Student File and R-Score Management System

Data Structure and Object-Oriented Programming

Louay Abaccha

2025-05-11

Table of Contents

1. Project Description

The goal of my project: Professors enter the marks of a student and then my system calculates their R-Score depending on each class weight . The system will then generate a transcript of the students data that they can print and view all of their information

In general my code while handle the :

- Management of students, courses, and professors
- Calculation of R-scores and transcript generation
- Data persistence via CSV files

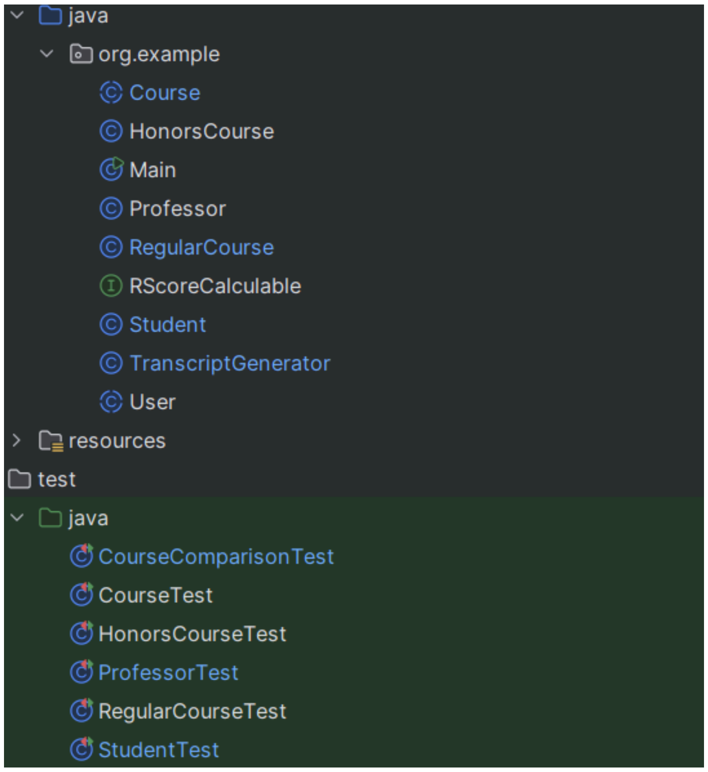
This is done through 2 major **Hierarchies :**

1) **User**
   Students —---- Professors

2) **Courses**
   Regularcourse —------- Enrichedcourse

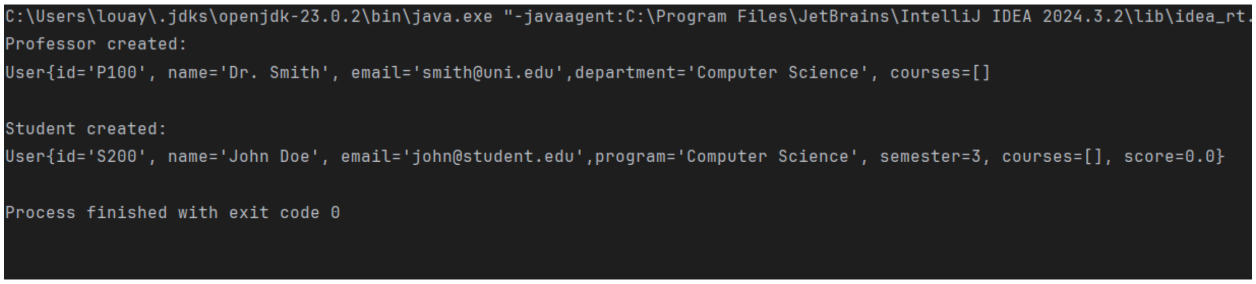
2. Program Features and Screenshots

My Code behold the following classes and Testing Units :



```
java
  org.example
    Course
    HonorsCourse
    Main
    Professor
    RegularCourse
    RScoreCalculable
    Student
    TranscriptGenerator
    User
resources
test
java
  CourseComparisonTest
  CourseTest
  HonorsCourseTest
  ProfessorTest
  RegularCourseTest
  StudentTest
```

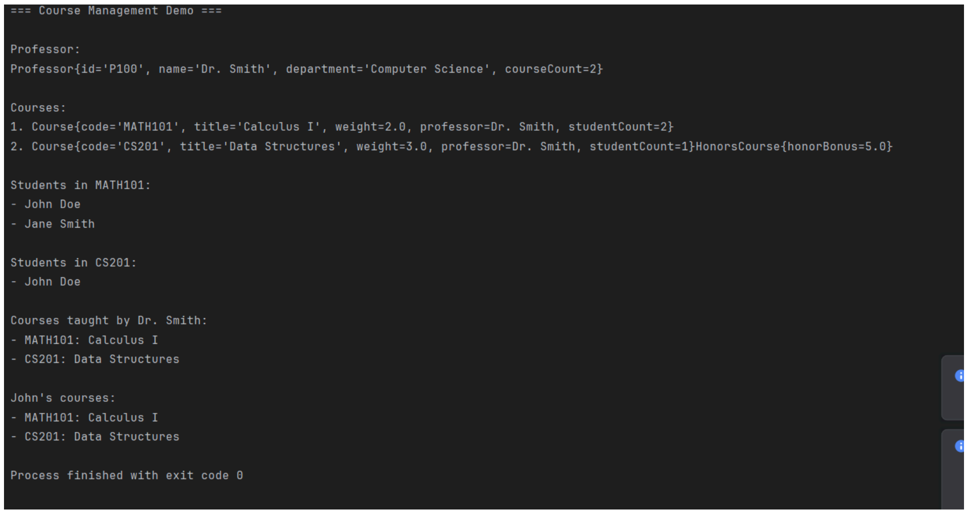
1. **Create a Teacher and a Student :**
   - Create a Professor with their Id , name , email , department and the number of courses they teach
   - Create a Student with their id , name , email , program , semester , course and score.
   - Uses data structures such as Arraylist and Queue



```
C:\Users\louay\.jdks\openjdk-23.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3.2\lib\idea_rt.
Professor created:
User{id='P100', name='Dr. Smith', email='smith@uni.edu',department='Computer Science', courses=[]

Student created:
User{id='S200', name='John Doe', email='john@student.edu',program='Computer Science', semester=3, courses=[], score=0.0}

Process finished with exit code 0
```

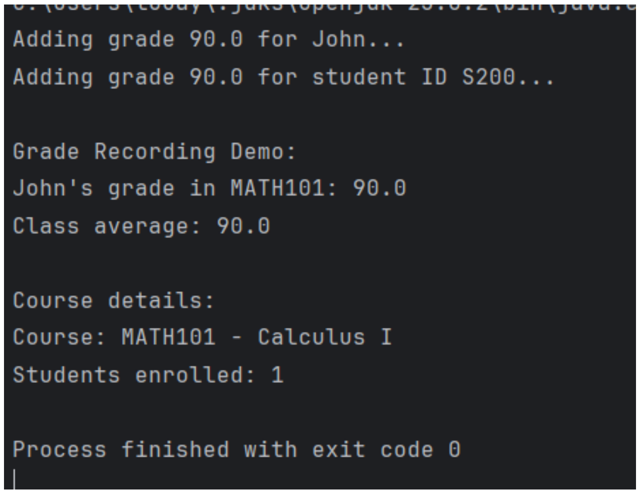
2. **Course Management**

   - Add/remove courses, enroll students
   - Uses Queue to efficiently store and process the program data



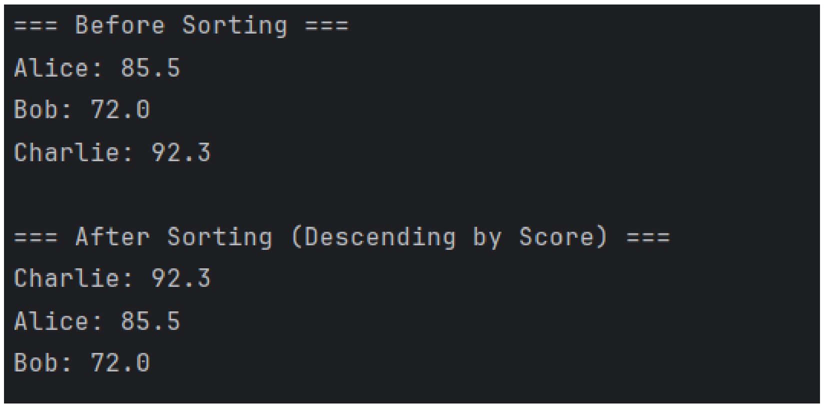
```
=== Course Management Demo ===

Professor:
Professor{id='P100', name='Dr. Smith', department='Computer Science', courseCount=2}

Courses:
1. Course{code='MATH101', title='Calculus I', weight=2.0, professor=Dr. Smith, studentCount=2}
2. Course{code='CS201', title='Data Structures', weight=3.0, professor=Dr. Smith, studentCount=1}HonorsCourse{honorBonus=5.0}

Students in MATH101:
- John Doe
- Jane Smith

Students in CS201:
- John Doe

Courses taught by Dr. Smith:
- MATH101: Calculus I
- CS201: Data Structures

John's courses:
- MATH101: Calculus I
- CS201: Data Structures

Process finished with exit code 0
```

3. **Grade Recording & R-Score Calculation**

   - Input grades, compute class averages and R-scores
   - Utilizes Stream processing with Lambda expressions
   - Method Overloading: addGrade() will have multiple versions (by student ID, by student name, with different parameter sets)
   - 

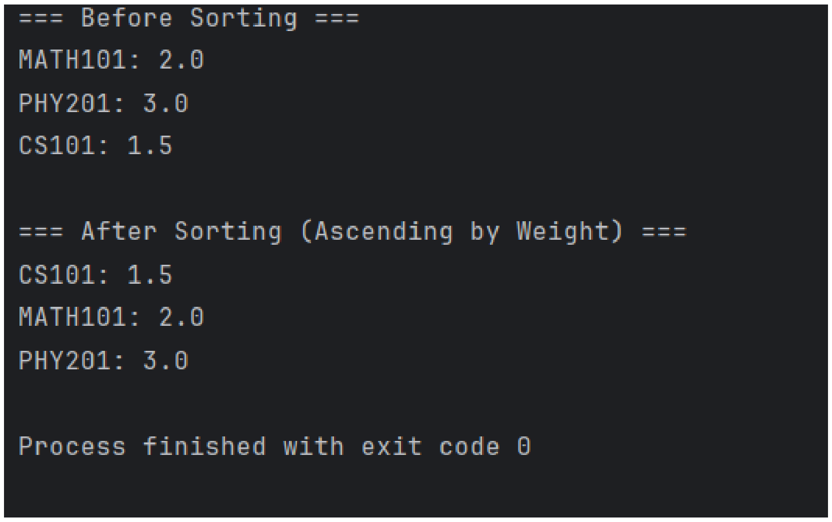
```
C:\Users\Coody\.jdks\openjdk-23.0.2\bin\java.e
Adding grade 90.0 for John...
Adding grade 90.0 for student ID S200...

Grade Recording Demo:
John's grade in MATH101: 90.0
Class average: 90.0

Course details:
Course: MATH101 - Calculus I
Students enrolled: 1

Process finished with exit code 0
```

4. **Student Sorting Demo (Using Comparator)**
   - **Compare student based on their R-scores**
   -



```
=== Before Sorting ===
Alice: 85.5
Bob: 72.0
Charlie: 92.3

=== After Sorting (Descending by Score) ===
Charlie: 92.3
Alice: 85.5
Bob: 72.0
```

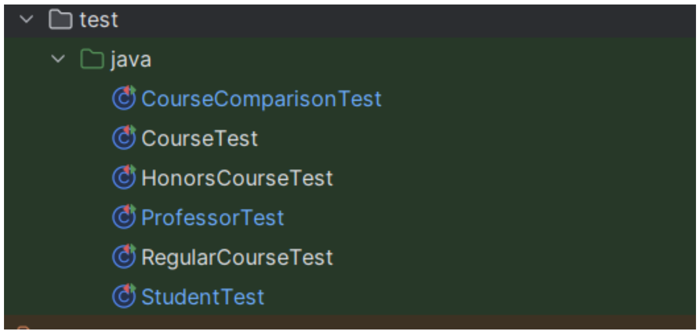
5. **Course Sorting Demo (Using Comparable)**
   - **Compare the course by natural ordering by weight**



```
=== Before Sorting ===
MATH101: 2.0
PHY201: 3.0
CS101: 1.5

=== After Sorting (Ascending by Weight) ===
CS101: 1.5
MATH101: 2.0
PHY201: 3.0

Process finished with exit code 0
```

6. **Junit testing :**

Done for all Methods present in the Code in these Classes :



```
test
  java
    CourseComparisonTest
    CourseTest
    HonorsCourseTest
    ProfessorTest
    RegularCourseTest
    StudentTest
```

- **RegularCourse.calculateRScore(...)**

Five scenarios (normal average, zero average, zero grade, negative grade, zero weight) to make sure basic R-score formula behaves exactly as expected.

- **HonorsCourse.applyBonus(...) & HonorsCourse.calculateRScore(...)**

Five different inputs to applyBonus (standard bonus, zero bonus, zero raw-grade, large bonus, fractional bonus).

Five scenarios for the honors-specific R-score (nonzero vs. zero class average, zero grade, negative grade-after-bonus clamped, fractional inputs).

- **Student.calculateRScore()**

Empty course list → empty map.

Single regular course → correct code→score entry.

Single honors course → correct boosted score.

Course with no grade recorded → defaults to 0.0 in the map.

- **Student.ScoreComparator**

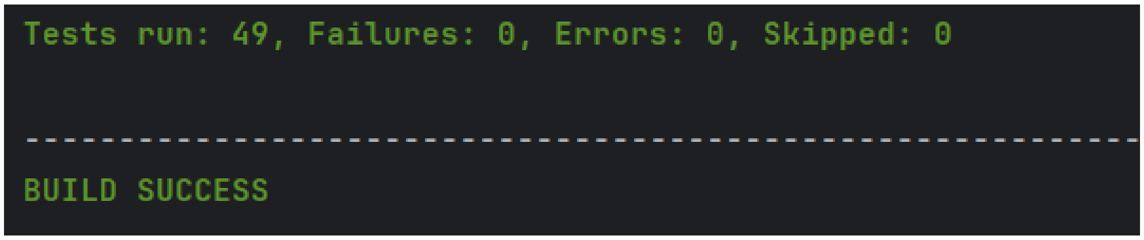
Five comparisons (descending order, equal scores, zero vs. positive, negative vs. positive, fractional values) to ensure students always sort in the right order.

- **Professor.assignCourse(...)**

Five tests covering adding to an empty list, duplicates, null handling, order preservation, and multiple adds.

- **Course.compareTo(...)**

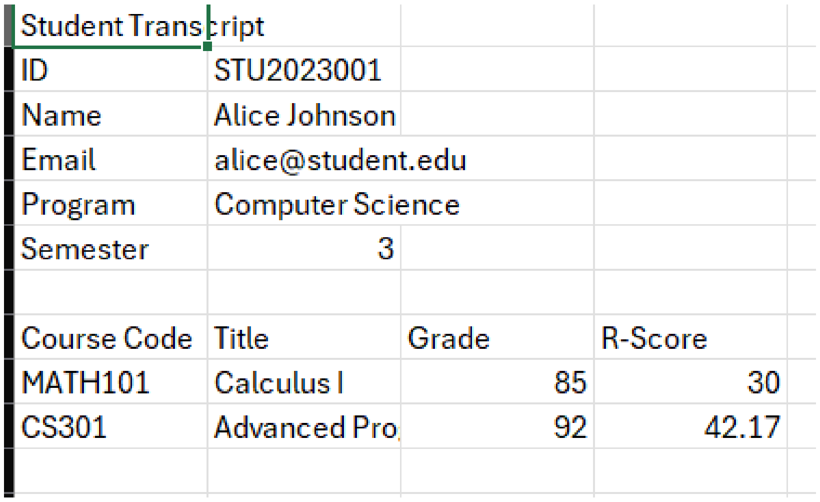
Five cases for smaller, equal, larger, zero vs. positive weights, and an end-to-end sort check.



```
Tests run: 49, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
```

6. **Text I/O : TranscriptGenerator**

- Purpose: To read student data
- Writes a CSV transcript for the given student to the specified file.



Student Transcript			
ID	STU2023001		
Name	Alice Johnson		
Email	alice@student.edu		
Program	Computer Science		
Semester	3		
Course Code	Title	Grade	R-Score
MATH101	Calculus I	85	30
CS301	Advanced Pro	92	42.17

3. Challenges

- **Testing Edge Cases:** Being able to think of multiple tests that could be done in order to make my code as functional such as test Negative grades, zero class averages, missing data.
- **Method Overloading Confusio**n

  I had trouble understanding how to properly implement multiple versions of addGrade() that would: Accept different parameter types (Student object vs. String ID) ,Handle edge cases (null checks, invalid grades) consistently across all versions.

- **Maintaining State**

  Got confused about: Whether calling addGrade() with student ID would properly find the Student object If later grade updates would overwrite previous ones correctly

- **GitHub**

  I had issues using Github , pushing and commit my code either through GitBash or the directly

4. Learning Outcomes

Reflect on what you learned:

- Deepened understanding of Java I/O (FileWriter,)
- Practice with JUnit testing and test-driven development , using it to help me enhance my code
- Improved debugging and refactoring skills
- Gained familiarity with CSV
- Learned how to write a clean code with useful comment and code description
- Learnt how to Organize myself with a big coding project
- I Learned (somehow) how to use Github , I know what to do and what to never do - Force push !