**Flood fill**

import java.awt.\*;

import java.awt.event.\*;

public class FloodFillDemo extends Frame implements MouseListener {

int width = 400, height = 400;

int[][] pixels = new int[width][height]; // 0=empty, 1=filled (boundary), 2=filled (color)

public FloodFillDemo() {

setTitle("Flood Fill Demo");

setSize(width, height);

addMouseListener(this);

// Create a simple rectangle boundary

for (int x = 100; x <= 300; x++) {

pixels[x][100] = 1;

pixels[x][300] = 1;

}

for (int y = 100; y <= 300; y++) {

pixels[100][y] = 1;

pixels[300][y] = 1;

}

setVisible(true);

}

@Override

public void paint(Graphics g) {

for (int x = 0; x < width; x++) {

for (int y = 0; y < height; y++) {

if (pixels[x][y] == 1) {

g.setColor(Color.BLACK); // boundary

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

} else if (pixels[x][y] == 2) {

g.setColor(Color.RED); // filled

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

}

}

}

}

// Flood fill algorithm (4-connected)

public void floodFill(int x, int y) {

if (x < 0 || x >= width || y < 0 || y >= height)

return;

if (pixels[x][y] != 0)

return;

pixels[x][y] = 2; // fill color

floodFill(x + 1, y);

floodFill(x - 1, y);

floodFill(x, y + 1);

floodFill(x, y - 1);

}

@Override

public void mouseClicked(MouseEvent e) {

int x = e.getX();

int y = e.getY();

floodFill(x, y);

repaint();

}

public void mousePressed(MouseEvent e) {}

public void mouseReleased(MouseEvent e) {}

public void mouseEntered(MouseEvent e) {}

public void mouseExited(MouseEvent e) {}

public static void main(String[] args) {

new FloodFillDemo();

}

}

**Boundary Fill**

import java.awt.\*;

import java.awt.event.\*;

public class BoundaryFillDemo extends Frame implements MouseListener {

int width = 400, height = 400;

int[][] pixels = new int[width][height]; // 0=empty, 1=boundary, 2=filled

public BoundaryFillDemo() {

setTitle("Boundary Fill Demo");

setSize(width, height);

addMouseListener(this);

// Create a simple rectangle boundary

for (int x = 100; x <= 300; x++) {

pixels[x][100] = 1;

pixels[x][300] = 1;

}

for (int y = 100; y <= 300; y++) {

pixels[100][y] = 1;

pixels[300][y] = 1;

}

setVisible(true);

}

@Override

public void paint(Graphics g) {

for (int x = 0; x < width; x++) {

for (int y = 0; y < height; y++) {

if (pixels[x][y] == 1) {

g.setColor(Color.BLACK); // boundary

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

} else if (pixels[x][y] == 2) {

g.setColor(Color.BLUE); // filled

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

}

}

}

}

// Boundary fill (4-connected)

public void boundaryFill(int x, int y, int fillColor, int boundaryColor) {

if (x < 0 || x >= width || y < 0 || y >= height)

return;

if (pixels[x][y] == boundaryColor || pixels[x][y] == fillColor)

return;

pixels[x][y] = fillColor;

boundaryFill(x + 1, y, fillColor, boundaryColor);

boundaryFill(x - 1, y, fillColor, boundaryColor);

boundaryFill(x, y + 1, fillColor, boundaryColor);

boundaryFill(x, y - 1, fillColor, boundaryColor);

}

@Override

public void mouseClicked(MouseEvent e) {

int x = e.getX();

int y = e.getY();

boundaryFill(x, y, 2, 1);

repaint();

}

public void mousePressed(MouseEvent e) {}

public void mouseReleased(MouseEvent e) {}

public void mouseEntered(MouseEvent e) {}

public void mouseExited(MouseEvent e) {}

public static void main(String[] args) {

new BoundaryFillDemo();

}

}

**Scanline fill Interactive**

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

public class InteractiveScanlineFill extends Frame implements MouseListener, KeyListener {

ArrayList<Point> points = new ArrayList<>();

boolean fill = false;

public InteractiveScanlineFill() {

super("Interactive Scanline Fill");

setSize(500, 500);

setLocationRelativeTo(null);

addMouseListener(this);

addKeyListener(this);

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

dispose();

System.exit(0);

}

});

setVisible(true);

}

@Override

public void paint(Graphics g) {

if (points.size() > 1) {

// Draw polygon outline

g.setColor(Color.BLACK);

for (int i = 0; i < points.size(); i++) {

Point p1 = points.get(i);

Point p2 = points.get((i + 1) % points.size());

g.drawLine(p1.x, p1.y, p2.x, p2.y);

}

}

if (fill && points.size() >= 3) {

scanlineFill(g);

}

}

private void scanlineFill(Graphics g) {

int n = points.size();

int[] px = new int[n];

int[] py = new int[n];

for (int i = 0; i < n; i++) {

px[i] = points.get(i).x;

py[i] = points.get(i).y;

}

int ymin = py[0], ymax = py[0];

for (int i = 1; i < n; i++) {

if (py[i] < ymin) ymin = py[i];

if (py[i] > ymax) ymax = py[i];

}

g.setColor(Color.ORANGE);

for (int y = ymin; y <= ymax; y++) {

int[] nodes = new int[n];

int nodesCount = 0;

int j = n - 1;

for (int i = 0; i < n; i++) {

if ((py[i] < y && py[j] >= y) || (py[j] < y && py[i] >= y)) {

nodes[nodesCount++] = px[i] + (y - py[i]) \* (px[j] - px[i]) / (py[j] - py[i]);

}

j = i;

}

// sort intersections

for (int i = 0; i < nodesCount - 1; i++) {

for (int k = i + 1; k < nodesCount; k++) {

if (nodes[i] > nodes[k]) {

int temp = nodes[i];

nodes[i] = nodes[k];

nodes[k] = temp;

}

}

}

// draw horizontal lines between pairs of intersections

for (int i = 0; i < nodesCount; i += 2) {

if (i + 1 < nodesCount) {

g.drawLine(nodes[i], y, nodes[i + 1], y);

}

}

}

}

// Mouse: add point on click

@Override

public void mouseClicked(MouseEvent e) {

if (!fill) {

points.add(e.getPoint());

repaint();

}

}

public void mousePressed(MouseEvent e) {}

public void mouseReleased(MouseEvent e) {}

public void mouseEntered(MouseEvent e) {}

public void mouseExited(MouseEvent e) {}

// Key: press F to fill

@Override

public void keyPressed(KeyEvent e) {

if (e.getKeyCode() == KeyEvent.VK\_F && points.size() >= 3) {

fill = true;

repaint();

}

}

public void keyTyped(KeyEvent e) {}

public void keyReleased(KeyEvent e) {}

public static void main(String[] args) {

new InteractiveScanlineFill();

}

}