Lecture 11

Graphical User Interface

References:

1. Tony Gaddis, Chapters 12, Starting out with Java: From Control Structures through Objects, 7 edition

1

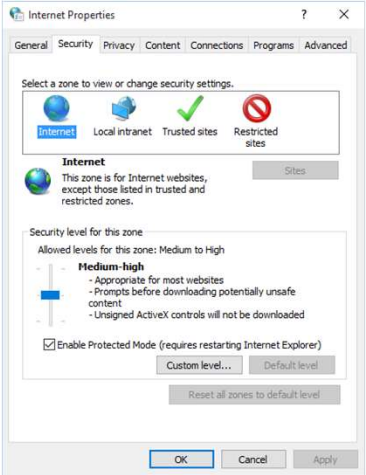
Topics

- Graphical User Interfaces
- Introduction to JavaFX
- Creating Scenes
- Displaying Images
- More About the `HBox`, `VBox`, and `GridPane` Layout Containers
- `Button` Controls and Events
- Reading Input with `TextField` Controls
- The `BorderPane` Layout Container
- The `ObserveableList` Interface

2

Graphical User Interfaces (1 of 3)

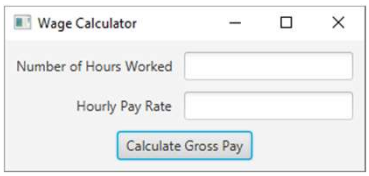
- *Graphical user interface (GUI)*
  - A graphical window or windows that provide interaction with the user



3

Graphical User Interfaces (2 of 3)

- A window in a GUI
  - Consists of several *controls*
    - Present data to the user
    - Allow interaction with the application
    - E.g., Buttons, labels, text fields, check boxes, and radio buttons



4

Graphical User Interfaces (3 of 3)

- Event-driven in a GUI
- An *event* is an action
  - That takes place within a program
  - E.g., the clicking of a button
- Event listener
  - A method that automatically executes when a specific event occurs

5

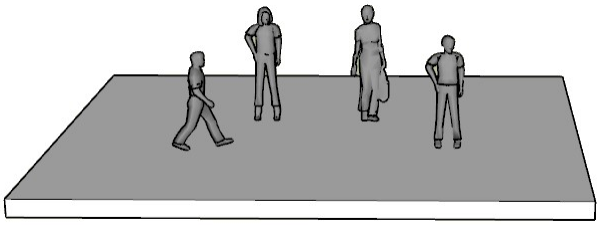
Introduction to JavaFX (1 of 7)

- Java GUI Frameworks
  - AWT (Abstract Window Toolkit), Swing, JavaFX
- JavaFX is a Java library
  - Used to develop applications that employ graphics
- You can use it to create:
  - GUI applications displaying 2D and 3D graphics
  - Standalone graphics applications running on your local computer
  - Applications running on a remote server
  - Applications embedded in a Web page

6

Introduction to JavaFX (2 of 7)

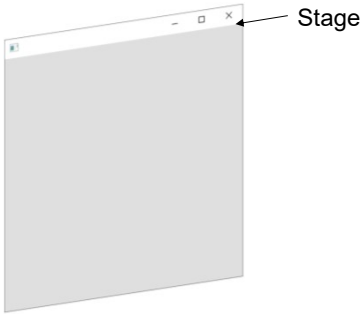
- JavaFX uses a theater metaphor
  - To describe the structure of a GUI
- A theater has a stage
  - On the stage, a scene performed by actors



7

Introduction to JavaFX (3 of 7)

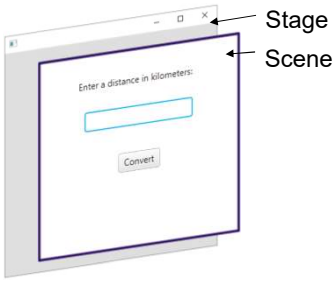
- Stage
  - is an empty window



8

Introduction to JavaFX (4 of 7)

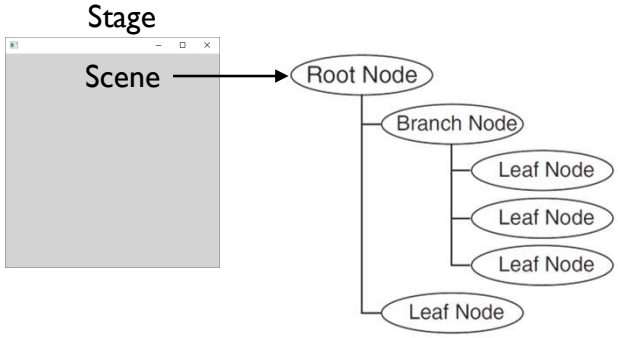
- Scene
  - A collection of GUI objects contained within the window
- GUI objects (controls or actors)
  - Actors make up the scene



9

Introduction to JavaFX (5 of 7)

- GUI objects in a scene
  - Organized as nodes in a *scene graph*
  - A tree-like hierarchical data structure



10

Introduction to JavaFX (6 of 7)

- The scene graph has three types of nodes:
  - Root Node: Only one root node, 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

11

Introduction to JavaFX (7 of 7)

- The `Application` Class
  - The foundation of all JavaFX applications
  - JavaFX applications must extend the `Application` class
  - Has an abstract method named `start()`,
    - the entry point for the application
  - Because the `start()` method is abstract,
    - Your application must override it

- Installing e(fx)clipse for javaFX:
  - Instruction posted on the BB

12

General Layout of a JavaFX Program

- Various `import` statements
- A class that extends the `Application` class
- A launch() method
  - Launch a standalone application
- A start() method
  - The main entry point for all JavaFX applications

13

```
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.
   }
}
```

Necessary `import` statements

14

```
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.
   }
}
```

Necessary `import` statements

A static `main` method that calls the inherited `launch` method

A class that extends the `Application` class

A `start` method that accepts a Stage argument. This method is called by the inherited `launch` method.

15

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 }
```

- Stage class – show() method
  - Attempts to show this Window by setting visibility to true

16

Creating Controls (1 of 2)

- Process for creating a control (GUI object)
  - Import the class for the control from the necessary `javafx` package:

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

  - Instantiate the class:

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


17

Creating Controls (2 of 2)

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

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

  - Instantiate the class:

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

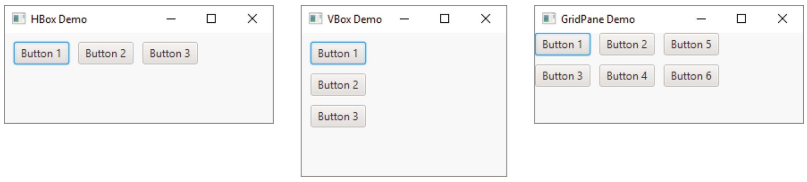

18

Layout Containers (1 of 2)

- Layout containers
  - To arrange the positions of controls on the screen
  - JavaFX provides many layout containers
- Start with
  - `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

19

Layout Containers (2 of 2)



- The layout container classes
  - `javafx.scene.layout` package

20

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);
```


21

Creating a Scene (1 of 2)

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

22

Creating a Scene (2 of 2)

```
// 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);
```


23

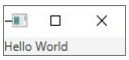
Adding the Scene to the Stage

- Once a `Scene` object created
  - Add it to the application's stage
- Create an instance of the `Stage` class (from the `javafx.stage` package)
- However, do not have to instantiate the `Stage` class
  - Created automatically and passed as an argument to the `start()` method.

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

}
```


24

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

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

   @Override
   public void start(Stage primaryStage)
   {
      Label messageLabel = new Label("Hello World");
      VBox vbox = new VBox(messageLabel);
      Scene scene = new Scene(vbox);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
}
```



Make a Label control
Put the Label in a VBox
Make the VBox the root node in the scene
Set the scene to the stage
Show the stage (display it)

25

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.geometry.Pos;

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

   @Override
   public void start(Stage primaryStage)
   {
      Label messageLabel = new Label("Hello World");
      VBox vbox = new VBox(messageLabel);
      vbox.setAlignment(Pos.CENTER);
      Scene scene = new Scene(vbox , 300, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
}
```

Center-align the VBox

Width Height

26

Displaying Images

- Need two JavaFX classes:
  - The `Image` class, from the `javafx.scene.image` package
    - To load an image into memory
  - The `ImageView` class, from the `javafx.scene.image` package
    - To create a node that displays the image

27

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

   // Create an ImageView object.
   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();
}
```

28

More About HBox and VBox Containers (1 of 2)

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

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

Spacing

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

Spacing

29

More About HBox and VBox Containers (2 of 2)

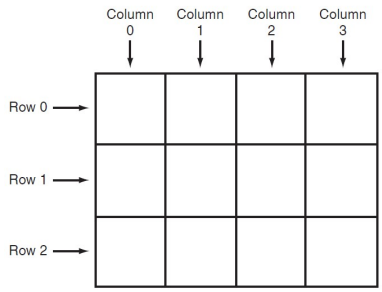
- *Padding*
  - Space that appears around the inside edge of a container
- The HBox and VBox containers have a **`setPadding()`** method.
- An **`Insets` object** as an argument to the `setPadding()` method
  - To specify the number of pixels of padding
- The `Insets` class in the `javafx.geometry` package

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


30

The GridPane Layout Container (1 of 5)

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



31

The GridPane Layout Container (2 of 5)

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

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

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

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


32

The GridPane Layout Container (3 of 5)

```
// 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
```


33

The GridPane Layout Container (4 of 5)

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

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


34

The GridPane Layout Container (5 of 5)

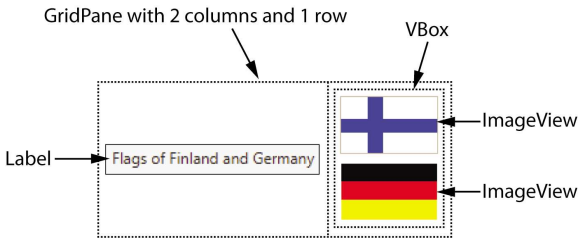
- The GridPane container
  - setPadding() method to set the padding around the container's inside edge

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


35

Using Multiple Layout Containers

- To get the particular screen layout
  - Have to nest layout containers



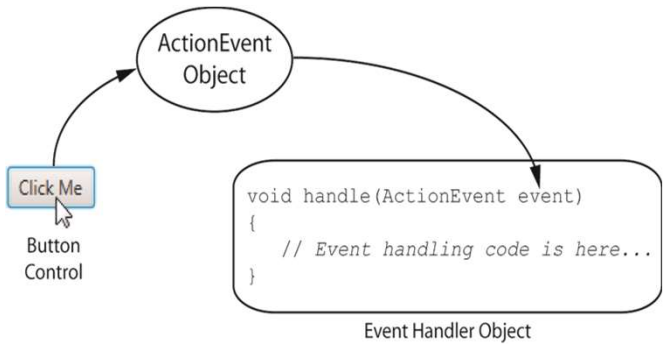
36

Handling Events (1 of 2)

- An *event* is an action
  - E.g., the clicking of a button
- Event object created by the control
  - When an event takes place
  - Contains information about the event
  - Connected to one or more event listeners

37

Handling Events (2 of 2)



38

Event Objects

- Instances of the `Event` class (from the `javafx.event` package), or one of its subclasses
- E.g., a `Button` click generates an `ActionEvent` object
  - `ActionEvent` is a subclass of the `Event` class

39

Event Handlers (1 of 2)

- Event handlers are objects
  - Event handler classes
    - Implement the `EventHandler` interface
  - from the `javafx.event` package
- The `EventHandler` interface
  - Specifies a `handle() method`
    - That has a parameter of the `Event` class (or one of its subclasses).

40

Event Handlers (2 of 2)

```
class ButtonClickHandler implements EventHandler<ActionEvent>
{
    @Override
    void handle(ActionEvent event)
    {
        // Write event handling code here.
    }
}
```


41

Registering an Event Handler

- The process of connecting an event handler object to a control
- E.g., `Button` controls have a method named `setOnAction` to register an event handler:

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

- When the user clicks the button,
  - the event handler object's `handle` method executed

42

Event Handling Example

- An application that initially displays this screen:

- When the user clicks the button, the screen changes to:

43

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.control.Label;
import javafx.scene.control.Button;
import javafx.geometry.Pos;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;

public class EventDemo extends Application
{
   private Label myLabel;

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

Continued…

44

```
   @Override
   public void start(Stage primaryStage)
   {
      // Create a Label and a Button.
      Label myLabel = new Label("Click the button to see a message.");
      Button myButton = new Button("Click Me");

      // Register an event handler.
      myButton.setOnAction(new ButtonClickHandler());

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

      // Create a Scene and display it.
      Scene scene = new Scene(vbox, 300, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }

   class ButtonClickHandler implements EventHandler<ActionEvent>
   {
      @Override
      public void handle(ActionEvent event)
      {
         myLabel.setText("Thanks for clicking the button!");
      }
   }
}
```


45

Reading Input with `TextField` Controls (1 of 2)

- At runtime, the user can type text into a `TextField` control
- Can retrieve the text that the user entered
- The `TextField` class in the `javafx.scene.control` package
- To create an empty `TextField`:

```
TextField myTextField = new TextField();
```


46

Reading Input with `TextField` Controls (2 of 2)

- Call the control's `getText` method
  - To retrieve the text that the user has typed into a `TextField` control
  - The method returns the value entered, as a `String`
- Example:

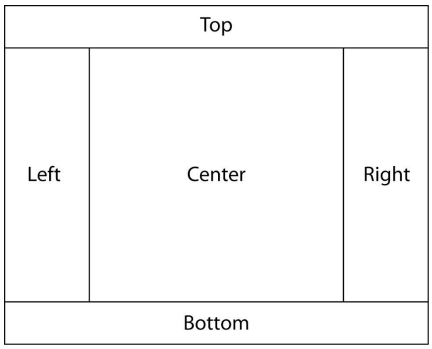
```
String input;
input = myTextField.getText();
```

- E.g., kiloConverter.java

47

The `BorderPane` Layout Container (1 of 4)

- The `BorderPane` layout container manages controls in five regions:



48

The BorderPane Layout Container (2 of 4)

- You do not usually put controls directly into a BorderPane region
- Typically, put controls into another type of layout container
- Then put that container into one of the BorderPane regions

49

The BorderPane Layout Container (3 of 4)

- The BorderPane class
  - in the javafx.scene.layout package
- Summary of constructors:

Constructor	Description
BorderPane()	The no-arg constructor creates an empty BorderPane container.
BorderPane(*center*)	This constructor accepts one argument. The node that is passed as the argument is placed in the BorderPane's center region.
BorderPane(*center*, *top*, *right*, *bottom*, *left*)	This constructor accepts five nodes as arguments: one to place in each region.

50

The BorderPane Layout Container (4 of 4)

- The BorderPane class provides the following methods to add controls to specific regions:
  - setCenter
  - setTop
  - setBottom
  - setLeft
  - setRight
- Example: BorderPaneDemo1.java

51

Other Controls

- RadioButton Controls
- CheckBox Controls
- ListView Controls
- ComboBox Controls
- Slider Controls
- TextArea Controls
- Menus
- The FileChooser Class

52