

实验报告☑ 实践报告□

课程名称： 软件详细设计R

实验、实践名称： 实验三 行为型设计模式实验

实验、实践地点： 软件学院C2实验室

专业班级： 软件2028班 学号： 2020007657

学生姓名： 常兆海

指导教师： 曹锐

2023年 4月 4日

目录

[1 实验目的和要求： 3](#_Toc131972311)

[1.1 目的： 3](#_Toc131972312)

[1.2 要求： 3](#_Toc131972313)

[2 实验平台: 4](#_Toc131972314)

[3 主要实验内容及结果: 5](#_Toc131972315)

[3.1 实验内容 5](#_Toc131972316)

[3.1.1 迭代器模式的运用 5](#_Toc131972317)

[3.1.2 观察者模式的运用 10](#_Toc131972318)

[3.1.3 策略模式的运用 15](#_Toc131972319)

[3.2 心得体会 19](#_Toc131972320)

# 实验目的和要求：

## 目的：

* 综合实例，熟练绘制常见的行为型设计模式结构图。
* 结合实例，使用 Java 实现常见的行为型设计模式。
* 通过实验，理解不同行为型设计模式的使用动机，掌握不同行为型设计模式的特点和运用 场合，学习如何使用代码实现这些设计模式。

## 要求：

* 独立完成实验
* 书写实验报告书

# 实验平台:

JDK-1.8 idea—2022.3

# 主要实验内容及结果:

## 实验内容

### 迭代器模式的运用

#### 案例背景：

课堂教学中学习了如何使用迭代器模式来模拟电视遥控器的实现，并使用了内部类的方式来实现迭代 器。在实验中，请将迭代器从具体聚合类（电视机类）中分离出来，重新实现电视遥控器的模拟，请画出 类图并编程实现。

#### 实现步骤：

* 参照教材中实例，画出电视机遥控器的类图
* 根据类图，实现上述类的具体代码以及用户类 Client，请注意将迭代器分离出来，形成单独 的类，此外还需要实现 XMLUtil 类来从 XML 配置文件中读取具体的电视遥控器类。
* 编译并运行代码，实现电视遥控器的模拟。

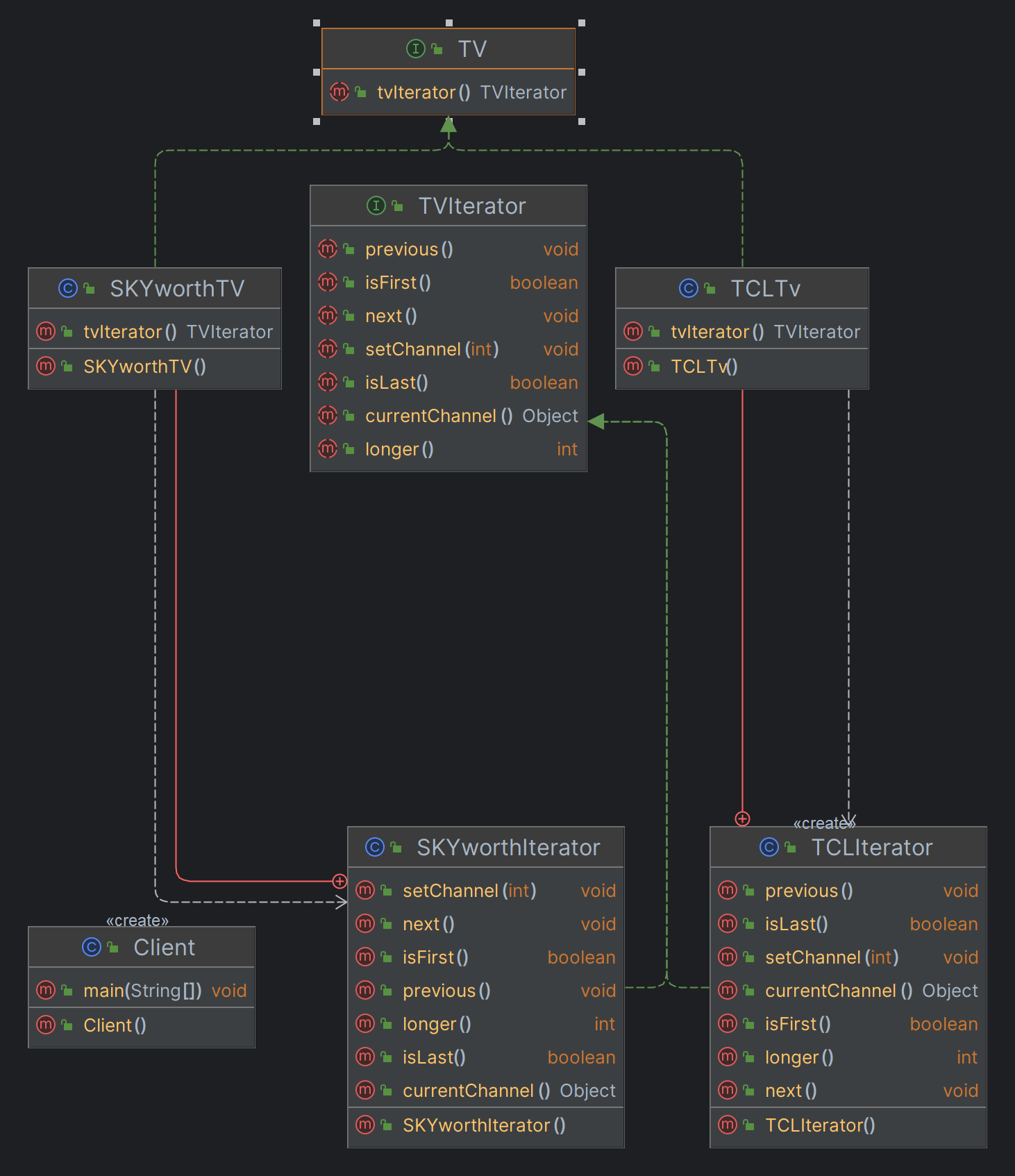
#### 案例总结：

在以下情况下可以使用迭代器模式：

* 需要访问一个聚合对象的内容而无需暴露它的内部表示时
* 需要为一个聚合对象提供多种遍历方式时
* 为遍历不同的聚合结构提供一个统一的接口，在该接口的实现类中为不同的聚合结构提供不同的遍历方式，而客户端可以一致性地操作该接口

#### 结果：

##### 类图



##### 代码实现

* TVIterator接口

public interface TVIterator {  
 void setChannel(int i);  
 void next();  
 void previous();  
 boolean isLast();  
 Object currentChannel();  
 boolean isFirst();  
  
 int longer();  
}

* TV接口

public interface TV {  
 TVIterator tvIterator();  
}

* TCLTv类

public class TCLTv implements TV{  
 private Object[] obj={"CCTV\_1","CCTV\_2","CCTV\_3","CCTV\_4","CCTV\_5","CCTV\_6","CCTV\_7","CCTV\_8","CCTV\_9","CCTV\_10","CCTV\_11"};  
 @Override  
 public TVIterator tvIterator() {  
 return new TCLIterator();  
 }  
 public class TCLIterator implements TVIterator{  
 private int index;  
 @Override  
 public void setChannel(int i) {  
 index = i;  
 }  
 @Override  
 public void next() {  
 if (index < longer()) {  
 index++;  
 }  
 }  
 @Override  
 public void previous() {  
 if (index > 0) {  
 index--;  
 }  
 }  
 @Override  
 public boolean isLast() {  
 return index == longer() - 1;  
 }  
 @Override  
 public Object currentChannel() {  
 return obj[index];  
 }  
 @Override  
 public boolean isFirst() {  
 return index == 0;  
 }  
 @Override  
 public int longer() {  
 return obj.length;  
 }  
 }  
}

* SKYworthTV类

public class SKYworthTV implements TV{  
 private Object[] obj={"CCTV\_1","CCTV\_2","CCTV\_3","CCTV\_4","CCTV\_5","CCTV\_6","CCTV\_7","CCTV\_8","CCTV\_9","CCTV\_10","CCTV\_11"};  
 @Override  
 public TVIterator tvIterator() {  
 return new SKYworthIterator();  
 }  
 public class SKYworthIterator implements TVIterator{  
 private int currentIndex;  
 @Override  
 public void setChannel(int i) {  
 currentIndex = i;  
 }  
 @Override  
 public void next() {  
 if (currentIndex < longer()) {  
 currentIndex++;  
 }  
 }  
 @Override  
 public void previous() {  
 if (currentIndex > 0) {  
 currentIndex--;  
 }  
 }  
 @Override  
 public boolean isLast() {  
 return currentIndex == longer() - 1;  
 }  
 @Override  
 public Object currentChannel() {  
 return obj[currentIndex];  
 }  
 @Override  
 public boolean isFirst() {  
 return currentIndex == 0;  
 }  
 @Override  
 public int longer() {  
 return obj.length;  
 }  
 }  
}

* 客户端类

public class Client {  
 public static void main(String[] args) {  
 TV tv;  
 TVIterator tvIterator;  
 tv = (TV) Xmlutil.getbBean();  
 tvIterator = tv.tvIterator();  
 System.out.println("常兆海 软件2028 班 2020007657 的电视机频道：");  
 while (!tvIterator.isLast()) {  
 System.out.println(tvIterator.currentChannel().toString());  
 tvIterator.next();  
 }  
 }  
}

##### 运行结果

常兆海 软件2028 班 2020007657 的电视机频道：  
CCTV\_1  
CCTV\_2  
CCTV\_3  
CCTV\_4  
CCTV\_5  
CCTV\_6  
CCTV\_7  
CCTV\_8  
CCTV\_9  
CCTV\_10

进程已结束,退出代码0



### 观察者模式的运用

#### 案例背景：

某在线股票系统需要提供以下功能：当股票购买者所购买的某只股票价格变化幅度达到 5%时，系统 将自动发送通知（包括新价格）给购买该股票的股民。现使用观察者模式设计该系统，绘制类图并编程实 现

#### 实现步骤：

* 根据题意，画出在线股票系统的类图，类图中应包括目标类 Stock，抽象观察者 Investor 以及具体 观察者 ConcreteInvestor。Stock 类中应该包含添加观察者的功能 attach（），移除观察者的功能 detach（），获取股票名称 getStockName（），设定股票名称 setStockName（），设定股票价格 setPrice（），获取股票功能 getPrice（）以及通知观察者的功能 notifyInvestor（）；观察者应该 有能够根据观察目标的改变作出反应的 upDate（）方法
* 根据类图，实现上述类的具体代码以及用户类 Client。
* 编译并运行程序，使得股民能够在价格变化超过 5%的时候收到通知。

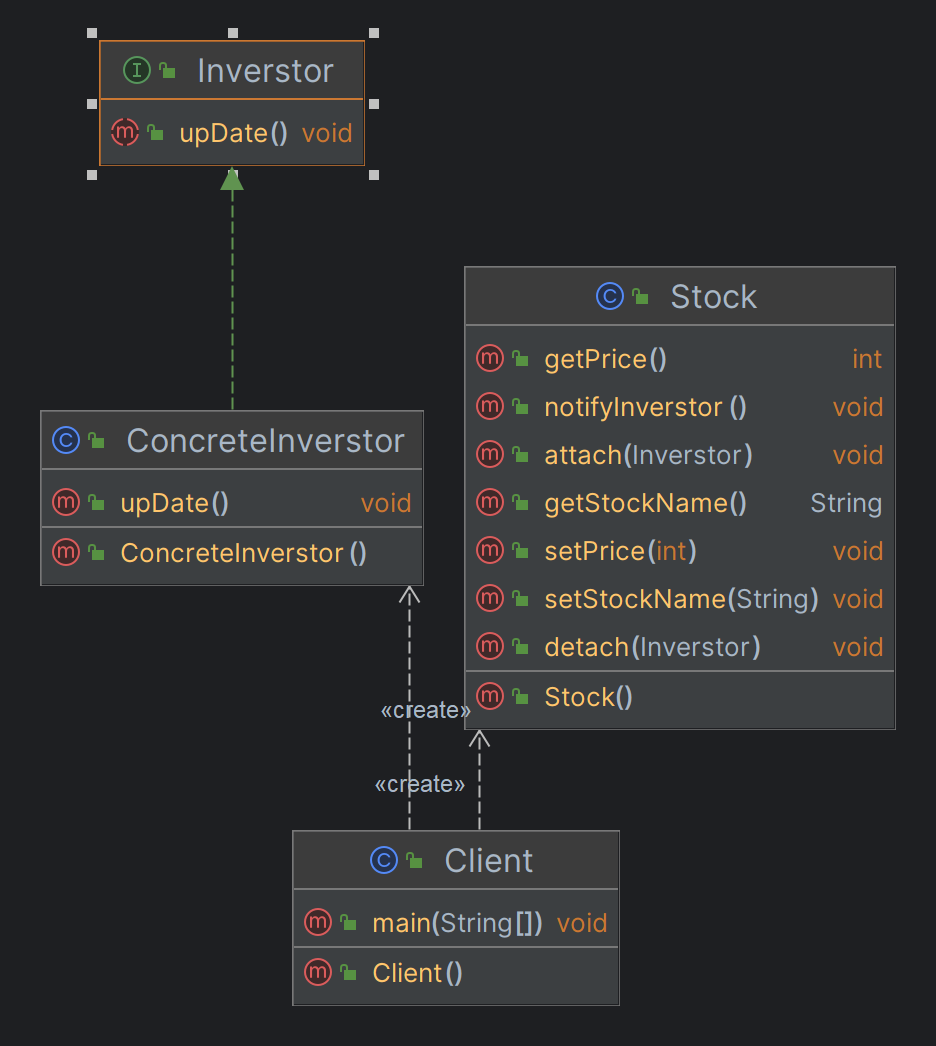
#### 案例总结：

在以下情况可以使用观察者模式：

* 一个抽象模型有两个方面，其中一个方面依赖于另一个方面，将这两个方面封装在独立的对象中 使它们可以各自独立地改变和复用
* 一个对象的改变将导致一个或多个其他对象发生改变，并且不知道具体有多少对象将发生改变， 也不知道这些对象是谁
* 需要在系统中创建一个触发链

#### 结果：

##### 类图



##### 代码实现

* Inverstor接口

public interface Inverstor {  
 void upDate();  
}

* Stock类

public class Stock {  
 protected ArrayList observes=new ArrayList();  
 protected String stockName;  
 protected ArrayList<Integer> price=new ArrayList();  
 public void attach(Inverstor observer){  
 observes.add(observer);  
 }  
 public void detach(Inverstor observer){  
 observes.remove(observer);  
 }  
 public String getStockName() {  
 return stockName;  
 }  
 public void setStockName(String stockName) {  
 this.stockName = stockName;  
 }  
 public int getPrice() {//获取增长额度  
 int i = price.size();  
 int a1 = price.get(i-1);  
 int a2 = price.get(i - 2);  
 return (a2 - a1) / a1 \* 100;  
 }  
 public void setPrice(int prices){  
 price.add(prices);  
 }  
 public void notifyInverstor () {  
 System.out.println("股票价格变化提醒:");  
 if (getPrice() > 5) {  
 for (Object o:observes){  
 ((Inverstor)o).upDate();  
 }  
 }  
// else System.out.println("股票价格影响" + getPrice() + "%， 不大");  
 }  
 }

* ConcreteInverstor类

public class ConcreteInverstor implements Inverstor{  
 @Override  
 public void upDate() {  
 System.out.println("常兆海 2020007657 软件2028班的股票价格发生变化");  
 }  
}

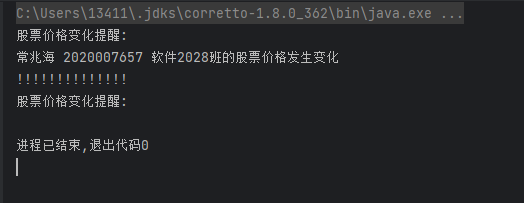
* 客户端类

public class Client {  
 public static void main(String[] args) {  
 Stock stock = new Stock();  
 Inverstor inverstor = new ConcreteInverstor();  
 stock.attach(inverstor);  
 stock.setStockName("股票：常兆海2020007657");  
 stock.getStockName();  
 stock.setPrice(100);  
 stock.setPrice(50);  
 stock.notifyInverstor();  
 System.out.println("!!!!!!!!!!!!!!");  
 stock.setPrice(100);  
 stock.setPrice(102);  
 stock.notifyInverstor();  
 }  
}

##### 运行结果

股票价格变化提醒:  
常兆海 2020007657 软件2028班的股票价格发生变化  
!!!!!!!!!!!!!!  
股票价格变化提醒:

进程已结束,退出代码0



### 策略模式的运用

#### 案例背景：

在介绍策略模式时，我们讲到了从不同角度出发，可以使用不同的出行策略的例子，教材中已经提供 了“旅游出行策略”的类图，用 Java 代码模拟实现“旅游出行策略”实例，要求使用配置文件存储具体策 略类的类名。在此基础上再增加一种新的旅游出行方式，如徒步旅行（WalkStrategy），修改原有类图及 方法，并编程实现。（教材 403 页第 1 题）

#### 实现步骤：

* 根据书上“旅游出行策略”类图，增加新的徒步旅行方式，画出新的类图。
* 根据类图，编写并实现代码，使用 XMLUtil 类来从 XML 文件中读取相应类名。
* 编译并运行代码，使代码能够模拟旅游出行策略。

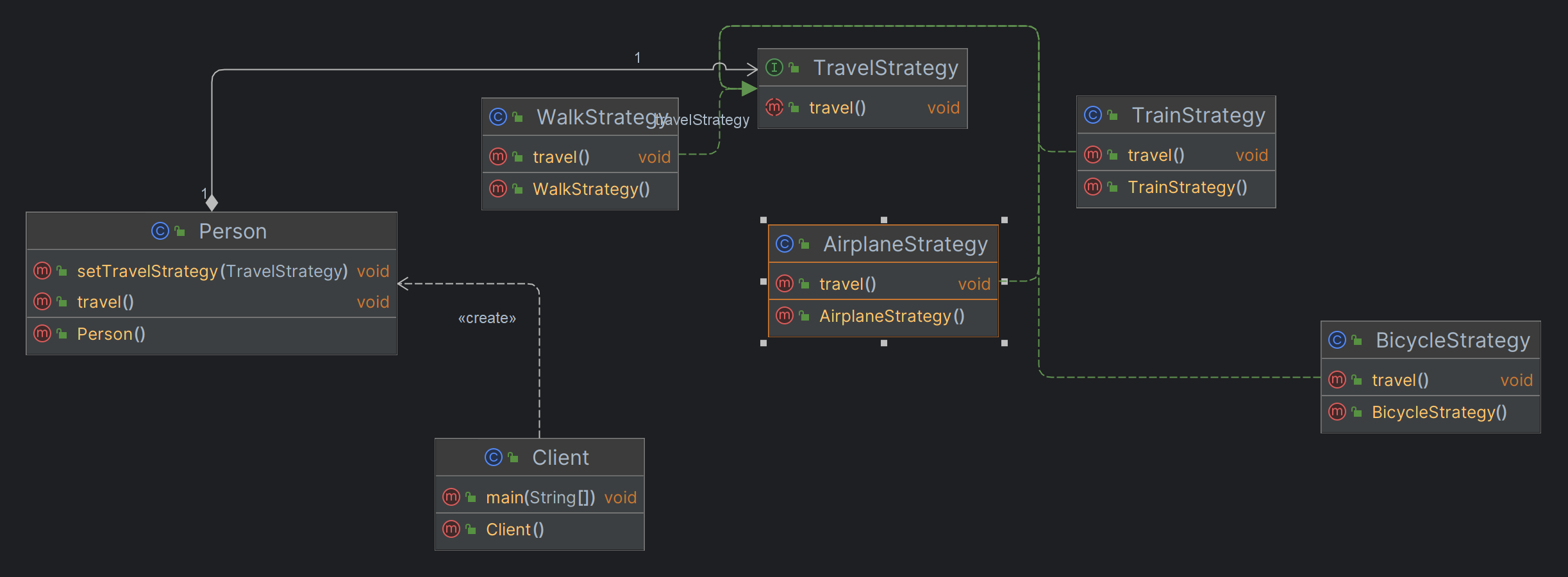
#### 案例总结：

在以下情况下可以使用策略模式：

* 如果在一个系统里面有许多类，他们之间的区别仅在于他们的行为，使用策略模式可以动态的让 一个对象在许多行为中选择一种行为。
* 一个系统需要动态的在几种算法中选择一种，那么可以将这些算法封装到一个个的具体算法类 中，而这些算法类是一个抽象算法类的子类。这些具体算法类有统一的接口，由于多态性原则， 客户端可以选择任意一个具体算法类，并只需要维持一个抽象算法类的对象。
* 如果一个对象有很多的行为，可以使用策略模式把这些行为转移到相应的具体策略类里面，这样 可以避免使用难以维护的多重条件选择语句。
* 不需要客户端直到复杂的，与算法相关的数据结构，在具体策略类中封装算法和相关的数据结构， 提高算法的保密性和安全性

#### 结果：

##### 类图



##### 代码实现

* TravelStrategy接口

public interface TravelStrategy {  
 void travel();  
}

* WalkStrategy 类

public class WalkStrategy implements TravelStrategy {  
 @Override  
 public void travel() {  
 System.out.println("走路去旅行");  
 }  
}

* TrainStrategy类

public class TrainStrategy implements TravelStrategy{  
 @Override  
 public void travel() {  
 System.out.println("坐火车");  
 }  
}

* BicycleStrategy类

public class BicycleStrategy implements TravelStrategy{  
 @Override  
 public void travel() {  
 System.out.println("常兆海 软件2028 班 2020007657选择 骑自行车");  
 }  
  
}

* Person类

public class Person {  
 private TravelStrategy travelStrategy;  
  
 public void setTravelStrategy(TravelStrategy travelStrategy) {  
 this.travelStrategy = travelStrategy;  
 }  
 public void travel(){  
 travelStrategy.travel();  
 }  
}

<?xml version="1.0"?>  
<config>  
 <className>WalkStrategy</className>  
</confi

* Xmlutil类

public class Xmlutil {  
 public static Object getbBean(){  
 try {  
 DocumentBuilderFactory documentBuilderFactory= DocumentBuilderFactory.newInstance();  
 DocumentBuilder builder=documentBuilderFactory.newDocumentBuilder();  
 Document document;  
 document= builder.parse(new File("src/main/java/实验三三/config.xml"));  
 NodeList nodeList=null;  
 Node classNode=null;  
 String cNamebefore="实验三三.";  
 String cName=null;  
 nodeList=document.getElementsByTagName("className");  
 classNode=nodeList.item(0).getFirstChild();  
 cName=classNode.getNodeValue();  
 Class c=Class.forName(cNamebefore+cName);  
 Object onj=c.newInstance();  
 return onj;  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
 }

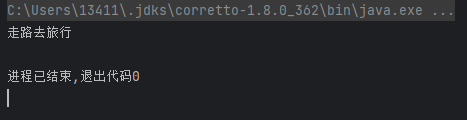
* 客户端类

public class Client {  
 public static void main(String[] args) {  
  
 Person person = new Person();  
 TravelStrategy tra= (TravelStrategy) Xmlutil.getbBean();  
 person.setTravelStrategy(tra);  
 person.travel();  
 }  
}

##### 运行结果

走路去旅行

进程已结束,退出代码0



## 心得体会

设计模式是前人结合实践总结出来的一套行之有效的理论，能够将实际业务需求转换为技术实现时，使系统更具有维护性、扩展性。在学习设计模式时，我不禁赞叹它巧妙的思路和优雅的方式。通过本此实验的学习，更好的理解了行为型设计模式，将理论学习中出现的不足在本次实践中得到了很好的补充。