

**Faculty of Engineering, Built Environment, and Information Technology**

|  |
| --- |
| **Course Details:**  Module Code and Title : TCS3344 ADVANCED PROGRAMMING  Program of Study : **Bachelor of Information Technology**  Lecturer’s Name : **Ms. Hafizah Razak**  Semester : **FEB – MAY 2023**  **Assignment Details:**  Title of Assignment : TCS3344 ADVANCED PROGRAMMING  Submission Date : **Thursday, 4st May @ 11.59PM**  .  .  Type (\* please tick [√]) : **GROUP** **INDIVIDUAL**  **Declaration by student(s):**  *I/We, hereby declare that the attached assignment is my/our own work and understand that if I/we am/are suspected of plagiarism or another form of cheating, my/our work will be referred to the Program Coordinator/Head of Faculty who may, as a result recommend to the Examinations Board on academic disciplinary action including expulsion for the SEGi University and Colleges.*  **Student’s Details:**  **Name: *YOSSEF GALALELDEEN SABER ABDELHAMID MOHAMED EISA***  **Student ID: SUKD2102049 Signature \_\_\_***Yossef***\_\_\_\_\_\_\_\_**  **.**  **Overall Marks:**  /100  **Reminder:**   1. Students are reminded to keep a copy of all the coursework submitted. 2. All LATE SUBMISSION will be DEDUCTED 10% a day up to a maximum of FIVE (5) days, where subsequently, the coursework will be awarded ZERO (0). |

|  |  |  |  |
| --- | --- | --- | --- |
| **SEGI UNIVERSITY**  **FACULTY OF ENGINEERING, BUILT ENVIRONMENT**  **AND INFORMATION TECHNOLOGY**  **Bachelor of Information Technology** | | | |
| **Module Name** | **:** | Advanced Programming |
| **Module Code** | **:** | TCS3344 |
| **Academic Year** | **:** | Feb – May 2023 |
| **Lecturer** | **:** | Ms Hafizah Razak |
| **Deadline** | **:** | Week 12 (Thursday, 4st May 2023) |

**General Instructions**

Use the following format for the preparation of the *assignment submission*.

* Microsoft Word
* Paper size : A4
* Font size : 14 points for title/heading, 12 points for contents and 8-9 point for headers/footers
* Font type: Arial
* Line spacing : 1.5
* Alignment: Justified
* Number all pages sequentially.
* Number all Figures and Tables sequentially and refer to them in the text.

**WARNING**

You are required to submit your work within the bounds of the University Infringement of Assessment Regulations. Plagiarism, paraphrasing and downloading large amounts of information from external sources will not be tolerated and will be dealt with severely. However, you should make full use of any source material, which would normally be an occasional sentence and/or paragraph (duly referenced) followed by your own critical analysis/evaluation.

The work must be entirely your own. You will receive no marks for work that is not your own. The safety of your assessments is your responsibility. You must not permit another student access to your work. Where referencing is required, Harvard or APA referencing system must be used.

* Only one person per group needs to be responsible for submitting the assignment.
* Please submit your assignment by or before the deadline.
* **LATE SUBMISSION** will be **DEDUCTED** **10% penalty** PER DAY up to a **maximum of FIVE (5) days**, where subsequently, the assignment will be awarded **ZERO (0)** mark.
* **ZERO (0)** mark will be given if there is any form of copying/plagiarism.
* **ZERO (0)** mark will be given if you fail to submit your work or fail to submit all the necessary files.

E-Learning management System

Table of Contents

[Problem Description: 2](#_Toc134104900)

[Functional Requirements: 2](#_Toc134104901)

[Registration Module: 2](#_Toc134104902)

[Course Registration Module: 2](#_Toc134104903)

[Student Result Module: 3](#_Toc134104904)

[Program Maintenance: 3](#_Toc134104905)

[Course Maintenance: 3](#_Toc134104906)

[Staff Maintenance: 3](#_Toc134104907)

[Usecase Diagram: 4](#_Toc134104908)

[Class Diagram: 5](#_Toc134104909)

[Description: 6](#_Toc134104910)

[Screenshots: 8](#_Toc134104911)

[Source Code: 16](#_Toc134104912)

# Problem Description:

The problem at hand is to develop a computerized E-Learning Management System for. The system should include modules for student registration, course registration, student result management, and maintenance of programs, courses, and staff records. The system should enable students to register courses for a particular semester, select class schedules, and receive confirmation letters upon completion of registration. Lecturers should be able to key in the necessary marks based on course assessments, and the system should generate appropriate grades based on the grading system. The system should also calculate the semester cumulative grade point (CPA) and generate the necessary reports such as results for students and overall results for courses for lecturers. Additionally, the system should allow administrative staff to maintain records of programs, courses, and staff members, and enable faculty staff to assign tutors to particular courses. The system should have two categories of staff members: academic and non-academic staff.

# Functional Requirements:

## Registration Module:

* The system shall allow new students to sign up for a user account by providing personal details such as name, identity card number, contact number, permanent address, email, etc.
* The system shall create a unique user login for each student, including a username and password.
* The system shall validate the entered user data and provide appropriate error messages in case of incorrect or missing data.

## Course Registration Module:

* The system shall allow students to register for courses for a specific semester and choose their preferred class schedules.
* The system shall generate a confirmation letter to the student upon successful course registration.
* The system shall allow admin to produce necessary reports such as course enrollment report.

## Student Result Module:

* The system shall allow lecturers to set grade for course assessments.
* The system shall generate appropriate grades based on the grading system.

## Program Maintenance:

* The system shall keep track of the program details such as program code, program name, academic year, and semesters.
* The system shall allow admin to update, delete, add, and search program records.

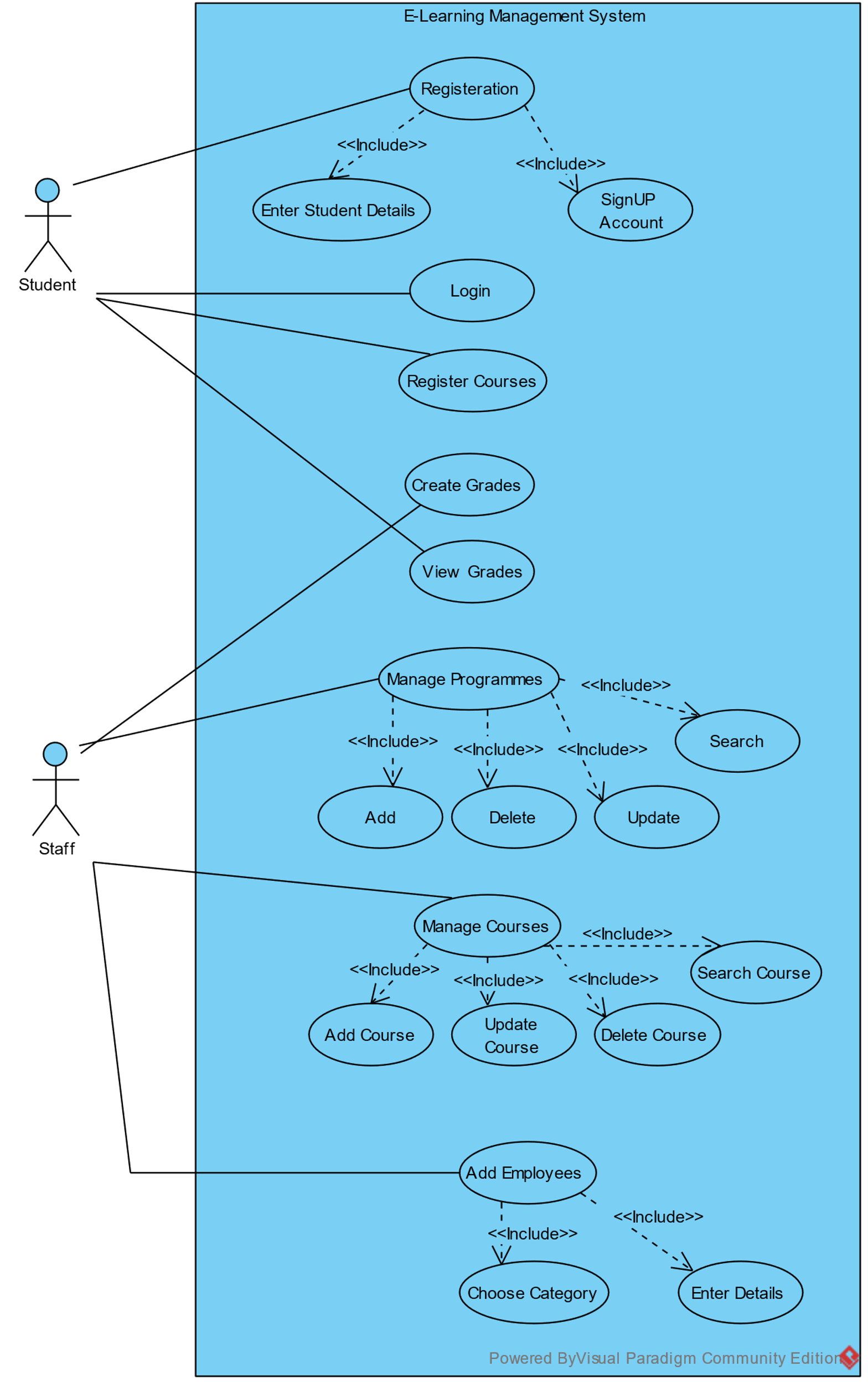
## Course Maintenance:

* The system shall allow the registrar to add, update, delete, and search for a particular course.
* The system shall allow faculty staff to assign tutors to a particular course.

## Staff Maintenance:

* The system shall allow the human resource manager to add employee details such as staff ID, name, identity card number, address, position, etc.
* The system shall distinguish between academic and non-academic staff categories.

# Usecase Diagram:



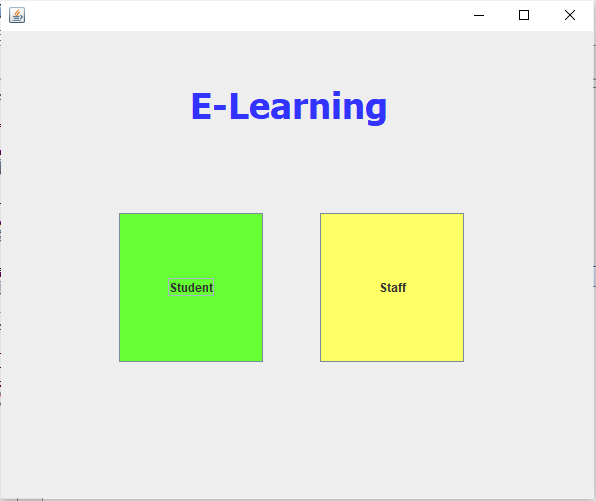
# Class Diagram:

## Description:

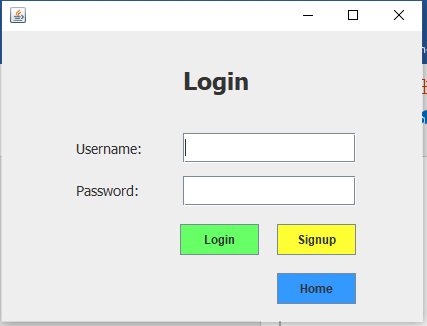
* The User class is the parent class for Student and Staff classes, which represent the two types of users in the system.
* The Student class extends the User class and adds three private fields: contactNumber, username, and password. It has a constructor that takes in values for all six fields and initializes the courses list as an empty ArrayList. It has getters and setters for the added fields and a registerCourse method that takes in a course and semester and adds it to the courses list if it doesn't already exist. It also has a getCourses method that takes in a course code and semester and returns the corresponding RegisteredCourse object if it exists in the courses list. It also has a getter for the courses list and a toString method.
* The Staff class also extends the User class and adds three private fields: id, position, and type. It has a constructor that takes in values for all six fields and getters and setters for each field.
* The Course class has four private fields: code, title, creditHrs, and tutor. It has a constructor that takes in values for all four fields and getters and setters for each field. It also has an updateDetails method that takes in values for title, tutor, and credit hours and updates the corresponding fields. It also has a toString method.
* The RegisteredCourse class has three private fields: course, semester, and grade. It has a constructor that takes in values for the course and semester fields and sets the grade field to "NA". It has getters and setters for each field and a toString method.
* ELearningManagementSystem that serves as a model for an e-learning management system. The class has several instance variables including staff, students, programs, courses, and currentStudent. The staff, students, programs, and courses instance variables are ArrayList objects that contain Staff, Student, Program, and Course objects respectively. The currentStudent instance variable is used to keep track of the currently logged in student.
* The class has a constructor that initializes the instance variables with some data. It creates two staff objects and adds them to the staff list. It also creates two student objects and adds them to the students list. Finally, it registers the first student in two courses by calling the registerCourse method of the Student class.
* The class has several methods that allow access to the data in the instance variables. These methods include login, signup, getCourses, getPrograms, getStudents, getStudent, addProgram, getProgram, addCourse, getCourse, removeProgram, removeCourse, getTutors, addStaff, registerCourse, and getRegisteredCourses.
* The login method takes a username and password as parameters and checks if there is a student with that username and password. If there is, it sets the currentStudent variable to the student object and returns true. Otherwise, it returns false.
* The signup method takes a Student object as a parameter and checks if there is already a student with the same username. If there isn't, it adds the student to the students list and returns true. Otherwise, it returns false.
* The getCourses method returns the courses list.
* The getPrograms method returns the programs list.
* The getStudents method returns the students list.
* The getStudent method takes a username as a parameter and searches for a student object with that username in the students list. If it finds a match, it returns the student object. Otherwise, it returns null.
* The addProgram method takes a Program object as a parameter and checks if there is already a program with the same code. If there isn't, it adds the program to the programs list and returns true. Otherwise, it returns false.
* The getProgram method takes a program code as a parameter and searches for a program object with that code in the programs list. If it finds a match, it returns the program object. Otherwise, it returns null.
* The addCourse method takes a Course object as a parameter and checks if there is already a course with the same code. If there isn't, it adds the course to the courses list and returns true. Otherwise, it returns false.
* The getCourse method takes a course code as a parameter and searches for a course object with that code in the courses list. If it finds a match, it returns the course object. Otherwise, it returns null.
* The removeProgram method takes a Program object as a parameter and removes it from the programs list.
* The removeCourse method takes a Course object as a parameter and removes it from the courses list.
* The getTutors method returns an ArrayList of Staff objects that have the type "Academic".
* The addStaff method takes a Staff object as a parameter and checks if there is already a staff member with the same ID.

# Screenshots:

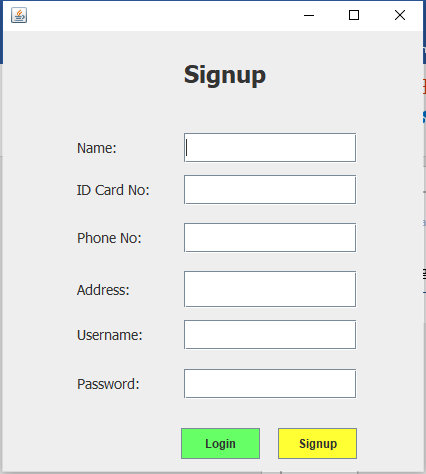
**MainFrame**



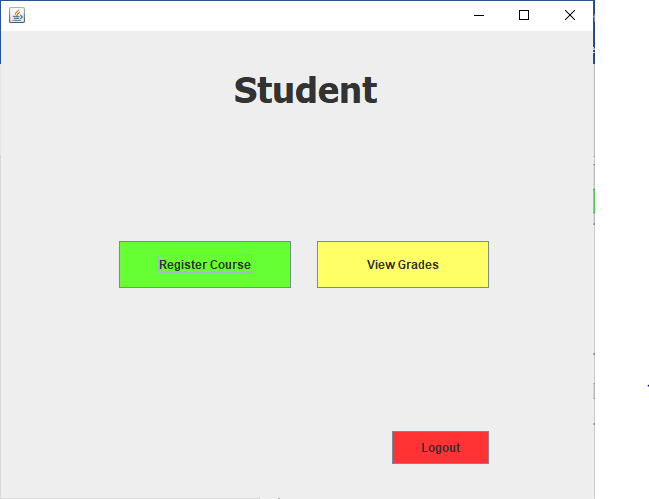
Student Login:



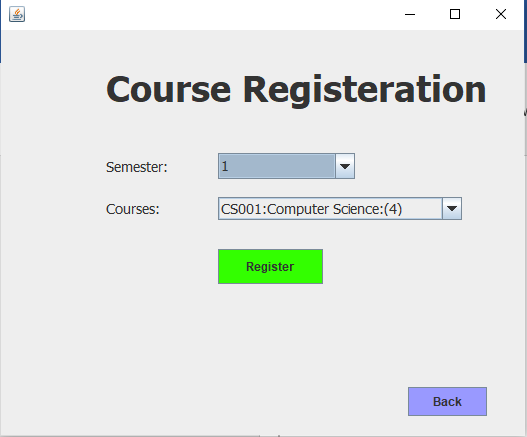
Student Signup:



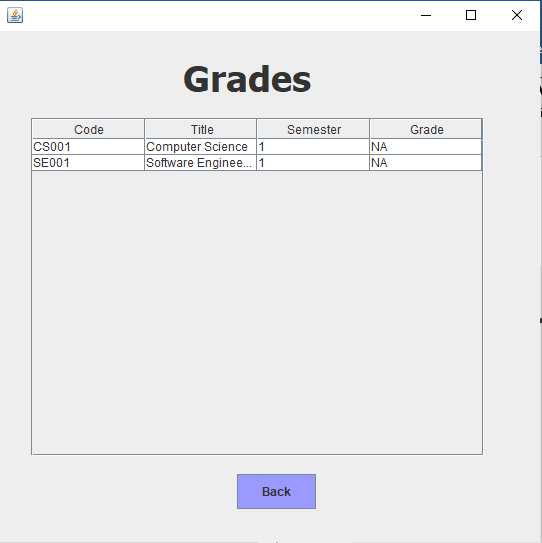
Student Home:



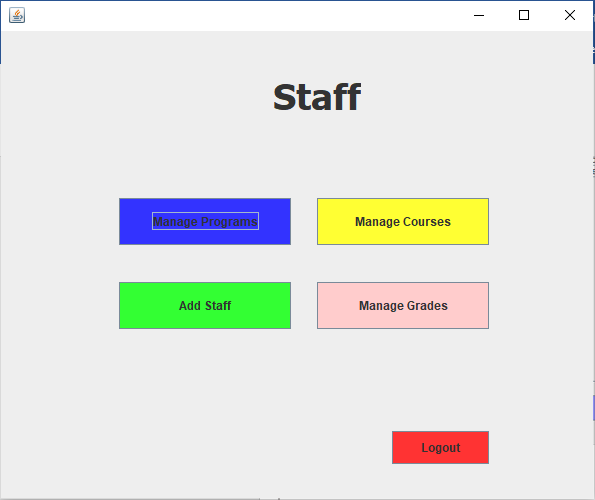
Register Course:



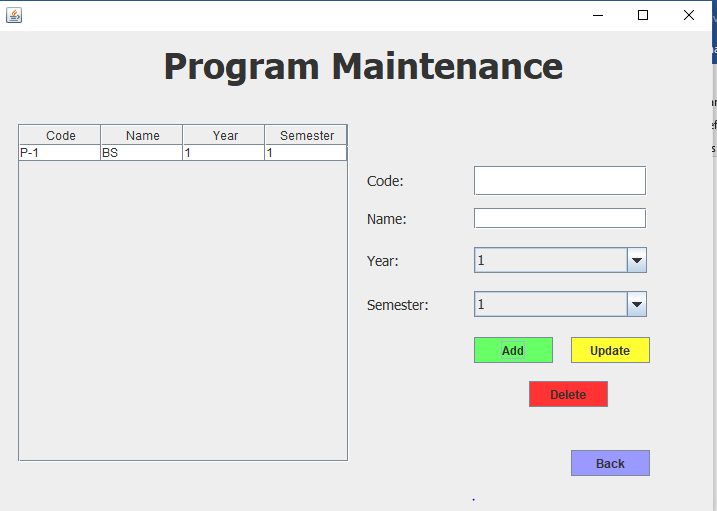
View Grades:



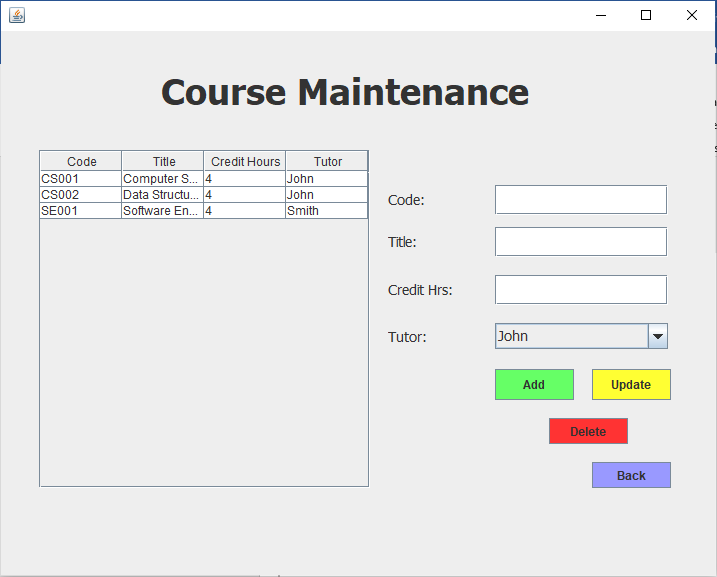
Staff Home:



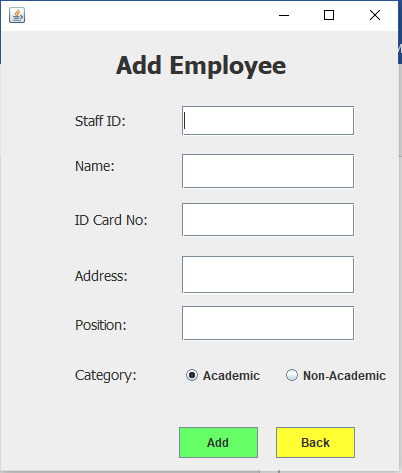
Program Maintenance:



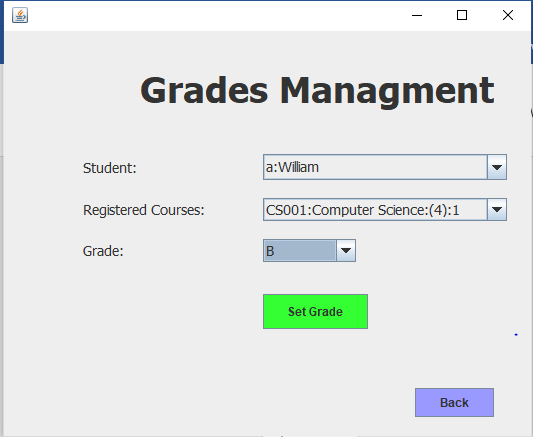
Course Maintenance:



Add Staff:



Manage Grades:



# Source Code:

|  |
| --- |
| User.java |

public class **User** {

private String name;

private String icNumber;

private String address;

// Constructor

public **User**(String name, String icNumber, String permanentAddress) {

this.name = name;

this.icNumber = icNumber;

this.address = permanentAddress;

}

// Getters and setters

public String **getName**() {

return name;

}

public void **setName**(String name) {

this.name = name;

}

public String **getIcNumber**() {

return icNumber;

}

public void **setIcNumber**(String icNumber) {

this.icNumber = icNumber;

}

public String **getPermanentAddress**() {

return address;

}

public void **setPermanentAddress**(String permanentAddress) {

this.address = permanentAddress;

}

@Override

public String **toString**() {

return super.toString(); //To change body of generated methods, choose Tools | Templates.

}

}

|  |
| --- |
| Student.java |

import java.util.ArrayList;

import java.util.List;

public class **Student** extends User {

private String contactNumber;

private String username;

private String password;

private List<RegisteredCourse> courses;

// Constructor

public **Student**(String name, String icNumber, String permanentAddress, String contactNumber, String username, String password) {

super(name, icNumber, permanentAddress);

this.contactNumber = contactNumber;

this.username = username;

this.password = password;

this.courses = new ArrayList<>();

}

public String **getContactNumber**() {

return contactNumber;

}

public void **setContactNumber**(String contactNumber) {

this.contactNumber = contactNumber;

}

public String **getUsername**() {

return username;

}

public void **setUsername**(String username) {

this.username = username;

}

public String **getPassword**() {

return password;

}

public void **setPassword**(String password) {

this.password = password;

}

public boolean **registerCourse**(Course course, int semester) {

System.*out*.println(course);

if (getCourses(course.getCode(), semester) != null) {

return false;

}

courses.add(new RegisteredCourse(course, semester));

return true;

}

public RegisteredCourse **getCourses**(String code, int semester) {

for (RegisteredCourse c : courses) {

if (c.getCourse().getCode().equals(code)

&& semester == c.getSemester()) {

return c;

}

}

return null;

}

public List<RegisteredCourse> **getCourses**() {

return courses;

}

@Override

public String **toString**() {

return username + ":" + getName();

}

}

|  |
| --- |
| Staff.java |

public class **Staff** extends User {

private String id;

private String position;

private String type;

public **Staff**(String id, String name, String icNumber, String permanentAddress, String position, String type) {

super(name, icNumber, permanentAddress);

this.id = id;

this.position = position;

this.type = type;

}

public String **getId**() {

return id;

}

public void **setId**(String id) {

this.id = id;

}

public String **getPosition**() {

return position;

}

public void **setPosition**(String position) {

this.position = position;

}

public String **getType**() {

return type;

}

public void **setType**(String type) {

this.type = type;

}

}

|  |
| --- |
| Course.java |

public class **Course** {

private String code;

private String title;

private int creditHrs;

private String tutor;

public **Course**(String code, String title, int creditHrs, String tutor) {

this.code = code;

this.title = title;

this.creditHrs = creditHrs;

this.tutor = tutor;

}

public String **getCode**() {

return code;

}

public String **getTitle**() {

return title;

}

public void **setTitle**(String title) {

this.title = title;

}

public int **getCreditHrs**() {

return creditHrs;

}

public void **setCreditHrs**(int creditHrs) {

this.creditHrs = creditHrs;

}

public String **getTutor**() {

return tutor;

}

public void **setTutor**(String tutor) {

this.tutor = tutor;

}

public void **updateDetails**(String title, String tutor, int crHrs) {

setTitle(title);

setTutor(tutor);

setCreditHrs(creditHrs);

}

@Override

public String **toString**() {

return code+":"+title+":("+creditHrs+")";

}

}

|  |
| --- |
| Program.java |

public class **Program** {

private String code;

private String name;

private int year;

private int semester;

public **Program**(String code, String name, int year, int semester) {

this.code = code;

this.name = name;

this.year = year;

this.semester = semester;

}

public String **getCode**() {

return code;

}

public String **getName**() {

return name;

}

public void **setName**(String name) {

this.name = name;

}

public int **getYear**() {

return year;

}

public void **setYear**(int year) {

this.year = year;

}

public int **getSemester**() {

return semester;

}

public void **setSemster**(int semester) {

this.semester = semester;

}

public void **updateDetails**(String name, int year, int semester) {

setName(name);

setYear(year);

setSemster(semester);

}

}

|  |
| --- |
| RegisteredCourse.java |

public class **RegisteredCourse** {

private Course course;

private int semester;

private String grade;

public **RegisteredCourse**(Course course, int semester) {

this.course = course;

this.semester = semester;

this.grade = "NA";

}

public Course **getCourse**() {

return course;

}

public void **setCourse**(Course course) {

this.course = course;

}

public int **getSemester**() {

return semester;

}

public void **setSemester**(int semester) {

this.semester = semester;

}

public String **getGrade**() {

return grade;

}

public void **setGrade**(String grade) {

this.grade = grade;

}

@Override

public String **toString**() {

return course.toString()+":"+semester;

}

}

|  |
| --- |
| ELearningManagementSystem.java |

import java.util.ArrayList;

import java.util.List;

public class **ELearningManagementSystem** {

private List<Staff> staff;

private List<Student> students;

private List<Program> programs;

private List<Course> courses;

private Student currentStudent;

public **ELearningManagementSystem**() {

staff = new ArrayList<>();

students = new ArrayList<>();

programs = new ArrayList<>();

courses = new ArrayList<>();

staff.add(new Staff("Emp1", "John", "1223435465", "8 b", "Lecturer", "Academic"));

staff.add(new Staff("Emp2", "Smith", "1223435465", "9 b", "Lecturer", "Academic"));

courses.add(new Course("CS001", "Computer Science", 4, "John"));

courses.add(new Course("CS002", "Data Structures", 4, "John"));

courses.add(new Course("SE001", "Software Engineering", 4, "Smith"));

students.add(new Student("William", "453453", "2 b 3", "3453454", "a", "a"));

students.add(new Student("Alen", "934848", "24 b 3", "3298904", "b", "b"));

students.get(0).registerCourse(courses.get(0), 1);

students.get(0).registerCourse(courses.get(2), 1);

}

public boolean **login**(String un, String pass) {

for (Student student : students) {

if (student.getUsername().equals(un) && student.getPassword().equals(pass)) {

currentStudent = student;

return true;

}

}

return false;

}

public boolean **signup**(Student student) {

if (getSudent(student.getUsername()) == null) {

students.add(student);

return true;

}

return false;

}

public List<Course> **getCourses**() {

return courses;

}

public List<Program> **getPrograms**() {

return programs;

}

public List<Student> **getStudents**() {

return students;

}

public Student **getSudent**(String username) {

for (Student student : students) {

if (student.getUsername().equals(username)) {

return student;

}

}

return null;

}

public boolean **addProgram**(Program program) {

if (getProgram(program.getCode()) == null) {

programs.add(program);

return true;

}

return false;

}

public Program **getProgram**(String code) {

for (Program program : programs) {

if (program.getCode().equals(code)) {

return program;

}

}

return null;

}

public boolean **addCourse**(Course course) {

if (getCourse(course.getCode()) == null) {

courses.add(course);

return true;

}

return false;

}

public Course **getCourse**(String code) {

for (Course course : courses) {

if (course.getCode().equals(code)) {

return course;

}

}

return null;

}

public void **removeProgram**(Program program) {

programs.remove(program);

}

public void **removeCourse**(Course course) {

courses.remove(course);

}

public ArrayList<Staff> **getTutors**() {

ArrayList<Staff> tutors = new ArrayList<>();

for (Staff s : staff) {

if (s.getType().equals("Academic")) {

tutors.add(s);

}

}

return tutors;

}

public boolean **addStaff**(Staff st) {

for (Staff s : staff) {

if (s.getId().equals(st.getId())) {

return false;

}

}

staff.add(st);

return true;

}

public boolean **registerCourse**(Course course, int semester) {

return currentStudent.registerCourse(course, semester);

}

public List<RegisteredCourse> **getRegisteredCourses**() {

return currentStudent.getCourses();

}

public static void ***main***(String[] args) {

ELearningManagementSystem system = new ELearningManagementSystem();

new MainFrame(system).setVisible(true);

}

}