Chapter 5: Methods

Yepang LIU (刘烨庞)

liuyp1@sustech.edu.cn

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

So Far…

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

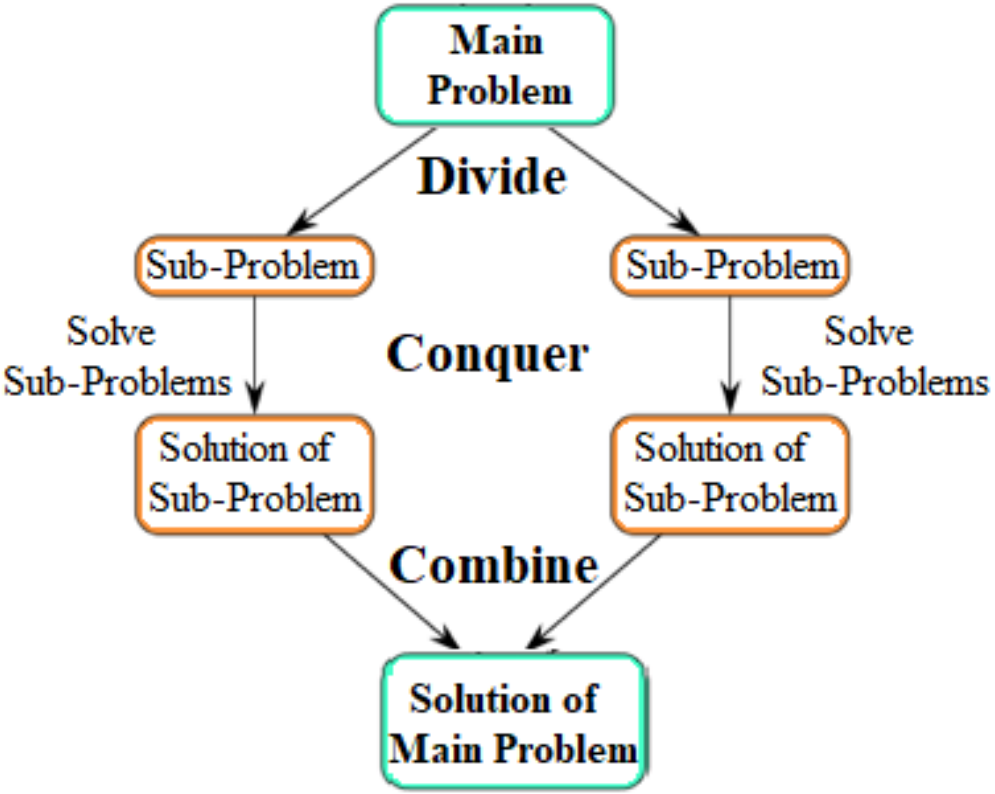
What if you are asked to **solve complex problems?**

Write a giant `main` method?

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

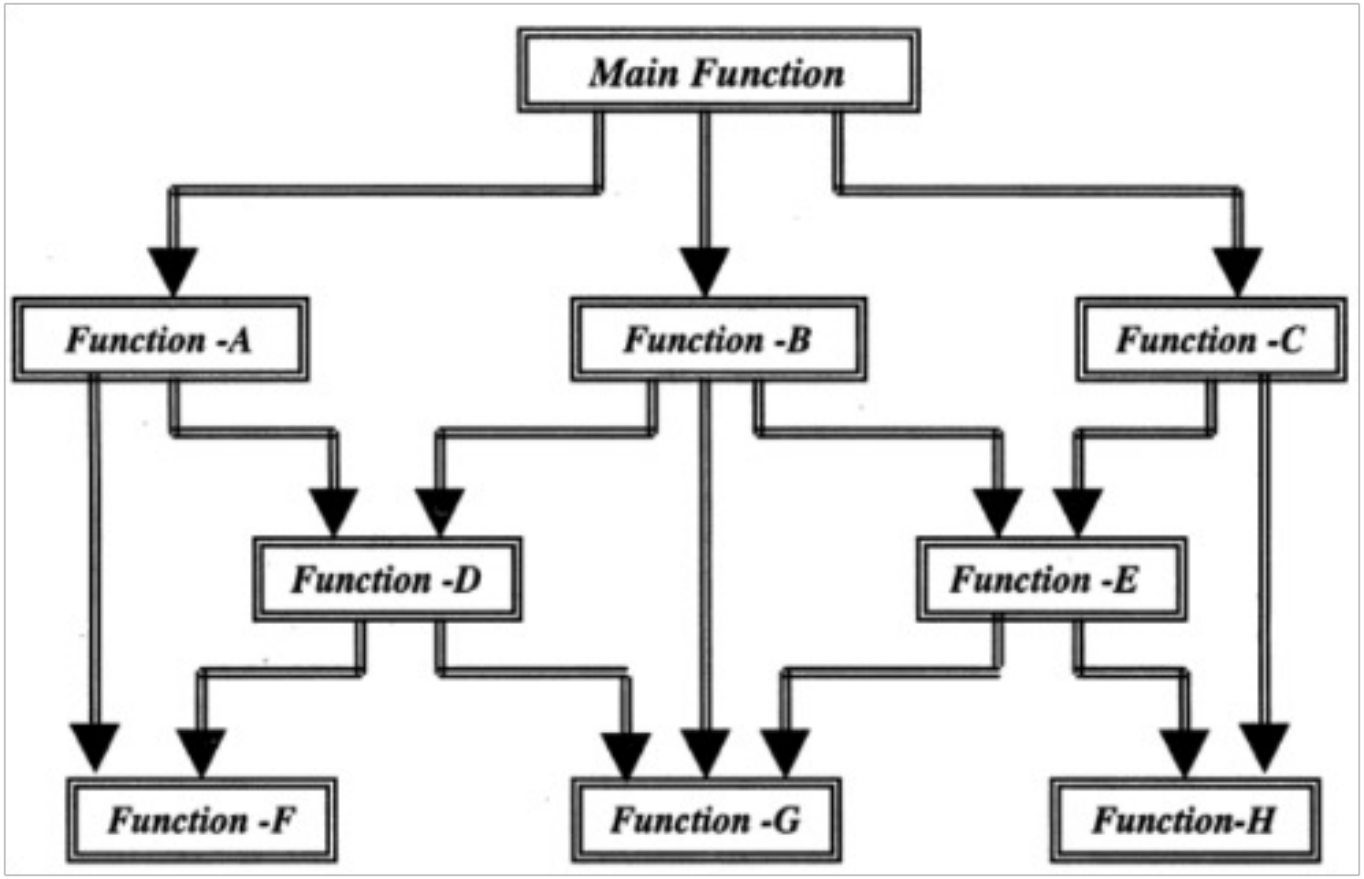
Divide and Conquer (分而治之)

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



©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Divide and Conquer (分而治之)



©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Modular programming

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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);
   }
}
```

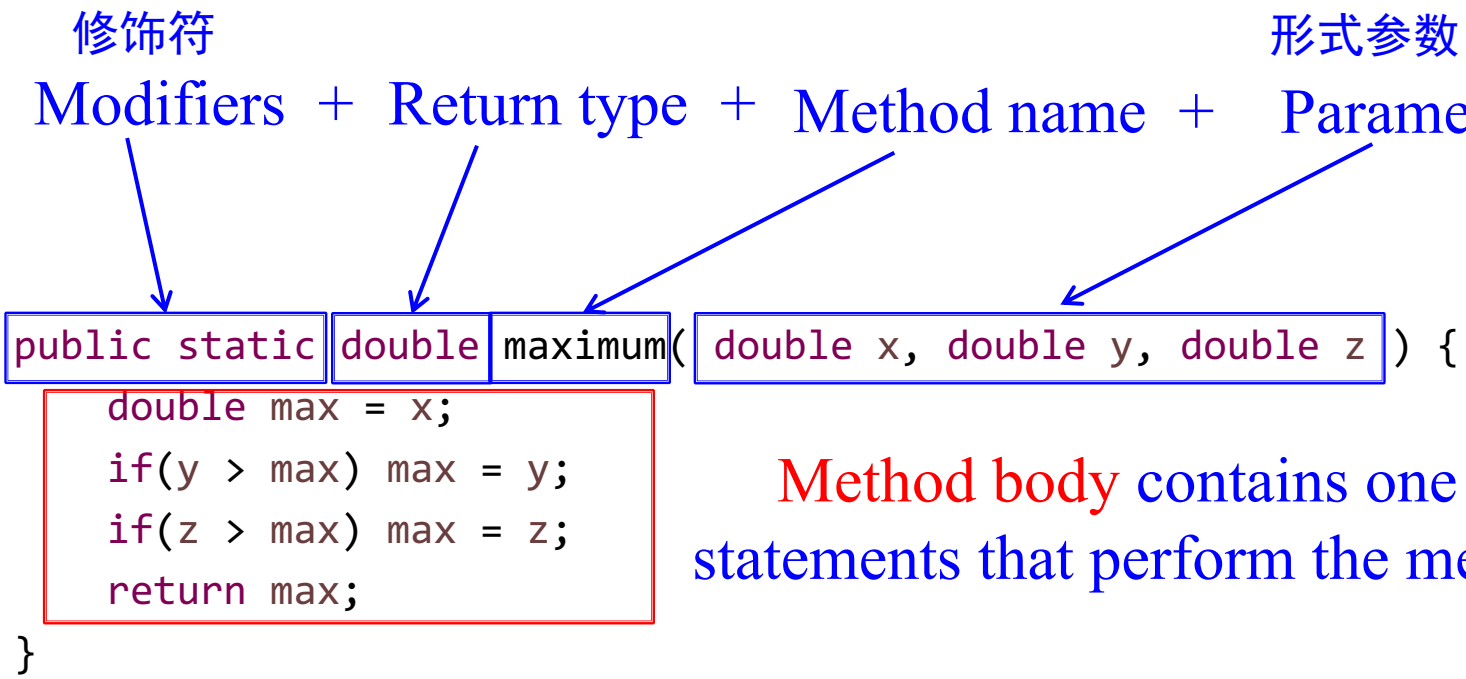
The class defines two methods

Find the largest of 3 double values

3 invocations of the method with different arguments

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Declaring a Method



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

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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`) `void println() // PrintStream class`
- Primitive values (e.g., an integer) `int nextInt() // Scanner class`
- References to objects, arrays `String nextLine() // Scanner class`

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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;
```

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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;
}
```

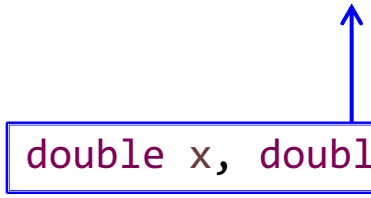
A method can have 0, 1, or more parameters

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

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

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

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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);
```

```
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.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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);
```

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

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

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

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

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Modular programming

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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.

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

Passing Primitive Type

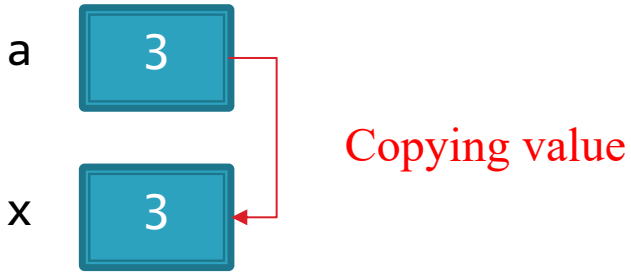
The value is simply copied

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

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

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

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



x becomes 9 after method call
a remains unchanged

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

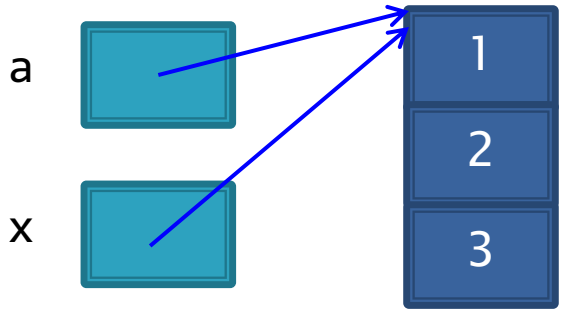
Passing Reference Type

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

```
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 (object reference).

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

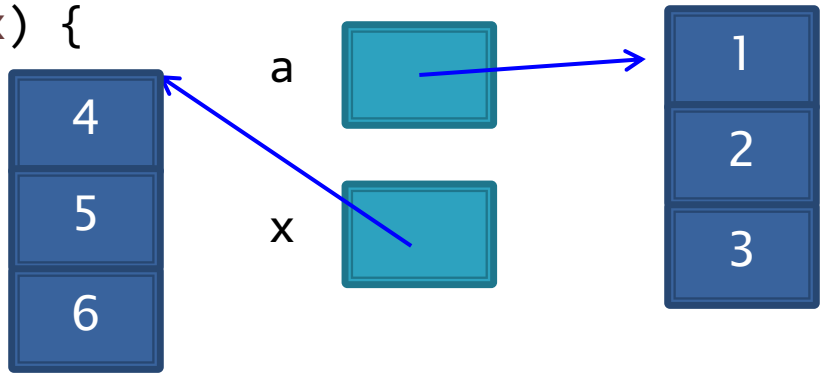
Passing Reference Type

```
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};
}
```



Copying value, which is a memory address.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

What's the output?

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

    triple(arr[0]);
    System.out.println("After: " + Arrays.toString(a));
}

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

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Using Command-Line Arguments

- We can pass arguments from the command line to a Java 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`.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

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

```
java InitArray 5 0 4
Index   Value
    0       0
    1       4
    2       8
    3      12
    4      16
```

```
java InitArray 8 1 2
Index   Value
    0       1
    1       3
    2       5
    3       7
    4       9
    5      11
    6      13
    7      15
```

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

©1992–2010 by Pearson Education, Inc.
All Rights Reserved.

```
1  // Fig. 6.15: InitArray.java
2  // Initializing an array using command-line arguments.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        // check number of command-line arguments
9        if ( args.length != 3 )
10          System.out.println(
11             "Error: Please re-enter the entire command, including\n" +
12             "an array size, initial value and increment." );
13       else
14       {
15          // get array size from first command-line argument
16          int arrayLength = Integer.parseInt( args[ 0 ] );
17          int[] array = new int[ arrayLength ]; // create array
18
19          // get initial value and increment from command-line arguments
20          int initialValue = Integer.parseInt( args[ 1 ] );
21          int increment = Integer.parseInt( args[ 2 ] );
22
```

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

©1992–2010 by Pearson Education, Inc.
All Rights Reserved.

```
23            // calculate value for each array element
24            for ( int counter = 0; counter < array.length; counter++ )
25               array[ counter ] = initialValue + increment * counter;
26
27            System.out.printf( "%s%8s\n", "Index", "Value" );
28
29            // display array index and value
30            for ( int counter = 0; counter < array.length; counter++ )
31               System.out.printf( "%5d%8d\n", counter, array[ counter ] );
32         } // end else
33      } // end main
34   } // 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.)

©1992–2010 by Pearson Education, Inc.
All Rights Reserved.

Argument Promotion

- Argument promotion—converting an argument's value, if possible, to the type that the method expects to receive in its corresponding parameter

`Math.sqrt()` expects to receives a `double` argument, but it is ok to write `Math.sqrt(4)`: java converts the `int` value 4 to the `double` value 4.0

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Promotion Rules

Specify which conversions are allowed (which conversions can be performed without losing data)

Type	Valid promotions
`double`	None
`float`	`double`
`long`	`float` or `double`
`int`	`long`, `float` or `double`
`char`	`int`, `long`, `float` or `double`
`short`	`int`, `long`, `float` or `double` (but not `char`)
`byte`	`short`, `int`, `long`, `float` or `double` (but not `char`)
`boolean`	None (`boolean` values are not considered to be numbers in Java)

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Variable-Length Argument Lists

- With variable-length argument lists, 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 type.

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

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

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Example

```
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;
    System.out.printf("average of d1 and d2: %f\n", average(d1, d2));
    System.out.printf("average of d1 ~ d3: %f\n", average(d1, d2, d3));
}
```

numbers: [10.0, 20.0]

numbers: [10.0, 20.0, 30.0]

average of d1 and d2: 15.000000
average of d1 ~ d3: 20.000000

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Modular programming

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Control flow for method calls

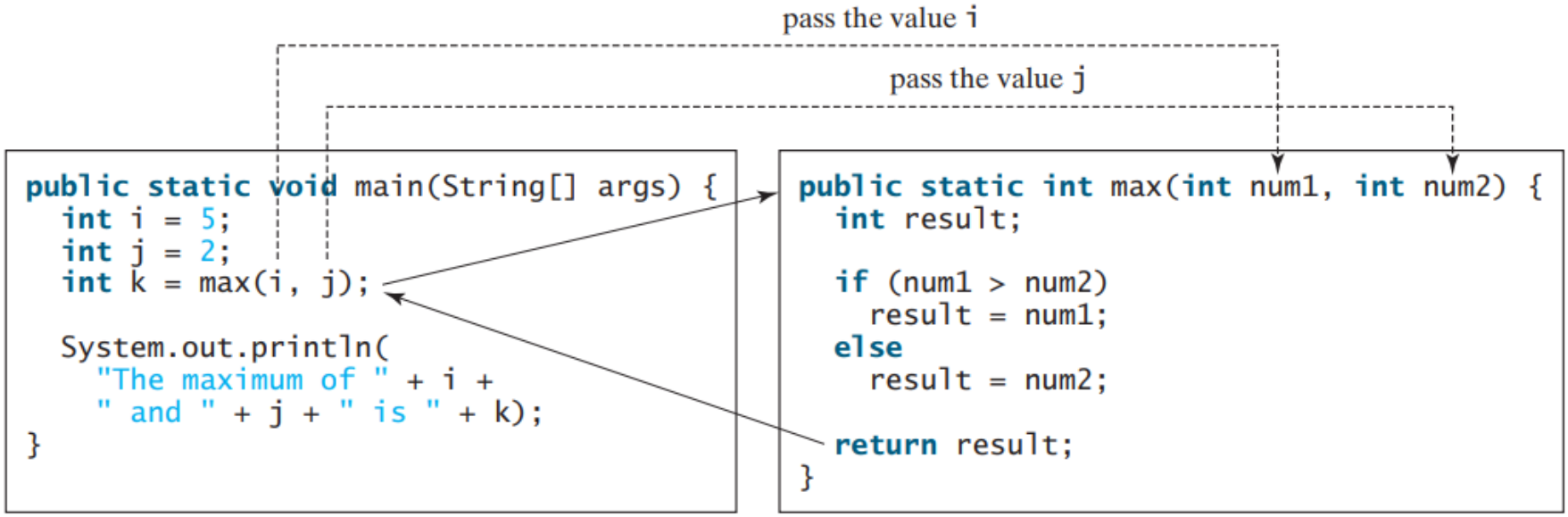


FIGURE 6.2 When the `max` method is invoked, the flow of control transfers to it. Once the `max` method is finished, it returns control back to the caller.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method-Call Stack

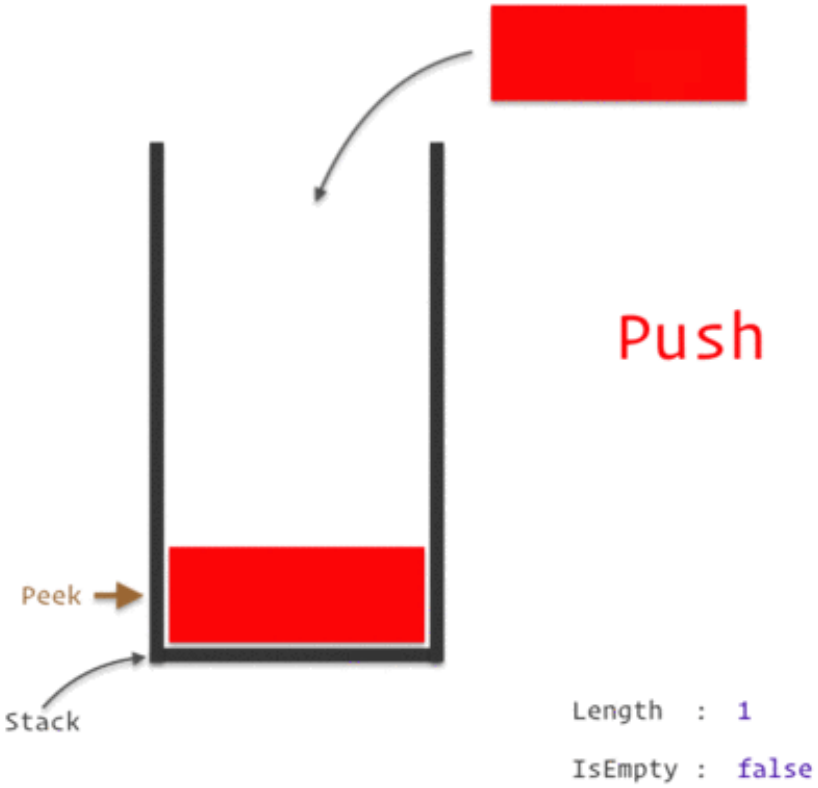
- Each time a method is invoked, the system creates an activation record (激活记录) that stores parameters and variables for the method and places the activation record in an area of memory known as a call stack (方法调用栈)

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method-Call Stack

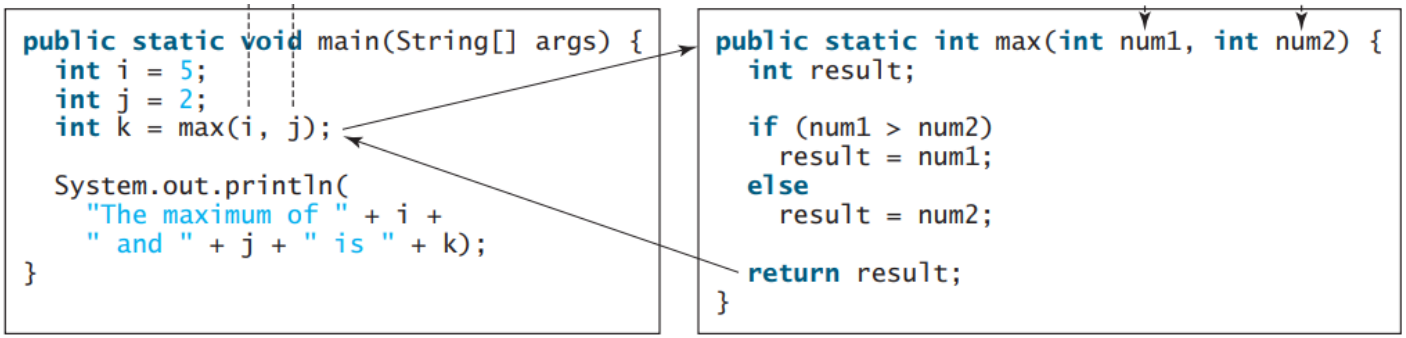
A real-life analogy

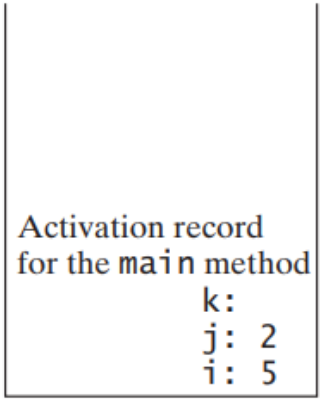
- When a method calls another method, the caller's activation record is kept intact, and a new activation record is created for the new method called and pushed to the stack.
- When a method finishes its work and returns to its caller, its activation record is popped from the call stack.
- A call stack 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.



©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method-Call Stack





(a) The `main` method is invoked.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method-Call Stack

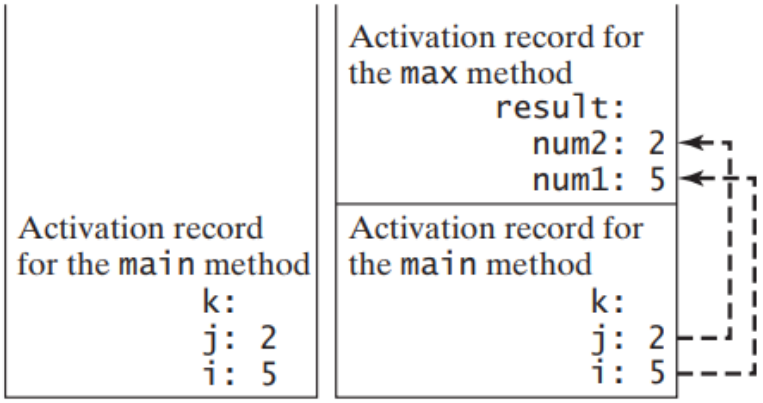
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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.

(b) The `max` method is invoked.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method-Call Stack

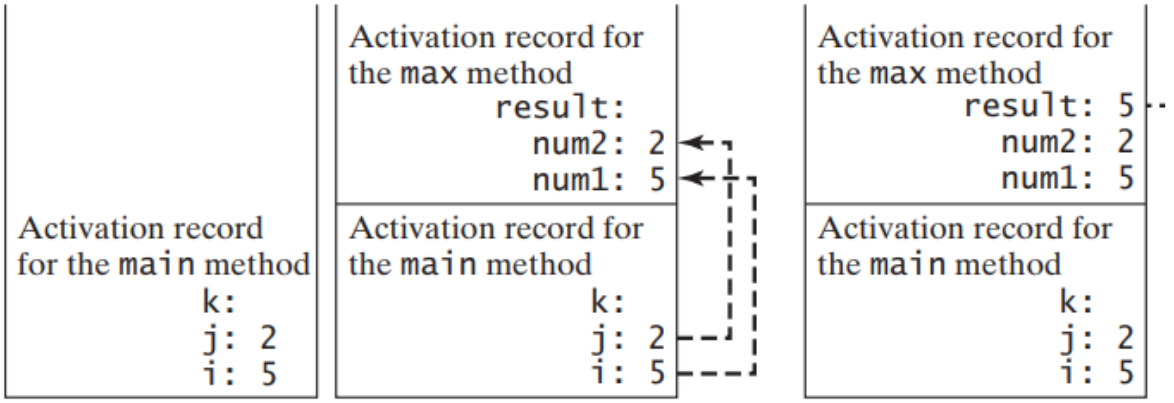
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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.

(b) The `max` method is invoked.

(c) The `max` method is being executed.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method-Call Stack

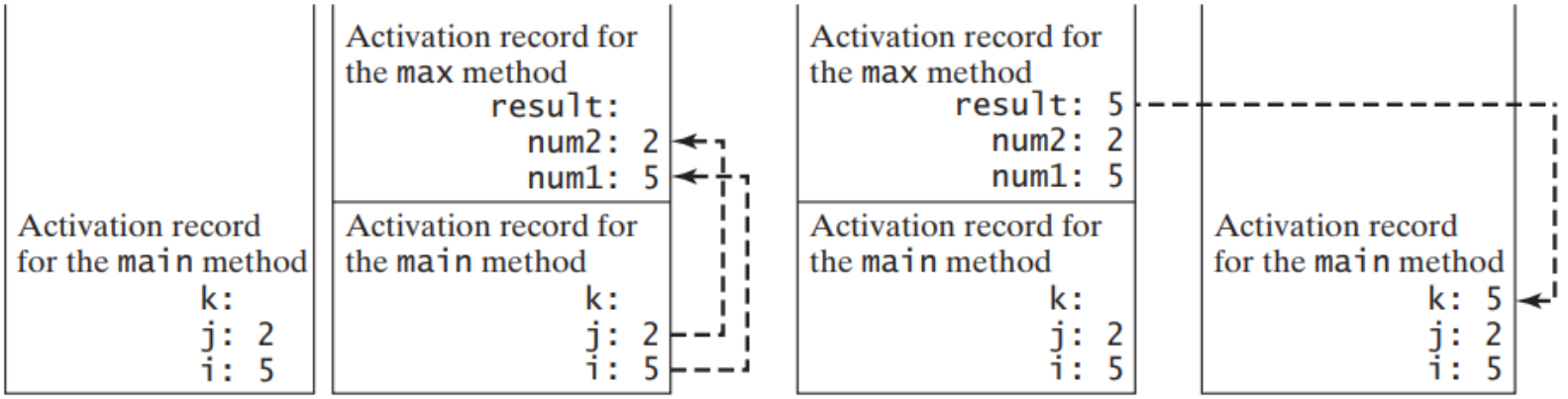
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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.

(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`.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method-Call Stack

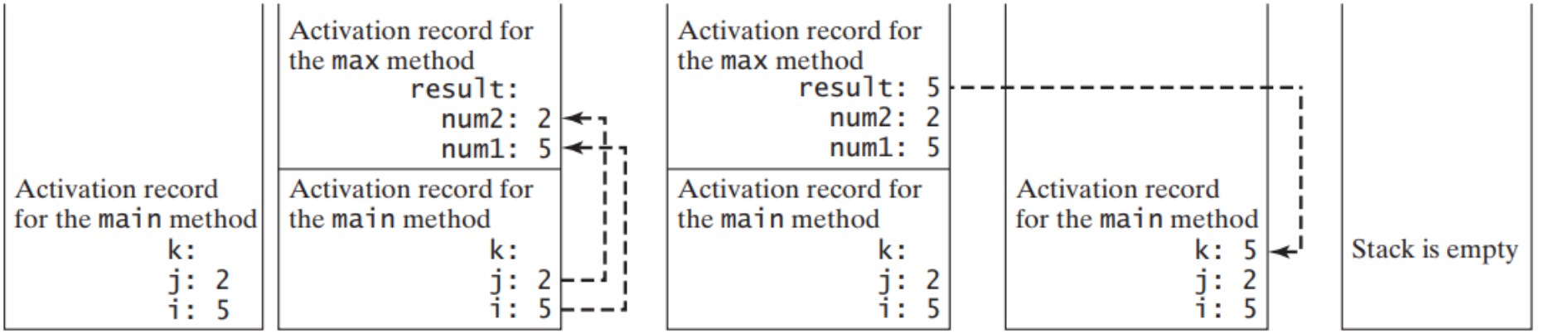
```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  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.

(b) The `max` method is invoked.

(c) The `max` method is being executed.

(d) The `max` method is finished and the return value is sent to `k`.

(e) The `main` method is finished.

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

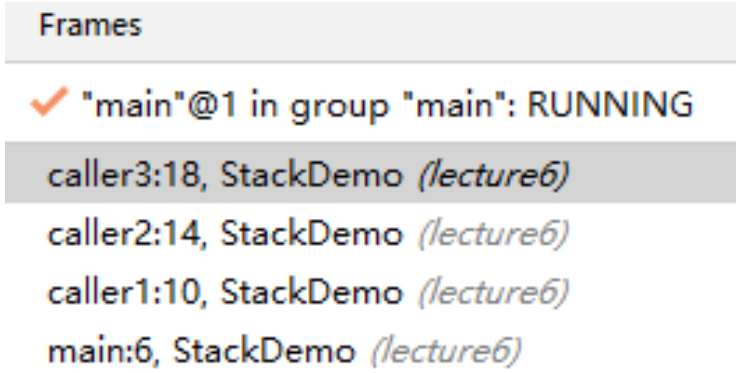
Inspecting the Call Stack

```
public class StackDemo{
   public static void main(String[] args) {
      int x = 10;
      caller1(x);
   }

   public static int caller1(int y) {
      return caller2(y);
   }

   public static int caller2(int y) {
      return caller3(y);
   }

   public static int caller3(int y) {
      return y/0;
   }
}
```




```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at lecture6.StackDemo.caller3(StackDemo.java:18)
    at lecture6.StackDemo.caller2(StackDemo.java:14)
    at lecture6.StackDemo.caller1(StackDemo.java:10)
    at lecture6.StackDemo.main(StackDemo.java:6)
```

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Modular programming

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method Signature

- Method signature (方法签名): A combination of the method's name and the number, types and order of its parameters.
- Compiler distinguishes methods by their signatures

```
public double calculateAnswer(double wingSpan, int numberOfEngines,
                              double length, double grossTons) {
  //do the calculation here
  return 0.0;
}
```

```
Signature: calculateAnswer(double, int, double, double)
```

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method Overloading (方法重载)

- Methods of the same name can be declared in the same class, as long as they have different sets of parameters
- Used to create several methods that perform the same/similar tasks on different types or different numbers of arguments
  - Java compiler selects the appropriate method to call by examining the number, types and order of the arguments in the call

```
static double    max(double a, double b)
static float     max(float a, float b)
static int       max(int a, int b)
static long      max(long a, long b)
```

```
int a = 2;
int b = 3;
Math.max(a, b);
```

Which version of `max`?

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method Overloading

- 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);`
- the compiler will be confused (since return type is ignored)
- Hence, there will be compilation error for defining multiple methods with the same signatures

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method Overloading

`System.out.println` methods are also 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.

©1992-2010 by Pearson Education, Inc. All Rights Reserved.

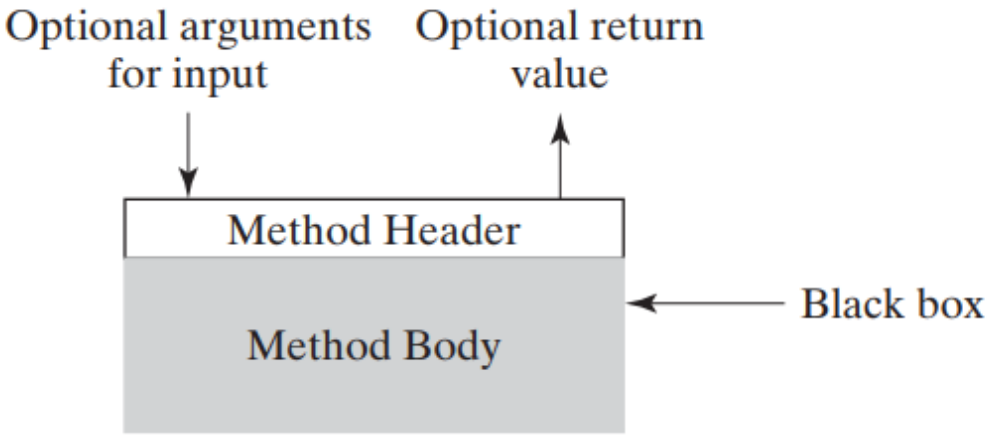
Objectives

- Method declaration and invocation
- Passing arguments
- Method call stack
- Method overloading
- Modular programming

©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Method Abstraction

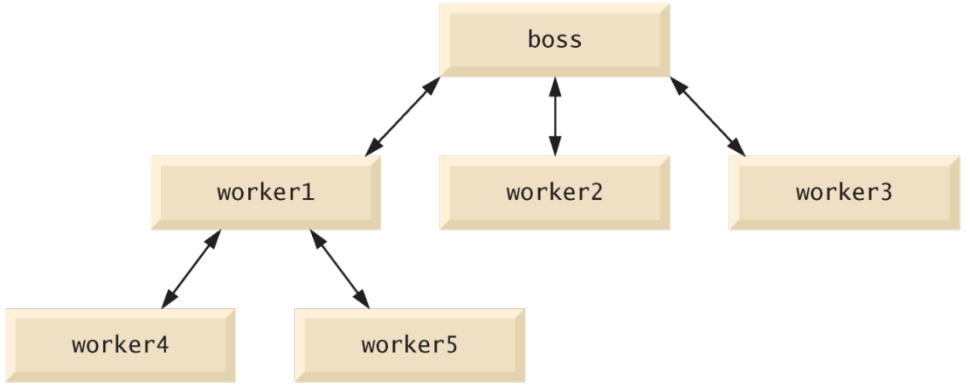
- Method abstraction separates the use of a method from its implementation.
- Clients can use the methods without knowing its implementation details



©1992–2010 by Pearson Education, Inc. All Rights Reserved.

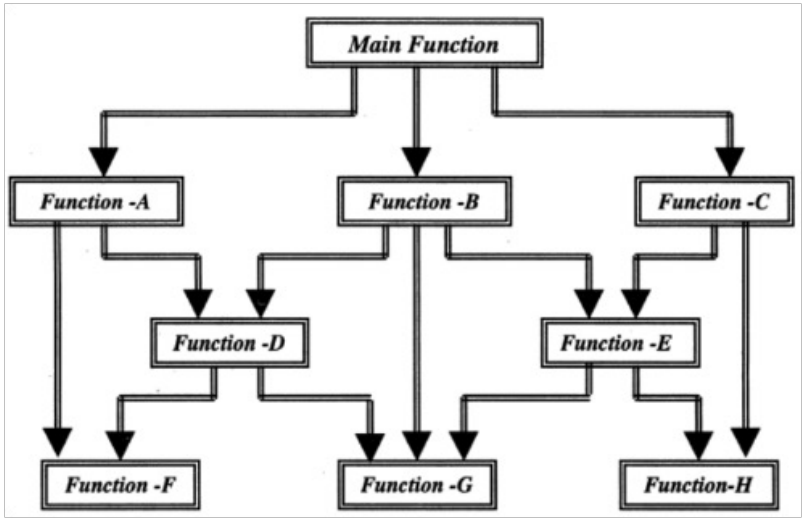
Program Modules

- Similar to the hierarchical form of management
  - A boss (the caller) asks a worker (the callee) to perform a task and report back (return) the results after completing the task
  - The boss method does not know how the worker method performs its designated tasks (method complexity is hidden)
  - The worker may also call other worker methods, unknown to the boss



©1992–2010 by Pearson Education, Inc. All Rights Reserved.

Benefits of Modular Programming
(模块化编程)



- Improved readability
- Reduce duplication
- Improve reusability
- Easier to update, test, and fix
- Easy collaboration

Modular programming facilitate the design, implementation, operation and maintenance of large programs

©1992-2010 by Pearson Education, Inc. All Rights Reserved.