EECS 2030: Programming Exam 1
Winter 2021
Deadline: Wednesday Feb 24th, 2021 11:59pm

In the first programming exam, we are going to assess your ability for the following skills:

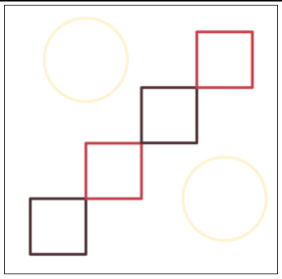
- Ability to design recursive functions.
- Ability to test your code thoroughly, through Junit tests.
- Ability to document your code.

Setting up the environment:

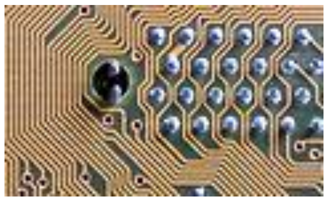
- ❖ Download the attached `ProgrammingExam1.jar` and `PE1.jar`.
- ❖ Create a java project in eclipse. It does not matter what is the name of the project.
- ❖ Right click on the name of the project and select 'import'
- ❖ From 'General' choose 'Archive file' and click next.
- ❖ Click 'Browse' to select the `ProgrammingExam1.jar` file that you have already download.
- ❖ Click 'Finish'.
- ❖ If all the steps were done correctly you should see `ProgrammingExam1` under `src` and `Junit5` under the project. In case `ProgrammingExam1`was not under `src`, drag and drop it to the `src` folder.
- ❖ Right click on the projected and select `Build Path` -> `Configure Build Path`. Click on `Libraries` tab and select `Add External Jars`. Select `PE1.jar -> open -> Apply and Close`.
- ❖ In the code, javaDoc comments have been provided. If you need to see the HTML version of it, make it as instructed in the lecture.

Before you start, we, the instructors, would like to remind you that this assignment is different from the labs that you have done, as it is accounted as an exam that you take home. This means you can neither discuss your solution with anybody including your peers nor get help from them. So please have a look at the statement at the top of `PE1.java` before you start doing the assignment. When you are finished you need to sign this statement by writing your information (full name and student number) in the gap provided. Please be aware that your assignment WILL NOT be marked if you do not sign the declaration and that your code will be checked by both a plagiarism detection tool and us to make sure your solution to this problem is not similar to others.

Problem Description:
There is a robot on an assembly line whose job is to take a plate of different materials (i.e. metal, plastic, wood, etc.) and drill a number of holes in it at specific (x, y) coordinates, or draw a geometric shapes on the plate. These plates are later used for a variety of applications.
As a very simple example, to make a mold for making chocolate boxes, the robot has drawn the following shapes, where the brown, red and cream color shapes are used to place dark, strawberry and chocolate respectively.



As a more complex example, the robot is commanded to draw and drill an electronic circuit board to show where resistors, capacitors etc. should be soldered. See the image below that was taken from Wikimedia as an example.



In this assignment, you are going to write recursive functions that command this robot to draw different shapes or drill at different coordinates in a plate.

How to Draw?

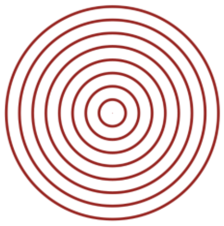
First, let's see a couple of examples of how you can draw different shapes. In the starter code, we have provided a complete documentation of how you can draw a point, line, square and circle. Play with these methods by changing their input parameters and see how they work.
You can also change the pen color in case it helps you in debugging the code.
Please note that we do not execute your `main()` method to check for the correctness of your code. Instead, we run Junit tests that call your method and check the output. This means you must stick to the format that is provided for you. (i.e. you do not change the input parameters of the method or methods' name etc.)

Task 0:

This task has no marks but without it, the rest of your code would not work. You should complete class Point, by implementing a constructor that initializes the `x` and `y` coordinate of a point.
Then you should implement the `midpoint()` method that returns a `Point`, which is in the middle of two given points. The coordinates of the middle point should be cut to two decimal points. e.g. 0.38764 should be converted to 0.38.

Task 1: warm-up

To start, you are required to implement the method called `nestedCircle()` recursively, which gets 6 input parameters and return a string. This method keeps drawing circles with the same center and different radius until the radius gets negative.



The output of the method is a list of all the circles' radius from the most outer to the most inner one. Please see the javaDoc of this method for more information.

Sample Call:

```
PE1.nestedCircle (0.5, 0.5, 0.4, 0.05, blankCanvas, "")
```

Corresponding Output:

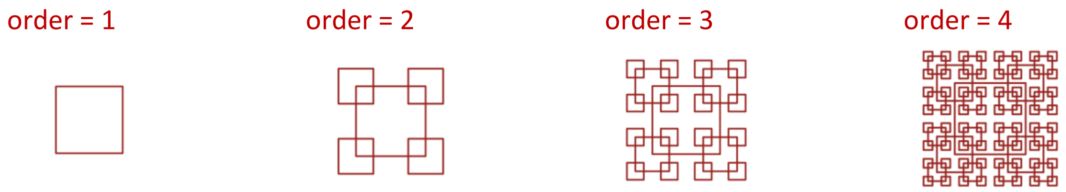
```
[0.4, 0.35, 0.3, 0.25, 0.2, 0.15, 0.1, 0.05, 0.0]
```

Marking Scheme:

- [10 points]: For the correctness of the method. Please note that you will not receive any points if your function is not recursive. Also, make sure that you test your code thoroughly, as we have only provided a subset of the test cases by which we test your code.
- [5 points]: For inner documentation (i.e. the comments that you write inside the method)

Task 2:

For this task, you are required to implement a recursive method that draws a set of squares where the center of one square falls on one corner of another square. The number of squares that should be drawn is given as an input parameter called `order`. If the order is one, then only one square is drawn. If the order is two, then 5 squares will be drawn when on each corner of the central square another square is drawn, whose side length is half the central square's side length. Please see the picture below that shows the drawing for order = 1 to 4.
The output of this method is a list of the coordinates of the smallest squares that are drawn. More information on this method can be found in the javaDoc.



Sample Call:

```
PE1.squares(0.5, 0.5, 0.2, 2, blankCanvas);
```

Corresponding Output:

```
[0.7, 0.7][0.7, 0.3][0.3, 0.7][0.3, 0.3]
```

Marking Scheme:

- [20 points]: For the correctness of the method. Please note that you will not receive any points if your function is not recursive. Also, make sure that you test your code thoroughly, as we have only provided a subset of the test cases by which we test your code.

- [5 points]: For inner documentation.

Task 3:

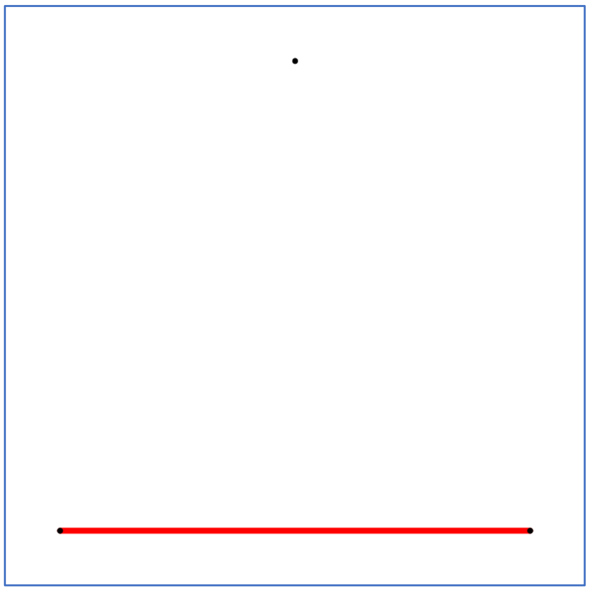
For this task, we are going to have nested triangles. The robot wants to drill a hole on the vertices of triangles and draw a line for the horizontal side of the triangles.
This method starts with three points, which are the vertices of the largest (the most outer) triangle.
A nested triangle is a triangle, whose vertices fall on the midpoints of the sides of the outer triangle.

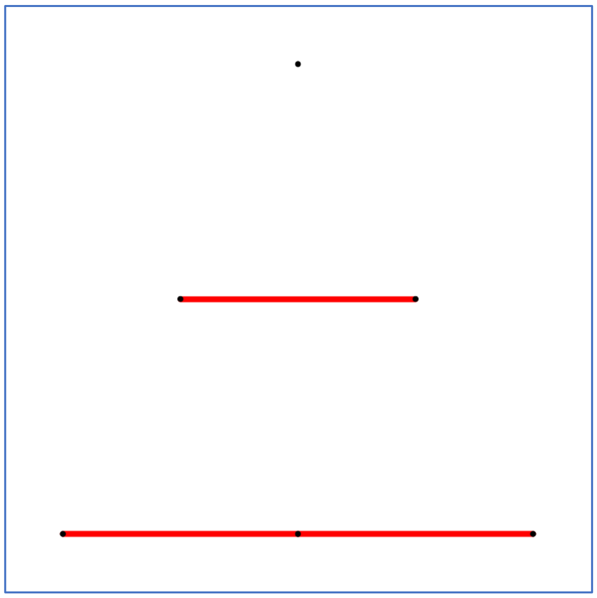
The method gets an integer called `order`, that specifies how deeply the recursive method should be called. If order = 1, only one horizontal line and three points are drawn, and three points are returned as the coordinates that should be drilled by the robot. Please see the pictures below where nested triangles with different orders are created by the robot.

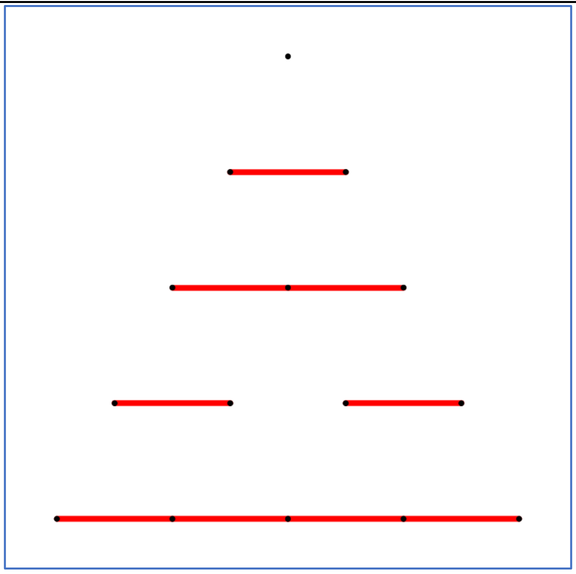
To make it more visible, the points that should be drilled and returned from the method are drawn in black. You don't need to change the color as we only check the return value of the method.

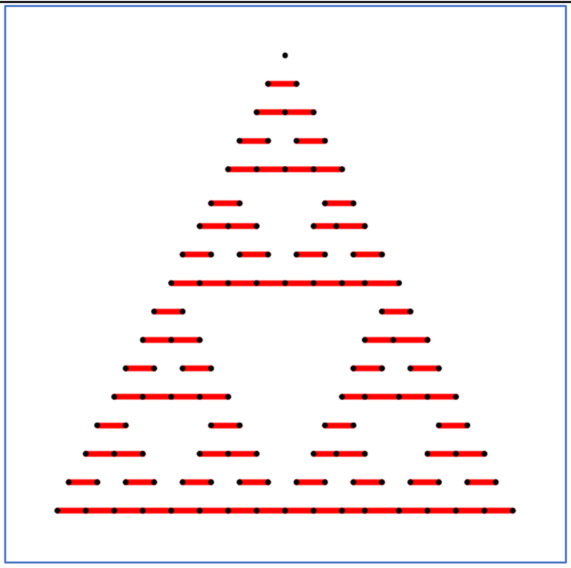
Since this method is recursive, it is possible that a drilled point (i.e. a triangle vertex) is computed more than once. The `ArrayList` that holds the points should not contain any duplicate points.

Also, for the ease of testing, we ask you to add `point.toString()` to the ArrayList. This method can be found in `Point` class.


order = 1 (one line and 3 points are drawn).


order= 2 (two lines and 6 points are drawn)


order= 3 (5 lines and 15 points are drawn)


order = 5

Sample Call 1:

```
     Point p1 = new Point(0.1, 0.1);
    Point p2 = new Point(0.9, 0.1);
    Point p3 = new Point(0.5, 0.9);

    ArrayList<String> points = new ArrayList<String>();
    points = PE1.drillPoints(p1, p2, p3, 1, blankCanvas, points);
```

Corresponding Output:

```
[0.1, 0.1][0.9, 0.1][0.5, 0.9]
```

Sample Call 2:

```
     Point p1 = new Point(0.1, 0.1);
    Point p2 = new Point(0.9, 0.1);
    Point p3 = new Point(0.5, 0.9);

    ArrayList<String> points = new ArrayList<String>();
    points = PE1.drillPoints(p1, p2, p3, 2, blankCanvas, points);
```

Corresponding Output:

```
[0.1, 0.1][0.5, 0.1][0.3, 0.5][0.9, 0.1][0.7, 0.5][0.5, 0.9]
```

Marking Scheme:

- [40 points]: for the correctness of the method. Please note that you will not receive any points if your function is not recursive. Also, make sure that you test your code thoroughly, as we have only provided a subset of the test cases by which we test your code.
- [5 points]: for inner documentation.

Marking Scheme:

- [15 points]: Task 1; as explained before.
- [25 points]: Task2; as explained before.
- [45 points]: Task 3; as explained before.
- [15 points]: code style; Initially you get this 15 points. For every deviation from code style conventions, we deduct 3 points up to 15 points. The convention includes correct indentation, meaningful variable names, and so on.

Checklist:

Before you submit, check this checklist to ensure that you have done all we asked you to do.

- I have finished task 1, 2 and 3.
- I have signed the statement on academic honesty.
- I have removed/commented out the `main()` method to ensure that I have tested my method with the sample tests provided and my methods signature conforms with the tester's requirement.

What to Submit:

- you only submit one file, which is the completed PE1.java file.

How to Submit:

- Submit your solution in eClass, where you have downloaded these instructions.