/\*\*

\* DrawPad.java

\*

\* @author rbender, mlucas, lmannion

\* @version 5/17/18

\* @contact bendryan14@qcsdstu.org

\*

\* Code Description: Defines a DrawPad class, encompassing the actual paint

\* methods, utilizing set colors and sizes

\*

\*/

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class DrawPad extends JPanel

{

private final int WIDTH = 600, HEIGHT = 400;

private int counter = 0;

private static int radius = 15;

private Circle circle;

private Circle circleArray [];

private static Color useColor;

/\*\*

\* Sets up this panel to listen for mouse events.

\*/

public DrawPad()

{

circleArray = new Circle [100];

addMouseListener (new DrawListener());

addMouseMotionListener (new DrawListener());

setPreferredSize (new Dimension(WIDTH, HEIGHT));

}

/\*

\* setColor(Color drawColor)

\*

\* @param drawColor new color to be set

\*

\* sets the color of future circles to be drawn

\*/

public void setColor(Color drawColor)

{

useColor = drawColor;

}

/\*

\* setRadius(int sizeNum)

\*

\* @param newRadius new radius to be set

\*

\* sets the radius of future circles to be drawn

\*/

public void setRadius(int newRadius)

{

radius = newRadius;

}

/\*\*

\* Clears the drawPad

\*

\*/

public void clear()

{

System.out.println("Made it to clear");

for(int i = 0; i < circleArray.length; i++)

circleArray[i] = null;

System.out.println("Made it to repaint()");

repaint();

}

/\*\*

\* resizes circleArray

\*

\*/

public void resizeCircleArray()

{

//local data member declared

Circle tempArray[] = new Circle [circleArray.length];

//conditional used to determine if the circle array is full,

//and will resize it if needed

if(null != circleArray[circleArray.length - 1])

{

//for-loop which copies circleArray into tempArray

for(int i = 0; i < circleArray.length; i++)

tempArray[i] = circleArray[i];

circleArray = new Circle[circleArray.length + 100];

//for-loop which copies tempArray into circleArray

for(int j = 0; j < tempArray.length; j++)

circleArray[j] = tempArray[j];

}

}

/\*\*

\* Draws the circles in circleArray, if any.

\*

\* @param page object on which to draw the circles

\*/

public void paintComponent (Graphics page)

{

super.paintComponent(page);

//for-loop which prints every circle in circleArray

for(int i = 0; i < circleArray.length; i++)

{

//conditional used to insure circleArray has available space left

//otherwise will resize circleArray

if(null != circleArray[i])

{

page.setColor(circleArray[i].getColor());

page.fillOval(circleArray[i].getX(), circleArray[i].getY(),

circleArray[i].getRadius(), circleArray[i].getRadius());

}

else

resizeCircleArray();

}

}

/\*\*

\* Represents the listener for Mouse events.

\*

\* @author rbender, mlucas, lmannion

\* @contact bendryan14@qcsdstu.org

\*/

private class DrawListener implements MouseListener, MouseMotionListener

{

/\*\*

\* Creates a new circle at the current location whenever the

\* mouse button is pressed and repaints.

\*

\* @param event the MouseEvent where the circle is to be placed

\*/

public void mousePressed (MouseEvent event)

{

//Circle object created

circle = new Circle(event.getPoint(), radius, useColor);

//conditional used to make sure array has enough space

if(null == circleArray[circleArray.length - 1])

{

circleArray[counter] = circle;

counter++;

}

else

{

resizeCircleArray();

}

repaint();

}

/\*\*

\* Continues to create a new circle at the current location whenever the

\* mouse button is dragged and repaints.

\*

\* @param event the MouseEvent where the circle is to be placed

\*/

public void mouseDragged(MouseEvent event)

{

//Circle object created

circle = new Circle(event.getPoint(), radius, useColor);

//conditional used to make sure array has enough space

if(null == circleArray[99])

{

circleArray[counter] = circle;

counter++;

}

else

{

resizeCircleArray();

circleArray[counter] = circle;

counter++;

repaint();

}

repaint();

}

/\*\*

\* Provides empty definitions for unused event methods.

\*

\*/

public void mouseEntered (MouseEvent event) {}

public void mouseExited (MouseEvent event) {}

public void mouseClicked (MouseEvent event) {}

public void mouseReleased (MouseEvent event) {}

public void mouseMoved(MouseEvent event){}

}

}