



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
1. Overview	2
Key Features	2
GUI Implementation.....	3
2. Class Descriptions	3
Teacher	4
AdministrativeStaff	5
Course.....	6
University.....	9
3. Project Structure.....	10
GUI based menu driven program:	12
4. UML Diagram.....	18
Explanation Of UML:.....	19

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

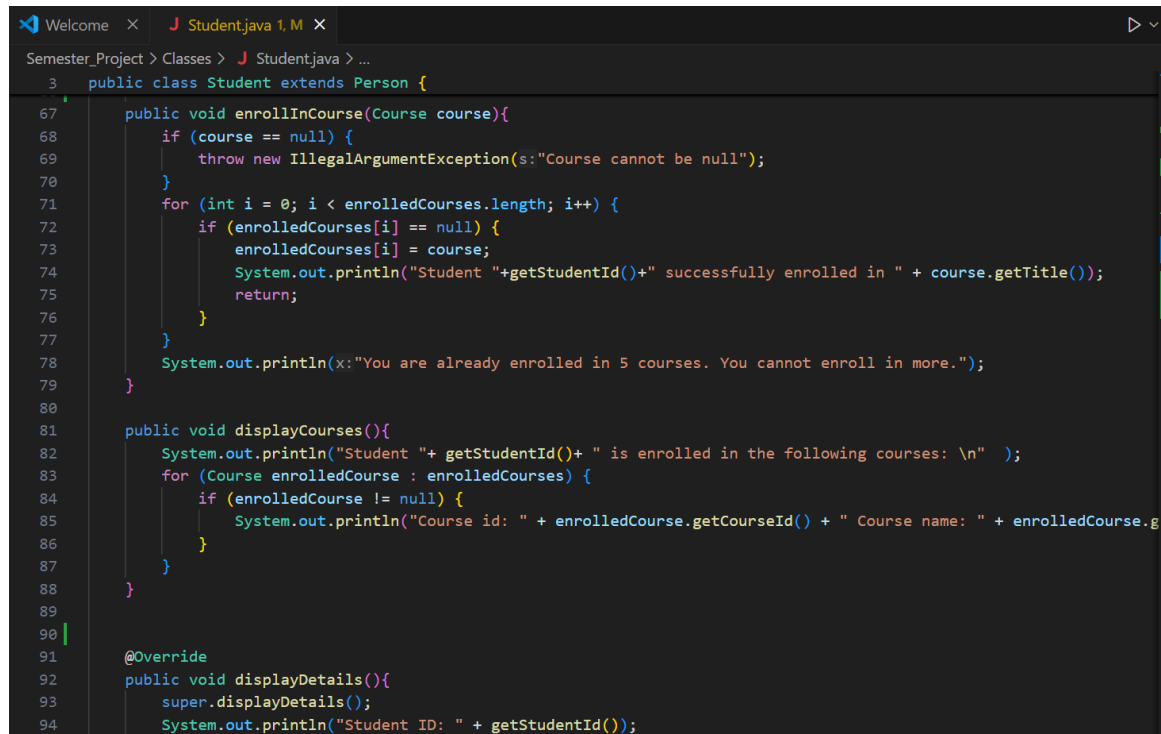
- enrollInCourse(Course course)

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.



```

3 public class Student extends Person {
67     public void enrollInCourse(Course course){
68         if (course == null) {
69             throw new IllegalArgumentException(s:"Course cannot be null");
70         }
71         for (int i = 0; i < enrolledCourses.length; i++) {
72             if (enrolledCourses[i] == null) {
73                 enrolledCourses[i] = course;
74                 System.out.println("Student "+getStudentId()+" successfully enrolled in " + course.getTitle());
75                 return;
76             }
77         }
78         System.out.println(x:"You are already enrolled in 5 courses. You cannot enroll in more.");
79     }
80
81     public void displayCourses(){
82         System.out.println("Student "+ getStudentId()+ " is enrolled in the following courses: \n" );
83         for (Course enrolledCourse : enrolledCourses) {
84             if (enrolledCourse != null) {
85                 System.out.println("Course id: " + enrolledCourse.getCourseId() + " Course name: " + enrolledCourse.g
86             }
87         }
88     }
89
90
91     @Override
92     public void displayDetails(){
93         super.displayDetails();
94         System.out.println("Student ID: " + getStudentId());

```

Teacher

Attributes:

- teacherID:
- name:
- specialization:
- coursesTaught:
- TotalTeacher(static):

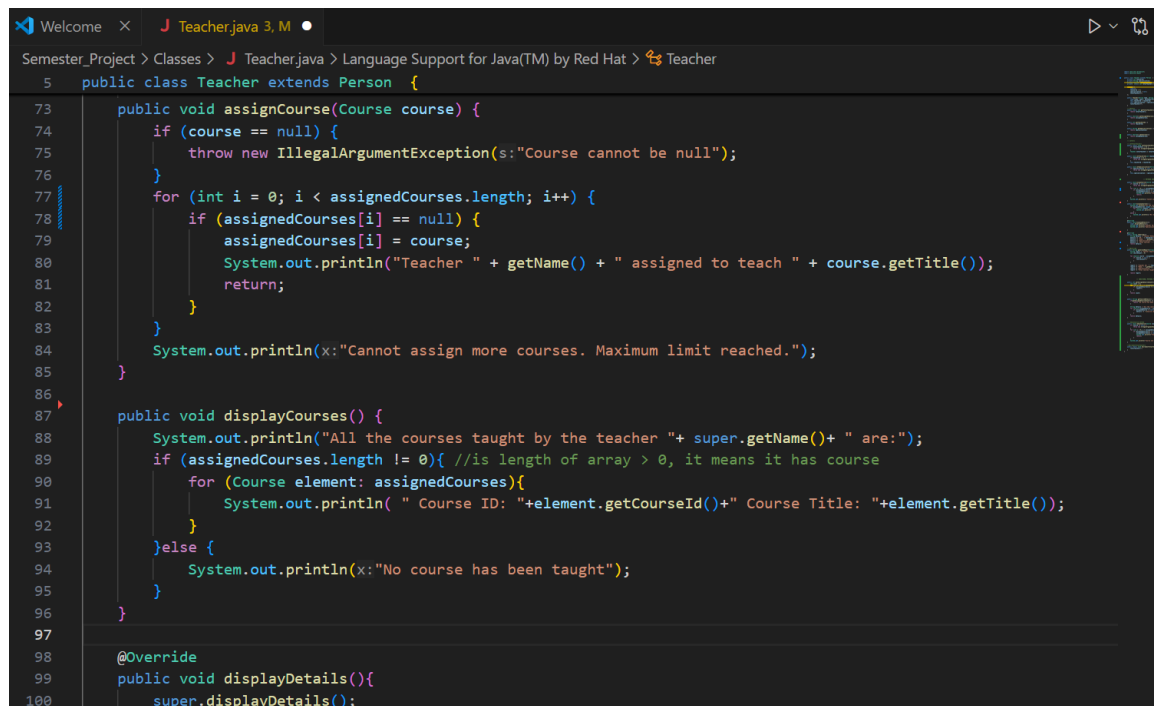
Methods

- 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.



```
5 public class Teacher extends Person {
73     public void assignCourse(Course course) {
74         if (course == null) {
75             throw new IllegalArgumentException(s:"Course cannot be null");
76         }
77         for (int i = 0; i < assignedCourses.length; i++) {
78             if (assignedCourses[i] == null) {
79                 assignedCourses[i] = course;
80                 System.out.println("Teacher " + getName() + " assigned to teach " + course.getTitle());
81                 return;
82             }
83         }
84         System.out.println(x:"Cannot assign more courses. Maximum limit reached.");
85     }
86
87     public void displayCourses() {
88         System.out.println("All the courses taught by the teacher "+ super.getName()+ " are:");
89         if (assignedCourses.length != 0){ //is length of array > 0, it means it has course
90             for (Course element: assignedCourses){
91                 System.out.println( " Course ID: "+element.getCourseId()+" Course Title: "+element.getTitle());
92             }
93         }else {
94             System.out.println(x:"No course has been taught");
95         }
96     }
97
98     @Override
99     public void displayDetails(){
100         super.displayDetails();
```

AdministrativeStaff

Attributes

- staffID:
- name:
- role:
- department:

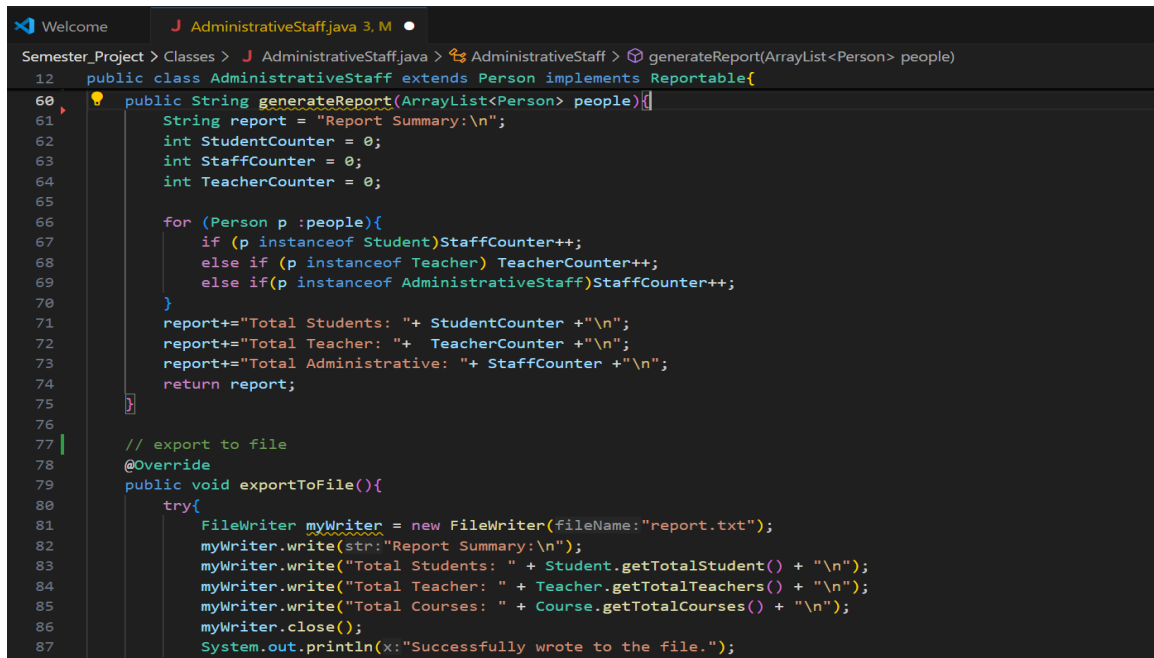
Methods

- 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.



```
12 public class AdministrativeStaff extends Person implements Reportable{
60     public String generateReport(ArrayList<Person> people){
61         String report = "Report Summary:\n";
62         int StudentCounter = 0;
63         int StaffCounter = 0;
64         int TeacherCounter = 0;
65
66         for (Person p :people){
67             if (p instanceof Student)StaffCounter++;
68             else if (p instanceof Teacher) TeacherCounter++;
69             else if(p instanceof AdministrativeStaff)StaffCounter++;
70         }
71         report+="Total Students: " + StudentCounter +"\n";
72         report+="Total Teacher: " + TeacherCounter +"\n";
73         report+="Total Administrative: " + StaffCounter +"\n";
74         return report;
75     }
76
77     // export to file
78     @Override
79     public void exportToFile(){
80         try{
81             FileWriter myWriter = new FileWriter(fileName:"report.txt");
82             myWriter.write(str:"Report Summary:\n");
83             myWriter.write("Total Students: " + Student.getTotalStudent() + "\n");
84             myWriter.write("Total Teacher: " + Teacher.getTotalTeachers() + "\n");
85             myWriter.write("Total Courses: " + Course.getTotalCourses() + "\n");
86             myWriter.close();
87             System.out.println(x:"Successfully wrote to the file.");
```

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)

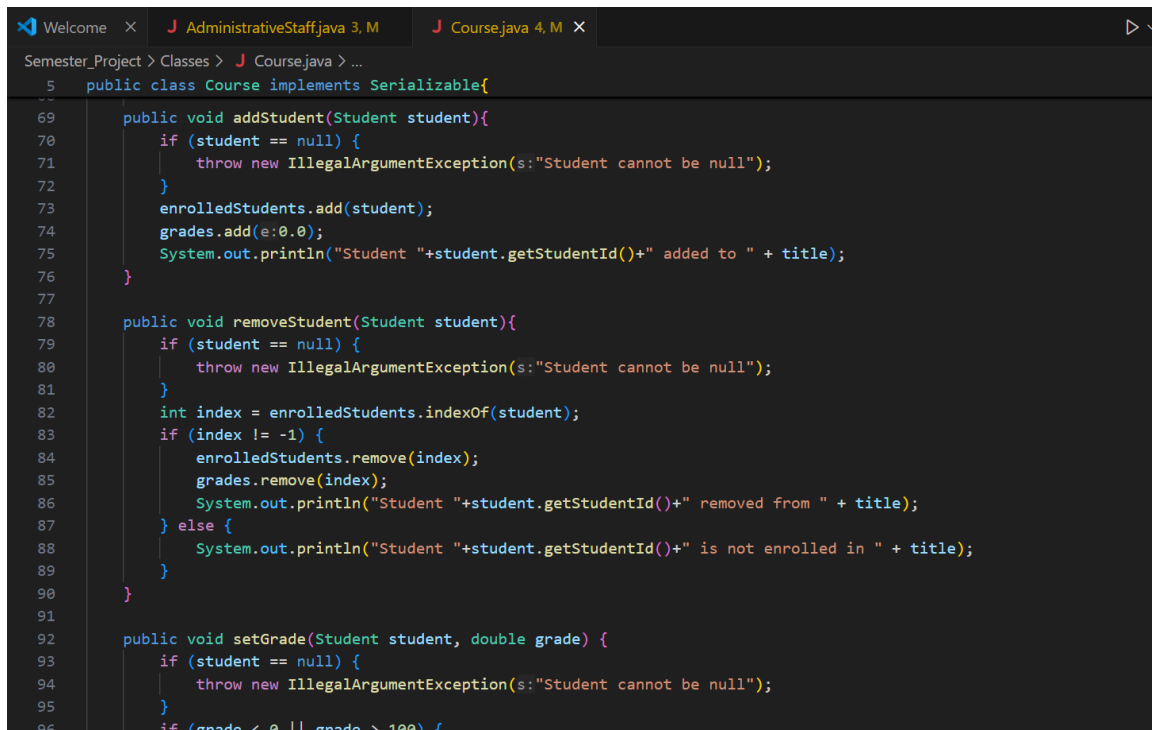
Removes a student from the course. Ensures data integrity by checking if the student is enrolled before removal. Demonstrates exception handling to manage invalid operations.

- 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.



```
5 public class Course implements Serializable{
69     public void addStudent(Student student){
70         if (student == null) {
71             throw new IllegalArgumentException(s:"Student cannot be null");
72         }
73         enrolledStudents.add(student);
74         grades.add(e:0.0);
75         System.out.println("Student "+student.getId()+" added to " + title);
76     }
77
78     public void removeStudent(Student student){
79         if (student == null) {
80             throw new IllegalArgumentException(s:"Student cannot be null");
81         }
82         int index = enrolledStudents.indexOf(student);
83         if (index != -1) {
84             enrolledStudents.remove(index);
85             grades.remove(index);
86             System.out.println("Student "+student.getId()+" removed from " + title);
87         } else {
88             System.out.println("Student "+student.getId()+" is not enrolled in " + title);
89         }
90     }
91
92     public void setGrade(Student student, double grade) {
93         if (student == null) {
94             throw new IllegalArgumentException(s:"Student cannot be null");
95         }
96         if (grade < 0 || grade > 100) {
```


Repository

Attributes

- Items:

Methods

- **add(T item):**

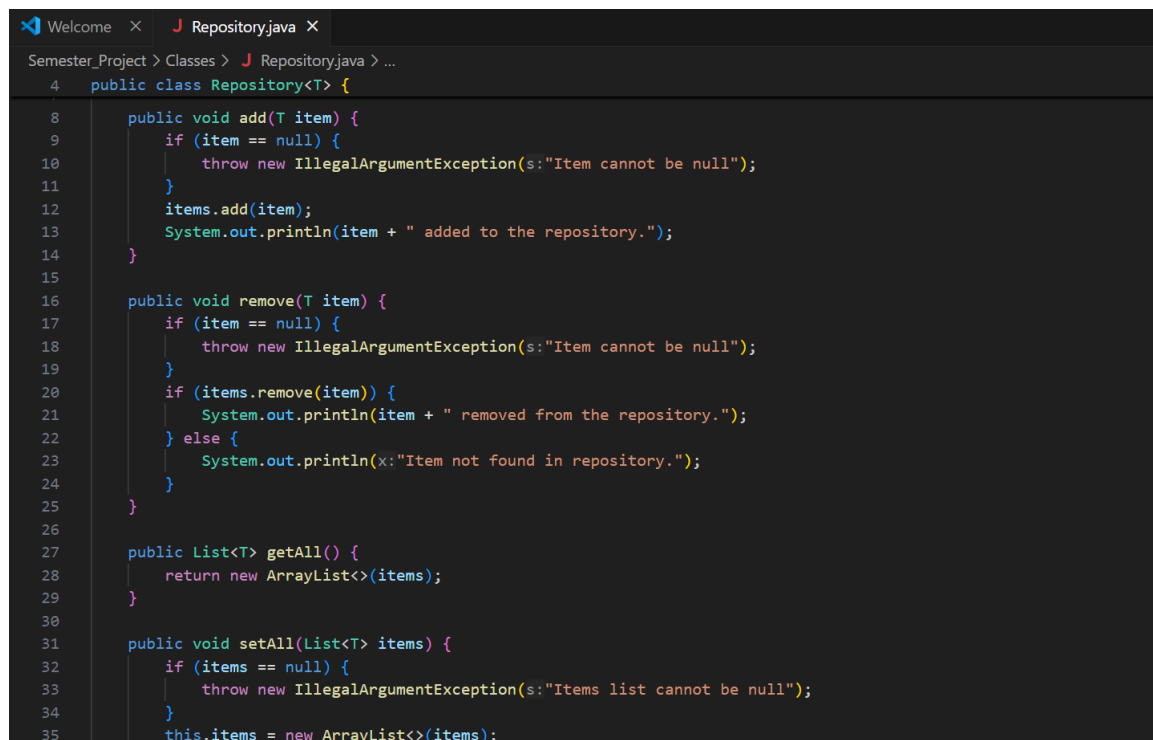
Adds a new object of type T to the repository, ensuring type safety through generics and maintaining data consistency.

- **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.



```
4 public class Repository<T> {
    8 public void add(T item) {
    9     if (item == null) {
   10         throw new IllegalArgumentException(s:"Item cannot be null");
   11     }
   12     items.add(item);
   13     System.out.println(item + " added to the repository.");
   14 }
   15
   16 public void remove(T item) {
   17     if (item == null) {
   18         throw new IllegalArgumentException(s:"Item cannot be null");
   19     }
   20     if (items.remove(item)) {
   21         System.out.println(item + " removed from the repository.");
   22     } else {
   23         System.out.println(x:"Item not found in repository.");
   24     }
   25 }
   26
   27 public List<T> getAll() {
   28     return new ArrayList<>(items);
   29 }
   30
   31 public void setAll(List<T> items) {
   32     if (items == null) {
   33         throw new IllegalArgumentException(s:"Items list cannot be null");
   34     }
   35     this.items = new ArrayList<>(items);
}
```

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.

```

5 public class University {
28
29     public ArrayList<Student> searchStudentByName(String name){
30         //Returns a list of students with matching names
31         ArrayList<Student> students = new ArrayList<>();
32         for(Student student : studentRepository.getAll()){
33             if(student.getName().contains(name)){
34                 students.add(student);
35             }
36         }
37         return students;
38     }
39
40     public ArrayList<Course> filterCoursesByCredits(int minCredits){
41
42         ArrayList<Course> courses = new ArrayList<>();
43
44         for(Course course : courseRepository.getAll()){
45             if(course.getCredits() >= minCredits){
46                 courses.add(course);
47             }
48         }
49         return courses;
50     }
51
52     // Save data to file
53     public void saveData(String filename) {
54         if (filename == null || filename.isEmpty()) {
55             System.err.println(x:"Invalid filename.");
56         }
57     }

```

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.

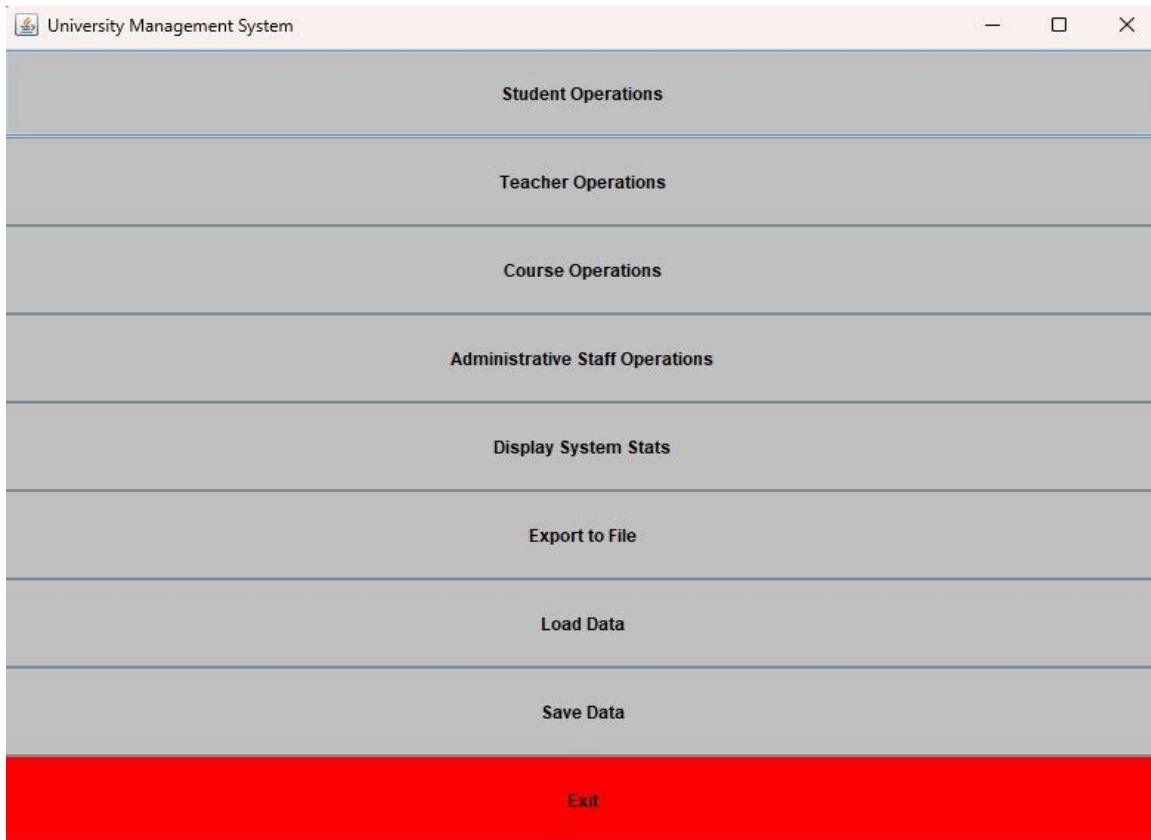
The utility classes, such as University and Repository, are responsible for data management and persistence. These classes enable loading and saving data from files, managing repositories of students, teachers, and courses, and generating system usage statistics.

This structure seamlessly integrates the user-facing GUI with the backend, enabling efficient and organized data management throughout the application.

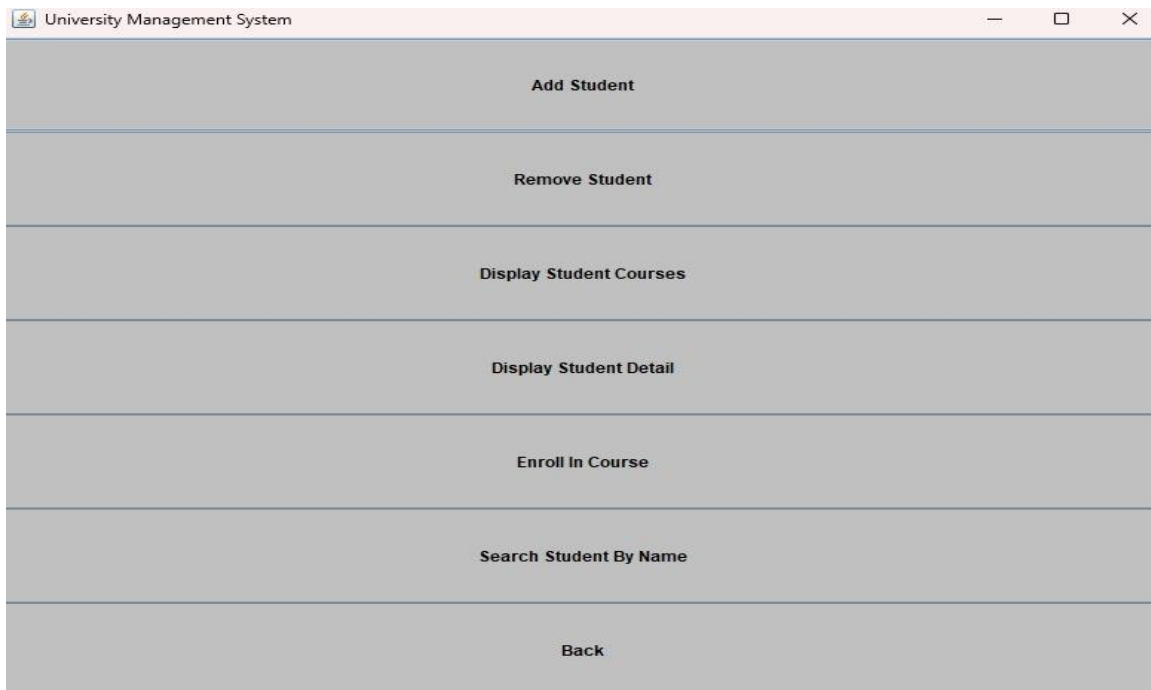
- **Interaction with Backend:** The GUI interacts with the backend University class to perform CRUD operations and generate reports. Each button triggers an action that updates the backend data and reflects the changes in the UI.
- **Error Handling:** The GUI includes input validation and exception handling. For instance, adding a student checks 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:



Student Sub-Panel:




Teacher Sub-Panel:

University Management System	—	□	×
Add Teacher			
Remove Teacher			
Assign Teacher to Course			
Display Teacher Detail			
Display Teacher Courses			
Back			

Course Sub-Panel:

University Management System	—	□	×
Add Course			
Remove Course			
Add Grade			
Calculate Average Grade			
Calculate Median Grade			
Filter Courses By Credits			
Remove Student From Course			
Back			

Administrative Staff Sub-Panel:

 University Management System— □ ×

Add Staff

Remove Staff

Display Staff Detail


Back

Sample Inputs:

University Management System	
Name:	Ali awan
Email:	ali@gmail.com
Date of Birth:(YYYY/MM/DD)	2004/12/22
Student ID:	66
Address:	Islamabad
<div>Add StudentBack</div>	

University Management System	
Name:	Ali awan
Email:	ali@gmail.com
Date of Birth:(YYYY/MM/DD)	2004/12/22
Student ID:	
Address:	Islamabad
<div>Add StudentBack</div>	

Message

 Student added successfully!

OK

University Management System

Name: Zahid Anwar

Email: zahidanwar@comsats.edu.pk

Teacher ID:

Specialization:

Add Teacher **Back**

Message X

i Teacher added successfully!

OK

Outputs:

Teacher ID: 11

Display Teacher Detail **Back**

Message X

i Teacher Details:
ID: 11
Name: tanveer
Specialization: cs
Total Courses: 5

OK

Validations:

University Management System

Name: Ali awan

Email: ali@gmail.com

Date of Birth:(YYYY/MM/DD) 2004/12/22

Student ID:

Address: Islamabad

Add Student Back

Message

Student with ID 66 already exists!

OK

Statistics:

University Management System

Display System Stats

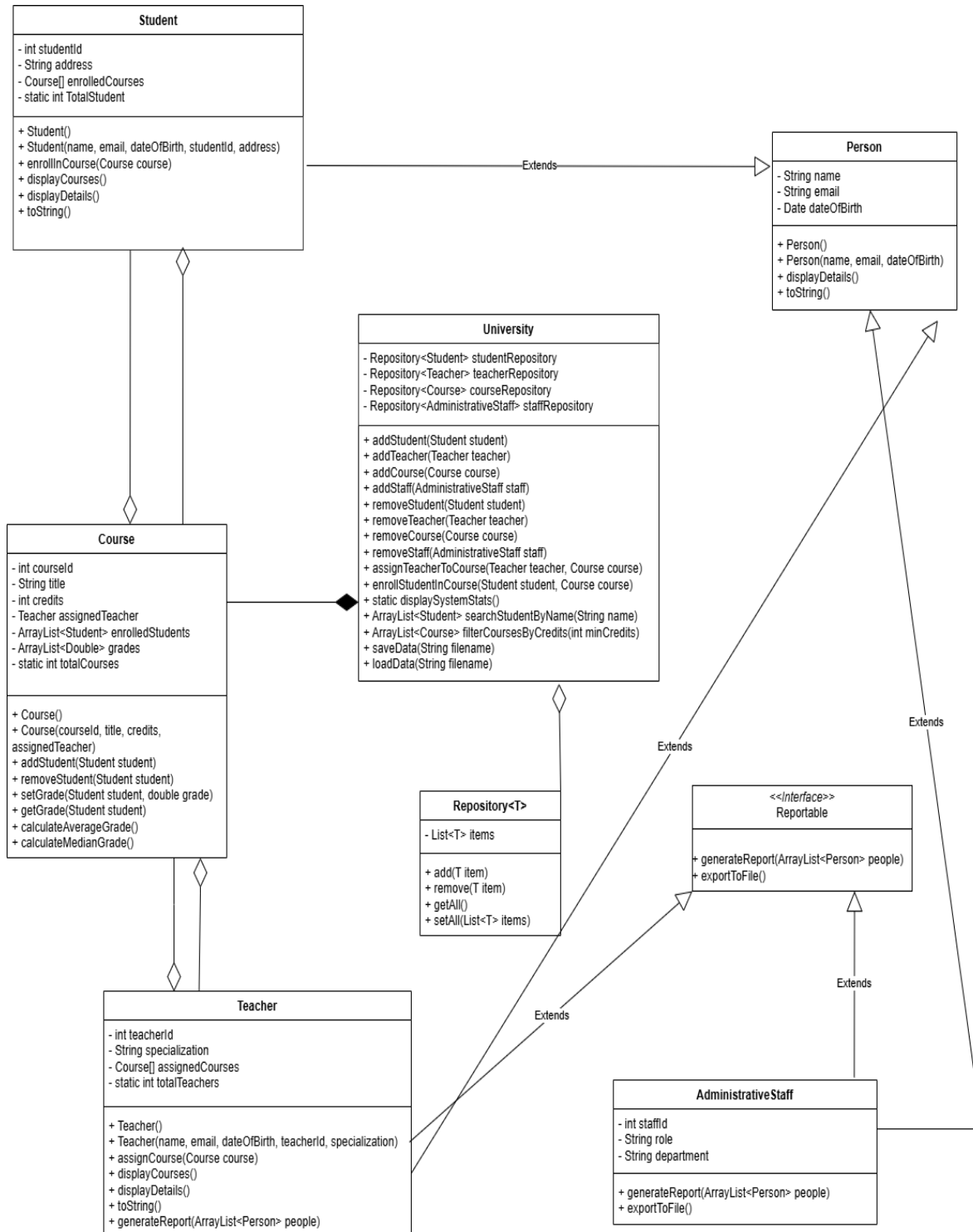
Message

System Statistics:
Total Students: 6
Total Teachers: 6
Total Courses: 5

OK

Back

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>