CS3231
Object Oriented Programming I

Name: ________________________________ (   ) Date: _______________

Chapter 7: Methods

Lesson Objectives

After completing the lesson, the student will be able to

- *define methods with formal parameters*
- *invoke methods with actual parameters (i.e., arguments)*
- *define methods with a return value*
- *define methods without a return value*
- *pass arguments by value*
- *use method overloading and understand ambiguous overloading*

7.1 Writing Methods

Methods can be used to define reusable code and organize and simplify coding. Suppose that you need to find the sum of integers from 1 to 10, from 20 to 37, and from 35 to 49, respectively. You may write the code as follows:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);
sum = 0;
for (int i = 20; i <= 37; i++)
  sum += i;
System.out.println("Sum from 20 to 37 is " + sum);
sum = 0;
for (int i = 35; i <= 49; i++)
 sum += i;
System.out.println("Sum from 35 to 49 is " + sum);
```

You may have observed that computing these sums from 1 to 10, from 20 to 37, and from 35 to 49 are very similar except that the starting and ending integers are different. It would be much more efficient if we could write the common code once and reuse it. This can be done by defining a method and invoking it.

PROGRAM 7-1

```
public class testSum {
  public static int sum(int i1, int i2) {
     int result = 0;
     for (int i = i1; i <= i2; i++)
        result += i;

     return result;
  }
  public static void main(String[] args){
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 20 to 37 is " + sum(20, 37));
    System.out.println("Sum from 35 to 49 is " + sum(35, 49));
  }
}
```

PROGRAM OUTPUT

```
Sum from 1 to 10 is 55
Sum from 20 to 37 is 513
Sum from 35 to 49 is 630
```

Lines 1–7 define the method named `sum()` with two parameters i1 and i2. The statements in the `main` method invoke `sum(1, 10)` to compute the sum from 1 to 10, `sum(20, 37)` to compute the sum from 20 to 37, and `sum(35, 49)` to compute the sum from 35 to 49.

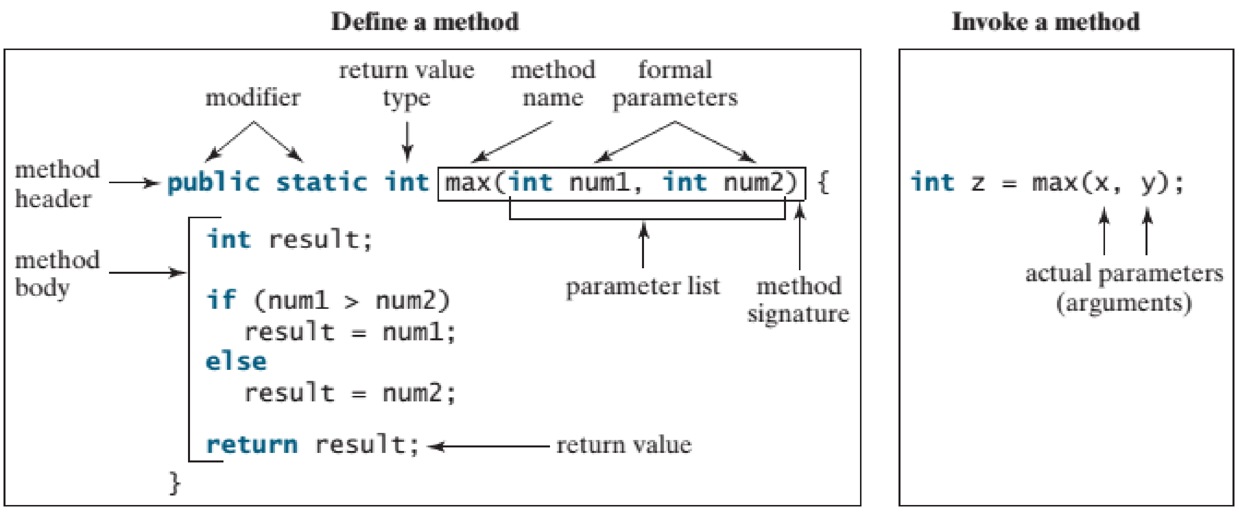
A method is a collection of statements grouped together to perform an operation. Earlier on, you have used predefined methods such as `System.out.println()`, `System.exit()`, `Math.pow()`, and `Math.random()`. These methods are defined in the Java API. In this section, you will learn how to define your own methods and apply method abstraction to solve complex problems.

7.1.1 Method Definition

A method definition consists of its method name, parameters, return value type, and body. The syntax for defining a method is as follows:

```
modifier returnValueType methodName(list of parameters) {
 // Method body;
}
```

Let's look at a method defined to find the larger between two integers. This method, named `max()`, has two int parameters, num1 and num2, the larger of which is returned by the method. Figure 7.1 illustrates the components of this method.



```
public static int max(int num1, int num2) {
   int result;

   if (num1 > num2)
     result = num1;
   else
     result = num2;

   return result;
}
```

```
int z = max(x, y);
```

Figure 7.1 A method definition consists of a method header and a method body

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. The reason for using it will be discussed in Chapter 9, Class Design.

A method may return a value. The returnValueType is the data type of the value the method returns. Some methods perform desired operations without returning a value. In this case, the returnValueType is the keyword `void`. For example, the returnValueType is `void` in the `main` method, as well as in `System.exit()`, and `System.out.println()`. **If a method returns a value, it is called a value-returning method; otherwise it is called a void method.**

The variables defined in the method header are known as **formal parameters** or simply parameters. A parameter is like a placeholder: when a method is invoked, you pass a value to the parameter. This value is referred to as an **actual parameter** or argument. The parameter list refers to the method's type, order, and number of the parameters. **The method name and the parameter list together constitute the method signature. Parameters are optional; that is, a method may contain no parameters.** For example, the `Math.random()` method has no parameters.

The method body contains a collection of statements that implement the method. The method body of the `max()` method uses an if statement to determine which number is larger and return the value of that number. In order for a value-returning method to return a result, a return statement using the keyword `return` is required. The method terminates when a return statement is executed. Some programming languages refer to methods as procedures and functions. In the method header, you need to declare each parameter separately. For instance, `max(int num1, int num2)` is correct, but `max(int num1, num2)` is wrong.

7.1.2 Calling a Method

Calling a method executes the code in the method. In a method definition, you define what the method is to do. To execute the method, you have to call or invoke it. There are two ways to call a method, depending on whether the method returns a value or not. **If a method returns a value, a call to the method is usually treated as a value.** For example,

```
int larger = max(3, 4);
```

calls `max(3, 4)` and assigns the result of the method to the variable larger. Another example of a call that is treated as a value is `System.out.println(max(3, 4));` which prints the return value of the method call `max(3, 4)` .

If a method returns `void`, a call to the method must be a statement. For example, the method `println()` returns `void`. The following call is a statement:

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

A value-returning method can also be invoked as a statement in Java. In this case, the caller simply ignores the return value. This is not often done, but it is permissible if the caller is not interested in the return value. 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. PROGRAM 7-2 shows a complete program that is used to test the `max()` method.

PROGRAM 7-2

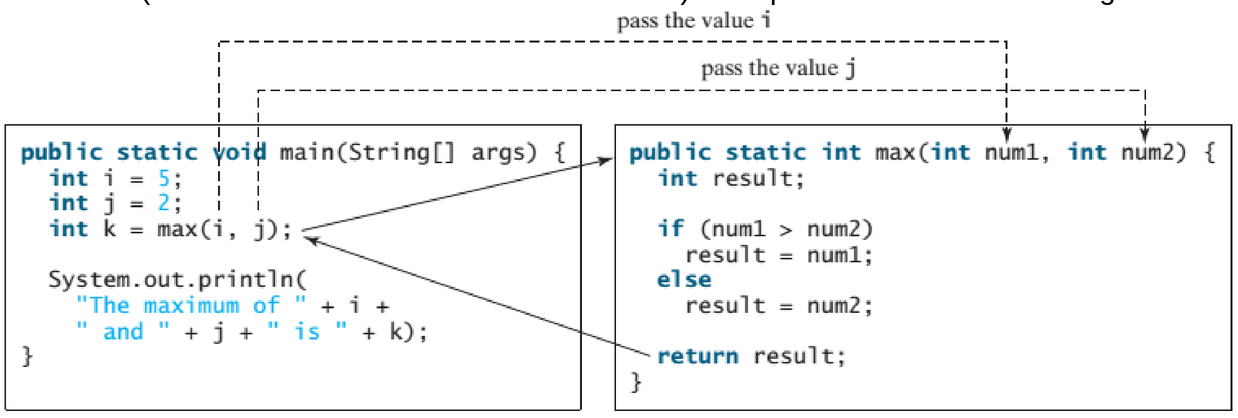
```
public class test {
  public static void main(String[] args) {
      int i = 5;
      int j = 2;
      int k = max(i, j);
      System.out.println("The maximum of " + i + " and " + j +
                            " is " + k);
  }
  public static int max(int num1, int num2){
      int result;
      if (num1 > num2)
         result = num1;
      else
         result = num2;
      return result;
  }
}
```

PROGRAM OUTPUT

```
The maximum of 5 and 2 is 5
```

This program contains the `main` method and the `max()` method. **The `main` method is just like any other method except that it is invoked by the JVM to start the program**. The `main` method's header is always the same. Like the one in this example, it includes the modifiers `public` and `static`, return value type `void`, method name `main`, and a parameter of the `String[]` type.

The statements in `main` may invoke other methods that are defined in the class that contains the `main` method or in other classes. In this example, the main method invokes `max(i, j)` , which is defined in the same class with the `main` method. When the `max()` method is invoked (line 6), variable i 's value 5 is passed to num1, and variable j 's value 2 is passed to num2 in the `max()` method. The flow of control transfers to the `max()` method, and the `max()` method is executed. When the return statement in the max method is executed, the `max()` method returns the control to its caller (in this case the caller is the `main` method). This process is illustrated in Figure 7.1.



```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
    "The maximum of " + i +
    " and " + j + " is " + k);
}
```

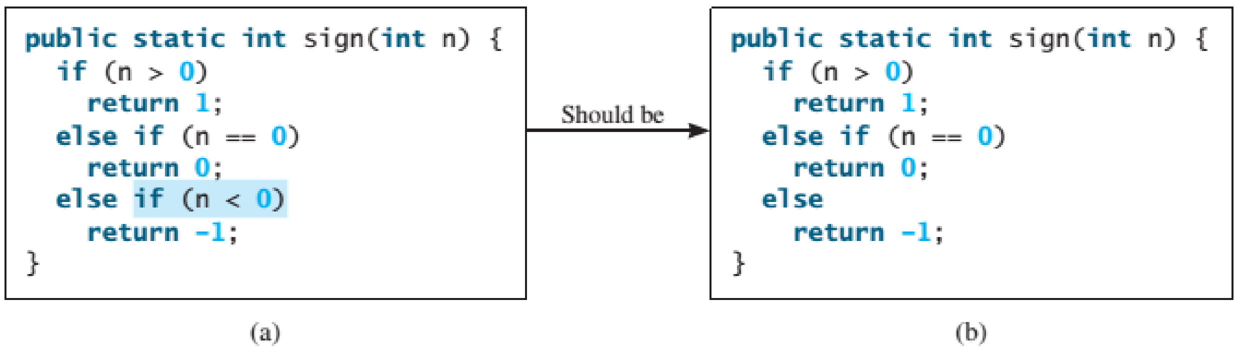
```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Figure 7.1 When the `max()` method is invoked, the flow of control transfers to it. Once the `max()` method is finished, it returns control back to the caller.

A return statement is required for a value-returning method. The method shown below in 7.2(a) is logically correct, but it has a compile error because the Java compiler thinks that this method might not return a value.



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

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

Figure 7.2 To fix this problem, delete if (n < 0) in 7.2(a), so the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

Methods enable code sharing and reuse. The `max()` method can be invoked from any class, not just TestMax. If you create a new class, you can invoke the `max()` method using `ClassName.methodName (i.e., TestMax.max).`

7.1.3 `void` Method

A `void` method does not return a value. PROGRAM 7-3 that defines a method named `printGrade()` and invokes it to print the grade for a given score. The `printGrade()` method is a `void` method because it does not return any value. A call to avoid method must be a statement. Therefore, it is invoked as a statement in line 4 in the `main` method. Like any Java statement, it is terminated with a semicolon.

PROGRAM 7-3

```
1   public class test{
2     public static void main(String[] args) {
3       System.out.print("The grade is ");
4       printGrade(78.5);
5
6       System.out.print("The grade is ");
7       printGrade(59.5);
8     }
9     public static void printGrade(double score) {
10      if (score >= 90.0) {
11         System.out.println('A' );
12      }
13      else if (score >= 80.0) {
14         System.out.println('B' );
15      }
16      else if (score >= 70.0) {
17         System.out.println('C' );
18      }
19      else if (score >= 60.0) {
20         System.out.println('D' );
21      }
22      else {
23         System.out.println('F' );
24      }
25    }
26  }
```

PROGRAM OUTPUT

```
The grade is C
The grade is F
```

7.2 Passing Arguments by Values

The arguments are passed by value to parameters when invoking a method. The power of a method is its ability to work with parameters. You can use `println()` to print any string and `max()` to find the maximum of any two int values**. When calling a method, you need to provide arguments, which must be given in the same order as their respective parameters in the method signature**. For example, the following method prints a message `n` times:

```
public static void nPrintln(String message, int n) {
for (int i = 0; i < n; i++)
 System.out.println(message);
}
```

You can use `nPrintln("Hello", 3)` to print Hello three times. The `nPrintln("Hello",3)` statement passes the actual string parameter Hello to the parameter message, passes 3 to n, and prints Hello three times. However, the statement `nPrintln(3, "Hello")` would be wrong. The data type of 3 does not match the data type for the first parameter, message, nor does the second argument, Hello, match the second parameter, n.

The arguments must match the parameters in order, number, and compatible type, as defined in the method signature. Compatible type means that you can pass an argument to a parameter without explicit casting, such as passing an int value argument to a double value parameter.

When you invoke a method with an argument, the value of the argument is passed to the parameter. This is referred to as pass-by-value. If the argument is a variable rather than a literal value, the value of the variable is passed to the parameter. The variable is not affected, regardless of the changes made to the parameter inside the method.

In PROGRAM 7-4, the value of x (1) is passed to the parameter n to invoke the increment method (line 5). The parameter n is incremented by 1 in the method (line 10), but x is not changed no matter what the method does.

PROGRAM 7-4

```
1   public class Increment {
2    public static void main(String[] args) {
3     int x = 1;
4     System.out.println("Before the call, x is " + x);
5     increment(x);
6     System.out.println("After the call, x is " + x);
7    }
8
9    public static void increment(int n){
10     n++;
11     System.out.println("n inside the method is " + n);
12   }
13  }
```

PROGRAM OUTPUT

```
Before the call, x is 1
n inside the method is 2
After the call, x is 1
```

7.3 Overloading Methods

Overloading methods enables you to define the methods with the same name as long as their signatures are different. The `max()` method that was used earlier works only with the int data type. But what if you need to determine which of two floating-point numbers has the maximum value? The solution is to create another method with the same name but different parameters, as shown in thefollowing code:

```
public static double max(double num1, double num2) {
if (num1 > num2)
  return num1;
else
  return num2;
}
```

If you call `max()` with int parameters, the max method that expects int parameters will be invoked; if you call `max()` with double parameters, the max method that expects double parameters will be invoked. This is referred to as **method overloading**; that is, **two methods have the same name but different parameter lists within one class**. **The Java compiler determines which method to use based on the method signature.**

PROGRAM 7-5 creates three methods. The first finds the maximum integer, the second finds the maximum double, and the third finds the maximum among three double values. All three methods are named `max()`.

PROGRAM 7-5

```
1   public class test{
2     public static void main(String[] args) {
3      // Invoke the max method with int parameters
4      System.out.println("The maximum of 3 and 4 is " + max(3, 4));
5
6      // Invoke the max method with the double parameters
7      System.out.println("The maximum of 3.0 and 5.4 is " + max(3.0,
8   5.4));
9
10     // Invoke the max method with three double parameters
11     System.out.println("The maximum of 3.0, 5.4, and 10.14 is " +
12     max(3.0, 5.4, 10.14));
13    }
14    // Return the max of two int values
15    public static int max(int num1, int num2) {
16      if (num1 > num2)
17         return num1;
18      else
19         return num2;
20     }
21     // Find the max of two double values
22     public static double max(double num1, double num2){
23        if (num1 > num2)
24           return num1;
25        else
26           return num2;
27     }
28     // Return the max of three double values
29     public static double max(double num1, double num2, double num3){
30        return max(max(num1, num2), num3);
31     }
    }
```

PROGRAM OUTPUT

```
The maximum of 3 and 4 is 4
The maximum of 3.0 and 5.4 is 5.4
The maximum of 3.0, 5.4, and 10.14 is 10.14
```

When calling `max(3, 4)` (line 6), the `max()` method for finding the maximum of two integers is invoked. When calling `max(3.0, 5.4)` (line 10), the `max()` method for finding the maximum of two doubles is invoked. When calling `max(3.0, 5.4, 10.14)` (line 14), the `max()` method for finding the maximum of three double values is invoked.

Can you invoke the `max()` method with an int value and a double value, such as `max(2, 2.5)` ? If so, which of the `max()` methods is invoked? The answer to the first question is yes. The answer to the second question is that the `max()` method for finding the maximum of two double values is invoked. The argument value 2 is automatically converted into a double value and passed to this method.

You may be wondering why the method `max(double, double)` is not invoked for the call `max(3, 4)`. Both `max(double, double)` and `max(int, int)` are possible matches for `max(3, 4)`. The Java compiler finds the method that best matches a method invocation. Since the method

`max(int, int)` is a better matches for max(3, 4) than `max(double, double)`, `max(int, int)` is used to invoke `max(3, 4)`.
Overloading methods can make programs clearer and more readable. Methods that perform the same function with different types of parameters should be given the same name. **Overloaded methods must have different parameter lists. You cannot overload methods based on different modifiers or return types.**

Sometimes there are **two or more possible matches for the invocation of a method, but the compiler cannot determine the best match**. This is referred to as **ambiguous invocation**. **Ambiguous invocation causes a compile error**. PROGRAM 7-6 will have a compile error.

PROGRAM 7-6

```
public class AmbiguousOverloading {
 public static void main(String[] args) {
    System.out.println(max(1, 2));
 }
 public static double max(int num1, double num2) {
    if (num1 > num2)
       return num1;
    else
       return num2;
 }
 public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
 }
}
```

Both `max(int, double)` and `max(double, int)` are possible candidates to match `max(1, 2)`. Because neither is better than the other, the invocation is ambiguous, resulting in a compile error.

[Reference]

[1] Introduction to Java Programming Comprehensive Version 10th Ed, Daniel Liang, 2016.
[2] Java How To Program 10th Ed, Paul Deitel, Harvey Deitel, 2016