Java Final Exam

✓ **Key Features and Functionalities**

1) **Student Information Input**: Users should be able to input basic student information such as name, student ID, and class.
2) **Subject and Grade Input:** Allow users to add different subjects and input grades for each subject.
3) **Calculations:** Calculate and display the average grade for each subject.

Calculate and display the overall GPA (Grade Point Average) or final percentage based on the entered grades.

4) **Grade Analysis:** Provide statistical analysis of grades, such as minimum, maximum, and standard deviation. Highlight areas of improvement or commendation based on the performance.
5) **Final Grade Determination:** Implement a grading system that converts overall averages into final letter grades or performance categories.
6) **User Authentication and Data Security:** implement user authentication to secure the data and ensure privacy. Safeguard against unauthorized access to student records.
7) **Reporting:** Generate printable or exportable reports summarizing student performance and grades.
8) **User-Friendly Interface:** Design an intuitive interface for ease of use.

Consider tooltips or help sections to guide users through the functionalities.

9) **Data Storage and Retrieval:** Allow for the storage and retrieval of student information and grades. Consider options for saving data locally or on a server, depending on the deployment environment.

How Users will interact with the application

Graphical User Interface (GUI):

- A graphical interface provides a more user-friendly experience.
- Users interact with the program through buttons, forms, and visual elements.

Java Frameworks that can be used

1. **JavaFX:**

- If you opt for a graphical user interface (GUI), JavaFX is a robust framework for building desktop applications. It provides a set of APIs for creating rich user interfaces.

2. **Spring Framework:**
    - Spring is a comprehensive framework that can be utilized for various aspects of the application, including dependency injection, data access, and security. Spring Boot, a part of the Spring framework, is particularly useful for building standalone applications with minimal configuration.

Student Information Storage and Organization:

Database (e.g., MySQL)

- Suitable for applications requiring persistent and structured storage.
- Allows for efficient querying, indexing, and retrieval of student data.
- Ensures data integrity and supports concurrent access.

Attributes each student will have

- studentID: A unique identifier for each student.
- First Name and last Name: The first and last name of the student.
- Age: The age of the student.
- Class Name: The class or grade level of the student (e.g., "10th Grade," "Sophomore," etc.).
- Grades: A Map to store subject grades, where the key is the subject name, and the value is the corresponding grade.

How users will add students

Graphical User Interface (GUI):

For a GUI, you would typically design a form where users can input the necessary information for a new student. Here's an example of how the "Add Student" form looks:

Explanation:

Users can enter the student's first name, last name, ID, age, and class.

There are buttons to add the student or cancel the operation.

<u>How the application will validate user input and handle potential errors</u>

Input Validation:

- **Required Fields**: Ensure that mandatory fields (e.g., student ID, name) are not left empty.
- **Data Format**: Validate inputs to match expected formats (e.g., numeric for age, proper format for student ID).
- **Range Checking:** Check that values fall within acceptable ranges (e.g., age must be a positive integer).
- **Data Consistency**: Verify that inputs are consistent and coherent (e.g., class name matches predefined options).
- **Error Messages:** Provide informative error messages to guide users on correcting their inputs.

Handling Errors and Exceptions:

- **Exception Handling:** Implement try-catch blocks to catch and handle exceptions gracefully.
- **User-Friendly Messages**: Display clear and concise error messages to users, indicating what went wrong and how to rectify it.
- **Logging**: Log detailed error information to aid in debugging and troubleshooting.
- **Graceful Degradation:** Design the application to gracefully handle errors without crashing or losing data.
- **Fallback Mechanisms**: Implement fallback mechanisms or default values to handle unexpected situations.

<u>How grades will be recorded and calculated</u>

To record and calculate student grades, we'll need a data structure to store the grades for each student and algorithms to calculate various metrics such as average grades, final grades, etc.

Data Structure:

We can use a map to store the grades for each student, where the key represents the subject name, and the value represents the grade. The student object will contain this map of grades.

Grade Calculation Algorithms:

Average Grade Calculation:

- Iterate over the grades of a student and calculate the average.
- Sum all grades and divide by the total number of subjects.

Final Grade Determination:

- Define a grading system with grade ranges mapped to final grades (e.g., A, B, C).
- Calculate the overall average grade using the above method.
- Determine the final grade based on the grading system.

How the application calculate the overall class Average

To calculate the overall class average, we need to aggregate the grades of all students in a class and compute the average. Here's how we can implement the necessary functionality:

Class for Managing Students and Grades:

First, let's define a class to manage students and their grades. This class will contain a list of students, and methods to add students, record grades, and calculate the class average.

How the application will display student grades and averages

Graphical User Interface (GUI):

For a GUI application, we can create a window where users can select a student and view their grades and average. Here's a simple design:

Components:

Student List: Display a list of students for selection.

Grades Table: Show the grades of the selected student, including subjects and corresponding grades.

Average Display: Display the average grade of the selected student.

<u>How the application generates reports based on thresholds or student performance</u>

To generate reports based on grade thresholds or student performance, the application can provide functionality to filter and analyze student data. Here's how we can implement this:

Graphical User Interface (GUI):

For a GUI application, we can create a reporting section where users can set grade thresholds or select criteria for generating reports. Here's a design:

Components:

Threshold Inputs: Allow users to set grade thresholds for different performance levels (e.g., A, B, C).

Report Criteria: Provide options for selecting criteria such as subject, class, or date range.

Generate Report Button: Trigger the generation of the report based on the specified criteria.

Report Output: Display the generated report, including statistics and insights based on the selected criteria.

<u>How the application will handle persistence</u>

- By using a database.

  <u>Test Plan:</u>
  1. Unit Tests:
  Student Class:
  Test adding grades for a student.
  Test calculating the average grade for a student.
  ClassManager Class:
  Test adding students to the class.
  Test recording grades for students.
  Test calculating the class average.
  2. End-to-End Tests:
  GUI Application:

Test adding a new student through the GUI.
Test recording grades for the added student through the GUI.
Test viewing individual student records through the GUI.
Test generating reports based on grade thresholds or student performance through the GUI.

CLI Application:

Test adding a new student through CLI commands.
Test recording grades for the added student through CLI commands.
Test viewing individual student records through CLI commands.
Test generating reports based on grade thresholds or student performance through CLI commands.

Test Execution:

1. Unit Tests:

Use JUnit or another testing framework to write and execute unit tests for individual components (Student class, ClassManager class).
Validate that each method behaves as expected and returns the correct results.

2. **End-to-End Tests:**

GUI Application:
Manually test each feature of the GUI application, following predefined test cases.
Verify that adding students, recording grades, viewing student records, and generating reports work as expected.
CLI Application:
Manually test each feature of the CLI application, executing predefined CLI commands.
Verify that adding students, recording grades, viewing student records, and generating reports work as expected.

<u>Documenting the application features, architecture and usage instruction</u>

Application Architecture:
The Student Grade Calculator application is built using Java and follows a modular architecture. It consists of the following components:

Student Class:
Represents a student and encapsulates student information such as name, ID, grades, etc.
ClassManager Class:
Manages students and their grades, providing functionalities to add students, record grades, and calculate class averages.
GUI (Graphical User Interface) Component:
Provides a graphical interface for users to interact with the application. It includes forms for adding students, recording grades, viewing student records, and generating reports.
CLI (Command-Line Interface) Component:
Allows users to interact with the application via command-line commands. It provides commands for adding students, recording grades, viewing student records, and generating reports.
Features:
Student Management:
Add new students with basic information such as name, ID, age, and class.
Grade Recording:
Record grades for students in different subjects.
Grade Calculation:
Calculate the average grade for each student and the overall class average.
Individual Student Records:
View individual student records, including their grades and average.
Report Generation:
Generate reports based on grade thresholds or student performance, providing statistical insights and analysis.
Usage Instructions:
GUI Application:
Launch the GUI application.
Use the provided forms to add new students, record grades, view student records, and generate reports.
Follow on-screen instructions and input necessary information.
Navigate through different sections using the provided buttons or tabs.
CLI Application:

Open the command-line interface (CLI).
Use predefined commands to interact with the application:
addstudent: Add a new student.
recordgrade: Record grades for a student.
viewstudent: View individual student records.
generatereport: Generate reports based on specified criteria.
Provide required parameters with each command (e.g., student ID, subject, grade thresholds).
Execute the commands and follow the displayed output or instructions.
System Requirements:
Java Runtime Environment (JRE) version 8 or higher.
Notes:
Ensure proper input validation and error handling to handle unexpected situations gracefully.
Save data persistently if necessary (e.g., using file storage or a database) to retain student information across sessions.

Example Usage:

GUI Application:

Launch the application.
Click on the "Add Student" button to add a new student.
Enter the student's information and click "Save."
Record grades for the added student using the "Record Grade" section.
View the student's records or generate reports as needed.

CLI Application:

Open the command-line interface.
Use commands such as addstudent, recordgrade, viewstudent, and generatereport with appropriate parameters.
Follow the prompts or instructions provided by the application.
Review the output or reports generated by the application.
Conclusion:
The Student Grade Calculator application provides a user-friendly interface for managing student grades and generating insightful reports. Whether using the GUI or CLI, users can efficiently record grades, analyze student performance, and gain valuable insights to support decision-making processes in academic settings.

<u>SECTION B</u>

<u>The key features of the Student Grade Calculator application include:</u>

Student Management:

Ability to enter and manage student information, including name, ID, age, and class.

Grade Recording:

Functionality to input grades for students in different subjects.

Grade Calculation:

Capability to calculate the average grade for each student based on their recorded grades.Calculation of the overall class average grade.

Individual Student Records:

Ability to view individual student records, including their grades, average grade, and other relevant information.

Report Generation:

Functionality to generate reports based on predefined grade thresholds or student performance criteria.

Reports should include statistical insights and analysis, such as grade distributions, average grades, highest/lowest grades, etc.

User Interface:

Graphical User Interface (GUI) or Command-Line Interface (CLI) for user interaction, depending on preference and application type.

Intuitive and user-friendly interface design to facilitate ease of use and navigation.

Input Validation and Error Handling:

Comprehensive input validation to ensure the accuracy and integrity of data entered by users.

Effective error handling mechanisms to gracefully manage unexpected scenarios and provide informative error messages to users.

Persistence:

Persistence of student data to retain information across sessions, ensuring that data is not lost when the application is closed or restarted.