Task Description

Fill a collection with several regular shapes (circle, regular triangle, square, regular hexagon). Determine the smallest bounding box, which contains all the shapes, and its sides parallel with an x or y axis. Each shape can be represented by its center and side length (or radius), if we assume that one side of the polygons are parallel with x axis, and its nodes lies on or above this side. Load and create the shapes from a text file. The first line of the file contains the number of the shapes, and each following line contains a shape. The first character will identify the type of the shape, which is followed by the center coordinate and the side length or radius. Manage the shapes uniformly, so derive them from the same super class.

Plan:

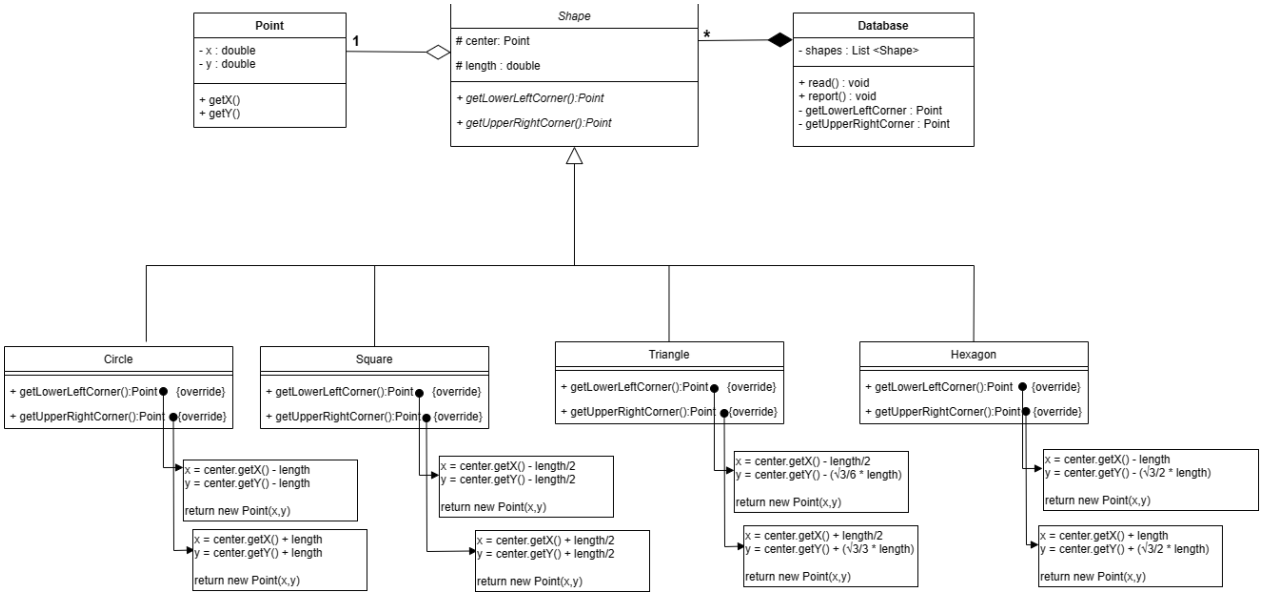
Let's focus on how we will determine the bounding box that contains all the shapes. Firstly, we do not have to calculate each corner of the bounding box. Calculating the lower left corner, and the upper right corner will be sufficient. To calculate these corners of the big bounding box, we should first be able to calculate the same corners for each shape's bounding box. Once we have the left lower corner values for all the shapes' bounding box, we will search for the minimum X value and the minimum X value. These two values will form the x and y values of the big bounding box's lower left corner point. We will do the same for the upper right corner point, with the only difference being that we will be comparing the upper right corners of each shape, and we will search for the maximum values these time around.

We will create a Point class which will represent any point in the two-dimensional plane. The point class will consist of an x and y coordinate. The x and y coordinate will be of Double data type. The reason being that this is a small program, in which we can afford to use doubles without any visible performance side-effect. By doing this, we will ensure the high precision for users.
We will create a Shape abstract class. It will serve as a base class for the 4 necessary shapes we need to implement. A shape will be represented by its center point and its length. It will provide methods to calculate the lower left and upper right corners of the bounding box, but will not implement any of these, because the implementation differs from shape to shape.

Next, we will implement the 4 shapes: circle, square, triangle, and hexagon. As mentioned, these shapes differ in their way of calculating the lower left and upper right corners of the bounding box. Hence, each shape will override the original implementation.

Finally, the database class. In this class we will carry out the "business logic". It will contain an array of shapes, which it will populate from txt files. The class will offer methods to determine the smallest bounding box, which will contain all the shapes, and another method to print the coordinates of this bounding box.

UML



Description of each method

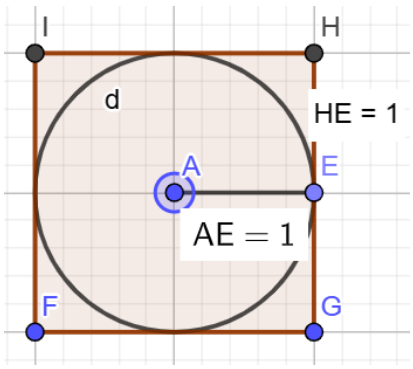
Getters and setters will not be covered in this section to save space and time.

Let's start by diving into how we calculate the lower left corner and the upper right corner of each shape

Circle

Determining the corners of a circle is fairly straightforward. A Circle will be represented by its center point(x,y) and the size of its radius. To get the upper right corner, we can simply add the radius to the x and y coordinates respectively. To get the lower left corner, we can simply subtract the radius from the x and y coordinates. This description sums up the **getLowerLeftCorner()** and **getUpperRightCorner()** methods of the circle class.

For example, the circle below has a center with coordinates (2,2) and a radius of length 1. The upper right corner of the bounding box (H) has coordinates (3,3). To obtain that result, we must first go right by 1 unit, and then up by 1 unit, which can be translated to adding the radius to the x and y coordinates of the center.

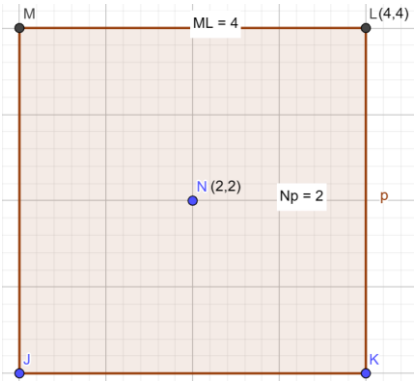


Square

The smallest bounding box of a square is simply the square itself. To get the upper right corner of the square we must add to the x and y coordinates of the center half of the side length. To obtain the lower left corner we must subtract half of the side length.

This description sums up the **getLowerLeftCorner()** and **getUpperRightCorner()** methods of the square class.

The image itself should suffice as an example.



Triangle

Computing the smallest bounding box of a triangle is a bit more complicated.
We must know the following basic geometric information:

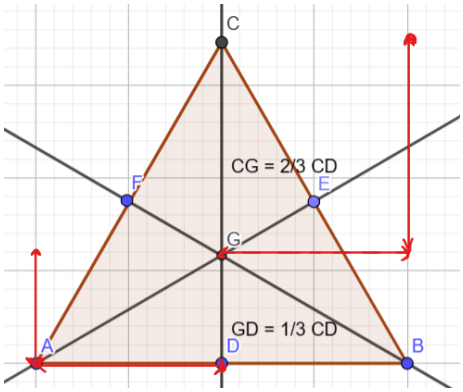
1) We consider the given center to be the geometrical center of the triangle, meaning it is the point of intersection of all the three medians of a triangle. The medians are divided into a 2:1 ratio by the centroid.
2) The height of an equilateral triangle can be given by the formula: ***h = (1/2) * √3 * a***

For the lower left corner point, we compute the x coordinate simply by subtracting length/2. To get the y coordinate, we must compute the height from the center to the base. The height from the centroid to the base is 1/3 of the total height. Hence if the total height can be computed by ***h = (1/2) * √3 * a***, we can subtract (√3/6 * length) from y to obtain the y coordinate of the bounding box.

For the upper right corner of the bounding box, we compute the x coordinate simply by adding length/2 to it. To get the y coordinate, consider the height from the center to the top of the triangle. The height from the centroid to the tip of the triangle is 2/3 of the total height. Hence if the total height can be computed by ***h = (1/2) * √3 * a***, we can add (√3/3 * length) to y to obtain the y coordinate of the bounding box.

This description sums up the **getLowerLeftCorner()** and **getUpperRightCorner()** methods of the triangle class.

You can refer to the following image.



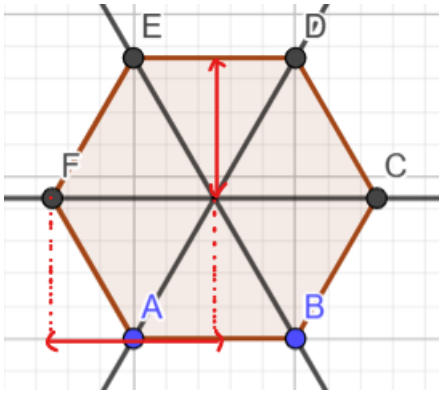
Hexagon

The hexagon is straightforward after the triangle, but we still need to refresh our geometry knowledge. Keep in mind that a hexagon is composed of 6 smaller triangles, and the distance from the center to the base/top of the hexagon is the height of one triangle.

For the upper right corner of the bounding box of a hexagon, we do the following. To compute x, we simply add the length value provided. To compute the y-coordinate we add ***√3/2 * length*** to it.

The same can be applied to the lower left corner.

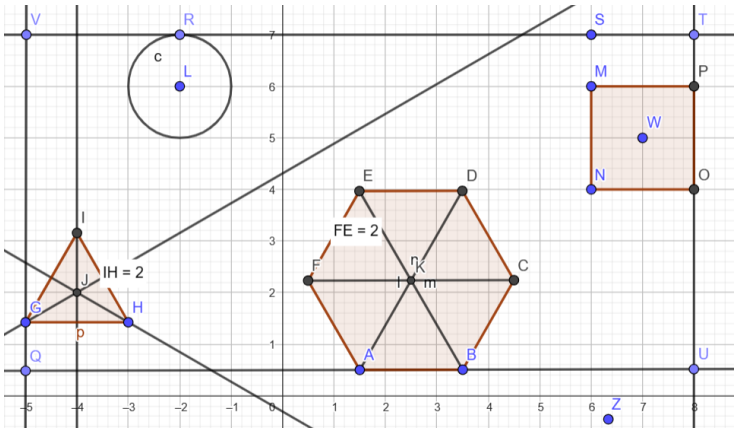
This description sums up the **getLowerLeftCorner()** and **getUpperRightCorner()** methods of the Hexagon class.



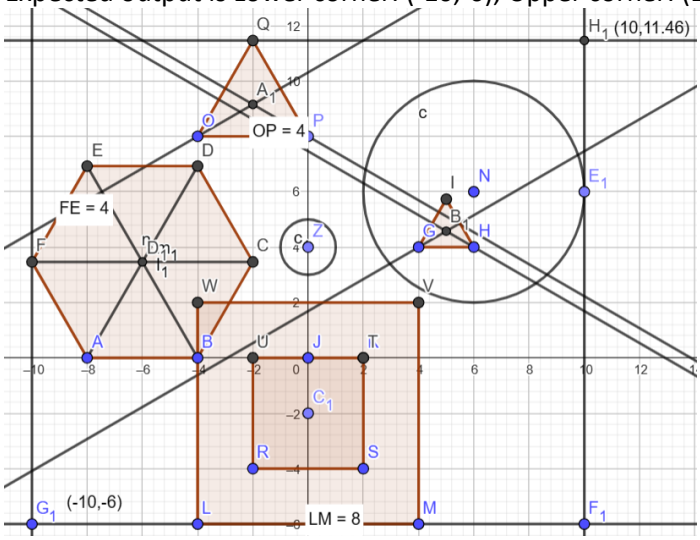
Test cases

1) **Testcase1.txt**: All the data are provided are valid

   Expected output is Lower corner: (-5.0, 0.50) , Upper corner: (8.0,7.0)
   The following is a graphical representation of the input.



2) **Testcase2.txt**: All data valid, more data are provided
   Expected output is Lower corner: (-10,-6), Upper corner: (11.49)



3) **Testcase3.txt**: provide the number of shapes in wrong format
   "blla" -> "Input can't be parsed"
   -3 123132 123 123123 -> "Input can't be parsed"
   -3 -> Number of shapes should not be negative or 0
4) **Testcase4.txt**: Invalid shape type name
   Instead of "H" for hexagon, we provide "joke" -> Type doesn't exist
5) **Testcase5.txt** : One of the data about the shape, such as center point is given in wrong format
   Instead of providing a number for the length, we provide "INVALID" -> Input can't be parsed