

# 设计模式之策略模式(Strategy Pattern)



六尺帐篷 (/u/f8e9b1c246f1)

2016.07.20 20:58\* 字数 1984 阅读 117 评论 0 喜欢 6

(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/5229761/notes/4884287)

策略模式，是我们接触到的第一个设计模式，也是较容易理解的一个模式。

我们可以给它下一个定义：

**定义了算法族，分别封装起来，让它们之间可以互相转换，此模式让算法的独立于使用算法的客户。**

维基百科上的定义是：**a software design pattern that enables an algorithm's behavior to be selected at runtime.**

维基百科上的强调了算法行为是在运行时决定的，这正是策略模式很关键的一点。

## 引子

假设我们现在要设计一个鸭子类Duck类，然后让不同的鸭子继承于它。我们把目光聚焦到鸭子的行为上。如果我们要给鸭子增加一个行为“fly”，第一个想法，在抽象类duck里添加一个fly方法就可，其余鸭子继承实现这个方法。

但是这就出现了一个问题，并不是所有鸭子都会飞，我们反而让一些本不具备这个fly行为的鸭子也具有该行为。那怎么办呢？

利用继承来提供鸭子的行为，会导致下面这些后果：

- 代码在多个子类中重复，如果两类不同鸭子需要同一种fly行为，我们就要在两个类里分别覆盖两次，这样万一维护起来是非常困难的
- 很难知道所有鸭子的全部行为
- 运行时的行为不容易改变
- 改变会一发动全身，造成其他鸭子不想要改变

## 设计原则1

软件开发中，我们常常需要遵守的设计原则是：

**把可能需要变化的地方独立出来，不要和那些不需要变化的代码混在一起**

这样代码变化引起的不经意后果变少，系统变得更有弹性

实际就是尽量让系统中某部分的改变不影响其他部分的变化。

## 提取鸭子的行为

根据设计原则，鸭子飞行的行为会发生变化，所以我们需要将fly行为单独提取出来。同理，我们提取出两个鸭子可能变化的行为fly和quack鸭叫。用两组类分别代表fly和quack行为。

## 设计原则2

那么我们如何那两组鸭子行为的类呢？这里引出第二个我们提出的设计原则：

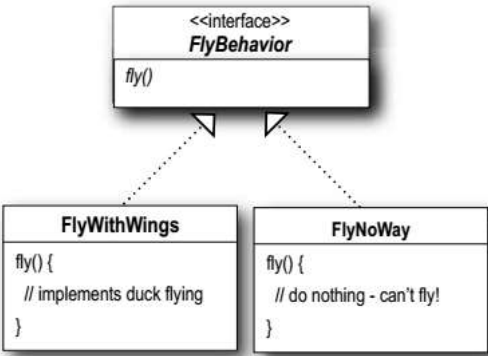
**面对接口编程，而不是面对实现编程**

这样就可以实现在运行时改变鸭子的行为。

我们不会直接指定特定的行为给鸭子。而是声明两个接口FlyBehavior和QuackBehavior。我们制造的其他一系列的类专门来实现FlyBehavior和QuackBehavior，这组就成为行为类，或者算法类。

用行为类来实现接口而不是利用duck类来实现。



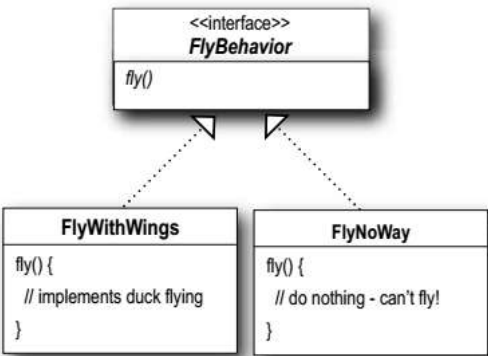


Paste\_Image.png

## 实现鸭子的行为

根据设计原则2，可以让飞行和鸭叫行为的动作被其他对象复用，因为这些行为已经与鸭子类无关了。

而且当我们新增一些行为的时候，不会影响到既有的行为类，也不会影响鸭子类。太棒了！



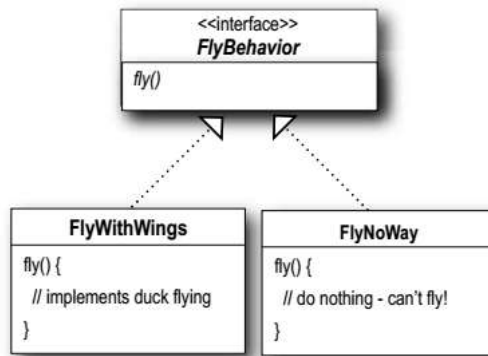
Paste\_Image.png

很多同学都觉得这里用类来代表行为是不是觉得很奇怪。在大家默认里，类应该是代表某种东西的，类应该拥有状态与行为。这里，我们需要纠正这个观点，一个行为也可以具有各种属性和函数。类不仅仅是用来代表东西事物的。

## 整合实现我们设计的鸭子类

首先，在duck类中加入两个实例变量，分别声明为两个接口的类型，每个鸭子对象都会动态的设置这些变量以便在运行时引用正确的行为类型





Paste\_Image.png

### duck类的实现

```

package strategyPattern;

//抽象的Duck类，所有鸭子都继承
public abstract class Duck {

    //为行为接口类型声明两个引用变量，所有鸭子类都继承它们
    FlyBehavior flyBehavior;
    QuackBehavior quackBehavior;

    public void setFlyBehavior(FlyBehavior flyBehavior) {
        this.flyBehavior = flyBehavior;
    }

    public void setQuackBehavior(QuackBehavior quackBehavior) {
        this.quackBehavior = quackBehavior;
    }

    public Duck() {

    }

    public abstract void display();

    public void performFly() {
        flyBehavior.fly(); //委托给fly行为类实现
    }

    public void performQuack() {
        quackBehavior.quack(); //委托给quack行为类实现
    }

    //swim是所有鸭子都共同拥有的方法，所以可以直接在在duck类中实现
    public void swim() {
        System.out.println("All ducks float, even decoys!");
    }
}
  
```

### FlyBehavior接口的实现：

```

package strategyPattern;

//所有飞行行为类必须实现的接口
public interface FlyBehavior {
    public void fly();
}
  
```

两个fly行为实现类，继承至FlyBehavior接口



```
package strategyPattern;

public class FlyWithWings implements FlyBehavior {

    @Override
    public void fly() {
        // TODO Auto-generated method stub
        System.out.println("I'm flying with wings!");
    }

}
```

```
package strategyPattern;

public class FlyNoway implements FlyBehavior {

    @Override
    public void fly() {
        // TODO Auto-generated method stub
        System.out.println("I can't fly!");
    }

}
```

Quack接口及其三个行为实现类:

```
package strategyPattern;

public class Quack implements QuackBehavior {

    @Override
    public void quack() {
        // TODO Auto-generated method stub
        System.out.println("Quack");
    }

}
```

```
package strategyPattern;

public class MuteQuack implements QuackBehavior {

    @Override
    public void quack() {
        // TODO Auto-generated method stub
        System.out.println("<<silence>>");
    }

}
```

```
package strategyPattern;

public class Squeak implements QuackBehavior {

    @Override
    public void quack() {
        // TODO Auto-generated method stub
        System.out.println("Squeak");
    }

}
```

```
package strategyPattern;

public interface QuackBehavior {
    public void quack();
}
```

一个MallardDuck类继承至Duck类:



```
package strategyPattern;

public class MallardDuck extends Duck {

    public MallardDuck() {
        quackBehavior = new Quack();
        flyBehavior = new FlyWithWings();
    }

    @Override
    public void display() {
        // TODO Auto-generated method stub
        System.out.println("I'm a real Mallard duck! ");
    }

}
```

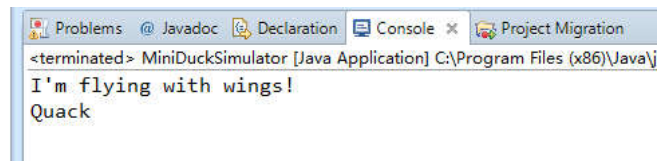
### 测试类

```
package strategyPattern;

public class MiniDuckSimulator {
    public static void main(String[] args) {
        Duck mallard = new MallardDuck();
        //这里调用mallardduck继承来的performFly方法，进而委托给该对象的quackBehavior对象处理
        //也就是最后是调用了继承来的quackBehavior引用对象的quack () 方法
        mallard.performFly();
        mallard.performQuack();

    }
}
```

### 运行结果：



Paste\_Image.png

在这里为了实现动态的改变鸭子的行为，我们可以新建一个flyrocketPowered行为类，然后动态的改变其行为：

```
package strategyPattern;

public class FlyRocketPowered implements FlyBehavior {

    @Override
    public void fly() {
        // TODO Auto-generated method stub
        System.out.println("I'm flying with a rocket ");
    }

}
```

```
package strategyPattern;

public class MiniDuckSimulator {
    public static void main(String[] args) {
        Duck mallard = new MallardDuck();
        //这里调用mallardduck继承来的performFly方法，进而委托给该对象的quackBehavior对象处理
        //也就是最后是调用了继承来的quackBehavior引用对象的quack () 方法
        mallard.performFly();
        mallard.performQuack();

        Duck model = new ModelDuck();
        model.performFly();
        model.setFlyBehavior(new FlyRocketPowered());
        model.performFly();

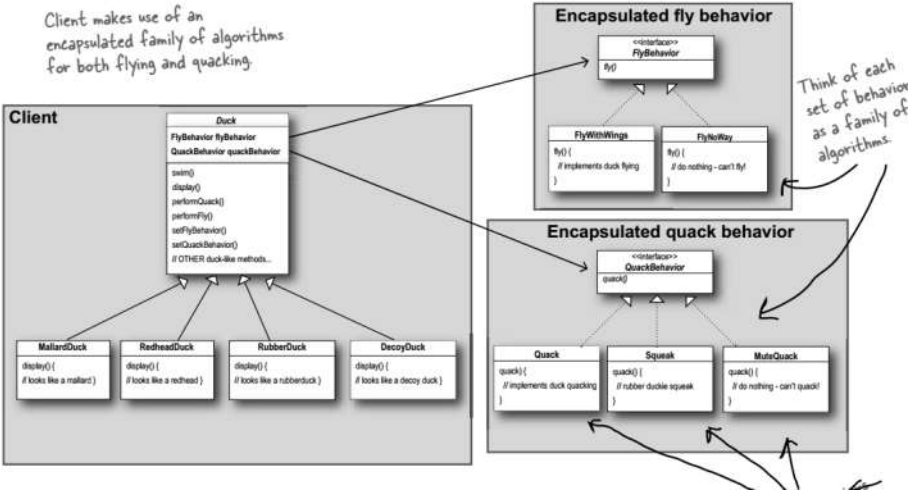
    }
}
```



运行结果：

```
Problems @ Javadoc Declaration Console x Project
<terminated> MiniDuckSimulator [Java Application] C:\Program File
I'm flying with wings!
Quack
I can't fly!
I'm flying with a rocket
```

Paste\_Image.png



Paste\_Image.png

每一个鸭子都有一个FlyBehavior和一个quackBehavior，好将飞行和鸭叫委托给他们代为处理。

当你将两个类结合起来使用时，如同本例，这就是组合composition。这种做法和继承不同的地方在于，鸭子的行为不是继承来的而是和适当行为对象那个组合来的。

设计原则3：

多用组合 少用继承

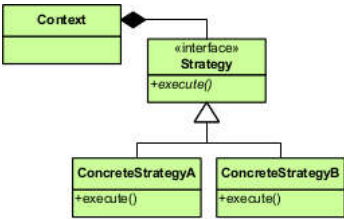
策略模式总结

三个设计原则：

- 封装变化，分开变化与不变
- 多用组合，少用继承
- 面向接口编程，而不是面对实现编程


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转换，此模式让算法的独立于使用算法的客户。



Paste\_Image.png

实现策略模式，我们需要对行为或算法实现各自的接口，具体的实现交给继承自这些接口的行为类，不需要在我们的主类鸭子中实现。主类鸭子声明两个接口的引用的实例变量，并设计set方法，这样就能在运行时动态的改变行为。实现独立和复用。



六尺帐篷 (/u/f8e9b1c246f1)

写了 245418 字，被 15240 人关注，获得了 1429 个喜欢

(/u/f8e9b1c246f1)

喜欢 6



更多分享

(http://cwb.assets.jianshu.io/notes/images/4884287

被以下专题收入，发现更多相似内容

投稿管理

+ 收入我的专题

-  程序员 (/c/NEt52a?utm\_source=desktop&utm\_medium=notes-included-collection)
-  Android知识 (/c/3fde3b545a35?utm\_source=desktop&utm\_medium=notes-included-collection)
-  Android... (/c/8404941e82d2?utm\_source=desktop&utm\_medium=notes-included-collection)
-  首页投稿 (/c/bDHhpK?utm\_source=desktop&utm\_medium=notes-included-collection)
-  iOS Dev... (/c/3233d1a249ca?utm\_source=desktop&utm\_medium=notes-included-collection)





