CS 213 – Software Methodology Spring 2016

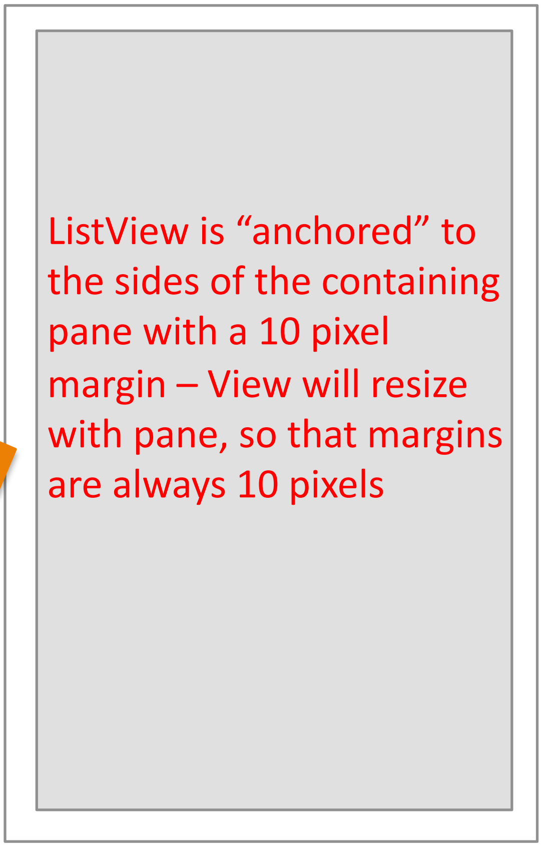
Lecture 5 – Feb 2

GUI - ListView

Step 1: ListView in AnchorPane

view/List.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.control.ListView?>

<AnchorPane
    xmlns="http://javafx.com/javafx/8.0.40"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller=”view.ListController">

    <ListView fx:id="listView"
       AnchorPane.topAnchor = "10"
       AnchorPane.leftAnchor = "10"
       AnchorPane.rightAnchor = "10"
       AnchorPane.bottomAnchor = "10"/>

</AnchorPane>
```


ListView is “anchored” to the sides of the containing pane with a 10 pixel margin – View will resize with pane, so that margins are always 10 pixels

ListView is empty at this point – need to populate it

Step 2: Populating with ObservableList

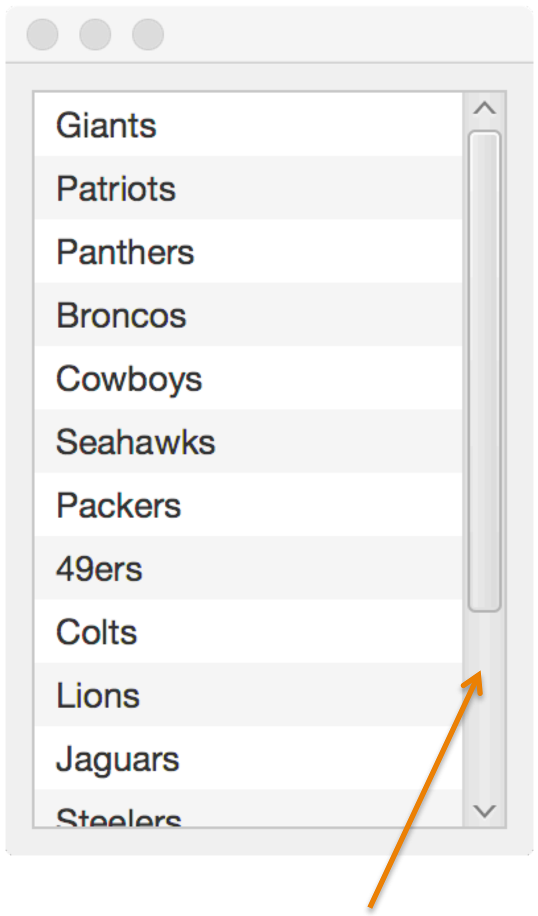
view.ListController

```
package view;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.ListView;

public class ListController {
   @FXML
   ListView<String> listView;

   private ObservableList<String> obsList;

   public void start() {
      // create an ObservableList
      // from an ArrayList
      obsList = FXCollections.observableArrayList(
                   "Giants",
                   "Patriots",
                   ...
                   "Jaguars”);
      listView.setItems(obsList);
   }
}
```

Step 3: Loading and Displaying

app.ListApp

```
package app;
...

public class ListApp extends Application {

   public void start(Stage primaryStage)
   throws Exception {
      FXMLLoader loader = new FXMLLoader();
      loader.setLocation(
         getClass().getResource("/view/List.fxml"));
      AnchorPane root = (AnchorPane)loader.load();

      ListController listController =
         loader.getController();
      listController.start();

      Scene scene = new Scene(root, 200, 300);
      primaryStage.setScene(scene);
      primaryStage.show();
   }

   public static void main(String[] args) {
      launch(args);
   }
}
```



Scroll bar automatically appears if list is longer than view area

Step 4: Listening to List Item Selection

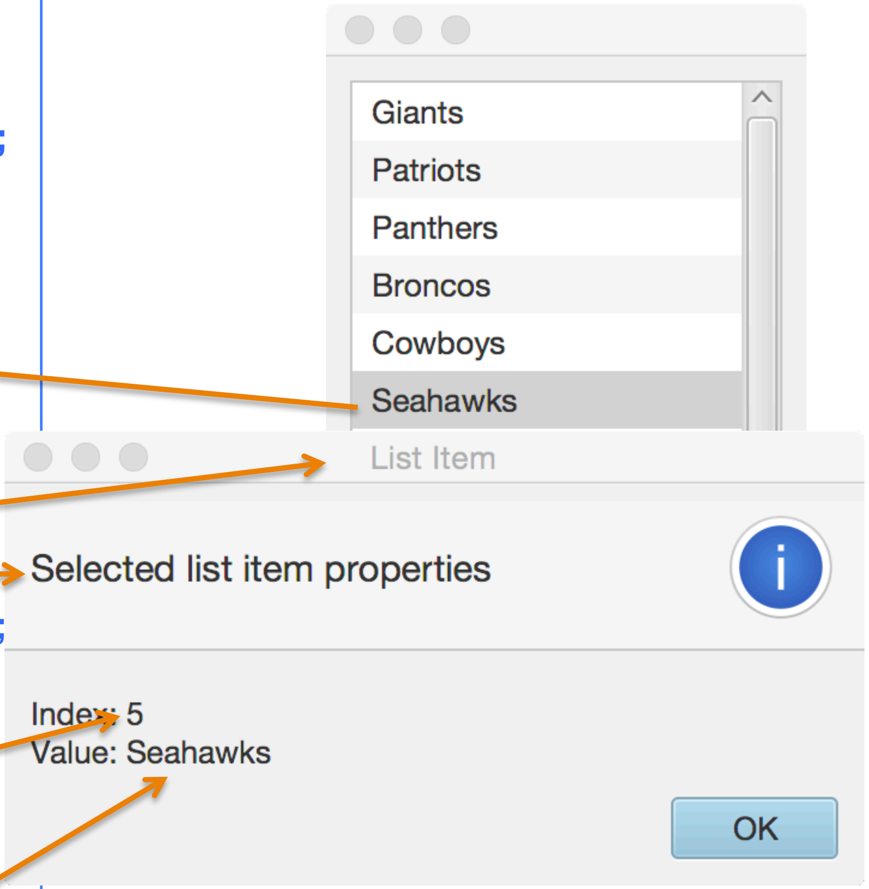
view.ListController

```
package listview;
...
import javafx.stage.Stage;

public class ListController {
   ...

   public void start(Stage mainStage) {
      ...
      // select the first item
      listView.getSelectionModel().select(0);

      // set listener for the items
      listView
        .getSelectionModel()
        .selectedItemProperty()
        .addListener(
           (obs, oldVal, newVal) ->
               showItem(mainStage));
   }
}
```

lambda expression for the `changed` method of the functional interface `javafx.beans.value.ChangeListener`

Step 5: Responding to List Item Selection

```
package listview;
...
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;

public class ListController {
   ...

   private void showItem(Stage mainStage) {
      Alert alert =
         new Alert(AlertType.INFORMATION);
      alert.initOwner(mainStage);
      alert.setTitle("List Item");
      alert.setHeaderText(
           "Selected list item properties");

      String content = "Index: " +
          listView.getSelectionModel()
                   .getSelectedIndex() +
          "\nValue: " +
          listView.getSelectionModel()
                   .getSelectedItem();

          alert.setContentText(content);
          alert.showAndWait();
   }
}
```



Giants
Patriots
Panthers
Broncos
Cowboys
Seahawks

Selected list item properties

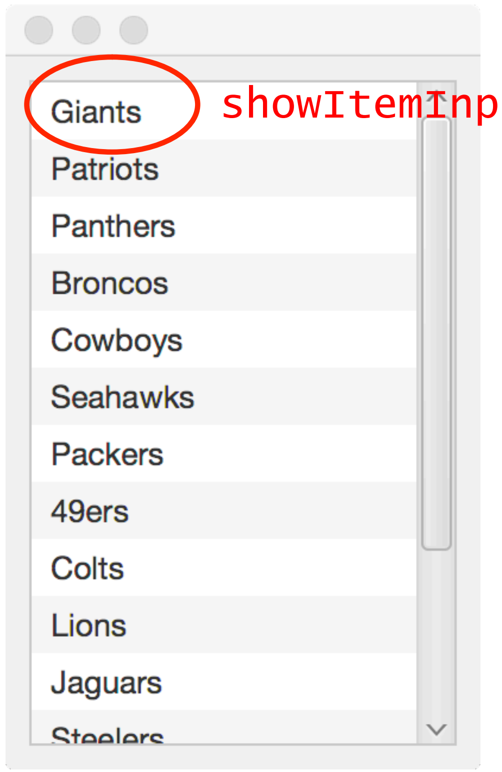
Index: 5
Value: Seahawks

The dialog will be up until user responds, and will not allow interaction with owner window: this makes the dialog "modal"

Enhancement: Change Item

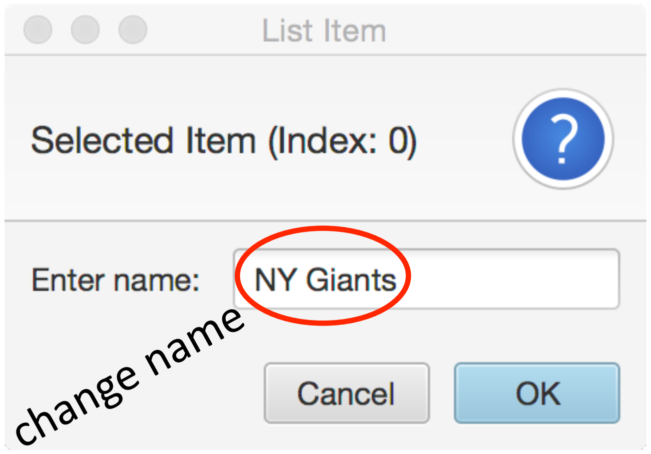
```
package listview;
...
import java.util.Optional; import javafx.scene.control.TextInputDialog;

public class ListController {
   ...
   public void start(Stage mainStage) {
      ...
      listView.
        ...
        .addListener((obs, oldVal, newVal) ->
               showItemInputDialog(mainStage));
   }

   private void showItemInputDialog(Stage mainStage) {
      String item = listView.getSelectionModel().getSelectedItem();
      int index = listView.getSelectionModel().getSelectedIndex();

      TextInputDialog dialog = new TextInputDialog(item);
      dialog.initOwner(mainStage); dialog.setTitle("List Item");
      dialog.setHeaderText("Selected Item (Index: " + index + ")");
      dialog.setContentText("Enter name: ");

      Optional<String> result = dialog.showAndWait();
      if (result.isPresent()) { obsList.set(index, result.get()); }
   }
}
```

ObservableList => ListView Auto Update


Giants
Patriots
Panthers
Broncos
Cowboys
Seahawks
Packers
49ers
Colts
Lions
Jaguars
Steelers

`showItemInputDialog`

`TextInputDialog`
`showAndWait()`



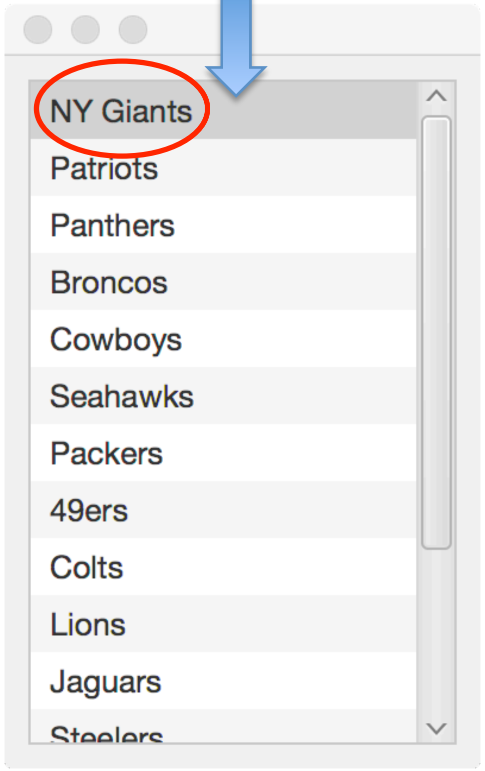
Selected Item (Index: 0)

Enter name: NY Giants

change name

`Optional<String> result`

`obsList.set(index, result.get())`


NY Giants
Patriots
Panthers
Broncos
Cowboys
Seahawks
Packers
49ers
Colts
Lions
Jaguars
Steelers

Updating `ObservableList` automatically refreshes the `ListView`