CSCI 2120:
Software Design & Development II

UNIT4: UI management

GUI framework

JavaFX: Event Handling

Overview

1. Introduction
2. What is an Event?
3. Event Hierarchy
4. Basic Types of Events
5. Events in JavaFX
6. Event Handling
7. Phases of Event Handling/Filtering
8. Event Handling vs Event Filtering
9. Using Anonymous classes vs. Lambda Expressions
10. Adding & Removing Event Filters/Handlers
11. Convenience Methods
12. Specific Event Types

Introduction

- In JavaFX, we can develop GUI applications, web applications and graphical applications. In such applications, whenever a user interacts with the application (nodes), an event is said to have been occurred.

- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

- In this lecture, we will explore event-driven programming.

What is an Event?

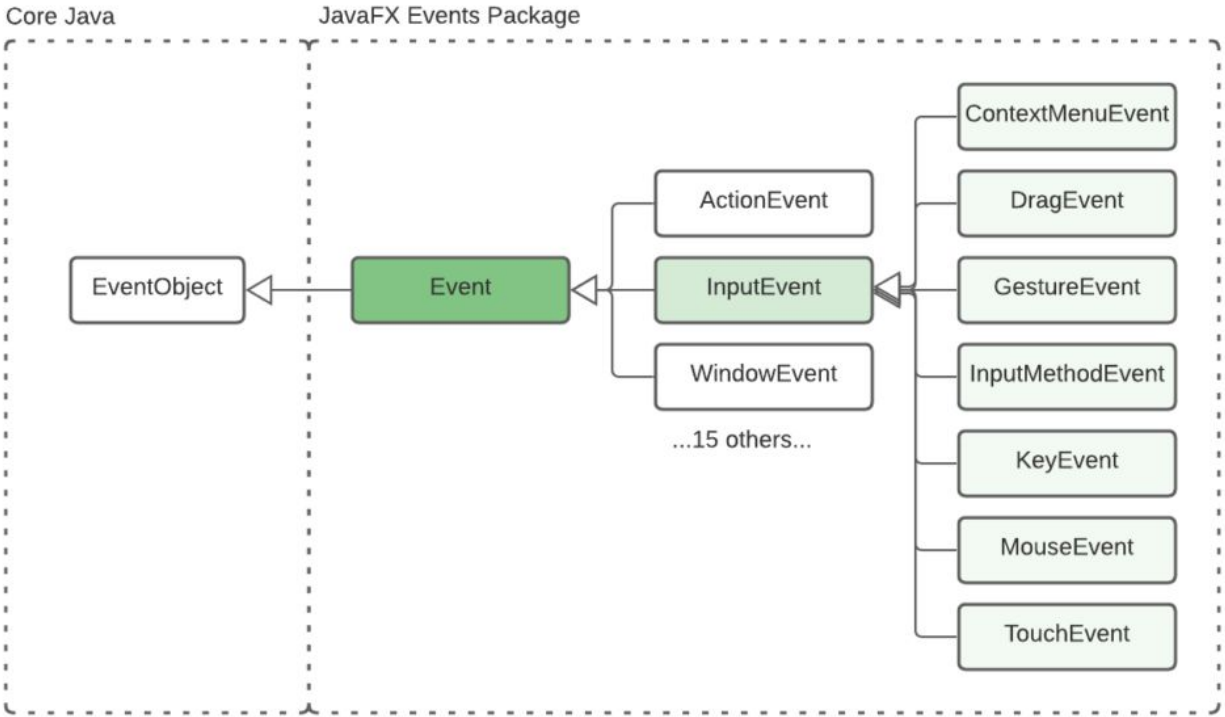
- An **Event** is any change of state in an input device (such as mouse or touch screen), a user action (**ActionEvent**), or a background task. Events can also be fired as a result of scroll or edit events on complex nodes such as **TableView** and **ListView.**

- At a very basic level, the **Event** object is a class with a surprisingly small number of parameters. It doesn't hold any executable code, nor does it run any code. It's a flag to the system that something has changed.

- JavaFX then provides significant support to run executable code, defined separately through the **EventHandler** class, on any changes of state.

- There are 90 separate types of event supported by JavaFX, with the capability to extend the **Event** class and define additional custom functionality.

Event Hierarchy



Categories of Events

The events can be broadly classified into the following two categories −

- **Foreground Events**

  Those events which require the direct interaction of a user. They are generated as consequences of a person interacting with the graphical components in a Graphical User Interface.

  - *Examples: clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.*

- **Background Events**

  Those events that don't require the interaction of end-user are known as background events.

  - *Examples: The operating system interruptions, hardware or software failure, timer expiry, operation completion.*

Events in JavaFX

JavaFX provides support to handle many types of events. The abstract class named `Event` of the package `javafx.event` is the base class for an event. There are many subclasses that extend and instantiate `Event.`

Type	Description
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.

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 −

Field	Description
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. For instance, with a click the 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.
Consumed	At any point in this process, including before an event has reached its target, the event can be consumed and the process immediately stops.

Event Handling

Assume that we have an application which has a Circle, Stop and Play Buttons inserted using a group object as follows −

If you click on the play button, the ***source*** will be the ***mouse,*** the ***target*** node will be the ***play button*** and the ***type*** of the event generated is the ***mouse click***.

Phases of Event Handling

Whenever an event is generated, JavaFX undergoes the following phases.

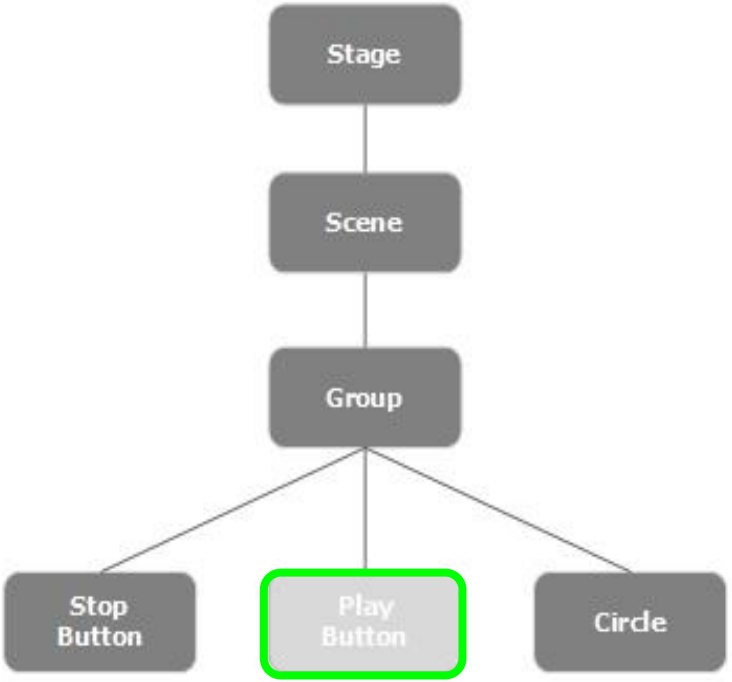
1. Target Selection
2. Route Construction
3. Event Capturing Phase
4. Event Bubbling Phase
5. Event Handlers and Filters

Phases of Event Handling - Target Selection

- The ***source*** node that generated the ***event***
- Say that your scene contains a group with a button in it. If you click on the button, the button becomes the event target.

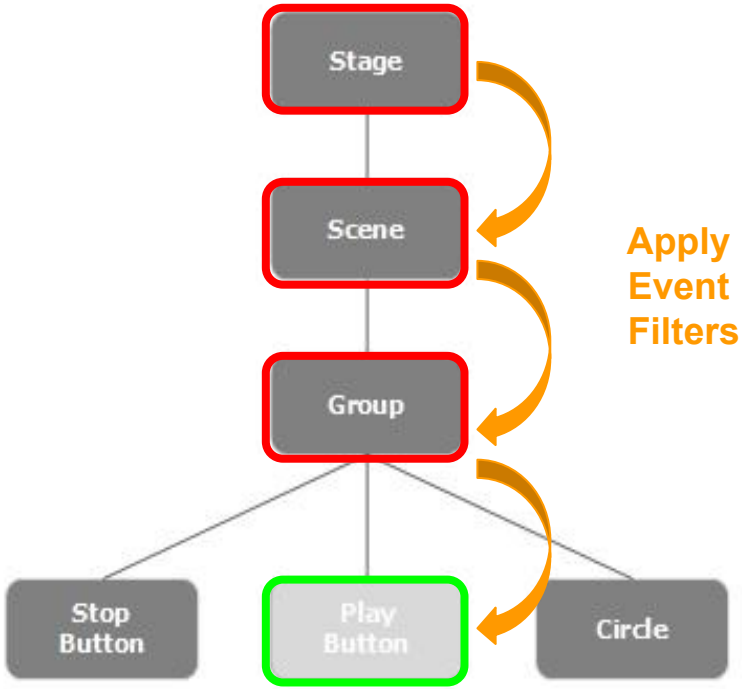
Phases of Event Handling - Route Construction

- 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.
- Below is the event dispatch chain for the event generated, when we click on the play button in the example scenario.



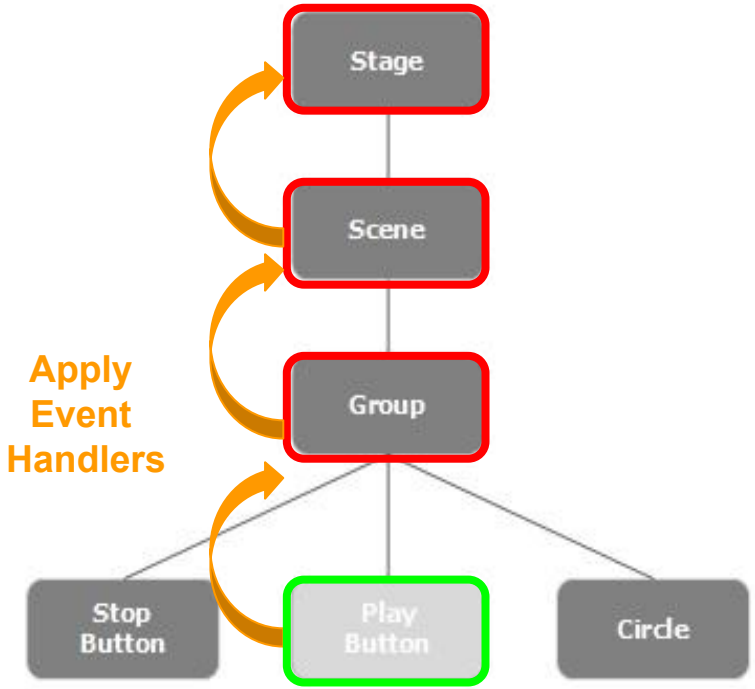
Phases of Event Handling - Event Capturing Phase

- After the construction of the event dispatch chain, the root node of the application dispatches the event.
- This event travels to all nodes in the dispatch chain (from top to bottom).
- If any of these nodes has a filter registered for the generated event, it will be executed.
- If none of the nodes in the dispatch chain has a filter for the event generated, then it is passed to the target node and finally the target node processes the event.



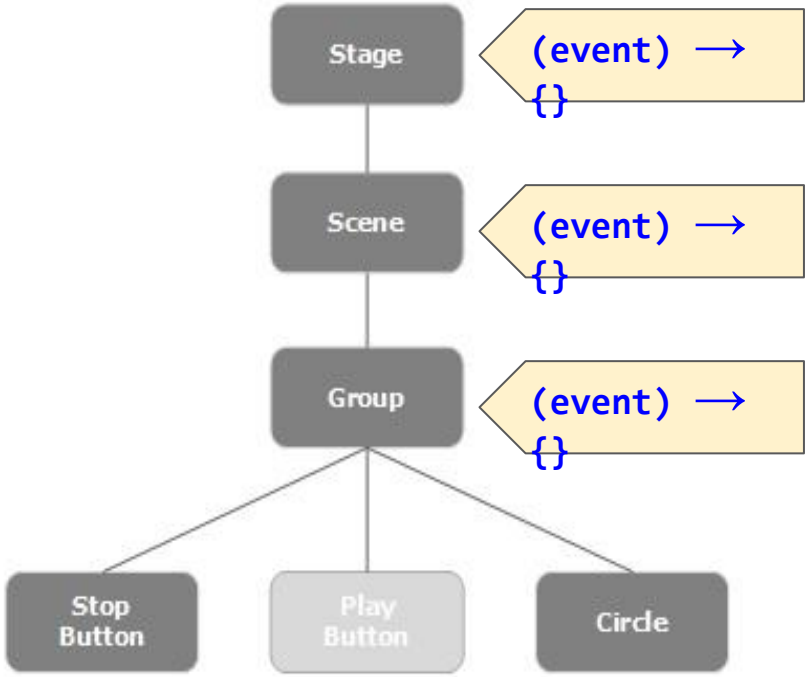
Phases of Event Handling - Event Bubbling Phase

- 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.



Phases of Event Handling - Event Handlers and Filters

- Event filters and handlers are those which contains application logic to process an event.
- A node can register to more than one handler/filter. 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.
- As mentioned earlier, during the event, processing is a filter that is executed during the event bubbling phase, a handler is executed.
- All the handlers and filters implement the interface EventHandler of the package javafx.event.



Phases of Event Handling - Filters vs Handlers

Whenever an event happens, it follows a process to determine which node in the scene graph should handle the event.
The process takes these steps:

- Target selection
- Route construction
- Event capturing ← **filters are triggered here**
- Event bubbling ← **handlers are triggered here**

Target Selection
Say that your scene contains a pane with a circle. If you click on the circle, the circle becomes the event target.

Route Construction
Next, JavaFX creates a route (or an event dispatch chain). In our example the chain would look like stage → scene → pane → circle

Event Capturing
The event gets carried through every event filter on the chain. As soon as one of the filters calls consume(), the chain stops and that node becomes the target. If no filter calls consume() the end of the chain (the circle) remains the target.

Event Bubbling
Next, the event get pushed through the chain again, but this time from the event target to the stage. So if the pane event filter called consume(), the following event handlers will be hit: pane → scene → stage

So the difference is not only when these handlers get activated, but also that event filters can prevent child nodes from receiving events.

Event Handling: *Anonymous class vs Lambda Expression*

Defining an EventHandler

The `EventHandler` object is an object with a single method, which we need to override when we instantiate it. As the EventHandler interface is **parameterized**, we need to specify what event type we're handling. In this case we'll use a MouseEvent.

Raw Copy

```
EventHandler<MouseEvent> eventHandler = new EventHandler<>() {
    @Override
    public void handle(MouseEvent event) {
        //executable code
    }
};
```

Event Handling: *Anonymous class vs Lambda Expression*

Replacing the definition with a lambda

An `EventHandler` is a functional interface (an interface with a single method). For convenience, the `handle()` method of the `EventHandler` interface can be specified with a lambda instead.

```
EventHandler<MouseEvent> eventHandler = event -> {
    //executable code
};
```

Event Handling: *Anonymous class vs Lambda Expression*

Anonymous Class

```
//Creating the event handler
EventHandler<Event> eventHandler = new EventHandler<Event>() {
   @Override
   public void handle(Event e) {
      //do stuff
   }
};
```

Lambda Expression

```
//Creating the event handler
EventHandler<Event> eventHandler = (event) -> {
   //do stuff
};
```

Event Filters/Handlers listen for Event Types

🔗 Adding and removing event filters and handlers

When we define and add an event handler to a node (or task), we define what *type* of that event we want our code to trigger from. As an example, if we want our code to trigger from *any* mouse event on the root node of our scene, we can use the event type `MouseEvent.ANY`.

If you want to remove an event handler at a later point in code, you will need to keep a reference to the `EventHandler` object you create.

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);
```

Event Handling Example - imports

Here is an example demonstrating the event handling in JavaFX using the event filters.

```
import static javafx.application.Application.launch;
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.input.MouseEvent;

import javafx.scene.Group;
import javafx.scene.Scene;
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;
```

Event Handling Example - class declaration - stubbed

```
public class EventFiltersExample extends Application {

   private Text setupText() { }
   private Circle setupCircle() { }
   private EventHandler<MouseEvent> getMouseHandler(Circle circle) { }
   private Scene setupScene(Circle circle, Text text) { }

   @Override
   public void start(Stage stage) { }

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

Event Handling Example - setupText()

```
private Text setupText(){
        Text text = new Text("Click circle to change color");          //Set the text
        text.setFont(Font.font(null, FontWeight.BOLD, 15));            //Set the font of the text
        text.setFill(Color.CRIMSON);                                   //Set the color of the text
        text.setX(150);                                                //Set X position of the text
        text.setY(50);                                                 //Set Y position of the text
        return text;
}
```

```
@Override
public void start(Stage stage) {
        Text text = setupText();
}
```

Event Handling Example - setupCircle()

```
private Circle setupCircle(){
        Circle circle = new Circle();                    //Make a Circle
        circle.setCenterX(300.0f);                       //Set X position of the circle
        circle.setCenterY(135.0f);                       //Set Y position of the circle
        circle.setRadius(25.0f);                         //Set radius of the circle
        circle.setFill(Color.BROWN);                     //Set the color of the circle
        circle.setStrokeWidth(20);                       //Set the stroke width of the circle
        return circle;
}
```

```
@Override
public void start(Stage stage) {
        Text text = setupText();
        Circle circle = setupCircle();
}
```

Event Handling Example - getMouseHandler()

```
private EventHandler<MouseEvent> getMouseHandler(Circle circle){
        //Creating the mouse event handler
        return (event) -> {
            System.out.println("Hello World");
            circle.setFill(Color.DARKSLATEBLUE);
        };
}
```

```
@Override
public void start(Stage stage) {
        Text text = setupText();
        Circle circle = setupCircle();
        EventHandler<MouseEvent> eventHandler = getMouseHandler(circle);      //Creating the mouse event handler
        circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);        //Registering the event filter
}
```

Event Handling Example - setupScene()

```
private Scene setupScene(Circle circle, Text text){
        Group root = new Group(circle, text);                    //Creating a Group object
        Scene scene = new Scene(root, 600, 300);                 //Creating a scene object
        scene.setFill(Color.LAVENDER);                           //Setting the fill color to the scene
        return scene;
}
```

```
@Override
public void start(Stage stage) {
        Text text = setupText();
        Circle circle = setupCircle();
        EventHandler<MouseEvent> eventHandler = getMouseHandler(circle);      //Creating the mouse event handler
        circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);        //Registering the event filter
        Scene scene = setupScene(circle, text);
}
```

Event Handling Example - start()

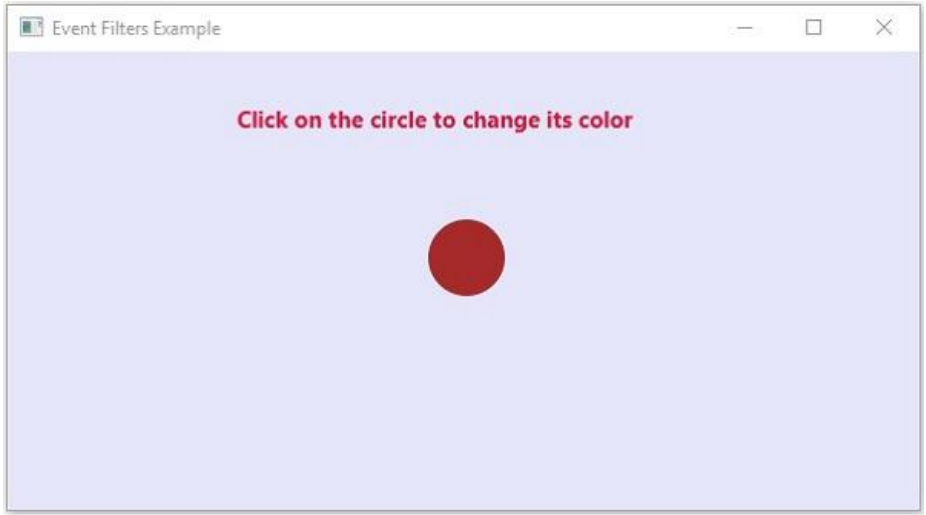
```
@Override
public void start(Stage stage) {
        Text text = setupText();
        Circle circle = setupCircle();
        EventHandler<MouseEvent> eventHandler = getMouseHandler(circle);      //Creating the mouse event handler
        circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);        //Registering the event filter
        Scene scene = setupScene(circle, text);
        stage.setTitle("Event Filters Example");                              //Setting title to the Stage
        stage.setScene(scene);                                                //Adding scene to the stage
        stage.show();                                                         //Displaying the contents of the stage
}
```

Event Handling Example - Compile & Execute

Compile and execute the saved java file from the command prompt using the following commands

```
javac EventFiltersExample.java
java EventFiltersExample
```

. On executing, the above program generates a JavaFX window as shown below.



Adding and Removing Event Handlers

To add an event handler to a node, you must register the handler using the method `addEventHandler()` of `Node` class as shown below.

```
//Creating the mouse event handler
EventHandler<javafx.scene.input.MouseEvent> eventHandler =
   new EventHandler<javafx.scene.input.MouseEvent>() {

   @Override
   public void handle(javafx.scene.input.MouseEvent e) {
      System.out.println("Hello World");
      circle.setFill(Color.DARKSLATEBLUE);
   }
};
//Adding the event handler
circle.addEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```

In the same way, you can remove an event handler using the method `removeEventHandler()` as shown below −

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

Event Handling Example - imports

Here is an example demonstrating the event handling in JavaFX using the event handlers.

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;

import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.Group;

import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;

import javafx.scene.control.TextField;
import javafx.scene.text.Text; ;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;

import javafx.scene.shape.Box
import javafx.scene.PerspectiveCamera;
import javafx.scene.transform.Rotate;
import javafx.animation.RotateTransition;
import javafx.util.Duration;
```

Event Handling Example - class declaration - stubbed

```
public class EventHandlersExample extends Application {

    private Box box;
    private Text text;
    private RotateTransition rotateTransition;
    private TextField textField;
    private Scene scene;

    private void setupBox(){ }
    private void setupText(){ }
    private void setupRotation(){ }
    private void setupTextField(){ }
    private void setupKeyHandler(){ }
    private void setupMouseHandler(){ }
    private void setupScene(){ }
    private void setupCamera(){ }

    @Override
    public void start(Stage stage) { }

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

Event Handling Example - setupBox()

```
private void setupBox(){
        //Drawing a Box
        box = new Box();

        //Setting the properties of the Box
        box.setWidth(150.0);
        box.setHeight(150.0);
        box.setDepth(100.0);

        //Setting the position of the box
        box.setTranslateX(350);
        box.setTranslateY(150);
        box.setTranslateZ(50);

        //Setting the material of the box
        PhongMaterial material = new PhongMaterial();
        material.setDiffuseColor(Color.DARKSLATEBLUE);

        //Setting the diffuse color material to box
        box.setMaterial(material);
}
```

Event Handling Example - setupText()

```
private void setupText(){
        //Setting the text
        text = new Text("Press a key to rotate box, click on box to stop");

        //Setting the font of the text
        text.setFont(Font.font(null, FontWeight.BOLD, 15));

        //Setting the color of the text
        text.setFill(Color.CRIMSON);

        //setting the position of the text
        text.setX(20);
        text.setY(50);
}
```

Event Handling Example - setupRotation()

```
private void setupRotation(){
        //Setting the rotation animation to the box
        rotateTransition = new RotateTransition();

        //Setting the duration for the transition
        rotateTransition.setDuration(Duration.millis(1000));

        //Setting the node for the transition
        rotateTransition.setNode(box);

        //Setting the axis of the rotation
        rotateTransition.setAxis(Rotate.Y_AXIS);

        //Setting the angle of the rotation
        rotateTransition.setByAngle(360);

        //Setting the cycle count for the transition
        rotateTransition.setCycleCount(50);

        //Setting auto reverse value to false
        rotateTransition.setAutoReverse(false);
}
```

Event Handling Example - setupTextField()

```
private void setupTextField(){
        //Creating a text field
        textField = new TextField();

        //Setting the position of the text field
        textField.setLayoutX(50);
        textField.setLayoutY(100);
}
```

Event Handling Example - setupKeyHandler()

```
private void setupKeyHandler(){
   //Handling the key typed event
   EventHandler<KeyEvent> eventHandlerTextField = (event) -> {
       //Playing the animation
       rotateTransition.play();
   };

   //Adding an event handler to the text field
   textField.addEventHandler(KeyEvent.KEY_TYPED, eventHandlerTextField);
}
```

Event Handling Example - setupMouseHandler()

```
private void setupMouseHandler(){
   //Handling the mouse clicked event(on box)
   EventHandler<MouseEvent> eventHandlerBox = (event) -> {
       rotateTransition.stop();
   };

   //Adding the event handler to the box
   box.addEventHandler(MouseEvent.MOUSE_CLICKED, eventHandlerBox);
}
```

Event Handling Example - setupScene()

```
private void setupScene(){
        //Creating a Group object
        Group root = new Group(box, textField, text);

        //Creating a scene object
        scene = new Scene(root, 600, 300);
}
```

Event Handling Example - setupCamera()

```
private void setupCamera(){
        //Setting camera
        PerspectiveCamera camera = new PerspectiveCamera(false);
        camera.setTranslateX(0);
        camera.setTranslateY(0);
        camera.setTranslateZ(0);
        scene.setCamera(camera);
}
```

Event Handling Example - start()

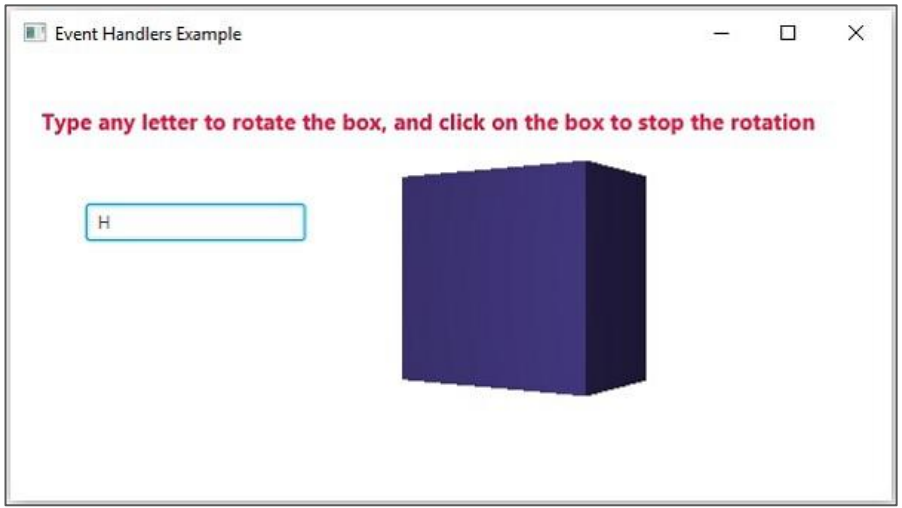
```
@Override
public void start(Stage stage) {
        setupBox();
        setupText();
        setupRotation();
        setupTextField();
        setupKeyHandler();
        setupMouseHandler();
        setupScene();
        setupCamera();

        stage.setTitle("Event Handlers Example");          //Setting title to the Stage
        stage.setScene(scene);                             //Adding scene to the stage
        stage.show();                                      //Displaying the contents of the stage
}
```

Event Handling Example - Compile & Execute

Compile and execute the saved java file from the command prompt using the following commands.

```
javac EventHandlersExample.java
java EventHandlersExample
```

On executing, the above program generates a JavaFX window displaying a text field and a 3D box as shown below –



If you type in the text field, the 3D box starts rotating along the x axis. If you click on the box again the rotation stops.

Using Convenience Methods for Event Handling

Sometimes it's expedient to use convenience methods like `setOnMousePressed()` to define event handlers. The syntax for event handlers is:

Here are a few examples:

- setOnMousePressed() (nodes)
- setOnKeyReleased() (nodes)
- setOnTaskRunning() (tasks)
- setOnScrollFinished() (nodes with virtual flow)

Using Convenience Methods for Event Handling

Convenience methods are useful because you can remove the handler without having to remember the reference by invoking the same method and passing a null reference:

```
setOnMousePressed(null)
```

They are also guaranteed to execute **after** any other event handlers of the **same type** on that node.

Using Convenience Methods for Event Handling

- 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 subclasses.

For example, to add a mouse event listener to a button, you can use the convenience method **setOnMouseClicked()** as shown below.

```
playButton.setOnMouseClicked( (event) -> {
   System.out.println("Hello World");
   pathTransition.play();
});
```

Convenience Methods Example - imports

The following program is an example that demonstrates the event handling in JavaFX using the convenience methods.

```
import javafx.animation.PathTransition;
import javafx.application.Application;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.paint.Color;

import javafx.scene.shape.Circle;
import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.stage.Stage;
import javafx.util.Duration;
```

Convenience Methods Example - class declaration

```
public class ConvinienceMethodsExample extends Application {
   private Circle setupCircle() { }
   private Path setupPath() { }
   private PathTransition setupPathTransition(Circle circle, Path path) { }
   private Button setupButton(String text, int x, int y) { }
   private void setupCircleHandler(Circle circle) { }
   private void setupPlayButtonHandler(Button playButton, PathTransition pathTransition) { }
   private void setupStopButtonHandler(Button stopButton, PathTransition pathTransition) { }
   private Scene setupScene(Node... nodes) { }

   @Override
   public void start(Stage stage) { }

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

Convenience Methods Example - setupCircle

```
private Circle setupCircle(){
	Circle circle = new Circle();		//Make a Circle
	circle.setCenterX(300.0f);		//Set X position of circle
	circle.setCenterY(135.0f);		//Set Y position of circle
	circle.setRadius(25.0f);		//Set radius of circle
	circle.setFill(Color.BROWN);		//Set color of circle
	circle.setStrokeWidth(20);		//Set stroke width of circle
	return circle;
}
```

```
@Override
public void start(Stage stage) {
	Circle circle = setupCircle();

}
```

Convenience Methods Example - setupCircleHandler

```
private void setupCircleHandler(Circle circle) {
       circle.setOnMouseClicked ( (event) -> {
           System.out.println("Hello World");
           circle.setFill(Color.DARKSLATEBLUE);
       });
}
```

```
@Override
public void start(Stage stage) {
       Circle circle = setupCircle();
       setupCircleHandler(circle);
}
```

Convenience Methods Example - setupPath

```
private Path setupPath(){
        Path path = new Path();                                          //Make a Path
        MoveTo moveTo = new MoveTo(208, 71);                             //Moving to the starting point
        LineTo line1 = new LineTo(421, 161);                             //Creating 1st line
        LineTo line2 = new LineTo(226,232);                              //Creating 2nd line
        LineTo line3 = new LineTo(332,52);                               //Creating 3rd line
        LineTo line4 = new LineTo(369, 250);                             //Creating 4th line
        LineTo line5 = new LineTo(208, 71);                              //Creating 5th line
        path.getElements().add(moveTo);                                  //Adding all elements to path
        path.getElements().addAll(line1, line2, line3, line4, line5);
        return path;
}
```

```
@Override
public void start(Stage stage) {
        Circle circle = setupCircle();
        setupCircleHandler(circle);
        Path path = setupPath();
}
```

Convenience Methods Example - setupPathTransition

```
private PathTransition setupPathTransition(Circle circle, Path path){
   PathTransition pathTransition = new PathTransition();                                    //Make the path transition
   pathTransition.setDuration(Duration.millis(1000));                                       //Set duration of transition
   pathTransition.setNode(circle);                                                          //Set node for transition
   pathTransition.setPath(path);                                                            //Set path for transition
   pathTransition.setOrientation(PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT); //Set orientation of path
   pathTransition.setCycleCount(50);                                                        //Set cycle count
   pathTransition.setAutoReverse(false);                                                    //Set auto reverse to true
   return pathTransition;
}
```

```
@Override
public void start(Stage stage) {
       Circle circle = setupCircle();
       setupCircleHandler(circle);
       Path path = setupPath();
       PathTransition pathTransition = setupPathTransition(circle, path);
}
```

Convenience Methods Example - setupButton

```
private Button setupButton(String text, int x, int y){
        Button button = new Button(text);
        playButton.setLayoutX(x);
        playButton.setLayoutY(y);
        return button;
}
```

```
@Override
public void start(Stage stage) {
        Circle circle = setupCircle();
        setupCircleHandler(circle);
        Path path = setupPath();
        PathTransition pathTransition = setupPathTransition(circle, path);
        Button playButton = setupButton("Play",300,250);                    //Create play button
        Button stopButton = setupButton("Stop",250,250 );                   //Create stop button
}
```

Convenience Methods Example - setupPlayButtonHandler

```
private void setupPlayButtonHandler(Button playButton, PathTransition pathTransition){
	playButton.setOnMouseClicked( (event) -> {
		System.out.println("Hello World");
		pathTransition.play();
	});
}
```

```
@Override
public void start(Stage stage) {
	Circle circle = setupCircle();
	setupCircleHandler(circle);
	Path path = setupPath();
	PathTransition pathTransition = setupPathTransition(circle, path);
	Button playButton = setupButton("Play",300,250);                    //Create play button
	Button stopButton = setupButton("Stop",250,250 );                   //Create stop button
	setupPlayButtonHandler(playButton, pathTransition);
}
```

Convenience Methods Example - setupStopButtonHandler

```
private void setupStopButtonHandler(Button stopButton, PathTransition pathTransition){
	stopButton.setOnMouseClicked( (event) -> {
		System.out.println("Hello World");
		pathTransition.stop();
	});
}
```

```
@Override
public void start(Stage stage) {
	Circle circle = setupCircle();
	setupCircleHandler(circle);
	Path path = setupPath();
	PathTransition pathTransition = setupPathTransition(circle, path);
	Button playButton = setupButton("Play",300,250);                    //Create play button
	Button stopButton = setupButton("Stop",250,250 );                   //Create stop button
	setupPlayButtonHandler(playButton, pathTransition);
	setupStopButtonHandler(stopButton, pathTransition);
}
```

Convenience Methods Example - setupScene

```
private Scene setupScene(Node... nodes){
        //Creating a Group object
        Group root = new Group(nodes);

        //Creating a scene object
        Scene scene = new Scene(root, 600, 300);
        scene.setFill(Color.LAVENDER);
        return scene;
}
```

```
@Override
public void start(Stage stage) {
        Circle circle = setupCircle();
        setupCircleHandler(circle);
        Path path = setupPath();
        PathTransition pathTransition = setupPathTransition(circle, path);
        Button playButton = setupButton("Play",300,250);                    //Create play button
        Button stopButton = setupButton("Stop",250,250 );                   //Create stop button
        setupPlayButtonHandler(playButton, pathTransition);
        setupStopButtonHandler(stopButton, pathTransition);
        Scene scene = setupScene(circle, playButton, stopButton);
}
```

Convenience Methods Example - start

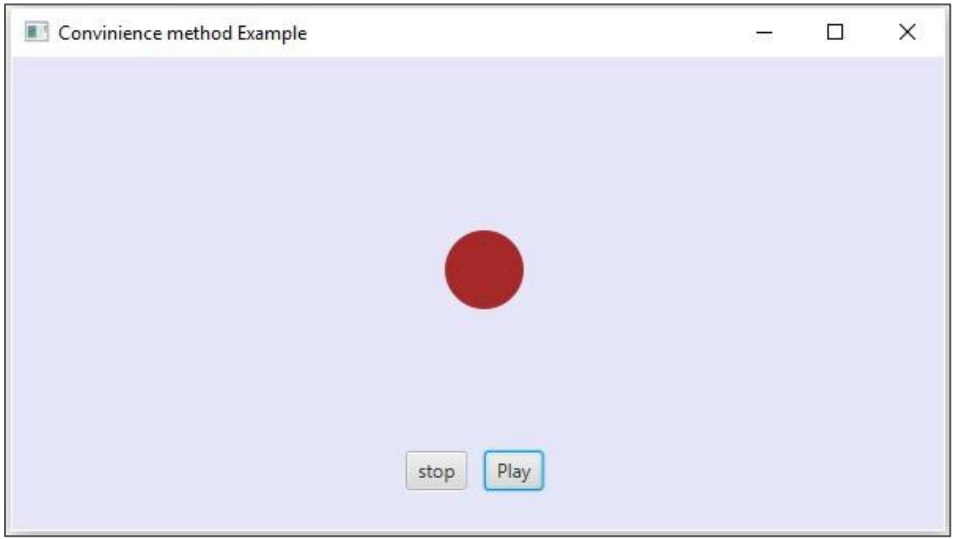
```
@Override
public void start(Stage stage) {
	Circle circle = setupCircle();
	setupCircleHandler(circle);
	Path path = setupPath();
	PathTransition pathTransition = setupPathTransition(circle, path);
	Button playButton = setupButton("Play",300,250);                    //Create play button
	setupPlayButtonHandler(playButton, pathTransition);
	Button stopButton = setupButton("Stop",250,250 );                   //Create stop button
	setupStopButtonHandler(stopButton, pathTransition);
	Scene scene = setupScene(circle, playButton, stopButton);
	stage.setTitle("Convenience Methods Example");                      //Set title to the Stage
	stage.setScene(scene);                                              //Add scene to the stage
	stage.show();                                                       //show GUI
}
```

Convenience Methods Example - Compile & Execute

Compile and execute the saved java file from the command prompt using the following commands.

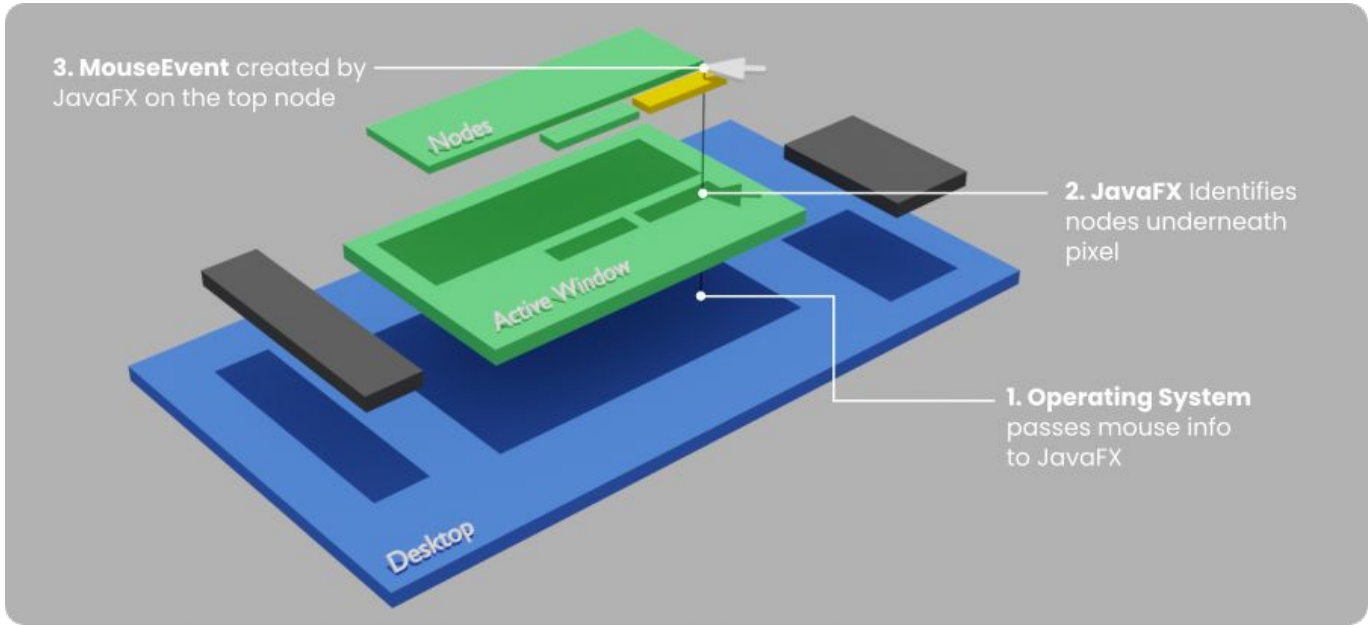
```
javac ConvinienceMethodsExample.java
java ConvinienceMethodsExample
```

On executing, the above program generates a JavaFX window as shown below. Here click on the play button to start the animation and click on the stop button to stop the animation.



Event Types - Operating System-Generated (Input) Events

Input events, such as mouse, keyboard and gesture events, happen above the level of the application, and are passed to our app by the operating system. The JavaFX background processes use this information to identify the right node to enact the event on.



Event Types - Operating System-Generated (Input) Events

a. Key Events

Key events are generated by the operating system whenever a key changes state – that is, it's pressed or released. If a key remains pressed for a long time, the OS will initiate additional key events.

Key Event Types

There are only three types of key event:

- KEY_PRESSED: Fired whenever any key is pressed, or if a key is held down for longer than the OS-defined *repeat delay*. After the repeat delay, a pressed event is generated at the repeat rate until the key is released.
- KEY_RELEASED: Fired whenever a key is released.
- KEY_TYPED: Only generated when a character-generating key is pressed *and* released. Invoking `getCharacter()` method will return the letter that was typed, factoring in modifying keys such as shift in cases where letters can have a case.

Event Types - Operating System-Generated (Input) Events

a. Key Events

How to use KeyEvent

KeyEvents are easy to attach to any class that extends EventTarget (remembering that's every control, chart, cell and shape).

Key events can be added using the convenience methods `setOnKeyPressed()`, `setOnKeyReleased()` or `setOnKeyTyped()`, for example:

```
textField.setOnKeyPressed(event -> {
    //executable code inside lambda expression
});
```

This will overwrite any previous behaviour set for that event type. Alternatively, they can be added by using the `addEventFilter()` and `addEventHandler()` methods:

```
textField.addEventHandler(KeyEvent.KEY_PRESSED, event -> {
    //executable code inside lambda expression
});
```

Event filters are an excellent way to set key commands that you want to operate at the Window or Scene-level, regardless of which control has focus.

Event Types - Operating System-Generated (Input) Events

b. Mouse Events

The MouseEvent class defines any change in the mouse cursor position or button states. As with key events, they are provided by the operating system to the application, alongside the default mouse behaviour expectations that have been set by the user.

Mouse Event Types

There are 10 supported mouse events in JavaFX, alongside the standard ANY type.

MOUSE_PRESSED MOUSE_RELEASED MOUSE_CLICKED MOUSE_ENTERED_TARGET MOUSE_ENTERED	MOUSE_EXITED_TARGET MOUSE_EXITED MOUSE_MOVED MOUSE_DRAGGED DRAG_DETECTED

Event Types - Operating System-Generated (Input) Events

c. Touch Events

JavaFX provides support for touch-sensitive screens on laptops, tablets and mobile devices, and can also generate mouse events in JavaFX.

Touch Event Types

There are four supported touch events in JavaFX, alongside the standard ANY type

- **TOUCH_PRESSED**: Fired whenever a new touch point is created, such as when a finger hits the screen. Unlike the mouse event, it is not re-triggered on waiting (see `TOUCH_STATIONARY`)
- **TOUCH_MOVED**: Fired whenever any touch point moves, not necessarily the most recently pressed.
- **TOUCH_RELEASED**: Fired whenever any touch point is released, not necessarily the most recently pressed.
- **TOUCH_STATIONARY**: Continuously fired whenever all touch points are stationary. This event also seems to be fired whenever touch points move in different directions (such as during a pinch gesture).

.

Event Types - Operating System-Generated (Input) Events

d. Gesture Events

Gestures are part of JavaFX's in-built support for touch-sensitive devices. There are four types of gesture: rotate, swipe, scroll and zoom.

Gesture Event Types

There are 13 types of specific gesture event in addition to the ANY event types.

.

Rotate Events:	Scroll Events:	Swipe Events:	Zoom Events:
ROTATE ROTATION_FINISHED ROTATION_STARTED	SCROLL SCROLL_FINISHED SCROLL_STARTED	SWIPE_DOWN SWIPE_LEFT SWIPE_RIGHT SWIPE_UP	ZOOM ZOOM_FINISHED ZOOM_STARTED

Event Types - JavaFX-originated Events

JavaFX also has the ability to create and dispatch its own events. There are three types of these events:

- Action Events
- Worker Events
- Edit, Modification and Sorting Events

Each of these gives JavaFX the ability to layer functionality over the user interface.

.

Event Types - JavaFX-originated Events

a. Action Events

Action events are designed to span multiple input types, defining a single piece of executable code to use in the case of a specific user action. Controls such as the TextField and buttons have action events.

This layers functionality over the user interface in a way that can span multiple user inputs. An action, from the perspective of the JavaFX system, is some sort of affirmative user action. Examples of this are:

- Clicking a button
- Hitting Enter when the button is in focus
- Hitting Enter in a TextField
- Arming or unarming a toggle button (using keyboard *or* mouse)
- Clicking (or striking Enter) on a menu button

Action Event Types

The `ActionEvent` class only has one `EventType` – **`ACTION`** – which makes it sound like a Michael Bay film, but in reality it's an accurate summary of everything an `ActionEvent` can do. A user clicking on a button doesn't have multiple states. It does one thing – commit to an action.

Event Types - JavaFX-originated Events

b. Worker State Events

Long-running tasks can easily be outsourced to workers such as Tasks or Services. It's can be useful to fire events as these progress.

WorkerStateEvent Types

The WorkerStateEvent has 6 types in addition to the obligatory ANY.

- ✓ `WORKER_STATE_READY`
- ✓ `WORKER_STATE_RUNNING`
- ✓ `WORKER_STATE_CANCELLED`
- ✓ `WORKER_STATE_SCHEDULED`
- ✓ `WORKER_STATE_FAILED`
- ✓ `WORKER_STATE_SUCCEEDED`

Target Selection

Unlike input events, which are targetted in the foreground of the UI, Worker State Events do not target a node in the interface.

Regardless of which class defines the task or sets listeners, the Target of a Task is the class that invokes the `run()` method on the task.

Event Types - JavaFX-originated Events

c. Edit, Modification and Sorting Events

These events span a number of uses, but they're all responsible for updating complex UI elements that can't be achieved through property binding.

.

Event Types

With the exception of the `SortEvent` and `ScrollToEvent` classes, each `Event` is defined as an inner class within the control to which they're useful.

- `EditEvent` (`TreeTableView.EditEvent`, `TreeView.EditEvent`, `ListView.EditEvent`)
- `TreeModificationEvent` (`CheckBoxTreeItem.TreeModificationEvent`, `TreeItem.TreeModificationEvent`)
- `CellEditEvent` (`TableColumn.CellEditEvent`, `TreeTableColumn.CellEditEvent`)
- `SortEvent`
- `ScrollToEvent`

END