第四章 面向对象与类

第四章：面向对象与类

第四章：面向对象与类

4.4.3 参数传递

调用方法时，先将实参赋给形参，然后再执行操作。JAVA传参数总是采用按值传递的方式，所谓值传递就是将实参值的副本传递给被调方法的形参。

- 参数是基本类型

  那么被调方法中对该参数的改变只是改变副本，而原始值保持不变；

- 参数是引用类型

  实际传递的是对象引用的副本，这就导致初始时形参和实参指向同一个对象。

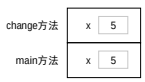
4.4.3 参数传递

1、传递基本数据类型

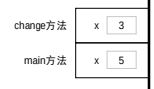
基本类型作为参数传递时，是传递变量值的拷贝，形参在被调方法栈帧上开辟新的存储单元存储拷贝值，所以，无论如何改变这个拷贝，原实参值不会改变

```
class PassParam1 {
  public static void change(int x){
      x=3;
      System.out.println(x);  //输出3
  }
  public static void main(String[] arg)  {
      int x=5 ;
      change(x);
      System.out.println(x);  //输出5
      }}
```

【运行结果】
3
5



(1) change方法被调用



(2) change方法被执行

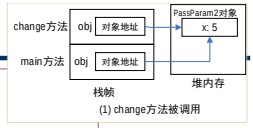
4.4.3 参数传递

2、传递引用类型

是把对象的引用（地址）拷贝一份传给形参，初始时，被调方法中的形参与主调方法中的实参指向同一个对象。

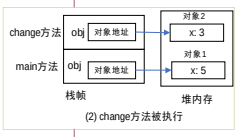
- **如果形参一直指向此对象**，则如果对对象属性进行修改，在主调方法中也能看到。
- **形参在方法体中还可以指向其他对象**，这样形参和实参将不再指向同一个对象。如果形参在方法中对其他对象进行改变，将不会影响原对象。

参数传递----传递引用类型（引用值拷贝）




```
class PassParam {
  int x;

 public static void change(PassParam obj)
 {  //obj=new PassParam(); //改变参数中存放的引用
     obj.x=3 ; //改变参数所指向实体的值
}
public static void main(String[] arg){
   PassParam obj=new PassParam();
   obj.x=5;
   change(obj);
   System.out.println(obj.x);
 } }
```



练习：

```
class Person{
    private String name;

   public Person(String name) {
      this.name=name;
    }
   public String getName()  {
       return name;
    }
   public void setName(String name) {
     this.name=name;
     }
  }//end of class
```

```
public class ByValueOrByReference{

public static void main(String []args) {
        Person p1=new Person("Zhangsan");
        Person p2=new Person("Lisi");
        changeName(p1,p2);
        System.out.println("p1: "+p1.getName()+" p2: "+p2.getName());
        swap(p1,p2);
        System.out.println("p1: "+p1.getName()+" p2: "+p2.getName());
    }
  public static void swap(Person p1,Person p2)
   {    Person temp=p1;
         p1=p2;
         p2=temp;
    }
 public static void changeName(Person p1,Person p2)
 {    p1.setName("Lisi");
      p2.setName("Zhangsan");
   }
 } //end of class
```

4.4.4 变长参数

如果一个方法的入口参数个数不能确定而又想定义这个方法，

- 传统方法是将多个参数放在一个数组中作为参数来传递，
- Java1.5之后，引入了变长参数来定义此类方法。

4.4.4 变长参数

例如，定义一个变长参数方法：

1、传统方法：

```
static int sumUp(int[] numbers) {
  int nSum=0;
  for(int i:numbers) nSum+=i;
  return nSum;
 }
```

调用方法

```
int a[]={12,13,20};
sumUp(a) ;
```

4.4.4 变长参数

- 变长参数是在变长类型与形参之间加三个点间隔符(…)来表示，具体形式如下：

  返回值 方法名（类型… 形参）

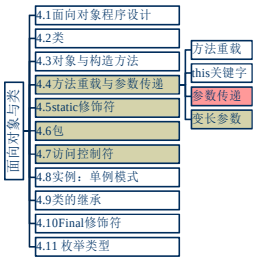
- 调用时，把要传递的实参逐一写到相应的位置即可.无需构建数组然后传入

Demo

```
class VariablePara{
   private static void sumUp(int... values){
      int sum=0;
      for(int i=0;i<values.length;i++){
          sum+=values[i];
        }
      System.out.println("the sum is:"+sum);
    }
  public static void main(String arg[]){
   sumUp(12,13,20);
    }
}
```

- 注意：
  - 1 如果一个方法还有其他的形参，只有最后一个形参可以被定义成可变参数形式。
  - 2 编译时，编译器最后会将可变形参转化为一个数组形参。处理个数可变的实参和处理数组实参的办法基本相同。所有实参都被保存在一个和形参同名的数组里。

第四章：面向对象与类



4.5 数据封装---static修饰符

问题：

为什么需要静态成员？

4.5 数据封装---static修饰符

- 当用类描述一类对象的属性及行为后，可以用new操作符来产生对象,否则并不存在任何实质的对象。产生对象后,对象方法和属性才可被访问。
- 但是有两种情况是上述设计无法解决的：

（1）第一种情况:当程序员希望不论产生多少个对象,甚至不存在任何对象的情况下,一些特定数据都是存在的且只存在一份，即需要一个类一级的全局共享变量。

（2）第二种情况是:程序员希望某些方法不需要访问对象属性，不必和对象绑定，这样即使没有产生任何对象,外界还是可以调用这个方法的。比如一些数学方法。

- 用static关键字

4.5 数据封装---static修饰符

- **类变量属于整个类，当系统第一次准备使用该类时，系统会为该类的类变量分配内存空间，此时类变量开始生效，直到类被卸载。该类所占有的内存才垃圾回收机制回收。**
- **与类相关的静态成员称为类变量或类方法，**
- **与实例相关的普通成员称为实例变量或实例方法。**
- **static关键字可以修饰字段、方法、语句块和类（只能修饰内部类，具体见第五章）**

4.5 .1 static字段

- static字段也称类/静态数据，被类的所有对象共享。
- 当系统第一次准备使用该类时，系统会为该类的类变量分配内存空间，存储在方法区中。此时类变量开始生效，直到类被卸载。该类所占有的内存才垃圾回收机制回收。

4.5 .1 static字段

1、定义static字段

static变量被该类的所有对象所共享，只能是类一级的成员，不能声明为方法的局部变量。而实例变量则是属于一个对象实例。

例如：

```
class MyCircle {
    public final static double PI=3.14159265; //静态数据
    private double radius ;            //实例数据
   }
```

4.5 .1 static字段

2、使用static字段

（1）static变量可以被同一类的其他方法直接访问。

（2）其他类可以通过此static成员所属类的类名访问它，而无需先创建对象。

- 类.静态变量

  如：

  MyCircle.PI

  System.in，System.out。

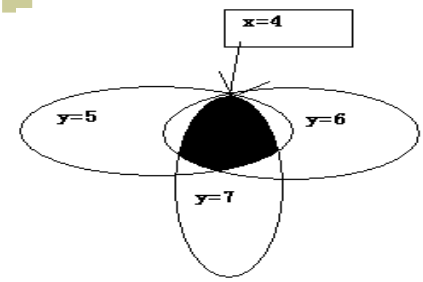
- 通过对象引用访问static变量也符合语法，但不常用。

  对象.静态变量：

  如：

```
MyCircle circleDemo=new MyCircle();
circleDemo.PI ;
```

【例4.14】定义垂直于x轴的特定直线上的点




```
class Point  {
       private static double x;
       private double y;
       public static void setX(double xtemp) { x=xtemp;
       }
       public void setY(double ytemp) { y=ytemp;
       }
       public void showInfo(){
          System.out.printf("the value of the point(%.1f,%.1f)\n",x,y);
        }}//end class

public class useStatic {
       public static void main(String arg[]){
          Point p1=new Point();
          Point p2=new Point();
          p1.setY(5);
          p2.setY(6);
          Point.setX(4);
          p1.showInfo();
          p2.showInfo();
        }} //end class
```

4.5.2静态方法（static　method）

- **用static关键字标识方法头的方法称为静态（static）方法或类方法。**
- **静态方法不向对象成员实施操作，因此在静态方法中，不能直接访问实例变量和方法，但是静态方法可以访问自身类中的静态域和方法。**

4.5.2 静态方法（static method）

1、定义静态方法

static 返回值类型 name([参数])
{……}

如：

- 主方法：public static void main(String str[])，可以在没创建对象情况下被调用。
- Math类，其中多数的数学运算都被定义成静态方法。

4.5.2 静态方法（static method）

2、使用静态方法

静态方法属于定义它的类，而且无需创建对象就

- **可直接通过类名访问它。**

  类名 . 类静态方法名(实参列表)

- **通过对象引用(无论是否为null) 调用，但实例方法必须通过非null的对象引用调用。**

  对象引用.类静态方法名（实参列表）

总结静态方法与实例方法的使用规则：

(1) 静态方法可以直接访问静态数据和其他静态方法，但不能直接引用实例变量和实例方法，因前者独立于任何对象，而后者与某具体对象相关联。

(2) 在静态方法中不能使用this、super关键字，this代表当前对象，super代表其父对象。

(3) 与静态方法相比，实例方法几乎没有什么限制：实例方法既可以调用实例数据和方法，又可以调用静态数据和方法，还可以使用supper、this关键字。

```
class FamilyMember{
        static private String surname="zhang"; //类变量surname用来表示家族成员的姓
        private String givenname;  //对象变量givennname用来表示家族成员的名

    static String getSurname(){//类方法getSurname()用来获得变量surname的值
                return surname;
   }
   static void changeSurname(String surname){ //用来改变静态变量姓
        //此处不能使用下面的语句，但可以使用类名的限定名
        //this.surname=surname;
        FamilyMember.surname=surname;
   }

   FamilyMember(){
        givenname="小刚";
   }

   FamilyMember(String givenname){
          //对于对象变量，可以使用this关键字
        this.givenname=givenname;
   }

   public String whatIsYourName(){
        //实例方法中既可以使用实例成员也可以使用静态成员
        return (surname+givenname);
   }
}
```

zhang
zhang
zhang san
li san
li

```
public class ClassMethod {
        public static void main(String args[]){
                //调用类方法可以使用带类名的限定名
                System.out.println(FamilyMember.getSurname());
                FamilyMember a=new FamilyMember(" san");
                //调用类方法也可以通过类的实例来调用
                System.out.println(a.getSurname());
                System.out.println(a.whatIsYourName());
                //类变量是共有的，即使在创建实例之后改变了类变量，实例也会知道
                FamilyMember.changeSurname("li ");
                System.out.println(a.whatIsYourName());
                FamilyMember b=null;
                System.out.println(b.getSurname());
          //System.out.println(b.whatIsYourName());// 非法操作，
                        //实例方法不能通过空引用调用。
        }}
```

4.5.3 静态块

- 静态语句块不属于任何一个方法，当类被加载时，虚拟机会执行静态块中的语句，且在类型的生命周期中只执行一次。
- 所以，可以利用静态块在类的加载阶段做一些初始化操作，如初始化静态数据。

```
class staticBlock{
 static int i;
 static {
  i=5;
  System.out.println("in the static block ,i=" +i);
}
 public staticBlock(){
  System.out.println("in the constructor");}
}

public class staticBlockTest {
       public static void main(String arg[]){
              staticBlock t=new staticBlock();
              staticBlock t2=new staticBlock();
        }
}
```

【运行结果】
in the static block ,i=5
in the constructor
in the constructor

Java语言提供的语法支持

[package 包名];

[import packagename(or packagename.class)] ;

[类修饰符] class 类名 [extends 超类名] [implements 接口列表]
{
[修饰符] data ;
[修饰符] method ;
}

4.6 package---包

包提供了一个名字空间用来把类组织起来，以便于对类的使用，管理和维护.

- 组织相关的源代码文件
- 防止类名冲突
- 提供包级别的封装和访问控制.

1、包的定义

- **有名包**
  - **[在源代码的最开始]**
  - **package** somepackage;
    package sompackage1.spackage2(多级包);
    例：package com.abc ; //表现为文件目录".\com\abc\"
- **无名包：**
  - **为没有显式声明包的类默认创建的包**

4.6 package---包

说明：

（1）包名通常全部由小写字母（多个单词也全部小写）组成。

（2）如果包名包含多个层次，每个层次用“.”分割。

（3）package 语句应该放在源文件的第一行，在每个源文件中只能有一个包定义语句。

（4）如果在源文件中没有定义包（默认包），那么字节码文件将会被放在源文件所在的文件夹中。在实际开发中，通常不会把类定义在默认包下。

（5）可以在不同的源文件中使用相同的包说明语句，这样就可以将不同文件中的类都包含到相同的程序包中了。

代码MyClass1.java
```
package mypackage;
public class MyClass1{

}
```

代码MyClass2.java
```
package mypackage;
class MyClass2{
}
class MyClass3{
}
```

编译、运行包中的类

- 1 compile
  javac -d <directory> 源文件/ *.java

2 run
java 包名.类名

```
package pk.a ;
public class TestPk
{
 public static void main(String arg[])
{
 System.out.println("test package ok");
}
}
```

4.6 package---包

2、编译、运行带包类

- javac -d . TestPk.java
- 运行时，需要指明含有main方法的类的带包全名pk.Testpk。
  java pk.Testpk

import 语句

1．使用全名

直接在类名的前面再加上包名（如果有的话），这就是类的全名。

```
class myDate extends java.util.Date{
        java.util.Date d=new java.util.Date();
        ……
}
```

2．使用import

```
import package1[.package2…].(classname|*);
```

如果要引入包中所有类，则可以用“*”来代替。注意，使用“*”只能表示本层次的所有类，不包括子层次的类。

```
import java.util.Date; //加载java.util包中的Date类
import java.awt.*;
import java.awt.event.*;
```

使用包中的类---import语句

注意：

(1) Java编译器会为所有程序自动引入包java.lang，因此不必用import语句引入java.lang包含的所有的类。

(2) 如果想引入第三方的类而不存放在JDK的内建目录，则需要用classpath环境变量指明第三方类文件的存放路径。

(3) 将一个无名包中的类引入另一个包中是不可能的，因此不建议开发时使用无名包。

使用包中的类---import语句

Jdk1.5之后的特性----静态引用

- JDK1.5引入新的特性：import static可以引入类中的静态成员。静态成员被引入后就可以直接使用，类名加或不加都可以。静态引用使我们可以象调用本地方法一样调用一个引入的方法。

- 使用形式：

```
import static 类名.静态成员
import static 类名.*
```

使用包中的类---import

☐当要获取一个随机数时，1.5版本以前的写法是：

```
double x = Math.random();
```

☐而在1.5版本中可以写为：

```
import static java.lang.Math.random; //程序开头
double x = random();

import static java.lang.System.*; //程序开头
out.println(“hello world”);
```

不过这种简写形式，似乎值得怀疑，因为代码清晰度不够

第四章：面向对象与类

面向对象与类
- 4.1面向对象程序设计
- 4.2类
- 4.3对象与构造方法
- 4.4方法重载与参数传递
  - 方法重载
  - this关键字
  - 参数传递
  - 变长参数
- 4.5static修饰符
- 4.6包
- 4.7访问控制符
- 4.8实例：单例模式
- 4.9类的继承
- 4.10Final修饰符
- 4.11 枚举类型

4.7 访问控制符

- Java提供了四级访问控制权限：公有(public)、受保护(protected)、包权限（默认权限）、私有（private）。
- 权限修饰符主要用来修饰类、数据域和方法。

```
[访问修饰符][其他修饰符]class 类名 [extends 父类名] [implements 接口列表] {

    [访问修饰符][其他修饰符] [成员变量]
    [访问修饰符][其他修饰符] [静态初始化块]
    [访问修饰符][其他修饰符] [构造方法]
    [访问修饰符][其他修饰符] [普通成员方法]
};
```

修饰符

表 4-2 访问权限修饰符

修饰符	权限级	修饰对象				可见性		
		类	字段	方法	同类	同包	不同包中的子类	不同包中的非子类
public	公有	√	√	√	*	*	*	*
protected	受保护		√	√	*	*	*	
无	包	√	√	√	*	*		
private	私有		√	√	*			

4.7 访问控制符

1、类访问权限修饰符

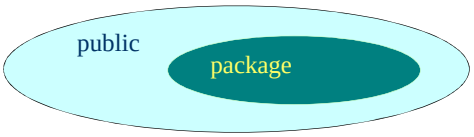
2、成员访问权限修饰符

first step :类访问权限

second step :成员访问权限

First----类访问权限

类的访问权限有两类：公有（public）、和包权限（无修饰符）



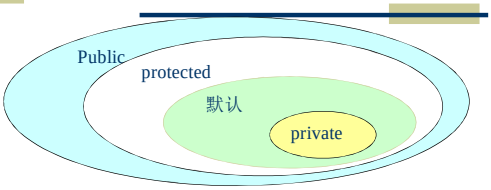
类访问权限

[public] class 类名

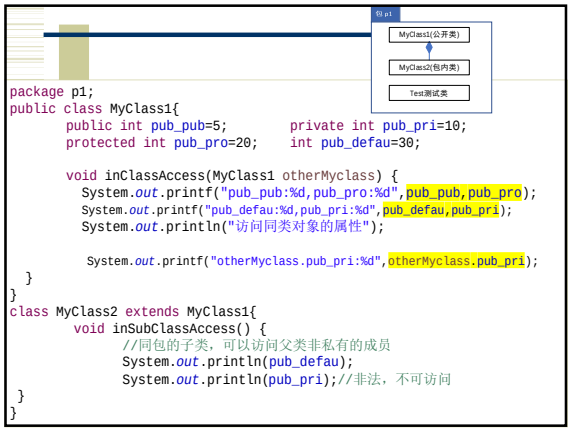
	相同的包	不同的包
public	y	y
默认	y	n

- （1）public 类，表示该类对其他类来说是可见的，无论其他类在包内还是包外。
- （2）无权限修饰符类，表示该类是包内类，对同一个包里的类可见。

Second:成员访问权限



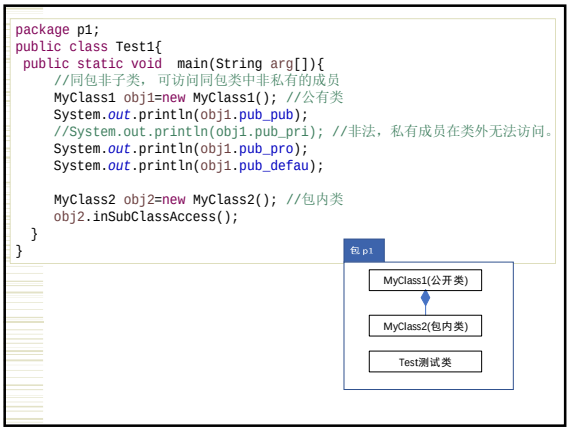
- 在class类型能被访问的前提下
  - private 成员: 被同类方法访问
  - 默认成员:被同类和同包内其他类方法访问
  - protected成员:可被包内类和 子类方法访问
  - public 成员:如果类型能被访问,被所有方法访问.




```
package p1;
public class MyClass1{
       public int pub_pub=5;        private int pub_pri=10;
       protected int pub_pro=20;    int pub_defau=30;

       void inClassAccess(MyClass1 otherMyclass) {
         System.out.printf("pub_pub:%d,pub_pro:%d",pub_pub,pub_pro);
         System.out.printf("pub_defau:%d,pub_pri:%d",pub_defau,pub_pri);
         System.out.println("访问同类对象的属性");

          System.out.printf("otherMyclass.pub_pri:%d",otherMyclass.pub_pri);
  }
}
class MyClass2 extends MyClass1{
        void inSubClassAccess() {
              //同包的子类，可以访问父类非私有的成员
              System.out.println(pub_defau);
              System.out.println(pub_pri);//非法，不可访问
 }
}
```



```
package p1;
public class Test1{
 public static void  main(String arg[]){
     //同包非子类，可访问同包类中非私有的成员
     MyClass1 obj1=new MyClass1(); //公有类
     System.out.println(obj1.pub_pub);
     //System.out.println(obj1.pub_pri); //非法，私有成员在类外无法访问。
     System.out.println(obj1.pub_pro);
     System.out.println(obj1.pub_defau);

     MyClass2 obj2=new MyClass2(); //包内类
     obj2.inSubClassAccess();
  }
}
```

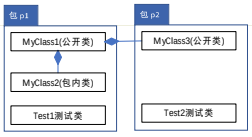


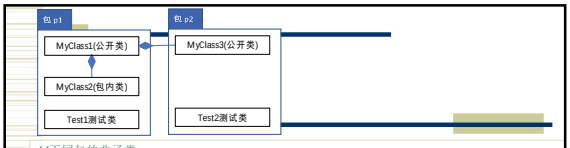
```
package p2;
import p1.MyClass1;
//import p1.MyClass2; //非法，MyClass2是P1包内类，对包外类MyClass2不可见
//不同包的子类
class MyClass3 extends MyClass1{
 public void func(MyClass1 superMC, MyClass3 otherC){
      //类只能继承不同包的父类的public、protected成员
          System.out.printf("pub_pub:%d,pub_pro:%d",pub_pub,pub_pro);

          //无法直接访问继承自父类的私有private成员和包权限成员
          System.out.printf("pub_defau:%d,pub_pri:%d",pub_defau,pub_pri);//非法

          //对同类型的其他对象othetC的成员权限，等同于对该类型当前对象成员的权限。
          System.out.printf("pub:%d,pro:%d", otherC.pub_pub, otherC.pub_pro);

          //无法访问父类对象的protected权限成员。
1.        System.out.printf("In Sup Myclass1,pub_pro:%d",superMC.pub_pro);//非法
  }}
```






```
//不同包的非子类
public class Test2{
 public static void main(String arg[]){
   //不同包的非子类，只能访问其他包的公开类的公开方法
   MyClass1 obj1=new MyClass1();
   System.out.println(obj1.pub_pub);
   System.out.println(obj1.pub_pri); //非法，私有成员对类外方法不可见
   System.out.println(obj1.pub_pro); //非法，受保护成员对非子类的类外方法不可见
   System.out.println(obj1.pub_defau); //非法，包权限成员对包外类不可见

 MyClass2 obj2=new MyClass2(); //非法，p1中的包内类MyClass2对p2中的类不可见

  //测试MyClass3的func()方法
   MyClass3  obj3=new MyClass3();
   MyClass3  otherMyClass3=new MyClass3();
   obj3.func(obj1, otherMyClass3);
 }}
```

访问控制符

- 面向对象设计的松耦合性，需要用到封装，其原则为
  （1）将对象的成员变量和实现细节尽量隐藏起来，不允许外部访问
  （2）把方法暴露出来，使调用者通过方法对成员变量进行安全访问和操作。