

# COMP1406 - Assignment #8

(Due: Monday, March 20th @ 10 am)

In this assignment you will continue your work from Assignment 7 by adding more features that make use of **Alert** boxes, **Dialog** boxes and **Menus**.



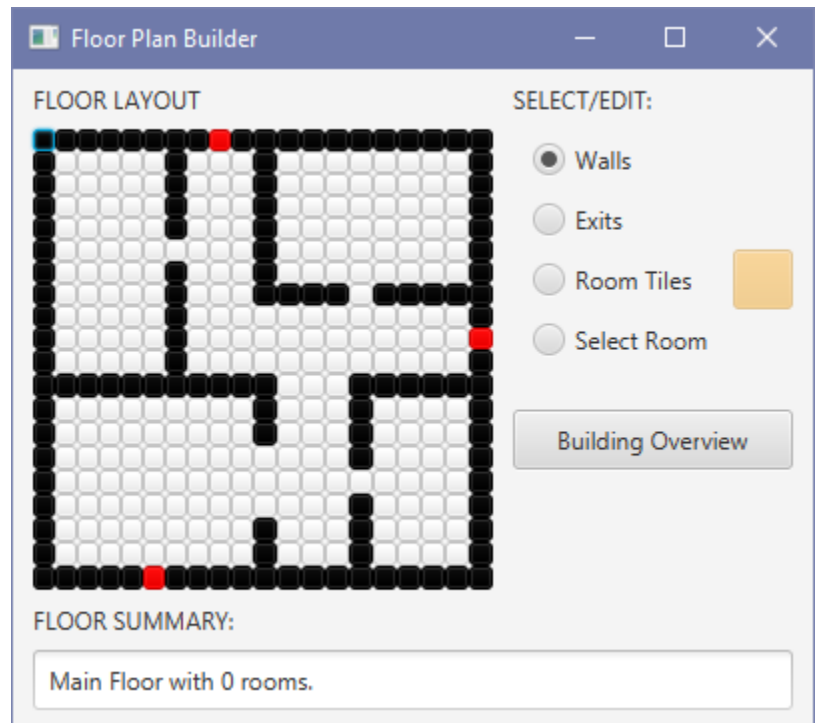
## (1) Making a More Interesting Building

To begin, download the updated model code for the **Exit**, **Room**, **FloorPlan** and **Building** classes that has been provided on the course CULearn page. The following four additional methods have been added to the **FloorPlan** class, which represent additional floors to a building:

```
public static FloorPlan floor2() { ... }  
public static FloorPlan floor3() { ... }  
public static FloorPlan floor4() { ... }  
public static FloorPlan floor5() { ... }
```

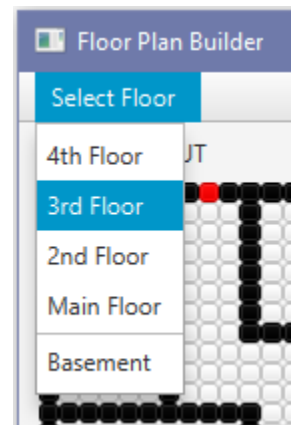
In addition, the **example()** method in the **Building** class has been altered to accommodate these additional floor plans.

You should be able to run your **FloorPlanApp** code from the previous assignment and see the user interface as shown below. Note that if you did not get your code working, please contact the TAs for help.



## (2) The MenuBar

Add a **MenuBar** to the **FloorBuilderApp** with a single menu named **Select Floor**. The menu should contain 5 **MenuItem**s, as well as a **SeparatorMenuItem** and should look exactly as shown here in the image. Notice that the names of the **FloorPlan** objects are to be shown in the menu (you MUST read these names from the **FloorPlan** objects, do not hard-code them in the menu code). Pay attention to the order of the floor plans as well. (Note that you will need to add about 20 pixels to your application's Scene height due to the added **MenuBar**).



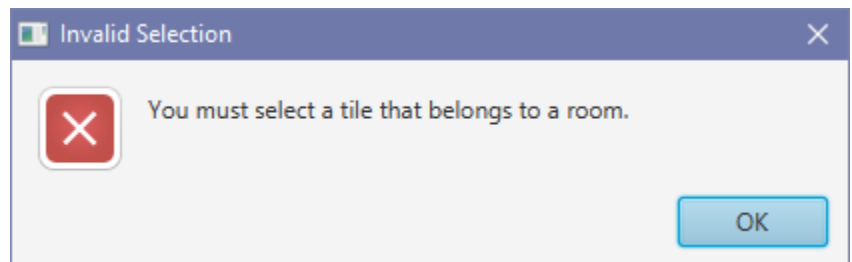
Add event handlers so that whenever the user selects a floor from the menu, the **currentFloor** variable is set accordingly and the view is updated to show that **FloorPlan** immediately. Test your code to make sure that you can see all 5 **FloorPlans**.

### (3) The RoomDetails Dialog

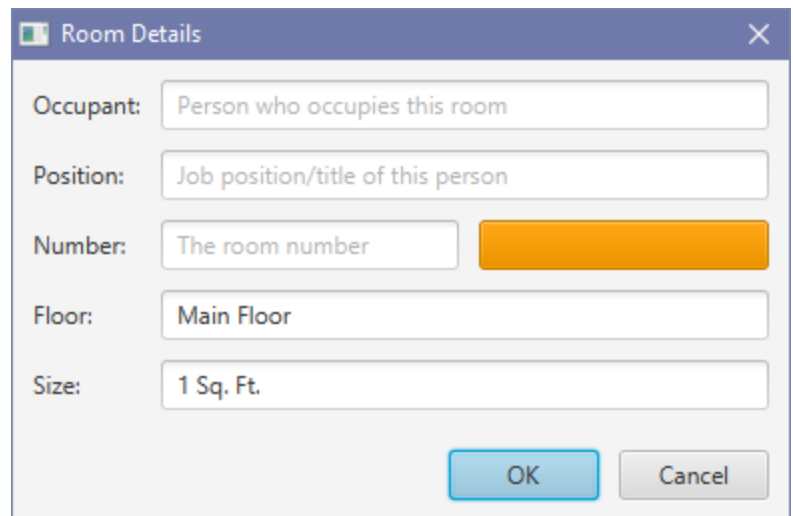
In the **Room** class, add the following instance variables and make public get/set methods for them as well:

```
private String    occupant;    // The name of the occupant of this room
private String    position;    // The position that the occupant of this room holds
private String    number;      // The room number (e.g., 102, 305B, etc.)
```

In the **FloorBuilderApp**, add code so that when the Select Room **RadioButton** is selected and the user then clicks on a grid location that does NOT correspond to a room tile ... then an **Alert** box (as shown here on the right) should appear. If the user selects **OK** on the above **Alert** box, then the application should go back to waiting for the user to select a room in order to get/set its information.

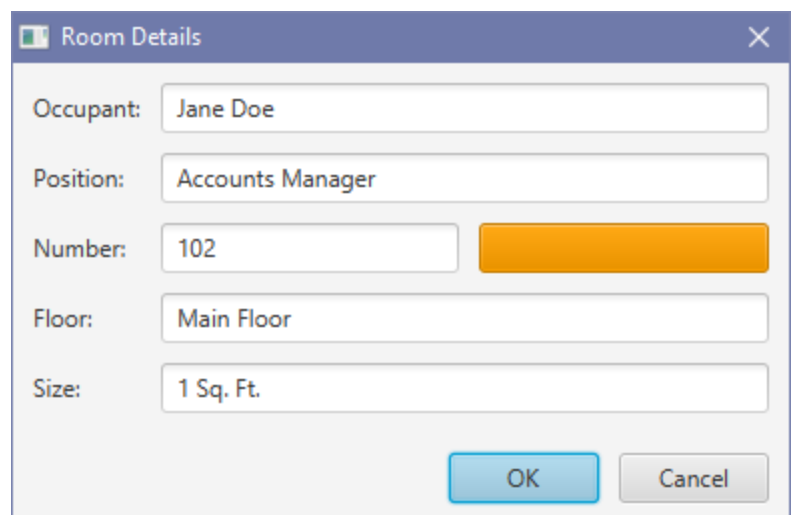


When in the Select Room mode, and the user successfully clicks on a tile that belongs to a **Room**, then the following **Dialog** box should appear (as shown on the right) allowing the user to view and edit information about that **Room**. The **Dialog** must be in a class called **RoomInfoDialog**.

A JavaFX Dialog box titled "Room Details". It contains several text input fields: "Occupant:" with placeholder text "Person who occupies this room", "Position:" with placeholder text "Job position/title of this person", "Number:" with placeholder text "The room number", "Floor:" with placeholder text "Main Floor", and "Size:" with placeholder text "1 Sq. Ft.". To the right of the "Number:" field is a solid orange rectangular button. At the bottom right, there are two buttons: a blue "OK" button and a grey "Cancel" button.

Take note of the specific "prompt text" that is used. Also, notice that the **Floor** and **Size** fields are non-editable and the color button is set to be non-focusTraversable (see JavaFX API).

The user may then enter data into the top three text fields, and it should look, for example, as shown here on the right:

A JavaFX Dialog box titled "Room Details", identical in layout to the previous one but with data entered. The "Occupant:" field contains "Jane Doe", the "Position:" field contains "Accounts Manager", and the "Number:" field contains "102". The "Floor:" field still shows "Main Floor" and the "Size:" field shows "1 Sq. Ft.". The orange button is still present to the right of the "Number:" field. The "OK" and "Cancel" buttons are at the bottom right.

When the user presses the **OK** button, the information for that room should be stored for the room that was clicked on, so that if the dialog box is re-opened, its contents should reflect these changes that you made.

However, if **CANCEL** is pressed, then none of the data entered by the user is saved into the room ... the data remains as it was before the dialog box was opened.

---

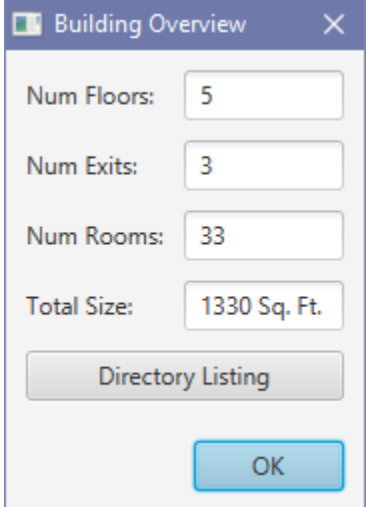
## (4) The BuildingDialog

Create a class called **BuildingDialog** that brings up the following dialog box when the user presses the **Building Overview** button from the main user interface:

The dialog box should show the number of floors in the building, as well as the current number of exits and rooms in the entire building. It should also show the total square footage of the building which is the number of non-wall floor tiles throughout the whole building. The text fields should not be editable. You can test your dialog's data by adding or removing rooms, exits and walls and make sure that these values change. When the user presses the **OK** button, the dialog box should close.

When the user presses the **Directory Listing** button, a **DirectoryDialog** should appear (see next part of assignment) while the **BuildingDialog** remains open.

---

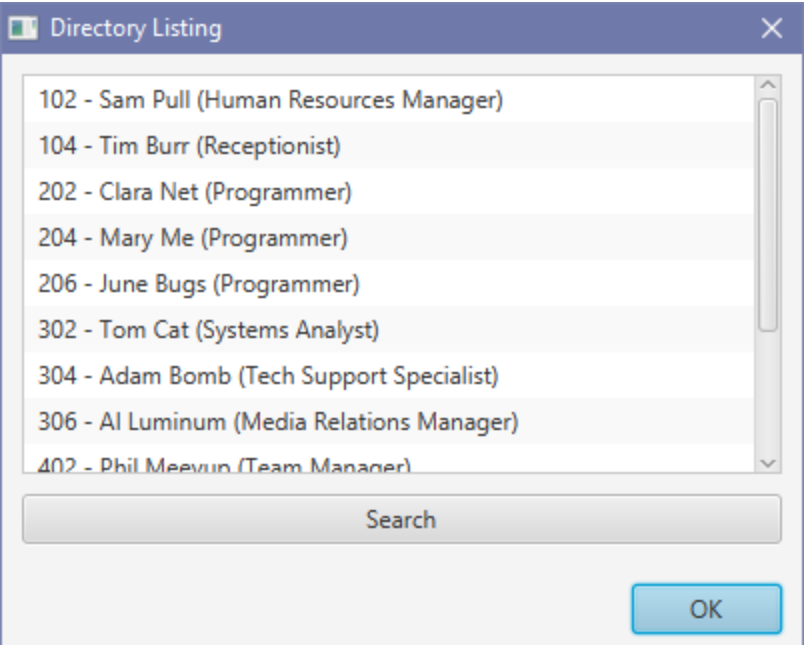


The screenshot shows a dialog box titled "Building Overview" with a close button (X) in the top right corner. It contains four non-editable text fields: "Num Floors:" with the value "5", "Num Exits:" with the value "3", "Num Rooms:" with the value "33", and "Total Size:" with the value "1330 Sq. Ft.". Below these fields is a button labeled "Directory Listing". At the bottom right is a blue button labeled "OK".

## (5) The DirectoryDialog

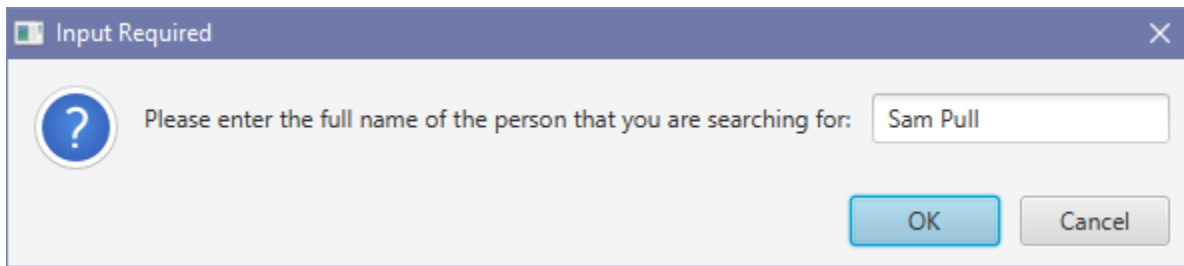
Create a class called **DirectoryDialog** that brings up the following dialog box when the user presses the **Directory Listing** button from the **BuildingDialog**.

The Dialog contains a **ListView<String>** that displays information about the rooms of the building. Each entry in the list should correspond to a **Room** in the building. Hence, if there are no rooms, then this list will be empty. Each entry should show the number of the room, the occupant of the room, and the position that that occupant holds in the company. The text must be formatted as shown in the list. Although the list above is sorted by room number, you DO NOT need to sort the list in any way. If the **OK** button is pressed, of the **Dialog** is closed with the **X**, then the program returns control to the **BuildingDialog**. (Note: When setting the preferred/min/max height of the **ListView**, do not use **Integer.MAX\_VALUE**, as it causes JavaFX to hang).

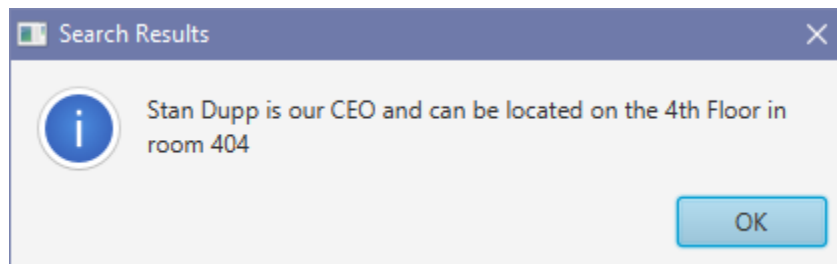


The screenshot shows a dialog box titled "Directory Listing" with a close button (X) in the top right corner. It contains a list view with the following entries: "102 - Sam Pull (Human Resources Manager)", "104 - Tim Burr (Receptionist)", "202 - Clara Net (Programmer)", "204 - Mary Me (Programmer)", "206 - June Bugs (Programmer)", "302 - Tom Cat (Systems Analyst)", "304 - Adam Bomb (Tech Support Specialist)", "306 - Al Luminum (Media Relations Manager)", and "402 - Phil Meewun (Team Manager)". Below the list is a button labeled "Search". At the bottom right is a blue button labeled "OK".

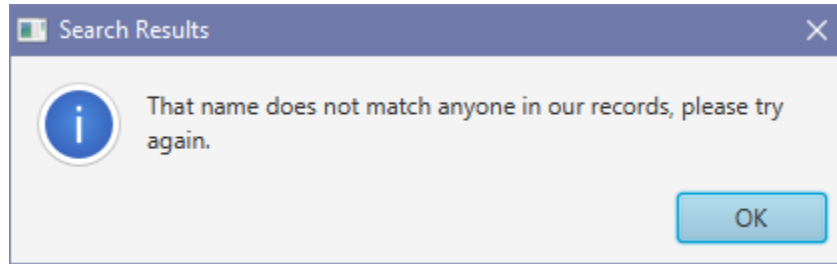
When the Search button is pressed, the following **TextInputDialog** should appear.



Once the user enters a person's name and presses **OK**, then your code should search the building for an occupant with that name and then display the following **Alert** dialog if found:



Note that the **Room** occupant's name is shown, as well as his/her position, the **FloorPlan** name and the **Room** number. You should follow the same sentence format. Of course, if CANCEL or X is pressed, then no **Alert** dialog should appear. If the person is unable to be found in the building, then the following **Alert** box should be shown:



---

### IMPORTANT SUBMISSION INSTRUCTIONS:

Submit your **ZIPPED IntelliJ project file** as you did during the first tutorial for assignment 0.

- YOU WILL LOSE MARKS IF YOU ATTEMPT TO USE ANY OTHER COMPRESSION FORMATS SUCH AS **.RAR**, **.ARC**, **.TGZ**, **.JAR**, **.PKG**, **.PZIP**.
  - If your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment **WELL BEFORE** it is due !
  - You **WILL** lose marks on this assignment if any of your files are missing. So, make sure that you hand in the correct files and version of your assignment. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.
-