

1.多态

1.1多态的概述（记忆）

- 什么是多态

同一个对象，在不同时刻表现出来的不同形态

- 多态的前提

- 要有继承或实现关系
- 要有方法的重写
- 要有父类引用指向子类对象

1.2多态中的成员访问特点（记忆）

- 成员访问特点

- 成员变量

编译看父类，运行看父类

- 成员方法

编译看父类，运行看子类

- 代码演示

- 动物类

```
public class Animal {  
    public int age = 40;  
  
    public void eat() {  
        System.out.println("动物吃东西");  
    }  
}
```

- 猫类

```
public class Cat extends Animal {  
    public int age = 20;  
    public int weight = 10;  
  
    @Override  
    public void eat() {  
        System.out.println("猫吃鱼");  
    }  
  
    public void playGame() {  
        System.out.println("猫捉迷藏");  
    }  
}
```

- 测试类

```
public class AnimalDemo {  
    public static void main(String[] args) {  
        //有父类引用指向子类对象  
        Animal a = new Cat();  
  
        System.out.println(a.age);  
        //        System.out.println(a.weight);  
  
        a.eat();  
        //        a.playGame();  
    }  
}
```

1.3 多态的好处和弊端（记忆）

- 好处

提高程序的扩展性。定义方法时候，使用父类型作为参数，在使用的时候，使用具体的子类型参与操作

- 弊端

不能使用子类的特有成员

1.4 多态中的转型（应用）

- 向上转型

父类引用指向子类对象就是向上转型

- 向下转型

格式：子类型 对象名 = (子类型)父类引用;

- 代码演示

- 动物类

```
public class Animal {  
    public void eat() {  
        System.out.println("动物吃东西");  
    }  
}
```

- 猫类

```

public class Cat extends Animal {
    @Override
    public void eat() {
        System.out.println("猫吃鱼");
    }

    public void playGame() {
        System.out.println("猫捉迷藏");
    }
}

```

◦ 测试类

```

public class AnimalDemo {
    public static void main(String[] args) {
        //多态
        //向上转型
        Animal a = new Cat();
        a.eat();
        //
        a.playGame();

        //向下转型
        Cat c = (Cat)a;
        c.eat();
        c.playGame();
    }
}

```

1.5多态的案例（应用）

- 案例需求

请采用多态的思想实现猫和狗的案例，并在测试类中进行测试

- 代码实现

- 动物类

```
public class Animal {
    private String name;
    private int age;

    public Animal() {
    }

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void eat() {
        System.out.println("动物吃东西");
    }
}
```

- o 猫类

```
public class Cat extends Animal {

    public Cat() {
    }

    public Cat(String name, int age) {
        super(name, age);
    }

    @Override
    public void eat() {
        System.out.println("猫吃鱼");
    }
}
```

- o 狗类

```

public class Dog extends Animal {

    public Dog() {
    }

    public Dog(String name, int age) {
        super(name, age);
    }

    @Override
    public void eat() {
        System.out.println("狗吃骨头");
    }
}

```

o 测试类

```

public class AnimalDemo {
    public static void main(String[] args) {
        //创建猫类对象进行测试
        Animal a = new Cat();
        a.setName("加菲");
        a.setAge(5);
        System.out.println(a.getName() + "," + a.getAge());
        a.eat();

        a = new Cat("加菲", 5);
        System.out.println(a.getName() + "," + a.getAge());
        a.eat();
    }
}

```

2.抽象类

2.1抽象类的概述（理解）

当我们在做子类共性功能抽取时，有些方法在父类中并没有具体的体现，这个时候就需要抽象类了！

在Java中，一个没有方法体的方法应该定义为抽象方法，而类中如果有抽象方法，该类必须定义为抽象类！

2.2抽象类的特点（记忆）

- 抽象类和抽象方法必须使用 **abstract** 关键字修饰

```

//抽象类的定义
public abstract class 类名 {}

//抽象方法的定义
public abstract void eat();

```

- 抽象类中不一定有抽象方法，有抽象方法的类一定是抽象类

- 抽象类不能实例化

抽象类如何实例化呢？参照多态的方式，通过子类对象实例化，这叫抽象类多态

- 抽象类的子类

要么重写抽象类中的所有抽象方法

要么是抽象类

2.3 抽象类的成员特点（记忆）

- 成员的特点

- 成员变量

- 既可以是变量
- 也可以是常量

- 构造方法

- 空参构造
- 有参构造

- 成员方法

- 抽象方法
- 普通方法

- 代码演示

- 动物类

```
public abstract class Animal {  
  
    private int age = 20;  
    private final String city = "北京";  
  
    public Animal() {}  
  
    public Animal(int age) {  
        this.age = age;  
    }  
  
    public void show() {  
        age = 40;  
        System.out.println(age);  
        // city = "上海";  
        System.out.println(city);  
    }  
  
    public abstract void eat();  
  
}
```

- 猫类

```
public class Cat extends Animal {  
    @Override  
    public void eat() {  
        System.out.println("猫吃鱼");  
    }  
}
```

- 测试类

```
public class AnimalDemo {  
    public static void main(String[] args) {  
        Animal a = new Cat();  
        a.eat();  
        a.show();  
    }  
}
```

2.4抽象类的案例（应用）

- 案例需求

请采用抽象类的思想实现猫和狗的案例，并在测试类中进行测试

- 代码实现

- 动物类

```
public abstract class Animal {  
    private String name;  
    private int age;  
  
    public Animal() {  
    }  
  
    public Animal(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    public abstract void eat();  
}
```

- 猫类

```
public class Cat extends Animal {  
  
    public Cat() {  
    }  
  
    public Cat(String name, int age) {  
        super(name, age);  
    }  
  
    @Override  
    public void eat() {  
        System.out.println("猫吃鱼");  
    }  
}
```

- 狗类


```

public class Dog extends Animal {

    public Dog() {
    }

    public Dog(String name, int age) {
        super(name, age);
    }

    @Override
    public void eat() {
        System.out.println("狗吃骨头");
    }
}

```

○ 测试类

```

public class AnimalDemo {
    public static void main(String[] args) {
        //创建对象，按照多态的方式
        Animal a = new Cat();
        a.setName("加菲");
        a.setAge(5);
        System.out.println(a.getName()+" "+a.getAge());
        a.eat();
        System.out.println("-----");

        a = new Cat("加菲",5);
        System.out.println(a.getName()+" "+a.getAge());
        a.eat();
    }
}

```

3. 接口

3.1 接口的概述（理解）

接口就是一种公共的规范标准，只要符合规范标准，大家都可以通用。

Java中的接口更多的体现在对行为的抽象！

3.2 接口的特点（记忆）

- 接口用关键字interface修饰

```

public interface 接口名 {}

```

- 类实现接口用implements表示

```
public class 类名 implements 接口名 {}
```

- 接口不能实例化

接口如何实例化呢？参照多态的方式，通过实现类对象实例化，这叫接口多态。

多态的形式：具体类多态，抽象类多态，接口多态。

- 接口的子类

要么重写接口中的所有抽象方法

要么子类也是抽象类

3.3接口的成员特点（记忆）

- 成员特点

- 成员变量

只能是常量 默认修饰符：public static final

- 构造方法

没有，因为接口主要是扩展功能的，而没有具体存在

- 成员方法

只能是抽象方法

默认修饰符：public abstract

关于接口中的方法，JDK8和JDK9中有一些新特性，后面再讲解

- 代码演示

- 接口

```
public interface Inter {  
    public int num = 10;  
    public final int num2 = 20;  
    //    public static final int num3 = 30;  
    int num3 = 30;  
  
    //    public Inter() {}  
  
    //    public void show() {}  
  
    public abstract void method();  
    void show();  
}
```

- 实现类

```

public class InterImpl extends Object implements Inter {
    public InterImpl() {
        super();
    }

    @Override
    public void method() {
        System.out.println("method");
    }

    @Override
    public void show() {
        System.out.println("show");
    }
}

```

o 测试类

```

public class InterfaceDemo {
    public static void main(String[] args) {
        Inter i = new InterImpl();
        // i.num = 20;
        System.out.println(i.num);
        // i.num2 = 40;
        System.out.println(i.num2);
        System.out.println(Inter.num);
    }
}

```

3.4接口的案例（应用）

- 案例需求

对猫和狗进行训练，他们就可以跳高了，这里加入跳高功能。

请采用抽象类和接口来实现猫狗案例，并在测试类中进行测试。

- 代码实现

- o 动物类

```
public abstract class Animal {  
    private String name;  
    private int age;  
  
    public Animal() {  
    }  
  
    public Animal(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    public abstract void eat();  
}
```

- 跳高接口

```
public interface Jumping {  
    public abstract void jump();  
}
```

- 猫类

```

public class Cat extends Animal implements Jumping {

    public Cat() {
    }

    public Cat(String name, int age) {
        super(name, age);
    }

    @Override
    public void eat() {
        System.out.println("猫吃鱼");
    }

    @Override
    public void jump() {
        System.out.println("猫可以跳高了");
    }

}

```

o 测试类

```

public class AnimalDemo {
    public static void main(String[] args) {
        //创建对象，调用方法
        Jumping j = new Cat();
        j.jump();
        System.out.println("-----");

        Animal a = new Cat();
        a.setName("加菲");
        a.setAge(5);
        System.out.println(a.getName()+" "+a.getAge());
        a.eat();
        //    a.jump();

        a = new Cat("加菲", 5);
        System.out.println(a.getName()+" "+a.getAge());
        a.eat();
        System.out.println("-----");

        Cat c = new Cat();
        c.setName("加菲");
        c.setAge(5);
        System.out.println(c.getName()+" "+c.getAge());
        c.eat();
        c.jump();
    }
}

```

3.5类和接口的关系（记忆）

- 类与类的关系

继承关系，只能单继承，但是可以多层继承

- 类与接口的关系

实现关系，可以单实现，也可以多实现，还可以在继承一个类的同时实现多个接口

- 接口与接口的关系

继承关系，可以单继承，也可以多继承

3.6 抽象类和接口的区别（记忆）

- 成员区别

- 抽象类

变量,常量; 有构造方法; 有抽象方法,也有非抽象方法

- 接口

常量; 抽象方法

- 关系区别

- 类与类

继承, 单继承

- 类与接口

实现, 可以单实现, 也可以多实现

- 接口与接口

继承, 单继承, 多继承

- 设计理念区别

- 抽象类

对类抽象, 包括属性、行为

- 接口

对行为抽象, 主要是行为

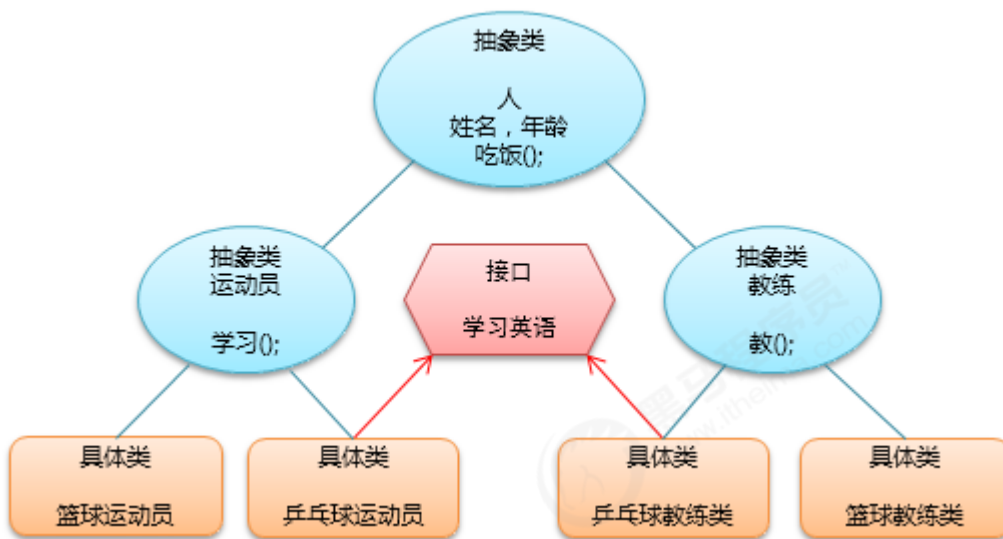
4. 综合案例

4.1 案例需求（理解）

我们现在有乒乓球运动员和篮球运动员，乒乓球教练和篮球教练。

为了出国交流，跟乒乓球相关的人员都需要学习英语。

请用所学知识分析，这个案例中有哪些具体类，哪些抽象类，哪些接口，并用代码实现。



4.2 代码实现（应用）

- 抽象人类

```
public abstract class Person {
    private String name;
    private int age;

    public Person() {
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public abstract void eat();
}
```

- 抽象运动员类

```

public abstract class Player extends Person {
    public Player() {
    }

    public Player(String name, int age) {
        super(name, age);
    }

    public abstract void study();
}

```

- 抽象教练类

```

public abstract class Coach extends Person {
    public Coach() {
    }

    public Coach(String name, int age) {
        super(name, age);
    }

    public abstract void teach();
}

```

- 学英语接口

```

public interface SpeakEnglish {
    public abstract void speak();
}

```

- 篮球教练

```

public class BasketballCoach extends Coach {
    public BasketballCoach() {
    }

    public BasketballCoach(String name, int age) {
        super(name, age);
    }

    @Override
    public void teach() {
        System.out.println("篮球教练教如何运球和投篮");
    }

    @Override
    public void eat() {
        System.out.println("篮球教练吃羊肉，喝羊奶");
    }
}

```


- 乒乓球教练

```
public class PingPangCoach extends Coach implements SpeakEnglish {  
  
    public PingPangCoach() {  
    }  
  
    public PingPangCoach(String name, int age) {  
        super(name, age);  
    }  
  
    @Override  
    public void teach() {  
        System.out.println("乒乓球教练教如何发球和接球");  
    }  
  
    @Override  
    public void eat() {  
        System.out.println("乒乓球教练吃小白菜，喝大米粥");  
    }  
  
    @Override  
    public void speak() {  
        System.out.println("乒乓球教练说英语");  
    }  
}
```

- 乒乓球运动员

```

public class PingPangPlayer extends Player implements SpeakEnglish {

    public PingPangPlayer() {
    }

    public PingPangPlayer(String name, int age) {
        super(name, age);
    }

    @Override
    public void study() {
        System.out.println("乒乓球运动员学习如何发球和接球");
    }

    @Override
    public void eat() {
        System.out.println("乒乓球运动员吃大白菜，喝小米粥");
    }

    @Override
    public void speak() {
        System.out.println("乒乓球运动员说英语");
    }
}

```

- 篮球运动员

```

public class BasketballPlayer extends Player {

    public BasketballPlayer() {
    }

    public BasketballPlayer(String name, int age) {
        super(name, age);
    }

    @Override
    public void study() {
        System.out.println("篮球运动员学习如何运球和投篮");
    }

    @Override
    public void eat() {
        System.out.println("篮球运动员吃牛肉，喝牛奶");
    }
}

```