软件构造第6次实验

实验报告

**班级**：软件工程14-3

**学号**：201401061038

**姓名**：张正锟

2016年4月27日

目录

[一、 设计 3](#_Toc450145401)

[1. 鸭子类继承图 3](#_Toc450145402)

[2. 鸡、鸭子适配器模式继承图 4](#_Toc450145403)

[3. 工厂继承图 4](#_Toc450145404)

[4. 观察者模式继承图 5](#_Toc450145405)

[5. 计数器（装饰者）继承图 5](#_Toc450145406)

[6. 策略模式（飞行、游泳）类图 6](#_Toc450145407)

[二、 实现 6](#_Toc450145408)

[Quackable接口实现 6](#_Toc450145409)

[抽象鸭子实现 6](#_Toc450145410)

[鸭鸣器、绿头鸭、红头鸭、橡皮鸭实现 8](#_Toc450145411)

[鸭子群Flock实现 10](#_Toc450145412)

[鸡和适配器实现 11](#_Toc450145413)

[鸭子和适配器实现 12](#_Toc450145414)

[抽象工厂、普通工厂、计数器工厂实现 13](#_Toc450145415)

[飞行策略实现 16](#_Toc450145416)

[游泳策略实现 16](#_Toc450145417)

[观察者、被观察者实现 17](#_Toc450145418)

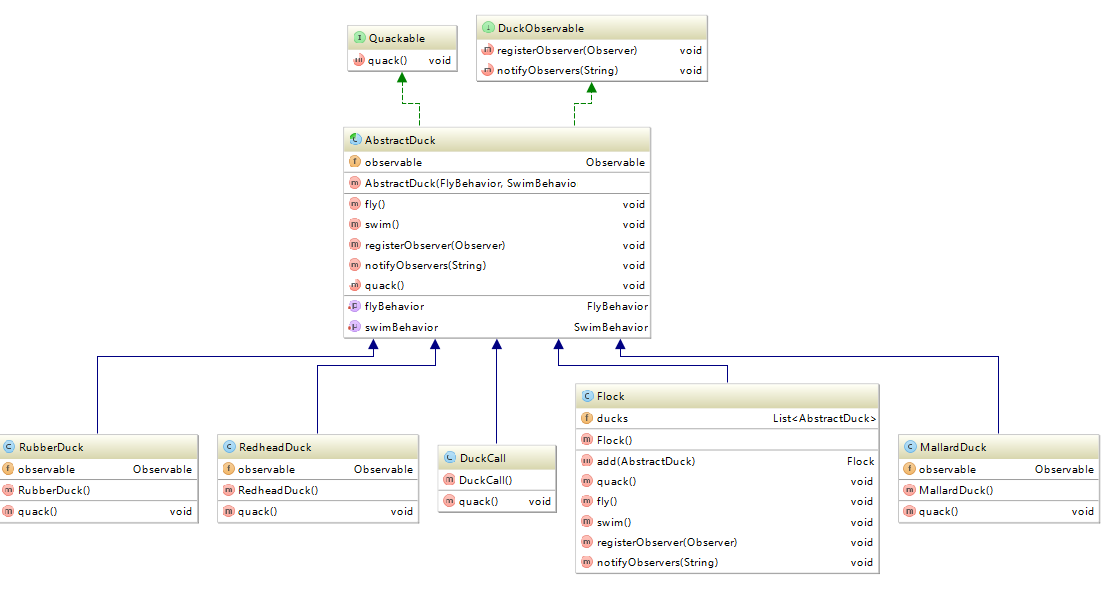
[三、 简单测试 18](#_Toc450145419)

[测试代码 18](#_Toc450145420)

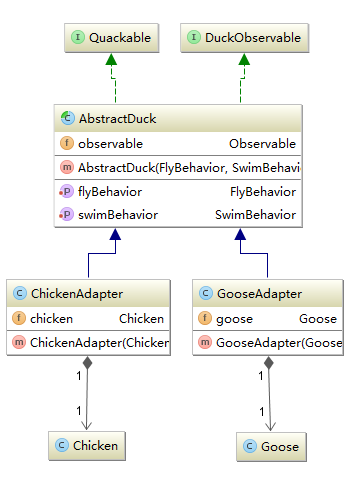
[测试结果 19](#_Toc450145421)

# 设计

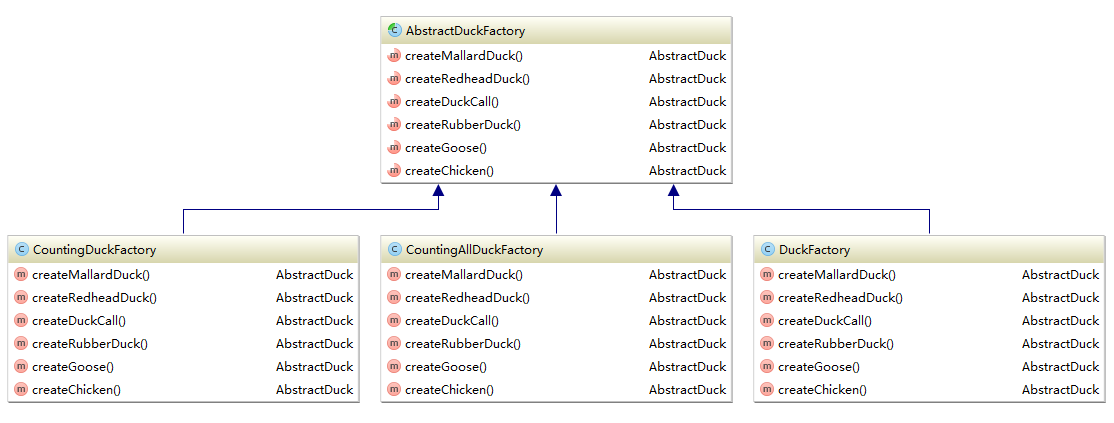
## 鸭子类继承图



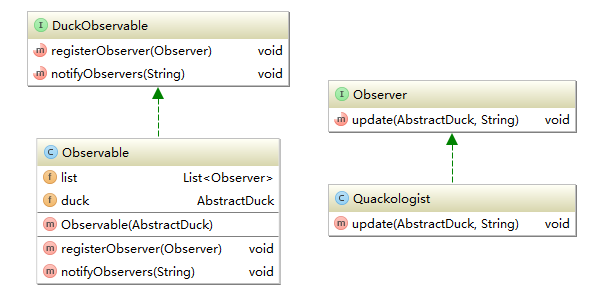
## 鸡、鸭子适配器模式继承图



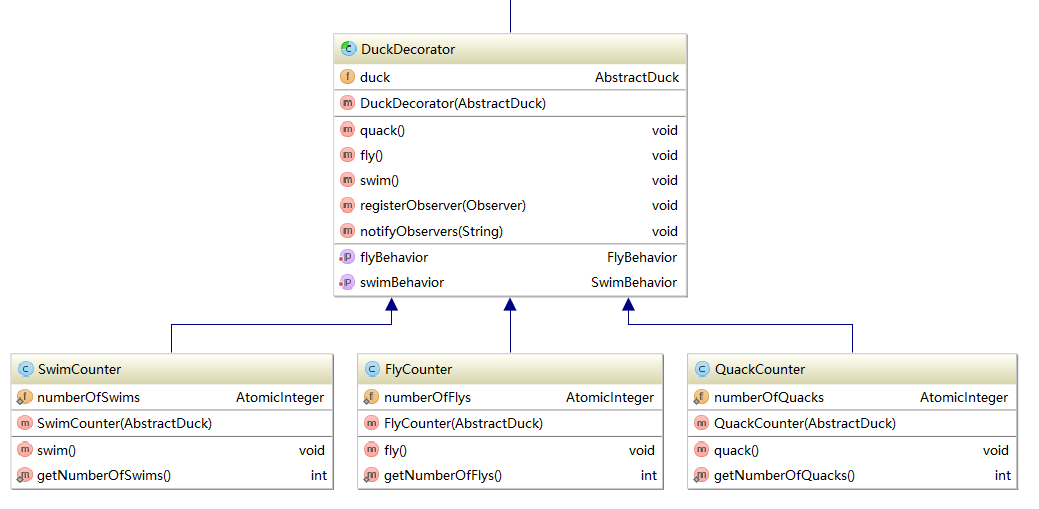
## 工厂继承图



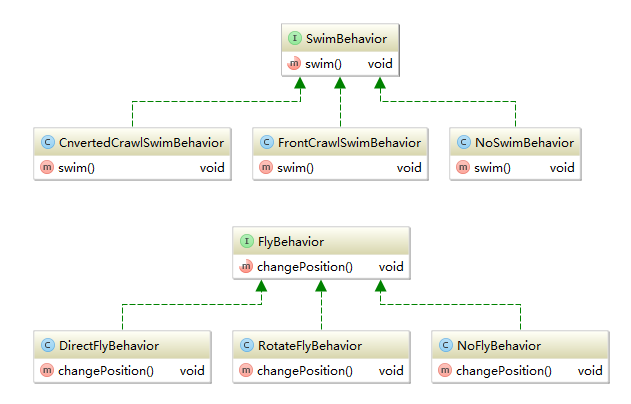
## 观察者模式继承图



## 计数器（装饰者）继承图



## 策略模式（飞行、游泳）类图



# 实现

## Quackable接口实现

**public interface** Quackable {  
 **void** quack();  
}

## 抽象鸭子实现

**package** com.zzkun.compound.ducks;  
  
**import** com.zzkun.compound.behavior.fly.FlyBehavior;  
**import** com.zzkun.compound.behavior.swim.SwimBehavior;  
**import** com.zzkun.compound.observer.DuckObservable;  
**import** com.zzkun.compound.observer.Observable;  
**import** com.zzkun.compound.observer.Observer;  
  
*/\*\*  
 \* 抽象鸭子基类  
 \* Created by kun on 2016/4/27.  
 \*/***public abstract class** AbstractDuck **implements** Quackable, DuckObservable {  
  
 **protected** FlyBehavior **flyBehavior**;  
 **protected** SwimBehavior **swimBehavior**;  
 **protected** Observable **observable**;  
  
 **public** AbstractDuck(FlyBehavior flyBehavior, SwimBehavior swimBehavior) {  
 **this**.**flyBehavior** = flyBehavior;  
 **this**.**swimBehavior** = swimBehavior;  
 **observable** = **new** Observable(**this**);  
 }  
  
 **public void** setFlyBehavior(FlyBehavior flyBehavior) {  
 **this**.**flyBehavior** = flyBehavior;  
 }  
  
 **public void** setSwimBehavior(SwimBehavior swimBehavior) {  
 **this**.**swimBehavior** = swimBehavior;  
 }  
  
  
 **public void** fly() {  
 **flyBehavior**.changePosition();  
 notifyObservers(**"fly"**);  
 }  
  
 **public void** swim() {  
 **swimBehavior**.swim();  
 notifyObservers(**"swim"**);  
 }  
  
 @Override  
 **public void** registerObserver(Observer observer) {  
 **observable**.registerObserver(observer);  
 }  
  
 @Override  
 **public void** notifyObservers(String info) {  
 **observable**.notifyObservers(info);  
 }  
  
 @Override  
 **public abstract void** quack();  
}

## 鸭鸣器、绿头鸭、红头鸭、橡皮鸭实现

**package** com.zzkun.compound.ducks;  
  
**import** com.zzkun.compound.behavior.fly.NoFlyBehavior;  
**import** com.zzkun.compound.behavior.swim.NoSwimBehavior;  
**import** com.zzkun.compound.observer.Observable;  
**import** com.zzkun.compound.observer.Observer;  
  
*/\*\*  
 \* 鸭鸣器  
 \* Created by kun on 2016/4/27.  
 \*/***public class** DuckCall **extends** AbstractDuck {  
  
 **public** DuckCall() {  
 **super**(**new** NoFlyBehavior(), **new** NoSwimBehavior());  
 **observable** = **new** Observable(**this**);  
 }  
  
 @Override  
 **public void** quack() {  
 System.***out***.println(**"Kwak"**);  
 notifyObservers(**"quack"**);  
 }  
}

**public class** MallardDuck **extends** AbstractDuck {  
  
 Observable **observable**;  
  
 **public** MallardDuck() {  
 **super**(**new** DirectFlyBehavior(), **new** FrontCrawlSwimBehavior());  
 **observable** = **new** Observable(**this**);  
 }  
  
 @Override  
 **public void** quack() {  
 System.***out***.println(**"Quack"**);  
 notifyObservers(**"quack"**);  
 }  
}

**package** com.zzkun.compound.ducks;  
  
**import** com.zzkun.compound.behavior.fly.DirectFlyBehavior;  
**import** com.zzkun.compound.behavior.swim.CnvertedCrawlSwimBehavior;  
**import** com.zzkun.compound.observer.Observable;  
**import** com.zzkun.compound.observer.Observer;  
  
*/\*\*  
 \* 红头鸭  
 \* Created by kun on 2016/4/27.  
 \*/***public class** RedheadDuck **extends** AbstractDuck {  
  
 Observable **observable**;  
  
 **public** RedheadDuck() {  
 **super**(**new** DirectFlyBehavior(), **new** CnvertedCrawlSwimBehavior());  
 **observable** = **new** Observable(**this**);  
 }  
  
 @Override  
 **public void** quack() {  
 System.***out***.println(**"Quack"**);  
 notifyObservers(**"quack"**);  
 }  
}

**package** com.zzkun.compound.ducks;  
  
**import** com.zzkun.compound.behavior.fly.RotateFlyBehavior;  
**import** com.zzkun.compound.behavior.swim.CnvertedCrawlSwimBehavior;  
**import** com.zzkun.compound.observer.Observable;  
**import** com.zzkun.compound.observer.Observer;  
  
*/\*\*  
 \* 橡皮鸭  
 \* Created by kun on 2016/4/27.  
 \*/***public class** RubberDuck **extends** AbstractDuck {  
  
 Observable **observable**;  
  
 **public** RubberDuck() {  
 **super**(**new** RotateFlyBehavior(), **new** CnvertedCrawlSwimBehavior());  
 **observable** = **new** Observable(**this**);  
 }  
  
 @Override  
 **public void** quack() {  
 System.***out***.println(**"Squeak"**);  
 notifyObservers(**"quack"**);  
 }  
}

## 鸭子群Flock实现

**package** com.zzkun.compound.ducks;  
  
**import** com.zzkun.compound.behavior.fly.NoFlyBehavior;  
**import** com.zzkun.compound.behavior.swim.NoSwimBehavior;  
**import** com.zzkun.compound.observer.Observer;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \* Created by kun on 2016/4/27.  
 \*/***public class** Flock **extends** AbstractDuck {  
  
 **private** List<AbstractDuck> **ducks** = **new** ArrayList<>();  
  
 **public** Flock() {  
 **super**(**new** NoFlyBehavior(), **new** NoSwimBehavior());  
 }  
  
 **public** Flock add(AbstractDuck duck) {  
 **ducks**.add(duck);  
 **return this**;  
 }  
  
 @Override  
 **public void** quack() {  
 **for**(AbstractDuck duck : **ducks**)  
 duck.quack();  
 }  
  
 @Override  
 **public void** fly() {  
 **for**(AbstractDuck duck : **ducks**)  
 duck.fly();  
 }  
  
 @Override  
 **public void** swim() {  
 **for**(AbstractDuck duck : **ducks**)  
 duck.swim();  
 }  
  
 @Override  
 **public void** registerObserver(Observer observer) {  
 **for**(AbstractDuck duck : **ducks**)  
 duck.registerObserver(observer);  
 }  
  
 @Override  
 **public void** notifyObservers(String info) {  
 *// 无需通知* }  
}

## 鸡和适配器实现

**package** com.zzkun.compound.chicken;  
  
*/\*\*  
 \* Created by kun on 2016/4/27.  
 \*/***public class** Chicken {  
 **public void** chuckle() {  
 System.***out***.println(**"Chuckle"**);  
 }  
}

**package** com.zzkun.compound.chicken;  
  
**import** com.zzkun.compound.behavior.fly.DirectFlyBehavior;  
**import** com.zzkun.compound.behavior.swim.FrontCrawlSwimBehavior;  
**import** com.zzkun.compound.behavior.swim.NoSwimBehavior;  
**import** com.zzkun.compound.ducks.AbstractDuck;  
**import** com.zzkun.compound.observer.Observable;  
  
*/\*\*  
 \* Created by kun on 2016/4/27.  
 \*/***public class** ChickenAdapter **extends** AbstractDuck {  
  
 **private** Chicken **chicken**;  
  
 **public** ChickenAdapter(Chicken chicken) {  
 **super**(**new** DirectFlyBehavior(), **new** NoSwimBehavior());  
 **this**.**chicken** = chicken;  
 **observable** = **new** Observable(**this**);  
 }  
  
 @Override  
 **public void** quack() {  
 **chicken**.chuckle();  
 notifyObservers(**"quack"**);  
 }  
}

## 鸭子和适配器实现

**package** com.zzkun.compound.goose;  
  
*/\*\*  
 \* 鸭子  
 \* Created by kun on 2016/4/27.  
 \*/***public class** Goose {  
 **public void** honk() {  
 System.***out***.println(**"Honk"**);  
 }  
}

**package** com.zzkun.compound.goose;  
  
**import** com.zzkun.compound.behavior.fly.DirectFlyBehavior;  
**import** com.zzkun.compound.behavior.swim.FrontCrawlSwimBehavior;  
**import** com.zzkun.compound.ducks.AbstractDuck;  
**import** com.zzkun.compound.ducks.Quackable;  
**import** com.zzkun.compound.observer.Observable;  
**import** com.zzkun.compound.observer.Observer;  
  
*/\*\*  
 \* Created by kun on 2016/4/27.  
 \*/***public class** GooseAdapter **extends** AbstractDuck {  
  
 **private** Goose **goose**;  
  
 **public** GooseAdapter(Goose goose) {  
 **super**(**new** DirectFlyBehavior(), **new** FrontCrawlSwimBehavior());  
 **this**.**goose** = goose;  
 **observable** = **new** Observable(**this**);  
 }  
  
 @Override  
 **public void** quack() {  
 **goose**.honk();  
 notifyObservers(**"quack"**);  
 }  
}

## 抽象工厂、普通工厂、计数器工厂实现

**public abstract class** AbstractDuckFactory {  
 **public abstract** AbstractDuck createMallardDuck();  
 **public abstract** AbstractDuck createRedheadDuck();  
 **public abstract** AbstractDuck createDuckCall();  
 **public abstract** AbstractDuck createRubberDuck();  
 **public abstract** AbstractDuck createGoose();  
 **public abstract** AbstractDuck createChicken();  
}

**public class** DuckFactory **extends** AbstractDuckFactory {  
 @Override  
 **public** AbstractDuck createMallardDuck() {  
 **return new** MallardDuck();  
 }  
  
 @Override  
 **public** AbstractDuck createRedheadDuck() {  
 **return new** RedheadDuck();  
 }  
  
 @Override  
 **public** AbstractDuck createDuckCall() {  
 **return new** DuckCall();  
 }  
  
 @Override  
 **public** AbstractDuck createRubberDuck() {  
 **return new** RubberDuck();  
 }  
  
 @Override  
 **public** AbstractDuck createGoose() {  
 **return new** GooseAdapter(**new** Goose());  
 }  
  
 @Override  
 **public** AbstractDuck createChicken() {  
 **return new** ChickenAdapter(**new** Chicken());  
 }  
}

**public class** CountingDuckFactory **extends** AbstractDuckFactory {  
 @Override  
 **public** AbstractDuck createMallardDuck() {  
 **return new** QuackCounter(**new** MallardDuck());  
 }  
  
 @Override  
 **public** AbstractDuck createRedheadDuck() {  
 **return new** QuackCounter(**new** RedheadDuck());  
 }  
  
 @Override  
 **public** AbstractDuck createDuckCall() {  
 **return new** QuackCounter(**new** DuckCall());  
 }  
  
 @Override  
 **public** AbstractDuck createRubberDuck() {  
 **return new** QuackCounter(**new** RubberDuck());  
 }  
  
 @Override  
 **public** AbstractDuck createGoose() {  
 **return new** QuackCounter(**new** GooseAdapter(**new** Goose()));  
 }  
  
 @Override  
 **public** AbstractDuck createChicken() {  
 **return new** QuackCounter(**new** ChickenAdapter(**new** Chicken()));  
 }  
}

**public class** CountingAllDuckFactory **extends** AbstractDuckFactory {  
  
 @Override  
 **public** AbstractDuck createMallardDuck() {  
 **return new** FlyCounter(**new** QuackCounter(**new** SwimCounter(**new** MallardDuck())));  
 }  
  
 @Override  
 **public** AbstractDuck createRedheadDuck() {  
 **return new** FlyCounter(**new** QuackCounter(**new** SwimCounter(**new** RedheadDuck())));  
 }  
  
 @Override  
 **public** AbstractDuck createDuckCall() {  
 **return new** FlyCounter(**new** QuackCounter(**new** SwimCounter(**new** DuckCall())));  
 }  
  
 @Override  
 **public** AbstractDuck createRubberDuck() {  
 **return new** FlyCounter(**new** QuackCounter(**new** SwimCounter(**new** RubberDuck())));  
 }  
  
 @Override  
 **public** AbstractDuck createGoose() {  
 **return new** FlyCounter(**new** QuackCounter(**new** SwimCounter(**new** GooseAdapter(**new** Goose()))));  
 }  
  
 @Override  
 **public** AbstractDuck createChicken() {  
 **return new** FlyCounter(**new** QuackCounter(**new** SwimCounter(**new** ChickenAdapter(**new** Chicken()))));  
 }  
}

## 飞行策略实现

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

**public class** DirectFlyBehavior **implements** FlyBehavior {  
 @Override  
 **public void** changePosition() {  
 System.***out***.println(**"鸭子直线飞向前方..."**);  
 }  
}

**public class** RotateFlyBehavior **implements** FlyBehavior {  
 @Override  
 **public void** changePosition() {  
 System.***out***.println(**"鸭子旋转飞行..."**);  
 }  
}

**public class** NoFlyBehavior **implements** FlyBehavior {  
 @Override  
 **public void** changePosition() {  
 System.***out***.println(**"不能飞行..."**);  
 }  
}

## 游泳策略实现

**public interface** SwimBehavior {  
 **void** swim();  
}

**public class** CnvertedCrawlSwimBehavior **implements** SwimBehavior {  
 @Override  
 **public void** swim() {  
 System.out.println(**"我正在仰泳..."**);  
 }  
}

**public class** FrontCrawlSwimBehavior **implements** SwimBehavior {  
 @Override  
 **public void** swim() {  
 System.***out***.println(**"我正在自由泳..."**);  
 }  
}

**public class** NoSwimBehavior **implements** SwimBehavior {  
 @Override  
 **public void** swim() {  
 System.***out***.println(**"我不会游泳..."**);  
 }  
}

## 观察者、被观察者实现

**public interface** DuckObservable {  
 **void** registerObserver(Observer observer);  
 **void** notifyObservers(String info);  
}

**public class** Observable **implements** DuckObservable {  
  
 List<Observer> **list** = **new** ArrayList<>();  
 AbstractDuck **duck**;  
  
 **public** Observable(AbstractDuck duck) {  
 **this**.**duck** = duck;  
 }  
  
 @Override  
 **public void** registerObserver(Observer observer) {  
 **list**.add(observer);  
 }  
  
 @Override  
 **public void** notifyObservers(String info) {  
 **for**(Observer observer : **list**)  
 observer.update(**duck**, info);  
 }  
}

**public interface** Observer {  
 **void** update(AbstractDuck duck, String info);  
}

**public class** Quackologist **implements** Observer {  
  
 @Override  
 **public void** update(AbstractDuck duck, String info) {  
 System.***out***.println(**"监视器："** + info + **" "** + duck);  
 }  
}

# 简单测试

## 测试代码

**public class** DuckTest {  
  
 **void** simulate(AbstractDuck duck) {  
 duck.quack();  
 }  
  
 **void** fly(AbstractDuck duck) {  
 duck.fly();  
 }  
  
 **void** swim(AbstractDuck duck) {  
 duck.swim();  
 }  
  
 **void** simulate(AbstractDuckFactory factory) {  
 AbstractDuck mallard1 = factory.createMallardDuck();  
 AbstractDuck mallard2 = factory.createMallardDuck();  
 Flock flockOfMallards = **new** Flock().add(mallard1).add(mallard2);  
  
 AbstractDuck redheadDuck = factory.createRedheadDuck();  
 AbstractDuck duckCall = factory.createDuckCall();  
 AbstractDuck rubberDuck = factory.createRubberDuck();  
 AbstractDuck gooseDuck = factory.createGoose();  
 Flock flockOfDucks = **new** Flock().add(redheadDuck).add(duckCall).add(rubberDuck).add(gooseDuck).add(flockOfMallards);  
  
 Quackologist quackologist = **new** Quackologist();  
 flockOfDucks.registerObserver(quackologist);  
  
 simulate(flockOfDucks);  
 System.***out***.println(**"鸭鸣次数："** + QuackCounter.*getNumberOfQuacks*());  
  
 fly(flockOfDucks);  
 System.***out***.println(**"飞行次数："** + FlyCounter.*getNumberOfFlys*());  
  
 swim(flockOfDucks);  
 System.***out***.println(**"游泳次数："** + SwimCounter.*getNumberOfSwims*());  
 }  
  
 **public static void** main(String[] args) {  
 AbstractDuckFactory duckFactory = **new** CountingAllDuckFactory();  
 **new** DuckTest().simulate(duckFactory);  
 }  
}

## 测试结果

Quack

监视器：quack com.zzkun.compound.ducks.RedheadDuck@7f31245a

Kwak

监视器：quack com.zzkun.compound.ducks.DuckCall@6d6f6e28

Squeak

监视器：quack com.zzkun.compound.ducks.RubberDuck@135fbaa4

Honk

监视器：quack com.zzkun.compound.goose.GooseAdapter@45ee12a7

Quack

监视器：quack com.zzkun.compound.ducks.MallardDuck@330bedb4

Quack

监视器：quack com.zzkun.compound.ducks.MallardDuck@2503dbd3

鸭鸣次数：6

鸭子直线飞向前方...

监视器：fly com.zzkun.compound.ducks.RedheadDuck@7f31245a

不能飞行...

监视器：fly com.zzkun.compound.ducks.DuckCall@6d6f6e28

鸭子旋转飞行...

监视器：fly com.zzkun.compound.ducks.RubberDuck@135fbaa4

鸭子直线飞向前方...

监视器：fly com.zzkun.compound.goose.GooseAdapter@45ee12a7

鸭子直线飞向前方...

监视器：fly com.zzkun.compound.ducks.MallardDuck@330bedb4

鸭子直线飞向前方...

监视器：fly com.zzkun.compound.ducks.MallardDuck@2503dbd3

飞行次数：6

我正在仰泳...

监视器：swim com.zzkun.compound.ducks.RedheadDuck@7f31245a

我不会游泳...

监视器：swim com.zzkun.compound.ducks.DuckCall@6d6f6e28

我正在仰泳...

监视器：swim com.zzkun.compound.ducks.RubberDuck@135fbaa4

我正在自由泳...

监视器：swim com.zzkun.compound.goose.GooseAdapter@45ee12a7

我正在自由泳...

监视器：swim com.zzkun.compound.ducks.MallardDuck@330bedb4

我正在自由泳...

监视器：swim com.zzkun.compound.ducks.MallardDuck@2503dbd3

游泳次数：6