package slateboard;

import java.awt.\*;

import java.applet.\*;

import java.awt.event.\*;

import java.awt.event.MouseMotionListener;

import java.awt.event.MouseEvent;

import java.lang.Math.\*;

import java.awt.Scrollbar.\*;

public class Whiteboard extends Applet implements ActionListener, AdjustmentListener, MouseListener, MouseMotionListener

{

/\* Maximum X and Maximum Y coordinate values. \*/

private final int MAX\_X = 800;

private final int MAX\_Y = 600;

/\* Operation Constants \*/

private final int NO\_OP = 0;

private final int PEN\_OP = 1;

private final int LINE\_OP = 2;

private final int ERASER\_OP = 3;

private final int CLEAR\_OP = 4;

private final int RECT\_OP = 5;

private final int OVAL\_OP = 6;

private final int FRECT\_OP = 7;

private final int FOVAL\_OP = 8;

private final int SPLINE\_OP = 9;

private final int POLY\_OP = 10;

/\* Current mouse coordinates \*/

private int mousex = 0;

private int mousey = 0;

/\* Previous mouse coordinates \*/

private int prevx = 0;

private int prevy = 0;

/\* Initial state falgs for operation \*/

private boolean initialPen = true;

private boolean initialLine = true;

private boolean initialEraser = true;

private boolean initialRect = true;

private boolean initialOval = true;

private boolean initialFRect = true;

private boolean initialFOval = true;

private boolean initialPolygon = true;

private boolean initialSpline = true;

/\* Main Mouse X and Y coordiante variables \*/

private int Orx = 0;

private int Ory = 0;

private int OrWidth = 0;

private int OrHeight = 0;

private int drawX = 0;

private int drawY = 0;

private int eraserLength = 5;

private int udefRedValue = 255;

private int udefGreenValue = 255;

private int udefBlueValue = 255;

/\* Primitive status & color variables \*/

private int opStatus = PEN\_OP;

private int colorStatus = 1;

private Color mainColor = new Color(0,0,0);

private Color xorColor = new Color(255,255,255);

private Color statusBarColor = new Color(166,166,255);

private Color userDefinedColor = new Color(udefRedValue,udefGreenValue,udefBlueValue);

/\* Operation Button definitions \*/

private Button penButton = new Button("Pen");

private Button lineButton = new Button("Line");

private Button eraserButton = new Button("Eraser");

private Button clearButton = new Button("Clear");

private Button rectButton = new Button("Rectangle");

private Button ovalButton = new Button("Oval");

private Button fillRectButton = new Button("Filled Rectangle");

private Button fillOvalButton = new Button("Filled Oval");

private Button splineButton = new Button("Spline");

private Button polygonButton = new Button("Polygon");

/\* Color Button definitions \*/

private Button blackButton = new Button("Black");

private Button blueButton = new Button("Blue");

private Button redButton = new Button("Red");

private Button greenButton = new Button("Green");

private Button purpleButton = new Button("Purple");

private Button orangeButton = new Button("Orange");

private Button pinkButton = new Button("Pink");

private Button grayButton = new Button("Gray");

private Button yellowButton = new Button("Yellow");

private Button userDefButton = new Button("User-Def");

/\* User defined Color variables \*/

private Scrollbar redSlider = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 255);

private Scrollbar blueSlider = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 255);

private Scrollbar greenSlider = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 255);

/\* Assorted status values for different variables \*/

private TextField colorStatusBar = new TextField(20);

private TextField opStatusBar = new TextField(20);

private TextField mouseStatusBar = new TextField(10);

private TextField redValue = new TextField(3);

private TextField greenValue = new TextField(3);

private TextField blueValue = new TextField(3);

/\* Labels for operation and color fields \*/

private Label operationLabel = new Label(" Tool mode:");

private Label colorLabel = new Label(" Color mode:");

private Label cursorLabel = new Label(" Cursor:");

/\* Sub panels of the main applet \*/

private Panel controlPanel = new Panel(new GridLayout(11,2,0,0));

private Panel drawPanel = new Panel();

private Panel statusPanel = new Panel();

private Panel udefcolPanel = new Panel(new GridLayout(3,2,0,0));

private Panel udefdemcolPanel = new Panel();

@Override

public void init()

{

setLayout(new BorderLayout());

/\* setup color buttons \*/

controlPanel.add(blackButton);

controlPanel.add(blueButton);

controlPanel.add(redButton);

controlPanel.add(greenButton);

controlPanel.add(purpleButton);

controlPanel.add(orangeButton);

controlPanel.add(pinkButton);

controlPanel.add(grayButton);

controlPanel.add(yellowButton);

controlPanel.add(userDefButton);

blueButton.setBackground(Color.blue);

redButton.setBackground(Color.red);

greenButton.setBackground(Color.green);

purpleButton.setBackground(Color.magenta);

orangeButton.setBackground(Color.orange);

pinkButton.setBackground(Color.pink);

grayButton.setBackground(Color.gray);

yellowButton.setBackground(Color.yellow);

userDefButton.setBackground(userDefinedColor);

/\* setup operation buttons \*/

controlPanel.add(penButton);

controlPanel.add(lineButton);

controlPanel.add(eraserButton);

controlPanel.add(clearButton);

controlPanel.add(rectButton);

controlPanel.add(ovalButton);

controlPanel.add(fillRectButton);

controlPanel.add(fillOvalButton);

controlPanel.add(splineButton);

controlPanel.add(polygonButton);

controlPanel.setBounds(0,0,100,300);

controlPanel.add(udefcolPanel);

controlPanel.add(udefdemcolPanel);

/\* Add user-defined RGB buttons to panel \*/

udefcolPanel.add(redValue);

udefcolPanel.add(redSlider);

udefcolPanel.add(greenValue);

udefcolPanel.add(greenSlider);

udefcolPanel.add(blueValue);

udefcolPanel.add(blueSlider);

/\* Add label and color text field \*/

statusPanel.add(colorLabel);

statusPanel.add(colorStatusBar);

/\* Add label and operation text field \*/

statusPanel.add(operationLabel);

statusPanel.add(opStatusBar);

/\* Add label and cursor text field \*/

statusPanel.add(cursorLabel);

statusPanel.add(mouseStatusBar);

/\* Set not editable \*/

colorStatusBar.setEditable(false);

opStatusBar.setEditable(false);

mouseStatusBar.setEditable(false);

statusPanel.setBackground(statusBarColor);

controlPanel.setBackground(Color.white);

drawPanel.setBackground(Color.white);

add(statusPanel, "North");

add(controlPanel, "West");

add(drawPanel, "Center");

/\* Setup action listener \*/

penButton.addActionListener(this);

lineButton.addActionListener(this);

eraserButton.addActionListener(this);

clearButton.addActionListener(this);

rectButton.addActionListener(this);

ovalButton.addActionListener(this);

fillRectButton.addActionListener(this);

fillOvalButton.addActionListener(this);

splineButton.addActionListener(this);

polygonButton.addActionListener(this);

blackButton.addActionListener(this);

blueButton.addActionListener(this);

redButton.addActionListener(this);

greenButton.addActionListener(this);

purpleButton.addActionListener(this);

orangeButton.addActionListener(this);

pinkButton.addActionListener(this);

grayButton.addActionListener(this);

yellowButton.addActionListener(this);

userDefButton.addActionListener(this);

redSlider.addAdjustmentListener(this);

blueSlider.addAdjustmentListener(this);

greenSlider.addAdjustmentListener(this);

/\* Adding component listeners to main panel (applet) \*/

drawPanel.addMouseMotionListener(this);

drawPanel.addMouseListener(this);

this.addMouseListener(this);

this.addMouseMotionListener(this);

updateRGBValues();

opStatusBar.setText("Pen");

colorStatusBar.setText("Black");

}

/\*

Method is called up when an action event has been preformed.

All button operations and some labels, text field operations

are handled in this method.

\*/

public void actionPerformed(ActionEvent e)

{

/\* Determine what action has occured \*/

/\* Set the relative values \*/

if (e.getActionCommand().equals("Pen"))

opStatus = PEN\_OP;

if (e.getActionCommand().equals("Line"))

opStatus = LINE\_OP;

if (e.getActionCommand().equals("Eraser"))

opStatus = ERASER\_OP;

if (e.getActionCommand().equals("Clear"))

opStatus = CLEAR\_OP;

if (e.getActionCommand().equals("Rectangle"))

opStatus = RECT\_OP;

if (e.getActionCommand().equals("Oval"))

opStatus = OVAL\_OP;

if (e.getActionCommand().equals("Filled Rectangle"))

opStatus = FRECT\_OP;

if (e.getActionCommand().equals("Filled Oval"))

opStatus = FOVAL\_OP;

if (e.getActionCommand().equals("Polygon"))

opStatus = POLY\_OP;

if (e.getActionCommand().equals("Spline"))

opStatus = SPLINE\_OP;

if (e.getActionCommand().equals("Black"))

colorStatus = 1;

if (e.getActionCommand().equals("Blue"))

colorStatus = 2;

if (e.getActionCommand().equals("Green"))

colorStatus = 3;

if (e.getActionCommand().equals("Red"))

colorStatus = 4;

if (e.getActionCommand().equals("Purple"))

colorStatus = 5;

if (e.getActionCommand().equals("Orange"))

colorStatus = 6;

if (e.getActionCommand().equals("Pink"))

colorStatus = 7;

if (e.getActionCommand().equals("Gray"))

colorStatus = 8;

if (e.getActionCommand().equals("Yellow"))

colorStatus = 9;

if (e.getActionCommand().equals("User-Def"))

colorStatus = 10;

initialPolygon = true;

initialSpline = true;

/\* Update Operations status bar, with current tool \*/

switch (opStatus)

{

case PEN\_OP : opStatusBar.setText("Pen");

break;

case LINE\_OP : opStatusBar.setText("Line");

break;

case ERASER\_OP: opStatusBar.setText("Eraser");

break;

case CLEAR\_OP : clearPanel(drawPanel);

break;

case RECT\_OP : opStatusBar.setText("Rectangle");

break;

case OVAL\_OP : opStatusBar.setText("Oval");

break;

case FRECT\_OP : opStatusBar.setText("Fill-Rectangle");

break;

case FOVAL\_OP : opStatusBar.setText("Fill-Oval");

break;

case POLY\_OP : opStatusBar.setText("Polygon");

break;

case SPLINE\_OP: opStatusBar.setText("Spline");

break;

}

/\* Update Color status bar, with current color \*/

switch (colorStatus)

{

case 1: colorStatusBar.setText("Black");

break;

case 2: colorStatusBar.setText("Blue");

break;

case 3: colorStatusBar.setText("Green");

break;

case 4: colorStatusBar.setText("Red");

break;

case 5: colorStatusBar.setText("Purple");

break;

case 6: colorStatusBar.setText("Orange");

break;

case 7: colorStatusBar.setText("Pink");

break;

case 8: colorStatusBar.setText("Gray");

break;

case 9: colorStatusBar.setText("Yellow");

break;

case 10: colorStatusBar.setText("User Defined Color");

break;

}

/\*

Set main color, to equivelent colorStatus value

\*/

setMainColor();

updateRGBValues();

}

public void adjustmentValueChanged(AdjustmentEvent e)

{

updateRGBValues();

}

/\*

Method will clear the whole drawPanel with

the current background color

\*/

public void clearPanel(Panel p)

{

opStatusBar.setText("Clear");

Graphics g = p.getGraphics();

g.setColor(p.getBackground());

g.fillRect(0,0,p.getBounds().width,p.getBounds().height);

}

/\*

Method will emulate a pen style graphic.

by drawing a line from the previous mouse corrdinates

to the current mouse coordinates.

Note: In initial attempt the previous mouse coordinates

are set to the current mouse coordinates so as

not to begin the pen graphic from an unwanted

arbitrary point.

\*/

public void penOperation(MouseEvent e)

{

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

/\*

In initial state setup default values

for mouse coordinates

\*/

if (initialPen)

{

setGraphicalDefaults(e);

initialPen = false;

g.drawLine(prevx,prevy,mousex,mousey);

}

/\*

Make sure that the mouse has actually

moved from its previous position.

\*/

if (mouseHasMoved(e))

{

/\*

set mouse coordinates to

current mouse position

\*/

mousex = e.getX();

mousey = e.getY();

/\*

draw a line from the previous mouse coordinates

to the current mouse coordinates

\*/

g.drawLine(prevx,prevy,mousex,mousey);

/\*

set the current mouse coordinates to

previous mouse coordinates for next time

\*/

prevx = mousex;

prevy = mousey;

}

}

/\*

Method will emulate a line drawing graphic.

By drawing a shadow line for an origin mouse

coordinate pair to a moving mouse coordinate

pair, until the mouse has been release from

dragmode.

\*/

public void lineOperation(MouseEvent e)

{

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

/\*

In initial state setup default values

for mouse coordinates

\*/

if (initialLine)

{

setGraphicalDefaults(e);

g.setXORMode(xorColor);

g.drawLine(Orx,Ory,mousex,mousey);

initialLine=false;

}

/\*

Make sure that the mouse has actually

moved from its previous position.

\*/

if (mouseHasMoved(e))

{

/\*

Delete previous line shadow

by xor-ing the graphical object

\*/

g.setXORMode(xorColor);

g.drawLine(Orx,Ory,mousex,mousey);

/\* Update new mouse coordinates \*/

mousex = e.getX();

mousey = e.getY();

/\* Draw line shadow \*/

g.drawLine(Orx,Ory,mousex,mousey);

}

}

/\*

Method will emulate a rectangle drawing graphic.

By drawing a shadow rectangle for an origin mouse

coordinate pair to a moving mouse coordinate

pair, until the mouse has been release from

dragmode.

\*/

public void rectOperation(MouseEvent e)

{

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

/\*

In initial state setup default values

for mouse coordinates

\*/

if (initialRect)

{

setGraphicalDefaults(e);

initialRect = false;

}

/\*

Make sure that the mouse has actually

moved from its previous position.

\*/

if (mouseHasMoved(e))

{

/\*

Delete previous rectangle shadow

by xor-ing the graphical object

\*/

g.setXORMode(drawPanel.getBackground());

g.drawRect(drawX,drawY,OrWidth,OrHeight);

/\* Update new mouse coordinates \*/

mousex = e.getX();

mousey = e.getY();

/\* Check new mouse coordinates for negative errors \*/

setActualBoundry();

/\* Draw rectangle shadow \*/

g.drawRect(drawX,drawY,OrWidth,OrHeight);

}

}

/\*

Method will emulate a oval drawing graphic.

By drawing a shadow oval for an origin mouse

coordinate pair to a moving mouse coordinate

pair, until the mouse has been release from

dragmode.

\*/

public void ovalOperation(MouseEvent e)

{

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

/\*

In initial state setup default values

for mouse coordinates

\*/

if (initialOval)

{

setGraphicalDefaults(e);

initialOval=false;

}

/\*

Make sure that the mouse has actually

moved from its previous position.

\*/

if (mouseHasMoved(e))

{

/\*

Delete previous oval shadow

by xor-ing the graphical object

\*/

g.setXORMode(xorColor);

g.drawOval(drawX,drawY,OrWidth,OrHeight);

/\* Update new mouse coordinates \*/

mousex = e.getX();

mousey = e.getY();

/\* Check new mouse coordinates for negative errors \*/

setActualBoundry();

/\* Draw oval shadow \*/

g.drawOval(drawX,drawY,OrWidth,OrHeight);

}

}

/\*

Method will emulate a filled-rectangle drawing graphic.

By drawing a shadow filled-rectangle for an origin mouse

coordinate pair to a moving mouse coordinate

pair, until the mouse has been release from

dragmode.

\*/

public void frectOperation(MouseEvent e)

{

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

/\*

In initial state setup default values

for mouse coordinates

\*/

if (initialFRect)

{

setGraphicalDefaults(e);

initialFRect=false;

}

/\*

Make sure that the mouse has actually

moved from its previous position.

\*/

if (mouseHasMoved(e))

{

/\*

Delete previous rectangle shadow

by xor-ing the graphical object

\*/

g.setXORMode(xorColor);

g.drawRect(drawX,drawY,OrWidth-1,OrHeight-1);

/\* Update new mouse coordinates \*/

mousex = e.getX();

mousey = e.getY();

/\* Check new mouse coordinates for negative errors \*/

setActualBoundry();

/\* Draw rectangle shadow \*/

g.drawRect(drawX,drawY,OrWidth-1,OrHeight-1);

}

}

/\*

Method will emulate a filled-oval drawing graphic.

By drawing a shadow filled-oval for an origin mouse

coordinate pair to a moving mouse coordinate

pair, until the mouse has been release from

dragmode.

\*/

public void fovalOperation(MouseEvent e)

{

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

/\*

In initial state setup default values

for mouse coordinates

\*/

if (initialFOval)

{

setGraphicalDefaults(e);

initialFOval = false;

}

/\*

Make sure that the mouse has actually

moved from its previous position.

\*/

if (mouseHasMoved(e))

{

/\*

Delete previous oval shadow

by xor-ing the graphical object

\*/

g.setXORMode(xorColor);

g.drawOval(drawX,drawY,OrWidth,OrHeight);

/\* Update new mouse coordinates \*/

mousex = e.getX();

mousey = e.getY();

/\* Check new mouse coordinates for negative errors \*/

setActualBoundry();

/\* Draw oval shadow \*/

g.drawOval(drawX,drawY,OrWidth,OrHeight);

}

}

/\*

Method will emulate a eraser graphic.

By drawing a filled rectangle of background color,

with the current mouse coordinates being the center

of the rectangle. This is done until the mouse has

been release from dragmode

\*/

@SuppressWarnings("static-access")

public void eraserOperation(MouseEvent e)

{

Graphics g = drawPanel.getGraphics();

/\*

In initial state setup default values

for mouse coordinates

\*/

if (initialEraser)

{

setGraphicalDefaults(e);

initialEraser = false;

g.setColor(mainColor.white);

g.fillRect(mousex-eraserLength, mousey-eraserLength,eraserLength\*2,eraserLength\*2);

g.setColor(Color.black);

g.drawRect(mousex-eraserLength,mousey-eraserLength,eraserLength\*2,eraserLength\*2);

prevx = mousex;

prevy = mousey;

}

/\*

Make sure that the mouse has actually

moved from its previous position.

\*/

if (mouseHasMoved(e))

{

g.setColor(mainColor.white);

g.drawRect(prevx-eraserLength, prevy-eraserLength,eraserLength\*2,eraserLength\*2);

/\* Get current mouse coordinates \*/

mousex = e.getX();

mousey = e.getY();

/\* Draw eraser block to panel \*/

g.setColor(mainColor.white);

g.fillRect(mousex-eraserLength, mousey-eraserLength,eraserLength\*2,eraserLength\*2);

g.setColor(Color.black);

g.drawRect(mousex-eraserLength,mousey-eraserLength,eraserLength\*2,eraserLength\*2);

prevx = mousex;

prevy = mousey;

}

}

/\*

Method will draw a polygon of N points on drawable surface

\*/

public void polygonOperation(MouseEvent e)

{

if (initialPolygon)

{

prevx = e.getX();

prevy = e.getY();

initialPolygon = false;

}

else

{

mousex = e.getX();

mousey = e.getY();

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

g.drawLine(prevx,prevy,mousex,mousey);

prevx = mousex;

prevy = mousey;

}

}

/\*

Not fully implemented spline operation

\*/

public void splineOperation(MouseEvent e)

{

if(initialSpline)

{

initialSpline = false;

}

}

/\*

Method determines weather the mouse has moved

from its last recorded position.

If mouse has deviated from previous position,

the value returned will be true, otherwise

the value that is returned will be false.

\*/

public boolean mouseHasMoved(MouseEvent e)

{

return (mousex != e.getX() || mousey != e.getY());

}

/\*

Method is a key segment in the operations where

there are more than 2 variables deviating to

makeup a graphical operation.

This method calculates the new values for the

global varibles drawX and drawY according to

the new positions of the mouse cursor.

This method eleviates the possibility that

a negative width or height can occur.

\*/

public void setActualBoundry()

{

/\*

If the any of the current mouse coordinates

are smaller than the origin coordinates, meaning

if drag occured in a negative manner, where either

the x-shift occured from right and/or y-shift occured

from bottom to top.

\*/

if (mousex < Orx || mousey < Ory)

{

/\*

if the current mouse x coordinate is smaller

than the origin x coordinate,

equate the drawX to be the difference between

the current width and the origin x coordinate.

\*/

if (mousex < Orx)

{

OrWidth = Orx - mousex;

drawX = Orx - OrWidth;

}

else

{

drawX = Orx;

OrWidth = mousex - Orx;

}

/\*

if the current mouse y coordinate is smaller

than the origin y coordinate,

equate the drawY to be the difference between

the current height and the origin y coordinate.

\*/

if (mousey < Ory)

{

OrHeight = Ory - mousey;

drawY = Ory - OrHeight;

}

else

{

drawY = Ory;

OrHeight = mousey - Ory;

}

}

/\*

Else if drag was done in a positive manner meaning

x-shift occured from left to right and or y-shift occured

from top to bottom

\*/

else

{

drawX = Orx;

drawY = Ory;

OrWidth = mousex - Orx;

OrHeight = mousey - Ory;

}

}

/\*

Method sets all the drawing varibles to the default

state which is the current position of the mouse cursor

Also the height and width varibles are zeroed off.

\*/

public void setGraphicalDefaults(MouseEvent e)

{

mousex = e.getX();

mousey = e.getY();

prevx = e.getX();

prevy = e.getY();

Orx = e.getX();

Ory = e.getY();

drawX = e.getX();

drawY = e.getY();

OrWidth = 0;

OrHeight = 0;

}

/\*

Method will be activated when mouse is being dragged.

depending on what operation is the opstatus, the switch

statement will call the relevent operation

\*/

public void mouseDragged(MouseEvent e)

{

updateMouseCoordinates(e);

switch (opStatus)

{

/\* If opStatus is PEN\_OP then call penOperation method \*/

case PEN\_OP : penOperation(e);

break;

/\* If opStatus is LINE\_OP then call lineOperation method \*/

case LINE\_OP : lineOperation(e);

break;

/\* If opStatus is RECt\_OP then call rectOperation method \*/

case RECT\_OP : rectOperation(e);

break;

/\* If opStatus is OVAL\_OP then call ovalOperation method \*/

case OVAL\_OP : ovalOperation(e);

break;

/\* If opStatus is FRECT\_OP then call frectOperation method \*/

case FRECT\_OP : frectOperation(e);

break;

/\* If opStatus is FOVAL\_OP then call fovalOperation method \*/

case FOVAL\_OP : fovalOperation(e);

break;

/\* If opStatus is ERASER\_OP then call eraserOperation method \*/

case ERASER\_OP: eraserOperation(e);

break;

}

}

/\*

Method will be activated when mouse has been release from pressed \

mode. At this stage the method will call the finalization routines

for the current operation.

\*/

public void mouseReleased(MouseEvent e)

{

/\* Update current mouse coordinates to screen \*/

updateMouseCoordinates(e);

switch (opStatus)

{

/\* If opStatus is PEN\_OP then call releasedPen method \*/

case PEN\_OP : releasedPen();

break;

/\* If opStatus is LINE\_OP then call releasedLine method \*/

case LINE\_OP : releasedLine();

break;

/\* If opStatus is RECT\_OP then call releasedRect method \*/

case RECT\_OP : releasedRect();

break;

/\* If opStatus is OVAL\_OP then call releasedOval method \*/

case OVAL\_OP : releasedOval();

break;

/\* If opStatus is FRECT\_OP then call releasedFrect method \*/

case FRECT\_OP : releasedFRect();

break;

/\* If opStatus is FOVAL\_OP then call releasedFoval method \*/

case FOVAL\_OP : releasedFOval();

break;

/\* If opStatus is ERASER\_OP then call releasedEraser method \*/

case ERASER\_OP : releasedEraser();

break;

}

}

/\*

Method will be activated when mouse enters the applet area.

This method will then update the current mouse x and coordinates

on the screen.

\*/

public void mouseEntered(MouseEvent e)

{

updateMouseCoordinates(e);

}

/\*

Method will set the main system color according to the

current color status.

\*/

public void setMainColor()

{

switch (colorStatus)

{

case 1 : mainColor = Color.black;

break;

case 2: mainColor = Color.blue;

break;

case 3: mainColor = Color.green;

break;

case 4: mainColor = Color.red;

break;

case 5: mainColor = Color.magenta;

break;

case 6: mainColor = Color.orange;

break;

case 7: mainColor = Color.pink;

break;

case 8: mainColor = Color.gray;

break;

case 9: mainColor = Color.yellow;

break;

case 10: mainColor = userDefinedColor;

break;

}

}

/\*

Method is invoked when mouse has been released

and current operation is Pen

\*/

public void releasedPen()

{

initialPen = true;

}

/\*

Method is invoked when mouse has been released

and current operation is Line

\*/

public void releasedLine()

{

if ((Math.abs(Orx-mousex)+Math.abs(Ory-mousey)) != 0)

{

System.out.println("Line has been released....");

initialLine = true;

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

g.drawLine(Orx,Ory,mousex,mousey);

}

}

/\*

Method is invoked when mouse has been released

and current operation is Eraser

\*/

@SuppressWarnings("static-access")

public void releasedEraser()

{

initialEraser = true;

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor.white);

g.drawRect(mousex-eraserLength,mousey-eraserLength,eraserLength\*2,eraserLength\*2);

}

/\*

Method is invoked when mouse has been released

and current operation is Rectangle

\*/

public void releasedRect()

{

initialRect = true;

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

g.drawRect(drawX,drawY,OrWidth,OrHeight);

}

/\*

Method is invoked when mouse has been released

and current operation is Oval

\*/

public void releasedOval()

{

initialOval = true;

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

g.drawOval(drawX,drawY,OrWidth,OrHeight);

}

/\*

Method is invoked when mouse has been released

and current operation is Fill-Rectangle

\*/

public void releasedFRect()

{

initialFRect = true;

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

g.fillRect(drawX,drawY,OrWidth,OrHeight);

}

/\*

Method is invoked when mouse has been released

and current operation is Fill-Oval

\*/

public void releasedFOval()

{

initialFOval = true;

Graphics g = drawPanel.getGraphics();

g.setColor(mainColor);

g.fillOval(drawX,drawY,OrWidth,OrHeight);

}

/\*

Method displays the mouse coordinates x and y values

and updates the mouse status bar with the new values.

\*/

public void updateMouseCoordinates(MouseEvent e)

{

String xCoor ="";

String yCoor ="";

if (e.getX() < 0) xCoor = "0";

else

{

xCoor = String.valueOf(e.getX());

}

if (e.getY() < 0) xCoor = "0";

else

{

yCoor = String.valueOf(e.getY());

}

mouseStatusBar.setText("x:"+xCoor+" y:"+yCoor);

}

/\*

Method updates user-defined values for udefRGB

\*/

public void updateRGBValues()

{

udefRedValue = redSlider.getValue();

udefGreenValue = greenSlider.getValue();

udefBlueValue = blueSlider.getValue();

if (udefRedValue > 255)

udefRedValue = 255;

if (udefRedValue < 0 )

udefRedValue =0;

if (udefGreenValue > 255)

udefGreenValue = 255;

if (udefGreenValue < 0 )

udefGreenValue =0;

if (udefBlueValue > 255)

udefBlueValue = 255;

if (udefBlueValue < 0 )

udefBlueValue =0;

redValue.setText(String.valueOf(udefRedValue));

greenValue.setText(String.valueOf(udefGreenValue));

blueValue.setText(String.valueOf(udefBlueValue));

userDefinedColor = new Color(udefRedValue,udefGreenValue,udefBlueValue);

userDefButton.setBackground(userDefinedColor);

Graphics g = udefdemcolPanel.getGraphics();

g.setColor(userDefinedColor);

g.fillRect(0,0,800,800);

}

/\*

Method updates mouse coordinates if mouse has been clicked

\*/

public void mouseClicked(MouseEvent e)

{

updateMouseCoordinates(e);

switch (opStatus)

{

//case 9 : splineOperation(e);

// break;

//case 10 : polygonOperation(e);

// break;

}

}

/\*

Method updates mouse coordinates if mouse has exited applet

\*/

public void mouseExited(MouseEvent e)

{

updateMouseCoordinates(e);

}

/\*

Method updates mouse coordinates if mouse has moved

\*/

public void mouseMoved(MouseEvent e)

{

updateMouseCoordinates(e);

}

/\*

Method updates mouse coordinates if mouse has been pressed

\*/

public void mousePressed(MouseEvent e)

{

updateMouseCoordinates(e);

}

} // End of Scribble