Chapter 6 Methods

先看一个问题

分别求出以下区间的整数和，1 到 10, 20 到 30, 35 到 45。

解答

```
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);
```

注意代码的相似性

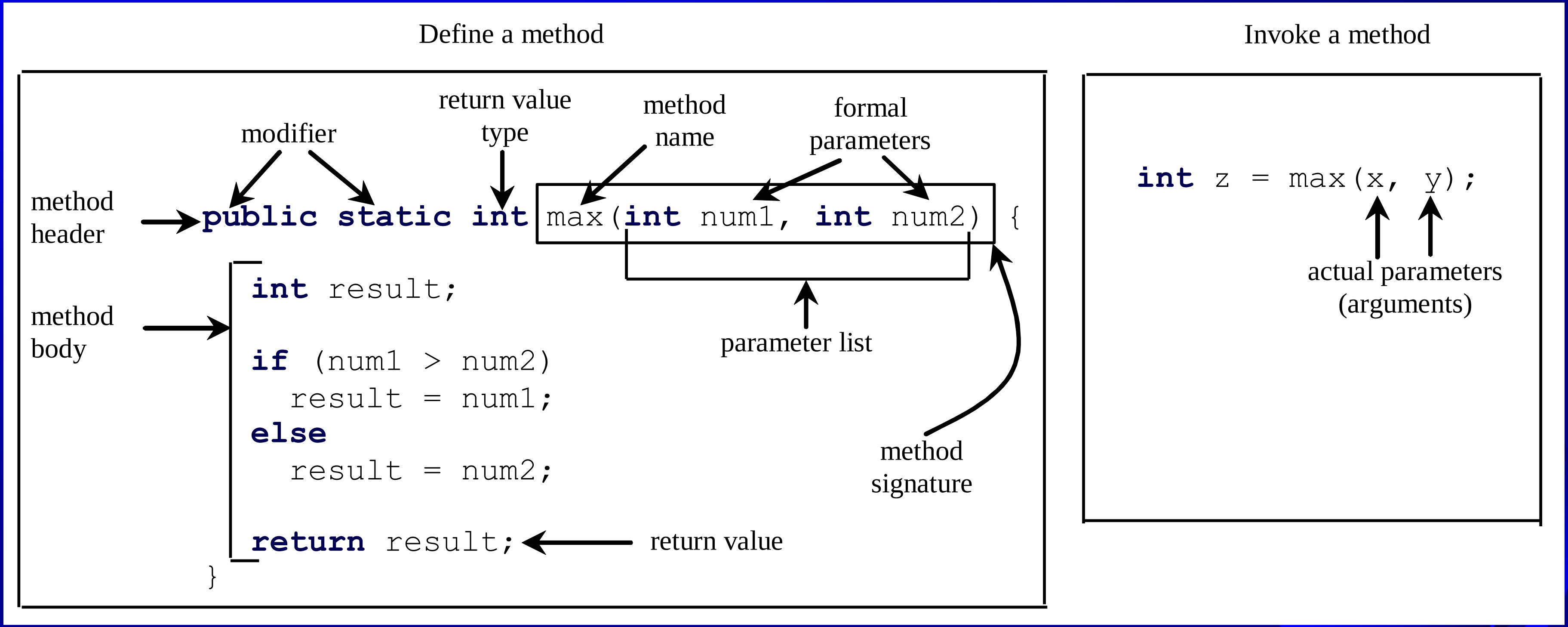
```java
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

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);
```

更好的解决方案

```
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));
}
```

定义一个方法

方法是一堆语句的组合，用来完成一个操作。

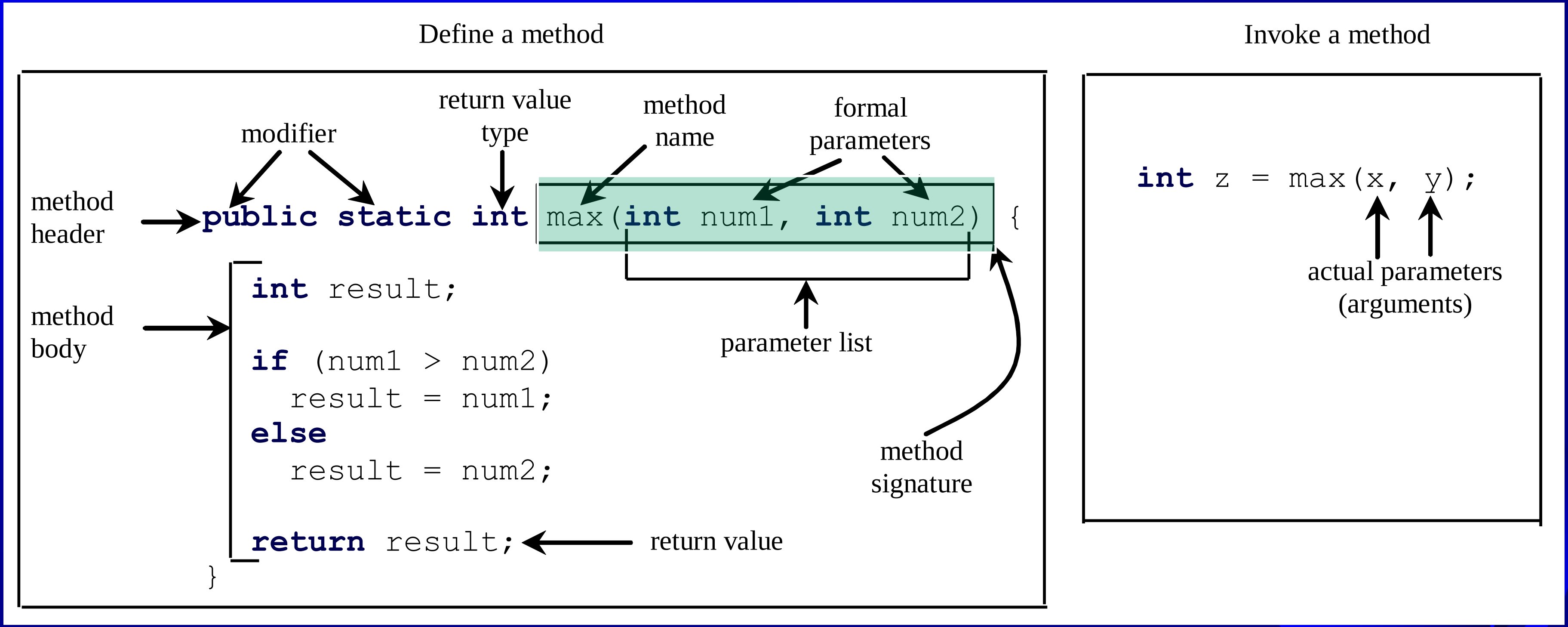


```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

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

方法签名

方法名和参数列表合起来叫做方法签名（method signature）。

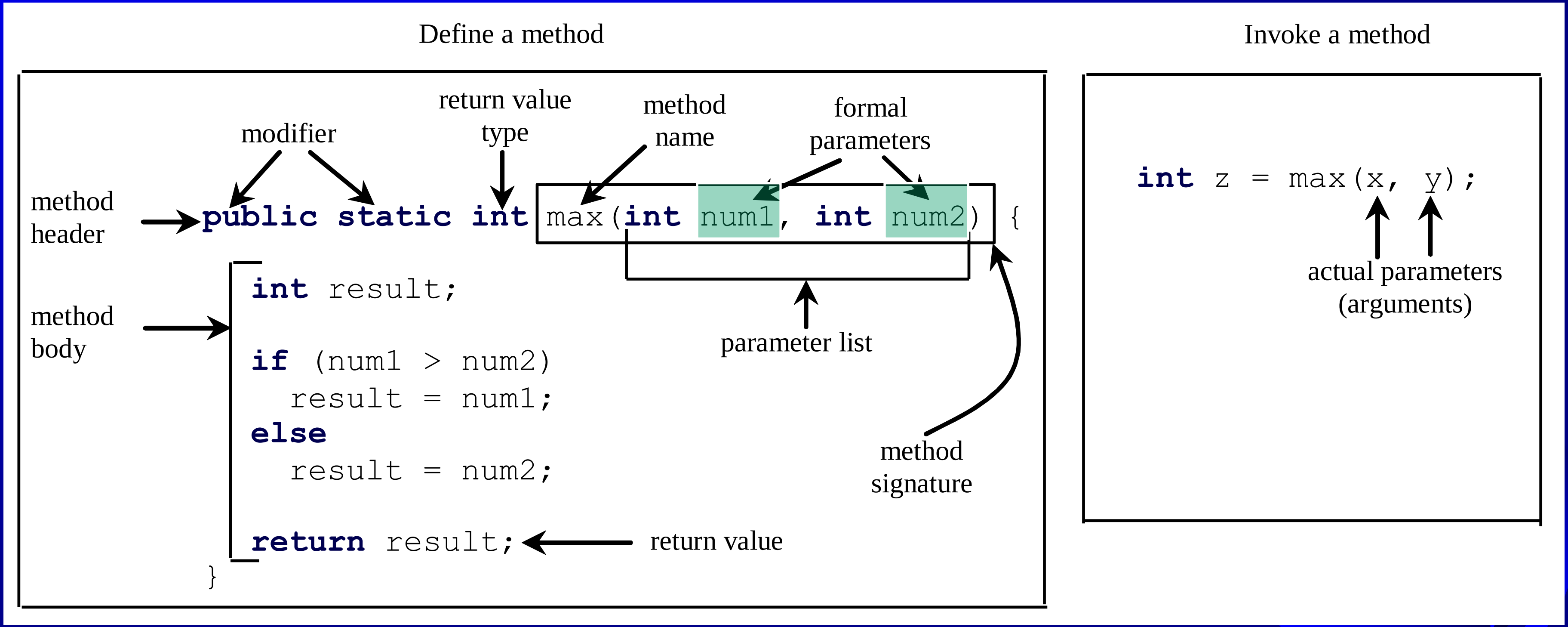


```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

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

形式参数

在方法头部定义的参数称为形式参数，简称形参。

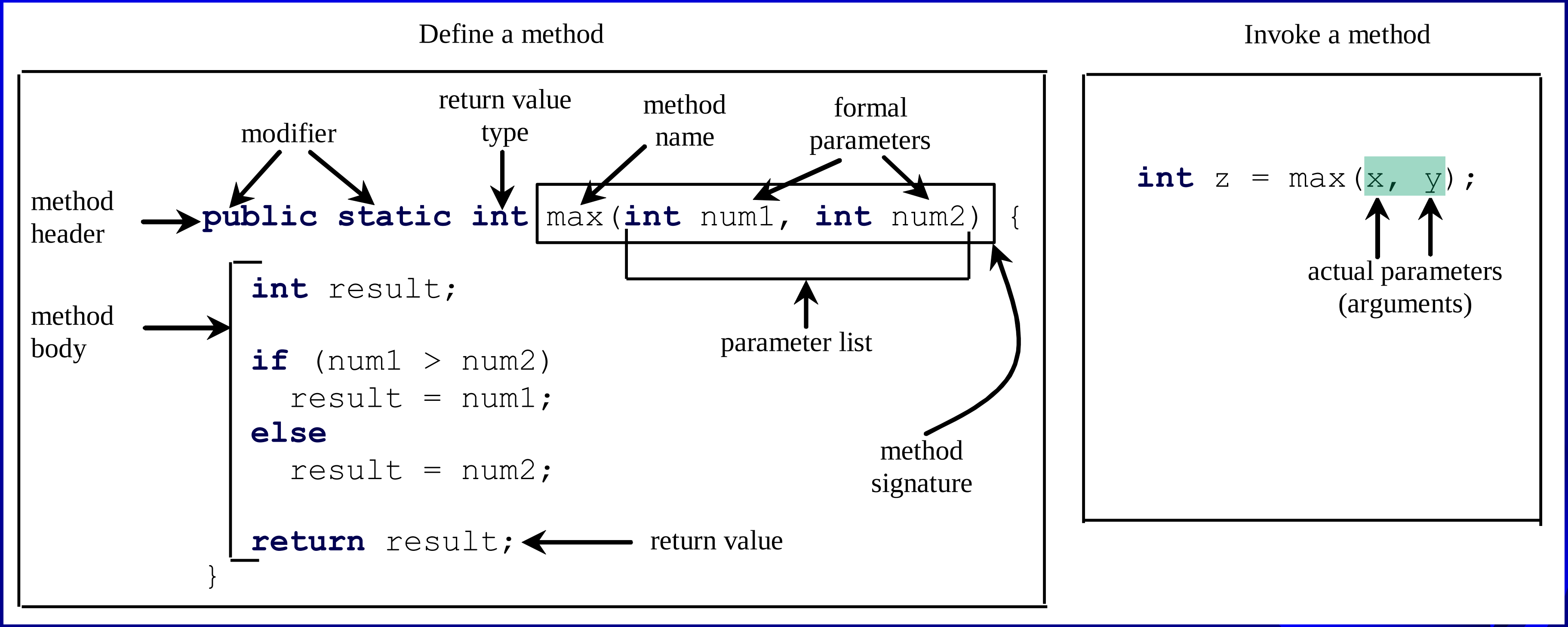


```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

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

实际参数

当调用一个方法时，需要传值给参数，这个传入的值就是实际参数，简称实参。

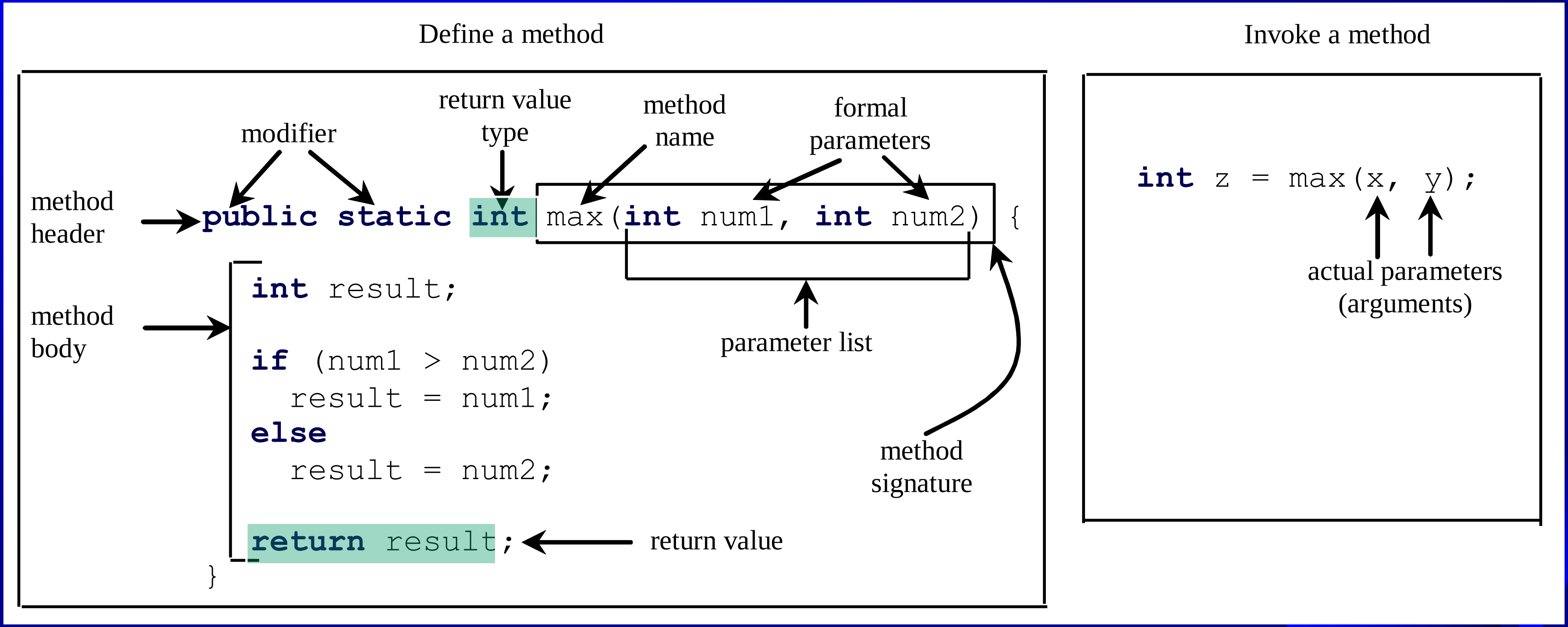


```
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

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

返回值

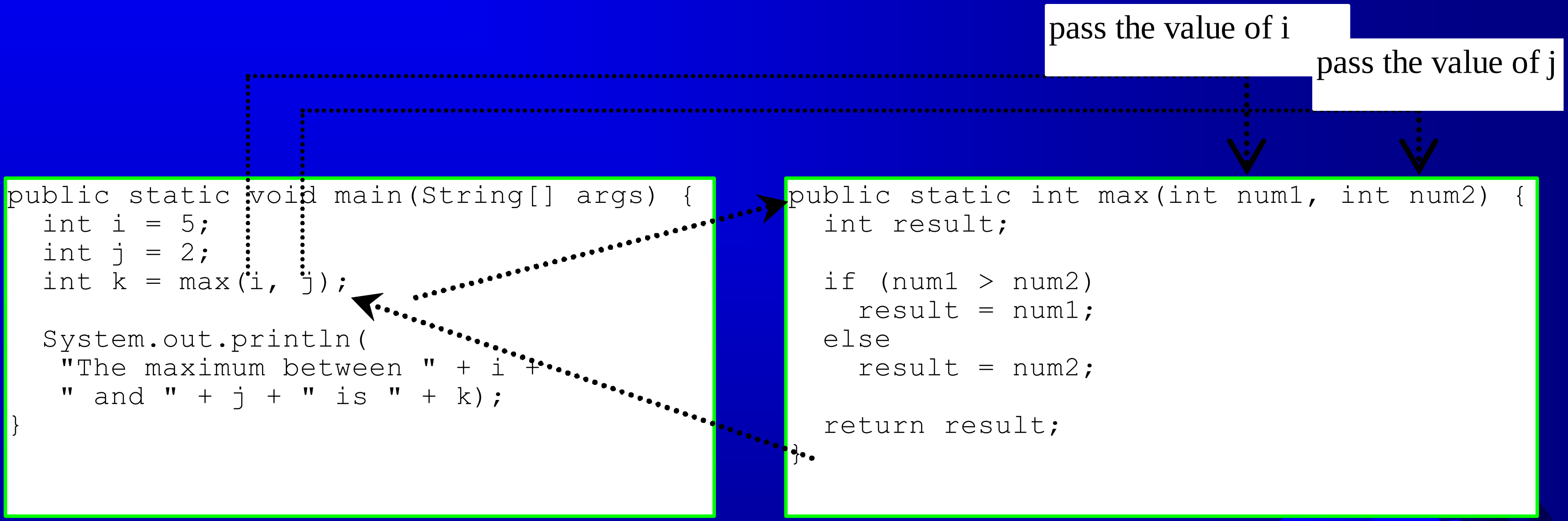
方法可以返回一个值，这个值是有类型的。如果没有什么值可以返回，可以不设置返回值，直接用void关键字。例如main函数就没有返回值。



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

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

  return result;
}
```

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

方法调用的例子



```
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;
}
```

单步执行一下

i 为 5

```java
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);
}
```

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

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

  return result;
}
```

单步执行一下

j 为 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;
}
```

单步执行一下

调用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;
}
```

单步执行一下

调用 max(i, j)
i 值传给 num1, j 值传给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;
}
```

单步执行一下

声明变量 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;
}
```

单步执行一下

(num1 > num2) 为 true, 因为 num1 为 5， num2 为 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;
}
```

单步执行一下

result 为 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;
}
```

单步执行一下

返回 result, 值为 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;
}
```

单步执行一下

从 max(i, j) 返回，并把返回值给 k

```
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;
}
```

单步执行一下

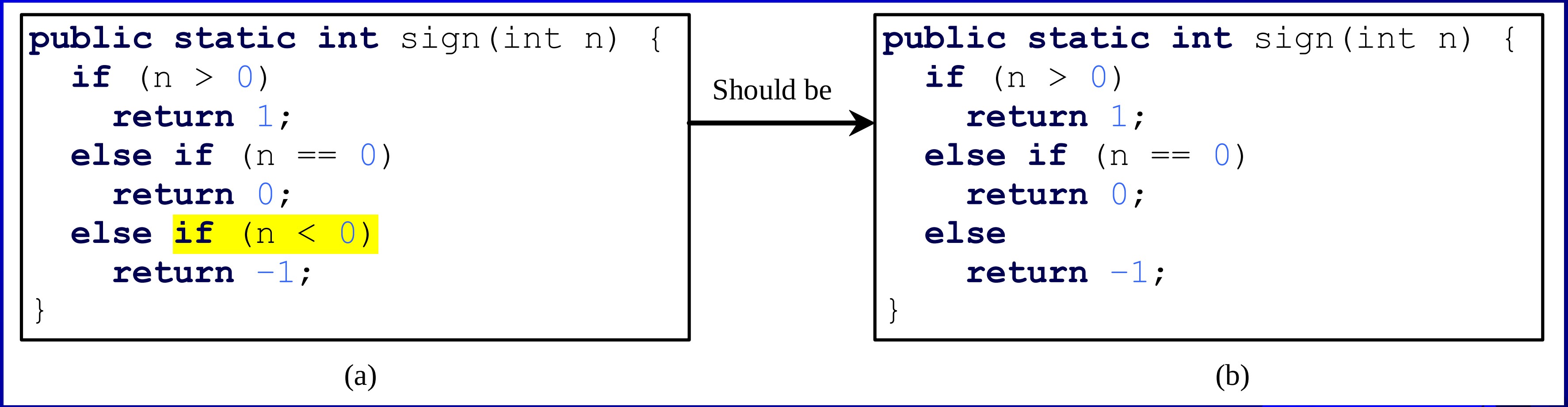
输出结果

```
public static void main(String      gs) {
  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;
}
```

注意

有返回值的方法，一定要有 return 语句，并且确保任何一个分支都能遇到 return 语句。下图 (a) 是逻辑正确的，不过Java会报编译错误，原因是Java认为万一所有if判断都不成立，程序就没有机会遇到 return 。



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

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

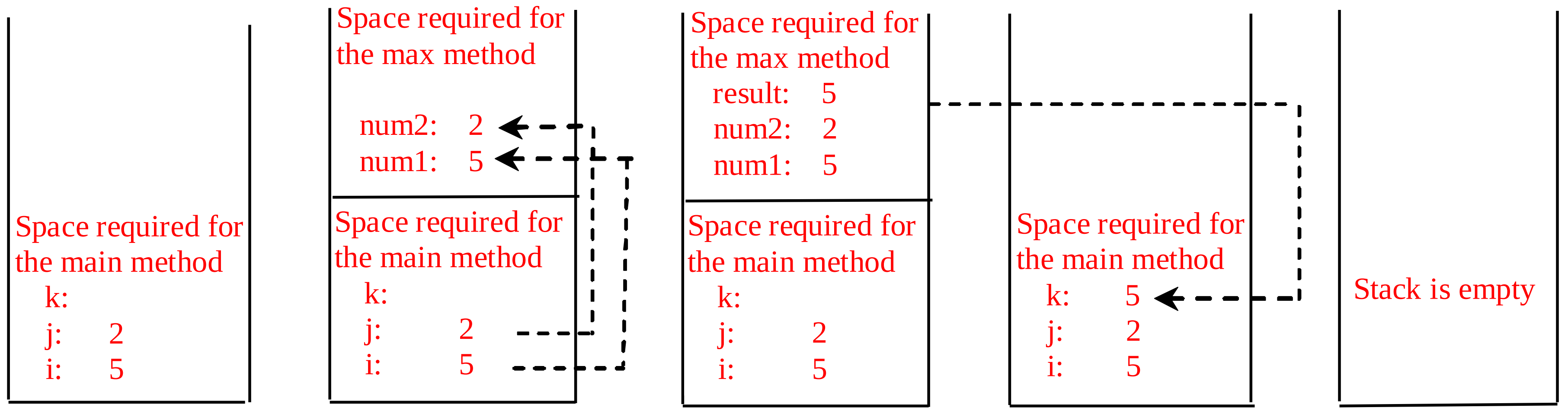
(a) (b)

为了改正这个问题，可以删除最后一个判断 *if (n < 0)*，改成(b)那个样子，就可以确保任何分支都有return。

从其它类调用TestMax类的方法

- 方法的好处是可以重用。例如刚才的 max 方法，除了TestMax类内部自己调用，你也可以在TestMax 这个类的外部调用。
- 在类的外部，调用类中的方法，Java的调用格式是：ClassName.methodName
- 具体到刚才的max方法，格式就是：TestMax.max

为了记录方法间的调用顺序，以便能够在调用结束后正确返回调用处，Java采用**栈**这种数据结构来记录调用信息。


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

(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.

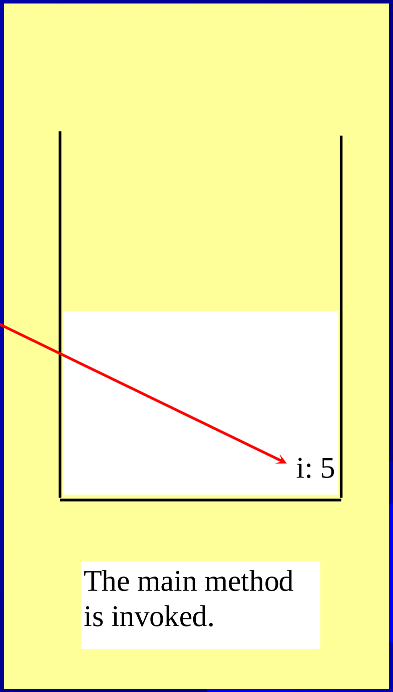
(e) The main method is finished.

调用栈

声明 i 并初始化

```
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;
}
```

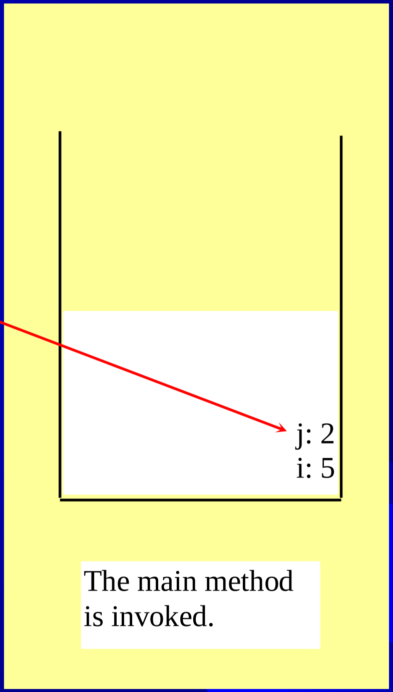


调用栈

声明 j 并初始化

```
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;
}
```

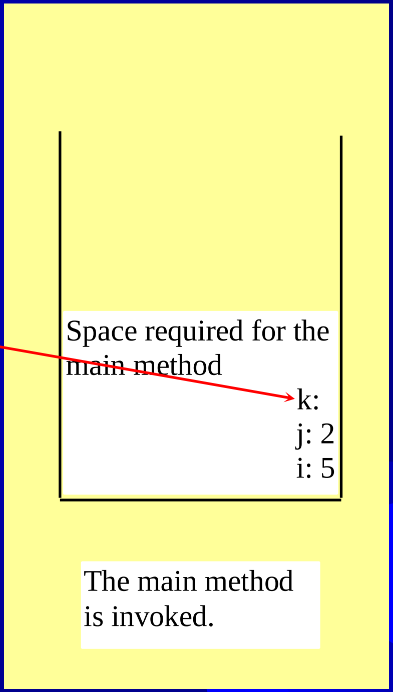


调用栈

声明 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;
}
```

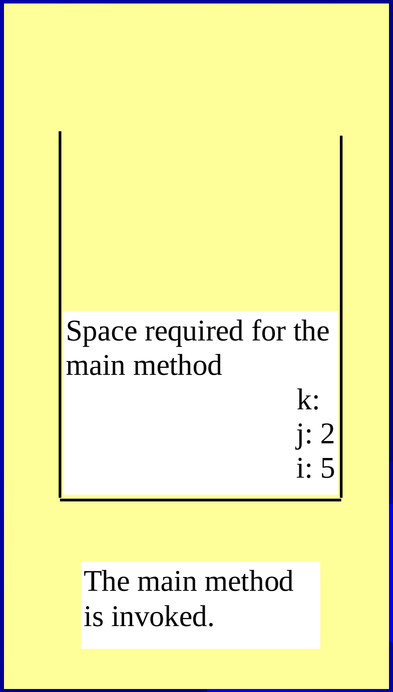

Space required for the main method
k:
j: 2
i: 5
The main method is invoked.

调用栈

调用 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;
}
```



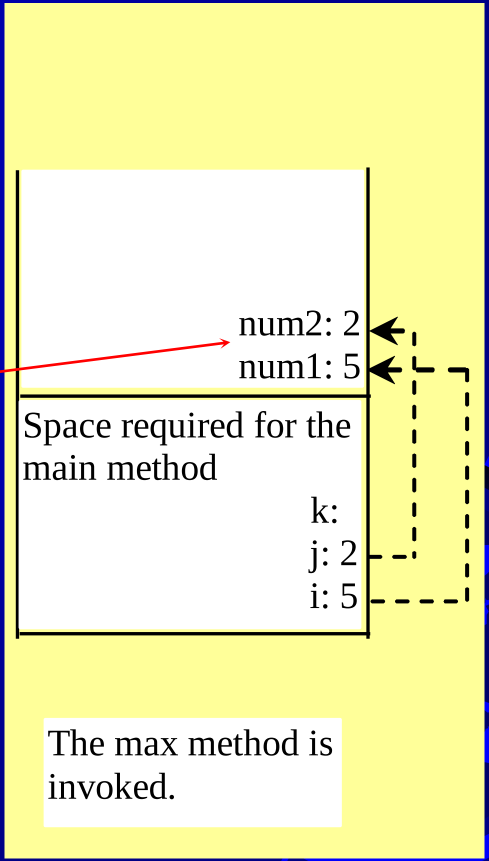
Space required for the main method
k:
j: 2
i: 5
The main method is invoked.

调用栈

将 i 和j 的值传给 num1和 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;
}
```



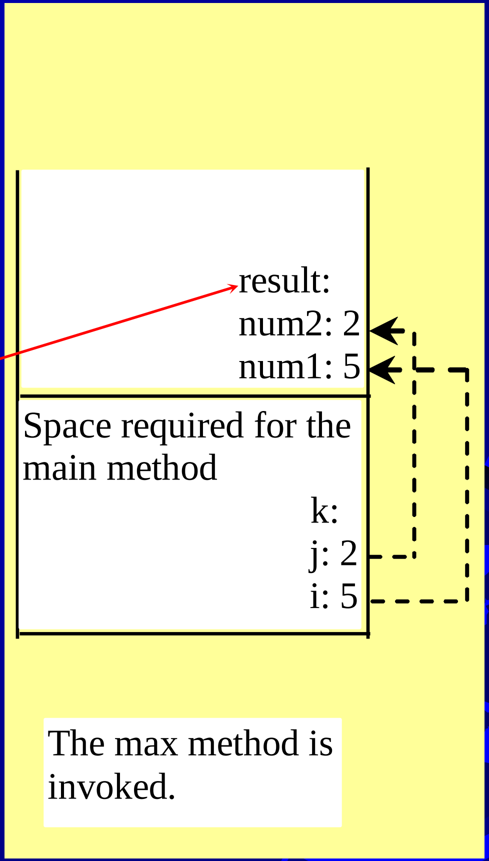
num2: 2
num1: 5
Space required for the main method
k:
j: 2
i: 5
The max method is invoked.

调用栈

```
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;
}
```

声明 result



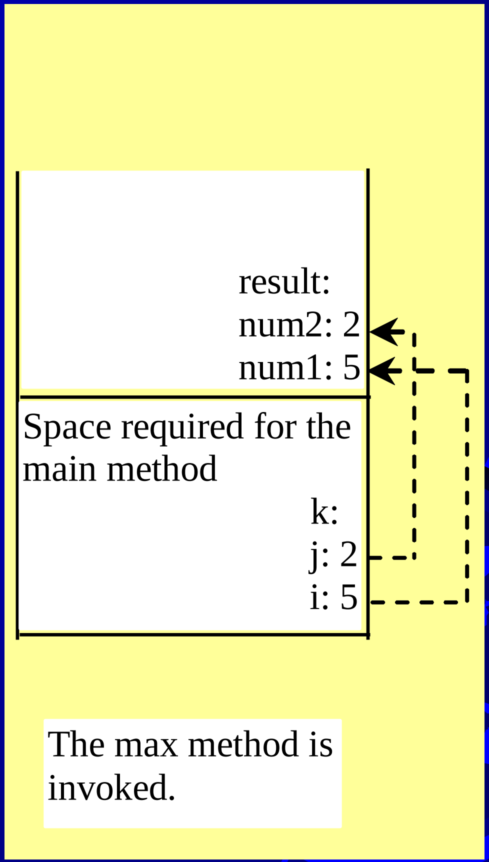
result:
num2: 2
num1: 5
Space required for the main method
k:
j: 2
i: 5
The max method is invoked.

调用栈

(num1 > num2) 为 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;
}
```



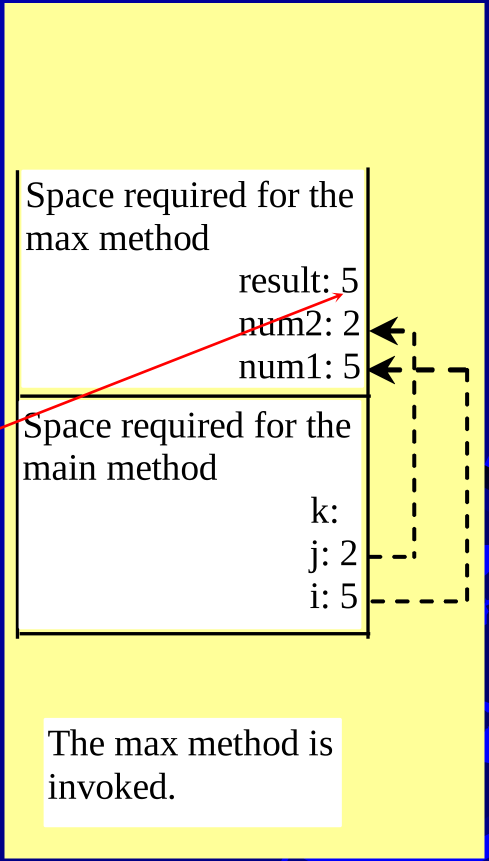
result:
num2: 2
num1: 5
Space required for the main method
k:
j: 2
i: 5
The max method is invoked.

调用栈

num1赋值给 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

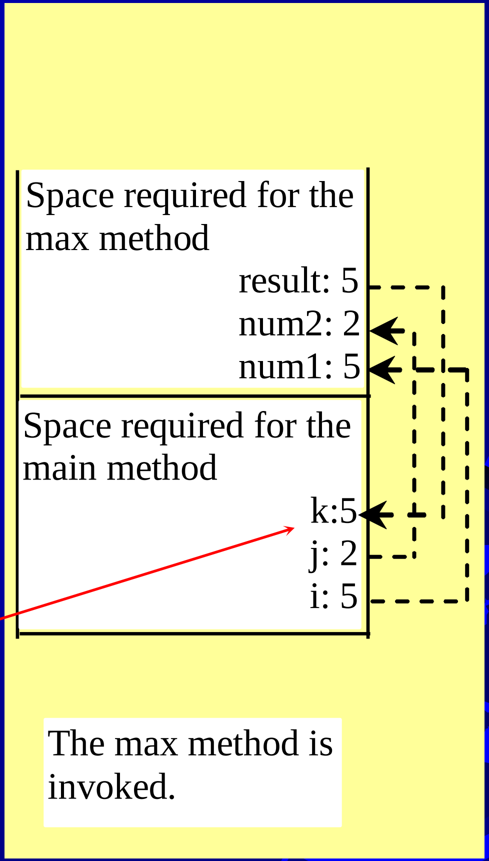
The max method is invoked.

调用栈

返回 result 的值并赋值给 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

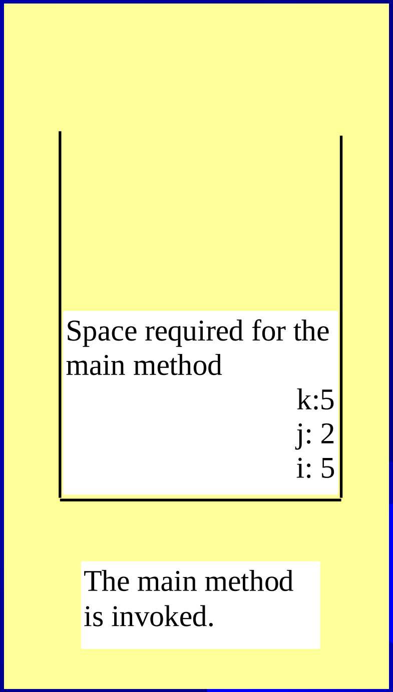
The max method is invoked.

调用栈

打印结果。注意max函数已经退栈。

```
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:5
j: 2
i: 5
The main method is invoked.

参数传递

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

上面这个函数，如果这样调用：

  nPrintln(“Welcome to Java”, 5);

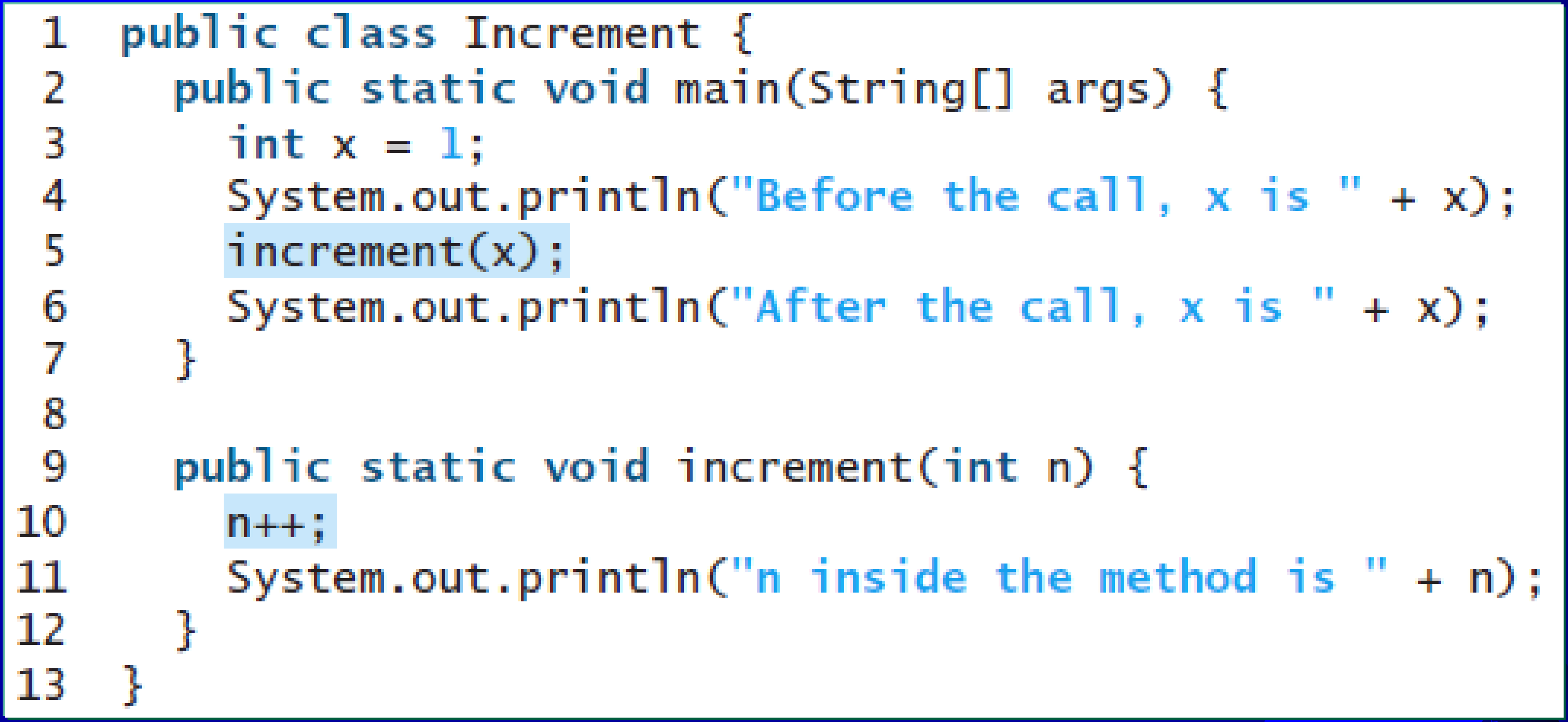
输出是什么?

如果这样调用：

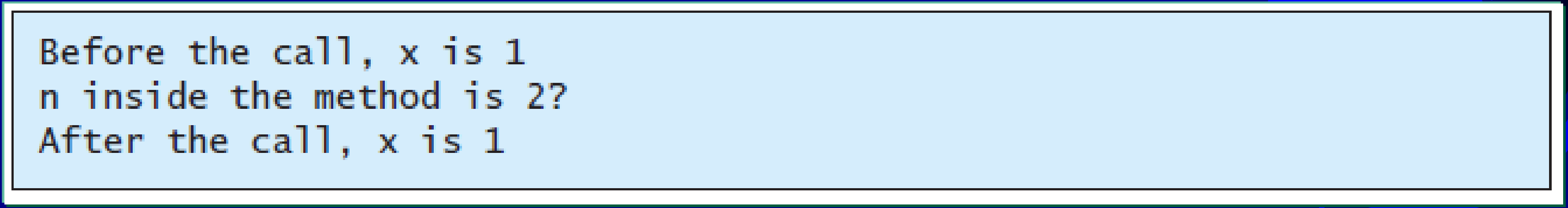
  nPrintln(“Computer Science”, 15);

输出是什么?

传值调用的例子

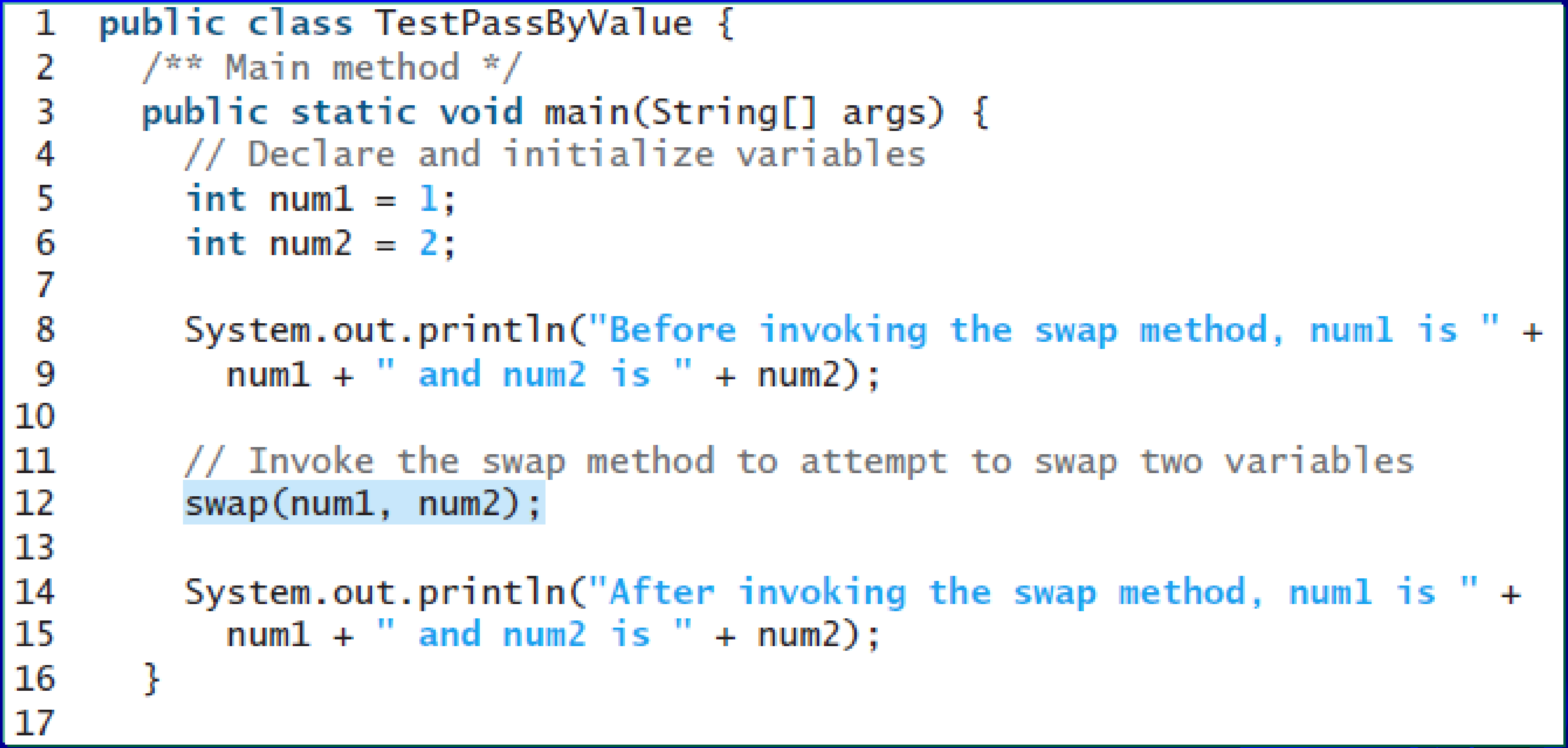


```java
public class Increment {
  public static void main(String[] args) {
    int x = 1;
    System.out.println("Before the call, x is " + x);
    increment(x);
    System.out.println("After the call, x is " + x);
  }

  public static void increment(int n) {
    n++;
    System.out.println("n inside the method is " + n);
  }
}
```



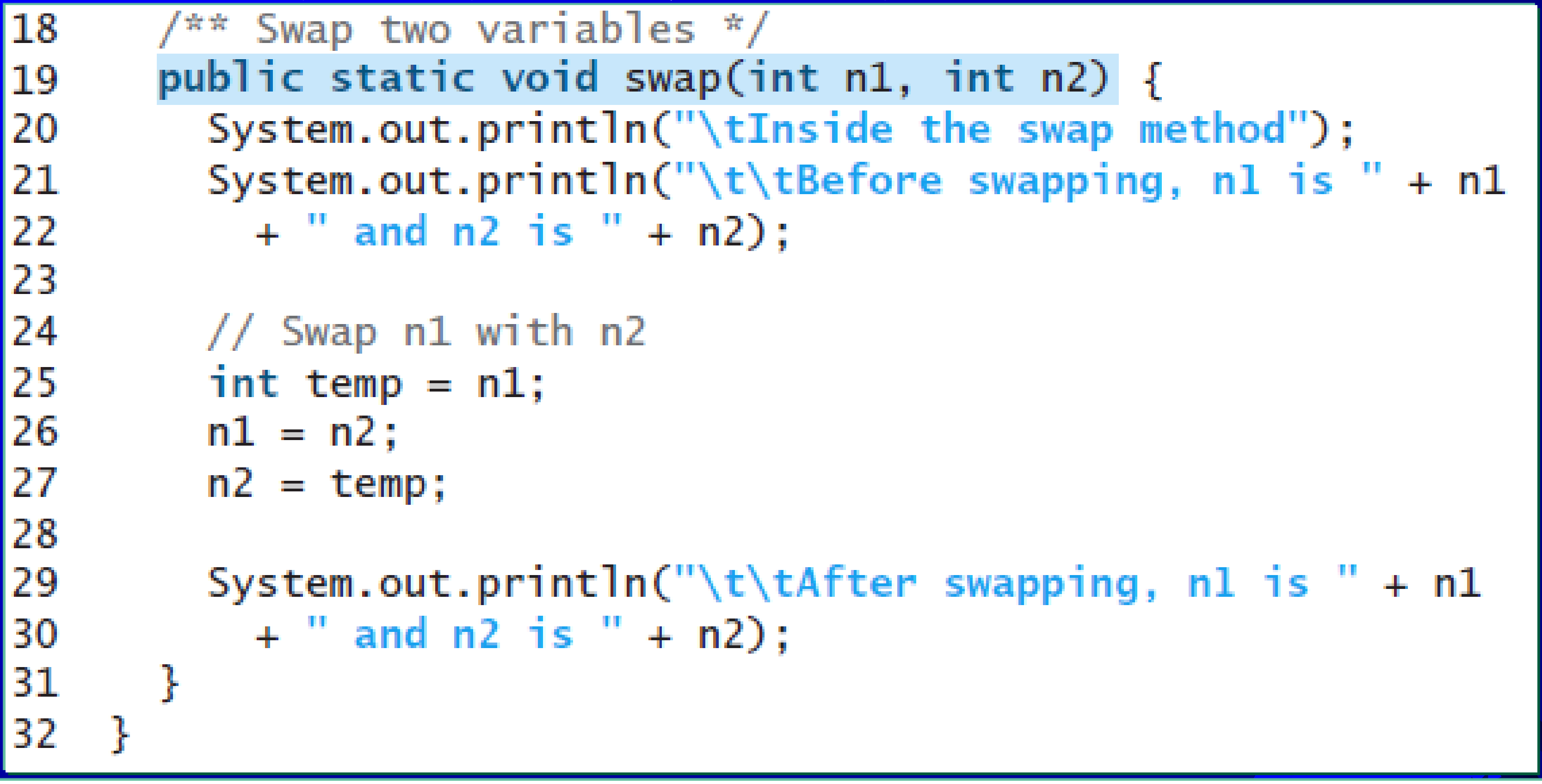
```
Before the call, x is 1
n inside the method is 2?
After the call, x is 1
```

另外一个例子—1/2

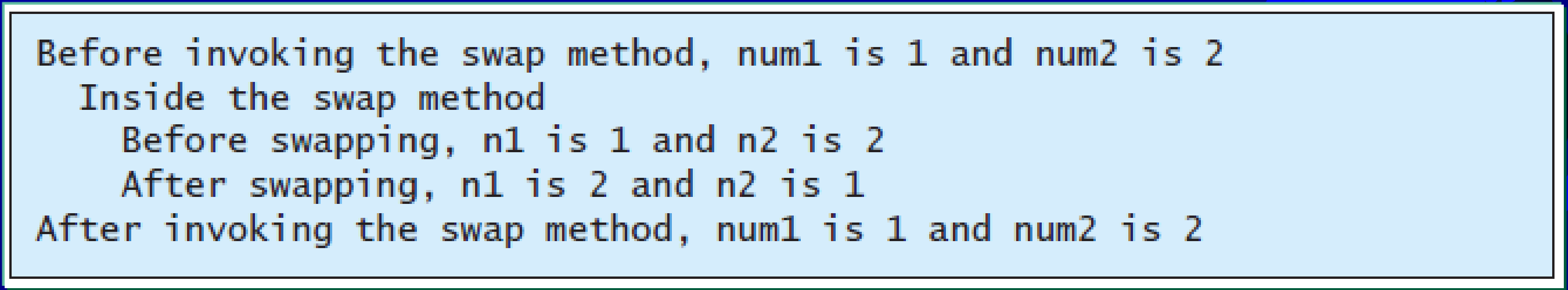


```java
public class TestPassByValue {
  /** Main method */
  public static void main(String[] args) {
    // Declare and initialize variables
    int num1 = 1;
    int num2 = 2;

    System.out.println("Before invoking the swap method, num1 is " +
      num1 + " and num2 is " + num2);

    // Invoke the swap method to attempt to swap two variables
    swap(num1, num2);

    System.out.println("After invoking the swap method, num1 is " +
      num1 + " and num2 is " + num2);
  }

```

另外一个例子—2/2

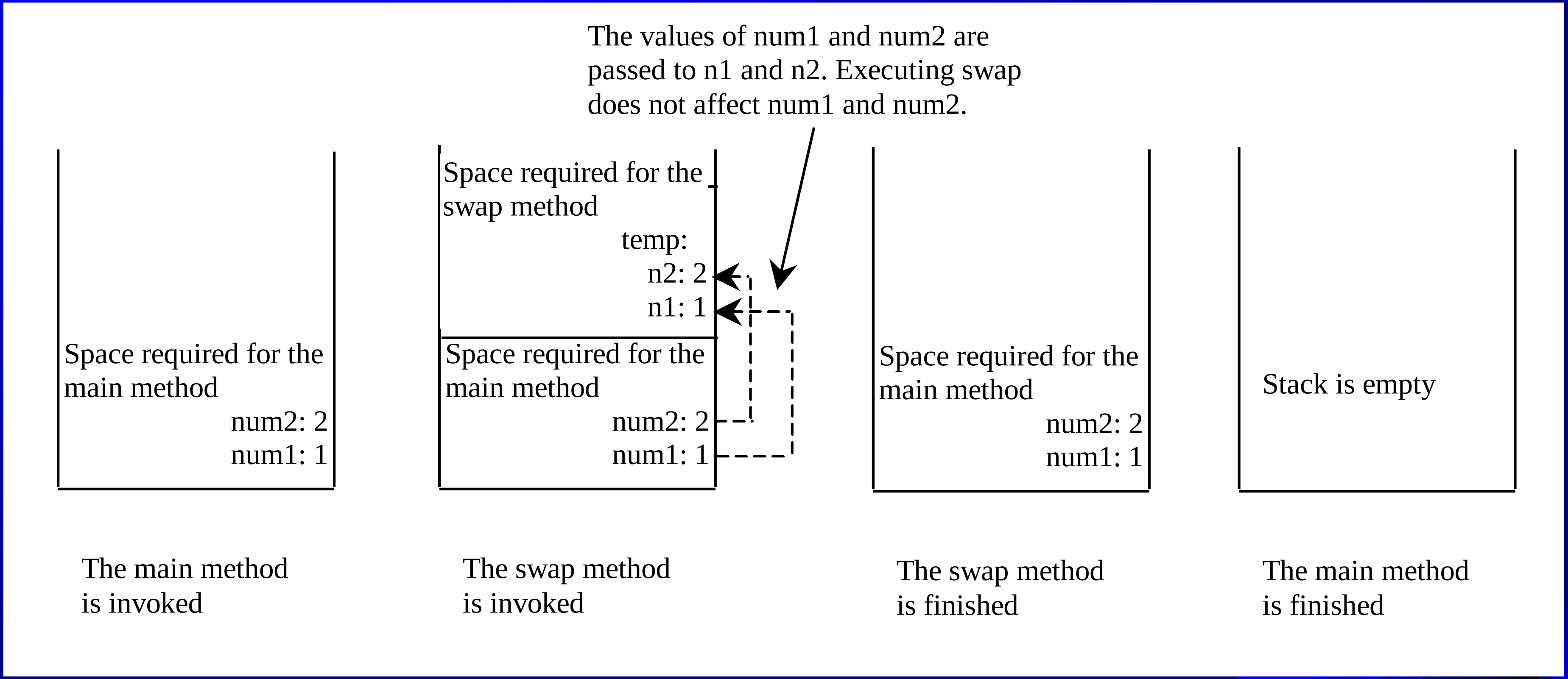


```java
    /** Swap two variables */
    public static void swap(int n1, int n2) {
      System.out.println("\tInside the swap method");
      System.out.println("\t\tBefore swapping, n1 is " + n1
        + " and n2 is " + n2);

      // Swap n1 with n2
      int temp = n1;
      n1 = n2;
      n2 = temp;

      System.out.println("\t\tAfter swapping, n1 is " + n1
        + " and n2 is " + n2);
    }
  }
```



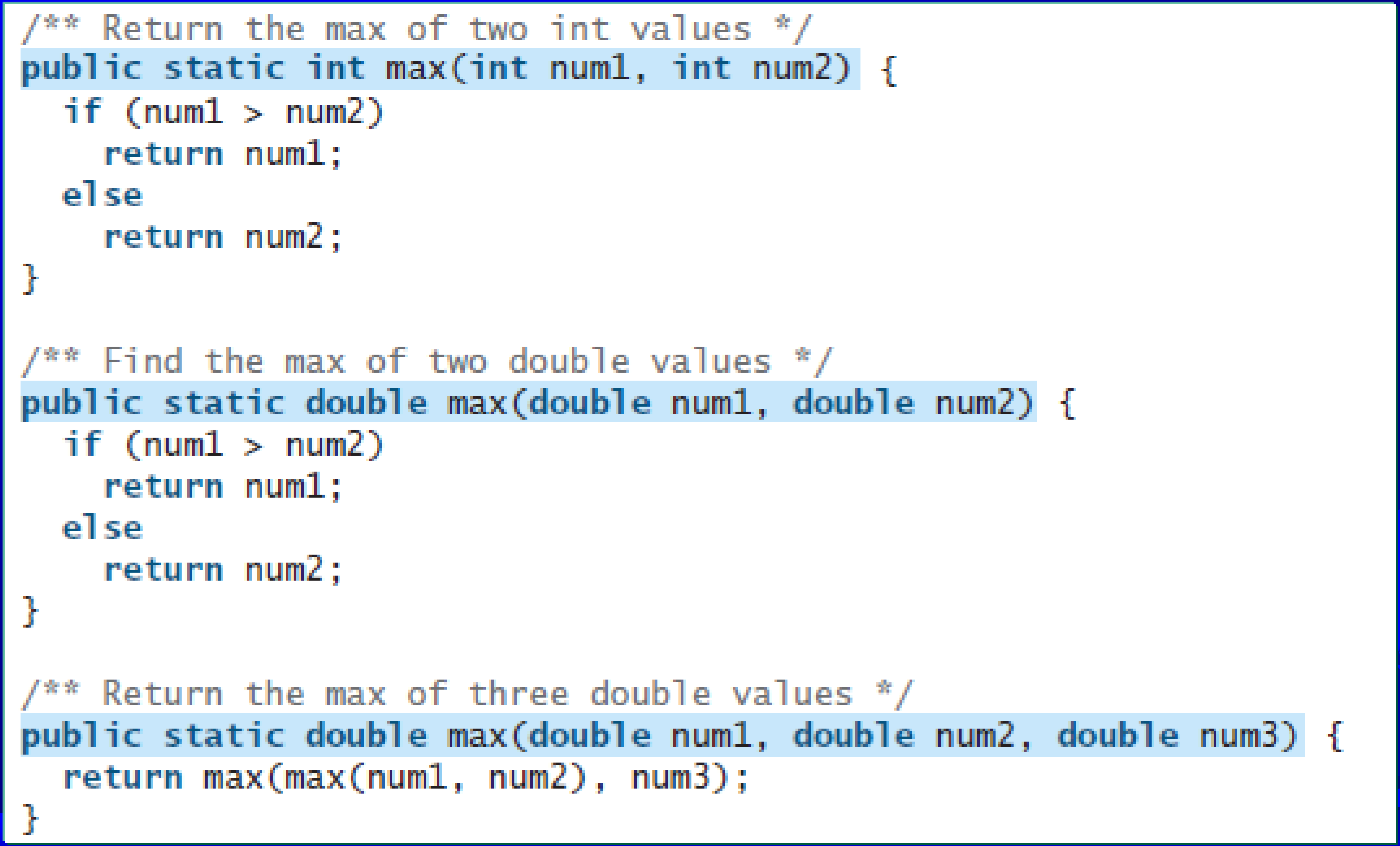
```
Before invoking the swap method, num1 is 1 and num2 is 2
  Inside the swap method
    Before swapping, n1 is 1 and n2 is 2
    After swapping, n1 is 2 and n2 is 1
After invoking the swap method, num1 is 1 and num2 is 2
```

传值调用图解


The values of num1 and num2 are passed to n1 and n2. Executing swap does not affect num1 and num2.
Space required for the main method
num2: 2
num1: 1
The main method is invoked
Space required for the swap method
temp:
n2: 2
n1: 1
Space required for the main method
num2: 2
num1: 1
The swap method is invoked
Space required for the main method
num2: 2
num1: 1
The swap method is finished
Stack is empty
The main method is finished

方法重载

方法重载指的是同一个类拥有多个相同名字的方法，例如：



```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);
}
```

方法重载的要素

- 方法名一定相同
- 参数列表一定不同，即至少满足以下一项：
  - 参数个数不同
  - 参数类型不同
- 返回值类型不能作为重载标识

歧义调用（Ambiguous Invocation）

由于Java会进行隐式的参数类型转换，因此当方法重载时，可能会出现多个方法都符合调用的实参的情况，此时Java无法决定被调用的方法是哪一个，于是会报编译出错。这种情况叫做歧义调用。

歧义调用的例子

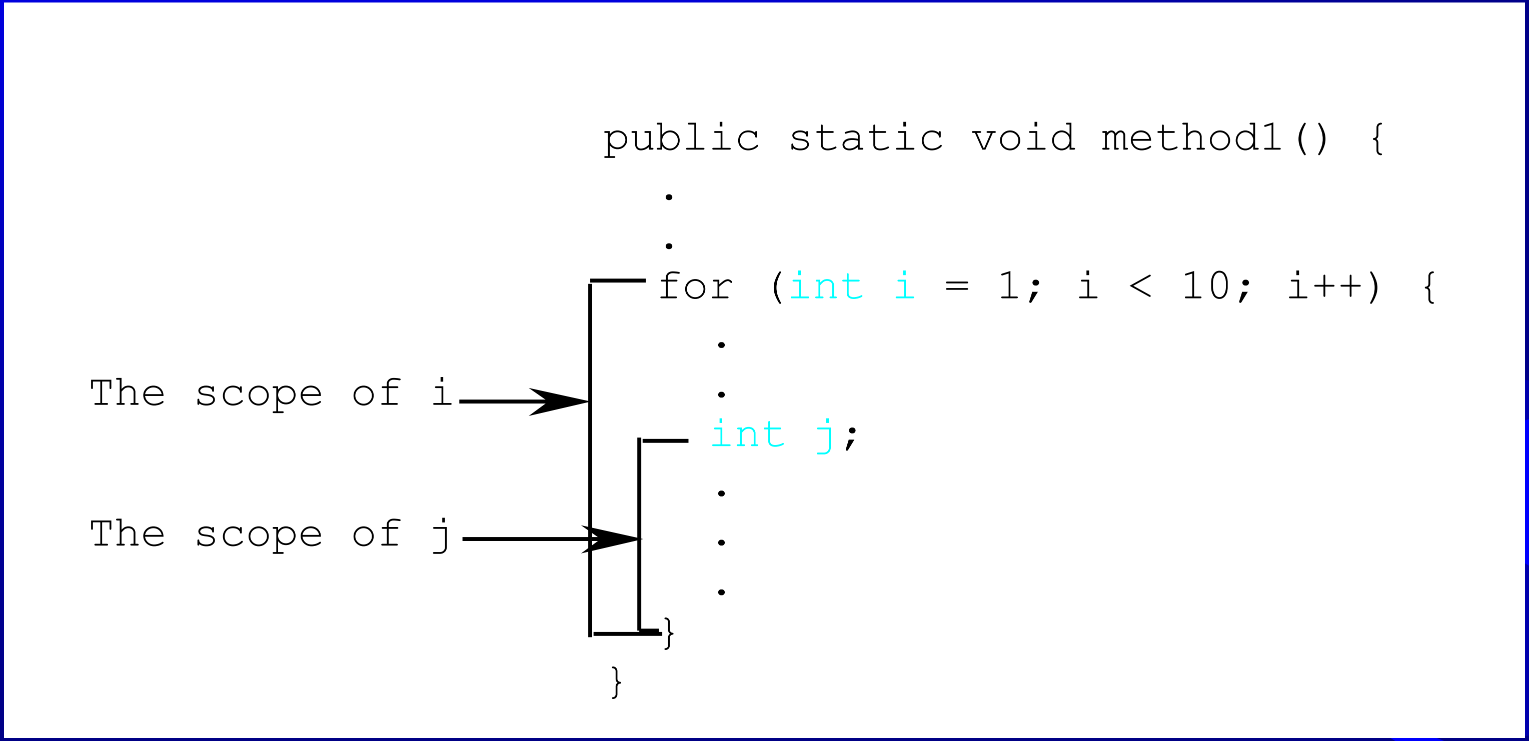
```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

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

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

变量的作用范围

- 局部变量: 在方法内部定义的变量
- 作用范围: 可以访问到该变量的代码部分
- 局部变量的作用范围是从声明的地方开始，直到它所在的语句块结束（也就是包含它的最近的那个右括号’}’）
- 局部变量需要先声明后使用。

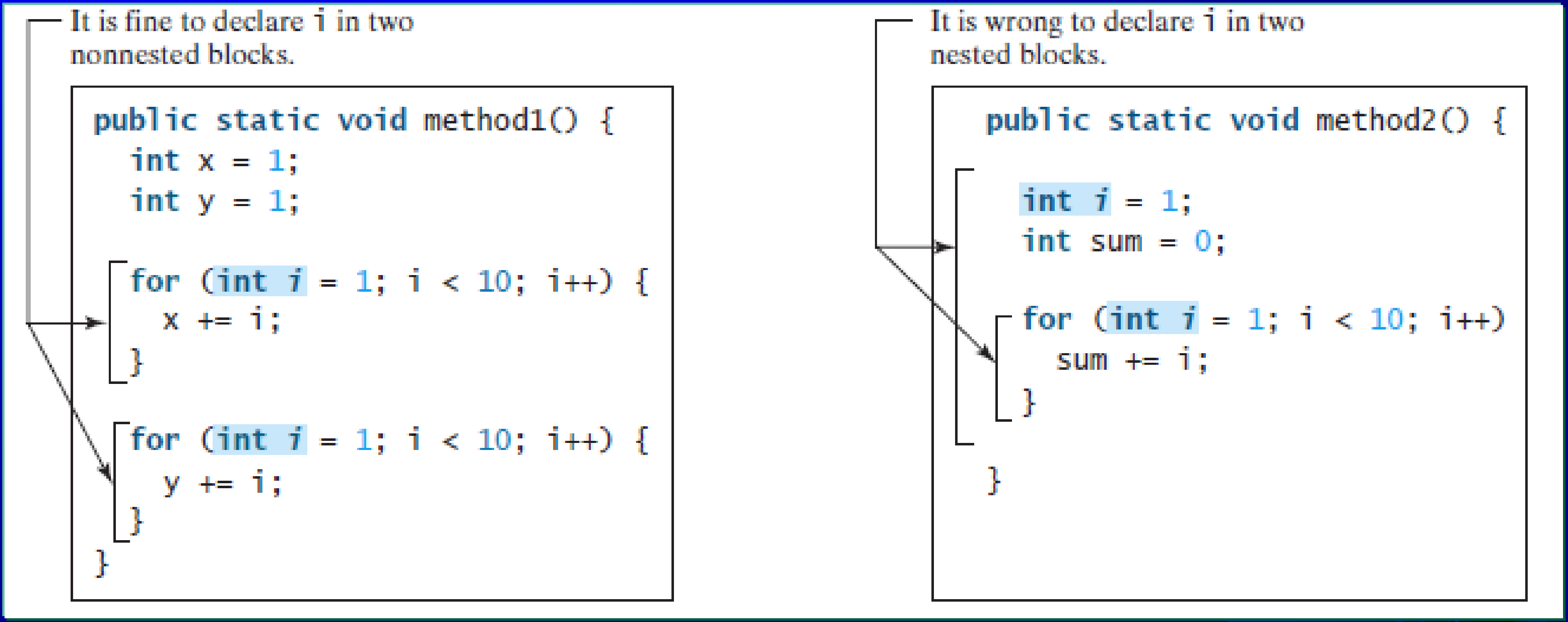
for语句的循环变量

for语句可以定义变量，此时该变量的作用范围仅仅局限于for内部，如下面的i；for语句内部也可以定义变量，作用范围也在for内部，如下面的j



```
public static void method1() {
   .
   .
   for (int i = 1; i < 10; i++) {
      .
      .
      int j;
      .
      .
      .
   }
}
```

具有嵌套关系的语句块中，不允许定义同名变量。注意以下两段代码的不同之处，左边正确，右边错误。


It is fine to declare i in two nonnested blocks.
It is wrong to declare i in two nested blocks.

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

```
public static void method2() {

    int i = 1;
    int sum = 0;

    for (int i = 1; i < 10; i++)
      sum += i;
    }

}
```

再看一个错误的例子

```
// With errors
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0; //error! Duplicate local variable x
    x += i;
  }
}
```

方法的好处

- 可重用。
- 隐藏实现细节。
- 降低编程复杂性（因为模块化了）。

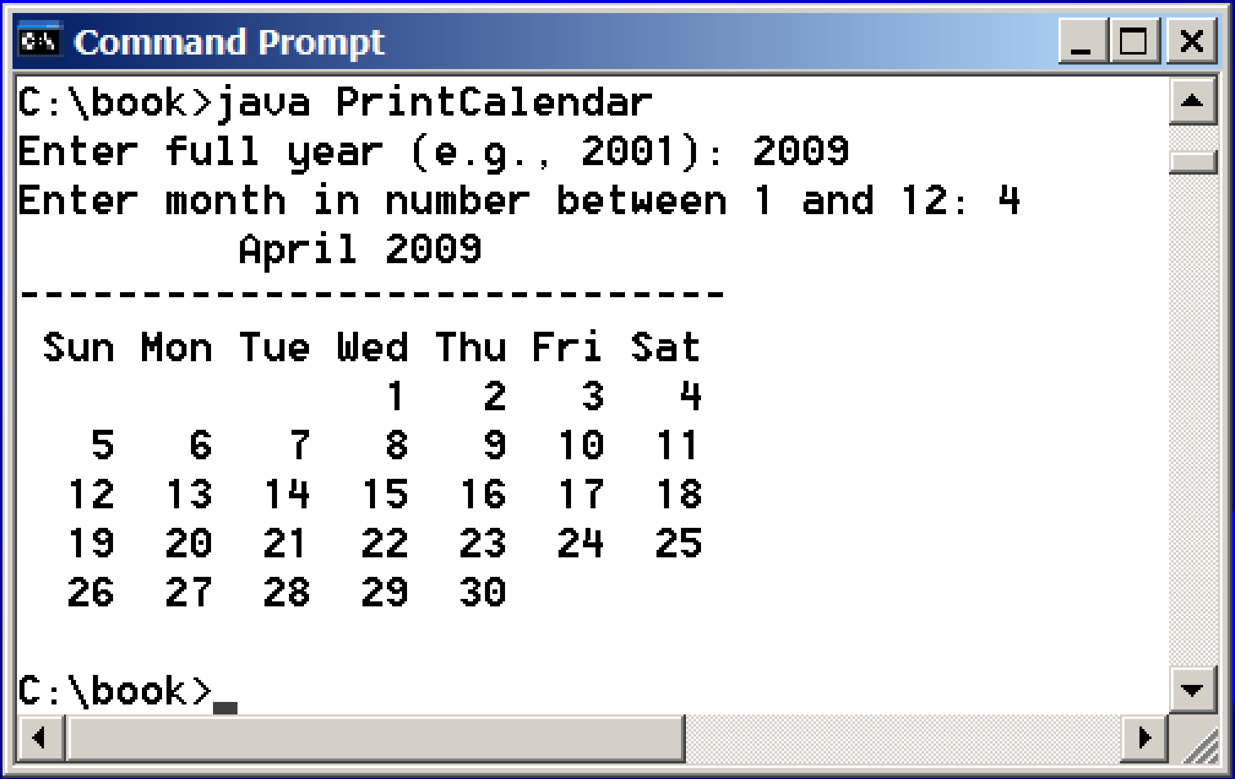
逐步细化的编程方法

编写大型程序的时候，分治法（divide and conquer）或者逐步细化（*stepwise refinement*）是最常用的做法。

下面用一个例题来阐述这种编程方法：

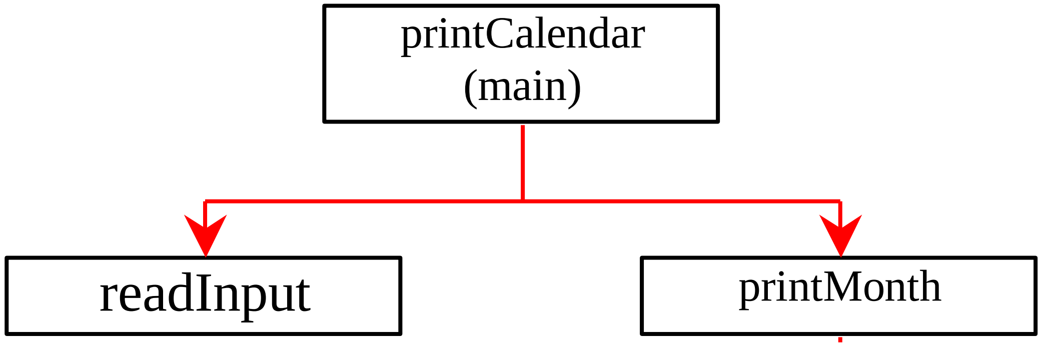
打印一个日历

编程实现日历打印。输入年和月份，输出该月份的日历。程序运行效果如图：

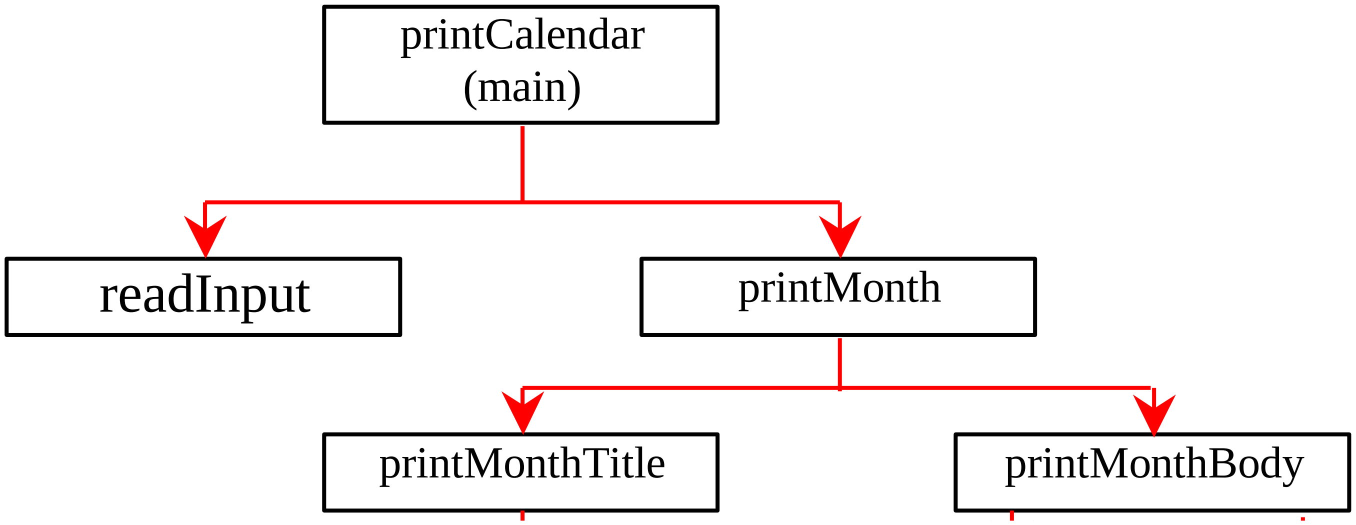


```
C:\book>java PrintCalendar
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 4
         April 2009
-----------------------------
 Sun Mon Tue Wed Thu Fri Sat
                   1   2   3   4
   5   6   7   8   9  10  11
  12  13  14  15  16  17  18
  19  20  21  22  23  24  25
  26  27  28  29  30

C:\book>_
```

设计图

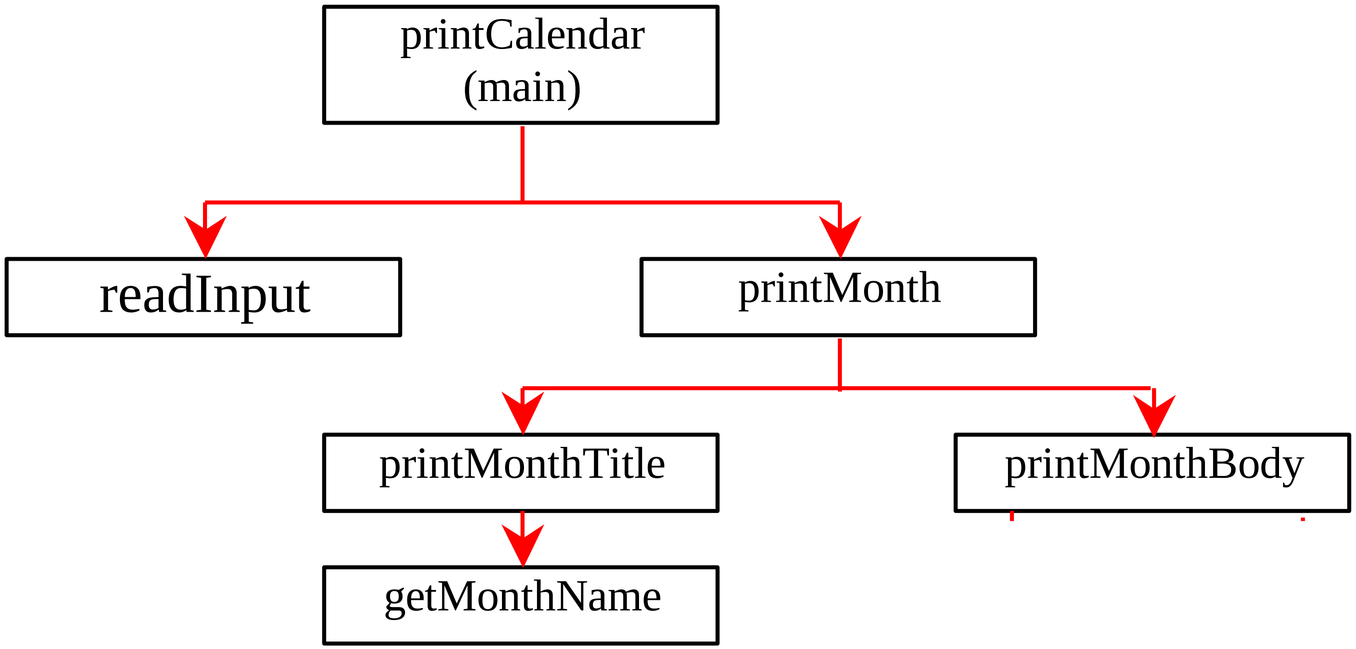


细化一点

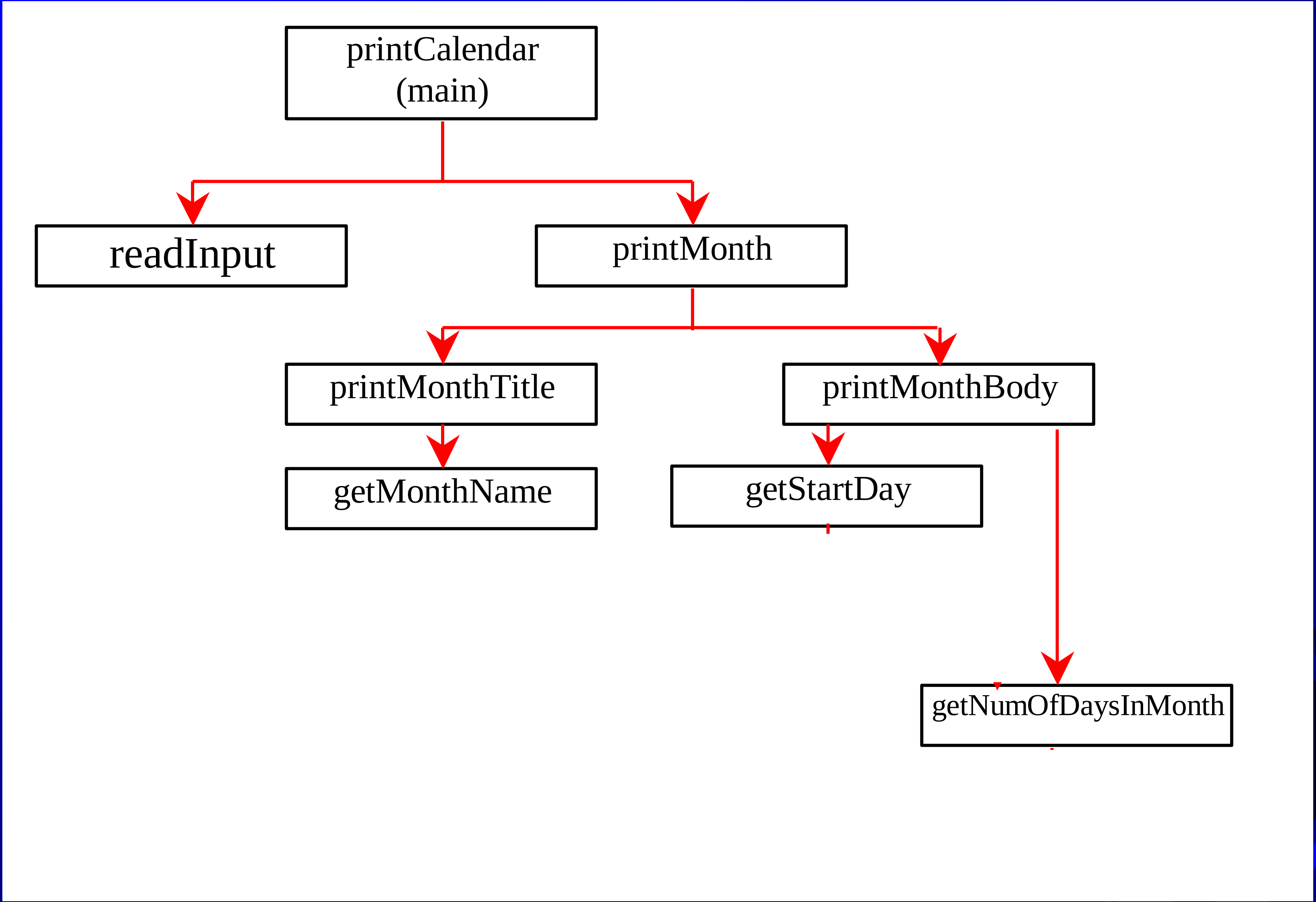


printCalendar (main)
readInput
printMonth
printMonthTitle
printMonthBody

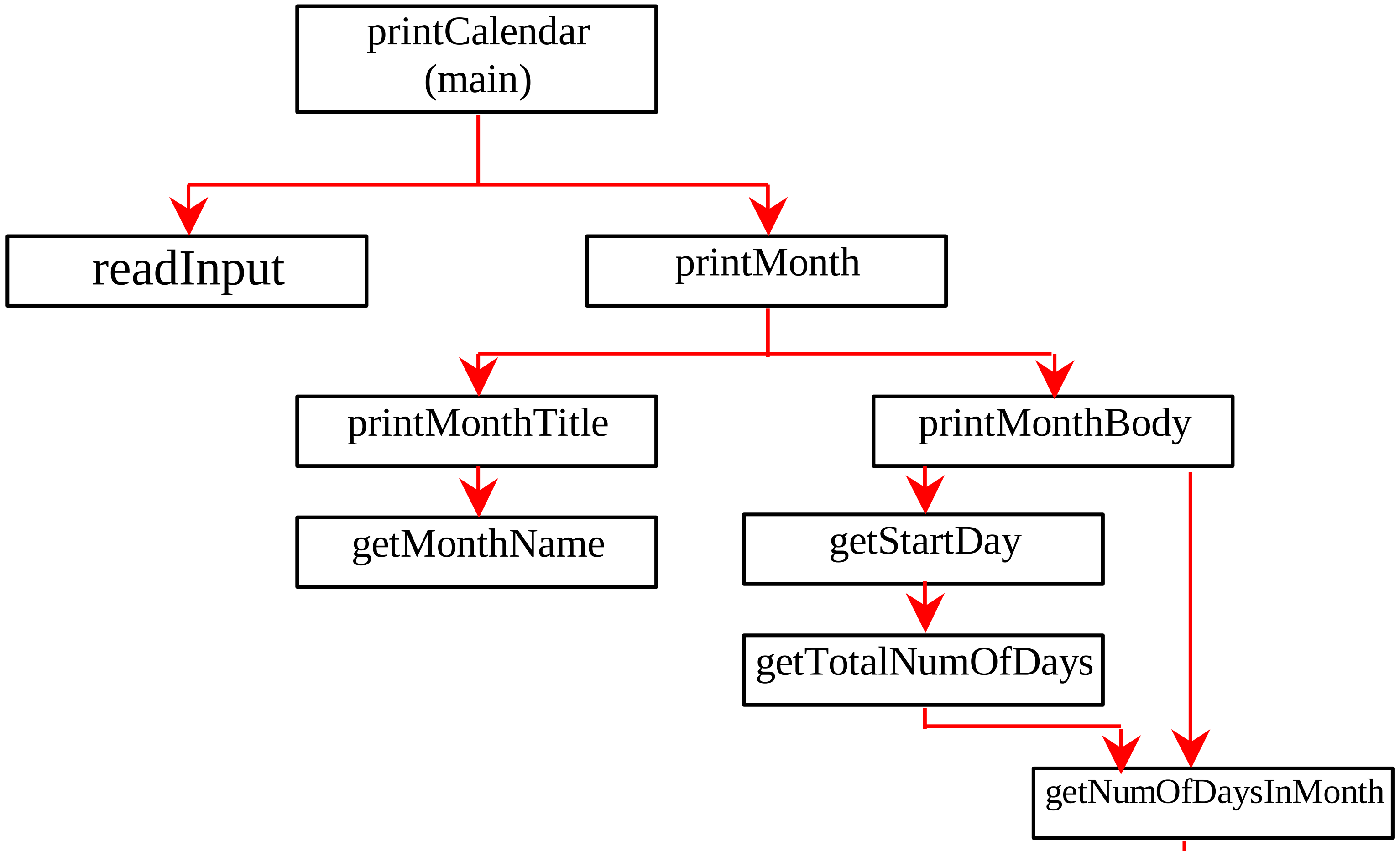
再细化一点


printCalendar
(main)
readInput
printMonth
printMonthTitle
printMonthBody
getMonthName

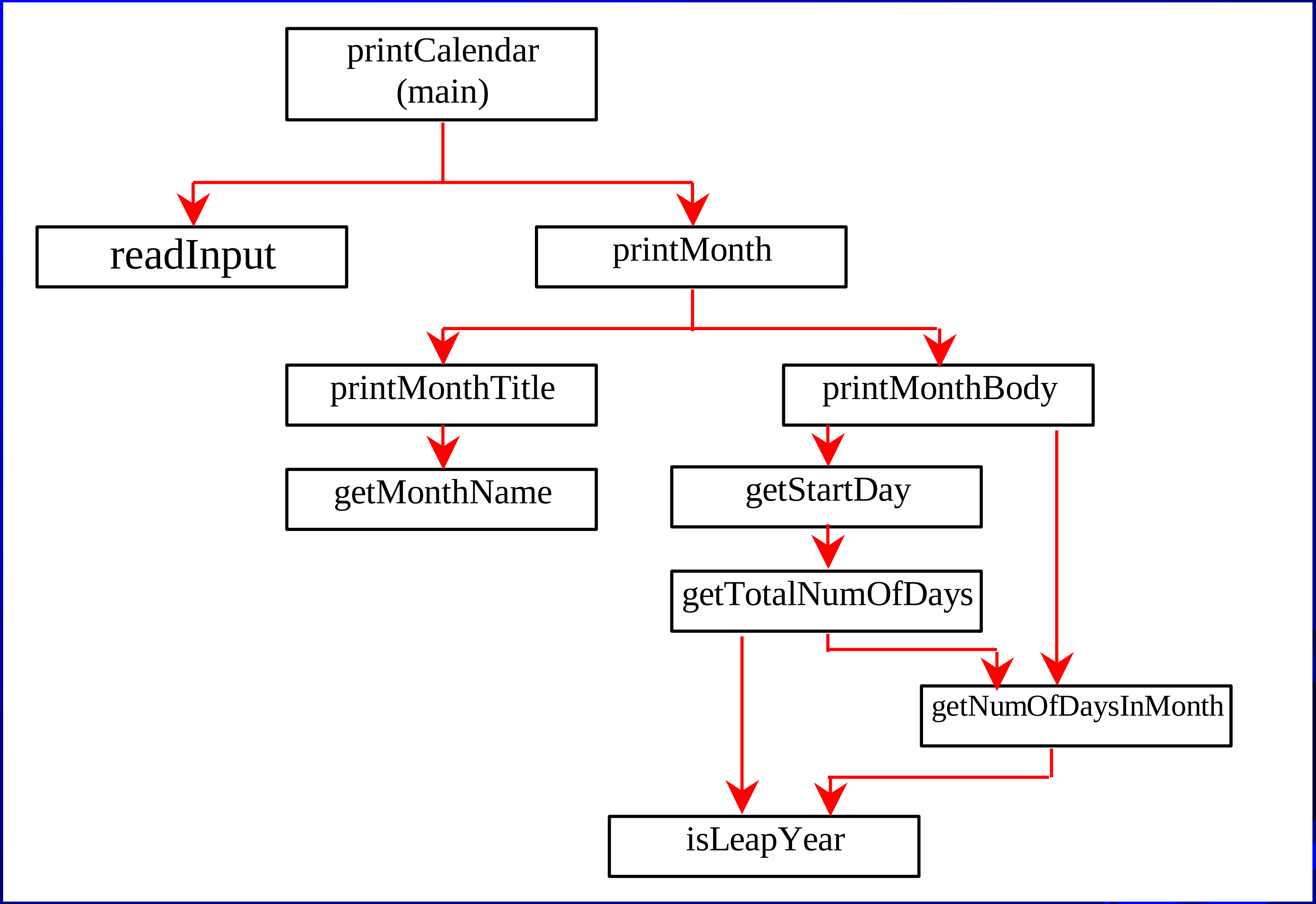
再再细化一点


printCalendar (main)
readInput
printMonth
printMonthTitle
printMonthBody
getMonthName
getStartDay
getNumOfDaysInMonth

再再再细化一点

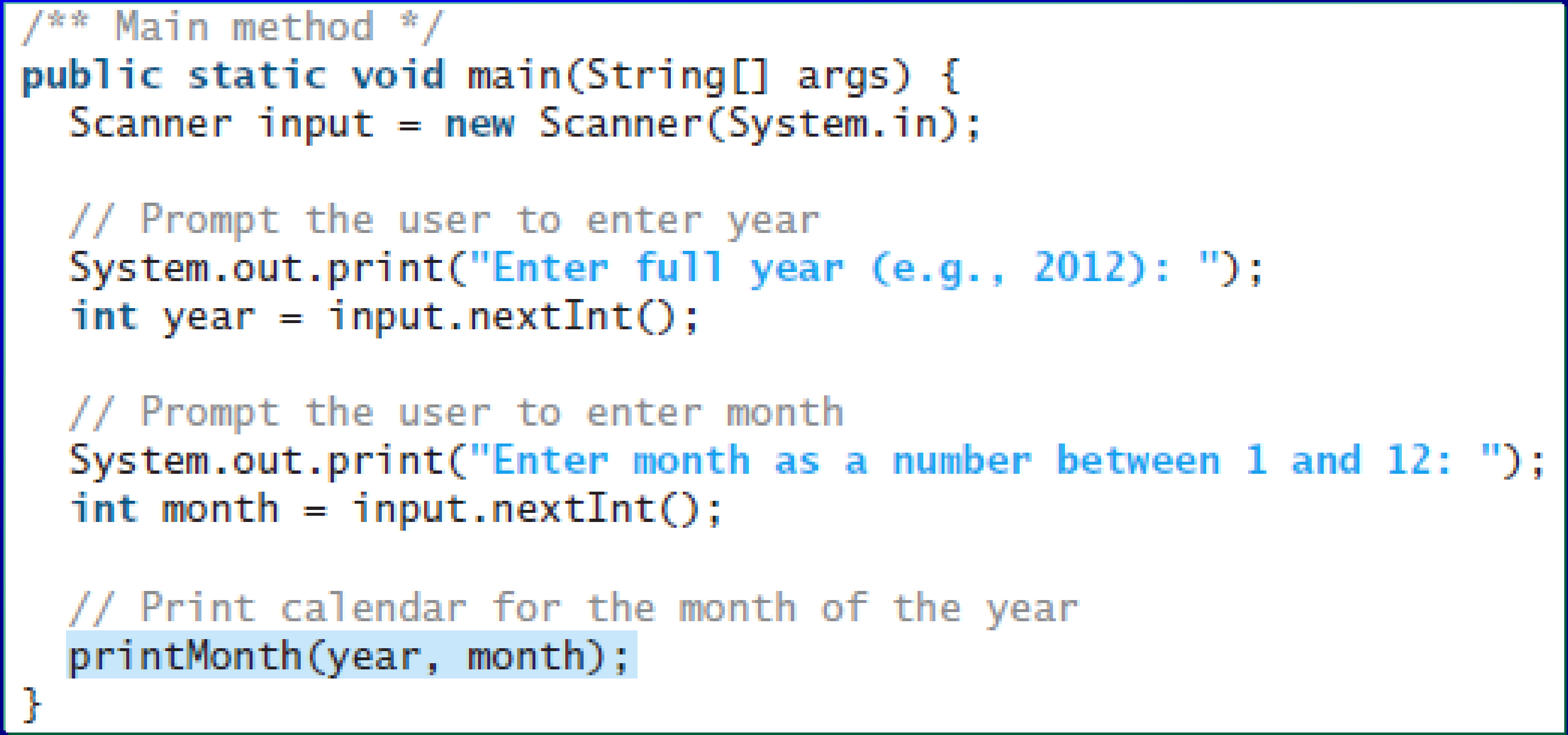

printCalendar (main)
readInput
printMonth
printMonthTitle
printMonthBody
getMonthName
getStartDay
getTotalNumOfDays
getNumOfDaysInMonth

最后一点细化


printCalendar (main)
readInput
printMonth
printMonthTitle
printMonthBody
getMonthName
getStartDay
getTotalNumOfDays
getNumOfDaysInMonth
isLeapYear

编码实现：自顶向下（ Top-down）

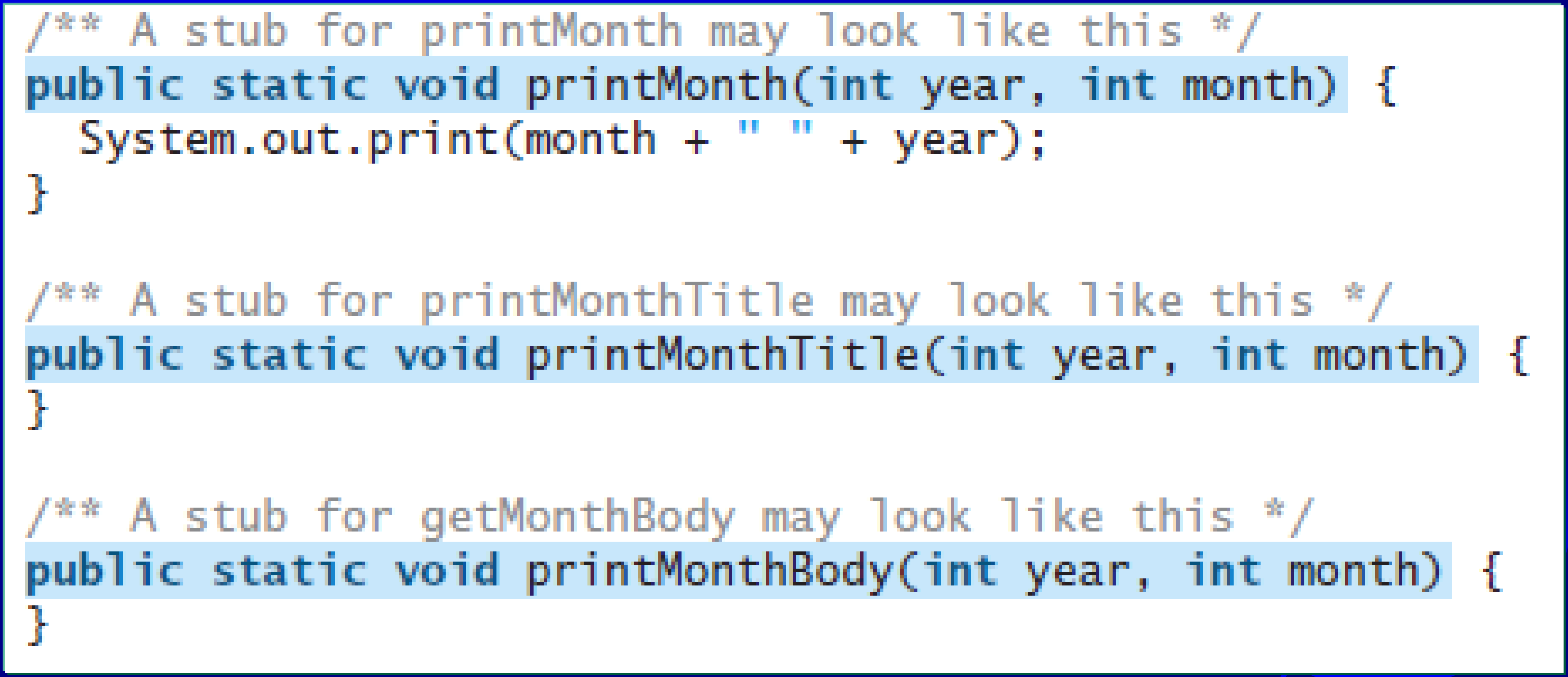
把刚才的设计图按照自顶向下的设计思路来实现，首先开始搭程序框架，下面是main函数的造型：



```java
/** Main method */
public static void main(String[] args) {
  Scanner input = new Scanner(System.in);

  // Prompt the user to enter year
  System.out.print("Enter full year (e.g., 2012): ");
  int year = input.nextInt();

  // Prompt the user to enter month
  System.out.print("Enter month as a number between 1 and 12: ");
  int month = input.nextInt();

  // Print calendar for the month of the year
  printMonth(year, month);
}
```

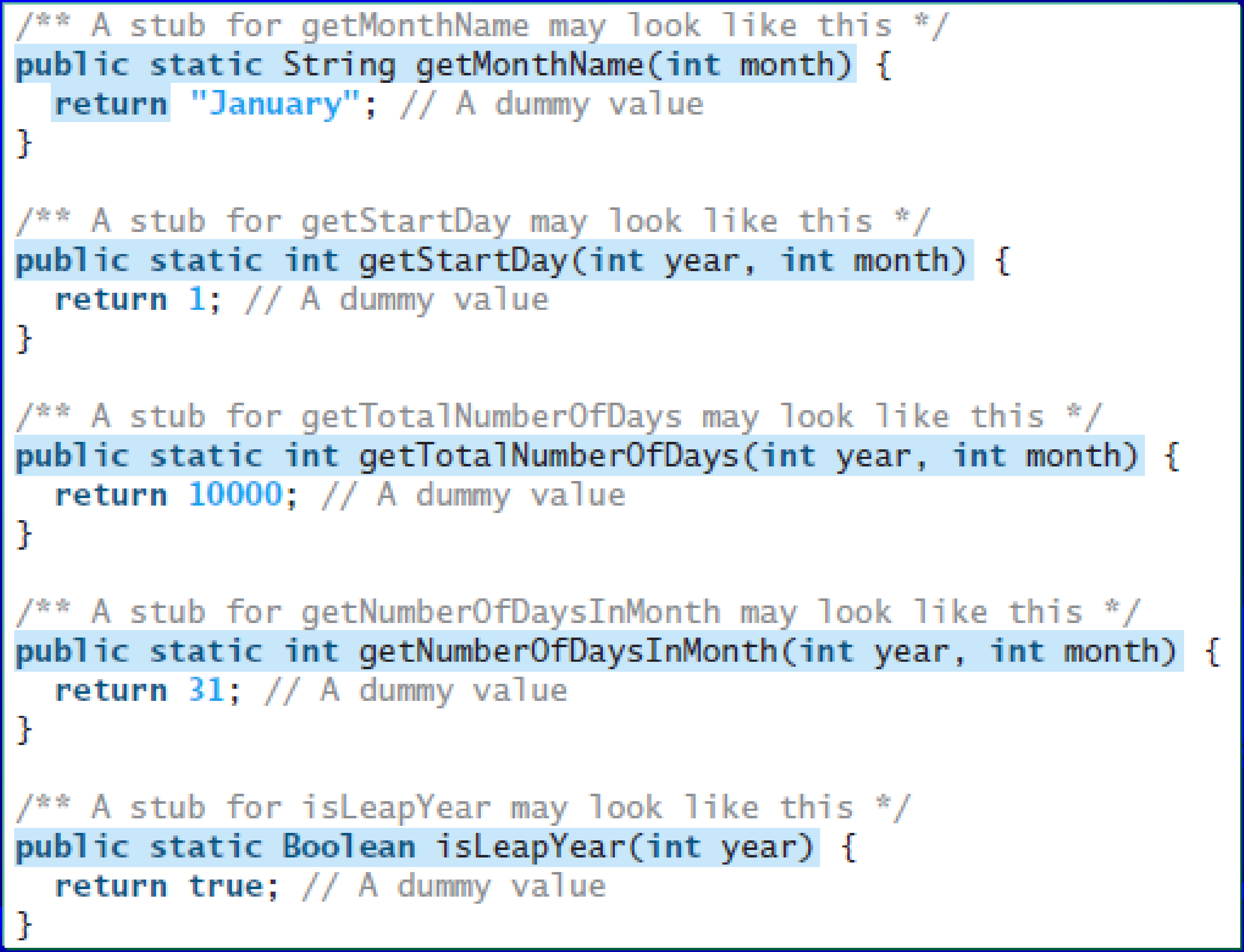
然后是其它函数

- 注意函数体都为空，因为我们还处在大框架的时候。



```
/** A stub for printMonth may look like this */
public static void printMonth(int year, int month) {
  System.out.print(month + " " + year);
}

/** A stub for printMonthTitle may look like this */
public static void printMonthTitle(int year, int month) {
}

/** A stub for getMonthBody may look like this */
public static void printMonthBody(int year, int month) {
}
```

还有几个函数



```
/** A stub for getMonthName may look like this */
public static String getMonthName(int month) {
  return "January"; // A dummy value
}

/** A stub for getStartDay may look like this */
public static int getStartDay(int year, int month) {
  return 1; // A dummy value
}

/** A stub for getTotalNumberOfDays may look like this */
public static int getTotalNumberOfDays(int year, int month) {
  return 10000; // A dummy value
}

/** A stub for getNumberOfDaysInMonth may look like this */
public static int getNumberOfDaysInMonth(int year, int month) {
  return 31; // A dummy value
}

/** A stub for isLeapYear may look like this */
public static Boolean isLeapYear(int year) {
  return true; // A dummy value
}
```

总结一下

- 上述的框架已经可以运行，虽然结果现在还不完善，至少可以确保整个设计思路没有遗漏。
- 这种逐步细化的搭框架编程方法，符合我们平时解决大问题的思维方式。
- 接下去的工作，就是一个个函数慢慢填空，这里就不再展开了……

编码实现: 自底向上（ Bottom-up）

- 还有一种编程思路和上面所讲的相反，它是先搭小模块，然后组装模块，最后形成一个大程序。
- 例如上面这个例子，自底向上的方式是先把每一个需要的函数写完，测试通过后，再组装到一起的。
- 无论哪种方法都是可用的，没有哪种更优，但是必有一款适合你。

THE END