JAVA PROGRAMMING

Chap 9 : Event Handling using Java FX

Introduction to Java FX Event Handling

- JavaFX provides us the flexibility to create various types of applications such as Desktop Applications, Web applications and graphical applications.
- In the modern day applications, the users play a vital role in the proper execution of the application. The user need to interact with the application in most of the cases.
- In JavaFX, An event is occurred whenever the user interacts with the application nodes.
- There are various sources by using which, the user can generate the event.
- For example, User can make the use of mouse or it can press any button on the keyboard or it can scroll any page of the application in order to generate an event.
- Hence, we can say that the events are basically the notifications that tell us that something has occurred at the user's end.
- A perfect Application is the one which takes the least amount of time in handling the events.
- **Types of Events**
- The events can be broadly classified into the following two categories −
- Foreground Events − These events require the direct interaction of a user. They are generated as consequences of a person interacting with the graphical components in a GUI. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.
- Background Events − These events that don't require the interaction of end-user. The operating system interruptions, hardware or software failure, timer expiry, operation completion are the example of background events.

Java FX Events

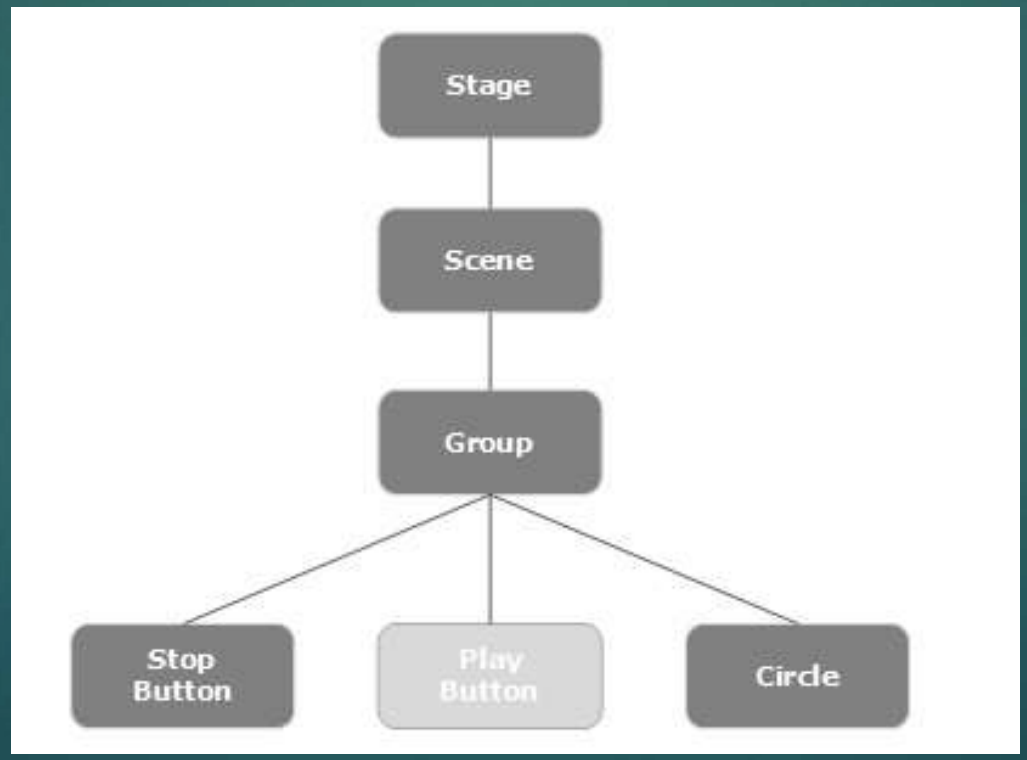
- Events are basically used to notify the application about the actions taken by the user.
- JavaFX provides the mechanism to capture the events, route the event to its target and letting the application handle the events.
- JavaFX provides support to handle a wide varieties of events.
- The class named Event of the package javafx.event is the base class for an event.
- An instance of any of its subclass is an event.
- JavaFX provides a wide variety of events. Some of them are are listed below.
- **Mouse Event** − This is an input event that occurs when a mouse is clicked. It is represented by the class named MouseEvent. It includes actions like mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target, etc.
- **Key Event** − This is an input event that indicates the key stroke occurred on a node. It is represented by the class named KeyEvent. This event includes actions like key pressed, key released and key typed.
- **Drag Event** − This is an input event which occurs when the mouse is dragged. It is represented by the class named DragEvent. It includes actions like drag entered, drag dropped, drag entered target, drag exited target, drag over, etc.
- **Window Event** − This is an event related to window showing/hiding actions. It is represented by the class named WindowEvent. It includes actions like window hiding, window shown, window hidden, window showing, etc.

Java FX Event Handling

- Event Handling is the mechanism that controls the event and decides what should happen, if an event occurs.
- This mechanism has the code which is known as an event handler that is executed when an event occurs.
- JavaFX provides handlers and filters to handle events.
- In JavaFX every event has −
  - Target − The node on which an event occurred. A target can be a window, scene, and a node.
  - Source − The source from which the event is generated will be the source of the event. In the above scenario, mouse is the source of the event.
  - Type − Type of the occurred event; in case of mouse event – mouse pressed, mouse released are the type of events.

Phases of Event Handling in JavaFX

- Also known as **Event Delivery Process**
- Whenever an event is generated, JavaFX undergoes the following phases in order to handle the events.
- **Route Construction :** Whenever an event is generated, the default/initial route of the event is determined by construction of an Event Dispatch chain.
- It is the path from the stage to the source Node.
- An event dispatch chain is created in the following image for the event generated on one of the scene graph node.



Phases of Event Handling in JavaFX

- **Event Capturing Phase** : Once the Event Dispatch Chain is created, the event is dispatched from the source node of the event.
- All the nodes are traversed by the event from top to bottom.
- If the event filter is registered with any of these nodes, then it will be executed.
- If any of the nodes are not registered with the event filter then the event is transferred to the target node.
- The target node processes the event in that case.
- **Event Bubbling Phase** : In the event bubbling phase, the event is travelled from the target node to the stage node (bottom to top).
- If any of the nodes in the event dispatch chain has a handler registered for the generated event, it will be executed.
- If none of these nodes have handlers to handle the event, then the event reaches the root node and finally the process will be completed.
- **Event Handlers and Filters** : Event Handlers and filters contains application logic to process an event.
- A node can be registered to more than one Event Filters.
- The interface javafx.event.EventHandler must be implemented by all the event handlers and filters.
- In case of parent–child nodes, you can provide a common filter/handler to the parents, which is processed as default for all the child nodes.
- during the event processing phase, a filter is executed and during the event bubbling phase, a handler is executed.

Adding and Removing Event Filter

- To add an event filter to a node, you need to register this filter using the method addEventFilter() of the Node class.

```
//Creating the mouse event handler
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
   @Override
   public void handle(MouseEvent e) {
      System.out.println("Hello World");
      circle.setFill(Color.DARKSLATEBLUE);
   }
};
//Adding event Filter
Circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

- In the same way, you can remove a filter using the method removeEventFilter() as shown below –

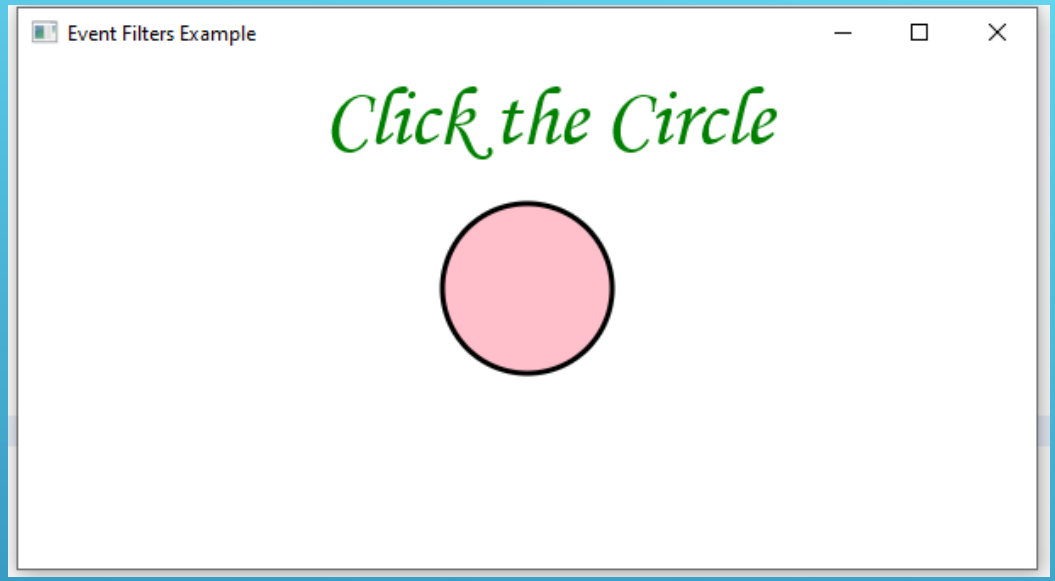
```
circle.removeEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

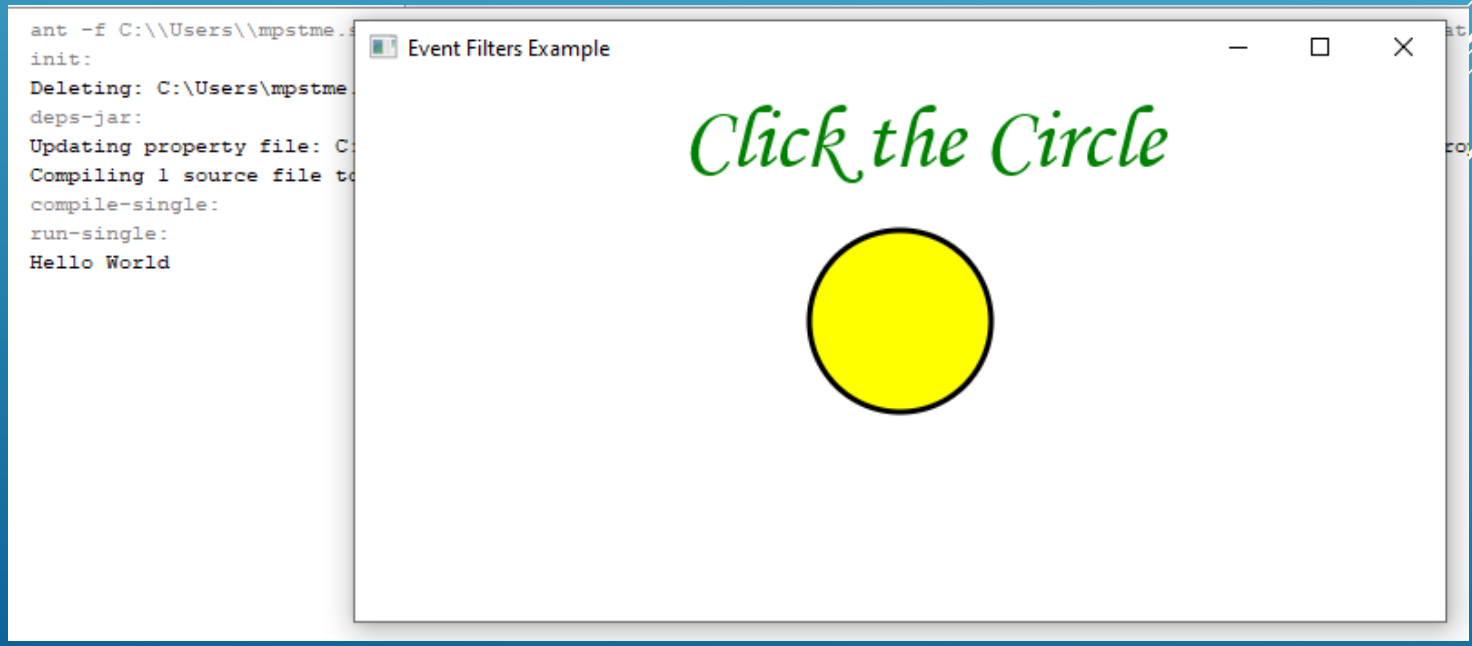
Adding and Removing Event Filter

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class EventHandling extends Application {
   @Override
   public void start(Stage stage) {
      Circle circle = new Circle();
      circle.setCenterX(300.0f);
      circle.setCenterY(135.0f);
      circle.setRadius(50);
      circle.setFill(Color.PINK);
      circle.setStrokeWidth(3);
      circle.setStroke(Color.BLACK);
      Text text = new Text("Click the Circle");
      text.setFont(Font.font("Monotype Corsiva", FontWeight.BOLD, 50));
      text.setFill(Color.GREEN);
      text.setX(180);
      text.setY(50);

      //Creating the mouse event handler
      EventHandler<MouseEvent> eventHandler = new
EventHandler<MouseEvent>() {
         @Override
         public void handle(MouseEvent e) {
            System.out.println("Hello World");
            circle.setFill(Color.YELLOW);
         }  };

      //Registering the event filter
      circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

Adding and Removing Event Filter

```
Group root = new Group(circle, text);
        Scene scene = new Scene(root, 600, 300);
        stage.setTitle("Event Filters Example");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```





Convenience Methods

- Some of the classes in JavaFX define event handler properties.
- By setting the values to these properties using their respective setter methods, you can register to an event handler. These methods are known as convenience methods.
- Most of these methods exist in the classes like Node, Scene, Window, etc., and they are available to all their sub classes.
- Some EventHandler properties along with their setter methods (convenience methods) are described in the following table.

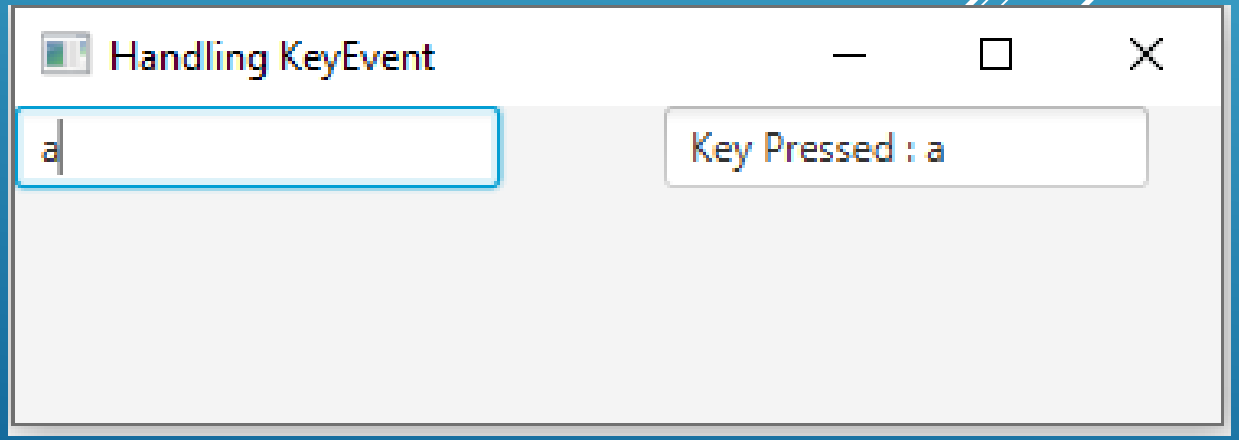
EventHandler Property	Description	Setter Methods
onDragDetected	This is of the type EventHandler of MouseEvent. This indicates a function which is to be called when the drag gesture is detected.	setOnDragDetected(EventHandler value)
onDragDone	This is of the type EventHandler of DragEvent.	setOnDragDone(EventHandler value)
onDragDropped	This is of the type EventHandler of DragEvent. This is assigned to a function which is to be called when the mouse is released during a drag and drop operation.	setOnDragDropped(EventHandler value)
onInputMethodTextChanged	This is of the type EventHandler of InputMethodEvent. This is assigned to a function which is to be called when the Node has focus and the input method text has changed.	setOnInputMethodTextChanged(EventHandler value)
onKeyPressed	This is of the type EventHandler of KeyEvent. This is assigned to a function which is to be called when the Node has focus and key is pressed.	setOnKeyPressed(EventHandler value)
onKeyReleased	This is of the type EventHandler of KeyEvent. This is assigned to a function which is to be called when the Node has focus and key is released.	setOnKeyReleased(EventHandler value)
onKeyTyped	This is of the type EventHandler of KeyEvent.This is assigned to a function which is to be called when the Node has focus and key is typed.	setOnKeyTyped(EventHandler value)

Convenience Methods

EventHandler Property	Description	Setter Methods
onMouseClicked	This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is clicked on the node.	setOnMouseClicked(EventHandler value)
onMouseDragEntered	This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture enters the node.	setOnMouseDragEntered(EventHandler value)
onMouseDragExited	This is of the type EventHandler of MouseDragEvent.This is assigned to a function which is to be called when a full press drag release gesture exits the node.	setOnMouseDragExited(EventHandler value)
onMouseDragged	This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when the mouse button is pressed and dragged on the node.	setOnMouseDragged(EventHandler value)
onMouseDragOver	This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture progresses within the node.	setOnMouseDragOver(EventHandler value)
onMouseDragReleased	This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture ends within the node.	setOnMouseDragReleased(EventHandler value)
onMouseEntered	This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse enters the node.	setOnMouseEntered(EventHandler value)
onMouseExited	This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse exits the node.	setOnMouseExited(EventHandler value)
onMouseMoved	This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse moves within the node and no button has been pushed.	setOnMouseMoved(EventHandler value)
onMousePressed	This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is pressed on the node.	setOnMousePressed(EventHandler value)
onMouseReleased	This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is released on the node.	setOnMouseReleased(EventHandler value)

Convenience Methods Example (setOnKeyPressed)

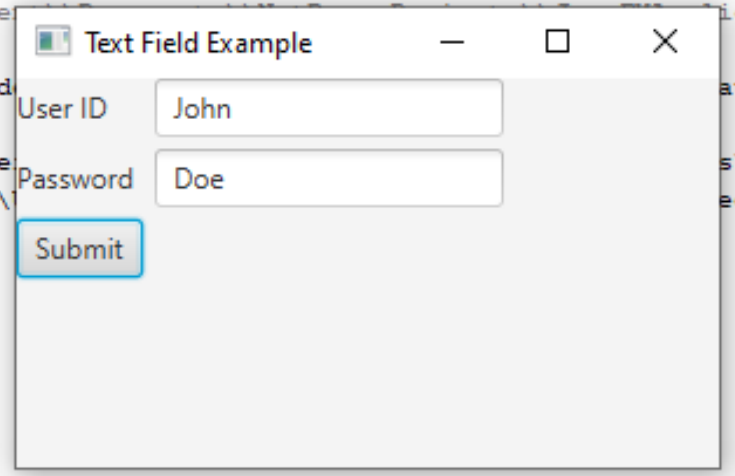
```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyEvent;
import javafx.stage.Stage;
import javafx.scene.layout.GridPane;
public class ConvenienceMethodsDemo extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        TextField tf1 = new TextField();
        TextField tf2 = new TextField();
        //Handling KeyEvent for textfield 1
        tf1.setOnKeyPressed(new EventHandler<KeyEvent>() {
            @Override
            public void handle(KeyEvent k) {
                // TODO Auto-generated method stub
                tf2.setText("Key Pressed :"+" "+k.getText());
            }    });
        //setting group and scene
        GridPane root = new GridPane();
        root.addRow(0, tf1, tf2);
        root.setHgap(50);
        Scene scene = new Scene(root,500,200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Handling KeyEvent");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    } }
```



Convenience Methods Example (setOnAction)

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
public class TextFieldDemo extends Application {
@Override
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Label user_id=new Label("User ID");
    Label password = new Label("Password");
    TextField tf1=new TextField();
    TextField tf2=new TextField();
    Button b = new Button("Submit");
    GridPane root = new GridPane();
    root.addRow(0, user_id, tf1);
    root.addRow(1, password, tf2);
    root.addRow(2, b);
    root.setHgap(5);
    root.setVgap(5);
    b.setOnAction(new EventHandler<ActionEvent>() {
@Override
    public void handle(ActionEvent args) {
    // TODO Auto-generated method stub
    System.out.println("User Id entered is : " + tf1.getText());
    System.out.println("Password entered is : " + tf2.getText());
            }
        } );
     Scene scene=new Scene(root,300,300);
     primaryStage.setScene(scene);
     primaryStage.setTitle("Text Field Example");
     primaryStage.show();
}
public static void main(String[] args) {
launch(args);
} }
```

```
ant -f C:\\Users\\mpstme.stude
init:
Deleting: C:\Users\mpstme.stud
deps-jar:
Updating property file: C:\Use
Compiling 1 source file to C:\
compile-single:
run-single:
User Id entered is : John
Password entered is : Doe
```



Event Handlers

- JavaFX facilitates us to use the Event Handlers to handle the events generated by Keyboard Actions, Mouse Actions, and many more source nodes.
- Event Handlers are used to handle the events in the Event bubbling phase. There can be more than one Event handlers for a single node.
- We can also use single handler for more than one node and more than one event type. In this part of the tutorial, we will discuss, how the Event Handlers can be used for processing events.
- **Adding an Event Handler**
- Event Handler must be registered for a node in order to process the events in the event bubbling phase.
- Event handler is the implementation of the EventHandler interface.
- The handle() method of the interface contains the logic which is executed when the event is triggered.
- To register the EventHandler, addEventHandler() is used.
- In this method, two arguments are passed. One is event type and the other is EventHandler object.
- The syntax of addEventHandler() is given below.

```
node.addEventHandler(<EventType>,new EventHandler<Event-Type>()
{
public void handle(<EventType> e)
{
//handling code
}       });
```

Event Handlers

- **Removing EventHandler**
- when we no longer need an EventHandler to process the events for a node or event types, we can remove the EventHandler by using the method removeEventHandler() method.
- This method takes two arguments, event type and EventHandler Object.
- node.removeEventHandler(<EventType>,handler);