Methods

Opening Problem

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

A Solution

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

sum = 0;
for (int i = 20; i <= 30; i++)
   sum = sum + i;
System.out.println("Sum from 20 to 30 is " + sum);

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

Repeated Code

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

sum = 0;
for (int i = 20; i <= 30; i++)
   sum = sum + i;
System.out.println("Sum from 20 to 30 is " + sum);

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

Solution Using Method *sum*

```java
public class sumMethod {
  public static void main(String[] args)
  {
    int result = sum(1, 10);
    System.out.println("Sum from 1 to 10 is:\t" + result);

    result = sum(20, 30);
    System.out.println("Sum from 20 to 30 is:\t" + result);

    result = sum(35, 45);
    System.out.println("Sum from 35 to 45 is:\t" + result);
  }

  public static int sum (int num1, int num2)
  {
    int sum = 0;
    for (int i = num1; i <= num2; i++)
        sum = sum + i;
    return sum;
  }
}
```

What is a Method?

Think of a method as a black box that contains the detailed implementation for a specific task. The method may take use inputs (parameters) and may retune an out with a specific type.
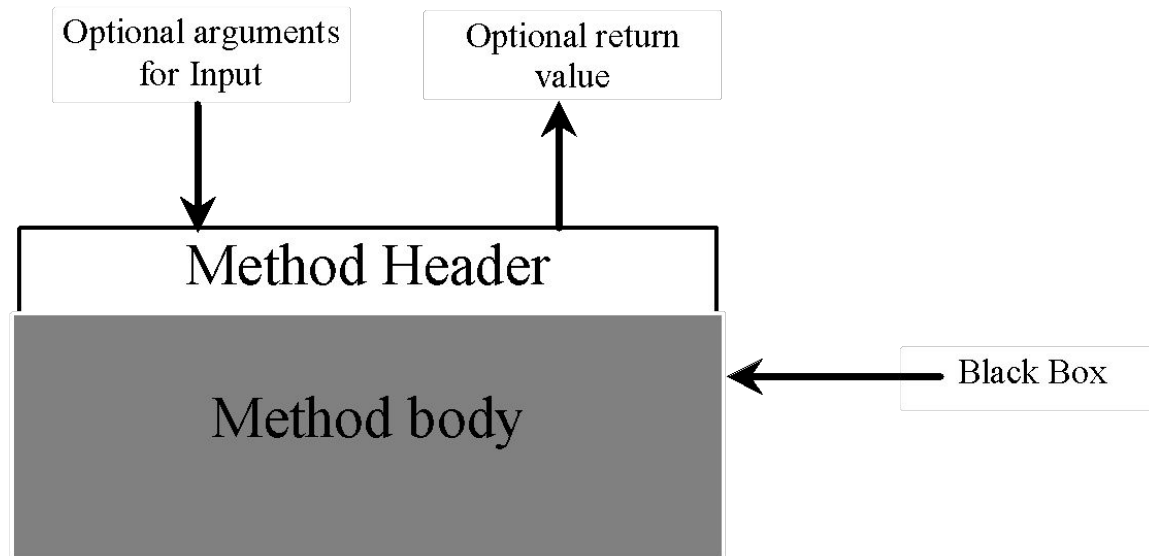
Benefits of Methods

- Write a method once and reuse it anywhere

- Promotes Information hiding (hide the implementation from the user)

- Facilitate modularity (break the code into manageable modules)
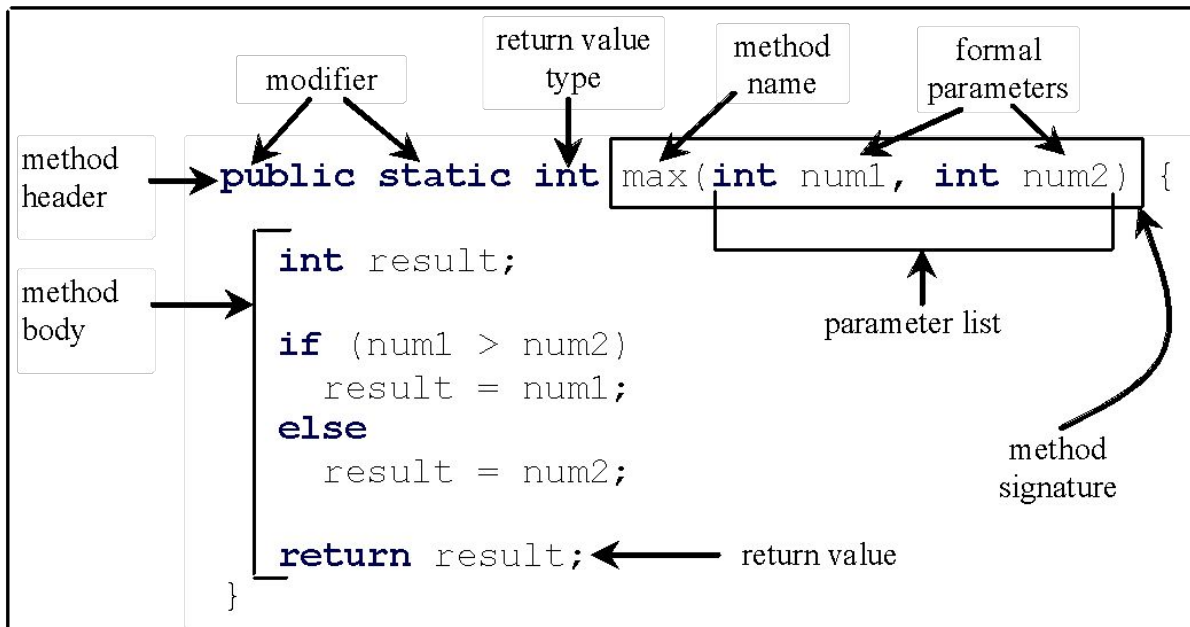
- Reduce code complexity (better maintenance)

Defining Methods

A method has a *header* and a *body*.
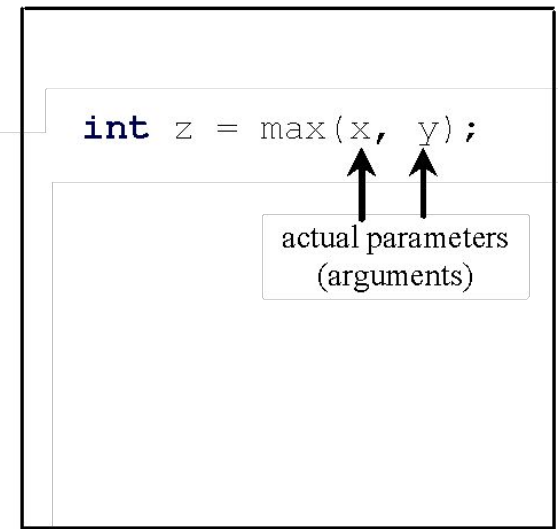
=> The *header* is the method declaration.

=> The *body* is a a collection of statements grouped together to perform an operation.
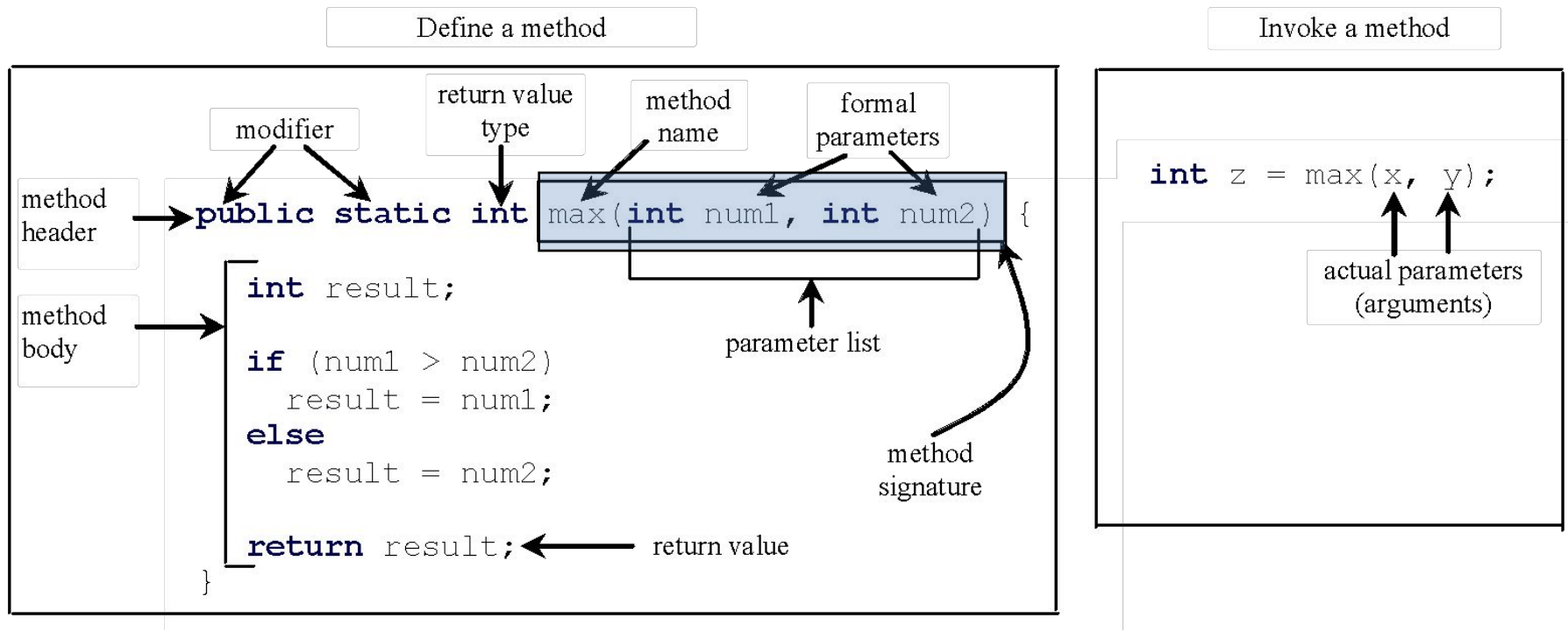
Define a method	Invoke a method

modifier · return value type · method name · formal parameters

method header

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

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

    return result;
}
```

method body

parameter list

method signature

return value

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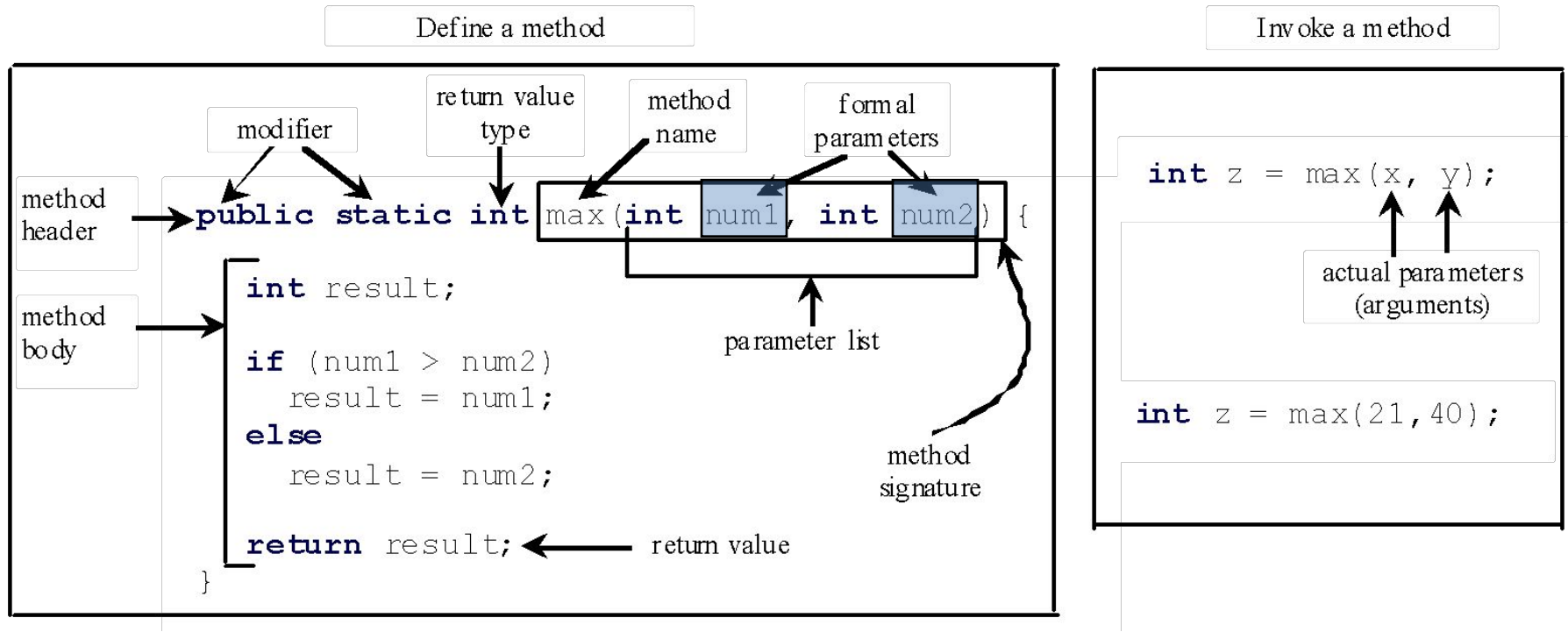
actual parameters (arguments)

Method Signature

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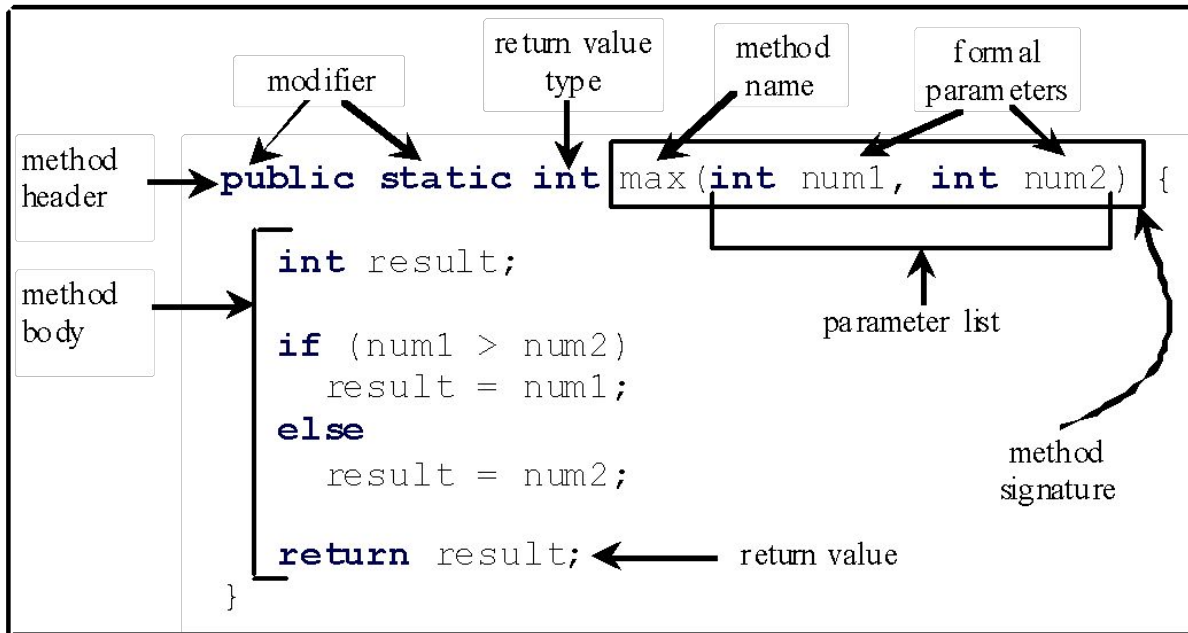

9

Formal Parameters

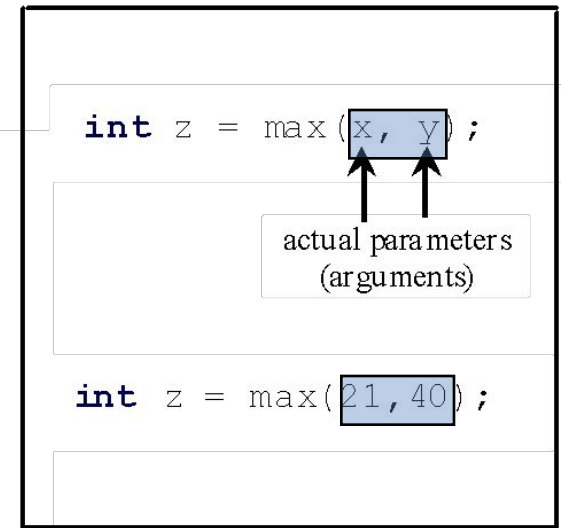The variables defined in the method header are known as formal parameters.

Actual Parameters

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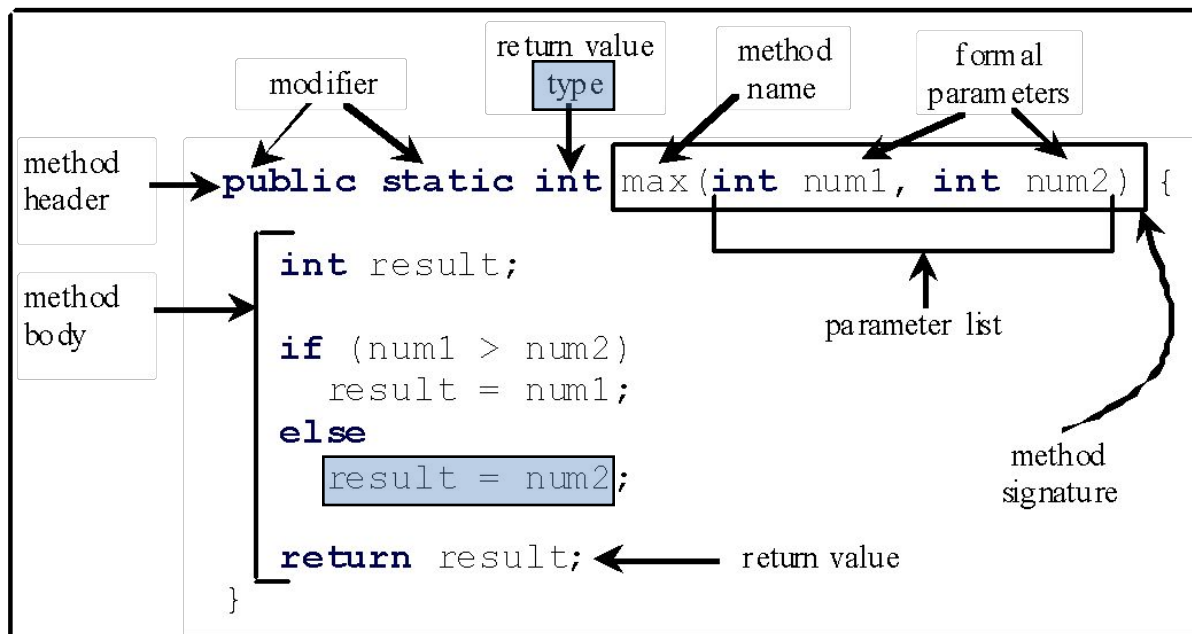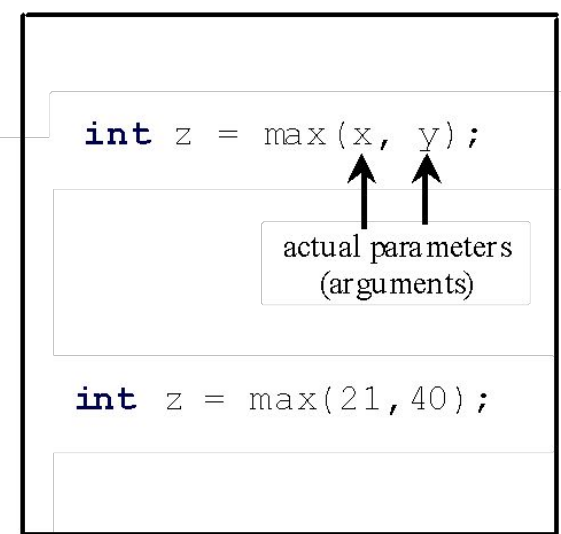
Return Value Type

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.

Calling Methods

Testing method **max**

This program demonstrates calling method **max** to return the largest of two **int** values.

Calling Methods, cont.

pass the value of i

pass the value of 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;
}
```

14
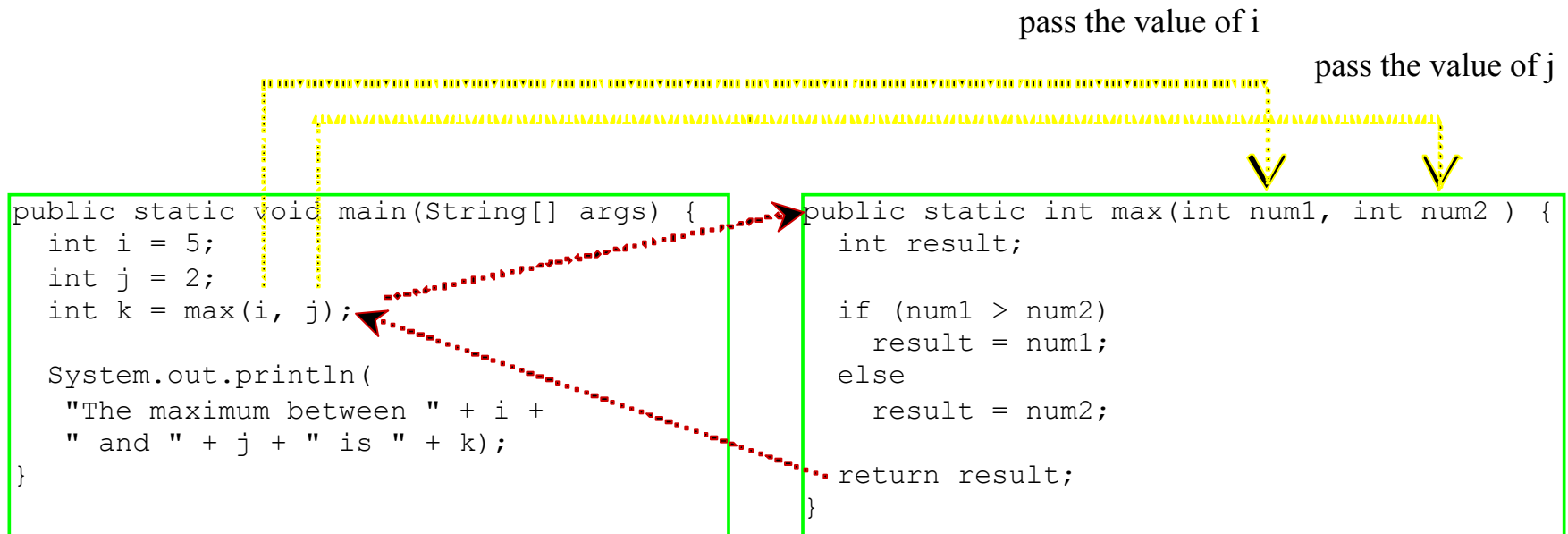
Trace Method Invocation

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 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)

```java
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);
}
```

```java
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;
}
```

20

Trace Method Invocation

result is now 5

```java
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);
}
```

```java
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

```java
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);
}
```

```java
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;
}
```

Program TestMax

```java
//class TestMax
public class TestMax
{
  public static void main(String[] args) // main method
  {
    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) // method max
  {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
  }
}
```

CAUTION

A <u>return</u> 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 <u>thinks</u> it is possible that this method does not return any value.

```java
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 →

```java
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 <u>return</u> statement to be reached regardless of how the <u>if</u> statement is evaluated.

Reuse Methods from Other Classes

One of the benefits of methods is for reuse. The <u>max</u> method (being public method) can be invoked from any other class besides <u>TestMax</u>.

If you create a new class <u>Test</u>, you can invoke method <u>max</u> using <u>ClassName.methodName</u> (e.g., <u>TestMax.max</u>).

You need to compile both classes to be able call method <u>max</u> from class <u>Test.</u>

Remember? Math.pow(a,b);   Math.sqrt(x);

Another Example

```java
// illustration of methods in java
import java.util.*;
public class TestMethods {
    public static void main (String[] arge)
    { int a = 10, b = 20;
        int addResult = Add(a,b); //method call Add
        System.out.println("Sum of a and b is " + addResult);

        String myMessage = "Hello World!"; // call method PrintMessage
        printMessage(myMessage);
    }

    // method definition
    public static int Add(int x, int y)
    { return (x+y); }

    // method definition
    public static void printMessage(String message)
    {
        for (int i = 1; i <= 5; i++)
            System.out.println(message);
    }
}
```

Runtime Stack

A runtime stack is a structure used to keep track of <u>active (currently running) methods</u> in the program, and <u>order of method calls</u>.

Each active method has "activation record" on the stack. The record is the <u>memory space for all local variables in the method.</u>

The top activation record on the stack represents the currently running (active) method in the program.

The bottom activation record represents the main method often program.

Once a method is no longer active, it is removed from the stack (always the top record is removed).
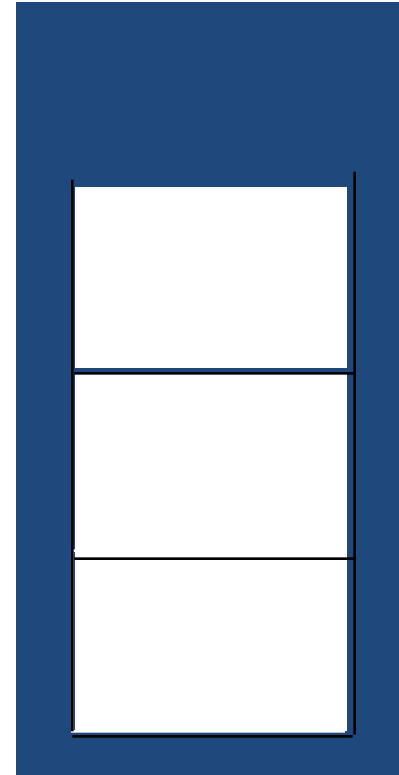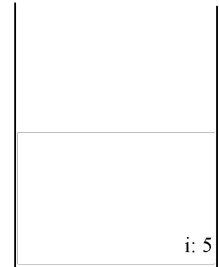
Trace Call Stack

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.

30

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;
}
```
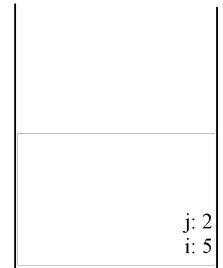
Space required for the main method

k:
j: 2
i: 5

The main method is invoked.

32

Trace Call Stack

Invoke max(i, j)

```java
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);
}
```

```java
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.

33

Trace Call Stack

pass the values of i and j to num1 and 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;
}
```

num2: 2
num1: 5

Space required for the main method

k:
j: 2
i: 5

The max method is invoked.

34

Trace Call Stack

pass the values of i and j to num1 and num2
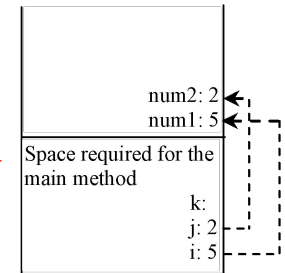
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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;
}
```

result:
num2: 2
num1: 5

Space required for the main method

k:
j: 2
i: 5

The max method is invoked.

35

Trace Call Stack

(num1 > num2) is true

```
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;
}
```
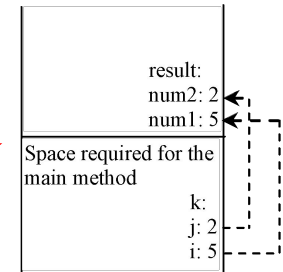
result:
num2: 2
num1: 5

Space required for the main method

k:
j: 2
i: 5

The max method is invoked.

36

Trace Call Stack

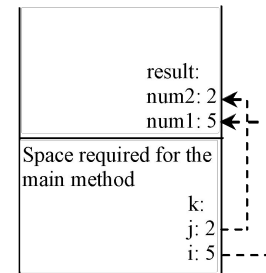Assign num1 to result

```
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;
}
```

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.

37

Trace Call Stack

Return result and assign it 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;
}
```
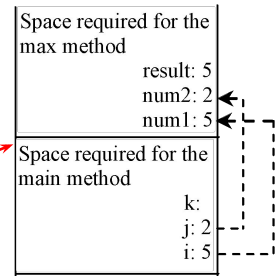
Space required for the max method	
	result: 5
	num2: 2
	num1: 5
Space required for the main method	
	k:5
	j: 2
	i: 5

The max method is invoked.

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;
}
```
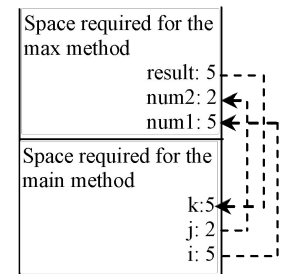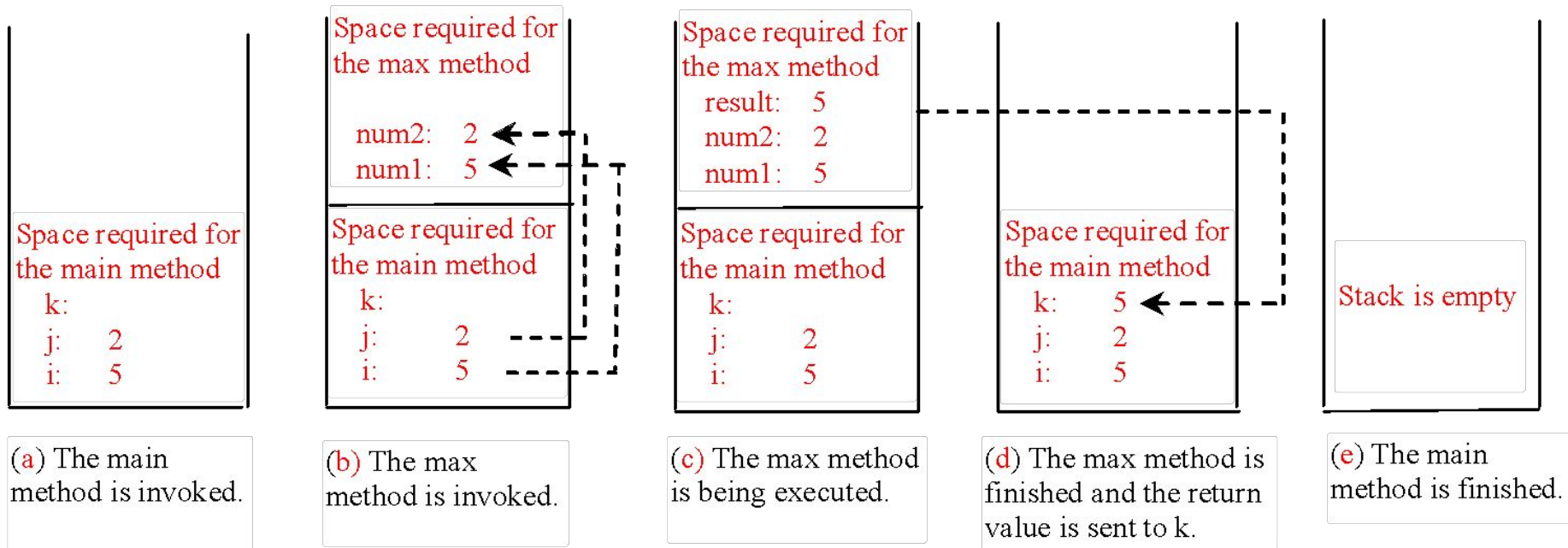
Space required for the main method

k:5
j: 2
i: 5

The main method is invoked.

Call Stacks

Space required for the max method

num2: 2
num1: 5

Space required for the main method
k:
j: 2
i: 5

(a) The main method is invoked.

Space required for the max method

num2: 2
num1: 5

Space required for the main method
k:
j: 2
i: 5

(b) The max method is invoked.

Space required for the max method
result: 5
num2: 2
num1: 5

Space required for the main method
k:
j: 2
i: 5

(c) The max method is being executed.

Space required for the main method
k: 5
j: 2
i: 5

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

Stack is empty

(e) The main method is finished.

40

void Method

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

```
public static void Even_Odd(int n)
{
   if ((n % 2) == 0)
     System.out.println(n + " is Even.");
   else
     System.out.println(n + " is Odd.");
}
```

See Listing 6.2, page 209, for example method:

public static void printGrade(double score)

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?

Pass by Value

It means that the value of the actual parameter (when a variable) is <u>copied</u> into the formal parameter.
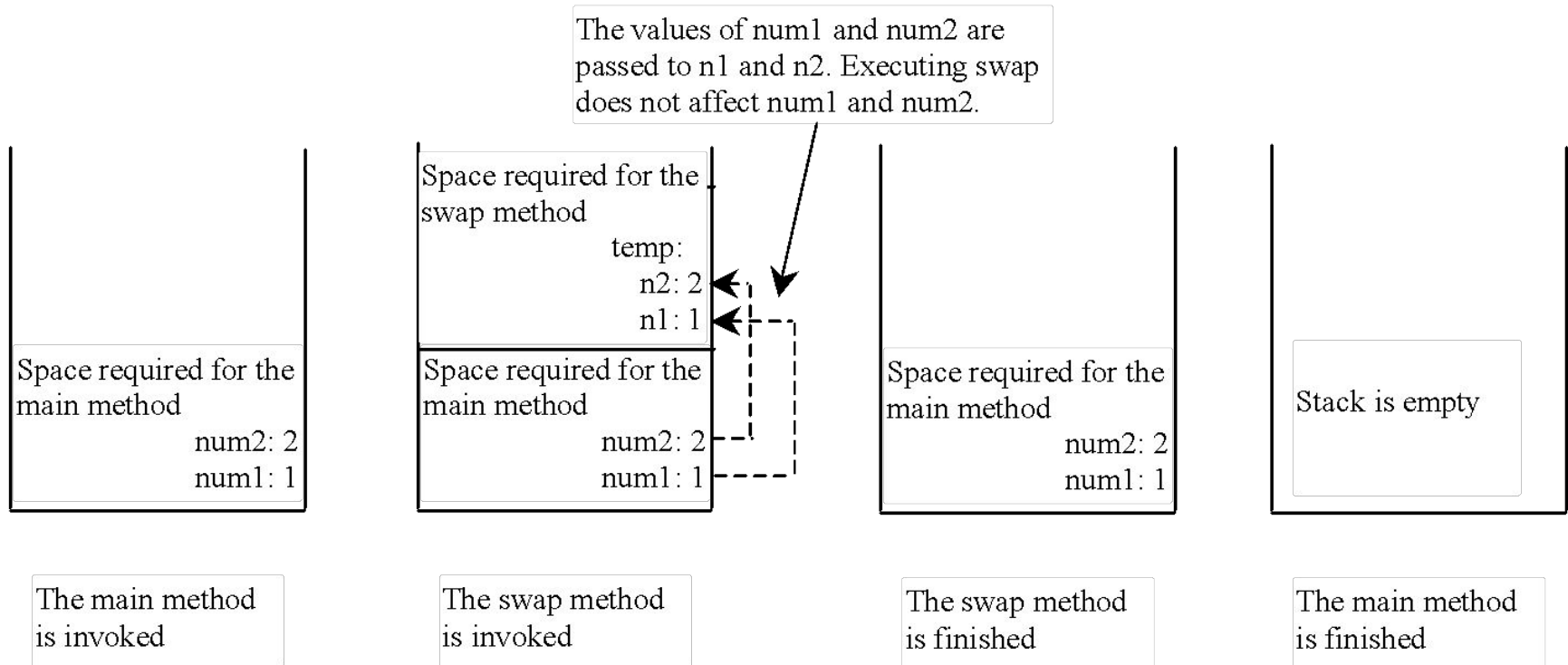
Whatever changes made to the formal parameter are local to the method and do not affect/change the value of the actual parameter.

Classic example: The Swap method.

Pass by Value Example

```java
public class TestPassByValue {
  public static void main (String[] arge){
    int num1 = 1;
    int num2 = 2;
    System.out.println("Before calling Swap: num1 = " + num1 +
        "  num2 = " + num1 + "\n");
    swap(num1, num2);
    System.out.println("After calling Swap:  num1 = " + num1 +
        "  num2 = " + num2 + "\n");
  }

  public static void swap(int n1, int n2){ // method swap
    System.out.println("Inside swap, before Swapping : n1 = " +
        n1 + "  n2 = " + n1 + "\n");
    int temp = n1;
    n1 = n2;
    n2 = temp;
    System.out.println("Inside swap, after Swapping:   n1 = " +
        n1 + "  n2 = " + n2 + "\n");
  }
}
```

Pass by Value Runtime Stack

The values of num1 and num2 are passed to n1 and n2. Executing swap does not affect num1 and num2.

Space required for the swap method

       temp:
      n2: 2
      n1: 1

Space required for the main method

     num2: 2
     num1: 1

Space required for the main method

     num2: 2
     num1: 1

Space required for the main method

     num2: 2
     num1: 1

Stack is empty

The main method is invoked

The swap method is invoked

The swap method is finished

The main method is finished

Modularizing Code

Modularization is software design concept that calls for writing code in modules.

Methods (as modules) 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.

Starting page 215, see listings 6.6 (GCD), 6.7 (Prime numbers), and 6.8 (converting decimal to hexadecimal). Each has at least one methods in addition to method `main()`.

Overloading Methods

Overloading is making a method to work with different types of parameters.

Example: Overloading the `max` Method

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


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

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 compilation error.

Ambiguous Invocation

```java
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1,2)); //Error
  }

  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
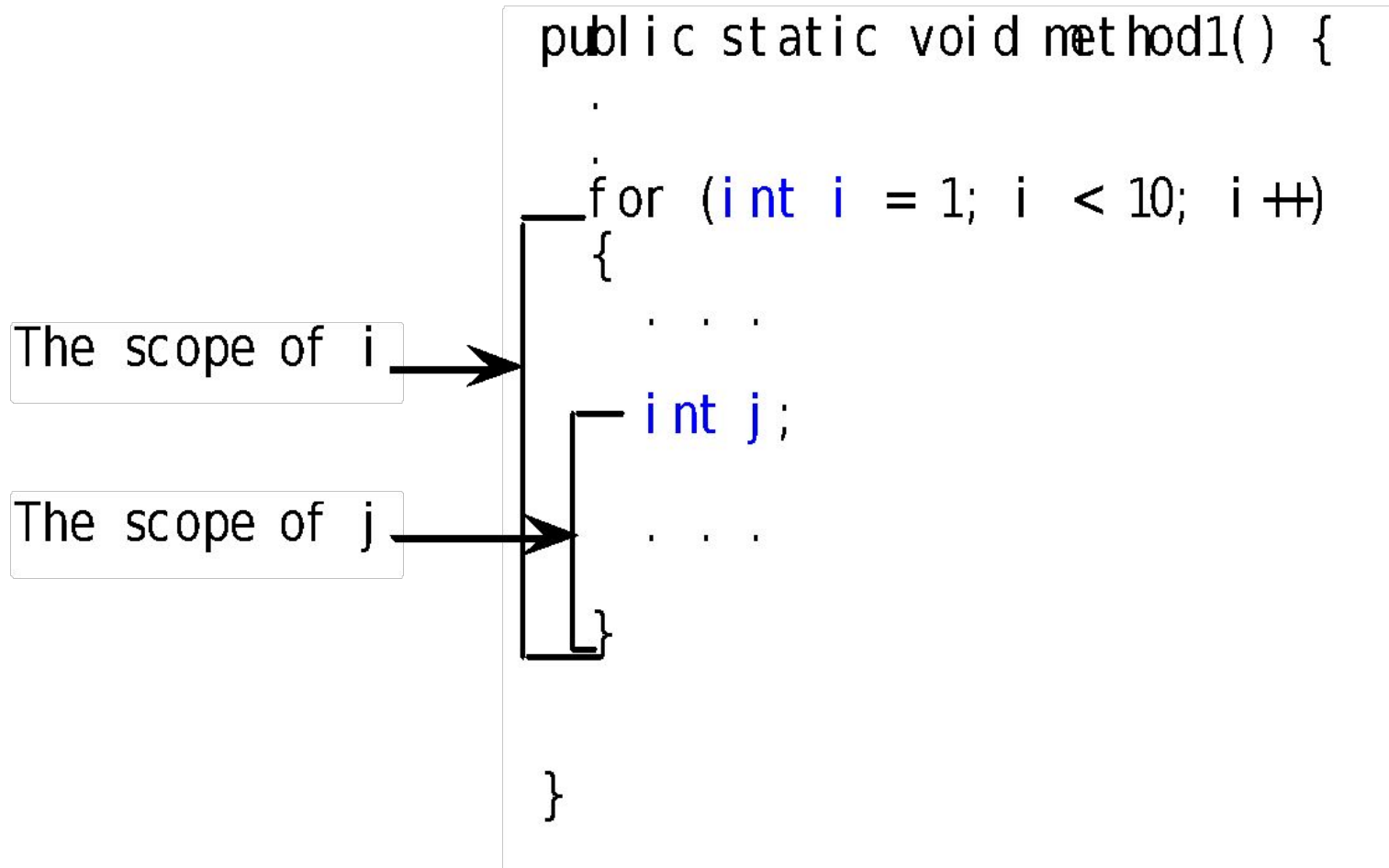
Local variable: a variable defined inside a method.

Scope: the part of the program where the variable can be referenced (accessible).

The scope of a local variable (also known as life-time) starts from its declaration point and continues to the end of the **block** that contains the variable. A local variable must be declared before it can be used.

Java Rule:

You can declare a local variable with the same name multiple time in different ***non-nesting blocks*** in a method, ***but you cannot declare a local variable twice in nested blocks.***
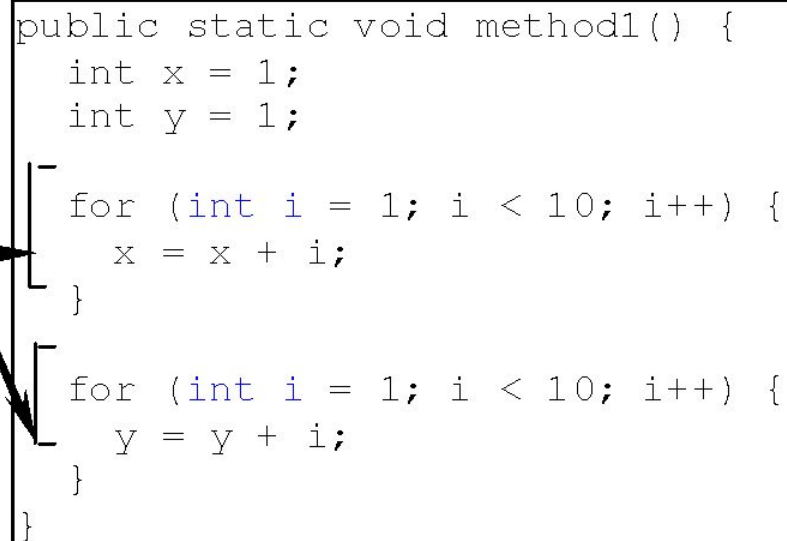
Scope of Local Variables, cont.

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

    . . .
  }

}
```

The scope of i →

The scope of j →
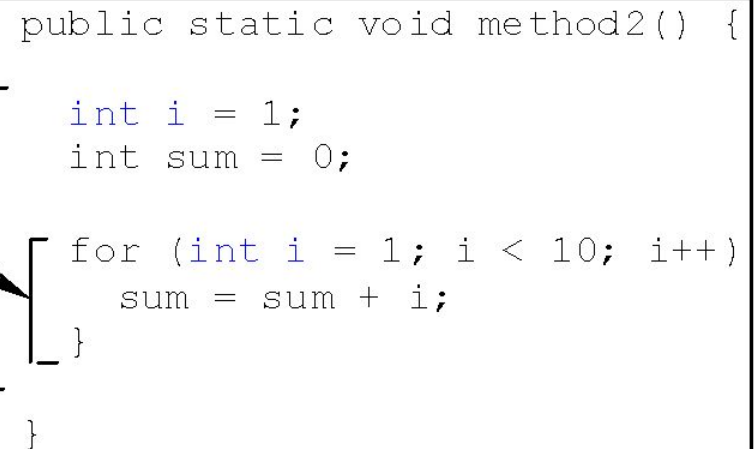
Scope of Local Variables, cont.

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

```java
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x = x + i;
  }

  for (int i = 1; i < 10; i++) {
    y = y + i;
  }
}
```

It is wrong to declare i in
two nesting blocks

```java
public static void method2() {

  int i = 1;
  int sum = 0;

  for (int i = 1; i < 10; i++)
    sum = sum + i;
  }

}
```

Scope of Local Variables, cont.

```java
// Homework: code with errors, can you find them?
public static void incorrectMethod()
{
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++)
  {
     int x = 0;
     int t = 0;
     x = x + i;
  }
  i = i + 10;
  y = y + 10;
  t = t + 10;
}
```