Application Program Development

Segment : Preparing View(s) for GUI

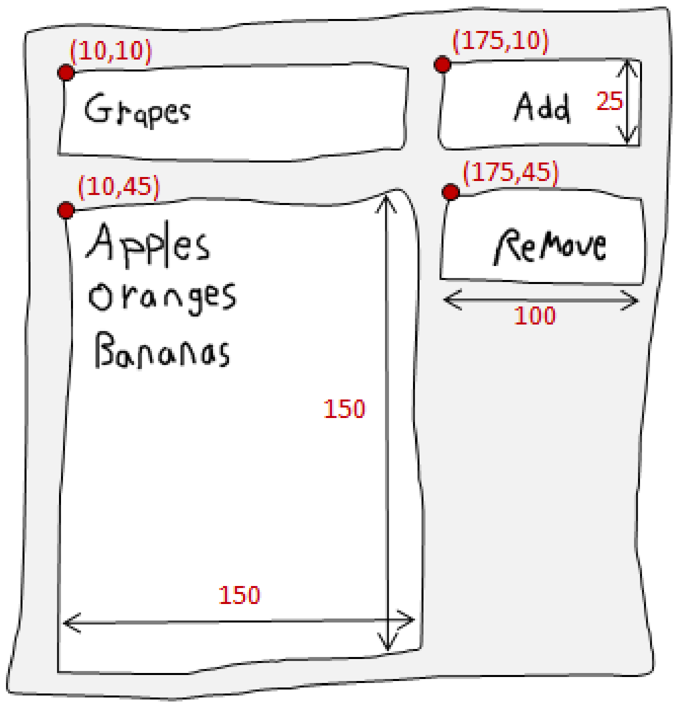
Mahboob Ali

Outcomes

- Preparing View classes
- Proper design need and implementation

- Once you have developed and properly tested the model for your application, you can then begin to write the user interface.
- Sometimes, however, it may be beneficial to have a rough idea as to what your user interface will do before you develop the model.
- For example, we did not write the **add()** and **remove()** methods in the **ItemList** class until we realized that we needed them based on what we want our completed application to do.

- The next step will be to develop the view for the application. In general, there may be <u>many views in an application</u>, just as there may be many models and controllers.
- For this example, we are only considering having one view.
- To keep things simple, we will develop our views as **Pane** objects that can be placed onto our **Application** windows.



```
import javafx.collections.FXCollections;
import javafx.scene.control.*;
import javafx.scene.layout.Pane;

public class GroceryListView extends Pane {
      public GroceryListView() {
      // Create and position the "new item" TextField
      TextField newItemField = new TextField();
      newItemField.relocate(10, 10); newItemField.setPrefSize(150, 25);

      // Create and position the "Add" Button
      Button addButton = new Button("Add");
      addButton.relocate(175, 10); addButton.setPrefSize(100, 25);

      // Create and position the "Remove" Button
      Button removeButton = new Button("Remove");
      removeButton.relocate(175, 45); removeButton.setPrefSize(100, 25);
```

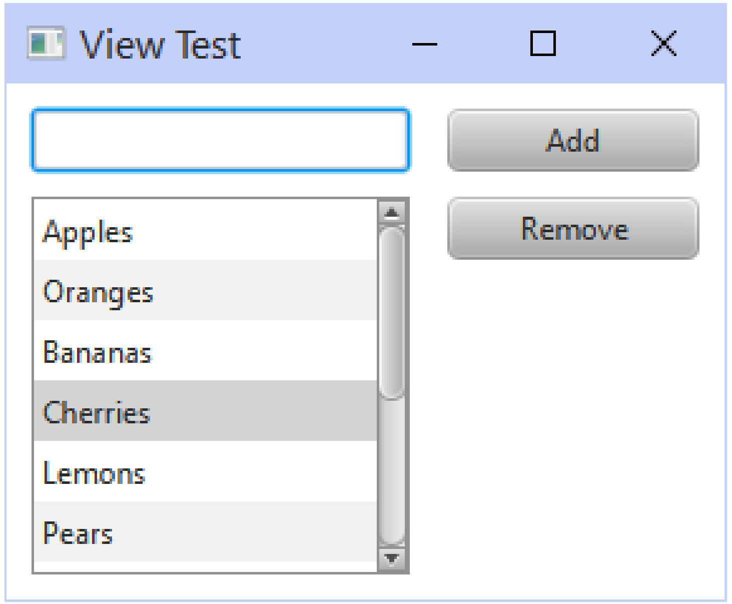
```
        // Create and position the grocery ListView with some groceries in it
        ListView<String> groceryList = new ListView<String>();

        String[] groceries = {"Apples", "Oranges", "Bananas", "Toilet Paper",
               "Ketchup", "Cereal", "Milk", "Cookies", "Cheese", "Frozen Pizza"};

        groceryList.setItems(FXCollections.observableArrayList(groceries));

        groceryList.relocate(10, 45);

        groceryList.setPrefSize(150, 150);


        // Add all the components to the window
        getChildren().addAll(newItemField, addButton, removeButton,
                                                           groceryList);

        }

    }
```

- For the purposes of a quick test to make sure that our view is properly formatted, we can create and run a simple program like this:

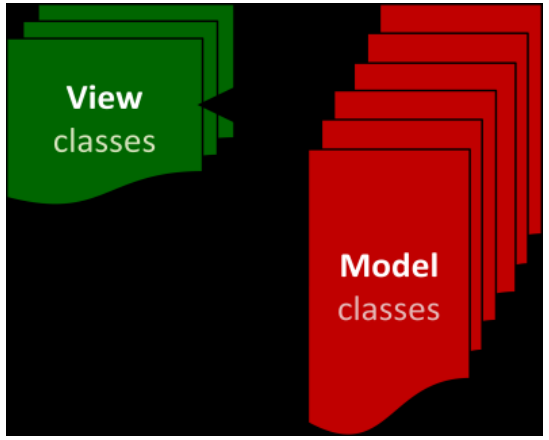
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class GroceryListViewTestProgram extends Application {
      public void start(Stage primaryStage) {
            GroceryListView myPanel = new GroceryListView();
            primaryStage.setTitle("View Test");
            primaryStage.setResizable(true);
            primaryStage.setScene(new Scene(myPanel, 285,205));
            primaryStage.show();
      }
      public static void main(String[] args) { launch(args); }
}
```

- Now, for the view to work properly it must refresh its look based on the most up-to-date information in the model.
- Therefore, we need a way of having the view update itself given a specific model.
- There are many ways to do this, but a simple way is to write a method called **update()** that will refresh the "look" of the view whenever it is called.
- Of course, to be able to update, the view must have access to the model.
- We can pass the model in as a parameter to the view constructor and store it as an attribute of the view:

```
public class GroceryListView extends Pane {
        private ItemList model; // The model to which this view is attached
        public GroceryListView(ItemList m) {
                model = m; // Store the model for access later
                ...
        }
        // This method is called whenever the model changes
        public void update() {
                // ... code for refreshing the view
        }
}
```



- Our **update()** method may be as simple as replacing the entire **ListView** with the new items currently stored in the model.
- So, regardless of what changes take place in the **ItemList** model, we simply read the list of items and re-populate the **ListView** with the latest items.
- To be able to do this, we will need access to that **ListView**. However, the **ListView** is defined as a local variable in the constructor.
- In order to be able to access it from the **update()** method, we will need to define the variable outside of the constructor as an instance variable.
- Then, to change the contents of the **ListView**, a quick search in the JAVA API informs us that there is a **setItems()** method which will allow us to pass in an array of items to show in the list.



```
public class GroceryListView extends Pane {
      private ItemList model; // model to which this view is attached
      private ListView<String> groceryList; // visible list representing model
      public GroceryListView(ItemList m) {
            model = m; // Store the model for access later
            ...
            groceryList = new ListView<String>();
            ...
      }
      // This method is called whenever the model changes
      public void update() {
            groceryList.setItems(FXCollections.observableArrayList(
                                                      model.getItems()));
      }
}
```

- Notice how the **ListView** is now easily accessible in the **update()** method.
- One problem, however, is that our model's array is always of size 100 regardless of how many items have been placed in it.
- The **setItems()** method will end up making a list of 100 items in it … leaving many blanks.
- We can add additional code here to make a new array (for display purposes) which has a length exactly equal to the size of the **items** array. Let's, change the **update()** method to this:

```
    // This method is called whenever the model changes
    public void update() {
        // Create and return a new array with the
        // exact size of the number of items in it
        String[] exactList = new String[model.getSize()];
        for (int i=0; i<model.getSize(); i++)
            exactList[i] = model.getItems()[i];

        groceryList.setItems(FXCollections.observableArrayList(exactList));
}
```

- We are aware that eventually the application will need to respond to user input through pressing the **Add** or **Remove** button, typing in the text field or selecting from the list.
- This will be part of the controller.
- However, in order to accomplish this, as you will soon see, we need to allow the controller to access the **Button**, **ListView** and **TextField** objects.
- We can allow such access by making instance variables for all four window components and then provide **get** methods to access them.
- As a final programming aspect of the view class, it is a good idea to call the **update()** method at the end of the constructor. That way, when the view is first created, it can be refreshed right away to show the true state of the model upon startup.

Updating our view class

```java
import javafx.collections.FXCollections;
import javafx.scene.control.*;
import javafx.scene.layout.Pane;

public class GroceryListView extends Pane {
        private ItemList model; // model to which this view is attached

        // The user interface components needed by the controller
        private ListView<String> groceryList;
        private Button addButton;
        private Button removeButton;
        private TextField newItemField; // public methods to allow access to window components
        public ListView<String> getList() { return groceryList; }
        public Button getAddButton() { return addButton; }
        public Button getRemoveButton() { return removeButton; }
        public TextField getNewItemField() { return newItemField; }
        public GroceryListView(ItemList m) { model = m; // Store the model for access later

        // Create and position the "new item" TextField
        newItemField = new TextField();
        newItemField.relocate(10, 10);
        newItemField.setPrefSize(150, 25);

        // Create and position the "Add" Button
        addButton = new Button("Add");
        addButton.relocate(175, 10);
        addButton.setPrefSize(100, 25);
```

Updating our view class

```
        // Create and position the "Remove" Button
        removeButton = new Button("Remove");
        removeButton.relocate(175, 45);
        removeButton.setPrefSize(100, 25);

        // Create and position the "fruit" ListView with some fruits in it
        groceryList = new ListView<String>(); String[] groceries = {"Apples", "Oranges", "Bananas", "Toilet
                                    Paper", "Ketchup", "Cereal", "Milk", "Cookies", "Cheese", "Frozen Pizza" };
        groceryList.setItems(FXCollections.observableArrayList(groceries));
        groceryList.relocate(10, 45);
        groceryList.setPrefSize(150, 150);

        // Add all the components to the window
        getChildren().addAll(newItemField, addButton, removeButton, groceryList);

        // Call update() to make sure model contents are shown
        update();
        }
        // This method is called whenever the model changes, to make
        // sure that the view shows the model's current state
        public void update() {
                // Create and return a new array with the
                // exact size of the number of items in it
                String[] exactList = new String[model.getSize()];
                for (int i=0; i<model.getSize(); i++)
                        exactList[i] = model.getItems()[i];
                groceryList.setItems(FXCollections.observableArrayList(exactList));
        }
}
```

Updating our view test class

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class GroceryListViewTestProgram2 extends Application {
      public void start(Stage primaryStage) {
            ItemList groceryList = new ItemList();
            groceryList.add("Apples");
            groceryList.add("Toilet Paper");
            groceryList.add("Ketchup");

            GroceryListView myPanel = new GroceryListView(groceryList);
            primaryStage.setTitle("View Test");
            primaryStage.setResizable(true);
            primaryStage.setScene(new Scene(myPanel, 285,205));
            primaryStage.show();
      }
      public static void main(String[] args) { launch(args); }
}
```