Chapter 12 Java FX

JDK and JavaFX

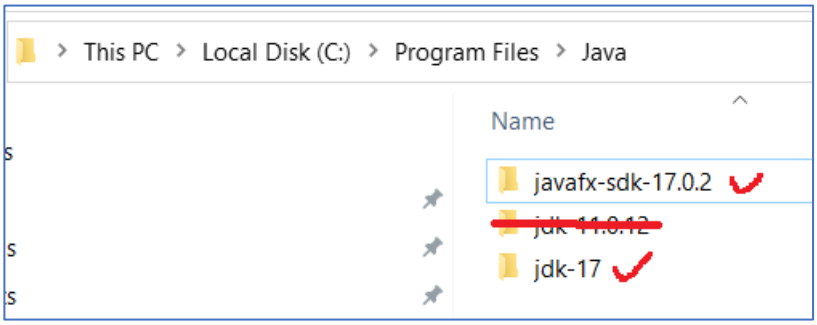
JavaFX is removed from JDK. You need to install it separately after you have installed JDK.

****

Download JavaFX

Go to https://gluonhq.com/products/javafx/ and download JavaFX SDK for your operating system.

The downloaded SDK is a zip file. Extract it and place it in the same directory as JDK.



Once installed, you need to configure the settings in IDEs to compile and run JavaFX programs.

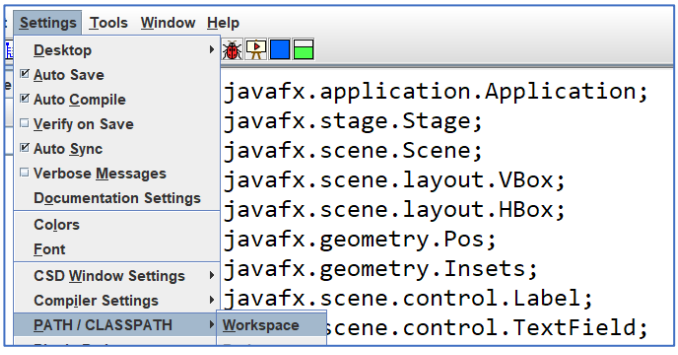
I recommend using different IDEs for different type of applications as shown in the following table.

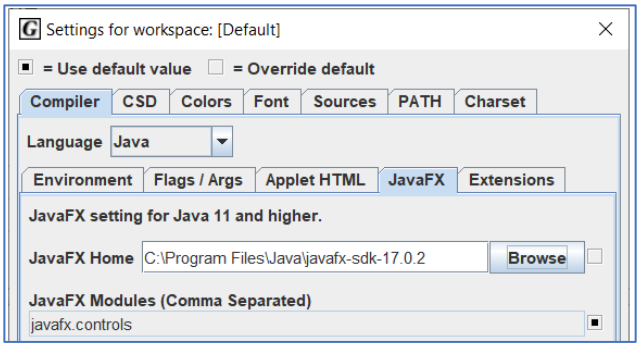
JavaFX FXML Application	"Regular" JavaFX Application	Java Swing Application
IntelliJ	jGRASP	NetBeans
	Eclipse	

Instructions for configuring the IDE's settings

****

Using JavaFX in jGRASP





****

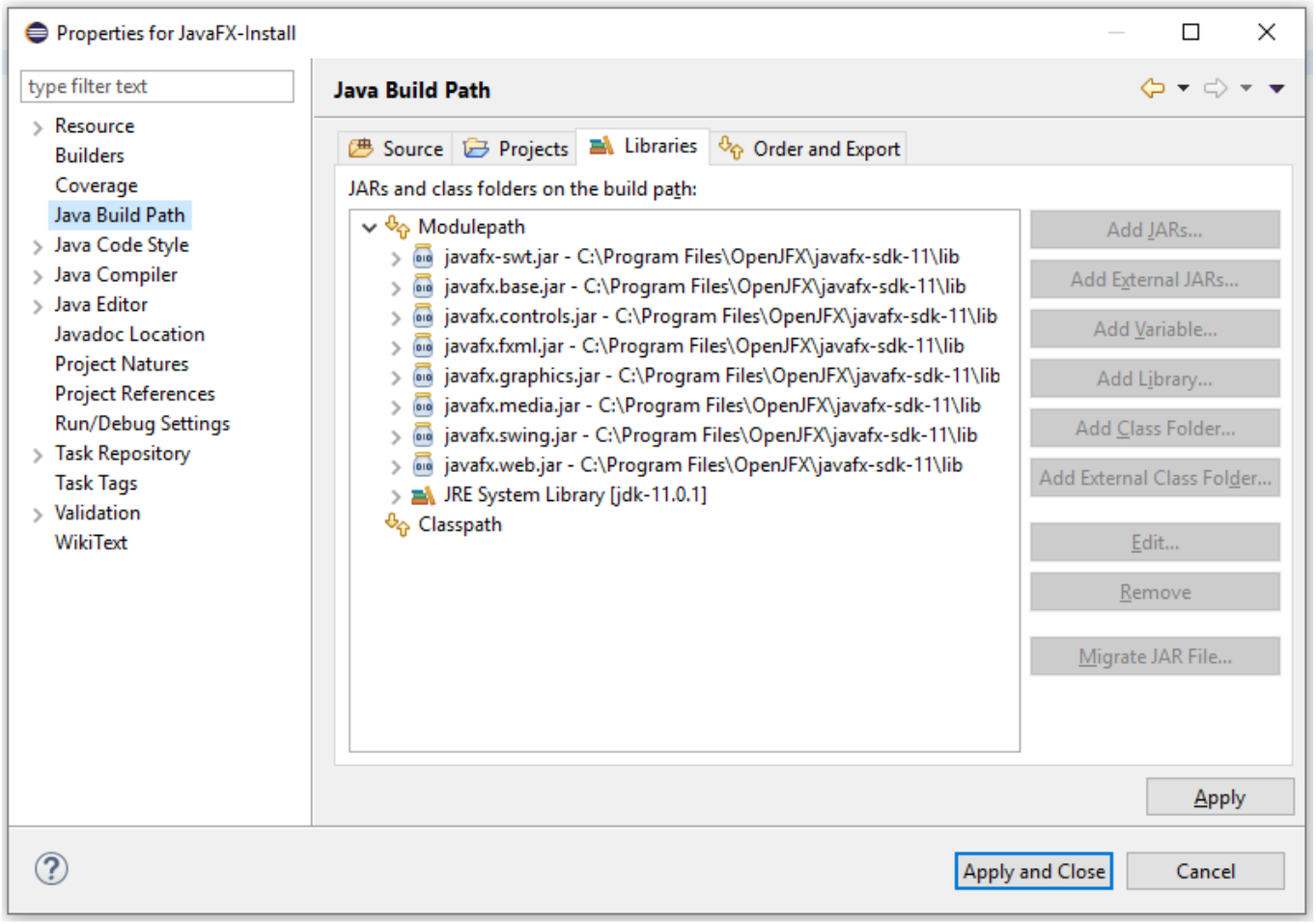
https://gist.github.com/stevenliebregt/bc62a382fc43064136b662ee62172ab3

Using JavaFX in Eclipse

To use JavaFX 11 with Eclipse you will need to do 2 things, add the module path to the VM arguments, and add the libraries to Eclipse.

Adding the libraries to Eclipse

To add the libraries, edit the build path, and add the JavaFX .jar files to the module path, do this by clicking on the `Add External JARS` button, and selecting all the JavaFX modules. The result will look something like this.



After doing this, the JavaFX imports should now be fixed.

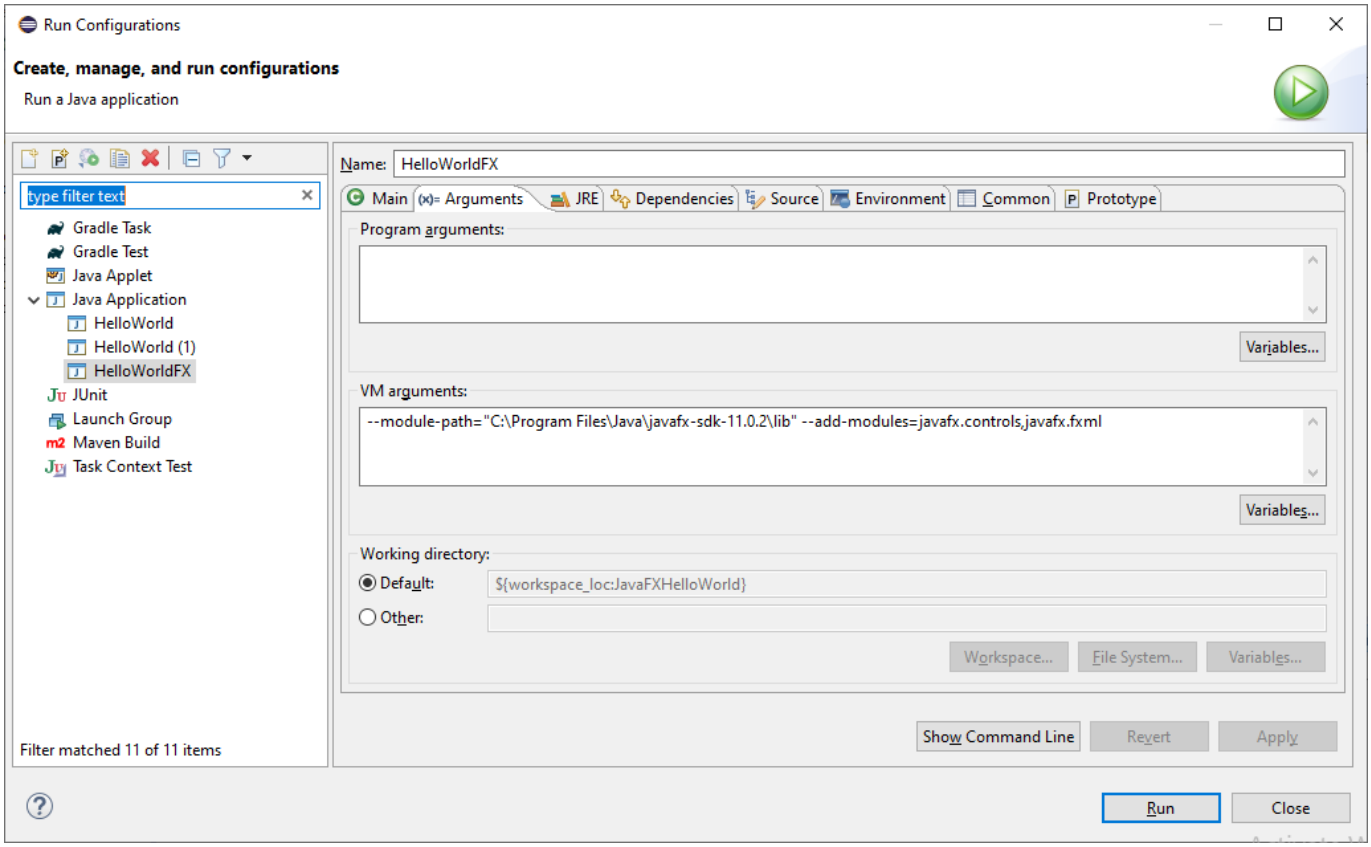
To also be able to compile and run the code, you should add the modules to the VM arguments too.

To do so, edit the `Run Configurations` and go to the `Arguments` tab. In the `VM arguments` field, enter the following lines, with the path changed to your actual OpenJFX install location.

```
--module-path="C:\Program Files\OpenJFX\javafx-sdk-11\lib" --add-modules=javafx.controls,javafx.fxml
```

If you need other JavaFX modules than `controls` and `fxml`, then these should also be added to the same `--add-modules` line, separated by a comma.

After this, your configuration will look like this.



****

Using JavaFX in IntelliJ

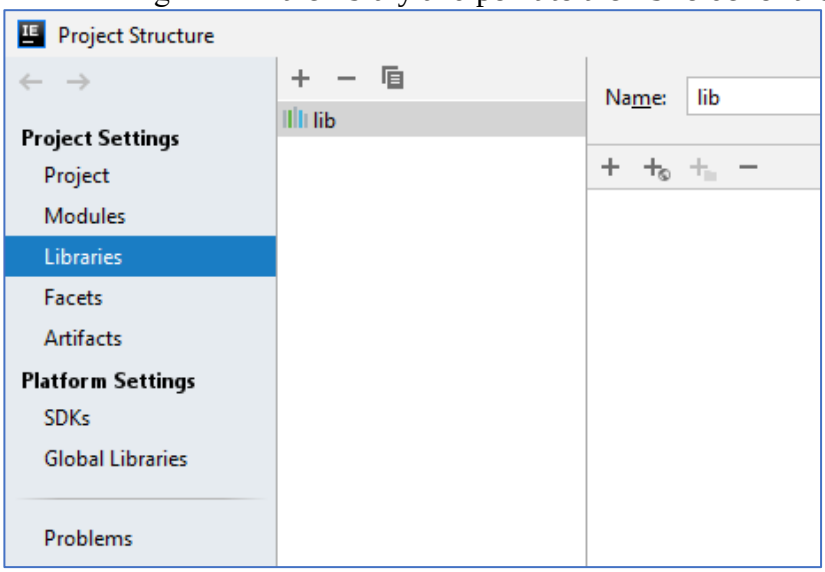
Go to **`File -> Project Structure -> Project`**

Set the project SDK and set the language level.

Create a library

Go to **File -> Project Structure -> Libraries**

Click “+” sign to add the library and point to the **lib** folder of the **JavaFX SDK**.



Define a Global Variable and Add VM options

Go to Preferences (File -> Settings) -> Appearance & Behavior -> Path Variables, and define the name of the variable as PATH_TO_FX, and browse to the **lib** folder of the **JavaFX SDK** to set its value, and click apply.

Then you can refer to this global variable when setting the VM options as:

```
--module-path ${PATH_TO_FX} --add-modules javafx.controls,javafx.fxml
```

Alternatively, you can add VM option to each individual project

Click on **`Run -> Edit Configurations...`** and add the **VM options**:

```
--module-path “C:\Program Files\Java\javafx-sdk-11.0.2\lib" --add-modules javafx.controls,javafx.fxml
```

****

12.1 Graphical User Interfaces

GUI (“gooey”)

In a GUI application, the user determines the order in which things happen.

The program responses to the actions of the user to handle the event.

12.2 Introduction to JavaFX

JavaFX is a standard Java library for creating GUI applications, as well as applications that display 2D and 3D graphics.

Controls

They are visual objects.

- Label
- TextField
- Button

Stages and Scenes

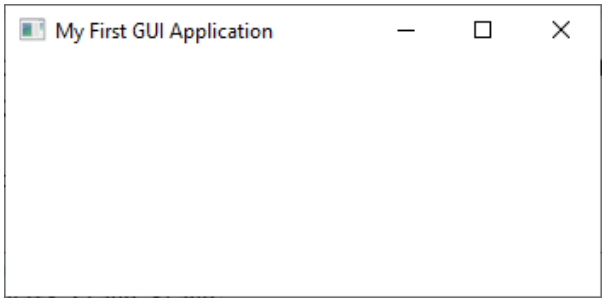
JavaFX Application	Theater
The stage is a window.	Stage
The scene is a collection of GUI controls contained within the window.	Scene
GUI objects make up the scene.	Actors

The Application **Class**

It is an abstract class that has an abstract method called **start()**, which is the entry point for the JavaFX application.

All JavaFX applications must extend it.

```java
import javafx.application.Application;
import javafx.stage.Stage;

/**
 *  A simple JavaFX GUI application
 */

public class MyFirstGUI extends Application
{
   public static void main(String[] args)
   {
      launch(args);   // Launch the application.
   }

   @Override
   public void start(Stage primaryStage)
   {
      // Set the window's title.
      primaryStage.setTitle("My First GUI Application");

      primaryStage.show();   // Show the window.
   }
}
```



The program crates a stage but does not have a scene.

The main() method does only one thing: calling the **launch()** method.

The launch() method is inherited from the Application class.
- It creates a Stage object
- It calls the start() method and passes a reference to the Stage object as an argument.

The start() method must be overridden.
- It has a parameter, primaryStage, to refer to the Stage object passed by the launch() method.

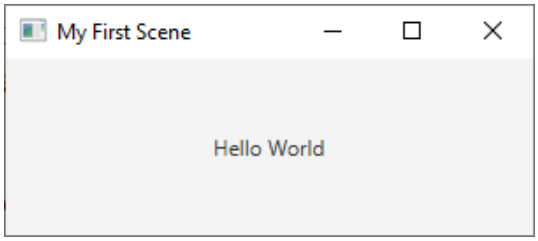
12.3 Creating Scenes

A scene is a collection of controls and other objects.

The steps to creating a scene:
1. Create the controls
2. Create a layout container of some type, and add controls to the container
3. Create a Scene object, and add the container to the Scene object
4. Add the Scene object to the stage.

HelloWorld2.java

```
public void start(Stage primaryStage)
{
   // Create a Label component.
   Label messageLabel = new Label("Hello World");

   // Put the Label in an HBox.
   HBox hbox = new HBox(messageLabel);

   // Create a Scene with the HBox as its root node.
   Scene scene = new Scene(hbox, 300, 100);

   // Set the scene's alignment to center.
   hbox.setAlignment(Pos.CENTER);

   // Add the Scene to the Stage.
   primaryStage.setScene(scene);

   // Set the stage title.
   primaryStage.setTitle("My First Scene");

   // Show the window.
   primaryStage.show();
}
```



Creating Controls

- Label control

Creating Layout Containers

Use layout containers to arrange the positions of controls on the scene.

HBox	Arranges controls in a single horizontal row
VBox	Arranges controls in a single vertical row
GridPane	Arranges controls in a grid with rows and columns

HBox hbox = new HBox(new Lable("Hello")); //Create an HBox and add a Label control

HBox hbox = new HBox(new Lable("Hello"), new Label("World")); //add two Label controls

Creating a Scene **Object**

Scene scene = new Scene(hbox); //create a Scene and add the HBox as the root node
//Label object with in the HBox is the leaf node.

Adding the Scene **Object to the Stage**

primaryStage.setScene(scene);

Setting the Size of the Scene

The default size is very small, just enough to display the contents of the scene. It is known as the preferred size.

You can add width in pixel and height in pixel as arguments.

Aligning Controls in an HBox **Layout Container**

hbox.setAlignment(Pos.CENTER);

Pos is in javafx.geometry package.

Pos.TOP_LEFT	Pos.TOP_CENTER	Pos.TOP_RIGHT
Pos.CENTER_LEFT	Pos.CENTER	Pos.CENTER_RIGHT
Pos.BOTTOM_LEFT	Pos.BOTTOM_CENTER	Pos.BOTTOM_RIGHT

12.4 Displaying Images

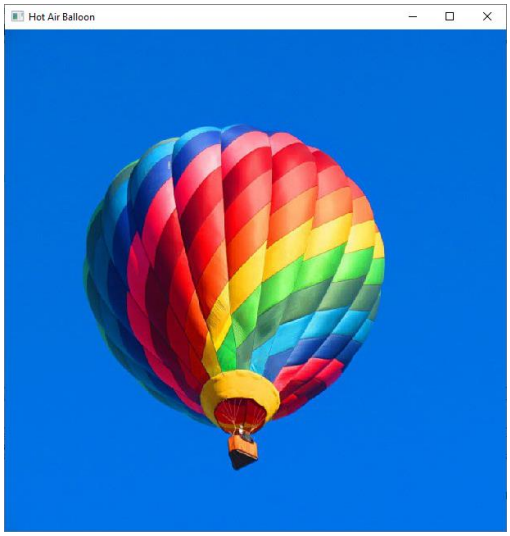
Two-step process:

1. Load the image into memory
2. Display the image

Import both *Image* and *ImageView* classes which are in the javafx.scene.image package.

ImageDemo.java

```
public void start(Stage primaryStage)
{
   // Create an Image component.
   Image image = new Image("file:HotAirBalloon.jpg");

   // Create an ImageView control.
   ImageView imageView = new ImageView(image);

   // Put the ImageView in an HBox.
   HBox hbox = new HBox(imageView);

   // Create a Scene with the HBox as its root node.
   Scene scene = new Scene(hbox);

   // Add the Scene to the Stage.
   primaryStage.setScene(scene);

   // Set the stage title.
   primaryStage.setTitle("Hot Air Balloon");

   // Show the window.
   primaryStage.show();
}
```



- The Image class supports bmp, jpeg, gif, and png file types.
- file: is the protocol indicating the file in on the local computer.
- Same directory

- Add Image object to ImageView object
- Add ImageView object to HBox

Loading Images from an Internet Location

Image image = new Image("http://www.gaddisbooks.com/images/HotAirBallon.jpg");

Setting the Size of an Image

imageView.setFitWidth(100);
imageView.setFitHeight(100);

Preserving the Image's Aspect Ratio

imageView.setPreserveRatio(true); true or false

Changing an ImageView Image

imageView.setImage(newImage);

12.5 More about the HBox, VBox, and GridPane Layout Containers

The HBox Layout Container

It arranges one or more controls in a single horizonal row.

You can also add **spacing** between images and **padding** around images in pixels.

```
// Put the ImageViews in an HBox.
HBox hbox = new HBox(moonIView, shipIView, sunsetIView);

// Put the ImageViews in an HBox with 10 pixels spacing.
HBox hbox = new HBox(10, moonIView, shipIView, sunsetIView);

// Put 30 pixels of padding around the HBox.
hbox.setPadding(new Insets(30));
```

The VBox Layout Container

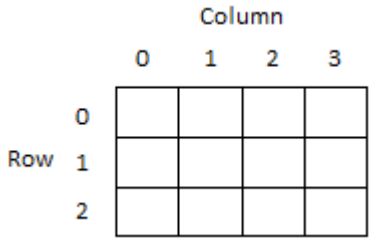
It arranges one or more controls in a single vertical row.

You can also add **spacing** between images and **padding** around images in pixels.

```
// Put the ImageViews in a VBox with 10 pixels spacing.
VBox vbox = new VBox(10, moonIView, shipIView, sunsetIView);

// Put 30 pixels of padding around the VBox.
vbox.setPadding(new Insets(30));
```

The GridPane **Layout Container**

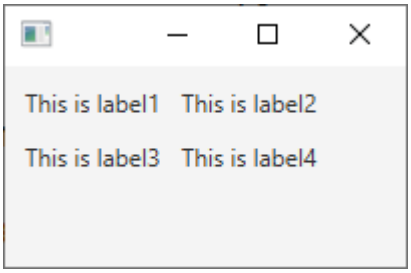
It arranges the contents in a grid with columns and rows.



GridPaneDemo.java

- Add controls at specific positions
- Set gaps between columns and rows

```
// Create some Label controls.
Label label1 = new Label("This is label1");
Label label2 = new Label("This is label2");
Label label3 = new Label("This is label3");
Label label4 = new Label("This is label4");

// Create a GridPane.
GridPane gridpane = new GridPane();

// Add the Labels to the GridPane.
gridpane.add(label1, 0, 0);
gridpane.add(label2, 1, 0);
gridpane.add(label3, 0, 1);
gridpane.add(label4, 1, 1);
gridpane.setHgap(10);
gridpane.setVgap(10);
gridpane.setPadding(new Insets(10));

// Create a Scene with the GridPane as its root node.
// The Scene is 200 pixels wide by 100 pixels high.
Scene scene = new Scene(gridpane, 200, 100);
```



Using Multiple Layout Containers in the Same Screen

- Use multiple layout containers
- Nest a layout inside another layout

NestedLayout.java

- Two images are added to a VBox
- The Label is added to the GridPane at column 0, row 0
- The VBox is added to the GridPane at column 1, row 0

```
// Create Image objects.
Image finlandImage = new Image("file:Finland.png");
Image germanyImage = new Image("file:Germany.png");

// Create the ImageView objects.
ImageView finlandIView = new ImageView(finlandImage);
ImageView germanyIView = new ImageView(germanyImage);

// Create a Label control.
Label messageLabel = new Label("Flags of Finland and Germany");

// Create a VBox layout container for the images.
VBox vbox = new VBox(10, finlandIView, germanyIView);

// Create a GridPane layout container.
GridPane gridpane = new GridPane();

// Add the Label and the VBox to the GridPane.
gridpane.add(messageLabel, 0, 0);    // Column 0, Row 0
gridpane.add(vbox, 1, 0);            // Column 1, Row 0

// Set the gap size between the Gridpane's columns.
gridpane.setHgap(10);

// Set the GridPane's padding.
gridpane.setPadding(new Insets(20));

// Create a Scene with the GridPane as its root node.
Scene scene = new Scene(gridpane);
```

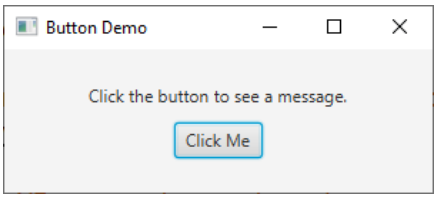
12.6 Button Controls and Events

Clicking a button causes an action event that the program should handle.

ButtonDeme.java

- Create a Button control and add it to the VBox
- The action event is not handled.

```
// Create a Label control.
Label myLabel = new Label("Click the button to see a message.");

// Create a Button control.
Button myButton = new Button("Click Me");

// Put the Label and Button in a VBox with 10 pixels of spacing.
VBox vbox = new VBox(10, myLabel, myButton);

// Create a Scene with the VBox as its root node.
Scene scene = new Scene(vbox, 300, 100);

// Set the scene's alignment to center.
vbox.setAlignment(Pos.CENTER);
```



Handling Events

When an event occurs, the control responsible for the event creates an event object that contains information about the event.

The control that triggered the event is the event source, for example, button.

The event object is the instance of the Event class or one of its subclasses.

- When a Button is clicked, the ActionEvent object is created.
- ActionEvent is the subclass of the Event class.

To handle the event, one or more event handlers is needed to connect to the event source.

The event handler is an object that responds to the event.

If the event source and the event handler are connected, a specific method in the event handler is called and the event object is passed as an argument to the method.

- This process is also called *event firing*.

Writing Event Handlers

The class must implement the *EventHandler* interface.

This interface has a void method named *handle()*.

```
class ButtonClickHandler implements EventHandler<ActionEvent> {

	@Override
	public void handle(ActionEvent event) {
	   myLabel.setText("Thanks for clicking the button!");
	}
}
```

Registering an Event Handler

Connecting the control with the handler.

```
myButton.setOnAction(new ButtonClickHandler());
```

Now, clicking *myButton* will trigger the ButtonClickHandler's handle() to be executed.

12.7 Reading Input with TextField Controls

```
class CalcButtonHandler implements EventHandler<ActionEvent>
{
   @Override
   public void handle(ActionEvent event)
   {
      // Get the kilometers.
      Double kilometers = Double.parseDouble(kiloTextField.getText());

      // Convert the kilometers to miles.
      Double miles = kilometers * 0.6214;

      // Display the results.
      resultLabel.setText(String.format("%,.2f miles", miles));
   }
}
```

kiloTextField.getText() //return the content in the TextField as a string

resultLabel.setText("Hello"); //set a string to the Label control

12.8 Using Anonymous Inner Classes and Lambda Expressions to Handle Events

Using Anonymous Inner Classes to Create Event Handlers

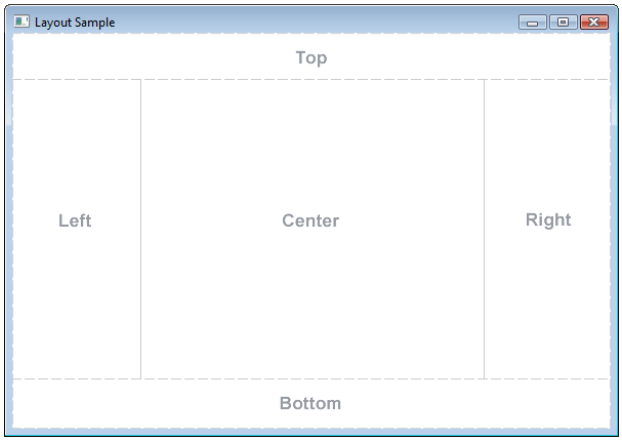
```
public void start(Stage primaryStage)
{
   // Create a Label control.
   Label myLabel = new Label("Click the button to see a message.");

   // Create a Button control.
   Button myButton = new Button("Click Me");
   myButton.setOnAction(new EventHandler<ActionEvent>()
   {
      @Override
      public void handle(ActionEvent event)
      {
         myLabel.setText("Thank you for clicking the button.");
      }
   });

   // Put the Label and Button in a VBox with 10 pixels of spacing.
   VBox vbox = new VBox(10, myLabel, myButton);
```

Using Lambda Expressions to Create Event Handlers

```
public void start(Stage primaryStage)
{
   // Create a Label control.
   Label myLabel = new Label("Click the button to see a message.");

   // Create a Button control.
   Button myButton = new Button("Click Me");
   myButton.setOnAction(event ->
   {
      myLabel.setText("Thank you for clicking the button.");
   });

   // Put the Label and Button in a VBox with 10 pixels of spacing.
   VBox vbox = new VBox(10, myLabel, myButton);
```

12.9 The BorderPane Layout Container

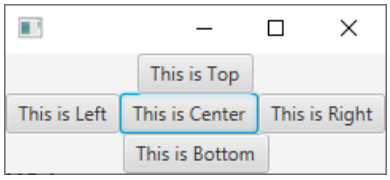
It displays the content in five regions: top, bottom, left, right, and center.



```
Button centerButton = new Button("This is Center");
Button topButton = new Button("This is Top");
Button bottomButton = new Button("This is Bottom");
Button leftButton = new Button("This is Left");
Button rightButton = new Button("This is Right");

// Add each button to its own layout container.
HBox centerHBox = new HBox(centerButton);
HBox topHBox = new HBox(topButton);
HBox bottomHBox = new HBox(bottomButton);
VBox leftVBox = new VBox(leftButton);
VBox rightVBox = new VBox(rightButton);

// Set the alignment for the top and bottom.
topHBox.setAlignment(Pos.CENTER);
bottomHBox.setAlignment(Pos.CENTER);

// Create a BorderPane.
BorderPane borderPane =
   new BorderPane(centerHBox, topHBox, rightVBox,
                  bottomHBox, leftVBox);
```

```
// Create a BorderPane.
BorderPane borderPane = new BorderPane();

// Add the buttons to the BorderPane's regions.
borderPane.setCenter(centerHBox);
borderPane.setTop(topHBox);
borderPane.setBottom(bottomHBox);
borderPane.setLeft(leftVBox);
borderPane.setRight(rightVBox);
```

```
// Create a BorderPane with a node in the center.
BorderPane borderPane = new BorderPane(centerHBox);

// Add the buttons to the BorderPane's regions.
borderPane.setTop(topHBox);
borderPane.setBottom(bottomHBox);
borderPane.setLeft(leftVBox);
borderPane.setRight(rightVBox);
```



12.10 The ObserveableList Interface

An object that implements the ObserveableList is a special type of list that can fire an event handler any time an item in the list changes.