Specification

📖 Background

By writing parameterized methods, your code can become much more flexible. For instance, you could make a method that draws a square with sides of length 100. However, if you want to draw a square with sides of length 200 you would have to write a new method or edit your previous one.

You could instead write a method that takes some parameter `int length`, which denotes the length of the sides. By using this parameter inside your method, you can use one block of code to draw a square of any size!

Here is the code for the parameterized method to draw a square of any size using a `Turtle`:

```
public static void drawSquare(Turtle t, int length) { for (int i = 0; i < 4; i++) { t.forward(length); t.left(90); } }
```

1️⃣ Task 1 - Parameterizing Shapes

For this task, you will implement 2 parameterized methods to draw shapes of given sizes using `Turtles`.

1. The first method should **draw a rectangle of a given width and height.** The method header should be as follows:

```
public static void drawRectangle(Turtle t, int width, int height) {
```

You **must** use a `for` loop to draw the rectangle!

2. The second method should **draw any shape of your choosing, but with at least one parameterized dimension.** The only limitation to your choice is that you cannot re-implement `drawSquare(...)` shown above, or implement another method that draws a rectangle.

Hint: This method should have a similar header to `drawRectangle()`

You **must** use a `for` loop to draw your shape!

You **must** call `drawRectangle()` and the method that draws the shape of your choosing **from the main method**.

2️⃣ Task 2 - More Fun with Parameters!

For the second task of this assignment, you need to choose **two** of the following options to implement in your program.

Option 1 - Random Regular Shapes

For this application you will implement a **method that draws a parameterized number of regular shapes that have randomly chosen edge lengths** (a regular shape is a shape with equal side lengths and equal angles). The maximum size of the edges will be bounded by a `maxSize` parameter. You should use the same randomly chosen length for each edge of

one regular shape, and then choose a new random side length before drawing the next regular shape. Note that you should be drawing the same regular shape, just with different edge lengths! Each length should fall in between 1 and `maxSize` inclusive. You can use the `nextInt(int limit)` method in the `Random` class to generate these lengths.

You can choose the behavior of your `Turtle` in between drawing each regular shape. For example, your `Turtle` could draw the shapes on top of each other, change its starting angle each time, or move to a new place on the canvas each time it goes to draw a shape. Each of these choices will produce an interesting result in its own way, so feel free to experiment with your choice and see what fun outputs you can produce!

The method header should be as follows:

```
public static void randomRegularShapes(Turtle t, Random rand, int numShapes, int maxSize) {
```

The parameters are used as follows:

- `Turtle t`
    - Used to draw the random-sized regular shape
- `Random rand`
    - Used to generate the random numbers used in the output
- `int numShapes`
    - The number of regular shapes this method should draw
- `int maxSize`
    - The maximum size for any edge of a randomly generated regular shape. Shapes should be generated with an edge size between 1 and `maxSize` inclusive.

> You may add more parameters if you would like, but you must have the four parameters above!

The angle that the Turtle should turn after each side can be found using the following formula:

```
double angle = 360.0 / numSides
```

where `numSides` should be replaced by the number of sides that your regular shape has.

Click Expand to see examples from previous students!

Expand

> **If you choose this option, make sure to call this method in the main method**.

Option 2 - Random Changes [*requires conditionals*]

For this application you will implement a **method that draws a shape with random perturbations in its edges**. This method will draw a shape with a parameterized edge size, angle between edges, and number of edges drawn. At each step, it will choose to add or subtract a randomly chosen value from the angle that it turns. The resulting shape will be the result of all of these small random changes.

The method header of this method would be as follows:

```
public static void randomChanges(Turtle t, Random rand,
int numEdges, double maxChange, int angle, int
sideLength) {
```

The parameters are used as follows:

- `Turtle t`
  - Used to draw the random shape
- `Random rand`
  - Used to create random numbers that are used in the output
- `int numEdges`
  - The number of random moves that this method should make
- `double maxChange`
  - The maximum change in the angle at each move
- `int angle`
  - The angle to turn by each move
- `int sideLength`
  - The length of each side of the shape

> You may add more parameters if you would like, but you must have the six parameters above!

Your `maxChange` variable cannot be larger than half the size of the `angle` you provide.

In order to randomly generate the change in the angle on each move, you should use the following formula:

```
double randomChange = maxChange * rand.nextDouble();
```

The `nextDouble()` method of the `Random` class will generate a number between 0 and 1.0. By multiplying this number by `maxChange`, we will

generate a random double between 0 and `maxChange`. You must *randomly* choose to either add or subtract this `randomChange` from your `angle` at each turn (use of conditionals). Importantly, this method should only turn the `Turtle` left.

Click Expand to see examples from previous students!

Expand

> **If you choose this option, make sure to call this method in the main method**.

Option 3 - Draw Grid

For this option, you should write a **method that draws a grid of a particular shape**. This method will draw a shape with a parameterized number of rows, number of columns, and shift. The shift is the separation distance between the shape you are drawing in each row and column. You should only pass a positive number of rows, columns, and shift.

The method header should be as follows:

```
public static void drawGrid(Turtle t, int rows, int columns, int shift) {
```

The parameters are used as follows:

- `Turtle t`
    - Used to draw the shape
- `int rows`
    - The number of rows to draw

- `int columns`
    - The number of columns to draw
- `int shift`
    - The distance between each shape in the grid

You may add more parameters if you would like, but you must have the four parameters above!

In order to reset the Turtle's position for each row, you should use the following formula:

```
t.setPosition(0, i * shift);
```

where `i` is the current row. The first parameter (`0`) is the X coordinate and the second parameter (`i * shift`) is the y coordinate. The formula above will reset the Turtle to the origin and then move the Turtle up by `shift`. You are welcome to set the first parameter in `setPosition` to be any value.

Here is some pseudocode for the method body to help get you started:

```
for each row: reset the Turtle's position (using setPosition from above) for each column: draw the shape move the Turtle forward by shift
```

Click Expand to see example grid drawings!

Expand

If you choose this option, make sure to call this method in the main method.

Option 4 - Draw n-gon

For the final option for Task 2, you must implement a **method that draws a regular polygon with a parameterized number of sides and length for these sides**. As the shape is regular, each side will have the same length.

Examples of regular polygons are the equilateral triangle, square, pentagon, hexagon, etc.

This method must have the following header:

```
public static void drawNgon(Turtle t, int numSides, int sideLength) {
```

The parameters are used as follows:

- `Turtle t`
  - Used to draw the shape
- `int numSides`
  - The number of sides to draw. If this value is 5, then a pentagon should be drawn for example.
- `int sideLength`
  - The length of each side to draw

> You may add more parameters if you would like, but you must have the three parameters above!

The angle that the Turtle should turn after each side can be found using the following formula:

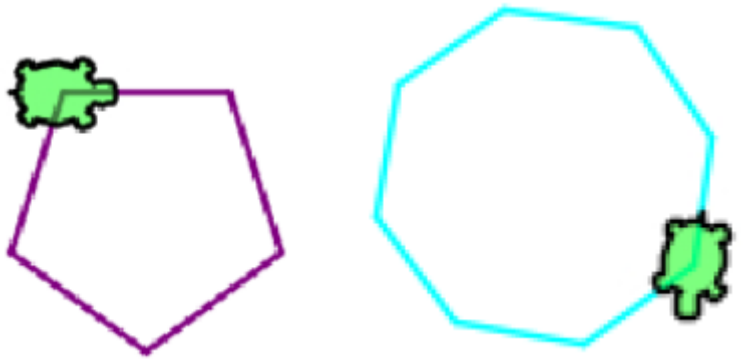
```
double angle = 360.0 / numSides
```

For example - using this formula, the angle that the Turtle would need to turn each time to make a pentagon would be:

```
360.0 / 5 = 72.0
```

Click Expand to see example n-gons from previous students!

Expand



If you choose this option, make sure to call this method in the main method.

Option 5 - Choose your own adventure!

This must involve innovation beyond repeating something you were already asked to do in the concrete exercise. If you want to get credit for this option, describe your idea on this discussion board thread to get approval no later than **11:59pm on Sunday, February 5**. Approval means that your idea, *if executed well*, can earn credit for this option -- it does not guarantee any particular grade.

3️⃣ Application - Create a Picture

By parameterizing methods to draw shapes of various sizes, you can create pictures!

For this final task, **you need to create a method that draws a picture composed of <u>three</u> or more different method calls**.

- These methods can be from Task 1, Task 2, or by creating more parameterized methods! We encourage you to think outside the box to produce some unique outputs!
- You must call at least three different methods.
- You can also choose to parameterize other aspects of the shape, such as the color of the lines drawn, or the number of times the shape is drawn.
- You must pass `Turtle` as a parameter and are welcome to pass more as needed to draw your picture.

A "picture" can just about be anything - you can choose to draw an object, a shape, etc. As long as your method contains three method calls to draw something, you should be good to go!

The following is an example picture to draw a snowman:

```
public static void drawSnowman(Turtle t) { drawFace(t, ...); drawBody(t, ...); drawButtons(t, ...); }
```

> **Make sure to call this method in the main method**.

Click Expand to see some other examples of pictures from previous students!

Expand

> Please note that although we encourage you to produce cool drawings, we are not assessing your artistic talent or your design abilities🎨! In fact, we encourage you to put your main focus of attention to the **quality of your code** rather than the quality of your drawings.

⏰ Recap

Required Method Calls

In the main method of your program, we expect to have these method calls:

1. 2 method calls for Task 1:
    - 1 method call for the `drawRectangle` method
    - 1 method call for your second method that draws any shape of your choosing
2. 2 method calls for Task 2:
    - 1 method call for your first Option choice
    - 1 method call for your second Option choice
3. 1 method call to draw your picture for Task 3

In total, you are expected to have at **least 5** method calls in the main method as specified above.