《Java 企业级应用》实验报告

年级、专业、班级	2021 级计卓 1 班	姓名	李宽宇	学号	20215279
实验题目	基于命令行的学生成绩管理系统				
实验时间	2024.3.16	实验地点	DS3401		
学年学期	2023-2024(2)	实验性质	□验证性 ■设计性 □综合性		

一、实验目的

1. 本次实验的目的是掌握 Java 企业级应用开发工具的使用方法，掌握 Java 面向对象程序编程技术，掌握常用工具类的使用。理解和使用设计模式。
2. 设计开发完成一个基于命令行的软件。
3. 抄袭计 0 分。

二、实验项目内容

设计开发完成基于命令行的学生成绩管理系统。

要求如下：

1、创建类实现基本对象和他们关系的管理。包括学生、教学班、课程、成绩、教师等。学生至少包含学号、姓名、性别等信息。教学班至少包含教师、课程名字、总人数、教学班号、开课学期等信息。课程至少包含课程编号、课程名字等信息。教师至少包含教师编号、姓名等信息。可以根据自己的分析设计增加其他类。

2、随机生成学生，数量不少于 100。课程数量不少于 3 门。一个教学班有一个教师上一门课程，一个教学班的学生数量不少于 20。教师数量不少于 6 个。一门课至少有两个老师上课。每个学生至少选择 3 门课程。一个学生在一个教学班上一门课，考试后取得一个成绩。一门课的成绩构成有 4 部分构成，包括平时成绩、期中考试、实验成绩和期末考试成绩，然后计算出综合成绩。自定义各项成绩的产生策略，均为整数。

3、给出一个主菜单，选择菜单项执行各项功能。例如执行一个菜单命令，可以生成一个教学班的所有学生的平时成绩。分阶段模拟教学过程，第一步，生成初始化数据，包括教师，学生、课程，教学班等；第二步，学生选课，自定义选课策略，为每门课程的教学班安排学生。第三步，获得平时成绩，期中成绩，实验成绩，期末成绩，最后计算综合成绩，要记录成绩取得的时间。

4、能够格式规范地显示一个教学班的学生的成绩，可以根据学号排序，可以根据成绩排序。可以统计学生各科、总成绩的分数段分布。可以通过名字或者学号查询一个学生的所有科目的成绩和总成绩。可以按照学号、各科

成绩和总成绩对所有学生进行排名显示。

5、可以实现自己的扩展功能或自定义功能。注意操作使用的方便性，注意类和类之间的关系。充分利用继承，多态等特性，使用上抽象类，接口，泛型，内部类等设计元素，使用好集合类。注意程序的总执行流程和分支执行流程。注意设计思想的表达，注意优化代码结构，优化类的职责分工，注意使用设计模式。代码有注释。

6、在报告中注明自己的创新点、特色等。

7、提交：（1）本实验报告，（2）源代码压缩文件 zip，（3）软件演示的 MP4 视频，视频大小不超过 20M，视频请在**搜狗浏览器**测试能否正常播放。注意源代码加注释。注意文件名称的规范性。文件名：学号姓名 1.docx，学号姓名 1.zip，学号姓名 1.mp4。**三个文件分别提交**。

三、实验过程或算法（写明创新点或特色、设计思想、设计模式的使用、程序的结构、功能关系图、类的说明和类之间的关系图、程序主要执行流程图，最后是核心源代码，截图等）

1. 创新点或特色

(1) 仿照关系型数据库的构建

(2) 前后端分离,各类职责明确单一，做到了高内聚、低耦合

(3) 继承、多态，接口、泛型、内部类、反射、修饰注解。详细见源代码部分。

其中，表格绘制为核心创新点，采用了接口（绘图接口）、泛型、反射、修饰注解来实现

传入类（泛型）->反射获取函数和函数名->修饰注解排序函数名生成表头->调用类的 get 函数获取数据->绘制表格

(4)使用集合框架 HashMap、日期类 LocalDate，丰富命令行输出的包 jansi，switch 表达式，数据处理采用大量的构造匿名对象、Lambda 表达式、Stream。充分利用异常处理。

（5）数据可视化

图 1：班级成绩柱状图

ClassRank	StudentId	StudentName	ClassName	AttendanceGrade	MidTermGrade	LabGrade	FinalGrade	ComprehensiveGrade	ComprehensiveGradeRecordedDate
36	1105	Charlie	Biology	83.0	90.0	69.0	36.0	69.5	2024-03-23
26	1105	Charlie	Physics	88.0	99.0	91.0	38.0	79.0	2024-03-23
21	1105	Charlie	Chemistry	87.0	90.0	89.0	51.0	79.25	2024-03-23

总成绩: 75.92 ,　已修 3 门课

图 2：个人成绩表

2. 设计思想

首先，数据存储方面，仿照关系型数据库的构建思路，优化类的职责分工。将学生、教学班、课程、成绩、教师设计成实体类，学生与课程班、老师与课程班、课程班与课程设计成关系类，学生成绩作为“学生与课程班”这一关系类的内部类。这样设计使得数据结构清晰：关系型数据库采用了表格形式来存储数据，每个表格都有固定的列和行，这种结构非常清晰易懂。

这样可以更好地管理和维护数据。由于关系型数据库采用了模块化和分层结构，因此可以很容易地进行扩展。

其次，前后端分离的设计思路，前端设计了命令行界面类，后端仿照关系型数据库进行构建，界面类调用封装好的函数对数据进行操作，使得程序呈现出高内聚、低耦合的特点。

再次，注重程序的实用性与操作使用的方便性，实现自己的扩展功能。（1）拓展了对所有数据的查询，自定义初始化（如指定学生数目），在模拟过程后可以选择修改数据（如为指定学生赋分），绘制表格和柱状图对数据进行可视化展示。（2）为方便用户操作，减少用户记忆，用户只需选择提供的选项即可以选择功能，班级编号和学号均支持查询。流程上：程序的总执行流程和分支执行流程。

然后，程序设计采用了多种先进的技术，包括但不限于继承、多态、接口、泛型、内部类、反射、修饰注解；使用多种工具，包括集合框架 HashMap、日期类 LocalDate，丰富命令行输出的包 jansi，switch 表达式，数据处理采用大量的构造匿名对象、Lambda 表达式、Stream。充分利用异常处理，以增强程序的鲁棒性和稳定性，例如反射调用的函数用 throws InvocationTargetException。使用 maven 管理项目。

此外，从代码的规范角度，包名:全小写;类名:首字母大写,每个单词的首字母大写;方法名:小写字母开头,每个单词的首字母大写;变量名: 写字母开头,每个单词的首字母大写;常量名:基本类型的常量名全大写。

最后是设计模式方面，采用了桥接（Bridge）、装饰器模式（Decorator Pattern）的设计模式，满足里氏替换原则，一些类满足单一职责原则。

3. 设计模式的使用

桥接（Bridge）是用于把抽象化与实现化解耦，使得二者可以独立变化。这种类型的设计模式属于结构型模式。

在设计绘制表格工具类时，采用了这一设计模式，目的是将抽象与实现分离，使它们可以独立地变化， 将绘制表格的 draw()抽象出来成为接口。符合依赖倒置原则。

```
1 usage   1 implementation
public interface Drawable {
    12 usages   1 implementation
    void draw();
}
```

图 3：绘制接口

此外，在设计时还满足里氏替换原则，代码共享，减少创建类的工作量，每个子类都拥有父类的属性和方法；提高代码的重用性；子类可以形似父类，但又异于父类。

装饰器模式(Decorator Pattern)允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。

在设计表格列的顺序时，采用了这一模式。由于表头时根据反射得到的函数名确定的，但 java 反射机制没有明确的返回顺序，定义修饰器修饰函数即可。

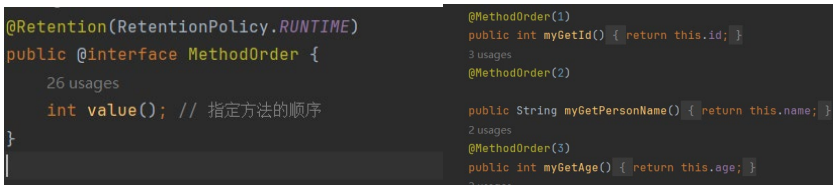


图 4：MethodOrder 的实现与使用

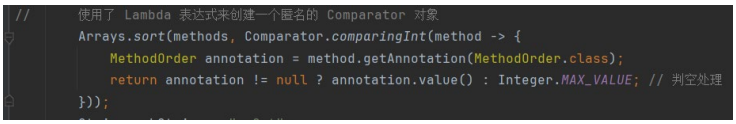


图 5：如何通过 MethodOrder 排序反射的函数

4. 程序的结构

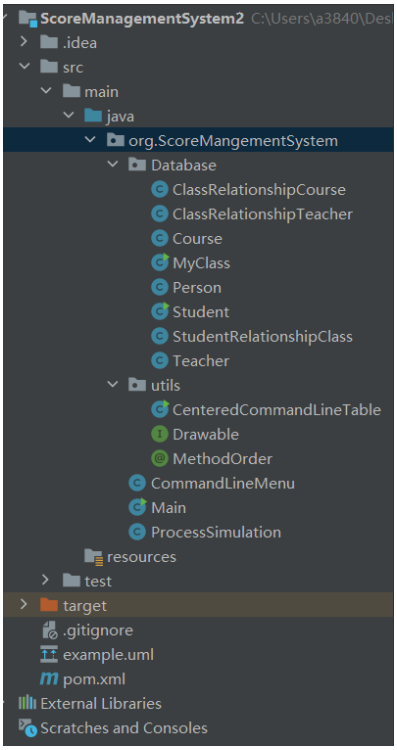


图 6：程序结构图

5. 功能关系图

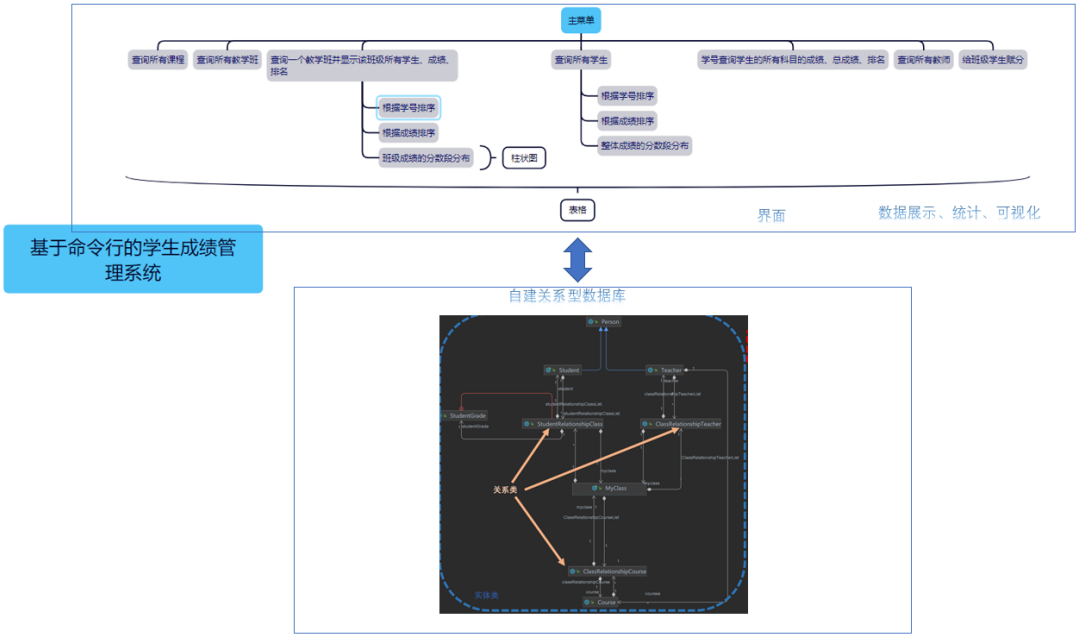


图 7：功能关系图

6. 类的说明和类之间的关系图

（1）实体类

Person:包括 id,姓名,年龄等人的基本信息,是 Student 和 Teacher 的父类。

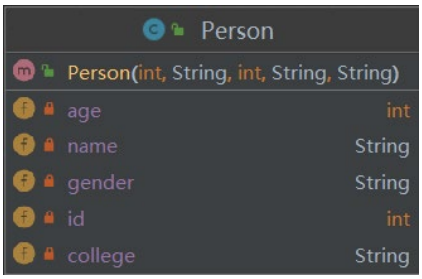


图 8：Person 类的成员变量

Student:包括 studentId,年级等学生的基本信息,此外还包括涉及关系类的外键列表，是 Person 的子类。

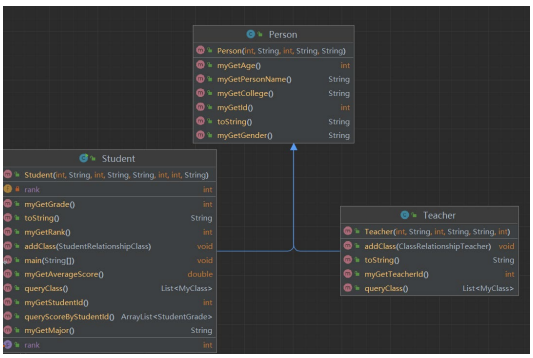


图 9:Student 和 Person 类

Teacher:包括 teacherId 等老师的基本信息，此外还包括涉及关系类的外键列表，是 Person 的子类。

MyClass:教学班的实体类,包括课程名字、总人数、教学班号、开课学期等信息。

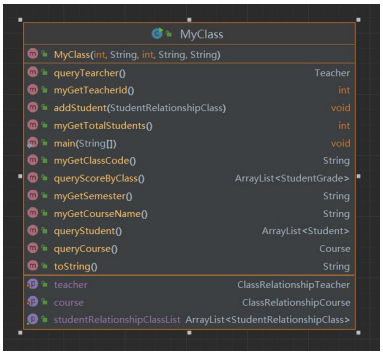


图 10：MyClass 类

Course：课程包含课程编号、课程名字等信息。

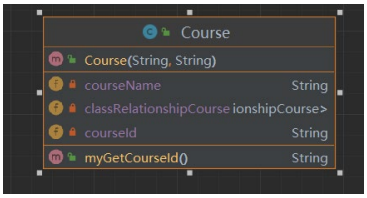


图 11：Course 类

（2）关系类

ClassRelationshipCourse：一对一关联教学班与课程。

ClassRelationshipTeacher：一对一关联教学班与教师。

StudentRelationshipClass：一对一关联教学班与学生，同时存储学生的成绩。

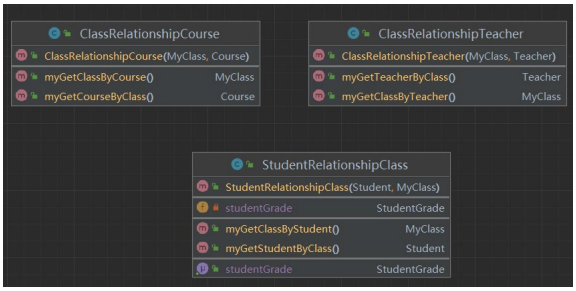


图 12：关系类

（3）工具类

CenteredCommandLineTabel<T>:绘制表格的工具类，利用泛型+反射机制绘制不同的表格。

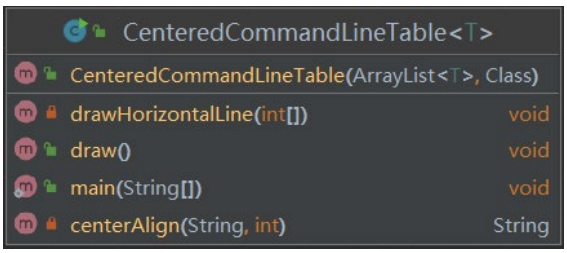


图 13：CenteredCommandLineTabel<T>类

（4）接口

MethodOrder：@Retention，需要通过反射来获取运行时注解，使得表格列的顺序固定。

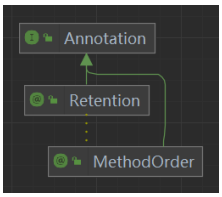


图 14：MethodOrder 接口

Drawable：为绘制表格实现了对外的绘图接口

```
1 usage   1 implementation
public interface Drawable {
    12 usages   1 implementation
    void draw();
}
```

图 15：Drawable 接口

（5）界面类

CommandLineMenu：打印界面,包括提示信息和输出查询结果。

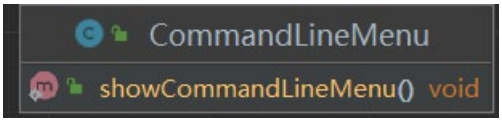


图 16：CommandLineMenu 接口

（6）类之间的关系图

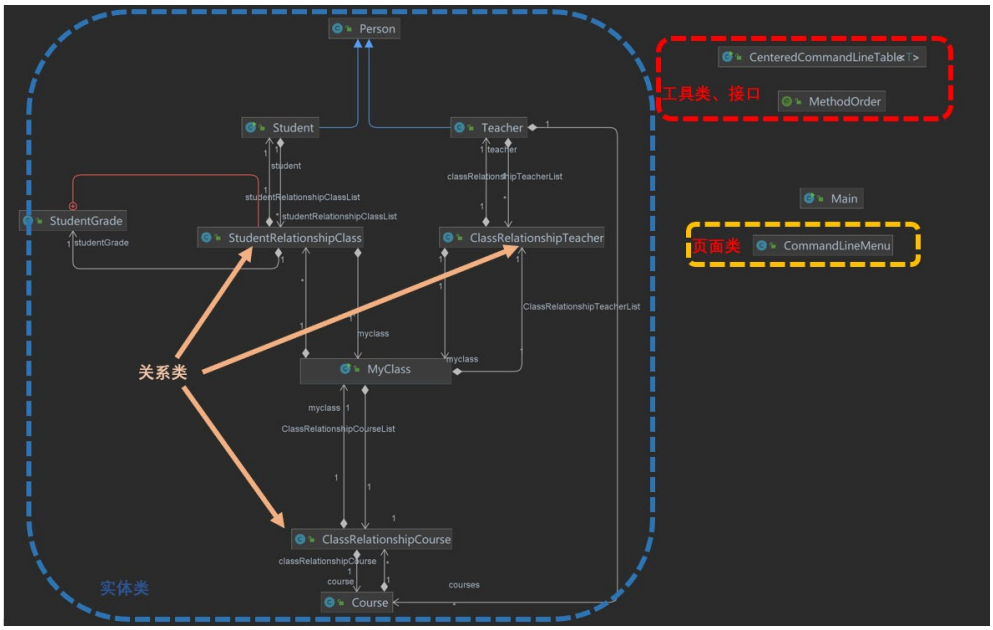


图 17：类之间的关系图

7.程序主要执行流程图

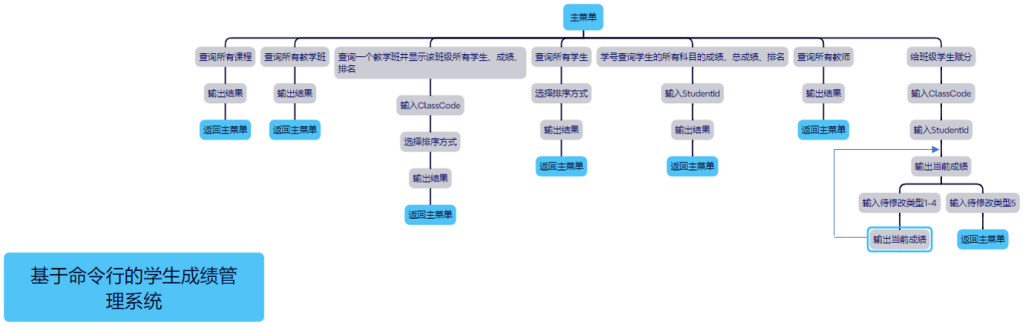


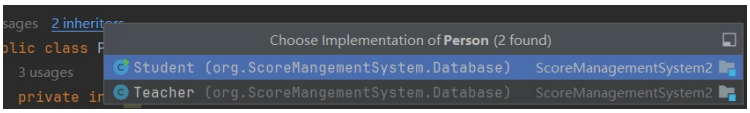
图 18：程序主要执行流程图

8. 核心源代码、截图

（1）继承、多态：Student 和 Teacher 均继承了 Person 类

```
3 usages  2 inheritors
public class Person {
    3 usages
    private int id;
    3 usages
    private String name;
    3 usages
    private int age;

    3 usages
    private String gender;
    3 usages
    private String college;
```



点击 2 inheritor 可以查看继承的情况。

（2）接口：代码实现了两个接口 Darwble 和 MethodOrder

```
package org.ScoreMangementSystem.utils;

1 usage  1 implementation
public interface Drawable {
    12 usages  1 implementation
    void draw();
}
```

```
@Retention(RetentionPolicy.RUNTIME)
public @interface MethodOrder {

    int value(); // 指定方法的顺序
}
```

Darwble 接口 CenteredCommandLineTable 实现 draw(),将抽象与实现分离，使它们可以独立地变化， 将绘制表格的 draw()抽象出来成为接口。符合依赖倒置原则。

```
public class CenteredCommandLineTable<T> implements Drawable
```

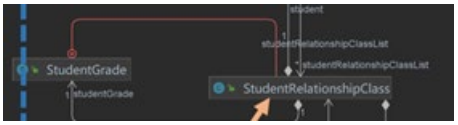
MethodOrder:见本节(6)修饰注解

(3)泛型：CenteredCommandLineTable 可以接收泛型的对象绘制表格，换言之，CenteredCommandLineTable 将 ArryList<T>输出为表格，简化了代码的可视化。构造函数如下：

```
public CenteredCommandLineTable(ArrayList<T> arrayList, Class typeClass) throws InvocationTargetException, IllegalAccessException {
    this.typeClass = typeClass;
      System.out.println("泛型参数类型: " + typeClass.getName());
    Method[] methods = typeClass.getMethods();
    使用了 Lambda 表达式来创建一个匿名的 Comparator 对象
    Arrays.sort(methods, Comparator.comparingInt(method -> {
        MethodOrder annotation = method.getAnnotation(MethodOrder.class);
        return annotation != null ? annotation.value() : Integer.MAX_VALUE; // 判空处理
    }));
    String subString = "myGet";
    int indexI = 1;
    int indexJ = 0;
    this.rows = arrayList.size()+1;
    this.columns = 0;
    for (Method method : methods) {
        if (method.getName().contains(subString)) {
            columns++;
        }
    }
    this.data = new String[rows][columns];
    for (Method method : methods) {
        if (method.getName().contains(subString)) {
            data[0][indexJ] = method.getName().replace(subString,  replacement: "");
            indexJ++;
        }
    }
```

```
    for (T t : arrayList) {
        indexJ = 0;
        for (Method method : methods) {
            if (method.getName().contains(subString)) {
                Object[] parameters = {};
                data[indexI][indexJ] = method.invoke(t, parameters).toString();
                indexJ++;
            }
        }
        indexI++;
    }
}
```

(4)内部类：在关系型数据库设计中，关系类常常可能包含一些属性，例如，学生某一门课的成绩——学生实体与课程（班）实体的关系下的 1 条属性。成绩包括平时成绩、实验成绩、成绩计算、更新日期等属性方法。如果只是作为学生课程班关系类的若干属性和方法，明显是不合适的，且这些属性和方法高度聚合。所以，这里通过在关系类 StudentRelationshipClass 中创建子类 StudentGrade。



调用：在类内部调用 StudentGrade

```
public class StudentRelationshipClass {
    3 usages
    private Student student;
    3 usages
    private MyClass myclass;

    2 usages
    private StudentGrade studentGrade;
```

在外边调用 StudentGrade

```
System.out.println("当前学生成绩：");
ArrayList<StudentRelationshipClass.StudentGrade> selectedGrade = new ArrayList<>();
for(StudentRelationshipClass.StudentGrade studentGrade:processSimulation.classCodeMap.get(ClassCode).queryScoreByClass()){
    if (studentGrade.myGetStudentId()==studentid){
        selectedGrade.add(studentGrade);
        break;
    }
}
```

(5)反射

在使用了泛型后，为了使表格绘制更加灵活，使用反射机制。流程如下：传入类（泛型）->反射获取函数和函数名->修饰注解排序函数名生成表头->调用类的 get 函数获取数据->绘制表格。

反射获取函数和函数名

```
public CenteredCommandLineTable(ArrayList<T> arrayList, Class typeClass)
    this.typeClass = typeClass;
      System.out.println("泛型参数类型: " + typeClass.getName());
    Method[] methods = typeClass.getMethods();
```

修饰注解排序函数名

```
使用了 Lambda 表达式来创建一个匿名的 Comparator 对象
Arrays.sort(methods, Comparator.comparingInt(method -> {
    MethodOrder annotation = method.getAnnotation(MethodOrder.class);
    return annotation != null ? annotation.value() : Integer.MAX_VALUE; // 判空处理
}));
```

通过函数名过滤特殊的方法，关键字：myGet，用于确定要调用哪些函数

```
String subString = "myGet";
int indexI = 1;
int indexJ = 0;
this.rows = arrayList.size()+1;
this.columns = 0;
for (Method method : methods) {
    if (method.getName().contains(subString)) {
        columns++;
    }
}
```

利用反射回来的函数名生成表头

```
this.data = new String[rows][columns];
for (Method method : methods) {
    if (method.getName().contains(subString)) {
        data[0][indexJ] = method.getName().replace(subString,  replacement: "");
        indexJ++;
    }
}
```

invoke 调用类的过滤后的反射出来的函数获取数据

```
for (Method method : methods) {
    if (method.getName().contains(subString)) {
        Object[] parameters = {};
        data[indexI][indexJ] = method.invoke(t, parameters).toString();
        indexJ++;
    }
}
```

(6)修饰注解：

MethodOrder:在设计表格列的顺序时使用。由于表头时根据反射得到的函数名确定的，但 java 反射机制没有明确的返回顺序，定义修饰器修饰函数即可。

```java
@Retention(RetentionPolicy.RUNTIME)
public @interface MethodOrder {
    26 usages
    int value(); // 指定方法的顺序
}
```

```java
@MethodOrder(1)
public int myGetId() { return this.id; }
3 usages
@MethodOrder(2)

public String myGetPersonName() { return this.name; }
2 usages
@MethodOrder(3)
public int myGetAge() { return this.age; }
```

通过 MethodOrder 排序反射的函数

```java
//      使用了 Lambda 表达式来创建一个匿名的 Comparator 对象
        Arrays.sort(methods, Comparator.comparingInt(method -> {
            MethodOrder annotation = method.getAnnotation(MethodOrder.class);
            return annotation != null ? annotation.value() : Integer.MAX_VALUE; // 判空处理
        }));
```

(7)集合框架 HashMap，通过主码查找对象更快

```java
    13 usages
    public Map<String,MyClass> classCodeMap = new HashMap<>();
    6 usages
    public Map<Integer, Student> studentIdMap = new HashMap<>();
```

(8)日期类 LocalDate

声明举例

```java
3 usages
private LocalDate comprehensiveGradeRecordedDate;
```

类内部使用

```java
public void recordGrades(int category, double grade) {
    switch (category){
        case 1: this.attendanceGrade = grade; attendanceGradRecordedDate=LocalDate.now();break;
        case 2: this.midTermGrade = grade; midTermGradeRecordedDate=LocalDate.now();break;
        case 3: this.labGrade = grade; labGradeRecordedDate=LocalDate.now();break;
        case 4: this.finalGrade = grade; finalGradeRecordedDate=LocalDate.now();break;
    }
    this.comprehensiveGrade = this.calculateCompositeGrade();
    this.comprehensiveGradeRecordedDate = LocalDate.now();
}
```

外部查询

```java
@MethodOrder(9)
public LocalDate myGetComprehensiveGradeRecordedDate() { return this.comprehensiveGradeRecordedDate; }
```

(9) jansi 用于彩色命令行输出

```java
import org.fusesource.jansi.Ansi;
import org.fusesource.jansi.AnsiConsole;
```

```java
scoreCounts.forEach((grade, count) -> {
    Ansi.Color color = switch (grade) {
        case "A-90分以上" -> Ansi.Color.GREEN;
        case "B-80分以上" -> Ansi.Color.BLUE;
        case "C-70分以上" -> Ansi.Color.YELLOW;
        case "D-70分以下" -> Ansi.Color.RED;
        default -> Ansi.Color.DEFAULT;
    };

    Ansi ansi = Ansi.ansi();
    ansi.fg(color).a( value: grade + ": ");

    for (int i = 0; i < count; i++) {
        ansi.a( value: '■'); // 输出方块字符
    }

    System.out.println(ansi.reset());
```

(10) switch 表达式

```
Ansi.Color color = switch (grade) {
    case "A-90分以上" -> Ansi.Color.GREEN;
    case "B-80分以上" -> Ansi.Color.BLUE;
    case "C-70分以上" -> Ansi.Color.YELLOW;
    case "D-70分以下" -> Ansi.Color.RED;
    default -> Ansi.Color.DEFAULT;
};
```

(11)构造匿名对象与 Lambda 表达式

```
使用了 Lambda 表达式来创建一个匿名的 Comparator 对象
Arrays.sort(methods, Comparator.comparingInt(method -> {
    MethodOrder annotation = method.getAnnotation(MethodOrder.class);
    return annotation != null ? annotation.value() : Integer.MAX_VALUE; // 判空处理
}));
```

(12) Stream：例如转化成 stram 后用内部类 StudentRelationshipClass.StudentGrade 的方法 myGetComprehensiveGrade 构造比较器，来排序.

```
classStudentGradeListSorted = classStudentGradeList.stream()
        .sorted(Comparator.comparingDouble(StudentRelationshipClass.StudentGrade::myGetComprehensiveGrade).reversed())
        .toList();
```

(13) 异常处理

下面的代码包括手写的判空处理和 throws 机制。

```
public CenteredCommandLineTable(ArrayList<T> arrayList, Class typeClass) throws InvocationTargetException, IllegalAccessException {
    this.typeClass = typeClass;
      System.out.println("泛型参数类型: " + typeClass.getName());
    Method[] methods = typeClass.getMethods();
    使用了 Lambda 表达式来创建一个匿名的 Comparator 对象
    Arrays.sort(methods, Comparator.comparingInt(method -> {
        MethodOrder annotation = method.getAnnotation(MethodOrder.class);
        return annotation != null ? annotation.value() : Integer.MAX_VALUE; // 判空处理
    }));
    String subString = "myGet";
    int indexI = 1;
    int indexJ = 0;
    this.rows = arrayList.size()+1;
    this.columns = 0;
    for (Method method : methods) {
        if (method.getName().contains(subString)) {
            columns++;
        }
    }
    this.data = new String[rows][columns];
    for (Method method : methods) {
        if (method.getName().contains(subString)) {
            data[0][indexJ] = method.getName().replace(subString,  replacement: "");
            indexJ++;
        }
    }
```

四、实验结果及分析和（或）源程序调试过程（界面截图和文字）、实验总结与体会

（一）实验结果及分析

(1)根据命令行提示，输入教师数目和学生数目

```
==== 基于命令行的学生成绩管理系统 ====
请输入教师数目:
5
请输入学生数目:
100
```

图 19：实验结果展示

(2)模拟过程进行初始化

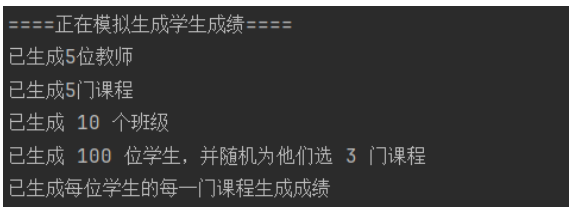


图 20：实验结果展示

(3)生成完成，进入主菜单

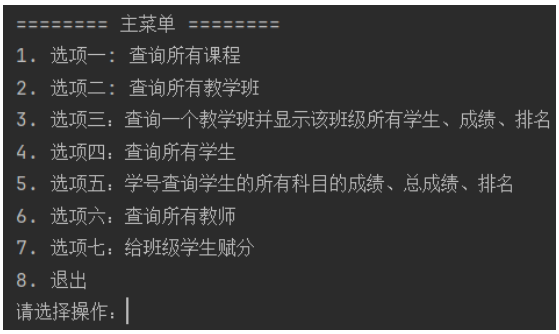


图 21：实验结果展示

(4)输入 1，查询所有课程显示表格

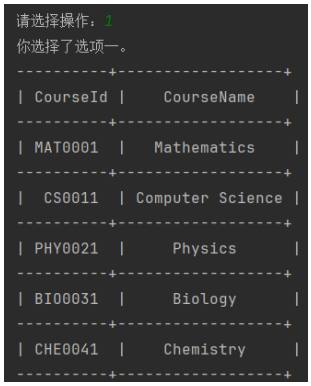


图 22：实验结果展示

(5)输入 2，查询所有教学班，显示表格

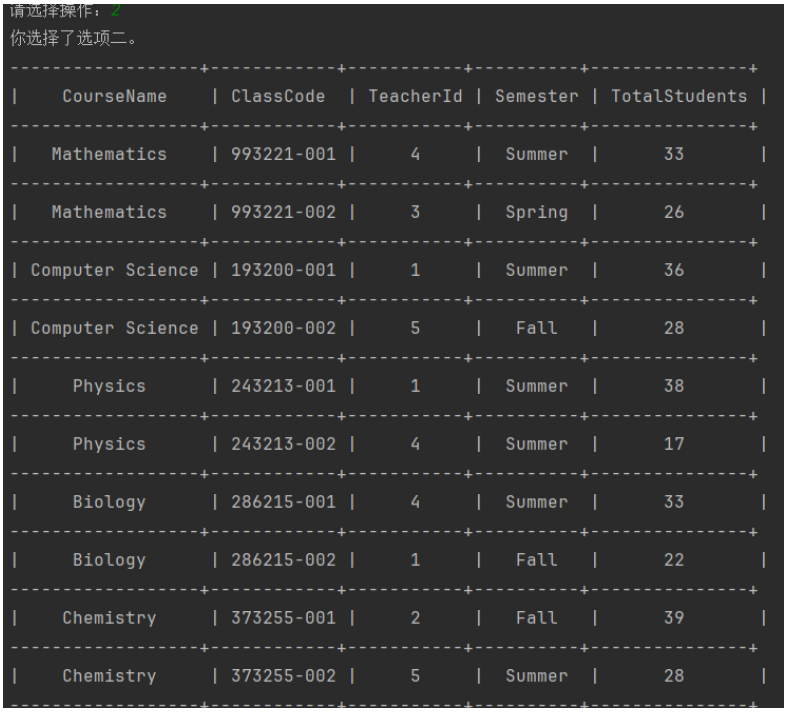


图 23：实验结果展示

(4)输入 3，输入教学班号 373255-001，如果记不清可以返回组菜单查询（输入 0 返回）

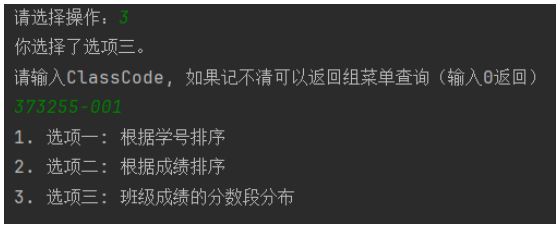


图 24：实验结果展示

（5）输入 1，则会根据学号排序后输出学生的成绩，也会展示班级排名

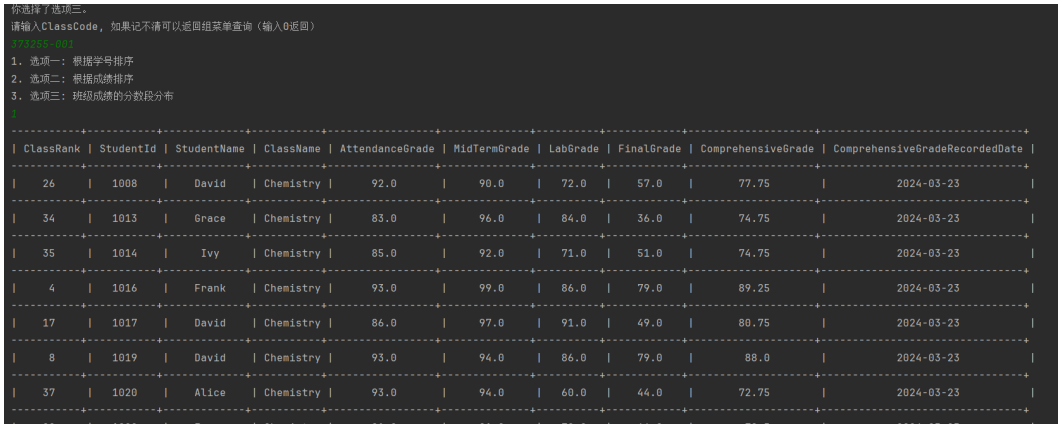


图 25：实验结果展示

（6）输入 2，则会根据成绩排序

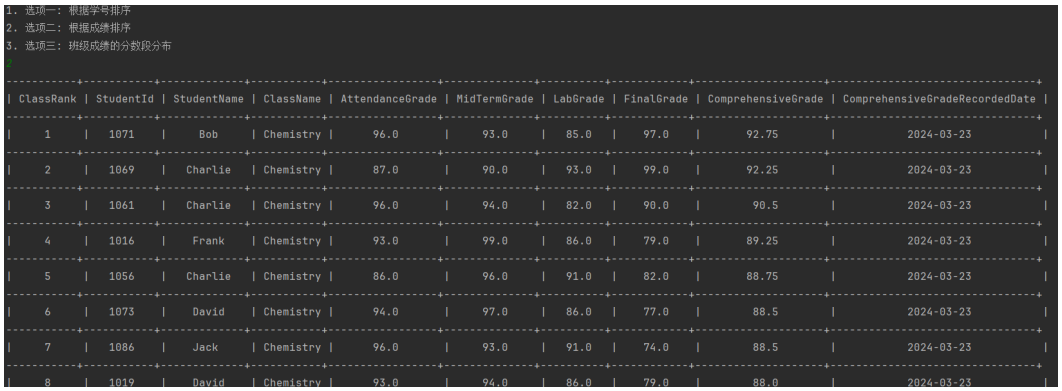


图 26：实验结果展示

（7）输入 3，则会绘制柱状图，显示班级学习情况

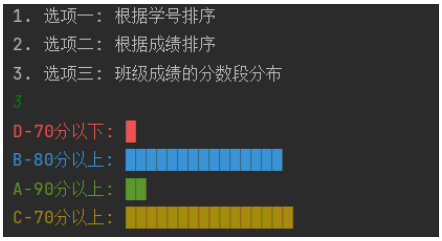


图 27：实验结果展示

（8）返回到主菜单，输入 4，查询全体的表格，包括综合成绩和排名

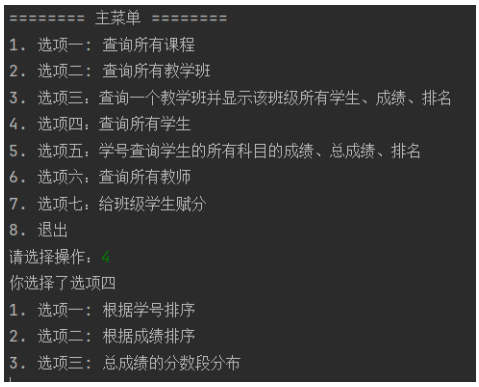


图 28：实验结果展示

（9）输入 1，则会根据学号排序后输出学生的成绩，也会展示整体排名

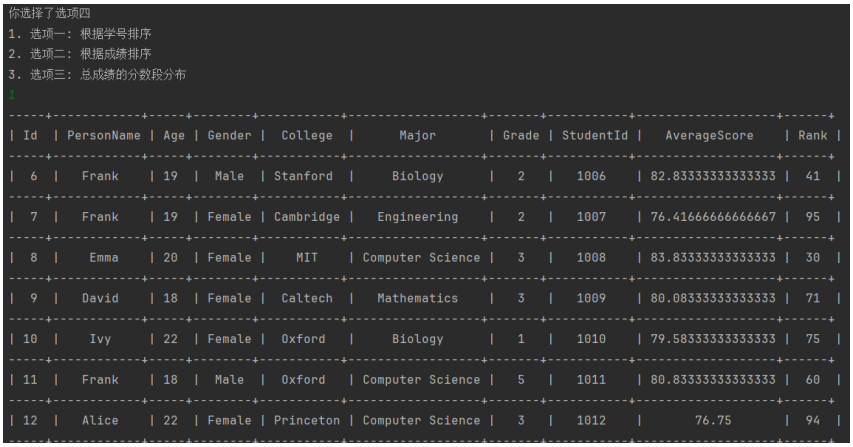


图 29：实验结果展示

（10）输入 2，则会根据成绩排序后输出学生的成绩，也会展示整体排名

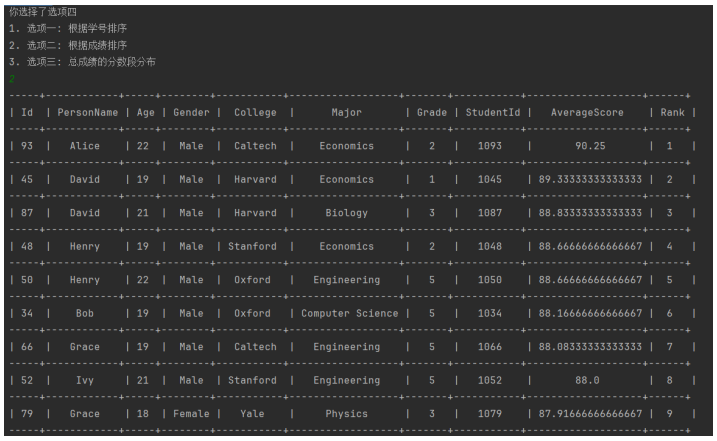


图 30：实验结果展示

（7）输入 3，则会绘制柱状图，显示全校学习情况

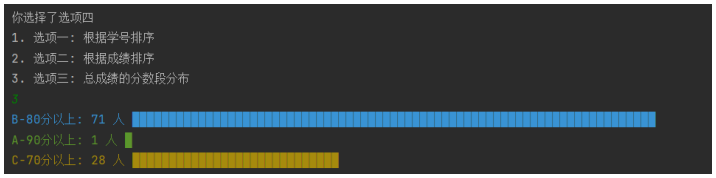


图 31：实验结果展示

（9）返回到主菜单，输入 5，指定学生学号 1105，返回各课成绩表格，平均成绩和已修科目

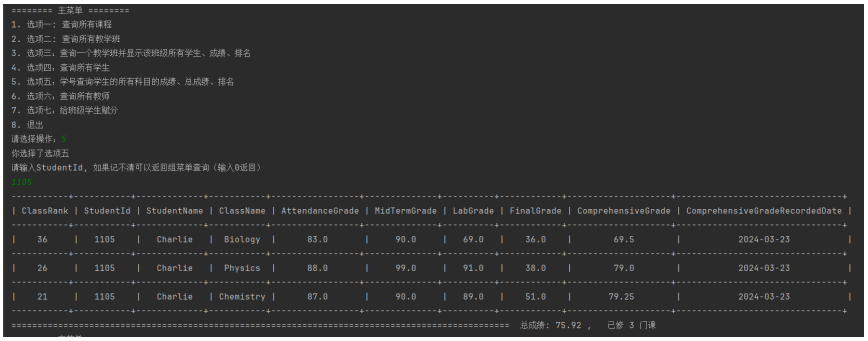


图 32：实验结果展示

（10）返回到主菜单，输入 6，查询所有教师

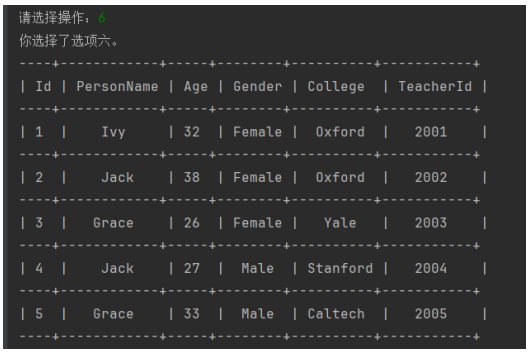


图 33：实验结果展示

（11）返回到主菜单，输入 7，给班级学生赋分，依次输入班级号 373255-001，1105，返回当前该学生这门课的成绩

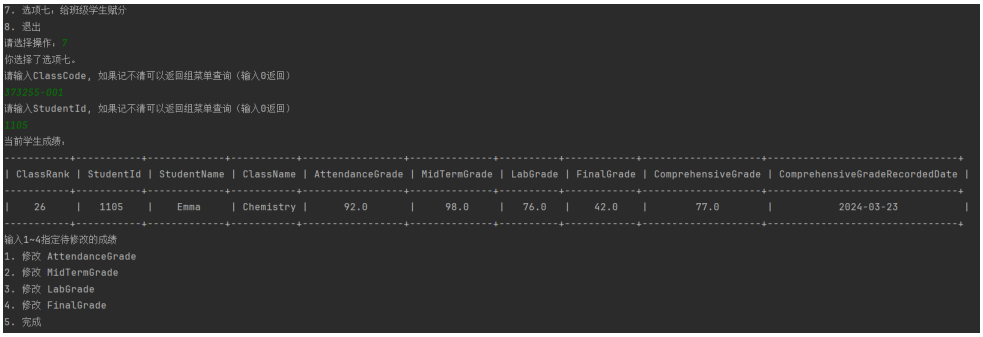


图 34：实验结果展示

（12）综合成绩由平时成绩，期中成绩，实验成绩，期末成绩构成，这里修改该学生的平时成绩，输入 1，和成绩 0；自动刷新排名和总分

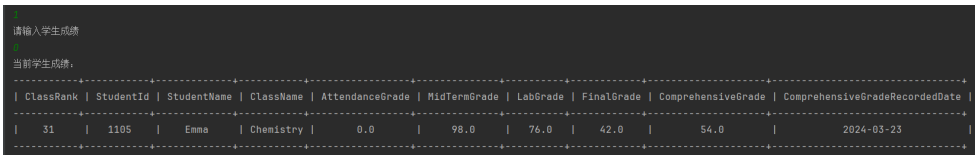


图 35：实验结果展示

（二）实验总结与体会

遇到的问题和解决：

（1）jansi import not found

解决:在 pom.xml 中引入依赖，刷新 maven

```xml
<dependencies>
    <dependency>
        <groupId>org.fusesource.jansi</groupId>
        <artifactId>jansi</artifactId>
        <version>2.4.0</version>
    </dependency>
</dependencies>
```

(2) 如何对成绩进行排序

解决：使用流排序

```
classStudentGradeListSorted =classStudentGradeList.stream()
        .sorted(Comparator.comparingInt(StudentRelationshipClass.StudentGrade::myGetStudentId))
        .toList();
```

（3）调用反射的函数时，method.invoke()要进行异常处理

```
for (T t : arrayList) {
    indexJ = 0;
    for (Method method : methods) {
        if (method.getName().contains(subString)) {
            Object[] parameters = {};
            data[indexI][indexJ] = method.invoke(t, parameters).toString();
            indexJ++;
        }
    }
    indexI++;
}
```

分析：

Method.invoke() 方法的签名如下：

```
public Object invoke(Object obj, Object... args) throws IllegalAccessException, IllegalArgumentException, InvocationTargetException
```

这个方法会在运行时动态地调用指定对象的方法。然而，由于反射是一种高度动态的特性，因此可能会出现多种异常情况，需要进行异常处理。
IllegalAccessException（非法访问异常）：当尝试访问一个方法、字段或构造函数时，但是访问权限不足时会抛出此异常。比如，如果方法是私有的，而你又没有相应的权限去访问它，就会抛出此异常。

IllegalArgumentException（非法参数异常）：如果传递给 Method.invoke() 方法的参数类型与实际方法所需的参数类型不匹配，或者传递的参数数量不正确，就会抛出此异常。

InvocationTargetException（调用目标异常）：当通过反射调用方法时，如果目标方法本身抛出了异常，那么这个异常会被包装在 InvocationTargetException 中抛出。
（4）在生成表头时，需要有序的反射回来的 methodjava 反射回来的 method 是怎么排序的。

Java 反射获取的 Method 数组中的方法未定义排序规则：Java 规范并没有具体规定反射返回的 Method 数组应该按照何种顺序排序，因此具体的排序规则可能会因 JVM 实现而异，或者在不同版本的 Java 中有所变化。
解决：使用自定义注解：可以为方法定义一个自定义注解，并在注解中指定方法的顺序。然后在进行反射时，可以先获取类中所有方法，然后根据自定义注解中定义的顺序进行排序或筛选。

实验报告填写说明：

1、第一、二部分由老师提供；

2、第三部分填写源程序或者算法，清单文件，资源文件等。源程序要符合程序编写风格（缩进、注释等）；

3、第四部分主要填写程序调试运行过程、结果（截图）、解决问题的方法、总结和体会等；

4、报告规范：包含报告页眉、报告的排版、内容是否填写，命名是否规范等。