Java面向对象编程4

- What?Why?How?

为什么使用多态5-1

- **宠物饿了，需要主人给宠物喂食**
  - 不同宠物吃的东西不一样

  - 不同宠物恢复后体力值不一样

为什么使用多态5-2

- **狗狗类**
  - 增加狗狗吃东西的方法
- **企鹅类**
  - 增加企鹅吃东西的方法
- **创建主人类**
  - 编写给狗狗喂东西的方法
  - 编写给企鹅喂东西的方法
- **编写测试方法**
  - 调用主人类给狗狗喂的方法
  - 调用主人类给企鹅喂的方法

为什么使用多态5-3

- 如果再领养XXX宠物，就需要给XXX喂食，怎么办？
  - 添加XXX类，继承Pet类，实现吃食方法
  - 修改Master类，添加给XXX喂食的方法

为什么使用多态5-4

主人类

```
public class Master {
    public void feed( Dog dog ) {
        dog.eat();
    }
    public void feed( Penguin pgn ) {
        pgn.eat();
    }
    public void feed( XXX xxx ) {
        xxx.eat();
    }
    … …
}
```

测试方法

```
… …
Master master = new Master();
master.feed(dog);
master.feed(penguin);
master.feed(xxx);
…
```

参数都是Pet类的子类

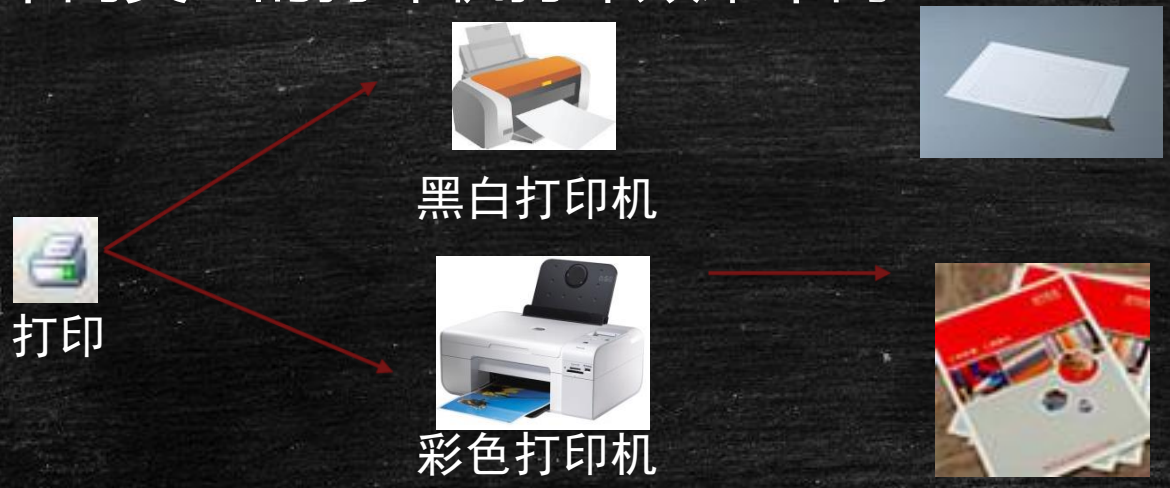
可否使用一个feed(Pet pet)
实现对所有宠物的喂食？

频繁修改代码，代码可扩展性、可维护性差，
如何优化？

使用多态优化设计

什么是多态

- 生活中的多态
  - 不同类型的打印机打印效果不同



同一种事物，由于条件不同，产生的结果也不同

父类引用，子类对象

- 程序中的多态

多态：同一个引用类型，使用不同的实例而执行不同操作

如何实现多态

- **使用多态实现思路**
  - 编写父类
  - 编写子类，子类重写父类方法
  - 运行时，使用父类的类型，子类的对象
    - 向上转型

实现多态的两个要素

```
Pet pet = new Dog();
```

自动类型转换

- **实现多态的两种形式**
  - 使用父类作为方法形参实现多态
  - 使用父类作为方法返回值实现多态

使用父类作为方法形参实现多态

- 使用多态优化主人给宠物喂食

主人类

```
public class Master {
    public void feed( Pet pet ) {
        pet.eat();
    }
}
```

使用父类作为方法形参

测试方法

```
… …
Pet pet = new Dog();
Master master = new Master();
master.feed( pet );
… …
```

同一种操作方式，不同的操作对象

上机练习——使用多态实现主人给宠物喂食

- 需求说明
  - 使用多态实现喂养宠物功能
  - 增加宠物猫并喂食，其健康值增加4

使用父类作为方法返回值实现多态

- **使用多态实现领养宠物**
  - 使用父类作为方法返回值
  - 实现思路
    - 在Master类添加领养方法getPet(String typeId)
    - 创建测试类，根据主人选择宠物类型编号来领养宠物

主人类

```
public class Master {
    public Pet getPet(String typeId ){
            ... ...
    }
}
```

使用父类作为返回值类型

父类到子类的转换4-1

- **实现主人与宠物玩耍功能**
  - 和狗狗玩接飞盘游戏，狗狗的健康值减少10，与主人亲密度增加5。
  - 和企鹅玩游泳游戏，企鹅的健康值减少10，与主人亲密度增加5。
- 给Dog添加接飞盘方法catchingFlyDisc( )
- 给Penguin添加游泳方法swimming( )
- 给主人添加play(Pet pet)方法

父类到子类的转换4-2

- 代码实现

狗狗类

```
public class Dog extends Pet {
    public void catchingFlyDisc() {
        … …
    }
}
```

主人类

```
public class Master {
    public void play(Pet pet){
        pet.catchingFlyDisc();
    }
}
```

报错，父类引用不能调用子类特有方法

企鹅类

```
public class Penguin  extends Pet {
    public void swimming () {
        … …
    }
}
```

测试类

```
… …
Pet pet = new Dog();
Master master = new Master();
master.pet(pet);
… …
```

父类到子类的转换4-3

- 父类到子类的转换
  - 向下转型（强制类型转换）

```
Pet pet = new Dog("欧欧", "雪娜瑞");
Dog dog = (Dog) pet;
Penguin png = (Penguin) pet;
```

报错，必须转换为父类指向的真实子类类型

- instanceof运算符

对象 instanceof 类或接口

instanceof通常和强制类型转换结合使用

父类到子类的转换4-4

- 优化主人与宠物玩耍

主人类

```
public class Master {
    public void play(Pet pet){
        if (pet instanceof Dog) {    //如果传入的是狗狗
            Dog dog = (Dog) pet;
            dog.catchingFlyDisc();
        }else if (pet instanceof Penguin) {   //如果传入的是企鹅
            Penguin pgn = (Penguin) pet;
            pgn.swimming();
        }
    }
}
```

上机练习2—使用多态实现主人领养宠物并与宠物玩耍

- 需求说明：
  - 主人根据宠物编号领养宠物
  - 主人和狗狗玩接飞盘游戏，狗狗健康值减少10，与主人亲密度增加5
  - 主人和企鹅玩游泳游戏，企鹅健康值减少10，与主人亲密度增加5

上机练习3——计算一次租赁多辆汽车的总租金2-1

- 训练要点：
  - 多态的使用
  - 使用父类类型作为方法参数
- 需求说明：
  - 在前面汽车租赁系统的基础上，实现计算多种车辆总租金的功能
  - 现在有客户租用
    - 2辆宝马
    - 1辆别克商务舱
    - 1辆金龙（34）座
    - 租5天共多少租金？

上机练习3——计算一次租赁多辆汽车的总租金2-2

- 实现思路：

1、创建车的对象，放在数组中

```
MotoVehile[] motos = new MotoVehile[4];
motos[0] = new Car("宝马550i","京NY28588");
motos[1] = new Car("宝马550i","京NNN328");
motos[2] = new Car("别克林荫大道","京NY28588");
motos[3] = new Bus("金龙",34);
```

2、循环调用calcRent()方法，计算总租金

```
public int calcTotalRent(MotoVehile[] motos,int days){
    int totalRent = 0;
    for(int i=0;i<motos.length;++i){
        totalRent += motos[i].calRent(days);
    }
    return totalRent;
}
```

上机练习4——购置新车2-1

- 训练要点：
  - 使用父类作为方法形参实现多态
  - 使用多态增强系统的扩展性和可维护性
- 需求说明：
  - 新购置了卡车，根据吨位，租金每吨每天50
  - 对系统进行扩展，计算汽车租赁的总租金

上机练习4——购置新车2-2

- 实现思路
  1.创建卡车类，实现calcRent ()方法
  2.修改统计租金代码

提前做完的同学可以尝试改进系统：
1. 循环从控制台选择汽车种类
2. 从控制台输入天数
3. 累加计算总租金

小结

类型转换

向上转型——子类转换为父类，自动进行类型转换

向下转型——父类转换为子类，结合instanceof运算符进行强制类型转换

实现多态的两种方式

使用父类作为方法形参实现多态

使用父类作为方法返回值实现多态

使用多态的好处?

多态可以减少类中代码量，可以提高代码的可扩展性和可维护性

引用变量的两种类型：

编译时类型（模糊一点，一般是一个父类）

由声明时的类型决定。

运行时类型（运行时，具体是哪个子类就是哪个子类）

由实际对应的对象类型决定。

多态的存在要有3个必要条件：

要有继承，要有方法重写，父类引用指向子类对象

多态 polymorphism

多态性是OOP中的一个重要特性，主要是用来实现动态联编的，换句话说，就是程序的最终状态只有在执行过程中才被决定而非在编译期间就决定了。这对于大型系统来说能提高系统的灵活性和扩展性。

java中如何实现多态?使用多态的好处?

引用变量的两种类型：

编译时类型（模糊一点，一般是一个父类）

由声明时的类型决定。

运行时类型（运行时，具体是哪个子类就是哪个子类）

由实际对应的对象类型决定。

多态的存在要有3个必要条件：

要有继承，要有方法重写，父类引用指向子类对象

多态示例代码

```java
package object;
public class TestPolym {

public static void main(String[] args) {
            Animal animal = new Dog();  //向上可以自动转型

            System.out.println(animal.age);    //属性调用时，仍然是基类的属性
//          animal.shout();

            animalCry(new Dog());
            //传的具体是哪一个类就调用哪一个类的方法。大大提高了程序的可扩展性。
            //如果没有多态，我们这里需要写很多重载的方法。如果增加一种动物，就需要

             //有了多态，只需要增加这个类继承Animal基类就可以了。
            animalCry(new Cat());

            Dog dog = (Dog) animal;  //编写程序时，如果想调用运行时类型的方法
            dog.gnawBone();
                                    System.out.println(dog instanceo
            System.out.println(animal instanceof Cat);
            System.out.println(animal instanceof Dog);

            }


            static void animalCry(Animal a){
                        a.shout();
            }

}
```

```java
class Animal {
            int age=10;
            public void shout(){
                System.out.println("叫了一声！");
            }
}
class Dog extends Animal {
            int age=28;
            public void shout() {
                System.out.println("旺旺旺！");
            }

            public void gnawBone(){
                System.out.println("我在啃骨头");
            }

}
class Cat extends Animal {
            int age=18;
            public void shout() {
            System.out.println("喵喵喵喵！");
            }
}
```

引用数据类型的类型转换

- 子类转换为父类：自动转换
  - 上转型对象不能操作子类新增的成员变量和方法。
  - 上转型对象可以操作子类继承或重写的成员变量和方法
  - 如果子类重写了父类的某个方法，上转型对象调用该方法时，是调用的重写方法。
- 父类转换为子类：强制转换
  - （绝不是做手术，而是父类的真面目就是一个子类，否则会出现类型转换错误）