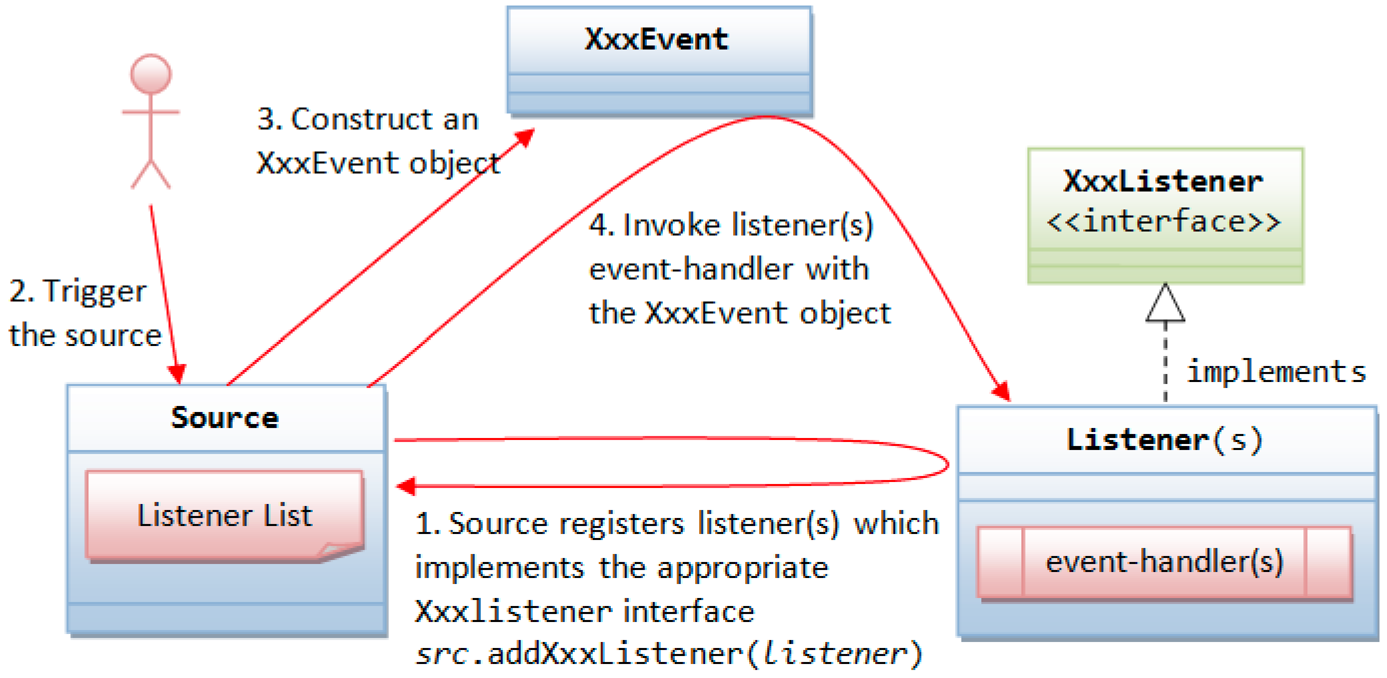
Event Handling

By: Dinesh Amatya

Event Handling

- A listener object is an instance of a class that implements a special interface called a listener interface.
- An event source is an object that can register listener objects and send them event objects.
- The event source sends out event objects to all registered listeners when that event occurs.
- The listener objects will then use the information in the event object to determine their reaction to the event.


XxxEvent
3. Construct an XxxEvent object
2. Trigger the source
4. Invoke listener(s) event-handler with the XxxEvent object
XxxListener <<interface>>
implements
Source
Listener List
Listener(s)
event-handler(s)
1. Source registers listener(s) which implements the appropriate Xxxlistener interface
src.addXxxListener(listener)

Nested (Inner) Classes

```
public class MyOuterClass {   // outer class defined here

   private int outerClassInstanceVariable;
   ......
   private class MyNestedClass1 {   // an nested class defined inside the outer class
          public void increaseCount()
                    {
                              outerClassInstanceVariable++;
                    }
   }
   public static class MyNestedClass2 { ... }  // an "static" nested class defined inside the
outer class
   ......
}
```

→nested class is a proper class
→ is the member of outer class
→ can access the private members/methods of the enclosing outer class
→ private inner class only accessible by enclosing outer class

Anonymous Inner Class

```
Class
{
JtextField textField = new JtextField();
Method()
{
	Jbutton btn = new JButton()

	btn.addActionListener(new ActionListener() {
	@Override
	public void actionPerformed(ActionEvent e) {
		 ++count;
		textField.setText(count + "");
	}
	});
}
}
```

→ The anonymous inner class is defined inside a method, instead of a member of the outer class (class member)
→ It is local to the method and cannot be marked with access modifier
→ An anonymous inner class must always extend a superclass or implement an interface. The keyword "extends" or "implements" is NOT required in its declaration
→ An anonymous inner class must implement all the abstract methods in the superclass or in the interface.

```
public class Calculator extends JFrame{
    JTextField field = new JTextField();

    public Calculator()
    {
        setSize(500,500);
        JPanel panelNorth = new JPanel();
        JPanel panelSouth = new JPanel();

        field.setSize(50,50);
        field.setText("Initial text");
        field.setVisible(true);

        panelNorth.add(field);

        JButton button1 = new JButton("1");
        JButton button2 = new JButton("2");

        button1.addActionListener(new MyListener(1));
        button2.addActionListener(new MyListener(2));

        panelSouth.add(button1);
        panelSouth.add(button2);

        add(panelNorth, BorderLayout.NORTH);
        add(panelSouth,BorderLayout.SOUTH);
        setVisible(true);
}
```

```
public class Calculator extends JFrame{
    JTextField field = new JTextField();

    public Calculator()
    {
        setSize(500,500);
        JPanel panelNorth = new JPanel();
        JPanel panelSouth = new JPanel();

        field.setSize(50,50);
        field.setText("Initial text");
        field.setVisible(true);

        panelNorth.add(field);

        JButton button1 = new JButton("1");
        JButton button2 = new JButton("2");

        button1.addActionListener(new MyListener(1));
        button2.addActionListener(new MyListener(2));

        panelSouth.add(button1);
        panelSouth.add(button2);

        add(panelNorth, BorderLayout.NORTH);
        add(panelSouth,BorderLayout.SOUTH);
        setVisible(true);
}
public class MyListener implements ActionListener
    {
        private  int buttonVal;
        public MyListener(int buttonVal)
        {
            this.buttonVal=buttonVal;
        }
        @Override
        public void actionPerformed(ActionEvent e) {
            System.out.println("in action performed:"+e);
            field.setEnabled(false);
            field.setText(buttonVal);
        }
    }


    public static void main(String[] args) {
        Calculator calculator = new Calculator();
    }
}
```

Adapter Classes

```
public interface WindowListener
{
        void windowOpened(WindowEvent e);
        void windowClosing(WindowEvent e);
        void windowClosed(WindowEvent e);
        void windowIconified(WindowEvent e);
        void windowDeiconified(WindowEvent e);
        void windowActivated(WindowEvent e);
        void windowDeactivated(WindowEvent e);
}


WindowListener listener = . . .;
frame.addWindowListener(listener);

class Terminator extends WindowAdapter
{
        public void windowClosing(WindowEvent e)
        {
                If (user agrees)
                System.exit(0);
        }
}
```

Key Events

```
textArea.addKeyListener(new KeyListener() {

        /** Handle the key typed event from the text field. */
        public void keyTyped(KeyEvent e) {
            System.out.println("typed");
            System.out.println(e.getKeyChar());
        }

        /** Handle the key pressed event from the text field. */
        public void keyPressed(KeyEvent e) {
            System.out.println("pressed");
        }

        /** Handle the key released event from the text field. */
        public void keyReleased(KeyEvent e) {
            System.out.println("released");
        }
```

Focus Events

```
label.addFocusListener(new FocusListener() {

     public void focusGained(FocusEvent e) {
        sout("Focus gained");
        sout (e.getComponent().getClass().getName());
    }

    public void focusLost(FocusEvent e) {
        sout("Focus lost");
    }
};
```

Item Events

```
component.addItemListener(new ItemListener() {

    public void itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED) {
        label.setVisible(true);
        ...
    } else {
        label.setVisible(false);
    }
}
};
```

Working with 2D shapes

```
class DrawComponent extends Jcomponent
{
public void paintComponent(Graphics g)
 {
        Graphics2D g2 = (Graphics2D) g;
// draw a rectangle
        Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width,
                height);
        g2.draw(rect);
// draw a diagonal line
         g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY +
                height));
// draw a circle with the same center
        double centerX = rect.getCenterX();
        double centerY = rect.getCenterY();
        double radius = 150;
         Ellipse2D circle = new Ellipse2D.Double();
        circle.setFrameFromCenter(centerX, centerY, centerX + radius,
                centerY + radius);
        g2.draw(circle);

    }

}
```

Mouse Events

MouseListener

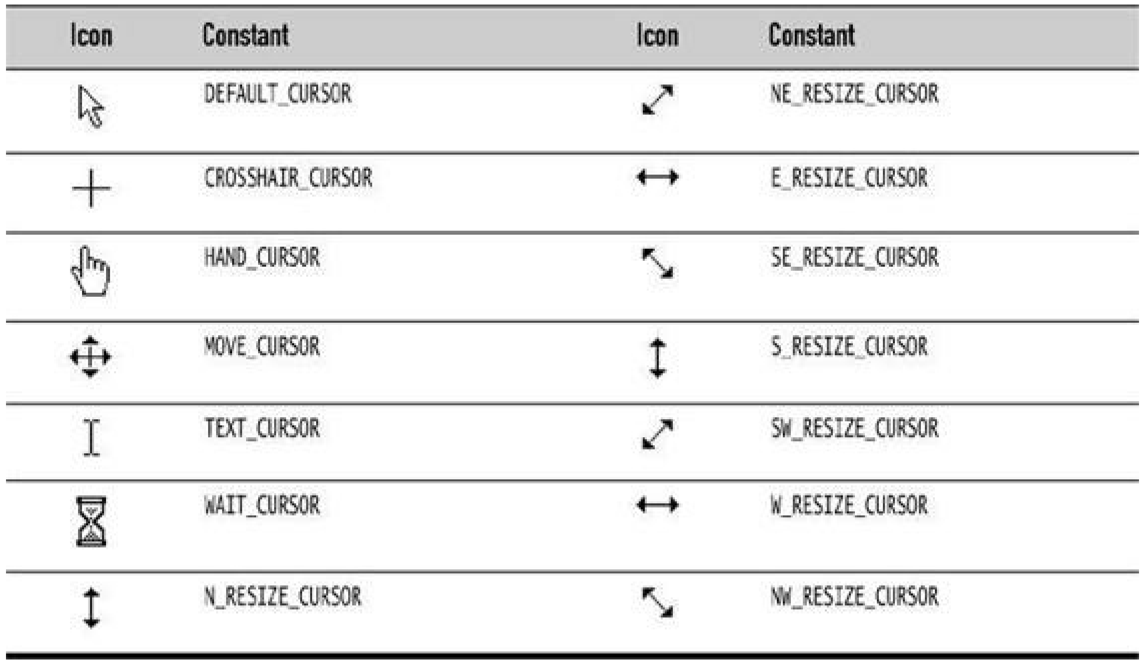
public void mouseClicked(MouseEvent e);

```
    /**
     * Invoked when a mouse button has been pressed on a component.
     */
    public void mousePressed(MouseEvent e);

    /**
     * Invoked when a mouse button has been released on a
component.
     */
    public void mouseReleased(MouseEvent e);

    /**
     * Invoked when the mouse enters a component.
     */
    public void mouseEntered(MouseEvent e);

    /**
     * Invoked when the mouse exits a component.
     */
    public void mouseExited(MouseEvent e);
```

Mouse Events

MouseMotionListener

public void mouseDragged(MouseEvent e);
public void mouseMoved(MouseEvent e);



Icon	Constant	Icon	Constant
↖	DEFAULT_CURSOR	⤢	NE_RESIZE_CURSOR
+	CROSSHAIR_CURSOR	↔	E_RESIZE_CURSOR
☝	HAND_CURSOR	⤡	SE_RESIZE_CURSOR
✥	MOVE_CURSOR	↕	S_RESIZE_CURSOR
I	TEXT_CURSOR	⤢	SW_RESIZE_CURSOR
⌛	WAIT_CURSOR	↔	W_RESIZE_CURSOR
↕	N_RESIZE_CURSOR	⤡	NW_RESIZE_CURSOR

component.setCursor(Cursor.getPredefinedCursor(Cursor
.CROSSHAIR_CURSOR));

References

- https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html
- http://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html
- http://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html