# Topic 6: JavaFX

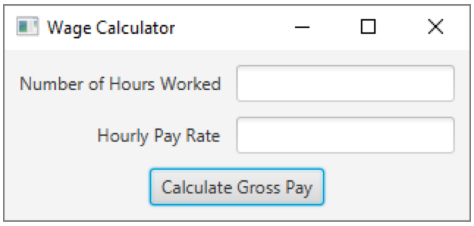
Part a: The basics of JavaFX

# JavaFX

## Basics

# Graphical User Interfaces

- A GUI is a graphical window or windows that provide interaction with the user.
- A window in a GUI commonly consists of several ***controls*** that present data to the user and/or allow interaction with the application.
- Some of the common GUI *controls* are buttons, labels, text fields, check boxes, and radio buttons.






# Graphical User Interfaces (2)

- Programs that operate in a GUI environment must be *event-driven*.
- An *event* is an action that takes place within a program, such as the clicking of a button.
- Part of writing a GUI application is creating event listeners.
- An *event listener* is a method that automatically executes when a specific event occurs.

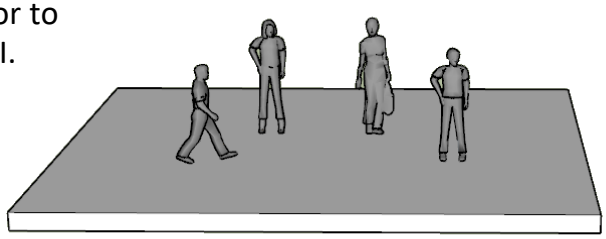
# Introduction to JavaFX

- *JavaFX* is a Java library for developing applications that employ graphics.
- You can use it to create:
  - GUI applications, as well as applications that display 2D and 3D graphics
  - standalone graphics applications that run on your local computer
  - applications that run from a remote server
  - applications that are embedded in a Web page

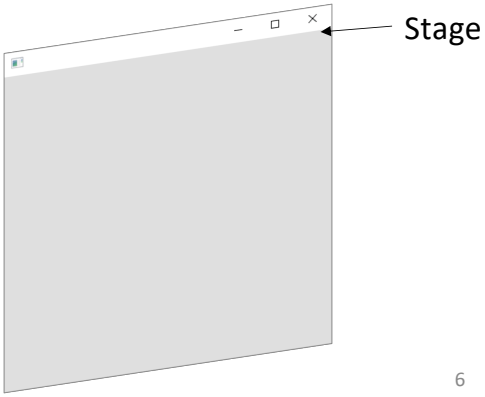


# Introduction to JavaFX (2)

- JavaFX uses a theater metaphor to describe the structure of a GUI.
- A theater has a stage
- On the stage, a scene is performed by actors

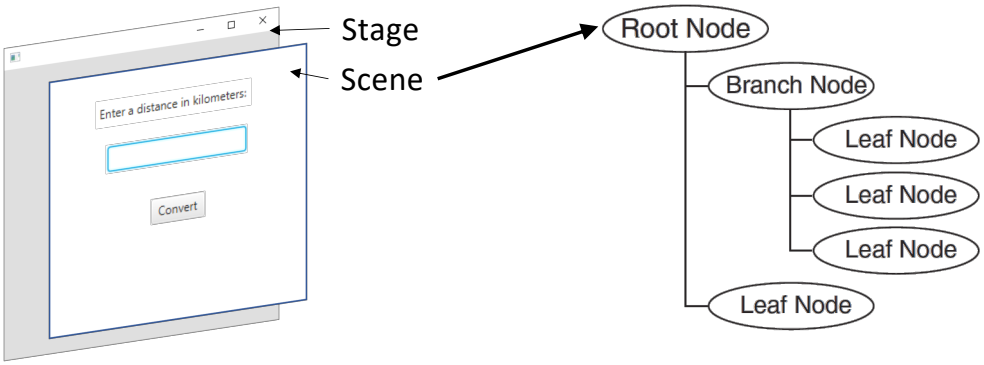


- In JavaFX, the stage is an empty window



# Introduction to JavaFX (3)

- The scene is a collection of GUI objects (controls) that are contained within the window.
- You can think of the GUI objects as the actors that make up the scene.
- In memory, the GUI objects in a scene are organized as nodes in a *scene graph*, which is a tree-like hierarchical data structure.






# Introduction to JavaFX (4)

- The scene graph has three types of nodes:
  - **Root Node:** There is only one root node, which is the parent of all the other nodes in the scene graph.
  - **Branch Node:** A node that can have other nodes as children
  - **Leaf Node:** A node that cannot have children

- The `Application` Class
  - An abstract class that is the foundation of all JavaFX applications
  - JavaFX applications must extend the `Application` class
  - The `Application` class has an abstract method named `start`, which is the entry point for the application
  - Because the `start` method is abstract, you must override it

# General Layout of a JavaFX Program

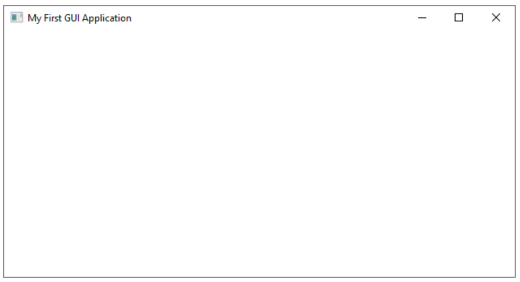
- Various `import` statements
- A class that extends the `Application` class
- A `start` method
- A `main` method

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
// ... Other import statements

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

   @Override
   public void start(Stage primaryStage)
   {
      // Insert startup code here.
   }
}
```



- MyFirstGUI.java

```
1  import javafx.application.Application;
2  import javafx.stage.Stage;
3
4  /**
5   * A simple JavaFX GUI application
6   */
7
8  public class MyFirstGUI extends Application
9  {
10    public static void main(String[] args)
11    {
12       // Launch the application.
13       launch(args);
14    }
15
16    @Override
17    public void start(Stage primaryStage)
18    {
19       // Set the stage title.
20       primaryStage.setTitle("My First GUI Application");
21
22       // Show the window.
23       primaryStage.show();
24    }
25 }
```



## Creating Controls

- Process for creating a control:
  - Import the class for the control from the necessary `javafx` package. Example:

```
import javafx.scene.control.Label;
```

  - Instantiate the class, calling the desired constructor. Example:

```
Label messageLabel = new Label("Hello
World");
```



## Creating Controls (2)

- Another example: Creating a Button
  - Import the `Button` class from the necessary `javafx` package:

```
import javafx.scene.control.Button;
```

  - Instantiate the class, calling the desired constructor:

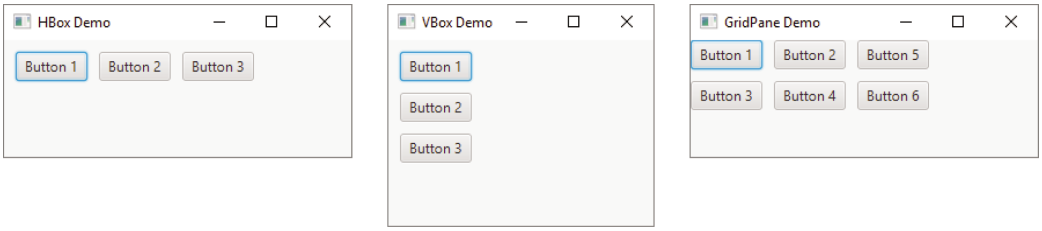
```
Button mybutton = new Button("Click Me");
```

# Layout Containers

- You use layout containers to arrange the positions of controls on the screen.
- JavaFX provides many layout containers.
- We will start with these:
  - `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.



# Layout Containers (2)




The layout container classes are in the `javafx.scene.layout` package.

# Adding Controls to a Layout Container

VBox


```
Button b1 = new Button("Button 1");
Button b2 = new Button("Button 2");
Button b3 = new Button("Button 3");

VBox vbox = new VBox(b1, b2, b3);
```



# Creating a Scene

- To create a scene, you instantiate the `Scene` class (in the `javafx.scene` package)
- Then, you add your root node to the scene

```
// Create a Label control.
Label messageLabel = new Label("Hello World");

// Create an HBox container and add the Label.
HBox hbox = new HBox(messageLabel);

// Create a Scene and add the HBox as the root node.
Scene scene = new Scene(hbox);
```

# Adding the Scene to the Stage

- Once you've created a `Scene` object, you add it to the application's stage.
- The stage is an instance of the `Stage` class (from the `javafx.stage` package)
- You do not have to instantiate the `Stage` class, however. It is created automatically, and passed as an argument to the `start` method.

```
@Override
public void start(Stage primaryStage)
{

}
```



# Displaying Images

- You need two JavaFX classes:
  - The `Image` class, from the `javafx.scene.image` package
    - Use this class to load an image into memory
  - The `ImageView` class, also from the `javafx.scene.image` package
    - Use this class to create a node that displays the image
    - ImageDemo.java

## More About HBox and VBox Containers

- To add spacing between the items in an HBox or VBox:

```
HBox hbox = new HBox(10, label1, label2, label3);
```

```
VBox vbox = new VBox(20, button1, button2,
button3);
```



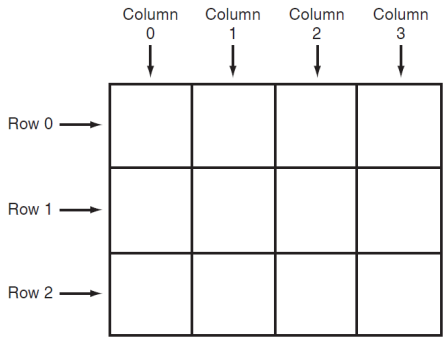
## More About HBox and VBox Containers (2)

- *Padding* is space that appears around the inside edge of a container.
- The HBox and VBox containers have a setPadding method.
- You pass an Insets object as an argument to the setPadding method.
- The Insets object specifies the number of pixels of padding.
- The Insets class is in the javafx.geometry package.

```
hbox.setPadding(new Insets(10));
```

# The `GridPane` Layout Container

- Arranges its contents in a grid with columns and rows.
- The columns and rows are identified by indexes.




- The `GridPane` class is in the `javafx.scene.layout` package.
- First, you instantiate the `GridPane` class, using the no-arg constructor:

```
GridPane gridpane = new GridPane();
```

- Then, you add controls to the container using the add method:

```
gridPaneObject.add(control, column, row);
```



# The `GridPane` Layout Container (3)

```
// 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);  // Column 0, Row 0
gridpane.add(label2, 1, 0);  // Column 1, Row 0
gridpane.add(label3, 0, 1);  // Column 0, Row 1
gridpane.add(label4, 1, 1);  // Column 1, Row 1
```

# The `GridPane` Layout Container (4)

- By default, there is no space between the rows and columns in a `GridPane`.
- To add horizontal spacing between the columns in a `GridPane`, call the container's `setHgap` method.
- To add vertical spacing between the rows in a `GridPane`, call the container's `setVgap` method.

```
GridPane gridpane = new GridPane();
gridpane.setHgap(10);
gridpane.setVgap(10);
```

- The `GridPane` container also has a `setPadding` method to set the padding around the container's inside edge:

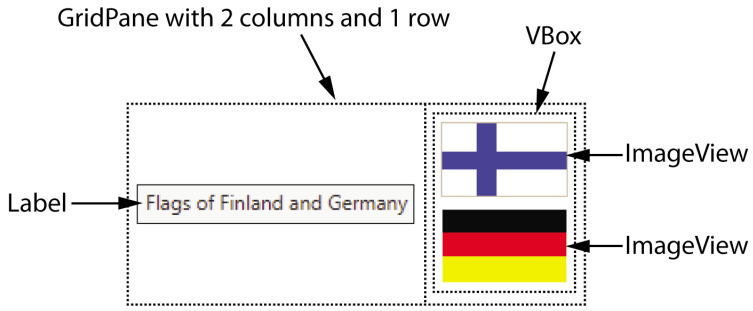
```
GridPane gridpane = new GridPane();
gridpane.setPadding(new Insets(10));
```

- GridPaneButtonDemo.java



# Using Multiple Layout Containers

- To get the particular screen layout that you desire, you will sometimes have to nest layout containers.




- NestedLayout.java