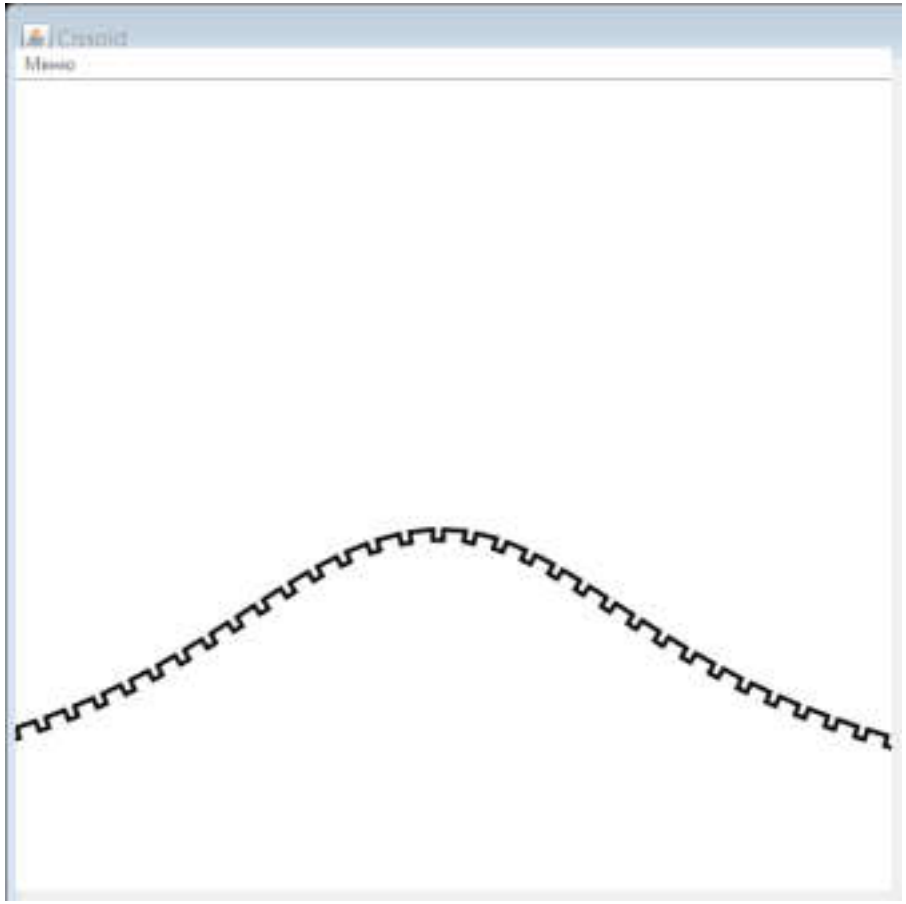

```
package com.company;

import java.awt.*;
import java.awt.geom.AffineTransform;
import java.awt.geom.PathIterator;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

public class Cissoid implements Shape {
    private int centerX;
    private int centerY;
    private int a;
    Cissoid(int centerX, int centerY, int param) {
        this.centerX = centerX;
        this.centerY = centerY;
        this.a = param;
    }

    @Override
    public Rectangle getBounds() {
        return null;
    }

    @Override
    public Rectangle2D getBounds2D() {
        return null;
    }

    @Override
    public boolean contains(double x, double y) {
        return false;
    }
```

```
@Override
public boolean contains(Point2D p) {
    return false;
}

@Override
public boolean intersects(double x, double y, double w, double h) {
    return false;
}

@Override
public boolean intersects(Rectangle2D r) {
    return false;
}

@Override
public boolean contains(double x, double y, double w, double h) {
    return false;
}

@Override
public boolean contains(Rectangle2D r) {
    return false;
}

@Override
public PathIterator getPathIterator(AffineTransform at) {
    return new ShapeIterator(at);
}

@Override
```

```
public PathIterator getPathIterator(AffineTransform at, double flatness) {
    return new ShapeIterator(at);
}

class ShapeIterator implements PathIterator {
    AffineTransform at;
    boolean done = false;
    double dalpha = Math.PI / 40000;
    boolean start = true;
    double alpha = -Math.PI/2+dalpha;

    ShapeIterator(AffineTransform at) {
        this.at = at;
    }

    @Override
    public int getWindingRule() {
        return WIND_NON_ZERO;
    }

    @Override
    public boolean isDone() {
        return done;
    }

    @Override
    public void next() {
        alpha += dalpha;
    }

    @Override
    public int currentSegment(float[] coordinate) {
```

```
    if (start) {
        coordinate[1] = (float) (2 * a * Math.pow(Math.tan(alpha), 2) / (1 + Math.pow(Math.tan(alpha), 2))) + centerX;
        coordinate[0] = (float) (2 * a * Math.pow(Math.tan(alpha), 3) / Math.pow(Math.tan(alpha), 2)) + centerY;
        start = false;
        if (at != null)
            at.transform(coordinate, 0, coordinate, 0, 1);
        return SEG_MOVETO;
    }
    if (alpha >= Math.PI/2-dalpha) {
        done = true;
        return SEG_CLOSE;
    }
    coordinate[1] = (float) (2 * a * Math.pow(Math.tan(alpha), 2) / (1 + Math.pow(Math.tan(alpha), 2))) + centerX;
    coordinate[0] = (float) (2 * a * Math.pow(Math.tan(alpha), 3) / Math.pow(Math.tan(alpha), 2)) + centerY;
    return SEG_LINETO;
}

@Override
public int currentSegment(double[] coordinate) {
    if (start) {
        coordinate[1] = (2 * a * Math.pow(Math.tan(alpha), 2) / (1 + Math.pow(Math.tan(alpha), 2))) + centerX;
        coordinate[0] = (2 * a * Math.pow(Math.tan(alpha), 3) / Math.pow(Math.tan(alpha), 2)) + centerY;
        start = false;
        return SEG_MOVETO;
    }
    if (alpha >= Math.PI/2-dalpha) {
        done = true;
        return SEG_CLOSE;
    }
    coordinate[1] = (2 * a * Math.pow(Math.tan(alpha), 2) / (1 + Math.pow(Math.tan(alpha), 2))) + centerX;
    coordinate[0] = (2 * a * Math.pow(Math.tan(alpha), 3) / Math.pow(Math.tan(alpha), 2)) + centerY;
    return SEG_LINETO;
```