\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*策略模式

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

设计原则：针对接口编程，而不是针对实现编程

设计原则：多用组合少用继承

**问题：设计一些种类的鸭子，比如绿头鸭，橡皮鸭等**

**方法：**

**1:**基本思路是先设计一个鸭子的超类，让各种种类的鸭子继承之，超类中包含鸭子共有的特性

class Duck{

quack();

swim();

display();//抽象方法，子类实现

}

class MallarDuck extends Duck{

MallarDuck();

display();//具体实现

}

class RedheadDuck extends Duck{

RedheadDuck();

display();//具体实现

}

**2:**现在想让部分鸭子能飞

首先想到在超类中加上fly()方法，但是想到有些类型鸭子fly方法不同，这样需要在每种子类中实现fly()方法,不能复用，于是另找方法，想到将fly()等方法单独出来，作为一个接口或者类，让需要的实现或继承之

作为接口还是类呢？如果作为接口，则需要的类实现这个接口，跟超类中加fly没什么区别

那就作为类吧

class fly1{

fly(){。。。};

}

class fly2{

fly(){。。。};

}

由于java中不能使用多继承，所以用组合

class MallarDuck extends Duck{

display();

fly1 flyinstant;

//怎么调用fly方法呢？用如下方法

flymethod(){

flyinstant.fly();

}

}

class RedheadDuck extends Duck{

display();

fly2 flyinstant;

//怎么调用fly方法呢？用如下方法

flymethod(){

flyinstant.fly();

}

}

这种方法有个更好的处理方法，会用到多态

interface flyimp{

fly();

}

class fly1 implements flyimp{

fly();

}

class fly2 implements flyimp{

fly();

}

class MallarDuck extends Duck{

display();

flyimp fly1;

flymethod(){

fly1.fly();

}

}

class RedheadDuck extends Duck{

display();

flyimp fly2;

flymethod(){

fly2.fly();

}

}

进一步可将flyimp 引用及flymethod放于超类中

interface flyimp{

fly();

}

class Duck{

quack();

swim();

display();//抽象方法，子类实现

flyimp flyinstant;

flymethod(){

flyinstant.fly();

}

}

class MallarDuck extends Duck{

display();

MallarDuck(){

flyinstant = new fly1();

}

}

class RedheadDuck extends Duck{

display();

RedheadDuck(){

flyinstant= new fly2();

}

}

**3:**一个完整实例

public abstract class Duck{

public Duck(){};

public void swim(){

printf(“swimming …\n”);

};//公用

public abstract void display();//抽象方法，子类实现

flyimp flyinstant;

flymethod(){

flyinstant.fly();

}

}

public class MallarDuck extends Duck{

public void display(){

printf(“mallarDuck display…\n”);

}

public MallarDuck(){

flyinstant = new fly1();

}

}

public class RedheadDuck extends Duck{

public void display(){

printf(“RedheadDuck display…\n”);

}

public RedheadDuck(){

flyinstant = new fly2();

}

}

public interface flyinter{

public void fly();

}

public class fly1 implements flyinter{

public void fly(){

printf(“fly method1…\n”);

}

}

public class fly2 implements flyinter{

public void fly(){

printf(“fly method2…\n”);

}

}

public class Simulator(){

public static void main(String[] args){

Duck duck1 = new MallarDuck();

duck1.swim();

duck1.display();

duck1.flymethod();

Duck duck2 = new RedheadDuck();

duck2.swim();

duck2.display();

duck2.flymethod();

}

}