# Graphical User Interface Programming

## Part 3 – Event Handling – 2

Chapter 10, Core Java, Volume I

# Contents

# Actions

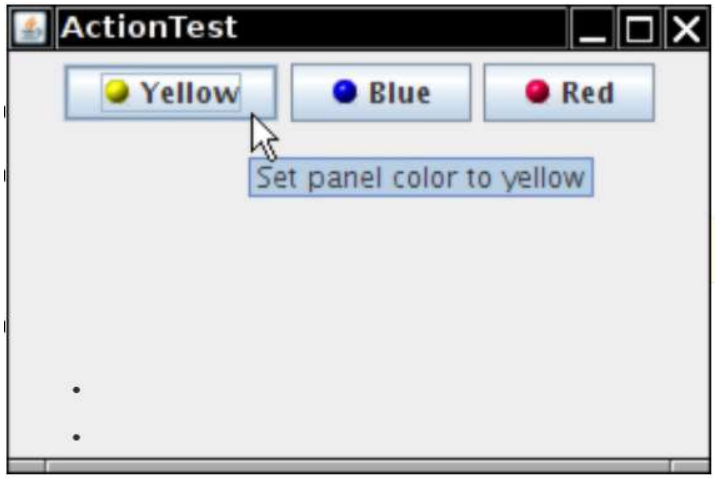
- Programs commonly provide multiple ways for selecting an action: menu, toolbar button, or keystroke.
- You can attach the same listener to several event sources.
- The Swing provides a very useful mechanism to encapsulate commands and to attach them to multiple event sources: `Action` interface
- `Action` interface combines listener code and action properties.
  - a description of the command (as a text string, an optinal icon, etc)
  - parameters that are necessary to carry out the command(e.g. background color)

- `Action` interface has the following methods:

```
void actionPerformed(ActionEvent event)
void setEnabled(false)
boolean isEnabled()
void puValue(String key, Object value)
Object getValue(String key)
```

# Actions

- Properties with keys:
  NAME
  SMALL_ICON
  SHORT_DESCRIPTION (for tooltips)
  MNEMONIC_KEY (for underlined characters in menus)




- Setting/getting properties

```
yellowAction.putValue(Action.SHORT_DESCRIPTION, "Set panel color to yellow");
yellowAction.putValue("color", Color.YELLOW);
Color c = yellowAction.getValue("color");
```

# Keyboard Commands

- Produce keystroke object:

```
KeyStroke ctrlYKey = KeyStroke.getKeyStroke("ctrl Y");
```

- Normally, keystrokes are sent to the component that has focus.
  - Tab key to move the focus.
  - When you press the space bar, the button with focus is clicked.

- To override, use input map that maps keystrokes to action map keys:

```
InputMap imap =
      panel.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
imap.put(ctrlYKey, "panel.yellow");
```

- Action map maps the action map key to the action:

```
ActionMap amap = panel.getActionMap();
amap.put("panel.yellow", yellowAction);
```

# Example: Action

```
package action;

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

 // A frame with a panel that demonstrates color change actions.
public class ActionFrame extends JFrame
{
   private JPanel buttonPanel;
   private static final int DEFAULT_WIDTH = 300;
   private static final int DEFAULT_HEIGHT = 200;
```

# Example: Action

```
public ActionFrame()
{
   setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
   buttonPanel = new JPanel();
   // define actions
   Action yellowAction = new ColorAction("Yellow", new ImageIcon("yellow-ball.gif"),
         Color.YELLOW);
   Action blueAction = new ColorAction("Blue", new ImageIcon("blue-ball.gif"), Color.BLUE);
   Action redAction = new ColorAction("Red", new ImageIcon("red-ball.gif"), Color.RED);

   // add buttons for these actions
   buttonPanel.add( new JButton(yellowAction) );
   buttonPanel.add( new JButton(blueAction) );
   buttonPanel.add( new JButton(redAction) );
```

# Example: Action

```
    // add panel to frame
    add(buttonPanel);

    // associate the Y, B, and R keys with names
    InputMap imap =
          buttonPanel.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
    imap.put(KeyStroke.getKeyStroke("ctrl Y"), "panel.yellow");
    imap.put(KeyStroke.getKeyStroke("ctrl B"), "panel.blue");
    imap.put(KeyStroke.getKeyStroke("ctrl R"), "panel.red");

    // associate the names with actions
    ActionMap amap = buttonPanel.getActionMap();
    amap.put("panel.yellow", yellowAction);
    amap.put("panel.blue", blueAction);
    amap.put("panel.red", redAction);
}
```

# Example: Action

```
public class ColorAction extends AbstractAction
{
   public ColorAction(String name, Icon icon, Color c)
   {
      putValue(Action.NAME, name);
      putValue(Action.SMALL_ICON, icon);
      putValue(Action.SHORT_DESCRIPTION, "Set panel color to " + name.toLowerCase());
      putValue("color", c);
   }

   public void actionPerformed(ActionEvent event)
   {
      Color c = (Color) getValue("color");
      buttonPanel.setBackground(c);
   }
} // end of inner class ColorAction
}
```

# Mouse Events

- MouseListener reports on mouse events in a component.
- When the mouse button is clicked, three listener methods are called:

```
void mousePressed(MouseEvent event)
void mouseReleased(MouseEvent event)
void mouseClicked(MouseEvent event)
```

- MouseAdapter implements all methods of MouseListener to do nothing.
- MouseEvent object has methods describing the event:

```
int getX()
int getY()
Point getPoint()
int getClickCount()
```

- MouseMotionListener tracks movement with mouseMoved and mouseDragged methods.

# Mouse Events

- To test for the state of all modal keys, such as ALT, CTRL, META, and the mouse buttons just after the event occurred.
  - use **getModifiersEx()** method
  - use masks

    BUTTON1_DOWN_MASK  BUTTON2_DOWN_MASK
    BUTTON3_DOWN_MASK  SHIFT_DOWN_MASK
    CTRL_DOWN_MASK  ALT_DOWN_MASK

- To test for right mouse click, use:

```
if ((event.getModifiersEx() & InputEvent.BUTTON3_DOWN_MASK) != 0)
```

- To check that SHIFT and BUTTON1(left mouse button) are down:

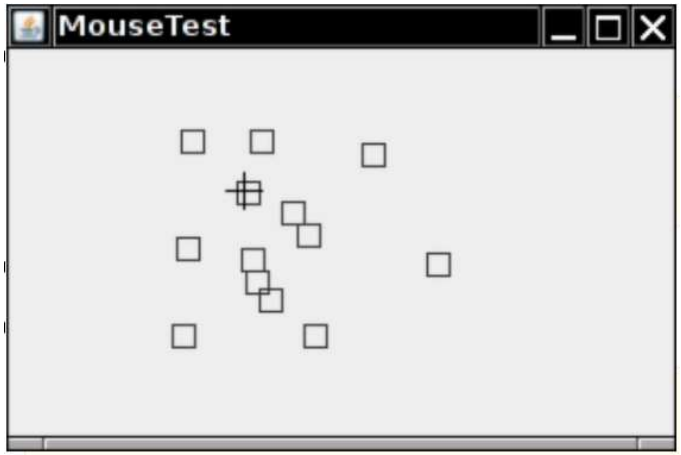
```
int onmask = SHIFT_DOWN_MASK | BUTTON1_DOWN_MASK;
if ((event.getModifiersEx() & onmask ) == onmask) { ... } // true even if CTRL key is pressed
```

# Example: Mouse Events (Listing 10.6)

```
package mouse;
import javax.swing.*;

public class MouseFrame extends JFrame
{
   public MouseFrame()
   {
      add(new MouseComponent());
      pack();
   }
}
```

- Mouse click outside any squares adds a new square.
- Double clikcs inside an existing square erase it.
- Drag the mouse with a square moves it.
- Moving the mouse over a sequre changes its cursor.



# Example: Mouse Events (Listing 10.6)

```
package mouse;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.*;
import javax.swing.*;

 // A component with mouse operations for adding and removing squares.
public class MouseComponent extends JComponent
{
   private static final int DEFAULT_WIDTH = 300;
   private static final int DEFAULT_HEIGHT = 200;
   private static final int SIDELENGTH = 10;
   private ArrayList<Rectangle2D> squares;
   private Rectangle2D current; // the square containing the mouse cursor
```

# Example: Mouse Events (Listing 10.6)

```
public MouseComponent()
{
   squares = new ArrayList<>();
   current = null;
   addMouseListener(new MouseHandler());
   addMouseMotionListener(new MouseMotionHandler());
}

public Dimension getPreferredSize() { return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT); }

public void paintComponent(Graphics g)
{
   Graphics2D g2 = (Graphics2D) g;
   // draw all squares
   for (Rectangle2D r : squares)
      g2.draw(r);
}
```

# Example: Mouse Events (Listing 10.6)

```
// Finds the first square containing a point.
public Rectangle2D find(Point2D p)
{
   for (Rectangle2D r : squares)
   {
      if (r.contains(p)) return r;
   }
   return null;
}
```

```
// Adds a square to the collection.
public void add(Point2D p)
{
   double x = p.getX();
   double y = p.getY();

   current = new Rectangle2D.Double
      (x - SIDELENGTH / 2, y - SIDELENGTH / 2,
       SIDELENGTH, SIDELENGTH);
   squares.add(current);
   repaint();
}
```

# Example: Mouse Events (Listing 10.6)

```
 // Removes a square from the collection.
public void remove(Rectangle2D s)
{
   if (s == null) return;
   if (s == current) current = null;
   squares.remove(s);
   repaint();
}

private class MouseHandler extends MouseAdapter  // MouseAdapter implements MouseListener
{
   public void mousePressed(MouseEvent event)
   {
      // add a new square if the cursor isn't inside a square
      current = find(event.getPoint());
      if (current == null) add(event.getPoint());
   }
```

# Example: Mouse Events (Listing 10.6)

```
    public void mouseClicked(MouseEvent event)
    {
       // remove the current square if double clicked
       current = find(event.getPoint());
       if (current != null && event.getClickCount() >= 2) remove(current);
    }
}

private class MouseMotionHandler implements MouseMotionListener
{
    public void mouseMoved(MouseEvent event)
    {
       // set the mouse cursor to cross hairs if it is inside a rectangle

       if (find(event.getPoint()) == null) setCursor(Cursor.getDefaultCursor());
       else setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
    }
```

# Example: Mouse Events (Listing 10.6)

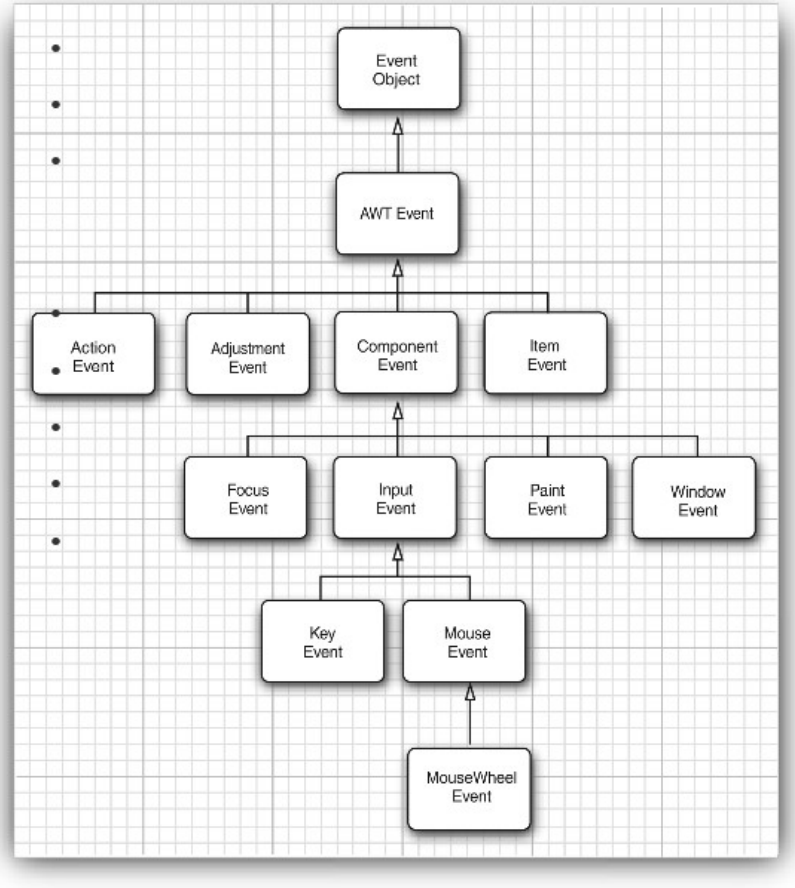
```
    public void mouseDragged(MouseEvent event)
    {
       if (current != null)
       {
          int x = event.getX();
          int y = event.getY();

          // drag the current rectangle to center it at (x, y)
          current.setFrame(x - SIDELENGTH / 2, y - SIDELENGTH / 2, SIDELENGTH, SIDELENGTH);
          repaint();
       }
    }
  }
}
```

# AWT Event Classes

- Semantic events are meaningful to application programmers:
  - ActionEvent (button, textfield, checkbox, menu, ...)
  - AdjustmentEvent (scrollbar)
  - ItemEvent (list box)

- Low-level events come from input sources:
  - KeyEvent
  - MouseEvent
  - MouseWheelEvent
  - FocusEvent
  - WindowEvent



# AWT Listener Classes

- **ActionListener** (button, menu item, checkbox, combo box, text field, timer)
- **AdjustmentListener** (scroll bar)
- **ItemListener** (combo box)
- **FocusListener** (component)
- **KeyListener** (component)
- **MouseListener**, **MouseMotionListener**, **MouseWheelListener** (component)
- **WindowListener**, **WindowFocusListener**, **WindowStateListener** (window)
- …