Semester Project

(CLO-4: Develop a GUI based project for a real-world problem in a team environment)

Project Overview:

Your task is to develop a **School Management System** that manages **Students**, **Teachers**, **Courses**, and **Administrative Staff**. The system will provide functionality to register students and staff, assign courses to teachers and students, generate reports, and store/retrieve data from files. This project should demonstrate the use of object-oriented programming concepts in Java, including **inheritance, composition, polymorphism, static members, interfaces, object arrays, generics, and file handling**.

Project Requirements:

Core Functionalities:

1. **User Roles**:
    - **Student:**
        1. **Attributes**: studentID, name, address, dateOfBirth, list of enrolled courses.
        2. **Methods:**
            1. enrollInCourse(Course course): Adds a course to the student's list of enrolled courses.
               Output: Confirms the enrollment by printing the course name and student ID.
               Example: "Student S001 successfully enrolled in Data Structures."
            2. displayCourses(): Lists all the courses the student is enrolled in.
               Output: Displays a formatted list of courses with their titles and credits.
    - **Teacher:**
        1. **Attributes**: teacherID, name, specialization, list of courses taught.
        2. **Methods**:
            1. assignCourse(Course course): Adds a course to the teacher's list of courses.
               Output: Confirms the course assignment by printing the teacher name and course title.
               Example: "Teacher T001 assigned to teach Advanced Algorithms."
            2. displayCourses(): Lists all the courses taught by the teacher.
               Output: Displays the course titles and IDs.
    - **Administrative Staff:**
        1. **Attributes**: staffID, name, role, and department.
        2. **Methods**:
            1. generateReport(List<Person> people): Generates a report summarizing the data for students, teachers, or courses, depending on the input list.
               Output: Returns a string report.
               Example:

```
Total Students: 3
Total Teachers: 2
Total Courses: 5
```

    - **Course Management**:
        1. **Attributes**: courseID, title, credits, assignedTeacher, list of enrolled students.
        2. **Methods**:
            1. addStudent(Student student): Adds a student to the course.
               Output: Displays confirmation.
               Example: "Student S002 added to Introduction to Java."

2. removeStudent(Student student): Removes a student from the course.
   Output: Displays confirmation or error if the student is not enrolled.
   Example: "Student S002 removed from Data Structures."
3. calculateAverageGrade(): Calculates the average grade of all students in the course.
   Output: Displays the average grade.
   Example: "Average grade for Introduction to Java: 85.6"
   .

2. **Inheritance:**
   - Create a base class Person with common attributes like name, email, and dateOfBirth. Use inheritance to create Student, Teacher, and AdministrativeStaff classes.

3. **Polymorphism and Dynamic Binding**:
   - Use overridden methods such as generateReport() in the AdministrativeStaff class and displayDetails() in the Student and Teacher classes to demonstrate polymorphism.
   - Override toString() in each class (e.g., Student, Teacher, AdministrativeStaff) to display class-specific details.
   - **Output**: A formatted string representation.
   - **Example** for Student:

     Student: S001, Name: Ali, Enrolled in: [Data Structures, Algorithms]

4. **Static Data Members and Methods**:
   - Use static counters to track the total number of students, teachers, and courses.
   - Static methods in ***University***:
     1. displaySystemStats(): Displays total counts for students, teachers, and courses.
     2. **Output:**

```
Total Students: 150
Total Teachers: 30
Total Courses: 20
```

5. **Interfaces**:
   - Create an interface Reportable with methods like generateReport() and exportToFile(). Implement this interface in AdministrativeStaff and Teacher.
   - **Methods**:
   - generateReport(): Outputs a summary of the data specific to the implementer.
   - **Example** for Administrative Staff:

```
Total Students: 150
Total Teachers: 30
Total Courses: 20
```

6. **Composition**:
    - A Course object should have a reference to a Teacher object and an array of Student objects.
7. **Object Arrays and ArrayList**:
    - Use object arrays to store Student and Course objects.
    - Use ArrayList for dynamic management of students enrolled in courses.

8. **File Handling**:
    - Save and retrieve data for students, teachers, and courses using files.
    - Implement functionality to load and save the current state of the system to files.
    - **Methods** in University:
        1. loadData(String filename): Loads students, teachers, and courses from a file.
           Output: Confirms success or error.
           Example: "Data loaded successfully from university_data.txt."
        2. saveData(String filename): Saves current system data to a file.
           Output: Confirms success.
           Example: "Data saved successfully to university_data.txt."

9. **Generics**:
    - Use a generic class Repository<T> to manage lists of students, teachers, and courses.
    - **Methods**:
        1. add(T item): Adds an object of type T to the repository.
           Output: Displays confirmation.
           Example: "Student S003 added to the repository."
        2. remove(T item): Removes an object of type T from the repository.
           Output: Displays confirmation or error.
           Example: *"Course CS101 removed from the repository."*
        3. getAll(): Returns a list of all objects of type T.

10. **Wrapper Classes**:
    - Use wrapper classes for processing numeric data, e.g., calculating the average grades of students in a course.
    - **Method** in Course:
        1. calculateMedianGrade(): Calculates and returns the median grade for students in a course.

11. **Static & Dynamic Typing**:
    - Demonstrate static typing with explicit type declarations.
    - Incorporate dynamic typing by using Object references and down casting where applicable.

Additional Requirements:

- **Menu-driven GUI Application**:
    - A user-friendly GUI interface for performing CRUD operations on students, teachers, and courses.

- **Reports**:
    - Generate detailed reports on:

        - Students enrolled in a course.
        - Teacher Workload Report: List all courses taught by each teacher.
          Example:
Teacher: Ali, Courses: [Data Structures, Algorithms]
        - Overall system statistics.

- **Error Handling**:
    - Include exception handling for invalid input and file operations.

- **Grading System**:
    - Each course should store grades for enrolled students.
    - Implement functionality to calculate average grades for a course.

- **Search and Filtering:**
    - Add methods in University:
        - searchStudentByName(String name)
          Output: Returns a list of students with matching names. Example: "Found students: Ali Asad, Ali Rehman"
        - filterCoursesByCredits(int minCredits)
          Output: Returns a list of courses with credits >= minCredits. Example: "Courses with at least 3 credits: Data Structures, Algorithms."

Evaluation Criteria:

1. **Implementation of OOP Principles (25%)**:
    - Correct use of inheritance, polymorphism, interfaces, and composition.

2. **Functional Completeness (25%)**:
    - All required functionalities are implemented and work as expected.

3. **Documentation and Reporting (10%)**:
    - Well-documented code and clear project report.

4. **Viva (40%)**:
    - Individual oral examination will be conducted.

Deliverables:

1. **Source Code**: Submit the complete project code.
2. **Documentation**:
    - Class diagram.
    - Explanation of functionality.
    - Sample input/output.
3. **Viva:** Oral examination