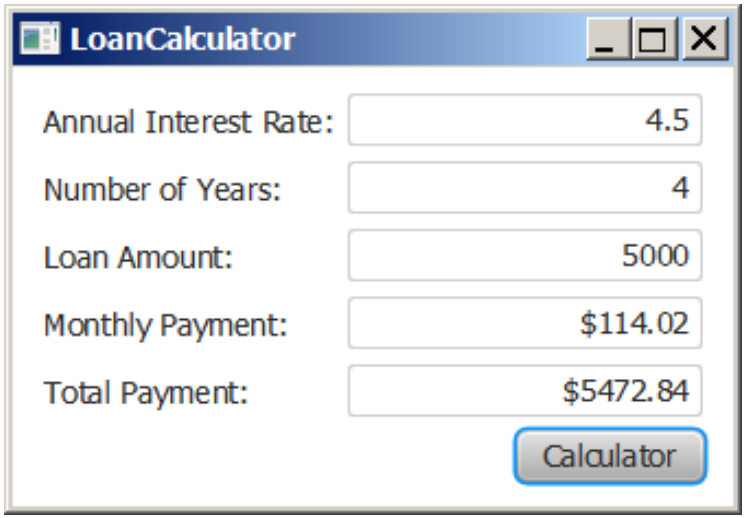
Event-Driven Programming (Delegation Model and Listeners)

Motivations

- Compute monthly payment and total payment for a loan. How do you accomplish the task?

- You have to use *event-driven programming* to write the code to respond to the button-clicking event.



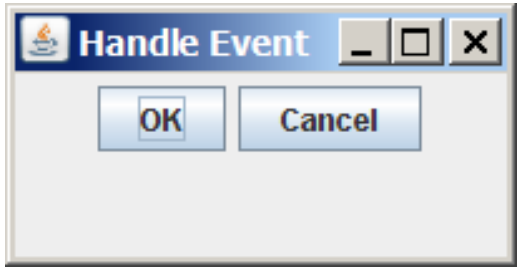
LoanCalculator

Procedural vs. Event-Driven Programming

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

Taste of Event-Driven Programming

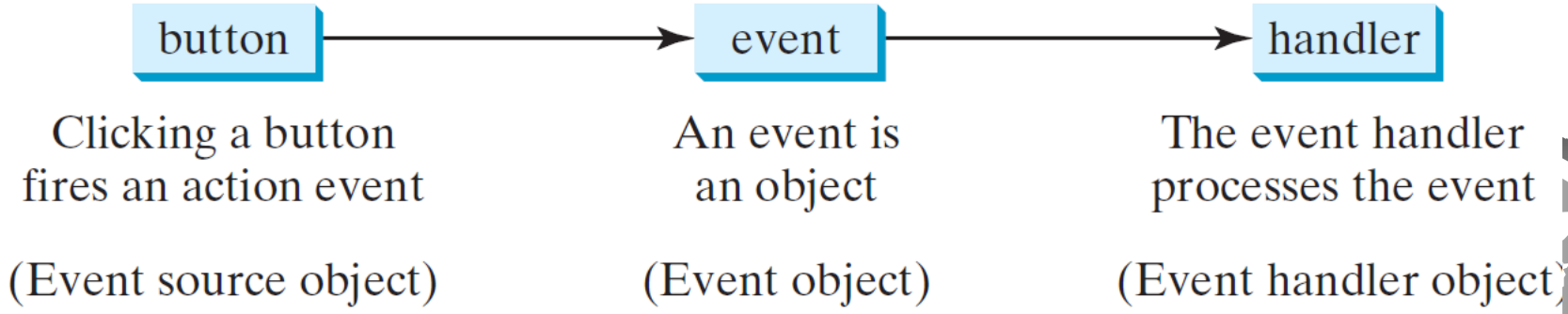
The example displays a button in the frame. A message is displayed on the console when a button is clicked.



HandleEvent

Handling GUI Events

- Source object (e.g., button)
- Listener object contains a method for processing the event.

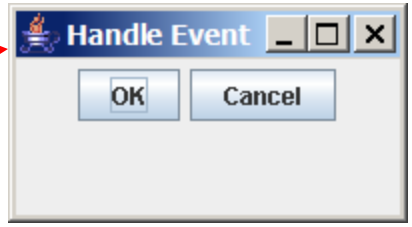


Trace Execution

```
public class HandleEvent extends Application {
  public void start(Stage primaryStage) {
    …
    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



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");
  }
}
```

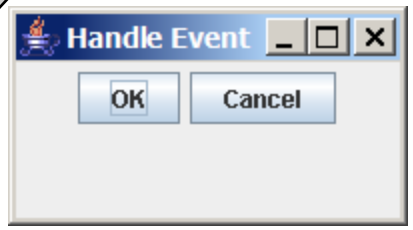
2. Click OK

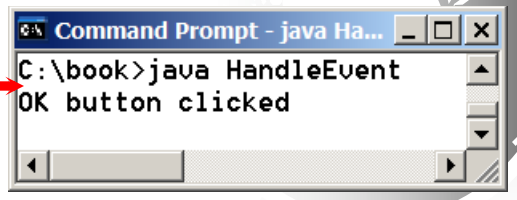
Trace Execution

```
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

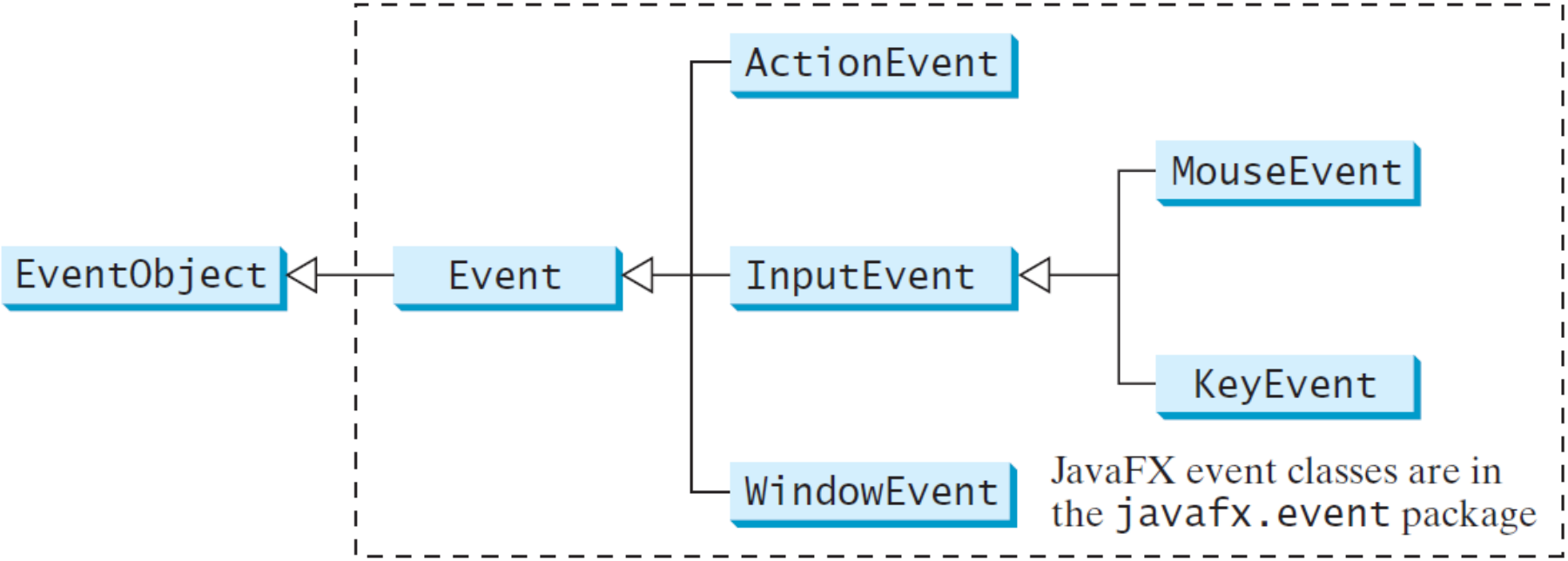




Events

- 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



Event Information

An event object contains

- whatever properties are pertinent to the event
- identification of the source object of the event using the getSource() instance method in the EventObject class

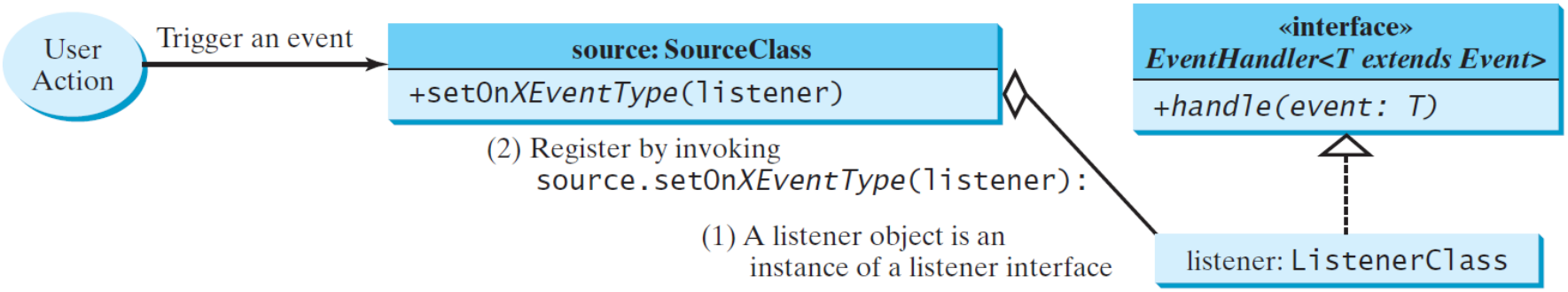
Subclasses of EventObject deal with special types of events such as:

- button actions
- window events
- mouse movements
- keystrokes

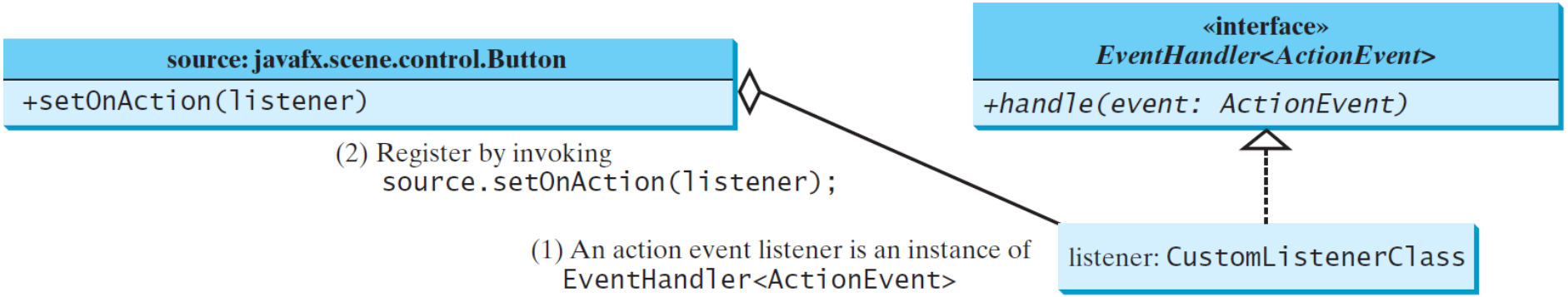
Selected User Actions and Handlers

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>)`

The Delegation Model



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



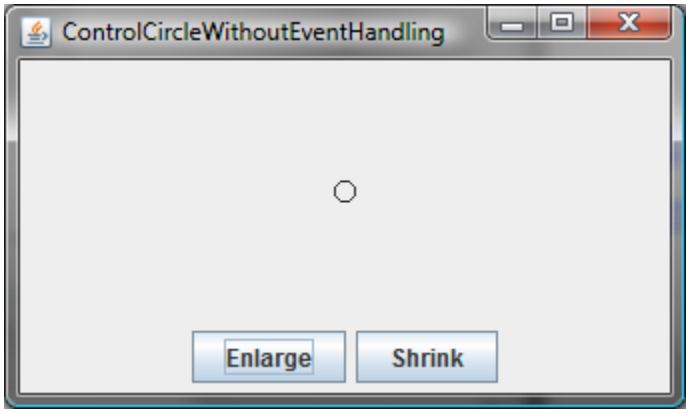
(b) A `Button` source object with an `ActionEvent`

The Delegation Model: Example

```
Button btOK = new Button("OK");
OKHandlerClass handler = new OKHandlerClass();
btOK.setOnAction(handler);
```

Example: First Version for ControlCircle (no listeners)

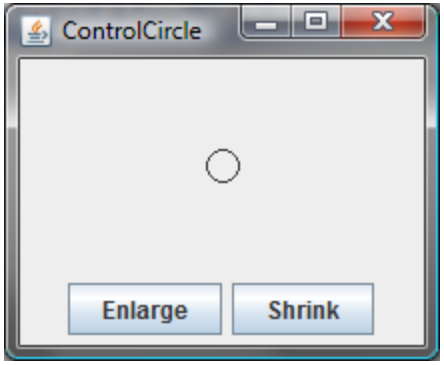
Now let us consider to write a program that uses two buttons to control the size of a circle.

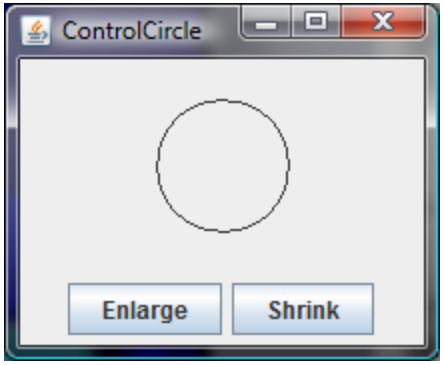


ControlCircleWithoutEventHandling

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

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 application class as an inner class.

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.

ShowInnerClass

Inner Classes, cont.

```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 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.