מטלה 4 בJAVA

שם: אורטל קמינקא

ת.ז: 302919691

DrawFrame

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.awt.BorderLayout;

import java.awt.Color;

/\*\*

\* DrawFrame class constitutes the main window of the application

\* (extends {@link javax.swing.JFrame}). This class is responsible

\* for creating all the GUI elements and organize them.

\*/

public class DrawFrame extends JFrame

{

private JButton undoButton, clearButton;

private JComboBox<String> colorChoices, shapeChoices;

private JCheckBox filledCheckBox;

private JPanel buttons; // buttons in north side

private JLabel statusLabel; // South side of frame

private DrawPanel drawArea; // center Jpanel for drawing

private static final String[] shapeName = { "Line", "Oval", "Rectangle"};

private static final String[] colorNames = {"Black", "Blue", "Cyan",

"Dark Gray", "Gray", "Green", "Light Gray", "Magenta",

"Orange", "Pink", "Red", "White", "Yellow"};

private static final Color[] colors = {Color.BLACK, Color.BLUE,

Color.CYAN, Color.DARK\_GRAY, Color.GRAY, Color.GREEN,

Color.LIGHT\_GRAY, Color.MAGENTA, Color.ORANGE, Color.PINK,

Color.RED, Color.WHITE, Color.YELLOW};

ButtonHandler handler;

ItemHandler IHandler;

/\*\*

\* default Constructor initializes the frame GUI section and buttons handlers.

\*/

DrawFrame()

{

super("Shape Drawings"); //the text in the up bar

setLayout(new BorderLayout());//set frame layout

//init the buttons

buttons = new JPanel();

undoButton=new JButton ("Undo");

buttons.add(undoButton);

clearButton=new JButton ("Clear");

buttons.add(clearButton);

colorChoices=new JComboBox<String> (colorNames);

buttons.add(colorChoices);

shapeChoices=new JComboBox<String> (shapeName);

buttons.add(shapeChoices);

filledCheckBox=new JCheckBox ("Filled");

buttons.add(filledCheckBox);

add(buttons, BorderLayout.NORTH); // add buttons to north size

statusLabel=new JLabel("(0,0)");

add(statusLabel,BorderLayout.SOUTH);

drawArea=new DrawPanel(statusLabel);

add(drawArea, BorderLayout.CENTER); // add buttons to north size

///////////////////

handler =new ButtonHandler();

undoButton.addActionListener(handler);

clearButton.addActionListener(handler);

////////////////////////

IHandler = new ItemHandler();

filledCheckBox.addItemListener(IHandler);

colorChoices.addItemListener(IHandler);

shapeChoices.addItemListener(IHandler);

}

/\*\*

\* listener for color, shape and filled sections and uses set functions of {@link DrawPanel}.

\*

\*/

private class ItemHandler implements ItemListener

{

@Override

public void itemStateChanged(ItemEvent event)

{

if(event.getSource()== filledCheckBox)

drawArea.setFilledShape(filledCheckBox.isSelected());

else if (event.getSource() == colorChoices)

drawArea.setCurrentColor(colors[colorChoices.getSelectedIndex()]);

else if (event.getSource() == shapeChoices)

drawArea.setShapeType(shapeChoices.getSelectedIndex());

}

}

/\*\*

\* check if undo or clear was pressed and using {@link DrawPanel} methodes

\* {@link DrawPanel#undo()} and {@link DrawPanel#clear()}.

\*/

private class ButtonHandler implements ActionListener

{

//handle button event

@Override

public void actionPerformed (ActionEvent event)

{

if (event.getSource()==undoButton)

drawArea.undo();

else if (event.getSource()==clearButton)

drawArea.clear();

}

}

}

DrawPanel

import javax.swing.\*;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.awt.event.MouseEvent;

import javax.swing.JPanel;

import javax.swing.JOptionPane;

import java.awt.event.MouseAdapter;

/\*\*

\* Smart drawing panel that handle mouse and mouseMotion actions.

\*/

public class DrawPanel extends JPanel

{

private MyShape[] shapes; // array of Shapes

private int shapeCount; //status how much Shapes we have

private MyShape currentShape;

private int shapeType; // 0 :Line, 1 :Oval, 2: Rectangle

private Color CurrentColor;

private boolean filledShape;

private JLabel statusText; // String containing shape statistic information

private boolean newShapeStart;

/\*\*

\* Constructor initializes Characteristics of shape and status Label

\* @param statusLabel reference of {@link javax.swing.JLabel} for corrdinate updating

\*/

public DrawPanel(JLabel statusLabel)

{

setBackground(Color.WHITE);

shapes=new MyShape[100];

this.statusText=statusLabel;

setShapeType(0);

setCurrentColor(Color.BLACK);

setFilledShape(false);

MouseHandler handler =new MouseHandler();

addMouseMotionListener(handler);

addMouseListener(handler);

}

// for each shape array, draw the individual shapes

/\*\*

\* paintComponent method draw the individual shapes for each shape array

\* @param g Graphics shape

\*/

public void paintComponent(Graphics g)

{

super.paintComponent(g);

for (int i = 0; i < shapeCount; i++)

shapes[i].draw(g);

if (newShapeStart)

currentShape.draw(g);

}

/\*\*

\* set the type of shape - in this program, using in DrawFrame as a action for buttons selections.

\* @param shapeType integer (0={@link MyLine}, 1={@link MyOval}, 2={@link MyRectangle}).

\*/

final public void setShapeType( int shapeType)

{ this.shapeType=shapeType;

}

/\*\*

\* set shape color - in this program, using in DrawFrame as a action for buttons selections.

\* @param CurrentColor {@link java.awt.color} type

\*/

final public void setCurrentColor( Color CurrentColor)

{ this.CurrentColor=CurrentColor;

}

/\*\*

\* set if type is filled (only for {@link MyOval}/{@link MyRectangle}) - in this program, using in DrawFrame as a action for buttons selections.

\* @param filledShape booleane for isFilled

\*/

final public void setFilledShape( boolean filledShape)

{ this.filledShape=filledShape;

}

/\*\*

\* erase the last shape that was drawed and updates the size of the shape.

\*/

public void undo()

{

if (shapeCount > 0)

shapeCount--;

repaint();

}

/\*\*

\* erase all the shapes that was draw. initilaze the size of the to zero.

\*/

public void clear()

{

shapeCount = 0;

repaint();

}

/\*\*

\* Define mouse action and motion for draw max 100 shape, using only left click of the mouse.

\*/

private class MouseHandler extends MouseAdapter implements MouseListener,

MouseMotionListener

{

// handle event when mouse pressed

@Override

public void mousePressed(MouseEvent event)

{

if (event.getButton() == MouseEvent.BUTTON1){ // left Button

if (shapeCount == shapes.length) // limited for array size, in this case - 100

{

JOptionPane.showMessageDialog(null, "Cannot exceed the current number of shapes:"+shapeCount,"Capacity Overload", JOptionPane.WARNING\_MESSAGE);

return;

}

//there is place foe shape

newShapeStart = true;

int x = event.getX();

int y = event.getY();

switch(shapeType)

{

case 0: currentShape = new MyLine(x, y, x, y, CurrentColor);

break;

case 1: currentShape = new MyOval(x, y, x, y, CurrentColor, filledShape);

break;

case 2: currentShape = new MyRectangle(x, y, x, y, CurrentColor, filledShape);

break;

}

}

}

// handle event when mouse released

@Override

public void mouseReleased(MouseEvent event)

{

if (event.getButton() == MouseEvent.BUTTON1) // // left Button Released

{

if (event.getX() != currentShape.getX1() || event.getY() != currentShape.getY1())

{

currentShape.setX2(event.getX());

currentShape.setY2(event.getY());

shapes[shapeCount] = currentShape;

shapeCount++;

}

newShapeStart=false;

repaint();

}

}

// MouseMotionListener event handlers

// handle event when user drags mouse with button pressed

@Override

public void mouseDragged(MouseEvent event)

{

if (newShapeStart)

{

currentShape.setX2(event.getX());

currentShape.setY2(event.getY());

repaint();

}

mouseMoved(event);

}

// handle event when user moves mouse

@Override

public void mouseMoved(MouseEvent event)

{

statusText.setText(String.format("(%d, %d)", event.getX(), event.getY()));

}

} // end inner class MouseHandler

} // end class DrawPanel

MyBoundedShape

import java.awt.Color;

import java.awt.Graphics;

/\*\*

\* this class Declaration of class MyBoundedShape.

\*/

public class MyBoundedShape extends MyShape

{

private boolean filled; // whether this shape is filled

/\*\*

\* MyBoundedShape constructor initializes private vars with default values

\*/

public MyBoundedShape()

{

super();

}

/\*\*

\* MyBoundedShape constructor with input values

\*@param x1 x coordinate of first endpoint

\*@param y1 y coordinate of first endpoint

\*@param x2 x coordinate of second endpoint

\*@param y2 y coordinate of second endpoint

\*@param myColor the color of the shape

\*@param filled if the shape is filled

\*/

public MyBoundedShape(int x1, int y1, int x2, int y2,

Color myColor, boolean filled)

{

super(x1,y1,x2,y2,myColor);

this.filled = filled;

}

/\*\*

\* isFilled method determines whether this shape is filled

\* @return if shape is filled

\*/

public boolean isFilled()

{

return filled;

}

/\*\*

\* setFilled method sets whether this shape is filled

\* @param filled if shape is filled

\*/

public void setFilled(boolean filled)

{

this.filled = filled;

}

/\*\*

\* getUpperLeftX method get upper left x coordinate

\* @return upper left x coordinate

\*/

public int getUpperLeftX()

{

return Math.min(super.getX1(), super.getX2());

}

/\*\*

\* getUpperLeftY method get upper left x coordinate

\* @return upper left x coordinate

\*/

public int getUpperLeftY()

{

return Math.min(super.getY1(), super.getY2());

}

/\*\*

\* getWidth method get shape width

\* @return shape width

\*/

public int getWidth()

{

return Math.abs(super.getX2() - super.getX1());

}

/\*\*

\* getHeight method get shape height

\* @return shape height

\*/

public int getHeight()

{

return Math.abs(super.getY2() - super.getY1());

}

/\*\*

\* draw method draws an rectangle in the specified color

\* @param g Graphics shape

\*/

public void draw(Graphics g)

{

g.setColor(super.getColor());

if (isFilled())

g.fillRect(getUpperLeftX(), getUpperLeftY(),

getWidth(), getHeight());

else

g.drawRect(getUpperLeftX(), getUpperLeftY(),

getWidth(), getHeight());

}

} // end class MyBoundedShape

MyLine

import java.awt.Color;

import java.awt.Graphics;

/\*\*

\* this class Declaration of class MyLine.

\*/

public class MyLine extends MyShape

{

/\*\*

\* MyLine constructor initializes private vars with default values

\*/

public MyLine()

{

super(); // call constructor to set values

} // end MyLine no-argument constructor

/\*\*

\* MyLine constructor initializes private vars with input values

\*@param x1 x coordinate of first endpoint

\*@param y1 y coordinate of first endpoint

\*@param x2 x coordinate of second endpoint

\*@param y2 y coordinate of second endpoint

\*@param myColor the color of the shape

\*/

public MyLine(int x1, int y1, int x2, int y2, Color myColor)

{

super(x1,y1,x2,y2,myColor);

}

/\*\*

\* draw method draw the line in the specified color

\* @param g Graphics shape

\*/

public void draw(Graphics g)

{

g.setColor(super.getColor());

g.drawLine(super.getX1(), super.getY1(), super.getX2(), super.getY2());

}

} // end class MyLine

MyOval

import java.awt.Color;

import java.awt.Graphics;

/\*\*

\* Declaration of class MyOval.

\*/

public class MyOval extends MyBoundedShape

{

/\*\*

\* MyOval constructor initializes private vars with default values

\*/

public MyOval()

{

super();

}

// constructor with input values

/\*\*

\* MyOval constructor with input values

\*@param x1 x coordinate of first endpoint

\*@param y1 y coordinate of first endpoint

\*@param x2 x coordinate of second endpoint

\*@param y2 y coordinate of second endpoint

\*@param myColor the color of the shape

\*@param filled if the shape is filled

\*/

public MyOval(int x1, int y1, int x2, int y2,

Color myColor, boolean filled)

{

super(x1,y1,x2,y2,myColor,filled);

}

/\*\*

\* draw method draws an oval in the specified color

\* @param g Graphics shape

\*/

public void draw(Graphics g)

{

g.setColor(super.getColor());

if (isFilled())

g.fillOval(getUpperLeftX(), getUpperLeftY(),

getWidth(), getHeight());

else

g.drawOval(getUpperLeftX(), getUpperLeftY(),

getWidth(), getHeight());

}

} // end class MyOval

MyRectangle

import java.awt.Color;

import java.awt.Graphics;

/\*\*

\* this class Declaration of class MyRectangle.

\*/

public class MyRectangle extends MyBoundedShape

{

/\*\*

\* MyRectangle constructor initializes private vars with default values

\*/

public MyRectangle()

{

super();

}

/\*\*

\* MyRectangle constructor with input values

\*@param x1 x coordinate of first endpoint

\*@param y1 y coordinate of first endpoint

\*@param x2 x coordinate of second endpoint

\*@param y2 y coordinate of second endpoint

\*@param myColor the color of the shape

\*@param filled if the shape is filled

\*/

public MyRectangle(int x1, int y1, int x2, int y2,

Color myColor, boolean filled)

{

super(x1,y1,x2,y2,myColor,filled);

}

/\*\*

\* draw method draws an rectangle in the specified color

\* @param g Graphics shape

\*/

public void draw(Graphics g)

{

g.setColor(super.getColor());

if (isFilled())

g.fillRect(getUpperLeftX(), getUpperLeftY(),

getWidth(), getHeight());

else

g.drawRect(getUpperLeftX(), getUpperLeftY(),

getWidth(), getHeight());

}

} // end class MyRectangle

MyShape

import java.awt.Color;

import java.awt.Graphics;

/\*\*

\* Declaration of abstract class MyShape.

\*/

public abstract class MyShape

{

private int x1; // x coordinate of first endpoint

private int y1; // y coordinate of first endpoint

private int x2; // x coordinate of second endpoint

private int y2; // y coordinate of second endpoint

private Color myColor; // color of this shape

/\*\*

\* MyShape constructor initializes private vars with default values

\*/

public MyShape()

{

this(0, 0, 0, 0, Color.BLACK); // call constructor

}

/\*\*

\* MyShape constructor with input values

\*@param x1 x coordinate of first endpoint

\*@param y1 y coordinate of first endpoint

\*@param x2 x coordinate of second endpoint

\*@param y2 y coordinate of second endpoint

\*@param myColor the color of the shape

\*/

public MyShape(int x1, int y1, int x2, int y2, Color myColor)

{

this.x1 = (x1 >= 0 ? x1 : 0);

this.y1 = (y1 >= 0 ? y1 : 0);

this.x2 = (x2 >= 0 ? x2 : 0);

this.y2 = (y2 >= 0 ? y2 : 0);

this.myColor = myColor;

}

/\*\*

\* setX1 method set the x-coordinate of the first point

\* @param x1 x coordinate of first endpoint

\*/

public void setX1(int x1)

{

this.x1 = (x1 >= 0 ? x1 : 0);

}

/\*\*

\* getX1 method get the x-coordinate of the first point

\* @return x coordinate of first endpoint

\*/

public int getX1()

{

return x1;

}

/\*\*

\* setX2 method set the x-coordinate of the second point

\* @param x2 x coordinate of second endpoint

\*/

public void setX2(int x2)

{

this.x2 = (x2 >= 0 ? x2 : 0);

}

/\*\*

\* getX2 method get the x-coordinate of the second point

\* @return x coordinate of second endpoint

\*/

public int getX2()

{

return x2;

}

/\*\*

\* setY1 method set the y-coordinate of the first point

\* @param y1 y coordinate of first endpoint

\*/

public void setY1(int y1)

{

this.y1 = (y1 >= 0 ? y1 : 0);

}

/\*\*

\* getY1 method get the y-coordinate of the first point

\* @return y coordinate of first endpoint

\*/

public int getY1()

{

return y1;

}

/\*\*

\* setY2 method set the y-coordinate of the second point

\* @param y2 y coordinate of second endpoint

\*/

public void setY2(int y2)

{

this.y2 = (y2 >= 0 ? y2 : 0);

}

/\*\*

\* getY2 method get the y-coordinate of the second point

\* @return y coordinate of second endpoint

\*/

public int getY2()

{

return y2;

}

/\*\*

\* setColor method set the color

\* @param myColor color of shape

\*/

public void setColor(Color myColor)

{

this.myColor = myColor;

}

/\*\*

\* getColor method get the color

\* @return color of shape

\*/

public Color getColor()

{

return myColor;

}

/\*\*

\* draw abstract method draws an shape in the specified color

\* @param g Graphics shape

\*/

public abstract void draw(Graphics g);

}

TestDraw

import javax.swing.JFrame;

import javax.swing.JLabel;

/\*\*

\* Test application to display a DrawPanel

\*/

public class TestDraw

{

/\*\*

\* static main method get from the user how much shapes he want and open the window for draw

\*/

public static void main(String[] args)

{

DrawFrame application = new DrawFrame();

application.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

application.setSize(500, 500);

application.setResizable(true);

application.setVisible(true);

} // end main

} // end class TestDraw



