 Fill in the blanks in each of the following statements:

a) ________ are used to document a program and improve its readability.

b) A decision can be made in a Java program with a(n) ________.

c) Calculations are normally performed by ________ statements.
d) The arithmetic operators with the same precedence as multiplication are ________ and ________.
e) When parentheses in an arithmetic expression are nested, the ________ set of parentheses is evaluated first.
f) A location in the computer's memory that may contain different values at various times throughout the execution of a program is called a(n) ________.

2.8 Write Java statements that accomplish each of the following tasks:
a) Display the message `"Enter an integer: "`, leaving the cursor on the same line.
b) Assign the product of variables `b` and `c` to variable `a`.
c) Use a comment to state that a program performs a sample payroll calculation.

2.9 State whether each of the following is *true* or *false*. If *false*, explain why.
a) Java operators are evaluated from left to right.
b) The following are all valid variable names: `_under_bar_`, `m928134`, `t5`, `j7`, `her_sales$`, `his_$account_total`, `a`, `b$`, `c`, `z` and `z2`.
c) A valid Java arithmetic expression with no parentheses is evaluated from left to right.
d) The following are all invalid variable names: `3g`, `87`, `67h2`, `h22` and `2h`.

2.10 Assuming that `x = 2` and `y = 3`, what does each of the following statements display?
a) `System.out.printf( "x = %d\n", x );`
b) `System.out.printf( "Value of %d + %d is %d\n", x, x, ( x + x ) );`
c) `System.out.printf( "x =" );`
d) `System.out.printf( "%d = %d\n", ( x + y ), ( y + x ) );`

2.11 Which of the following Java statements contain variables whose values are modified?
a) `p = i + j + k + 7;`
b) `System.out.println( "variables whose values are modified" );`
c) `System.out.println( "a = 5" );`
d) `value = input.nextInt();`

2.12 Given that $y = ax^3 + 7$, which of the following are correct Java statements for this equation?
a) `y = a * x * x * x + 7;`
b) `y = a * x * x * ( x + 7 );`
c) `y = ( a * x ) * x * ( x + 7 );`
d) `y = ( a * x ) * x * x + 7;`
e) `y = a * ( x * x * x ) + 7;`
f) `y = a * x * ( x * x + 7 );`

2.13 State the order of evaluation of the operators in each of the following Java statements, and show the value of `x` after each statement is performed:
a) `x = 7 + 3 * 6 / 2 - 1;`
b) `x = 2 % 2 + 2 * 2 - 2 / 2;`
c) `x = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) );`

2.14 Write an application that displays the numbers 1 to 4 on the same line, with each pair of adjacent numbers separated by one space. Use the following techniques:
a) Use one `System.out.println` statement.
b) Use four `System.out.print` statements.
c) Use one `System.out.printf` statement.

2.15 *(Arithmetic)* Write an application that asks the user to enter two integers, obtains them from the user and prints their sum, product, difference and quotient (division). Use the techniques shown in Fig. 2.7.

2.16 *(Comparing Integers)* Write an application that asks the user to enter two integers, obtains them from the user and displays the larger number followed by the words `"is larger"`. If the numbers are equal, print the message `"These numbers are equal"`. Use the techniques shown in Fig. 2.15.

2.17 *(Arithmetic, Smallest and Largest)* Write an application that inputs three integers from the user and displays the sum, average, product, smallest and largest of the numbers. Use the techniques shown in Fig. 2.15. [*Note:* The calculation of the average in this exercise should result in an integer representation of the average. So, if the sum of the values is 7, the average should be 2, not 2.3333….]

2.18 *(Displaying Shapes with Asterisks)* Write an application that displays a box, an oval, an arrow and a diamond using asterisks (*), as follows:

```
*********     ***       *        *
*       *   *     *    ***      * *
*       *  *       *  *****    *   *
*       *  *       *    *     *     *
*       *  *       *    *    *       *
*       *  *       *    *     *     *
*       *  *       *    *      *   *
*       *   *     *     *       * *
*********     ***       *        *
```

2.19 What does the following code print?

```
System.out.println( "*\n**\n***\n****\n*****" );
```

2.20 What does the following code print?

```
System.out.println( "*" );
System.out.println( "***" );
System.out.println( "*****" );
System.out.println( "****" );
System.out.println( "**" );
```

2.21 What does the following code print?

```
System.out.print( "*" );
System.out.print( "***" );
System.out.print( "*****" );
System.out.print( "****" );
System.out.println( "**" );
```

2.22 What does the following code print?

```
System.out.print( "*" );
System.out.println( "***" );
System.out.println( "*****" );
System.out.print( "****" );
System.out.println( "**" );
```

2.23 What does the following code print?

```
System.out.printf( "%s\n%s\n%s\n", "*", "***", "*****" );
```

2.24 *(Largest and Smallest Integers)* Write an application that reads five integers and determines and prints the largest and smallest integers in the group. Use only the programming techniques you learned in this chapter.

2.25 *(Odd or Even)* Write an application that reads an integer and determines and prints whether it's odd or even. [*Hint:* Use the remainder operator. An even number is a multiple of 2. Any multiple of 2 leaves a remainder of 0 when divided by 2.]

2.26 *(Multiples)* Write an application that reads two integers, determines whether the first is a multiple of the second and prints the result. [*Hint:* Use the remainder operator.]

2.27 *(Checkerboard Pattern of Asterisks)* Write an application that displays a checkerboard pattern, as follows:

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

2.28 *(Diameter, Circumference and Area of a Circle)* Here's a peek ahead. In this chapter, you learned about integers and the type `int`. Java can also represent floating-point numbers that contain decimal points, such as 3.14159. Write an application that inputs from the user the radius of a circle as an integer and prints the circle's diameter, circumference and area using the floating-point value 3.14159 for π. Use the techniques shown in Fig. 2.7. [*Note:* You may also use the predefined constant `Math.PI` for the value of π. This constant is more precise than the value 3.14159. Class `Math` is defined in package `java.lang`. Classes in that package are imported automatically, so you do not need to `import` class `Math` to use it.] Use the following formulas (r is the radius):

$$\begin{aligned}
&\mathit{diameter} = 2r\\
&\mathit{circumference} = 2\pi r\\
&\mathit{area} = \pi r^2
\end{aligned}$$

Do not store the results of each calculation in a variable. Rather, specify each calculation as the value that will be output in a `System.out.printf` statement. The values produced by the circumference and area calculations are floating-point numbers. Such values can be output with the format specifier `%f` in a `System.out.printf` statement. You'll learn more about floating-point numbers in Chapter 3.

2.29 *(Integer Value of a Character)* Here's another peek ahead. In this chapter, you learned about integers and the type `int`. Java can also represent uppercase letters, lowercase letters and a considerable variety of special symbols. Every character has a corresponding integer representation. The set of characters a computer uses together with the corresponding integer representations for those characters is called that computer's character set. You can indicate a character value in a program simply by enclosing that character in single quotes, as in `'A'`.

You can determine a character's integer equivalent by preceding that character with `(int)`, as in

```java
(int) 'A'
```

An operator of this form is called a cast operator. (You'll learn about cast operators in Chapter 4.) The following statement outputs a character and its integer equivalent:

```java
System.out.printf(
   "The character %c has the value %d\n", 'A', ( (int) 'A' ) );
```

When the preceding statement executes, it displays the character `A` and the value `65` (from the Unicode® character set) as part of the string. The format specifier `%c` is a placeholder for a character (in this case, the character `'A'`).

Using statements similar to the one shown earlier in this exercise, write an application that displays the integer equivalents of some uppercase letters, lowercase letters, digits and special symbols. Display the integer equivalents of the following: `A B C a b c 0 1 2 $ * + /` and the blank character.

2.30 *(Separating the Digits in an Integer)* Write an application that inputs one number consisting of five digits from the user, separates the number into its individual digits and prints the digits separated from one another by three spaces each. For example, if the user types in the number `42339`, the program should print

```
4   2   3   3   9
```

Assume that the user enters the correct number of digits. What happens when you execute the program and type a number with more than five digits? What happens when you execute the program and type a number with fewer than five digits? [*Hint:* It's possible to do this exercise with the techniques you learned in this chapter. You'll need to use both division and remainder operations to "pick off" each digit.]

2.31 *(Table of Squares and Cubes)* Using only the programming techniques you learned in this chapter, write an application that calculates the squares and cubes of the numbers from 0 to 10 and prints the resulting values in table format, as shown below. [*Note:* This program does not require any input from the user.]

```
number  square  cube
0       0       0
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
```

2.32 *(Negative, Positive and Zero Values)* Write a program that inputs five numbers and determines and prints the number of negative numbers input, the number of positive numbers input and the number of zeros input.

Making a Difference

2.33 *(Body Mass Index Calculator)* We introduced the body mass index (BMI) calculator in Exercise 1.10. The formulas for calculating BMI are

$$BMI = \frac{weightInPounds \times 703}{heightInInches \times heightInInches}$$

or

$$BMI = \frac{weightInKilograms}{heightInMeters \times heightInMeters}$$

Create a BMI calculator that reads the user's weight in pounds and height in inches (or, if you prefer, the user's weight in kilograms and height in meters), then calculates and displays the user's body mass index. Also, display the following information from the Department of Health and Human Services/National Institutes of Health so the user can evaluate his/her BMI:

```
BMI VALUES
Underweight: less than 18.5
Normal:      between 18.5 and 24.9
Overweight:  between 25 and 29.9
Obese:       30 or greater
```

[*Note:* In this chapter, you learned to use the `int` type to represent whole numbers. The BMI calculations when done with `int` values will both produce whole-number results. In Chapter 3 you'll learn to use the `double` type to represent numbers with decimal points. When the BMI calculations are performed with `double`s, they'll both produce numbers with decimal points—these are called "floating-point" numbers.]

2.34 *(World Population Growth Calculator)* Use the web to determine the current world population and the annual world population growth rate. Write an application that inputs these values, then displays the estimated world population after one, two, three, four and five years.

2.35 *(Car-Pool Savings Calculator)* Research several car-pooling websites. Create an application that calculates your daily driving cost, so that you can estimate how much money could be saved by car pooling, which also has other advantages such as reducing carbon emissions and reducing traffic congestion. The application should input the following information and display the user's cost per day of driving to work:

a) Total miles driven per day.
b) Cost per gallon of gasoline.
c) Average miles per gallon.
d) Parking fees per day.
e) Tolls per day.