Java Exams Answers

Ngane Emmanuel

2024-01-27

# 1 Section A

## 1.1 Question: What are the key features and functionalities required for the Student Grade Calculator application?

The key features and functionalities required for the Student Grade Calculator application are:

* Entering student information
* Inputting grades for different subjects
* Calculating overall averages and final grades
* Generating reports
* Viewing and analyzing student performance

## 1.2 Question: How will the user interact with the application? Will it be through a command-line interface or a graphical user interface (GUI)?

The user will interact with the application through a graphical user interface (GUI).

## 1.3 Question: What Java libraries or frameworks can be used to simplify the development process?

To simplify the development process of the Student Grade Calculator application in Java, libraries or frameworks such as Java Swing or JavaFX can be used for creating the graphical user interface (GUI) if chosen, and JUnit can be used for unit testing.

## 1.4 Question: How will student information be stored and organized? Will you use a database, file storage, or an in-memory data structure?

A database will be used to store student information

## 1.5 Question: What attributes should each student have (e.g., name, ID, grades)? Design the student object and its properties.

Each student should have attributes such as Name, ID, Grades, Attendance Scores, participation scores.

| Member | Type | Methods | Setter | Average |
| --- | --- | --- | --- | --- |
| name | String | getName(): String | setName(name: String): void | calculateAverage(): double |
| ID | String | getID(): String | setID(ID: String): void | calculateAverage(): double |
| grades | List | getGrades(): List | setGrades(grades: List): void | calculateAverage(): double |

## 1.6 Question: How will users add new students? Design the user interface or command-line commands for adding students.

Users can add new students by providing the necessary information through the user interface. A form will be displayed to fill in the information of the new user and validation will be for making sure the information entered meets the standard and also to avoid duplicate user entery.

## 1.7 Question: How will the application validate user inputs and handle potential errors or exceptions?

The application will validate user inputs and handle potential errors or exceptions in the following ways:

* **Input Validation**: Robust validation logic will be im[plemented to ensure user input meets the required criteria. Some major areas include checking for empty fields, validating input formats (e.g., email addresses, data types), and verifying data ranges (e.g., age).
* **Clear Error Messages**: Provide descriptive error messages that clearly communicate the nature of the error. Display them near the relevant form fields or in a dedicated error message area for easy visibility.
* **Real-Time Validation**: Consider implementing real-time validation to provide immediate feedback as users interact with the form. This allows them to correct mistakes promptly.
* **Form Submission**: Perform a final round of validation when the user submits the form. Ensure all required fields are filled correctly. If errors are found, display an aggregated error message summarizing the issues and allowing users to make corrections.
* **Exception Handling**: Implement robust exception handling to gracefully handle unexpected errors during data processing and external dependencies. Provide informative error messages to guide users in understanding and resolving any issues.

## 1.8 Question: How will grades be recorded and calculated? Design the data structure and algorithms to store and calculate student grades

Grades can be recorded and calculated using a suitable data structure like an ArrayList to store students and their respective grades.

### 1.8.1 Grade Calculation Algorithm

public class GradeCalculator {  
 public static void main(String[] args) {  
 int[] scores = {85, 90, 78, 92, 88}; // Example scores  
   
 double average = calculateAverage(scores);  
 char grade = calculateGrade(average);  
   
 System.out.println("Average Score: " + average);  
 System.out.println("Grade: " + grade);  
 }  
   
 public static double calculateAverage(int[] scores) {  
 int sum = 0;  
 for (int score : scores) {  
 sum += score;  
 }  
   
 return (double) sum / scores.length;  
 }  
   
 public static char calculateGrade(double average) {  
 if (average >= 90) {  
 return 'A';  
 } else if (average >= 80) {  
 return 'B';  
 } else if (average >= 70) {  
 return 'C';  
 } else if (average >= 60) {  
 return 'D';  
 } else {  
 return 'F';  
 }  
 }  
}

This algorithm calculates grades based on a set of scores. It includes two methods: calculateAverage to calculate the average score, and calculateGrade to determine the corresponding grade based on the average score. The program then prints the average score and the grade.

## 1.9 Question: How will the application calculate the overall class average? Implement the necessary functionality for calculating class averages.

The application can calculate the overall class average by considering the weighted averages of all students.

### 1.9.1 Java Class Average Calculation

import java.util.Scanner;  
  
public class ClassAverageCalculator {  
 public static double calculateClassAverage(int[] scores) {  
 int sum = 0;  
 for (int score : scores) {  
 sum += score;  
 }  
   
 return (double) sum / scores.length;  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
   
 System.out.print("Enter the number of students: ");  
 int numStudents = scanner.nextInt();  
   
 int[] scores = new int[numStudents];  
   
 // Prompt for and read the scores for each student  
 for (int i = 0; i < numStudents; i++) {  
 System.out.print("Enter the score for student " + (i + 1) + ": ");  
 scores[i] = scanner.nextInt();  
 }  
   
 double classAverage = ClassAverageCalculator.calculateClassAverage(scores);  
   
 System.out.println("Class Average: " + classAverage);  
   
 scanner.close();  
 }  
}

## 1.10 Question: How will the application display student grades and averages? Design the user interface or command-line interface for viewing individual student records.

The application can display student grades and averages through the user interface. It will prompt the user to select a student and display the student’s details, grades, and average.

## 1.11 Question: How will the application generate reports based on grade thresholds or student performance? Implement the necessary functionality for generating reports.

The application can generate reports based on grade thresholds or student performance by displaying students with grades below a certain threshold, sorting students by their averages, and identifying the worst-performing students. *The implementation can vary based on the specific requirements of the reports.*

## 1.12 Question: How will the application handle persistence? If using a database or file storage, implement the necessary code to store and retrieve student information.

### 1.12.1 Application Persistence with JPA and Database

To handle persistence and store/retrieve student information in the application, we will use a database with JPA (Java Persistence API). Here’s an outline of the steps involved:

#### 1.12.1.1 1. Define the Student Entity

First, we will define the Student entity class. This class will represent the student information and will be mapped to a corresponding table in the database. Here’s how the Student class is defined:

import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
  
@Entity  
public class Student {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
   
 private String name;  
 private int score;  
   
 // Getters and setters...  
}

#### 1.12.1.2 Set up Database Connection

Next, we will set up the database connection configuration. This typically involves specifying the database URL, username, password, and other connection properties. We configured these properties in a separate configuration file persistence.xml

#### 1.12.1.3 Implement Data Access Object (DAO) Class

To interact with the database and perform CRUD operations on the Student entity, we will create a Data Access Object (DAO) class. Here’s is how the StudentDAO class is implemented:

import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import org.springframework.stereotype.Repository;  
import org.springframework.transaction.annotation.Transactional;  
  
@Repository  
@Transactional  
public class StudentDAO {  
 @PersistenceContext  
 private EntityManager entityManager;  
   
 public void createStudent(Student student) {  
 entityManager.persist(student);  
 }  
   
 public Student getStudentById(Long id) {  
 return entityManager.find(Student.class, id);  
 }  
   
 public void updateStudent(Student student) {  
 entityManager.merge(student);  
 }  
   
 public void deleteStudent(Long id) {  
 Student student = entityManager.find(Student.class, id);  
 entityManager.remove(student);  
 }  
}

#### 1.12.1.4 Usage in Application

To use the StudentDAO class in the application, we can inject it as a dependency in other components or services. Here’s how it is used:

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
@Service  
public class StudentService {  
 private final StudentDAO studentDAO;  
   
 @Autowired  
 public StudentService(StudentDAO studentDAO) {  
 this.studentDAO = studentDAO;  
 }  
   
 public void createStudent(Student student) {  
 studentDAO.createStudent(student);  
 }  
   
 public Student getStudentById(Long id) {  
 return studentDAO.getStudentById(id);  
 }  
   
 public void updateStudent(Student student) {  
 studentDAO.updateStudent(student);  
 }  
   
 public void deleteStudent(Long id) {  
 studentDAO.deleteStudent(id);  
 }  
}