**Student Management System Documentation**

**Table of Contents**

**1.   Problem Statement**

**2.   Introduction**

**3.   Objectives**

**4.   Features**

**5.   System Design**

**6.   Class Diagrams**

**7.   Implementation Details**

**8.   Limitations**

**9.   Future Enhancements**

**10.   Conclusion**

1. **Problem Statement**

Educational institutions require an efficient system to manage student records, examination details, and performance tracking. Manual record-keeping is error-prone, time-consuming, and lacks real-time accessibility.

**Key Challenges:**

* Difficulty in maintaining accurate student records.
* Manual calculation of grades and percentages leads to errors.
* No centralized system to track academic performance.
* Inefficient examination result management.

**Solution:**

A   Student Management System (SMS)   that:

* Stores student details (name, roll number, DOB, email, course).
* Manages examination records (subject-wise marks, grades, percentage).
* Automates grade calculation and performance analysis.
* Provides a command-line interface for easy interaction.

1. **Introduction**

This project is a   Java-based Student Management System   developed using   Object-Oriented Programming (OOP)   principles. It eliminates manual record-keeping by providing a digital solution for managing student data and examination results.

**Key Aspects:**

* -   No File Handling:   Uses in-memory storage (arrays) for simplicity.
* -   Command-Line Interface (CLI):   Easy to use without a GUI.
* -   Automated Grading System:   Calculates grades and percentages.
* -   Modular Design:   Follows OOP concepts (Encapsulation, Abstraction, Composition).

**3. Objectives**

* 1.   Store Student Data:   Name, roll number, email, course, etc.
* 2.   Manage Examinations:   Record marks for 5 subjects.
* 3.   Automate Grading:   Calculate grades (A+, A, B, etc.) and percentage.
* 4.   Display Results:   View student details and exam performance.
* 5.   Avoid Manual Errors:   Reduce human intervention in calculations.

**4. Features**

* | Feature | Description |
* |---------|------------|
* |   Add Student   | Stores student details (name, roll no., email, course). |
* |   Add Exam Details   | Records marks for 5 subjects and calculates grades. |
* |   View Student Info   | Displays student details using roll number. |
* |   View Exam Results   | Shows subject-wise marks, grades, and overall percentage. |
* |   Grade Automation   | Converts marks into grades (A+, A, B, etc.). |
* |   In-Memory Storage   | Uses arrays (no file/database dependency). |

**5. System Design**

* 5.1 Class Structure

1.   `Student` Class

   - Stores personal details (name, roll no., email, course).

   - Methods: `displayDetails()`.

2.   `Subject` Class

   - Manages subject name, marks, grade, and percentage.

   - Methods: `calculateGradeAndPercentage()`.

3.   `Examination` Class

   - Stores 5 subjects, calculates overall performance.

   - Methods: `addSubject()`, `calculateOverallPerformance()`.

4.   `StudentManagementSystem` (Main Class)

   - CLI menu for user interaction.

   - Manages student and exam records.

      5.2 Flowchart

```

1. Start Program

2. Display Menu:

   - Add Student

   - Add Exam

   - View Student

   - View Exam

   - Exit

3. Perform Selected Operation

4. Repeat Until Exit

**6. Class Diagrams**

      Student Class

```java

class Student {

    - name: String

    - rollNumber: int

    - dob: String

    - email: String

    - studentId: String

    - course: String

    + displayDetails()

}

```

      Examination Class

```java

class Examination {

    - subjects: Subject[]

    - overallPercentage: double

    - overallGrade: String

    + addSubject()

    + calculateOverallPerformance()

    + displayExamResults()

}

```

      Relationships

-   StudentManagementSystem   uses   Student   and   Examination  .

-   Examination   contains   Subject   objects.

**7. Implementation Details**

      7.1 Data Storage

-   Arrays   used instead of `ArrayList`/`HashMap`.

- Fixed-size storage (`Student[100]`, `Subject[5]`).

      7.2 Key Methods

-   `findStudentByRollNumber()`   → Linear search in array.

-   `calculateGradeAndPercentage()`   → Converts marks to grades.

-   `addSubject()`   → Stores marks and computes results.

      7.3 Input/Output Handling

-   `Scanner`   for CLI input.

- Formatted output for readability.

**8. Limitations**

* 1.   No Persistent Storage:   Data lost after program exit.
* 2.   Fixed Capacity:   Max 100 students, 5 subjects.
* 3.   No Data Validation:   No checks for duplicate roll numbers.
* 4.   Basic CLI:   No advanced UI/UX.

**9. Future Enhancements**

* 1.   File/Database Integration:   Save records permanently.
* 2.   GUI Development:   Switch from CLI to JavaFX/Swing.
* Advanced Features:
  + Attendance tracking.
  + Fee management.
  + Multi-user login (Admin/Student).
* 4.   Data Validation:   Prevent duplicate entries.

**10. Conclusion**

This   Student Management System   simplifies academic record-keeping by automating grade calculation and providing a structured way to manage student data. While it currently uses in-memory storage, it can be extended with file handling or databases for real-world use.

**Why This Project?**

- Demonstrates   core Java & OOP concepts  .

- Solves a   real-world problem   in education systems.

- Foundation for   scalable enhancements   (GUI, databases).

  End of Documentation