Event Programming in JavaFX

CSE 114, Computer Science 1
SUNY Korea

Pradipta De

Acknowledgement for the slides: Dr. Paul Fodor (CS Stony Brook)
Event Programming

• Procedural programming is executed in procedural/statement order.

• In event-driven programming, code is executed upon activation of events.

• Operating Systems constantly monitor events
  • Ex: keystrokes, mouse clicks, etc…

• The OS:
  • sorts out these events
  • reports them to the appropriate programs
Where do we come in?

- For each control (button, combo box, etc.):
  - define an event handler
  - construct an instance of event handler
  - tell the control who its event handler is

Event Handler?

- code with response to event
- a.k.a. event listener
Java’s Event Handling

- An **event source** is a GUI control
- **JavaFX**: Button, ChoiceBox, ListView, etc.

- different types of sources:
  - can detect different types of events
  - can register different types of listeners (handlers)

http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm
Java’s Event Handling

• When the user interacts with a control (source):
  • an *event object* is constructed
  • the event object is sent to all registered *listener objects*
  • the listener object (handler) responds as you defined it to
Event Listeners (Event Handler)

- Defined by you, the application programmer
  - you customize the response
- How?
  - Inheritance & Polymorphism
- You define your own listener class
  - implement the appropriate interface
  - define responses in all necessary methods
Event Objects

- Contain information about the event
- Like what?
  - location of mouse click
  - event source that was interacted with
  - etc.
- Listeners use them to properly respond
  - different methods inside a listener object can react differently to different types of interactions
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;

public class HandleEvent extends Application {
    public void start(Stage primaryStage) {
        HBox pane = new HBox(10);
        Button btOK = new Button("OK");
        Button btCancel = new Button("Cancel");
        OKHandlerClass handler1 = new OKHandlerClass();
        btOK.setOnAction(handler1);
        CancelHandlerClass handler2 = new CancelHandlerClass();
        btCancel.setOnAction(handler2);
        pane.getChildren().addAll(btOK, btCancel);
        Scene scene = new Scene(pane);
        primaryStage.setScene(scene);
        primaryStage.show();
    } /*main*/
}

class OKHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}

class CancelHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Cancel button clicked");
    }
}
Handling GUI Events

- Source object: button.
- An event is generated by external user actions such as mouse movements, mouse clicks, or keystrokes.
- An event can be defined as a type of signal to the program that something has happened.
- Listener object contains a method for processing the event.

Diagram:

- **button**: Clicking a button fires an action event (Event source object)
- **event**: An event is an object (Event object)
- **handler**: The event handler processes the event (Event handler object)
Event Classes

JavaFX event classes are in the `javafx.event` package
Event Information

• An event object contains whatever properties are pertinent to the event:
  • the *source object* of the event using the getSource() instance method in the EventObject class.
• The subclasses of EventObject deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes.
## Selected User Actions and Handlers

<table>
<thead>
<tr>
<th>User Action</th>
<th>Source Object</th>
<th>Event Type Fired</th>
<th>Event Registration Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click a button</td>
<td>Button</td>
<td>ActionEvent</td>
<td>setOnAction(EventHandler&lt;ActionEvent&gt;())</td>
</tr>
<tr>
<td>Press Enter in a text field</td>
<td>TextField</td>
<td>ActionEvent</td>
<td>setOnAction(EventHandler&lt;ActionEvent&gt;())</td>
</tr>
<tr>
<td>Check or uncheck</td>
<td>RadioButton</td>
<td>ActionEvent</td>
<td>setOnAction(EventHandler&lt;ActionEvent&gt;())</td>
</tr>
<tr>
<td>Check or uncheck</td>
<td>CheckBox</td>
<td>ActionEvent</td>
<td>setOnAction(EventHandler&lt;ActionEvent&gt;())</td>
</tr>
<tr>
<td>Select a new item</td>
<td>ComboBox</td>
<td>ActionEvent</td>
<td>setOnAction(EventHandler&lt;ActionEvent&gt;())</td>
</tr>
<tr>
<td>Mouse pressed</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td>setOnMousePressed(EventHandler&lt;MouseEvent&gt;())</td>
</tr>
<tr>
<td>Mouse released</td>
<td></td>
<td></td>
<td>setOnMouseReleased(EventHandler&lt;MouseEvent&gt;())</td>
</tr>
<tr>
<td>Mouse clicked</td>
<td></td>
<td></td>
<td>setOnMouseClicked(EventHandler&lt;MouseEvent&gt;())</td>
</tr>
<tr>
<td>Mouse entered</td>
<td></td>
<td></td>
<td>setOnMouseEntered(EventHandler&lt;MouseEvent&gt;())</td>
</tr>
<tr>
<td>Mouse exited</td>
<td></td>
<td></td>
<td>setOnMouseExited(EventHandler&lt;MouseEvent&gt;())</td>
</tr>
<tr>
<td>Mouse moved</td>
<td></td>
<td></td>
<td>setOnMouseMoved(EventHandler&lt;MouseEvent&gt;())</td>
</tr>
<tr>
<td>Mouse dragged</td>
<td></td>
<td></td>
<td>setOnMouseDragged(EventHandler&lt;MouseEvent&gt;())</td>
</tr>
<tr>
<td>Key pressed</td>
<td>Node, Scene</td>
<td>KeyEvent</td>
<td>setOnKeyPressed(EventHandler&lt;KeyEvent&gt;())</td>
</tr>
<tr>
<td>Key released</td>
<td></td>
<td></td>
<td>setOnKeyReleased(EventHandler&lt;KeyEvent&gt;())</td>
</tr>
<tr>
<td>Key typed</td>
<td></td>
<td></td>
<td>setOnKeyTyped(EventHandler&lt;KeyEvent&gt;())</td>
</tr>
</tbody>
</table>
The Delegation Model

1. A listener object is an instance of a listener interface

2. Register by invoking
   \( \text{source.setOnXEventType(listener);} \)

(a) A generic source object with a generic event \( T \)

(b) A Button source object with an ActionEvent
ControlCircle program that uses two buttons to control the size of a circle

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;

public class ControlCircle extends Application {
    private CirclePane circlePane = new CirclePane();

    @Override
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        Button btEnlarge = new Button("Enlarge");
        Button btShrink = new Button("Shrink");
        hBox.getChildren().add(btEnlarge);
        hBox.getChildren().add(btShrink);
        btEnlarge.setOnAction(new EnlargeHandler());
        BorderPane borderPane = new BorderPane();
        borderPane.setCenter(circlePane);
        borderPane.setBottom(hBox);
        BorderPane.setAlignment(hBox, Pos.CENTER);
        Scene scene = new Scene(borderPane, 200, 150);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```
ControlCircle program that uses two buttons to control the size of a circle

```java
// Inner Class
class EnlargeHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        circlePane.enlarge();
    }
}

class CirclePane extends StackPane {
    private Circle circle = new Circle(50);
    public CirclePane() {
        getChildren().add(circle);
        circle.setStroke(Color.BLACK);
        circle.setFill(Color.WHITE);
    }
    public void enlarge() {
        circle.setRadius(circle.getRadius() + 2);
    }
    public void shrink() {
        circle.setRadius(circle.getRadius() > 2 ? circle.getRadius() - 2 : circle.getRadius());
    }
}
```
Inner Class Listeners

• A listener class is designed specifically to create a listener object for a GUI component (e.g., a button).

• Any object instance of the inner handler class has access to all GUI fields of the outer class.

• It will not be shared by other applications.
The Inner class is a class that is a member of another class.

An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.

An inner class is compiled into a class named OuterClassName$InnerClassName.class.
Inner Classes

• An inner class can be declared public, protected, or private subject to the same visibility rules applied to a member of the class.

• An inner class can be declared static:
  • The static inner class can be accessed using the outer class name,
  • However, a static inner class cannot access nonstatic members of the outer class.
Anonymous Inner Classes

• Inner class listeners can be shortened using anonymous inner classes: inner classes without a name.
  • It combines declaring an inner class and creating an instance of the class in one step.
  • An anonymous inner class is declared as follows:

```java
new SuperClassName/InterfaceName() {
    // Implement or override methods in superclass/interface
    // Other methods if necessary
}
```
Anonymous Inner Classes

- An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- An anonymous inner class always uses the no-arg constructor from its superclass to create an instance.
- If an anonymous inner class implements an interface, the constructor is Object().
- An anonymous inner class is compiled into a class named OuterClassName\$n.class, where n is the count of inner classes.
Anonymous Inner Classes

```java
public void start(Stage primaryStage) {
    // Omitted
    btEnlarge.setOnAction(
        new EnlargeHandler());
}

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

(a) Inner class EnlargeListener

```java
public void start(Stage primaryStage) {
    // Omitted
    btEnlarge.setOnAction(
        new EnlargeHandler()
            implements EventHandler<ActionEvent> {
                public void handle(ActionEvent e) {
                    circlePane.enlarge();
                }
            });
}
```

(b) Anonymous inner class
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;

public class AnonymousHandlerDemo extends Application {
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        Button btNew = new Button("New");
        Button btOpen = new Button("Open"); // btSave, btPrint btns.
        hBox.getChildren().addAll(btNew, btOpen);
        // Create and register the handler
        btNew.setOnAction(new EventHandler<ActionEvent>() {
            @Override // Override the handle method
            public void handle(ActionEvent e) {
                System.out.println("Process New");
            }
        });
        btOpen.setOnAction(new EventHandler<ActionEvent>() {
            @Override // Override the handle method
            public void handle(ActionEvent e) {
                System.out.println("Process Open");
            }
        });
    }
}

Scene scene = new Scene(hBox, 300, 50);
primaryStage.setTitle("AnonymousHandlerDemo");
primaryStage.setScene(scene);
primaryStage.show();

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

Lambda expression is a new feature in Java 8.

- Predefined functions for the type of the input.
- Lambda expressions can be viewed as an anonymous method with a concise syntax.

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

(a) Anonymous inner class event handler

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

(b) Lambda expression event handler
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;

public class LambdaHandlerDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Hold two buttons in an HBox
        HBox hBox = new HBox();
        hBox.setSpacing(10);
        hBox.setAlignment(Pos.CENTER);
        Button btNew = new Button("New");
        Button btOpen = new Button("Open");
        Button btSave = new Button("Save");
        Button btPrint = new Button("Print");
        hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);
        btNew.setOnAction(e -> { System.out.println("Process New"); });
        btOpen.setOnAction(e -> { System.out.println("Process Open"); });
        btSave.setOnAction(e -> { System.out.println("Process Save"); });
        btPrint.setOnAction(e -> { System.out.println("Process Print"); });
        Scene scene = new Scene(hBox, 300, 50);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
Basic Syntax for a Lambda Expression

• The basic syntax for a lambda expression is either:
  (type1 param1, type2 param2, ...) -> expression
  or
  (type1 param1, type2 param2, ...) -> { statements; }

• The data type for a parameter may be explicitly declared or implicitly inferred by the compiler.

• The parentheses can be omitted if there is only one parameter without an explicit data type.
Single Abstract Method Interface (SAM)

• The statements in the lambda expression is all for that method.
• If it contains multiple methods, the compiler will not be able to compile the lambda expression.
• So, for the compiler to understand lambda expressions, the interface must contain exactly one abstract method.
• Such an interface is known as a functional interface, or a Single Abstract Method (SAM) interface.
### javafx.scene.input.MouseEvent

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getButton()</code></td>
<td>Returns the number of mouse clicks associated with this event.</td>
</tr>
<tr>
<td><code>getClickCount()</code></td>
<td>Returns the x-coordinate of the mouse point in the event source node.</td>
</tr>
<tr>
<td><code>getX()</code></td>
<td>Returns the y-coordinate of the mouse point in the event source node.</td>
</tr>
<tr>
<td><code>getY()</code></td>
<td>Returns the x-coordinate of the mouse point in the scene.</td>
</tr>
<tr>
<td><code>getSceneX()</code></td>
<td>Returns the y-coordinate of the mouse point in the scene.</td>
</tr>
<tr>
<td><code>getSceneY()</code></td>
<td>Returns the x-coordinate of the mouse point in the screen.</td>
</tr>
<tr>
<td><code>getScreenX()</code></td>
<td>Returns the y-coordinate of the mouse point in the screen.</td>
</tr>
<tr>
<td><code>getScreenY()</code></td>
<td>Returns true if the <code>Alt</code> key is pressed on this event.</td>
</tr>
<tr>
<td><code>isAltDown()</code></td>
<td>Returns true if the <code>Control</code> key is pressed on this event.</td>
</tr>
<tr>
<td><code>isControlDown()</code></td>
<td>Returns true if the <code>Meta</code> button is pressed on this event.</td>
</tr>
<tr>
<td><code>isMetaDown()</code></td>
<td>Returns true if the <code>Shift</code> key is pressed on this event.</td>
</tr>
<tr>
<td><code>isShiftDown()</code></td>
<td></td>
</tr>
</tbody>
</table>

Indicates which mouse button has been clicked.
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;
public class MouseEventDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Text text = new Text(20, 20, "Programming is fun");
        pane.getChildren().addAll(text);
        text.setOnMouseDragged(e -> {
            text.setX(e.getX());
            text.setY(e.getY());
        });
        Scene scene = new Scene(pane, 300, 100);
        primaryStage.setTitle("MouseEventDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
The `KeyEvent` Class

```java
class javafx.scene.input.KeyEvent

+ getCharacter(): String
  Returns the character associated with the key in this event.
+ getCode(): KeyCode
  Returns the key code associated with the key in this event.
+ getText(): String
  Returns a string describing the key code.
+ isAltDown(): boolean
  Returns true if the Alt key is pressed on this event.
+ isControlDown(): boolean
  Returns true if the Control key is pressed on this event.
+ isMetaDown(): boolean
  Returns true if the mouse Meta button is pressed on this event.
+ isShiftDown(): boolean
  Returns true if the Shift key is pressed on this event.
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;

public class KeyEventDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Text text = new Text(20, 20, "A");
        text.setFocusTraversable(true);
        pane.getChildren().add(text);
        text.setOnKeyPressed(e -> {
            switch (e.getCode()) {
            case KeyCode.DOWN: text.setY(text.getY() + 10); break;
            case KeyCode.UP: text.setY(text.getY() - 10); break;
            case KeyCode.LEFT: text.setX(text.getX() - 10); break;
            case KeyCode.RIGHT: text.setX(text.getX() + 10); break;
            default:
                if (Character.isLetterOrDigit(e.getText().charAt(0)))
                    text.setText(e.getText());
            }
        });
        Scene scene = new Scene(pane);
        primaryStage.setTitle("KeyEventDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
# The KeyCode Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>The Home key</td>
<td>CONTROL</td>
<td>The Control key</td>
</tr>
<tr>
<td>END</td>
<td>The End key</td>
<td>SHIFT</td>
<td>The Shift key</td>
</tr>
<tr>
<td>PAGE_UP</td>
<td>The Page Up key</td>
<td>BACK_SPACE</td>
<td>The Backspace key</td>
</tr>
<tr>
<td>PAGE_DOWN</td>
<td>The Page Down key</td>
<td>CAPS</td>
<td>The Caps Lock key</td>
</tr>
<tr>
<td>UP</td>
<td>The up-arrow key</td>
<td>NUM_LOCK</td>
<td>The Num Lock key</td>
</tr>
<tr>
<td>DOWN</td>
<td>The down-arrow key</td>
<td>ENTER</td>
<td>The Enter key</td>
</tr>
<tr>
<td>LEFT</td>
<td>The left-arrow key</td>
<td>UNDEFINED</td>
<td>The keyCode unknown</td>
</tr>
<tr>
<td>RIGHT</td>
<td>The right-arrow key</td>
<td>F1 to F12</td>
<td>The function keys from F1 to F12</td>
</tr>
<tr>
<td>ESCAPE</td>
<td>The Esc key</td>
<td>0 to 9</td>
<td>The number keys from 0 to 9</td>
</tr>
<tr>
<td>TAB</td>
<td>The Tab key</td>
<td>A to Z</td>
<td>The letter keys from A to Z</td>
</tr>
</tbody>
</table>
public class ControlCircleWithMouseAndKey extends Application {
    private CirclePane circlePane = new CirclePane();

    @Override
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        hBox.setSpacing(10);
        hBox.setAlignment(Pos.CENTER);
        Button btEnlarge = new Button("Enlarge");
        Button btShrink = new Button("Shrink");
        hBox.getChildren().add(btEnlarge);
        hBox.getChildren().add(btShrink);
        btEnlarge.setOnAction(e -> circlePane.enlarge());
        btShrink.setOnAction(e -> circlePane.shrink());
        circlePane.setOnMouseClicked(e -> {
            if (e.getButton() == MouseButton.PRIMARY) {
                circlePane.enlarge();
            } else if (e.getButton() == MouseButton.SECONDARY) {
                circlePane.shrink();
            }
        });
        circlePane.setOnKeyPressed(e -> {
            if (e.getCode() == KeyCode.U) {
                circlePane.enlarge();
            } else if (e.getCode() == KeyCode.D) {
                circlePane.shrink();
            }
        });
        BorderPane borderPane = new BorderPane();
        borderPane.setCenter(circlePane);
        borderPane.setBottom(hBox);
        BorderPane.setAlignment(hBox, Pos.CENTER);
        Scene scene = new Scene(borderPane, 200, 150); ...
Listeners for Observable Objects

• You can add a listener to process a value change in an observable object (an instance of Observable).
• Every binding property is an instance of Observable.
• Observable contains the addListener(InvalidationListener listener) method for adding a listener.
• Once the value is changed in the property, a listener is notified.
• The listener class should implement the InvalidationListener interface, which uses the invalidated(Observable o) method to handle the property value change.
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class ObservablePropertyDemo {
    public static void main(String[] args) {
        DoubleProperty balance = new SimpleDoubleProperty();
        balance.addListener(new InvalidationListener() {
            public void invalidated(Observable ov) {
                System.out.println("The new value is " +
                    balance.doubleValue());
            }
        });
        balance.set(4.5);
    }
}

Output:
The new value is 4.5
Animation

- JavaFX provides the **Animation** class with the core functionality for all animations.

```java
javafx.animation.Animation

-autoReverse: BooleanProperty
-cycleCount: IntegerProperty
-rate: DoubleProperty
-status: ReadOnlyObjectProperty
<Animation.Status>

+pause(): void
+play(): void
+stop(): void
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

- Defines whether the animation reverses direction on alternating cycles.
- Defines the number of cycles in this animation.
- Defines the speed and direction for this animation.
- Read-only property to indicate the status of the animation.

- Pauses the animation.
- Plays the animation from the current position.
- Stops the animation and resets the animation.
PathTransition

```java
javafx.animation.PathTransition

- duration: ObjectProperty<Duration>
- node: ObjectProperty<Node>
- orientation: ObjectProperty<PathTransition.OrientationType>
- path: ObjectProperty<Shape>

+PathTransition()
+PathTransition(duration: Duration, path: Shape)
+PathTransition(duration: Duration, path: Shape, node: Node)
```

The duration of this transition.
The target node of this transition.
The orientation of the node along the path.
The shape whose outline is used as a path to animate the node move.

Creates an empty PathTransition.
Creates a PathTransition with the specified duration and path.
Creates a PathTransition with the specified duration, path, and node.

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.Circle;
import javafx.animation.PathTransition;
import javafx.animation.Timeline;
import javafx.util.Duration;

public class PathTransitionDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Rectangle rectangle = new Rectangle(0, 0, 25, 50);
        rectangle.setFill(Color.ORANGE);
        Circle circle = new Circle(125, 100, 50);
        circle.setFill(Color.WHITE);
        circle.setStroke(Color.BLACK);
        pane.getChildren().addAll(circle, rectangle);
        // Create a path transition
        PathTransition pt = new PathTransition();
        pt.setDuration(Duration.millis(4000));
        pt.setPath(circle);
        pt.setNode(rectangle);
    }
}
```
pt.setOrientation(
    PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
pt.setCycleCount(Timeline.INDEFINITE);
pt.setAutoReverse(true);
pt.play(); // Start animation
circle.setOnMousePressed(e -> pt.pause());
circle.setOnMouseReleased(e -> pt.play());
Scene scene = new Scene(pane, 250, 200);
primaryStage.setTitle("PathTransitionDemo");
primaryStage.setScene(scene);
primaryStage.show();
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Line;
import javafx.animation.PathTransition;
import javafx.scene.image.ImageView;
import javafx.util.Duration;

public class FlagRisingAnimation extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        ImageView imageView = new ImageView("us.jpg");
        pane.getChildren().add(imageView);
        PathTransition pt = new PathTransition(
            Duration.millis(10000),
            new Line(100, 200, 100, 0),
            imageView);
        pt.setCycleCount(5);
        pt.play(); // Start animation
        Scene scene = new Scene(pane, 250, 200);
        primaryStage.setScene(scene); primaryStage.show();
    }
}
The **FadeTransition** class animates the change of the opacity in a node over a given time.

```java
javafx.animation.FadeTransition

- duration: ObjectProperty<Duration>
- node: ObjectProperty<Node>
- fromValue: DoubleProperty
- toValue: DoubleProperty
- byValue: DoubleProperty

+ FadeTransition()
+ FadeTransition(duration: Duration)
+ FadeTransition(duration: Duration, node: Node)
```

- The duration of this transition.
- The target node of this transition.
- The start opacity for this animation.
- The stop opacity for this animation.
- The incremental value on the opacity for this animation.

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Ellipse;
import javafx.animation.FadeTransition;
import javafx.animation.Timeline;
import javafx.util.Duration;

class FadeTransitionDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Ellipse ellipse = new Ellipse(10, 10, 100, 50);
        ellipse.setFill(Color.RED);
        ellipse.setStroke(Color.BLACK);
        ellipse.centerXProperty().bind(pane.widthProperty().divide(2));
        ellipse.centerYProperty().bind(pane.heightProperty().divide(2));
        ellipse.radiusXProperty().bind(pane.widthProperty().multiply(0.4));
        ellipse.radiusYProperty().bind(pane.heightProperty().multiply(0.4));
        pane.getChildren().add(ellipse);
        // Apply a fade transition to ellipse
        FadeTransition ft = new FadeTransition(Duration.millis(3000), ellipse);
        ft.setFromValue(1.0);
        ft.setToValue(0.1);
        ft.setCycleCount(Timeline.INDEFINITE);
        ft.setAutoReverse(true);
        ft.play(); // Start animation
        // Control animation
        ellipse.setOnMousePressed(e -> ft.pause());
        ellipse.setOnMouseReleased(e -> ft.play());
    }
}
Timeline

- **PathTransition** and **FadeTransition** define specialized animations.
- The `javafx.animation.Timeline` class can be used to program any animation using one or more `javafx.animation.KeyFrames`.
  - **KeyFrame** defines target values at a specified point in time for a set of variables that are interpolated along a **Timeline**.
    - Each **KeyFrame** is executed sequentially at a specified time interval.
- **Timeline** inherits from **Animation**.
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.util.Duration;

public class TimelineDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        StackPane pane = new StackPane();
        Text text = new Text(20, 50, "Programming if fun");
        text.setFill(Color.RED);
        pane.getChildren().add(text);
        // Create a handler for changing text
        EventHandler<ActionEvent> eH = e -> {
            if (text.getText().length() != 0) {
                text.setText("");  // Text is empty
            } else {
                text.setText("Programming is fun");
            }
        };
        Timeline animation = new Timeline(new KeyFrame(Duration.millis(500), eH));
        animation.setCycleCount(Timeline.INDEFINITE);
        // Start animation
        animation.play();
    }
}

// Pause and resume animation
text.setOnMouseClicked(e -> {
    if (animation.getStatus() == Animation.Status.PAUSED) {
        animation.play();
    } else {
        animation.pause();
    }
});

Scene scene = new Scene(pane, 250, 50);
primaryStage.setTitle("TimelineDemo");
primaryStage.setScene(scene);
primaryStage.show();
JavaFX support for mobile devices

- JavaFX has event programming support for mobile devices:
  
  `javafx.scene.input.SwipeEvent`,
  `javafx.scene.input.TouchEvent`,
  `javafx.scene.input.ZoomEvent`.

- Example:

  [http://docs.oracle.com/javase/8/javafx/events-tutorial/gestureeventsjava.htm](http://docs.oracle.com/javase/8/javafx/events-tutorial/gestureeventsjava.htm)

  [http://docs.oracle.com/javase/8/javafx/events-tutorial/toucheventsjava.htm](http://docs.oracle.com/javase/8/javafx/events-tutorial/toucheventsjava.htm)