Method

- We have already used the methods:
- math methods
- methods system.out.print

- In this Chapter, we learn how to define our own custom methods.

- Methods can be used to define reusable code and organize and simplify coding and make code easy to maintain.

Method

- A **method** is a named list of statements.
- A method definition consists of the new method's name and a block of statements, such as

public static void printPizzaArea() { /* block of statements */ }.

  - public: the method may be called from any class in the program.
  - static: the method is associated with the class.
  - Return type: what value will be returned to the place of method call. void means nothing is returned. Void method does not return a value, it performs some actions.

- A **method call** is an invocation of a method's name, causing the method's statements to execute.

- **Example**: Write a method that prints "Hello!".

- Why use methods?
  - Reduce redundancy
  - Improve readability

- **Example**:

(1)Write a method that prints "Hello!".

```
public static void printHello() {
        System.out.println("Hello!");
 }
```

```
public class NewClass2 {

   public static void printHello() {                    // define method
       System.out.println("Hello!");
    }

    public static void main(String[] args) {
     printHello();                                      // call method printHello()

  }
}
```

Output:

Hello!

(2) Write method sayHelloTo() to print "Hello, ***!”

```
public static void sayHelloTo(String name) {        // name is the method parameter
    System.out.println("Hello, " + name + "!");
}
```

```
public class NewClass{

    public static void sayHelloTo(String name) {        // name is the method parameter
        System.out.println("Hello, " + name + "!");
    }

    public static void main(String[] args) {
        sayHelloTo("John");                              // "John" is the method argument
  }
}
```

Out put:

Hello, John!

(3) SomeoneSayHelloTo() to say hello to someone from someone else

```
public static void someoneSayHelloTo(String nameFrom, String nameTo) {
        System.out.println(nameFrom + " says: Hello, " + nameTo + "!");
```

```
public class NewClass{

   public static void someoneSayHelloTo(String nameFrom, String nameTo) {
      System.out.println(nameFrom + " says: Hello, " + nameTo + "!");
   }

  public static void main(String[] args) {

      someoneSayHelloTo("Mike", "John");
      someoneSayHelloTo("John", "Mike");
 }
}
```

Output:

Mike says: Hello, John!

John says: Hello, Mike!

Parameters and Arguments

A programmer can influence a method's behavior via an input.

- A **parameter** is a method input specified in a method definition.

  Ex: A pizza area method might have diameter as an input.

- An **argument** is a value provided to a method's parameter during a method call.

  Ex: A pizza area method might be called as printPizzaArea(12.0) or as printPizzaArea(16.0).

Write a method getPizzaArea that calculates the area of a pizza with given diameter. Use this method to calculate the area of a 12-inch pizza and a 16-inch pizza.

- Formula:

  radius = diameter / 2

  area = pi * radius * radius

```
public static double getPizzaArea(double diameter) {
    double radius = diameter / 2.0;
    double circleArea = Math.PI * Math.pow(radius, 2);
    return circleArea;
```

```
public class NewClass{
    public static double getPizzaArea(double diameter) {
        double radius = diameter / 2.0;
        double circleArea = Math.PI * Math.pow(radius, 2);
        return circleArea;
    }

    public static void main(String[] args) {
        System.out.println("The area of 12\" of a pizza is: " + getPizzaArea(12.0) );
        System.out.println("The area of 16\" of a pizza is: " + getPizzaArea(16.0) );
  }
  }
```

Output:

The area of 12" of a pizza is: 113.09733552923255

The area of 16" of a pizza is: 201.06192982974676

```
public class NewClass{
    public static double getPizzaArea(double diameter) {
        double radius = diameter / 2.0;
        double circleArea = Math.PI * Math.pow(radius, 2);
        return circleArea;
    }
    public static void main(String[] args) {
        System.out.printf("12 inch pizza has %.2f calories.\n",getPizzaArea(12.0));
        System.out.printf("16 inch pizza has %.2f calories.\n",getPizzaArea(16.0));
    }
}
```

Output:

12 inch pizza has 113.10 calories.
16 inch pizza has 201.06 calories.

- A method definition may have multiple parameters, separated by commas. Parameters are assigned with argument values by position: First parameter with first argument, second with second, etc.

- A method definition with no parameters must still have the parentheses.

Returning A Value from A Method

- A method may return one value using a **return statement**.
- A method can only return one item.
- The return type needs to be declared at the beginning.
- Type **void** indicates that a method does not return any value.

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;
}
```

```
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));
}
```

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;
}
```

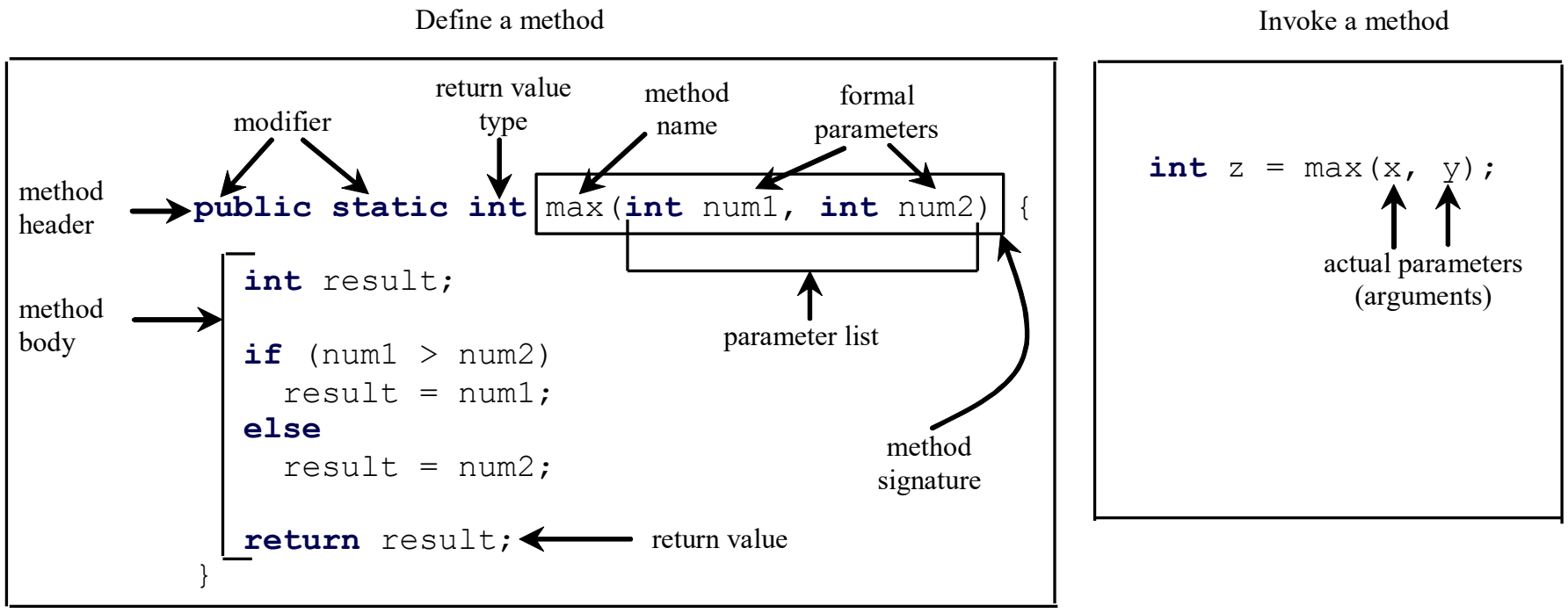
Invoke a method

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


actual parameters (arguments)

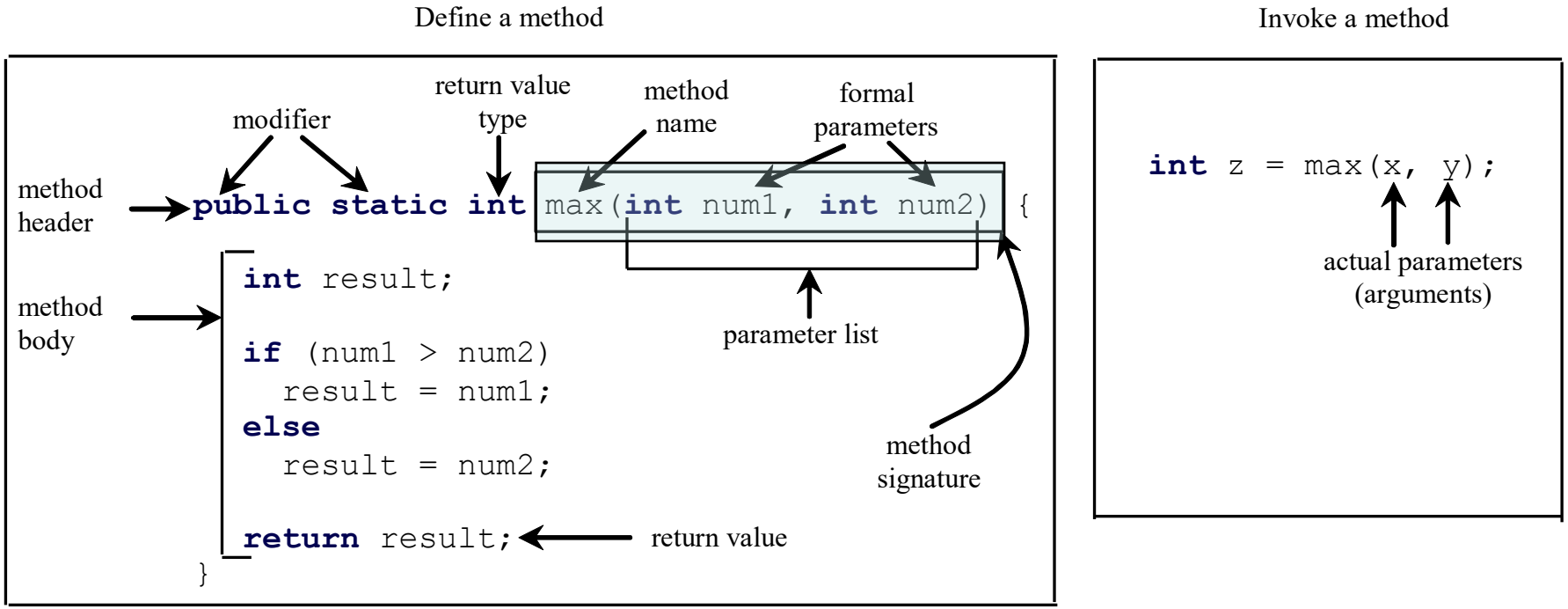
Defining Methods

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



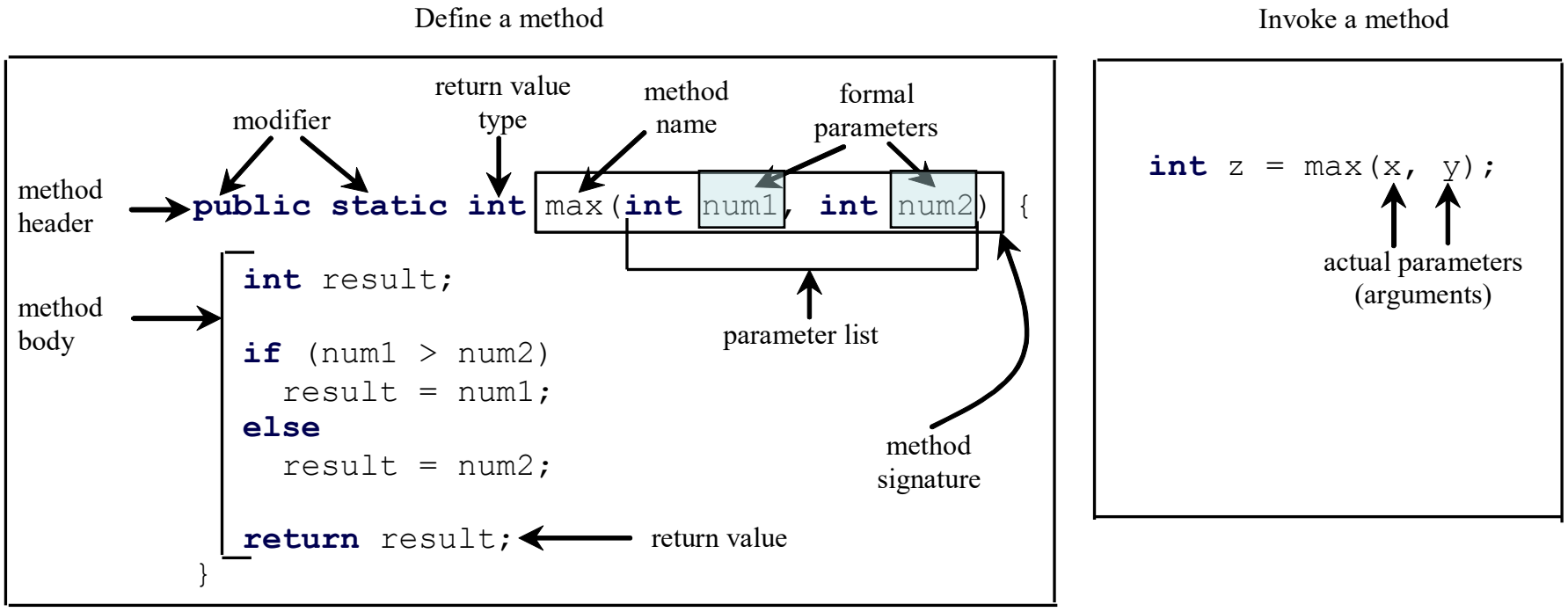
Method Signature

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



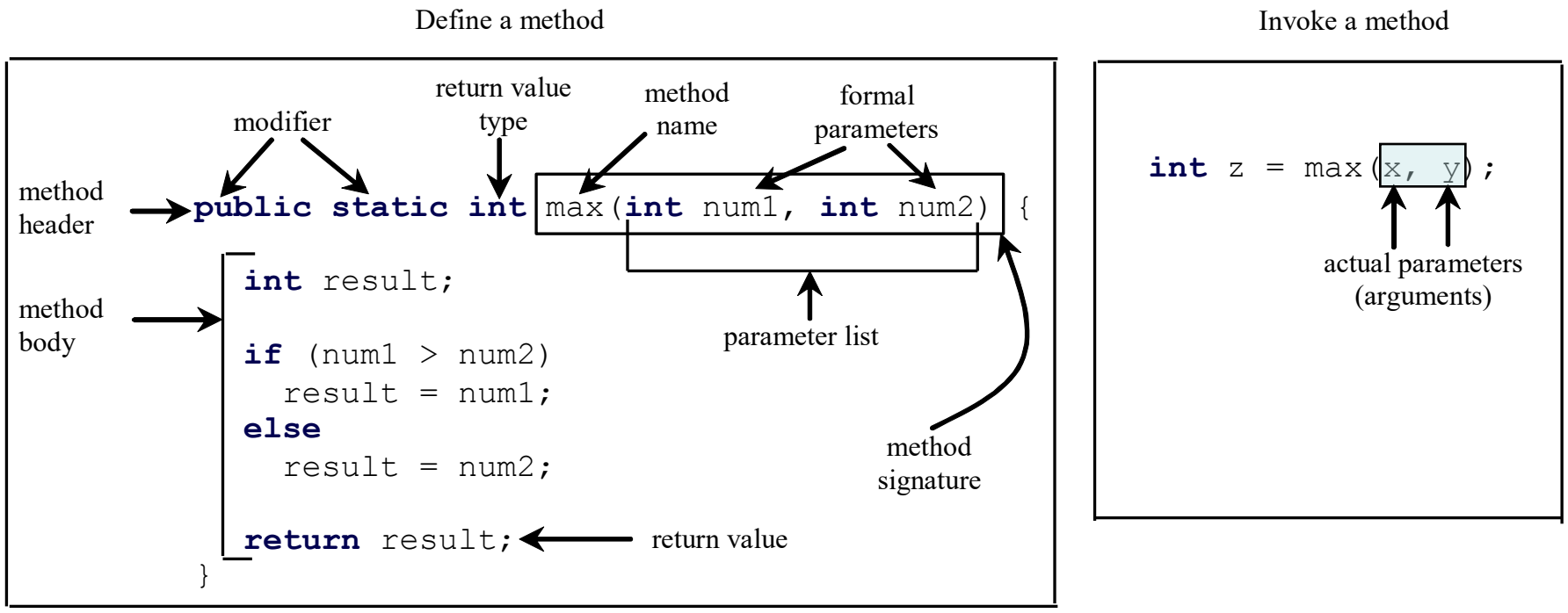
Formal Parameters

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



Actual Parameters

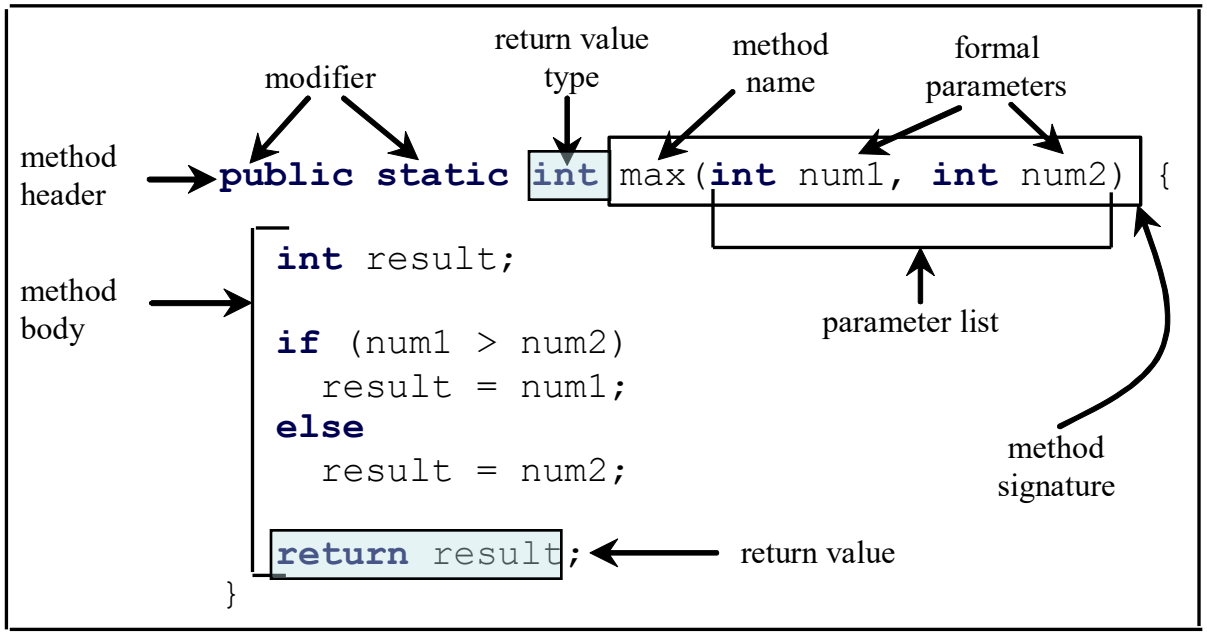
When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.



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 returnValueType in the main method is void.

Define a method



Invoke a method

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

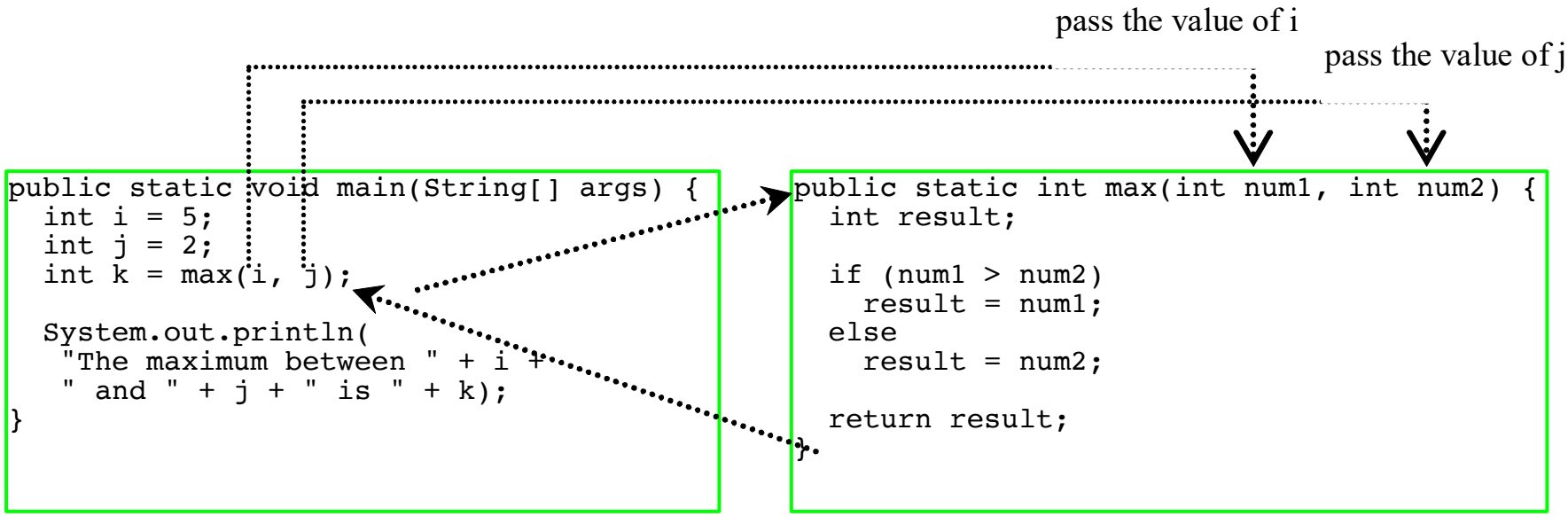
actual parameters (arguments)

Calling Methods

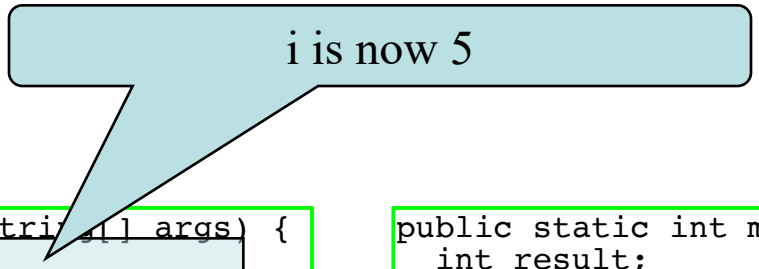
Testing the `max` method

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

Calling Methods, cont.



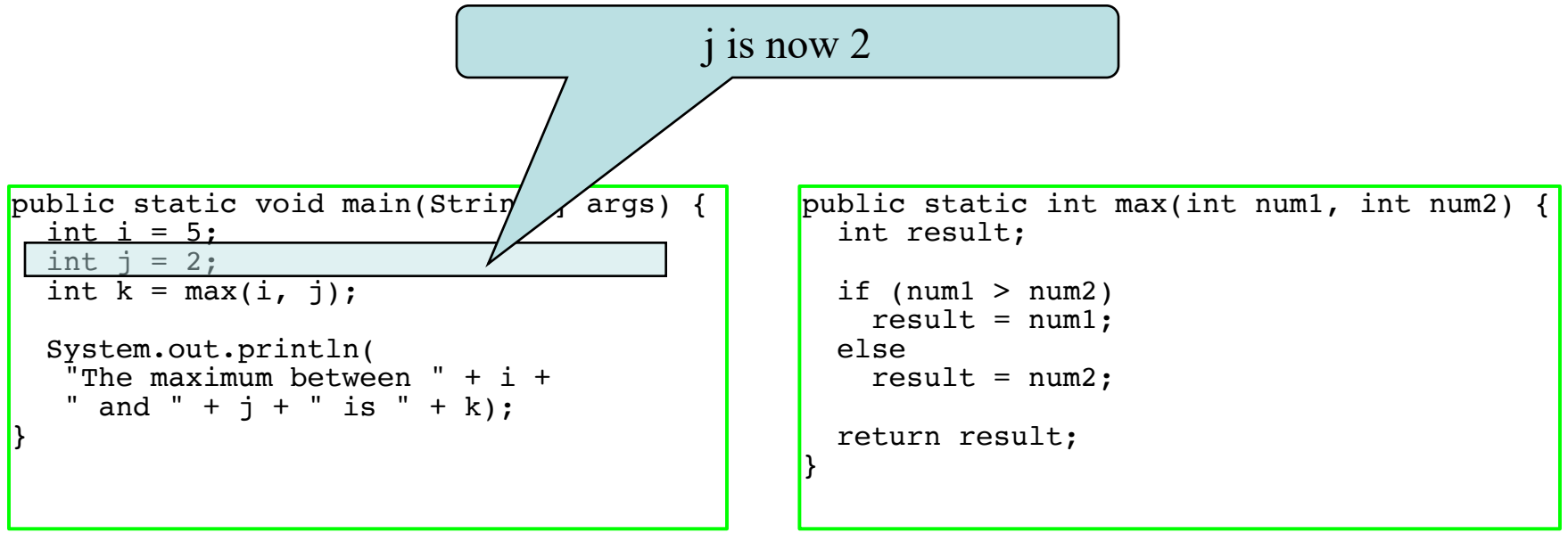
Trace Method Invocation



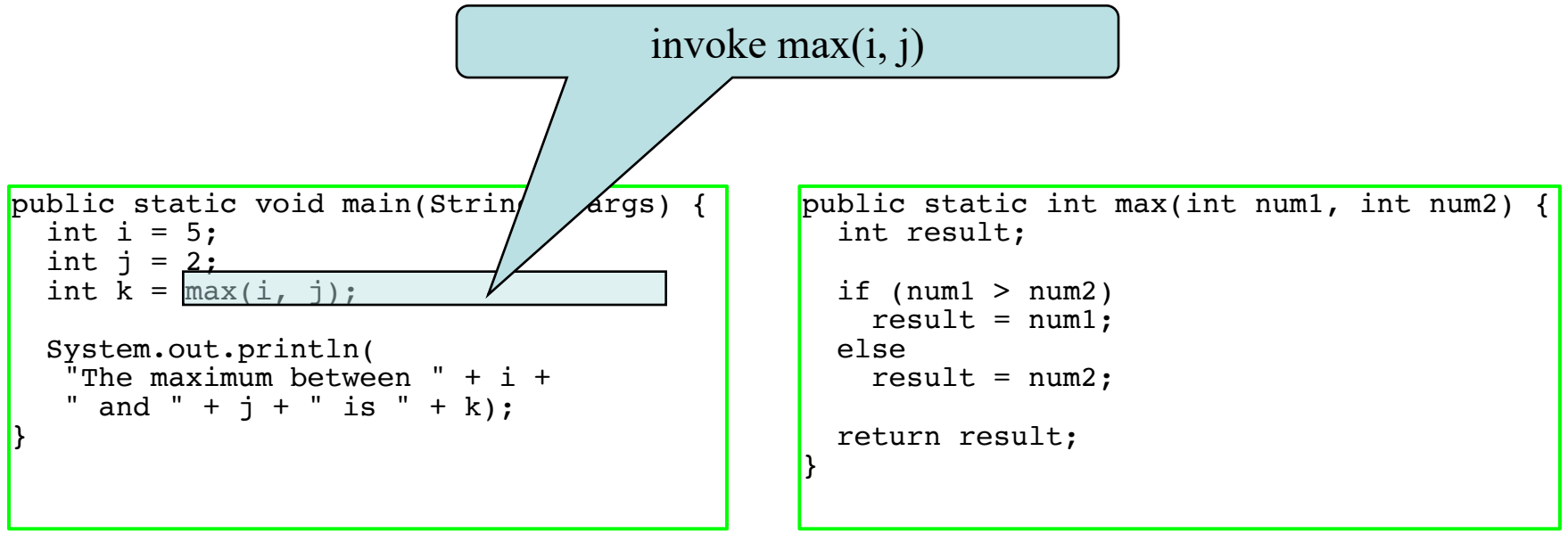

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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



Trace Method Invocation



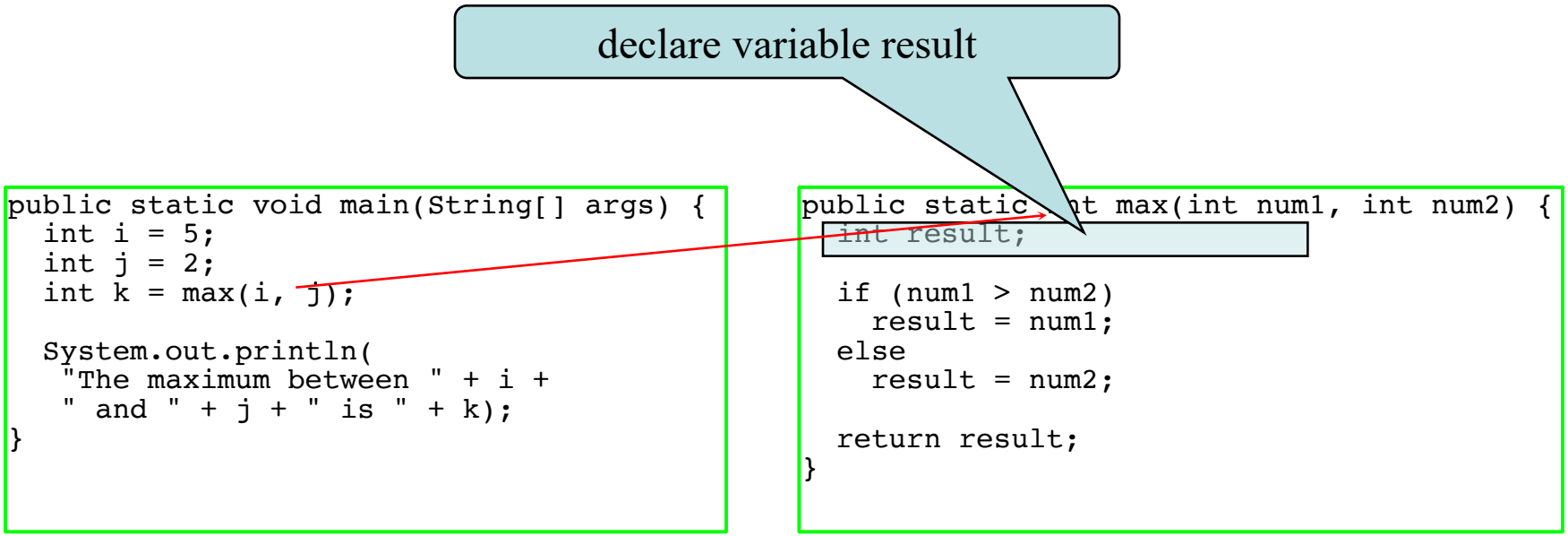
Trace Method Invocation

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



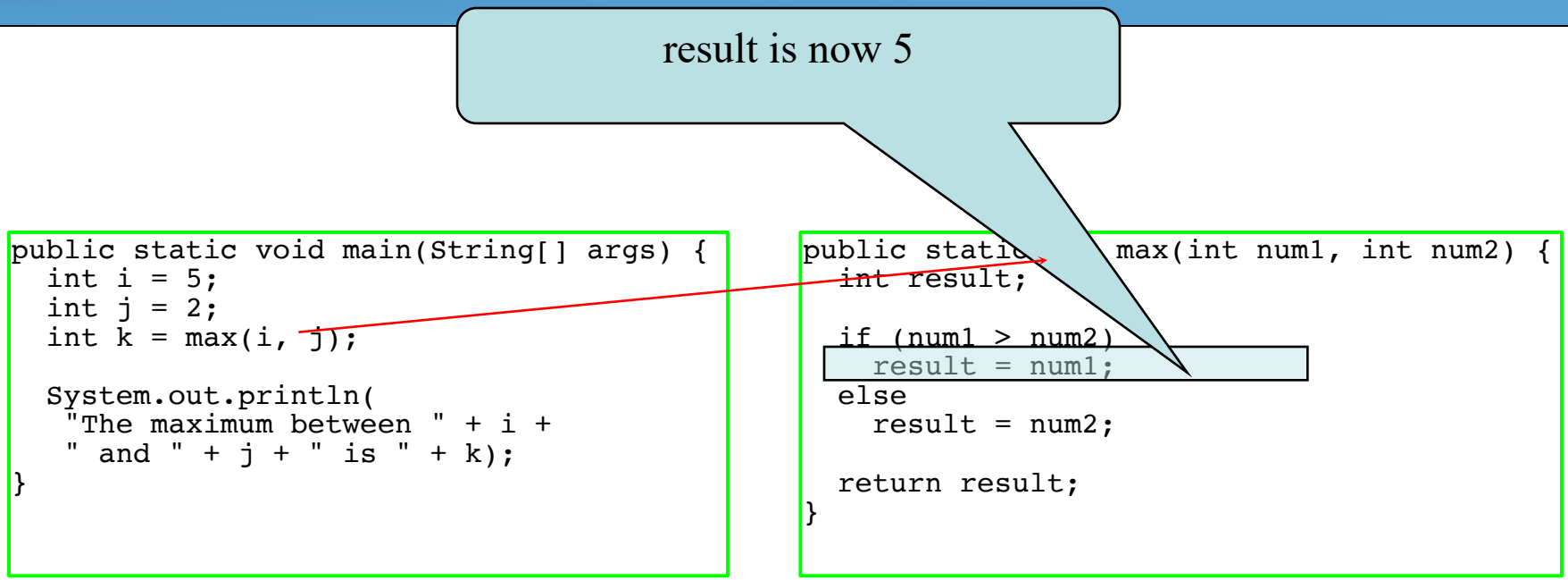
Trace Method Invocation

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

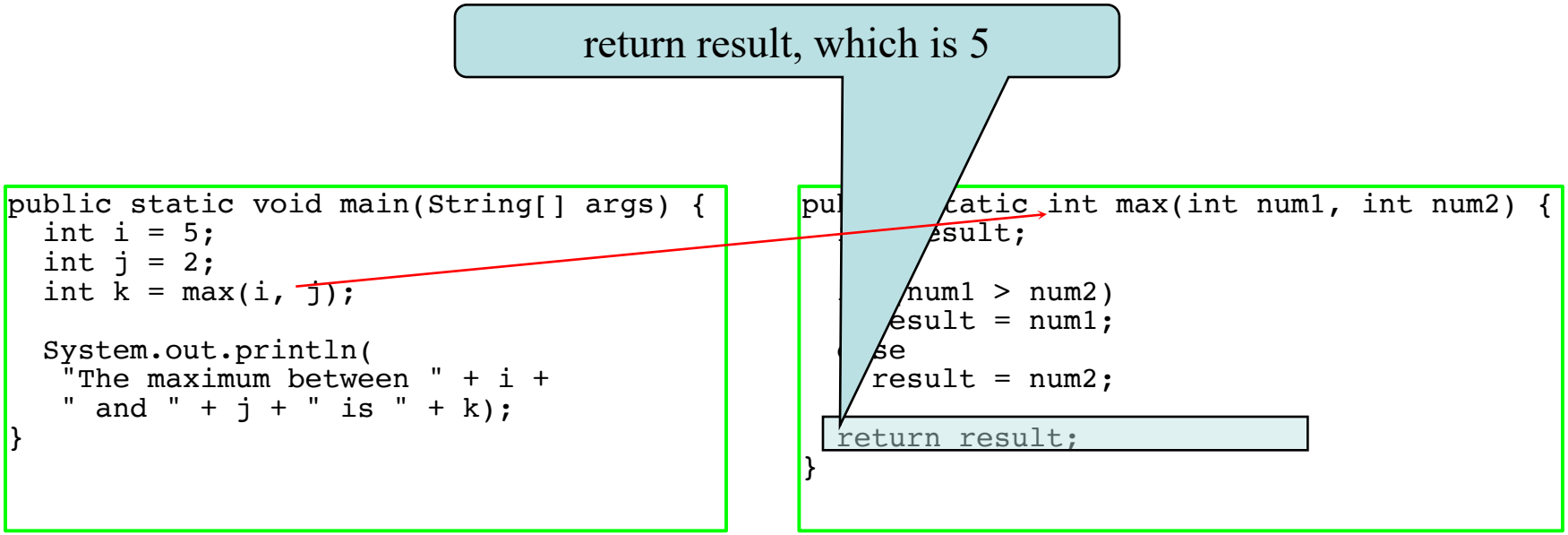
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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;
}
```

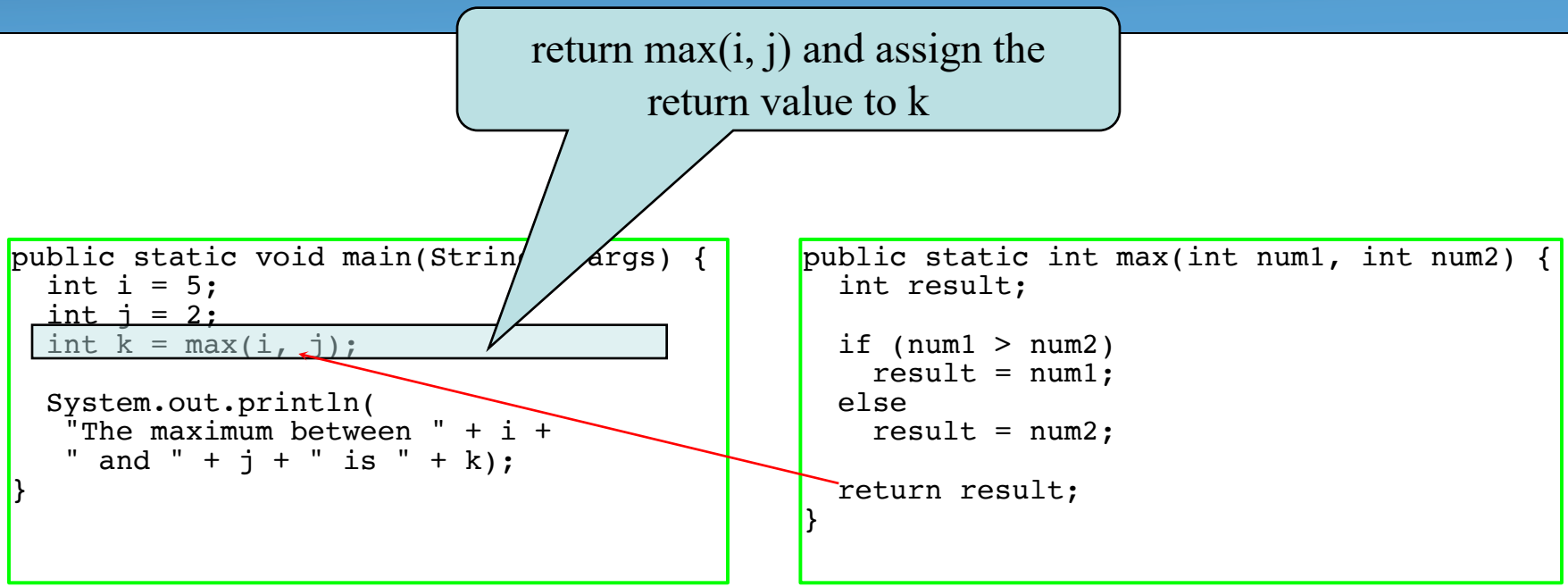
Trace Method Invocation



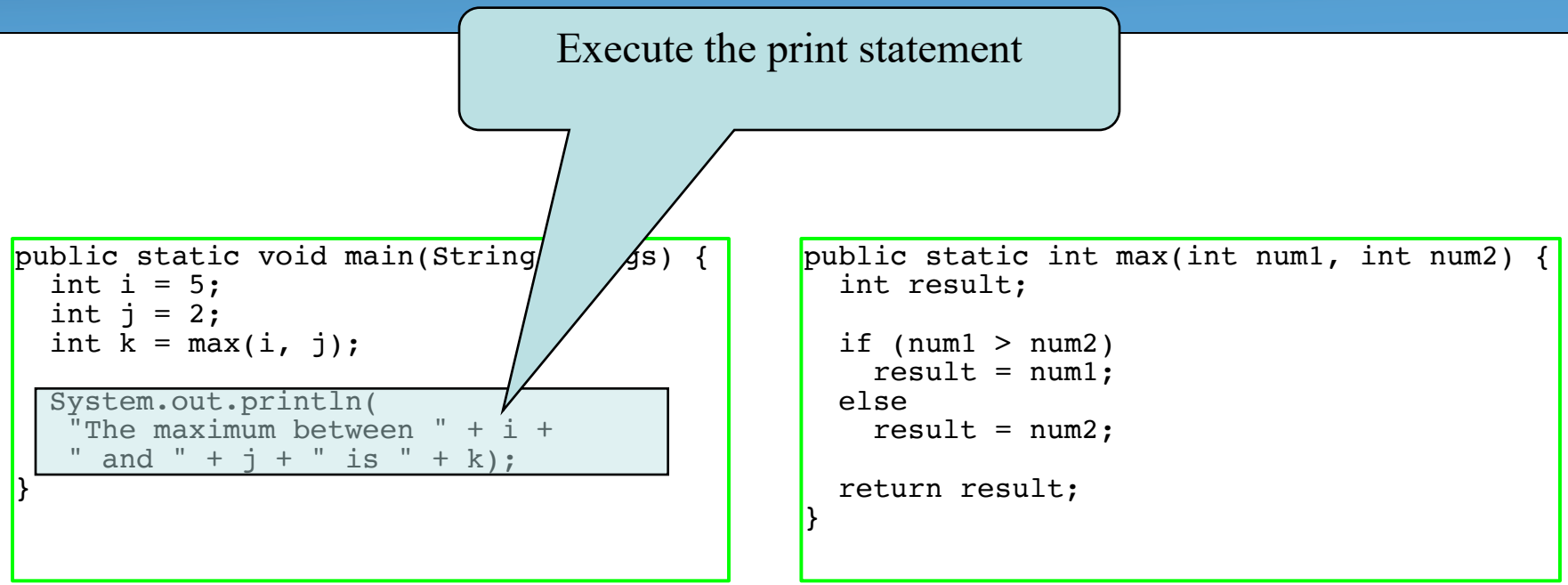
Trace Method Invocation



Trace Method Invocation



Trace Method Invocation



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.