11. Methods

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- Creating a Method
- Calling a Method
- Passing Parameters
- Overloading Methods
- The Scope of Local Variables
- Method Abstraction
- References

Creating a Method

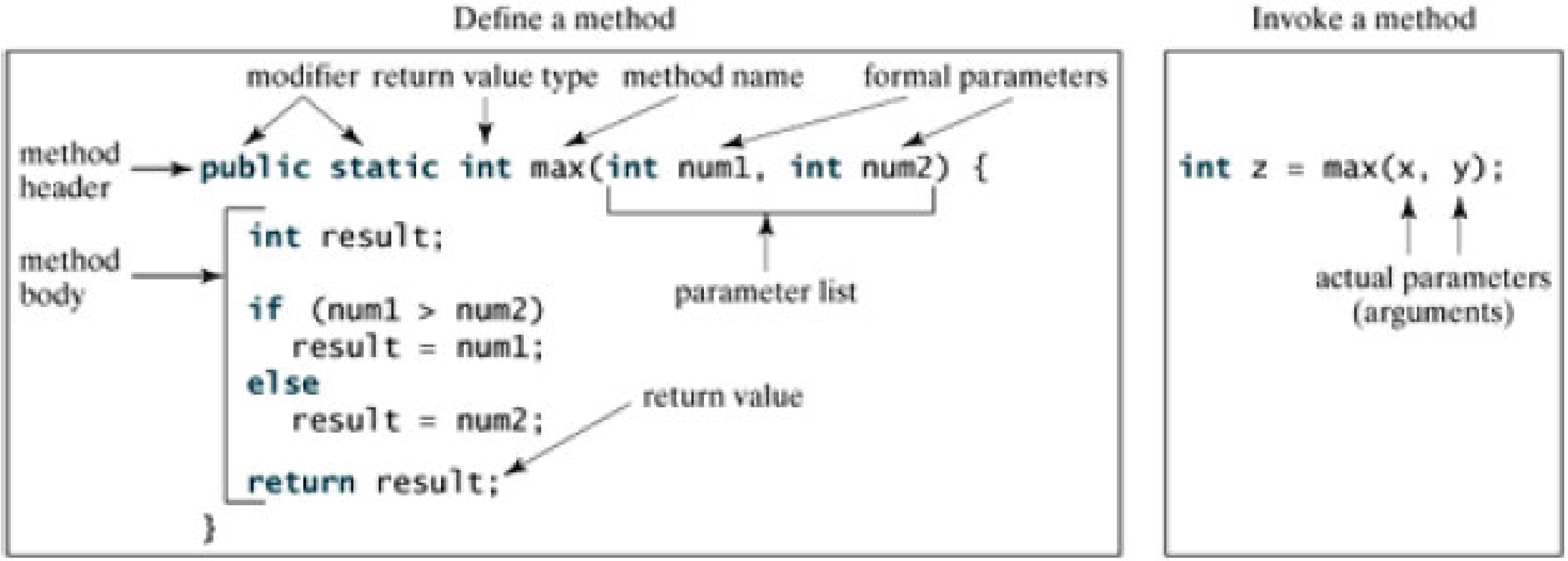
Creating a Method

- A method is a collection of statements that are grouped together to perform an operation.
- Methods called **functions** or **procedures** in other languages
- In general, a method has the following syntax:

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

Creating a Method

- A method created to find which of two integers is bigger.



Define a method

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

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

  return result;
}
```

Invoke a method

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

The Components of a Method

- Method declarations have six components, in order:
  - Modifiers
  - The return type
  - The method name
  - The parameters
  - An exception list (discussed later)
  - The method body

The Modifiers

- The **modifier**, which is optional, tells the compiler how to call the method.

The Return Type

- A method may return a value.
- The returnValueType is the data type of the value the method returns.
- If the method does not return a value, the returnValueType is the keyword void.
- For example, the returnValueType in the main method is void.

The Return Type

- The method types:
  - **Nonvoid method:** A method that returns a value
  - **Void method:** A method that that does not return a value

- In other languages,
  - **Function**: a method with a nonvoid return value type
  - **Procedure**: a method with a void return value type

The Method Name

- A method name can be any legal identifier
- By convention,
  - the first (or only) word in a method name should be a verb in lowercase.
  - In a multiword name that begins with a verb in lowercase, followed by adjectives, nouns, etc.
  - In multiword names, the first letter of each of the second and following words should be capitalized.

The Method Name

- Here are some examples:

  run

  runFast

  getBackground

  getFinalData

  compareTo

  setX

  isEmpty

The Parameters

- The variables defined in the method header are known as **formal parameters**.
  - You need to declare a separate data type for each parameter.
  - For instance,
    - Wrong: int num1, num2
    - Correct: int num1, int num2.
- When a method is invoked, you pass a value to the parameter. This value is referred to as **actual parameter** or **argument**.

Method signature

- Two of the components of a method declaration comprise the **method signature:**
  - the method's name
  - the parameter list
- An example of a method declaration:

```
public double calculateAnswer(double wingSpan, int
    numberOfEngines, double length, double grossTons)
{
    // do the calculation here
}
```

- The signature of the method declared above is:

```
calculateAnswer(double, int, double, double)
```

Method Body

- The **method body** contains a collection of statements that define what the method does.
- The method terminates when a return statement is executed.
- The keyword return is required for a nonvoid method to return a result.

Calling a Method

Calling a Method

- To use a method, you have to call or invoke it.
- There are two ways to call a method.
- If the method returns a value, a call to the method is usually treated as a value. For example:

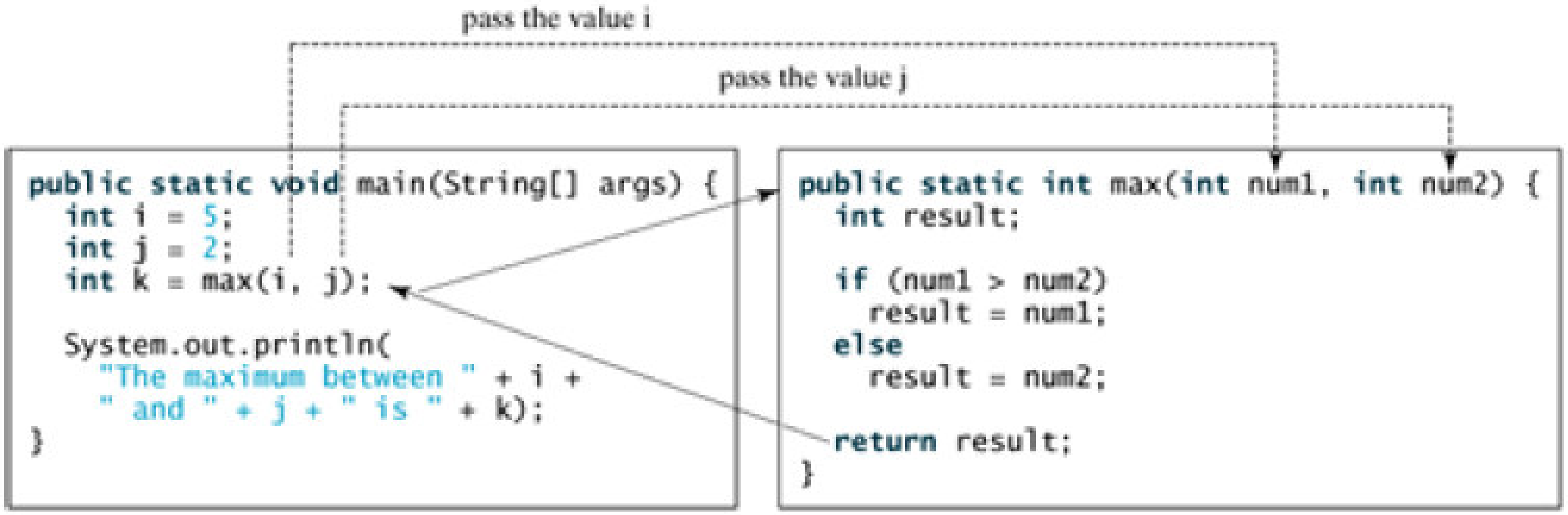
```
int larger = max(3, 4);
System.out.println(max(3, 4));
```

- If the method returns void, a call to the method must be a statement. For example:

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

Calling a Method

- Example: TestMax.java
- When the max method is invoked, the flow of control transfers to the max method. Once the max method is finished, it returns the control back to the caller.



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

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

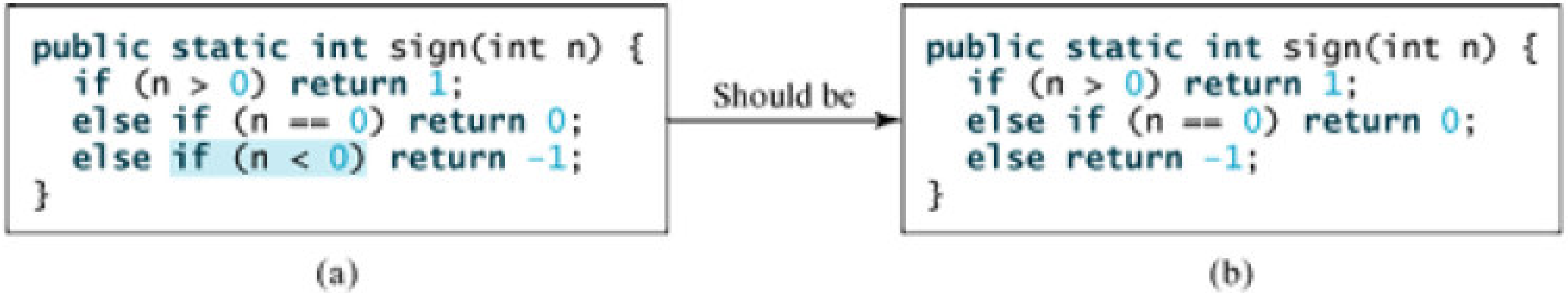
```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

Caution

- A return statement is required for a nonvoid method.
- The method shown left below in (a) is logically correct, but it has a compilation error:



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

(a)

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

(b)

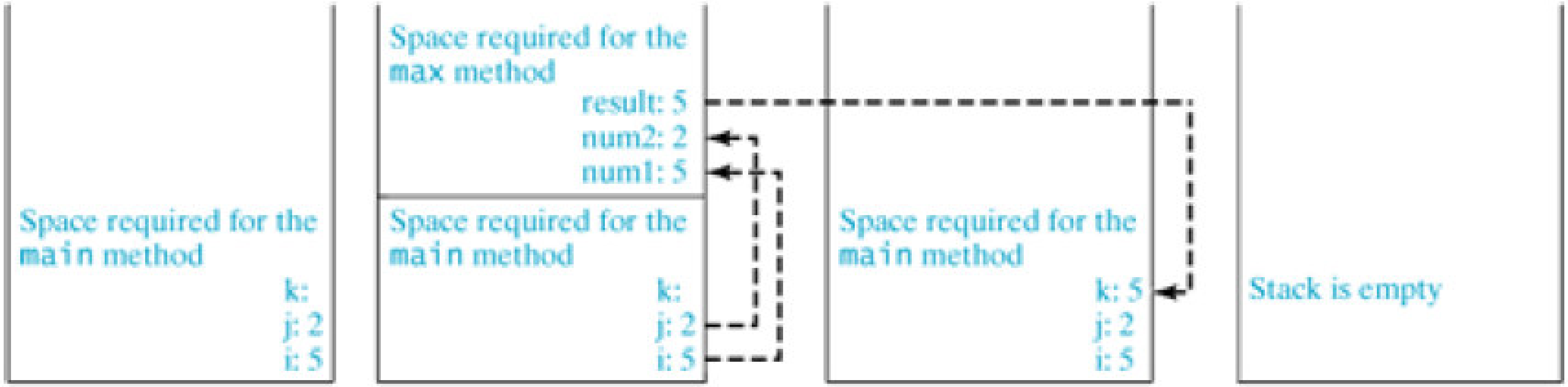
Reuse Methods from Other Classes

- One of the benefits of methods is for reuse.
- The max method can be invoked from any class besides TestMax.
- You can invoke the max method from other classes using ClassName.methodName
  - i.e., TestMax.max

Call Stacks

- **Stack**
  - Each time a method is invoked, the system stores parameters and variables in an area of memory, known as a **stack**, which stores elements in last-in first-out fashion.
- 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.

Call Stacks


Space required for the main method
k:
j: 2
i: 5
Space required for the max method
result: 5
num2: 2
num1: 5
Space required for the main method
k:
j: 2
i: 5
Space required for the main method
k: 5
j: 2
i: 5
Stack is empty

(a) The main method is invoked.

(b) The max method is invoked.

(c) The max method is finished and the return value is sent to k.

(d) The main method is finished.

A void Method Example

- An example of void method:
  - TestVoidMethod.java

Passing Parameters

Passing Parameters

- The arguments must match the parameters in order, number, and compatible type, as defined in the method signature.
- When you invoke a method with a parameter, the value of the **argument** is passed to the **parameter**.
- This is referred to as **pass-by-value**.

Passing Parameters

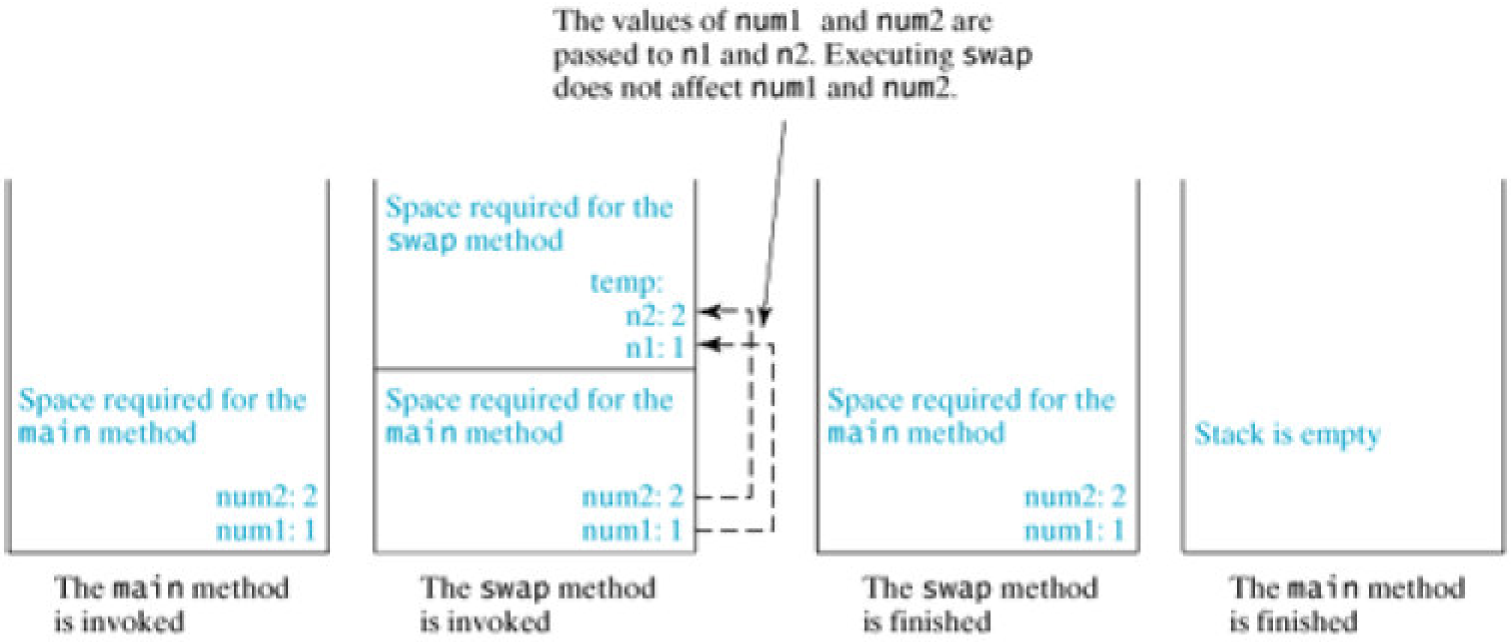
- If the argument is a **variable** rather than a **literal** value, the value of the variable is passed to the parameter.
- The variable is not affected, regardless of the changes made to the parameter inside the method.

Passing Parameters

- Example:
  - TestPassByValue.java
- The program output:

```
Before invoking the swap method, num1 is 1 and num2 is 2
    Inside the swap method
        Before swapping n1 is 1 n2 is 2
        After swapping n1 is 2 n2 is 1
After invoking the swap method, num1 is 1 and num2 is 2
```

Passing Parameters


The values of num1 and num2 are passed to n1 and n2. Executing swap does not affect num1 and num2.
Space required for the main method
num2: 2
num1: 1
The main method is invoked
Space required for the swap method
temp:
n2: 2
n1: 1
Space required for the main method
num2: 2
num1: 1
The swap method is invoked
Space required for the main method
num2: 2
num1: 1
The swap method is finished
Stack is empty
The main method is finished

Passing Parameters

- Another case is to change the parameter name n1 in swap to num1.
- What effect does this have?
- No change occurs because it makes no difference whether the parameter and the argument have the same name.
- For simplicity, Java programmers often say
  - passing an argument x to a parameter y,
  - which actually means passing the value of x to y.

Overloading Methods

Overloading Methods

- **Method overloading** is referred when two methods have the same name but different parameter lists within one class.
- Java can distinguish between methods with different method signatures.
- The Java compiler determines which method is used based on the method signature.

Overloading Methods

- The max method that was used earlier works only with the int data type.
- But what if you need to find which of two floating-point numbers has the maximum value?
- The solution is to create another method with the same name but different parameters.

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

Overloading Methods

- Example:
  - TestMethodOverloading.java
- The program output:

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

Overloading Methods

- Can you invoke the max method with an int value and a double value, such as max(2, 2.5)?
- Yes, 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.

Ambiguous invocation

- Sometimes there are two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.
- This is referred to as **ambiguous invocation**.
- Ambiguous invocation causes a **compilation error**.
- Example:
  - AmbiguousOverloading.java

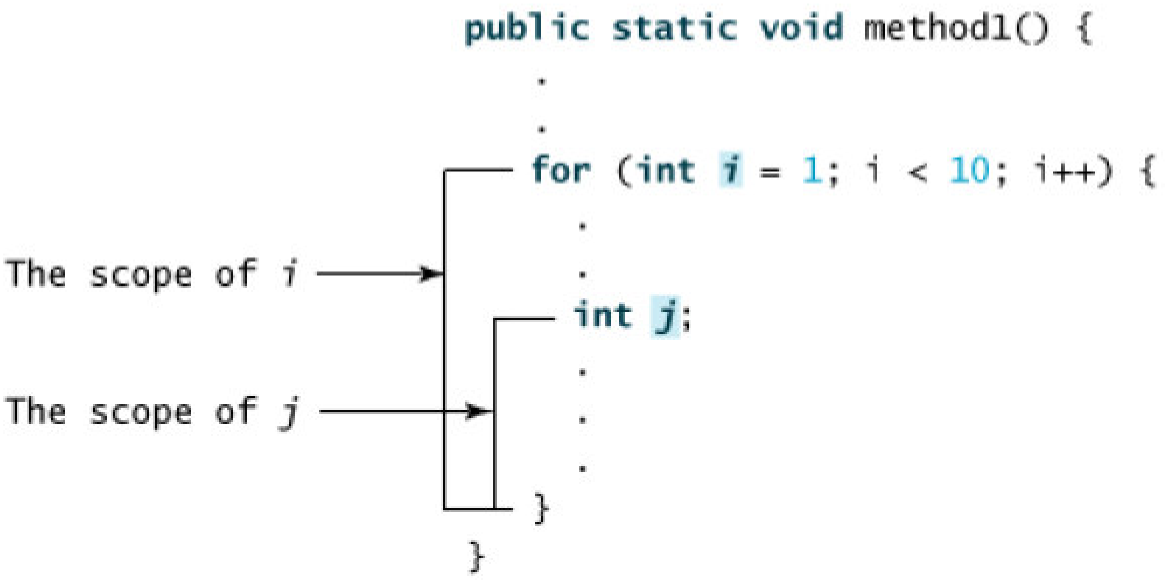
The Scope of Local Variables

The Scope of Local Variables

- The **scope of a variable** is the part of the program where the variable can be referenced.
- variable defined inside a method is referred to as 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 parameter** is actually a local variable. The scope of a method parameter covers the entire method.

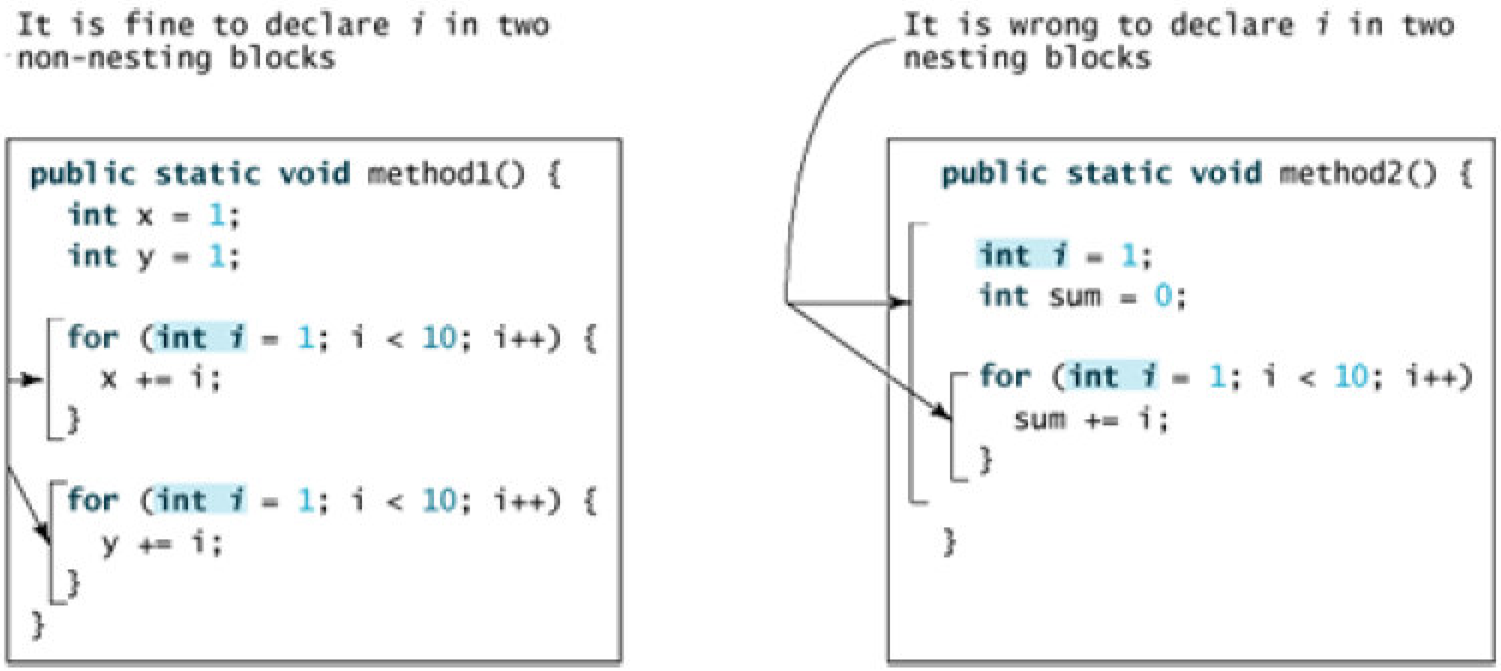
The Scope of Local Variables

- A variable declared in the initial action part of a for loop header has its scope in the entire loop.
- But a variable declared inside a for loop body has its scope limited in the loop body from its declaration to the end of the block that contains the variable



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

The Scope of Local Variables



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;
  }

}
```

- 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

The Scope of Variables

- Do not declare a variable inside a block and then attempt to use it outside the block. Here is an example of a common mistake:

```
for (int i = 0; i < 10; i++)
{
 }
System.out.println(i);
```

- The last statement would cause a syntax error because variable i is not defined outside of the for loop.

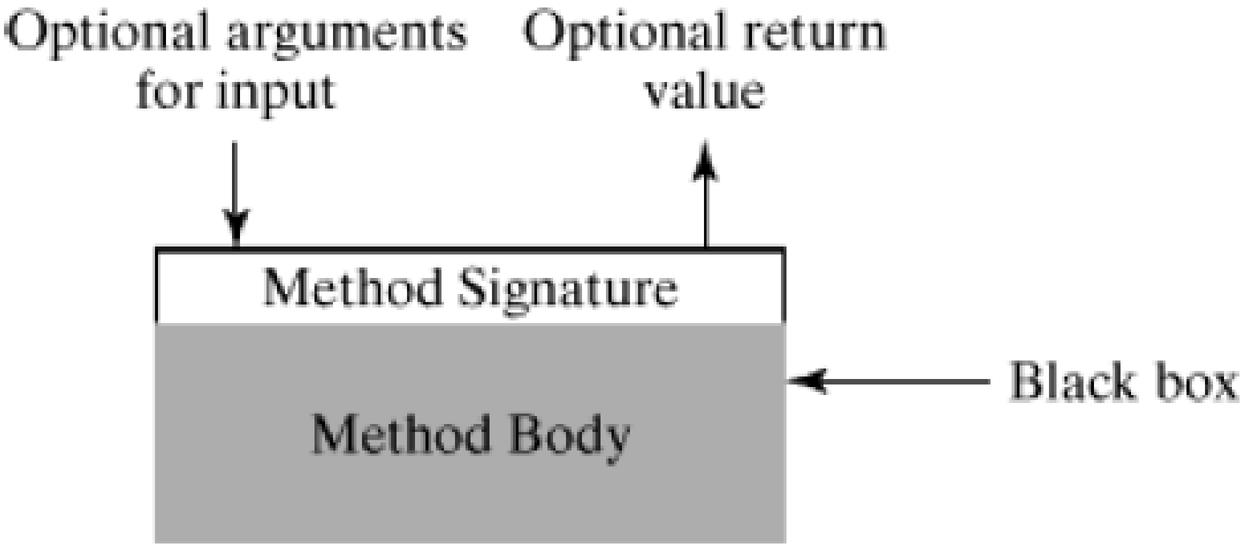
Method Abstraction

Method Abstraction

- **Method abstraction** is achieved by separating the **use of a method** from **its implementation**.
- 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**.
- If you decide to change the implementation, the client program will not be affected, provided that you do not change the method signature.

Method Abstraction

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



Method Abstraction

- You have already used the System.out.println method to print.
- You know how to write the code to invoke this method in your program, but as a user of this method, you are not required to know how it is implemented.

Method Abstraction

- The concept of method abstraction can be applied to the process of developing programs.
- When writing a large program, you can use the "**divide and conquer**" strategy, also known as **stepwise refinement**, to decompose it into subproblems.
- The subproblems can be further decomposed into smaller, more manageable problems.

References

References

- Y. Daniel Liang, **Introduction to Java Programming**, Sixth Edition, Pearson Education, 2007. (Chapter 5)
- S. Zakhour and et el., **The Java Tutorial: A Short Course on the Basics**, 4th Edition, Prentice Hall, 2006. (Chapter 4)

The End