

# day10面向对象多态

- 学习目标
  - 对象的多态性
  - 多态的必要因素
  - 多态的语法格式
  - 多态中成员的特性
  - 多态中的转型
  - 抽象类定义
  - 抽象方法定义
  - 抽象类使用
  - 抽象类的成员特点
  - 抽象类的多态调用
  - 抽象类的继承体系价值

## 1. 对象的多态性

引入：生活中的多态性！你自己的身份是学生，你的身份职场精英，患者。在不同的时期不同环境，状态是不同的。

生活中的多态性：一个事物具备的不同形态。

### 1.1 对象多态性前提

- 必须有继承或者是接口实现
- 必须有方法的重写

**多态的语法规则：**父类或者接口的引用指向自己的子类的对象

```
1 | 父类 变量(对象名) = new 子类对象(); //多态写法
```

对象调用方法，执行的子类的方法重写

### 1.2 多态中成员的特点

- 多态中成员变量的特点
  - 编译：父类中没有成员变量，编译失败
  - 运行：运行父类中的成员变量
- 多态中成员方法的特点
  - 编译：父类中没有成员方法，编译失败
  - 运行：运行子类的方法重写
- 简练：成员方法编译看左边，运行看右边。成员变量都是左边

```
Person p = new Student();
```

```

1 public class Person {
2     String s = "父类成员";
3
4     public void eat(){
5         System.out.println("人在吃饭");
6     }
7 }

```

```

1 public class Student extends Person {
2     String s = "子类成员";
3
4     public void eat(){
5         System.out.println("学生吃饭");
6     }
7 }

```

```

1 public static void main(String[] args) {
2     Person p = new Student();
3     //对象p,子类对象,调用成员变量s
4     System.out.println(p.s);
5     //子类对象调用方法
6     p.eat();
7 }

```

```

public class Person {
    String s = "父类成员";

    public void eat() {
        System.out.println("人在吃饭");
    }
}

public class Student extends Person {
    String s = "子类成员";

    public void eat() {
        System.out.println("学生吃饭");
    }
}

```

```

public static void main(String[] args) {
    Person p = new Student();

    System.out.println(p.s);

    p.eat();
}

```

类类型    对象, 地址

Person p = new Student();    定义变量

javac 编译器, 检查语法  
Person类存在, new对象不存在的

System.out.println(p.s);  
Person中检查有没有这个变量, 或者是方法

Person p = new Student();

p 父类的类型

成员变量的话, 找父类的, 找super位置

p.eat();

继承: 扩展和延伸

重写: 父类的功能实现了扩展

Person p = new Student();

p.eat();

eat方法, 运行了父类的话

设计思想看: 扩展全无

子类可以删除, 没有扩展延伸的功能

方法的执行: JVM动态绑定到子类对象

## 1.3 多态的转型

多态的程序中, 不能调用子类的特有成员!!

只能调用子类父类的共有成员!!

```

1 转后类型 变量名 = (转后类型)要转的数据; //公式

```

```

1 public static void main(String[] args) {
2     //创建对象,多态性
3     //父类 = new 任意子类对象() 扩展
4     Animal animal = new Cat();
5     animal.eat();
6     //Cat类的特有功能 catchMouse()方法
7     //类型转换,强制
8     //Cat提升为了Animal,转回Cat类型
9     Cat c = (Cat)animal;
10    c.catchMouse();
11 }

```

#### 基本数据类型的转换

自动转换: byte -> short -> int -> long -> float -> double

int a = 1; byte b = 1; b+a 结果是int

short s = 1;

int i = 1;

s = s + 1      s+1 结果是 int类型

s = (short)(s+1) OK的写法

类型强制转换的公式

转后类型 变量名 = (转后类型)要转的数据

#### 引用数据类型的转换

Animal animal = new Cat(); =符合双方类型不同

上面的程序, 出现类型转换, 自动的

自动类型转换, 小类型转大类型. 父类为大, 子类为小

Animal animal = new Cat(); ↑

Cat类型自动转换, 提升为了父类Animal类型: 类型向上转型

需要使用子类的特有成员: 必须进行强制转换

目标: 是已经提升为父类的Cat类型, 在转回Cat类型

转后类型 变量名 = (转后类型)要转的数据

Cat 变量名 = (Cat)animal; ↓ 类型向下转型

```

Cat c = (Cat)animal;
c.catchMouse();

```

补充: 多态性提升扩展性, 是否需要强制转换, 根据实际功能需求来决定.

## 1.4 多态中的转型异常

异常ClassCastException 类型转换异常, 在多态中经常发生.

是在进行类型的强制转换的时候发生, 我们现在的案例是Dog不能转成Cat.

需要解决这个异常: 对象是Cat转Cat, 是Dog换Dog

运算符: 比较运算符, 结果是boolean类型

运算符是关键字 instanceof

instanceof的语法格式:

```

1 对象名 instanceof 类的名字
2 解析: 比较这个对象, 是不是由这个类产生的
3 c instanceof Cat 解释: c对象是不是Cat类产生的, 如果是结果就是true

```

强制类型转换之前的安全性判断

```

1 public static void main(String[] args) {
2     //多态创建对象
3     Animal animal = new Dog();
4     animal.eat();
5     //判断 animal是不是Cat类的对象
6     //boolean b = animal instanceof Dog ;

```

```

7      //System.out.println(b);
8      //调用子类的特有方法
9      if (animal instanceof Cat){
10         //if为true,强制转换为Cat
11         Cat c = (Cat)animal;
12         c.catchMouse();
13     }
14     if (animal instanceof Dog){
15         Dog d = (Dog)animal;
16         d.lookHome();
17     }
18 }

```

## 1.5 多态的转型案例

```

1  public static void main(String[] args) {
2      //创建对象,多态
3      Person p1 = new Faculty();
4      //p1对象的属性,赋值本科 degree 子类的特有成员
5      //判断p1对象是不是Faculty类产生
6      if (p1 instanceof Faculty){
7          Faculty f = (Faculty)p1;
8          f.setDegree("本科");
9          System.out.println(f.getDegree());
10     }
11     Person p2 = new Staff();
12     //判断p2对象是不是Staff类产生
13     if (p2 instanceof Staff){
14         Staff s = (Staff)p2;
15         s.setDuty("职员");
16         System.out.println( s.getDuty());
17     }
18 }

```

## 2. 抽象类 abstract

抽象的概念：凡是说不清楚的都是抽象

例子：我买了一台手机,买了一直笔,都是抽象概念.

具体: 华为Meta40Pro ,金属, 16G+512

程序中：我知道这个功能存在,但是怎么完成就说不清楚,程序中也出现了抽象.

### 2.1 抽象方法定义

使用关键字 abstract定义抽象方法

```

1  权限修饰符 abstract 返回值类型 方法名字(参数列表) ;
2  abstract关键字
3  抽象方法没有方法体，不需要{},直接分号结束

```

当一个类中的方法是抽象方法的时候,这个类必须是抽象类,在类的关键字class前面使用abstract修饰.

```
1 public abstract class 类名{}
```

```
1 public abstract class Animal {
2     /**
3      * 动物吃什么?
4      * 说不清楚,抽象,可以不说
5      */
6     public abstract void eat();
7 }
8
```

## 2.2 抽象类的使用方式

- 抽象类不能实例化对象,不能new对象.
  - 为什么不能建立对象,类中有没有主体的方法存在,建立对象调用抽象方法是绝对的错误,因此不能建立对象.
- 需要子类继承抽象类,重写抽象方法.
- **创建子类对象**
- 使用多态性创建对象,调用方法执行子类的重写

```
1 public class Cat extends Animal{
2
3     /**
4      * 重写父类的方法
5      * 去掉修饰符 abstract
6      * 添加主体 {}
7      */
8     public void eat(){
9         System.out.println("猫吃鱼");
10    }
11 }
```

```
1 public static void main(String[] args) {
2     //创建Animal的子类对象
3     Animal animal = new Cat();
4     //eat方法不可能执行父类,运行子类的重写
5     animal.eat();
6 }
```

## 2.3 抽象类中成员的定义

### 2.3.1 抽象类中能否定义成员变量

可以定义成员变量,成员变量私有修饰,提供方法 get/set,由子类的对象使用

```

1 public abstract class Animal {
2     //抽象类中能否定义成员变量
3     private String name;
4     public abstract void eat();
5
6     public String getName() {
7         return name;
8     }
9
10    public void setName(String name) {
11        this.name = name;
12    }
13 }

```

```

1 public static void main(String[] args) {
2     Animal animal = new Cat();
3     animal.eat();
4     //animal对象调用方法 get/ set
5     animal.setName("tom");
6     String name = animal.getName();
7     System.out.println(name);
8 }

```

### 2.3.2 抽象类中有构造方法吗

抽象类中有构造方法,不写有默认的

```

1 public abstract class Animal {
2
3     public Animal(){
4         System.out.println("Animal的构造方法");
5     }
6
7     public Animal(String name){
8         this.name = name;
9         System.out.println("有参数String的构造方法");
10    }
11
12    //抽象类中能否定义成员变量
13    private String name;
14    public abstract void eat();
15
16    public String getName() {
17        return name;
18    }
19
20    public void setName(String name) {
21        this.name = name;
22    }
23 }

```

```

1 public class Cat extends Animal {
2
3     public Cat(){
4         //调用父类的有参数构造方法
5         super("张三");
6     }
7
8     @Override
9     public void eat() {
10        System.out.println("猫吃鱼");
11    }
12 }
13

```

### 2.3.3 抽象中能否不定义抽象方法

抽象类中,可以不定义出抽象方法.

但是,如果有抽象方法存在,这个类必须是抽象类

## 2.4 子类还是抽象类的问题

当一个子类继承一个抽象类的时候,子类必须重写全部的抽象方法.假如子类重写了部分抽象方法,这个子类依然还是抽象类.

```

1 public abstract class Animal {
2     public abstract void eat();
3     public abstract void sleep();
4 }

```

```

1 /**
2  * Cat继承父类Animal,Cat类拥有了父类的成员
3  * 父类有什么,我就有什么
4  */
5 public abstract class Cat extends Animal {
6     public void eat(){}
7     /**
8      * 方法sleep没有重写
9      * 还是一个抽象的方法
10     */
11     // public abstract void sleep();
12 }

```

## 2.5 员工案例

```

1 /**
2  * 公司类
3  * 定义的是所有员工的共性内容
4  */
5 public abstract class Company {
6     private String name; //员工姓名
7     private String id; // 员工编号,唯一标识
8

```

```
9      //工作行为,具体到某个岗位是不同,无法写出具体的工作内容
10     public abstract void work();
11
12
13     public String getName() {
14         return name;
15     }
16
17     public void setName(String name) {
18         this.name = name;
19     }
20
21     public String getId() {
22         return id;
23     }
24
25     public void setId(String id) {
26         this.id = id;
27     }
28 }
```

```
1  /**
2   * 研发部类
3   */
4  public class Development extends Company{
5      //重写工作的抽象方法
6      //work方法中,输出自己的姓名和工号呢
7      @Override
8      public void work() {
9          //调用父类的方法
10         System.out.println(super.getName()+"::"+super.getId()+"研发部的员工在开
发程序");
11     }
12 }
```

```
1  /**
2   * 定义财务部类
3   */
4  public class Financial extends Company {
5      @Override
6      public void work() {
7          System.out.println(super.getName()+"::"+super.getId()+"财务部员工在算
账");
8      }
9  }
```

```
1  public static void main(String[] args) {
2      //创建对象,子类对象,多态性
3      Company c1 = new Development();
4      //父类的方法,属性赋值
5      c1.setName("张三");
6      c1.setId("研发部001");
7      //System.out.println(c1.getName() + "::"+c1.getId());
8      c1.work();
9  }
```



```
9
10     Company c2 = new Financial();
11     c2.setName("李四");
12     c2.setId("财务部001");
13     c2.work();
14 }
```

## 3. 接口 interface

### 3.1 接口无处不在

身边的接口有哪些,笔记本上USB接口,HDMI,TypeC接口,插座

USB接口:连接鼠标,键盘,摄像头,手机,移动硬盘,电风扇.设备的工作原理不同,但是都可以连接到USB接口上,完成他的任务.说明了一个问题:这些设备都满足USB的接口规范!!

**接口:就是一个规范,或者称为标准**,无论什么设备,只要符合接口标准,就可以正常使用.

接口的扩展性很强大.

### 3.2 Java中接口定义

当一个抽象类中的所有方法全部是抽象的时候,可以将这个抽象类换一个更加贴切的名词,叫他接口. 接口是特殊的抽象类.

定义接口,使用关键字 interface

语法规则:

```
1 | public interface 接口名{}
```

接口在编译后,依然还是.class文件

### 3.3 接口中成员定义 (JDK1.7 版本)

- 成员变量
  - 成员变量的定义是具有固定格式
  - 成员变量的修饰符是固定 public static final

```
1 | public static final 数据类型 变量名 = 值 ;
```

- 成员方法
  - 成员方法的定义是具有固定格式
  - 成员方法的修饰符固定为 public abstract

```
1 | public abstract 返回值类型 方法名(参数列表) ;
```

### 3.4 接口的使用方式

- 接口不能建立对象,不能new
- 需要定义类,实现接口(继承类,在接口中称为实现,理解为继承)
  - 实现接口,使用新的关键字 implements
  - 实现的格式 `class 类 implements 接口名{}`
- 重写接口中的抽象方法
- 创建子类的对象

```
1  /**
2   * 定义好的接口
3   */
4  public interface MyInterFace {
5      //接口的成员变量
6      public static final int A = 1;
7      //接口的成员方法
8      public abstract void myInter();
9  }
```

```
1  /**
2   * 定义MyInterFace接口的实现类
3   * 重写接口的抽象方法
4   */
5  public class MyInterFaceImpl implements MyInterFace{
6      public void myInter(){
7          System.out.println("实现类实现接口,重写方法");
8      }
9  }
```

```
1  public static void main(String[] args) {
2      //创建对象,多态性,创建接口实现类的对象
3      MyInterFace my = new MyInterFaceImpl();
4      my.myInter();
5      //输出接口中的成员A的值
6      System.out.println(my.A);
7  }
```

### 3.5 接口的多实现

类和类之间单继承,局限性的问题.接口的出现,是对单继承的改良,允许一个类同时实现多个接口.

语法格式:

```
1  class 类名 implements 接口A,接口B{}
```

实现类,重写实现的多有接口中的抽象方法

```
1  public interface A {
2      public abstract void a();
3  }
```

```
1 public interface B {
2     public abstract void b();
3 }
```

```
1 /**
2  * 实现接口A和B
3  */
4 public class C implements A,B{
5     @Override
6     public void a() {
7         System.out.println("重写A接口方法");
8     }
9
10    @Override
11    public void b() {
12        System.out.println("重写B接口方法");
13    }
14 }
15
```

```
1 public static void main(String[] args) {
2     C c = new C();
3     c.a();
4     c.b();
5 }
```