# CS 213 – Software Methodology

Spring 2023

*Sesh Venugopal*

Feb 6

GUI using FXML

## Fahrenheit-Celsius Converter

Version 2

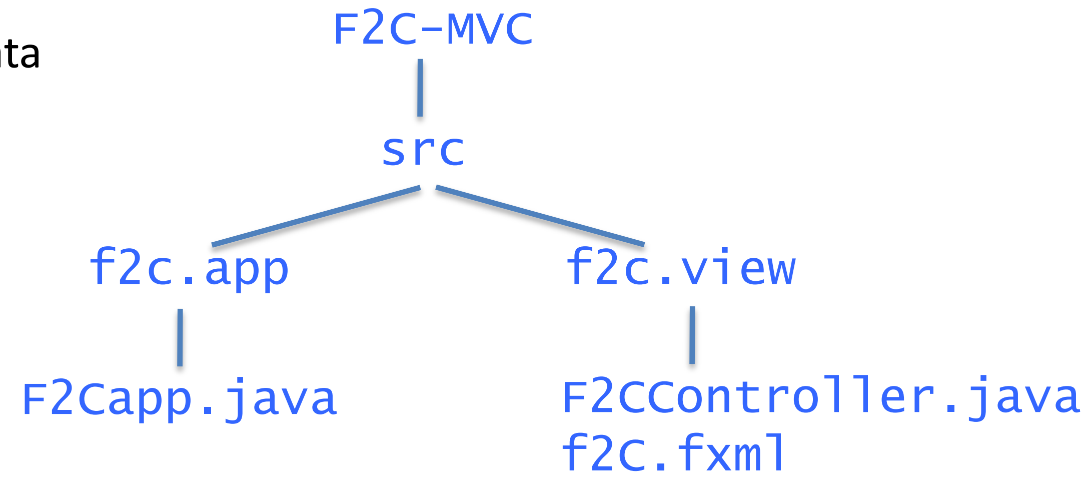
UI implemented in FXML
(markup language like HTML)

# The MVC Code Architecture (Model-View-Controller)

Model is the set of classes
that store and manage application data

View is the set of Java classes
and non-Java design artifacts
(e.g. xml, css, etc.) that implement
the user interface

Controller is the set of classes that
broker between Model and View



NOTE:

1. Each of the M, V, and C parts of thane application need not always be in its own separate package
2. JavaFX uses the term "controller" to mean a Java class that holds the UI objects (e.g. `F2CController`) – this is different from the controller part of the MVC architecture that holds core application logic

# View: Layout using fxml

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

<?import javafx.scene.layout.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.text.*?>
<?import javafx.geometry.*?>
```

Don't forget imports!! (Editor won't flag errors for unresolved tags.)

Some of the tags may be different if you use a version > 11

```
<GridPane
    xmlns="http://javafx.com/javafx/11"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="f2c.view.F2CController"
    vgap="10" hgap="10">
```

Name space for Java FX tags (e.g. Text)

Name space for FXML tags (e.g. fx:controller)

Controller class to which the UI will be mapped

Row and column indexes default to 0

```
    <Text text="Fahrenheit" GridPane.valignment="BOTTOM"/>
    <Button text="&gt;&gt;&gt;" GridPane.columnIndex="1" />
    <Text text="Celsius" GridPane.columnIndex="2" GridPane.valignment="BOTTOM"/>
    <TextField prefColumnCount="10" promptText="-40.0" GridPane.rowIndex="1" />
    <Button text="&lt;&lt;&lt;" GridPane.rowIndex="1" GridPane.columnIndex="1" />
    <TextField prefColumnCount="10" promptText="-40.0"
           GridPane.rowIndex="1" GridPane.columnIndex="2" />
    <padding>
        <Insets top="10" right="10" bottom="10" left="10"/>
    </padding>
</GridPane>
```

## View: Set up SceneBuilder

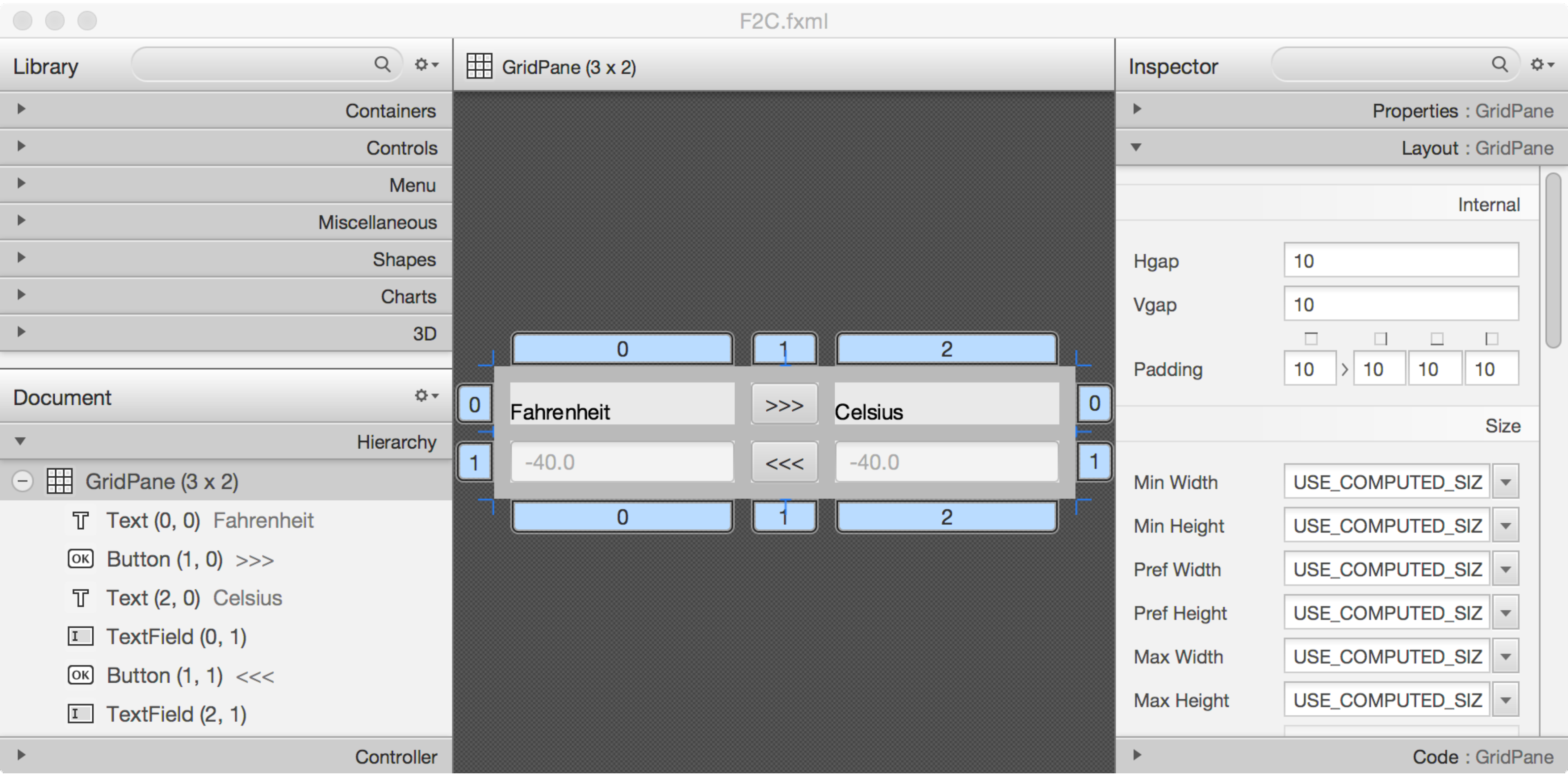
- Get SceneBuilder at Gluon:
  https://gluonhq.com/products/scene-builder/

  Download and install SceneBuilder 19 (works with Java 11 or higher)

- You can open up the SceneBuilder application and load up any fxml file to create/edit a layout using its drag-and-drop abilities to place widgets, and editor to set widget properties

- You can construct UIs exclusively using SceneBuilder interface, or you can write up the UI fxml file in an editor and optionally verify/polish using SceneBuilder

# Verify fxml Layout with SceneBuilder



F2C.fxml

Library

Containers
Controls
Menu
Miscellaneous
Shapes
Charts
3D

Document

Hierarchy

GridPane (3 x 2)
- Text (0, 0) Fahrenheit
- Button (1, 0) >>>
- Text (2, 0) Celsius
- TextField (0, 1)
- Button (1, 1) <<<
- TextField (2, 1)

Controller

GridPane (3 x 2)

| Fahrenheit | >>> | Celsius |
|---|---|---|
| -40.0 | <<< | -40.0 |

Inspector

Properties : GridPane

Layout : GridPane

Internal

| | |
|---|---|
| Hgap | 10 |
| Vgap | 10 |
| Padding | 10 10 10 10 |

Size

| | |
|---|---|
| Min Width | USE_COMPUTED_SIZ |
| Min Height | USE_COMPUTED_SIZ |
| Pref Width | USE_COMPUTED_SIZ |
| Pref Height | USE_COMPUTED_SIZ |
| Max Width | USE_COMPUTED_SIZ |
| Max Height | USE_COMPUTED_SIZ |

Code : GridPane

(In SceneBuilder, do Preview -> Show Preview in Window to simulate layout behavior)

# fxml Layout – Id’ing widgets

```
...

<Text text="Fahrenheit" GridPane.valignment="BOTTOM"/>

<Button fx:id=“f2c” text="&gt;&gt;&gt;" GridPane.columnIndex="1" />

<Text text="Celsius" GridPane.columnIndex="2" GridPane.valignment="BOTTOM"/>

<TextField fx:id=“f” prefColumnCount="10" promptText="-40.0"
    GridPane.rowIndex="1" />

<Button fx:id=“c2f” text="&lt;&lt;&lt;" GridPane.rowIndex="1"
    GridPane.columnIndex="1" />

<TextField fx:id=“c” prefColumnCount="10" promptText="-40.0"
          GridPane.rowIndex="1" GridPane.columnIndex="2" />

<padding>
    <Insets top="10" right="10" bottom="10" left="10"/>
</padding>
```

# fxml Layout – Naming Event Handlers

```
...

<Text text="Fahrenheit" GridPane.valignment="BOTTOM"/>

<Button fx:id=“f2c” text="&gt;&gt;&gt;" GridPane.columnIndex="1"
    onAction=“#convert” />

<Text text="Celsius" GridPane.columnIndex="2" GridPane.valignment="BOTTOM"/>

<TextField fx:id=“f” prefColumnCount="10" promptText="-40.0"
    GridPane.rowIndex="1" />

<Button fx:id=“c2f” text="&lt;&lt;&lt;" GridPane.rowIndex="1"
    GridPane.columnIndex="1" onAction=“#convert” />

<TextField fx:id=“c” prefColumnCount="10" promptText="-40.0"
          GridPane.rowIndex="1" GridPane.columnIndex="2" />

<padding>
    <Insets top="10" right="10" bottom="10" left="10"/>
</padding>
```

# Controller that shadows FXML UI (Java Code)

```
package f2c.view;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class F2CController {

    @FXML Button f2c;
    @FXML Button c2f;
    @FXML TextField f;
    @FXML TextField c;

    public void convert(ActionEvent e) {
        Button b = (Button)e.getSource();
        if (b == f2c) {
            float fval = Float.valueOf(f.getText());
            float cval = (fval-32)*5/9;
            c.setText(String.format("%5.1f", cval));
        } else {
            float cval = Float.valueOf(c.getText());
            float fval = cval*9/5+32;
            f.setText(String.format("%5.1f", fval));
        }
    }
}
```

This `f2c.view.Controller` class is the one that is referenced in the fxml file:

```
<GridPane
    xmlns="http://javafx.com/javafx/11"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="f2c.view.F2CController"
    vgap="10" hgap="10">
```

The JavaFX framework uses the term “controller” to mean a class that is tied to an fxml file.

In MVC terms, the JavaFX controller is actually a part of the View

The C of MVC is the controller part that is separate from any View component

## Controller – Java Code

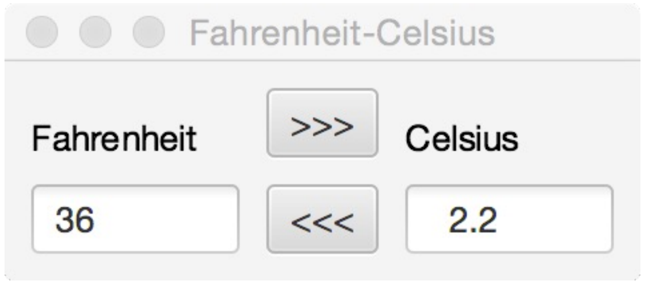
```
package f2c.view;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class F2CController {

    @FXML Button f2c;
    @FXML Button c2f;
    @FXML TextField f;
    @FXML TextField c;

    public void convert(ActionEvent e) {
        Button b = (Button)e.getSource();
        if (b == f2c) {
            float fval = Float.valueOf(f.getText());
            float cval = (fval-32)*5/9;
            c.setText(String.format("%5.1f", cval));
        } else {
            float cval = Float.valueOf(c.getText());
            float fval = cval*9/5+32;
            f.setText(String.format("%5.1f", fval));
        }
    }
}
```

@FXML directive links widget to fxml element:
var name in code = id in layout

Name of method = name assigned in # directive in fxml file for onAction attribute

# Main App for View/Controller

```
package f2c.app;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
...

public class F2CApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("/f2c/view/f2C.fxml"));


        GridPane root = (GridPane)loader.load();

        Scene scene = new Scene(root);
        ...
    }

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



Top-level layout tag in fxml file

Creating loader with full path name of fxml file, relative to project name as root

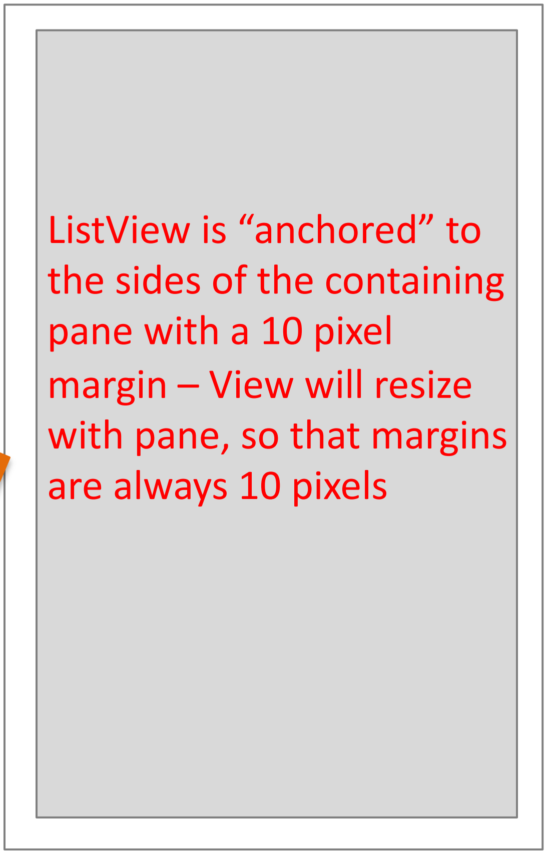
Loading creates Java objects for various widgets and layouts in the fxml file

## ListView, Dialogs

# 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/11"
    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

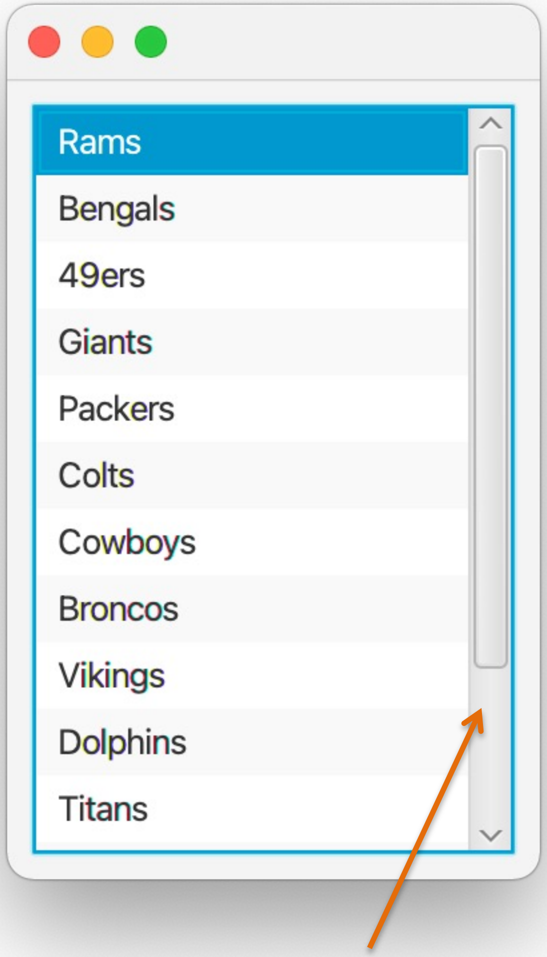
view.ListController

```
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(
                 ”Rams",
                 ”Bengals",
                 ...
                 "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

Remember:
DO NOT CREATE A CONTROLLER INSTANCE with `new` – it will not have any connection to the FXML-sourced widgets with which the user will interact

The way to get at the controller instance that links to the FXML layout is to call `getController()` on the `FXMLLoader` AFTER you call `load()` on it