



# Java

## 面向对象程序设计

软件学院 贾伟峰



# 面向对象三大特征

封装

代码封装在类、方法中。程序运行时，把类“实例化”为对象，让对象执行操作，面向对象编程。

继承

?

多态

?



# Java中的继承

## What

代表一种所属关系  
狗、猫属于动物类；  
杨树、柳树属于树类。

.....

狗、猫继承自动物；  
杨树、柳树继承自树。

## How

Dog extends Animal  
Postgraduate extends Student

01

02

03

04

## Why

写过的代码不想再写，能不能复用？  
Animal类写过，Dog类能不能复用？  
Student类写过，研究生类能不能复用？

## When

不同类之间有所属关系的时候。

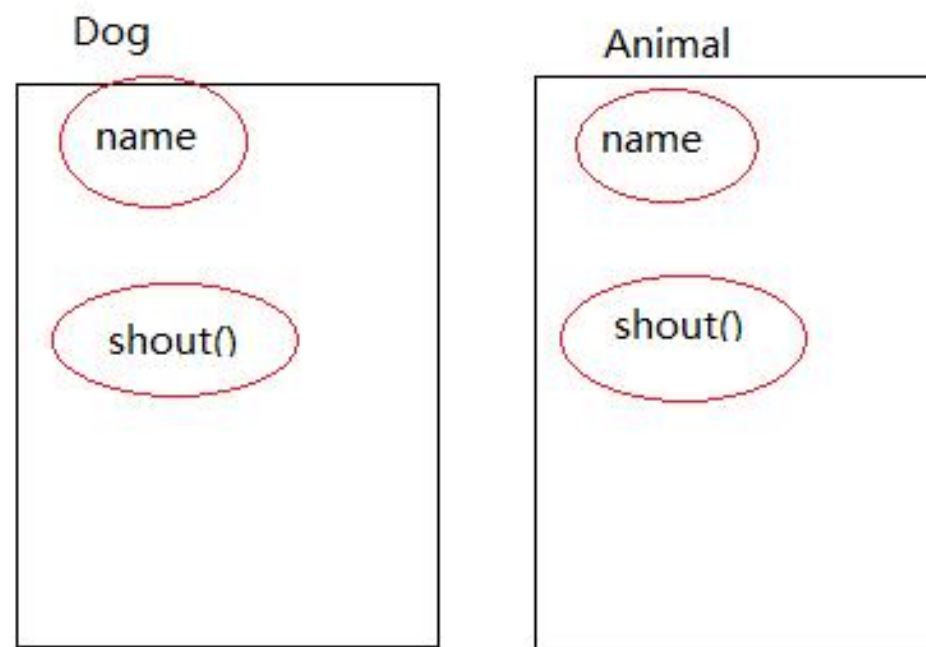
## 继承：添加新的方法

```
1 class Animal{
2     String name;
3
4     void shout(){
5         System.out.println("动物发出叫声");
6     }
7 }
8 class Dog extends Animal{
9     public void printName(){
10         System.out.println("name = " + name);
11     }
12 }
13
14 public class Example01{
15     public static void main(String[] args){
16         Dog dog = new Dog();
17         dog.name = "沙皮狗";
18         dog.printName();
19         dog.shout();
20     }
21 }
```

## 继承：重写父类方法

```
1 class Animal{
2     void shout() {
3         System.out.println("动物发出叫声");
4     }
5 }
6 class Dog extends Animal{
7     void shout() {
8         System.out.println("汪汪.....");
9     }
10 }
11
12 public class Example02{
13     public static void main(String[] args){
14         Dog dog = new Dog();
15         dog.shout();
16     }
17 }
```

有继承、有“创新”。子类在父类基础上“发扬光大”。



子类继承自父类的成员变量和方法，同样的名称，不同的拷贝，如何区分它们？





*super*

站在子类，“遥望”父类，super可以取到父类的成员变量和方法（包括构造方法）。

P118-120

## super与构造方法

```
1 class Animal{
2     //有带参数的构造方法，系统默认提供的无参数构造方法就失效了。
3     public Animal(String name){
4         System.out.println("我是一只"+name);
5     }
6 }
7 class Dog extends Animal{
8     public Dog(){
9         //此处不写调用父类构造方法的代码，相当于写上了super();而super()这个无参的父类构造方法
10        //是不存在的。因此此处不写代码，系统会报错。
11        super("沙皮狗");//super调用父类的构造方法。写上他就不会报错啦！
12    }
13 }
14 }
15 public class Example04{
16     public static void main(String[] args){
17         Dog dog = new Dog();
18     }
19 }
```

## super与构造方法

```
1 class Animal{
2     public Animal(){
3         System.out.println("我只是一只动物");
4     }
5     //有带参数的构造方法，系统默认提供的无参数构造方法就失效了，除非你自行提供。
6     public Animal(String name){
7         System.out.println("我是一只"+name);
8     }
9 }
10 class Dog extends Animal{
11     public Dog(){
12         //此处不写代码，相当于写上了super();而super()这个无参的父类构造方法，现在已经有了。
13     }
14 }
15
16
17 public class Example05{
18     public static void main(String[] args){
19         Dog dog = new Dog();
20     }
21 }
```

Object: 所有类的父类, p134-135, toString()方法



不想被继承，不想被重写，不想被改变。

# final关键词

## 写final

类、方法、变量

## 不写final

类、方法、变量

VS

类无法继承

方法无法被重写

变量首次赋值后不再可变

类可以继承

方法可以重写

变量是可变的

# 类、方法的功能不确定，怎么办？

```
1 public class Main{  
2     public static void main(String args[]){  
3         Tv tv;  
4         String brandName = XMLUtilTV.getBrandName();  
5         tv = TVFactory.produceTv(brandName);  
6         tv.play();  
7     }  
8 }
```





A diagram illustrating the features of an abstract class. It consists of a large red circle with the text '抽象类' (Abstract Class) in white. Surrounding this central circle are four smaller dark gray circles, each containing a white number: '01' at the top, '02' at the top-right, '03' at the bottom-right, and '04' at the bottom. To the right of this diagram is a list of four bullet points describing the features of abstract classes.

# 抽象类

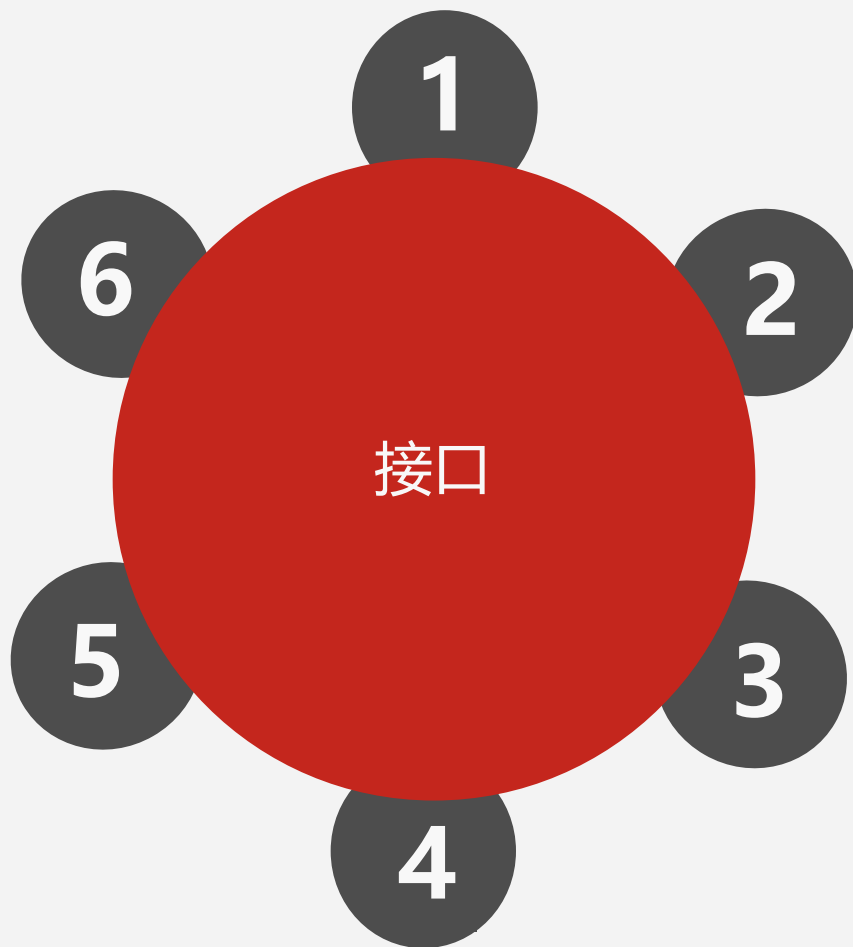
- 特征：功能不确定

- 抽象方法：只有名字，无代码。所在类为抽象类

- 抽象类：可以包含，也可以不包含抽象方法

- 关键词：abstract

```
1  abstract class Animal{  
2      abstract void shout();  
3  }  
4  class Dog extends Animal{  
5      void shout(){  
6          System.out.println("汪汪.....");  
7      }  
8  }  
9  public class Example10{  
10     public static void main(String[] args){  
11         Dog dog = new Dog();  
12         dog.shout();  
13     }  
14 }
```



## 更抽象

所有方法都是抽象的  
(public abstract)  
所有变量都是全局变量  
(public static final)

## 无法实例化

## 用类实现接口

implements  
P126, 例4-11  
P127, 例4-12  
P128, 倒数第3行,  
实现多个接口

## interface

## 接口可继承接口

extends p127, 第8行

## 可继承类的同时实现接口

extends 类  
implements 接口  
p129

```
1 interface Animal{//接口中所有的方法都是抽象的。将来是要被其他类实现的。  
2     int ID=1;  
3     void run();  
4     void breathe();  
5 }  
6 class Dog implements Animal{  
7     public void breathe(){  
8         System.out.println("狗在呼吸");  
9     }  
10    public void run(){  
11        System.out.println("狗在跑");  
12    }  
13 }  
14 public class Example11{  
15     public static void main(String[] args){  
16         Dog dog = new Dog();  
17         dog.breathe();  
18         dog.run();  
19     }  
20 }
```



“继承” 之下有 “多态”

## 多态：子类当作父类去使用，运行的是子类的方法（运行时绑定）

```
1 interface Animal{
2     void shout();
3 }
4 class Cat implements Animal{
5     public void shout(){
6         System.out.println("喵喵.....");
7     }
8 }
9 class Dog implements Animal{
10    public void shout(){
11        System.out.println("汪汪.....");
12    }
13 }
14 public class Example13{
15     public static void main(String[] args){
16         Animal an1 = new Cat();
17         Animal an2 = new Dog();
18         animalShout(an1);
19         animalShout(an2);
20     }
21     public static void animalShout(Animal an){
22         an.shout();
23     }
24 }
```

## 多态：使用匿名内部类作为参数传递

```
1 //匿名内部类
2 interface Animal{
3     void shout();
4 }
5 public class Example19{
6     public static void main(String[] args){
7         animalShout(new Animal(){
8             public void shout(){
9                 System.out.println("喵喵.....");
10            }
11        });
12
13        animalShout(new Animal(){
14            public void shout(){
15                System.out.println("汪汪.....");
16            }
17        });
18    }
19    public static void animalShout(Animal an){
20        an.shout();
21    }
22 }
```

## 多态应用——简单工厂模式\*

```
1 public abstract class Tv{
2     public abstract void play();
3 }
```

```
1 public class HaierTv extends Tv{
2     public void play(){
3         System.out.println("This is HaierTv.");
4     }
5 }
```

```
1 public class HisenseTv extends Tv{
2     public void play(){
3         System.out.println("This is HisenseTv.");
4     }
5 }
```

```
1 public class TVFactory{
2     public static Tv produceTv(String brand){
3         if(brand.equals("HaierTv"))
4             return new HaierTv();
5
6         if(brand.equals("HisenseTv"))
7             return new HisenseTv();
8
9         return null;
10    }
11 }
```

```
1 <?xml version="1.0"?>
2 <config>
3     <brandName>HisenseTv</brandName>
4 </config>
```

```
1 import javax.xml.parsers.*;
2 import org.w3c.dom.*;
3 import org.xml.sax.SAXException;
4 import java.io.*;
5 public class XMLUtilTV{
6     public static String getBrandName(){
7         try{
8             DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
9             DocumentBuilder builder = dFactory.newDocumentBuilder();
10            Document doc = builder.parse(new File("configTV.xml"));
11
12            NodeList nl = doc.getElementsByTagName("brandName");
13            Node classNode = nl.item(0).getFirstChild();
14            String brandName = classNode.getNodeValue().trim();
15
16            return brandName;
17        } catch (Exception e){
18            System.out.println(e.getMessage());
19            return null;
20        }
21    }
22 }
23 }
```

```
1 public class Main{
2     public static void main(String args[]){
3         Tv tv;
4         String brandName = XMLUtilTV.getBrandName();
5         tv = TVFactory.produceTv(brandName);
6         tv.play();
7     }
8 }
```



## 多态应用——工厂方法模式（多分支语句的多态替换）\*

```
1 public abstract class Tv{
2     public abstract void play();
3 }
```

```
1 public class HaierTv extends Tv{
2     public void play(){
3         System.out.println("This is HaierTv.");
4     }
5 }
```

```
1 public class HisenseTv extends Tv{
2     public void play(){
3         System.out.println("This is HisenseTv.");
4     }
5 }
```

```
1 public interface TVFactory{
2     public TV produceTV();
3 }
```

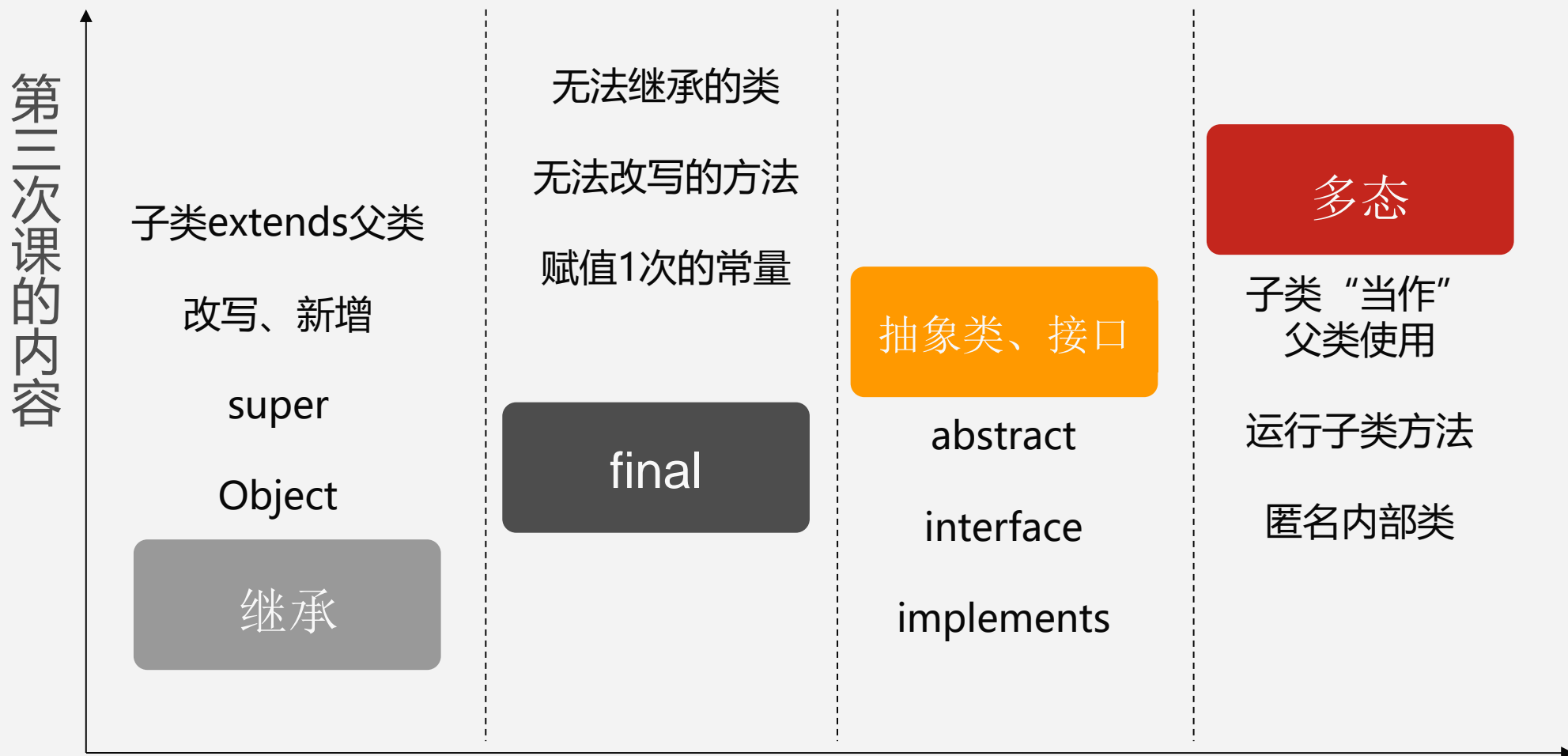
```
1 public class HaierTVFactory implements TVFactory{
2     public TV produceTV(){
3         return new HaierTV();
4     }
5 }
```

```
1 public class HisenseTVFactory implements TVFactory{
2     public TV produceTV(){
3         return new HisenseTV();
4     }
5 }
```

```
1 <?xml version="1.0"?>
2 <config>
3     <className>HaierTVFactory</className>
4 </config>
```

```
1 import javax.xml.parsers.*;
2 import org.w3c.dom.*;
3 import org.xml.sax.SAXException;
4 import java.io.*;
5 public class XMLUtil{
6     public static Object getBean(){
7         try{
8             DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
9             DocumentBuilder builder = dFactory.newDocumentBuilder();
10            Document doc = builder.parse(new File("config.xml"));
11
12            NodeList nl = doc.getElementsByTagName("className");
13            Node classNode = nl.item(0).getFirstChild();
14            String cName = classNode.getNodeValue();
15
16            Class c = Class.forName(cName);
17            Object obj = c.newInstance();
18            return obj;
19        }
20        catch(Exception e){
21            e.printStackTrace();
22            return null;
23        }
24    }
25 }
```

```
1 public class Main{
2     public static void main(String args[]){
3         TV tv;
4         TVFactory factory;
5         factory = (TVFactory)XMLUtil.getBean();
6         tv = factory.produceTV();
7         tv.play();
8     }
9 }
```



## 课下任务

### 一、小组合作

7人一组。小组成员要通力合作，保证每位同学提交作业，以免影响小组成绩。下次上课前每小组推荐1名代表，参与最佳作业的评选。

### 二、作业

1.实现教材示例代码，并用github或者gitee提交，代码要有相应提交记录（提交以上次代码为基础，需先把远程库pull下来之后，在本地添加新的代码提交）。

2.结合课件例子和网络资源，深入理解多态，并编写Java代码进行相应实践：实现简单工厂模式和工厂方法模式，并使用github或者gitee托管，代码要有相应提交记录（提交以上次代码为基础，需先把远程库pull下来之后，在本地添加新的代码提交）；

3.结合以上作业谈谈你对继承和多态的认识（内要粘贴本人github或者gitee账户链接）。严禁抄袭！