Appendix B: Source Code

For better formatting and thus reading experience, please read the source code from .\Product\Project Folder\src

```
import controllers.main.MainController;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.io.*;
import java.util.Locale;

import static controllers.main.MainController.connectToDB;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Locale.setDefault(Locale.US);
        readDirectoryFile();
        connectToDB();
        Parent                              root                              =
FXMLLoader.load(getClass().getResource("/fxmls/Login.fxml"));
        primaryStage.setTitle("Room Allocation System");
        primaryStage.setScene(new Scene(root));
        primaryStage.setResizable(false);
        primaryStage.show();
        primaryStage.setOnCloseRequest(event                              ->
MainController.writeDirectoryFile());
    }

    public static void main(String[] args) {
        launch(args);
    }

    private void readDirectoryFile() {
        try {
            String                            directory                           =
(MainController.class.getProtectionDomain().getCodeSource().getLocation().t
oURI()).getPath().replace("\\", "/");
            directory = directory.substring(0, directory.lastIndexOf("/") + 1)
+ "/Directory.txt";
            File file = new File(directory);
            if (file.exists()) {
                BufferedReader br = new BufferedReader(new FileReader(file));
                MainController.fileName = br.readLine();
                MainController.directory = br.readLine();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
package GA;

import functional.Room;
import functional.StudentString;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class DNA {
    private List<StudentString> unallocatedStudents;
    private List<Room> genes;
    private int fitness;

    public boolean isNotValid() {
        return notValid;
    }

    private boolean notValid = false;

    public double getDeviationCount() {
        return deviationCount;
    }

    private double deviationCount;
    private List<Room> fixedGenes;
    private double fitnessFactor;

    public double getFitnessFactor() {
        return fitnessFactor;
    }

    DNA(DNA dna, List<Room> fixedGenes) {
        this.genes = deepCopyRooms(dna.getGenes());
        this.fixedGenes = deepCopyRooms(fixedGenes);
    }

    DNA(List<StudentString> unallocatedStudents, List<Room> fixedGenes) {
        this.unallocatedStudents = unallocatedStudents;
        this.fixedGenes = deepCopyRooms(fixedGenes);
        this.genes = deepCopyRooms(fixedGenes);
        buildDNA();
    }
    private void buildDNA() {
        Random random = new Random();
        for (StudentString unallocatedStudent: unallocatedStudents) {
            String sexRoom = "";
            Room gene;
            int bedsAvailable;
            do {
                gene = genes.get(random.nextInt(genes.size()));
                if (gene.getSexRoom().equals("Boy")) sexRoom = "male";
                if (gene.getSexRoom().equals("Girl")) sexRoom = "female";
                bedsAvailable        =        gene.getMaxResidents()        -
gene.getStudents().size();
            }    while    (!sexRoom.equals(unallocatedStudent.getSex())    ||
bedsAvailable == 0);
            gene.getStudents().add(unallocatedStudent);
        }
    }

    public List<Room> getGenes() {
        return genes;
    }

    public int getFitness() {
        return fitness;
    }

    public int calcFitness() {
        int fitness = 0;
        int mode;
        List<Integer> roomsStudents = new ArrayList<>();
        for (Room gene: genes) {
            roomsStudents.add(gene.getStudents().size());
        }
        mode = getMode(roomsStudents);
        deviationCount = 0;
        for (Room gene: genes) {
            deviationCount += Math.abs(gene.getStudents().size() - mode);
        }
        for (Room gene: genes) {
            List<StudentString> students = gene.getStudents();
            List<String> countries = new ArrayList<>();
            List<String> continents = new ArrayList<>();
            for (StudentString student: students) {
                countries.add(student.getCountry());
                continents.add(student.getContinent());
            }
            Collections.sort(countries);
            Collections.sort(continents);
            int continentConflicts = 0;
            int countryConflicts = 0;
            for (int i = 1; i < countries.size(); i++) {
                if (countries.get(i-1).equals(countries.get(i))) {
                    countryConflicts++;
                }
            }
            for (int i = 1; i < continents.size(); i++) {
                if (continents.get(i-1).equals(continents.get(i))) {
                    continentConflicts++;
                }
            }
            if (countryConflicts == 0 && continentConflicts == 0) {fitness +=
9;}
            else if (countryConflicts == 0 && continentConflicts == 1)
{fitness += 8;}
            else if (countryConflicts == 1 && continentConflicts == 1 &&
(gene.getStudents().size() == mode)) {fitness += 7;}
            else if (countryConflicts == 1 && continentConflicts == 1 &&
(gene.getStudents().size() < mode)) {fitness += 6;}
            else if (countryConflicts == 0 && continentConflicts == 2 &&
(gene.getStudents().size() == mode)) {fitness += 6;}
            else if (countryConflicts == 0 && continentConflicts == 2 &&
(gene.getStudents().size() < mode)) {fitness += 5;}
            else if (countryConflicts == 1 && continentConflicts == 2 &&
(gene.getStudents().size() == mode)) {fitness += 5;}
            else if (countryConflicts == 1 && continentConflicts == 2 &&
(gene.getStudents().size() < mode)) {fitness += 4;}
        }
        this.fitnessFactor = 1 / (1 + Math.pow(2.7, 0.05 * (deviationCount -
35)));
        this.fitness = Math.round((float) fitness * (float) fitness * (float)
fitnessFactor);
        return this.fitness;
    }

    public DNA mutate(double mutationRate) {
        Random random = new Random();
        if (random.nextDouble() < mutationRate) {
```

```
        if (random.nextDouble() < 0.3) {
            int maxStudents = genes.get(0).getStudents().size();
            int minStudents = genes.get(0).getStudents().size();
            for (Room gene: genes) {
                if (gene.getStudents().size() > maxStudents) maxStudents = 
gene.getStudents().size();
                else   if   (gene.getStudents().size()   <   minStudents) 
minStudents = gene.getStudents().size();
            }
            List<Room> maxStudentBoysRooms = new ArrayList<>();
            List<Room> maxStudentGirlsRooms = new ArrayList<>();
            List<Room> minStudentBoysRooms = new ArrayList<>();
            List<Room> minStudentGirlsRooms = new ArrayList<>();
            for (Room gene: genes) {
                if   (gene.getStudents().size()   ==   maxStudents   && 
gene.getSexRoom().equals("Boy")) maxStudentBoysRooms.add(gene);
                else  if  (gene.getStudents().size()  ==  maxStudents  && 
gene.getSexRoom().equals("Girl")) maxStudentGirlsRooms.add(gene);
                else  if  (gene.getStudents().size()  ==  minStudents  && 
gene.getSexRoom().equals("Boy")) minStudentBoysRooms.add(gene);
                else  if  (gene.getStudents().size()  ==  minStudents  && 
gene.getSexRoom().equals("Girl")) minStudentGirlsRooms.add(gene);
            }
            exchangeRoom(random,                    maxStudentBoysRooms, 
minStudentBoysRooms);
            exchangeRoom(random,                   maxStudentGirlsRooms, 
minStudentGirlsRooms);
        } else {
            Room room1;
            Room room2;
            do {
                room1 = genes.get(random.nextInt(genes.size()));
            } while (room1.getStudents().size() == 0);
            do {
                room2 = genes.get(random.nextInt(genes.size()));
            }          while          (room2          ==          room1 
||           !room2.getSexRoom().equals(room1.getSexRoom())           || 
room2.getStudents().size() == 0);
            StudentString student1 = null;
            StudentString student2 = null;
            int count = 0;
            do {
                student1                                               = 
room1.getStudents().get(random.nextInt(room1.getStudents().size()));
```

```
                    count++;
                }        while        (count        <        50        &&
fixedGenes.get(getIndex(room1)).getStudents().contains(student1));
                if (count == 50) {
                    student1 = null;
                } else {
                    count = 0;
                    do {
                        student2                                          =
room2.getStudents().get(random.nextInt(room2.getStudents().size()));
                        count++;
                    }        while        (count        <        50        &&
fixedGenes.get(getIndex(room2)).getStudents().contains(student2));
                    if (count == 50) student2 = null;
                }
                if (student1 != null && student2 != null) {
                    room1.getStudents().remove(student1);
                    room2.getStudents().remove(student2);
                    room1.getStudents().add(student2);
                    room2.getStudents().add(student1);
                }
            }

            /*}*/
            /*List<Room> modeRooms = new ArrayList<>();
            List<Room> deviatedRooms = new ArrayList<>();
            for (Room gene : genes) {
                if (Math.abs(gene.getStudents().size() - mode) >= 1)
                    deviatedRooms.add(gene);
                else if (gene.getStudents().size() == mode)
                    modeRooms.add(gene);
            }
            if (deviatedRooms.size() > 0) {
                Room                              room1                  =
deviatedRooms.get(random.nextInt(deviatedRooms.size()));
                Room room2;
                do {
                    room2 = modeRooms.get(random.nextInt(modeRooms.size()));
                } while (!room2.getSexRoom().equals(room1.getSexRoom()));
                StudentString student;
                int count = 0;
                if (room1.getStudents().size() < room2.getStudents().size())
{
                    do {
```

```
                    student                                                  =
room2.getStudents().get(random.nextInt(room2.getStudents().size()));
                    count++;
                }       while       (count       <       50       &&
fixedGenes.get(getIndex(room2)).getStudents().contains(student));
                if (count < 50){
                    room1.getStudents().add(student);
                }
            } else {
                do {
                    student                                                  =
room1.getStudents().get(random.nextInt(room1.getStudents().size()));
                    count++;
                }       while       (count       <       50       &&
fixedGenes.get(getIndex(room1)).getStudents().contains(student));
                if (count < 50){
                    room2.getStudents().add(student);
                }
            }
        } else {

        }*/
    }
    return new DNA(this, this.fixedGenes);
}

private  void  exchangeRoom(Random  random,  List<Room>  maxStudentRooms,
List<Room> minStudentRooms) {
    if (!maxStudentRooms.isEmpty() && !minStudentRooms.isEmpty()) {
        Room                        roomMore                                 =
maxStudentRooms.get(random.nextInt(maxStudentRooms.size()));
        Room                        roomLess                                 =
minStudentRooms.get(random.nextInt(minStudentRooms.size()));
        StudentString student;
        int count = 0;
        do {
            student                                                          =
roomMore.getStudents().get(random.nextInt(roomMore.getStudents().size()));
            count++;
        }        while        (count        <        50        &&
fixedGenes.get(getIndex(roomMore)).getStudents().contains(student));
        if (count < 50){
            roomLess.getStudents().add(student);
            roomMore.getStudents().remove(student);
```

```java
            }
        }
    }


    public int getMode(List<Integer> list) {
        int mode = list.get(0);
        int maxCount = 0;
        for (int i = 0; i < list.size(); i++) {
            int value = list.get(i);
            int count = 0;
            for (int j = 0; j < list.size(); j++) {
                if (list.get(j) == value) count++;
                if (count > maxCount) {
                    mode = value;
                    maxCount = count;
                }
            }
        }
        if (maxCount > 1) {
            return mode;
        }
        return 0;
    }

    public static List<Room> deepCopyRooms(List<Room> rooms) {
        List<Room> newRooms = new ArrayList<>();
        for (Room room: rooms) {
            newRooms.add(new Room(room));
        }
        return newRooms;
    }

    private int getIndex(Room room) {
        int index = -1;
        for (int i = 0; i < genes.size(); i++) {
            if (room.equals(genes.get(i)))
                index = i;
        }
        return index;
    }
}
```

```
package GA;

import functional.Room;
import functional.StudentString;
import controllers.main.MainController;

import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Population {

    public DNA getBestOne() {
        return bestOne;
    }

    private DNA bestOne;
    private int populationNum;
    private double mutationRate;
    private DNA[] population;
    private int generationCount = 1;
    private int sumFitness;
    private int counterForStop = 0;
    private DNA[] nextGeneration;
    private List<Integer> bestFitnesses = new ArrayList<>();
    private List<Room> fixedGenes;
    private int stoppingCondition;

    public List<StudentString> getUnallocatedStudents() {
        return unallocatedStudents;
    }

    private List<StudentString> unallocatedStudents;

    public   Population(int   populationNum,   double   mutationRate,   int
stoppingCondition, List<Room> fixedGenes, int year) {
        this.unallocatedStudents = findUnallocatedStudents(year);
        this.stoppingCondition = stoppingCondition;
        this.fixedGenes = fixedGenes;
        this.populationNum = populationNum;
        this.mutationRate = mutationRate;
        population = new DNA[populationNum];
        System.out.println("Generation " + generationCount);
        System.out.println("------------------------------");
        for (int i = 0; i < populationNum; i++) {
            this.population[i] = new DNA(unallocatedStudents, fixedGenes);
        }
    }

    public void naturalSelection() {
        Random random = new Random();
        nextGeneration = new DNA[populationNum];
        nextGeneration[0] = new DNA(bestOne, fixedGenes);
        for (int i = 1; i < populationNum; i++) {
            DNA parent;
            int randomFitness = random.nextInt(sumFitness);
            int index = -1;
            while (randomFitness >= 0) {
                randomFitness -= population[++index].getFitness();
            }
            parent = population[index];
            DNA child = parent.mutate(mutationRate);
            nextGeneration[i] = child;
        }
        System.arraycopy(nextGeneration,0,population,0,populationNum);
        generationCount++;
        System.out.println("Generation " + generationCount);
        System.out.println("------------------------------");
    }


    public void calcFitness() {
        int bestFitness = 0;
        sumFitness = 0;
        for (int i = 0; i < populationNum; i++) {
            int currentFitness = population[i].calcFitness();
            if (currentFitness > bestFitness) {
                bestOne = population[i];
                bestFitness = currentFitness;
            }
            sumFitness += currentFitness;
        }
        System.out.println(bestOne.getFitness()        +        "\n"        +
bestOne.getFitnessFactor() + "\n" + bestOne.getDeviationCount());
    }
    public boolean evaluate() {
        bestFitnesses.add(bestOne.getFitness());
        int size = bestFitnesses.size();
        if (size > 1) {
            if  (bestFitnesses.get(size-1).equals(bestFitnesses.get(size-2)))
{
                counterForStop++;
            }
            else {
                counterForStop = 0;
            }
        }
        return counterForStop <= stoppingCondition;
    }

    public static List<StudentString> findUnallocatedStudents(int year) {
        List<StudentString> unallocatedStudents = new ArrayList<>();
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            List<Integer> allocatedStudentIds = new ArrayList<>();
            while (rs.next()) {
                for (int i = 6; i <= numberOfColumns; i++) {
                    int studentId = rs.getInt(i);
                    if (studentId != 0)
                        allocatedStudentIds.add(studentId);
                }
            }
            ResultSet rs1 = stmt.executeQuery("SELECT Id FROM Students;");
            List<Integer> allStudentIds = new ArrayList<>();

            while (rs1.next()) {
                allStudentIds.add(rs1.getInt(1));
            }
            for (Integer allocatedStudentId: allocatedStudentIds) {
                allStudentIds.remove(allocatedStudentId);
            }
            List<Integer> unallocatedStudentIds = allStudentIds;
            for (Integer studentId: unallocatedStudentIds) {
                StudentString student = new StudentString();
                ResultSet  rs2  =  stmt.executeQuery("SELECT  *  FROM  Students
WHERE Id = " + studentId + ";");
                rs2.next();
                int studentYear = rs2.getInt("Year");
                if (year != 3 && studentYear != year) {
                    continue;
                }
                student.setId(studentId);
                student.setGivenName(rs2.getString("GivenName"));
                student.setFamilyName(rs2.getString("FamilyName"));
                student.setSex(rs2.getString("Sex"));
                student.setCountry(rs2.getString("Country"));
                student.setContinent(rs2.getString("Continent"));
                student.setYear(studentYear);
                unallocatedStudents.add(student);
            }
            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return unallocatedStudents;
    }
}
```

```
package functional;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.EventHandler;
import javafx.scene.control.ComboBox;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;

public class AutoCompleteComboBox {

    public interface AutoCompleteComparator<T> {
        boolean matches(String typedText, T objectToCompare);
    }

    public    static<T>    void    setAutoComplete(ComboBox<T>    comboBox,
AutoCompleteComparator<T> comparatorMethod) {
        ObservableList<T> data = comboBox.getItems();

        comboBox.setEditable(true);
        comboBox.getEditor().focusedProperty().addListener(observable -> {
            if (comboBox.getSelectionModel().getSelectedIndex() < 0) {
                comboBox.getEditor().setText(null);
            }
        });
        comboBox.addEventHandler(KeyEvent.KEY_PRESSED, t -> comboBox.hide());
        comboBox.addEventHandler(KeyEvent.KEY_RELEASED, new EventHandler<>()
{

            private boolean moveCaretToPos = false;
            private int caretPos;

            @Override
            public void handle(KeyEvent event) {
                if (event.getCode() == KeyCode.UP) {
                    caretPos = -1;
                    moveCaret(comboBox.getEditor().getText().length());
                    return;
                } else if (event.getCode() == KeyCode.DOWN) {
                    if (!comboBox.isShowing()) {
                        comboBox.show();
                    }
                    caretPos = -1;
                    moveCaret(comboBox.getEditor().getText().length());
                    return;
                } else if (event.getCode() == KeyCode.BACK_SPACE) {
                    moveCaretToPos = true;
                    caretPos = comboBox.getEditor().getCaretPosition();
                } else if (event.getCode() == KeyCode.DELETE) {
                    moveCaretToPos = true;
                    caretPos = comboBox.getEditor().getCaretPosition();
                } else if (event.getCode() == KeyCode.ENTER) {
                    return;
                }
                if (event.getCode() == KeyCode.RIGHT || event.getCode() ==
KeyCode.LEFT || event.getCode().equals(KeyCode.SHIFT) ||
event.getCode().equals(KeyCode.CONTROL)
                    || event.isControlDown() || event.getCode() ==
KeyCode.HOME
                    || event.getCode() == KeyCode.END || event.getCode()
== KeyCode.TAB) {
                    return;
                }

                ObservableList<T> list = FXCollections.observableArrayList();
                for (T aData : data) {
                    if (aData != null && comboBox.getEditor().getText() != null
&& comparatorMethod.matches(comboBox.getEditor().getText(), aData)) {
                        list.add(aData);
                    }
                }
                String t = comboBox.getEditor().getText();

                comboBox.setItems(list);
                comboBox.getEditor().setText(t);
                if (!moveCaretToPos) {
                    caretPos = -1;
                }
                moveCaret(t.length());
                if (!list.isEmpty()) {
                    comboBox.show();
                }
            }

            private void moveCaret(int textLength) {
                if (caretPos == -1) {
                    comboBox.getEditor().positionCaret(textLength);
                } else {
                    comboBox.getEditor().positionCaret(caretPos);
                }
                moveCaretToPos = false;
            }
        });
    }

    public static<T> T getComboBoxValue(ComboBox<T> comboBox){
        if (comboBox.getSelectionModel().getSelectedIndex() < 0) {
            return null;
        } else {
            return
comboBox.getItems().get(comboBox.getSelectionModel().getSelectedIndex());
        }
    }
}
```

```
package functional;

import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

import java.io.IOException;

public class HandleButton {

    public void handleCancelButton(Button cancelButton) {
        Stage currentStage = (Stage) cancelButton.getScene().getWindow();
        currentStage.close();
    }

    public  void  handleNextButton(Button  button,  String  fxml)  throws
IOException {
        Parent layout = FXMLLoader.load(getClass().getResource(fxml));
        Stage currentStage = (Stage) button.getScene().getWindow();
        currentStage.setScene(new Scene(layout));
        currentStage.setResizable(false);
        currentStage.centerOnScreen();
        currentStage.show();
    }

    public  void  handlePreviousButton(Button  previousButton,  String  fxml)
throws IOException{
        handleNextButton(previousButton,fxml);
    }
}
```

```
package functional;

import controllers.main.MainController;
import javafx.beans.property.SimpleStringProperty;

import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class Room {
    private int id;
    private SimpleStringProperty room;
    private SimpleStringProperty building;
    private int maxResidents;
    private SimpleStringProperty sexRoom;
    private List<StudentString> students = new ArrayList<>();

    public Room() {

    }

    public Room(Room room) {
        this.id = room.getId();
        this.room = new SimpleStringProperty(room.getRoom());
        this.building = new SimpleStringProperty(room.getBuilding());
        this.maxResidents = room.getMaxResidents();
        this.sexRoom = new SimpleStringProperty(room.getSexRoom());
        this.students = new ArrayList<>(room.getStudents());
    }

    public List<StudentString> getStudents() {
        return students;
    }

    public boolean isEqualTo(Room room) {
        return            (this.getRoom().equals(room.getRoom())            &&
this.getBuilding().equals(room.getBuilding()));
    }

    public String getSexRoom() {
        return sexRoom.get();
    }
    public void setSexRoom(String sexRoom) {
        this.sexRoom = new SimpleStringProperty(sexRoom);
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getRoom() {
        return room.get();
    }

    public void setRoom(String room) {
        this.room = new SimpleStringProperty(room);
    }

    public String getBuilding() {
        return building.get();
    }

    public void setBuilding(String building) {
        this.building = new SimpleStringProperty(building);
    }

    public int getMaxResidents() {
        return maxResidents;
    }

    public void setMaxResidents(int maxResidents) {
        this.maxResidents = maxResidents;
    }

    public boolean isEmpty() {
        boolean isEmpty = true;
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE \"Room
No./Name\"  =  \""  +  room.get()  +  "\"  AND  \"Building  No./Name\"  =  \""  +
building.get() + "\";");
```

```
            ResultSetMetaData rsmd = rs.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            rs.next();
            for (int i = 6; i <= numberOfColumns; i++) {
                if (rs.getInt(i) != 0) {
                    isEmpty = false;
                    break;
                }
                if (!isEmpty) break;
            }
            stmt.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
        return isEmpty;
    }
}
```

```
package functional;

import controllers.main.MainController;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ComboBox;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.Locale;

public class Student {

    private int id;
    private SimpleStringProperty givenName;
    private SimpleStringProperty familyName;
    private SimpleIntegerProperty year;
    private SimpleStringProperty sex;
    private ComboBox<String> countryCB;
    private SimpleStringProperty continent;
    private CheckBox allocated = new CheckBox();
    public CheckBox getAllocated() {
        return allocated;
    }
    public Student() {
        Locale.setDefault(Locale.US);
        allocated.setOnAction(event -> {
            if (allocated.isSelected()) allocated.setSelected(false);
            else allocated.setSelected(true);
        });
        ObservableList<String>                  countries                    =
FXCollections.observableArrayList();
        String[] countryCodes = Locale.getISOCountries();
        for (String countryCode: countryCodes) {
            Locale locale = new Locale("", countryCode);
            countries.add(locale.getDisplayCountry());
        }
        countryCB = new ComboBox<>();
        countryCB.setItems(countries);
        AutoCompleteComboBox.setAutoComplete(countryCB,          (typedText,
itemToCompare)                                                            ->
itemToCompare.toLowerCase().contains(typedText.toLowerCase())             ||
itemToCompare.equals(typedText));
    }
    public Integer getYear() {
        return year.get();
    }

    public void setYear(Integer year) {
        this.year = new SimpleIntegerProperty(year);
    }

    public String getSex() {
        return sex.get();
    }

    public void setCountryValue(String countryValue) {
        countryCB.setValue(countryValue);
    }

    public void setSex(String sex) {
        this.sex = new SimpleStringProperty(sex);
    }

    public ComboBox<String> getCountryCB() {
        return countryCB;
    }

    public void setCountryCB(ComboBox<String> countryCB) {
        this.countryCB = countryCB;
    }

    public String getContinent() {
        return continent.get();
    }

    public void setContinent(String continent) {
        this.continent = new SimpleStringProperty(continent);
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
```

```java
        this.id = id;
    }

    public String getGivenName() {
        return givenName.get();
    }

    public void setGivenName(String givenName) {
        this.givenName = new SimpleStringProperty(givenName);
    }

    public String getFamilyName() {
        return familyName.get();
    }

    public void setFamilyName(String familyName) {
        this.familyName = new SimpleStringProperty(familyName);
    }

    public static boolean isAllocated(int id) {
        boolean isAllocated = false;
        try {

            Statement stmt = MainController.c.createStatement();

            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            while (rs.next()) {
                for (int i = 6; i <= numberOfColumns; i++) {
                    if (rs.getInt(i) == id) {
                        isAllocated = true;
                        break;
                    }
                }
                if (isAllocated) break;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return isAllocated;
    }
}
```

```
package functional;

import controllers.main.MainController;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.scene.control.CheckBox;

import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class StudentString {
    private int id;
    private SimpleStringProperty givenName;
    private SimpleStringProperty familyName;
    private SimpleIntegerProperty year;
    private SimpleStringProperty sex;
    private SimpleStringProperty country;
    private SimpleStringProperty continent;
    private CheckBox allocated = new CheckBox();

    public StudentString() {
    }

    public StudentString(StudentString student) {
        this.id = student.getId();
        this.givenName = new SimpleStringProperty(student.getGivenName());
        this.familyName = new SimpleStringProperty(student.getFamilyName());
        this.year = new SimpleIntegerProperty(student.getYear());
        this.sex = new SimpleStringProperty(student.getSex());
        this.country = new SimpleStringProperty(student.getCountry());
        this.continent = new SimpleStringProperty(student.getContinent());
        this.allocated = new CheckBox();
    }

    public CheckBox getAllocated() {
        return allocated;
    }


    public int getId() {
        return id;
    }
```

```
public void setId(int id) {
    this.id = id;
}

public String getGivenName() {
    return givenName.get();
}

public void setGivenName(String givenName) {
    this.givenName = new SimpleStringProperty(givenName);
}

public String getFamilyName() {
    return familyName.get();
}

public void setFamilyName(String familyName) {
    this.familyName = new SimpleStringProperty(familyName);
}

public void setYear(int year) {
    this.year = new SimpleIntegerProperty(year);
}

public int getYear() {
    return year.get();
}

public String getSex() {
    return sex.get();
}

public void setSex(String sex) {
    this.sex = new SimpleStringProperty(sex);
}

public String getCountry() {
    return country.get();
}

public void setCountry(String country) {
    this.country = new SimpleStringProperty(country);
}
```

```
public String getContinent() {
    return continent.get();
}

public void setContinent(String continent) {
    this.continent = new SimpleStringProperty(continent);
}

public String getRoom() {
    String room = null;
    try {
        Statement stmt = MainController.c.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
        ResultSetMetaData rsmd = rs.getMetaData();
        int numberOfColumns = rsmd.getColumnCount();
        while (rs.next()) {
            for (int i = 6; i <= numberOfColumns; i++) {
                if (rs.getInt(i) == this.id) {
                    room = rs.getString(2);
                }
            }
        }
        stmt.close();
        return room;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

public String getBuilding() {
    String building = null;
    try {
        Statement stmt = MainController.c.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
        ResultSetMetaData rsmd = rs.getMetaData();
        int numberOfColumns = rsmd.getColumnCount();
        while (rs.next()) {
            for (int i = 6; i <= numberOfColumns; i++) {
                if (rs.getInt(i) == this.id) {
                    building = rs.getString(3);
                }
            }
        }
```

```
            stmt.close();

            return building;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    public boolean isAllocated(int id) {
        return Student.isAllocated(id);
    }
}
```

```
package controllers.configurations;

import functional.HandleButton;
import functional.Room;
import controllers.main.MainController;
import controllers.newFile.RoomConfigController;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;

import java.net.URL;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.ResourceBundle;

public class AddOrDeleteRoomController implements Initializable {

    private        ObservableList<Room>        roomsObservableList        =
FXCollections.observableArrayList();

    private List<Room> addedRooms = new ArrayList<>();
    private List<Room> deletedRooms = new ArrayList<>();

    @FXML
    private TableView<Room> roomTableView;

    @FXML
    private TableColumn<Room, Integer> idColumn;

    @FXML
    private TableColumn<Room, String> roomColumn;

    @FXML
    private TableColumn<Room, String> buildingColumn;
    @FXML
    private TableColumn<Room, Integer> maxResidentsColumn;

    @FXML
    private TableColumn<Room, String> sexRoomColumn;

    @FXML
    private ComboBox<String> sexComboBox;

    @FXML
    private Button previousButton;

    @FXML
    private Button finishButton;

    @FXML
    private Button cancelButton;

    @FXML
    private Button addButton;

    @FXML
    private Button deleteButton;

    @FXML
    private TextField roomTextField;

    @FXML
    private TextField buildingTextField;

    @FXML
    private TextField maxResidentsTextField;

    @FXML
    void addClick(ActionEvent event) {
        Room addedRoom = RoomConfigController.addButtonClicked(roomTextField,
buildingTextField, maxResidentsTextField, sexComboBox, roomsObservableList,
roomTableView);
        if (addedRoom != null) addedRooms.add(addedRoom);
        roomTextField.clear();
        buildingTextField.clear();
        maxResidentsTextField.clear();
        sexComboBox.setValue("");
    }
    @FXML
    void cancelClick(ActionEvent event) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText("Confirmation Dialog");
        alert.setContentText("If  you  cancel,  all  the  changes  will  be
lost.\nAre you sure to cancel?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            HandleButton button = new HandleButton();
            button.handleCancelButton(cancelButton);
        }
    }

    @FXML
    void deleteClick(ActionEvent event) {
        if (roomTableView.getSelectionModel().isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Information Dialog");
            alert.setHeaderText(null);
            alert.setContentText("Please Choose a Row to Delete!");
            alert.showAndWait();
        } else {
            Room                        roomToDelete                        =
roomTableView.getSelectionModel().getSelectedItem();
            if (addedRooms.contains(roomToDelete)) {
                addedRooms.remove(roomToDelete);
                roomsObservableList.remove(roomToDelete);
                roomTableView.refresh();
            } else {
                String room = roomToDelete.getRoom();
                String building = roomToDelete.getBuilding();
                try {
                    Statement stmt = MainController.c.createStatement();
                    ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE
\"Room  No./Name\"  =  \""  +  room  +  "\"  AND  \"Building  No./Name\"  =  \""  +
building + "\";");
                    rs.next();
                    int studentCount = 0;
                    for  (int  i  =  6;  i  <  6  +  roomToDelete.getMaxResidents();
i++) {
                        int studentId = rs.getInt(i);
                        if (studentId != 0 ) {
```

```
                        studentCount++;
                    }
                }
                stmt.close();
                if (studentCount != 0) {
                    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
                    alert.setTitle("Confirmation Dialog");
                    alert.setHeaderText("Confirmation Dialog");
                    alert.setContentText("There is/are " + studentCount +
" student(s) allocated in this room.\n" +
                            "If   you   delete,   the   students   will   be
unallocated.\n" +
                            "Are you sure to delete the room?");
                    Optional<ButtonType> result = alert.showAndWait();
                    if (result.get() == ButtonType.OK) {
                        roomsObservableList                               =
RoomConfigController.deleteButtonClicked(roomTableView,
roomsObservableList);
                        deletedRooms.add(roomToDelete);
                    }
                } else {
                    roomsObservableList                                   =
RoomConfigController.deleteButtonClicked(roomTableView,
roomsObservableList);
                    deletedRooms.add(roomToDelete);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

@FXML
void finishClick(ActionEvent event) {
    try {
        if (deletedRooms.size() != 0 || addedRooms.size() != 0) {


            Statement stmt = MainController.c.createStatement();

            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
```

```
                rs.close();
                for (Room roomToDelete: deletedRooms) {
                    stmt.executeUpdate("DELETE   FROM   Rooms   WHERE   \"Room
No./Name\" = \"" + roomToDelete.getRoom() + "\" AND \"Building No./Name\" =
\"" + roomToDelete.getBuilding() + "\";");
                    MainController.c.commit();
                }
                for (Room roomToAdd: addedRooms) {
                    stmt.executeUpdate("INSERT  INTO  Rooms  ('Room  No./Name',
'Building   No./Name',   'Max   Residents',   'Boy/girl')   VALUES   (\""   +
roomToAdd.getRoom()  +  "\",  \""  +  roomToAdd.getBuilding()  +  "\",  "  +
roomToAdd.getMaxResidents() + ", \"" + roomToAdd.getSexRoom() + "\");");
                    MainController.c.commit();
                }
                int maxRoomCapacity = 0 ;
                for (Room room: roomsObservableList) {
                    if (room.getMaxResidents() > maxRoomCapacity) {
                        maxRoomCapacity = room.getMaxResidents();
                    }
                }
                if (maxRoomCapacity > (numberOfColumns - 5)) {
                    for (int i = (numberOfColumns - 5); i < maxRoomCapacity;
i++) {
                        stmt.executeUpdate("ALTER   TABLE   Rooms   ADD   COLUMN
'Student " + (i + 1) + "' INTEGER;");
                        MainController.c.commit();
                    }
                }
                stmt.close();

            }
            Stage currentStage = (Stage) roomTableView.getScene().getWindow();
            currentStage.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        idColumn.setSortable(false);
        roomColumn.setSortable(false);
        buildingColumn.setSortable(false);
        maxResidentsColumn.setSortable(false);
        sexRoomColumn.setSortable(false);
        sexComboBox.getItems().addAll("Boy", "Girl");
        idColumn.setStyle("-fx-alignment: CENTER;");
        roomColumn.setStyle("-fx-alignment: CENTER;");
        buildingColumn.setStyle("-fx-alignment: CENTER;");
        maxResidentsColumn.setStyle("-fx-alignment: CENTER;");
        sexRoomColumn.setStyle("-fx-alignment: CENTER;");
        idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));
        roomColumn.setCellValueFactory(new PropertyValueFactory<>("room"));
        buildingColumn.setCellValueFactory(new
PropertyValueFactory<>("building"));
        maxResidentsColumn.setCellValueFactory(new
PropertyValueFactory<>("maxResidents"));
        sexRoomColumn.setCellValueFactory(new
PropertyValueFactory<>("sexRoom"));
        roomsObservableList = RoomConfigController.populateTableView();
        roomsObservableList.sort(MainController::roomComparator);
        roomTableView.setItems(roomsObservableList);
    }

    public static boolean contains(List<Room> rooms, Room room) {
        if (room.getId() == 0) {
            boolean flag = false;
            for (Room room1: rooms) {
                if (room.isEqualTo(room1)) {
                    flag = true;
                    break;
                }
            }
            return flag;
        } else {
            int occurrences = 0;
            for (Room room1: rooms) {
                if (room.isEqualTo(room1)) {
                    occurrences++;
                }
            }
            return occurrences > 1;
        }
    }
}
```

```
package controllers.configurations;

import functional.Room;
import controllers.main.MainController;
import controllers.newFile.RoomConfigController;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.ButtonType;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.ComboBoxTableCell;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.stage.Stage;
import javafx.util.converter.IntegerStringConverter;

import java.net.URL;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.Optional;
import java.util.ResourceBundle;

public class UpdateRoomInfoController implements Initializable {

    private        ObservableList<Room>        roomsObservableList        =
FXCollections.observableArrayList();

    @FXML
    private TableView<Room> roomTableView;

    @FXML
    private TableColumn<Room, Integer> idColumn;

    @FXML
    private TableColumn<Room, String> roomColumn;

    @FXML
    private TableColumn<Room, String> buildingColumn;
    @FXML
    private TableColumn<Room, Integer> maxResidentsColumn;

    @FXML
    private TableColumn<Room, String> sexRoomColumn;

    @FXML
    public void changeRoomSexEvent(TableColumn.CellEditEvent editedCell) {
        Room                            roomSelected                         =
roomTableView.getSelectionModel().getSelectedItem();
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Dialog");
            alert.setHeaderText("Confirmation Dialog");
            alert.setContentText("Changing     the     room     to     "     +
editedCell.getNewValue().toString().toLowerCase() + "'s room will cause the
original students allocated in this room deleted!\n" +
                    "Are you sure to change it?");
            Optional<ButtonType> result = alert.showAndWait();
            if (result.get() == ButtonType.OK) {
                try {
                    Statement stmt = MainController.c.createStatement();
                    String room = roomSelected.getRoom();
                    String building = roomSelected.getBuilding();
                    for (int i = 1; i < 1 + roomSelected.getMaxResidents();
i++) {
                        stmt.executeUpdate("UPDATE Rooms SET \"Student " + i +
"\" = NULL WHERE \"Room No./Name\" = \"" + room + "\" AND \"Building No./Name\"
= \"" + building + "\";");
                        MainController.c.commit();
                    }
                    String sexRoom = "";
                    if  (roomSelected.getSexRoom().equals("Boy"))  sexRoom  =
"Girl";
                    if  (roomSelected.getSexRoom().equals("Girl"))  sexRoom  =
"Boy";
                    stmt.executeUpdate("UPDATE Rooms SET \"Boy/Girl\" = \"" +
sexRoom  +  "\"  WHERE  \"Room  No./Name\"  =  \""  +  room  +  "\"  AND  \"Building
No./Name\" = \"" + building + "\";");
                    MainController.c.commit();
                    stmt.close();
roomSelected.setSexRoom(editedCell.getNewValue().toString());
                    roomTableView.refresh();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                roomSelected.setSexRoom(editedCell.getOldValue().toString());
                roomTableView.refresh();
            }
        }
    }


    @FXML
    public void changeMaxCapacityEvent(TableColumn.CellEditEvent editedCell)
{
        Room                          roomSelected                        =
roomTableView.getSelectionModel().getSelectedItem();
        int studentCount = 0;
        int oldMaxCapacity = 0;
        try {
            Statement stmt = MainController.c.createStatement();
            String room = roomSelected.getRoom();
            String building = roomSelected.getBuilding();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE \"Room
No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" + building +
"\";");
            rs.next();
            for (int i = 6; i < 6 + roomSelected.getMaxResidents(); i++) {
                int studentId = rs.getInt(i);
                if (studentId != 0 ) {
                    studentCount++;
                }
            }
            ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs1.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            oldMaxCapacity = numberOfColumns - 5;
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            int                         changedCapacity                       =
Integer.parseInt(editedCell.getNewValue().toString());
            if (studentCount > changedCapacity) {

roomSelected.setMaxResidents(Integer.parseInt(editedCell.getOldValue().toSt
ring()));
                roomTableView.refresh();
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("An Error has Occurred!");
                alert.setContentText(studentCount + " students have been
allocated in this room!\n" +
                        "Delete " + (studentCount - changedCapacity) + "
student(s) to reduce the maximum residents to " + changedCapacity + "!");
                alert.showAndWait();
            } else {
                try {
                    Statement stmt = MainController.c.createStatement();
                    String room = roomSelected.getRoom();
                    String building = roomSelected.getBuilding();
                    stmt.executeUpdate("UPDATE Rooms SET 'Max Residents' = " +
changedCapacity + " WHERE \"Room No./Name\" = \"" + room + "\" AND \"Building
No./Name\" = \"" + building + "\";");
                    MainController.c.commit();
                    if (changedCapacity > oldMaxCapacity) {
                        for (int i = oldMaxCapacity; i < changedCapacity; i++)
{
                            stmt.executeUpdate("ALTER TABLE Rooms ADD COLUMN
'Student " + (i+1) + "' INTEGER;");
                            MainController.c.commit();
                        }
                    }
                    stmt.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                roomSelected.setMaxResidents(changedCapacity);
            }
        } catch (Exception e) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error Dialog");
            alert.setHeaderText("An Error has Occurred!");
            alert.setContentText("Please Enter an INTEGER for Maximum
Residents!");
            alert.showAndWait();
        }
    }

    @FXML
    void changeRoomCellEvent(TableColumn.CellEditEvent editedCell) {
        Room                          roomSelected                         =
roomTableView.getSelectionModel().getSelectedItem();
        String newRoomName = editedCell.getNewValue().toString();
        String oldRoomName = editedCell.getOldValue().toString();
        if (!newRoomName.equals(oldRoomName)) {
            roomSelected.setRoom(newRoomName);
            roomTableView.refresh();
            if      (AddOrDeleteRoomController.contains(roomsObservableList,
roomSelected)) {
                roomSelected.setRoom(oldRoomName);
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("An Error has Occurred!");
                alert.setContentText("This   room   already   exists   in   the
scheme!");
                alert.showAndWait();
            } else {
                try {


                    Statement stmt = MainController.c.createStatement();

                    String building = roomSelected.getBuilding();
                    stmt.executeUpdate("UPDATE Rooms SET \"Room No./Name\" =
\"" + newRoomName + "\" WHERE \"Room No./Name\" = \"" + oldRoomName + "\"
AND \"Building No./Name\" = \"" + building + "\";");
                    MainController.c.commit();
                    stmt.close();

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }


    @FXML
    void changeBuildingCellEvent(TableColumn.CellEditEvent editedCell) {
        Room                          roomSelected                          =
roomTableView.getSelectionModel().getSelectedItem();
        String newBuildingName = editedCell.getNewValue().toString();
        String oldBuildingName = editedCell.getOldValue().toString();
        if (!newBuildingName.equals(oldBuildingName)) {
            roomSelected.setBuilding(newBuildingName);
            if      (AddOrDeleteRoomController.contains(roomsObservableList,
roomSelected)) {
                roomSelected.setBuilding(oldBuildingName);
                roomTableView.refresh();
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("An Error has Occurred!");
                alert.setContentText("This   room   already   exists   in   the
scheme!");
                alert.showAndWait();
            } else {
                try {
                    Statement stmt = MainController.c.createStatement();
                    String room = roomSelected.getRoom();
                    stmt.executeUpdate("UPDATE Rooms SET \"Building No./Name\"
= \"" + newBuildingName + "\" WHERE \"Room No./Name\" = \"" + room + "\" AND
\"Building No./Name\" = \"" + oldBuildingName + "\";");
                    MainController.c.commit();
                    stmt.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }



    @FXML
    void finishClick(ActionEvent event) {
        Stage currentStage = (Stage) roomTableView.getScene().getWindow();
        currentStage.close();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        buildingColumn.setSortable(true);
        sexRoomColumn.setSortable(true);
        roomTableView.setEditable(true);
        roomColumn.setEditable(true);
        buildingColumn.setEditable(true);
        maxResidentsColumn.setEditable(true);
        sexRoomColumn.setEditable(true);
        idColumn.setEditable(false);
        roomColumn.setCellFactory(TextFieldTableCell.forTableColumn());
        buildingColumn.setCellFactory(TextFieldTableCell.forTableColumn());

maxResidentsColumn.setCellFactory(TextFieldTableCell.forTableColumn(new
IntegerStringConverter()));
        sexRoomColumn.setCellFactory(ComboBoxTableCell.forTableColumn("Girl",
"Boy"));
        idColumn.setStyle("-fx-alignment: CENTER;");
        roomColumn.setStyle("-fx-alignment: CENTER;");
        buildingColumn.setStyle("-fx-alignment: CENTER;");
        maxResidentsColumn.setStyle("-fx-alignment: CENTER;");
        sexRoomColumn.setStyle("-fx-alignment: CENTER;");
        idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));
        roomColumn.setCellValueFactory(new PropertyValueFactory<>("room"));
        buildingColumn.setCellValueFactory(new
PropertyValueFactory<>("building"));
        maxResidentsColumn.setCellValueFactory(new
PropertyValueFactory<>("maxResidents"));
        sexRoomColumn.setCellValueFactory(new
PropertyValueFactory<>("sexRoom"));
        roomsObservableList = RoomConfigController.populateTableView();
        roomsObservableList.sort(MainController::roomComparator);
        roomTableView.setItems(roomsObservableList);
    }

}
```

```
package controllers.configurations;


import controllers.newFile.StudentConfig2Controller;
import controllers.newFile.StudentConfigController;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.stage.FileChooser;
import javafx.stage.Modality;
import javafx.stage.Stage;

import java.io.File;

public class UploadYear1StudentController {

    private int id = 1;

    @FXML
    Button uploadButton;

    @FXML
    void uploadClicked(ActionEvent event) {
        Stage mainStage = null;
        final FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().addAll(new
FileChooser.ExtensionFilter("CSV Files", "*.csv"));
        File selectedFile = fileChooser.showOpenDialog(mainStage);
        if (selectedFile != null) {
            try {
                StudentConfigController.writeToDB(selectedFile, 1, this.id);
                this.id++;
                Alert alert = new Alert(Alert.AlertType.INFORMATION);
                alert.setTitle("Information Dialog");
                alert.setHeaderText(null);
                alert.setContentText("The    file    has    been    successfully
uploaded!");
                alert.showAndWait();
                Stage stage =  (Stage) uploadButton.getScene().getWindow();
                stage.close();
                showNextStage();
            } catch (Exception e) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("An Error Occurred!");
                alert.setContentText("Please make sure the format of the CSV
file and upload again");
            }
        }
    }

    private void showNextStage() {
        try {
            Stage stage = new Stage();
            FXMLLoader                loader                =                new
FXMLLoader(getClass().getResource("/fxmls/newFile/StudentConfig2.fxml"));
            stage.setScene(new Scene(loader.load()));
            stage.setTitle("Student Configuration");
            StudentConfig2Controller controller = loader.getController();
            stage.setOnShown(event1 -> {
                controller.setDeleteDB(false);
                controller.getCancelButton().setDisable(true);
            });
            stage.setOnCloseRequest(event1 -> {
                Alert alert = new Alert(Alert.AlertType.WARNING);
                alert.setTitle("Warning Dialog");
                alert.setHeaderText("Warning Dialog");
                alert.setContentText("You  must  complete  the  table  before
exiting!");
                alert.showAndWait();
                event1.consume();
            });
            stage.initModality(Modality.APPLICATION_MODAL);
            stage.showAndWait();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
package controllers.configurations;


import controllers.newFile.StudentConfig2Controller;
import controllers.newFile.StudentConfigController;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.stage.FileChooser;
import javafx.stage.Modality;
import javafx.stage.Stage;

import java.io.File;

public class UploadYear2StudentController {

    private int id = 1;

    @FXML
    Button uploadButton;

    @FXML
    void uploadClicked(ActionEvent event) {
        Stage mainStage = null;
        final FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().addAll(new
FileChooser.ExtensionFilter("CSV Files", "*.csv"));
        File selectedFile = fileChooser.showOpenDialog(mainStage);
        if (selectedFile != null) {
            try {
                StudentConfigController.writeToDB(selectedFile, 2, this.id);
                this.id++;
                Alert alert = new Alert(Alert.AlertType.INFORMATION);
                alert.setTitle("Information Dialog");
                alert.setHeaderText(null);
                alert.setContentText("The    file    has    been    successfully
uploaded!");
                alert.showAndWait();
                Stage stage =  (Stage) uploadButton.getScene().getWindow();
                stage.close();
                showNextStage();
            } catch (Exception e) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("An Error Occurred!");
                alert.setContentText("Please make sure the format of the CSV
file and upload again");
            }
        }
    }

    private void showNextStage() {
        try {
            Stage stage = new Stage();
            FXMLLoader                loader                =                new
FXMLLoader(getClass().getResource("/fxmls/newFile/StudentConfig2.fxml"));
            stage.setScene(new Scene(loader.load()));
            stage.setTitle("Student Configuration");
            StudentConfig2Controller controller = loader.getController();
            stage.setOnShown(event1 -> {
                controller.setDeleteDB(false);
                controller.getCancelButton().setDisable(true);
            });
            stage.setOnCloseRequest(event1 -> {
                Alert alert = new Alert(Alert.AlertType.WARNING);
                alert.setTitle("Warning Dialog");
                alert.setHeaderText("Warning Dialog");
                alert.setContentText("You  must  complete  the  table  before
exiting!");
                alert.showAndWait();
                event1.consume();
            });
            stage.initModality(Modality.APPLICATION_MODAL);
            stage.showAndWait();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
package controllers.configurations;

import controllers.main.MainController;
import functional.AutoCompleteComboBox;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import java.io.*;
import java.net.URL;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;
import java.util.ResourceBundle;

public class Year1StudentAddClickedController implements Initializable {

    public boolean isOkButtonClicked() {
        return okButtonClicked;
    }

    private boolean okButtonClicked = false;

    private Map<String,String> countries = new HashMap<>();

    public TextField getFamilyNameTF() {
        return familyNameTF;
    }

    public TextField getGivenNameTF() {
        return givenNameTF;
    }

    public ComboBox<String> getSexCB() {
        return sexCB;
    }

    public ComboBox<String> getCountryCB() {
        return countryCB;
    }
    public TextField getContinentTF() {
        return continentTF;
    }

    private Map<String,String> continents = new HashMap<>();

    @FXML
    private TextField familyNameTF;

    @FXML
    private TextField givenNameTF;

    @FXML
    private ComboBox<String> sexCB;

    @FXML
    private ComboBox<String> countryCB;

    @FXML
    private TextField continentTF;

    @FXML
    void okClicked(ActionEvent event) {
        if                (givenNameTF.getText().isEmpty()                 ||
familyNameTF.getText().isEmpty()) {
            showAlert();
        } else {
            try {
                sexCB.getValue().isEmpty();
                AutoCompleteComboBox.getComboBoxValue(sexCB).isEmpty();
                AutoCompleteComboBox.getComboBoxValue(countryCB).isEmpty();
                okButtonClicked = true;
                Stage            currentStage            =            (Stage)
continentTF.getScene().getWindow();
                currentStage.close();
            } catch (Exception e) {
                showAlert();
            }
        }
    }

    public static void showAlert() {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error Dialog");
        alert.setHeaderText(null);
        alert.setContentText("Please Fill in all Fields!");
        alert.showAndWait();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        initializingContents(continentTF,   sexCB,   countries,   countryCB,
continents);
    }

    static void initializingContents(TextField continentTF, ComboBox<String>
sexCB, Map<String, String> countries, ComboBox<String> countryCB, Map<String,
String> continents) {
        continentTF.setEditable(false);
        sexCB.getItems().addAll("male", "female");
        Locale.setDefault(Locale.US);
        for (String countryCode : Locale.getISOCountries()) {
            Locale locale = new Locale("", countryCode);
            countries.put(locale.getDisplayCountry(),
countryCode.toUpperCase());
            countryCB.getItems().add(locale.getDisplayCountry());
        }
        AutoCompleteComboBox.setAutoComplete(countryCB,          (typedText,
itemToCompare)                                                             ->
itemToCompare.toLowerCase().contains(typedText.toLowerCase())            ||
itemToCompare.equals(typedText));
        continents.put("AS", "Asia");
        continents.put("EU", "Europe");
        continents.put("NA", "North America");
        continents.put("AF", "Africa");
        continents.put("AN", "Antarctica");
        continents.put("SA", "South America");
        continents.put("OC", "Oceania");
        countryCB.setOnHidden(event -> {
            try {
                String countryCode = countries.get(countryCB.getValue());
                InputStream                        in                        =
ClassLoader.getSystemClassLoader().getResourceAsStream("country_continent.c
sv");
                InputStreamReader isr = new InputStreamReader(in);
                BufferedReader br = new BufferedReader(isr);
                while (br.ready()) {
                    String[] line = br.readLine().split(",");
                    if (line[0].equals(countryCode)) {
                        continentTF.setText(continents.get(line[1]));
                        break;
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }
}
```

```
package controllers.configurations;

import functional.AutoCompleteComboBox;
import functional.Student;
import controllers.main.MainController;
import controllers.newFile.StudentConfig2Controller;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.ComboBoxTableCell;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.input.KeyEvent;
import javafx.stage.Modality;
import javafx.stage.Stage;

import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.*;

import                                                              static
controllers.configurations.Year2StudentController.searchGivenName;

public class Year1StudentController implements Initializable {

    private Map<String,String> countries = new HashMap<>();
    private Map<String,String> continents = new HashMap<>();

    private List<Student> editedStudents = new ArrayList<>();

    private      ObservableList<Student>      studentObservableList      =
FXCollections.observableArrayList();
    private List<Integer> deletedStudentsIds = new ArrayList<>();
    private List<Student> addedStudents = new ArrayList<>();
    @FXML
    private Button checkButton;

    @FXML
    private Button okButton;

    @FXML
    private Button cancelButton;

    @FXML
    private TableView<Student> studentTableView;

    @FXML
    private TableColumn<Student, String> givenNameColumn;

    @FXML
    private TableColumn<Student, String> familyNameColumn;

    @FXML
    private TableColumn<Student, String> sexColumn;

    @FXML
    private TableColumn<Student, String> countryColumn;

    @FXML
    private TableColumn<Student, String> continentColumn;

    @FXML
    private TableColumn<Student, CheckBox> allocatedColumn;

    @FXML
    private Button addButton;

    @FXML
    private Button deleteButton;

    @FXML
    private TextField searchTextField;

    @FXML
    void searchTyped(KeyEvent event) {
        searchGivenName(searchTextField,                    studentTableView,
studentObservableList);
    }
    @FXML
    void checkClicked(ActionEvent event) {
        Student                         selectedStudent                          =
studentTableView.getSelectionModel().getSelectedItem();
        checkClickedContents(selectedStudent);
    }

    static void checkClickedContents(Student selectedStudent) {
        if (selectedStudent.getAllocated().isSelected()) {
            String room = "";
            String building = "";
            try {
                int studentId = selectedStudent.getId();
                Statement stmt = MainController.c.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
                ResultSetMetaData rsmd = rs.getMetaData();
                int columNum = rsmd.getColumnCount();
                while (rs.next()) {
                    for (int i = 6; i <= columNum; i++) {
                        if (rs.getInt(i) == studentId) {
                            room = rs.getString(2);
                            building = rs.getString(3);
                            break;
                        }
                    }
                    if (!room.equals("") && !building.equals("")) break;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Information Dialog");
            alert.setHeaderText(null);
            alert.setContentText("The    selected    student    is    allocated
in...\nBuilding: " + building + "\nRoom: " + room);
            alert.showAndWait();
        } else {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Information Dialog");
            alert.setHeaderText(null);
            alert.setContentText("The selected student is not allocated in any
room!");
            alert.showAndWait();
        }
    }

    @FXML
    void addClicked(ActionEvent event) throws IOException {
        Stage stage = new Stage();
        FXMLLoader                loader                 =                 new
FXMLLoader(getClass().getResource("/fxmls/configurations/Year1StudentAddCli
cked.fxml"));
        stage.setScene(new Scene(loader.load()));
        Year1StudentAddClickedController controller = loader.getController();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setOnHiding(event1 -> {
            if (controller.isOkButtonClicked()) {
                Student addedStudent = new Student();

addedStudent.setGivenName(controller.getGivenNameTF().getText());

addedStudent.setFamilyName(controller.getFamilyNameTF().getText());
                addedStudent.setSex(controller.getSexCB().getValue());

addedStudent.setCountryValue(controller.getCountryCB().getValue());

addedStudent.setContinent(controller.getContinentTF().getText());
                addedStudent.setYear(1);
                studentObservableList.add(addedStudent);
                addedStudents.add(addedStudent);
                searchGivenName(searchTextField,            studentTableView,
studentObservableList);
            }
        });
        stage.showAndWait();
    }

    @FXML
    void cancelClick(ActionEvent event) {
        cancelClickContents(studentTableView);
    }

    static void cancelClickContents(TableView<Student> studentTableView) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText("Confirmation Dialog");
        alert.setContentText("By clicking OK, you will lose all changes you
just made!\n" +
                "Are you sure to continue?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            Stage stage = (Stage) studentTableView.getScene().getWindow();
            stage.close();
        }
    }

    @FXML
    void deleteClicked(ActionEvent event) {
        Student                        studentToDelete                        =
studentTableView.getSelectionModel().getSelectedItem();
        deleteClickedContents(studentToDelete,                 addedStudents,
studentObservableList, studentTableView, deletedStudentsIds);
        searchGivenName(searchTextField,                   studentTableView,
studentObservableList);
    }

    static void deleteClickedContents(Student studentToDelete, List<Student>
addedStudents,        ObservableList<Student>        studentObservableList,
TableView<Student> studentTableView, List<Integer> deletedStudentsIds) {
        if (addedStudents.contains(studentToDelete)) {
            addedStudents.remove(studentToDelete);
            studentObservableList.remove(studentToDelete);
            studentTableView.refresh();
        } else {
            deletedStudentsIds.add(studentToDelete.getId());
            studentObservableList.remove(studentToDelete);
            studentTableView.refresh();
        }
    }

    @FXML
    void okClick(ActionEvent event) {
        updateStudents(editedStudents);
        deleteStudents(deletedStudentsIds);
        addStudents(addedStudents);
        Stage stage = (Stage) studentTableView.getScene().getWindow();
        stage.close();
    }

    public static void updateStudents(List<Student> editedStudents) {
        for (Student student: editedStudents) {
            try {
                Statement stmt = MainController.c.createStatement();
                stmt.executeUpdate("UPDATE Students SET " +
                        "GivenName = \"" + student.getGivenName() + "\", " +
                        "FamilyName = \"" + student.getFamilyName() + "\", " +
                        "Sex = \"" + student.getSex() + "\", " +
                        "Country              =              \""              +
AutoCompleteComboBox.getComboBoxValue(student.getCountryCB()) + "\", " +
                        "Continent = \"" + student.getContinent() + "\" WHERE
" +
                        "Id = " + student.getId() + ";");
                MainController.c.commit();
                stmt.close();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void deleteStudents(List<Integer> deletedStudentIds) {
        for (Integer studentId : deletedStudentIds) {
            try {
                Statement stmt = MainController.c.createStatement();
                ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
                ResultSetMetaData rsmd = rs1.getMetaData();
                int numberOfColumns = rsmd.getColumnCount();
                int roomId = 0;
                int columnNum = 0;
                while (rs1.next()) {
                    for (int i = 6; i <= numberOfColumns; i++) {
                        if (rs1.getInt(i) == studentId) {
                            roomId = rs1.getInt(1);
                            columnNum = i;
                            break;
                        }
                    }
                    if (columnNum != 0) break;
                }
                rs1.close();
                if (roomId != 0) {
                    stmt.executeUpdate("UPDATE   Rooms   SET   'Student   "   +
(columnNum - 5) + "' = NULL WHERE Id = " + roomId + ";");
                    MainController.c.commit();
                }
                stmt.executeUpdate("DELETE  FROM  Students  WHERE  Id  =  "  +
studentId + ";");
                MainController.c.commit();
                stmt.close();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void addStudents(List<Student> addedStudents) {
        try {
            Statement stmt = MainController.c.createStatement();
            for (Student addedStudent: addedStudents) {
                stmt.executeUpdate("INSERT    INTO    Students    (GivenName,
FamilyName, Sex, Country, Continent, 'Year') VALUES " +
                        "('" + addedStudent.getGivenName()
                        + "', '" + addedStudent.getFamilyName()
                        + "', '" + addedStudent.getSex()
                        +                "',                '"                +
AutoCompleteComboBox.getComboBoxValue(addedStudent.getCountryCB())
                        + "', '" + addedStudent.getContinent()
                        + "', " + addedStudent.getYear() + ")");
                MainController.c.commit();
            }
            stmt.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }



    @Override
    public void initialize(URL location, ResourceBundle resources) {
        initializingContents(1,  studentTableView,  deleteButton,  countries,
continents,   sexColumn,   countryColumn,   continentColumn,   givenNameColumn,
familyNameColumn,  allocatedColumn,  studentObservableList,  editedStudents,
checkButton);
    }

    static    void    initializingContents(int    year,    TableView<Student>
studentTableView,  Button  deleteButton,  Map<String,  String>  countries,
Map<String,  String>  continents,  TableColumn<Student,  String>  sexColumn,
TableColumn<Student,  String>  countryColumn,  TableColumn<Student,  String>
continentColumn,     TableColumn<Student,     String>     givenNameColumn,
TableColumn<Student, String> familyNameColumn, TableColumn<Student, CheckBox>
allocatedColumn, ObservableList<Student> studentObservableList, List<Student>
editedStudents, Button checkButton) {
        studentTableView.setEditable(true);
        deleteButton.setDisable(true);
        checkButton.setDisable(true);
        Locale.setDefault(Locale.US);
        for (String countryCode : Locale.getISOCountries()) {
            Locale locale = new Locale("", countryCode);
            countries.put(locale.getDisplayCountry(),
countryCode.toUpperCase());
        }
        continents.put("AS", "Asia");
        continents.put("EU", "Europe");
        continents.put("NA", "North America");
        continents.put("AF", "Africa");
        continents.put("AN", "Antarctica");
        continents.put("SA", "South America");
        continents.put("OC", "Oceania");
        sexColumn.setStyle("-fx-alignment: CENTER;");
        countryColumn.setStyle("-fx-alignment: CENTER;");
        continentColumn.setStyle("-fx-alignment: CENTER;");
        allocatedColumn.setStyle("-fx-alignment: CENTER;");
        givenNameColumn.setCellValueFactory(new
PropertyValueFactory<>("givenName"));
        givenNameColumn.setCellFactory(TextFieldTableCell.forTableColumn());
        familyNameColumn.setCellValueFactory(new
PropertyValueFactory<>("familyName"));
        familyNameColumn.setCellFactory(TextFieldTableCell.forTableColumn());
        sexColumn.setCellValueFactory(new PropertyValueFactory<>("sex"));
        sexColumn.setCellFactory(ComboBoxTableCell.forTableColumn("male",
"female"));
        countryColumn.setCellValueFactory(new
PropertyValueFactory<>("countryCB"));
        continentColumn.setCellValueFactory(new
PropertyValueFactory<>("continent"));
        allocatedColumn.setCellValueFactory(new
PropertyValueFactory<>("allocated"));
        populateTableView(year, studentObservableList);
        for (Student student: studentObservableList) {
            student.getCountryCB().setOnHidden(e -> {
                StudentConfig2Controller.showContinent(student,    countries,
continents, studentTableView);
                editedStudents.add(student);
            });
        }
        studentTableView.setItems(studentObservableList);

studentTableView.getSelectionModel().selectedItemProperty().addListener((v,
oldValue, newValue) -> {
            deleteButton.setDisable(false);
            checkButton.setDisable(false);
        } );
    }

    public  static  void  populateTableView(int  year,  ObservableList<Student>
studentObservableList) {
        try {
            studentObservableList.clear();
            Statement stmt = MainController.c.createStatement();
            ResultSet  rs  =  stmt.executeQuery("SELECT  *  FROM  Students  WHERE
Year = " + year + ";");
            while (rs.next()) {
                Student student = new Student();
                student.setId(rs.getInt("Id"));
                student.setGivenName(rs.getString("GivenName"));
                student.setFamilyName(rs.getString("FamilyName"));
                student.setSex(rs.getString("Sex"));
                student.setCountryValue(rs.getString("Country"));
                student.setContinent(rs.getString("Continent"));
                if (student.isAllocated(rs.getInt("Id"))) {
                    student.getAllocated().setSelected(true);
                } else {
                    student.getAllocated().setSelected(false);
                }
                studentObservableList.add(student);
            }
            stmt.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @FXML
    void changeContinent(TableColumn.CellEditEvent editedCell) {
        Student                         editingStudent                        =
studentTableView.getSelectionModel().getSelectedItem();
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {
            editingStudent.setContinent(editedCell.getNewValue().toString());
            addToEditedStudents(editingStudent,              editedStudents,
addedStudents);
        }
    }

    @FXML
    void changeFamilyName(TableColumn.CellEditEvent editedCell) {
        Student                         editingStudent                        =
studentTableView.getSelectionModel().getSelectedItem();
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {

editingStudent.setFamilyName(editedCell.getNewValue().toString());
            addToEditedStudents(editingStudent,              editedStudents,
addedStudents);
        }
    }

    @FXML
    void changeGivenName(TableColumn.CellEditEvent editedCell) {
        Student                         editingStudent                        =
studentTableView.getSelectionModel().getSelectedItem();
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {
            editingStudent.setGivenName(editedCell.getNewValue().toString());
            addToEditedStudents(editingStudent,              editedStudents,
addedStudents);
        }
    }

    @FXML
    void changeSex(TableColumn.CellEditEvent editedCell) {
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Dialog");
            alert.setHeaderText("Please Confirm...");
            alert.setContentText("Changing the student's sex will cause the
student to be deleted from current room.\n Are you sure to proceed?");
            Optional<ButtonType> result = alert.showAndWait();
            Student                        editingStudent                    =
studentTableView.getSelectionModel().getSelectedItem();
            if (result.get() == ButtonType.OK) {
                String newSex = editedCell.getNewValue().toString();
                editingStudent.setSex(newSex);
                int studentId = editingStudent.getId();
                try {
                    Statement stmt = MainController.c.createStatement();
                    ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
                    ResultSetMetaData rsmd = rs.getMetaData();
                    int columnNum = rsmd.getColumnCount();
                    int a = 0;
                    while (rs.next()) {
                        for (int i = 6; i <= columnNum; i++) {
                            if (rs.getInt(i) == studentId) {
                                stmt.executeUpdate("UPDATE Rooms SET \"Student
" + (i-5) + "\" = NULL WHERE \"Room No./Name\" = \"" + rs.getString(2) + "\"
AND \"Building No./Name\" = \"" + rs.getString(3) + "\";");
                                MainController.c.commit();
                                a = 1;
                                break;
                            }
                        }
                        if (a == 1) break;
                    }
                    stmt.executeUpdate("UPDATE Students SET Sex = \"" + newSex
+ "\" WHERE Id = " + studentId + ";");
                    MainController.c.commit();
                    populateTableView(1, studentObservableList);
                    studentTableView.refresh();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                editingStudent.setSex(editedCell.getOldValue().toString());
                studentTableView.refresh();
            }
```

```
        }
    }

    public  static  void  addToEditedStudents(Student  student,  List<Student>
editedStudents, List<Student> addedStudents) {
        if                                   (!editedStudents.contains(student)
&& !addedStudents.contains(student)) {
            editedStudents.add(student);
        }
    }
}
```

```
package controllers.configurations;

import functional.AutoCompleteComboBox;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;
import java.util.ResourceBundle;
import                                                               static
controllers.configurations.Year1StudentAddClickedController.showAlert;

public class Year2StudentAddClickedController implements Initializable {
    public boolean isOkButtonClicked() {
        return okButtonClicked;
    }

    private boolean okButtonClicked = false;

    private Map<String,String> countries = new HashMap<>();

    public TextField getFamilyNameTF() {
        return familyNameTF;
    }

    public TextField getGivenNameTF() {
        return givenNameTF;
    }

    public ComboBox<String> getSexCB() {
        return sexCB;
    }

    public ComboBox<String> getCountryCB() {
        return countryCB;
    }

    public TextField getContinentTF() {
        return continentTF;
    }
    private Map<String,String> continents = new HashMap<>();

    @FXML
    private TextField familyNameTF;

    @FXML
    private TextField givenNameTF;

    @FXML
    private ComboBox<String> sexCB;

    @FXML
    private ComboBox<String> countryCB;

    @FXML
    private TextField continentTF;

    @FXML
    void okClicked(ActionEvent event) {
        if                (givenNameTF.getText().isEmpty()                 ||
familyNameTF.getText().isEmpty()) {
            showAlert();
        } else {
            try {
                sexCB.getValue().isEmpty();
                AutoCompleteComboBox.getComboBoxValue(sexCB).isEmpty();
                AutoCompleteComboBox.getComboBoxValue(countryCB).isEmpty();
                okButtonClicked = true;
                Stage            currentStage            =            (Stage)
continentTF.getScene().getWindow();
                currentStage.close();
            } catch (Exception e) {
                showAlert();
            }
        }
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        Year1StudentAddClickedController.initializingContents(continentTF,
sexCB, countries, countryCB, continents);
    }
}
```

```
package controllers.configurations;

import functional.Student;
import controllers.main.MainController;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.KeyEvent;
import javafx.stage.Modality;
import javafx.stage.Stage;
import java.io.IOException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.*;

import static controllers.configurations.Year1StudentController.*;

public class Year2StudentController implements Initializable {

    private Map<String,String> countries = new HashMap<>();
    private Map<String,String> continents = new HashMap<>();

    private List<Student> editedStudents = new ArrayList<>();

    private       ObservableList<Student>       studentObservableList       =
FXCollections.observableArrayList();
    private List<Integer> deletedStudentsIds = new ArrayList<>();
    private List<Student> addedStudents = new ArrayList<>();

    private List<String> givenNames = new ArrayList<>();



    @FXML
    private Button checkButton;

    @FXML
    private Button okButton;
    @FXML
    private Button cancelButton;

    @FXML
    private TableView<Student> studentTableView;

    @FXML
    private TableColumn<Student, String> givenNameColumn;

    @FXML
    private TableColumn<Student, String> familyNameColumn;

    @FXML
    private TableColumn<Student, String> sexColumn;

    @FXML
    private TableColumn<Student, String> countryColumn;

    @FXML
    private TableColumn<Student, String> continentColumn;

    @FXML
    private TableColumn<Student, CheckBox> allocatedColumn;

    @FXML
    private Button addButton;

    @FXML
    private Button deleteButton;

    @FXML
    private TextField searchTextField;

    @FXML
    void searchTyped(KeyEvent event) {
        searchGivenName(searchTextField,                    studentTableView,
studentObservableList);
    }

    public    static    void    searchGivenName(TextField    searchTextField,
TableView<Student>         studentTableView,         ObservableList<Student>
studentObservableList) {
        if (searchTextField.getText().isEmpty()) {
            studentTableView.setItems(studentObservableList);
            studentTableView.refresh();
        } else {
            ObservableList<Student>                  newList                    =
FXCollections.observableArrayList();
            for (Student student: studentObservableList) {
                if
(student.getGivenName().toLowerCase().contains(searchTextField.getText().to
LowerCase())) newList.add(student);
            }
            studentTableView.setItems(newList);
            studentTableView.refresh();
        }
    }

    @FXML
    void checkClicked(ActionEvent event) {
        Student                         selectedStudent                        =
studentTableView.getSelectionModel().getSelectedItem();
        Year1StudentController.checkClickedContents(selectedStudent);
    }

    @FXML
    void addClicked(ActionEvent event) throws IOException {
        Stage stage = new Stage();
        FXMLLoader                 loader                 =                 new
FXMLLoader(getClass().getResource("/fxmls/configurations/Year2StudentAddCli
cked.fxml"));
        stage.setScene(new Scene(loader.load()));
        Year2StudentAddClickedController controller = loader.getController();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setOnHiding(event1 -> {
            if (controller.isOkButtonClicked()) {
                Student addedStudent = new Student();

addedStudent.setGivenName(controller.getGivenNameTF().getText());

addedStudent.setFamilyName(controller.getFamilyNameTF().getText());
                addedStudent.setSex(controller.getSexCB().getValue());

addedStudent.setCountryValue(controller.getCountryCB().getValue());

addedStudent.setContinent(controller.getContinentTF().getText());
                addedStudent.setYear(2);
                studentObservableList.add(addedStudent);
                addedStudents.add(addedStudent);
                searchGivenName(searchTextField,            studentTableView,
studentObservableList);
            }
        });
        stage.showAndWait();
    }

    @FXML
    void cancelClick(ActionEvent event) {
        Year1StudentController.cancelClickContents(studentTableView);
    }

    @FXML
    void deleteClicked(ActionEvent event) {
        Student                        studentToDelete                        =
studentTableView.getSelectionModel().getSelectedItem();
        Year1StudentController.deleteClickedContents(studentToDelete,
addedStudents, studentObservableList, studentTableView, deletedStudentsIds);
        searchGivenName(searchTextField,                    studentTableView,
studentObservableList);
    }


    @FXML
    void okClick(ActionEvent event) {
        updateStudents(editedStudents);
        deleteStudents(deletedStudentsIds);
        addStudents(addedStudents);
        Stage stage = (Stage) studentTableView.getScene().getWindow();
        stage.close();
    }

    @FXML
    void changeContinent(TableColumn.CellEditEvent editedCell) {
        Student                        editingStudent                        =
studentTableView.getSelectionModel().getSelectedItem();
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {
            editingStudent.setContinent(editedCell.getNewValue().toString());
            addToEditedStudents(editingStudent,              editedStudents,
addedStudents);
```

```
        }
    }

    @FXML
    void changeFamilyName(TableColumn.CellEditEvent editedCell) {
        Student                         editingStudent                         =
studentTableView.getSelectionModel().getSelectedItem();
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {

editingStudent.setFamilyName(editedCell.getNewValue().toString());
            addToEditedStudents(editingStudent,              editedStudents,
addedStudents);
        }
    }

    @FXML
    void changeGivenName(TableColumn.CellEditEvent editedCell) {
        Student                         editingStudent                         =
studentTableView.getSelectionModel().getSelectedItem();
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {
            editingStudent.setGivenName(editedCell.getNewValue().toString());
            addToEditedStudents(editingStudent,              editedStudents,
addedStudents);
        }
    }

    @FXML
    void changeSex(TableColumn.CellEditEvent editedCell) {
        if
(!editedCell.getNewValue().toString().equals(editedCell.getOldValue().toStr
ing())) {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Dialog");
            alert.setHeaderText("Please Confirm...");
            alert.setContentText("Changing the student's sex will cause the
student to be deleted from current room.\n Are you sure to proceed?");
            Optional<ButtonType> result = alert.showAndWait();
            Student                         editingStudent                      =
studentTableView.getSelectionModel().getSelectedItem();
            if (result.get() == ButtonType.OK) {
                String newSex = editedCell.getNewValue().toString();
                editingStudent.setSex(newSex);
                int studentId = editingStudent.getId();
                try {
                    Statement stmt = MainController.c.createStatement();
                    ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
                    ResultSetMetaData rsmd = rs.getMetaData();
                    int columnNum = rsmd.getColumnCount();
                    int a = 0;
                    while (rs.next()) {
                        for (int i = 6; i <= columnNum; i++) {
                            if (rs.getInt(i) == studentId) {
                                stmt.executeUpdate("UPDATE Rooms SET \"Student
" + (i-5) + "\" = NULL WHERE \"Room No./Name\" = \"" + rs.getString(2) + "\"
AND \"Building No./Name\" = \"" + rs.getString(3) + "\";");
                                MainController.c.commit();
                                a = 1;
                                break;
                            }
                        }
                        if (a == 1) break;
                    }
                    stmt.executeUpdate("UPDATE Students SET Sex = \"" + newSex
+ "\" WHERE Id = " + studentId + ";");
                    MainController.c.commit();
                    populateTableView(2, studentObservableList);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                editingStudent.setSex(editedCell.getOldValue().toString());
                studentTableView.refresh();
            }
        }
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        Year1StudentController.initializingContents(2,     studentTableView,
deleteButton,    countries,    continents,    sexColumn,    countryColumn,
continentColumn,    givenNameColumn,    familyNameColumn,    allocatedColumn,
studentObservableList, editedStudents, checkButton);
    }
}
```

```
package controllers.login;
import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXPasswordField;
import com.jfoenix.controls.JFXTextField;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;

public class LoginController {

    @FXML
    private JFXTextField usernameInput;
    @FXML
    private JFXPasswordField passwordInput;
    @FXML
    private JFXButton cancelButton;
    @FXML
    private Label wrongPasswordMessage;
    @FXML
    void cancel(ActionEvent event) {
        Stage currentStage = (Stage)cancelButton.getScene().getWindow();
        currentStage.close();
    }
    @FXML
    void login(ActionEvent event) throws Exception{
        if            (usernameInput.getText().equals("MUWCI")            &&
passwordInput.getText().equals("muwci2018")) {
            Parent                        mainScreen                        =
FXMLLoader.load(getClass().getResource("/fxmls/main/Main.fxml"));
            Stage currentStage = (Stage)cancelButton.getScene().getWindow();
            currentStage.setScene(new Scene(mainScreen));
            currentStage.setMaximized(true);
            currentStage.setResizable(true);
            currentStage.show();
        } else {
            wrongPasswordMessage.visibleProperty().set(true);
            passwordInput.clear();
        }
    }
}
```

```
package controllers.main;

import functional.HandleButton;
import functional.StudentString;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

import java.net.URL;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

public class AddYear1StudentController implements Initializable {

    private     ObservableList<StudentString>     studentObservableList     =
FXCollections.observableArrayList();

    public void setSelectedItem(StudentString selectedItem) {
        this.selectedItem = selectedItem;
    }

    private StudentString selectedItem;

    public void setBoyGirl(int boyGirl) {
        this.boyGirl = boyGirl;
    }

    private int boyGirl; // boy = 0, girl = 1;

    @FXML
    private Button okButton;

    @FXML
    private Button cancelButton;

    @FXML
    private TableView<StudentString> studentTableView;

    @FXML
    private TableColumn<StudentString, String> givenNameColumn;

    @FXML
    private TableColumn<StudentString, String> familyNameColumn;

    @FXML
    private TableColumn<StudentString, String> sexColumn;

    @FXML
    private TableColumn<StudentString, String> countryColumn;

    @FXML
    private TableColumn<StudentString, String> continentColumn;

    @FXML
    void keyTyped(KeyEvent event) {
        selectedItem                                                    =
studentTableView.getSelectionModel().getSelectedItem();
    }

    @FXML
    void mouseClicked(MouseEvent event) {
        selectedItem                                                    =
studentTableView.getSelectionModel().getSelectedItem();
    }

    @FXML
    void cancelClicked(ActionEvent event) {
        selectedItem = null;
        HandleButton button = new HandleButton();
        button.handleCancelButton(cancelButton);
    }

    @FXML
    StudentString okClick(ActionEvent event) {
        Stage currentStage = (Stage) okButton.getScene().getWindow();
        currentStage.close();
        return selectedItem;
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {

        initializingContents(okButton,      studentTableView,      sexColumn,
countryColumn, continentColumn, givenNameColumn, familyNameColumn);
    }

    static         void         initializingContents(Button         okButton,
TableView<StudentString> studentTableView, TableColumn<StudentString, String>
sexColumn,      TableColumn<StudentString,      String>      countryColumn,
TableColumn<StudentString,             String>             continentColumn,
TableColumn<StudentString,             String>             givenNameColumn,
TableColumn<StudentString, String> familyNameColumn) {
        okButton.setDisable(true);
        studentTableView.getSelectionModel()
                .selectedItemProperty()
                .addListener((observable, oldValue, newValue) -> {
                    okButton.setDisable(false);
                });
        sexColumn.setStyle("-fx-alignment: CENTER;");
        countryColumn.setStyle("-fx-alignment: CENTER;");
        continentColumn.setStyle("-fx-alignment: CENTER;");
        givenNameColumn.setCellValueFactory(new
PropertyValueFactory<>("givenName"));
        familyNameColumn.setCellValueFactory(new
PropertyValueFactory<>("familyName"));
        sexColumn.setCellValueFactory(new PropertyValueFactory<>("sex"));
        countryColumn.setCellValueFactory(new
PropertyValueFactory<>("country"));
        continentColumn.setCellValueFactory(new
PropertyValueFactory<>("continent"));
    }

    public void populateTableView() {
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs1.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            List<Integer> allocatedStudentIds = new ArrayList<>();
            while (rs1.next()) {
                for (int i = 6; i <= numberOfColumns; i++) {
                    int studentId = rs1.getInt(i);
                    if (studentId != 0) {
                        allocatedStudentIds.add(studentId);
                    }
                }
            }
            ResultSet rs;
            if (boyGirl == 0) {
                rs = stmt.executeQuery("SELECT * FROM Students Where Year = "
+ 1 + " AND \"Sex\" = \"male\";");
            } else {
                rs = stmt.executeQuery("SELECT * FROM Students Where Year = "
+ 1 + " AND \"Sex\" = \"female\";");
            }
            addStudentToList(rs, studentObservableList, allocatedStudentIds);
            stmt.close();

            studentTableView.setItems(studentObservableList);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void addStudentToList(ResultSet rs,
ObservableList<StudentString> studentObservableList, List<Integer>
allocatedStudentIds) throws SQLException {
        while (rs.next()) {
            Integer studentId = rs.getInt("Id");
            if (!allocatedStudentIds.contains(studentId)) {
                StudentString student = new StudentString();
                student.setId(studentId);
                student.setGivenName(rs.getString("GivenName"));
                student.setFamilyName(rs.getString("FamilyName"));
                student.setSex(rs.getString("Sex"));
                student.setCountry(rs.getString("Country"));
                student.setContinent(rs.getString("Continent"));
                student.setYear(rs.getInt("Year"));
                studentObservableList.add(student);
            }
        }
    }
}
```

```
package controllers.main;

import functional.HandleButton;
import functional.StudentString;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

public class AddYear2StudentController implements Initializable {

    private     ObservableList<StudentString>     studentObservableList     =
FXCollections.observableArrayList();

    public void setSelectedItem(StudentString selectedItem) {
        this.selectedItem = selectedItem;
    }
    private int boyGirl; // boy = 0, girl = 1;

    public void setBoyGirl(int boyGirl) {
        this.boyGirl = boyGirl;
    }

    private StudentString selectedItem;

    @FXML
    private Button okButton;

    @FXML
    private Button cancelButton;
    @FXML
    private TableView<StudentString> studentTableView;

    @FXML
    private TableColumn<StudentString, String> givenNameColumn;

    @FXML
    private TableColumn<StudentString, String> familyNameColumn;

    @FXML
    private TableColumn<StudentString, String> sexColumn;

    @FXML
    private TableColumn<StudentString, String> countryColumn;

    @FXML
    private TableColumn<StudentString, String> continentColumn;

    @FXML
    void cancelClick(ActionEvent event) {
        selectedItem = null;
        HandleButton button = new HandleButton();
        button.handleCancelButton(cancelButton);
    }

    @FXML
    void keyTyped(KeyEvent event) {
        selectedItem                                                    =
studentTableView.getSelectionModel().getSelectedItem();
    }

    @FXML
    void mouseClicked(MouseEvent event) {
        selectedItem                                                    =
studentTableView.getSelectionModel().getSelectedItem();
    }

    @FXML
    StudentString okClick(ActionEvent event) {
        Stage currentStage = (Stage) okButton.getScene().getWindow();
        currentStage.close();
        return selectedItem;
    }
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        AddYear1StudentController.initializingContents(okButton,
studentTableView, sexColumn, countryColumn, continentColumn, givenNameColumn,
familyNameColumn);
    }

    public void populateTableView() {
        try {


            Statement stmt = MainController.c.createStatement();

            ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs1.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            List<Integer> allocatedStudentIds = new ArrayList<>();
            while (rs1.next()) {
                for (int i = 6; i <= numberOfColumns; i++) {
                    int studentId = rs1.getInt(i);
                    if (studentId != 0) {
                        allocatedStudentIds.add(studentId);
                    }
                }
            }
            ResultSet rs;
            if (boyGirl == 0) {
                rs = stmt.executeQuery("SELECT * FROM Students Where Year = "
+ 2 + " AND \"Sex\" = \"male\";");
            } else {
                rs = stmt.executeQuery("SELECT * FROM Students Where Year = "
+ 2 + " AND \"Sex\" = \"female\";");
            }
            AddYear1StudentController.addStudentToList(rs,
studentObservableList,allocatedStudentIds);
            stmt.close();

            studentTableView.setItems(studentObservableList);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
package controllers.main;

import GA.Population;
import com.jfoenix.controls.JFXTreeView;
import com.sun.tools.javac.Main;
import functional.Room;
import functional.StudentString;
import javafx.application.Platform;
import javafx.beans.binding.Bindings;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.concurrent.Service;
import javafx.concurrent.Task;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.stage.DirectoryChooser;
import javafx.stage.FileChooser;
import javafx.stage.Modality;
import javafx.stage.Stage;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.*;
import java.net.URL;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.ResourceBundle;

public class MainController implements Initializable {

    public static Connection c;
    ObservableList<StudentString>                 students                 =
FXCollections.observableArrayList();

    ObservableList<TreeItem<String>>          buildingsTreeItems          =
FXCollections.observableArrayList();

    TreeItem<String> root;

    public static String fileName;
    public static String directory;

    public boolean finishedAllocation;

    @FXML
    private TextField boyGirlTextField;

    @FXML
    private TextField roomCapacityTextField;

    @FXML
    private TextField bedsAvailableTextField;

    @FXML
    private MenuItem fileNew;

    @FXML
    private MenuItem fileOpen;

    @FXML
    private MenuItem fileQuit;

    @FXML
    private TableView<StudentString> roomTableView;

    @FXML
    private TableColumn<StudentString, String> givenNameColumn;

    @FXML
    private TableColumn<StudentString, String> familyNameColumn;

    @FXML
    private TableColumn<StudentString, String> sexColumn;
    @FXML
    private TableColumn<StudentString, String> countryColumn;

    @FXML
    private TableColumn<StudentString, String> continentColumn;

    @FXML
    private TableColumn<StudentString, String> yearColumn;

    ContextMenu contextMenu = new ContextMenu();
    Menu addMenu = new Menu("Add");
    MenuItem addYear1Student = new MenuItem("Year 1 Student...");
    MenuItem addYear2Student = new MenuItem("Year 2 Student...");
    MenuItem removeMenuItem = new MenuItem("Remove");
    MenuItem moveMenuItem = new MenuItem("Move to...");
    MenuItem switchMenuItem = new MenuItem("Switch with...");

    @FXML
    private JFXTreeView<String> treeView = new JFXTreeView<>();

    List<String> buildingNames = new ArrayList<>();

    @FXML
    void fileNewClicked(ActionEvent event) throws IOException {
        Parent                          directoryLayout                          =
FXMLLoader.load(getClass().getResource("/fxmls/newFile/DirectoryView.fxml")
);
        Stage fileNewStage = new Stage();
        fileNewStage.setScene(new Scene(directoryLayout));
        fileNewStage.setTitle("newFile Room Allocation");
        fileNewStage.setResizable(false);
        fileNewStage.initModality(Modality.APPLICATION_MODAL);
        fileNewStage.centerOnScreen();
        fileNewStage.showAndWait();
        showTreeView();
    }

    @FXML
    void fileOpenClicked(ActionEvent event) throws SQLException{
        Stage currentStage = (Stage) treeView.getScene().getWindow();
        final FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().addAll(new
FileChooser.ExtensionFilter("sqlite Files", "*.sqlite"));
        File selectedFile = fileChooser.showOpenDialog(currentStage);
        if (selectedFile != null) {
            MainController.fileName = selectedFile.getName();
            MainController.directory                                        =
selectedFile.getAbsolutePath().replace("\\", "/");
            MainController.directory = MainController.directory.substring(0,
MainController.directory.length() - MainController.fileName.length() - 1);
            MainController.fileName  =  MainController.fileName.substring(0,
MainController.fileName.length() - 7);
            connectToDB();
            roomTableView.setItems(null);
            roomCapacityTextField.setText("");
            bedsAvailableTextField.setText("");
            boyGirlTextField.setText("");
        }
        showTreeView();
    }

    @FXML
    void fileQuitClicked(ActionEvent event) {
        writeDirectoryFile();
        System.exit(0);
    }
    @FXML
    void addOrDeleteClicked(ActionEvent event) {
        openWindow("/fxmls/configurations/AddOrDeleteRoom.fxml",        "Room
Configuration");
        showTreeView();
    }

    @FXML
    void updateRoomInfoClicked(ActionEvent event) {
        openWindow("/fxmls/configurations/UpdateRoomInfo.fxml",         "Room
Configuration");
        showTreeView();
    }

    @FXML
    void year1ConfigClicked(ActionEvent event) {
        openWindow("/fxmls/configurations/Year1Student.fxml",     "Year     1
Student Configuration");
        showTreeView();
    }

    @FXML
    void year2ConfigClicked(ActionEvent event) {
        openWindow("/fxmls/configurations/Year2Student.fxml",     "Year     2
Student Configuration");
        showTreeView();
    }

    @FXML
    void clearYear1AllocationClicked(ActionEvent event) {
        clearAllocation(1);
    }

    @FXML
    void clearYear2AllocationClicked(ActionEvent event) {
        clearAllocation(2);
    }


    private void clearAllocation(int year) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText("Please Confirm...");
        alert.setContentText("By clicking OK, all Year " + year + " allocation
information will be deleted.\nAre you sure to proceed?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            try {
                List<Integer> studentIds = new ArrayList<>();
                Statement stmt = MainController.c.createStatement();
                ResultSet  rs1  =  stmt.executeQuery("SELECT  id  FROM  Students
WHERE \"Year\" = " + year + ";");
                while (rs1.next()) {
                    studentIds.add(rs1.getInt(1));
                }
                ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms");
                ResultSetMetaData rsmd = rs.getMetaData();
                int columnNum = rsmd.getColumnCount();
                while (rs.next()) {
                    for (int i = 6; i <= columnNum; i++) {
                        Statement stmt1 = MainController.c.createStatement();
                        if (studentIds.contains(rs.getInt(i))) {
                            stmt1.executeUpdate("UPDATE  Rooms  SET  \"Student  "
+ (i-5) + "\" = NULL WHERE id = " + rs.getInt(1) + ";");
                        }
                        c.commit();
                        stmt1.close();
                    }
                }
                stmt.close();
                Alert alert1 = new Alert(Alert.AlertType.INFORMATION);
                alert1.setTitle("Information Dialog");
                alert1.setHeaderText(null);
                alert1.setContentText("Year " + year + " Students Cleared!");
                alert1.show();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        showTreeView();
    }

    @FXML
    void deleteYear1Clicked(ActionEvent event) {
        deleteStudentClicked(1);
    }

    @FXML
    void deleteYear2Clicked(ActionEvent event) {
        deleteStudentClicked(2);
    }

    private void deleteStudentClicked(int year) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText("Please Confirm...");
        alert.setContentText("By clicking OK, all Year " + year + " students
information will be deleted.\nAre you sure to proceed?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            try {
                List<Integer> studentIds = new ArrayList<>();
                Statement stmt = MainController.c.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM Students WHERE
\"Year\" = " + year + ";");
                while (rs.next()) {
                    studentIds.add(rs.getInt(1));
                }
                rs.close();
                ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
                int columnNum = rs1.getMetaData().getColumnCount();
                int rowId = 1;
                while (rs1.next()) {
                    for (int i = 6; i <= columnNum; i++) {
                        if (studentIds.contains(rs1.getInt(i))) {
                            Statement                  stmt1                  =
MainController.c.createStatement();
                            stmt1.executeUpdate("UPDATE Rooms SET \"Student "
+ (i-5) + "\" = NULL WHERE Id = " + rowId + ";");
                        }
                    }
                    rowId++;
                }
                stmt.executeUpdate("DELETE FROM Students WHERE \"Year\" = " +
year + ";");
                c.commit();
                stmt.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        showTreeView();
    }

    @FXML
    void upgradeClicked(ActionEvent event) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText("Please Confirm...");
        alert.setContentText("By clicking OK, the Year 1 Students will be
transferred to Year 2 students, and All information of current Year 2 students
will be deleted!\n" +
                "Are you sure to proceed?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            try {
                Statement stmt = MainController.c.createStatement();
                ResultSet  rs  =  stmt.executeQuery("SELECT  Id  FROM  Students
WHERE Year = " + 2 + ";");
                List<Integer> year2Ids = new ArrayList<>();
                while (rs.next()) {
                    year2Ids.add(rs.getInt(1));
                }
                for (Integer year2Id: year2Ids) {
                    ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
                    ResultSetMetaData rsmd = rs1.getMetaData();
                    int numberOfColumns = rsmd.getColumnCount();
                    int id = 1;
                    while (rs1.next()) {
                        for (int i = 1; i <= numberOfColumns; i++) {
                            if (year2Ids.contains(rs1.getInt(i))) {
                                Statement stmt1 = c.createStatement();
                                stmt1.executeUpdate("UPDATE Rooms SET \"Student
" + (i-5) + "\" = NULL WHERE Id = " + id + ";");
                                c.commit();
                            }
                        }
                        id++;
                    }
                    stmt.executeUpdate("DELETE FROM Students WHERE Id = " +
year2Id + ";");
                    MainController.c.commit();
                }
                stmt.executeUpdate("UPDATE Students SET Year = 2 WHERE Year =
1;");
                MainController.c.commit();
                stmt.close();
                Alert alert1 = new Alert(Alert.AlertType.INFORMATION);
                alert1.setTitle("Information Dialog");
                alert1.setHeaderText(null);
                alert1.setContentText("Year 1 students are now Year 2
students.\nPlease upload the new Year 1 Students.");
                alert1.showAndWait();
            } catch (Exception e) {
                e.printStackTrace();
            }
            showTreeView();
        }
    }

    @FXML
    void allocateStudentClicked(ActionEvent event) throws IOException {
        if (informationIsNotComplete()) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error Dialog");
            alert.setHeaderText("An Error has Occurred!");
            alert.setContentText("Students' information is not complete!");
            alert.showAndWait();
        } else {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Dialog");
            alert.setHeaderText("Confirmation Dialog");
            alert.setContentText("Please choose the group you wish to allocate.
All the unallocated students in this group will be automatically allocated
to optimize for maximum diversity.");
            ButtonType buttonTypeYear1 = new ButtonType("Year 1");
            ButtonType buttonTypeYear2 = new ButtonType("Year 2");
            ButtonType buttonTypeCancel = new ButtonType("Cancel",
ButtonBar.ButtonData.CANCEL_CLOSE);
            alert.getButtonTypes().setAll(buttonTypeYear1, buttonTypeYear2,
buttonTypeCancel);
            Optional<ButtonType> result = alert.showAndWait();
            if (result.get() == buttonTypeYear1) {
                allocate(1);
            }
            if (result.get() == buttonTypeYear2) {
                allocate(2);
            }
        }
    }

    private void allocate(int year) throws IOException {
        boolean bedsAreEnough = true;
        if (boyBedsNumNotEnough()) {
            Alert alert1 = new Alert(Alert.AlertType.ERROR);
            alert1.setTitle("Error Dialog");
            alert1.setHeaderText("An Error has Occurred!");
            alert1.setContentText("Beds for boys are not enough!");
            alert1.showAndWait();
            bedsAreEnough = false;
        }
        if (girlBedsNumNotEnough()) {
            Alert alert1 = new Alert(Alert.AlertType.ERROR);
            alert1.setTitle("Error Dialog");
            alert1.setHeaderText("An Error has Occurred!");
            alert1.setContentText("Beds for girls are not enough!");
            alert1.showAndWait();
            bedsAreEnough = false;
        }
        if (bedsAreEnough) {
            List<Room> fixedGenes = getFixedGenes();
            Population population = new Population(1000, 0.1, 1000, fixedGenes,
year);
            if (population.getUnallocatedStudents().size() != 0) {
                SimpleDoubleProperty progress = new SimpleDoubleProperty(-
1.0);
                Stage stage = new Stage();
                FXMLLoader              loader              =              new
FXMLLoader(getClass().getResource("/fxmls/main/RunningGA.fxml"));
                stage.setScene(new Scene(loader.load()));
                RunningGAController controller = loader.getController();
                controller.getProgressIndicator().setProgress(-1.0f);
                stage.initModality(Modality.APPLICATION_MODAL);
                stage.setTitle("Genetic Algorithm Running");
                stage.setResizable(false);
                stage.centerOnScreen();
                stage.setOnCloseRequest(Event::consume);
                stage.show();

controller.getProgressIndicator().progressProperty().bind(progress);
                Service<Void> backgroundThread = new Service<>() {
                    @Override
                    protected Task<Void> createTask() {
                        return new Task<>() {
                            @Override
                            protected Void call() {
                                runGA(controller,     progress,     population,
fixedGenes);
                                return null;
                            }
                        };
                    }
                };
                backgroundThread.start();
            } else {
                Alert alert1 = new Alert(Alert.AlertType.ERROR);
                alert1.setTitle("Error Dialog");
                alert1.setHeaderText("An Error has Occurred!");
                alert1.setContentText("All students are allocated!");
                alert1.showAndWait();
            }
        }
    }

    private boolean informationIsNotComplete() {
        try {
            Statement stmt = c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Students;");
            while (rs.next()) {
                for (int i = 2; i <= 6; i++) {
                    if (rs.getString(i) == null || rs.getString(i).isEmpty())
                        return true;
                }
                if (rs.getInt(7) == 0)
                    return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    @FXML
    void mouseClicked(MouseEvent event) {

addMenu.disableProperty().bind(Bindings.createBooleanBinding(this::treeView
Validation));

removeMenuItem.disableProperty().bind(Bindings.createBooleanBinding(this::t
ableViewValidation));

moveMenuItem.disableProperty().bind(Bindings.createBooleanBinding(this::tab
leViewValidation));

switchMenuItem.disableProperty().bind(Bindings.createBooleanBinding(this::t
ableViewValidation));
        MouseButton mouseButton = event.getButton();
        if (mouseButton.equals(MouseButton.SECONDARY)) {
            contextMenu.show(treeView.getScene().getWindow(),
event.getScreenX(), event.getScreenY());
            addYear1Student.setOnAction(event1 -> {
                if (bedsAvailableTextField.getText().equals("0")) {
                    showNoSpareCapacityAlert();
                } else {
                    int boyGirl = 0;
                    if (boyGirlTextField.getText().equals("Boy")) boyGirl = 0;
                    if (boyGirlTextField.getText().equals("Girl")) boyGirl = 1;
                    String                       room                       =
treeView.getSelectionModel().getSelectedItem().getValue();
                    String                     building                     =
treeView.getSelectionModel().getSelectedItem().getParent().getValue();
                    addToTableView(getStudent(1,  boyGirl),  room,  building,
false);

bedsAvailableTextField.setText(Integer.toString(Integer.parseInt(roomCapaci
tyTextField.getText()) - roomTableView.getItems().size()));
                }
            });
            addYear2Student.setOnAction(event1 -> {
                if (bedsAvailableTextField.getText().equals("0")) {
                    showNoSpareCapacityAlert();
                } else {
                    int boyGirl = 0;
                    if (boyGirlTextField.getText().equals("Boy")) boyGirl = 0;
                    if (boyGirlTextField.getText().equals("Girl")) boyGirl = 1;
                    String                        room                     =
treeView.getSelectionModel().getSelectedItem().getValue();
                    String                      building                   =
treeView.getSelectionModel().getSelectedItem().getParent().getValue();
                    addToTableView(getStudent(2,  boyGirl),  room,  building,
false);

bedsAvailableTextField.setText(Integer.toString(Integer.parseInt(roomCapaci
tyTextField.getText()) - roomTableView.getItems().size()));
                }
            });
            removeMenuItem.setOnAction(event1 -> removeStudent());
            moveMenuItem.setOnAction(event1 -> {
                StudentString                    student                   =
roomTableView.getSelectionModel().getSelectedItem();
                List<String> rooms = getRoomsList(false, student);
                ChoiceDialog<String> dialog = new ChoiceDialog<>("",rooms);
                dialog.setTitle("Choice Dialog");
                dialog.setHeaderText("Move Student To...");
                dialog.setContentText("Choose the room:");
                Optional<String> result = dialog.showAndWait();
                String building;
                String room;
                if (result.isPresent()) {
                    building = result.get().split(",")[0];
                    room = result.get().split(",")[1];
                    if
(room.equals(treeView.getSelectionModel().getSelectedItem().getValue())   &&
building.equals(treeView.getSelectionModel().getSelectedItem().getParent().
getValue())) {
                        Alert alert = new Alert(Alert.AlertType.ERROR);
                        alert.setTitle("Error Dialog");
                        alert.setHeaderText("An Error has Occurred");
                        alert.setContentText("The selected student is already
in this room!");
                        alert.showAndWait();
                    } else {
                        if (hasSpareCapacity(room, building)) {
                            try {

                                Statement stmt =
MainController.c.createStatement();

                                ResultSet rs = stmt.executeQuery("SELECT * FROM
Rooms WHERE \"Room No./Name\" = \"" + room + "\" AND \"Building No./Name\" =
\"" + building + "\";");
                                rs.next();
                                int roomCapacity = rs.getInt(4);
                                int columnIndex = 0;
                                for (int i = 6; i < 6 + roomCapacity; i++) {
                                    int studentId = rs.getInt(i);
                                    if (studentId == 0) {
                                        columnIndex = i;
                                        break;
                                    }
                                }
                                rs.close();
                                String columnHeader = "Student " +
Integer.toString(columnIndex - 5);
                                stmt.executeUpdate("UPDATE Rooms SET '" +
columnHeader + "' = " +
roomTableView.getSelectionModel().getSelectedItem().getId() + " WHERE \"Room
No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" + building +
"\";");
                                MainController.c.commit();
                                stmt.close();
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                            removeStudent();
                        } else {
                            showNoSpareCapacityAlert();
                        }
                    }
                }
            });
            switchMenuItem.setOnAction(event1 -> {
                StudentString                    student                    =
roomTableView.getSelectionModel().getSelectedItem();
                List<String> rooms = getRoomsList(true, student);
                ChoiceDialog<String> dialog = new ChoiceDialog<>("",rooms);
                dialog.setTitle("Choice Dialog");
                dialog.setHeaderText("Switch Students");
                dialog.setContentText("Switch with the student in room:");
                Optional<String> result = dialog.showAndWait();
                String building;
                String room;
                if (result.isPresent()) {
                    building = result.get().split(",")[0];
                    room = result.get().split(",")[1];
                    if
(room.equals(treeView.getSelectionModel().getSelectedItem().getValue())   &&
building.equals(treeView.getSelectionModel().getSelectedItem().getParent().
getValue())) {
                        Alert alert = new Alert(Alert.AlertType.ERROR);
                        alert.setTitle("Error Dialog");
                        alert.setHeaderText("An Error has Occurred");
                        alert.setContentText("The selected student is already
in this room!");
                        alert.showAndWait();
                    } else {
                        Room room1 = new Room();
                        room1.setRoom(room);
                        room1.setBuilding(building);
                        if (room1.isEmpty()) {
                            Alert alert = new Alert(Alert.AlertType.ERROR);
                            alert.setTitle("Error Dialog");
                            alert.setHeaderText("An Error has Occurred");
                            alert.setContentText("There is no student allocated
in the selected room!");
                            alert.showAndWait();
                        } else {
                            try {
                                Stage stage = new Stage();
                                stage.setTitle("Switch with...");
                                FXMLLoader         loader         =         new
FXMLLoader(getClass().getResource("/fxmls/main/SwitchStudents.fxml"));
```

```
                                stage.setScene(new Scene(loader.load()));
                                SwitchStudentsController      controller      =
loader.getController();
                                stage.initModality(Modality.APPLICATION_MODAL);
                                stage.setResizable(false);
                                stage.setOnCloseRequest(event2              ->
controller.setSelectedItem(null));
                                stage.setOnShowing(event2                   ->
controller.populateTableView(room, building));
                                stage.setOnHiding(event2 -> {
                                    if (controller.getSelectedItem() != null) {
                                        StudentString         student1         =
controller.getSelectedItem();
                                        StudentString         student2         =
roomTableView.getSelectionModel().getSelectedItem();
                                        switchStudentsInDB(student1, student2);
                                    }
                                });
                                stage.showAndWait();
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }

populateContent(treeView.getSelectionModel().getSelectedItem());
            });
        }
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        roomCapacityTextField.setEditable(false);
        bedsAvailableTextField.setEditable(false);
        showTreeView();
        addMenu.getItems().addAll(addYear1Student, addYear2Student);
        contextMenu.getItems().addAll(addMenu,      removeMenuItem,      new
SeparatorMenuItem(), moveMenuItem,switchMenuItem);
        treeView.getSelectionModel()
                .selectedItemProperty()
                .addListener((observable,     oldValue,     newValue)     ->
populateContent(newValue));
    }
    private void populateContent(TreeItem<String> newValue) {
        if (treeView.getSelectionModel().getSelectedItem() != null) {
            int flag = 1;
            for (int i = 0; i < buildingNames.size(); i++) {
                if (buildingNames.get(i).equals(newValue.getValue())) {
                    flag = 0;
                    break;
                }
            }
            if (flag == 0) {
                bedsAvailableTextField.setText("");
                roomCapacityTextField.setText("");
                boyGirlTextField.setText("");
                roomTableView.setItems(null);
            } else {
                String room = newValue.getValue();
                String building = newValue.getParent().getValue();
                givenNameColumn.setStyle("-fx-alignment: CENTER;");
                familyNameColumn.setStyle("-fx-alignment: CENTER;");
                sexColumn.setStyle("-fx-alignment: CENTER;");
                countryColumn.setStyle("-fx-alignment: CENTER;");
                continentColumn.setStyle("-fx-alignment: CENTER;");
                yearColumn.setStyle("-fx-alignment: CENTER;");
                givenNameColumn.setCellValueFactory(new
PropertyValueFactory<>("givenName"));
                familyNameColumn.setCellValueFactory(new
PropertyValueFactory<>("familyName"));
                sexColumn.setCellValueFactory(new
PropertyValueFactory<>("sex"));
                countryColumn.setCellValueFactory(new
PropertyValueFactory<>("country"));
                continentColumn.setCellValueFactory(new
PropertyValueFactory<>("continent"));
                yearColumn.setCellValueFactory(new
PropertyValueFactory<>("year"));
                try {
                    Statement stmt = MainController.c.createStatement();
                    ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE
\"Room No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" +
building + "\";");
                    rs.next();
                    int roomCapacity = rs.getInt(4);
                    String boyGirl = rs.getString(5);
                    rs.close();

roomCapacityTextField.setText(Integer.toString(roomCapacity));
                    boyGirlTextField.setText(boyGirl);
                    ResultSet  rs1  =  stmt.executeQuery("SELECT  *  FROM  Rooms
WHERE \"Room No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" +
building + "\";");
                    rs1.next();
                    students.clear();
                    for (int i = 6; i < 6 + roomCapacity; i++) {
                        int studentId = rs1.getInt(i);
                        if (studentId == 0) {
                            continue;
                        }
                        Statement stmt1 = MainController.c.createStatement();
                        ResultSet  rs2  =  stmt1.executeQuery("SELECT  *  FROM
Students WHERE Id = " + studentId + ";");
                        StudentString student = new StudentString();
                        student.setId(rs2.getInt("Id"));
                        student.setContinent(rs2.getString("Continent"));
                        student.setCountry(rs2.getString("Country"));
                        student.setSex(rs2.getString("Sex"));
                        student.setFamilyName(rs2.getString("FamilyName"));
                        student.setGivenName(rs2.getString("GivenName"));
                        student.setYear(rs2.getInt("Year"));
                        students.add(student);
                        stmt1.close();
                    }
                    roomTableView.setItems(students);
                    try {
                        int studentNum = roomTableView.getItems().size();

bedsAvailableTextField.setText(Integer.toString(roomCapacity - studentNum));
                    } catch (NullPointerException e) {

bedsAvailableTextField.setText(Integer.toString(roomCapacity));
                    }
                    stmt.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
    private boolean treeViewValidation() {
        TreeItem<String>                    selectedItem                    =
treeView.getSelectionModel().getSelectedItem();
        if (selectedItem == (null)) {
            return true;
        } else {
            return (buildingsTreeItems.contains(selectedItem));
        }
    }

    private boolean tableViewValidation() {
        return (roomTableView.getSelectionModel().getSelectedItem() == null);
    }

    private StudentString getStudent(int year, int boyGirl) { // boy = 0,
girl = 1;
        try {
            Stage stage = new Stage();
            if (year == 1) {
                FXMLLoader              loader              =              new
FXMLLoader(getClass().getResource("/fxmls/main/AddYear1Student.fxml"));
                stage.setScene(new Scene(loader.load()));
                stage.setTitle("Add Year 1 Student");
                AddYear1StudentController controller = loader.getController();
                stage.initModality(Modality.APPLICATION_MODAL);
                stage.setResizable(false);
                stage.setOnCloseRequest(event                                ->
controller.setSelectedItem(null));
                stage.setOnShowing(event -> {
                    controller.setBoyGirl(boyGirl);
                    controller.populateTableView();
                });
                stage.showAndWait();
                return controller.okClick(new ActionEvent());
            } else {
                FXMLLoader              loader              =              new
FXMLLoader(getClass().getResource("/fxmls/main/AddYear2Student.fxml"));
                stage.setScene(new Scene(loader.load()));
                stage.setTitle("Add Year 2 Student");
                AddYear2StudentController controller = loader.getController();
                stage.initModality(Modality.APPLICATION_MODAL);
                stage.setResizable(false);
                stage.setOnCloseRequest(event                                ->
controller.setSelectedItem(null));
                stage.setOnShowing(event -> {
                    controller.setBoyGirl(boyGirl);
                    controller.populateTableView();
                });
                stage.showAndWait();
                return controller.okClick(new ActionEvent());
            }
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    private void addToTableView(StudentString student, String room, String
building, boolean isMove) {
        if (student != null) {
            try {
                Statement stmt = MainController.c.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE
\"Room No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" +
building + "\";");
                rs.next();
                int roomCapacity = rs.getInt(4);
                int columnIndex = 0;
                for (int i = 6; i < 6 + roomCapacity; i++) {
                    int studentId = rs.getInt(i);
                    if (studentId == 0) {
                        columnIndex = i;
                        break;
                    }
                }
                String columnHeader = "Student " +
Integer.toString(columnIndex - 5);
                stmt.executeUpdate("UPDATE Rooms SET '" + columnHeader + "' =
" + student.getId() + " WHERE \"Room No./Name\" = \"" + room + "\" AND
\"Building No./Name\" = \"" + building + "\";");
                MainController.c.commit();
                stmt.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            if (!isMove) {
                students.add(student);
                roomTableView.refresh();
            }
        }
    }

    void removeStudent() {
        StudentString                    studentToRemove                    =
roomTableView.getSelectionModel().getSelectedItem();
        students.remove(studentToRemove);
        roomTableView.setItems(students);
        try {
            String                              room                        =
treeView.getSelectionModel().getSelectedItem().getValue();
            String                            building                      =
treeView.getSelectionModel().getSelectedItem().getParent().getValue();

            Statement stmt = MainController.c.createStatement();

            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE \"Room
No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" + building +
"\";");
            rs.next();
            int columnIndex = 0;
            for      (int      i      =      6;      i      <      6      +
Integer.parseInt(roomCapacityTextField.getText()); i++) {
                int studentId = rs.getInt(i);
                if (studentId == studentToRemove.getId()) {
                    columnIndex = i;
                    break;
                }
            }
            String columnHeader = "Student " + Integer.toString(columnIndex -
5);
            stmt.executeUpdate("UPDATE Rooms SET \"" + columnHeader + "\" = "
+ null + " WHERE \"Room No./Name\" = \"" + room + "\" AND \"Building No./Name\"
= \"" + building + "\";");
            MainController.c.commit();
            stmt.close();

bedsAvailableTextField.setText(Integer.toString(Integer.parseInt(roomCapaci
tyTextField.getText()) - roomTableView.getItems().size()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void showNoSpareCapacityAlert() {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Information Dialog");
        alert.setHeaderText("No Available Bed");
        alert.setContentText("Sorry, there is no available bed in this room!");
        alert.showAndWait();
    }

    private boolean hasSpareCapacity(String room, String building) {
        int spareCapacity = 0;
        try {

            Statement stmt = MainController.c.createStatement();

            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE \"Room
No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" + building +
"\";");
            rs.next();
            int roomCapacity = rs.getInt(4);
            for (int i = 6; i < 6 + roomCapacity; i++) {
                if (rs.getInt(i) == 0) spareCapacity++;
            }
            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return (spareCapacity > 0);
    }

    private void showTreeView() {
        roomTableView.setItems(null);
        bedsAvailableTextField.setText("");
        boyGirlTextField.setText("");
        roomCapacityTextField.setText("");
        File   file   =   new   File(MainController.directory   +   "/"   +
MainController.fileName + ".sqlite");
        treeView.setRoot(new TreeItem<>());
        if (file.exists()) {
            buildingNames.clear();
            root = new TreeItem<>();
            treeView.setRoot(root);
            try {
                Statement stmt = MainController.c.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms");
                List<Room> rooms = new ArrayList<>();
                while (rs.next()) {
                    Room room = new Room();
                    room.setBuilding(rs.getString(3));
                    room.setRoom(rs.getString(2));
                    rooms.add(room);
                }
                rooms.sort(MainController::roomComparator);
                TreeItem<String> currentTreeItem = null;
                for (Room room: rooms) {
                    if (!buildingNames.contains(room.getBuilding())) {
                        buildingNames.add(room.getBuilding());
                        currentTreeItem = new TreeItem<>(room.getBuilding());
                        root.getChildren().add(currentTreeItem);
                    }
                    currentTreeItem.getChildren().add(new
TreeItem<>(room.getRoom()));
                }
                buildingsTreeItems = root.getChildren();
                stmt.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void writeDirectoryFile() {
        try {
            String                          directory                           =
(MainController.class.getProtectionDomain().getCodeSource().getLocation().t
oURI()).getPath().replace("\\", "/");
            directory = directory.substring(0, directory.lastIndexOf("/") + 1)
+ "/Directory.txt";
            File file = new File(directory);
            file.createNewFile();
            FileWriter fw = new FileWriter(directory,false);
            PrintWriter pw = new PrintWriter(new BufferedWriter(fw));
            pw.println(MainController.fileName);
            pw.println(MainController.directory);
            pw.flush();
            pw.close();
            fw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void openWindow(String fxml, String windowTitle) {
        try {
            Parent                      directoryLayout                        =
FXMLLoader.load(getClass().getResource(fxml));
            Stage fileNewStage = new Stage();
            fileNewStage.setScene(new Scene(directoryLayout));
            fileNewStage.setTitle(windowTitle);
            fileNewStage.setResizable(false);
            fileNewStage.initModality(Modality.APPLICATION_MODAL);
            fileNewStage.centerOnScreen();
            fileNewStage.showAndWait();
            showTreeView();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private  List<String>  getRoomsList(boolean  switchRoom,  StudentString
student) {
        List<String> rooms = new ArrayList<>();
        try {
            Statement stmt = MainController.c.createStatement();
            for (TreeItem<String> building: root.getChildren()) {
                String currentBuilding = building.getValue();
                for (TreeItem<String> room: building.getChildren()) {
                    ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE
\"Room No./Name\" = \"" + room.getValue() + "\" AND \"Building No./Name\" =
\"" + currentBuilding + "\";");
                    int columnNum = rs.getMetaData().getColumnCount();
                    rs.next();
                    if (rs.getString(5).equals(boyGirlTextField.getText()) &&
(!rs.getString(2).equals(student.getRoom())
|| !rs.getString(3).equals(student.getBuilding()))) {
                        if (switchRoom) {
                            boolean roomIsEmpty = true;
                            for (int i = 6; i<= columnNum; i++) {
                                if (rs.getInt(i) != 0) {
                                    roomIsEmpty = false;
                                    break;
                                }
                            }
                            if (!roomIsEmpty) {
                                rooms.add(currentBuilding      +      ","      +
room.getValue());
                            }
                        } else {
                            int maxCapacity = rs.getInt(4);
                            int studentNum = 0;
                            for (int i = 6; i <= columnNum; i++) {
                                if (rs.getInt(i) != 0) studentNum++;
                            }
                            if (studentNum < maxCapacity) {
                                rooms.add(currentBuilding      +      ","      +
room.getValue());
                            }
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rooms;
    }

    private  void  switchStudentsInDB(StudentString  student1,  StudentString
student2) {
        String student1Room = student1.getRoom();
        String student2Room = student2.getRoom();
        String student1Building = student1.getBuilding();
        String student2Building = student2.getBuilding();
        try {

            Statement stmt = MainController.c.createStatement();

            ResultSet  rs1  =  stmt.executeQuery("SELECT  *  FROM  Rooms  WHERE
\"Room No./Name\" = \"" + student1Room + "\" AND \"Building No./Name\" = \""
+ student1Building + "\";");
            ResultSetMetaData rsmd = rs1.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            int columnNum1 = 0;
            rs1.next();
            for (int i = 6; i <= numberOfColumns; i++) {
                if (student1.getId() == rs1.getInt(i)) {
                    columnNum1 = i;
                    break;
                }
            }
            stmt.executeUpdate("UPDATE Rooms SET \"Student " + (columnNum1-5)
+ "\" = " + student2.getId() + " WHERE \"Room No./Name\" = \"" + student1Room
+ "\" AND \"Building No./Name\" = \"" + student1Building + "\";");
            MainController.c.commit();
            ResultSet  rs2  =  stmt.executeQuery("SELECT  *  FROM  Rooms  WHERE
\"Room No./Name\" = \"" + student2Room + "\" AND \"Building No./Name\" = \""
+ student2Building + "\";");
            int columnNum2 = 0;
            rs2.next();
            for (int i = 6; i <= numberOfColumns; i++) {
                if (student2.getId() == rs2.getInt(i)) {
                    columnNum2 = i;
                    break;
                }
            }
            stmt.executeUpdate("UPDATE Rooms SET \"Student " + (columnNum2-5)
+ "\" = " + student1.getId() + " WHERE \"Room No./Name\" = \"" + student2Room
+ "\" AND \"Building No./Name\" = \"" + student2Building + "\";");
            MainController.c.commit();
            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void connectToDB() {
        try {
            Class.forName("org.sqlite.JDBC");
            MainController.c  =  DriverManager.getConnection("jdbc:sqlite:"  +
MainController.directory + "/" + MainController.fileName + ".sqlite");
            MainController.c.setAutoCommit(false);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @FXML
    void viewRoomClicked(ActionEvent event) throws IOException {
        FXMLLoader                loader                =                new
```

```
FXMLLoader(getClass().getResource("/fxmls/view/Room.fxml"));
        Stage stage = new Stage();
        stage.setScene(new Scene(loader.load()));
        stage.setTitle("Room Information");
        stage.setResizable(false);
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.centerOnScreen();
        stage.showAndWait();
    }

    @FXML
    void viewStudentClicked(ActionEvent event) throws IOException {
        FXMLLoader                loader                =                new
FXMLLoader(getClass().getResource("/fxmls/view/Student.fxml"));
        Stage stage = new Stage();
        stage.setScene(new Scene(loader.load()));
        stage.setTitle("Room Information");
        stage.setResizable(false);
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.centerOnScreen();
        stage.showAndWait();
    }

    private boolean boyBedsNumNotEnough() {
        int boyNum = 0;
        int numBoyBeds = 0;
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Students;");
            while (rs.next()) {
                if (rs.getString(4).equals("male")) boyNum++;
            }
            ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
            while (rs1.next()) {
                if (rs1.getString(5).equals("Boy")) {
                    numBoyBeds += rs1.getInt(4);
                }
            }
            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return (numBoyBeds < boyNum);
    }
    private boolean girlBedsNumNotEnough() {
        int girlNum = 0;
        int numGirlBeds = 0;
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Students;");
            while (rs.next()) {
                if (rs.getString(4).equals("female")) girlNum++;
            }
            ResultSet rs1 = stmt.executeQuery("SELECT * FROM Rooms;");
            while (rs1.next()) {
                if (rs1.getString(5).equals("Girl")) {
                    numGirlBeds += rs1.getInt(4);
                }
            }
            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return (numGirlBeds < girlNum);
    }

    private List<Room> getFixedGenes() {
        List<Room> fixedGenes = new ArrayList<>();
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            while (rs.next()) {
                Room room = new Room();
                room.setId(rs.getInt(1));
                room.setRoom(rs.getString(2));
                room.setBuilding(rs.getString(3));
                room.setMaxResidents(rs.getInt(4));
                room.setSexRoom(rs.getString(5));
                for (int i = 6; i <= numberOfColumns; i++) {
                    int studentId = rs.getInt(i);
                    if (studentId != 0) {
                        Statement stmt1 = MainController.c.createStatement();
                        ResultSet  rs1  =  stmt1.executeQuery("SELECT  *  FROM
Students WHERE Id = " + studentId + ";");
                        rs1.next();
                        StudentString student = new StudentString();
                        student.setId(studentId);
                        student.setGivenName(rs1.getString("GivenName"));
                        student.setFamilyName(rs1.getString("FamilyName"));
                        student.setSex(rs1.getString("Sex"));
                        student.setCountry(rs1.getString("Country"));
                        student.setContinent(rs1.getString("Continent"));
                        student.setYear(rs1.getInt("Year"));
                        room.getStudents().add(student);
                    }
                }
                fixedGenes.add(room);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return fixedGenes;
    }

    @FXML
    void exportClicked(ActionEvent event) throws IOException {
        Stage stage = new Stage();
        FXMLLoader                loader                  =                 new
FXMLLoader(getClass().getResource("/fxmls/main/ShowUnallocatedStudents.fxml
"));
        stage.setScene(new Scene(loader.load()));
        ShowUnallocatedStudentsController            controller             =
loader.getController();
        stage.setOnHiding(event1 -> {
            if (controller.isProceed()) {
                String directory = chooseDirectory();
                if (directory != null) {
                    if (exportToExcel(directory)) {
                        Alert alert = new Alert(Alert.AlertType.INFORMATION);
                        alert.setTitle("Information Dialog");
                        alert.setHeaderText(null);
                        alert.setContentText("The  Excel  File  is  successfully
generated!");
                        alert.show();
                    } else  {
                        Alert alert = new Alert(Alert.AlertType.ERROR);
                        alert.setTitle("Error Dialog");
                        alert.setHeaderText(null);
                        alert.setContentText("An Unexpected Error has Occurred!
Please retry.");
                        alert.show();
                    }
                }
            }
        });
        stage.setOnShown(event1 -> {
            if (controller.getStudentTableView().getItems().isEmpty()) {
                controller.setProceed(true);
                controller.getStage().close();
            }
        });
        stage.show();
    }

    private void runGA(RunningGAController controller, SimpleDoubleProperty
progress, Population population, List<Room> fixedGenes) {
        population.calcFitness();
        do {
            population.naturalSelection();
            population.calcFitness();
        } while (population.evaluate());
        Platform.runLater(() -> {
            progress.set(1.0);
            controller.getLabel1().setVisible(false);
            controller.getLabel2().setVisible(false);
            controller.getLabel3().setVisible(false);
            controller.getOkButton().setDisable(false);
            controller.getOkButton().setVisible(true);
        });
        List<Room> bestAllocation = population.getBestOne().getGenes();
        for (Room room: bestAllocation) {
            try {
                int roomId = room.getId();
                List<StudentString> students = room.getStudents();
                List<StudentString> studentsToRemove = new ArrayList<>();
                for (StudentString student: students) {
                    if                              (fixedGenes.get(roomId-
1).getStudents().contains(student))
                        studentsToRemove.add(student);
                }
                for (StudentString studentToRemove: studentsToRemove) {
                    students.remove(studentToRemove);
                }
```

```
                for (StudentString student: students) {
                    Statement stmt = MainController.c.createStatement();
                    ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE
Id = " + room.getId() + ";");
                    ResultSetMetaData rsmd = rs.getMetaData();
                    int numberOfColumns = rsmd.getColumnCount();
                    rs.next();
                    for (int i = 6; i <= numberOfColumns; i++) {
                        if (rs.getInt(i) == 0) {
                            Statement                    stmt1                    =
MainController.c.createStatement();
                            stmt1.executeUpdate("UPDATE Rooms SET \"Student "
+ (i-5) + "\" = " + student.getId() + " WHERE Id = " + room.getId() + ";");
                            MainController.c.commit();
                            break;
                        }
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        Platform.runLater(() -> showTreeView());
    }

    @FXML
    void uploadYear1StudentClicked(ActionEvent event) throws IOException {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText("Please Confirm...");
        alert.setContentText("The   uploaded   students   will   overwrite   the
existing students.\nAre you sure to proceed?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            openWindow("/fxmls/configurations/UploadYear1Student.fxml",
"Year 1 Student Upload");
        }
    }

    @FXML
    void uploadYear2StudentClicked(ActionEvent event) throws IOException {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText("Please Confirm...");
        alert.setContentText("The  uploaded  students  will  overwrite  the
existing students.\nAre you sure to proceed?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            openWindow("/fxmls/configurations/UploadYear2Student.fxml",
"Year 2 Student Upload");
        }
    }

    private String chooseDirectory() {
        String directory;
        final DirectoryChooser dirChooser = new DirectoryChooser();
        Stage currentStage = (Stage) roomTableView.getScene().getWindow();
        File file = dirChooser.showDialog(currentStage);
        if (file != null) {
            directory = file.getAbsolutePath().replace("\\", "/");
        } else {
            directory = null;
        }
        return directory;
    }

    private boolean exportToExcel(String directory) {
        try {
            Statement stmt = c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnNum = rsmd.getColumnCount();
            XSSFWorkbook workbook = new XSSFWorkbook();
            XSSFSheet sheet = workbook.createSheet(fileName);
            XSSFRow header = sheet.createRow(0);
            header.createCell(0).setCellValue("Building");
            header.createCell(1).setCellValue("Room");
            header.createCell(2).setCellValue("Boy/Girl");
            for (int i = 6; i <= columnNum; i++) {
                header.createCell(i-3).setCellValue("Student " + (i-5));
            }
            List<Room> rooms = new ArrayList<>();
            while (rs.next()) {
                Room room = new Room();
                room.setRoom(rs.getString(2));
                room.setBuilding(rs.getString(3));
                room.setSexRoom(rs.getString(5));
                for (int j = 6; j <= columnNum; j++) {
                    int studentId = rs.getInt(j);
                    if (studentId != 0) {
                        Statement stmt1 = MainController.c.createStatement();
                        ResultSet  rs1  =  stmt1.executeQuery("SELECT  *  FROM
Students WHERE Id = " + studentId + ";");
                        rs1.next();
                        StudentString student = new StudentString();
                        student.setFamilyName(rs1.getString(3));
                        student.setGivenName(rs1.getString(2));
                        room.getStudents().add(student);
                    }
                }
                rooms.add(room);
            }
            rooms.sort(MainController::roomComparator);
            int i = 1;
            for (Room room: rooms) {
                XSSFRow row = sheet.createRow(i);
                row.createCell(0).setCellValue(room.getBuilding());
                row.createCell(1).setCellValue(room.getRoom());
                row.createCell(2).setCellValue(room.getSexRoom());
                int k = 3;
                for (StudentString student: room.getStudents()) {
                    row.createCell(k).setCellValue(student.getGivenName()  +  "
" + student.getFamilyName());
                    k++;
                }
                i++;
            }
            FileOutputStream          fileOutputStream          =          new
FileOutputStream(directory + "/" + fileName + ".xlsx");
            workbook.write(fileOutputStream);
            fileOutputStream.close();
            return true;
        } catch (NoClassDefFoundError | IOException | SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    public static int roomComparator(Room o1, Room o2) {
        String building1 = o1.getBuilding();
        String building2 = o2.getBuilding();
        int result = building1.compareTo(building2);
        if (result != 0) {
            return result;
        } else {
            String room1 = o1.getRoom();
            String room2 = o2.getRoom();
            if (room1.length() < room2.length()) return -1;
            else if (room1.length() > room2.length()) return 1;
            else return room1.compareTo(room2);
        }
    }
}
```

```
package controllers.main;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ProgressIndicator;
import javafx.stage.Stage;

public class RunningGAController {
    public ProgressIndicator getProgressIndicator() {
        return progressIndicator;
    }
    @FXML
    private ProgressIndicator progressIndicator;
    public Label getLabel1() {
        return label1;
    }
    public Label getLabel2() {
        return label2;
    }
    public Label getLabel3() {
        return label3;
    }
    public Button getOkButton() {
        return okButton;
    }
    @FXML
    private Label label1;
    @FXML
    private Label label2;
    @FXML
    private Label label3;

    @FXML
    private Button okButton;

    @FXML
    void okClicked(ActionEvent event) {
        Stage stage = (Stage) okButton.getScene().getWindow();
        stage.close();
    }

}
```

```
package controllers.main;

import GA.Population;
import functional.StudentString;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import java.net.URL;
import java.util.List;
import java.util.ResourceBundle;

public class ShowUnallocatedStudentsController implements Initializable {
    public boolean isProceed() {
        return proceed;
    }
    public void setProceed(boolean proceed) {
        this.proceed = proceed;
    }
    private boolean proceed;
    @FXML
    private Button okButton;
    @FXML
    private Button cancelButton;
    public TableView<StudentString> getStudentTableView() {
        return studentTableView;
    }
    @FXML
    private TableView<StudentString> studentTableView;
    @FXML
    private TableColumn<StudentString, String> givenNameColumn;
    @FXML
    private TableColumn<StudentString, String> familyNameColumn;
    @FXML
    private TableColumn<StudentString, String> sexColumn;
    @FXML
    private TableColumn<StudentString, String> countryColumn;
    @FXML
    private TableColumn<StudentString, String> continentColumn;

    @FXML
    void cancelClick(ActionEvent event) {
        proceed = false;
        Stage stage = (Stage) okButton.getScene().getWindow();
        stage.close();
    }

    @FXML
    void okClick(ActionEvent event) {
        proceed = true;
        Stage stage = (Stage) okButton.getScene().getWindow();
        stage.close();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        studentTableView.setEditable(false);
        sexColumn.setStyle("-fx-alignment: CENTER;");
        countryColumn.setStyle("-fx-alignment: CENTER;");
        continentColumn.setStyle("-fx-alignment: CENTER;");
        givenNameColumn.setCellValueFactory(new
PropertyValueFactory<>("givenName"));
        familyNameColumn.setCellValueFactory(new
PropertyValueFactory<>("familyName"));
        sexColumn.setCellValueFactory(new PropertyValueFactory<>("sex"));
        countryColumn.setCellValueFactory(new
PropertyValueFactory<>("country"));
        continentColumn.setCellValueFactory(new
PropertyValueFactory<>("continent"));
        List<StudentString>              unallocatedStudents              =
Population.findUnallocatedStudents(3);
        ObservableList<StudentString>              students              =
FXCollections.observableArrayList();
        students.addAll(unallocatedStudents);
        studentTableView.setItems(students);
    }

    Stage getStage() {
        return (Stage) studentTableView.getScene().getWindow();
    }
}
```

```
package controllers.main;

import functional.HandleButton;
import functional.StudentString;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

public class SwitchStudentsController implements Initializable {

    private    ObservableList<StudentString>    studentObservableList    =
FXCollections.observableArrayList();

    public StudentString getSelectedItem() {
        return selectedItem;
    }

    public void setSelectedItem(StudentString selectedItem) {
        this.selectedItem = selectedItem;
    }

    private StudentString selectedItem;

    @FXML
    private Button okButton;

    @FXML
    private Button cancelButton;
    @FXML
    private TableView<StudentString> studentTableView;

    @FXML
    private TableColumn<StudentString, String> givenNameColumn;

    @FXML
    private TableColumn<StudentString, String> familyNameColumn;

    @FXML
    private TableColumn<StudentString, String> sexColumn;

    @FXML
    private TableColumn<StudentString, String> countryColumn;

    @FXML
    private TableColumn<StudentString, String> continentColumn;

    @FXML
    void keyTyped(KeyEvent event) {
        selectedItem                                                     =
studentTableView.getSelectionModel().getSelectedItem();
    }

    @FXML
    void mouseClicked(MouseEvent event) {
        selectedItem                                                     =
studentTableView.getSelectionModel().getSelectedItem();
    }

    @FXML
    void cancelClick(ActionEvent event) {
        selectedItem = null;
        HandleButton button = new HandleButton();
        button.handleCancelButton(cancelButton);
    }

    @FXML
    void okClick(ActionEvent event) {
        Stage currentStage = (Stage) okButton.getScene().getWindow();
        currentStage.close();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        AddYear1StudentController.initializingContents(okButton,
studentTableView, sexColumn, countryColumn, continentColumn, givenNameColumn,
familyNameColumn);
    }

    public void populateTableView(String room, String building) {
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms WHERE \"Room
No./Name\" = \"" + room + "\" AND \"Building No./Name\" = \"" + building +
"\";");
            ResultSetMetaData rsmd = rs.getMetaData();
            int numberOfColumns = rsmd.getColumnCount();
            rs.next();
            List<Integer> studentIds = new ArrayList<>();
            for (int i = 6; i <= numberOfColumns; i++) {
                int studentId = rs.getInt(i);
                if (studentId != 0) {
                    studentIds.add(studentId);
                }
            }
            rs.close();
            for (Integer studentId: studentIds) {
                ResultSet rs1 = stmt.executeQuery("SELECT * FROM Students
WHERE Id = " + studentId + ";");
                StudentString student = new StudentString();
                student.setId(studentId);
                student.setGivenName(rs1.getString("GivenName"));
                student.setFamilyName(rs1.getString("FamilyName"));
                student.setSex(rs1.getString("Sex"));
                student.setCountry(rs1.getString("Country"));
                student.setContinent(rs1.getString("Continent"));
                student.setYear(rs1.getInt("Year"));
                studentObservableList.add(student);
            }
            stmt.close();

            studentTableView.setItems(studentObservableList);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
package controllers.newFile;


import com.sun.tools.javac.Main;
import functional.HandleButton;
import controllers.main.MainController;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.AnchorPane;
import javafx.stage.DirectoryChooser;
import javafx.stage.Stage;

import java.io.File;
import java.io.IOException;
import java.sql.DriverManager;

public class DirectoryController {

    static String oldFileName;
    static String oldDirectory;

    @FXML
    private TextField scheduleNameInput;

    @FXML
    private TextField scheduleDirectoryInput;

    @FXML
    private Button selectDirectoryButton;

    @FXML
    private Button cancelButton;

    @FXML
    private Button nextButton;

    @FXML
    private Label warningLabel;

    @FXML AnchorPane ap;

    @FXML
    void cancelClick(ActionEvent event) throws IOException {
        HandleButton button = new HandleButton();
        button.handleCancelButton(cancelButton);
    }

    @FXML
    void nextClick(ActionEvent event) throws IOException {
        if (scheduleNameInput.getText().isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Warning");
            alert.setHeaderText(null);
            alert.setContentText("Enter a file name to create a new room
allocation");
            alert.showAndWait();
        } else if (scheduleDirectoryInput.getText().isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Warning");
            alert.setHeaderText(null);
            alert.setContentText("Choose a directory to create a new room
allocation");
            alert.showAndWait();
        } else {
            oldFileName = MainController.fileName;
            oldDirectory = MainController.directory;
            MainController.fileName = scheduleNameInput.getText();
            MainController.directory                                =
scheduleDirectoryInput.getText().replace("\\", "/");
            File   file   =   new   File(MainController.directory   +   "/"   +
MainController.fileName +".sqlite");
            if (file.exists()) {
                MainController.fileName = oldFileName;
                MainController.directory = oldDirectory;
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("An Error has Occurred!");
                alert.setContentText("File Already Exists!\n" +
                        "Open the file by clicking File -> Open in the main
page.");
                alert.showAndWait();
                Stage stage = (Stage) cancelButton.getScene().getWindow();
                stage.close();
            } else {
                MainController.connectToDB();
                Parent                        layout                          =
FXMLLoader.load(getClass().getResource("/fxmls/newFile/RoomConfig.fxml"));
                Stage stage = (Stage) nextButton.getScene().getWindow();
                stage.setScene(new Scene(layout));
                stage.setResizable(false);
                stage.centerOnScreen();
                stage.setOnCloseRequest(e                                    ->
deleteDB(DirectoryController.oldFileName,
DirectoryController.oldDirectory));
                stage.show();
            }
        }
    }

    public static void deleteDB(String oldFileName, String oldDirectory) {
        try {
            MainController.c.close();
            File   file   =   new   File(MainController.directory   +   "/"   +
MainController.fileName +".sqlite");
            file.delete();
            MainController.fileName = oldFileName;
            MainController.directory = oldDirectory;
            MainController.connectToDB();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    @FXML
    void selectDirectoryClick(ActionEvent event) {
        final DirectoryChooser dirChooser = new DirectoryChooser();
        Stage currentStage = (Stage) nextButton.getScene().getWindow();
        File file = dirChooser.showDialog(currentStage);
        if (file != null) {
            scheduleDirectoryInput.setText(file.getAbsolutePath());
        }
    }
}
```

```
package controllers.newFile;

import functional.HandleButton;
import functional.Room;
import controllers.configurations.AddOrDeleteRoomController;
import controllers.main.MainController;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;

import java.io.IOException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ResourceBundle;


public class RoomConfigController implements Initializable {


    private        ObservableList<Room>        roomsObservableList        =
FXCollections.observableArrayList();

    @FXML
    private TableView<Room> roomTableView;

    @FXML
    private TableColumn<Room, Integer> idColumn;

    @FXML
    private TableColumn<Room, String> roomColumn;

    @FXML
    private TableColumn<Room, String> buildingColumn;

    @FXML
    private TableColumn<Room, Integer> maxResidentsColumn;

    @FXML
    private TableColumn<Room, String> sexRoomColumn;
    @FXML
    private ComboBox<String> sexComboBox;

    @FXML
    private Button nextButton;

    @FXML
    private Button cancelButton;

    @FXML
    private Button addButton;

    @FXML
    private Button deleteButton;

    @FXML
    private TextField roomTextField;

    @FXML
    private TextField buildingTextField;

    @FXML
    private TextField maxResidentsTextField;

    @FXML
    void addClick(ActionEvent event) {
        addButtonClicked(roomTextField,                    buildingTextField,
maxResidentsTextField, sexComboBox, roomsObservableList, roomTableView);
    }

    @FXML
    void deleteClick(ActionEvent event) {
        deleteButtonClicked(roomTableView, roomsObservableList);
    }

    @FXML
    void cancelClick(ActionEvent event) throws IOException {
        DirectoryController.deleteDB(DirectoryController.oldFileName,
DirectoryController.oldDirectory);
        HandleButton button = new HandleButton();
        button.handleCancelButton(cancelButton);
    }
    @FXML
    void nextClick(ActionEvent event) throws IOException {
        writeToDB();
        createStudentColumns();
        HandleButton button = new HandleButton();

button.handleNextButton(nextButton,"/fxmls/newFile/StudentConfig.fxml");
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        idColumn.setSortable(false);
        roomColumn.setSortable(false);
        buildingColumn.setSortable(false);
        maxResidentsColumn.setSortable(false);
        sexRoomColumn.setSortable(false);
        sexComboBox.getItems().addAll("Boy", "Girl");
        idColumn.setStyle("-fx-alignment: CENTER;");
        roomColumn.setStyle("-fx-alignment: CENTER;");
        buildingColumn.setStyle("-fx-alignment: CENTER;");
        maxResidentsColumn.setStyle("-fx-alignment: CENTER;");
        sexRoomColumn.setStyle("-fx-alignment: CENTER;");
        idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));
        roomColumn.setCellValueFactory(new PropertyValueFactory<>("room"));
        buildingColumn.setCellValueFactory(new
PropertyValueFactory<>("building"));
        maxResidentsColumn.setCellValueFactory(new
PropertyValueFactory<>("maxResidents"));
        sexRoomColumn.setCellValueFactory(new
PropertyValueFactory<>("sexRoom"));
        roomsObservableList = populateTableView();
        roomTableView.setItems(roomsObservableList);
    }

    public static ObservableList<Room> populateTableView() {
        ObservableList<Room>              roomsObservableList               =
FXCollections.observableArrayList();
        try {


            Statement stmt = MainController.c.createStatement();

            String sql =  "CREATE TABLE IF NOT EXISTS Rooms " +
                    "(Id      INTEGER PRIMARY KEY AUTOINCREMENT," +
                    "'Room No./Name'     TEXT               NOT NULL, " +
                    "'Building No./Name' TEXT               NOT NULL, " +
                    "'Max Residents'     INT                NOT NULL," +
                    "'Boy/Girl'         TEXT               NOT NULL)";
            stmt.executeUpdate(sql);
            MainController.c.commit();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms");
            while (rs.next()) {
                Room room = new Room();
                room.setId(rs.getInt("Id"));
                room.setRoom(rs.getString("Room No./Name"));
                room.setBuilding(rs.getString("Building No./Name"));
                room.setMaxResidents(rs.getInt("Max Residents"));
                room.setSexRoom(rs.getString("Boy/Girl"));
                roomsObservableList.add(room);

            }
            stmt.close();

            return roomsObservableList;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public void writeToDB() {
        try {



            Statement stmt = MainController.c.createStatement();

            String sql1 = "DROP TABLE IF EXISTS Rooms";
            stmt.executeUpdate(sql1);
            MainController.c.commit();
            String sql =  "CREATE TABLE Rooms " +
                    "(Id            INTEGER   PRIMARY KEY   AUTOINCREMENT," +
                    "'Room No./Name'     TEXT               NOT NULL, " +
                    "'Building No./Name' TEXT               NOT NULL, " +
                    "'Max Residents'     INT                NOT NULL," +
                    "'Boy/Girl'         TEXT               NOT NULL)";
            stmt.executeUpdate(sql);
            MainController.c.commit();
            for (Room room: roomsObservableList) {
                String sql2 = "INSERT INTO Rooms ('Room No./Name','Building
No./Name','Max Residents', 'Boy/Girl') " +
                        "VALUES ('" + room.getRoom() + "'" +  "," + "'" +
room.getBuilding() + "'" + "," + room.getMaxResidents() +
                        ",'" + room.getSexRoom() + "');";
                stmt.executeUpdate(sql2);
            }
            stmt.close();
            MainController.c.commit();

        } catch (Exception e) {
            System.err.println(  e.getClass().getName()  +  ":  "  +
e.getMessage() );
        }
    }

    private void createStudentColumns() {
        try {
            Statement stmt = MainController.c.createStatement();

            ResultSet rs = stmt.executeQuery("SELECT \"Max Residents\" FROM
Rooms;");
            int maxRoomCapacity = 0 ;
            while (rs.next()) {
                int current = rs.getInt(1);
                if (current > maxRoomCapacity)
                    maxRoomCapacity = current;
            }
            rs.close();
            for (int i = 0; i < maxRoomCapacity; i++) {
                stmt.executeUpdate("ALTER TABLE Rooms ADD COLUMN 'Student " +
(i+1) + "' INTEGER;");
                MainController.c.commit();
            }
            stmt.close();
            MainController.c.commit();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Room addButtonClicked(TextField roomTextField, TextField
buildingTextField,   TextField   maxResidentsTextField,   ComboBox<String>
```

```
sexComboBox,   ObservableList<Room>   roomsObservableList,   TableView<Room>
roomTableView) {
        Room addRoom = new Room();
        if (roomTextField.getText().isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error Dialog");
            alert.setHeaderText("There is an Error!");
            alert.setContentText("Please Enter the Room No./Name!");
            alert.showAndWait();
            roomTextField.requestFocus();
            return null;
        }
        if (buildingTextField.getText().isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error Dialog");
            alert.setHeaderText("There is an Error!");
            alert.setContentText("Please Enter the Building No./Name!");
            alert.showAndWait();
            buildingTextField.requestFocus();
            return null;
        }
        addRoom.setRoom(roomTextField.getText());
        addRoom.setBuilding(buildingTextField.getText());
        if          (!AddOrDeleteRoomController.contains(roomsObservableList,
addRoom)) {
            try {
                int                        maxResidents                        =
Integer.parseInt(maxResidentsTextField.getText());
                addRoom.setMaxResidents(maxResidents);
            } catch (NumberFormatException e) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("There is an Error!");
                alert.setContentText("Please   Enter   an   INTEGER   for   Max
Residents");
                alert.showAndWait();
                maxResidentsTextField.requestFocus();
                return null;
            }
            try {
                sexComboBox.getValue().isEmpty();
                addRoom.setSexRoom(sexComboBox.getValue());
            } catch (Exception e) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("There is an Error!");
                alert.setContentText("Please choose boy/girl's dormitory");
                alert.showAndWait();
                sexComboBox.requestFocus();
                return null;
            }
            addRoom.setId(roomsObservableList.size()+1);
            roomsObservableList.add(addRoom);
            roomTableView.setItems(roomsObservableList);
            roomTextField.clear();
            roomTextField.requestFocus();
            return addRoom;
        } else {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error Dialog");
            alert.setHeaderText("An Error has Occurred!");
            alert.setContentText("This room is already added in the scheme!");
            alert.showAndWait();
            return null;
        }
    }

    public  static  ObservableList<Room>  deleteButtonClicked(TableView<Room>
roomTableView, ObservableList<Room> roomsObservableList) {
        if (roomTableView.getSelectionModel().isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Information Dialog");
            alert.setHeaderText(null);
            alert.setContentText("Please Choose a Row to Delete!");
            alert.showAndWait();
        } else {
            int                          selectedIndex                          =
roomTableView.getSelectionModel().getSelectedIndex();
            for (int i = selectedIndex + 1; i < roomsObservableList.size();
i++) {
                roomsObservableList.get(i).setId(i);
            }
            roomsObservableList.remove(selectedIndex);
            roomTableView.setItems(roomsObservableList);
        }
        return roomsObservableList;
    }
}
```

```
package controllers.newFile;

import com.sun.tools.javac.Main;
import functional.AutoCompleteComboBox;
import functional.HandleButton;
import functional.Student;
import controllers.main.MainController;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;

import java.io.*;
import java.net.URISyntaxException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;
import java.util.ResourceBundle;

public class StudentConfig2Controller implements Initializable {

    private Map<String,String> countryToCountryCode = new HashMap<>();
    private Map<String,String> continentCodeToContinent = new HashMap<>();

    public void setDeleteDB(boolean deleteDB) {
        this.deleteDB = deleteDB;
    }

    private boolean deleteDB = true;



    private      ObservableList<Student>      studentsObservableList      =
FXCollections.observableArrayList();



    @FXML
    private TableView<Student> studentTableView;
    @FXML
    private TableColumn<Student, Integer> idColumn;

    @FXML
    private TableColumn<Student, String> givenNameColumn;

    @FXML
    private TableColumn<Student, String> familyNameColumn;

    @FXML
    private TableColumn<Student, Integer> yearColumn;

    @FXML
    private TableColumn<Student, String> sexColumn;

    @FXML
    private TableColumn<Student, ComboBox<String>> nationalityColumn;

    @FXML
    private TableColumn<Student, String> continentColumn;

    @FXML
    private Button finishButton;

    public Button getCancelButton() {
        return cancelButton;
    }

    @FXML
    private Button cancelButton;

    @FXML
    void cancelClick(ActionEvent event) {
        if (deleteDB) {
            DirectoryController.deleteDB(DirectoryController.oldFileName,
DirectoryController.oldDirectory);
            HandleButton button = new HandleButton();
            button.handleCancelButton(cancelButton);
        }
    }

    @FXML
    void finishClick(ActionEvent event) throws IOException {
        saveAndSwitchScene();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        Locale.setDefault(Locale.US);
        for (String countryCode : Locale.getISOCountries()) {
            Locale locale = new Locale("", countryCode);
            countryToCountryCode.put(locale.getDisplayCountry(),
countryCode.toUpperCase());
        }
        continentCodeToContinent.put("AS", "Asia");
        continentCodeToContinent.put("EU", "Europe");
        continentCodeToContinent.put("NA", "North America");
        continentCodeToContinent.put("AF", "Africa");
        continentCodeToContinent.put("AN", "Antarctica");
        continentCodeToContinent.put("SA", "South America");
        continentCodeToContinent.put("OC", "Oceania");
        yearColumn.setStyle("-fx-alignment: CENTER;");
        idColumn.setStyle("-fx-alignment: CENTER;");
        sexColumn.setStyle("-fx-alignment: CENTER;");
        nationalityColumn.setStyle("-fx-alignment: CENTER;");
        continentColumn.setStyle("-fx-alignment: CENTER;");
        idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));
        givenNameColumn.setCellValueFactory(new
PropertyValueFactory<>("givenName"));
        familyNameColumn.setCellValueFactory(new
PropertyValueFactory<>("familyName"));
        yearColumn.setCellValueFactory(new PropertyValueFactory<>("year"));
        sexColumn.setCellValueFactory(new PropertyValueFactory<>("sex"));
        nationalityColumn.setCellValueFactory(new
PropertyValueFactory<>("countryCB"));
        continentColumn.setCellValueFactory(new
PropertyValueFactory<>("continent"));
        populateTableView(1);
        populateTableView(2);
        for (Student student: studentsObservableList) {
            student.getCountryCB().setOnHidden(e  ->  showContinent(student,
countryToCountryCode, continentCodeToContinent, studentTableView));
        }
        studentTableView.setItems(studentsObservableList);
    }


    public  static  void  showContinent(Student  student,  Map<String,  String>
countryToCountryCode,    Map<String,    String>    continentCodeToContinent,
TableView<Student> studentTableView) {
        try {
            String                          countryCode                          =
countryToCountryCode.get(student.getCountryCB().getValue());
            InputStream                            in                            =
ClassLoader.getSystemClassLoader().getResourceAsStream("country_continent.c
sv");
            InputStreamReader isr = new InputStreamReader(in);
            BufferedReader br = new BufferedReader(isr);
            while (br.ready()) {
                String[] line = br.readLine().split(",");
                if (line[0].equals(countryCode)) {

student.setContinent(continentCodeToContinent.get(line[1]));
                    break;
                }
            }
            studentTableView.refresh();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void populateTableView(int year) {
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet  rs  =  stmt.executeQuery("SELECT  *  FROM  Students  WHERE
Year = " + year + ";");
            while (rs.next()) {
                Student student = new Student();
                student.setId(rs.getInt("Id"));
                student.setGivenName(rs.getString("GivenName"));
                student.setFamilyName(rs.getString("FamilyName"));
                student.setYear(year);
                student.setSex(rs.getString("Sex"));
                student.setCountryValue(rs.getString("Country"));
                student.setContinent(rs.getString("Continent"));
                studentsObservableList.add(student);
            }
            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public String getText(ComboBox<String> comboBox) {
        return AutoCompleteComboBox.getComboBoxValue(comboBox);
    }

    public void saveAndSwitchScene() {
        try {
            for (Student student: studentsObservableList) {
                getText(student.getCountryCB()).isEmpty();
            }
            try {
                Statement stmt = MainController.c.createStatement();
                int id = 1;
                for (Student student: studentsObservableList) {
                    String country = getText(student.getCountryCB());
                    String sql = "UPDATE Students SET Country = '"+ country
+"', Continent = \"" + student.getContinent() + "\" WHERE Id = "+ id +";";
                    stmt.executeUpdate(sql);
                    MainController.c.commit();
                    id++;
                }
                HandleButton button = new HandleButton();
                button.handleCancelButton(cancelButton);
            } catch (Exception e) {
                e.printStackTrace();
            }
        } catch (RuntimeException e) {
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Warning");
            alert.setHeaderText(null);
            alert.setContentText("Please fill in all details of students");
            alert.showAndWait();
        }
    }
}
```

```
package controllers.newFile;

import functional.HandleButton;
import controllers.main.MainController;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Statement;

public class StudentConfigController {

    private int id = 1;

    private boolean isUpload1Clicked = false;
    private boolean isUpload2Clicked = false;

    @FXML
    private Button previousButton;

    @FXML
    private Button cancelButton;

    @FXML
    private Button nextButton;

    @FXML
    void cancelClick(ActionEvent event) throws IOException {
        DirectoryController.deleteDB(DirectoryController.oldFileName,
DirectoryController.oldDirectory);
        HandleButton button = new HandleButton();
        button.handleCancelButton(cancelButton);
    }

    @FXML
    void previousClick(ActionEvent event) throws IOException {
        HandleButton button = new HandleButton();
        button.handlePreviousButton(previousButton,
"/fxmls/newFile/RoomConfig.fxml");
    }

    @FXML
    void nextClick(ActionEvent event) throws IOException {
        HandleButton button = new HandleButton();
        button.handleNextButton(nextButton,
"/fxmls/newFile/StudentConfig2.fxml");
    }
    @FXML
    void year1UploadButtonClick(ActionEvent event) {
       upload(1);
    }
    @FXML
    void year2UploadButtonClick(ActionEvent event) {
        upload(2);
    }
    public void upload(int year) {
        Stage mainStage = null;
        final FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().addAll(new
FileChooser.ExtensionFilter("CSV Files", "*.csv"));
        File selectedFile = fileChooser.showOpenDialog(mainStage);
        if (selectedFile != null) {
            try {
                writeToDB(selectedFile, year, this.id);
                this.id++;
                if (year == 1) isUpload1Clicked = true;
                if (year == 2) isUpload2Clicked = true;
                if (isUpload1Clicked && isUpload2Clicked) {
                    nextButton.setDisable(false);
                }
                Alert alert = new Alert(Alert.AlertType.INFORMATION);
                alert.setTitle("Information Dialog");
                alert.setHeaderText(null);
                alert.setContentText("The   file   has   been   successfully
uploaded!");
                alert.showAndWait();
            } catch (Exception e) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Dialog");
                alert.setHeaderText("An Error Occurred!");
                alert.setContentText("Please make sure the format of the CSV
file and upload again");
            }
        }
    }

    public static void writeToDB(File selectedFile, int year, int id) {
        try {
            Statement stmt = MainController.c.createStatement();
            if (id == 1) {
                String sql2 = "CREATE TABLE IF NOT EXISTS Students" +
                        "(Id INTEGER PRIMARY KEY   AUTOINCREMENT," +
                        " GivenName      TEXT      NOT NULL, " +
                        " FamilyName     TEXT      NOT NULL," +
                        " Sex           TEXT," +
                        " Country        TEXT," +
                        " Continent      TEXT," +
                        " Year           INTEGER   NOT NULL);";
                stmt.executeUpdate(sql2);
                MainController.c.commit();
                stmt.executeUpdate("DELETE FROM Students WHERE \"Year\" = " +
year + ";");
                MainController.c.commit();
            }
            BufferedReader      br      =      new      BufferedReader(new
FileReader(selectedFile));
            while (br.ready()) {
                String[] record = br.readLine().split(",");
                if (record[2].equals("m")) record[2] = "male";
                if (record[2].equals("f")) record[2] = "female";
                String sql3 = "INSERT INTO Students (GivenName, FamilyName,
'Year', 'Sex') VALUES" +
                        "('" + record[0] + "','" + record[1] + "'," + year +
", '" + record[2] + "');";
                stmt.executeUpdate(sql3);
                MainController.c.commit();
            }
            stmt.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
package controllers.view;

import controllers.main.MainController;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.TextField;
import javafx.scene.layout.AnchorPane;

import java.net.URL;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ResourceBundle;

public class RoomController implements Initializable {
    @FXML
    private AnchorPane pane;

    @FXML
    private TextField totalRoomsTextField;

    @FXML
    private TextField girlRoomsTextField;

    @FXML
    private TextField boyRoomsTextField;

    @FXML
    private TextField girlBedsTextField;

    @FXML
    private TextField boyBedsTextField;


    @FXML
    private TextField totalBedsTextField;


    @Override
    public void initialize(URL location, ResourceBundle resources) {
        int numRows = 0;
        int numBoyRows = 0;
        int numGirlRows = 0;
        int numBoyBeds = 0;
        int numGirlBeds = 0;
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms;");
            while (rs.next()) {
                numRows++;
                if (rs.getString(5).equals("Boy")) {
                    numBoyRows++;
                    numBoyBeds += rs.getInt(4);
                }
                if (rs.getString(5).equals("Girl")) {
                    numGirlRows++;
                    numGirlBeds += rs.getInt(4);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        totalBedsTextField.setText((numBoyBeds + numGirlBeds) + "");
        totalRoomsTextField.setText(numRows + "");
        boyBedsTextField.setText(numBoyBeds + "");
        boyRoomsTextField.setText(numBoyRows + "");
        girlBedsTextField.setText(numGirlBeds + "");
        girlRoomsTextField.setText(numGirlRows + "");
    }
}
```

```java
package controllers.view;

import controllers.main.MainController;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.TextField;
import javafx.scene.layout.AnchorPane;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ResourceBundle;

public class StudentController implements Initializable {

    @FXML
    private AnchorPane pane;
    @FXML
    private TextField totalNumTextField;
    @FXML
    private TextField girlNumTextField;
    @FXML
    private TextField boyNumTextField;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        int totalNum = 0;
        int boyNum = 0;
        int girlNum = 0;
        try {
            Statement stmt = MainController.c.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Students;");
            while (rs.next()) {
                totalNum++;
                if (rs.getString(4).equals("male")) boyNum++;
                if (rs.getString(4).equals("female")) girlNum++;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        totalNumTextField.setText(totalNum + "");
        boyNumTextField.setText(boyNum + "");
        girlNumTextField.setText(girlNum + "");
    }
}
```