Week4
Event-Driven Programming

Dr. Abdul Razzaq

Abdul.Razzaq@ul.ie

JavaFX Part

UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

JavaFX Controls

Label, CheckBox, ChoiceBox, Slider, ProgressBar, DatePicker, MenuBar, RadioButton and TabPane.

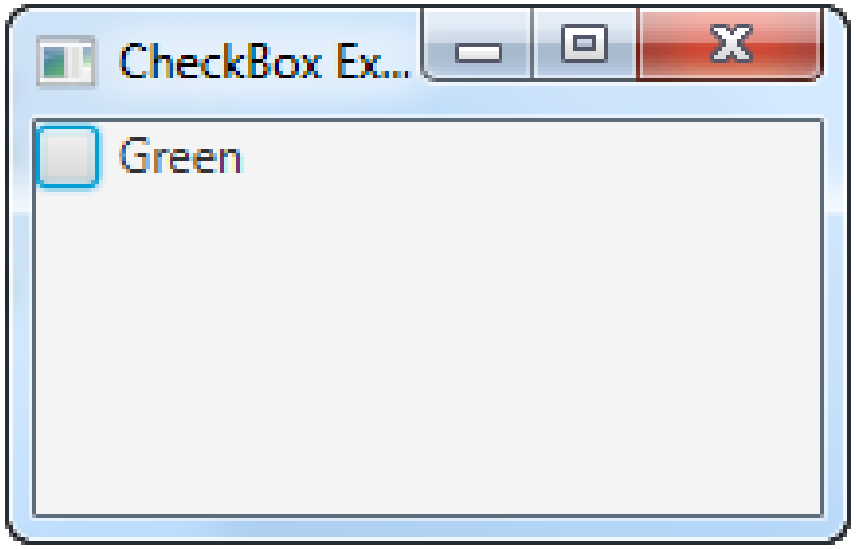
JavaFX Label

Label is a non-editable text control. You have already used that!

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;
public class LabelExperiments extends Application  {
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle(“Label Experiment 1");
        Label label = new Label("My Label");
        Scene scene = new Scene(label, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

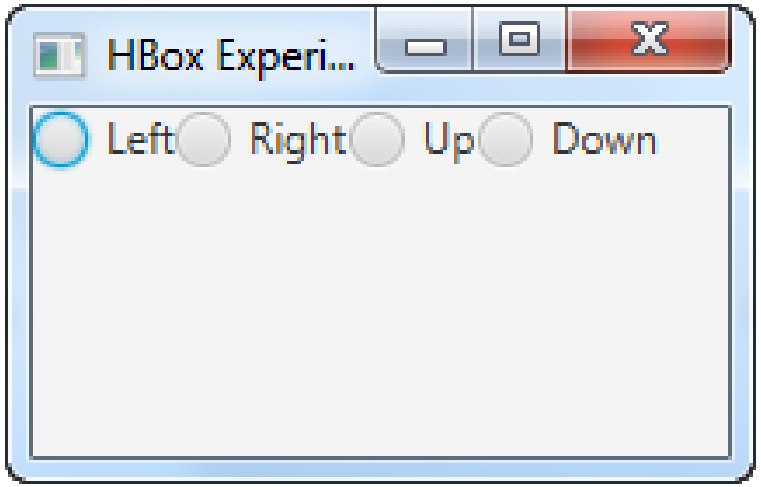
JavaFX CheckBox

CheckBox is a tri-state selection control box showing a checkmark or tick mark when checked. The control has two states by default: checked and unchecked. The **setAllowIndeterminate()** enables the third state: indeterminate



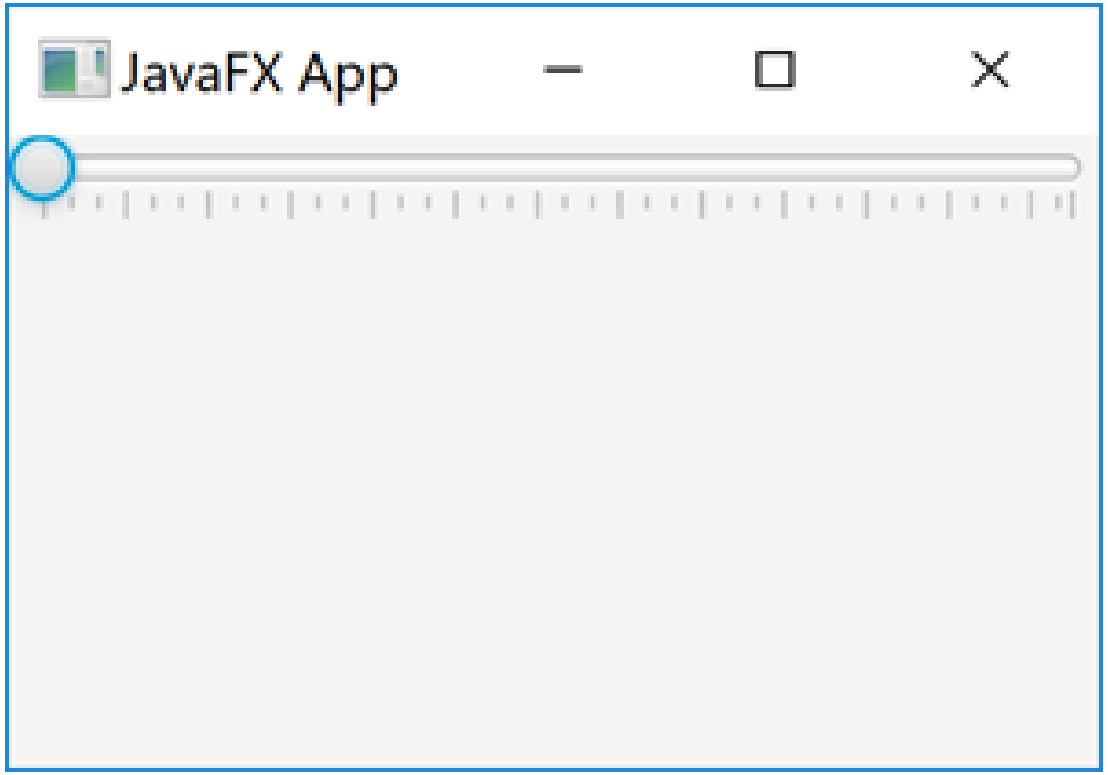
JavaFX RadioButton

RadioButton is usually used to create mutually exclusive series of items. Only one RadioButton can be selected when placed in a ToggleGroup.



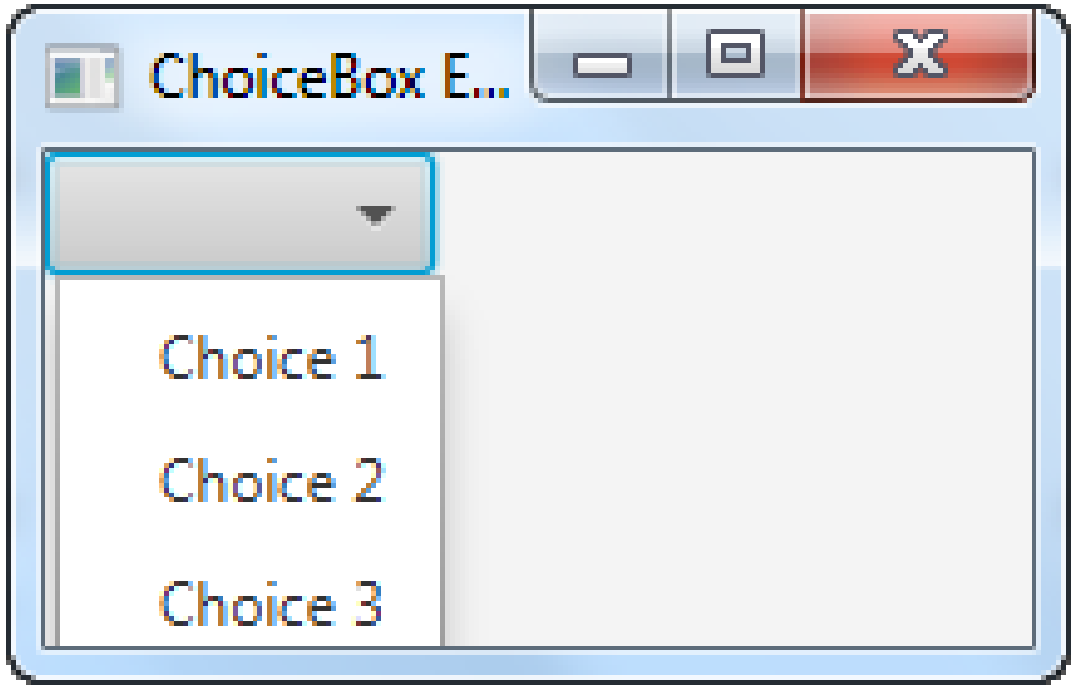
JavaFX Slider

Slider is a control that lets the user graphically select a value by sliding a knob within a bounded interval. The slider can optionally show tick marks and labels indicating different slider position values.



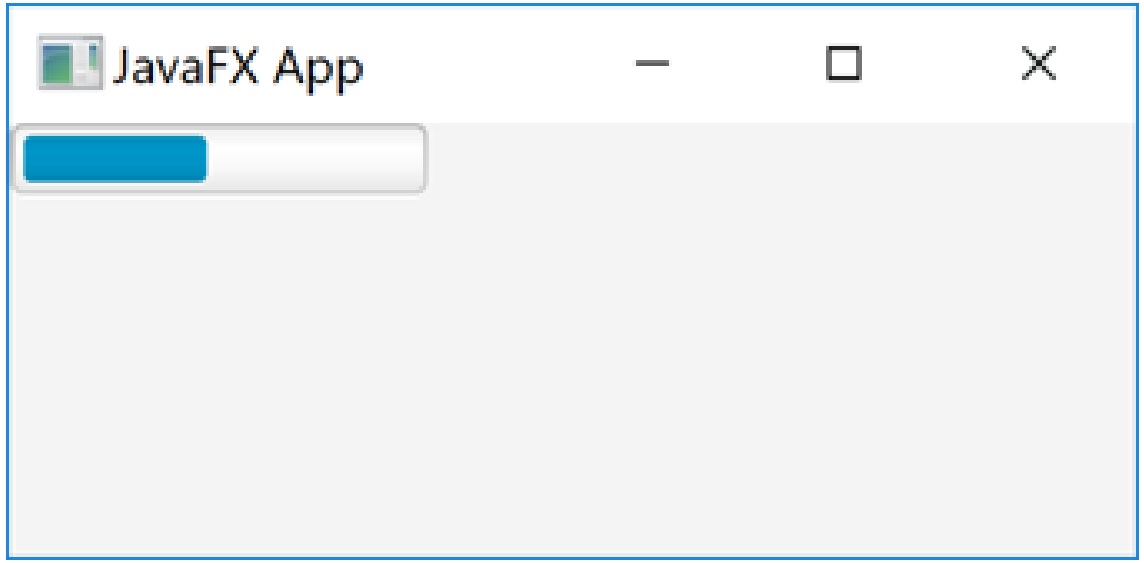
JavaFX ChoiceBox

ChoiceBox is used for presenting the user with a small set of predefined choices. When the user clicks on the box, a list of choices is shown. Only one option can be selected at a time. When this list is not showing, the currently selected choice is shown.



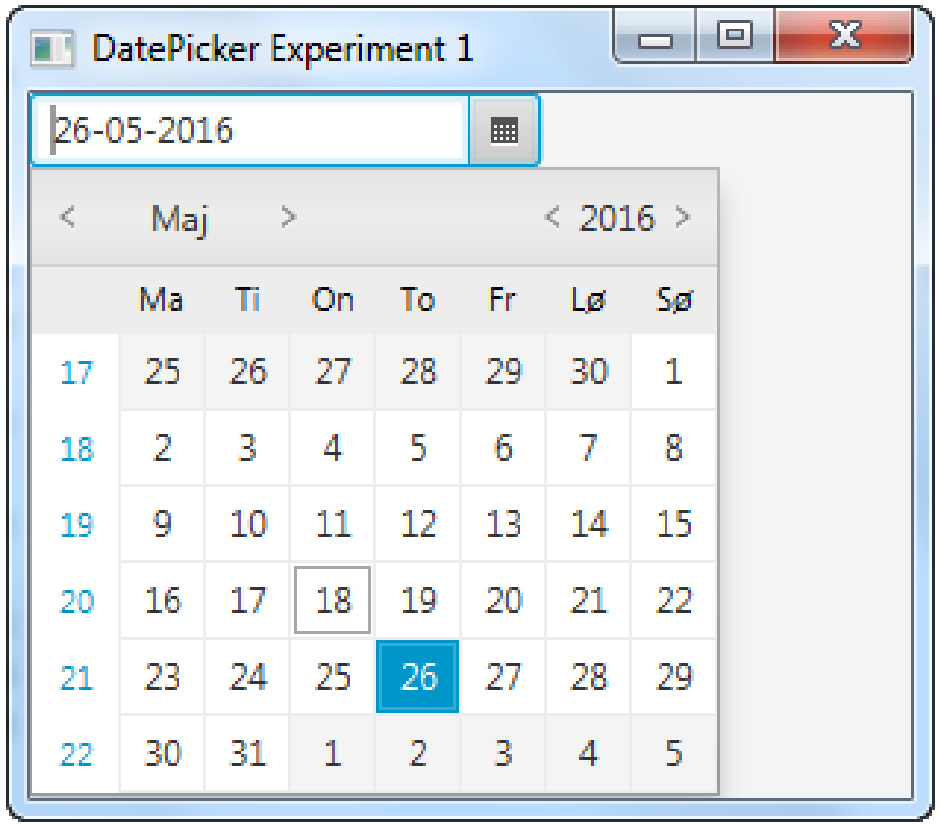
JavaFX ProgressBar

ProgressBar is a control that indicates the processing of a particular task with a completion bar.



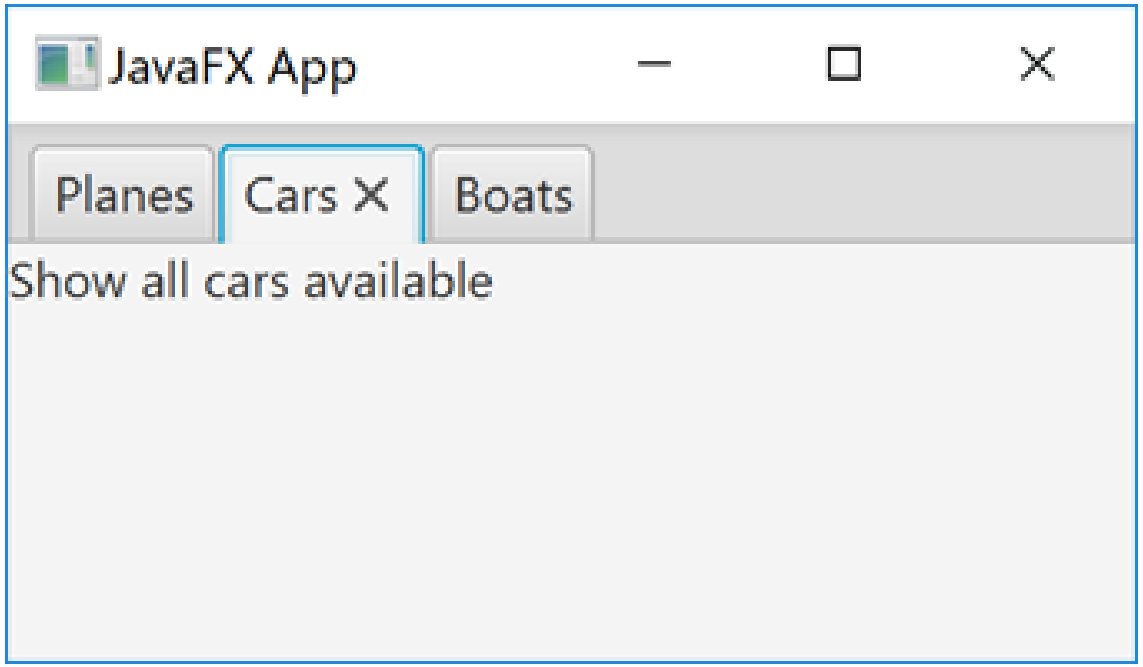
JavaFX DatePicker

DatePicker is a control for choosing a date.



JavaFX TabPane

TabPane is a control that allows switching between a group of Tabs. Only one tab is visible at a time. Tabs in a TabPane can be positioned at any of the four side of the window. The default side is the top side.



Other Controls

- https://jenkov.com/tutorials/javafx/tabpane.html
- https://o7planning.org/11529/javafx-alert-dialog **(Alerts --> Mostly required events to be handled)**

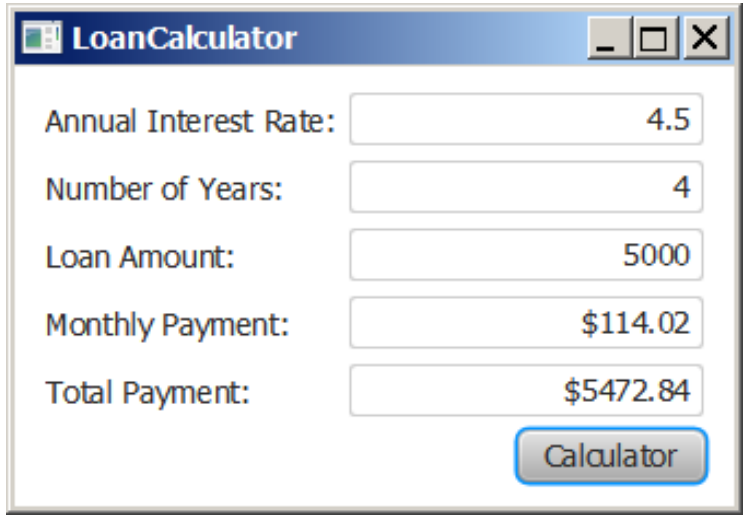
UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

Event-Driven Programming

How to make buttons do something?

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.



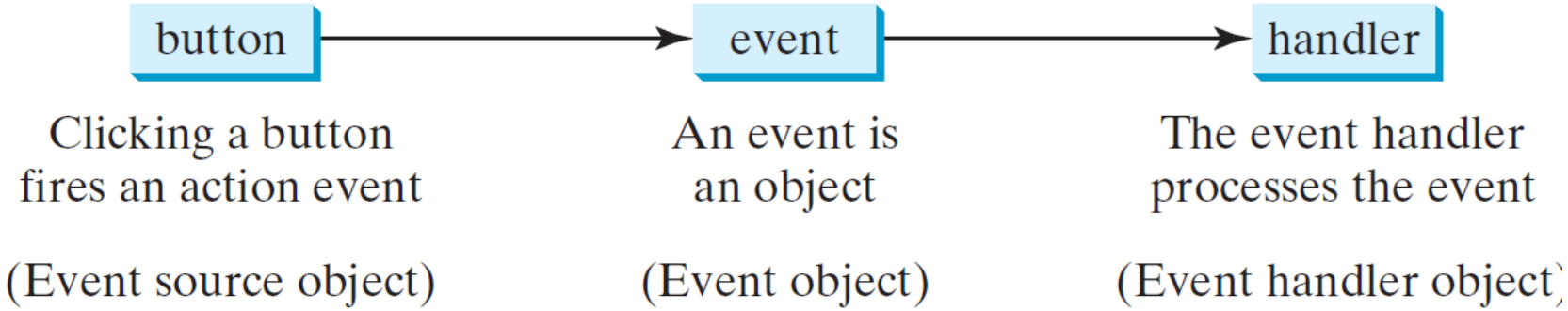
Recall the Procedural vs. Event-Driven Programming

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

Handling GUI Events

Source object (e.g., button)

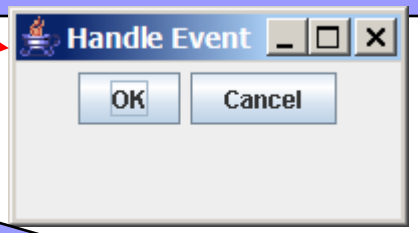
Listener object contains a method for processing the event.



Trace Execution

```
public class HandleEvent extends Application {
  public void start(Stage primaryStage) {
    …
    Button btOK = new Button("OK");
    Button btCancel = new Button("Cancel");
    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

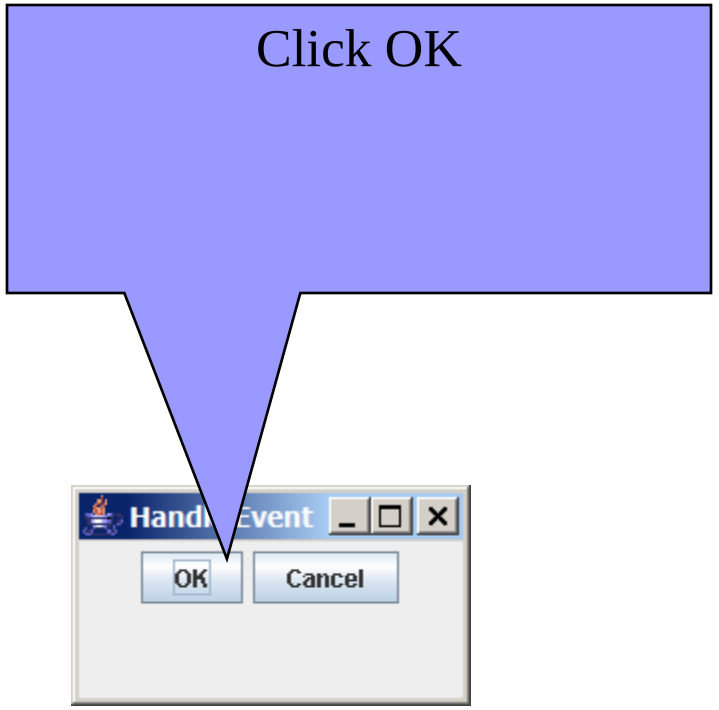


3. Add(Assign) the corresponding Event Listener to OKButton

2. Create an Event Handler Class for OK button

Trace Execution

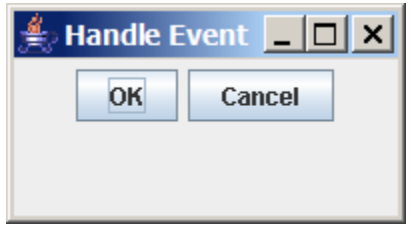
```
public class HandleEvent extends Application {
  public void start(Stage primaryStage) {
    …
    Button btOK = new Button(“OK");
    Button btCancel = new Button(“Cancel");
    OKHandlerClass handler1 = new OKHandlerClass();
    btOK.setOnAction(handler1);
    CancelHandlerClass handler2 = new 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");
  }
}
```

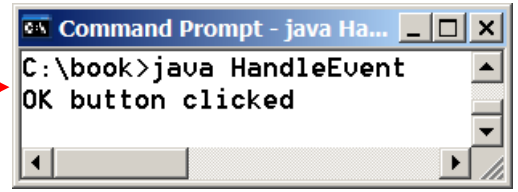


Trace Execution

```
public class HandleEvent extends Application {
  public void start(Stage primaryStage) {
    …
    Button btOK = new Button(“OK");
    Button btCancel = new Button(“Cancel");
    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");
  }
}
```

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





Inner Classes

Inner class: A class is a member of another class.

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.

Inner Classes

```java
public class Test {
  ...
}

public class A {
  ...
}
```

(a)

```java
public class Test {
  ...

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

(b)

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

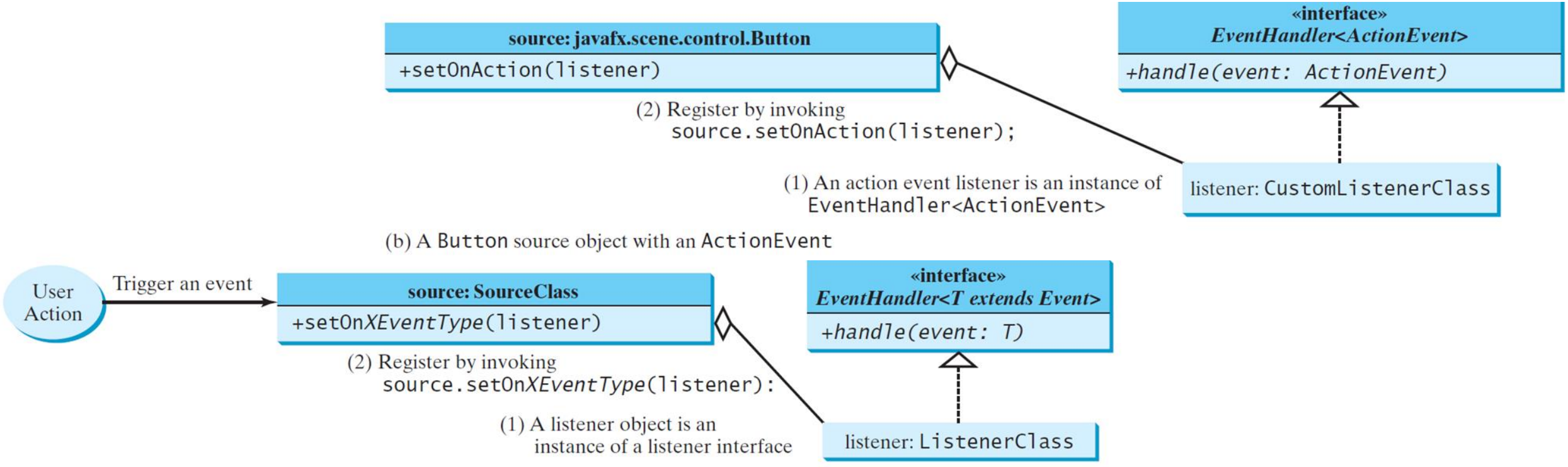
(c)

Inner 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.

The Delegation Model

```
EventHandler<ActionEvent> event1 = new EventHandler<ActionEvent>() {
                    public void handle(ActionEvent e){
                    // deal your event here
                    }
            };
myButton.setOnAction(event1);
```



Delegation Event Model Example

Anonymous Inner Classes

- 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:

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

  btEnlarge.setOnAction(
    new EnlargeHandler());
}

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

(a) Inner class EnlargeListener

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

  btEnlarge.setOnAction(
    new class EnlargeHandlner
      implements EventHandler<ActionEvent>() {
      public void handle(ActionEvent e) {
        circlePane.enlarge();
      }
    });
}
```

(b) Anonymous inner class

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

Basic Syntax for a Lambda Expression

The basic syntax for a lambda expression is either

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

or

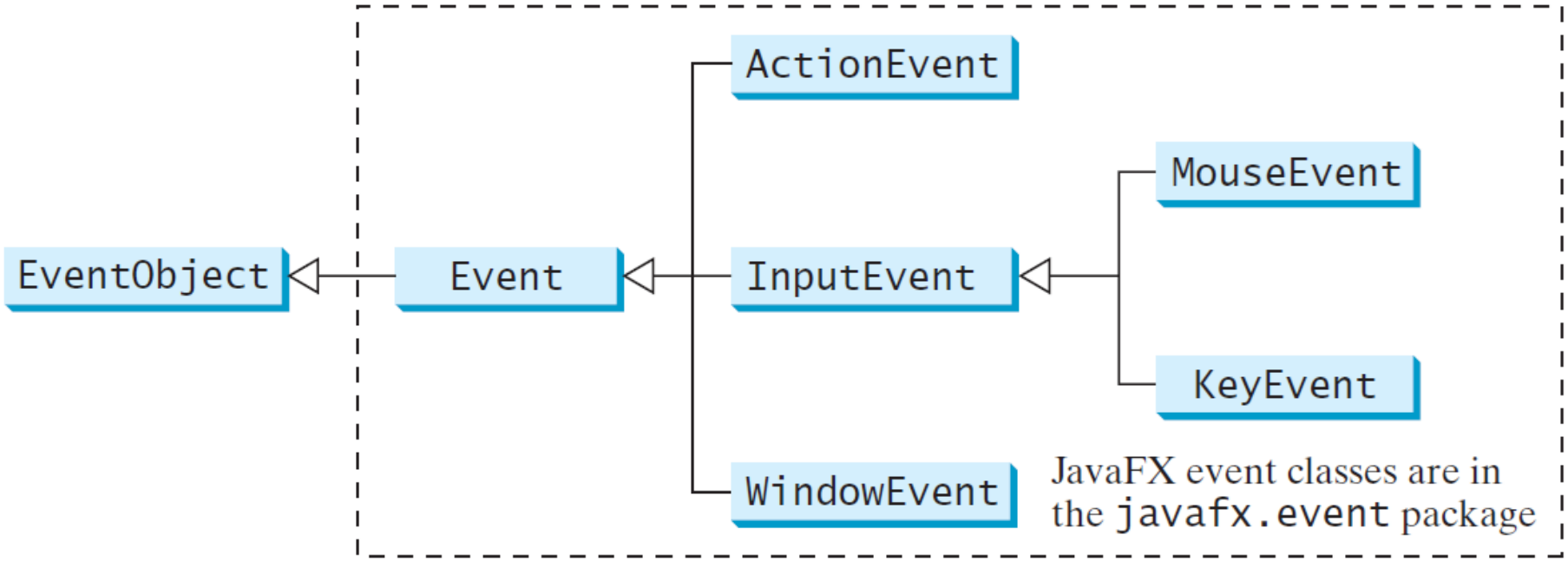
```
(type1 param1, type2 param2, ...) -> { statements; }
```

- 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.

DatePicker: Lambda Example

```
@Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("DatePicker Experiment 1");
        DatePicker datePicker = new DatePicker();
        Label lblDate = new Label("Date will be displayed here");
        Button button = new Button("Read Date");
        button.setOnAction(action -> {
            LocalDate value = datePicker.getValue();
            lblDate.setText(value.toString());
        });
        HBox hbox = new HBox(datePicker, button, lblDate);
        Scene scene = new Scene(hbox, 300, 240);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

Event Classes



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>)`

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

Example: Mouse and Key Events

https://www.tutorialspoint.com/javafx/javafx_event_handling.htm

Assignment : All Mouse Keyboards Events