

Criterion C: Development

Table of Contents:

Structure of the Program	2
Packages and Libraries Used	3
Complex Functionalities.....	4
<i>Map a Country to its Continent</i>	<i>4</i>
<i>Integer Validation with Try & Catch</i>	<i>6</i>
<i>Sort List<Room> Using Self-defined Comparator</i>	<i>6</i>
<i>Context Menu Disable Validation</i>	<i>7</i>
<i>Genetic Algorithm (GA)</i>	<i>9</i>
<i>Run GA in the Background Thread While Showing a Progress Indicator</i>	<i>13</i>

Structure of the Program

The program was developed with Java and the JavaFX library. Therefore, each FXML file (GUI) comes with one controller class that controls the action of that specific window. All windows are linked together with buttons that are programmed to load other scenes that evoke their specific controller classes. I have chosen JavaFX over Swing to make the GUIs of the software because the structural advantage that JavaFX provides: it allows me to create GUIs that are separated from the functionalities of each interface, so that the structure are more clear and easier to maintain.

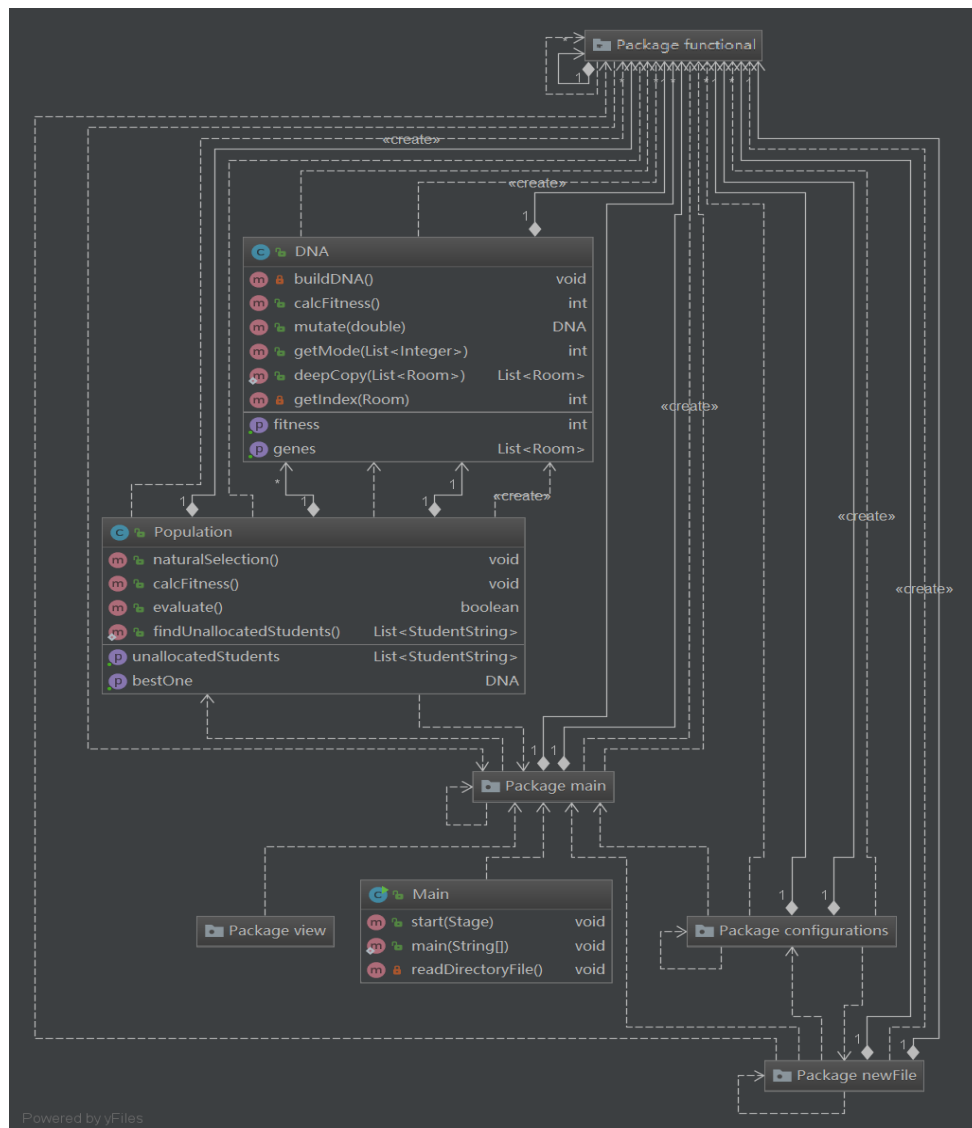


Figure 1. UML Diagram of the Project (Generated by IntelliJ IDEA)

In Figure 1, package diagrams are used instead of class diagrams because the class diagrams are too complicated and unreadable.

Packages and Libraries Used

Internal Libraries:¹

- **java.io.***
 - Provides for system input and output through data streams, serialization and the file system.
- **javafx.***
 - Provides all the functionalities for JavaFX to operate.
- **java.util.**
 - ◆ **Locale**
 - Represents a specific geographical, political, or cultural region.
 - ◆ **Map**
 - An object that maps keys to values.
 - ◆ **HashMap**
 - Hash table based implementation of the Map interface
 - ◆ **List**
 - Provides An ordered collection (also known as a sequence).
 - ◆ **ArrayList**
 - Resizable-array implementation of the List interface.
 - ◆ **Random**
 - An instance of this class is used to generate a stream of pseudorandom numbers.
- **java.sql.***
 - Provides the API for accessing and processing data stored in a data source.

External Libraries:

- **com.jfoenix.***²
 - A javaFX material design library.
- **org.apache.poi.***³
 - A Java API to create, modify, and display Microsoft Documents.
- **AutoCompleteComboBox**⁴ (Viccari)
 - Provides autocomplete feature to Combo Boxes.

¹ <https://docs.oracle.com/javase/8/>

² <http://www.jfoenix.com/>

³ <https://poi.apache.org/index.html>

⁴ <https://stackoverflow.com/questions/19924852/autocomplete-combobox-in-javafx>

Complex Functionalities

1. Map a Country to its Continent

In the software, in the configuration process of students, the user is required to choose the country where each student is from from Combo Boxes. The list of countries in Combo Boxes are initialized using the java.util.Locale class.

```
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);
```

Here countryCB is a property of ComboBox type in my Student Class.

Given Name	Family Name	Sex	Country	Continent	Allocated
Aasif	A	male	India	Asia	✓
Abar	A	female	Kenya	Africa	✓
Abdul	A	male	Bangladesh	Asia	✓
Adel	A	male	Libya	Africa	✓
Adelaide	A	female	France	Europe	✓
Ahaan	A	male	India	Asia	✓
Ahmed	A	male	Andorra		
Akop	A	male	United Arab Emirates		
Akshita	A	female	Afghanistan		
Aldo	A	male	Antigua & Barbuda		
Alexander	A	male	Anguilla		
			Albania		
			Armenia		
			Angola		
			Antarctica		

When choosing the country of a student from the combo box, his/her continent is expected to be changed automatically according to the change in the country.

To achieve this goal, I have imported CSV file in which the first data in every record is the 2-digit Country Code, the second data is its 2-digit continent code. ("ISO 3166 Country Codes With Associated Continent « Maxmind Developer Site")

1	AD,EU
2	AE,AS
3	AF,AS
4	AG,NA
5	AI,NA
6	AL,EU
7	AM,AS
8	AN,NA
9	AO,AF
10	AP,AS

Therefore, I have implemented two HashMap

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

for (String countryCode : Locale.getISOCountries()) {
    Locale locale = new Locale("", countryCode);
    countryToCountryCode.put(locale.getDisplayCountry(), countryCode.toUpperCase());
}
countryCodeToContinentCode.put("AS", "Asia");
countryCodeToContinentCode.put("EU", "Europe");
countryCodeToContinentCode.put("NA", "North America");
countryCodeToContinentCode.put("AF", "Africa");
countryCodeToContinentCode.put("AN", "Antarctica");
countryCodeToContinentCode.put("SA", "South America");
countryCodeToContinentCode.put("OC", "Oceania");
```

The “countryToCountryCode” maps the full name of the countries to their 2-digit country codes, then, the “continentCodeToContinent” maps the 2-digit continent code to the full name of the continent.

In order to get the corresponding continent code of the country, a search in the CSV file is needed. I have used BufferedReader to read the file line by line to achieve this.

```
public static void showContinent(Student student, Map<String,String> countryToCountryCode, Map
String countryCode = countryToCountryCode.get(student.getCountryCB().getValue());
File file = new File(pathname "src/data/country_continent.csv");
try {
    BufferedReader br = new BufferedReader(new FileReader(file));
    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();
}
}
```

This “showContinent” method is called inside “setOnHidden()” method of the Country ComboBox, so whenever the combo box is closed, this method will be called. By changing the “continent” property of the student object and refresh the table, the new continent value will be updated.

I declared this method as “public static” because in other parts of my software this functionality is also needed. In this way I can reuse my code.

2. Integer Validation with Try & Catch

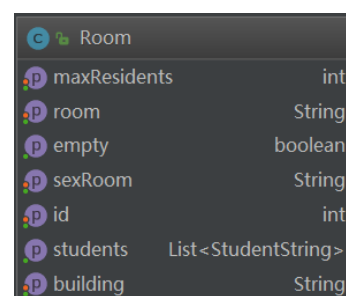
In the room configuration process, I need the inputs of the maximum capacity of each room from the user. In order for the program to running correctly, I need to validate whether the inputs from the user are integer values. Therefore, I implemented a try & catch block to achieve this functionality.

```
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;
}
```

I have used the property of “Integer.parseInt()” that if the String that passed in to this method is not a parsable integer, it will throw an “NumberFormatException”. Therefore, if the user entered non-parsable value, it will go to the catch block and display the Error message, asking for re-enter from the user.

3. Sort List<Room> Using Self-defined Comparator

Room is a class that has the following properties:



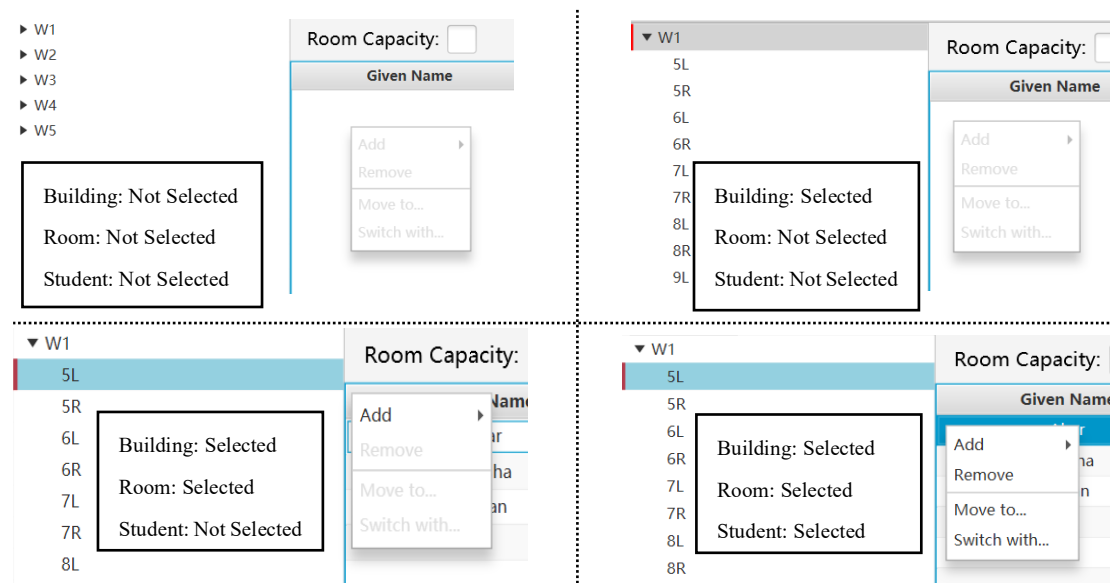
Room	
maxResidents	int
room	String
empty	boolean
sexRoom	String
id	int
students	List<StudentString>
building	String

Upon opening the main window, the tree view of rooms on the left needs to be initialized using Room objects that are created using the data from the database. However, the record in the database may not be in order because of the user inputs or adding or deleting operation. Therefore, a comparator is needed to compare the room properties and building properties of each room object to sort them.

```
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);
    }
}
```

In this roomComparator I designed, the building names are being compared lexicographically first, and if two rooms have the same building name, then their room names are compared. Rooms are first compared with their length first, if they have the same length, then lexicographically. This is because in the case of rooms like 3A, 11A, if compared lexicographically, 11A would be in front of 3A.

4. Context Menu Disable Validation



In the main window, by right clicking, a context menu containing “Add, Remove, Move To, Switch With” four functionalities will be display. However, if the user is not selecting any room in the tree view on the left, then all of the options should be disabled to prevent error. And if the user has selected a room, but not any students, then only “Add” option should be enabled. Elsewise, if the user has selected a student in a room, then all four options should be enabled. To achieve this goal, I have created two validation method that return boolean values to bind with “Add” and “Remove, Move To, Switch With”.

```
addMenu.disableProperty().bind(Bindings.createBooleanBinding(this::treeViewValidation));
```

```
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);
}
```

treeViewValidation() is bound with the disable property of “Add”. Therefore, as long as the user dose not choose the any tab or the user only selected the building tab, the “Add” function will be disabled.

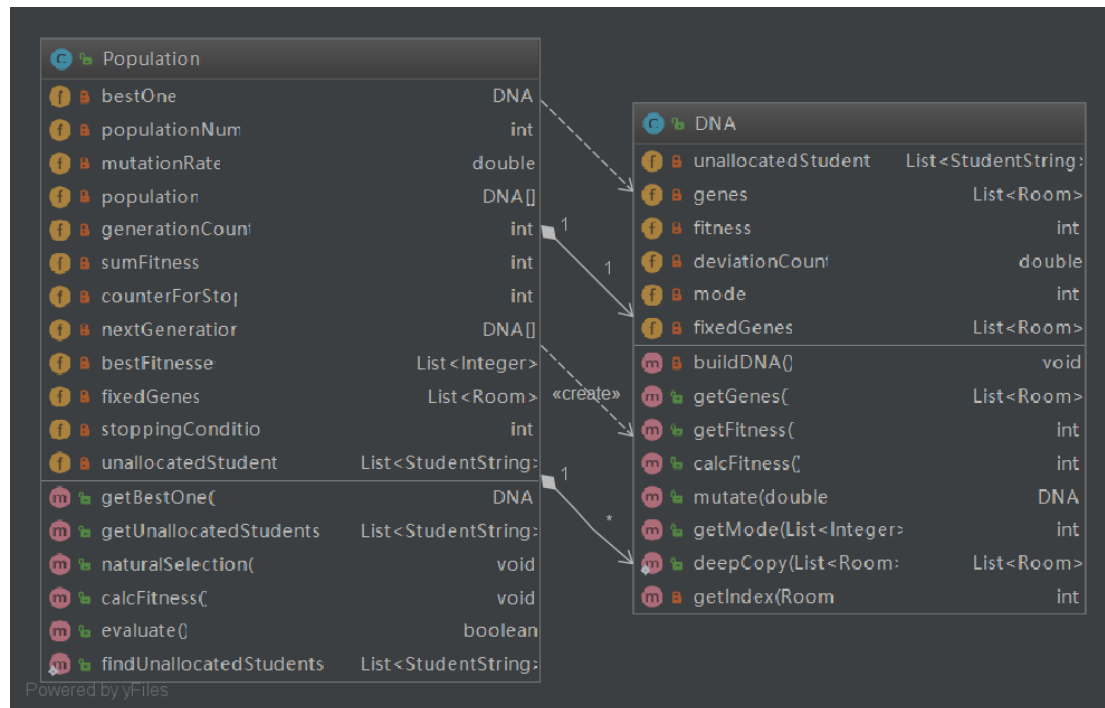
```
removeMenuItem.disableProperty().bind(Bindings.createBooleanBinding(this::tableViewValidation));
moveMenuItem.disableProperty().bind(Bindings.createBooleanBinding(this::tableViewValidation));
switchMenuItem.disableProperty().bind(Bindings.createBooleanBinding(this::tableViewValidation));
```

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

tableViewValidation() is bound with the disable properties of “Remove”, “Move To”, and “Switch With”, it requires the user to choose a student in the TableView to enable these functionalities.

5. Genetic Algorithm (GA)

The core of this software is the genetic algorithm to allocated the students, optimizing the diversity in all rooms. The genetic algorithm is a met-heuristic algorithm that is inspired by Charles Darwin's theory of natural selection. I have designed two classes for the operation of GA – Population class and DNA class. The UML diagram between two classes are shown below.



When the genetic algorithm starts, a Population object will be instantiated, the constructor in Population class will call the constructor in DNA class repetitively for 1000 times, and each time, a random room allocation based on the “fixedGenes” (List of Rooms that contains the allocated students that cannot be altered) and unallocatedStudent (List of unallocated students) will be created. This allocation is stored in the “List<Room> genes” in their respective DNA object. And these 1000 DNA objects are stored in the “DNA[] population” in the Population Object.

```

public Population(int populationNum, double mutationRate, int stoppingCondition, List<Room> fixedGenes) {
    this.unallocatedStudents = findUnallocatedStudents();
    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);
    }
}

```

```

DNA(List<StudentString> unallocatedStudents, List<Room> fixedGenes) {
    this.unallocatedStudents = unallocatedStudents;
    this.fixedGenes = deepCopy(fixedGenes);
    this.genes = deepCopy(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);
    }
}

```

After the instantiation of the Population object, the calcFitness() method in the Population object will be called. In this method, calcFitness() method in DNA class will be called for each DNA in the DNA array. And their fitness values will be stored in their respective “fitness” variable. The sum of all fitness values for 1000 DNA objects are stored in the “sumFitness” variable in the Population object. Also, the best DNA which has the highest fitness value in this generation will be recorded as “bestOne”.

This is the calcFitness() method in the Population class.

```

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());
}

```

This is the calcFitness() method in the DNA class.

```
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;
}
```

Then, naturalSelection() method will be called, the “bestOne” will directly be passed into the next generation. For the rest of DNAs, using “Roulette Wheel Selection”⁵ algorithm based on the fitness of each DNA object, the next generation of the same size of population will be selected. Each selected DNA object will have 2% of chance to go through mutation process, which alters the allocation of students by a little, giving some varieties in the population.

⁵ https://en.wikipedia.org/wiki/Fitness_proportionate_selection

```

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, srcPos: 0, population, destPos: 0, populationNum);
    generationCount++;
    System.out.println("Generation " + generationCount);
    System.out.println("-----");
}

```

Roulette Wheel Selection

Then calcFitness() will be called to calculate fitness for the new Generation. After, evaluate() method will be called to check whether it should stop evolving and output the result. After testing, I have set the stopping condition to 1000 generations without any breakthrough in the highest fitness value of each generation. I have achieved this by using an integer list to store the highest fitness value in each generation.

```

private List<Integer> bestFitnesses = new ArrayList<>();

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;
}

```

The algorithm will run in loop until the stopping condition is met.

```

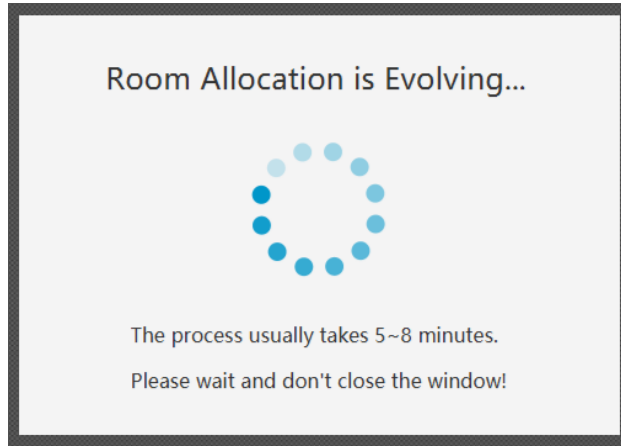
do {
    population.naturalSelection();
    population.calcFitness();
} while (population.evaluate());

```

Then the database will be updated with “List<Room> gene” in the DNA object with the highest fitness value.

6. Run GA in the Background Thread While Showing a Progress Indicator

I have designed a Progress Indicator window that will animate the spinning of a circle while the genetic algorithm is running.



Therefore, I have implemented the Service class⁶ in `javafx.concurrent`. A Service is a non-visual component encapsulating the information required to perform some work on one or more background threads. ("Service (Javafx 8)").

```
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();
```

WORD COUNT: 1267

⁶ <https://docs.oracle.com/javase/8/javafx/api/javafx/concurrent/Service.html>

Works Cited

"Home: Java Platform, Standard Edition (Java SE) 8 Release 8". Docs.Oracle.Com, 2018, <https://docs.oracle.com/javase/8/>. Accessed 22 Aug 2018.

Viccari, Mateus. "Autocomplete Combobox In Javafx". Stack Overflow, 2018, <https://stackoverflow.com/questions/19924852/autocomplete-combobox-in-javafx>. Accessed 22 Aug 2018.

"ISO 3166 Country Codes With Associated Continent « Maxmind Developer Site". Dev.Maxmind.Com, 2018, https://dev.maxmind.com/geoip/legacy/codes/country_continent/. Accessed 22 Aug 2018.

"Service (Javafx 8)". Docs.Oracle.Com, 2018, <https://docs.oracle.com/javase/8/javafx/api/javafx/concurrent/Service.html>. Accessed 22 Aug 2018.