



Chapter 6 Methods

Opening Problem

Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.



Problem

```
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 <= 30; i++)  
    sum += i;  
System.out.println("Sum from 20 to 30 is " + sum);
```

```
sum = 0;  
for (int i = 35; i <= 45; i++)  
    sum += i;  
System.out.println("Sum from 35 to 45 is " + sum);
```



Problem

```
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 <= 30; i++)  
    sum += i;
```

```
System.out.println("Sum from 20 to 30 is " + sum);
```

```
sum = 0;  
for (int i = 35; i <= 45; i++)  
    sum += i;
```

```
System.out.println("Sum from 35 to 45 is " + sum);
```



Solution

```
public static int sum(int i1, int i2) {  
    int sum = 0;  
    for (int i = i1; i <= i2; i++)  
        sum += i;  
    return sum;  
}
```

MethodDemo

```
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 30 is " + sum(20, 30));  
    System.out.println("Sum from 35 to 45 is " + sum(35, 45));  
}
```

Objectives

- Defining a method
- Calling a method
- Void method
- Passing parameter by values
- Overloading method
- Scope of variables



Defining Methods

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

Define a method

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

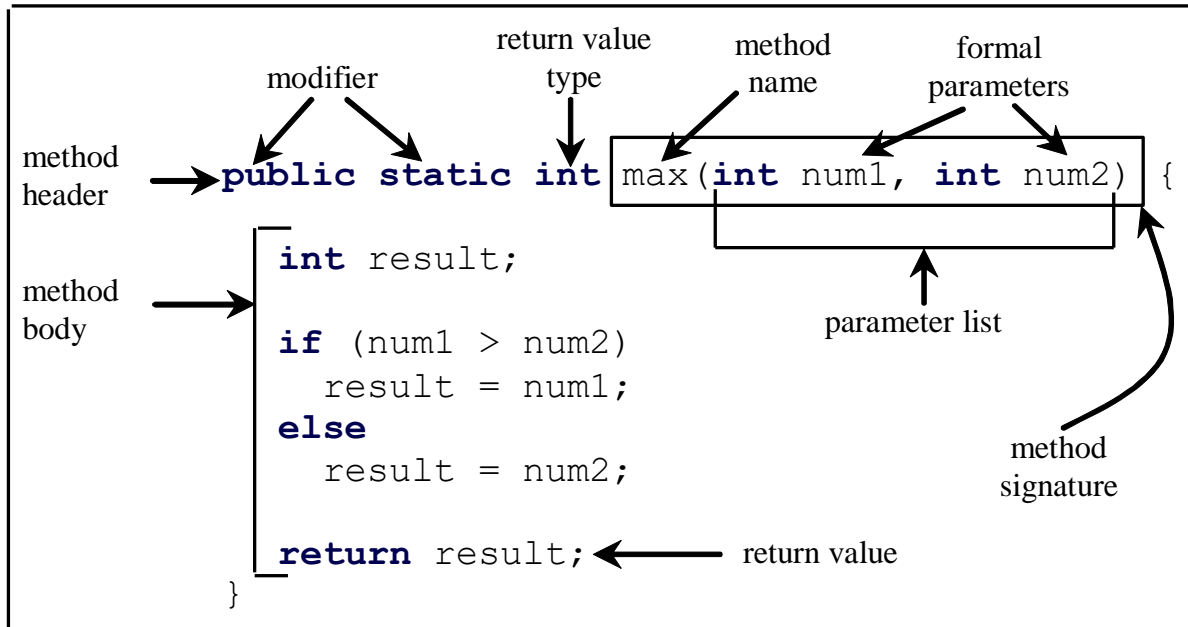
Invoke a method

```
int z = max(x, y);  
          ↑  ↑  
    actual parameters  
    (arguments)
```

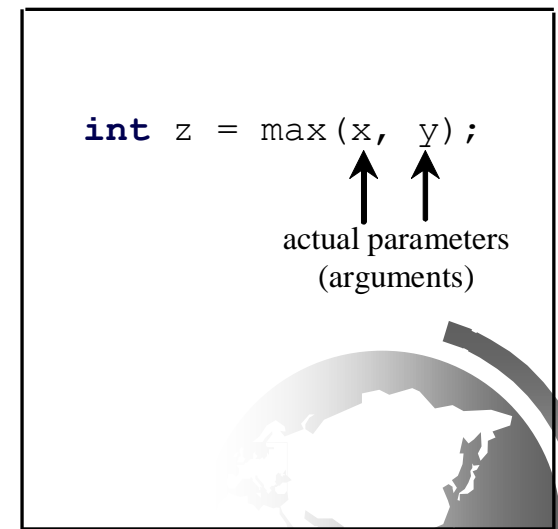
Defining Methods

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

Define a method



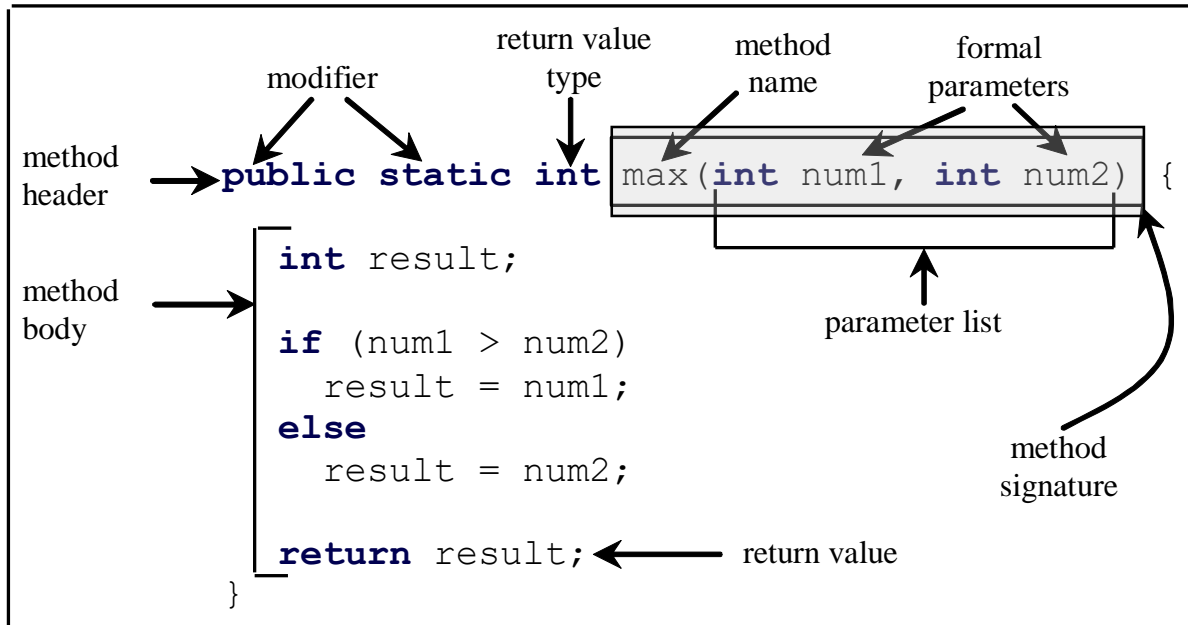
Invoke a method



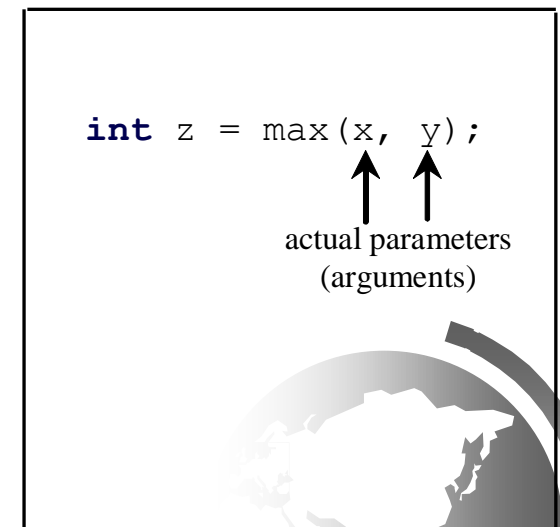
Method Signature

Method signature is the combination of the method name and the parameter list.

Define a method



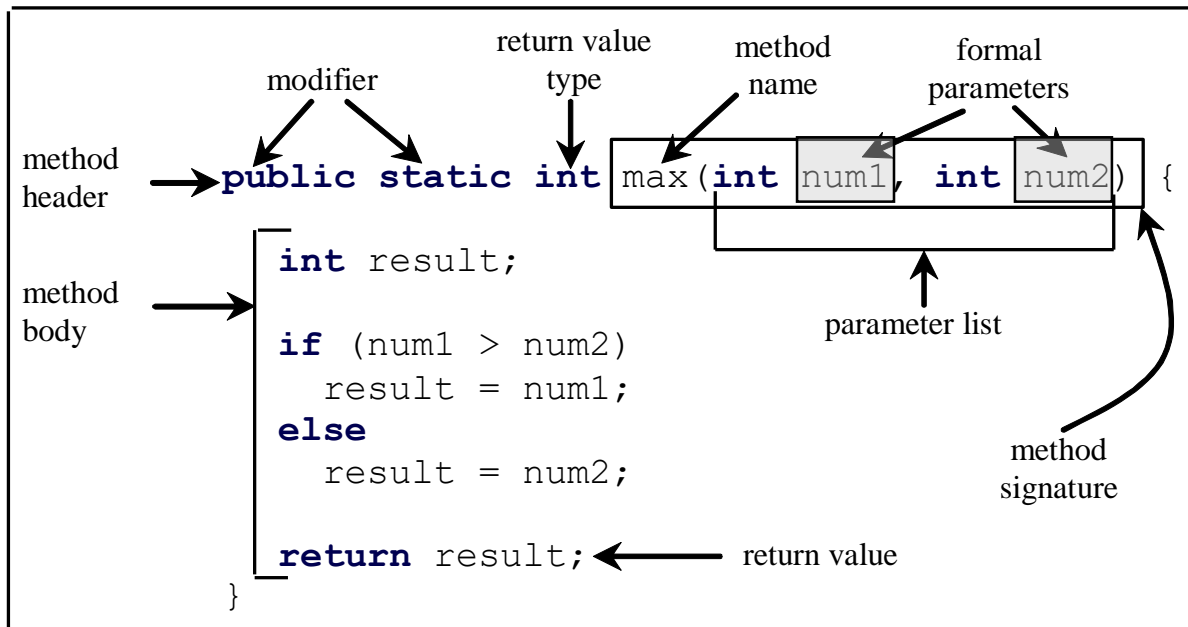
Invoke a method



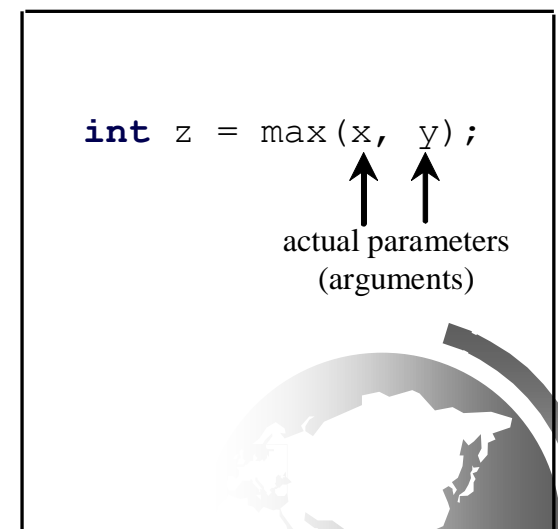
Formal Parameters

The variables defined in the method header are known as *formal parameters*.

Define a method



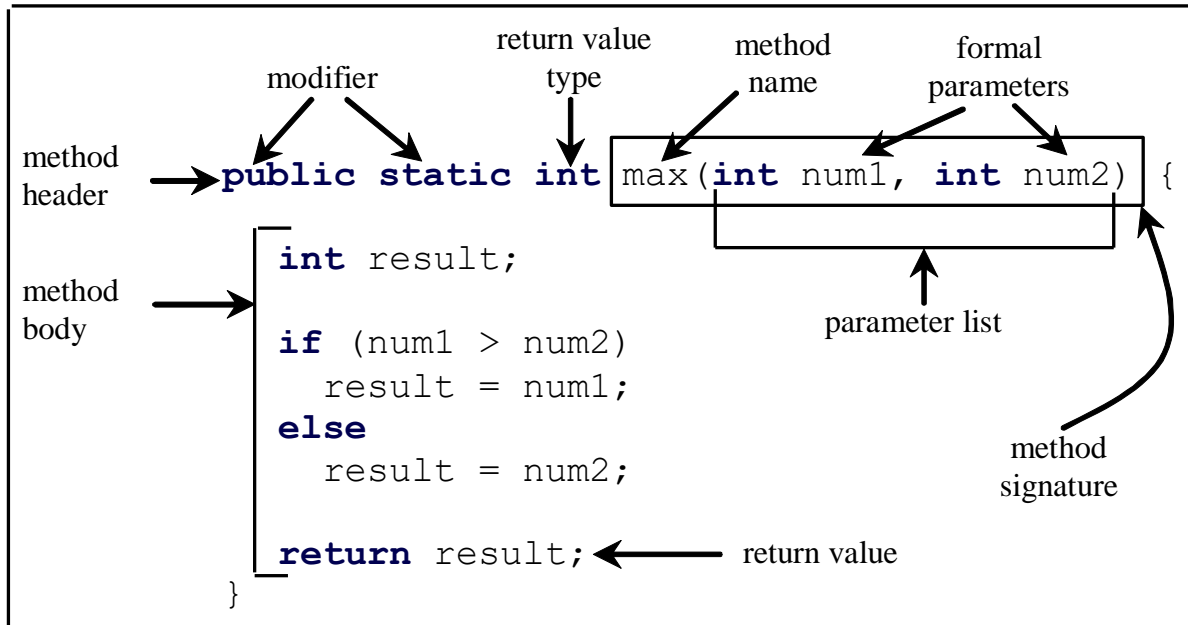
Invoke a method



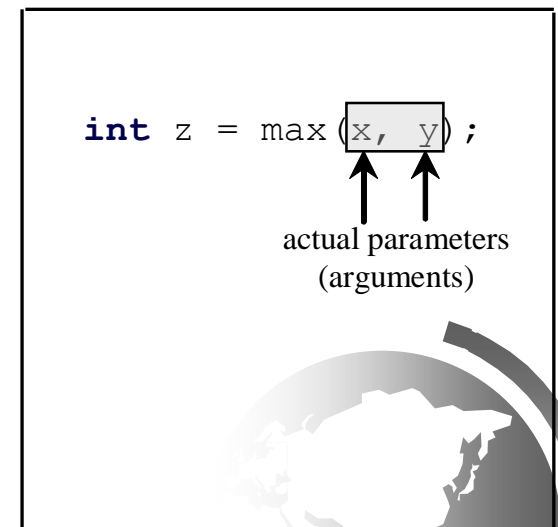
Actual Parameters

When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter* or *argument*.

Define a method



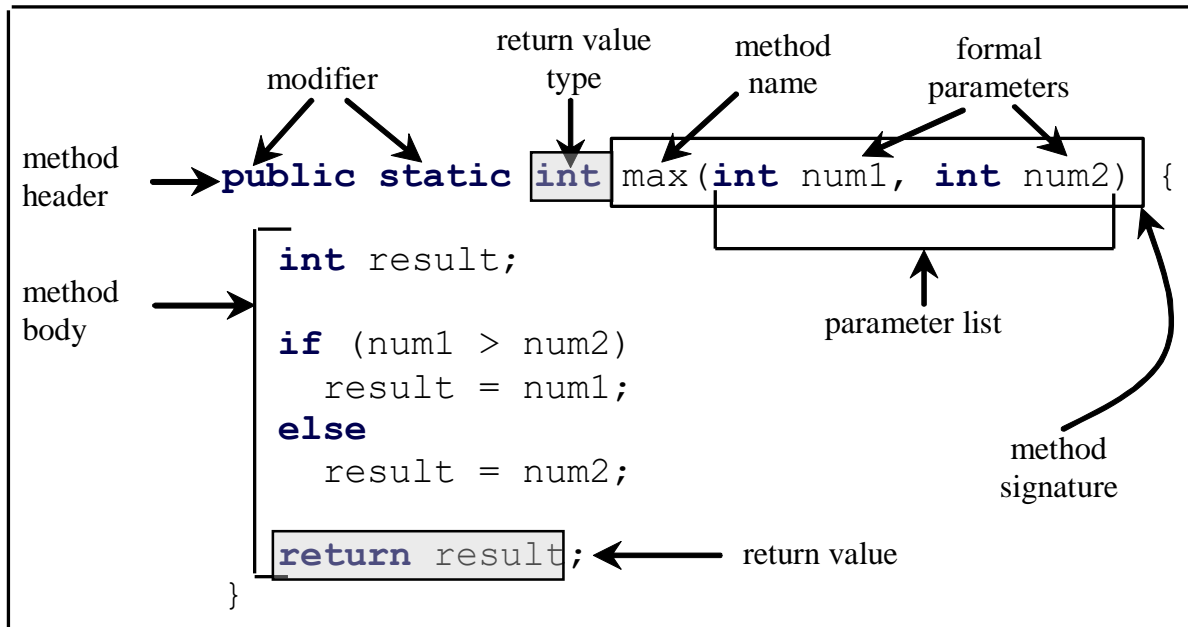
Invoke a method



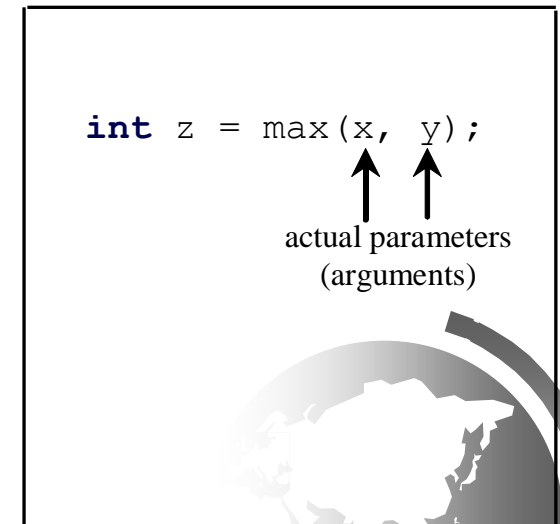
Return Value Type

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.

Define a method



Invoke a method



Calling Methods

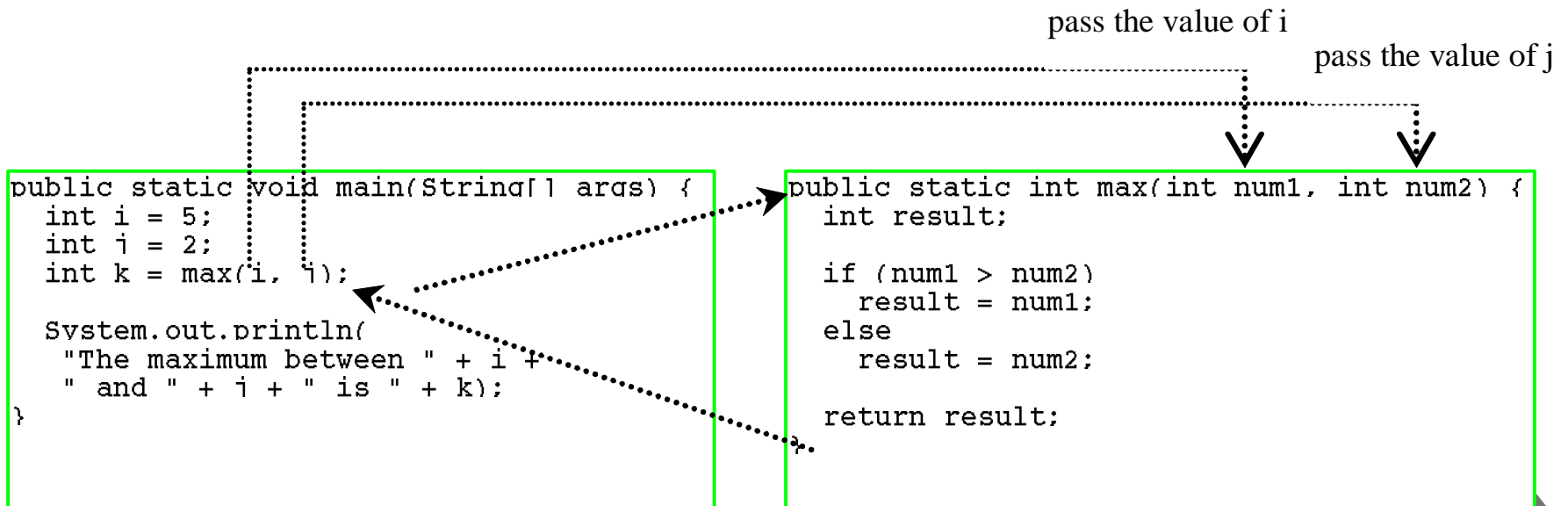
Testing the `max` method

This program demonstrates calling a method `max` to return the largest of the `int` values

TestMax



Calling Methods, cont.



Trace Method Invocation

i is now 5

```
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;  
}
```



Trace Method Invocation

j is now 2

```
public static void main(String[] args) {  
    int i = 5;  
    int i = 2;  
    int k = max(i, i);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + i + " is " + k);  
}
```

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



Trace Method Invocation

invoke max(i, j)

```
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;  
}
```



Trace Method Invocation

invoke max(i, j)
Pass the value of i to num1
Pass the value of j to num2

```
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;  
}
```



Trace Method Invocation

declare variable result

```
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;  
}
```



Trace Method Invocation

(num1 > num2) is true since num1
is 5 and num2 is 2

```
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;  
}
```



Trace Method Invocation

result is now 5

```
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;  
}
```



Trace Method Invocation

return result, which is 5

```
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;  
}
```



Trace Method Invocation

return max(i, j) and assign the
return value to k

```
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;  
}
```



Trace Method Invocation

Execute the print statement

```
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 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)

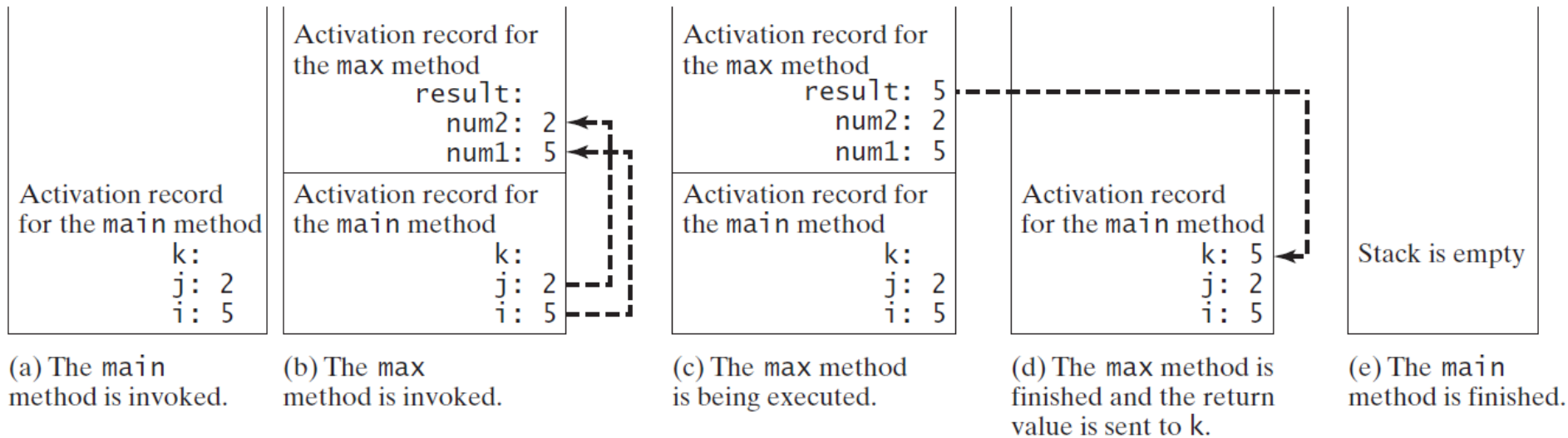
To fix this problem, delete if ($n < 0$) in (a), so that the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

Reuse Methods from Other Classes

NOTE: One of the benefits of methods is for reuse. The max method can be invoked from any class besides TestMax. If you create a new class Test, you can invoke the max method using ClassName.methodName (e.g., TestMax.max).



Call Stacks



Trace Call Stack

i is declared and initialized

```
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;  
}
```

i: 5

The main method
is invoked.

Trace Call Stack

j is declared and initialized

```
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;  
}
```

j: 2
i: 5

The main method
is invoked.

Trace Call Stack

Declare k

```
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;  
}
```

Space required for the
main method

k:
j: 2
i: 5

The main method
is invoked.

Trace Call Stack

Invoke max(i, j)

```
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;  
}
```

Space required for the
main method

k:
j: 2
i: 5

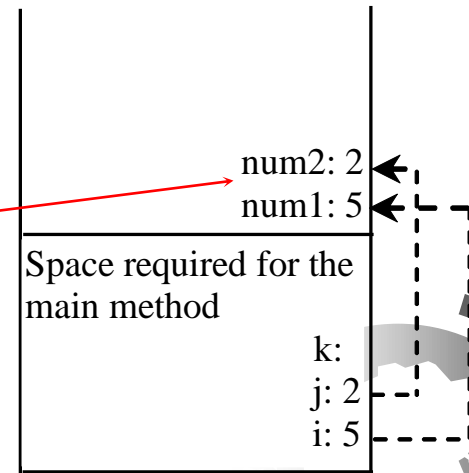
The main method
is invoked.

Trace Call Stack

```
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);  
}
```

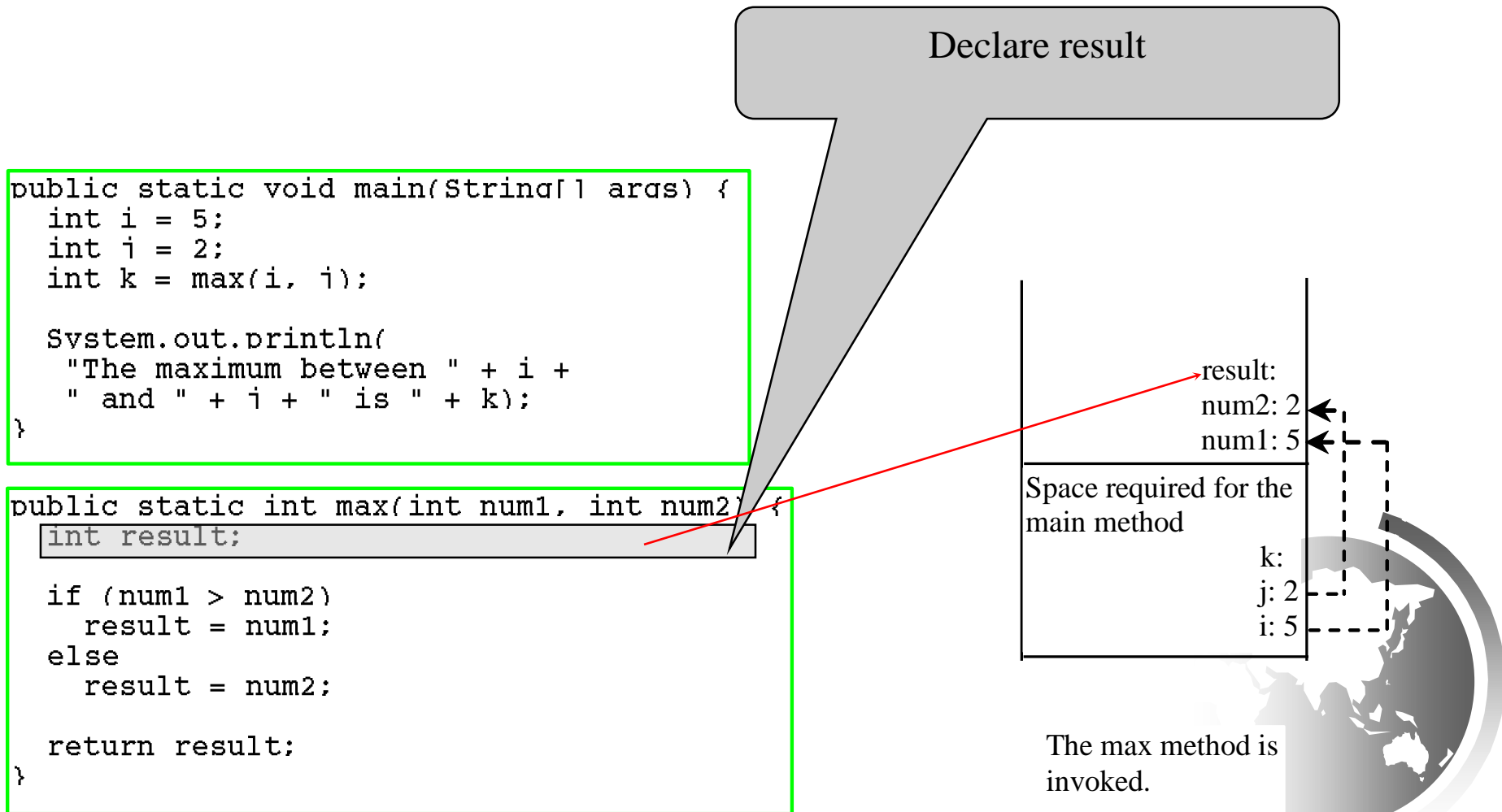
```
public static int max(int num1, int num2) {  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else  
        result = num2;  
  
    return result;  
}
```

pass the values of i and j to num1
and num2



The max method is
invoked.

Trace Call Stack

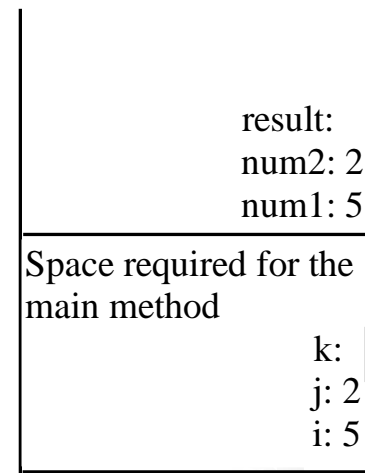


Trace Call Stack

```
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;  
}
```

(num1 > num2) is true



The max method is invoked.

Trace Call Stack

```
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;  
}
```

Assign num1 to result

Space required for the
max method

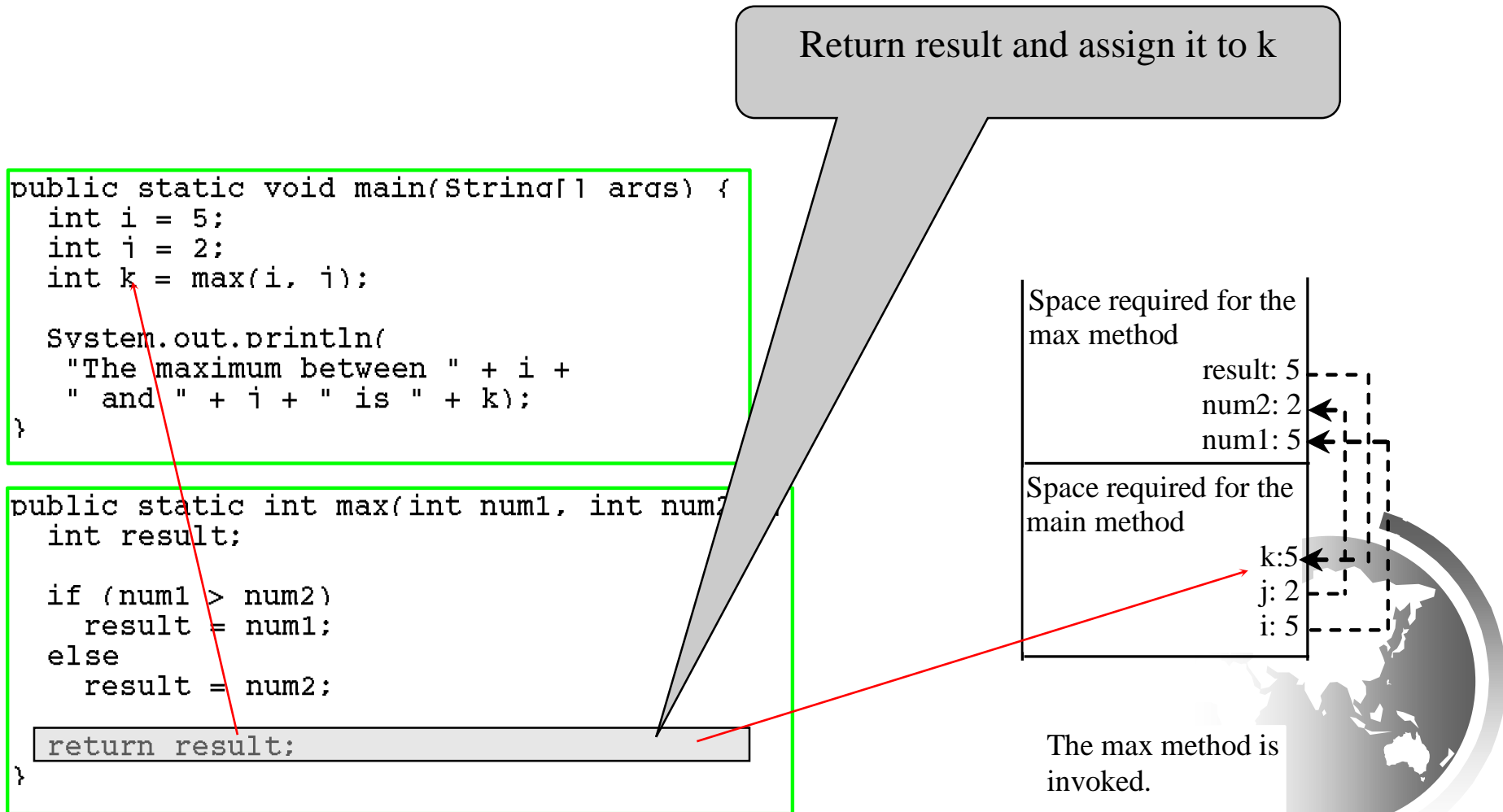
result: 5
num2: 2
num1: 5

Space required for the
main method

k:
j: 2
i: 5

The max method is
invoked.

Trace Call Stack



Trace Call Stack

Execute print statement

```
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;  
}
```

Space required for the
main method

k:5
j: 2
i: 5

The main method
is invoked.

void Method Example

This type of method does not return a value. The method performs some actions.

TestVoidMethod

TestReturnGradeMethod



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”);



Pass by Value

This program demonstrates passing values to the methods.

Increment



Pass by Value

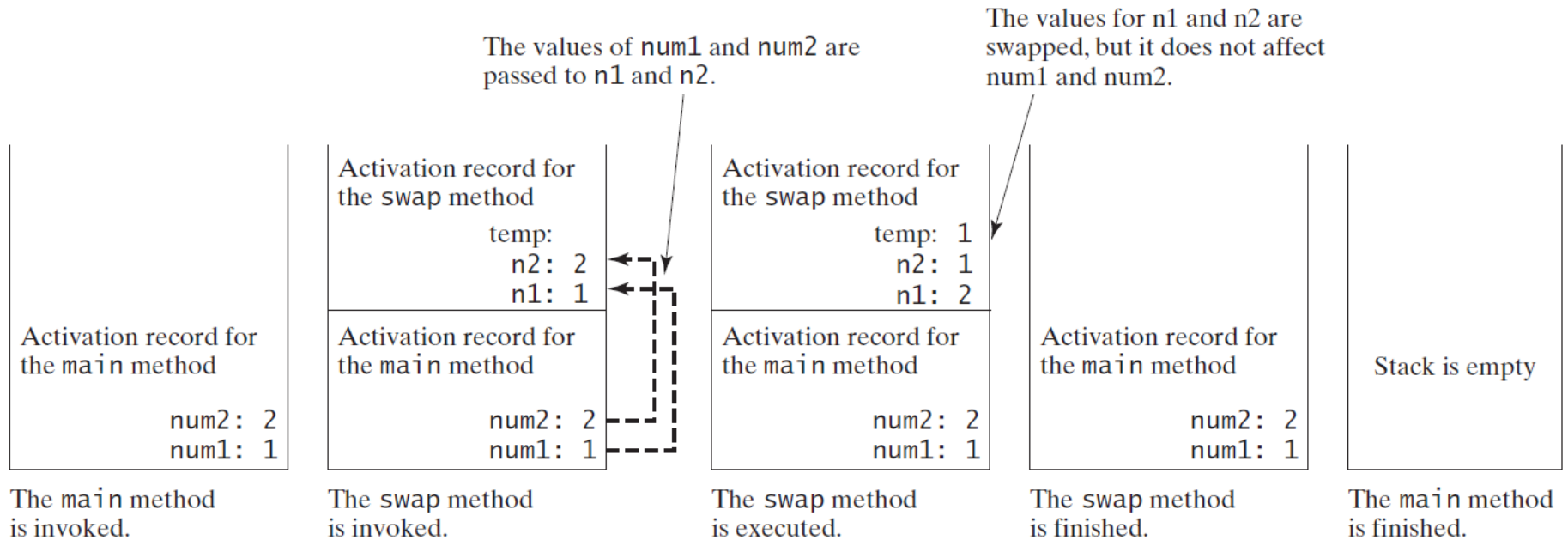
Testing Pass by value

This program demonstrates passing values to the methods.

TestPassByValue



Pass by Value, cont.



Modularizing Code

Methods can be used to reduce redundant coding and enable code reuse. Methods can also be used to modularize code and improve the quality of the program.

GreatestCommonDivisor

PrimeNumber

GreatestCommonDivisorMethod

PrimeNumberMethod



Case Study: Converting Hexadecimals to Decimals

Write a method that converts a hexadecimal number into a decimal number.

ABCD =>

$$A*16^3 + B*16^2 + C*16^1 + D*16^0$$

$$= ((A*16 + B)*16 + C)*16 + D$$

$$= ((10*16 + 11)*16 + 12)*16 + 13 = ?$$

Hex2Dec



Overloading Methods

Overloading methods enable you to define the methods with the same name as long as their parameter lists are different.

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

TestMethodOverloading



Ambiguous Invocation

Sometimes there may be 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 is a compile error.



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, cont.

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.



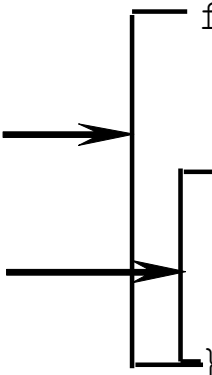
Scope of Local Variables, cont.

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 and 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 i →

The scope of j →



Scope of Local Variables, cont.

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;  
    }  
}
```

Scope of Local Variables, cont.

```
// Fine with no errors
public static void correctMethod() {
    int x = 1;
    int y = 1;
    // i is declared
    for (int i = 1; i < 10; i++) {
        x += i;
    }
    // i is declared again
    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
```



Scope of Local Variables, cont.

// With errors

```
public static void incorrectMethod() {  
    int x = 1;  
    int y = 1;  
    for (int i = 1; i < 10; i++) {  
        int x = 0;  
        x += i;  
    }  
}
```

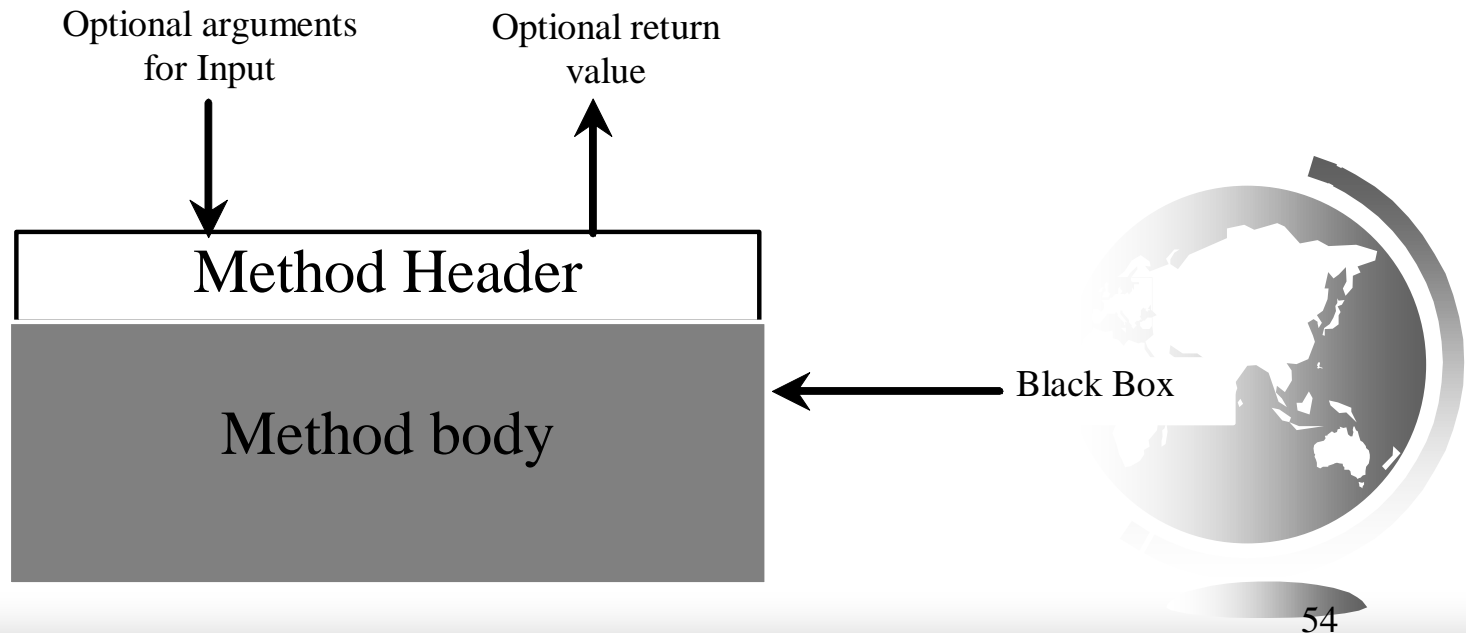


Method Abstraction

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

The details of the implementation are encapsulated in the method and hidden from the user.

This is also known as information hidden or encapsulation.



Benefits of Methods

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

