

CS1150

Principles of Computer Science

Methods

Yanyan Zhuang

Department of Computer Science

<http://www.cs.uccs.edu/~yzhuang>

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;
}
```

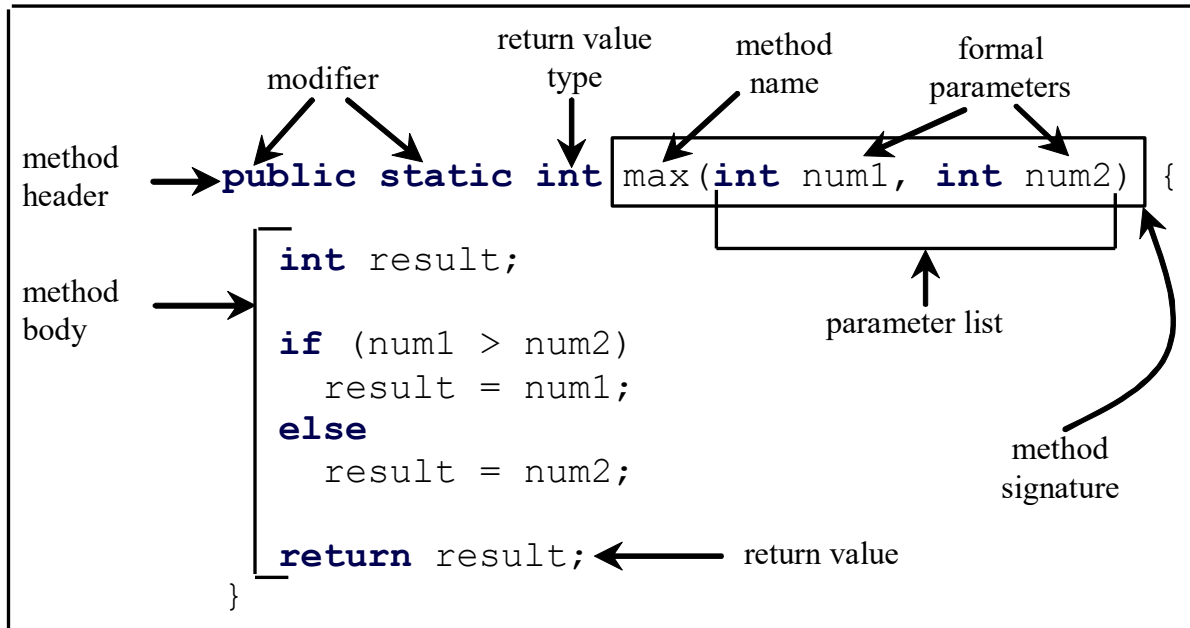
Invoke a method

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

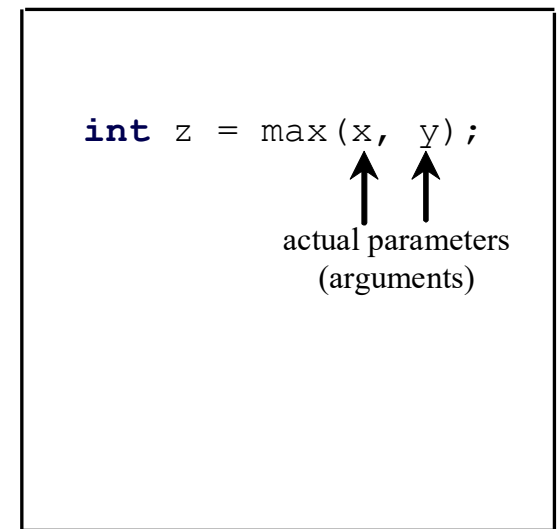
Defining Methods

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

Define a method



Invoke a method



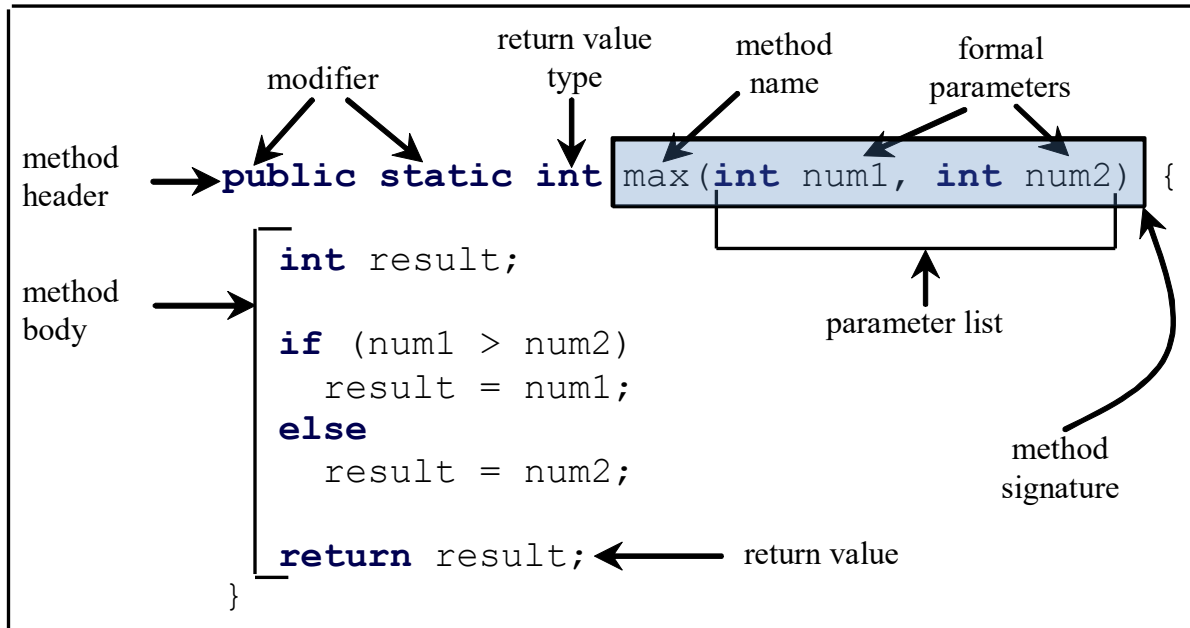
Method Signature

Method signature: the method name and the parameter list

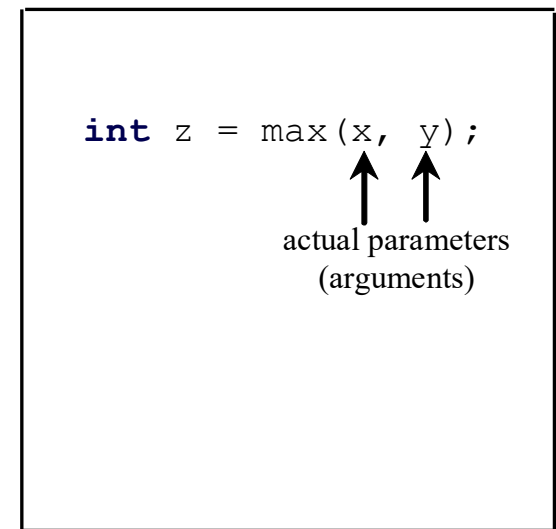
Method name: a programmer defined name

Parameter list: information that is coming into the method

Define a method



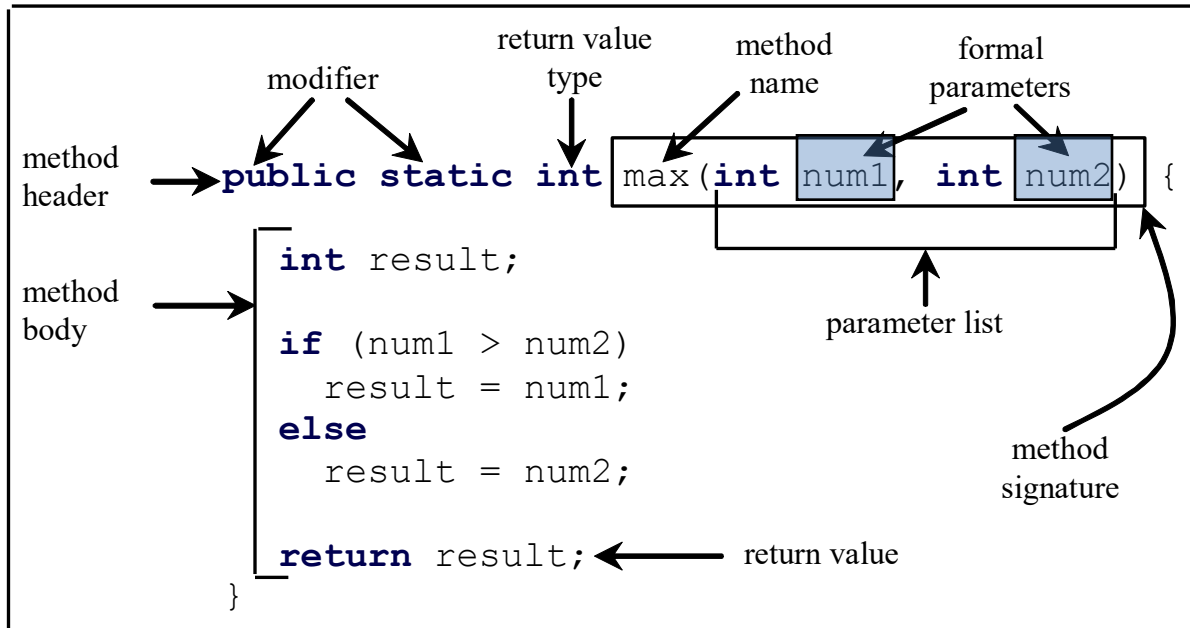
Invoke a method



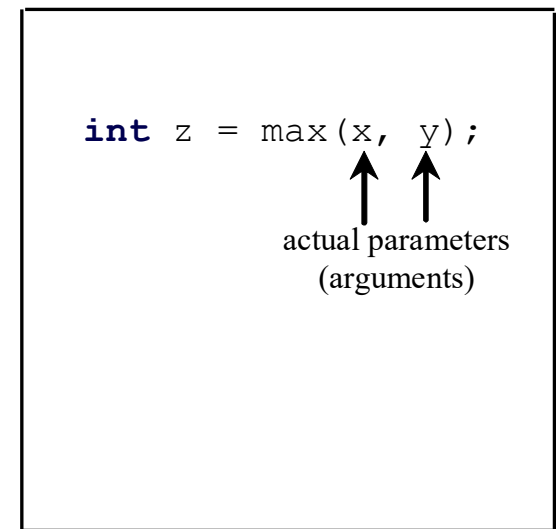
Formal Parameters

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

Define a method



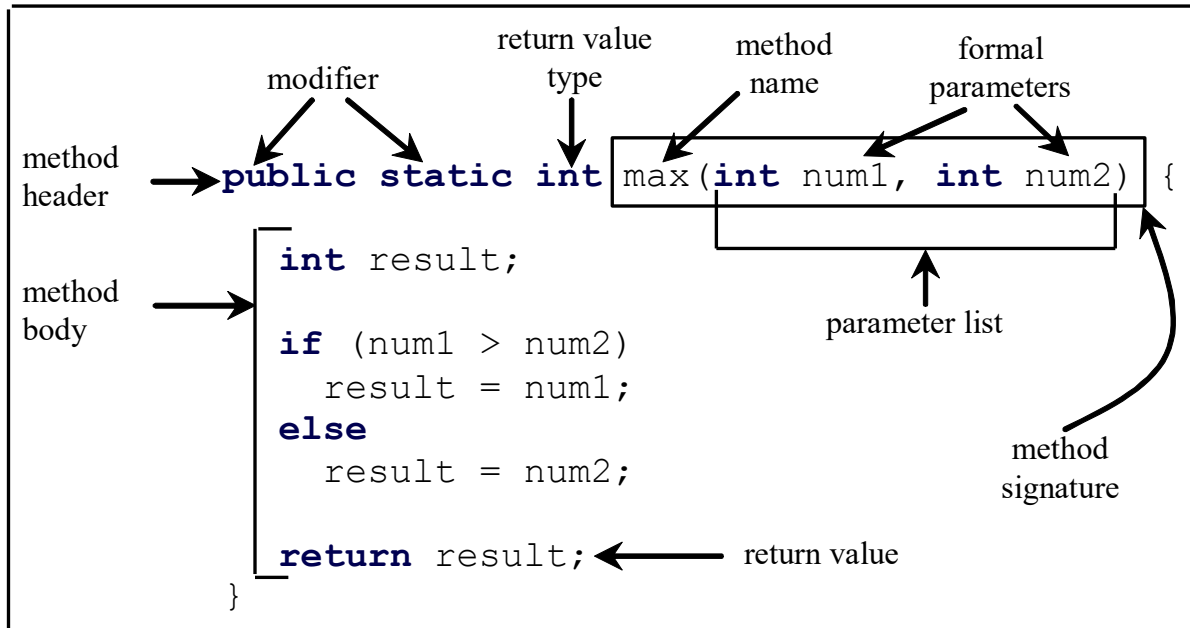
Invoke a method



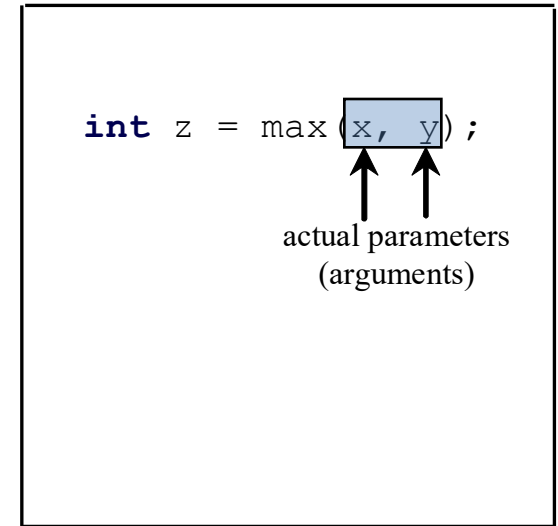
Actual Parameters

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

Define a method



Invoke a method



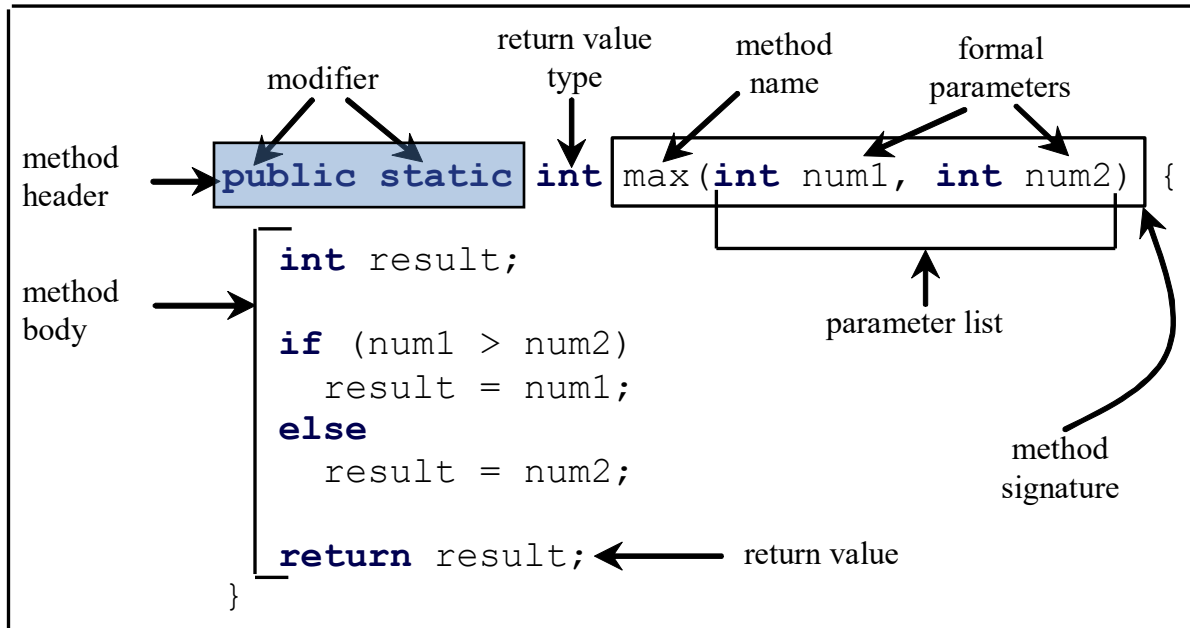
Modifiers

Both public and static are method's modifier (for now, all methods are public static)

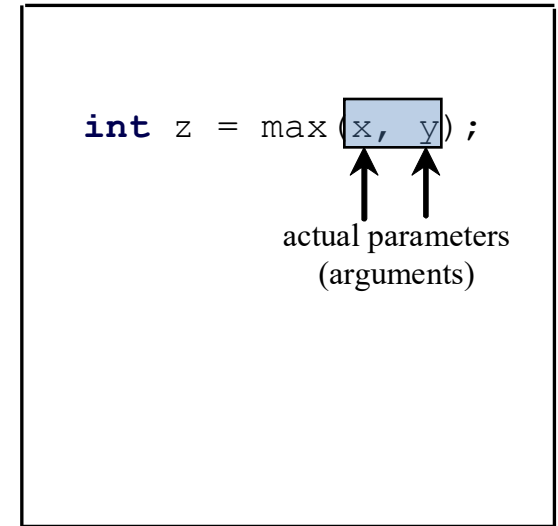
public: access to the method is public (visible to all other classes)

static: the method belong to the class, not an individual instance

Define a method



Invoke a method

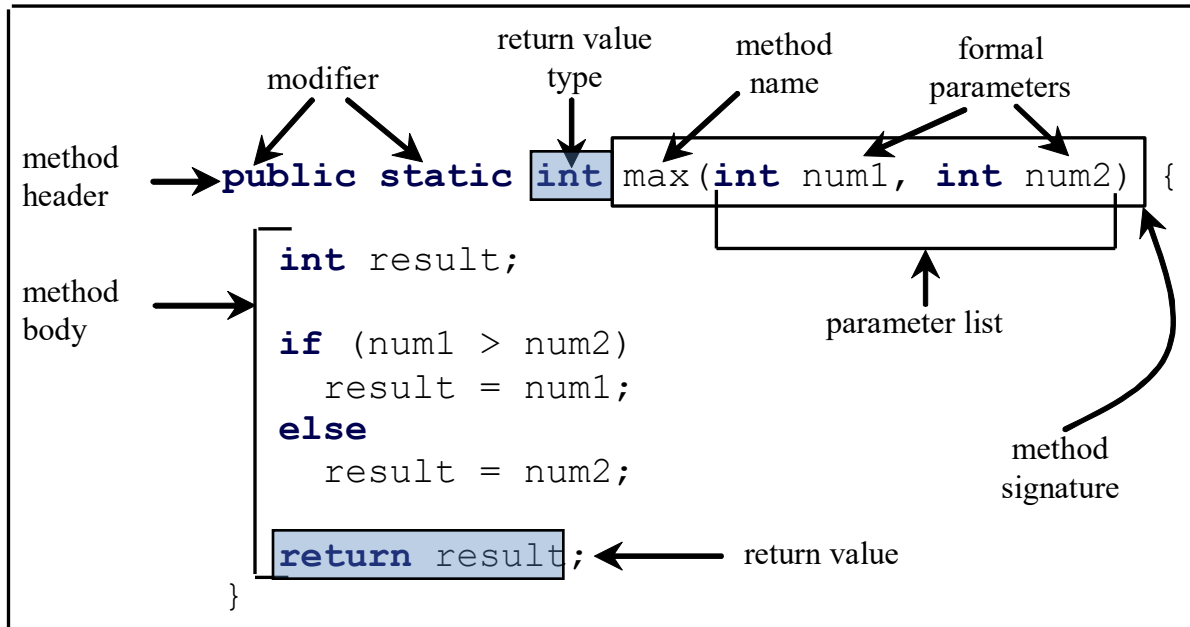


Return Value Type

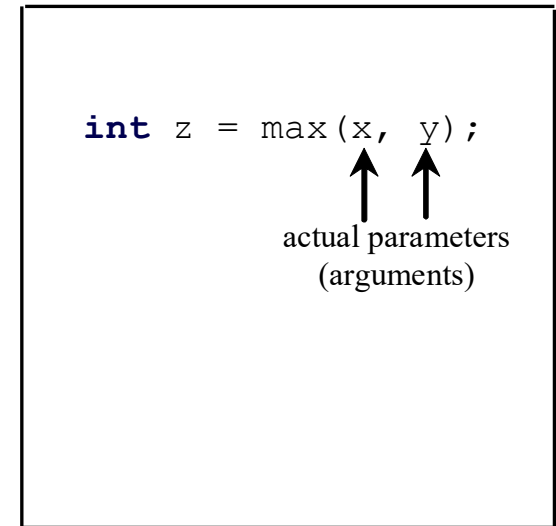
A method may return a value. The returnValueType is the data type of the value the method returns.

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

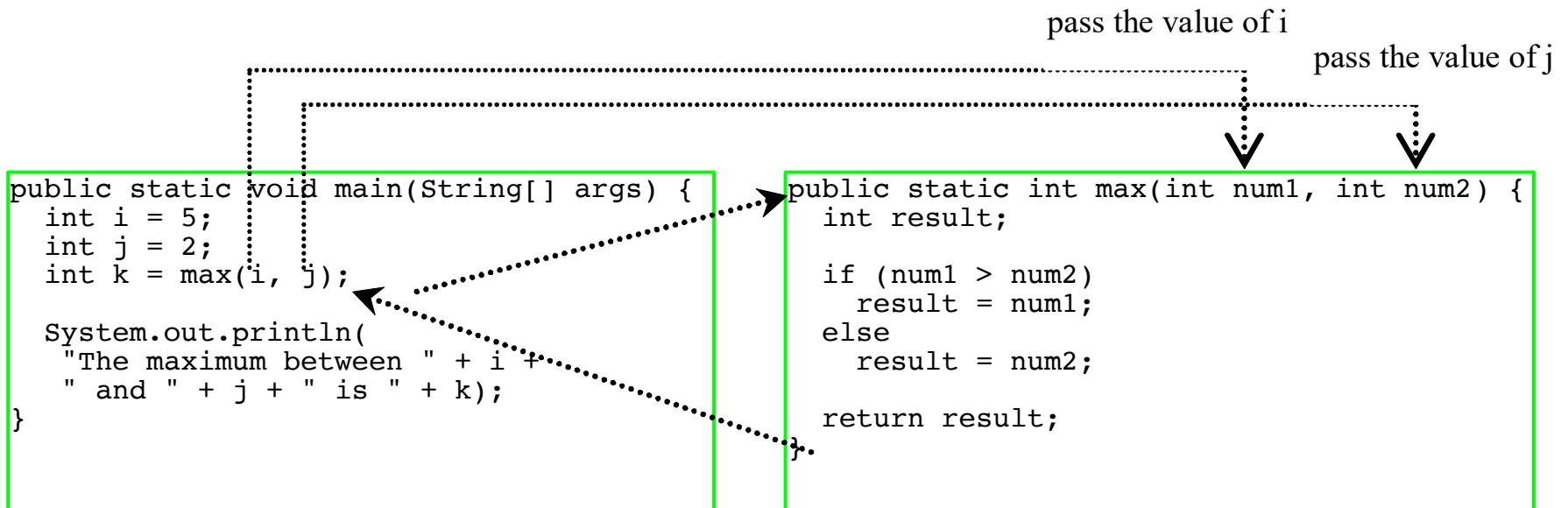


Rules for Methods

- A method may or may not return a value
- A method must declare a return type!
 - If a method returns a value
 - ▶ Return type is the data type of the value being returned
 - ▶ The **return** statement is used to return the value (O01FindingMax.java)
 - If a method does not return a value
 - ▶ Return type in this case is **void**
 - ▶ No return statement is needed (O02FindingMax.java)
- The values you pass in must match the **order and type** of the parameters declared in the method



Calling Methods



The main method is where the code starts

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)

```
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)
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;  
}
```

Trace Method Invocation

result 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

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

```
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

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;  
}
```

Example: O01FindingMax.java and O02FindingMax.java

CAUTION

A return statement is required for a value-returning method. The method shown below in (a) is logically correct, but it has an error because Java 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 Java will see a return statement to be reached regardless of how the if statement is evaluated.

Example: FindingSign.java

Value-Returning Methods

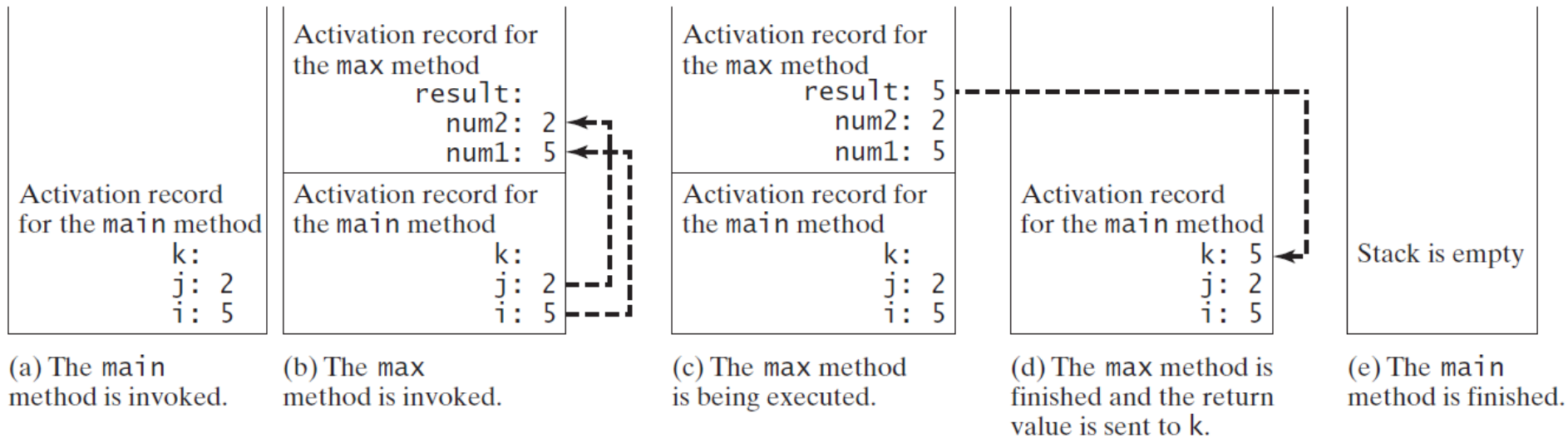
- Return a value to the caller
 - Call to a value returning method can be treated as a value or a statement returning a value
- SumIntegers.java example
 - The main method is where the code starts
 - sum is called: The value entered by the user is the actual parameter
 - ▶ This means the value stored in number1 and number2 is sent to the method
 - Control enters the sum method
 - ▶ Substitute actual parameter (i.e. number1 and number2) into the formal parameter (i.e. i1 and i2)
 - ▶ The return statement forces control to return to the caller (main) & sends to caller an integer value

Void Methods

- Does not return a value to the caller
 - The return type must be set to **void**
 - No return statement is used
 - Control returns to caller when the methods closing curly brace is reached
- Call to a void method must be made as a statement
 - See `SumIntegersVoid.java` example



Call Stacks



Call Stacks

- The call stack is used to keep track of **active** methods
- Stack is a **data structure**
 - Like....plate stacks at the buffet line: pop a plate off the stack, push a plate onto the stack
- Stacks are **last in, first out** (last item pushed onto stack is 1st item popped off)



Call Stacks

- When method is called (invoked)
 - Information for that method is pushed onto the stack
 - ▶ Values of parameters and variables
 - ▶ Current line of code executing
 - ▶ This information is called a "frame"
- When the method returns (completes)
 - Information for that method is popped off of the stack (frame is removed)
- Method on top of stack is the current executing method
- Method stays on top until return or the ending curly brace is reached



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;  
}
```



The main method
is invoked.

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.

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;
}
```

Space required for the
main method

k:
j: 2
i: 5

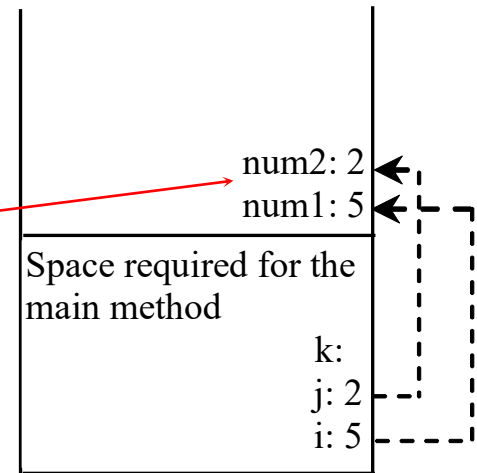
The main method
is invoked.

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;  
}
```



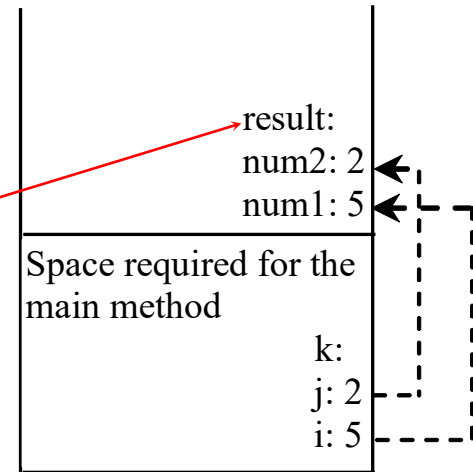
The max method is invoked.

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;  
}
```

Declare result



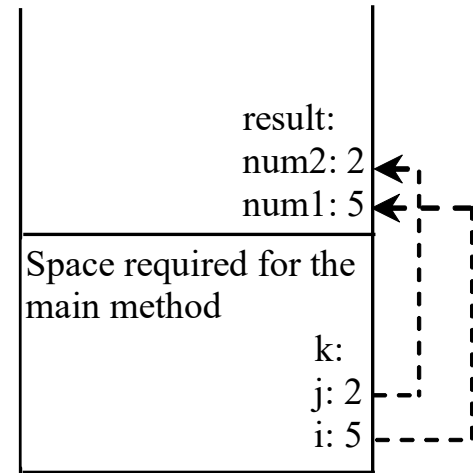
The max method is invoked.

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;  
}
```



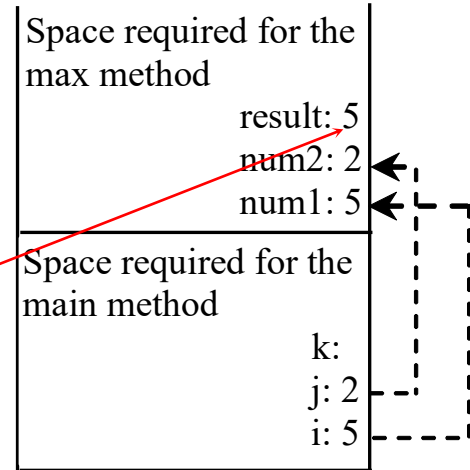
The max method is invoked.

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;  
}
```



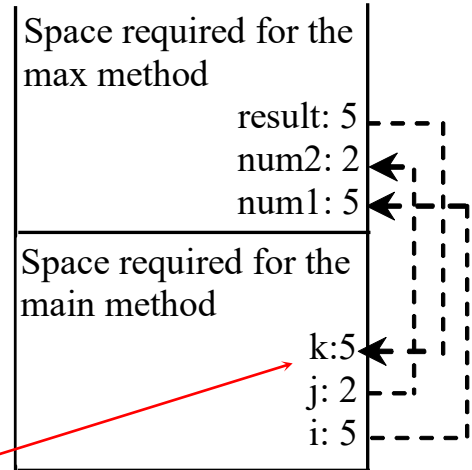
The max method is invoked.

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;  
}
```



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);  
}
```

Space required for the
main method

k:5
j:2
i:5

The main method
is invoked.

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