Methods

CSE 114: Introduction to Object-Oriented Programming

Paul Fodor

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

Contents

- Motivation: Opening Problem
- Why write methods?
- Defining Methods:
  - Method Signature: modifiers,
    - Formal and Actual Parameters
    - Return Value Type
  - Calling Methods
- Call Stacks
- Benefits of Methods: Reuse, Method Abstraction and Information hiding
- Call-by-value
- Overloading
- CAUTION: all execution paths need returns (if a value is returned)

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Motivation: Opening Problem

Find multiple sums of integers:

- from 1 to 10,
- from 20 to 30,
- from 35 to 45,

...

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Opening Problem

- Repeat/copy code:

```
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);
```


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Opening Problem Solution

- Use 1 method and invoke it multiple times!

```
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));
}
```


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Why write methods?

- To shorten your programs
  - avoid writing identical code twice or more
- To modularize your programs
  - fully tested methods can be trusted
- To make your programs more:
  - readable
  - reusable
  - testable
  - debugable

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

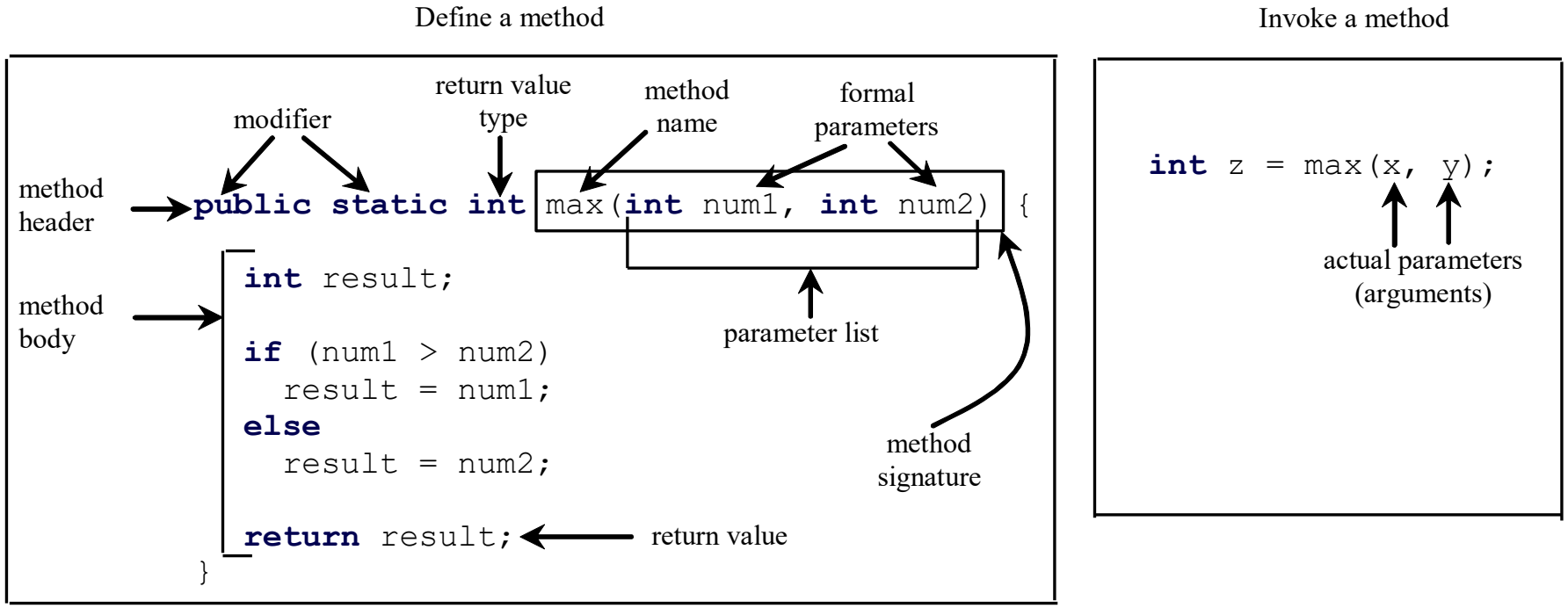
Rule of thumb

- If you have to perform some operation in more than in one place in your program, write a method to implement this operation and have other parts of the program use it

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Defining Methods

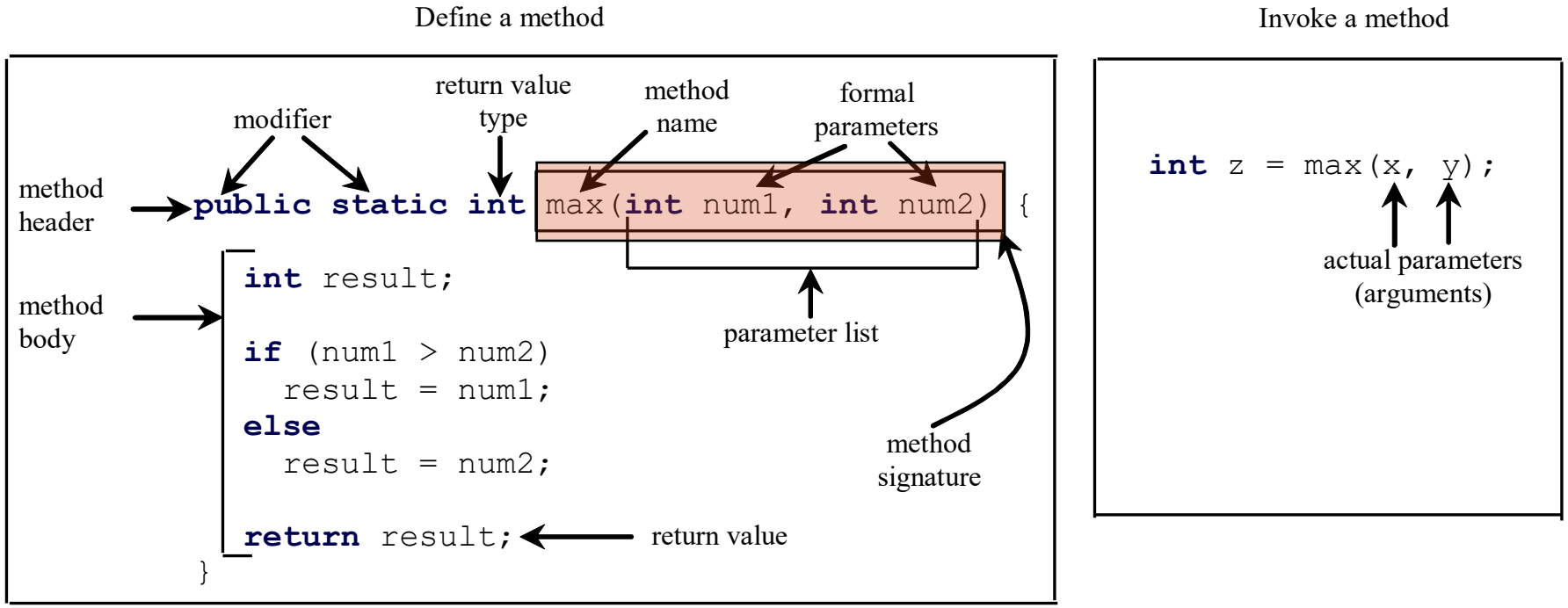
- A *method* is a collection of statements that are grouped together to perform an operation.



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Method Signature

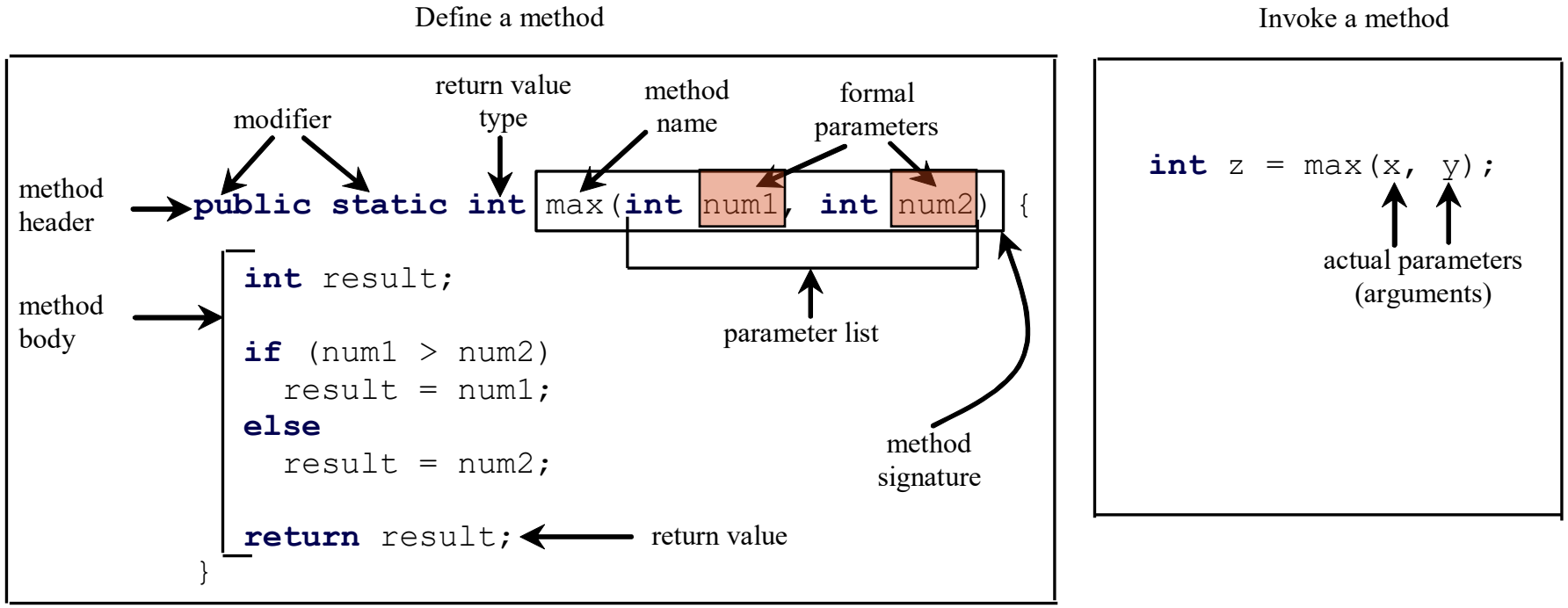
- *Method signature* is the combination of the method name and the parameter list.



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Formal Parameters

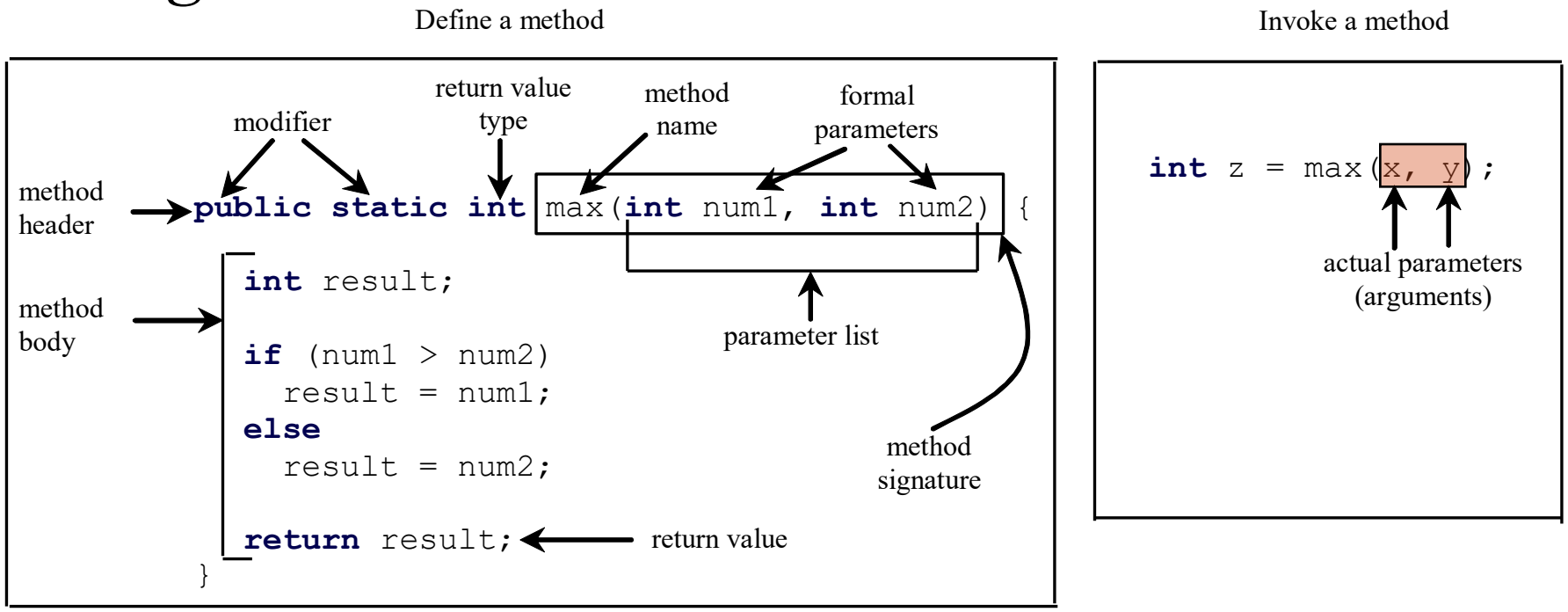
- The variables defined in the method header are known as *formal parameters*.



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Actual Parameters

- When a method is invoked, you pass values to the formal parameter with *actual parameters* or *arguments*.

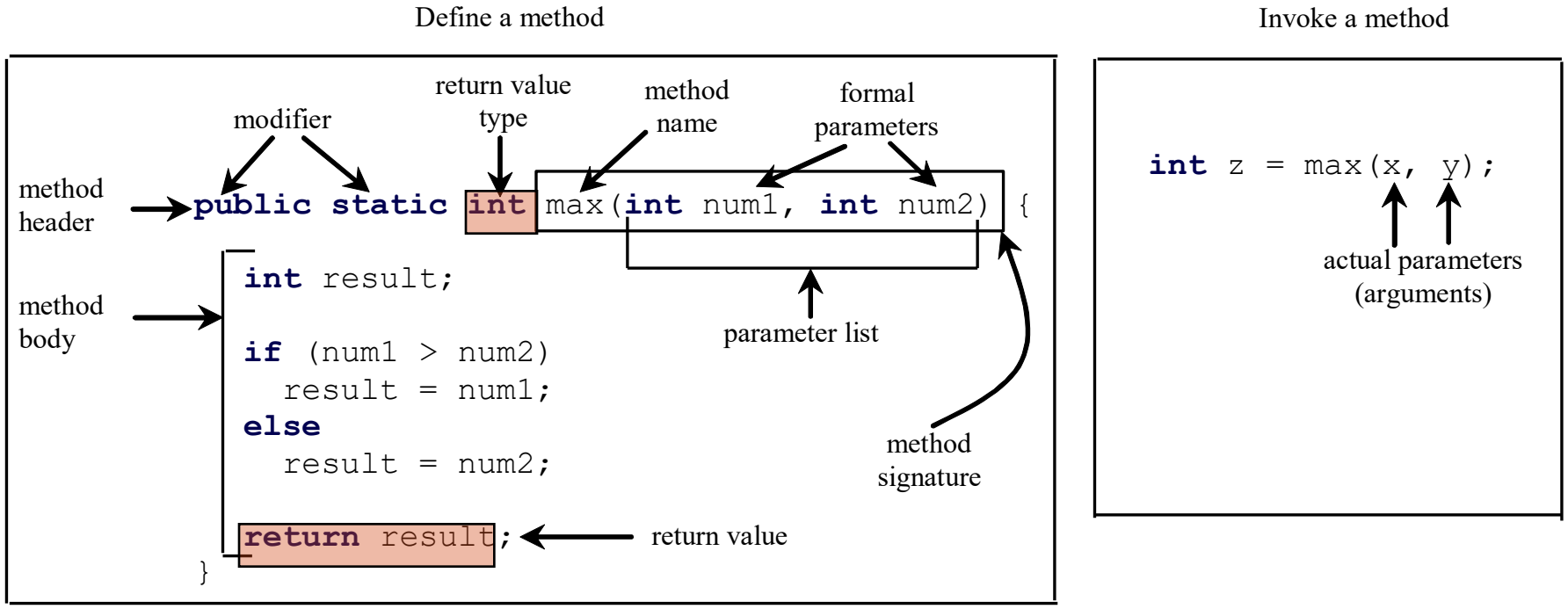


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Return Value Type

- A method may return a value

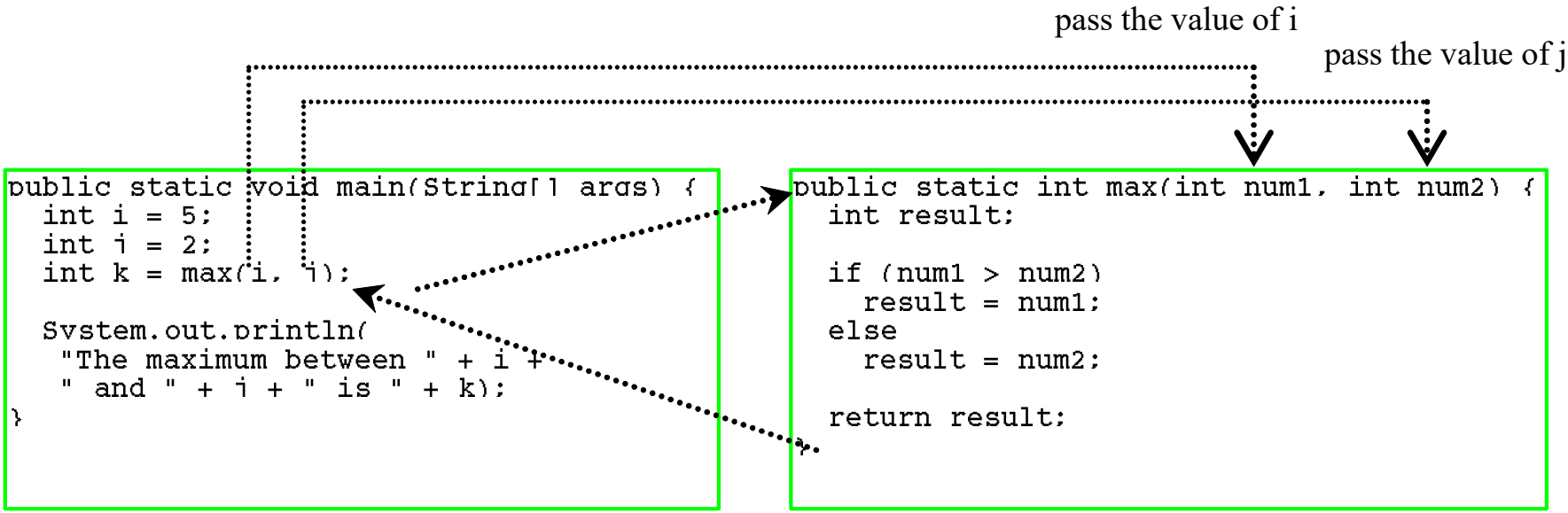
The *returnValueType* is the data type of the value the method returns.



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

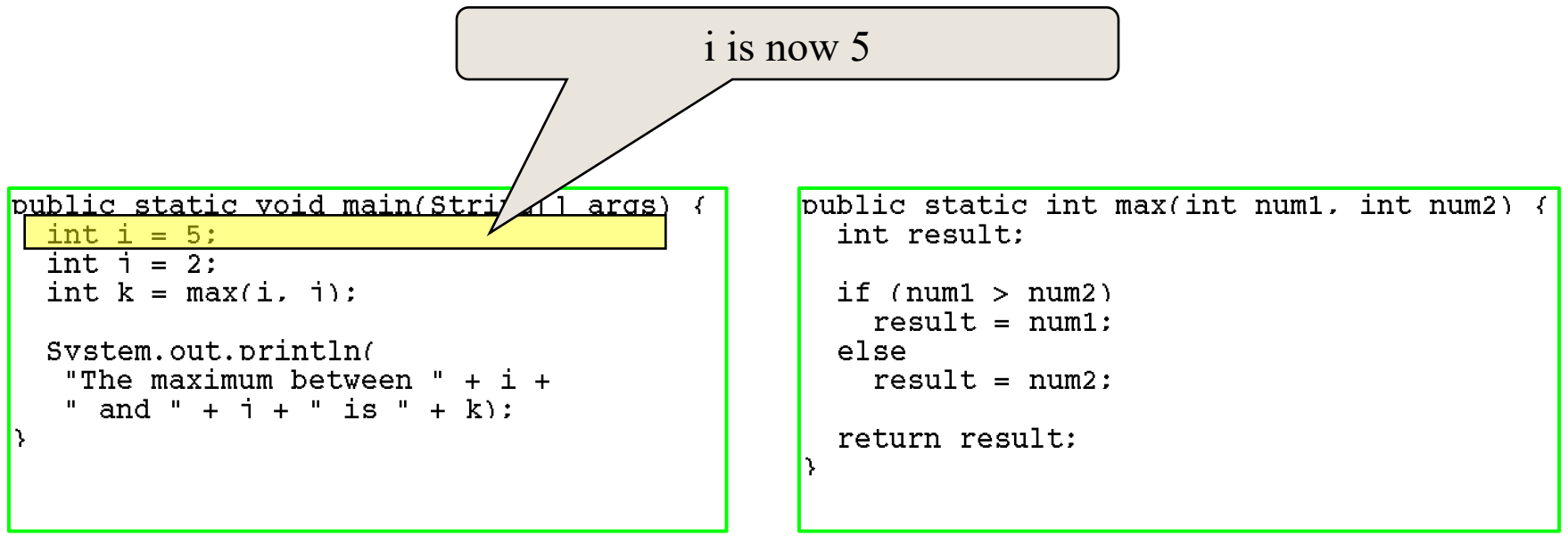
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Calling Methods



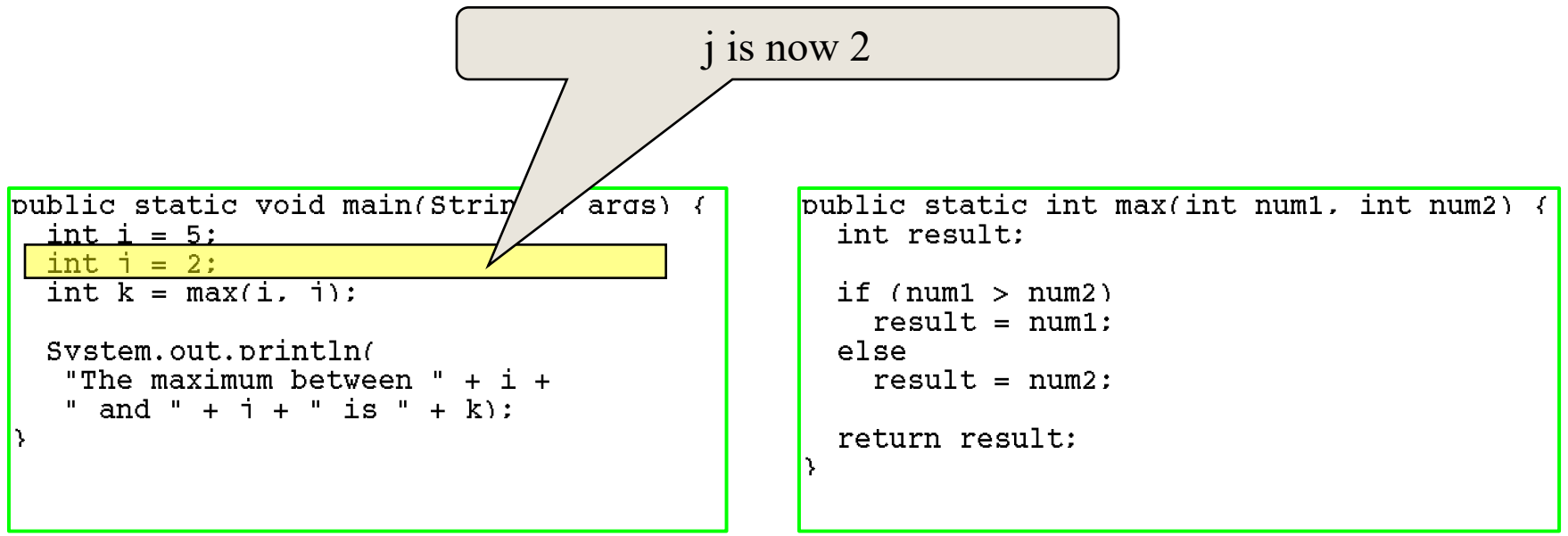
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



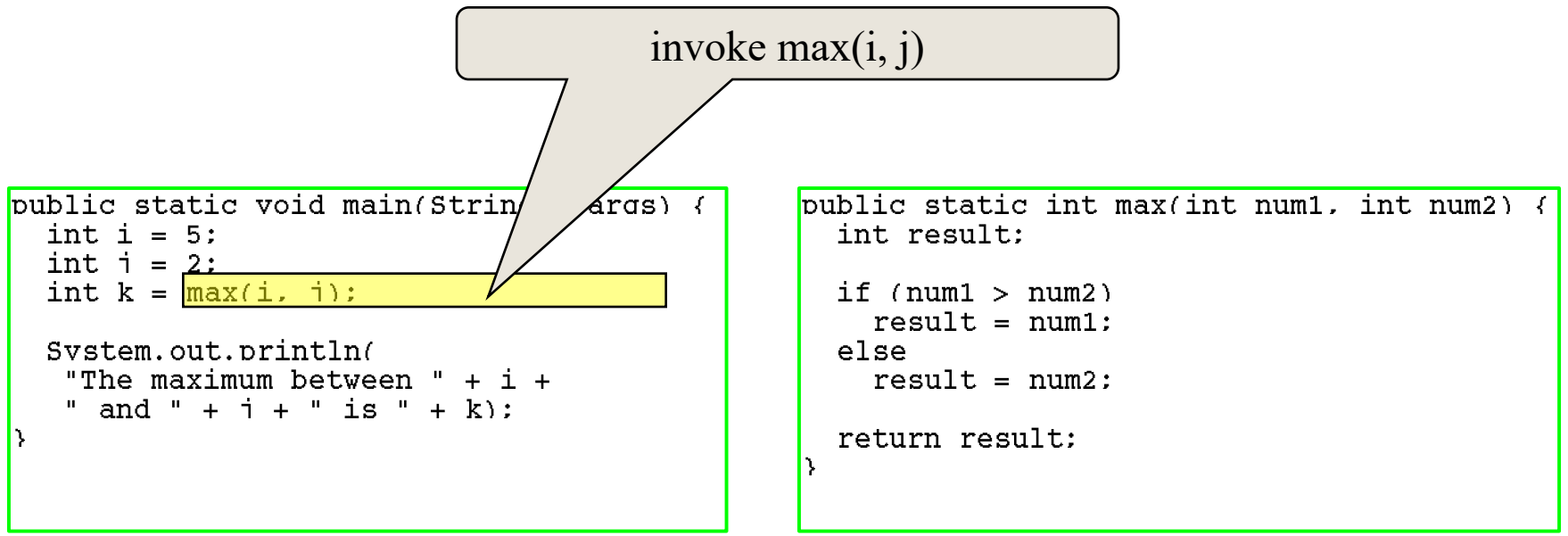
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



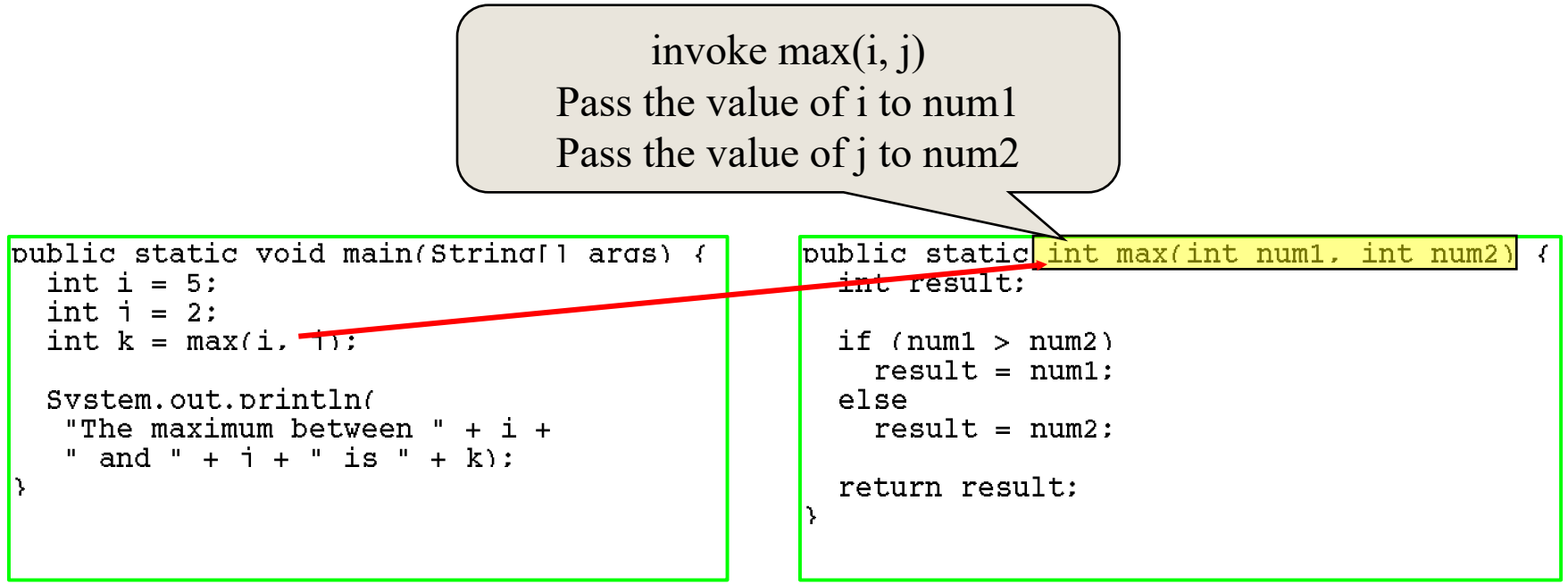
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



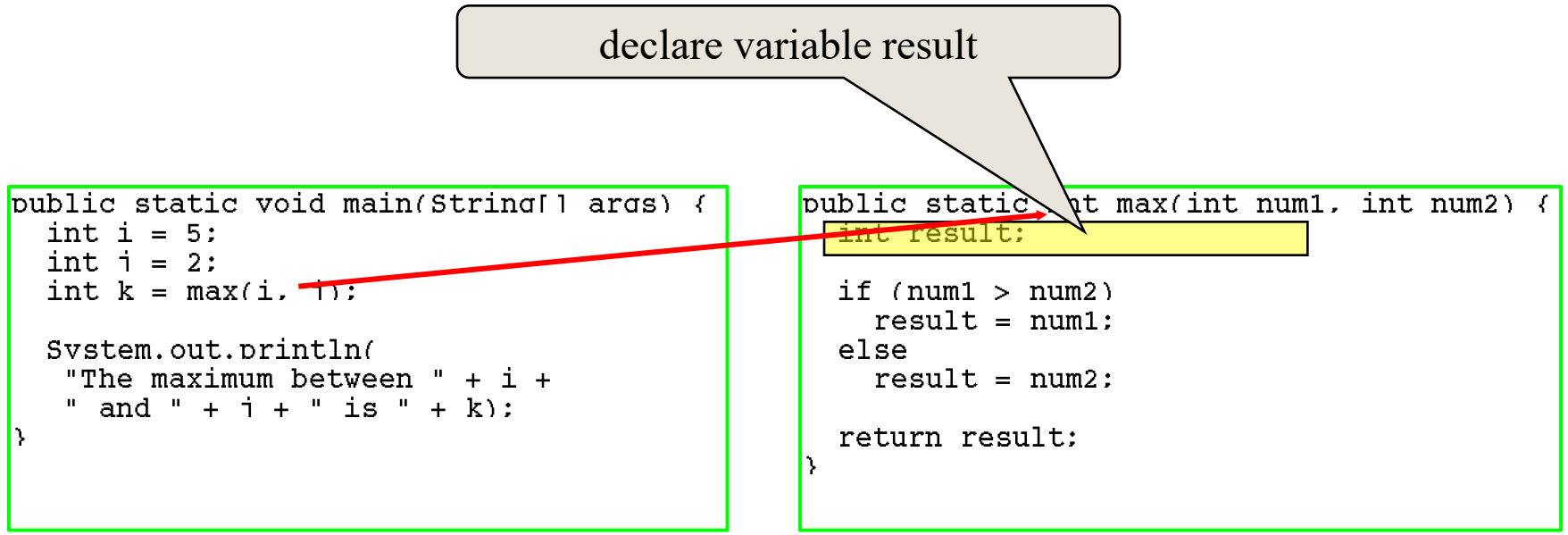
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



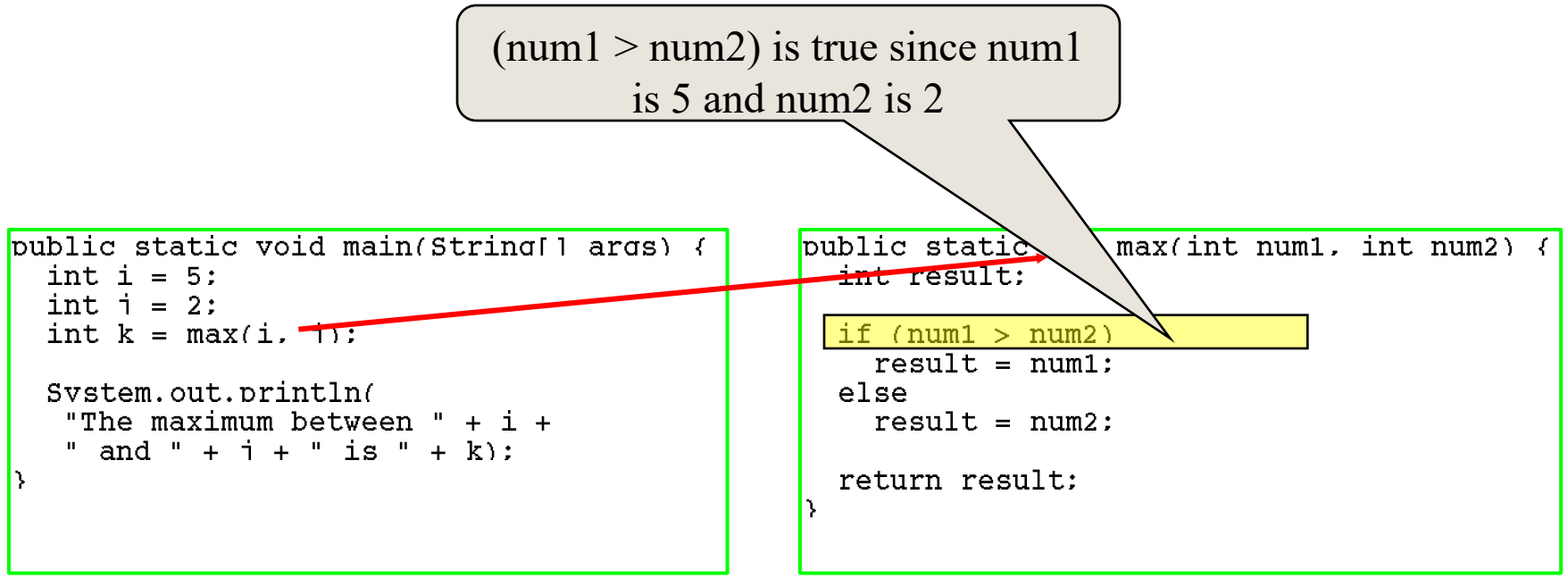
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



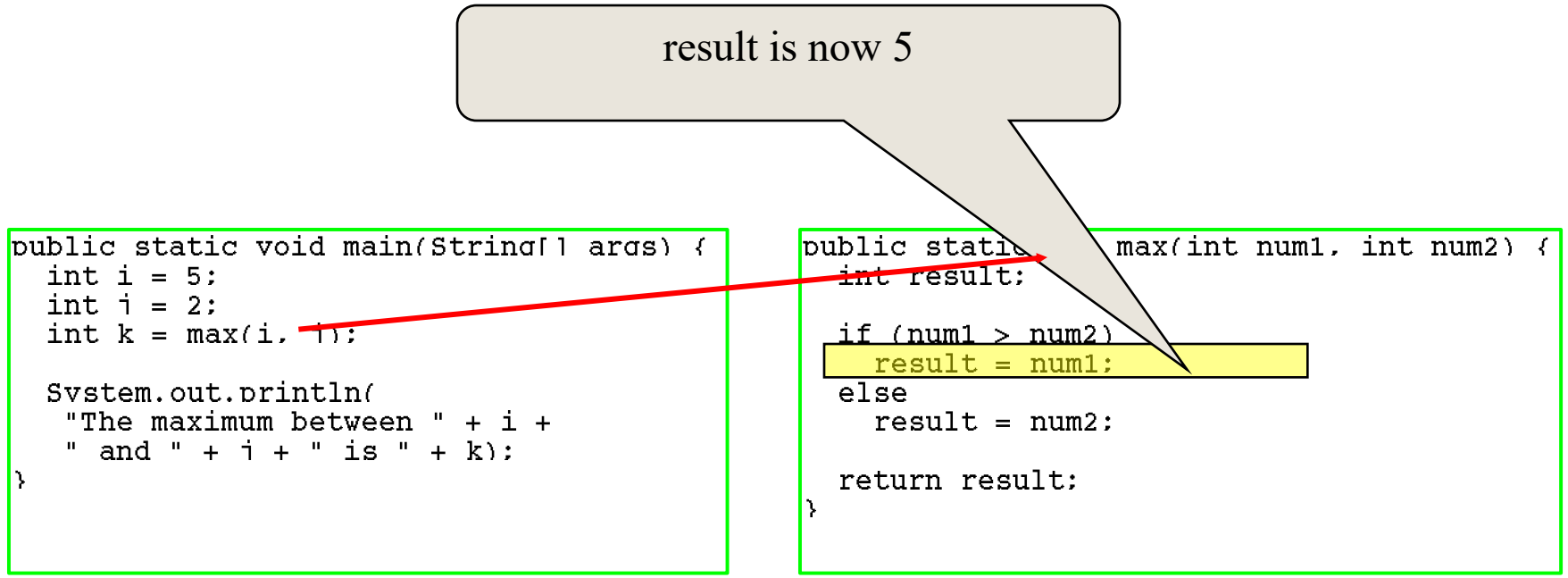
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



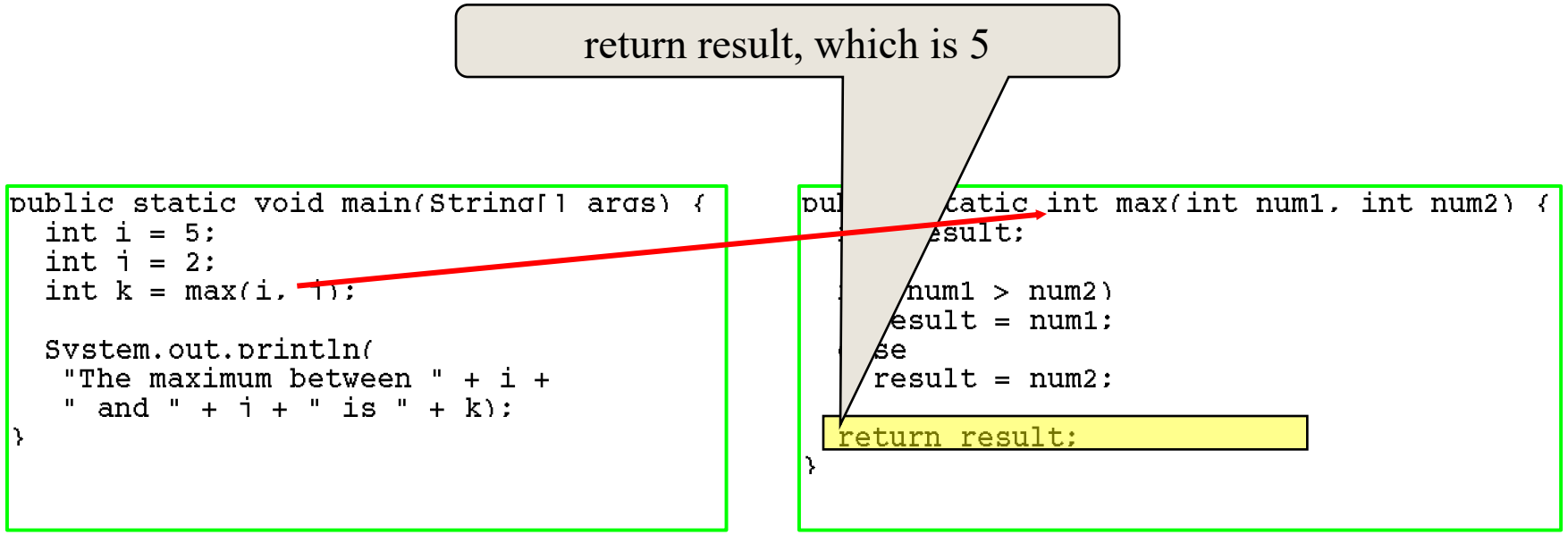
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



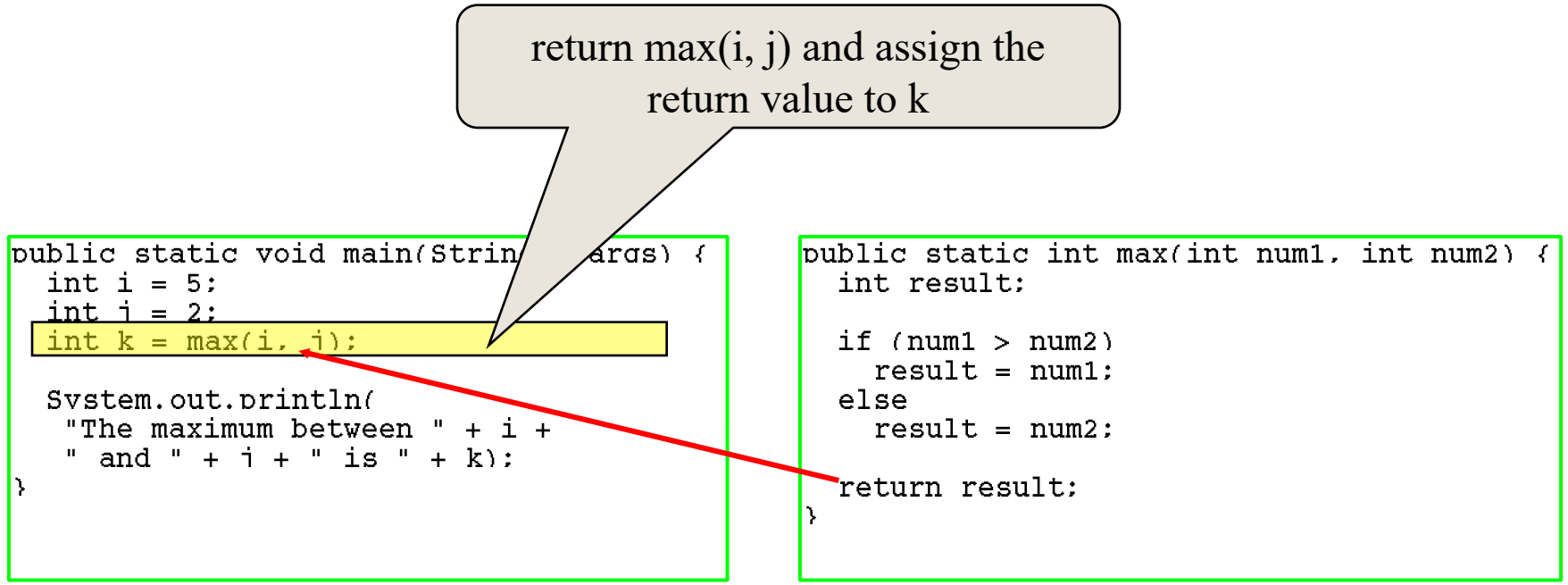
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



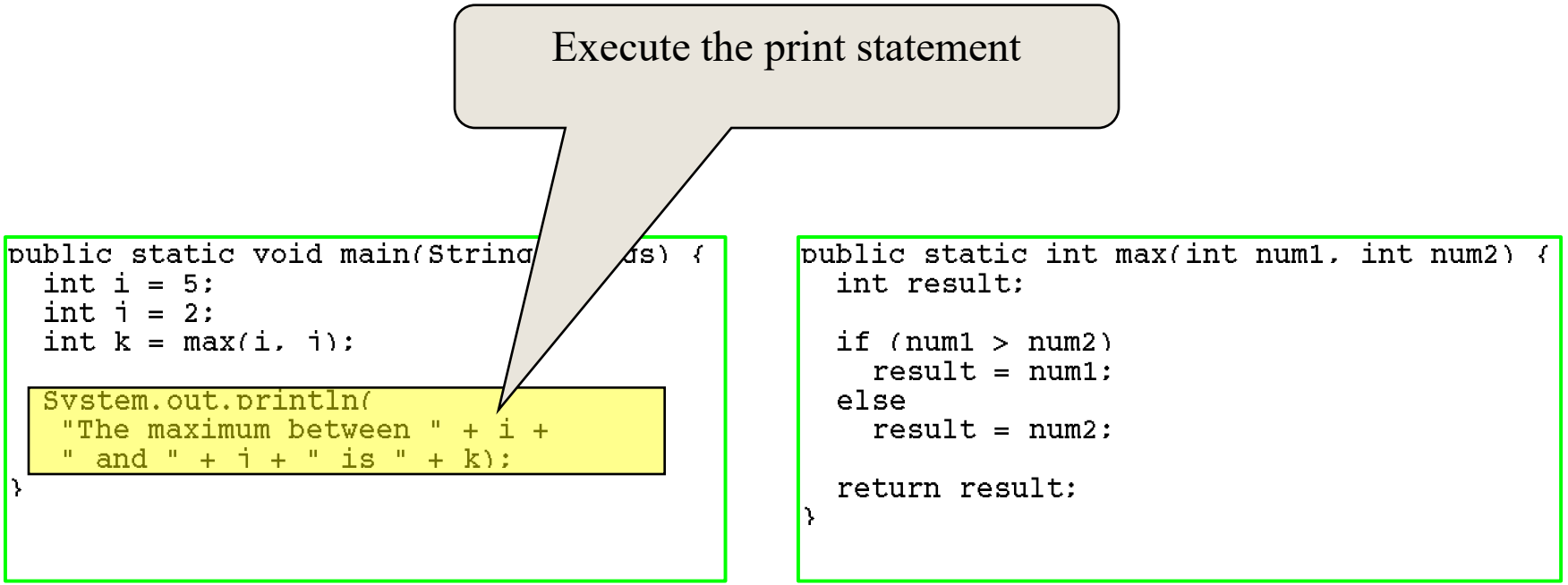
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Trace Method Invocation



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

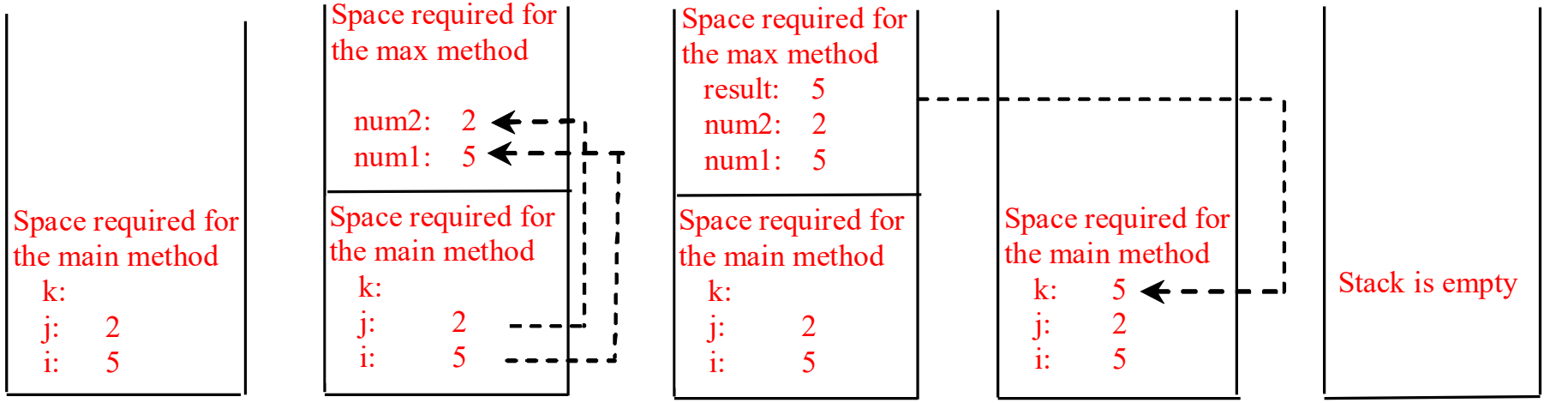
Trace Method Invocation



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Call Stacks

Methods are executed using a stack data structure



(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.

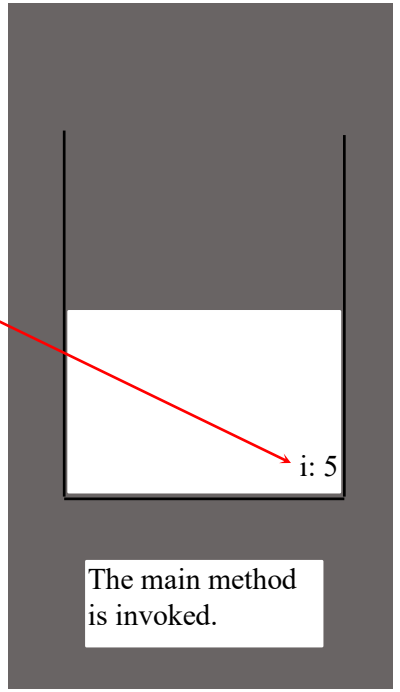
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```

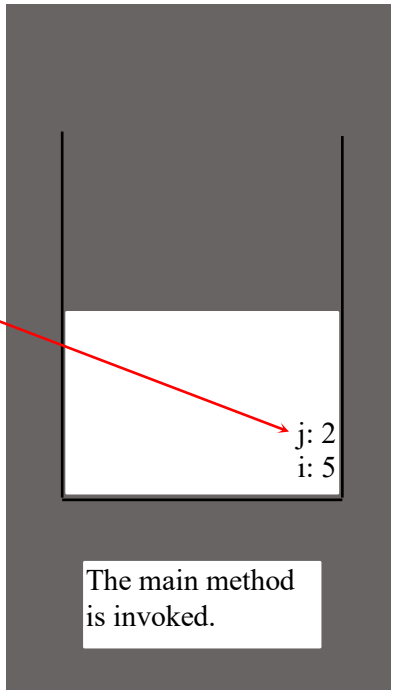


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```

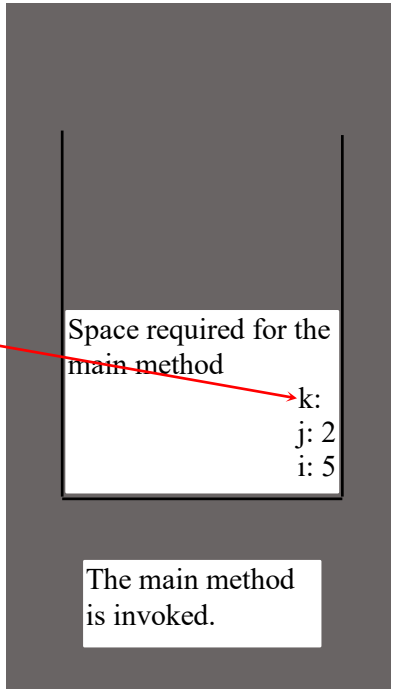


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```

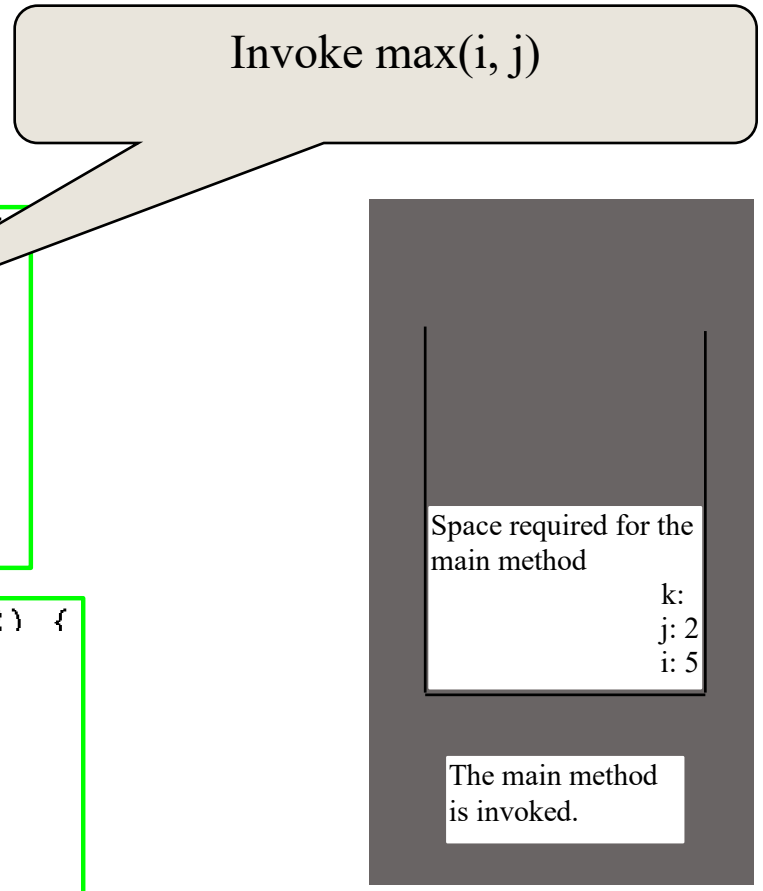


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```

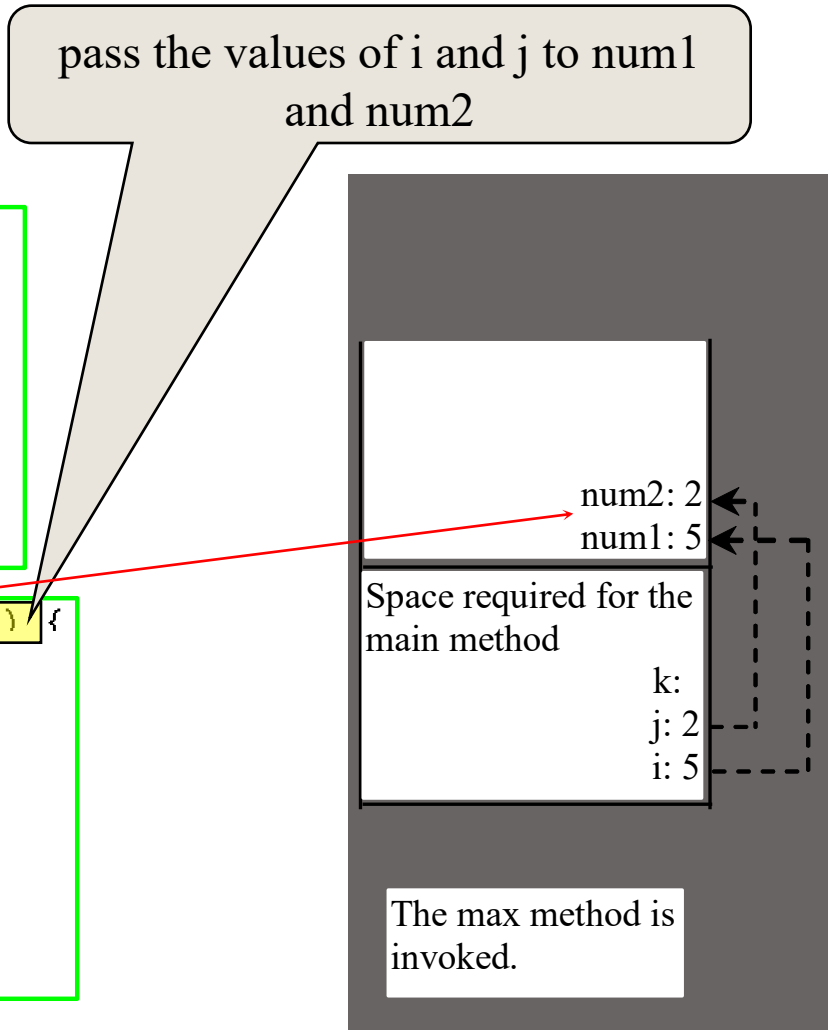


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```

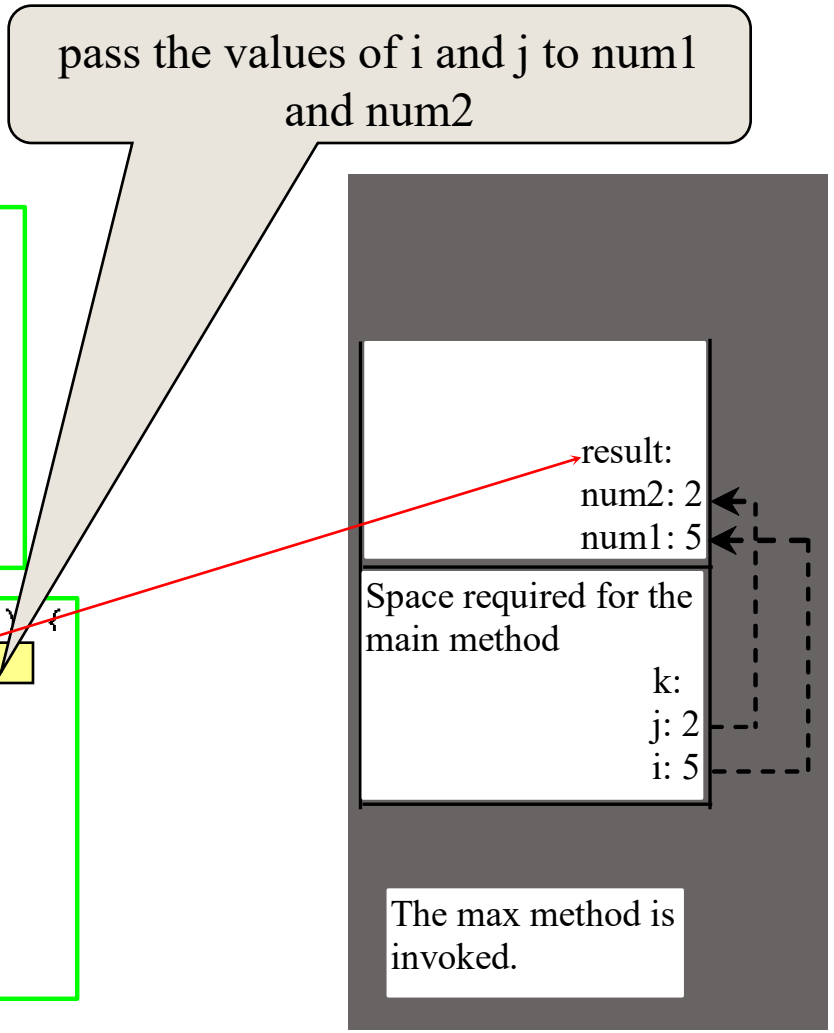


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```



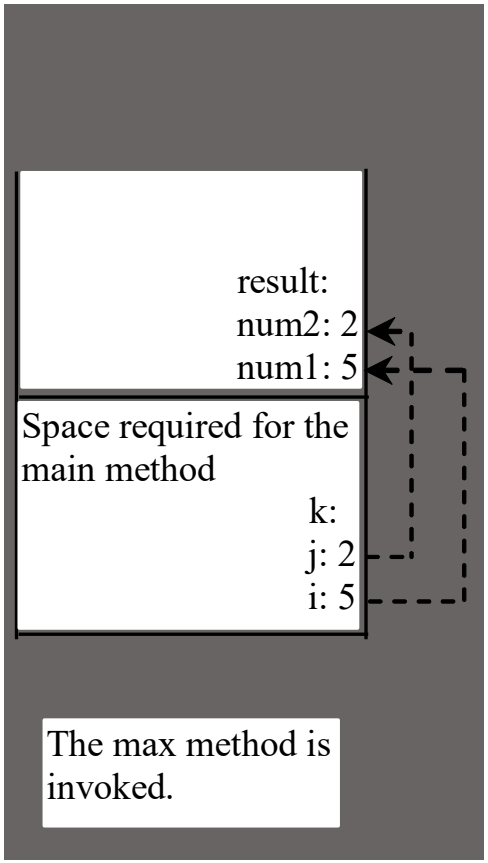
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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

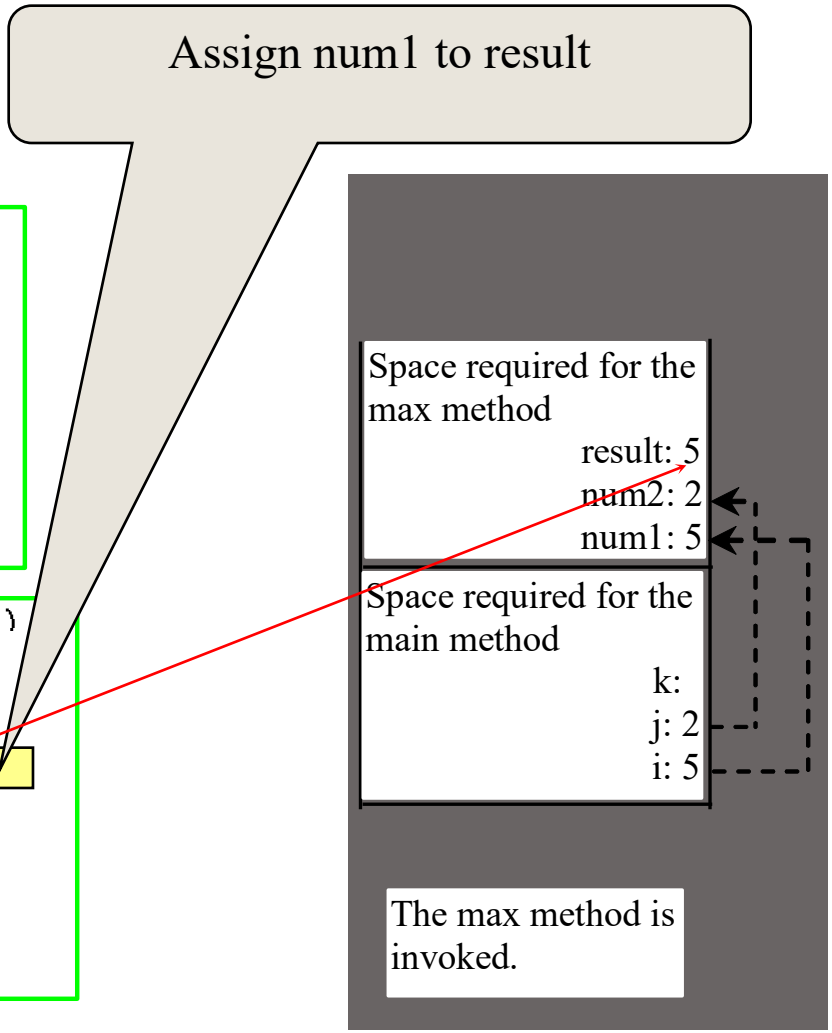


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```

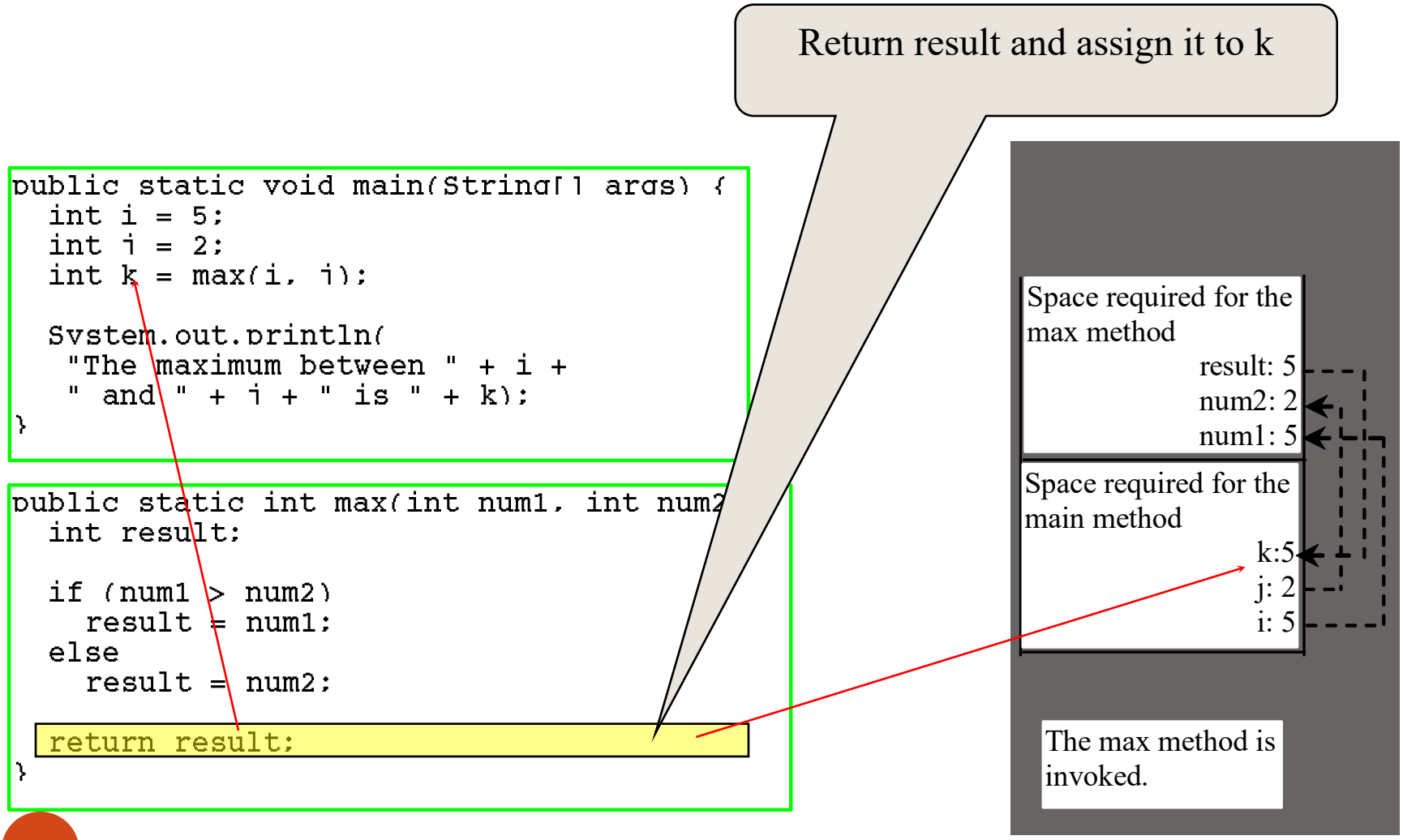


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```



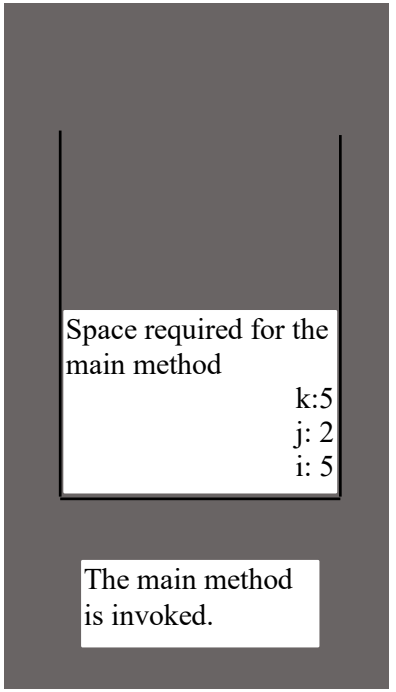
(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

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;
}
```

Execute print statement



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

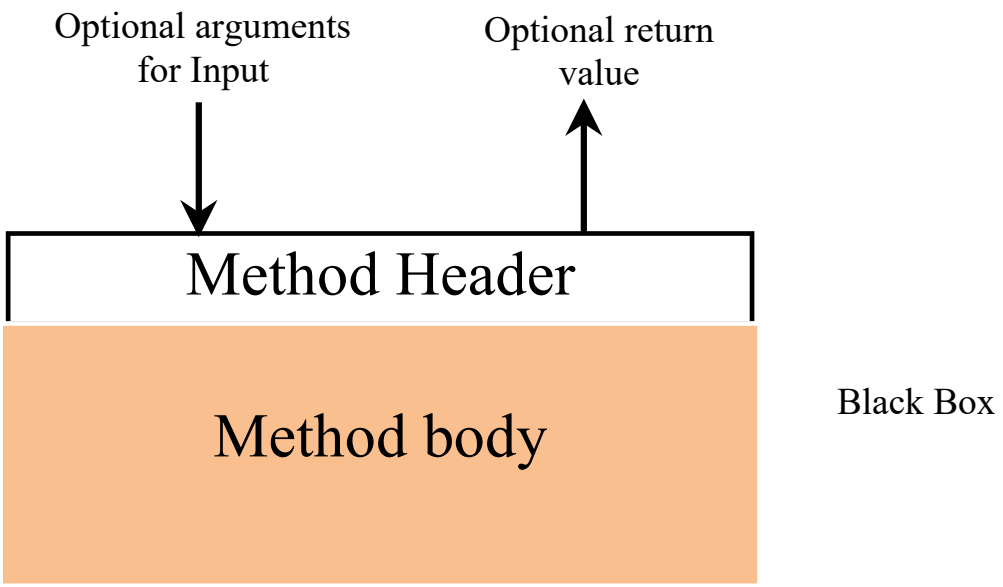
Benefits of Methods

1. Reuse: Write a method once and reuse it anywhere.
2. **Method Abstraction and Information hiding:**
   - Hide the implementation from the user.
     - Reduces complexity of the program.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Method Abstraction

Application Programming Interface (API) = the method body is a black box that contains the detailed implementation for the method.



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Call-by-value

- Method formal parameters are ***copies of the original data (arguments)***.
- Consequence?
  - methods cannot assign („=“) new values to primitive type formal arguments and affect the original passed variables.
- Why?
  - changing argument values changes the copy, not the original.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

```
public class Test {
      public static void main(String[] args) {
            int num = 1;
            m(num);
            System.out.println(num); // still 1
      }
      public static void m(int n) {
            n = 2;
      }
}
```

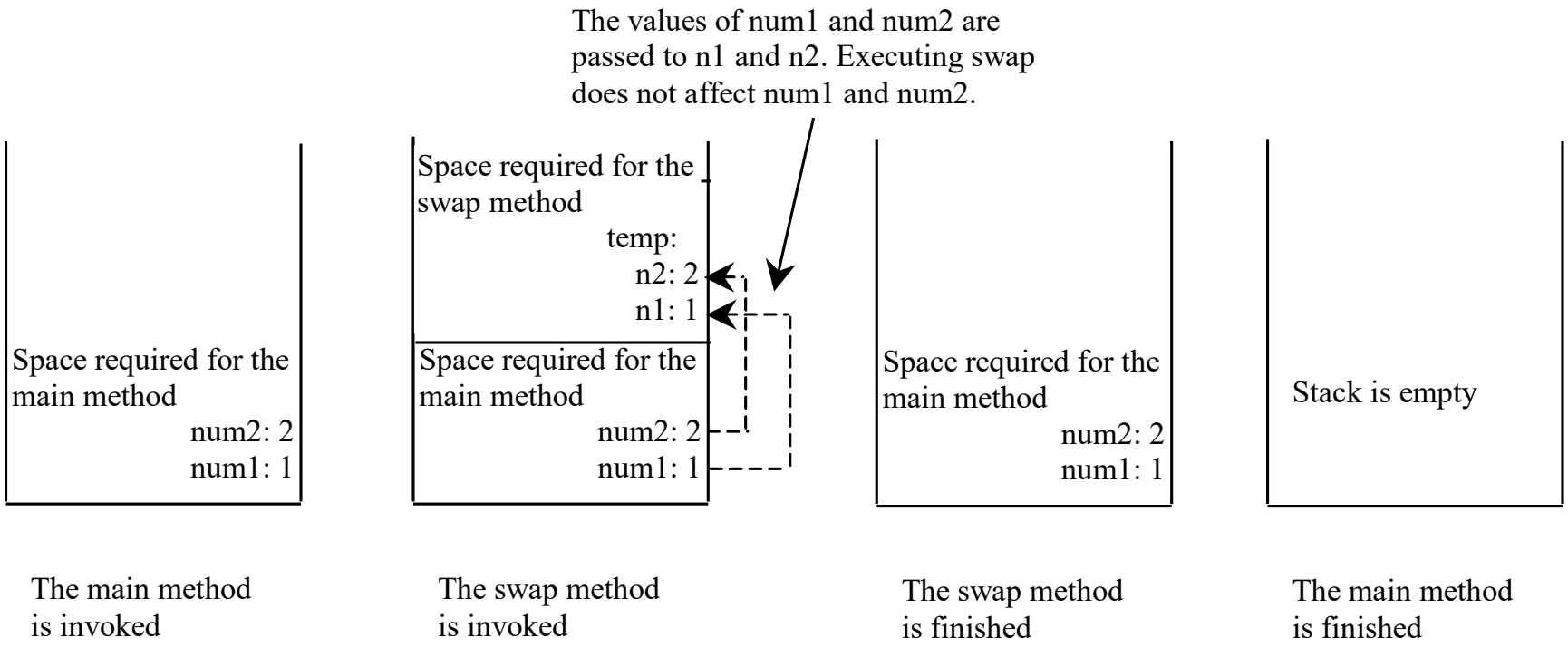

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

A method cannot swap primitive type arguments

```
public class Test {
  public static void main(String[] args) {
        int num1 = 1;
        int num2 = 2;
        swap(num1,num2);
        System.out.println(num1 + " " + num2);//still 1 2
  }
  public static void swap(int n1, int n2) {
        int temp = n1;
        n1 = n2;
        n2 = temp;
        System.out.println(n1 + " " + n2); // 2 1
  }
}
```


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Call-by-value



(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Overloading

- ***Method overloading*** is the ability to create multiple methods of the same name with different signatures and implementations:

```
public class Overloading {
  public static int max(int num1, int num2) {
    if (num1 > num2)
      return num1;
    return num2;
  }
  public static double max(double num1, double num2) {
    if (num1 > num2)
      return num1;
    return num2;
  }
  public static void main(String[] args) {
    System.out.println(max(1, 2)); // 2 (as an int)
    System.out.println(max(4, 2.3)); // 4.0 (as a double)
  }
}
```


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Overloading

- Overloaded methods must differ either by the types of their parameters or by their number of arguments)
- Method ***matching*** is the process to find the method implementation for the call:
  - it uses a "***best match***" algorithm to cast the actual parameters' types to the formal parameter types
  - For example:

```
System.out.println(max(1, 2)); // matches the ints method
System.out.println(max(4, 2.3)); // matches the doubles one
```


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Overloading & 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*. and it is a compilation error.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

Overloading & Ambiguous Invocation

```
public class AmbiguousOverloading {
  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;
  }
  public static void main(String[] args) {
    System.out.println(max(1, 2)); // compiler error here
                       // it matches equally both methods
  }
}
```


(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)

CAUTION: all execution paths

- A return statement is required for a value-returning method.

The method shown below has a compilation error because the Java compiler thinks it possible that this method does not return any value if the condition is false in the last if statement.

```
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.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)