THỰC HÀNH CÔNG CỤ VÀ MÔI TRƯỜNG VÀ LẬP TRÌNH 2

TS. Võ Phương Bình – Email: binhvp@dlu.edu.vn
Information Technology Faculty - Dalat University
Website: http://it.dlu.edu.vn/ivp-lab

LAB 5 (4 tiết): Sử dụng sự kiện chuột và bàn phím

A. Mục tiêu:

- Hiểu biết và sử dụng được các sự kiện chuột và bàn phím
- Xử lý các sự kiện trên môi trường GUI

B. Kết quả sau khi hoàn thành:

- Sử dụng được các thành phần thiết kế để có thể tạo ra các giao diện.
- Xây dựng các ứng dụng có sự kiện chuột và bàn phím.
- Nhúng vào website tự thiết kế đơn giản.

C. Luyện tập:

Quy tắc sử dụng sự kiện:

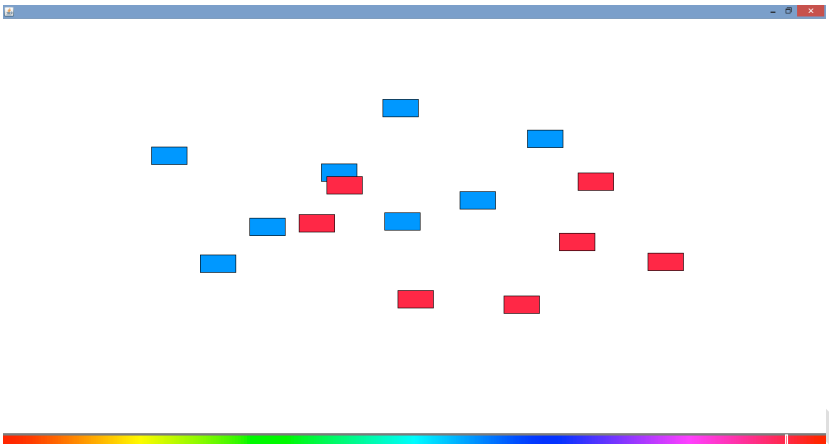
- Khai báo kế thừa interface phù hợp với sự kiện cần xử lý.
- Phải định nghĩa lại tất cả các phương thức của interface được kế thừa.
- Ví dụ:

1. Cần xử lý các sự kiện ActionEvent như click trên Button, Menu thì kế thừa interface ActionListener. Ta phải khai báo phương thức *actionPerformed(ActionEvent e)*.
2. Cần xử lý về sự kiện chuột thì kế thừa interface MouseListener. Ta phải định nghĩa các phương thức sau:

```
public void mousePressed(MouseEvent evt);
public void mouseReleased(MouseEvent evt);
public void mouseClicked(MouseEvent evt);
public void mouseEntered(MouseEvent evt);
public void mouseExited(MouseEvent evt);
```

D. Bài tập.

Hãy viết chương trình xử lý sự kiện chọn màu và nhấp phải chuột vẽ hình chữ nhật như minh họa sau:



Hướng dẫn:

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;


class RainbowPalette extends JPanel implements MouseListener {


  /* The currently selected color is stored in the variable
     selectedColor.  The hue of this color -- a float value
     in the range 0.0F to 1.0F is stored in selectedHue.
     The value of selectedHue is used to determine where
     to draw the hilite on the palette */


  private float selectedHue = 0;

  private Color selectedColor = Color.getHSBColor(0,1,1);
```

```
RainbowPalette() {
      // Constructor.  Set the component to listen for mouse clicks
      // on itself, and set the preferred size.  The gray background
      // color will show around the edges of the colored palette.
   addMouseListener(this);
   setPreferredSize( new Dimension(256, 24) );
   setBackground(Color.gray);
}

public Color getSelectedColor() {
      // Return the color that is currently selected in the palette.
   return selectedColor;
}

public void paintComponent(Graphics g) {
       // Draw the palette, and add a white rectangle to hilite
       // the selected color.
   super.paintComponent(g);
   int width = getWidth();
   int height = getHeight();
   for (int i = 0; i < width - 8; i++) {
      float hue = (float)i / (width-8);
      g.setColor( Color.getHSBColor(hue, 1, 1) );
```

```
      g.drawLine(i+4,4,i+4,height-5);
    }
    int x = 4 + (int)(selectedHue*(width-8));  // x-coord of selected color.
    g.setColor(Color.white);
    g.drawRect(x-2,3,2,height-7);  // Draw the hilite.
    g.drawRect(x-3,2,4,height-5);
}

public void mousePressed(MouseEvent evt) {
        // When the user clicks on the component, select the
        // color that the user clicked.  But make sure that
        // the selectedHue is in the legal range, 0 to 1.
    int x = evt.getX();
    selectedHue = (float)x / (getSize().width - 4);
    if (selectedHue < 0)
      selectedHue = 0;
    else if (selectedHue > 1)
      selectedHue = 1;
    selectedColor = Color.getHSBColor(selectedHue, 1, 1);
    repaint();
}

public void mouseReleased(MouseEvent evt) { }
```

```
   public void mouseClicked(MouseEvent evt) { }
   public void mouseEntered(MouseEvent evt) { }
   public void mouseExited(MouseEvent evt) { }

}  // end class RainbowPalette



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



public class SimpleDrawRects extends JFrame {

   RainbowPalette colorInput;  // A palette of colors that appears
                               // at the bottom of the applet.  The
                               // RainbowPalette class is non-standard,
                               // and is defined externally to this file.

   public SimpleDrawRects() {
         // Set up the applet with a drawing surface and palette.
      setBackground(Color.black); // shows along border
```

```
   Rects canvas = new Rects();
   colorInput = new RainbowPalette();
   getContentPane().add(canvas, BorderLayout.CENTER);
   getContentPane().add(colorInput, BorderLayout.SOUTH);
}

public Insets getInsets() {
      // Leave space for a black border around the applet.
   return new Insets(2,2,2,2);
}

//------------------------- Nested classes ------------------


static class ColoredRect {
      // Represents the data for one colored rectangle.
   int x, y, width, height;   // Location and size of rect.
   Color color;               // Color of rect.
}


class Rects extends JPanel
                    implements MouseListener, MouseMotionListener {
```

```
    // This class is a canvas that shows some colored rectangles.
    // The user adds a rectangle by right-clicking on the canvas.
    // The user can delete a rectangle by Alt-clicking it, and can
    // move it out in front of the other rectangles by Shift-clicking
    // it.  The user can also click-and-drag rectangles to move them
    // around the canvas.

private ArrayList rects;   // The colored rectangles are represented by objects
                           // of type ColoredRect that are stored in this ArrayList.


/* Variables for implementing dragging. */

private boolean dragging;      // This is true when dragging is in progress.
private ColoredRect dragRect;  // The rect that is being dragged (if dragging is true).
private int offsetx, offsety;  // The distance from the upper left corner of the
                               //   dragRect to the point where the user clicked
                               //   the rect.  This offset is maintained as the
                               //   rect is dragged.


Rects() {
```

```
    // Constructor.  The canvas listens for mouse events, and an
    // ArrayList is created to hold the ColoredRects.
  setBackground(Color.white);
  addMouseListener(this);
  addMouseMotionListener(this);
  rects = new ArrayList();
}

ColoredRect findRect(int x, int y) {
    // Find the topmost rect that contains the point (x,y).
    // Return null if no rect contains that point.
    // The rects in the ArrayList are considered in reverse order
    // so that if one lies on top of another, the one on top
    // is seen first and is returned.
  for (int i = rects.size() - 1;  i >= 0;  i--) {
    ColoredRect rect = (ColoredRect)rects.get(i);
    if ( x >= rect.x && x < rect.x + rect.width
            && y >= rect.y && y < rect.y + rect.height )
        return rect;  // (x,y) is inside this rect.
  }
  return null;
}
```

```
void bringToFront(ColoredRect rect) {
        // If rect != null, move it out in front of the other
        // rects by moving it to the last position in the ArrayList.
   if (rect != null) {
      rects.remove(rect);  // Remove rect from current position.
      rects.add(rect);     // Put rect in the ArrayList in last position.
      repaint();
   }
}

void deleteRect(ColoredRect rect) {
        // If rect != null, remove it from the ArrayList and from the screen.
   if (rect != null) {
      rects.remove(rect);
      repaint();
   }
}

public void paintComponent(Graphics g) {
      // Draw all the rects in the ArrayList.
   super.paintComponent(g);  // Fills with background color, white.
   for (int i = 0; i < rects.size(); i++) {
      ColoredRect rect = (ColoredRect)rects.get(i);
```

```
      g.setColor(rect.color);
      g.fillRect(rect.x, rect.y, rect.width, rect.height);
      g.setColor(Color.black);
      g.drawRect(rect.x, rect.y, rect.width - 1, rect.height - 1);
    }
}

public void mousePressed(MouseEvent evt) {
        // The user clicked on the canvas.  This can have several effects...

    if (dragging)  // If dragging is already in progress, just return.
      return;

    if (evt.isMetaDown()) {
          // User right-clicked or command clicked.  Make a new
          // rectangle and add it to the canvas.  Every rectangle is
          // 60 pixels wide and 30 pixels tall.  The point where the
          // user clicked is at the center of the rectangle.  It's
          // color is the selected color in the colorInput palette.
       ColoredRect rect = new ColoredRect();
       rect.x = evt.getX() - 30;
       rect.y = evt.getY() - 15;
       rect.width = 60;
```

```
    rect.height = 30;
    rect.color = colorInput.getSelectedColor();
    rects.add(rect);
    repaint();
}
else if (evt.isShiftDown()) {
        // User shift-clicked.  More the rect that the user
        // clicked (if any) to the front.  Note that findRect()
        // might return null, but bringToFront() accounts for that.
   bringToFront( findRect( evt.getX(), evt.getY() ) );
}
else if (evt.isAltDown()) {
        // User alt-clicked or middle-clicked.  Delete the rect
        // that the user clicked.
   deleteRect( findRect( evt.getX(), evt.getY() ) );
}
else {
         // This is a simple left-click.  Start dragging the
         // rect that the user clicked (if any).
   dragRect = findRect( evt.getX(), evt.getY() );
   if (dragRect != null) {
      dragging = true;   // Begin a drag operation.
      offsetx = evt.getX() - dragRect.x;
```

```
        offsety = evt.getY() - dragRect.y;
      }


    }

} // end mousePressed()


public void mouseReleased(MouseEvent evt) {
        // End the drag operation, if one is in progress.
    if (dragging == false)
      return;
    dragRect = null;
    dragging = false;
}


public void mouseDragged(MouseEvent evt) {
        // Continue the drag operation if one is in progress.
        // Move the rect that is being dragged to the current
        // mouse position.  But clamp it so that it can't
        // be more than halfway off the screen.

    if (dragging == false)
      return;
```

```
    dragRect.x = evt.getX() - offsetx;  // Get new postion of rect.
    dragRect.y = evt.getY() - offsety;

    /* Clamp (x,y) to a permitted range, as described above. */

    if (dragRect.x < - dragRect.width / 2)
       dragRect.x = - dragRect.width / 2;
    else if (dragRect.x + dragRect.width/2 > getSize().width)
       dragRect.x = getSize().width - dragRect.width / 2;
    if (dragRect.y < - dragRect.height / 2)
       dragRect.y = - dragRect.height / 2;
    else if (dragRect.y + dragRect.height/2 > getSize().height)
       dragRect.y = getSize().height - dragRect.height / 2;

    /* Redraw the canvas, with the rect in its new position. */

    repaint();

  }  // end mouseDragged()

  public void mouseClicked(MouseEvent evt) { }
  public void mouseEntered(MouseEvent evt) { }
```

```
    public void mouseExited(MouseEvent evt) { }

    public void mouseMoved(MouseEvent evt) { }

  }  // end nested class Rects

}  // end class SimpleDrawRects
```

E. Kết quả thực hành.

- Sinh viên thực hành ứng dụng trên GUI.
- Thời gian thực hành: 4 tiết.

F. Đánh giá:

- Kiểm tra lại chương trình, thử các kết quả.
- Bắt các lỗi bằng cách sử dụng các phần bắt lỗi: try – catch.

G.Phụ lục(file đi kèm).

----------Hết----------