一、 题目与要求

1．问题描述

题目：课程信息管理系统

说明：课程信息管理系统，分为教师和学生两端，教师可以在系统里添加课程，学生可以选课。同时可以查看各自的课表。教师可以在系统里给学生打分，学生可以查看自己的成绩单。

2．功能要求

* 1. 用户角色分为学生和教师，需要分别设计界面。
  2. 用户可注册，注册完成之后进行登录。
  3. 教师端登录之后功能：
     1. 可添加课程，课程信息应包括课程代码，课程名，学时，开课学期，课程类别（必修，选修），课程性质（公选，专业课），学分，上课地点，上课时间（每周，哪一周）。
     2. 对于课程可以进行增删改查。
     3. 对于每个课程可以查看学生的选课情况，查看选课人员表。
     4. 教师可以给学生成绩。
     5. 教师课表查看。
  4. 学生端登录之后的功能：
     1. 可以查看系统内的所有课程。
     2. 可以选课。
     3. 可以查看自己的选课情况。
     4. 可以查看自己的成绩和绩点。
     5. 学生课表查看。

1. **分析与设计**
   1. 功能与实现：
      1. 基础设计：
         1. 程序基本实现框架化。
            1. 数据模型类（如用户模型，课程模型，选课记录模型）在 Model 包存放，体现面向对象编程的思想。

每个数据模型都具有自身所具有的属性以及相应的方法。每个数据模型类必须要具有的方法是

String toString()：将类里的属性序列化输出成字符串。

String fromString(String):将外来的字符串转回相应的对象。

* + - * 1. 数据模型助手类在 ModelController 包存放。实现了类似数据库的功能。

这些类的构造方法具有将磁盘里的数据读入到内存中的数组列表的功能，同时会构造一个钩子，在程序被结束时会自动触发保存。

抽象类 ModelMasterController：

ArrayList<Object> getRecordsFromFile()：从指定的与类名同名的文件里读入数据。

saveRecordsToFile()：将当前内存中的数据写入到磁盘。

addObjectToList(Object object):将数据对象储存到内存当中。

deleteObjectFromList(Object object)：从内存中删除一个对象，会进行比对，参数传对象即可。

updateObjectToList(int index, Object object)：更新内存中一个对象。

int getIndex(Object object) ：获取相应对象的下标。

Object selectSingleObjectFromList(HashMap<String, String> key\_value):用指定的条件去筛选一个对象出来。

ArrayList<Object> selectManyObjectsFromList(HashMap<String, String> key\_value)：用指定的条件去筛选多个对象出来。

ArrayList<Object> getAllObjectsFromList()：将内存中所有对象都返回。

相应数据模型的助手类（ModelUserController等）均继承实现 上面这个抽象类，继承时只需要在构造方法里调用父类的构造方法并在参数里指定类名即可。此处使用了Java的反射机制。

* + - * 1. UI和逻辑处理部分：

抽象类MasterController：

其他类均继承实现该类，该类具有以下方法：

抽象方法 addToPane()：用于内容填充，其他类继承实现。

showDialog(String)：用于显示提示框。

isFill()：用于检查输入是否完整，在被继承实现时子类可以将要检查的 TextField 添加到 list\_check\_fill : HashSet<Object>中，然后在提交时调用该函数进行检查。

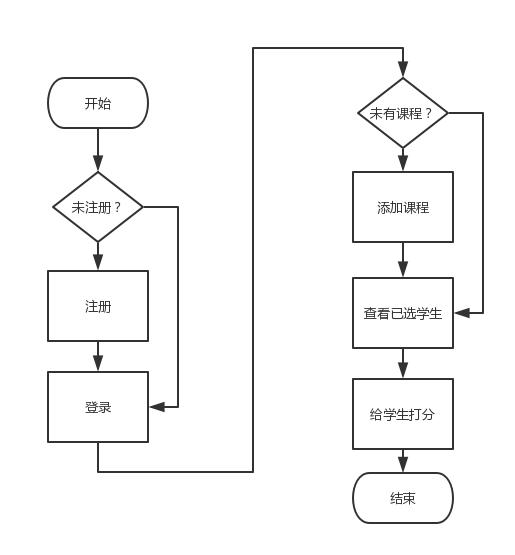
同时由于本类实现了 ActionListener ，所以下面的子类还要实现 actionPerformed(ActionEvent e)方法。

其他子类，继承实现 MasterController，一个子类对应一个窗口。

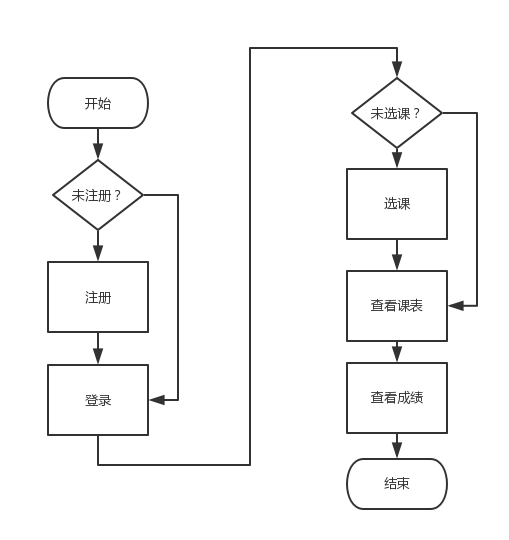
* + - * 1. 入口包：

在 Entry包中，Main 类，实例化数据助手类，新建窗口，同时有一个 SessionHold 类，用于实现类似于浏览器 Cookie 的机制，比如保存当前登录的用户之类的。

* + 1. 登录以及注册：
       1. 注册 UIController/RegController：
          1. 可以输入信息，进行注册，注册时可以选择角色。
          2. 调用模型类：User
          3. 调用模型助手类：ModelUserController
       2. 登录 UIController/LoginController：
          1. 可以输入用户名和密码尝试登录，登录时无需选择角色，如果登录成功会自动跳转到相应的角色界面。
    2. 教师端：
       1. 主界面UIController/TeacherMainController：
          1. 大部分教师端的功能均在此类中实现，包括课程的删改查，课程打分，查看学生列表，查看自己的课表等。
       2. 课程添加界面UIController/Teacher/AddCourseController：
          1. 课程的添加在这里。
    3. 学生端：
       1. 主界面UIController/StudentMainController：
          1. 学生端的功能均在此类中实现，包括课程的查看，选课，以及查看自己的成绩和自己的课表。
  1. 程序流程图：
     1. 教师端：

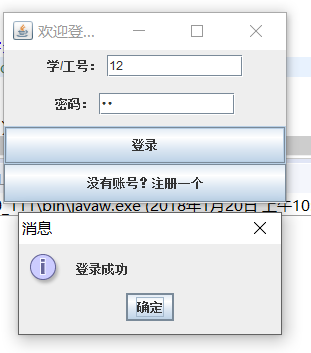


* + 1. 学生端



* 1. 外部文件储存格式
     1. Model.User.dat:
        1. 储存用户信息，将属性用|分割来保存。
     2. Model.Course.dat
        1. 储存课程信息，将属性用|分割来保存。
     3. Model.Choosen.dat
        1. 储存选课信息，将属性用|分割来保存。

1. **实现**
   1. 登录与注册界面
      1. 登录
         1. 截图：



* + - 1. 登录逻辑代码：

public void onLoginButtonClick() {

HashMap<String, String> queryCondition = new HashMap<String, String>();

queryCondition.put("user\_number", text\_user\_number.getText());

queryCondition.put("user\_password", text\_user\_password.getText());

User tempUser = (User) Main.ModelUser.selectSingleObjectFromList(queryCondition);

if(tempUser != null) {

this.showDialog("登录成功");

Main.SessionHold.put("CurrentLoginUser", (Object)tempUser);

if(tempUser.role\_type == 1) {

//老师

new TeacherMainController();

this.dispose();

} else {

//学生

new StudentMainController();

this.dispose();

}

} else {

this.showDialog("用户名或者密码不正确，请检查后再登录！");

}

}

* + 1. 注册
       1. 截图：



* + - 1. 注册逻辑代码：

public void onRegButtonClick() {

if(!this.isFill()) {

this.showDialog("信息填写不全，请检查后再提交！");

return;

}

if(!text\_user\_password.getText().equals(text\_user\_repeat\_password.getText())) {

this.showDialog("两次密码输入不一致，请检查后再提交！");

return;

}

//注册逻辑

HashMap<String, String> queryCondition = new HashMap<String, String>();

queryCondition.put("user\_number", text\_user\_number.getText());

if(Main.ModelUser.selectSingleObjectFromList(queryCondition) != null) {

this.showDialog("学工号已被注册，请检查后再提交！");

return;

}

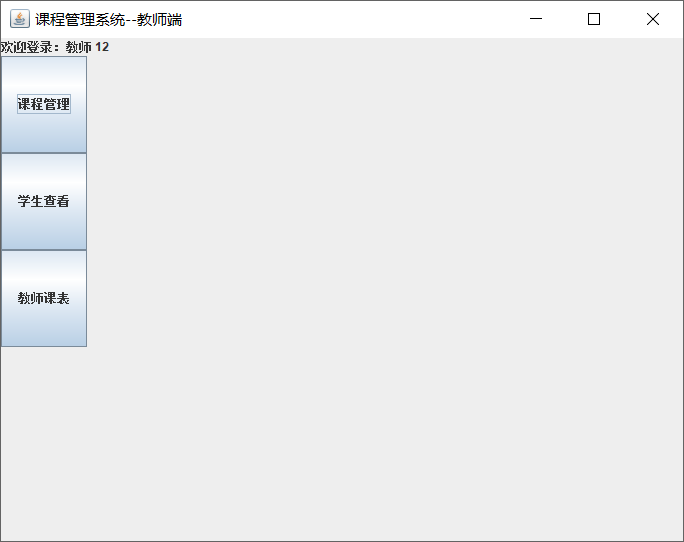
Main.ModelUser.addObjectToList(new User(text\_user\_number.getText(), text\_user\_name.getText(), text\_user\_password.getText(), (button\_role\_teacher.isSelected() ? 1 : 2)));

this.showDialog("注册成功，请登录！");

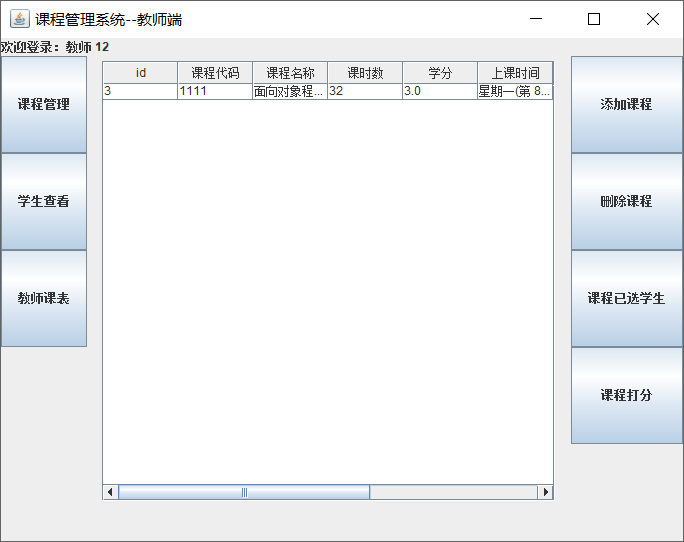
this.onBackButtonClick();

}

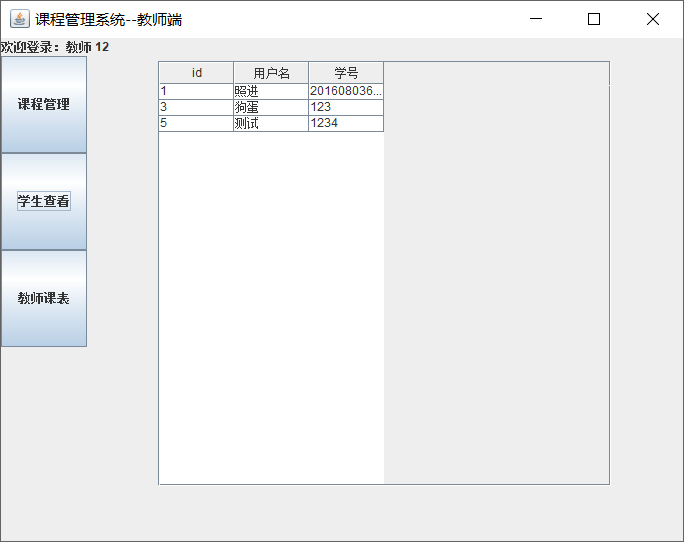
* 1. 教师端
     1. 截图：
        1. 主界面



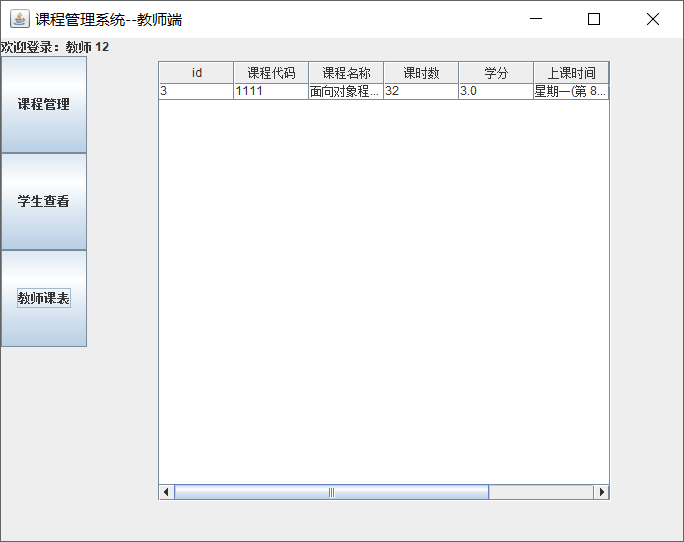
* + - 1. 课程管理



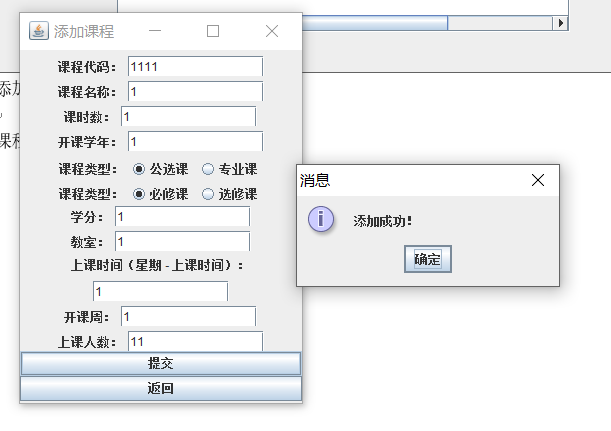
* + - 1. 学生查看



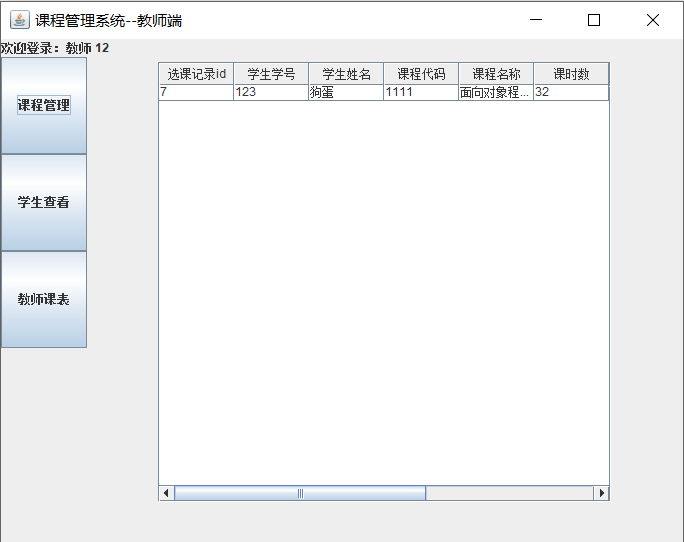
* + - 1. 教师课表（只显示当前学年课表）

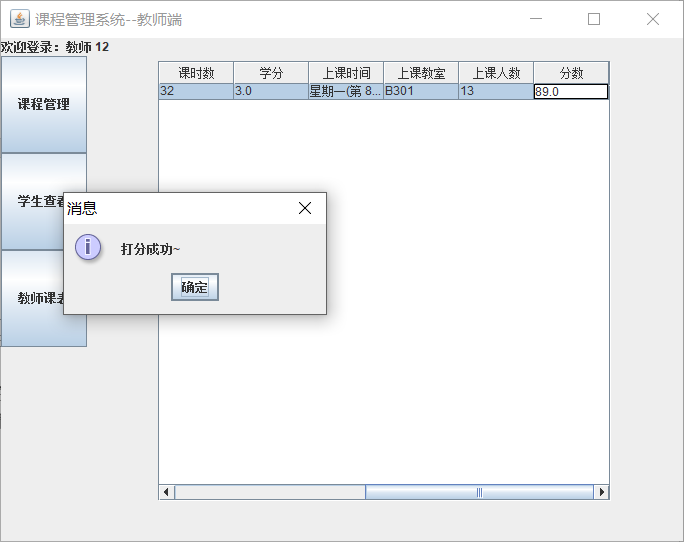


* + - 1. 添加课程

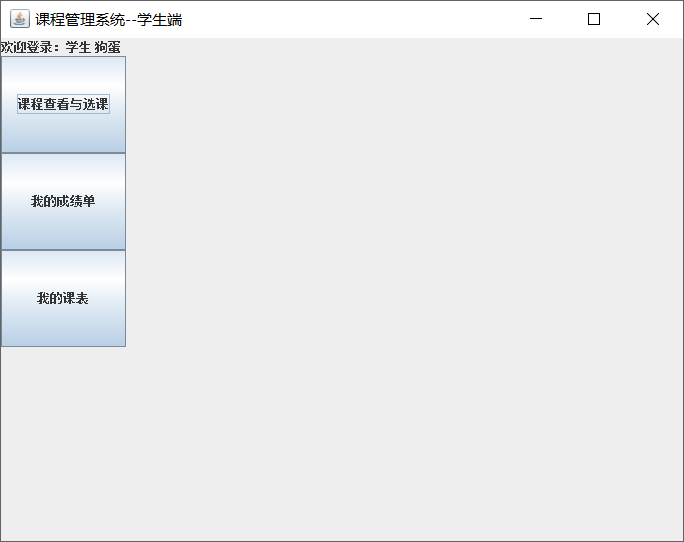


* + - 1. 课程相关操作

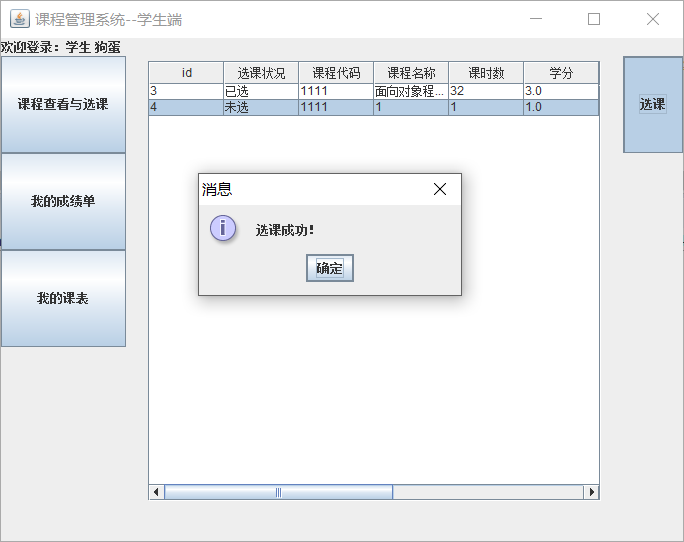




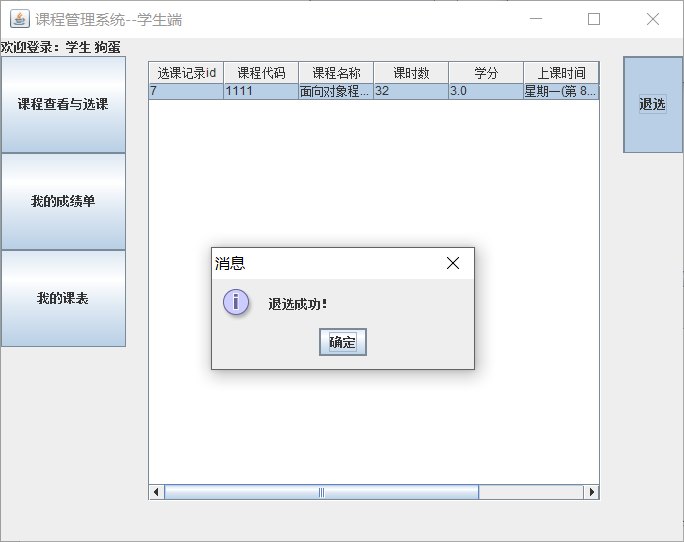
* + 1. 代码：
       1. 代码过多，此处不做节选。
  1. 学生端
     1. 截图：
        1. 学生端主界面



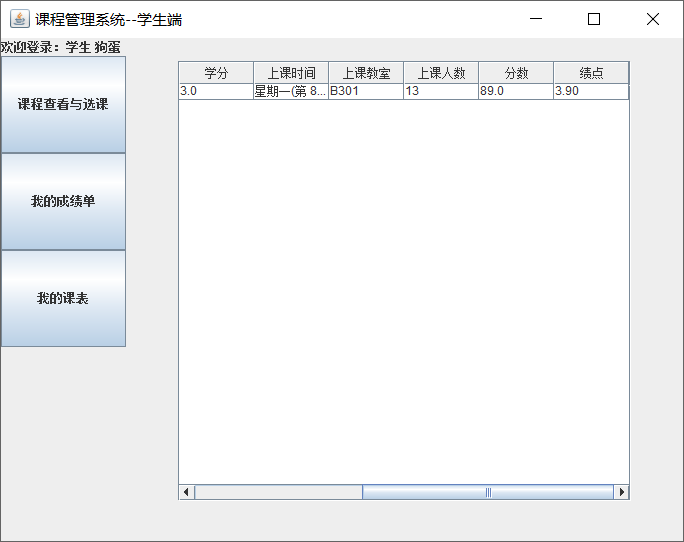
* + - 1. 查看课程以及选课



* + - 1. 查看课表（只显示当前学年课表）



* + - 1. 查看成绩



* + 1. 代码：
       1. 代码过多，此处不做节选。

1. **总结**

在整个的开发过程中，我遇到了一些问题，在解决这些问题的过程中我也有一些心得和体会，下面进行一些总结。

* 1. 架构的选择上

实训的时候，很多人都选择了直接在一个文件里写，之前我图省事也想过这样做，这样做的话肯定很快就能开发完了。但是，基于对于自己开发的考量和对自己新年的计划里要遵守开发规范的承诺，我选择将架构做好做细，在Java应用程序里尽量体现MVC的思想，结果也证明我基本做到了，虽然最开始开发的时候速度慢了一些进度有些落后，但一旦完成这个框架，开发的效率是指数级增长的，而且开发出来的程序便于维护，也比那些面条一样的代码运行稳定许多。

* 1. 功能规划

在一开始设计这个课程管理系统的时候，我设想过许多功能，但最后我都一个个的给修剪掉了，做那么多的功能可能固然绚丽，但也要考虑到自己的精力和技术水平。特别是实训只有短短的五天时间，真正在编码的时间可能比这个少的多，在这样的情况下要尽可能完美地实现功能就只能减少功能的数量，并且把这些功能做精，这是正确的策略。

* 1. 龟毛的JTable

在做数据呈现的时候，既然是用Java来写GUI的程序，那么就与之前C语言的命令行界面有很大的不同了，我在搜索Java的表格组件时，发现swing里提供了一个名叫 JTable的组件，在别人的介绍和截图里发现这个组件拿来做数据呈现非常合适。但我真正在使用这个组件的时候，才发现其实没那么容易。把我自己的数据给灌到 JTable 里，找到最合适的方法可费了一大把劲。再比如，把JTable的表头给比较合适地显示出来也是费了一番功夫，这些都得去琢磨它的API文档，去网上看看别人的程序，从中学习，总结出最适合自己的方法。

* 1. 奇巧淫技的使用

我在做主界面上内容的更替时，一开始发现如果不用鼠标缩放一下窗口的话，新的内容就不会出来。尝试了各种姿势来解决这个问题都没有成功，上网搜索了一下似乎也没人遇到过这种问题。那么我就想，不就是要改变一下窗口尺寸才能重新画图吗，那么好，我就把窗口放大一个像素再弄回来，这样新内容就出来了。有时候，这些小技巧也能解决蛮多问题。

再比如，在写数据模型助手类的时候，有很多参数都是可变的，这里我就尝试应用了 Java 的反射机制，在运行时再决定那些原本要在写程序时就要决定的东西，非常方便，代码的复用性也高很多了。

通过本次实训，我第一次尝试写了一个完整的 Java 应用程序，进一步理解了面向对象的思想，收获很多，限于篇幅到此为止。

1. **附件**

源代码见压缩包