Unit 7
JAVAFX basics, Event-driven programming and Animations

Prof. Ravikumar R Natarajan

Dept. of CE

Contents

- **Basic structure of JAVAFX program,**
- **Panes,**
- **UI control and shapes,**
- **Property binding,**
- **The Color and the Font class,**
- **The Image and Image-View class,**
- **Layout panes and shapes,**
- Events and Events sources,
- Registering Handlers and Handling Events,
- Inner classes,
- Anonymous inner class handlers,
- Mouse and key events,
- Listeners for observable objects,
- Animation

Introduction

- When Java was introduced, the GUI (Graphical User Interface) classes were bundled in a library known as the Abstract Windows Toolkit (AWT).
- AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
- In addition, AWT is prone to platform-specific bugs.
- The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as Swing components.

Introduction

- Swing components are painted directly on canvases using Java code.
- Swing components depend less on the target platform and use less of the native GUI resources.
- **Swing is designed for developing desktop GUI applications.**
- It is now replaced by a completely new GUI platform known as JavaFX.
- JavaFX incorporates modern GUI technologies to enable you to develop rich Internet applications.

Introduction to JAVAFX

- **JavaFX is a new framework for developing Java GUI programs.**
- A JavaFX application can run seamlessly on a desktop and from a Web browser.
- Additionally, JavaFX provides a multi-touch support for touch- enabled devices such as tablets and smart phones.
- JavaFX has a built-in 2D, 3D, animation support, video and audio playback.
- **JavaFX is an excellent pedagogical tool for learning object-oriented programming.**

Basic structure of JAVAFX program

- The abstract class `javafx.application.Application` defines the essential framework for writing JavaFX programs.
- **Every JavaFX program is defined in a class that extends** `javafx.application.Application.`
- A JavaFX program can run stand-alone or from a Web browser.

Run a program – CLI (linux)

- Export Path to the library JAR files

```
export PATH_TO_FX=path/to/javafx-
sdk/lib
```

- Compile your program

```
javac --module-path
$PATH_TO_FX --add-modules
javafx.controls
MyJavaFX.java
```

- Run your program

```
java --module-path
$PATH_TO_FX --add-modules
javafx.controls MyJavaFX
```

Run a program – CLI (Windows)

- Export Path to the library JAR files

```
set PATH_TO_FX="path\to\javafx-
sdk\lib"
```

- Compile your program

```
javac --module-path
%PATH_TO_FX% --add-modules
javafx.controls
MyJavaFX.java
```

- Run your program

```
java --module-path
%PATH_TO_FX% --add-modules
javafx.controls MyJavaFX
```

Run a program - IDE (Eclipse)

- Create a new User Library
- Include the jars under the lib folder from JavaFX
- Add user library to your project
- Add VM arguments:

Linux: `--module-path /path/to/javafx-sdk/lib --add-modules javafx.controls`

Windows: `--module-path "\path\to\javafx-sdk\lib" --add-modules javafx.controls`

JavaFX Application

- The main class overrides the start method defined in `javafx.application.Application`.
- After a JavaFX application is launched, the JVM constructs an instance of the class using its no-arg constructor and invokes its start method.
- The start method normally places UI controls in a scene and displays the scene in a stage.

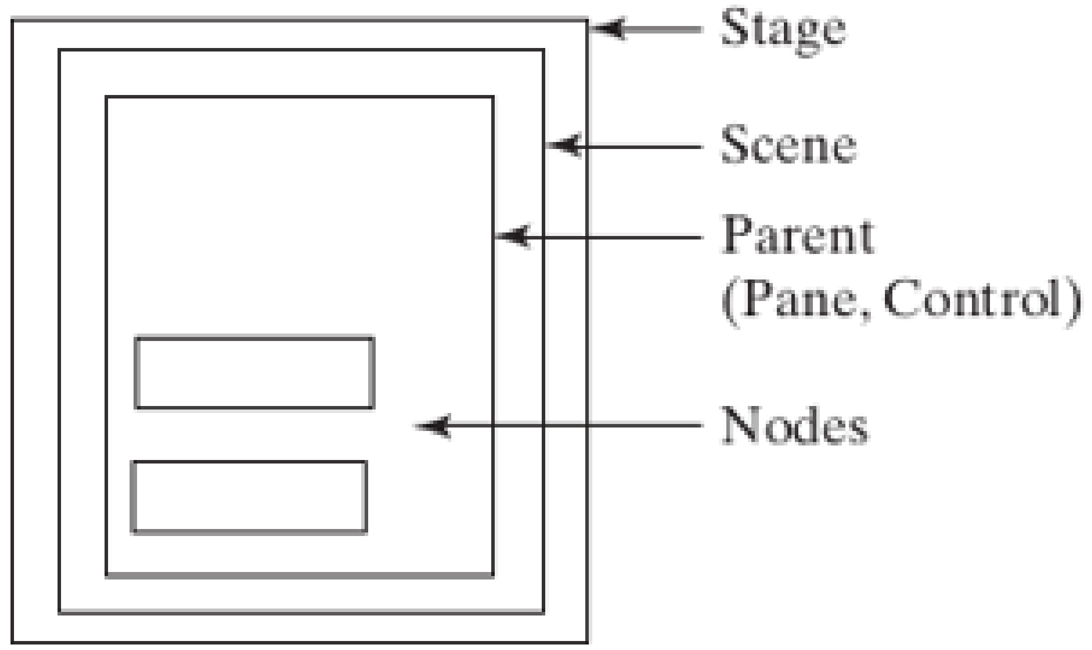
JavaFX Application

- A Scene object can be created using the constructor Scene(node, width, height).
- This constructor specifies the width and height of the scene and places the node in the scene.

- A Stage object is a window.
- A Stage object called primary stage is automatically created by the JVM when the application is launched.
- You can create additional stages if needed.

JavaFX Application

Panes are used to hold nodes

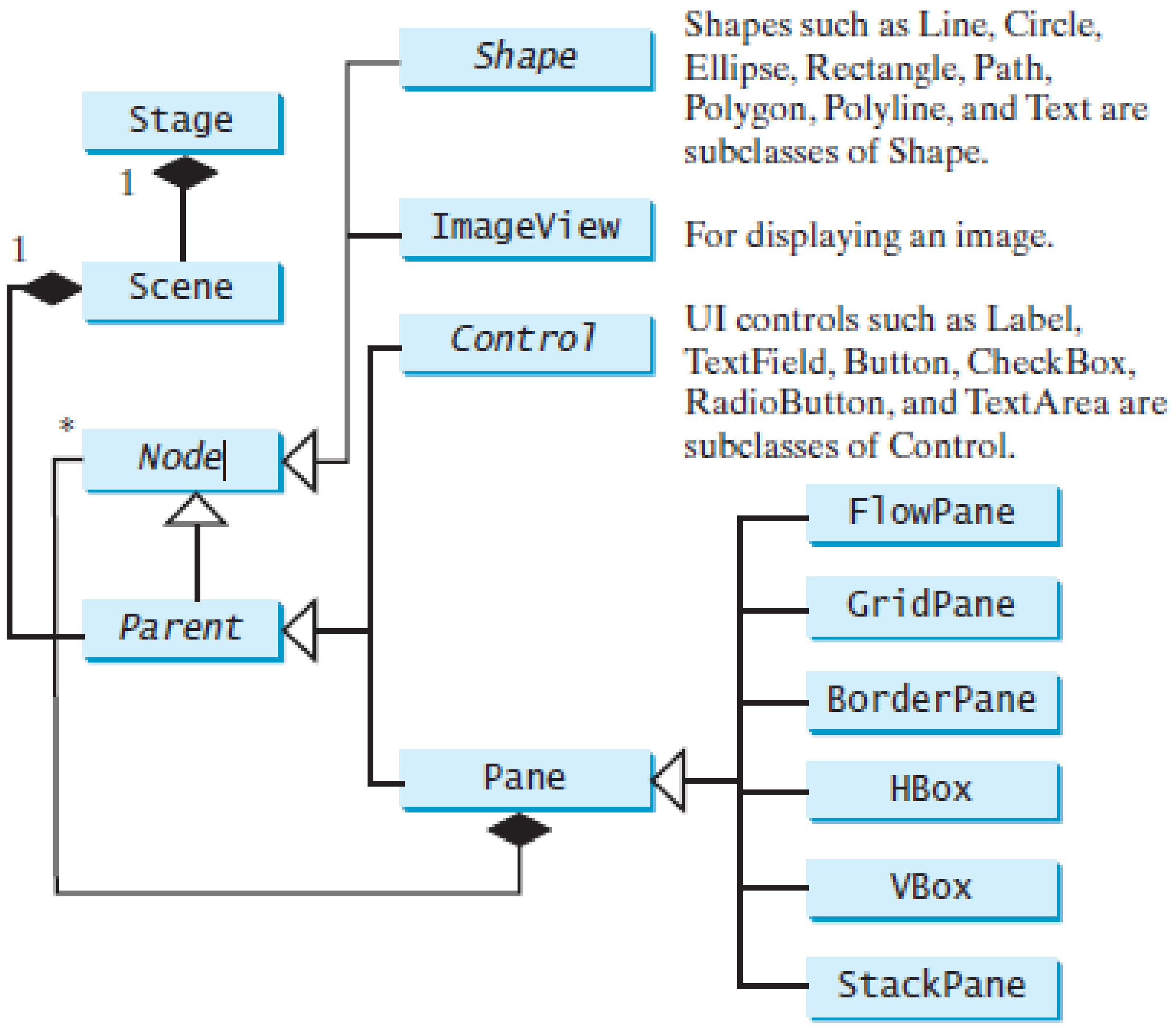


Source: Introduction to Java Programming, By: Y. Daniel Liang.

Panes, UI Control and Shapes

- **Panes, UI controls, and shapes are subtypes of Node.**
- Nodes without container occupies the entire window, So a better approach is to use container classes, called panes, for automatically laying out the nodes in a desired location and size.
- A node is a visual component such as a shape, an image view, a UI control, or a pane.
- A shape refers to a text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.
- A UI control refers to a label, button, check box, radio button, text field, text area, etc.

Panes, UI Control and Shapes


Stage
1
1
Scene
*
Node
Parent
Shape
ImageView
Control
Pane
FlowPane
GridPane
BorderPane
HBox
VBox
StackPane
Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.
For displaying an image.
UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.

Nodes can be shapes, image views, UI controls, and panes

StackPane and Button

```
 public void start(Stage primaryStage) {

 StackPane pane = new StackPane();

 pane.getChildren().add(new Button("OK"));

 Scene scene = new Scene(pane, 200, 50);
   // Set the stage title
   primaryStage.setTitle("Button in a pane");

   // Place the scene in the stage
   primaryStage.setScene(scene);

   primaryStage.show(); // Display the stage
}
```

Pane and Circle

```
// Create a circle and set its properties
Circle circle = new Circle();
circle.setCenterX(100);
circle.setCenterY(100);
circle.setRadius(50);
circle.setStroke(Color.BLACK;
circle.setFill(Color.RED);
```

Pane and Circle (Cont.)

```
// Create a pane to hold the circle
Pane pane = new Pane();
pane.getChildren().add(circle);
// Create a scene and place it in the stage and show

Scene scene = new Scene(pane, 200, 200);
primaryStage.setTitle("ShowCircle");
primaryStage.setScene(scene);
primaryStage.show();
```

Simple JavaFX Program to Display Button

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
public class MyJavaFX extends Application {
public void start(Stage primaryStage) {
Button b = new Button("OK");
Pane r = new Pane();
r.getChildren().addAll(b);
Scene scene = new Scene(b, 200, 250);
 primaryStage.setTitle("MyJavaFX"); // Set the stage title
 primaryStage.setScene(scene); // Place the scene in the stage
 primaryStage.show(); // Display the stage
 }
public static void main(String[] args) {
 Application.launch(args);
 } }
```

Property binding

- You can bind a target object to a source object.
- **A change in the source object will be automatically reflected in the target object.**
- JavaFX introduces a new concept called property binding that enables a **target object to be bound to a source object**.
- The target object is called a **binding** object or a binding property and the source object is called a **bindable** object or observable object.

Property binding

- If the value in the source object changes, the target object is also changed automatically.
- Example:
- circle.centerXProperty().bind(pane.widthProperty().divide(2));
- circle.centerYProperty().bind(pane.heightProperty().divide(2));
- Refer reference program: 1

Common Properties and Methods for Nodes

- The abstract Node class defines many properties and methods that are common to all nodes.
- Style:
- JavaFX style properties are similar to cascading style sheets (CSS) used to specify the styles for HTML elements in a Web page. So, the style properties in JavaFX are called JavaFX CSS.
- In JavaFX, a style property is defined with a prefix –fx- .
- **circle.setStyle("-fx-stroke: black; -fx-fill: red;");**

Common Properties and Methods for Nodes

- Rotate:
- The rotate property enables you to specify an angle in degrees for rotating the node from its center.
- If the degree is positive, the rotation is performed clockwise; otherwise, it is performed counterclockwise.
- button.setRotate(80);

The Color Class

- JavaFX defines the **abstract Paint class** for painting a node.
- The **javafx.scene.paint.Color** is a concrete subclass of Paint, which is used to encapsulate colors.
- The Color class can be used to create colors.

The Color Class methods

- Color(r: double, g: double, b: double, opacity: double)
- brighter(): Color
- darker(): Color
- color(r: double, g: double, b: double): Color
- color(r: double, g: double, b: double, opacity: double): Color
- rgb(r: int, g: int, b: int): Color
- rgb(r: int, g: int, b: int, opacity: double): Color
- +getter methods…

The Color Class

- The Color class is immutable. Once a Color object is created, its properties cannot be changed.

- Many standard colors such as BEIGE, BLACK, BLUE, BROWN, CYAN, DARKGRAY, GOLD, GRAY, GREEN, LIGHTGRAY, MAGENTA, NAVY, ORANGE, PINK, RED, SILVER, WHITE, and YELLOW defined as
- constants in the Color class.

The Font Class

- A Font class describes font name, weight, and size.
- You can set fonts for rendering the text.
- The javafx.scene.text.Font class is used to create fonts.
- A Font is defined by its name, weight, posture, and size.
- Name: Times, Courier, and Arial are the examples of the font names.
- You can obtain a list of available font family names by invoking the static getFamilies() method.
- Posture: The font postures are two constants: FontPosture.ITALIC and FontPosture.REGULAR.

The Font Class

- A Font instance can be constructed using its constructors or using its static methods.
- Example 1:
- **Font font1 = new Font("SansSerif", 16);**
- **Font font2 = Font.font("Times New Roman", FontWeight.BOLD, FontPosture.ITALIC, 12);**
- Example 2:
- **Label label = new Label("JavaFX");** label.setFont(Font.font("Times New Roman", FontWeight.BOLD,
- FontPosture.ITALIC, 20));
- pane.getChildren().add(label);

The Font Class Methods

- Font(size: double)
- Font(name: String, size: double)
- font(name: String, size: double)
- font(name: String, w: FontWeight, size: double)
- font(name: String, w: FontWeight, p: FontPosture, size: double)
- getFamilies(): List<String>
- getFontNames(): List<String>
- +getter methods…

The Image and ImageView Classes

- The Image class represents a graphical image and the **ImageView class can be used to display an image.**
- The **javafx.scene.image.Image** class represents a graphical image and is used for loading an image from a specified filename or a URL.
- The javafx.scene.image.ImageView is a node for displaying an image.
- An ImageView can be created from an Image object.

The Image and ImageView Classes

- › Example:
- Image image = new Image("images/avatar.jpg");
- ImageView imageView = new ImageView(image);

- › Alternatively, you can create an ImageView directly from a file or a URL as follows:
- ImageView imageView = new ImageView("images/avatar.jpg");

Layout Panes

- JavaFX provides many types of panes for automatically laying out nodes in a desired location and size.
- Example:
- Pane, StackPane, FlowPane, GridPane, BorderPane, HBox, VBox.

Layout Panes

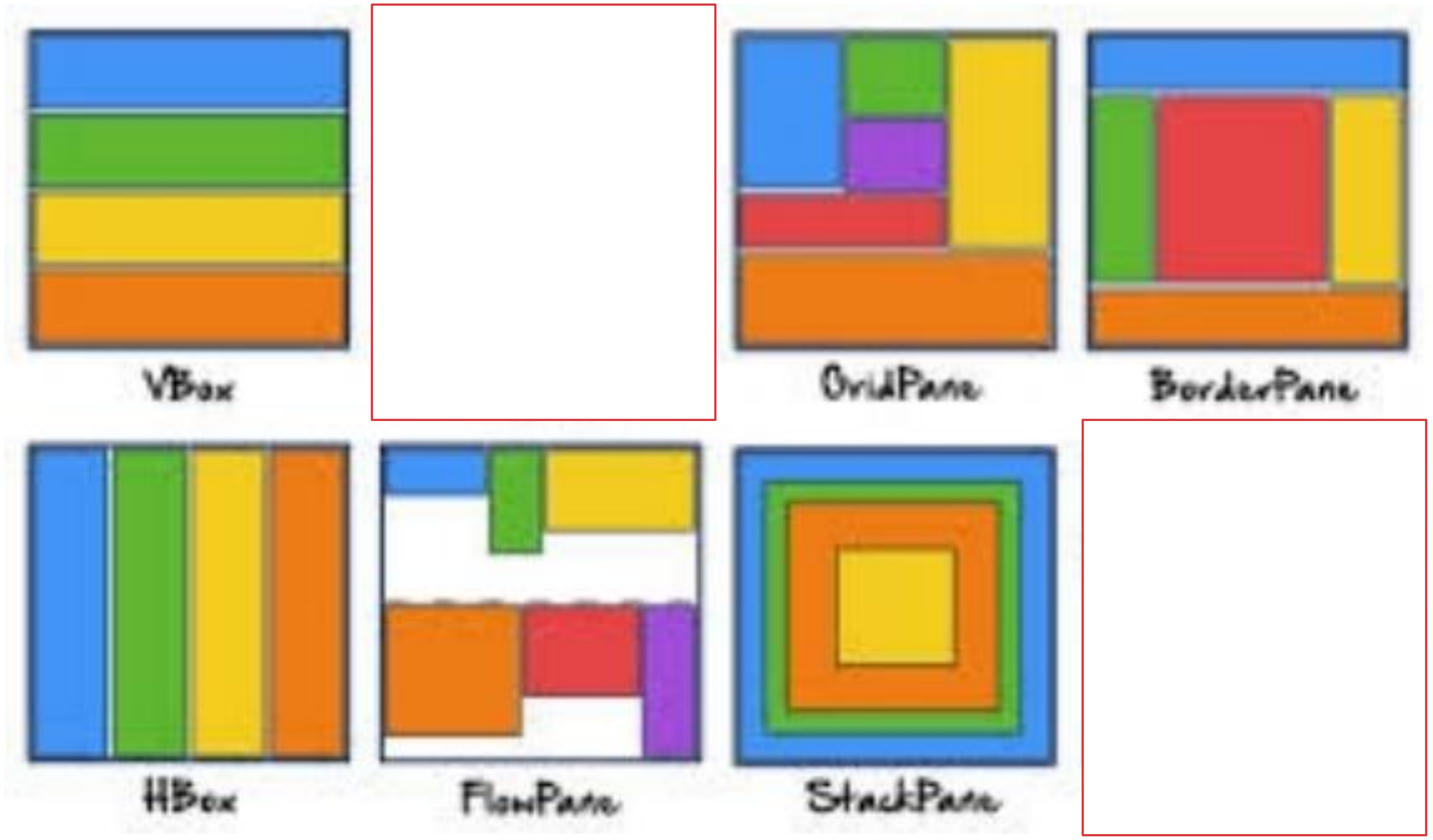
Class	Description
Pane	Base class for layout panes. It contains the getChildren() method for returning a list of nodes in the pane.
StackPane	Places the nodes on top of each other in the center of the pane.
FlowPane	Places the nodes row-by-row horizontally or column-by-column vertically.
GridPane	Places the nodes in the cells in a two-dimensional grid.
BorderPane	Places the nodes in the top, right, bottom, left, and center regions.
HBox	Places the nodes in a single row.
VBox	Places the nodes in a single column.

Layout Panes

Class	Description
Pane	Base class for layout panes. It contains the getChildren() method for returning a list of nodes in the pane.
StackPane	Places the nodes on top of each other in the center of the pane.
FlowPane	Places the nodes row-by-row horizontally or column-by-column vertically.
GridPane	Places the nodes in the cells in a two-dimensional grid.
BorderPane	Places the nodes in the top, right, bottom, left, and center regions.
HBox	Places the nodes in a single row.
VBox	Places the nodes in a single column.

Layout Panes

- Syntax: FlowPane pane = new FlowPane();
- Similarly for rest of the panes the class name will be changed.



Shapes

- **JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.**
- The Shape class is the abstract base class that defines the common properties for all shapes.
- Properties like: fill, stroke, and strokeWidth.
- We will look at Text, Line, Circle, Rectangle, Ellipses, Arc, Polygon, and Polyline Classes.
- All these are subclasses of Shape.

Upcoming Classes

- Text Class
- Line Class
- Rectangle Class
- Circle Class
- Ellipses Class
- Arc Class
- Polygon/Polyline Class

Text Class

- The Text class defines a node that displays a string at a starting point (x,y).
- A Text object is usually placed in a pane.
- The pane’s upper-left corner point is (0,0).
- A string may be displayed in multiple lines separated by \n.

Text Class Properties

- text: StringProperty // Defines the text to be displayed.
- x: DoubleProperty // Defines the x-coordinate of text (default 0).
- y: DoubleProperty // Defines the y-coordinate of text (default 0).
- underline: BooleanProperty // Defines if each line has an underline (default false).
- strikethrough: BooleanProperty // Defines if each line has a line through it (default false).
- font: ObjectProperty<Font> // Defines the font for the text.

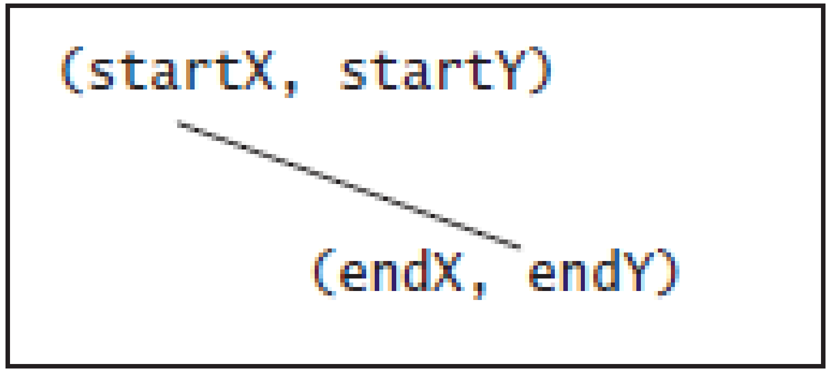
Text Class Methods

- Text() // Creates an empty Text.
- Text(text: String) // Creates a Text with the specified text.
- Text(x: double, y: double, text: String) // Creates a Text with the specified x and y - coordinates and text.
- + getter and setter methods.

Line Class: Properties, Methods

- startX: DoubleProperty // The x-coordinate of the start point.
- startY: DoubleProperty // The y-coordinate of the start point.
- endX: DoubleProperty // The x-coordinate of the end point.
- endY: DoubleProperty // The y-coordinate of the end point.

- Line() // Creates an empty Line.
- Line(startX: double, startY:double, endX: double, endY:double) // Creates a Line with the specified starting and ending points.
- +getter and setter methods



Rectangle Class

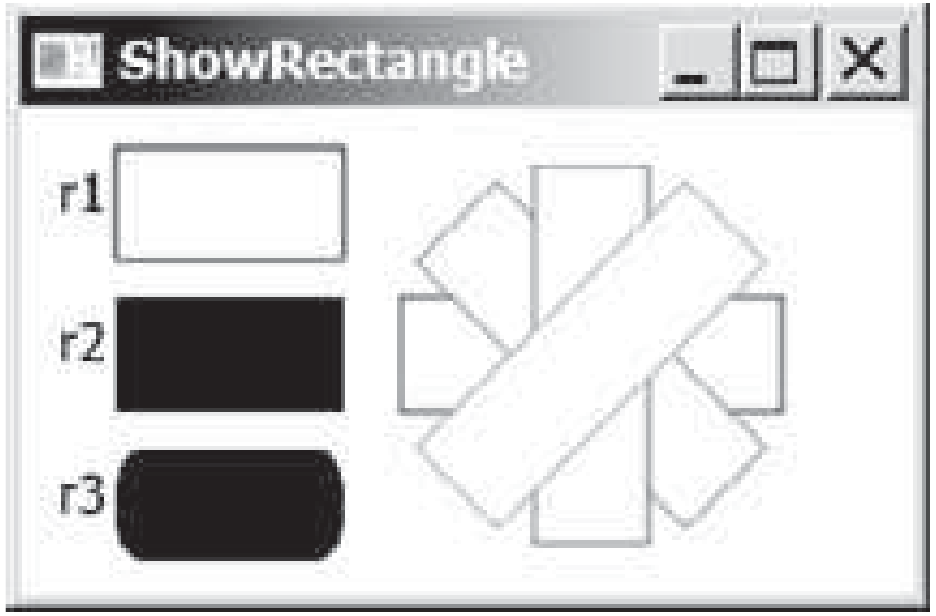
- A rectangle is defined by the parameters x, y, width, height, arcWidth, and arcHeight.
- The rectangle's upper-left corner point is at (x,y)
- Parameter aw (arcWidth) is the horizontal diameter of the arcs at the corner.
- Parameter ah (arcHeight) is the vertical diameter of the arcs at the corner.

Rectangle Class Properties

- x: DoubleProperty // The x-coordinate of the upper-left corner of the rectangle (default 0).
- y:DoubleProperty // The y-coordinate of the upper-left corner of the rectangle (default 0).
- width: DoubleProperty // The width of the rectangle (default: 0).
- height: DoubleProperty // The height of the rectangle (default: 0).
- arcWidth: DoubleProperty // The arcWidth of the rectangle (default: 0).
- arcHeight: DoubleProperty // The arcHeight of the rectangle (default: 0).

Rectangle Class Methods

- Rectangle() // Creates an empty Rectangle.
- Rectanlge(x: double, y:double, width: double, height: double) // Creates a Rectangle with the specified upper-left corner point, width, and height.
- +getter and setter methods



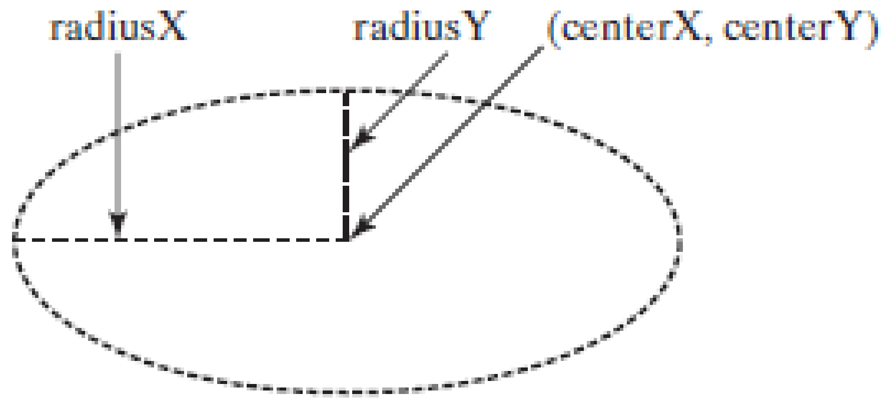
Circle Class: Properties, Methods

- centerX: DoubleProperty // The x-coordinate of the center of the circle (default 0).
- centerY: DoubleProperty // The y-coordinate of the center of the circle (default 0).
- radius: DoubleProperty // The radius of the circle (default: 0).

- Circle() // Creates an empty Circle.
- Circle(x: double, y: double) // Creates a Circle with the specified center.
- Circle(x: double, y: double, radius: double) // Creates a Circle with the specified center and radius.
- +getter and setter methods

Ellipses Class: Properties

- centerX: DoubleProperty // The x-coordinate of the center.
- centerY: DoubleProperty // The y-coordinate of the center.
- radiusX: DoubleProperty // The horizontal radius of the ellipse.
- radiusY: DoubleProperty // The vertical radius of the ellipse.
- Default value for all parameters are 0.



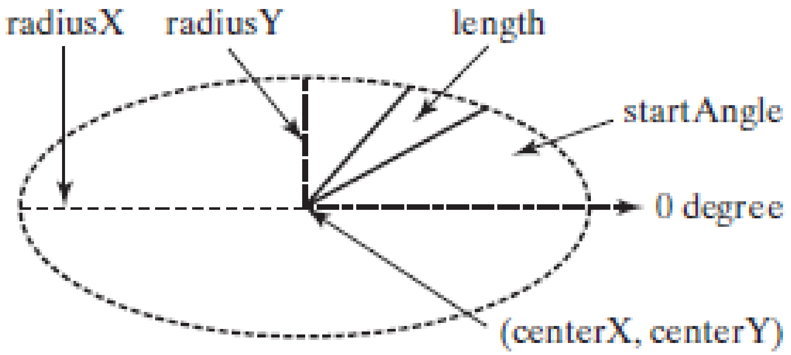
(a) Ellipse(centerX, centerY, radiusX, radiusY)

Ellipses Class: Methods

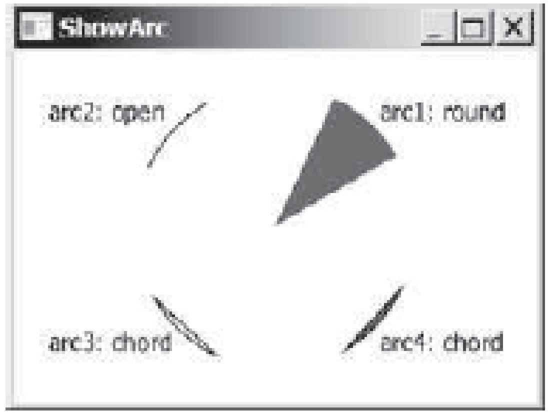
- Ellipse() // Creates an empty Ellipse.
- Ellipse(x: double, y: double) // Creates an Ellipse with the specified center.
- Ellipse(x: double, y: double, radiusX: double, radiusY: double) // Creates an Ellipse with the specified center and radiuses.
- +getter and setter methods

Arc Class

- An arc is conceived as part of an ellipse.
- Defined by the parameters centerX, centerY, radiusX, radiusY, startAngle, length, and an arc type (ArcType.OPEN, ArcType.CHORD, or ArcType.ROUND).
- The parameter startAngle is the starting angle; and length is the spanning angle.
- Angles are measured in degrees and follow the usual mathematical conventions.



(a) `Arc(centerX, centerY, radiusX, radiusY, startAngle, length)`



(b) Multiple ellipses are displayed

Arc Class Properties

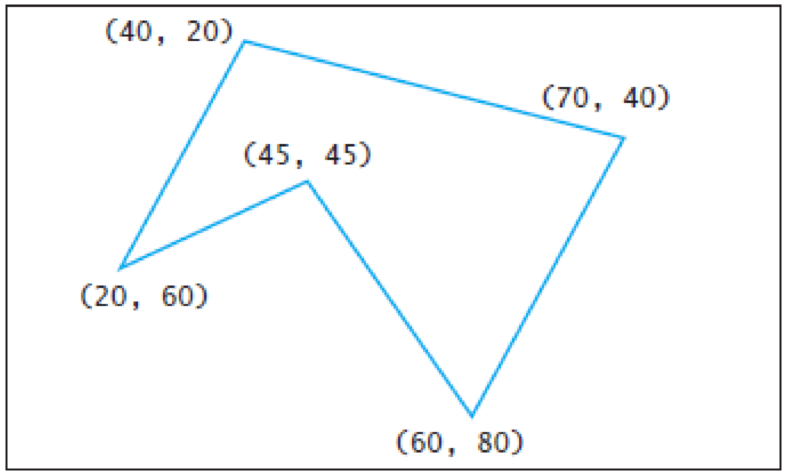
- centerX: DoubleProperty // The x-coordinate of the center.
- centerY: DoubleProperty // The y-coordinate of the center.
- radiusX: DoubleProperty // The horizontal radius of the ellipse.
- radiusY: DoubleProperty // The vertical radius of the ellipse.
- startAngle: DoubleProperty // The start angle of the arc in degrees.
- length: DoubleProperty // The angular extent of the arc in degrees.
- type: ObjectProperty<ArcType> // The closure type of the arc (ArcType.OPEN, ArcType.CHORD, ArcType.ROUND).

Arc Class Methods

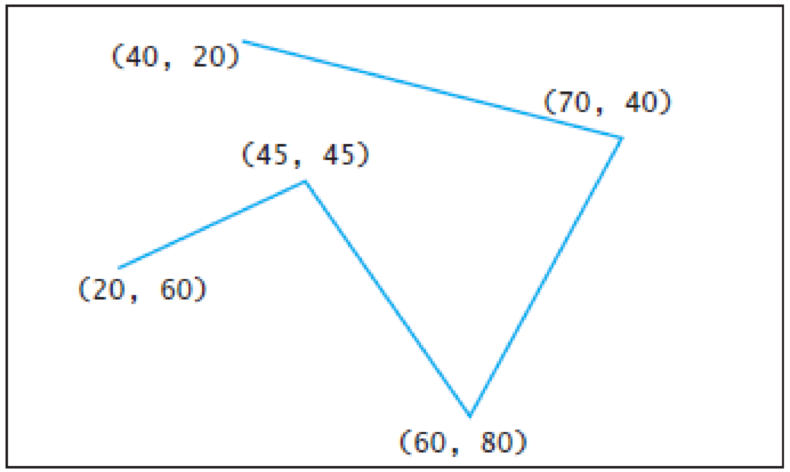
- Arc() // Creates an empty Arc.
- Arc(x: double, y: double, radiusX: double, radiusY: double, startAngle: double, length: double) // Creates an Arc with the specified arguments.
- +getter and setter methods

Polygon/Polyline Class

- › The Polygon class defines a polygon that connects a sequence of points.
- › The Polyline class is similar to the Polygon class except that the Polyline class is not automatically closed.



(a) Polygon



(b) Polyline

Polygon/Polyline Class Methods

- Polygon() // Creates an empty Polygon.
- Polygon(double... points) // Creates a Polygon with the given points.
- getPoints(): ObservableList<Double> // Returns a list of double values as x and y coordinates of the points.
- +getter and setter methods

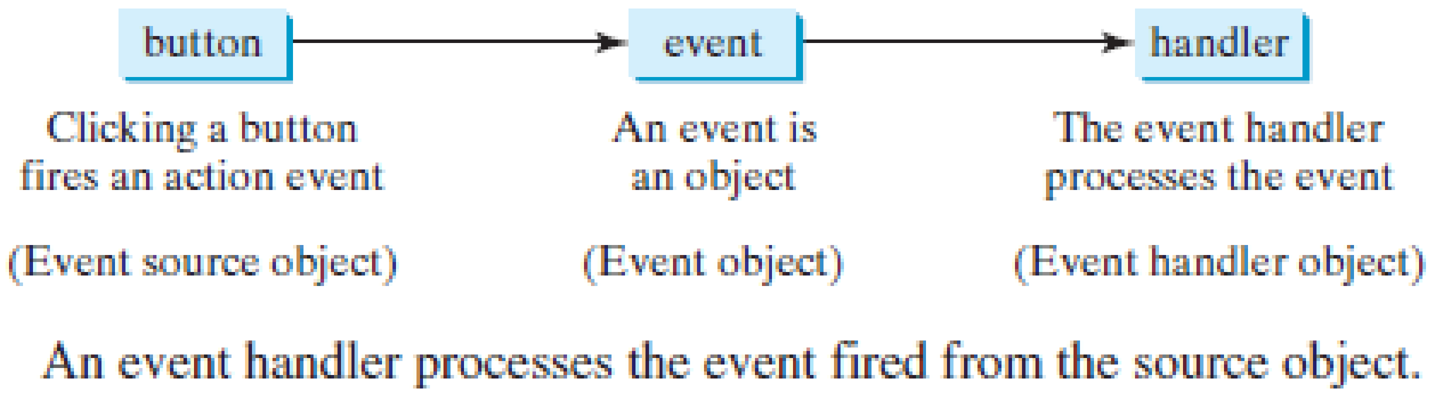
Event Driven Programming

Event-driven programming

- **You can write code to process events such as a button click, mouse movement, and keystrokes.**
- **When we run a Java GUI program, the program interacts with the user, and the events drive its execution. This is called event-driven programming.**
- An event can be defined as a signal to the program that something has happened.
- Events are triggered by external user actions, such as button click, mouse movement, mouse clicks, keystrokes etc.
- We can write code to process such events.
- The program can choose to respond to or ignore an event.

Events and Events sources

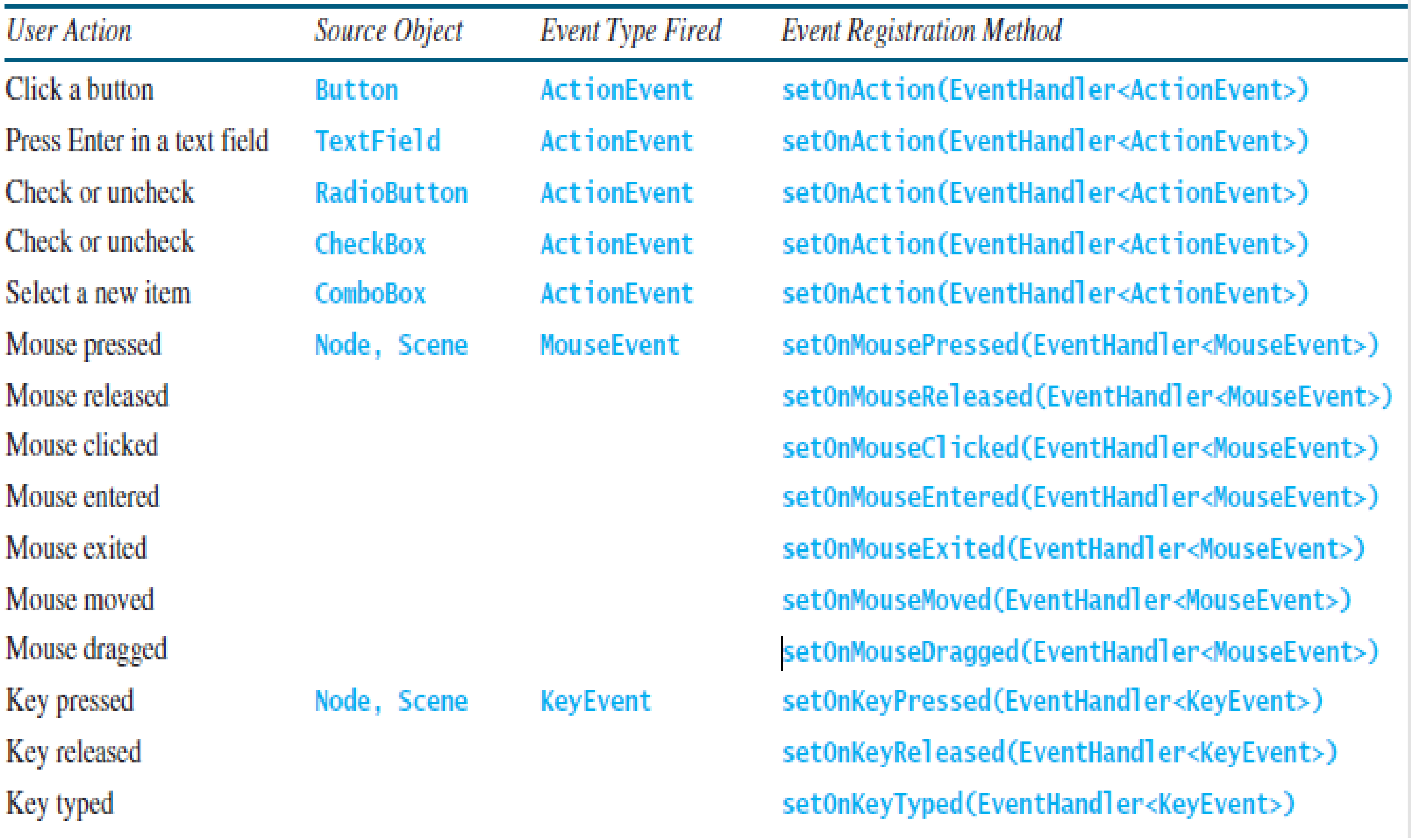
- **An event is an object created from an event source.**
- Firing an event means to create an event and delegate the handler to handle the event.
- The component that creates an event and fires it is called the event source object, or simply source object or source component.



An event handler processes the event fired from the source object.

Source: Introduction to Java Programming, By: Y. Daniel Liang.

Events and Events sources



User Action	*Source Object*	*Event Type Fired*	*Event Registration Method*
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>)`

Registering Handlers and Handling Events

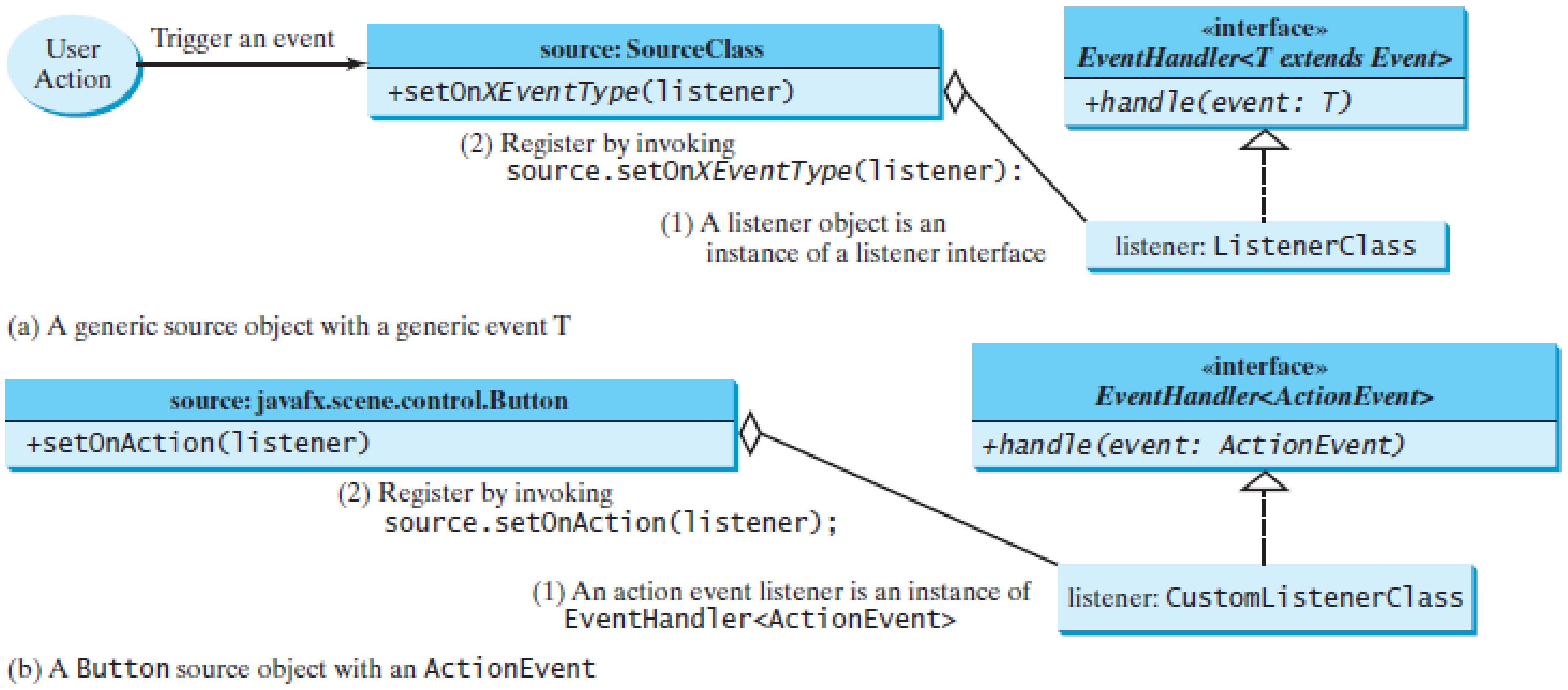
- **A handler/listener is an object that must be registered with an event source object, and it must be an instance of an appropriate event-handling interface.**
- Java uses a delegation-based model for event handling: a source object fires an event, and an object interested in the event handles it. The latter object is called an event handler or an event listener.
- Not all objects can be handler/listener for an event.
- To be a handler/listener of an event, two requirements must be met.

Registering Handlers and Handling Events

1. The handler object must be an instance of the corresponding event-handler interface to ensure that the handler has the correct method for processing the event.
2. The EventHandler object handler must be registered with the event source object using the method source.setOnAction(handler).

- › For example,
- › For ActionEvent, the method is setOnAction.
- › For a mouse pressed event, the method is setOnMousePressed.
- › For a key pressed event, the method is setOnKeyPressed.

Registering Handlers and Handling Events


User Action
Trigger an event
source: SourceClass
+setOnXEventType(listener)
«interface»
EventHandler<T extends Event>
+handle(event: T)
(2) Register by invoking
source.setOnXEventType(listener);
(1) A listener object is an instance of a listener interface
listener: ListenerClass
(a) A generic source object with a generic event T
source: javafx.scene.control.Button
+setOnAction(listener)
«interface»
EventHandler<ActionEvent>
+handle(event: ActionEvent)
(2) Register by invoking
source.setOnAction(listener);
(1) An action event listener is an instance of EventHandler<ActionEvent>
listener: CustomListenerClass
(b) A Button source object with an ActionEvent

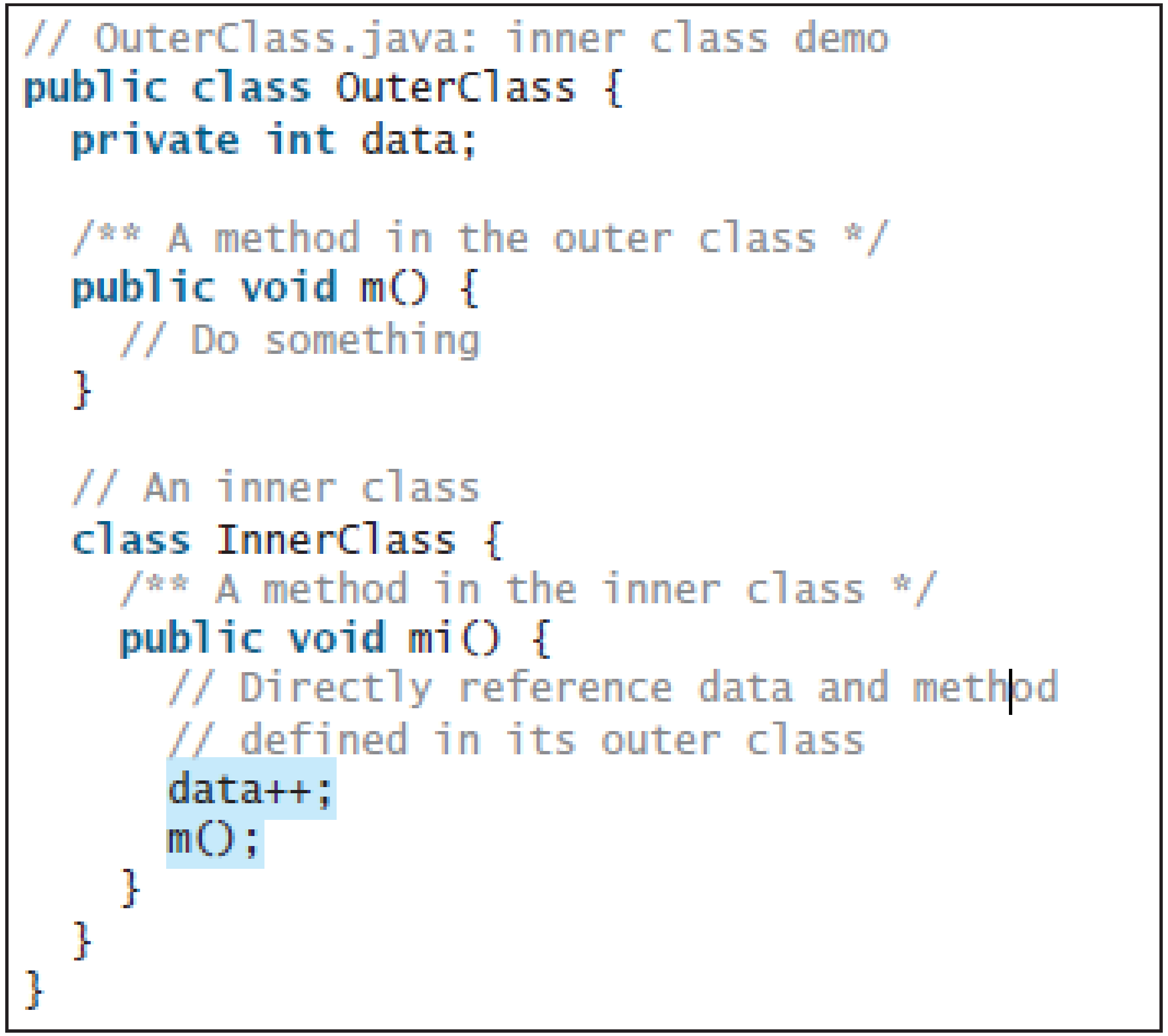
Inner Classes

- An inner class, or nested class, is a class defined within the scope of another class.
- Inner classes are useful for defining handler classes.
- Normally, we define a class as an inner class if it is used only by its outer class.

Inner Class

- **An inner class has the following features:**
- An inner class is compiled into a class named
- OuterClassName$InnerClassName.class.
- An inner class can reference the data and the methods defined in the outer class.
- An inner class can be defined with a visibility modifier subject to the same visibility rules applied to a member of the class.
- **An inner class can be defined as static . A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class.**
- Objects of an inner class are often created in the outer class. But you can also create an object of an inner class from another class.

Inner Class

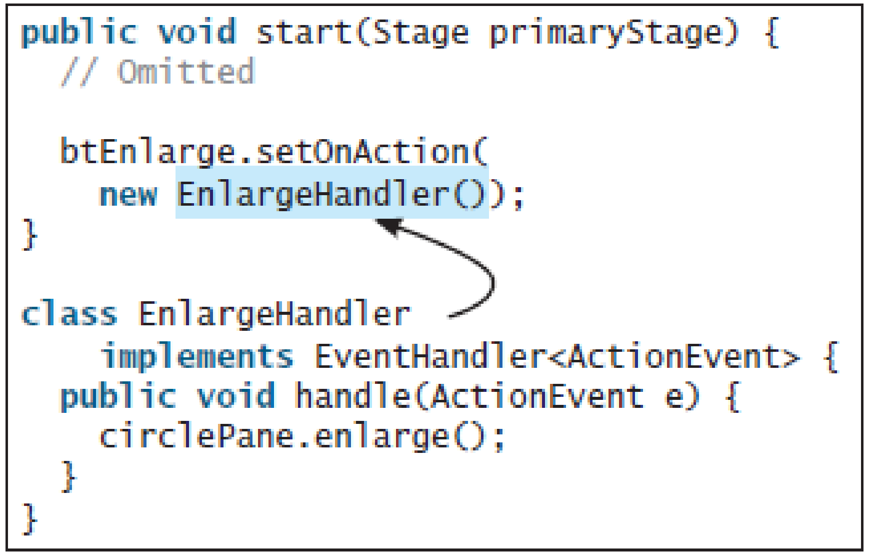


```
// OuterClass.java: inner class demo
public class OuterClass {
  private int data;

  /** A method in the outer class */
  public void m() {
    // Do something
  }

  // An inner class
  class InnerClass {
    /** A method in the inner class */
    public void mi() {
      // Directly reference data and method
      // defined in its outer class
      data++;
      m();
    }
  }
}
```

Anonymous Inner Class Handlers

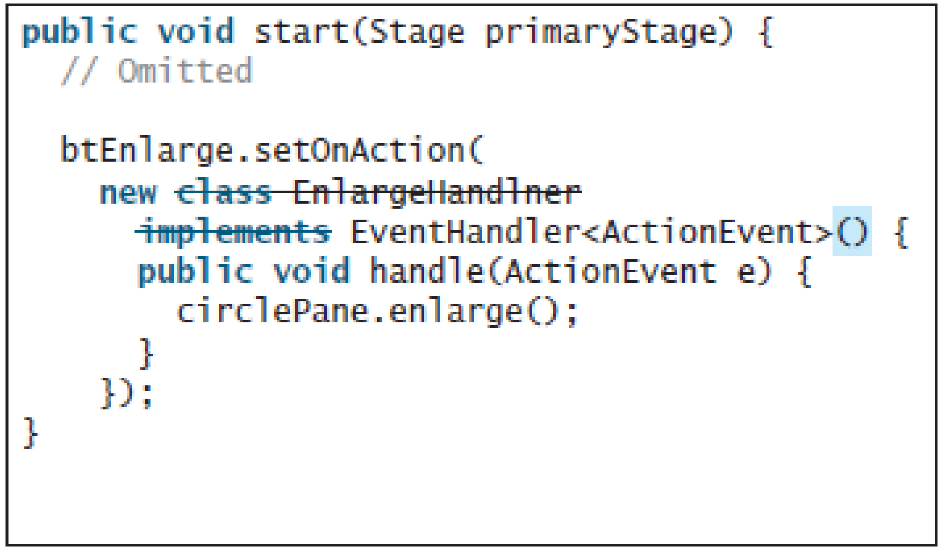
- An anonymous inner class is an inner class without a name.
- It combines defining an inner class and creating an instance of the class into one step.

Anonymous Inner Class Handlers



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

(b) Anonymous inner class

Anonymous Inner Class Handlers

- Since an anonymous inner class is a special kind of inner class, it is treated like an inner class with the following features:
- 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. For example, if the outer class Test has two anonymous inner classes, they are compiled into Test$1.class and Test$2.class.

Mouse Events

- A MouseEvent is fired whenever a mouse button is pressed, released, clicked, moved, or dragged on a node or a scene.

- MouseEvent Methods:
- getButton(): MouseButton // Indicates which mouse button has been clicked.
- getClickCount(): int // Returns the number of mouse clicks associated with this event.
- getX(): double // Returns the x-coordinate of the mouse point in the event source node.

Mouse Events

- getY(): double // Returns the y-coordinate of the mouse point in the event source node.
- getSceneX(): double // Returns the x-coordinate of the mouse point in the scene.
- getSceneY(): double // Returns the y-coordinate of the mouse point in the scene.
- getScreenX(): double // Returns the x-coordinate of the mouse point in the screen.
- getScreenY(): double // Returns the y-coordinate of the mouse point in the screen.
- 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.

Mouse Events

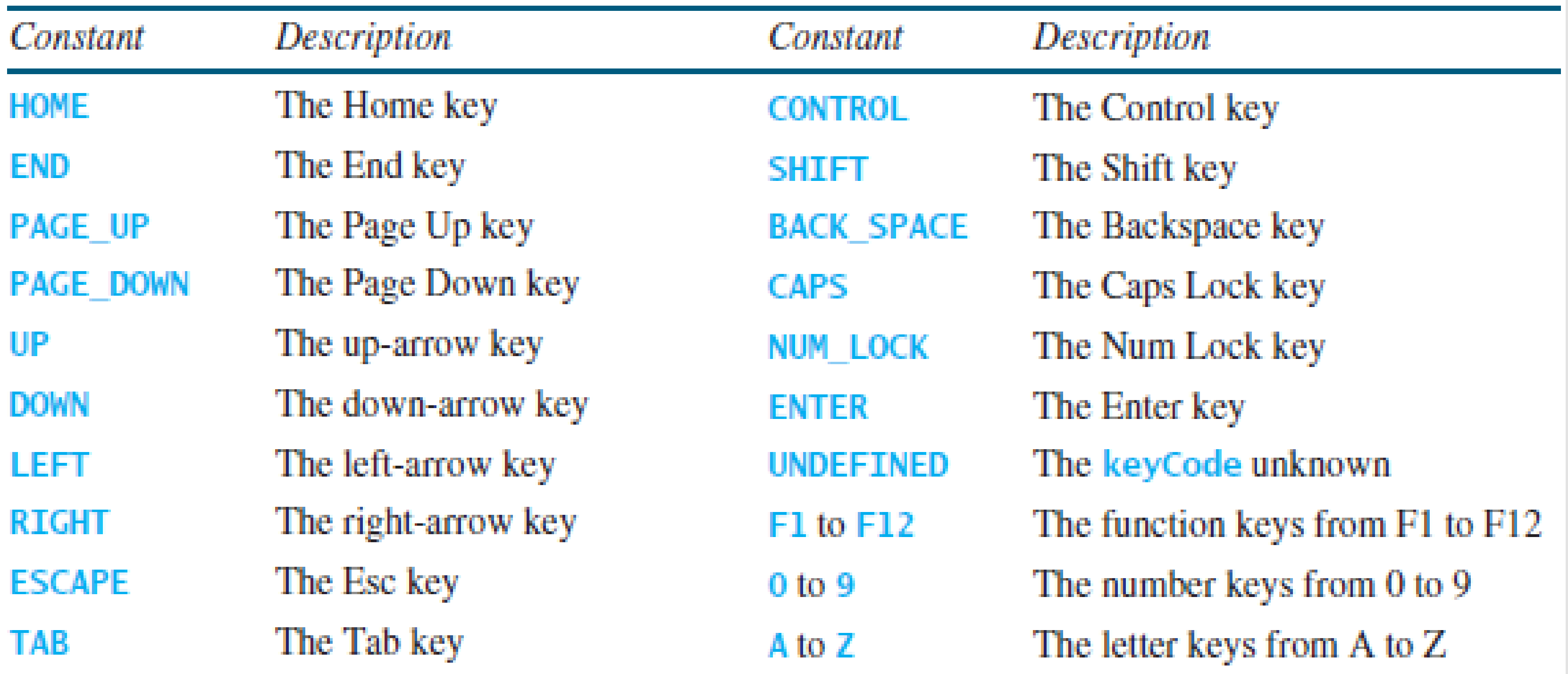
- Four constants - PRIMARY, SECONDARY, MIDDLE, and NONE - are defined in MouseButton to indicate the left, right, middle, and none mouse buttons.
- You can use the getButton() method to detect which button is pressed.

Key Events

- A KeyEvent is fired whenever a key is pressed, released, or typed on a node or a scene.
- **Key Event Methods**
- 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.

Key Events

- Every key event has an associated code that is returned by the getCode() method in KeyEvent.
- The key codes are constants defined in KeyCode class.



Constant	*Description*	*Constant*	*Description*
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

Key Events

- For the key-pressed and key-released events,
- getCode() returns the value as defined in the table,
- getText() returns a string that describes the key code, and
- getCharacter() returns an empty string.
- For the key-typed event,
- getCode() returns UNDEFINED and
- getCharacter() returns the Unicode character or a sequence of characters associated with the key-typed event.

Listeners for Observable Objects

- You can add a listener to process a value change in an observable object.
- Every binding property is an instance of Observable.
- An instance of Observable is known as an observable object, which contains the addListener(InvalidationListener listener) method for adding a listener.
- The listener class must implement the InvalidationListener interface to override the invalidated(Observable o) method for handling the value change.
- Once the value is changed in the Observable object, the listener is notified by invoking its invalidated(Observable o) method.

The Animation Class

- JavaFX provides the abstract Animation class with the core functionality for all animations.
- Animation Class: Properties
- autoReverse: BooleanProperty // Defines whether the animation reverses direction on alternating cycles.
- cycleCount: IntegerProperty // Defines the number of cycles in this animation.
- rate: DoubleProperty // Defines the speed for this animation.
- status: ReadOnlyObjectProperty <Animation.Status> // Read-only property to indicate the status of the animation.

The Animation Class

- Animation Class: Methods
- pause(): void // Pauses the animation.
- play(): void // Plays the animation from the current position.
- stop(): void // Stops the animation and resets the animation.
- +getter and setter methods

- Animation is the abstract class.
- Many concrete subclasses of Animation also provided in JavaFX.
- Like: PathTransition, FadeTransition etc.

The Duration class

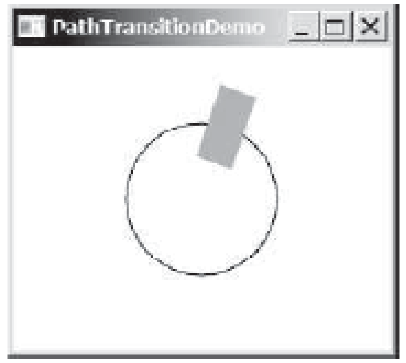
- The Duration class defines a duration of time.
- It is an immutable class.
- The class defines constants INDEFINITE, ONE, UNKNOWN, and ZERO to represent an indefinte duration, 1 milli-second, unknown, and 0 duration.
- You can use new Duration(double millis) to create an instance of Duration.
- There are add, subtract, multiply, and divide methods to perform arithmetic operations.
- There are toHours(), toMinutes(), toSeconds(), and toMillis() to return the number of hours, minutes, seconds, and milliseconds in the duration.
- You can also use compareTo to compare two durations.

Animation: PathTransition

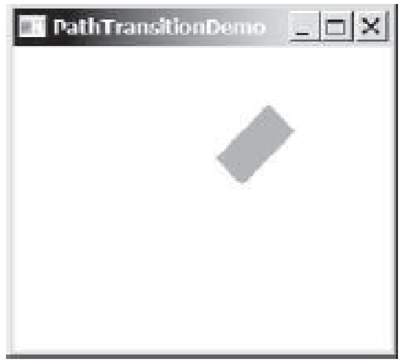
- The PathTransition class animates the the moves of a node along a path from one end to the other over a given time. **PathTransition is a subtype of Animation.**
- PathTransition Class: Properties
- duration: ObjectProperty<Duration> // The duration of this transition.
- node: ObjectProperty<Node> // The target node of this transition.
- orientation: ObjectProperty<PathTransition.OrientationType> // The orientation of the node along the path.
- path: ObjectType<Shape> // The shape whose outline is used as a path to animate the node move.

Animation: PathTransition

- PathTransition Class: Methods
- PathTransition() // Creates an empty PathTransition.
- PathTransition(duration: Duration,path: Shape) // Creates a PathTransition with the specified duration and path.
- PathTransition(duration: Duration,path: Shape, node: Node)
- // Creates a PathTransition with the specified duration, path, and node.
- +getter and setter methods



(a)



(b)

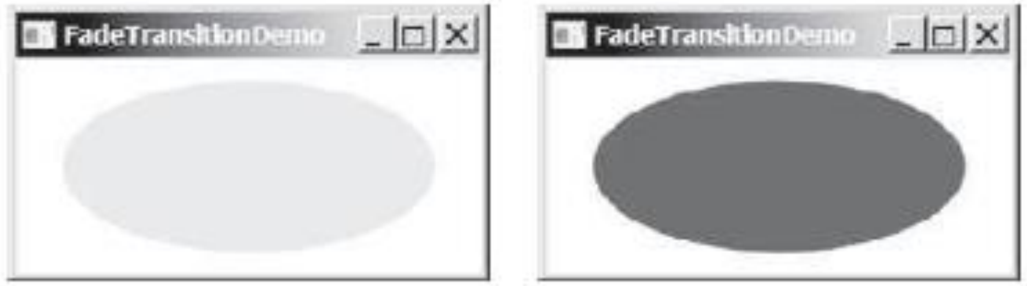
The PathTransition animates a rectangle moving along the circle.

Animation: FadeTransition

- **The FadeTransition class animates the change of the opacity in a node over a given time. FadeTransition is a subtype of Animation.**
- **FadeTransition Class: Properties**
- duration: ObjectProperty<Duration> // The duration of this transition.
- node: ObjectProperty<Node> // The target node of this transition.
- fromValue: DoubleProperty // The start opacity for this animation.
- toValue: DoubleProperty // The stop opacity for this animation.
- byValue: DoubleProperty // The incremental value on the opacity for this animation.

Animation: FadeTransition

- **FadeTransition Class: Methods**
- FadeTransition() // Creates an empty FadeTransition.
- FadeTransition(duration: Duration) // Creates a FadeTransition with the specified duration.
- FadeTransition(duration: Duration,node: Node) // Creates a FadeTransition with the specified duration and node.
- +getter and setter methods



The FadeTransition animates the change of opacity in the ellipse.

Summary

- Basic structure of JAVAFX program,
- Panes,
- UI control and shapes,
- Property binding,
- the Color and the Font class,
- the Image and Image-View class,
- layout panes and shapes,
- Events and Events sources,
- Registering Handlers and Handling Events,
- Inner classes,
- anonymous inner class handlers,
- mouse and key events,
- listeners for observable objects,
- animation

Reference Programs

Property Binding #Program 1

```
import javafx.application.Application; import javafx.scene.Scene; import javafx.scene.layout.Pane; import javafx.scene.paint.Color;

import javafx.scene.shape.Circle; import javafx.stage.Stage;

public class BindingCircle extends Application {

public void start(Stage primaryStage) {

Pane pane = new Pane();

Circle circle = new Circle();

circle.centerXProperty().bind(pane.widthProperty().divide(2));

circle.centerYProperty().bind(pane.heightProperty().divide(2));

circle.setRadius(50);

circle.setStroke(Color.BLACK);

circle.setFill(Color.WHITE);

pane.getChildren().add(circle); // Add circle to the pane

Scene scene = new Scene(pane, 200, 200);

primaryStage.setTitle("ShowCircleCentered"); // Set the stage title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

}

 public static void main(String[] args) {

      Application.launch(args);   }   }
```

END OF UNIT - 7