CS 213 – Software Methodology

Spring 2023

Sesh Venugopal

Feb 1

Graphical User Interface

Preparing to build GUIs in Java FX

To write Java FX GUI programs, you need Java SDK version >= 11

Install Java FX SDK Version 19 for your computer platform architecture:
https://gluonhq.com/products/javafx/

(If you have an older version of Java FX SDK, it's fine – earliest is Java FX 11.)

You need to set up your IDE to work with Java FX.

You will be building non-modular projects, so follow instructions for **non modular** projects in the following Java FX doc page (also has instructions for other IDEs):

https://openjfx.io/openjfx-docs/#IDE-Eclipse

Pay attention to point #3 – Add VM arguments.

- Make sure to replace "/path/to" with the actual path on your computer
- Make sure to uncheck **Use the -XstartOnFirstThread argument when launching with SWT**

Fahrenheit-Celsius Converter

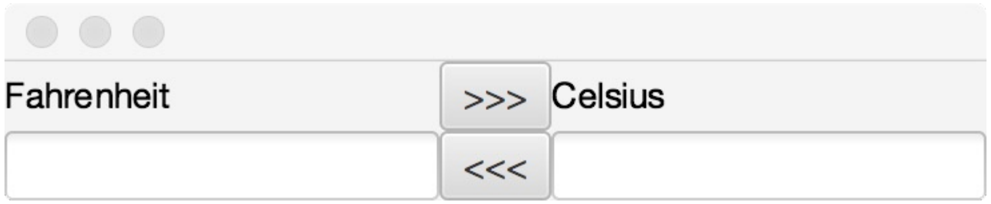
Version 1

Programmatic Layout

Programmatic Layout – Widgets/ Layout

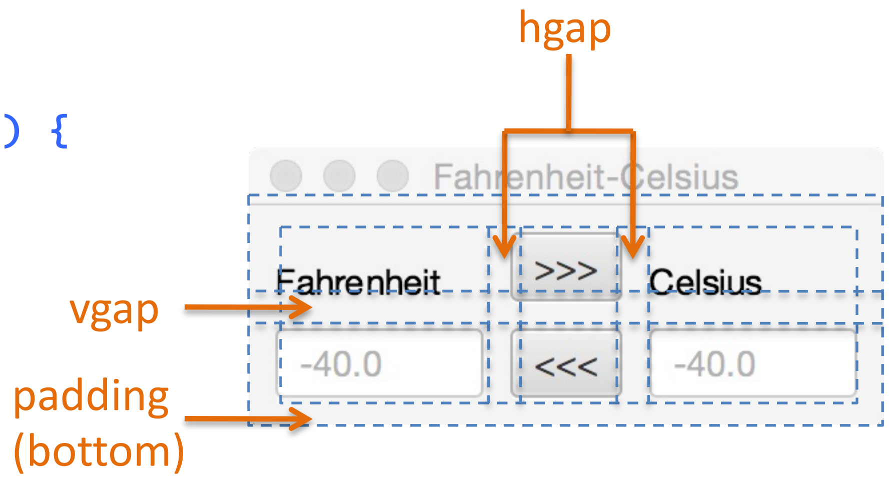
```
@Override
public void start(Stage primaryStage) {
    GridPane root = makeGridPane();
    Scene scene = new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.show();
}

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

```
private static GridPane makeGridPane() {

    // all the widgets
    Text fText = new Text("Fahrenheit");
    Text cText = new Text("Celsius");
    TextField f = new TextField();
    TextField c = new TextField();
    Button f2c = new Button(">>>");
    Button c2f = new Button("<<<");

    GridPane gridPane = new GridPane();
    gridPane.add(fText, 0, 0);
    gridPane.add(f2c, 1, 0);
    gridPane.add(cText, 2, 0);
    gridPane.add(f, 0, 1);
    gridPane.add(c2f, 1, 1);
    gridPane.add(c, 2, 1);

    return gridPane;
}
```



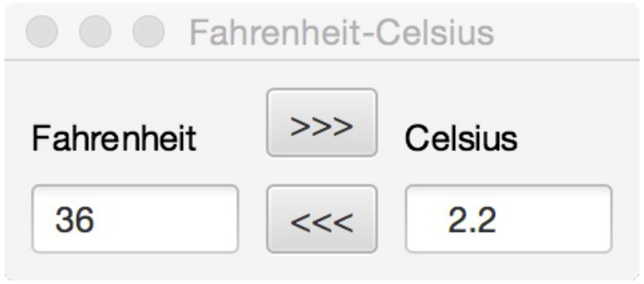
Programmatic Layout – Gaps/Alignment

```
@Override
public void start(Stage primaryStage) {
     ...
     primaryStage.setTitle("Fahrenheit-Celsius");
     primaryStage.setResizable(false);
     primaryStage.show();
}

private static GridPane makeGridPane() {

     // all the widgets
     ...

     f.setPrefColumnCount(5);
     f.setPromptText("-40.0");
     c.setPrefColumnCount(5);
     c.setPromptText("-40.0");
     gridPane.setHgap(10);
     gridPane.setVgap(10);                    t   r   b   l
     gridPane.setPadding(new Insets(10,10,10,10));
     GridPane.setValignment(fText, VPos.BOTTOM);
     GridPane.setValignment(cText, VPos.BOTTOM);

     return gridPane;
}
```



text aligned with bottom of its grid cell

Programmatic Layout – Event Handling



```
private static GridPane makeGridPane() {

    ... // all the widgets

    ... // gaps and alignment

    // event handling
    f2c.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
            float fval = Float.valueOf(f.getText());
            float cval = (fval-32)*5/9;
            c.setText(String.format("%5.1f", cval));
        }
    });

    c2f.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
            float cval = Float.valueOf(c.getText());
            float fval = cval*9/5+32;
            f.setText(String.format("%5.1f", fval));
        }
    });

    return gridPane;
}
```

Fahrenheit-Celsius Converter

Event Handling

Register an event listener with the f2c button

```
f2c.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
            float fval = Float.valueOf(f.getText());
            float cval = (fval-32)*5/9;
            c.setText(String.format("%5.1f", cval));
        }
    });
```

When a button is clicked on the GUI, an "`action event`" is triggered

The GUI event manager (that is running in the background)

- creates an `ActionEvent` instance,
- then calls the `handle` method of all event listeners that are registered on that button
- passing the `ActionEvent` instance as parameter

Register an event listener with the f2c button

```
f2c.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
           float fval = Float.valueOf(f.getText());
           float cval = (fval-32)*5/9;
           c.setText(String.format("%5.1f", cval));
        }
    });
```

The event listener for a button needs to be an `EventHandler` object that deals with `ActionEvent`

Being an `EventHandler` object means implementing the `handle` method.

Register an event listener with the f2c button

```
f2c.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
            float fval = Float.valueOf(f.getText());
            float cval = (fval-32)*5/9;
            c.setText(String.format("%5.1f", cval));
        }
    });
```

An `EventHandler` object may be customized to handle different types of events. Here, it must be customized to handle `ActionEvent`

```
f2c.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
            ...
        }
    });
```

So, to the `setOnAction` method we need to send an instance of a class that implements such an event handler

```
f2c.setOnAction(   );
```

But `EventHandler` is not a class, it is an *interface*

This means we need to
- first define a class that implements the `EventHandler` interface customized for `ActionEvent`
- then create an instance of this class
- then send in that instance as the argument to `setOnAction`

What does it mean to define a class that implements an interface?

We'll get the details of it very soon when we study interfaces in depth.

For now, just know that you will need to write something like this:

```
class SomeName implements EventHandler<ActionEvent> {
    public void handle(ActionEvent e) {
        ... // appropriate logic for event handling
    }
}
```

Turns out, you don't need to do this if you only need to use a particular event handling logic once or twice in your application in a very localized manner

- Then skip the explicit class name and use an anonymous class

- Like this:

```
~~class SomeName implements ~~EventHandler<ActionEvent> {
    public void handle(ActionEvent e) {
        ... // appropriate logic for event handling
    }
}
```

- So you are left with this anonymous definition:

```
EventHandler<ActionEvent> {
    public void handle(ActionEvent e) {
        ... // appropriate logic for event handling
    }
}
```

Remember, `EventHandler` is NOT a class name, it's an interface name

- Since you don't have a class name, creating an instance must be done right alongside the anonymous definition:

```
new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {
        ... // appropriate logic for event handling
    }
}
```

- For initialization, your only option is to call the no-arg constructor

Since the name of the constructor must be the same as the name of the class, and the anonymous class has no name, you can't define a constructor

- And you stick this entire code in as parameter to the `setOnAction` method

```
f2c.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
            ...
        }
    });
```

Anonymous Classes – Usage

Anonymous classes are a compact way to define a class and create an instance, to be used in a highly *localized* manner

On the other hand, if you want the same event handling code to be used in many places, then you are better off defining a named class once, and using instances of it in all those places

An anonymous class can be used in any context, not just for GUIs or event handling

An anonymous class is not inherently tied to an interface, it can be used independently of an interface

JavaFX Documentation

JavaFX documentation project: https://fxdocs.github.io/docs/html5/

https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

JavaFX API: https://docs.oracle.com/javase/8/javafx/api/toc.htm