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 5.2 (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:

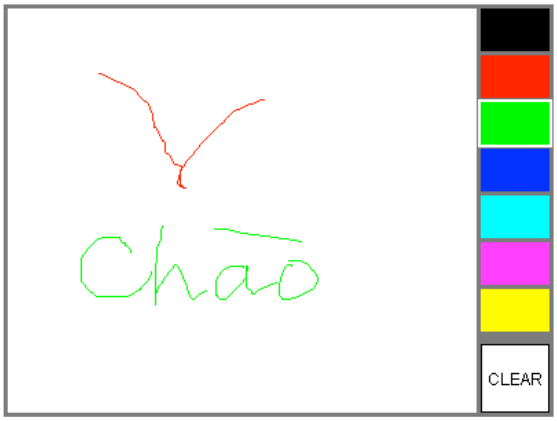
  public void mousePressed(MouseEvent evt);
  public void mouseReleased(MouseEvent evt);
  public void mouseClicked(MouseEvent evt);
  public void mouseEntered(MouseEvent evt);
  public void mouseExited(MouseEvent evt);

D. Bài tập.

Bài 1:

Viết chương trình paint đơn giản, cho phép dùng chuột vẽ như minh họa sau:

Minh họa:



Hướng dẫn:

```
/*
   A simple program where the user can sketch curves in a variety of
   colors.  A color palette is shown on the right of the applet.
   The user can select a drawing color by clicking on a color in the
   palette.  Under the colors is a "Clear button" that the user
   can press to clear the sketch.  The user draws by clicking and
   dragging in a large white area that occupies most of the applet.
   The user's drawing is not persistant.  It is cleared if the
   applet is resized.  If it is covered, in whole or part, and
   then uncovered, the part was covered is gone.
*/


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

```
import java.applet.*;
import javax.swing.JFrame;

public class SimplePaint extends JFrame
        implements MouseListener, MouseMotionListener {

  private final static int
          BLACK = 0,
          RED = 1,           // Some constants to make
          GREEN = 2,         // the code more readable.
          BLUE = 3,          // These numbers code for
          CYAN = 4,          // the differnet drawing colors.
          MAGENTA = 5,
          YELLOW = 6;

  private int currentColor = BLACK;  // The currently selected drawing color,
                        //   coded as one of the above constants.

  /* The following variables are used when the user is sketching a
     curve while dragging a mouse. */

  private int prevX, prevY;     // The previous location of the mouse.
```

```
private boolean dragging;      // This is set to true while the user is drawing.

private Graphics graphicsForDrawing;  // A graphics context for the applet
                                      // that is used to draw the user's curve.



public SimplePaint() {
      // When the applet is first created, the applet is set to
      // listen for mouse events and mouse motion events from
      // itself.
   addMouseListener(this);
   addMouseMotionListener(this);
}



public void update(Graphics g) {
      // Redefine update so it does not fill the applet with the
      // background color before calling paint().  This is OK
      // since the paint() method always draws over the entire
      // surface of the applet.
    paint(g);
}
```

```
public void paint(Graphics g) {
      // Draw the contents of the applet.  Since no information is
      // saved about what the user has drawn, the user's drawing
      // is erased whenever this routine is called.

   int width = getSize().width;    // Width of the applet.
   int height = getSize().height;  // Height of the applet.

   int colorSpacing = (height - 56) / 7;
        // Distance between the top of one colored rectangle in the palette
        // and the top of the rectangle below it.  The height of the
        // rectangle will be colorSpacing - 3.  There are 7 colored rectangles,
        // so the available space is divided by 7.  The available space allows
        // for the gray border and the 50-by-50 CLEAR button.

   /*  First, fill in the white drawing area, allowing for
       a three-pixel border at top, bottom, and left and allowing
       for the 56-pixel wide strip on the right that is occupied by
       the color palette and CLEAR button.  (I could just fill the
       whole applet with white and then draw over some of it, but
       that leads to increased flickering when the applet is redrawn.)
```

```
*/

g.setColor(Color.white);
g.fillRect(3, 3, width - 59, height - 6);

/* Draw a 3-pixel border around the applet in gray.  This has to be
   done by drawing three rectangles of different sizes. */

g.setColor(Color.gray);
g.drawRect(0, 0, width-1, height-1);
g.drawRect(1, 1, width-3, height-3);
g.drawRect(2, 2, width-5, height-5);

/* Draw a 56-pixel wide gray rectangle along the right edge of the applet.
   The color palette and Clear button will be drawn on top of this.
   (This covers some of the same area as the border I just drew. */

g.fillRect(width - 56, 0, 56, height);

/* Draw the "Clear button" as a 50-by-50 white rectangle in the lower right
   corner of the applet, allowing for a 3-pixel border. */

g.setColor(Color.white);
```

```
g.fillRect(width-53,  height-53, 50, 50);
g.setColor(Color.black);
g.drawRect(width-53, height-53, 49, 49);
g.drawString("CLEAR", width-48, height-23);

/* Draw the seven color rectangles. */

g.setColor(Color.black);
g.fillRect(width-53, 3 + 0*colorSpacing, 50, colorSpacing-3);
g.setColor(Color.red);
g.fillRect(width-53, 3 + 1*colorSpacing, 50, colorSpacing-3);
g.setColor(Color.green);
g.fillRect(width-53, 3 + 2*colorSpacing, 50, colorSpacing-3);
g.setColor(Color.blue);
g.fillRect(width-53, 3 + 3*colorSpacing, 50, colorSpacing-3);
g.setColor(Color.cyan);
g.fillRect(width-53, 3 + 4*colorSpacing, 50, colorSpacing-3);
g.setColor(Color.magenta);
g.fillRect(width-53, 3 + 5*colorSpacing, 50, colorSpacing-3);
g.setColor(Color.yellow);
g.fillRect(width-53, 3 + 6*colorSpacing, 50, colorSpacing-3);

/* Draw a 2-pixel white border around the color rectangle
```

```
      of the current drawing color. */

   g.setColor(Color.white);
   g.drawRect(width-55, 1 + currentColor*colorSpacing, 53, colorSpacing);
   g.drawRect(width-54, 2 + currentColor*colorSpacing, 51, colorSpacing-2);

} // end paint()



private void changeColor(int y) {
      // Change the drawing color after the user has clicked the
      // mouse on the color palette at a point with y-coordinate y.
      // (Note that I can't just call repaint and redraw the whole
      // applet, since that whould erase the user's drawing.)

   int width = getSize().width;          // Width of applet.
   int height = getSize().height;        // Height of applet.
   int colorSpacing = (height - 56) / 7;  // Space for one color rectangle.
   int newColor = y / colorSpacing;      // Which color number was clicked?

   if (newColor < 0 || newColor > 6)     // Make sure the color number is valid.
      return;
```

```
   /* Remove the hilite from the current color, by drawing over it in gray.
      Then change the current drawing color and draw a hilite around the
      new drawing color.  */

   Graphics g = getGraphics();
   g.setColor(Color.gray);
   g.drawRect(width-55, 1 + currentColor*colorSpacing, 53, colorSpacing);
   g.drawRect(width-54, 2 + currentColor*colorSpacing, 51, colorSpacing-2);
   currentColor = newColor;
   g.setColor(Color.white);
   g.drawRect(width-55, 1 + currentColor*colorSpacing, 53, colorSpacing);
   g.drawRect(width-54, 2 + currentColor*colorSpacing, 51, colorSpacing-2);
   g.dispose();

} // end changeColor()


private void setUpDrawingGraphics() {
      // This routine is called in mousePressed when the
      // user clicks on the drawing area.  It sets up the
      // graphics context, graphicsForDrawing, to be used
      // to draw the user's sketch in the current color.
```

```
graphicsForDrawing = getGraphics();
switch (currentColor) {
  case BLACK:
    graphicsForDrawing.setColor(Color.black);
    break;
  case RED:
    graphicsForDrawing.setColor(Color.red);
    break;
  case GREEN:
    graphicsForDrawing.setColor(Color.green);
    break;
  case BLUE:
    graphicsForDrawing.setColor(Color.blue);
    break;
  case CYAN:
    graphicsForDrawing.setColor(Color.cyan);
    break;
  case MAGENTA:
    graphicsForDrawing.setColor(Color.magenta);
    break;
  case YELLOW:
    graphicsForDrawing.setColor(Color.yellow);
    break;
```

```
   }
} // end setUpDrawingGraphics()



public void mousePressed(MouseEvent evt) {
      // This is called when the user presses the mouse anywhere
      // in the applet.  There are three possible responses,
      // depending on where the user clicked:  Change the
      // current color, clear the drawing, or start drawing
      // a curve.  (Or do nothing if user clicks on the border.)

   int x = evt.getX();   // x-coordinate where the user clicked.
   int y = evt.getY();   // y-coordinate where the user clicked.

   int width = getSize().width;    // Width of the applet.
   int height = getSize().height;  // Height of the applet.

   if (dragging == true)  // Ignore mouse presses that occur
      return;             //    when user is already drawing a curve.
                          //    (This can happen if the user presses
                          //    two mouse buttons at the same time.)

   if (x > width - 53) {
```

```
            // User clicked to the right of the drawing area.
            // This click is either on the clear button or
            // on the color palette.
        if (y > height - 53)
          repaint();       //  Clicked on "CLEAR button".
        else
          changeColor(y);  // Clicked on the color palette.
      }
      else if (x > 3 && x < width - 56 && y > 3 && y < height - 3) {
            // The user has clicked on the white drawing area.
            // Start drawing a curve from the point (x,y).
        prevX = x;
        prevY = y;
        dragging = true;
        setUpDrawingGraphics();
      }

} // end mousePressed()


public void mouseReleased(MouseEvent evt) {
        // Called whenever the user releases the mouse button.
        // If the user was drawing a curve, the curve is done,
```

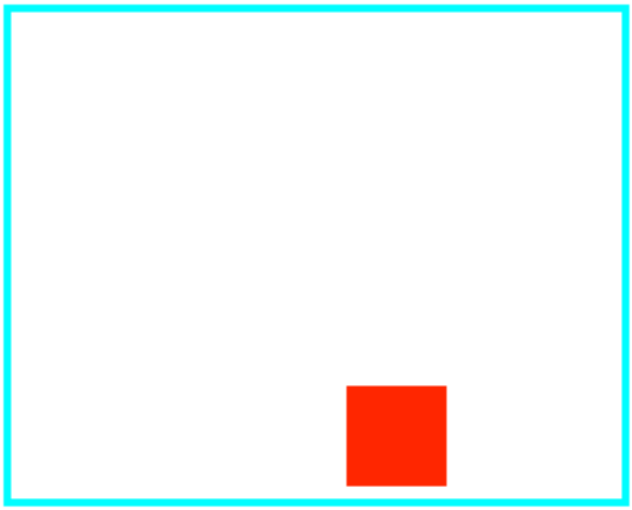
```
        // so we should set drawing to false and get rid of
        // the graphics context that we created to use during
        // the drawing.
    if (dragging == false)
       return;  // Nothing to do because the user isn't drawing.
    dragging = false;
    graphicsForDrawing.dispose();
    graphicsForDrawing = null;
}



public void mouseDragged(MouseEvent evt) {
        // Called whenever the user moves the mouse
        // while a mouse button is held down.  If the
        // user is drawing, draw a line segment from the
        // previous mouse location to the current mouse
        // location, and set up prevX and prevY for the
        // next call.  Note that in case the user drags
        // ouside of the drawing area, the values of
        // x and y are "clamped" to lie within this
        // area.  This avoids drawing on the color palette
        // or clear buton.
```

```
    if (dragging == false)
      return;  // Nothing to do because the user isn't drawing.

    int x = evt.getX();   // x-coordinate of mouse.
    int y = evt.getY();   // y=coordinate of mouse.

    if (x < 3)                    // Adjust the value of x,
      x = 3;                      //   to make sure it's in
    if (x > getSize().width - 57)       //   the drawing area.
      x = getSize().width - 57;

    if (y < 3)                    // Adjust the value of y,
      y = 3;                      //   to make sure it's in
    if (y > getSize().height - 4)       //   the drawing area.
      y = getSize().height - 4;

    graphicsForDrawing.drawLine(prevX, prevY, x, y);  // Draw the line.

    prevX = x;  // Get ready for the next line segment in the curve.
    prevY = y;

} // end mouseDragged.
```

```
   public void mouseEntered(MouseEvent evt) { }   // Some empty routines.
   public void mouseExited(MouseEvent evt) { }    //    (Required by the MouseListener
   public void mouseClicked(MouseEvent evt) { }   //    and MouseMotionListener
   public void mouseMoved(MouseEvent evt) { }     //    interfaces).
```

} // end class SimplePaint

Bài 2:
Viết chương trình xử lý sự kiện bàn phím như minh họa sau: sử dụng bàn phím điều hướng (mũi tên left, right, up, down) để di chuyển hình vuông màu đỏ.



Hướng dẫn:

```
/*
   A colored square
   is drawn on the JFrame.  By pressing the arrow keys, the user can move
   the square up, down, left, or right.  By pressing the keys
   R, G, B, or K, the user can change the color of the square to red,
   green, blue, or black, respectively.  Of course, none of the keys
   will have any effect if the applet does not have the keyboard input
   focus.  The applet changes appearance when it has the input focus.
   A cyan-colored border is drawn around it.  When it does not have
   the input focus, the message "Click to activate" is displayed
```

```
   and the border is gray.
*/


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


public class KeyboardAndFocusDemo extends JFrame
                  implements KeyListener, FocusListener, MouseListener {
                      // (Note:  MouseListener is implemented only so that
                      //         the applet can request the input focus when
                      //         the user clicks on it.)

   static final int SQUARE_SIZE = 40;  // Length of a side of the square.

   Color squareColor;  // The color of the square.

   int squareTop, squareLeft;  // Coordinates of top-left corner of square.

   boolean focussed = false;   // True when this applet has input focus.

   DisplayPanel canvas;  // The drawing surface on which the applet draws,
                         // belonging to a nested class DisplayPanel, which
                         // is defined below.

   public KeyboardAndFocusDemo () {
        // Initialize the applet; set it up to receive keyboard
        // and focus events.  Place the square in the middle of
        // the applet, and make the initial color of the square red.
        // Then, set up the drawing surface.

      squareTop = getSize().height / 2 - SQUARE_SIZE / 2;
      squareLeft = getSize().width / 2 - SQUARE_SIZE / 2;
      squareColor = Color.red;

      canvas = new DisplayPanel();  // Create drawing surface and
      setContentPane(canvas);       //    install it as the applet's content
pane.

      canvas.setBackground(Color.white);  // Set the background color of the
canvas.

      canvas.addFocusListener(this);   // Set up the applet to listen for
events
      canvas.addKeyListener(this);     //                            from the
canvas.
      canvas.addMouseListener(this);

   } // end init();


   class DisplayPanel extends JPanel {
          // An object belonging to this nested class is used as
          // the content pane of the applet.  It displays the
          // moving square on a white background with a border
          // that changes color depending on whether this
          // component has the input focus or not.
```

```
   public void paintComponent(Graphics g) {

      super.paintComponent(g);  // Fills the panel with its
                                // background color, which is white.

      /* Draw a 3-pixel border, colored cyan if the applet has the
         keyboard focus, or in light gray if it does not. */

      if (focussed)
         g.setColor(Color.cyan);
      else
         g.setColor(Color.lightGray);

      int width = getSize().width;  // Width of the applet.
      int height = getSize().height; // Height of the applet.
      g.drawRect(0,0,width-1,height-1);
      g.drawRect(1,1,width-3,height-3);
      g.drawRect(2,2,width-5,height-5);

      /* Draw the square. */

      g.setColor(squareColor);
      g.fillRect(squareLeft, squareTop, SQUARE_SIZE, SQUARE_SIZE);

      /* If the applet does not have input focus, print a message. */

      if (!focussed) {
         g.setColor(Color.magenta);
         g.drawString("Click to activate",7,20);
      }

   }  // end paintComponent()

 } // end nested class DisplayPanel



// ------------------- Event handling methods ----------------------

public void focusGained(FocusEvent evt) {
      // The applet now has the input focus.
   focussed = true;
   canvas.repaint();  // redraw with cyan border
}


public void focusLost(FocusEvent evt) {
      // The applet has now lost the input focus.
   focussed = false;
   canvas.repaint();  // redraw without cyan border
}


public void keyTyped(KeyEvent evt) {
       // The user has typed a character, while the
       // applet has the input focus.  If it is one
       // of the keys that represents a color, change
       // the color of the square and redraw the applet.
```

```
      char ch = evt.getKeyChar();  // The character typed.

      if (ch == 'B' || ch == 'b') {
         squareColor = Color.blue;
         canvas.repaint();
      }
      else if (ch == 'G' || ch == 'g') {
         squareColor = Color.green;
         canvas.repaint();
      }
      else if (ch == 'R' || ch == 'r') {
         squareColor = Color.red;
         canvas.repaint();
      }
      else if (ch == 'K' || ch == 'k') {
         squareColor = Color.black;
         canvas.repaint();
      }

   }  // end keyTyped()


   public void keyPressed(KeyEvent evt) {
          // Called when the user has pressed a key, which can be
          // a special key such as an arrow key.  If the key pressed
          // was one of the arrow keys, move the square (but make sure
          // that it doesn't move off the edge, allowing for a
          // 3-pixel border all around the applet).

      int key = evt.getKeyCode();  // keyboard code for the key that was
pressed

      if (key == KeyEvent.VK_LEFT) {
         squareLeft -= 8;
         if (squareLeft < 3)
            squareLeft = 3;
         canvas.repaint();
      }
      else if (key == KeyEvent.VK_RIGHT) {
         squareLeft += 8;
         if (squareLeft > getSize().width - 3 - SQUARE_SIZE)
            squareLeft = getSize().width - 3 - SQUARE_SIZE;
         canvas.repaint();
      }
      else if (key == KeyEvent.VK_UP) {
         squareTop -= 8;
         if (squareTop < 3)
            squareTop = 3;
         canvas.repaint();
      }
      else if (key == KeyEvent.VK_DOWN) {
         squareTop += 8;
         if (squareTop > getSize().height - 3 - SQUARE_SIZE)
            squareTop = getSize().height - 3 - SQUARE_SIZE;
         canvas.repaint();
      }

   }  // end keyPressed()
```

```
    public void keyReleased(KeyEvent evt) {
       // empty method, required by the KeyListener Interface
    }


    public void mousePressed(MouseEvent evt) {
         // Request that the input focus be given to the
         // canvas when the user clicks on the applet.
       canvas.requestFocus();
    }


    public void mouseEntered(MouseEvent evt) { }  // Required by the
    public void mouseExited(MouseEvent evt) { }   //    MouseListener
    public void mouseReleased(MouseEvent evt) { } //       interface.
    public void mouseClicked(MouseEvent evt) { }


} // end class KeyboardAndFocusDemo
```

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----------