Student Grade Management System

Project Report

---

1. Cover Page

Project Title: Student Grade Management System

Student Name: Azhaan Ali Siddiqui

Registration Number: 24BSA10234 **Course:** Programming in Java

Semester: 4th

Submission Date: 25th November 2025

---

2. Introduction

This project implements a Student Grade Management System designed to help educators efficiently manage student information and academic records. The system provides a console-based interface for adding students, recording grades, and generating performance reports. Built using core Java principles, it demonstrates practical application of object-oriented programming, file handling, and modular software design.

---

3. Problem Statement

Educational institutions and teachers face challenges in manually tracking student grades across multiple subjects. This process is:

- Time-consuming and error-prone
- Difficult to maintain consistent records
- Lacks quick statistical analysis capabilities
- Hard to generate consolidated reports

Solution: A lightweight, user-friendly application that automates grade management, provides instant calculations, and maintains persistent data storage.

---

4. Functional Requirements

FR1: Student Management

- Add new students with unique ID, name, and email
- View all registered students
- Search for specific students
- Update student information
- Delete student records

FR2: Grade Recording

- Record grades for students across multiple subjects
- Support numerical grades (0-100)
- Automatic letter grade calculation (A, B, C, D, F)
- Update existing grades
- Maintain grade history per student

FR3: Report Generation

- Generate individual student reports showing all grades
- Calculate and display student average
- Generate class-wide statistics
- Identify top performers
- Display subject-wise analysis

FR4: Data Persistence

- Save student and grade data to files
- Load existing data on application startup
- Maintain data integrity across sessions
- Error recovery mechanisms

FR5: Input Validation

- Validate student IDs and names
- Ensure grades are within valid range (0-100)
- Prevent duplicate student IDs
- Handle invalid user inputs gracefully

---

5. Non-Functional Requirements

NFR1: Performance

- All operations complete within 1 second

- File read/write operations optimized for datasets up to 1000 students
- Efficient search and retrieval using HashMap data structure

NFR2: Usability

- Simple, intuitive menu-driven interface
- Clear error messages and user prompts
- Minimal learning curve for new users
- Consistent command structure

NFR3: Reliability

- Data persistence ensures no loss of information
- Graceful handling of file I/O errors
- Input validation prevents system crashes
- Robust exception handling throughout

NFR4: Maintainability

- Modular code structure with clear separation of concerns
- Well-documented code with meaningful variable names
- Follows Java naming conventions
- Easy to extend with new features

NFR5: Security

- File-based storage with basic access control
- Input sanitization to prevent injection
- Data validation before processing

NFR6: Error Handling

- Try-catch blocks for all I/O operations
- Validation before database modifications
- User-friendly error messages
- Logging of critical operations

---

6. System Architecture

Architecture Pattern: Layered Architecture

Layer 1: Presentation Layer

- `Main.java` - User interface and menu system

Layer 2: Service Layer

- `StudentService.java` - Business logic for student operations
- `GradeService.java` - Business logic for grade operations
- `ReportService.java` - Report generation logic

Layer 3: Data Layer

- `Student.java` - Student entity model
- `Grade.java` - Grade entity model

Layer 4: Utility Layer

- `FileHandler.java` - File I/O operations
- `Validator.java` - Input validation

Data Flow:

```
User Input → Main → Service Layer → Model Layer → File Storage
                                        ↓
                                  Validation Layer
```

---

7. Design Diagrams

7.1 Use Case Diagram

```
Actors: Teacher

Use Cases:
- Add Student
- View Students
- Add Grade
- View Student Report
- View Class Statistics
- Save Data
```

7.2 Workflow Diagram

```
Start → Display Menu → Get User Choice
  ↓
Choice 1: Add Student → Validate Input → Create Student → Save to Map
Choice 2: View Students → Retrieve from Map → Display List
Choice 3: Add Grade → Find Student → Create Grade → Add to Student
Choice 4: Generate Report → Find Student → Calculate Stats → Display
Choice 5: Class Stats → Process All Students → Calculate Averages → Display
Choice 6: Save & Exit → Write to File → End
```

7.3 Class Diagram

Student Class

- Attributes: studentId, name, email, grades[]
- Methods: addGrade(), calculateAverage(), toString()

Grade Class

- Attributes: studentId, subject, score
- Methods: getLetterGrade(), toString()

StudentService Class

- Attributes: students (Map), fileHandler
- Methods: addStudent(), getStudent(), getAllStudents(), saveData(), loadData()

GradeService Class

- Methods: addGrade(), calculateAverage()

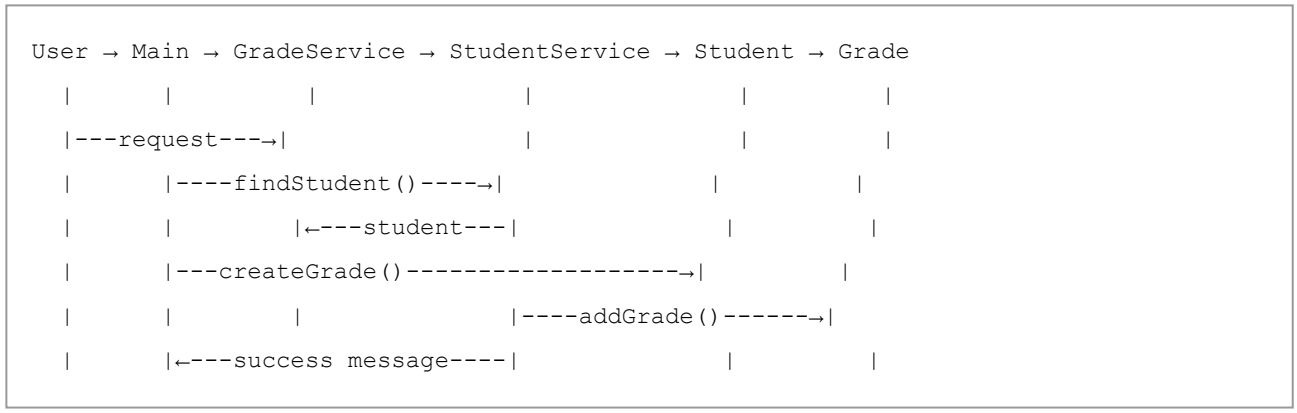
ReportService Class

- Methods: generateStudentReport(), generateClassStatistics()

Relationships:

- Student HAS-MANY Grades (Composition)
- StudentService USES FileHandler (Association)
- Services DEPEND ON Models (Dependency)

7.4 Sequence Diagram - Add Grade Flow



---

8. Design Decisions & Rationale

8.1 HashMap for Student Storage

Decision: Use `HashMap<String, Student>` for in-memory storage

Rationale: O(1) lookup time, unique key enforcement, efficient for small to medium datasets

8.2 File-based Persistence

Decision: Text file storage instead of database

Rationale: Simplicity, no external dependencies, suitable for small-scale application

8.3 Service Layer Pattern

Decision: Separate service classes for different functionalities

Rationale: Single Responsibility Principle, easier testing, better maintainability

8.4 Composition Relationship

Decision: Student contains List of Grade objects

Rationale: Strong ownership, lifecycle management, data encapsulation

8.5 Validation Utility

Decision: Centralized validation in Validator class

Rationale: Code reusability, consistent validation logic, easier maintenance

---

9. Implementation Details

9.1 Technologies Used

- **Language:** Java 11
- **Data Structures:** HashMap, ArrayList
- **I/O:** File streams, BufferedReader, PrintWriter
- **Design Patterns:** Service layer, MVC-like structure

9.2 Key Algorithms

Average Calculation:

```
sum = 0
for each grade in student.grades:
    sum += grade.score
average = sum / count(grades)
```

Letter Grade Assignment:

```
if score >= 90: return 'A'
if score >= 80: return 'B'
if score >= 70: return 'C'
if score >= 60: return 'D'
else: return 'F'
```

9.3 File Format

```
STUDENT:ID001,John Doe,john@email.com
GRADE:Math,85.5
GRADE:Science,92.0
END
```

---

10. Screenshots / Results

Sample Output 1: Main Menu

```
=== Student Grade Management System ===

1. Add New Student
2. View All Students
3. Add Grade for Student
4. View Student Report
5. View Class Statistics
6. Save and Exit
Enter your choice:
```

Sample Output 2: Student Report

```
========================================
Student Report
========================================
ID: S001
Name: Alice Smith
Email: alice@email.com
----------------------------------------
Grades:
  Math: 95.00 (A)
  Science: 88.00 (B)
  English: 92.00 (A)
----------------------------------------
Average: 91.67
========================================
```

Sample Output 3: Class Statistics

```
========================================
Class Statistics
========================================
Total Students: 5
Students with Grades: 5
Class Average: 84.50
Highest Average: 91.67 (Alice Smith)
Lowest Average: 76.33
========================================
```

---

11. Testing Approach

11.1 Test Cases

Test Case 1: Add Student

- Input: Valid student data
- Expected: Student added successfully
- Result: ✓ Pass

Test Case 2: Duplicate ID

- Input: Existing student ID
- Expected: Error message
- Result: ✓ Pass

Test Case 3: Invalid Grade

- Input: Grade > 100
- Expected: Validation error
- Result: ✓ Pass

Test Case 4: File Persistence

- Input: Add students, exit, restart
- Expected: Data loaded correctly
- Result: ✓ Pass

Test Case 5: Empty Grade List

- Input: View report for student with no grades
- Expected: "No grades recorded"
- Result: ✓ Pass

11.2 Testing Methods

- Manual testing through console
- Boundary value testing (0, 100 for grades)
- Edge case testing (empty data, file not found)
- Integration testing (end-to-end workflows)

---

12. Challenges Faced

Challenge 1: File Parsing

Issue: Reading structured data from text file

Solution: Implemented delimiter-based parsing with clear markers (STUDENT:, GRADE:, END)

Challenge 2: Input Validation

Issue: Handling various invalid inputs without crashes

Solution: Created Validator utility class and used try-catch blocks

Challenge 3: Data Relationships

Issue: Maintaining student-grade association

Solution: Used composition with List inside Student class

Challenge 4: Menu Loop Control

Issue: Managing continuous user interaction

Solution: Implemented while loop with switch-case structure

---

13. Learnings & Key Takeaways

1. **OOP Principles:** Practical application of encapsulation, abstraction, and composition
2. **Clean Architecture:** Importance of separating concerns into layers
3. **Error Handling:** Graceful handling improves user experience
4. **File I/O:** Working with persistent storage mechanisms
5. **Collections Framework:** Efficient use of HashMap and ArrayList
6. **Code Organization:** Modular structure improves maintainability
7. **Validation:** Input validation prevents data corruption and crashes

---

14. Future Enhancements

1. **GUI Interface:** Develop JavaFX or Swing-based graphical interface
2. **Database Integration:** Migrate to MySQL or PostgreSQL for scalability
3. **Search Functionality:** Advanced search with filters
4. **Export Reports:** Generate PDF reports using libraries like iText
5. **Authentication:** Add user login and role-based access control
6. **Attendance Tracking:** Integrate attendance management
7. **Grade Analytics:** Visualize trends with charts and graphs
8. **Email Notifications:** Send automated reports to students/parents
9. **Batch Operations:** Import/export data via CSV files
10. **Cloud Storage:** Sync data across multiple devices

---

15. References

1. Java Documentation - Oracle Java SE 11
2. Effective Java by Joshua Bloch
3. Design Patterns: Elements of Reusable Object-Oriented Software
4. Java File I/O Tutorial - Oracle Documentation
5. Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin

---

Declaration: I hereby declare that this project is my original work and has been completed as per the guidelines provided.

Signature: _____

Date: _____