SparkINN Novice Solution Pvt. Ltd

Week 2 Java Weekend Task: Building a Simple Student Management System

---

Objective:
In this weekend task, you will apply the concepts learned in the Java course to build a simple **Student Management System**. This task will help you consolidate your understanding of Java object-oriented programming (OOP), classes, objects, methods, loops, and file handling. You will create a system where users can add, view, edit, and delete student records. The task covers key Java concepts such as encapsulation, inheritance, file handling, and exception handling.

Task Overview:

You will be required to:

1. Set Up the Java Development Environment:

- Install **Java Development Kit (JDK)** and **IntelliJ IDEA** or any other Java IDE.
- Ensure that your environment is correctly set up by creating a simple "Hello, World!" program.

2. Create the Project Structure:

- Create a new project called `StudentManagementSystem`.
- Inside the `src/` folder, create the following packages:
  - **models/**: For defining the Student class and related data models.
  - **services/**: For handling operations such as adding, editing, deleting, and viewing student records.
  - **main/**: For the entry point of your application (e.g., `Main.java`).
  - **utils/**: For utility classes, such as file handling and input validation.

---

3. Defining the Student Class:

- **Problem**: Create a `Student` class inside the `models` package with the following attributes:
  - **id** (integer)
  - **name** (string)
  - **age** (integer)
  - **grade** (string)

- **Hint**: Use Java **encapsulation** by making the attributes private and providing getter and setter methods for each field.
- **Code Example**:

```
public class Student {
    private int id;
    private String name;
    private int age;
    private String grade;

    // Constructor
    public Student(int id, String name, int age, String grade) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.grade = grade;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }

    public String getGrade() { return grade; }
    public void setGrade(String grade) { this.grade = grade; }
}
```

---

4. Implementing CRUD Operations in the `services` Package:

- **Problem**: Create a `StudentService` class to handle CRUD (Create, Read, Update, Delete) operations for student records.
    - **Add Student**: Implement a method that adds a new student record to an ArrayList or List.
    - **View Students**: Implement a method that displays all the student records.
    - **Edit Student**: Implement a method to update student details using the student ID.
    - **Delete Student**: Implement a method that deletes a student record by ID.
- **Code Example** (adding a student):

```
public class StudentService {
    private List<Student> students = new ArrayList<>();

    public void addStudent(Student student) {
        students.add(student);
    }

    public void viewStudents() {
        for (Student student : students) {
            System.out.println(student.getId() + " " +
student.getName() + " " + student.getAge() + " " +
student.getGrade());
        }
    }
```

```
}
```

5. Handling Input and Output (I/O):

- **Problem**: Use Java **file handling** to store and retrieve student records from a file.
  - Use the `FileWriter` and `BufferedReader` classes to write the list of students to a text file.
  - Read the student records from the file when the application starts.
  - Handle exceptions using **try-catch** blocks.
- **Code Example** (writing to a file):

```
public void saveToFile() throws IOException {
    FileWriter writer = new FileWriter("students.txt");
    for (Student student : students) {
        writer.write(student.getId() + "," + student.getName() + ","
+ student.getAge() + "," + student.getGrade() + "\n");
    }
    writer.close();
}
```

6. Exception Handling:

- **Problem**: Implement proper **exception handling** to manage any runtime errors (e.g., invalid input or file access issues).
  - Use custom exceptions where necessary, such as handling scenarios where a student ID is not found during editing or deletion.
- **Hint**: Catch IOExceptions and InputMismatchExceptions for user input.

---

7. Menu-Driven Console Application (Main.java):

- **Problem**: In the `Main` class, create a console-based menu where the user can:
  - Add a new student.
  - View all students.
  - Edit a student's details.
  - Delete a student record.
  - Save the student list to a file.
- **Hint**: Use a `Scanner` to get user input and switch-case for the menu options.
- **Code Example**:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    StudentService studentService = new StudentService();

    int choice;
    do {
        System.out.println("1. Add Student");
        System.out.println("2. View Students");
        System.out.println("3. Edit Student");
        System.out.println("4. Delete Student");
        System.out.println("5. Save & Exit");
        choice = scanner.nextInt();

        switch (choice) {
```

```
                case 1:
                    // Add student logic
                    break;
                case 2:
                    // View student logic
                    break;
                // other cases...
            }
        } while (choice != 5);
    }
```

---

8. Bonus Task (Optional): Adding Advanced Features

- Implement sorting functionality to display students by name or age.
- Add validation to ensure age is within a valid range (e.g., 10–100).
- Use Java **Streams API** to filter and sort student records.

Submission:

- Submit the entire project folder in a compressed format (.zip).
- Ensure all functionalities are working as expected.
- Include a README file with instructions on how to run the program.