public static void drawPersonalArt(Turtle turtle) {
    /* I made an AutoHotKey script that (basically) recorded the mouse position whenever I click. So I just traced an image with my mouse, clicking whenever I wanted. */
    List<List<Integer>> points = new ArrayList<List<Integer>>();
    points.add(Arrays.asList(0, 0));
    points.add(Arrays.asList(2, 8));
    points.add(Arrays.asList(2, 8));
    ...  
    drawTurtleAtPoints(turtle, points, 1);
}

public static void drawTurtleAtPoints(Turtle turtle, List<List<Integer>> points, int pointMultiplier) {
    List<Integer> xpoints = new ArrayList<Integer>();
    List<Integer> ypoints = new ArrayList<Integer>();
    // create iterators to go through the list of coordinates
    Iterator<List<Integer>> iterator = points.iterator();
    while (iterator.hasNext()) {
        // the iterator will return a list of two points
        List<Integer> currentPoints = iterator.next();
        // the subiterator will go through each of those two points
        Iterator<Integer> subiterator = currentPoints.iterator();
        xpoints.add(subiterator.next() * pointMultiplier);
        ypoints.add(subiterator.next() * pointMultiplier);
    }
    List<Double> headings = calculateHeadings(xpoints, ypoints);
    List<Integer> distances = calculateLengths(xpoints, ypoints);
    for(int i=0; i<xpoints.size()-1; i++) {
        moveToPoint(turtle, headings.get(i),
                    distances.get(i));
    }
}
/**
 * Personal Art, fractal drawn by recursively iterating the following shape
 * * # * 
 * * # * 
 * * # * 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * Each side is replaced by a copy of itself, akin to the koch curve Increase canvas width, height, and max_iterations to see larger image (according to your screen resolution)
 * @param turtle
 */
public static void drawPersonalArt(Turtle turtle) {
    MIN_SIDE_LENGTH = 3;
    turtle.turn(-90.0);
    int ITERATION_FACTOR = 3;
    int MAX_ITERATIONS = 6;
    int TOTAL_LENGTH = MIN_SIDE_LENGTH * (int)Math.pow(ITERATION_FACTOR, MAX_ITERATIONS);
    turtle.forward(TOTAL_LENGTH / 2);
    turtle.turn(180.0);
    drawFractal(turtle, MAX_ITERATIONS);
}

/**
 * This is a method to take a particular 16x16 image and print it
 * @param turtle used to draw
 * @param image a 16x16 array filled with darkness values for
 * the 'pixels'
 */
public static void drawPersonalArt(Turtle turtle, int[][] image) {
    // get to a good area to draw grid
    turtle.turn(270);
    turtle.forward(250);
    turtle.turn(270);
    turtle.forward(128);
    turtle.turn(180);
    //Drawing the image from 'image'
    int longSide = 256;
    int shortSide = 16;
    drawRegularPolygon(turtle, 4, longSide);
    // the level of darkness
    int value;
    for (int i = 0; i < longSide; i+=shortSide){
        // move along a row
        for (int j = 0; j < longSide; j+=shortSide){
            // choosing a color
            value = image[15-(i/shortSide)][j/shortSide];
            // draw a 'pixel'
            drawDarkSquare(turtle, shortSide, value);
            turtle.turn(90); // turn to the right
            // go one side length
            turtle.forward(shortSide);
            //turn right side up
            turtle.turn(270);
        }
        // move up one column
        turtle.turn(270);
        turtle.forward(longSide);
        turtle.turn(90);
        turtle.forward(shortSide);
    }
}

/**
 * Personal Art, fractal drawn by recursively iterating the following shape
 * * # * 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * * # # 
 * Each side is replaced by a copy of itself, akin to the koch curve Increase canvas width, height, and max_iterations to see larger image (according to your screen resolution)
 * @param turtle
 */
public static void drawPersonalArt(Turtle turtle) {
    MIN_SIDE_LENGTH = 3;
    turtle.turn(-90.0);
    int ITERATION_FACTOR = 3;
    int MAX_ITERATIONS = 6;
    int TOTAL_LENGTH = MIN_SIDE_LENGTH * (int)Math.pow(ITERATION_FACTOR, MAX_ITERATIONS);
    turtle.forward(TOTAL_LENGTH / 2);
    turtle.turn(180.0);
    drawFractal(turtle, MAX_ITERATIONS);
}

static final double[] ANGLES = new double[]{0.0, 120.0, 120.0, 60.0, 120.0, -60.0, 0.0};
static void drawFractal(Turtle turtle, int depth) {
    if (depth <= 0) {
        turtle.forward(MIN_SIDE_LENGTH);
        return;
    }
    depth--;
    for (double d : ANGLES) {
        drawFractal(turtle, depth);
    }
}
public static void drawPersonalArt(Turtle turtle) {
 /** I had a bit of fun with this.
  * Basically just a lot of tinkering, and I found a
  * star :D */
  int sideLength=1;
  double degree=150;
  while (sideLength<500){
    turtle.forward(sideLength);
    turtle.turn(degree);
    turtle.forward(sideLength);
    turtle.turn(degree);
    turtle.forward(sideLength);
    turtle.turn(degree);
    sideLength++;
    sideLength++;
  }
}

public static void drawPersonalArt(Turtle turtle) {
  int step = 20;
  int sides = 7;
  for(int i=0; i<360; i++){
    int length = 100;
    while(length>10){
      drawRegularPolygon(turtle, sides, length);
      length -= step;
    }
    turtle.turn(11);
    step++;
    sides--;
    if (sides<3){
      sides = 7;
    }
  }
}

public static void drawPersonalArt(Turtle turtle) {
  for (int i = 0; i < 150; i++) {
    drawRegularPolygon(turtle, 900, i%3);
    turtle.turn(90);
    turtle.forward(i);
  }
}
public static void drawPersonalArt(Turtle turtle) {
    drawRegularPolygon(turtle, 6, 100);
turtle.turn(60.0);
drawRegularPolygon(turtle, 6, 100);
turtle.turn(60.0);
drawRegularPolygon(turtle, 6, 100);
turtle.turn(60.0);
drawRegularPolygon(turtle, 6, 100);
turtle.turn(60.0);
drawRegularPolygon(turtle, 6, 100);
turtle.turn(60.0);
drawRegularPolygon(turtle, 6, 100);
turtle.turn(30.0);
drawRegularPolygon(turtle, 3, (int)Math.round(100.00*Math.sqrt(3.00)));
turtle.turn(60.0);
drawRegularPolygon(turtle, 3, (int)Math.round(100.00*Math.sqrt(3.00)));
turtle.turn(60.0);
drawRegularPolygon(turtle, 3, (int)Math.round(100.00*Math.sqrt(3.00)));
turtle.turn(60.0);
drawRegularPolygon(turtle, 3, (int)Math.round(100.00*Math.sqrt(3.00)));
turtle.turn(60.0);
drawRegularPolygon(turtle, 3, (int)Math.round(100.00*Math.sqrt(3.00)));
}

public static void drawPersonalArt(Turtle turtle) {
//Entitled "Not Quite A Sierpinski Triangle"
//Initialize the turtle position
turtle.turn(180);
turtle.forward(225);
turtle.turn(180);
int depth = 5; //Note: if you change this, also change it in drawMiniRightTriangle.
int length = 512;
drawMiniRightTriangle(turtle, depth, length, 1);
turtle.forward((int)Math.round(length/Math.sqrt(5)*2));
turtle.turn(180);
drawMiniRightTriangle(turtle, depth, 512, 1);
}

public static void drawMiniRightTriangle(Turtle turtle, int depth, double sideLength, int parity) {
    //It'll be a pinwheel fractal!
double angle = Math.atan2(2, 1) * 180 / Math.PI;
    int shortLength = (int)Math.round(sideLength/5);
    int longLength = (int)Math.round(sideLength*2/5);
    //This is all best explained with an actual drawing.
    //Essentially, it draws four smaller sub-triangles by recursively calling this function
    //with a smaller depth and smaller side-length.
    //Everything else is the nuts and bolts of getting those sub-triangles in the right
    //spot.
turtle.turn(-angle * parity);
turtle.forward(shortLength);
turtle.turn(90*parity);
    if (depth==0) turtle.forward(longLength);
    else drawMiniRightTriangle(turtle, depth-1, sideLength/Math.sqrt(5), -parity);
    turtle.turn(-90*parity);
turtle.forward(shortLength);
turtle.turn(90*parity);
    if (depth==0) turtle.forward(longLength);
    else drawMiniRightTriangle(turtle, depth-1, sideLength/Math.sqrt(5), -parity);
    turtle.turn(180);
turtle.forward(longLength);
    if (depth==0) turtle.forward(longLength);
    else drawMiniRightTriangle(turtle, depth-1, sideLength/Math.sqrt(5), parity);
    turtle.forward(shortLength);
    turtle.turn(180);
turtle.forward(shortLength);
turtle.turn(90 * parity);
    if (depth==0) turtle.forward(longLength);
    else drawMiniRightTriangle(turtle, depth-1, sideLength/Math.sqrt(5), -parity);
    turtle.turn((angle-180)*parity);
    //Due to slight rounding error, because turtle.forward takes an int,
    //the very last line the triangle draws sometimes looks funny at the largest depth (5)
    //This statement gets rid of that for better aesthetic effect.
    if (depth != 5)
        turtle.forward((int)Math.round(sideLength/Math.sqrt(5)*2));
}
```java
public static void drawPersonalArt(Turtle turtle) {
    for (int i = 1; i < 300; i++) {
        drawRegularPolygon(turtle, 3+i%6, i/10);
        turtle.turn(360/(10+(i*.1)));
        turtle.forward((i/5));
    }
}

public static void drawPersonalArt(Turtle turtle) {
    int count = 100;
    while (count > 0) {
        for (int i = 0; i < 720; i++) {
            turtle.forward((int)(Math.PI*Math.pow(3,2)*count/140));
            turtle.turn(i*0.5);
        }
        count--;
        turtle.turn(70);
        for (int i = 0; i < 6; i++) {
            turtle.forward(20);
            turtle.turn(5);
        }
        turtle.forward(20);
        drawTurtle(turtle);
    }
}

private static void drawTurtle(Turtle turtle) {
    turtle.forward(5);
    turtle.turn(30);
    ...}
```
public static void drawPersonalArt(Turtle turtle) {
    ArrayList<Integer> xCoords = new ArrayList<Integer>(Arrays.asList(new Integer[]{0, ... 93}));
    ArrayList<Integer> yCoords = new ArrayList<Integer>(Arrays.asList(new Integer[]{0, ... -123}));
    List<Integer> distances = calculateDistances(xCoords, yCoords);
    List<Double> headings = calculateHeadings(xCoords, yCoords);
    for (int i = 0; i < distances.size(); i++) {
        turtle.turn(headings.get(i));
        turtle.forward(distances.get(i));
    }
}

public static void drawPersonalArt(Turtle turtle) {
    // Snowflake made up of Koch snowflakes
    // experiment with the angle.
    // Using angles that divide 360 evenly will work best
    // e.g 45, 22.5, 60, etc. Going too small increases clutter
    int length=160;
    int depth =4;
    double angle= 72.0;
    int num=(int) Math.floor(360/angle);
    for(int i=0;i<num+1;i++) {
        drawKoch(turtle, length, depth, 45.0);
        turtle.forward(-136);
        drawKoch(turtle,length,depth,-45.0);
        turtle.forward(-136);
        turtle.turn(angle);
    }
}

public static void drawPersonalArt(Turtle turtle) {
    turtle.forward(200);
    turtle.turn(90);
    turtle.forward(-300);
    drawSpirograph(turtle, 600, 0.8);
    turtle.forward(-200);
    drawSpirograph(turtle, 300, 0.8);
    turtle.forward(-400);
    drawSpirograph(turtle, 400, 0.356);
    turtle.forward(400);
    drawSpirograph(turtle, 300, 0.15);
    turtle.turn(125);
    turtle.forward(360);
    drawSpirograph(turtle, 270, 0.245);
    turtle.turn(225);
    turtle.forward(20);
    drawSpirograph(turtle, 140, 0.354);
}
public static void drawPersonalArt(Turtle turtle) {
    for (int n=0; n<25; n++) {
        turtle.forward(20);
        turtle.turn(10);
    }
    for (int n=24; n<35; n++) {
        turtle.forward(20);
        turtle.turn(355);
    }
    turtle.turn(145);
    for (int n=34; n<45; n++) {
        turtle.forward(20);
        turtle.turn(-5);
    }
    for (int n=44; n<71; n++) {
        turtle.forward(20);
        turtle.turn(10);
    }
    turnAndMove(turtle, 75, 150);
    turnAndMove(turtle, 270, 70);
    turnAndMove(turtle, 270, 50);
    ...
17 contd.

```java
//color pixel below
turtle.forward(2);
turtle.turn(180);
turtle.forward(1);
turtle.turn(90);
}
else{
turtle.forward(1);
turtle.turn(90);
}
}
else if(row < h - 1 && Math.random() > lum(img.getRGB(col, row + 1))){
    //color pixel below
    turtle.forward(2);
turtle.turn(180);
turtle.forward(1);
turtle.turn(-90);
} else{
turtle.forward(1);
turtle.turn(90);
}
}
if(row % 2 == 1){
    //if scanning left to right
    for(int col = w; col >= 0; col--){
        if(Math.random() > lum(img.getRGB(col, row - 1))){
            //color pixel above
            turtle.turn(-90);
turtle.forward(1);
turtle.turn(180);
            if(Math.random() > lum(img.getRGB(col, row + 1))){
                //color pixel below
                turtle.forward(2);
turtle.turn(180);
turtle.forward(1);
turtle.turn(-90);
            } else{
turtle.forward(1);
turtle.turn(90);
            }
        }
    }
}
else{
    //scanning right to left
    for(int col = w - 1; col >= 0; col--){
        if(Math.random() > lum(img.getRGB(col, row - 1))){
            //color pixel above
            turtle.turn(90);
turtle.forward(1);
turtle.turn(180);
            if(Math.random() > lum(img.getRGB(col, row + 1))){
                //color pixel below
                turtle.forward(2);
turtle.turn(180);
turtle.forward(1);
turtle.turn(-90);
            } else{
turtle.forward(1);
turtle.turn(90);
            }
        }
    }
}
row += 3;
}
//repeating for last set of rows, with extra logic

//given pixel in hex, returns luminance of pixel as value in [0, 1]
public static double lum(int pixel){
    int r = (pixel >> 16) & 0xff;
    int g = (pixel >> 8) & 0xff;
    int b = pixel & 0xff;
    return (.2126*r + .7152*g + .0722*b)/255;
}
```
private static double dragonAngleCalc(int index) {
    boolean flip = false;
    double temp = Math.pow(2, Math.floor(Math.log(index + 1) / Math.log(2))); // temp is highest power of 2 less than index
    index = 2 * (Math.floor(2 * (temp - 1)) - index); // flips index around temp, adjusting for off by 1
    if (flip) return 270.0; else return 90.0;
}

public static void drawDragonCurve(Turtle turtle, int iterations) {
    final int stepSize = 2718 / ((int) Math.pow(iterations, 2.718));
    final int turns = (int) Math.pow(2, iterations) - 1;
    for (int i = 0; i < turns; ++i) {
        turtle.forward(stepSize);
        turtle.turn(dragonAngleCalc(i));
    }
}

public static void drawPersonalArt(Turtle turtle) {
    final int iterations = 15;
    drawDragonCurve(turtle, iterations); // I'm going to try a dragon curve.
    for (int i = 0; i < steps.size(); i++) {
        if (num == 0) steps.add(1);
        else steps.add(0);
    }
    for (int num : steps) {
        if (num == 0) {
            turtle.forward(5);
            turtle.turn(90);
        } else {
            turtle.forward(5);
            turtle.turn(270);
        }
    }
}