



第三章面向对象技术

南京农业大学
谢忠红



3.1 类和对象的定义和使用

- **Java** 是一种面向对象的编程语言（面向过程）
- 面向对象的定义：
 - 万物皆对象，客观世界是各种对象的集合
 - 复杂的对象由简单的对象组合而成
 - 每个对象都是唯一的（单独分配内存）
 - 对象具有属性和行为，他们之间进行消息发送
 - 对象都属于某个类



类和对象的概念

- 什么是类？什么是对象？
- **对象**——在我们所处的客观世界中，每一个有明确的意义和边界的事物都可以看作一个对象（Object）例如：某台电脑，一辆汽车
- **对象类**（简称类Class）——我们可以把具有相似特征的事物归为一类，也就是所把具有相同属性的对象看成一个对象类。例如：“电视机类”

类----电视机的设计图 对象---各家各户的电视机

- **Java** 是通过先定一个类，然后按照类的结构创建一个又一个对象。



类与对象的区别

- **类**是在一组对象的基础上，通过抽象和概括获得的一个概念。
- **类**是一个**抽象**的概念，**对象** 是一个具体的概念。
- **对象**是由数据和方法紧密结合的一个封装体，具有信息隐藏的能力。
- **对象**可以通过方法（函数）与其它对象进行通信。



使用Java如何创建一个类

类声明格式:

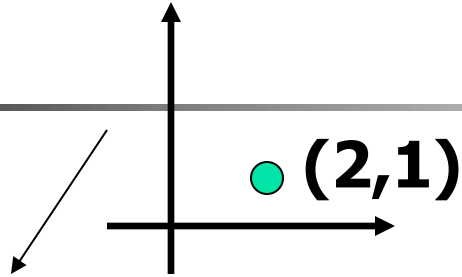
```
<修饰符>  class <类名>
{
    <属性声明>
    <构造函数>
    <行为声明>
}
```

修饰符: public|private|空

类名: 一个自定义的修饰符号

```
public class person{
    private String name;
    int age=12;
    int weight=30;
    public person() { ...}
    public void speak(){...}
    public String go(){..}
    public void work(int a)
        {...} }
```

抽象类举例



```
class point {  
    int x , y;                //属性声明  
    point( ) {x=0;y=0; }      //默认的构造函数  
    point( int x1, int y1){x=x1 ;y=y1;} //自定义构造函数  
    void print(){ System.out.println("(" + x + "," + y + ")"}  
}
```



构造函数——在创建对象的时候自动调用的一个方法，该方法与类名同名。

构造函数的作用 **(1)** 主要是用来初始化对象的数据属性。 **(2)** 创建对象时自动调用



构造函数的特点：

- 1、构造函数的函数名与类名相同
- 2、构造函数没有返回值(但是不加**void**修饰符)
- 3、构造函数可以没有参数——默认构造函数
也可以有参数——自定义构造函数

返回



通过类创建对象

格式:

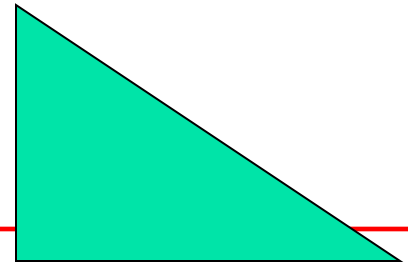
类型 对象名= **new** 类型 ([参数列表]) ;

比如:

```
Point p1= new Point ( 10,10,15),
```

```
Point p2= new Point (20,5,40 );
```

例：三角形类



```
public class Triangle{
```

```
//属性声明，分别定义了三条边a,b,c
```

```
public float a=0.0f , b=0.0f, float c=0.0f;
```

```
//构造函数
```

```
Triangle(float a1,float b1,float c1){
```

```
a=a1;b=b1;c=c1; }
```

```
//行为声明，定义了三角形面积求解
```

```
public float Cir() {float d=a+b+c; return d;}
```

```
}
```

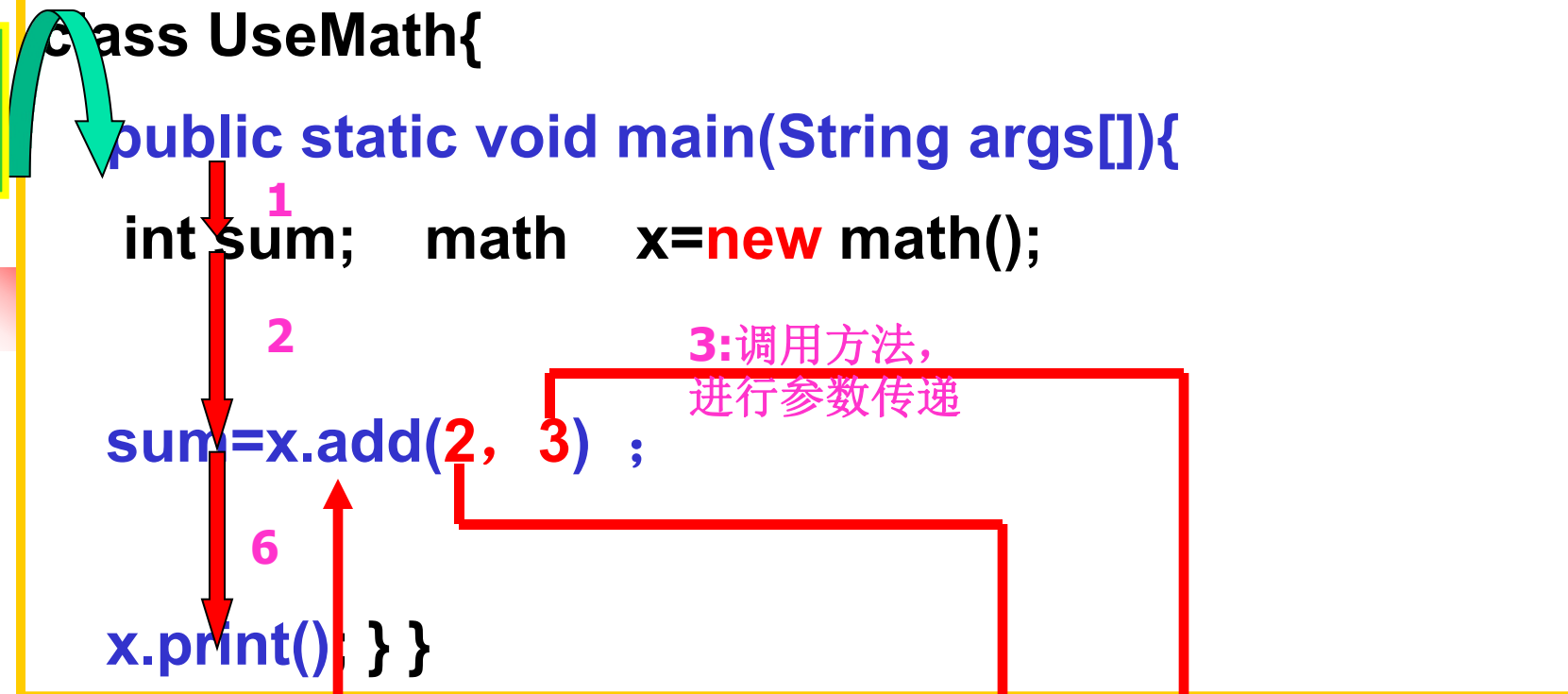
使用三角形类

```
public class    triangleTest{  
    public static void main(String[ ] args){  
        float f=0.0f,s=0.0f;  
  
        Triangle t1=new Triangle(3,3,3); //创建了一个三角形t1这个对象  
  
        f=t1.Cir();  
        System.out.println("三角形t1的周长是:" +f);  
        t1.a=12.1f;    t1.b=15.0f;    t1.c=17.5f;  
        f=t1.Cir();  
        System.out.println("三角形t1的周长是:" +f);  
    }  
}
```

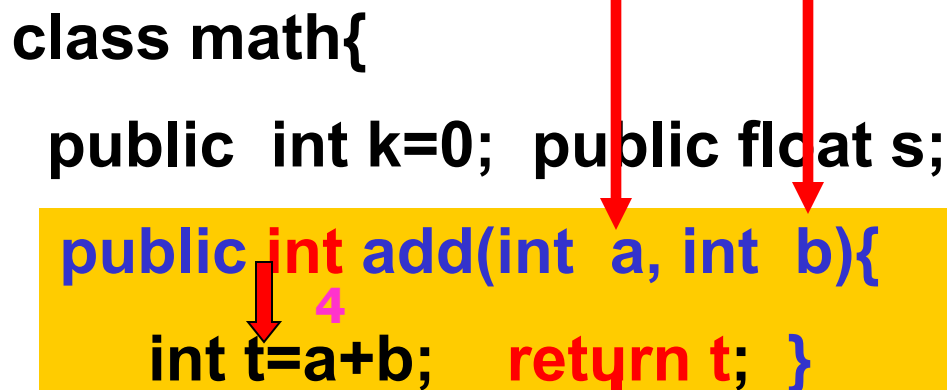
2方法的定义和使用

方法声明

＜修饰符＞＜返回值类型＞＜方法名＞（形参1, 形参2, ...） {
 局部变量声明;
 语句组; }



5方法
名带回
返回值





实例变量和局部变量

- 类内定义的实例变量;
- 方法中定义的局部变量;

```
class point {
```

```
    int x, y;           //实例变量
```

```
    void init (int a, int b ){
```

```
        int x=0;       //局部变量;
```

```
        x=a;  x=b;      }
```

```
}
```

实例变量和局部变量

```
class Point {  
    protected int x, y;           //实例变量  
    point ( int x1, int y1)  { x=x1 ;y=y1;}  
    public void setValue(){      x=5;y=10;}  
    public void printValue(){    int x=15;           //局部变量  
        System.out.println(++x); }  
    public void printInsValue(){ x=5*x;y=y/5;  
        System.out.println(x+y);      }  
    public static void main(String args[ ]){ //在类中  
        Point p1=new Point(10,10);  
        p1.setValue(); p1.printValue(); p1.printaInsValue() ; }  
}
```



实例变量和局部变量的不同点

- (1)实例变量：属于对象，随着对象的存在而存在；局部变量：随着方法的调用而存在。
- (2)局部变量可以掩盖实例变量。
- (3) C++支持全局变量而Java不支持

构造函数的再说明



```
■ class Date {  
    ■ int year,month,day;  
    ■ void print() { }  
    ■ }
```

如果没有构造器，那么系统自动创建一个默认构造器

```
■ class Date {  
■     int year,month,day;  
■     void print() {  
■         System.out.println("date is "+year+'-'+month+'-'+day); }  
■ }
```

```
■ class Date {  
■     int year,month,day;  
■     Date(){ }  
■     void print() {  
■         System.out.println("date is "+year+'-'+month+'-'+day); }  
■ }
```

多个构造器

```
class Point{
    int x,y,z;
    Point() { x =1;y=1;} //默认的构造函数
    Point(int new_x,int new_y)
        { x=new_x,y=new_y;}
    Point(int new_x,int new_y,int new_z)
        { x=new_x,y=new_y;z=new_z;}
}
```

其中无
参构造
其实默
认的构
造器

```
class UsePoint{
    Point point_A=new Point();
    Point point_B=new Point(5,7);
    point point_c=new Point(10,20,87)
}
```



■ 面向对象的三大特征

- (1) 封装性
- (2) 继承性
- (3) 多态性

- **封装的定义**: 类的设计者把类设计成一个**黑匣子**，使用者只能看见类定义的公共方法而**看不见方法实现的细节**；也不能对类的数据进行操作。



什么是封装呢？



如何封装？

Java 种只要在实例变量的前面加上**Private**修饰符就可以达到封装的目的。

Public(公有的): 如果某属性或方法声明为**public**，表示该属性或方法可以被外部访问

Private(私有的): 如果某个属性或方法声明为**private**，则该属性或方法只能在类内部访问。

- `public class MyDate {` `//类的封装`
- `private int year,month,day;` `//成员变量,私有的`
- `public MyDate(int y,int m,int d){` `//对私有成员赋值`
- `year = y; month = m; day = d ;}`
- `public void setYear(int y){year=y;}`
- `public void setDay(int d){dayr=d;}`
- `public int getYear(){return year; }` `//返回私有成员以便利用`
- `public int getMonth(){return month; }`
- `public int getDay() {return day; }`
- `public void print() {`
- `System.out.println("date is "+year+`:`+month+`:`+day);` `}`
- `}`

对吗？

```
class EncapsueDate{  
    public static void main(String args[]) {  
        MyDate a = new MyDate(2002,6,28);  
        a.day=13;  
        System.out.println(a.year+":"+a.month  
                             +":"+a.day);  
        a.print();  
    }  
}
```

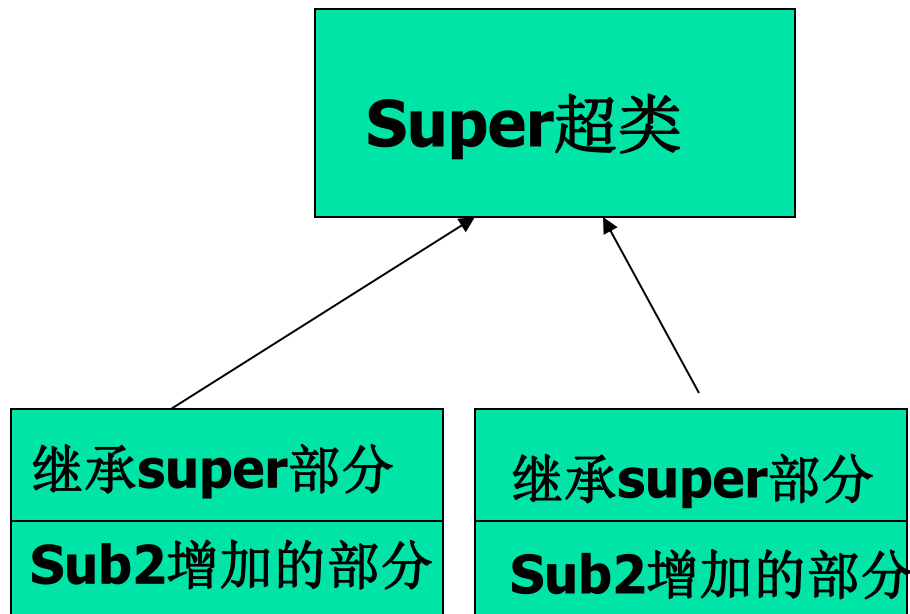
对吗？

怎样修改呢？

Java中的继承

- **DEF:** 子类**继承**父类的属性和方法（可以**修改**父类的属性或**重载**父类的方法），以及在父类的基础上**添加**新的属性和方法则就叫做继承。

- 在 Java 中，有一个被称为 *Object* 的特殊超类，所有的类都直接或间接地继承 *Object* 类



继承语法：

```
class className extends superClassName{  
    各实例变量  
    方法的定义  
}
```

《注意》 没有extends，默认父类为Object

↓ 只能有一个父类，即单继承

↓ 子类继承父类的全部成员

↓ 类继承具有传递性

父类： // People类的声明

```
class People    {  
    public String  name; public int   age; public char  sex;  
  
    People(String name,int age,char sex ){  
        this.name=name;this.age=age;this.sex=sex; }  
  
    void setNameAgeSex(String name, int  age, char sex){  
        this.name=name,this.age=age,this.sex=sex;}  
  
    String getName(){return  name;}           // 查询姓名  
  
}
```

子类 Teacher类:

```
class Teacher extends People{  
    String course //教授课程
```

```
Teacher( String name, int age, char sex , String course ){  
    this.name=name;    this.age=age;  
    this.sex=sex;        this.course=course ; }  

```

```
String whichCourse() { return course ; } // 查询所授课程
```

```
void setValue(String name, int age, char sex , boolean isTeaching){  
    ■ setNameAgeSex(name, age, sex);  
    ■ this.course=course; }  

```

```
void print() {
```

```
    System.out.println(“ ”+ name+ “ ”+sex+ “ ”+age+“ ”+ course);}
```

```
}
```

有关继承必须说明的几点：

- (1) 子类能够继承父类中声明的所有成员变量，
不能直接使用private类型的成员变量。
- (2) 如果子类声明了一个与父类相同的成员变量
(成员方法)，*子类的变量（方法）会覆盖
超类的成员变量（方法）。*
- (3) 子类的构造函数与父类构造函数的关系

举例:

class Employee { //父类

```
private String name; private int age; private float wage;  
Employee(String n,int a;float w ){ name=n;age=a;wage=w;  
setValue(String n; int a; float w){name=n;age=a;wage=w;  
String getName(){return name;} ...  
print(){ System.out.println(name+ ":" + age+ ":" + wage);} }
```

对吗?

怎样
使程
序更
简洁?

class Manager extends Employee { //子类

```
string depart; string title;  
Manager(String n,int a;float w, String d,String t){  
    super(n,a,w); Department=d; title=t; }  
print(){ System.out.println(name+ ":" + age+ ":" + wage);  
        System.out.println(depart+ ":" + title);} }
```



类的成员覆盖

定义:在子类中新增加的成员变量或成员方法的名称与父类相同。

- **成员变量:** 子类存在和父类完全相同实例变量，这样在子类中父类实例变量好像不存在。
- **成员方法:** 子类存在和父类完全相同的方法，这样在子类中父类方法好像不存在。

覆盖习题1

```
class A{  
    int x = 1234;  
    void show( ) {  
        System.out.println("class A : ");  
    }  
}
```



```
class B extends A {  
    double x = 567.89;  
    void show( ) {  
        int x=45;  
        System.out.println("class A: x"+super . x);  
        System.out.println("class B: x"+x);  
        super.show( );  
        System.out.println("class B : ");  
    }  
    public static void main(String args[]){  
        B x=new B();  
        System.out.println(x.x);  
        x.show();  
    }  
}
```



程序运行结果？

覆盖的习题2

阅读下面的程序

```
■ class Person {  
■     String name, depart;  
■     public void printValue(){  
■         System.out.println("name:"+name"depart: "+depart);}  
■ }  
■ public class Teacher extends Person {  
■     int salary;  
■     public void printValue(){  
■         // doing the same as in the parent method printValue()  
■         // including print the value of name and depart.  
■         System.out.println("salary is "+salary); }  
■ }
```

■ 下面的哪些表达式可以加入printValue()方法的"doing the same as..."部分()

- A. printValue(); B. this.printValue();
- C. person.printValue(); D. super.printValue();



再论构造函数

- **this** 引用和**super**引用
- 关键字**this**
- **DEF:**代表一个特殊的对象即当前对象。
- **(1)**指代对象
- **(2)**指代构造函数

this代表一个特殊的对象即当前对象

例: *Class Button*{

- char c;
- boolean equals(**Button other**){
- if (this==other) return true;
- return false;
- }
- }

地址传递



class test Button {

public static void main(String args[]){

Button t1=new *Button* (), t3=new *Button* ();

t3=t1;

System.out.println(**t3.equals(t1)**); }}

- **this** 指代对象可以用于解决实例变量被局部变量屏蔽的问题

```
■ public class MyDate {           //类的封装
■     private int year,month,day;
■     public MyDate (int year,int month,int day){
■         this.year = year;      this.month = month;
■         this.day = day;    }
■     public setDay (int day) {    this.day = day; }
■ }
```

date1=new MyDate(2002,5,8) 构造函数中的 **this.year** 就是指指的是**date1**的实例变量**year**

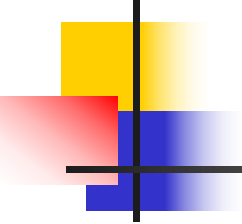


关键字super

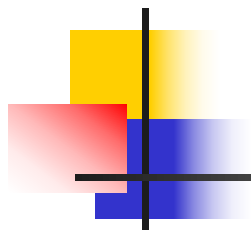
- 构造函数是一种特殊的方法，子类不能继承超类的构造函数，
- 但子类构造函数可以通过super调用超类的构造函数。

```
class point{  
    int x, y;  
    point(int x, int y){  this.x=x;      this.y=y;  
        System.out.println("父类构造函数被调用！ "); }  
}
```

```
class circle extends point {  
    int radius;  
    circle(int r, int x, int y){  
        super(x, y);    //调用父类的构造函数  
        radius=r;  
        System.out.println("子类构造函数被调用！ "); }  
}
```

```
public class testInherence {  
    public static void main(String args[ ]) {  
        circle    c1;  
        c1=new circle(1,1,1);  
    }  
}
```



有关构造函数，
请注意下列几个例子

```
或: class point{  
    int x=1, y=1;  
    point(){  
    }  
}
```

此处相当于有一个
`super();`

子类

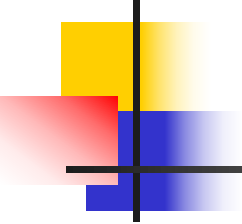
```
class circle extends point{  
    int r;  
    circle(int r){ radius=r; }  
    public static void main(String args[]){  
        circle c1=new circle(20);  
        System.out.println("x="+c1.x  
            +"y="+c1.y+"r="+c1.r);  
    }  
}
```

程序运行结果?



■ 总结:

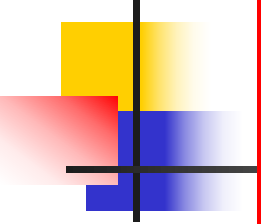
- 若父类没有定义构造函数或只有一个默认构造函数
- 那么子类先执行父类默认的构造函数（对父类数据的初始化），然后再执行自己的构造函数



若父类中
含参构造函数?



答案：子类可以在自己的构造函数中使用
super调用它。



```
class circle extends point{  
    int radius; //  
    circle(){ }  
    circle(int r, int x, int y){  
        radius=r;this.x=x;this.y=y;}  
}
```

```
class Cylinder extends Circle{  
    private double height;  
    public Cylinder(int x1,int y1,double r1,double ){  
        super(); height=h; }  
}
```

这个程序能编译成功吗？

```
■ class Point{  int  x, y;  
    Point( ){  
    Point(int x,int y){  this.x=x ; this.y=y; }  
    }  
■ class Circle  extends Point{  
    int r;  
    Circle (){ super(); x=10;y=10;r=10;}  
    Circle(int r){  
        super(20,20) ;  this.r=r;    }  
    }
```

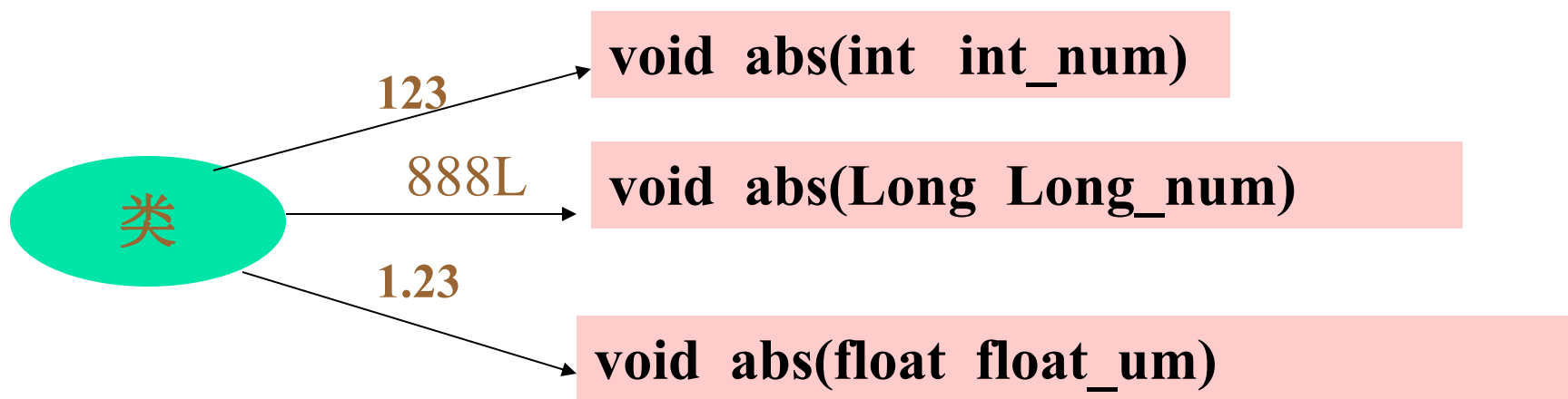


类的多态性

- **(1)方法的覆盖(-简单的多态)**
- **(2)方法重载**
- **(3)构造函数的重载**

方法的重载

在**Java**中，同一个方法名可以被定义多次，但**要求参数表不能完全一样**。调用方法时，系统是通过方法名和参数确定所调用的具体方法。这种现象叫做方法或函数的**重载**。



```
class mysqure{  
    int square( int x ) { return x * x; }  
    double square(double y ) { return y *y*; }  
}
```

```
public class MethodOverload {  
    public static void main( String[] g ) {  
        mysqure s=new mysqure();  
        System.out.println( "The square of integer  
            17 is " + s.square( 17 ) );  
        System.out.println( "The square of double  
            17.5 is " + s.square( 17 ) );  
    }}
```

构造函数的重载（特殊的方法）

```
■ public class Person{  
■     static int count=0;  
■     protected String name;     protected int age;  
■     public Person(String n1,int a1) {  
■         name = n1; age = a1; count++; }  
■     public Person(String n1) { //构造方法重载  
■         this(n1,0); } //调用本类的构造方法  
■     public Person(int a1){this("未知名",a1); }  
■     public void print() {  
■         System.out.print("count="+count+" ");  
■         System.out.println(" "+name+", "+age); }  
■ }
```

■ 方法重载的价值：

- 允许使用一个普通的方法名，来访问一系列相关的方法，利于程序员的记忆。

■ 方法重载的注意点：

- **(1)** 参数必须不同，即参数个数不同、类型也可以不同。
- **(2)** 重载可以出现在一个类中，也可以出现在父类与子类的继承关系中，并且重载方法的特征一定不完全相同。
- **(3)** 只有返回值不同不能算是方法重载。