1. Introduction:

1.1 Purpose:

The purpose of this document is to provide the domain of the system and also the users with the needed information to understand this system wholly, how does it work, how to overcome any issues that might occur during operation and the process flow of all the services offered by the system.

1.2 List of Definitions:

* Functional Requirements: Are the required capabilities of the software by the client that it should be able to perform.
* Non-Functional requirements: Are the required capabilities for the software to run smoothly, be reliable and others (or in other words, not requested by the client specifically but must be present in a software system) i.e. Hardware, Code implementation, Local and Global Standards, etc.
* Class Model: Is a model that works on connecting between the different classes (Modules) in a graphical way to represent the hierarchy of the system and to show the functionalities of each class in specific.
* Use-Case Diagram: A diagram that represents how the process flow of the different system modules works hand in hand.
* Narrative Descriptions: Are descriptions of the different modules of the system and how to make them function.

1.3 Scope:

* Name of the product: “**Registration System**”.
* The software mainly works on registering any new data regarding

Students, professors and offered courses.

1.4 Overview:

* General Description.
* System Requirements.
* Use-Case diagrams.
* Narrative Description of Use Cases.
* Requirements Validation.
* Class Model.
* State Diagram.
* Interaction Diagram.
* Detailed Class Diagram.
* User Interface Design.
* Client-Object Relation Diagram.
* Detailed Design.
* Testing.
* Estimated Project Cost.
* User Guide.

2. General Description:

2.1. Product Perspective:

As it is obvious in the product’s name, this kind of system is dedicated for registering any new entries to the system of the faculty and handles how it is stored in the database, but from a much wider perspective of the faculty’s different systems as a whole, we’d find that the Registration system works hand in hand with other systems , so for example, the payment system works on verifying the payment of a student for his required academic fees, verifies also that monthly salaries are being paid to the professors , Executives, etc. and so on , so in order for a student to be applied to a course in the registration system , the payment system needs to verify that he did actually pay his tuitions, and so is the product a part(module) that functions in a bigger, integrated system.

2.2. General Capabilities:

* Add/drop/withdraw courses for students.
* Check if there are available vacancies in the course.
* Check for available halls.
* Register new students, courses, professors on the faculty’s system.

2.3. General Constraints:

The general constraints of this system lies in that it is such a rigid system that is designed based on a waterfall model that does not support evolution and hence does not support adaptability, Moreover, Security is not implemented in this system and so it is not fortified against any malware attacks which would lead to the loss of all the data on the system.

2.4. User Characteristics:

2.4.1. Professors:

Professors' role mainly lays in registering new students in the offered courses, assigns themselves new courses as long as the numbers of credit hours of the courses they teach are 9 credit hours or less, and is also able to open or close registration of any course.

2.5. Environmental Description:

Our System is designed so that it can be operated on lowly-efficient computers with any operating system either it’s a new or an old version, also the software will be utilizing the hard drives or database servers, whichever is available, so it’s a must to have any kind of hardware dedicated for storage in order for the system to function.

2.6. Assumptions and Dependencies:

In the registration system, we do assume that add/drop/withdraw operations must be offered during semesters so that students could be assigned to the right courses and therefore, store the correct information about each student during the semester in the faculty’s database, same applies for the professors regarding the courses they teach and also the salaries they’re getting paid which must be processed according to the number of credit hours assigned to their courses.

1. System Requirements:
   1. Functional Requirements:
      1. User add or drop or withdraw a course to a student
      2. No student could be registered in a course twice.
      3. The student couldn’t drop or withdraw a course unless it's already registered in it.
      4. The student can't add a closed course.
      5. User make a professor teach a certain course.
      6. Each professor can't teach more than 9 credit hours per semester.
      7. Users add a student to course name list.
      8. Each course can't have more than 1 instructor and 100 students.
      9. The system should be able to calculate each student's payment based on his registered credit hours.
      10. The system should be able to calculate each professor's salary based on his teaching credit hours.
      11. The system should reserve an empty class room for the used in the provided time slot.
      12. The user can't reserve a classroom after 20:00 o'clock.
      13. The user should be able to open or close any course he wants.
      14. The user can't register a new student and a professor at the same time.
      15. The user can't add a new course with an invalid professor's name.
   2. Non-functional Requirements:
      1. The program should be written in java programming language.
      2. The program should be delivered by 5th of May.
      3. The program shouldn’t take more than 50 Megabytes of memory.
      4. The program should have a user guide.
      5. The system should have the faculty logo on the home page.
      6. The program should work on any laptop or PC running windows operating system.
      7. Any process in the program shouldn't take more than 3 minutes.

4. Use case diagram:

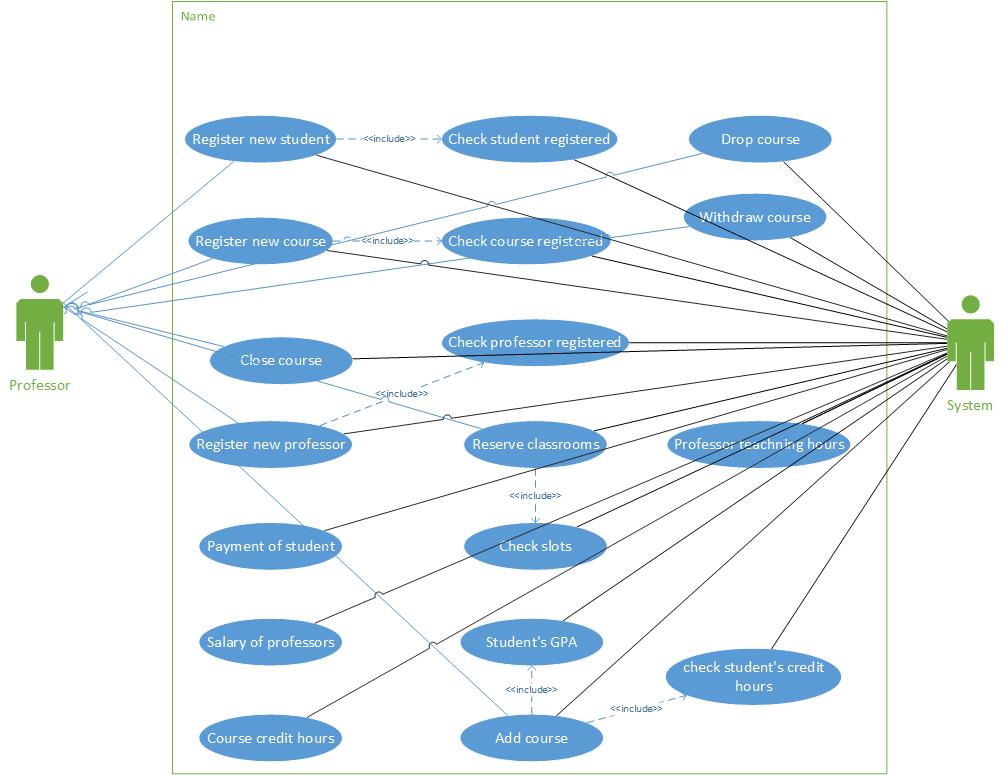


Figure 1:Use case Diagram

5. Narrative description of Use cases:

5.1 Register new professor Use case :

Use case name: Register new professor

Related Requirements: System approval

Goal in Context: Request to register professor in the system

Preconditions: Professor is no previously registered

Successful End Condition: Professor is registered in the system

Failed End Condition: Professor is rejected

Primary Actors: Administrator

Included cases: Check professor is registered

Main Flow:

1 Administrator request to add professor

2 Administrator enters professor information

Include::Check Professor Information is checked

Professor is registered

3 Professor is registered to the system

5.2 Close Course Use case :

Use case name: Close Course

Related Requirements: System approval

Goal in Context: Professor requests to close a registered course

Preconditions: Course is already registered and available

Successful End Condition: Course is closed

Failed End Condition: Course closing is rejected

Primary Actors: Administrator

Secondary Actors: None

Included cases: Course Registered

Main Flow:

1 Professor requests the system to close a

course

2 Professor enters course name and code

Include::Course Course information is checked Is Registered

3 Course is closed

5.3 Add course use case

Use case name: add course

Related Requirements: Administrator approval

Goal in Context: Administrator requests to add a course to a student

Preconditions: Course is not already registered at the student

Successful End Condition: Course is already added

Failed End Condition: Course addition is rejected

Primary Actors: Administrator

Secondary Actors: None

Included cases: None

Main Flow:

1. Administrator requests to add a course for a Student

2 Administrator enters course name and code

3 System checks if it is already added at the student

4 Course is added to the student

Extensions:

3.1 Course is already added

3.2 Course adding is rejected

5.4 Reserve Classroom use case

Use case name: Reserve classroom

Related Requirements: System approval

Goal in Context: Administrator requests to reserve a room

Preconditions: Classroom is not already booked

Successful End Condition: Classroom is reserved

Failed End Condition: Classroom reservation is rejected

Primary Actors: Administrator

Secondary Actors: None

Included cases: Check slots

Main Flow:

1 Administrator requests the system to reserve a CR.

2 Administrator enters classroom number

3 Administrator enters time slot

Include::Check Time slot is checked Slots

4 Classroom is reserved for the desired

time slot

5.5 Add new course use case

Use case name: add course

Related Requirements: Administrator approval

Goal in Context: Administrator requests to add a course to a student

Preconditions: Course is not already registered at the student

Successful End Condition: Course is added

Failed End Condition: Course addition is rejected

Primary Actors: Administrator

Secondary Actors: None

Included cases: None

Main Flow:

1. Administrator requests to add a course for a student
2. Administrator enters course name and code
3. System checks if it is already added at

the student

1. Course is added to the student

Extensions:

3.1 Course is already added

3.2 Course adding is rejected

5.6 Drop course use case

Use case name: drop course

Related Requirements: Administrator approval

Goal in Context: Administrator requests to drop a course for a student

Preconditions: Course is not already registered at the student

Successful End Condition: Course is dropped

Failed End Condition: Course dropping is rejected

Primary Actors: Administrator

Secondary Actors: None

Included cases: None

Main Flow:

1. Administrator requests to drop a course for a

Student

2 Administrator enters course name and code

3 System checks if it is added for the student

4 Course is dropped for the student

Extensions:

3.1 Course is not found

3.2 Course dropping is rejected

5.7 Withdraw course use case

Use case name: withdraw course

Related Requirements: Administrator approval

Goal in Context: Administrator requests to withdraw a course for a student

Preconditions: Course is not already registered at the student

Successful End Condition: Course is withdrawn

Failed End Condition: Course withdrawing is rejected

Primary Actors: Administrator

Secondary Actors: None

Included cases: None

Main Flow:

1. Administrator requests to withdraw a course

for a student

1. Administrator enters course name and code

3 System checks if it is added for the student

4 Course is withdrawn for the student

Extensions:

3.1 Course is not found

3.2 Course withdrawing is rejected

7. Class model:

The super users of the system presented in the professors should be able to register new students, add new courses to the faculty, add/drop or withdraw courses for the currently registered students and register new professors ,in addition to that the system should be able to calculate the payment of the students and the professors salaries, each student could add and drop courses based on his GPA with maximum number of 20 credit hours ;as all the courses have a fixed number of 3, credit hours, while the professor could teach courses with maximum of 3 courses per semester.

The system should be able to reserve an empty classroom whenever a super user needs one, based on the reserved time slots of each classroom where each time slot is fixed to 3 hours and the user could only reserve a hall from the beginning of each slot and the classroom is marked as a reserved one until the end of its 3 hours slot.

**Nouns' extraction:**

* The nouns that can be identified are:

Classroom, student, professor, course, credit hour, payment,

Salary, GPA, system, faculty, super user, slots.

* Nouns that lies outside the problem boundaries:

System, slots, credit hour, GPA, faculty, super user, payment,

Salary.

* Abstract nouns that represent ideas (classes' candidates):

Classroom, student, professor, course.

8. State Diagram:

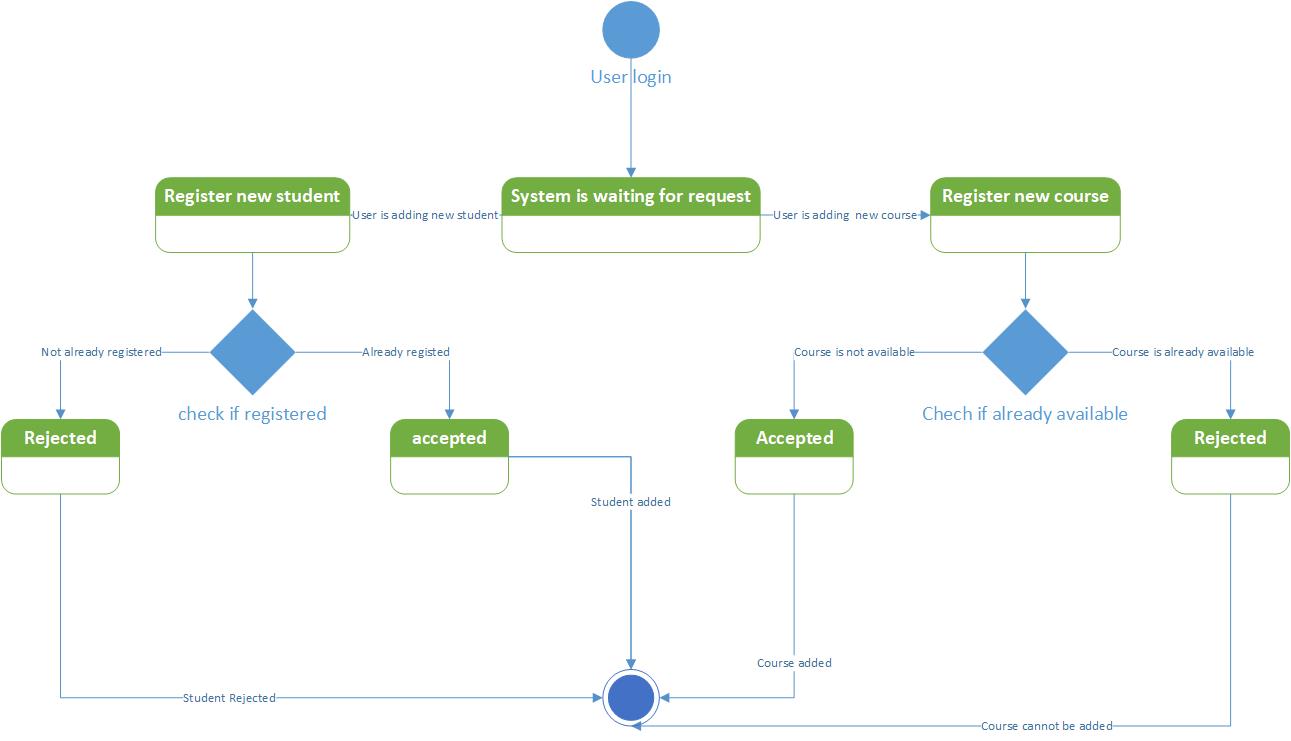


Figure 2.1:State Diagram

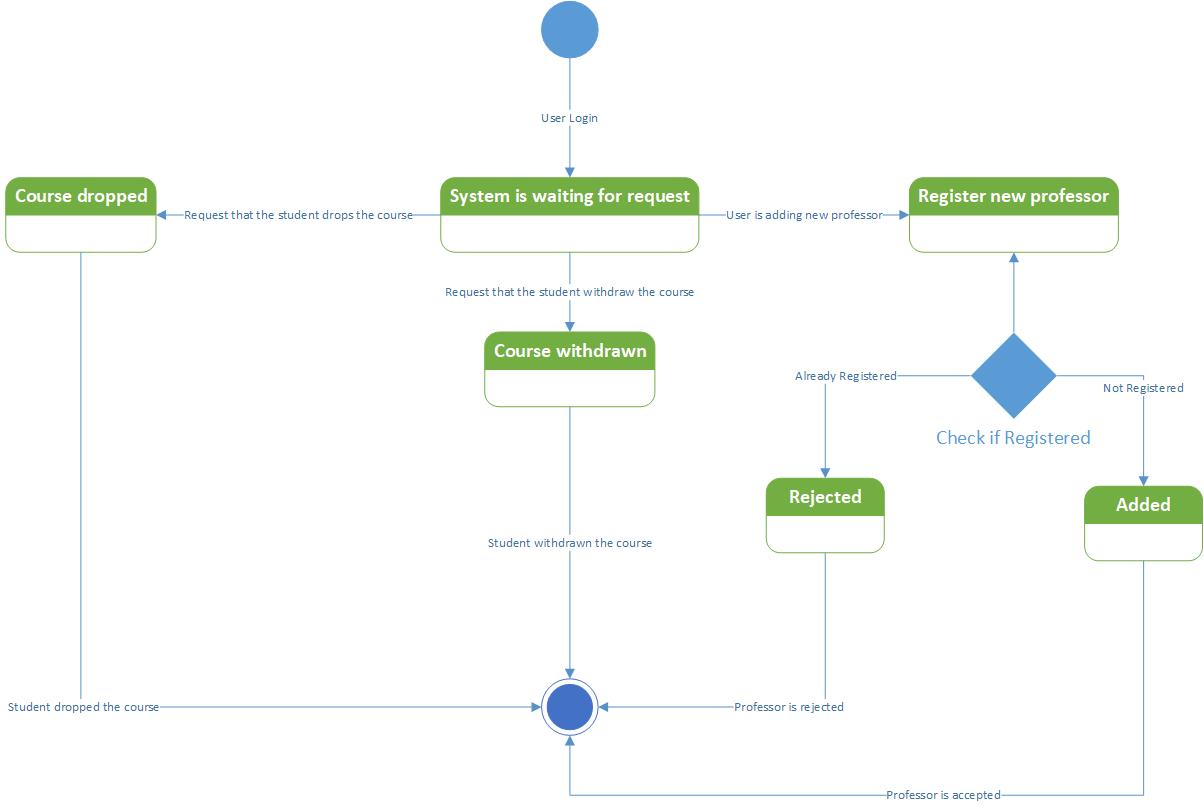


Figure 2.2:State Diagram

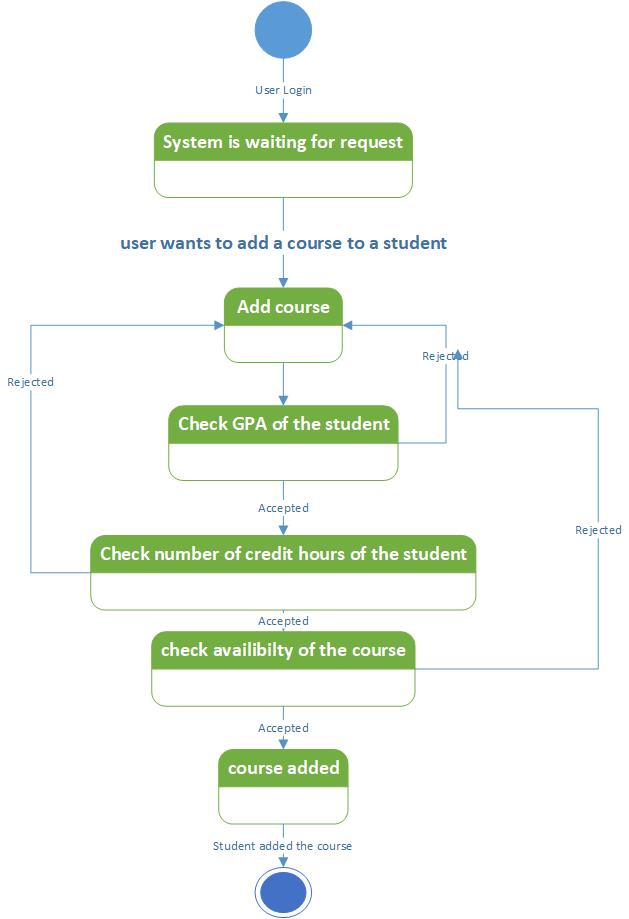


Figure 2.3:State Diagram

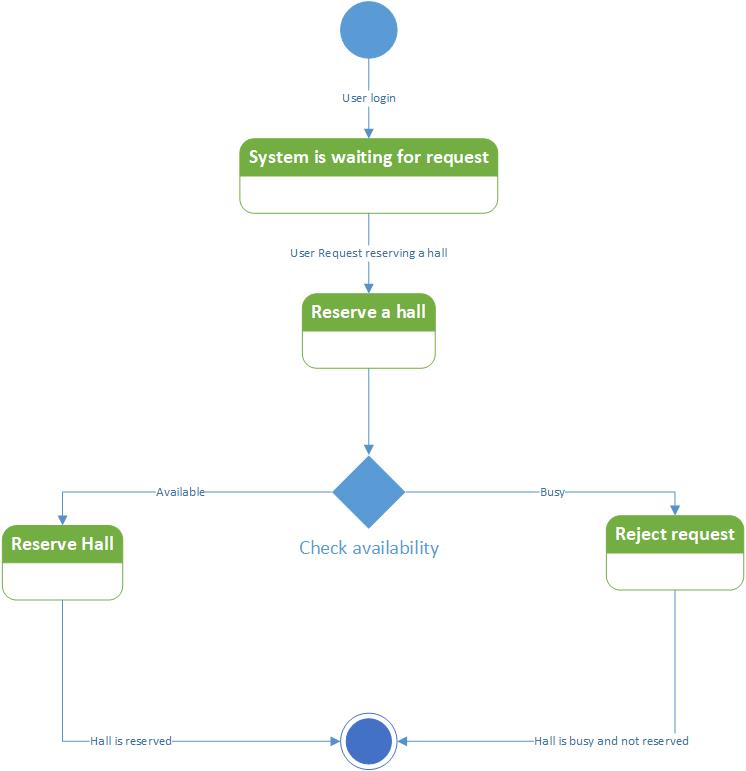


Figure 2.4:State Diagram

9. Interaction Diagram:

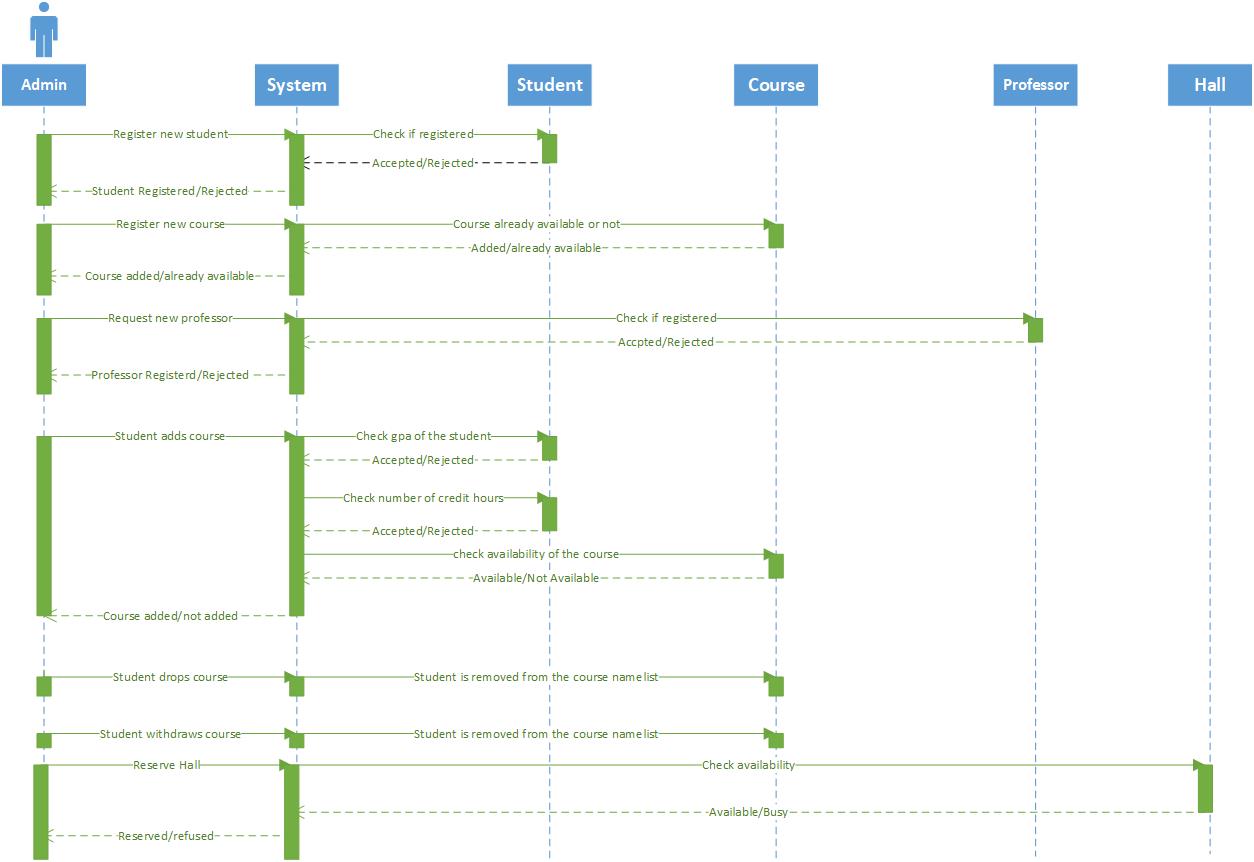


Figure 3: Sequence Diagram

11. Detailed Class Diagram:

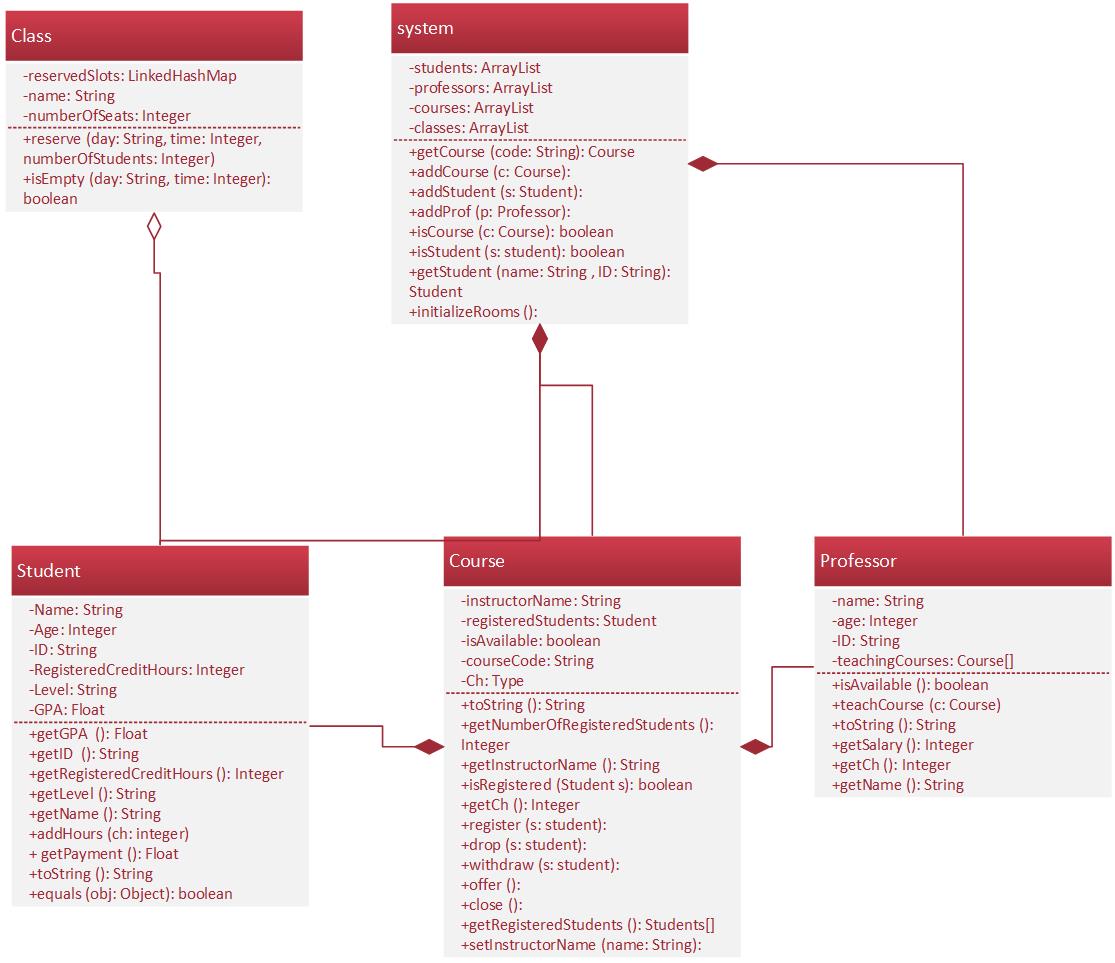


Figure 4: Detailed Class diagram

**12. User Interface Design:**

Our project consists of 5 windows which are:

* **Home page window:**

It consists of two panels; the top panel is a panel containing the

the faculty name and logo.

The second and the last panel consists of 4 buttons, each does a

different operation and initialize one of the other 4 windows

when clicked.

Each button has its own logo and tip text also each button

changes its color when is hovered over by the mouse.

* **Add a new course window:**

It consists of a title label on its top middle, 3 text fields for the

Course data to be entered in, each has a label right on the top of

It.

It also contains 2 radio buttons and a submit blue button on its

bottom, which saves the data if certain conditions were met

when the user clicks on it and the user could enter the data of

another course then or close the window and go back to the

home page.

* **Add/drop course window:**

It consists of a title label on its top middle, 3 text fields

2 for the student data and one for the course code, each

has its own label on the top of it.

It also contains 3 radio buttons and a submit button at the bottom, the radio buttons are for add, drop or withdraw options, the submit button is to save the entered data and do the necessary operation on them and then wait for the user to continue or close and go back to the home page.

* **Reserve an empty classroom window:**

It consists of a title, 2 text field each with its own label on the top of it and a button on the bottom of the page to save the entered data and do the operation when clicked, then pop up a message if certain conditions were met and then wait for the user to repeat or close the window and go back to the home page.

* **Registration window:**

It consists of a title, 3 text fields for the new faculty member's data to be entered in, each has its own description label on the top of it.2 radio buttons and a submit button at the bottom, the radio buttons are to choose whether the new member is a professor or a student and the submit button save the entered data and do the operation on them or pop up an error if wrong data were entered, and then it waits for the user to repeat the operation or close the window and go back to the home page.

13. Client-Object Relation Diagram:

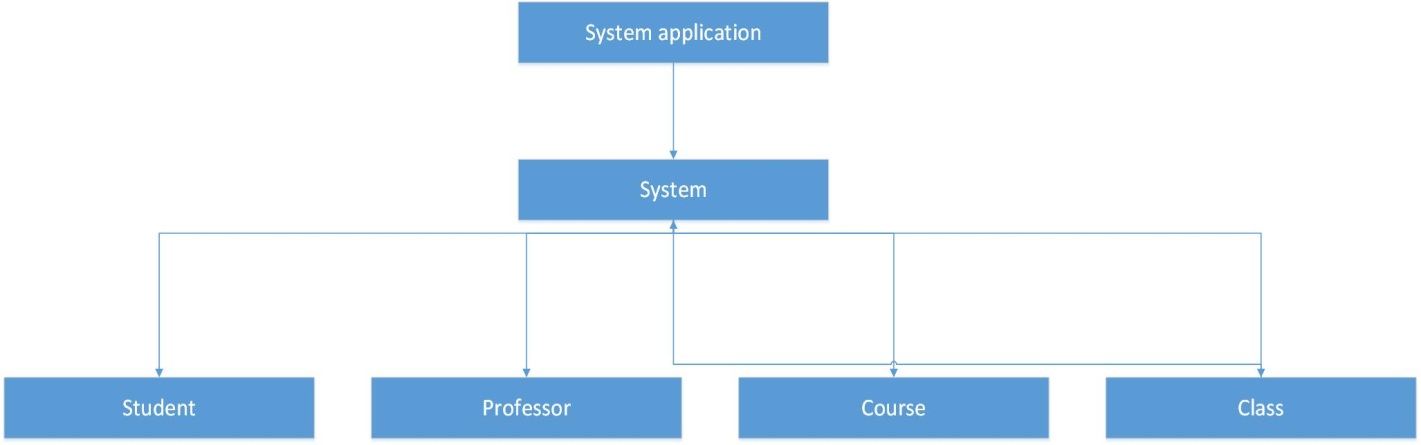


Figure 5:Client-object relation Diagram

14. Detailed Design:

Void application home page (void) {

While(True) {

If (register new member button is clicked) {

Registration window opens;

The user should start entering the new member's

name, age and ID;

The user chooses if the new member is a professor

Or a student then clicks submit;

If (Student radio button is clicked)

{

A new student with the entered data is added to

The students' system array list;

}

If (Professor radio button is clicked)

{

A new professor with the entered data is added

To the professors' array list;

}

Else {An error message is displayed;}

}

If (Reserve an empty classroom button is clicked)

{

The user should enter the desired day and time;

If (Submit button is clicked)

{

If (the user entered desired times)

{

The system pops up a message with the

Reserved classroom's name;

}

Else {the system pops up an error message;}

}

}

If (Add a new course button is clicked)

{

The user is asked to enter the course details;

The user must choose whether he wants to

Open that course as a new one or close it;

If (Submit button is clicked) {

If (the user entered all the desired data)

{

The system adds a new course with

The entered data to its array list of

Courses;

}

Else {the system pops up an error message ;}

}

}

If (Add/Drop button is clicked)

{

The user is asked to enter the student

Data and the course code;

The system checks if the entered data

Is related to a current student or not

And so, does with the course code;

If (Submit button is clicked)

{

If (the student credit hours < 20 for

A GPA of 2.0 or more OR

The credit hours < 16 for

A GPA less than 2.0)

{

The system adds the student with the

Entered data to the course whose code

Is entered array of students;

}

Else {The do nothing;}

}

}

}

}

15. Testing:

* In this section, we work on testing the inner workings of each module in our system.
* In order for the testing to be done correctly, we need to check that the code structure is working in the way it’s intended to be by using some test cases in order to assert that it do function correctly.
* We are going to take the class “Courses” from our files as an example and test it using the following test cases which are sent to the constructor of the class:  
  1)<”CSE128”,ch = sa,”Dr. Gamal abdelshafy”>  
  2)<”CSE128”,ch = “Ibrahim”,”Dr. Mohamed el dakroury”>  
  3)<97,ch = 7,”123”>
* The code Structure:  
  package Faculty;

import Users.Professor;

import Users.Student;

public class Course {

private Student[] registeredStudents;

private String courseCode;

private int ch;

private boolean isAvailable;

private String instructorName; //null by default

public Course(String courseCode,int ch,String instructorName){

this.courseCode=courseCode;

this.ch=ch;

this.registeredStudents=new Student[160];

this.instructorName=instructorName;

for(int i=0;i<160;i++)

registeredStudents[i]=null;

}

@Override

public boolean equals(Object obj){ if(((Course)obj).instructorName.equals(this.instructorName)&&((Course)obj).courseCode.equals(this.courseCode))

return true;

return false;

}

@Override

public String toString(){

return "\n"+"Course:"+"\n"+this.courseCode+"\n"+"Instructor name: "+this.instructorName+"\n"+

"Credit Hours: "+this.ch+"\n"+"Is available: "+this.isAvailable;

}

public int getNumberOfRegisteredStudents() {

int n=0;

for(int i=0;i<160;i++){

if(registeredStudents[i]!=null)

++n;

}

return n;

}

public boolean isAvailable() {

return isAvailable;

}

public boolean isRegistered(Student s){

for(int i=0;i<160;i++)

if(this.registeredStudents[i]==s)

return true;

return false;

}

public void register(Student s){ if(this.isAvailable&&!(isRegistered(s))&&this.isAvailable&&s.getRegisteredCreditHours()+this.ch<=20 &&s.getGPA()>=2.0||s.getRegisteredCreditHours()+this.ch<=12&&s.getGPA()<2.0)

{

for(int i=0;i<160;i++){

if(this.registeredStudents[i]==null){

this.registeredStudents[i]=s;

this.registeredStudents[i].addHours(this.ch);

break;

}

}

}

}

public String getCourseCode(){return this.courseCode;}

public void drop(Student s){

if(isRegistered(s)){

for(int i=0;i<160;i++)

if(this.registeredStudents[i]==s) {

registeredStudents[i].addHours(-this.ch);

registeredStudents[i] = null;

break;

}

}

}

public void withdraw(Student s){

if(isRegistered(s)){

for(int i=0;i<160;i++){

if(registeredStudents[i]==s){

registeredStudents[i]=null;

break; //don't decrement the student credit hours as he pays the CH of the withdrawn courses

}

}

}

}

public void offer(){

this.isAvailable=true;

}

public void close(){

this.isAvailable=false;

}

public Student[] getRegisteredStudents() {

return this.registeredStudents;

}

public int getCh() {

return ch;

}

public void setInstructorName(String instructorName) {

this.instructorName = instructorName;

}

public String getInstructorName() {

return instructorName;

}

}

* The expected output in each case:  
  1)course code =CSE128,ch = sa, instructorname=Dr. Gamal abdelshafy  
  2)Error should occur in such a test case as in the second parameter a String is sent to a variable of data type integer, so this path is considered infeasible.

3)Course code =123 , ch =3 ,Instructorname=123

An error will occur due to entering an invalid instructor's name.

16. Estimated Project Cost:

* Room rent=5k.
* 5Laptops = 14k.
* 5 Engineers each one with salary 2k, total salaries=10k.
* Electricity = 1k.
* Food supplies =1.5k.
* Total cost = 31.5k.

17. User Guide:

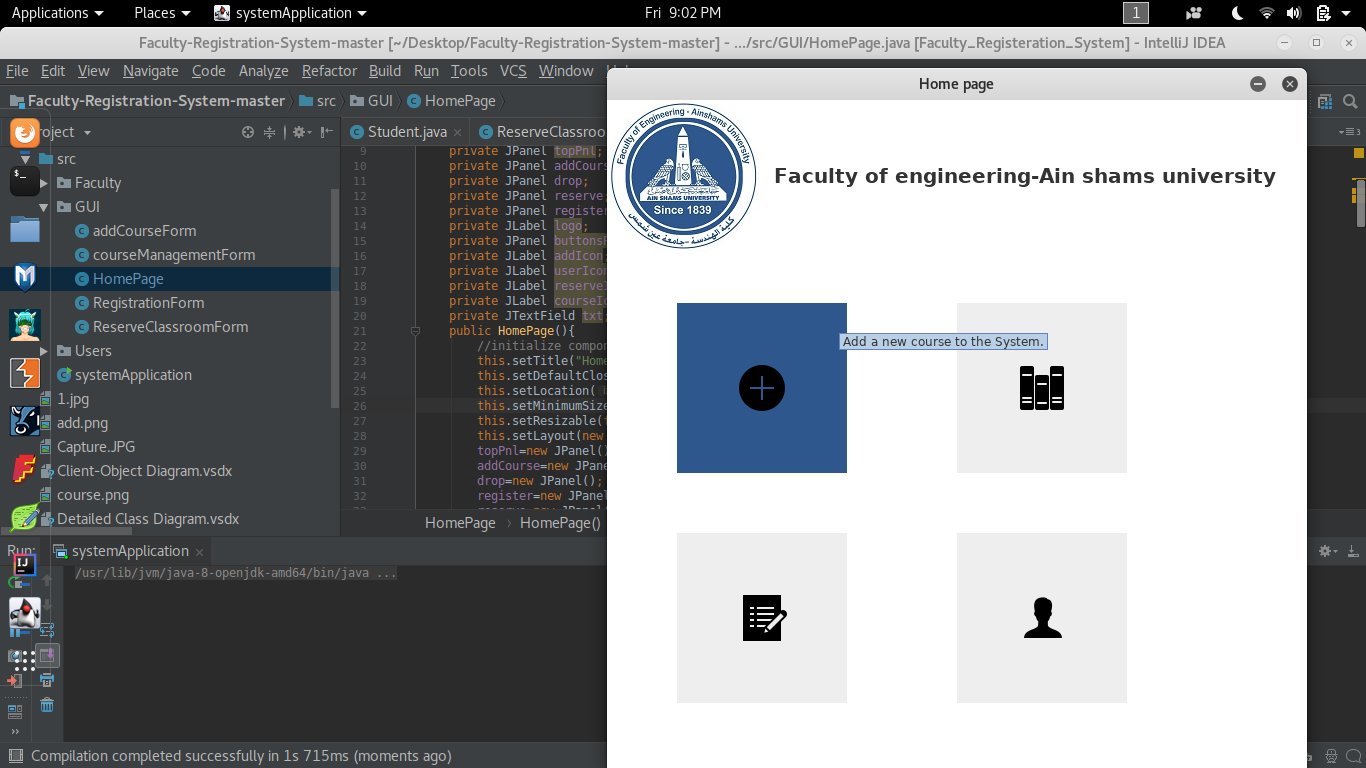


Figure 6: Add a new Course

Figure 4: Add a new Course1

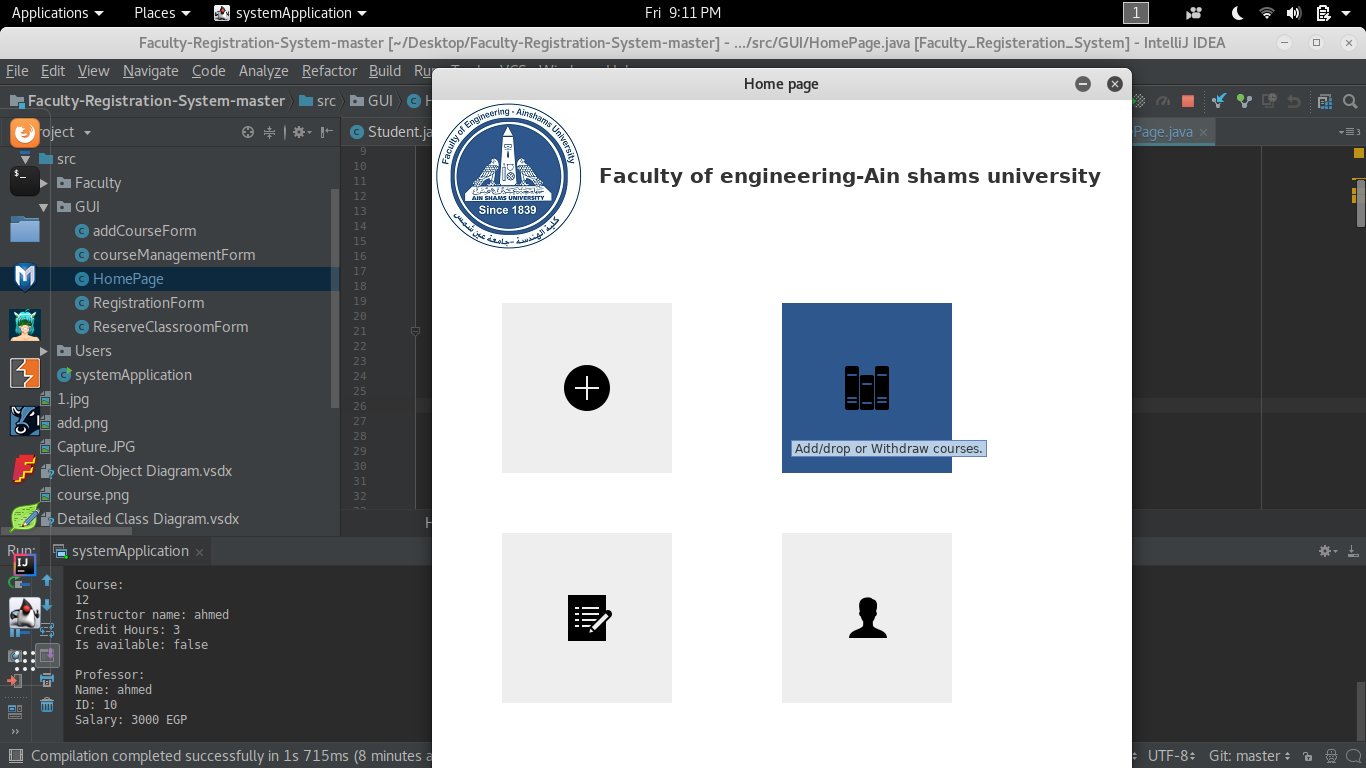
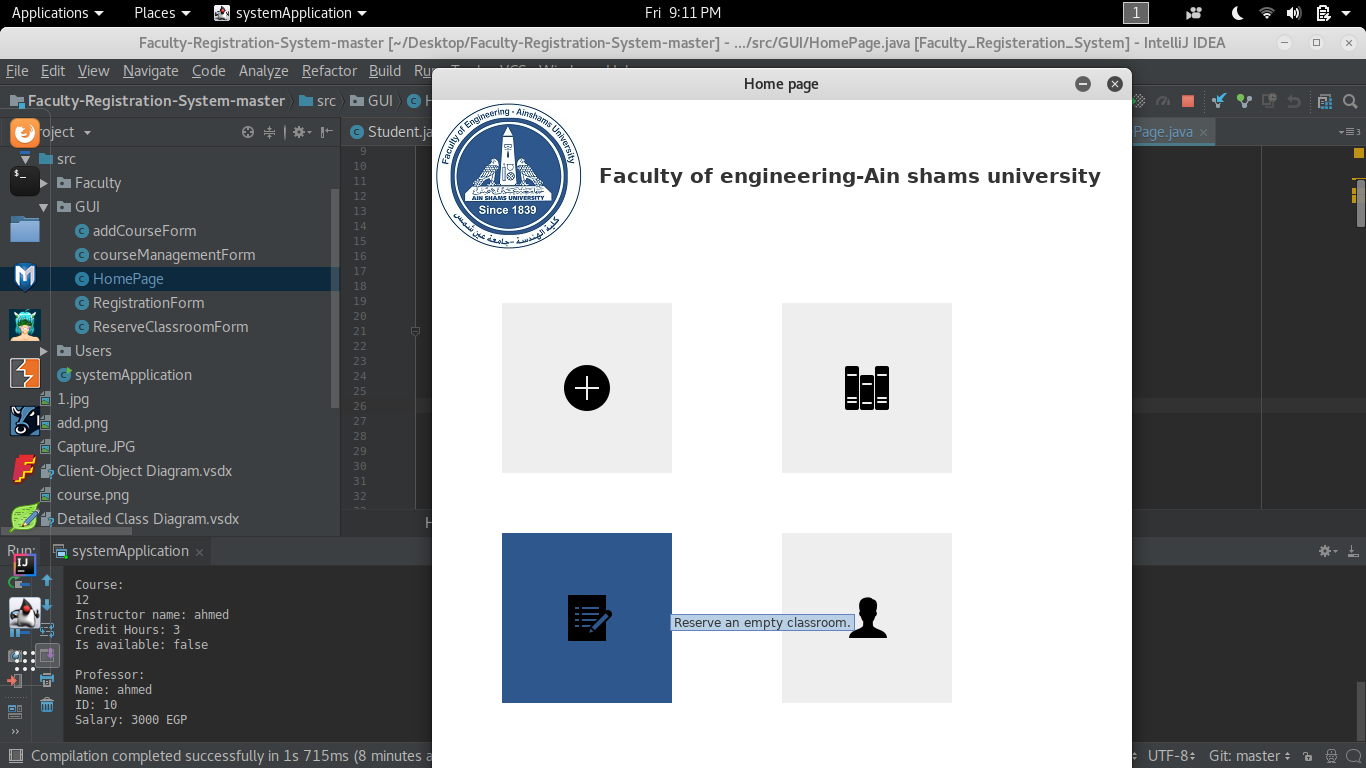


Figure 7: Add and drop 1



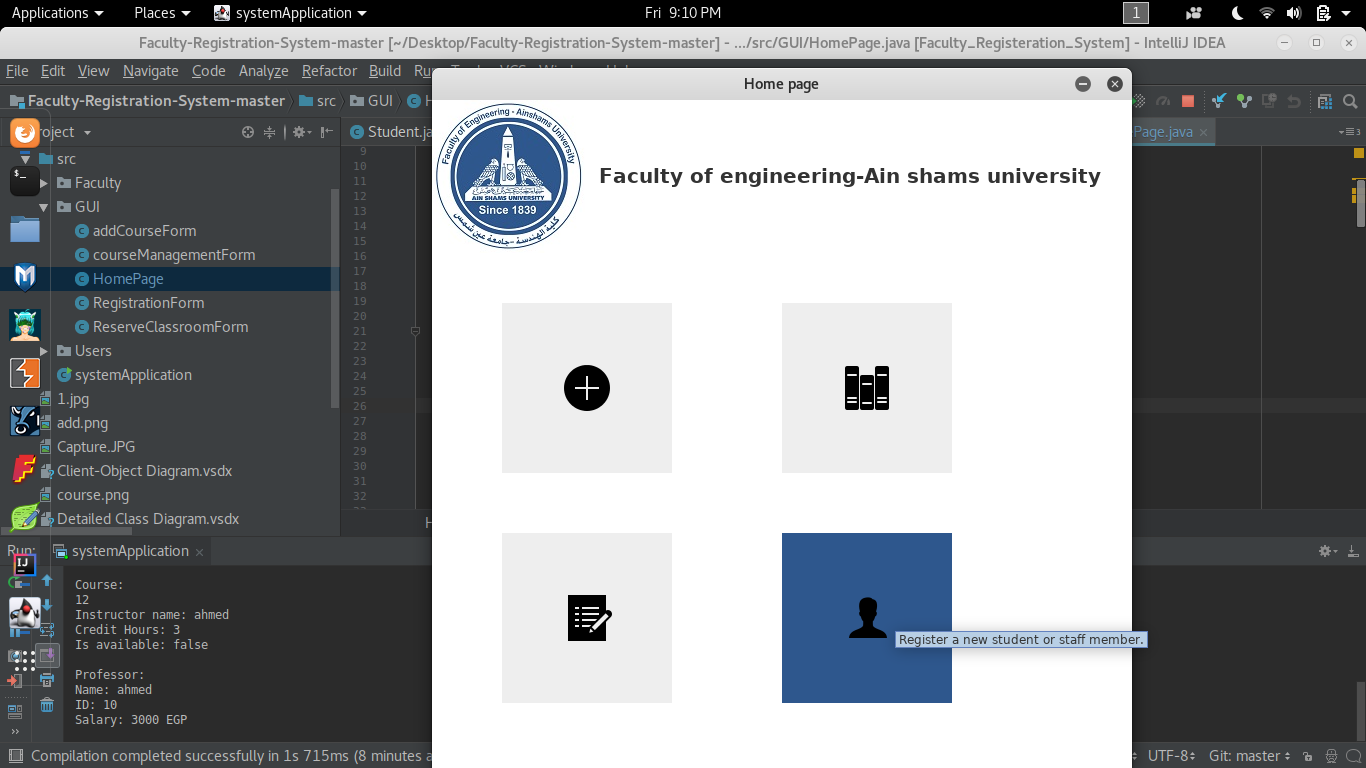
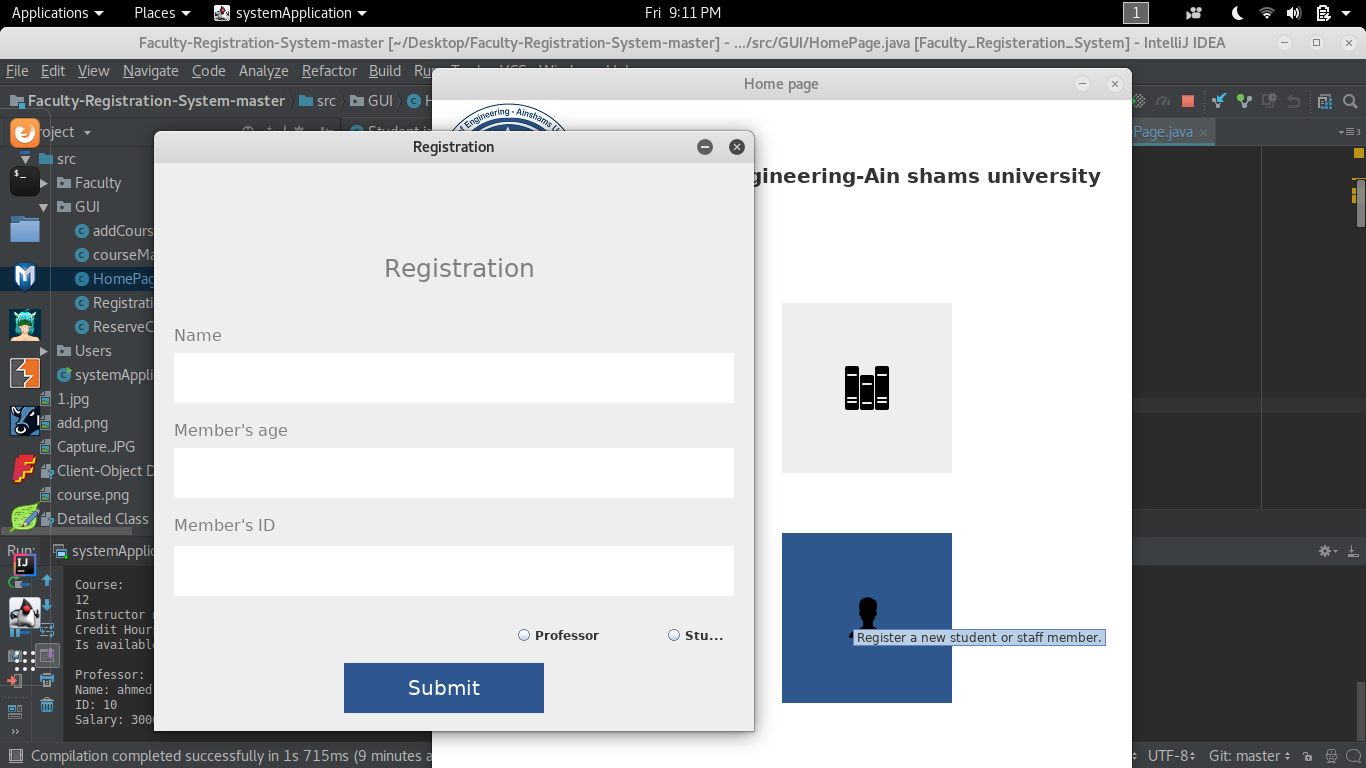


Figure 9.1: Register a new member form

Figure 9 : Register a new member1

Figure 8: Reserve a new classroom 1

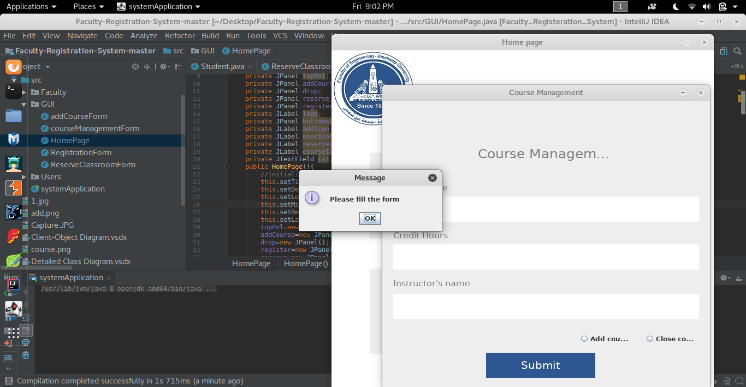


Figure 6.1: Course management Form. 1