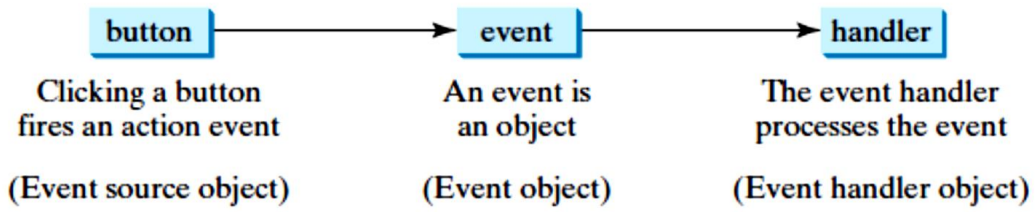
Event-Driven Programming and Animations

You can write code to process events such as a button click, mouse movement, and keystrokes.
To respond to a button click, you need to write the code to process the button-clicking action. The button is an event source object—where the action originates. You need to create an object capable of handling the action event on a button. This object is called an event handler.



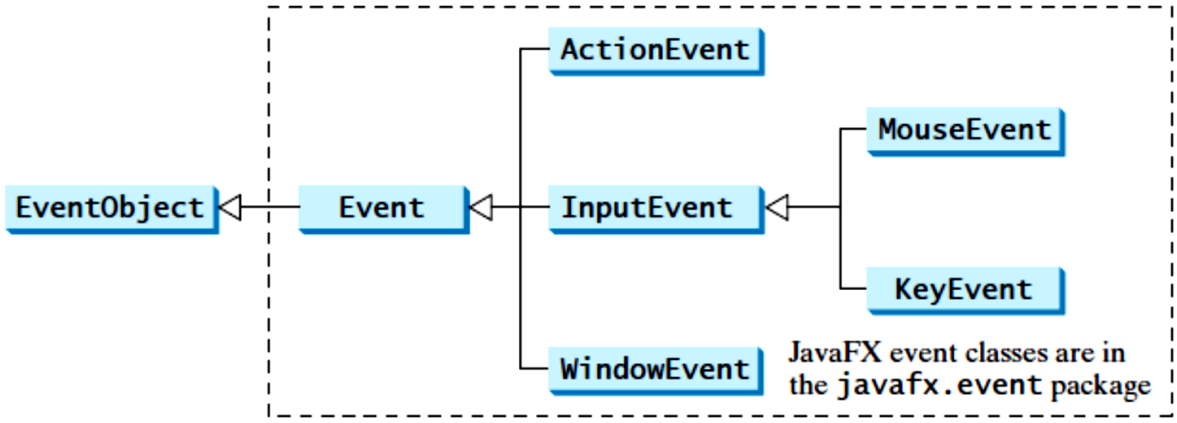
There are four important terms you need to know:

✓ Event: An object that's created when the user does something noteworthy with a component, such as clicking it.

✓ Event source: The object on which the event initially occurred.

✓ Event target: The node that the event is directed at. This is usually the button or other control that the user clicked or otherwise manipulated. (In most cases, the event source and the event target are the same.)

✓ Event handler: The object that listens for events and handles them when they occur. The event-listener object must implement the Event Handler interface, which defines a single method named handle (see below Table). The Event Handler interface is defined in the package javafx.event.



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>)

Not all objects can be handlers for an action event. To be a handler of an action event, two requirements must be met:

1. The object must be an instance of the **EventHandler<T extends Event>** interface. This interface defines the common behavior for all handlers. **<T extends Event>** denotes that **T** is a generic type that is a subtype of **Event**.
2. The **EventHandler** object **handler** must be registered with the event source object using an appropriate method such as **source.setOnAction(handler)**.

There are three steps you must take to handle a JavaFX event:

1. Create an event source: An event source is simply a control, such as a button, mouse, keyboard that can generate events in the start method, you can create the button like this:

```
Button btn = new Button("Click me please!");
```

2. Create an event handler : To create an event handler, you must create an object that implements the EventHandler interface and provides an implementation of the handle method.

There are four ways to create an event handler:

FIRST: Add implements EventHandler to the program's Application class and provide an implementation of the handle method.

SECOND : Create an inner class that implements EventHandler within the Application class.

THIRD : Create an anonymous class that implements EventHandler.

FORTH : Use a Lambda expression to implement the handle method.

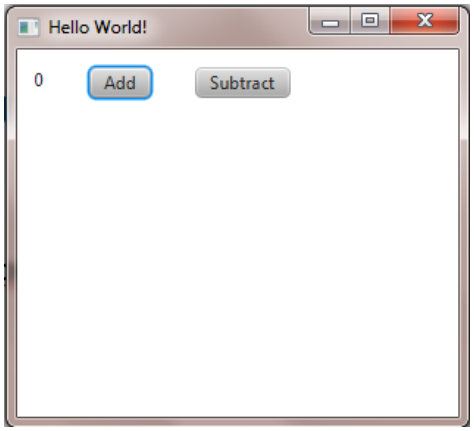
3. ***Register the event handler with the event source:*** The final step is to register the event handler with the event source so that the handle method is called whenever the event occurs.

For example, a Button control provides a setOnAction method that lets you register an event handler for the action event. In the setOnAction method, you specify the event handler object as a parameter.

TO Create an event handler **,** There are four ways:
First : Add implements EventHandler to the program's Application class and provide an implementation of the handle method.
Second : Create an inner class that implements EventHandler within the Application class.
Third : Create an anonymous class that implements EventHandler.
Fourth : Use a Lambda expression to implement the handle method.

First Method : *Add implements EventHandler to the program's Application class.*

```
package javafxapplication39;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.control.Label;
import javafx.geometry.Insets;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

public class JavaFXApplication39 extends Application {
   int icounter = 0 ;
   Label lb = new Label(Integer.toString(icounter));

   @Override
   public void start(Stage primaryStage) {

      // create Addition Button
      Button btnAdd = new Button();
      btnAdd.setText("Add");
      btnAdd.setOnAction(new EventHandler<ActionEvent>() {
         @Override
         public void handle(ActionEvent event) {
            icounter++;
            lb.setText(Integer.toString(icounter));
         }
      });

      // create Subtraction Button
      Button btnSub = new Button();
      btnSub.setText("Subtract");
```



```
    btnSub.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            icounter--;
            lb.setText(Integer.toString(icounter));
        }
    });

    HBox root = new HBox();
    root.setPadding(new Insets(12,12,12,12));
    root.setSpacing(60);
    root.getChildren().addAll(lb, btnAdd, btnSub);

    Scene scene = new Scene(root, 300, 250);

    primaryStage.setTitle("Hello World!");
    primaryStage.setScene(scene);
    primaryStage.show();
  } }
```

Second Method : *Create an inner class that implements EventHandler within the Application class.*

// Circle Enlarge and Shrink

```
package javafxapplication2;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.shape.*;
import javafx.scene.paint.*;
import javafx.geometry.*;
import javafx.scene.control.Button;
import javafx.event.*;

public class JavaFXApplication2 extends Application {
 private circlepane cp = new circlepane();

@Override
public void start(Stage primaryStage) {

   HBox hBox = new HBox();
   hBox.setSpacing(5);
```

```
    hBox.setAlignment(Pos.CENTER);
    Button btEnlarge = new Button("Enlarge");
    Button btShrink = new Button("Shrink");
    hBox.getChildren().addAll(btEnlarge, btShrink);

    btEnlarge.setOnAction(new EnlargeHandler());
    btShrink.setOnAction(new shrinkHandler());

    BorderPane borderPane = new BorderPane();
    borderPane.setCenter(cp);
    borderPane.setBottom(hBox);
    BorderPane.setAlignment(hBox, Pos.CENTER);

    Scene scene = new Scene(borderPane, 300, 300);
    primaryStage.setTitle("ShowImage"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
    }

class EnlargeHandler implements EventHandler<ActionEvent>{
   @Override
   public void handle(ActionEvent e){
      cp.enlarge();
   }

class shrinkHandler implements EventHandler<ActionEvent>{
   @Override
   public void handle(ActionEvent e){
      cp.shrink();
   }

}

class circlepane extends StackPane{
   private Circle circle = new Circle(50);

   public circlepane(){
      circle.setStroke(Color.BLACK);
      circle.setFill(Color.BLUE);
      getChildren().add(circle);
}
public void enlarge(){
    circle.setRadius(circle.getRadius()+5);
}

public void shrink(){
```

```
    circle.setRadius(circle.getRadius()> 5 ? circle.getRadius()- 5 :circle.getRadius());
}
}
}
```

Third Method : *Create an anonymous class that implements EventHandler.*

```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new EnlargeHandler());
}

class EnlargeHandler
    implements EventHandler<ActionEvent> {
  public void handle(ActionEvent e) {
    circlePane.enlarge();
  }
}
```

(a) Inner class EnlargeListener

```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new class EnlargeHandler
      implements EventHandler<ActionEvent>() {
      public void handle(ActionEvent e) {
        circlePane.enlarge();
      }
    });
}
```

(b) Anonymous inner class

```
package javafxapplication38;

import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.geometry.Insets;
import javafx.scene.control.*;
import javafx.event.*;

public class JavaFXApplication38 extends Application implements EventHandler <ActionEvent> {

    Button btnAdd;
    Button btnSubtract;
    Label lbl;
    int iCounter = 0;

    @Override
    public void start(Stage primaryStage){

        // Create the Add button
        btnAdd = new Button();
        btnAdd.setText("Add");
        btnAdd.setOnAction((EventHandler<ActionEvent>) this);
```

```
        // Create the Subtract button
        btnSubtract = new Button();
        btnSubtract.setText("Subtract");
        btnSubtract.setOnAction((EventHandler<ActionEvent>) this);

        // Create the Label
        lbl = new Label();
        lbl.setText(Integer.toString(iCounter));

        // Add the buttons and label to an HBox pane
        HBox pane = new HBox(10);
        pane.setSpacing(30);
        pane.setPadding(new Insets(12, 12, 12, 12));
        pane.getChildren().addAll(lbl, btnAdd, btnSubtract);

        // Add the layout pane to a scene
        Scene scene = new Scene(pane, 200, 75);
        // Add the scene to the stage, set the title
        // and show the stage
        primaryStage.setScene(scene);
        primaryStage.setTitle("Add/Sub");
        primaryStage.show();
        }

@Override
   public void handle(ActionEvent e)
   {
                if (e.getSource()==btnAdd)
                        {      iCounter++;      }
                else
                        {    if (e.getSource()==btnSubtract)
                         {       iCounter--;       }     }
   lbl.setText(Integer.toString(iCounter));
  } // End of Handle
}
```

Another Example :

```
package javafxapplication53;

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

```
import javafx.geometry.Pos;

public class JavaFXApplication53 extends Application {

   @Override
   public void start(Stage primaryStage) {
      // Hold two buttons in an HBox
      HBox hBox = new HBox();
      hBox.setSpacing(10);
      hBox.setAlignment(Pos.CENTER);
      Button btNew = new Button("New");
      Button btOpen = new Button("Open");
      Button btSave = new Button("Save");
      Button btPrint = new Button("Print");
      hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);

      // Create and register the handler
      btNew.setOnAction(new EventHandler<ActionEvent>() {
      @Override                                    // Override the handle method
      public void handle(ActionEvent e) {
         System.out.println("Process New");
       }
      });

      btOpen.setOnAction(new EventHandler<ActionEvent>() {
      @Override                                    // Override the handle method
      public void handle(ActionEvent e) {
         System.out.println("Process Open");
        }
      });

      btSave.setOnAction(new EventHandler<ActionEvent>() {
      @Override                                    // Override the handle method
      public void handle(ActionEvent e) {
         System.out.println("Process Save");
         }
      });

      btPrint.setOnAction(new EventHandler<ActionEvent>() {
      @Override                                     // Override the handle method
      public void handle(ActionEvent e) {
         System.out.println("Process Print");
          }
      });

      // Create a scene and place it in the stage
```

```
        Scene scene = new Scene(hBox, 300, 50);
        primaryStage.setTitle("AnonymousHandlerDemo"); // Set title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
}
```

Fourth Method : *Use a Lambda expression to implement the handle method.*

Lambda expressions can be used to greatly simplify coding for event handling. Lambda expression is a new feature in Java 8. For example, the following code in (a) can be greatly simplified using a lambda expression in (b) in three lines.

```
btEnlarge.setOnAction(
  new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      // Code for processing event e
    }
  }
});
```

(a) Anonymous inner class event handler

```
btEnlarge.setOnAction(e -> {
  // Code for processing event e
});
```

(b) Lambda expression event handler

```
package javafxapplication54;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;

//don't need to open ***** import javafx.event.EventHandler;

public class JavaFXApplication54 extends Application {

    @Override
    public void start(Stage primaryStage) {
        // Hold two buttons in an HBox
        HBox hBox = new HBox();
        hBox.setSpacing(10);
        hBox.setAlignment(Pos.CENTER);
```

```
    Button btNew = new Button("New");
    Button btOpen = new Button("Open");
    Button btSave = new Button("Save");
    Button btPrint = new Button("Print");
    hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);

    // Create and register the handler
    // uses a declared type.
    btNew.setOnAction((ActionEvent e) -> { System.out.println("Process New");  });
    // uses an inferred type since the type can be determined by the compiler.
    btOpen.setOnAction((e) -> {  System.out.println("Process Open");  });
    // omits the parentheses for a single inferred type.
    btSave.setOnAction(e -> { System.out.println("Process Save") ;  });
    // omits the braces for a single statement in the body.
    btPrint.setOnAction(e -> System.out.println("Process Print"));

    // Create a scene and place it in the stage
    Scene scene = new Scene(hBox, 300, 50);
    primaryStage.setTitle("LambdaHandlerDemo"); // Set title
    primaryStage.setScene(scene);
    primaryStage.show();
  }
}
```

Different use of lambda e