CSCI 2120:
Software Design & Development II

UNIT4: UI management

GUI framework

JavaFX Controls: Button

Overview

Introduction

- A JavaFX `Button` control enables a JavaFX application to have some action executed when the application user clicks the button.
- The JavaFX `Button` control is represented by the class `javafx.scene.control.Button` .
- A JavaFX `Button` can have a text and an icon on it which indicate to the user what clicking the button will do.

Creating a Button

You create a button control by creating an instance of the `Button` class.

Here is a JavaFX `Button` instantiation example:

```
Button button = new Button("My Label");
```

The text to be displayed on the button is passed as parameters to the `Button` constructor.

Adding a Button to the Scene Graph

For a JavaFX `Button` to be visible the button object must be added to the scene graph. This means adding it to a `Scene` object, or as child of a layout which is attached to a `Scene` object.

Here is an example that attaches a JavaFX `Button` to the scene graph:

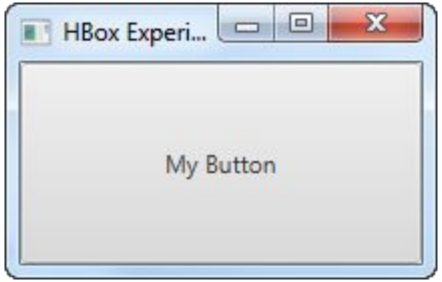
Adding a Button to the Scene Graph

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.stage.Stage;

public class ButtonExperiments extends Application  {
   @Override
   public void start(Stage primaryStage) throws Exception {
       primaryStage.setTitle("HBox Experiment 1");

       Button button = new Button("My Button");

       Scene scene = new Scene(button, 200, 100);
       primaryStage.setScene(scene);
       primaryStage.show();

   }

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

Adding a Button to the Scene Graph

Notice that the `Button` is added directly to the `Scene` object. Normally you would nest the `Button` inside a layout component of some kind. We have left that out here to keep the example simple. See the lectures about layout components to see how they work.

The result of running the above JavaFX `Button` example is an application that looks like this:



Notice that the button takes up all the space available in the window. That is why it is hard to see the edges of the button. When a JavaFX button is added to a layout component you can more easily see the edges of the button.

Button Text

There are two ways to set the text of a JavaFX button.

1. The first way is to pass the text to the `Button` constructor. We have already seen this in earlier examples.

2. The second way to set the button text is by calling the `setText()` method on the `Button` instance. This can be done after the `Button` instance is created. Thus it can be used to change the text of a `Button` that is already visible.

   Here is an example how how calling `setText()` on a JavaFX `Button` looks:

```
button.setText("Click me if you dare!");
```

Button Text → Button Text Size

You can set the text size of a JavaFX `Button`.

You do so using the CSS property `-fx-text-size`.

This CSS property is explained in the section about **Button CSS Styling**

Button Text → Button Text Wrap

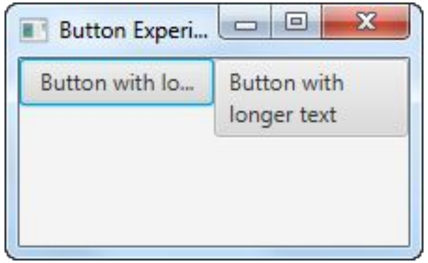
The JavaFX `Button` control supports text wrapping of the button text. By text wrapping is meant that if the text is too long to be displayed on a single line inside the button, the text is broken onto multiple lines.

You enable text wrapping on a JavaFX Button instance using the method `setWrapText()`. The `setWrapText()` method takes a single boolean parameter. If you pass a value of `true` to `setWrapText()` then you enable text wrapping. If you pass a value of `false` to `setWrapText()` then you disable text wrapping.

Here is an example that enables text wrapping on a JavaFX button:

```
button.setWrapText(true);
```

Here is a screenshot of two JavaFX buttons one of which has text wrapping enabled:



Button Font

You can specify what font the text on a JavaFX `Button` should be rendered with via its `setFont()` method. You can learn more about creating fonts in my **JavaFX Fonts** lecture.

Here is an example of setting a font on a JavaFX `Button`:

```
Button button = new Button("Click me!");

Font font = Font.font("Courier New", FontWeight.BOLD, 36);

button.setFont(font);
```

Default Button Mode

A JavaFX `Button` can be set into a ***default mode***. When a `Button` is in default mode it is rendered differently, so the user can see that this is the default button.

The ***default button*** is intended to be used for the "default choice" in a dialog or form. Thus, it becomes easier for the user to select the choice that the user is most likely making most often.

The ***default button*** of a dialog or form has some additional keyboard shortcuts to help the user click it:

- Windows + Linux
  - If no other button has focus, pressing the ENTER keyboard key will activate the default button.
  - If the default button has focus, pressing the ENTER keyboard key will activate the default button.
- Mac
  - Only the default button can be activated by pressing the ENTER keyboard key. All other buttons are activated by pressing the the SPACE keyboard key.

Setting a JavaFX `Button` as the default button is done via its `setDefaultButton()` method:

```
button.setDefaultButton(true);
```

Cancel Button Mode

A JavaFX Button can be set into ***cancel mode***. When a `Button` is in cancel mode it can be activated more easily by pressing the ESC keyboard key - if no other node in the Scene graph consumes this key press.

Setting a JavaFX `Button` in cancel mode (as cancel button) is done via its `setCancelButton()` method.

Here is an example of setting a JavaFX `Button` in cancel mode:

```
buttonDefault.setCancelButton(true);
```

Button Image

It is possible to display an image inside a button next to the button text. The JavaFX `Button` class contains a constructor that can take a `Node` as extra parameter.

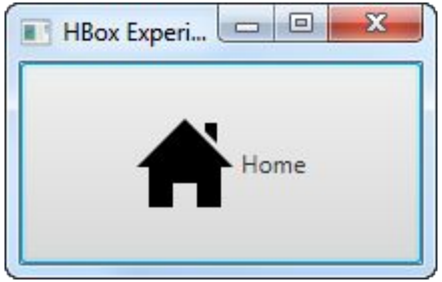
Here is a JavaFX label example that adds an image to the button using an **JavaFX ImageView** component:

Button Image

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import java.io.FileInputStream;

public class ButtonExperiments extends Application  {
   @Override
   public void start(Stage primaryStage) throws Exception {
       primaryStage.setTitle("HBox Experiment 1");
       FileInputStream input = new FileInputStream("resources/images/iconmonstr-home-6-48.png");
       Image image = new Image(input);
       ImageView imageView = new ImageView(image);
       Button button = new Button("Home", imageView);
       Scene scene = new Scene(button, 200, 100);
       primaryStage.setScene(scene);
       primaryStage.show();
   }

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

Button Image

The result of running the above JavaFX `Button` example is an application that looks like this:



Button Size

The JavaFX Button class contains a set of methods you can use to set the button size. The methods controlling the button size are:

```
button.setMinWidth()
button.setMaxWidth()
button.setPrefWidth()

button.setMinHeight()
button.setMaxHeight()
button.setPrefHeight()

button.setMinSize()
button.setMaxSize()
button.setPrefSize()
```

Code Explanation:

The methods `setMinWidth()` and `setMaxWidth()` sets the minimum and maximum width the button should be allowed to have. The method `setPrefWidth()` sets the preferred width of the button. When there is space enough to display a button in its preferred width, JavaFX will do so. If not, JavaFX will scale the button down until it reaches its minimum width.

The methods `setMinHeight()` and `setMaxHeight()` sets the minimum and maximum height the button should be allowed to have. The method `setPrefHeight()` sets the preferred height of the button. When there is space enough to display a button in its preferred height, JavaFX will do so. If not, JavaFX will scale the button down until it reaches its minimum height.

Button Size

The methods `setMinSize()`, `setMaxSize()` and `setPrefSize()` sets both width and height for the button in a single call. Thus, these methods takes both a `width` and a `height` parameter.

For instance, calling:

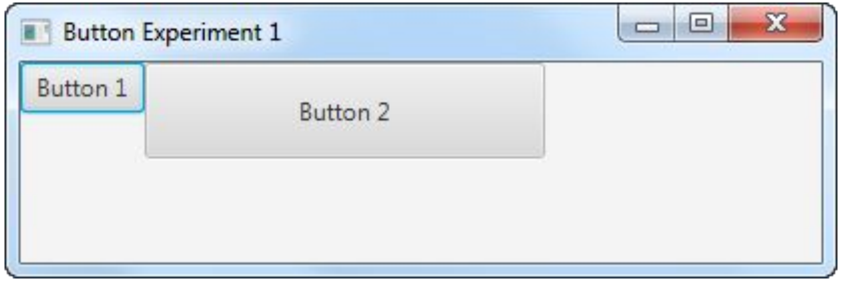
```
button.setMaxSize(100, 200);
```

is equivalent to calling

```
button.setMaxWidth(100);
button.setMaxHeight(200);
```

Button Size

Here is a screenshot of two JavaFX buttons. The first button has the default size calculated from its button text and the layout component it is nested inside. The second button has a preferred width of 200 and height of 48 set on it:



Button Events

In order to respond to the click of a button you need to attach an event listener to the Button object. Here is how that looks:

```
button.setOnAction(new EventHandler() {
   @Override
   public void handle(ActionEvent actionEvent) {
       //... do something in here.
       }
});
```

Button Events

Here is how attaching a click event listener looks with a **Java Lambda expression**:

```
button.setOnAction(actionEvent ->  {
   //... do something in here.
});
```

Button Events

Finally, let us see a full example that changes the text of a **JavaFX Label** when the button is clicked:

Button Events

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Label;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;


public class ButtonExperiments extends Application  {
   @Override
   public void start(Stage primaryStage) throws Exception {
       primaryStage.setTitle("HBox Experiment 1");
       Label label = new Label("Not clicked");
       Button button = new Button("Click");
       button.setOnAction(value ->  {
           label.setText("Clicked!");
       });
       HBox hbox = new HBox(button, label);
       Scene scene = new Scene(hbox, 200, 100);
       primaryStage.setScene(scene);
       primaryStage.show();
   }

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

Button Mnemonic

You can set a mnemonic on a JavaFX `Button` instance. A ***mnemonic*** is a keyboard key which activates the button when pressed in conjunction with the ALT key. Thus, a mnemonic is a keyboard shortcut to activating the button.

The ***mnemonic*** for a button is specified inside the button text. You mark which key is to be used as ***mnemonic*** by placing an underscore character (_) in front of the character in the button text you want to set as mnemonic for that button. The underscore character will not be displayed in the button text.

Here is an example setting a ***mnemonic*** for a button:

```
button.setMnemonicParsing(true);

button.setText("_Click");
```

Notice that it is necessary to first call `setMnemonicParsing()` on the button with a value of `true`. This instructs the button to parse mnemonics in the button text. If you call this method with a value of `false` instead, the underscore character in the button text will just show up as text, and will not be interpreted as a mnemonic.

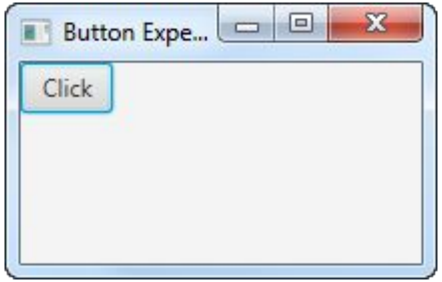
The second line sets the text `_Click` on the button. This tells the button to use the key c as mnemonic. ***Mnemonics*** are case insensitive, so it does not have to be an a uppercase C that activates the button.

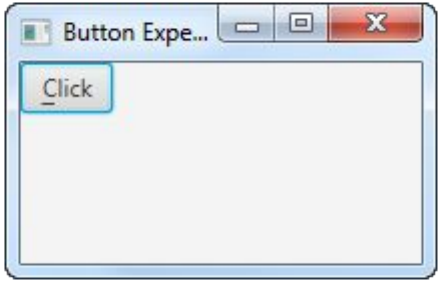
Button Mnemonic → Activate button

To activate the button you can now press `ALT-C` (both at the same time). That will activate the button just as if you had clicked it with the mouse.

You can also first press the `ALT` key once. That will show the mnemonic of the button in the button text. You can then press the `c` key. If you press `ALT` and then `ALT` again, the mnemonic is first shown, then hidden again. When the mnemonic is visible you can activate the button with the mnemonic key alone, without `ALT` pressed at the same time. When the mnemonic is not visible you have to press both `ALT` and the mnemonic key at the same time to activate the button.

Here are two screenshots showing what it looks like when the mnemonic is invisible and visible:





Button CSS Styles

You can style a JavaFX button using CSS styles. The JavaFX `Button` control supports the following CSS styles:

```
-fx-border-width
-fx-border-color
-fx-background-color
-fx-font-size
-fx-text-fill
```

Button CSS Styles

Here is an example setting the background color of a JavaFX button to red:

```
Button button = new Button("My Button");

button.setStyle("-fx-background-color: #ff0000; ");
```

This example sets the style directly on the button via the `setStyle()` method, but you can also style a JavaFX button via style sheets. See the **JavaFX CSS Styling** lecture for more information about using CSS stylesheets with JavaFX.

Button CSS Styles → Example

Here is a JavaFX button example which creates 4 different buttons. Each button has a CSS style set on them. After the code example there is included a screenshot of how the buttons look with the given styling.

Button CSS Styles → Example

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class ButtonExperiments extends Application  {
   @Override
   public void start(Stage primaryStage) throws Exception {
       primaryStage.setTitle("Button Experiment 1");
       Button button1 = new Button("Button 1");
       Button button2 = new Button("Button 2");
       Button button3 = new Button("Button 3");
       Button button4 = new Button("Button 4");
       button1.setStyle("-fx-border-color: #ff0000; -fx-border-width: 5px;");
       button2.setStyle("-fx-background-color: #00ff00");
       button3.setStyle("-fx-font-size: 2em; ");
       button4.setStyle("-fx-text-fill: #0000ff");
       HBox hbox = new HBox(button1, button2, button3, button4);
       Scene scene = new Scene(hbox, 400, 100);
       primaryStage.setScene(scene);
       primaryStage.show();
   }

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

Code Explanation:

The ***first button*** has both the `-fx-border-width` and `-fx-border-color` CSS properties set. This results in a 5 pixel wide red border for the button.

The ***second button*** has the `-fx-background-color` CSS property set. This results in a green background color for the button.

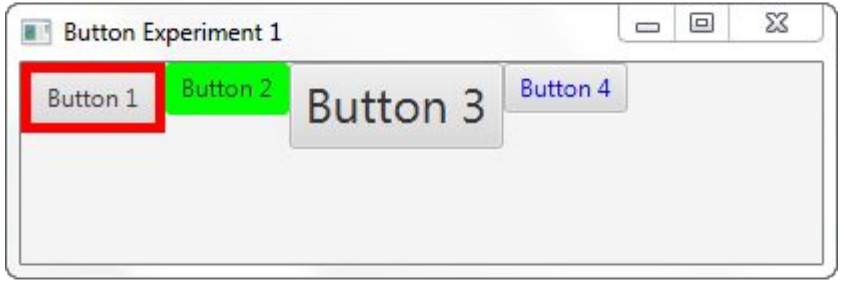
The ***third button*** has the `-fx-font-size` CSS property set. This results in a button with a text that is 2 times as big as normal.

The ***fourth button*** has the `-fx-text-fill` CSS property set. This results in a button with a blue text color.

You can combine the CSS styles for a JavaFX button simply by setting multiple CSS properties on it, like the first button in the example above has.

Button CSS Styles → Example

Here is a screenshot of the 4 JavaFX buttons with their CSS styling:



Disable Button

You can disable a JavaFX `Button` via its `setDisable()` method. The `setDisable()` method takes a boolean parameter which specify if the button should be disabled or not. A value of `true` means the button will be ***disabled***, and a value of `false` means it will not be disabled - which means ***enabled***.

Here is an example of disabling a JavaFX `Button` via its `setDisable()` method:

```
Button button = new Button();
button.setText("Click me!");

// here the app is running, and something happens so the button should now be disabled.

button.setDisable(true);


// again the app runs, and something happens so the button should now be enabled again.

button.setDisable(false);
```

Button FXML

- It is possible to declare a JavaFX `Button` inside a **JavaFX FXML** file.
- We won't explain ***FXML*** in detail here, go to the ***FXML*** lecture for more.
- In this section, we will examine the parts of ***FXML*** related to the JavaFX `Button` control.

Button FXML

Here is a very simple ***FXML*** file example that shows how to declare a `Button`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>
<VBox xmlns:fx="http://javafx.com/fxml" spacing="20">
   <children>
       <Button fx:id="button1" text="Click me!" onAction="#buttonClicked"/>
   </children>
</VBox>
```

Code Explanation:

It is the `<Button>` element that declares the `Button` control. The `fx:id` attribute can be used to wire up the declared `Button` to a `Button` member variable inside an ***FXML*** `Controller` object.

The text attribute is used to set the corresponding text property value in the `Button` instance created for this `Button` declaration. That is the text displayed on the button.

The `onAction` attribute is used to link the `Button's onAction` event to a method in the ***FXML*** `controller` object. In this example it is the method named `buttonClicked()` in the ***FXML*** controller.

Button FXML

Here is how the corresponding ***FXML*** `controller` object's class looks.

Notice the name of the `button1 Button` member variable matches the `fx:id` attribute value in the ***FXML*** file. This member variable must be `public`, by the way.

Notice also the method `buttonClicked()` which is referenced from the `Button onAction` attribute in the ***FXML*** file.

```
import javafx.event.Event;
import javafx.fxml.FXML;
import javafx.scene.control.Button;

public class ButtonFXMLController {
   public Button button1 = null;
   private int   button1ClickCount = 0;

   @FXML
   public void buttonClicked(Event e){
       this.button1ClickCount++;
       String text = "Button1 clicked " + this.button1ClickCount + " times";
       System.out.println(text);
       button1.setText(text);
   }
}
```

Button FXML → controller

And here is the full JavaFX app that loads the ***FXML*** file and creates the ***FXML*** `controller` etc. :

Button FXML → controller

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import java.io.File;
import java.net.URL;

public class ButtonFXMLExample extends Application {
   public static void main(String[] args) {
       launch(args);
   }

   @Override
   public void start(Stage primaryStage) throws Exception {
       FXMLLoader loader = new FXMLLoader();
       ButtonFXMLController controller = new ButtonFXMLController();
       loader.setController(controller);
       File fxmlFile = new File("assets/fxml/button-example.fxml");
       URL fxmlUrl = fxmlFile.toURI().toURL();
       loader.setLocation(fxmlUrl);
       VBox vbox = loader.<VBox>load();
       Scene scene = new Scene(vbox);
       primaryStage.setScene(scene);
       primaryStage.setTitle("Button FXML Example");
       primaryStage.setWidth(300);
       primaryStage.setHeight(300);
       primaryStage.show();
   }
}
```

Button Transformations

You can add **JavaFX transformations** to a JavaFX `Button` such as scaling, rotation, translation etc.

You add transformation objects to a JavaFX `Button` like this:

```
button.getTransforms().add(scaleTransformation);
```

Button Transformations

Here is a full example of adding a `Scale` transformation to a JavaFX Button:

Button Transformations

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.scene.transform.Scale;
import javafx.stage.Stage;

public class ButtonTransformationExample extends Application {
   public static void main(String[] args) {
       launch(args);
   }

   @Override
   public void start(Stage primaryStage) {
       Button button = new Button();
       button.setText("Click me!");
       button.setOnAction((event) -> {
           System.out.println("Button clicked!");
       });
       Scale scaleTransformation = new Scale();
       scaleTransformation.setX(3.0);
       scaleTransformation.setY(2.0);
       scaleTransformation.setPivotX(0);
       scaleTransformation.setPivotY(0);
       button.getTransforms().add(scaleTransformation);
       VBox  vbox  = new VBox(button);
       Scene scene = new Scene(vbox);
       primaryStage.setScene(scene);
       primaryStage.setWidth(512);
       primaryStage.setHeight(256);
       primaryStage.show();
   }
}
```

END