A blue and white logo

Description automatically generated**COMSATS University Islamabad**

Department of Computer Science

**CSC241 – Object Oriented Programming**

**School Management System**

**Project Documentation**

**Submitted To:** Ms. Sajida Kalsoom

**Submitted By:** Muhammad Abuzer Zia (FA23-BCS-055)

Muhmmad Siraj (FA23-BCS-077)

**Class:** BSCS-3B

**Date of Submission:** 22-12-2024

Contents

[School Management System Project Documentation 2](#_Toc185801072)

[1. Overview 2](#_Toc185801073)

[Key Features 2](#_Toc185801074)

[GUI Implementation 3](#_Toc185801075)

[2. Class Descriptions 3](#_Toc185801076)

[Teacher 4](#_Toc185801077)

[AdministrativeStaff 5](#_Toc185801078)

[Course 6](#_Toc185801079)

[University 9](#_Toc185801080)

[3. Project Structure 10](#_Toc185801081)

[GUI based menu driven program: 12](#_Toc185801082)

[4. UML Diagram 18](#_Toc185801083)

[Explanation Of UML: 19](#_Toc185801084)

# School Management System Project Documentation

# 1. Overview

The School Management System is a robust **GUI-based** application developed in Java to address the complex needs of managing a school's operations. It allows for efficient handling of students, teachers, administrative staff, and courses. The project leverages the power of Object-Oriented Programming principles, including inheritance, polymorphism, abstraction, and encapsulation, ensuring modularity and scalability.  
  
The application includes advanced features such as file handling for data persistence, exception handling for error management, and a dynamic reporting system. The user-friendly GUI, implemented using Java Swing, facilitates **CRUD** operations, course assignments, and data analysis in an interactive manner. With its modular design, this system can be easily extended to accommodate additional functionalities.

## Key Features

The following features highlight the capabilities of the School Management System:  
  
1. **Inheritance:** Common attributes and methods are shared using a base Person class, which is extended by Student, Teacher, and AdministrativeStaff.  
2. **Polymorphism:** Demonstrated through overridden methods like generateReport() and displayDetails(), enabling dynamic behavior.  
3. **Abstraction:** Achieved via the Reportable interface, which defines a contract for generating reports.  
4. **Encapsulation:** Private attributes ensure data security, with controlled access through getter and setter methods.  
5. **Generics:** The Repository<T> class facilitates type-safe management of students, teachers, and courses.  
6. **Object Arrays and ArrayList**: Provide flexible and efficient storage solutions for managing data.  
7. **File Handling:** Ensures data persistence by saving and retrieving system data.  
8. **Exception Handling:** Handles invalid inputs and file errors gracefully, ensuring system stability.

### GUI Implementation

The GUI is developed using Java Swing and provides users with interactive menus for managing data. Operations include creating, reading, updating, and deleting records, assigning courses, and generating reports.

## 2. Class Descriptions

#### Student Class

#### Attributes:

#### studentID:

#### name:

#### address:

#### dateOfBirth:

#### enrolledCourses:

* TotalStudents(static):

#### Methods

* enrollInCourse(Course course)

This method allows a student to enroll in a course. It adds the course to the student's enrolled courses object arrays and confirms enrollment by printing a message. Demonstrates encapsulation as the method updates private data (enrolledCourses) through a controlled interface.

* displayCourses()

Displays all courses the student is enrolled in. Demonstrates abstraction by hiding implementation details and presenting a user-friendly output.

A screen shot of a computer screen

Description automatically generated

### Teacher

#### Attributes:

#### teacherID:

#### name:

#### specialization:

#### coursesTaught:

* TotalTeacher(static):

#### Methods

* assignCourse(Course course)

Assigns a course to the teacher. The course is added to the teacher's taught courses object arrays and a confirmation message is displayed. Demonstrates encapsulation and ensures data consistency.

* displayCourses()

Lists all courses the teacher is teaching. Demonstrates polymorphism if overridden in subclasses.

A screen shot of a computer program

Description automatically generated

### AdministrativeStaff

#### Attributes

#### staffID:

#### name:

#### role:

#### department:

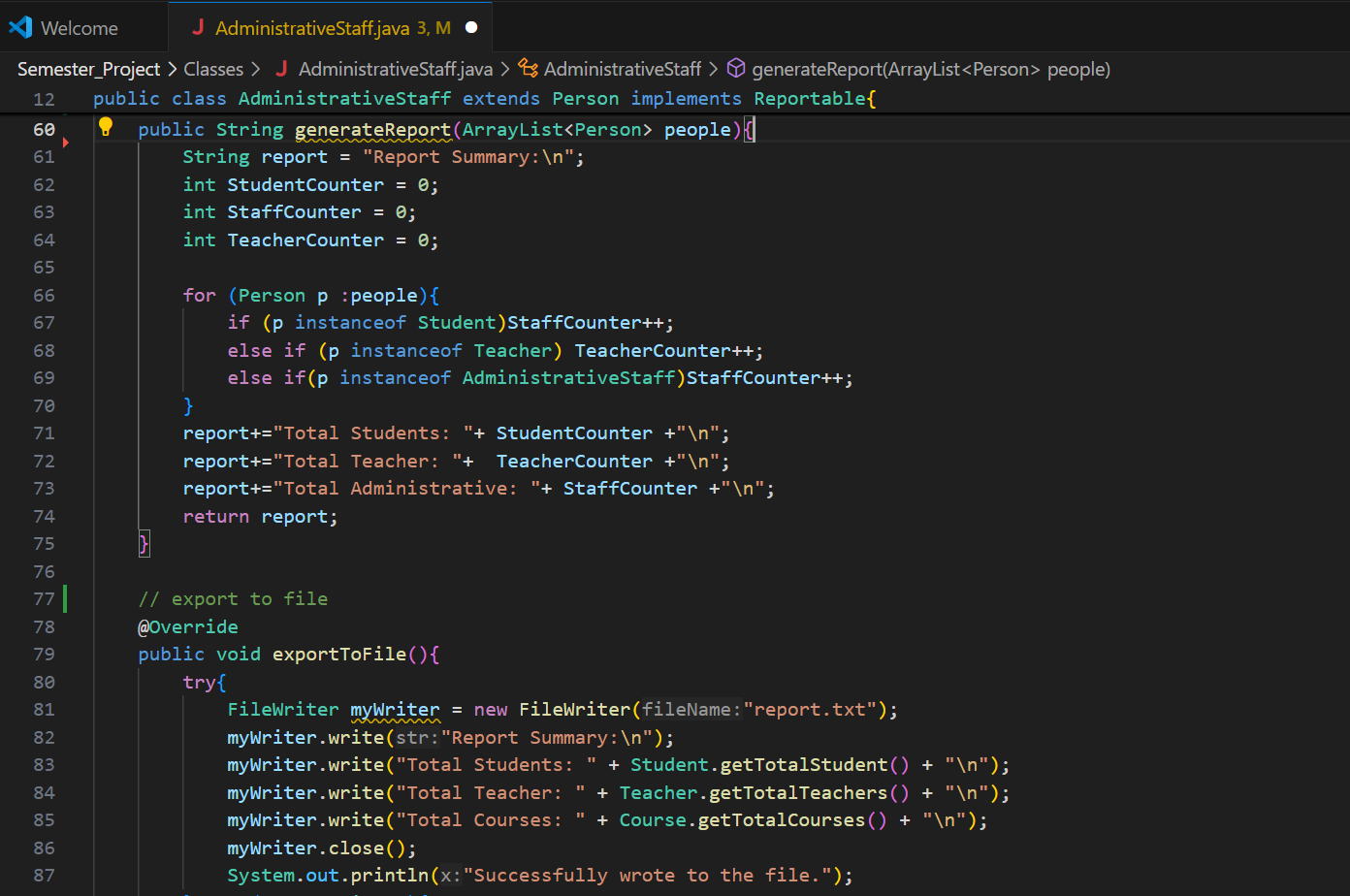
#### Methods

* generateReport(List<Person> people)

Generates a summary report for a given list of people, such as students or teachers. The method demonstrates polymorphism by overriding the generateReport() method in the Reportable interface and ensures modularity in report creation.

* exportToFile()

Exports the static counters data to the file. Such as the total number of students, teachers and courses.



### Course

#### Attributes

* courseID:
* title:
* credits:
* assignedTeacher:
* enrolledStudents:
* grades:
* TotalCourses:

#### Methods

* addStudent(Student student)

Adds a student to the course. Confirms the addition with a message. Demonstrates composition by managing relationships between courses and students.

* removeStudent(Student student)

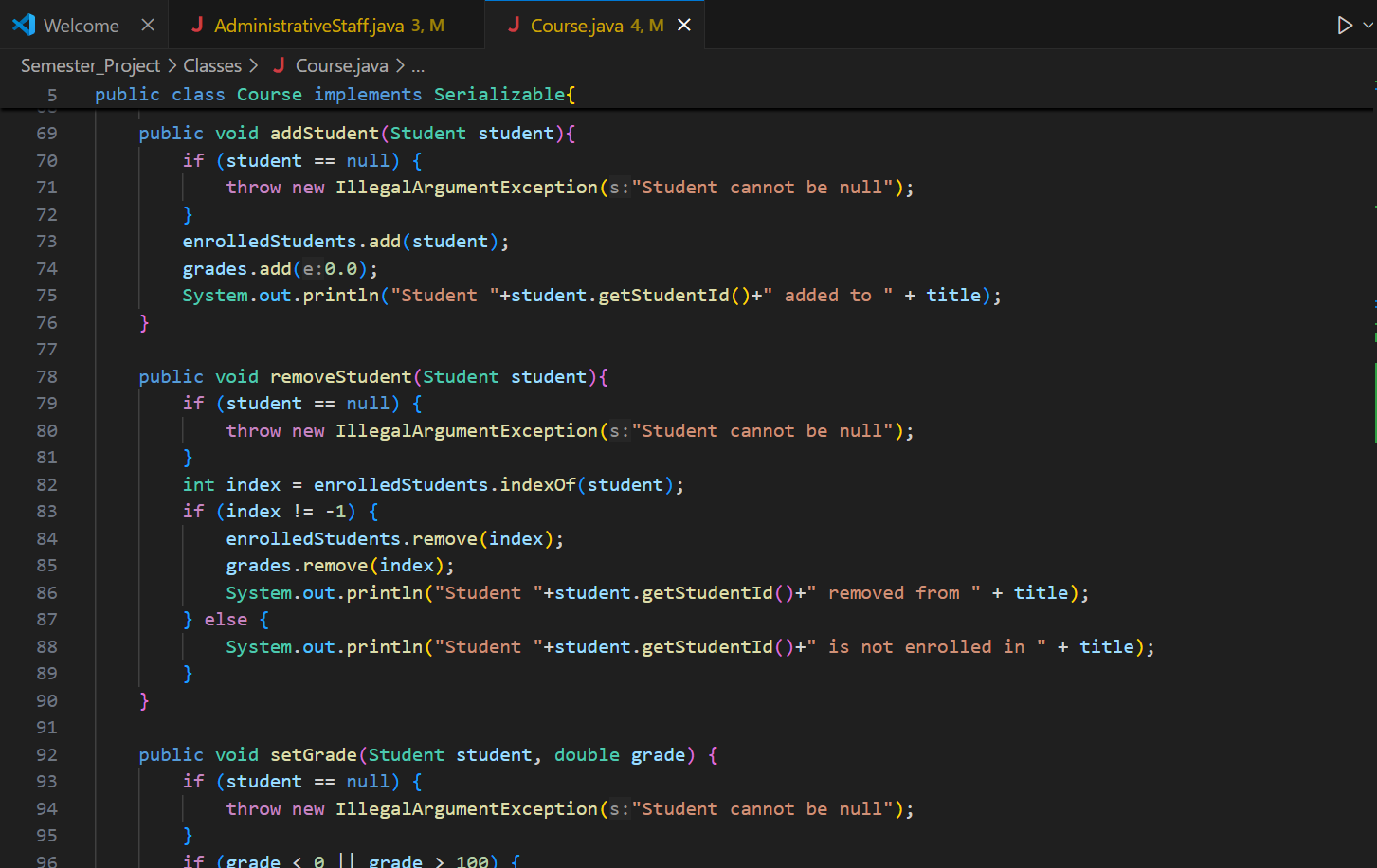
Removes a student from the course. Ensures data integrity by checking if the student is enrolled before removal. Demonstrates exception handling to manage invalid operations.

* calculateAverageGrade()

Calculates the average grade of all enrolled students. Uses wrapper classes to process numeric data.

* calculateMedianGrade()

Calculate the median grade for all enrolled students.



**Repository**

#### Attributes

* Items:

#### Methods

* **add(T item):**

Adds a new object of type T to the repository, ensuring type safety through generics and maintaining data consistency.

* **remove(T item):**

Removes a specified object from the repository, using exception handling to manage cases where the object is not found.

* **getAll():**

Retrieves all objects stored in the repository, providing abstraction by hiding the underlying data structure.

A screen shot of a computer program

Description automatically generated

### University

#### Attributes

* studentsRepository:
* teachersRepository:
* coursesRepository:
* AdministrativeStaffRepository:

#### Methods

* loadData(String filename)

Loads students, teachers, and courses from a file to restore the system's state. Demonstrates file handling and ensures smooth recovery of data.

* saveData(String filename)

Saves the current state of the system, including all students, teachers, and courses, to a file. Uses exception handling to manage file write errors.

* displaySystemStats()

Displays the total number of students, teachers, and courses in the system. Demonstrates the use of static data members to track counts.

* searchStudentByName(String name)

Searches for students with matching names in the repository and returns a list of results, demonstrating abstraction for efficient data retrieval.

* filterCoursesByCredits(int minCredits)

Filters and returns courses with credits greater than or equal to the specified minimum, showcasing abstraction and data management.

A screen shot of a computer program

Description automatically generated

**Exception and File Handling**

The project incorporates exception handling to manage invalid inputs, such as invalid IDs or file read/write errors. For instance, methods like 'loadData' and 'saveData' in the 'University' class handle file-related exceptions to ensure data integrity.

File handling is used to persist data for students, teachers, and courses. The 'saveData' method saves the current state to a file, while the 'loadData' method retrieves data from a file. This ensures that the system can resume its state after a restart.

## 3. Project Structure

* The School Management System is organized into three main components to ensure modularity and maintainability:  
    
  The graphical user interface (GUI), developed using Java Swing, provides users with a platform for managing school data. It allows operations such as adding, updating, and deleting records for students, teachers, courses, and administrative staff. Additionally, the GUI facilitates generating reports and assigning courses, ensuring an intuitive experience.  
    
  The core classes define the primary entities of the system, including Student, Teacher, AdministrativeStaff, and Course. Each class encapsulates relevant attributes and methods to model the behavior of these entities. For example, the Student class handles course enrollment, while the Teacher class manages course assignments.  
    
  The utility classes, such as University and Repository, are responsible for data management and persistence. These classes enable loading and saving data from files, managing repositories of students, teachers, and courses, and generating system usage statistics.  
    
  This structure seamlessly integrates the user-facing GUI with the backend, enabling efficient and organized data management throughout the application.
* Interaction with Backend: The GUI interacts with the backend University class to perform CRUD operations and generate reports. Each button triggers an action that updates the backend data and reflects the changes in the UI.
* Error Handling: The GUI includes input validation and exception handling. For instance, adding a student check if fields are filled and ensures the ID is unique.
* File Operations: The GUI provides options to save and load system data, ensuring persistence across sessions. These operations are integrated with the backends’ file handling methods

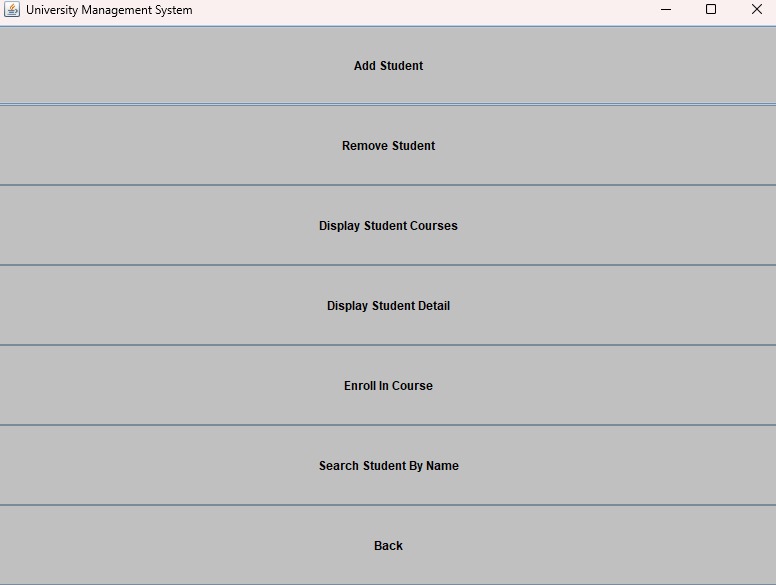
### GUI based menu driven program:

#### Main Panel:

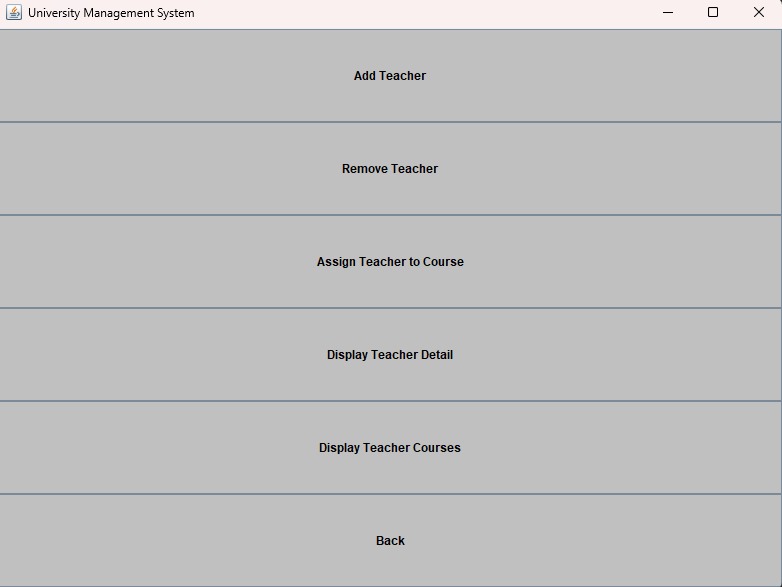
A screenshot of a computer

Description automatically generated

#### Student Sub-Panel:



#### Teacher Sub-Panel:



#### Course Sub-Panel:

A screenshot of a computer

Description automatically generated

#### Administrative Staff Sub-Panel:

A screenshot of a computer

Description automatically generated

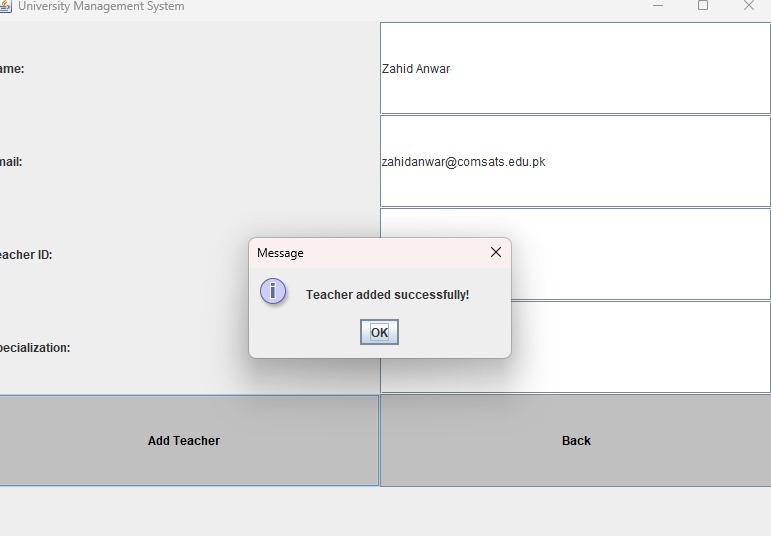
#### Sample Inputs:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

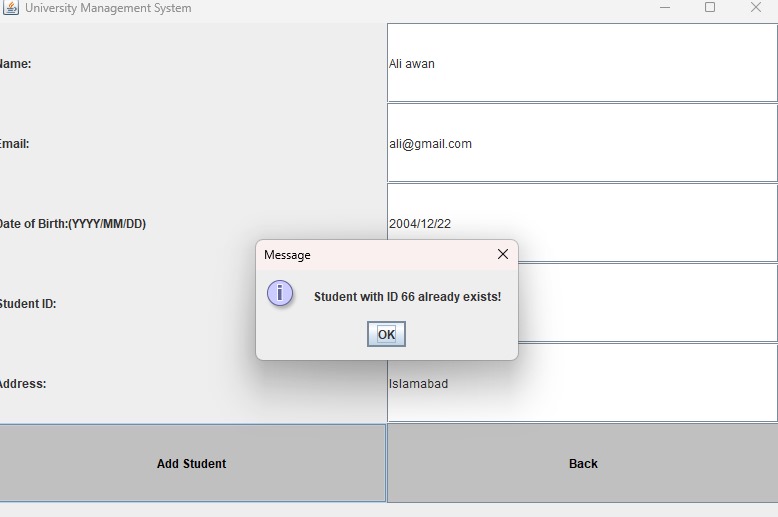


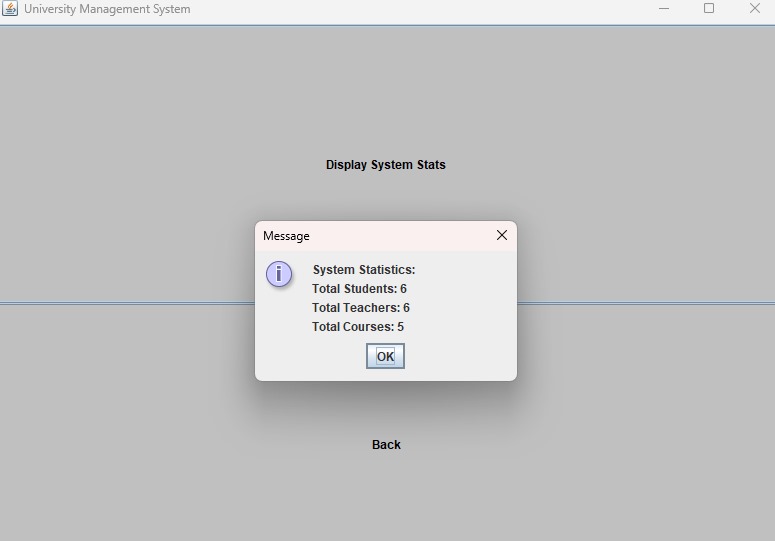
#### Outputs:

A screenshot of a computer

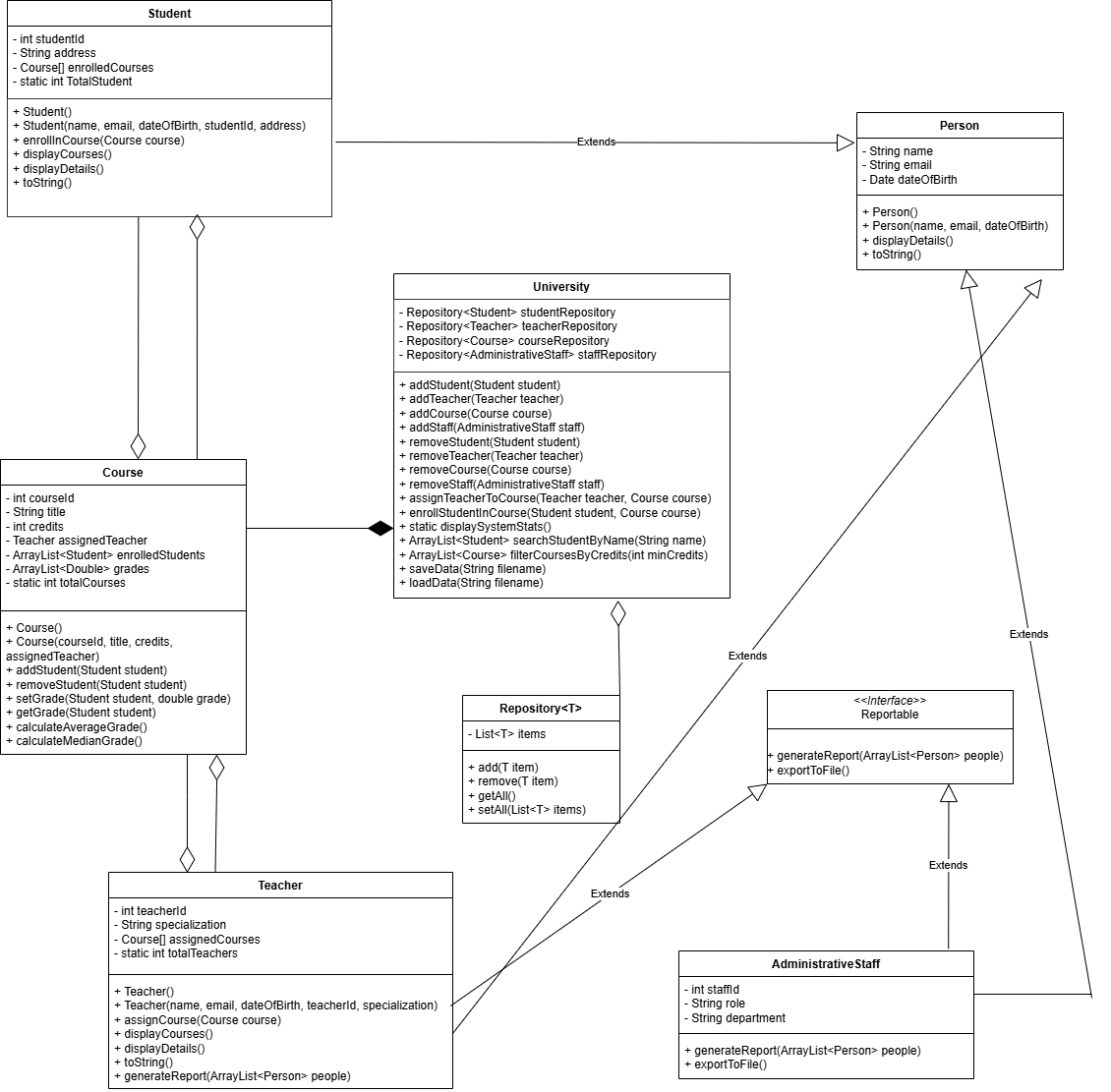
Description automatically generated

#### Validations:

Statistics:



## 4. UML Diagram



Getters and Setters are omitted for clarity of the diagram

### Explanation Of UML:

**1. Inheritance Relationships:**

Person is the parent class for:

Student extends Person

Teacher extends Person

AdministrativeStaff extends Person.

**2. Interface Implementation:**

AdministrativeStaff implements Reportable

**3. Composition:**

University to Course:

University fully owns and manages Course objects

**4. Aggregation:**

University to Repository: University has repositories, but they could exist

independently

Course to Teacher: Course has an assigned teacher, but teacher exists independently

Course to Student: Course has enrolled students but they exist independently

Teacher to Course: Teacher has assigned courses but courses exist independently

Student to Course: Student has enrolled courses but courses exist independently

**5. Generic Relationships:**

Repository<T> is used by university for different types:

Repository<Student>

Repository<Teacher>

Repository<Course>

Repository<AdministrativeStaff>