mlyann / awesome-gradebook

Type / to search

Code Issues Pull requests Actions Projects Security Insights Settings

awesome-gradebook

Private

1 Branch 1 Tag Go to file t Go to file Add file Code

boldthinkingcat Merge remote-tracking branch 'origin/main'

29cac28 · 1 minute ago 87 Commits

.idea	Score。	last month
data	FIX BUG and UPDATE TEST...	1 hour ago
src	Testcase update	1 minute ago
.gitignore	openai integration	last month
README.md	Update README.md	25 minutes ago
REV10-ML+Logo.png	Add files via upload	3 days ago
pom.xml	Basic function for all UI,	3 days ago

About

A final project in CSC335

Readme

Activity

1 star

1 watching

0 forks

Releases

1 tags

Create a new release

Packages

No packages published

README

Gradebook App (CSC 335 Final Project)

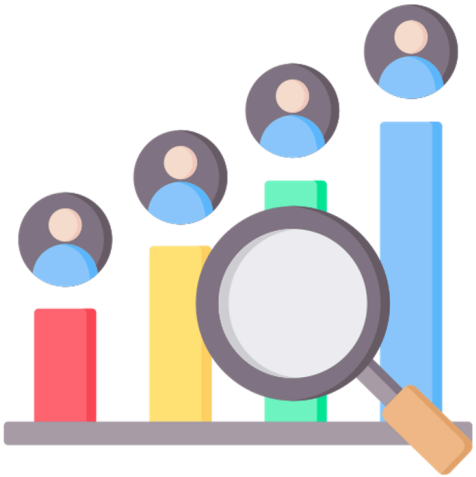
JAVA INTELLIJ

Contributors 3

mlyann Minglai Yang

YuJerry1 Jerry Yu

Welcome to the **Gradebook App**! This console-based Java application allows you to manage a Gradebook, calculating GPAs, add/remove assignments, grade the assignments, and customize different weights for each quizzes. It is a project for **CSC 335 Final Project**, taught by **Instructor Malenie Lotz** with the assistance of **TA Paulina**. The authors of this project are [Haocheng Cao](), [Minglai Yang](), [Jerry Yu]() and [Alan Soto](). This project will be open-source after the grades released.



GRADEBOOK

Table of Contents

1. Overview
2. Features
3. Project Structure
4. Getting Started
5. How to Use
6. Sample Flow
7. Design
8. Author & Acknowledgments
9. License

Overview

The Gradebook System is a command-line (CLI) application that mirrors the essentials of a D2L-style gradebook for both students and teachers:

- Separate views for students and teachers, each exposing only the actions relevant to that role.
- Flexible grading logic supporting both total-points and category-weighted schemes, with optional dropped assignments.
- Rich analytics – class averages, medians, GPA, per-category statistics, and identification of ungraded work.
- Roster and group utilities to add, remove, sort, or batch-import students.

Features

1. Course Management

- View current and completed courses (both roles).
- Teachers: add / remove courses, mark courses as completed.

2. Roster Management

- Teachers:
- Import students from text / CSV files.
- Add / remove individual students.
- Sort roster by first name, last name, username, or assignment grade.
- Create and manage student groups.

3. Assignment Management

- Teachers:
- Add or delete assignments (with points or category tagging).
- Bulk list “ungraded” assignments for quick grading.
- Students: view all graded / ungraded assignments per course.

4. Grading & Analytics

- Two grading modes per course:
- Total-points: Final Grade(%) = total points earned / total points possible.
- Category-weighted: weight-based aggregation with optional lowest-n drops per category.

Teachers:

- Enter / edit grades.
- Compute class average and median on any assignment.
- View per-student current average and overall course statistics.
- Assign final letter grades (A–E) from computed averages.

Students:

- Compute class average on completed work.
- Calculate GPA across all completed courses or selected subsets.

5. Student View

- List enrolled courses with completion status.
- Drill down to assignment details, grades, and teacher feedback.
- On-demand calculation of class average and cumulative GPA.

6. Teacher View

- Dashboard of all taught courses with quick links to roster, assignments, analytics, and grading setup.
- Dedicated “Analytics” panel summarizing class distribution, category stats, dropped assignments, and ungraded items.

7. Grading Setup

- Interactive CLI wizard for selecting grading mode, defining categories, setting weights (%), and configuring drop rules.
- All the other points will be extra credits!

Project Structure

```
org.fp/
├── AdminUI.java           // Admin interface
├── Assignment.java        // Represents an assignment with metadata
├── BaseController.java    // Shared functionality for controllers
├── Course.java            // Represents a course with roster, assignments, grading config
├── DataStore.java         // Handles saving/loading course and user data
├── DecryptVIC.java        // Decrypt logic for secure data
├── EncryptVIC.java        // Encrypt logic for secure data
├── GPT.java               // Handles GPT-powered feedback integration
├── Grade.java             // Enum or class for final letter grades
├── GradeCalculator.java   // Handles grading logic
├── IDGen.java             // Unique ID generation utility
├── LibraryModel.java      // Central model containing all student/teacher/course/assignment data
├── LibraryUsers.java      // Utility for managing users
├── LoginUI.java           // CLI login menu and user routing
├── ProgressBar.java       // Utility for showing progress indicators
├── Score.java             // Represents a student's score for a specific assignment
├── Student.java           // Student entity with personal and course-related data
├── StudentController.java // Logic layer between student UI and model
├── StudentUI.java         // CLI for students
├── TablePrinter.java      // Formats and prints tables to console
├── Teacher.java           // Teacher entity with managed courses
├── TeacherController.java // Logic layer between teacher UI and model
├── TeacherUI.java         // CLI for teachers
├── VICData.java           // Handles encrypted data configuration
└── VICOperations.java     // Encryption/decryption operations for secure data handling
```

- **TeacherUI, StudentUI, LoginUI – The View / UI**
  Display menus, accept input, show results.

- **TeacherController / StudentController – The Controllers**
  Utility class used for Act as intermediaries between UI (e.g., TeacherUI, StudentUI) and LibraryModel. and printing lists of data in a table-like structure to the console.

- **LibraryModel – The Model**
  Manages the state of the system (students, teachers, courses, assignments, grades, relationships, grading logic).

Getting Started

Prerequisites

- **Java 8** or higher.
- A Java-compatible IDE (e.g., IntelliJ, Eclipse, VS Code) or the ability to compile via the command line. We use IntelliJ for this project!!

Installation & Compilation

1. Clone or download this repository:

```
git clone https://github.com/mlyann/awesome-gradebook.git
```

2. Open the project in your preferred IDE or navigate into the project folder via terminal:

```
cd awesome-gradebook
```

3. Ensure that the package structure (fp) is respected if you are using an IDE.
4. Compile the code (if using command line):

```
javac fp/*.java
```

5. Running the Application After compilation, run the main application class:

```
java fp.LoginUI
```

Sample Flow

Below is a brief example of how a typical session might proceed in the console:

1. **Main Menu**

```
1) Register  2) Login  3) Exit
👉 Choice: 2
Username: Ming Yang
Password: ****************
✅ Login success: TEACHER/STUDENT
```

2. **Teacher Menu** It will show the teacher menu if you are a teacher. And give a list of your courses.

```
====================================================
           🎉 Courses taught by 2 2 (sorted by none) 🎉
====================================================
+-----+-------------+--------------+
| No. | Course Name | Description  |
+-----+-------------+--------------+
| 1   | CS252       | ASM          |
| 2   | CS335       | Obj-Oriented |
+-----+-------------+--------------+
1) 🔍 Select a course
m) 🛠️ Course Management
s) 🔀 Change sort
0) 🚪 Exit
👉 Choice:
```

You can pick either a course or go to the course management menu. If you select a course, it will show the course menu.

```
====================================================
           🎉 Assignments for Course: CS252 (sorted by none) 🎉
====================================================
+-----+-----------------+------------+------------+-------------------------------------------+-------------+------
----------------------+------------+
| No. | Assignment Name | Assigned   | Due        | Progress                                  | Submissions |
Graded                     | Published? |
+-----+-----------------+------------+------------+-------------------------------------------+-------------+------
----------------------+------------+
| 1   | HW 1            | 2025-04-01 | 2025-04-06 | [####----------------] ⏳ 4 days left     | 5/7         |
```

```
[####################] 5/5 | ❌ No     |
| 2   | HW 2            | 2025-04-01 | 2025-04-06 | [####----------------] ⏳ 4 days left     | 5/7         |
[################----] 4/5 | ❌ No     |
| 3   | HW 3            | 2025-04-01 | 2025-04-06 | [####----------------] ⏳ 4 days left     | 7/7         |
[##############------] 5/7 | ❌ No     |
| 4   | HW 4            | 2025-04-01 | 2025-04-06 | [####----------------] ⏳ 4 days left     | 6/7         |
[################----] 5/6 | ❌ No     |
| 5   | Project 1       | 2025-04-01 | 2025-04-11 | [##------------------] ⏳ 9 days left     | 6/7         |
[################----] 5/6 | ❌ No     |
| 6   | Project 2       | 2025-04-01 | 2025-04-11 | [##------------------] ⏳ 9 days left     | 6/7         |
[################----] 5/6 | ❌ No     |
| 7   | Quiz 1          | 2025-04-01 | 2025-04-03 | [##########----------] ⏳ 1 day left      | 7/7         |
[##############------] 5/7 | ❌ No     |
| 8   | Quiz 2          | 2025-04-01 | 2025-04-03 | [##########----------] ⏳ 1 day left      | 5/7         |
[########------------] 2/5 | ❌ No     |
| 9   | Quiz 3          | 2025-04-01 | 2025-04-03 | [##########----------] ⏳ 1 day left      | 4/7         |
[###############-----] 3/4 | ❌ No     |
+-----+-----------------+------------+------------+-------------------------------------------+-------------+------
----------------------+------------+
```

3. **Teacher Operations**

```
a) 📄 Assignments    r) 👥 Roster    g) 🏁 Final Grades    c) ⚙️ Grading setup    s) 🔍 Search    f) 🧱 Filter
o) 🔀 Sort    d) 🛠️ Assignments Manage    n) 📊 Analytics    m) ✅ Mark Completed    0) 🔙 Back```
```

4. **Student Menu**

```
=====================================================
           🎉 Courses of Aria Griffin (sorted by none) 🎉
=====================================================
+-----+-------------+--------------+-------0------+
+-----+-------------+--------------+--------------+
| 1   | CS335       | MALENIE LOTZ | ✅ Completed |
| 2   | CS252       | ASM          | ✅ Completed |
+-----+-------------+--------------+--------------+
```

5. **Student Operations**

```
1) 🔍 Select a course    s) 🔀 Change sort    p) 🤖 Personal feedback    g) 📈 GPA    0) 🚪 Exit
👉 Choice:
```

You can pick either a course or go to the GPA menu. If you select a course, it will show the course menu. Below is a sample for Student's GPA report. Here Student can also call GPT for feedback.

```
==================================================
 Course       | Percent    | Pts    | Grade
--------------------------------------------------
 CS252        |   92.60% |   4    | A
 CS335        |  164.08% |   4    | A
--------------------------------------------------
 OVERALL      |  128.34% | 4.00 | A
==================================================

====================================================
           🎉 Courses of Aria Griffin (sorted by none) 🎉
====================================================
+-----+-------------+--------------+-------------+
| No. | Course Name | Description  | Status      |
+-----+-------------+--------------+-------------+
| 1   | CS335       | MALENIE LOTZ | ✅ Completed |
| 2   | CS252       | ASM          | ✅ Completed |
+-----+-------------+--------------+-------------+
```

Design

1. Clear separation of concerns between front and backend code

Layer	Folder
Model	`LibraryModel`
View	`StudentUI, TeacherUI`

Layer	Folder
Controller	`StudentController, TeacherController`

Model (`LibraryModel`)

- Hold state
- Contain business rules & calculations

View (`StudentUI, TeacherUI`)

- Present data already prepared by Controller
- Collect raw user input

Controller (`StudentController, TeacherController`)

- Convert UI actions into model calls
- Enforce validation & transactions

2. Data structures and Java library features

Feature	Why we chose it	Where you can see it
`List / ArrayList`	Order-preserving, maps naturally to tables (rosters, assignments).	`TeacherController.cachedStudents`, `StudentController.cachedAssignments`
`Map<K,V> / HashMap`	O(1) look-ups by ID; perfect for **course → assignments** or **student → scores**.	`LibraryModel.courseMap`, `Course.categoryWeights`, `TeacherController.groupedAssignments`
`Set / HashSet`	Fast membership tests without duplicates	`TeacherController.deletedAssignmentIDs`, `deletedStudentIDs`
Enums	no illegal strings, also small and FLYWEIGHT	`Assignment.SubmissionStatus`, `Grade`, `TeacherUI.ViewMode`

Feature	Why we chose it	Where you can see it
`Comparator*`	Multi-key sorts with classes.	`StudentController.sortCachedAssignmentsByName()`
`switch` expressions	concise branching on enums.	`TeacherUI.nextRosterSort()`
Read resources such as CSV	Auto-closing I/O, operations.	`LibraryModel.loadStudentsFromDirectory()`
`Collections.unmodifiableMap`	Expose **read-only** for DEEPCOPY.	`Course.getCategoryWeights()`
`DirectoryStream<Path>` (NIO2)	Fast, memory-light directory walks with glob filters.	`LibraryModel.loadStudentsFromDirectory()`

Key usage

```java
// TeacherController
groupedAssignments = model.getAssignmentsInCourse(courseID)
                          .stream()
                          .collect(Collectors.groupingBy(Assignment::getAssignmentName));

private double computeWeightedPercentage(String sid, String cid) {
    Map<String, List<Score>> byCat = new HashMap<>();

    // bucket scores per category
    for (Assignment a : getAssignmentsForStudentInCourse(sid, cid)) {
        var s = getScoreForAssignment(a.getAssignmentID());
        if (s != null) byCat.computeIfAbsent(a.getCategory(), _ -> new ArrayList<>()).add(s);
    }
    return course.getCategoryWeights().entrySet().stream()
                 .mapToDouble(e -> {
                     var list = byCat.getOrDefault(e.getKey(), List.of());
                     list.sort(Comparator.comparingDouble(Score::getPercentage));   // low→high
                     var kept  = list.stream().skip(course.getDropCountForCategory(e.getKey()));
                     int earned = kept.mapToInt(Score::getEarned).sum();
                     int total  = kept.mapToInt(Score::getTotal).sum();
                     return total == 0 ? 0
                                       : e.getValue() * earned / total;
```

```
                }).sum() * 100;
}
```

Using Dates to Calculate Progress

```
static String fullBar(LocalDate start, LocalDate due, LocalDate today) {
    long total = DAYS.between(start, due);
    long done  = DAYS.between(start, today);
    int filled = (int) Math.max(0, Math.min(20, 20 * done / total));
    return "[" + "#".repeat(filled) + "-".repeat(20 - filled) + "] "
           + (today.isAfter(due) ? "⏳ Done" : "");
}
```

3. Correct and thoughtful use of composition, inheritance, and/or interfaces

Mechanism	Why we used it	Concrete places in the code
Inheritance	Share cross-cutting logic, caches and helper methods among every controller.	`java class BaseController (abstract) … class StudentController extends BaseController class TeacherController extends BaseController `
Composition (has-a)	Model real-world relationships and keep the inheritance tree shallow.	*Course has many Assignments →* `Course.addAssignment()` *Student has many assignmentIDs →* `Student.addAssignment()` *TeacherController owns a* `LibraryModel` *reference – it* ***does not*** *extend the model.*

We also use the following design principles to ensure our code is clean and maintainable:

4. Encapsulation

- All mutable fields are private – callers must use getters / setters, e.g. private Map<String, Assignment> assignments inside Course.
- Controllers never give references – every public getter returns deep copies, so UI code can't mutate the model accidentally.

Deep-copy pattern we use:

```java
// in the Course.java - COPY CONSTRUCTOR
public Course(Course src) {
   this.courseID          = src.courseID;          // immutable → straight copy
   this.courseName        = src.courseName;
   this.courseDescription = src.courseDescription;
   this.teacherID         = src.teacherID;
   this.assignments = new HashMap<>();
   for (var e : src.assignments.entrySet()) {
      this.assignments.put(e.getKey(), new Assignment(e.getValue())); // <-- new instance!
   }
   this.useWeightedGrading = src.useWeightedGrading;
   this.categoryWeights    = new HashMap<>(src.categoryWeights);
   this.categoryDropCount  = new HashMap<>(src.categoryDropCount);
}
```

When a controller exposes cached data:

```java
// BaseController.java
public List<Course> getCachedCourses() {
   List<Course> safe = new ArrayList<>();
   for (Course c : cachedCourses) safe.add(new Course(c));  // deep copy
   return safe;
}
```

Model also return Copies:

```java
// LibraryModel.java
public Course getCourse(String id) {
    Course raw = courseMap.get(id);
    return (raw == null) ? null : new Course(raw);  // deep copy again
}
```

Private state + accessors – fields stay private; only read-only views

```java
// Course.java
private final Map<String, Assignment> assignments;

public Assignment getAssignmentByID(String id) {
    Assignment raw = assignments.get(id);
    return (raw == null) ? null : new Assignment(raw);   // returns a COPY
}
```

5. Avoidance of antipatterns

Temporary Field

no GPA or class-average fields exist; they are derived on demand.

```java
// LibraryModel.java
public double calculateGPA(String stuID) { … }
public double calculateClassAverage(String cid) { … }
```

DUPLICATE CODE

- Functions are in LibraryModel (computeWeightedPercentage, computeTotalPointsPercentage).
- UI & controllers call those helpers instead of re-implementing.

PRIMITIVE OBSESSION

Wrap them in A, B, C, D, E, F:

```java
enum Grade { A,B,C,D,E,F }              // domain concept, not just 'char'
enum SubmissionStatus { UNSUBMITTED, SUBMITTED_UNGRADED, GRADED }

record Score(int earned, int total) {      // self-checking value object
double getPercentage() { return 100.0*earned/total; }
}
```

GOD CLASS

Strict MVC keeps responsibilities small:

- **LibraryModel**: holds state, does calculations.
- **TeacherController, StudentController**: mediate model->view
- **StudentUI, TeacherUI**: show menus, get input, print results.
- We also have a lot of small classes to split different functionalities.

6. Use of design patterns

- **Flyweight – `IDGen`**
  - **Intent :** share a *single* pool of counters for every object-prefix (`STU-` , `TCH-` , `CRS-` , `ASG-`) so we never create redundant counter state.
  - **How :**

```java
// IDGen.java
public final class IDGen {
    private static final Map<String,Integer> COUNTERS = new HashMap<>();
    public static synchronized String generate(String prefix){
        int next = COUNTERS.merge(prefix, 1, Integer::sum) - 1;
        return prefix + next;
    }
    public static void initialize(String prefix,int start){ COUNTERS.put(prefix,start); }
}
```

    Every call returns a lightweight `String` , while the heavy state (`COUNTERS`) is *shared*.

- **Strategy – runtime-selectable sorting**
  - **Intent :** allow the caller to switch "how to sort" **without** touching `TeacherController` / `StudentController` internals.
  - **How :** an `enum` + a bank of `Comparator` lambdas are the concrete *strategies*; the `switch` merely picks one.

```java
// TeacherController.jave
switch (sort) {
```

```
        case NAME         -> names.sort(namePrefixThenNumber());
        case ASSIGN_DATE  -> names.sort(dateCmp(a -> a.getAssignDate()));
        case DUE_DATE     -> names.sort(dateCmp(a -> a.getDueDate()));
        case SUBMISSION   -> names.sort(Comparator.comparingInt(this::submitted).reversed());
        case GRADED_PERCENT -> names.sort(Comparator.comparingDouble(this::gradedPct).reversed());
        default -> names.sort(String::compareToIgnoreCase);
    }
    // each helper returns a Comparator = concrete Strategy
```

The UI just calls `nextSort(sort)` and the controller swaps in a different *strategy* object on the fly.

- **Iterator – safe traversal without leaking internals**
  - **Intent :** clients iterate over model data **without** holding live references.
  - **How :** every "getter" hands back a *copy* that still supports Java's built-in `Iterator` .

```
// LibraryModel.java
public Collection<Student> getAllStudents() {
    List<Student> copy = new ArrayList<>();
    for (Student s : studentMap.values()) copy.add(new Student(s)); // deep-copy
    return Collections.unmodifiableCollection(copy);   // exposes only an Iterator
}

// usage
for (Student s : model.getAllStudents()) { … }   // ← external code can iterate safely
```

The client gets an *iterator* view, but the mutable map inside `LibraryModel` stays encapsulated.

7. Input validation

- **Constructor-level guards** – never let a broken object exist.

```
// Assignment.java
public Assignment(String name,String stuID,String cid,
                  LocalDate assign,LocalDate due){
    if (stuID==null || cid==null || assign==null || due==null)
        throw new IllegalArgumentException("Assignment requires all fields.");
```

```
        ...
    }
```

- **Safe numeric parsing in the console UI**

```java
// TeacherUI.java
String choice = sc.nextLine().trim();
if (choice.matches("[1-9][0-9]*") &&
        Integer.parseInt(choice) <= courseData.size()) {
        ...
        } else {
        System.out.println("❌ Invalid choice.");
}
```

** ID Existence**

```java
// StudentController.setCurrentStudent() method
if (model.studentExists(id)) {
    currentStudentID = id;
} else {
    System.out.println("❌ Tried to set non-existent student: " + id);
}
```

8. **Explain what any AI-generated code**

We pasted the suggested unicode icons into our `TeacherUI` / `StudentUI` `System.out.println(...)` calls.

```
1) 🔍 Select a course    s) 🔀 Change sort    p) 🤖 Personal feedback    g) 📈 GPA    0) 🚪 Exit
👉 Choice:
```

Contributing

Contributions are what make the open source community such an amazing place to learn, inspire, and create. Any contributions you make are **greatly appreciated**.

If you have a suggestion that would make this better, please fork the repo and create a pull request. You can also simply open an issue with the tag "enhancement". Don't forget to give the project a star! Thanks again!

1. Fork the Project
2. Create your Feature Branch (`git checkout -b feature/AmazingFeature`)
3. Commit your Changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the Branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

Contact

Haocheng Cao - @Haocheng - cao8@arizona.edu

Minglai Yang - @Ming - mingly@arizona.edu

Jerry Yu - @Jerry - jerryyu1@arizona.edu

Alan Soto - @Alan - asgalan04@arizona.edu

Project Link: https://github.com/mlyann/music-store

Acknowledgments

We deeply appreciate the guidance and support of our instructor Malenie Lotz and TA Paulina Aguirre throughout this journey. 😊

Mentors:

- Instructor: Malenie Lotz (CSC 335)
- Teaching Assistant: Paulina

License