

A · P · U

**ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION**

Module Code	:	CT038-3-2-ODJ
Module Title	:	ONLINE CONSULTATION HOURS BOOKING SYSTEM
Intake Code	:	UC2F1905SE
Lecturer Name	:	Lee Kim Keong
Hand out Date	:	29 May 2019
Hand in Date	:	29 July 2019
Tutorial No.	:	3

Student ID	Student Name
TP046057	Ong Shao An
TP045925	Ong Lip Yang

Table of Contents

1.0 Introduction.....	4
2.0 Use Case Diagram.....	5
3.0 Class Diagram	9
4.0 OOP Concept	10
4.1 Abstract	10
4.2 Inheritance.....	11
4.3 Encapsulation	12
4.4 Polymorphism	15
4.4.1 Overloading	15
4.4.2 Overriding.....	16
5.0 Code Snippet	20
5.0.1 Pre-requisite of software	20
5.1 GUI Code	21
5.1.1 Redirect Code between JFrames	21
5.1.2 Viewing Code in JTable / Input Fields.....	22
5.1.3 Creating a text file entry	23
5.1.4 Updating a text file entry	24
5.1.5 Deleting a text file entry	24
5.1.6 Searching an entry.....	25
5.1.7 Clearing text fields.....	26
5.1.8 Printing table results	26
5.2 Back End.....	27
5.2.1 filereader.java	27
5.2.2 Fileaccess.java	28
5.2.3 filewriter.java	29
5.2.4 textbinary.java	30
5.3 Class, Attributes and Method	31
5.3.1 User class	31
5.3.2 Profile Reader Class.....	33
5.3.3 Consultation Venue Class.....	35

5.3.4 Consultation Hour Class	39
5.3.5 Java Team class	47
6.0 Validation Explanation	48
6.0.1 Creating a consultation venue	48
6.0.2 Updating a consultation venue.....	48
6.0.4 Creating and editing a consultation hour	49
6.0.5 Cancelling a consultation hour	51
6.0.6 Deleting a consultation hour	52
6.0.7 Booking a consultation hour (Students).....	52
6.0.8 Cancelling a booked consultation hour (Students)	53
7.0 Text File Access.....	54
8.0 Interface & User Manual.....	55
8.1 Lecturer.....	56
8.2 Student	60
9.0 Additional Feature.....	63
9.1 Admin (Additional Feature)	63
9.2 Chat System	67
10.0 Conclusion	68
11.0 Workload Matrix	69
12.0 References	70

1.0 Introduction

In this assignment, we will be designing a software to allow students and lecturers to manage their consultation hours online. We will also need to develop the software based on object-oriented programming concepts to ensure the functionality of the system.

In addition, a supporting document will be created to reflect the design of the implementation codes and the implementation details that utilizes the object-oriented programming concepts which is the current documentation that you are reading.

Upon completion of this assignment, we should be able to design solutions for a software using object-oriented paradigm and translate it into software application that exploit the strength of object-oriented paradigm as well as demonstrating object-oriented concepts and their responsibilities in the existing system.

This software that is designed will be used by the administration, students and lecturers of a certain academic institution with functionalities pertaining to consultation hours booking for the students as well as lecturers. With this software, it is expected that the efficiency for booking consultation hours will be increased and thus eliminating traditional methods which are more time and energy consuming.

2.0 Use Case Diagram

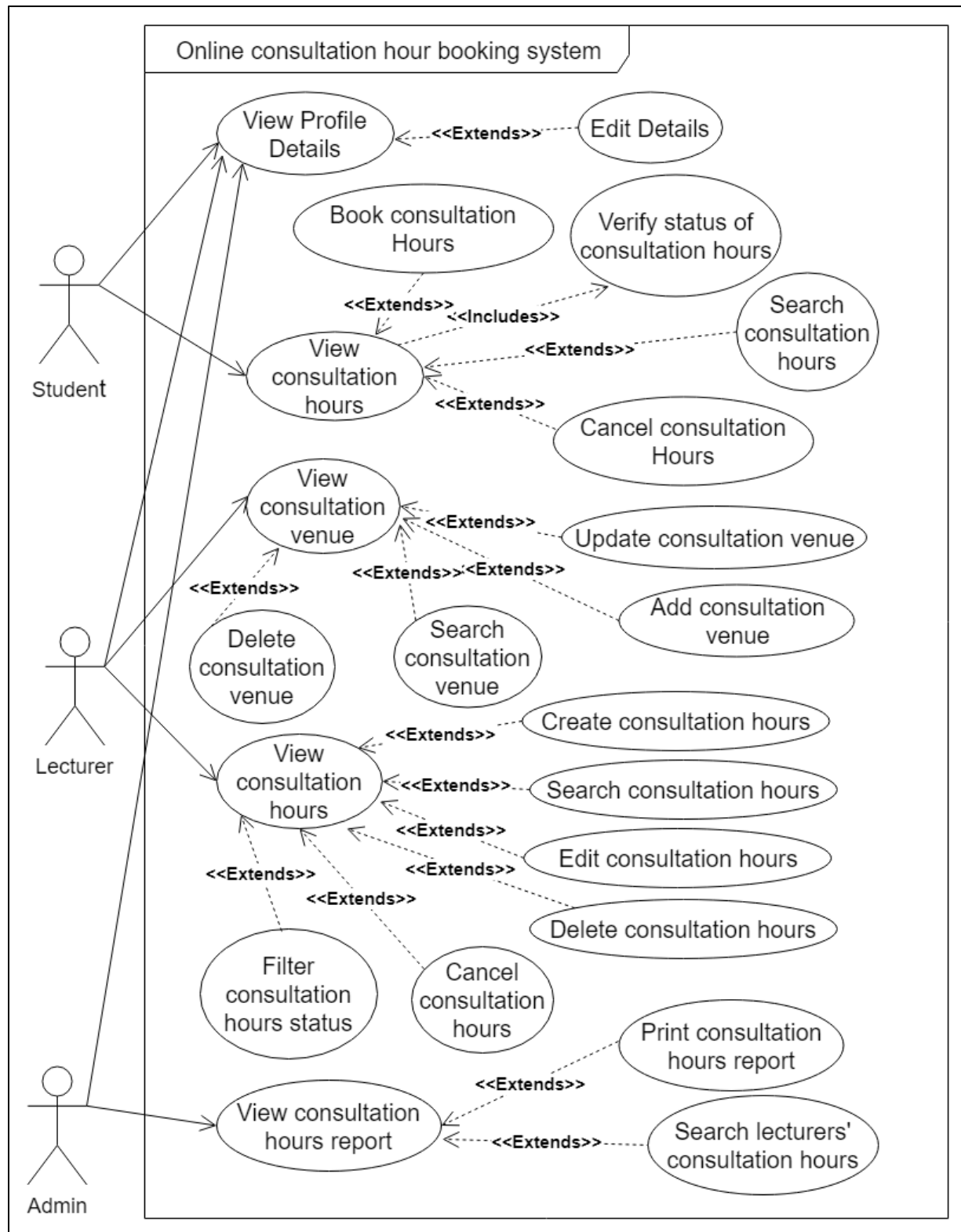


Figure 1: Use Case Diagram

Use Case Description: View Profile Details	
Name	View Profile Details
Description	View personal details
Actor(s) Perform	Lecturer, Admin, Student
Pre-condition	Student, Admin and Lecturer have their own account
Post-condition	View their own profile details
<<includes>>	None
<<extends>>	Edit Details

Use Case Description: Edit Details	
Name	Edit Details
Description	Edit personal details
Actor(s) Perform	Lecturer, Admin, Student
Pre-condition	Student, Admin and Lecturer have their own account
Post-condition	Update their own profile details
<<includes>>	None
<<extends>>	View Profile Details

Use Case Description: View Consultation Hours	
Name	View Consultation Hours
Description	View consultation Hours
Actor(s) Perform	Student
Pre-condition	When lecturer created consultation hours
Post-condition	Student can view the created consultation hours
<<includes>>	Verify status of consultation hours
<<extends>>	Book Consultation Hours, Search Consultation Hours and Cancel Consultation Hours
<<extends>> Functions	Student can book, search and cancel consultation hours

Use Case Description: Verify status of Consultation Hours	
Name	Verify status of Consultation Hours
Description	Verify status of consultation Hours whether is booked or canceled
Actor(s) Perform	Student
Pre-condition	When student view the consultation hours
Post-condition	Student can book and cancel based on the status
<<includes>>	None
<<extends>>	View Consultation Hours

Use Case Description: View Consultation Hours	
Name	View consultation hours
Description	View consultation booking hours that was created by the specific lecturer logged in.
Actor(s) Perform	Lecturer
Pre-condition	Lecturer can view updated consultation booking hours made by them only.
Post-condition	Lecturer can view created consultation booking hours
<<includes>>	None
<<extends>>	Create, Edit, Delete, Cancel, Filter and Search created consultation booking hours.
<<extends>> Functions	Lecturer can create, edit, delete, cancel, filter and search created consultation booking hours.

Use Case Description: View Consultation Venue	
Name	View consultation venue
Description	View consultation venue that is created by all lecturers.
Actor(s) Perform	Lecturer
Pre-condition	Lecturer can view created venue
Post-condition	Lecturer can view updated created venue.
<<includes>>	None
<<extends>>	Update, Delete, Search , Add consultation hours
<<extends>> Functions	Lecturer can update, delete, search and add consultation hours

Use Case Description: View Consultation Hours Report	
Name	View consultation hours report
Description	View consultation booking hours that was created by all the lecturers, as well as statuses and students assigned to the booking hours.
Actor(s) Perform	Admin
Pre-condition	Admin can print and search consultation hours report.
Post-condition	Admin can view search results of consultation hours report, print that specific search results.
<<includes>>	None
<<extends>>	Print consultation hours report, search lecturers' consultation hours.
<<extends>> Functions	Admin can print consultation hours report and search lecturers' consultation hours.

3.0 Class Diagram

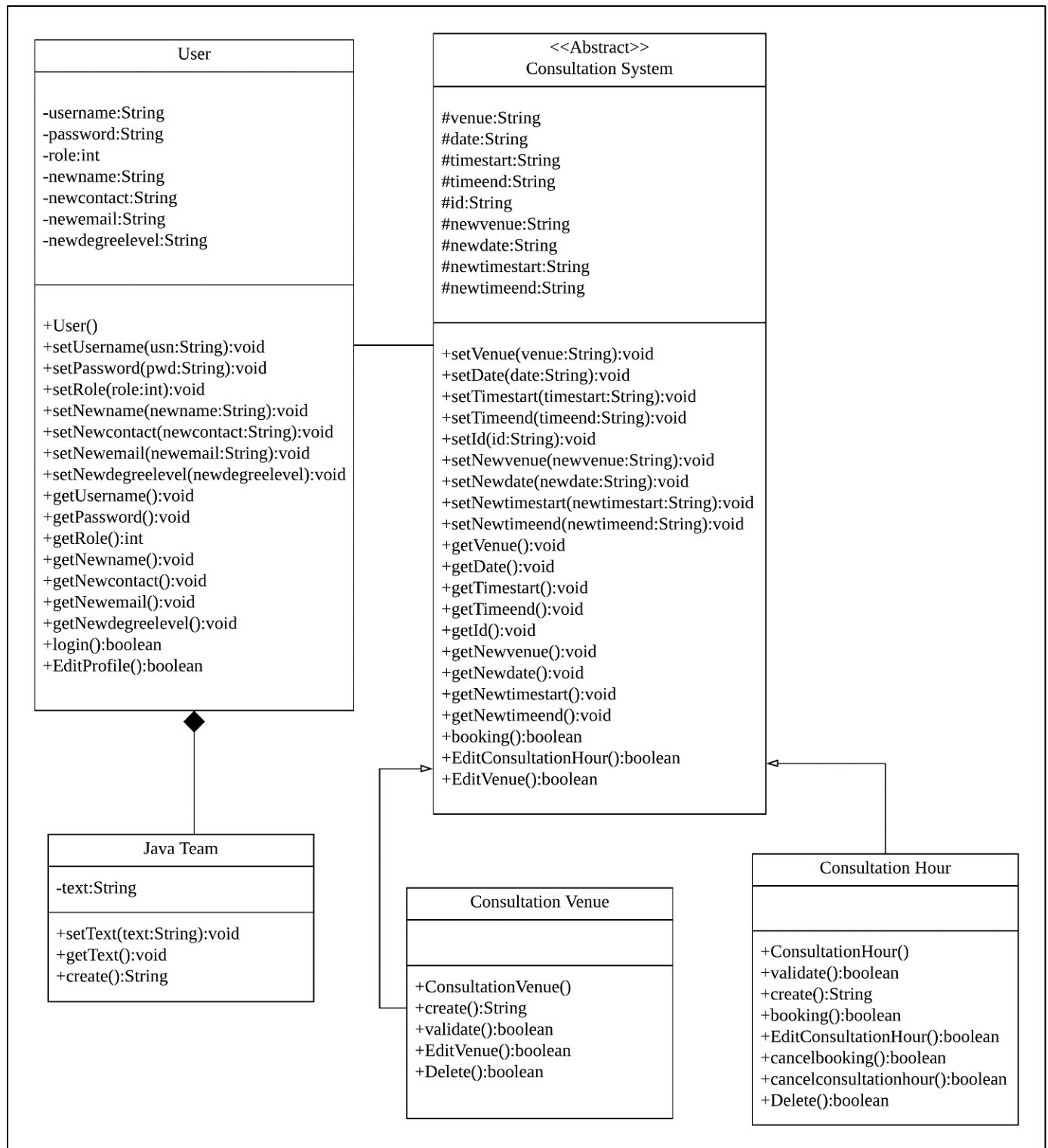


Figure 2: Class Diagram

4.0 OOP Concept

4.1 Abstract

An abstract class, in the context of Java, it is a superclass that cannot be instantiated and is used to state or define all general characteristics. An object cannot be created from a Java abstract class; trying to instantiate an abstract class only produces a compiler error. The abstract class is declared using the keyword `abstract`. Subclasses extended which is inheritance from an abstract class have all the abstract class's attributes, in addition to attributes specific to each subclass. The abstract class states the class characteristics and methods for implementation, thus defining a whole interface. (Janssen, 2008) Picture below is an example of Abstract class for Consultation System class.

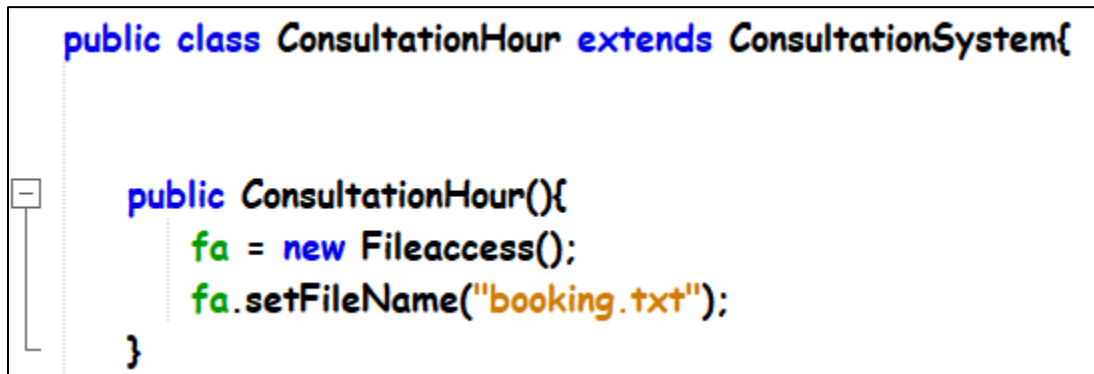
```
public abstract class ConsultationSystem {  
  
    protected FileAccess fa;  
    protected String venue, date, timestart, timeend;  
    protected String id;  
  
    protected String newvenue, newdate, newtimestart, newtimeend;  
}
```

Figure 3: Abstract Class

Venue, date, timestart, timeend, id, newvenue, newdate, newtimestart and newtimeend are protected strings which means they are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class. The protected access modifier cannot be applied to class and interfaces. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

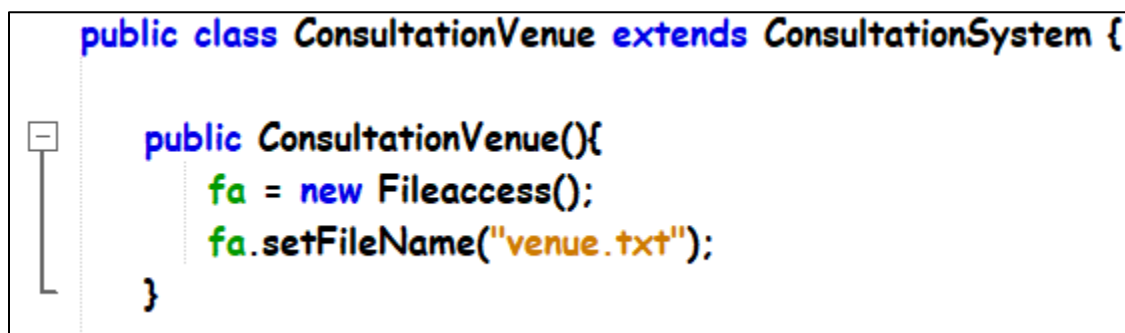
4.2 Inheritance

Inheritance supports the concept of “reusability” for example, when we want to create a new class and there is already a class that exists some of the codes that we want to define again, so we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.



```
public class ConsultationHour extends ConsultationSystem{  
  
    public ConsultationHour(){  
        fa = new Fileaccess();  
        fa.setFileName("booking.txt");  
    }  
}
```

Figure 4: Inheritance



```
public class ConsultationVenue extends ConsultationSystem {  
  
    public ConsultationVenue(){  
        fa = new Fileaccess();  
        fa.setFileName("venue.txt");  
    }  
}
```

Figure 5: Inheritance

Sub Class (Consultation Hour and Consultation Venue) both connect from the super class Consultation System. Both consultation hour and consultation venue share some same attributes and some methods from the super class. (Anon., 2009)

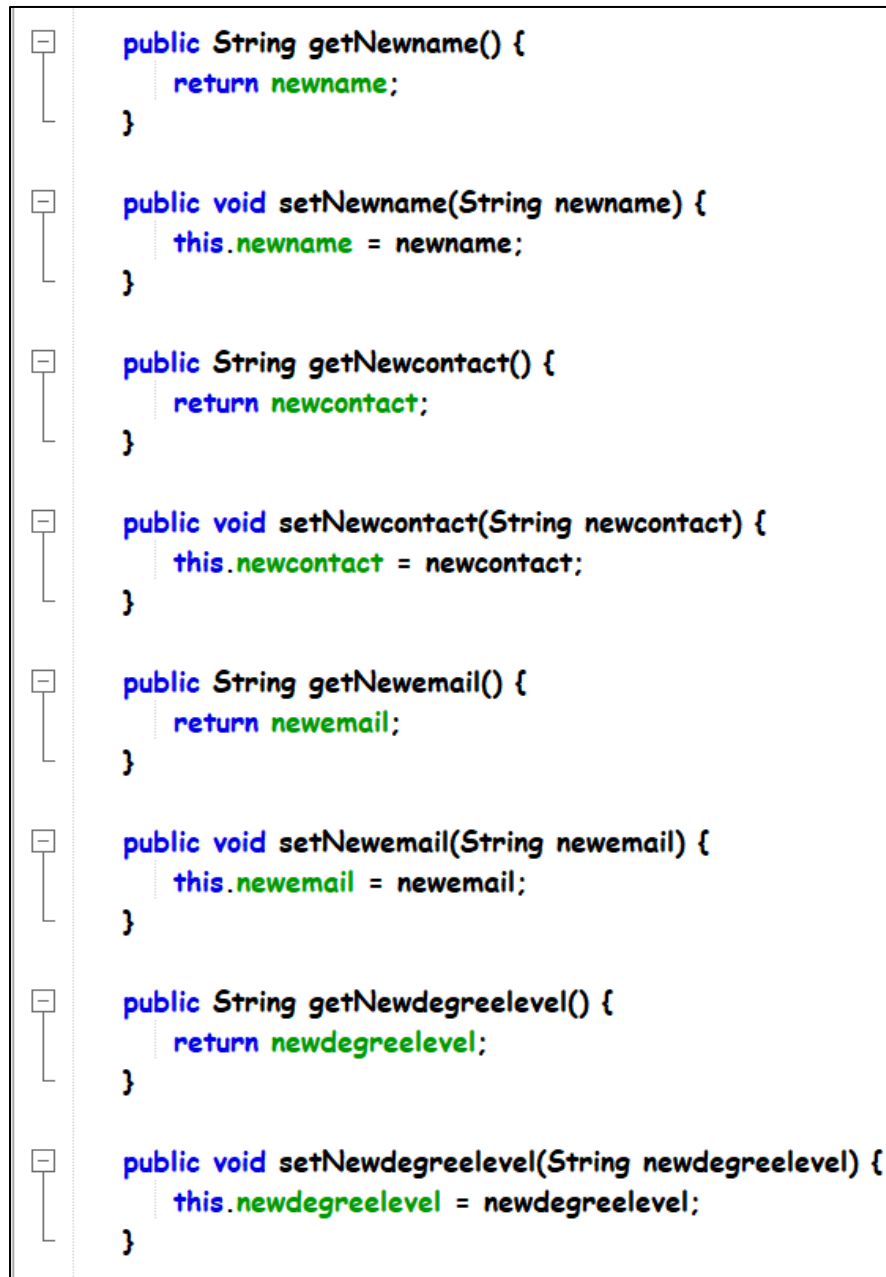
4.3 Encapsulation

Encapsulation in Java is mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

To achieve encapsulation in Java 1st, declare the variables of a class as private. Then, provide public setter and getter methods to modify and view the variables values.

```
public class User {  
  
    private Fileaccess fa;  
    private String username,password;  
    private int role;  
  
    private String newname, newcontact, newemail, newdegreelevel;  
}
```

Figure 6 Encapsulation



```
public String getNewname() {  
    return newname;  
}  
  
public void setNewname(String newname) {  
    this.newname = newname;  
}  
  
public String getNewcontact() {  
    return newcontact;  
}  
  
public void setNewcontact(String newcontact) {  
    this.newcontact = newcontact;  
}  
  
public String getNewemail() {  
    return newemail;  
}  
  
public void setNewemail(String newemail) {  
    this.newemail = newemail;  
}  
  
public String getNewdegreelevel() {  
    return newdegreelevel;  
}  
  
public void setNewdegreelevel(String newdegreelevel) {  
    this.newdegreelevel = newdegreelevel;  
}
```

Figure 7: Encapsulation

The public setXXX() and getXXX() methods are the access points of the instance variables of the User class. Normally, these methods are referred as getters and setters. If any class that wants to access the variables should access them through these getters and setters.

```
User u = new User();  
User.currentUser = txtUsername.getText();  
u.setNewname(txtName.getText());  
u.setNewcontact(txtContactNumber.getText());  
u.setNewemail(txtEmailAddress.getText());  
u.setNewdegreelevel(txtDegreeLevel.getText());
```




Figure 8: Encapsulation

```
txtName.setText(prObj.getName());  
txtDegreeLevel.setText(prObj.getLevel());  
txtContactNumber.setText(prObj.getContact());  
txtEmailAddress.setText(prObj.getEmail());
```




Figure 9: Encapsulation

Picture above shown that all the variables access them through these getters and setters.
(Anon., 2019)

4.4 Polymorphism

4.4.1 Overloading

Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists. (JavaTpoint, 2011)

Invalid case of method overloading is that argument list, for example if two methods have the same name, same parameters and have different return type, then this is not a proper valid method overloading example and this will throw a compilation error. Example like `int add(int, int)` and `int add(int, int)`.

```
public User(String username, String password, int role, String newname, String newcontact, String newemail, String newdegreelevel) {  
    this.username = username;  
    this.password = password;  
    this.role = role;  
    this.newname = newname;  
    this.newcontact = newcontact;  
    this.newemail = newemail;  
    this.newdegreelevel = newdegreelevel;  
}  
  
public User(){  
    fa = new Fileaccess();  
    fa.setFileName("user.txt");  
}
```

Figure 10: Overloading

The figure above shows the proper overloading method, they both do not have the same parameters, for the 1st constructor, it can be used in the registration function, but there is no registration function in this assignment, so it can be used as a reference for future improvement. The 2nd constructor is to create a file access object to access a specific text file. (SINGH, 2014)

4.4.2 Overriding

The benefit of overriding is that ability to define a behavior that's specific to the subclass type, which means a subclass can implement a super class method based on its own requirement. In object-oriented concepts, overriding means to override the functionality of an existing method.

Rules for Method Overriding is that the argument list should be exactly same as that of the overridden method. The return type should be same, or a subtype of the return type declared in the original overridden method in the superclass. The access level cannot be more restrictive than the overridden method's access level. For example: If the superclass method is declared public then the overriding method in the sub class cannot be either private or protected. (Anon., 2019)

Next, instance methods can be overridden only if they are inherited by the subclass. A method declared final cannot be overridden, a static cannot be overridden but can be re-declared. If a method cannot be inherited, then it cannot be overridden. A subclass within the same package as the instance's superclass can override any superclass method that is not declared private. A subclass in a different package can only override the non-final methods declared public or protected. An overriding method can throw any unchecked exceptions, regardless of whether the overridden method throws the exceptions or not. However, the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method. Constructors cannot be overridden. (Anon., 2019)


```

public boolean booking(){
    JOptionPane.showMessageDialog(null, "Fail to book");
    return false;
}

public boolean EditConsultationHour(){
    JOptionPane.showMessageDialog(null, "Update Fail");
    return false;
}

public boolean EditVenue() {
    JOptionPane.showMessageDialog(null, "Fail to Edit");
    return false;
}

```

Figure 11: Overriding

The picture above shown that there are some methods are used inside the Super class which is the Consultation System's abstract class.

```

@Override
public boolean EditConsultationHour() {
    File filename = new File("booking.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("/");
        if (split[7].equals(id)) {
            String newRecordLine = String.join("/", newvenue, newdate, newtimestart, newtimeend, split[4], split[5], split[6], split[7]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }else{
        return false;
    }
}

```

Figure 12: Overriding

The override notation is shown out because the method is being used again in the inheritances class which is child class to perform its own requirements.

```
@Override
public boolean booking() {
    File filename = new File("booking.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("/");
        if (split[7].equals(id)) {

            String newRecordLine = String.join("/", split[0], split[1], split[2], split[3], "Booked", User.currentUser, split[6], split[7]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return true;
}else{
    return false;
}
}
```

Figure 13: Overriding

The override notation is shown out because the method is being used again in the inheritances class which is child class to perform its own requirements. It is same with the previous example.

```
@Override
public boolean EditVenue() {
    File filename = new File("venue.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("//");

        if (split[1].equals(id)) {
            String newRecordLine = String.join("//", newvenue, split[1]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }else{
        return false;
    }
}
```

Figure 14: Overriding

The override notation is shown out because the method is being used again in the inheritances class which is child class to perform its own requirements. It is the same with the previous example.

5.0 Code Snippet

5.0.1 Pre-requisite of software

This section will explain all the codes that are used in the assignment, which are included in the software. Before proceeding, we would like to clarify the usage of an external JAR file, which is the LGoodDatePicker, in which the documentation referencing for this component usage can be found in the reference list below. (BlakeTNC, 2018)

The instructions to load the JAR file for the project to load properly are as follows; When first opening the project folder, NetBeans IDE will prompt an error message stating that a component used in the project could not be found and would request to locate the said component. Click on “Resolve” and locate the JAR file under the folder “Consultation”. Please note that the project would not load correctly should the component is not located properly.

This add-on element behaves like a calendar and time picker that easily allows user to choose the date and time without trouble. The elements obtained are time and date that are readable and convertible according to Java’s programming standards.

5.1 GUI Code

This section will emphasize on the codes of files found under the folder “GUI”. Validation codes will be briefly touched and will be discussed in-depth in the following section.

5.1.1 Redirect Code between JFrames

```
Report rpt = new Report();  
rpt.setLocationRelativeTo(null);  
rpt.setVisible(true);  
this.dispose();
```

Figure 15: Code to redirect Jframes

The first line of the code is declaring a variable for the JFrame Form that is intended to be opened, and in this case the variable is “rpt”. Following that, the next line of code is to set the behavior of the form so that it stays constant at the center of the user’s display regardless of the resolution. The third line is to make the form visible, and the fourth line is to dispose of the current form that is opened, which results to only the new form being visible. (Prasad, 2014)

This code concept is used where users need to go back to the previous page, or to the main menu. Buttons that commonly have this code are “Back”, “Menu”, “Log Out”, and so on.

5.1.2 Viewing Code in jTable / Input Fields

```
File file = new File("booking.txt");

try
{
    FileReader fr = new FileReader(file);
    BufferedReader br = new BufferedReader(fr);
    DefaultTableModel model = (DefaultTableModel)tableBooking.getModel();
    model.setRowCount(0);
    Object[] lines2 = br.lines().toArray();

    for(int i=0; i< lines2.length; i++)
    {
        String[] row = lines2[i].toString().split("/");
        if(row[6].equals(User.currentUser)){
            model.addRow(row);
        }
    }
    br.close();
}
catch(IOException Ex){}
```

Figure 16: Code for viewing data

The file name to retrieve the data was first declared into a variable called “file”, and then a try catch statement was created. In the try statement, a FileReader and a BufferedReader is declared. The table model which is “tableBooking” in this case is also declared, and the row count of the table is set to “0” to prevent any unwanted empty lines when displaying the data. (user586399, 2013)

An object was created to store the array, and a loop was created to constantly print out the arrays and split them into different columns with “/” as the identifier with the condition that the user is equivalent to the user that is logged in.

This code concept is applied with slight variance where data is required to be read from the text files.

5.1.3 Creating a text file entry

```
String venue = txtVenue.getSelectedItem().toString();
String date = txtDate.getDateStringOrSuppliedString("(null)");
String timestart = txtTimeStart.getTimeStringOrSuppliedString("(null)");
String timeend = txtTimeEnd.getTimeStringOrSuppliedString("(null)");

ConsultationHour ch = new ConsultationHour();
ch.setVenue(venue);
ch.setDate(date);
ch.setTimeStart(timestart);
ch.setTimeEnd(timeend);

//Creation method starts here.
if(ch.create() != null){
    model.addRow(new String[]{venue, date, timestart, timeend,
        "Available", "None", User.currentUser, ch.getId() });
    JOptionPane.showMessageDialog(rootPane, "Insert Successfully");
}else{
    JOptionPane.showMessageDialog(rootPane, "Technical error encountered.");
}
```

Figure 17: Code for creating text file entry

The input variables were declared and converted to string type using each of their own conversion syntaxes (JodaStephen, 2015). The file in the “module” folder was called and the input variables that were declared was passed on to the file using the encapsulation method. When the file is created successfully, the table will add the newly inserted row, and a pop-up with the success message is shown. If the create success fails, an error message is shown. The create function declared is determined by a Boolean variable.

This code concept is applied to all areas in the software such as adding a new venue, or a new consultation hour with slight variations.

5.1.4 Updating a text file entry

```
ConsultationHour ch = new ConsultationHour();

User.currentUser = txtUsername.getText();
ch.setId(tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 7).toString());
ch.setNewvenue(txtVenue.getSelectedText().toString());
ch.setNewdate(txtDate.getDate().toString());
ch.setNewtimestart(txtTimeStart.getTime().toString());
ch.setNewtimeend(txtTimeEnd.getTime().toString());

if(ch.EditConsultationHour()){
    JOptionPane.showMessageDialog(rootPane, "Edit Successfully");
    readfile();
}else{
    JOptionPane.showMessageDialog(rootPane, "Fail to Edit");
}
```

Figure 18: Code for updating text file entry

This code has similarities to creating a new text file entry, but the difference is in the method of obtaining the data.

This code concept is applied to all areas in the software where there are updating methods used such as changing the status of a consultation hour, updating a consultation venue and others with slight variations in the code concept.

5.1.5 Deleting a text file entry

```
ConsultationHour con = new ConsultationHour();

con.setId(tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 7).toString());
if (con.Delete()){
    JOptionPane.showMessageDialog(rootPane, "Deleted");
}else {
    JOptionPane.showMessageDialog(rootPane, "Fail to Delete");
}
readfile();
```

Figure 19: Code for deleting text file entry

The file in the “module” folder is declared, and then the JTable model is also declared. The entry to delete is determined by the selected row and converted to string. When passed through the delete function stated in the module file, it deletes the entry.

This code concept is applied to all areas in the software where an entry is deleted such as deleting a venue or an available consultation hour.

5.1.6 Searching an entry

```
LocalDate Search = txtDate.getDate();
String name = txtUsername.getText();

File file = new File("booking.txt");

if(Search == null){
    JOptionPane.showMessageDialog(rootPane, "Please select a Date to search");
    readfile();
    return;
}else{

    try
    {
        FileReader fr = new FileReader(file);
        BufferedReader br = new BufferedReader(fr);
        DefaultTableModel model = (DefaultTableModel)tableBooking.getModel();
        model.setRowCount(0);
        Object[] lines2 = br.lines().toArray();

        for(int i=0; i< lines2.length; i++)
        {
            String[] row = lines2[i].toString().split("/");
            if(row[1].contains(Search.toString()) && row[6].contains(name)){
                model.addRow(row);
            }
        }
        br.close();
    }
    catch(IOException Ex){}
}
```

Figure 20: Code for searching an entry in a text file

The input fields were obtained and declared into variables, along with the file name that the data is supposed to be stored in. A if else function was created to prevent logical errors. The first IF condition ensures that the date filter is not left blank. The else function then reads the data from the text file which the code looks similar with the code in section 5.1.2, but with the variations the variable must match for the data to be displayed.

This code concept is applied in all areas of the software where lecturers can search for venues that were created, and users can search for consultation hours that were listed.

5.1.7 Clearing text fields

```
txtUsername.setText("");  
txtPassword.setText("");
```

Figure 21: Code for clearing text fields

This code snippet clears the input fields. It is applied in the login menu where when users click on the “Reset” button clears all the login credentials.

5.1.8 Printing table results

```
try  
{  
    tableReport.print();  
}  
catch (Exception e)  
{}
```

Figure 22: Code for printing table results

This code snippet opens a pop-up where users can print the data that is displayed in the JTable. It is used in the admin function where they can print all consultation hours for reference purposes.

5.2 Back End

This section will explain all the codes that are present in the “backend” folder, which are Fileaccess.java, textbinary.java, filereader.java and filewriter.java.

5.2.1 filereader.java

```
public abstract class filereader {  
    protected File filename;  
  
    public void setFileName(String fn) {  
        this.filename = new File(fn);  
    }  
  
    public abstract ArrayList<String> readAll();  
  
    public abstract boolean write(String record);  
}
```

Figure 23: Code of filereader.java

This code is used to pass a filename and let the filename remain protected and avoided to be publicly declared. The obtained filename is only then passed on to a setter and declared.

The first public abstract ArrayList<String> is to read all the data in a text file and store them as strings, whereas the next one is where the declared record variable which contains all the data is written to the text file.

5.2.2 Fileaccess.java

```
public class Fileaccess extends FileReader{  
    //Polymorphism:: Method overriding  
    @Override  
    public ArrayList<String> readAll() {  
        ArrayList<String> list = new ArrayList();  
        //Scanner/BufferedReader  
        try (Scanner scan = new Scanner(this.filename)) {  
            //reading  
            while (scan.hasNextLine()) {  
                String line = scan.nextLine();  
                //populate it into list object  
                list.add(line);  
            }  
            scan.close();  
        } catch (IOException ex) {  
            //any exception/error happen  
            ex.printStackTrace();  
        }  
        return list;  
    }  
}
```

Figure 24: Part 1 of Fileaccess.java

```
@Override  
public boolean write(String record) {  
    //PrintWriter  
    try (PrintWriter out = new PrintWriter(new FileWriter(this.filename, true))) {  
        out.println(record); //write to file  
        //if 'write' is succeeded  
        out.close();  
        return true;  
    } catch (IOException ex) {  
        //any exception/error  
        System.out.println("Error: Failed to write.");  
        return false;  
    }  
}  
  
public boolean write(ArrayList<String> data) {  
    throw new UnsupportedOperationException("Not supported yet."); //To change body  
}
```

Figure 25: Part 2 of Fileaccess.java

This file extends to the previous file, which is `filereader.java` that was explained in section 5.1.1. The overriding method of polymorphism was used to declare a new `readAll` method and read the data of the file that was protected and passed on to this child class.

The overriding of the write file looks at writing the declared string “record” to the filename that was passed on from the parent class. Both these methods can be called from the GUI or modules folder classes.

5.2.3 `filewriter.java`

```
public class filewriter extends filereader{
    @Override
    public ArrayList<String> readAll() {
        ArrayList<String> data = null;
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(this.filename))) {
            data = (ArrayList<String>) in.readObject();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            return data;
        }
    }
}
```

Figure 26: Part 1 of `filewriter.java`

```
@Override
public boolean write(String record) {
    boolean success = false;
    File f = this.filename;
    ArrayList<String> data = null;
    if (f.exists()) {
        data = this.readAll();
    } else {
        try {
            f.createNewFile();
        } catch (IOException ex) {
            Logger.getLogger(textbinary.class.getName()).log(Level.SEVERE, null, ex);
        }
        data = new ArrayList();
    }
    if (data != null) {
        data.add(record);
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(this.filename))) {
            out.writeObject(data);
            success = true;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            return success;
        }
    } else {
        return success;
    }
}
```

Figure 27: Part 2 of `filewriter.java`

The overriding of the first ReadAll method is to allow the selected file's data to be read, whereas the overriding of the second write method is to first determine if a file exists, if the file does not exist, a file with the pre-determined file name will be created. Once the file is found, the string variable record is then written into the text file.

5.2.4 textbinary.java

```
public class textbinary extends FileAccess{
    @Override
    public ArrayList<String> readAll() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public boolean write(String record) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

Figure 28: Code for textbinary.java

This class was created for the main purpose of throwing the exception of the abstract classes that was declared in filereader.java due to exceptions not being able to be created in the same abstract class that it was declared.

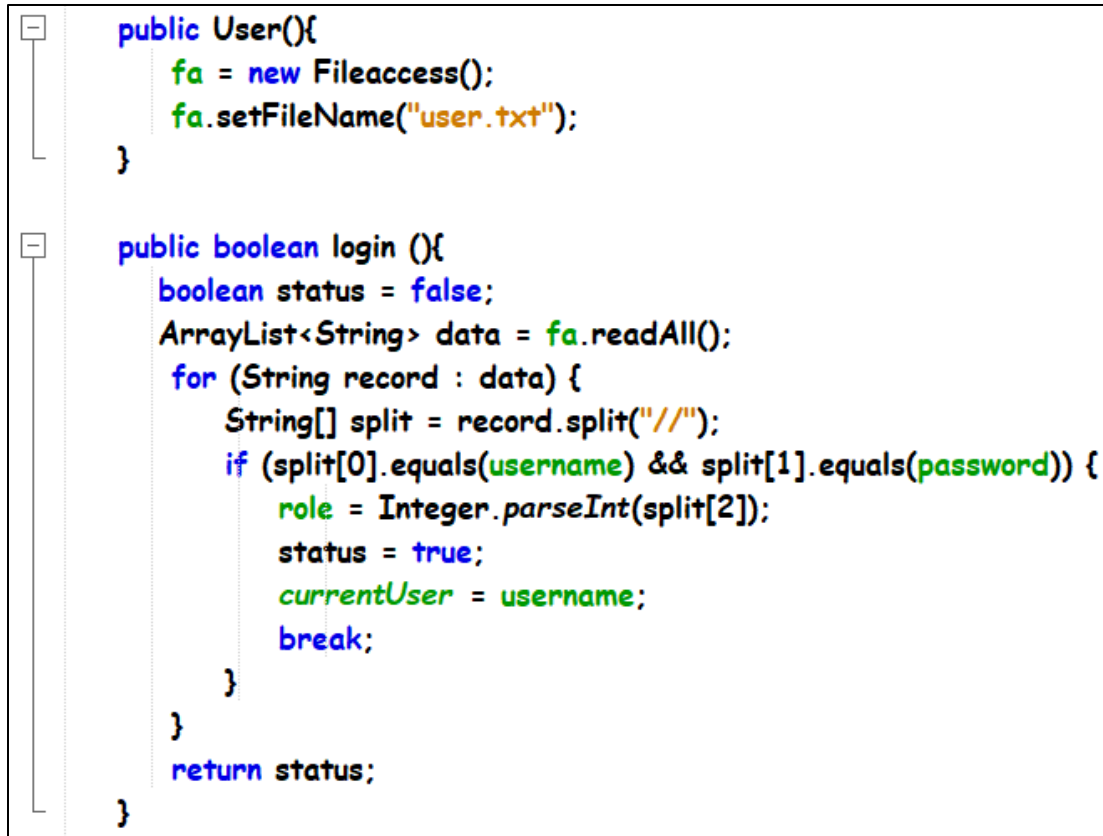
5.3 Class, Attributes and Method

5.3.1 User class

```
public class User {  
  
    private Fileaccess fa;  
    private String username,password;  
    private int role;  
  
    private String newname, newcontact, newemail, newdegreelevel;  
}
```

Figure 29: User Class

User class contains some attributes which are String that means as an object, integer as “int” to set it into numbering and backed file which is “Fileaccess”. In front of them, they contain private keyword which mean is an access modifier used for attributes, method and constructors to making them only accessible within the User class. (W3School, 1999)



```
public User(){
    fa = new FileAccess();
    fa.setFileName("user.txt");
}

public boolean login (){
    boolean status = false;
    ArrayList<String> data = fa.readAll();
    for (String record : data) {
        String[] split = record.split("/");
        if (split[0].equals(username) && split[1].equals(password)) {
            role = Integer.parseInt(split[2]);
            status = true;
            currentUser = username;
            break;
        }
    }
    return status;
}
```

Figure 30: User class

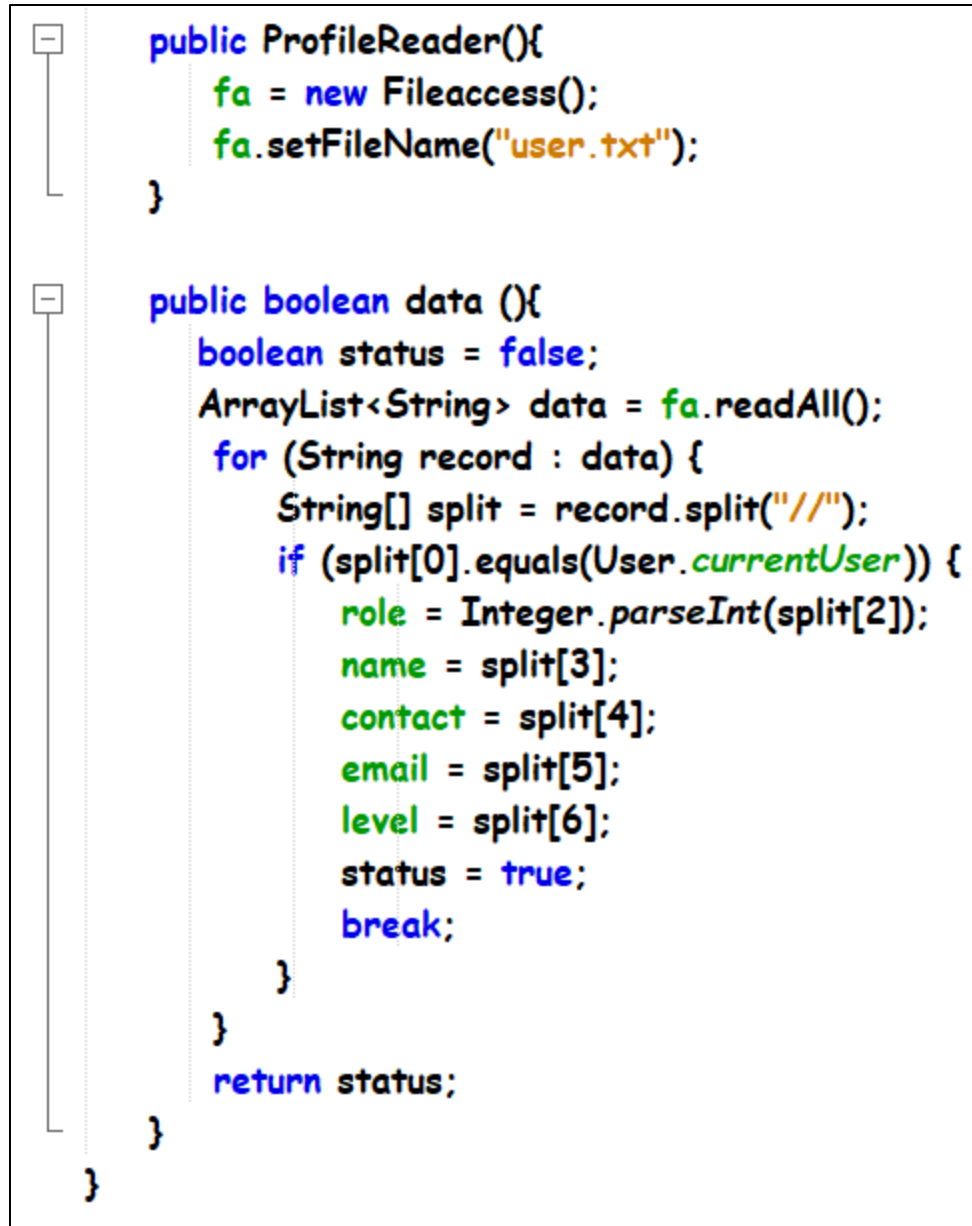
Inside the user class it also contains constructor to set the file access to “user.txt”. Then inside the login method, it will read based on the “user.txt” to find out the username, password and role. The purpose to do that is to make sure when it performs the operation every user can login into the correct pages and get the session of the user. (Anon., 2019)

5.3.2 Profile Reader Class

```
public class ProfileReader {  
    private Fileaccess fa;  
    private String name, level, contact, email;  
    private int role;  
}
```

Figure 31: Profile Reader Class

Profile Reader class contains some attributes which are String that means as an object, integer as “int” to set it into numbering and backed file which is “Fileaccess”. In front of them, they contain private keyword which mean is an access modifier used for attributes, method and constructors to making them only accessible within the Profile Reader class. (W3School, 1999)



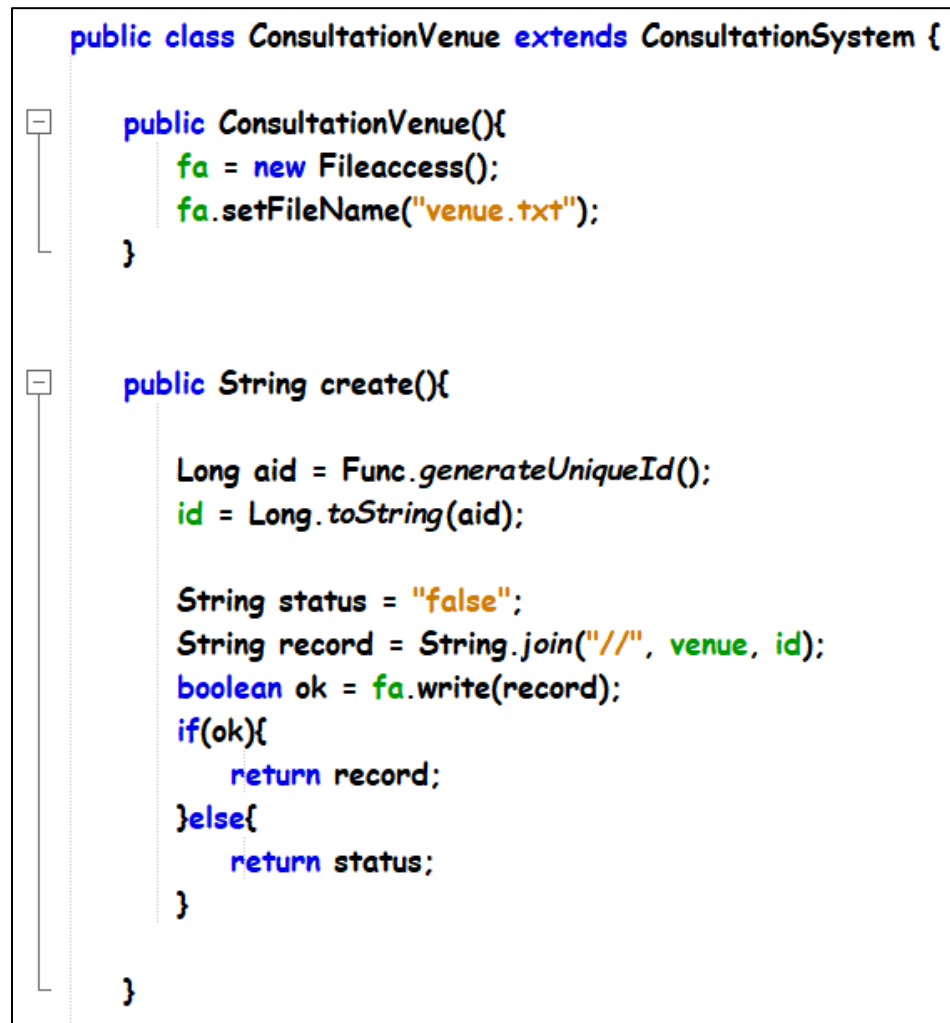
```
public ProfileReader(){
    fa = new Fileaccess();
    fa.setFileName("user.txt");
}

public boolean data (){
    boolean status = false;
    ArrayList<String> data = fa.readAll();
    for (String record : data) {
        String[] split = record.split("//");
        if (split[0].equals(User.currentUser)) {
            role = Integer.parseInt(split[2]);
            name = split[3];
            contact = split[4];
            email = split[5];
            level = split[6];
            status = true;
            break;
        }
    }
    return status;
}
```

Figure 32: Profile Reader Class

Inside the Profile Reader class, it also contains constructor to set the file access to “user.txt”. Then inside the data method which will operate to read the “user.txt” file and make sure it gets the right session user then get all the details of the user. (Anon., 2019)

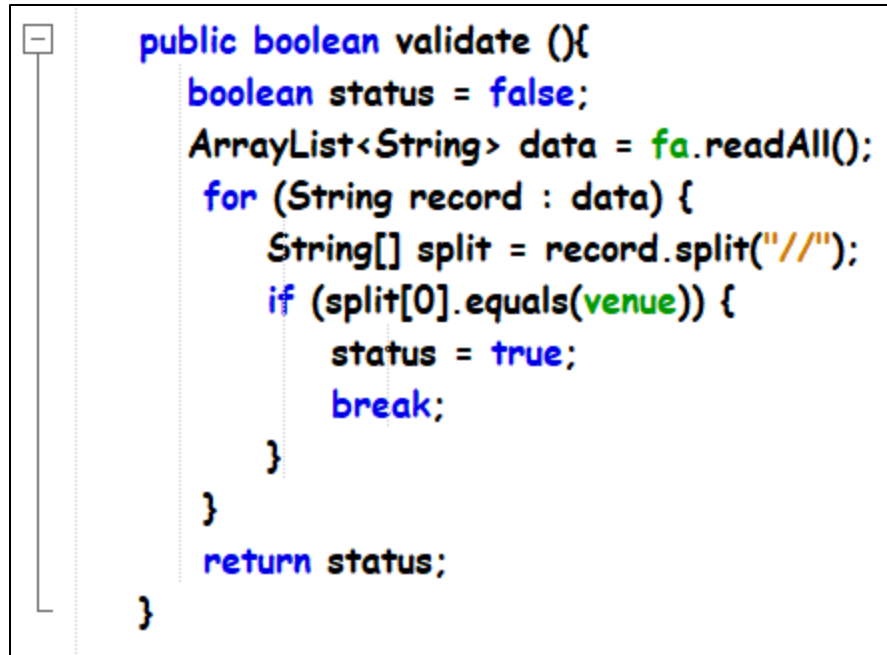
5.3.3 Consultation Venue Class



```
public class ConsultationVenue extends ConsultationSystem {  
  
    public ConsultationVenue(){  
        fa = new Fileaccess();  
        fa.setFileName("venue.txt");  
    }  
  
    public String create(){  
  
        Long aid = Func.generateUniqueId();  
        id = Long.toString(aid);  
  
        String status = "false";  
        String record = String.join("//", venue, id);  
        boolean ok = fa.write(record);  
        if(ok){  
            return record;  
        }else{  
            return status;  
        }  
    }  
}
```

Figure 33: Consultation Venue Class

Consultation Venue class will extend the Consultation System class which is parent class that will pass the attributes to Consultation Venue which is child class. Inside consultation venue class, there is a constructor there to set the file name to access "venue.txt". Then inside create method, random ID will be generated when creating new venue and all the record will write into the venue.txt once it created successfully.

A screenshot of a code editor window. On the left side, there is a vertical line with a small square icon at the top, representing a class structure or a list. The main area of the editor contains the following Java code:

```
public boolean validate (){
    boolean status = false;
    ArrayList<String> data = fa.readAll();
    for (String record : data) {
        String[] split = record.split("//");
        if (split[0].equals(venue)) {
            status = true;
            break;
        }
    }
    return status;
}
```

Figure 34: Consultation Venue Class

In Consultation Venue the class includes the validate method to make sure all the venue is read correctly.

```
@Override
public boolean EditVenue() {
    File filename = new File("venue.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("/");

        if (split[1].equals(id)) {
            String newRecordLine = String.join("/", newvenue, split[1]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }else{
        return false;
    }
}
```

Figure 35: Consultation Venue Class

The override method will be explained on the polymorphism above. For the Edit method inside Consultation Venue class, it will read the venue.txt text file to save the old list inside array then new list will be the details that updated. It will read based on the id and write the new list to replace the old list and maintain the id.

```
public boolean Delete() {  
  
    boolean success = false;  
  
    File filename = new File("venue.txt");  
    ArrayList<String> old_list = fa.readAll();  
    ArrayList<String> new_list = new ArrayList();  
  
    for (String oldLine : old_list) {  
  
        String[] split = oldLine.split("/");  
        if (!split[1].equals(id)) {  
            new_list.add(oldLine);  
        }  
    }  
  
    try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {  
        for (String line : new_list) {  
            out.println(line);  
            success = true;  
        }  
        out.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    if (old_list.size() == 1){  
        success = true;  
    }  
  
    return success;  
}
```

Figure 36: Consultation Venue Class

Delete method in Consultation Venue class is to delete the venue that has created by lecturer. It will remove the deleted venue in the venue.txt text file. It will read based on id to delete the correct row in the venue.txt text file.

5.3.4 Consultation Hour Class

```
public class ConsultationHour extends ConsultationSystem{  
  
    public ConsultationHour(){  
        fa = new Fileaccess();  
        fa.setFileName("booking.txt");  
    }  
}
```

Figure 37: Consultation Hour Class

In Consultation Hour class it will extend the Consultation System which will pass the attributes to the child class from Consultation System to the Consultation Hour. Then the constructor to set the file access to “booking.txt”.

```
public boolean validate (){  
    boolean status = false;  
    ArrayList<String> data = fa.readAll();  
    for (String record : data) {  
        String[] split = record.split("/");  
        if (split[0].equals(venue) && split[1].equals(date) && split[2].equals(timestart) &&  
            split[3].equals(timeend) && split[4].equals("Available") && split[6].equals(User.currentUser)) {  
            status = true;  
            break;  
        }else if (split[0].equals(venue) && split[1].equals(date) && (split[2].equals(timestart) ||  
            split[3].equals(timeend)) && split[4].equals("Available") && split[6].equals(User.currentUser)) {  
            status = true;  
            break;  
        }  
    }  
    return status;  
}
```

Figure 38: Consultation Hour Class

This method includes in Consultation Hour class to do the validation when creating consultation hour, it will read based on the “booking.txt” to get all the consultation hour details. It will validate that venue, date, timestart, timeend, status cannot be duplicated with the previous created consultation hour when creating the consultation hour based on the same lecturer. It also validates that whether timestart or timeend one of them cannot be duplicate when creating consultation hour class.

```
public String create(){  
    Long aid = Func.generateUniqueId();  
    id = Long.toString(aid);  
  
    String status = "false";  
    String record = String.join("/", venue, date, timestart, timeend, "Available", "None", User.currentUser, id);  
    boolean ok = fa.write(record);  
    if(ok){  
        return record;  
    }else{  
        return status;  
    }  
}
```

Figure 39: Consultation Hour Class

This create method in Consultation Hour class is to create a new consultation hour. It will operate by lecturer, once the operator works it will generate a random id for the consultation hour and store all the details into "booking.txt".


```
@Override
public boolean EditConsultationHour() {
    File filename = new File("booking.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("//");
        if (split[7].equals(id)) {
            String newRecordLine = String.join("//", newvenue, newdate, newtimestart, newtimeend, split[4], split[5], split[6], split[7]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }else{
        return false;
    }
}
```

Figure 40: Consultation Hour Class

This edit consultation hour method in Consultation Hour class, is to operate the update consultation hour. The override method will be explained on the polymorphism part above. The edit consultation hour will operate by lecturer, it will read all the “booking.txt” details as old list record, and when a lecturer updates the consultation hour, the old list will be replaced by the new list based on the id that the lecturer selected.

```
public boolean cancelconsultationhour() {
    File filename = new File("booking.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("/");
        if (split[7].equals(id)) {

            String newRecordLine = String.join("/", split[0], split[1], split[2], split[3], "Canceled", split[5], split[6], split[7]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }else{
        return false;
    }
}
```

Figure 41: Consultation Hour Class

This cancel consultation hour method in Consultation Hour class will operate to cancel the status of the consultation hour. It can be done by the lecturer, once the consultation hour is cancelled the text file which is "booking.txt" will replace the old list into a new list, which means changing the status of consultation hour. It will read based on the id that lecturer selected to replace it into the correct line inside the text file.

```
public boolean Delete() {  
  
    boolean success = false;  
  
    File filename = new File("booking.txt");  
    ArrayList<String> old_list = fa.readAll();  
    ArrayList<String> new_list = new ArrayList();  
  
    for (String oldLine : old_list) {  
  
        String[] split = oldLine.split("/");  
  
        if (!split[7].equals(id)) {  
            new_list.add(oldLine);  
        }  
    }  
  
    try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {  
        for (String line : new_list) {  
            out.println(line);  
            success = true;  
        }  
        out.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    if (old_list.size() == 1){  
        success = true;  
    }  
    return success;  
}
```

Figure 42: Consultation Hour Class

This delete method in consultation hour class will operate to delete the consultation hour. It can only be initiated by lecturer, it will be based on the id that lecturer selected to remove the correct row inside the "booking.txt" text file.

```

@Override
public boolean booking() {
    File filename = new File("booking.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("//");
        if (split[7].equals(id)) {

            String newRecordLine = String.join("//", split[0], split[1], split[2], split[3], "Booked", User.currentUser, split[6], split[7]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }else{
        return false;
    }
}

```

Figure 43: Consultation Hour Class

The booking method in consultation hour class will operate when students want to book consultation hours. It will show the student name who booked the consultation hour and change the status into Booked, so others cannot rebook the same consultation hour. When students book a consultation hour it will change the status and session of student name inside the text file which is "booking.txt". It will be based on the id that student booked to change the status and student name to the correct row in the text file.

```

public boolean cancelbooking() {
    File filename = new File("booking.txt");
    ArrayList<String> old_list = fa.readAll();
    ArrayList<String> new_list = new ArrayList();

    for (String oldLine : old_list) {

        String[] split = oldLine.split("/");
        if (!split[5].equals(currentUser)){
            JOptionPane.showMessageDialog(null, "You cannot cancel others booked consultation hour");
            return false;
        }
        if (split[7].equals(id)) {
            String newRecordLine = String.join("/", split[0], split[1], split[2], split[3], "Available", "None", split[6], split[7]);
            new_list.add(newRecordLine);
        } else {
            new_list.add(oldLine);
        }
    }

    if(new_list.size() > 0){
        try(PrintWriter out = new PrintWriter(new FileWriter(filename))) {
            for (String line : new_list) {
                out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }else{
        return false;
    }
}

```

Figure 44: Consultation Hour Class

Cancel booking method inside the consultation hour class will operate when students want to cancel consultation that is booked by the same student. It will operate based on id and read the text file “booking.txt” to find the match line to update the status and name into the text file. Inside the method, there is a validation to validate the only same student can cancel his or her consultation hour, where others will not be allowed to do so.

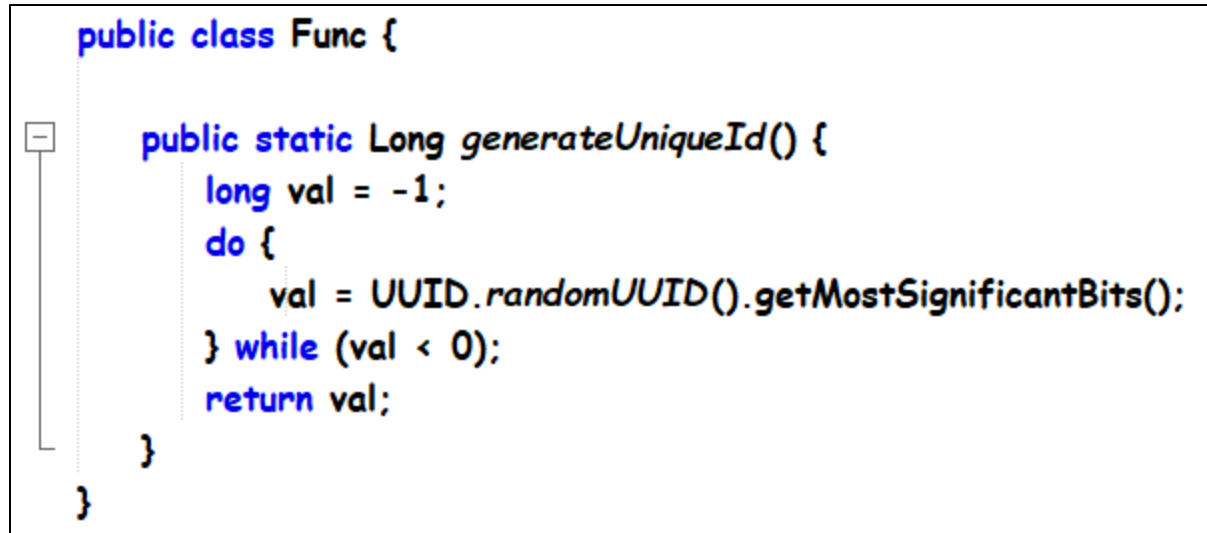


Figure 45: Func Class

The Func Class is to show the method that generates a unique id to create consultation hour and venue.

5.3.5 Java Team class

```
public class JavaTeam {  
  
    private Fileaccess fa;  
    private String text;  
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");  
    LocalDateTime now = LocalDateTime.now();  
  
    public JavaTeam(){  
        fa = new Fileaccess();  
        fa.setFileName("feedback.txt");  
    }  
  
    public String getText() {  
        return text;  
    }  
  
    public void setText(String text) {  
        this.text = text;  
    }  
  
    public String create(){  
  
        String status = "false";  
        String record = String.join(" ", dtf.format(now), User.currentUser, text );  
  
        boolean ok = fa.write(record);  
        if(ok){  
            return record;  
        }else{  
            return status;  
        }  
    }  
}
```

Figure 46: Java Team class

It uses to create a feedback by inserting the local time, date, session user and the text into the feedback.txt.

6.0 Validation Explanation

This section explains the validation that is used to prevent logical errors in the software process. Please take note that this section only explains the codes with pure validation, meaning most codes here are just one time IF conditions that just returns if they are met without executing any other codes.

6.0.1 Creating a consultation venue

```
if(venue.equals("")){  
    JOptionPane.showMessageDialog(this, "Please Insert the Venue");  
    return;  
}  
  
if(cv.validate()){  
    JOptionPane.showMessageDialog(rootPane, "Venue Duplicated");  
    return;  
}
```

Figure 47: Validation code for creating a consultation venue

The first IF statement is to ensure that the venue input field is not intentionally left blank, whereas the second IF statement is to prevent duplication of the venue. The validate function is explained above in the “modules” folder section.

6.0.2 Updating a consultation venue

```
if (tableVenue.getSelectionModel().isSelectionEmpty()){  
    JOptionPane.showMessageDialog(rootPane, "Please Select a Specific Row to Update");  
    return;  
}  
  
if(txtVenue.getText().equals("")){  
    JOptionPane.showMessageDialog(rootPane, "Please Fill in the Venue");  
    return;  
}
```

Figure 48: Validation code for updating a consultation venue

The first IF statement is to ensure that a row in the table containing data is selected for a data to be updated. The second IF statement is to ensure that the venue input field is not intentionally left blank.

6.0.3 Validating user roles

```
if(prObj.data()){  
    if(prObj.getRole() == 1){  
        StudentMenu menu_gui = new StudentMenu();  
        menu_gui.setLocationRelativeTo(null); //center  
        menu_gui.setVisible(true);  
        this.dispose();  
    }  
    else if(prObj.getRole() == 0){  
        LecturerMenu menu_gui = new LecturerMenu();  
        menu_gui.setLocationRelativeTo(null); //center  
        menu_gui.setVisible(true);  
        this.dispose();  
    }  
    else if(prObj.getRole() == 2){  
        AdminMenu menu_gui = new AdminMenu();  
        menu_gui.setLocationRelativeTo(null); //center  
        menu_gui.setVisible(true);  
        this.dispose();  
    }  
}
```

Figure 49: Code to validate user roles

This code is used to validate user roles and navigate them to the respective menus (JFrame forms). In this case, “0” is meant as lecturer role, “1” is meant as student role, and “2” is meant as admin role. This code concept is also applied at situations where the same page is used to display data, but their functions vary slightly based on the roles, for example manage profile.

6.0.4 Creating and editing a consultation hour

(Please refer to the image in the next page)

```

if (venue == null || date.equals("(null)") || timestart.equals("(null)") || timeend.equals("(null)")){
    JOptionPane.showMessageDialog(rootPane, "Please fill in all the fields.");
    return;
}

if (date1.isBefore(todayDate)) {
    JOptionPane.showMessageDialog(rootPane, "Please choose a valid date.");
    return;
}

if (time2.isBefore(time1) || time2.equals(time1)) {
    JOptionPane.showMessageDialog(rootPane, "Please check start and end time logic.");
    return;
}

if (time1.isBefore(timelow) && date1.isEqual(todayDate)) {
    JOptionPane.showMessageDialog(rootPane, "Please select a start time after current time. Time zone used is Asia/Shanghai.");
    return;
}

if (time1.isBefore(startlimit) || time2.isAfter(endlimit)){
    JOptionPane.showMessageDialog(rootPane, "Please select a time within 8:30am to 8:30pm.");
    return;
}

if (date1.getDayOfWeek() == DayOfWeek.SATURDAY || date1.getDayOfWeek() == DayOfWeek.SUNDAY){
    JOptionPane.showMessageDialog(rootPane, "Please book a consultation on weekdays only.");
    return;
}

if(ch.validate()){
    JOptionPane.showMessageDialog(rootPane, "Clashing of entries.");
    return;
}

//Creation method starts here.
if(ch.create() != null){
    model.addRow(new String[]{venue, date, timestart, timeend, "Available", "None", User.currentUser, ch.getId() });
    JOptionPane.showMessageDialog(rootPane, "Insert Successfully");
}else{
    JOptionPane.showMessageDialog(rootPane, "Technical error encountered.");
}

```

Figure 50: Validation code for editing and creating a consultation hour

The first validation is to ensure that all fields are entered, followed by making sure the date entered must at least be equals to today's date (Bourque, 2014). The end time should be greater than the start time constantly in the third validation.

For the fourth validation, it is to check that the time entered for a consultation does not exceed today's date and current time that has the Asia/Shanghai timezone (Oracle, n.d.). The following one sets a time limit for the consultation hour booking, whereas the sixth one sets the booking days to weekdays only (Cristian, 2011). The last validation method prevents the duplication of data.

It is required to note that the validation for create and editing a consultation hour is the same, with an additional verification in edit which is ensuring there is a selected row with data such as mentioned in the examples above.

6.0.5 Cancelling a consultation hour

```
if (tableBooking.getSelectionModel().isSelectionEmpty()){
    JOptionPane.showMessageDialog(rootPane, "Please Select a Specific Row to Cancel");
    return;
}

con.setId(tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 7).toString());
String booked = tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 4).toString();

if(booked.equals("Canceled")){
    JOptionPane.showMessageDialog(this, "This has been canceled");
    return;
}
```

Figure 51: Validation code for cancelling a consultation hour

The first validation is to ensure that a valid data row is selected, and the second validation is to ensure that the status of the consultation hour that is attempting to be cancelled must not be cancelled in the first place.

6.0.6 Deleting a consultation hour

```

if (tableBooking.getSelectionModel().isSelectedEmpty()){
    JOptionPane.showMessageDialog(rootPane, "Please Select a Specific Row to Delete");
    return;
}

String booked = tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 4).toString();

if(booked.equals("Canceled")){
    JOptionPane.showMessageDialog(this, "Canceled Consultation Hour cannot Deleted");
    return;
}

```

Figure 52: Validation code for deleting a consultation hour

The first validation is to ensure when deleting a specific row with data is selected, and the second validation is to ensure that a consultation hour with the canceled status cannot be deleted.

6.0.7 Booking a consultation hour (Students)

```

ConsultationHour con = new ConsultationHour();
if (tableBooking.getSelectionModel().isSelectedEmpty()){
    JOptionPane.showMessageDialog(rootPane, "Please Select a Specific Row to Book");
    return;
}
con.setId(tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 7).toString());
String booked = tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 4).toString();

if(booked.equals("Booked")){
    JOptionPane.showMessageDialog(this, "This has been booked.");
    return;
}

if(booked.equals("Canceled")){
    JOptionPane.showMessageDialog(this, "This has been canceled.");
    return;
}

```

Figure 53: Validation code for booking a consultation hour

The first validation is to ensure that when booking a table row with data is selected. The second and third validation is to ensure that the status of the consultation hour that is trying to be booked is not booked nor cancelled.

6.0.8 Cancelling a booked consultation hour (Students)

```
if (tableBooking.getSelectionModel().isSelectionEmpty()) {
    JOptionPane.showMessageDialog(rootPane, "Please Select a Specific Row to Cancel");
    return;
}

con.setId(tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 7).toString());
String booked = tableBooking.getModel().getValueAt(tableBooking.getSelectedRow(), 4).toString();

if (booked.equals("Available")) {
    JOptionPane.showMessageDialog(this, "Please Book First");
    return;
}

if (booked.equals("Canceled")) {
    JOptionPane.showMessageDialog(this, "This has been canceled");
    return;
}
```

Figure 54: Validation code for cancelling a booked consultation hour

This validation method is the same as the validation methods that were stated in section 6.0.7, except for the status of the consultation hour trying to be cancelled must not be available or cancelled.

7.0 Text File Access

“**user.txt**” will be accessed by modules/User.java and modules/ProfileReader.java.

Username//Password//Role//Name//Contact Number//Email Address//Degree Level

“**venue.txt**” will be accessed by modules/ConsultationVenue.java.

Venue//Venue ID

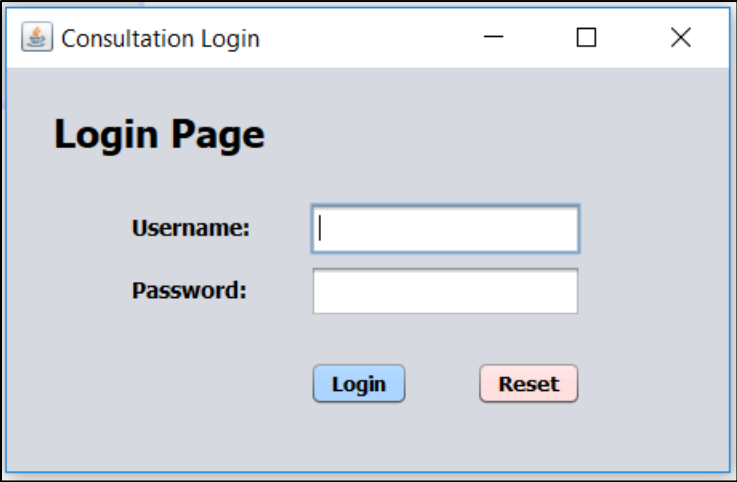
“**booking.txt**” will be accessed by modules/ConsultationHour.java.

Venue//Date//Time Start//Time End//Status//Student Name//Lecturer Name//Consultation ID

“**feedback.txt**” will be accessed by modules/JavaTeam.java

Date Time: User: Text

8.0 Interface & User Manual



The image shows a window titled "Consultation Login". Inside the window, the text "Login Page" is displayed at the top. Below this, there are two text input fields. The first field is labeled "Username:" and the second field is labeled "Password:". Below the password field, there are two buttons: a blue "Login" button and a red "Reset" button.

Figure 55: Login

Login Page will be used by student, lecturer and admin who will login into their own menu page. Users will insert their username and password into the username text field and password text field. The Login button is to let user login into their own menu page. The Reset button is to clear the text field.

8.1 Lecturer

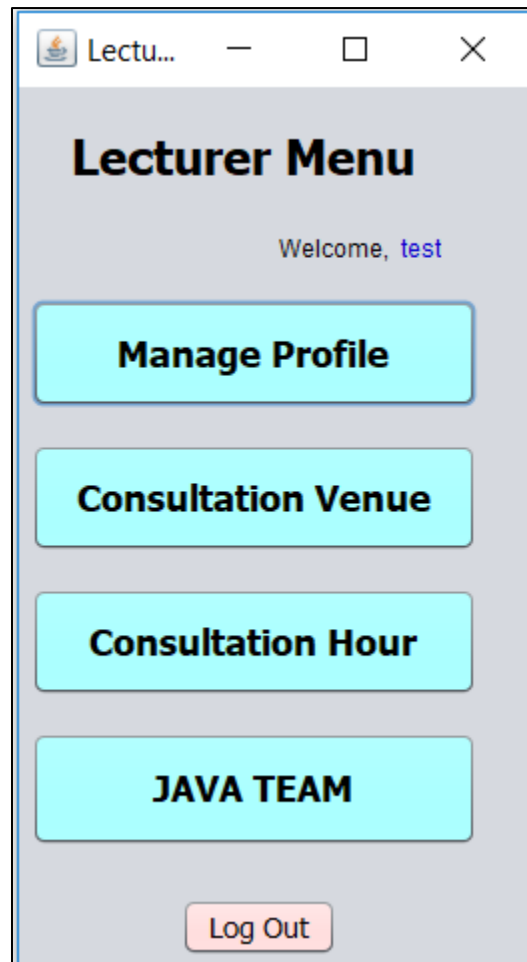
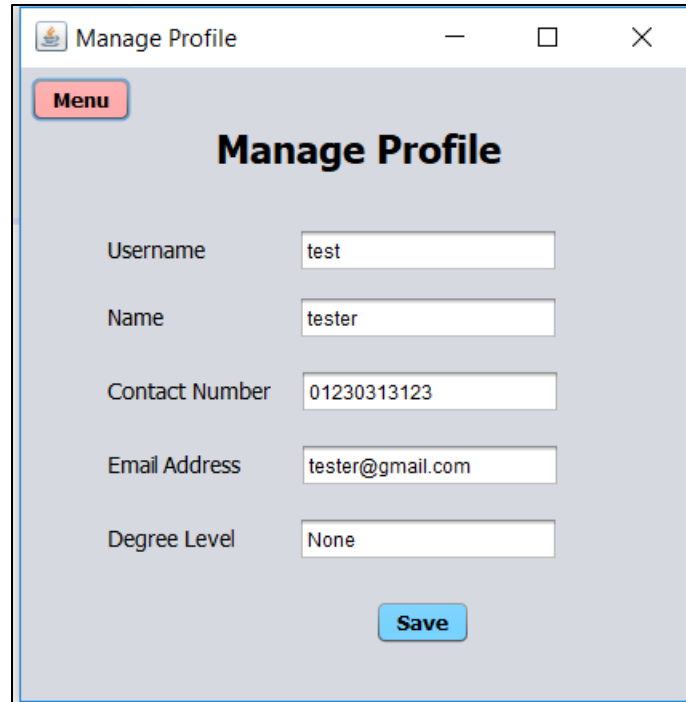


Figure 56: Lecturer Menu

If lecturer login successfully, lecturer will be directed to this lecturer menu page. In the menu page, lecturer can access to manage profile, consultation venue, consultation hour and log out. Lecturer can manage their own profile by clicking the Manage Profile button, to create consultation venue lecturer can click the Consultation Venue button, to create consultation hour lecturer can click the Consultation Hour button, lecturer can access to the Java Team to chat with students and lecturer can logout by clicking the Log Out button.



The screenshot shows a web browser window titled "Manage Profile". In the top-left corner of the page content, there is a red button labeled "Menu". The main heading is "Manage Profile" in bold black text. Below the heading, there are five form fields, each with a label to its left: "Username" with the value "test", "Name" with the value "tester", "Contact Number" with the value "01230313123", "Email Address" with the value "tester@gmail.com", and "Degree Level" with the value "None". At the bottom center of the form area is a blue button labeled "Save".

Figure 57: Lecturer Manage Profile

If lecturers click the Manage Profile button in the Lecturer Menu Page, it will direct the lecturer to Manage Profile Page. In this page, lecturer can update their name, contact number and email address by typing their new name or contact number or email address to the right text field. Then lecturer can click the Save button to make sure the details are updated. After finishing editing, lecturer can click on the Menu button to go back to Lecturer Menu Page.

The screenshot shows a web application window titled "Create consultation venue". In the top-left corner, there is a red button labeled "Menu". The main heading is "Create consultation venue". Below the heading, there is a text input field labeled "Venue Name" and a blue button labeled "Search". Below the input field, there is a table with two columns: "Venue Name" and "Id". The table contains three rows of data. To the right of the table, there are three buttons: a blue "Add" button, a blue "Update" button, and a red "Delete" button.

Venue Name	Id
B-6-1	3389109157084...
B-6-10	6545989497215...
B-6-5	5692732638594...

Figure 58: Lecturer Create Consultation Venue

Lecturer can create consultation venue by clicking the Consultation Venue button in the Lecturer Menu Page. In this page, lecturer can create the new Venue by inserting the new venue in the venue text field then click on Add button to add the new venue, the new venue will be shown in the table along with venue id when the venue is successfully created. Lecturer can search a specific venue by typing the venue into the venue text field then click on the Search button to search the venue, the search result will be displayed in the table.

Lecturer can edit the venue by clicking the table row, the venue in the table row will display in the venue text field once a specific row is clicked. So, lecturer can change the venue in the text field then click the Update button to update the venue, the new venue will show in the table once it is successfully updated. Lecturer can also delete the venue by clicking on a specific table row then click on the Delete button to delete the venue, the deleted venue will remove from the table. Lecturer can click on the Menu button to go back to Lecturer Menu page.

Define Consultation Hour

Menu

Lecturer Name: test Create

Venue: B-6-2 ▼

Date: ... Search

Time: ▼ ▼

Venue	Date	Start Time	End Time	Status	Student	Lecturer	ID
B-6-2	2019-07-18	00:30	01:30	Canceled	eren	test	55664933276986...
B-6-9	2019-07-29	12:30	13:30	Available	None	test	15885629229111...
B-6-4	2019-07-30	12:30	13:30	Booked	eren	test	91098124723492...
B-6-4	2019-07-30	14:00	15:00	Available	None	test	65173584839785...

Edit Function

Edit Consultation

Cancel Consultation

Delete Consultation

View Function

View Booked

View Canceled

Refresh

Figure 59: Lecturer Define Consultation Hour

Lecturer can create consultation hour by clicking on the Consultation Hour button in the Lecturer Menu Page. In the Define Consultation Hour page lecturer can create, view, add, delete, update and search the consultation hour. Lecturer can fill in all the details then click the Create button to create a new consultation hour for student. Lecturer can search for the consultation by choosing the right date then click on Search button, the result will display in the table.

Lecturer can edit the consultation hour by clicking on a specific row in the table, so lecturer can change the details in the text field there then click on Edit Consultation button to update the consultation hour, the result will display in the table once it is updated successfully. Lecturer can cancel consultation by clicking on the specific row in the table that show Status equal to “Available” then click on the Cancel Consultation button, the result will display in the table once it successfully canceled.

Lecturer can delete the consultation hour by clicking on a specific row in the table then click Delete Consultation button, the result will be removed once it successfully deleted. For the View Function, lecturer can view which consultation is booked by clicking on the View Booked button, the result will display in the table. Lecturer can view canceled consultation hour by click on the View Canceled button, the result will display in the table. When lecturer want to view back all the records, lecturer can click on the Refresh button, all the records will be displayed in the table.

8.2 Student

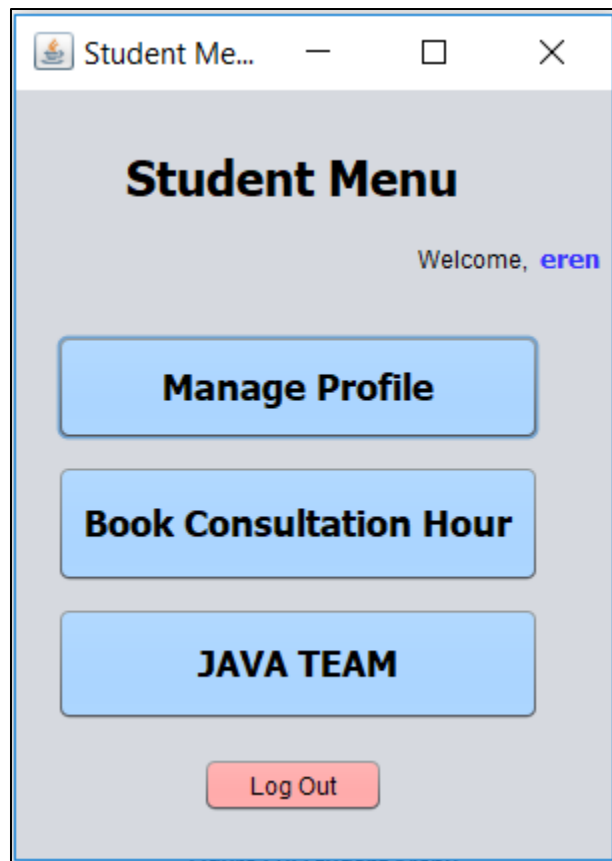
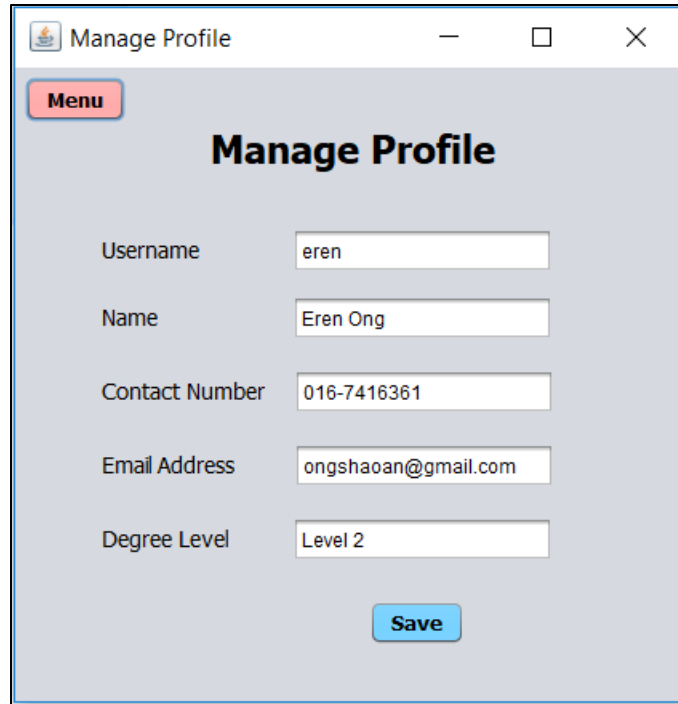


Figure 60: Student Menu

If student login successfully in the Login Page, it will direct student to the Student Menu page. Student can access to manage profile, book consultation hour and logout. Student can click on Manage Profile button to manage own profile. Student can click on Book Consultation Hour button to book consultation hour student can access to Java Team to chat with lecturer and student can click on Log Out button to logout.



The screenshot shows a web browser window titled "Manage Profile". In the top-left corner of the page content, there is a red button labeled "Menu". The main heading is "Manage Profile" in bold black text. Below the heading, there are five text input fields, each with a label to its left: "Username" (containing "eren"), "Name" (containing "Eren Ong"), "Contact Number" (containing "016-7416361"), "Email Address" (containing "ongshaoan@gmail.com"), and "Degree Level" (containing "Level 2"). At the bottom center of the form is a blue button labeled "Save".

Figure 61: Student Manage Profile

Student can access their own profile by clicking on the Manage Profile button in the Student Menu. Student can view or change their details in Manage Profile page, student can update their own details by typing in the new details into the correct text field then click Save button to update new details. Student can go back to student menu by clicking on the Menu button.

Book Consultation Hour

[Back](#)

Lecturer Name [Search](#)

Venue	Date	Start Time	End Time	Status	Student	Lecturer	Id
B-6-2	2019-07-18	00:30	01:30	Canceled	eren	test	556649332...
B-6-9	2019-07-29	12:30	13:30	Available	None	test	158856292...
B-6-4	2019-07-30	12:30	13:30	Booked	eren	test	910981247...
B-6-4	2019-07-30	14:00	15:00	Available	None	test	651735848...
B-6-4	2019-07-30	14:00	15:00	Available	None	siti	651735848...

[Book](#) [Cancel Book](#)

Figure 62: Student Book Consultation Hour

Student can book consultation hour by clicking on the Book Consultation Hour button in the Student Menu page. In this page, student can book the consultation hour by clicking on a specific table row, then click Book button to book the consultation hour. The result will display in the table by showing changing the Status into Booked.

Student can cancel booked consultation hour by clicking on the specific row in the table that show Status is equals to Booked then click on the Cancel Book button to cancel the consultation hour. Student can search for specific lecturer to view the lecturer available consultation hour by inserting the lecturer name into lecturer name text field then click on the Search button. Student can go back to student page by click on the Back button.

9.0 Additional Feature

9.1 Admin (Additional Feature)

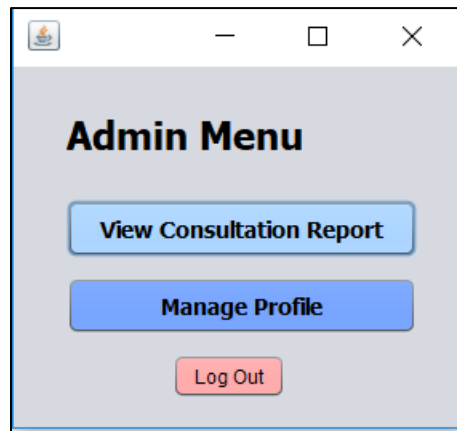
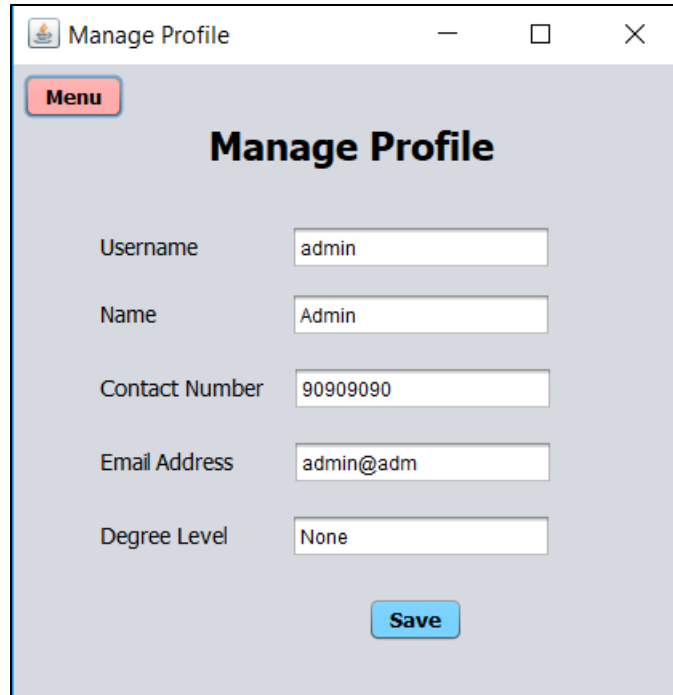


Figure 63: Admin Menu

When admin logs successfully in the Login page, it will direct admin to the Admin Menu. In the Admin menu, admin can access to View Consultation Report and Manage admin own profile. Admin can also log out in the Admin Menu.



Manage Profile

Menu

Manage Profile

Username

Name

Contact Number

Email Address

Degree Level

Save

Figure 64: Admin Manage Profile

Admin can access their own profile by clicking on the Manage Profile button in the Admin Menu. Admin can view or change their details in Manage Profile page, Admin can update their own details by typing in the new details into the correct text field then click Save button to update new details. Admin can go back to Admin Menu by clicking on the Menu button.

Admin Report

View Report

[Back](#)

Lecturer Name: [Search](#)

Venue	Date	Time Start	Time End	Status	Student Name	Lecturer Name	ID
B-6-2	2019-07-18	00:30	01:30	Canceled	eren	test	5566493327...
B-6-9	2019-07-29	12:30	13:30	Available	None	test	1588562922...
B-6-4	2019-07-30	12:30	13:30	Booked	eren	test	9109812472...
B-6-4	2019-07-30	14:00	15:00	Available	None	test	6517358483...
B-6-4	2019-07-30	14:00	15:00	Available	None	siti	6517358483...

[Print](#) [Refresh](#)

Figure 65: Admin Print Report

Admin can view consultation report by clicking on the View Consultation Report button in the Admin Menu page. Admin can view all the lecturer consultation and can also print the consultation in pdf format. Admin can search for a specific lecturer to view for the lecturer consultation hour by inserting the lecturer name in the text field then click on the Search button. Admin can refresh the page to view all the records in the table. Admin can go back to the Admin Menu page by clicking on the Back button.

Venue	Date	Time Start	Time End	Status	Student Name	Lecturer Name	ID
B-6-2	2019-07-18	00:30	01:30	Canceled	eren	test	5566493327...
B-6-9	2019-07-29	12:30	13:30	Available	None	test	1588562922...
B-6-4	2019-07-30	12:30	13:30	Booked	eren	test	9109812472...
B-6-4	2019-07-30	14:00	15:00	Available	None	test	6517358483...
B-6-4	2019-07-30	14:00	15:00	Available	None	siti	6517358483...

Figure 66: Result Printing

This is the result print and view in pdf format when Admin clicks the Print button in the View Report page.

9.2 Chat System

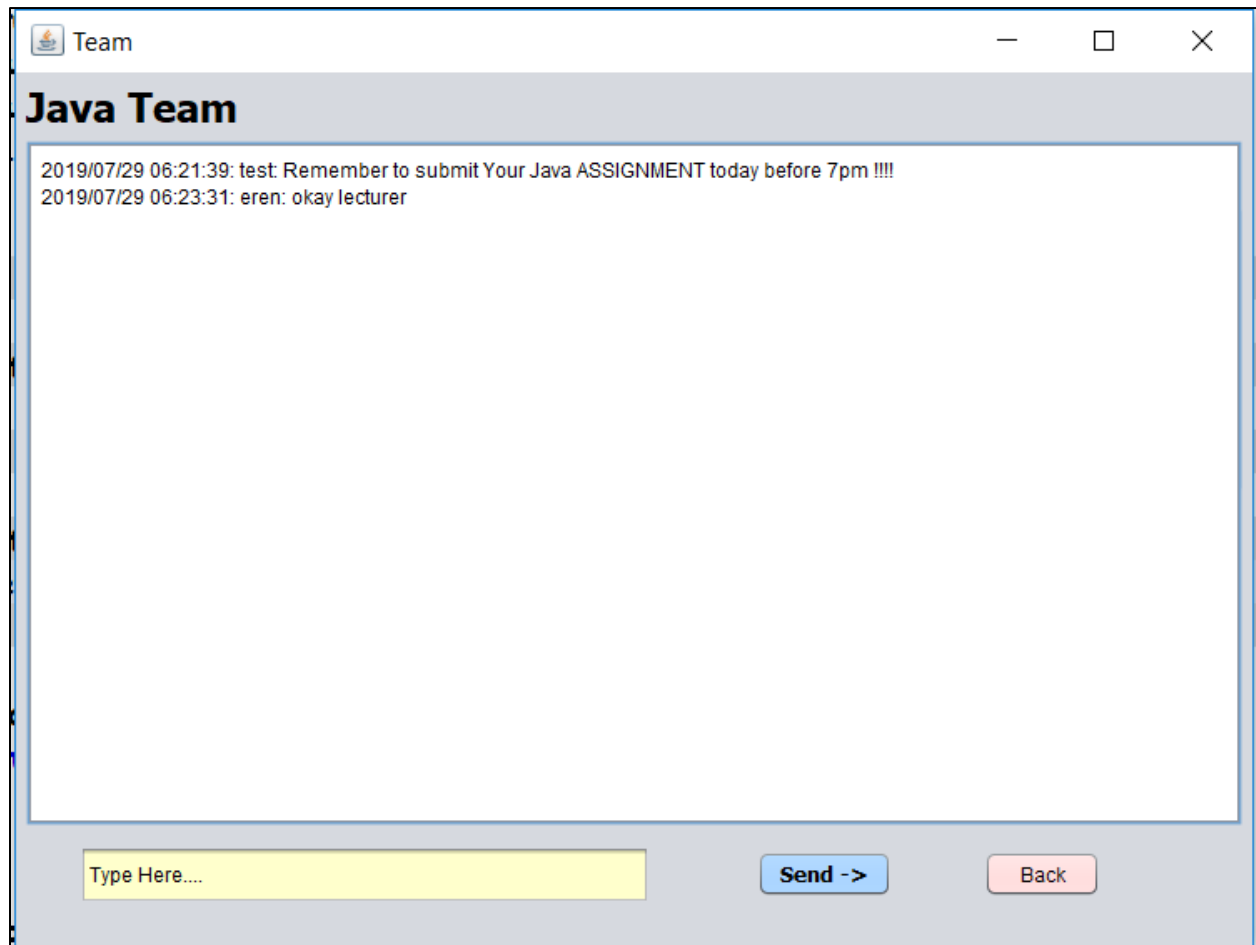


Figure 67: Chat System

The Java Team page contains both lecturer and student menus. The purpose is to provide for lecturers to inform any important things to students. It basically is a chat system like Microsoft Team; it involves every student in the system. Students and lecturers can type in text in the text field that shows "Type Here...." then click on the Send button to send the chat to the text area. They can go back to their own menu by clicking the back button.

10.0 Conclusion

In this assignment, we have learnt about various object-oriented programming concepts such as abstraction, inheritance, encapsulation and polymorphism and how to properly apply them in this assignment. We have also learnt how to design solutions for the requirements and create a software-based solution that exploits the strength of object-oriented paradigm.

Besides that, we have also touched on how to implement a software concept so that it relates with the real-world situation, and with enough validation techniques to ensure that minimal logical errors will ensue with the usage of the software. This also ensures that we apply enough logical and analytical thinking skills to develop innovative solutions for a range of problems.

Moreover, we acknowledged on understanding and grasping the Java programming language to solve the requirements and demonstrate them through the functions in the existing system. Communication skills were also obtained as all team members must communicate effectively and professionally as an individual or as a team member to ensure that the assignment can be completed efficiently and on time.

11.0 Workload Matrix

Name	Tasks	Signature
Ong Lip Yang TP045925	<p>OOP: Encapsulation & Polymorphism</p> <p>Coding part: -student book, search and cancel consultation hour -lecturer edit profile -lecturer edit, view, delete and search consultation hour -admin search, print -Java Team</p> <p>Documentation: -class diagram -GUI, user manual -code snippet (modules) -conclusion</p>	
Ong Shao An TP046057	<p>OOP: Abstraction & Inheritance</p> <p>Coding part: -Login function -student edit profile -lecturer create consultation venue -lecturer create consultation hour -admin edit profile</p> <p>Documentation: -use case diagram -code snippet (GUI Code & backend) -validation explanation -introduction</p>	

12.0 References

Anon., 2009. *GeeksforGeeks*. [Online]

Available at: <https://www.geeksforgeeks.org/inheritance-in-java/>
[Accessed 18 7 2019].

Anon., 2019. *geeksforgeeks*. [Online]

Available at: <https://www.geeksforgeeks.org/overriding-in-java/>
[Accessed 21 7 2019].

Anon., 2019. *tutorialpoints*. [Online]

Available at: https://www.tutorialspoint.com/java/java_methods
[Accessed 12 7 2019].

Anon., 2019. *tutorialpoints*. [Online]

Available at: https://www.tutorialspoint.com/java/java_overriding
[Accessed 21 7 2019].

Anon., 2019. *tutorialspoint*. [Online]

Available at: https://www.tutorialspoint.com/java/java_encapsulation
[Accessed 18 7 2019].

BlakeTNC, 2018. *LGoodDatePicker*. [Online]

Available at:

<https://github.com/LGoodDatePicker/LGoodDatePicker/blob/master/Project/src/main/java/com/github/fgooddatepicker/demo/FullDemo.java>

[Accessed 2 July 2019].

Bourque, B., 2014. *Stack Overflow*. [Online]

Available at: <https://stackoverflow.com/questions/24927701/how-to-compare-time-string-with-current-time-in-java>
[Accessed 25 July 2019].

Cristian, 2011. *Stack Overflow*. [Online]

Available at: <https://stackoverflow.com/questions/4637238/how-to-check-if-its-saturday-sunday>
[Accessed 25 July 2019].

Janssen, D., 2008. *techopedia*. [Online]

Available at: <https://www.techopedia.com/definition/24335/abstract-class-java>
[Accessed 13 7 2019].

JavaTpoint, 2011. *javapoints*. [Online]

Available at: <https://www.javatpoint.com/method-overloading-in-java>
[Accessed 23 7 2019].

JodaStephen, 2015. *Stack Overflow*. [Online]

Available at: <https://stackoverflow.com/questions/30788369/convert-string-to-localtime-without-nanoofseconds>
[Accessed 20 July 2019].

Oracle, n.d. *Oracle Docs*. [Online]

Available at: <https://docs.oracle.com/javase/8/docs/api/java/time/ZonedDateTime.html>

[Accessed 25 July 2019].

Prasad, 2014. *Stack Overflow*. [Online]

Available at: <https://stackoverflow.com/questions/21903251/how-to-redirect-to-another-window-in-java-swing>

[Accessed 4 July 2019].

SINGH, C., 2014. *BeginnersBook*. [Online]

Available at: <https://beginnersbook.com/2013/05/method-overloading/>

[Accessed 19 7 2019].

user586399, 2013. *Stack Overflow*. [Online]

Available at: <https://stackoverflow.com/questions/14368268/populating-jcombobox-with-a-text-file>

[Accessed 10 July 2019].

W3School, 1999. *W3School*. [Online]

Available at: https://www.w3schools.com/java/ref_keyword_private.asp

[Accessed 12 7 2019].