

Event Programming in JavaFX

CSE 114, Computer Science I

Stony Brook University

<http://www.cs.stonybrook.edu/~cse114>

Event Programming

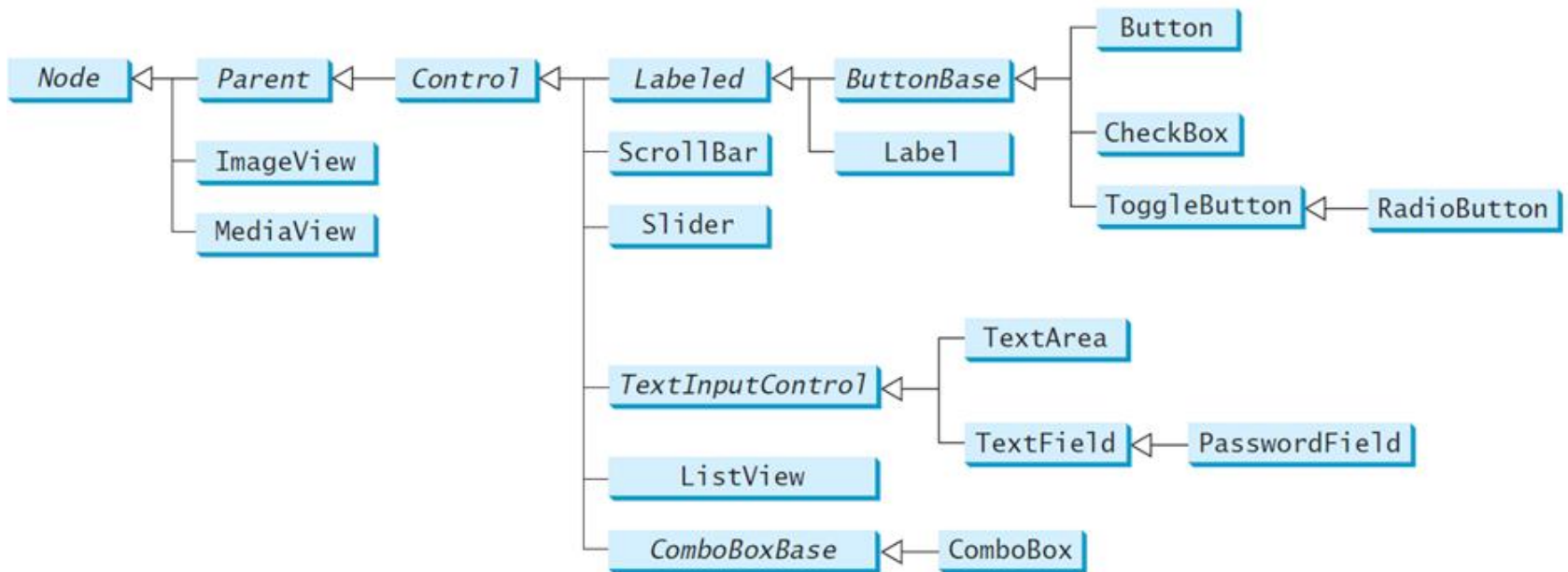
- Procedural programming is executed in procedural/statement order.
- In event-driven programming, code is executed upon activation of events.
- Operating Systems constantly monitor events
 - Ex: keystrokes, mouse clicks, etc...
- The OS:
 - sorts out these events
 - reports them to the appropriate programs

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.



http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

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

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

```

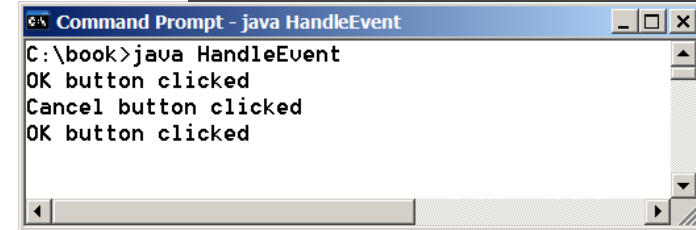
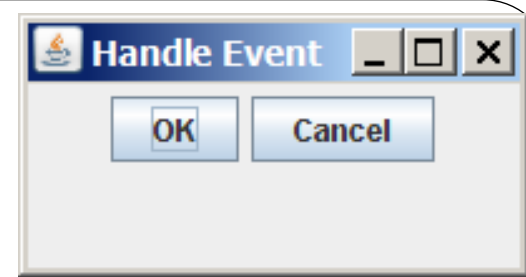
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;

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");
    }
}

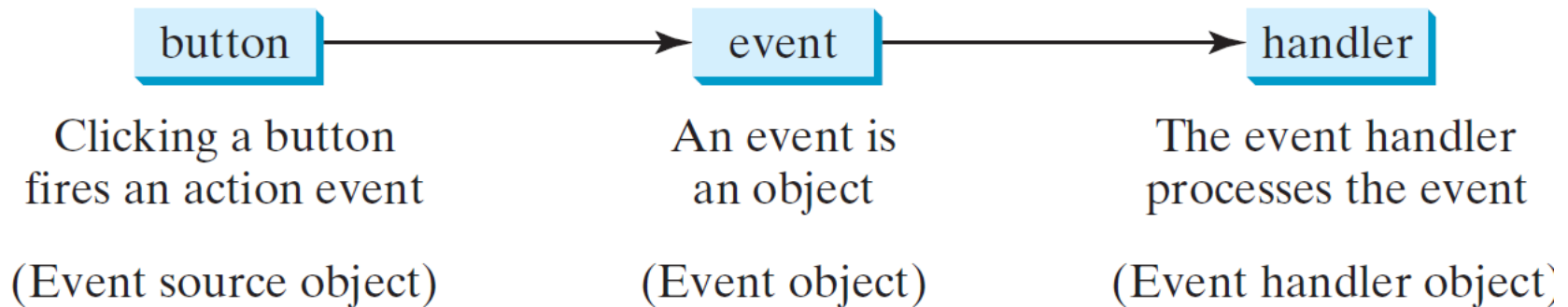
class CancelHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Cancel button clicked");
    }
}

```

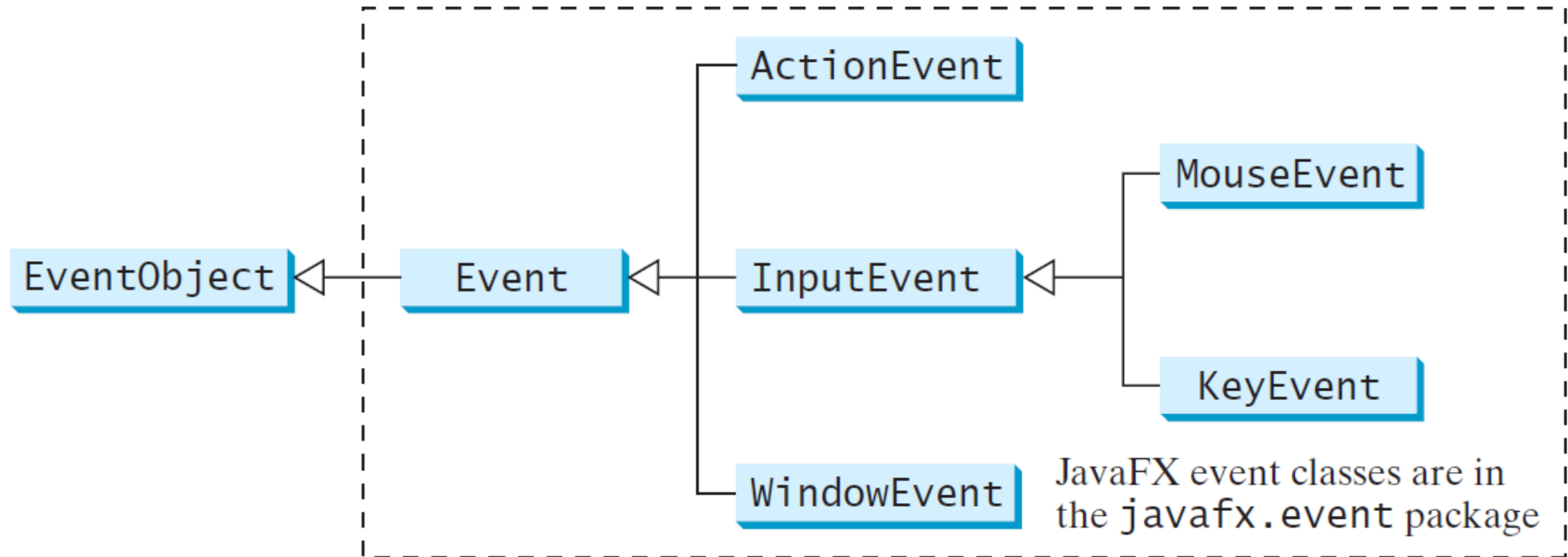


Handling GUI Events

- 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.



Event Classes



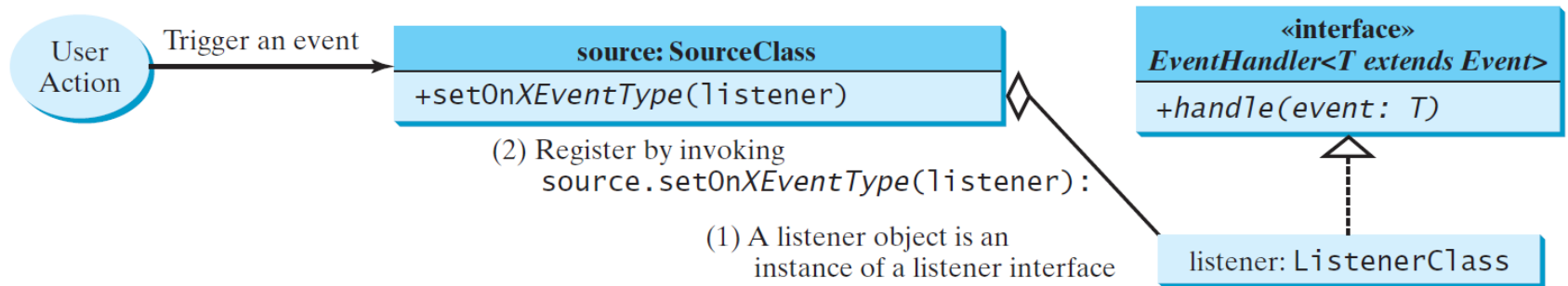
Event Information

- 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.

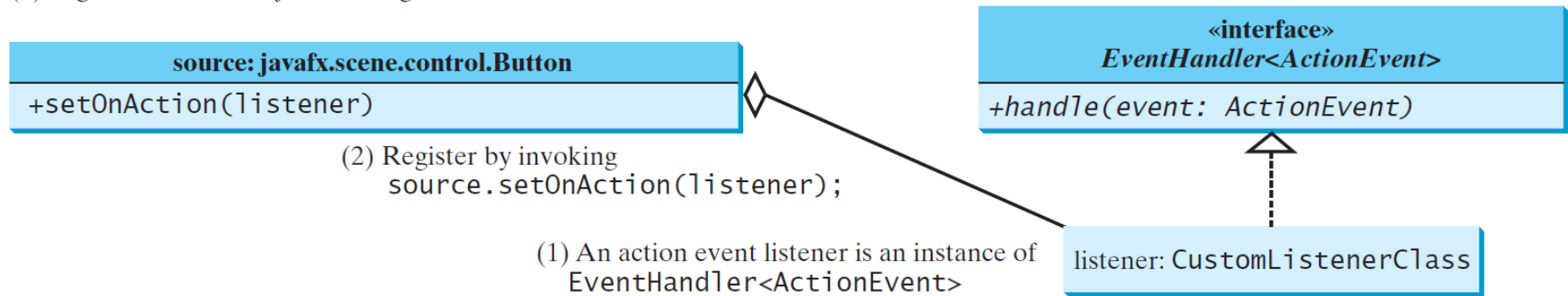
Selected User Actions and Handlers

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

The Delegation Model



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

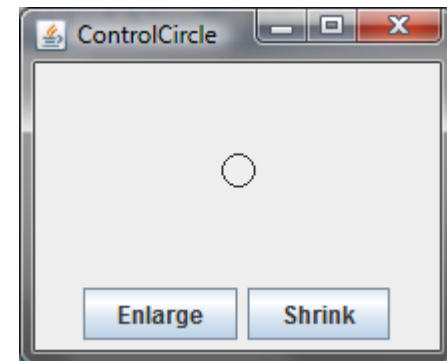
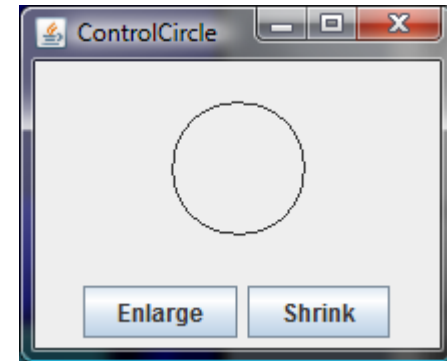


(b) A Button source object with an ActionEvent

ControlCircle program that uses two buttons to control the size of a circle

```
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;

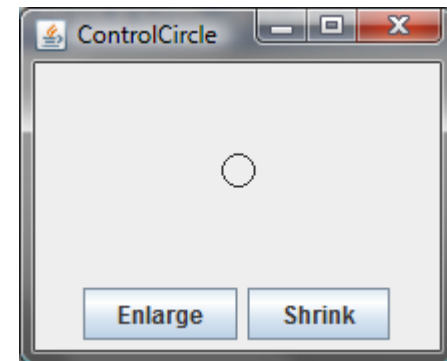
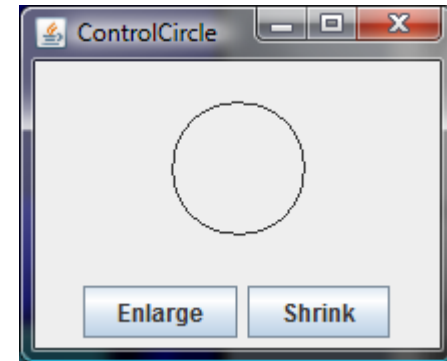
public class ControlCircle extends Application {
    private CirclePane circlePane = new CirclePane();
    @Override
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        Button btEnlarge = new Button("Enlarge");
        Button btShrink = new Button("Shrink");
        hBox.getChildren().add(btEnlarge);
        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

```
// 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.

Inner Classes

```
public class OuterClass {
    private int data = 0;
    OuterClass() {
        InnerClass y = new InnerClass();
        y.m2();
    }
    public void m1() {
        data++;
    }
    public static void main(String[] args) {
        OuterClass x = new OuterClass();
        System.out.println(x.data);
    }
    class InnerClass {
        public void m2() {
            /* Directly reference data and
               method defined in outer class */
            data++;
            m1();
        }
    }
}
```

- The **Inner** class is a class 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:

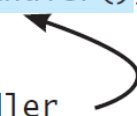
```
new SuperClassName/InterfaceName() {  
    // Implement or override methods in superclass/interface  
    // Other methods if necessary  
}
```

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

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



(a) Inner class EnlargeListener

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

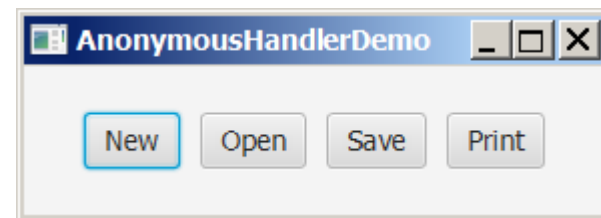
(b) Anonymous inner class

```

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);  
    }  
}
```

Simplifying Event Handling Using Lambda Expressions

- *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.

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

(a) Anonymous inner class event handler

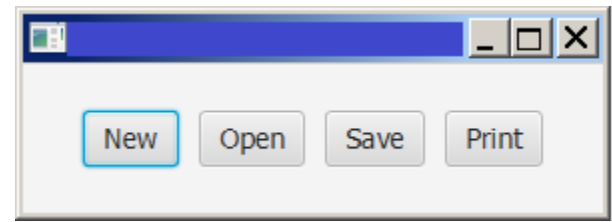
```
btEnlarge.setOnAction(e -> {  
    // Code for processing event e  
});
```

(b) Lambda expression event handler


```

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);
    }
}

```



run-single:
 Process New
 Process Open
 Process Save
 Process Print

Basic Syntax for a Lambda Expression

- The basic syntax for a lambda expression is either:
 $(\text{type1 param1}, \text{type2 param2}, \dots) \rightarrow \text{expression}$
or
 $(\text{type1 param1}, \text{type2 param2}, \dots) \rightarrow \{ \text{statements}; \}$
- 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)

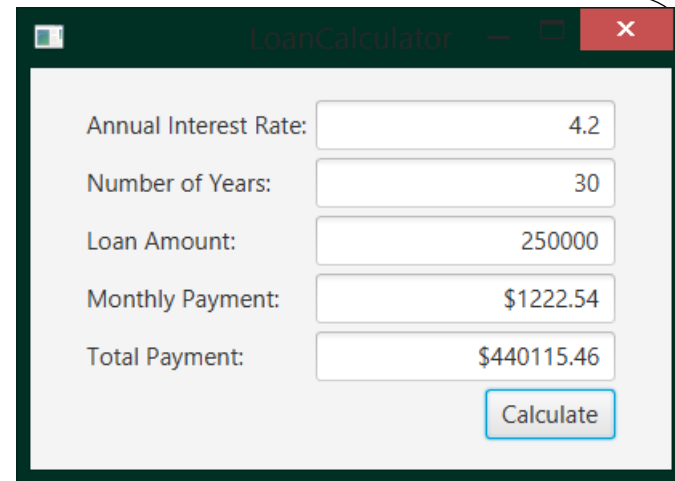
- The statements in the lambda expression is all for that method.
 - If it contains multiple methods, the compiler will not be able to compile the lambda expression.
 - So, for the compiler to understand lambda expressions, the interface must contain exactly one abstract method.
 - Such an interface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

Loan Calculator

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.geometry.Pos;
import javafx.geometry.HPos;

public class LoanCalculator extends Application {
    private TextField tfAnnualInterestRate = new TextField();
    private TextField tfNumberOfYears = new TextField();
    private TextField tfLoanAmount = new TextField();
    private TextField tfMonthlyPayment = new TextField();
    private TextField tfTotalPayment = new TextField();
    private Button btCalculate = new Button("Calculate");

    @Override
    public void start(Stage primaryStage) {
        // Create UI
        GridPane gridPane = new GridPane();
        gridPane.setHgap(5);
        gridPane.setVgap(5);
        gridPane.add(new Label("Annual Interest Rate:"), 0, 0);
        gridPane.add(tfAnnualInterestRate, 1, 0);
        gridPane.add(new Label("Number of Years:"), 0, 1);
        gridPane.add(tfNumberOfYears, 1, 1);
        gridPane.add(new Label("Loan Amount:"), 0, 2);
        gridPane.add(tfLoanAmount, 1, 2);
        gridPane.add(new Label("Monthly Payment:"), 0, 3);
        gridPane.add(tfMonthlyPayment, 1, 3);
        gridPane.add(new Label("Total Payment:"), 0, 4);
        gridPane.add(tfTotalPayment, 1, 4);
        gridPane.add(btCalculate, 1, 5);
    }
}
```



Annual Interest Rate:	4.2
Number of Years:	30
Loan Amount:	250000
Monthly Payment:	\$1222.54
Total Payment:	\$440115.46
<button>Calculate</button>	

```

        btCalculate.setOnAction(e -> calculateLoanPayment());
        Scene scene = new Scene(gridPane, 400, 250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    private void calculateLoanPayment() {
        // Get values from text fields
        double interest = Double.parseDouble(tfAnnualInterestRate.getText());
        int year = Integer.parseInt(tfNumberOfYears.getText());
        double loanAmount = Double.parseDouble(tfLoanAmount.getText());
        // Create a loan object
        Loan loan = new Loan(interest, year, loanAmount);
        // Display monthly payment and total payment
        tfMonthlyPayment.setText(String.format("%.2f", loan.getMonthlyPayment()));
        tfTotalPayment.setText(String.format("%.2f", loan.getTotalPayment()));
    }
    public static void main(String[] args) {
        launch(args);
    }
}

class Loan implements java.io.Serializable {
    private double annualInterestRate;
    private int numberOfYears;
    private double loanAmount;
    private java.util.Date loanDate;
    public Loan(double annualInterestRate, int numberOfYears, double loanAmount) {
        this.annualInterestRate = annualInterestRate;
        this.numberOfYears = numberOfYears;
        this.loanAmount = loanAmount;
        loanDate = new java.util.Date();
    }
    public double getAnnualInterestRate() {
        return annualInterestRate;
    }
    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }
}

```

```

public int getNumberOfYears() {
    return numberOfYears;
}
public void setNumberOfYears(int numberOfYears) {
    this.numberOfYears = numberOfYears;
}
public double getLoanAmount() {
    return loanAmount;
}
public void setLoanAmount(double loanAmount) {
    this.loanAmount = loanAmount;
}
public double getMonthlyPayment() {
    double monthlyInterestRate = annualInterestRate / 1200;
    double monthlyPayment = loanAmount * monthlyInterestRate / (1
        - (Math.pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
    return monthlyPayment;
}
public double getTotalPayment() {
    double totalPayment = getMonthlyPayment() * numberOfYears * 12;
    return totalPayment;
}
public java.util.Date getLoanDate() {
    return loanDate;
}
}

```

MouseEvent

javafx.scene.input.MouseEvent

```
+getButton(): MouseButton  
+getClickCount(): int  
+getX(): double  
+getY(): double  
+getSceneX(): double  
+getSceneY(): double  
+getScreenX(): double  
+getScreenY(): double  
+isAltDown(): boolean  
+isControlDown(): boolean  
+isMetaDown(): boolean  
+isShiftDown(): boolean
```

Indicates which mouse button has been clicked.

Returns the number of mouse clicks associated with this event.

Returns the *x*-coordinate of the mouse point in the event source node.

Returns the *y*-coordinate of the mouse point in the event source node.

Returns the *x*-coordinate of the mouse point in the scene.

Returns the *y*-coordinate of the mouse point in the scene.

Returns the *x*-coordinate of the mouse point in the screen.

Returns the *y*-coordinate of the mouse point in the screen.

Returns true if the **Alt** key is pressed on this event.

Returns true if the **Control** key is pressed on this event.

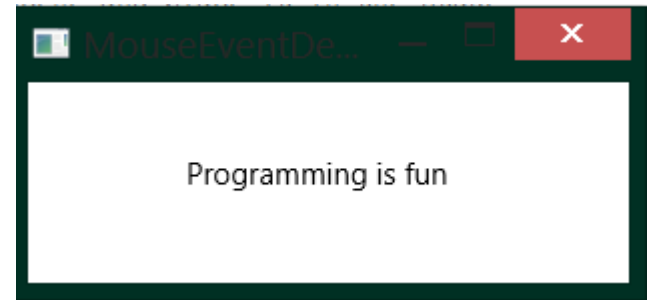
Returns true if the mouse **Meta** button is pressed on this event.

Returns true if the **Shift** key is pressed on this event.

```
// Move the text with the mouse 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

javafx.scene.input.KeyEvent

```
+getCharacter(): String  
+getCode(): KeyCode  
+getText(): String  
+isAltDown(): boolean  
+isControlDown(): boolean  
+isMetaDown(): boolean  
+isShiftDown(): boolean
```

Returns the character associated with the key in this event.

Returns the key code associated with the key in this event.

Returns a string describing the key code.

Returns true if the **Alt** key is pressed on this event.

Returns true if the **Control** key is pressed on this event.

Returns true if the mouse **Meta** button is pressed on this event.

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();
    }
}

```

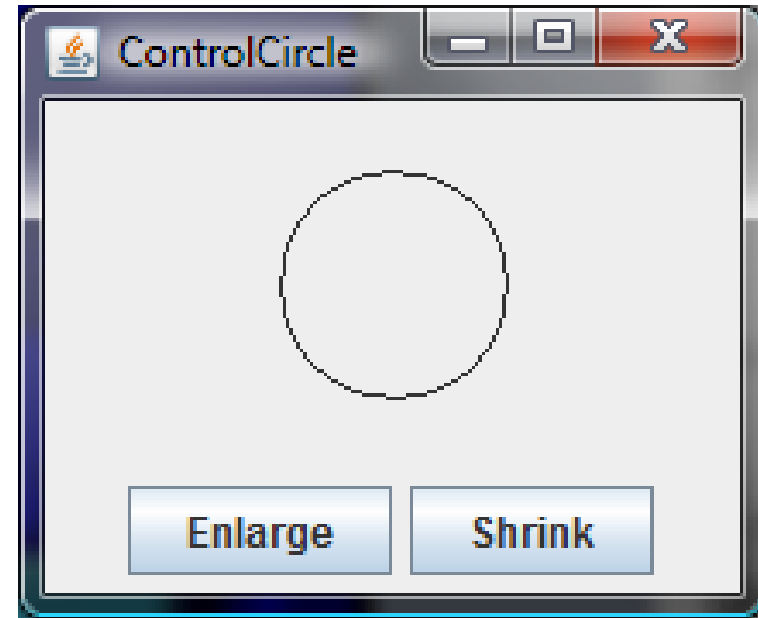


The KeyCode Constants

<i>Constant</i>	<i>Description</i>	<i>Constant</i>	<i>Description</i>
HOME	The Home key	CONTROL	The Control key
END	The End key	SHIFT	The Shift key
PAGE_UP	The Page Up key	BACK_SPACE	The Backspace key
PAGE_DOWN	The Page Down key	CAPS	The Caps Lock key
UP	The up-arrow key	NUM_LOCK	The Num Lock key
DOWN	The down-arrow key	ENTER	The Enter key
LEFT	The left-arrow key	UNDEFINED	The keyCode unknown
RIGHT	The right-arrow key	F1 to F12	The function keys from F1 to F12
ESCAPE	The Esc key	0 to 9	The number keys from 0 to 9
TAB	The Tab key	A to Z	The letter keys from A to Z

Control Circle with Mouse and Key

```
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.

Listeners for Observable Objects

```
import javafx.beans.InvalidListener;  
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 InvalidListener() {  
            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.

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

javafx.animation.PathTransition

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

+PathTransition()
+PathTransition(duration: Duration,
 path: Shape)
+PathTransition(duration: Duration,
 path: Shape, node: 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.

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.

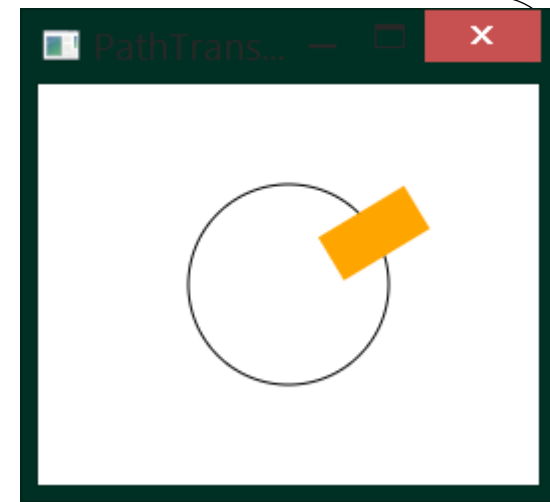
Creates an empty PathTransition.

Creates a PathTransition with the specified duration and path.

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


```
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);
    }
}
```

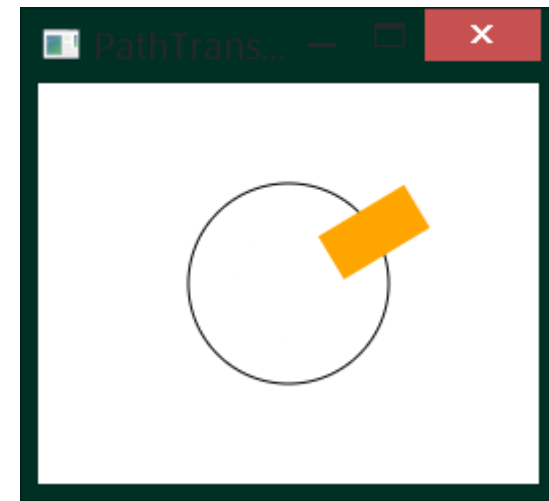


```

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();
}

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

```



```
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();
    }
}
```



FadeTransition

The **FadeTransition** class animates the change of the opacity in a node over a given time.

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 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.

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.

Creates an empty `FadeTransition`.

Creates a `FadeTransition` with the specified duration.

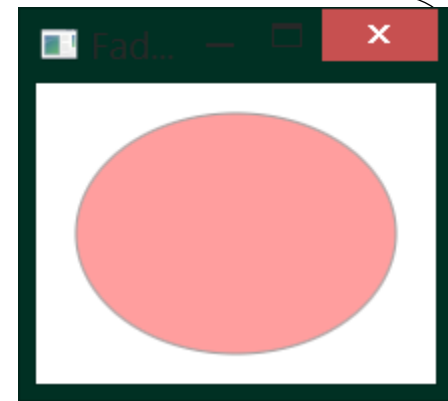
Creates a `FadeTransition` with the specified duration and node.

```

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;

public 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

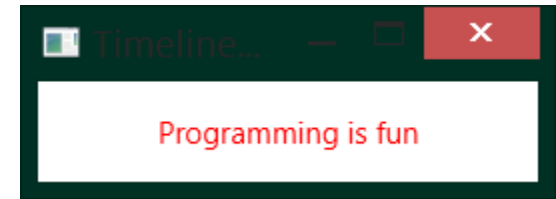
- **PathTransition** and **FadeTransition** define specialized animations.
- The `javafx.animation.Timeline` class can be used to program any animation using one or more `javafx.animation.KeyFrames`
 - **KeyFrame** defines target values at a specified point in time for a set of variables that are interpolated 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("");
            } else {
                text.setText("Programming is fun");
            }
        };
        Timeline animation = new Timeline(new KeyFrame(Duration.millis(500), eH));
        animation.setCycleCount(Timeline.INDEFINITE);
        // Start animation
        animation.play();
    }
}

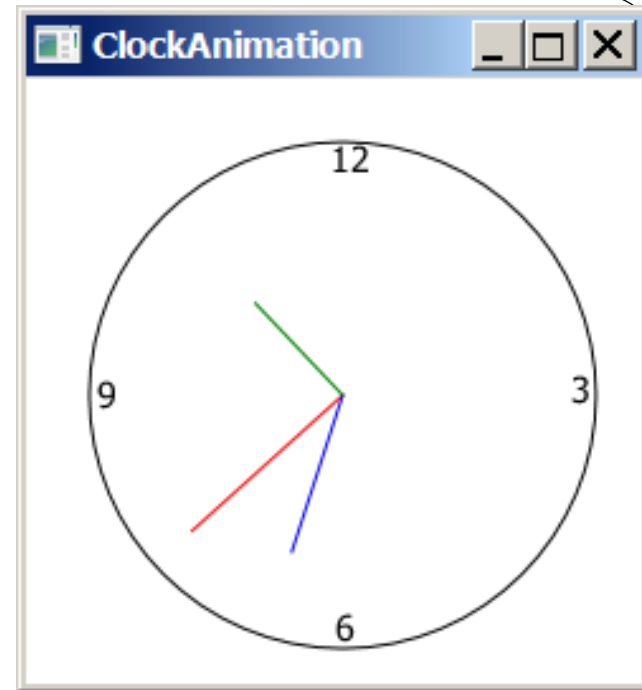
```



```
// 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();
}
```


Clock Animation

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.util.Duration;
public class ClockAnimation extends Application {
    @Override
    public void start(Stage primaryStage) {
        ClockPane clock = new ClockPane(); // Create a clock
        // Create a handler for animation
        EventHandler<ActionEvent> eventHandler = e -> {
            clock.setCurrentTime(); // Set a new clock time
        };
        // Create an animation for a running clock
        Timeline animation = new Timeline(
            new KeyFrame(Duration.millis(1000), eventHandler));
        animation.setCycleCount(Timeline.INDEFINITE);
        animation.play(); // Start animation
        Scene scene = new Scene(clock, 250, 250);
        primaryStage.setTitle("ClockAnimation");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



```
// ClockPane:
```

```
import java.util.Calendar;
import java.util.GregorianCalendar;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
public class ClockPane extends Pane {
    private int hour;
    private int minute;
    private int second;
    // Clock pane's width and height
    private double w = 250, h = 250;
    public ClockPane() {
        setCurrentTime();
    }
    public ClockPane(int hour, int minute, int second) {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
        paintClock();
    }
    public int getHour() {
        return hour;
    }
    public void setHour(int hour) {
        this.hour = hour;
        paintClock();
    }
}
```

```

public int getMinute() {
    return minute;
}
public void setMinute(int minute) {
    this.minute = minute;
    paintClock();
}
public int getSecond() {
    return second;
}
public void setSecond(int second) {
    this.second = second;
    paintClock();
}
public double getW() {
    return w;
}
public void setW(double w) {
    this.w = w;
    paintClock();
}
public double getH() {
    return h;
}
public void setH(double h) {
    this.h = h;
    paintClock();
}
public void setCurrentTime() {
    Calendar calendar = new GregorianCalendar();
    this.hour = calendar.get(Calendar.HOUR_OF_DAY);
    this.minute = calendar.get(Calendar.MINUTE);
    this.second = calendar.get(Calendar.SECOND);
    paintClock(); // Repaint the clock
}

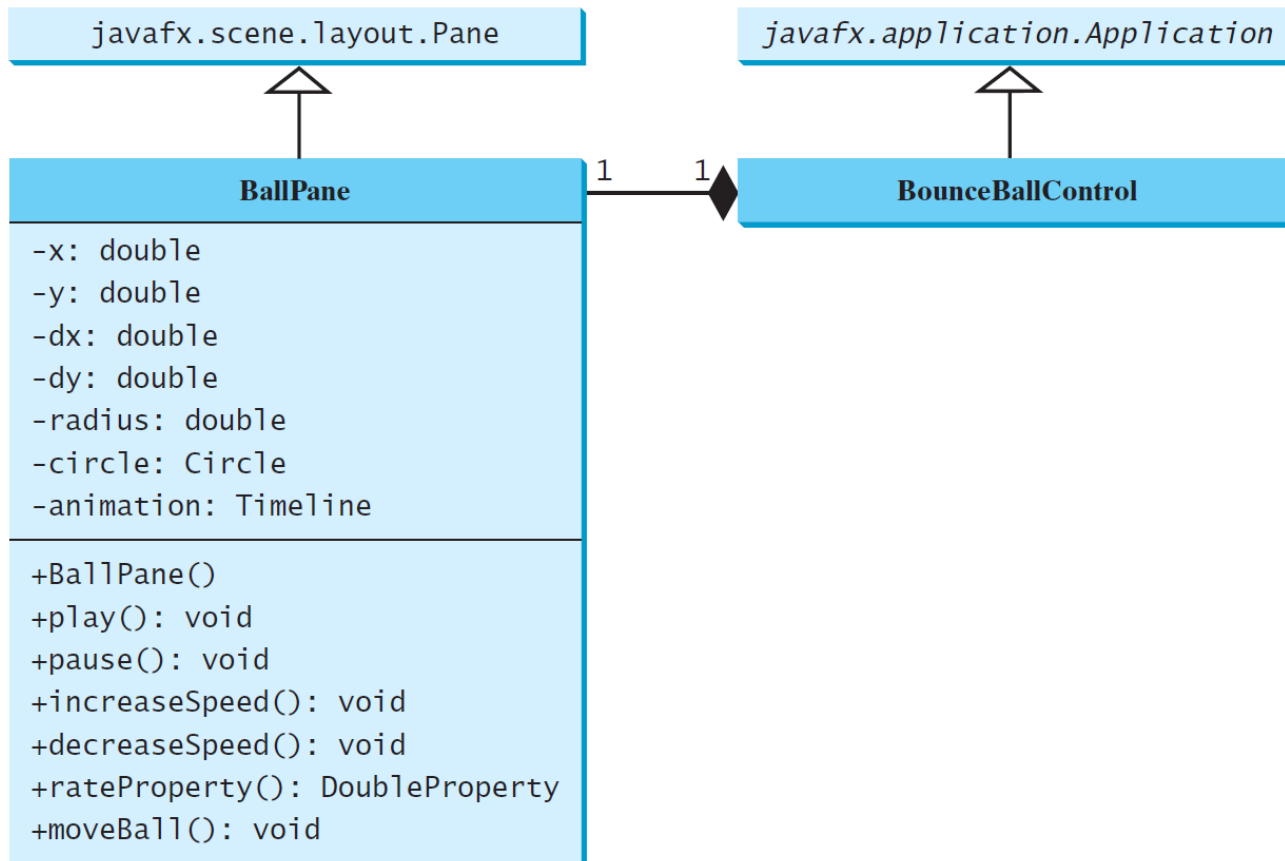
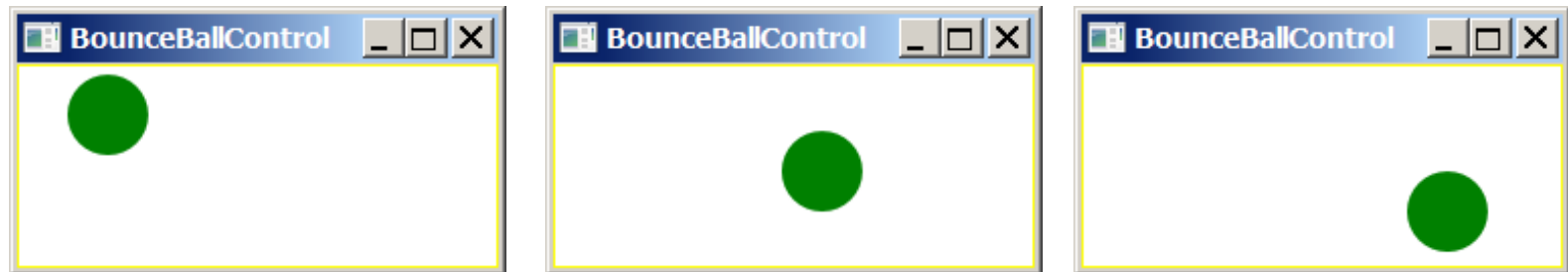
```

```

private void paintClock() {
    // Initialize clock parameters
    double clockRadius = Math.min(w, h) * 0.8 * 0.5;
    double centerX = w / 2;
    double centerY = h / 2;
    // Draw circle
    Circle circle = new Circle(centerX, centerY, clockRadius);
    circle.setFill(Color.WHITE);
    circle.setStroke(Color.BLACK);
    Text t1 = new Text(centerX - 5, centerY - clockRadius + 12, "12");
    Text t2 = new Text(centerX - clockRadius + 3, centerY + 5, "9");
    Text t3 = new Text(centerX + clockRadius - 10, centerY + 3, "3");
    Text t4 = new Text(centerX - 3, centerY + clockRadius - 3, "6");
    // Draw second hand
    double sLength = clockRadius * 0.8;
    double secondX = centerX + sLength * Math.sin(second * (2 * Math.PI / 60));
    double secondY = centerY - sLength * Math.cos(second * (2 * Math.PI / 60));
    Line sLine = new Line(centerX, centerY, secondX, secondY);
    sLine.setStroke(Color.RED);
    // Draw minute hand
    double mLength = clockRadius * 0.65;
    double xMinute = centerX + mLength * Math.sin(minute * (2 * Math.PI / 60));
    double minuteY = centerY - mLength * Math.cos(minute * (2 * Math.PI / 60));
    Line mLine = new Line(centerX, centerY, xMinute, minuteY);
    mLine.setStroke(Color.BLUE);
    // Draw hour hand
    double hLength = clockRadius * 0.5;
    double hourX = centerX + hLength * Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
    double hourY = centerY - hLength * Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12));
    Line hLine = new Line(centerX, centerY, hourX, hourY);
    hLine.setStroke(Color.GREEN);
    getChildren().clear();
    getChildren().addAll(circle, t1, t2, t3, t4, sLine, mLine, hLine);
}

```

Bouncing Ball



```

import javafx.scene.layout.Pane;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.beans.property.DoubleProperty;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.util.Duration;
public class BallPane extends Pane {
    public final double radius = 20;
    private double x = radius, y = radius;
    private double dx = 1, dy = 1;
    private Circle circle = new Circle(x, y, radius);
    private Timeline animation;
    public BallPane() {
        circle.setFill(Color.GREEN); // Set ball color
        getChildren().add(circle); // Place a ball into this pane
        // Create an animation for moving the ball
        animation = new Timeline(new KeyFrame(Duration.millis(50), e -> moveBall()));
        animation.setCycleCount(Timeline.INDEFINITE);
        animation.play(); // Start animation
    }
    public void play() {
        animation.play();
    }
    public void pause() {
        animation.pause();
    }
    public void increaseSpeed() {
        animation.setRate(animation.getRate() + 0.1);
    }
    public void decreaseSpeed() {
        animation.setRate(
            animation.getRate() > 0 ? animation.getRate() - 0.1 : 0);
    }
}

```

```
public DoubleProperty rateProperty() {  
    return animation.rateProperty();  
}  
  
protected void moveBall() {  
    // Check boundaries  
    if (x < radius || x > getWidth() - radius) {  
        dx *= -1; // Change ball move direction  
    }  
    if (y < radius || y > getHeight() - radius) {  
        dy *= -1; // Change ball move direction  
    }  
  
    // Adjust ball position  
    x += dx;  
    y += dy;  
    circle.setCenterX(x);  
    circle.setCenterY(y);  
}  
}
```

```

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.input.KeyCode;
public class BounceBallControl extends Application {
    @Override
    public void start(Stage primaryStage) {
        BallPane ballPane = new BallPane(); // Create a ball pane
        // Pause and resume animation
        ballPane.setOnMousePressed(e -> ballPane.pause());
        ballPane.setOnMouseReleased(e -> ballPane.play());
        // Increase and decrease animation
        ballPane.setOnKeyPressed(e -> {
            if (e.getCode() == KeyCode.UP) {
                ballPane.increaseSpeed();
            } else if (e.getCode() == KeyCode.DOWN) {
                ballPane.decreaseSpeed();
            }
        });
        Scene scene = new Scene(ballPane, 250, 150);
        primaryStage.setTitle("BounceBallControl");
        primaryStage.setScene(scene);
        primaryStage.show();
        // Must request focus after the primary stage is displayed
        ballPane.requestFocus();
    }

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

```


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/toucheventsjava.htm>