



# 面向对象程序设计Java

江春华

电子科技大学信息与软件工程学院

# 内 容

## 第4章 类和对象

1

面向对象程序设计

2

类的创建

3

类的成员

4

类成员和实例成员

5

对象创建与使用

# 面向对象程序设计

- ❖ 面向对象程序（OOP）依照现实世界的实体的特点，把复杂的现实的事物按它们所共有的状态和行为抽象并封装成。
- ❖ 使现实世界中的概念在计算机程序中变成模块。它包括构造程序的特征以及组织数据和算法的机制。

# 面向对象程序设计

❖ 面向对象程序语言有三个特征：

- 封装性
- 多态性
- 继承性

❖ 面向对象程序设计具有许多优点。与现实世界的“对象”或者“物体”相比，编程“对象”都有自己存在共通的状态和行为。

# 面向对象程序设计

- ❖ 对基于“数据”本身，而非“功能”的面向对象程序设计，从抽象的概念上分析所要处理的数据是哪些类(class)，到各类数据可进行的操作方法，都是通过各个封装好的数据模块组合起来解决问题。
- ❖ 一个数据模块称为一个对象(object)，它是抽象出来的某个类(class)的一个实例。类就是一个引用数据类型，对象则是这个引用数据类型的变量。对象就是经过实例化的实体的引用。

# 面向对象程序设计

- ❖ **类**的变量描述对象的属性，类的方法体现对象的行为。封装在类中的这些数据的变量和方法，为对象的创建提供了模板。
- ❖ **对象**之间的交互作用是通过对象的消息机制实现的。所谓**消息**是对象对自身方法的访问，一个消息的产生就是一个对象方法的调用。
- ❖ 类和对象是面向对象程序设计中的基本概念，也是编程方法的基础。类就是Java的执行单位。  
Java运行的就是Java类本身

# 类的创建

❖ **类**是Java语言的最基本概念，组成Java程序的基本要素。类是Java的执行单位，Java运行的就是Java类本身。它封装了该类对象的变量和方法。

❖ 一个类包括有三个部分：

- 类声明；
- 类成员；
- 类的构造器(方法)。

❖ 类的一般格式是：

```
classDeclaration{  
    classBody  
}
```



# 类的创建

## ❖ 类的一般声明形式:

```
[public] class <clsName> extends <supCls> implements <intf>
```

- **class** 是表示创建类的关键字;
- **<clsName>** 是Java合法标识符;
- **[public]** 是可选项, 表示该类是**public**类; 类的可选项还有**abstract**、**final** 等等;
- **extends <superCls>** 则是继承性表示, 该类继承了类**<superCls>**
- **implements <intf>** 则是对接口实现表示, 该类实现了接口**<intf>**



# 类体定义

## ❖ 类的体包含有:

- 成员变量。在类中创建的变量，表示对象属性；
- 成员方法。类的方法表示对象的行为或能力；
- 类的构造器 (方法)。为创建类的实例所使用。

## ❖ 类的一般格式是:

```
classDeclaration{  
    memberVariableDeclaration  
    memberMethodDeclaration  
    classStructorDeclaration  
}
```

# 类的成员变量

## ❖ 成员变量

- 在类中创建的变量，表示对象属性；
- 作用域在类中是全局的，能被类中所有方法所访问；
- 在创建时有初始化值。

例： `int a, b;` //成员变量a和b都有一个初始值为0。

## ❖ 成员变量的定义还有许多可加的修饰符，用于声明成员变量的访问控制权限和使用限制。

- 访问控制权限的修饰符有public、protected、private等；
- 使用限制的修饰符有final、abstract、static、transient、volatile等。

# 类的成员变量初始化

## ❖ 成员变量初始化分两种：

➤ 创建的变量时初始化值。如第2章中表2-6所示；

例： `int a, b;` //成员变量a和b都有一个初始值为0。

➤ 创建时赋初值；

例： `int x = 10, y = 20;` //x, y分别赋初值10, 20。

➤ 在类的构造器中对成员变量的赋初值。

例： `Point() {`  
     `x = 100; //x赋初值100`  
     `y = 200; //y赋初值200`  
     `}`

# 类的成员变量——常量

- ❖ 使用修饰符 `final` 修饰的变量就像常量一样地使用，称其为常量符号。
- ❖ 常量符号数据只能读取，不能改变。通常常量符号标识符全用大写字母，单词间用 “\_” 分隔。

例：

```
final int MAX_MONTH = 12;
final int MAX_DAY = 31;
final int MAX_WEEK = 7;
```

# 类的成员方法

❖ 方法的创建分为二个部分，一个是方法声明，另一个是方法体。

格式为：

```
methodDeclaration{  
    methodBody  
}
```

方法声明格式：

**[accessControl] returnType methodName(paraList)**

方法体格式：

语句及语句块组成；

# 类的成员方法

## ❖ 方法声明

**[accessControl] returnType methodName(paraList)**

[accessControl]是可选项，为访问控制修饰符，限定访问权限。

returnType是方法返回数据类型，它表示方法返回时返回数据的类型。

methodName是方法名，它是Java合法标识符。

paraList是方法参数列表，表示方法调用时所带参数。称为形式参数。

例如：

```
int getMax(int a, int b)
void setData(int x, int y)
```

# 类的成员方法

## ❖ 方法体

语句及语句块组成

包含有：

局部变量声明；

流程控制语句；

语句块；

return语句；

当方法返回类型不是void时，方法中必须有带表达式return语句：

**return expression;**

当方法返回类型是void时，方法中可以有return语句，但是不能带表达式：

**return;**



# 类的成员方法

## ❖ 方法创建示例

```
int getMax(int a, int b){
    int temp = a;
    if(a<b){
        temp = b;
    }
    return temp;
}
```

```
void setData(int x, int y){
    this.x = x;
    this.y = y;
}
```

# 类的成员方法

## ❖ 方法调用时的参数传递

方法在被调用时，其参数的数据传递是**值传递**，即实际参数传值给形式参数。

实际参数也称为**实参**，是指调用方法时所带的参数，这些参数具有具体值。

形式参数也称为**形参**，它在方法创建时定义，在结构上起占位之用。

### 形式参数是简单类型

在方法调用时，实际参数将其存储单元的数据赋值给形式参数。当方法调用结束返回时，无论形式参数的值发生什么的变化都**不会影响实际参数的值**。

### 形式参数是引用类型

➤在方法中，引用类型的参数没有发生引用的改变，则形式参数对引用中的变量值改变自然**会影响到实际参数**引用中变量的值。

➤引用类型的参数发生了引用改变，它就不再对实际参数引用空间变量进行操作，则该引用类型的参数无论再做任何处理都**不会再**对实际参数引用空间的变量产生影响。

# 类的成员方法

## ❖ 形式参数是简单类型示例

```
void setData(int a, int b){  
    a = 10;  
    b = 20;  
}
```

```
x = 1;  
y = 2;  
p.setData(x, y);  
//实参x,y的值不受影响
```

参数a,b是**简单类型**，其值发生改变，但是**不会**影响调用的实参。

## ❖ 形式参数是引用类型示例1

```
void setRefer1(Point p){  
    p.x = 10;  
    p.y = 20;  
}
```

```
s.x = 1;  
s.y = 2;  
t.setRefer1(s);  
//实参s的x,y值受影响
```

参数p是**引用类型**，没有改变引用，其变量x,y值发生改变，**会**影响调用的实参。

## ❖ 形式参数是引用类型示例2

```
void setRefer2(Point p){  
    p = new Point();  
    p.x = 10;  
    p.y = 20;  
}
```

```
s.x = 1;  
s.y = 2;  
t.setRefer2(s);  
//实参s的x,y值不受影响
```

参数p是**引用类型**，改变引用，其变量x,y值发生改变，**不会**影响调用的实参。

# 方法过载

## ❖ 方法过载

- 在同一个类中创建的具有相同方法名，但是参数不同的方法。
- 参数不同：
  - 数量不同；
  - 数量相同，但是对应的类型不同。
- 方法过载也是多态性表现之一。
- 方法过载中的方法由调用时的实参决定调用的方法是那个。

## ❖ 示例：

```
void setData(int a, int b)
void setData(Point p)           //个数不同
void setData(float a, float b)  //类型不同
void setData()                  //个数不同
```

# 类的构造器

## ❖ 类构造器 (方法)

- 每一类都有自己的构造方法，或者称为类的构造器。构造方法是用来创建一个类的实例的。

- 具有特点和作用：

```
Point(int x, int y)
```

无返回类型

~~p.Point(10,20)~~

类名作构造方法名：

参数和语句体，但没有返回类型的声明。如果有返回类

p=new Point(10,20);

方法就再不是构造方法，而是

类的成员方法，所以不能用对

```
Point(int a,int b){
    x = a;
    y = b;
}
```

➤ 构造方法的调用是由new运算符实现；

```
Point(){
    this(0,0);
}
```

方法返回的是这个类的实例的引用；

方法中的语句实现对成员变量的初始化；

方法的方法重载。一个类可以有多个构造方法；

➤ 构造方法之间通过this()形式相互调用。

# 类的构造器

## ❖ 类构造器 (方法)

- 类的构造方法又可以分为两种

- 默认构造方法

- 非默认构造方法。

- **默认构造方法**是指不带参数的构造方法，可以有语句，也可以没有语句；

- **非默认构造方法**是指带参数的构造方法。

```
Point() {
    x = 0;
    y = 0;
}
```

```
Point(int a, int b) {
    x = a;
    y = b;
}
```

# 类的构造器

- ❖ 类创建时**没有创建构造器**，则在编译时编译器**自动为该类添加**一个默认构造器，即没有创建构造器的类可以直接使用默认构造器。
- ❖ 类创建时如果有任何**构造器被创建**，则在编译时编译器不再为该类创建默认构造器。即创建了构造方法的类，不再自动有一个默认的构造方法，或者说**默认构造器失效**，除非也创建一个不带参数的构造器。



# 成员变量初始化

## ❖ 类的成员变量初始化

### ■ 成员变量初始化三个步:

- 创建时分配存储空间赋上缺省的初值,

```
int a; //a有初始值
```

- 在成员变量声明时赋的初值;

```
int a = 10; //a赋初值10
```

- 在创建实例时, 由构造器的赋初值。

```
Point(int a, int b) {
    x = a;
    y = b;
}
```

类型	值
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
引用类型	null

# 实例成员和类成员

## ❖ 实例成员

- 通过创建实例才能访问和使用的成员；  
创建对象，由对象访问。例：**p.x = 10;**
- 定义时无**static**修饰的成员；  
例：**class Point{**  
    **int x, y; //x,y为实例成员变量**  
    **}**
- 实例成员不能用类名直接访问。  
例：**Point.x** 访问是不合法的。

# 实例成员和类成员

## ❖ 类成员

- 有 **static** 修饰的成员;

例: **static int a;**

- 也可以称为静态成员;

- 类成员: 可以用类名直接访问:

<类名> · <成员名>

例: **Point.a = 10;**

# 实例成员和类成员

## ❖ 类成员

- 类成员也可以用对象名来访问，但是该类的所有对象都共享类成员变量。

例：

```
Point p = new Point();
```

```
Point.a = 10;
```

```
p.a = 100;
```

类变量**a**既能用类**Point**访问，又能由对象**p**访问。

# 对象创建与使用

## ❖ 对象的生命周期:

创建、使用和销毁三个阶段。

■ 创建：分两步。

➤ 声明：

```
Point p;
```

➤ 实例化：

通过**new**运算符调用构造器，通过赋值=对这个实例引用。

```
p = new Point();
```

# 对象创建与使用

易混淆的三个基础概念：对象、实例和引用

## ❖ 对象：

- 是一个变量，它的存储开销是一个地址存储单元。

## ❖ 实例：

- 是在存储空间分配的存储堆。是垃圾回收的目标。

## ❖ 引用：

- 对象通过它存储的实例的起始地址对实例实现访问。

# 对象创建与使用

## ❖ 对象的生命周期:

### ■ 对象使用

#### ➤ 通过对象名对成员的访问:

<对象名>·<成员>

```
p.x = 100;
```

```
int temp = p.getData();
```

### ■ 销毁:

#### ➤ 垃圾回收: 实例开销的回收。

#### ➤ 由**JVM**自动完成。

#### ➤ 调用**finalize()**方法处理。



# 对象创建与使用举例

❖ 类Point有成员变量x和y，成员方法setData()

```
class Point{
    int x,y;
    Point(int a, int b){
        x = a;
        y = b;
    }
    Point(){
        this(0,0);
    }
    void setData(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```

# 对象创建与使用举例

对象声明

引用

实例化

```
TestPoint:
public class TestPoint{
    public static void main(String[] arg){
        Point p1 = new Point();//对象p1创建
        Point p2 = new Point();//对象p2创建
        p1.setData(10,20);
        p2.setData(11,22);
        p2 = p1;
        p1 = null;
    }
}
```

对象使用

垃圾回收: **p2**原实例化

**p1**原实例化不能回收  
**p2**仍在引用

# 思考问题

1

如何把客观世界中的对象转换为软件中的"对象"?

2

如何确定对象所需要的属性?

3

如何体现对象的行为和能力?

# 第4章作业

## 本章习题

### 习题1-10题

1-6题必做

7-8题中选一

9题选做

10题选做



# Q & A

电子科技大学信息与软件工程学院