【案例4-2】饲养员喂养动物

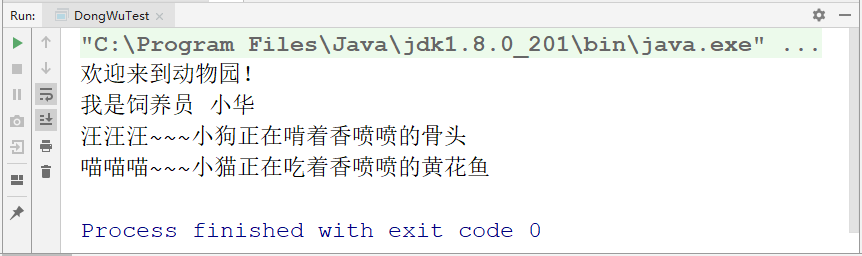
**【案例目标】**

1. **案例描述**

饲养员在给动物喂食时，给不同的动物喂不同的食物，而且在每次喂食时，动物都会发出欢快的叫声。例如，给小狗喂骨头，小狗会汪汪叫；给小猫喂食，小猫会喵喵叫。

本案例要求编写一个程序模拟饲养员喂食动物的过程，案例要求如下：

1. 饲养员给小狗喂骨头，小狗汪汪叫。
2. 饲养员给小猫喂小鱼，小猫喵喵叫。
3. **运行结果**



**【案例目标】**

* 学会分析“饲养员喂养动物”程序设计的实现思路。
* 根据思路独立完成“饲养员喂养动物”程序设计的源代码编写、编译及运行。
* 学会灵活运用接口解决多继承问题。

**【案例分析】**

在这个动物园里，涉及的对象有饲养员，各种不同动物以及各种不同的食物。这样很容易抽象出3个类Feeder、Animal和Food。假设只考虑猫类和狗类动物，则由Animal类派生出Cat类、Dog类、同样由Food类可以进一步派生出其子类Bone、Fish。因为他们之间存在这明显的is-a关系。

同样的，鱼是一种食物，但实际上，鱼也是一种动物，Cat类和Dog类继承了Animal的很多属性和方法，如果将Animal定义为接口，Animal中是不能定义成员变量和成员方法的，Food类中虽然也有变量但是数量比Animal少，所以我们考虑将Food定义为接口，此时可以说“鱼是一种动物，同时也是一种食物”。

**【案例实现】**

Animal.java

1. public abstract class Animal {
2. private String name;
3. public Animal(String name) {
4. this.name = name;
5. }
6. public abstract void shout();
7. public abstract void eat(Food food);
8. public String getName() {
9. return name;
10. }
11. public void setName(String name) {
12. this.name = name;
13. }
14. }

上述代码中，定义了抽象类Animal，第4行代码声明了动物名称name，第7~13行代码提供了name的getter和setter方法。同时，编写了动物发声的抽象方法shout()和动物吃饭的抽象方法eat()。

Dog.java

1. public class Dog extends Animal {
2. public Dog(String name) {
3. super(name);
4. }
5. @Override
6. public void shout() {
7. System.out.print ("汪汪汪~~~");
8. }
9. @Override
10. public void eat(Food food) {
11. System.out.println(getName() + "正在啃着香喷喷的" + food.getName());
12. }
13. }

上述代码中，定义了Dog类，并继承了Animal类，第 2~4行代码是在Dog类中定义类构造方法并在构造方法中调用了父类的构造方法。第6~12行代码重写了父类Animal动物发声的抽象方法shout()和动物吃饭的抽象方法eat()。

Cat.java

1. public class Cat extends Animal{
2. public Cat(String name) {
3. super(name);
4. }
5. public void shout() {
6. System.out.print("喵喵喵~~~");
7. }
8. public void eat(Food food) {
9. System.out.println(getName() + "正在吃着香喷喷的"+food.getName());
10. }
11. }

上述代码中，定义了Cat类，并继承了Animal类，第2~4行代码是在Cat类中定义类构造方法并在构造方法中调用了父类的构造方法。第5~10行代码重写了父类Animal动物发声的抽象方法shout()和动物吃饭的抽象方法eat()。

Food.java

1. public interface Food {
2. public abstract String getName();
3. }

上述代码中，定义了一个Food接口，在接口中编写了一个抽象方法getName()。

Bone.java

1. public class Bone implements Food{
2. @Override
3. public String getName() {
4. return "骨头";
5. }
6. }

上述代码中，定义了一个Bone类并实现了Food接口，在Bone类中实现了Food接口的getName()方法。

Fish.java

1. public class Fish extends Animal implements Food{
2. public Fish(String name) {
3. super(name);
4. }
5. @Override
6. public void shout() {
7. }
8. @Override
9. public void eat(Food food) {
10. }
11. }

上述代码中，定义了一个Fish类，Fish类继承了Animal类并实现了Food接口，第2~4行代码是在Fish类中定义了构造方法并在构造方法中调用了父类Food的构造方法；第5~10行代码是继承了Animal类的shout()方法和eat()方法。

Feeder.java

1. public class Feeder {
2. private String name;
3. public Feeder(String name) {
4. this.name = name;
5. }
6. public void speak() {
7. System.out.println("欢迎来到动物园！");
8. System.out.println("我是饲养员 "+getName());
9. }
10. public void feed(Animal a, Food food) {
11. a.eat(food);
12. }
13. public String getName() {
14. return name;
15. }
16. public void setName(String name) {
17. this.name = name;
18. }
19. }

上述代码中，定义了一个Feader类，Feader类中声明了饲养员名称name并提供了name属性的getter和setter方法；定义了构造方法；同时，第6~12行代码编写了饲养员说话的speak()方法和饲养动物的feed()方法。

DongWuTest.java

1. public class DongWuTest {
2. public static void main(String[] args) {
3. Feeder feeder = new Feeder("小华");
4. feeder.speak();
5. Dog dog = new Dog("小狗");
6. dog.shout();
7. Food food = new Bone();
8. feeder.feed(dog, food);
9. Cat cat = new Cat("小猫");
10. cat.shout();
11. food = new Fish("黄花鱼");
12. feeder.feed(cat, food);
13. }
14. }

上述代码中，定义了一个测试类DongWuTest，第3~4行代码是在DongWuTest类中创建了一个feeder对象，并调用了饲养员说话的方法speak()；然后，第7~12行代码是创建了food对象、dog对象和cat对象，并在创建dog和cat对象时传入动物的名称，最后通过feeder对象调用feed()方法模拟饲养员喂养动物，分别通过dog和cat对象调用shout()方法模拟动物发声。