6.4 Methods with Multiple Parameters

Methods often require more than one piece of information to perform their tasks. We now consider how to write your own methods with *multiple* parameters.

Figure 6.3 uses a method called `maximum` to determine and return the largest of three `double` values. In `main`, lines 11–15 prompt the user to enter three `double` values, then read them from the user. Line 18 calls method `maximum` (declared in lines 25–39) to determine the largest of the three values it receives as arguments. When method `maximum` returns the result to line 18, the program assigns `maximum`’s return value to local variable `result`. Then line 21 outputs the maximum value. At the end of this section, we’ll discuss the use of operator + in line 21.

```
1 // Fig. 6.3: MaximumFinder.java
2 // Programmer-declared method maximum with three d
3 import java.util.Scanner;
4
5 public class MaximumFinder {
6    public static void main(String[] args) {
7       // create Scanner for input from command win
8       Scanner input = new Scanner(System.in);
9
10      // prompt for and input three floating-point
11      System.out.print(
           "Enter three floating-point values separa
        double number1 = input.nextDouble(); // read
        double number2 = input.nextDouble(); // read
        double number3 = input.nextDouble(); // read

        // determine the maximum value
        double result = maximum(number1, number2, nu

        // display maximum value
        System.out.println("Maximum is: " + result);
    }

    // returns the maximum of its three double para
    public static double maximum(double x, double y
        double maximumValue = x; // assume x is the

        // determine whether y is greater than maxim
        if (y > maximumValue) {
            maximumValue = y;
        }

        // determine whether z is greater than maxim
        if (z > maximumValue) {
            maximumValue = z;
        }

        return maximumValue;
    }
}
```

```
Enter three floating-point values separated by spaces
Maximum is: 9.35
```

```
Enter three floating-point values separated by spaces
Maximum is: 12.45
```

```
   Enter three floating-point values separated by spaces
                  Maximum is: 10.54
```

Fig. 6.3

Programmer-declared method maximum with three double parameters.

The `public` and `static` Keywords

Method `maximum`’s declaration begins with keyword `public` to indicate that the method is “available to the public”—it can be called from methods of other classes. The keyword `static` enables the `main` method (another `static` method) to call `maximum` as shown in line 18 without qualifying the method name with the class name `MaximumFinder`—`static` methods in the same class can call each other directly. Any other class that uses `maximum` must fully qualify the method name, as in `MaximumFinder.maximum(10, 30, 20)`.

Method `maximum`

Consider `maximum`'s declaration (lines 25–39). Line 25 indicates that it returns a `double` value, that the method's name is `maximum` and that the method requires three `double` parameters (`x`, `y` and `z`) to accomplish its task. Multiple parameters are specified as a comma-separated list. When `maximum` is called from line 18, the parameters `x`, `y` and `z` are initialized with copies of the values of arguments `number1`, `number2` and `number3`, respectively. There must be one argument in the method call for each parameter in the method declaration. Each argument also must be *consistent* with the corresponding parameter's type. For example, a parameter of type `double` can receive values like 7.35, 22 or –0.03456, but not `String`s like `"hello"` nor the `boolean` values `true` or `false`. Section 6.7 discusses the argument types that you can provide in a method call for each primitive-type parameter.

To determine the maximum value, we initially assume that parameter `x` contains the largest value, so line 26 declares local variable `maximumValue` and initializes it with the value of parameter `x`. Of course, it's possible that parameter `y` or `z` contains the actual largest value, so we must compare each of these with `maximumValue`. Lines 29–31 determine whether `y` is greater than `maximumValue`. If so, line 30 assigns `y` to `maximumValue`. Lines 34–36 determine whether `z` is greater than `maximumValue`. If so, line 35 assigns `z` to `maximumValue`. At this point the largest value resides in `maximumValue`, so line 38 returns that value to line 18. When program control returns to where `maximum` was called, `maximum`'s parameters `x`, `y` and `z` no longer exist in memory.

Software Engineering Observation 6.5

Methods can return at most one value, but the returned value could be a reference to an object that contains many values in its instance variables.

Software Engineering Observation 6.6

Variables should be declared as fields only if they're required for use in more than one method of the class or if the program should save their values between calls to the class's methods.

Common Programming Error 6.1

Declaring method parameters of the same type as `float x, y` *instead of* `float x, float y` *is a syntax error—a type is required for each parameter in the parameter list.*

Implementing Method `maximum` by Reusing Method `Math.max`

The entire body of our maximum method could also be implemented with two calls to `Math.max`, as follows:

```
return Math.max(x, Math.max(y, z));
```

The first call to `Math.max` specifies arguments `x` and `Math.max(y, z)`. *Before* any method can be called, its arguments must be evaluated to determine their values. If an argument is a method call, the method call must be performed to determine its return value. So, in the preceding statement, `Math.max(y, z)` is evaluated to determine the maximum of `y` and `z`. Then the result is passed as the second argument to the other call to `Math.max`, which returns the larger of its two arguments. This is a good example of *software reuse*—we find the largest of three values by reusing `Math.max`, which finds the larger of two values. Note how concise this code is compared to lines 26–38 of Fig. 6.3.

Assembling Strings with String Concatenation

Java allows you to assemble `String` objects into larger strings by using operators `+` or `+=`. This is known as **string concatenation**. When both operands of operator `+` are `String` objects, operator `+` creates a new `String` object containing the characters of the left operand followed by those of the right operand—so the expression `"hello"+"there"` creates the `String` `"hello there"`.

In line 21 of Fig. 6.3, the expression

```
"Maximum is: " + result
```

uses operator `+` with operands of types `String` and `double`. *Every primitive value and object in Java can be represented as a* `String`. When one of the `+` operator's operands is a `String`, the other is converted to a `String`, then the two are *concatenated*. In line 21, the `double` value is converted to its `String` representation and placed at the end of `"Maximum is: "`. If there are any *trailing zeros* in a `double` value, these will be *discarded* when the number is converted to a `String`—for example, 9.3500 would be represented as 9.35.

Primitive values used in `String` concatenation are converted to `String`s. A `boolean` concatenated with a `String` is converted to the `String` `"true"` or `"false"`. *All objects have a* `toString` *method that returns a* `String` *representation of the object.*(We discuss `toString` in more detail in subsequent chapters.) When an object is concatenated with a `String`, the object's `toString` method is called implicitly and returns the object's `String` representation.

Method `toString` also can be called explicitly.

You can break large `String` literals into several smaller `String`s and place them on multiple lines of code for readability. In this case, the `String`s can be reassembled using concatenation. We discuss the details of `String`s in Chapter 14.

Common Programming Error 6.2

It's a syntax error to break a `String` *literal across lines. If necessary, you can split a* `String` *into several smaller* `String`*s and use concatenation to form the desired* `String`.

Common Programming Error 6.3

Confusing the`+` *operator used for string concatenation with the* `+` *operator used for addition can lead to strange results. Java evaluates the operands of an operator from left to right. For example, if integer variable* `y` *has the value* `5`*, the expression* `"y + 2 =" + y + 2` *results in the string* `"y + 2 = 52"`*, not* `"y + 2 = 7"`*, because first the value of* `y` *(5) is concatenated to the string* `"y + 2 = "`*, then the value* `2` *is* *concatenated to the new larger string* `"y + 2 = 5"`. *The expression* `"y + 2  =" + (y + 2)` *produces the desired result* `"y + 2 = 7"`.