1. 定义

策略模式定义了算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。

2. 组成

· 抽象策略角色： 策略类，通常由一个接口或者抽象类实现。

* 具体策略角色：包装了相关的算法和行为。
* 环境角色：持有一个策略类的引用，最终给客户端调用。

3. 需求

现在要设计一个鸭子超类，实现鸭子会飞的功能等。

解决：

（1）直接设计一个超类Duck在其中添加一个fly()方法。

问题：并不是所有的鸭子都会飞（橡皮鸭子）

（2）设计一个飞行接口Flyable()，其它所有会飞的实现此接口，同时继承Duck类。

问题：无法代码复用，重复的代码会变多。

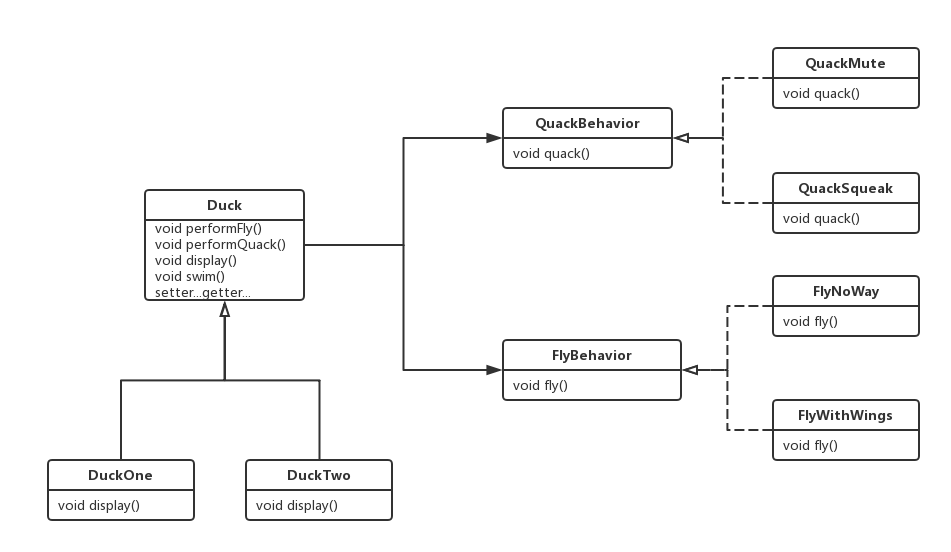
（3）在Duck中加入fly接口的实例变量，然后设计多个飞行类实现fly接口，在Duck的子类中设置飞行行为。

优点：面向接口编程，而非面向实现编程。

缺点：一但设置行为，就不可更改。

（4）在Duck中增加setter方法用于设置行为。

4. 画图



5. demo

Duck抽象类：

**public abstract class** Duck {  
 FlyBehavior **flyBehavior**;  
 QuackBehavior **quackBehavior**;  
 **public** Duck(){}  
 **public void** performFly(){  
 **flyBehavior**.fly();  
 }  
 **public void** performQuack(){  
 **quackBehavior**.quack();  
 }  
 **public void** swim(){  
 System.***out***.println(**"小鸭子都会游泳..."**);  
 }  
 **public void** display(){  
 System.***out***.println(**"我是鸭子"**);  
 }  
 **public** FlyBehavior getFlyBehavior() {  
 **return flyBehavior**;  
 }  
 **public void** setFlyBehavior(FlyBehavior flyBehavior) {  
 **this**.**flyBehavior** = flyBehavior;  
 }  
 **public** QuackBehavior getQuackBehavior() {  
 **return quackBehavior**;  
 }  
 **public void** setQuackBehavior(QuackBehavior quackBehavior) {  
 **this**.**quackBehavior** = quackBehavior;  
 }  
}

Duck的一个继承类：

**public class** DuckOne **extends** Duck {  
 **public** DuckOne(){  
 **flyBehavior** = **new** FlyWithWings();  
 **quackBehavior** = **new** QuackSqueak();  
 }  
 **public void** display(){  
 System.***out***.println(**"我是一只假鸭子..."**);  
 }  
}

Duck的一个行为接口：

**public interface** FlyBehavior {  
 **public void** fly();  
}

Duck的一个行为的实现：

**public class** FlyNoWay **implements** FlyBehavior {  
 @Override  
 **public void** fly() {  
 System.***out***.println(**"我是一个不会飞的鸭子..."**);  
 }  
}

main方法测试：

**public static void** main(String[] args){  
 *//定义duckOne* Duck duckOne = **new** DuckOne();  
 duckOne.setFlyBehavior(**new** FlyNoWay());  
 duckOne.setQuackBehavior(**new** QuackMute());  
 *//定义duckTwo* Duck duckTwo = **new** DuckTwo();  
 duckTwo.setFlyBehavior(**new** FlyWithWings());  
 duckTwo.setQuackBehavior(**new** QuackSqueak());  
 *//duckOne------* duckOne.display();  
 duckOne.performFly();  
 duckOne.performQuack();  
 *//duckTwo------* duckTwo.display();  
 duckTwo.performFly();  
 duckTwo.performQuack();  
}

运行结果：

