**西北工业大学实验报告**

课程名称: 高级语言程序设计实验

实验课次周次: 第二次（第八周）

实验地点: 实验大楼B309-1

学生姓名：

学 号:

实验时间: 2024-04-20

提交时间: 2024-04-26

|  |
| --- |
| **成绩评定：（满分为100分）**  **指导教师评语：**  **指导教师签名：** |

目 录

[1 Java实验六 异常处理 2](#_Toc32450)

[1.1 实验内容 2](#_Toc28601)

[1.2 Java源代码文件说明 2](#_Toc2914)

[1.3 Java实验结果 5](#_Toc22767)

[2 Java实验七 学生信息管理系统 9](#_Toc2591)

[2.1 实验内容 9](#_Toc30393)

[2.2 Java源代码文件说明 9](#_Toc1887)

[2.3 Java实验结果 16](#_Toc3388)

[3 UML实验一 UML建模基础 17](#_Toc25251)

[3.1 实验内容 17](#_Toc16935)

[3.2 UML结果文件说明 17](#_Toc1637)

[3.3 UML实验结果 18](#_Toc19328)

[4 UML实验二 用例图 19](#_Toc22556)

[4.1 实验内容 19](#_Toc10796)

[4.2 UML结果文件说明 20](#_Toc26947)

[4.3 UML实验结果 22](#_Toc30038)

[5 实验总结 24](#_Toc31113)

[5.1 实验中遇到的问题以及解决方法 24](#_Toc9819)

[5.2 实验体会 24](#_Toc7065)

# Java实验六 异常处理

## 实验内容

### **1.**基本指导：

编写一个程序，同时捕获数组越界和被0除的异常，说明异常处理语句try-catch-finally的处理机制。

### **2.**思考问题：

1. 本程序中throws和throw语句的作用是什么？
2. 本程序中是如何定义用户自定义异常的？
3. 本程序是如何处理程序产生的用户自定义异常的？
4. 如果将程序中的“public void regist(int num) throws MyException”改为“public void regist(int num) ”，会出现什么样的情况？

### **3.**上机作业：

编写一个程序，将字符串转换成数字。请使用try-catch-finally语句处理转换过程中可能出现的异常。

## Java源代码文件说明

### 基本指导：

public class Catch Demo {

public static void main(String[] args) {

int number[]={4,8,16,32,64,128,256,512};

int denom[]={2,0,4,4,0,8};

for(int i=0;i<number.length ;i++){

try {

System.out.println(number[i] + " / " + denom[i] + "is " +

number[i] / denom[i]);

}catch(ArithmeticException exc){

System.out.println("Can't be divided be zero");

}

catch(ArrayIndexOutOfBoundsException exc){

System.out.println("No matching element found.");

}

}

}

}

异常处理添加finally块：

finally{

System.out.println("Finally巳执行");

}

### 思考问题：

class UserException extends Exception{

private int idnumber;

public UserException(String message,int id){

super(message);

this.idnumber=id;}

public int getId(){

return idnumber;

}

}

public class TestException {

public void regist(int num) throws UserException{

if(num<0){

throw new UserException("人数为负值，不合理",3);

}

System.out.println("登记人数: "+num);

}

public void manager(){

try{

regist(-100);

}catch(UserException e){

System.out.println("登记出错，类别："+e.getId());

}

System.out.println("本次登记操作结束");

}

public static void main(String[] args) {

TestException t=new TestException();

t.manager();

}

}

### 上机作业

import java.util.Scanner;

public class StringToNumber {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("请输入一个字符串：");

String input = scanner.nextLine();

try {

int number = Integer.parseInt(input);

System.out.println("转换后的数字为：" + number);

} catch (NumberFormatException e) {

System.out.println("无法转换（有非数字字符）");

} finally {

System.out.println("程序结束。");

}

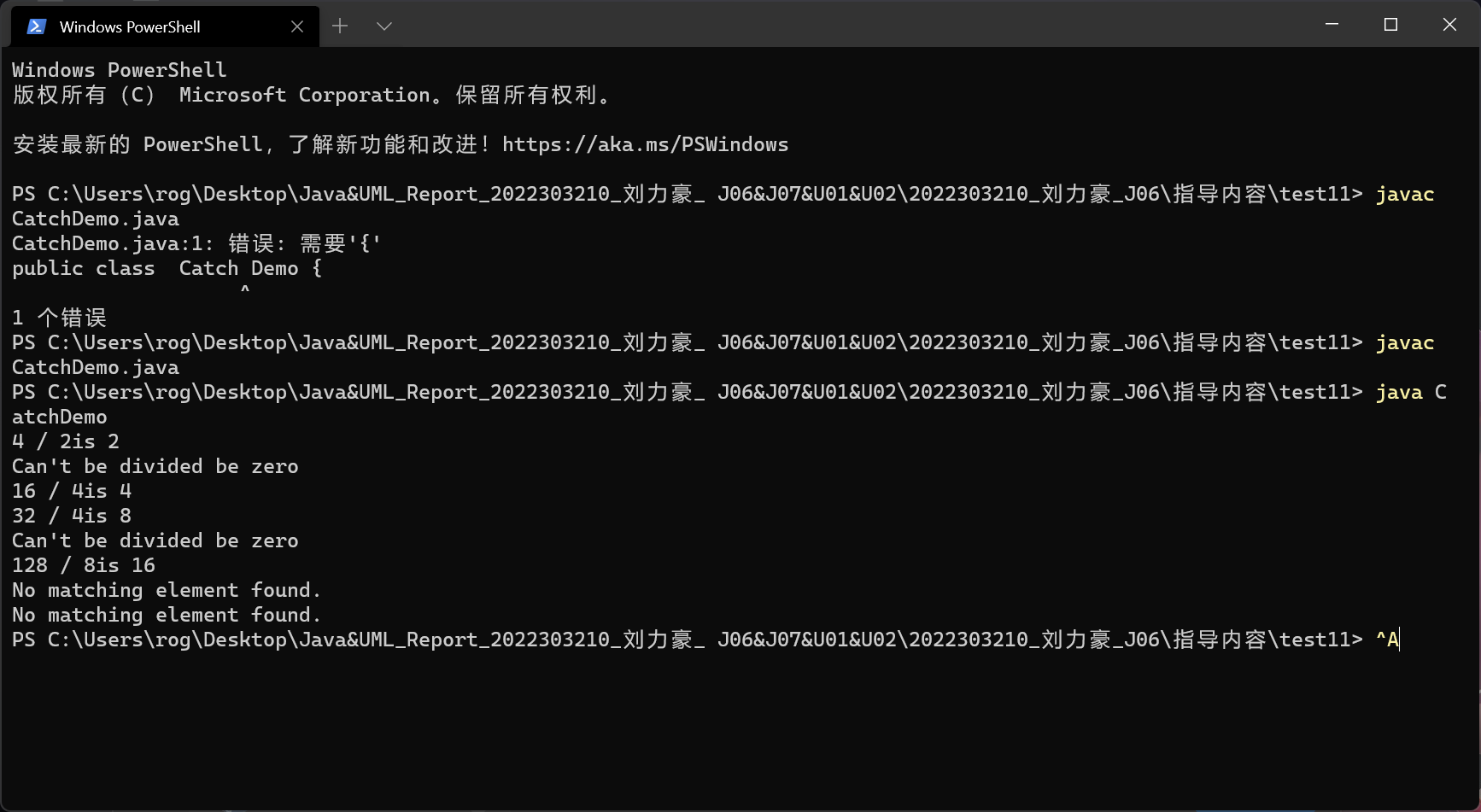
}

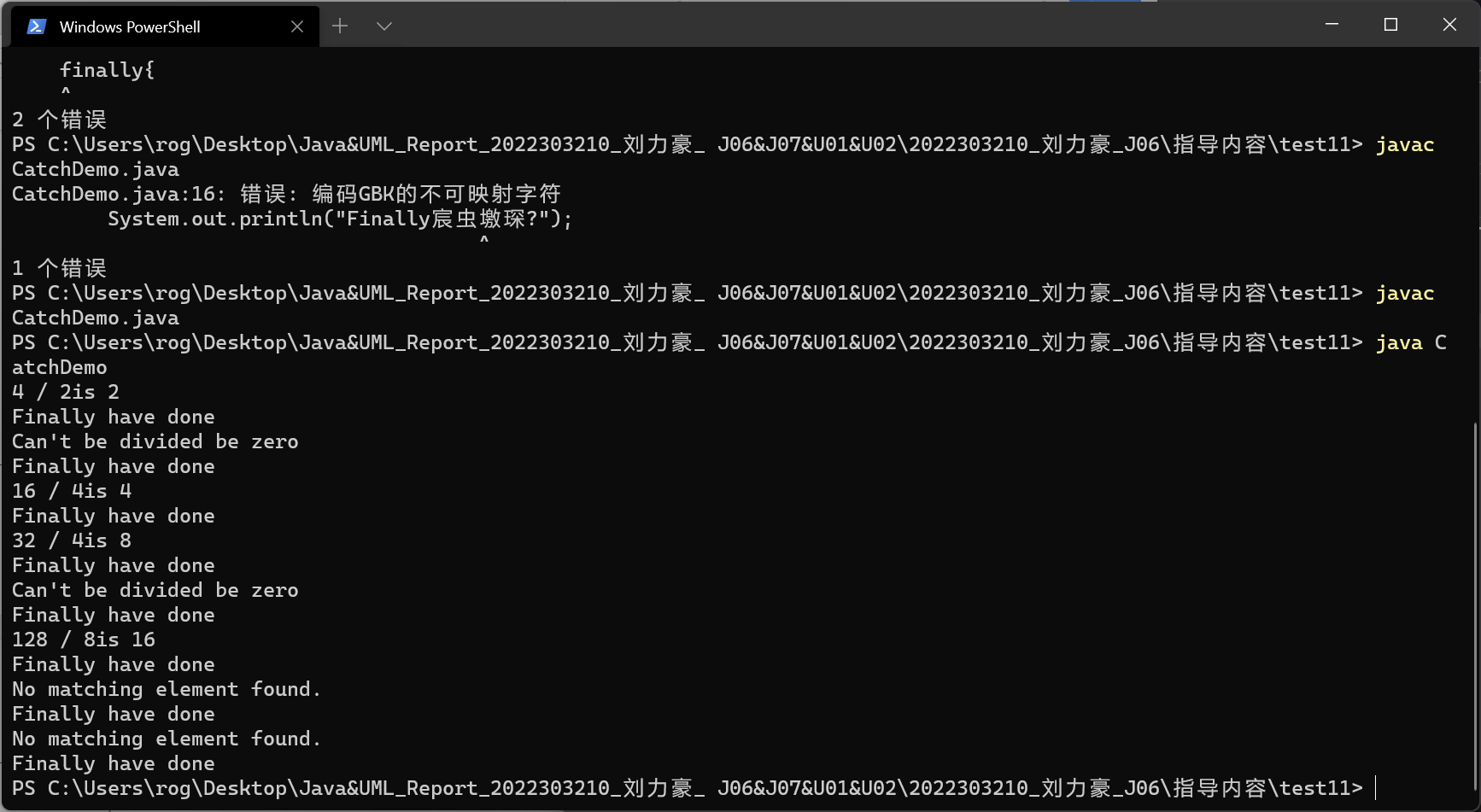
}

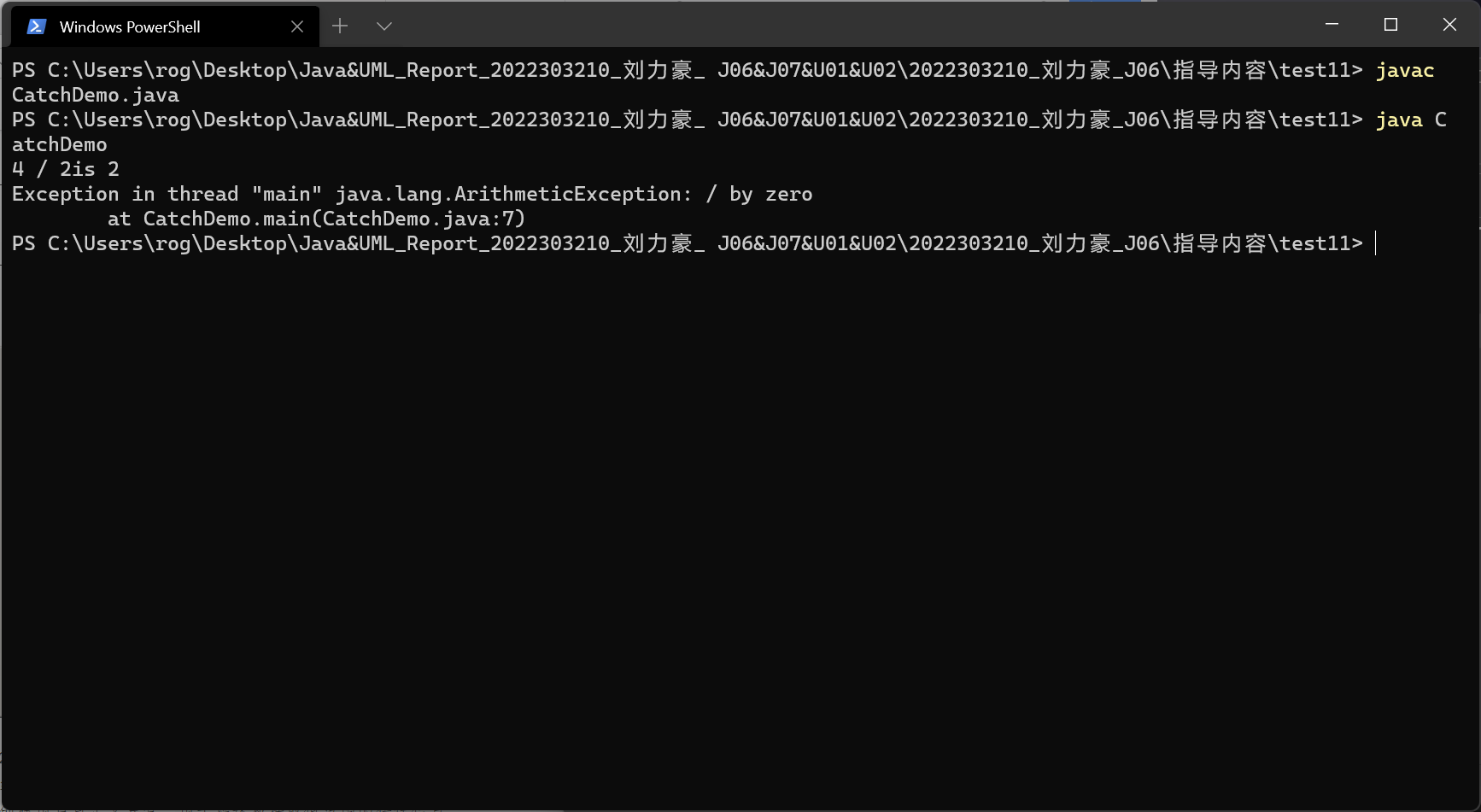
## Java实验结果

### 基本指导：

运行结果：



加finally块后：  


没有异常处理：  


分析原因：

如果没有异常处理，直接输出两个数组对应元素相除的结果，当遇到除数为零时，会抛出ArithmeticException。这会导致程序终止，并在控制台上显示异常堆栈跟踪信息。

原因是当除数为零时，除法运算是不被定义的，因此会引发算术异常。在没有异常处理的情况下，程序无法捕获和处理异常，导致程序崩溃。

代码分析：

异常处理：代码中使用了try-catch语句块来处理可能抛出的异常。在try块中，执行可能引发异常的代码；在catch块中，捕获并处理特定类型的异常。如果抛出的异常与某个catch块匹配，那么对应的代码块将会被执行。

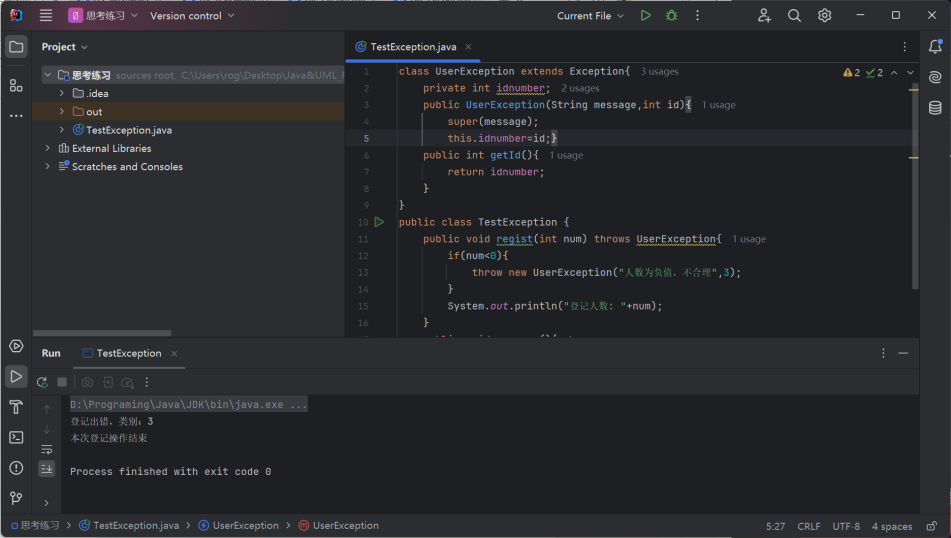
ArithmeticException：在try块中，除法运算number[i] / denom[i]可能会引发ArithmeticException，当除数为零时。在这种情况下，控制流将转到与ArithmeticException匹配的catch块，打印出"Can't be divided by zero"。

ArrayIndexOutOfBoundsException：在try块中，访问数组denom的索引i可能超出其有效范围，这会引发ArrayIndexOutOfBoundsException。在这种情况下，控制流将转到与ArrayIndexOutOfBoundsException匹配的catch块，打印出"No matching element found."。

finally块：代码中添加了一个finally块。finally块中的代码将在try-catch块执行完毕之后无论是否发生异常都会被执行。在这个例子中，finally块打印出"Finally已执行"。

### 思考问题：

运行结果：



问题回答：

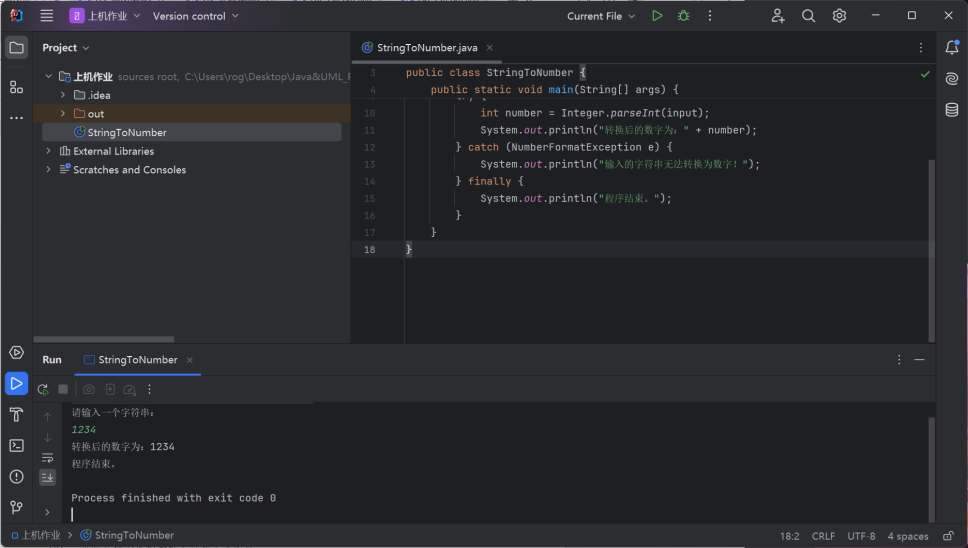
1. 程序中的throws关键字用于声明方法可能抛出的异常，以通知调用该方法的代码需要处理可能的异常情况。在这个例子中，regist方法声明了可能抛出UserException异常。throw关键字用于手动抛出异常。在这个例子中，当num小于0时，使用throw关键字抛出一个UserException异常。
2. 本程序中定义了一个名为UserException的用户自定义异常类。这个类继承自Exception类，表示它是一个异常类。它还有一个私有的idnumber成员变量，用于存储异常的类别。构造函数UserException接受两个参数，一个是异常消息message，另一个是异常的类别id。在构造函数中，通过调用父类的构造函数super(message)来设置异常消息，并将id值赋给idnumber成员变量。还有一个getId方法用于获取异常的类别id。
3. 本程序通过使用try-catch语句块来处理可能产生的用户自定义异常。在manager方法中，调用regist方法，并在try块中执行。如果regist方法抛出了UserException异常，catch块中的代码将被执行，打印出异常的类别id。  
   无论是否发生异常，程序都会继续执行，最后打印出"本次登记操作结束"。

（4）如果将程序中的“public void regist(int num) throws MyException”改为“public void regist(int num) ”，将不再声明可能抛出UserException异常。这意味着调用regist方法的代码不再需要处理可能的异常情况，也不需要使用try-catch语句块来捕获和处理异常。如果在regist方法中抛出了异常，程序将会终止并打印出异常的堆栈跟踪信息。

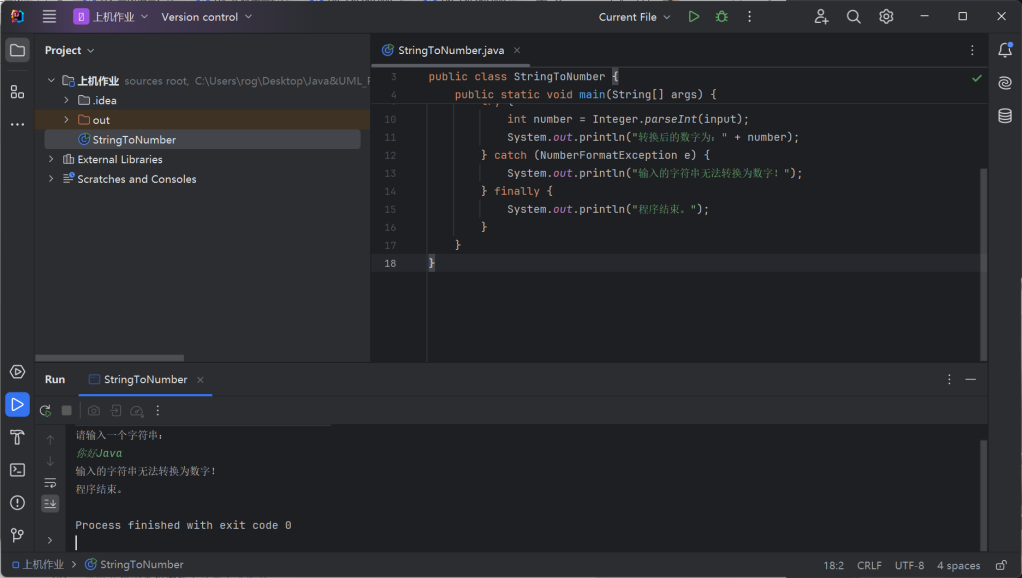
### 上机作业：

运行结果：

正确运行：



错误输入：



代码分析：

1.使用java.util.Scanner类来获取用户输入，创建Scanner对象：通过new Scanner(System.in)创建一个Scanner对象，从标准输入读取用户的输入，

2.使用Integer.parseInt(input)将字符串input转换为整数。如果转换成功，将得到一个整数值，并将其存储在名为number的变量中。

3.处理转换异常：如果字符串无法转换为整数（即包含非数字字符），将抛出NumberFormatException异常。通过catch块捕获该异常，并输出"无法转换（有非数字字符）"的消息。

4.无论转换是否成功，finally块中的代码都会执行。在这里，输出"程序结束。"的消息

# Java实验七 学生信息管理系统

## 实验内容

**题目1．学生信息管理系统**

## Java源代码文件说明

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class Student implements Serializable {

private static final long serialVersionUID = 1L;

private String studentId;

private String name;

private String birthDate;

private String gender;

private String className;

private String major;

public Student(String studentId, String name, String birthDate, String gender, String className, String major) {

this.studentId = studentId;

this.name = name;

this.birthDate = birthDate;

this.gender = gender;

this.className = className;

this.major = major;

}

public String getStudentId() {

return studentId;

}

public String getName() {

return name;

}

public String getBirthDate() {

return birthDate;

}

public String getGender() {

return gender;

}

public String getClassName() {

return className;

}

public String getMajor() {

return major;

}

@Override

public String toString() {

return "Student{" +

"studentId='" + studentId + '\'' +

", name='" + name + '\'' +

", birthDate='" + birthDate + '\'' +

", gender='" + gender + '\'' +

", className='" + className + '\'' +

", major='" + major + '\'' +

'}';

}

}

class StudentManager {

private List<Student> studentList;

public StudentManager() {

studentList = new ArrayList<>();

}

public void addStudent(Student student) {

studentList.add(student);

System.out.println("学生信息添加成功！");

}

public Student findStudent(String studentId) {

for (Student student : studentList) {

if (student.getStudentId().equals(studentId)) {

return student;

}

}

return null;

}

public void deleteStudent(String studentId) {

for (Student student : studentList) {

if (student.getStudentId().equals(studentId)) {

studentList.remove(student);

System.out.println("学生信息删除成功！");

return;

}

}

System.out.println("找不到学号为 " + studentId + " 的学生信息！");

}

public void listAllStudents() {

if (studentList.isEmpty()) {

System.out.println("学生信息为空！");

} else {

for (Student student : studentList) {

System.out.println(student);

}

}

}

public void saveToFile(String filename) {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename))) {

oos.writeObject(studentList);

System.out.println("学生信息保存成功！");

} catch (IOException e) {

System.out.println("保存学生信息失败：" + e.getMessage());

}

}

public void loadFromFile(String filename) {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {

studentList = (List<Student>) ois.readObject();

System.out.println("学生信息加载成功！");

} catch (IOException | ClassNotFoundException e) {

System.out.println("加载学生信息失败：" + e.getMessage());

}

}

}

public class StudentMngMain {

public static void main(String[] args) {

StudentManager studentManager = new StudentManager();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("学生信息管理系统");

System.out.println("1. 添加学生信息");

System.out.println("2. 查询学生信息");

System.out.println("3. 删除学生信息");

System.out.println("4. 列出所有学生信息");

System.out.println("5. 保存学生信息到文件");

System.out.println("6. 从文件加载学生信息");

System.out.println("0. 退出系统");

System.out.print("请输入指令：");

int choice = scanner.nextInt();

scanner.nextLine(); // 读取换行符

switch (choice) {

case 1:

System.out.print("请输入学号：");

String studentId = scanner.nextLine();

System.out.print("请输入姓名：");

String name = scanner.nextLine();

System.out.print("请输入出生日期：");

String birthDate = scanner.nextLine();

System.out.print("请输入性别：");

String gender = scanner.nextLine();

System.out.print("请输入班级：");

String className = scanner.nextLine();

System.out.print("请输入专业：");

String major = scanner.nextLine();

Student student = new Student(studentId, name, birthDate, gender, className, major);

studentManager.addStudent(student);

break;

case 2:

System.out.print("请输入要查询的学号：");

String queryStudentId = scanner.nextLine();

Student queryStudent = studentManager.findStudent(queryStudentId);

if (queryStudent != null) {

System.out.println("查询结果：");

System.out.println(queryStudent);

} else {

System.out.println("找不到学号为 " + queryStudentId + " 的学生信息！");

}

break;

case 3:

System.out.print("请输入要删除的学号：");

String deleteStudentId = scanner.nextLine();

studentManager.deleteStudent(deleteStudentId);

break;

case 4:

studentManager.listAllStudents();

break;

case 5:

System.out.print("请输入要保存到的文件名：");

String saveFilename = scanner.nextLine();

studentManager.saveToFile(saveFilename);

break;

case 6:

System.out.print("请输入要加载的文件名：");

String loadFilename = scanner.nextLine();

studentManager.loadFromFile(loadFilename);

break;

case 0:

System.out.println("感谢使用学生信息管理系统，拜拜~");

System.exit(0);

default:

System.out.println("请重新输入！");

}

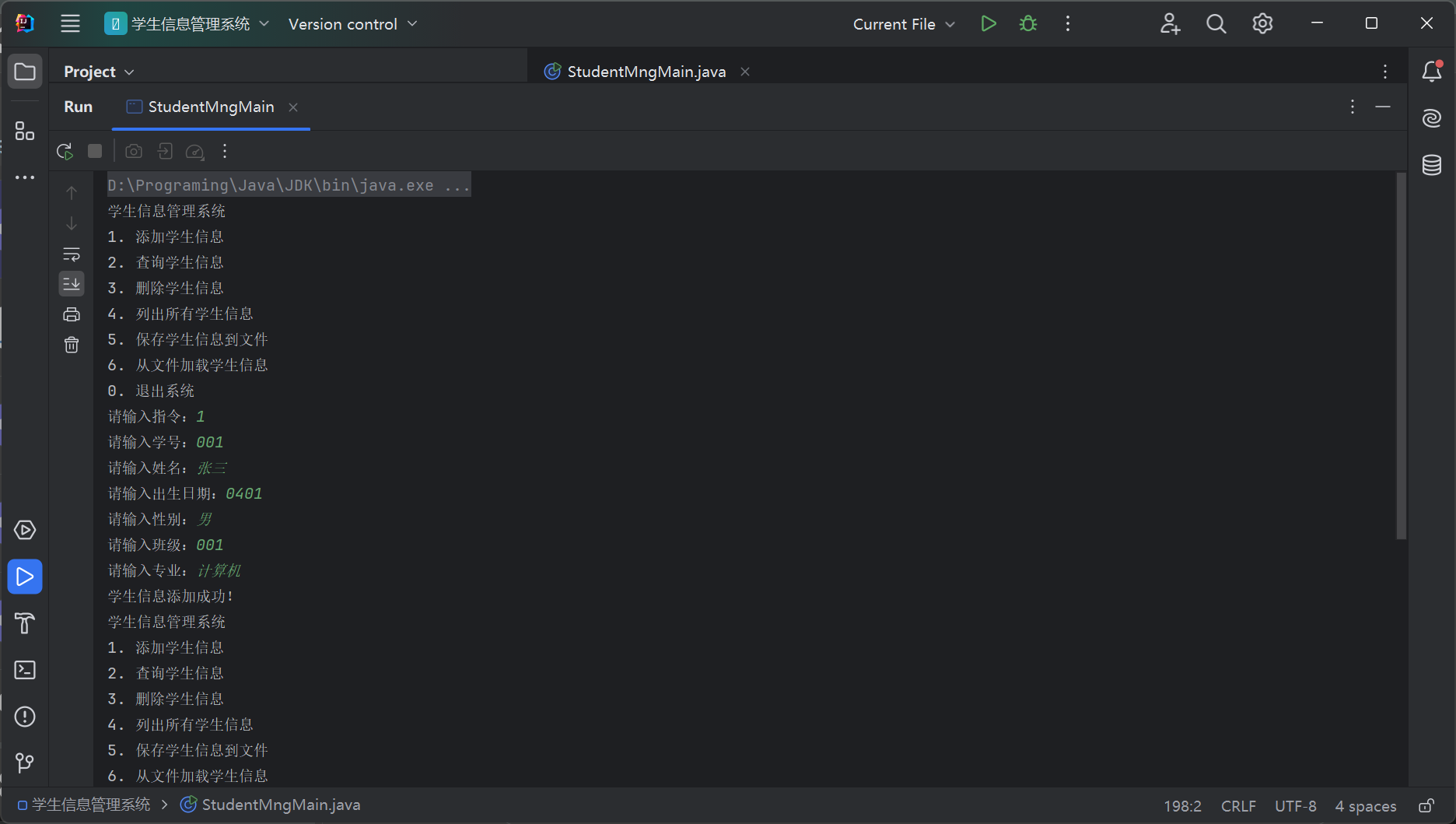
}

}

}

## Java实验结果

运行结果：

  
生成的文件：



代码分析：

Student 类：表示学生对象，具有学号、姓名、出生日期、性别、班级和专业等属性。

StudentManager 类：学生管理器类，用于管理学生信息。它包括以下功能方法：

addStudent：添加学生信息到学生列表。

findStudent：根据学号查找学生信息。

deleteStudent：根据学号删除学生信息。

listAllStudents：列出所有学生信息。

saveToFile：将学生信息保存到文件。

loadFromFile：从文件加载学生信息。

StudentMngMain 类：主程序类，包含 main 方法。它创建了一个 StudentManager 对象和一个 Scanner 对象，实现了一个简单的命令行界面。用户可以通过输入指令来执行相应的学生信息管理操作，包括添加、查询、删除、列出和保存/加载学生信息等。

在主程序的循环中，根据用户输入的指令进行相应的操作，直到用户选择退出系统，通过使用序列化和反序列化将学生信息保存到文件并从文件加载学生信息，以实现数据的持久化存储。

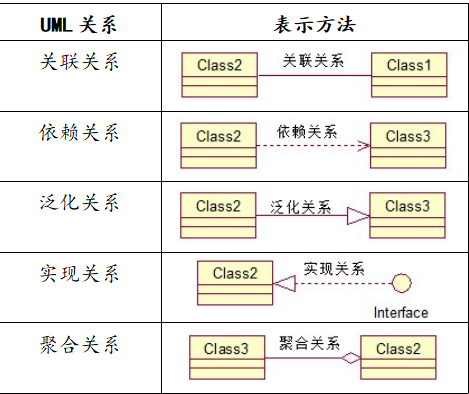
# UML实验一 UML建模基础

## 实验内容

### 1.练习使用建模工具建立各种UML图形，并对图形进行相应编辑和修改。

### 2.认识各种UML关系及可见性符号，并用工具表示出来

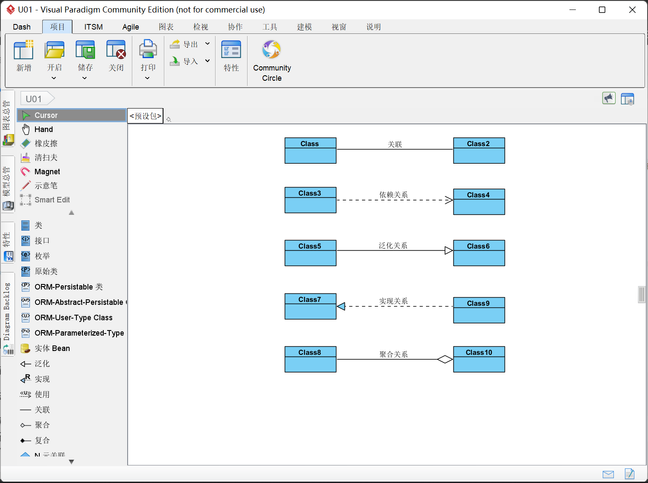
## UML结果文件说明



|  |  |
| --- | --- |
| **可见性** | **表示方法** |
| 公有（Public） | ““+” |
| 私有（Private） | “-” |
| 受保护（Protected） | “#” |
| 包内公有（Package） | “~” |

## UML实验结果

实验运行图：



分析讨论：

UML是一种功能强大的，面向对象的可视化系统分析的建模语言，它的各个模型可以帮助开发人员更好地理解业务流程,建立更可靠，更完善的系统模型。从而使用户和开发人员对问题的描述达到相同的理解，以减少语义差异，保障分析的正确性。

通过UML建模，可以达到4个目的：①模型有助于按照实际情况或按照所需要的样式对系统进行可视化；②模型能够规约系统的结构或行为；③模型给出了指导构造系统的模板；④模型对做出的决策进行文档化。

建模是一项经过检验并被广为接受的工程技术，模型提供了系统的蓝图。模型可以是结构性的，强调系统的组织。它也可以是行为性的，强调系统的动态方面。

随着软件系统越来越复杂，要求我们必须实现从需求到详细设计的模型化规划，这是一个非常重要的环节，后续的编码、测试、维护都只是按照前面的规划计划进行，一个非常详细的设计，利用UML几乎能将整个系统架构和代码架构通过UML表示出来，不但简化了复杂系统的开发难度，也规范了系统的软件开发过程，使系统更加可控、可靠、可实现。

因此在软件工程中UML建模是非常重要并且必要的。

# UML实验二 用例图

## 实验内容

### 画出用例图：

某大学的图书馆系统，学生可以使用该系统进行书籍的检索、借出、归还。借出时学生还可以根据需要查看自己已借的图书的一览。另外，在进行检索、借出、归还的时候需要通过学生信息系统认证学生信息是否是有效的。

## UML结果文件说明

### 1.什么是用例，什么是场景？用例和场景之间的关系是怎样的？

解：场景是用例的具体实例化，它描述了在特定条件下系统的行为和交互。场景提供了对用例的具体执行过程的描述，包括用户与系统之间的交互步骤、输入输出以及预期结果。

关系：用例是对系统功能的抽象描述，而场景是对用例的具体实例化描述。用例可以有多个场景，每个场景描述了用例在不同情况下的执行过程

### 2.用例图中有哪些组成元素？在UML中是如何表示的？

解：

椭圆形表示的参与者：表示与系统进行交互的外部实体，可以是用户、其他系统或设备等。

方框形表示的用例：表示系统提供的功能或服务。

关联线：连接参与者和用例，表示参与者与用例之间的交互关系。

包含关系：用于表示一个用例包含另一个用例的功能或行为。

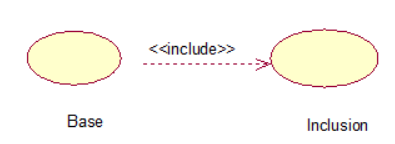
扩展关系：用于表示一个用例可以扩展另一个用例的功能或行为。

泛化关系：用于表示用例之间的继承关系，其中一个用例是另一个用例的特化。

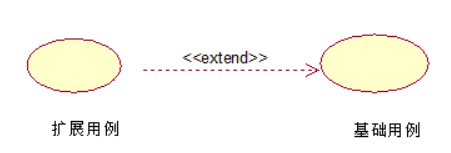
### 3.用例与用例之间的包含关系、扩展关系和泛化关系各代表什么含义？它们之间有何区别？对以上三种关系各举一例，画出用例图，并进行说明。

解：

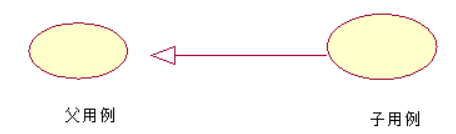
1. 包含关系:一个用例包含另一个用例的功能或行为。被包含的用例是基础用例的一部分，用例在执行过程中会调用被包含的用例。被包含的用例是可选的，可以在基础用例的不同条件下进行包含。



1. 扩展关系:一个用例可以扩展另一个用例的功能或行为。这意味着扩展用例提供了基础用例的可选功能，可以根据特定条件触发。扩展关系使用扩展点指定扩展的位置。



1. 泛化关系:用例之间的继承关系，其中一个用例是另一个用例的特化。这意味着子用例继承了父用例的功能和行为，并可能添加、修改或覆盖一些特定的行为。泛化关系用于建立用例的继承层次结构。



4.

(1)但小李认为该模型不符合“用例建模”的思想，存在明显的错误。说明错误所在，并说明应该如何修改。

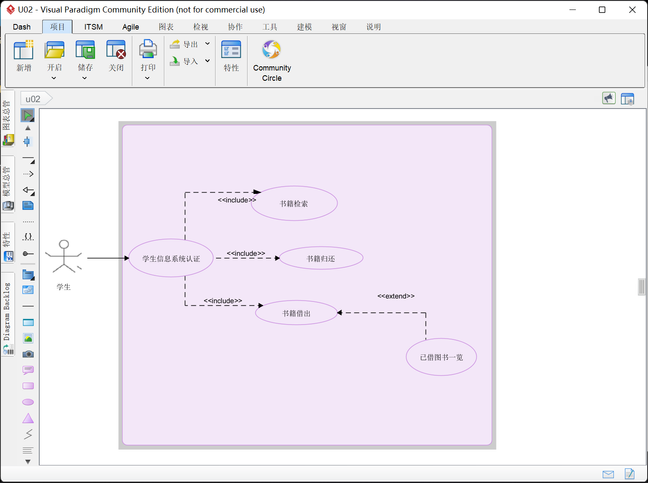
(2)在上图中构造型“《include》”表示的是什么意思，它与“《extend》”之间的区别是什么？

- (1)解:用例图首先没有绘制出信息管理系统的系统边界；参与者和用例的关联应当是实线不带箭头表示；用例之间的关系存在错误，应当是打开房源信息包含了{登录信息，录入信息{确认提交信息}，修改信息}；存在用例之间逻辑问题的构建漏洞；

- (2)解：<<indclude>>是指用例的包含关系，由源用例引出虚线箭头指向目标（被包含）的用例，表示两个用例的包含关系；而<<extend>>可视为拓展关系，拓展关系链接拓展用例和基用例，不能单独出现，反方向看很像包含关系

## UML实验结果

### 运行图



在图书管理系统用例图中，学生属于用户在系统外，学生通过认证进入系统

系统包括了3个功能即：检索，借出，归还，同时借出时还能使用拓展功能已借的图书的一览。

4.

### 5.分析与讨论：

(1) 用例图的重要作用：

- 用于捕捉和分析系统的功能需求：用例图可以帮助识别系统的功能需求，并以图形化的方式展示这些需求之间的关系和交互。

- 用于沟通和共享需求：用例图提供了一种可视化的方式来描述和传达系统需求，让开发团队和利益相关者之间更好地理解和共享需求。

- 用于指导系统设计和开发：用例图可以作为系统设计的基础，指导开发团队进行系统的架构设计和模块划分。

- 用于验证和验证系统需求：用例图可以用于验证系统需求的完整性和一致性，帮助发现和解决潜在的问题和冲突。

用例图适用的场合：

- 用例图适用于各种软件开发项目，尤其是面向对象的系统。以下是一些适合使用用例图的场合：

- 需求分析阶段：用例图可用于识别和分析系统的功能需求，帮助捕捉用户需求和业务流程。

- 系统设计阶段：用例图可用于指导系统架构设计和模块划分，帮助确定系统的功能模块和它们之间的关系。

- 与利益相关者沟通：用例图是一种易于理解的图形化工具，可以帮助开发团队与利益相关者共享和讨论需求，促进沟通和理解。

- 验证系统需求：用例图可以用于验证和确认系统需求的完整性和一致性，帮助发现和解决潜在的问题和冲突。

(2) 用例分析技术和结构化分析之间的关系和区别：

关系和区别：

- 用例分析技术和结构化分析都是软件工程中的需求分析方法，用于分析和描述系统的功能需求。

- 用例分析技术强调系统的功能和用户交互，以用例图和用例规约为主要工具，从用户的角度出发，描述系统的功能需求，以驱动系统的设计和开发。

- 结构化分析强调系统的结构和行为，以数据流图和数据字典为主要工具，从数据的角度出发，描述系统的数据流和处理过程，以驱动系统的设计和开发。

- 用例分析技术和结构化分析可以互补使用，用例分析技术关注系统的功能需求和用户交互，而结构化分析关注系统的结构和行为，两者可以结合使用来完整地描述和分析系统的需求。

(3) 使用用例图时应该注意以下问题：

- 用例的粒度：用例应该具有适当的粒度，即不过于细化也不过于笼统。每个用例应该描述一个有意义的用户操作或系统功能，以便更好地理解和管理系统的需求。

- 用例的边界：用例应明确定义参与者和系统之间的交互边界。确保每个用例都有明确的参与者，并清楚界定了系统对参与者的响应和输出。

- 用例的关系：用例之间的关系应该准确反映系统的功能和用户交互。根据实际需求，使用适当的关系（如包含关系、扩展关系、泛化关系等）来表示用例之间的依赖关系和关联关系。

- 用例的命名和描述：用例的命名应该清晰、简洁，并能准确地描述用例的功能。用例的描述应该具备清晰的语义，准确地描述用户操作和系统响应的流程和条件。

- 用例的优先级和复杂性：根据需求的重要性和紧急程度，为每个用例分配适当的优先级。同时，评估每个用例的复杂性，确保用例的规模和复杂性在可控范围内。

# 实验总结

## 实验中遇到的问题以及解决方法

1.问题：实现关系不在类图的一级面板里

解决方法：与同学交流软件使用心得，在泛化-实现关系中找到

1. 问题：绘图软件的汉化问题

解决方法：在CSDN上搜索汉化方发，成功汉化

1. 问题：在Java复杂程序编程时，不使用idea编译会出现中文无法编译情况

解决方法：在终端中输入 Javac encodeing UTF-8 文件.java

即可将.java文件以UTF-8字符编码格式编译，实现汉字编译成功

**2 实验体会**

## 实验体会

通过完成Java实验，我学到了以下几点：我理解了如何定义类、创建对象，并且学会了如何使用对象的属性和方法。这个实验让我熟悉了集合类的基本用法，如添加元素、遍历集合等。。我学会了如何使用ObjectOutputStream和ObjectInputStream来将对象写入文件和从文件中读取对象。我学到了如何使用Scanner类来获取用户输入，并且掌握了使用条件语句和循环语句来控制程序的流程。通过实验，我学会了如何使用try-catch块来捕获和处理异常，以及如何给用户提供有用的错误信息。

我对Java编程语言的基础知识有了更深入的理解，掌握了一些常用的编程技巧和实践经验。我意识到面向对象编程的优势，以及如何将其应用于实际问题的解决。此外，我还加强了对文件操作和异常处理的认识，这对于开发更复杂的应用程序非常重要。

总的来说，这个实验提供了一个很好的平台，让我能够通过实践来巩固和应用所学的知识。它不仅加深了我对Java编程的理解，也培养了我的问题解决能力和编程实践经验。通过这个实验，我也更加自信地迈出了继续学习和探索Java编程世界的步伐。

而通过绘制UML类图和用例图，真正动手进行类图的操作，发现这个东西很实用，之前看工程的代码的时候，代码多且复杂，和容易看着看着就找不到之前的变量和函数在哪个文件中了，可以利用UML画出其类图，并表示出其之间的关系，会更加简洁和方便。