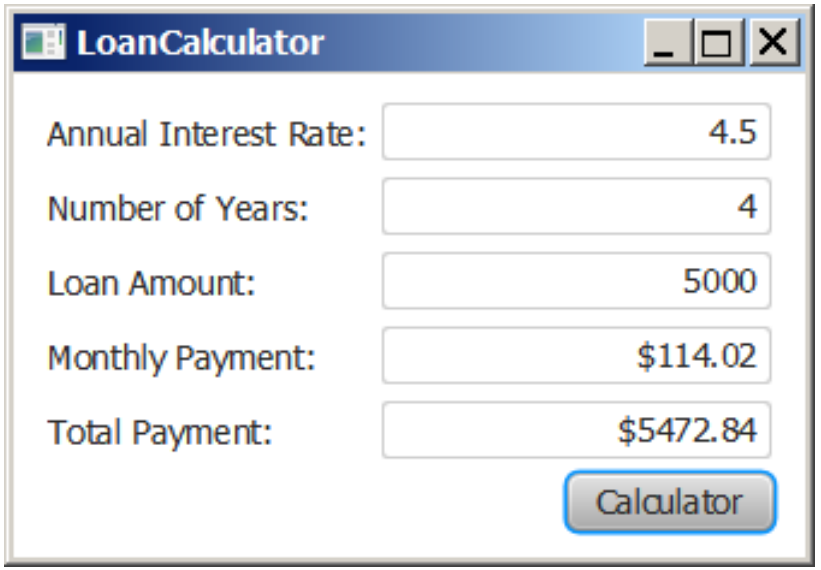
Chapter 15 Event-Driven Programming and Animations

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.



Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order. User cannot interrupt the execution flow and cannot interact with the program.

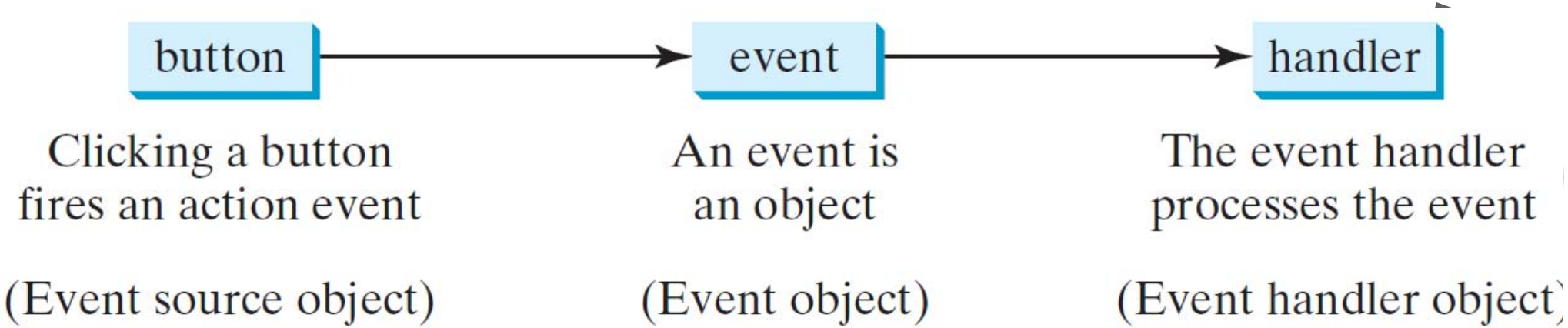
- In event-driven programming, code is executed upon activation of events. An event is initialized by a user's action and the corresponding response to the event is executed by the program.

Handling GUI Events

- An event is an object created from an event source (e.g., button).
- An event can be defined as a signal to the program that something has happened.
- Events are triggered by external user actions, such as mouse movements, mouse clicks, and keystrokes.

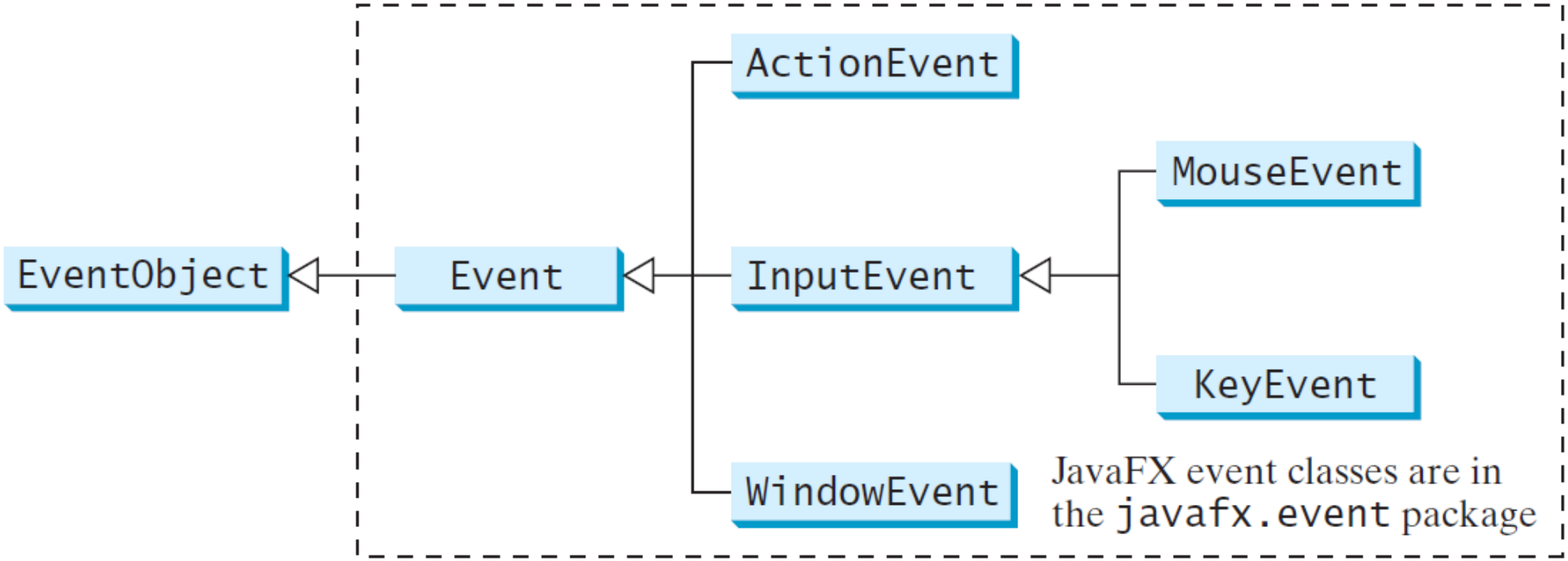
Handling GUI Events

- Firing an event means to create an event object and pass it to an object of type event handler that handles; i.e., executes the response to the event.
  - Source object (e.g., button)
  - Listener object contains a method for processing the event.



Event Classes

- An event is an instance of an **Event** class.
- The root class of the Java event classes is **java.util.EventObject**.
- The root class of the JavaFX event classes is **javafx.event.Event**.



Event Information

- An event object contains whatever properties that are relevant 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, mouse movements, and keystrokes.
- For example, when clicking a button, the button creates and fires an ActionEvent. Here, the button is an event source object, and an ActionEvent is the event object fired by the source object

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>)`

The Delegation Model

- 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.

The Delegation Model

- 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.
  - JavaFX defines a unified handler interface EventHandler<T extends Event> for an event T.
  - The handler interface contains the handle(T e) method for processing the event.
  - For example, the handler interface for ActionEvent is EventHandler<ActionEvent>; each handler for ActionEvent should implement the handle(ActionEvent e) method for processing an ActionEvent.

The Delegation Model

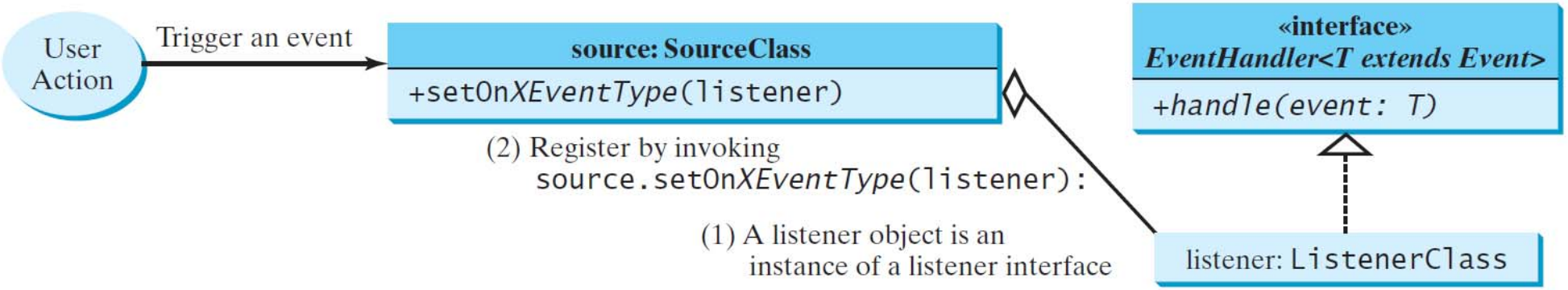
- For an object to be a handler for an event on a source object, two things are needed (continued):

  2.
  - The handler object must be registered by the source object.
  - Registration methods depend on the event type.
  - For ActionEvent, the method is setOnAction().
  - For a mouse pressed event, the method is setOnMousePressed().
  - For a key pressed event, the method is setOnKeyPressed().

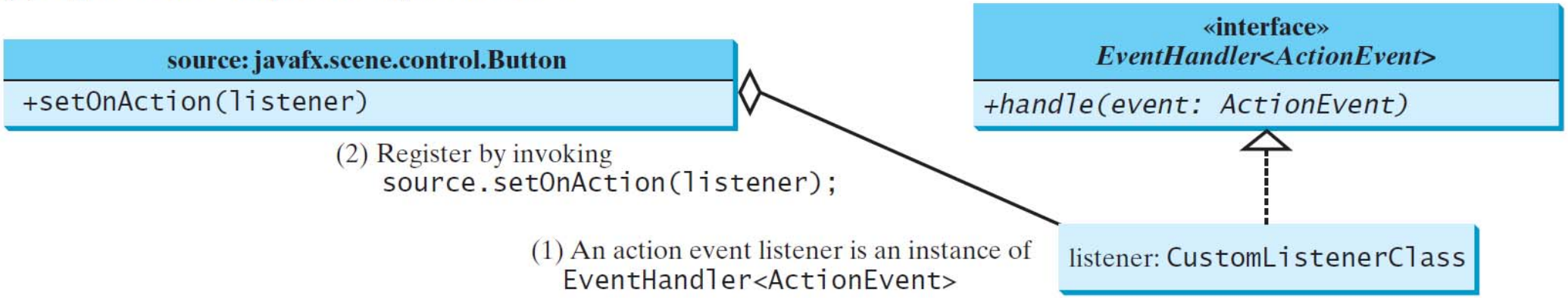
The Delegation Model

- For example, when you click a button, the Button object fires an **ActionEvent** and passes it to invoke the handler's **handle**(**ActionEvent e**) method to handle the event.
- The event object (**e**) contains information relevant to the event, which can be obtained using certain methods.
- For example, you can use **e.getSource**() to obtain the source object that fired the event.

The Delegation Model



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



(b) A `Button` source object with an `ActionEvent`

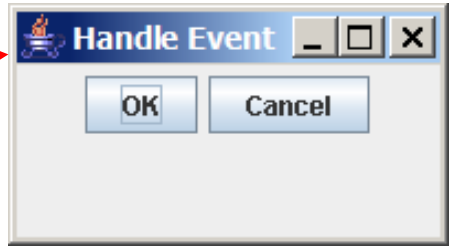
animation

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");
  }
}
```

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

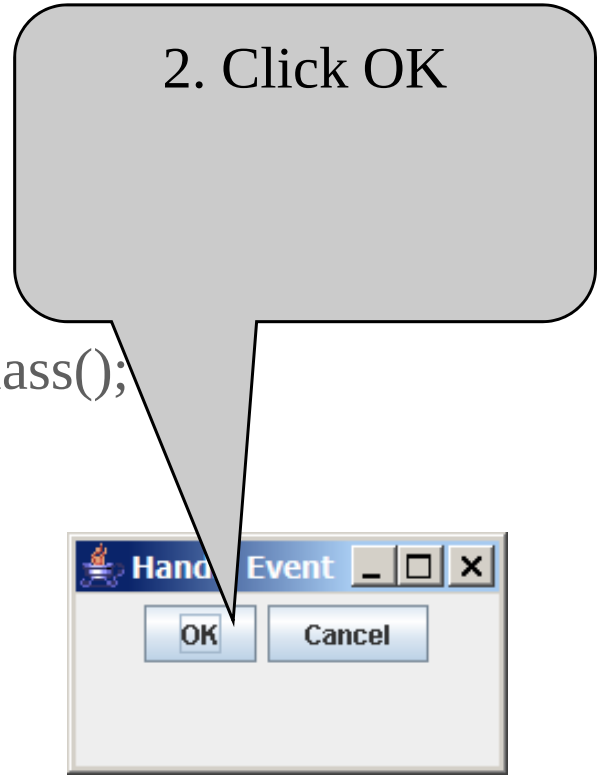


animation

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");
  }
}
```



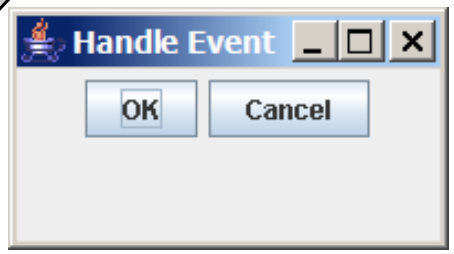
animation

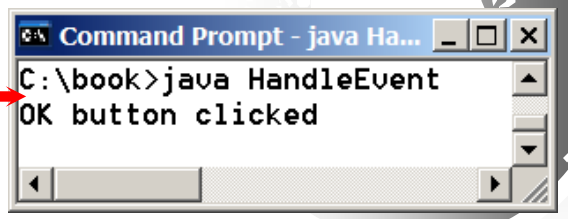
Trace Execution

```
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");
  }
}
```

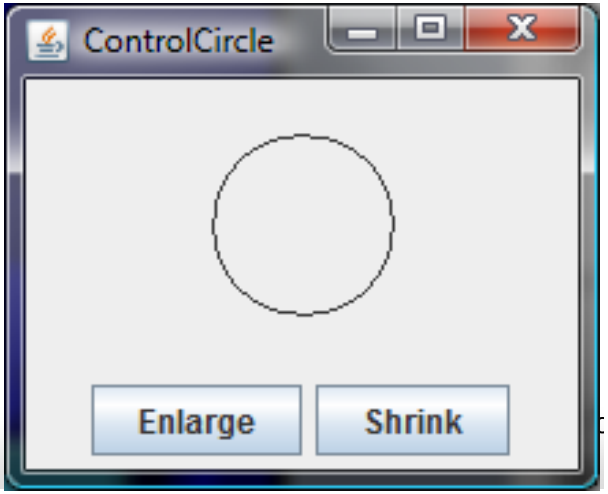
3. The JVM invokes the listener's handle method





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.




```
public class ControlCircleWithoutEventHandling extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    Circle circle = new Circle(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
    pane.getChildren().add(circle);

    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);

    BorderPane borderPane = new BorderPane();
    borderPane.setCenter(pane);
    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
  }
}
```

Example ControlCircle

- How to use the buttons to enlarge or shrink the circle?
- When the Enlarge button is clicked, we want the circle to be repainted with a larger radius.
- When the Shrink button is clicked, we want the circle to be repainted with a smaller radius.
- How can we accomplish this?

Example ControlCircle

- First, we define a new class named **CirclePane** for displaying the circle in a pane.
- This new class displays a circle and provides the **enlarge** and **shrink** methods for increasing and decreasing the radius of the circle.
- It is a good strategy to design a class to model a circle pane with supporting methods so that these related methods along with the circle are coupled in one object.

Example ControlCircle

- Second, inside the ControlCircle class, we create a CirclePane object as a private data field in the ControlCircle class.
- The methods in the ControlCircle class can now access the CirclePane object through this data field.

Example ControlCircle

- Third, we define a handler class named **EnlargeHandler** that implements EventHandler<ActionEvent>.
- In order to make the reference variable circlePane accessible from the handle method, define EnlargeHandler as an inner class of the ControlCircle class.
- Inner classes are defined inside another class. We use an inner class here and will introduce it fully in the next lecture.

Example ControlCircle

- Finally, we register the handler for the **Enlarge** button and implement the handle method in EnlargeHandler to invoke circlePane.enlarge().

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

```
class CirclePane extends StackPane {
  private Circle circle = new Circle(50);

  public CirclePane() {
    getChildren().add(circle);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
  }
  public void enlarge() {
    circle.setRadius(circle.getRadius() + 2);
  }

  public void shrink() {
    circle.setRadius(circle.getRadius() > 2 ?
      circle.getRadius() - 2 : circle.getRadius());
  }
}
```

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

```java
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
  }

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