Introduction to Methods

Part 3: Behavior of a Method, Method Overloading, and More Method Examples

Course: CPSC 1150

Instructor: Dr. Bita Shadgar

Lecture 13

Learning Outcomes

- Define a void method
- Invoke a method
- Recognize parameters of arguments of a method
- Pass the parameters by value
- Apply methods for overloading

Void methods

Definition

Void methods are methods that do not return any information.

- Void methods just perform some task, i.e., printing
  - They do not return a value
- In the method header, put void in place of a return type

Example

```
public static void printNum(int n){
   System.out.print("n = " + n + ".");
}
```

Invoking a method

- To invoke a method. . .
  - Type the name of the method
  - Next type the argument list in parentheses
    - Must match the parameter list from the method header
- Invoking a method with a return value
  - The method should (usually) be invoked in a way that uses

Example

```
double cost = input.nextDouble();
System.out.println(Math.round(4.3));
```

- Invoking a void method
  - The method must be invoked as a standalone statement

Example

```
System.out.println("A void method");
```

Parameters vs. arguments

- **Parameters** are placeholders used in a method definition, in order to represent the inputs
  - Also known as formal parameters
- **Arguments** are the actual variables or literals that get passed in to a method when the method is invoked
  - Also known as actual parameters

Example

Consider the max method which finds the maximum of two integer inputs.

Method header: `public static int max(int num1, int num2)`

- num1 and num2 are parameters

Method invocation: `int z = max(x,y);`

- x and y are arguments

Header vs. signature vs. invocation

Method header

```
public static int max(int num1, int num2)
```

– Part of method definition
– Inside a class, but not inside another method
– Contains all information about a method (except what it does)

Method signature

```
max(int num1, int num2)
```

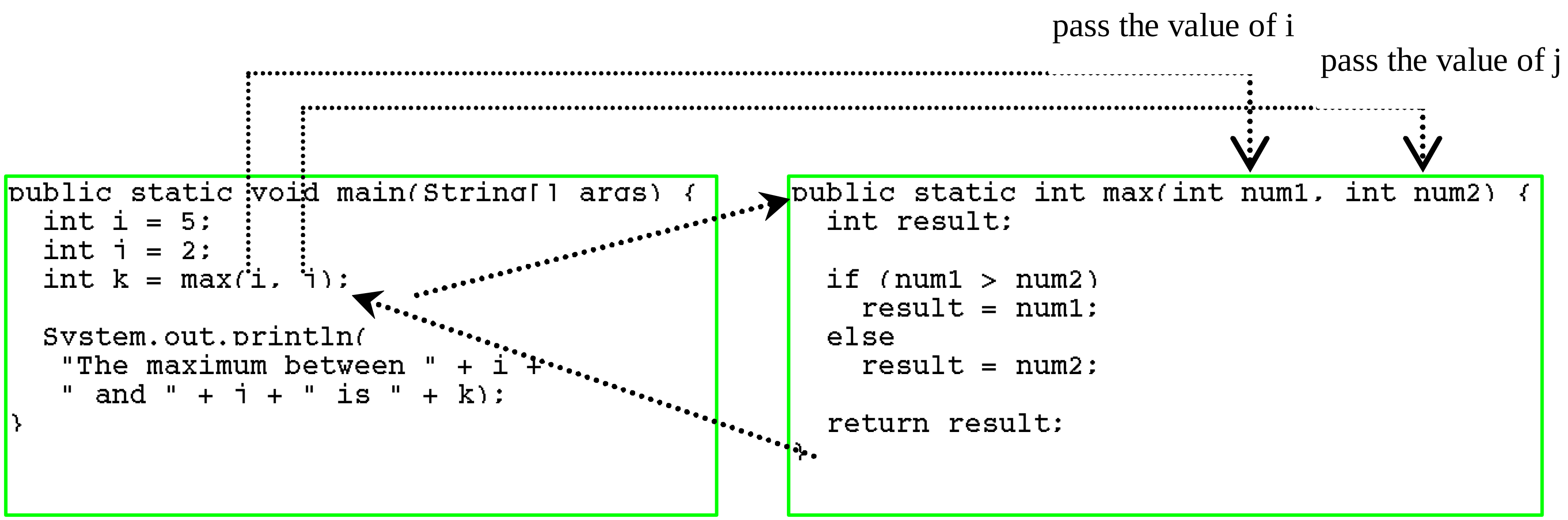
– Part of method header
– Demonstrates what types of variables to pass into a method

Method invocation

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

– This goes inside main or some other method

Calling Methods



```
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

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(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;
}
```

Trace Method Invocation

invoke max(i, j)

```
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;
}
```

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       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      max(int num1, int num2) {
  int result;

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

  return result;
}
```

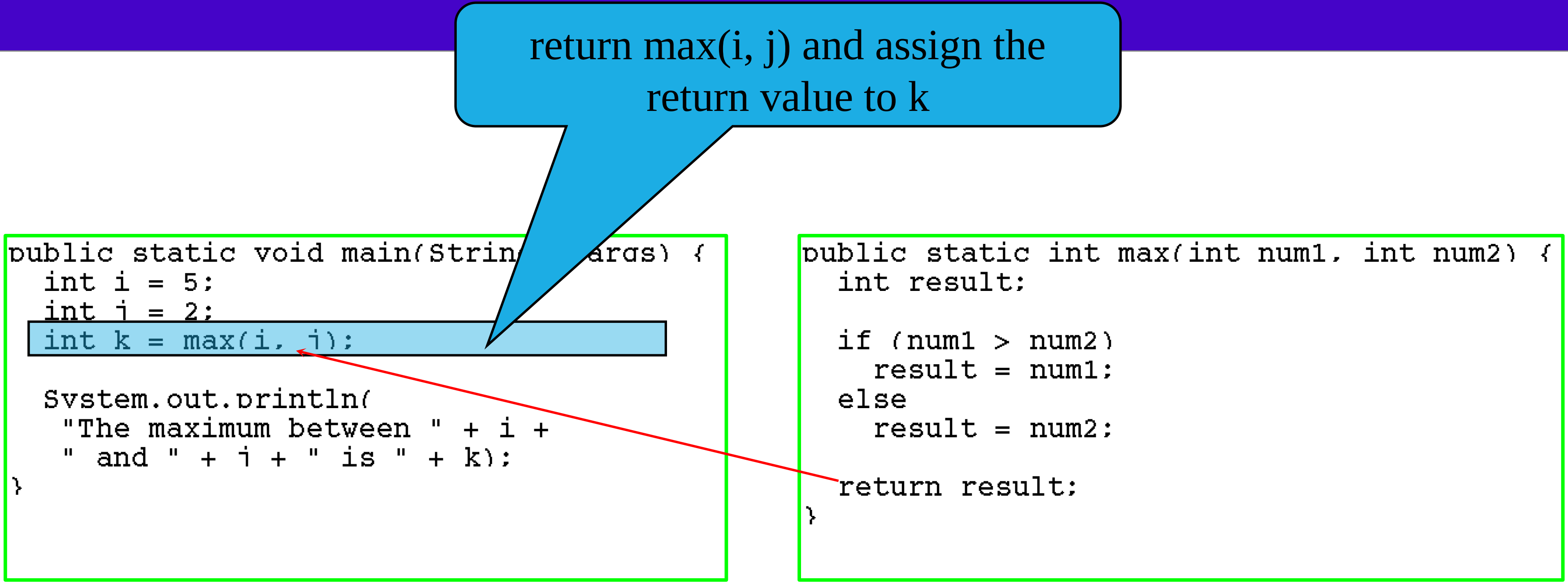
Trace Method Invocation

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);
}
```

```
pul      tatic int max(int num1, int num2) {
          esult;

            num1 > num2)
           esult = num1;
          se
         result = num2;

  return result;
}
```

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

Execute the print statement

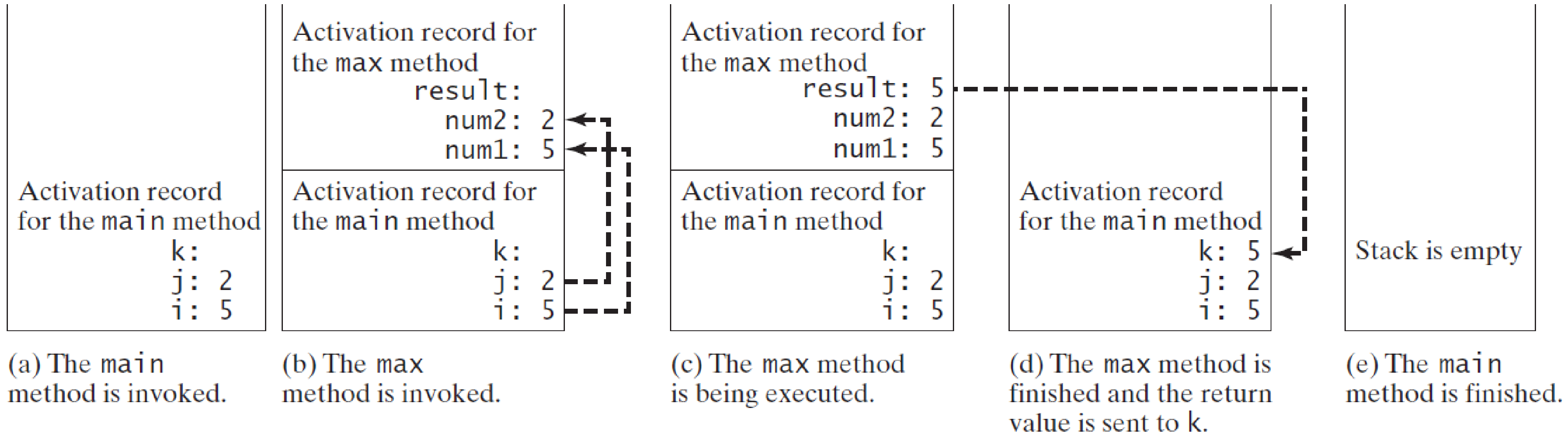
```
public static void main(String          s) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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 call stack

- The **call stack** (or stack) stores the values of variables from different methods when control is passed between the methods
- When a method terminates, all variables from that method are deleted from the stack
- The stack operates in a last-in, first-out (LIFO) manner
- To understand what the stack is, an example is necessary

Call Stacks

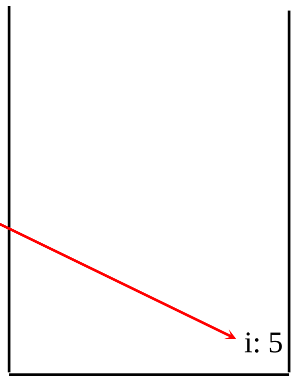

Activation record
for the main method
k:
j: 2
i: 5
(a) The main
method is invoked.
Activation record for
the max method
result:
num2: 2
num1: 5
Activation record for
the main method
k:
j: 2
i: 5
(b) The max
method is invoked.
Activation record for
the max method
result: 5
num2: 2
num1: 5
Activation record for
the main method
k:
j: 2
i: 5
(c) The max method
is being executed.
Activation record
for the main method
k: 5
j: 2
i: 5
(d) The max method is
finished and the return
value is sent to k.
Stack is empty
(e) The main
method is finished.

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;
}
```



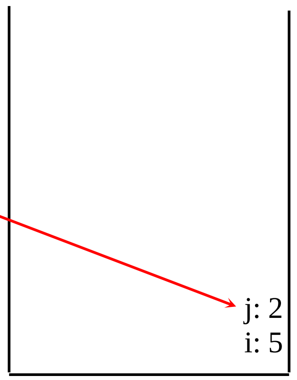
The main method is invoked.

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;
}
```



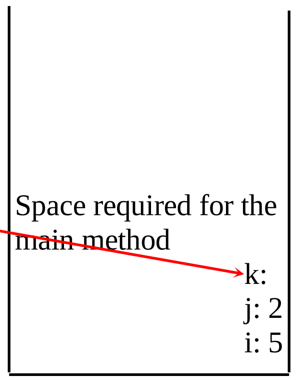
The main method is invoked.

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;
}
```



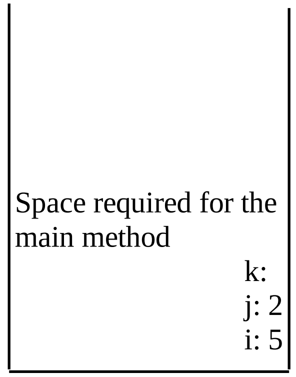
The main method
is invoked.

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;
}
```



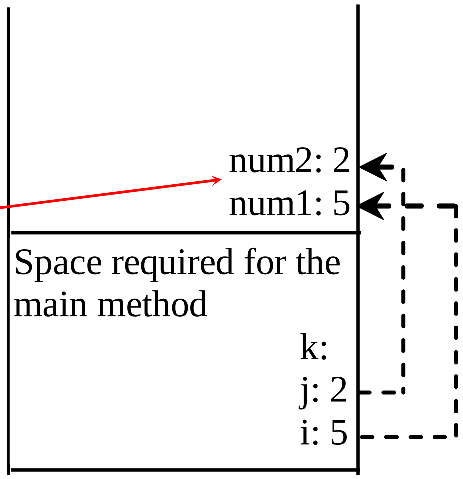
The main method is invoked.

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;
}
```



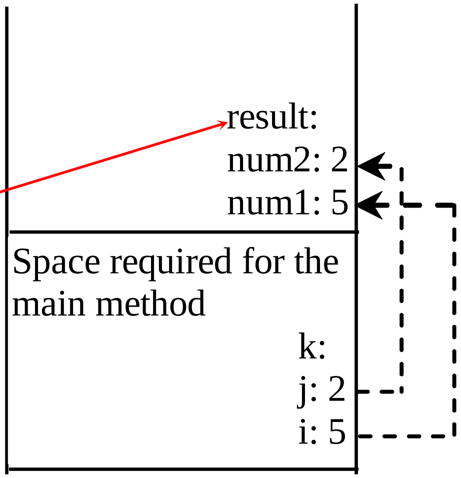
The max method is invoked.

Trace Call Stack

Declare 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;
}
```



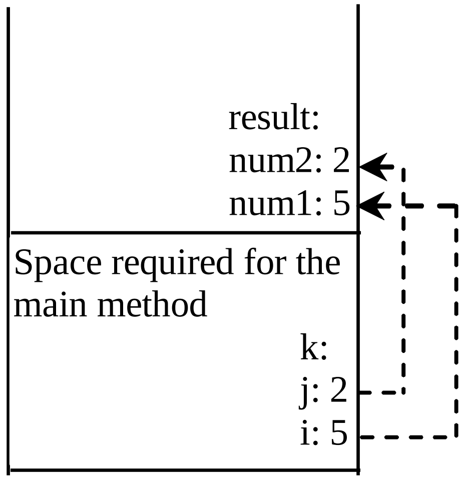
The max method is invoked.

Trace Call Stack

(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;
}
```



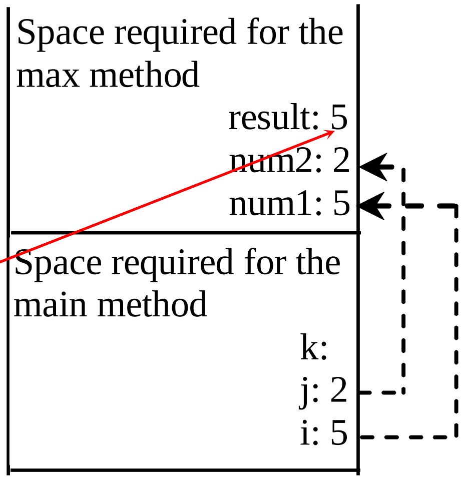
The max method is invoked.

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;
}
```


Space required for the max method
result: 5
num2: 2
num1: 5
Space required for the main method
k:
j: 2
i: 5

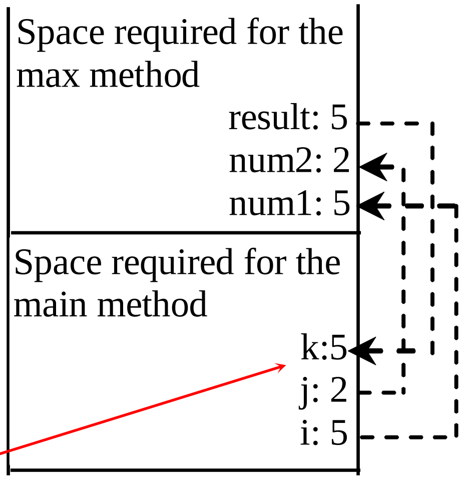
The max method is invoked.

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;
}
```



Space required for the max method
result: 5
num2: 2
num1: 5
Space required for the main method
k:5
j: 2
i: 5

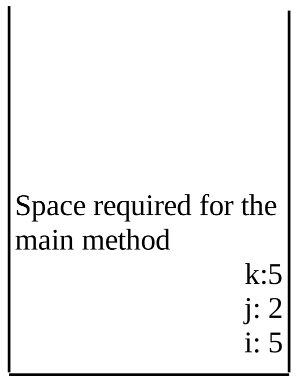
The max method is invoked.

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);
}
```

```
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.

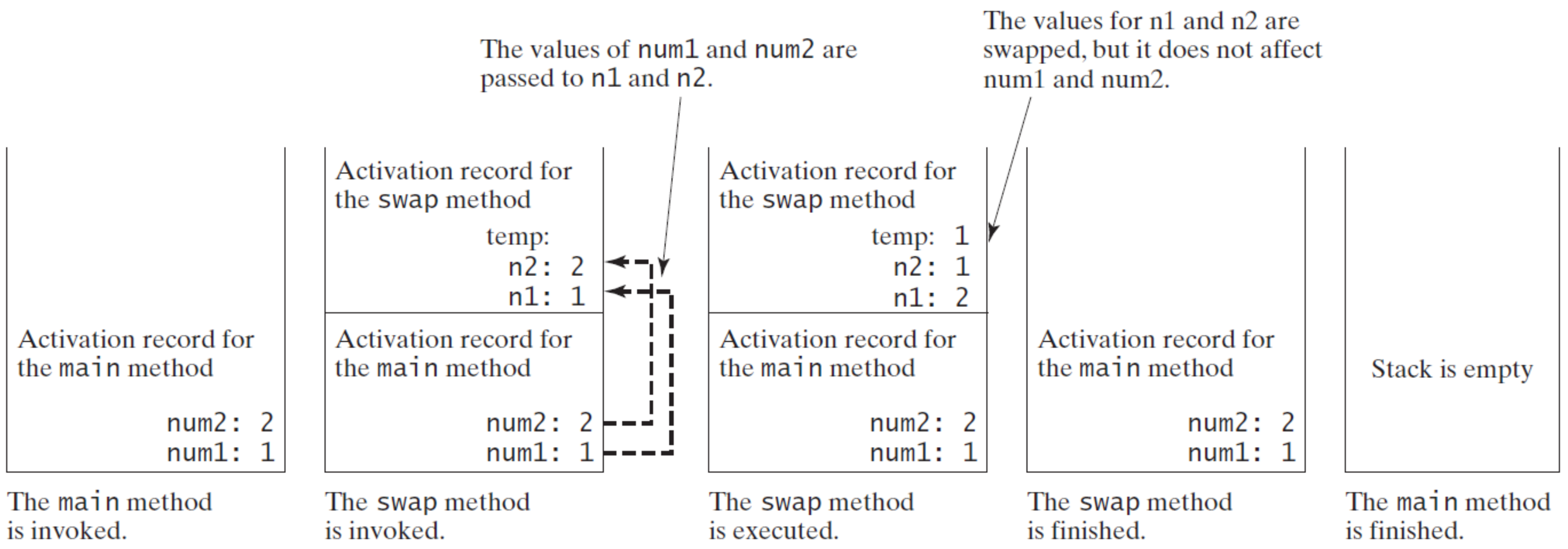
Pass by value

- Variables themselves cannot be altered by methods when they are used as arguments
- Instead, the arguments’ values get passed to the method’s parameters
- If you alter a parameter inside a method. . .
  - The parameter will be affected
  - The associated argument (variable) will not
- In this sense, variables are local to their methods

Example

Let’s look at the `TestPassByValue` program from the textbook to understand this.

Pass by Value, cont.


The values of num1 and num2 are passed to n1 and n2.
The values for n1 and n2 are swapped, but it does not affect num1 and num2.
Activation record for the main method
num2: 2
num1: 1
The main method is invoked.
Activation record for the swap method
temp:
n2: 2
n1: 1
Activation record for the main method
num2: 2
num1: 1
The swap method is invoked.
Activation record for the swap method
temp: 1
n2: 1
n1: 2
Activation record for the main method
num2: 2
num1: 1
The swap method is executed.
Activation record for the main method
num2: 2
num1: 1
The swap method is finished.
Stack is empty
The main method is finished.

CAUTION

```
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 $\longrightarrow$

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

(b)

- Return a value for every cases in the program.
  - 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.

Same task, different types?

- Let’s say we want to make a method called max which finds the maximum of two numbers
- What type should we choose for the input?
- Let’s say we make it an integer
  - Let’s see what happens in the max program.
- We want to be able to accept either an integer or a floating-point type. . .

Method overloading

Definition

Method **overloading** is when there are several method definitions with the same name but with different parameter lists.

- Parameter lists are considered different when they have. . .
  - Different types
  - A different number of parameters
- **Warning:** Just having different parameter names or a different return type is **not enough** for overloading
  - The compiler will think you are trying to define the same method twice

When to use overloading

- Overloading should be used when:
  - You want to have an optional argument
  - You want to define similar methods with different input types
- Overloading should not be used for:
  - Methods that solve noticeably different problems

Example

Let’s look back at our max method and overload it, using the program OverloadMax.

Invoking an overloaded method

- Question: How does the compiler pick which definition to use for an overloaded method?
- Answer: If there is only one parameter list that matches, it picks that one. If there is more than one, it picks the best match.
- Question: What if there are two equally good matches?
- Answer: That’s called **ambiguous invocation** and it causes a compile error

More Practice

Example

TwinPrime to print all the twin prime pairs up to 1000.

Two prime number are twin, if their difference is 2, e.g. (3,5), (5,7), and (11, 13) are twins

Example

Circle to ask the user for radius of a circle, test if it is valid radius and compute its perimeter and area.

Example

Hex2Dec to ask the user for a Hex number and convert it to decimal number.