Unleash the Power of JavaFX

Angela Caicedo
Java Evangelist, Oracle

Introduction

The JavaFX platform is the evolution of the Java client platform designed to enable application developers to easily create and deploy rich internet applications (RIAs) that behave consistently across multiple platforms. Built on Java technology, the JavaFX platform provides a rich set of graphics and media API with high-performance hardware-accelerated graphics and media engines that simplify development of data-driven enterprise client applications.

Key Features in the latest release

The main focus areas for the JavaFX 2 release include the following features:

- **Full integration with JDK 7** is now available. As of the release of JavaFX SDK 2.2 and Java SE 7 update 6, the JavaFX SDK is fully integrated with the Java SE 7 Runtime Environment (JRE) and Development Kit (JDK). A standalone download of JavaFX 2 SDK for Windows will remain available for users of JDK 6 until Oracle releases the last Java SE 6 public update on November 2012. This integration with the JDK 7 removes the need to download and install JavaFX 2 SDK separately.

- **Java APIs for JavaFX** that provide all the familiar language features (such as generics, annotations, and multithreading) that Java developers are accustomed to using. The APIs are designed to be friendly to alternative JVM languages, such as JRuby and Scala. Because the JavaFX capabilities are available through Java APIs, you can continue to use your favorite Java developer tools (such as IDEs, code refactoring, debuggers, and profilers) to develop JavaFX applications.

- **A new graphics engine** to handle modern graphics processing units (GPUs). The basis of this new engine is a hardware accelerated graphics pipeline, called Prism, that is coupled with a new windowing toolkit, called Glass. This graphics engine provides the foundation for current and future advancements for making rich graphics simple, smooth, and fast.

- **FXML, a new declarative markup language** that is XML-based and is used for defining the user interface in a JavaFX application. It is not a compiled language and, hence, does not require you to recompile the code every time you make a change to the layout.

- **A new media engine** that supports playback of the web multimedia content. It provides a stable, low latency media framework that is based on the GStreamer multimedia framework.

- **A web component** that gives the capability of embedding web pages within a JavaFX application using the WebKit HTML rendering technology. Hardware accelerated rendering is made available using Prism.

- **A wide variety of built-in UI controls**, which include Charts, Tables, Menus, and Panes. Additionally, an API is provided to allow third parties to contribute UI controls that the user community can use.

- **An application packager** that takes the guess out of building an easy to deploy standalone desktop application containing all the Java runtime libraries needed to install and run a JavaFX application.

- **Available on Windows, Mac OS X, and Linux platforms**. As of JavaFX 2.2 release, JavaFX is available on all major desktop platforms, ensuring a consistent runtime experience for developers and end users alike. Oracle ensures synchronized releases and updates on all three platforms, and offers an extensive support program for companies running mission-critical applications.

In this HOL you will have the opportunity to get your hands on real exercises, to learn and play with some of these latest features of JavaFX. We will be using NetBeans and JavaFX Scene Builder along the way. So, lets get started and have some fun.

Exercises:

1. A little bit of everything: JavaFX, NetBeans, Scene Builder, and FXML
2. Styling your application using CSS
3. JavaFX, JavaScript, WebView and HTML5
4. Canvas API

1. A little bit of everything: JavaFX, NetBeans, Scene Builder, and FXML

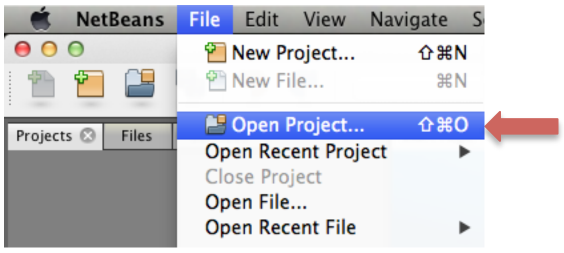
In this exercise we are going to learn how to create impressive user interfaces using JavaFX Scene Builder and NetBeans. We are going to take an existent Swing application, and turn it into an amazing interface using JavaFX and CSS.

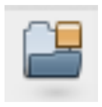
1.1 Getting Ready.

Before we start lets run the Swing application we are going to turn into a great JavaFX app.

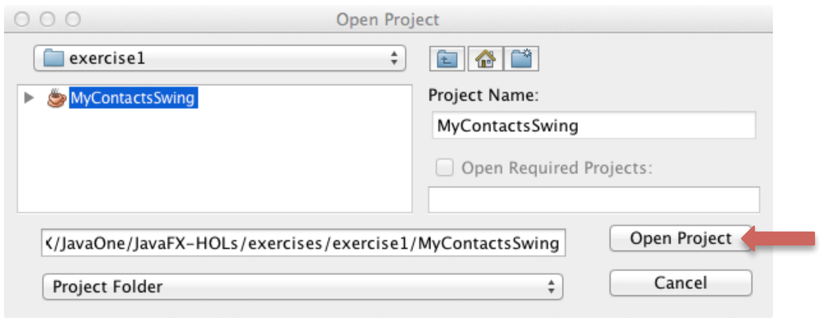
1. From `NetBeans`, open `MyContactsSwing` project. This project it's located inside the `<LAB_ROOT>/exercises/excercise1/` directory.

    - You can either open the project from the File -> Open Project menu.

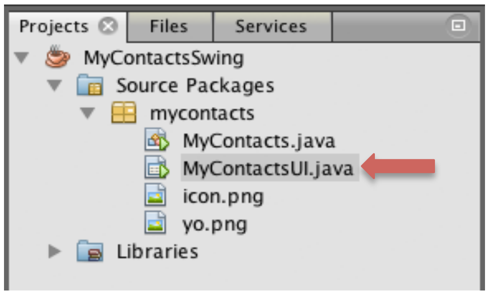




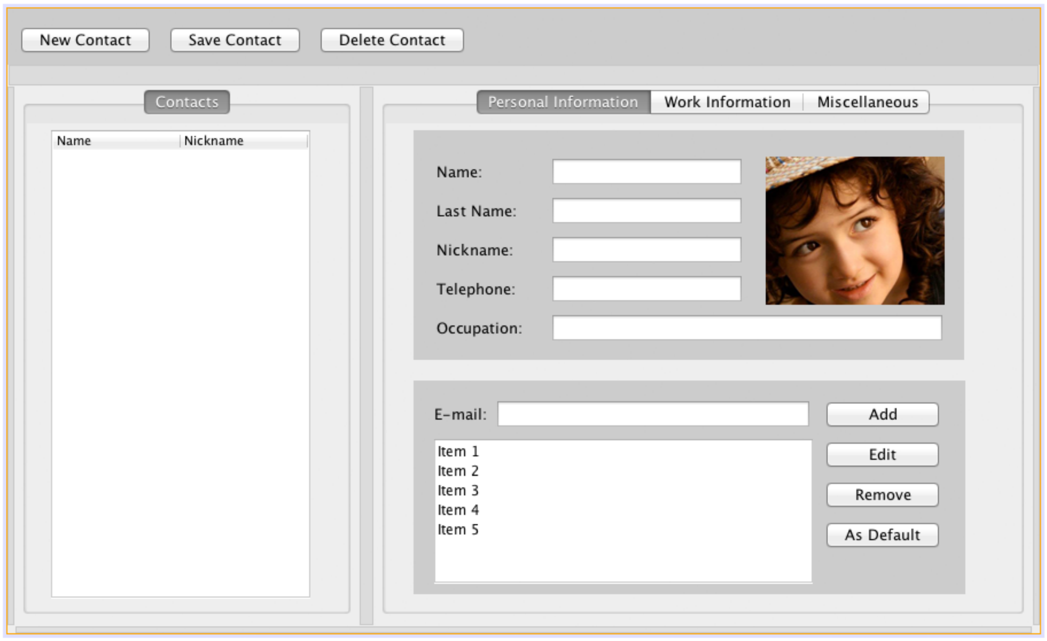
    - Or open the project through the open project icon.
    - Select MyContactsSwing.
    - Click on "Open Project"



2. From Projects, extends Source Packages -> mycontacts, and double click on MyContactsUI.java



3. See the user interface that we are going to build.

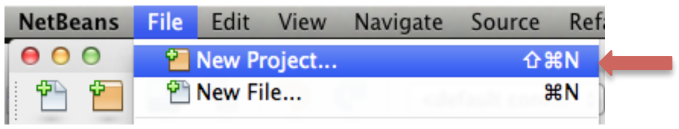


4. If you like, you can go head and run the application using the "Run" icon.

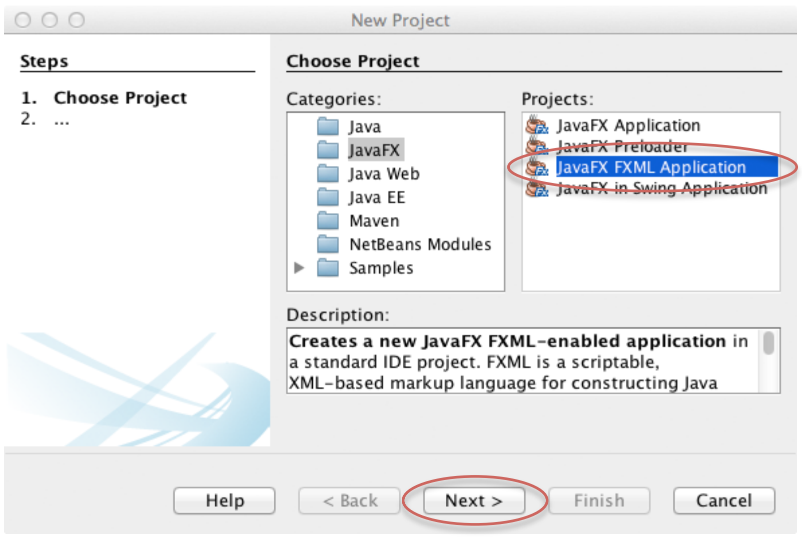
Now that you have an idea of the interface, lets create and even improve it using JavaFX and CSS.

1. Create a JavaFX project, and name it MyContactsJavaFX.
   - Select New Project:

- You can use the New Project icon , or select File -> New Project option from the main menu.

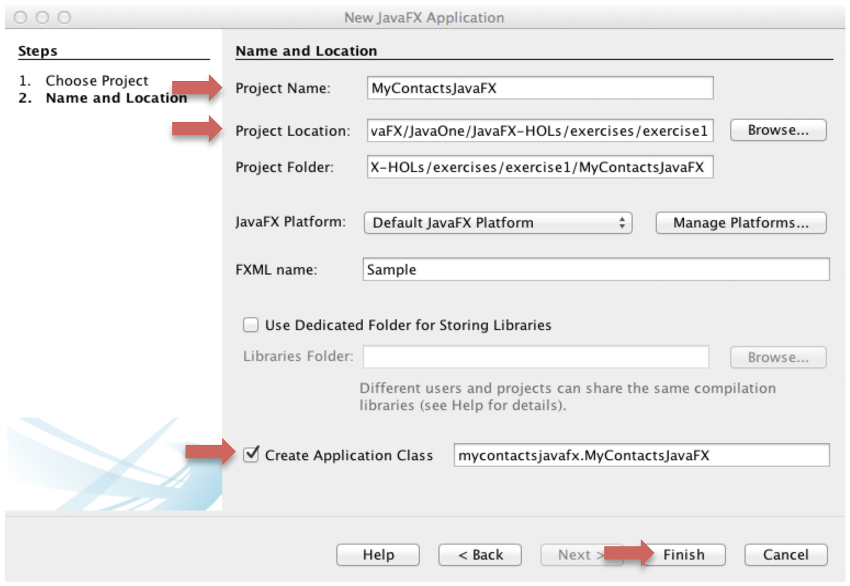


- From the “New Project” window select:
  - From Categories: JavaFX
  - From Projects: JavaFX FXML Application



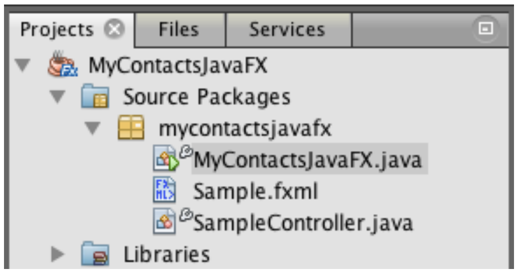
Description:
Creates a new JavaFX FXML-enabled application in a standard IDE project. FXML is a scriptable, XML-based markup language for constructing Java

- You can enter any name you want and select any directory for the project location. For the purpose of this lab, we are going to use MyContactsJavaFX for the project name, and we are going to select exercises/exercise1 as the project location. Make sure the “Create Application Class” option is selected. Then click “Finish”.



Project Name: MyContactsJavaFX
Project Location: vaFX/JavaOne/JavaFX-HOLs/exercises/exercise1
Project Folder: X-HOLs/exercises/exercise1/MyContactsJavaFX
JavaFX Platform: Default JavaFX Platform
FXML name: Sample
☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).
☑ Create Application Class: mycontactsjavafx.MyContactsJavaFX

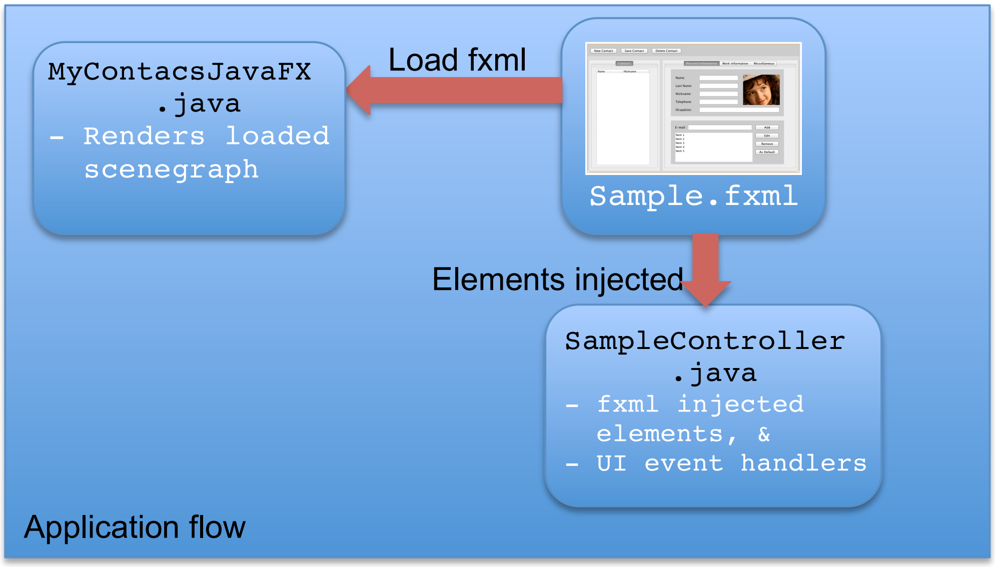
- Your new project gets created, and few files were added to it:
    - MyContactsJavaFX.java: Main class
    - Sample.fxml: FXML file containing the UI definition. Scene Builder will modify this file.
    - SampleController.java. Controller class.
- Next session will explain in details how these files work together.



- Run MyContactsJavaFX project.

1.2. How does the application work?

Before we go any further in the exercise, lets understand the flow of the application: The user interface is defined in the sample.fxml file. There is a SampleController that will be injected with fxml elements, and will define any required event handlers for the interface. The main class is MyContactsJavaFX class and it's in charge of loading the fxml file, displaying the interface and starting the application.



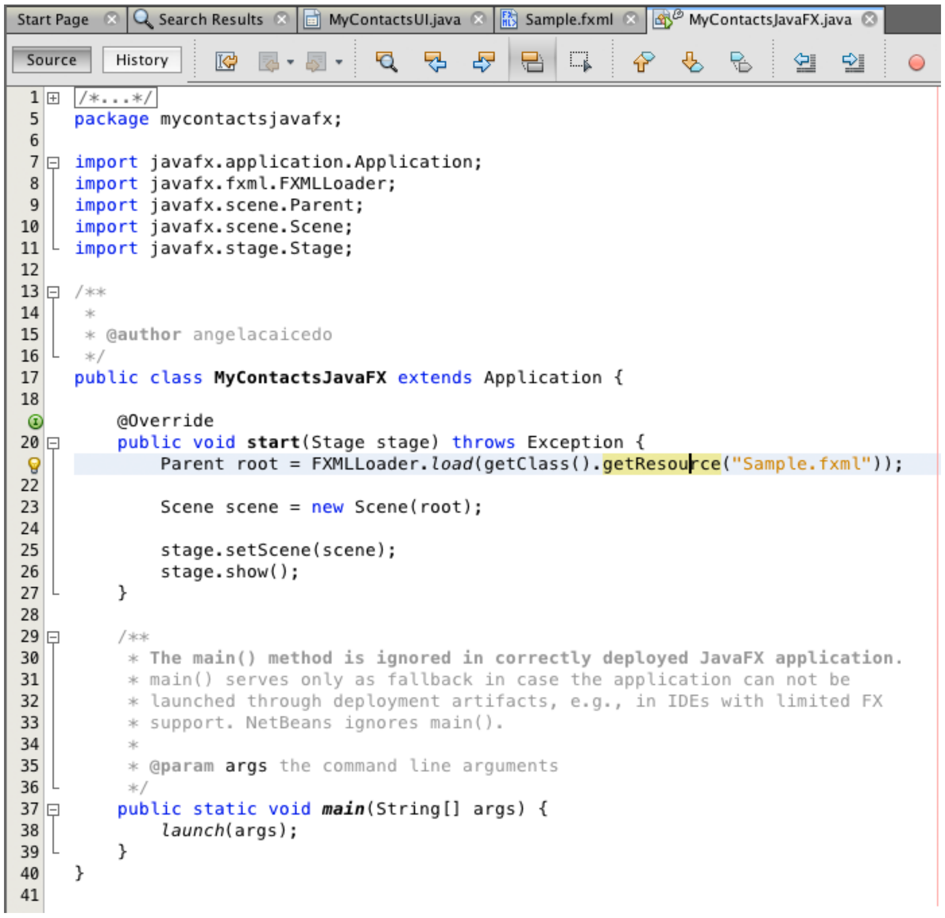
- Double click on MyContactsJavaFX to view the code.
- `MyContactsJavaFX.java` is the main class. It extends the `Application` class (line 17th), as it's required for every JavaFX application.

  We can also observe the `start` method (line 20-27). This method loads the FXML file. In this exercise, we will be using the Scene Builder to create and modify the user interface, and this fxml file will be updated accordingly.

  Once the file is loaded, the method returns a `Parent` object (`root`), which represents the main container defined in the FXML file, usually an AnchorPane.

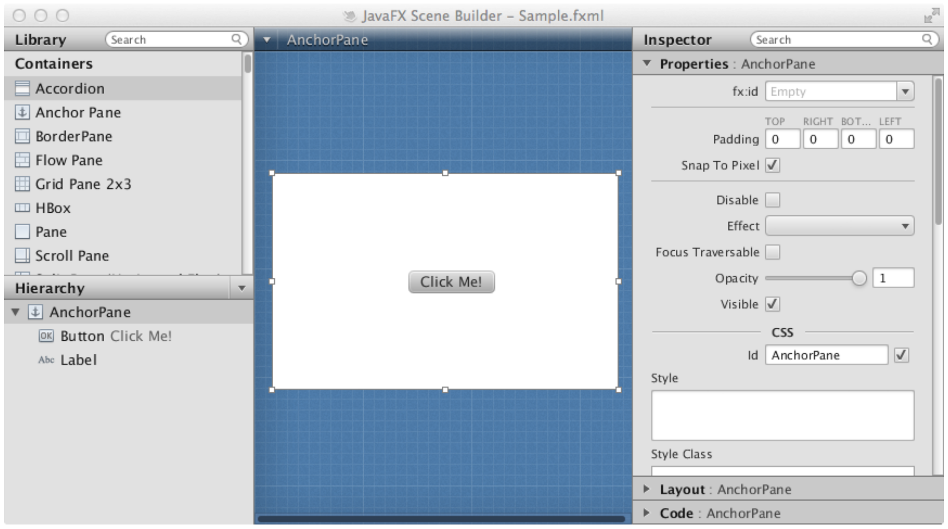
  In line 23 a new Scene is created, and the scenegraph's root is provided. Finally, lines 25-26 set the scene to the stage and show the stage.

Finally, we have the main method (lines37-39), to launch the application



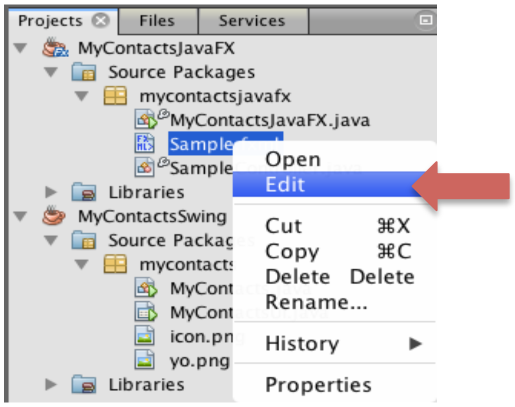
```
 1  /*...*/
 5  package mycontactsjavafx;
 6  
 7  import javafx.application.Application;
 8  import javafx.fxml.FXMLLoader;
 9  import javafx.scene.Parent;
10  import javafx.scene.Scene;
11  import javafx.stage.Stage;
12  
13  /**
14   *
15   * @author angelacaicedo
16   */
17  public class MyContactsJavaFX extends Application {
18  
        
        @Override
20      public void start(Stage stage) throws Exception {
            Parent root = FXMLLoader.load(getClass().getResource("Sample.fxml"));
22  
23          Scene scene = new Scene(root);
24  
25          stage.setScene(scene);
26          stage.show();
27      }
28  
29      /**
30       * The main() method is ignored in correctly deployed JavaFX application.
31       * main() serves only as fallback in case the application can not be
32       * launched through deployment artifacts, e.g., in IDEs with limited FX
33       * support. NetBeans ignores main().
34       *
35       * @param args the command line arguments
36       */
37      public static void main(String[] args) {
38          launch(args);
39      }
40  }
41  
```

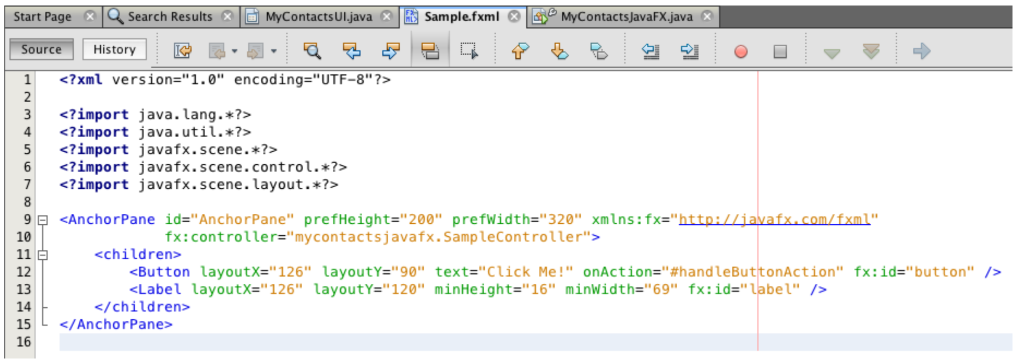
- `Sample.fxml` is a file that defines the user interface for the JavaFX applicaton. This file is written using FXML, a scriptable, XML-based markup language for constructing Java object graphs. This file will be modified using Scene Builder, and it will be loaded by the main application. If you double click into this file, the Scene Builder will be loaded, and the file will be rendered in the tool's design area.



fx:id Empty
Padding TOP 0 RIGHT 0 BOT... 0 LEFT 0
Snap To Pixel ✓
Disable
Effect
Focus Traversable
Opacity 1
Visible ✓
CSS
Id AnchorPane ✓
Style
Style Class

To edit the file manually, you need to select the file and right click to it, then select "Edit". This allows you to open the file for editing within NetBeans.





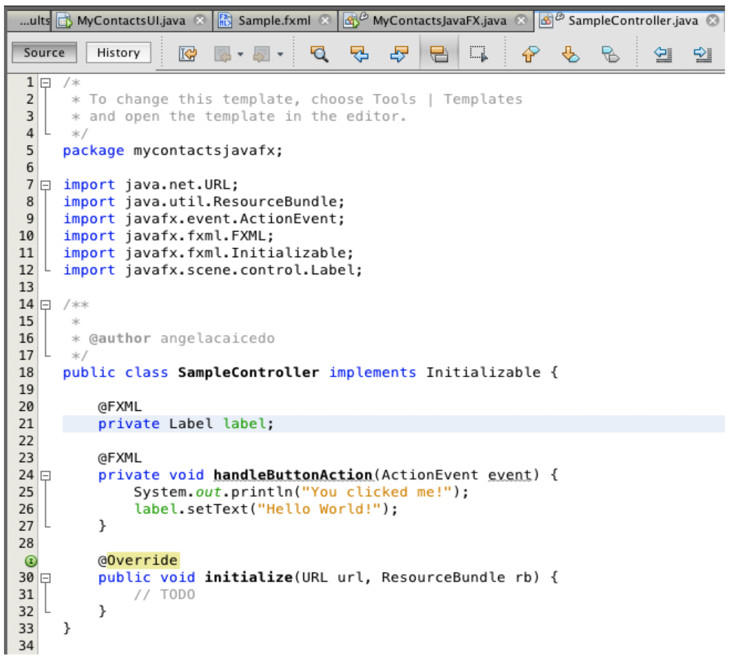
```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <?import java.lang.*?>
4   <?import java.util.*?>
5   <?import javafx.scene.*?>
6   <?import javafx.scene.control.*?>
7   <?import javafx.scene.layout.*?>
8
9   <AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" xmlns:fx="http://javafx.com/fxml"
10              fx:controller="mycontactsjavafx.SampleController">
11      <children>
12          <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction" fx:id="button" />
13          <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
14      </children>
15  </AnchorPane>
16
```

From the code, you can see an `AnchorPane` (line 9) as the main container, and two children: a `Button` (line 12) and a `Label` (line 13).

`AnchorPane` also specifies a controller (line 10), which allow us to connect the fxml file with the java code. Don't worry about it yet; we will explain it in more detail later on.

- `SampleController.java`. This java class will be used to inject the fxml elements declared by sample.fxml. Line 20-21 shows how `label` gets injected from the fxml file using the @FXML annotation.

  The controller also defines methods that will be handling events (mouse events, key events and likes). Lines 23-27 show a method handler, `handleButtonAction`. This method will be called when the button gets an action: it will print in the console and it'll update the label's text to "Hello World".



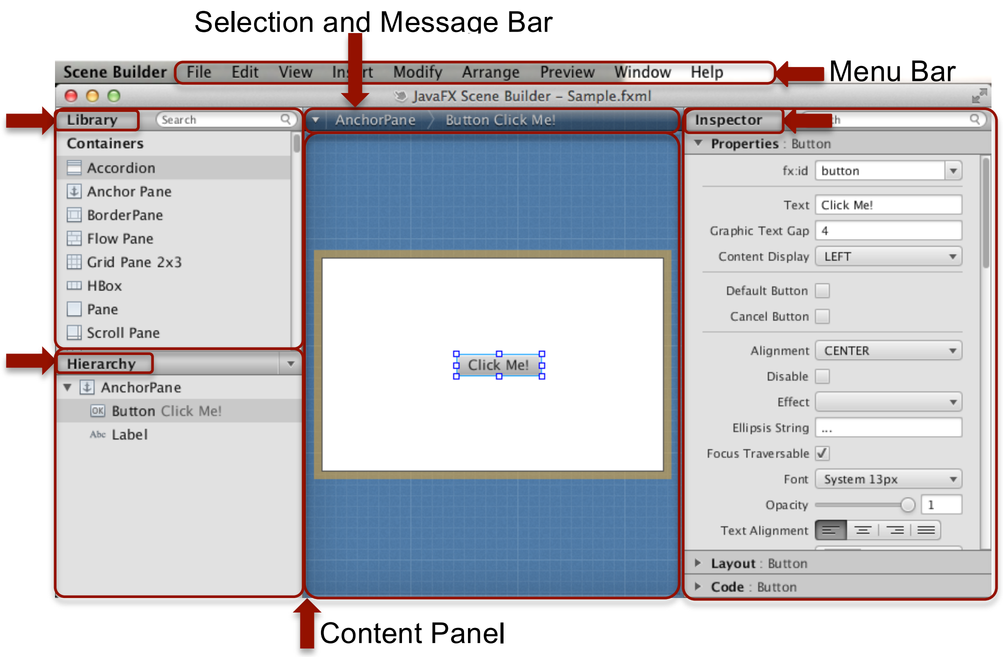
```
 1  /*
 2   * To change this template, choose Tools | Templates
 3   * and open the template in the editor.
 4   */
 5  package mycontactsjavafx;
 6  
 7  import java.net.URL;
 8  import java.util.ResourceBundle;
 9  import javafx.event.ActionEvent;
10  import javafx.fxml.FXML;
11  import javafx.fxml.Initializable;
12  import javafx.scene.control.Label;
13  
14  /**
15   *
16   * @author angelacaicedo
17   */
18  public class SampleController implements Initializable {
19  
20      @FXML
21      private Label label;
22  
23      @FXML
24      private void handleButtonAction(ActionEvent event) {
25          System.out.println("You clicked me!");
26          label.setText("Hello World!");
27      }
28  
        @Override
30      public void initialize(URL url, ResourceBundle rb) {
31          // TODO
32      }
33  }
34  
```

Before we go any further lets see how Scene Builder works, and how it generates the FXML file.

- Scene Builder should be open with Sample.fxml. (If you don't see it, double click the sample.fxml file to reopen it).

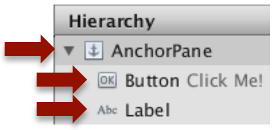
  Scene builder has few panels:
  - **Menu Bar:** Provides access to the menu of commands available in JavaFX Scene Builder.
  - **Selection and Message Bar:** Displays the path to a selected element. It also displays any error or status messages.

- **Content Panel:** The scene container for the GUI elements that make up your FXML layout.
- **Library Panel:** Lists the available JavaFX GUI elements or controls that you can use to build your FXML layout. You select the GUI elements from this panel and add them to the Content panel or the Hierarchy panel.
- **Hierarchy Panel:** Displays a tree view representation of the FXML layout that you are building in the Content panel. Elements that are not visible in the Content panel can be placed in focus by using the Hierarchy panel.
- **Inspector Panel:** Contains the Properties, Layout, and Code sections. The Properties and Layout sections help you manage the properties of the selected UI element in the Content panel or in the Hierarchy panel. The Code section enables you to manage the controller source information and event handling actions to use for the selected UI element. The Inspector panel also contains a Search text field that enables you to isolate specific properties that you want to modify.

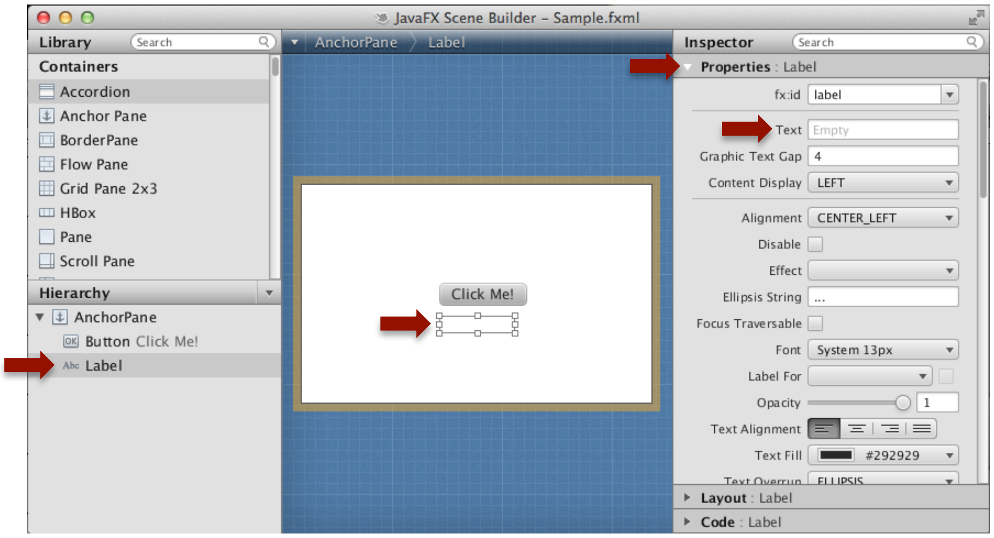


fx:id button
Text Click Me!
Graphic Text Gap 4
Content Display LEFT
Default Button ☐
Cancel Button ☐
Alignment CENTER
Disable ☐
Effect
Ellipsis String ...
Focus Traversable ✓
Font System 13px
Opacity 1
Text Alignment

Now, lets go back to the generated code. Label was one of the elements injected in the controller, but if you pay attention to your content pane, you see that `label` seems to be missing. Because the label doesn't have any text associated with it, you can't really see the label on the screen. We do know that the label is there by looking into the hierarchy's pane, you can clearly see the `Button` and `Label` inside the `AnchorPane`.



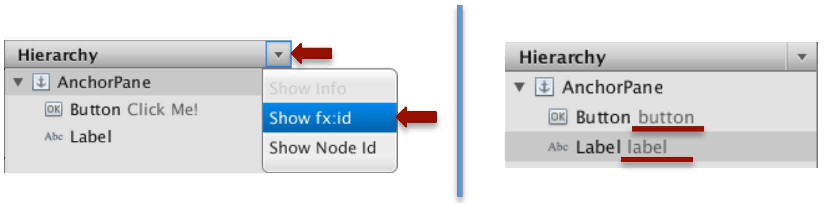
- Click on Label in the hierarchy panel. You can notice that an area in the content panel get highlighted, this is actually where the empty label is located inside the screen. To see the label's properties, click on Properties inside the inspector panel.



fx:id: label
Text: Empty
Graphic Text Gap: 4
Content Display: LEFT
Alignment: CENTER_LEFT
Disable
Effect
Ellipsis String: ...
Focus Traversable
Font: System 13px
Label For
Opacity: 1
Text Alignment
Text Fill: #292929
Text Overrun: ELLIPSIS

There is one property called `fx:id`. This is the id that will be use in the java controller for the injection of the label, so this `fx:id` has to match with the controller's `Label` definition. Also notice that the text property of the label is empty, as we mentioned before.

- If you want to see all the `fx:id` for your components, click on the hierarchy pane's drop down list, and select show fx:id.

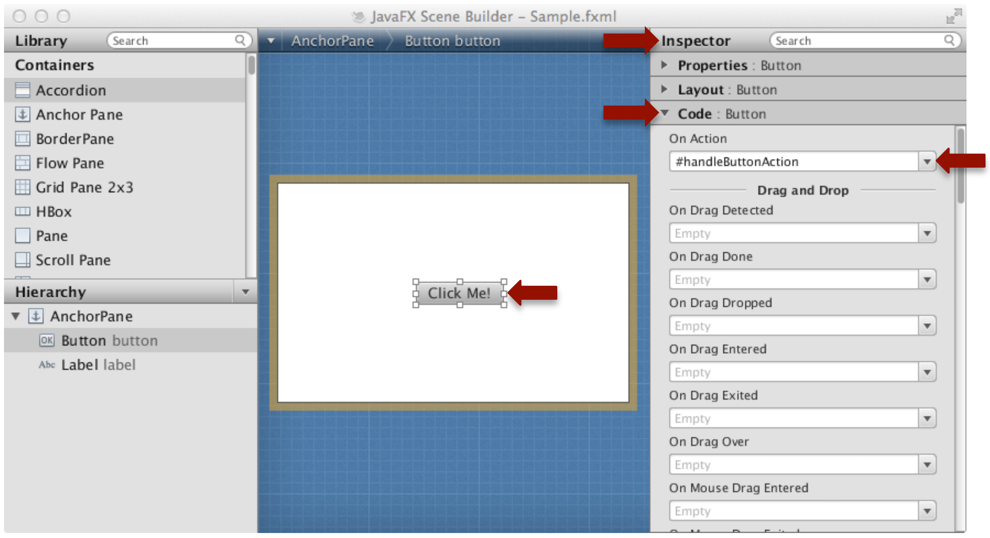


- Select `Button`: you can either click on it in the hierarchy panel or in the content panel.

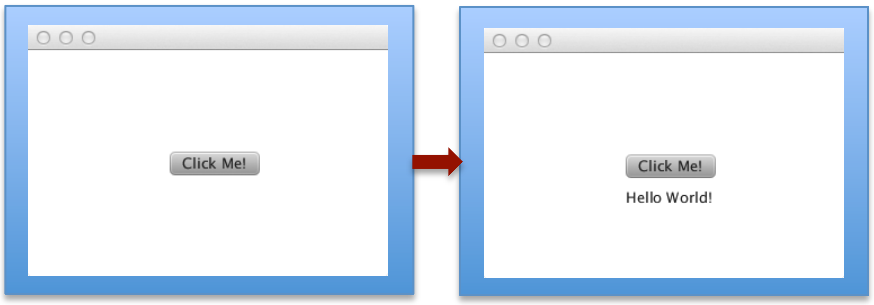
- From the inspector panel select the *Code* section.
- At the top of this section there is a *"On Action"* area, notice the syntax to invoke the `handleButtonAction` on the controller:

`#<methodName>`

This means, whenever the button gets the action event, the `handleButtonAction` method in the controller class will be invoked.



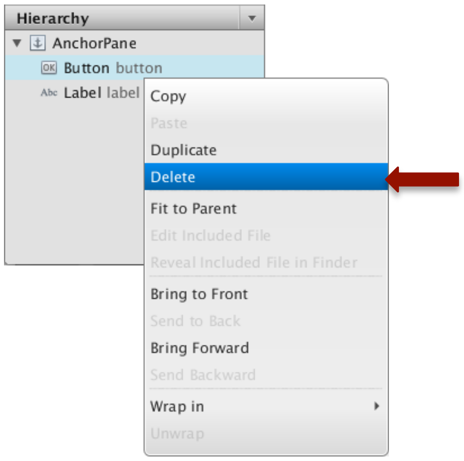
Now lets go back to NetBeans and run `MyContactsJavaFX` project. Click on the button and see how the label gets updated.



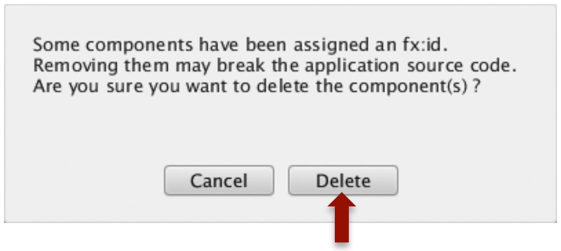
1.3. Designing your New Screen

Now, we are ready to start creating our new interface using Scene Builder.

- Make sure Scene Builder is open, if not, just double-click on `Sample.fxml`
- Delete `button` and `label`.
    - From the hierarchy panel, right click on `button` and select Delete option.

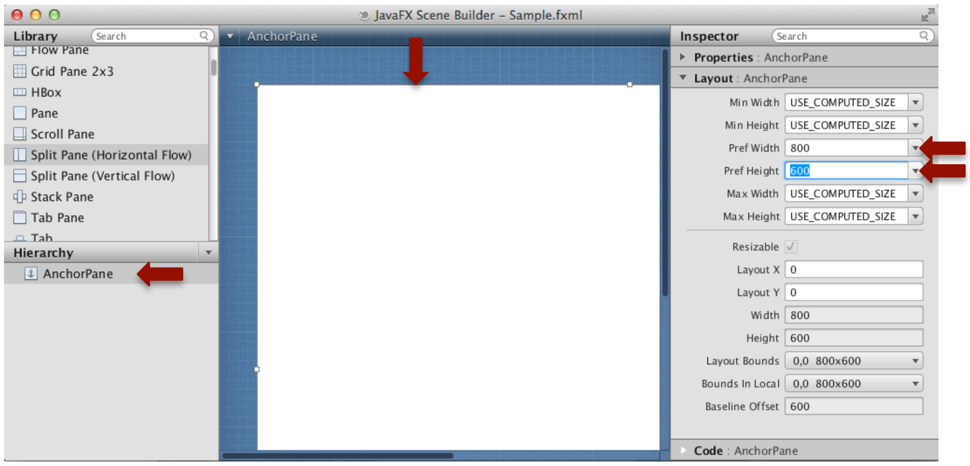


    - Because `button` has an assigned `fx:id`, we get a confirmation message for this delete action. Go head and click on `Delete`, we will clean up the controller class later.


Some components have been assigned an fx:id.
Removing them may break the application source code.
Are you sure you want to delete the component(s) ?

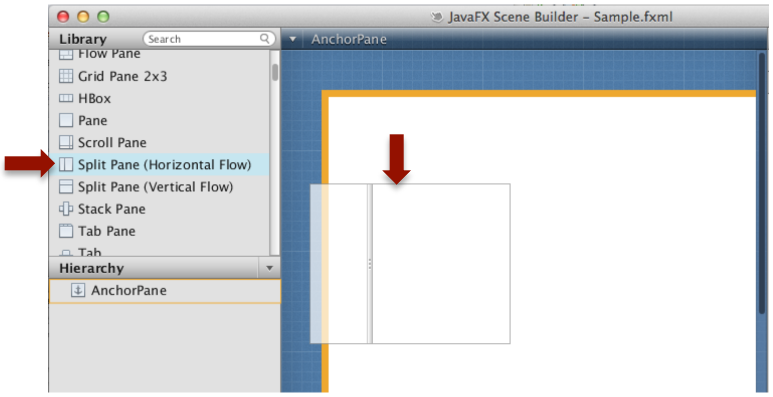
    - Repeat the same actions for `label.`

- Resize your screen.
    - Select `AnchorPane` either from the hierarchy panel or the content area.
    - In the inspector panel, expand `Layout` group.
    - Set the `Pref Width` to 800, and the `Pref Height` to 600

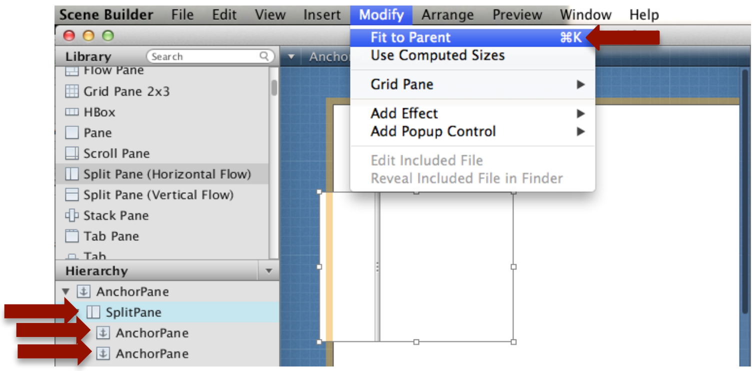


Min Width USE_COMPUTED_SIZE
Min Height USE_COMPUTED_SIZE
Pref Width 800
Pref Height 600
Max Width USE_COMPUTED_SIZE
Max Height USE_COMPUTED_SIZE
Resizable ✓
Layout X 0
Layout Y 0
Width 800
Height 600
Layout Bounds 0,0 800x600
Bounds In Local 0,0 800x600
Baseline Offset 600

- Add a `Split Pane`.
    - From Library panel, scroll down and select `Split Pane (Horizontal Flow).`
    - Drag it and drop it inside the `AnchorPane`, either in the content panel or in the hierarchy panel. Notice how the destination container gets highlighted in orange as you move around. This allows you to know exactly where you are going to drop the component.

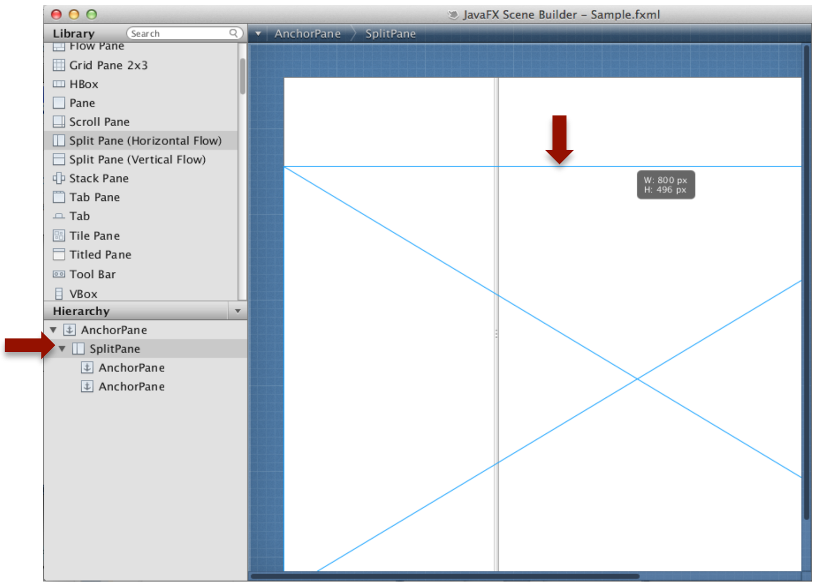


- Resize the split pane to fit the entire `AnchorPane`.
    - You can manually resize the split pane or select "Modify -> Fit to Parent" from the main menu.

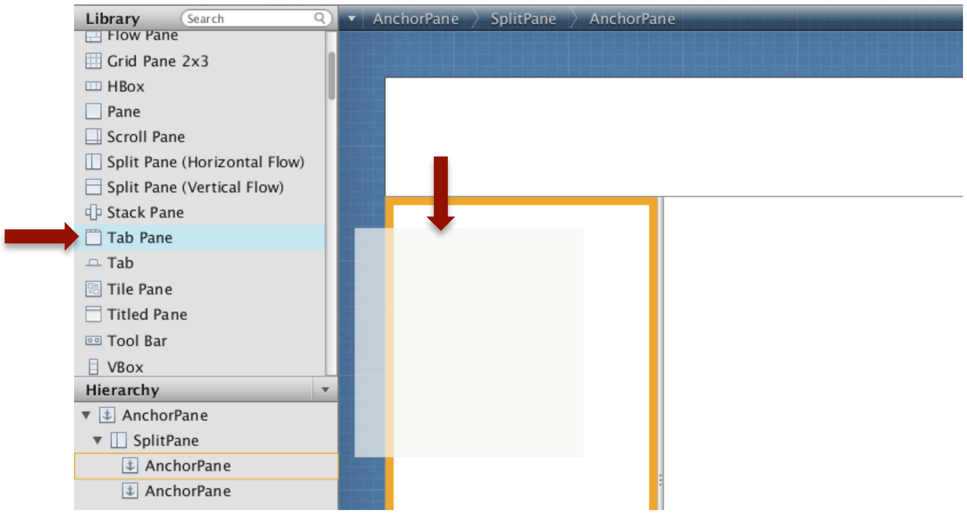


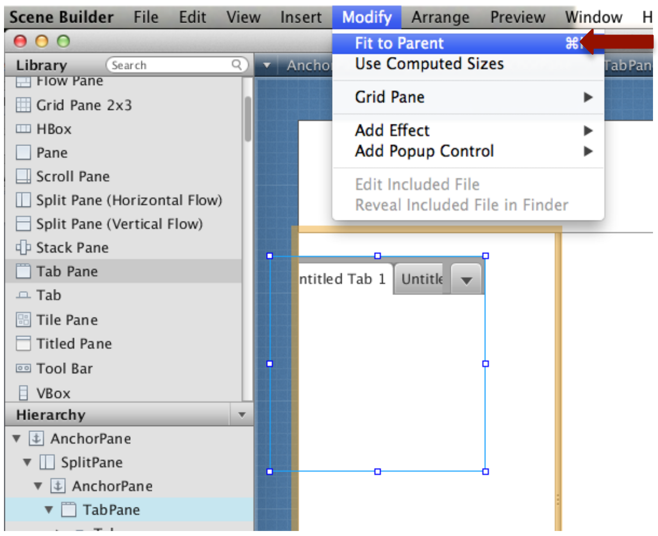
Notice inside the hierarchy panel how the ui components start creating a tree structure. See how `SplitPane` has two inner `AnchorPanes` one on the right and one on the left. Drag the `SplitPane` divider and make `AnchorPane` on the left side to occupy 30% or the entire screen.

- Make up space for the button's bar at the top.
    - Selecting the correct panel to resize can be tricky, especially if we have more than one container. For this reason we recommend selecting the `SplitPane` directly from the hierarchy panel.
    - Drag down the `SplitPane's` top border, to create a small space at the top for the buttons.



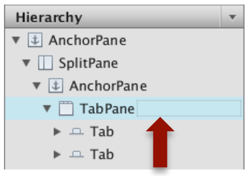
- Insert two `Tab Panes`, one per each `AnchorPane` inside the `SplitPane.` Make sure they fill the container completely.

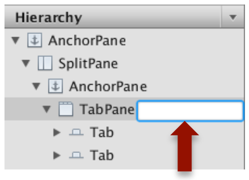


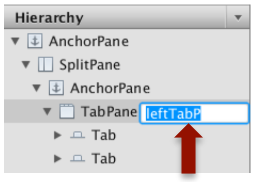


- Set the first `TabPane's` id to `leftTabP`
  - Select the `TabPane`
  - Expand Properties in the Inspector panel
  - Set fx:id to `leftTabP`
  - You can also set the fx:id on the Hierarchy panel
    - Place your mouse just right to the TabPane, and notice the highlighted square. You can click inside it and set the id.
    - If you don't see this area, click on hierarchy panel's drop down list, and select show fx:id

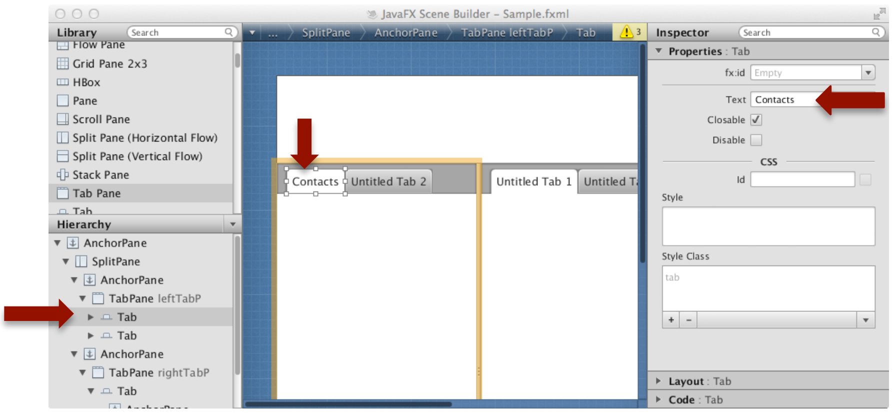
        - Don't worry about the warning on the message bar, it will be fixed when we add the code on the controller.





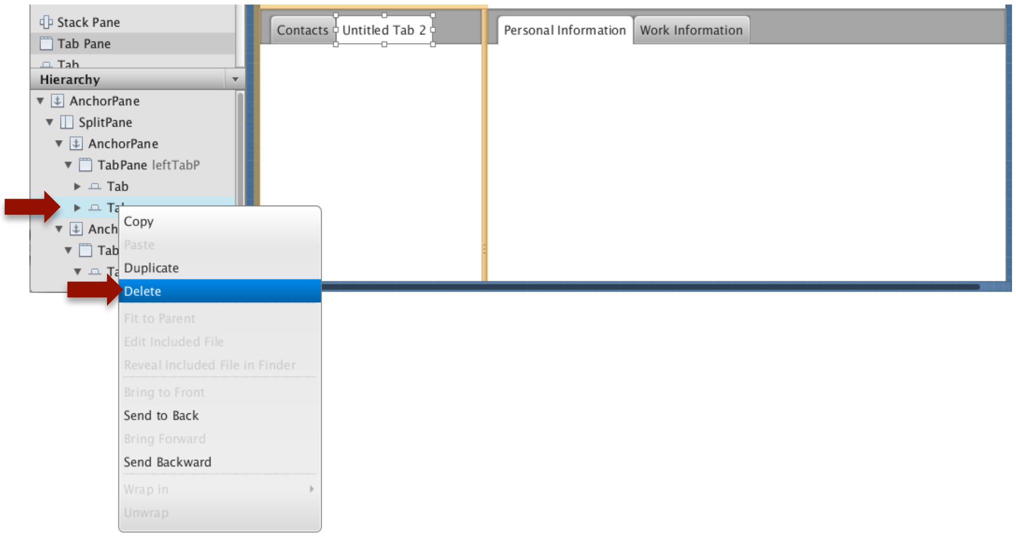


- Rename the second TabPane to rightTabP (Follow previous steps)
- Set the texts for each `Tab`.
    - Select the first Tab in the `leftTabP` TabPane
    - Expand Properties in the Inspector panel
    - Set the Text property to Contacts

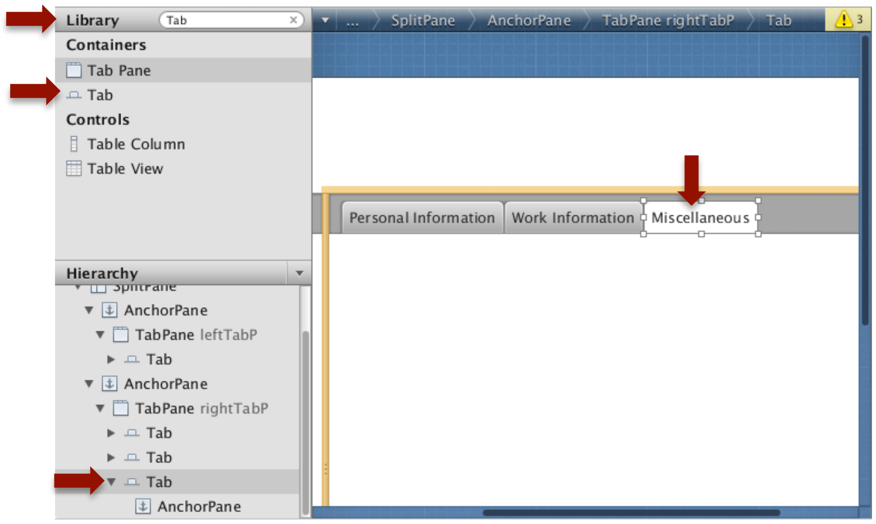


    - You can also double click directly on the Tab's text, to change it's value.
    - For `rightTabP TabPane` use the following texts:
        - "Personal Information"
        - "Work Information"
- From leftTabP delete the second Tab.
    - Inside the hierarchy panel, select the second Tab inside leftTabP and right click on it.

- Select Delete option from the menu.

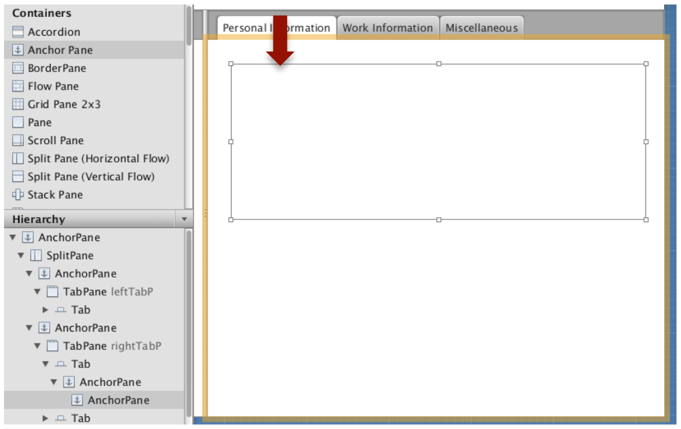


- Add a new Tab to `rightTabP TabPanel`
  - Type "Ta" in the search area of Library Panel
  - Select Tab, drag it and drop it on rightTabP. Notice the new Tab.
  - Rename the Tab's text to Miscellaneous

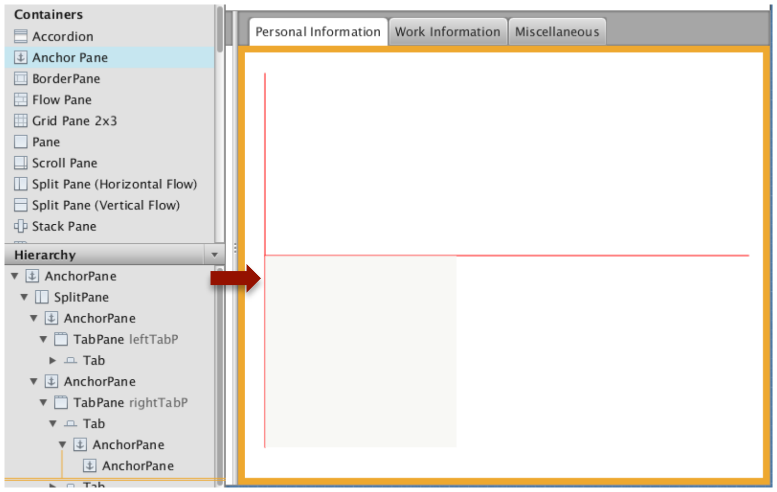


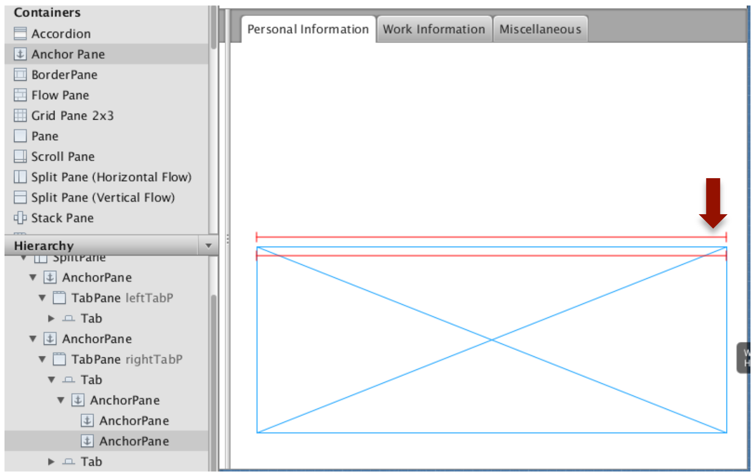
- Lets design the Personal Information `Tab Panel`
  - Click on "Personal Information" tab
  - From the Hierarchy panel, you can notice that this Tab panel already has an `AnchorPane`.
  - Add two new `AnchorPanes` inside the "Personal Information"'s AnchorPane. Make sure these new panels are really inside. Remember to grab the `AnchorPanes` from the library and drop them either in the hierarchy panel or in the content panel.

- Re-arrange them. Remember we are trying to create the same interface we showed at the beginning of this exercise. In this case, these new `AnchorPanes` are been used to group components only.



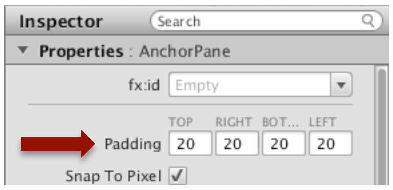
  - Use the guidelines as you drop the components, so they are aligned and they have the same size



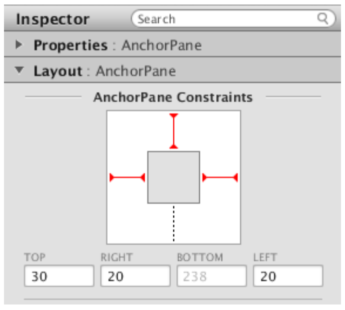


ORACLE®

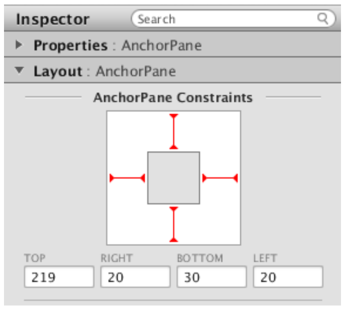
- Set the `AnchorPanes'` padding information
  - Select each `AnchorPane`
  - From Inspector panel extend the properties group.
  - Set the Padding to be 20, 20, 20, 20



- Set the layout information for the `AnchorPanes`.
  - Select the first `AnchorPane` you added to Personal Information `TabPane`.
  - In the Inspector panel extend Layout section.
  - Set constrains to top, right and left. We don't want this `AnchorPane` to grow vertically, as the component that will take most of the space in vertical resizing is the `ListView`, contained in the other `AnchorPane`.

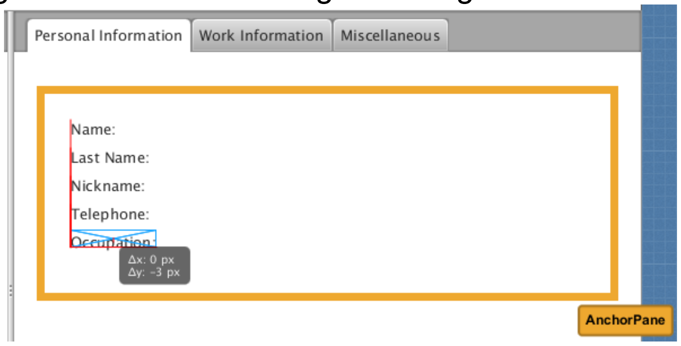


  - Select the second `AnchorPane`, and select the layout constraints to be top, right, bottom and left. We want this panel to grow in all directions.

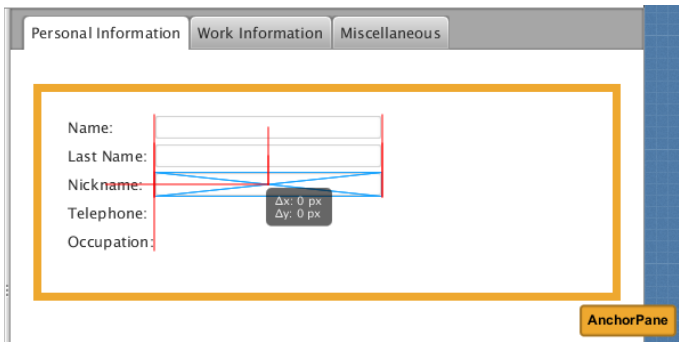


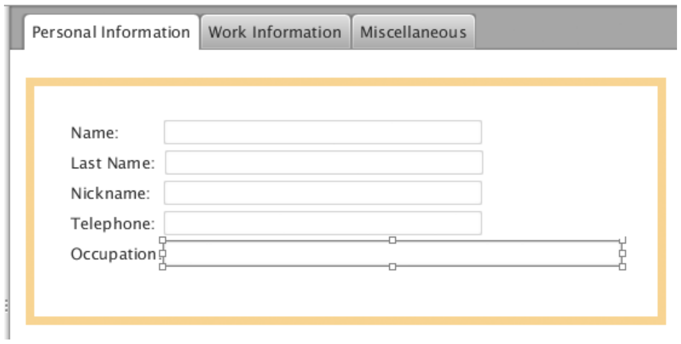
- Add all the UI components for the top `AnchorPane`
  - Add 5 `Labels` with the following displayed text:

        - Name
        - Last Name
        - Nickname
        - Telephone
        - Occupation
    - Align them to the left using the tool guidelines



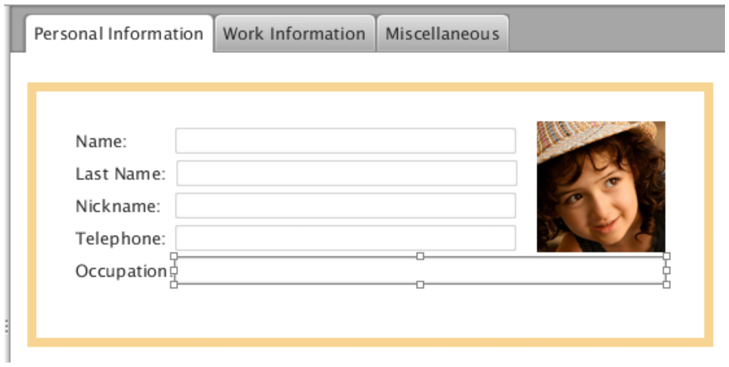
    - Add 5 TextFields and align to the left, leaving a little space for the image. Make sure all the `TextFields` have the same size, except the last one, that will cover the entire horizontal space.



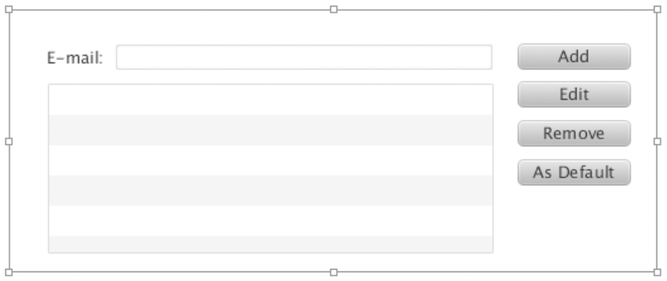


    - Lets include the picture
        - Copy the icon.png file from <LAB_ROOT>/exercises/exercise1/media to <LAB_ROOT>/exercises/exercise1/MyContactsJavaFX/src/mycontactsjavafx/

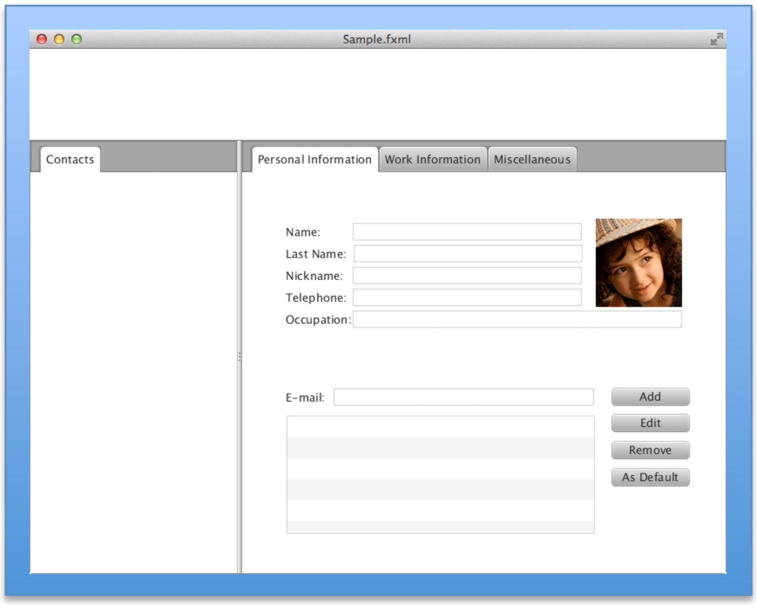
- From the main menu select: File -> Import -> Media
- Select icon.png
- Resize it as appropriate



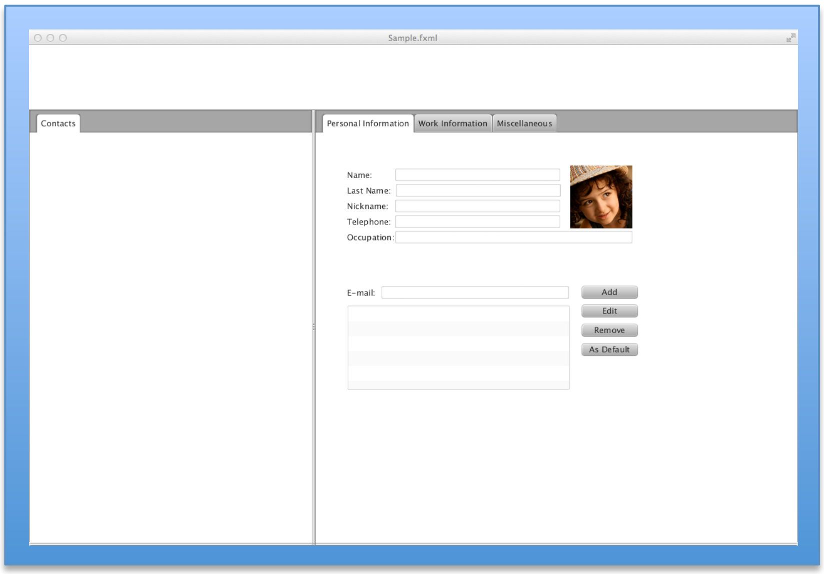
- Now that you got the idea of how to place components, lets create the following display for the lower `AnchorPane`. Remember to use guidelines for helping you to set alignments and component dimensions.



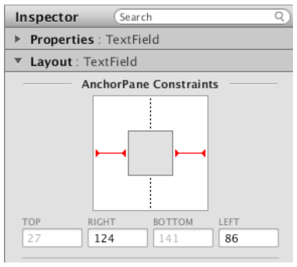
- Lets set the layout information so our application will resize properly.
  - From the main menu, select Preview -> Preview in Window



- The application looks good, but if you try to resize your screen, components don't resize properly.

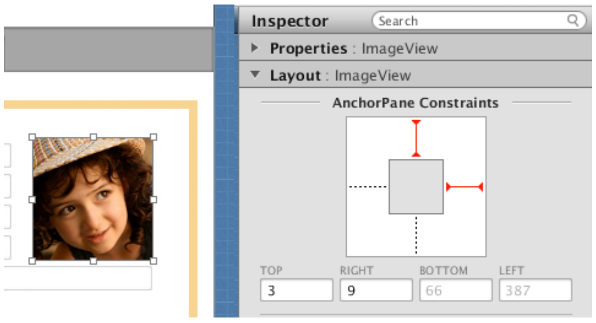


- Select each of the `TextFields` and anchor them to the right and to the left of their container, as we want them to always grow horizontally.



TOP	RIGHT	BOTTOM	LEFT
27	124	141	86

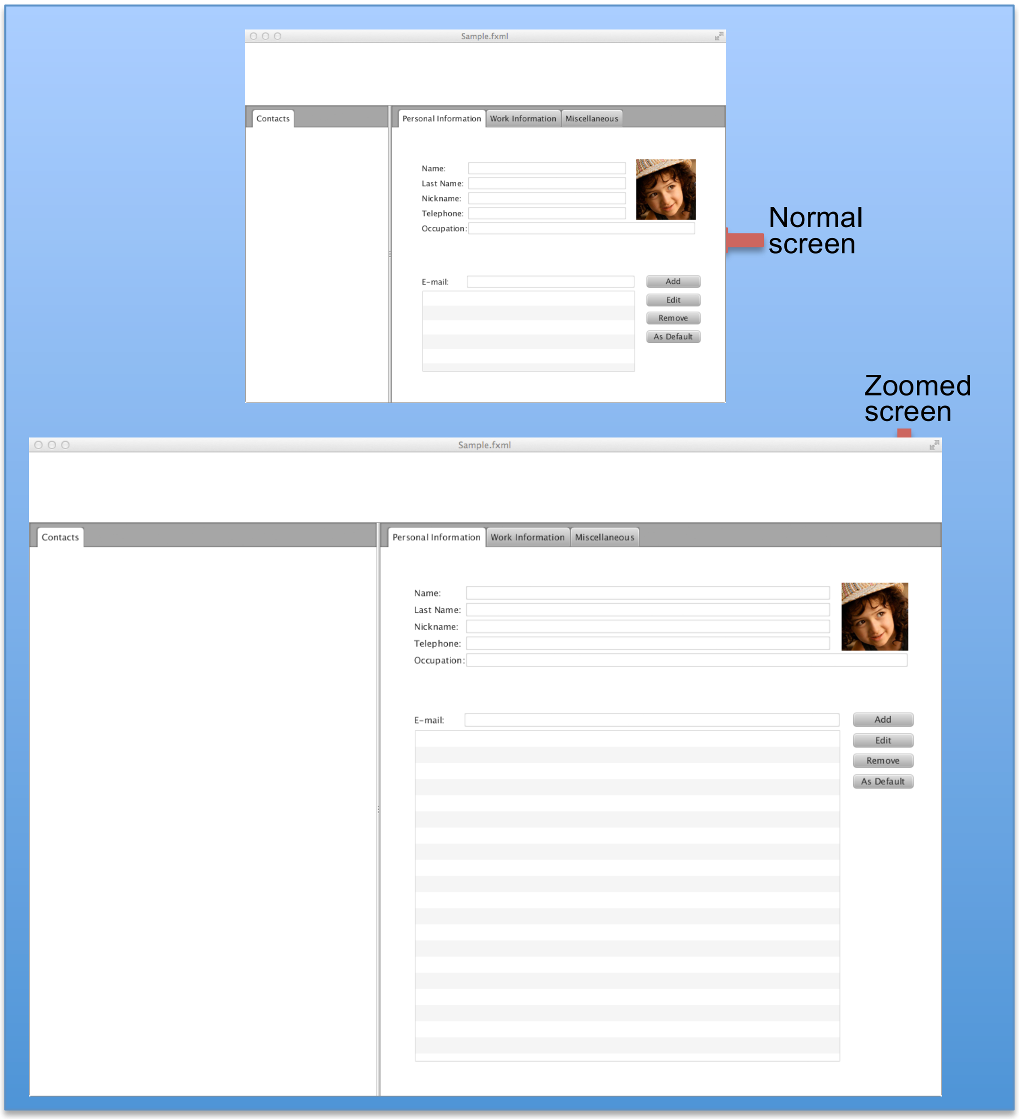
- Now, anchor the image to the container's top and right borders.



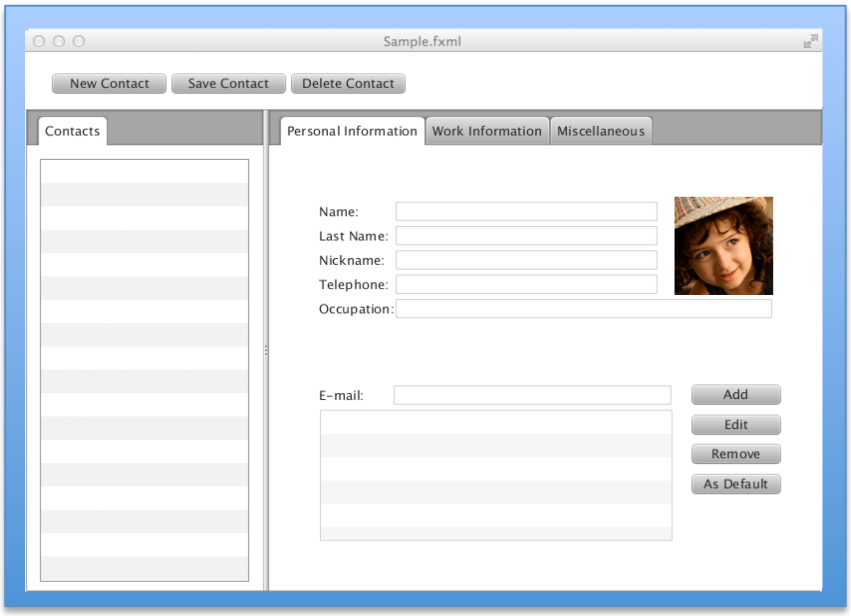
TOP	RIGHT	BOTTOM	LEFT
3	9	66	387

- Set the following anchoring for the second `AnchorPane's` components
  - Anchor all the `Buttons` to the top and right border. (Stay on the right and do not resize)
  - Anchor the `TextField` to the left and right border (Grow horizontally)
  - Anchor the `ListView` to the top, right, bottom and left border. (Grow horizontally and vertically)

- Preview the application again, and make sure it does resize correctly.

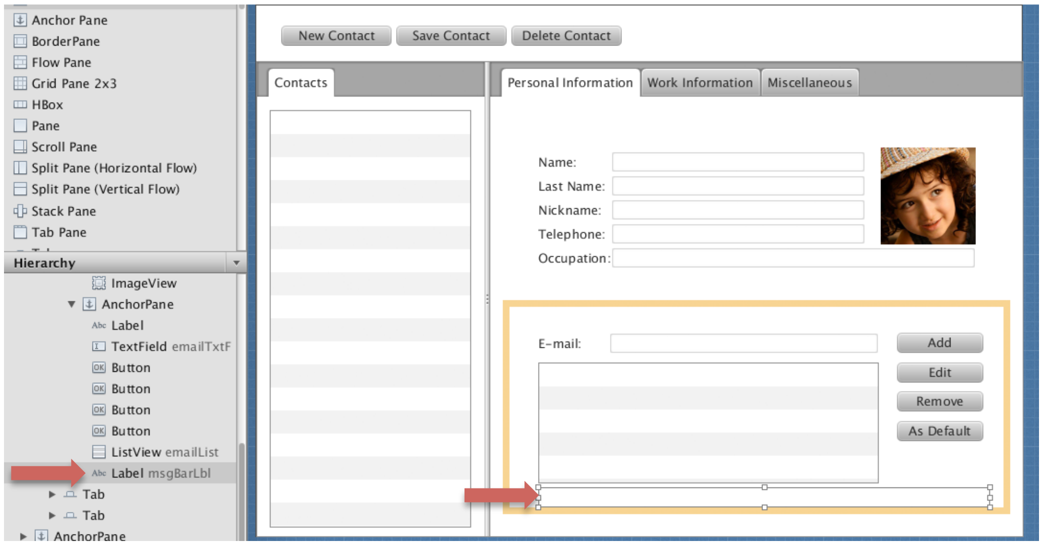


- Now that you have a better understanding of how the Scene Builder works, lets finish the user interface. These are the components that need to be added:
  - Add a `ListView` to the Contacts `TabPane`
  - Add a new `AnchorPane` on the top, to create the menu with `Buttons`. Remember this container won't be visible until we use CSS and apply some nice effects (Exercise 2).
  - In this newly created `AnchorPane` add three `Buttons`: New Contact, Save Contact, and Delete Contact.
    - Select the `Buttons`, and from the main menu, select Arrange -> Wrap In -> HBox. This will create an `HBox` as a new container for the three `Buttons`.



- Preview your screen and make sure it still resize properly.
- Save your changes.
- Add a message bar
  - At the bottom of email's `AnchorPane`, add a `Label` to display messages.
  - Set the new `Label's` text empty

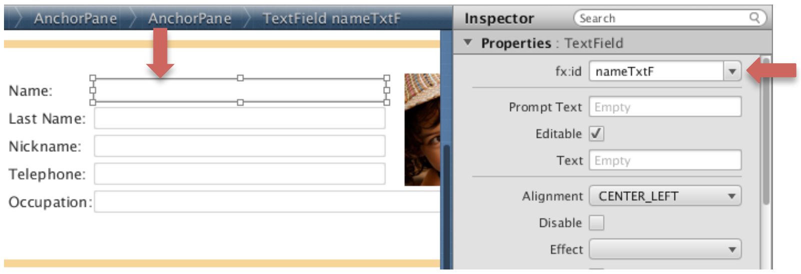
- Set Label's fx:id to msgBarLbl



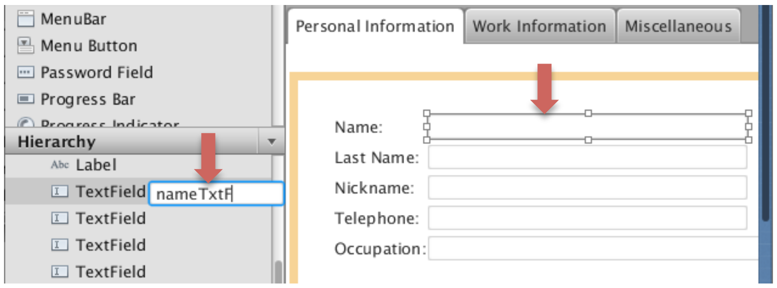
1.4. Programming the controller

In this exercise we are going to program the E-mail section of the UI interface. First at all, to be able to have access the UI components in the controller class, we need to inject the fxml elements, and we need to provide a fx:id for the elements to be injected.

- Provide the fx:id for the components to be injected
  - From Scene Builder, select the `TextField` next to the `Label` Name.
  - Set the fx:id to "nameTxtF"
    - You can set the fx:id inside the Properties section in the Inspector panel



    - You can also set the fx:id by clicking next to the `TextField` in the Hierarchy panel.



  - Set the following fx:ids
    - `TextField` next to "*Last Name*" `Label`: lastNameTxtF
    - `TextField` next to "*Nickname*" `Label`: nicknameTxtF
    - `TextField` next to "*Telephone*" `Label`: phoneTxtF
    - `TextField` next to "*Occupation*" `Label`: occupationTxtF
    - `TextField` next to "*E-mail*" `Label`: emailTxtF
    - `ListView`: emailList
    - Label: msgBarLbl

- Inject the fxml elements into the controller class
    - Lets go back to NetBeans
    - Make sure MyContactsJavaFX project is open
    - Open SampleController.java for editing
    - In the generated code you can notice an annotation @FXML (line 20). This `Label` named "*label*" will be injected with the fxml element that has "*label*" as fx:id (defined in the Sample.fxml file)
    - Now, in our case we got rid of that label from the beginning, and new UI components were added.
        - Remove lines 20th and 21st

          ```
          @FXML
          private TextField nameTxtF;
          ```

        - Remove line 26th.

          ```
          label.setText("Hello World!");
          ```

        - Inject all the fxml elements that were provided with a fx:id in the previous exercises
            - The fx:ids `rightTabP` and `leftTabP` are not really required by the controller, so you can go head and remove this ids. This will get rid of any warning messages in the Scene Builder.

```
22. @FXML
23. private TextField nameTxtF;
24. @FXML
25. private TextField lastNameTxtF;
26. @FXML
27. private TextField nicknameTxtF;
28. @FXML
29. private TextField phoneTxtF;
30. @FXML
31. private TextField occupationTxtF;
32. @FXML
33. private TextField emailTxtF;
34. @FXML
35. private TextArea emailTxtA;
36. @FXML
37. private Label msgBarLbl;
```

- Verify that all the fxml elements were injected without any issues. You can use assertions for this:

    -
      ```
      assert nameTxtF != null : "fx:id=\"nameTxtF\" was not
                                 injected: check your FXML file.";
      ```

    - The Controller class implements the `Initializable` interface, then checking for issues with the injection of components in the `initialize` method is the right thing to do.

```
@Override
public void initialize(URL url, ResourceBundle rb) {
  assert nameTxtF != null : "fx:id=\"nameTxtF\" was not injected:
                             check your FXML file.";
  assert lastNameTxtF != null : "fx:id=\"lastNameTxtF\" was not injected:
                                 check your FXML file.";
  assert nicknameTxtF != null : "fx:id=\"nicknameTxtF\" was not injected:
                                 check your FXML file.";
  assert phoneTxtF != null : "fx:id=\"phoneTxtF\" was not injected:
                              check your FXML file.";
  assert occupationTxtF != null :"fx:id=\"occupationTxtF\" was not injected:
                                  check your FXML file.";
  assert emailTxtF != null : "fx:id=\"emailTxtF\" was not injected:
                              check your FXML file.";
  assert emailTxtA != null : "fx:id=\"emailTxtA\" was not injected:
                              check your FXML file.";
}
```

- Initialize the `ListView` `emailList` with some emails.
    - The elements of the `ListView` are stored in an `ObservableList`. This `ObservableList` is automatically observed by the `ListView`, such that any changes that occur inside the `ObservableList` will be automatically shown in the `ListView` itself. Because we created the `ListView` from the Scene Builder, passing the `ObservableList` into the `ListView` constructor is not feasible, therefore we need to call the `setItems` method at the controller's initialization process.
        - Create the `ObservableList` with some predefined emails.

```
private ObservableList<String> emailObservable =
           FXCollections.observableArrayList(
                    "angela.caicedo@oracle.com",
                    "angela.caicedo@xxx.com",
                    "angela.caicedo@vmm.com");
```

        - Invoke the `setItems` method on the `ListView` to set the list's items.

```
emailList.setItems(emailObservable);
```

- Program events handlers
  - In this part of the exercise we are going to create four methods to take care of the `ActionEvents` for each `Button` in the E-mail panel:
    - Lets program “Add” `Button`:
      - The method needs to grab the text typed in the `emailTxtF`, and add it to the `ListView` emailList. To be able to add an item to the `emailList`, we need to do it through the `ObservableList` it’s bind to, in our case the collection is `emailObservable`.
      - `emailTxtF` contents should be cleaned up.
      - The `msgBarLbl` should be updated with a message about the action.

```
@FXML
private void addAction(ActionEvent event) {
      //If user typed an email
      if(!emailTxtF.getText().isEmpty()){
          //grab the text from the TextField and add it to the
          //ObservableList binded to emailList
          emailObservable.add(emailTxtF.getText());
          //Display the appropriate message in the msgBarLbl
          msgBarLbl.setText(emailTxtF.getText() + " has been
                                                  Added!!!");
          //Clean up the emailTxtF
          emailTxtF.setText("");
      }else{
          //If user didn’t type anything ask for email to be
          //entered first
          msgBarLbl.setText("Type Your Email First!...");
      }
  }
```

    - Lets program “Edit” `Button`:
      - The method needs to grab the selected item from `ListView emailList`. You need to get the selected item through the list’s selection model.
      - `emailTxtF` contents should be cleaned up.

- The `msgBarLbl` should be updated with a message about the action.

```
@FXML
private void editAction(ActionEvent event) {
    //Get the list's selected items through its SelectionModel
    ObservableList<String> email = emailList.
                         getSelectionModel().getSelectedItems();
    //If there is selected item in the list
    if(email.size() > 0){
         //Update the message bar
         msgBarLbl.setText(email.get(0) + " seleted for
                                            editing!...");
         //Displayed the selected item in emailTxtF
         emailTxtF.setText(email.get(0));
         //Remove the item from the ListView, it will be added
         //again after editing
         emailObservable.removeAll(email.get(0));
    }else{
         //If no item was selected in the ListView, show it
         //in the message bar
         msgBarLbl.setText("Select an E-mail to be Edit!...");
    }
}
```

- Lets program "Remove" `Button`:
  - The method needs to grab the selected item from `ListView emailList`. You need to get the selected item through the list's selection model.
  - The selected item gets remove from the `ListView's ObsevableList`
  - The `msgBarLbl` should be updated with a message about the action.

```
@FXML
private void removeAction(ActionEvent event) {
   //Get the list's selected items through its SelectionModel
   ObservableList<String> email = emailList.
                        getSelectionModel().getSelectedItems();
  //If there is selected item in the list
  if(email.size() > 0){
      //Update the message bar
       msgBarLbl.setText(email.get(0) + " has been
                                       Deleted!...");
       //Remove item from emaiObservable list
      emailObservable.removeAll(email.get(0));
   }else{
      //If no item was selected in the ListView, show it
      //in the message bar
      msgBarLbl.setText("Select an E-mail to be
                                       Deleted!...");
   }
}
```

- Add a `FadeTransition` to the `msgBarLbl Label`.

- msgBarLbl has been really useful to keep the user informed about the status of the application. But we want the message to be displayed for a short period of time, and then disappear. One easy way to do this is using `FadeTransitions`.
  We can define a `FadeTransition` for the `Label`: when an action happens, the user will be notified, and a message will be displayed just for 3 seconds, after that, the message will disappear. The trick here is to play with the `opacity` of the `Label` component.
  So lets define our new Transition.

```
private  FadeTransition labelFading;
...

labelFading = new FadeTransition(Duration.millis(3000),
                                 msgBarLbl);
labelFading.setFromValue(1.0);
labelFading.setToValue(0.0);
```

- Now, every time there is an action, and a message gets displayed, we will play this animation.

```
labelFading.play();
```

- One last touch, lets make the msgBarLbl red.

```
        msgBarLbl.setTextFill(Color.RED);
```

SampleController.java now looks like this:

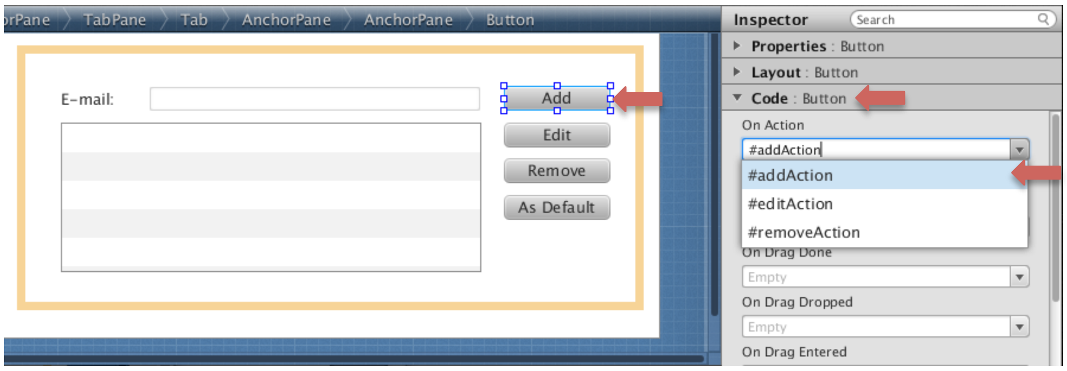
```
package mycontactsjavafx;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.animation.FadeTransition;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.TextField;
import javafx.scene.paint.Color;
import javafx.util.Duration;

/
*
@author angelacaicedo
```

```
*/
public class SampleController implements Initializable {

  @FXML
  private TextField nameTxtF;
  @FXML
  private TextField lastNameTxtF;
  @FXML
  private TextField nicknameTxtF;
  @FXML
  private TextField phoneTxtF;
  @FXML
  private TextField occupationTxtF;
  @FXML
  private TextField emailTxtF;
  @FXML
  private ListView emailList;
  @FXML
  private Label msgBarLbl;


  private  ObservableList<String> emailObservable =
           FXCollections.observableArrayList("angela.caicedo@oracle.com",
                                             "angela.caicedo@xxx.com",
                                             "angela.caicedo@vmm.com");
  private  FadeTransition labelFading;

  @FXML
  private void addAction(ActionEvent event) {
    if(!emailTxtF.getText().isEmpty()){
       emailObservable.add(emailTxtF.getText());
       msgBarLbl.setText(emailTxtF.getText() + " has been Added!!!");
       emailTxtF.setText("");
    }else{
       msgBarLbl.setText("Type Your Email First!...");
    }
    labelFading.play();
  }

  @FXML
  private void removeAction(ActionEvent event) {
    ObservableList<String> email = emailList.getSelectionModel().
                                             getSelectedItems();
    if(email.size() > 0){
      msgBarLbl.setText(email.get(0) + " has been Deleted!...");
      emailObservable.removeAll(email.get(0));
    }else{
      msgBarLbl.setText("Select an E-mail to be Deleted!...");
    }
    labelFading.play();
  }

  @FXML
  private void editAction(ActionEvent event) {
     ObservableList<String> email = emailList.getSelectionModel().
                                             getSelectedItems();
     if(email.size() > 0){
       msgBarLbl.setText(email.get(0) + " seleted for editing!...");
       emailTxtF.setText(email.get(0));
       emailObservable.removeAll(email.get(0));
     }else{
       msgBarLbl.setText("Select an E-mail to be Edit!...");
     }
     labelFading.play();
  }
```

```
    @Override
    public void initialize(URL url, ResourceBundle rb) {
      assert nameTxtF != null : "fx:id=\"nameTxtF\" was not injected:
                                check your FXML file.";
      assert lastNameTxtF != null :"fx:id=\"lastNameTxtF\" was not injected:
                                check your FXML file.";
      assert nicknameTxtF != null :"fx:id=\"nicknameTxtF\" was not injected:
                                check your FXML file.";
      assert phoneTxtF != null : "fx:id=\"phoneTxtF\" was not injected:
                              check your FXML file.";
      assert occupationTxtF != null : "fx:id=\"occupationTxtF\" was not
                                   injected: check your FXML file.";
      assert emailTxtF != null : "fx:id=\"emailTxtF\" was not injected:
                              check your FXML file.";
      assert emailList != null : "fx:id=\"emailTxtA\" was not injected:
                              check your FXML file.";
      assert msgBarLbl != null : "fx:id=\"msgBarLbl\" was not injected:
                              check your FXML file.";

      labelFading = new FadeTransition(Duration.millis(3000), msgBarLbl);
      labelFading.setFromValue(1.0);
      labelFading.setToValue(0.0);

      msgBarLbl.setTextFill(Color.RED);
      emailList.setItems(emailObservable);
    }
  }
```

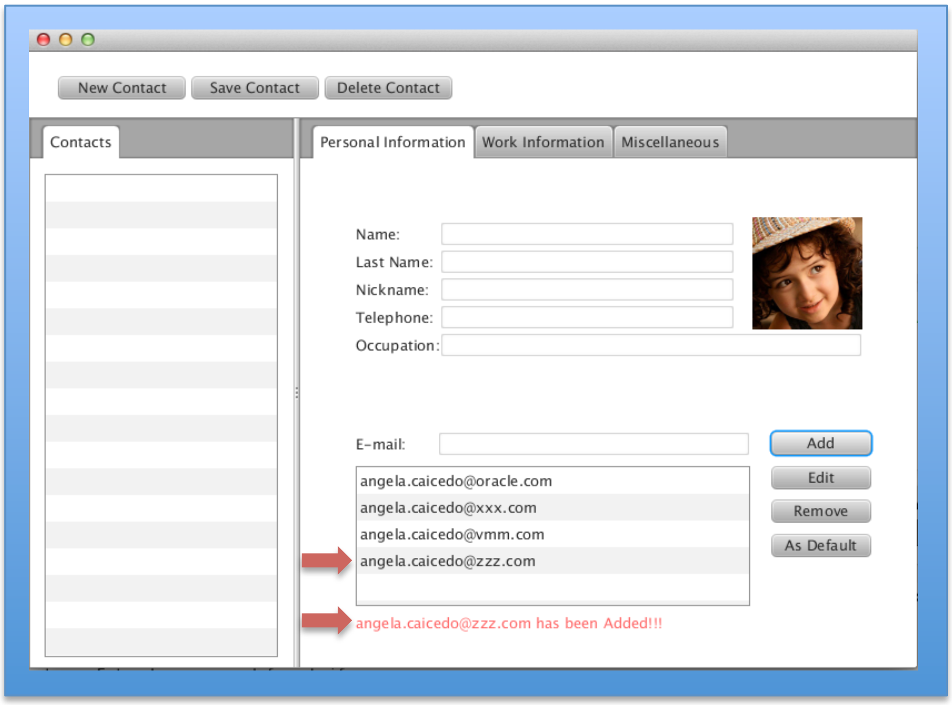
- Associate the `Buttons` to it's action handlers
  - Now that we have defined some action handle methods in the controller class, we can associate them to their corresponding `Button` in the Scene Builder
  - Go to Scene Builder and make sure Sample.fxml is open.
  - Select the "Add" `Button`
  - From the Inspector panel, expand the Code section
  - From the "On Action" section, expand the drop-down list, and see that now you have 3 methods available to invoke. Select the `#addAction` method

Note: If you don't see the methods, go back to NetBeans and make sure you save your project.



- Repeat the same steps to associate "Edit" `Button` with `#editAction` method and "Remove" `Button` with `#removeAction` method.
- Save the fxml file.

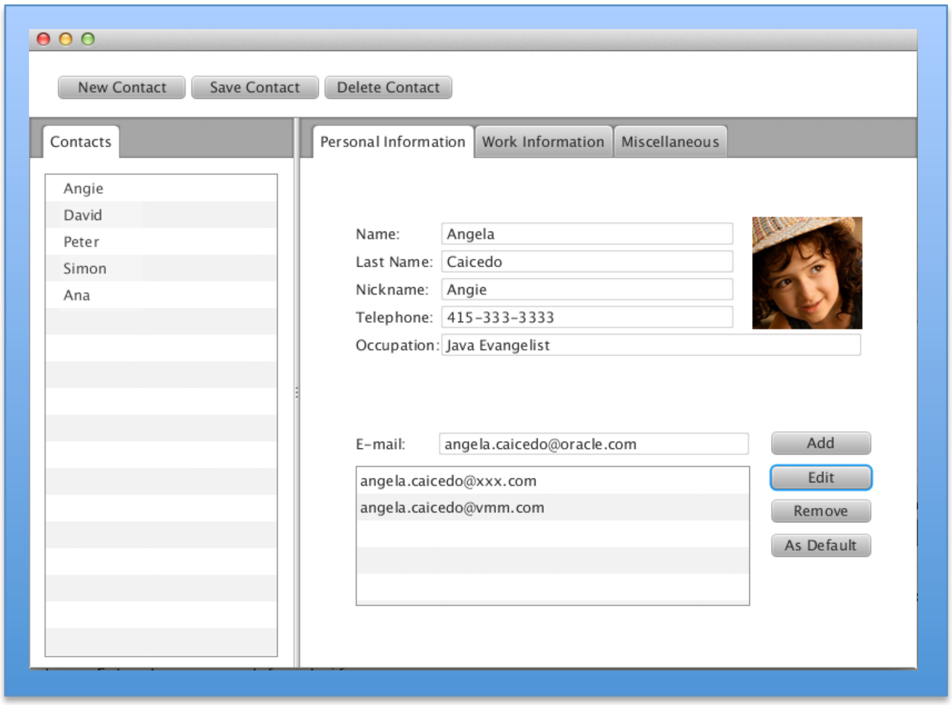
• Run the application from NetBeans, and test the buttons.



angela.caicedo@oracle.com
angela.caicedo@xxx.com
angela.caicedo@vmm.com
angela.caicedo@zzz.com

angela.caicedo@zzz.com has been Added!!!

• Finish the Application.
  - Create a `Contact` class with `name, lastName, nickname, telephone, occupation` and `email` fields.
  - Program the event handler methods for the New Contact, Save Contact and Delete Contact buttons:
    - "New Contact": Clears the screen for new contact information
    - "Save Contact": Save data and add the nickname to the contact list
    - "Delete Contact": Delete the selected contact from contact list
  - Add a new `Label`, next to the `Buttons` to show the status of Contacts updates. Create the appropriate animations to hide the message bar after 3seconds.
  - Add the required fx:ids



Angie
David
Peter
Simon
Ana

Name:	Angela
Last Name:	Caicedo
Nickname:	Angie
Telephone:	415-333-3333
Occupation:	Java Evangelist
E-mail:	angela.caicedo@oracle.com

angela.caicedo@xxx.com
angela.caicedo@vmm.com

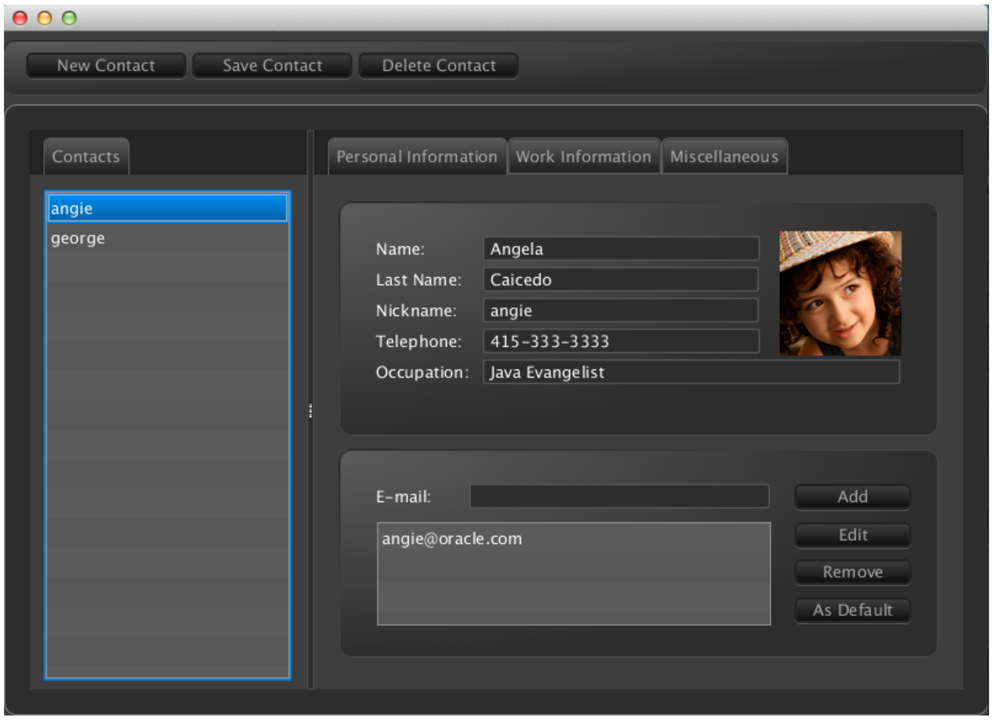
2. Styling your application using CSS

You can create a custom look, also called a skin, for your JavaFX application with cascading style sheets (CSS). CSS contains style definitions that control the look of user interface elements.

JavaFX CSS are based on the W3C CSS version 2.1 specification (available at http://www.w3.org/TR/CSS21/) with some additions from current work on version 3 of the specification and some extensions that support specific JavaFX features.

In the previous exercise we build a JavaFX application using NetBeans and JavaFX Scene Builder, in this exercise we are going to create a custom look for it. You can notice from the snapshots bellow that they have their own flavors, and to change from one to another is just a change in the CSS files they are using. No a single line of code needs to be re-written.

You can see a Black theme, a Midnight Blue theme and a Dark Red theme. In this exercise we are going to see how to create this css files, and how to apply them to the ui components.

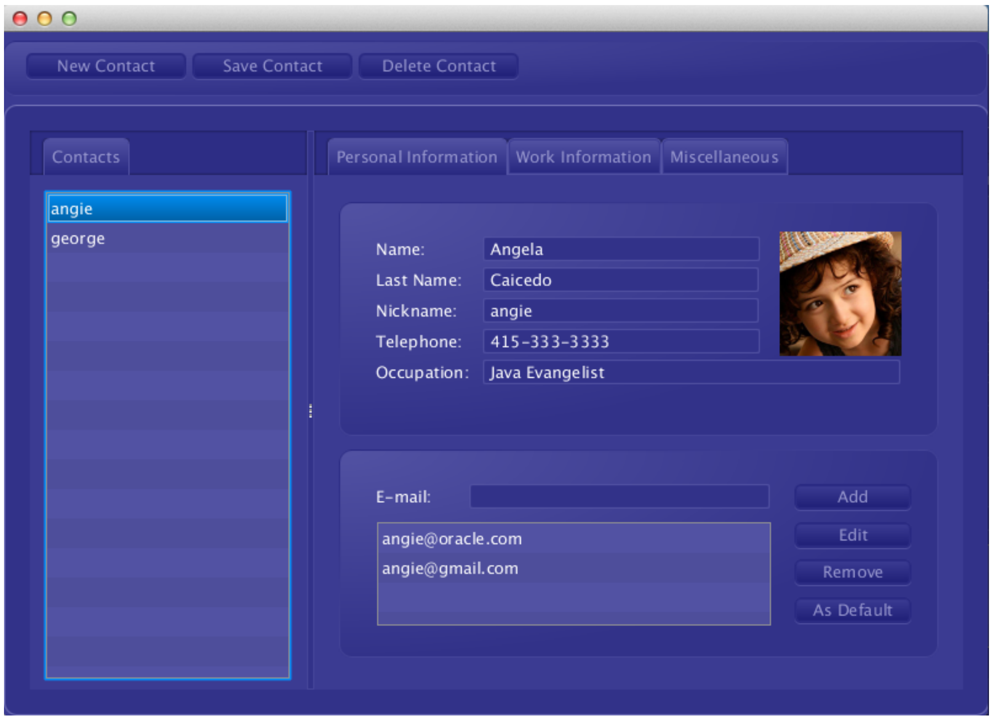


angie
george

Name: Angela
Last Name: Caicedo
Nickname: angie
Telephone: 415-333-3333
Occupation: Java Evangelist

E-mail:
angie@oracle.com

Black screen

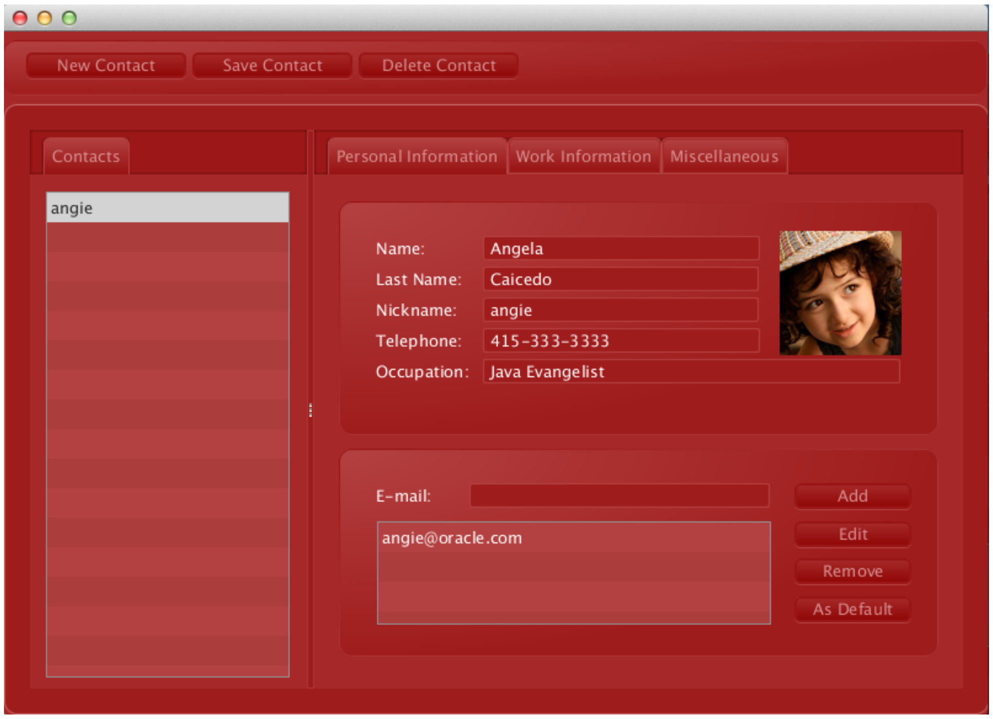


angie
george

Name: Angela
Last Name: Caicedo
Nickname: angie
Telephone: 415-333-3333
Occupation: Java Evangelist

E-mail:
angie@oracle.com
angie@gmail.com

Midnight Blue



angie

Name: Angela
Last Name: Caicedo
Nickname: angie
Telephone: 415-333-3333
Occupation: Java Evangelist

E-mail:
angie@oracle.com

Dark red

2.1 Getting started with CSS

JavaFX applications use a default style sheet, caspian.css, which can be found in the JavaFX runtime jar file. You can create your own style sheets to override the default one.

First you will need to create a file with extension .css and save it in the same directory as the main JavaFX application. This file consist of a set of styles, each style has a group of rules that set the properties of the style. Each style has a name and its definitions are enclosed in braces.

```
.my-gradientpane {
     -fx-background-radius: 10;
     -fx-border-radius: 10;
     -fx-border-width: 1;
     -fx-border-color: darkblue;
     -fx-background-color: blue;
}
```

Style classes correspond to class names, you will find `.button`, or `.label` styles for `Buttons` and `Labels` classes. You can also have compound styles, as some classes include elements that can have their own style definition, for example: `.check-box .label` define the style rules for the label included in the `CheckBoxes`.

The rules for a style definition assign values to properties associated with the class. Rule property names correspond to the names of the properties for a class. The convention for property names with multiple words is to separate the words with a hyphen (-). Property names for styles in JavaFX style sheets are preceded by -fx-. Property names and values are separated by a colon (:). Rules are terminated with a semicolon (;). For example -fx-background-color: black;

The .root style class is applied to the root node of the Scene instance. Because all nodes in the scene graph are a descendant of the root node, styles in the .root style class can be applied to any node.

One of the greatest advantages of using CSS is that you can override the default values of a component, just include the style in your style sheet and assign different properties. For example, if you want all your buttons to have a blue background, a yellow border and rounded corners, rather than the default Caspian style, you will have:

```
.button{
   -fx-background-color: blue;
   -fx-border-color: yellow;
   -fx-border-radius: 5;
   -fx-padding: 3 6 6 6;
}
```

You can also create a class style by adding a definition for it to your style sheet. For example:

```
.my-button{
   -fx-text-fill: green;
   -fx-border-color: darkgreen;
   -fx-border-radius: 25;
   -fx-padding: 3 6 6 6;
}
```

Now, you can assign the style on your code:

```
Button myStyledButton = new Button("Click Me");
myStyledButton.getStyleClass().add("my-button");
```

JavaFX Scene Builder has an entire section for assigning the style classes to your components, and we will study that in more details in the next section.

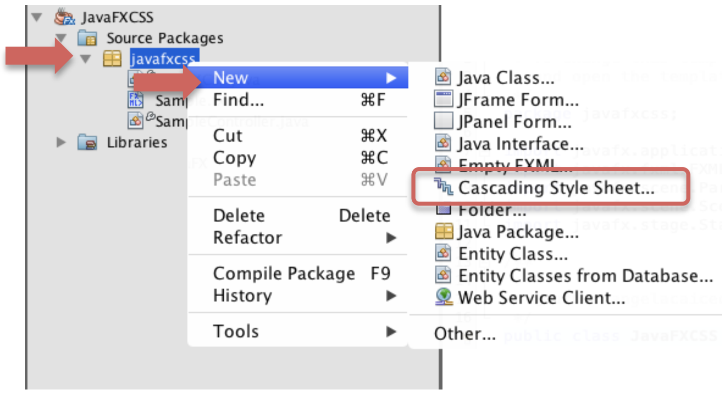
If you don't want to go for a CSS file, you also have the option of setting style properties for a node within the code of your application. Rules set within the code take precedence over styles from a style sheet. For example

```
Button myStyledButton = new Button("Click Me");
myStyledButton.setStyle("-fx-background-color: blue;
                         -fx-border-color: yellow;”);
```

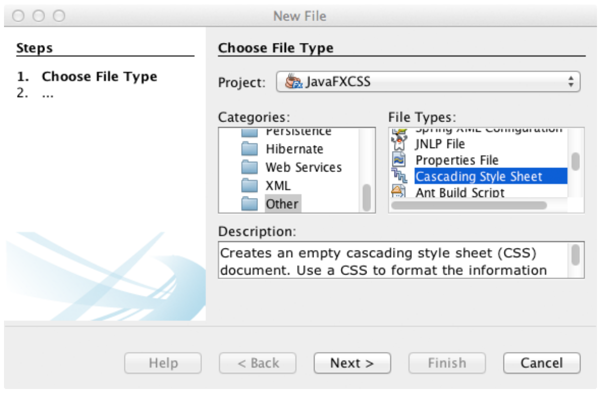
2.2 My First Application using CSS:

- Create a JavaFX project, and name it JavaFXCSS.
    - Select New Project: You can use the New Project icon , or select File -> New Project option from the main menu. (For more detailed instructions you can go to the previous exercise)
    - From the "New Project" window select:
        - From Categories: JavaFX
        - From Projects: JavaFX FXML Application
    - You can enter any name you want and select any directory for the project location. For the purpose of this lab, we are going to use JavaFXCSS for the project name, and we are going to select exercises/exercise2 as the project location.
    - Make sure the "Create Application Class" is selected.
    - Then click "Finish".

- Create the CSS file
    - Expand your project
    - Right click on the package javafxcss
    - From the popup menu, click on "New ->"
    - Select "Cascading Style Sheet..."

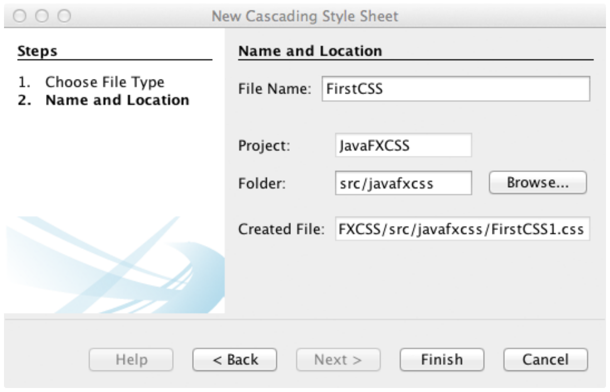


    - If "Cascading Style Sheet..." is not visible, from New, go all the way down to the end of the list and select "Other...".
        - From the New File window, select:
            - From Categories: Other
            - From File Types: Cascading Style Sheet
        - Click on Next



Description:
Creates an empty cascading style sheet (CSS) document. Use a CSS to format the information

- You can select anything you want for File Name, in our case we are using FirstCSS.
- Take the defaults for Project and Folder option
- Click on Finish



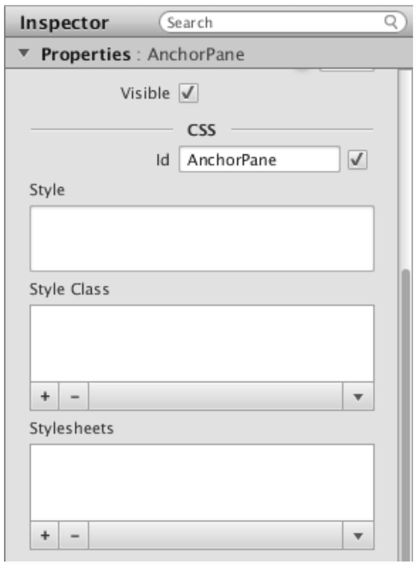
File Name: FirstCSS

Project: JavaFXCSS

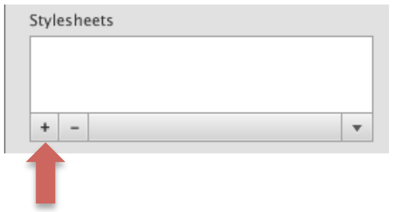
Folder: src/javafxcss

Created File: FXCSS/src/javafxcss/FirstCSS1.css

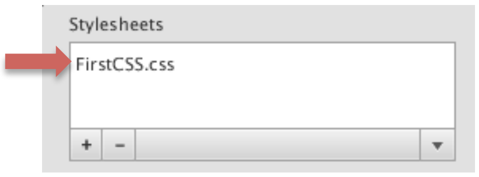
- Associate the CSS file to your JavaFX application using Scene Builder
  - In NetBeans double click on Sample.fxml to open up Scene Builder
  - In the Hierarchy panel, select AnchorPane
  - In the Inspector panel, select Properties
  - Notice that there is an entire section about CSS



- In the Stylesheets, click the "+" sign



- Select FirstCSS.css and select Open.
- Verify that the css is now part of the available Stylesheets for the project.



- Edit FirstCSS.css file.
  - Back in NetBeans, edit FirstCSS,css file. If the file is not open, just double click on it.
  - Remove everything from this file, we want to start from a clean one.
  - .root section: provides the overall default font and colors used by the rest of the sections.

```
.root {
     master-color: black;
     -fx-background-color: derive(master-color, 40%);
     -fx-control-inner-background: derive(master-color, 60%);
 }
```

In our sample we define a master-color to be black. Then we define –fx-background-color in terms of the master-color using the function derive. JavaFX supports some color computation functions. These compute new colors from input colors at the time the color style is applied. This enables a color theme to be specified using a single base color and to have variant colors computed from that base color. There are two color functions: derive() and ladder():

- `derive( <color> , <number>% ).` The derive function takes a color and computes a brighter or darker version of that color. The second parameter is the brightness offset, ranging from -100% to 100%. Positive percentages indicate brighter colors and negative percentages indicate darker colors. A value of -100% means completely black, 0% means no change in brightness, and 100% means completely white.
- ladder(<color> , <color-stop> [, <color-stop>]+). The ladder function interpolates between colors. The effect is as if a gradient is created using the stops provided, and then the brightness of the provided <color> is used to index a color value within that gradient. At 0% brightness, the color at the 0.0 end of the gradient is used; at 100% brightness, the color at the 1.0 end of the gradient is used; and at 50% brightness, the color at 0.5, the midway point of the gradient, is used. Note that no gradient is actually rendered. This is merely an interpolation function that results in a single color.

Another great thing is the ability to use gradients instead of just flat colors. We have linear-gradient and radial-gradient:

- `linear-gradient( [ [from <point> to <point>] | [ to <side-or-corner>], ]? [ [ repeat | reflect ], ]? <color-stop>[, <color-stop>]+)`

  where `<side-or-corner> = [left | right] || [top | bottom]`

  Linear gradient creates a gradient going though all the stop colors along the line between the "from" `<point>` and the "to" `<point>`. If the points are percentages, then they are relative to the size of the area being filled. Percentage and length sizes can not be mixed in a single gradient function.

  For example: `linear-gradient(to bottom right, red, black)` will create a gradient from top left to bottom right of the filled area with red at the top left corner and black at the bottom right.

- `radial-gradient([ focus-angle <angle>, ]? [ focus-distance <percentage>, ]? [ center <point>, ]? radius [ <length> | <percentage> ] [ [ repeat | reflect ], ]? <color-stop>[, <color-stop>]+)`

Lets now define some styles for the UI controls, in our case the AnchorPane and the Button. In general, each JavaFX control will have a section that defines the following:

```
.control-name {
  -fx-skin: "com.sun.javafx.scene.control.skin.ControlNameSkin";
  -fx-background-color: a, b, c, d
  -fx-background-insets: e, f, g, h
  -fx-background-radius: i, j, k, l
  -fx-padding: m
  -fx-text-fill: n
}
```

where:

-fx-background-color, -fx-background-insets, and -fx-background-radius are parallel arrays that specify background colors for the control.

-fx-background represents a sequence of colors for regions that will be drawn, one on top of the other.

-fx-background-insets is a comma separated list of insets that represent the top right bottom left insets from the edge of the control for each color specified in the -fx-background-color list. A single size for an inset means the same inset will be used for the top right bottom left values. A negative inset will draw outside the bounds of the control.

-fx-background-radius is a comma separated list of values that represent the radii of the top right, bottom right, bottom left, and top left corners of the rectangle associated with the rectangle from the -fx-background-color list. As with insets, a single size for a radius means the same radius will be used for all corners.

Typically, the following values will be used:

a/e/i = -fx-shadow-highlight-color, 0 0 -1 0, 5

Draws a background highlight dropped 1 pixel down with corners with a 5 pixel radius.

b/f/j = -fx-outer-border, 0, 5
: Draws an outer border the size of the control (insets = 0) and with corners with a 5 pixel radius.

c/g/k = -fx-inner-border, 1, 4
: Draws an inner border inset 1 pixel from the control edge and with corners with a smaller radius (radius = 4).

d/h/l = -fx-body-color, 2, 3
: Draws the body last, inset 2 pixels from the control edge and with corners with an even smaller radius (radius = 3).

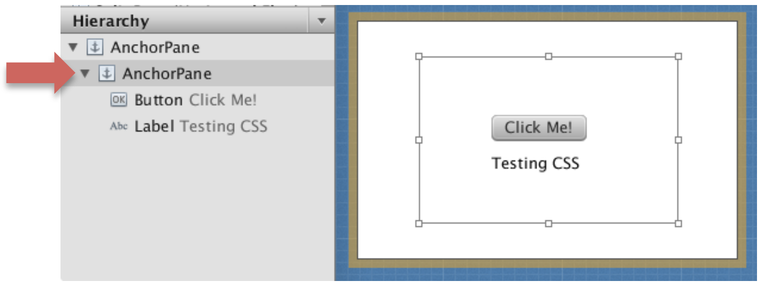
m = Padding from the edge of the control to the outer edge of the skin content.

n = If specified, the color chosen for -fx-text-fill should match the innermost color from -fx-background-colors (e.g., 'd'). See the -fx-text-fill entry in .scene for more information.

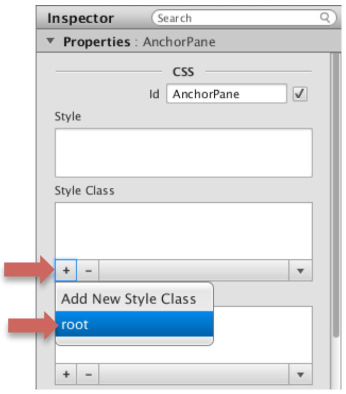
The control will also typically define pseudoclass sections for when it is focused, when the mouse is hovering over it ("hover") and when the mouse button is being held down on it (e.g., "armed").

Now that we have a better understanding of some JavaFX CSS basics, lets start defining the controls' styles.

- In Scene Builder add a new AnchorPane
- Resize the new AnchorPane to be smaller than the container
- Move the Button and the Label, so they are now inside the new AnchorPane. It's easier to move them in the Hierarchy pane
- Set the Label's text to "Testing CSS"

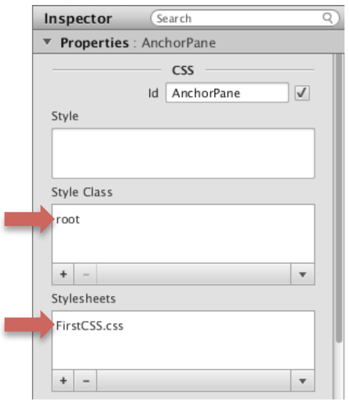


- Select the main container AnchorPane
- In the Inspector, go to the CSS section in the Properties panel
- Click on the "+" sign in the Style Class.
- From the pop up menu, select root. Now, all the definitions made in .root will be available to all components



Id AnchorPane

root



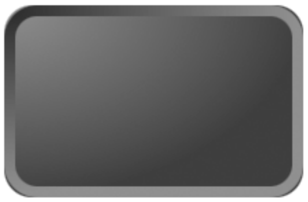
Id AnchorPane

Style Class: root

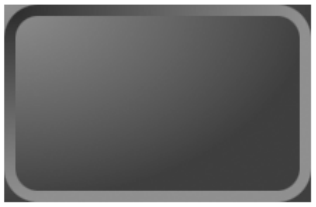
Stylesheets: FirstCSS.css

- Now, lets go back to NetBeans
  - Lets create a new style

```
1. .my-gradientpane {
2.     -fx-background-radius: 20;
3.     -fx-border-radius: 20;
4.     -fx-border-width: 8;
5.     -fx-border-color: radial-gradient(radius 100%,
                                    derive(master-color,20%),
                                    derive(master-color,80%));
6.     -fx-background-color: radial-gradient(radius 100%,
                                         derive(master-color,80%),
                                         derive(master-color,30%));
7. }
```

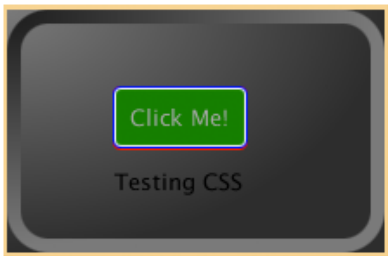


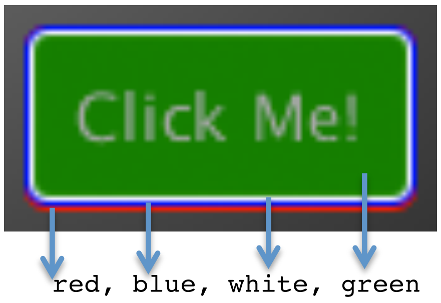
- What I'm trying to create is something a bit exaggerated, so you can see exactly what we are doing. In the second part of this exercise, you will create a nice looking interface.
  - We define our style in line 1
  - Then in line 2 and 3 we set the rounded corners. Notice that we set both, the radius for the background and for the border. This is important as if you don't set the radius for the background, you will still see a square container. Something like the picture bellow:



  - In line 4, set the border width, a bit big, but it will allow us to see the radial gradient.
  - And then, the border and background color, using the radial gradients as we explained before.
  - It's worth to notice, that we generate new colors from the `master-color` define in .root style. So we create two light shades of black, one lighter than the other.
- Now lets create a style for the button. We can simply override the default one by including the .button new definitions in our css

```
.button {
    -fx-background-color: red, blue, white, green;
    -fx-text-fill: derive(master-color,110%);
}
```



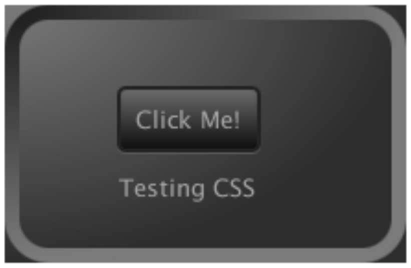


By putting different colors in the background color you can notice the position of the regions: first a shadow (red), then an outer border (blue), then inner border(white) and finally the body background(green).

Or if you preffer, you can use the master-color to generate a new tone:

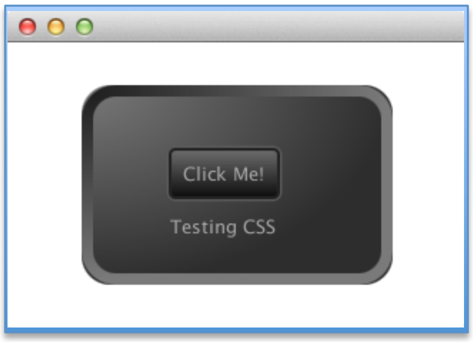
```
.button {
  -fx-background-color: derive(master-color,60%),
                        derive(master-color,20%),
                        derive(master-color,25%),
                        linear-gradient(to bottom,
                                        derive(master-color,70%),
                                        derive(master-color,10%));
  -fx-text-fill: derive(master-color,110%);
```

```
      -fx-padding: 10;
    }
```

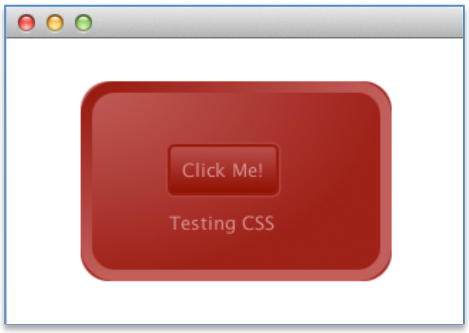


Your FirstCSS.css file including a label style looks like this:

```
.root {
   master-color: black;
   -fx-background: derive(master-color, 40%);
   -fx-control-inner-background: derive(master-color, 60%);
}

.my-gradientpane {
   -fx-border-radius: 20;
   -fx-border-width: 8;
   -fx-border-color: radial-gradient(radius 100%,
                                  derive(master-color,20%),
                                  derive(master-color,80%));
   -fx-background-color: radial-gradient(radius 100%,
                                  derive(master-color,80%),
                                  derive(master-color,30%));
}

.button {
   -fx-background-color: derive(master-color,60%),
                         derive(master-color,20%),
                         derive(master-color,25%),
                         linear-gradient(to bottom,
                                   derive(master-color,70%),
                                   derive(master-color,10%));
   -fx-text-fill: derive(master-color,110%);
   -fx-padding: 10;
}

.label{
    -fx-text-fill: derive(master-color,110%);
}
```

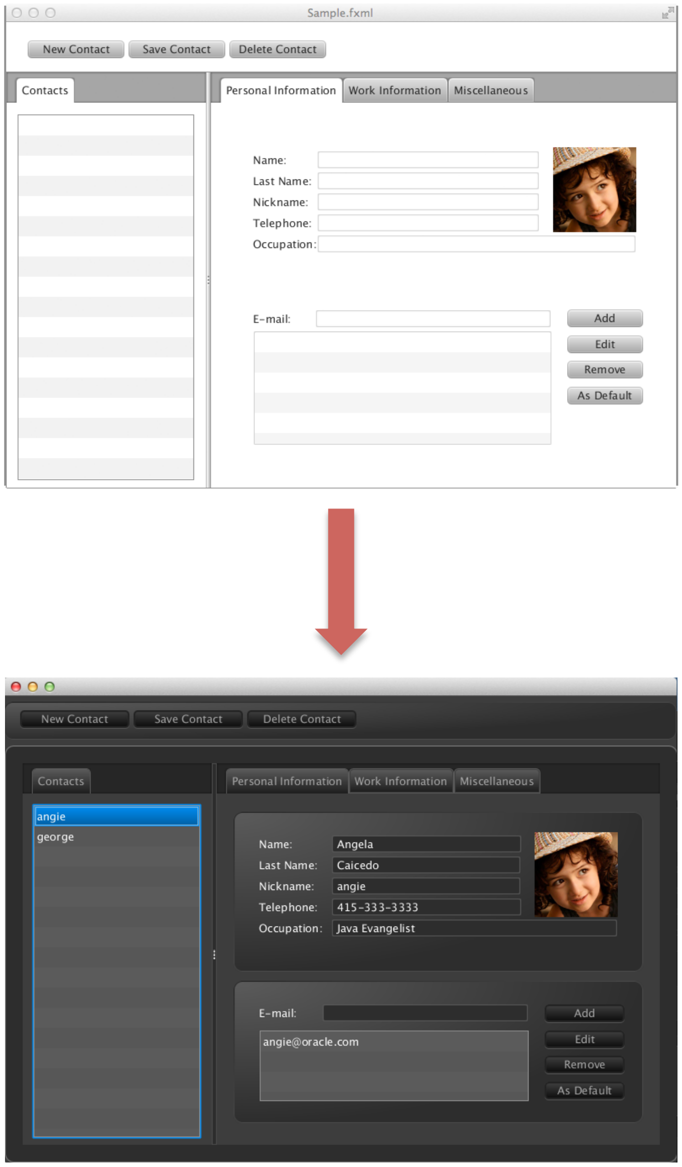


Now, because all the styles defined in our css are based on the master-color defined in the root style, just by changing the color you can have a total new look and feel. For example, replace "`master-color: black;`" with "`master-color: darkred;`" and see the result. Try different colors, and have fun!



2.3 Now lets create the complete css file.

Lets create the necessary CSS file to transform the application from the first exercise into the styled application shown bellow.



Name:
Last Name:
Nickname:
Telephone:
Occupation:
E-mail:

angie
george

Name: Angela
Last Name: Caicedo
Nickname: angie
Telephone: 415-333-3333
Occupation: Java Evangelist
E-mail:
angie@oracle.com

This is some hint about the styles defined in the solution:

- .root
- .split-pane
- .split-pane *.split-pane-divider
- .tab

- .tab-content-area
- .tab .tab-label
- .tab-pane *.tab-header-background
- .text-field
- .label
- .button
- .button:focused
- .my-gradientpane   //to define the rounded pane

Remember, there isn't a unique solution for this exercise. For your help a copy of the Caspian.css file was included inside the exercises/exercise2 directory. You can open this file in NetBeans and get ideas about how to personalize some of the components.

Our solution for this exercise can be found at solutions/exercise2 directory.

3. JavaFX, JavaScript, WebView and HTML5

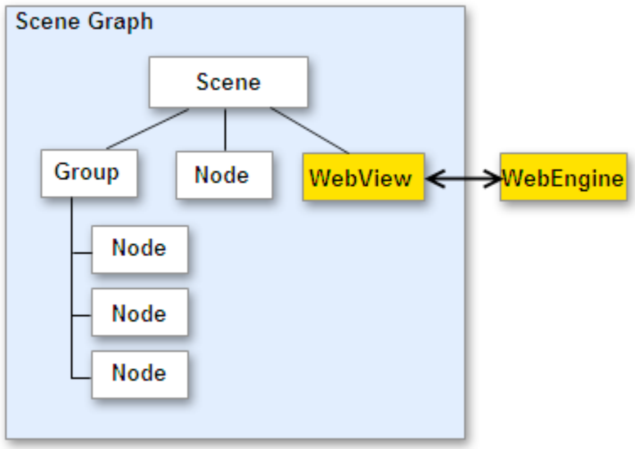
One of the coolest features in JavaFX is the possibility to include HTML content in your JavaFX applications. JavaFX includes an embedded browser, a user interface component that provides a web viewer and full browsing functionality through its API.

The embedded browser component is based on WebKit, an open source web browser engine. It supports Cascading Style Sheets (CSS), JavaScript, Document Object Model (DOM), and HTML5.

The embedded browser enables you to perform the following tasks in your JavaFX applications:

- Render HTML content from local and remote URLs
- Obtain Web history
- Execute JavaScript commands
- Perform upcalls from JavaScript to JavaFX
- Manage web pop-up windows
- Apply effects to the embedded browser

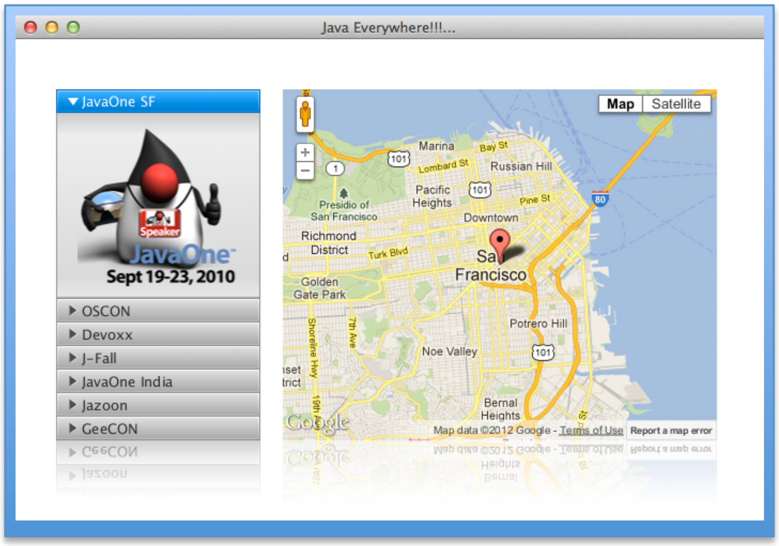
The embedded browser inherits all fields and methods from the Node class, and therefore, it has all its features.



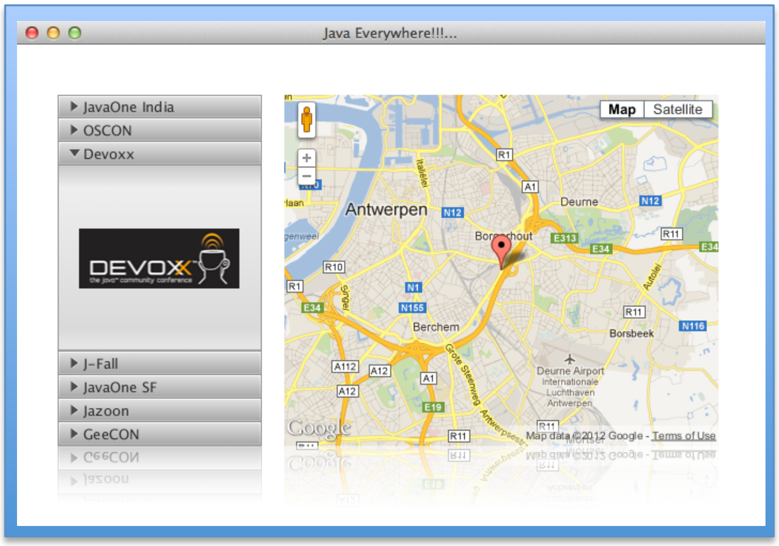
Architecture of the Embedded Browser

3.1 Lets see what we are going to build

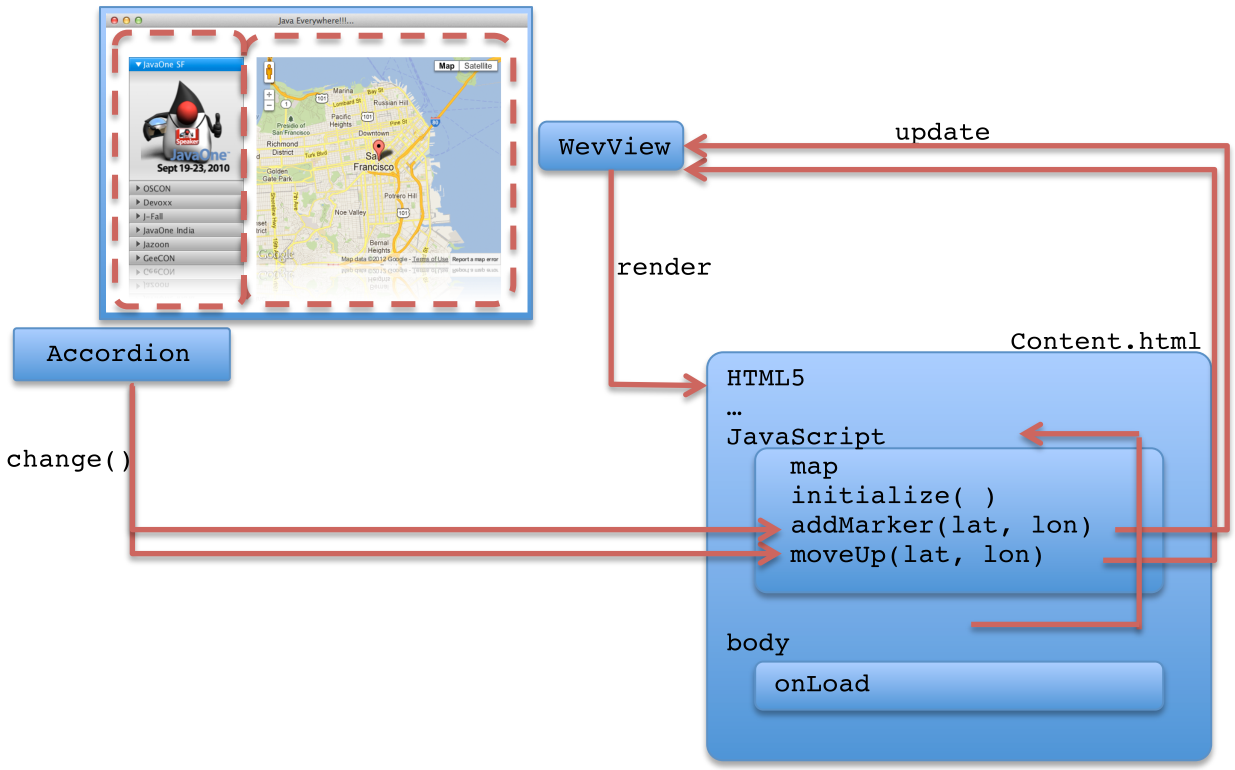
- From NetBeans open Project JavaFX-HTML5 located at `<LAB_ROOT>/solutions/exercise3/`
- Run it.



The application shows the location of some of the most important Java conferences around the world. As you click on the name of the conference, the map next to it gets updated with the location of the conference.



3.2. How does it work?

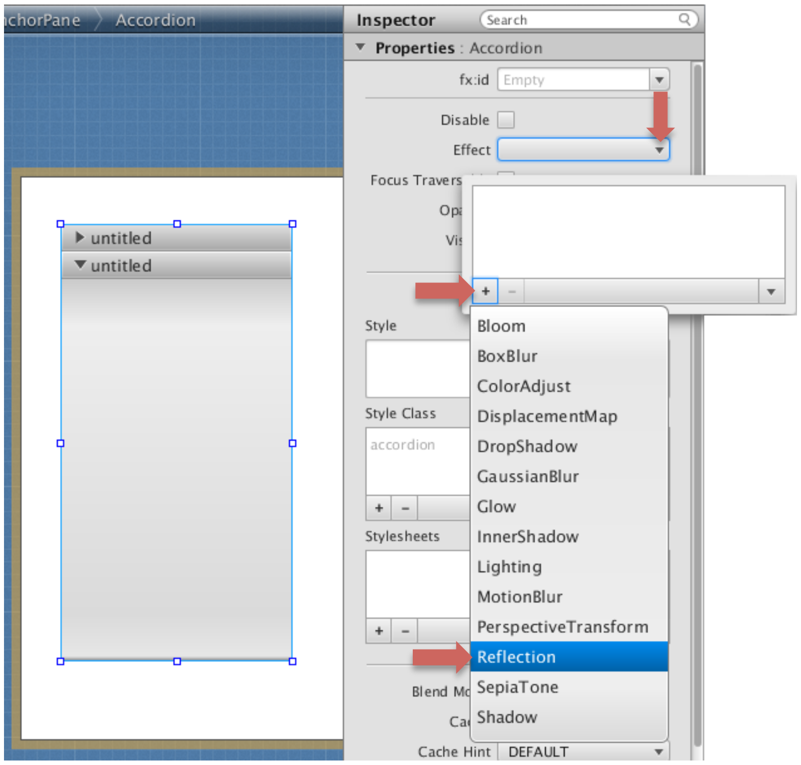


3.3 User Interface

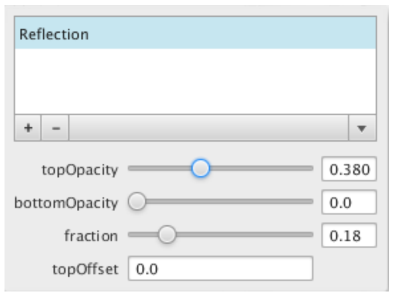
- Create a new JavaFX project, and name it JavaFX_HTML5.
    - Select New Project
    - From the "New Project" window select:
        - From Categories: JavaFX
        - From Projects: JavaFX FXML Application
    - You can enter any name you want and select any directory for the project location. For the purpose of this lab, we are going to use JavaFX_HTML5 for the project name, and we are going to select exercises/exercise3 as the project location.
    - Make sure the "Create Application Class" is selected.
    - Then click "Finish".

- Design the basic UI layout using Scene Builder:
    - Double click on Sample.fxml

- Once you are inside Scene Builder get rid of the `Button` and the `Label`.
- Resize AnchorPane to 700x450
- Insert and Accordion:
  - Resize, and anchor it to the top and bottom.
  - Set the fx:id to “conferenceAccordion”
- Insert a WebView
  - Resize and anchor it to top, left, bottom and right.
  - Set the fx:id to “webView”
- Add some effect
  - Select conferenceAccordion
  - Inside the properties section of the inspector panel, select the dropdown list named Effect.
  - Click on the “+” sign and select Reflection from the menu

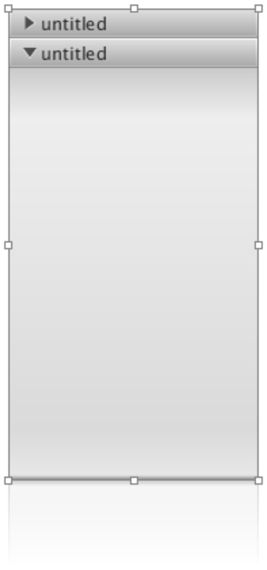


  - Set the following values for the reflection:
    - topOpacity 0.246
    - bottomOpacity 0.0
    - fraction 0.18
    - topOffset 0.0



topOpacity 0.380
bottomOpacity 0.0
fraction 0.18
topOffset 0.0

- Notice the nice reflection effect.



- Repeat the same steps for `WebView`

  - Delete the two `TitledPane` contained by default in Accordion. We will be adding this `TitledPanes` from SampleController class.
  - Save the changes and close Scene Builder

- Program the SampleController

  - Remove `label` and `handleButtonAction` method
  - Inject the accordion and the webview in the sample controller. For more details on how to do this, go back to exercise 1.

```
@FXML
private Accordion conferenceAccordion;
@FXML
private WebView webView;
```

- Create a public inner class called `ConferencePane`
  - This class should inherit from TitlePane
  - Contain two private double values: `lat` and `lon`, to store the latitude and longitude of the event
  - Have a constructor containing as parameters:
    - the title for the pane,
    - the node,
    - the latitude and
    - longitude for the conference

```
public class ConferencePane extends TitledPane {
    private final double lat;
    private final double lon;

    private ConferencePane(String label, Node node,
                           double lat, double lon) {
        super(label, node);
        this.lat = lat;
        this.lon = lon;
    }
 }
```

- Create a new method named createConference:
  - Should receive 4 parameters:
    - Name of the conference (String)
    - Latitude for the conference (double)
    - Longitude for the conference (double)
    - Image URL for the conference (String)
  - Should create and return a ConferencePane object, using the information provided as parameters.

```
private ConferencePane createConference(String name,
                                        final double lat,
                                        final double lon,
                                        String imageUrl) {
   return new ConferencePane(name,
                             new ImageView(new Image(imageUrl)),
                             lat, lon);
}
```

- Program the initialize method
  - Create and load the TitlePane for the Accordion
    - Create a TitlePane named sf, with the following information:
      - Name "JavaOne SF",
      - Latitude: 37.775057,
      - Longitude: -122.416534,
      - URL: "http://steveonjava.com/wp-content/uploads
        /2010/07/JavaOne-2010-Speaker.png");
    - Set this TitlePane as the extended one, so when the app is loaded it will show JavaOne details
    - Add sf to the `conferenceAccordion` created in Scene Builder

```
final TitledPane sf = createConference("JavaOne SF",
                 37.775057, -122.416534,
                "http://steveonjava.com/wp-content/uploads/
                 2010/07/JavaOne-2010-Speaker.png");
conferenceAccordion.getPanes().add(sf);
sf.setExpanded(true);
conferenceAccordion.setExpandedPane(sf);
```

  - Add the rest of the TitlePanes to conferenceAccordion

```
conferenceAccordion.getPanes().add(createConference("OSCON",
                 45.515008, -122.693253,
                 "http://steveonjava.com/wp-content/
                 uploads/2011/05/oscon.png"));

conferenceAccordion.getPanes().add(createConference(
                 "Devoxx", 51.206883, 4.44,
                 "http://steveonjava.com/wp-content/
                 uploads/2010/07/LogoDevoxxNeg150.png"));

conferenceAccordion.getPanes().add(createConference(
                 "J-Fall", 52.219913, 5.474253,
                 "http://steveonjava.com/wp-content/
                 uploads/2011/11/jfall3.png"));

conferenceAccordion.getPanes().add(createConference(
                 "JavaOne India", 17.385371, 78.484268,
                 "http://steveonjava.com/wp-content/uploads/
                 2011/03/javaone-india.png"));

conferenceAccordion.getPanes().add(createConference(
                 "Jazoon", 47.382079, 8.528137,
                 "http://steveonjava.com/wp-content/uploads/
                 2010/04/jazoon.png"));

conferenceAccordion.getPanes().add(createConference(
                 "GeeCON", 50.064633, 19.949799,
                 "http://steveonjava.com/wp-content/uploads/
                 2011/03/geecon.png"));
```

- SampleController should look like this:

```
public class SampleController implements Initializable {

    @FXML
    private Accordion conferenceAccordion;
    @FXML
    private WebView webView;


    @Override
    public void initialize(URL url, ResourceBundle rb) {
      final TitledPane sf = createConference("JavaOne SF",
                  37.775057, -122.416534,
                 "http://steveonjava.com/wp-content/uploads/
                  2010/07/JavaOne-2010-Speaker.png");
      conferenceAccordion.getPanes().add(sf);
      sf.setExpanded(true);
      conferenceAccordion.setExpandedPane(sf);

      conferenceAccordion.getPanes().add(
                  createConference("OSCON",
                  45.515008, -122.693253,
                  "http://steveonjava.com/wp-content/
                  uploads/2011/05/oscon.png"));

      conferenceAccordion.getPanes().add(
                   createConference(
                  "Devoxx", 51.206883, 4.44,
                  "http://steveonjava.com/wp-content/
                  uploads/2010/07/LogoDevoxxNeg150.png"));

      conferenceAccordion.getPanes().add(
                   createConference(
                  "J-Fall", 52.219913, 5.474253,
                  "http://steveonjava.com/wp-content/
                  uploads/2011/11/jfall3.png"));

      conferenceAccordion.getPanes().add(
                  createConference(
                  "JavaOne India", 17.385371, 78.484268,
                  "http://steveonjava.com/wp-content/uploads/
                  2011/03/javaone-india.png"));
```

```
        conferenceAccordion.getPanes().add(
                    createConference(
                   "Jazoon", 47.382079, 8.528137,
                   "http://steveonjava.com/wp-content/uploads/
                   2010/04/jazoon.png"));

        conferenceAccordion.getPanes().add(
                    createConference(
                   "GeeCON", 50.064633, 19.949799,
                   "http://steveonjava.com/wp-content/uploads/
                   2011/03/geecon.png"));
      }

      private ConferencePane createConference(String name,
                                          final double lat,
                                          final double lon,
                                          String imageUrl) {
            return new ConferencePane(name,
                         new ImageView(new Image(imageUrl)),
                         lat, lon);
       }

      public class ConferencePane extends TitledPane {
        private final double lat;
        private final double lon;

        private ConferencePane(String label, Node node,
                               double lat, double lon) {
            super(label, node);
            this.lat = lat;
            this.lon = lon;
        }
      }
  }
```

- Copy content.html file
  - Right click on the javafx_html5 project and select New -> Other -> HTML File
  - Remove anything on it
  - Add the following code

```
<!DOCTYPE html>
<html>
   <head>
     <meta name="viewport" content="initial-scale=1.0,
                           user-scalable=no" />
     <style type="text/css">
            html { height: 100% }
```

```
              body { height: 100%; margin: 0px; padding: 0px }
              #map_canvas { height: 100% }
       </style>
       <script type="text/javascript"
                     src="http://maps.google.com/maps/
                          api/js?sensor=true">
       </script>
       <script type="text/javascript">
          var map ;
          function initialize() {
             var latlng = new google.maps.LatLng(37.775057,
                                               -122.416534);
             var myOptions = {
                      zoom: 12,
                      center: latlng,
                      mapTypeId: google.maps.MapTypeId.ROADMAP
             };
             map = new google.maps.Map(
                        document.getElementById("map_canvas"),
                        myOptions);
             addMarker(37.775057, -122.416534);
         }

         function addMarker(lat, lng) {
             var newmarker = new google.maps.Marker({
                   position: new google.maps.LatLng(lat, lng),
                   map: map
             });
         }

         function moveMap(lat, lng) {
             var latlng = new google.maps.LatLng(lat, lng);
             map.setCenter(latlng);
         }
      </script>
      </head>
      <body onload="initialize()">
          <div id="map_canvas" style="width:100%;
                                        height:100%"></div>
      </body>
  </html>
```

- Notice that there are 3 methods:
    - initialize, create the map from google maps, and add a marker
    - addMarker: add a marker on the lat,lon coordinate of the event
    - moveMap: position the map to a particular location

- Now, we just need to connect a couple of things:
  - First we need to load the content.html file using the webView

    - Create an engine variable to hold the webEngine associated with the webView
    - Load content.html in the initialize method

```
...
WebEngine engine;
...

engine = webView.getEngine();
webView.getEngine().load("file:///Users/angelacaicedo/
JavaFX-HOLs/exercises/exercise3/JavaFX-
HTML5/src/javafx/html5/content.html");
```

- Create a new method `navigateTo` inside the inner class `ConferencePane`.
    - This method will execute some scripts using the web engine. We want to call `moveMap` and `addMarker` from our JavaScript defined inside the content.html file.

```
public void navigateTo() {
  engine.executeScript("moveMap("+lat+ ", " + lon + ");");
  engine.executeScript("addMarker("+lat+ ", " + lon + ");");
}
```

- Finally the last step will be to call the `navigateTo` method, every time the accordion expands, so we can update the details of the selected conference.

```
conferenceAccordion.expandedPaneProperty().addListener(
       new ChangeListener<TitledPane>() {
            public void changed(
                  ObservableValue<? extends  TitledPane> ov,
                  TitledPane t, TitledPane t1) {
                if (t1 != null) {
                    ((ConferencePane)t1).navigateTo();
                }
            }
        });
```

- The final SampleController.java should look like this:

```
public class SampleController implements Initializable {

    WebEngine engine;

    @FXML
    private Accordion conferenceAccordion;
    @FXML
    private WebView webView;


    @Override
    public void initialize(URL url, ResourceBundle rb) {
      engine = webView.getEngine();
      webView.getEngine().load("file:///Users/angelacaicedo/
                   JavaFX-HOLs/exercises/exercise3/JavaFX-
                   HTML5/src/javafx/html5/content.html");


      final TitledPane sf = createConference("JavaOne SF",
                 37.775057, -122.416534,
                "http://steveonjava.com/wp-content/uploads/
                 2010/07/JavaOne-2010-Speaker.png");
      conferenceAccordion.getPanes().add(sf);
      sf.setExpanded(true);
      conferenceAccordion.setExpandedPane(sf);

      conferenceAccordion.getPanes().add(
                 createConference("OSCON",
                 45.515008, -122.693253,
                 "http://steveonjava.com/wp-content/
                 uploads/2011/05/oscon.png"));

      conferenceAccordion.getPanes().add(
                  createConference(
                 "Devoxx", 51.206883, 4.44,
                 "http://steveonjava.com/wp-content/
                 uploads/2010/07/LogoDevoxxNeg150.png"));

      conferenceAccordion.getPanes().add(
                  createConference(
                 "J-Fall", 52.219913, 5.474253,
                 "http://steveonjava.com/wp-content/
                 uploads/2011/11/jfall3.png"));

      conferenceAccordion.getPanes().add(
                 createConference(
                 "JavaOne India", 17.385371, 78.484268,
                 "http://steveonjava.com/wp-content/uploads/
                 2011/03/javaone-india.png"));

      conferenceAccordion.getPanes().add(
                  createConference(
                 "Jazoon", 47.382079, 8.528137,
                 "http://steveonjava.com/wp-content/uploads/
```

```
                   2010/04/jazoon.png"));

        conferenceAccordion.getPanes().add(
                    createConference(
                   "GeeCON", 50.064633, 19.949799,
                   "http://steveonjava.com/wp-content/uploads/
                   2011/03/geecon.png"));
       conferenceAccordion.expandedPaneProperty().addListener(
         new ChangeListener<TitledPane>() {
              public void changed(
                    ObservableValue<? extends  TitledPane> ov,
                    TitledPane t, TitledPane t1) {
                  if (t1 != null) {
                      ((ConferencePane)t1).navigateTo();
                  }
              }
          });

     }

      private ConferencePane createConference(String name,
                                             final double lat,
                                             final double lon,
                                             String imageUrl) {
            return new ConferencePane(name,
                           new ImageView(new Image(imageUrl)),
                           lat, lon);
       }

      public class ConferencePane extends TitledPane {
        private final double lat;
        private final double lon;

        private ConferencePane(String label, Node node,
                         double lat, double lon) {
            super(label, node);
            this.lat = lat;
            this.lon = lon;
        }

        public void navigateTo() {
         engine.executeScript("moveMap("+lat+", "+lon+");");
         engine.executeScript("addMarker("+lat+", "+lon+");");
        }
      }
  }
```

- Run and test the application. Make sure everything works properly

4. Canvas API

Canvas is an image that can be drawn on using a set of graphics commands provided by a GraphicsContext. A Canvas is constructed with a width and height that specifies the size of the image into which the canvas drawing commands are rendered. All drawing operations are clipped to the bounds of that image.

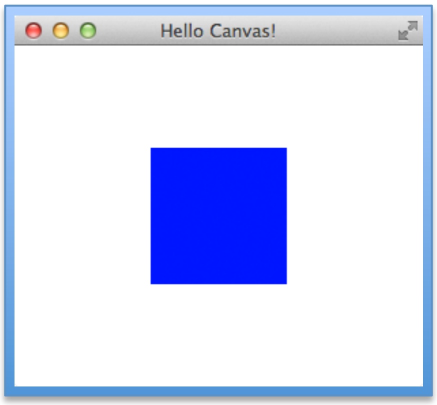
The Canvas class is used to issue draw calls using a buffer. Each call pushes the necessary parameters onto the buffer where it is executed on the image of the Canvas node.

A Canvas only contains one GraphicsContext, and only one buffer. If it is not attached to any scene, then it can be modified by any thread, as long as it is only used from one thread at a time. Once a Canvas node is attached to a scene, it must be modified on the JavaFX Application Thread.

Calling any method on the GraphicsContext is considered modifying its corresponding Canvas and is subject to the same threading rules.

A GraphicsContext also manages a stack of state objects that can be saved or restored at anytime.

```
public class CanvasFirst extends Application {

  @Override
  public void start(Stage primaryStage) {
    final Canvas canvas = new Canvas(250, 250);
    GraphicsContext gc = canvas.getGraphicsContext2D();

    gc.setFill(Color.BLUE);
    gc.fillRect(75, 75, 100, 100);

    StackPane root = new StackPane();
    root.getChildren().add(canvas);

     Scene scene = new Scene(root, 300, 250);

     primaryStage.setTitle("Hello Canvas!");
     primaryStage.setScene(scene);
     primaryStage.show();
    }

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

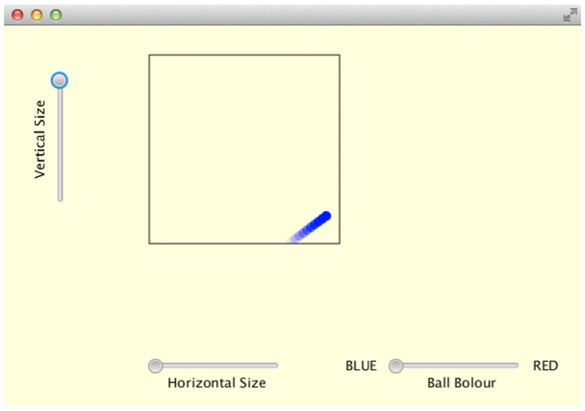


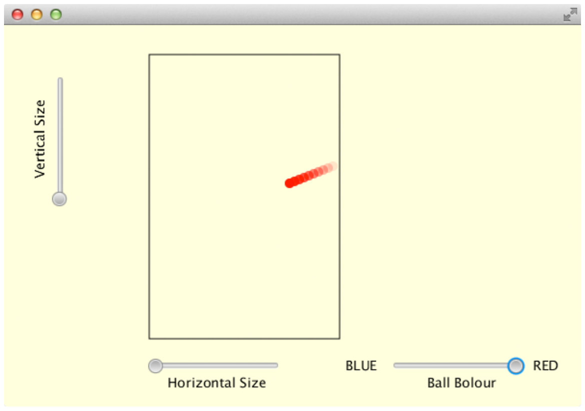
In this sample, we create a drawing area (Canvas) of 250 x 250 (Line 4). Then, GraphicContext gc draws a blue rectangle on it (lines 6-7). Finally the canvas component is added to the scene graph and it gets displayed.

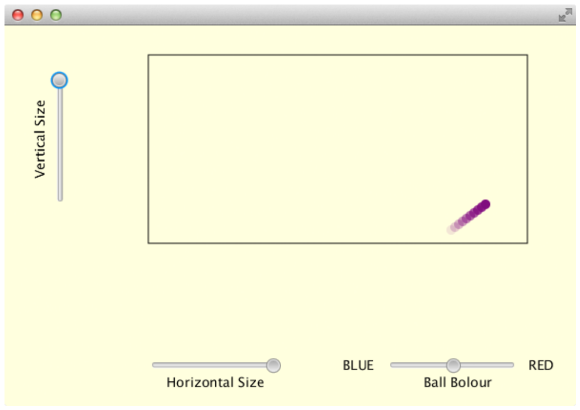
4.1 What are we going to build?

Lets run the application we are going to build in this exercise:

- From NetBeans, open Project CanvasDemo located at `<LAB_ROOT>/solutions/exercise4/`
- Run it
- Use the Sliders to change the size of the canvas, and the color of the ball.







4.2 Lets see the code

- Create a canvas with dimensions 200x200.
- Position the canvas at coordinate 150x30
- Add the canvas to the scenegraph
- Get the Graphic context for future reference.

```
private void createCanvas() {
     /* Create the canvas and get the GraphicsContext ready for
        all updates */
     canvas = new Canvas(200, 200);
     canvas.setTranslateX(150);
     canvas.setTranslateY(30);
     getChildren().add(canvas);

     gc = canvas.getGraphicsContext2D();
 }
```

- Create five Labels with the following attributes:

Text	Horizontal Size
Position	(170, 380)
Font	14

Text	Vertical Size
Position	(-5, 125)
Font	14
Rotation	270

Text	BLUE
Position	(355, 362)
Font	14

Text	RED
Position	(550, 362)
Font	14

Text	Ball Colour
Position	(440, 380)
Font	14

- Add the Labels to the scene graph

```
private void createLabels(){
      Font f = new Font(14);

      /* Label for canvas width slider */
      getChildren().add(TextBuilder.
              create().
              text("Horizontal Size").
              translateX(170).
              translateY(380).
              font(f).
              build());

      /* Label for canvas height slider */
      getChildren().add(TextBuilder.
              create().
              text("Vertical Size").
              translateX(-5).
              translateY(125).
              font(f).
              rotate(270).
              build());

      /* Label for colour control slider */
      getChildren().add(TextBuilder.
              create().
              text("BLUE").
              translateX(355).
              translateY(362).
              font(f).
              build());
      getChildren().add(TextBuilder.
              create().
              text("RED").
              translateX(550).
              translateY(362).
              font(f).
              build());
      getChildren().add(TextBuilder.
              create().
              text("Ball Bolour").
              translateX(440).
              translateY(380).
              font(f).
              build());
  }
```

- Create 3 Sliders with the following attributes:

Variable name	xSlider
min	200
max	400
Orientation	Orientation.Horizontal
Position	(150,350)

Variable name	ySlider
min	200
max	300
Orientation	Orientation.Vertical
Position	(50,50)
Rotation	180

Variable name	colourSlider
min	0
max	100
Orientation	Orientation.Horizontal
Position	(400,350)

- Add the Sliders to the scene graph

```
private void createSliders() {
        //Create Sliders
        /* Slider to control canvas width */
        xSlider = SliderBuilder.create().
                min(200).
                max(400).
                orientation(Orientation.HORIZONTAL).
                translateX(150).
                translateY(350).
                build();

        /* Slider to control canvas height */
        ySlider = SliderBuilder.create().
                min(200).
                max(300).
                orientation(Orientation.VERTICAL).
                translateX(50).
                translateY(50).
                build();
        ySlider.setRotate(180);

        /* Slider to control the colour of the ball in the canvas */
        colourSlider = SliderBuilder.create().
                min(0).
                max(100).
                orientation(Orientation.HORIZONTAL).
                translateX(400).
                translateY(350).
                build();

        //Add Sliders
        getChildren().add(xSlider);
        getChildren().add(ySlider);
        getChildren().add(colourSlider);
    }
```

- Bind the canvas' height to the ySlider, and the canvas' width to the xSlider

```
    private void createCanvas() {

        ...
        //Create the binding with the sliders
        canvas.heightProperty().bind(ySlider.valueProperty());
        canvas.widthProperty().bind(xSlider.valueProperty());

        ...

    }
```

Binding the color blue and the color red to the slider is not straightforward. Rather than binding to a specific property, we are binding to a more complex expression. One solution is to do low level binding, in our case using `DoubleBinding`. We required two `DoubleBinding` implementations: one for the red color and one for the blue one. Every time the `colourSlider` changes, the `computeValue` will be invoked, and the appropriate value will be return.

```
    private void createBindings(){

        //Bind red and blue color to the colourSlider
        redDB = new DoubleBinding() {
            {
                super.bind(colourSlider.valueProperty());
            }
            @Override
            protected double computeValue() {
                return colourSlider.getValue()/100.0;
            }
        };

        blueDB = new DoubleBinding() {
            {
                super.bind(colourSlider.valueProperty());
            }
            @Override
            protected double computeValue() {
                return (100.0 - colourSlider.getValue())/100.0;
            }
        };
    }
```

We have the `updateCanvas` method, where all the painting happens. Remember than in previous code we got the `GraphicContext gc`, here we use it to perform all the painting.

```
private void updateCanvas() {


      /* Clear the canvas and draw a border */
      width = canvas.getWidth();
      height = canvas.getHeight();
      gc.clearRect(0, 0, width, height);
      gc.setStroke(Color.BLACK);
      gc.strokeRect(1, 1, width - 2, height - 2);

      /* If the ball trail size is 10 remove the first element
         so we keep it a constant length*/
      if (trail.size() == 10) {
          trail.remove(0);
      }

      int i = 1;

      /**
       * Draw the elements of the trail of the ball
       */
      for (Point2D p : trail) {
        double opacity = i * 0.1;
        gc.setFill(new Color(redDB.get(),0,blueDB.get(),opacity));
        gc.fillOval(p.getX(), p.getY(), 10, 10);
        i++;
      }

      /**
       * Update the position of the ball
       */
      x += (5 * Math.cos(angle));
```

```
        y += (5 * Math.sin(angle));
        double degAngle = Math.toDegrees(angle);

        if (x > (width - 10)) {
            if (degAngle < 180) {
                degAngle = 180 - degAngle;
            } else {
                degAngle = 180 + dAngle;
            }
        }

        if (x < 0) {
            if (degAngle > 180) {
                degAngle = 360 - dAngle;
            } else {
                degAngle = dAngle;
            }
        }

        if (y > (height - 10)) {
            if (degAngle > 90) {
                degAngle = 360 - degAngle;
            } else {
                degAngle = 360 - dAngle;
            }
        }

        if (y < 0) {
            if (degAngle > 270) {
                degAngle = dAngle;
            } else {
                degAngle = 180 - dAngle;
            }
        }

        angle = Math.toRadians(degAngle);

        /* Draw the ball */
        gc.setFill(new Color(redDB.get(), 0, blueDB.get(), 1.0));
        gc.fillOval(x, y, 10, 10);
        trail.add(new Point2D(x, y));
    }
```

Remember to invoke the `updateCanvas` method from `createCanvas`, to refresh the screen once it is created

```
    private void createCanvas() {

        ...
        updateCanvas();
     }
```

Finally, the method `run` is used to update things correctly: using the JavaFX application thread. We use the Method Platform.runLater method, which run the specified Runnable on the JavaFX Application Thread at some unspecified time in the future. This method, which may be called from any thread, will post the Runnable to an event queue and then return immediately to the caller.

```
@Override
public void run() {
    /**
    * In order to update things correctly we must use the JavaFX
    * application thread to do the updates. This thread puts a
    * job onto that thread every 40ms.
    */
    while (true) {
       Platform.runLater(new Runnable() {
            @Override
            public void run() {
               updateCanvas();
            }
       });
       try {
           Thread.sleep(40);
       } catch (InterruptedException ie) {
            // Ignore
       }
    }
}
```

Finally the whole class is ready, have a look at CanvasTestFrame.java that puts everything together.

```
public class CanvasTestFrame extends Parent implements Runnable {

    private static final int LEFT = 1;
    private static final int RIGHT = 2;
    private static final int UP = 3;
    private static final int DOWN = 4;
    private Canvas canvas;
    private Slider xSlider;
    private Slider ySlider;
    private Slider colourSlider;

    private double angle = Math.toRadians(30);
    private int dAngle = 30;
    private GraphicsContext gc;
    private double width;
    private double height;
    private int x = 0;
    private int y = 0;
    private ArrayList<Point2D> trail = new ArrayList<>();
```

```
      private DoubleBinding redDB;
      private DoubleBinding blueDB;


      public CanvasTestFrame() {

          width = 200;
          height = 200;
          createSliders();
          createBindings();
          createLabels();
          createCanvas();

      }

      private void createCanvas() {
          canvas = new Canvas(200, 200);
          canvas.setTranslateX(150);
          canvas.setTranslateY(30);
          getChildren().add(canvas);

          canvas.heightProperty().bind(ySlider.valueProperty());
          canvas.widthProperty().bind(xSlider.valueProperty());

          gc = canvas.getGraphicsContext2D();
          updateCanvas();
      }

      private void createLabels(){
          Font f = new Font(14);

          getChildren().add(TextBuilder.
                  create().
                  text("Horizontal Size").
                  translateX(170).
                  translateY(380).
                  font(f).
                  build());


         getChildren().add(TextBuilder.
                  create().
                  text("Vertical Size").
                  translateX(-5).
                  translateY(125).
                  font(f).
                  rotate(270).
                  build());
         getChildren().add(TextBuilder.
                  create().
                  text("BLUE").
                  translateX(355).
                  translateY(362).
                  font(f).
                  build());
          getChildren().add(TextBuilder.
                  create().
                  text("RED").
                  translateX(550).
                  translateY(362).
                  font(f).
                  build());
          getChildren().add(TextBuilder.
                  create().
                  text("Ball Bolour").
```

```
                translateX(440).
                translateY(380).
                font(f).
                build());
    }


    private void createBindings(){
        redDB = new DoubleBinding() {
            {
                super.bind(colourSlider.valueProperty());
            }
            @Override
            protected double computeValue() {
                return colourSlider.getValue()/100.0;
            }
        };

        blueDB = new DoubleBinding() {
            {
                super.bind(colourSlider.valueProperty());
            }
            @Override
            protected double computeValue() {
                return (100.0 - colourSlider.getValue())/100.0;
            }
        };
    }


    private void createSliders() {

        xSlider = SliderBuilder.create().
                min(200).
                max(400).
                orientation(Orientation.HORIZONTAL).
                translateX(150).
                translateY(350).
                build();


        ySlider = SliderBuilder.create().
                min(200).
                max(300).
                orientation(Orientation.VERTICAL).
                translateX(50).
                translateY(50).
                build();
        ySlider.setRotate(180);


        colourSlider = SliderBuilder.create().
                min(0).
                max(100).
                orientation(Orientation.HORIZONTAL).
                translateX(400).
                translateY(350).
                build();


        getChildren().add(xSlider);
        getChildren().add(ySlider);
        getChildren().add(colourSlider);
    }


    private void updateCanvas() {
        width = canvas.getWidth();
        height = canvas.getHeight();
        gc.clearRect(0, 0, width, height);
        gc.setStroke(Color.BLACK);
        gc.strokeRect(1, 1, width - 2, height - 2);

        if (trail.size() == 10) {
            trail.remove(0);
        }

        int i = 1;

        for (Point2D p : trail) {
            double opacity = i * 0.1;
            gc.setFill(new Color(redDB.get(), 0, blueDB.get(), opacity));
            gc.fillOval(p.getX(), p.getY(), 10, 10);
            i++;
        }

        x += (5 * Math.cos(angle));
        y += (5 * Math.sin(angle));
        double degAngle = Math.toDegrees(angle);

        if (x > (width - 10)) {
            if (degAngle < 180) {
                degAngle = 180 - degAngle;
            } else {
                degAngle = 180 + dAngle;
            }
        }

        if (x < 0) {
            if (degAngle > 180) {
                degAngle = 360 - dAngle;
            } else {
                degAngle = dAngle;
            }
        }

        if (y > (height - 10)) {
            if (degAngle > 90) {
                degAngle = 360 - degAngle;
            } else {
                degAngle = 360 - dAngle;
            }
        }

        if (y < 0) {
            if (degAngle > 270) {
                degAngle = dAngle;
            } else {
                degAngle = 180 - dAngle;
            }
        }

        angle = Math.toRadians(degAngle);

        /* Draw the ball */
        gc.setFill(new Color(redDB.get(), 0, blueDB.get(), 1.0));
        gc.fillOval(x, y, 10, 10);
        trail.add(new Point2D(x, y));
    }


    @Override
    public void run() {

        while (true) {
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                    updateCanvas();
                }
            });
            try {
                Thread.sleep(40);
            } catch (InterruptedException ie) {
                // Ignore
            }
        }
    }
}
```

Now lets test our class:

- Create a new JavaFX project, and name it CanvasDemo.
  - Select New Project
  - From the "New Project" window select:
    - From Categories: JavaFX
    - From Projects: JavaFX Application
  - You can enter any name you want and select any directory for the project location. For the purpose of this lab, we are going to use CanvasDemo for the project name, and we are going to select exercises/exercise4 as the project location.
  - Make sure the "Create Application Class" is selected.
  - Then click "Finish".
- Add the CanvasTestFrame.java file to the project.
  - Right click on canvasdemo package and select New -> Java Class.
  - Use CanvasTestFrame for the class name
  - Replace the content of this file with the code we just study previously.
- Update CanvasDemo.java
  - Double click on CanvasDemo and replace the `start` method with the following code, basically we are instantiating the CanvasTestFrame we just study, and starting the thread that update the canvas painting.

```
@Override
public void start(Stage stage) throws Exception {
  CanvasTestFrame tf = new CanvasTestFrame();
  Scene scene = new Scene(tf, 600, 400);
  stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
    @Override
    public void handle(WindowEvent t) {
      System.exit(0);
    }
  });
  stage.setScene(scene);
  stage.show();
  new Thread(tf).start();
}
```

- Run your project and test your Canvas.

5. The End.

This is the end of the HOL, I hope you enjoyed it.

THANK YOU