**Sketching**

Our first example was just a static drawing. Time to learn about *interactive* computer graphics. This means learning about **events**. Here is a little canvas you can sketch on, with a little main method so it can be run as an application:



Drawing Panel

import java.awt.BorderLayout;

import java.awt.Point;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.MouseMotionAdapter;

import java.awt.Graphics;

import java.io.Serial;

import java.util.ArrayList;

import java.util.List;

import javax.swing.SwingUtilities;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class CodeXam extends JPanel { //The drawing is defined to be a JPanel, which comes from the package javax.swing. A JPanel is a component you can draw in.

/\*

    \*Simply put, we use the serialVersionUID attribute to remember versions of a Serializable class to verify that a loaded class and the serialized object are compatible.

    \*The serialVersionUID attributes of different classes are independent. Therefore, it is not necessary for different classes to have unique values

    \*A serializable class can declare its own serialVersionUID explicitly by declaring a field named serialVersionUID that must be static, final, and of type long

    \*It is not necessary for two classes to have unique values. It means two different classes can have same serialVersionUID value\*/

    @Serial

    private static final long serialVersionUID = - 43L; //components have a silly serial version UID. Just put one in.

    private final List<List<Point>> curves = new ArrayList<>();

    public CodeXam() {

        // Register event listeners on construction of the panel.

        addMouseListener(new MouseAdapter() {

            public void mousePressed(MouseEvent e) {

                var newCurve = new ArrayList<Point>();

                newCurve.add(new Point(e.getX(), e.getY()));

                curves.add(newCurve);

            }

        });

        /\*You can read the component’s current size at any time with the graphics object’s getWidth() and getHeight() methods.

        Often, but not always, you’re drawing will be relative to these values.\*/

        addMouseMotionListener(new MouseMotionAdapter() {

            public void mouseDragged(MouseEvent e) {

                curves.get(curves.size() - 1).add(new Point(e.getX(), e.getY()));

                /\*\*

                 \* The panel saves the state of the drawing as a list of curves, where each curve is a list of points.

                 \* Pressing the mouse button starts a new curve; dragging the mouse adds the current location of the mouse to the current curve.

                 \* The entire drawing is rendered when needed, as usual, in paintComponent().

                 \*Notice the call to repaint when dragging. This tells Java to redraw the panel as soon as it can.

                 \*/

                repaint(0, 0, getWidth(), getHeight()); //The upper-left corner of the component is at (0,0). x-coordinates extend to the right; y-coordinates extend to the bottom.

            }

        });

    }

    public void paintComponent(Graphics g) {

        super.paintComponent(g);

        for (var curve: curves) {

            var previousPoint = curve.get(0);

            for (var point: curve) {

                g.drawLine(previousPoint.x, previousPoint.y, point.x, point.y);

                previousPoint = point;

            }

        }

    }

    /\*\*

     \* A little driver in case you want to sketch as a stand-alone application.

     \* We stuck a main method in there only so we can easily see the panel.

     \* In practice, we should make components just do their own thing,

     \* then write a separate application class with a main method.

     \*/

    public static void main(String[] args) { //

        SwingUtilities.invokeLater(() -> {

            var frame = new JFrame("Simple Sketching Program");

            frame.getContentPane().add(new CodeXam(), BorderLayout.CENTER);

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

            frame.setSize(400, 300);

            frame.setVisible(true);

        });

    }

}

The panel saves the state of the drawing as a list of curves, where each curve is a list of points. Pressing the mouse button starts a new curve; dragging the mouse adds the current location of the mouse to the current curve. The entire drawing is rendered when needed, as usual, in paintComponent().

Notice the call to repaint when dragging. This tells Java to redraw the panel as soon as it can.

Let’s break it down, and learn the basics as we go:

* The drawing is defined to be a JPanel, which comes from the package javax.swing. A jpanel is a **component** you can draw in.
* We stuck a main method in there only so you can easily see the panel. In practice, you would make your components just do their own thing, then write a separate application class with a main method.
* Every time the system detects that a component needs to be painted (e.g., it is made visible for the first time, or its window was resized), it (eventually) calls its paintComponent method and passes it a Graphics object. So doing graphics in Java is all about writing components and defining their paintComponent method.
* You can read the component’s current size at any time with the graphics object’s getWidth() and getHeight() methods. Often, but not always, you’re drawing will be relative to these values.
* The upper-left corner of the component is at (0,0). x-coordinates extend to the right; y-coordinates extend to the bottom.
* The graphics object is the thing that has all the good stuff in it. [Browse the documentation for this class now](https://docs.oracle.com/en/java/javase/13/docs/api/java.desktop/java/awt/Graphics.html).
* To draw or fill ovals and rectangles, you specify the upper-left coordinates, the width, and the height. Set the color before drawing with setColor.
* Yes, components have a silly serial version UID. Just put one in.
* Yes, the main method looks crazy. We’ll cover details in class.