**计算机科学与工程学院 实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验课程名称** | | **Java语言及程序设计** | | | **实验总成绩** |  |
| **专业** | **人工智能** | | **班级** | **2101（未来实验班）** | **指导教师签字** | **陈郭成** |
| **学号** | **20215530** | | **姓名** | **王培东** | **实验报告批改时间** |  |
| **实验报告分项成绩**   |  |  |  | | --- | --- | --- | | **序号** | **实验项目** | **成绩** | | **1** | **HelloWorld和面向对象入门、进阶** |  | | **2** | **文件的读和写** |  | | **3** |  |  | | **4** |  |  | | **5** |  |  | | **6** |  |  | | **7** |  |  | | **8** |  |  | | | | | | | |
| **实验课程总结**  从以下方面总结：1.实验体现知识应用和初步研究能力；2. 反映基本观察、发现问题和分析问题能力；3. 实验项目内容或者实验课程是否存在问题及下一年度改进意见）4.其他方面  这里要填写自己真实的体验、收获、意见等。 | | | | | | |

**实验一**

**实验目的与要求：**

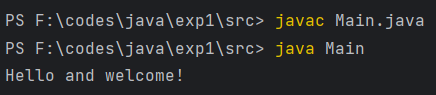
1. Java开发环境的熟悉。
2. 使用Eclipse编辑、编译、运行、调试Java程序。
3. 以选课系统业务为例完成面向对象入门、进阶。

**实验知识点：**

1. JVM、JRE、JDK的安装位置与区别。

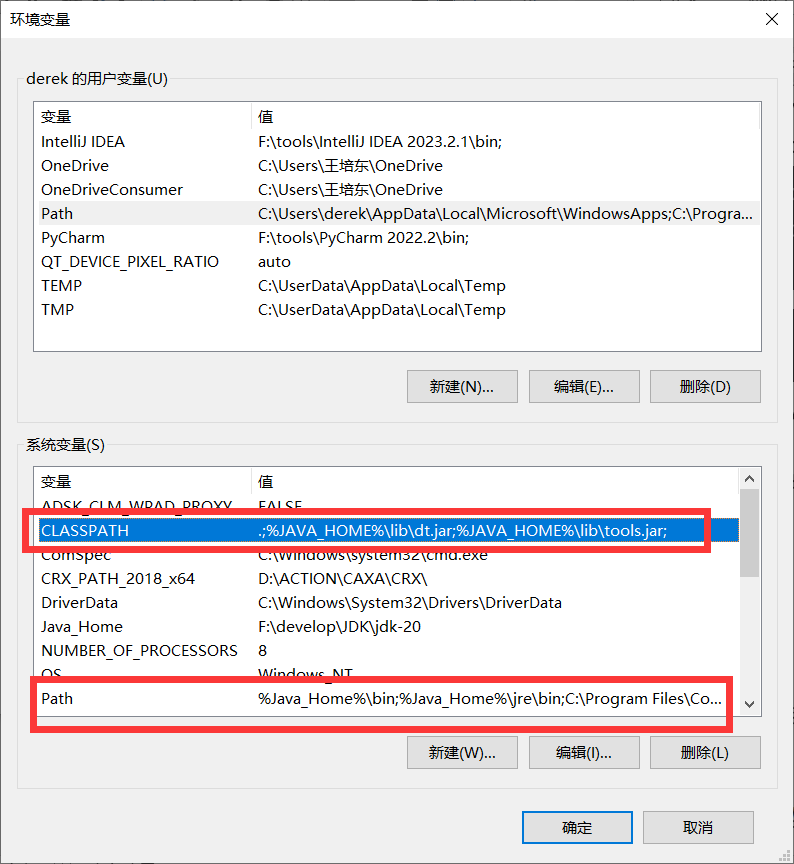
JDK(Java Development Kit)是Java开发工具，包含了编译器javac、文档注释工具javadoc、调试器jdb、反编译工具javap、运行时环境JRE等，JRE安装在JDK里面，JRE是运行已编译Java程序所需的所有内容的集合，主要包含基础类库、JVM等。JVM安装在JRE下。

1. 命令行运行javac, java, javac -cp, java -cp 。





1. 环境变量配置（PATH，CLASSPATH，SOURCEPATH的设定方法与应用）。



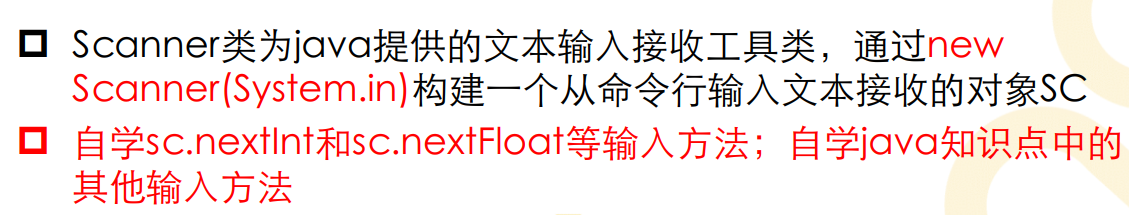
4. 包管理：package；import；javac -d。

5. Java基础语法：类型与类型转换；变量；运算符；流程控制（判断分支、循环）；类的继承、多态、封装。

6. JDK帮助文档的使用。

**实验内容：**

（请同学，按课件里的内容把预备知识、步骤，程序框图、调试好程序的源代码、结果截图及存在的问题写在下面，最好有类图和系统框架图）



sc.nextInt():读入一个int类型的变量

sc.nextFloat():读入一个float类型的变量

其他输入方法：

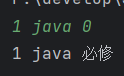
System.in.read() ： 从控制台以ascii码方式读取一个字符

DataInputStream din = new DataInputStream(System.in); 和scanner差不多 eg: double x = 0; x = Double.parseDouble(din.readLine());

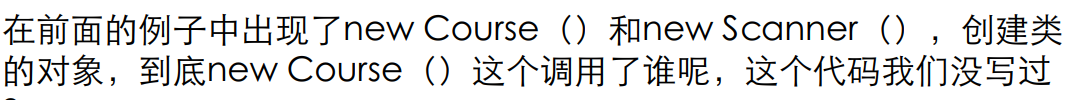
JOptionPane相关方法 String str1 = JOptionPane.showInputDialog(“输入消息框”,“0”); 此语句会显示一个输入消息框，可以输入数据。数据以字符串形式读入。

a = Integer.parseInt(str1); 读入一个整型a



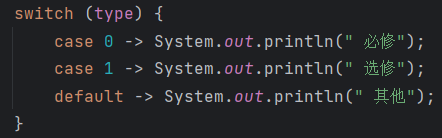


Course c = new Course();  
Scanner sc = new Scanner(System.*in*);  
c.id = sc.nextInt();  
c.name = sc.next();  
c.type = sc.nextInt();  
c.show();

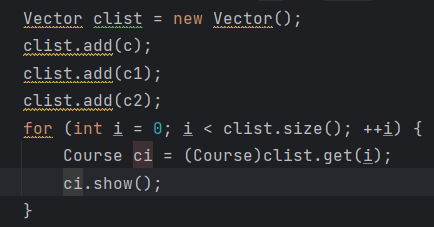


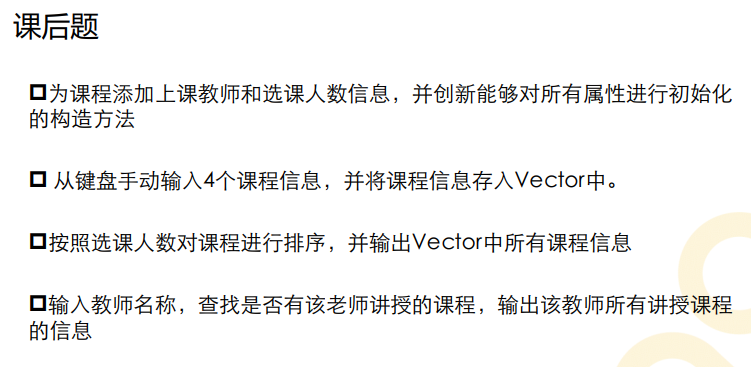
调用了Course()的构造函数。没写过，但Java为所有类创建了一个默认的构造函数。Java所有新建的类对象都继承于Object，类中有隐含了一个从Object中继承的构造方法

Switch：

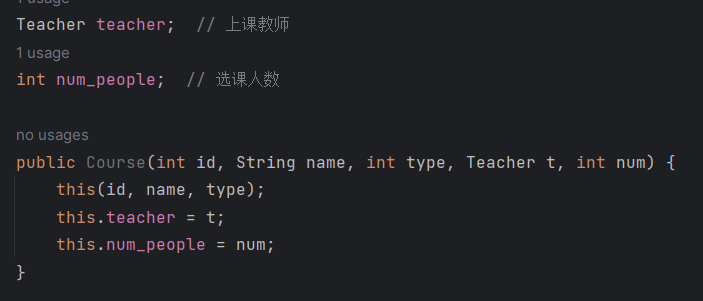


for :

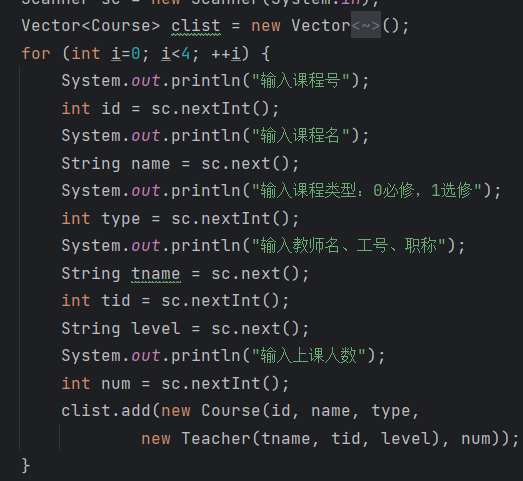


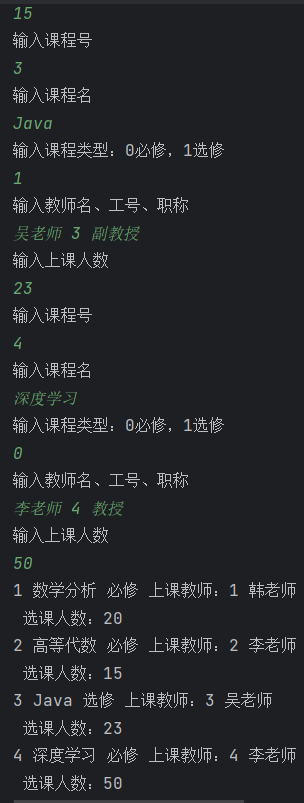
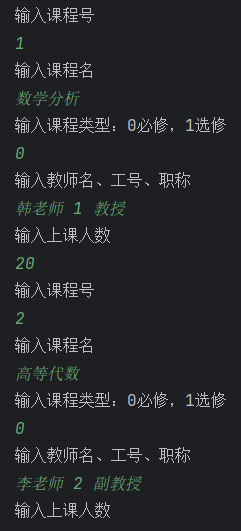


1.

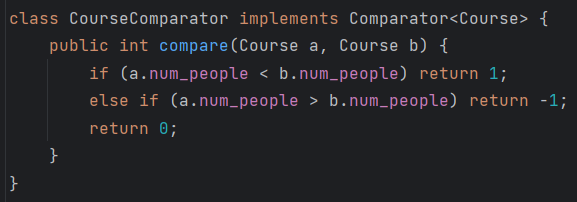


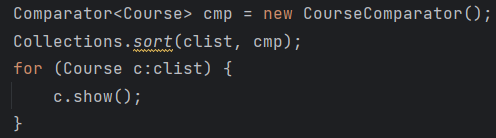
2.

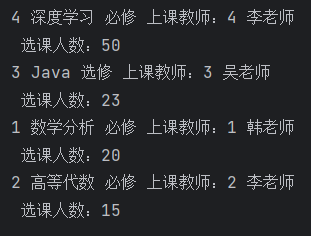




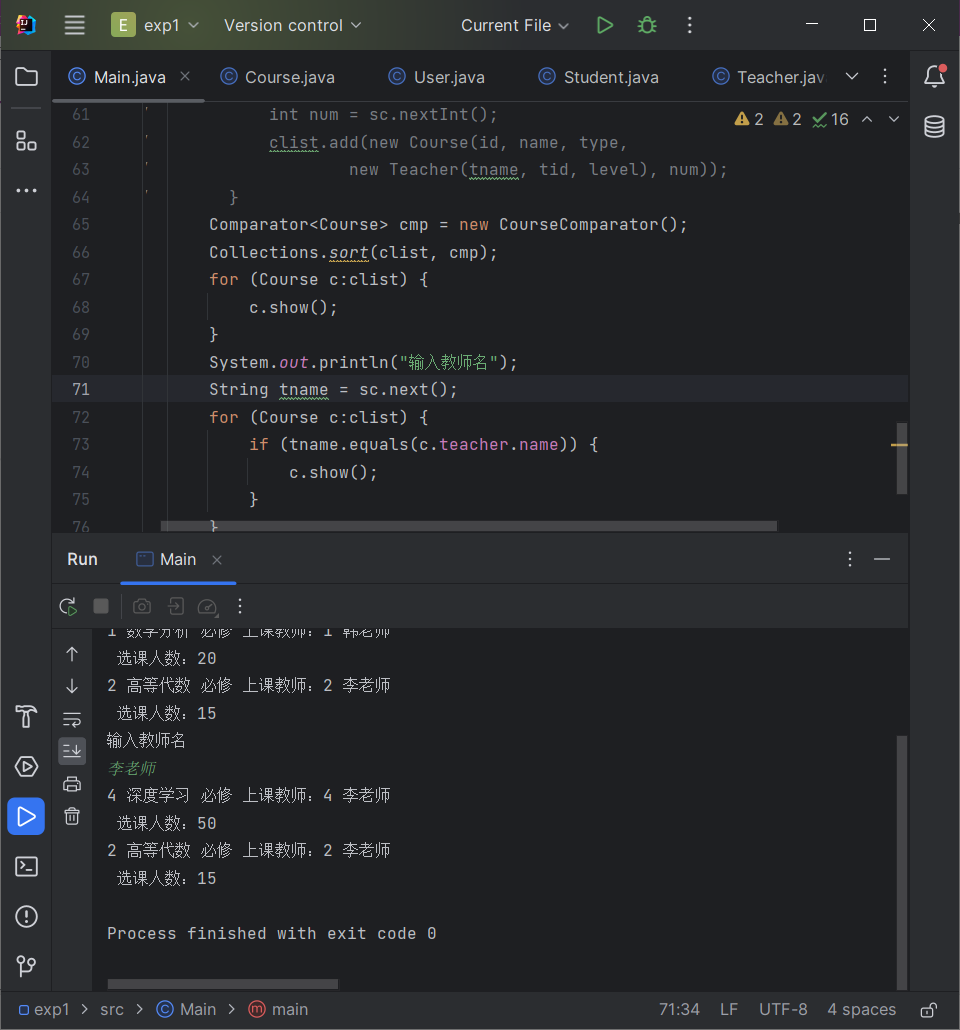
3.



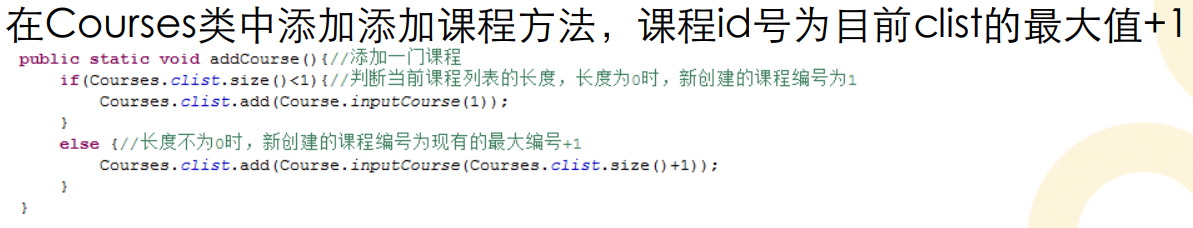




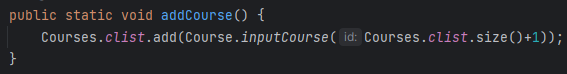
4.



在实验一课后题解析中，增加课程的实现方法为：



完全多此一举，没必要特判，size<1就是0。

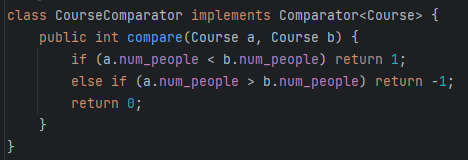


这样写完全够了。

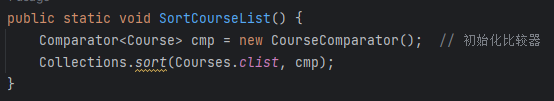
排序方法那块，我没有按一下写法：

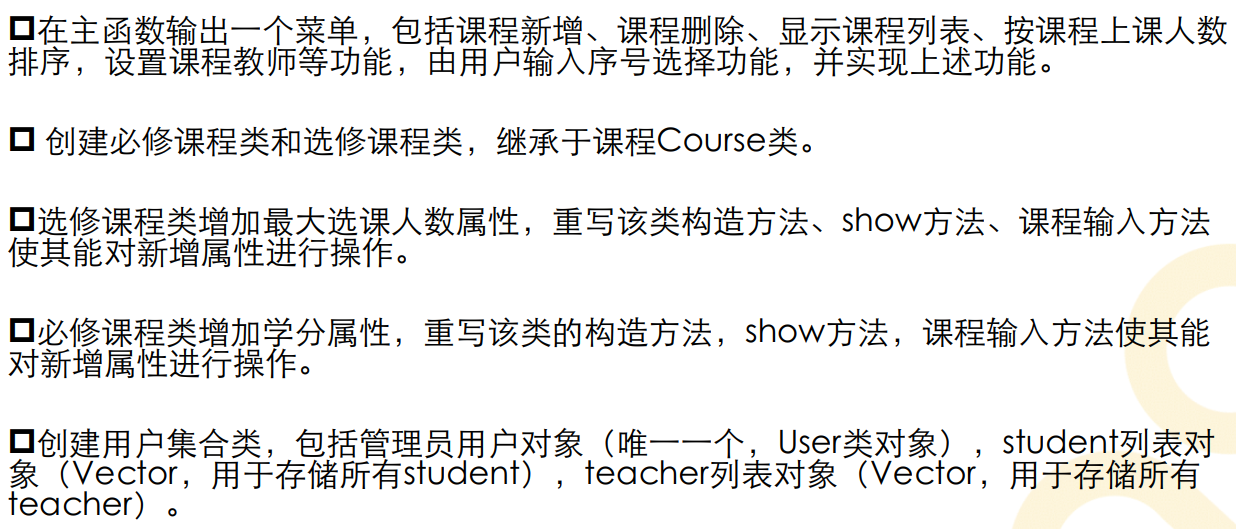


我重载了Comparator<T> 接口：

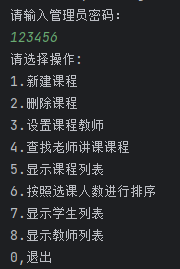


用选课人数来排序课程，使得代码更加简洁：



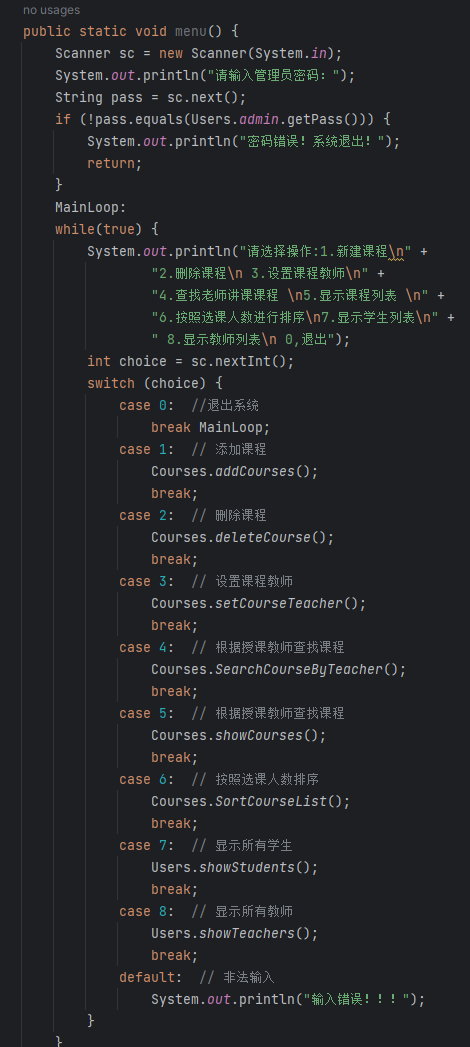


菜单：

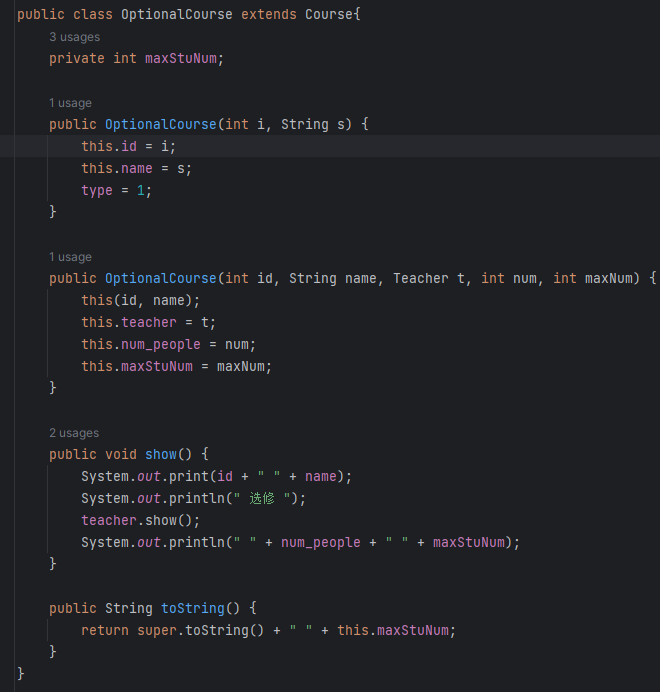


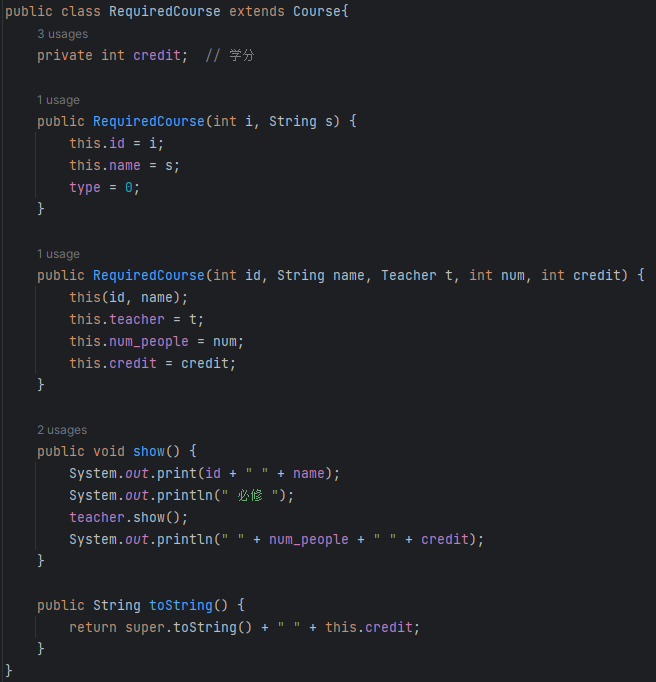


实现：

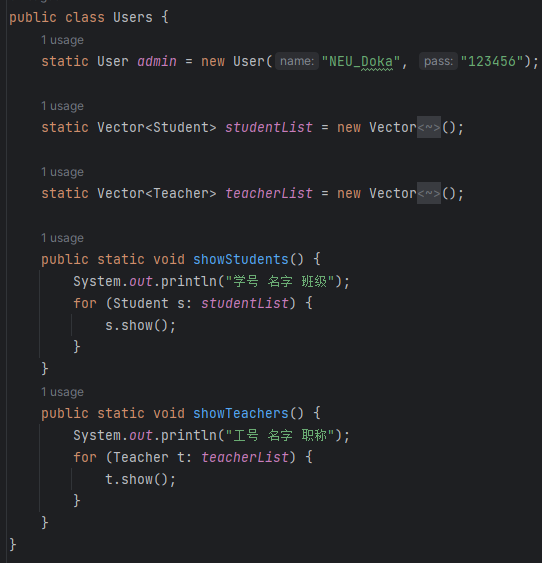


增设选修课必修课类：





用户集合类：



**实验二**

**实验目的与要求：**

1. 完成实验一课后题1.为课程添加上课教师和选课人数信息，并创新能 够对所有属性进行初始化的构造方法。
2. 完成实验一课后题2.从键盘手动输入4个课程信息，并将课程信息存入Vector中。
3. 完成实验一课后题3.按照选课人数对课程进行排序，并输出Vector中所有课程信息.
4. 完成实验一课后题4.输入教师名称，查找是否有该老师讲授的课程，输出该教师所有讲授课程的信息。
5. 完成实验一课后题5.在主函数输出一个菜单，包括课程新增、课程删除、显示课程列表、按课程上课人数排序，设置课程教师等功能，由用户输入序号选择功能，并实现上述功能。
6. 完成实验一课后题6.创建必修课程类和选修课程类，继承于课程Course类。
7. 完成实验一课后题7.选修课程类增加最大选课人数属性，重写该类构造方法、show方法、课程输入方法使其能对新增属性进行操作。
8. 完成实验一课后题8.必修课程类增加学分属性，重写该类的构造方法，show方法，课程输入方法使其能对新增属性进行操作。
9. 完成实验一课后题9.创建用户集合类，包括管理员用户对象（唯一一个，User类对象），student列表对象（Vector，用于存储所有student），teacher列表对象（Vector，用于存储所有teacher）。
10. 文件的写入。
11. 文件的读取。

**实验知识点：**

1. 类的重构。
2. 类的构造方法。

3. Vector类。

4. bufferReader对象。

5. fileReader对象。

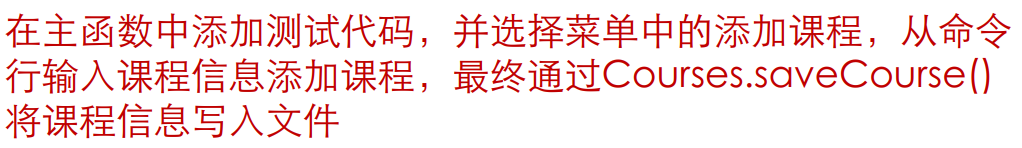
6. JDK帮助文档的使用。

**实验内容：**

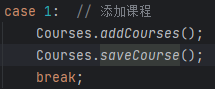
（请同学，按课件里的内容把预备知识、步骤，程序框图、调试好程序的源代码、结果截图及存在的问题写在下面，最好有类图和系统框架图）

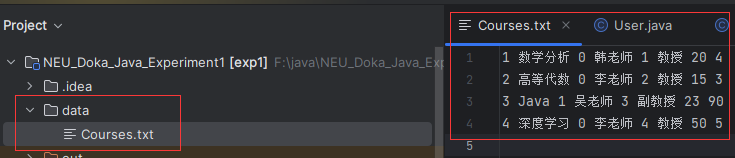
**（4）至（12）均已在实验一中完成**

文件的写入：

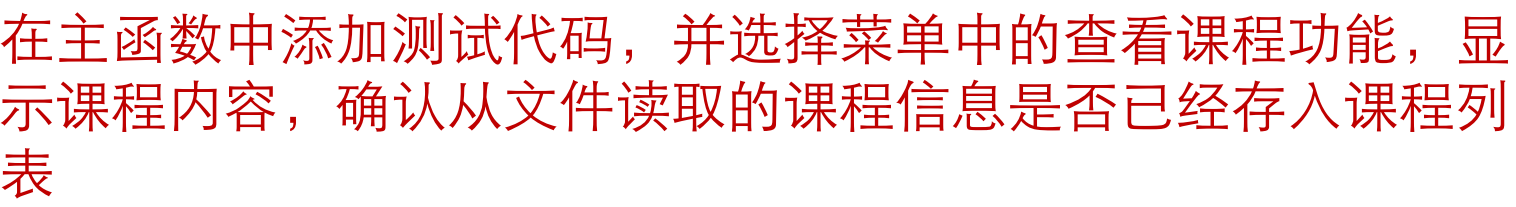


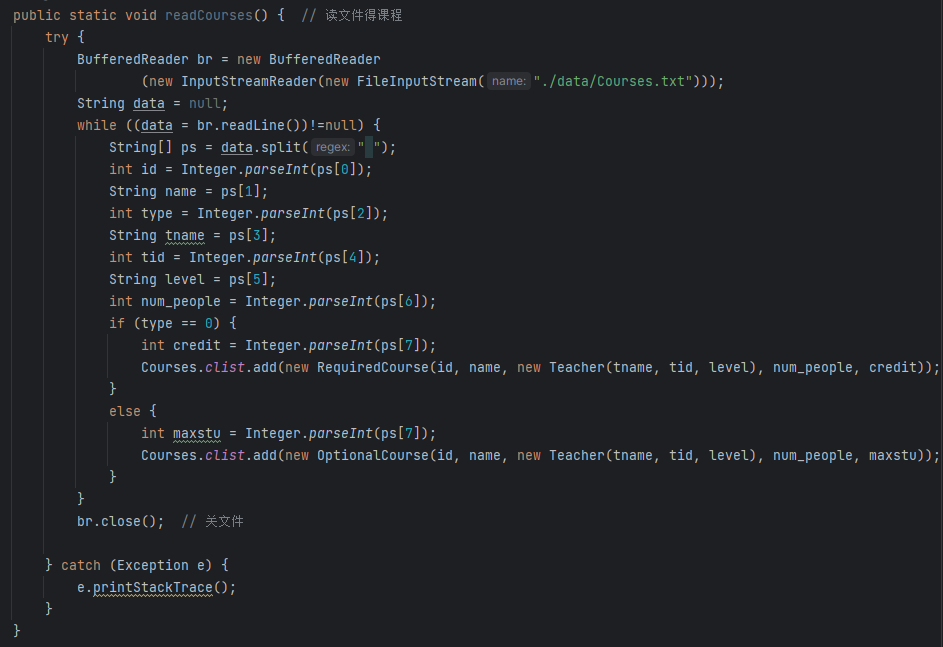


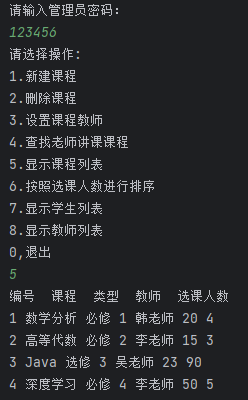
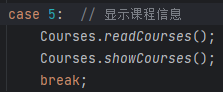


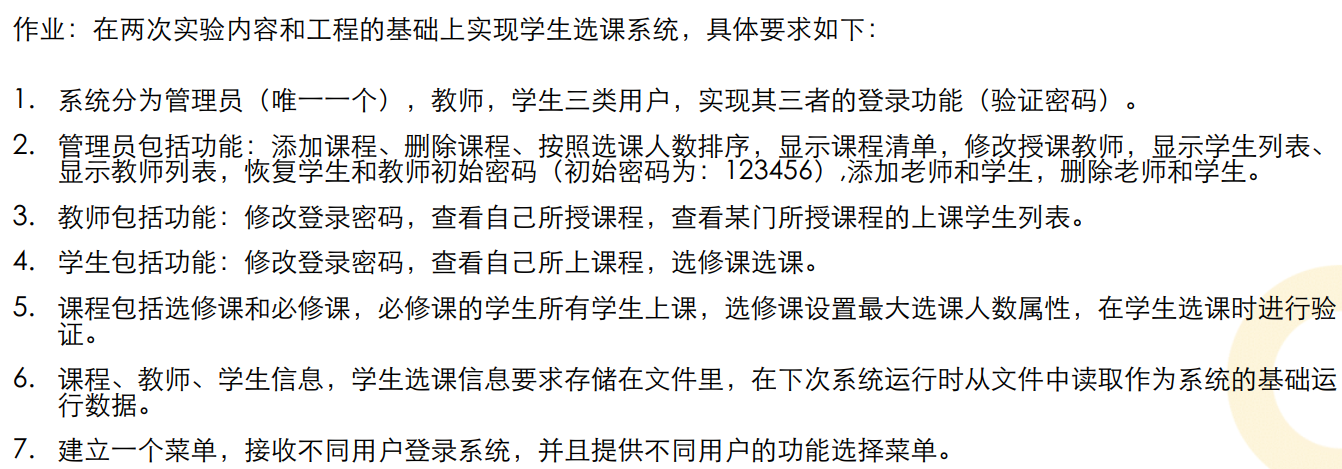


文件写入成功。





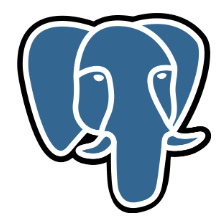




观察后续业务需求 要存储教师 学生 还有学生选课相关

信息 故想用数据库来存储相关信息 Java有jdbc，访问数据库很方便，但是实验一设计的似乎不是很好，Student类继承自User类 登录时要用name和password 但学生之间重名很常见，不应当用name唯一标识一个学生，用学号更合适，但就这样吧，前面已经这样写了。

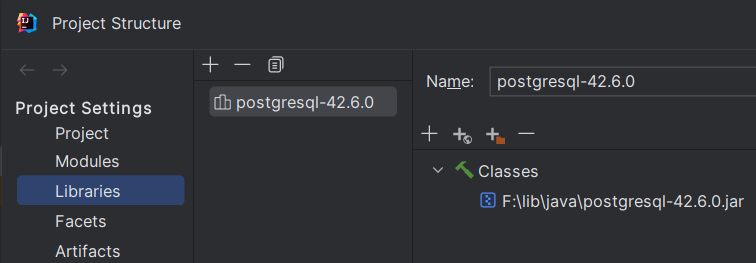
数据库系统用的PostgreSQL x64 10



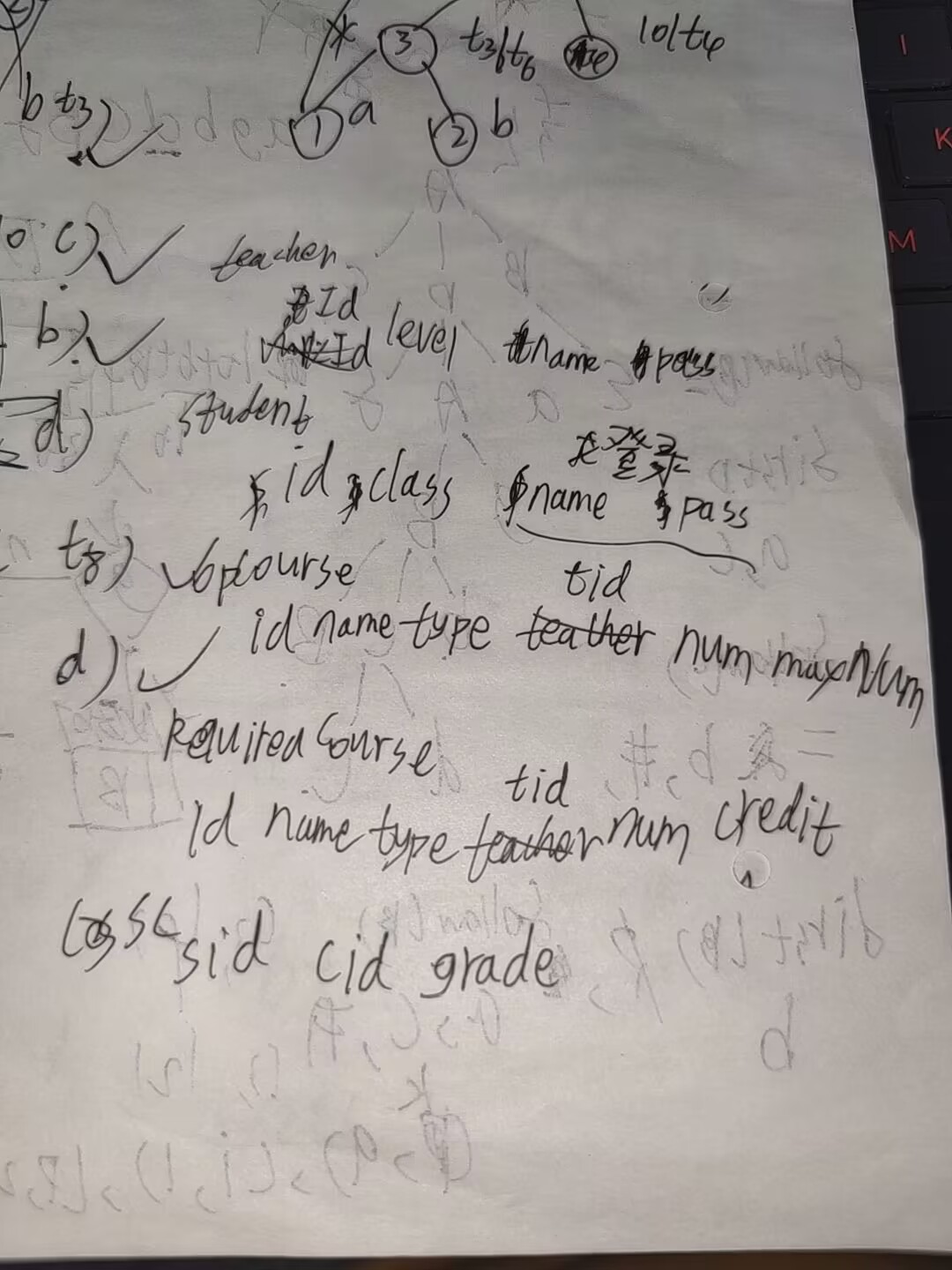
1.先安装JDBC 配置编程环境，安装网址

[**https://jdbc.postgresql.org/download**](https://jdbc.postgresql.org/download)

安装完成后，在项目工程上添加这个jar的依赖：

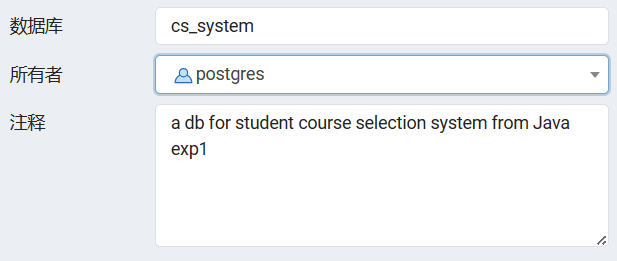


2.设计数据库：



如上图，本数据库有五个关系 学生、教师、选修课、必修课、选课。学生和课程之间通过选课联系起来，这样就可以方便得查询自己所上课程及其成绩。课程关系中，存有教师id，这样可以通过联合教师关系和课程关系查得教师所授课程，利用学生、课程、选课三个关系，可以联合查得教师所授某门课程的上课学生列表。登陆功能可以在相应用户表中查询姓名密码是否匹配。

3.创建数据库与所需表：（这一步也可以用jdbc链接数据库做，但直接在图形化界面操作更方便）



创建数据库cs\_system

创建各个关系表：（各属性的类型与前面实验所做一致）

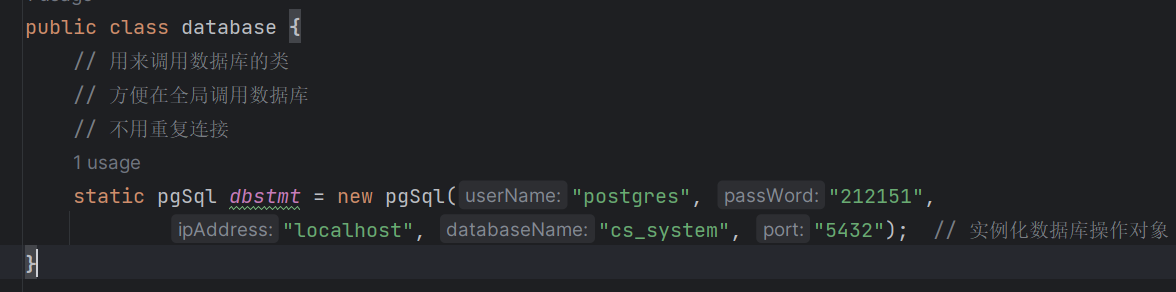
 



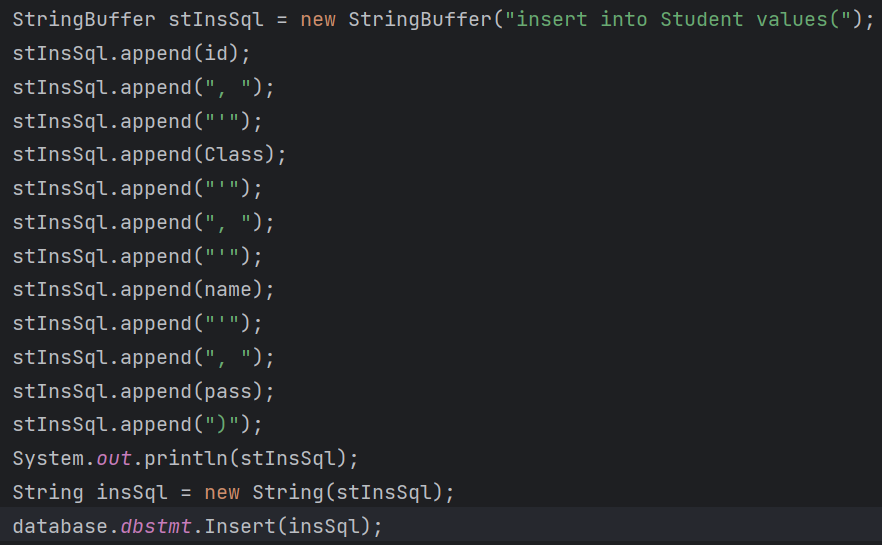
以后信息都存在数据库，启动时从数据库读取数据，这样相较于把信息存在txt文件更安全，数据库有安全访问机制，而txt文件谁都能打开。因此要重写之前实现的信息读写功能，在修改信息的方法中都要同步修改数据库中的信息。

编写了pgSQL类，方便对数据库进行各种操作：

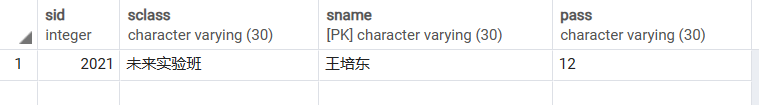
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.Statement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.ResultSetMetaData;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
  
public class pgSql {  
 private final Connection connect; //  
 private final String userName; // 用户名  
 private final String passWord; // 密码  
 private final String ipAddress; // ip地址  
 private final String databaseName; // 数据库名字  
 private final String port; // 数据库服务接口 默认为5432  
  
 public pgSql(String userName, String passWord, String ipAddress, String databaseName, String port) {  
 this.userName = userName;  
 this.passWord = passWord;  
 this.ipAddress = ipAddress;  
 this.databaseName = databaseName;  
 this.port = port;  
 this.connect = this.Connect();  
 }  
  
 // 建立链接  
 private Connection Connect() {  
 Connection c = null;  
 try {  
 Class.*forName*("org.postgresql.Driver"); // 自动注册driver  
 c = DriverManager  
 .*getConnection*("jdbc:postgresql://" + this.ipAddress + ":" + this.port + "/" + this.databaseName,  
 this.userName, this.passWord); // 构建连接  
 } catch (Exception e) { // 异常处理  
 e.printStackTrace();  
 System.*err*.println(e.getClass().getName() + ": " + e.getMessage());  
 System.*exit*(0);  
 }  
 return c;  
 }  
  
 // 关流操作  
 public void close() {  
 Connection c = this.connect;  
 try {  
 c.close();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
  
 // 查询操作  
 public List<HashMap<String, Object>> Select(String sql) {  
 //得到之前建立的connection  
 Connection c = this.connect;  
 //创建statement  
 Statement stmt = null;  
 //创建返回最终查询的数据集合  
 List<HashMap<String, Object>> list = new ArrayList<>();  
 try {  
 //初始化操作对象  
 stmt = c.createStatement();  
 //执行sql语句  
 ResultSet rs = stmt.executeQuery(sql);  
  
 ResultSetMetaData metaData = rs.getMetaData(); // 获取元数据  
 int columnCount = metaData.getColumnCount(); // 列的数量  
  
 while (rs.next()) {  
 HashMap<String, Object> map = new HashMap<>();  
 for (int i = 1; i <= columnCount; i++) {  
 // 获取列名  
 String name = metaData.getColumnName(i);  
 // 获取对应的元素  
 Object object = rs.getObject(i);  
 map.put(name, object);  
 }  
 list.add(map);  
 }  
 //关流操作  
 rs.close();  
 stmt.close();  
 } catch (SQLException throwable) {  
 throwable.printStackTrace();  
 }  
 return list;  
 }  
  
 // 插入操作  
 public Boolean Insert(String sql) {  
 //得到之前建立的connection  
 Connection connect = this.connect;  
 //创建statement  
 Statement stmt = null;  
 int count = 0;  
 try {  
 // 初始化创建对象  
 stmt = connect.createStatement();  
 // 取消自动提交  
 connect.setAutoCommit(false);  
 // 执行添加操作  
 count = stmt.executeUpdate(sql);  
  
 // 关流  
 stmt.close();  
 // 手动提交事务  
 connect.commit();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 try {  
 // 回滚事务 这里其实没有必要 一般将多个语句封装为一个事务  
 connect.rollback();  
 } catch (Exception e1) {  
 e1.printStackTrace();  
 }  
 }  
 return count != 0;  
 }  
  
 // 更新操作  
 public Boolean Update(String sql) {  
 //得到之前建立的connection  
 Connection connect = this.connect;  
 //创建statement  
 Statement stmt = null;  
 int count = 0;  
 try {  
 // 初始化创建对象  
 stmt = connect.createStatement();  
 // 取消自动提交  
 connect.setAutoCommit(false);  
 // 执行添加操作  
 count = stmt.executeUpdate(sql);  
  
 // 关流  
 stmt.close();  
 // 手动提交事务  
 connect.commit();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 try {  
 // 回滚事务 这里其实没有必要 一般将多个语句封装为一个事务  
 connect.rollback();  
 } catch (Exception e1) {  
 e1.printStackTrace();  
 }  
 }  
 return count != 0;  
 }  
  
 //删除  
 public void Delete(String sql) {  
 //得到之前建立的connection  
 Connection c = this.Connect();  
 Statement stmt = null;  
 try {  
 c.setAutoCommit(false);  
 stmt = c.createStatement();  
  
 stmt.executeUpdate(sql);  
  
 stmt.close();  
 // 手动提交事务  
 c.commit();  
 } catch (SQLException throwable) {  
 throwable.printStackTrace();  
 try {  
 // 回滚事务  
 c.rollback();  
 } catch (Exception e1) {  
 e1.printStackTrace();  
 }  
 }  
 }  
}

同时，创立database类，用来方便反复访问数据库，这样就不用重复链接了：

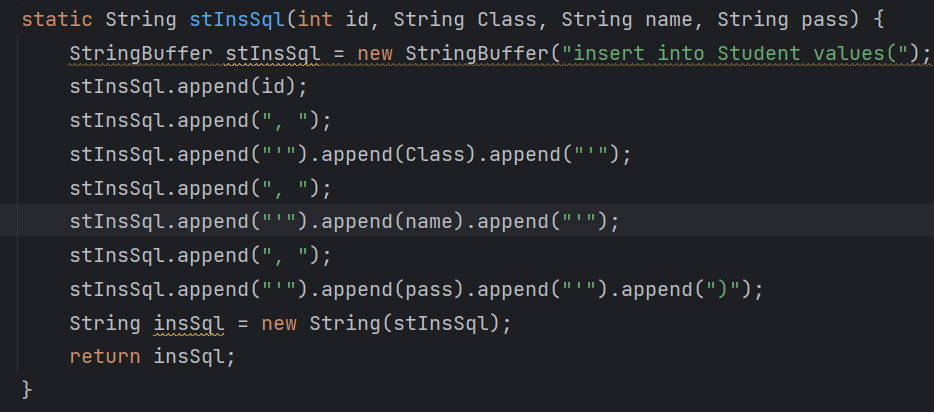
在addStudents方法中加入数据库插入：



测试：

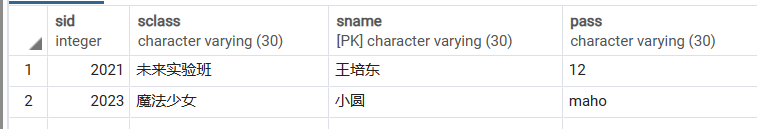


前面得到SQL语句的过程有些冗长，封装一下：



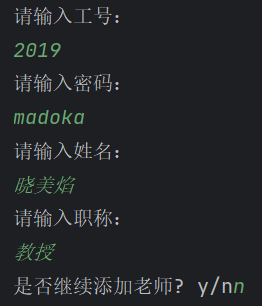
测试：

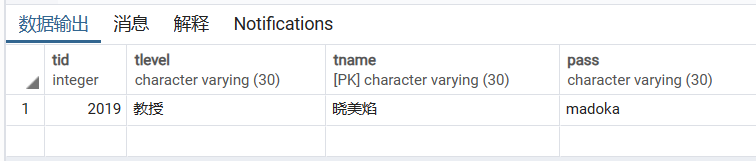




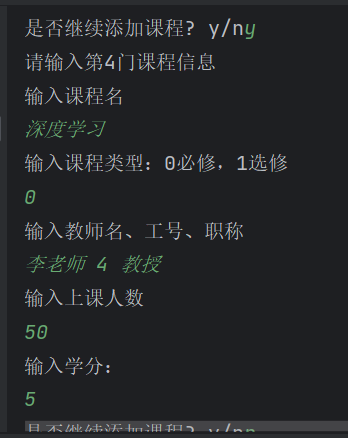
插入教师那些操作同理，不再赘述。

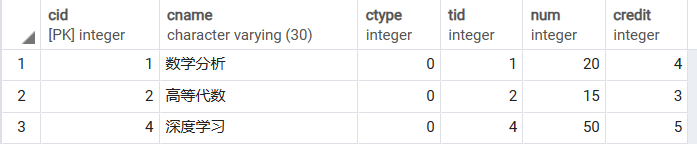
测试：





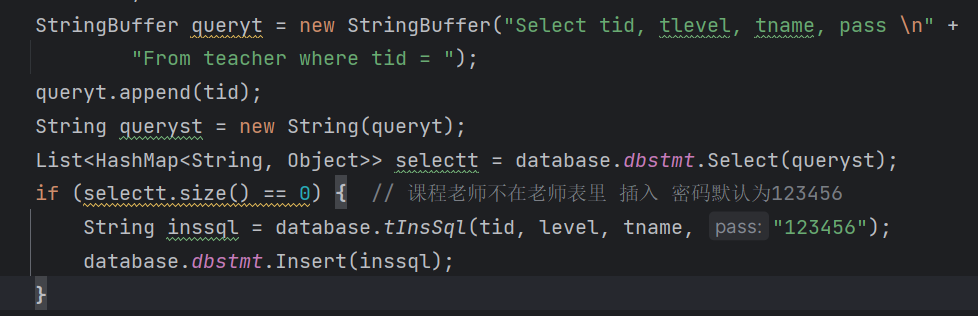
测试课程数据库：



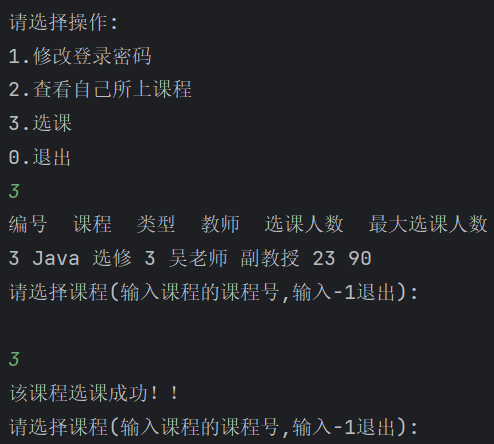




发现：这样单纯插入课程时只修改课程关系表不合理，因为只存了教师工号，而课程类的成员变量是整个教师类。在初始化时，要通过教师工号在教师表中查询得到教师的所有信息，再传给课程类的构造函数。如果插入课程时录入的教师信息在教师表中找不到，就会出错。因此，在插入课程时，在教师表中查询是否有该教师，若没有，帮其自动初始化。

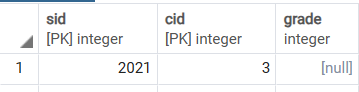


选课：









查课：

