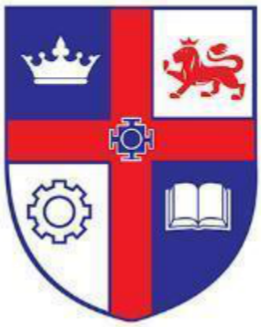
LONDON METROPOLITAN UNIVERSITY



islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CS4001NI Programming

COURSEWORK-2

Assessment Weightage & Type

30% Individual Coursework

Semester and Year

Spring 2021

Student Name: Sujen Shrestha

Group: N4

London Met ID: 20049250

College ID: NP01NT4S210105

Assignment Due Date: 20th August, 2021

Assignment Submission Date: 20th August, 2021

I confirm that I understand my coursework needs to be submitted online via Google classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submission will be treated as non-submission and a mark of zero will be awarded.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Introduction to project content

Java is a class-based, object-oriented programming language that can be developed and executed for any device platforms. It is a fast, secure and reliable platform for program development. Therefore, it is commonly used for creating programs and developing applications in desktop computers, gaming consoles, mobile phones, etc. (Guru99, 2021)

BlueJ is an Integrated Development Environment (IDE) for Java programming language, created mainly for educational purpose, but also useful for small-scale program development. It was developed in Monash University particularly to help in learning the object-oriented programming. (N K, 2021)

This is the second coursework of programming module. The coursework was completed using various tools like Bluej, Microsoft Word, Command Prompt and Draw.io. The purpose of this coursework is to create a “INGCollege” class which has various variables, methods, objects and functions. This class is created for developing a graphical user interface for a program that stores details of Course, Academic Course and Non-Academic Course. The class consists of various elements like menu bar, labels, text fields, buttons, etc. which enables the user to perform various actions such as add, register or remove a specific type of course.

The class consists of two array lists which stores the data given by the user and operates by using the variables and methods from the previously created classes (i.e., “Course”, “AcademicCourse” and “NonAcademicCourse).

The program consists of various structures of constructors, accessors, mutators, conditional statements and various other functions for performing actions on specific attributes. The “INGCollege” class contains a main method which can access and manipulate other methods and their functions from all the other classes of within the project.

2. Class Diagram

A class diagram is a static diagram which represents the static view of an application. Class diagrams are used for visualising, describing, documenting and constructing executable codes for a software application. It describes the attributes and operations of a class and also the constraints applied on the system. The class diagrams are commonly used in the designing of object-oriented systems since they are the only UML diagrams, that can be mapped directly with object-oriented languages. A class diagram displays a collection of classes, associations collaborations, interfaces and constraints. It is also known as a structural diagram. (Tutorialspoint, 2021)

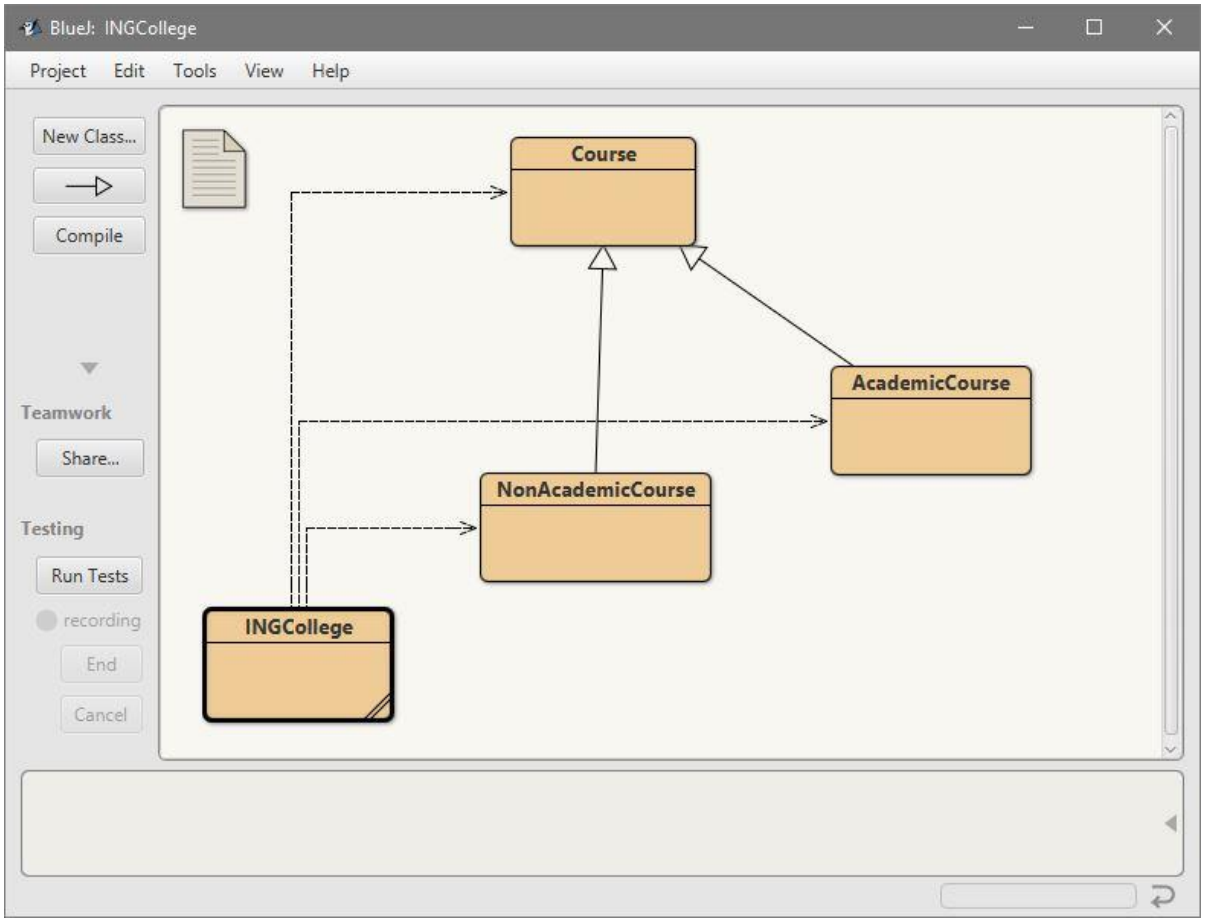


Figure 1: Class diagram of the classes of INGCollege project

2.1 INGCollege

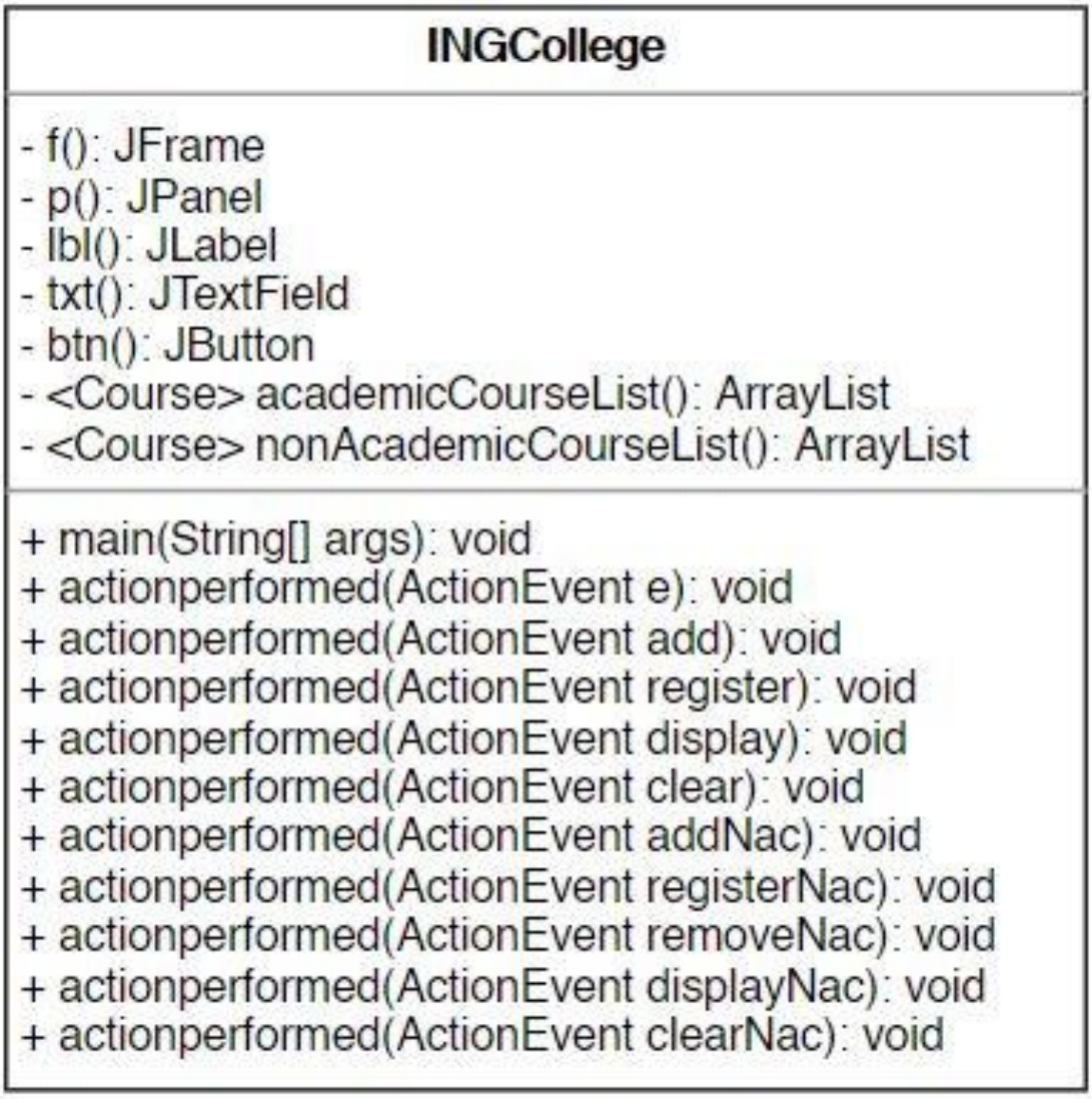


Figure 2: INGCollge Class Diagram

3. Pseudocode

A pseudocode is an unofficial way of coding description which does not need any programming language syntax or semantics. It is used for developing an outline of a program to understand the methods used in it. It is not an actual programming language so it cannot be compiled. It only summarizes the programs methods. (The Economic Times, 2021)

3.1 INGCollege class pseudocode

```
IMPORT packages in the program
CREATE class INGCollege
    DEFINE main method
        CREATE frame
        CREATE panels
        CREATE menu
        CREATE labels
        CREATE textfields
        CREATE buttons
    DEFINE actionPerformed method for academic button
        SET the value of academic panel visibility to true
        SET the value of non academic panel visibility to false
    DEFINE actionPerformed method for nonacademic button
        SET the value of academic panel visibility to false
        SET the value of non-academic panel visibility to true
    CREATE arraylist for AcademicCourse
    DEFINE actionPerformed method for academic add button
        GET all the textfields data
        IF any of the required textfield is empty
            SHOW dialog box with suitable message
        END IF
        ELSE
```

GET data from textfields and initialize and store them in respective variables
TRY
CONVERT duration and noa to number
END TRY
CATCH
SHOW dialog box with suitable message
END CATCH
CREATE variable id and store data from textfield of courseID
FOR
EXTRACT objects of arraylist and create and store them in object of Course
IF courseID in Course object equals id
SHOW dialog box with suitable message
END IF
END FOR
CREATE object for AcademicCourse and store the obtained data
ADD object to arraylist of AcademicCourse
END ELSE
DEFINE actionPerformed method for academic register button
IF any of the required textfield is empty
SHOW dialog box with suitable message
END IF
ELSE
GET data from textfields initialize and store them in respective variables
FOR
ITERATE the index values of the objects in arraylist
IF courseID in arraylist of AcademicCourse equals user's input

```
                                    EXTRACT objects from arraylist and downcast them
                                    to AcademicCourse type object ac
                                    IF ac is not equal to getIsRegistered
                                        CALL the register method from
                                        AcademicCourse
                                        SHOW dialog box with suitable message
                                    END IF
                                    ELSE
                                        SHOW dialog box with suitable message
                                    END ELSE
                                END IF
                            END FOR
                        END ELSE
                DEFINE actionPerformed method for academic display button
                    IF any of the required textfield is empty
                        SHOW dialog box with suitable message
                    END IF
                    ELSE
                        SHOW dialog box with suitable message along with the extracted
                        values from the textfields
                    END ELSE
                DEFINE actionPerformed method for academic clear button
                    SET all the textfields to empty
                CREATE arraylist for NonAcademicCourse
                DEFINE actionPerformed method for non academic add button
                    GET all the textfields data
                    IF any of the required textfield is empty
                        SHOW dialog box with suitable message
                    END IF
                    ELSE
```

GET data from textfields and initialize and store them in respective variables
TRY
CONVERT duration to number
END TRY
CATCH
SHOW dialog box with suitable message
END CATCH
CREATE variable idNac and store data from textfield of courseID
FOR
EXTRACT objects of arraylist and create and store them in object of Course
IF courseID in Course object equals idNac
SHOW dialog box with suitable message
END IF
END FOR
CREATE object for NonAcademicCourse and store the obtained data
ADD object to arraylist of NonAcademicCourse
END ELSE
DEFINE actionPerformed method for non academic register button
IF any of the required textfield is empty
SHOW dialog box with suitable message
END IF
ELSE
GET data from textfields and initialize and store them in respective variables
FOR
ITERATE the index values of the objects in arraylist
IF courseID in arraylist of NonAcademicCourse equals user's input

EXTRACT objects from arraylist and downcast them to NonAcademicCourse nac type
IF nac is not equal to getIsRegistered
CALL the register method from NonAcademicCourse class
SHOW dialog box with suitable message
END IF
ELSE
SHOW dialog box with suitable message
END ELSE
END IF
END FOR
END ELSE
DEFINE actionPerformed method for academic remove button
IF any of the required textfield is empty
SHOW dialog box with suitable message
END IF
ELSE
GET data from textfields and initialize and store them in respective variables
FOR
ITERATE the index values of the objects in arraylist
IF courseID in arraylist of NonAcademicCourse equals user's input
EXTRACT objects from arraylist and downcast them to NonAcademicCourse nac type
IF nac is not equal to getIsRegistered
CALL the remove method from NonAcademicCourse class
SHOW dialog box with suitable message
END IF

```
                    ELSE
                        SHOW dialog box with suitable message
                    END ELSE
                END IF
            END FOR
        END ELSE
    DEFINE actionPerformed method for non academic display button
        IF any of the required textfield is empty
            SHOW dialog box with suitable message
        END IF
        ELSE
            SHOW dialog box with suitable message along with the extracted
            values from the textfields
        END ELSE
    DEFINE actionPerformed method for non academic clear button
        SET all the textfields to empty
```

4. Method Description

In this program various methods have been used. We have four classes which have used the following methods:

4.1 INGCollege Class

The methods used in INGCollege class are given below:

- Void main (String[] args)

  This is the main method which accesses and manipulates other methods and their functions from all the other classes of within the project.
- actionPerformed(ActionEvent e)

  This is an ActionListener for btnAcademic which executes when the button is clicked and sets the visibility of academic panel to true and non academic panel to false
- actionPerformed(ActionEvent e)

  This is an ActionListener for btnNonAcademic which executes when the button is clicked and sets the visibility of academic panel to false and non academic panel to true
- actionPerformed(ActionEvent add)

  This is an ActionListener for btnAdd which executes when the button is clicked. If the required textfields are empty it returns a message dialog box to notify the user to fill up all the fields else, it gets the data from textfields and stores it in their respective variables. Also, it checks if the id in AcademicCourse arraylist is equal to the new id, if they match then it returns a message dialog box to notify the user that the id already exists. Then it creates an object of AcademicCourse and adds it to its arraylist.
- actionPerformed(ActionEvent register)

  This is an ActionListener for btnRegister which executes when the button is clicked. If the required textfields are empty it returns a message dialog box to notify the user to fill up all the fields else, it gets the data from textfields and stores it in their respective variables. Then, it iterates through the index values arraylist of AcademicCourse. If the courseID in arraylist of AcademicCourse

equals the user's input then it extracts objects from the arraylist and downcasts them to its original AcademicCourse type in ac object. Then it checks if the ac is equal to getIsRegistered method, if not then it accesses the register method from AcademicCourse class and pops up a message dialog box showing "Academic Course has been registered". Otherwise, it pops up another message showing "The id does not exist".

- actionPerformed(ActionEvent display)

  This is an ActionListener for btnDisplay which executes when the button is clicked. If the required textfields are empty it returns a message dialog box to notify the user that the course has not been registered. Otherwise, it shows a dialog box with suitable message along with the extracted values from the textfields.

- actionPerformed(ActionEvent clear)

  This is an ActionListener for btnClear which executes when the button is clicked and sets all the textfields in academic course panel to empty.

- actionPerformed(ActionEvent addNac)

  This is an ActionListener for btnAddNac which executes when the button is clicked. If the required textfields are empty it returns a message dialog box to notify the user to fill up all the fields else, it gets the data from textfields and stores it in their respective variables. Also, it checks if the id in NonAcademicCourse arraylist is equal to the new id, if they match then it returns a message dialog box to notify the user that the id already exists. Then it creates an object of NonAcademicCourse and adds it to its arraylist.

- actionPerformed(ActionEvent registerNac)

  This is an ActionListener for btnRegisterNac which executes when the button is clicked. If the required textfields are empty it returns a message dialog box to notify the user to fill up all the fields else, it gets the data from textfields and stores it in their respective variables. Then, it iterates through the index values arraylist of NonAcademicCourse. If the courseID in arraylist of NonAcademicCourse equals the user's input then it extracts objects from the arraylist and downcasts them to its original NonAcademicCourse type in nac

object. Then it checks if the nac is equal to getIsRegistered method, if not then it accesses the register method from NonAcademicCourse class and pops up a message dialog box showing “Non Academic Course has been registered”. Otherwise, it pops up another message showing “The id does not exist”.

- actionPerformed(ActionEvent removeNac)

  This is an ActionListener for btnRemoveNac which executes when the button is clicked. If the required textfields are empty it returns a message dialog box to notify the user to fill up all the fields else, it gets the data from textfields and stores it in their respective variables. Then, it iterates through the index values arraylist of NonAcademicCourse. If the courseID in arraylist of NonAcademicCourse equals the user’s input then it extracts objects from the arraylist and downcasts them to its original NonAcademicCourse type in nac object. Then it checks if the nac is equal to getIsRemoved method, if not then it accesses the remove method from NonAcademicCourse class and pops up a message dialog box showing “Non Academic Course has been removed”. Otherwise, it pops up another message showing “The id does not exist”.

- actionPerformed(ActionEvent displayNac)

  This is an ActionListener for btnDisplayNac which executes when the button is clicked. If the required textfields are empty it returns a message dialog box to notify the user that the course has not been registered. Otherwise, it iterates through the index values arraylist of NonAcademicCourse. then it extracts objects from the arraylist and downcasts them to its original NonAcademicCourse type in nac object. Then it checks if the nac is equal to getIsRemoved method, if not then it shows a dialog box with suitable message along with the extracted values from the textfields.

- actionPerformed(ActionEvent clearNac)

  This is an ActionListener for btnClear which executes when the button is clicked and sets all the textfields in academic course panel to empty.

5. Testing Part

Test 1

Test No:	1
Objective:	To compile and run the program using command prompt
Action:	>> The file path should be navigated and compiled using command prompt. >> The program should be opened using command prompt.
Expected Result:	The program should compile and run without any error.
Actual Result:	The program was compiled and run without any error.
Conclusion:	The test is successful.

Table 1: To compile and run the program using command prompt

Output:

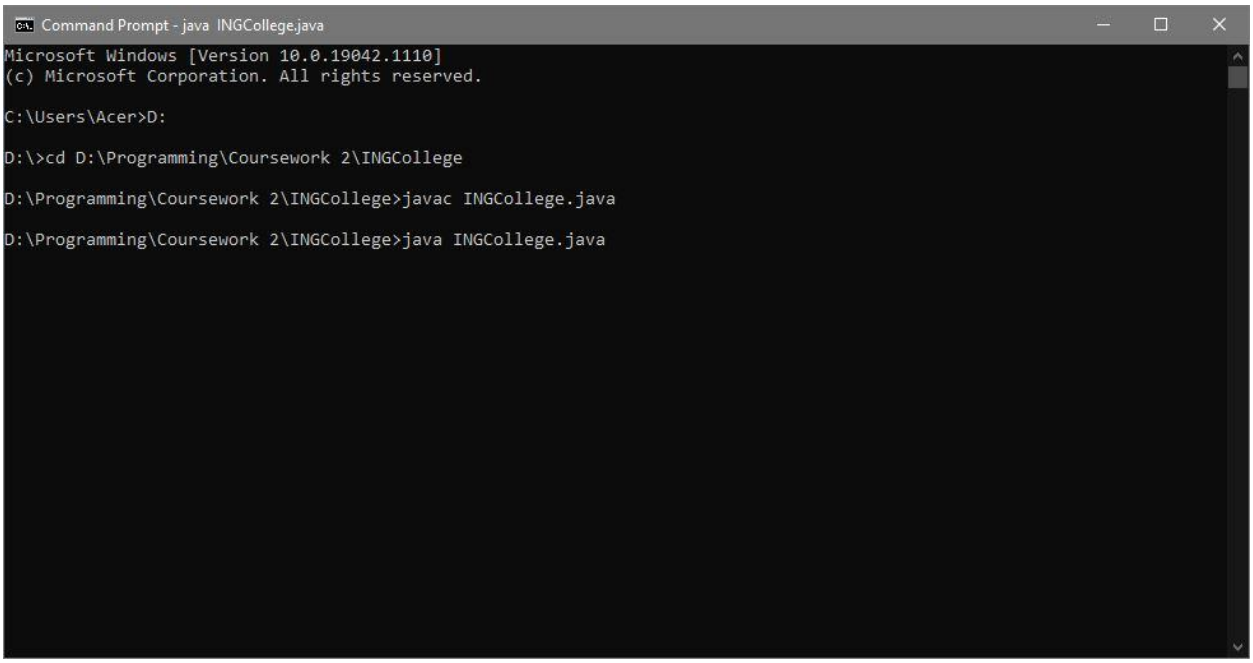


Figure 3: Compiling and running the program from command prompt

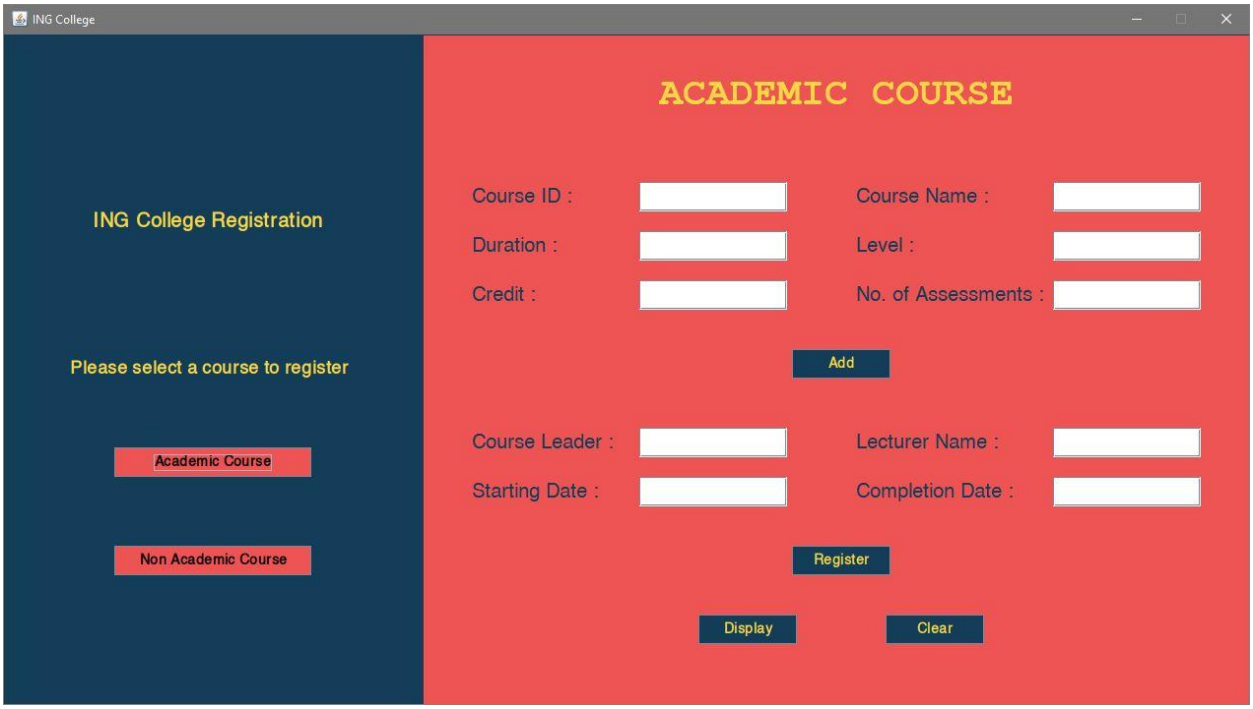


Figure 4: Output of program after running from command prompt

Test 2

Test No:	2
Objective:	To click on the Add button of Academic Course when text fields are empty, re-click it after entering the values in the text fields and again click it after entering the same values.
Action:	>> The Add button of Academic Course is clicked without any values entered. >> The Add button is clicked after the following values are input: Course ID = 001 Course Name = Programming Duration = 1 Level = 4 Credit = 30 No. of Assessments = 3 >> The Add button is clicked with the same values as input.
Expected Result:	Firstly, the it should give an alert message. Then it should add the academic course. Finally, it should display an alert message.
Actual Result:	The required output was achieved for all cases.
Conclusion:	The test is successful.

Table 2: To test the functionality of Add button of Academic Course under various conditions

Output results:

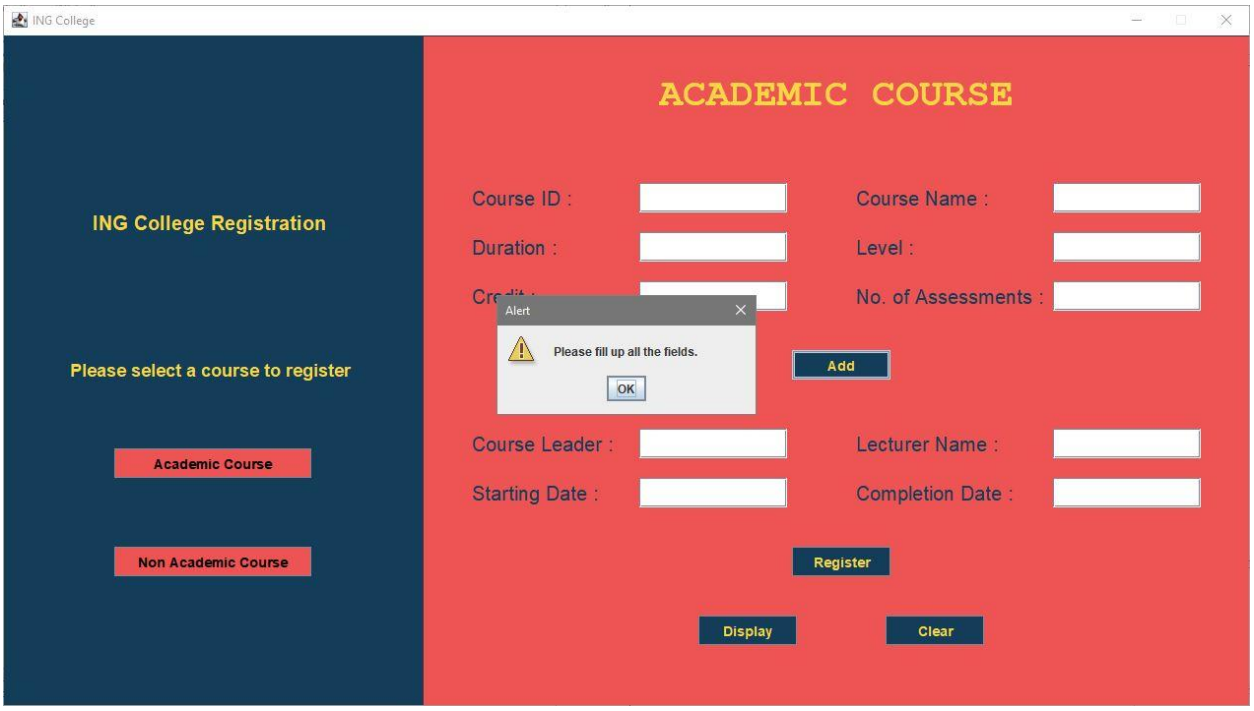


Figure 5: Add button of Academic Course clicked when fields are empty

Figure 6: Add button of Academic Course clicked after entering data in the required fields

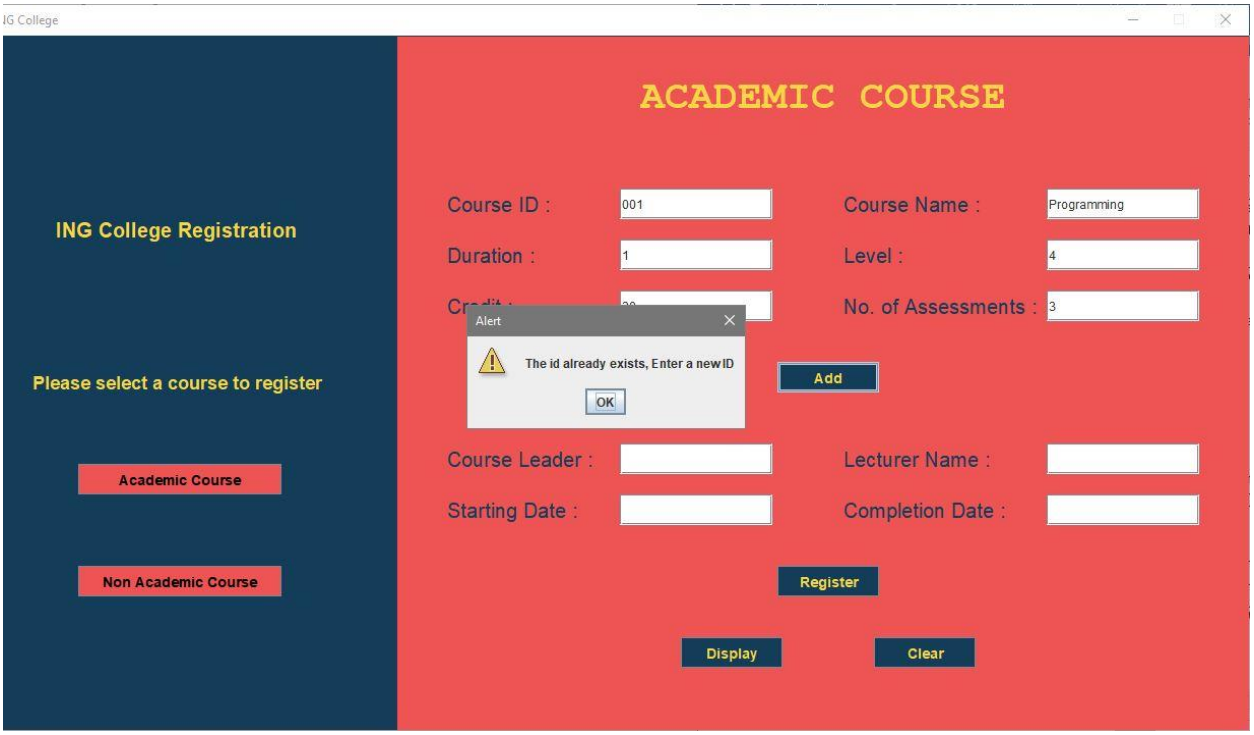


Figure 7: Add button of Academic Course clicked after the course has already been added

Test 3

Test No:	3
Objective:	To click the Register button of Academic Course when text fields are empty, re-click it after entering values and again click on it with the same data.
Action:	>> The Register button of Academic Course is clicked without any values entered. >> The Register button is clicked after the following values are input: Course Leader = Dhurba Sen Lecturer Name = Binay Adhikari Starting Date = 2021-03-08 Completion Date = 2022-03-01. >> The Register button is clicked with the same values as input.
Expected Result:	Firstly, the it should give an alert message. Then it should register the academic course. Finally, it should display an alert message.
Actual Result:	The required output was achieved for all cases.
Conclusion:	The test is successful.

Table 3: To test the functionality of Register button of Academic Course under various conditions

Output results:

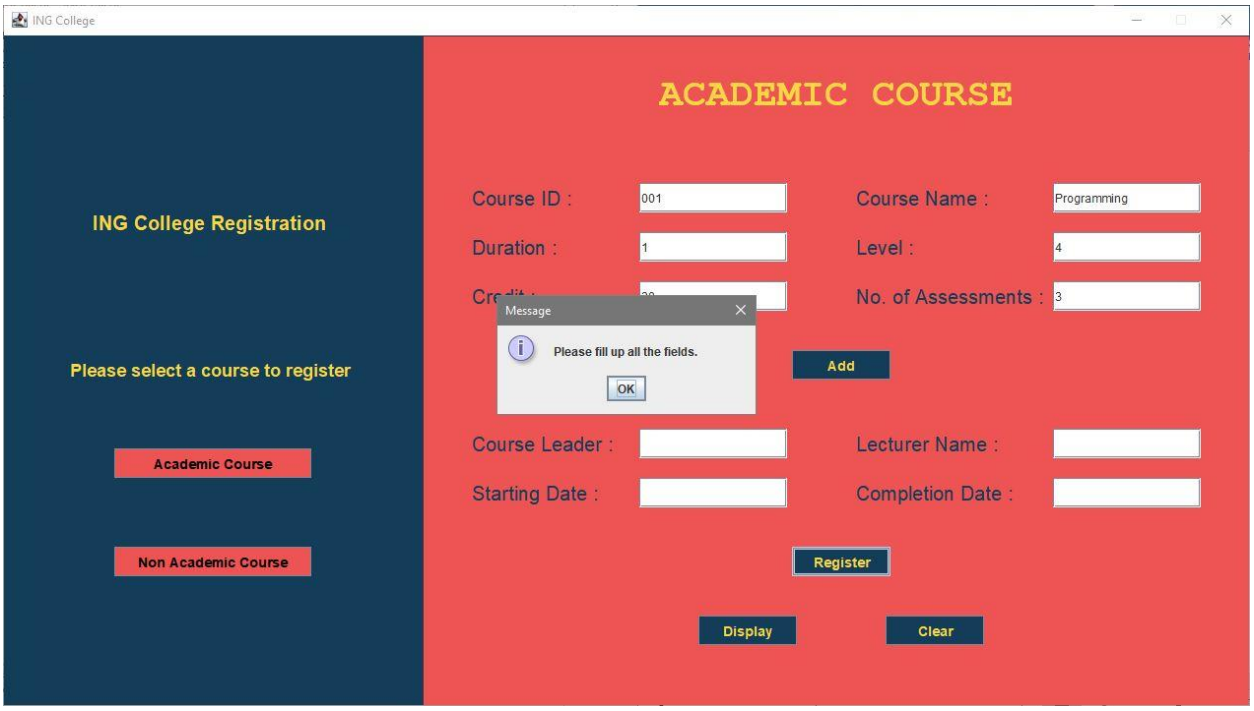


Figure 8: Register button of Academic Course clicked with empty fields

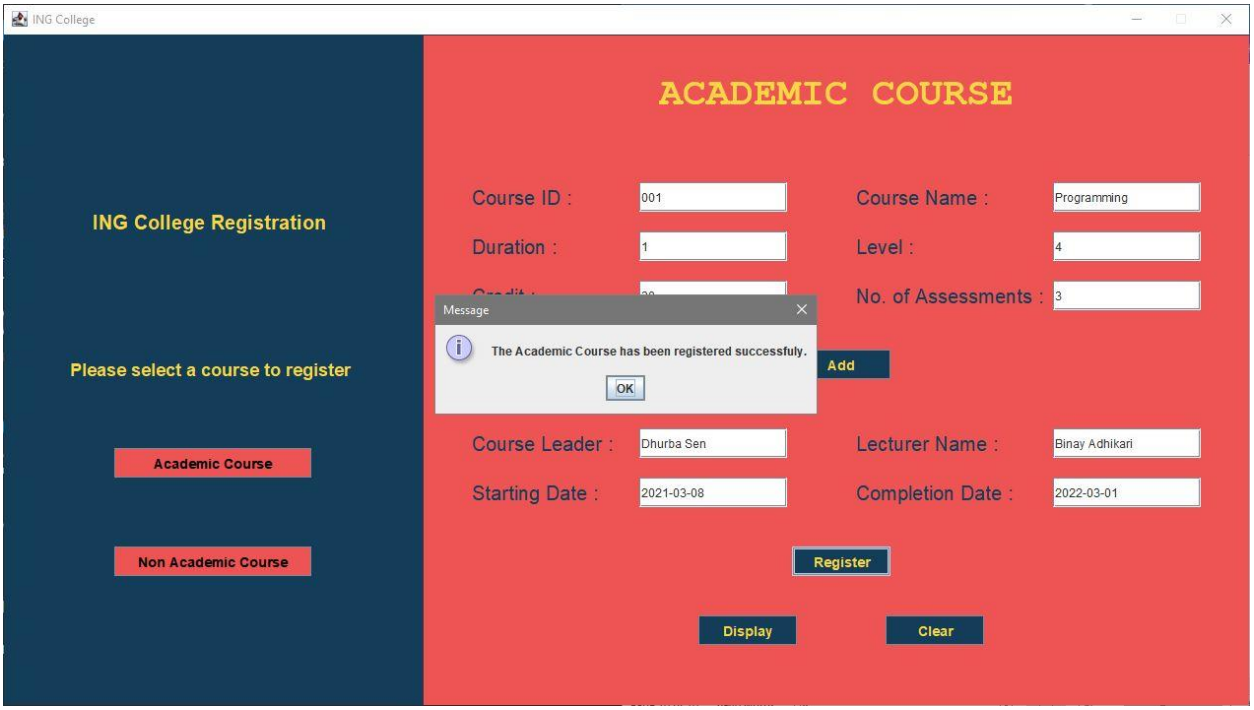


Figure 9: Register button of Academic Course clicked after entering data in the required fields

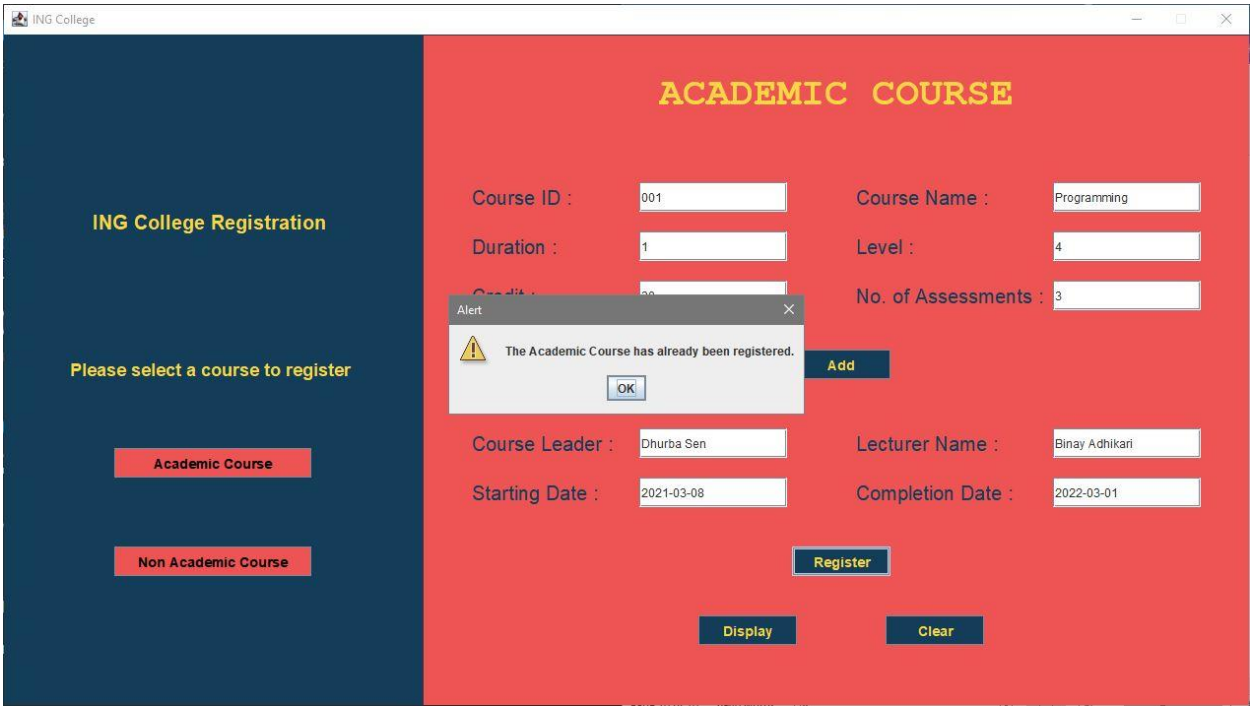


Figure 10: Register button of Academic Course clicked after the course has already been registered

Test 4

Test No:	4
Objective:	To click on the Add button of Non Academic Course when text fields are empty, re-click it after entering the values in the text fields and again click it after entering the same values.
Action:	>> The Add button of Non Academic Course is clicked without any values entered. >> The Add button is clicked after the following values are input: Course ID = 1001 Course Name = Music Duration = 1 Prerequisites = Knowledge of Music Notations >> The Add button is clicked with the same values as input.
Expected Result:	Firstly, the it should give an alert message. Then it should add the non academic course. Finally, it should display an alert message.
Actual Result:	The required output was achieved for all cases.
Conclusion:	The test is successful.

Table 4: To test the functionality of Add button of Non Academic Course under various conditions

Output results:

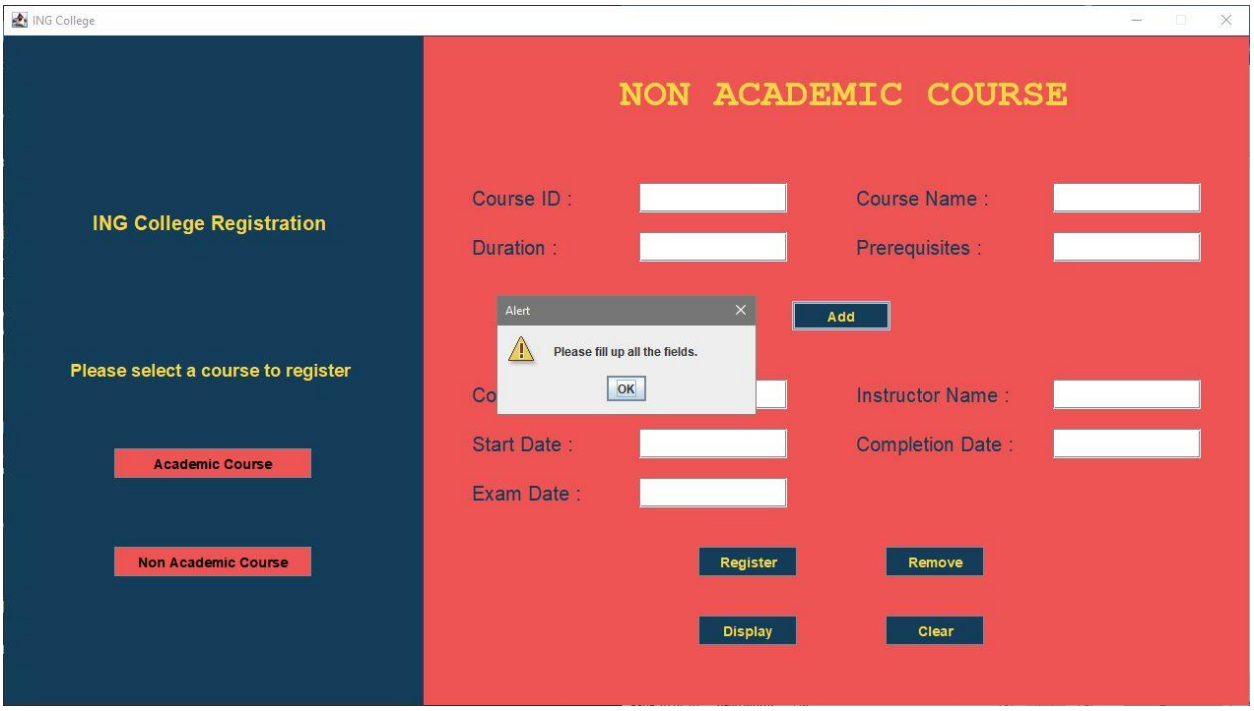


Figure 11:Add button of Non Academic Course clicked when fields are empty

Figure 12: Add button of Non Academic Course clicked after entering data in the required fields

Figure 13:Add button of Non Academic Course clicked after the course has already been added

Test 5

Test No:	5
Objective:	To click the Register button of Non Academic Course when text fields are empty, re-click it after entering values and again click on it with the same data.
Action:	>> The Register button of Non Academic Course is clicked without any values entered. >> The Register button is clicked after the following values are entered: Course Leader = Raju Lama Lecturer Name = Pramod Kharel Starting Date = 2021-05-15 Completion Date = 2021-04-12 Exam Date = 2021-04-20 >> The Register button is clicked with the same values as input.
Expected Result:	Firstly, the it should give an alert message. Then it should register the academic course. Finally, it should display an alert message.
Actual Result:	The required output was achieved for all cases.
Conclusion:	The test is successful.

Table 5: To test the functionality of Register button of Non Academic Course under various conditions

Output results:

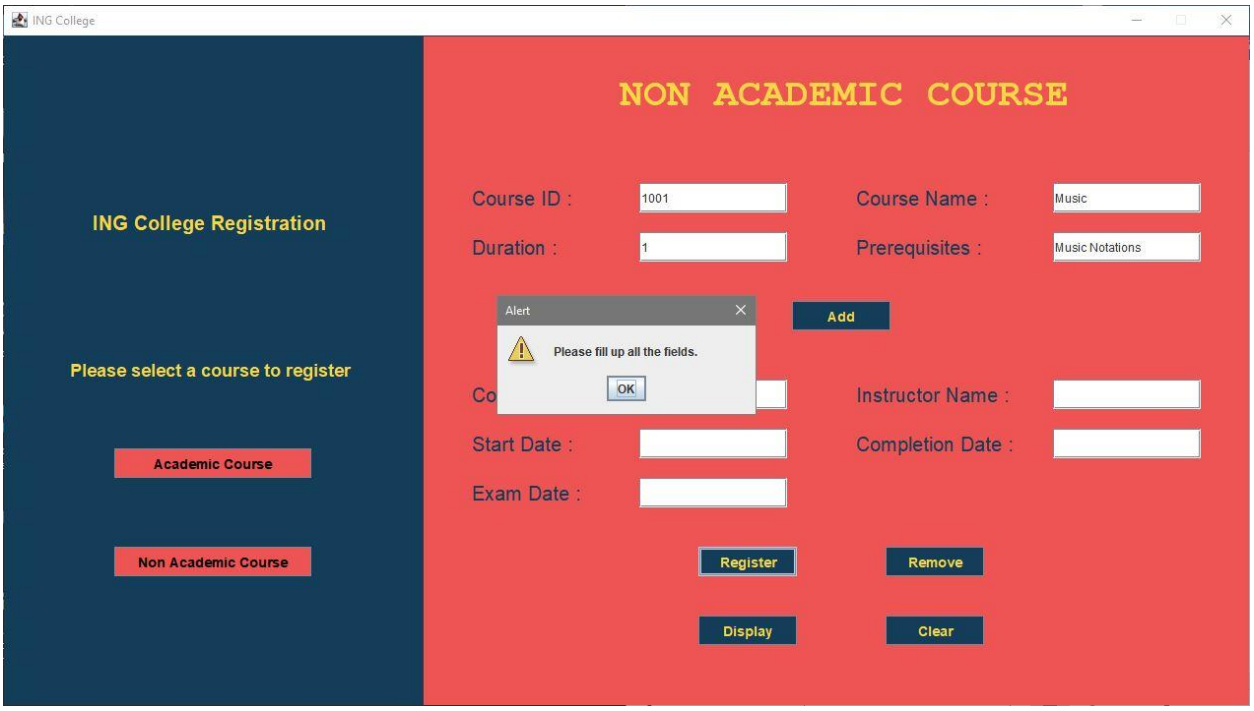


Figure 14: Register button of Non Academic Course clicked with empty fields

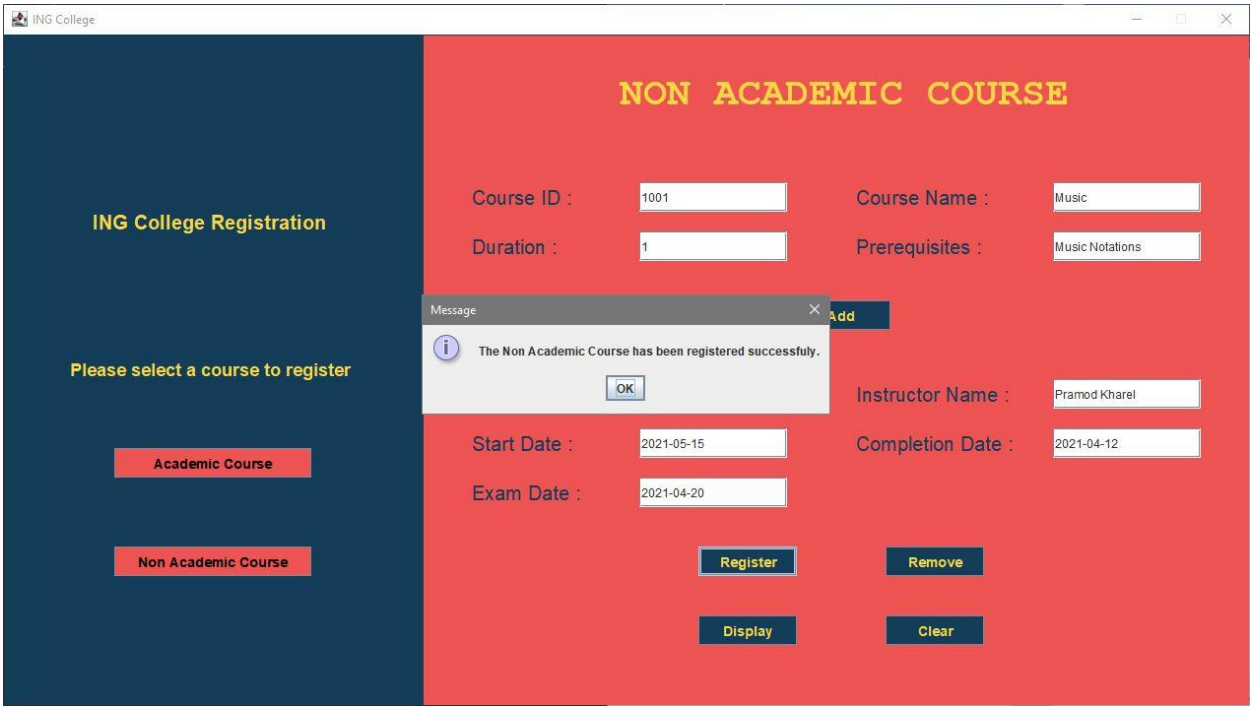


Figure 15: Register button of Academic Course clicked after entering data in the required fields

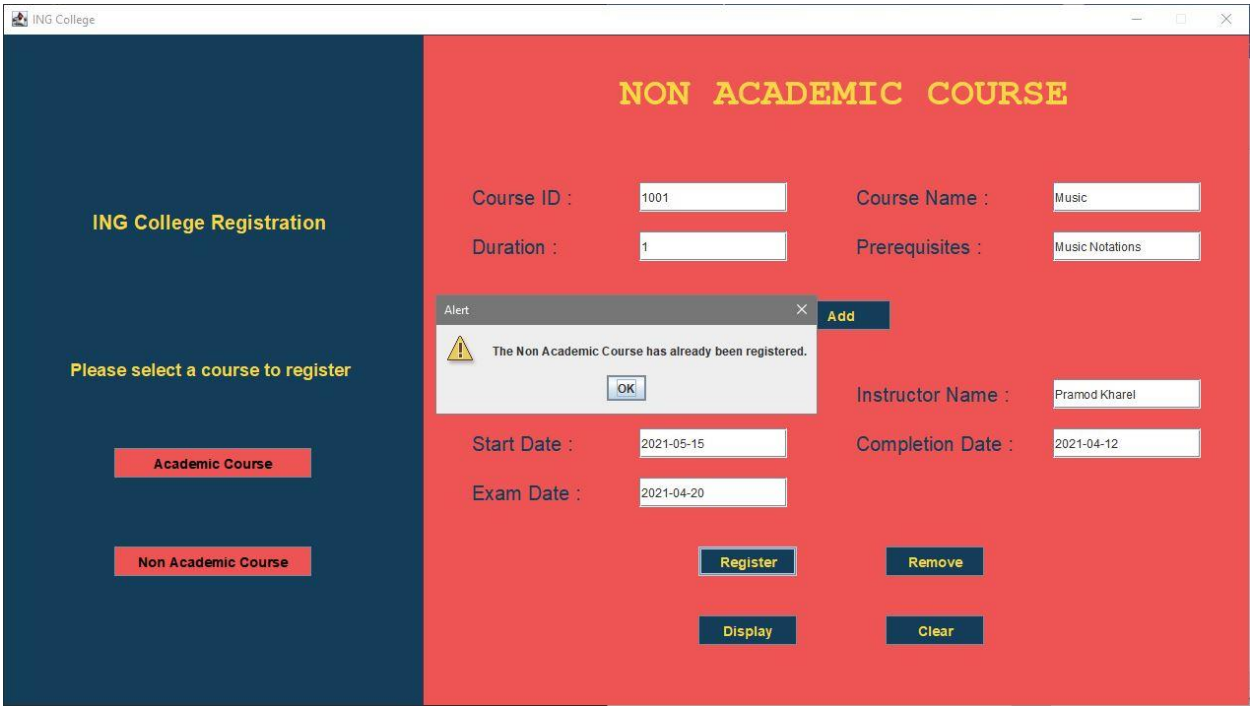


Figure 16: Register button of Non Academic Course clicked after the course has already been registered

Test 6

Test No:	6
Objective:	To click the Remove button when text fields are empty, re-click it after entering values and again click on it with the same data.
Action:	>> The Register button of Academic Course is clicked without any values entered. >> The Register button is clicked after the following values are input: Course Leader = Dhurba Sen Lecturer Name = Binay Adhikari Starting Date = 2021-03-08 Completion Date = 2022-03-01. >> The Register button is clicked with the same values as input.
Expected Result:	Firstly, the it should give an alert message. Then it should register the academic course. Finally, it should display an alert message.
Actual Result:	The required output was achieved for all cases.
Conclusion:	The test is successful.

Table 6: To test the functionality of Remove button of Non Academic Course under various conditions

Output:

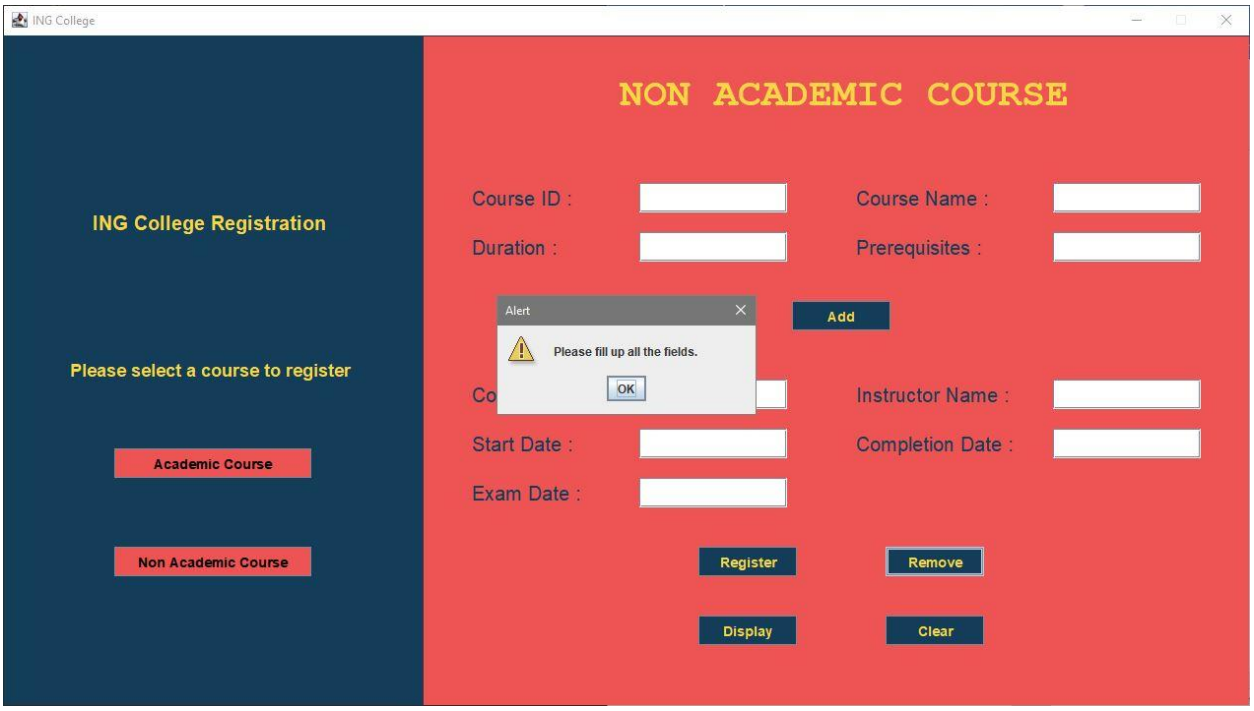


Figure 17: Remove button of Non Academic Course clicked when fields are empty

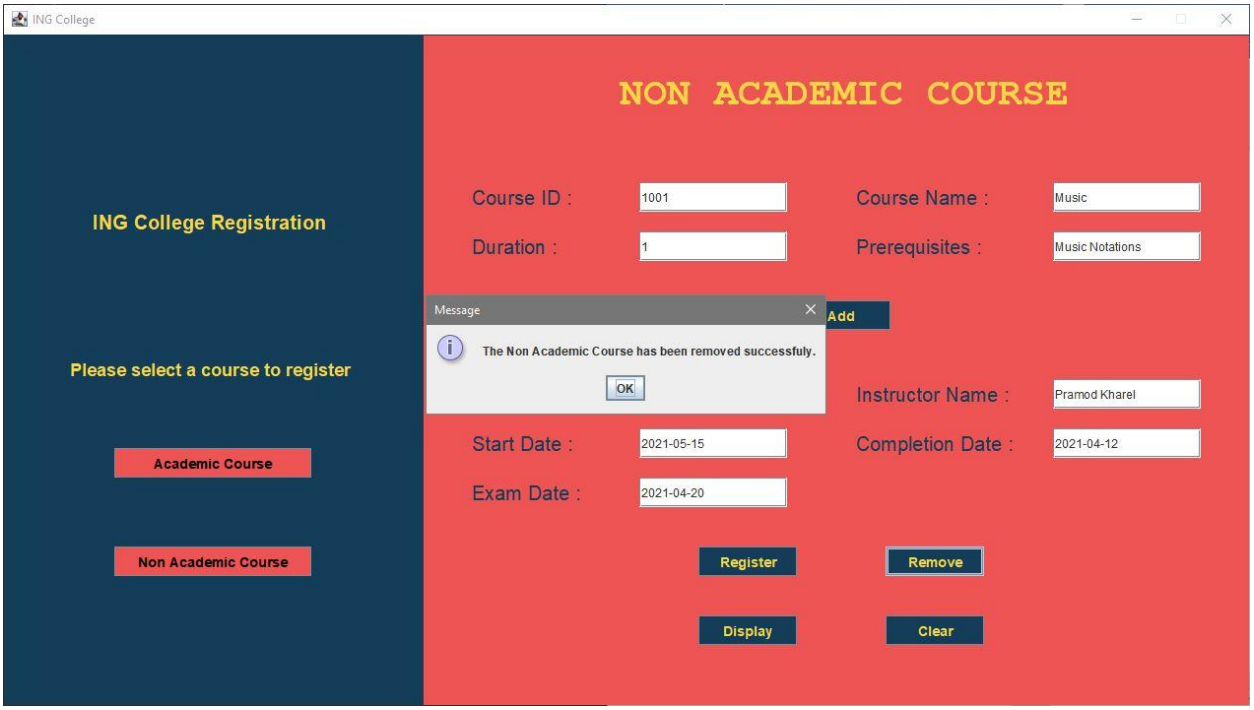


Figure 18: Remove button of Non Academic Course clicked after entering data in the required fields

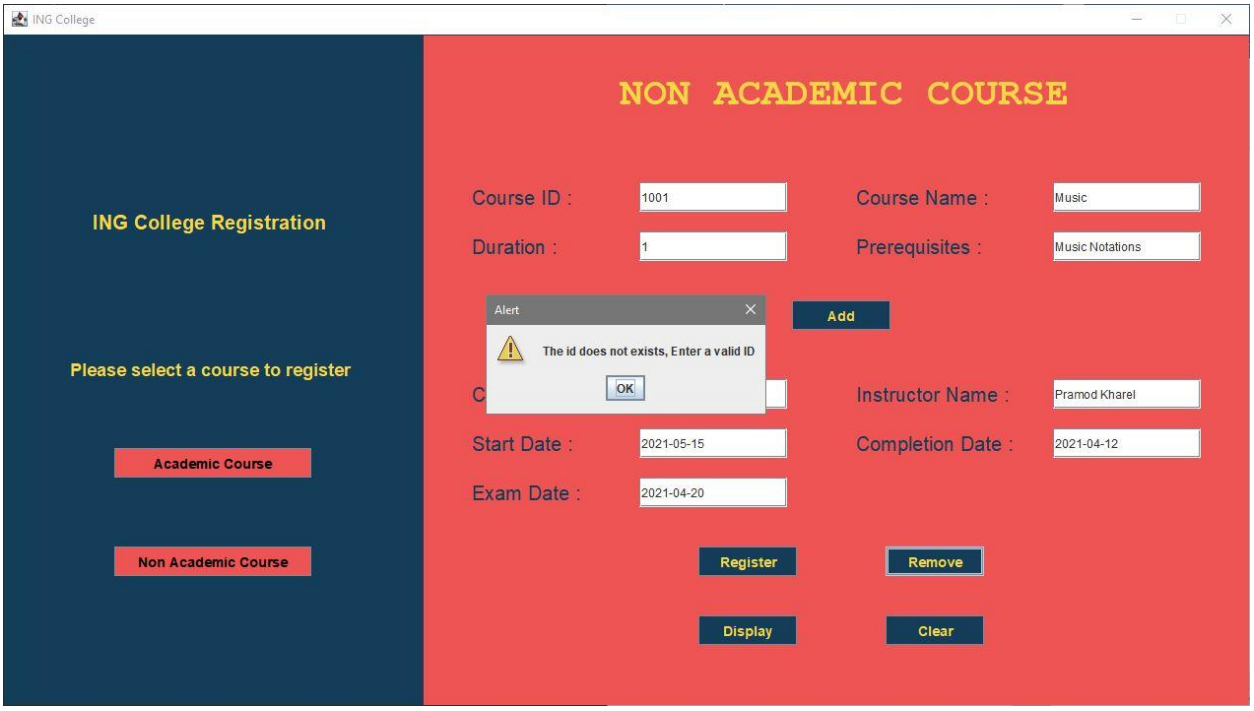


Figure 19: Remove button of Non Academic Course clicked after the course has already been removed

6. Error Detection

There were various errors that were detected during coursework while coding in Bluej. These problems were solved by observing the nature of the problem in detail. The various types of errors that arose while coding are as follows:

6.1 Syntax Error

A syntax error is an error that occurs when the arrangement of structure in the source code of a program is incorrect. The computer programs must follow the correct structure or syntax in order to compile successfully. Any portion of the code which does not conform to the syntax of a programming language produces a syntax error. (Christensson, 2012)

One of the syntax errors detected while compiling the program is given below:

A curly bracket was missing for else condition inside the method.

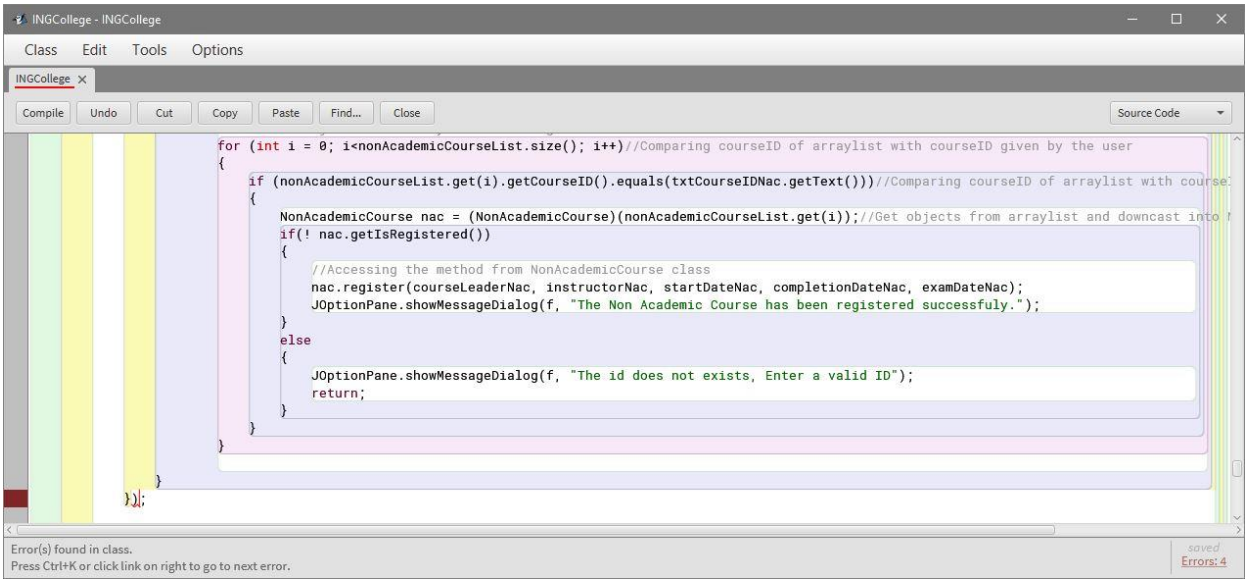


Figure 20: Syntax error in the program

The error was solved by closing the curly bracket in the required position.

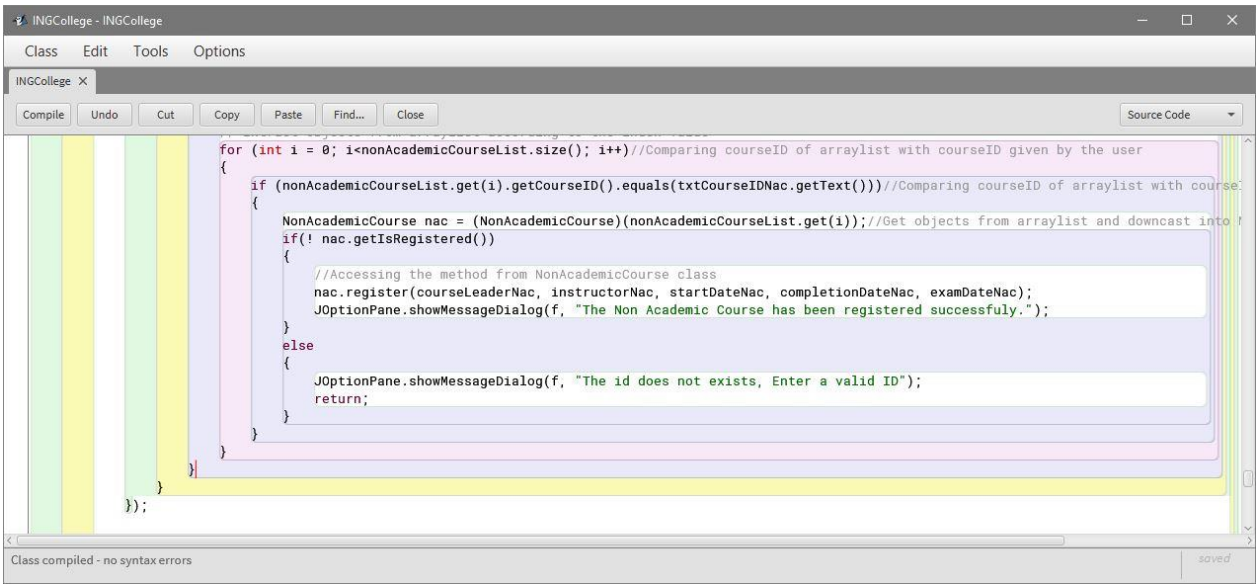


Figure 21: Correction of syntax error

6.2 Semantic Error

A semantic error is the error that occurs when wrong variable or operator is used, or when an operation is done in a wrong order. These types of errors are detected at the time of compilation. These types of errors are grammatically or syntactically correct so it is a little more difficult to find out than the syntax error. (Javatpoint, 2021)

One of the semantic errors detected during the compilation of the program is given below:

Else block was on the outer if condition so the message dialog box was not displaying in the program.

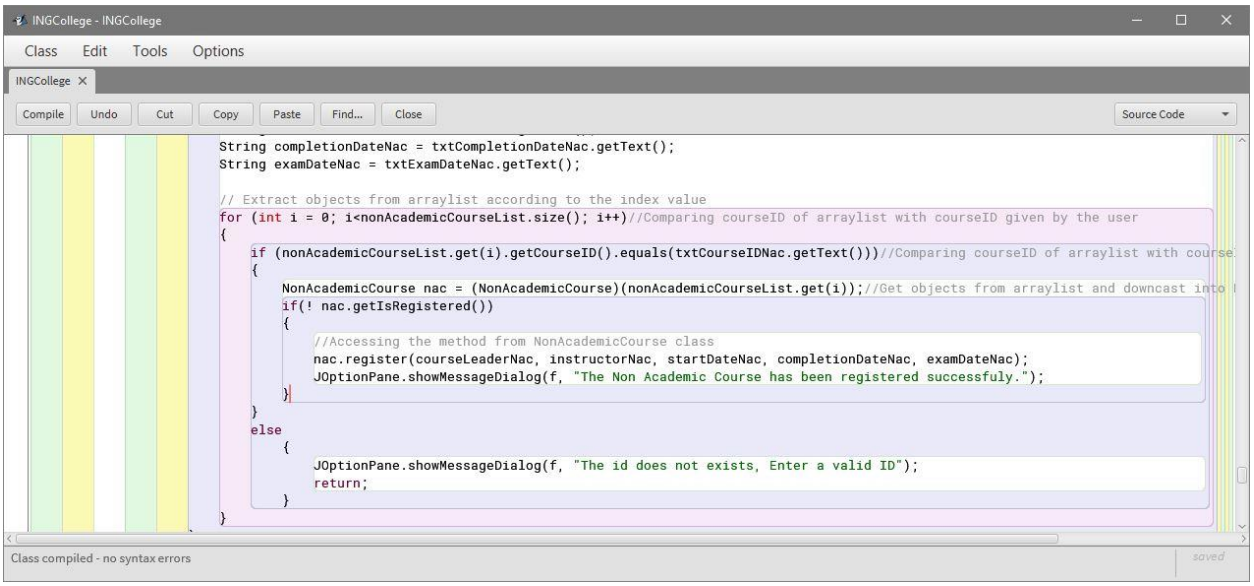


Figure 22: Semantic error in the program

The error was solved by placing the else block in the inner if condition.

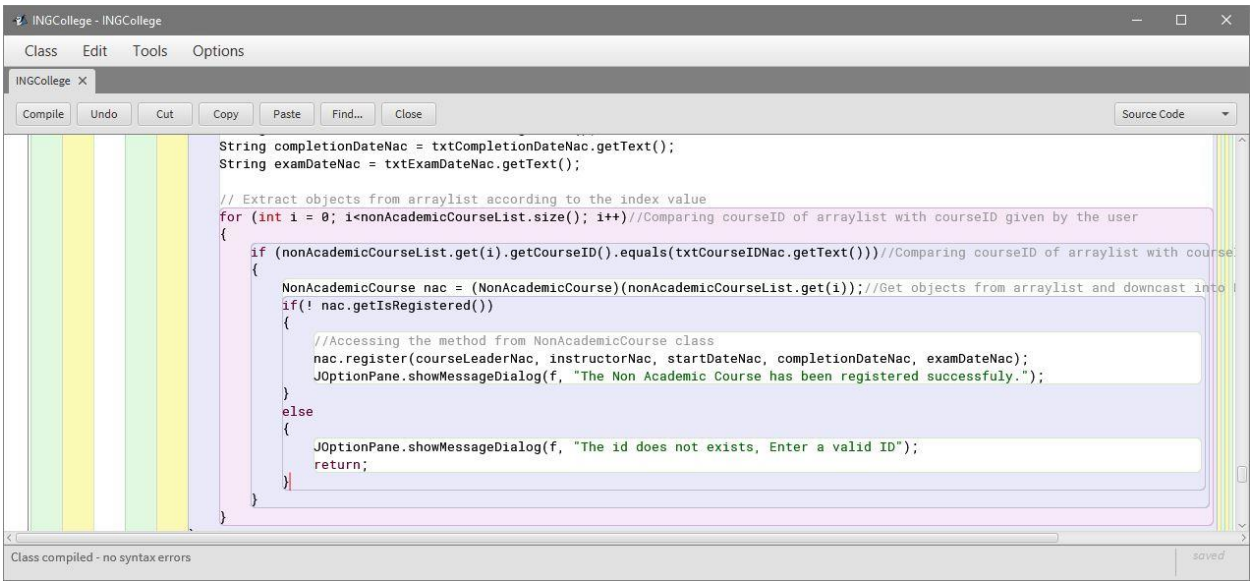


Figure 23: Correction of semantic error

6.3 Logic Error

A logic or logical error is the error that occurs when a mistake in the programs source code causes incorrect or unexpected result. This type of error is the most difficult to find out than other types of errors as all the syntax and semantics are correct. This type of error generally occurs when a different operator is used, a typo is made or when a programmer misunderstands the required output to be something else. (Christensson, 2012)

One of the logical errors detected after the compilation of the program is given below:

Incorrect function is used in the given method which caused the method to not function as required.

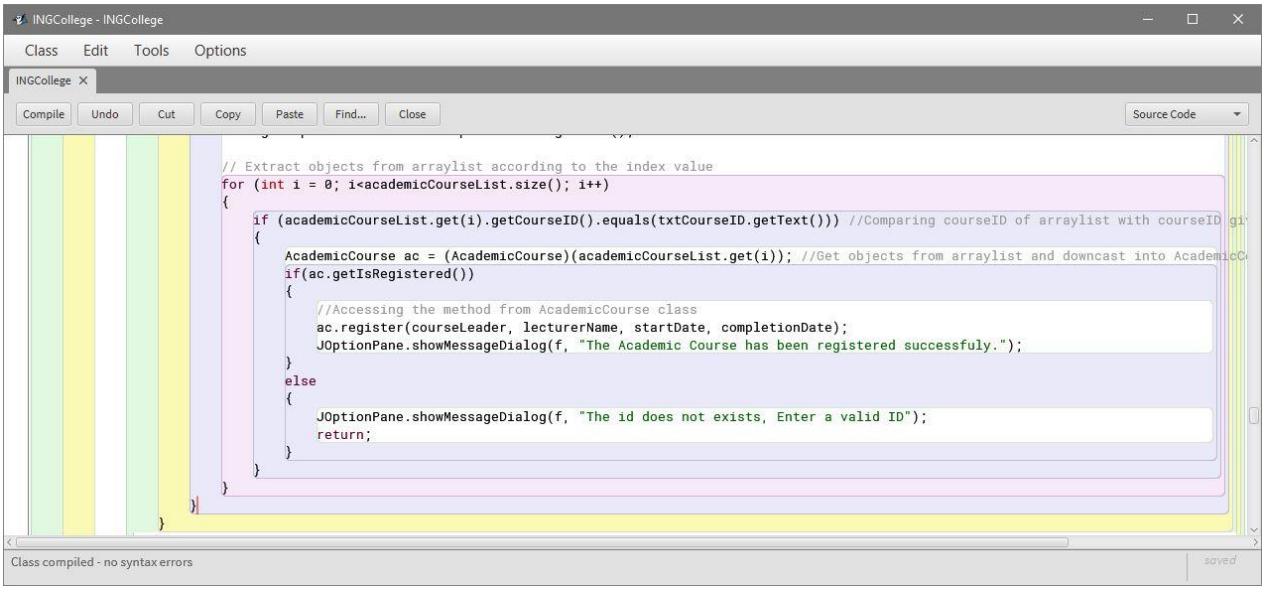


Figure 24: Logic error in the program

The error was solved by using the not operator in the required conditional statement.

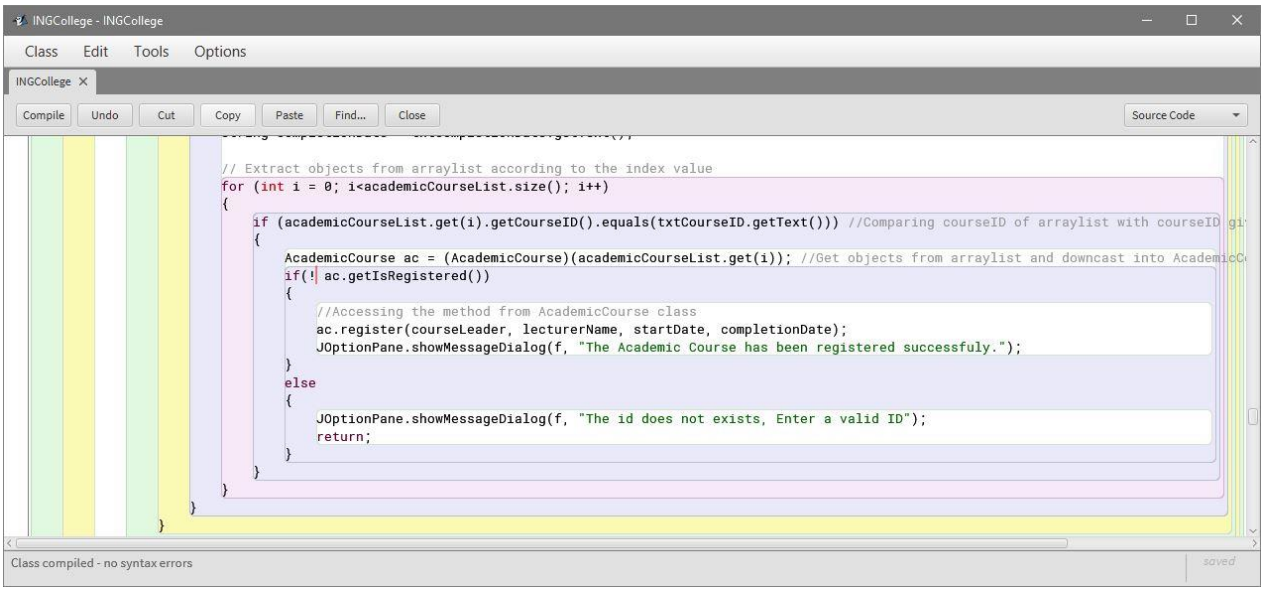


Figure 25: Correction of logic error

7. Conclusion

This is the second coursework for programming module. In this project, I had to develop a GUI for and add certain methods to make the buttons functional as required. I was already familiar with a lot of the terms involved in this project because of my involvement with java in the previous semester. Because of that, I was able to develop most part of the code smoothly. However, there were various new methods and functions that had to be implemented to create the required program and make it fully functional As per the requirement of this coursework I had to implement various logic and functions which was not present in the previous coursework.

I learnt about most of them from various websites, books and the lecture class. This helped me gain some perspective about how the specific keywords and functions work in this programming language. Many times, I could not grasp a particular idea or concept. So, I asked my lecturers about those topics. They were very supportive and consistently guided me when I was confused in various aspects of the course. Even while developing this coursework, I made various errors in the coding methods and the program was not executing as required.

A lot of it I learnt through trial and error and online research. There were specific activities to implement in this project which I could not find online and got stuck in them for a long time. In those times I contacted my lecturers through email messages, they articulated the nature of the problem and guided me in a way that was very easy for me to understand. In this way, I completed this project by taking references from various online sites, inquiring about particular issues with my lecturers and putting an effort to understand and implement various techniques required in the coursework.

References

Christensson, P., 2012. *Logic Error Definition.* [Online]
Available at: https://techterms.com/definition/logic_error
[Accessed 20 May 2021].
Christensson, P., 2012. *Syntax Error Definition.* [Online]
Available at: https://techterms.com/definition/syntax_error
[Accessed 20 May 2021].
Guru99, 2021. *Guru99.* [Online]
Available at: https://www.guru99.com/java-platform.html
[Accessed 17 May 2021].
Javatpoint, 2021. *Semantic Error.* [Online]
Available at: https://www.javatpoint.com/semantic-error
[Accessed 20 May 2021].
N K, R., 2021. *Java Tutorial by N K Raju.* [Online]
Available at: https://sites.google.com/site/javatutorialbynkraju/1-introduction/5-what-is-bluej
[Accessed 17 May 2021].
The Economic Times, 2021. *Software-Development.* [Online]
Available at: https://economictimes.indiatimes.com/definition/pseudocode
[Accessed 17 May 2021].
Tutorialspoint, 2021. *UML - Class Diagram.* [Online]
Available at: https://www.tutorialspoint.com/uml/uml_class_diagram.htm
[Accessed 17 May 2021].

Appendix

INGCourse Class

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;

/**
 * The INGCollege class is created for developing a GUI that stores the details of Course , AcademicCourse and NonAcademicCourse classes.
It contains a main method which accesses the functions and methods of every other classes within the project.

 * @author (Sujen Shrestha)
 * @Group N4
 * @ID (20049250)
 */
public class INGCollege
{
    public static void main (String[] args)
    {
        //Top level Container storing UI elements
        JFrame f = new JFrame ();
        f.setTitle("ING College");
        f.setBounds(40,5,1280,720);
        f.setResizable(false);
        f.setDefaultCloseOperation(f.EXIT_ON_CLOSE);

        //Jpanel for storing other panels
```

```
JPanel p = new JPanel ();
p.setBounds(0,0,1265,658);
p.setBorder(BorderFactory.createLineBorder(Color.decode("#143D59")));
p.setBackground(Color.decode("#143D59"));
p.setLayout(null);
f.add(p);

//Jpanel for sidebar
JPanel pHome = new JPanel ();
pHome.setBounds(0,0,425,658);
pHome.setBorder(BorderFactory.createLineBorder(Color.decode("#143D59")));
pHome.setBackground(Color.decode("#143D59"));
pHome.setLayout(null);
p.add(pHome);

//Jlabel for sidebar
JLabel lblWelcome = new JLabel ("ING College Registration");
lblWelcome.setBounds(90,150,450,80);
lblWelcome.setFont(new Font("helvetica", Font.BOLD, 20));
lblWelcome.setForeground(Color.decode("#FAD744"));
pHome.add(lblWelcome);

JLabel lblMessage = new JLabel ("Please select a course to register");
lblMessage.setBounds(67,300,450,80);
lblMessage.setFont(new Font("helvetica", Font.BOLD, 18));
lblMessage.setForeground(Color.decode("#FAD744"));
pHome.add(lblMessage);

//JButton for sidebar
JButton btnAcademic = new JButton("Academic Course");
btnAcademic.setBounds(112,420,200,30);
```

```
btnAcademic.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnAcademic.setBackground(Color.decode("#EF5455"));
btnAcademic.setForeground(Color.decode("#000000"));
pHome.add(btnAcademic);

JButton btnNonAcademic = new JButton("Non Academic Course");
btnNonAcademic.setBounds(112,520,200,30);
btnNonAcademic.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnNonAcademic.setBackground(Color.decode("#EF5455"));
btnNonAcademic.setForeground(Color.decode("#000000"));
pHome.add(btnNonAcademic);

//JPanel for Academic Course
JPanel pAc = new JPanel ();
pAc.setBounds(425,0,840,690);
pAc.setBorder(BorderFactory.createLineBorder(Color.decode("#2B3252")));
pAc.setBackground(Color.decode("#EF5455"));
pAc.setLayout(null);
p.add(pAc);

//JLabels for academic course
JLabel lblTitle = new JLabel ("ACADEMIC COURSE");
lblTitle.setFont(new Font("Courier New", Font.BOLD, 40));
lblTitle.setForeground(Color.decode("#FAD744"));
lblTitle.setBounds(240,40,400,40);
pAc.add(lblTitle);

JLabel lblCourseID = new JLabel ("Course ID : ");
lblCourseID.setBounds(50,150,200,30);
lblCourseID.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCourseID.setForeground(Color.decode("#143D59"));
```

```
pAc.add(lblCourseID);

JLabel lblCourseName = new JLabel ("Course Name : ");
lblCourseName.setBounds(440,150,200,30);
lblCourseName.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCourseName.setForeground(Color.decode("#143D59"));
pAc.add(lblCourseName);

JLabel lblDuration = new JLabel ("Duration : ");
lblDuration.setBounds(50,200,200,30);
lblDuration.setFont(new Font("helvetica", Font.PLAIN, 20));
lblDuration.setForeground(Color.decode("#143D59"));
pAc.add(lblDuration);

JLabel lblLevel = new JLabel ("Level : ");
lblLevel.setBounds(440,200,200,30);
lblLevel.setFont(new Font("helvetica", Font.PLAIN, 20));
lblLevel.setForeground(Color.decode("#143D59"));
pAc.add(lblLevel);

JLabel lblCredit = new JLabel ("Credit : ");
lblCredit.setBounds(50,250,200,30);
lblCredit.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCredit.setForeground(Color.decode("#143D59"));
pAc.add(lblCredit);

JLabel lblAssessments = new JLabel ("No. of Assessments : ");
lblAssessments.setBounds(440,250,200,30);
lblAssessments.setFont(new Font("helvetica", Font.PLAIN, 20));
lblAssessments.setForeground(Color.decode("#143D59"));
pAc.add(lblAssessments);
```

```
JLabel lblCourseLeader = new JLabel ("Course Leader : ");
lblCourseLeader.setBounds(50,400,200,30);
lblCourseLeader.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCourseLeader.setForeground(Color.decode("#143D59"));
pAc.add(lblCourseLeader);

JLabel lblLecturer = new JLabel ("Lecturer Name : ");
lblLecturer.setBounds(440,400,200,30);
lblLecturer.setFont(new Font("helvetica", Font.PLAIN, 20));
lblLecturer.setForeground(Color.decode("#143D59"));
pAc.add(lblLecturer);

JLabel lblStartDate = new JLabel ("Starting Date : ");
lblStartDate.setBounds(50,450,200,30);
lblStartDate.setFont(new Font("helvetica", Font.PLAIN, 20));
lblStartDate.setForeground(Color.decode("#143D59"));
pAc.add(lblStartDate);

JLabel lblCompletionDate = new JLabel ("Completion Date : ");
lblCompletionDate.setBounds(440,450,200,30);
lblCompletionDate.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCompletionDate.setForeground(Color.decode("#143D59"));
pAc.add(lblCompletionDate);

//JTextFields for academic course
JTextField txtCourseID = new JTextField();
txtCourseID.setBounds(220,150,150,30);
pAc.add(txtCourseID);

JTextField txtCourseName = new JTextField();
```

```
txtCourseName.setBounds(640,150,150,30);
pAc.add(txtCourseName);

JTextField txtDuration = new JTextField();
txtDuration.setBounds(220,200,150,30);
pAc.add(txtDuration);

JTextField txtLevel = new JTextField();
txtLevel.setBounds(640,200,150,30);
pAc.add(txtLevel);

JTextField txtCredit = new JTextField();
txtCredit.setBounds(220,250,150,30);
pAc.add(txtCredit);

JTextField txtAssessments = new JTextField();
txtAssessments.setBounds(640,250,150,30);
pAc.add(txtAssessments);

JTextField txtCourseLeader = new JTextField();
txtCourseLeader.setBounds(220,400,150,30);
pAc.add(txtCourseLeader);

JTextField txtLecturer = new JTextField();
txtLecturer.setBounds(640,400,150,30);
pAc.add(txtLecturer);

JTextField txtStartDate = new JTextField();
txtStartDate.setBounds(220,450,150,30);
pAc.add(txtStartDate);
```

```
JTextField txtCompletionDate = new JTextField();
txtCompletionDate.setBounds(640,450,150,30);
pAc.add(txtCompletionDate);

//JButtons for academic course

JButton btnAdd = new JButton("Add");
btnAdd.setBounds(375,320,100,30);
btnAdd.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnAdd.setBackground(Color.decode("#143D59"));
btnAdd.setForeground(Color.decode("#FAD744"));
pAc.add(btnAdd);

JButton btnRegister = new JButton("Register");
btnRegister.setBounds(375,520,100,30);
btnRegister.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnRegister.setBackground(Color.decode("#143D59"));
btnRegister.setForeground(Color.decode("#FAD744"));
pAc.add(btnRegister);

JButton btnDisplay = new JButton("Display");
btnDisplay.setBounds(280,590,100,30);
btnDisplay.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnDisplay.setBackground(Color.decode("#143D59"));
btnDisplay.setForeground(Color.decode("#FAD744"));
pAc.add(btnDisplay);

JButton btnClear = new JButton("Clear");
btnClear.setBounds(470,590,100,30);
btnClear.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnClear.setBackground(Color.decode("#143D59"));
```

```
btnClear.setForeground(Color.decode("#FAD744"));
pAc.add(btnClear);

pAc.setVisible(true);

//Panel for Non Academic Course
JPanel pNac = new JPanel ();
pNac.setBounds(425,0,840,690);
pNac.setBorder(BorderFactory.createLineBorder(Color.decode("#2B3252")));
pNac.setBackground(Color.decode("#EF5455"));
pNac.setLayout(null);
p.add(pNac);

//JLabels for non academic
JLabel lblTitleNac = new JLabel ("NON ACADEMIC COURSE");
lblTitleNac.setFont(new Font("Courier New", Font.BOLD, 40));
lblTitleNac.setForeground(Color.decode("#FAD744"));
lblTitleNac.setBounds(200,40,550,40);
pNac.add(lblTitleNac);

JLabel lblCourseIDNac = new JLabel ("Course ID : ");
lblCourseIDNac.setBounds(50,150,200,30);
lblCourseIDNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCourseIDNac.setForeground(Color.decode("#143D59"));
pNac.add(lblCourseIDNac);

JLabel lblCourseNameNac = new JLabel ("Course Name : ");
lblCourseNameNac.setBounds(440,150,200,30);
lblCourseNameNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCourseNameNac.setForeground(Color.decode("#143D59"));
pNac.add(lblCourseNameNac);
```

```
JLabel lblDurationNac = new JLabel ("Duration : ");
lblDurationNac.setBounds(50,200,200,30);
lblDurationNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblDurationNac.setForeground(Color.decode("#143D59"));
pNac.add(lblDurationNac);

JLabel lblPrerequisiteNac = new JLabel ("Prerequisites : ");
lblPrerequisiteNac.setBounds(440,200,200,30);
lblPrerequisiteNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblPrerequisiteNac.setForeground(Color.decode("#143D59"));
pNac.add(lblPrerequisiteNac);

JLabel lblCourseLeaderNac = new JLabel ("Course Leader : ");
lblCourseLeaderNac.setBounds(50,250,200,30);
lblCourseLeaderNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCourseLeaderNac.setForeground(Color.decode("#143D59"));
pNac.add(lblCourseLeaderNac);

JLabel lblInstructorNac = new JLabel ("Instructor Name : ");
lblInstructorNac.setBounds(440,250,200,30);
lblInstructorNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblInstructorNac.setForeground(Color.decode("#143D59"));
pNac.add(lblInstructorNac);

JLabel lblStartDateNac = new JLabel ("Start Date : ");
lblStartDateNac.setBounds(50,400,200,30);
lblStartDateNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblStartDateNac.setForeground(Color.decode("#143D59"));
pNac.add(lblStartDateNac);
```

```
JLabel lblCompletionDateNac = new JLabel ("Completion Date : ");
lblCompletionDateNac.setBounds(440,400,200,30);
lblCompletionDateNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblCompletionDateNac.setForeground(Color.decode("#143D59"));
pNac.add(lblCompletionDateNac);

JLabel lblExamDateNac = new JLabel ("Exam Date : ");
lblExamDateNac.setBounds(50,450,200,30);
lblExamDateNac.setFont(new Font("helvetica", Font.PLAIN, 20));
lblExamDateNac.setForeground(Color.decode("#143D59"));
pNac.add(lblExamDateNac);

//JTextFields for non academic
JTextField txtCourseIDNac = new JTextField();
txtCourseIDNac.setBounds(220,150,150,30);
pNac.add(txtCourseIDNac);

JTextField txtCourseNameNac = new JTextField();
txtCourseNameNac.setBounds(640,150,150,30);
pNac.add(txtCourseNameNac);

JTextField txtDurationNac = new JTextField();
txtDurationNac.setBounds(220,200,150,30);
pNac.add(txtDurationNac);

JTextField txtPrerequisiteNac = new JTextField();
txtPrerequisiteNac.setBounds(640,200,150,30);
pNac.add(txtPrerequisiteNac);

JTextField txtCourseLeaderNac = new JTextField();
txtCourseLeaderNac.setBounds(220,250,150,30);
```

```
pNac.add(txtCourseLeaderNac);

JTextField txtInstructorNac = new JTextField();
txtInstructorNac.setBounds(640,250,150,30);
pNac.add(txtInstructorNac);

JTextField txtStartDateNac = new JTextField();
txtStartDateNac.setBounds(220,400,150,30);
pNac.add(txtStartDateNac);

JTextField txtCompletionDateNac = new JTextField();
txtCompletionDateNac.setBounds(640,400,150,30);
pNac.add(txtCompletionDateNac);

JTextField txtExamDateNac = new JTextField();
txtExamDateNac.setBounds(220,450,150,30);
pNac.add(txtExamDateNac);

//JButtons for Non Academic Course

JButton btnAddNac = new JButton("Add");
btnAddNac.setBounds(375,320,100,30);
btnAddNac.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnAddNac.setBackground(Color.decode("#143D59"));
btnAddNac.setForeground(Color.decode("#FAD744"));
pNac.add(btnAddNac);

JButton btnRegisterNac = new JButton("Register");
btnRegisterNac.setBounds(280,520,100,30);
btnRegisterNac.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnRegisterNac.setBackground(Color.decode("#143D59"));
```

```
btnRegisterNac.setForeground(Color.decode("#FAD744"));
pNac.add(btnRegisterNac);

JButton btnRemoveNac = new JButton("Remove");
btnRemoveNac.setBounds(470,520,100,30);
btnRemoveNac.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnRemoveNac.setBackground(Color.decode("#143D59"));
btnRemoveNac.setForeground(Color.decode("#FAD744"));
pNac.add(btnRemoveNac);

JButton btnDisplayNac = new JButton("Display");
btnDisplayNac.setBounds(280,590,100,30);
btnDisplayNac.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnDisplayNac.setBackground(Color.decode("#143D59"));
btnDisplayNac.setForeground(Color.decode("#FAD744"));
pNac.add(btnDisplayNac);

JButton btnClearNac = new JButton("Clear");
btnClearNac.setBounds(470,590,100,30);
btnClearNac.setFont(new Font("HELVETICA", Font.BOLD, 14));
btnClearNac.setBackground(Color.decode("#143D59"));
btnClearNac.setForeground(Color.decode("#FAD744"));
pNac.add(btnClearNac);

pHome.setVisible(true);
pNac.setVisible(false);
p.setVisible(true);
f.setVisible(true);

//button actions
btnAcademic.addActionListener(new ActionListener()
```

```
        {
            public void actionPerformed(ActionEvent e)
            {
                pAc.setVisible(true);
                pNac.setVisible(false);
            }
        });

    btnNonAcademic.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                pAc.setVisible(false);
                pNac.setVisible(true);
            }
        });

    ArrayList <Course> academicCourseList = new ArrayList<Course>();

    btnAdd.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent add)
            {
                if (txtCourseID.getText().isEmpty() || txtCourseName.getText().isEmpty() ||
txtDuration.getText().isEmpty()
                || txtLevel.getText().isEmpty() || txtCredit.getText().isEmpty() ||
txtAssessments.getText().isEmpty())
                {
                    JOptionPane.showMessageDialog(f, "Please fill up all the
fields.","Alert",JOptionPane.WARNING_MESSAGE);
                    return;
```

```
            }
            else
            {
                String courseID = txtCourseID.getText();
                String courseName = txtCourseName.getText();
                String courseLeader = txtCourseLeader.getText();
                int duration = 0;
                try
                {
                    duration = Integer.parseInt(txtDuration.getText());
                }
                catch (NumberFormatException e)
                {
                    JOptionPane.showMessageDialog(f, "Please enter a valid number for
duration.","Alert",JOptionPane.WARNING_MESSAGE);
                    return;
                }
                String level = txtLevel.getText();
                String credit = txtCredit.getText();
                int assessments = 0;
                try
                {
                    assessments = Integer.parseInt(txtAssessments.getText());
                }
                catch (NumberFormatException e)
                {
                    JOptionPane.showMessageDialog(f, "Please enter a valid number for
No. of assessments.","Alert",JOptionPane.WARNING_MESSAGE);
                    return;
                }
```

```
                String id = txtCourseID.getText();
                for (Course c : academicCourseList)// Extract objects of arraylist and
create and store them in object of Course c
                {
                   if (c.getCourseID().equals(id))// Comparing courseID in Course object
with id
                   {
                      JOptionPane.showMessageDialog(f, "The id already exists, Enter a
new ID","Alert",JOptionPane.WARNING_MESSAGE);
                      return;
                   }
                }

                AcademicCourse obj = new AcademicCourse(courseID, courseName,
duration, level, credit, assessments);
                academicCourseList.add(obj);
             }
          }
       });

    btnRegister.addActionListener(new ActionListener()
       {
          public void actionPerformed(ActionEvent register)
          {
             if (txtCourseLeader.getText().isEmpty() || txtLecturer.getText().isEmpty() ||
             txtStartDate.getText().isEmpty() || txtCompletionDate.getText().isEmpty())
             {
                JOptionPane.showMessageDialog(f, "Please fill up all the fields.");
                return;
             }
             else
```

```
            {
                String courseLeader = txtCourseLeader.getText();
                String lecturerName = txtLecturer.getText();
                String startDate = txtStartDate.getText();
                String completionDate = txtCompletionDate.getText();

                // Extract objects from arraylist according to the index value
                for (int i = 0; i<academicCourseList.size(); i++)
                {
                    //Comparing courseID of arraylist with courseID given by the user
                    if
(academicCourseList.get(i).getCourseID().equals(txtCourseID.getText()))
                    {
                        //Get objects from arraylist and downcast into AcademicCourse
                        AcademicCourse ac =
(AcademicCourse)(academicCourseList.get(i));
                        if(! ac.getIsRegistered())
                        {
                            //Accessing the method from AcademicCourse class
                            ac.register(courseLeader, lecturerName, startDate,
completionDate);
                            JOptionPane.showMessageDialog(f, "The Academic Course has
been registered successfuly.");
                            return;
                        }
                        else
                        {
                            JOptionPane.showMessageDialog(f, "The Academic Course has
already been registered.","Alert",JOptionPane.WARNING_MESSAGE);
                            return;
                        }
```

```
                    }
                }
            }
        }
    });

    btnDisplay.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent display)
        {
            if (txtCourseID.getText().isEmpty() || txtCourseName.getText().isEmpty() ||
txtDuration.getText().isEmpty()
            || txtLevel.getText().isEmpty() || txtCredit.getText().isEmpty() ||
txtAssessments.getText().isEmpty())
            {
                JOptionPane.showMessageDialog(f,"The Academic Course has not
been registered.","Alert",JOptionPane.WARNING_MESSAGE);
            }
            else{
                JOptionPane.showMessageDialog(f, "ACADEMIC COURSE" + "\n\n" +
                    "The Course ID of the course is: " + txtCourseID.getText() + "\n" +
                    "The Course Name of the course is: " + txtCourseName.getText() +
"\n" +
                    "The Duration of the course is: " + txtDuration.getText() + "\n" +
                    "The Level of the coursee is: " + txtLevel.getText() + "\n" +
                    "The Credit present in the course is: "+ txtCredit.getText() + "\n" +
                    "The No. of Assessments in the course: " + txtAssessments.getText()
+ "\n" +
                    "The Course Leader of the course is: " + txtCourseLeader.getText() +
"\n" +
                    "The Lecturer of the course is: " + txtLecturer.getText()+"\n" +
```

```
                    "The Starting Date of course is: " + txtStartDate.getText() + "\n" +
                    "The Completion Date of the course is: " +
txtCompletionDate.getText());
                }
            }
        });

    btnClear.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent clear)
            {
                txtCourseID.setText("");
                txtCourseName.setText("");
                txtDuration.setText("");
                txtLevel.setText("");
                txtCredit.setText("");
                txtAssessments.setText("");
                txtCourseLeader.setText("");
                txtLecturer.setText("");
                txtStartDate.setText("");
                txtCompletionDate.setText("");
            }
        });

    ArrayList <Course> nonAcademicCourseList = new ArrayList<Course>();

    btnAddNac.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent addNac)
            {
```

```
            if (txtCourseIDNac.getText().isEmpty() ||
txtCourseNameNac.getText().isEmpty() ||
            txtDurationNac.getText().isEmpty() ||
txtPrerequisiteNac.getText().isEmpty())
            {
               JOptionPane.showMessageDialog(f, "Please fill up all the
fields.","Alert",JOptionPane.WARNING_MESSAGE);
               return;
            }
            else
            {
               String courseIDNac = txtCourseIDNac.getText();
               String courseNameNac = txtCourseNameNac.getText();
               int durationNac = 0;
               try
               {
                  durationNac = Integer.parseInt(txtDurationNac.getText());
               }
               catch (NumberFormatException e)
               {
                  JOptionPane.showMessageDialog(f, "Please enter a valid number for
duration.","Alert",JOptionPane.WARNING_MESSAGE);
                  return;
               }
               String prerequisiteNac = txtPrerequisiteNac.getText();

               String idNac = txtCourseIDNac.getText();
               for (Course cNac : nonAcademicCourseList)// Extract objects of arraylist
and create and store them in object of Course cNac
               {
```

```
                    if (cNac.getCourseID().equals(idNac))// Comparing courseID in
Course object with id
                    {
                        JOptionPane.showMessageDialog(f, "The id already exists, Enter a
new ID","Alert",JOptionPane.WARNING_MESSAGE);
                        return;
                    }
                }

                NonAcademicCourse objNac = new NonAcademicCourse(courseIDNac,
courseNameNac, durationNac, prerequisiteNac);
                nonAcademicCourseList.add(objNac);
            }
        }
    });

    btnRegisterNac.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent registerNac)
        {
            if (txtCourseLeaderNac.getText().isEmpty() ||
txtInstructorNac.getText().isEmpty() ||
            txtStartDateNac.getText().isEmpty() ||
txtCompletionDateNac.getText().isEmpty() || txtExamDateNac.getText().isEmpty())
            {
                JOptionPane.showMessageDialog(f, "Please fill up all the
fields.","Alert",JOptionPane.WARNING_MESSAGE);
                return;
            }
            else
            {
```

```
                String courseLeaderNac = txtCourseLeaderNac.getText();
                String instructorNac = txtInstructorNac.getText();
                String startDateNac = txtStartDateNac.getText();
                String completionDateNac = txtCompletionDateNac.getText();
                String examDateNac = txtExamDateNac.getText();

                // Extract objects from arraylist according to the index value
                for (int i = 0; i<nonAcademicCourseList.size(); i++)
                {
                   //Comparing courseID of arraylist with courseID given by the user
                   if
(nonAcademicCourseList.get(i).getCourseID().equals(txtCourseIDNac.getText()))
                   {
                      //Get objects from arraylist and downcast into NonAcademicCourse
                      NonAcademicCourse nac =
(NonAcademicCourse)(nonAcademicCourseList.get(i));
                      if(! nac.getIsRegistered())
                      {
                         //Accessing the method from NonAcademicCourse class
                         nac.register(courseLeaderNac, instructorNac, startDateNac,
completionDateNac, examDateNac);
                         JOptionPane.showMessageDialog(f, "The Non Academic Course
has been registered successfuly.");
                         return;
                      }
                      else
                      {
                         JOptionPane.showMessageDialog(f, "The Non Academic Course
has already been registered.","Alert",JOptionPane.WARNING_MESSAGE);
                         return;
                      }
```

```
                    }
                }
            }
        }
    });

    btnRemoveNac.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent removeNac)
        {
            if (txtCourseLeaderNac.getText().isEmpty() ||
txtInstructorNac.getText().isEmpty() ||
            txtStartDateNac.getText().isEmpty() ||
txtCompletionDateNac.getText().isEmpty() || txtExamDateNac.getText().isEmpty())
            {
                JOptionPane.showMessageDialog(f, "Please fill up all the
fields.","Alert",JOptionPane.WARNING_MESSAGE);
                return;
            }
            else
            {
                // Extract objects from arraylist according to the index value
                for (int i = 0; i<nonAcademicCourseList.size(); i++)
                {
                    //Comparing courseID of arraylist with courseID given by the user
                    if
(nonAcademicCourseList.get(i).getCourseID().equals(txtCourseIDNac.getText()))
                    {
                        //Get objects from arraylist and downcast into NonAcademicCourse
type
```

```
                    NonAcademicCourse nac =
(NonAcademicCourse)(nonAcademicCourseList.get(i));
                    if(! nac.getIsRemoved())
                    {
                        //Accessing the method from NonAcademicCourse class
                        nac.remove();
                        JOptionPane.showMessageDialog(f, "The Non Academic Course
has been removed successfuly.");
                        return;
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(f, "The id does not exists,
Enter a valid ID","Alert",JOptionPane.WARNING_MESSAGE);
                        return;
                    }
                }
            }
        }
    });

    btnDisplayNac.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent displayNac)
        {
            if (txtCourseLeaderNac.getText().isEmpty() ||
txtInstructorNac.getText().isEmpty() ||
            txtStartDateNac.getText().isEmpty() ||
txtCompletionDateNac.getText().isEmpty() || txtExamDateNac.getText().isEmpty())
            {
```

```
                JOptionPane.showMessageDialog(f, "The Non Academic Course has
not been registered.","Alert",JOptionPane.WARNING_MESSAGE);
                return;
            }
            else{
                // Extract objects from arraylist according to the index value
                for(int i = 0; i < nonAcademicCourseList.size(); i++)
                {
                    //Get objects from arraylist and downcast into NonAcademicCourse
                    NonAcademicCourse nac =
(NonAcademicCourse)(nonAcademicCourseList.get(i));
                    if(! nac.getIsRemoved())
                    {
                        JOptionPane.showMessageDialog(f,"NON ACADEMIC COURSE" +
"\n\n" +
                            "The Course ID of the course is: " + txtCourseIDNac.getText() +
"\n" +
                            "The Course Name of the course is: " +
txtCourseNameNac.getText() + "\n" +
                            "The Duration of the course is: " + txtDurationNac.getText() + "\n"
+
                            "The Prerequisites of the course is: " +
txtPrerequisiteNac.getText() + "\n" +
                            "The Course Leader of the course is: " +
txtCourseLeaderNac.getText() + "\n" +
                            "The Instructor of the course is: " + txtInstructorNac.getText() +
"\n" +
                            "The Starting Date of course is: " + txtStartDateNac.getText() +
"\n" +
                            "The Completion Date of the course is: " +
txtCompletionDateNac.getText() + "\n" +
```

```
                        "The Exam Date of the course is: " + txtExamDateNac.getText());
                    }
                    else{
                        JOptionPane.showMessageDialog(f,"The course has been
removed!","Alert",JOptionPane.WARNING_MESSAGE);
                    }
                }
            }
        }
    });

    btnClearNac.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent clearNac)
        {
            // Setting the values of all the text fields to empty
            txtCourseIDNac.setText("");
            txtCourseNameNac.setText("");
            txtDurationNac.setText("");
            txtPrerequisiteNac.setText("");
            txtCourseLeaderNac.setText("");
            txtInstructorNac.setText("");
            txtStartDateNac.setText("");
            txtCompletionDateNac.setText("");
            txtExamDateNac.setText("");
        }
    });
  }
}
```

Course Class

```
/**Course class is a parent of the AcademicCourse and NonAcademicCourse class
 * It's various methods can be called from the child classes.

 */public class Course
{
    //Instance Variable Declaration
    private String courseID;
    private String courseName;
    private String courseLeader;
    private int duration;

    //Creating a parameterized constructor
    public Course(String courseID, String courseName, int duration){
        this.courseID = courseID;
        this.courseName = courseName;
        this.courseLeader = "";
        this.duration = duration;
        }

    //Assigning accessor methods to return and initialize the values of variables
    public String getCourseID()
    {
        return this.courseID;
    }

    public String getCourseName()
    {
        return this.courseName;
    }
```

```
    public String getCourseLeader()
    {
        return this.courseLeader;
    }

    public int getDuration()
    {
        return this.duration;
    }

    //Using mutator method to set the values of variables
    public void setCourseLeader(String courseLeader){
        this.courseLeader = courseLeader;
    }

    public void display(){
        if(this.courseLeader.equals("")){ /*method executes if the courseLeader is empty
String*/
            System.out.println("The Course details are:");
            System.out.println("Course ID: " + courseID + "\n" + "Course Name: " +
courseName + "\n" + "Duration: "
            + duration);
        }
        else{/*method executes when courseLeader is not empty*/
            System.out.println("The Course details are:");
            System.out.println("Course ID: " + courseID + "\n" + "Course Name: " +
courseName + "\n" + "Duration: "
            + duration + "\n" + "Course Leader: " + courseLeader);
        }
    }
```

```
}
```

AcademicCourse Class

```
/** The AcademicCourse class is a subclass of the Course class
 * It inherits various methods from the parent class

 */
public class AcademicCourse extends Course
{
    //Instance Variable Declaration
    private String lecturerName;
    private String level;
    private String credit;
    private String startingDate;
    private String completionDate;
    private int numberOfAssessments;
    private boolean isRegistered;

    //Creating a parameterzed constructor
    AcademicCourse(String courseID, String courseName, int duration, String level,
String credit,int numberOfAssessments) {
        super(courseID, courseName, duration);/*Calling Super Class Constructor*/
        this.lecturerName = "";
        this.level = level;
        this.credit = credit;
        this.startingDate = "";
        this.completionDate = "";
        this.numberOfAssessments = numberOfAssessments;
        this.isRegistered = false;
```

```
    }

    //Assigning accessor methods to return and initialize the values of variables
    public String getLecturerName()
    {
        return this.lecturerName;
    }

    public int getNumberOfAssessments()
    {
        return this.numberOfAssessments;
    }

    //Using mutator method to set the values of variables
    public void setLecturerName(String lecturer){
        this.lecturerName = lecturer;
    }

    public void setNumberOfAssessments(int assessments){
        this.numberOfAssessments = assessments;
    }

    //Method to register a course
    public void register(String courseLeader, String lecturerName, String startingDate,
String completionDate) {
        if(this.isRegistered == true){ /*method executes if the condtion is true*/
            System.out.println("The academic course has already been registered.");
            System.out.println("Lecturer name: " + lecturerName + "\n" + "Starting Date:  " +
startingDate + "\n"
             + "Completion Date: " + completionDate);
        }
```

```
        else{/*else this method is executed*/
            super.setCourseLeader(courseLeader);
            this.lecturerName = lecturerName;
            this.startingDate = startingDate;
            this.completionDate = completionDate;
            this.isRegistered = true;
            System.out.println("The academic course has been registered successfully.");
            System.out.println("Course Leader: " + courseLeader + "\n"+ "Lecturer Name: "
+ lecturerName + "\n" +
            "Starting Date: " + startingDate + "\n" + "Completion Date: " + completionDate);
        }
    }

    public void display(){
        super.display();/*Calling display method from parent class*/
        if(this.isRegistered == true){/*Method executes if the condition is true*/
            System.out.println("Lecturer Name: " + lecturerName + "\n" + "Level: " + level
            + "\n" + "Credit: " + credit + "\n" + "Starting Date: " + startingDate + "\n" +
"Completion Date: " +
            completionDate + "\n" + "No. of assessments: " + numberOfAssessments);
        }
    }
}
```

NonAcademicCourse Class

```
/** The NonAcademicCourse class is a subclass of the Course class
 * It inherits various methods from the parent class

 */
```

```
public class NonAcademicCourse extends Course
{
    //Instance Variable Declaration
    private String instructorName;
    private String startDate;
    private String completionDate;
    private String examDate;
    private String prerequisite;
    private boolean isRegistered;
    private boolean isRemoved;

    NonAcademicCourse(String courseID, String courseName, int duration, String
prerequisite) {
        super(courseID, courseName, duration);//super class constructor
        this.instructorName = instructorName;
        this.startDate = startDate;
        this.completionDate = completionDate;
        this.examDate = examDate;
        this.prerequisite = prerequisite;
        this.isRegistered = false;
        this.isRemoved = false;
    }

    public String getInstructorName()
    {
        return this.instructorName;
    }

    public String getStartDate()
    {
        return this.startDate;
```

```
    }

    public String getCompletionDate()
    {
        return this.completionDate;
    }

    public String getExamDate()
    {
        return this.examDate;
    }

    public String getPrerequisite()
    {
        return this.prerequisite;
    }

    //Using mutator method to set the values of variables
    public void setInstructorName(String instructorName){
        if(this.isRegistered == false) {/*Method executes if the condition is false*/
            this.instructorName = instructorName;
        }
        else {/*Else this method gets executed*/
            System.out.println("The Course has already been registered. Instructor cannot
change");
        }
    }

    //Method to register a course
    public void register(String courseLeader, String instructorName, String startDate,
String completionDate, String examDate) {
```

```
        if(this.isRegistered == false) {/*Method eecutes if isRegistered is true*/
            super.setCourseLeader(courseLeader);
            setInstructorName(instructorName);
            this.startDate = startDate;
            this.completionDate = completionDate;
            this.examDate = examDate;
            isRegistered = true;
            System.out.println("The course has been registered.");
            System.out.println("Course leader: " + courseLeader + "\n" + "Instructor name: "
+ instructorName +"\n" +
            "Start Date: " + startDate  + "\n" + "Completion Date: " + completionDate + "\n" +
"Exam Date: " + examDate);
        }
        else{/*Else this method gets executed*/
            System.out.println("The course has already been registered.");
        }
    }

    //Method to remove a course
    public void remove(){
        super.setCourseLeader("");/*Calling setter from parent class*/
        this.instructorName = "";
        this.startDate = "";
        this.completionDate = "";
        this.examDate = "";
        this.isRegistered = false;
        this.isRemoved = true;
        if (this.isRemoved == true){/* Method executes if isRemoved is true*/
            System.out.println("The course has been removed.");
        }
    }
```

```
    public void display(){
        super.display();/*Calling display method from parent class*/
        if(this.isRegistered == true){/*Method executes if the condition is true*/
            System.out.println("Instructor Name: " + instructorName + "\n" + "Start Date: " +
startDate +
            "\n" + "Completion Date: " + completionDate + "\n" + "Exam Date: " +
examDate);
        }
    }
}
```