**JavaFX Overview**

**What is JavaFX?**

JavaFX is a Java library used to develop Desktop applications as well as Rich Internet Applications (RIA). The applications built in JavaFX, can run on multiple platforms including Web, Mobile and Desktops.

JavaFX is intended to replace swing in Java applications as a GUI framework. However, It provides more functionalities than swing. Like Swing, JavaFX also provides its own components and doesn't depend upon the operating system. It is lightweight and hardware accelerated. It supports various operating systems including Windows, Linux and Mac OS.

**History of JavaFX**

JavaFX was developed by Chris Oliver. Initially the project was named as **Form Follows Functions (F3)** . It is intended to provide the richer functionalities for the GUI application development. Later, Sun Micro-systems acquired F3 project as **JavaFX** in June, 2005.

Sun Micro-systems announces it officially in **2007** at **W3 Conference**. In October 2008, JavaFX 1.0 was released. In 2009, ORACLE corporation acquires Sun Micro-Systems and released JavaFX 1.2. the latest version of JavaFX is JavaFX 1.8 which was released on 18th March 2014.

**Features of JavaFX**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Java Library | It is a Java library which consists of many classes and interfaces that are written in Java. |
| FXML | FXML is the XML based Declarative mark up language. The coding can be done in FXML to provide the more enhanced GUI to the user. |
| Scene Builder | Scene Builder generates FXML mark-up which can be ported to an IDE. |
| Web view | Web pages can be embedded with JavaFX applications. Web View uses WebKitHTML technology to embed web pages. |
| Built in UI controls | JavaFX contains Built-in components which are not dependent on operating system. The UI component are just enough to develop a full featured application. |
| CSS like styling | JavaFX code can be embedded with the CSS to improve the style of the application. We can enhance the view of our application with the simple knowledge of CSS. |
| Swing interoperability | The JavaFX applications can be embedded with swing code using the Swing Node class. We can update the existing swing application with the powerful features of JavaFX. |
| Canvas API | Canvas API provides the methods for drawing directly in an area of a JavaFX scene. |
| Rich Set of APIs | JavaFX provides a rich set of API's to develop GUI applications. |
| Integrated Graphics Library | An integrated set of classes are provided to deal with 2D and 3D graphics. |
| Graphics Pipeline | JavaFX graphics are based on Graphics rendered pipeline(prism). It offers smooth graphics which are hardware accelerated. |
| High Performance Media Engine | The media pipeline supports the playback of web multimedia on a low latency. It is based on a Gstreamer Multimedia framework. |
| Self-contained application deployment model | Self Contained application packages have all of the application resources and a private copy of Java and JavaFX Runtime. |

**JavaFX UI Controls**

This part of the tutorial provides you the in-depth knowledge of JavaFX UI controls. The graphical user interface of every desktop application mainly considers UI elements, layouts and behaviour.

The UI elements are the one which are actually shown to the user for interaction or information exchange. Layout defines the organization of the UI elements on the screen. Behaviour is the reaction of the UI element when some event is occurred on it.

However, the package **javafx.scene.control** provides all the necessary classes for the UI components like Button, Label, etc. Every class represents a specific UI control and defines some methods for their styling.

|  |  |  |
| --- | --- | --- |
| **SN** | **Control** | **Description** |
| 1 | [Label](https://www.javatpoint.com/javafx-label) | Label is a component that is used to define a simple text on the screen. Typically, a label is placed with the node, it describes. |
| 2 | [Button](https://www.javatpoint.com/javafx-button) | Button is a component that controls the function of the application. Button class is used to create a labelled button. |
| 3 | [RadioButton](https://www.javatpoint.com/javafx-radiobutton) | The Radio Button is used to provide various options to the user. The user can only choose one option among all. A radio button is either selected or deselected. |
| 4 | [CheckBox](https://www.javatpoint.com/javafx-checkbox) | Check Box is used to get the kind of information from the user which contains various choices. User marked the checkbox either on (true) or off(false). |
| 5 | [TextField](https://www.javatpoint.com/javafx-textfield) | Text Field is basically used to get the input from the user in the form of text. javafx.scene.control.TextField represents TextField |
| 6 | [PasswordField](https://www.javatpoint.com/javafx-passwordfield) | PasswordField is used to get the user's password. Whatever is typed in the passwordfield is not shown on the screen to anyone. |
| 7 | [HyperLink](https://www.javatpoint.com/javafx-hyperlink) | HyperLink are used to refer any of the webpage through your appication. It is represented by the class **javafx.scene.control.HyperLink** |
| 8 | [Slider](https://www.javatpoint.com/javafx-slider) | Slider is used to provide a pane of options to the user in a graphical form where the user needs to move a slider over the range of values to select one of them. |
| 9 | [ProgressBar](https://www.javatpoint.com/javafx-progressbar) | Progress Bar is used to show the work progress to the user. It is represented by the class **javafx.scene.control.ProgressBar**. |
| 10 | [ProgressIndicator](https://www.javatpoint.com/javafx-progress-indicator) | Instead of showing the analogue progress to the user, it shows the digital progress so that the user may know the amount of work done in percentage. |
| 11 | [ScrollBar](https://www.javatpoint.com/javafx-scrollbar) | JavaFX Scroll Bar is used to provide a scroll bar to the user so that the user can scroll down the application pages. |
| 12 | [Menu](https://www.javatpoint.com/javafx-menu) | JavaFX provides a Menu class to implement menus. Menu is the main component of any application. |
| 13 | [ToolTip](https://www.javatpoint.com/javafx-tooltip) | JavaFX ToolTip is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application. |

**JavaFX Label**

**javafx.scene.control.Label** class represents label control. As the name suggests, the label is the component that is used to place any text information on the screen. It is mainly used to describe the purpose of the other components to the user. You can not set a focus on the label using the Tab key.

**Package: javafx.scene.control**

**Constructors:**

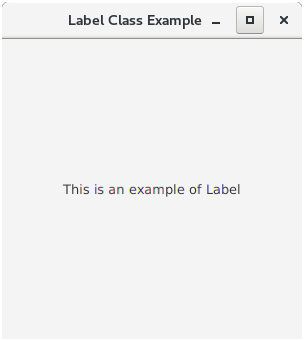
1. Label(): creates an empty Label
2. Label(String text): creates Label with the supplied text
3. Label(String text, Node graphics): creates Label with the supplied text and graphics

**Adding a Node to the scene graph**

The following code implements Label into our Application.

1. package application;
3. import javafx.application.Application;
4. import javafx.scene.Scene;
5. import javafx.scene.control.Label;
6. import javafx.scene.layout.StackPane;
7. import javafx.stage.Stage;
9. public class LabelTest extends Application {
11. @Override
12. public void start(Stage primaryStage) throws Exception {
13. // TODO Auto-generated method stub
14. Label my\_label=new Label("This is an example of Label");
15. StackPane root = new StackPane();
16. Scene scene=new Scene(root,300,300);
17. root.getChildren().add(my\_label);
18. primaryStage.setScene(scene);
19. primaryStage.setTitle("Label Class Example");
20. primaryStage.show();
22. }
23. public static void main(String[] args) {
24. launch(args);
25. }
26. }

**Output:**



**avaFX Button**

JavaFX button control is represented by **javafx.scene.control.Button** class. A button is a component that can control the behaviour of the Application. An event is generated whenever the button gets clicked.

**How to create a Button?**

Button can be created by instantiating Button class. Use the following line to create button object.

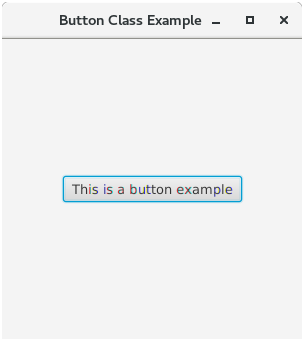
1. Button btn = new Button("My Button");

**Adding a Button to the scene graph**

To visualize the button on the screen, we must attach it to the scene object. The following code creates a button and adds it to the scene object.

1. package application;
2. import javafx.application.Application;
3. import javafx.scene.Scene;
4. import javafx.scene.control.Button;
5. import javafx.scene.layout.StackPane;
6. import javafx.stage.Stage;
8. public class ButtonTest extends Application {
10. @Override
11. public void start(Stage primaryStage) throws Exception {
12. // TODO Auto-generated method stub
14. StackPane root = new StackPane();
15. Button btn=new Button("This is a button");
16. Scene scene=new Scene(root,300,300);
17. root.getChildren().add(btn);
18. primaryStage.setScene(scene);
19. primaryStage.setTitle("Button Class Example");
20. primaryStage.show();
22. }
23. public static void main(String[] args) {
24. launch(args);
25. }
26. }

**Output:**



**Setting the Text of the Button**

There are two ways of setting the text on the button.

1. Passing the text into the class constructor
2. By calling setText("text") method

**Wrapping Button Text**

We can wrap the text of the button into multiple lines if the text to be displayed is too long. This can be done by calling a setter method **setWrapText(boolean)** on the instance of Button class. Pass the boolean value **true** in the method wherever required.

1. Btn.setWrapText(true);

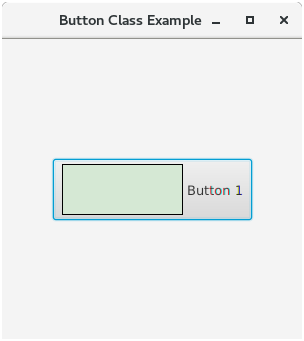
**Setting the image on the button**

Button class contains a constructor which can accept graphics along with the text displayed on the button. The following code implements image on the button.

1. package application;
3. import java.io.FileInputStream;
5. import javafx.application.Application;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Button;
8. import javafx.scene.image.Image;
9. import javafx.scene.image.ImageView;
10. import javafx.scene.layout.StackPane;
11. import javafx.stage.Stage;
13. public class ButtonTest extends Application {
15. @Override
16. public void start(Stage primaryStage) throws Exception {
17. // TODO Auto-generated method stub

20. FileInputStream input=new FileInputStream("/home/javatpoint/Desktop/JavaFX/Images/colored\_label.png");
21. Image image = new Image(input);
22. ImageView img=new ImageView(image);
24. StackPane root = new StackPane();
25. Button btn=new Button("Button 1",img);
26. btn.setWrapText(true);
27. Scene scene=new Scene(root,300,300);
28. root.getChildren().add(btn);
29. primaryStage.setScene(scene);
30. primaryStage.setTitle("Button Class Example");
31. primaryStage.show();
33. }
34. public static void main(String[] args) {
35. launch(args);
36. }
37. }

**Output:**



**JavaFX Menu**

JavaFX provides a Menu class to implement menus. Menu is the main component of a any application. In JavaFX, **javafx.scene.control**.Menu class provides all the methods to deal with menus. This class needs to be instantiated to create a Menu.

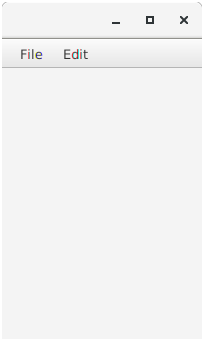
The following sample of code shows the implementation of JavaFX menu.

1. ManuBar menubar = new MenuBar(); //creating MenuBar
2. Menu MenuName = new Menu("Menu Name"); //creating Menu
3. MenuItem MenuItem1 = new MenuItem("Menu Item 1 Name"); //creating Menu Item
4. MenuName.getItems().add(MenuItem1); //adding Menu Item to the Menu
5. menubar.getMenus().add(MenuName); //adding Menu to the MenuBar

**EXAMPLE:**

1. package application;
2. import javafx.application.Application;
3. import javafx.scene.Scene;
4. import javafx.scene.control.\*;
5. import javafx.scene.layout.BorderPane;
6. import javafx.stage.Stage;
7. public class MenuExample extends Application {
8. public static void main(String[] args) {
9. launch(args);
10. }
12. @Override
13. public void start(Stage primaryStage) throws Exception {
14. // TODO Auto-generated method stub
15. BorderPane root = new BorderPane();
16. Scene scene = new Scene(root,200,300);
17. MenuBar menubar = new MenuBar();
18. Menu FileMenu = new Menu("File");
19. MenuItem filemenu1=new MenuItem("new");
20. MenuItem filemenu2=new MenuItem("Save");
21. MenuItem filemenu3=new MenuItem("Exit");
22. Menu EditMenu=new Menu("Edit");
23. MenuItem EditMenu1=new MenuItem("Cut");
24. MenuItem EditMenu2=new MenuItem("Copy");
25. MenuItem EditMenu3=new MenuItem("Paste");
26. EditMenu.getItems().addAll(EditMenu1,EditMenu2,EditMenu3);
27. root.setTop(menubar);
28. FileMenu.getItems().addAll(filemenu1,filemenu2,filemenu3);
29. menubar.getMenus().addAll(FileMenu,EditMenu);
30. primaryStage.setScene(scene);
31. primaryStage.show();
33. }
34. }

**Output:**



**JavaFX 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. This part of the tutorial This part of the tutorial describes the way in which the events are handled in JavaFX.

**Types of Events**

In general, the events are mainly classified into the following two types.

**1. Foreground Events**

Foreground events are mainly occurred due to the direct interaction of the user with the GUI of the application. Such as clicking the button, pressing a key, selecting an item from the list, scrolling the page, etc.

**2. Background Events**

Background events doesn't require the user's interaction with the application. These events are mainly occurred to the operating system interrupts, failure, operation completion, etc.

**Processing Events in JavaFX**

In JavaFX, 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 the class **javafx.event.Event** which contains all the subclasses representing the types of Events that can be generated in JavaFX. Any event is an instance of the class **Event** or any of its subclasses.

There are various events in JavaFX i.e. MouseEvent, KeyEvent, ScrollEvent, DragEvent, etc. We can also define our own event by inheriting the class **javafx.event.Event**.

The properties of an event is described in the following table.

|  |  |  |
| --- | --- | --- |
| **SN** | **Property** | **Description** |
| 1 | Event Type | It is the type of the event that is being generated. It is basically the instance of EventType class. It is hierarchical. The instance of EventType class is further classified into various type of events for example KeyEvent class contains KEY\_PRESSED, KEY\_RELEASED, and KEY\_TYPED types. |
| 2 | Source | It represents source of the event i.e. the origin which is responsible to generate the event. |
| 3 | Target | It is the node on which the event is generated. It remains unchanged for the generated event. It is the instance of any of the class that implements the EventTarget interface. |

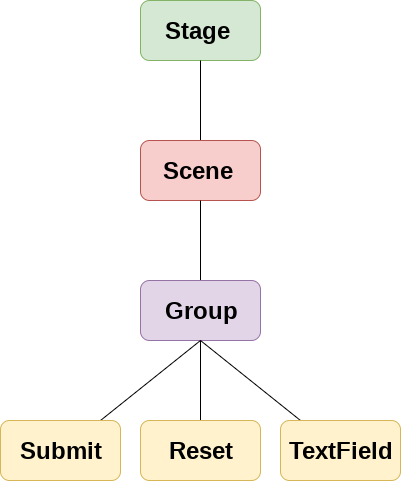
**Event Delivery Process**

The following steps need to be followed in order to handle the events.

**1. Route Construction**

An Event Dispatch Chain is created in order to determine the default route of the event, whenever it is generated. The Event dispatch chain contains the path from the stage to the Node on which the event is generated.

An event dispatch chain is created in the following image for the event generated on one of the scene graph node.



**2. 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.

**3. Event Bubbling**

Once the event is processed by the target node or by any of the registered filter, the event traverses all the nodes again from the bottom to the stage node. If any of these nodes are registered with the event filter, then it will get executed otherwise the process gets completed.

**4. 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.

**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.

1. node.addEventHandler(<EventType>,new EventHandler<Event-Type>()
2. {
3. public void handle(<EventType> e)
4. {
5. //handling code
6. }
7. });

**Convenience Methods**

JavaFX provides the convenience methods which can be used to handle events within our JavaFX application. They provide an easy way to create and register event handlers to respond to KeyEvent, MouseEvent, Action Event, Drag & Drop Events and many more.

Node class contains various Event Handler properties which can be set to the User defined Event Handlers using the setter methods defined in the class.

Setting, EventHandler properties of the Node class, to the user defined event handlers, automatically registers the handlers to receive the corresponding event types.

The EventHandler properties along with their setter methods (convenience methods) are described in the following table.

|  |  |  |
| --- | --- | --- |
| **EventHandler Property** | **Description** | **Setter Methods** |
| onContextMenuRequested | This is of the type EventHandler of ContextMenuEvent. This is assigned to a function which is to be called when the context menu is requested on the node. | setOnContextMenuRequested(EventHandler value ) |
| 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 ) |
| onDragEntered | This is of the type EventHandler of DragEvent. This is assigned to a function which is to be called when the drag gesture enters the node. | setOnDragEntered(EventHandler value ) |
| onDragExited | This is of the type EventHandler of DragEvent. This is assigned to a function which is to be called when the drag gesture exits the node. | setOnDragExited(EventHandler value ) |
| onDragOver | This is of the type EventHandler of DragEvent.This is assigned to a function which is to be called when the drag gesture progresses within the node. | setOnDragOver(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 ) |
| 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 ) |
| onRotate | This is of the type EventHandler of RotateEvent. This is assigned to a function which is to be called when the rotation action is performed on the node. | setOnRotate(EventHandler value ) |
| onRotationFinished | This is of the type EventHandler of RotateEvent. This is assigned to a function which is to be called when the rotation gesture ends. | setOnRotationFinished(EventHandler value ) |
| onRotationStarted | This is of the type EventHandler of RotateEvent. This is assigned to a function which is to be called when the rotation gesture is first detected. | setOnRotationStarted(EventHandler value ) |
| onScrollFinished | This is of the type EventHandler of ScrollEvent. This is assigned to a function which is to be called when the scroll gesture ends. | setOnScrollFinished(EventHandler value ) |
| onScroll | This is of the type EventHandler of ScrollEvent. This is assigned to a function which is to be called when the scroll action is performed. | setOnScroll(EventHandler value ) |
| onScrollStarted | This is of the type EventHandler of ScrollEvent. This is assigned to a function which is to be called when the scrolling gesture is detected. | setOnScrollStarted(EventHandler value ) |
| onSwipeDown | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the downwards swipe happens on the node. | setOnSwipeDown(EventHandler value ) |
| onSwipeUP | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the upwards swipe happens on the node. | setOnSwipeUP(EventHandler value ) |
| onSwipeLeft | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the leftwards swipe happens on the node. | setOnSwipeLeft(EventHandler value ) |
| onSwipeRight | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the Rightwards swipe happens on the node. | setOnSwipeRight(EventHandler value ) |
| onTouchMoved | This is of the type EventHandler of TouchEvent. This is assigned to a function which is to be called when the touch point is moved on the node. | setOnTouchMoved(EventHandler value ) |
| onTouchReleased | This is of the type EventHandler of TouchEvent. This is assigned to a function which is to be called when the touch point is released on the node. | setOnTouchReleased(EventHandler value ) |
| onTouchStationary | This is of the type EventHandler of TouchEvent. This is assigned to a function which is to be called when the touch point is pressed and stays still | setOnTouchStationary(EventHandler value ) |
| onZoomFinished | This is of the type EventHandler of ZoomEvent. This is assigned to a function which is to be called when the zooming gesture ends. | setOnZoomFinished(EventHandler value ) |
| onZoom | This is of the type EventHandler of ZoomEvent. This is assigned to a function which is to be called when the zooming gesture is performed. | setOnZoom(EventHandler value ) |
| onZoomStarted | This is of the type EventHandler of ZoomEvent. This is assigned to a function which is to be called when the zooming gesture is detected. | setOnZoomStarted(EventHandler value ) |

Convenience methods for the event handlers registration has the format like

1. setOnEvent-type(EventHandler<? super event-class> value)

where the Event type is the type of the event that is to be handled through the defined functions, for example, **setOnMouseMoved()** will be the convenience method to register the event handler for the event Mouse\_Moved.

**setOnAction() Example for a Button Action**

In the following example, **setOnAction()** method is illustrated. The EventHandler registered with the setOnAction() method is called when the Play button is clicked and it is set to rotate the rectangle on the screen.

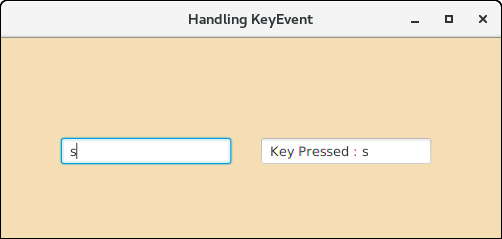
Pause button is also registered with the EventHandler which is set to stop the rotation of rectangle.

1. package application;
2. import javafx.animation.RotateTransition;
3. import javafx.application.Application;
4. import javafx.event.ActionEvent;
5. import javafx.event.EventHandler;
6. import javafx.scene.Group;
7. import javafx.scene.Scene;
8. import javafx.scene.control.Button;
9. import javafx.scene.paint.Color;
10. import javafx.scene.shape.Rectangle;
11. import javafx.stage.Stage;
12. import javafx.util.Duration;
14. public class EventHandlerExample extends Application {
15. public static void main(String[] args) {
16. Application.launch(args);
17. }
19. @Override
20. public void start(Stage primaryStage) {
22. // Creating Rectangle
23. Rectangle rect = new Rectangle(100,100,120,120);
25. // Setting Stroke and colour for the rectangle
26. rect.setFill(Color.RED);
27. rect.setStroke(Color.BLACK);
29. // Instantiating RotateTransition class
30. RotateTransition rotate = new RotateTransition();
32. //Setting properties for the Rotate Transition class
33. rotate.setAutoReverse(false);
34. rotate.setByAngle(360);
35. rotate.setCycleCount(500);
36. rotate.setDuration(Duration.millis(500));
37. rotate.setNode(rect);
39. //Creating the play button
40. Button btn = new Button();
42. //Setting properties for the play button
43. btn.setText("Play");
44. btn.setTranslateX(100);
45. btn.setTranslateY(250);
47. //defining the convenience method to register the event handler to handle the Action event.
48. btn.setOnAction(new EventHandler<ActionEvent>() {
49. public void handle(ActionEvent event) {
51. rotate.play();
52. }
53. });
55. //Creating the pause button
56. Button btn1 = new Button("Pause");
58. //Setting propertied for the pause button
59. btn1.setTranslateX(160);
60. btn1.setTranslateY(250);
62. //Handling event for the pause button click event
63. btn1.setOnAction(new EventHandler<ActionEvent>() {
65. @Override
66. public void handle(ActionEvent arg0) {
67. // TODO Auto-generated method stub
68. rotate.pause();
69. }
71. });
73. //Configuring group and scene
74. Group root = new Group();
75. Scene scene = new Scene(root, 400, 350);
76. root.getChildren().addAll(btn,rect,btn1);
77. primaryStage.setScene(scene);
78. primaryStage.setTitle("Handling Events");
79. primaryStage.show();
80. }
81. }

**setOnKeyEvent() Example for a Key Event**

The setOnKeyEvent() method can be used for registering the Event Handler logic for the key event generated on a node. In the following example, two text fields are created as the node, the key pressed in the first text field is set as the text in the second text field.

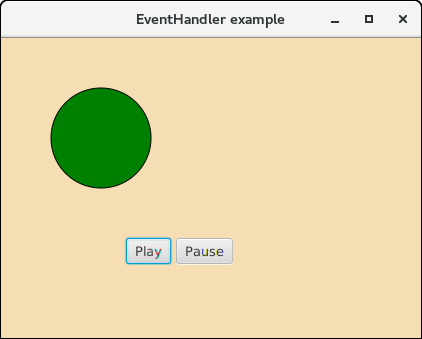
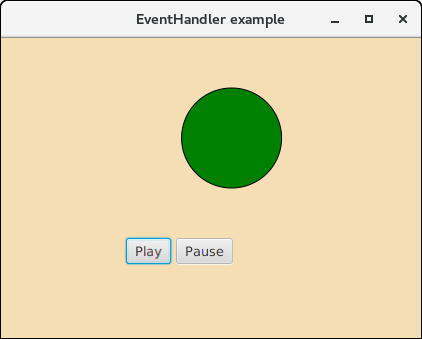
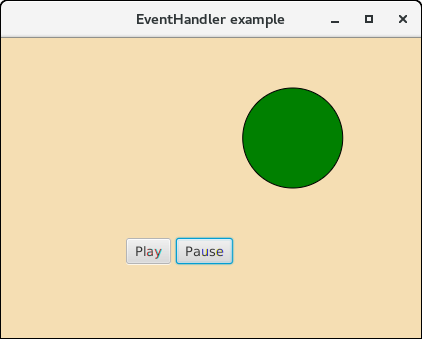
1. package application;
2. import javafx.application.Application;
3. import javafx.event.EventHandler;
4. import javafx.scene.Group;
5. import javafx.scene.Scene;
6. import javafx.scene.control.TextField;
7. import javafx.scene.input.KeyEvent;
8. import javafx.scene.paint.Color;
9. import javafx.stage.Stage;
10. public class JavaFX\_KeyEvent extends Application{
12. @Override
13. public void start(Stage primaryStage) throws Exception {
15. // TODO Auto-generated method stub
17. //Creating TextFields and setting position for them
18. TextField tf1 = new TextField();
19. TextField tf2 = new TextField();
20. tf1.setTranslateX(100);
21. tf1.setTranslateY(100);
22. tf2.setTranslateX(300);
23. tf2.setTranslateY(100);
25. //Handling KeyEvent for textfield 1
26. tf1.setOnKeyPressed(new EventHandler<KeyEvent>() {
28. @Override
29. public void handle(KeyEvent key) {
30. // TODO Auto-generated method stub
31. tf2.setText("Key Pressed :"+" "+key.getText());
32. }
34. });
36. //setting group and scene
37. Group root = new Group();
38. root.getChildren().addAll(tf2,tf1);
39. Scene scene = new Scene(root,500,200,Color.WHEAT);
40. primaryStage.setScene(scene);
41. primaryStage.setTitle("Handling KeyEvent");
42. primaryStage.show();
43. }
44. public static void main(String[] args) {
45. launch(args);
46. }
48. }



**Example**

In the following example, same event handler is registered with two different buttons. The event source is discriminated in the **handle()** method. The circle starts translating in the positive X direction when the **Play** button is clicked while It gets paused when the **Pause** button is clicked.

1. package application;
2. import javafx.animation.TranslateTransition;
3. import javafx.application.Application;
4. import javafx.event.EventHandler;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Button;
8. import javafx.scene.input.MouseEvent;
9. import javafx.scene.paint.Color;
10. import javafx.scene.shape.Circle;
11. import javafx.stage.Stage;
12. import javafx.util.Duration;
13. public class JavaFX\_EventHandler extends Application{
14. @Override
15. public void start(Stage primaryStage) throws Exception {
16. // TODO Auto-generated method stub
17. //Creating Circle and setting the color and stroke in the circle
18. Circle c = new Circle(100,100,50);
19. c.setFill(Color.GREEN);
20. c.setStroke(Color.BLACK);
22. //creating play button and setting coordinates for the button
23. Button btn = new Button("Play");
24. btn.setTranslateX(125);
25. btn.setTranslateY(200);
27. // creating pause button and setting coordinate for the pause button
28. Button btn1 = new Button("Pause");
29. btn1.setTranslateX(175);
30. btn1.setTranslateY(200);
32. //Instantiating TranslateTransition class to create the animation
33. TranslateTransition trans = new TranslateTransition();
35. //setting attributes for the TranslateTransition
36. trans.setAutoReverse(true);
37. trans.setByX(200);
38. trans.setCycleCount(100);
39. trans.setDuration(Duration.millis(500));
40. trans.setNode(c);
42. //Creating EventHandler
43. EventHandler<MouseEvent> handler = new EventHandler<MouseEvent>() {
45. @Override
46. public void handle(MouseEvent event) {
47. // TODO Auto-generated method stub
49. if(event.getSource()==btn)
50. {
51. trans.play(); //animation will be played when the play button is clicked
52. }
53. if(event.getSource()==btn1)
54. {
55. trans.pause(); //animation will be paused when the pause button is clicked
56. }
57. event.consume();
58. }
60. };
62. //Adding Handler for the play and pause button
63. btn.setOnMouseClicked(handler);
64. btn1.setOnMouseClicked(handler);
66. //Creating Group and scene
67. Group root = new Group();
68. root.getChildren().addAll(c,btn,btn1);
69. Scene scene = new Scene(root,420,300,Color.WHEAT);
70. primaryStage.setScene(scene);
71. primaryStage.setTitle("EventHandler example");
72. primaryStage.show();
73. }
74. public static void main(String[] args) {
75. launch(args);
76. }
77. }

**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.

1. node.removeEventHandler(<EventType>,handler);