# 综合性实验报告

华南农业大学信息学院

软件体系结构综合性实验报告

郑丁豪

201725050227

2020年1月完稿

目 录

目录

[综合性实验报告 1](#_Toc29481713)

[模式一：单例模式 4](#_Toc29481714)

[(1) 解决问题类型 4](#_Toc29481715)

[(2) 解决设计思路 4](#_Toc29481716)

[(3) 具体待解决的问题 4](#_Toc29481717)

[(4) UML设计 4](#_Toc29481718)

[(5) 核心代码 4](#_Toc29481719)

[(6) 实现效果 6](#_Toc29481720)

[(7) 模式的优点 6](#_Toc29481721)

[(8) 模式的不足 6](#_Toc29481722)

[模式二：多例模式 7](#_Toc29481723)

[(1) 解决问题类型 7](#_Toc29481724)

[(2) 解决设计思路 7](#_Toc29481725)

[(3) 具体待解决的问题 7](#_Toc29481726)

[(4) UML设计 7](#_Toc29481727)

[(5) 核心代码 7](#_Toc29481728)

[(6) 实现效果 11](#_Toc29481729)

[(7) 模式的优点 11](#_Toc29481730)

[(8) 模式的不足 11](#_Toc29481731)

[模式三：简单工厂模式 12](#_Toc29481732)

[(1) 解决问题类型 12](#_Toc29481733)

[(2) 解决设计思路 12](#_Toc29481734)

[(3) 具体待解决的问题 12](#_Toc29481735)

[(4) UML设计 12](#_Toc29481736)

[(5) 核心代码 12](#_Toc29481737)

[(6) 实现效果 14](#_Toc29481738)

[(7) 模式的优点 15](#_Toc29481739)

[(8) 模式的不足 15](#_Toc29481740)

[模式四：工厂模式 16](#_Toc29481741)

[(1) 解决问题类型 16](#_Toc29481742)

[(2) 解决设计思路 16](#_Toc29481743)

[(3) 具体待解决的问题 16](#_Toc29481744)

[(4) UML设计 16](#_Toc29481745)

[(5) 核心代码 16](#_Toc29481746)

[(6) 实现效果 18](#_Toc29481747)

[(7) 模式的优点 18](#_Toc29481748)

[(8) 模式的不足 18](#_Toc29481749)

[模式五：抽象工厂模式 19](#_Toc29481750)

[(1) 解决问题类型 19](#_Toc29481751)

[(2) 解决设计思路 19](#_Toc29481752)

[(3) 具体待解决的问题 19](#_Toc29481753)

[(4) UML设计 19](#_Toc29481754)

[(5) 核心代码 19](#_Toc29481755)

[(6) 实现效果 23](#_Toc29481756)

[(7) 模式的优点 23](#_Toc29481757)

[(8) 模式的不足 23](#_Toc29481758)

[模式六：对象适配器模式 24](#_Toc29481759)

[(1) 解决问题类型 24](#_Toc29481760)

[(2) 解决设计思路 24](#_Toc29481761)

[(3) 具体待解决的问题 24](#_Toc29481762)

[(4) UML设计 24](#_Toc29481763)

[(5) 核心代码 24](#_Toc29481764)

[(6) 实现效果 27](#_Toc29481765)

[(7) 模式的优点 31](#_Toc29481766)

[(8) 模式的不足 31](#_Toc29481767)

[模式七：类适配器模式 32](#_Toc29481768)

[(1) 解决问题类型 32](#_Toc29481769)

[(2) 解决设计思路 32](#_Toc29481770)

[(3) 具体待解决的问题 32](#_Toc29481771)

[(4) UML设计 32](#_Toc29481772)

[(5) 核心代码 32](#_Toc29481773)

[(6) 实现效果 33](#_Toc29481774)

[(7) 模式的优点 33](#_Toc29481775)

[(8) 模式的不足 33](#_Toc29481776)

[模式八：安全性组合模式 34](#_Toc29481777)

[(1) 解决问题类型 34](#_Toc29481778)

[(2) 解决设计思路 34](#_Toc29481779)

[(3) 具体待解决的问题 34](#_Toc29481780)

[(4) UML设计 34](#_Toc29481781)

[(5) 核心代码 34](#_Toc29481782)

[(6) 实现效果 37](#_Toc29481783)

[(7) 模式的优点 38](#_Toc29481784)

[(8) 模式的不足 38](#_Toc29481785)

[模式九：一致性组合模式 39](#_Toc29481786)

[(1) 解决问题类型 39](#_Toc29481787)

[(2) 解决设计思路 39](#_Toc29481788)

[(3) 具体待解决的问题 39](#_Toc29481789)

[(4) UML设计 39](#_Toc29481790)

[(5) 核心代码 39](#_Toc29481791)

[(6) 实现效果 42](#_Toc29481792)

[(7) 模式的优点 42](#_Toc29481793)

[(8) 模式的不足 42](#_Toc29481794)

[综合性实验的心得体会 43](#_Toc29481795)

## 模式一：单例模式

### (1) 解决问题类型

当前某个类只能有一个实例。

### (2) 解决设计思路

由于国家当前只能有一个主席，所以主席这个类必须使用单例模式。

### (3) 具体待解决的问题

建立一个图形界面，在用户输入主席的名称点击确定后，在label中显示创建的对象名称和对象。再次或者多次输入主席名称，点击创建，在前面显示创建结果的下一行显示本次创建对象的名称和对象。

### (4) UML设计

利用UML描述你的设计



### (5) 核心代码

分析核心的代码

Chairman.java

**package** cn.edu.scau.cmi.zhengdinghao.singleton;

**public** **class** Chairman {

**private** String name = "Nobody";

**public** **static** Chairman *chairman* = **new** Chairman();

**private** Chairman() {

}

**public** **static** Chairman setChairman(String name) {

*chairman*.name = name;

**return** *chairman*;

}

**public** **static** Chairman getChairman() {

**return** *chairman*;

}

**public** String getName() {

**return** name;

}

}

Singleton.java

**package** cn.edu.scau.cmi.zhengdinghao.singleton;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JTextField;

**public** **class** Singleton **extends** JFrame{

JTextField text;

JButton button;

JLabel label;

**public** **void** run() {

setBounds(550,240,500,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("单例模式");

setLayout(**new** FlowLayout(5));

getContentPane().add(**new** JLabel("请输入主席名称"));

text = **new** JTextField(10);

getContentPane().add(text);

button = **new** JButton("确定");

getContentPane().add(button);

button.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Chairman chairman = Chairman.*setChairman*(text.getText());

label = **new** JLabel("\n"+chairman.getName()+chairman);

add(label);

setVisible(**true**);

}

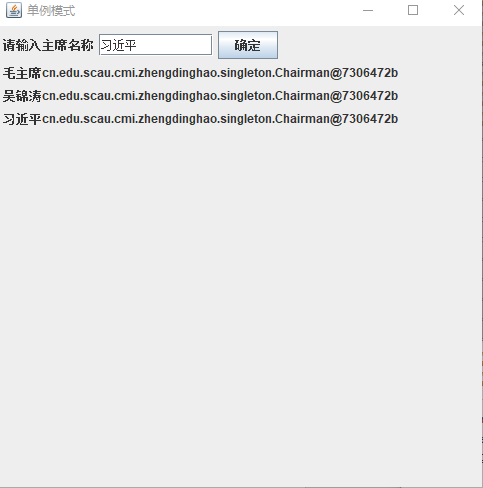
});

setVisible(**true**);

}

}

### (6) 实现效果



### (7) 模式的优点

简单易实现

### (8) 模式的不足

复杂问题时难以满足需求

## 模式二：多例模式

### (1) 解决问题类型

当软件需要固定数量的实例时。

### (2) 解决设计思路

用多例模式生成固定数量的实例。

### (3) 具体待解决的问题

使用单例模式的思想实现多例模式，确保系统中某个类的对象只能存在有限个，如中华人民共和国十大元帅(Marshal)，设计并实现。界面的基本要求和前面一致。

### (4) UML设计



### (5) 核心代码

分析核心的代码

Marshal.java

**package** cn.edu.scau.cmi.zhengdinghao.multition;

**public** **class** Marshal {

**private** String name;

**public** **static** Marshal *m1* = **new** Marshal();

**public** **static** Marshal *m2* = **new** Marshal();

**public** **static** Marshal *m3* = **new** Marshal();

**public** **static** Marshal *m4* = **new** Marshal();

**public** **static** Marshal *m5* = **new** Marshal();

**public** **static** Marshal *m6* = **new** Marshal();

**public** **static** Marshal *m7* = **new** Marshal();

**public** **static** Marshal *m8* = **new** Marshal();

**public** **static** Marshal *m9* = **new** Marshal();

**public** **static** Marshal *m10* = **new** Marshal();

**private** Marshal() {

}

**public** **static** Marshal getMarshal(**int** num) {

**switch**(num) {

**case** 1:

**return** *m1*;

**case** 2:

**return** *m2*;

**case** 3:

**return** *m3*;

**case** 4:

**return** *m4*;

**case** 5:

**return** *m5*;

**case** 6:

**return** *m6*;

**case** 7:

**return** *m7*;

**case** 8:

**return** *m8*;

**case** 9:

**return** *m9*;

**case** 10:

**return** *m10*;

**default**:

**return** **null**;

}

}

**public** **static** Marshal setMarshal(**int** num,String name) {

**switch**(num) {

**case** 1:

*m1*.name = name;

**return** *m1*;

**case** 2:

*m2*.name = name;

**return** *m2*;

**case** 3:

*m3*.name = name;

**return** *m3*;

**case** 4:

*m4*.name = name;

**return** *m4*;

**case** 5:

*m5*.name = name;

**return** *m5*;

**case** 6:

*m6*.name = name;

**return** *m6*;

**case** 7:

*m7*.name = name;

**return** *m7*;

**case** 8:

*m8*.name = name;

**return** *m8*;

**case** 9:

*m9*.name = name;

**return** *m9*;

**case** 10:

*m10*.name = name;

**return** *m10*;

**default**:

**return** **null**;

}

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

}

Multition.java

**package** cn.edu.scau.cmi.zhengdinghao.multition;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JTextField;

**public** **class** Multition **extends** JFrame {

JTextField text;

JButton button;

JLabel label;

**public** **void** run() {

setBounds(550,240,500,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("多例模式");

setLayout(**new** FlowLayout(5));

getContentPane().add(**new** JLabel("请输入元帅序号与名字用空格隔开"));

text = **new** JTextField(10);

getContentPane().add(text);

button = **new** JButton("确定");

getContentPane().add(button);

button.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Marshal marshal = Marshal.*setMarshal*(Integer.*parseInt*(text.getText().split("\\s")[0]),text.getText().split("\\s")[1]);

label = **new** JLabel("\n"+marshal.getName()+marshal);

add(label);

setVisible(**true**);

}

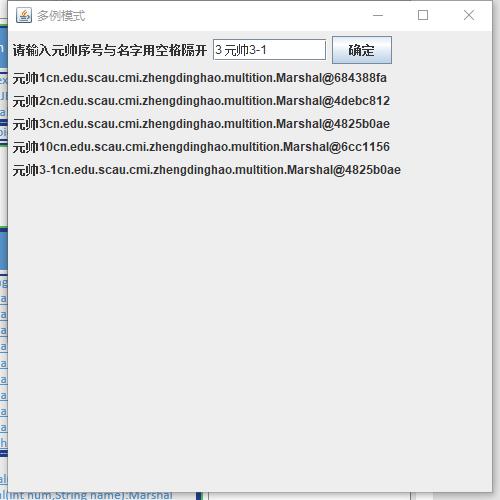
});

setVisible(**true**);

}

}

### (6) 实现效果



### (7) 模式的优点

与单例模式相同

### (8) 模式的不足

与单例模式相同

## 模式三：简单工厂模式

### (1) 解决问题类型

通过一个工厂来创建多种需要的实例

### (2) 解决设计思路

通过简单工厂模式来解决问题

### (3) 具体待解决的问题

肉厂(MeatFactory)可以生产猪肉(Pork)，牛肉(Beef)，鸡肉(Chicken)等肉类(Meat)，使用简单工厂方法实现肉类的生产。

在界面上选择某一种肉类，在输出框中显示获得的对象的行为。

### (4) UML设计



### (5) 核心代码

分析核心的代码

Factory.java

**package** cn.edu.scau.cmi.zhengdinghao.simpleFactory;

**public** **class** Factory {

**public** **static** Meat getMeat(String string) {

**if**(string.equalsIgnoreCase("pork")) {

**return** **new** Pork();

}**else** **if**(string.equalsIgnoreCase("beef")) {

**return** **new** Beef();

}**else** **if**(string.equalsIgnoreCase("chicken")) {

**return** **new** Chicken();

}**else** {

System.***out***.println("no such meat.");

**return** **null**;

}

}

}

SimpleFactory.java

**package** cn.edu.scau.cmi.zhengdinghao.simpleFactory;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**public** **class** SimpleFactory **extends** JFrame {

JButton pork,beef,chicken;

JLabel label;

**public** **void** run() {

setBounds(550,240,500,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("简单工厂模式");

setLayout(**new** FlowLayout(5));

pork = **new** JButton("猪肉");

beef = **new** JButton("牛肉");

chicken = **new** JButton("鸡肉");

getContentPane().add(pork);

getContentPane().add(beef);

getContentPane().add(chicken);

pork.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Meat meat = Factory.*getMeat*("Pork");

label = **new** JLabel(meat.type());

add(label);

setVisible(**true**);

}

});

beef.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Meat meat = Factory.*getMeat*("Beef");

label = **new** JLabel(meat.type());

add(label);

setVisible(**true**);

}

});

chicken.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Meat meat = Factory.*getMeat*("Chicken");

label = **new** JLabel(meat.type());

add(label);

setVisible(**true**);

}

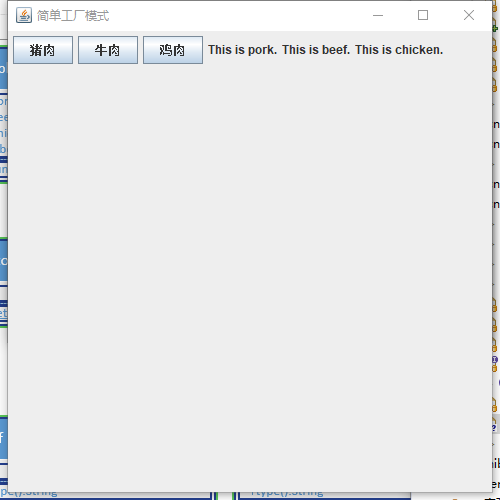
});

setVisible(**true**);

}

}

### (6) 实现效果



### (7) 模式的优点

实现简单

### (8) 模式的不足

不易扩展，耦合度高

## 模式四：工厂模式

### (1) 解决问题类型

不同种类的实例用不同的工厂生产

### (2) 解决设计思路

使用工厂模式解决问题

### (3) 具体待解决的问题

不同类型的肉厂生产不同的肉类，PorkMeatFactory生产猪肉，BeefMeatFactory生产牛肉，ChickenMeatFactory生产鸡肉。使用工厂模式实现这一功能。

### (4) UML设计



### (5) 核心代码

MeatFactory.java

**package** cn.edu.scau.cmi.zhengdinghao.factory;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**public** **class** MeatFactory **extends** JFrame{

JButton pork,beef,chicken;

JLabel label;

Factory pFactory = **new** PorkFactory();

Factory bFactory = **new** BeefFactory();

Factory cFactory = **new** ChickenFactory();

**public** **void** run() {

setBounds(550,240,500,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("工厂模式");

setLayout(**new** FlowLayout(5));

pork = **new** JButton("猪肉");

beef = **new** JButton("牛肉");

chicken = **new** JButton("鸡肉");

getContentPane().add(pork);

getContentPane().add(beef);

getContentPane().add(chicken);

pork.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Meat pork = pFactory.getMeat();

label = **new** JLabel(pork.type());

add(label);

setVisible(**true**);

}

});

beef.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Meat beef = bFactory.getMeat();

label = **new** JLabel(beef.type());

add(label);

setVisible(**true**);

}

});

chicken.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Meat chicken = cFactory.getMeat();

label = **new** JLabel(chicken.type());

add(label);

setVisible(**true**);

}

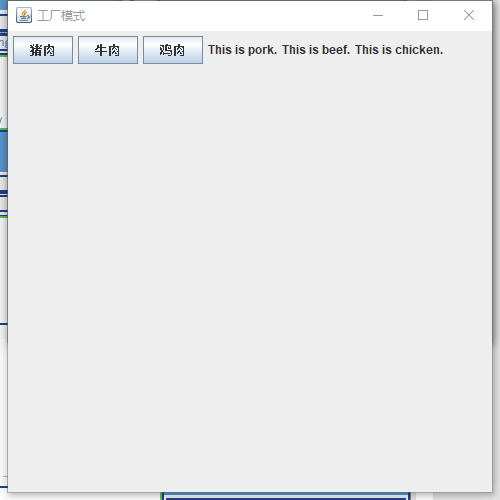
});

setVisible(**true**);

}

}

### (6) 实现效果



### (7) 模式的优点

耦合度低，易于扩展

### (8) 模式的不足

太多种实例时软件会使工厂太多显得臃肿

## 模式五：抽象工厂模式

### (1) 解决问题类型

每个工厂生产不同系列的实例

### (2) 解决设计思路

使用抽象工厂模式

### (3) 具体待解决的问题

温氏(Wens)集团、天邦(Tianbang)股份、傲农(Aonong)生物是我国三家大型的养殖企业，可以养猪，养牛，养鸡。利用抽象工厂模式实现用户购买不同品牌(Brand)的不同肉类(Meat)产品。

### (4) UML设计



### (5) 核心代码

Factory.java

**package** cn.edu.scau.cmi.zhengdinghao.abstractFactory;

**public** **interface** Factory {

**public** Pork getPork();

**public** Beef getBeef();

**public** Chicken getChicken();

}

AbstractFactory.java

**package** cn.edu.scau.cmi.zhengdinghao.abstractFactory;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**public** **class** AbstractFactory **extends** JFrame{

Factory wFactory = **new** WensFactory();

Factory tFactory = **new** TianbangFactory();

Factory aFactory = **new** AonongFactory();

JButton wPork,wBeef,wChicken,tPork,tBeef,tChicken,aPork,aBeef,aChicken;

JLabel label;

**public** **void** run() {

setBounds(550,240,500,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("工厂模式");

setLayout(**new** FlowLayout(5));

wPork = **new** JButton("温氏猪肉");

wBeef = **new** JButton("温氏牛肉");

wChicken = **new** JButton("温氏鸡肉");

tPork = **new** JButton("天邦猪肉");

tBeef = **new** JButton("天邦牛肉");

tChicken = **new** JButton("天邦鸡肉");

aPork = **new** JButton("傲农猪肉");

aBeef = **new** JButton("傲农牛肉");

aChicken = **new** JButton("傲农鸡肉");

getContentPane().add(wPork);

getContentPane().add(wBeef);

getContentPane().add(wChicken);

getContentPane().add(tPork);

getContentPane().add(tBeef);

getContentPane().add(tChicken);

getContentPane().add(aPork);

getContentPane().add(aBeef);

getContentPane().add(aChicken);

wPork.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Pork wp = wFactory.getPork();

label = **new** JLabel(wp.typePork());

add(label);

setVisible(**true**);

}

});

wBeef.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Beef wb = wFactory.getBeef();

label = **new** JLabel(wb.typeBeef());

add(label);

setVisible(**true**);

}

});

wChicken.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Chicken wc = wFactory.getChicken();

label = **new** JLabel(wc.typeChicken());

add(label);

setVisible(**true**);

}

});

tPork.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Pork tp = tFactory.getPork();

label = **new** JLabel(tp.typePork());

add(label);

setVisible(**true**);

}

});

tBeef.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Beef tb = tFactory.getBeef();

label = **new** JLabel(tb.typeBeef());

add(label);

setVisible(**true**);

}

});

tChicken.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Chicken tc = tFactory.getChicken();

label = **new** JLabel(tc.typeChicken());

add(label);

setVisible(**true**);

}

});

aPork.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Pork ap = aFactory.getPork();

label = **new** JLabel(ap.typePork());

add(label);

setVisible(**true**);

}

});

aBeef.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Beef ab = aFactory.getBeef();

label = **new** JLabel(ab.typeBeef());

add(label);

setVisible(**true**);

}

});

aChicken.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

Chicken ac = aFactory.getChicken();

label = **new** JLabel(ac.typeChicken());

add(label);

setVisible(**true**);

}

});

setVisible(**true**);

}

}

### (6) 实现效果



### (7) 模式的优点

在工厂模式低耦合的基础上使得实例种类多的情况下工厂也可以依旧保持一个相对较少的数量不会使得臃肿。

### (8) 模式的不足

实现较为复杂，逻辑结构也容易混乱。某些情况下会导致耦合度高。

## 模式六：对象适配器模式

### (1) 解决问题类型

两个软件模块接口不一致时采用一个适配器类来进行适配

### (2) 解决设计思路

用对象适配器来适配两个模块之间的接口

### (3) 具体待解决的问题

类SortUtil可以把整数排序List sortInt(List<int> intList)，这个方法请自己实现；学生有学号(int类型)，姓名(String类型)这两个域。在客户端输入学生的学号和姓名后，及时显示按照学号从小到大排序了的学生名单。排序功能使用类适配器和对象适配器两种方式实现。

### (4) UML设计



### (5) 核心代码

分析核心的代码

Adapter.java

**package** cn.edu.scau.cmi.zhengdinghao.objectAdapter;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.List;

**public** **class** Adapter **implements** StudentSortUtil{

SortUtil sortUtil;

@Override

**public** **void** sortInt(List<Integer> intList) {

sortUtil.sortInt(intList);

}

**public** **void** sortStudent(List<Student> listStudent) {

Collections.*sort*(listStudent, **new** Comparator<Student>() {

@Override

**public** **int** compare(Student s1,Student s2) {

**int** x = s1.getId()-s2.getId();

**if**(x > 0){

**return** 1;

}**else** **if** (x < 0){

**return** -1;

}

**return** 0;

}

});

}

}

ObjectAdapter.java

**package** cn.edu.scau.cmi.zhengdinghao.objectAdapter;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JTextField;

**public** **class** ObjectAdapter **extends** JFrame{

JTextField text1;

JTextField text2;

JButton button;

JLabel label;

List<Student> listStudent;

Student student;

Adapter adapter;

**public** **void** run() {

setBounds(550,240,550,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("对象适配器");

setLayout(**new** FlowLayout(5));

listStudent = **new** ArrayList<Student>();

adapter = **new** Adapter();

getContentPane().add(**new** JLabel("请分别输入需要排序的学生学号和姓名"));

text1 = **new** JTextField(10);

getContentPane().add(text1);

text2 = **new** JTextField(10);

getContentPane().add(text2);

button = **new** JButton("确定");

getContentPane().add(button);

label = **new** JLabel();

getContentPane().add(label);

button.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

student = **new** Student(Integer.*parseInt*(text1.getText()),text2.getText());

listStudent.add(student);

adapter.sortStudent(listStudent);

String string = "<html>";

**for**(Student student:listStudent) {

string+=("<br>"+student.getName());

}

string+="</html>";

label.setText(string);

setVisible(**true**);

}

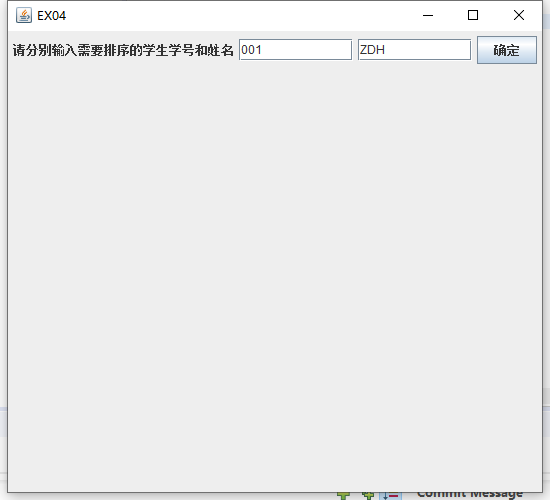
});

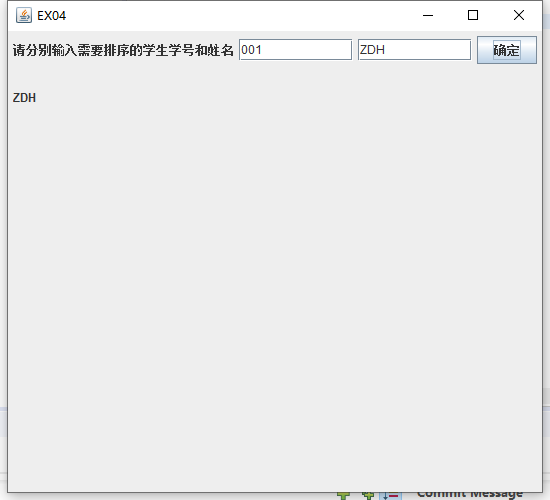
setVisible(**true**);

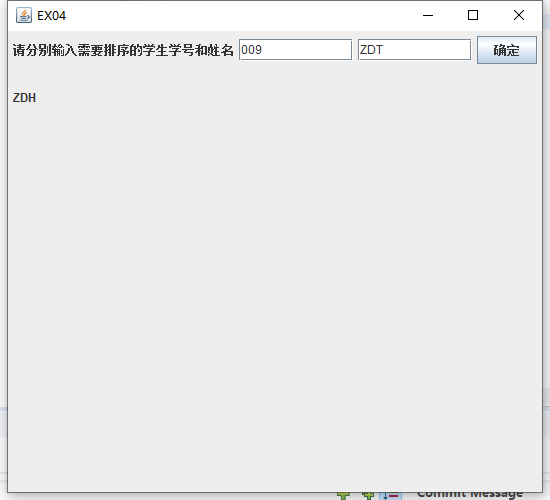
}

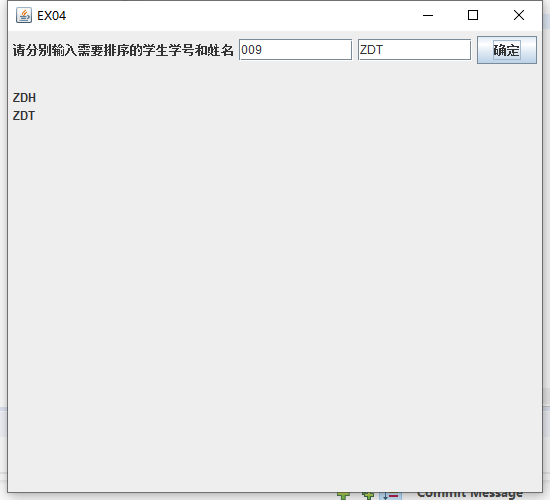
}

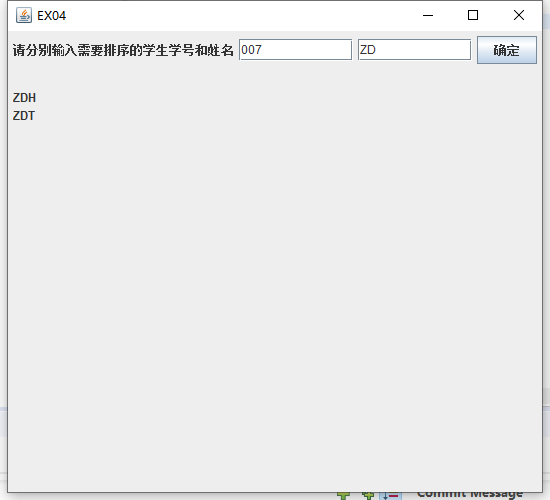
### (6) 实现效果

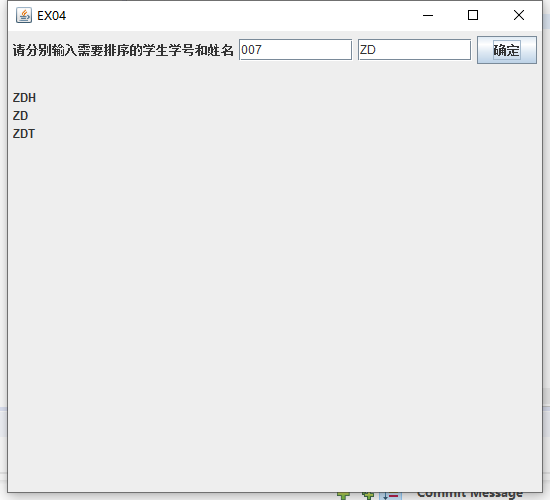


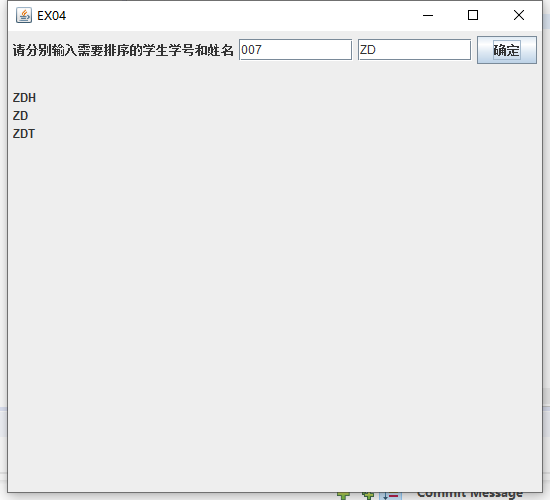


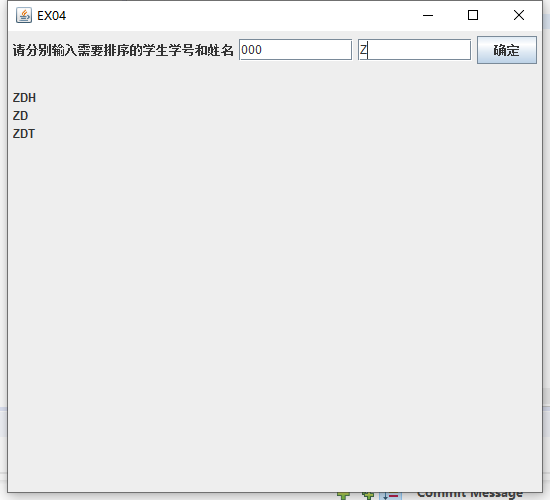












### (7) 模式的优点

能够适配两个接口不兼容的模块

### (8) 模式的不足

耦合度高，想替换适配器类中的方法时较为麻烦

## 模式七：类适配器模式

### (1) 解决问题类型

两个软件模块接口不一致时采用一个适配器类来进行适配

### (2) 解决设计思路

用类适配器来适配两个模块之间的接口

### (3) 具体待解决的问题

类SortUtil可以把整数排序List sortInt(List<int> intList)，这个方法请自己实现；学生有学号(int类型)，姓名(String类型)这两个域。在客户端输入学生的学号和姓名后，及时显示按照学号从小到大排序了的学生名单。排序功能使用类适配器和对象适配器两种方式实现。

### (4) UML设计



### (5) 核心代码

Adapter.java

**package** cn.edu.scau.cmi.zhengdinghao.classAdapter;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.List;

**public** **class** Adapter **extends** SortUtil **implements** StudentSortUtil{

@Override

**public** **void** sortStudent(List<Student> listStudent) {

Collections.*sort*(listStudent, **new** Comparator<Student>() {

@Override

**public** **int** compare(Student s1,Student s2) {

**int** x = s1.getId()-s2.getId();

**if**(x > 0){

**return** 1;

}**else** **if** (x < 0){

**return** -1;

}

**return** 0;

}

});

}

}

### (6) 实现效果

与对象适配器相同

### (7) 模式的优点

能够适配两个接口不兼容的模块

### (8) 模式的不足

无法多继承

## 模式八：安全性组合模式

### (1) 解决问题类型

将部分和整体不加区别的对待

### (2) 解决设计思路

使用安全性组合模式处理

### (3) 具体待解决的问题

个人(Person)与团队(Team)可以形成一个组织(Organization)：组织有两种：个人组织和团队组织，多个个人可以组合成一个团队，不同的个人与团队可以组合成一个更大的团队。创建个人，创建团队，每创建一个团队，每次团队成员有变化，就递归输出所有的成员。

### (4) UML设计



### (5) 核心代码

分析核心的代码

Person.java

**package** cn.edu.scau.cmi.zhengdinghao.composite;

**public** **class** Person **extends** Organization{

**public** String name;

**public** String getName() {

**return** name;

}

**public** Person(String name) {

**this**.name = name;

}

}

Teams.java

**package** cn.edu.scau.cmi.zhengdinghao.composite;

**import** java.util.ArrayList;

**public** **class** Team **extends** Organization{

**public** String name;

ArrayList<Organization> organizations;

**public** Team(String name) {

**this**.name = name;

organizations = **new** ArrayList<>();

}

**public** **void** addChild(Organization organization) {

organizations.add(organization);

}

**public** **void** removeChild(Organization organization) {

organizations.remove(organization);

}

**public** String getName() {

**int** len = organizations.size();

String allName = name;

**for**(**int** i = 0;i < len;i++) {

Organization organization = organizations.get(i);

allName +=("<br>" + organization.getName());

}

**return** allName;

}

}

Composite.java

**package** cn.edu.scau.cmi.zhengdinghao.composite;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** java.util.ArrayList;

**import** javax.swing.ButtonGroup;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JRadioButton;

**import** javax.swing.JTextField;

**public** **class** Composite **extends** JFrame {

ArrayList<Organization> organizations = **new** ArrayList<>();

ArrayList<Team> teams = **new** ArrayList<>();

Organization o = **null**;

JTextField name;

JButton button;

JRadioButton teamButton,personButton;

ButtonGroup group = **new** ButtonGroup();

JLabel label;

**public** **void** run() {

setBounds(550,240,500,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("安全组合模式");

setLayout(**new** FlowLayout(5));

getContentPane().add(**new** JLabel("请输入创建对象的名字以及所属组织名用空格隔开(无所属则无需输入)"));

name = **new** JTextField(10);

getContentPane().add(name);

button = **new** JButton("确定");

getContentPane().add(button);

teamButton = **new** JRadioButton("团队组织");

personButton = **new** JRadioButton("个人组织");

teamButton.setSelected(**true**);

group.add(teamButton);

group.add(personButton);

getContentPane().add(teamButton);

getContentPane().add(personButton);

getContentPane().add(**new** JLabel("所属组织的成员有:"));

label = **new** JLabel();

add(label);

button.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

**if**(teamButton.isSelected()) {

o = **new** Team(name.getText().split("\\s")[0]);

teams.add((Team)o);

organizations.add(o);

**if**(name.getText().split("\\s").length != 1) {

**for**(Team t:teams) {

**if**(name.getText().split("\\s")[1].equals(t.name)) {

t.addChild(o);

label.setText("<html>"+t.getName()+"</html>");

setVisible(**true**);

}

}

}

}**else** {

o = **new** Person(name.getText().split("\\s")[0]);

organizations.add(o);

**if**(name.getText().split("\\s").length != 1) {

**for**(Team t:teams) {

**if**(name.getText().split("\\s")[1].equals(t.name)) {

t.addChild(o);

label.setText("<html>"+t.getName()+"</html>");

setVisible(**true**);

}

}

}

}

}

});

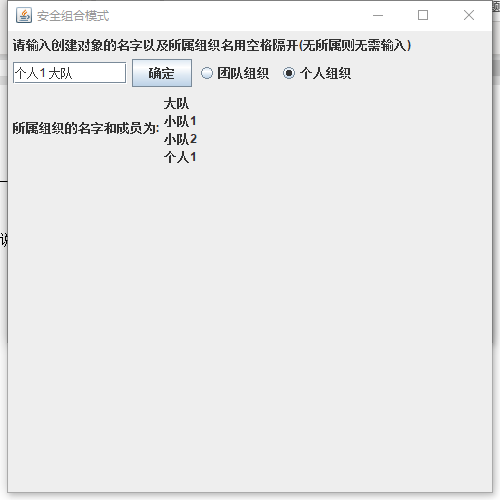
setVisible(**true**);

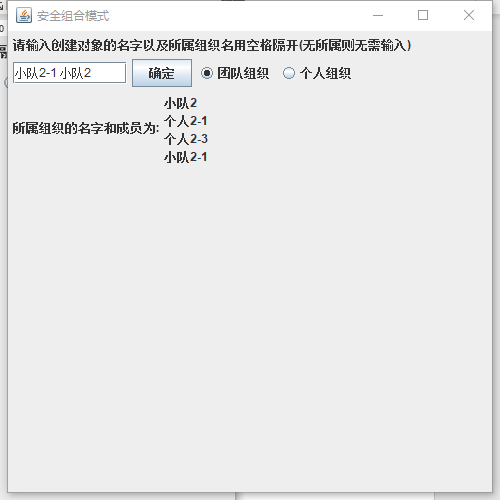
}

}

### (6) 实现效果

利用模式实现后运行的界面并简要说明





### (7) 模式的优点

安全，可以在编译是就能发现树叶类非法使用了组合类才能拥有的方法

### (8) 模式的不足

接口不一致

## 模式九：一致性组合模式

### (1) 解决问题类型

将部分和整体不加区别的对待

### (2) 解决设计思路

使用一致性性组合模式处理

### (3) 具体待解决的问题

个人(Person)与团队(Team)可以形成一个组织(Organization)：组织有两种：个人组织和团队组织，多个个人可以组合成一个团队，不同的个人与团队可以组合成一个更大的团队。创建个人，创建团队，每创建一个团队，每次团队成员有变化，就递归输出所有的成员。

### (4) UML设计



### (5) 核心代码

分析核心的代码

Person.java

**package** cn.edu.scau.cmi.zhengdinghao.OComposite;

**public** **class** Person **extends** Organization{

**public** String name;

**public** String getName() {

**return** name;

}

**public** Person(String name) {

**this**.name = name;

}

**public** **void** addChild(Organization organization) {

};

**public** **void** removeChild(Organization organization) {

};

}

Organizaion.java

**package** cn.edu.scau.cmi.zhengdinghao.OComposite;

**public** **abstract** **class** Organization {

**public** **abstract** String getName();

**public** **abstract** **void** addChild(Organization organization);

**public** **abstract** **void** removeChild(Organization organization);

}

OCcomposite.java

**package** cn.edu.scau.cmi.zhengdinghao.OComposite;

**import** java.awt.FlowLayout;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** java.util.ArrayList;

**import** javax.swing.ButtonGroup;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JRadioButton;

**import** javax.swing.JTextField;

**public** **class** OCcomposite **extends** JFrame {

ArrayList<Organization> organizations = **new** ArrayList<>();

ArrayList<Team> teams = **new** ArrayList<>();

Organization o = **null**;

JTextField name;

JButton button;

JRadioButton teamButton,personButton;

ButtonGroup group = **new** ButtonGroup();

JLabel label;

**public** **void** run() {

setBounds(550,240,500,500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

setTitle("一致性组合模式");

setLayout(**new** FlowLayout(5));

getContentPane().add(**new** JLabel("请输入创建对象的名字以及所属组织名用空格隔开(无所属则无需输入)"));

name = **new** JTextField(10);

getContentPane().add(name);

button = **new** JButton("确定");

getContentPane().add(button);

teamButton = **new** JRadioButton("团队组织");

personButton = **new** JRadioButton("个人组织");

teamButton.setSelected(**true**);

group.add(teamButton);

group.add(personButton);

getContentPane().add(teamButton);

getContentPane().add(personButton);

getContentPane().add(**new** JLabel("所属组织的名字和成员为"));

label = **new** JLabel();

add(label);

button.addMouseListener(**new** MouseAdapter(){

@Override

**public** **void** mousePressed(MouseEvent e) {

**if**(teamButton.isSelected()) {

o = **new** Team(name.getText().split("\\s")[0]);

teams.add((Team)o);

organizations.add(o);

**if**(name.getText().split("\\s").length != 1) {

**for**(Team t:teams) {

**if**(name.getText().split("\\s")[1].equals(t.name)) {

t.addChild(o);

label.setText("<html>"+t.getName()+"</html>");

setVisible(**true**);

}

}

}

}**else** {

o = **new** Person(name.getText().split("\\s")[0]);

organizations.add(o);

**if**(name.getText().split("\\s").length != 1) {

**for**(Team t:teams) {

**if**(name.getText().split("\\s")[1].equals(t.name)) {

t.addChild(o);

label.setText("<html>"+t.getName()+"</html>");

setVisible(**true**);

}

}

}

}

}

});

setVisible(**true**);

}

}

### (6) 实现效果

与安全性组合模式相同

### (7) 模式的优点

相比安全性组合模式而言达到了接口一致

### (8) 模式的不足

要对非法使用了叶子类的方法的情况作出相对应的处理措施

## 综合性实验的心得体会

这次综合实验是对之前几次实验的总结。通过回顾这些实验，我不仅巩固了这些实验之前让我学到了的思想，也让我获得了新的收获。在本次回顾中，我对之前的代码进行了纠错和优化，对程序的结构进行了梳理。希望通过本次实验我可以将这些收获到的更好的吸收加强运用到以后的软件开发之中。