



the
UNIVERSITY
of
GREENWICH

COMP1752 – Object-Oriented Programming



[Your name or enrolment number]

University of Greenwich

[Date]

Table of Contents

1	Introduction.....	3
2	Class Diagram.....	4
2.1	The Class Diagram.....	4
2.2	A Brief Explanation of the Diagram	4
3	The Code for the Extra Classes Created	6
3.1	Class	6
3.2	Class	21
3.3	Class	22
3.4	Class	23
4	White Box Testing.....	59
4.1	Testing for class	59
4.2	Testing for class	59
4.3	Testing for class	59
4.4	Testing for class	59
5	Screen Shots of the Program Working.....	60
5.1	Screen 1 -	60
5.2	Screen 1 – A brief description.....	64
5.3	Screen 2 -	65
5.4	Screen 2 – A brief description.....	70
5.5	Screen 3 -	71
5.6	Screen 3 – A brief description.....	73
6	The Evaluation.....	81
6.1	What went well?.....	81
6.2	What went less well?.....	81
6.3	What was learned?	82
6.4	How would a similar task be completed differently?.....	82
6.5	How could the course be improved?.....	82
7	Self-Grading.....	83

1 INTRODUCTION

Having learnt plenty of object-oriented programming techniques in the lectures, the tutorials and during my revision time, this coursework has proven to be a perfect place where to put in practice what I have been learning through these past weeks. Due to the nature of the requirements that were asked to the students(us) to meet, for the program be built, I was able to put in practice most of the techniques that I have learnt. It is a program that without you noticing, forces you to use object-oriented techniques. For example, the list of courses. This forced me to create an object that would help me to carry out most of the tasks that had to be done for every course in the list of course.

Even though at the beginning I did not know how I was going to do this course, I managed to catch up with the work thanks to the tutorial work and the help of some of tutorial teachers.

As well as this, it is important to say that the program that we were asked to program, although is not the first program that I have coded; is the most relevant program that I have built until now. This is because of the purpose of the program and its relevance with my student life. This could be the first program that I have created (with help of course) that I am going to actually use.

2 CLASS DIAGRAM

Include a UML class diagram showing the all the classes that you have implemented and the relationship between them. Your narrative for section 2.2 should also describe the design decisions you made and the OOP techniques used (200-400 words).

2.1 THE CLASS DIAGRAM

(The class diagram will come attached with this document since the its in a picture and it is too big to be seen properly if inserted in the document)

2.2 A BRIEF EXPLANATION OF THE DIAGRAM

The *CoursePage* class which the main class extends to the *JFrame* (even though is not included in the diagram and implements the *ActionListener* and the *Key Listener*. Although the latter is not used).

The following fields and methods: *myCourse*(an object of the class *Course*), *courses*, *crsework*; *addCourse(String x)*, *showCourses()*, *showCourseWork(String x)*, *deleteCourse(String dltcourse)*, *dleteCourseWork(String Cwork)* ; were added by me to the initial *CoursePage* class.

Secondly the class *Course* was created by me to carry the task of the requirements 3.1; 3.2; and 3.4, which is to add and delete courses and course works from the list of courses and list course works.

The *Course* class extends to the *CommonCode* class inheriting all fields and methods from it. It contains the following fields: *crseName*, *crseNamePath*, *check*, *checkCW*, *ArrayList<String> theCourse*, *chekReq*, *checkIfReqOpen*, *dltOnlyCW*. This class has been propely encapsulated. Whenever it is called by the *CoursePage* and *CourseWork* classes, it uses methods to get the names of the courses, courseworks and requirements that the user has selected(*setCurrentCourse*), to provide the paths for the files to be read from and written to(*getCurrentCoursePath*); to delete courses and courseworks(and their files) and some decision methods(*getIfOpenCW*,*setifOpenCW*, etc) that are used to let know the *CoursePage* class wich methods to run according to which commands are being activated from the *actionPerformed* method.

In the *AllNotes* (which extends to the *CommonCode* class) class I added some methods to get the names of the courses that have been selected from the course drop-down list in the *CoursePage*.

Furthermore, I implemented the class *CommonCode* adding a method that would allow to append text to a file. I wrote the method because I noticed that the method used to write into a file delete all the content in the file and wrote it all over again; which I did not want to do all the time.

Finally, the *CourseWork* class was created to open a second window to manage the course works, the requirements of the coursework and the notes added to those, specially the notes for the requirements of the coursework. It is slightly similar to the *CoursePage* class in the sense that it is structured in the same way and both of them implement and extend to the same interfaces and classes. But a button to go back to the main course window is added, and when this window closes the whole program does not.

3 THE CODE FOR THE EXTRA CLASSES CREATED

Add the code you have written for each of the classes implemented here. Copy and paste relevant code - actual code please, not screen shots! Make it easy for the tutor to read. Add explanation if necessary – though your in-code comments should be clear enough.

3.1 CLASS ...COURSE

```
package myproject;

import java.io.*;
import java.util.ArrayList;
import java.io.File;
import java.io.IOException;
import java.nio.file.DirectoryNotEmptyException;
import java.nio.file.Files;
import java.nio.file.NoSuchFileException;
import java.nio.file.Paths;

/**
 *
 * @author oe7863y
 */
public class Course extends CommonCode {

    public static String crseName;
    public static String crseNamePath;
    private static boolean check;
    private boolean checkCW=false;

    ArrayList<String> theCourse = new ArrayList<>();
```

```

private static boolean chekReq;
private boolean checkIfReqOpen=false;
private boolean dltOnlyCW;

//Following class set the name of the course that is being selected in the list of courses
public void setCurrentCourse(String x) {
    String name = x;
    crseName = name;
    crseNamePath = appDir + "/" + name + ".txt";//gets the name of the course and adds a path
to it to create a file
}

//Following method get the path created in the method setCurrentCourse and returns it when its
called
public String getCurrentCoursePath() {

    return crseNamePath;
}

//Used to get the name of the course, coursework or requirements selected
public String getCurrentCourse() {

    return crseName;
}

//Creates a new course file
public void createCourse(String x,String path, String a) {
    ArrayList<String> course = new ArrayList<>();//Variable to add the new course o a list
    ArrayList<String> thecourses = new ArrayList<>();

    CreateCourseFile(path,a);

```



```

        course.clear();//List is cleared to make sure its empty before receiving any new course
        course.add(x);//course is being added to a list

        //Next code adds a space to separate in the files where all the name of courses ar stored
        //Adds the new course to a new list
        for (int i = 0; i < course.size(); i++) {
            String courses = course.get(i) + "\t";
            thecourses.add(courses);

        }
        if ("Courses".equals(getCurrentCourse())){
            File file= new File(appDir+ "/" + x + ".txt");

            //chekc if the file alredy exist and if it doesn't
            //it creates a new one
            try{
                if(!file.exists()){
                    file.createNewFile();
                }else{
                    System.out.println("File "+x+" alredy exists.");
                }

            }catch(IOException e){
                // System.out.println("File "+x+" alredy exists.");
                //this is to create a file to store a list of requirements
                //if a new coursework is created
            }

        }

        //appens the new course to the list of courses
        try {

```

```

        appendTextFile(path, thecourses);
    } catch (IOException ex) {
        System.out.println("Problem! " + path);
    }

}

//reads from a file and passes it to a list
public ArrayList<String> readCourses() {
    ArrayList<String> course = new ArrayList<>();
    ArrayList<String> readingcourse = new ArrayList<>();
    String path = getCurrentCoursePath();
    course = readTextFile(path);

    for (String str : course) {
        int i = 0;
        i++;
        String[] tmp = str.split("\t");

        readingcourse.add(tmp[i - 1]);
    }
    return readingcourse;
}

//Creates a new file for the new course, coursework or requirements
//Gets the new course from the CourseParge class
public void CreateCourseFile(String x,String a) {
    String path=x;
    File file= new File(path);

```

```

//check if the file already exist and if it doesn't
//it creates a new one
try{
if(!file.exists()){
file.createNewFile();
}else{
System.out.println("File "+x+" already exists.");
}

} catch(IOException e){
//this is to create a file to store a list of requirements
//if a new coursework is created
}
if (a!=""){
File Specfile= new File(a);
try{
if(!Specfile.exists()){
Specfile.createNewFile();
}else{
System.out.println("File "+a+" already exists.");
}

} catch(IOException e){

}

}

}

//whenever a course is deleted this method is used to delete its according file

```

```

public void deleteCourseFile(String x, ArrayList<String> a) {
    ArrayList<String> course = new ArrayList<>();//to get the list from a file
    ArrayList<String> backupcourse = new ArrayList<>();//get the list sorted
    course = a;//take the list courses
    course.remove(x);//the wanted course id removed from the list
    String path = appDir + "//Courses.txt";

    //Separate the names of the courses and put it in a new list
    //and rewrite the new edited list of courses to its file
    for (int i = 0; i < course.size(); i++) {
        String courses = course.get(i) + "\t";
        backupcourse.add(courses);
    }
    try {
        writeTextFile(path, backupcourse);
    } catch (IOException ex) {
        System.out.println("Problem! " + path);
    }

    deleteCW(x);//delete all the courses that belong to the deleted course
    String notePath = appDir + "/" + x + ".txt";
    //delete the file of the course and its notes
    try
    {
        Files.deleteIfExists(Paths.get(notePath));
    }
    catch(NoSuchFileException e)
    {

```

```

        System.out.println("No such file/directory exists");
    }
    catch(DirectoryNotEmptyException e)
    {
        System.out.println("Directory is not empty.");
    }
    catch(IOException e)
    {
        System.out.println("Invalid permissions.");
    }

    System.out.println("Deletion successful.");

}

public void deleteCWFiles(String x, ArrayList<String> a){

    ArrayList<String> course = new ArrayList<>();//to get the list from a file
    ArrayList<String> backupcourse = new ArrayList<>();//get the list sorted
    course = a;//take the list courses
    course.remove(x);//the wanted course id removed from the list
    String path = getCurrentCoursePath();

    //Separate the names of the courses and put it in a new list
    //and rewrite the new edited list of courses to its file
    for (int i = 0; i < course.size(); i++) {
        String courses = course.get(i) + "\t";
        backupcourse.add(courses);
    }
}

```

```

    }
    try {
        writeTextFile(path, backupcourse);
    } catch (IOException ex) {
        System.out.println("Problem! " + path);
    }

```

```

String notePath = appDir + "/" + x + ".txt";

```

```

//delete the file of the course and its notes

```

```

try
{
    Files.deleteIfExists(Paths.get(notePath));
}
catch(NoSuchFileException e)
{
    System.out.println("No such file/directory exists");
}
catch(DirectoryNotEmptyException e)
{
    System.out.println("Directory is not empty.");
}
catch(IOException e)
{
    System.out.println("Invalid permissions.");
}

System.out.println("Deletion successful.");

```

```

}

//deletes all the course works belonging to the deleted course

private void deleteCW(String x){

    //path to file containig the list of courseworks
    String pathCW= appDir + "/" + x + "CourseWork"+" .txt";
    String pathCWreq="";

    ArrayList<String> cWList1= new ArrayList<>();
    ArrayList<String> cWList2= new ArrayList<>();
    ArrayList<String> cWList3= new ArrayList<>();

    cWList1 = readTextFile(pathCW);//get the list of courseworks from the file
    ArrayList<String> reqList= new ArrayList<>();

    //creates list of course works separated
    for (String str : cWList1) {

        int i = 0;

        i++;

        String[] tmp = str.split("\t");

        cWList2.add(tmp[i - 1]);

        int j=0;

        //each coursework has requiremts

        //these are the paths for the file containing the list requirements
        pathCWreq=appDir+"/" + cWList2.get(j)+"Requirements"+" .txt";

        //reads from the requirements files
        cWList3=readTextFile(pathCWreq);

        //creates list of the requiremts in a course work
        for (String list : cWList3) {

            int a = 0;

            a++;

```

```

String[] temp = list.split("\t");

reqList.add(temp[a - 1]);

int b=0;

//these are the paths for the files of the requirements
String reqfiles= appDir+"//"+ reqList.get(b)+".txt";

//delete all requirement's files
try
{
    Files.deleteIfExists(Paths.get(reqfiles));
}
catch(NoSuchFileException e)
{
    System.out.println("No such file/directory exists");
}
catch(DirectoryNotEmptyException e)
{
    System.out.println("Directory is not empty.");
}
catch(IOException e)
{
    System.out.println("Invalid permissions.");
}

b++;
}

//deletes all the coursework files
try

```



```

    {
        Files.deleteIfExists(Paths.get(pathCWreq));
    }
    catch(NoSuchFileException e)
    {
        System.out.println("No such file/directory exists");
    }
    catch(DirectoryNotEmptyException e)
    {
        System.out.println("Directory is not empty.");
    }
    catch(IOException e)
    {
        System.out.println("Invalid permissions.");
    }
    j++;
}

// deletes the list of courseworks
try
{
    Files.deleteIfExists(Paths.get(pathCW));
}
catch(NoSuchFileException e)
{
    System.out.println("No such file/directory exists");
}
catch(DirectoryNotEmptyException e)
{
    System.out.println("Directory is not empty.");
}

```

```

    }
    catch(IOException e)
    {
        System.out.println("Invalid permissions.");
    }
}

//method similar to the one before
//used when only courseworks want to be deleted
public void deleteCW(String x,String y){
    String thePath= appDir+"//" + x + "CourseWork"+" .txt";
    ArrayList<String> courseW = new ArrayList<>();
    ArrayList<String> courseW2 = new ArrayList<>();
    ArrayList<String> readingcourseW = new ArrayList<>();
    courseW=readTextFile(thePath);
    for (String str : courseW) {
        int i = 0;
        i++;
        String[] tmp = str.split("\t");

        readingcourseW.add(tmp[i -1]);
    }
    readingcourseW.remove(y);
    String thePath2= appDir+"//" + y+ "Requirements"+" .txt";
    courseW2=readTextFile(thePath2);
    ArrayList<String> cwList = new ArrayList<>();
    for (String list : courseW2) {
        int a = 0;

```

```

a++;
String[] temp = list.split("\t");

cwList.add(temp[a - 1]);

int b=0;
String cwfiles= appDir+"//"+ cwList.get(b)+".txt";
try
{
    Files.deleteIfExists(Paths.get(cwfiles));
}
catch(NoSuchFileException e)
{
    System.out.println("No such file/directory exists");
}
catch(DirectoryNotEmptyException e)
{
    System.out.println("Directory is not empty.");
}
catch(IOException e)
{
    System.out.println("Invalid permissions.");
}
    b++;
}
try
{
    Files.deleteIfExists(Paths.get(thePath2));
}

```

```

        catch(NoSuchFileException e)
        {
            System.out.println("No such file/directory exists");
        }
        catch(DirectoryNotEmptyException e)
        {
            System.out.println("Directory is not empty.");
        }
        catch(IOException e)
        {
            System.out.println("Invalid permissions.");
        }
    }

    //this method is used for stetic
    //add all the courses to a file when a new course is added
    //and "file not found" appers in the list of courses
    public void IncludeItem(String y, ArrayList<String> course){
        ArrayList<String> thecourses= new ArrayList<>();
        for (int i = 0; i < course.size(); i++) {
            String courses = course.get(i) + "\t";
            thecourses.add(courses);

        }
    }
    try {
        writeTextFile(y, thecourses);
    } catch (IOException ex) {
        System.out.println("Problem! " + y);
    }
}

```

```

    }

}

//checks if either a course or coursework is being created
//input is set
public void setcheckCourseOrWork(boolean x){

check=x;

}

public boolean getCheck(){
return check;
}

//checks if the action comand for the coursework has been activated
public void setifOpenCW(boolean b) {
    checkCW=b;

}

public boolean getIFopenCW(){
    boolean checking=false;
return checkCW;
}

void setcheckCWOreq(boolean b) {
    chekReq= b;
}

```

```
//checks if action comand for requiremts has been accesed
```

```
void setifOpenReq(boolean b) {  
    checkIfReqOpen=b;  
}
```

```
boolean getIFopenReq() {  
    boolean checking=false;  
    return checkIfReqOpen;  
}
```

```
}
```

3.2 CLASS ...*COMMONCODE*

```
//appends text to a file
```

```
public void appendTextFile(String fn, ArrayList<String> outputText)  
throws FileNotFoundException, IOException {
```

```
File fileName = new File(fn);
```

```
//Filewriter is set to true to let it know that we don't want to overwrite
```

```
//Text is added
```

```
Writer output = new BufferedWriter(new FileWriter(fileName,true));
```

```
try {
```

```
for (int i = 0; i < outputText.size(); i++) {
```

```
output.write(outputText.get(i).toString() + "\n");
```

```
}
```

```
} catch (Exception e) {
```

```
System.out.println(e.getMessage());
```

```
throw e;
```

```
} finally {
```

```

output.close();
}
}

```

3.3 CLASS ...*ALLNOTES*

//gets the name of the course that has been selected

```

public void setCurrentCourse(String path) {
    String currentPath = appDir + "/" + path + ".txt";
    thePath = currentPath;
//return thePath;
}

```

//returns the name of the course

```

public String getCurrentCourse() {

    return thePath;
}

```

Implemented for the WriteAllNotes Method in the All notes Class

```

try {
    if(getCheckifAmmend()!=true){
        appendTextFile(path, writeNote);
    }
    else{
        writeTextFile(path,writeNote );
    }
} catch (IOException ex) {
    System.out.println("Problem! " + path);
}

```

3.4 CLASS ...COURSEPAGE

Method 1

```
//adds a course to the course list

public void addCourse(String x) {
    //checks if either a course or coursework is created
    boolean crseOrCW = myCourse.getCheck();

    ArrayList<String> course = new ArrayList<>();
    String getCourse;
    String spec;
    crse = x;
    // courses.add(x);
    // courses.add("COMP1753");
    // courses.add(x);

    //adds a new course or creates a coursework
    if (crseOrCW == true) {
        //add a new coursework to the list of courseworks of the course
        getCourse = myCourse.appDir + "/" + courseList.getSelectedItem().toString() +
"CourseWork" + ".txt";

        //creates a path for the requiremenst of the coursework
spec=myCourse.appDir + "/" + crse+ "Requirements" + ".txt";

        myCourse.setCurrentCourse(courseList.getSelectedItem().toString() + "CourseWork");
        course = myCourse.readCourses();
        if(course.contains("File not found")){
```



```

        course.remove("File not found");
        myCourse.IncludeItem(getCourse, course);
    }

    //created the course
    myCourse.createCourse(crse, getCourse,spec/*,courses*/);
    course = myCourse.readCourses();
    int i = course.indexOf(crse);

    //new course added to the drop down list
    courseWorkList.addItem(course.get(i));
} else {
    //adds a course
    getCourse = myCourse.appDir + "//Courses.txt";
    myCourse.setCurrentCourse("Courses");
    course = myCourse.readCourses();

    //check if "File not found is the list and deletes it"
    if(course.contains("File not found")){
        course.remove("File not found");
        myCourse.IncludeItem(getCourse, course);
    }
    myCourse.createCourse(crse, getCourse,""/*,courses*/);
    course.clear();
    course = myCourse.readCourses();
    int i = course.indexOf(crse);
    courseList.addItem(course.get(i));
}

}

```

METHOD 2

```

        //Shows the list of courses
        public void showCourses() {
            ArrayList<String> course = new ArrayList<>();
            boolean check;
            // check = myCourse.getCheck();
            myCourse.setCurrentCourse("Courses");
            //creates a path
            String path= myCourse.appDir+ "//Courses.txt";

            course = myCourse.readCourses();
            for (String crso : course) {
                //Course added to the course drop-down list
                courseList.addItem(crso);
            }
        }
    }
}

```

METHOD 3

```

public void showCourseWork(String x){
    ArrayList<String>coursework= new ArrayList<>();
    String path= x+"CourseWork";
    myCourse.setCurrentCourse(path);
    courseWorkList.removeAllItems();
    coursework= myCourse.readCourses();
    for(String crsework: coursework){
        courseWorkList.addItem(crsework);
    }
}

```

METHOD 4

```

private void deleteCourse(String dltcourse) {

```

```

myCourse.setCurrentCourse("Courses");
String path = myCourse.appDir + "//Courses.txt";
// String dltpath="//"+dltcourse+".txt";
ArrayList<String> course = new ArrayList<>();
course = myCourse.readCourses();
int i = course.indexOf(dltcourse);
myCourse.deleteCourseFile(dltcourse, course);

courseList.removeItem(course.get(i));

}

```

METHOD 5

```

private void dleteCourseWork(String cwork) {
    myCourse.setCurrentCourse(crse+"CourseWork");
    String path = myCourse.appDir + "//"+cwork+".txt";
    // String dltpath="//"+dltcourse+".txt";
    ArrayList<String> coursework = new ArrayList<>();
    coursework = myCourse.readCourses();

    int i = coursework.indexOf(cwork);

    courseWorkList.removeItem(cwork);
    myCourse.deleteCWFiles(cwork, coursework);
}

```

ACTIONPEERFORMED METHOD

The check methods called from the Course class are there so that every time you choose a course or coursework from the list of courses and courseworks, the list will automatically update and show the courseworks for a specific course and the requirements for a specific coursework.

As well as this, you will also be shown the notes inside of the requirements

@Override

```
public void actionPerformed(ActionEvent e) {  
    // System.out.println("actionPerformed not coded yet.");  
    if ("NewNote".equals(e.getActionCommand())) {  
        //Checks that no empty spaces are inputed  
        //shows an error message  
        if (txtNewNote.getText().isEmpty() || txtNewNote.getText().trim().length() <= 0  
/*empty.equals(txtNewNote.getText())*/) {  
            //repeat = true;  
            JOptionPane.showConfirmDialog(null, "The note is empty. Please enter a note", "Error  
Message", JOptionPane.INFORMATION_MESSAGE);  
        } else {  
            crse = courseList.getSelectedItem().toString();  
            addNote(txtNewNote.getText(), crse);  
            txtNewNote.setText("");  
        }  
    }  
    if ("Close".equals(e.getActionCommand())) {  
  
    }  
    if ("Exit".equals(e.getActionCommand())) {  
        System.exit(0);  
    }  
    if ("Course".equals(e.getActionCommand())) {  
        crse = courseList.getSelectedItem().toString();
```

```

//Let know that this action commad has been activated

//not run the methods that belong to the coursework coommads
myCourse.setcheckCourseOrWork(false);
myCourse.setifOpenCW(true);


//When this is accesed the coursework command is triggered
showCourseWork(crse);
showNotes(crse);
//set this to false a again
//if it gets to the coursework command as false
//then the methods in there are executed
myCourse.setifOpenCW(false);
System.out.println(crse);
// myCourse.setifOpenCW(false);

}

if ("AddNewCourse".equals(e.getActionCommand())) {
    //check that the user enters the correct inout
    String input = javax.swing.JOptionPane.showInputDialog("Enter new course name: ");
    String empty = " ";
    if (input.isEmpty() || input.trim().length() <= 0) {
        JOptionPane.showConfirmDialog(null, "Course not entered. Please enter a course",
        "Error Message", JOptionPane.INFORMATION_MESSAGE);
    } else {
        myCourse.setcheckCourseOrWork(false);
        addCourse(input);
    }
}
}

```

```

if ("DeleteCourse".equals(e.getActionCommand())) {
    crse = courseList.getSelectedItem().toString();
    deleteCourse(crse);
    txtNewNote.setText("");
}

if ("AddCoursework".equals(e.getActionCommand())) {
    //checks User enters the correct input
    String input = javax.swing.JOptionPane.showInputDialog("Enter new coursework name:");
    String empty = " ";
    if (input.isEmpty() || input.trim().length() <= 0) {
        JOptionPane.showConfirmDialog(null, "Course not entered. Please enter a course",
        "Error Message", JOptionPane.INFORMATION_MESSAGE);
    } else {
        myCourse.setcheckCourseOrWork(true);
        addCourse(input);
    }
}

if ("DeleteCoursework".equals(e.getActionCommand())) {
crse= courseList.getSelectedItem().toString();
crsework=courseWorkList.getSelectedItem().toString();
    myCourse.deleteCW(crse, crsework);
    dleteCourseWork(crsework);
}

if ("Coursework".equals(e.getActionCommand())) {
    crsework = courseList.getSelectedItem().toString();

```

```

myCourse.setcheckCourseOrWork(true);

boolean checking=myCourse.getIFopenCW();

//only if the course command has not been accessed
//prevents from opening multiple new windows
if(checking==false){
    //open a new JFrame for the coursework and the requirements
    CourseWork cwork= new CourseWork();

}

}

if ("SearchKeyword".equals(e.getActionCommand())) {
    String lyst = allNotes.searchAllNotesByKeyword("", 0, search.getText());
    txtDisplayNotes.setText(lyst);
}

}

```

Class 5...CourseWork

This class is almost the same as the *CoursePage* class with slight small changes. It used to add requirements to the coursework instead of adding courses and coursework.

/*

- * To change this license header, choose License Headers in Project Properties.
- * To change this template file, choose Tools | Templates
- * and open the template in the editor.

```
*/  
  
package myproject;  
  
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.Dimension;  
import java.awt.FlowLayout;  
import java.awt.Font;  
import java.awt.Toolkit;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
import java.awt.event.WindowEvent;  
import java.io.File;  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;  
import java.util.Calendar;  
import javax.swing.BorderFactory;  
import javax.swing.Box;  
import javax.swing.BoxLayout;  
import javax.swing.Icon;  
import javax.swing.ImageIcon;  
import javax.swing.JButton;  
import javax.swing.JComboBox;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JMenu;  
import javax.swing.JMenuBar;  
import javax.swing.JMenuItem;
```



```

import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JToolBar;

/**
 *
 * @author USER
 */
public class CourseWork extends JFrame implements ActionListener, KeyListener {

```

```

    JPanel pnl = new JPanel(new BorderLayout());
    JTextArea txtNewNote = new JTextArea();
    JTextArea txtDisplayNotes = new JTextArea();
    ArrayList<String> note = new ArrayList<>();
    ArrayList<String> courses = new ArrayList<>();
    JComboBox SpecList = new JComboBox();
    JComboBox courseList= new JComboBox();
    JComboBox courseWorkList= new JComboBox();
    //getInfo data = new getInfo();

```

```

    String crse = "";
    String crsework = "";
    String req="";
    AllNotes allNotes = new AllNotes();
    Course myCourse = new Course();

```

```

JTextField search = new JTextField();
public static String courseName;
public static String CWname;

```

```

public CourseWork() {

```

```

    model();
    view();
    controller();

```

```

}

```

```

protected JMenuItem makeMenuItem(

```

```

    String txt,
    String actionCommand,
    String toolTipText,
    Font fnt) {

```

```

    JMenuItem mnuItem = new JMenuItem();
    mnuItem.setText(txt);
    mnuItem.setActionCommand(actionCommand);
    mnuItem.setToolTipText(toolTipText);
    mnuItem.setFont(fnt);
    mnuItem.addActionListener(this);
    return mnuItem;

```

```

}

```

```

protected JButton makeButton(

```

```

    String imageName,
    String actionCommand,

```

```

        String toolTipText,
        String altText) {

    //Create and initialize the button.
    JButton button = new JButton();
    button.setToolTipText(toolTipText);
    button.setActionCommand(actionCommand);
    button.addActionListener(this);

    //Look for the image.
    String imgLocation = System.getProperty("user.dir")
        + "\\icons\\"
        + imageName
        + ".png";

    File fyle = new File(imgLocation);
    if (fyle.exists() && !fyle.isDirectory()) {
        // image found
        Icon img;
        img = new ImageIcon(imgLocation);
        button.setIcon(img);
    } else {
        // image NOT found
        button.setText(altText);
        System.err.println("Resource not found: " + imgLocation);
    }

    return button;
}

```

```

@Override

public void actionPerformed(ActionEvent e) {

    if ("NewNote".equals(e.getActionCommand())) {

        req = SpecList.getSelectedItem().toString();

        if (txtNewNote.getText().isEmpty() || txtNewNote.getText().trim().length() <=
0||(req=="File not found"||req=="")/*empty.equals(txtNewNote.getText())*/) {

            //repeat = true;

            if(req=="File not found"||req==""){

                JOptionPane.showConfirmDialog(null, "The file does not exist", "Error Message",
JOptionPane.INFORMATION_MESSAGE);

            }else{

                JOptionPane.showConfirmDialog(null, "The note is empty. Please enter a note", "Error
Message", JOptionPane.INFORMATION_MESSAGE);}

            } else {

                addNote(txtNewNote.getText(), req);

                txtNewNote.setText("");

            }

        }

        if ("ReturnToCourseNotes".equals(e.getActionCommand())) {

            WindowEvent winevent = new WindowEvent(this, WindowEvent.WINDOW_CLOSING);

            Toolkit.getDefaultToolkit().getSystemEventQueue().postEvent(winevent);

        }

        if ("Exit".equals(e.getActionCommand())) {

```

```

        //only close the window current window and not the whole program
        WindowEvent winevent = new WindowEvent(this,
WindowEvent.WINDOW_CLOSING);
        Toolkit.getDefaultToolkit().getSystemEventQueue().postEvent(winevent);
    }

    if ("courseList".equals(e.getActionCommand())) {
        crse = courseList.getSelectedItem().toString();
        //Let know that this action command has been activated
        myCourse.setcheckCourseOrWork(false);
        myCourse.setifOpenCW(true);

        showCourseWork(crse);
        crsework = courseWorkList.getSelectedItem().toString();
        showRequirements(crsework);
        req= SpecList.getSelectedItem().toString();

        showNotes(req);
        //set it back to false
        myCourse.setifOpenCW(false);
        System.out.println(crse);

    }

    if ("courseWorkList".equals(e.getActionCommand())) {
        // crse = courseList.getSelectedItem().toString();
        boolean checking=myCourse.getIFopenCW();
        if(checking==false){

```

```

crsework = courseWorkList.getSelectedItem().toString();
    req = SpecList.getSelectedItem().toString();
    myCourse.setcheckCWOOrReq(false);
    myCourse.setifOpenReq(true);
    showRequirements(crsework);
    showNotes(req);
    myCourse.setifOpenReq(false);
}

}

```

```

if ("Requirements".equals(e.getActionCommand())) {

```

```

    boolean checking=myCourse.getIFopenReq();
    boolean check2=myCourse.getIFopenCW();
    if(checking==false && check2==false){
        req = SpecList.getSelectedItem().toString();
        // myCourse.setcheckCourseOrWork(false);
        // myCourse.setifOpenCW(true);
        showNotes(req);
        // showCourseWork(crse);
        // myCourse.setifOpenCW(false);
        System.out.println(req);
        // myCourse.setifOpenCW(false);
    }
}

```

```

if ("AddNewRequirement".equals(e.getActionCommand())) {

```

```

    // String input = javax.swing.JOptionPane.showInputDialog("Enter new course name: ");

```

```

// addCourse(input);

crsework=courseWorkList.getSelectedItem().toString();

String input = javax.swing.JOptionPane.showInputDialog("Enter new course name: ");

String empty = " ";

if (input.isEmpty() || input.trim().length() <= 0||(crsework=="File not
found"||crsework=="")) {

    if(crsework=="File not found"||crsework==""){

        JOptionPane.showConfirmDialog(null, "The file does not exist", "Error Message",
JOptionPane.INFORMATION_MESSAGE);

    }else{

        JOptionPane.showConfirmDialog(null, "Course not entered. Please enter a course",
"Error Message", JOptionPane.INFORMATION_MESSAGE);}

    } else {

        myCourse.setcheckCourseOrWork(false);

        addRequirements(input);

    }

}

if ("DeleteRequirement".equals(e.getActionCommand())) {

    req = SpecList.getSelectedItem().toString();

    deleteRequirements(req);

    txtNewNote.setText("");

}

if ("SaveNote".equals(e.getActionCommand())) {

    if (txtNewNote.getText().isEmpty() || txtNewNote.getText().trim().length() <= 0
/*empty.equals(txtNewNote.getText())*/) {

```

```

        //repeat = true;

        JOptionPane.showConfirmDialog(null, "The note is empty. Please enter a note", "Error
Message", JOptionPane.INFORMATION_MESSAGE);
    } else {
        crse = SpecList.getSelectedItem().toString();
        addNote(txtNewNote.getText(), crse);
        txtNewNote.setText("");
    }

}

if ("SearchKeyword".equals(e.getActionCommand())) {
    String lyst = allNotes.searchAllNotesByKeyword("", 0, search.getText());
    txtDisplayNotes.setText(lyst);
}
}

@Override

public void keyTyped(KeyEvent ke) {
    // throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.
}

@Override

public void keyPressed(KeyEvent ke) {
    // throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.
}

@Override

public void keyReleased(KeyEvent ke) {

```



```
//throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.
```

```
}
```

```
public void model() {
```

```
    showCourses();
```

```
    crse = courseList.getSelectedItem().toString();
```

```
    showCourseWork(crse);
```

```
    crsework=courseWorkList.getSelectedItem().toString();
```

```
    showRequirements(crsework);
```

```
    req= SpecList.getSelectedItem().toString();
```

```
    showNotes(req);
```

```
//throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.
```

```
}
```

```
public void view() {
```

```
    Font fnt = new Font("Georgia", Font.PLAIN, 24);
```

```
    JMenuBar menuBar = new JMenuBar();
```

```
    JMenu course = new JMenu("Requirements");
```

```
    course.setToolTipText("List of requirements for this coursework");
```

```
    course.setFont(fnt);
```

```
    course.add(makeMenuItem("Add new requirement", "AddNewRequirement", " Add a new
requirement", fnt));
```

```
    course.addSeparator();
```

```
course.add(makeMenuItem("Delete a requirement", "DeleteRequirement", " Delete an  
existing requirement", fnt));
```

```
course.addSeparator();
```

```
course.add(makeMenuItem("Amend a requirement", "AmendRequirement", " Edit an  
existing requirement", fnt));
```

```
menuBar.add(course);
```

```
menuBar.add(makeMenuItem("Exit", "Exit", "Close this program", fnt));
```

```
courseList.setFont(fnt);
```

```
courseList.setToolTipText("Course List");
```

```
courseList.setMaximumSize(courseList.getPreferredSize());
```

```
courseList.addActionListener(this);
```

```
courseList.setActionCommand("courseList");
```

```
menuBar.add(courseList);
```

```
courseWorkList.setFont(fnt);
```

```
courseWorkList.setToolTipText("Coursework List");
```

```
courseWorkList.setMaximumSize(courseWorkList.getPreferredSize());
```

```
courseWorkList.addActionListener(this);
```

```
courseWorkList.setActionCommand("courseWorkList");
```

```
menuBar.add(courseWorkList);
```

```
SpecList.setFont(fnt);
```

```
SpecList.setToolTipText("Requirements");
```

```
SpecList.setMaximumSize(SpecList.getPreferredSize());
```

```

SpecList.addActionListener(this);
SpecList.setActionCommand("Requirements");

menuBar.add(SpecList);

this.setJMenuBar(menuBar);

JToolBar toolBar = new JToolBar();
// Setting up the ButtonBar
JButton button = null;
button = makeButton("Left", "ReturnToCourseNotes",
    "Return to the Course notes page",
    "Note");
toolBar.add(button);
toolBar.addSeparator();    // This forces anything after it to the right.
button = makeButton("Save", "SaveNote",
    "Save a note",
    "SaveIt");
toolBar.add(button);
toolBar.add(Box.createHorizontalGlue());
add(toolBar, BorderLayout.NORTH);

search.setMaximumSize(new Dimension(6900, 30));
search.setFont(fnt);
toolBar.add(search);
toolBar.addSeparator();
button = makeButton("search", "SearchKeyword", "Search for this text.", "Search");
toolBar.add(button);
add(toolBar, BorderLayout.NORTH);

```

```

JPanel pnlWest = new JPanel();
pnlWest.setLayout(new BoxLayout(pnlWest, BoxLayout.Y_AXIS));
pnlWest.setBorder(BorderFactory.createLineBorder(Color.black));

txtNewNote.setFont(fnt);
pnlWest.add(txtNewNote);

JButton btnAddNote = new JButton("Add note");
btnAddNote.setActionCommand("SaveNote");
btnAddNote.addActionListener(this);
pnlWest.add(btnAddNote);

add(pnlWest, BorderLayout.WEST);

JPanel cen = new JPanel();
cen.setLayout(new BoxLayout(cen, BoxLayout.Y_AXIS));
cen.setBorder(BorderFactory.createLineBorder(Color.black));
txtDisplayNotes.setFont(fnt);
cen.add(txtDisplayNotes);

add(cen, BorderLayout.CENTER);

setExtendedState(JFrame.MAXIMIZED_BOTH);
setTitle(/*data.courseList.getSelectedItem().toString()+"CourseWork");
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

setVisible(true); // Needed to ensure that the items can be seen.

```

```
}
```

```
private void showNotes(String crse) {  
    // throw new UnsupportedOperationException("Not supported yet."); //To change body of  
generated methods, choose Tools | Templates.
```

```
    AllNotes theNotes = new AllNotes();
```

```
    theNotes.setCurrentCourse(crse);
```

```
    theNotes.readAllNotes();
```

```
    String txtNotes = "";
```

```
    for (Note n : theNotes.getAllNotes()) {
```

```
        txtNotes += n.getNote() + "\n";
```

```
    }
```

```
    txtDisplayNotes.setText(txtNotes);
```

```
}
```

```
public void showRequirements(String x) {
```

```
    ArrayList<String> course = new ArrayList<>();
```

```
    boolean check;
```

```
    check = myCourse.getCheck();
```

```
    myCourse.setCurrentCourse(x+"Requirements");
```

```
    SpecList.removeAllItems();
```

```
    course = myCourse.readCourses();
```

```
    //course.remove("File not found");
```

```
    for (String crso : course) {
```

```
        SpecList.addItem(crso);
```

```
    }
```

```
}
```

```
    public void showCourseWork(String x){  
        ArrayList<String>coursework= new ArrayList<>();  
        String path= x+"CourseWork";  
        myCourse.setCurrentCourse(path);  
        courseWorkList.removeAllItems();  
        coursework= myCourse.readCourses();  
        for(String crsework: coursework){  
            courseWorkList.addItem(crsework);  
        }  
    }  
}
```

```
    public void showCourses() {  
        ArrayList<String> course = new ArrayList<>();  
        boolean check;  
        check = myCourse.getCheck();  
        myCourse.setCurrentCourse("Courses");  
        String path= myCourse.appDir+ "//Courses.txt";  
  
        course = myCourse.readCourses();  
        for (String crso : course) {  
  
            courseList.addItem(crso);  
        }  
    }  
}
```

```
    private void addNote(String text, String coursename) {
```

```

allNotes.addNote(allNotes.getMaxID(), coursename, text);

// note.clear();
addAllNotes();

}

private void addAllNotes() {
    String txtNotes = "";
    AllNotes Notes = new AllNotes();
    Notes.setCurrentCourse(crse);
    Notes.readAllNotes();
    for (Note n : Notes.getAllNotes()) {
        txtNotes += n.getNote() + "\n";
    }

    txtDisplayNotes.setText(txtNotes);
}

private void deleteRequirements(String dltcourse) {
    myCourse.setCurrentCourse(crsework+"Requirements");
    String path = myCourse.appDir + "//Courses.txt";
    // String dltpath="//"+dltcourse+".txt";
    ArrayList<String> course = new ArrayList<>();
    course = myCourse.readCourses();
    int i = course.indexOf(dltcourse);
    SpecList.removeItem(course.get(i));
    myCourse.deleteCourseFile(dltcourse, course);

}

```

```

public void addRequirements(String x) {
    // myCourse.setCurrentCourse(x);
    // courses=myCourse.readCourses();
    boolean crseOrCW = myCourse.getCheck();

    ArrayList<String> course = new ArrayList<>();
    String getCourse;
    String spec;
    String fileName = x;
    // courses.add(x);
    // courses.add("COMP1753");
    // courses.add(x);

    getCourse = myCourse.appDir + "/" + crsework + "Requirements.txt";
    myCourse.setCurrentCourse(crsework + "Requirements");
    String reqName = myCourse.appDir + "/" + x + ".txt";
    myCourse.createCourse(fileName, getCourse, reqName/*,courses*/);
    course = myCourse.readCourses();
    int i = course.indexOf(fileName);
    SpecList.addItem(course.get(i));

}

private void controller() {
    //System.out.println("controller not coded yet.");
    addAllNotes();
}
}

```


CLASS 6...AMMEND

/*

* To change this license header, choose License Headers in Project Properties.

* To change this template file, choose Tools | Templates

* and open the template in the editor.

*/

package myproject;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Font;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.WindowEvent;

import java.io.File;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.BorderFactory;

import javax.swing.Box;

import javax.swing.BoxLayout;

import javax.swing.Icon;

import javax.swing.ImageIcon;

import javax.swing.JButton;

```

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JToolBar;
import javax.swing.text.BadLocationException;

/**
 *
 * @author USER
 */
public class Ammend extends JFrame implements GetData, ActionListener, KeyListener {

    JTextArea txtDisplayNotes = new JTextArea();
    JPanel pnl = new JPanel(new BorderLayout());
    Font fnt = new Font("Georgia", Font.PLAIN, 24);
    ArrayList<String> notes = new ArrayList<>();
    AllNotes allNotes = new AllNotes();
    private static String theObjName;
    private static String theCommand;
    //to count how many notes are going to
    private static int counting = 1;

    Ammend(String fileName, String command) {
        //gets name of the file where the notes are
        this.theObjName = fileName;
        //gets the commands that are being used
        this.theCommand = command;
        model();
    }

```

```

        view();
        controller();
    }

    private void model() {
        checkWhatToAmmend();
    }

    private void checkWhatToAmmend() {
        String checking = getCommandName();
        String getfile = getObjName();
        if (checking.contains("AmendCourse")) {
            showCourse(getfile);
        }
        if (checking.contains("AmendCoursework")) {
            showCW(getfile);
        }
        if (checking.contains("AmendRequirement")) {
            showReq(getfile);
        }
        if (checking.contains("AmendNotesInRequirements")) {
            showReqNotes(getfile);
        }
    }

    protected JButton makeButton(
        String imageName,
        String actionCommand,
        String toolTipText,

```

```

        String altText) {

    //Create and initialize the button.
    JButton button = new JButton();
    button.setToolTipText(toolTipText);
    button.setActionCommand(actionCommand);
    button.addActionListener(this);

    //Look for the image.
    String imgLocation = System.getProperty("user.dir")
        + "\\icons\\"
        + imageName
        + ".png";

    File fyle = new File(imgLocation);
    if (fyle.exists() && !fyle.isDirectory()) {
        // image found
        Icon img;
        img = new ImageIcon(imgLocation);
        button.setIcon(img);
    } else {
        // image NOT found
        button.setText(altText);
        System.err.println("Resource not found: " + imgLocation);
    }

    return button;
}

```

```

private void view() {
    JToolBar toolBar = new JToolBar();
    // Setting up the ButtonBar
    JButton button = null;
    button = makeButton("Left", "ReturnToCourseNotes",
        "Return to the Course notes page",
        "Note");
    toolBar.add(button);
    toolBar.addSeparator();    // This forces anything after it to the right.
    button = makeButton("Save", "SaveNote",
        "Save a note",
        "SaveIt");
    toolBar.add(button);
    toolBar.add(Box.createHorizontalGlue());
    add(toolBar, BorderLayout.NORTH);

    JPanel cen = new JPanel();
    cen.setLayout(new BoxLayout(cen, BoxLayout.Y_AXIS));
    cen.setBorder(BorderFactory.createLineBorder(Color.black));
    txtDisplayNotes.setFont(fnt);
    cen.add(txtDisplayNotes);

    add(cen, BorderLayout.CENTER);

    setExtendedState(JFrame.MAXIMIZED_BOTH);
    setTitle(/*data.courseList.getSelectedItemAt().toString()+*/"Ammend Window");
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    setVisible(true);
}

```

```

    }

    private void controller() {
        addAllNotes();
    }

    @Override
    public void actionPerformed(ActionEvent ae) {
        if ("SaveNote".equals(ae.getActionCommand())) {

            if (txtDisplayNotes.getText().isEmpty() || txtDisplayNotes.getText().trim().length() <= 0
/*empty.equals(txtNewNote.getText())*/) {

                //repeat = true;

                JOptionPane.showConfirmDialog(null, "The note is empty. Please enter a note", "Error
Message", JOptionPane.INFORMATION_MESSAGE);
            } else {
                try {
                    // crse = SpecList.getSelectedItem().toString();
                    int lncount = txtDisplayNotes.getLineCount() - 1;

                    int lncount2, lncount3, textLegth;
                    String textLine;

                    //this code get every line of text from the text area
                    for (int i = 0; i < lncount;) {
                        //gets the beginig of the line
                        lncount2 = txtDisplayNotes.getLineStartOffset(i);

                        //gets the end of the line
                        lncount3 = txtDisplayNotes.getLineEndOffset(i);

                        //calculates the length of the line

```

```

        textLegth = (Incount3 - Incount2) - 1;
        //gets the line of texts
        textLine = txtDisplayNotes.getText(Incount2, textLegth);
        System.out.println(Incount);
        addNote(textLine, getObjName(), counting);
        i++;
        //keeps track of how many lines of texts there are
        counting++;
    }

    System.out.println(Incount);

    // addNote(txtDisplayNotes.getText(), getObjName());
    txtDisplayNotes.setText("");
} catch (BadLocationException ex) {
    Logger.getLogger(Ammend.class.getName()).log(Level.SEVERE, null, ex);
}
}

}

if ("ReturnToCourseNotes".equals(ae.getActionCommand())) {
//prevents the whole program from closin
    WindowEvent winevent = new WindowEvent(this,
WindowEvent.WINDOW_CLOSING);
    Toolkit.getDefaultToolkit().getSystemEventQueue().postEvent(winevent);
}

}

```

@Override

```
public void keyTyped(KeyEvent ke) {  
    throw new UnsupportedOperationException("Not supported yet."); //To change body of  
generated methods, choose Tools | Templates.  
}
```

@Override

```
public void keyPressed(KeyEvent ke) {  
    throw new UnsupportedOperationException("Not supported yet."); //To change body of  
generated methods, choose Tools | Templates.  
}
```

@Override

```
public void keyReleased(KeyEvent ke) {  
    throw new UnsupportedOperationException("Not supported yet."); //To change body of  
generated methods, choose Tools | Templates.  
}
```

@Override

```
public void setObjName(String x) {  
    this.theObjName = x;  
}
```

@Override

```
public String getObjName() {  
  
    return this.theObjName;  
}
```


@Override

```
public void setCommandName(String x) {  
    this.theCommand = x;  
  
}
```

@Override

```
public String getCommandName() {  
    return this.theCommand;  
}
```

```
private void showCourse(String file) {
```

```
    throw new UnsupportedOperationException("Not supported yet."); //To change body of  
generated methods, choose Tools | Templates.
```

```
}
```

```
private void showCW(String file) {
```

```
    throw new UnsupportedOperationException("Not supported yet."); //To change body of  
generated methods, choose Tools | Templates.
```

```
}
```

```
private void showReq(String file) {
```

```
    throw new UnsupportedOperationException("Not supported yet."); //To change body of  
generated methods, choose Tools | Templates.
```

```
}
```

```
private void showReqNotes(String file) {
```

```
    //Shows the notes to be edited in the text area of the panel
```

```
    showNotes(file);
```

```
}
```

```

private void showNotes(String crse) {
    // throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.

    AllNotes theNotes = new AllNotes();
    theNotes.setCurrentCourse(crse);
    theNotes.readAllNotes();
    String txtNotes = "";

    for (Note n : theNotes.getAllNotes()) {
        txtNotes += n.getNote() + "\n";
    }

    txtDisplayNotes.setText(txtNotes);
}

private void addNote(String text, String courseName, int countedLns) {
    //this if statements decides wheter notes are to be written to a file
    //Or appended to them
    //If there is only one line of text then edited notes are written
    //if there are more than one, the ones after the first one are appended
    if (countedLns == 1) {
        allNotes.setCheckifAmmend(true);
    } else if (countedLns > 1) {
        allNotes.setCheckifAmmend(false);
    }

    allNotes.addNote(allNotes.getMaxID(), courseName, text);

    addAllNotes();
}

```

```
}
```

```
private void addAllNotes() {  
    String txtNotes = "";  
    AllNotes Notes = new AllNotes();  
    Notes.setCurrentCourse(getObjName());  
    Notes.readAllNotes();  
    for (Note n : Notes.getAllNotes()) {  
        txtNotes += n.getNote() + "\n";  
    }  
}
```

```
}
```

```
}
```

CLASS 7(Interface)...getData

```
package myproject;
```

```
/**
```

```
 *
```

```
 * @author USER
```

```
 */
```

```
interface getData {
```

```
    void setObjName(String x);
```

```
String getObjName();  
void setCommandName(String x);  
String getCommandName();  
  
}
```

4 WHITE BOX TESTING

Design white box testing of your system (using a test table) and provide evidence of both the functionality of your program and the testing results.

4.1 TESTING FOR CLASS ...

4.2 TESTING FOR CLASS ...

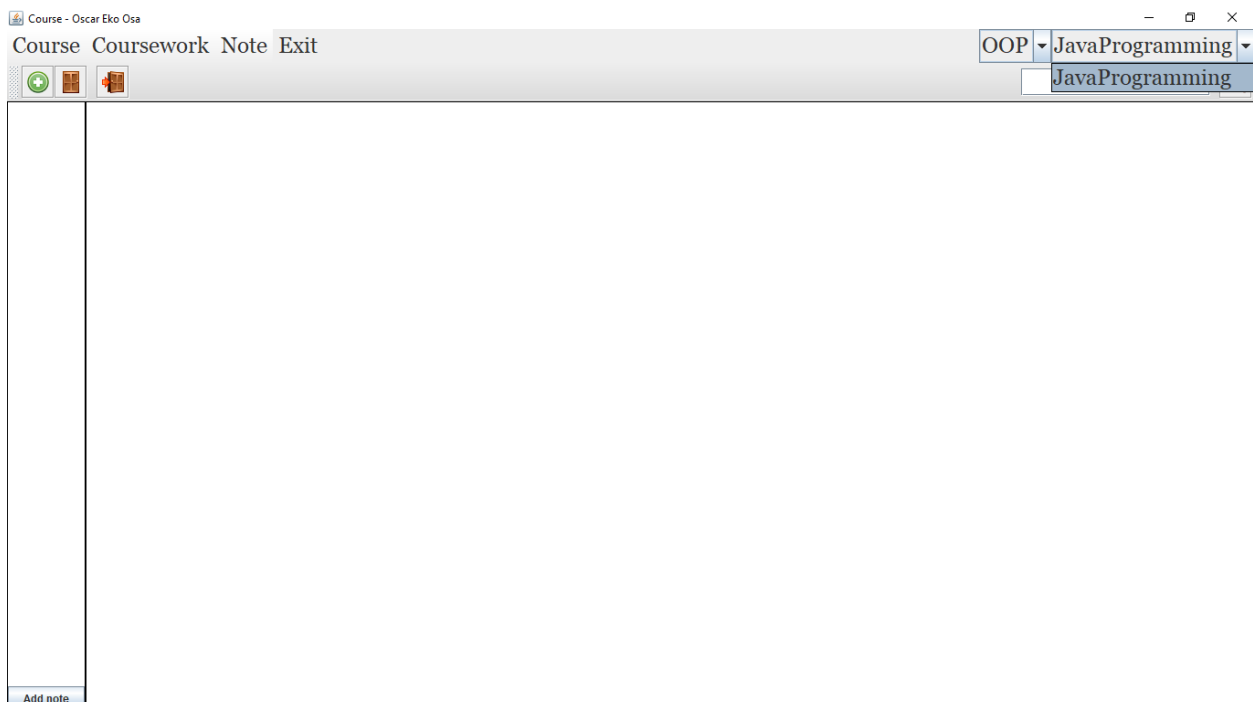
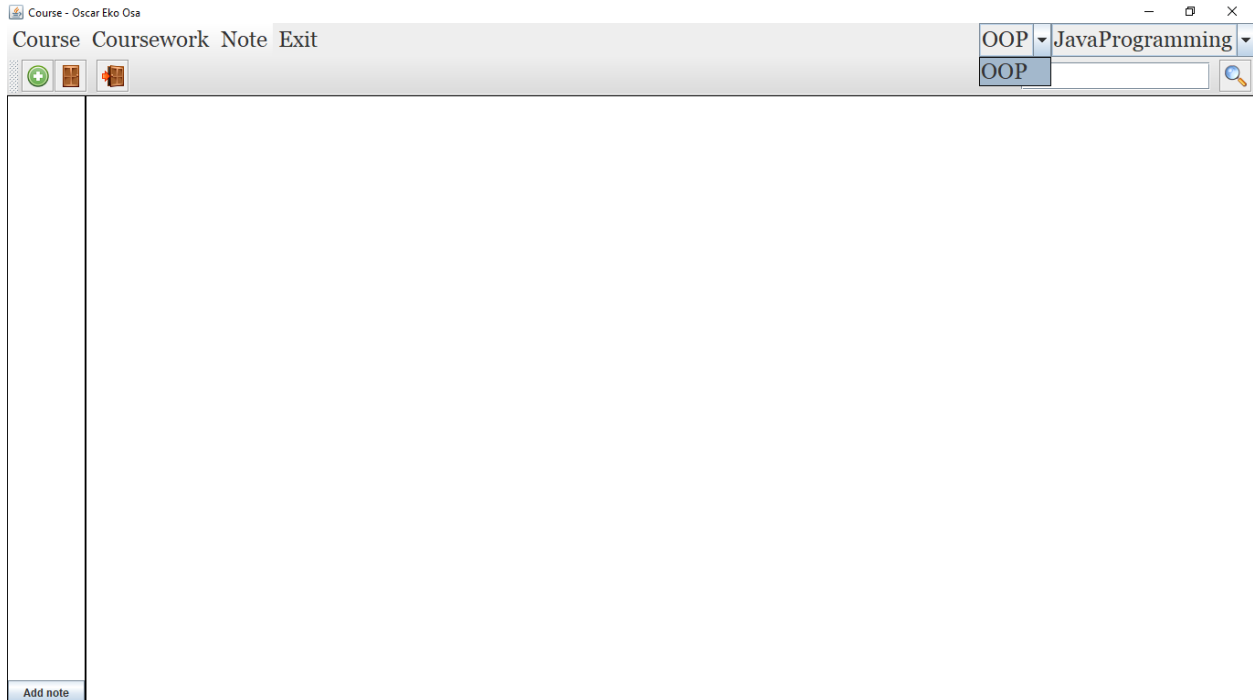
4.3 TESTING FOR CLASS ...

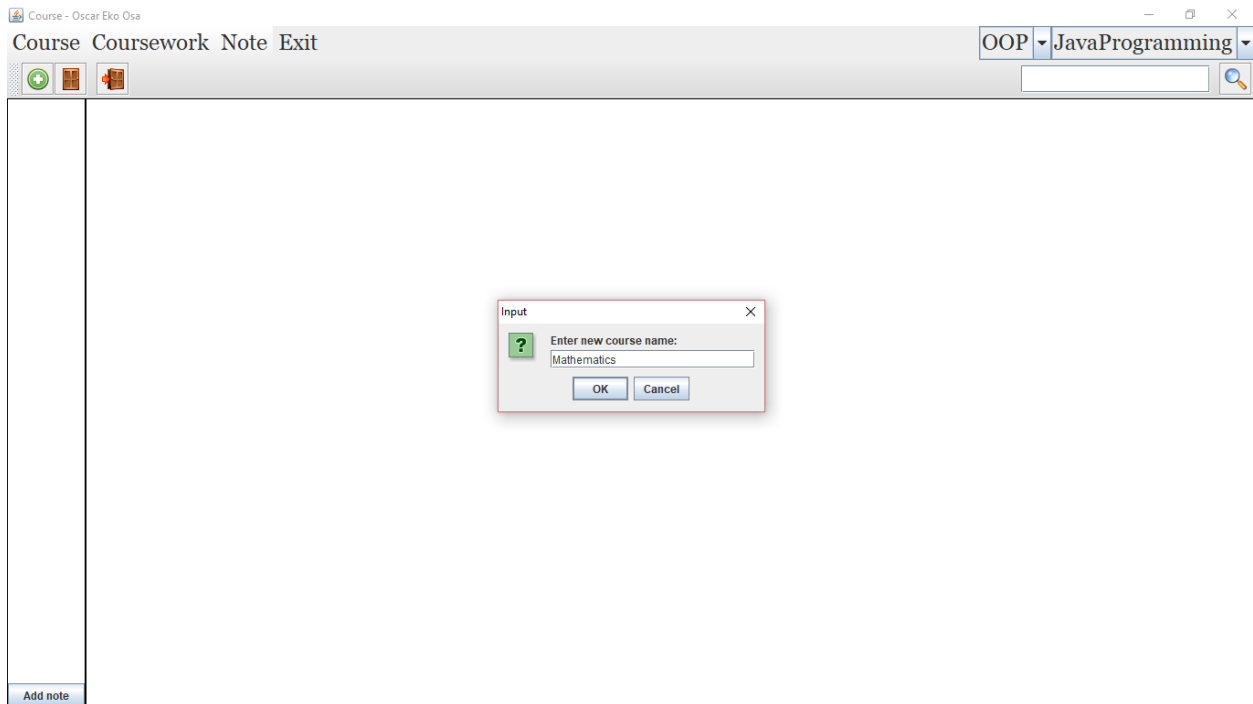
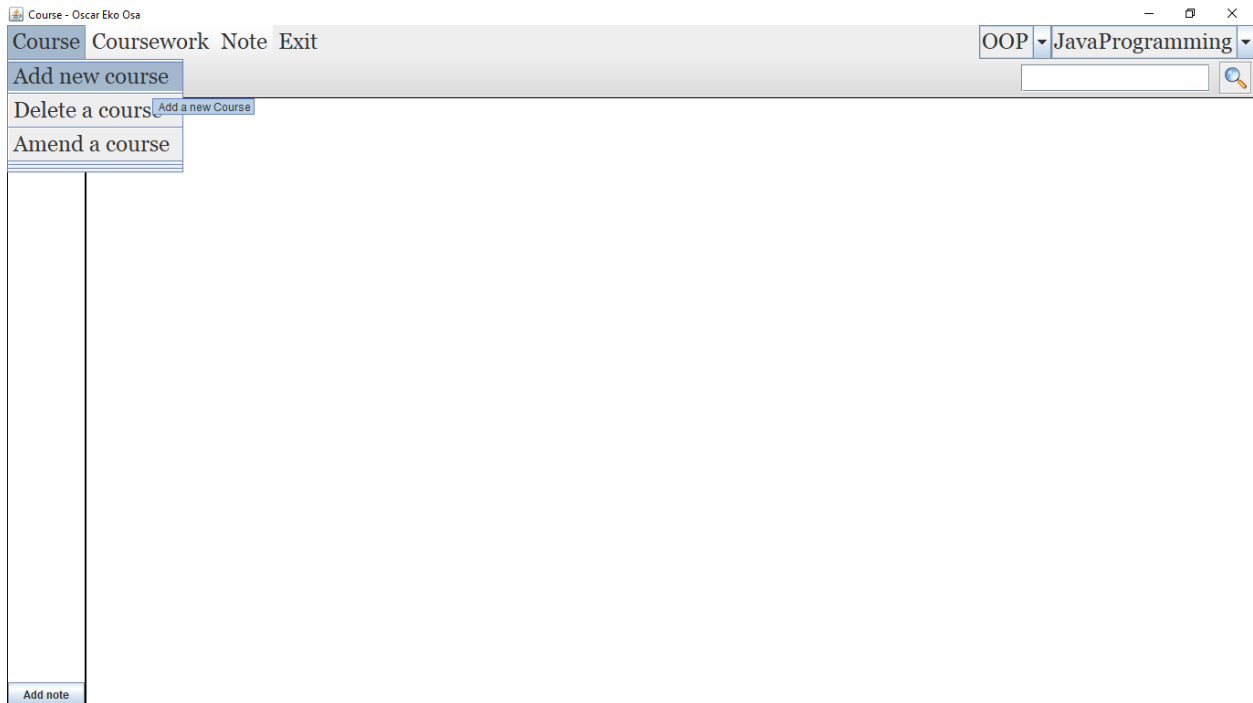
4.4 TESTING FOR CLASS ...

5 SCREEN SHOTS OF THE PROGRAM WORKING

Provide screen shots that demonstrate the features that you have implemented. Give a brief description for each (up to 100 words) to explain which features are being demonstrated. Make sure that the screen shots make clear what you have implemented and achieved.

5.1 SCREEN 1 – OPENING THE PROGRAM AND ADDING COURSES AND COURSE





Course - Oscar Eko Osa

Course Coursework Note Exit

OOP JavaProgramming

OOP

Mathematics

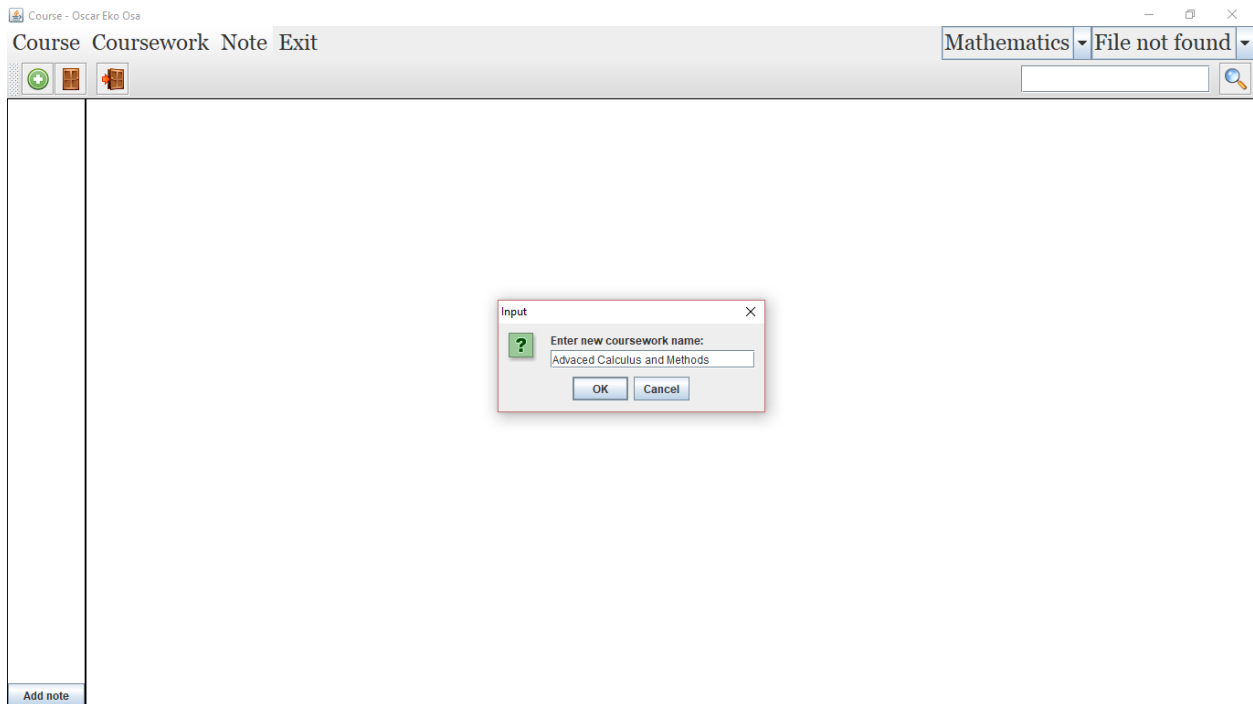
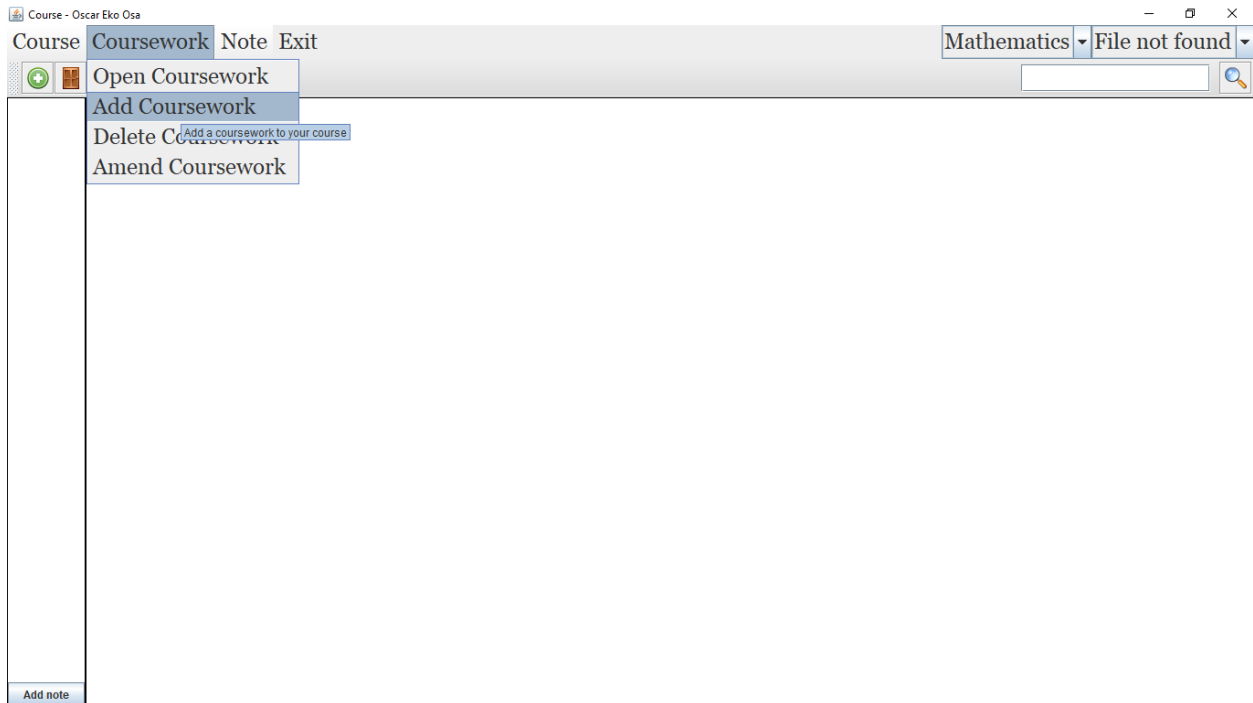
Add note

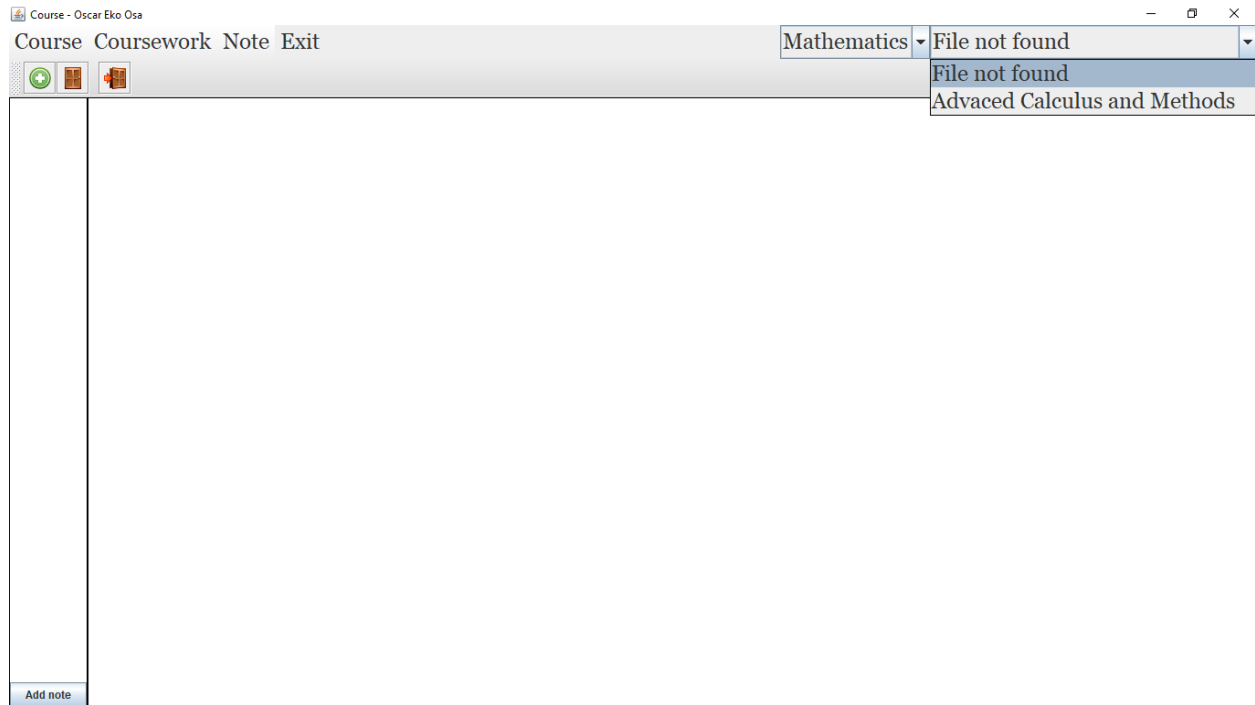
Course - Oscar Eko Osa

Course Coursework Note Exit

Mathematics File not found

Add note

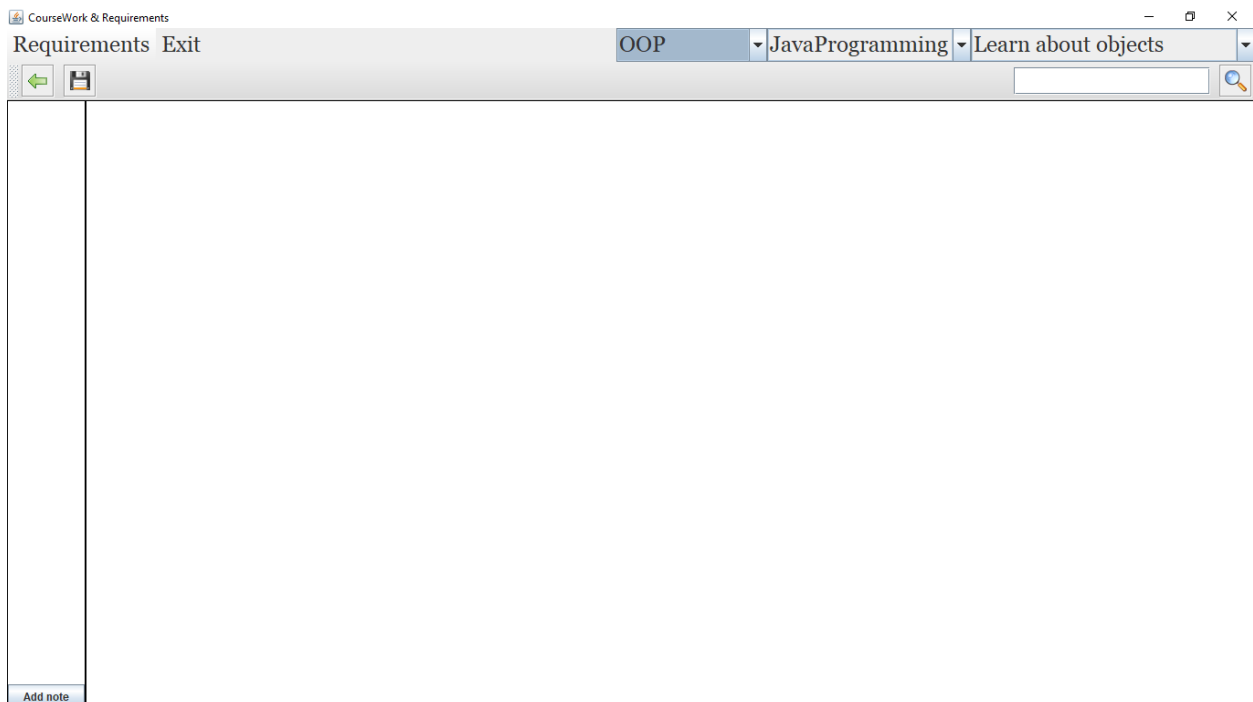
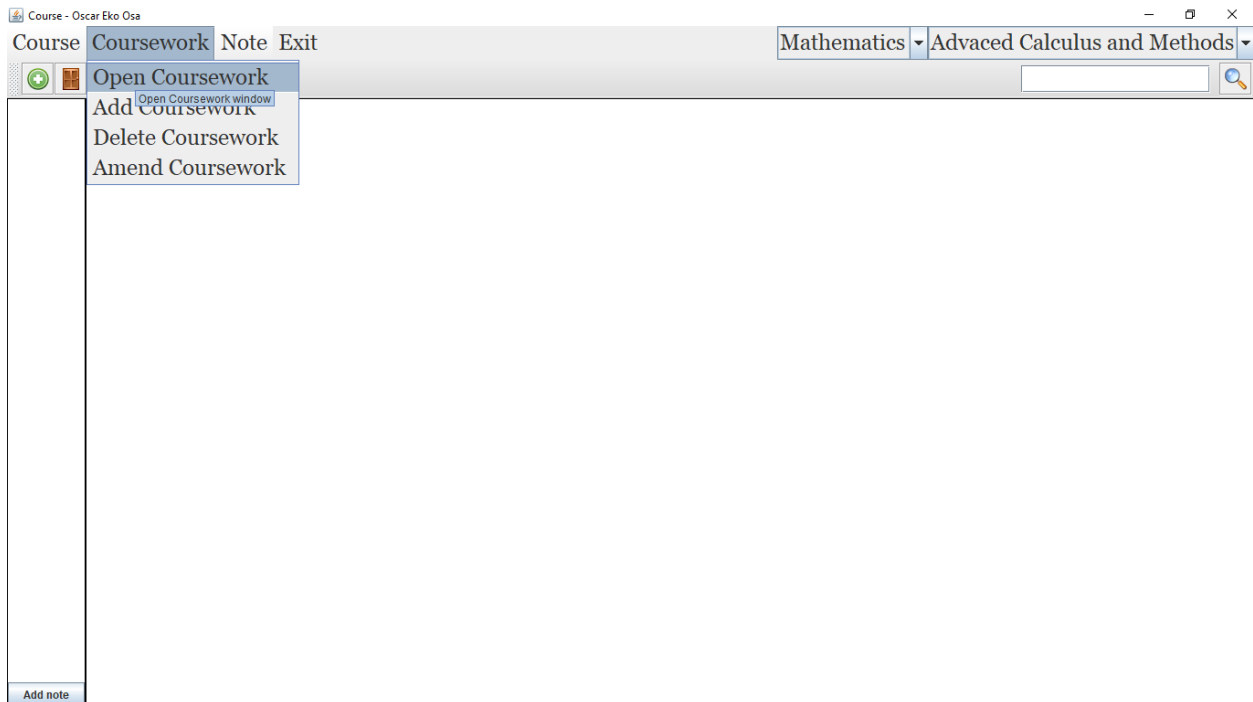




5.2 SCREEN 1 – A BRIEF DESCRIPTION

The program is running here, you can see at the top left of the screen with a menu. The “Course” and “Coursework” options were added by me. They are drop-down menus that offer you different options (Requirements of the program). At the top right part of the program you can see two drop-down lists; the first contains the list of courses and the second one the list of the courseworks. Every time you choose a different course it shows you its list of courseworks.

5.3 SCREEN 2 -COURSEWORK SCREEN



CourseWork & Requirements

Requirements Exit

OOP JavaProgramming Learn about objects

Object are basically classes
Lets see if it works in here

Add note

CourseWork & Requirements

Requirements Exit

OOP JavaProgramming Learn about objects

Objects are really useful beacause you can reuse them

Object are basically classes
Lets see if it works in here

Add note

CourseWork & Requirements

Requirements Exit

OOP ▾ JavaProgramming ▾ Learn about objects ▾

← ↻ 🔍

Object are basically classes
Lets see if it works in here
Objects are really useful beacause you can reuse them

Add note

CourseWork & Requirements

Requirements Exit

Mathematics ▾ Advaced Calculus and Methods ▾ Bernoulli Formulas ▾

← ↻ 🔍

Add note

CourseWork & Requirements

Requirements Exit

Mathematics ▾ Advanced Calculus and Methods ▾ Bernoulli Formulas ▾

← ↻

$y' + p(x)y = q(x)y^n$

Add note

CourseWork & Requirements

Requirements Exit

Mathematics ▾ Advanced Calculus and Methods ▾ Bernoulli Formulas ▾

← ↻



$y' + p(x)y = q(x)y^n$

Add note


CourseWork & Requirements

Requirements Exit

Mathematics ▾ Advanced Calculus and Methods ▾ Bernoulli Formulas ▾

$y' + p(x)y = q(x)y^n$





Add note


CourseWork & Requirements

Requirements Exit

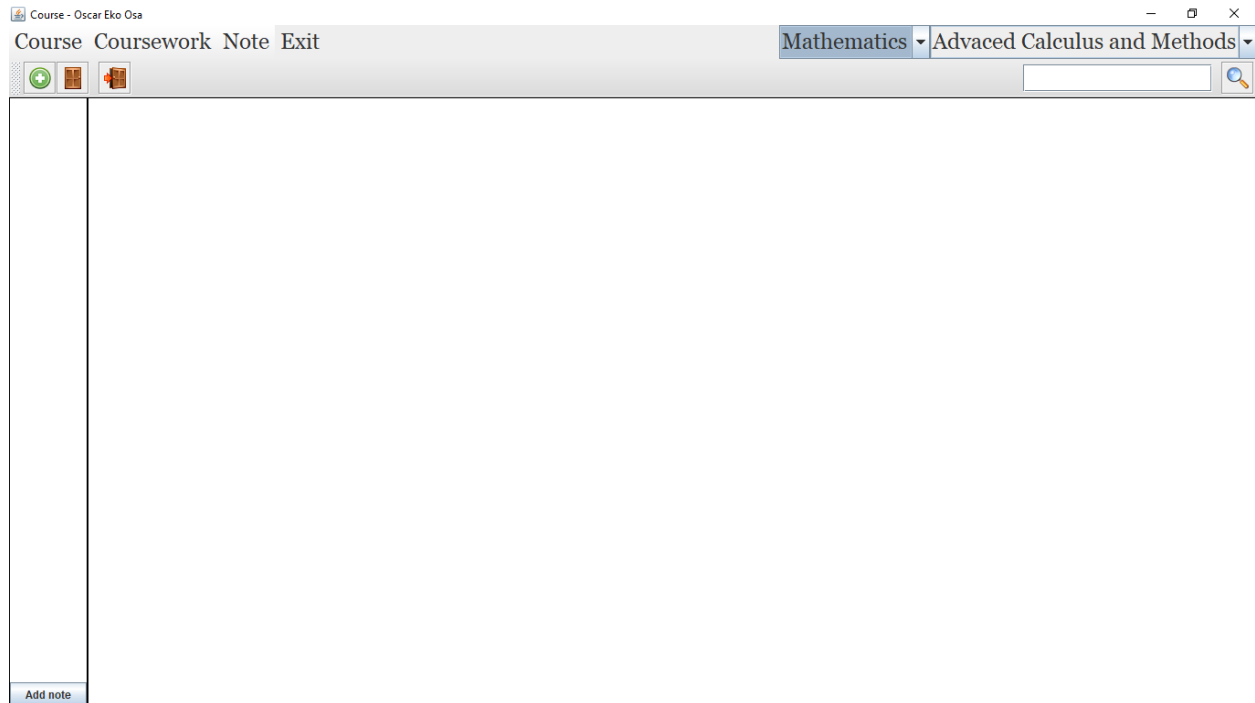
Mathematics ▾ Advanced Calculus and Methods ▾ Bernoulli Formulas ▾

  [Return to the Course notes page](#)

$y' + p(x)y = q(x)y^n$



Add note



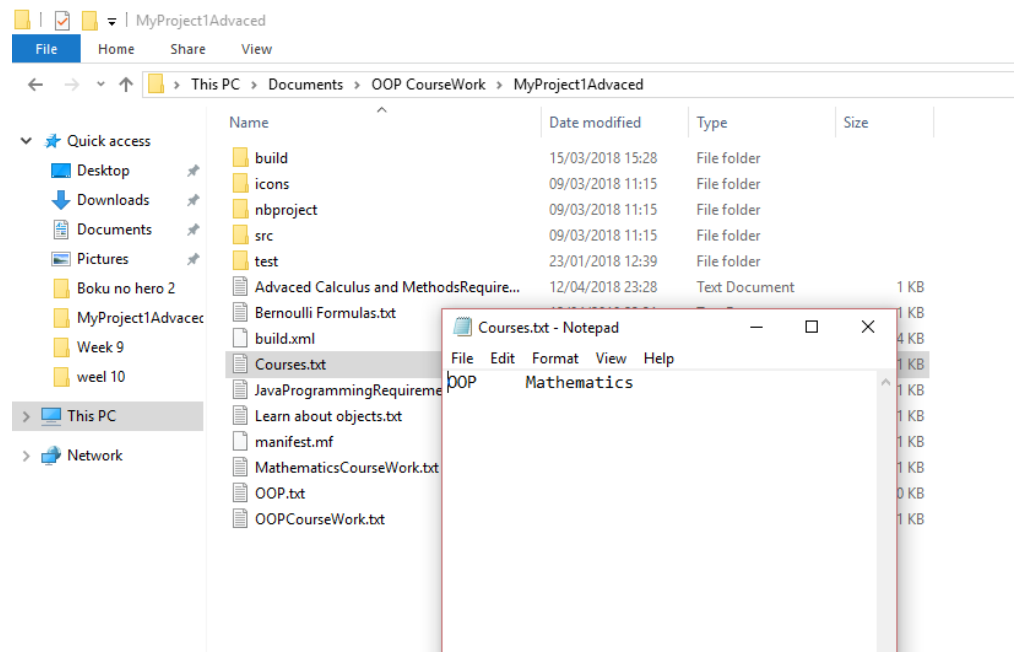
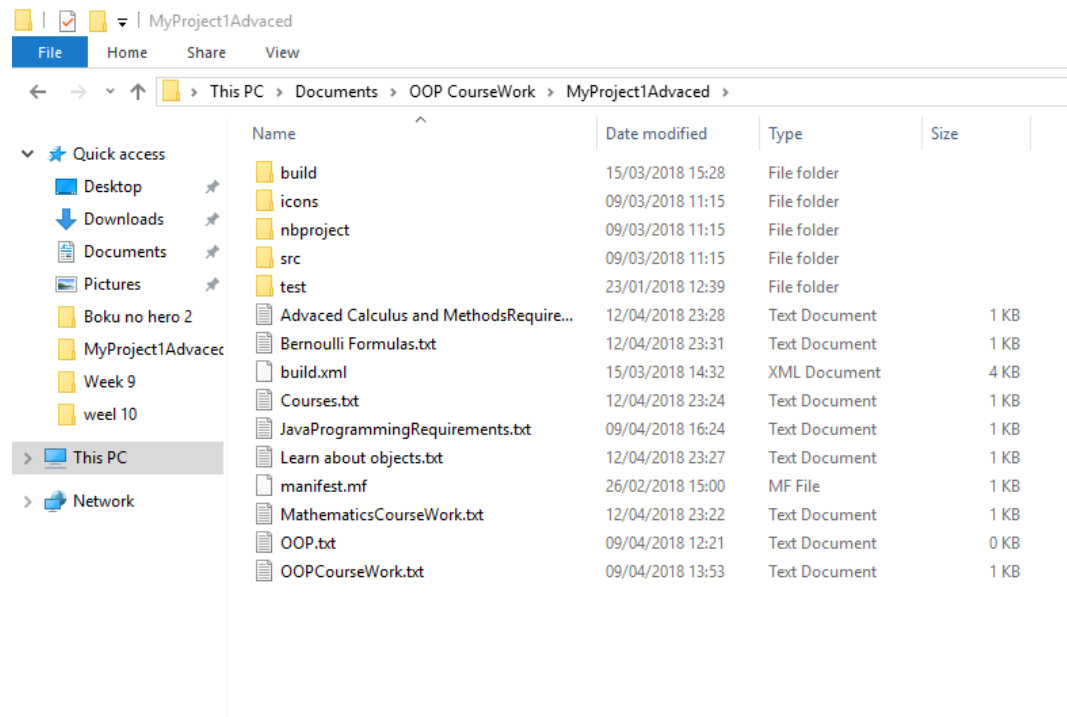
5.4 SCREEN 2 – A BRIEF DESCRIPTION

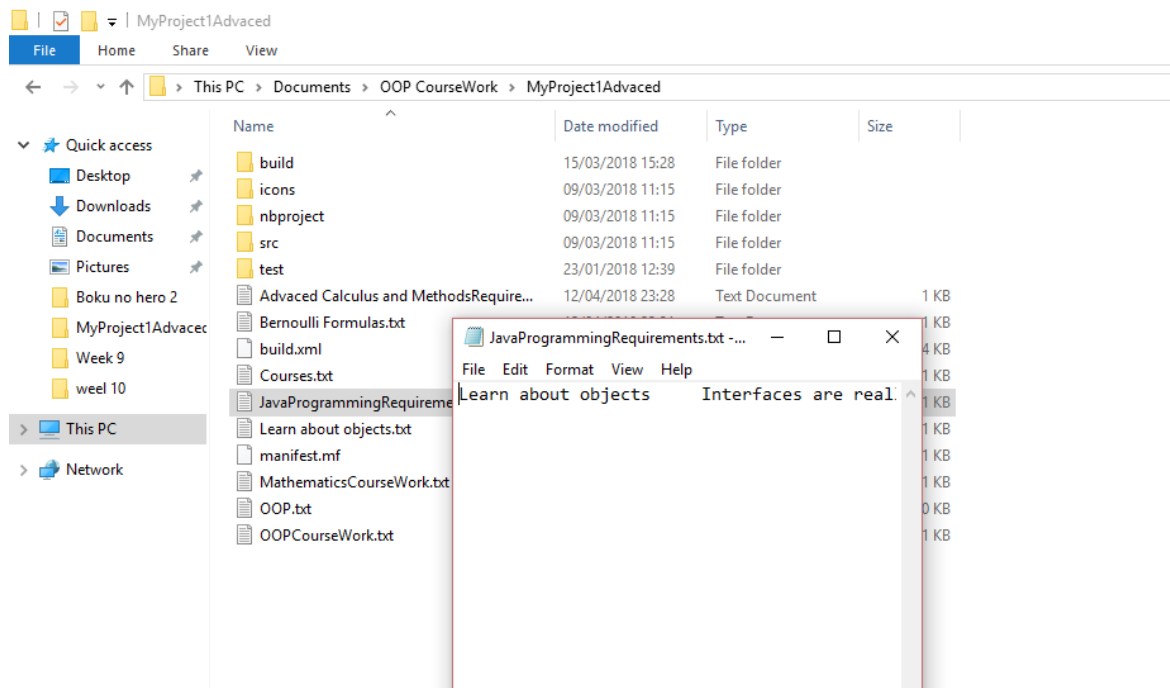
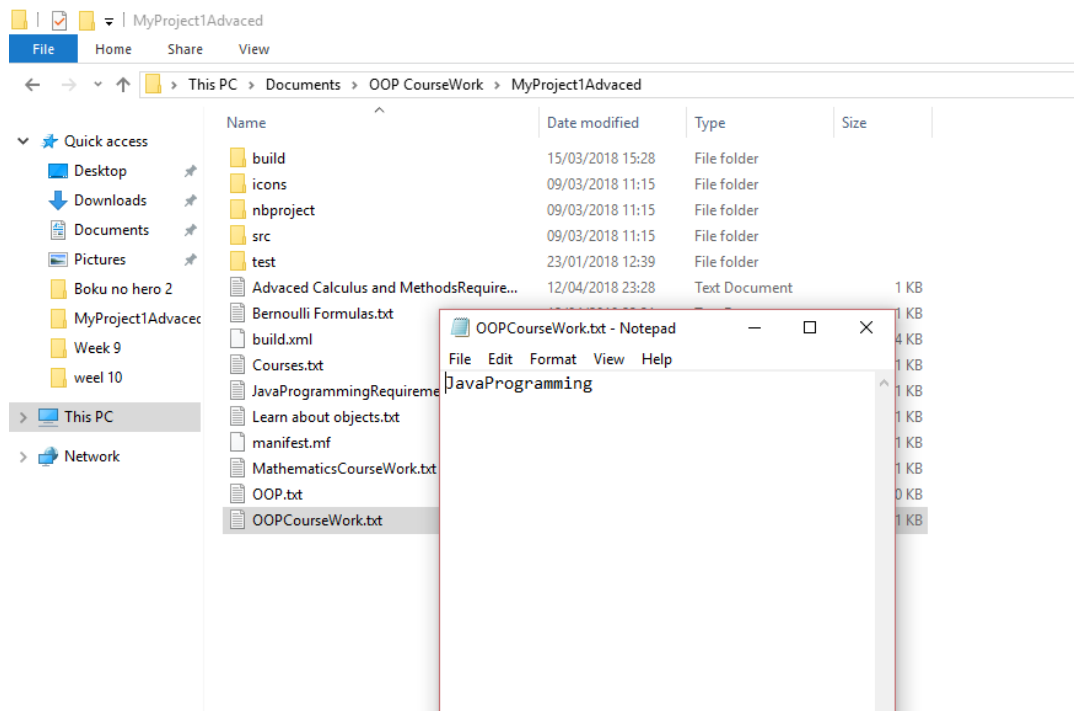
When we click onto the “Open coursework” option of the Coursework menu it opens a new window. We can see that every time we want to add a new requirement to the list of requirements we a dialog box pop up where we write the name.

We can also see how the notes are added to a specific requirement and every time we change a course the coursework list and requirements also changes.

And finally, when the button with the arrow pointing to the left is clicked on, it goes back to the main window.

5.5 SCREEN 3 - ...



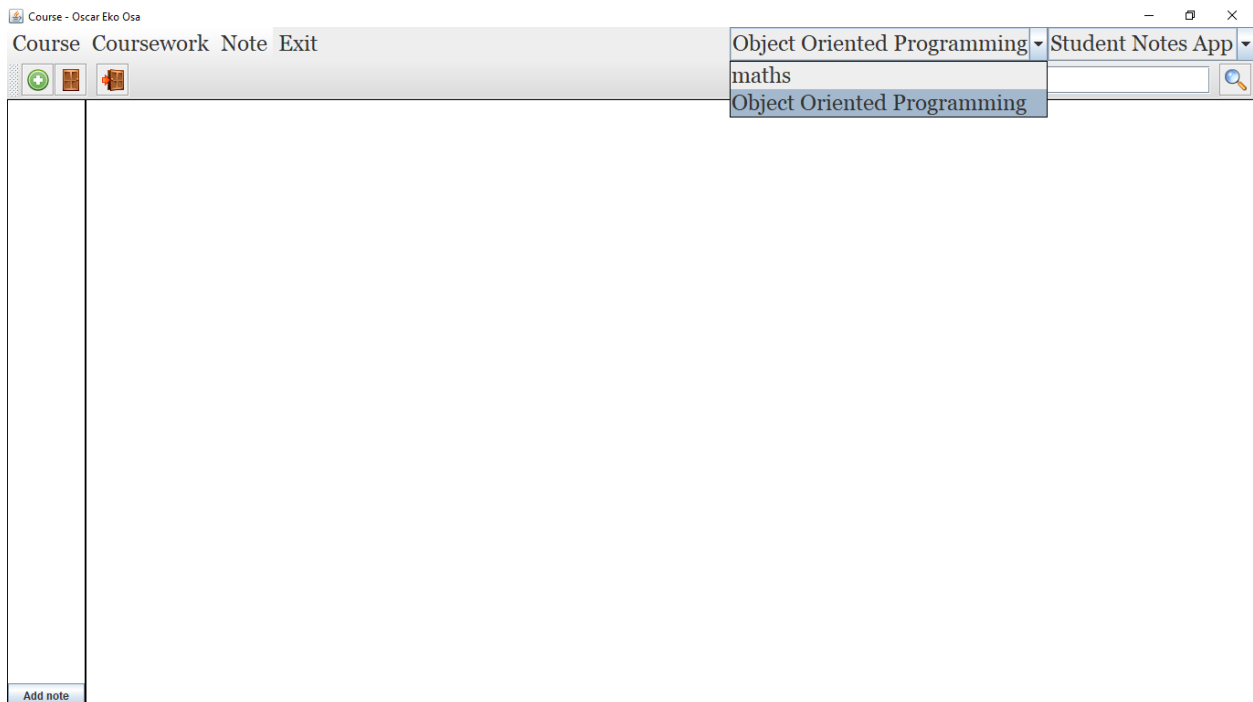




5.6 SCREEN 3 – A BRIEF DESCRIPTION

This shows how all the files for every course, coursework and requirements are created

5.7 SCREEN 4...



Course - Oscar Eko Osa

Course Coursework Note Exit

Object Oriented Programming Student Notes App

Add new course

Delete a course

Amend a course Delete an existing course

Add note

Course - Oscar Eko Osa

Course Coursework Note Exit

maths Calculus

maths

Add note

Course - Oscar Eko Osa

Course Coursework Note Exit

+

+

+

maths

Calculus

Calculus

Add note

Course - Oscar Eko Osa

Course Coursework Note Exit

+

+

+

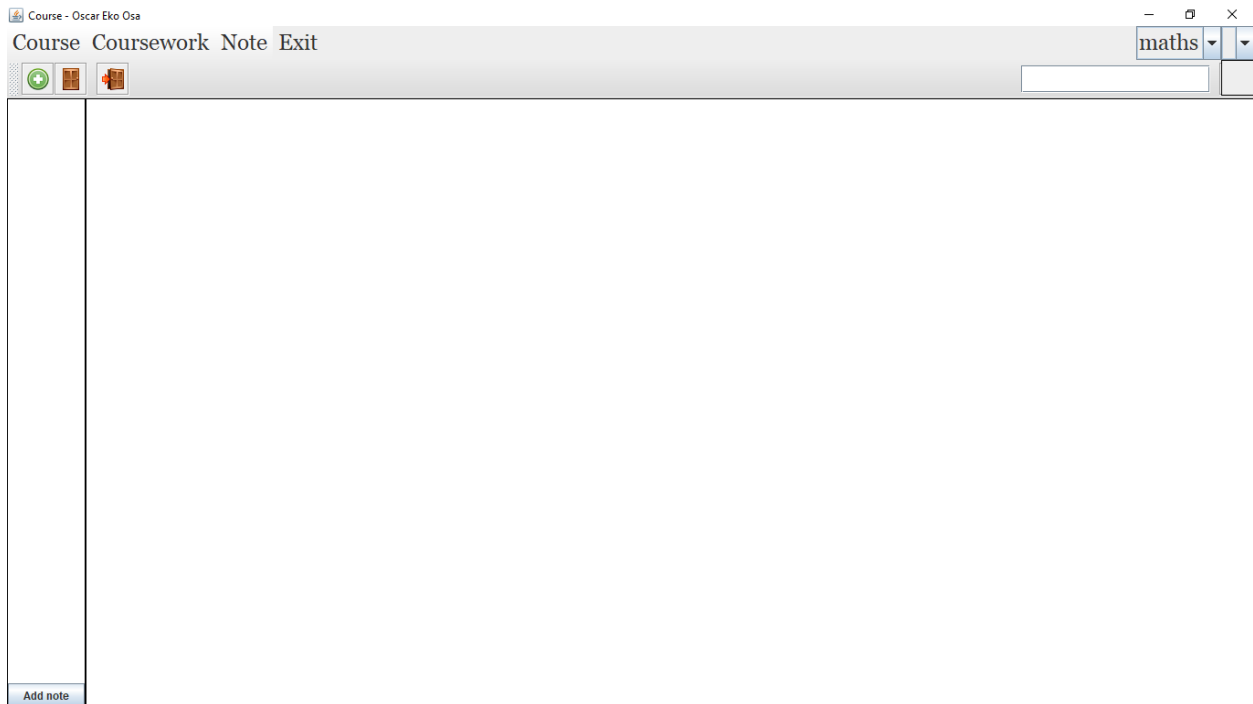
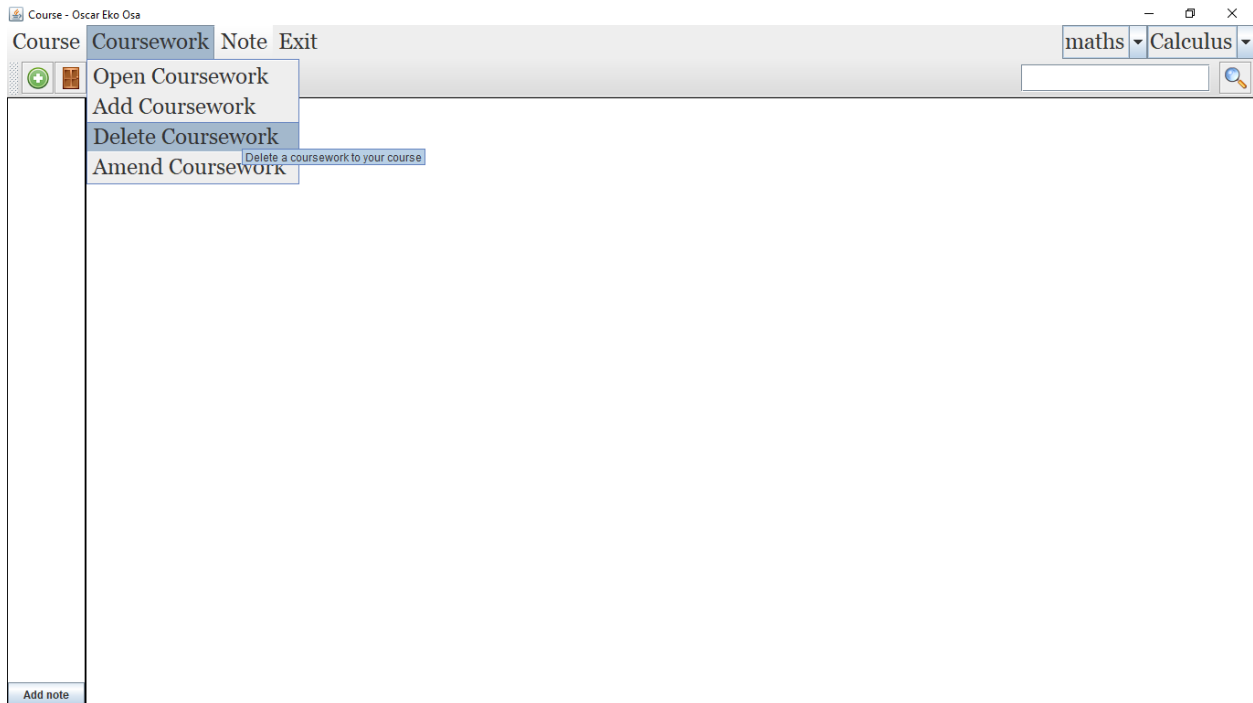
maths

Calculus

Calculus

Add note

Page 75 of 84



5.7 SCREEN 4 – A BRIEF DESCRIPTION

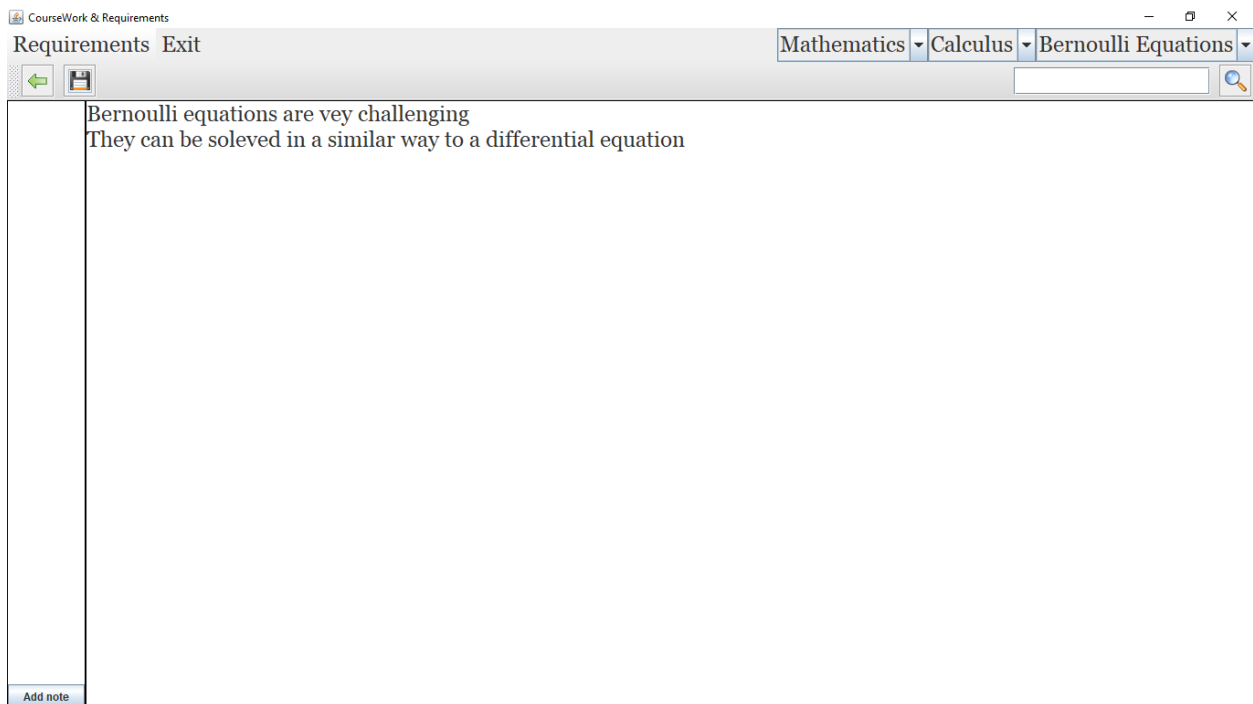
In the previous screenshot we can see how I implemented the option of deleting courses which is what happened in the case of the Object Oriented Programming course. When a whole course is

deleted we can see that its courseworks are also deleted and both course and coursework disappeared from the course and coursework drop-down lists.

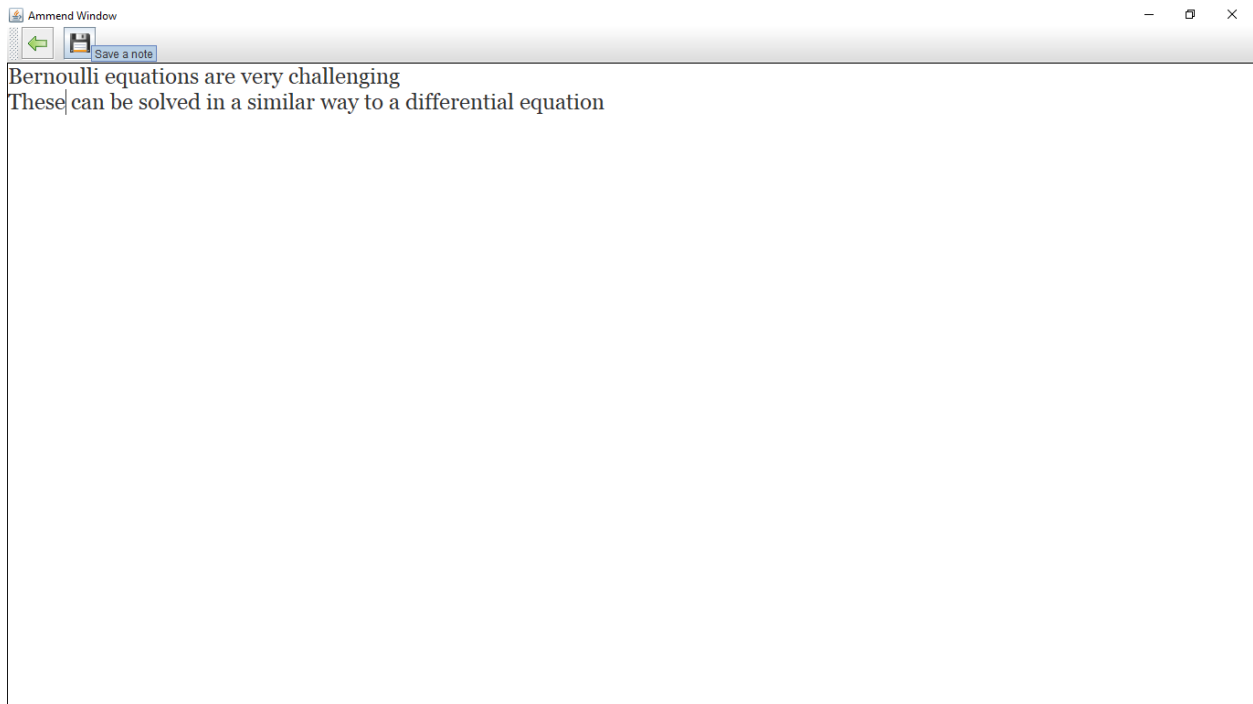
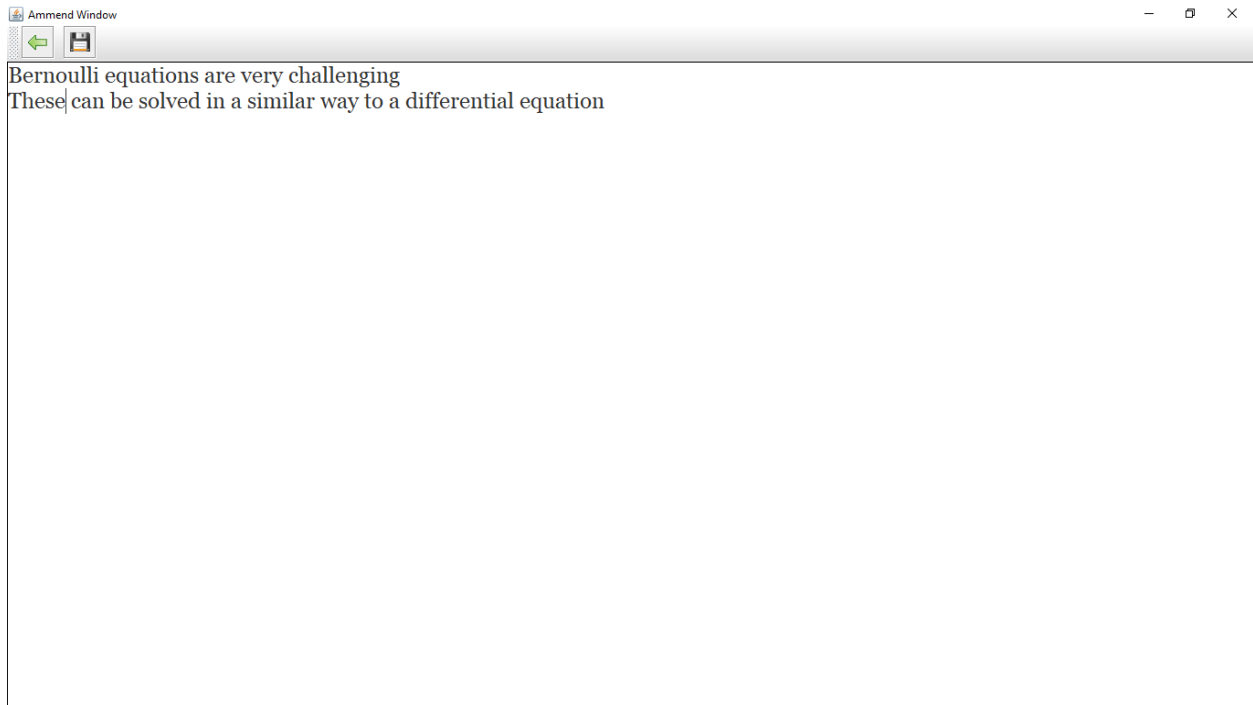
Or instead of that, we decide to only delete the course of a course which is what we did with the calculus coursework of the maths course.

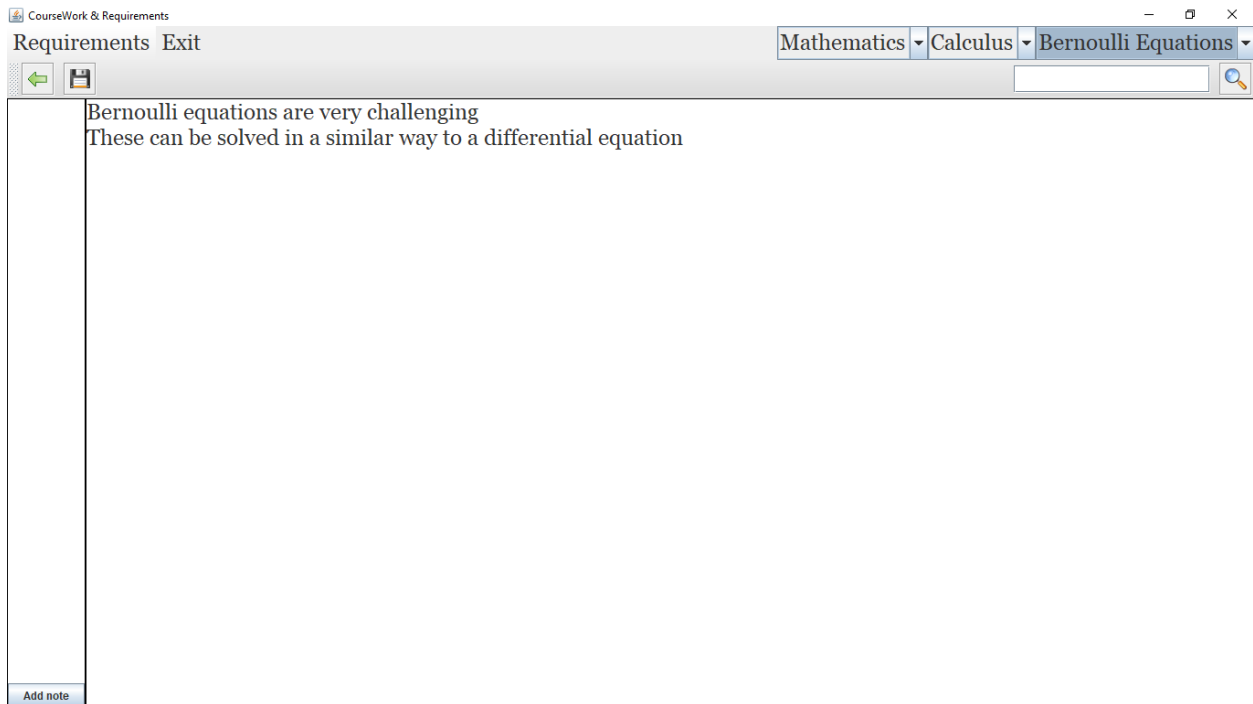
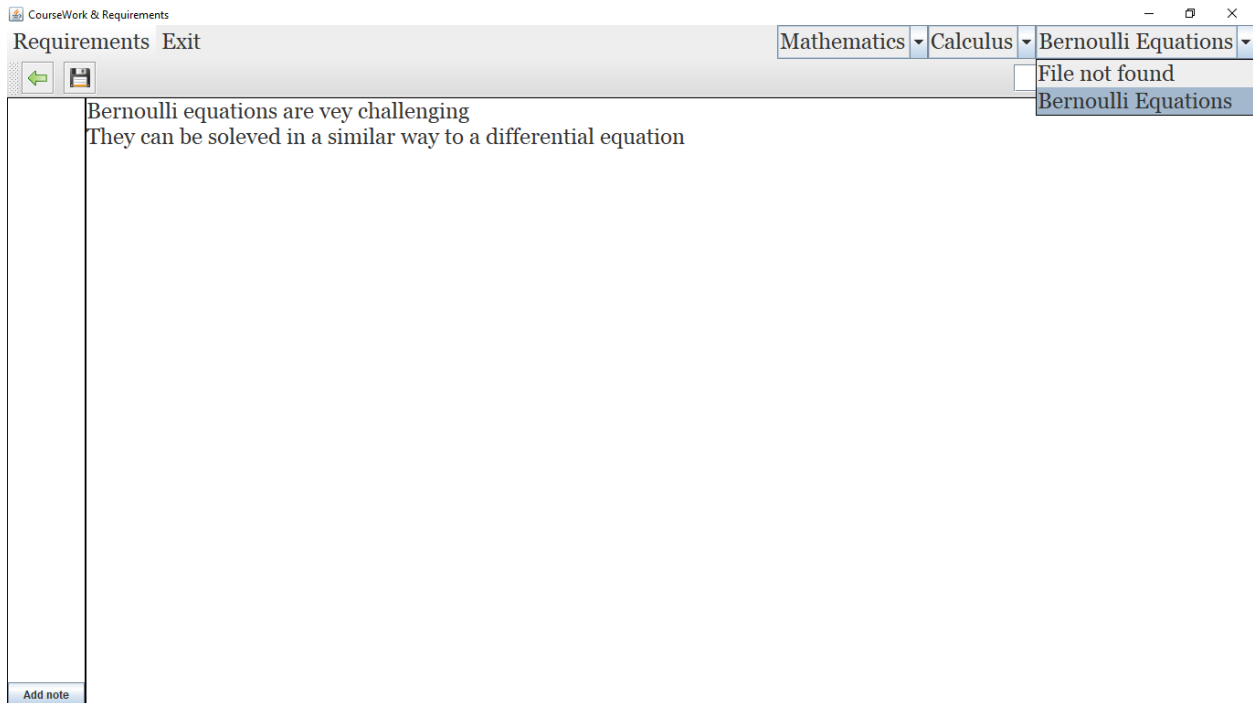
Since we can delete courses and courseworks I also allowed the user to be able to delete requirements from a coursework.

5.8 SCREEN 5...



Bernoulli equations are very challenging
They can be solved in a similar way to a differential equation





5.8 Screen 5- A Brief description

In these screenshots I am showing the *Ammend* class that I implemented using the *getData* interface. The amend option is an option that is give to the user to edit or change the notes that they took for their coursework requirements.

We can see how a new window containing the notes in the current requirements files are shown (the spelling mistakes were done in purpose to demonstrate this). We can see how the notes are also edited and then saved. After saving the edited notes, I went back to the coursework window and selected the same requirements. It can be observed how the notes are updated and the new and edited notes are displayed in the screen.

6 THE EVALUATION

Give a summary of the program and discuss what you would do if you had more time to work on the program. Answer the following questions for the reflection and write at least 400 words overall.

6.1 WHAT WENT WELL?

Most of the thing worked perfectly and most of the requirements were completed. I successfully achieved to add courses, courseworks and requirements to courseworks to a dropdown list of courses, courseworks and requirements. Which were all the requirements that we were asked to do for our Object-Oriented Programming course. As well as this as managed to create a file for every course, coursework and requirements that was added.

Although I didn't stop here. I wanted the program to look realistic, so I added a few new features; like the option to delete a course, a coursework and requirements. Furthermore, I also added the option to edit the notes that the student takes for his coursework. So that if any mistakes are made they could be corrected without having to delete the whole file and start again.

6.2 WHAT WENT LESS WELL?

I programmed it in my own way. So, my code is much different from the one that was given to us in the tutorials. Therefore, I found it more difficult to deal with the bugs that I was founding in my code and fix them due to the great difference with the code in comparison to the code of my peers.

Although I tried to get rid of most of the bugs some still remain in there.

1. The enumeration of the notes (the Id number) are not generated properly and sometimes do not follow the order they should be following.
2. When deleting my files, I found that is some occasions the methods coded to delete the files were not deleting them as some files were not closed properly after being accessed.
3. I was unable to delete the "File not found" string from lists dynamically. Although when the program is restarted again it disappears.

6.3 WHAT WAS LEARNED?

1. Organization is the key. And debugging and commenting are really helpful, especially when you have moved from one point and want to go back to have a look at, it helps remembering what certain parts of the code do. As well as good coding style.

6.4 HOW WOULD A SIMILAR TASK BE COMPLETED DIFFERENTLY?

Let people start from scratch their program so that we can do it in our own way. Since when code is given to you is helpful, but each programmer is unique a code in a different way.

6.5 HOW COULD THE COURSE BE IMPROVED?

Group and individual separate coursework would strengthen the programming and collaboration abilities of the students. They will also have to work with different programming styles.

7 SELF-GRADING

Please assess yourself objectively for each section shown below and then enter the total mark you expect to get.

Viva 1 (3)

- Not attended or no work demonstrated – 0
- Work demonstrated was not up to the standard expected – 1
- Work demonstrated was up to the standard expected – 2
- Work demonstrated exceeded the standard expected – 3

Viva 2 (3)

- Not attended or no work demonstrated – 0
- Work demonstrated was not up to the standard expected – 1
- Work demonstrated was up to the standard expected – 2
- Work demonstrated exceeded the standard expected – 3

Viva 3 (4)

- Not attended or no work demonstrated – 0
- Work demonstrated was not up to the standard expected – 1
- Work demonstrated was up to the standard expected – 2
- Work demonstrated exceeded the standard expected – 3 or 4

For this section I think I got : 6 out of 10
--

Professional programming style (10)

Coding (up to 5)

- Indenting has not been used – 0
- Indenting has been used occasionally – 1
- Indenting has been used, but not regularly – 2
- Indenting has been used regularly – 3 to 4
- All code has been indented correctly – 5

Naming of variables, procedure and classes (up to 5)

- Professional naming has not been used – 0
- Professional naming has been used occasionally – 1
- Professional naming has been used, but not regularly – 2
- Professional naming has been used regularly – 3 to 4
- All items have been named professionally correctly – 5

For this section I think I got : 8 out of 10
--

Use of OOP techniques (50)

Abstraction (20)

- No extra classes or objects have been created – 0
- Classes and objects have been created superficially – 1 to 7
- Classes and objects have been created and used correctly – 8 to 13
- New and useful classes and objects have been created – 14 to 17
- The use of classes and objects exceeds the specification – 18 to 20

Encapsulation (10)

- No encapsulation has been used – 0
- Class variables have been encapsulated superficially – 1 to 3
- Class variables have been encapsulated correctly – 4 to 6
- The use of encapsulation exceeds the specification – 7 to 10

Inheritance (10)

- No inheritance has been used – 0
- Classes have been inherited superficially – 1 to 3
- Classes have been inherited correctly – 4 to 6
- The use of inheritance exceeds the specification – 7 to 10

Polymorphism (10)

- No polymorphism has been used – 0
- A procedure has been polymorphised – 1 to 3
- A procedure has been polymorphised and used appropriately – 4 to 6
- The use of polymorphism exceeds the specification – 4 to 10

For this section I think I got : 43 out of 50

Testing (10)

- Testing has not been demonstrated in the documentation – 0
- Little white box testing has been documented – 1 to 3
- White box testing has been documented for all the coursework – 4 to 6
- White box testing has been documented for the whole program – 7

For this section I think I got : 0 out of 10

Evaluation (10)

- No evaluation was shown in the documentation – 0
- The evaluation shows a lack of thought – 1 to 3
- The evaluation shows thought – 4 or 5
- The evaluation shows clearly demonstrates increased awareness – 6 or 7

For this section I think I got : 7 out of 10

Documentation (10)

- The documentation cannot be understood on first reading – 0
- The documentation is readable, but a section(s) are missing – 1 to 3
- The documentation is complete – 4 to 6
- The documentation is complete and of a high standard – 7

For this section I think I got : 6 out of 10

I think my overall mark would be : 70 out of 100