链滴

设计模式 |01 策略模式

作者：qq692310342
原文链接：https://ld246.com/article/1567425402907
来源网站：链滴

开头说几句

博主博客地址： https://www.jeffcc.top/

设计模式系列是我通过看Head First 的《设计模式》中学到的知识总结，这是本不错的设计模式入门籍，强烈推荐！

什么是策略模式

权威定义：策略模式定义了算法簇，使不同的行为分别封装起来，让他们可以相互替换，此模式让算的变化独立于使用算法的对象！

我的理解：将相同的行为共同封装进入超类，不同的行为单独封装，以实现功能！

设计原则

1. 找到应用中可能需要变化的部分，并且把他们独立起来，通过面向接口的编程方式进行独立划分，需要和那些不需要变化的代码混合在一起。

2. 针对接口（超类型）编程，而不是针对实现编程。

3. 多用组合，少用继承。

模式实例

背景

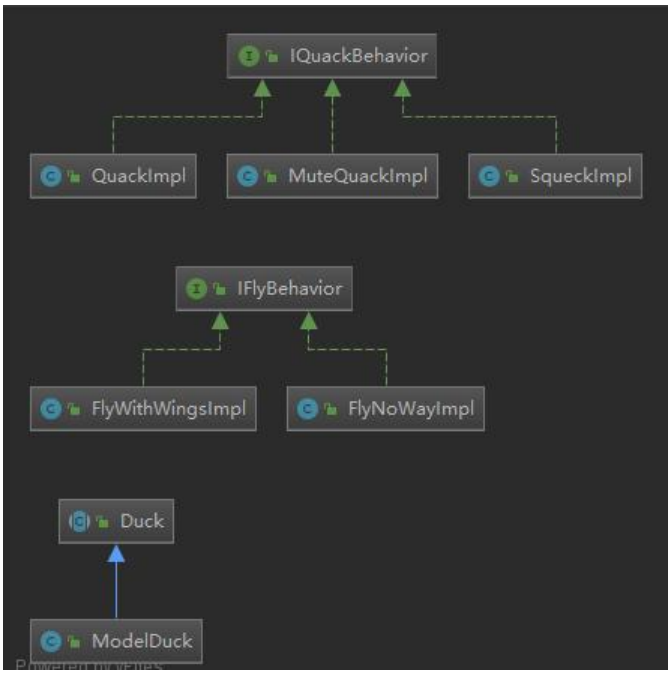
1. 需要定义一个鸭子超类，用于显示组合鸭子的共性的行为；

2. 需要我们定义出一些鸭子的所具有的一些特性：例如飞行和叫声的行为单独接口组合

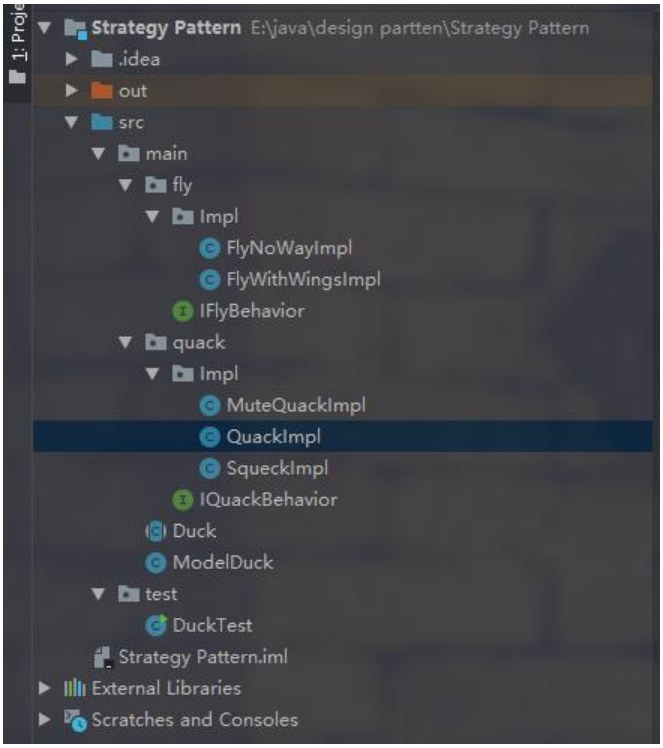
3. 实现鸭子的相关正确的行为，（不会出现橡皮鸭会飞等的情况）

代码实现

项目类图



项目结构



鸭子的超类

```
package main;
```

```
import main.fly.IFlyBehavior;
import main.quack.IQuackBehavior;

/**
 * 模拟策略模式
 * 行为对象代替继承
 */
public abstract class Duck {

//    定义一个飞行的行为
    private IFlyBehavior flyBehavior;
//    定义一个叫声的行为
    private IQuackBehavior quackBehavior;

    public Duck(IFlyBehavior flyBehavior, IQuackBehavior quackBehavior) {
//        构造函数声明
        this.flyBehavior = flyBehavior;
        this.quackBehavior = quackBehavior;
    }

//    鸭子的行为
    public abstract void displace();

//    组合一层鸭子的飞行行为，不使用继承
    public void performFly(){
        flyBehavior.fly();
    }
//    组合一层鸭子的叫声行为，不使用继承
    public void performQuack(){
        quackBehavior.quack();
    }

//    手动设置飞行行为
    public void setFlyBehavior(IFlyBehavior flyBehavior) {
        this.flyBehavior = flyBehavior;
    }
//    手动设置叫声行为
    public void setQuackBehavior(IQuackBehavior quackBehavior) {
        this.quackBehavior = quackBehavior;
    }
}
```

模型鸭类

```
package main;

import main.fly.Impl.FlyNoWayImpl;
import main.quack.Impl.MuteQuackImpl;

/**
 * 模型鸭
 * 继承鸭子行为
 */
public class ModelDuck extends Duck {
```

```
//    给模型鸭设定一个 不能飞 不会叫的行为
    public ModelDuck(){
        super(new FlyNoWayImpl(),new MuteQuackImpl());
    }

    @Override
    public void displace() {
        System.out.println("I am a model duck!");
    }
}
```

叫声行为接口

```
package main.quack;

/**
 * 叫声行为接口
 */
public interface IQuackBehavior {
    void quack();
}
```

飞行行为接口

```
package main.fly;

/**
 * 飞行行为接口
 */
public interface IFlyBehavior {
    void fly();
}
```

不能飞的实现类

```
package main.fly.Impl;

import main.fly.IFlyBehavior;

/**
 * 不能飞的实现类
 */
public class FlyNoWayImpl implements IFlyBehavior {
    @Override
    public void fly() {
        System.out.println("I can not fly!");
    }
}
```

可以飞的实现类

```
package main.fly.Impl;

import main.fly.IFlyBehavior;

/**
 * 可以飞的实现类
 */
public class FlyWithWingsImpl implements IFlyBehavior {

    @Override
    public void fly() {
        System.out.println("I can fly！ ");
    }
}
```

叫声行为接口

```
package main.quack;

/**
 * 叫声行为接口
 */
public interface IQuackBehavior {
    void quack();
}
```

不能叫的实现类

```
package main.quack.Impl;

import main.quack.IQuackBehavior;

/**
 * 不能叫的实现类
 */
public class MuteQuackImpl implements IQuackBehavior {
    @Override
    public void quack() {
        System.out.println("Silence");
    }
}
```

哇哇叫的实现类

```
package main.quack.Impl;

import main.quack.IQuackBehavior;

/**
```

```
 * 哇哇叫的实现类
 */
public class QuackImpl implements IQuackBehavior {
    @Override
    public void quack() {
        System.out.println("Qucak");
    }
}
```

呱呱叫的实现类

```
package main.quack.Impl;

import main.quack.IQuackBehavior;

/**
 * 呱呱叫
 */
public class SqueckImpl implements IQuackBehavior {
    @Override
    public void quack() {
        System.out.println("Squeak");
    }
}
```

测试类

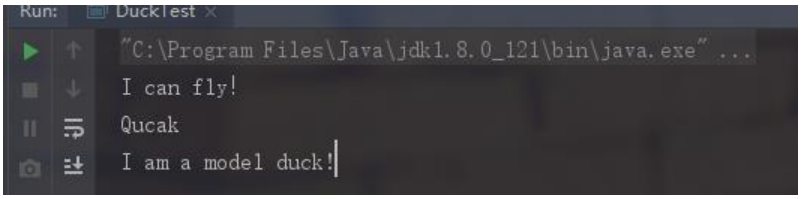
```
package test;

import main.Duck;
import main.ModelDuck;
import main.fly.Impl.FlyWithWingsImpl;
import main.quack.Impl.QuackImpl;

/**
 * 测试类
 */
public class DuckTest{

    public static void main(String[] args) {
//        创建一个模型鸭的实例
        Duck duck = new ModelDuck();
        duck.setFlyBehavior(new FlyWithWingsImpl());
        duck.setQuackBehavior(new QuackImpl());
        duck.performFly();
        duck.performQuack();
        duck.displace();
    }
}
```

输出结果：



END

2019年9月2日19:55:03