CHAPTER

2

ELEMENTARY PROGRAMMING

Objectives

- To write Java programs to perform simple computations (§2.2).
- To obtain input from the console using the `Scanner` class (§2.3).
- To use identifiers to name variables, constants, methods, and classes (§2.4).
- To use variables to store data (§§2.5–2.6).
- To program with assignment statements and assignment expressions (§2.6).
- To use constants to store permanent data (§2.7).
- To name classes, methods, variables, and constants by following their naming conventions (§2.8).
- To explore Java numeric primitive data types: `byte`, `short`, `int`, `long`, `float`, and `double` (§2.9.1).
- To perform operations using operators `+`, `-`, `*`, `/`, and `%` (§2.9.2).
- To perform exponent operations using `Math.pow(a, b)` (§2.9.3).
- To write integer literals, floating-point literals, and literals in scientific notation (§2.10).
- To write and evaluate numeric expressions (§2.11).
- To obtain the current system time using `System.currentTimeMillis()` (§2.12).
- To use augmented assignment operators (§2.13).
- To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.14).
- To cast the value of one type to another type (§2.15).
- To describe the software development process and apply it to develop the loan payment program (§2.16).
- To represent characters using the `char` type (§2.17).
- To represent a string using the `String` type (§2.18).
- To obtain input using the `JOptionPane` input dialog boxes (§2.19).

2.1 Introduction

The focus of this chapter is on learning elementary programming techniques to solve problems.

In Chapter 1 you learned how to create, compile, and run very basic Java programs. Now you will learn how to solve problems by writing programs. Through these problems, you will learn elementary programming using primitive data types, variables, constants, operators, expressions, and input and output.

Suppose, for example, that you need to take out a student loan. Given the loan amount, loan term, and annual interest rate, can you write a program to compute the monthly payment and total payment? This chapter shows you how to write programs like this. Along the way, you learn the basic steps that go into analyzing a problem, designing a solution, and implementing the solution by creating a program.

2.2 Writing a Simple Program

Writing a program involves designing a strategy for solving the problem and then using a programming language to implement that strategy.

problem

Let's first consider the simple problem of computing the area of a circle. How do we write a program for solving this problem?

algorithm

pseudocode

Writing a program involves designing algorithms and translating algorithms into programming instructions, or code. An *algorithm* describes how a problem is solved by listing the actions that need to be taken and the order of their execution. Algorithms can help the programmer plan a program before writing it in a programming language. Algorithms can be described in natural languages or in *pseudocode* (natural language mixed with some programming code). The algorithm for calculating the area of a circle can be described as follows:

1. Read in the circle's radius.
2. Compute the area using the following formula:

$$area = radius \times radius \times \pi$$

3. Display the result.

Tip

It's always good practice to outline your program (or its underlying problem) in the form of an algorithm before you begin coding.

When you *code*—that is, when you write a program—you translate an algorithm into a program. You already know that every Java program begins with a class definition in which the keyword `class` is followed by the class name. Assume that you have chosen `ComputeArea` as the class name. The outline of the program would look like this:

```
public class ComputeArea {
  // Details to be given later
}
```

As you know, every Java program must have a `main` method where program execution begins. The program is then expanded as follows:

```
public class ComputeArea {
  public static void main(String[] args) {
    // Step 1: Read in radius

    // Step 2: Compute area
```

To fix the error, break the string into separate substrings, and use the concatenation operator (+) to combine them:

break a long string

```
System.out.println("Introduction to Java Programming, " +
  "by Y. Daniel Liang");
```

Tip

This example consists of three steps. It is a good approach to develop and test these steps incrementally by adding them one at a time.

incremental development and testing

2.1 Identify and fix the errors in the following code:

MyProgrammingLab™

```
1  public class Test {
2    public void main(string[] args) {
3      int i;
4      int k = 100.0;
5      int j = i + 1;
6
7      System.out.println("j is " + j + " and
8        k is " + k);
9    }
10 }
```

2.3 Reading Input from the Console

Reading input from the console enables the program to accept input from the user.

In Listing 2.1, the radius is fixed in the source code. To use a different radius, you have to modify the source code and recompile it. Obviously, this is not convenient, so instead you can use the `Scanner` class for console input.

VideoNote
Obtain input

Java uses `System.out` to refer to the standard output device and `System.in` to the standard input device. By default, the output device is the display monitor and the input device is the keyboard. To perform console output, you simply use the `println` method to display a primitive value or a string to the console. Console input is not directly supported in Java, but you can use the `Scanner` class to create an object to read input from `System.in`, as follows:

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

The syntax `new Scanner(System.in)` creates an object of the `Scanner` type. The syntax `Scanner input` declares that `input` is a variable whose type is `Scanner`. The whole line `Scanner input = new Scanner(System.in)` creates a `Scanner` object and assigns its reference to the variable `input`. An object may invoke its methods. To invoke a method on an object is to ask the object to perform a task. You can invoke the methods listed in Table 2.1 to read various types of input.

For now, we will see how to read a number that includes a decimal point by invoking the `nextDouble()` method. Other methods will be covered when they are used. Listing 2.2 rewrites Listing 2.1 to prompt the user to enter a radius.

LISTING 2.2 ComputeAreaWithConsoleInput.java

```
1  import java.util.Scanner; // Scanner is in the java.util package
2
3  public class ComputeAreaWithConsoleInput {
4    public static void main(String[] args) {
5      // Create a Scanner object
6      Scanner input = new Scanner(System.in);
7
```

import class

create a Scanner

Check Point

MyProgrammingLab™

2.2 How do you write a statement to let the user enter an integer or a double value from the keyboard?

2.3 What happens if you entered `5a` when executing the following code?

```
double radius = input.nextDouble();
```

2.4 Identifiers

Key Point

Identifiers are the names that identify the elements such as classes, methods, and variables in a program.

identifiers

identifier naming rules

As you see in Listing 2.3, `ComputeAverage`, `main`, `input`, `number1`, `number2`, `number3`, and so on are the names of things that appear in the program. In programming terminology, such names are called *identifiers*. All identifiers must obey the following rules:

- An identifier is a sequence of characters that consists of letters, digits, underscores (`_`), and dollar signs (`$`).
- An identifier must start with a letter, an underscore (`_`), or a dollar sign (`$`). It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A for a list of reserved words.)
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.

For example, `$2`, `ComputeArea`, `area`, `radius`, and `showMessageDialog` are legal identifiers, whereas `2A` and `d+4` are not because they do not follow the rules. The Java compiler detects illegal identifiers and reports syntax errors.

case sensitive

Note

Since Java is case sensitive, `area`, `Area`, and `AREA` are all different identifiers.

descriptive names

Tip

Identifiers are for naming variables, constants, methods, classes, and packages. Descriptive identifiers make programs easy to read. Avoid using abbreviations for identifiers. Using complete words is more descriptive. For example, `numberOfStudents` is better than `numStuds`, `numOfStuds`, or `numOfStudents`. We use descriptive names for complete programs in the text. However, we will occasionally use variables names such as `i`, `j`, `k`, `x`, and `y` in the code snippets for brevity. These names also provide a generic tone to the code snippets.

the $ character

Tip

Do not name identifiers with the `$` character. By convention, the `$` character should be used only in mechanically generated source code.

Check Point

MyProgrammingLab™

2.4 Which of the following identifiers are valid? Which are Java keywords?

`miles`, `Test`, `a++`, `--a`, `4#R`, `$4`, `#44`, `apps`
`class`, `public`, `int`, `x`, `y`, `radius`

2.5 Variables

Key Point

Variables are used to represent values that may be changed in the program.

why called variables?

As you see from the programs in the preceding sections, variables are used to store values to be used later in a program. They are called variables because their values can be changed. In

scope of a variable

Every variable has a scope. The *scope of a variable* is the part of the program where the variable can be referenced. The rules that define the scope of a variable will be introduced gradually later in the book. For now, all you need to know is that a variable must be declared and initialized before it can be used. Consider the following code:

```
int interestRate = 0.05
int interest = interestrate * 45
```

This code is wrong, because `interestRate` is assigned a value `0.05`, but `interestrate` has not been declared and initialized. Java is case sensitive, so it considers `interestRate` and `interestrate` to be two different variables.

2.6 Assignment Statements and Assignment Expressions

An assignment statement designates a value for a variable. An assignment statement can be used as an expression in Java.

assignment statement
assignment operator

After a variable is declared, you can assign a value to it by using an *assignment statement.* In Java, the equal sign (`=`) is used as the *assignment operator.* The syntax for assignment statements is as follows:

```
variable = expression;
```

expression

An *expression* represents a computation involving values, variables, and operators that, taking them together, evaluates to a value. For example, consider the following code:

```
int y = 1;              // Assign 1 to variable y
double radius = 1.0;    // Assign 1.0 to variable radius
int x = 5 * (3 / 2);    // Assign the value of the expression to x
x = y + 1;              // Assign the addition of y and 1 to x
area = radius * radius * 3.14159; // Compute area
```

You can use a variable in an expression. A variable can also be used in both sides of the = operator. For example,

```
x = x + 1;
```

In this assignment statement, the result of `x + 1` is assigned to `x`. If `x` is `1` before the statement is executed, then it becomes `2` after the statement is executed.

To assign a value to a variable, you must place the variable name to the left of the assignment operator. Thus, the following statement is wrong:

```
1 = x; // Wrong
```

Note

In mathematics, `x = 2 * x + 1` denotes an equation. However, in Java, `x = 2 * x + 1` is an assignment statement that evaluates the expression `2 * x + 1` and assigns the result to `x`.

assignment expression

In Java, an assignment statement is essentially an expression that evaluates to the value to be assigned to the variable on the left side of the assignment operator. For this reason, an assignment statement is also known as an *assignment expression.* For example, the following statement is correct:

```
System.out.println(x = 1);
```

which is equivalent to

```
x = 1;
System.out.println(x);
```

If a value is assigned to multiple variables, you can use this syntax:

```
i = j = k = 1;
```

which is equivalent to

```
k = 1;
j = k;
i = j;
```

Note

In an assignment statement, the data type of the variable on the left must be compatible with the data type of the value on the right. For example, `int x = 1.0` would be illegal, because the data type of `x` is `int`. You cannot assign a `double` value (`1.0`) to an `int` variable without using type casting. Type casting is introduced in Section 2.15.

2.7 Named Constants

A named constant is an identifier that represents a permanent value.

constant

The value of a variable may change during the execution of a program, but a *named constant,* or simply *constant,* represents permanent data that never changes. In our `ComputeArea` program, π is a constant. If you use it frequently, you don't want to keep typing `3.14159`; instead, you can declare a constant for π. Here is the syntax for declaring a constant:

```
final datatype CONSTANTNAME = value;
```

final keyword

A constant must be declared and initialized in the same statement. The word `final` is a Java keyword for declaring a constant. For example, you can declare π as a constant and rewrite Listing 2.1 as follows:

```
// ComputeArea.java: Compute the area of a circle
public class ComputeArea {
  public static void main(String[] args) {
    final double PI = 3.14159; // Declare a constant

    // Assign a radius
    double radius = 20;

    // Compute area
    double area = radius * radius * PI;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

benefits of constants

There are three benefits of using constants: (1) You don't have to repeatedly type the same value if it is used multiple times; (2) if you have to change the constant value (e.g., from `3.14` to `3.14159` for `PI`), you need to change it only in a single location in the source code; and (3) a descriptive name for a constant makes the program easy to read.

2.8 Naming Conventions

Sticking with the Java naming conventions makes your programs easy to read and avoids errors.

Make sure that you choose descriptive names with straightforward meanings for the variables, constants, classes, and methods in your program. As mentioned earlier, names are case sensitive. Listed below are the conventions for naming variables, methods, and classes.

name variables and methods

- Use lowercase for variables and methods. If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word—for example, the variables `radius` and `area` and the method `showMessageDialog`.

name classes

- Capitalize the first letter of each word in a class name—for example, the class names `ComputeArea`, `System`, and `JOptionPane`.

name constants

- Capitalize every letter in a constant, and use underscores between words—for example, the constants `PI` and `MAX_VALUE`.

It is important to follow the naming conventions to make your programs easy to read.

name classes

Caution

Do not choose class names that are already used in the Java library. For example, since the `System` class is defined in Java, you should not name your class `System`.

MyProgrammingLab™

2.5 What are the benefits of using constants? Declare an `int` constant `SIZE` with value `20`.

2.6 What are the naming conventions for class names, method names, constants, and variables? Which of the following items can be a constant, a method, a variable, or a class according to the Java naming conventions?

```
MAX_VALUE, Test, read, readInt
```

2.7 Translate the following algorithm into Java code:

Step 1: Declare a `double` variable named `miles` with initial value `100`.

Step 2: Declare a `double` constant named `KILOMETERS_PER_MILE` with value `1.609`.

Step 3: Declare a `double` variable named `kilometers`, multiply `miles` and `KILOMETERS_PER_MILE`, and assign the result to `kilometers`.

Step 4: Display `kilometers` to the console.

What is kilometers after Step 4?

2.9 Numeric Data Types and Operations

Java has six numeric types for integers and floating-point numbers with operators `+`, `-`, `*`, `/`, and `%`.

2.9.1 Numeric Types

Every data type has a range of values. The compiler allocates memory space for each variable or constant according to its data type. Java provides eight primitive data types for numeric values, characters, and Boolean values. This section introduces numeric data types and operators.

Table 2.2 lists the six numeric data types, their ranges, and their storage sizes.

2.9.3 Exponent Operations

`Math.pow(a, b)` method

The `Math.pow(a, b)` method can be used to compute a^b. The `pow` method is defined in the `Math` class in the Java API. You invoke the method using the syntax `Math.pow(a, b)` (i.e., `Math.pow(2, 3)`), which returns the result of a^b (2^3). Here `a` and `b` are parameters for the `pow` method and the numbers `2` and `3` are actual values used to invoke the method. For example,

```
System.out.println(Math.pow(2, 3)); // Displays 8.0
System.out.println(Math.pow(4, 0.5)); // Displays 2.0
System.out.println(Math.pow(2.5, 2)); // Displays 6.25
System.out.println(Math.pow(2.5, -2)); // Displays 0.16
```

Chapter 5 introduces more details on methods. For now, all you need to know is how to invoke the `pow` method to perform the exponent operation.

MyProgrammingLab™

2.8 Find the largest and smallest `byte`, `short`, `int`, `long`, `float`, and `double`. Which of these data types requires the least amount of memory?

2.9 Show the result of the following remainders.

```
56 % 6
78 % -4
-34 % 5
-34 % -5
5 % 1
1 % 5
```

2.10 If today is Tuesday, what will be the day in 100 days?

2.11 What is the result of `25 / 4`? How would you rewrite the expression if you wished the result to be a floating-point number?

2.12 Are the following statements correct? If so, show the output.

```
System.out.println("25 / 4 is " + 25 / 4);
System.out.println("25 / 4.0 is " + 25 / 4.0);
System.out.println("3 * 2 / 4 is " + 3 * 2 / 4);
System.out.println("3.0 * 2 / 4 is " + 3.0 * 2 / 4);
```

2.13 Write a statement to display the result of $2^{3.5}$.

2.14 Suppose `m` and `r` are integers. Write a Java expression for mr^2 to obtain a floating-point result.

2.10 Numeric Literals

A literal *is a constant value that appears directly in a program.*

literal

For example, `34` and `0.305` are literals in the following statements:

```
int numberOfYears = 34;
double weight = 0.305;
```

2.10.1 Integer Literals

An integer literal can be assigned to an integer variable as long as it can fit into the variable. A compile error will occur if the literal is too large for the variable to hold. The statement `byte b = 128`, for example, will cause a compile error, because `128` cannot be stored in a variable of the `byte` type. (Note that the range for a byte value is from `-128` to `127`.)

2.11 Evaluating Expressions and Operator Precedence

Java expressions are evaluated in the same way as arithmetic expressions.

Writing a numeric expression in Java involves a straightforward translation of an arithmetic expression using Java operators. For example, the arithmetic expression

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

can be translated into a Java expression as:

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9 * (4 / x + (9 + x) / y)
```

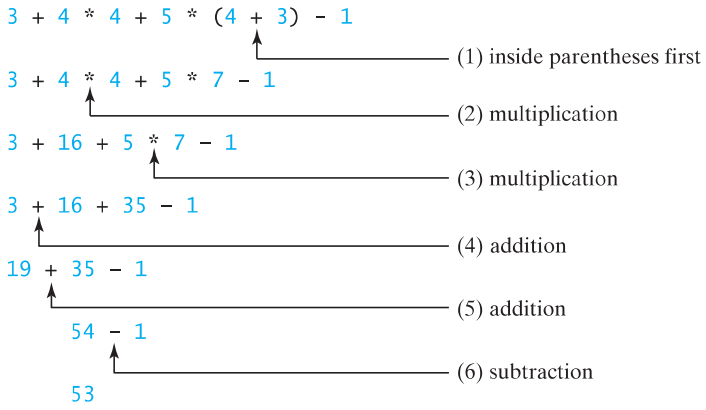
evaluating an expression

Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Java expression. Operators contained within pairs of parentheses are evaluated first. Parentheses can be nested, in which case the expression in the inner parentheses is evaluated first. When more than one operator is used in an expression, the following operator precedence rule is used to determine the order of evaluation.

operator precedence rule

- Multiplication, division, and remainder operators are applied first. If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
- Addition and subtraction operators are applied last. If an expression contains several addition and subtraction operators, they are applied from left to right.

  Here is an example of how an expression is evaluated:



Listing 2.5 gives a program that converts a Fahrenheit degree to Celsius using the formula $celsius = (\frac{5}{9})(fahrenheit - 32)$.

LISTING 2.5 FahrenheitToCelsius.java

```
1  import java.util.Scanner;
2
3  public class FahrenheitToCelsius {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6
7      System.out.print("Enter a degree in Fahrenheit: ");
```

```
 8      double fahrenheit = input.nextDouble();
 9
10      // Convert Fahrenheit to Celsius
11      double celsius = (5.0 / 9) * (fahrenheit - 32);
12      System.out.println("Fahrenheit " + fahrenheit + " is " +
13        celsius + " in Celsius");
14    }
15  }
```

divide

```
Enter a degree in Fahrenheit: 100 Enter
Fahrenheit 100.0 is 37.77777777777778 in Celsius
```

line#	fahrenheit	celsius
8	100	
11		37.77777777777778

Be careful when applying division. Division of two integers yields an integer in Java. $\frac{5}{9}$ is translated to `5.0 / 9` instead of `5 / 9` in line 11, because `5 / 9` yields `0` in Java.

integer vs. decimal division

2.17 How would you write the following arithmetic expression in Java?

$$\frac{4}{3(r + 34)} - 9(a + bc) + \frac{3 + d(2 + a)}{a + bd}$$

MyProgrammingLab™

2.12 Case Study: Displaying the Current Time

You can invoke `System.currentTimeMillis()` *to return the current time.*

The problem is to develop a program that displays the current time in GMT (Greenwich Mean Time) in the format hour:minute:second, such as 13:19:8.

VideoNote
Use operators / and %

The `currentTimeMillis` method in the `System` class returns the current time in milliseconds elapsed since the time `00:00:00` on January 1, 1970 GMT, as shown in Figure 2.2. This time is known as the *UNIX epoch*. The epoch is the point when time starts, and `1970` was the year when the UNIX operating system was formally introduced.

`currentTimeMillis`
UNIX epoch

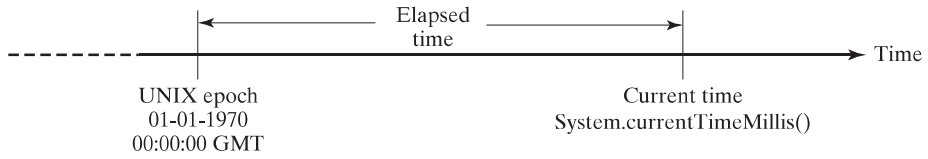


FIGURE 2.2 The `System.currentTimeMillis()` returns the number of milliseconds since the UNIX epoch.

You can use this method to obtain the current time, and then compute the current second, minute, and hour as follows.

1. Obtain the total milliseconds since midnight, January 1, 1970, in `totalMilliseconds` by invoking `System.currentTimeMillis()` (e.g., `1203183068328` milliseconds).

variables \ line#	4	7	10	13	16	19	22
totalMilliseconds	1203183068328						
totalSeconds		1203183068					
currentSecond			8				
totalMinutes				20053051			
currentMinute					31		
totalHours						334217	
currentHour							17

In the sample run, a single digit `8` is displayed for the second. The desirable output would be `08`. This can be fixed by using a function that formats a single digit with a prefix `0` (see Exercise 5.37).

2.13 Augmented Assignment Operators

The operators `+`, `-`, `*`, `/`, and `%` can be combined with the assignment operator to form augmented operators.

Very often the current value of a variable is used, modified, and then reassigned back to the same variable. For example, the following statement increases the variable `count` by `1`:

```
count = count + 1;
```

Java allows you to combine assignment and addition operators using an augmented (or compound) assignment operator. For example, the preceding statement can be written as:

```
count += 1;
```

The `+=` is called the *addition assignment operator*. Table 2.4 shows other augmented assignment operators.

addition assignment operator

TABLE 2.4 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Caution

There are no spaces in the augmented assignment operators. For example, `+ =` should be `+=`.

Note

Like the assignment operator (`=`), the operators (`+=`, `-=`, `*=`, `/=`, `%=`) can be used to form an assignment statement as well as an expression. For example, in the following code, `x += 2` is a statement in the first line and an expression in the second line.

```
x += 2; // Statement
System.out.println(x += 2); // Expression
```

2.14 Increment and Decrement Operators

The increment (++) and decrement (--) operators are for incrementing and decrementing a variable by 1.

increment operator (++)
decrement operator (--)

The `++` and `--` are two shorthand operators for incrementing and decrementing a variable by `1`. These are handy, because that's often how much the value needs to be changed in many programming tasks. For example, the following code increments `i` by `1` and decrements `j` by `1`.

```
int i = 3, j = 3;
i++; // i becomes 4
j--; // j becomes 2
```

postincrement
postdecrement

`i++` is pronounced as `i` plus plus and `i--` as `i` minus minus. These operators are known as *postfix increment* (or postincrement) and *postfix decrement* (or postdecrement), because the operators `++` and `--` are placed after the variable. These operators can also be placed before the variable. For example,

```
int i = 3, j = 3;
++i; // i becomes 4
--j; // j becomes 2
```

preincrement
predecrement

`++i` increments `i` by `1` and `--j` decrements `j` by `1`. These operators are known as *prefix increment* (or preincrement) and *prefix decrement* (or predecrement).

As you see, the effect of `i++` and `++i` or `i--` and `--i` are the same in the preceding examples. However, their effects are different when they are used in statements that do more than just increment and decrement. Table 2.5 describes their differences and gives examples.

TABLE 2.5 Increment and Decrement Operators

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

2.20 How do you obtain the current minute using the `System.currentTimeMillis()` method?

2.15 Numeric Type Conversions

Floating-point numbers can be converted into integers using explicit casting.

Can you perform binary operations with two operands of different types? Yes. If an integer and a floating-point number are involved in a binary operation, Java automatically converts the integer to a floating-point value. So, `3 * 4.5` is same as `3.0 * 4.5`.

You can always assign a value to a numeric variable whose type supports a larger range of values; thus, for instance, you can assign a `long` value to a `float` variable. You cannot, however, assign a value to a variable of a type with a smaller range unless you use *type casting*. *Casting* is an operation that converts a value of one data type into a value of another data type. Casting a type with a small range to a type with a larger range is known as *widening a type*. Casting a type with a large range to a type with a smaller range is known as *narrowing a type*. Java will automatically widen a type, but you must narrow a type explicitly.

casting
widening a type
narrowing a type

The syntax for casting a type is to specify the target type in parentheses, followed by the variable's name or the value to be cast. For example, the following statement

```
System.out.println((int)1.7);
```

displays `1`. When a `double` value is cast into an `int` value, the fractional part is truncated.

The following statement

```
System.out.println((double)1 / 2);
```

displays `0.5`, because `1` is cast to `1.0` first, then `1.0` is divided by `2`. However, the statement

```
System.out.println(1 / 2);
```

displays `0`, because `1` and `2` are both integers and the resulting value should also be an integer.

Caution

Casting is necessary if you are assigning a value to a variable of a smaller type range, such as assigning a `double` value to an `int` variable. A compile error will occur if casting is not used in situations of this kind. However, be careful when using casting, as loss of information might lead to inaccurate results.

possible loss of precision

Note

Casting does not change the variable being cast. For example, `d` is not changed after casting in the following code:

```
double d = 4.5;
int i = (int)d;  // i becomes 4, but d is still 4.5
```

Note

In Java, an augmented expression of the form `x1 op= x2` is implemented as `x1 = (T)(x1 op x2)`, where `T` is the type for `x1`. Therefore, the following code is correct.

casting in an augmented expression

```
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
```

`sum += 4.5` is equivalent to `sum = (int)(sum + 4.5)`.

```
System.out.println("f is " + f);
System.out.println("i is " + i);
```

2.24 If you change `(int)(tax * 100) / 100.0` to `(int)(tax * 100) / 100` in line 11 in Listing 2.7, what will be the output for the input purchase amount of `197.556`?

2.16 Software Development Process

The software development life cycle is a multi-stage process that includes requirements specification, analysis, design, implementation, testing, deployment, and maintenance.

VideoNote
Software development process

Developing a software product is an engineering process. Software products, no matter how large or how small, have the same life cycle: requirements specification, analysis, design, implementation, testing, deployment, and maintenance, as shown in Figure 2.3.

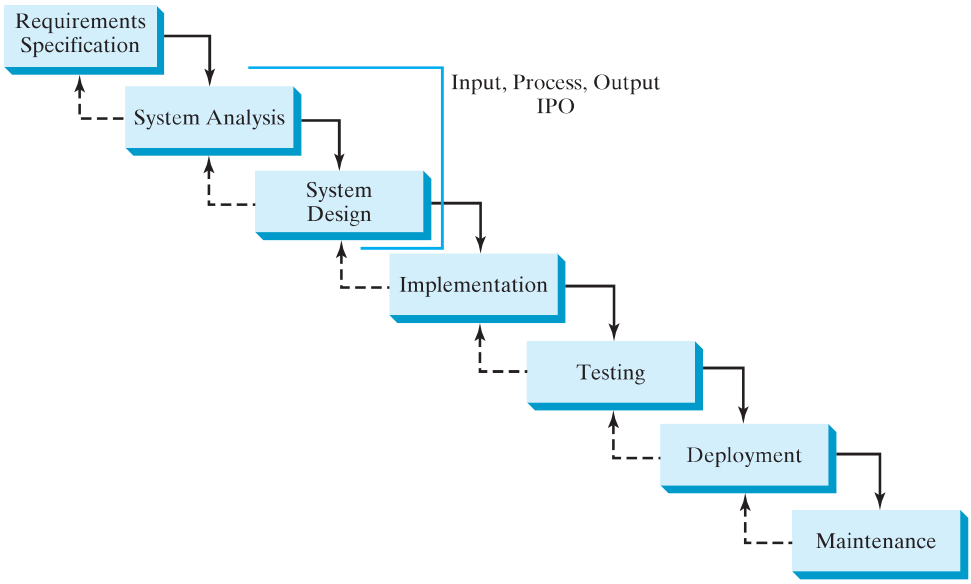


FIGURE 2.3 At any stage of the software development life cycle, it may be necessary to go back to a previous stage to correct errors or deal with other issues that might prevent the software from functioning as expected.

requirements specification

Requirements specification is a formal process that seeks to understand the problem that the software will address and to document in detail what the software system needs to do. This phase involves close interaction between users and developers. Most of the examples in this book are simple, and their requirements are clearly stated. In the real world, however, problems are not always well defined. Developers need to work closely with their customers (the individuals or organizations that will use the software) and study the problem carefully to identify what the software needs to do.

system analysis

System analysis seeks to analyze the data flow and to identify the system's input and output. When you do analysis, it helps to identify what the output is first, and then figure out what input data you need in order to produce the output.

system design

System design is to design a process for obtaining the output from the input. This phase involves the use of many levels of abstraction to break down the problem into manageable components and design strategies for implementing each component. You can view each component as a subsystem that performs a specific function of the system. The essence of system analysis and design is input, process, and output (IPO).

IPO

Check Point

MyProgrammingLab™

2.25 How would you write the following arithmetic expression?

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

2.17 Character Data Type and Operations

Key Point

A character data type represents a single character.

char type

In addition to processing numeric values, you can process characters in Java. The character data type, `char`, is used to represent a single character. A character literal is enclosed in single quotation marks. Consider the following code:

```
char letter = 'A';
char numChar = '4';
```

The first statement assigns character `A` to the `char` variable `letter`. The second statement assigns digit character `4` to the `char` variable `numChar`.

char literal

Caution

A string literal must be enclosed in quotation marks (`" "`). A character literal is a single character enclosed in single quotation marks (`' '`). Therefore, `"A"` is a string, but `'A'` is a character.

2.17.1 Unicode and ASCII code

encoding

Computers use binary numbers internally. A character is stored in a computer as a sequence of 0s and 1s. Mapping a character to its binary representation is called *encoding*. There are different ways to encode a character. How characters are encoded is defined by an *encoding scheme*.

Unicode

Java supports *Unicode*, an encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages. Unicode was originally designed as a 16-bit character encoding. The primitive data type `char` was intended to take advantage of this design by providing a simple data type that could hold any character. However, it turned out that the 65,536 characters possible in a 16-bit encoding are not sufficient to represent all the characters in the world. The Unicode standard therefore has been extended to allow up to 1,112,064 characters. Those characters that go beyond the original 16-bit limit are called *supplementary characters*. Java supports the supplementary characters. The processing and representing of supplementary characters are beyond the scope of this book. For simplicity, this book considers only the original 16-bit Unicode characters. These characters can be stored in a `char` type variable.

original Unicode

supplementary Unicode

A 16-bit Unicode takes two bytes, preceded by `\u`, expressed in four hexadecimal digits that run from `\u0000` to `\uFFFF`. Hexadecimal numbers are introduced in Appendix F, Number Systems. For example, the English word `welcome` is translated into Chinese using two characters, 欢迎. The Unicodes of these two characters are `\u6B22\u8FCE`.

Listing 2.9 gives a program that displays two Chinese characters and three Greek letters.

LISTING 2.9 DisplayUnicode.java

```
1 import javax.swing.JOptionPane;
2
3 public class DisplayUnicode {
4   public static void main(String[] args) {
5     JOptionPane.showMessageDialog(null,
6       "\u6B22\u8FCE \u03b1 \u03b2 \u03b3",
7       "\u6B22\u8FCE Welcome",
```

2.31 Show the output of the following program:

```
public class Test {
  public static void main(String[] args) {
    char x = 'a';
    char y = 'c';

    System.out.println(++x);
    System.out.println(y++);
    System.out.println(x - y);
  }
}
```

2.18 The `String` Type

A string is a sequence of characters.

The `char` type represents only one character. To represent a string of characters, use the data type called `String`. For example, the following code declares the message to be a string with the value `"Welcome to Java"`.

```
String message = "Welcome to Java";
```

`String` is a predefined class in the Java library, just like the classes `System`, `JOptionPane`, and `Scanner`. The `String` type is not a primitive type. It is known as a *reference type*. Any Java class can be used as a reference type for a variable. Reference data types will be thoroughly discussed in Chapter 8, Objects and Classes. For the time being, you need to know only how to declare a `String` variable, how to assign a string to the variable, and how to concatenate strings.

concatenate strings and numbers

As first shown in Listing 2.1, two strings can be concatenated. The plus sign (+) is the concatenation operator if one of the operands is a string. If one of the operands is a nonstring (e.g., a number), the nonstring value is converted into a string and concatenated with the other string. Here are some examples:

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

If neither of the operands is a string, the plus sign (+) is the addition operator that adds two numbers.

The augmented `+=` operator can also be used for string concatenation. For example, the following code appends the string `"and Java is fun"` with the string `"Welcome to Java"` in `message`.

```
message += " and Java is fun";
```

So the new `message` is `"Welcome to Java and Java is fun"`.

If `i = 1` and `j = 2`, what is the output of the following statement?

```
System.out.println("i + j is " + i + j);
```

The output is `"i + j is 12"` because `"i + j is "` is concatenated with the value of `i` first. To force `i + j` to be executed first, enclose `i + j` in the parentheses, as follows:

```
System.out.println("i + j is " + (i + j));
```

2.19 Getting Input from Input Dialogs

An input dialog box prompts the user to enter an input graphically.

Key Point

JOptionPane class

You can obtain input from the console. Alternatively, you can obtain input from an input dialog box by invoking the `JOptionPane.showInputDialog` method, as shown in Figure 2.4.

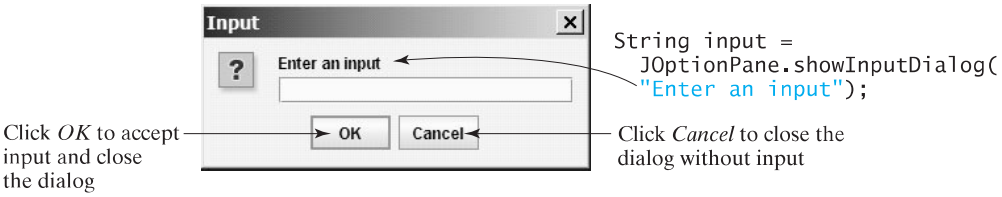


FIGURE 2.4 The input dialog box enables the user to enter a string.

When this method is executed, a dialog is displayed to enable you to enter an input value. After entering a string, click *OK* to accept the input and close the dialog box. The input is returned from the method as a string.

showInputDialog method

There are several ways to use the `showInputDialog` method. For the time being, you need to know only two ways to invoke it.

One is to use a statement like this one:

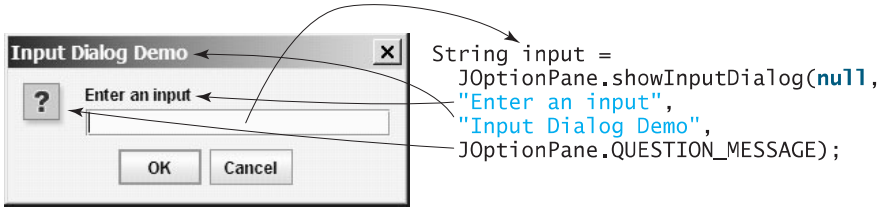
```
JOptionPane.showInputDialog(x);
```

where `x` is a string for the prompting message.

The other is to use a statement such as the following:

```
String string = JOptionPane.showInputDialog(null, x,
  y, JOptionPane.QUESTION_MESSAGE);
```

where `x` is a string for the prompting message and `y` is a string for the title of the input dialog box, as shown in the example below.



2.19.1 Converting Strings to Numbers

The input returned from the input dialog box is a string. If you enter a numeric value such as `123`, it returns `"123"`. You have to convert a string into a number to obtain the input as a number.

Integer.parseInt method

To convert a string into an `int` value, use the `Integer.parseInt` method, as follows:

```
int intValue = Integer.parseInt(intString);
```

where `intString` is a numeric string such as `123`.

Double.parseDouble method

To convert a string into a `double` value, use the `Double.parseDouble` method, as follows:

```
double doubleValue = Double.parseDouble(doubleString);
```

where `doubleString` is a numeric string such as `123.45`.

The `Integer` and `Double` classes are both included in the `java.lang` package, and thus they are automatically imported.