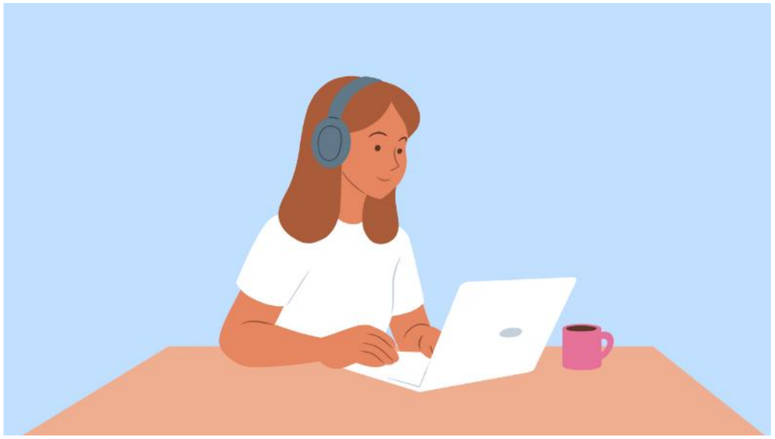
Chapter 5: Methods

TAO Yida

taoyd@sustech.edu.cn

So Far…



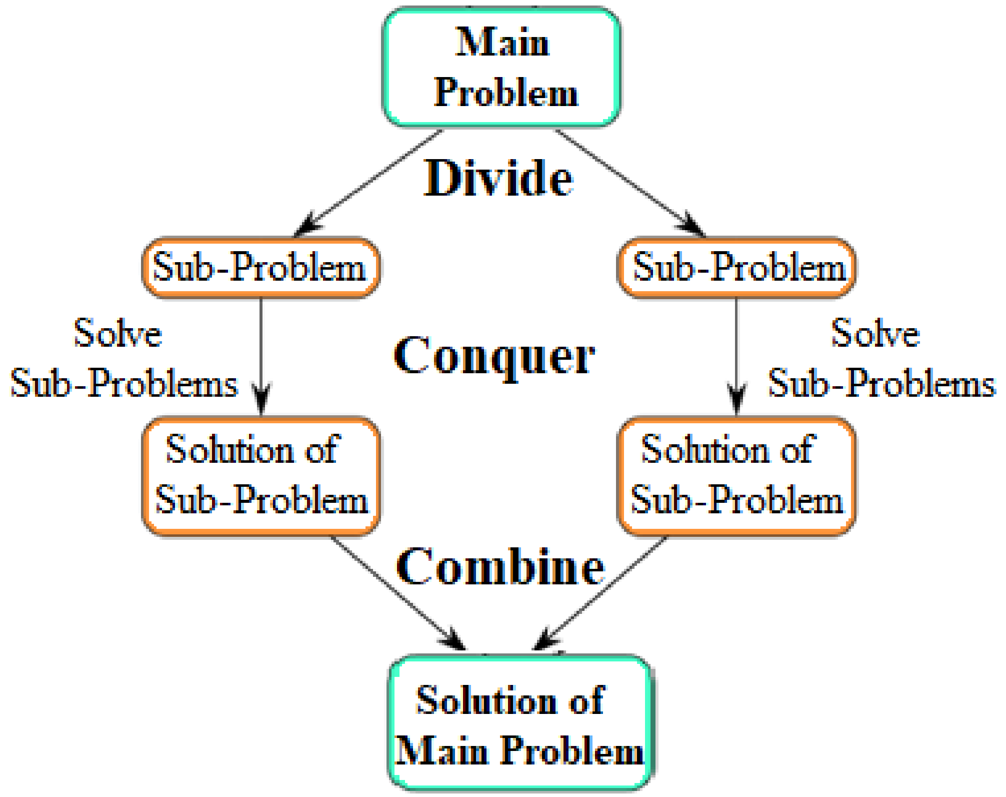
- The programs we have written so far solve simple problems
- They are short and everything fits well in a `main` method

What if you are asked to **solve complex problems?**

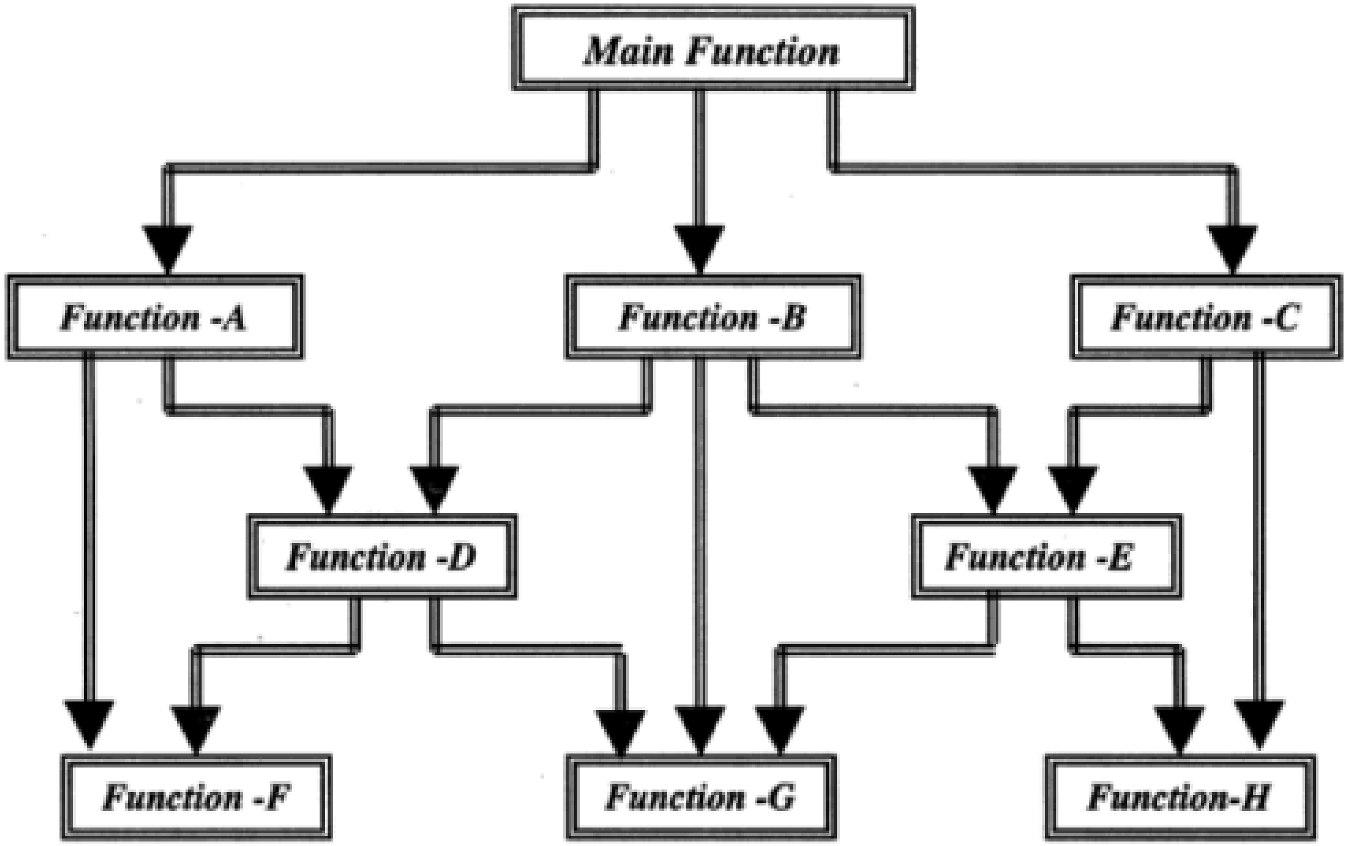
Write a giant `main` method?

Divide and Conquer (分而治之)

Decompose a big/complex task into smaller one and solve each of them



Divide and Conquer (分而治之)



Methods we’ve used so far?

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Command-line arguments

Using Methods

```
public class MaximumFinder {

    public static double maximum(double x, double y, double z) {
        double max = x;
        if(y > max) max = y;
        if(z > max) max = z;
        return max;
    }

    public static void main(String[] args) {
        double result1 = maximum(3.1,  3.2,  3.0);
        double result2 = maximum(70,   90,   10);
        double result3 = maximum(45, 10.1,  1);
        System.out.printf("%.1f %.1f %.1f", result1, result2, result3);
    }
}
```

The class defines two methods

Find the largest of 3 double values

Declaring a Method

修饰符 形式参数（形参）

Modifiers + Return type + Method name + Parameters

```
public static double maximum( double x, double y, double z ) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

Method body contains one or more statements that perform the method’s task

Return Type of Methods

Return type: the type of data the method returns to its caller (e.g., main method).

```
public static double maximum(double x, double y, double z) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

A method may return

- Nothing (void)
- Primitive values (e.g., an integer)
- References to objects, arrays

```
void println()

int nextInt()
```

Return Type of Methods

Return type: the type of data the method returns to its caller (e.g., main method).

```
public static double maximum(double x, double y, double z) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

We can also `return` an expression:

```
return number1 + number2 + number3;
```

Method Parameters

A comma-separated list of parameters, meaning that the method requires additional information from the caller to perform its task.

```
public static double maximum( double x, double y, double z ) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

A method can have 0, 1, or more parameters

Empty parentheses (0 parameter): the method does not need additional information to perform its task

Method Parameters

Each parameter must specify a type and an identifier

```
public static double maximum( double x, double y, double z ) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

[Scope] A method’s parameters are local variables of that method and can be used only in that method’s body

Calling a Method

- In a method definition, you define what the method does.
- To execute the method, you have to call or invoke (调用) it

Arguments (实际参数/实参)

```
double result = maximum(1.0, 2.0, 3.0);
```

$$\begin{tikzcd} 1.0 \arrow[d] & 2.0 \arrow[d] & 3.0 \arrow[d] \\ x & y & z \end{tikzcd}$$

```
public static double maximum( double x, double y, double z ) {
    …
}
```

The number, order and type of arguments (实参) and parameters (形参) must be consistent.

Calling a Method

- In a method definition, you define what the method does.
- To execute the method, you have to call or invoke (调用) it

Arguments (实际参数/实参)

```
double result = maximum(1.0, 2.0, 3);
```

```
public static double maximum( double x, double y, double z ) {
    …
}
```

Argument promotion: `maximum()` expects to receives a `double` argument, but it is ok to pass an int 3. Java converts the `int` value 3 to the `double` value 3.0

Calling a Method

- Before any method can be called, its arguments must be evaluated to determine their values
- If an argument is a method call, the method call must be performed to determine its return value

```
System.out.println(maximum(1.0, 2.0, 3.0));

Math.pow( Math.pow(x2-x1, 2) + Math.pow(y2-y1, 2) , 0.5 );
```

Objectives

- Method declaration and invocation
- Passing arguments
- Method overloading
- Method call stack
- Command-line arguments

Passing Arguments in Method Calls

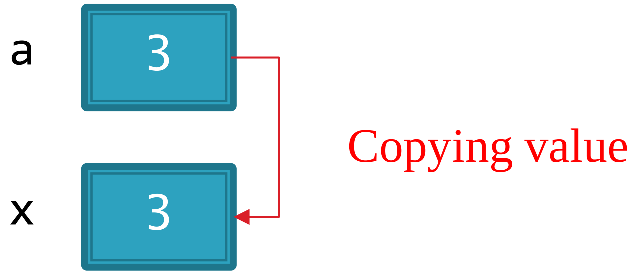
- In Java, all arguments are passed by value (按值传递), meaning that the method gets a copy of all parameter values
- A method call can pass two types of values to the called method:
  - Primitive types: passing copies of primitive values
  - Reference types: passing copies of references to objects.

Passing Primitive Type

```
public static void main(String[] args) {
    int a = 3;
    System.out.println("Before: " + a);

    triple(a);
    System.out.println("After: " + a);
}
```

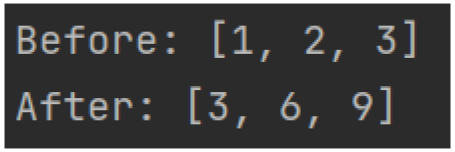
```
Before: 3
After: 3
```

```
public static void triple(int x) {
    x *= 3;
}
```



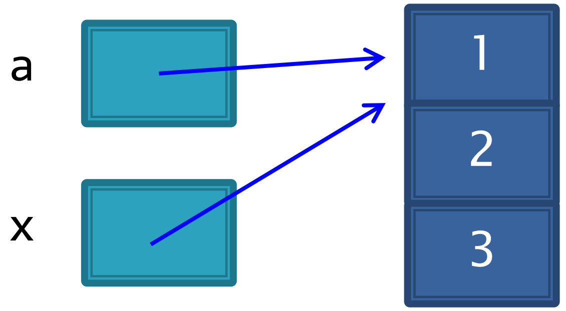
x becomes 9 after method call
a remains unchanged

Passing Reference Type



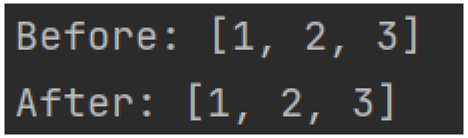
```
Before: [1, 2, 3]
After: [3, 6, 9]
```

```java
public static void main(String[] args) {
    int[] a = {1, 2, 3};
    System.out.println("Before: " + Arrays.toString(a));

    triple(a);
    System.out.println("After: " + Arrays.toString(a));
}

public static void triple(int[] x) {
    for(int i = 0; i < x.length; i++)
        x[i] *= 3;
}
```



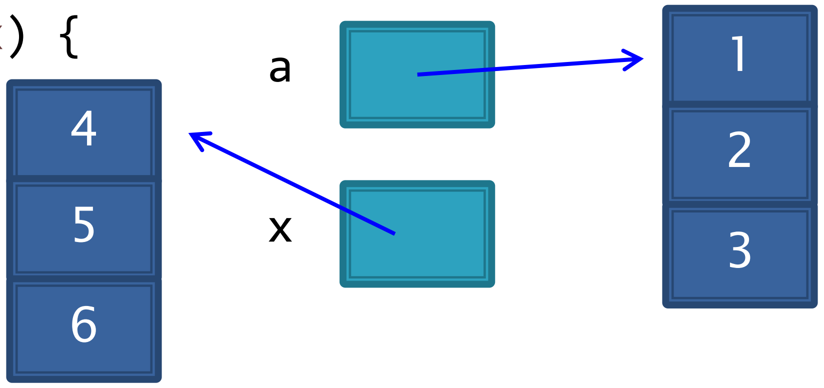
Copying value again. Difference is that the value is a memory address.

Passing Reference Type



```
Before: [1, 2, 3]
After: [1, 2, 3]
```

```
public static void main(String[] args) {
    int[] a = {1, 2, 3};
    System.out.println("Before: " + Arrays.toString(a));

    triple(a);
    System.out.println("After: " + Arrays.toString(a));
}

public static void triple(int[] x) {
    x = new int[]{4,5,6};
}
```



x refers to a new array

Q1: What is arr?

```
public static void main(String[] args) {
    int[] arr = {1, 2, 3};
    triple(arr[0]);

    // what is arr now?
}

public static void triple(int x) {
    x *= 3;
}
```

Q2: What is arr?

```
public static void main(String[] args) {
    int[][] arr = {{1, 2, 3}, {4, 5}};

    triple(arr[0], 2);
    triple(arr[1], 1);

    // what is arr now?

}

public static void triple(int[] x, int i) {
    x[i] = x[i-1] * 3;

}
```

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Command-line arguments

What’s the control flow of method invocations?

Define/invoke max method

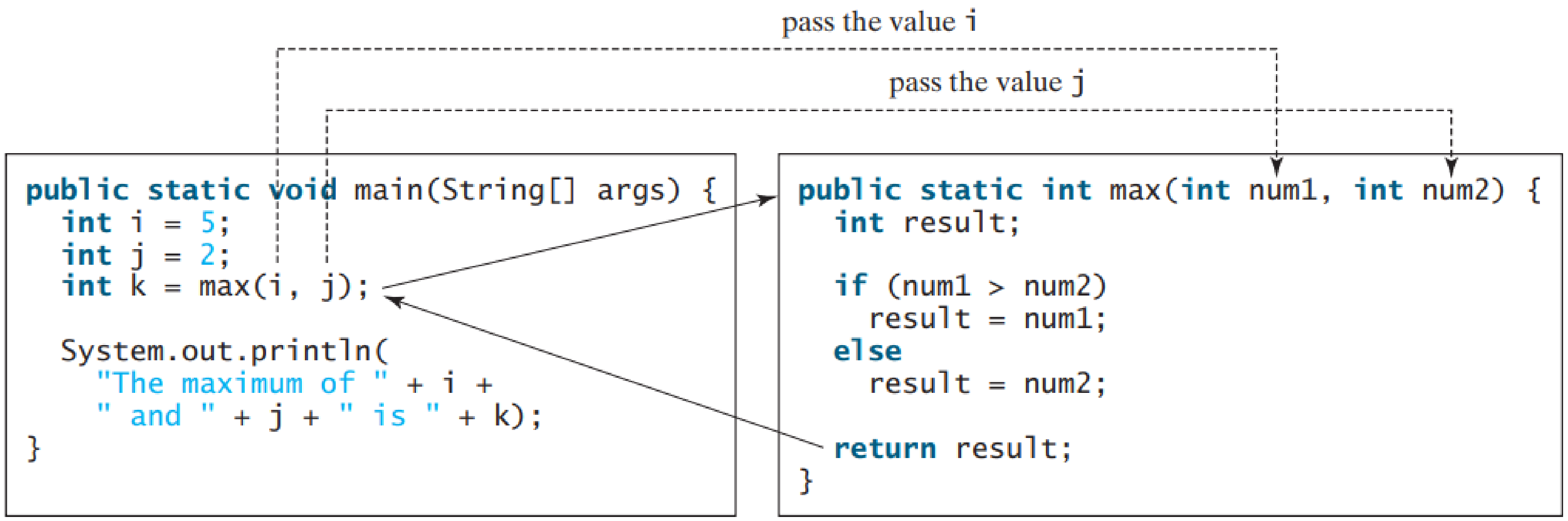
main method

invoke max

define method

```java
public class TestMax {
  /** Main method */
  public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum of " + i +
      " and " + j + " is " + k);
  }

  /** Return the max of two numbers */
  public static int max(int num1, int num2) {
    int result;

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

    return result;
  }
}
```

Control flow for method calls



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

  System.out.println(
    "The maximum of " + 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;
}
```

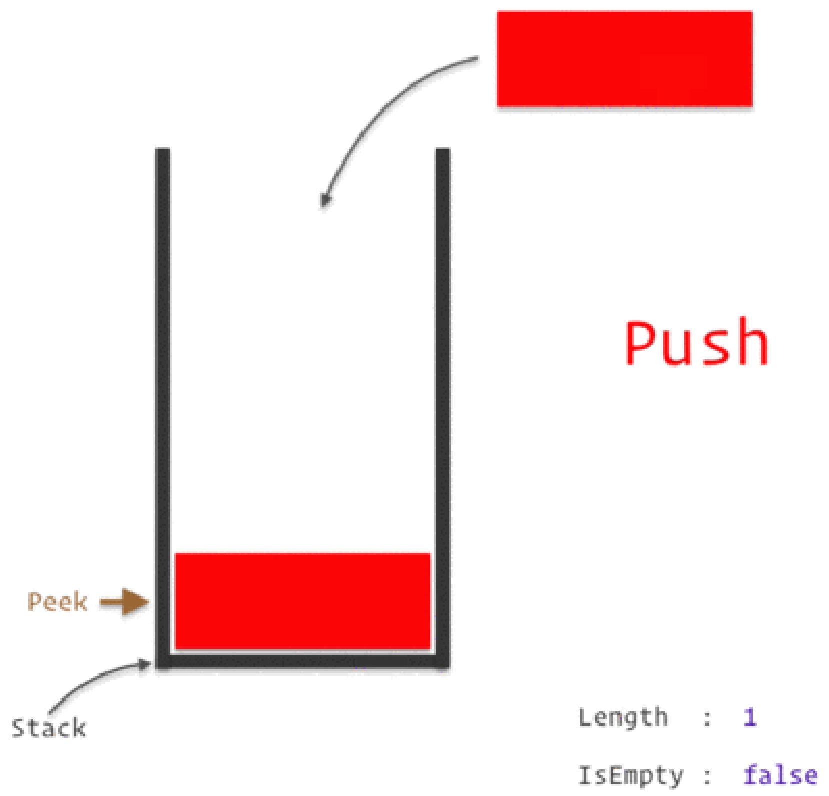
FIGURE 6.2 When the max method is invoked, the flow of control transfers to it. Once the max method is finished, it returns control back to the caller.

Method-Call Stack

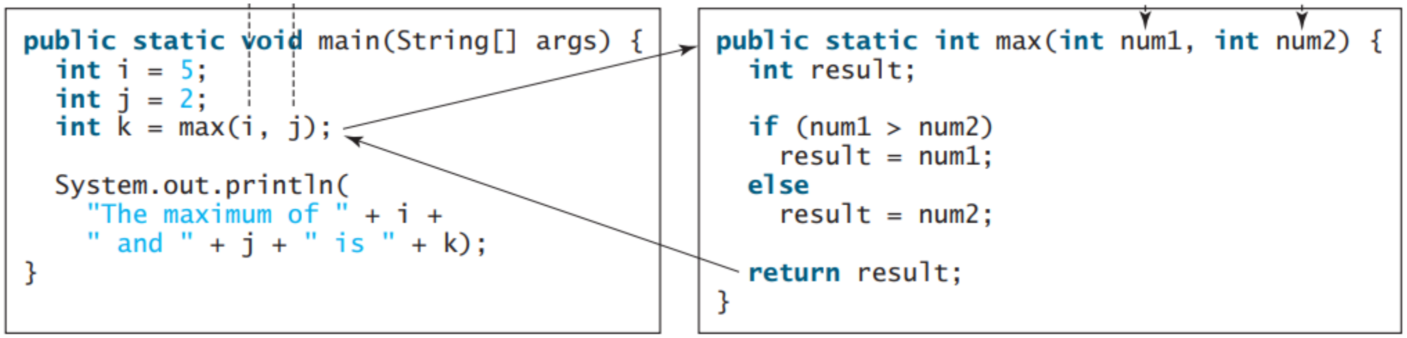
- Each time a method is invoked, the system creates an activation record (激活记录) that stores parameters and variables for the method and places the activation record in an area of memory known as a call stack (方法调用栈)

Method-Call Stack

- When a method calls another method, the caller's activation record is kept intact, and a new activation record is created for the new method called and pushed to the stack.
- When a method finishes its work and returns to its caller, its activation record is popped from the call stack.
- A call stack stores the activation records in a **last-in, first-out** fashion: The activation record for the method that is invoked last is removed first from the stack.

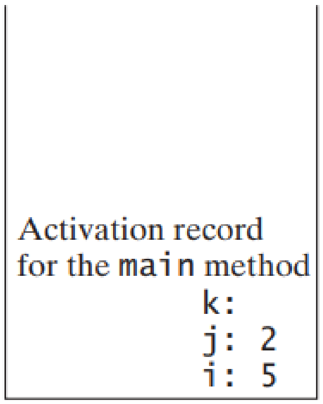


Method-Call Stack



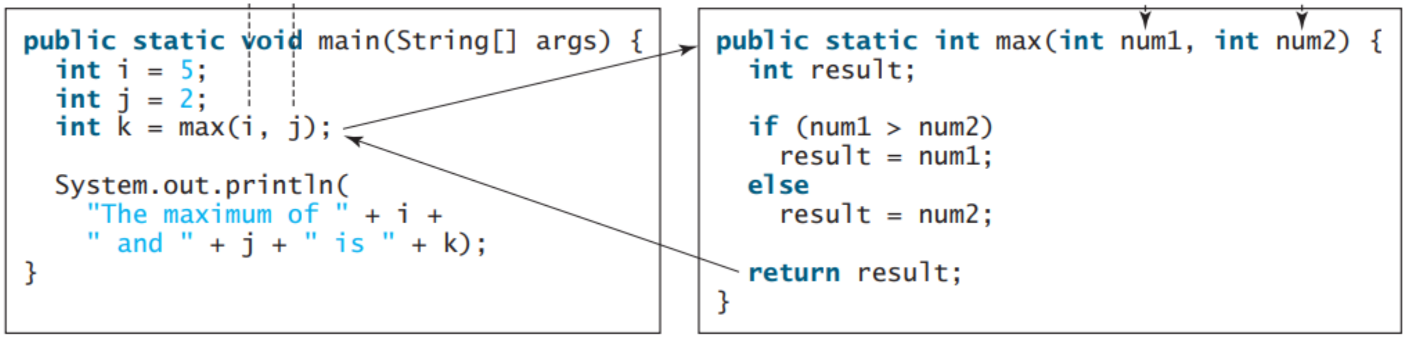
```
public static void main(String[] args) {
   int i = 5;
   int j = 2;
   int k = max(i, j);

   System.out.println(
      "The maximum of " + 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;
}
```



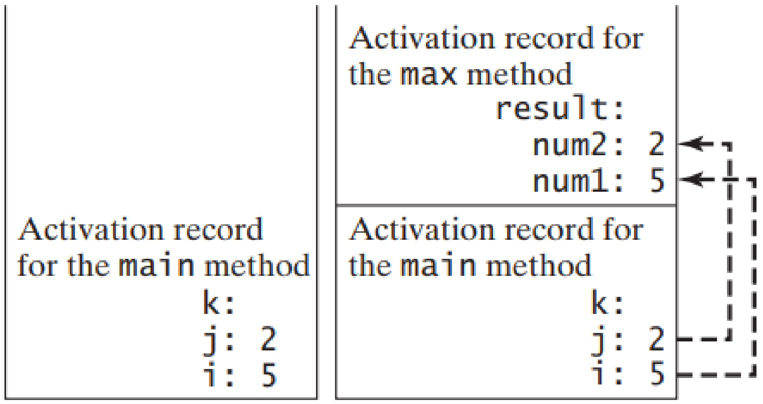
(a) The `main` method is invoked.

Method-Call Stack



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

  System.out.println(
    "The maximum of " + 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;
}
```

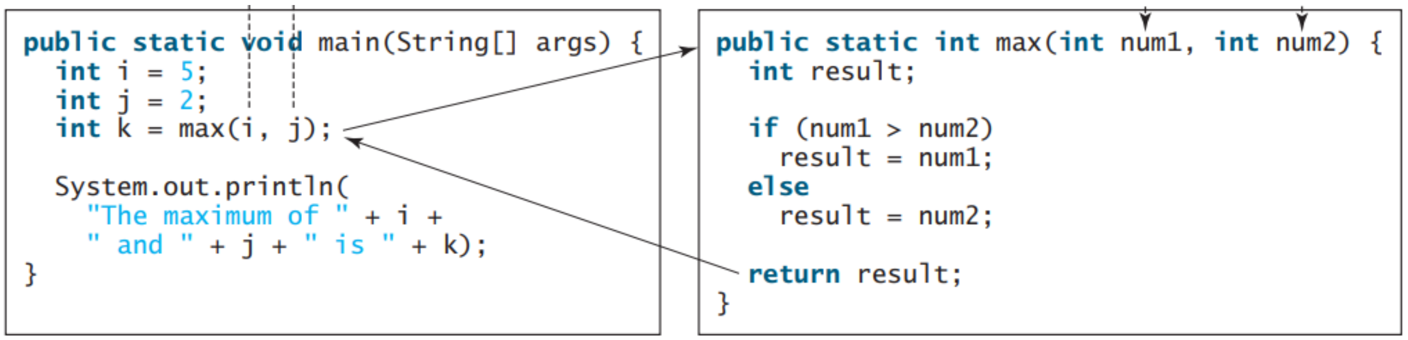


Activation record for the main method
k:
j: 2
i: 5
Activation record for the max method
result:
num2: 2
num1: 5
Activation record for the main method
k:
j: 2
i: 5

(a) The main method is invoked.

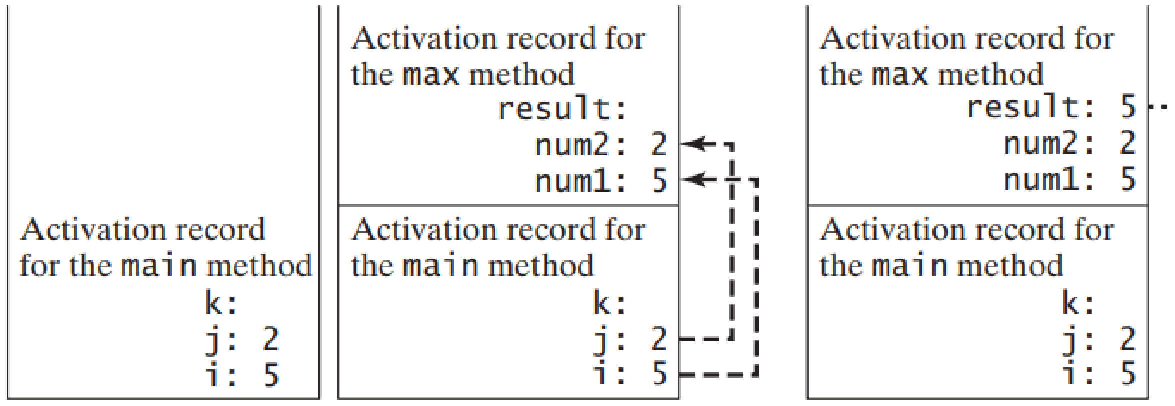
(b) The max method is invoked.

Method-Call Stack



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

  System.out.println(
    "The maximum of " + 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;
}
```



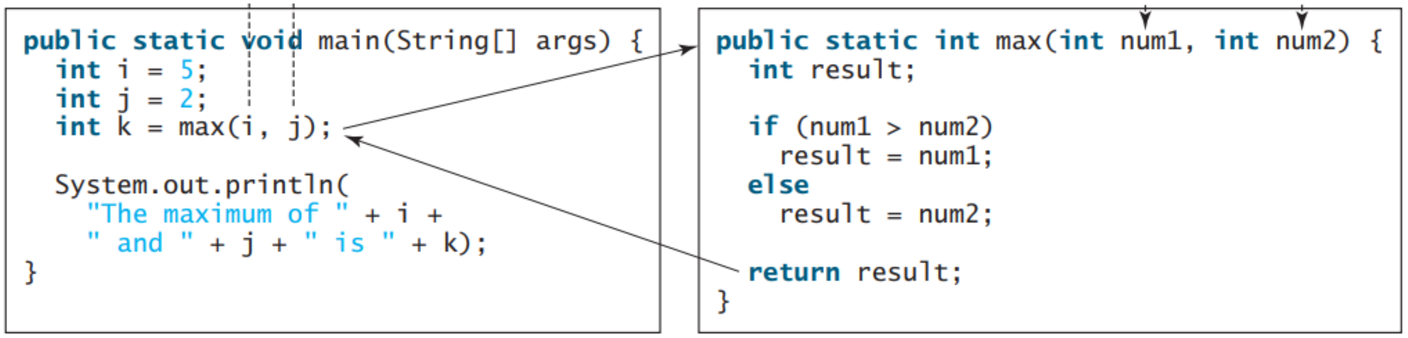
Activation record for the main method
k:
j: 2
i: 5
Activation record for the max method
result:
num2: 2
num1: 5
Activation record for the main method
k:
j: 2
i: 5
Activation record for the max method
result: 5
num2: 2
num1: 5
Activation record for the main method
k:
j: 2
i: 5

(a) The main method is invoked.

(b) The max method is invoked.

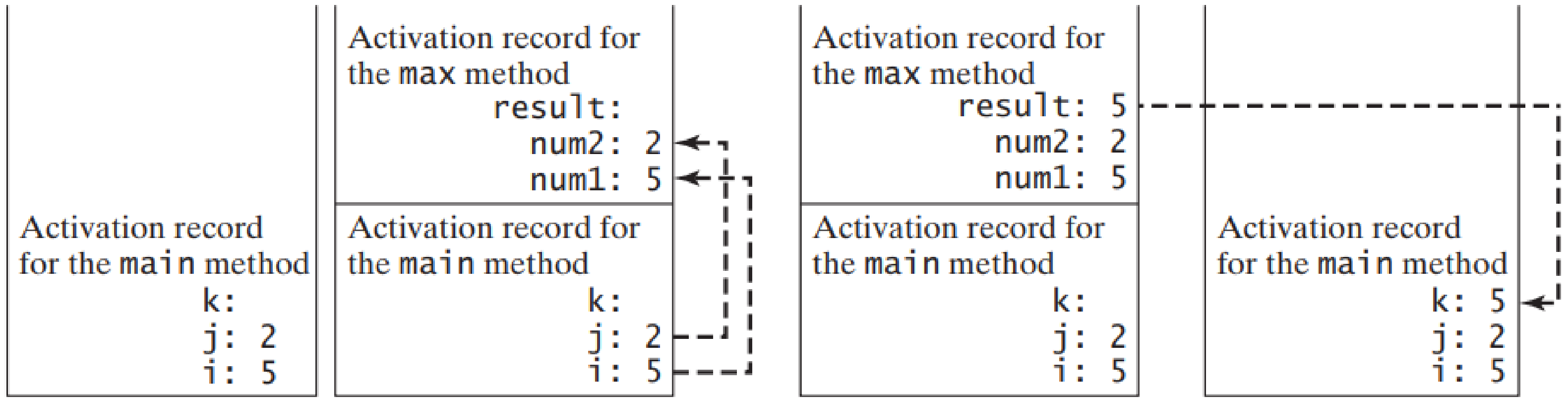
(c) The max method is being executed.

Method-Call Stack



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

  System.out.println(
    "The maximum of " + 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;
}
```



Activation record for the main method
k:
j: 2
i: 5
Activation record for the max method
result:
num2: 2
num1: 5
Activation record for the main method
k:
j: 2
i: 5
Activation record for the max method
result: 5
num2: 2
num1: 5
Activation record for the main method
k:
j: 2
i: 5
Activation record for the main method
k: 5
j: 2
i: 5

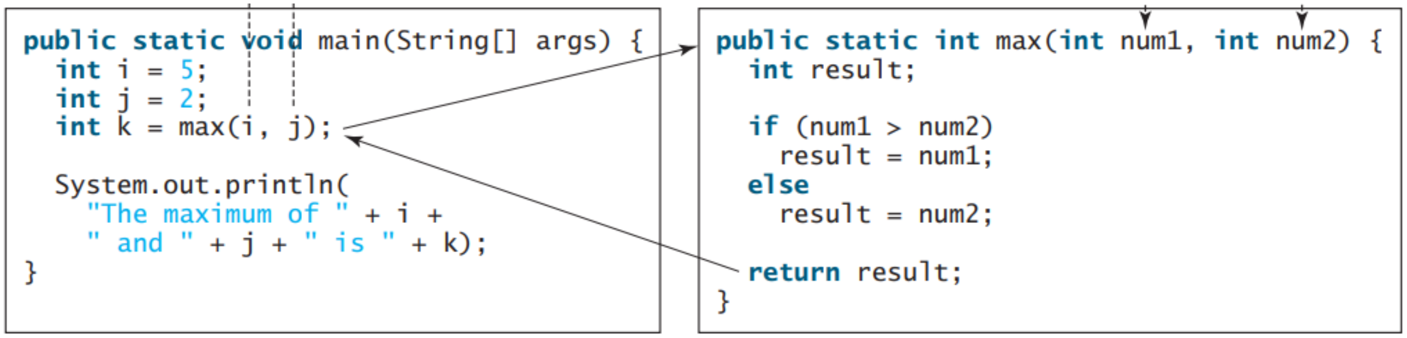
(a) The `main` method is invoked.

(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`.

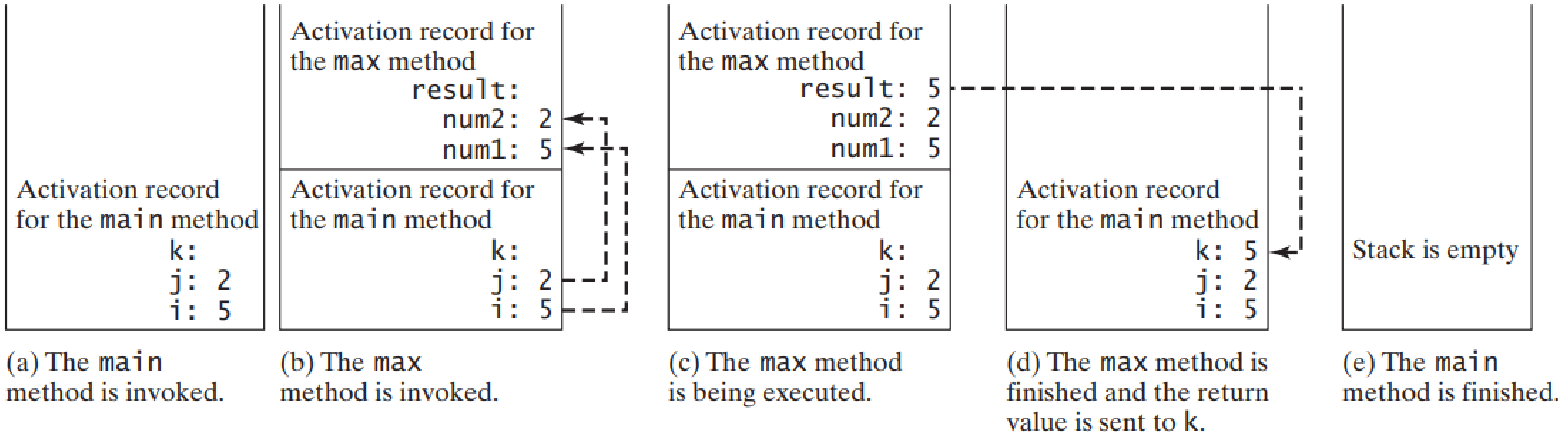
Method-Call Stack



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

  System.out.println(
    "The maximum of " + 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;
}
```

Same as argument passing, returning a primitive type copies the value, returning a reference type copies the reference.


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.

Return to the Caller

A `return` statement is not needed for a `void` method

```java
public class TestVoidMethod {
  public static void main(String[] args) {
    System.out.print("The grade is ");
    printGrade(78.5);

    System.out.print("The grade is ");
    printGrade(59.5);
  }

  public static void printGrade(double score) {
    if (score >= 90.0) {
      System.out.println('A');
    }
    else if (score >= 80.0) {
      System.out.println('B');
    }
    else if (score >= 70.0) {
      System.out.println('C');
    }
    else if (score >= 60.0) {
      System.out.println('D');
    }
    else {
      System.out.println('F');
    }
  }
}
```

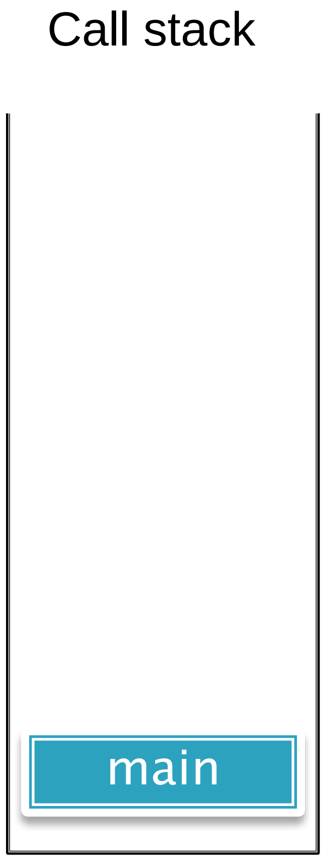
```
The grade is C
The grade is F
```

But `return` can be used for terminating the method and returning to the method's caller

```java
public static void printGrade(double score) {
  if (score < 0 || score > 100) {
    System.out.println("Invalid score");
    return;
  }

  if (score >= 90.0) {
    System.out.println('A');
  }
  else if (score >= 80.0) {
    System.out.println('B');
  }
  else if (score >= 70.0) {
    System.out.println('C');
  }
  else if (score >= 60.0) {
    System.out.println('D');
  }
  else {
    System.out.println('F');
  }
}
```

Method Call Chain

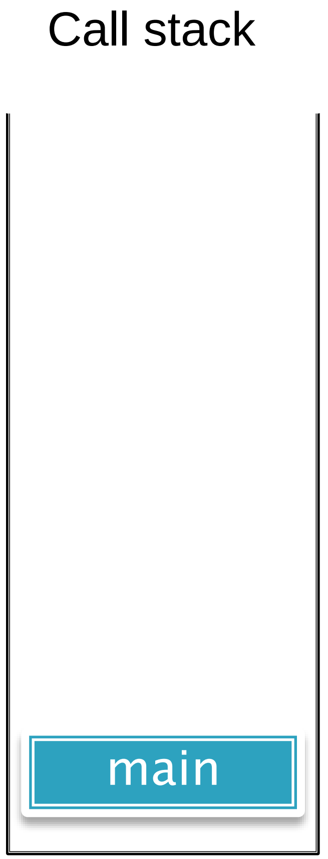
```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```



Console output

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

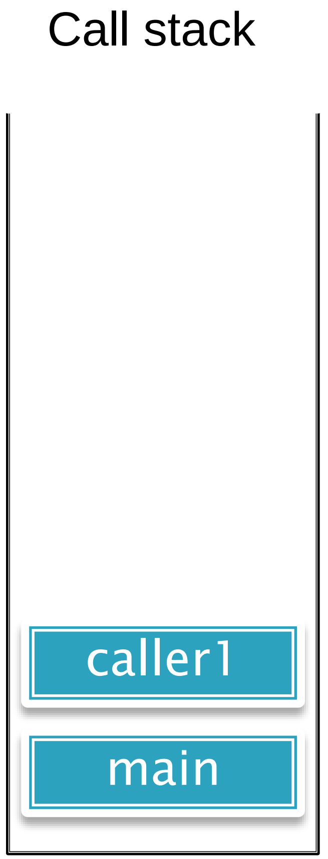


Console output

```
enter main
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```



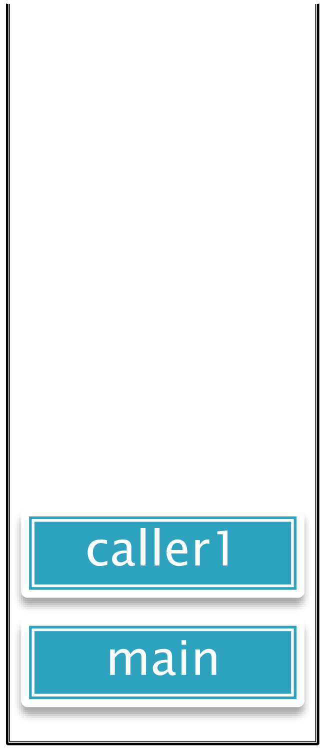
Console output

```
enter main
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



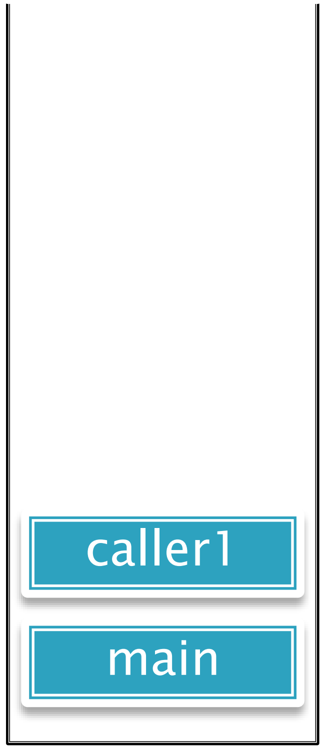
Console output

```
enter main
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



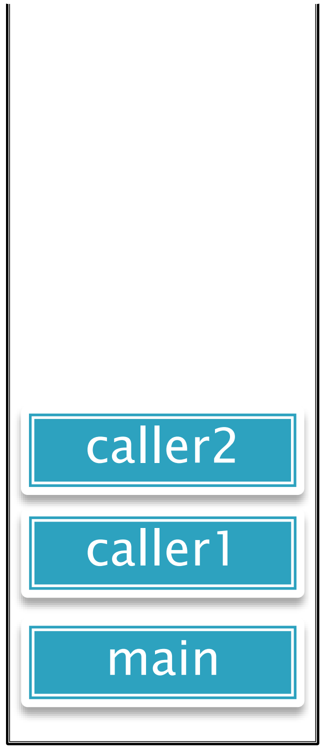
Console output

```
enter main
enter caller1
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



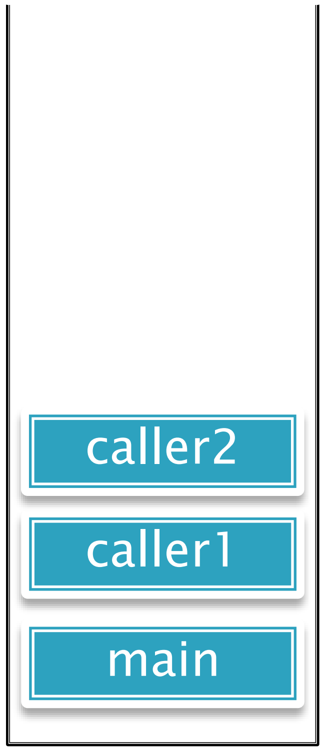
Console output

```
enter main
enter caller1
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



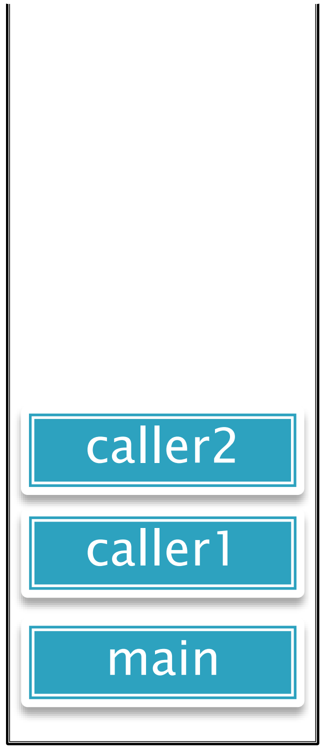
Console output

```
enter main
enter caller1
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



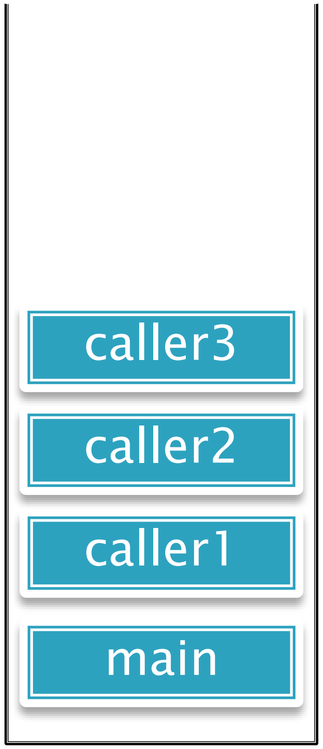
Console output

```
enter main
enter caller1
enter caller2
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



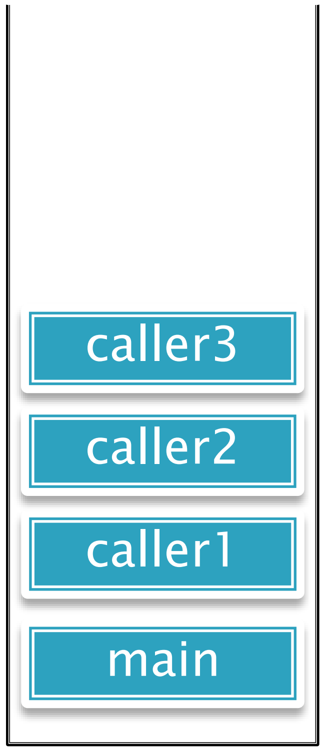
Console output

```
enter main
enter caller1
enter caller2
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



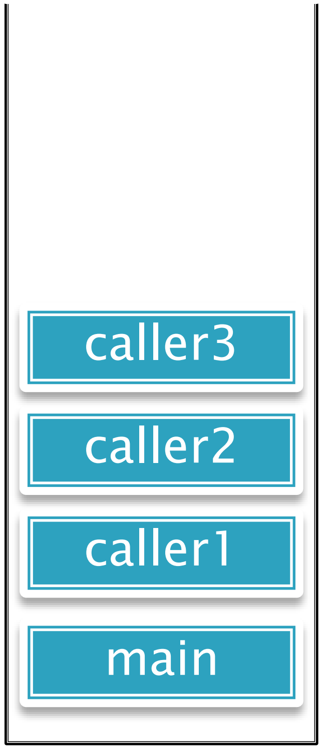
Console output

```
enter main
enter caller1
enter caller2
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



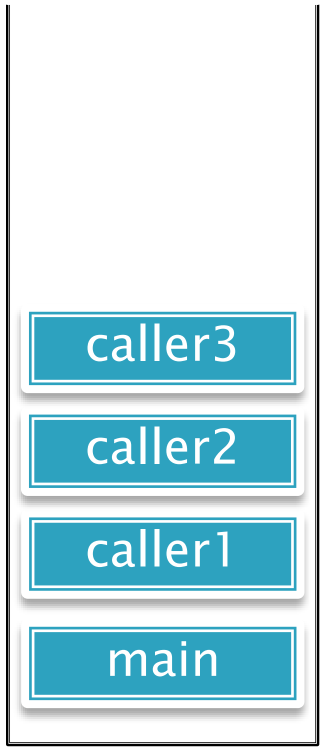
Console output

```
enter main
enter caller1
enter caller2
caller3 executed
```

Method Call Chain

```java
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



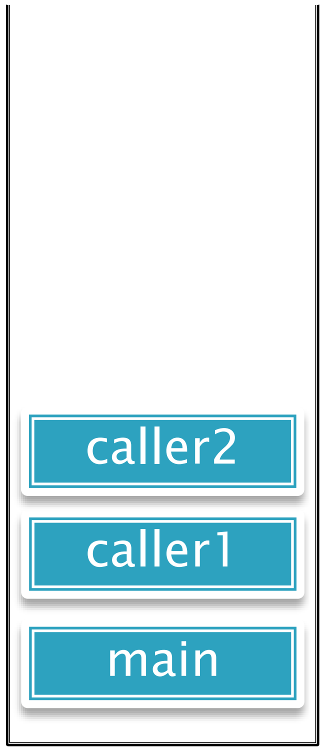
Console output

```
enter main
enter caller1
enter caller2
caller3 executed
```

Method Call Chain

```java
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack

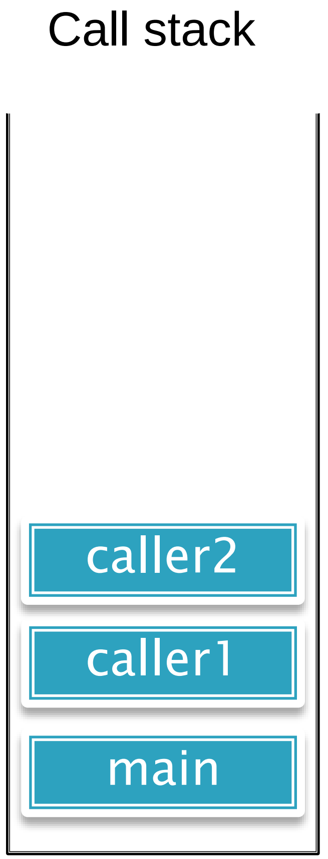


Console output

```
enter main
enter caller1
enter caller2
caller3 executed
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```



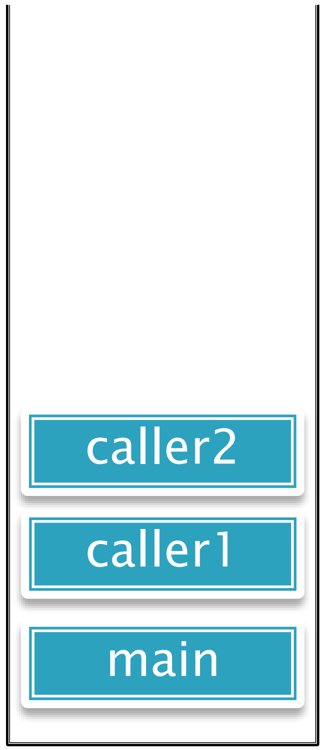
Console output

```
enter main
enter caller1
enter caller2
caller3 executed
exit caller2
```

Method Call Chain

```java
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



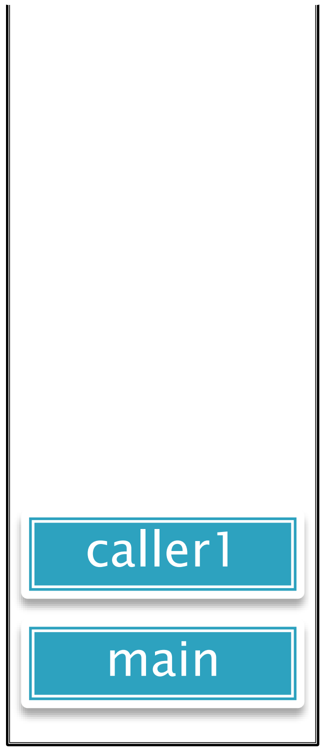
Console output

```
enter main
enter caller1
enter caller2
caller3 executed
exit caller2
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



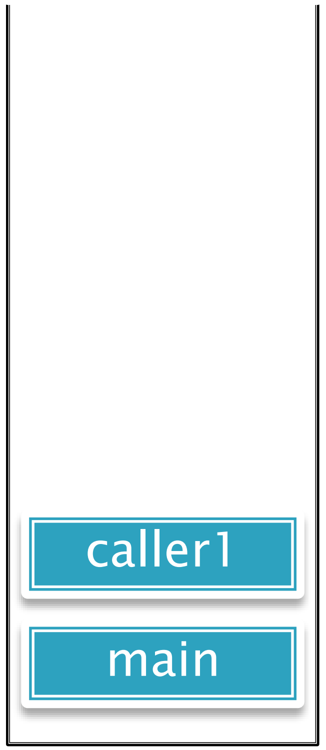
Console output

```
enter main
enter caller1
enter caller2
caller3 executed
exit caller2
```

Method Call Chain

```java
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



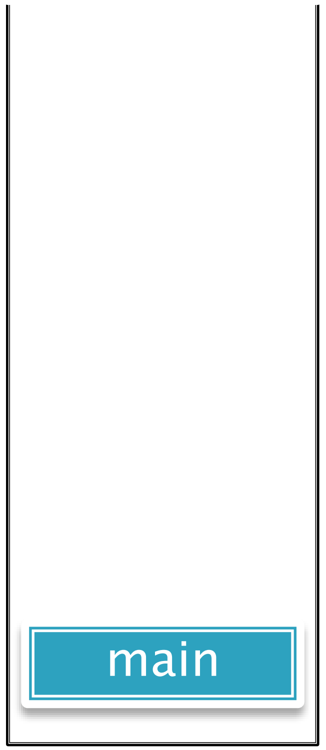
Console output

```
enter main
enter caller1
enter caller2
caller3 executed
exit caller2
exit caller1
```

Method Call Chain

```java
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack



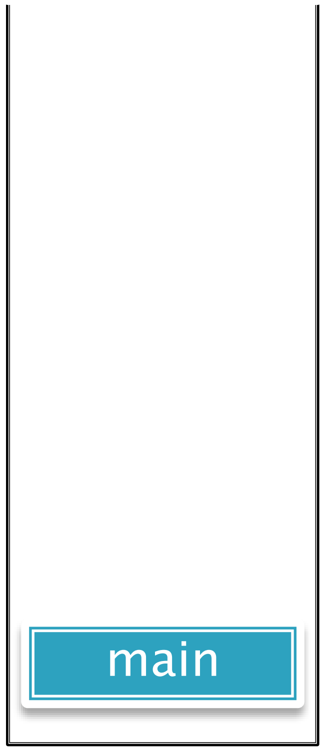
Console output

```
enter main
enter caller1
enter caller2
caller3 executed
exit caller2
exit caller1
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```

Call stack

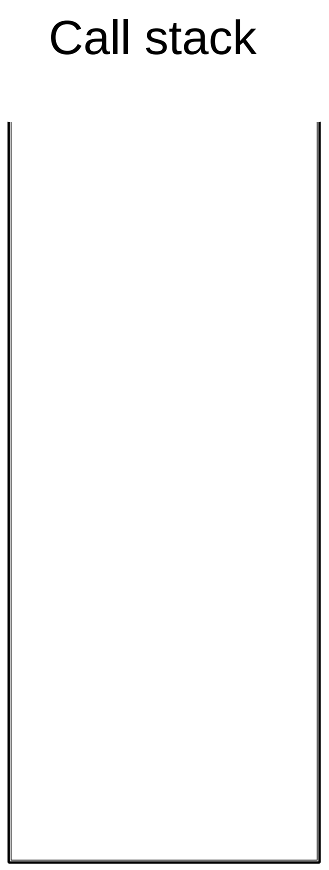


Console output

```
enter main
enter caller1
enter caller2
caller3 executed
exit caller2
exit caller1
exit main
```

Method Call Chain

```
public class CallChainDemo {
    public static void main(String[] args) {
        System.out.println("enter main");
        caller1();
        System.out.println("exit main");
    }

    public static void caller1() {
        System.out.println("enter caller1");
        caller2();
        System.out.println("exit caller1");
    }

    public static void caller2() {
        System.out.println("enter caller2");
        caller3();
        System.out.println("exit caller2");
    }

    public static void caller3() {
        System.out.println("caller3 executed");
    }
}
```



Console output

```
enter main
enter caller1
enter caller2
caller3 executed
exit caller2
exit caller1
exit main
```

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Command-line arguments

Method Overloading (方法重载)

Overloaded methods are methods in the same class that:

- Have the same method name (perform similar tasks)
- Have different type and number of parameters

```java
/** Return the max of two int values */
public static int max(int num1, int num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

/** Find the max of two double values */
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

/** Return the max of three double values */
public static double max(double num1, double num2, double num3) {
  return max(max(num1, num2), num3);
}
```

Method Overloading (方法重载)

```java
public static void main(String[] args) {
  // Invoke the max method with int parameters
  System.out.println("The maximum of 3 and 4 is "
    + max(3, 4));

  // Invoke the max method with the double parameters
  System.out.println("The maximum of 3.0 and 5.4 is "
    + max(3.0, 5.4));

  // Invoke the max method with three double parameters
  System.out.println("The maximum of 3.0, 5.4, and 10.14 is "
    + max(3.0, 5.4, 10.14));
}
```

```java
/** Return the max of two int values */
public static int max(int num1, int num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

/** Find the max of two double values */
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}

/** Return the max of three double values */
public static double max(double num1, double num2, double num3) {
  return max(max(num1, num2), num3);
}
```

Java compiler selects the appropriate method to call by examining the number and types of the arguments in the call

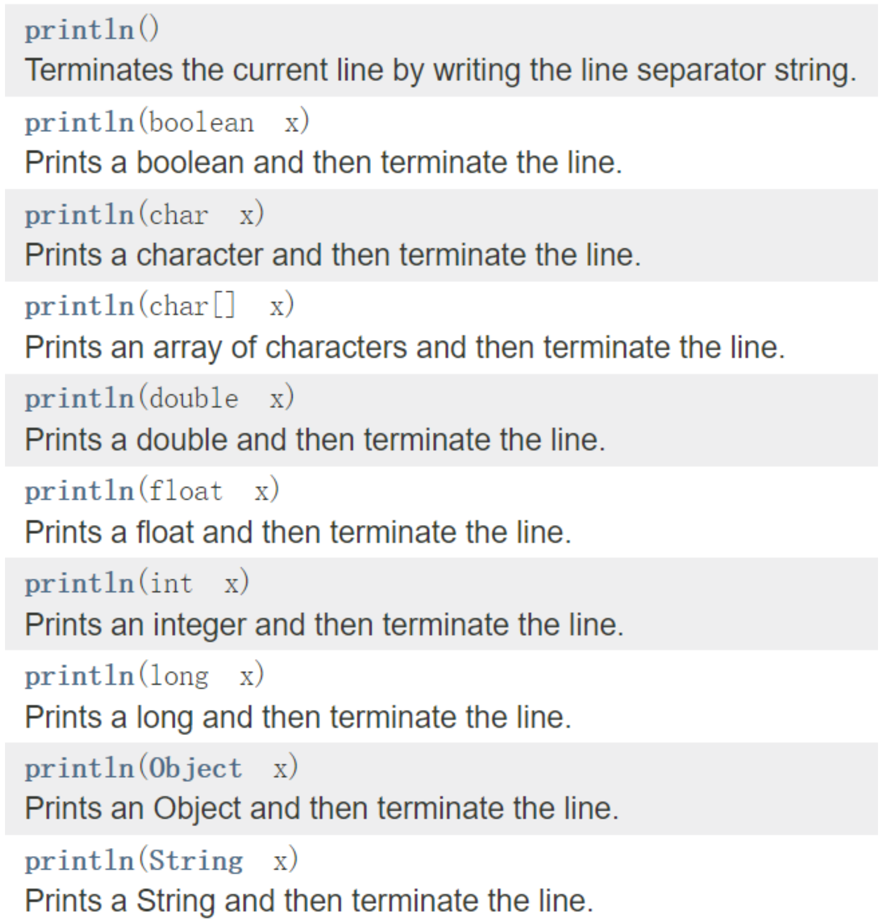
Method Overloading (方法重载)

- Method calls cannot be distinguished by return type. If you have overloaded methods only with different return types:
  - `int square(int a)`
  - `double square(int a)`
- and you called the method as follows
  - `square(2);` X
- the compiler will be confused on which methods to call, hence having compilation errors.

Method Overloading in Java

`System.out.println`

are overloaded

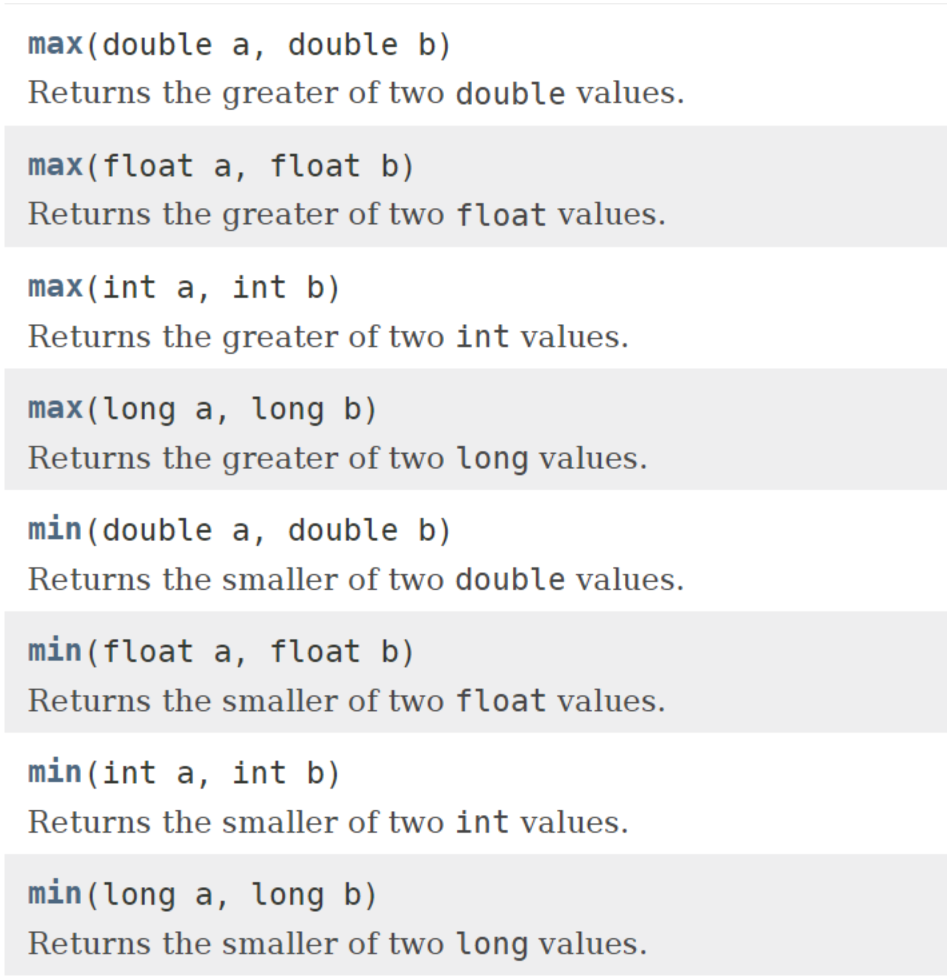


`println()`	Terminates the current line by writing the line separator string.
`println(boolean x)`	Prints a boolean and then terminate the line.
`println(char x)`	Prints a character and then terminate the line.
`println(char[] x)`	Prints an array of characters and then terminate the line.
`println(double x)`	Prints a double and then terminate the line.
`println(float x)`	Prints a float and then terminate the line.
`println(int x)`	Prints an integer and then terminate the line.
`println(long x)`	Prints a long and then terminate the line.
`println(Object x)`	Prints an Object and then terminate the line.
`println(String x)`	Prints a String and then terminate the line.

Method Overloading in Java

`Math.max, Math.min`

are overloaded



`max(double a, double b)`	Returns the greater of two `double` values.
`max(float a, float b)`	Returns the greater of two `float` values.
`max(int a, int b)`	Returns the greater of two `int` values.
`max(long a, long b)`	Returns the greater of two `long` values.
`min(double a, double b)`	Returns the smaller of two `double` values.
`min(float a, float b)`	Returns the smaller of two `float` values.
`min(int a, int b)`	Returns the smaller of two `int` values.
`min(long a, long b)`	Returns the smaller of two `long` values.

Variable-Length Argument Lists

- With variable-length argument lists (可变长参数列表), you can create methods that receive an unspecified number of arguments.
- A type followed by an ellipsis (...) in a method's parameter list indicates that the method receives a variable number of arguments of that particular type.

```
public static double average(double... numbers)
```

Variable-Length Argument Lists

Java treats the variable-length argument list as an array of the specified type.

```
public class VarArgsDemo {

    public static double average(double... numbers) {
        double total = 0.0;
        for(double d : numbers) total += d;
        return total / numbers.length;
    }

    public static void main(String[] args) {
        double d1 = 10.0, d2 = 20.0, d3 = 30.0;

        double avg1 = average(d1); // 10.0
        double avg2 = average(d1, d2); // 15.0
        double avg3 = average(d1, d2, d3); // 20.0

        double avg0 = average(); // NaN
    }

}
```

Same as `average([10.0])`

Same as `average([10.0, 20.0])`

Same as `average([10.0, 20.0, 30.0])`

Same as `average([])`

Variable-Length Argument Lists

Can occur only once in a parameter list, and the ellipsis, together with its type, must be placed at the end of the parameter list.

```
void average(int a, double... numbers){}  √
```

```
void average(double... numbers, int a){}  ✗
```

Vararg parameter must be the last in the list

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Command-line arguments

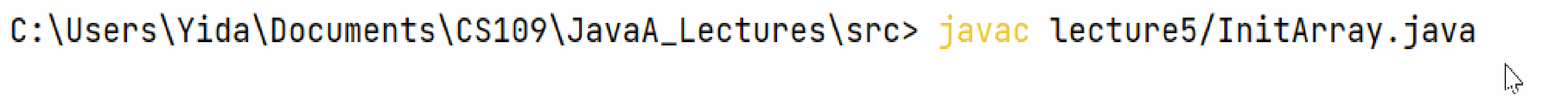
Passing Command-Line Arguments

- We can pass arguments from the command line to an application by including a parameter of type `String[]` in the parameter list of `main`.

```
public static void main(String[] args)
```

- By convention, this parameter is named `args`.
- When an application is executed using the `java` command, Java passes the command-line arguments that appear after the class name in the `java` command to the `main` method as `String`s in the array `args`.

Ex: Initialize an array by specifying its size, first element, and interval



```
C:\Users\Yida\Documents\CS109\JavaA_Lectures\src> javac lecture5/InitArray.java
```

```java
// Fig. 6.15: InitArray.java
// Initializing an array using command-line arguments.

public class InitArray
{
   public static void main( String[] args )
   {
      // check number of command-line arguments
      if ( args.length != 3 )
         System.out.println(
            "Error: Please re-enter the entire command, including\n" +
            "an array size, initial value and increment." );
      else
      {
         // get array size from first command-line argument
         int arrayLength = Integer.parseInt( args[ 0 ] );
         int[] array = new int[ arrayLength ]; // create array

         // get initial value and increment from command-line arguments
         int initialValue = Integer.parseInt( args[ 1 ] );
         int increment = Integer.parseInt( args[ 2 ] );

```

Fig. 6.15 | Initializing an array using command-line arguments. (Part 1 of 3.)

```java
            // calculate value for each array element
            for ( int counter = 0; counter < array.length; counter++ )
               array[ counter ] = initialValue + increment * counter;

            System.out.printf( "%s%8s\n", "Index", "Value" );

            // display array index and value
            for ( int counter = 0; counter < array.length; counter++ )
               System.out.printf( "%5d%8d\n", counter, array[ counter ] );
         } // end else
      } // end main
   } // end class InitArray
```

```
java InitArray
Error: Please re-enter the entire command, including
an array size, initial value and increment.
```

Fig. 6.15 | Initializing an array using command-line arguments. (Part 2 of 3.)