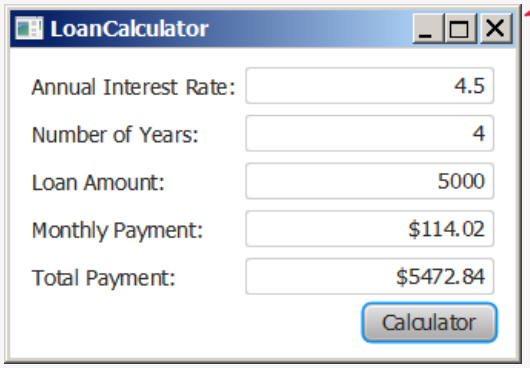
Chapter 15
Event-Driven Programming and Animations

Motivations

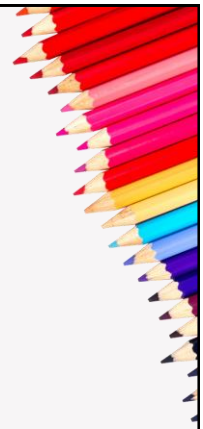
- Suppose you want to write a GUI program that lets the user enter a loan amount, annual interest rate, and number of years and click the Compute Payment button to obtain the monthly payment and total payment. How do you accomplish the task? You have to use event-driven programming to write the code to respond to the button-clicking event.



LoanCalculator Run

2

Objectives (1)



- To get a taste of event-driven programming (§15.1).
- To describe events, event sources, and event classes (§15.2).
- To define handler classes, register handler objects with the source object, and write the code to handle events (§15.3).
- To define handler classes using inner classes (§15.4).
- To define handler classes using anonymous inner classes (§15.5).
- To simplify event handling using lambda expressions (§15.6).

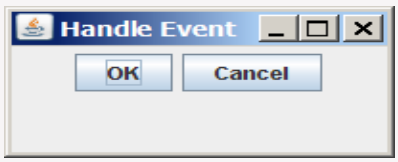
3

Objectives (2)

- To develop a GUI application for a loan calculator (§15.7).
- To write programs to deal with MouseEvents (§15.8).
- To write programs to deal with KeyEvents (§15.9).
- To create listeners for processing a value change in an observable object (§15.10).
- To use the Animation, PathTransition, FadeTransition, and Timeline classes to develop animations (§15.11).
- To develop an animation for simulating a bouncing ball (§15.12).
- To draw, color, and resize a map (§15.13).

15.1 Introduction

- Procedural programming is executed in procedural order.
- In event-driven programming, code is executed upon activation of events.
- The example displays a button in the frame. A message is displayed on the console when a button is clicked.

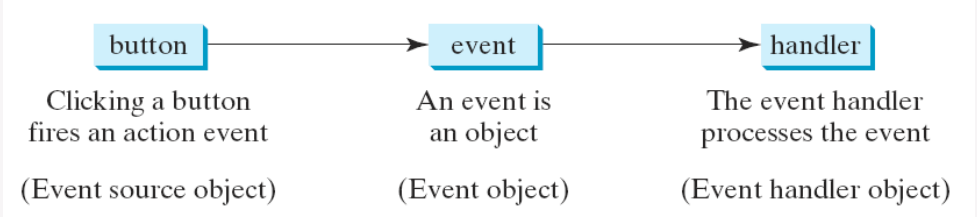


HandleEvent Run

5

Handling GUI Events

- To respond to a button click, you need to write the code to process the button-clicking action.
- The button is an *event source object—where the action originates.*
- *You need to create* an object capable of handling the action event on a button. This object is called an *event handler.*



Handling GUI Events

- Not all objects can be handlers for an action event. To be a handler of an action event, two requirements must be met:
- The object must be an instance of the EventHandler<T extends Event> interface.
  - This interface defines the common behavior for all handlers.
  - <T extends Event> denotes that T is a generic type that is a subtype of Event.
- The EventHandler object handler must be registered with the event source object using the method *source.setOnAction(handler)*.

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");
  }
}
```

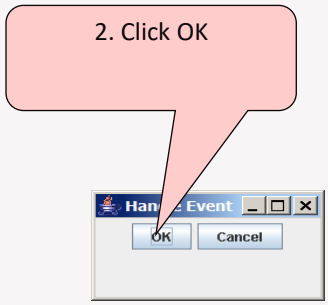
1. Start from the main method to create a window and display it

8

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");
  }
}
```

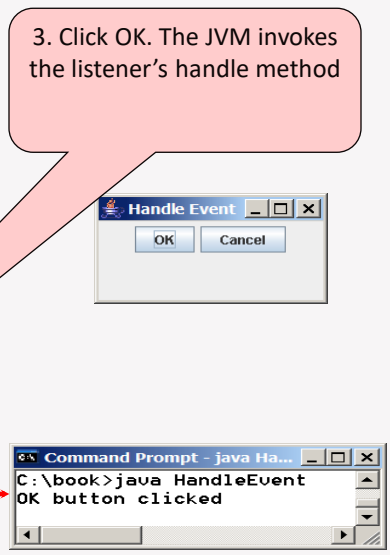


9

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");
  }
}
```

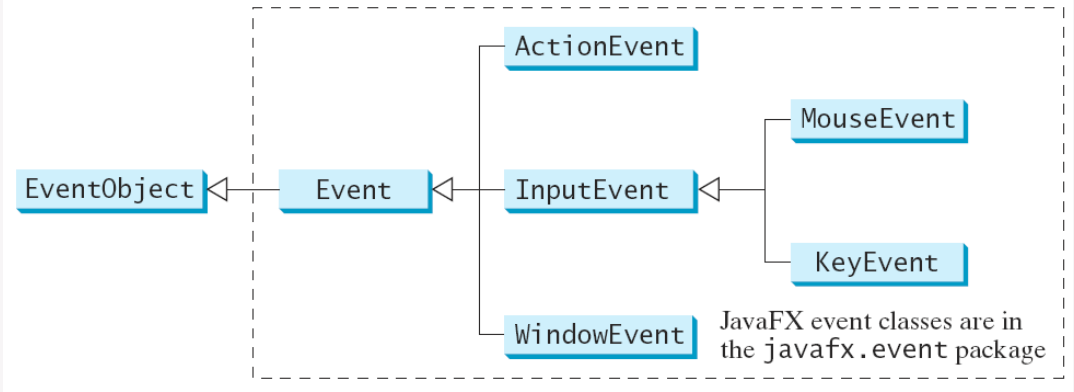


15.2 Events and Event Sources

- 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.
- 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.
- An event in JavaFX is an object of the **javafx.event.Event** class.

11

Event Classes



12

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.
- Table 16.1 lists external user actions, source objects, and event types generated.

13

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>)`

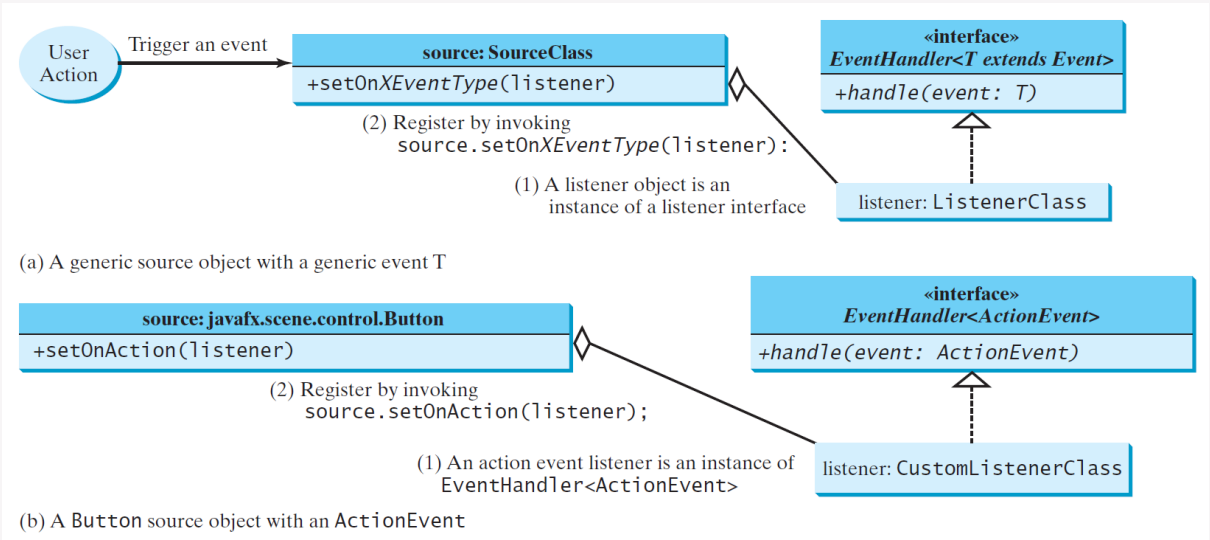
14

15.3 Registering Handlers and Handling Events

- 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.*
- For an object to be a handler for an event on a source object, two things are needed:

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 handler object must be registered by the source object.*

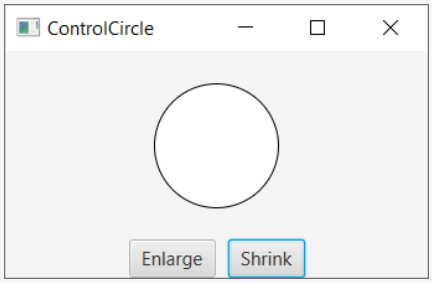
The Delegation Model



16

Example: First Version for ControlCircle (no listeners)

- Now let us consider to write a program that uses two buttons to control the size of a circle.



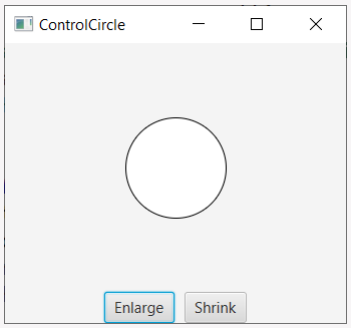
ControlCircleWithoutEventHandling

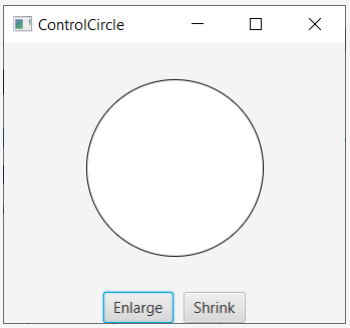
Run

17

Example: Second Version for ControlCircle (with listener for Enlarge)

- Now let us consider to write a program that uses two buttons to control the size of a circle.





ControlCircle

Run

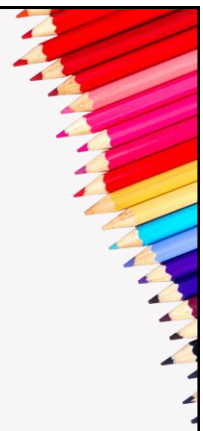
18

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 are defined inside another class.
- The handler class EnlargeHandler is defined as an inner class of the ControlCircle class, that makes the reference variable circlePane accessible from the handle method.

19

15.4 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.*
- Advantages: In some applications, you can use an inner class to make programs simple.
- 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.

20

Inner Classes

```
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)

21

Inner Classes

- 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.
- 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.

22

15.5 Anonymous Inner Class Handlers

- 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
}
```

23

Anonymous Inner Class

```
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

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.

25

Anonymous Inner Class Handlers

```
// Create and register the handler
btUp.setOnAction(new EventHandler<ActionEvent>() {
  @Override // Override the handle method
  public void handle(ActionEvent e) {
    text.setY(text.getY() > 10 ? text.getY() - 5 : 10);
  }
});
```

AnonymousHandlerDemo

Run

26

15.6 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.

```
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

27

Basic Syntax for a Lambda Expression

- The basic syntax for a lambda expression is either

```
(type1 param1, type2 param2, ...) -> expression
```

- or

```
(type1 param1, type2 param2, ...) -> { statements; }
```

- The data type for a parameter may be explicitly declared or implicitly inferred by the compiler.
- The parentheses can be omitted if there is only one parameter without an explicit data type.

28

Lambda Expression

```
(ActiionEvent e) -> {
  circlePane.enlarge(); }
```

(a) Lambda expression with one statement

```
(e) -> {
  circlePane.enlarge(); }
```

(b) Omit parameter data type

```
e -> {
  circlePane.enlarge(); }
```

(c) Omit parentheses

```
e ->
  circlePane.enlarge()
```

(d) Omit braces

Lambda Expression

- The compiler treats a lambda expression as if it is an object created from an anonymous inner class.
- The compiler processes a lambda expression in three steps:
  (1) identify the lambda expression type,
  (2) identify the parameter types, and
  (3) identify statements.

Lambda Expression

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

(1) The compiler recognizes that the lambda expression is an object of the `EventHandler<ActionEvent>` type, because the expression is an argument in the `setOnAction` method.

(2) The compiler recognizes that `e` is a parameter of the `ActionEvent type`, since the `EventHandler<ActionEvent>` interface defines the handle method with a parameter of the `ActionEvent` type.

(3) The compiler recognizes that the code for processing event `e` are the statements in the handle method.

Single Abstract Method Interface (SAM)

- The EventHandler interface contains just one method. The statements in the lambda expression is all for that method. If the interface 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.

LambdaHandlerDemo Run

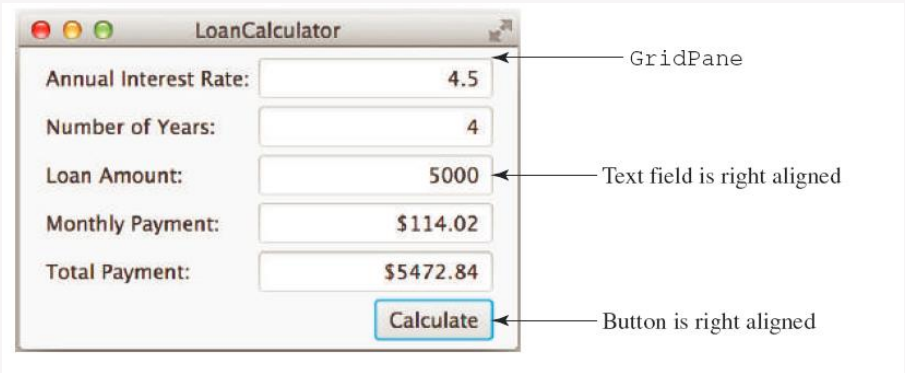
32

Single Abstract Method Interface (SAM)

- In essence, a lambda expression creates an object and the object performs a function by invoking this single method.
- Thus, a SAM interface is also known as a *functional interface*, and an instance of a functional interface is known as a *function object*.
- Since a lambda expression is squarely on defining a function, a lambda expression is also called a *lambda function*.
- The terms lambda expression and lambda function are interchangeable.
- You can define a custom functional interface and use it in a lambda expression.

TestLambda.java

Case Study: Loan Calculator



LoanCalculator

Run

34

15.8 Mouse Events

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

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.

MouseEventDemo

Run

15.9 Key Events

- A KeyEvent is fired whenever a key is pressed, released, or typed on a node or a scene. Key events enable the use of the keys to control and perform actions or get input from the keyboard.
- Only a focused node can receive KeyEvent. **requestFocus()**

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.

KeyEventDemo

Run

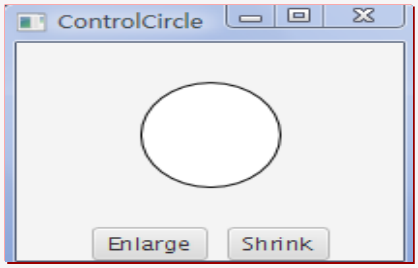
The KeyCode Constants

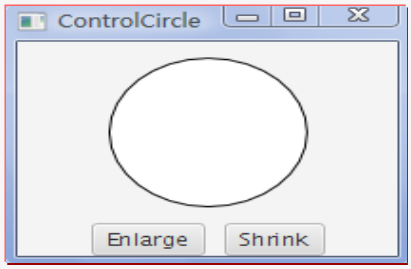
Constant	*Description*	*Constant*	*Description*
HOME	The Home key	CONTROL	The Control key
END	The End key	SHIFT	The Shift key
PAGE_UP	The Page Up key	BACK_SPACE	The Backspace key
PAGE_DOWN	The Page Down key	CAPS	The Caps Lock key
UP	The up-arrow key	NUM_LOCK	The Num Lock key
DOWN	The down-arrow key	ENTER	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

37

Example: Control Circle with Mouse and Key

- We can now add more control for our ControlCircle example in Listing 15.3 to increase/decrease the circle radius by clicking the left/right mouse button or by pressing the U and D keys.





ControlCircleWithMouseAndKey

Run

38

15.10 Listeners for Observable Objects

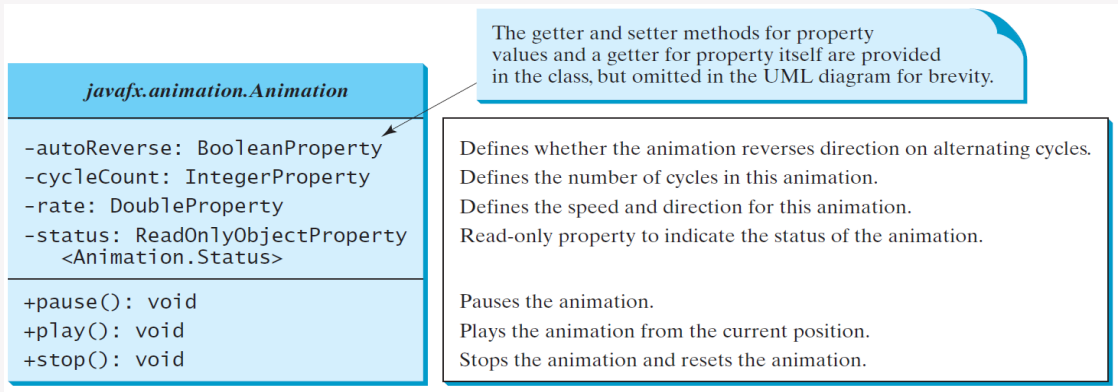
- You can add a listener to process a value change in an observable object.
- An instance of Observable is known as an observable object, which contains the addListener(InvalidationListener listener) method for adding a listener. Once the value is changed in the property, a listener is notified. The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

ObservablePropertyDemo Run

ResizableCircleRectangle Run

15.11 Animation

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

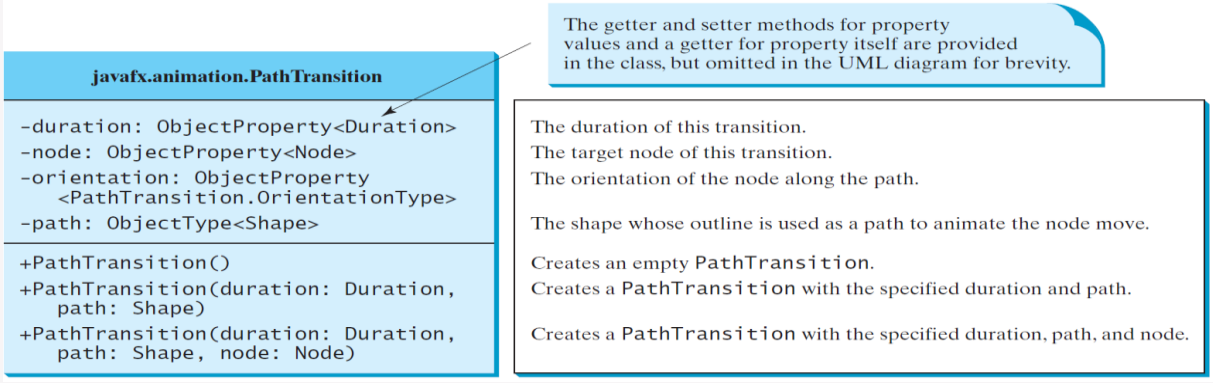


FlagRisingAnimation Run

40

PathTransition

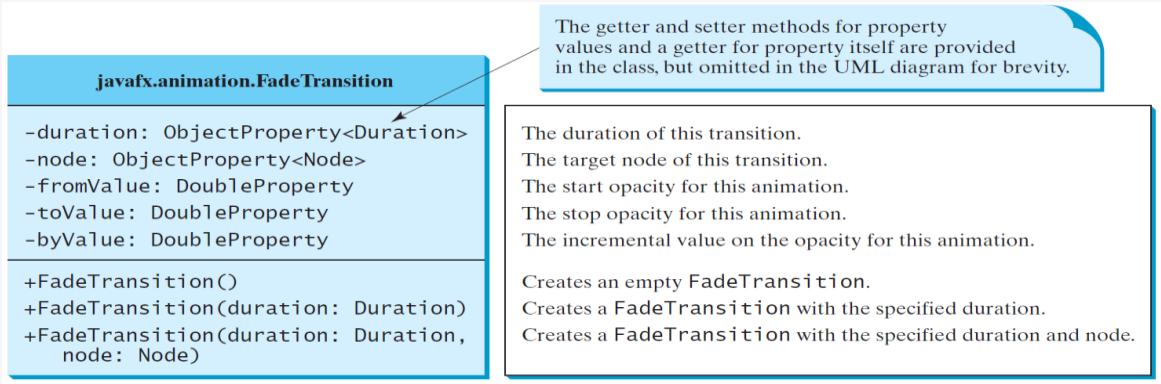
- The PathTransition class animates the the moves of a node along a path from one end to the other over a given time.



PathTransitionDemo Run

FadeTransition

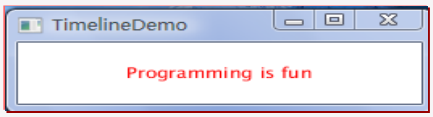
- The **FadeTransition** class animates the change of the opacity in a node over a given time.

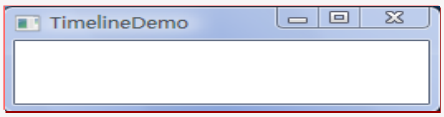


FadeTransitionDemo Run

Timeline

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



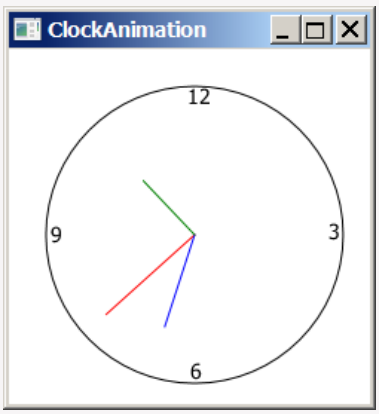


TimelineDemo

Run

43

Clock Animation

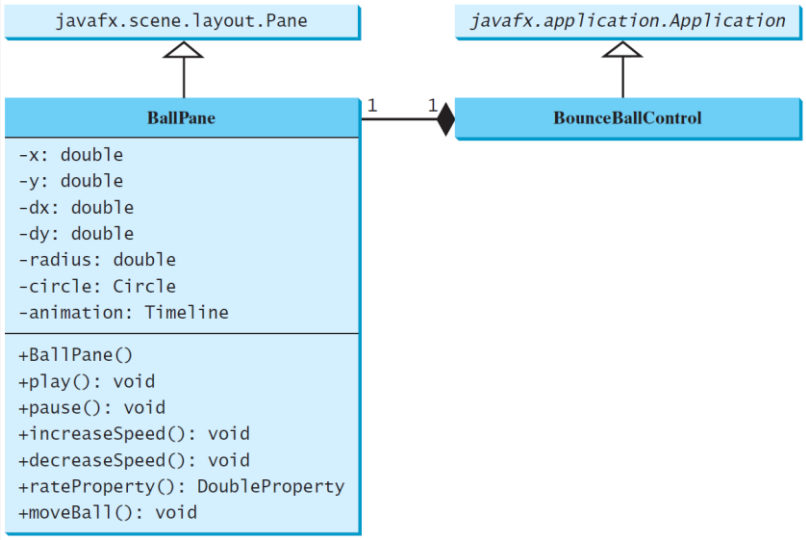


ClockAnimation

Run

44

Case Study: Bouncing Ball



BallPane

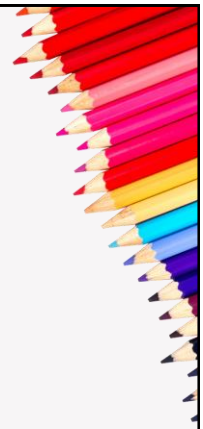
BounceBallControl

Run

45

Chapter Summary

Chapter Summary

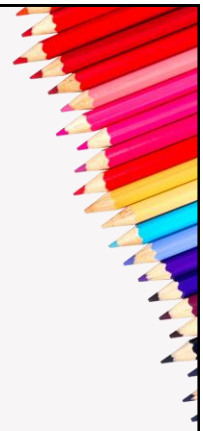


- The root class of the JavaFX event classes is javafx.event.Event, which is a subclass of java.util.EventObject. The subclasses of Event deal with special types of events, such as action events, window events, mouse events, and key events. If a node can fire an event, any subclass of the node can fire the same type of event.
- The handler object's class must implement the corresponding event-handler interface. JavaFX provides a handler interface EventHandler<T extends Event> for every event class T. The handler interface contains the handle(T e) method for handling event e.
- The handler object must be registered by the source object. Registration methods depend on the event type.

Chapter Summary

- An inner class, or nested class, is defined within the scope of another class. An inner class can reference the data and methods defined in the outer class in which it nests.
- An anonymous inner class can be used to shorten the code for event handling. Furthermore, a lambda expression can be used to greatly simplify the event-handling code for functional interface handlers.
- A functional interface is an interface with exactly one abstract method. This is also known as a single abstract method (SAM) interface.

Chapter Summary



- A MouseEvent is fired whenever a mouse button is pressed, released, clicked, moved, or dragged on a node or a scene. The getButton() method can be used to detect which mouse button is pressed for the event.
- A KeyEvent is fired whenever a key is pressed, released, or typed on a node or a scene. The getCode() method can be used to return the code value for the key.
- The abstract Animation class provides the core functionalities for animations in JavaFX. PathTransition, FadeTransition, and Timeline are specialized classes for implementing animations.

Programming Exercises

1, 3, 4, 6, 8, 9, 11, 15,

18, 21, 27, 29, 30, 32