

# 第6章 方法

## 目录

contents



**6.1 方法的定义**



**6.2 调用方法**



**6.3 方法的参数传递**



**6.4 方法的重载**



**6.5 方法局部变量的作用域**

# 6.1 方法的定义

◆方法(method)是为执行一个复杂操作组合在一起的语句集合。一个类中可以声明多个方法。

◆语法：采用BNF范式 (Backus-Naur Form, 巴科斯范式)描述  
用来描述计算机语言语法的符号集

MethodDeclaration:

MethodHeader MethodBody

MethodHeader:

Modifiers<sub>opt</sub> ResultType Identifier(FormalParameterList<sub>opt</sub>) Throws<sub>opt</sub>

Modifiers:

public protected private static abstract final  
synchronized native strictfp

ResultType:

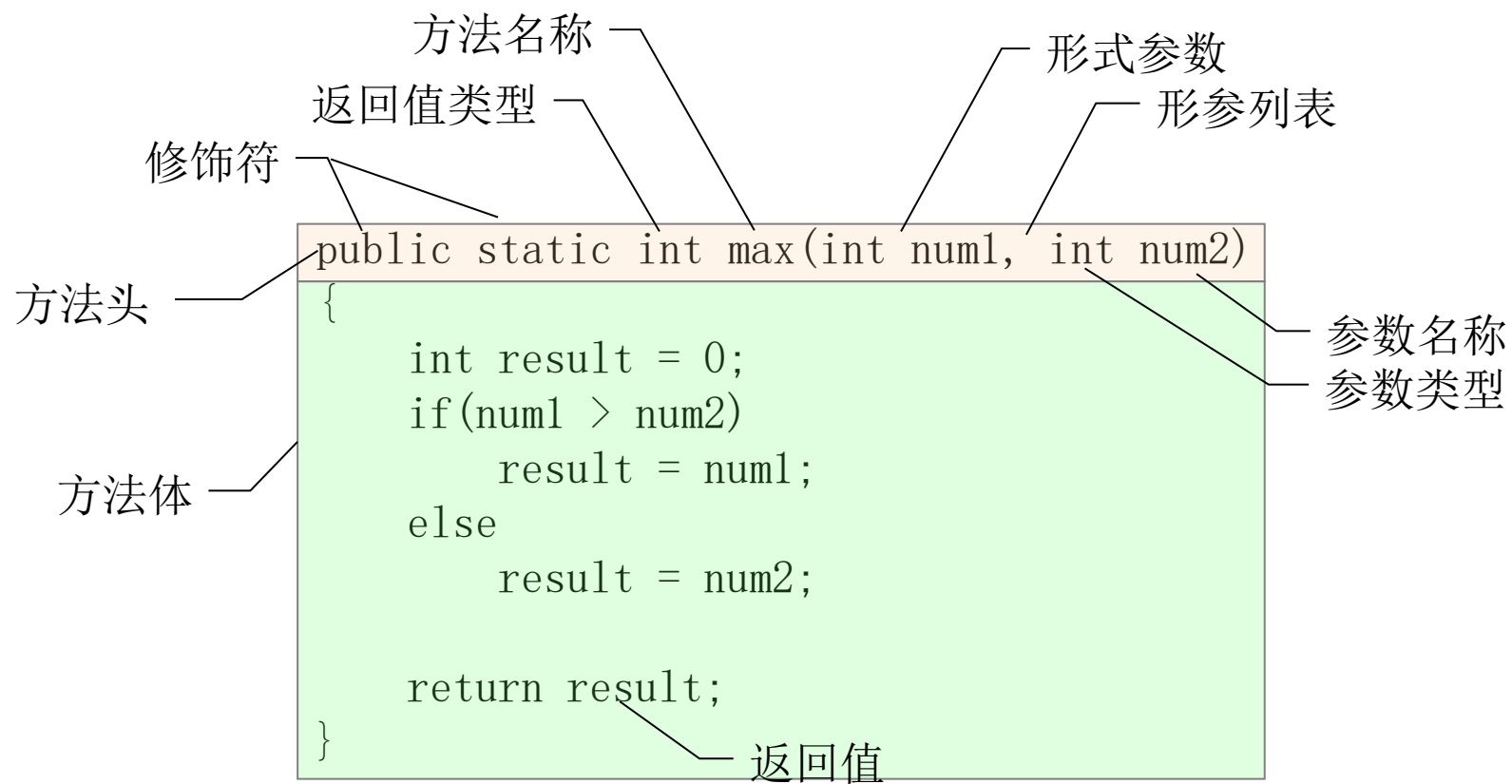
Type  
void

MethodBody:

{ statements }

# 6.1 方法的定义

## 方法定义示例



# 6.1 方法的定义

- ◆ **方法签名**(Method Signature)指**方法名称+形参列表（不含返回类型）**。一个类中不能包含**方法签名相同**的多个方法。
- ◆ **方法头中**声明的变量称为**形参**(formal parameter)。当调用方法时，可向形参传递一个值，这个值称为**实参**(actual parameter / argument)。**形参可以使用final进行修饰**，表示方法内部不允许修改该参数(类似C++的const)。
- ◆ **形参不允许有默认值，最后一个可为变长参数（可用...或数组定义，参见第7章数组）**。**方法里不允许定义static局部变量**。
- ◆ 方法可以有一个**返回值**(return value)。如果方法没有返回值，返回值类型为void，但构造函数没有返回值（不能加void）。

# 6.1 方法的定义

```
package hust.cs.javacourse.ch6;

public class FinalParameterTest {
    public static void m(int i, final int j){
        i = 10;    //i可以被重新赋值
//        j = 20;    //方法体里final int j 不能被重新赋值
    }
}
```

## 6.2 调用方法

◆声明方法只给出方法的定义。要执行方法，必须调用(call/invoke)方法。

➤如果方法有返回值，通常将方法调用作为一个值来处理（可放在一个表达式里）。

```
int large = max(3, 4) * 2;           System.out.println(max(3,4));
```

➤如果方法没有返回值，方法调用必须是一条语句。

```
System.out.println( "Welcome to Java!" );
```

◆当调用方法时，程序控制权从调用者转移至被调用的方法。当执行return语句或到达方法结尾时，程序控制权转移至调用者。

## 6.2 调用方法

### ◆方法分类的类方法和实例方法

- ◆实例方法必须用对象名调用（对象名：指向对象的引用变量名）

- ◆静态方法：可用类名调用，也可用对象名调用，**提倡用“类名.方法名”调用**，如Math.random()

- ◆调用**当前类中**的类方法：可直接用“方法名”，也可用“类名.方法名”（推荐）

- ◆调用**当前类中**的实例方法可用“方法名”或“this.方法名”调用（推荐）。

- ◆调用**其它类中**的类方法：用“类名.方法名”或“对象名.方法名”

- ◆调用**其它类的**实例方法：必须用对象名.方法名

- ◆所有类方法提倡用“类名.方法名”调用（这样可读性好，一看到类名，就知道调用的是类方法）

- ◆子类**实例函数**里用“super.方法名”调用父类**实例方法**。（*super是引用，通过super只能调用实例方法*）

## 6.2 调用方法

```
package hust.cs.javacourse.ch6;
public class A {
    public static void staticMethodOfA1() { }
    public void instancMethodOfA1() { }
    public void instancMethodOfA2() {
        //调用实例方法必须通过对象引用
        instancMethodOfA1();           //调用当前类的另一个实例方法，实际上和下面语句等价
        this.instancMethodOfA1();      //推荐用this.调用当前类的另一个实例方法，this就是指向当前对象的引用

        //调用静态方法
        A.staticMethodOfA1();          //推荐通过类名调用静态方法
        staticMethodOfA1();            //如果调用当前类的静态方法，类名可以省略
        B.staticMethodOfB();           //调用另外一个类的静态方法必须用类名

        //调用另外一个类的实例方法必须通过指向另外一个类的对象的引用
        new B().instancMethodOfB();
    }
}
```

```
package hust.cs.javacourse.ch6;
public class B{
    public static void staticMethodOfB() {}
    public void instancMethodOfB() { }
}
```



# 6.2 调用方法

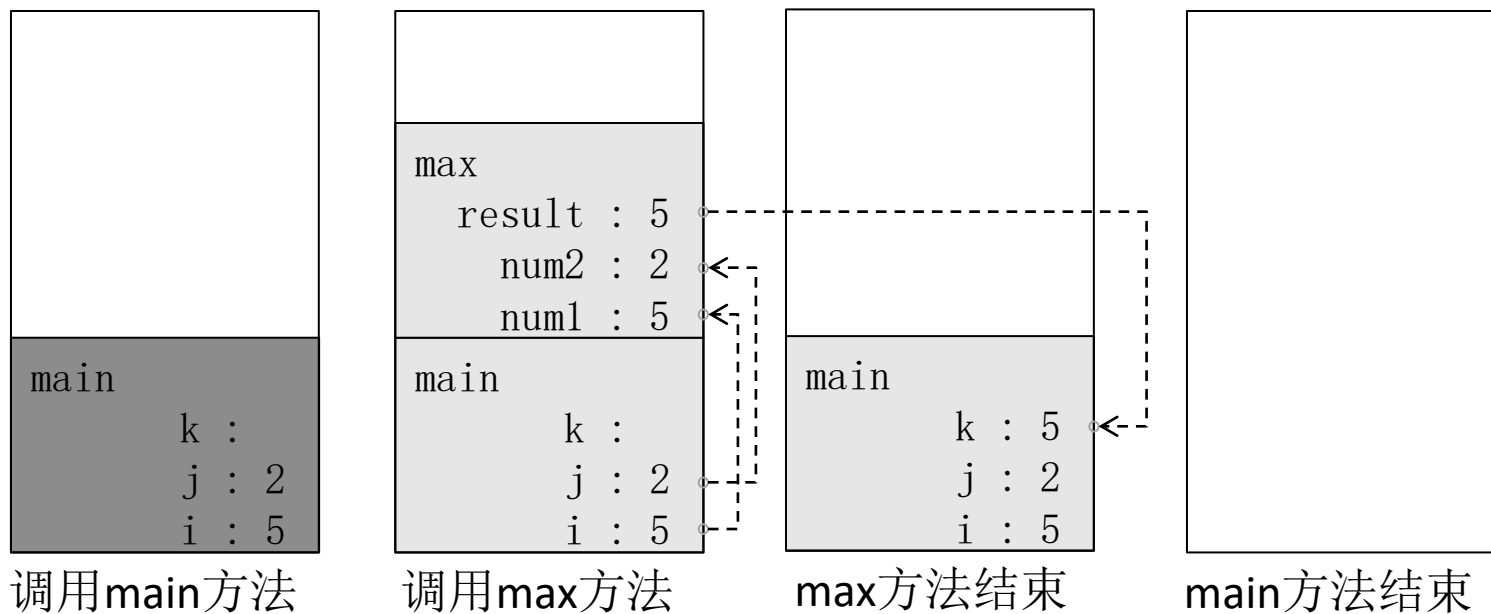
## 方法调用示例

```
public class TestMax {  
    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;  
        result = (num1 > num2) ? num1 : num2;  
        return result ;  
    }  
}
```

## 6.2 调用方法

### 调用堆栈

◆每当调用一个方法时，系统将该方法参数、局部变量存储在一个内存区域中，这个内存区域称为调用堆栈(call stack)。当方法结束返回到调用者时，系统自动释放相应的调用栈。



```
public class TestMax {  
    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;  
        result = (num1 > num2) ? num1 : num2;  
        return result ;  
    }  
}
```

## 6.3 方法的参数传递

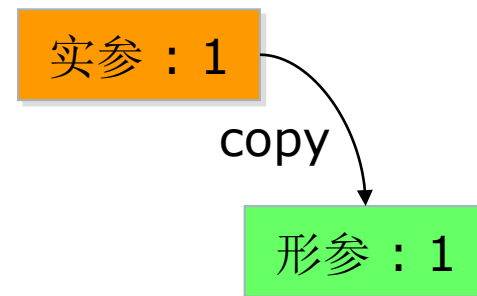
◆如果方法声明中包含形参，调用方法时，必须提供实参。

➤实参的类型必须与形参的类型兼容：如子类实参可传递给父类形参。

➤实参顺序必须与形参的顺序一致。

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

```
nPrintln("Hello", 3); //正确  
nPrintln(3, "Hello"); //错误
```



◆当调用方法时，**基本数据类型**的实参值的副本被传递给方法的形参。基本类型参数是传值调用  
方法内部对形参的修改不影响实参值。(Call by value)

◆对象类型的参数是引用调用 (Call by reference)

# 6.3 方法的参数传递

## 传递值参示例

```
public class TestPassByValue {  
  
    public static void main(String[] args) {  
        int num1 = 1;  
        int num2 = 2;  
        System.out.println("调用swap方法之前: num1 = " + num1 + ", num2 = " + num2);  
  
        swap(num1, num2);  
        System.out.println("调用swap方法之后: num1 = " + num1 + ", num2 = " + num2);  
    }  
  
    public static void swap(int n1, int n2) {  
        System.out.println("\t在swap方法内: ");  
        System.out.println("\t\t交换之前: n1 = " + n1 + ", n2 = " + n2);  
  
        int temp = n1;  
        n1 = n2;  
        n2 = temp;  
  
        System.out.println("\t\t交换之后: n1 = " + n1 + ", n2 = " + n2);  
    }  
}
```

实参 : num1

copy

形参 : n1

基本类型参数是传值调用

## 6.4 方法的重载

- ◆ 方法重载(overloading)是指方法名称相同，但形参列表不同的方法。**仅返回类型不同**的方法不是合法的重载。一个类中可以包含多个重载的方法（同名的方法可以重载多个版本）。
- ◆ 形参列表不同指
  - ◆ 参数个数不同，或
  - ◆ 至少一个参数类型不同
- ◆ 当调用方法时，Java编译器会根据实参的个数和类型寻找最合适的方法进行调用。
- ◆ 调用时匹配成功的方法可能多于一个，则会产生编译二义性错误，称为歧义调用(ambiguous invocation)。

# 6.4 方法的重载

## 方法重载示例

```
public class TestMethodOverloading {  
  
    /** Return the max between two int values */  
    public static int max(int num1, int num2) {  
        return (num1 > num2) ? num1:num2;  
    }  
  
    /** Return the max between two double values */  
    public static double max(double num1, double num2) {  
        return (num1 > num2) ? num1:num2;  
    }  
  
    /** Return the max among three double values */  
    public static double max(double num1, double num2, double num3) {  
        return max(max(num1, num2), num3);  
    }  
}
```

# 6.4 方法的重载

## 方法重载示例

```
public class TestMethodOverloading {  
    /** Main method */  
    public static void main(String[ ] args) {  
        // Invoke the max method with int parameters  
        System.out.println("The maximum between 3 and 4 is "  
            + max(3, 4)); //调用max(int num1, int num2)  
  
        // Invoke the max method with the double parameters  
        System.out.println("The maximum between 3.0 and 5.4 is "  
            + max(3.0, 5.4)); // max(double num1, double num2)  
  
        // Invoke the max method with three double parameters  
        System.out.println("The maximum between 3.0, 5.4, and 10.14 is "  
            + max(3.0, 5.4, 10.14));  
    }  
}
```

## 6.4 方法的重载

### 有歧义的重载方法示例

```
public class AmbiguousOverloading {  
    public static void main(String[] args) {  
        //System.out.println(max(1, 2)); //该调用产生歧义  
        //以下任一函数的参数都相容（都能自动转换），编译无法确定用哪个函数  
    }  
  
    public static double max(int num1, double num2) {  
        return (num1 > num2)?num1:num2;  
    }  
  
    public static double max(double num1, int num2) {  
        return (num1 > num2)?num1:num2;  
    }  
}
```



```
class TestMethod2 {  
    public static void main(String[] args) {  
        int a=2;  
        overloadMethod(a);  
    }  
    static int overloadMethod(char a){  
        System.out.println("char---int");  
        return a;  
    }  
    static void overloadMethod(double a){  
        System.out.println("double---void");  
    }  
}
```

## 6.5 方法局部变量的作用域

- ◆方法内部声明的变量称为局部变量(local variable)。方法的形参等同于局部变量。
- ◆局部变量的作用域(scope)指程序中可以使用该变量的部分。
- ◆局部变量的作用域从它的声明处开始，直到包含该变量的程序块{}结束。局部变量在使用前必须先赋值。局部变量的生命期和其作用域相同,因为{}结束时，局部变量出栈。
- ◆在方法中，可以在不同的非嵌套程序块中以相同的名称多次声明局部变量。但不能在嵌套的块中以相同的名称多次声明局部变量。
- ◆在for语句的初始动作部分声明的变量，作用域是整个循环体。在for语句循环体中声明的变量，作用域从变量声明开始到循环体结束。

# 6.5 方法局部变量的作用域

## 局部变量作用域示例

```
public class TestLocalVariable {
    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++) { //正确：两个循环未嵌套，二个for语句的i互不影响
            y += i;
        }
    }
    //错误，变量i在嵌套的语句块中声明：不能在嵌套块里声明同名的变量
    public static void method2( ) {
        int i = 1;
        int sum = 0;
        //        for (int i = 1; i < 10; i++) { //
        //            sum += i;
        //        }
    }
}
```