UNSW Business School
Information Systems and Technology Management

INFS1609/2609 Fundamentals of Business Programming

Lecture 6
Methods

yenni.tim@unsw.edu.au

INFS1609/2609 Fundamentals of Business Programming

Topics for this week:

- To understand the concept of method abstraction in program development
- To define methods with formal parameters
- To invoke methods with actual parameters (i.e., arguments)
- To define methods with and without a return value
- To pass arguments by value
- To use method overloading and understand ambiguous overloading
- To determine the scope of variables

Main references
- *Textbook: Chapter 6*
- *Other online references posted on Ed*

Opening Problem

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

You have learned to do this:

```
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);
```

You have learned to do this:

Computing these sums are very similar, except that the starting and ending integers are different

```
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);
```

We can group them into **methods!**

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

With the method, we just have to **invoke it** to compute the sum:

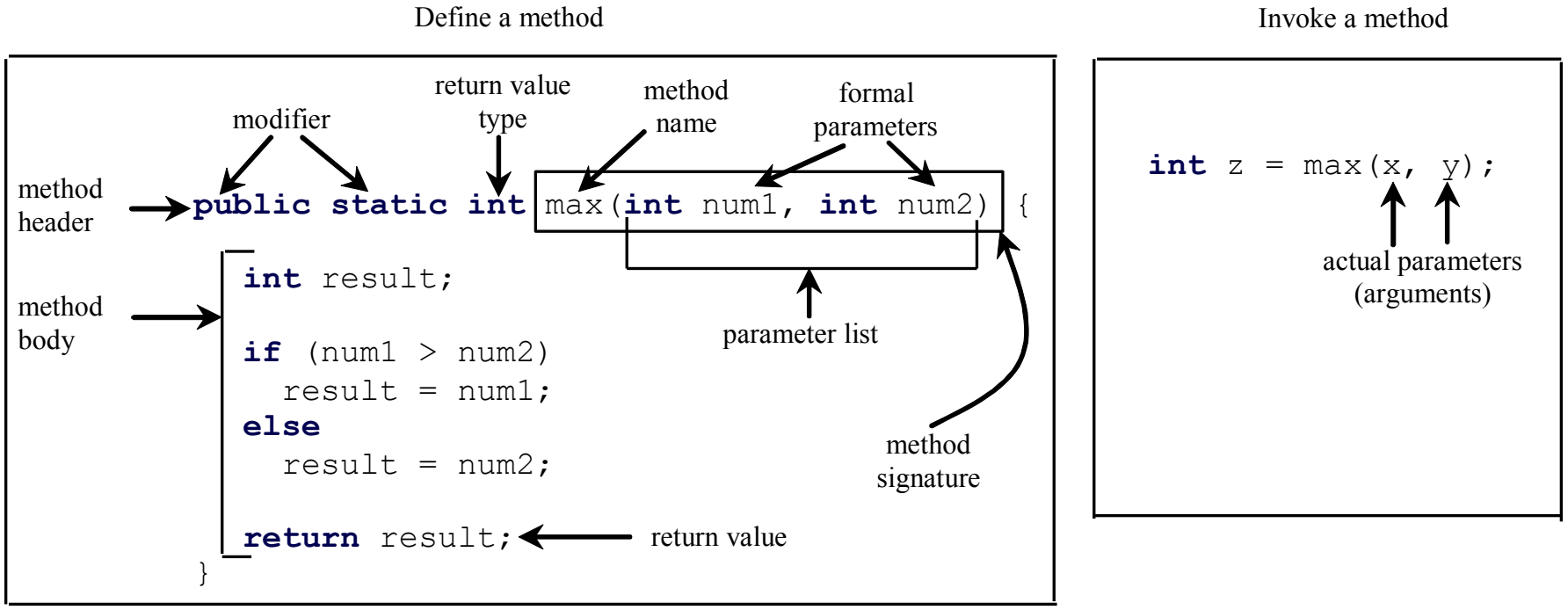
```
public static void main(String[] args){
 System.out.println("Sum from 1 to 10 is " +
                    sum(1, 10));
 System.out.println("Sum from 20 to 30 is " +
                    sum(20, 30));
 System.out.println("Sum from 35 to 45 is " +
                    sum(35, 45));
}
```

Benefits of Methods:

- Write a method once, you can **reuse** it anywhere
- Information hiding (**abstraction**) to hide the detailed implementation from the user
- Reduce **complexity** and redundant coding (modularize codes and improve the quality of the program)

Method

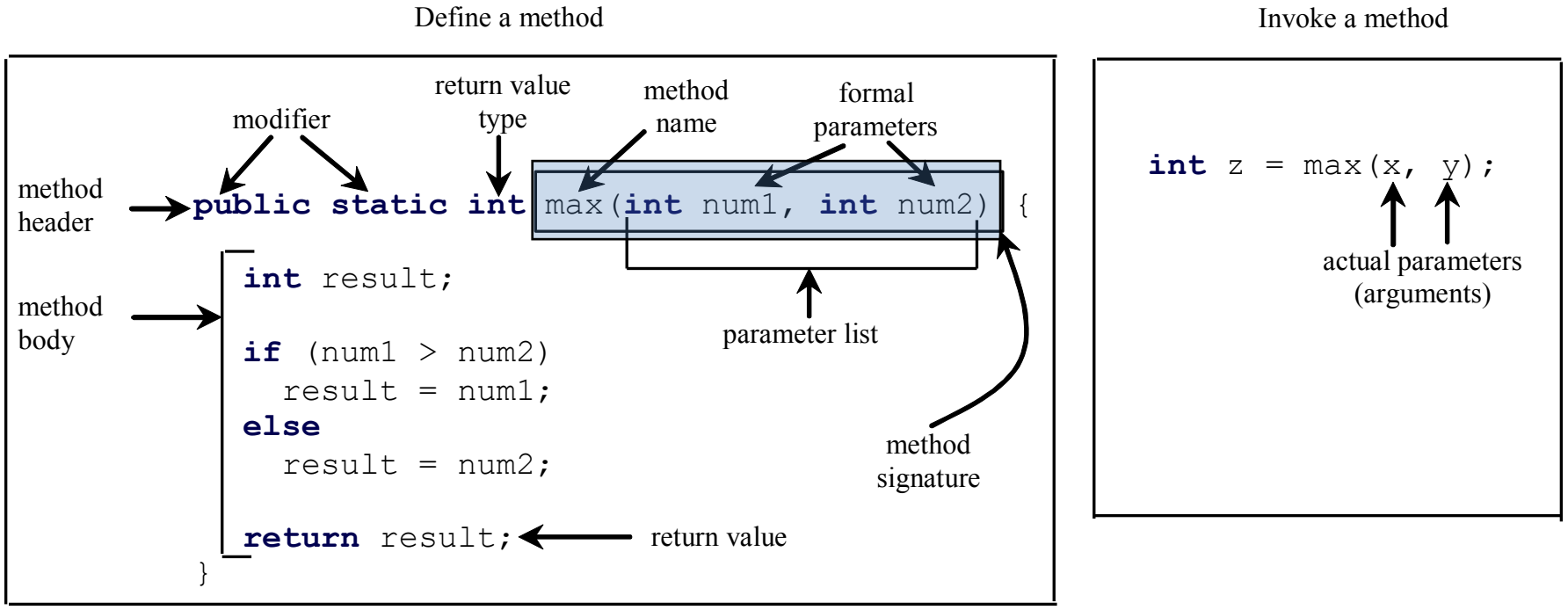
A method is **a collection of statements** that are grouped together to perform an **operation**



Method Signature

Watch Supplementary Videos on Ed

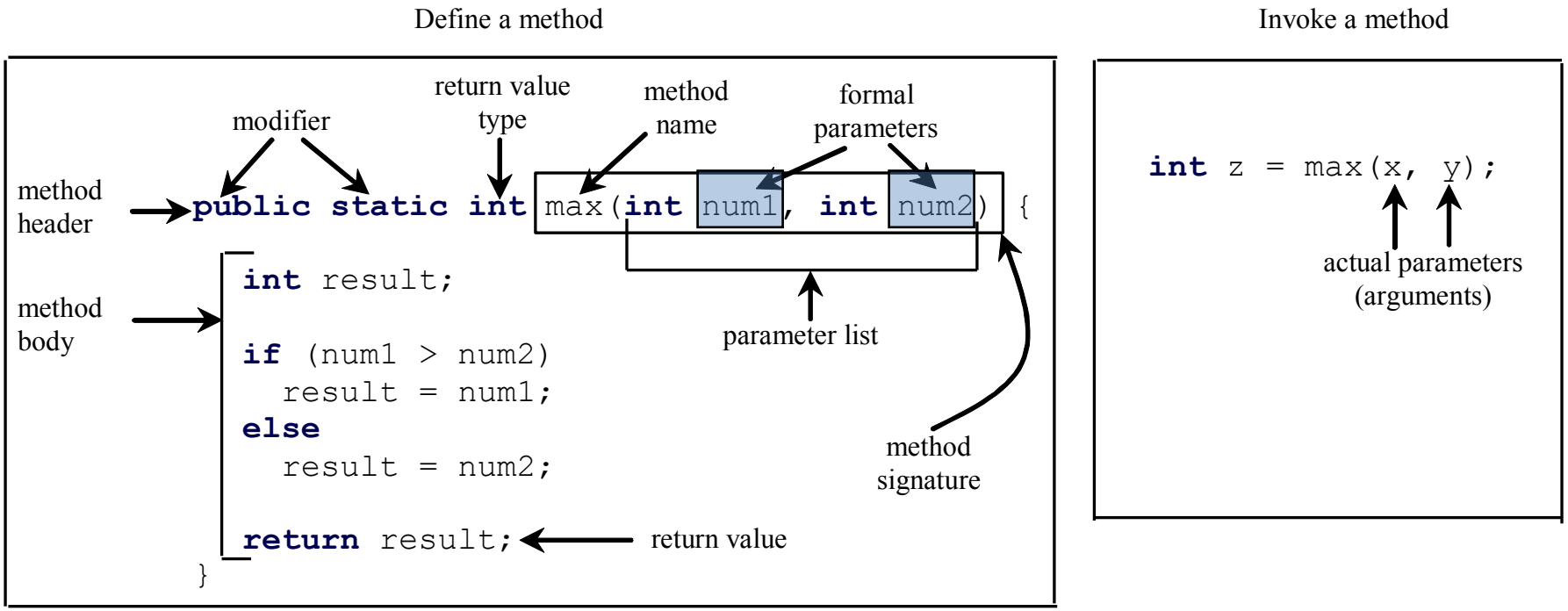
Combination of the method name and the parameter list



(Formal) Parameters

- Variables defined in the method header are known as formal parameters (or simply *parameters*)
- You can use any data type for a parameter of a method
- When you declare a parameter to a method, you provide a **name** for that parameter
- This name is used within the method body to refer to the ***passed-in argument***

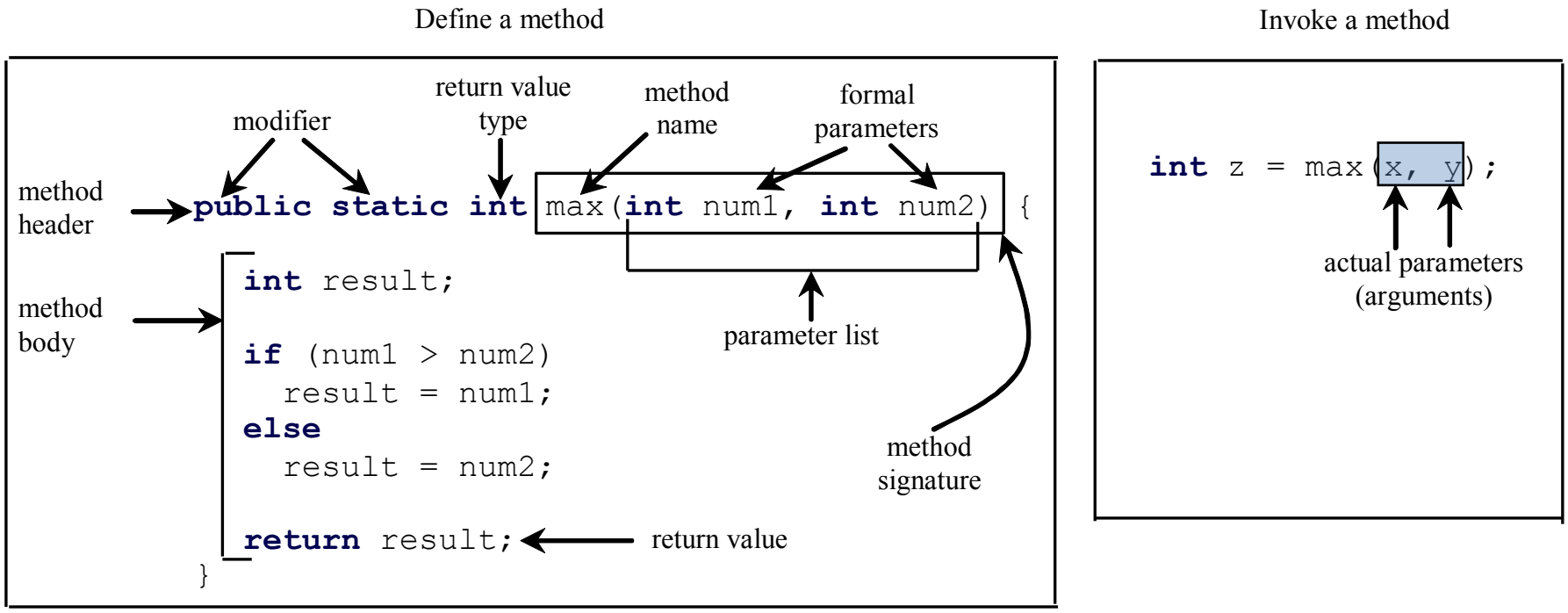
(Formal) Parameters:



(Actual) Parameters or Arguments

- When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or* ***argument***
- When calling a method, you provide arguments, which must be given in the same order as their respective parameters in the method signature, i.e., **match the declaration's parameters** in **type** and **order**

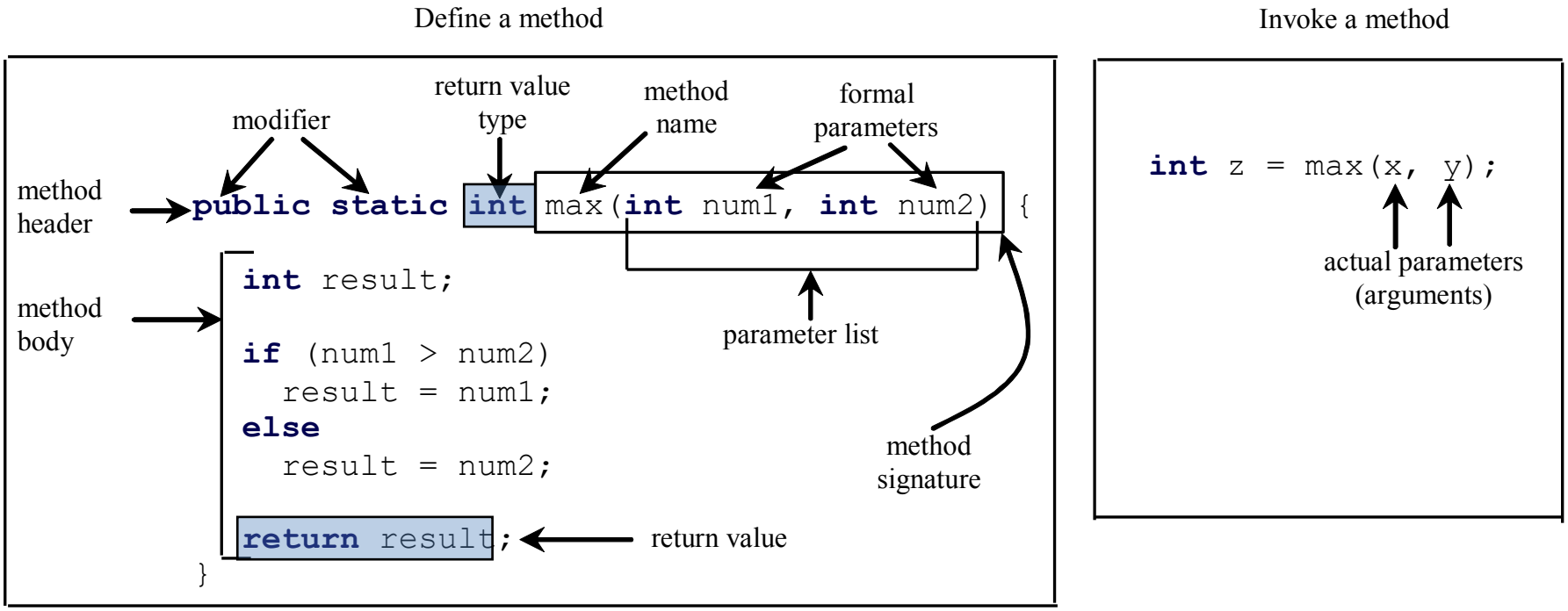
(Actual) Parameters or **Argument:**



Return Value

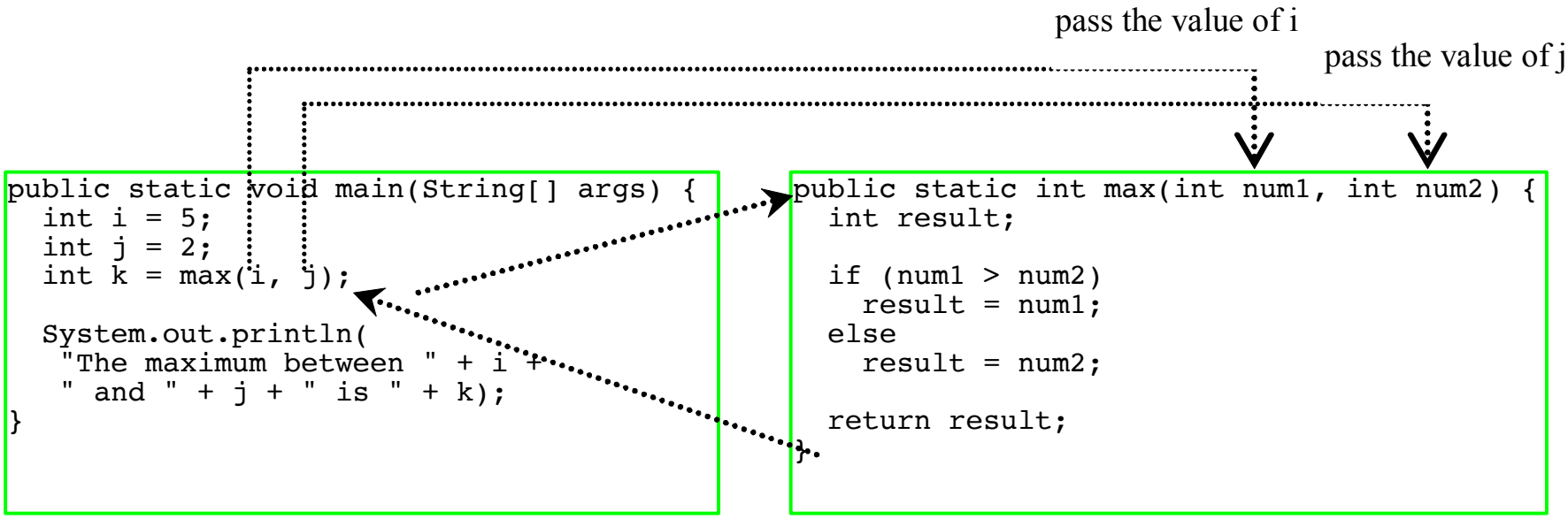
- A method may return a value
- The return value has a returnValueType, which is the data type of the value the method returns
- If the method does not return a value, the returnValueType is the keyword `void`
- For example: your **main method**

Return Value



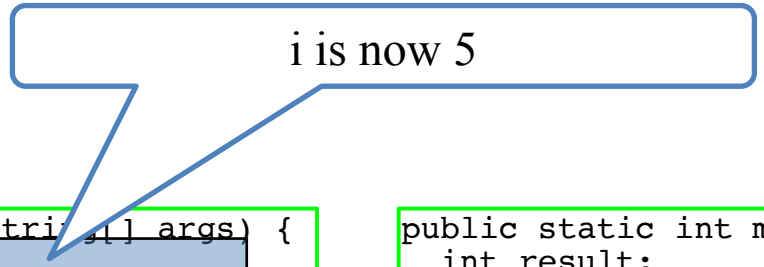
INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:



INFS1609/2609 Fundamentals of Business Programming

Tracing 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);
}
```

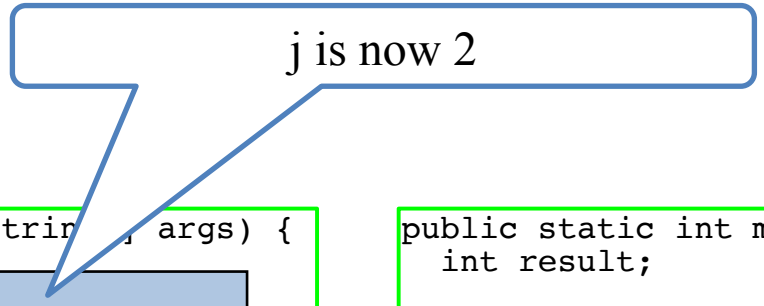
```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:




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

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

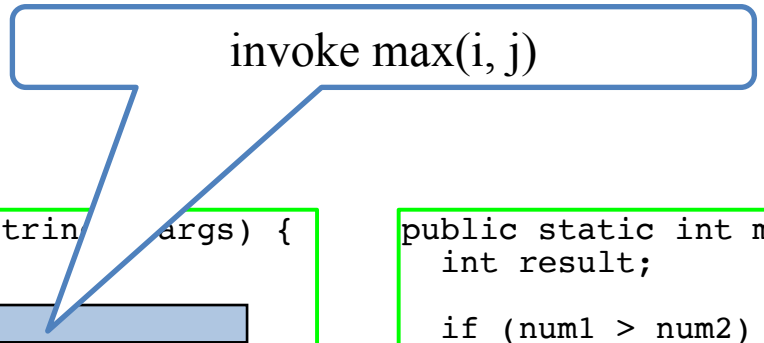
```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:




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

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

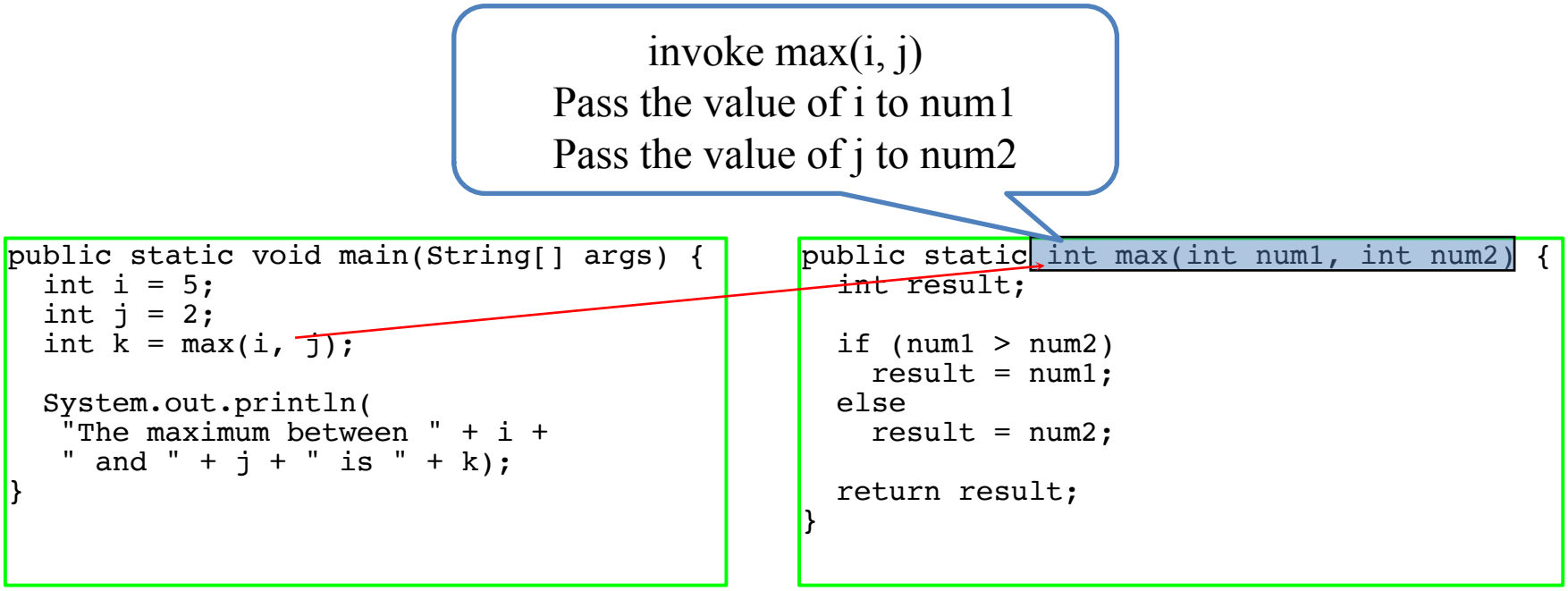
```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

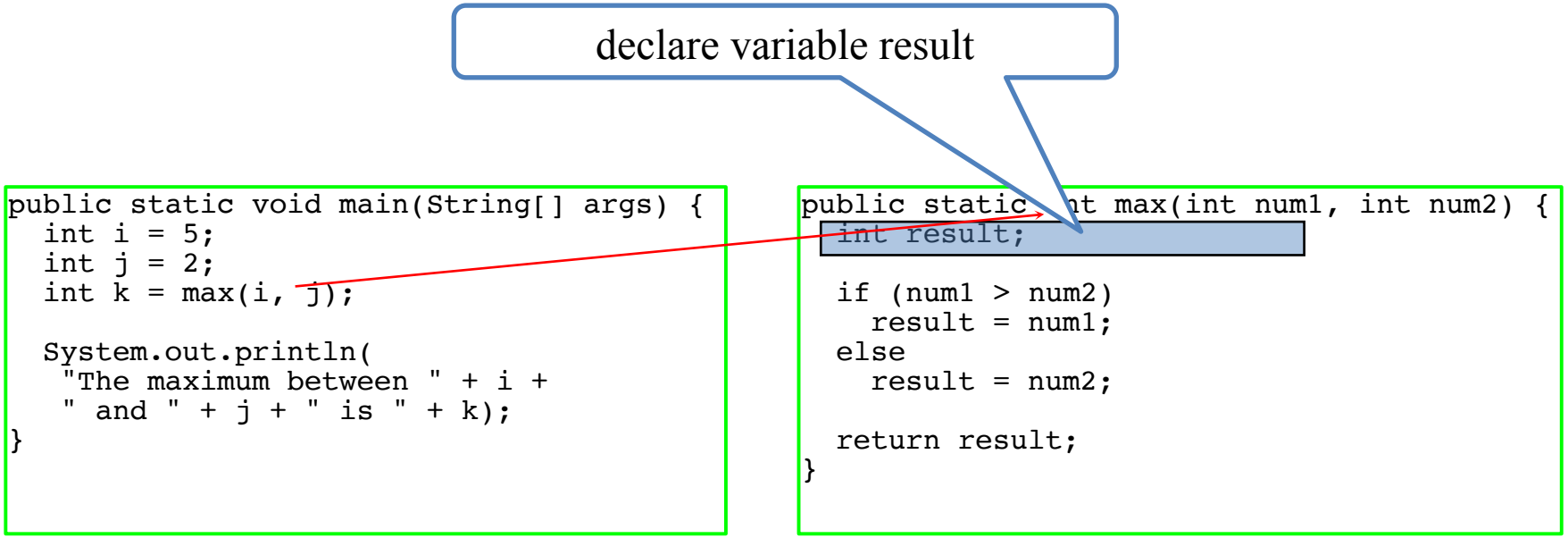
INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:



INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:



Tracing 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);
}
```

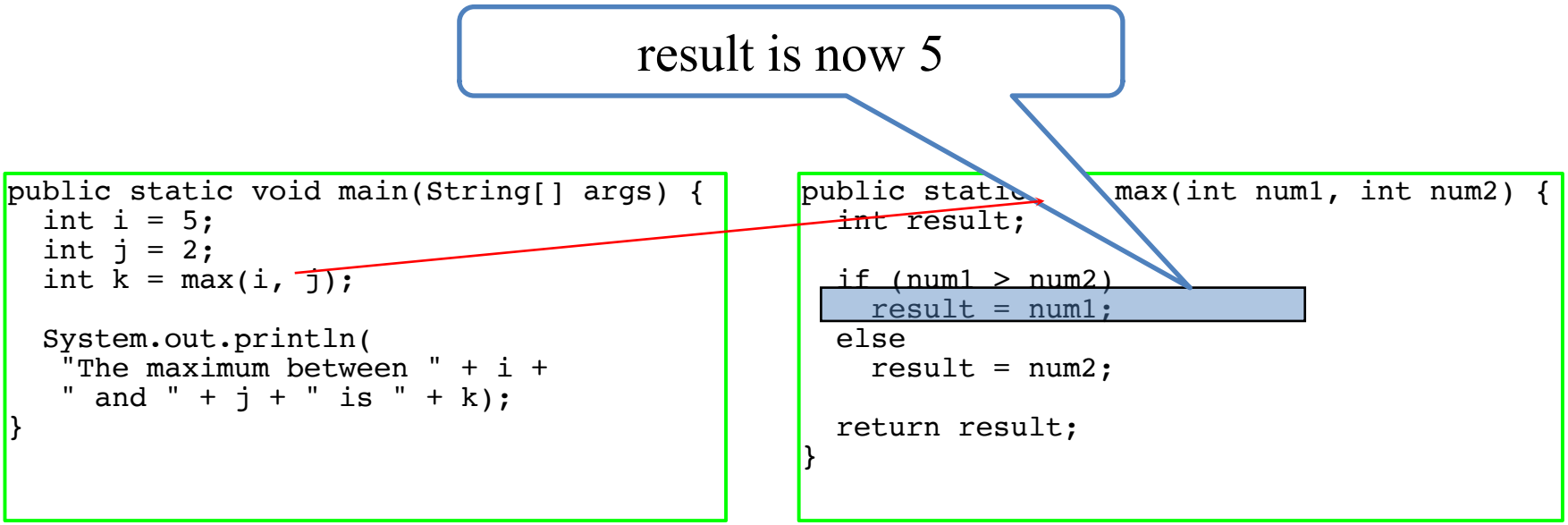
```
public static      max(int num1, int num2) {
  int result;

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

  return result;
}
```

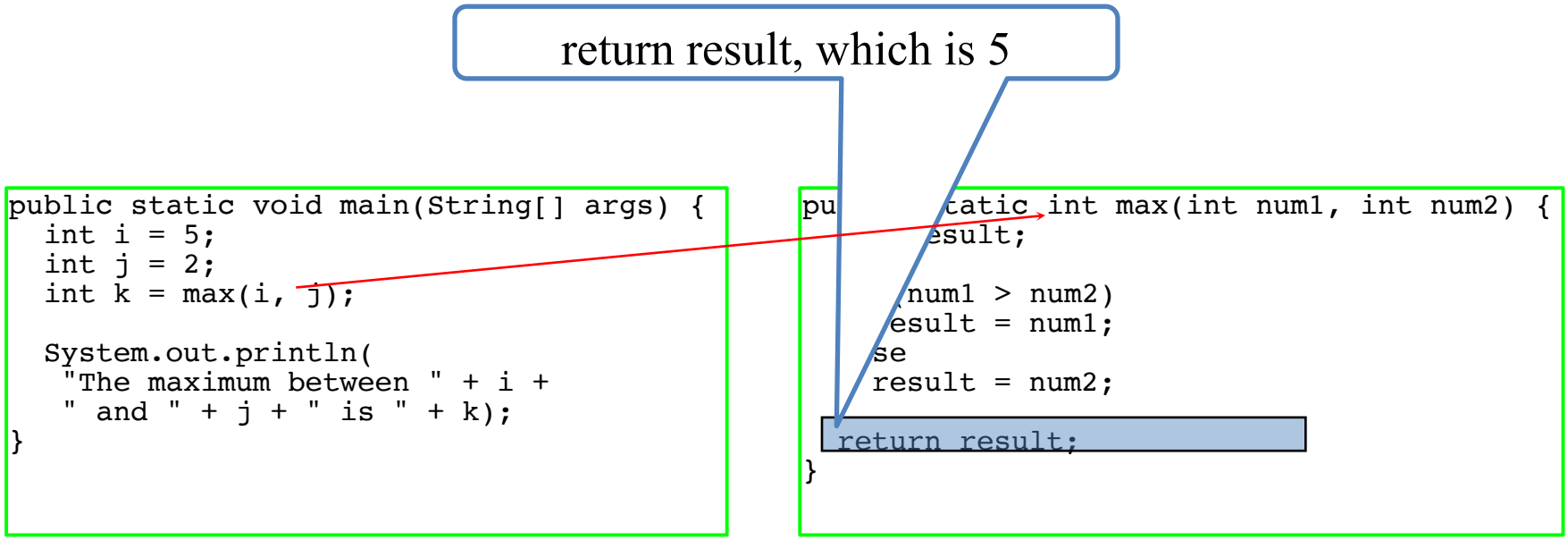
INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:



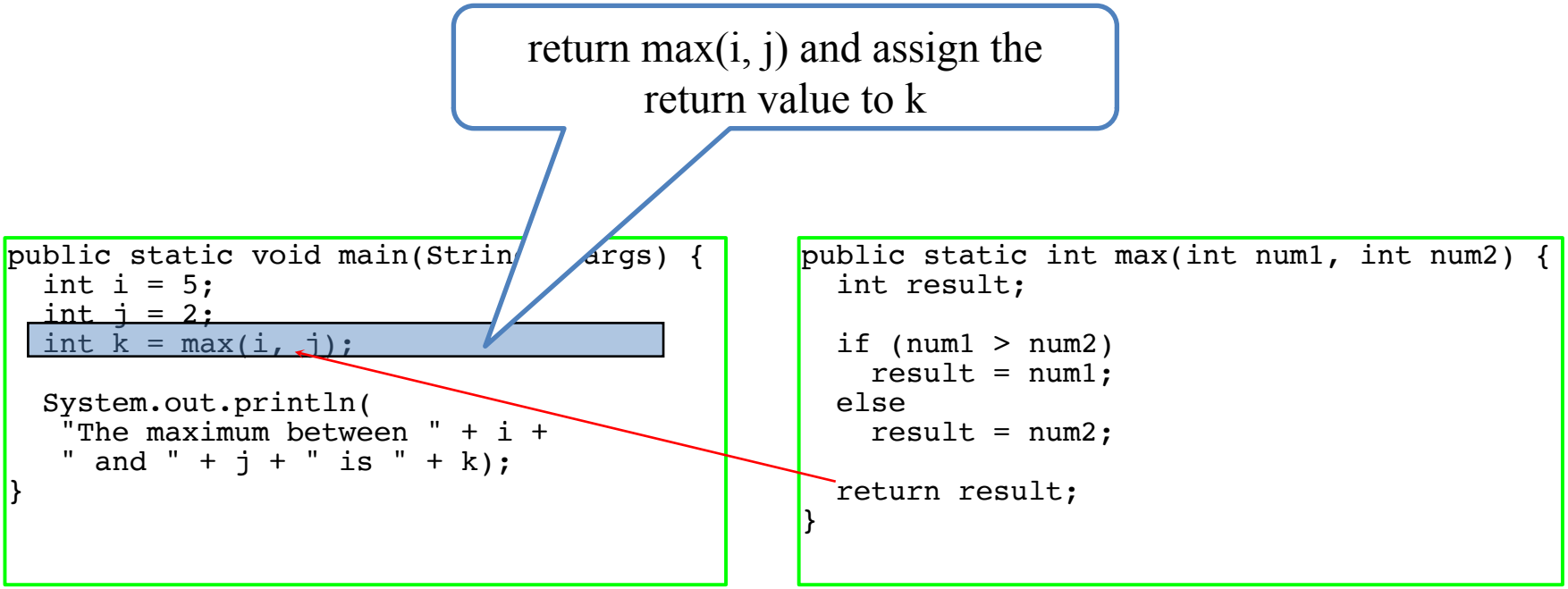
INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:



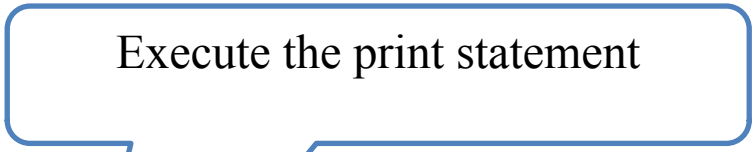
INFS1609/2609 Fundamentals of Business Programming

Tracing Method Invocation:



INFS1609/2609 Fundamentals of Business Programming

Tracing 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;
}
```

Return Value:

- 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 is 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.

A `void` method example

https://liveexample.pearsoncmg.com/html/TestVoidMethod.html

A method with a `char` return value

https://liveexample.pearsoncmg.com/html/TestReturnGradeMethod.html

Passing Parameters

Watch Supplementary Videos on Ed

Suppose we have a method:

```
public static void printMessage(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

1. What is the output if you invoke the method using:
   `printMessage("Welcome to Java", 5);`

2. What is the output if you invoke the method using:
   `printMessage("Why Java", 15);`

Passing Parameters

Watch Supplementary Videos on Ed

Suppose we have a method:

```
public static void printMessage(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

Question: can you invoke the method using the following?

```
printMessage(5, "Welcome to Java");
```

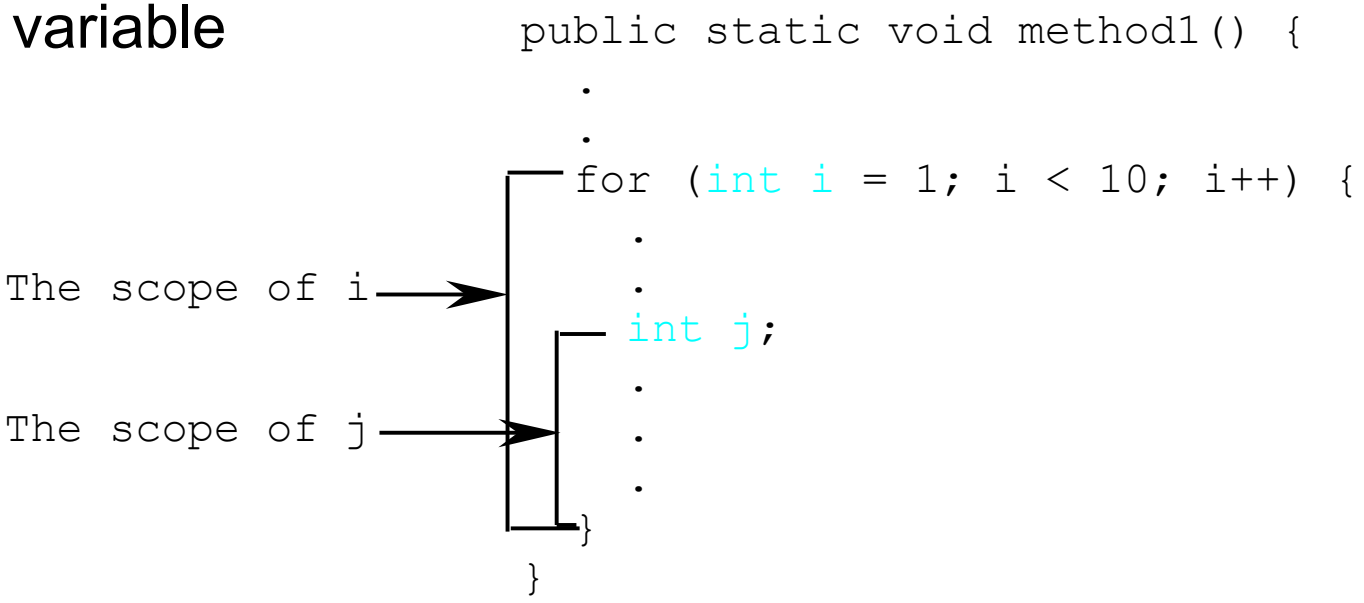
Pass by Values

Watch Supplementary Videos on Ed

- The power of a method is the ability to work with parameters
- When you invoke a method with an argument, the **value** of the argument is passed to the parameter
- Which means, **if your argument is a variable**, it is the **value** of the variable that is passed to the parameter
- Regardless of the changes made to the parameter inside the method, the variable is **not affected**

Local Variables and Scope:

- 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



Local Variables and Scope:

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;
  }

}
```

Overloading Methods:

Watch Supplementary Videos on Ed

- Define methods with the same name as long as their signatures are different
- The compiler determines which method to use based on the method signature

https://liveexample.pearsoncmg.com/html/TestMethodOverloading.html

Reflection

What have you learned today?

Thank you!