Directions

Use the `Turtle` package to complete tasks from the four sections described below below to earn points. For each of the tasks you complete, you should define a method that draws the described shape. In order to see the shape drawn, you will have to *call* your function in the `main` method. For example, if you write `drawHexagon`, you need to call it in `main` to actually see it:

```
1 void main() {
2   drawHexagon();
3 }
4
5 void drawHexagon() {
6   // Your code here
7 }
```

To earn full credit for a drawing, your code should create these drawings efficiently, i.e. using loops instead of repeated code.

You may earn extra credit by completing additional problems but you will only earn *1 extra credit point* for every *4 points* you earn above the required amount of points for each section.

Tips

- You can set how fast the turtle moves by adding a call to the `speed` function to the beginning of your `main` method. This determines the delay between the turtle's movements. Calling `speed(0)` makes the turtle move as fast as possible.
- While working on a particular drawing, make sure it is the only function called in `main` so you don't have to wait for other drawings to finish.

Rubric

Criteria	Points
Section A Tasks *Using loops to repeat code.*	6
Section B Tasks *Using abstraction in the form of subprocedures.*	6
Section C Tasks *Using parametrized subprocedures.*	6
Section D *Create your own masterpiece using parametrized subprocedures.*	4
Total	22

Turtle Methods

- `forward(double distance)`: moves the turtle forward by `distance`.
- `wait(int time)`: pauses the program for `time` milliseconds.
- `speed(double delay)`: sets the speed of the animation, where `delay` is the milliseconds it takes to do one action.
- `forward(double distance)`: moves the turtle forward by `distance`.
- `backward(double distance)`: moves the turtle backward by `distance`.
- `left(double angle)`: turns the turtle left by `angle` degrees.
- `right(double angle)`: turns the turtle right by `angle` degrees.
- `getDirection()`: returns the direction the turtle is facing (ignoring tilt).
- `setDirection(double direction)`: sets the turtle's direction, where `direction` is the angle counter-clockwise from east.
- `home()`: moves the turtle to (0,0) and sets it facing east.
- `face(double x, double y)`: sets the turtle to face towards the coordinates `(x, y)`.
- `towards(double x, double y)`: returns the direction towards `(x, y)` from the turtle's current position.
- `distance(double x, double y)`: returns the distance from the turtle's current position to `(x, y)`.
- `setPosition(double x, double y, double direction)`: sets the turtle's position to `(x, y)` and its direction to `direction`.
- `setPosition(double x, double y)`: sets the turtle's position to `(x, y)`.
- `width(double penWidth)`: sets the width of the turtle's path to `penWidth`.
- `up()`: lifts the turtle's tail so it won't draw while moving.
- `down()`: puts the turtle's tail down so it will draw while moving.
- `penColor(String penColor)`: sets the turtle's path color to `penColor`.
- `stamp()`: leaves a copy of the turtle's current shape on the canvas.
- `dot()`: places a dot three times the size of the turtle's pen width on the canvas.
- `dot(String color)`: places a dot three times the size of the turtle's pen width, in the specified `color`.
- `clear()`: clears all the turtle's drawings, but keeps the turtle's settings (color, location, direction, shape).
- `reset()`: waits for 2 seconds, clears the screen, and resets the turtle's position to the home location.

Submitting

When you are ready to submit your code, you *must* organize your `TurtleShapes.java` file as follows:

1. Create four `void` methods just below the `main` method: `sectionA()`, `sectionB()`, `sectionC()`, and `sectionD()`.
2. Call `sectionA` in the `main` method. I will add in the others when grading (you should test that they all work first!).
3. In each of the "`section`" functions you should call each the methods for each drawing you completed for that section.
4. After each drawing, the `reset` method to pause your program for `2` seconds, clear the screen, and reset the turtle.
5. At the beginning of `sectionA` set the speed of the turtle using `speed(100)`. Add `speed(0)` at the beginning of the `sectionB`, `sectionC`, and `sectionD` methods.

Here is an example:

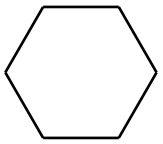
```
1  void main() {
2    sectionA();
3  }
4
5  void sectionA() {
6    speed(100);
7    drawHexagon();
8    reset();
9    drawCircle();
10   reset();
11   // ...
12 }
13
14 void sectionB() {
15   speed(0);
16   drawCheckerboard();
17   reset();
18   // ...
19 }
20
21 void sectionC() {
22   // ...
23 }
24
25 void sectionD() {
26   // ...
27 }
28
29 // Actual draw methods below here
30
31 // Section A
32
33 void drawHexagon() {
34   // ...
35 }
36
37 void drawDodecagon() {
38   // ...
39 }
40
41 // ...
42
43 // Section B
44
45 void drawCheckerboard() {
46   // ...
47 }
48
49 // Remaining sections and drawings
   below here
50 // ...
```

Section A

Complete at least 6 points worth of tasks from this section.

1. `drawHexagon()`: Draws a regular hexagon.

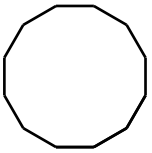
1 PT MILD



Hexagon

2. `drawDodecagon()`: Draws a regular dodecagon. That is, a twelve-sided, regular polygon.

1 PT MILD



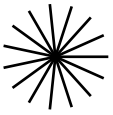
Dodecagon

3. `drawCircle()`: Draws an approximation of a circle.

1 PT MILD

4. `drawStarburst(int n)`: Takes an `int` argument `n` and draws the "starburst" shape with `n` "spokes" as depcited alongside.

1 PT MED



Starburst

5. `drawInitials()`: Draws your initials. Neatness matters!

1 PT MED

6. `drawStar()`: Draws a 5-pointed-star as depcited alongside.

1 PT MED

5-pointed-star

7. `drawPolygon(int n)`: Takes an `int` argument `n` and draws a regular polygon with `n` sides.

1 PT MED

8. `drawHeart()`: Draws a heart, as depcited alongside. It does not need to match the sample exactly.

1 PT SPICY

Heart

9. `drawSpiral()`: Draws a spiral, as depicted alongside. It does not need to be an exact match to the sample image. *Hint: it's similar to drawing a circle, but you need to change something as you draw.*

1 PT SPICY

Spiral

10. `drawCircle(double radius)`: Takes a `double` argument `radius` and draws a circle with the given radius.

1 PT SPICY

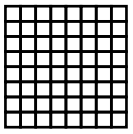
Section B

Complete at least 6 points worth of tasks from this section.

For this section, you must make use of *abstraction* and *subprocedures*. That is, your code for these drawings should not be contained in a single method. For example, `drawCheckerboard` should call another method, maybe something like `drawRow` or `drawSquare`.

1. `drawCheckerboard()`: Draws an 8×8 checkerboard. The squares do not need to be colored in.

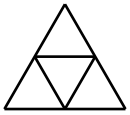
2 PT MED



Checkerboard

2. `drawTriforce()`: Draws a triforce made of multiple triangles.

2 PT MED



Triforce

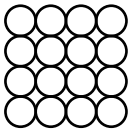
3. `drawCircleOfShapes()`: Draws a circle of shapes. You can use circles, squares, triangles, or any shape you would like.

2 PT MED

Shape circle

4. `drawCircleGrid()`: Draws a 4×4 grid of circles. The challenge here comes from making the drawing neat, without extra lines.

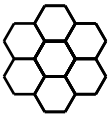
2 PT SPICY



Grid of circles

5. `drawHoneycomb()`: Draws a honeycomb made out of hexagons.

2 PT SPICY



Honeycomb

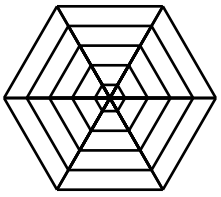
Section C

Complete at least 6 points worth of tasks from this section.

For this section, you must make use of *parameterized subprocedures* when creating these drawings. As in Section B, your code for drawing these shapes should not be contained in a single

1. `drawSpiderweb()`: Draws a spiderweb shape (*we'll do this one together*).

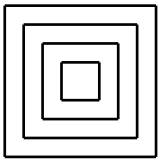
2 PT MED



Spiderweb

2. `drawNestedSquares()`: Draws a series of nested squares. The squares should be centered inside one another.

3 PT MED



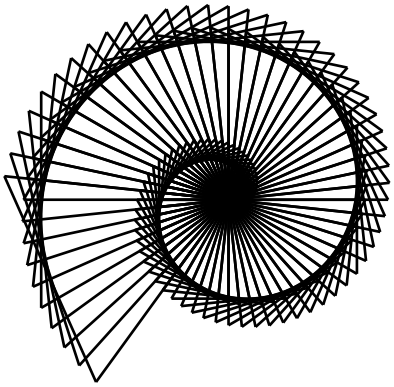
Nested squares

3. `drawShapeSpiral()`: Draws the spiral of shapes shown alongside.

3 PT MED

4. `drawSnowflake()`: Draws the snowflake as shown alongside.

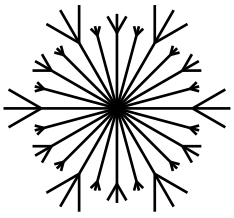
3 PT SPICY



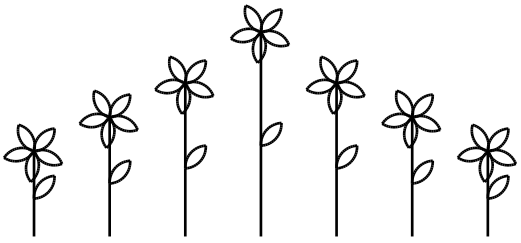
Another spiral

5. `drawFlowers()`: Draws the garden of flowers as seen alongside.

3 PT SPICY



Snowflake



Flower garden

Section D: Your Masterpiece

This section is worth 4 points. You will create your own drawing for this section.

1. `drawMasterpiece()`: Draws your own masterpiece. You can draw whatever you would like, but it must require the use of parameterized subprocedures as in section C. The complexity of your drawing should be a bit more complex than the drawings you did for section C.

4 PT