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 6 (4 tiết): Áp dụng các sự kiện chuột, bàn phím và các hàm vẽ

A. Mục tiêu:

- Sử dụng các sự kiện chuột, bàn phím kết hợp các hàm vẽ tạo ra các hình ảnh trên môi trường Frame.
- Xử lý các sự kiện trên môi trường GUI.

B. Kết quả sau khi hoàn thành:

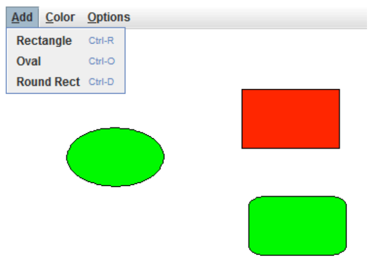
- Sử dụng được các thành phần thiết kế để có thể tạo ra các giao diện, vẽ các hình ảnh và tô màu cho các hình ảnh đó.
- Xây dựng các ứng dụng có sự kiện chuột và bàn phím.
- Nhúng vào website tự thiết kế đơn giản.

C. Yêu cầu:

Viết chương trình tạo hệ thống menu và xử lý menu như sau:

- Menu Add: vẽ thêm một đối tượng hình học (hình chữ nhật, hình chữ nhật móc và hình Oval).

Minh họa



D. Hướng dẫn

- Tạo hệ thống menu như yêu cầu:
  - Các MenuItem tùy chọn phải đặt trong ButtonGroup:
  Ví dụ:

```
ButtonGroup colorGroup = new ButtonGroup();
```

```
        red = new JRadioButtonMenuItem("Red");
        shapeColorMenu.add(red);   // Add to menu.
        colorGroup.add(red);       // Add to button group.

        green = new JRadioButtonMenuItem("Green");
        shapeColorMenu.add(green);
        colorGroup.add(green);

        blue = new JRadioButtonMenuItem("Blue");
        shapeColorMenu.add(blue);
        colorGroup.add(blue);
          .
          .
          .
```

- Dùng phương thức isSelected() để kiểm tra item được chọn (giá trị trả về là true nếu được chọn):
Ví dụ:

```
    if (red.isSelected())
       shape.setColor(Color.red);
     else if (green.isSelected())
       shape.setColor(Color.green);
     else if (blue.isSelected())
       shape.setColor(Color.blue);
     .
     .
     .
```

- Thực thi các interface ActionListener, MouseListener, MouseMotionListener để xử lý các sự kiện:
  - Xử lý sự kiện menu trong phương thức actionPerformed(ActionEvent evt).
  - Xử lý chuột trong các phương thức:

```
    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);
```

- Kế thừa lớp JPanel để vẽ các đối tượng hình học trong phương thức paintComponent(Graphics g).
- Dùng ArrayList để thêm và xóa các đối tượng hình học. Có thể dùng lớp ảo hay interface để mô tả các đối tượng hình học, với phương thức ảo (chung) là draw(Graphics g).

Hướng dẫn:

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

import javax.swing.*;

import java.util.ArrayList;



public class ShapeDrawFrame extends JFrame {


  public static void main(String[] args) {

       // The main routine just creates an object belonging

       // to this class, which will appear as a window on the

       // screen.  Although the main routines ends, the program

       // continues until the user closes the window.

    new ShapeDrawFrame();

  }


  // The following variables are declared as instance variables

  // since the objects are created in the constructor method but are

  // used in the ShapeCanvas class.


  JCheckBoxMenuItem addLargeShapes;  // Controls whether newly added

                                     // shapes will be large or small.


  JCheckBoxMenuItem addBorderedShapes;  // Controls whether newly added
```

```
                         // will have a black border.

JRadioButtonMenuItem red, green, blue,       // Control the color
                 cyan, magenta, yellow,  //    of newly added shapes.
                 black, gray, white;

JPopupMenu popup;  // The pop-up menu, which is used for editing shapes.



public ShapeDrawFrame() {
    // The constructor.  Set up the frmae's GUI, and show the frame.
    // The frame contains a canvas, or drawing area, and a menu bar.
    // The pop-up menu is also created in this constructor.

  super("Shape Draw");  // Call the constructor from the superclass,
                   // to specify a title to appear at the top
                   // of the window.

  /* Create the "canvas" which displays the shapes and serves as
     the content pane of the frame. */

  ShapeCanvas canvas = new ShapeCanvas();
  setContentPane(canvas);
```

```
/* Create the menu bar and the menus */

JMenuBar menubar = new JMenuBar();
setJMenuBar(menubar);

JMenu addShapeMenu = new JMenu("Add");
addShapeMenu.setMnemonic('A');
menubar.add(addShapeMenu);
JMenu shapeColorMenu = new JMenu("Color");
shapeColorMenu.setMnemonic('C');
menubar.add(shapeColorMenu);
JMenu optionsMenu = new JMenu("Options");
optionsMenu.setMnemonic('O');
menubar.add(optionsMenu);


/* Create menu items for adding shapes to the canvas,
   and add them to the "Add" menu.  The canvas serves
   as ActionListener for these menu items. */

JMenuItem rect = new JMenuItem("Rectangle");
```

```
rect.setAccelerator( KeyStroke.getKeyStroke("ctrl R") );

addShapeMenu.add(rect);

rect.addActionListener(canvas);

JMenuItem oval = new JMenuItem("Oval");

oval.setAccelerator( KeyStroke.getKeyStroke("ctrl O") );

addShapeMenu.add(oval);

oval.addActionListener(canvas);

JMenuItem roundrect = new JMenuItem("Round Rect");

roundrect.setAccelerator( KeyStroke.getKeyStroke("ctrl D") );

addShapeMenu.add(roundrect);

roundrect.addActionListener(canvas);



/* Create the JRadioButtonMenuItems that control the color

   of a newly added shape, and add them to the "Color"

   menu.  There is no ActionListener for these menu items.

   The canvas checks for the currently selected color when

   it adds a shape to the canvas.  A ButtonGroup is used

   to make sure that only one color is selected. */


ButtonGroup colorGroup = new ButtonGroup();

red = new JRadioButtonMenuItem("Red");

shapeColorMenu.add(red);
```

```
colorGroup.add(red);

red.setSelected(true);

green = new JRadioButtonMenuItem("Green");

shapeColorMenu.add(green);

colorGroup.add(green);

blue = new JRadioButtonMenuItem("Blue");

shapeColorMenu.add(blue);

colorGroup.add(blue);

cyan = new JRadioButtonMenuItem("Cyan");

shapeColorMenu.add(cyan);

colorGroup.add(cyan);

magenta = new JRadioButtonMenuItem("Magenta");

shapeColorMenu.add(magenta);

colorGroup.add(magenta);

yellow = new JRadioButtonMenuItem("Yellow");

shapeColorMenu.add(yellow);

colorGroup.add(yellow);

black = new JRadioButtonMenuItem("Black");

shapeColorMenu.add(black);

colorGroup.add(black);

gray = new JRadioButtonMenuItem("Gray");

shapeColorMenu.add(gray);

colorGroup.add(gray);
```

```
white = new JRadioButtonMenuItem("White");
shapeColorMenu.add(white);
colorGroup.add(white);



/* Create the "Clear" menu item, and add it to the
   "Options" menu.  The canvas will listen for events
   from this menu item. */

JMenuItem clear = new JMenuItem("Clear");
clear.setAccelerator( KeyStroke.getKeyStroke("ctrl C") );
clear.addActionListener(canvas);
optionsMenu.add(clear);
optionsMenu.addSeparator();  // Add a separating line to the menu.



/* Create the JCheckBoxMenuItems and add them to the Options
   menu.  There is no ActionListener for these items because
   the canvas class will check their state when it adds a
   new shape. */

addLargeShapes = new JCheckBoxMenuItem("Add Large Shapes");
addLargeShapes.setSelected(true);
```

```
optionsMenu.add(addLargeShapes);

addBorderedShapes = new JCheckBoxMenuItem("Add Shapes with Border");

addBorderedShapes.setSelected(true);

optionsMenu.add(addBorderedShapes);

optionsMenu.addSeparator();



/* Create a menu for background colors, and add it to the

   "Options" menu.  It will show up as a hierarchical sub-menu. */


JMenu background = new JMenu("Background Color");

optionsMenu.add(background);

background.add("Red").addActionListener(canvas);

background.add("Green").addActionListener(canvas);

background.add("Blue").addActionListener(canvas);

background.add("Cyan").addActionListener(canvas);

background.add("Magenta").addActionListener(canvas);

background.add("Yellow").addActionListener(canvas);

background.add("Black").addActionListener(canvas);

background.add("Gray").addActionListener(canvas);

background.add("White").addActionListener(canvas);
```

```
/* Create the pop-up menu and add commands for editing a
   shape.  This menu is not used until the user performs
   the pop-up trigger mouse gesture on a shape. */

popup = new JPopupMenu();
popup.add("Delete Shape").addActionListener(canvas);
popup.add("Bring to Front").addActionListener(canvas);
popup.addSeparator();
popup.add("Make Large").addActionListener(canvas);
popup.add("Make Small").addActionListener(canvas);
popup.addSeparator();
popup.add("Add Black Border").addActionListener(canvas);
popup.add("Remove Black Border").addActionListener(canvas);
popup.addSeparator();
popup.add("Set Color to Red").addActionListener(canvas);
popup.add("Set Color to Green").addActionListener(canvas);
popup.add("Set Color to Blue").addActionListener(canvas);
popup.add("Set Color to Cyan").addActionListener(canvas);
popup.add("Set Color to Magenta").addActionListener(canvas);
popup.add("Set Color to Yellow").addActionListener(canvas);
popup.add("Set Color to Black").addActionListener(canvas);
popup.add("Set Color to Gray").addActionListener(canvas);
popup.add("Set Color to White").addActionListener(canvas);
```

```
   /* Set the "DefaultCloseOperation" for the frame.  This determines
      what happens when the user clicks the close box of the frame.
      It is set here so that System.exit() will be called to end
      the program when the user closes the window. */

   setDefaultCloseOperation(EXIT_ON_CLOSE);

   /* Set the size and location of the frame, and make it visible. */

   setLocation(20,50);
   setSize(550,420);
   show();

} // end constructor

//---- Nested class definitions ---
//
// The remainder of the ShapeDraw class consists of static nested class definitions.
// These are just like regular classes, except that they are defined inside
// another class (and hence have full names, when used outside this class, such
// as ShapeDraw.ShapeCanvas).
```

```
class ShapeCanvas extends JPanel
            implements ActionListener, MouseListener, MouseMotionListener {

   // This class represents a "canvas" that can display colored shapes and
   // let the user drag them around.  It uses an off-screen images to
   // make the dragging look as smooth as possible.

  ArrayList shapes = new ArrayList();
     // holds a list of the shapes that are displayed on the canvas


  ShapeCanvas() {
     // Constructor: set background color to white
     // set up listeners to respond to mouse actions
   setBackground(Color.white);
   addMouseListener(this);
   addMouseMotionListener(this);
  }

  public void paintComponent(Graphics g) {
     // In the paint method, all the shapes in ArrayList are
     // copied onto the canvas.
   super.paintComponent(g);  // First, fill with background color.
   int top = shapes.size();
   for (int i = 0; i < top; i++) {
      Shape s = (Shape)shapes.get(i);
      s.draw(g);
   }
}

public void actionPerformed(ActionEvent evt) {
      // Called to respond to action events from the
      // menus or pop-up menu.
   String command = evt.getActionCommand();
   if (command.equals("Clear")) {
      shapes.clear(); // Remove all items from the ArrayList
      repaint();
   }
   else if (command.equals("Rectangle"))
      addShape(new RectShape());
   else if (command.equals("Oval"))
      addShape(new OvalShape());
   else if (command.equals("Round Rect"))
      addShape(new RoundRectShape());
   else if (command.equals("Red"))
```

```
  setBackground(Color.red);
else if (command.equals("Green"))
  setBackground(Color.green);
else if (command.equals("Blue"))
  setBackground(Color.blue);
else if (command.equals("Cyan"))
  setBackground(Color.cyan);
else if (command.equals("Magenta"))
  setBackground(Color.magenta);
else if (command.equals("Yellow"))
  setBackground(Color.yellow);
else if (command.equals("Black"))
  setBackground(Color.black);
else if (command.equals("Gray"))
  setBackground(Color.gray);
else if (command.equals("White"))
  setBackground(Color.white);
else if (clickedShape != null) {
     // Process a command from the pop-up menu.
  if (command.equals("Delete Shape"))
    shapes.remove(clickedShape);
  else if (command.equals("Bring to Front")) {
    shapes.remove(clickedShape);
    shapes.add(clickedShape);
  }
  else if (command.equals("Make Large"))
    clickedShape.setSize(100,60);
  else if (command.equals("Make Small"))
    clickedShape.setSize(50,30);
  else if (command.equals("Add Black Border"))
    clickedShape.setDrawOutline(true);
  else if (command.equals("Remove Black Border"))
    clickedShape.setDrawOutline(false);
  else if (command.equals("Set Color to Red"))
    clickedShape.setColor(Color.red);
  else if (command.equals("Set Color to Green"))
    clickedShape.setColor(Color.green);
  else if (command.equals("Set Color to Blue"))
    clickedShape.setColor(Color.blue);
  else if (command.equals("Set Color to Cyan"))
    clickedShape.setColor(Color.cyan);
  else if (command.equals("Set Color to Magenta"))
    clickedShape.setColor(Color.magenta);
  else if (command.equals("Set Color to Yellow"))
    clickedShape.setColor(Color.yellow);
  else if (command.equals("Set Color to Black"))
```

```
        clickedShape.setColor(Color.black);
      else if (command.equals("Set Color to Gray"))
        clickedShape.setColor(Color.gray);
      else if (command.equals("Set Color to White"))
        clickedShape.setColor(Color.white);
      repaint();
   }
} // end actionPerformed()



void addShape(Shape shape) {
      // Add the shape to the canvas, and set its size, color
      // and whether or not it should have a black border.  These
      // properties are determined by looking at the states of
      // various menu items.  The shape is added at the top-left
      // corner of the canvas.
   if (red.isSelected())
      shape.setColor(Color.red);
   else if (blue.isSelected())
      shape.setColor(Color.blue);
   else if (green.isSelected())
      shape.setColor(Color.green);
   else if (cyan.isSelected())
```

```
      shape.setColor(Color.cyan);
   else if (magenta.isSelected())
      shape.setColor(Color.magenta);
   else if (yellow.isSelected())
      shape.setColor(Color.yellow);
   else if (black.isSelected())
      shape.setColor(Color.black);
   else if (white.isSelected())
      shape.setColor(Color.white);
   else
      shape.setColor(Color.gray);
   shape.setDrawOutline( addBorderedShapes.isSelected() );
   if (addLargeShapes.isSelected())
      shape.reshape(3,3,100,60);
   else
      shape.reshape(3,3,50,30);
   shapes.add(shape);
   repaint();
} // end addShape()



// ------------------- This rest of this class implements dragging ---------------------
```

```
Shape clickedShape = null;  // This is the shape that the user clicks on.
                        // It becomes the draggedShape is the user is
                        // dragging, unless the user is invoking a
                        // pop-up menu.  This variable is used in
                        // actionPerformed() when a command from the
                        // pop-up menu is processed.

Shape draggedShape = null;  // This is null unless a shape is being dragged.
                        // A non-null value is used as a signal that dragging
                        // is in progress, as well as indicating which shape
                        // is being dragged.

int prevDragX;  // During dragging, these record the x and y coordinates of the
int prevDragY;  //    previous position of the mouse.

public void mousePressed(MouseEvent evt) {
    // User has pressed the mouse.  Find the shape that the user has clicked on, if
    // any.  If there is no shape at the position when the mouse was clicked, then
    // ignore this event.  If there is then one of three things will happen:
    // If the event is a pop-up trigger, then the pop-up menu is displayed, and
    // the user can select from the pop-up menu to edit the shape.  If the user was
    // holding down the shift key, then bring the clicked shape to the front, in
    // front of all the other shapes.  Otherwise, start dragging the shape.
```

```
if (draggedShape != null) {
    // A drag operation is already in progress, so ignore this click.
    // This might happen if the user clicks a second mouse button before
    // releasing the first one(?).
  return;
}
int x = evt.getX();  // x-coordinate of point where mouse was clicked
int y = evt.getY();  // y-coordinate of point
clickedShape = null;  // This will be set to the clicked shape, if any.
for ( int i = shapes.size() - 1; i >= 0; i-- ) {
     // Check shapes from front to back.
  Shape s = (Shape)shapes.get(i);
  if (s.containsPoint(x,y)) {
     clickedShape = s;
     break;
  }
}
if (clickedShape == null) {
    // The user did not click on a shape.
  return;
}
else if (evt.isPopupTrigger()) {
    // The user wants to see the pop-up menu
```

```
        popup.show(this,x-10,y-2);
    }
    else if (evt.isShiftDown()) {
          // Bring the clicked shape to the front
        shapes.remove(clickedShape);
        shapes.add(clickedShape);
        repaint();
    }
    else {
          // Start dragging the shape.
        draggedShape = clickedShape;
        prevDragX = x;
        prevDragY = y;
    }
}

public void mouseDragged(MouseEvent evt) {
        // User has moved the mouse.  Move the dragged shape by the same amount.
    if (draggedShape == null) {
            // User did not click a shape.  There is nothing to do.
        return;
    }
    int x = evt.getX();
```

```
   int y = evt.getY();

   draggedShape.moveBy(x - prevDragX, y - prevDragY);

   prevDragX = x;

   prevDragY = y;

   repaint();      // redraw canvas to show shape in new position

}


public void mouseReleased(MouseEvent evt) {

      // User has released the mouse.  Move the dragged shape, and set

      // draggedShape to null to indicate that dragging is over.

      // If the shape lies completely outside the canvas, remove it

      // from the list of shapes (since there is no way to ever move

      // it back on screen).  However, if the event is a popup trigger

      // event, then show the popup menu instead.

   if (draggedShape == null) {

       // User did not click on a shape. There is nothing to do.

     return;

   }

   int x = evt.getX();

   int y = evt.getY();

   if (evt.isPopupTrigger()) {

       // Check whether the user is trying to pop up a menu.

       // (This should be checked in both the mousePressed() and
          // mouseReleased() methods.)
        popup.show(this,x-10,y-2);
      }
      else {
        draggedShape.moveBy(x - prevDragX, y - prevDragY);
        if ( draggedShape.left >= getSize().width || draggedShape.top >= getSize().height
||
              draggedShape.left + draggedShape.width < 0 ||
              draggedShape.top + draggedShape.height < 0 ) {  // shape is off-screen
          shapes.remove(draggedShape);  // remove shape from list of shapes
        }
        repaint();
      }
      draggedShape = null;  // Dragging is finished.
    }

    public void mouseEntered(MouseEvent evt) { }   // Other methods required for
MouseListener and
    public void mouseExited(MouseEvent evt) { }    //          MouseMotionListener
interfaces.
    public void mouseMoved(MouseEvent evt) { }
    public void mouseClicked(MouseEvent evt) { }

  }  // end class ShapeCanvas
```

```
// ------- Nested class definitions for the abstract Shape class and three -----
// -------------------- concrete subclasses of Shape. --------------------------


static abstract class Shape {

    // A class representing shapes that can be displayed on a ShapeCanvas.
    // The subclasses of this class represent particular types of shapes.
    // When a shape is first constructed, it has height and width zero
    // and a default color of white.

  int left, top;      // Position of top left corner of rectangle that bounds this shape.
  int width, height;  // Size of the bounding rectangle.
  Color color = Color.white;  // Color of this shape.
  boolean drawOutline;  // If true, a black border is drawn on the shape

  void reshape(int left, int top, int width, int height) {
      // Set the position and size of this shape.
    this.left = left;
    this.top = top;
    this.width = width;
    this.height = height;
```

```
}


void setSize(int width, int height) {
      // Set the size of this shape
   this.width = width;
   this.height = height;
}


void moveBy(int dx, int dy) {
       // Move the shape by dx pixels horizontally and dy pixels vertically
       // (by changing the position of the top-left corner of the shape).
   left += dx;
   top += dy;
}


void setColor(Color color) {
       // Set the color of this shape
   this.color = color;
}


void setDrawOutline(boolean draw) {
       // If true, a black outline is drawn around this shape.
   drawOutline = draw;
   }

   boolean containsPoint(int x, int y) {
        // Check whether the shape contains the point (x,y).
        // By default, this just checks whether (x,y) is inside the
        // rectangle that bounds the shape.  This method should be
        // overridden by a subclass if the default behavior is not
        // appropriate for the subclass.
      if (x >= left && x < left+width && y >= top && y < top+height)
        return true;
      else
        return false;
   }

   abstract void draw(Graphics g);
        // Draw the shape in the graphics context g.
        // This must be overridden in any concrete subclass.

}  // end of class Shape




static class RectShape extends Shape {
      // This class represents rectangle shapes.
   void draw(Graphics g) {
      g.setColor(color);
      g.fillRect(left,top,width,height);
      if (drawOutline) {
        g.setColor(Color.black);
        g.drawRect(left,top,width,height);
      }
   }
}


static class OvalShape extends Shape {
     // This class represents oval shapes.
   void draw(Graphics g) {
      g.setColor(color);
      g.fillOval(left,top,width,height);
      if (drawOutline) {
        g.setColor(Color.black);
        g.drawOval(left,top,width,height);
      }
   }
   boolean containsPoint(int x, int y) {
        // Check whether (x,y) is inside this oval, using the
        // mathematical equation of an ellipse.
      double rx = width/2.0;   // horizontal radius of ellipse
      double ry = height/2.0;  // vertical radius of ellipse
      double cx = left + rx;   // x-coord of center of ellipse
      double cy = top + ry;    // y-coord of center of ellipse
      if ( (ry*(x-cx))*(ry*(x-cx)) + (rx*(y-cy))*(rx*(y-cy)) <= rx*rx*ry*ry )
        return true;
      else
     return false;
   }
}



static class RoundRectShape extends Shape {
     // This class represents rectangle shapes with rounded corners.
     // (Note that it uses the inherited version of the
     // containsPoint(x,y) method, even though that is not perfectly
     // accurate when (x,y) is near one of the corners.)
   void draw(Graphics g) {
      g.setColor(color);
      g.fillRoundRect(left,top,width,height,width/3,height/3);
      if (drawOutline) {
            g.setColor(Color.black);

            g.drawRoundRect(left,top,width,height,width/3,height/3);

          }

        }

      }



}  // end class ShapeDrawFrame
```

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------------------