Applications Programming

Lab - GUI Lists

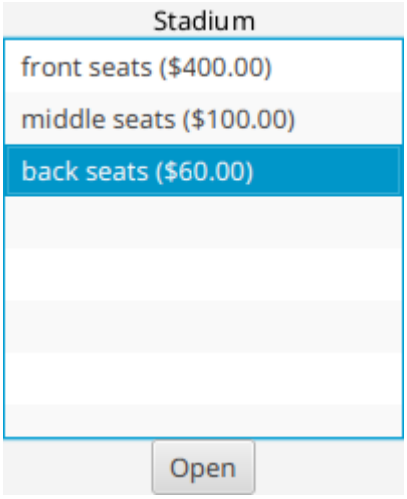
The process:

1. Create the application class.
2. Create the view in FXML.
3. Create the controller in Java.
4. Modify the model to support JavaFX properties.
5. Implement the event handlers.

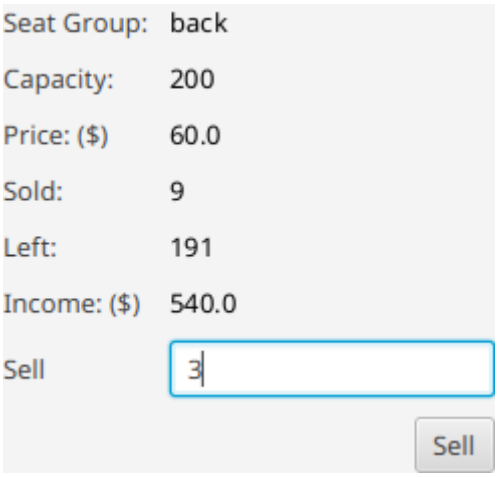
Tutor demo

Main class: `StadiumApplication`

Make an application that sells seats in a stadium. The seats are divided into different groups and each seat group has a different price and capacity. The user interface has two windows as shown below:



Click "Open" →



The user selects a seat group from the list and clicks open. This opens the selected seat group in a new window where the user can view the seat group and sell seats in that group.

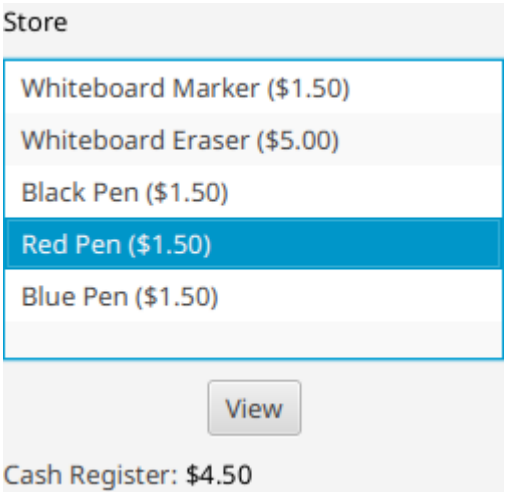
Your tutor will code the solution.

Student Specification #1 (Estimated time: 20-30 minutes)

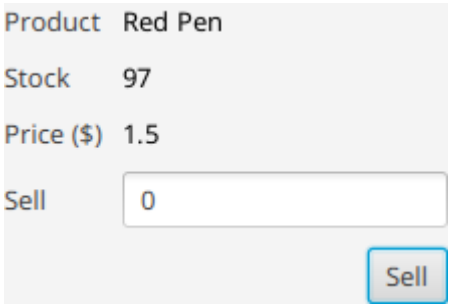
Main class: `StoreApplication`

Please try to finish this exercise quickly. In the remaining lab time, you are then encouraged to make a start on the first window of your assignment.

Make a graphical user interface for the program you developed in the earlier lists lab: a store with a list of products and a cash register. The user interface has two windows as follows:



Click "View" →



In the first window, the user selects a product from the list and clicks the "View" button. This opens the second window which shows the details for the selected product and allows the user to sell that product.

Step 1. Open the lecture notes as a reference.

Step 2. Download and open the skeleton code in either NetBeans or your preferred environment. Browse through the included files, and notice that the `au.edu.uts.ap.javafx` package has also been included to help with opening windows and linking the model with the view and controller.

Step 3. Modify class StoreController to extend Controller<Store>. You need to use <Store> because the model is a Store.

Step 4. Modify the `fx:controller` attribute in store.fxml to look like this:

```
fx:controller="controller.StoreController"
```

You need to write it this way because the `StoreController` class is now inside the `controller` package.

Step 5. Write one line of code in the StoreApplication class to show the store.fxml window like this:

```
ViewLoader.showStage(the_model, "the_view.fxml", "the window title", the_stage);
```

Instructions are written in the comments within the main class. Run the application. A window should appear containing a "Store" heading, a "View" button and a "Cash Register:" label.

Step 6. Add a ListView to store.fxml with width 250 and height 150. The store.fxml contains a comment with instructions on where to insert the ListView. Run the application to check that the ListView appears.

Step 7. Find this code in the Store model:

```
private LinkedList products = new LinkedList();
```

Change this list to be an observable list instead of a linked list, and make sure the getProduct() method also returns an observable list.

Step 8. Add a "store" property to the StoreController class like this:

```
public final Store getStore() { return model; }
```

Note that StoreController has inherited a `model` field from its superclass and this was automatically injected with the store model that was passed in to the ViewLoader in Step 5 above.

Step 9. Add binding code (either in FXML or Java) to display the list of products and the contents of the cash register in the view.

Binding code to display the list:

```
items="${controller.store.products}"
```

Binding code to display the cash:

```
cashTxt.textProperty().bind(getStore().getCashRegister().cashProperty().asString("$%.2f"));
```

Run the application to verify that the products and cash are shown.

Step 10. Apply the **ListView getter pattern** in StoreController so that you can get the currently selected product.

```
private Product getSelectedProduct() {
    ...
}
```

The pattern should be in your notes.

Step 11. We are now going to make the second window work. Add an fx:controller attribute to the product.fxml file to link it to ProductController, like this:

```
fx:controller="controller.ProductController"
```

Step 12. Write a handler for the "View" button so that clicking it will open /view/product.fxml using the currently selected product. You may encounter a compile error that your code has unhandled exceptions. To resolve this, declare your button handler method to throw the exception. For example:

```
@FXML private void handleView(ActionEvent event) throws Exception {
    ...
}
```

Run the application to ensure that the second window is shown after clicking the button. Also test the "Sell" button. Verify that not only is the stock updated in the second window, but so too is the cash register updated in the main window.

Step 13. Finally, add CSS code to your style.css file to make the buttons black with yellow text. When you hover over a button, the background colour should change to blue. Also style the heading so that it has a larger, bold font with colour blue. For examples of CSS code, you may refer to last week's slides.

Exercise 2

The first exercise was intentionally made a bit easier so that you will have time to get started on your assignment. Unlike the lab, the assignment requires you to write all the code yourself. While attempting the assignment, you may refer to your working lab code above for ideas.

Now, you can use the same techniques as above to build the first window containing a ListView for assignment 2. Try to do as much of the assignment as you can based on topics that you have learnt up to this week, and you are welcome to ask your tutor for help along the way.