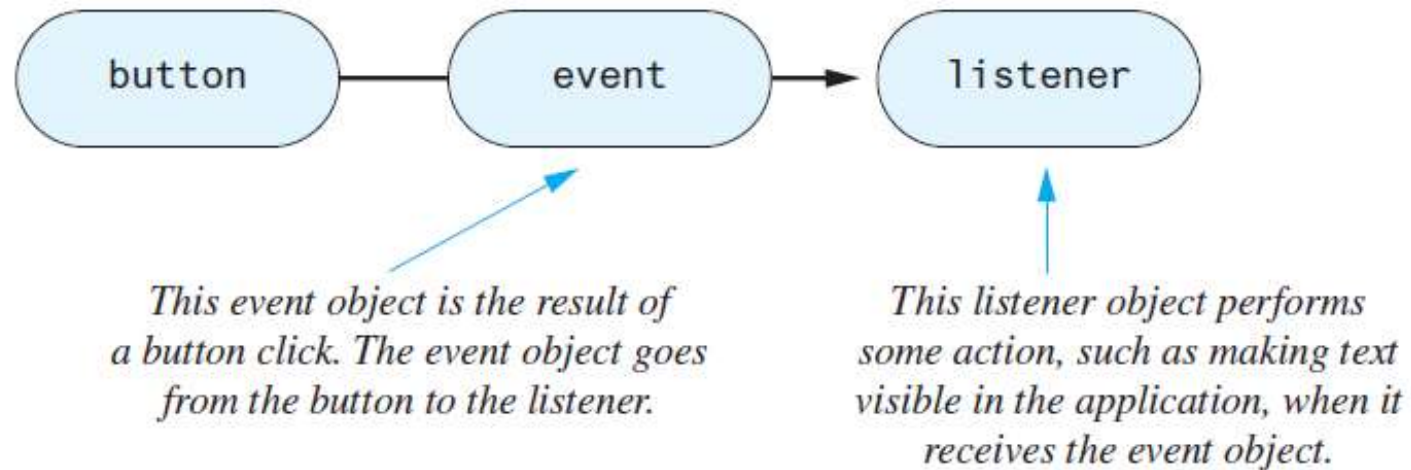




8.5 Graphics Supplement: Outline

- Event-Driven Programming





Event Firing and Event Listeners

- An **event object** is generated when something happens, like a button is clicked, the mouse is moved, the mouse is over a component, the mouse leaves a component, etc.
- You specify what happens when the event occurs by creating **an event handler or listener**



An Event Handler for a Button Click

FIGURE 8.7 Buttons and an Event Handler

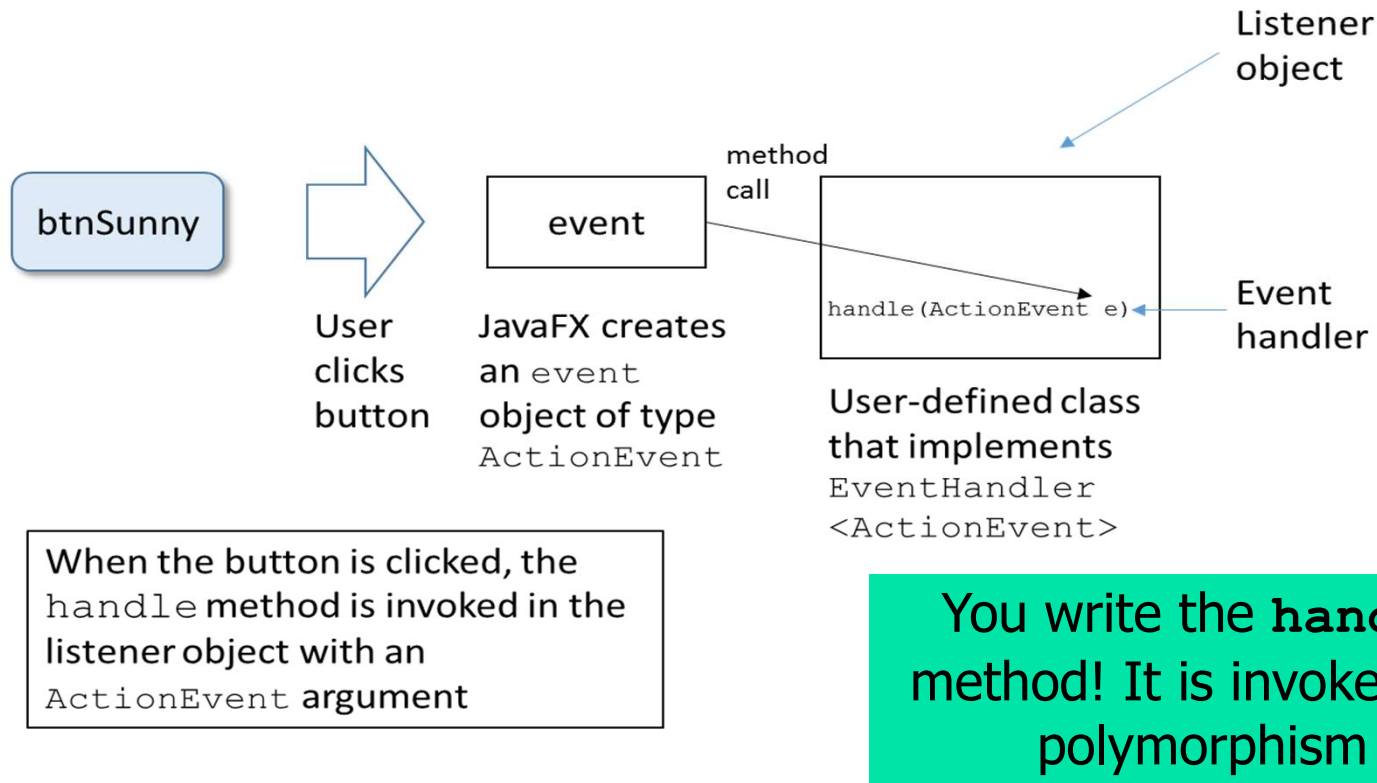
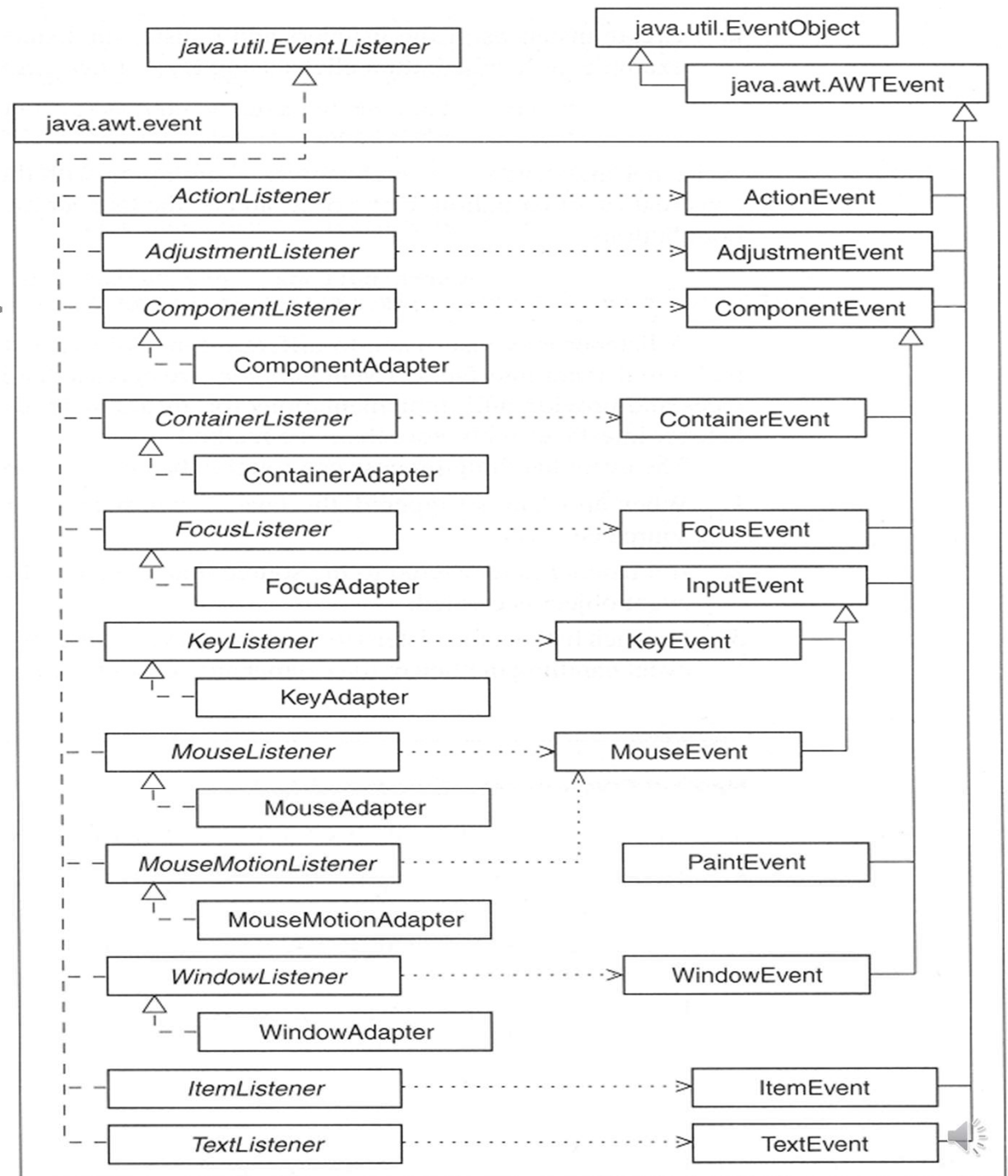




Figure 6.9
The event classes
and listeners



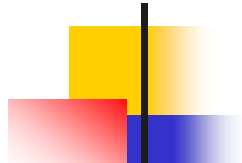


Steps in Handling Events in Java programs

- 1) Determine the types of events to be handled and their associated listener interfaces. For example, to handle button clicks, the event class is the `ActionEvent` and the associated listener interface is `ActionListener`.
- 2) Declare listener classes that implement the listener interfaces and all the methods of the interfaces. For example, to handle button clicks, the listener class must implement the interface `ActionListener`.

```
class MyButtonHandler implements ActionListener {  
    public void actionPerformed(ActionEvent event) {  
        //... handle button click  
    }  
}
```





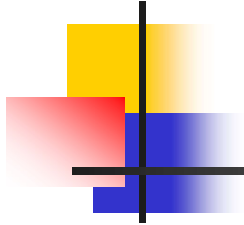
- 3) Create instances of the components, which are the sources of the events. For example,

```
Button button1 = new Button("One");
```

- 4) Create instances of the listeners and register the listeners to their sources. For example, to handle button clicks using MyButtonHandler use

```
MyButtonHandler handler = new MyButtonHandler();  
button1.addActionListener(handler);
```





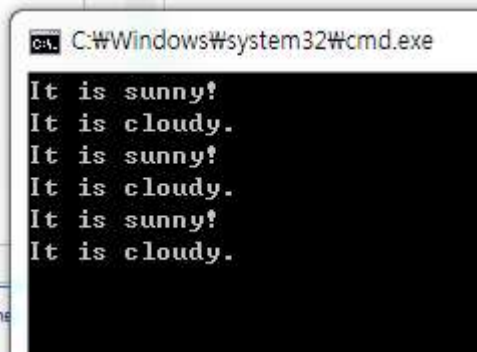
- Three alternatives of **Listener**
 - 1) Separate class to handle events
 - 2) Same class as the window
 - 3) Anonymous inner class



1) Event Handling with a Separate Class

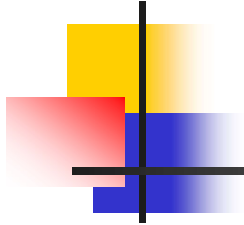
- a class whose purpose is specifically to handle a particular event, in this case, to react to a button click

The program outputs “It is sunny!” to the console when the “Sunny” button is clicked, and “It is cloudy.” When the “Cloudy” button is clicked.

A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt displays the following output:

```
It is sunny!  
It is cloudy.  
It is sunny!  
It is cloudy.  
It is sunny!  
It is cloudy.
```





- event handling demo 1
 - listing 8.20, **ButtonDemo1**
- event handler
 - listing 8.21, **HandleButtonClick**



listing 8.20, **ButtonDemo1**

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

/**
Simple demonstration of programming buttons in a JavaFX application.
This version outputs a message when clicked.
*/
public class ButtonDemo1 extends Application
{
    public static void main(String[] args)
    {
        launch(args);
    }
}
```



@Override

public void start(Stage primaryStage) throws Exception

{

VBox root = new VBox();

Button btnSunny;

Button btnCloudy;

btnSunny = new Button("Sunny");

btnCloudy = new Button("Cloudy");

// Create an event object to handle the button click.

// The "handle" method in HandleButtonClick will be

// invoked when the button is clicked.

HandleButtonClick clickEvent = new HandleButtonClick();

btnSunny.setOnAction(clickEvent);

// We can also create the HandleButtonClick object without

// a named reference by creating it inside the call to setOnAction

btnCloudy.setOnAction(new HandleButtonClick("It is cloudy."));

root.getChildren().add(btnSunny);

root.getChildren().add(btnCloudy);

Scene scene = new Scene(root, 300, 100);

primaryStage.setTitle("Button Event Handling Demo");

primaryStage.setScene(scene);

primaryStage.show();

}

}





Register an EventHandler

setOnAction

```
public final void setOnAction(EventHandler<ActionEvent> value)
```

Sets the value of the property onAction.

Property description:

The button's action, which is invoked whenever the button is fired. This may be due to the user clicking on the button with the mouse, or by a touch event, or by a key press, or if the developer programmatically invokes the `fire()` method.

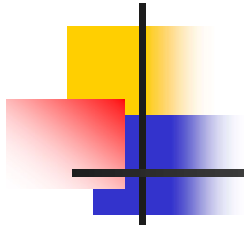


listing 8.21, **HandleButtonClick**

```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

/**
This class handles a button click and outputs a message.
The handle method is invoked when the button is clicked.
*/
public class HandleButtonClick implements EventHandler<ActionEvent>
{
    private String message;
    public HandleButtonClick()
    {
        message = "It is sunny!";
    }
    public HandleButtonClick(String customMessage)
    {
        message = customMessage;
    }
    @Override
    public void handle(ActionEvent event)
    {
        System.out.println(message);
    }
}
```





javafx.event

Interface `EventHandler``<T extends Event>`

Type Parameters:

T - the event class this handler can handle

All Superinterfaces:

`EventListener`

All Known Implementing Classes:

`WeakEventHandler`

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.



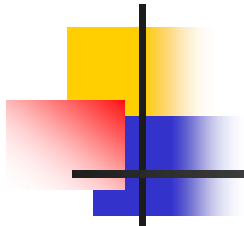


The program outputs "It is sunny!" to the console when the "Sunny" button is clicked, and "It is cloudy." When the "Cloudy" button is clicked.

A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt displays the following output:

```
It is sunny!  
It is cloudy.  
It is sunny!  
It is cloudy.  
It is sunny!  
It is cloudy.
```





Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method and Description	
void	handle (T event) Invoked when a specific event of the type for which this handler is registered happens.	

Method Detail

handle

void handle(T event)

Invoked when a specific event of the type for which this handler is registered happens.

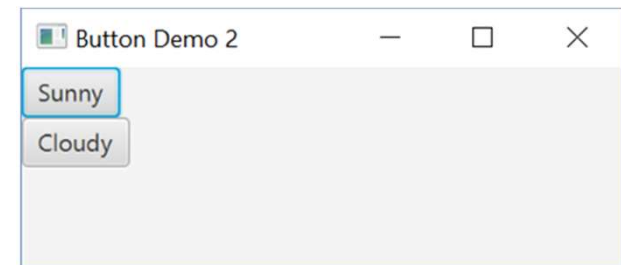
Parameters:

event - the event which occurred

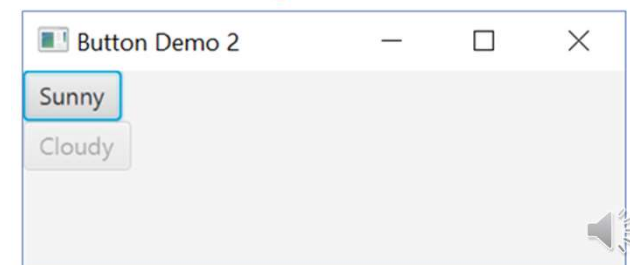


2) Event Handling in the Main GUI Class

- We can put **the event handler** in the same class as the GUI; this allows the event handler to have easy access to any GUI class variables
 - Add the handler and implement the ActionListener
- **event handling demo 2**
 - listing 8.22, **ButtonDemo2**



After “Sunny” is clicked



listing 8.22, **ButtonDemo2**

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

/**
Demonstration of event handling within the ButtonDemo2 class.
*/
public class ButtonDemo2 extends Application implements
EventHandler<ActionEvent>
{
    private Button btnSunny;
    private Button btnCloudy;

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



```
@Override
public void handle(ActionEvent event)
{
    // This method can access the member variables
    // which reference the other GUI controls
    if (event.getSource() instanceof Button)
    {
        Button btnClicked = (Button) event.getSource();
        if (btnClicked.getText().equals("Sunny"))
        {
            // Disable the cloudy button if sunny clicked
            btnCloudy.setDisable(true);
        }
        else if (btnClicked.getText().equals("Cloudy"))
        {
            // Disable the sunny button if cloudy clicked
            btnSunny.setDisable(true);
        }
    }
}
```



@Override

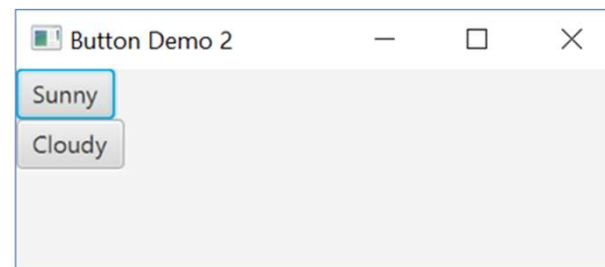
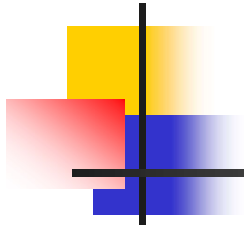
```
public void start(Stage primaryStage) throws Exception
{
    VBox root = new VBox();
    btnSunny = new Button("Sunny");
    btnCloudy = new Button("Cloudy");

    btnSunny.setOnAction(this);
    btnCloudy.setOnAction(this);

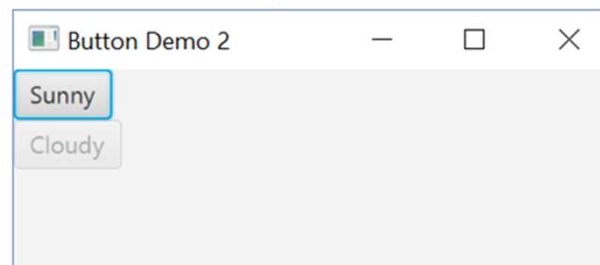
    root.getChildren().add(btnSunny);
    root.getChildren().add(btnCloudy);

    Scene scene = new Scene(root, 300, 100);
    primaryStage.setTitle("Button Demo 2");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```





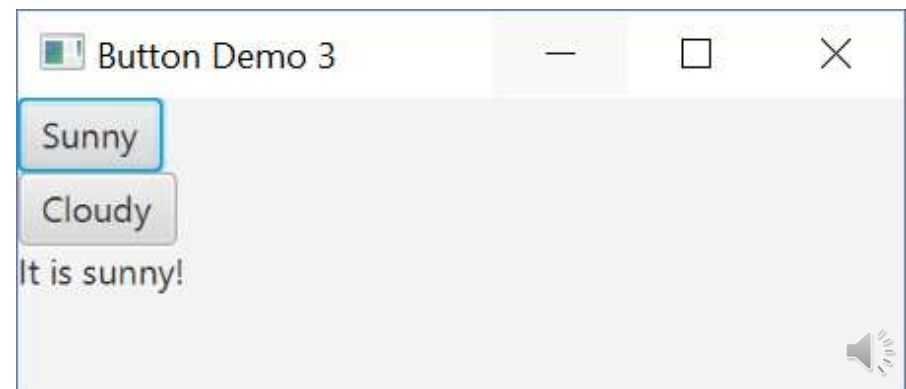
After "Sunny" is clicked



3) Event Handling in an Anonymous Inner Class

- We can create a new event handler by creating an **anonymous inner class** for each event we want to handle
 - This is a class with no name that we declare and instantiate at the same time
- event handling demo 3,
 - listing 8.23, **ButtonDemo3**

(After clicking “Sunny”)



listing 8.23, **ButtonDemo3**

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

/**
Event handling with an anonymous inner class.
*/
public class ButtonDemo3 extends Application
{
    public static void main(String[] args)
    {
        launch(args);
    }
}
```



```
@Override
public void start(Stage primaryStage) throws Exception
{
    VBox root = new VBox();
    Button btnSunny;
    Button btnCloudy;
    Label lblMessage;
    btnSunny = new Button("Sunny");
    btnCloudy = new Button("Cloudy");
    lblMessage = new Label("Click a button.");

    // Create an anonymous inner class to handle btnSunny
    btnSunny.setOnAction(new EventHandler<ActionEvent>()
    {
        @Override
        public void handle(ActionEvent event)
        {
            lblMessage.setText("It is sunny!");
        }
    });
};
```

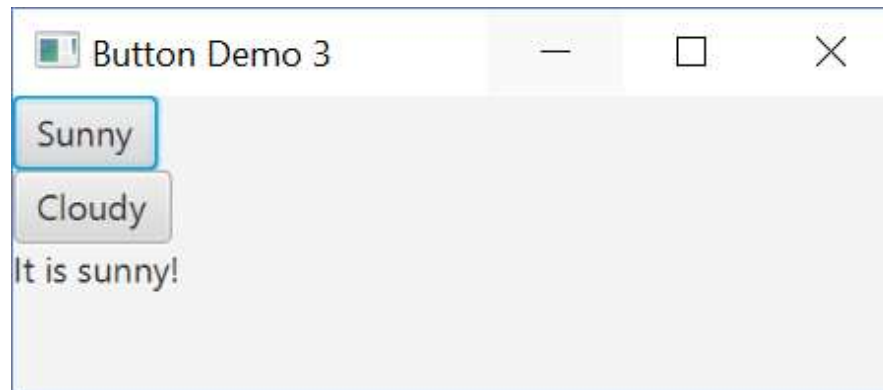
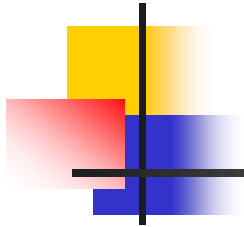



```
// Create an anonymous inner class to handle btnCloudy
btnCloudy.setOnAction(new EventHandler<ActionEvent>()
{
    @Override
    public void handle(ActionEvent event)
    {
        lblMessage.setText("It is cloudy!");
    }
});

root.getChildren().add(btnSunny);
root.getChildren().add(btnCloudy);
root.getChildren().add(lblMessage);

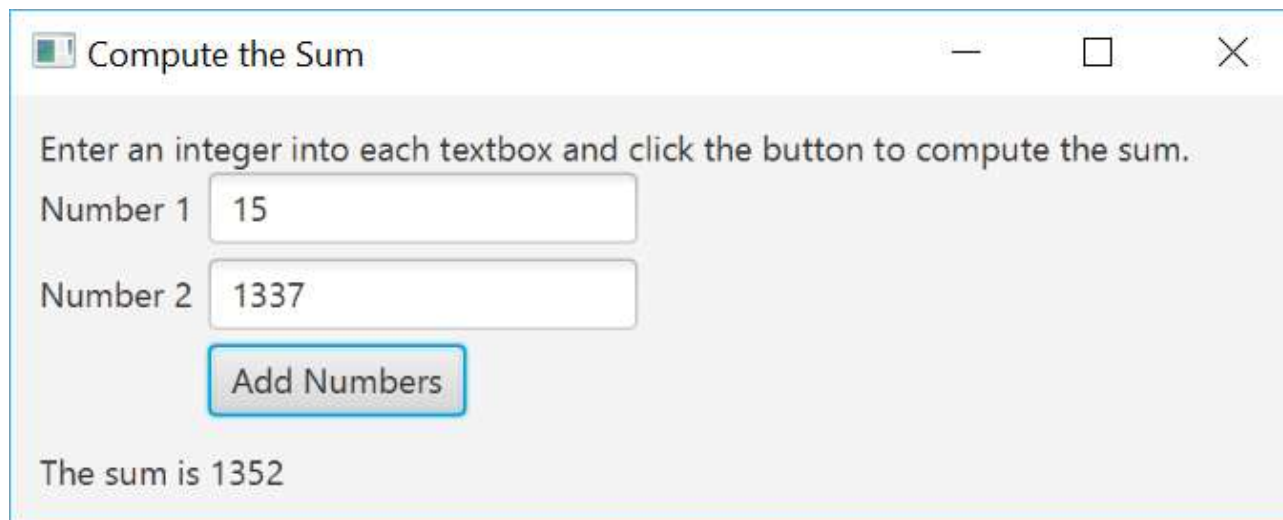
Scene scene = new Scene(root, 300, 100);
primaryStage.setTitle("Button Demo 3");
primaryStage.setScene(scene);
primaryStage.show();
}
}
```





Event Handling Example – Adding Numbers

- An example that ties together layouts (GridPane with BorderPane) and event handling to add two numbers together is shown in AddingNumbersApp
- event handling demo 4
 - listing 8.24, **AddingNumbersApp**



The screenshot shows a Java Swing window titled "Compute the Sum". Inside the window, there is a text label that says "Enter an integer into each textbox and click the button to compute the sum." Below this label are two text input fields. The first field is labeled "Number 1" and contains the value "15". The second field is labeled "Number 2" and contains the value "1337". Below these fields is a button labeled "Add Numbers". At the bottom of the window, there is a text label that says "The sum is 1352".



listing 8.24, **AddingNumbersApp**

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.BorderPane;
import javafx.scene.control.TextField;
import javafx.scene.control.Label;
import javafx.scene.control.Button;
import javafx.geometry.Insets;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

/**
This application embeds a GridPane in the center of a
BorderPane. It allows the user to enter numbers into text fields
which are added together in the button click event. The sum is
displayed in a label in the bottom region of the BorderPane.
*/
public class AddingNumbersApp extends Application
{
    public static void main(String[] args)
    {
        launch(args);
    }
}
```



@Override

public void start(Stage primaryStage) throws Exception

{

BorderPane root = new BorderPane();

// Margin of 10 pixels

root.setPadding(new Insets(10,10,10,10));

Button btnAdd;

TextField txtNum1, txtNum2;

Label lblSum;

// Add a label message in the top. We create the

// label without a named reference since the label

// is read-only; we never change it so no reference is needed.

**root.setTop(new Label("Enter an integer into each textbox " +
"and click the button to compute the sum.));**

// The label that will display the sum goes into the bottom.

// Initially it is just a blank string.

lblSum = new Label("");

root.setBottom(lblSum);



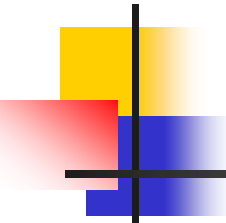
```
// Create a GridPane in the center of the BorderPane
GridPane center = new GridPane();
center.setVgap(5);
center.setHgap(5);
txtNum1 = new TextField("0"); // Default text of 0
txtNum1.setPrefWidth(150);
txtNum2 = new TextField("0");
txtNum2.setPrefWidth(150);
center.add(new Label("Number 1"), 0, 0);
center.add(new Label("Number 2"), 0, 1);
center.add(txtNum1, 1, 0);
center.add(txtNum2, 1, 1);
btnAdd = new Button("Add Numbers");
center.add(btnAdd, 1, 2);
root.setCenter(center);
```



```
// Set the event handler when the button is clicked
btnAdd.setOnAction(new EventHandler<ActionEvent>()
{
    @Override
    public void handle(ActionEvent event)
    {
        int num1 = Integer.parseInt(txtNum1.getText());
        int num2 = Integer.parseInt(txtNum2.getText());
        int sum = num1 + num2;
        lblSum.setText("The sum is " + sum);
    }
});

Scene scene = new Scene(root, 450, 150);
primaryStage.setTitle("Compute the Sum");
primaryStage.setScene(scene);
primaryStage.show();
}
```





Compute the Sum

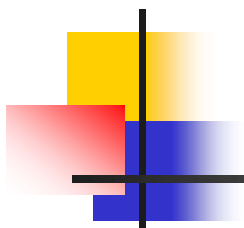
Enter an integer into each textbox and click the button to compute the sum.

Number 1

Number 2

The sum is 1352





בב
ע



Listing A Window interface Derived from JFrame

```
JButton sunnyButton = new JButton("Sunny");
    sunnyButton.addActionListener(this);
    contentPane.add(sunnyButton);

    JButton cloudyButton = new JButton("Cloudy");
    cloudyButton.addActionListener(this);
    contentPane.add(cloudyButton);
}

public void actionPerformed(ActionEvent e)
{
    String actionCommand = e.getActionCommand( );
    Container contentPane = getContentPane( );

    if (actionCommand.equals("Sunny"))
        contentPane.setBackground(Color.BLUE);
    else if (actionCommand.equals("Cloudy"))
        contentPane.setBackground(Color.GRAY);
    else
        System.out.println("Error in button interface.");
}

public void windowClosing(WindowEvent e)
{
    System.exit(0);
}

public void windowActivated(WindowEvent e) { }

.....
}
```

@Override

public void start(Stage primaryStage) throws Exception

{

VBox root = new VBox();

Button btnSunny;

Button btnCloudy;

btnSunny = new Button("Sunny");

btnCloudy = new Button("Cloudy");

// Create an event object to handle the button click.

// The "handle" method in HandleButtonClick will be

// invoked when the button is clicked.

HandleButtonClick clickEvent = new HandleButtonClick();

btnSunny.setOnAction(clickEvent);

// We can also create the HandleButtonClick object without

// a named reference by creating it inside the call to setOnAction

btnCloudy.setOnAction(new HandleButtonClick("It is cloudy."));

root.getChildren().add(btnSunny);

root.getChildren().add(btnCloudy);

Scene scene = new Scene(root, 300, 100);

primaryStage.setTitle("Button Event Handling Demo");

primaryStage.setScene(scene);

primaryStage.show();

}

}