Criterion C: Development

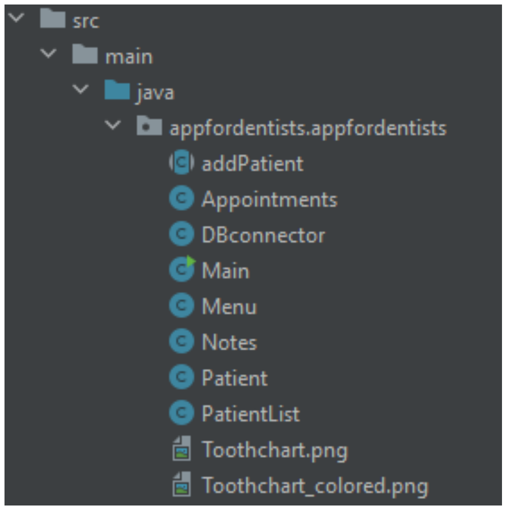
Used Techniques:

1. SQL (Structured Query Language) is used for communication between the program and the database.
2. JavaFX is used to work with different windows, create control and graphical components and place them in the window.
3. FileInputStream and ImageView are used to read images; pixelReader is used to get the color of a pixel
4. Use of additional libraries
5. User-defined objects
6. User-defined methods (with parameters, with return values)
7. Different data types, arrays
8. Loops
9. Nested loops
10. Simple and complex selections.
11. Searching.
12. Sorting.
13. Use of flags.

Program file:

AppForDentists.jar

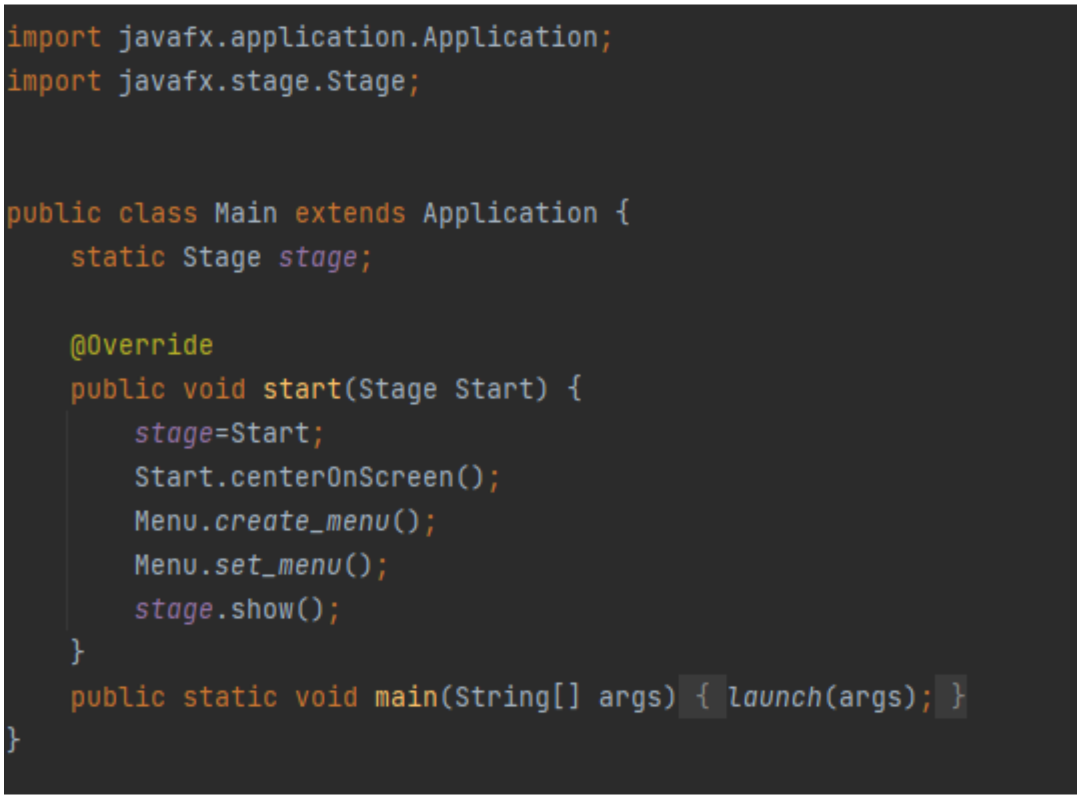
Class structure:



```
src
  main
    java
      appfordentists.appfordentists
        addPatient
        Appointments
        DBconnector
        Main
        Menu
        Notes
        Patient
        PatientList
        Toothchart.png
        Toothchart_colored.png
```

Figure 1: class structure

Main class:



```java
import javafx.application.Application;
import javafx.stage.Stage;


public class Main extends Application {
    static Stage stage;

    @Override
    public void start(Stage Start) {
        stage=Start;
        Start.centerOnScreen();
        Menu.create_menu();
        Menu.set_menu();
        stage.show();
    }
    public static void main(String[] args) { launch(args); }
}
```

Figure 2: Main class

The program starts with $main()$ method. It calls $launch(args)$ method which starts JavaFX application and calls $start()$ function. Global variable $stage$ is crated, which is the primary window of the application. $Menu.create\_menu()$ and $Menu.set\_menu()$ are called and then $stage$ is displayed on the screen. Main class extends Application to use $start()$ and $launch()$ functions.

Menu class:

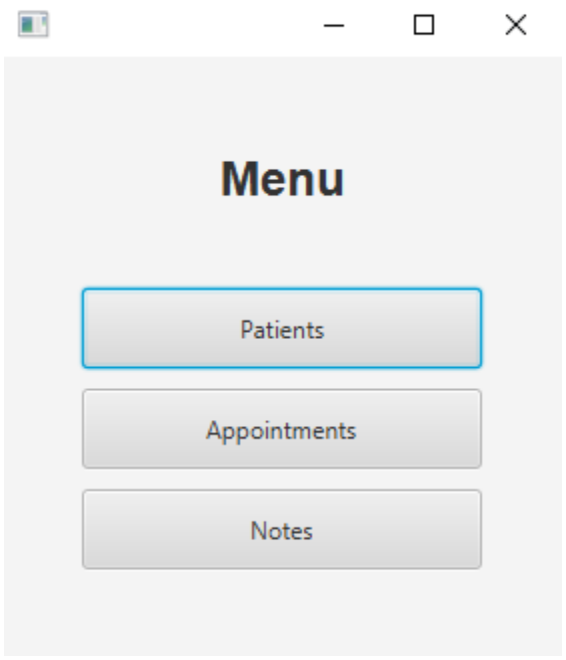
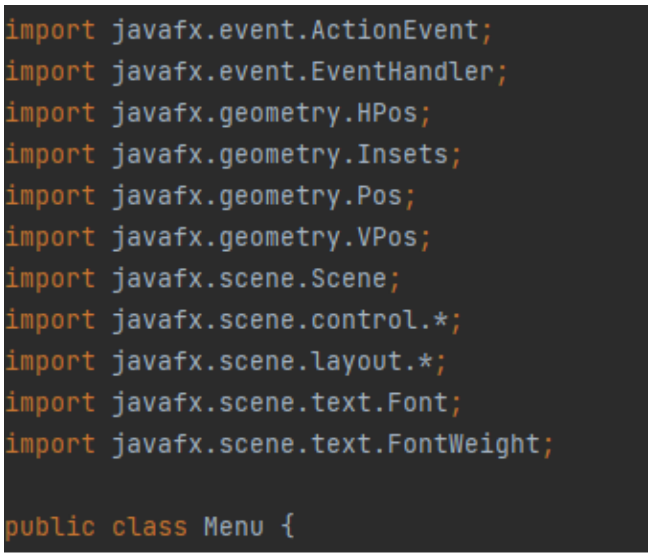


Figure 3: Menu page

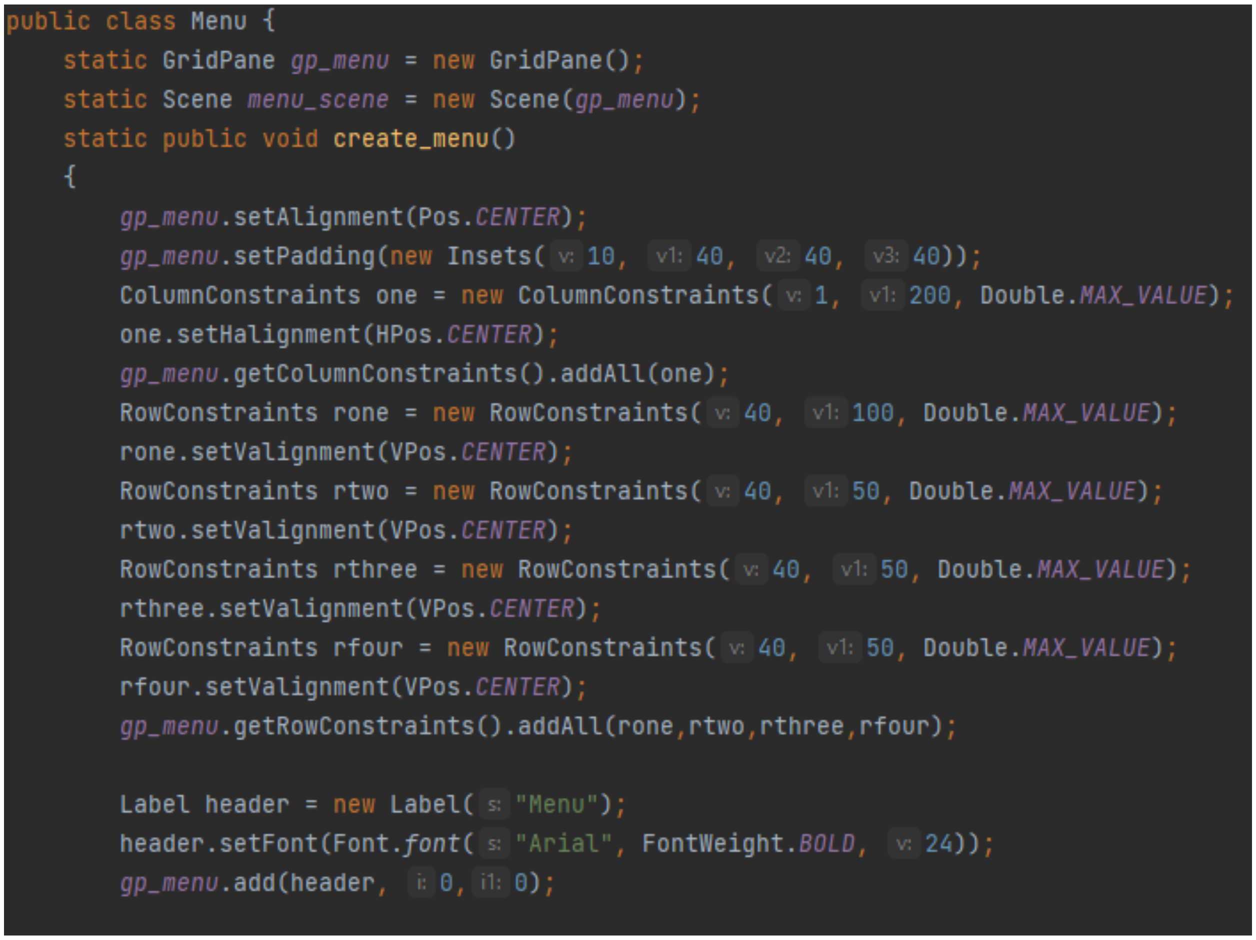
Menu page is displayed at the start of the program.



```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;

public class Menu {
```

Figure 4: imported libraries in Menu class

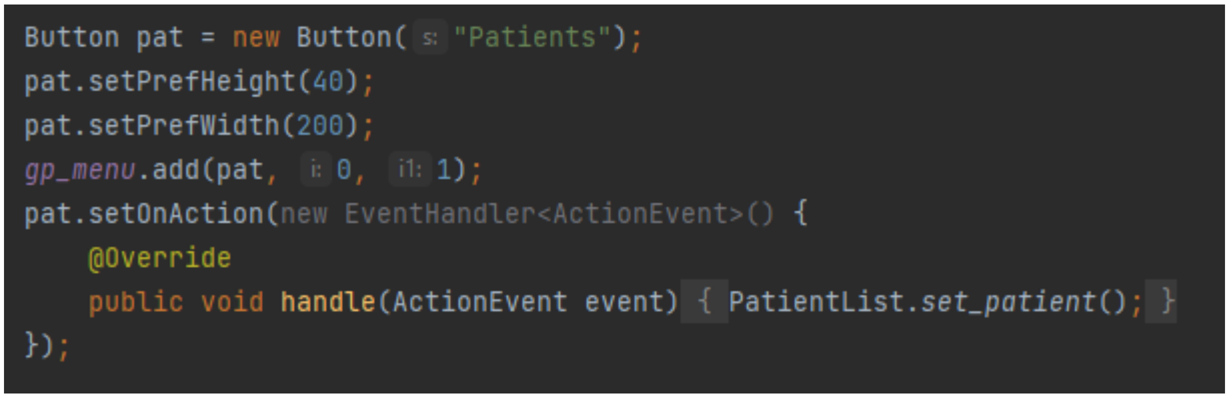
Menu.create_menu():



```java
public class Menu {
    static GridPane gp_menu = new GridPane();
    static Scene menu_scene = new Scene(gp_menu);
    static public void create_menu()
    {
        gp_menu.setAlignment(Pos.CENTER);
        gp_menu.setPadding(new Insets( v: 10,  v1: 40,  v2: 40,  v3: 40));
        ColumnConstraints one = new ColumnConstraints( v: 1,  v1: 200, Double.MAX_VALUE);
        one.setHalignment(HPos.CENTER);
        gp_menu.getColumnConstraints().addAll(one);
        RowConstraints rone = new RowConstraints( v: 40,  v1: 100, Double.MAX_VALUE);
        rone.setValignment(VPos.CENTER);
        RowConstraints rtwo = new RowConstraints( v: 40,  v1: 50, Double.MAX_VALUE);
        rtwo.setValignment(VPos.CENTER);
        RowConstraints rthree = new RowConstraints( v: 40,  v1: 50, Double.MAX_VALUE);
        rthree.setValignment(VPos.CENTER);
        RowConstraints rfour = new RowConstraints( v: 40,  v1: 50, Double.MAX_VALUE);
        rfour.setValignment(VPos.CENTER);
        gp_menu.getRowConstraints().addAll(rone,rtwo,rthree,rfour);

        Label header = new Label( s: "Menu");
        header.setFont(Font.font( s: "Arial", FontWeight.BOLD,  v: 24));
        gp_menu.add(header,  i: 0, i1: 0);
```

Figure 5: *create_menu*() method

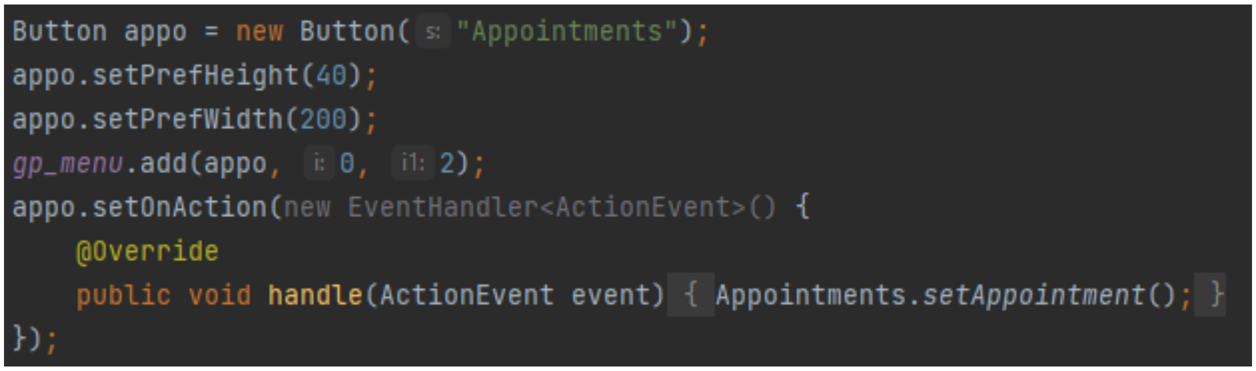
First, it creates GUI of the Menu page and stores them in *gp_menu* which is contained by *menu_scene* scene.



```java
Button pat = new Button( s: "Patients");
pat.setPrefHeight(40);
pat.setPrefWidth(200);
gp_menu.add(pat,  i: 0,  i1: 1);
pat.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) { PatientList.set_patient(); }
});
```

Figure 6: Button “Patients”

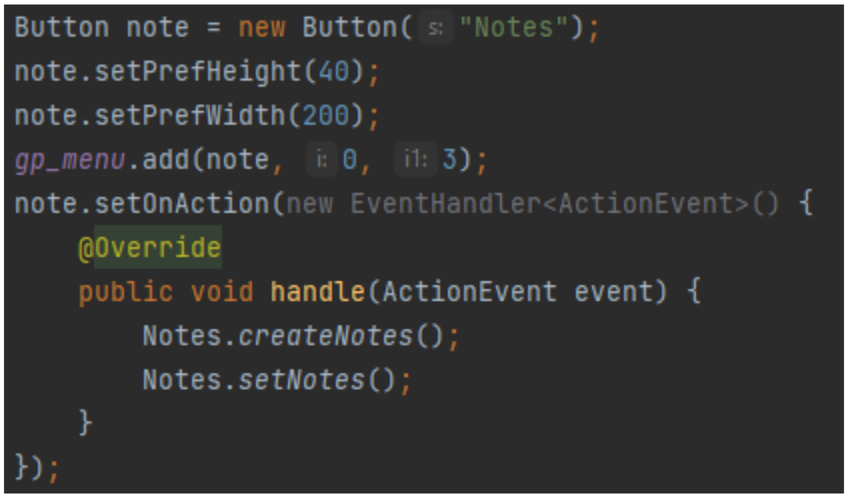
“Patients” button is created. When the button is pressed, $PatientList.set\_patient()$ is called and PatientList page is displayed.



```java
Button appo = new Button( s: "Appointments");
appo.setPrefHeight(40);
appo.setPrefWidth(200);
gp_menu.add(appo, i: 0, i1: 2);
appo.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) { Appointments.setAppointment(); }
});
```

Figure 7: Button “Appointments”

When the button is pressed, $Appointments.setAppointment()$ is called and Appointments page is displayed.

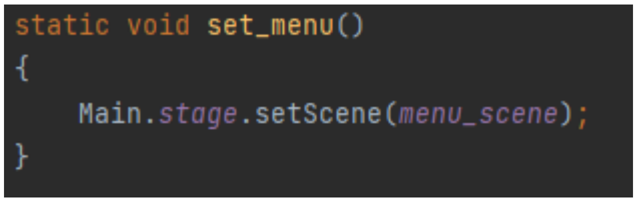


```java
Button note = new Button( s: "Notes");
note.setPrefHeight(40);
note.setPrefWidth(200);
gp_menu.add(note, i: 0, i1: 3);
note.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        Notes.createNotes();
        Notes.setNotes();
    }
});
```

Figure 8: Button “Notes”

When the button is pressed, $Notes.createNotes()$ and $Notes.setNotes()$ are called and Notes page is displayed.

$Menu.set\_menu()$:



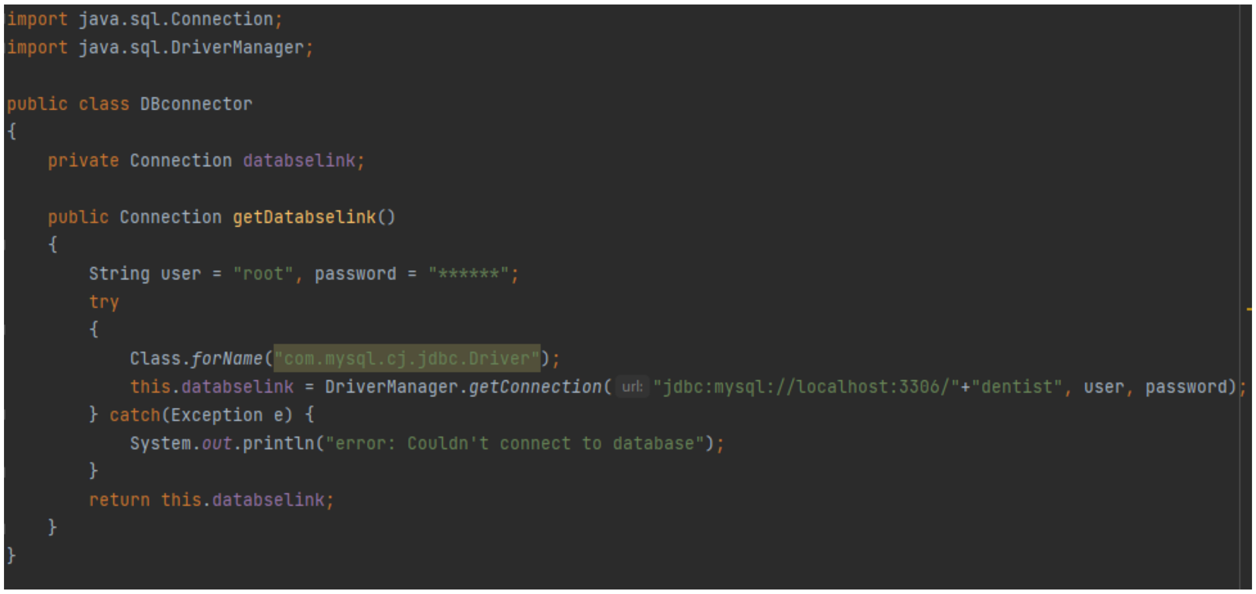
```
static void set_menu()
{
    Main.stage.setScene(menu_scene);
}
```

Figure 9: $set\_menu()$ method

This method sets the primary window to Menu page.

DBconnector class:

As I mentioned in criterions A and B the program uses a database to store the data of patients. SQL commands are used to communicate with the database.



```
import java.sql.Connection;
import java.sql.DriverManager;

public class DBconnector
{
    private Connection databselink;

    public Connection getDatabselink()
    {
        String user = "root", password = "******";
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            this.databselink = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/"+"dentist", user, password);
        } catch(Exception e) {
            System.out.println("error: Couldn't connect to database");
        }
        return this.databselink;
    }
}
```

Figure 10: DBconnector class

The method $DBconnector.getDatabaselink()$ returns value of a connection to the database called “dentist”. The database is stored locally. If any problem occurs, the program will output "error: Couldn't connect to database."

Notes class:

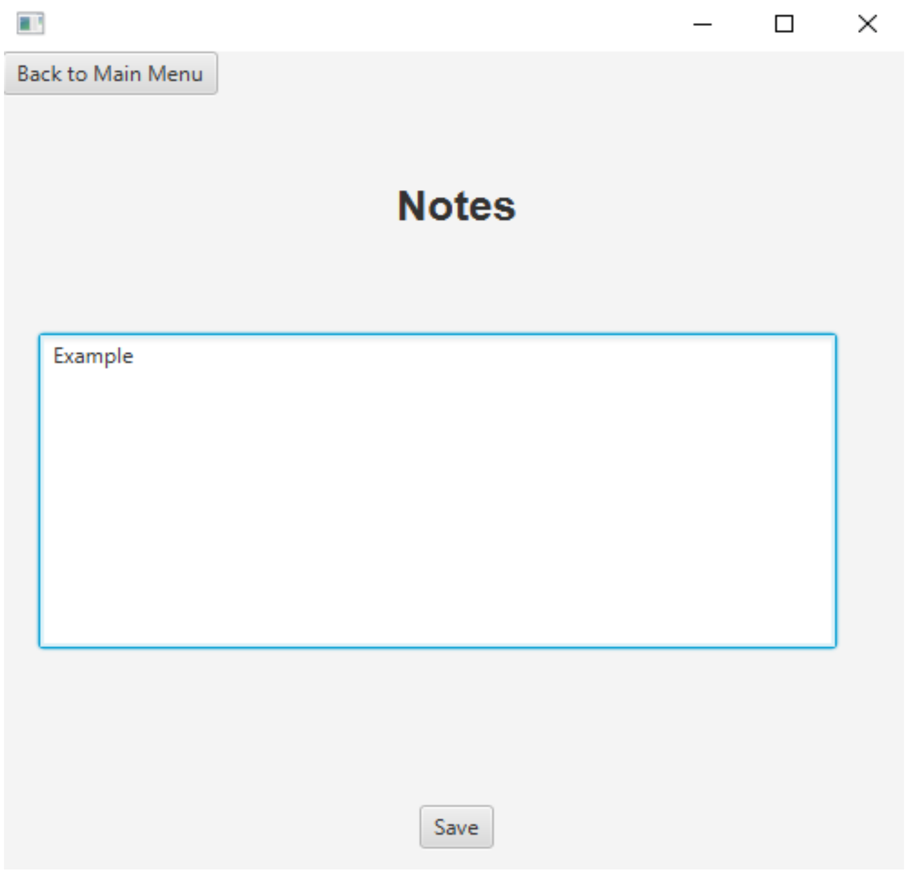
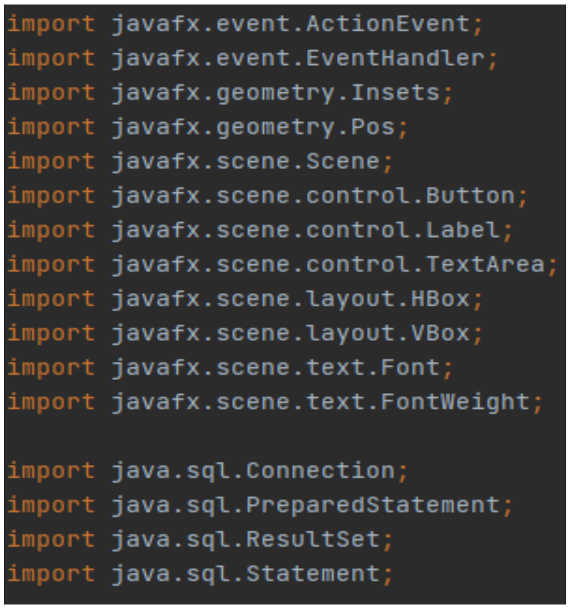
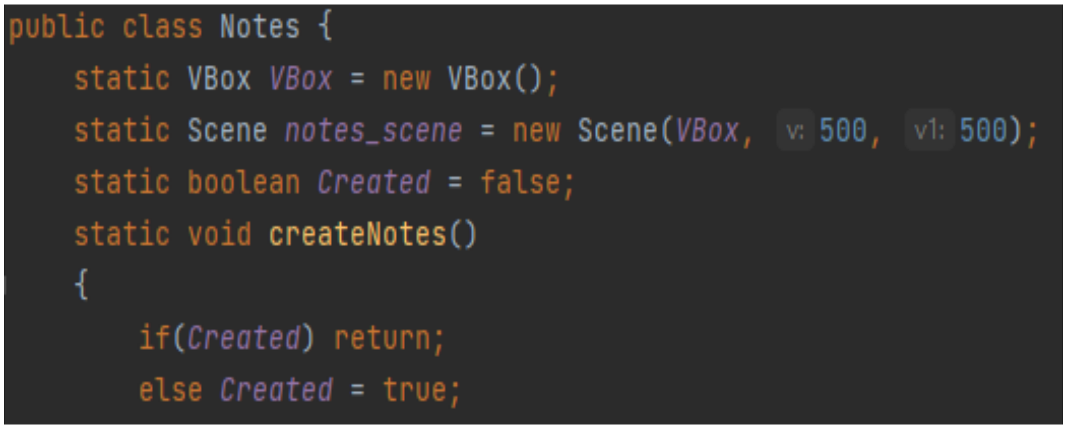


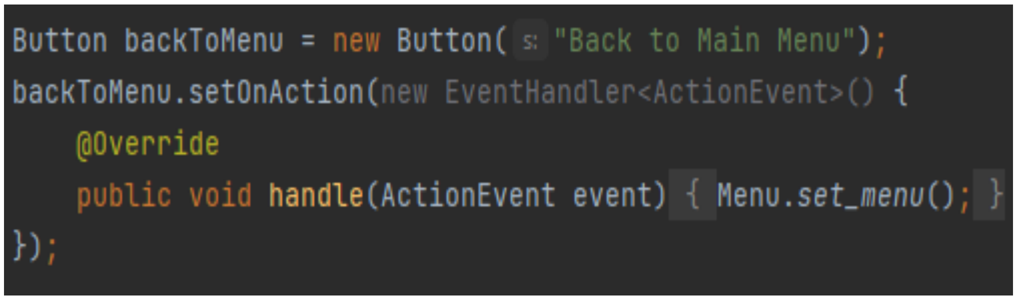
Figure 11: Notes page


```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
```
Figure 12: Imported libraries in Notes class

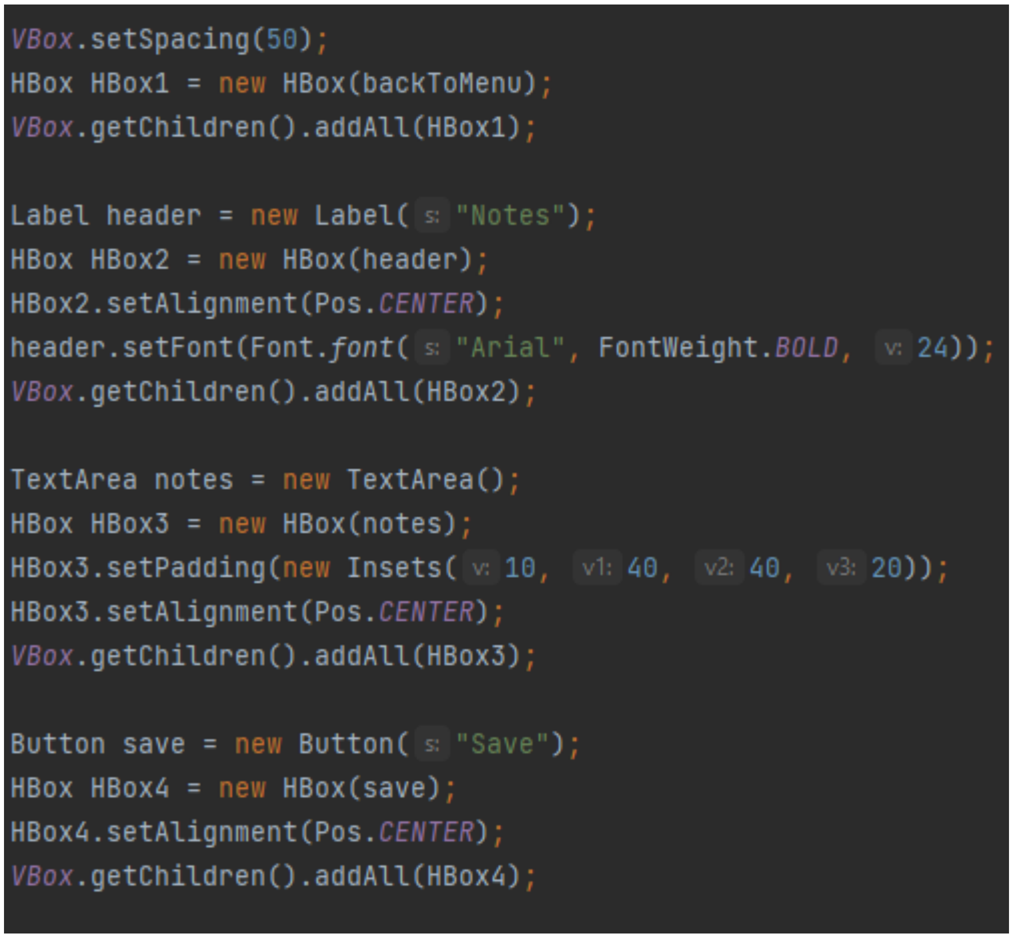
$Notes.createNotes()$:


```
public class Notes {
    static VBox VBox = new VBox();
    static Scene notes_scene = new Scene(VBox,  v: 500,  v1: 500);
    static boolean Created = false;
    static void createNotes()
    {
        if(Created) return;
        else Created = true;
```
Figure 13: Notes class

$Notes.createNotes()$ constructs the Notes page. Graphical components are created and stored in $notes\_scene$. The boolean variable $Created$ is a flag that shows whether or not $Notes.createNotes()$ has already been invoked. If $Created$ is true, it signifies that the Notes page has already been produced and is stored in $notes\_scene$, therefore the method $Notes.createdNote()$ is skipped. If $Created$ is false, it becomes true.

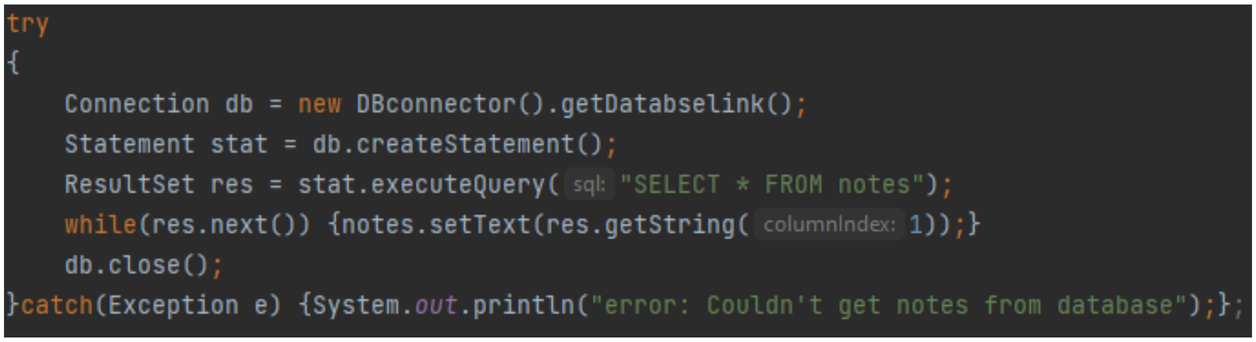

```
Button backToMenu = new Button( s: "Back to Main Menu");
backToMenu.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) { Menu.set_menu(); }
});
```
Figure 14: Button “Back to Main Menu”

When the button is clicked, $Menu.set\_menu()$ is called so the primary window is set to Menu page.



```java
VBox.setSpacing(50);
HBox HBox1 = new HBox(backToMenu);
VBox.getChildren().addAll(HBox1);

Label header = new Label( s: "Notes");
HBox HBox2 = new HBox(header);
HBox2.setAlignment(Pos.CENTER);
header.setFont(Font.font( s: "Arial", FontWeight.BOLD,  v: 24));
VBox.getChildren().addAll(HBox2);

TextArea notes = new TextArea();
HBox HBox3 = new HBox(notes);
HBox3.setPadding(new Insets( v: 10,  v1: 40,  v2: 40,  v3: 20));
HBox3.setAlignment(Pos.CENTER);
VBox.getChildren().addAll(HBox3);

Button save = new Button( s: "Save");
HBox HBox4 = new HBox(save);
HBox4.setAlignment(Pos.CENTER);
VBox.getChildren().addAll(HBox4);
```

Figure 15: GUI of the Notes page is created

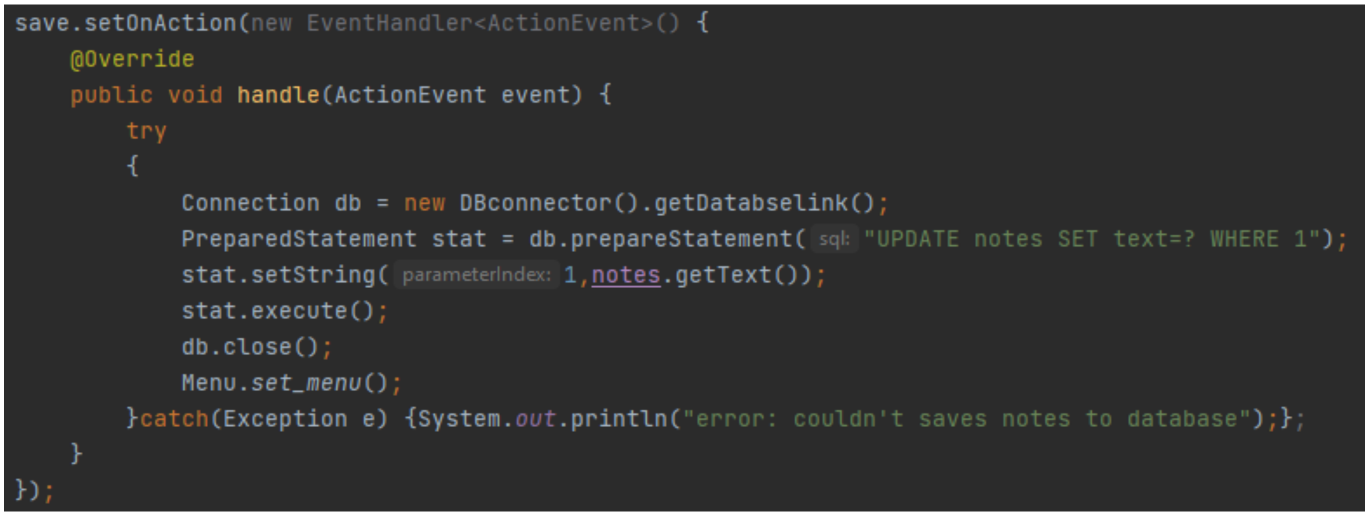


```java
try
{
    Connection db = new DBconnector().getDatabselink();
    Statement stat = db.createStatement();
    ResultSet res = stat.executeQuery( sql: "SELECT * FROM notes");
    while(res.next()) {notes.setText(res.getString( columnIndex: 1));}
    db.close();
}catch(Exception e) {System.out.println("error: Couldn't get notes from database");};
```

Figure 16: Getting the notes stored in database

An SQL command "SELECT * FROM notes" is used to get the data from table notes. As mentioned in criterion B, table notes has only 1 row and 1 column so $while(res.next())$ will be executed only once.

Inside the loop, the text of $notes$ is set to previously saved value, so that the previously saved text will be displayed. If any problem occurs the program will print an error message.

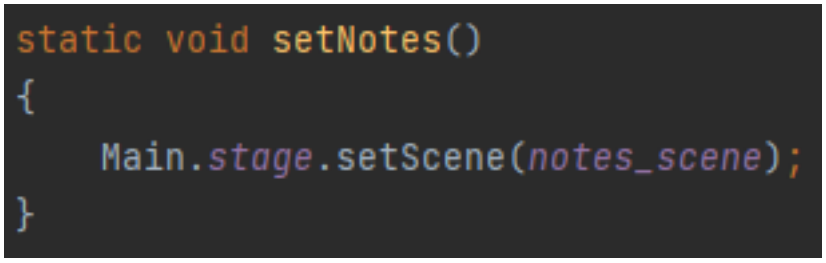


```
save.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        try
        {
            Connection db = new DBconnector().getDatabselink();
            PreparedStatement stat = db.prepareStatement( sql: "UPDATE notes SET text=? WHERE 1");
            stat.setString( parameterIndex: 1,notes.getText());
            stat.execute();
            db.close();
            Menu.set_menu();
        }catch(Exception e) {System.out.println("error: couldn't saves notes to database");};
    }
});
```

Figure 17: Button “Save”

When the button is clicked, an SQL command “UPDATE notes SET text=? WHERE 1” is used to update the value in table notes to the inserted text. Then the Menu page is displayed.

$Notes.setNotes()$:

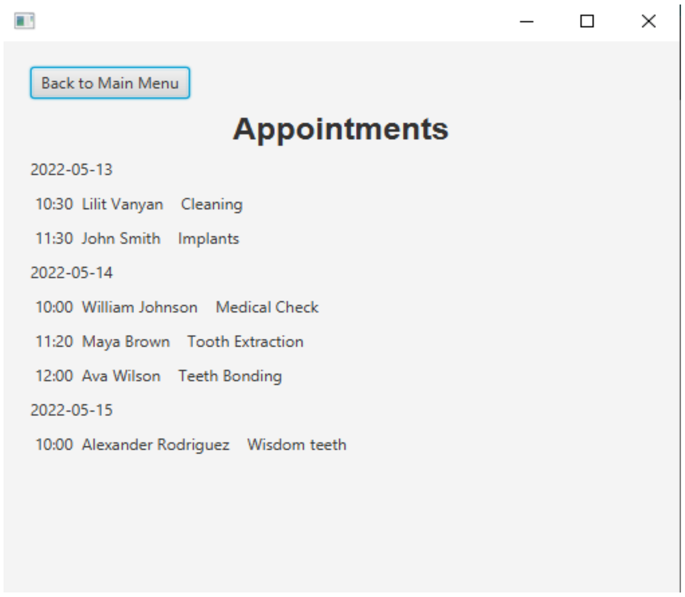


```
static void setNotes()
{
    Main.stage.setScene(notes_scene);
}
```

Figure 18: $setNotes()$ method

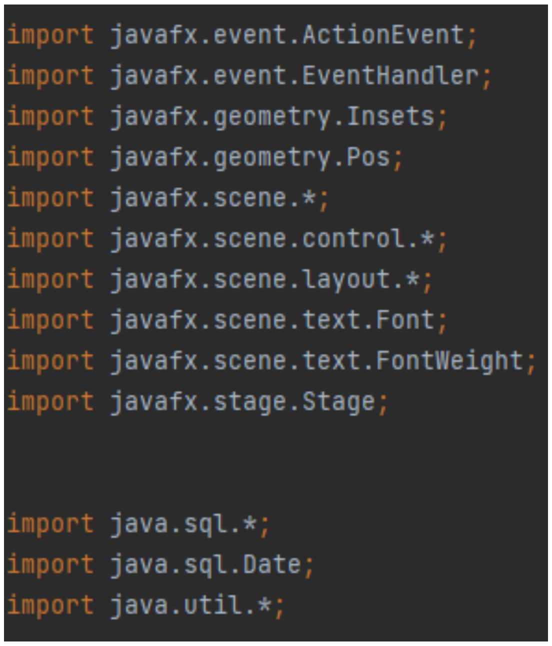
This method sets the primary window to Notes page.

Appointments class:



```
Appointments
2022-05-13
 10:30  Lilit Vanyan    Cleaning
 11:30  John Smith    Implants
2022-05-14
 10:00  William Johnson    Medical Check
 11:20  Maya Brown    Tooth Extraction
 12:00  Ava Wilson    Teeth Bonding
2022-05-15
 10:00  Alexander Rodriguez    Wisdom teeth
```

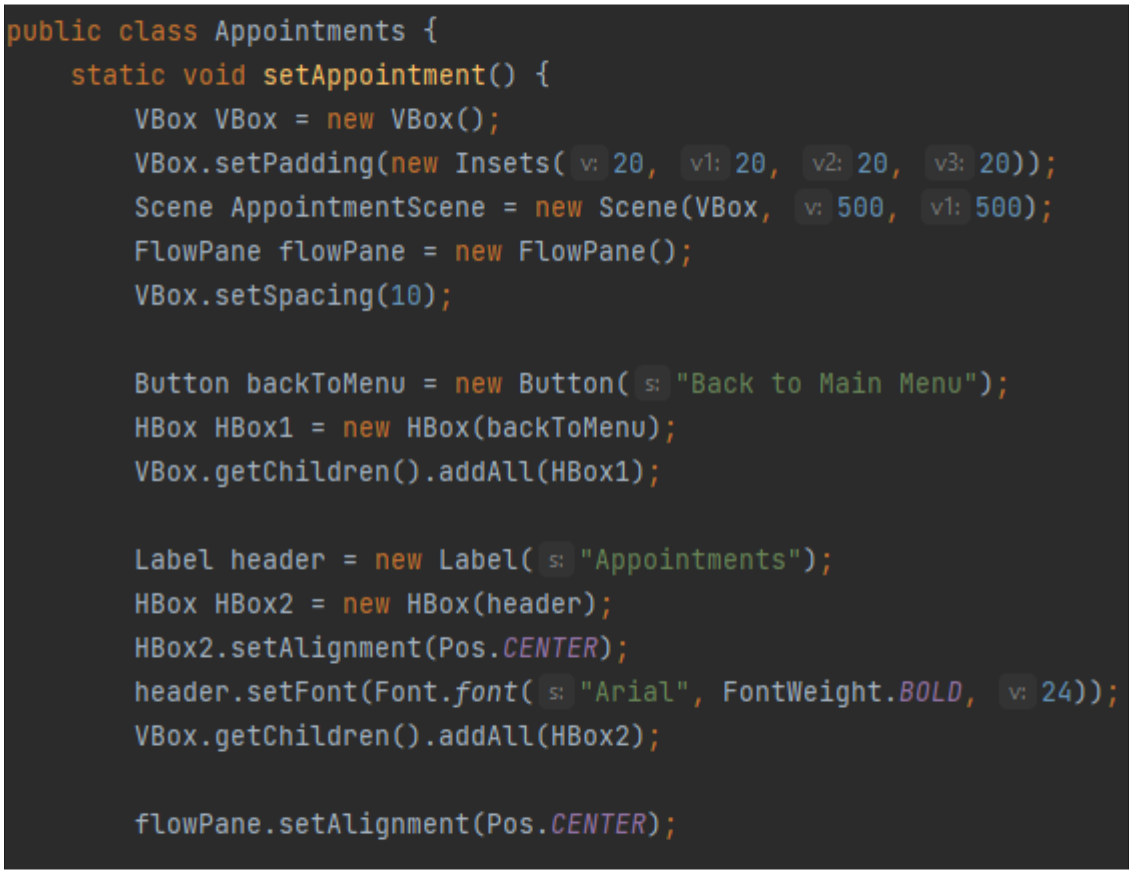
Figure 19: Appointments page



```java
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.Stage;


import java.sql.*;
import java.sql.Date;
import java.util.*;
```

Figure 20: Imported libraries in Appointment class

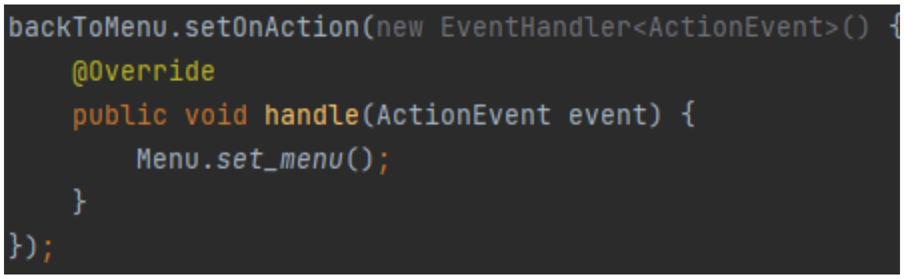
$Appointments.setAppointment()$:



```
public class Appointments {
    static void setAppointment() {
        VBox VBox = new VBox();
        VBox.setPadding(new Insets( v: 20,  v1: 20,  v2: 20,  v3: 20));
        Scene AppointmentScene = new Scene(VBox,  v: 500,  v1: 500);
        FlowPane flowPane = new FlowPane();
        VBox.setSpacing(10);

        Button backToMenu = new Button( s: "Back to Main Menu");
        HBox HBox1 = new HBox(backToMenu);
        VBox.getChildren().addAll(HBox1);

        Label header = new Label( s: "Appointments");
        HBox HBox2 = new HBox(header);
        HBox2.setAlignment(Pos.CENTER);
        header.setFont(Font.font( s: "Arial", FontWeight.BOLD,  v: 24));
        VBox.getChildren().addAll(HBox2);

        flowPane.setAlignment(Pos.CENTER);
```

Figure 21: $Appointments.setAppointment()$ function

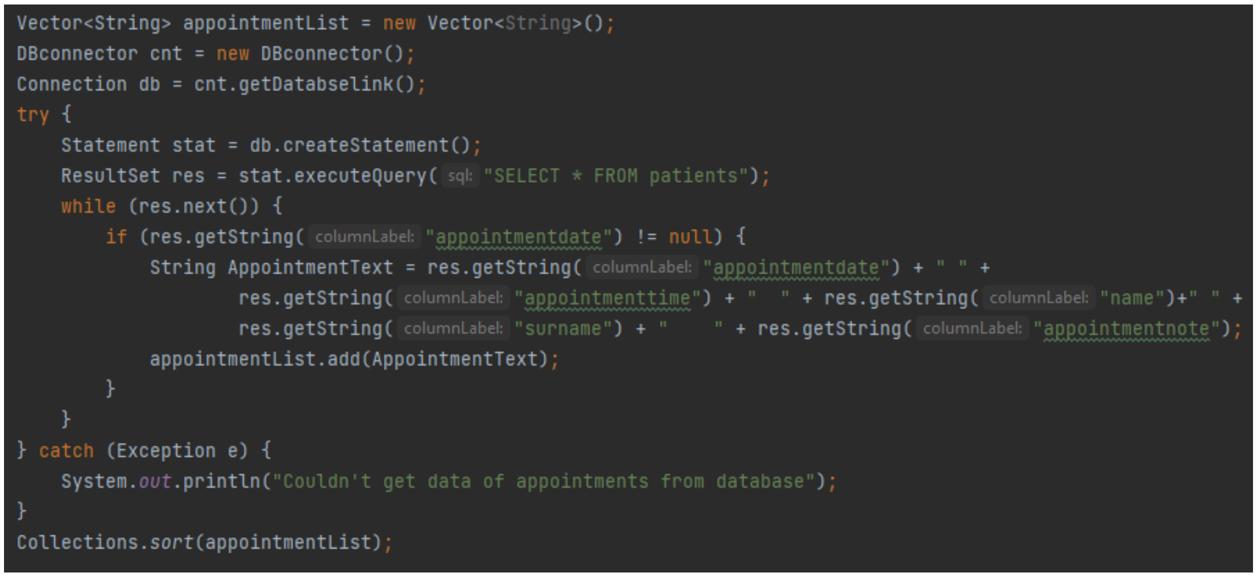
First, it creates GUI of the Appointments page and stores them in $VBox$ which is contained by $AppointmentScene$ scene.



```
backToMenu.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        Menu.set_menu();
    }
});
```

Figure 22: Button “Back to Main Menu”

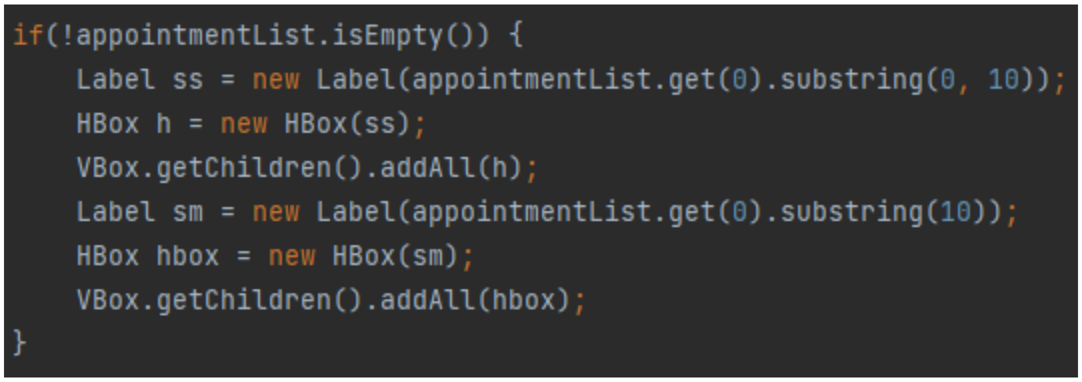
When the button is clicked, $Menu.set\_menu()$ is called so the primary window is set to Menu page.



```java
Vector<String> appointmentList = new Vector<String>();
DBconnector cnt = new DBconnector();
Connection db = cnt.getDatabselink();
try {
    Statement stat = db.createStatement();
    ResultSet res = stat.executeQuery( sql: "SELECT * FROM patients");
    while (res.next()) {
        if (res.getString( columnLabel: "appointmentdate") != null) {
            String AppointmentText = res.getString( columnLabel: "appointmentdate") + " " +
                    res.getString( columnLabel: "appointmenttime") + "  " + res.getString( columnLabel: "name")+" " +
                    res.getString( columnLabel: "surname") + "    " + res.getString( columnLabel: "appointmentnote");
            appointmentList.add(AppointmentText);
        }
    }
} catch (Exception e) {
    System.out.println("Couldn't get data of appointments from database");
}
Collections.sort(appointmentList);
```

Figure 23: Getting appointments' data from database

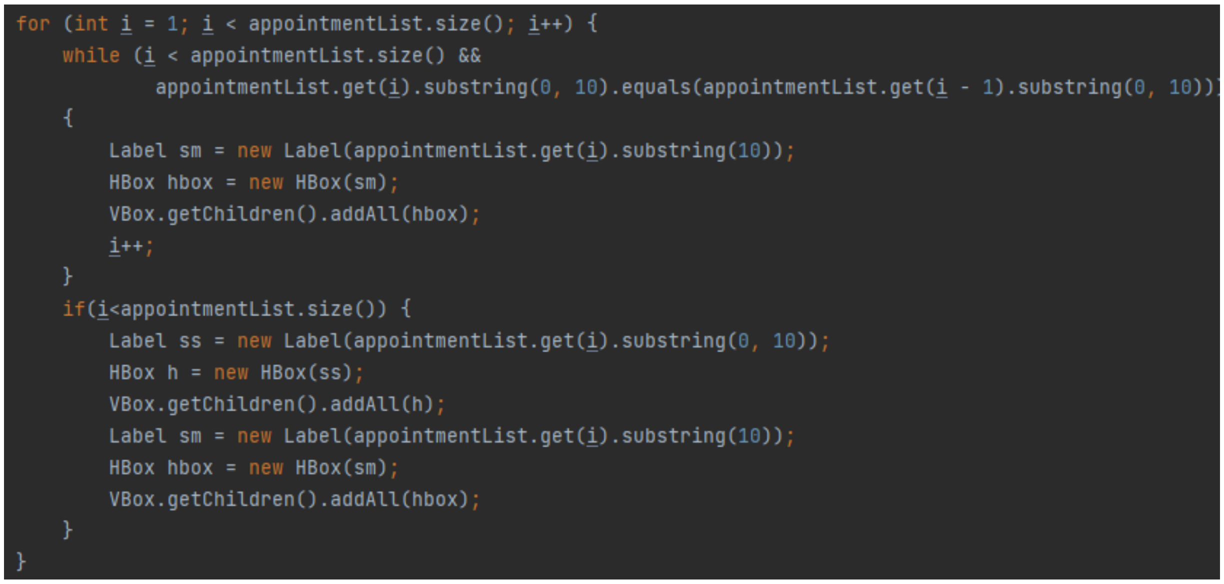
Vector *appointmentList* stores the appointments as strings. An SQL command "SELECT * FROM patients" is used to get the data from table patients. While loop is used to go through every row in table patients. If the field "appointmentdate" is not empty, then the concatenation of date, time, name, surname and note of the appointment is added to *appointmentList*. If any problem occurs an error message is printed. Then the appointments in *appointmentList* are sorted in alphabetical order. As a result, appointments will also be sorted in chronological order because the date and time are at the start of the strings.



```java
if(!appointmentList.isEmpty()) {
    Label ss = new Label(appointmentList.get(0).substring(0, 10));
    HBox h = new HBox(ss);
    VBox.getChildren().addAll(h);
    Label sm = new Label(appointmentList.get(0).substring(10));
    HBox hbox = new HBox(sm);
    VBox.getChildren().addAll(hbox);
}
```

Figure 24: *appointmentList* is not empty

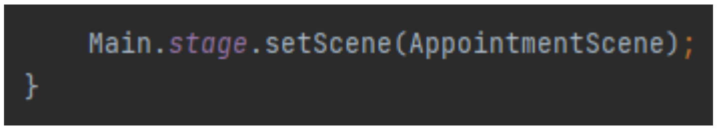
If *appointmentList* is not empty then the date (first 10 characters) of the first appointment and the appointment are displayed.



```
for (int i = 1; i < appointmentList.size(); i++) {
    while (i < appointmentList.size() &&
            appointmentList.get(i).substring(0, 10).equals(appointmentList.get(i - 1).substring(0, 10))
    {
        Label sm = new Label(appointmentList.get(i).substring(10));
        HBox hbox = new HBox(sm);
        VBox.getChildren().addAll(hbox);
        i++;
    }
    if(i<appointmentList.size()) {
        Label ss = new Label(appointmentList.get(i).substring(0, 10));
        HBox h = new HBox(ss);
        VBox.getChildren().addAll(h);
        Label sm = new Label(appointmentList.get(i).substring(10));
        HBox hbox = new HBox(sm);
        VBox.getChildren().addAll(hbox);
    }
}
```

Figure 25: Displaying appointments

While loop is used to display the appointments without date as long as their date equals the date of the previous one. Then the date of the i-th appointment and the i-th appointment are displayed. For loop is used to repeat these 2 processes for all appointments.

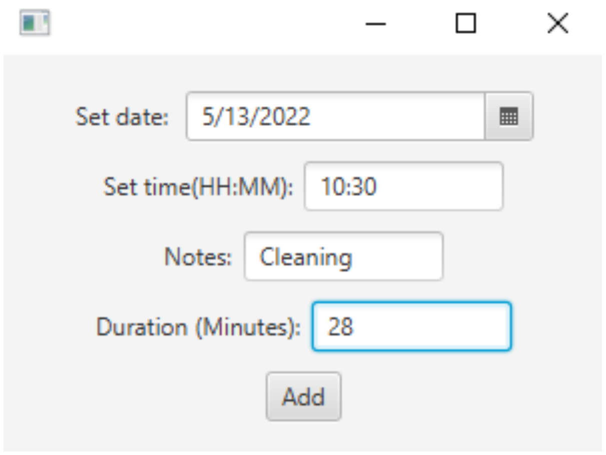


```
    Main.stage.setScene(AppointmentScene);
}
```

Figure 26: The primary window is set to Appointments page

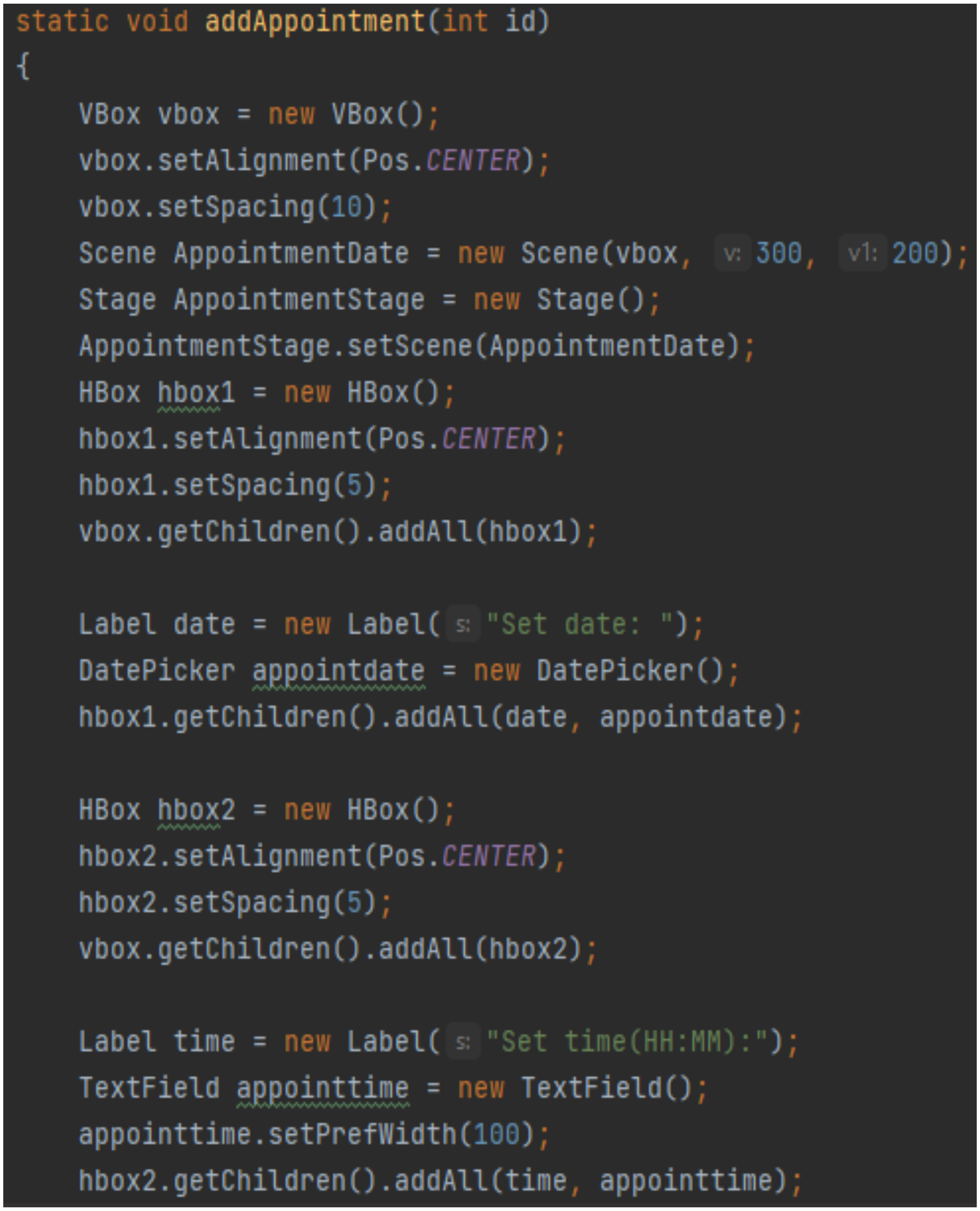
Appointments.addAppointment(intid):

id is the id of the patient of the appointment.



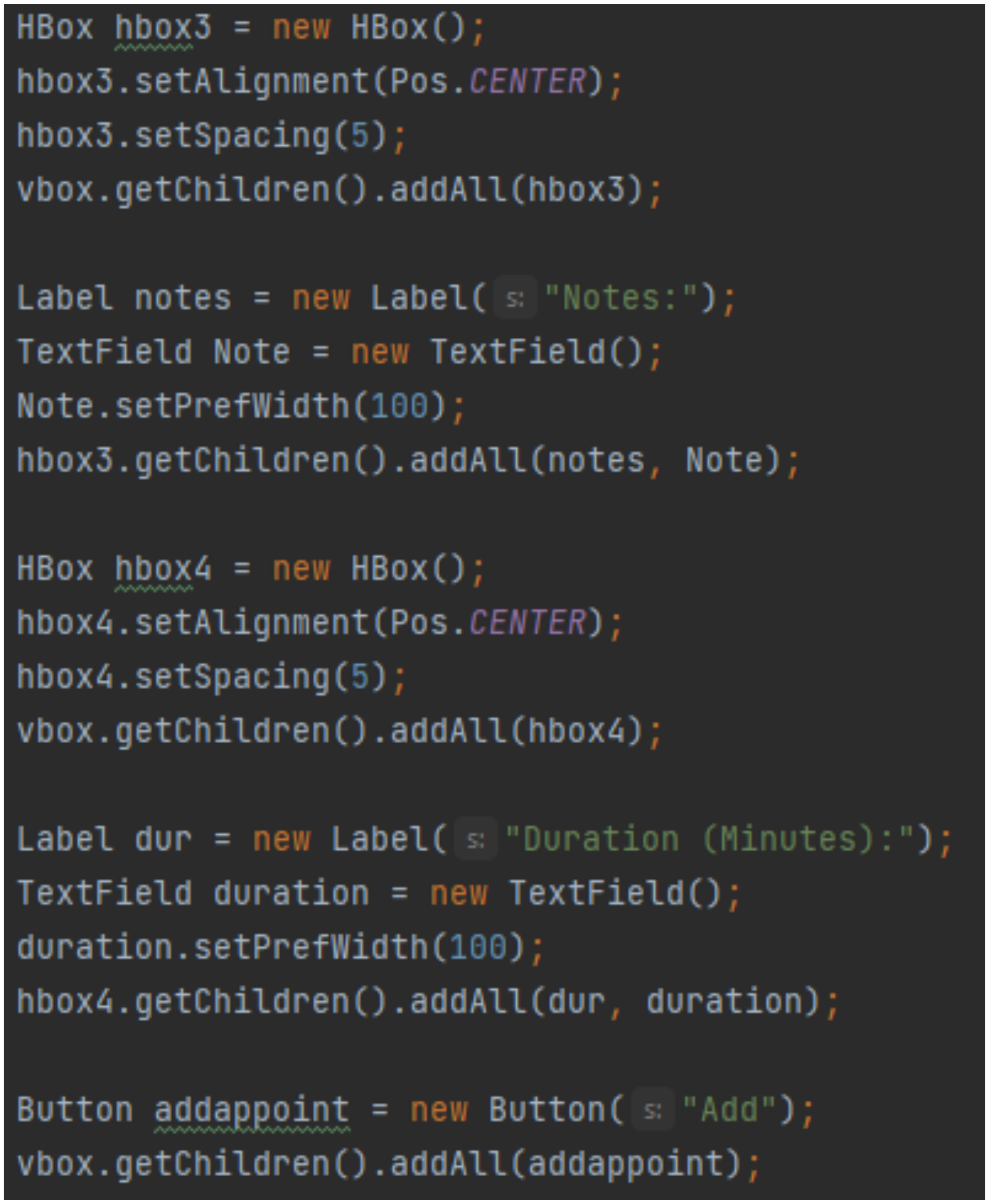
Set date:	5/13/2022
Set time(HH:MM):	10:30
Notes:	Cleaning
Duration (Minutes):	28

Figure 27: A new window is opened to schedule an appointment



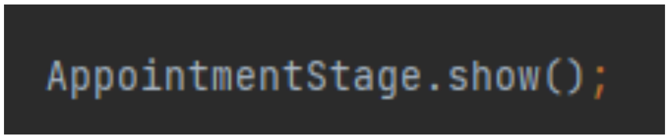
```java
static void addAppointment(int id)
{
    VBox vbox = new VBox();
    vbox.setAlignment(Pos.CENTER);
    vbox.setSpacing(10);
    Scene AppointmentDate = new Scene(vbox,  v: 300,  v1: 200);
    Stage AppointmentStage = new Stage();
    AppointmentStage.setScene(AppointmentDate);
    HBox hbox1 = new HBox();
    hbox1.setAlignment(Pos.CENTER);
    hbox1.setSpacing(5);
    vbox.getChildren().addAll(hbox1);

    Label date = new Label( s: "Set date: ");
    DatePicker appointdate = new DatePicker();
    hbox1.getChildren().addAll(date, appointdate);

    HBox hbox2 = new HBox();
    hbox2.setAlignment(Pos.CENTER);
    hbox2.setSpacing(5);
    vbox.getChildren().addAll(hbox2);

    Label time = new Label( s: "Set time(HH:MM):");
    TextField appointtime = new TextField();
    appointtime.setPrefWidth(100);
    hbox2.getChildren().addAll(time, appointtime);
```

Figure 28: GUI of AddAppointment page is created



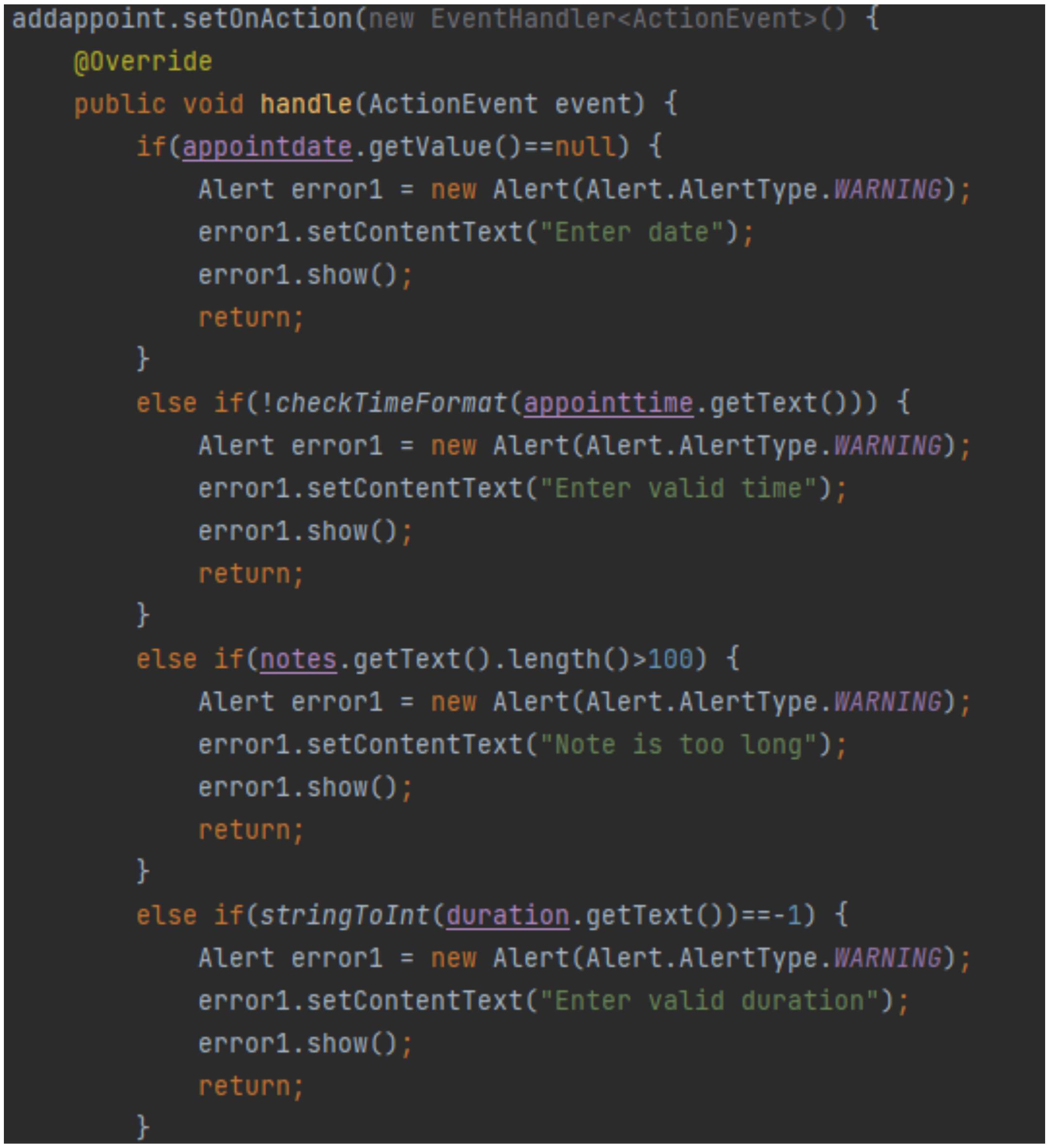
```
HBox hbox3 = new HBox();
hbox3.setAlignment(Pos.CENTER);
hbox3.setSpacing(5);
vbox.getChildren().addAll(hbox3);

Label notes = new Label( s: "Notes:");
TextField Note = new TextField();
Note.setPrefWidth(100);
hbox3.getChildren().addAll(notes, Note);

HBox hbox4 = new HBox();
hbox4.setAlignment(Pos.CENTER);
hbox4.setSpacing(5);
vbox.getChildren().addAll(hbox4);

Label dur = new Label( s: "Duration (Minutes):");
TextField duration = new TextField();
duration.setPrefWidth(100);
hbox4.getChildren().addAll(dur, duration);

Button addappoint = new Button( s: "Add");
vbox.getChildren().addAll(addappoint);
```

Figure 29: GUI of AddAppointment page is created



```
AppointmentStage.show();
```

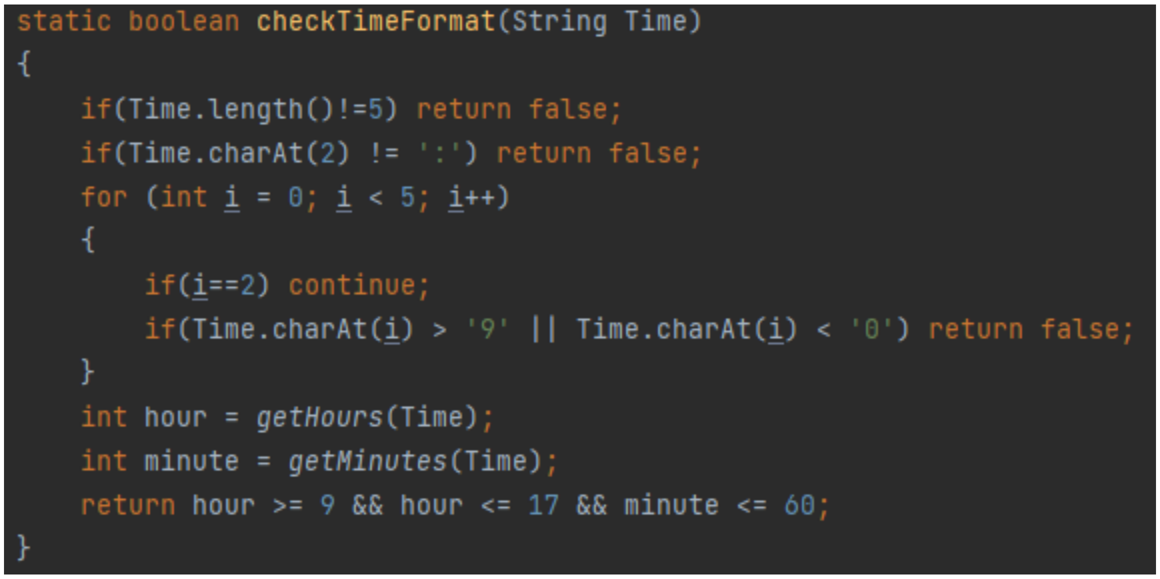
Figure 30: AddAppointment page is displayed on a new window



```java
addappoint.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        if(appointdate.getValue()==null) {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Enter date");
            error1.show();
            return;
        }
        else if(!checkTimeFormat(appointtime.getText())) {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Enter valid time");
            error1.show();
            return;
        }
        else if(notes.getText().length()>100) {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Note is too long");
            error1.show();
            return;
        }
        else if(stringToInt(duration.getText())==-1) {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Enter valid duration");
            error1.show();
            return;
        }
```

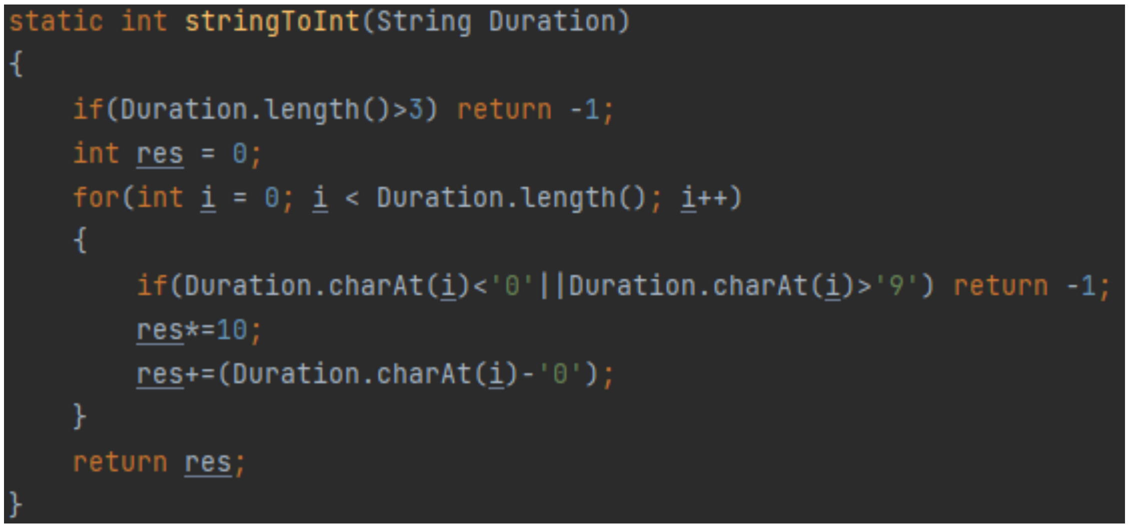
Figure 31: Button “Add” is clicked

When the button is clicked, validity of the entered information is checked and a warning message is generated accordingly.



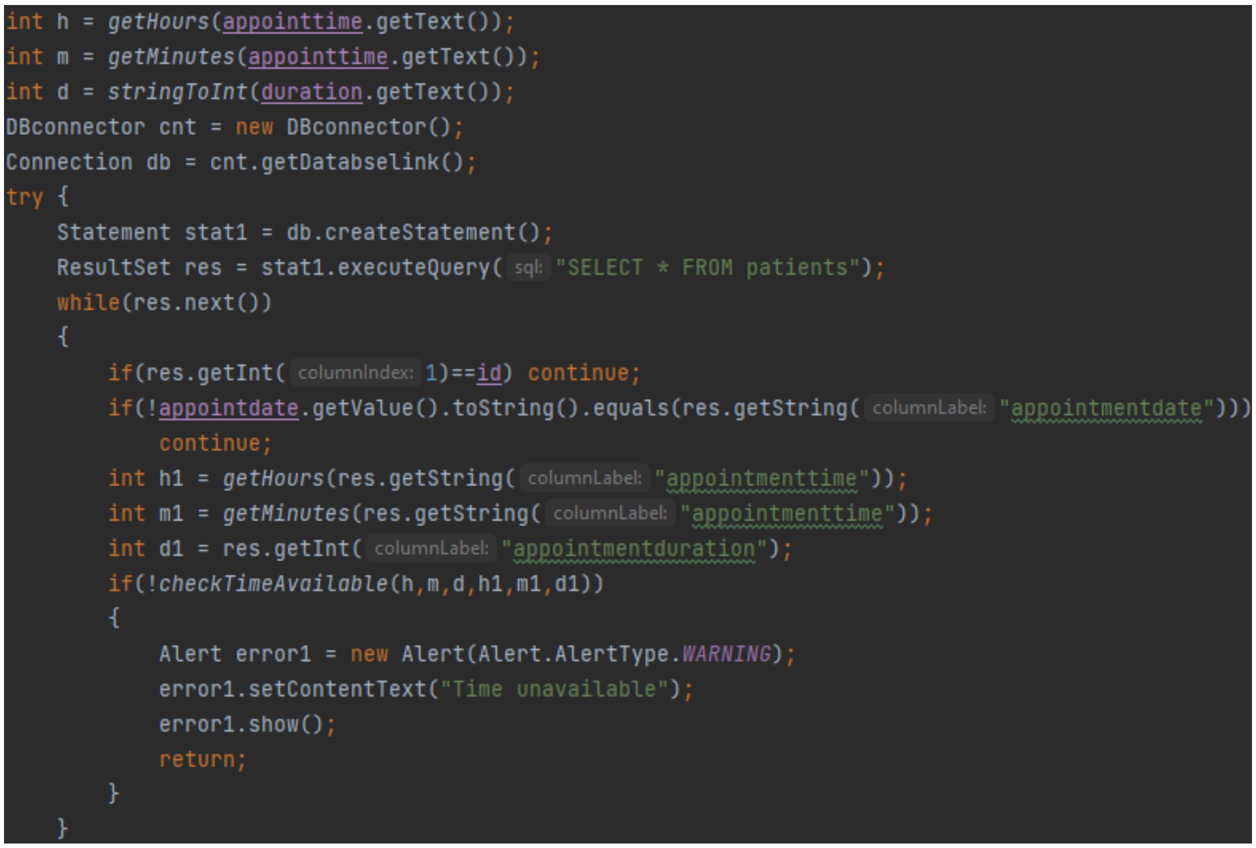
```java
static boolean checkTimeFormat(String Time)
{
    if(Time.length()!=5) return false;
    if(Time.charAt(2) != ':') return false;
    for (int i = 0; i < 5; i++)
    {
        if(i==2) continue;
        if(Time.charAt(i) > '9' || Time.charAt(i) < '0') return false;
    }
    int hour = getHours(Time);
    int minute = getMinutes(Time);
    return hour >= 9 && hour <= 17 && minute <= 60;
}
```

Figure 32: The $Appointments.checkTimeFormat(String\ Time)$ method checks if the entered time corresponds to “HH:MM” format



```java
static int stringToInt(String Duration)
{
    if(Duration.length()>3) return -1;
    int res = 0;
    for(int i = 0; i < Duration.length(); i++)
    {
        if(Duration.charAt(i)<'0'||Duration.charAt(i)>'9') return -1;
        res*=10;
        res+=(Duration.charAt(i)-'0');
    }
    return res;
}
```

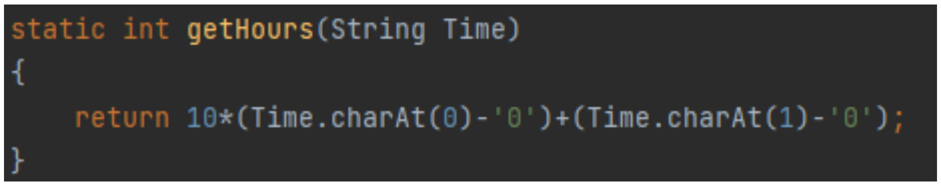
Figure 33: The $Appointments.stringToInt(String\ Duration)$ method checks id the entered duration format is valid and returns its integer value



```
int h = getHours(appointtime.getText());
int m = getMinutes(appointtime.getText());
int d = stringToInt(duration.getText());
DBconnector cnt = new DBconnector();
Connection db = cnt.getDatabselink();
try {
    Statement stat1 = db.createStatement();
    ResultSet res = stat1.executeQuery( sql: "SELECT * FROM patients");
    while(res.next())
    {
        if(res.getInt( columnIndex: 1)==id) continue;
        if(!appointdate.getValue().toString().equals(res.getString( columnLabel: "appointmentdate")))
            continue;
        int h1 = getHours(res.getString( columnLabel: "appointmenttime"));
        int m1 = getMinutes(res.getString( columnLabel: "appointmenttime"));
        int d1 = res.getInt( columnLabel: "appointmentduration");
        if(!checkTimeAvailable(h,m,d,h1,m1,d1))
        {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Time unavailable");
            error1.show();
            return;
        }
    }
```

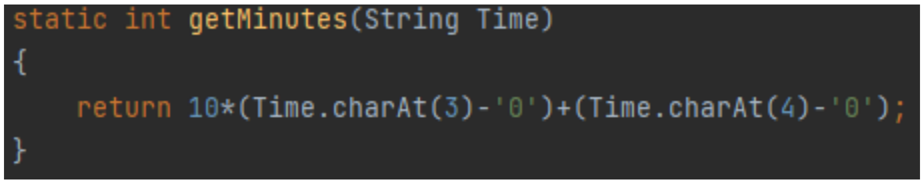
Figure 34: Time availability is checked

The program goes through every row of table patients. The program checks if there is another patient that has an appointment scheduled on the same day and their appointments overlap. If so, a warning message is generated.



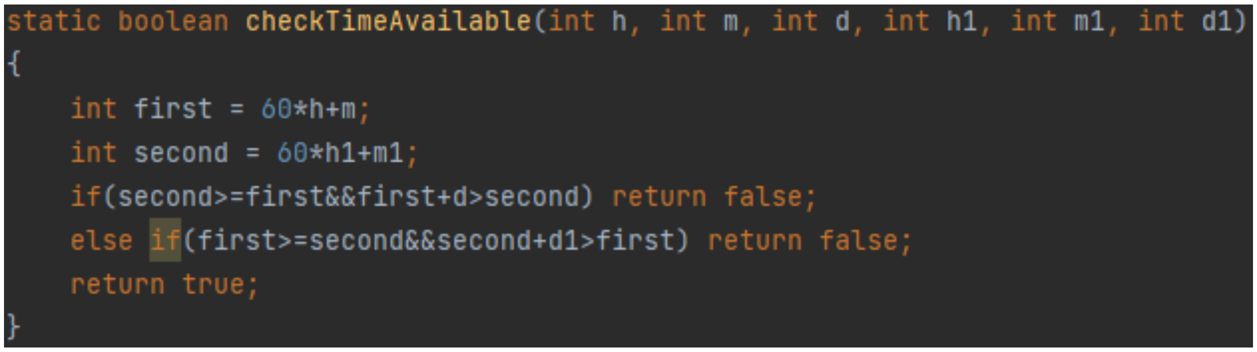
```
static int getHours(String Time)
{
    return 10*(Time.charAt(0)-'0')+(Time.charAt(1)-'0');
}
```

Figure 35: The $Appointments.getHours(String\ Time)$ method



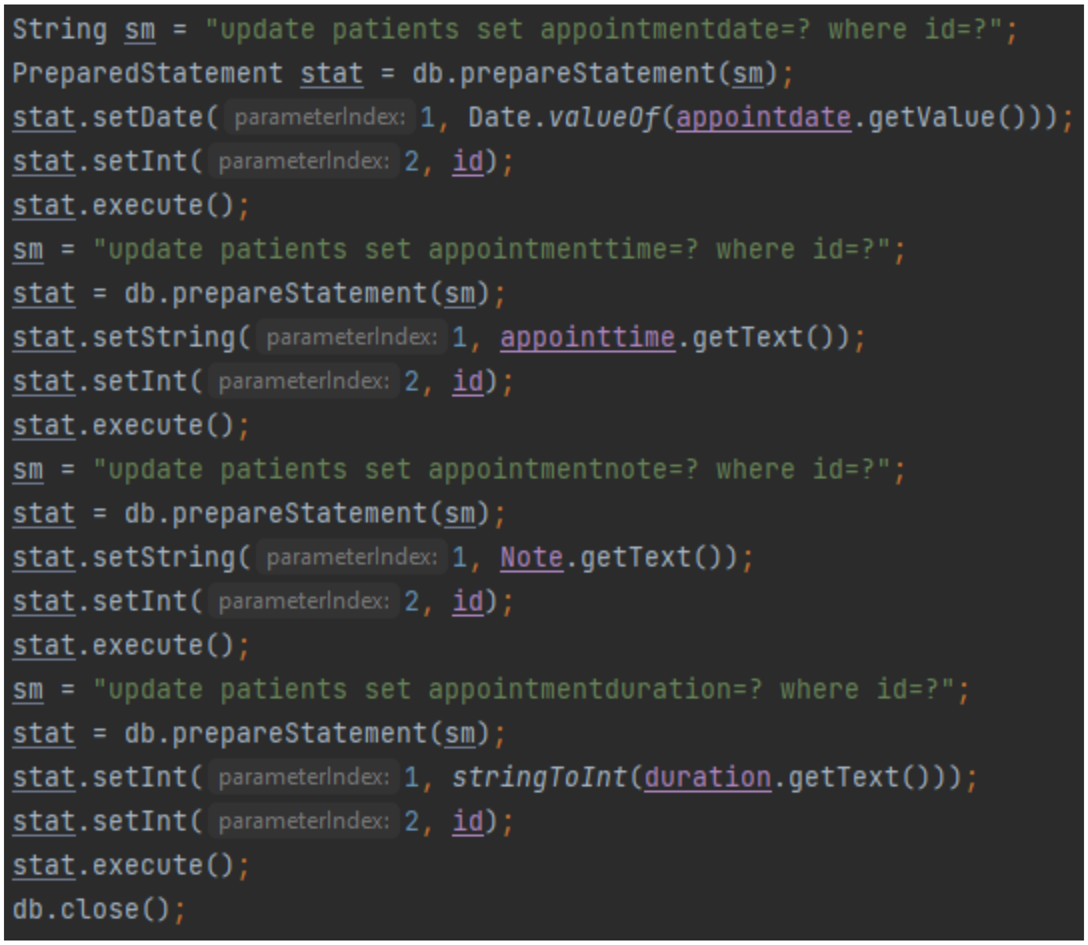
```
static int getMinutes(String Time)
{
    return 10*(Time.charAt(3)-'0')+(Time.charAt(4)-'0');
}
```

Figure 36: The $Appointments.getMinutes(String\ Time)$ method



```java
static boolean checkTimeAvailable(int h, int m, int d, int h1, int m1, int d1)
{
    int first = 60*h+m;
    int second = 60*h1+m1;
    if(second>=first&&first+d>second) return false;
    else if(first>=second&&second+d1>first) return false;
    return true;
}
```

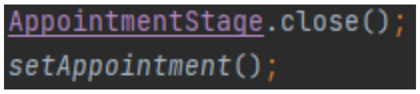
Figure 37: The $Appointments.checkTimeAvailable(int\ h, int\ m, int\ d, int\ h1, int\ m1, int\ d1)$ method checks if two given time intervals overlap or not



```java
String sm = "update patients set appointmentdate=? where id=?";
PreparedStatement stat = db.prepareStatement(sm);
stat.setDate( parameterIndex: 1, Date.valueOf(appointdate.getValue()));
stat.setInt( parameterIndex: 2, id);
stat.execute();
sm = "update patients set appointmenttime=? where id=?";
stat = db.prepareStatement(sm);
stat.setString( parameterIndex: 1, appointtime.getText());
stat.setInt( parameterIndex: 2, id);
stat.execute();
sm = "update patients set appointmentnote=? where id=?";
stat = db.prepareStatement(sm);
stat.setString( parameterIndex: 1, Note.getText());
stat.setInt( parameterIndex: 2, id);
stat.execute();
sm = "update patients set appointmentduration=? where id=?";
stat = db.prepareStatement(sm);
stat.setInt( parameterIndex: 1, stringToInt(duration.getText()));
stat.setInt( parameterIndex: 2, id);
stat.execute();
db.close();
```

Figure 38: Added appointment is saved accordingly in table patients

An SQL command “update table_name set column_name=value where id=value” is used to update the patient’s information in table patients.



```
AppointmentStage.close();
setAppointment();
```

Figure 39: AddAppointment page is closed and the primary window is set to Appointments Page

PatientList class:

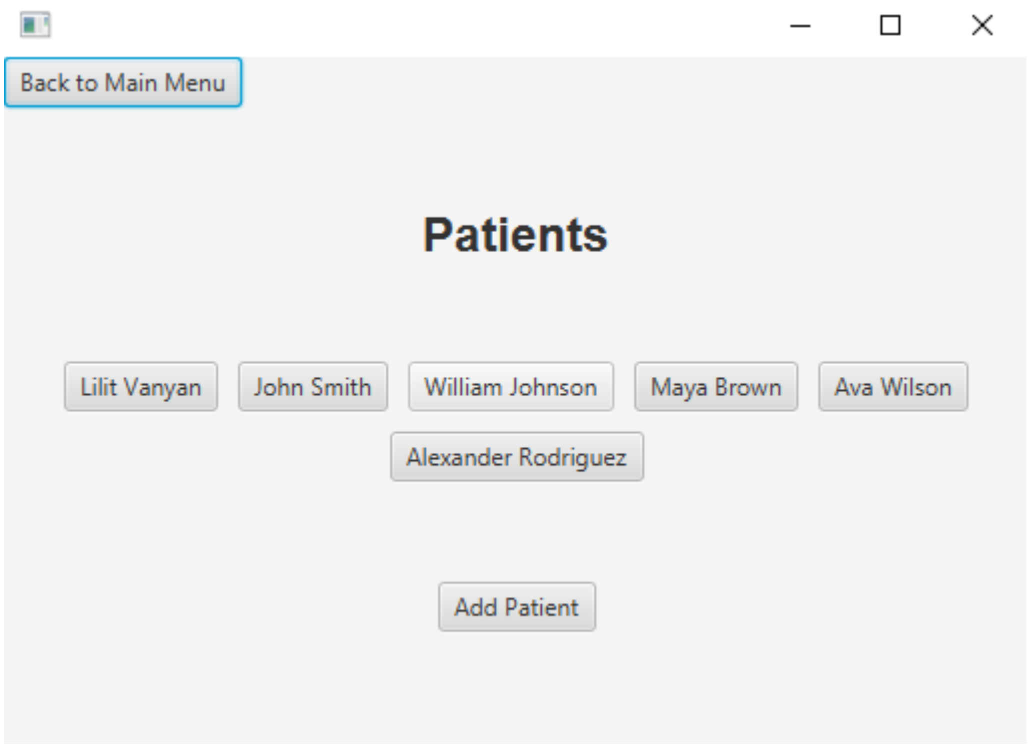
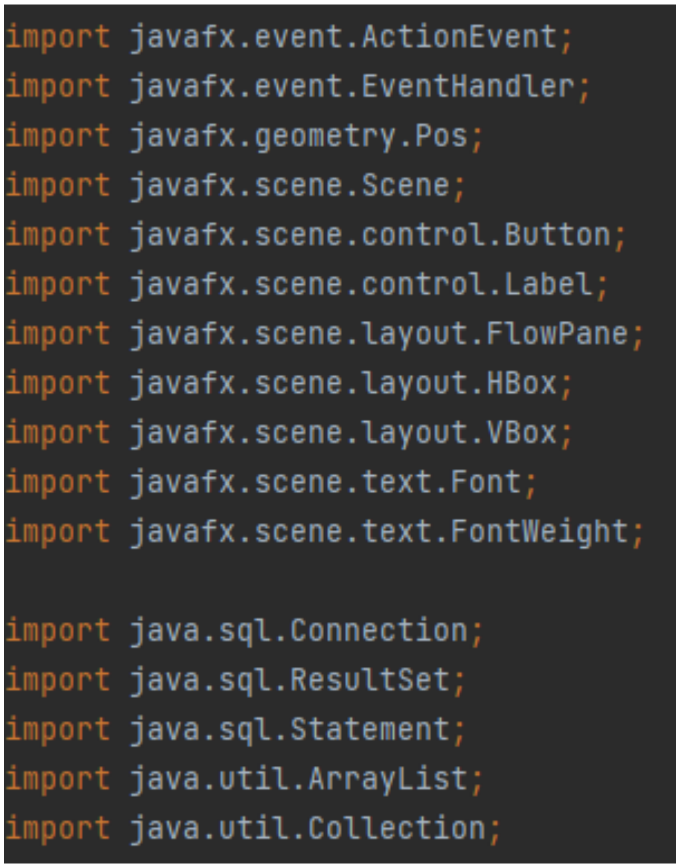


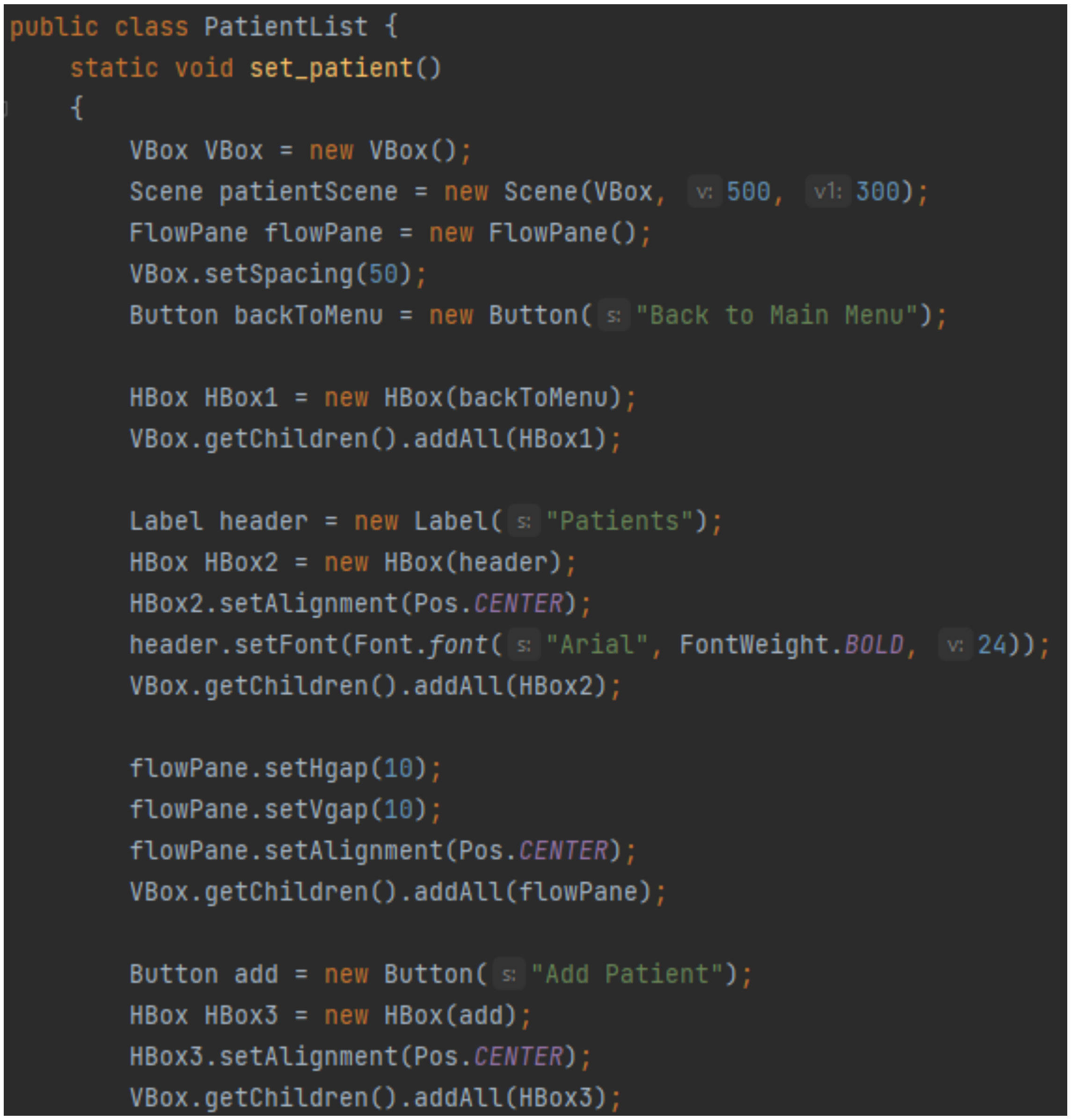
Figure 40: PaitentList page



```java
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collection;
```

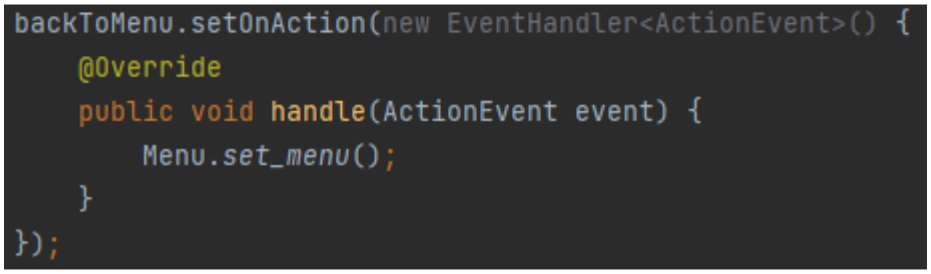
Figure 41: Libraries imported in PaitentList class

$PatientList$. set_paitent():



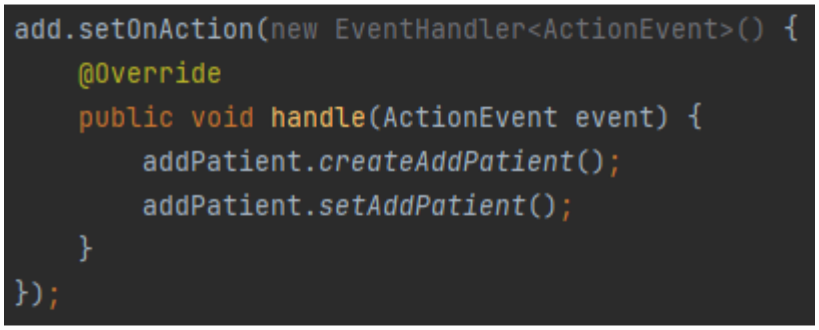
```java
public class PatientList {
    static void set_patient()
    {
        VBox VBox = new VBox();
        Scene patientScene = new Scene(VBox, v: 500, v1: 300);
        FlowPane flowPane = new FlowPane();
        VBox.setSpacing(50);
        Button backToMenu = new Button( s: "Back to Main Menu");

        HBox HBox1 = new HBox(backToMenu);
        VBox.getChildren().addAll(HBox1);

        Label header = new Label( s: "Patients");
        HBox HBox2 = new HBox(header);
        HBox2.setAlignment(Pos.CENTER);
        header.setFont(Font.font( s: "Arial", FontWeight.BOLD, v: 24));
        VBox.getChildren().addAll(HBox2);

        flowPane.setHgap(10);
        flowPane.setVgap(10);
        flowPane.setAlignment(Pos.CENTER);
        VBox.getChildren().addAll(flowPane);

        Button add = new Button( s: "Add Patient");
        HBox HBox3 = new HBox(add);
        HBox3.setAlignment(Pos.CENTER);
        VBox.getChildren().addAll(HBox3);
```

Figure 42: GUI of PaitentList page is created



```java
backToMenu.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        Menu.set_menu();
    }
});
```

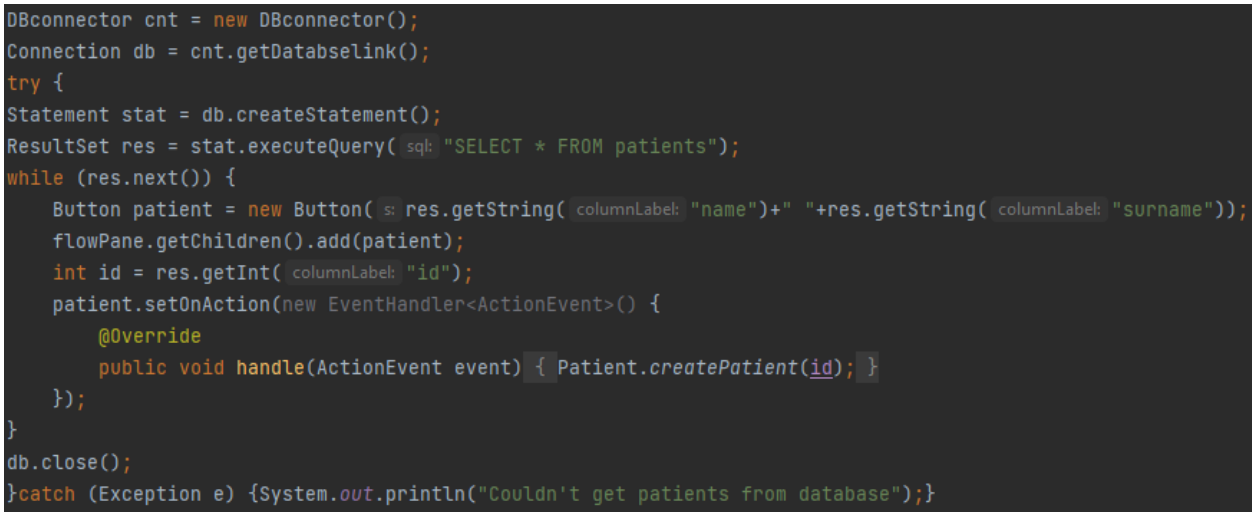
Figure 43: Button “Back to Main Menu” is clicked



```
add.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        addPatient.createAddPatient();
        addPatient.setAddPatient();
    }
});
```

Figure 44: Button "Add Patient" is clicked

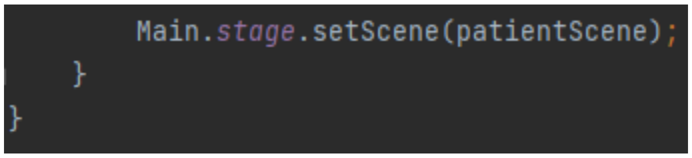
When this button is clicked the AddPatient page will open.



```
DBconnector cnt = new DBconnector();
Connection db = cnt.getDatabselink();
try {
Statement stat = db.createStatement();
ResultSet res = stat.executeQuery( sql: "SELECT * FROM patients");
while (res.next()) {
    Button patient = new Button( s: res.getString( columnLabel: "name")+" "+res.getString( columnLabel: "surname"));
    flowPane.getChildren().add(patient);
    int id = res.getInt( columnLabel: "id");
    patient.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) { Patient.createPatient(id); }
    });
}
db.close();
}catch (Exception e) {System.out.println("Couldn't get patients from database");}
```

Figure 45: Displaying the buttons

A button is created for each patient with its name on it. When a button is clicked $Patient.createPatient(id)$ method is called with corresponding id. If any problem occurs an error message will be printed.



```
        Main.stage.setScene(patientScene);
    }
}
```

Figure 46: The primary window is set to PatientList page

AddPatient class:

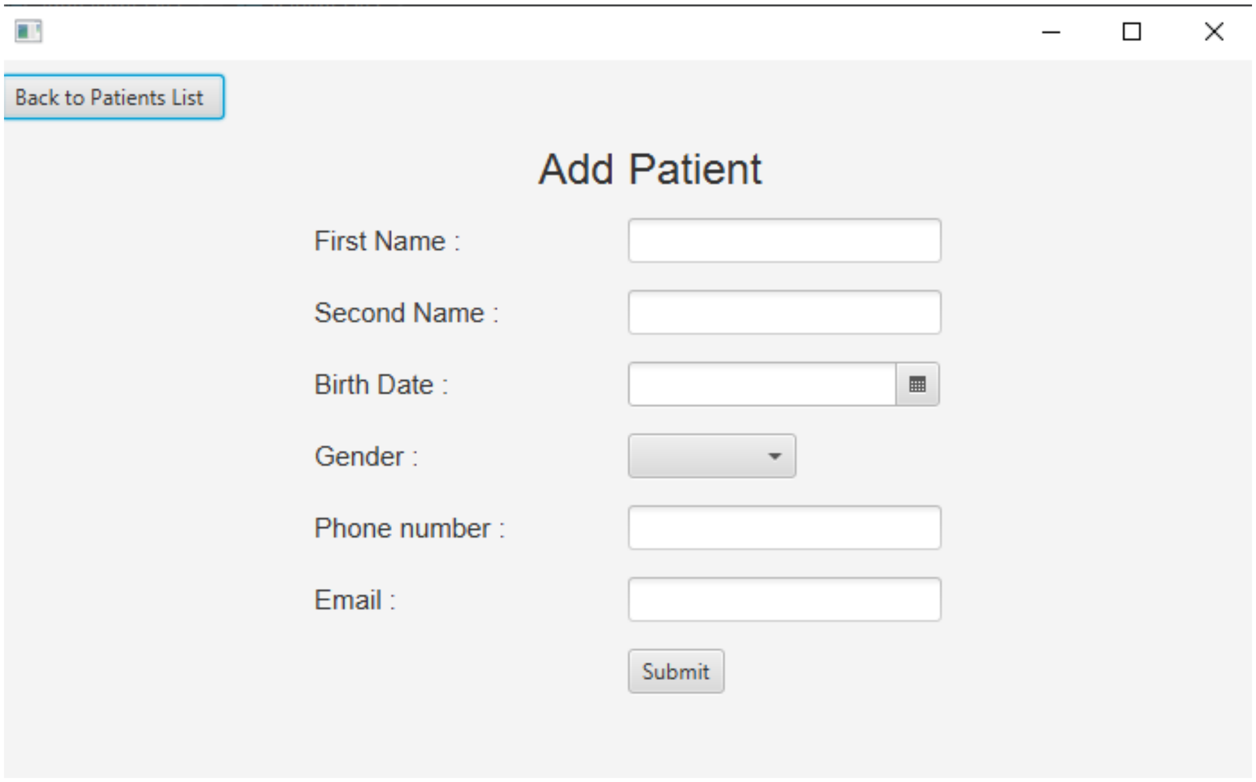
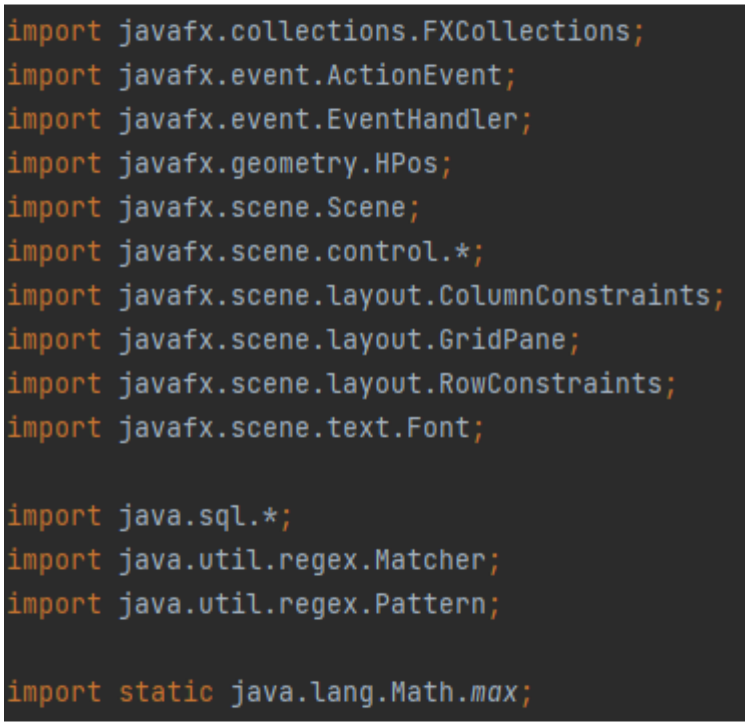


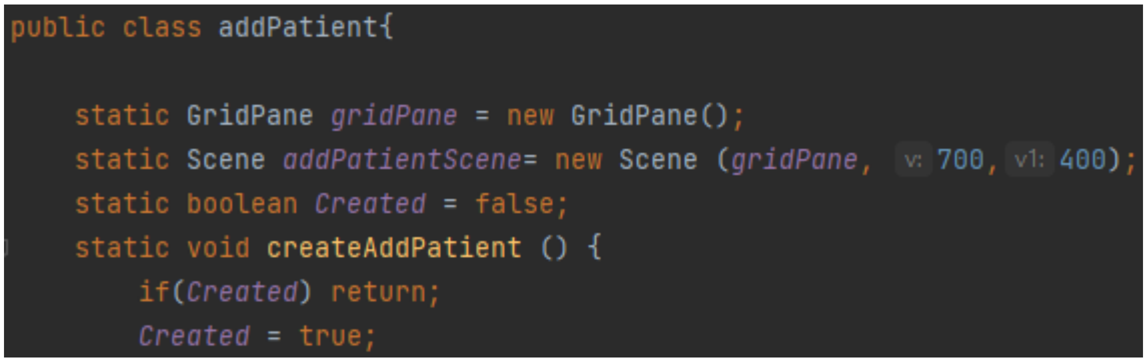
Figure 47: AddPatient page



```
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.RowConstraints;
import javafx.scene.text.Font;

import java.sql.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import static java.lang.Math.max;
```

Figure 48: Imported libraries in AddPatient class

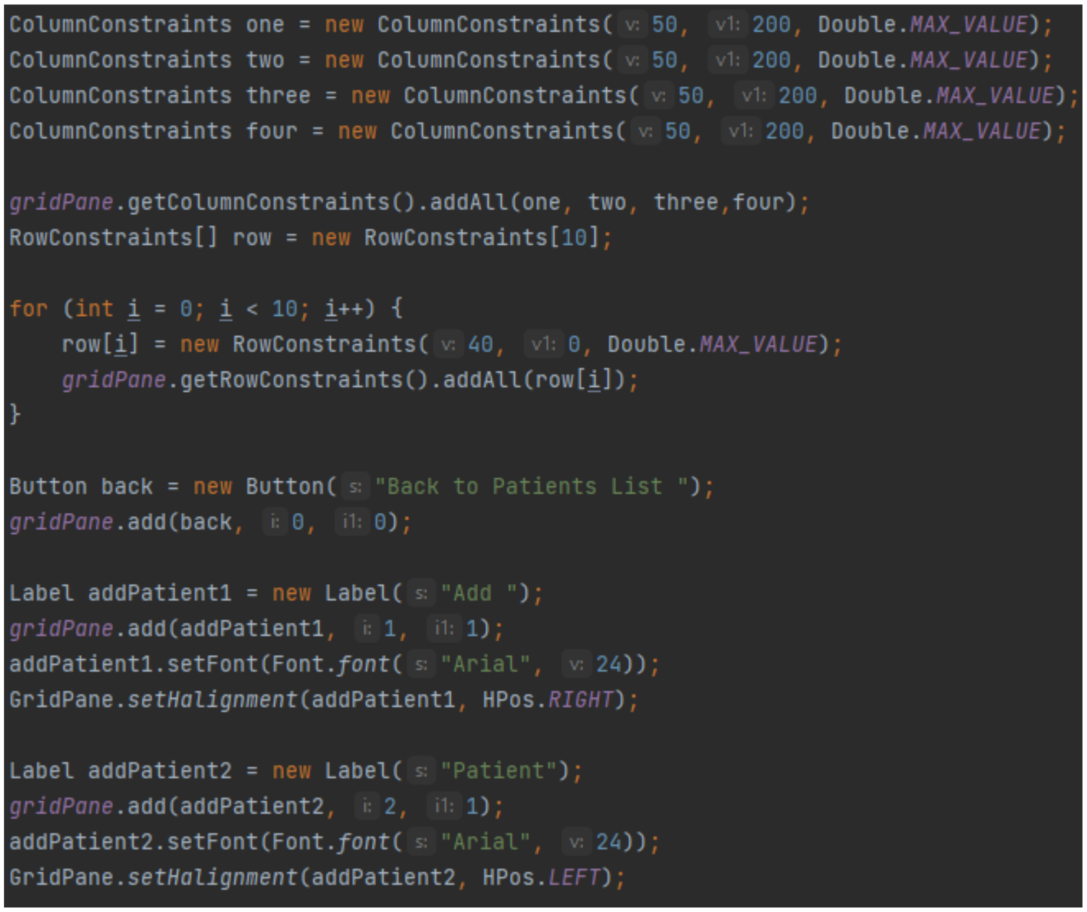
AddPatient.createAddPatient():



```java
public class addPatient{

    static GridPane gridPane = new GridPane();
    static Scene addPatientScene= new Scene (gridPane, v: 700, v1: 400);
    static boolean Created = false;
    static void createAddPatient () {
        if(Created) return;
        Created = true;
```

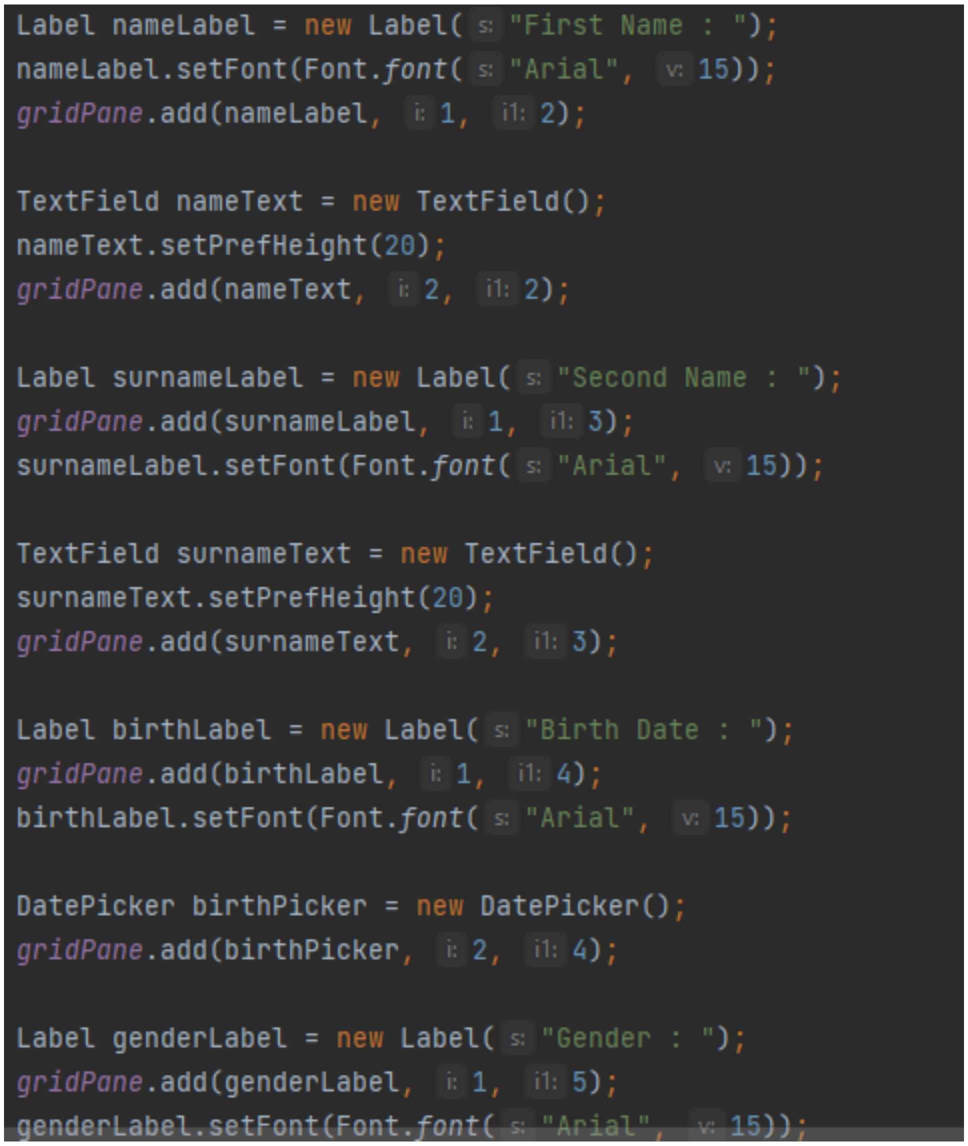
Figure 49: *AddPatient.createAddPatient*() method

AddPatient.createAddPatient() constructs the AddPatient page. Graphical components are created and stored in *addPatientScene* scene. *Created* is used similarly as in Notes class, i.e., to skip when *createAddPatient*() is called multiple times.



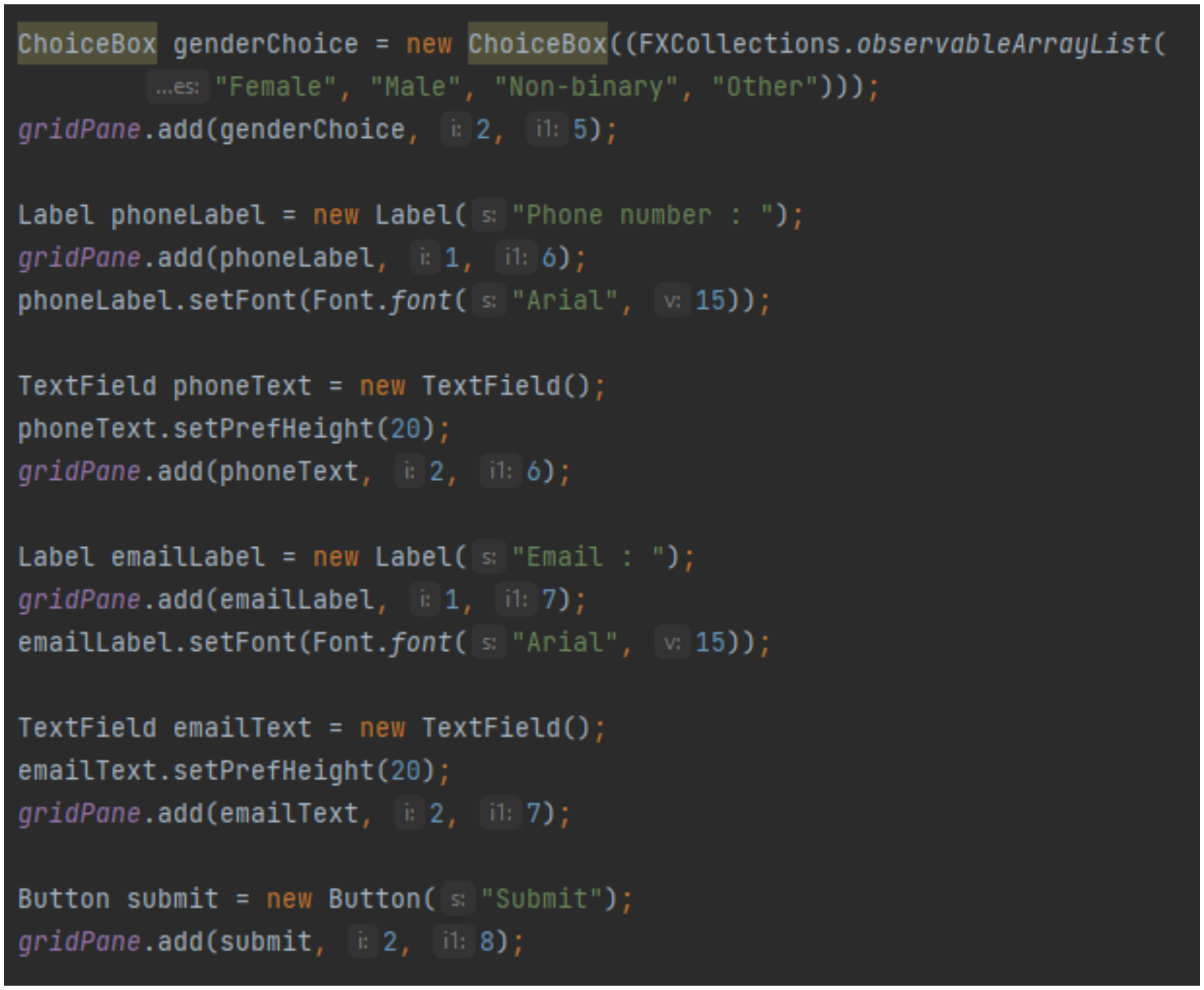
```java
ColumnConstraints one = new ColumnConstraints( v: 50,  v1: 200, Double.MAX_VALUE);
ColumnConstraints two = new ColumnConstraints( v: 50,  v1: 200, Double.MAX_VALUE);
ColumnConstraints three = new ColumnConstraints( v: 50,  v1: 200, Double.MAX_VALUE);
ColumnConstraints four = new ColumnConstraints( v: 50,  v1: 200, Double.MAX_VALUE);

gridPane.getColumnConstraints().addAll(one, two, three,four);
RowConstraints[] row = new RowConstraints[10];

for (int i = 0; i < 10; i++) {
    row[i] = new RowConstraints( v: 40,  v1: 0, Double.MAX_VALUE);
    gridPane.getRowConstraints().addAll(row[i]);
}

Button back = new Button( s: "Back to Patients List ");
gridPane.add(back,  i: 0,  i1: 0);

Label addPatient1 = new Label( s: "Add ");
gridPane.add(addPatient1,  i: 1,  i1: 1);
addPatient1.setFont(Font.font( s: "Arial",  v: 24));
GridPane.setHalignment(addPatient1, HPos.RIGHT);

Label addPatient2 = new Label( s: "Patient");
gridPane.add(addPatient2,  i: 2,  i1: 1);
addPatient2.setFont(Font.font( s: "Arial",  v: 24));
GridPane.setHalignment(addPatient2, HPos.LEFT);
```

Figure 50: GUI of AddPatient page is created



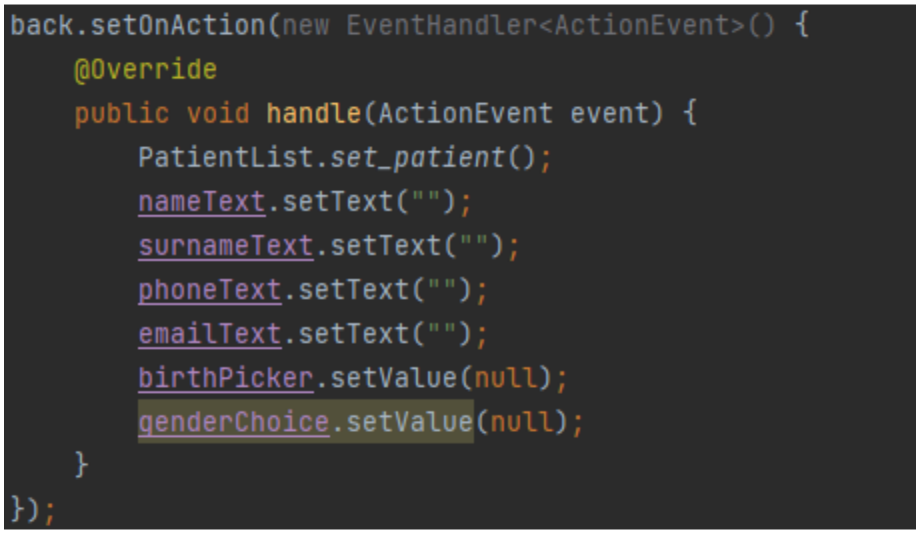
```java
Label nameLabel = new Label( s: "First Name : ");
nameLabel.setFont(Font.font( s: "Arial",  v: 15));
gridPane.add(nameLabel,  i: 1,  i1: 2);

TextField nameText = new TextField();
nameText.setPrefHeight(20);
gridPane.add(nameText,  i: 2,  i1: 2);

Label surnameLabel = new Label( s: "Second Name : ");
gridPane.add(surnameLabel,  i: 1,  i1: 3);
surnameLabel.setFont(Font.font( s: "Arial",  v: 15));

TextField surnameText = new TextField();
surnameText.setPrefHeight(20);
gridPane.add(surnameText,  i: 2,  i1: 3);

Label birthLabel = new Label( s: "Birth Date : ");
gridPane.add(birthLabel,  i: 1,  i1: 4);
birthLabel.setFont(Font.font( s: "Arial",  v: 15));

DatePicker birthPicker = new DatePicker();
gridPane.add(birthPicker,  i: 2,  i1: 4);

Label genderLabel = new Label( s: "Gender : ");
gridPane.add(genderLabel,  i: 1,  i1: 5);
genderLabel.setFont(Font.font( s: "Arial",  v: 15));
```

Figure 51: GUI of AddPatient page is created



```
ChoiceBox genderChoice = new ChoiceBox((FXCollections.observableArrayList(
        ...es: "Female", "Male", "Non-binary", "Other")));
gridPane.add(genderChoice, i: 2, i1: 5);

Label phoneLabel = new Label( s: "Phone number : ");
gridPane.add(phoneLabel, i: 1, i1: 6);
phoneLabel.setFont(Font.font( s: "Arial", v: 15));

TextField phoneText = new TextField();
phoneText.setPrefHeight(20);
gridPane.add(phoneText, i: 2, i1: 6);

Label emailLabel = new Label( s: "Email : ");
gridPane.add(emailLabel, i: 1, i1: 7);
emailLabel.setFont(Font.font( s: "Arial", v: 15));

TextField emailText = new TextField();
emailText.setPrefHeight(20);
gridPane.add(emailText, i: 2, i1: 7);

Button submit = new Button( s: "Submit");
gridPane.add(submit, i: 2, i1: 8);
```

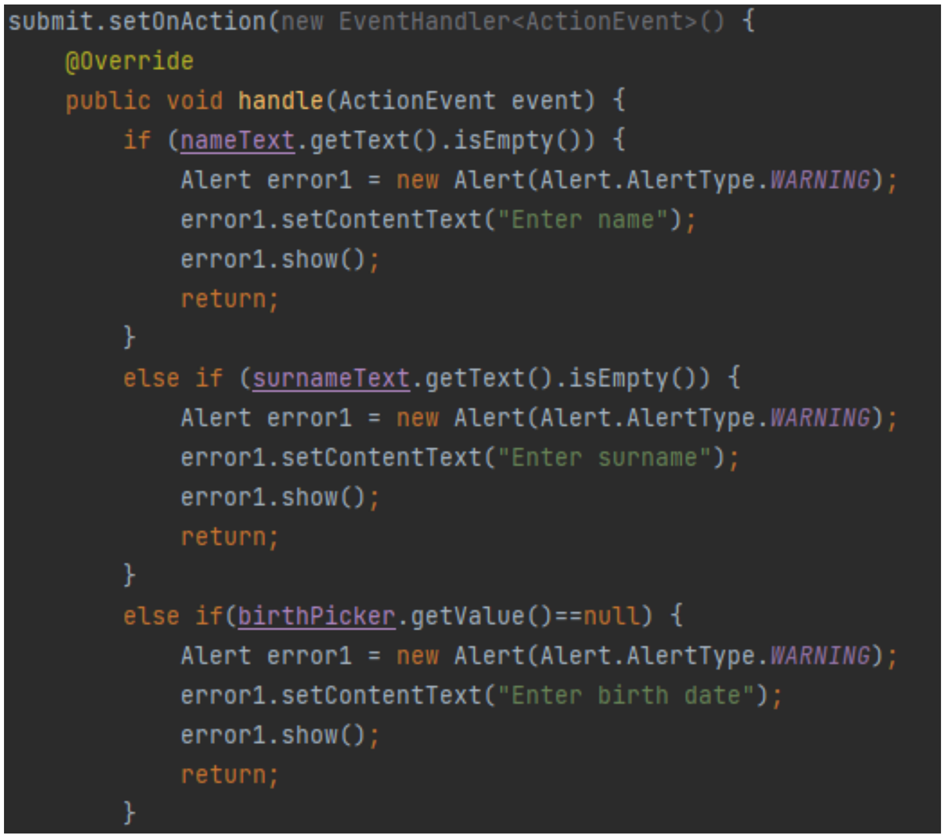
Figure 52: GUI of the AddPatient page is created



```
back.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        PatientList.set_patient();
        nameText.setText("");
        surnameText.setText("");
        phoneText.setText("");
        emailText.setText("");
        birthPicker.setValue(null);
        genderChoice.setValue(null);
    }
});
```

Figure 53: Button “Back to Patients List” clicked

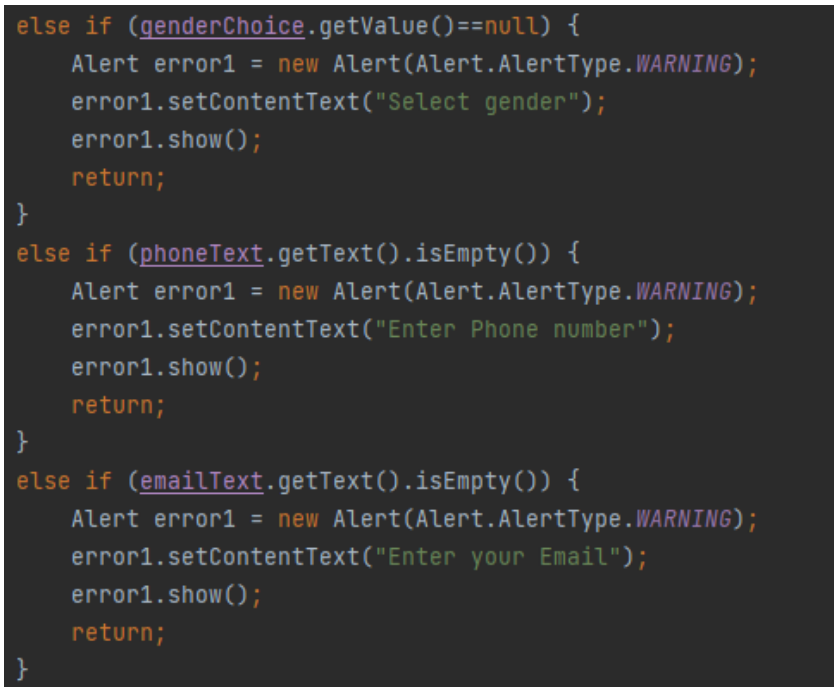
When the button is clicked all the fields are emptied and PatientList page is displayed.



```
submit.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        if (nameText.getText().isEmpty()) {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Enter name");
            error1.show();
            return;
        }
        else if (surnameText.getText().isEmpty()) {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Enter surname");
            error1.show();
            return;
        }
        else if(birthPicker.getValue()==null) {
            Alert error1 = new Alert(Alert.AlertType.WARNING);
            error1.setContentText("Enter birth date");
            error1.show();
            return;
        }
```

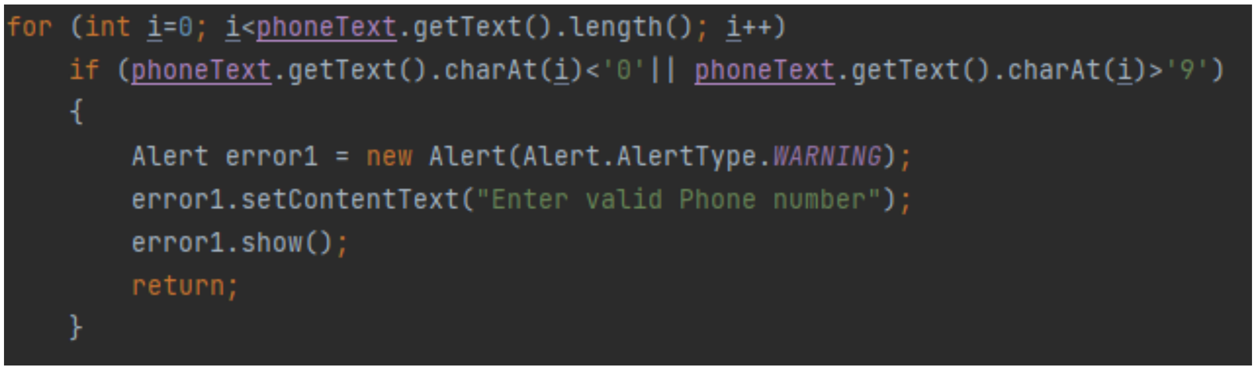
Figure 54: Button "Submit" is clicked

When the button is clicked the program starts to check if the entered information is valid. If something isn't right the program gives a corresponding warning message.



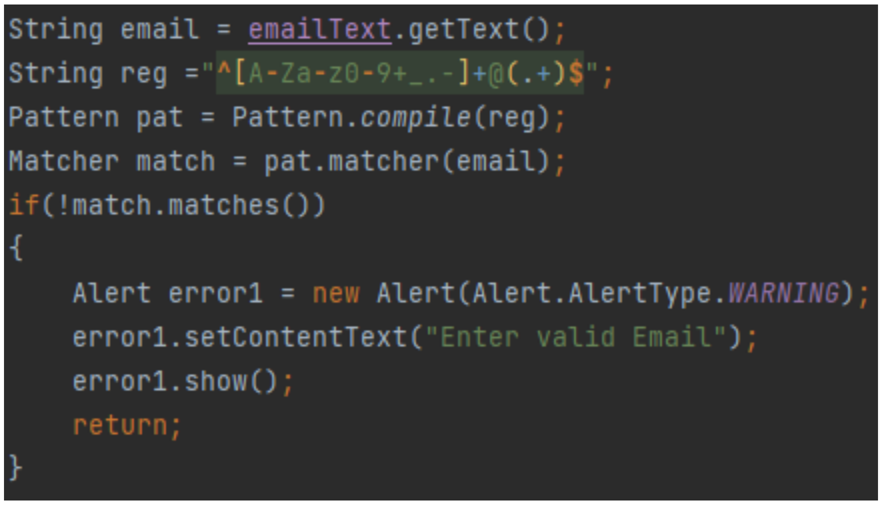
```
else if (genderChoice.getValue()==null) {
    Alert error1 = new Alert(Alert.AlertType.WARNING);
    error1.setContentText("Select gender");
    error1.show();
    return;
}
else if (phoneText.getText().isEmpty()) {
    Alert error1 = new Alert(Alert.AlertType.WARNING);
    error1.setContentText("Enter Phone number");
    error1.show();
    return;
}
else if (emailText.getText().isEmpty()) {
    Alert error1 = new Alert(Alert.AlertType.WARNING);
    error1.setContentText("Enter your Email");
    error1.show();
    return;
}
```

Figure 55: Button "Submit" is clicked. Checking the validity of entered information



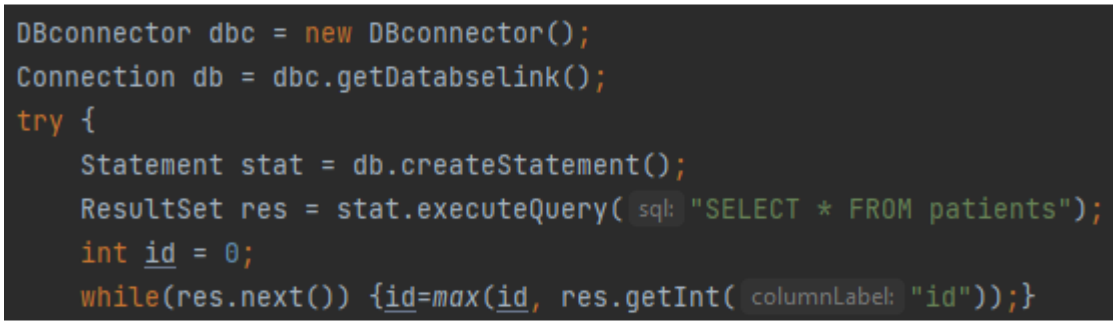
```
for (int i=0; i<phoneText.getText().length(); i++)
    if (phoneText.getText().charAt(i)<'0'|| phoneText.getText().charAt(i)>'9')
    {
        Alert error1 = new Alert(Alert.AlertType.WARNING);
        error1.setContentText("Enter valid Phone number");
        error1.show();
        return;
    }
```

Figure 56: Checking the validity of the phone number entered



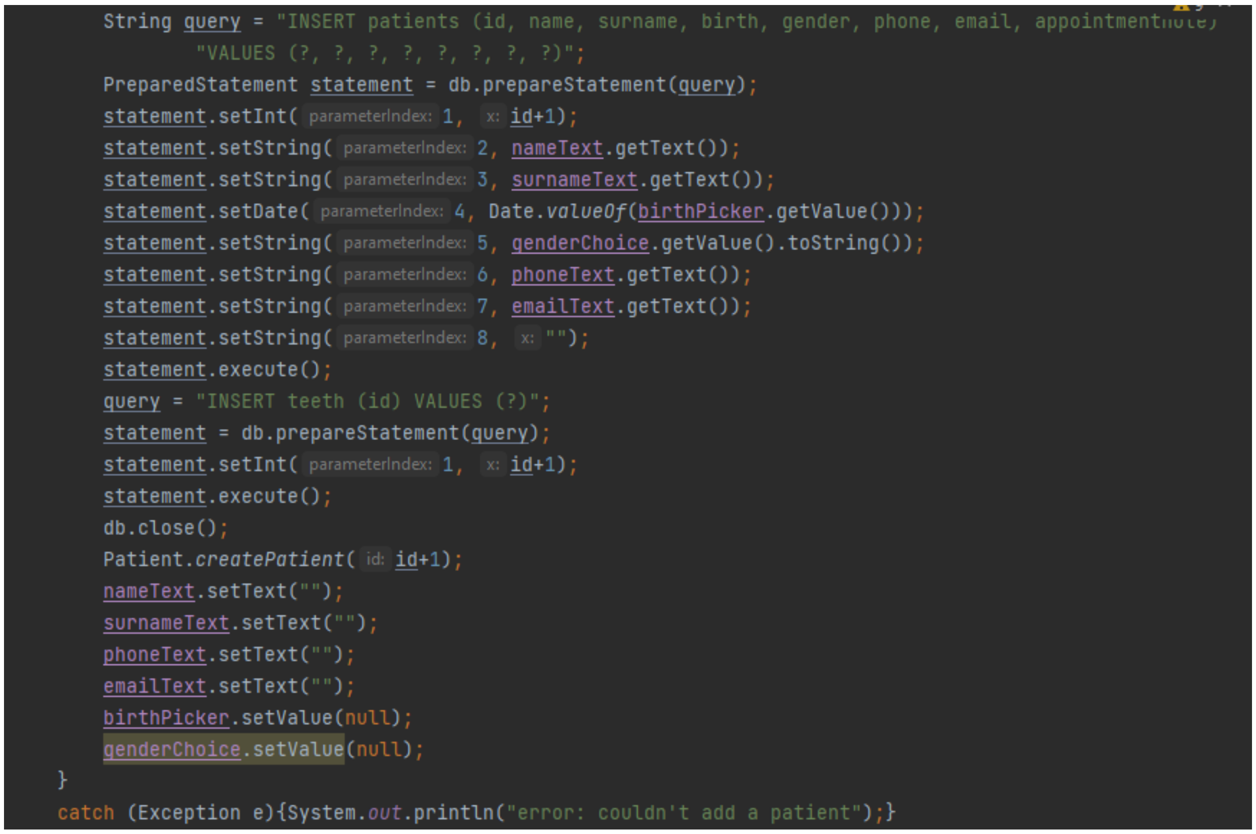
```
String email = emailText.getText();
String reg ="^[A-Za-z0-9+_.-]+@(.+)$";
Pattern pat = Pattern.compile(reg);
Matcher match = pat.matcher(email);
if(!match.matches())
{
    Alert error1 = new Alert(Alert.AlertType.WARNING);
    error1.setContentText("Enter valid Email");
    error1.show();
    return;
}
```

Figure 57: Checking if the entered email is in “text1@text2.tetx3” format as mentioned in criterion B



```
DBconnector dbc = new DBconnector();
Connection db = dbc.getDatabselink();
try {
    Statement stat = db.createStatement();
    ResultSet res = stat.executeQuery( sql: "SELECT * FROM patients");
    int id = 0;
    while(res.next()) {id=max(id, res.getInt( columnLabel: "id"));}
```

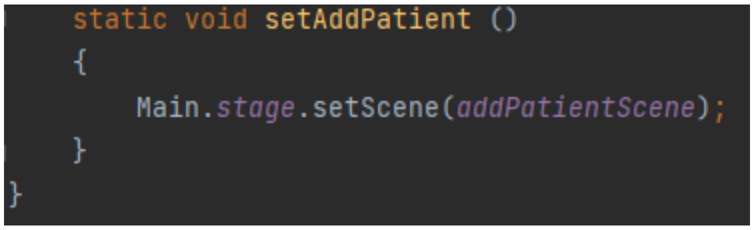
Figure 58: Generating a unique id for the new patient as mentioned in criterion B



```
        String query = "INSERT patients (id, name, surname, birth, gender, phone, email, appointmentnote)
                "VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement statement = db.prepareStatement(query);
        statement.setInt( parameterIndex: 1,  x: id+1);
        statement.setString( parameterIndex: 2, nameText.getText());
        statement.setString( parameterIndex: 3, surnameText.getText());
        statement.setDate( parameterIndex: 4, Date.valueOf(birthPicker.getValue()));
        statement.setString( parameterIndex: 5, genderChoice.getValue().toString());
        statement.setString( parameterIndex: 6, phoneText.getText());
        statement.setString( parameterIndex: 7, emailText.getText());
        statement.setString( parameterIndex: 8,  x: "");
        statement.execute();
        query = "INSERT teeth (id) VALUES (?)";
        statement = db.prepareStatement(query);
        statement.setInt( parameterIndex: 1,  x: id+1);
        statement.execute();
        db.close();
        Patient.createPatient( id: id+1);
        nameText.setText("");
        surnameText.setText("");
        phoneText.setText("");
        emailText.setText("");
        birthPicker.setValue(null);
        genderChoice.setValue(null);
    }
    catch (Exception e){System.out.println("error: couldn't add a patient");}
```

Figure 59: Saving the entered data in table patients accordingly

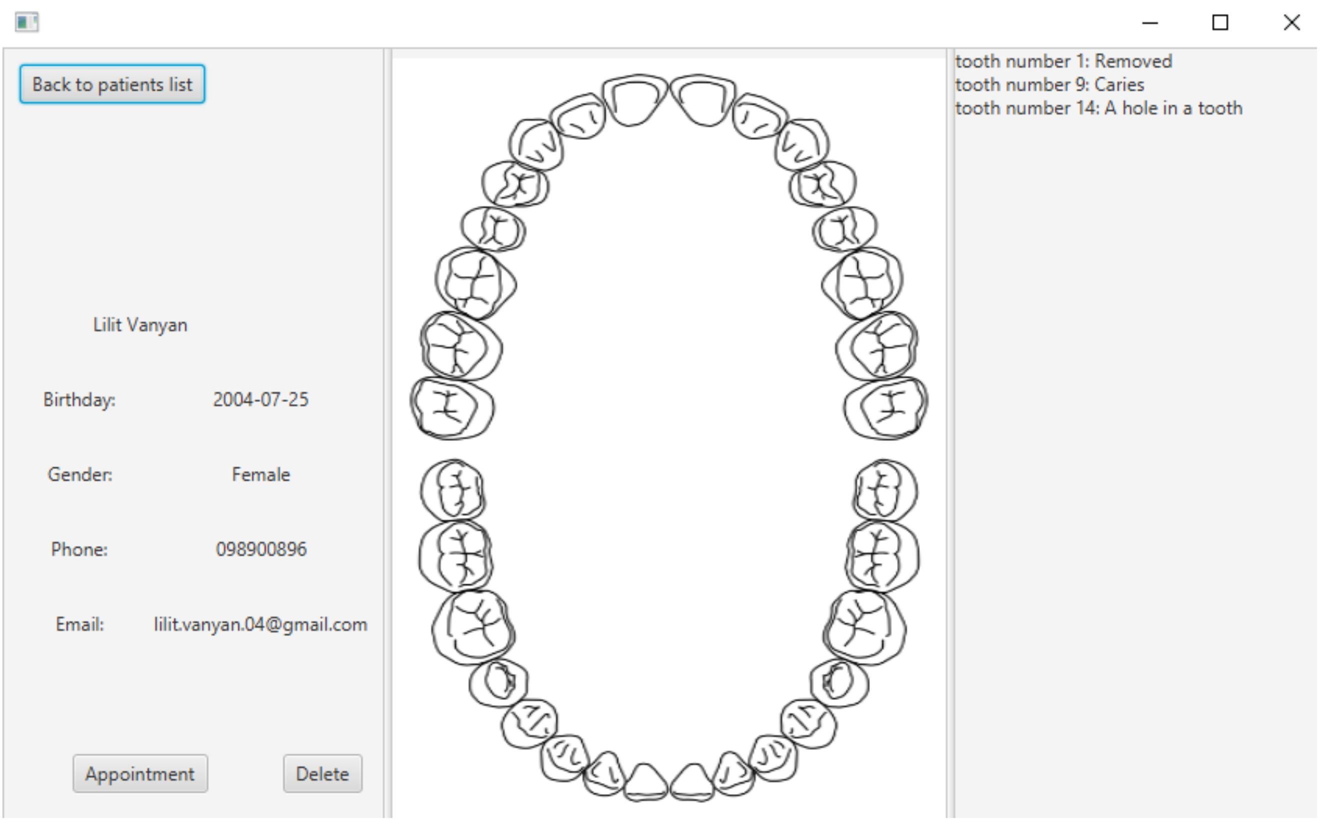
$AddPatient.setAddPatient()$:



```
    static void setAddPatient ()
    {
        Main.stage.setScene(addPatientScene);
    }
}
```

Figure 60: Setting the primary window to AddPatient page

Patient Class:



Lilit Vanyan

Birthday: 2004-07-25

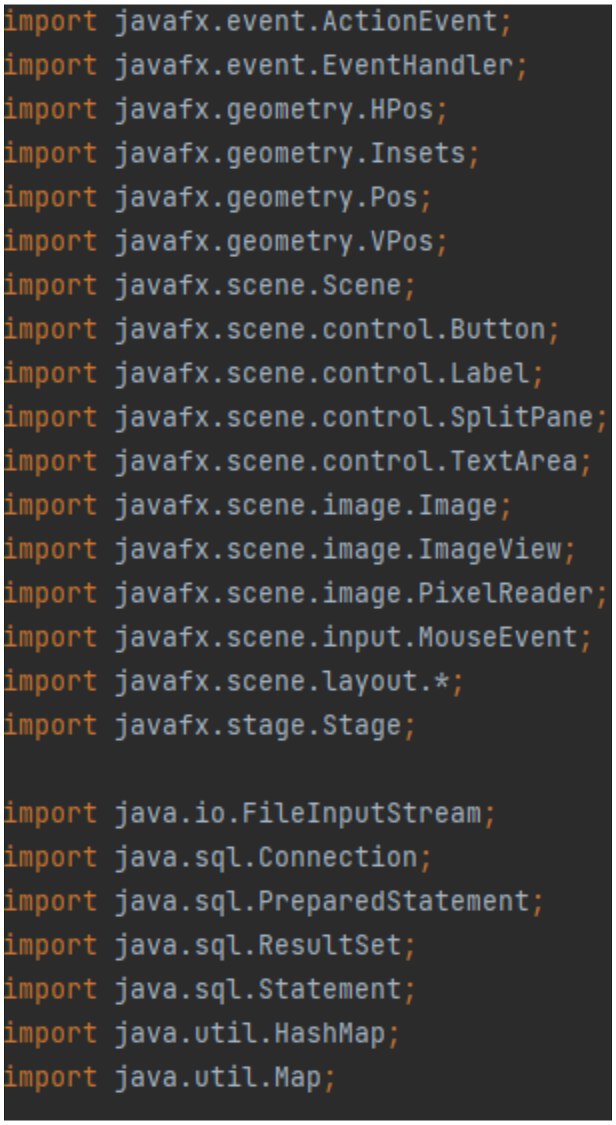
Gender: Female

Phone: 098900896

Email: lilit.vanyan.04@gmail.com

tooth number 1: Removed
tooth number 9: Caries
tooth number 14: A hole in a tooth

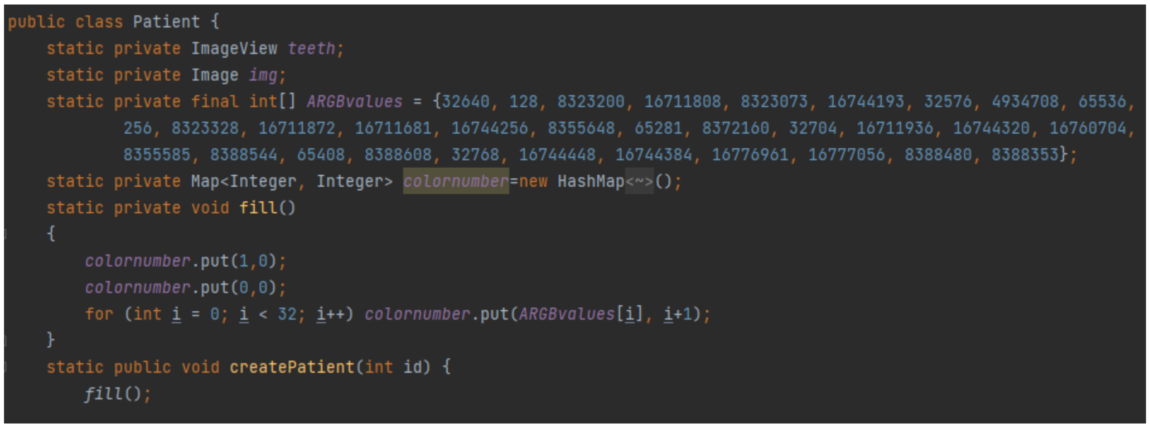
Figure 61: Patient page



```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.geometry.VPos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.SplitPane;
import javafx.scene.control.TextArea;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.image.PixelReader;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.*;
import javafx.stage.Stage;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map;
```

Figure 62: Imported libraries in Patient class

$Patient.createPatient(int\ id)$:



```java
public class Patient {
    static private ImageView teeth;
    static private Image img;
    static private final int[] ARGBvalues = {32640, 128, 8323200, 16711808, 8323073, 16744193, 32576, 4934708, 65536,
            256, 8323328, 16711872, 16711681, 16744256, 8355648, 65281, 8372160, 32704, 16711936, 16744320, 16760704,
            8355585, 8388544, 65408, 8388608, 32768, 16744448, 16744384, 16776961, 16777056, 8388480, 8388353};
    static private Map<Integer, Integer> colornumber=new HashMap<~>();
    static private void fill()
    {
        colornumber.put(1,0);
        colornumber.put(0,0);
        for (int i = 0; i < 32; i++) colornumber.put(ARGBvalues[i], i+1);
    }
    static public void createPatient(int id) {
        fill();
```

Figure 63: Patient class

As mentioned in criterion B, $ARGBvalues$ is manually created, where $ARGBvalues[i]$ represents the ARGB value of a tooth in Figure 65. $colornumber[i]$ represents the number of the tooth which ARGB value is equal to i. $fill()$ fills the values into the $colornumber$.

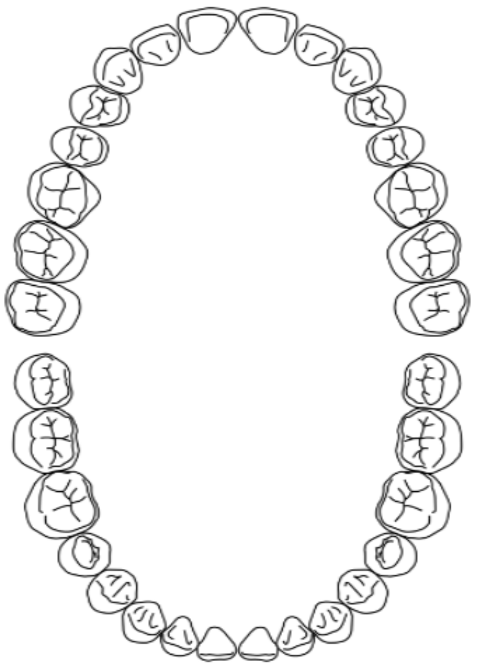


Figure 64: Tooth chart displayed on Patient page

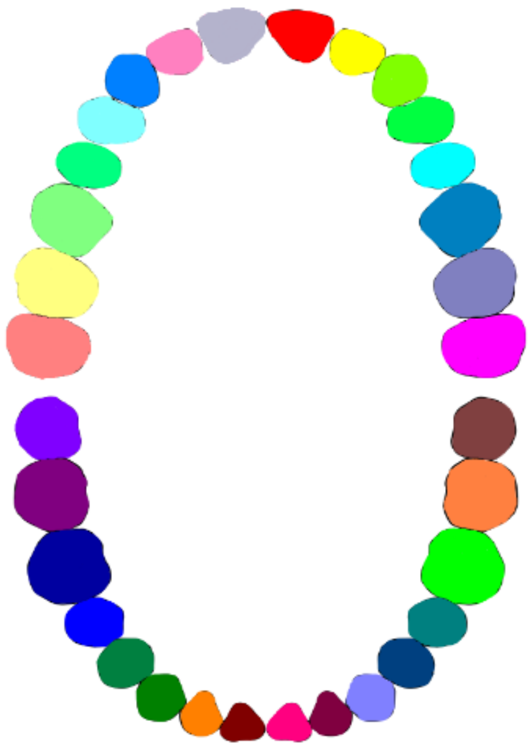
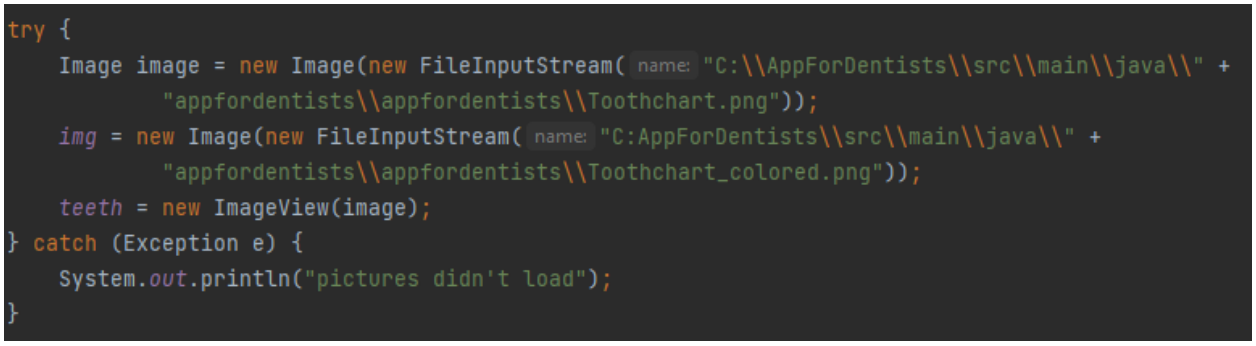
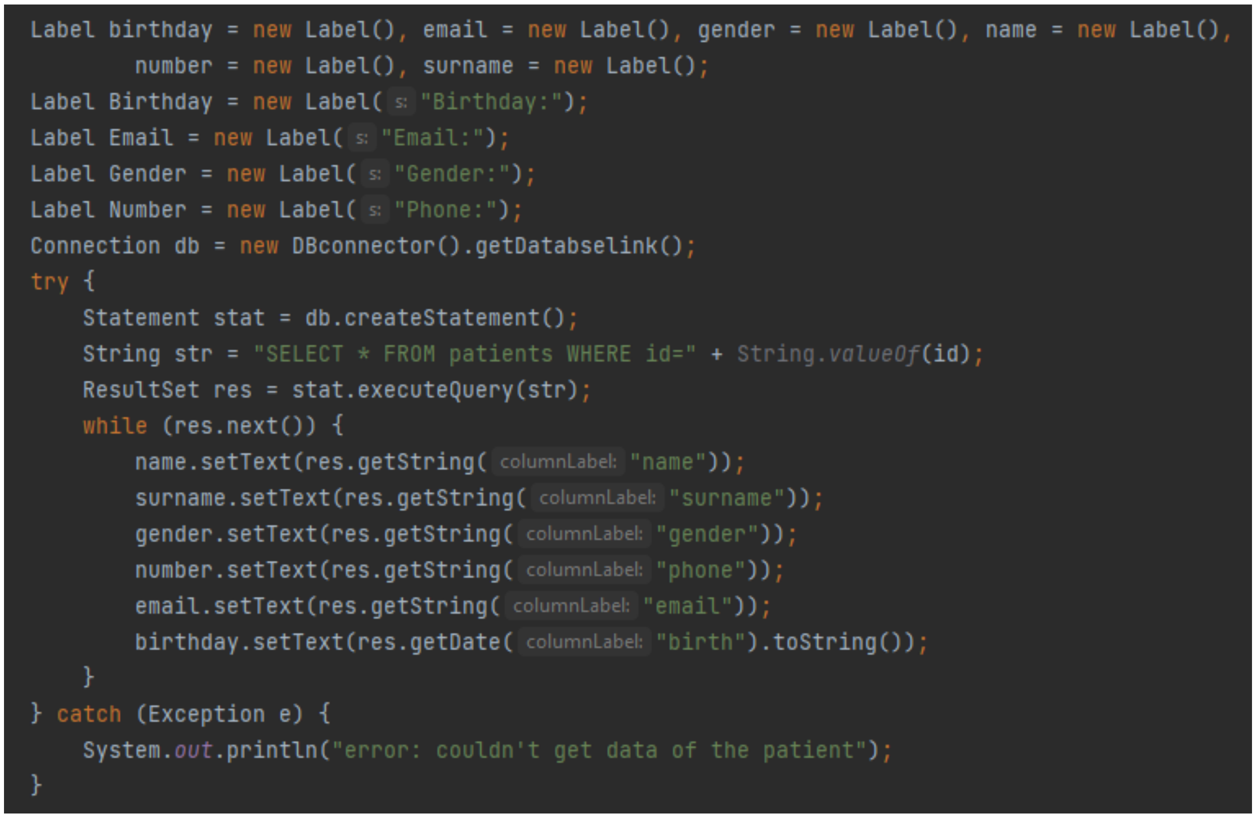


Figure 65: A manually colored tooth chart saved in project files



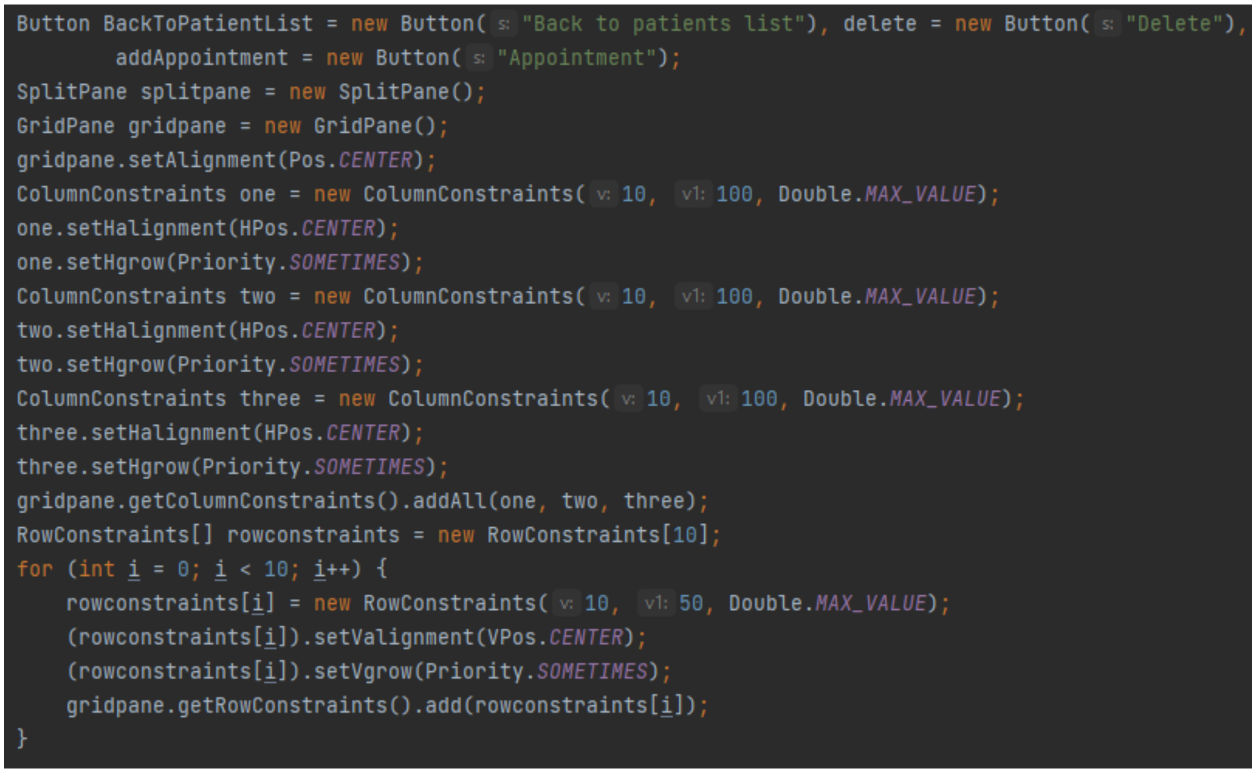
```
try {
    Image image = new Image(new FileInputStream( name: "C:\\AppForDentists\\src\\main\\java\\" +
            "appfordentists\\appfordentists\\Toothchart.png"));
    img = new Image(new FileInputStream( name: "C:AppForDentists\\src\\main\\java\\" +
            "appfordentists\\appfordentists\\Toothchart_colored.png"));
    teeth = new ImageView(image);
} catch (Exception e) {
    System.out.println("pictures didn't load");
}
```

Figure 66: Loading the tooth charts accordingly



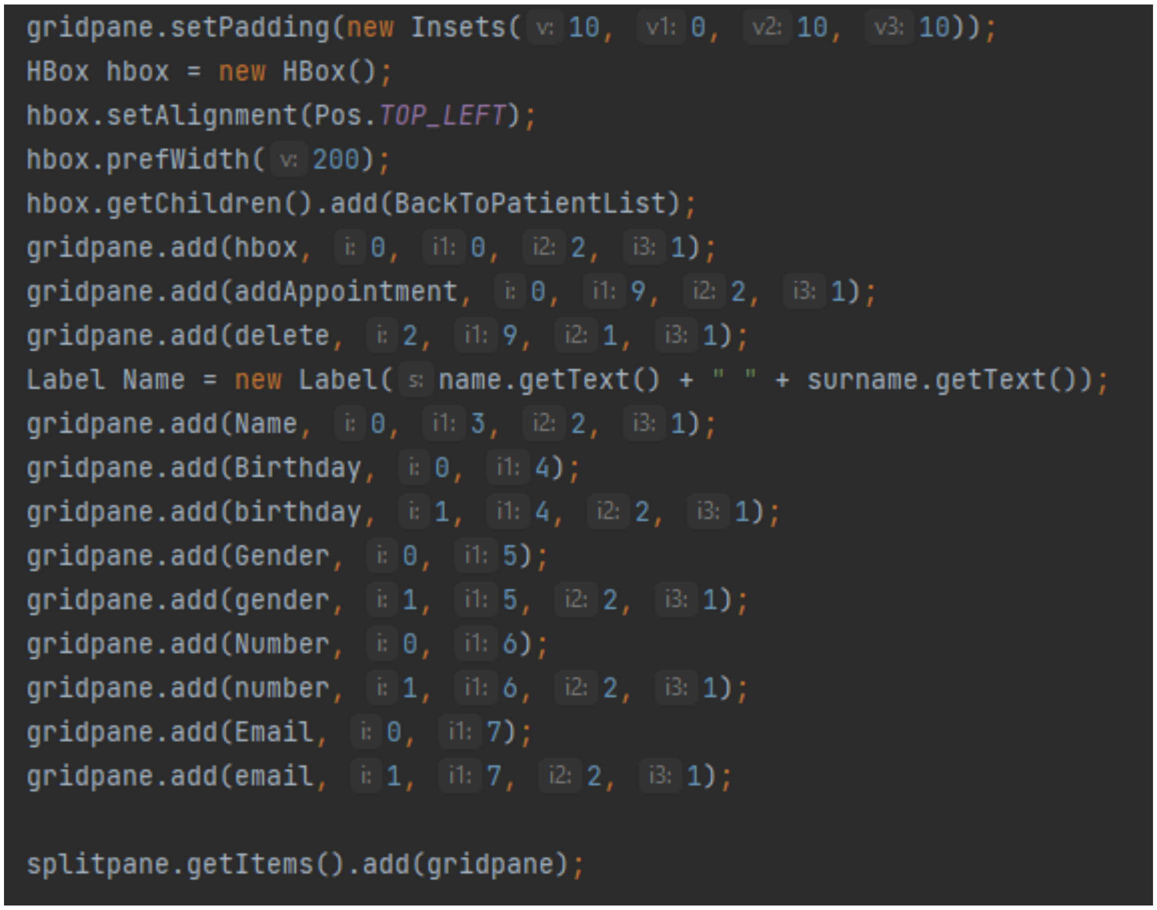
```
Label birthday = new Label(), email = new Label(), gender = new Label(), name = new Label(),
        number = new Label(), surname = new Label();
Label Birthday = new Label( s: "Birthday:");
Label Email = new Label( s: "Email:");
Label Gender = new Label( s: "Gender:");
Label Number = new Label( s: "Phone:");
Connection db = new DBconnector().getDatabselink();
try {
    Statement stat = db.createStatement();
    String str = "SELECT * FROM patients WHERE id=" + String.valueOf(id);
    ResultSet res = stat.executeQuery(str);
    while (res.next()) {
        name.setText(res.getString( columnLabel: "name"));
        surname.setText(res.getString( columnLabel: "surname"));
        gender.setText(res.getString( columnLabel: "gender"));
        number.setText(res.getString( columnLabel: "phone"));
        email.setText(res.getString( columnLabel: "email"));
        birthday.setText(res.getDate( columnLabel: "birth").toString());
    }
} catch (Exception e) {
    System.out.println("error: couldn't get data of the patient");
}
```

Figure 67: Getting the data of the patient from table patients and displaying them



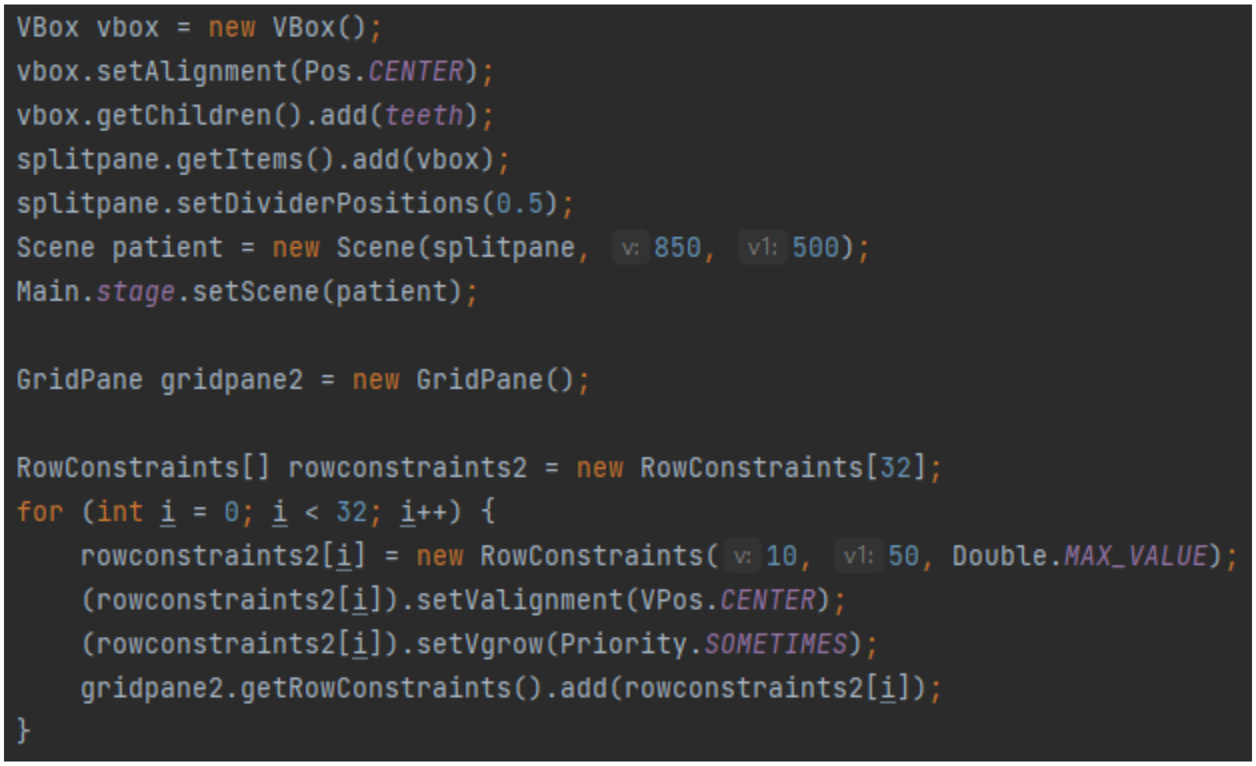
```
Button BackToPatientList = new Button( s: "Back to patients list"), delete = new Button( s: "Delete"),
        addAppointment = new Button( s: "Appointment");
SplitPane splitpane = new SplitPane();
GridPane gridpane = new GridPane();
gridpane.setAlignment(Pos.CENTER);
ColumnConstraints one = new ColumnConstraints( v: 10,  v1: 100, Double.MAX_VALUE);
one.setHalignment(HPos.CENTER);
one.setHgrow(Priority.SOMETIMES);
ColumnConstraints two = new ColumnConstraints( v: 10,  v1: 100, Double.MAX_VALUE);
two.setHalignment(HPos.CENTER);
two.setHgrow(Priority.SOMETIMES);
ColumnConstraints three = new ColumnConstraints( v: 10,  v1: 100, Double.MAX_VALUE);
three.setHalignment(HPos.CENTER);
three.setHgrow(Priority.SOMETIMES);
gridpane.getColumnConstraints().addAll(one, two, three);
RowConstraints[] rowconstraints = new RowConstraints[10];
for (int i = 0; i < 10; i++) {
    rowconstraints[i] = new RowConstraints( v: 10,  v1: 50, Double.MAX_VALUE);
    (rowconstraints[i]).setValignment(VPos.CENTER);
    (rowconstraints[i]).setVgrow(Priority.SOMETIMES);
    gridpane.getRowConstraints().add(rowconstraints[i]);
}
```

Figure 68: GUI of the Patient page is created



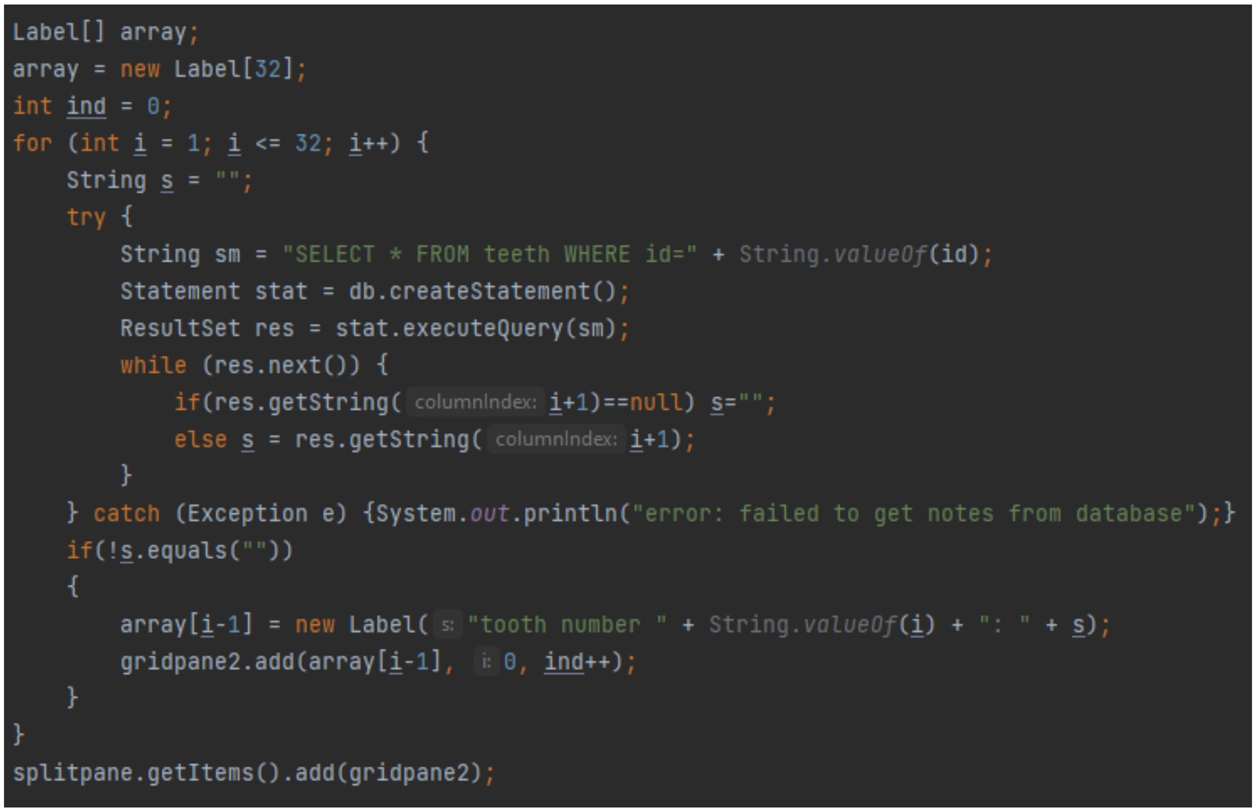
```
gridpane.setPadding(new Insets( v: 10,  v1: 0,  v2: 10,  v3: 10));
HBox hbox = new HBox();
hbox.setAlignment(Pos.TOP_LEFT);
hbox.prefWidth( v: 200);
hbox.getChildren().add(BackToPatientList);
gridpane.add(hbox,  i: 0,  i1: 0,  i2: 2,  i3: 1);
gridpane.add(addAppointment,  i: 0,  i1: 9,  i2: 2,  i3: 1);
gridpane.add(delete,  i: 2,  i1: 9,  i2: 1,  i3: 1);
Label Name = new Label( s: name.getText() + " " + surname.getText());
gridpane.add(Name,  i: 0,  i1: 3,  i2: 2,  i3: 1);
gridpane.add(Birthday,  i: 0,  i1: 4);
gridpane.add(birthday,  i: 1,  i1: 4,  i2: 2,  i3: 1);
gridpane.add(Gender,  i: 0,  i1: 5);
gridpane.add(gender,  i: 1,  i1: 5,  i2: 2,  i3: 1);
gridpane.add(Number,  i: 0,  i1: 6);
gridpane.add(number,  i: 1,  i1: 6,  i2: 2,  i3: 1);
gridpane.add(Email,  i: 0,  i1: 7);
gridpane.add(email,  i: 1,  i1: 7,  i2: 2,  i3: 1);


splitpane.getItems().add(gridpane);
```

Figure 69: GUI of the Patient page is created



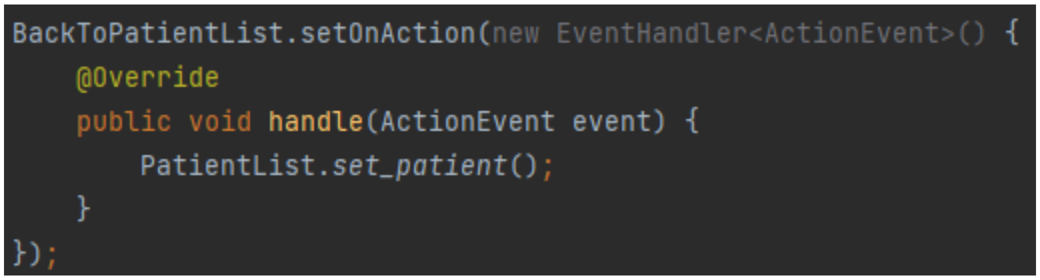
```java
VBox vbox = new VBox();
vbox.setAlignment(Pos.CENTER);
vbox.getChildren().add(teeth);
splitpane.getItems().add(vbox);
splitpane.setDividerPositions(0.5);
Scene patient = new Scene(splitpane, v: 850, v1: 500);
Main.stage.setScene(patient);

GridPane gridpane2 = new GridPane();

RowConstraints[] rowconstraints2 = new RowConstraints[32];
for (int i = 0; i < 32; i++) {
    rowconstraints2[i] = new RowConstraints( v: 10, v1: 50, Double.MAX_VALUE);
    (rowconstraints2[i]).setValignment(VPos.CENTER);
    (rowconstraints2[i]).setVgrow(Priority.SOMETIMES);
    gridpane2.getRowConstraints().add(rowconstraints2[i]);
}
```

Figure 70: GUI of the Patient page is created



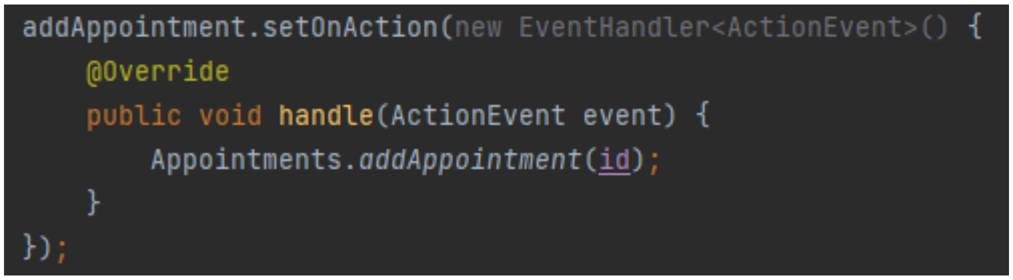
```java
Label[] array;
array = new Label[32];
int ind = 0;
for (int i = 1; i <= 32; i++) {
    String s = "";
    try {
        String sm = "SELECT * FROM teeth WHERE id=" + String.valueOf(id);
        Statement stat = db.createStatement();
        ResultSet res = stat.executeQuery(sm);
        while (res.next()) {
            if(res.getString( columnIndex: i+1)==null) s="";
            else s = res.getString( columnIndex: i+1);
        }
    } catch (Exception e) {System.out.println("error: failed to get notes from database");}
    if(!s.equals(""))
    {
        array[i-1] = new Label( s: "tooth number " + String.valueOf(i) + ": " + s);
        gridpane2.add(array[i-1], i: 0, ind++);
    }
}
splitpane.getItems().add(gridpane2);
```

Figure 71: Available notes are displayed on the right side of the Patient page as mentioned in criterion B



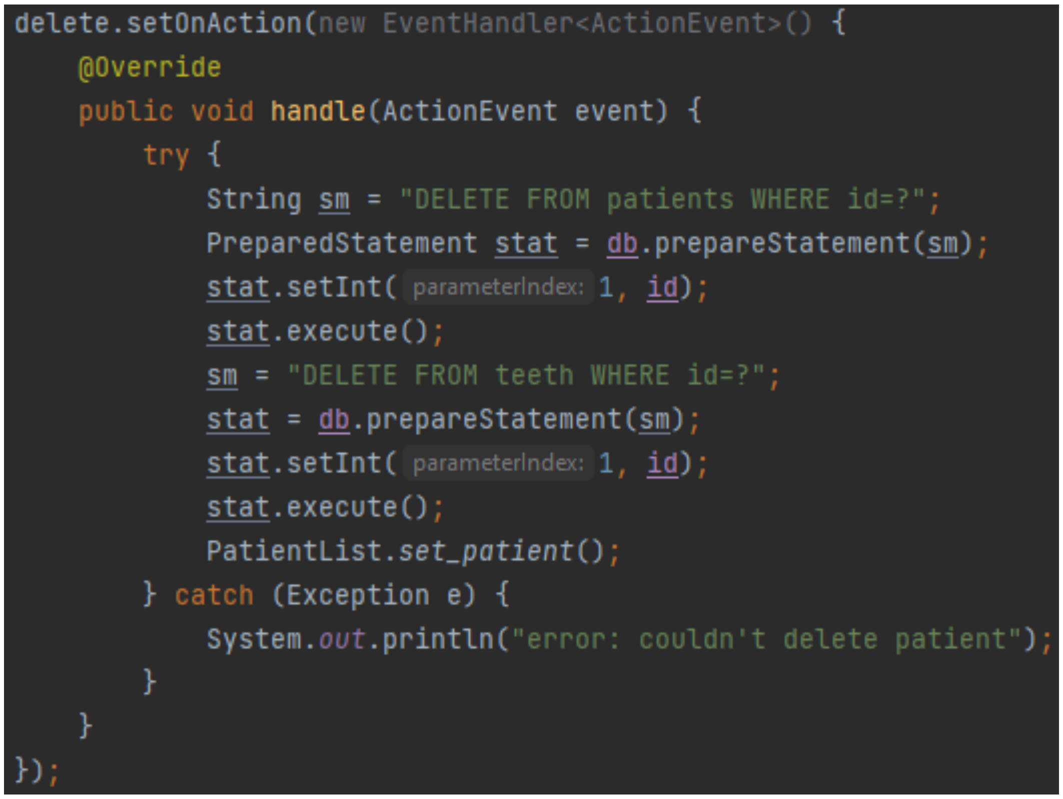
```java
BackToPatientList.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        PatientList.set_patient();
    }
});
```

Figure 72: Button "Back to Patients List" is clicked



```java
addAppointment.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        Appointments.addAppointment(id);
    }
});
```

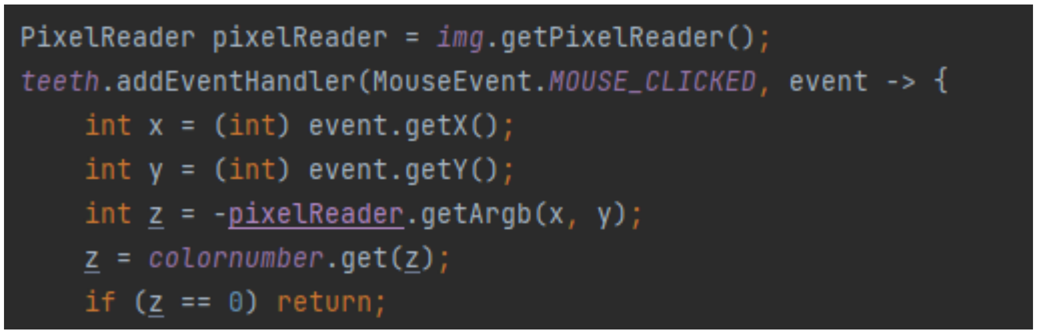
Figure 73: Button "Appointment" is clicked. A window opens to add a new appointment



```java
delete.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        try {
            String sm = "DELETE FROM patients WHERE id=?";
            PreparedStatement stat = db.prepareStatement(sm);
            stat.setInt( parameterIndex: 1, id);
            stat.execute();
            sm = "DELETE FROM teeth WHERE id=?";
            stat = db.prepareStatement(sm);
            stat.setInt( parameterIndex: 1, id);
            stat.execute();
            PatientList.set_patient();
        } catch (Exception e) {
            System.out.println("error: couldn't delete patient");
        }
    }
});
```

Figure 74: Button "Delete" is clicked

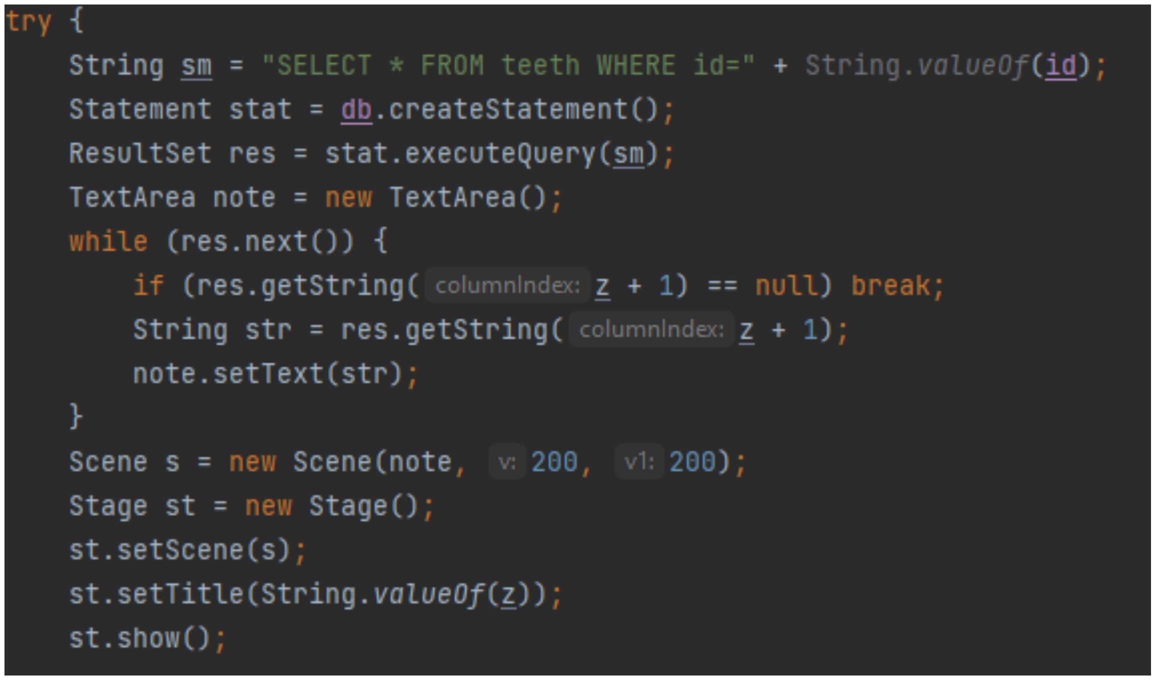
An SQL command "DELETE FROM table_name where id=value" is used to delete the patient from tables patient and teeth.



```
PixelReader pixelReader = img.getPixelReader();
teeth.addEventHandler(MouseEvent.MOUSE_CLICKED, event -> {
    int x = (int) event.getX();
    int y = (int) event.getY();
    int z = -pixelReader.getArgb(x, y);
    z = colornumber.get(z);
    if (z == 0) return;
```

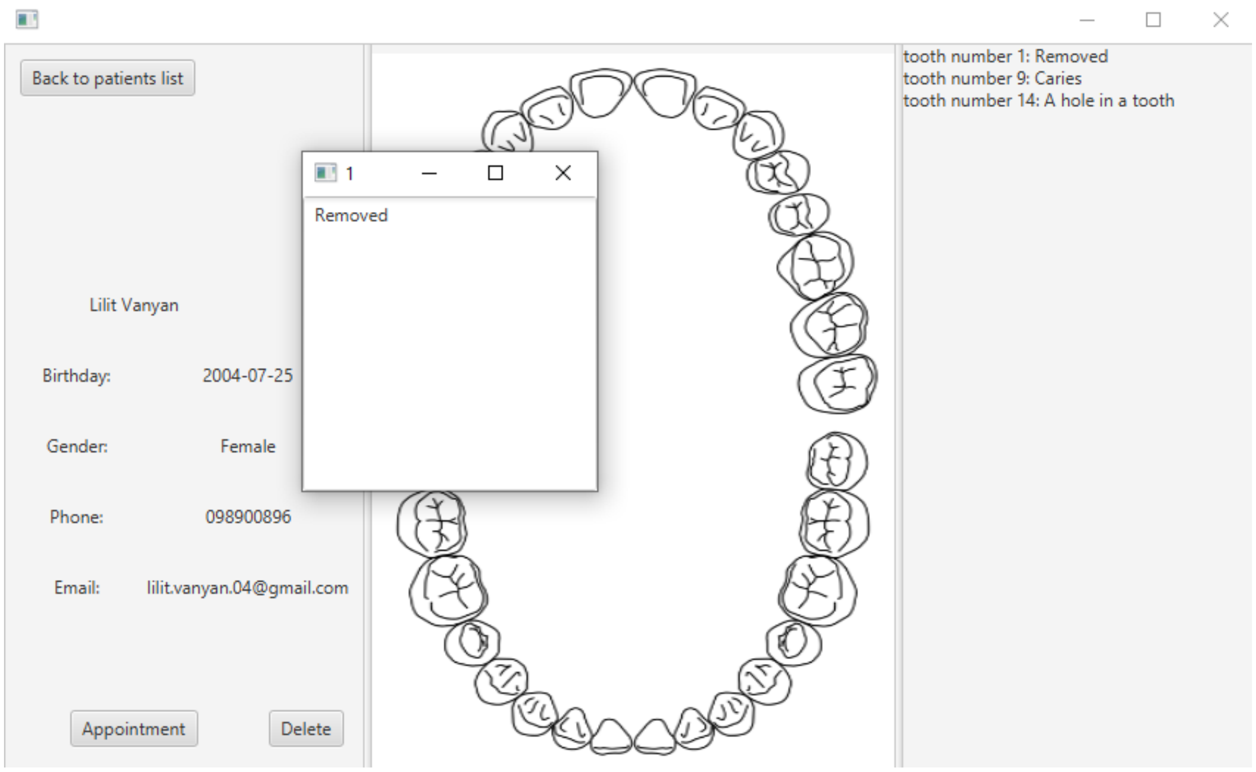
Figure 75: Tooth chart is clicked

This code finds the number of tooth that was clicked (criterion B).



```
try {
    String sm = "SELECT * FROM teeth WHERE id=" + String.valueOf(id);
    Statement stat = db.createStatement();
    ResultSet res = stat.executeQuery(sm);
    TextArea note = new TextArea();
    while (res.next()) {
        if (res.getString( columnIndex: z + 1) == null) break;
        String str = res.getString( columnIndex: z + 1);
        note.setText(str);
    }
    Scene s = new Scene(note,  v: 200,  v1: 200);
    Stage st = new Stage();
    st.setScene(s);
    st.setTitle(String.valueOf(z));
    st.show();
```

Figure 76: Tooth chart is clicked. A new window opens with the already existing note on it (Figure 77)



Back to patients list

1

Removed

Lilit Vanyan

Birthday: 2004-07-25

Gender: Female

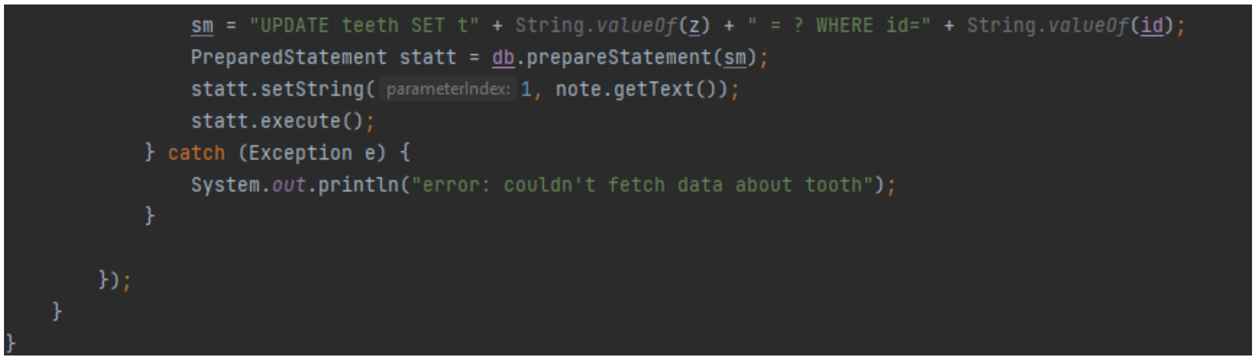
Phone: 098900896

Email: lilit.vanyan.04@gmail.com

Appointment Delete

tooth number 1: Removed
tooth number 9: Caries
tooth number 14: A hole in a tooth

Figure 77: A new window opens to make notes with the previous note on it.



```
                sm = "UPDATE teeth SET t" + String.valueOf(z) + " = ? WHERE id=" + String.valueOf(id);
                PreparedStatement statt = db.prepareStatement(sm);
                statt.setString( parameterIndex: 1, note.getText());
                statt.execute();
            } catch (Exception e) {
                System.out.println("error: couldn't fetch data about tooth");
            }

        });
    }
}
```

Figure 78: The changed note is saved in table teeth accordingly

Word count: 979 (not counting screenshots and captions)