```
// This program is copyright VUW.
// You are granted permission to use it to construct your answer to a COMP102 assignment.
// You may not distribute it in any other way without permission.

/* Code for COMP102 - 2024T3, Assignment 3
 * Name:
 * Username:
 * ID:
 */

import ecs100.*;
import java.awt.Color;
import java.lang.Math;

/** PatternsDrawer:
 * Draws four different repetitive patterns. */

public class PatternsDrawer{

    public static final double PATTERN_LEFT = 50.0;   // Left side of the pattern
    public static final double PATTERN_TOP = 50.0;    // Top of the pattern
    public static final double PATTERN_SIZE = 300.0;  // The size of the pattern on the window

    /**
     * Draws a star pattern consisting of a circle containing black rays (separated by white
regions)
     * Asks the user for the number of rays, and works out the angle of the rays.
     * Hint: use filled arcs to draw the rays,
     */
    public void drawStar(){
        UI.clearGraphics();
        UI.setColor(Color.black);
        double num = UI.askInt("How many rays:");
        /*# YOUR CODE HERE */
        UI.drawOval(PATTERN_LEFT,PATTERN_TOP,PATTERN_SIZE,PATTERN_SIZE);
        for (double i=0;i<=360;i+=360/num){
            UI.fillArc(PATTERN_LEFT,PATTERN_TOP,PATTERN_SIZE,PATTERN_SIZE,i,360/(num*2));
        }

    }

    /**
     * drawTriGrid draws a triangular grid of squares (all the same size)
     * that makes dark circles appear in the intersections when you look at it.
     * The gap between the squares should be 1/3 the size of the squares.
     * It asks the user for the number of rows.
     * The top row should have one square; the next row should have two squares;
     * and so on. The last row will have as many squares as there are rows.
     */
    public void drawTriGrid(){
        UI.clearGraphics();
        UI.setColor(Color.black);
        double num = UI.askInt("How many rows:");

        double step = PATTERN_SIZE / num ;
        double squaresize = step / 4 * 3;

        /*# YOUR CODE HERE */
        for(double i=0;i <num ;i++){
            for (double j=0; j<=i ;j++){
                UI.fillRect(PATTERN_LEFT + j * step, PATTERN_TOP + (i * step
),squaresize,squaresize);

            }
        }
    }
    /**
     * Draws a rectangle containing a row of random sized blue circles
     * The width of the rectangle is PATTERN_SIZE.
```

```
     * The method asks user for the height of the rectangle, which should be
     * less than the width: the method should keep asking for a height until
     * the user gives a height less than the width.
     * It then draws the rectangle and fills it from left to right with
     * random sized circles (size is between 3 units and the height)
     *
     * Hint: use a while loop to ask for the height,
     * Hint: use another while loop to work out the size and draw each circle,
     *  stopping when the next circle won't fit.
     *  At the end, if there is still room, draw the right sized circle to fill the gap.
     */
    public void drawRandomCircles(){
        /*# YOUR CODE HERE */

        double height = PATTERN_SIZE;
        double diameter = 0;
        double limit = PATTERN_LEFT + PATTERN_SIZE;

        while (height >= PATTERN_SIZE) {
            height = UI.askDouble("Height ? ");

            // If height is not less than width, ask again
            if (height >= PATTERN_SIZE) {
                UI.println("Height must be less than width. Please try again.");
            }
        }

        UI.drawRect(PATTERN_LEFT,PATTERN_TOP,PATTERN_SIZE,height);
        UI.setColor(Color.blue);
        double one = 0 ;

        while(diameter < PATTERN_SIZE){


            double size = Math.random() * height;

            one = PATTERN_SIZE - diameter;

            if ( PATTERN_LEFT + diameter > PATTERN_SIZE  ){

                break;
            }

            UI.fillOval(PATTERN_LEFT + diameter,PATTERN_TOP + (height /2 - size/2),size,size);
            diameter += size;


        }
        UI.fillOval(PATTERN_LEFT + diameter,PATTERN_TOP + (height /2 - one/2),one,one);



    }




    /**
     * Draws a square spiral pathway made of little circles.
     *  Asks the user for the size of the circles
     *   and computes the number of circles needed to fit the PATTERN_SIZE
     *  Then draws all the circles from the outside to the inside.
     *  The gap between the legs of the spiral should be the width of the circles
     */
    public void drawSpiralPath(){
        /*# YOUR CODE HERE */
        // Ask the user for the size of the circles
        String input = UI.askString("Enter the size of the circles (diameter):");
```

```
        int circleSize = Integer.parseInt(input);

        // Constants for the pattern area
        final double PATTERN_LEFT = 50.0;   // Left side of the pattern
        final double PATTERN_TOP = 50.0;    // Top of the pattern
        final double PATTERN_SIZE = 300.0;  // The size of the spiral's square area

        // Calculate the number of circles that can fit into the PATTERN_SIZE area
        int numCircles = (int) (PATTERN_SIZE / circleSize);

        // Starting coordinates: adjust to fit the specified left and top padding
        double x = PATTERN_LEFT + PATTERN_SIZE / 2;
        double y = PATTERN_TOP + PATTERN_SIZE / 2;

        // The movement direction for the spiral: right, down, left, up
        double dx = circleSize; // Start by moving right
        double dy = 0;

        int step = 1; // Step counter to switch directions
    int currentDirection = 0; // 0 - right, 1 - down, 2 - left, 3 - up
    int currentStep = 0; // To track steps in each direction (right, down, etc.)

    // Loop to draw circles in a spiral pattern
    for (int i = 0; i < numCircles; i++) {
        // Draw a circle at the current position
        UI.fillOval(x - circleSize / 2, y - circleSize / 2, circleSize, circleSize);

        // Update position for the next circle
        x += dx;
        y += dy;
        currentStep++;

         // Check if we need to change direction
        if (currentStep == step) {
            currentStep = 0;
            // Turn 90 degrees (right -> down -> left -> up)
            switch (currentDirection) {
                case 0: // Right
                    dx = 0;
                    dy = circleSize;
                    break;
                case 1: // Down
                    dx = -circleSize;
                    dy = 0;
                    break;
                case 2: // Left
                    dx = 0;
                    dy = -circleSize;
                    break;
                case 3: // Up
                    dx = circleSize;
                    dy = 0;
                    break;
            }

            // Update the direction
            currentDirection = (currentDirection + 1) % 4;

            // After two direction changes (right-down-left-up), increase the step size
            if (currentDirection == 0 || currentDirection == 2) {
                step++;
            }
        }
    }
```

```
    }

    /**
     * Set up the GUI and buttons
     */
    public void setupGUI(){
        UI.initialise();
        UI.addButton("Clear",UI::clearPanes);
        UI.addButton("Core: Star", this::drawStar);
        UI.addButton("Core: TriGrid", this::drawTriGrid);
        UI.addButton("Completion: Random", this::drawRandomCircles);
        UI.addButton("Challenge: Spiral", this:: drawSpiralPath);
        UI.addButton("Quit", UI::quit);
    }

    /**
     * main: create object and call setupGUI
     */
    public static void main(String[] arguments){
        PatternsDrawer pd = new PatternsDrawer();
        pd.setupGUI();
    }

}
```