

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： 实验四 设计模式综合实验

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 11日

目录

[1 实验目的和要求： 3](#_Toc131972365)

[1.1 目的： 3](#_Toc131972366)

[1.2 要求： 3](#_Toc131972367)

[2 实验平台: 4](#_Toc131972368)

[3 主要实验内容及结果: 5](#_Toc131972369)

[3.1 实验内容 5](#_Toc131972370)

[3.1.1 体播放器的实现 5](#_Toc131972371)

[3.1.2 会议管理系统的实现 11](#_Toc131972372)

[3.2 心得体会 15](#_Toc131972373)

# 实验目的和要求：

## 目的：

* 综合实例，熟练选用相应的多种设计模式。
* . 结合实例，使用 Java 实现多种设计模式的混用。
* 通过实验，理解不同设计模式的使用动机，掌握不同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以及设计模式的混合使用。

## 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

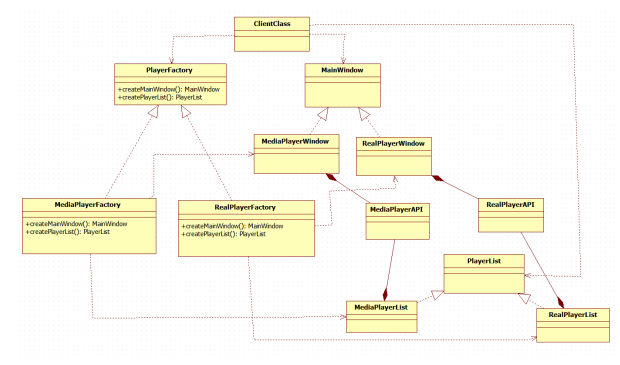
# 主要实验内容及结果:

## 实验内容

### 体播放器的实现

#### 案例背景：

Windows Media Player 和 RealPlayer 是两种常用的媒体播放器，它们的 API 结构和调用方法存在差别， 现在你的应用程序需要支持这两种播放器 API，并且将来可能还需要支持新的媒体播放器，根据给出的类 图，回答该应用程序使用了哪些设计模式，并编程实现这个应用程序



#### 实现步骤：

* 根据类图，回答这个应用程序使用了哪几种设计模式
* 根据类图，实现这个应用程序的具体代码以及工具类 XMLUtil，使应用程序能够使用两种播 放器的 API。
* 编译并运行代码，实现应用程序对两种播放器的调用。

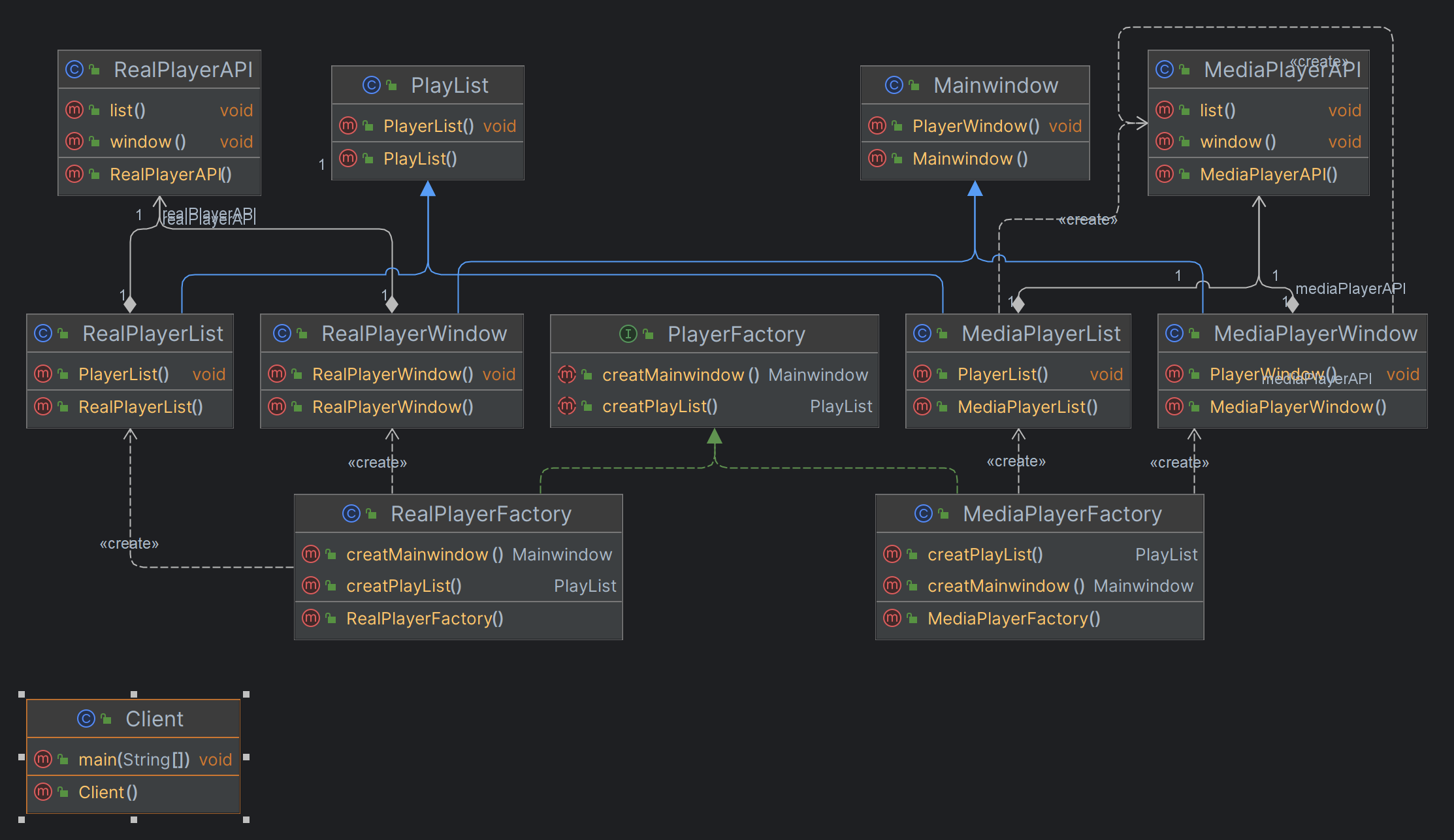
#### 案例总结：

* 设计模式的选用需要根据具体要求来进行
* 多种设计模式可以进行混合使用

#### 结果：

##### 设计模式选择：

* 工厂模式
* 适配器模式



##### 代码实现

* MediaPlayerAPI类

public class MediaPlayerAPI {   
   
 public void window() {   
 System.out.println("mediaplayerwindowapi");   
 }   
   
 public void list() {   
 System.out.println("mediaplayerlistapi");   
 }   
}

* RealPlayerAPI类

public class RealPlayerAPI {   
 public void window() {   
 System.out.println("realwindow");   
 }   
 public void list() {   
 System.out.println("reallist");   
 }   
}

* PlayerFactory 接口

public interface PlayerFactory {   
 public Mainwindow creatMainwindow();   
 public PlayList creatPlayList();   
}

* MediaPlayerFactory类

public class MediaPlayerFactory implements PlayerFactory {   
 @Override   
 public Mainwindow creatMainwindow() {   
 System.out.println("MediaPlayerFactorywuindow");   
 return new MediaPlayerWindow();   
 }   
   
 @Override   
 public PlayList creatPlayList() {   
 System.out.println( "MediaPlayerFactorylist");   
 return new MediaPlayerList();   
 }   
}

* RealPlayerFactory类

public class RealPlayerFactory implements PlayerFactory{   
 @Override   
 public Mainwindow creatMainwindow() {   
 System.out.println("RealPlayerFactorywindow");   
 return new RealPlayerWindow();   
 }   
 @Override   
 public PlayList creatPlayList() {   
 System.out.println( "RealPlayerFactorylist");   
 return new RealPlayerList();   
 }   
}

-PlayList 类

public class PlayList {   
 public void PlayerList() {   
 }   
}

* MediaPlayerList类

public class MediaPlayerList extends PlayList{   
 private MediaPlayerAPI mediaPlayerAPI=new MediaPlayerAPI();   
 public void PlayerList() {   
 System.out.println("MediaPlayerList");   
 mediaPlayerAPI.list();   
 }   
}

* RealPlayerList类

public class RealPlayerList extends PlayList {   
 private RealPlayerAPI realPlayerAPI;   
 public void PlayerList() {   
 System.out.println("RealPlayerList");   
 realPlayerAPI.list();   
 }   
}

* Mainwindow类

public class Mainwindow {   
 public void PlayerWindow() {   
 }   
   
}

* MediaPlayerWindow 类

public class MediaPlayerWindow extends Mainwindow {   
 private MediaPlayerAPI mediaPlayerAPI =new MediaPlayerAPI();   
   
 @Override   
 public void PlayerWindow() {   
 System.out.println("MediaPlayerWindow");   
 mediaPlayerAPI.window();   
 }   
}

* RealPlayerWindow类

public class RealPlayerWindow extends Mainwindow{   
 private RealPlayerAPI realPlayerAPI ;   
 public void RealPlayerWindow() {   
 System.out.println("RealPlayerWindow");   
 realPlayerAPI.window();   
 }   
}

<?xml version="1.0"?>   
<config>   
 <className>MediaPlayerFactory</className>   
</config>

* Xmlutil类

ublic 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/实验四一/config.xml"));   
 NodeList nodeList=null;   
 Node classNode=null;   
 String cNamebefore="实验四一.Factory.";   
 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;   
 }   
   
 }   
 }

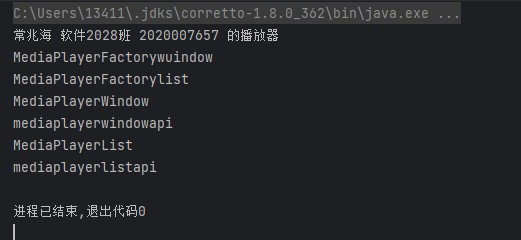
* Client类

public class Client {   
 public static void main(String[] args) {   
   
 PlayerFactory playerFactory = (PlayerFactory) Xmlutil.getbBean();   
   
 System.out.println("常兆海 软件2028班 2020007657 的播放器");   
 Mainwindow window;   
 PlayList list;   
 window = playerFactory.creatMainwindow();   
 list=playerFactory.creatPlayList();   
 window.PlayerWindow();   
 list.PlayerList();   
 }   
}

##### 运行结果

常兆海 软件2028班 2020007657 的播放器  
MediaPlayerFactorywuindow  
MediaPlayerFactorylist  
MediaPlayerWindow  
mediaplayerwindowapi  
MediaPlayerList  
mediaplayerlistapi

进程已结束,退出代码0



### 会议管理系统的实现

#### 案例背景：

#### 实现步骤：

#### 案例总结：

#### 结果：

##### 类图

##### 代码实现

* MySubject抽象类

public abstract class MySubject {   
 protected ArrayList observes=new ArrayList();   
 //注册   
 public void attach(User observer){   
 observes.add(observer);   
 }   
 public void detach(User observer){   
 observes.remove(observer);   
 }   
 public abstract void meeting();   
}

* User抽象类

public abstract class User {   
   
 public void add(User user){};   
 public void remove(User user){};   
 public User getchild(int i){return null;};   
 public abstract void action();   
}

* MeetingList类

public class MeetingList extends MySubject {   
 @Override   
 public void meeting() {   
   
 for (Object o : observes   
 ) { System.out.println("软件2028班 常兆海202007657 的会议1安排");   
 ((User) o).action();   
 }   
 }   
   
}

* Employee 类

public class Employee extends User{   
   
   
 @Override   
 public void action() {   
 System.out.println("向某个或者某些员工（Employee）发送会议通知，");   
 }   
}

* Department类

public class Department extends User{   
 private ArrayList list=new ArrayList();   
 @Override   
 public void add(User user) {   
 list.add(user);}   
   
 @Override   
 public void remove(User user) {   
 list.remove(user);   
 }   
   
 @Override   
 public User getchild(int i) {   
 return (User) list.get(i);   
 }   
   
 @Override   
 public void action() {   
 System.out.println("某些部门（Department）发送会议通知，");   
 for (Object o : list   
 ) {   
 ((User) o).action();   
 }   
 }   
}

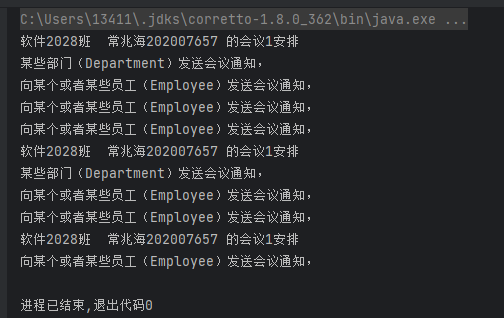
* Client类

public class Client {   
   
 public static void main(String[] args) {   
 MySubject meetingList = new MeetingList();   
   
 User user1, user2, user3, user4, user5, users1, users2;   
 users1 = new Department();   
 users2 = new Department();   
   
 user1 = new Employee();   
 user2 = new Employee();   
 user3 = new Employee();   
 user4 = new Employee();   
 user5 = new Employee();   
   
 users1.add(user1);   
 users1.add(user2);   
 users1.add(user3);   
 users2.add(user4);   
 users2.add(user5);   
   
 meetingList.attach(users1);//向部门1发送通知   
   
 meetingList.attach(users2);//向部门2发送通知   
   
 meetingList.attach(user3);//向员工3发送通知   
   
 meetingList.meeting();//通过观察者模式实现：如果员工或者部门希望能够收到会议通知，必须先注册到一个会议列表（MeetingList）中， 在发送通知时，系统将遍历会议列表，逐个将会议通知发送给注册用户（User）   
   
 }   
}

##### 运行结果

软件2028班 常兆海202007657 的会议1安排  
某些部门（Department）发送会议通知，  
向某个或者某些员工（Employee）发送会议通知，  
向某个或者某些员工（Employee）发送会议通知，  
向某个或者某些员工（Employee）发送会议通知，  
软件2028班 常兆海202007657 的会议1安排  
某些部门（Department）发送会议通知，  
向某个或者某些员工（Employee）发送会议通知，  
向某个或者某些员工（Employee）发送会议通知，  
软件2028班 常兆海202007657 的会议1安排  
向某个或者某些员工（Employee）发送会议通知，

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，更好的理解了各种设计模式，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。