48024 Applications Programming

Assignment 2

Topics: OO design, GUI, MVC, tables, lists
Objectives: This assignment supports objectives 3 - 5
Due date: 11:59pm Wednesday 19th of October 2016
Weight: 20%

1. Individual work

All work is individual. You may discuss ideas, approaches and problems, but you should write every line of code yourself except for code copied from the lecture notes, lecture code or lab code. More information about Academic Misconduct can be found at:
http://www.gsu.uts.edu.au/rules/student/section-16.html

2. Skeleton code

As a starting point for this assignment, you must use the skeleton code provided on PLATE (https://plate.it.uts.edu.au/) under Assessments / Assignment 2 / Skeleton Code.

Included in the skeleton code is a text file called `progress.txt` which must be filled in by you as you progress through the assignment (see “PLATE marking” and “Submission and Return” below).

3. Expected workload

The time to do the assignment to a distinction level (i.e. a mark between 75% to 84%) has been estimated at 15-20 hours for a student of average ability who has completed all the tutorial and lab exercises.

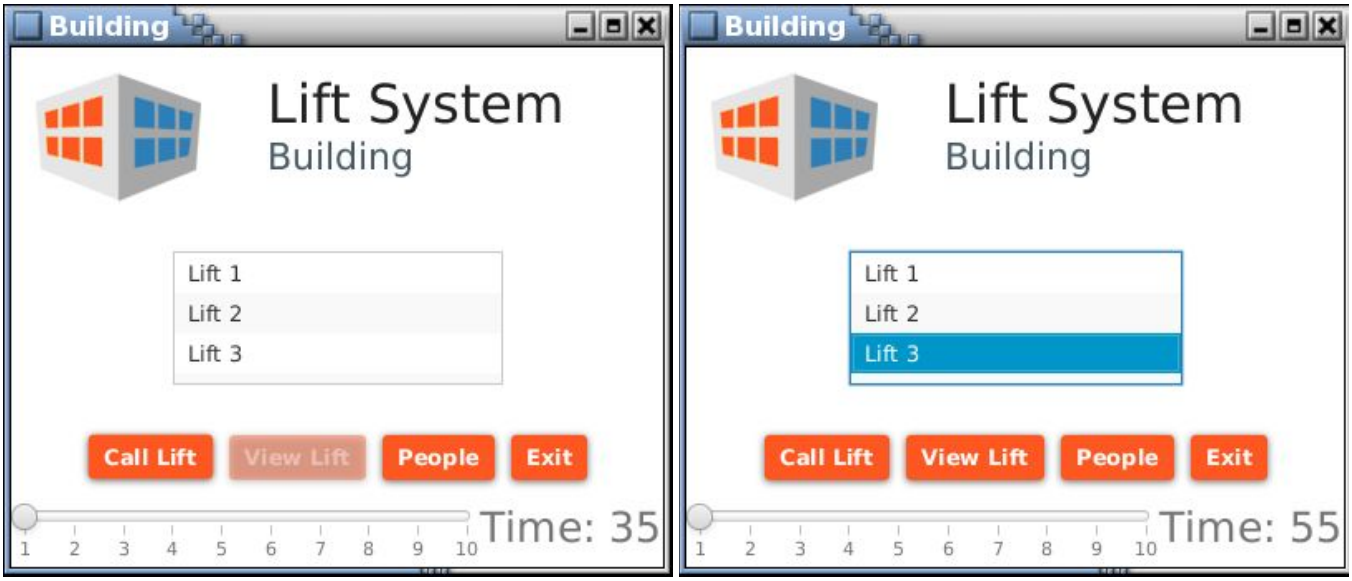
4. Specification

The lift system that you created in Assignment 1 has been very well received by the client, and they have requested that you develop a graphical user interface for the system.

The following subsections specify the required functionality of the new system.

Core Functionality

Building window (35%)



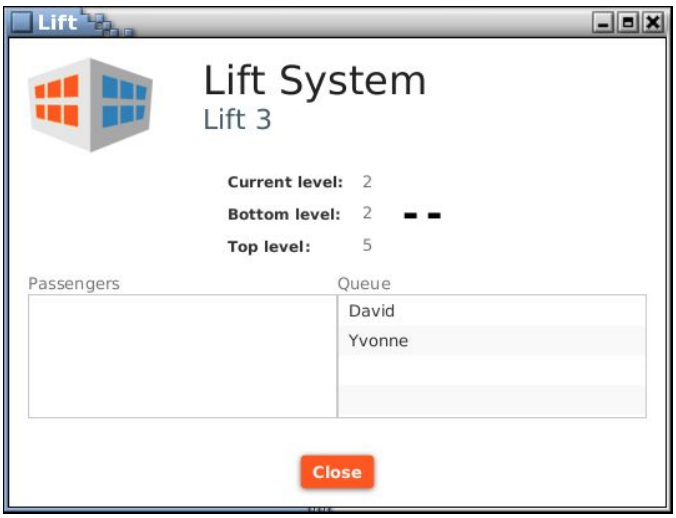
When the application is launched, the “Building” window appears showing a list of lifts, and some buttons to control and monitor the system. The lift operation works similarly to Assignment 1 via the operate() procedure except that the operate() procedure is automatically called by the system at a default rate of once every second. Below the buttons is a slider to change how many seconds to wait between each operate() call. To the right of the slider is a time display that indicates how many operate() calls have happened so far and reflects the “time” property inside the Building class. This time display updates in real time using the observer pattern. The buttons work as follows:

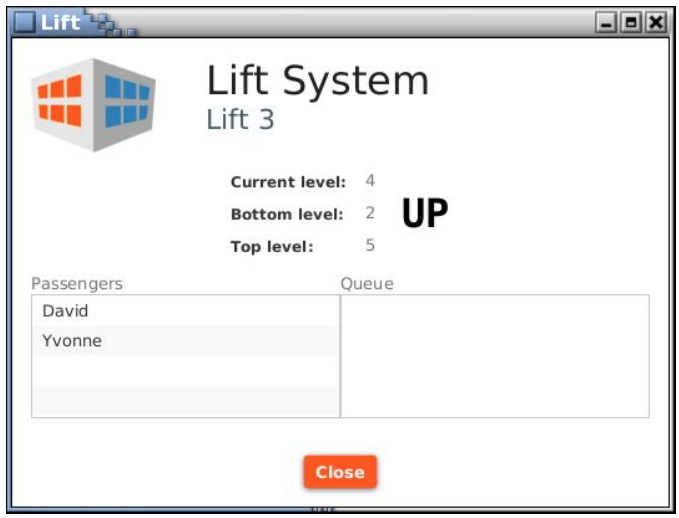
- The “Call Lift” button opens the “Call Lift” window (see below)
- The “View Lift” button opens the “Lift” window (see below). Note that the “View Lift” button is enabled only when a lift is selected in the list.
- The “People” button opens the “People” window (see below)
- The “Exit” button will stop the automatic operation of the lift, and then close this window.

For this assignment, you need to get the buttons working in this order: first, get the “Exit” button working, then the “View Lift” button, then the “Call Lift” button, then finally the “People” button. That also means that you need to get the “Lift” window working first before working on the “Call Lift” window, and then finally you can develop the “People” window.

Note: when opening any new window, the main “Building” window should still be open in the background.

Lift window (30%)



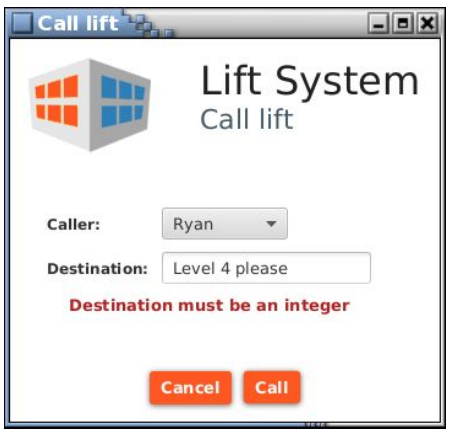


The “Lift” window shows the number, current level, bottom level, top level, current direction, passenger list and queue list of the selected lift. The direction should be either “UP”, “DOWN” or “--” (for stationary). Alternatively, show the direction as 1, -1 or 0 but for fewer marks (see “Marking Scheme” below). As the lift moves and picks up or drops off passengers, the details shown in this window are all instantly updated using the observer pattern. Clicking the “Close” button closes this window.

Call Lift window (20%)

Note: To implement this feature, you must set the `AUTO_CALL` constant in the `Building` class to `false`.

The “Call Lift” window shows a ComboBox for selecting the person who is calling the lift, and a TextField for entering their destination. Pressing the “Cancel” button closes this window. Normally, pressing the “Call” button calls a lift for the selected person for the selected destination, then closes the window. But if the user didn’t select a caller, or didn’t input a valid integer as the destination, or there was no suitable lift for the caller, then show an error and don’t close the window. These error cases are shown below:



People window (10%)

The "People" window shows a TableView of people with 5 columns: the person's ID, name, current level, destination and whether or not they are aboard. Each column value should be formatted as shown in the screenshot to the right using custom cell value factories with code like this:

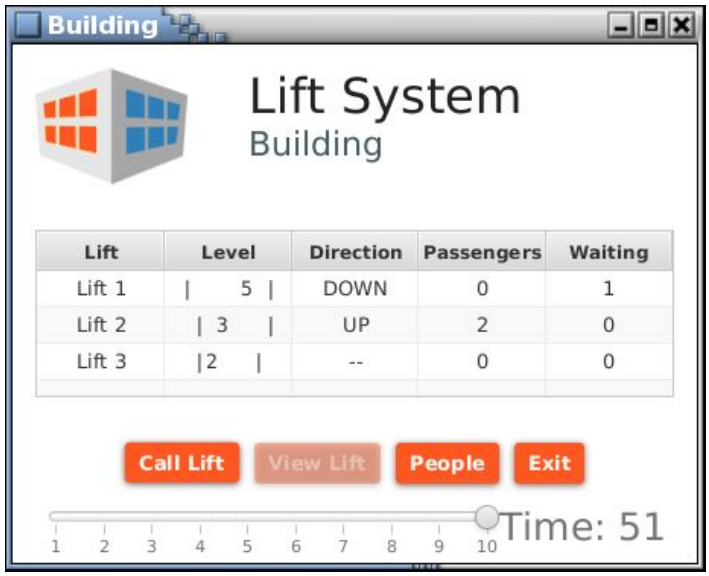
```
column.setCellValueFactory(cellData -> ... );
```

where `...` must be replaced by an observable value such as a property. Hint: The `Bindings` class (`import javafx.beans.binding.*`) can be used to produce more complex observable values. The documentation for this class is here:

https://docs.oracle.com/javase/8/javafx/api/javafx/beans/binding/Bindings.html

Clicking the "Close" button closes this window.

Advanced Functionality (5%)



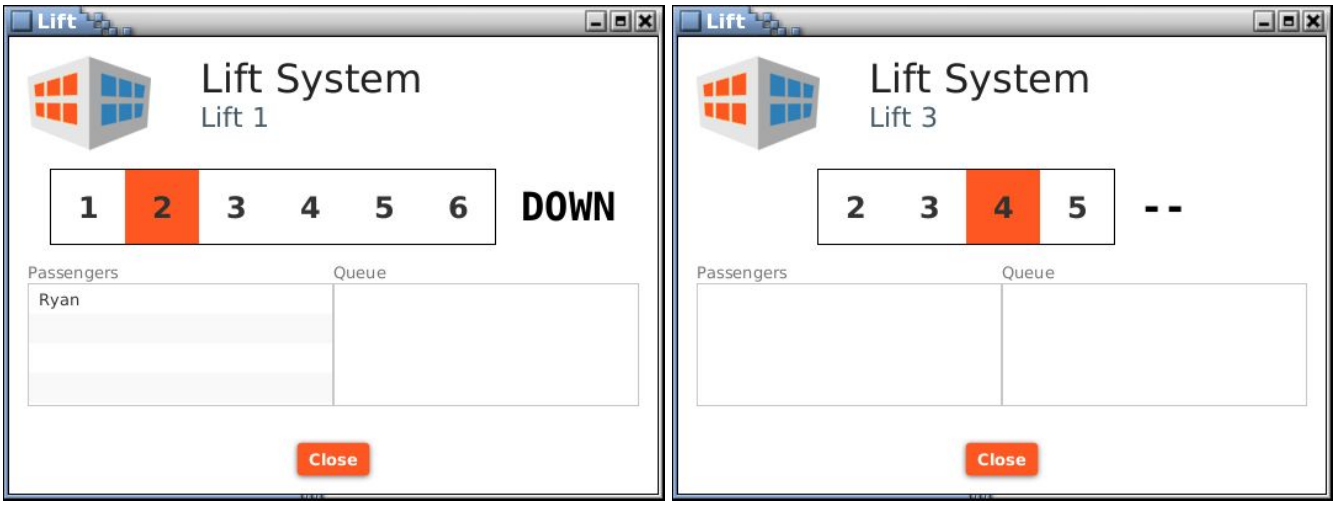
Modify the main "Building" window to display the list of lifts in a TableView with 5 columns: the lift number, level, current direction, number of passengers and number of people waiting in the queue. Each column value should be formatted as shown below, and the values displayed should be updated as the lift moves up and down, and picks up and drops off passengers, by using the observer pattern.

The Level indicator in the second column is a string with a single number between two vertical bars (i.e. '|'). The number indicates the current level that the lift is on, and it's position between the two bars indicates where this level is relative to the bottom and top floor of the lift. As the lift moves down, the number moves toward the left bar. As the lift moves up, the number moves toward the right bar. If the bottom level of the lift is N levels higher than the bottom level of the building, insert N leading spaces before the left bar. To make sure the same levels on different lifts line up, you also need to use the Monospaced font for this particular column (but NOT for any other columns). To implement this last feature of the assignment, you will be expected to do some research. For suggested material, read:
http://code.makery.ch/blog/javafx-8-tableview-cell-renderer/

Bonus Functionality (Extra 20%)

Implementing the bonus feature will allow you to potentially score up to 120% for Assignment 2. This equates to a potential extra 4 weighted marks for the subject. Any bonus marks earned will count toward your total mark for the subject (which itself will be capped at 100%).

Modify the “Lift” window to display a visual indication of the lift’s current level as follows:



This new level indicator must be implemented as a custom node which you define as a Java class, and then use from FXML like this:

```
<LevelView bottom="1" top="6" level="2"/>
```

Note: when you use this node in your FXML file, the bottom, top and level properties should of course be bound to the actual values of the lift.

5. Requirements

Layout

To get full marks, you should layout your windows to look as close as possible to the screenshots. This means that you should try to duplicate the spacing between and around nodes that is shown in the screenshots, and the width and height of the nodes, and the alignment of the nodes. All TableViews stretch their columns to fit the width of the table. This is achieved by setting the columnResizePolicy:

```
<TableView>
      <columnResizePolicy>
            <TableView fx:constant="CONSTRAINED_RESIZE_POLICY"/>
      </columnResizePolicy>
      ...
</TableView>
```

Style

A CSS file is included in the skeleton code with the following style classes and id styles:

- The `root` style class should be used for the root node of each window.
- The `heading` style class should be used for the "Lift System" heading on each window.
- The `subheading` style class should be used for the subheading below "Lift System" on each window (i.e. the text such as "Building", "Lift 3", "Call lift", "People").
- The `time` style class should be used for the time indicator.
- The `direction` style class should be used for the direction indicator in the "Lift" window.
- The `errorText` id style should be used for error messages.

Code

Your solution must satisfy the following code requirements:

- Your solution must follow an MVC architecture.
- Your package structure should place the models in a package named "model", the views in a package named "view" and the controllers in a package named "controller".
- The models must notify the views of changes by correctly applying the JavaFX property patterns and observable lists. Model data that can change must be observable. Model data that never changes does not need to be observable.
- Each view must be defined in FXML.
- All error messages must be displayed in catch blocks in response to exceptions being thrown. For example, when calling a lift to a destination that is outside of any lift's range, the model throws an exception and the controller catches it and shows it.
- The "Exit" button must call the shutdown() method on the building before closing the window.

6. Peer marking and demonstration

In your scheduled week 11 lab class (20/10/2016) you must demonstrate your assignment to your tutor and be prepared explain parts of your code to your tutor if requested. If you are unable to explain your code, it may impact your marks. Your presence is required at this class. Any student who is not present without being granted prior permission may have up to 50% of their marks for this assignment deducted.

In addition to demonstrating your assignment, you will also be assigned two other students to peer mark, and two other students will be assigned to peer mark you. The purpose of this peer marking is to mark the functionality of your application which cannot be tested by PLATE. Your marks for functionality will be based on these peer marks after they are moderated by the subject coordinator. Aside from marks for the functionality, the subject coordinator will also mark your code to ensure that all code requirements have been met. Your final mark will be a combination of marks for functionality and marks for code (See "Marking scheme"). Note that you can only be marked for features that can be demonstrated to work.

Marking the code and analysing spoofing, cheating and plagiarism is done in the two weeks following the due date. If you are suspected of Academic Misconduct, I will forward your case to the Misconduct Committee and will notify you by email. Your mark will be finalised within 2 weeks of the due date.

7. Submission to PLATE

READ THIS ENTIRE SECTION CAREFULLY

Included in the skeleton code is a file called `progress.txt` which you must fill out as you progress through the assignment. This file will contain lines such as these:

[?] The Building window is working.
[?] The Lift window is working.
[?] The Call window is working.
...etc...

As you make progress on your assignment, you must edit this file by changing each [?] into a [y]. For example, after you get the Building window to work, you edit this file as follows:

[y] The Building window is working.
[?] The Lift window is working.
[?] The Call window is working.
...etc...

Important: Each time you make progress by changing a [?] into a [y], you must submit your progress to PLATE. If you skip a submission to PLATE to reflect your progress on a particular feature, then your marks for that feature won't count! That is, you are only marked for those features where you submit evidence of your progress. Be very careful to always submit your progress as soon as you make progress so that you don't lose any marks unnecessarily. At a minimum, you are required to make a submission after changing each [?] to a [y]. However, you can make more than the minimum required of submissions, for example, if you notice a bug and you want to fix it and resubmit, or you want to simply improve your code and resubmit. There is no penalty for submitting more than the minimum required submissions.

Your solution is to be submitted to PLATE at https://plate.it.uts.edu.au/ under Applications Programming / Assessments / Assignment 2. Your assignment should be submitted as a JAR file that includes:

- All Java source files required to compile your assignment.
- All FXML, CSS and image files required to run your assignment.
- The progress.txt file at the top level of your project directory structure.

Based on your submitted progress.txt file, PLATE will calculate a mark. This mark should NOT be considered in any way as your final mark. Rather, it should be considered as a "potential" mark. On the week 11 demonstration and peer marking day, the system will try to assign you to peer mark other students who have a similar potential mark as yourself.

There is no scheduled late submission period. An extension of up to one week may be given by the subject coordinator before the due date; you have to supply documentary evidence of your claim. An extension CANNOT be given after the due date.

You may also apply for special consideration for reasons including unexpected health, family or work problems. More information about how to apply for special consideration can be found at http://www.sau.uts.edu.au/assessment/consideration.html.

8. Marking scheme

Task	**Mark**
Building window - 8%: All nodes are shown and FXML is used(*) - 5%: The 3 lifts are shown - 3%: Layout is correct - 3%: Fonts and colours are correct - 3%: Button enabling/disabling is correct - 5%: The time updates via the observer pattern(*) - 5%: The slider adjusts the time speed - 3%: The buttons all work and the “Exit” button calls shutdown() in building (*)	35%
Lift window - 8%: All nodes are shown and FXML is used(*) - 2%: Layout is correct - 2%: Fonts and colours are correct - 4%: All lift information is visible - 4%: All lift information is correctly formatted (e.g. direction is UP/DOWN/--) - 8%: Lift information updates via the observer pattern(*) - 2%: The “Close” button works	30%
Call Lift window - 4%: All nodes are shown and FXML is used(*) - 2%: Layout is correct - 2%: Fonts and colours are correct - 2%: The “You must select a caller” error works using exceptions(*) - 2%: The “Destination must be an integer” error works using exceptions(*) - 2%: The “No suitable lift found” error works using exceptions(*) - 2%: The “Cancel” button works - 4%: The “Call” button works	20%
People window - 1%: All nodes are shown using FXML(*) and the layout is correct - 1%: The ID column is displayed and formatted correctly - 1%: The Name column is displayed and formatted correctly - 1%: The Level column is displayed and formatted correctly - 1%: The Destination column is displayed and formatted correctly - 1%: The Aboard column is displayed and formatted correctly - 4%: All column values update via the observer pattern(*)	10%
Advanced functionality - 3%: The lift column is formatted correctly with the Monospaced font and updates via the observer pattern(*) - 2%: The other columns are formatted correctly and update via the observer pattern(*)	5%
Bonus functionality - Your mark will be based on the quality of your code(*), whether the feature is fully functional and also how accurate the visual appearance is.	20%

(*) The code will be checked by the subject coordinator in the 2 weeks following the due date.

5. Online support

The Assignment 2 discussion board has been set up so that students can ask questions, and other students can reply. I will post a reply only if I think the student response was wrong, or in the case of correcting a mistake in the assignment specification.

You must not post Java code to the discussion board. The board is there to help you, not to provide the solution. Posting your code is academic misconduct and will reported. Each time this rule is violated, I will delete the code and post a comment of the form: “Strike 1: Posting code”. After 3 strikes, the discussion board will be deleted because it did not work.

FAQs (Frequently Asked Questions) and their answers are posted on UTSOnline in Assignments/2/faq. If you have a question, check the FAQ first; it may already be answered there. You should read the FAQ at least once before you hand in your solution, but to be safe check it every couple of days. Anything posted on the FAQ is considered to be part of the assignment specification. The FAQ will be frozen (no new entries) two days before the due date; no questions will be answered after it is frozen.

If anything about the specification is unclear or inconsistent, contact me and I will try to make it clearer by replying to you directly and posting the common questions and answers to the FAQ. This is similar to working on the job, where you ask your client if you are unsure what has to be done, but then you write all the code to do the task. Email Ryan.Heise@uts.edu.au to ask for any clarifications or corrections to the assignment.