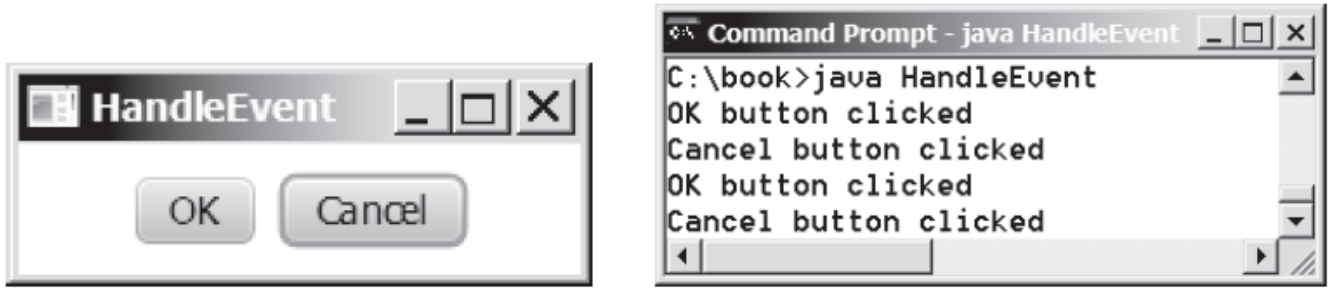
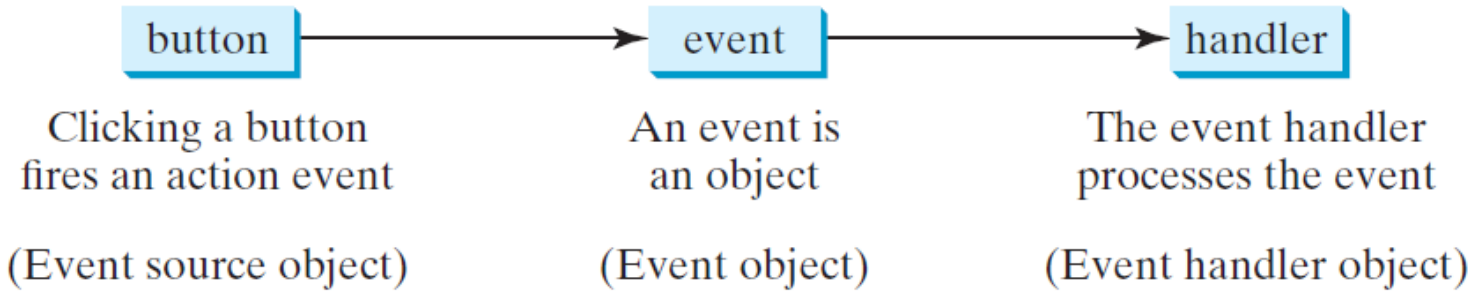
Event-Driven Programming



(a) (b)

(a) The program displays two buttons.

(b) A message is displayed in the console when a button is clicked.



An event handler processes the event fired from the source object.

Not all objects can be handlers for an action event. To be a handler of an action event, two requirements must be met:

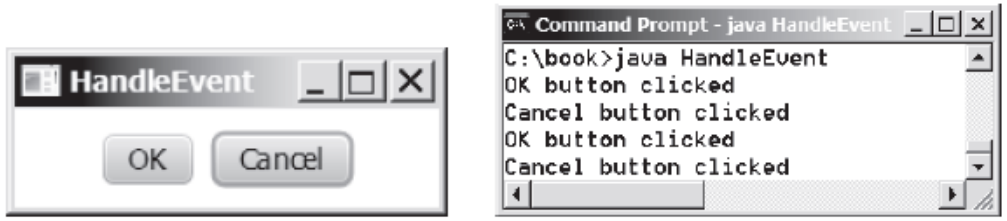
`EventHandler` interface

1. 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`.

`setOnAction(handler)`

2. The `EventHandler` object `handler` must be registered with the event source object using the method `source.setOnAction(handler)`.

The `EventHandler<ActionEvent>` interface contains the `handle(ActionEvent)` method for processing the action event. Your handler class must override this method to respond to the event. Listing 15.1 gives the code that processes the `ActionEvent` on the two buttons. When you click the *OK* button, the message "OK button clicked" is displayed. When you click the *Cancel* button, the message "Cancel button clicked" is displayed, as shown in Figure 15.2.



(a) (b)

FIGURE 15.2 (a) The program displays two buttons. (b) A message is displayed in the console when a button is clicked.

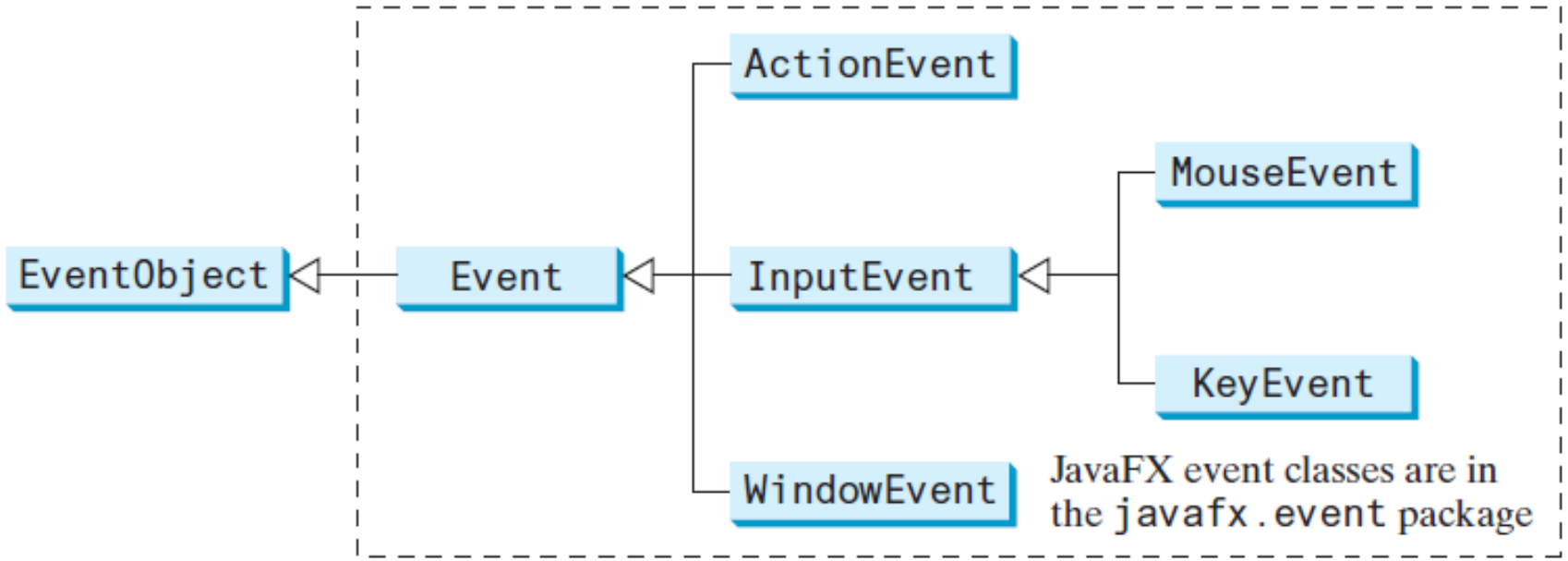
HandleEvent.java

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

public class HandleEvent extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    HBox pane = new HBox(10);
    pane.setAlignment(Pos.CENTER);
    Button btOK = new Button("OK");
    Button btCancel = new Button("Cancel");
    OKHandlerClass handler1 = new OKHandlerClass();
    btOK.setOnAction(handler1);
    CancelHandlerClass handler2 = new CancelHandlerClass();
    btCancel.setOnAction(handler2);
    pane.getChildren().addAll(btOK, btCancel);

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

create handler
register handler
create handler
register handler

```java
10 public class HandleEvent extends Application {
11   @Override // Override the start method in the Application class
12   public void start(Stage primaryStage) {
13     // Create a pane and set its properties
14     HBox pane = new HBox(10);
15     pane.setAlignment(Pos.CENTER);
16     Button btOK = new Button("OK");
17     Button btCancel = new Button("Cancel");
18     OKHandlerClass handler1 = new OKHandlerClass();
19     btOK.setOnAction(handler1);
20     CancelHandlerClass handler2 = new CancelHandlerClass();
21     btCancel.setOnAction(handler2);
22     pane.getChildren().addAll(btOK, btCancel);
23
24     // Create a scene and place it in the stage
25     Scene scene = new Scene(pane);
26     primaryStage.setTitle("HandleEvent"); // Set the stage title
27     primaryStage.setScene(scene); // Place the scene in the stage
28     primaryStage.show(); // Display the stage
29   }
30 }
31
32 class OKHandlerClass implements EventHandler<ActionEvent> {
33   @Override
34   public void handle(ActionEvent e) {
35     System.out.println("OK button clicked");
36   }
37 }
38
39 class CancelHandlerClass implements EventHandler<ActionEvent> {
40   @Override
41   public void handle(ActionEvent e) {
42     System.out.println("Cancel button clicked");
43   }
44 }
```

create handler
register handler
create handler
register handler
handler class
handle event
handler class
handle event



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

TABLE 15.1 User Action, Source Object, Event Type, Handler Interface, and Handler

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>)`

create/register handler

handler class

```
27
28      // Create and register the handler
29      btEnlarge.setOnAction(new EnlargeHandler());
30
31      BorderPane borderPane = new BorderPane();
32      borderPane.setCenter(circlePane);
33      borderPane.setBottom(hBox);
34      BorderPane.setAlignment(hBox, Pos.CENTER);
35
36      // Create a scene and place it in the stage
37      Scene scene = new Scene(borderPane, 200, 150);
38      primaryStage.setTitle("ControlCircle"); // Set the stage title
39      primaryStage.setScene(scene); // Place the scene in the stage
40      primaryStage.show(); // Display the stage
41    }
42
43    class EnlargeHandler implements EventHandler<ActionEvent> {
44      @Override // Override the handle method
45      public void handle(ActionEvent e) {
46        circlePane.enlarge();
47      }
48    }
49  }
50
```

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.

```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)

FIGURE 15.7 An inner class is defined as a member of another class.

Anonymous Inner-Class Handlers

An anonymous inner class is an inner class without a name. It combines defining an inner class and creating an instance of the class into one step.

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

(b) Anonymous inner class

Simplifying Event Handling Using Lambda Expressions

Lambda expressions can be used to greatly simplify coding for event handling.

```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

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. The curly braces can be omitted if there is only one statement. For example, the following lambda expressions are all equivalent. *Note there is no semicolon after the statement in (d).*

```
(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

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

It is processed as follows:

Step 1: The compiler recognizes that the object must be an instance of `EventHandler<ActionEvent>`, since the expression is an argument of the `setOnAction` method as shown in the following figure:

```
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.

Step 2: Since the `EventHandler` interface defines the `handle` method with a parameter of the `ActionEvent` type, the compiler recognizes that `e` is a parameter of the `ActionEvent` type.

Step 3: The compiler recognizes that the code for processing `e` is the statements in the body of the `handle` method.

```
19      // Hold four buttons in an HBox
20      Button btUp = new Button("Up");
21      Button btDown = new Button("Down");
22      Button btLeft = new Button("Left");
23      Button btRight = new Button("Right");
24      HBox hBox = new HBox(btUp, btDown, btLeft, btRight);
25      hBox.setSpacing(10);
26      hBox.setAlignment(Pos.CENTER);
27
28      BorderPane borderPane = new BorderPane(pane);
29      borderPane.setBottom(hBox);
30
31      // Create and register the handler
32      btUp.setOnAction((ActionEvent e) -> {                    lambda handler
33        text.setY(text.getY() > 10 ? text.getY() - 5 : 10);
34      });
35
36      btDown.setOnAction((e) -> {                              lambda handler
37        text.setY(text.getY() < pane.getHeight() ?
38          text.getY() + 5 : pane.getHeight());
39      });
40
41      btLeft.setOnAction(e -> {                                lambda handler
42        text.setX(text.getX() > 0 ? text.getX() - 5 : 0);
43      });
44
45      btRight.setOnAction(e ->                                 lambda handler
46        text.setX(text.getX() < pane.getWidth() - 100?
47          text.getX() + 5 : pane.getWidth() - 100)
48      );
```