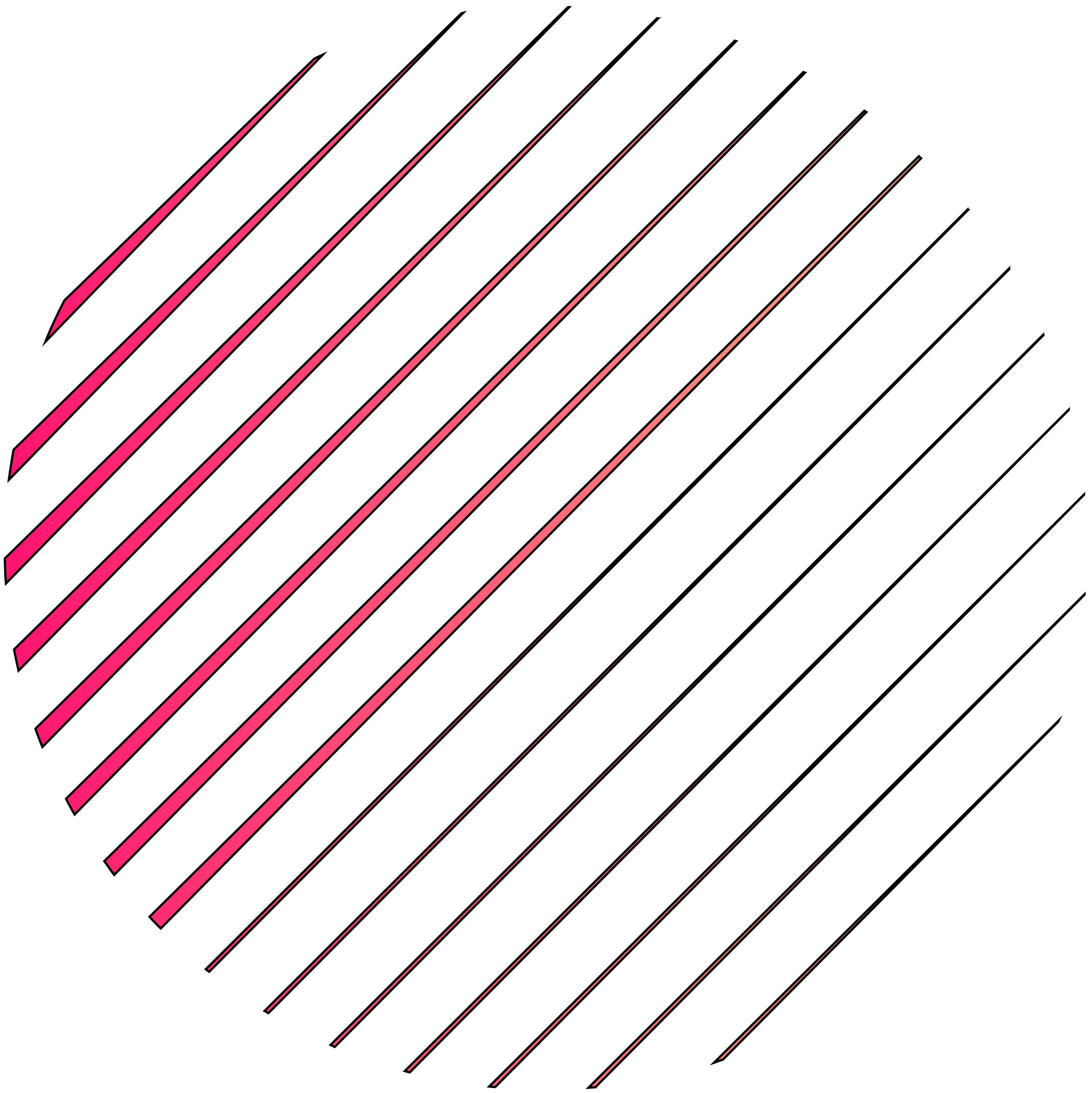
Java Programming



DATE 20 July – 20 August
COURSE TITLE: Internship

STUDENT'S NAME: ShehrYar Ahmed Khan
Student Id : CA/JU1/34691

1. Introduction

Purpose

This document provides comprehensive technical and functional documentation for the `ClassGradeManager` Java Swing application. It details the application's design, implementation, features, and usage, serving as a definitive guide for developers, maintainers, and end-users.

Scope

The scope of this document encompasses the entire `ClassGradeManager` application, including its graphical user interface (GUI), core business logic for grade management and GPA calculation, data structures, and user interaction flows. It does not cover external dependencies beyond standard Java Swing libraries.

Audience

This documentation is intended for:

- **Developers**: For understanding the codebase, contributing to development, and debugging.
- **Maintainers**: For performing updates, bug fixes, and performance optimizations.
- **Quality Assurance (QA) Testers**: For verifying functionality and identifying issues.
- **End-Users**: For learning how to effectively use the application.

2. System Overview

Application Description

The `ClassGradeManager` is a standalone desktop application built using Java Swing, designed to streamline the process of managing student academic records. It provides a user-friendly interface for inputting student information, adding individual course grades, calculating Grade Point Averages (GPAs), and generating comprehensive academic summaries and class-wide statistics. The application aims to simplify grade tracking for educators or individuals managing personal academic progress.

Key Features

- **Intuitive User Interface**: A clean and organized GUI facilitates easy data entry and navigation.
- **Student Enrollment**: Allows the addition of multiple students, each with a predefined number of courses.
- **Course Management**: Supports adding course names, credit hours, and letter grades (A+ to F) for each student.
- **Individual GPA Calculation**: Computes the GPA for each student based on their entered courses and grades, highlighting their best and worst course performances.
- **Detailed Student Summaries**: Generates a tabular breakdown of all courses taken by a student, including credits, grades, and quality points.
- **Comprehensive Class Statistics**: Provides an aggregated view of the entire class's academic performance, including average GPA, highest and lowest GPAs, and a detailed grade distribution across all courses.
- **Data Reset Functionality**: A "Clear All" option to reset the application state and remove all stored student data.
- **Modern Dark Theme**: Implements a visually appealing dark theme using Nimbus Look and Feel for enhanced user experience, especially in low-light environments.

- **Robust Input Validation**: Ensures data integrity by validating user inputs for student names, number of courses, credit hours, and grade formats, providing informative error messages.

3. Architecture and Design

The `ClassGradeManager` follows a simplified Model-View-Controller (MVC) pattern, though not strictly enforced, where the `ClassGradeManager` class acts as both the View and Controller, interacting with `Student` and `Course` objects as the Model.

Class Diagram (Conceptual)

RunCopy code

```
+-----------------------+       +------------------+       +------------------+
| ClassGradeManager     |       | Student          |       | Course           |
+-----------------------+       +------------------+       +------------------+
| - students: ArrayList |       | - name: String   |       | - name: String   |
| - currentStudent: Student |   | - numCourses: int |      | - credits: double |
| - UI Components       |       | - courses: ArrayList<Course> | | - grade: String |
+-----------------------+       | - gpa: double    |       +------------------+
| + ClassGradeManager() |       +------------------+       | + Course()       |
| + applyDarkTheme()    |       | + Student()      |       | + getName()      |
| + createStyledButton()|       | + addCourse()    |       | + getCredits()   |
| + addStudent()        |       | + getName()      |       | + getGrade()     |
| + addCourse()         |       | + getNumCourses() |      +------------------+
| + nextStudent()       |       | + getCourses()   |
| + isValidGrade()      |       | + getGpa()       |
| + calculateStudentGPA() |     | + setGpa()       |
| + convertGradeToPoints() |    +------------------+
| + clearAll()          |
| + showStudentSummary() |
| + showClassStatistics() |
+-----------------------+
```

Component Breakdown

- `ClassGradeManager` **(Main Application Class)**:
  - Manages the main `JFrame` and its layout (`BorderLayout`).
  - Contains `JPanel` instances for input, results, and student-specific course entry.
  - Holds `JTextField` components for user input and `JTextArea` for output.
  - Manages `JButton` components and their associated `ActionListener` implementations.
  - Orchestrates the flow of data between UI components and the `Student`/`Course` models.
  - Maintains an `ArrayList` of `Student` objects to keep track of all enrolled students.
- `Student` **Class**:
  - Encapsulates student-specific data: name, expected number of courses, and a dynamic list of `Course` objects.
  - Responsible for adding courses to its internal list.
  - Stores its calculated GPA.

- `Course` **Class**:
    - Encapsulates course-specific data: name, credit hours, and the assigned letter grade.
    - A simple data holder (POJO - Plain Old Java Object).

UI/UX Design Principles

- **Clarity and Simplicity**: The layout is straightforward, with distinct panels for input and results.
- **Consistency**: Buttons and input fields maintain a consistent style and color scheme.
- **Feedback**: The `resultArea` provides immediate feedback on actions (e.g., student added, course added, GPA calculated). `JOptionPane` is used for error and informational messages.
- **Accessibility**: The dark theme aims to reduce eye strain, and clear labels are provided for all input fields.
- **Progressive Disclosure**: The `studentPanel` (for course input) and `addCourseButton`/`nextStudentButton` are initially hidden and become visible only after a student is successfully added, reducing initial clutter.

4. Technical Implementation Details

Core Logic: GPA Calculation

The GPA calculation is performed in the `calculateStudentGPA()` method. It iterates through all `Course` objects associated with the `currentStudent`.

For each course, it:

1. Retrieves the `grade` and `credits`.
2. Converts the `grade` to numerical quality points using `convertGradeToPoints()`.
3. Accumulates `totalQualityPoints` (Grade Points × Credits) and `totalCredits`.
4. Tracks the `bestGrade` and `worstGrade` based on quality points to identify the best and worst performing courses.

The GPA is then calculated using the standard formula:GPA=Total Quality PointsTotal Credits

Grade Conversion Logic

The `convertGradeToPoints(String grade)` method is crucial for GPA calculation. It uses a `switch` statement to map common letter grades (including plus/minus variations) to their standard 4.0 scale quality points.

Grade	Quality Points
A+, A	4.0
A-	3.7
B+	3.3
B	3.0
B-	2.7
C+	2.3
C	2.0
C-	1.7
D+	1.3
D	1.0
D-	0.7
F	0.0

The `isValidGrade(String grade)` method uses a regular expression `[A-F][+-]?` to validate the format of the entered grade, ensuring it adheres to expected letter grade patterns.

Data Structures

- `ArrayList<Student> students`: Stores all `Student` objects added to the system, allowing for class-wide statistics.
- `ArrayList<Course> courses` (within `Student` class): Stores all `Course` objects for a particular student.

UI Styling and Theming

The application utilizes the **Nimbus Look and Feel** for a modern appearance. The `applyDarkTheme()` method customizes Nimbus's default colors to achieve a dark theme:

- `control`, `info`, `nimbusLightBackground`: Set to dark grey (`Color(30, 30, 30)`).

- `text`, `nimbusSelectedText`: Set to white/light grey for readability.
- `nimbusRed`: Used for button backgrounds and titled border lines (`Color(169, 46, 34)`), providing a consistent accent color.

Custom `JButton` styling is applied via `createStyledButton()` to ensure uniform appearance, including background color, foreground color, font, and border.

Input Validation

Robust input validation is implemented to prevent errors and guide the user:

- **Empty Fields**: Checks if `studentNameField`, `numCoursesField`, `courseNameField`, `creditField`, and `gradeField` are empty.
- **Numeric Input**: Uses `Integer.parseInt()` and `Double.parseDouble()` within `try-catch` blocks to handle `NumberFormatException` for `numCoursesField` and `creditField`.
- **Positive Values**: Ensures `numCourses` and `credits` are greater than zero.
- **Valid Grades**: `isValidGrade()` checks if the entered grade matches the `[A-F][+-]?` pattern.
- **Contextual Validation**: Prevents GPA calculation or summary display if no courses have been added for the current student.

All validation failures trigger a `JOptionPane.showMessageDialog()` with an appropriate error message.

5. User Guide

Getting Started

1. Run the `ClassGradeManager.java` file.
2. The application window will appear with a dark theme.

Adding a Student

1. In the "Student Information" panel:
   - Enter the student's full name in the "Student Name:" field.
   - Enter the total number of courses this student will take in the "Number of Courses:" field.
2. Click the "Add Student" button.
3. The `resultArea` will confirm the student's addition and prompt you to add courses. The "Add Course for Current Student" panel will become visible.

Adding Courses

1. After adding a student, in the "Add Course for Current Student" panel:
   - Enter the course name in the "Course Name:" field.
   - Enter the credit hours for the course (e.g., 3.0, 4.5) in the "Credit Hours:" field.
   - Enter the letter grade received (e.g., A, B+, C-, F) in the "Grade (A-F):" field.
2. Click the "Add Course" button.
3. The `resultArea` will confirm the course addition.
4. Repeat steps 1-3 until all courses for the current student have been added. The "Add Course" button will disable once the specified number of courses is reached.

Calculating GPA

1. Once all courses for a student are added (or at any point after adding at least one course), click the "Calculate GPA" button.

2. The `resultArea` will display the student's GPA, total courses, total credits, and their best and worst course grades.

Viewing Student Summary

1. To see a detailed list of all courses for the current student, click the "Show Summary" button.
2. The `resultArea` will display a table showing each course's name, credits, grade, and calculated quality points.

Viewing Class Statistics

1. After adding multiple students and their courses, click the "Class Statistics" button.
2. The `resultArea` will display aggregated data for all students, including:
    - Total number of students.
    - Average GPA of the class.
    - Highest and lowest GPAs, along with the student names.
    - A distribution of all grades entered across all students.

Clearing Data

1. To clear all student data, course data, and reset the application to its initial state, click the "Clear All" button.
2. The `resultArea` will be cleared, and input fields will be reset.

6. Future Enhancements

- **Data Persistence**: Implement saving and loading student data to/from a file (e.g., CSV, JSON, or binary serialization) to retain information between application sessions.
- **Student Selection**: Add a mechanism (e.g., a `JComboBox` or `JList`) to select and switch between previously added students without needing to re-enter their data.
- **Course Editing/Deletion**: Allow users to modify or remove existing courses for a student.
- **Error Handling Improvements**: More granular error messages and visual cues for invalid input fields.
- **Reporting**: Generate printable reports of student summaries or class statistics.
- **User Accounts**: Implement basic user authentication for multiple users to manage their own classes.
- **Advanced Statistics**: Add more statistical analysis, such as standard deviation of GPAs, or performance trends.

7. Contact Information

For support, inquiries, or contributions, please contact:
Shehryar Ahmed Khan CodeAlpha Internship GitHub: www.github.com/n1x5-slayer