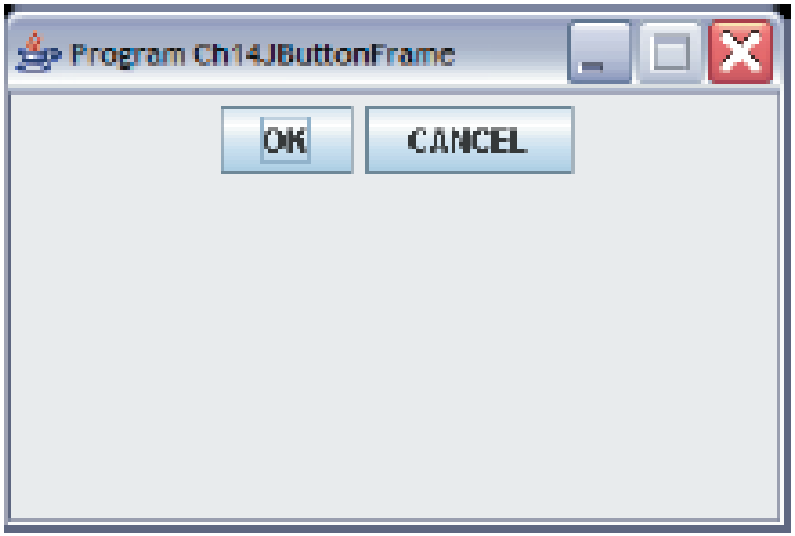
Event Handling

- Placing buttons

Placing a Button

- A **JButton** object is a GUI component that represents a pushbutton.
- Here's an example of how we place a button with **FlowLayout**.

```
contentPane.setLayout(
        new FlowLayout());
okButton
    = new JButton("OK");
cancelButton
    = new JButton("CANCEL");
contentPane.add(okButton);
contentPane.add(cancelButton);
```



Event Handling

- An action involving a GUI object, such as clicking a button, is called an *event.*
- The mechanism to process events is called *event handling*.
- The event-handling model of Java is based on the concept known as the *delegation-based event model.*
- With this model, event handling is implemented by two types of objects:
  - event source objects
  - event listener objects

Event Source Objects

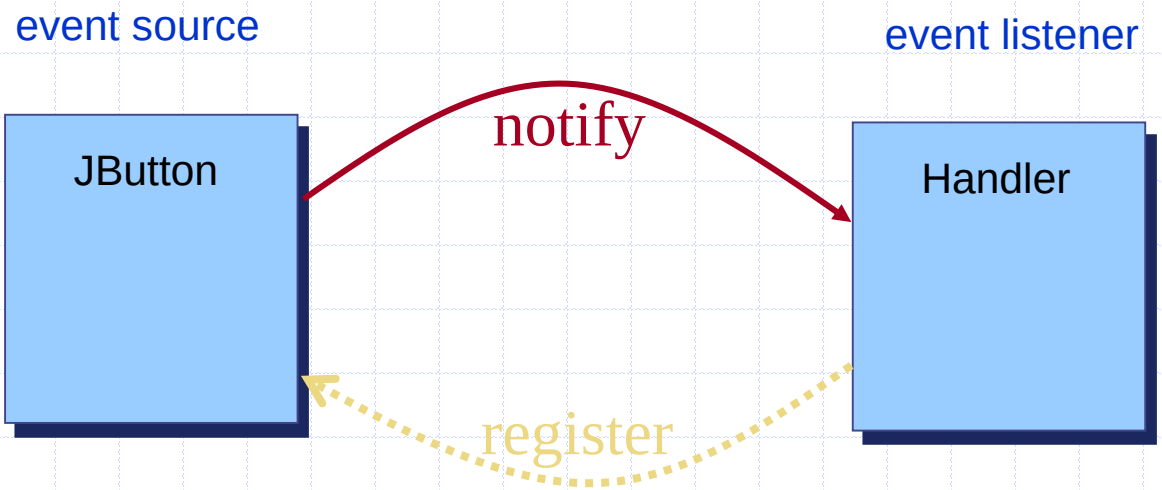
- An event source is a GUI object where an event occurs. We say an event source generates events.
  - **Buttons,**
  - **text boxes,**
  - **list boxes,**
  - **menus**

  are common event sources in GUI-based applications.

Event Listener Objects

- An event listener object is an object that includes a method that gets executed in response to the generated events.
- A listener must be associated, or registered, to a source, so it can be notified when the source generates events.

Connecting Source and Listener

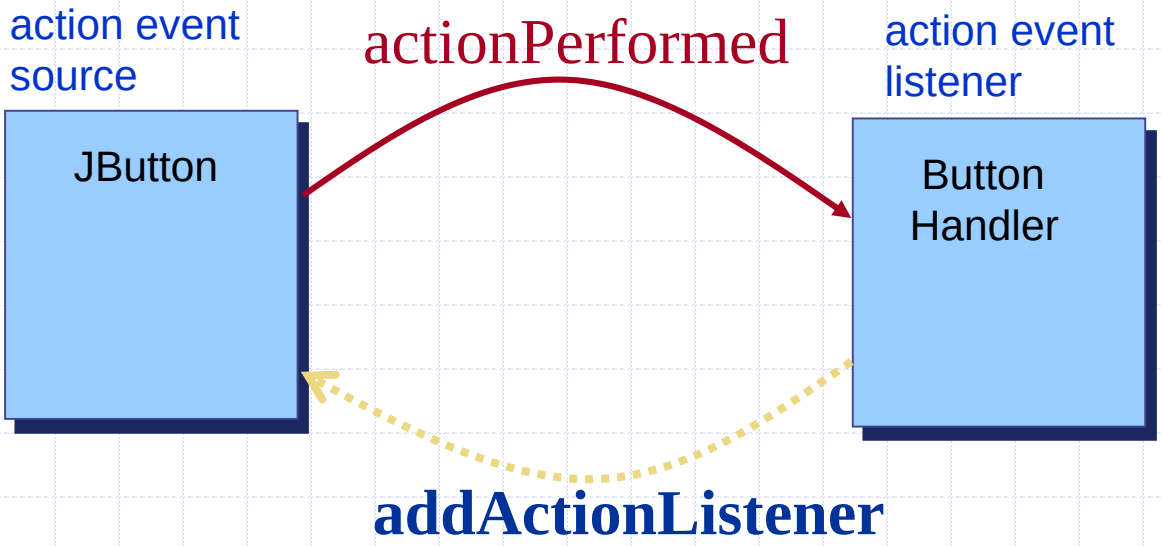


A listener must be **registered** to a event source. Once registered, it will get **notified** when the event source generates events.

Event Types

- Registration and notification are specific to event types
  - **Mouse listener handles mouse events**
  - **Item listener handles item selection events**
  - **and so forth**
- Among the different types of events, the action event is the most common.
  - Clicking on a button generates an action event
  - Selecting a menu item generates an action event
  - and so forth
- Action events are generated by action event sources and handled by action event listeners.

Handling Action Events




```
JButton button = new JButton("OK");
ButtonHandler handler = new ButtonHandler( );

button.addActionListener(handler);
```

ActionListener Interface

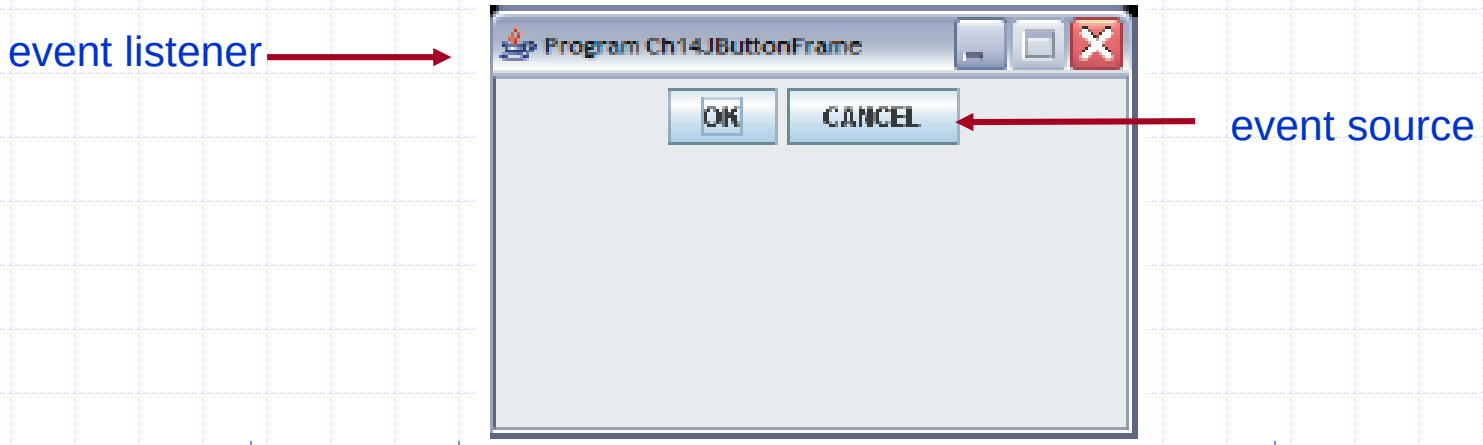
- When we call the **addActionListener** method of an event source, we must pass an instance of a class that implements the **ActionListener** interface.
- The ActionListener interface includes one method named **actionPerformed**.
- A class that implements the ActionListener interface must therefore provide the method body of **actionPerformed**.
- Since actionPerformed is the method that will be called when an action event is generated, this is the place where we put a code we want to be executed in response to the generated events.

The ButtonHandler Class

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class ButtonHandler implements ActionListener {
    . . .
    public void actionPerformed(ActionEvent event) {
        JButton clickedButton = (JButton) event.getSource();

        JRootPane rootPane = clickedButton.getRootPane( );
        Frame     frame    = (JFrame) rootPane.getParent();

        frame.setTitle("You clicked " + clickedButton.getText());
    }
}
```

Container as Event Listener

- Instead of defining a separate event listener such as ButtonHandler, it is much more common to have an object that contains the event sources be a listener.
  - Example: We make this frame a listener of the action events of the buttons it contains.



JButtonFrameHandler

```
. . .
class myJButtonFrameHandler extends JFrame
                                implements ActionListener {
   . . .
   public void actionPerformed(ActionEvent event) {
        JButton clickedButton
                    = (JButton) event.getSource();

        String  buttonText = clickedButton.getText();

        setTitle("You clicked " + buttonText);
    }
}
```