Event-Driven Programming and Animations

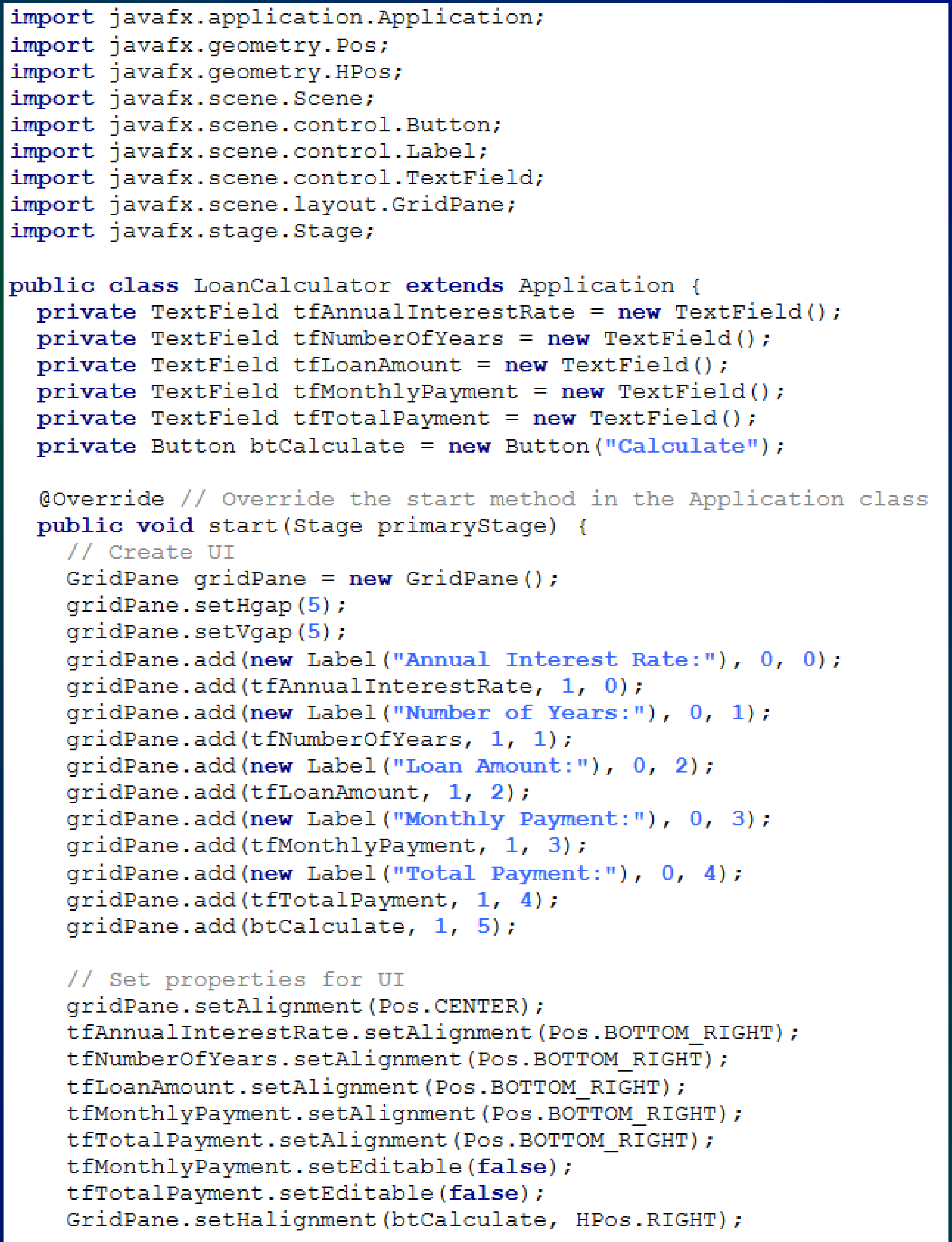
Chapter 15

Motivations

Suppose you want to write a GUI program that lets the user enter a loan amount, annual interest rate, and number of years and click the *Compute Payment* button to obtain the monthly payment and total payment.

How do you accomplish the task? You have to use *event-driven programming* to write the code to respond to the button-clicking event.

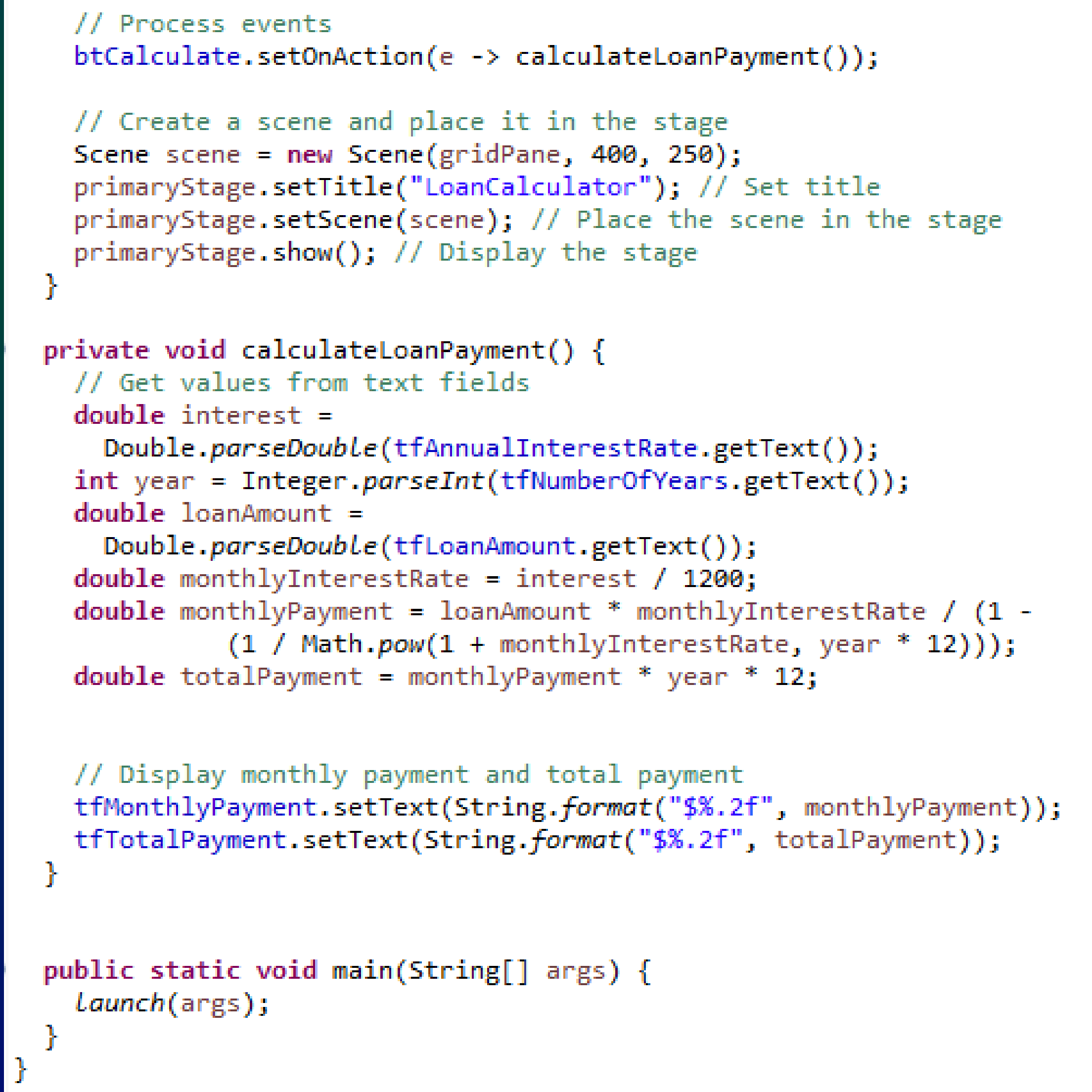
Loan Calculator



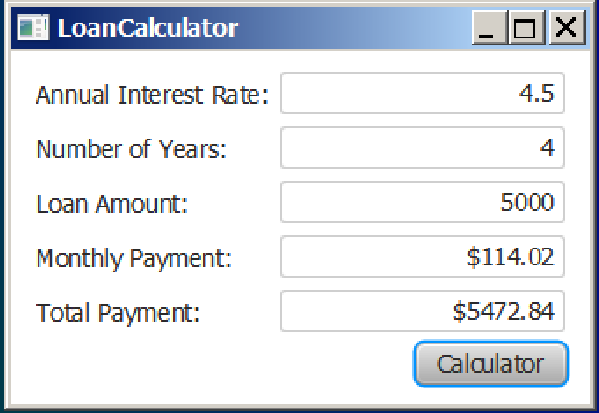
```java
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

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 // Override the start method in the Application class
  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);

    // Set properties for UI
    gridPane.setAlignment(Pos.CENTER);
    tfAnnualInterestRate.setAlignment(Pos.BOTTOM_RIGHT);
    tfNumberOfYears.setAlignment(Pos.BOTTOM_RIGHT);
    tfLoanAmount.setAlignment(Pos.BOTTOM_RIGHT);
    tfMonthlyPayment.setAlignment(Pos.BOTTOM_RIGHT);
    tfTotalPayment.setAlignment(Pos.BOTTOM_RIGHT);
    tfMonthlyPayment.setEditable(false);
    tfTotalPayment.setEditable(false);
    GridPane.setHalignment(btCalculate, HPos.RIGHT);
```

Part 1

Loan Calculator



```
    // Process events
    btCalculate.setOnAction(e -> calculateLoanPayment());

    // Create a scene and place it in the stage
    Scene scene = new Scene(gridPane, 400, 250);
    primaryStage.setTitle("LoanCalculator"); // Set title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  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());
    double monthlyInterestRate = interest / 1200;
    double monthlyPayment = loanAmount * monthlyInterestRate / (1 -
              (1 / Math.pow(1 + monthlyInterestRate, year * 12)));
    double totalPayment = monthlyPayment * year * 12;


    // Display monthly payment and total payment
    tfMonthlyPayment.setText(String.format("$%.2f", monthlyPayment));
    tfTotalPayment.setText(String.format("$%.2f", totalPayment));
  }


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



Annual Interest Rate:	4.5
Number of Years:	4
Loan Amount:	5000
Monthly Payment:	$114.02
Total Payment:	$5472.84

Part 2

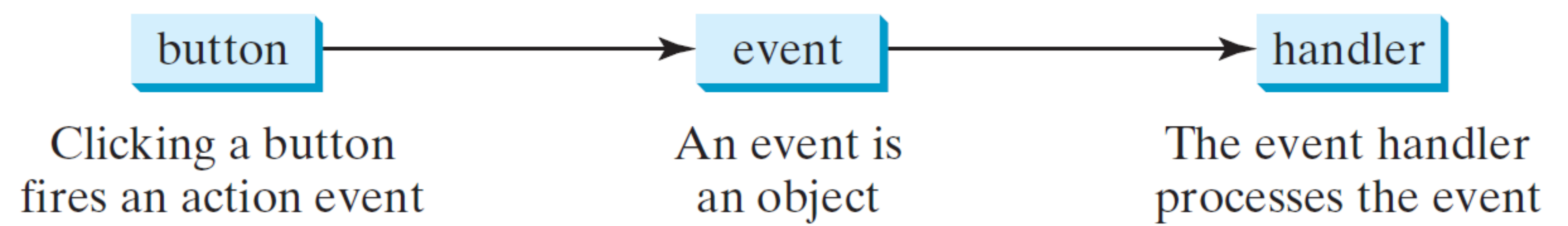
Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order.
- In *event-driven programming*, code is executed upon activation of events.

Handling GUI Events

Source object (e.g., button)

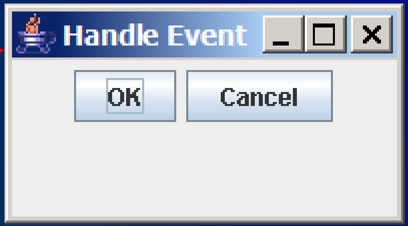
Listener object contains a method for processing the event.


button
event
handler
Clicking a button fires an action event
An event is an object
The event handler processes the event

Trace Execution

```
public class HandleEvent extends Application {
  public void start(Stage primaryStage) {
    …
    OKHandlerClass handler1 = new OKHandlerClass();
    btOK.setOnAction(handler1);
    CancelHandlerClass handler2 = new CancelHandlerClass();
    btCancel.setOnAction(handler2);
    …
    primaryStage.show(); // Display the stage
  }
}

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

1. Start from the main method to create a window and display it

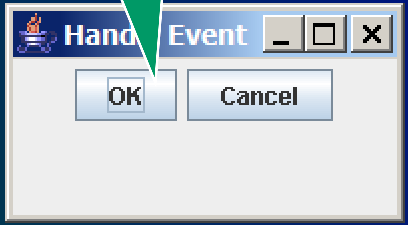


animation

Trace Execution

```
public class HandleEvent extends Application {
  public void start(Stage primaryStage) {
    …
    OKHandlerClass handler1 = new OKHandlerClass();
    btOK.setOnAction(handler1);
    CancelHandlerClass handler2 = new CancelHandlerClass();
    btCancel.setOnAction(handler2);
    …
    primaryStage.show(); // Display the stage
  }
}

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

2. Click OK

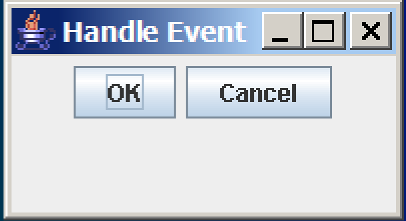


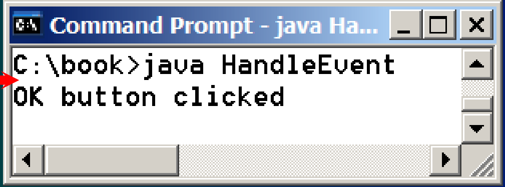
animation

Trace Execution

```
public class HandleEvent extends Application {
  public void start(Stage primaryStage) {
    …
    OKHandlerClass handler1 = new OKHandlerClass();
    btOK.setOnAction(handler1);
    CancelHandlerClass handler2 = new CancelHandlerClass(
    btCancel.setOnAction(handler2);
    …
    primaryStage.show(); // Display the stage
  }
}

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

3. Click OK. The JVM invokes the listener’s handle method





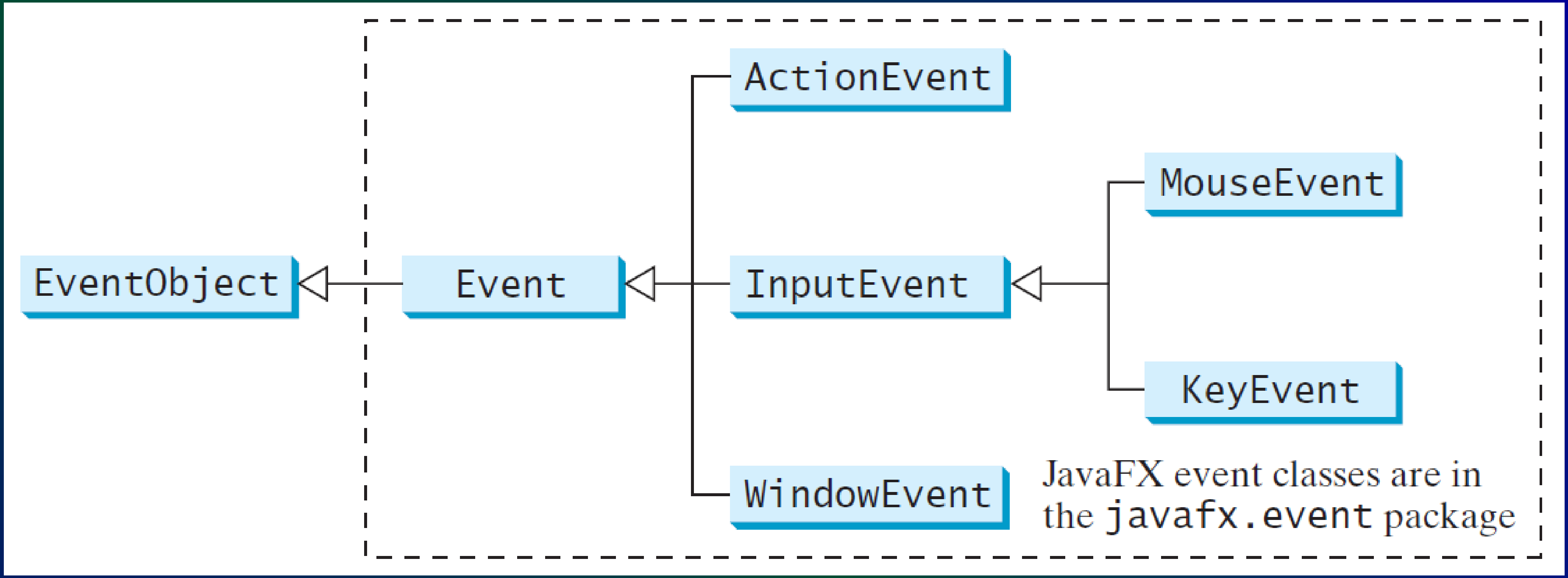
```
C:\book>java HandleEvent
OK button clicked
```

Events

- An *event* can be defined as a type of signal to the program that something has happened.

- The event is generated by external user actions such as mouse movements, mouse clicks, or keystrokes.

Event Classes

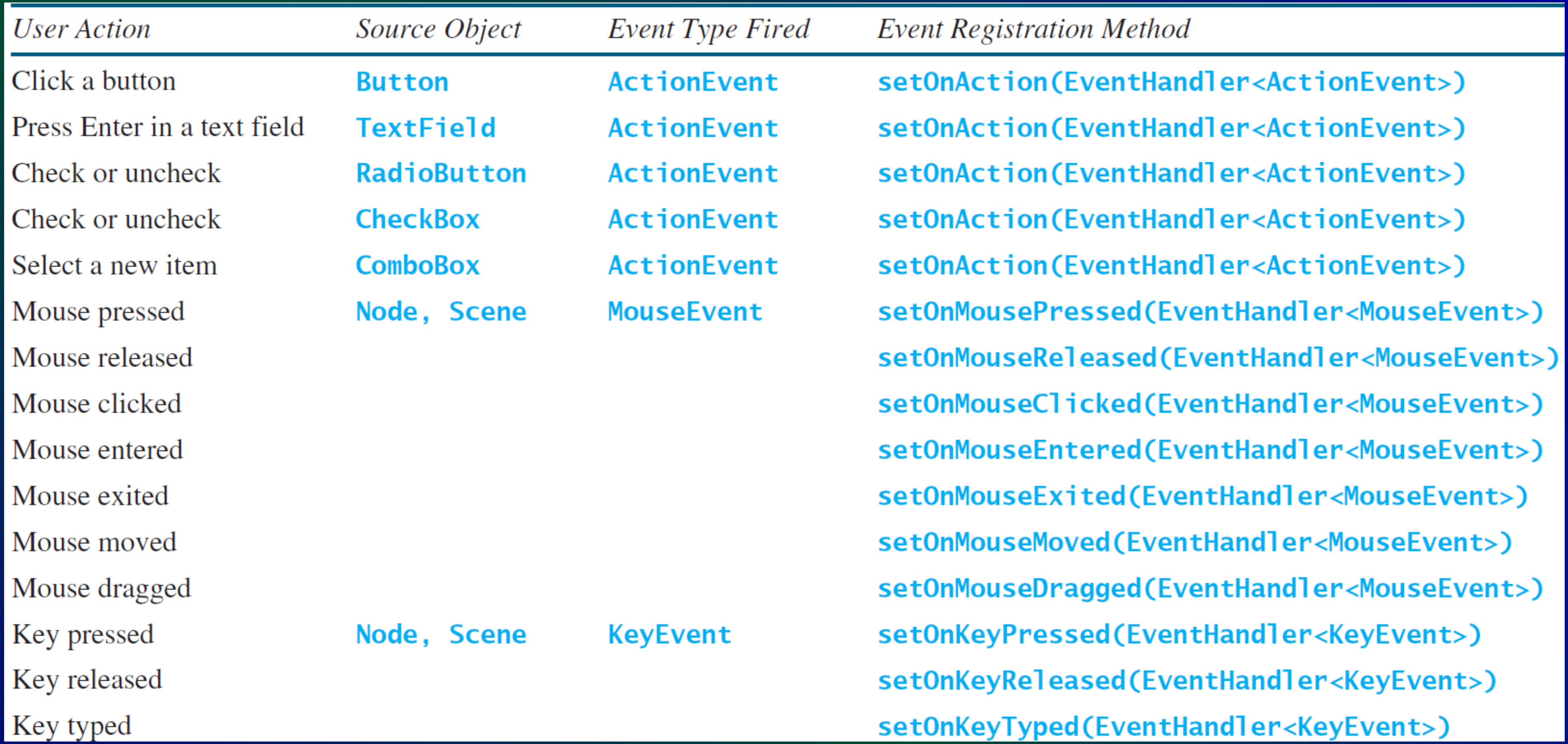


JavaFX event classes are in the `javafx.event` package

Event Information

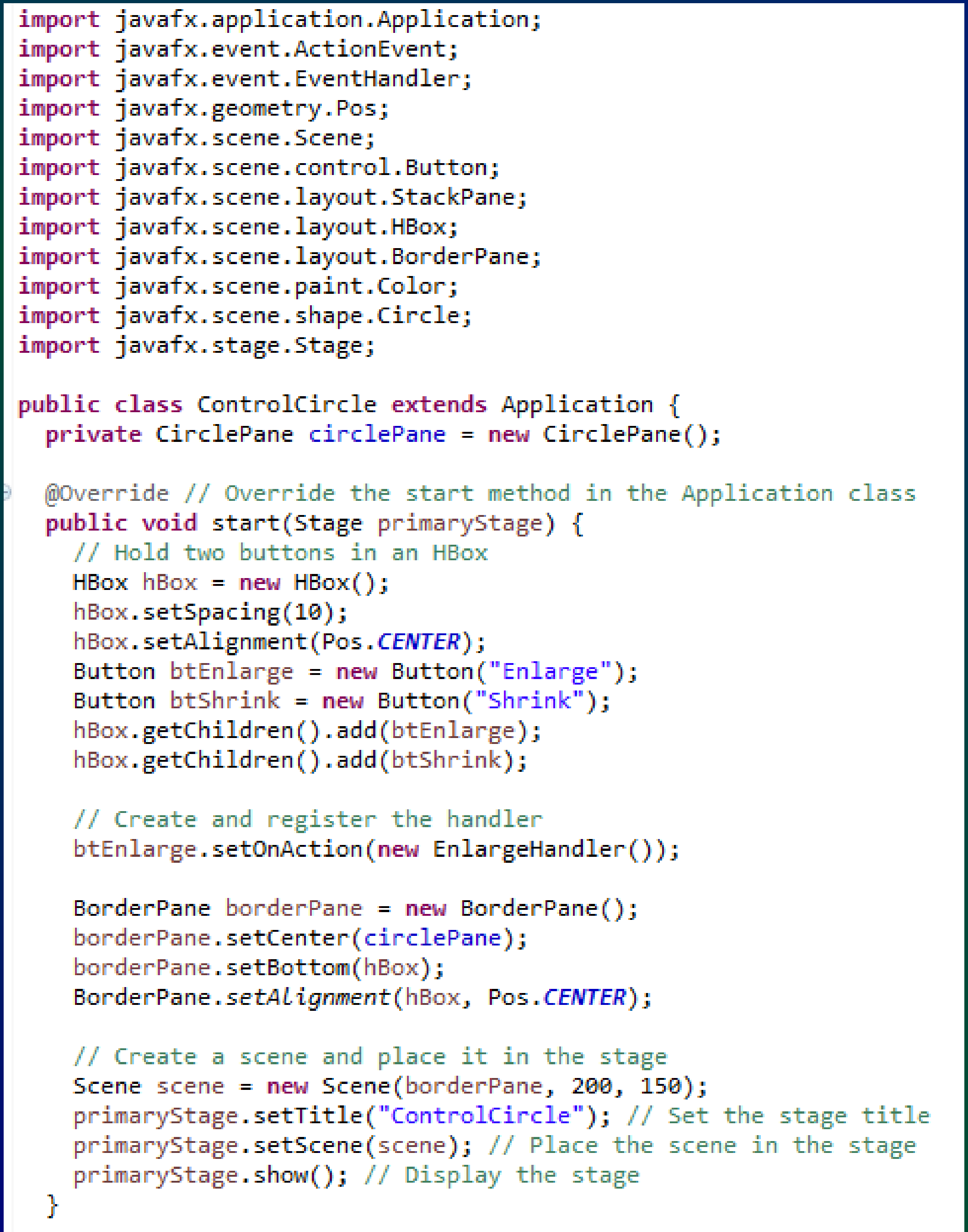
- An event object contains whatever properties are pertinent to the event.
- You can identify the source object of the event using the getSource() instance method in the EventObject class.
- The subclasses of EventObject deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes.

Selected User Actions and Handlers



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>)`

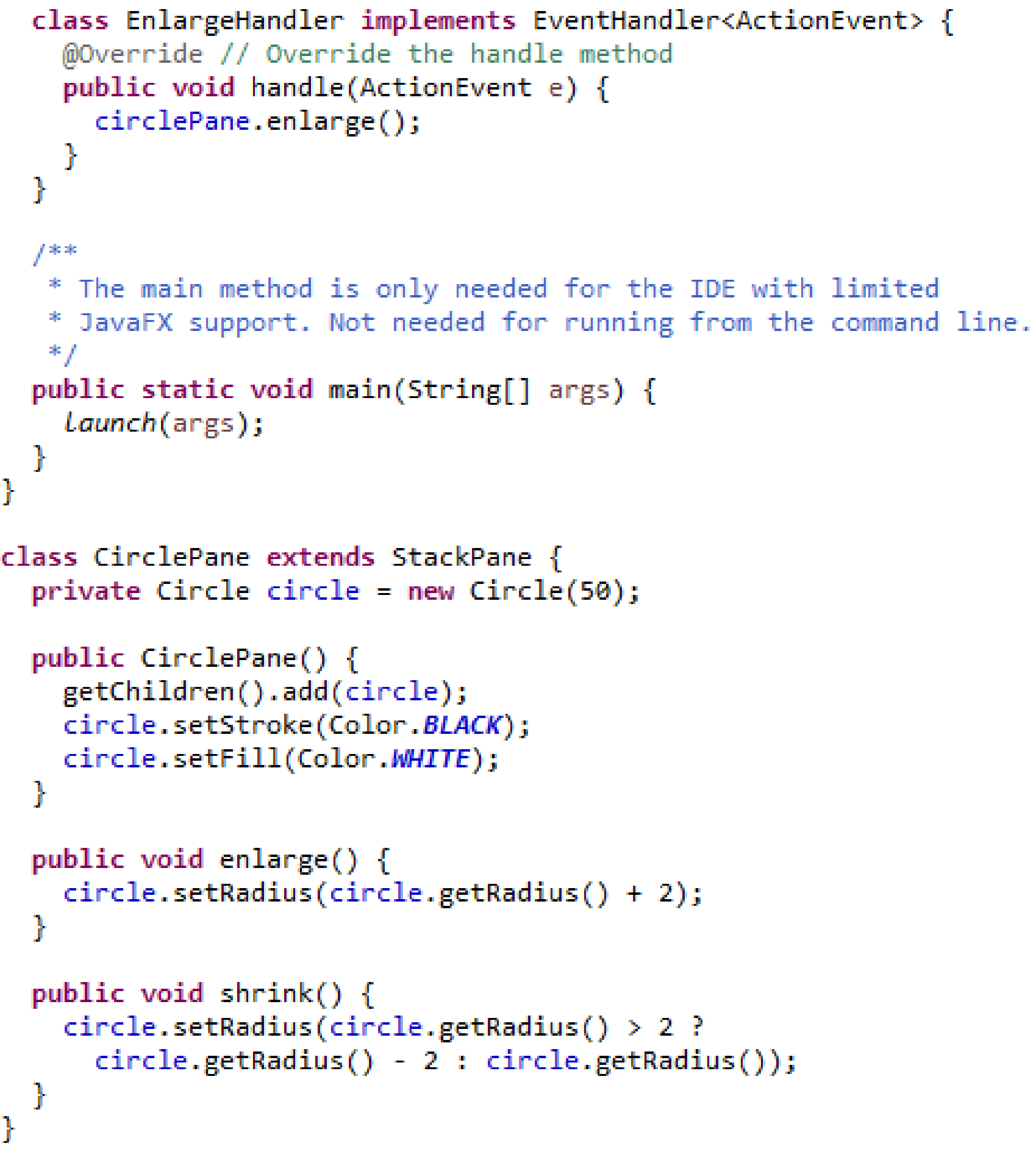
Example ConctrolCircle (with listener for Enlarge)



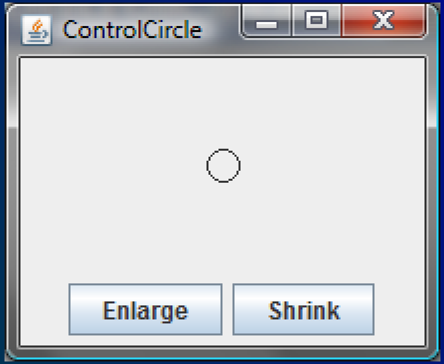
```java
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.BorderPane;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

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

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Hold two buttons in an HBox
    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);

    // Create and register the handler
    btEnlarge.setOnAction(new EnlargeHandler());

    BorderPane borderPane = new BorderPane();
    borderPane.setCenter(circlePane);
    borderPane.setBottom(hBox);
    BorderPane.setAlignment(hBox, Pos.CENTER);

    // Create a scene and place it in the stage
    Scene scene = new Scene(borderPane, 200, 150);
    primaryStage.setTitle("ControlCircle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
```

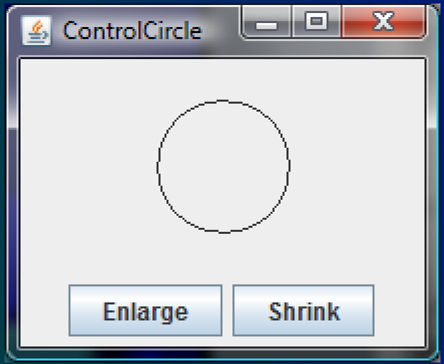
Part 1

Example ConctrolCircle (with listener for Enlarge)



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

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}

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());
  }
}
```





Part 2

Inner Class Listeners

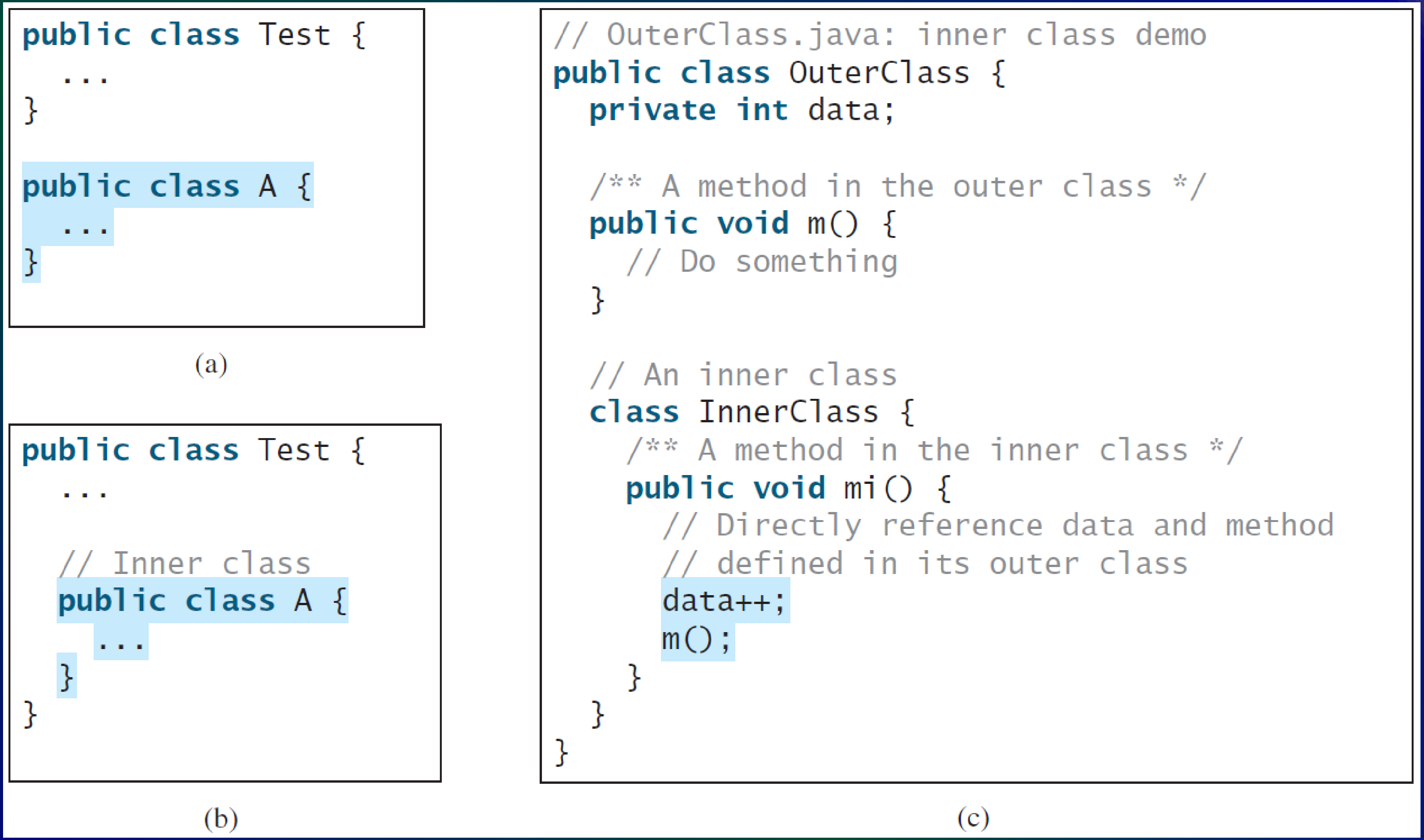
- A listener class is designed specifically to create a listener object for a GUI component (e.g., a button).
- It will not be shared by other applications.
- So, it is appropriate to define the listener class inside the frame class as an inner class.

Inner Classes

Inner class: 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.

Inner Classes, cont.



```
public class Test {
  ...
}

public class A {
  ...
}
```

(a)

```
public class Test {
  ...

  // Inner class
  public class A {
    ...
  }
}
```

(b)

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

(c)

Inner Classes (cont.)

- Inner classes can make programs simple and concise.
- An inner class supports the work of its containing outer class and is compiled into a class named *OuterClassName$InnerClassName*.class.
- For example, the inner class InnerClass in OuterClass is compiled into *OuterClass$InnerClass*.class .

Inner Classes (cont.)

- 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. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class

Anonymous Inner Classes

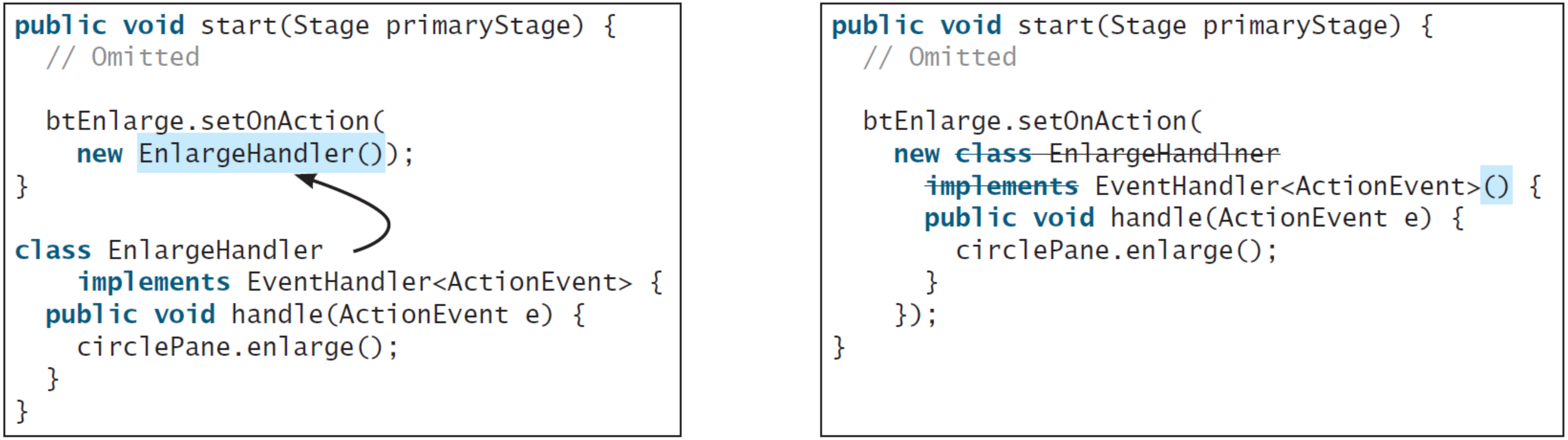
- An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- An anonymous inner class always uses the no-arg 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. For example, if the outer class Test has two anonymous inner classes, these two classes are compiled into Test$1.class and Test$2.class.

Anonymous Inner Classes (cont.)

- Inner class listeners can be shortened using anonymous inner classes.
- An *anonymous inner class* is an inner class 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 or interface
  // Other methods if necessary
}
```

Anonymous Inner Classes (cont.)



```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new EnlargeHandler());
}

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

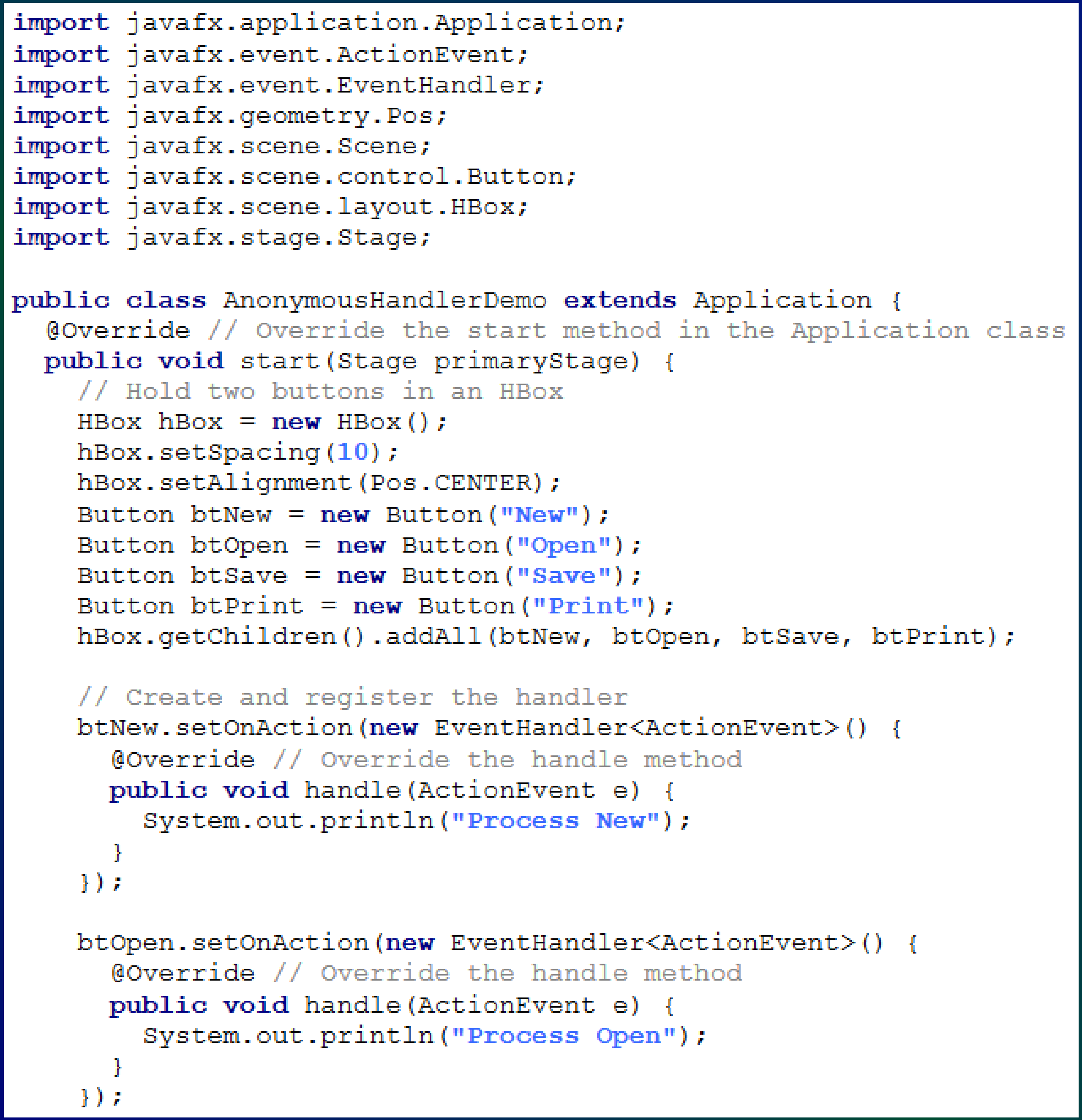
(a) Inner class EnlargeListener

```
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

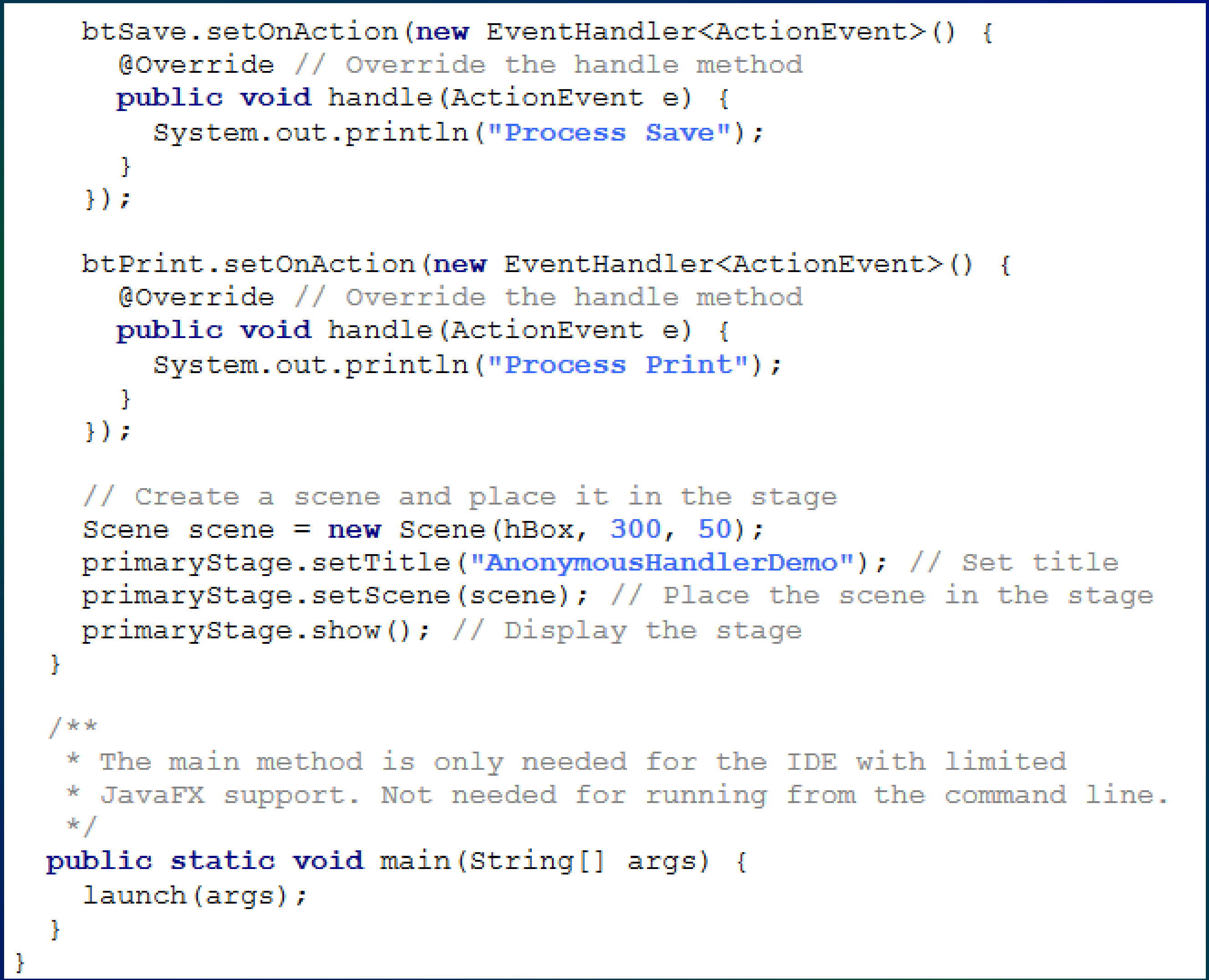
Example

Anonymous Handler



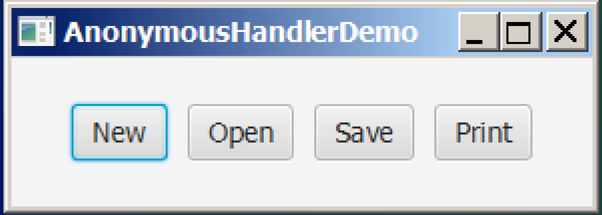
```java
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class AnonymousHandlerDemo extends Application {
  @Override // Override the start method in the Application class
  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);

    // 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");
      }
    });
```

Part 1



```
    btSave.setOnAction(new EventHandler<ActionEvent>() {
      @Override // Override the handle method
      public void handle(ActionEvent e) {
        System.out.println("Process Save");
      }
    });

    btPrint.setOnAction(new EventHandler<ActionEvent>() {
      @Override // Override the handle method
      public void handle(ActionEvent e) {
        System.out.println("Process Print");
      }
    });

    // Create a scene and place it in the stage
    Scene scene = new Scene(hBox, 300, 50);
    primaryStage.setTitle("AnonymousHandlerDemo"); // Set title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

Example Anonymous Handler



Part 2

Simplifying Event Handing Using Lambda Expressions

Lambda expression is a new feature in Java 8. Lambda expressions can be viewed as an anonymous method with a concise syntax. For example, the following code in (a) can be greatly simplified using a lambda expression in (b) in three lines.

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

(a) Anonymous inner class event handler

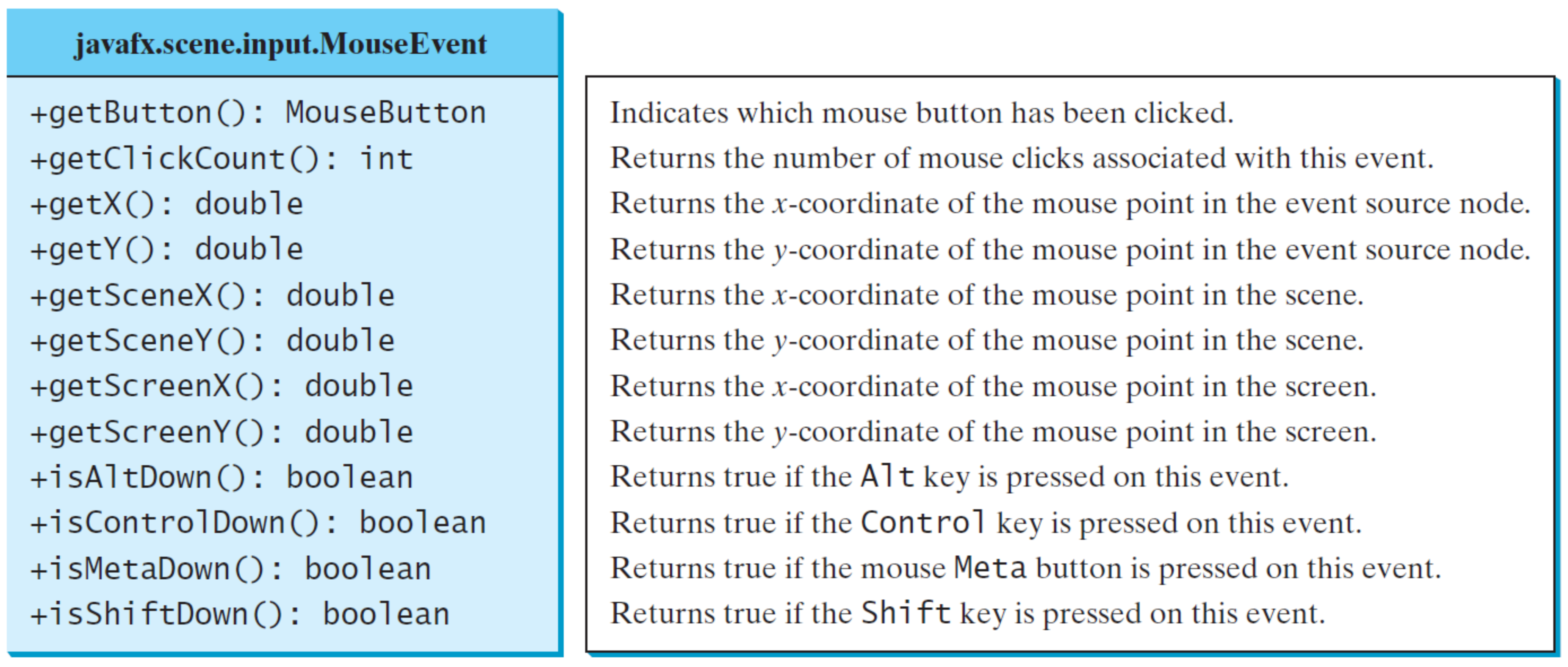
```java
btEnlarge.setOnAction(e -> {
  // Code for processing event e
});
```

(b) Lambda expression event handler

Single Abstract Method Interface (SAM)

- For the compiler to understand lambda expressions, the interface must contain exactly one abstract method.
- 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.
- Such an interface is known as a *functional interface*, or a *Single Abstract Method* (SAM) interface.

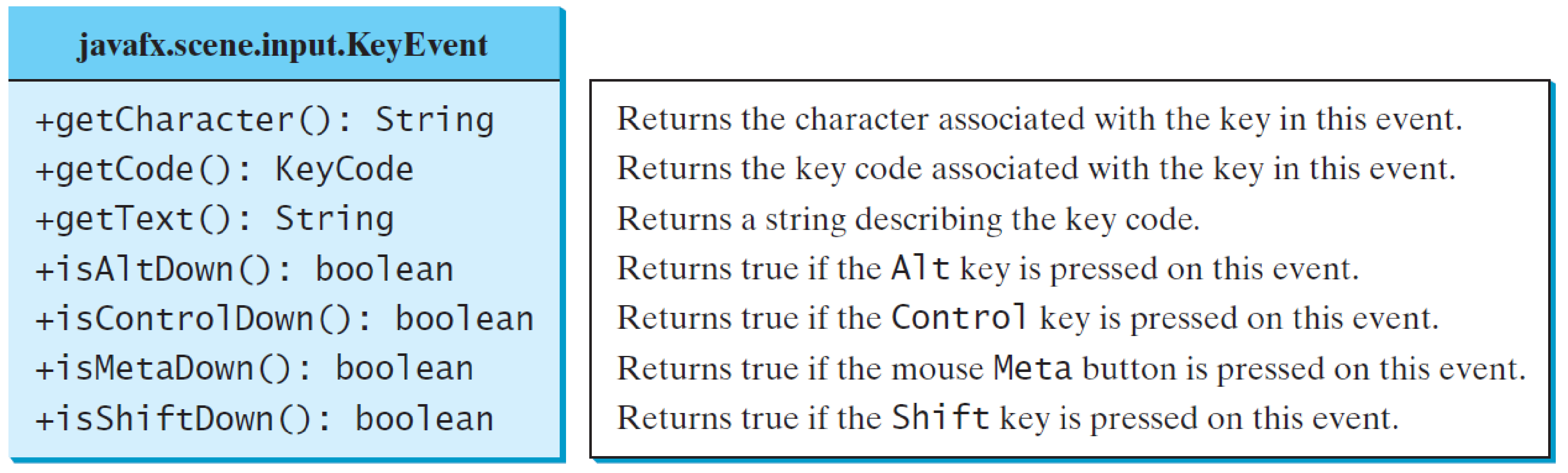
MouseEvent



javafx.scene.input.MouseEvent

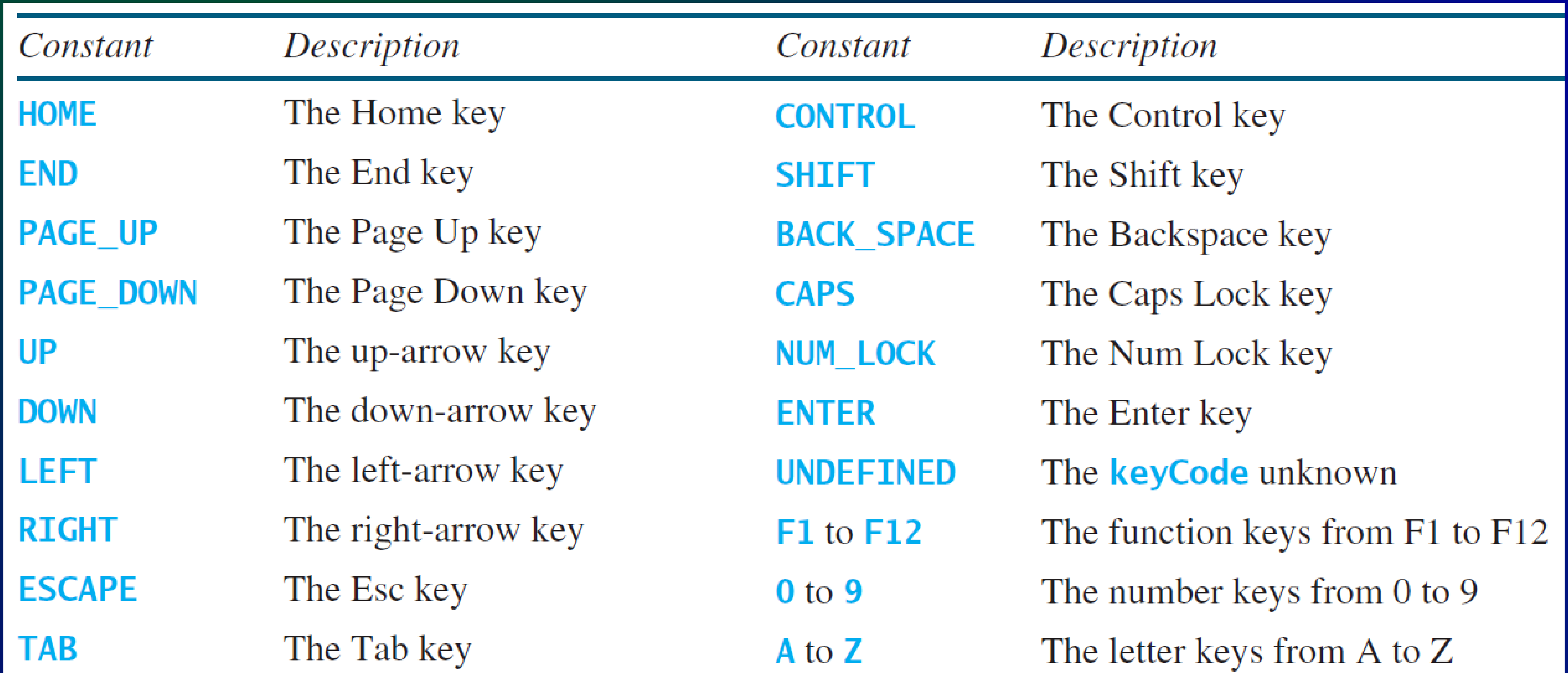
`+getButton(): MouseButton`	Indicates which mouse button has been clicked.
`+getClickCount(): int`	Returns the number of mouse clicks associated with this event.
`+getX(): double`	Returns the *x*-coordinate of the mouse point in the event source node.
`+getY(): double`	Returns the *y*-coordinate of the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

The KeyEvent 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.

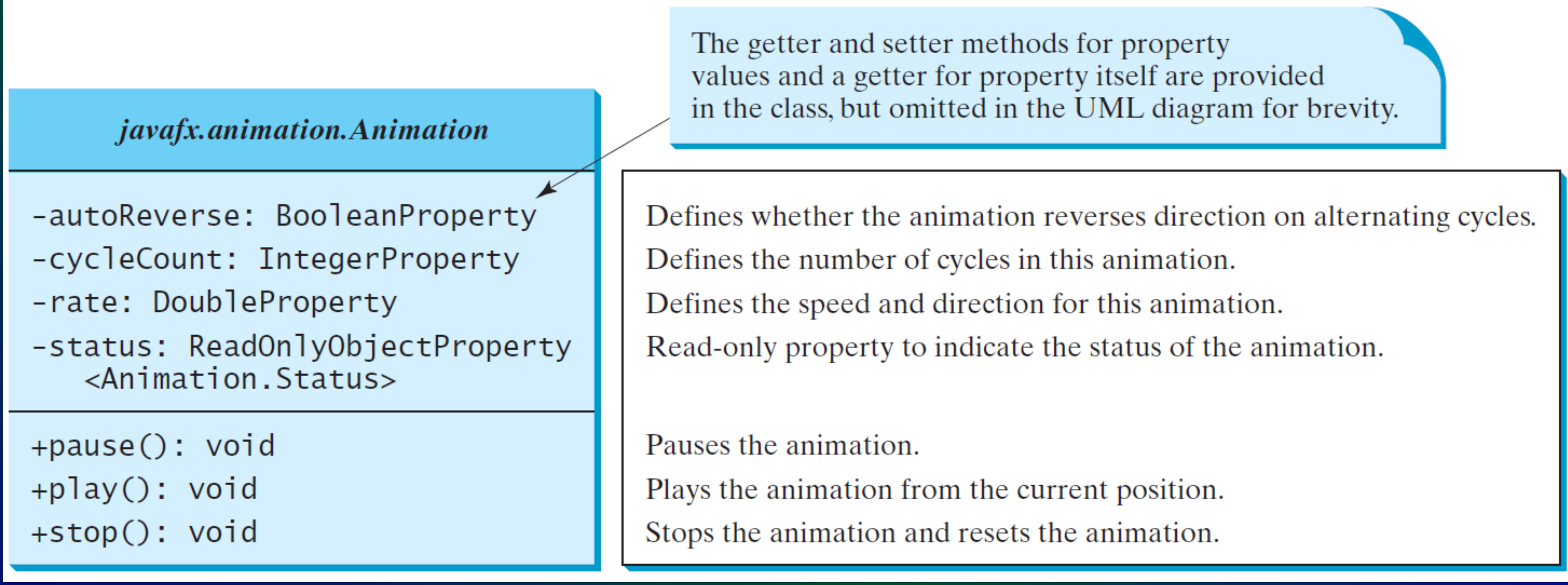
The KeyCode Constants



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

Animation

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



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.

javafx.animation.Animation

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

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

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.

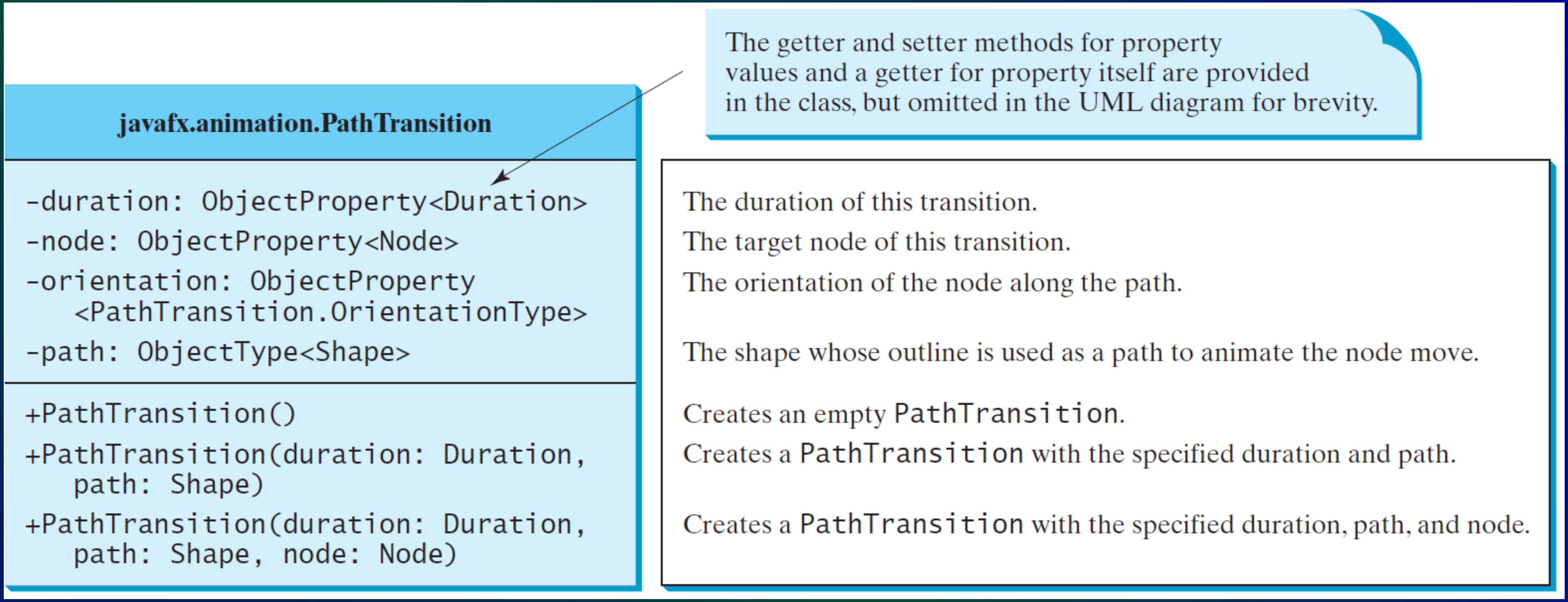
Pauses the animation.

Plays the animation from the current position.

Stops the animation and resets the animation.

PathTransition

Transitions in JavaFX provide the means to incorporate animations in an internal timeline. Transitions can be composed to create multiple animations that are executed in parallel or sequentially.



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.

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 duration of this transition.

The target node of this transition.

The orientation of the node along the path.

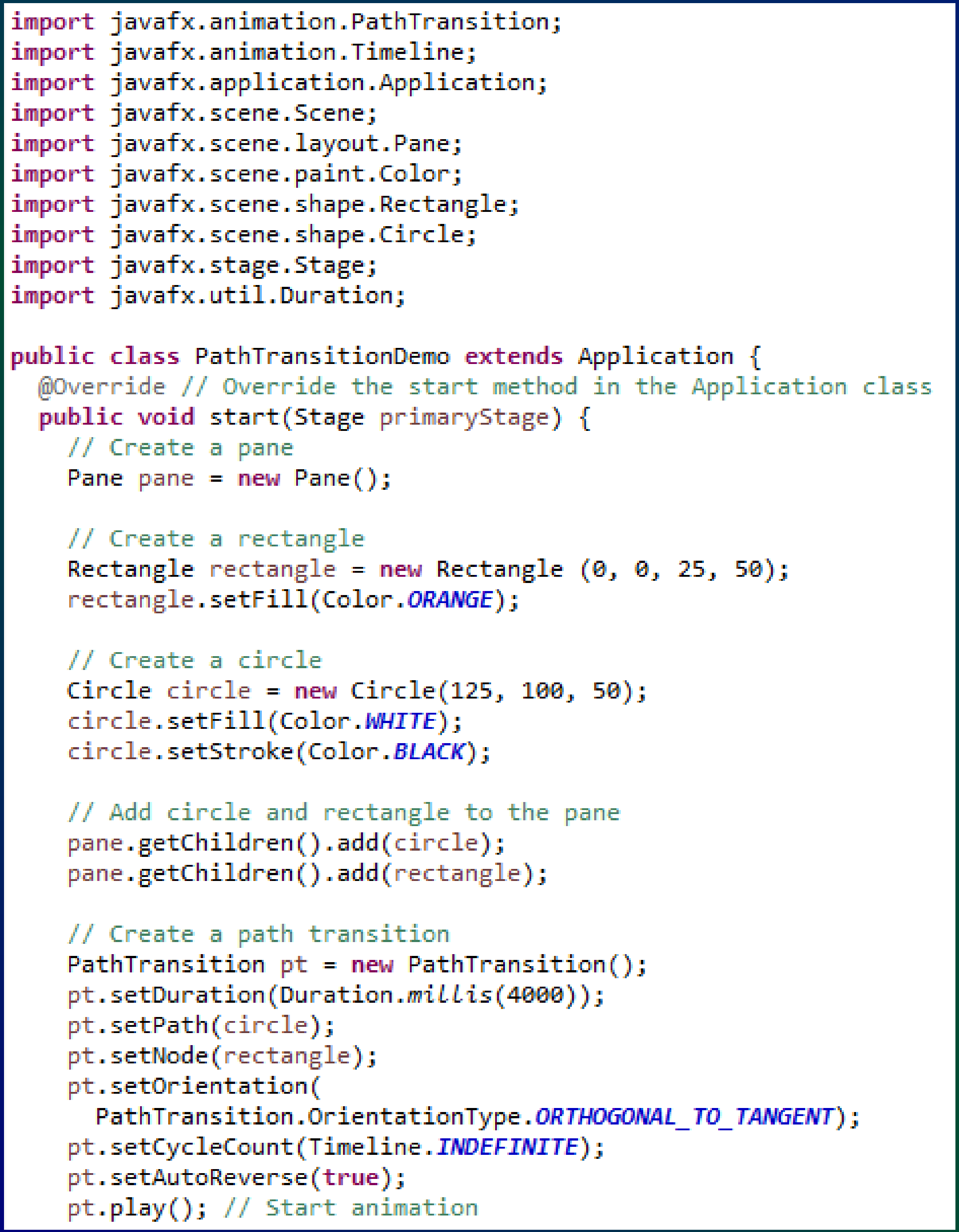
The shape whose outline is used as a path to animate the node move.

Creates an empty `PathTransition`.

Creates a `PathTransition` with the specified duration and path.

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

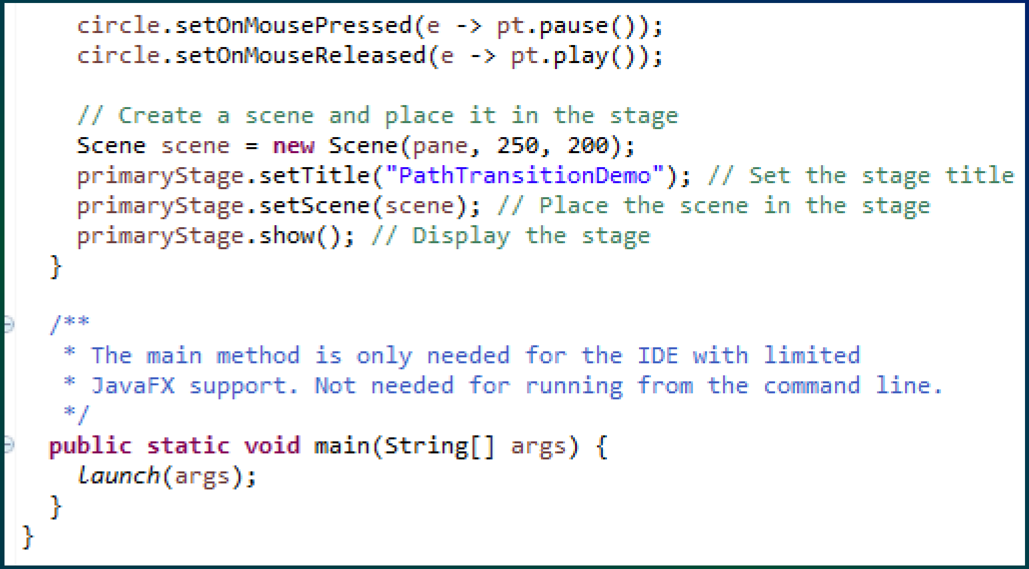
Example PathTransition



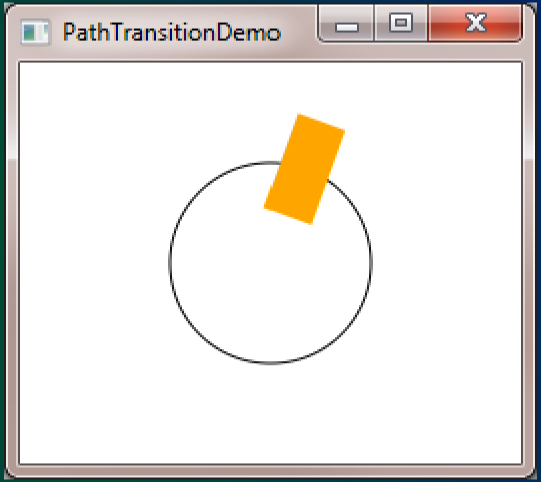
```
import javafx.animation.PathTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class PathTransitionDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    // Create a rectangle
    Rectangle rectangle = new Rectangle (0, 0, 25, 50);
    rectangle.setFill(Color.ORANGE);

    // Create a circle
    Circle circle = new Circle(125, 100, 50);
    circle.setFill(Color.WHITE);
    circle.setStroke(Color.BLACK);

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

Part 1

Example PathTransition



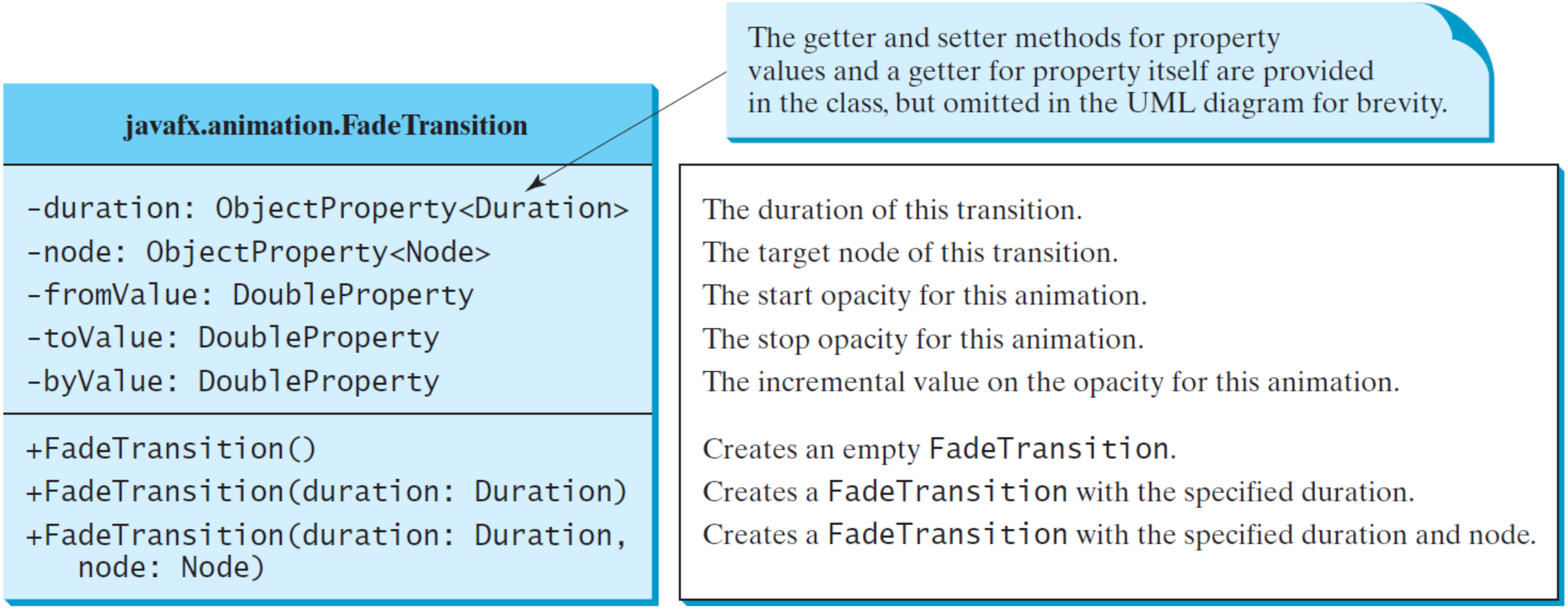
```
    circle.setOnMousePressed(e -> pt.pause());
    circle.setOnMouseReleased(e -> pt.play());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 200);
    primaryStage.setTitle("PathTransitionDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```



Part 2

FadeTransition

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



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.

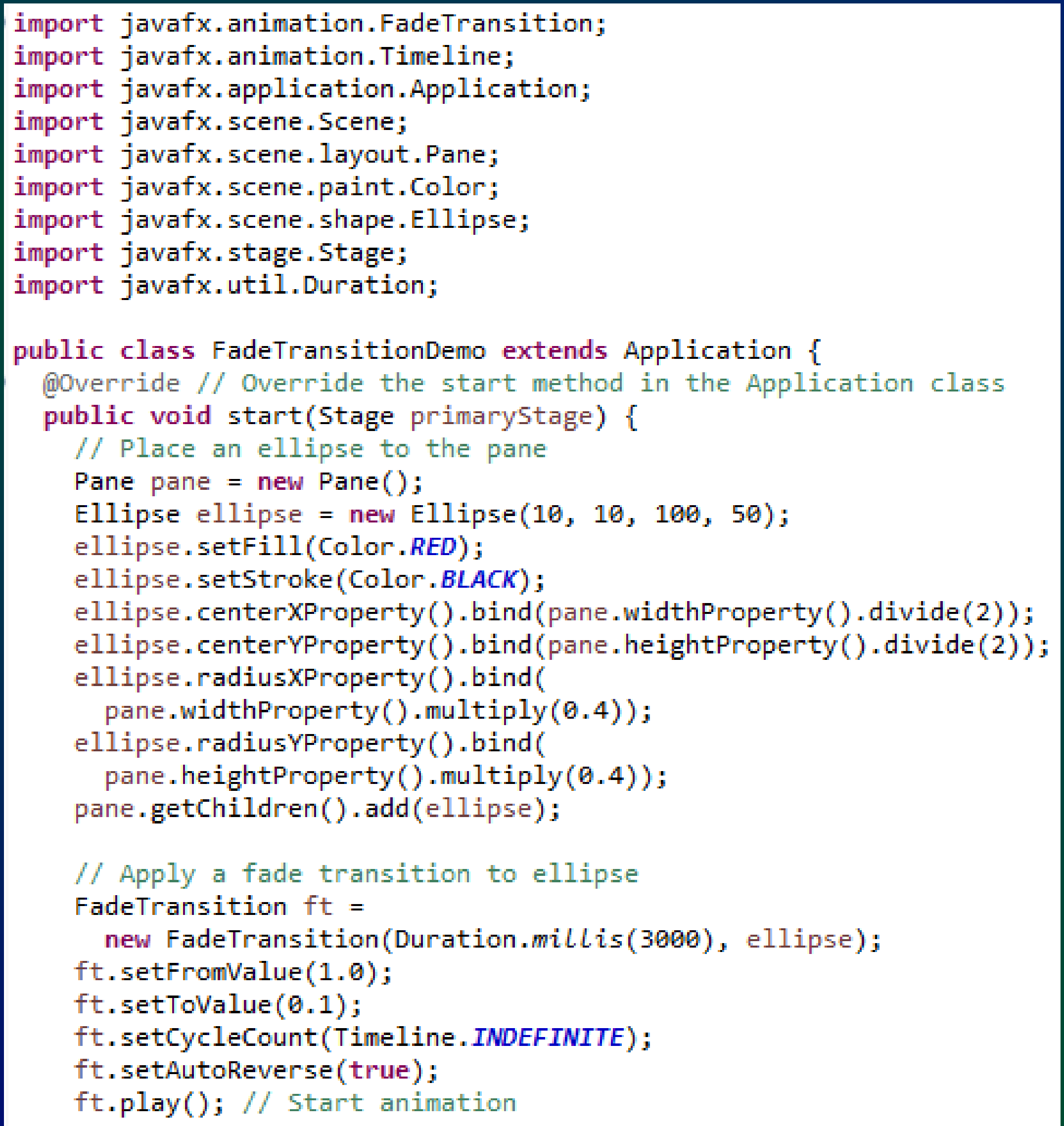
javafx.animation.FadeTransition

```
-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.

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

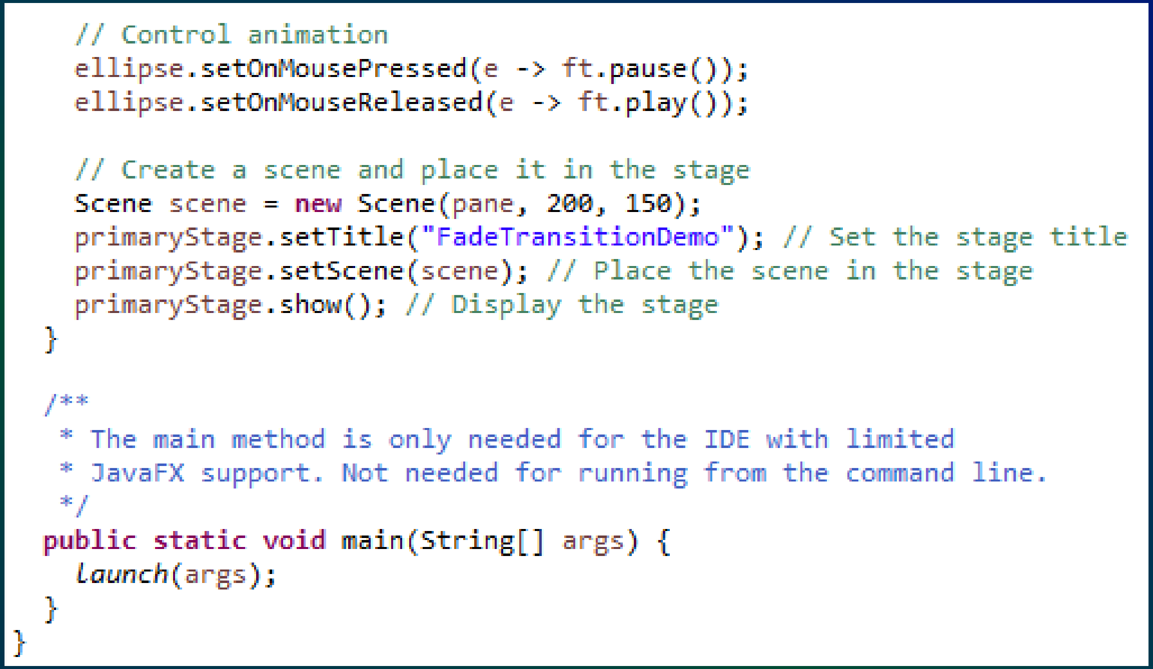
Example FadeTransition



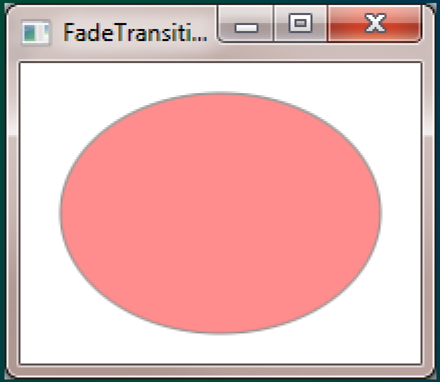
```
import javafx.animation.FadeTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Ellipse;
import javafx.stage.Stage;
import javafx.util.Duration;

public class FadeTransitionDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Place an ellipse to the pane
    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
```

Part 1

Example FadeTransition



```
    // Control animation
    ellipse.setOnMousePressed(e -> ft.pause());
    ellipse.setOnMouseReleased(e -> ft.play());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 150);
    primaryStage.setTitle("FadeTransitionDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

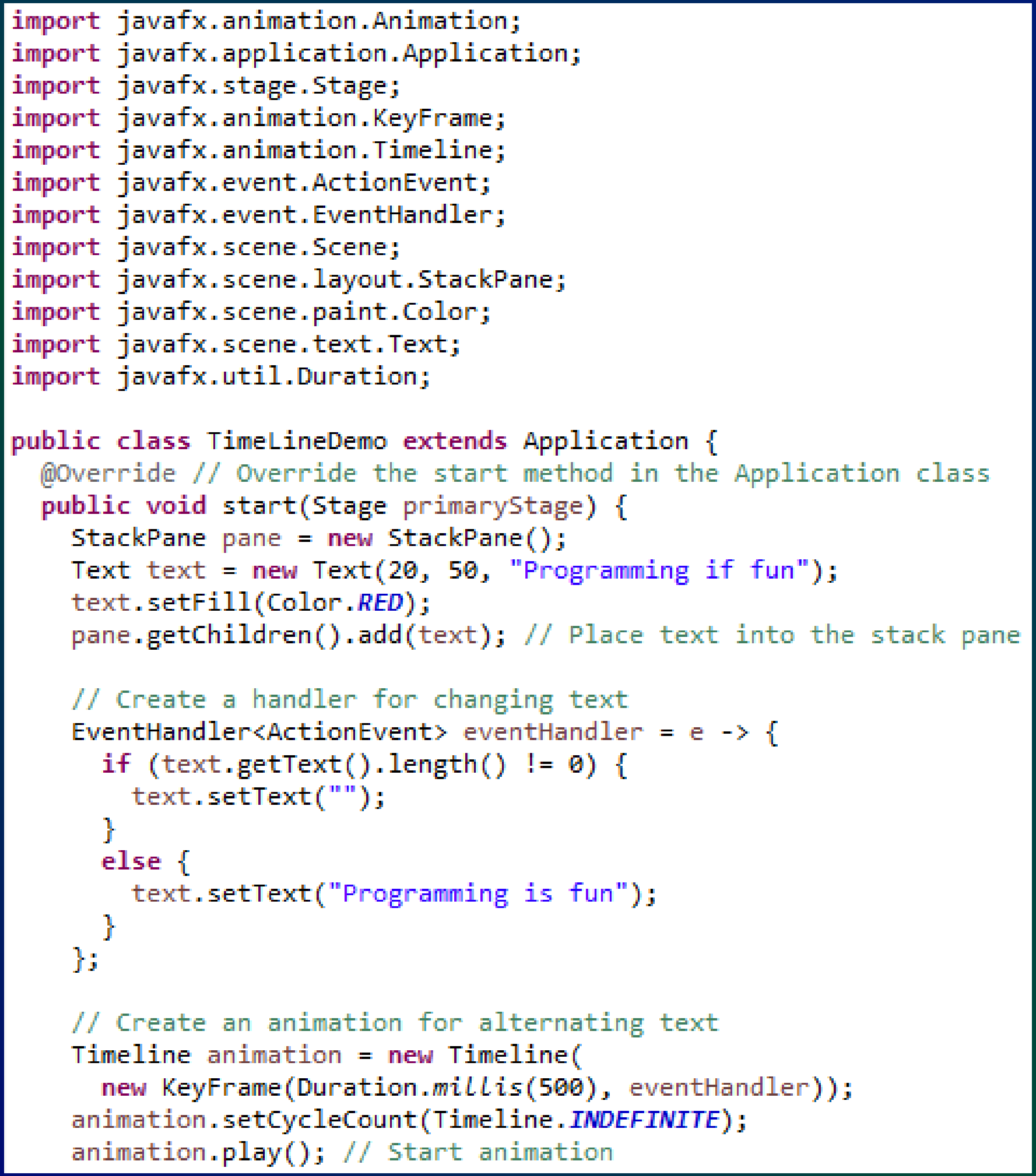


Part 2

Timeline

- **PathTransition** and **FadeTransition** define specialized animations.
- The **Timeline** class can be used to program any animation using one or more **KeyFrames**.
- Each **KeyFrame** is executed sequentially at a specified time interval.
- **Timeline** inherits from **Animation**.

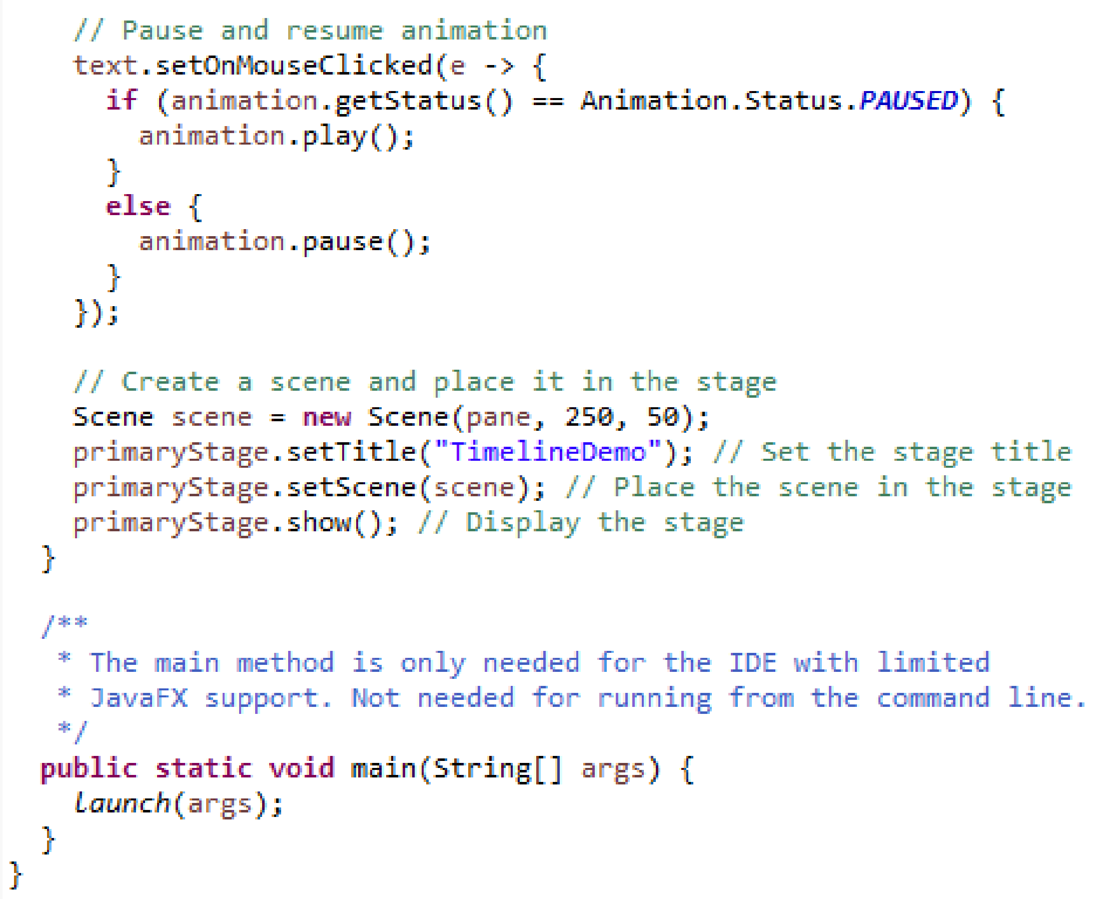
Example Timeline



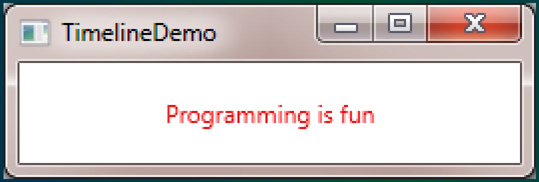
```java
import javafx.animation.Animation;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.util.Duration;

public class TimeLineDemo extends Application {
  @Override // Override the start method in the Application class
  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); // Place text into the stack pane

    // Create a handler for changing text
    EventHandler<ActionEvent> eventHandler = e -> {
      if (text.getText().length() != 0) {
        text.setText("");
      }
      else {
        text.setText("Programming is fun");
      }
    };

    // Create an animation for alternating text
    Timeline animation = new Timeline(
      new KeyFrame(Duration.millis(500), eventHandler));
    animation.setCycleCount(Timeline.INDEFINITE);
    animation.play(); // Start animation
```

Part 1

Example Timeline



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

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 50);
    primaryStage.setTitle("TimelineDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }

  /**
   * The main method is only needed for the IDE with limited
   * JavaFX support. Not needed for running from the command line.
   */
  public static void main(String[] args) {
    launch(args);
  }
}
```

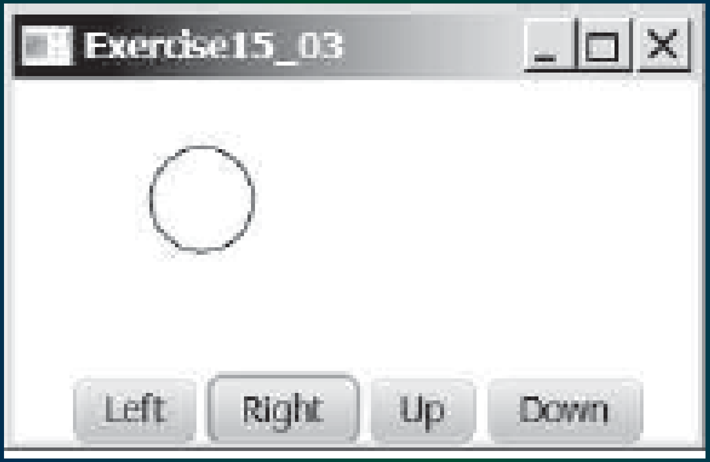


Programming is fun

Part 1

Programming Challenge

Write a program that moves the ball in a pane. You should define a pane class for displaying the ball and provide the methods for moving the ball left, right, up, and down, as shown in the figure below. Check the boundary to prevent the ball from moving out of sight completely.

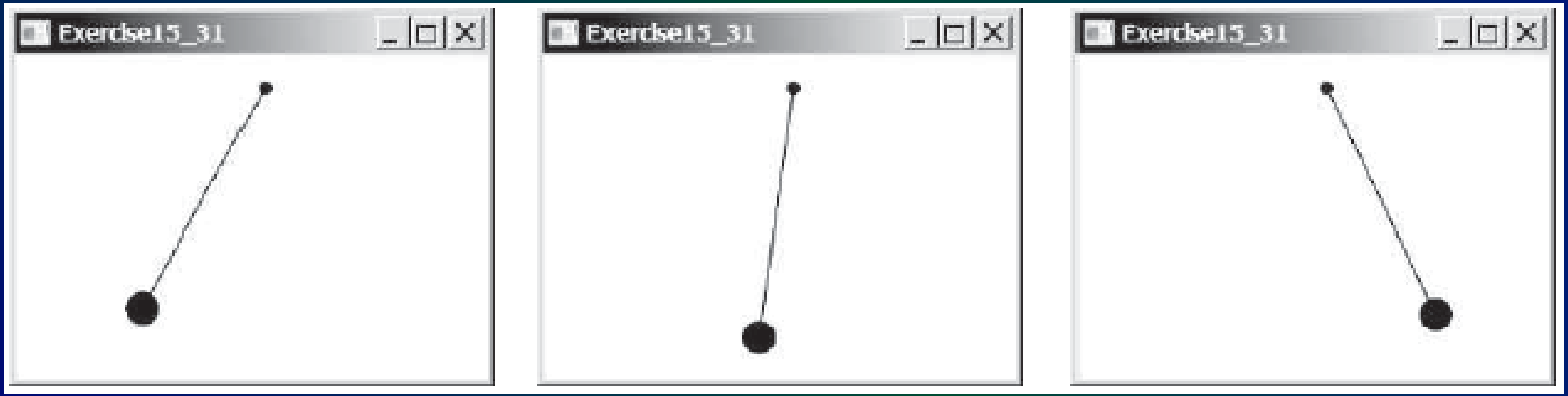


Programming Challenge

Write a program that animates a pendulum swinging, as shown in the figure below.

Press the UP arrow key to increase the speed and the DOWN key to decrease it.

Press the *S* key to stop animation and the *R* key to resume it.


Exercise15_31
Exercise15_31
Exercise15_31