

Methods

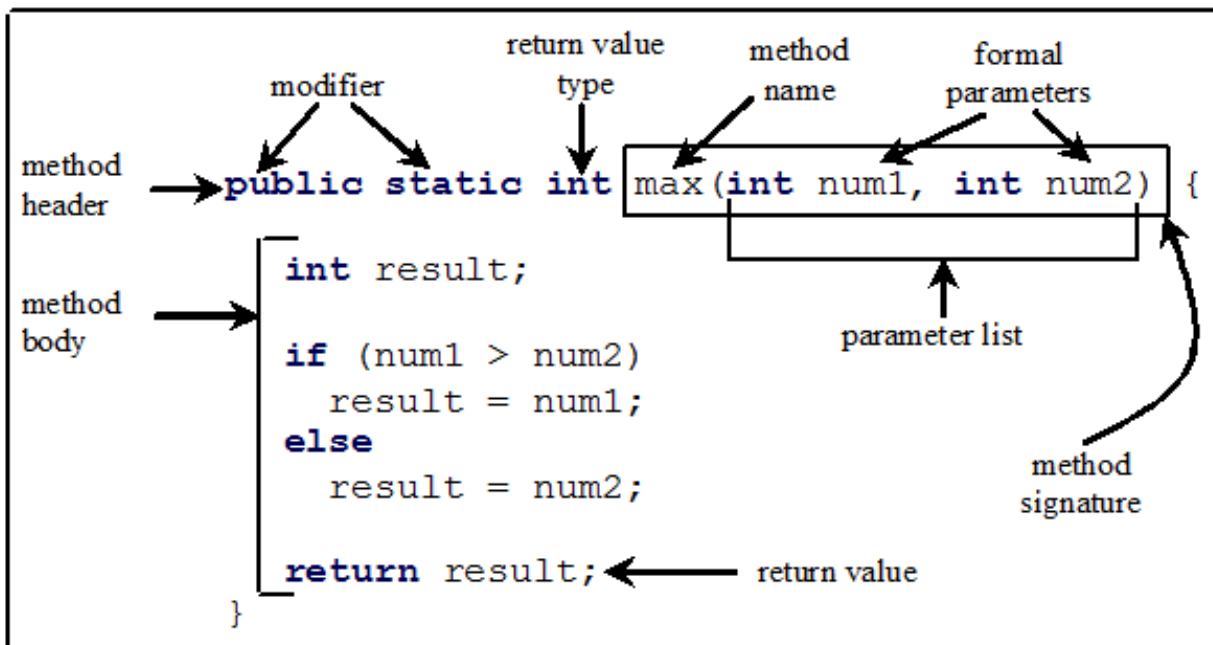
Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All



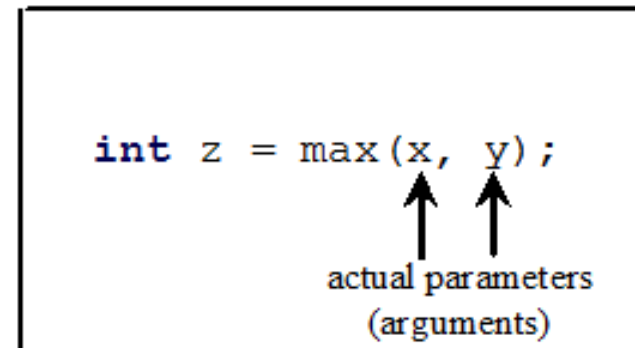
Defining Methods

❖ A method is a collection of statements that are grouped together to perform an operation.

Define a method



Invoke a method



CAUTION

- ❖ A **return** statement is required for a value-returning method.
- ❖ The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

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

(a)

Should be

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

(b)

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

Passing Parameters

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

❖ Suppose you invoke the method using

nPrintln("Welcome to Java", 5);

What is the output?

❖ Suppose you invoke the method using

nPrintln("Computer Science", 15);

What is the output?

❖ Can you invoke the method using

nPrintln(15, "Computer Science");



Ambiguous Invocation

```
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;  
    }  
}
```



Scope of Local Variables

- ❖ A **local variable**: a variable defined inside a method.
- ❖ **Scope**: the part of the program where the variable can be referenced.
- ❖ 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 before it can be used.



Scope of Local Variables

- ❖ You can declare a local variable with the same name multiple times in different **non-nesting** blocks in a method, but you cannot declare a local variable twice in nested blocks.

It is fine to declare `i` in two non-nesting blocks

```
public static void method1() {  
    int x = 1;  
    int y = 1;  
  
    for (int i = 1; i < 10; i++) {  
        x += i;  
    }  
  
    for (int i = 1; i < 10; i++) {  
        y += i;  
    }  
}
```

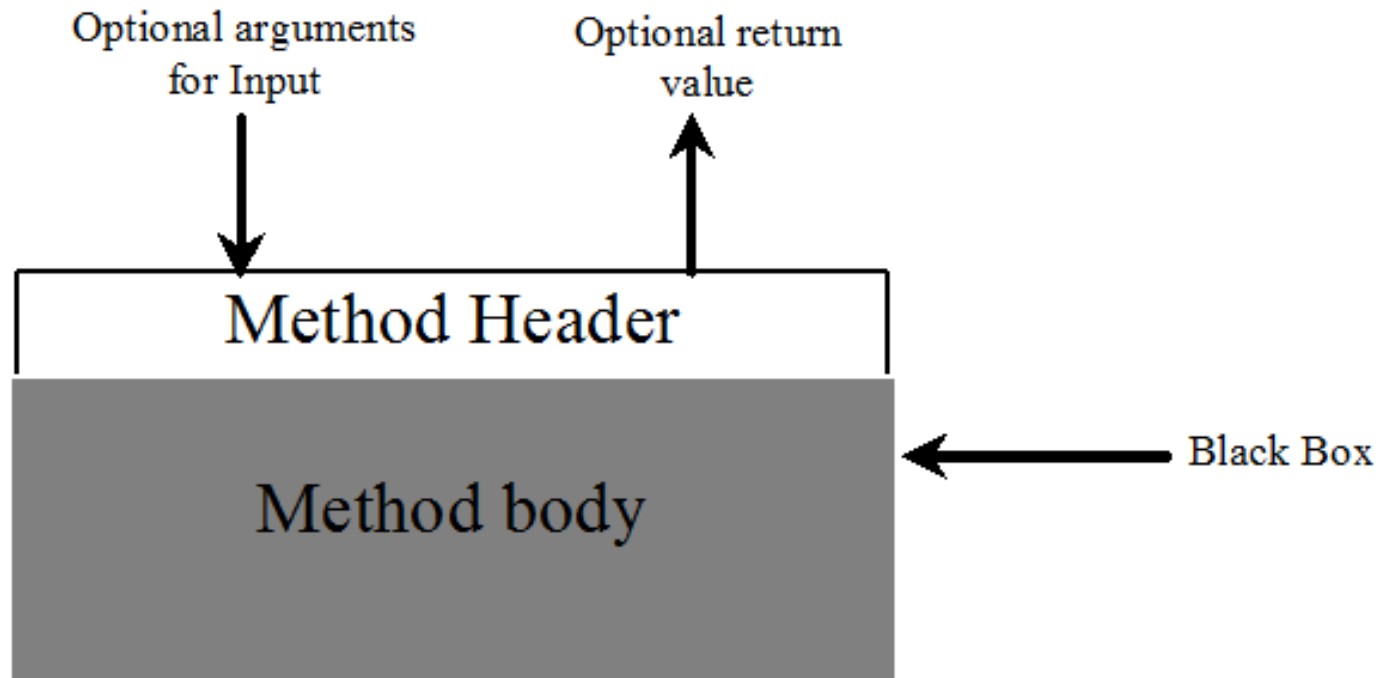
It is wrong to declare `i` in two nesting blocks

```
public static void method2() {  
    int i = 1;  
    int sum = 0;  
  
    for (int i = 1; i < 10; i++)  
    {  
        sum += i;  
    }  
}
```



Method Abstraction

❖ You can think of the method body as a black box that contains the detailed implementation for the method.



Benefits of Methods

- Write a method once and **reuse** it anywhere.
- **Information hiding**. Hide the implementation from the user.
- **Reduce complexity**.



The **Math** Class

❖ Class constants:

- **PI**

- **E**

❖ Class methods:

- Trigonometric Methods

- Exponent Methods

- Rounding Methods

- min, max, abs, and random Methods



Trigonometric Methods

❖ **sin**(double a)

❖ **cos**(double a)

❖ **tan**(double a)

❖ **acos**(double a)

❖ **asin**(double a)

❖ **atan**(double a)

Radians

Math.toRadians(90)

Examples:

Math.sin(0) returns 0.0

Math.sin(Math.PI / 6) returns 0.5

Math.sin(Math.PI / 2) returns 1.0

Math.cos(0) returns 1.0

Math.cos(Math.PI / 6) returns 0.866

Math.cos(Math.PI / 2) returns 0.0



Exponent Methods

- ❖ **exp(double a)**
Returns **e** raised to the power of a.
- ❖ **log(double a)**
Returns the natural logarithm of a.
- ❖ **log10(double a)**
Returns the 10-based logarithm of a.
- ❖ **pow(double a, double b)**
Returns a raised to the power of b.
- ❖ **sqrt(double a)**
Returns the square root of a.

Examples:

<code>Math.exp(1)</code>	returns 2.71
<code>Math.log(2.71)</code>	returns 1.0
<code>Math.pow(2, 3)</code>	returns 8.0
<code>Math.pow(3, 2)</code>	returns 9.0
<code>Math.pow(3.5, 2.5)</code>	returns 22.917
<code>Math.sqrt(4)</code>	returns 2.0
<code>Math.sqrt(10.5)</code>	returns 3.24



Rounding Methods

- ❖ **double `ceil`(double x)** x rounded up to its nearest integer. This integer is returned as a double value.
- ❖ **double `floor`(double x)** x is rounded down to its nearest integer. This integer is returned as a double value.
- ❖ **double `rint`(double x)** x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.
- ❖ **int `round`(float x)** Return (int)Math.floor(x+0.5).
- ❖ **long `round`(double x)** Return (long)Math.floor(x+0.5).



min, max, and abs

❖ **max(a, b)** and **min(a, b)**

Returns the maximum or minimum of two parameters.

❖ **abs(a)**

Returns the absolute value of the parameter.

❖ **random()**

Returns a random double value in the range [0.0, 1.0).

Examples:

Math.max(2, 3) returns 3

Math.max(2.5, 3) returns 3.0

Math.min(2.5, 3.6) returns 2.5

Math.abs(-2) returns 2

Math.abs(-2.1) returns 2.1



The **random** Method

❖ Generates a random **double** value greater than or equal to 0.0 and less than 1.0

$(0 \leq \text{Math.random()} < 1.0)$

`(int) (Math.random() * 10)` → Returns a random integer between 0 and 9.

`50 + (int) (Math.random() * 50)` → Returns a random integer between 50 and 99.

In general:

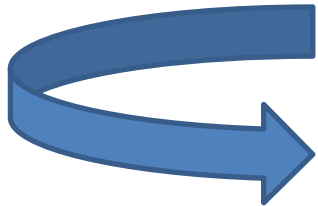
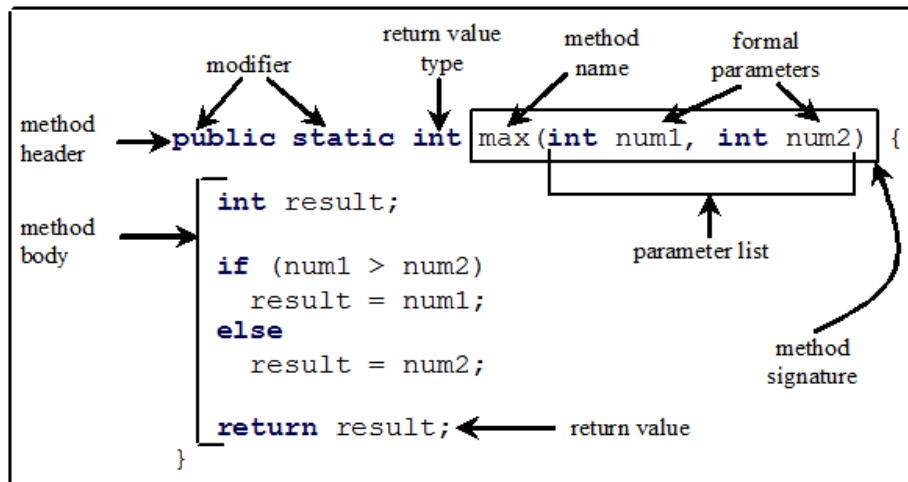
`a + Math.random() * b` → Returns a random number between a and a + b, excluding a + b.



Overloading Methods

❖ *Overloading methods enables you to define the methods with the same name as long as their signatures are different.*

Define a method



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



```
1 public class TestMethodOverloading {
2     /** Main method */
3     public static void main(String[] args) {
4         // Invoke the max method with int parameters
5         System.out.println("The maximum of 3 and 4 is "
6             + max(3, 4));
7
8         // Invoke the max method with the double parameters
9         System.out.println("The maximum of 3.0 and 5.4 is "
10             + max(3.0, 5.4));
11
12         // Invoke the max method with three double parameters
13         System.out.println("The maximum of 3.0, 5.4, and 10.14 is "
14             + max(3.0, 5.4, 10.14));
15     }
16
17     /** Return the max of two int values */
18     public static int max(int num1, int num2) {
19         if (num1 > num2)
20             return num1;
21         else
22             return num2;
23     }
24
25     /** Find the max of two double values */
26     public static double max(double num1, double num2) {
27         if (num1 > num2)
28             return num1;
29         else
30             return num2;
31     }
32
33     /** Return the max of three double values */
34     public static double max(double num1, double num2, double num3) {
35         return max(max(num1, num2), num3);
36     }
37 }
```

