第九章：多态

1. 没有多态存在的问题

代码块

```
package com.powernode.polymorphic21;

//动物
class Animal{
    public void sleep(){
        System.out.println("Animal.sleep");
    }
    public void eat(){
        System.out.println("Animal.eat");
    }
}
//猫
class Cat extends Animal{
    @Override
    public void eat() {
        System.out.println("Cat.eat鱼");
    }

    @Override
    public void sleep() {
        System.out.println("Cat.sleep");
    }
}
//狗
class Dog extends Animal{
    @Override
    public void eat() {
        System.out.println("Dog.eat骨头");
    }

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


```

```
38  public class Test {
39      public static void main(String[] args) {
40          Cat cat = new Cat();
41          cat.eat();
42          cat.sleep();
43
44          Dog dog = new Dog();
45          dog.eat();
46          dog.sleep();
47          /**
48           * 以上代码可以进行优化，eat和sleep存在代码冗余
49           */
50      }
51  }
```

2. 使用多态来解决

代码块

```
1   package com.powernode.polymorphic22;
2
3   //动物
4   class Animal{
5       public void sleep(){
6           System.out.println("Animal.sleep");
7       }
8       public void eat(){
9           System.out.println("Animal.eat");
10      }
11  }
12  //猫
13  class Cat extends Animal{
14      @Override
15      public void eat() {
16          System.out.println("Cat.eat鱼");
17      }
18
19      @Override
20      public void sleep() {
21          System.out.println("Cat.sleep");
22      }
23  }
24  //狗
25  class Dog extends Animal{
26      @Override
27      public void eat() {
```

```
28            System.out.println("Dog.eat骨头");
29        }
30
31        @Override
32        public void sleep() {
33            System.out.println("Dog.sleep");
34        }
35    }
36
37
38    public class Test {
39        public static void main(String[] args) {
40            /* Cat cat = new Cat();
41             cat.eat();
42             cat.sleep();*/
43             //父类 父对象 = new 子类([实参列表])
44             Animal ac = new Cat();
45             /*ac.eat();
46             ac.sleep();*/
47             method(ac);
48
49             /*Dog dog = new Dog();
50             dog.eat();
51             dog.sleep();*/
52             Animal ad = new Dog();
53            /* ad.eat();
54             ad.sleep();*/
55             method(ad);
56        }
57        public static void method(Animal animal){
58            animal.eat();
59            animal.sleep();
60        }
61    }
```

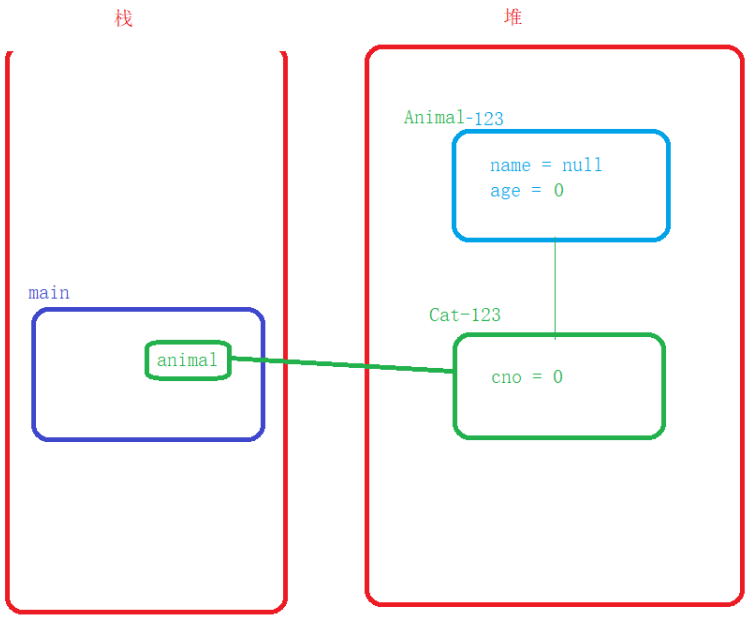
3. 多态的概述

1. 多态：父类引用指向子类对象
2. 多态的作用：
   a. 提高程序的扩展性
   b. 易于维护
   c. 符合高内聚，低耦合的开发原则

```java
package com.powernode.polymorphic22;

//动物
class Animal{
    public void sleep(){
        System.out.println("Animal.sleep");
    }
    public void eat(){
        System.out.println("Animal.eat");
    }
}
//猫
class Cat extends Animal{
    @Override
    public void eat() {
        System.out.println("Cat.eat鱼");
    }

    @Override
    public void sleep() {
        System.out.println("Cat.sleep");
    }
}
//狗
class Dog extends Animal{
    @Override
    public void eat() {
        System.out.println("Dog.eat骨头");
    }

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


public class Test {
    public static void main(String[] args) {
       /* Cat cat = new Cat();
        cat.eat();
        cat.sleep();*/
        //父类 父对象 = new 子类([实参列表])
        Animal ac = new Cat();
        /*ac.eat();
        ac.sleep();*/
        method(ac);
```

```
48
49          /*Dog dog = new Dog();
50          dog.eat();
51          dog.sleep();*/
52          Animal ad = new Dog();
53         /* ad.eat();
54          ad.sleep();*/
55          method(ad);
56      }
57      public static void method(Animal animal){
58          animal.eat();
59          animal.sleep();
60      }
61  }
```

```
package com.powernode.polymorphic23;

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

    public void sleep(){
        System.out.println("Animal.sleep");
    }
}
class Cat extends Animal{
    private int cno;

    @Override
    public void sleep() {
        System.out.println("Cat.sleep");
    }
}
class Test{
    public static void main(String[] args) {
        Animal animal = new Cat();
        animal.sleep();
    }
}
```



作业

1.继承练习

1.编写Animal类，包含名字(name)、年龄(age)等属性，getDetails方法用于返回Animal的详细信息

2.编写Fish(鱼)子类，继承Animal类，添加特有属性(scale-鳞片)和方法(swim)

3.编写Cat子类，继承Animal类，添加特有属性(hairColor-毛的颜色)和方法(catchMouse-抓老鼠)

4.编写Test类，在main方法中创建Fish和Cat对象，分别访问对象中特有的属性、方法，以及从Animal类继承的属性和方法并打印输出

2.方法覆盖

在Animal类中添加eat方法。

在Cat类中，覆盖eat 方法和getDetails方法，方法返回Cat吃什么以及Cat的详细信息。

在Test类中调用调用eat方法和getDetails方法，确认输出结果。

3.super调用父类的方法

在Cat类中，改写Cat覆盖的getDetails方法，使用super调用

在Test类中调用getDetails方法，确认输出结果。

4.super调用父类的有参构造器

在Cat类和Animal类中添加有参数的构造器，使用参数初始化各属性值。

在Cat类的构造器中调用父类有参数构造器。

创建Cat实例，调用getDetails方法获取输出结果，确认属性值。

5.多态调用

1.编写一个Person类，声明一个eat()方法。

2.编写一个Student类，继承Person类，添加swim（）方法

3.Student类重写父类的eat()方法。

4.编写Test类

1.创建Person 的对象，调用eat方法

2.创建Student 的对象，调用eat方法和 swim（）

3.使用多态创建对象,调用eat方法

4. 向下转型

1. 在多态情况下，如果拿到的是父类对象，需要调用子类特有的属性和方法，就需要向下转型
2. 父类对象 转换为 子对象

代码块

```
1   package com.powernode.polymorphic05;
2
3   class Animal{
4       public void eat(){
5           System.out.println("Aniaml.eat");
6       }
7       public void sleep(){
8           System.out.println("Animal.sleep");
9       }
10  }
11  class Cat extends Animal{
12      @Override
13      public void eat() {
14          System.out.println("Cat.eat鱼");
15      }
16
17      @Override
18      public void sleep() {
```

```
        System.out.println("Cat.sleep");
    }

    public void catchMouse(){
        System.out.println("Cat.catchMouse");
    }
}
public class Test {
    public static void main(String[] args) {
        Animal ac = new Cat();
        ac.eat();
        ac.sleep();
        //ac对象只能调用父类存在的方法，不能调用子类特有的方法
        //ac.catchMouse();
        //如果需要调用子类特有的方法，就需要向下转型
        double d = 1.2;
        int i = (int)d;
        //向下转型：其实就是强制转换
        Cat cat = (Cat) ac;
        cat.catchMouse();

    }
}
```

5. 向下转型带来的安全隐患

代码块

```
package com.powernode.polymorphic06;

class Animal{
    public void eat(){
        System.out.println("Aniaml.eat");
    }
    public void sleep(){
        System.out.println("Animal.sleep");
    }
}
class Cat extends Animal{
    @Override
    public void eat() {
        System.out.println("Cat.eat鱼");
    }

    @Override
    public void sleep() {
```

```
19          System.out.println("Cat.sleep");
20      }
21
22      public void catchMouse(){
23          System.out.println("Cat.catchMouse");
24      }
25  }
26  class Dog extends Animal{
27      @Override
28      public void eat() {
29          System.out.println("Dog.eat");
30      }
31
32      @Override
33      public void sleep() {
34          System.out.println("Dog.sleep");
35      }
36
37  }
38  public class Test {
39      public static void main(String[] args) {
40          Animal ac = new Cat();
41          method(ac);
42
43          Animal ad = new Dog();
44          method(ad);
45      }
46      public static void method(Animal animal){
47          animal.eat();
48          animal.sleep();
49          /**
50           * 1.调用Cat特有的方法catchMouse
51           * 2.就需要向下转型
52           * 3.错误: ClassCastException : 类转换异常
53           *    Dog cannot be cast to class Cat
54           */
55          Cat cat = (Cat) animal;
56          cat.catchMouse();
57      }
58  }
```

6. instanceof解决安全隐患

代码块

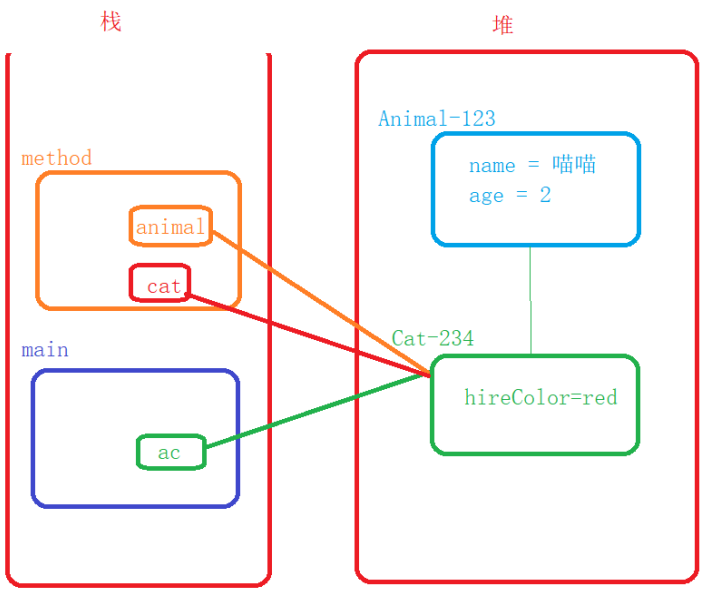
```
1   package com.powernode.polymorphic07;
```

```java

class Animal{
    public void eat(){
        System.out.println("Aniaml.eat");
    }
    public void sleep(){
        System.out.println("Animal.sleep");
    }
}
class Cat extends Animal{
    @Override
    public void eat() {
        System.out.println("Cat.eat鱼");
    }

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

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

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

}
public class Test {
    public static void main(String[] args) {
        Animal ac = new Cat();
        method(ac);

        Animal ad = new Dog();
        method(ad);
    }
    public static void method(Animal animal){
        animal.eat();
        animal.sleep();
```

```
49          /**
50           * 1.调用Cat特有的方法catchMouse
51           * 2.就需要向下转型
52           * 3.错误: ClassCastException : 类转换异常
53           *    Dog cannot be cast to class Cat
54           * 4.animal 有可能指向Cat对象，也有可能指向Dog对象
55           * 5.所以所以我们在强制转换之前要添加判断
56           */
57          if (animal instanceof Cat) {//判断animal是否指向Cat对象，如果指向返回true，否则返回false
58              Cat cat = (Cat) animal;
59              cat.catchMouse();
60          }else{
61              System.out.println("没有指向Cat对象");
62          }
63
64      }
65  }
```

7. 多态内存分析

```
class Animal{
   private String name;
   private int age;
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
class Cat extends Animal{
    private String hireColor;

    public Cat(String name, int age, String hireColor) {
        super(name, age);
        this.hireColor = hireColor;
    }
}
public class Test {
    public static void main(String[] args) {
        Animal ac = new Cat("喵喵",2,"red");
        method(ac);
    }
    public static void method(Animal animal){
        if (animal instanceof Cat) {
            Cat cat = (Cat) animal;
            System.out.println(cat);
        }
```



代码块

```
1  package com.powernode.polymorphic08;
2
3  class Animal{
4      private String name;
```

```
    private int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
class Cat extends Animal{
    private String hireColor;

    public Cat(String name, int age, String hireColor) {
        super(name, age);
        this.hireColor = hireColor;
    }
}
public class Test {
    public static void main(String[] args) {
        Animal ac = new Cat("喵喵",2,"red");
        method(ac);
    }
    public static void method(Animal animal){
        if (animal instanceof Cat) {//判断animal是否指向Cat对象，如果指向返回true，否则返回false
            Cat cat = (Cat) animal;
            System.out.println(cat);
        }
    }
}
```

8. 多态中的属性和方法

代码块

```
package com.powernode.polymorphic09;

public class Animal {
    public String name = "Animal.name";
    public int age = 2;

    public void eat(){
        System.out.println("Animal.eat");
    }
}
class Cat extends Animal{
    public String name = "Cat.name";

```

```
14      @Override
15      public void eat() {
16          System.out.println("Cat.eat");
17      }
18  }
19  class Test{
20      public static void main(String[] args) {
21          Animal animal = new Cat();
22          animal.eat();//子类的
23          System.out.println(animal.name);//父类的
24
25          /**
26           * 1.Java中的多态，指的是方法多态
27           * 2.属性和方法的调用规则：
28           *     1.属性:编译看父类，运行看父类
29           *     2.方法：编译看父类，运行看子类
30           */
31
32      }
33  }
```

9. 随堂练习

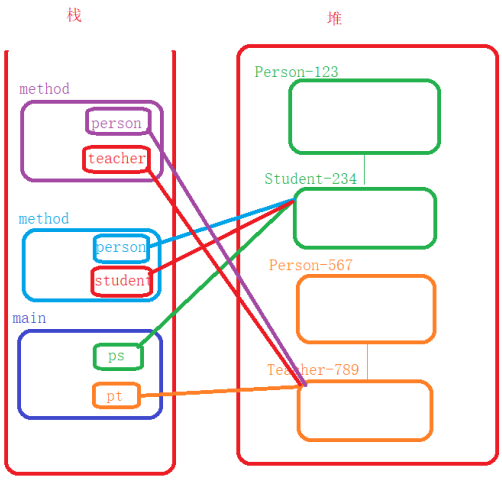
1. 编写Person类，添加eat方法，sleep方法
2. 编写Student类，重写eat的方法和sleep方法，添加特有的方法doHomeWork
3. 编写Teacher类，重写eat的方法和sleep方法，添加特有的方法lecture
4. 编写Test类
    a. 使用多态创建Student对象
    b. 使用多态创建Teacher对象
    c. 编写一个method方法参数是Person对象
    d. method方法中
        i. 调用eat和sleep的方法
        ii. 判断person的指向
        iii. 如果指向Student对象，强转后，调用doHomeWork
        iv. 如果指向Teacher对象，强转后，调用lecture

代码块

```java
package com.powernode.polymorphic10;

import javax.swing.*;

class Person{
    public void eat(){
        System.out.println("Person.eat");
    }
    public void sleep(){
        System.out.println("Person.sleep");
    }
}
class Student extends Person{
    @Override
    public void eat() {
        System.out.println("Student.eat");
    }

    @Override
    public void sleep() {
        System.out.println("Student.sleep");
    }
    public void doHomeWork(){
        System.out.println("Student.doHomeWork");
    }
}
class Teacher extends Person{
    @Override
    public void eat() {
        System.out.println("Teacher.eat");
    }

    @Override
    public void sleep() {
        System.out.println("Teacher.sleep");
    }
    public void lecture(){
        System.out.println("Teacher.lecture");
    }
}
public class Test {
    public static void main(String[] args) {
        Person ps = new Student();
        method(ps);
        Person pt = new Teacher();
        method(pt);
    }
```

```
48
49          public static void method(Person person) {
50              person.eat();
51              person.sleep();
52              if (person instanceof Student) {
53                  Student student = (Student) person;
54                  student.doHomeWork();
55              }else if(person instanceof Teacher){
56                  Teacher teacher = (Teacher) person;
57                  teacher.lecture();
58              }
59
60          }
61      }
```

```
public class Test {
    public static void main(String[] args) {
        Person ps = new Student();
        method(ps);
        Person pt = new Teacher();
        method(pt);
    }

    public static void method(Person person) {
        person.eat();
        person.sleep();
        if (person instanceof Student) {
            Student student = (Student) person;
            student.doHomeWork();
        }else if(person instanceof Teacher){
            Teacher teacher = (Teacher) person;
            teacher.lecture();
        }

    }
}
```




```
class Person{
    public void eat(){
        System.out.println("Person.eat");
    }
    public void sleep(){
        System.out.println("Person.sleep");
    }
}
class Student extends Person{
    //......
    public void doHomeWork(){
        System.out.println("Student.doHomeWork");
    }
}
class Teacher extends Person{
  //......
    public void lecture(){
        System.out.println("Teacher.lecture");
    }
}
```

10.