SWEG3101

Object Oriented Programming

Department of Software Engineering
AASTU

Chapter Seven

Introduction to GUI in Java

Objectives

- To become familiar with common user-interface components, such as radio buttons, check boxes, and menus
- To use layout managers to arrange user-interface components in a container
- To handle events generated by user interactions with GUIs

Outline

- Introduction
- GUI Components
- Layout Management
- Event Handling in Java

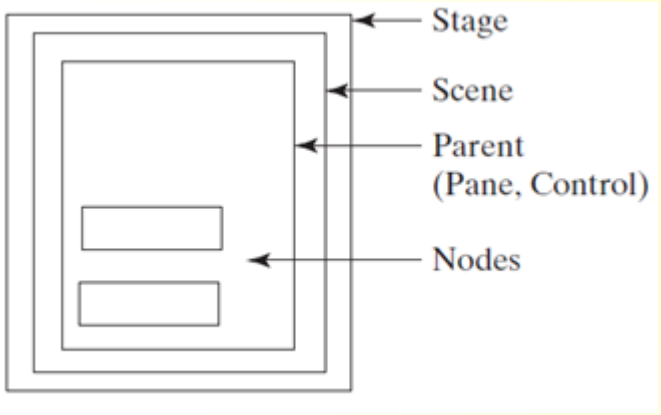
Introduction

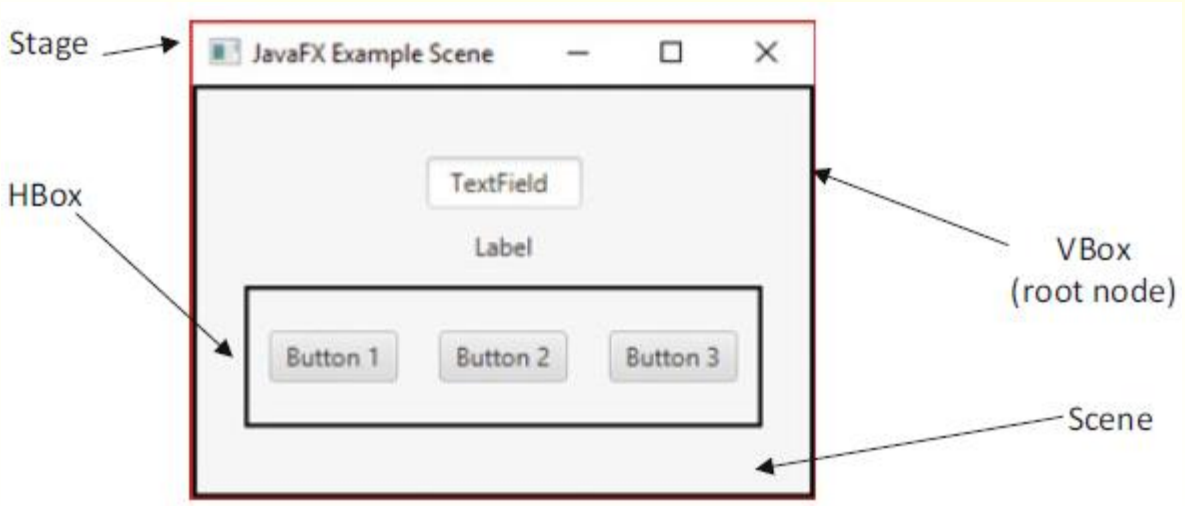
- A graphical user interface (GUI) presents a user-friendly mechanism for interacting with an application.
- GUI provides result to end user in response to raised events.
- There are three main sets of visual components and containers for user interface design in JAVA:
  - AWT (Abstract Windows Toolkit) - in package **java.awt**
  - Swing - in package **javax.swing**
  - JavaFX(part of Java 8, 2014)
- When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT).
- Swing and AWT are replaced by the JavaFX platform
  - Support both desktop and web application
  - provides a multi-touch support for touch enabled devices
  - has a built-in 2D, 3D, animation support

Structure of JavaFX Application

- Several classes need to be imported when you write a JavaFX application.
  - The Application class - provides the basic functionality of the program
    - All JavaFX applications extend from the Application class
    - manages life cycle of JavaFX
  - The Stage class(defined in **javafx.stage.Stage**)
    - The top-level window(container) in which the application runs
  - The Scene class(defined in **javafx.scene.Scene**)
    - Holds content inside a window
    - The contents of the stage - the graphic itself
  - Nodes - the items that make up the scene
    - Is the base class for scene graph nodes
    - Defined in javafx.scene.Node
- JavaFX programs begin execution with a main() method
  - launch() method – launches JavaFX applications
  - The main method is only needed for the IDE with limited JavaFX support. Not needed for running from the command line.

Relationship of Stage and Scene





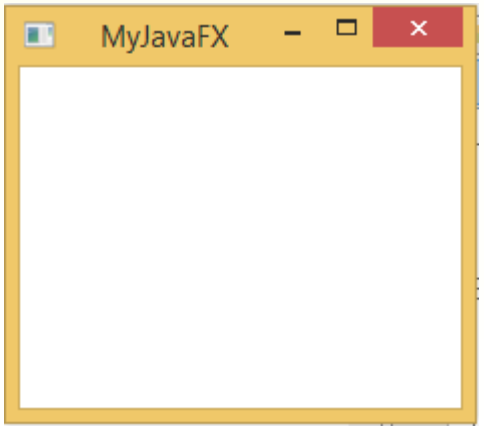
Panes and UI Controls

- A UI control refers to a label, button, check box, radio button, text field, text area, etc.
- A node is a visual component such as a shape, an image view, a UI control, or a pane
  - Nodes can be containers - components that hold other nodes
  - Containers can contain other containers
    - Root node - top level node
    - Parent(containing) and children(contained)
- A pane is a container class to which you can add one or more user-interface elements.
  - Defined in *javafx.scene.layout*
- A shape refers to a line, circle, ellipse, rectangle, arc, polygon, polyline, etc.

Example 1: Display Stage

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

public class DisplayStage extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
 //set properties
 primaryStage.setTitle("MyJavaFX"); // Set the stage title
 primaryStage.setResizable(true);  //If false, prevent the user from resizing
 primaryStage.setX(210);
 primaryStage.setY(0);
 primaryStage.setMinHeight(250);
 primaryStage.setMinWidth(300);
 primaryStage.setMaxHeight(350);
primaryStage.setMaxWidth(400);
 primaryStage.show(); // Display the stage
 }

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

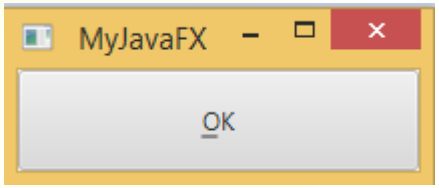
Output



Example 2: Adding Button

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

public class AddButton extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a button and place it in the scene
    Button btOK = new Button();
    btnok.setMnemonicParsing(true);
    btnok.setText("_OK");
    Scene scene = new Scene(btOK, 200, 250);
    primaryStage.setTitle("MyJavaFX"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

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

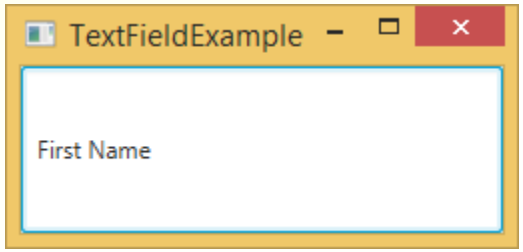
Output



Example 3: Adding TextField

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

public class AddButton extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a TextField and place it in the scene
    TextField tfName = new TextField (“First Name”);
    Scene scene = new Scene(tfName, 200, 250);
    primaryStage.setTitle(“TextFieldExample"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

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

Output



Example 5: Adding TextArea and ComboBox

```
TextArea taNote = new TextArea("This is a text area");
taNote.setPrefColumnCount(20);
taNote.setPrefRowCount(5);
taNote.setWrapText(true);
taNote.setStyle("-fx-text-fill: red");
taNote.setFont(Font.font("Times", 20));
```

- **ComboBox** is defined as a generic class. The generic type **T** specifies the element type for the elements stored in a combo box.
- **Example**:

```
ComboBox<String> cbo = new ComboBox<>();
cbo.getItems().addAll("Item 1", "Item 2",
"Item 3", "Item 4");
cbo.setStyle("-fx-color: red");
cbo.setValue("Item 1");
```

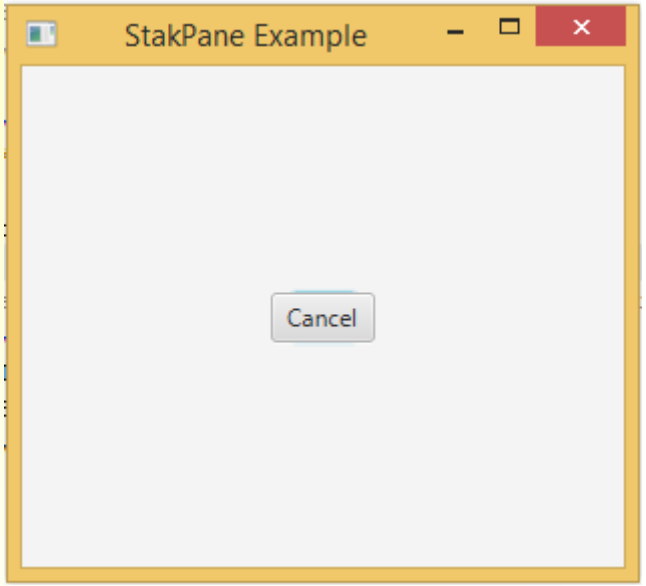
Layout Panes

JavaFX provides many types of panes for organizing nodes in a container.

Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

Example: StackPane

Output

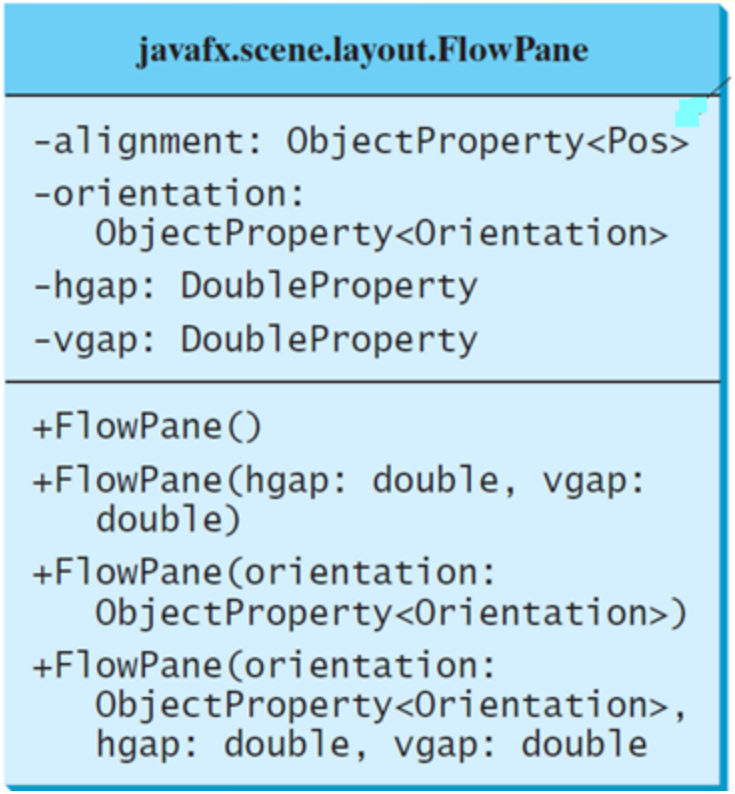



```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class StackPaneExample extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button btnOK = new Button();
        btnOK.setText("OK");
        Button btnCancel = new Button();
        btnCancel.setText("Cancel");
        StackPane root = new StackPane(btnOK, btnCancel);
        //root.getChildren().addAll(btnOK, btnCancel); //get a list of all the child nodes then add
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("StakPane Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

FlowPane

- Arranges the nodes in the pane horizontally from left to right or vertically from top to bottom in the order in which they were added. When one row or one column is filled, a new row or column is started.
- You can specify the way the nodes are placed horizontally or vertically using one of two constants:
  - **Orientation.HORIZONTAL** or
  - **Orientation.VERTICAL**.
- You can also specify the gap between the nodes in pixels

FlowPane Class Diagram



The overall alignment of the content in this pane (default: `Pos.LEFT`).

The orientation in this pane (default: `Orientation.HORIZONTAL`).

The horizontal gap between the nodes (default: 0).

The vertical gap between the nodes (default: 0).

Creates a default `FlowPane`.

Creates a `FlowPane` with a specified horizontal and vertical gap.

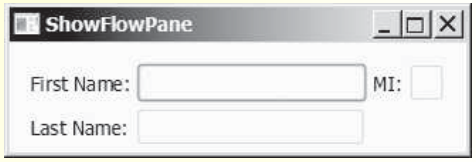
Creates a `FlowPane` with a specified orientation.

Creates a `FlowPane` with a specified orientation, horizontal gap and vertical gap.

Example: FlowPane

```
import javafx.application.Application;
import javafx.geometry.Insets;   //Insets specifies the size of the border of a pane(T,R,B,L)
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.control.Label;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;
public class ShowFlowPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        FlowPane pane = new FlowPane();
        pane.setPadding(new Insets(11, 12, 13, 14));
        pane.setAlignment(Pos.CENTER_LEFT);//center vertically and right horizontally
        pane.setHgap(5);
        pane.setVgap(5);
        pane.getChildren().addAll(new Label("First Name:"), new TextField(), new Label("MI:"));
        TextField txtMi = new TextField();
        txtMi.setPrefColumnCount(1);    //Sets the preferred size of the text field in columns
        pane.getChildren().addAll(txtMi, new Label("Last Name:"),new TextField());
        Scene scene = new Scene(pane, 300, 150);
        primaryStage.setTitle("ShowFlowPane");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

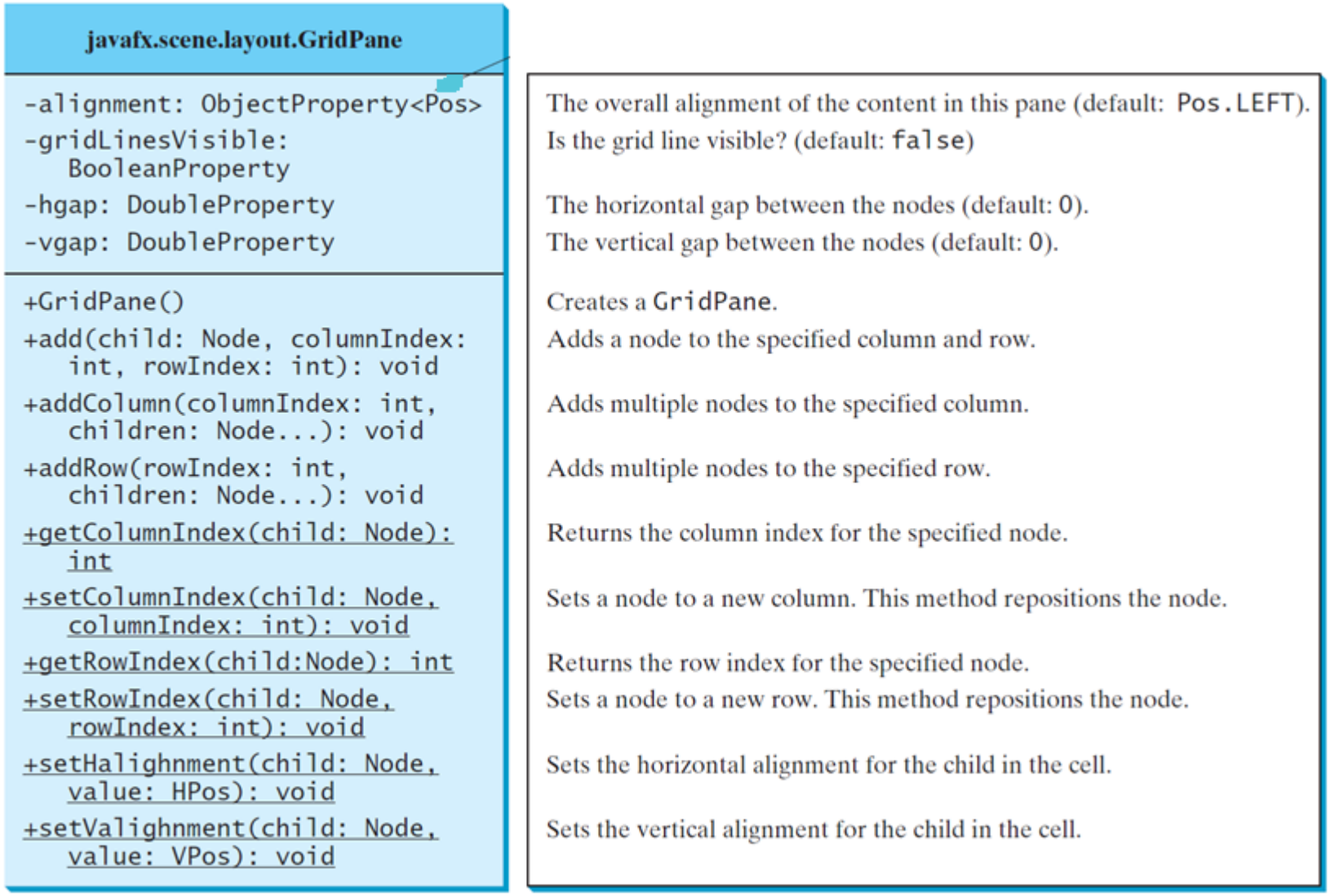
Output



GridPane

- A **GridPane** arranges nodes in a grid (matrix) formation.
- The nodes are placed in the specified column and row indices.

GridPane Class Diagram



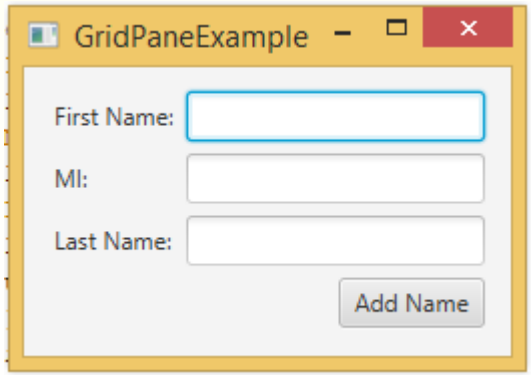
Example: GridPane

```
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class ShowGridPane extends Application {

    @Override
    public void start(Stage primaryStage) {
        // Create a pane and set its properties
        GridPane pane = new GridPane();
        pane.setAlignment(Pos.CENTER);   //place the nodes in the center of the pane
        pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
        pane.setHgap(5.5);
        pane.setVgap(5.5);
```

Example: GridPane(cont'd)

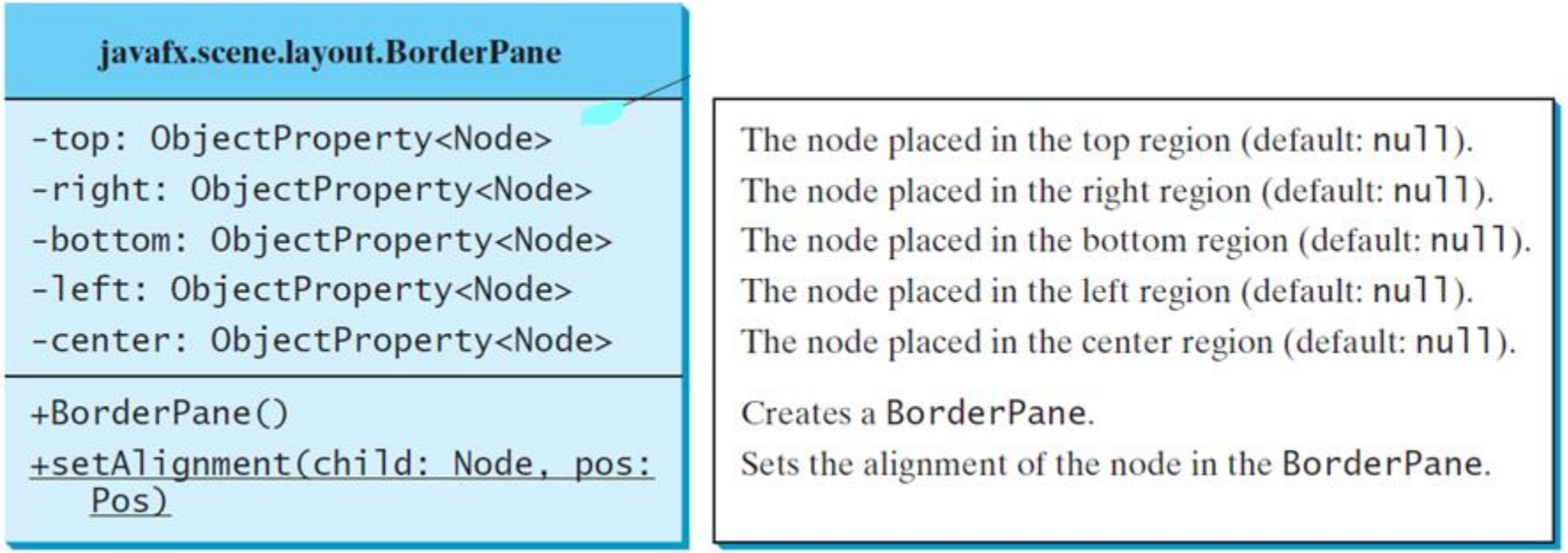
```
        // Place nodes in the pane
        pane.add(new Label("First Name:"), 0, 0);
        pane.add(new TextField(), 1, 0);
        pane.add(new Label("MI:"), 0, 1);
        pane.add(new TextField(), 1, 1);
        pane.add(new Label("Last Name:"), 0, 2);
        pane.add(new TextField(), 1, 2);
        Button btAdd = new Button("Add Name");
        pane.add(btAdd, 1, 3);
        //Set the horizontal alignment for the node
        GridPane.setHalignment(btAdd, HPos.RIGHT);

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane);
        primaryStage.setTitle("GridPaneExample");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Output



BorderPane

- A **BorderPane** can place nodes in five regions: top, bottom, left, right, and center.
  - Methods: **setTop(node)**, **setBottom(node)**, **setLeft(node)**, **setRight(node)**, and **setCenter(node)**.
- Class Diagram:



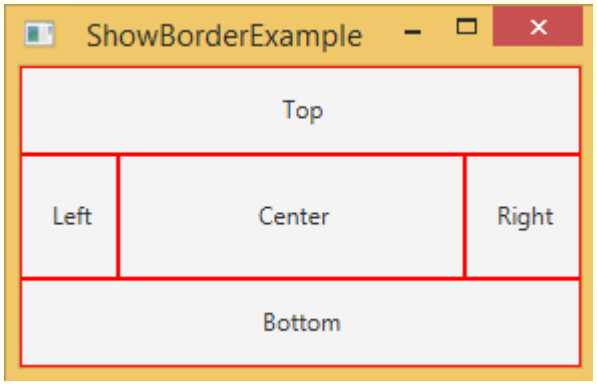
Example: BorderPane

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class ShowBorderPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        BorderPane pane = new BorderPane();
        //Place nodes in the pane
        pane.setTop(new CustomPane("Top"));
        pane.setRight(new CustomPane("Right"));
        pane.setBottom(new CustomPane("Bottom"));
        pane.setLeft(new CustomPane("Left"));
        pane.setCenter(new CustomPane("Center"));

        Scene scene = new Scene(pane, 280, 150);
        primaryStage.setTitle("ShowBorderExample");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Example: BorderPane(cont'd)

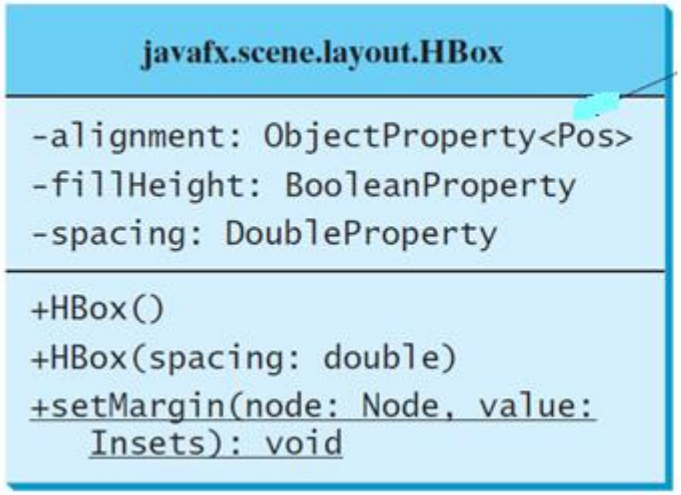
```
// Define a custom pane to hold a label in the center of the pane
class CustomPane extends StackPane {
    public CustomPane(String title) {
        getChildren().add(new Label(title));
        setStyle("-fx-border-color: red");
        setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
    }
}
```

Output



HBox and VBox

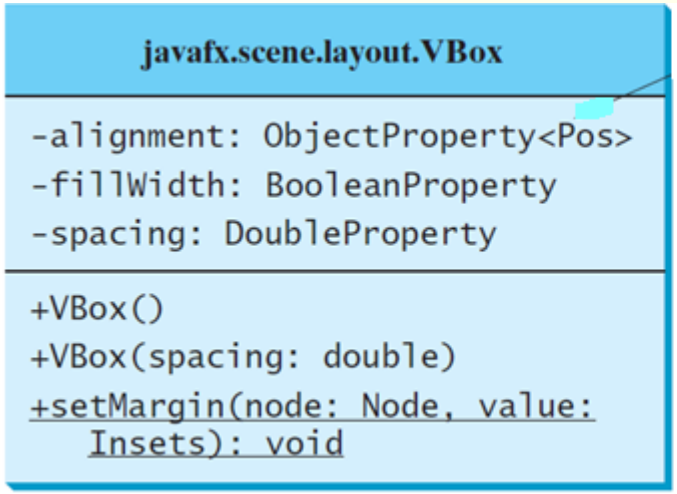
- **HBox** lays out its children in a single horizontal row.



The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).
Is resizable children fill the full height of the box (default: `true`).
The horizontal gap between two nodes (default: 0).

Creates a default HBox.
Creates an HBox with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

- **VBox** lays out its children in a single vertical column.



The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).
Is resizable children fill the full width of the box (default: `true`).
The vertical gap between two nodes (default: 0).

Creates a default VBox.
Creates a VBox with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

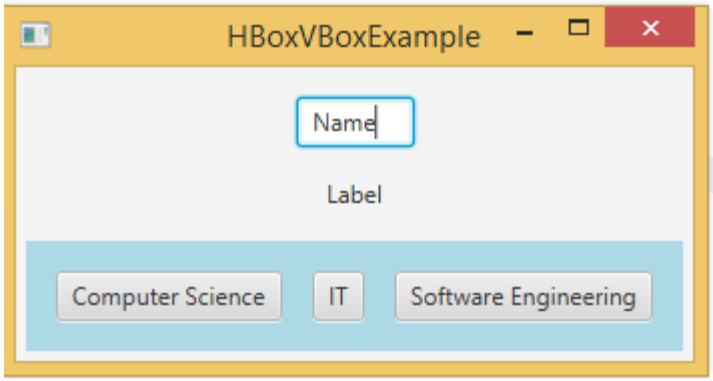
Example

```
public class ShowHBoxVBox extends Application {

    @Override
    public void start(Stage primaryStage) {
        VBox vBox = new VBox(15);
        vBox.setAlignment(Pos.CENTER);
        vBox.setPadding(new Insets(15, 5, 5, 5));
        TextField txtName = new TextField("Name");
        txtName.setPrefColumnCount(4);
        vBox.getChildren().addAll(txtName);
        vBox.getChildren().add(new Label("Label"));
        vBox.getChildren().add(getHBox());

        Scene scene = new Scene(vBox);
        primaryStage.setTitle("HBoxVBoxExample");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

Example(cont'd)

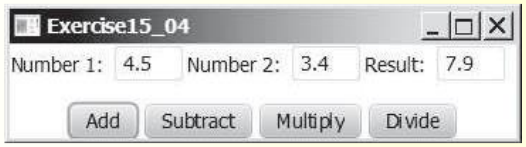
```
private HBox getHBox() {
        HBox hBox = new HBox(15);
        hBox.setPadding(new Insets(15, 15, 15, 15));
        hBox.setStyle("-fx-background-color: lightblue");
        hBox.getChildren().add(new Button("Computer Science"));
        hBox.getChildren().add(new Button("IT"));
        hBox.getChildren().add(new Button("Software Engineering"));

        return hBox;
 }
}
```

Output



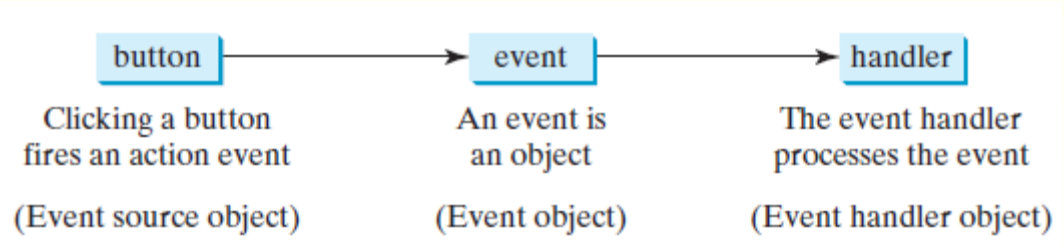
Exercise

- Create the following GUI.



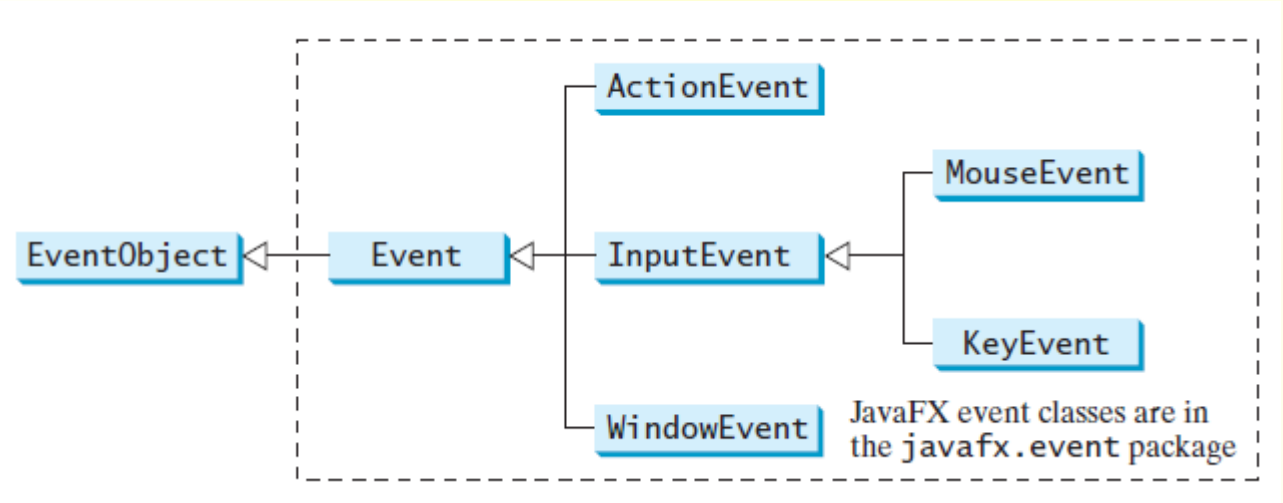
Event Handling

- Event and Event Source
  - An event can be defined as a signal to the program that something has happened.
  - Can be:
    - external such as mouse movements, button clicks, and keystrokes
    - internal program activities such as a timer.
  - The component that creates an event and fires it is called the event source object *or source component.*
    - For example, a button is the source object for a button-clicking action event.
  - Event handlers: the object that listens for events and handles them when they occur.



  - javafx.event.Event –is the root class the JavaFX event classes

Event Handling (cont'd)



Hierarchical Relationships of some event classes

Event Handling (cont'd)

User Action	*Source Object*	*Event Type Fired*	*Event Registration Method*
Click a button	`Button`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Press Enter in a text field	`TextField`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`RadioButton`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`CheckBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Select a new item	`ComboBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Mouse pressed	`Node, Scene`	`MouseEvent`	`setOnMousePressed(EventHandler<MouseEvent>)`
Mouse released			`setOnMouseReleased(EventHandler<MouseEvent>)`
Mouse clicked			`setOnMouseClicked(EventHandler<MouseEvent>)`
Mouse entered			`setOnMouseEntered(EventHandler<MouseEvent>)`
Mouse exited			`setOnMouseExited(EventHandler<MouseEvent>)`
Mouse moved			`setOnMouseMoved(EventHandler<MouseEvent>)`
Mouse dragged			`setOnMouseDragged(EventHandler<MouseEvent>)`
Key pressed	`Node, Scene`	`KeyEvent`	`setOnKeyPressed(EventHandler<KeyEvent>)`
Key released			`setOnKeyReleased(EventHandler<KeyEvent>)`
Key typed			`setOnKeyTyped(EventHandler<KeyEvent>)`

User Action, Source Type, Event Type, and Handler Interface

Registering Handlers and Handling Events

- Java uses a delegation-based model for event handling:
  - a source object fires an event, and an object interested in the event handles it(event handler or an event listener)
- For an object to be a handler for an event on a source object, two things are needed:
  - **The handler object must be an instance of the corresponding event-handler interface** to ensure that the handler has the correct method for processing the event.
    - For example, the handler interface for **ActionEvent** is **EventHandler<ActionEvent>**; each handler for **ActionEvent** should implement the **handle(ActionEvent e)** method for processing an **ActionEvent**.
  - **The handler object must be registered by the source object.** Registration methods depend on the event type. For **ActionEvent, the method is setOnAction.**

Example 1

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import javafx.geometry.Pos;
public class HandleEvent extends Application {

    @Override
    public void start(Stage primaryStage) {
        // Create a pane and set its properties
        HBox pane = new HBox(10);
        pane.setAlignment(Pos.CENTER);
        Button btOK = new Button("OK");
        Button btCancel = new Button("Cancel");
        OKHandlerClass handler1 = new OKHandlerClass();   //create handler
        btOK.setOnAction(handler1);       //register handler
        CancelHandlerClass handler2 = new CancelHandlerClass();
        btCancel.setOnAction(handler2);
        pane.getChildren().addAll(btOK, btCancel);
```

Example 1(cont'd)

```
        // Create a scene and place it in the stage
        Scene scene = new Scene(pane, 250,50);
        primaryStage.setTitle("HandleEvent"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
    class OKHandlerClass implements EventHandler<ActionEvent> {
        @Override
        public void handle(ActionEvent e) {
            System.out.println("OK button clicked");
        }
    }

    class CancelHandlerClass implements EventHandler<ActionEvent> {
        @Override
        public void handle(ActionEvent e) {
            System.out.println("Cancel button clicked");
        }
    }
}
```

Output

Example 2: Add two numbers

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;
import javafx.stage.StageStyle;
public class AddNumber extends Application {
    TextField txffirstno = new TextField();
    TextField txfsecondno = new TextField();
    TextField txfresult = new TextField();
    Label lblplus = new Label("+");
    Button btnequal = new Button("=");
    Button btnclear = new Button("Clear");
```

Example 2 (cont'd)

```
@Override
public void start(Stage primaryStage) {
    FlowPane pane = new FlowPane();
    pane.setPadding(new Insets(11, 12, 13, 14));
    pane.setAlignment(Pos.CENTER_LEFT);//center vertically and right horizontally
    pane.setHgap(5);
    pane.setVgap(5);
    pane.getChildren().addAll(txffirstno, lblplus, txfsecondno, btnequal, txfresult, btnclear);
    AddHandlerClass addhandler = new AddHandlerClass();
    btnequal.setOnAction(addhandler);
    btnclear.setOnAction(addhandler);
    Scene scene = new Scene(pane, 300, 120);
    primaryStage.setTitle("Simple Calc");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Example 2 (cont'd)

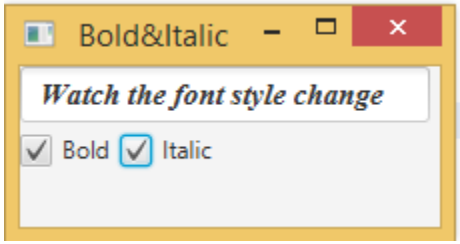
```
class AddHandlerClass implements EventHandler<ActionEvent> {
    public void handle(ActionEvent e) {
        double fno, sno, sum;
        try{
            fno =  Double.parseDouble(txffirstno.getText());
            sno =  Double.parseDouble(txfsecondno.getText());
        }
        catch(NumberFormatException ex){
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.initStyle(StageStyle.UTILITY);
            alert.setTitle("Empty Field");
            alert.setHeaderText(null);
            alert.setContentText("Plaease Ener only Number");
            alert.showAndWait();
            return;
        }
        if(e.getSource()==btnequal){
            sum = fno + sno;
            txfresult.setText(""+sum);
        }
        else{
            txffirstno.setText("");
            txfsecondno.setText("");
            txfresult.setText("");
        }
    }   }
```

Example 3: CheckBox

```
//packages here
public class CheckBoxEg extends Application {
    TextField txfresult = new TextField( "Watch the font style change");
    CheckBox chkbold = new CheckBox( "Bold" );
    CheckBox chkitalic = new CheckBox( "Italic" );
    @Override
    public void start(Stage primaryStage) {
        FlowPane pane = new FlowPane();
        pane.getChildren().addAll(txfresult, chkbold, chkitalic);
        CheckBoxHandler handler = new CheckBoxHandler();
        chkbold.setOnAction(handler);
        chkitalic.setOnAction(handler);
        Scene scene = new Scene(pane,150,180);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

Example 3: CheckBox(cont'd)

```
    // private inner class for ItemListener event handling
    class CheckBoxHandler implements EventHandler<ActionEvent> {
        @Override
        public void handle(ActionEvent e) {
            FontWeight fontweight=FontWeight.NORMAL; // controls bold font style
            FontPosture fontposture=FontPosture.REGULAR; // controls italic font style
            // process bold checkbox events
            if(e.getSource() == chkbold ){
                fontweight = (chkbold.isSelected())? FontWeight.BOLD : FontWeight.NORMAL;
                fontposture = (chkitalic.isSelected())? FontPosture.ITALIC : FontPosture.REGULAR;
            }
            if(e.getSource() == chkitalic ){
                fontweight = (chkbold.isSelected())? FontWeight.BOLD : FontWeight.NORMAL;
                fontposture = (chkitalic.isSelected())? FontPosture.ITALIC: FontPosture.REGULAR;
            }
            txfresult.setFont(Font.font("Serif", fontweight, fontposture,15));
        }
    }
}
```



Activity

- Create the following GUI and
- handle events