CHAPTER SUMMARY

1. Making programs modular and reusable is one of the central goals in software engineering. Java provides many powerful constructs that help to achieve this goal. Methods are one such construct.

2. The method header specifies the modifiers, return value type, method name, and parameters of the method. The static modifier is used for all the methods in this chapter.

3. A method may return a value. The returnType is the data type of the value the method returns. If the method does not return a value, the returnType is the keyword void.

4. The parameter list refers to the type, order, and number of the parameters of a method. The method name and the parameter list together constitute the method signature. Parameters are optional; that is, a method may contain no parameters.

5. A return statement can also be used in a void method for terminating the method and returning to the method’s caller. This is useful occasionally for circumventing the normal flow of control in a method.

6. The arguments that are passed to a method should have the same number, type, and order as the parameters in the method signature.

7. When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method-ending closing brace is reached.

8. A value-returning method can also be invoked as a statement in Java. In this case, the caller simply ignores the return value.

9. Each time a method is invoked, the system stores parameters and local variables in a space known as a stack. When a method calls another method, the caller’s stack space is kept intact, and new space is created to handle the new method call. When a method finishes its work and returns to its caller, its associated space is released.

10. A method can be overloaded. This means that two methods can have the same name, as long as their method parameter lists differ.

11. A variable declared in a method is called a local variable. The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared and initialized before it is used.
12. *Method abstraction* is achieved by separating the use of a method from its implementation. The client can use a method without knowing how it is implemented. The details of the implementation are encapsulated in the method and hidden from the client who invokes the method. This is known as *information hiding* or *encapsulation*.

13. Method abstraction modularizes programs in a neat, hierarchical manner. Programs written as collections of concise methods are easier to write, debug, maintain, and modify than would otherwise be the case. This writing style also promotes method reusability.

14. When implementing a large program, use the top-down or bottom-up coding approach. Do not write the entire program at once. This approach seems to take more time for coding (because you are repeatedly compiling and running the program), but it actually saves time and makes debugging easier.

**Review Questions**

Sections 5.2–5.4

5.1 What are the benefits of using a method? How do you define a method? How do you invoke a method?

5.2 What is the return type of a *main* method?

5.3 Can you simplify the *max* method in Listing 5.1 using the conditional operator?

5.4 True or false? A call to a method with a *void* return type is always a statement itself, but a call to a value-returning method is always a component of an expression.

5.5 What would be wrong with not writing a return statement in a value-returning method? Can you have a return statement in a *void* method? Does the return statement in the following method cause syntax errors?

```java
public static void xMethod(double x, double y) {
    System.out.println(x + y);
    return x + y;
}
```

5.6 Define the terms parameter, argument, and method signature.

5.7 Write method headers for the following methods:

- Computing a sales commission, given the sales amount and the commission rate.
- Printing the calendar for a month, given the month and year.
- Computing a square root.
- Testing whether a number is even, and returning *true* if it is.
- Printing a message a specified number of times.
- Computing the monthly payment, given the loan amount, number of years, and annual interest rate.
- Finding the corresponding uppercase letter, given a lowercase letter.
5.8 Identify and correct the errors in the following program:

```java
public class Test {
    public static method1(int n, m) {
        n += m;
        method2(3.4);
    }

    public static int method2(int n) {
        if (n > 0) return 1;
        else if (n == 0) return 0;
        else if (n < 0) return -1;
    }
}
```

5.9 Reformat the following program according to the programming style and documentation guidelines proposed in §2.16, “Programming Style and Documentation.” Use the next-line brace style.

```java
public class Test {
    public static double method1(double i, double j) {
        while (i < j) {
            j--;
        }
        return j;
    }
}
```

Sections 5.5–5.7

5.10 How is an argument passed to a method? Can the argument have the same name as its parameter?

5.11 What is pass-by-value? Show the result of the following programs:

(a) 
```java
public class Test {
    public static void main(String[] args) {
        int max = 0;
        max(1, 2, max);
        System.out.println(max);
    }

    public static void max(int value1, int value2, int max) {
        if (value1 > value2)
            max = value1;
        else
            max = value2;
    }
}
```

(b) 
```java
public class Test {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 6) {
            method1(i, 2);
            i++;
        }
    }

    public static void method1(int i, int num) {
        for (int j = 1; j <= i; j++) {
            System.out.print(num + " ");
            num *= 2;
        }
    }
    System.out.println();
}
```
5.12 For (a) in the preceding question, show the contents of the stack just before the method `max` is invoked, just as `max` is entered, just before `max` is returned, and right after `max` is returned.

**Section 5.8**

5.13 What is method overloading? Is it permissible to define two methods that have the same name but different parameter types? Is it permissible to define two methods in a class that have identical method names and parameter lists but different return value types or different modifiers?

5.14 What is wrong in the following program?

```java
public class Test {
    public static void method(int x) {
    }

    public static int method(int y) {
        return y;
    }
}
```

**Section 5.9**

5.15 Identify and correct the errors in the following program:

```java
1 public class Test {
2     public static void main(String[] args) {
3         nPrintln("Welcome to Java!", 5);
4     }
5 }
6 public static void nPrintln(String message, int n) {
7     int n = 1;
```
8     for (int i = 0; i < n; i++)
9         System.out.println(message);
10     }
11 }

Section 5.10

5.16 True or false? The argument for trigonometric methods represents an angle in radians.

5.17 Write an expression that returns a random integer between 34 and 55. Write an expression that returns a random integer between 0 and 999. Write an expression that returns a random number between 5.5 and 55.5. Write an expression that returns a random lowercase letter.

5.18 Evaluate the following method calls:

(a) Math.sqrt(4)
(b) Math.sin(2 * Math.PI)
(c) Math.cos(2 * Math.PI)
(d) Math.pow(2, 2)
(e) Math.log(Math.E)
(f) Math.exp(1)
(g) Math.max(2, Math.min(3, 4))
(h) Math.rint(-2.5)
(i) Math.ceil(-2.5)
(j) Math.floor(-2.5)
(k) Math.round(-2.5F)
(l) Math.round(-2.5)
(m) Math.rint(2.5)
(n) Math.ceil(2.5)
(o) Math.floor(2.5)
(p) Math.round(2.5F)
(q) Math.round(2.5)
(r) Math.round(Math.abs(-2.5))

Programming Exercises

Sections 5.2–5.9

5.1 (Math: pentagonal numbers) A pentagonal number is defined as \( n(3n-1)/2 \) for \( n = 1, 2, \ldots \), and so on. So, the first few numbers are 1, 5, 12, 22, \ldots. Write the following method that returns a pentagonal number:

\[
\text{public static int getPentagonalNumber(int n)}
\]

Write a test program that displays the first 100 pentagonal numbers with 10 numbers on each line.

5.2* (Summing the digits in an integer) Write a method that computes the sum of the digits in an integer. Use the following method header:

\[
\text{public static int sumDigits(long n)}
\]

For example, \( \text{sumDigits(234)} \) returns 9 \((2 + 3 + 4)\).

(Hint: Use the % operator to extract digits, and the / operator to remove the extracted digit. For instance, to extract 4 from 234, use \(234 \% 10\) \(= 4\). To remove 4 from 234, use \(234 / 10\) \(= 23\). Use a loop to repeatedly extract and remove the digit until all the digits are extracted. Write a test program that prompts the user to enter an integer and displays the sum of all its digits.)