

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： 行为型模式

实验、实践地点： 行勉A105

专业班级： 软件2121班 学号： 2020005731

学生姓名： 张帅

指导教师： 曹若琛

2023年 3 月 30 日

目录

[一、实验目的和要求 3](#_Toc12455)

[二、实验平台 3](#_Toc21102)

[三、主要实验内容及结果 3](#_Toc24016)

[1.迭代器模式的运用 3](#_Toc18107)

[1.1 实验内容 3](#_Toc4923)

[1.2 代码实现 4](#_Toc24228)

[1.3 运行结果 7](#_Toc21914)

[2.观察者模式的运用 8](#_Toc24917)

[2.1 实验内容 8](#_Toc14273)

[2.2 代码实现 9](#_Toc9561)

[2.3 运行结果 11](#_Toc12521)

[3.策略模式的运用 12](#_Toc23646)

[3.1 实验内容 12](#_Toc19970)

[3.2 代码实现 13](#_Toc16833)

[3.3 运行结果 14](#_Toc24178)

[四、心得体会 14](#_Toc27336)

**一、实验目的和要求**

（1）实验目的

1．综合实例，熟练绘制常见的行为型设计模式结构图。

2. 结合实例，使用 Java 实现常见的行为型设计模式。

3. 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。

（2）实验要求

1．独立完成实验。

2．书写实验报告书。

**二、实验平台**

使用JDK-11.0.1，IntelliJ IDEA 2020.3.1x64，笔记本电脑。

**三、主要实验内容及结果**

**1.迭代器模式的运用**

**1.1 实验内容**

（1）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。

（2）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。

（3）案例总结：

在以下情况下可以使用迭代器模式：

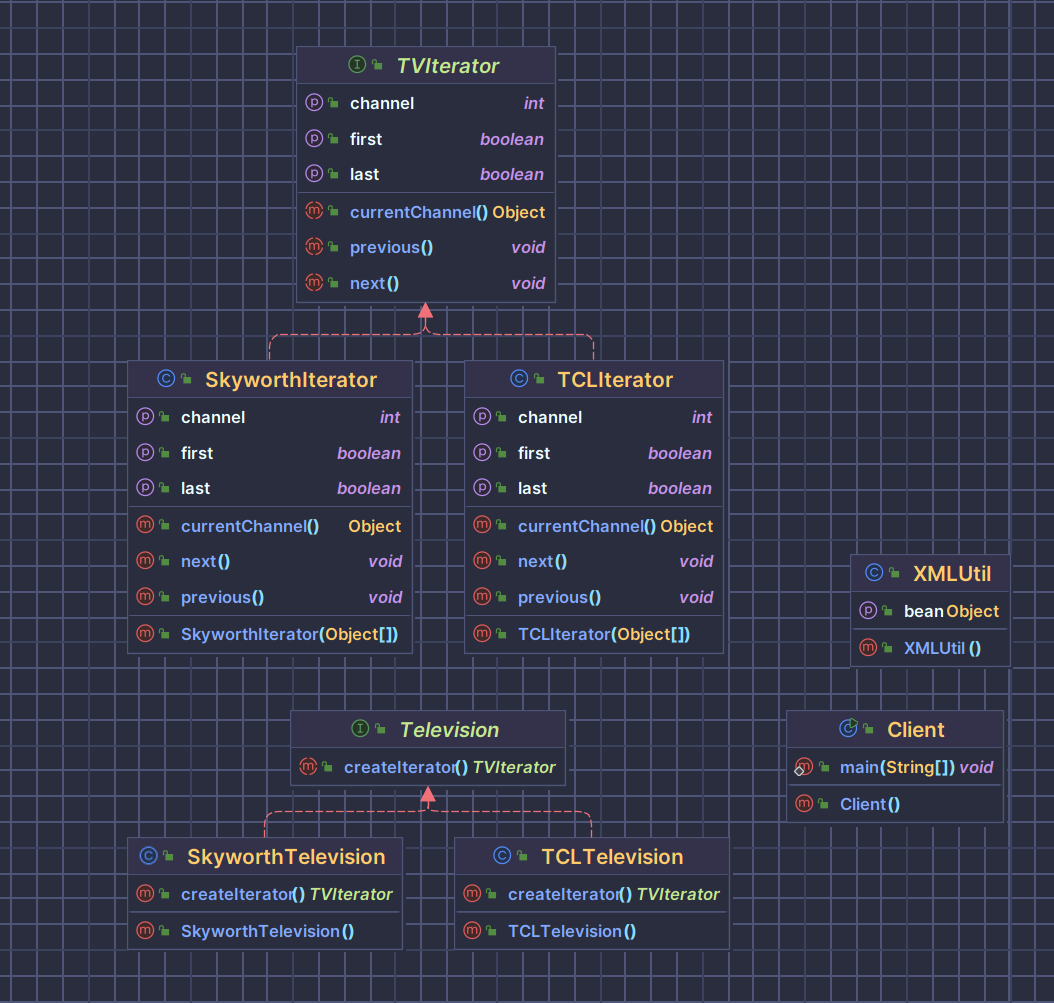
 需要访问一个聚合对象的内容而无需暴露它的内部表示时。

 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。

**1.2 代码实现**

绘制类图：



**抽象遥控器类 TVIterator**：

package edu.zhshio.iterator.lab.support;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午6:12  
 \*/  
  
public interface TVIterator {  
  
 void setChannel(int i);  
  
 void next();  
  
 void previous();  
  
 boolean isLast();  
  
 Object currentChannel();  
  
 boolean isFirst();  
}

**具体遥控器类 TCLIterator：**

package edu.zhshio.iterator.lab.iterators;  
  
import edu.zhshio.iterator.lab.support.TVIterator;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午6:31  
 \*/  
  
public class TCLIterator implements TVIterator {  
  
 private final Object[] objs;  
  
 private int cur;  
  
  
 @Override  
 public void setChannel(int i) {  
 if (i > objs.length) {  
 throw new RuntimeException("越界警告, 设置失败");  
 }  
 this.cur = i;  
 }  
  
 @Override  
 public void next() {  
 if (cur < objs.length) {  
 cur++;  
 }  
 }  
  
 @Override  
 public void previous() {  
 if (cur > 0) {  
 cur--;  
 }  
 }  
  
 @Override  
 public boolean isLast() {  
 return cur == objs.length;  
 }  
  
 @Override  
 public Object currentChannel() {  
 return objs[cur];  
 }  
  
 @Override  
 public boolean isFirst() {  
 return cur == 0;  
 }  
  
 public TCLIterator(Object[] objs) {  
 this.objs = objs;  
 cur = 0;  
 }  
  
}

**抽象电视类 Television：**

package edu.zhshio.iterator.lab.support;  
  
public interface Television {  
  
 TVIterator createIterator();  
  
}

**具体电视类 SkyworthTelevision：**

package edu.zhshio.iterator.lab.tvs;  
  
import edu.zhshio.iterator.lab.iterators.SkyworthIterator;  
import edu.zhshio.iterator.lab.support.TVIterator;  
import edu.zhshio.iterator.lab.support.Television;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午6:17  
 \*/  
  
public class SkyworthTelevision implements Television {  
  
 private Object[] objs = {  
 "CCTV-1",  
 "CCTV-2",  
 "CCTV-3",  
 "CCTV-4",  
 "CCTV-5",  
 "CCTV-6",  
 "CCTV-7",  
 "CCTV-8"  
 };  
 @Override  
 public TVIterator createIterator() {  
 return new SkyworthIterator(objs);  
 }  
  
}

**具体电视类 TCLTelevision：**

package edu.zhshio.iterator.lab.tvs;  
  
import edu.zhshio.iterator.lab.iterators.TCLIterator;  
import edu.zhshio.iterator.lab.support.TVIterator;  
import edu.zhshio.iterator.lab.support.Television;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午6:18  
 \*/  
  
public class TCLTelevision implements Television {  
  
 private Object[] objs = {  
 "湖南卫视",  
 "北京卫视",  
 "上海卫视",  
 "湖北卫视",  
 "黑龙江卫视",  
 "山西卫视",  
 "山东卫视",  
 "广西卫视",  
 "广东卫视"  
 };  
  
 @Override  
 public TVIterator createIterator() {  
 return new TCLIterator(objs);  
 }  
  
}

**辅助文件 XMLUtil：**

package edu.zhshio.iterator.lab.utils;  
  
  
  
import org.w3c.dom.Document;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.parsers.ParserConfigurationException;  
import java.io.File;  
import java.io.IOException;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/3/7 11:15  
 \*/  
  
public class XMLUtil {  
  
 public static Object getBean() {  
  
 DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();  
 try {  
 DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();  
 Document document = documentBuilder.parse(new File("src/main/java/edu/zhshio/iterator/lab/resource/beanconfig.xml"));  
  
 NodeList mealBuilderNames = document.getElementsByTagName("beanName");  
 Node mealBuilderName = mealBuilderNames.item(0).getFirstChild();  
 String mealBuilerNameVal = mealBuilderName.getNodeValue().trim();  
  
 Class<?> aClass = Class.forName(mealBuilerNameVal);  
 return aClass.newInstance();  
  
 } catch (ParserConfigurationException e) {  
 throw new RuntimeException(e);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 } catch (ClassNotFoundException e) {  
 throw new RuntimeException(e);  
 } catch (InstantiationException e) {  
 throw new RuntimeException(e);  
 } catch (IllegalAccessException e) {  
 throw new RuntimeException(e);  
 } catch (org.xml.sax.SAXException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

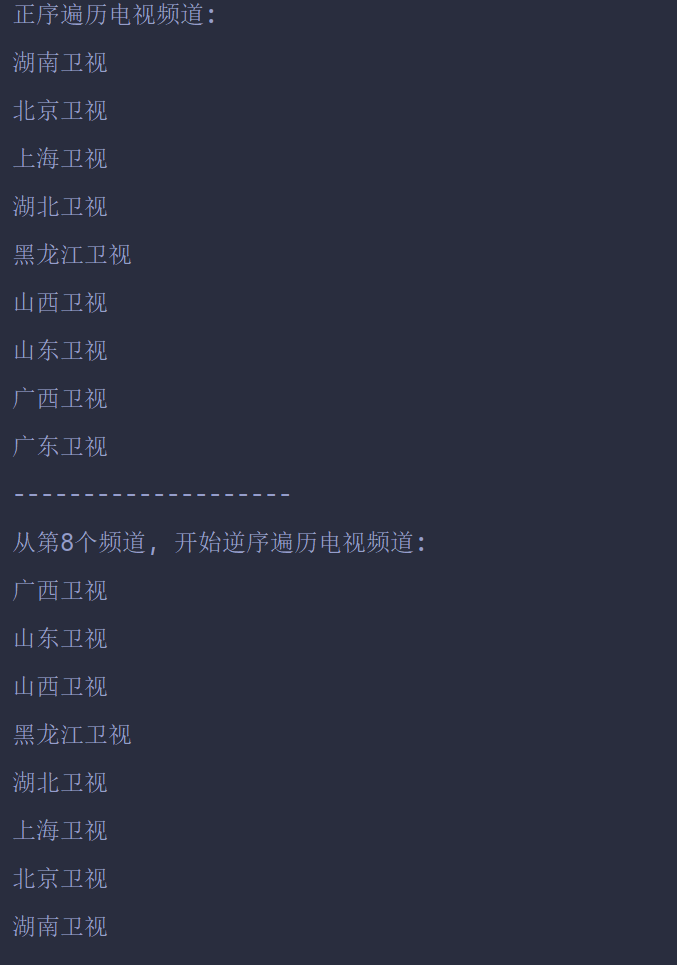
**配置文件 beanconfig.xml:**

<?xml version="1.0"?>  
<config>  
 <beanName>edu.zhshio.iterator.lab.tvs.TCLTelevision</beanName>  
</config>

**用户测试类 Client：**

package edu.zhshio.iterator.lab;  
  
import edu.zhshio.iterator.lab.support.TVIterator;  
import edu.zhshio.iterator.lab.support.Television;  
import edu.zhshio.iterator.lab.utils.XMLUtil;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午6:45  
 \*/  
  
public class Client {  
 public static void main(String[] args) {  
 Television tv = (Television) XMLUtil.getBean();  
 display(tv);  
 System.out.println("--------------------");  
 reverseDisplay(tv, 8);  
 }  
  
 public static void display(Television tv) {  
 TVIterator i = tv.createIterator();  
 System.out.println("正序遍历电视频道:");  
 while (!i.isLast()) {  
 System.out.println(i.currentChannel().toString());  
 i.next();  
 }  
 }  
  
 public static void reverseDisplay(Television tv, int last) {  
 TVIterator i = tv.createIterator();  
 System.out.println("从第" + last + "个频道, 开始逆序遍历电视频道:");  
 i.setChannel(last);  
 while (!i.isFirst()) {  
 i.previous();  
 System.out.println(i.currentChannel().toString());  
 }  
 }  
}

**1.3 运行结果**



**2.观察者模式的运用**

**2.1 实验内容**

（1）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。

（2）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。

（4）案例总结：

在以下情况可以使用观察者模式：

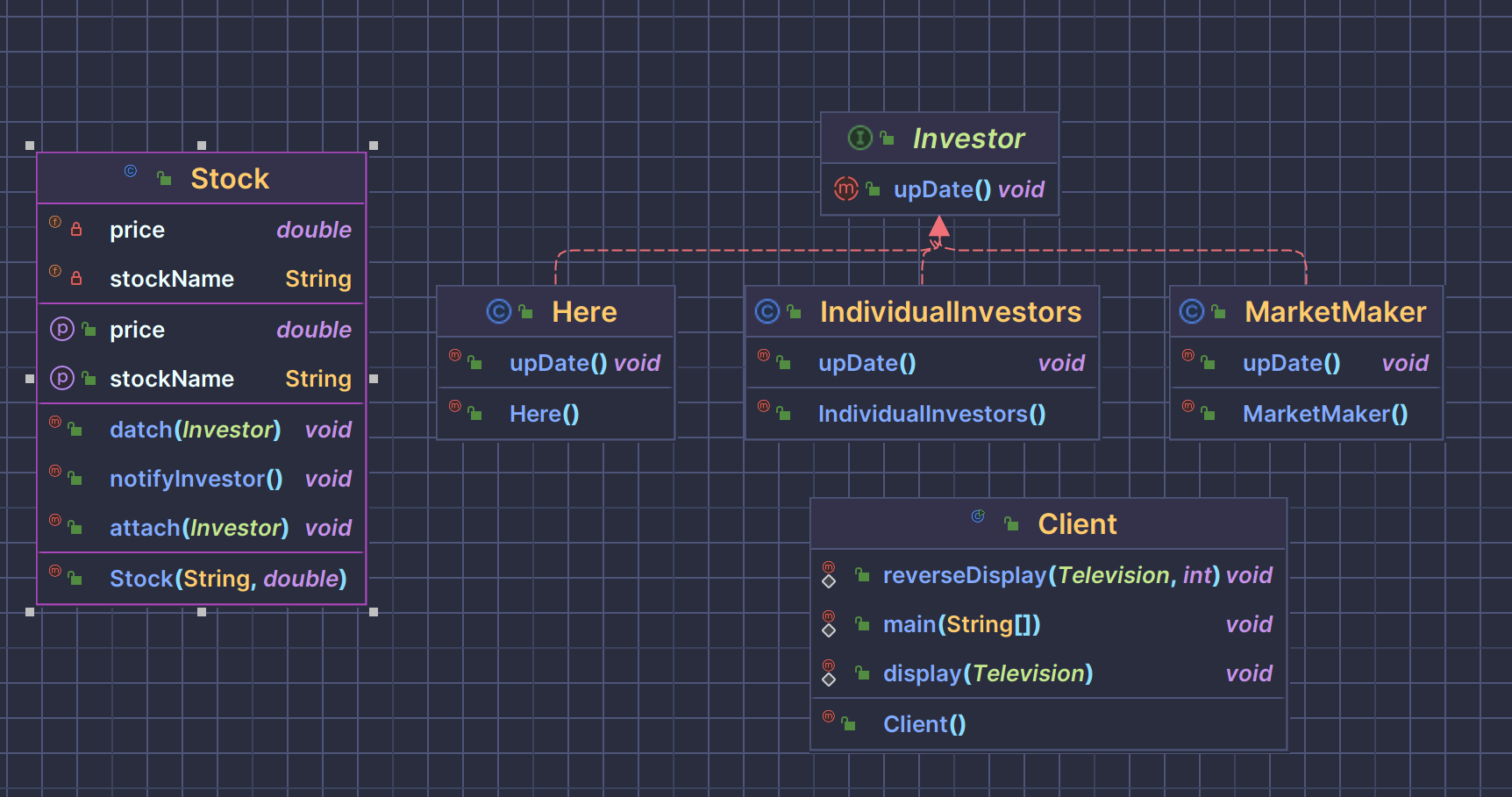
 一个抽象模型有两个方面，其中一个方面依赖于另一个方面，将这两个方面封装在独立的对象中使它们可以各自独立地改变和复用。

 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。

**2.2 代码实现**

绘制类图：



**目标类 Stock：**

package edu.zhshio.observer.lab.stock;  
  
import edu.zhshio.observer.lab.support.Investor;  
  
  
import java.util.LinkedList;  
import java.util.List;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:20  
 \*/  
  
public class Stock {  
  
 private List<Investor> observers;  
  
 private String stockName;  
 private double price;  
  
 public void attach(Investor investor) {  
 observers.add(investor);  
 }  
  
 public void datch(Investor investor) {  
 observers.removeIf(observer -> investor.equals(observer));  
 }  
  
 public void notifyInvestor() {  
 observers.stream()  
 .forEach(observer -> {  
 observer.upDate();  
 });  
 }  
  
 public String getStockName() {  
 return stockName;  
 }  
  
 public void setStockName(String stockName) {  
 this.stockName = stockName;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 double change = Math.abs(price - this.price);  
 System.out.println(stockName + ": " + this.price + " -> " + price);  
 if (change >= price \* 5e-2) {  
 System.out.println("------------众生象------------");  
 notifyInvestor();  
 System.out.println("-----------------------------");  
 }  
 this.price = price;  
 }  
  
 public Stock(String stockName, double price) {  
 this.observers = new LinkedList<>();  
 this.stockName = stockName;  
 this.price = price;  
 }  
}

**抽象观察者类 Investor：**

public interface Investor {  
 public void upDate();  
}

**具体观察者类 Here：**

package edu.zhshio.observer.lab.investors;  
  
import edu.zhshio.observer.lab.support.Investor;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:38  
 \*/  
  
public class Here implements Investor {  
 @Override  
 public void upDate() {  
 System.out.println("大户: 大家抓紧机会抄底, 牛市要来啦");  
 }  
}

**具体观察者类 IndividualInvestors：**

package edu.zhshio.observer.lab.investors;  
  
import edu.zhshio.observer.lab.support.Investor;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:35  
 \*/  
  
public class IndividualInvestors implements Investor {  
  
 @Override  
 public void upDate() {  
 System.out.println("散户: 完了, 又要被割韭菜啦");  
 }  
}

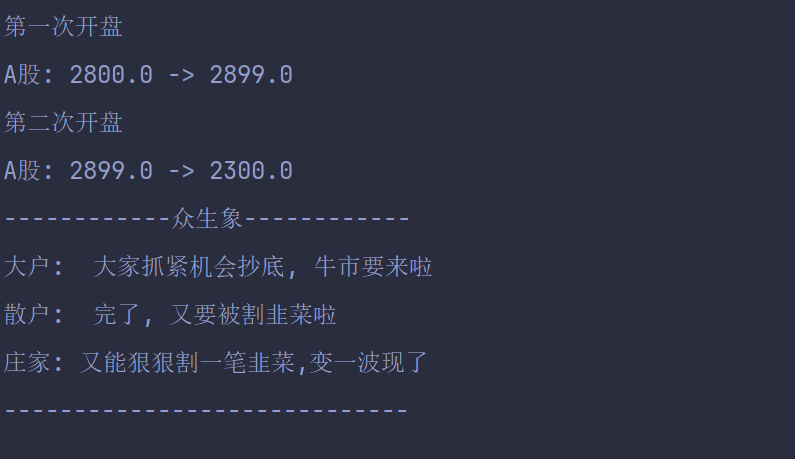
**具体观察者类MarketMaker：**

package edu.zhshio.observer.lab.investors;  
  
import edu.zhshio.observer.lab.support.Investor;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:36  
 \*/  
  
public class MarketMaker implements Investor {  
 @Override  
 public void upDate() {  
 System.out.println("庄家: 又能狠狠割一笔韭菜,变一波现了");  
 }  
}

**用户测试类 Client：**

package edu.zhshio.observer.lab;  
  
import edu.zhshio.observer.lab.investors.Here;  
import edu.zhshio.observer.lab.investors.IndividualInvestors;  
import edu.zhshio.observer.lab.investors.MarketMaker;  
import edu.zhshio.observer.lab.stock.Stock;  
  
import java.util.concurrent.TimeUnit;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:38  
 \*/  
  
public class Client {  
 public static void main(String[] args) throws InterruptedException {  
 Stock stockA = new Stock("A股", 2800);  
 Here here = new Here();  
 IndividualInvestors individualInvestors = new IndividualInvestors();  
 MarketMaker marketMaker = new MarketMaker();  
  
 stockA.attach(here);  
 stockA.attach(individualInvestors);  
 stockA.attach(marketMaker);  
  
 System.out.println("第一次开盘");  
 stockA.setPrice(2899);  
 TimeUnit.SECONDS.sleep(3);  
 System.out.println("第二次开盘");  
 stockA.setPrice(2300);  
 }  
}

**2.3 运行结果**



**3.策略模式的运用**

**3.1 实验内容**

（1）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）

（2）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。

（3）案例总结：

在以下情况下可以使用策略模式：

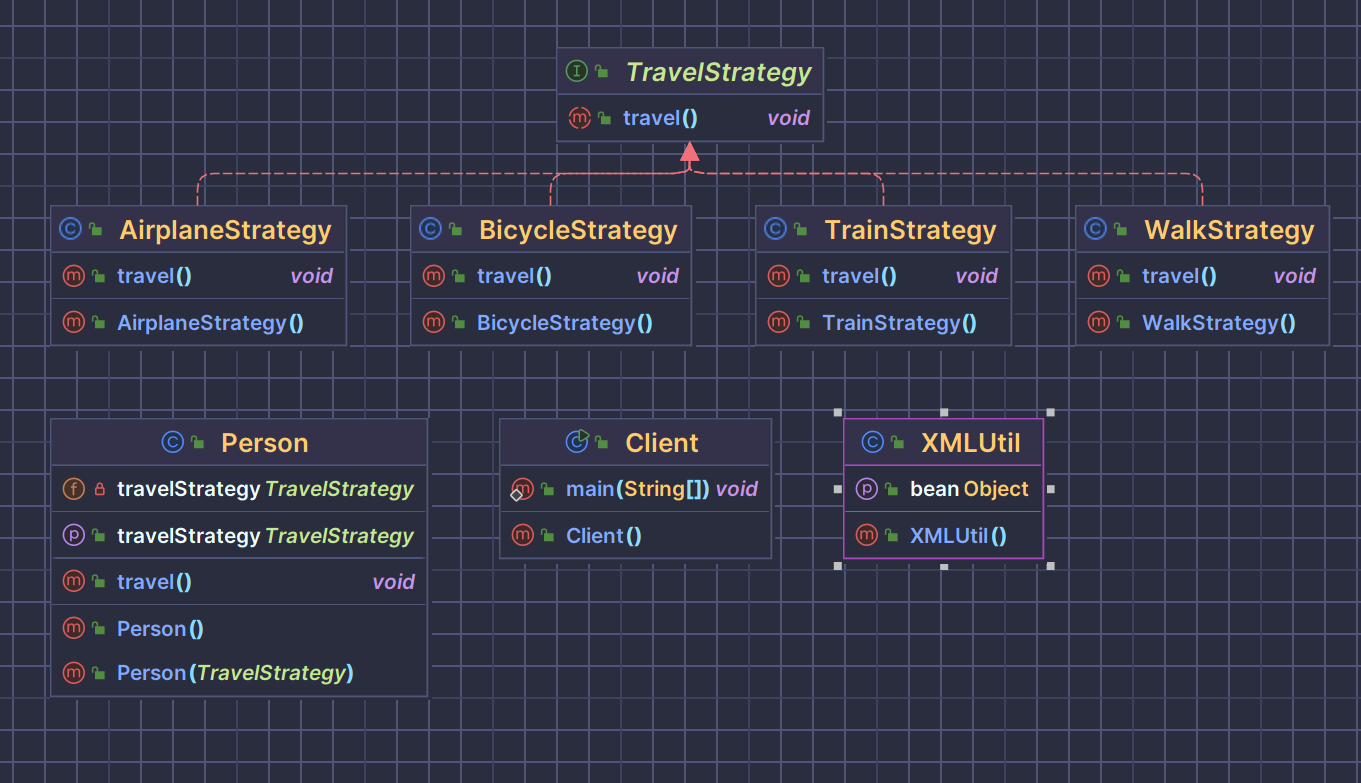
 如果在一个系统里面有许多类，他们之间的区别仅在于他们的行为，使用策略模式可以动态的让一个对象在许多行为中选择一种行为。

 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。

**3.2 代码实现**

**绘制类图：**

**抽象策略类 TravelStrategy:**

package edu.zhshio.strategy.lab.support;  
  
public interface TravelStrategy {  
 void travel();  
}

**具体策略类 AirplaneStrategy:**

package edu.zhshio.strategy.lab.strategies;  
  
import edu.zhshio.strategy.lab.support.TravelStrategy;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:53  
 \*/  
  
public class AirplaneStrategy implements TravelStrategy {  
 @Override  
 public void travel() {  
 System.out.println("飞机旅游, 就是一个快!");  
 }  
}

**具体策略类 TrainStrategy:**

package edu.zhshio.strategy.lab.strategies;  
  
import edu.zhshio.strategy.lab.support.TravelStrategy;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:53  
 \*/  
  
public class TrainStrategy implements TravelStrategy {  
 @Override  
 public void travel() {  
 System.out.println("火车旅游, 别忘了欣赏沿途的风景");  
 }  
}

**具体策略类 BicycleStrategy:**

package edu.zhshio.strategy.lab.strategies;  
  
import edu.zhshio.strategy.lab.support.TravelStrategy;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:54  
 \*/  
  
public class BicycleStrategy implements TravelStrategy {  
 @Override  
 public void travel() {  
 System.out.println("自行车游, 绿色环保");  
 }  
}

**具体策略类 WalkStrategy:**

package edu.zhshio.strategy.lab.strategies;  
  
import edu.zhshio.strategy.lab.support.TravelStrategy;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:57  
 \*/  
  
public class WalkStrategy implements TravelStrategy {  
 @Override  
 public void travel() {  
 System.out.println("徒步行走, 跨越山和大海!");  
 }  
}

**环境类 Person：**

package edu.zhshio.strategy.lab.travelers;  
  
import edu.zhshio.strategy.lab.support.TravelStrategy;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午7:58  
 \*/  
  
public class Person {  
 private TravelStrategy travelStrategy;  
  
 public void travel() {  
 System.out.println("我要去旅游啦");  
 travelStrategy.travel();  
 }  
  
 public TravelStrategy getTravelStrategy() {  
 return travelStrategy;  
 }  
  
 public void setTravelStrategy(TravelStrategy travelStrategy) {  
 this.travelStrategy = travelStrategy;  
 }  
  
 public Person(TravelStrategy travelStrategy) {  
 this.travelStrategy = travelStrategy;  
 }  
  
 public Person() {  
 }  
}

package edu.zhshio.strategy.lab.travelers;

**辅助类 XMLUtil：**

package edu.zhshio.strategy.lab.util;

/\*\*  
 \* @Auther: 张帅  
 \* @Date: 2024/3/7 - 03 - 07 - 11:15  
 \* @Description: edu.zhshio.builder.util  
 \* @version: 1.0  
 \*/  
  
import org.w3c.dom.Document;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
import org.xml.sax.SAXException;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.parsers.ParserConfigurationException;  
import java.io.File;  
import java.io.IOException;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/3/7 11:15  
 \*/  
  
public class XMLUtil {  
  
 public static Object getBean() {  
  
 DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();  
 try {  
 DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();  
 Document document = documentBuilder.parse(new File("src/main/java/edu/zhshio/strategy/lab/resource/beanconfig.xml"));  
  
 NodeList mealBuilderNames = document.getElementsByTagName("beanName");  
 Node mealBuilderName = mealBuilderNames.item(0).getFirstChild();  
 String mealBuilerNameVal = mealBuilderName.getNodeValue().trim();  
  
 Class<?> aClass = Class.forName(mealBuilerNameVal);  
 return aClass.newInstance();  
  
 } catch (ParserConfigurationException e) {  
 throw new RuntimeException(e);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 } catch (SAXException e) {  
 throw new RuntimeException(e);  
 } catch (ClassNotFoundException e) {  
 throw new RuntimeException(e);  
 } catch (InstantiationException e) {  
 throw new RuntimeException(e);  
 } catch (IllegalAccessException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

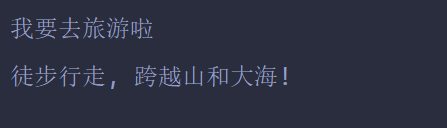
**配置文件 beanconfig.xml：**

<?xml version="1.0"?>  
<config>  
 <beanName>edu.zhshio.strategy.lab.strategies.WalkStrategy</beanName>  
</config>

**用户测试类 Client：**

package edu.zhshio.strategy.lab;  
  
import edu.zhshio.strategy.lab.support.TravelStrategy;  
import edu.zhshio.strategy.lab.travelers.Person;  
import edu.zhshio.strategy.lab.util.XMLUtil;  
  
/\*\*  
 \* @description:  
 \* @author: zs  
 \* @time: 2024/4/14 下午8:02  
 \*/  
  
public class Client {  
 public static void main(String[] args) {  
 TravelStrategy travelStrategy = (TravelStrategy) XMLUtil.getBean();  
 Person person = new Person(travelStrategy);  
 person.travel();  
 }  
}

**3.3 运行结果**



**四、心得体会**

行为型模式实验内容包括，迭代器模式、观察者模式和策略模式。因为设计模式较多非常容易记混，所以做实验之前我习惯先把课本的例题过一遍，再自己写代码，实验过程中没遇到问题，下面我还是分别对这三种设计模式说一下自己的理解。

首先是迭代器模式，顾名思义，它就是用来访问聚合对象的，对于遥控器和电视机，此时遥控器就是一个迭代器，而电视机就是它所对应的聚合对象，人们根本不需要关心这些频道是如何存储到电视里面的，他们只知道如何去用就可以了。为了拓展方便，需要定义一个抽象迭代器和一个抽象聚合类，在具体的聚合类中自然要调用迭代器中的方法，去进行电视台的迭代，这里可以按照课本上的方法，将具体的迭代器类作为具体聚合类的内部类，就是说我们不必要再去创建具体的迭代器类了，但全部功能都放在具体聚合类中有违开闭原则，因为迭代器也可能增加新的功能，所以也可以直接创一个具体迭代器类，在聚合类中直接传参并调用具体迭代器类即可。

观察者模式，股民观察股票价格动荡，从而做出反应。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。解决办法，在系统中创建一个触发链。一个目标可以有很多的观察者，在目标类中添加方法实现对观察者的增加和删除，同时目标类是要发出警告消息的，提供一个notify方法用于通知每一个观察者这里发出一个警告，所有的观察者都必须对此警告作为相应的反应，实现方式是遍历观察者集合，并调用其update方法，观察者类中的update方法就是观察者收到警告后做出的反应。

策略模式中包含策略类和环境类。具体策略类可以有很多，继承至抽象策略类，环境类其实是使用策略的角色，在环境类中维护一个对抽象策略类的引用实例，用于定义所采用的策略。在测试类中给环境类传入具体策略参数，继而在环境类中调用具体策略类的处理方法，最后在控制台输出结果。