Chapter 6 Methods

Objectives

To define methods, invoke methods, and pass arguments to a method

To develop reusable code that is modular, easy-to-read, easy-to-debug, and easy-to-maintain.

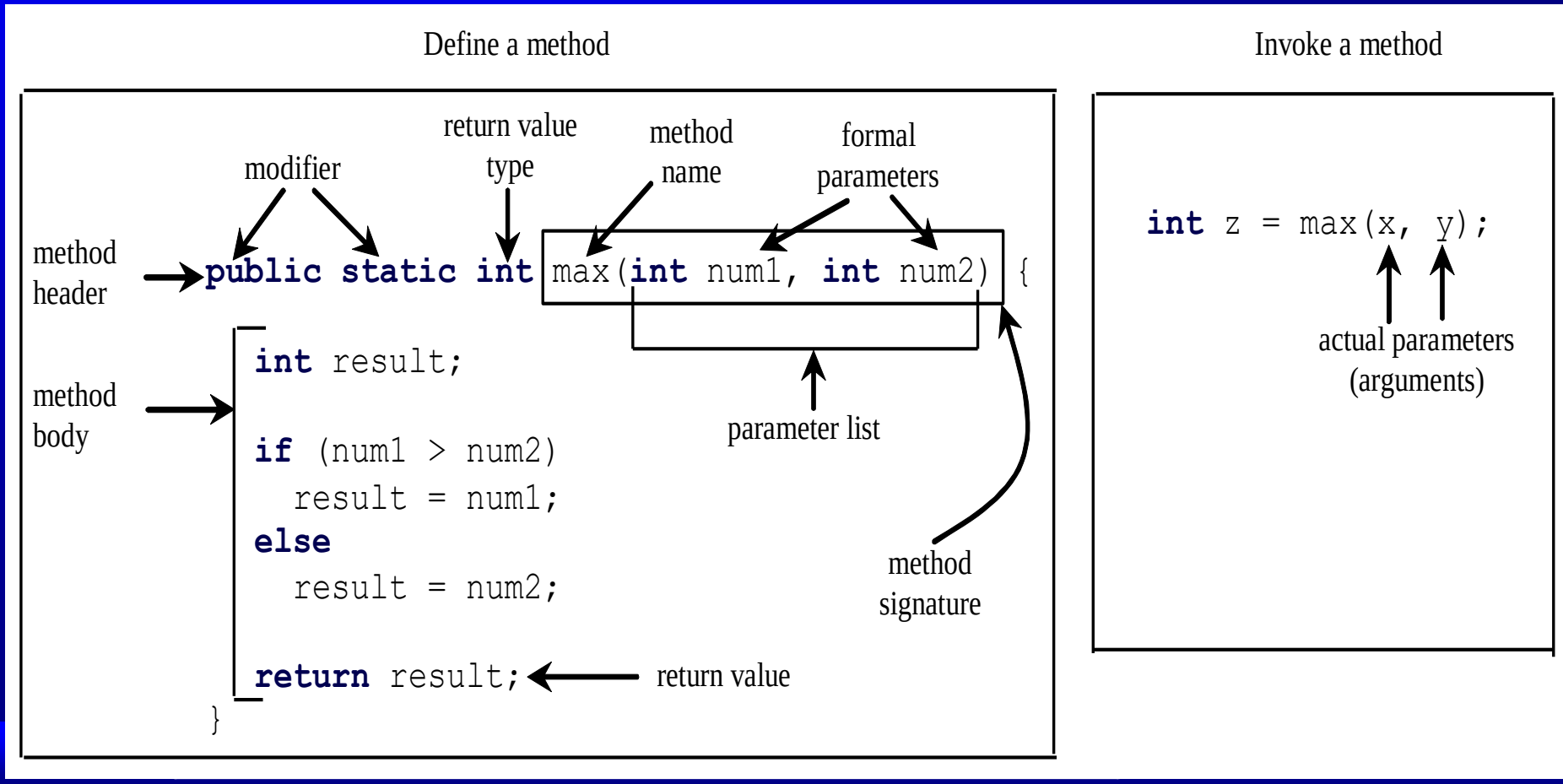
To use method overloading and understand ambiguous overloading

To design and implement overloaded methods

To determine the scope of variables

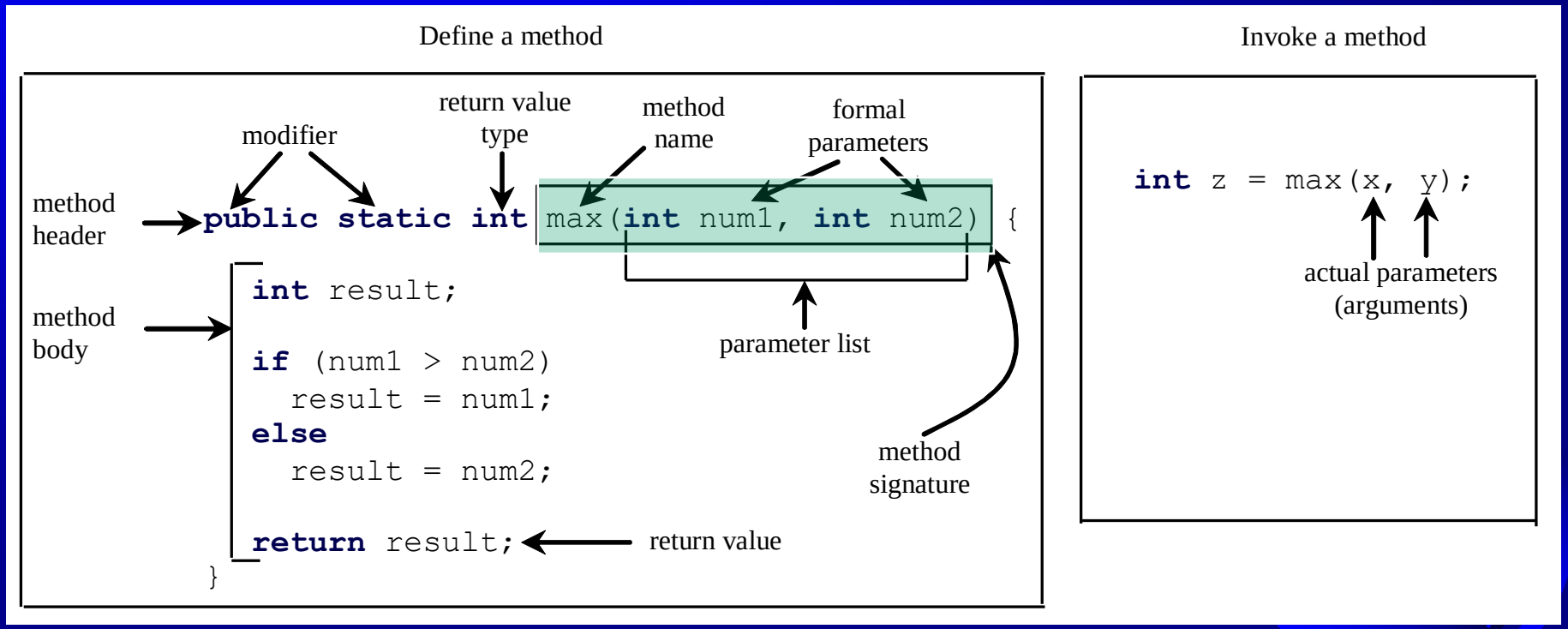
Defining Methods

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



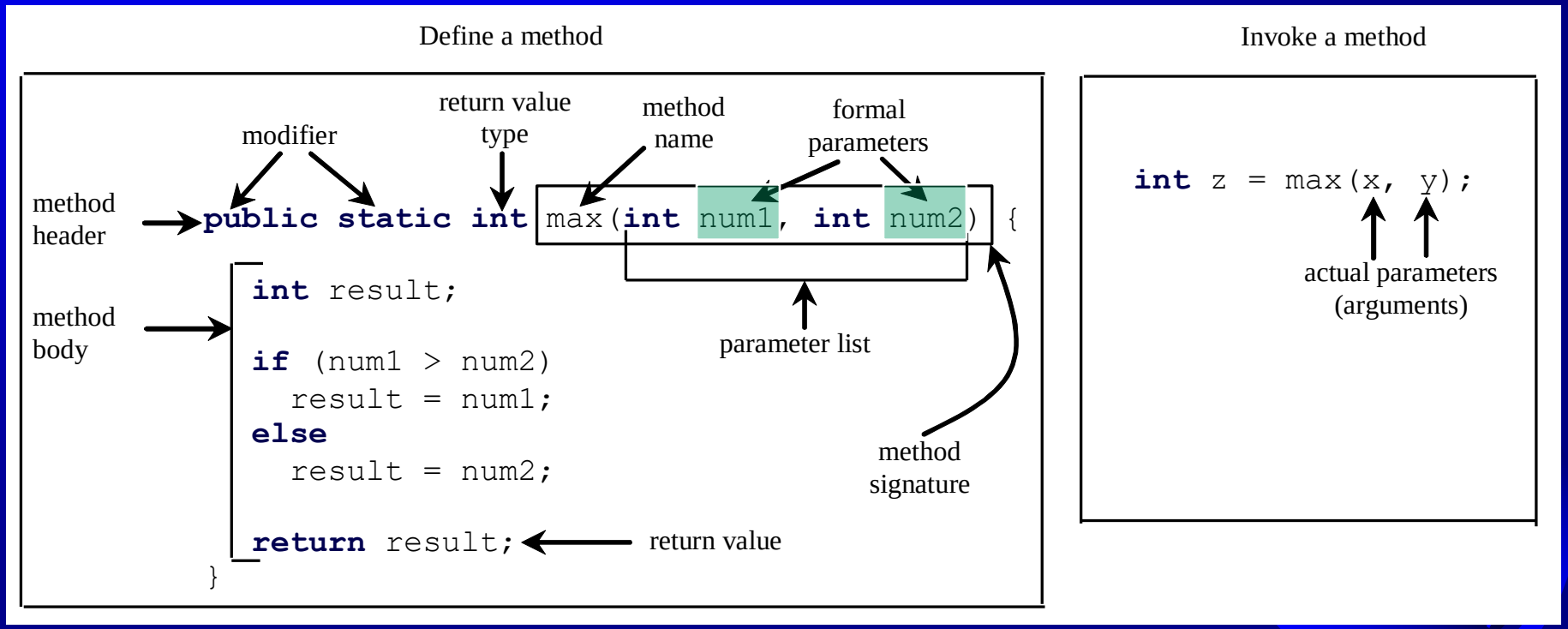
Method Signature

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



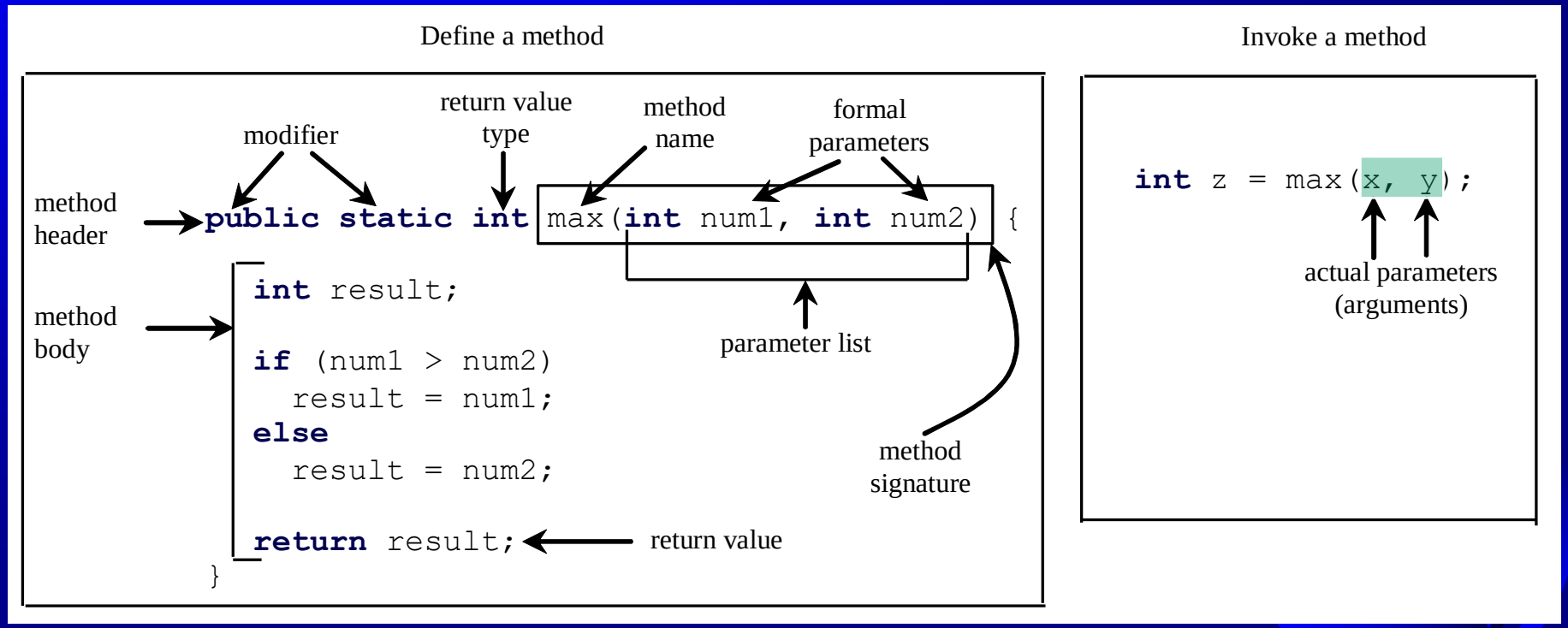
Formal Parameters

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



Actual Parameters

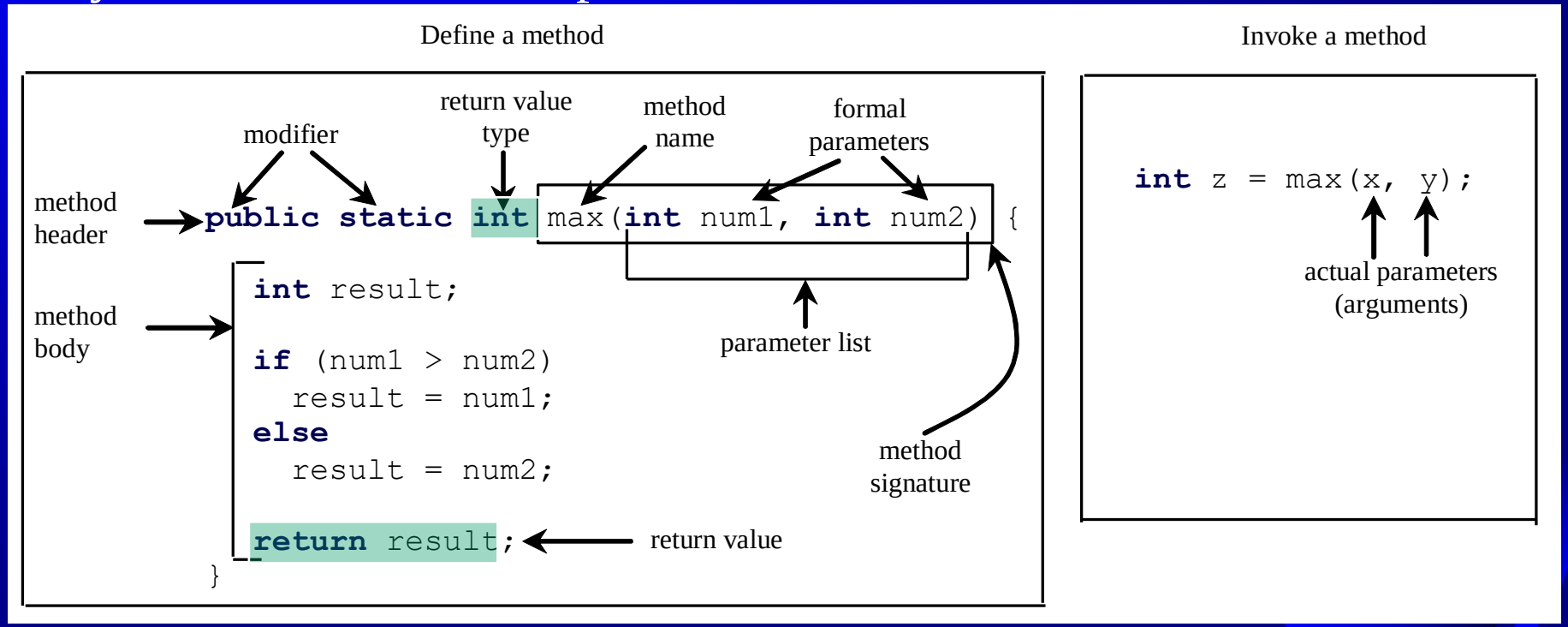
When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter* or *argument*.



Return Value Type

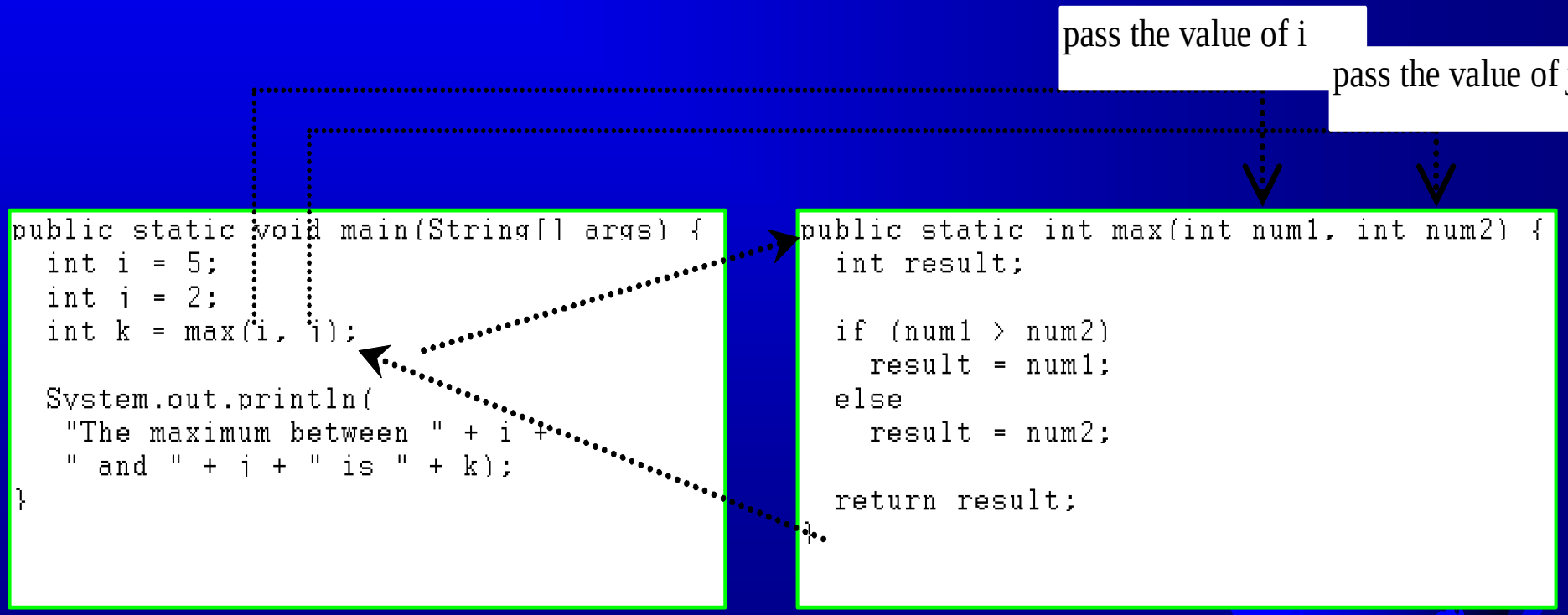
A method may return a value. The returnValueType is the data type of the value the method returns.

If the method does not return a value, the returnValueType is the keyword void. For example, the main method .



Calling Methods

Example. Call a method *max* to return the largest of the `int` values



animation

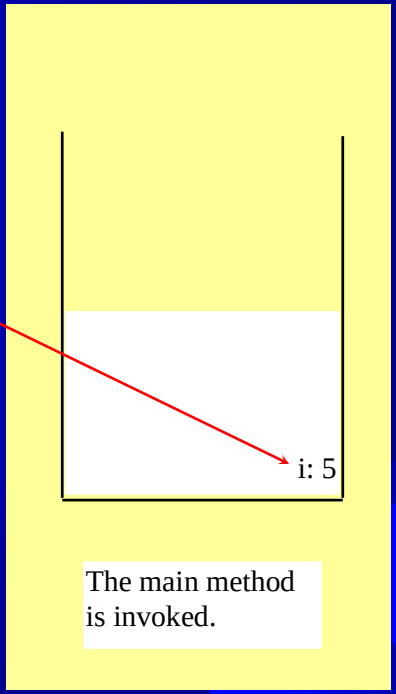
Trace Call Stack

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.

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;
}
```



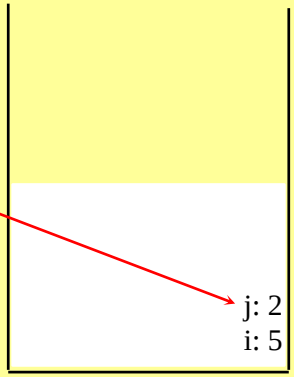
animation

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;
}
```



The main method is invoked.

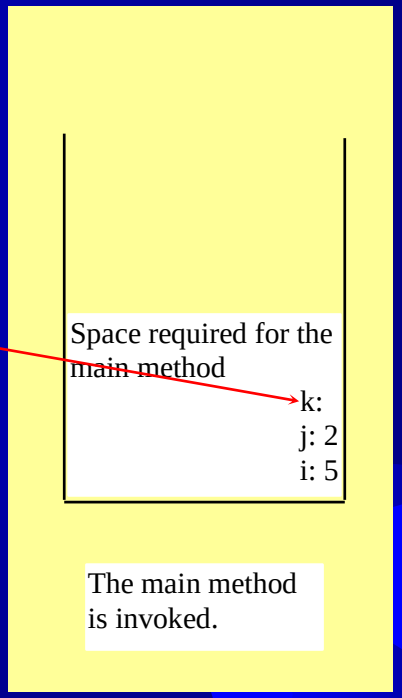
animation

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;
}
```



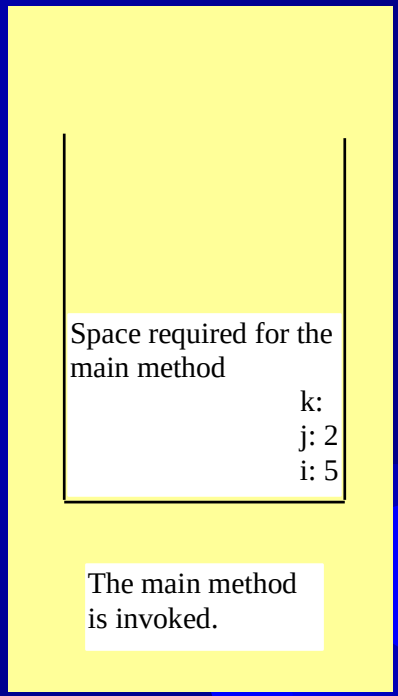
animation

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;
}
```



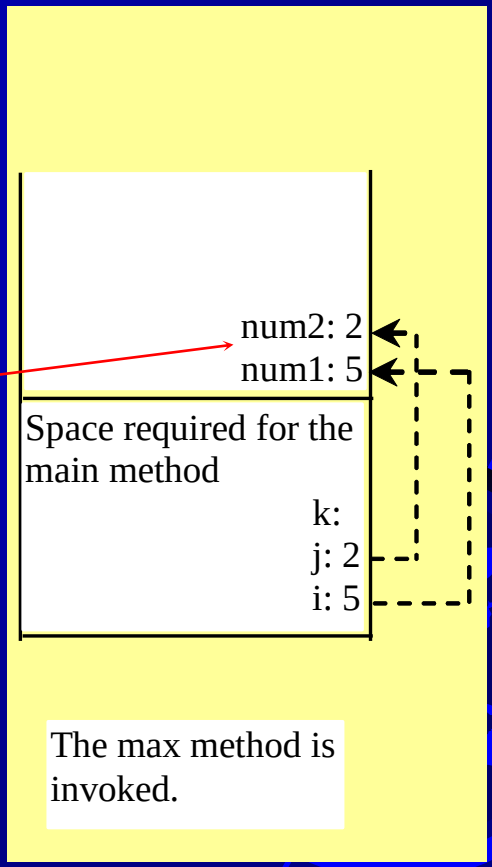
animation

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;
}
```



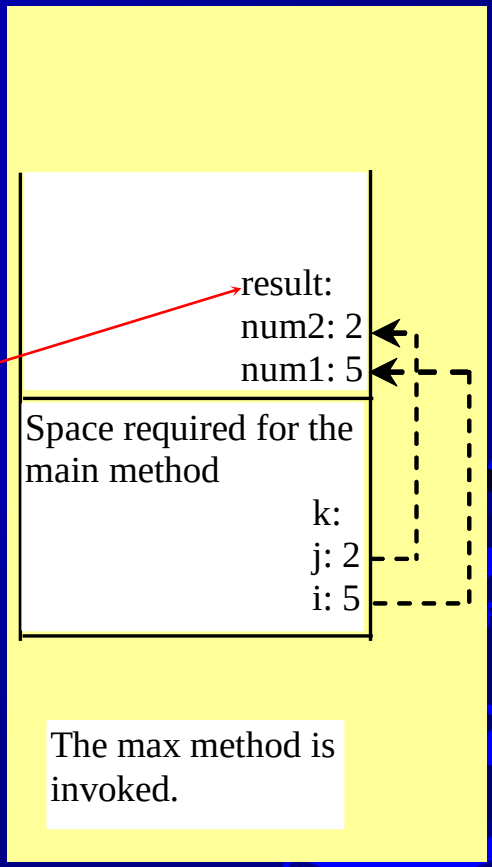
animation

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;
}
```



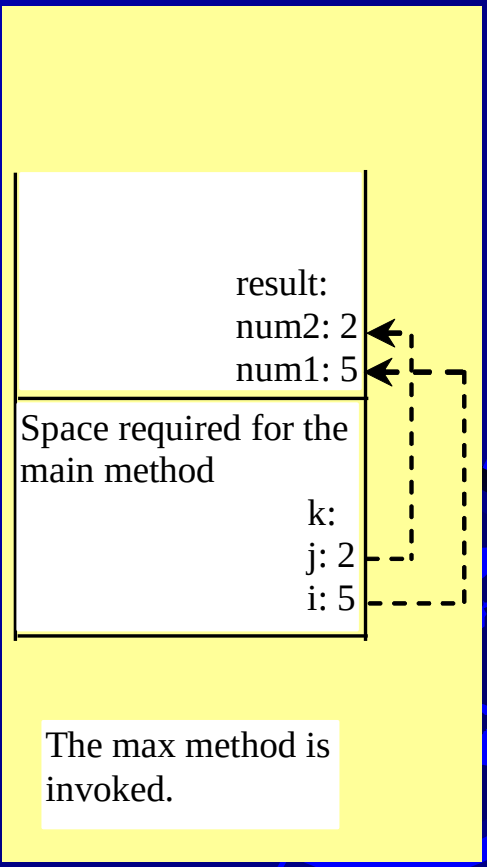
animation

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;
}
```



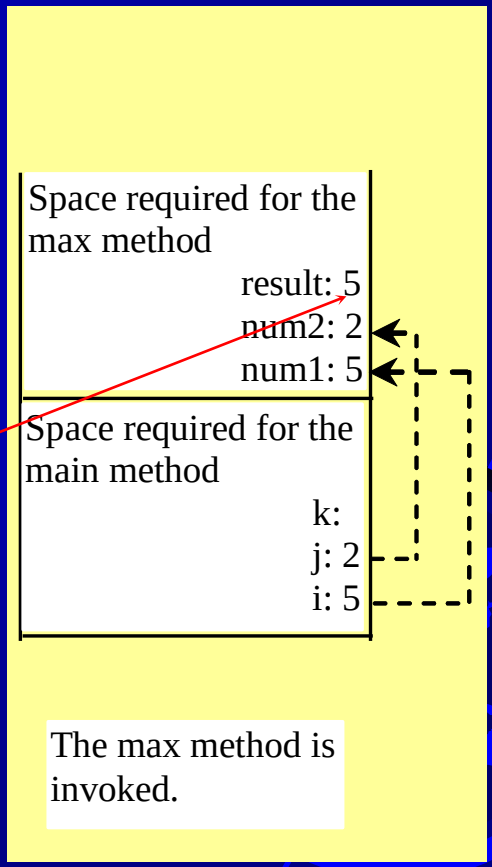
animation

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



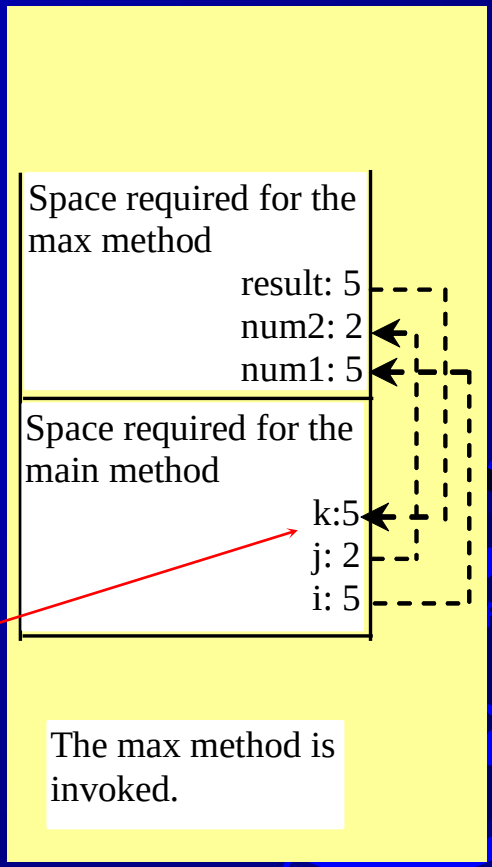
animation

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;
}
```



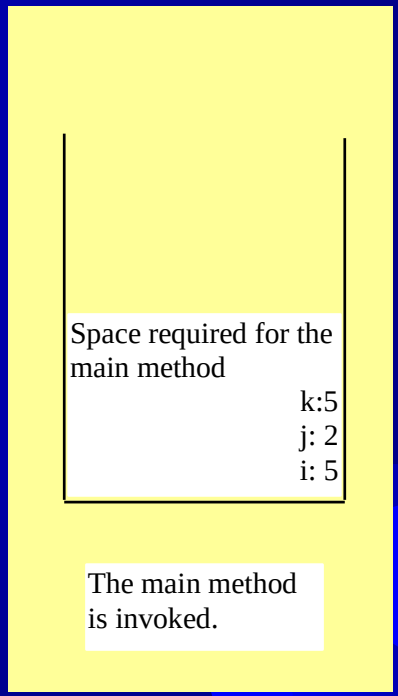
animation

Trace Call Stack

Execute print statement

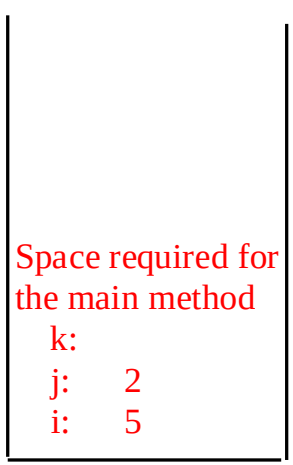
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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;
}
```

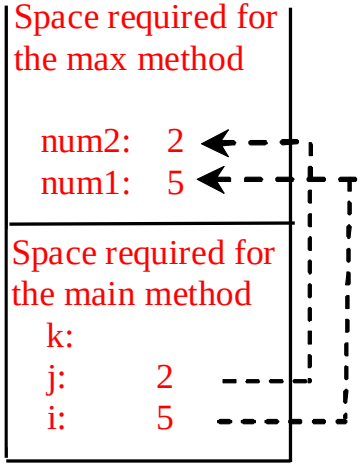



```
The maximum between 5 and 2 is 5
```

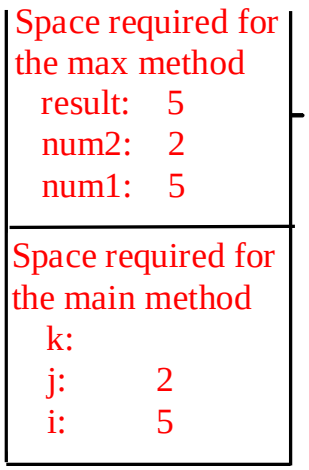
Call Stacks



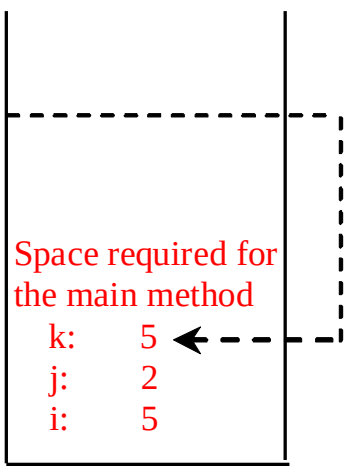
(a) The main method is invoked.



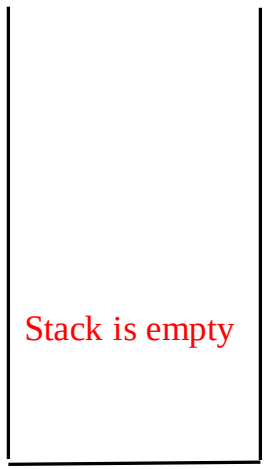
(b) The max method is invoked.



(c) The max method is being executed.



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



(e) The main method is finished.

void Method

This type of method does not return a value.

TestVoidMethod.java

```
1  public class TestVoidMethod {
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
10   public static void printGrade(double score) {
11     if (score >= 90.0) {
12       System.out.println('A');
13     }
14     else if (score >= 80.0) {
15       System.out.println('B');
16     }
17     else if (score >= 70.0) {
18       System.out.println('C');
19     }
20     else if (score >= 60.0) {
21       System.out.println('D');
22     }
23     else {
24       System.out.println('F');
25     }
26   }
27 }
```

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

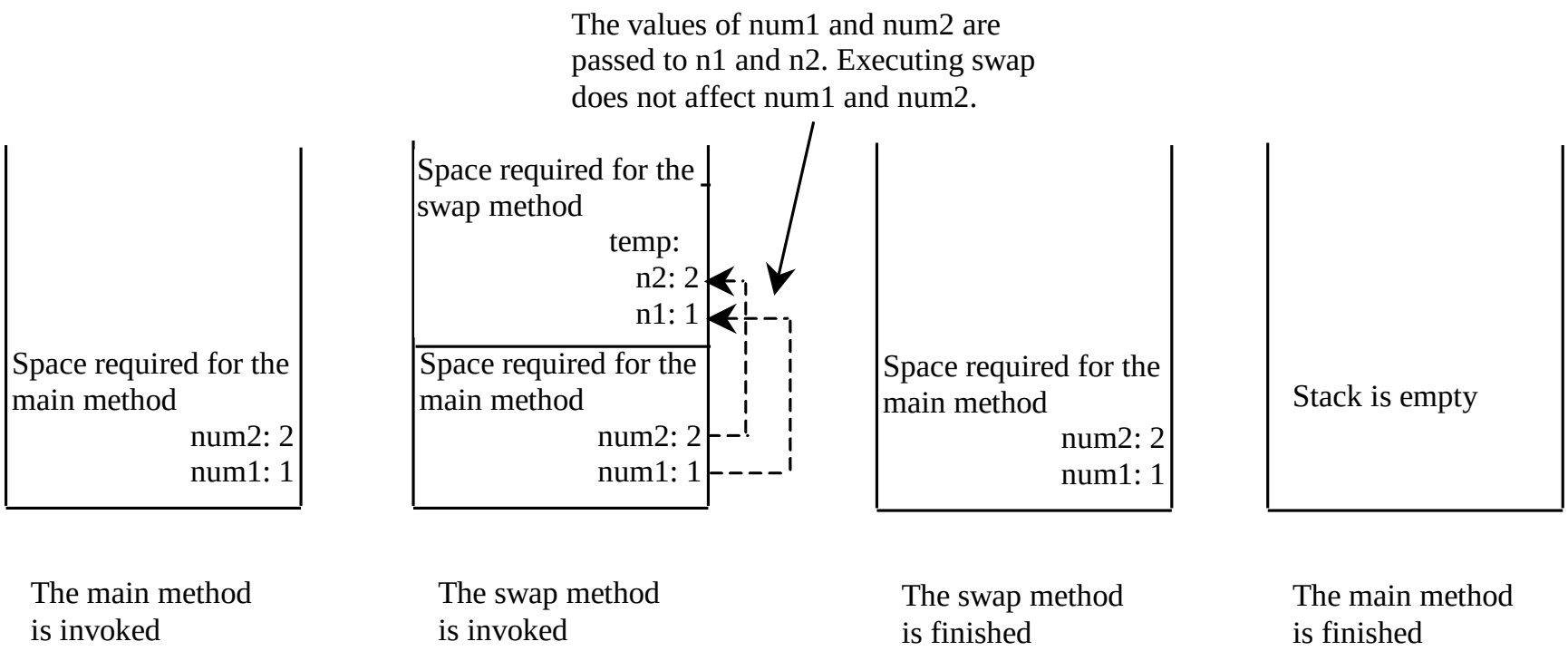
Passing Parameters by values

TestPassByValue.java

```
1  public                  Value {
2    /** I
3    public static void main(String[] args) {
4      // Declare and initialize variables
5      int num1 = 1;
6      int num2 = 2;
7
8      System.out.println("Before invoking the swap method, num1 is " +
9        num1 + " and num2 is " + num2);
10
11     // Invoke the swap method to attempt to swap two variables
12     swap(num1, num2);
13
14     System.out.println("After invoking the swap method, num1 is " +
15       num1 + " and num2 is " + num2);
16   }
17
18   /** Swap two variables */
19   public static void swap(int n1, int n2) {
20     System.out.println("\tInside the swap method");
21     System.out.println("\t\tBefore swapping n1 is " + n1
22       + " n2 is " + n2);
23
24     // Swap n1 with n2
25     int temp = n1;
26     n1 = n2;
27     n2 = temp;
28
29     System.out.println("\t\tAfter swapping n1 is " + n1
30       + " n2 is " + n2);
31   }
32 }
```

```
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
```

Pass by Value, cont.



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

invoke a method

from the same class: *methodName(…)*

from another class: *ClassName.methodName(…)*

Caution

The arguments must match the parameters in order, number, and compatible type, as defined in he method signature.

- Compatible type: means that you can pass without explicit casting

Eg. passing an int value argument to a double parameter.

Modularizing Code

Methods can be used to

reduce redundant coding and enable code reuse.

modularize code and improve the quality of the program.

GreatestCommonDivisor.java

```
1  import java.util.Scanner;
2
3  public class GreatestCommonDivisor {
4    /** Main method */
5    public static void main(String[] args) {
6      // Create a Scanner
7      Scanner input = new Scanner(System.in);
8
9      // Prompt the user to enter two integers
10     System.out.print("Enter first integer: ");
11     int n1 = input.nextInt();
12     System.out.print("Enter second integer: ");
13     int n2 = input.nextInt();
14
15     int gcd = 1; // Initial gcd is 1
16     int k = 2; // Possible gcd
17     while (k <= n1 && k <= n2) {
18       if (n1 % k == 0 && n2 % k == 0)
19         gcd = k; // Update gcd
20       k++;
21     }
22
23     System.out.println("The greatest common divisor for " + n1 +
24       " and " + n2 + " is " + gcd);
25   }
26 }
```

Rewrite it using a method?

Using a method

GreatestCommonDivisorMethod.java

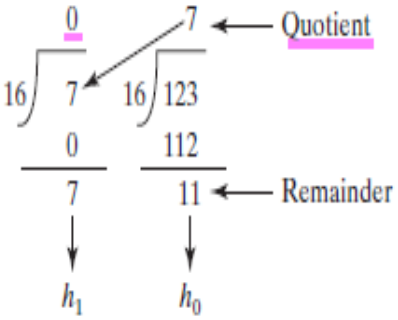
```
 1 import java.util.Scanner;
 2
 3 public class GreatestCommonDivisorMethod {
 4   /** Main method */
 5   public static void main(String[] args) {
 6     // Create a Scanner
 7     Scanner input = new Scanner(System.in);
 8
 9     // Prompt the user to enter two integers
10     System.out.print("Enter first integer: ");
11     int n1 = input.nextInt();
12     System.out.print("Enter second integer: ");
13     int n2 = input.nextInt();
14
15     System.out.println("The greatest common divisor for " + n1 +
16       " and " + n2 + " is " + gcd(n1, n2) );
17   }
18
19   /** Return the gcd of two integers */
20   public static int gcd(int n1, int n2) {
21     int gcd = 1; // Initial gcd is 1
22     int k = 2;  // Possible gcd
23
24     while (k <= n1 && k <= n2) {
25       if (n1 % k == 0 && n2 % k == 0)
26         gcd = k; // Update gcd
27       k++;
28     }
29
30     return gcd; // Return gcd
31   }
32 }
```

Problem: Write a method to Convert Decimals to Hexadecimals

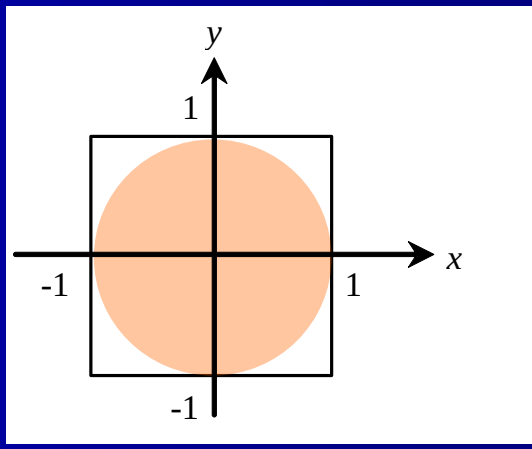
```
/** Convert a decimal to a hex as a string */
public static String decimalToHex(int decimal) {
  String hex = "";

  while (decimal != 0) {
    int hexValue = decimal % 16;
    hex = toHexChar(hexValue) + hex;
    decimal = decimal / 16;
  }

  return hex;
}

/** Convert an integer to a single hex digit
public static char toHexChar(int hexValue) {
  if (hexValue <= 9 && hexValue >= 0)
    return (char)(hexValue + '0');
  else  // hexValue <= 15 && hexValue >= 10
    return (char)(hexValue - 10 + 'A');
  }
}
```

These numbers can be found by successively dividing d by 16 until the quotient is 0. The remainders are $h_0, h_1, h_2, \ldots, h_{n-2}, h_{n-1}$, and h_n.

For example, the decimal number 123 is 7B in hexadecimal. The conversion is done as follows:



Problem: *Monte Carlo Simulation*



- A technique that uses random numbers and probability to solve problems. This method has a wide range of applications in computational mathematics, physics, chemistry, and finance.

- This example uses it for estimating π.

- Write a program that randomly generates **1000000** points in the square and let numberOfHits denote the number of points that fall in the circle.
  - circleArea / squareArea = π / 4.

    So, **numberOfHits** ≈ **1000000 * (π / 4)**.

- π ≈ 4 * numberOfHits / 1000000.

MonteCarloSimulation.java

```
1  public class MonteCarloSimulation {
2    public static void main(String[] args) {
3      final int NUMBER_OF_TRIALS = 10000000;
4      int numberOfHits = 0;
5
6      for (int i = 0; i < NUMBER_OF_TRIALS; i++) {
7        double x = Math.random() * 2.0 - 1;
8        double y = Math.random() * 2.0 - 1;
9        if (x * x + y * y <= 1)
10         numberOfHits++;
11     }
12
13     double pi = 4.0 * numberOfHits / NUMBER_OF_TRIALS;
```

generate random points

check inside circle

estimate pi

```
14     System.out.println("PI is " + pi);
15   }
16 }
```

```
PI is 3.14124
```

- $-1 \leq x < 1$
- $0 \leq x+1 < 2$
- $0 \leq (x+1)/2 < 1$
- $0 \leq$ random < 1
- random=(x+1)/2
- x= random*2 -1

Overloading Methods

two methods have the same name but different parameter lists within one class.

The Java compiler determines which method is used based on the **method signature**.

```
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;
}
```

TestMethodOverloading.java

```
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 between 3 and 4 is "
6        + max(3, 4));
7
8      // Invoke the max method with the double parameters
9      System.out.println("The maximum between 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 between 3.0, 5.4, and 10.14 is "
14       + max(3.0, 5.4, 10.14));
15   }
16
17   /** Return the max between 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 between 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 among three double values */
34   public static double max(double num1, double num2, double num3) {
35     return max(max(num1, num2), num3);
36   }
37 }
```

Ambiguous Invocation

Ambiguous invocation is a **compilation error !**

Ambiguous invocation: there are two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.

Ambiguous Invocation

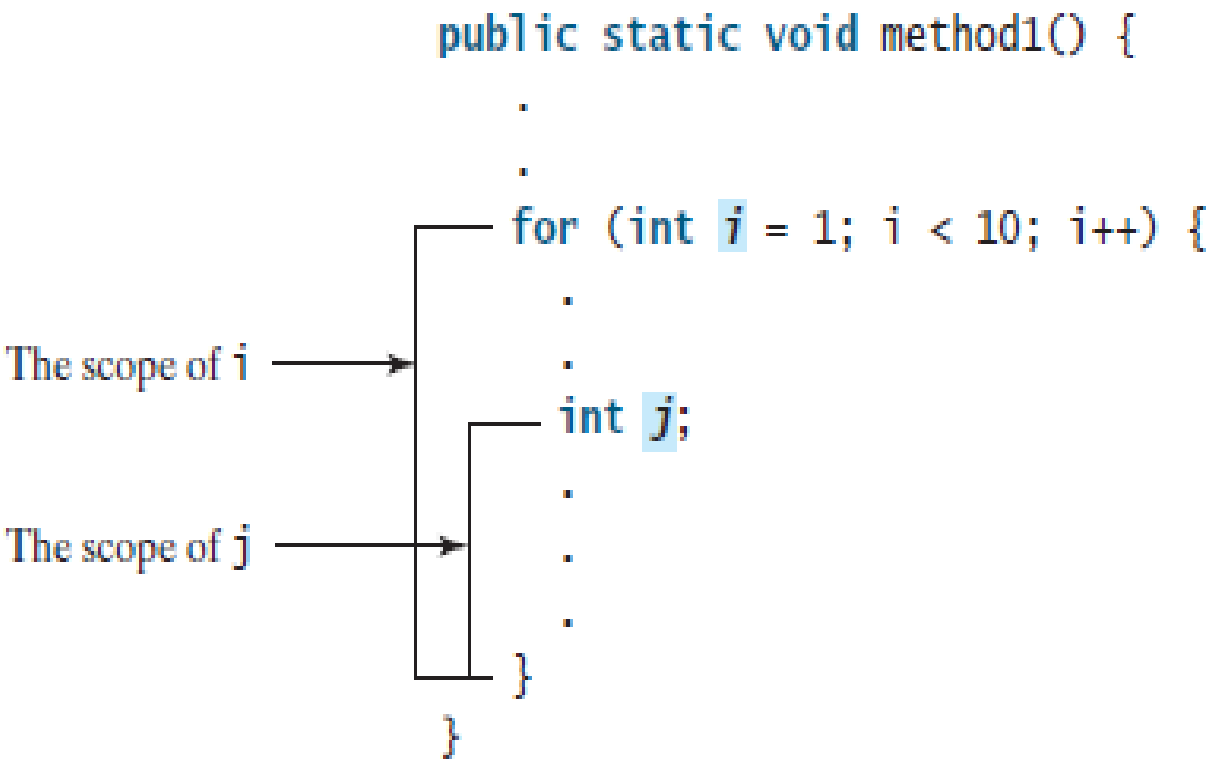
```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2) );
  }

  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: defined inside a method.

Must be declared before it can be used.

Scope: from its declaration to the end of the block that contains the variable.



A variable declared in the initial action part of a `for`-loop header has its scope in the entire loop.

Scope of Local Variables, cont.

It is fine to declare `i` in two nonnested 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 nested blocks

```
public static void method2() {

  int i = 1;
  int sum = 0;

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

}
```

A variable can be declared multiple times in nonnested blocks but only once in nested blocks.

Wrong?

```
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.

Case Study:
Generating **Random Characters**

Each character has a unique Unicode

- between 0 and FFFF in hexadecimal

  (65535 in decimal)

To generate a random character is to generate a random integer between 0 and 65535(inclusive):

- (int) (Math.random() * (65535 + 1))

generate a **random lowercase letter :**

(char) ('a' + Math.random() * ('z' - 'a' + 1))

- The Unicode for lowercase letters are consecutive
- The char operand is cast into a **number** if the other operand is a number or a character.

To generate a random character **between any two characters ch1** and **ch2** with ch1 < ch2:

(char)(ch1 + Math.random() * (ch2 – ch1 + 1))