SEMESTER 2 - SESSION 2019/2020

UNIVERSITI MALAYSIA SABAH

KK14203 OBJECT-ORIENTED PROGRAMMING

INDIVIDUAL PROJECT

NAME	**NICHOLAS WONG**
MATRIC NO.	**BI19110023**
SECTION	**1**
TELEPHONE NO.	**011-1028 8929**
APPLICATION NAME	**SCHOOL INFIRMARY RECORD APP**

LECTURER: DR. MOHD SHAMRIE SAININ

User Manual

1. The **`main()`** method can be found in the Java class called **`"School_Infirmary_Record"`**. Starting up the app will display the Login Page (Figure 1).

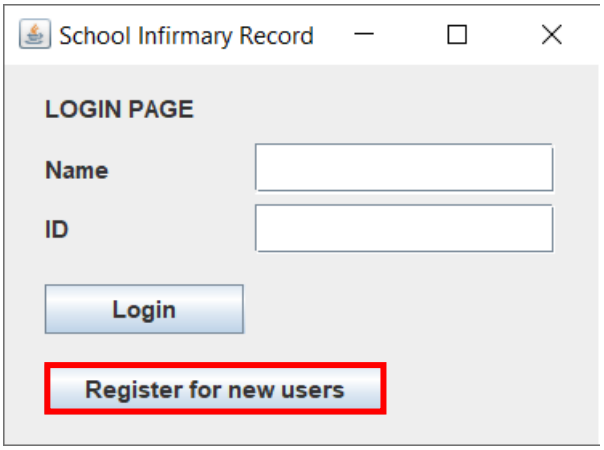

Figure 1: The Login Page

2. In order to log into the app, you will first need to register. Click on the "Register for new users" button (Figure 1) to bring you to the Register Page (Figure 2).

3. The Register Page allows you to register as "Student" or "Staff". Based on Figure 2, we will be registered as "Staff" under the name "Tony Stark".

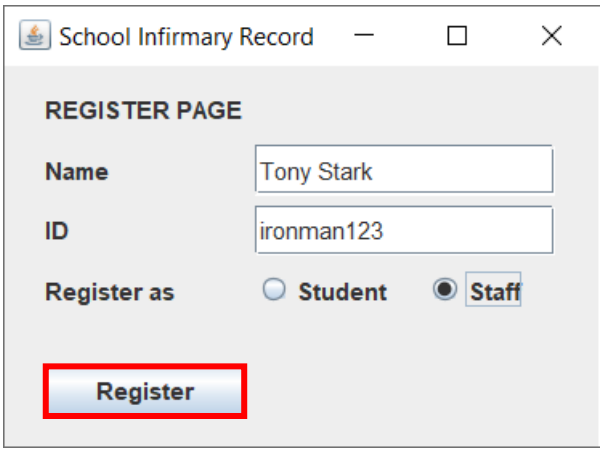

Figure 2: The Register Page

3. Clicking on the "Register" button (Figure 2) will add your name into a text file called "Users". The app will then bring you back to the Login Page (Figure 1).

4. Enter the registered name and ID, then click on the "Login" button (Figure 3). If registered as "Staff", the app will bring you to the Staff Page (Figure 4). If registered as "Student", the app will bring you to the Student Records Page (Figure 11).

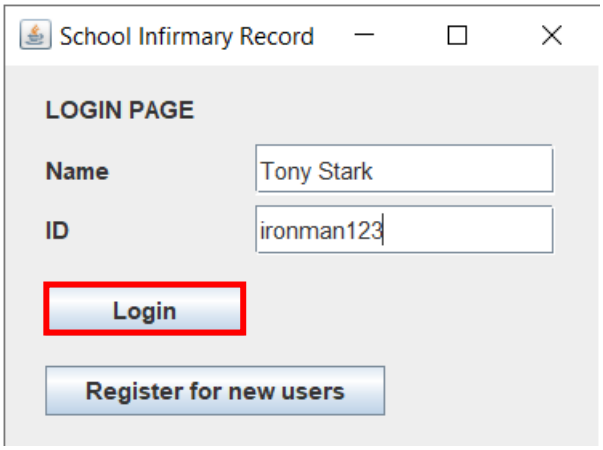

Figure 3: Fill in the empty fields and press the "Login" button

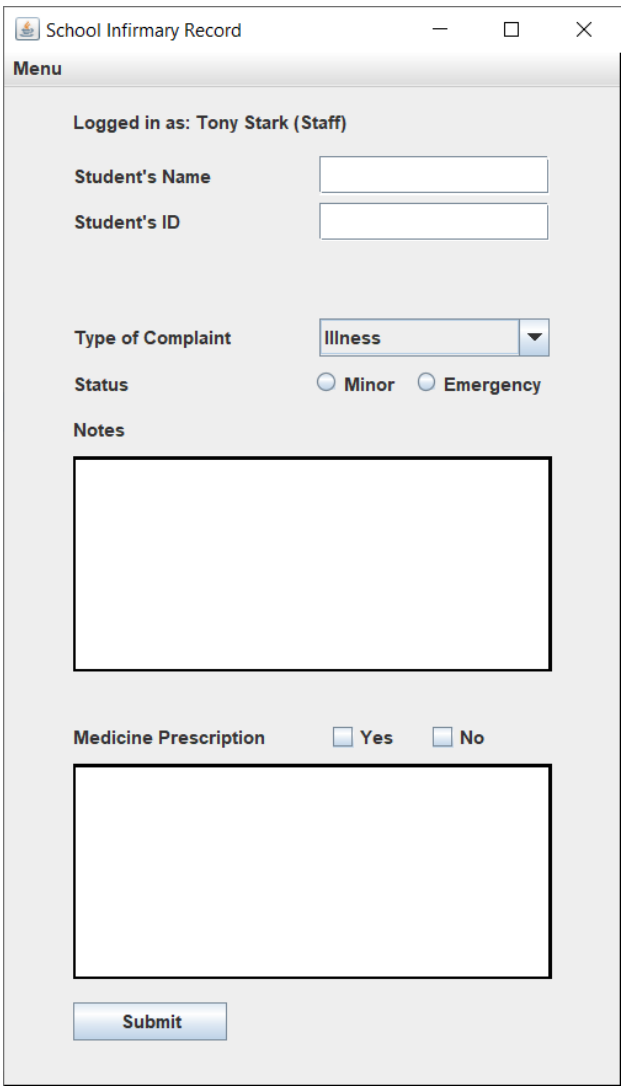

Figure 4: The Staff Page

5. After logging in as "Staff", entering a student's name and ID will automatically register that individual as a "Student". The staff can then record the student's medical complaints and prescribe medicine to them if necessary (Figure 5).

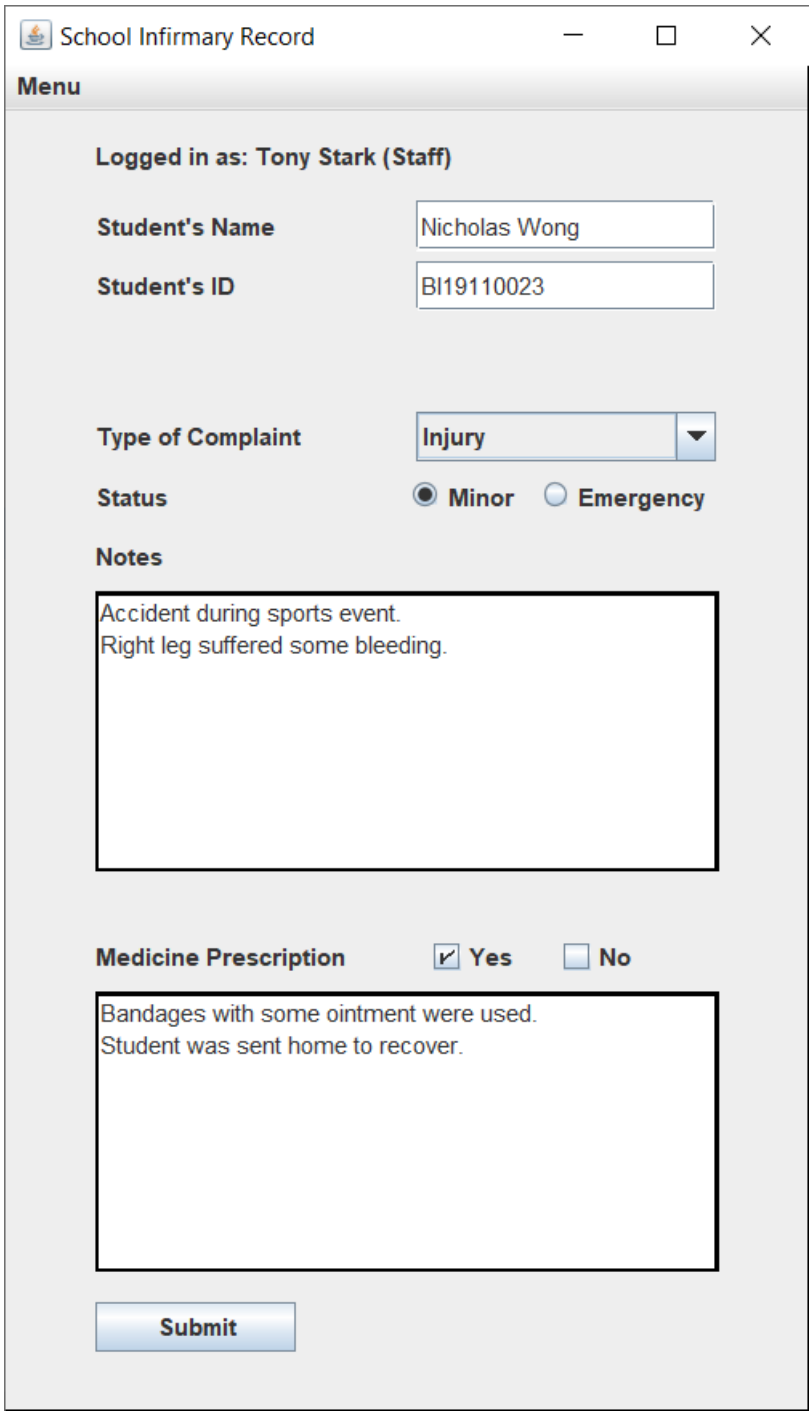

Figure 5: Fill in the details into the empty fields

6. Click on the "Submit" button to finalise the record (Figure 6). The name "Nicholas Wong" will then be added into the "Users" text file. The student will be able to login to the app to view their past records. The recorded information will be stored into a text file called "Records".

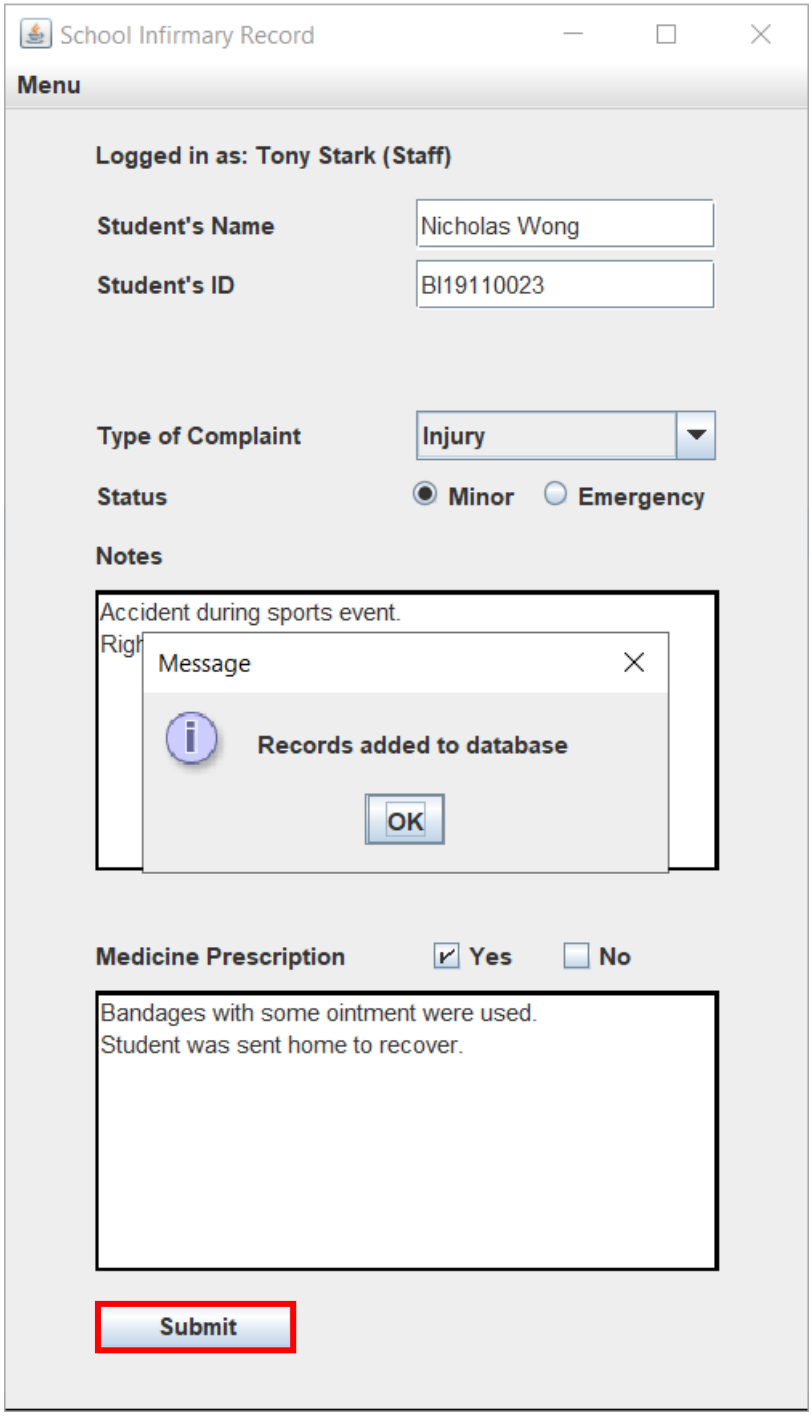

Figure 6: Pressing the "Submit" button will finalise the appointment

7. To view previous infirmary records, go to the top left corner of the app and click on the "Menu" button. Select "Infirmary Records" (Figure 7).

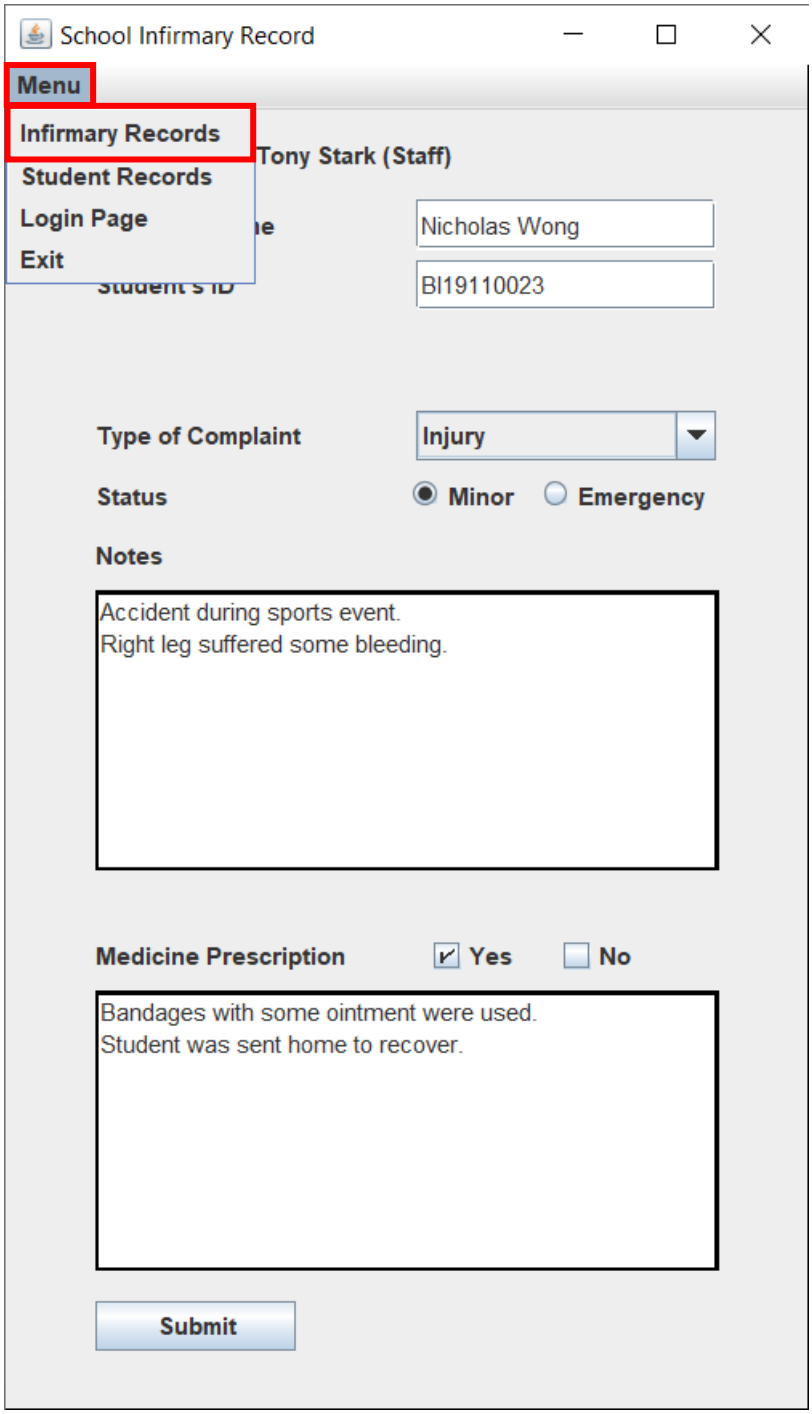

Figure 7: Click on the "Menu" button and select "Infirmary Records"

8. The Infirmary Records Page will be visible and display all previous records (Figure 8).

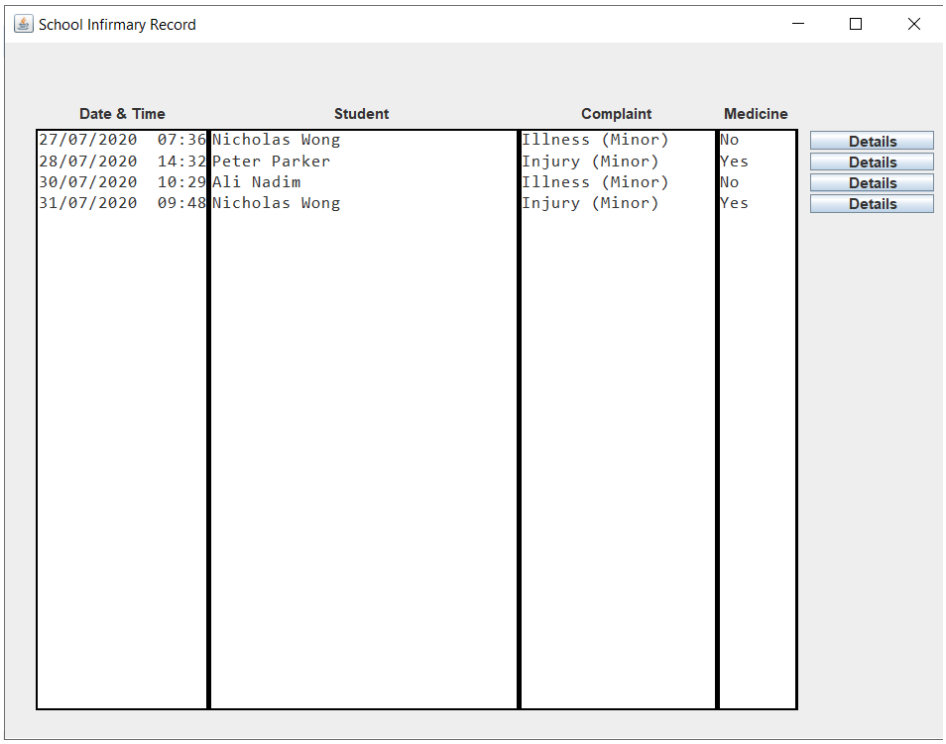

Figure 8: The Infirmary Records Page

9. Clicking on the "Details" button will display all relevant information pertaining to the record.

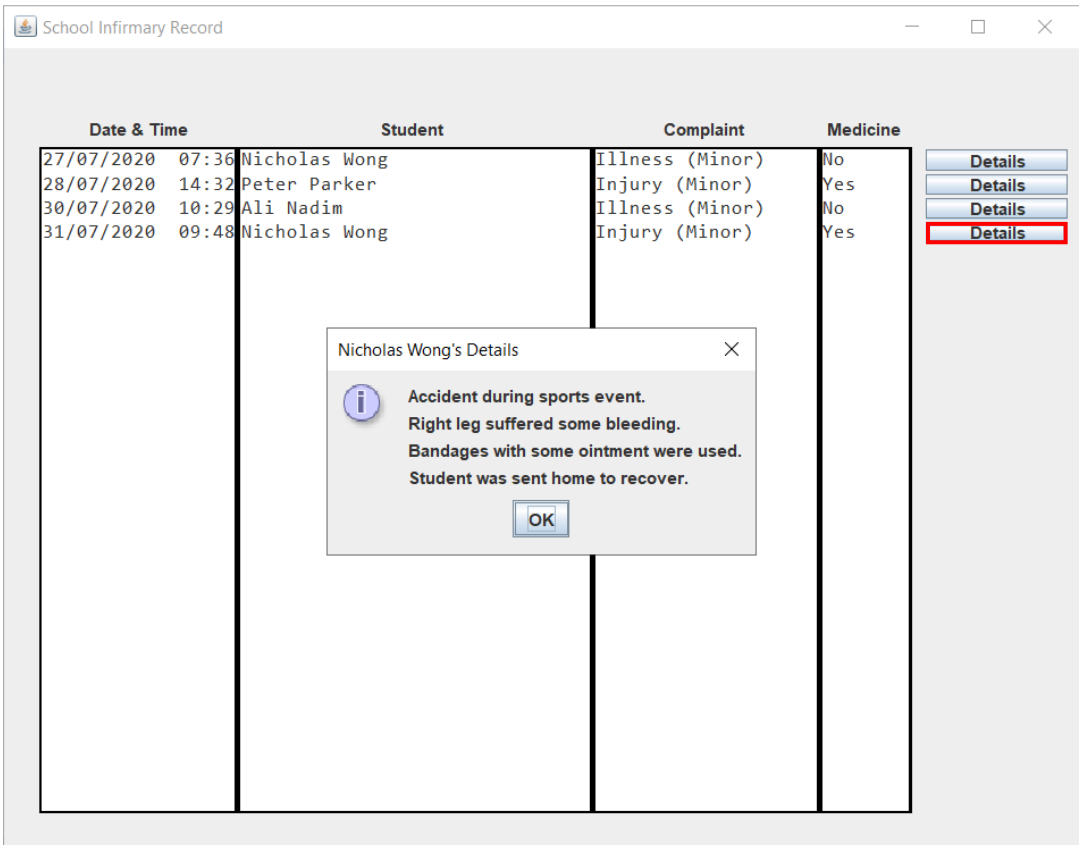

Figure 9: The "Details" button will show further information

10. If a staff member wishes to view a particular student's records, select "Student Records" from the "Menu". Then, fill in the ID of the student you wish to view and click "Enter".

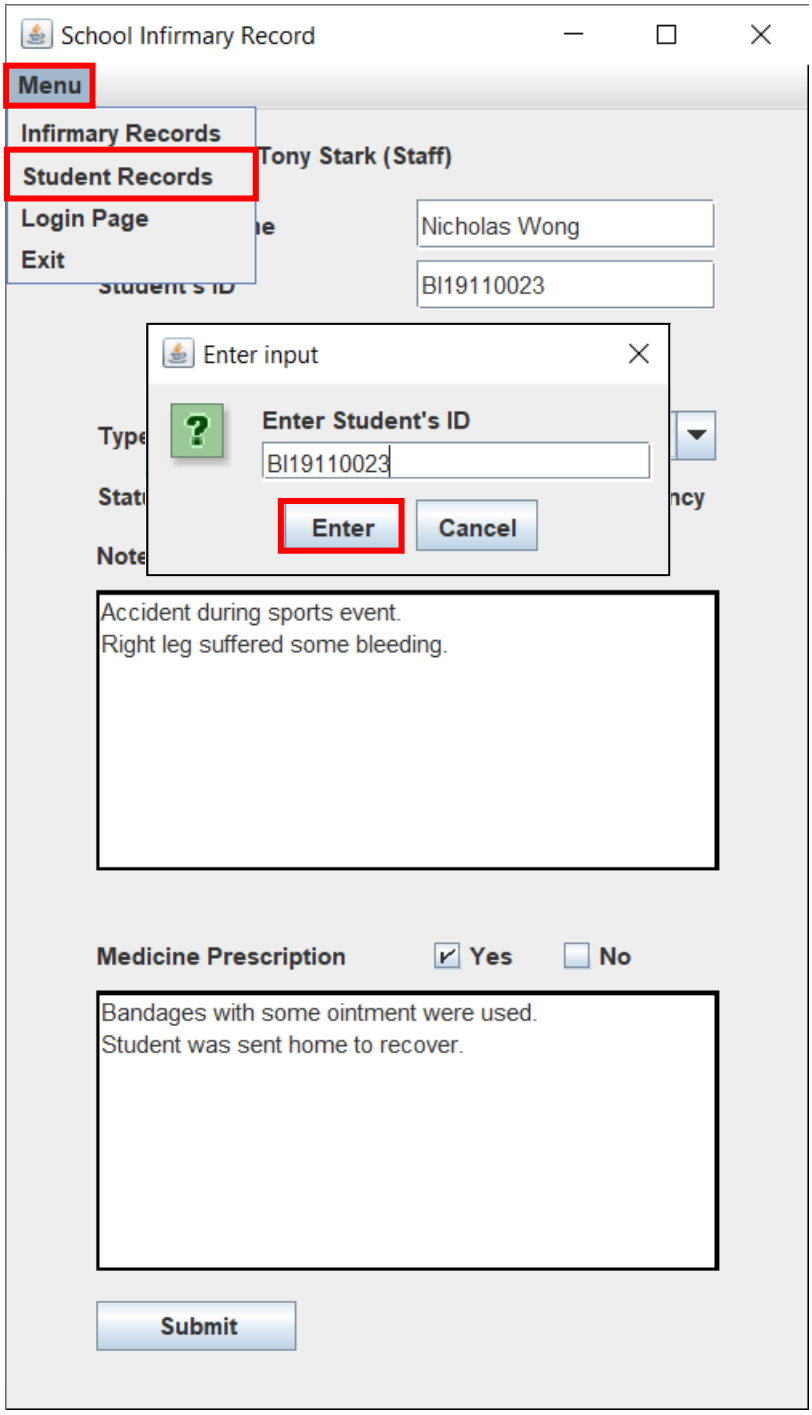

Figure 10: Select "Student Records" from the "Menu" and enter the student's ID

11. The Student Records Page will be visible and display all of the student's previous records (Figure 11). The "Details" button works similarly to the one in the Infirmary Records Page (Figure 9).

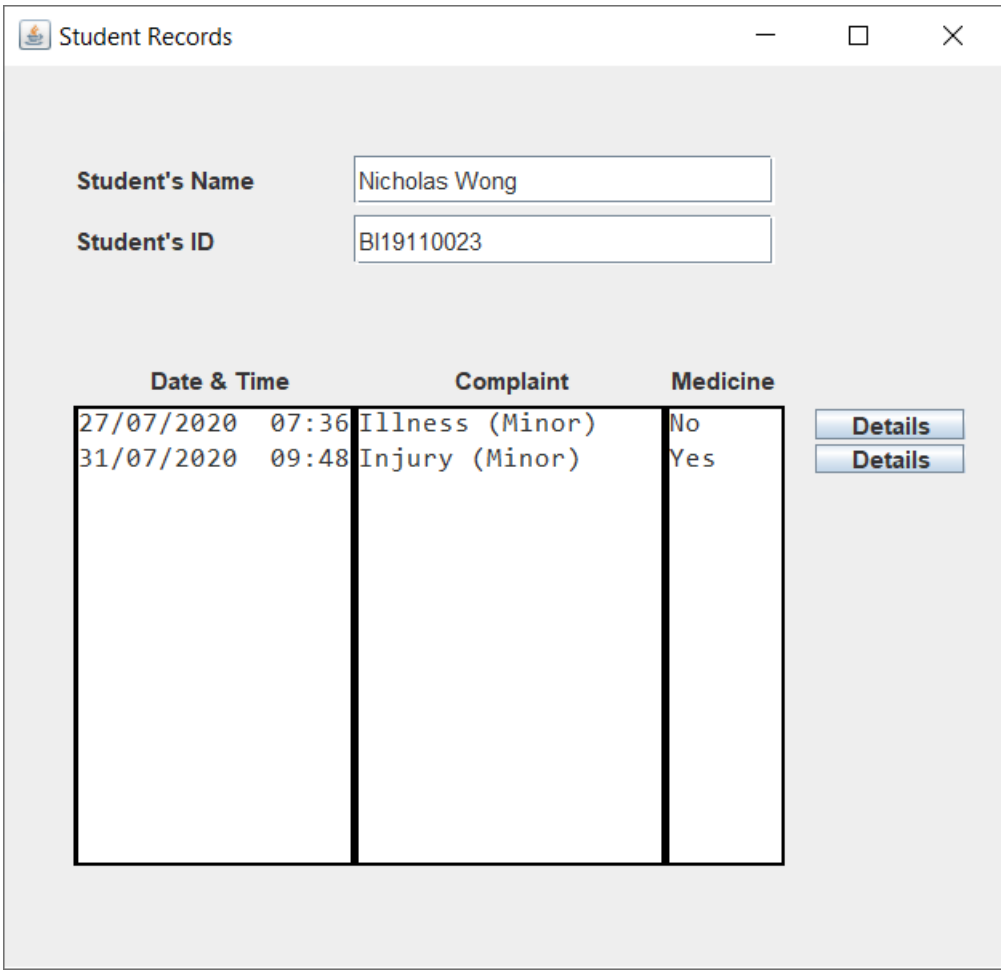

Figure 11: The Student Records Page

12. In the "Menu", select "Login Page" if you wish to return to the Login Page (Figure 1), or select "Exit" to exit the program.

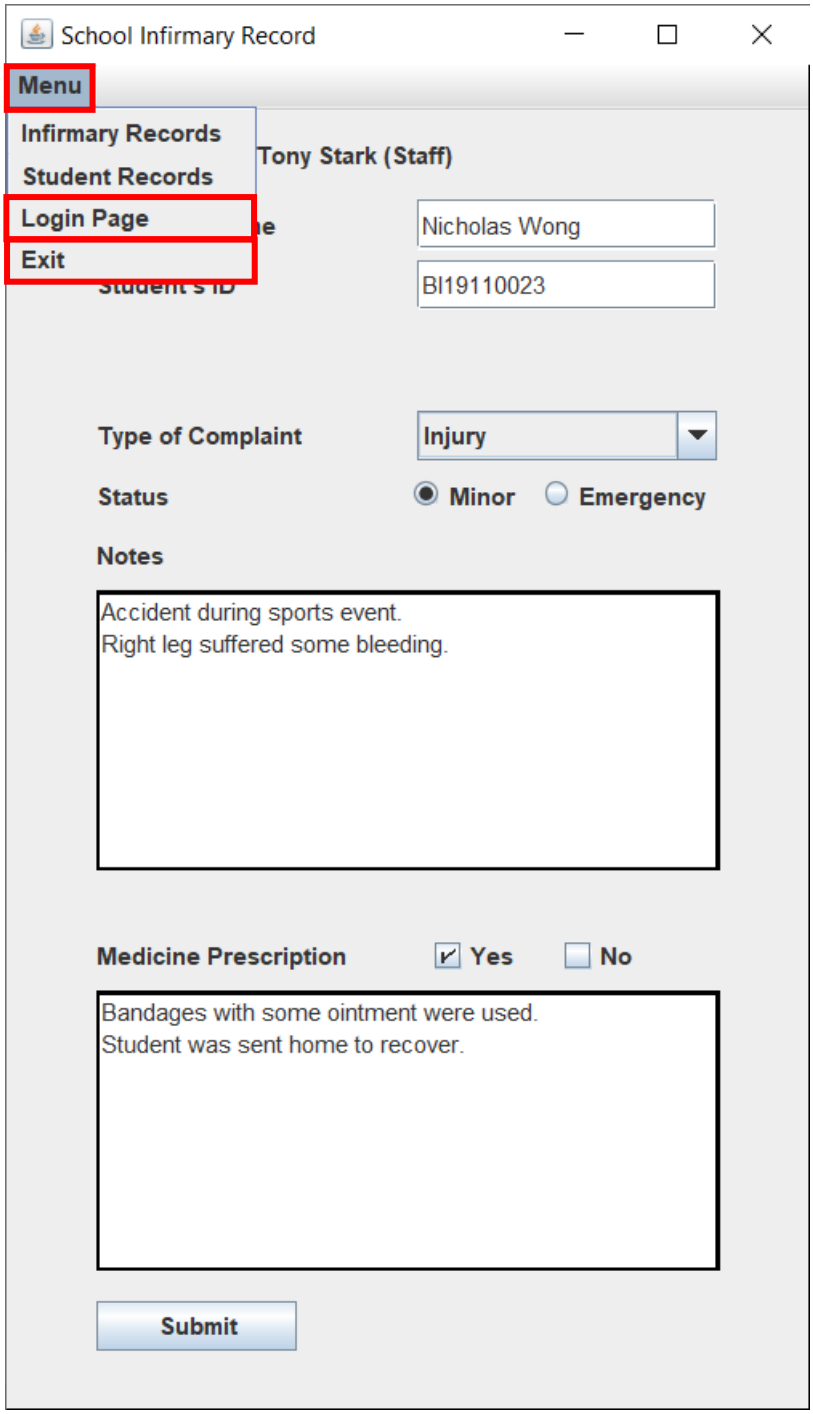

Figure 12: Select "Login Page" or "Exit"

Object Oriented Concept Implementation

1. Objects & Classes

This app uses objects and classes to group similar methods and variables. Here are some examples of the classes used in the program:

`LoginFrame, LoginPanel, RegisterFrame, RegisterPanel, StaffFrame, StaffPanel, StudentFrame, StudentPanel, RecordsFrame, RecordsPanel.`

Classes containing the name **"`Frame`"** inherits the **`JFrame`** class, while classes containing the name **"`Panel`"** inherits the **`JPanel`** class. The **"`Frame`"** classes build the frames by adding the panels. The **"`Panel`"** classes build the **`JPanel`** components such as buttons and text fields.

The **`main()`** method is found in the class called **"`School_Infirmary_Record`"**, and contains the following code to start the program: **`new LoginFrame();`**

2. Encapsulation

All variables that are defined in the classes are private members. They can be accessed and modified by some public methods.

For example, in the **"`LoginPanel`"** class, a **`JButton`** type called **"`btnRegister`"** was declared as a private member. A getter method **"`getBtnRegister()`"** was made to return the **"`btnRegister`"**. This allows the **"`LoginFrame`"** class to access the **"`btnRegister`"** and apply an action listener to it. If the button is pressed, the current frame **"`LoginFrame`"** will be set to be not visible.

This action listener cannot be applied within the **"`Panel`"** class as it has no access towards the **"`Frame`"** class. However, once this action listener is applied in the **"`Frame`"** class, it can use the **"`getBtnRegister()`"** method to access the **"`btnRegister`"** and allow the current frame itself to be hidden using the code: **`this.setVisible(false);`**

3. Inheritance

The program also uses inheritance to avoid duplicates of the same methods. One such example is the **"FileUsers"** class. It is a **superclass** and it contains methods to read and write text files. Its **subclasses** include the **"LoginPanel"**, **"RegisterPanel"**, **"StaffPanel"**, **"StudentPanel"**, and **"RecordsPanel".** These panels require the use for read and write files. Since they are the **subclasses**, they can access the methods that are present in the **superclass**. Thus, avoiding the need for duplicate methods.

4. Abstraction

Abstraction is also one of the concepts utilised in the program. One such example is the abstract class **"Frames"**. This abstract class includes the abstract method **"initialiseFrame()"**. The classes that inherits this abstract class include the **"LoginFrame"**, **"RegisterFrame"**, and **"RecordsFrame"**. Each of these **"Frame"** classes will initialise their frames differently depending on which **"Panel"** class they wish to create the object from.

5. Interface

Finally, an interface called **"ButtonDetails"** was implemented in the program. It contains some abstract methods such as **"makeBtn()"**, **"btnEvent()"**, and **"printDetails()"**. Classes that implements this interface are the **"RecordsPanel"** and **"StudentPanel"**. The **"makeBtn()"** method will create a new **JButton** component each time a new record is submitted. This method will set the position of each newly created button. The **"btnEvent()"** method will create an action listener for each of the buttons. And finally, the **"printDetails()"** method will read from the text file **"Records.txt"** to display it in the **Message Dialog Box**, after the button is clicked.

Read and Write Implementation

Two text files are used in this program, **"Users.txt"** and **"Records.txt"**. They can be found in the folder which contains the Java classes. If they are not present in the folder, starting the program will automatically create those text files.

In the superclass **"FileUsers"**, the method **"addToUsers()"** will accept the arguments for the user's name, ID, and user type (Staff or Student). It will then write the data into the **"Users.txt"** file. The method **"addToRecords()"** will accept the arguments for the user's name, ID and the medical complaints, and will write the data into the **"Records.txt"** file. This method will also add a **"*"** to the last line for each submitted record to mark the end of writing the file. Both of these methods use **PrintWriter** and **FileWriter** to write data into the text files.

The method **"getFromFile()"** will accept an argument for the name of the text file we wish to read from. For example, in the **"LoginPanel"** class, to get the text from the **"Users.txt"** file, the following code is used:

```
private ArrayList<String> textUser = new ArrayList<String>();
textUser = (ArrayList<String>) getFromFile("Users.txt");
```

The same goes in the **"RecordsPanel"** class. To get the text from the **"Records.txt"** file, the following code is used:

```
private ArrayList<String> textRecord = new ArrayList<String>();
textRecord = (ArrayList<String>) getFromFile("Records.txt");
```

The read data is cast into an **ArrayList<String>** type to allow the program to easily search for the **String** data by accessing the indexes of the **ArrayList**. The **ArrayList** also contains useful methods such as **get()**, **contains()**, **indexOf()** to allow ease in searching for data. When getting data from the **textRecord ArrayList**, it will return each data until the **ArrayList** returns a **"*"**, signifying the end of the record.

The app comes with default registered users and records, as seen in the User Manual. To reset the app and restore it into a blank state, please delete from the folder the existing text files: **Users.txt**
Records.txt