6-th October 2019

**1. assignment/6. Task**

**Sadi Mamedov**

LHXOTB

sadimamedov7@gmail.com

Group 2

# Task

Choose a point on the plane, and fill a collection with several regular shapes (circle, regular triangle, square, regular hexagon). ***How many shapes contain the given point?***

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.

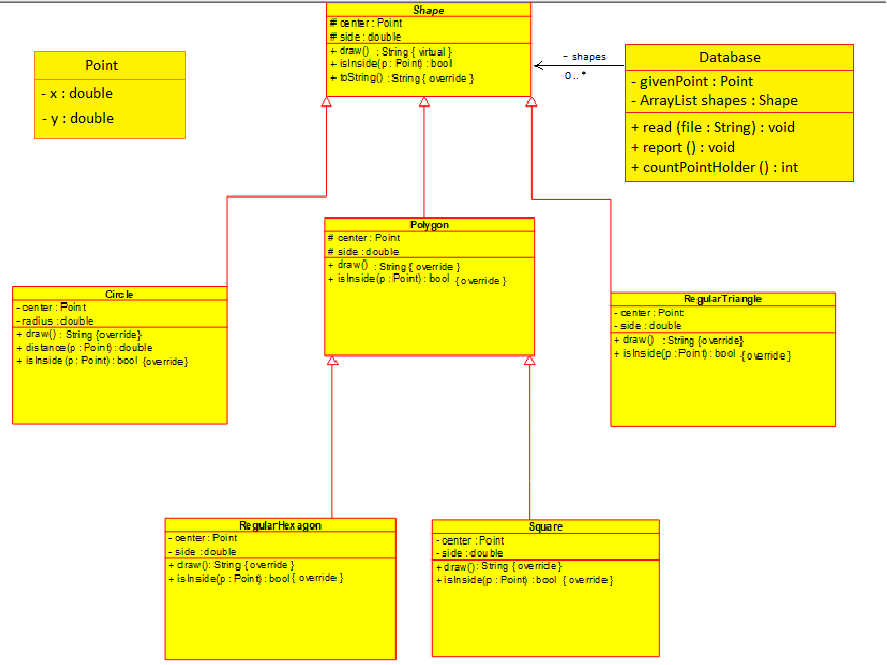
Load and create the shapes from a text file. The first line of the file contains the number of the

shapes, and each following line contain 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.

# Class Hierarchy / UML



Description of the classes:

**#1 Class Point:** Defines the data structure of the two dimensional coordinate system providing

axes (x – axis: double) and ordinate (y –axis: double) member variables.

**#2 Class Shape**: Contains an abstract shape, and consists of two private fields *side(double)* and center *Point(double ,double)* where point is the data type of class Point defined by attributes x and y.

Shape class plays the role of a super class so all the other different shapes will inherit from this.

*\*Method draw()* 🡪 abstract method of type void, illustrates the kind of object by name returning String value.

*\*Method toString()* 🡪 method of type String, overrides Object class’s method to exhibit attributes’ values to the System output.

*\*Method isInside()* 🡪 abstract method of type Boolean, checks if the point entered by user is indeed inside the shape.

**#3 Class Polygon**: Extends the class Shape by having same attributes and overriding abstract methods. It consists of two private fields *side(double)* and center *Point(double ,double).*

*\*Method draw()* 🡪 overriding method from super class Shape , illustrates the kind of object by name returning String value “Polygon”.

*\*Method isInside()* 🡪 overriding method from super class Shape, checks if the point entered by user is indeed inside the shape.

**#4 Class Circle**: Extends the class Shape by having same attributes, overriding abstract methods as well as assigning value of member variable side to radius which is particular to the circle. It consists of two private fields *radius(double)* and center *Point(double ,double).*

*\*Method draw()* 🡪 overriding method from super class Shape , illustrates the kind of object by name returning String value “Circle”.

*\*Method distance()* 🡪 method of type double, calculates the distance between the center of coordinate system and given arbitrary point.

*\*Method isInside()* 🡪 overriding method from super class Shape, checks if the point entered by user is indeed inside the shape. Returns true, if the distance is less than the radius of circle

**#5 Class RegularTriangle** : Extends the class Shape by having same attributes and overriding abstract methods. It consists of two private fields *side(double)* and center *Point(double ,double).*

*\*Method draw()* 🡪 overriding method from super class Shape , illustrates the kind of object by name returning String value “RegularTriangle”.

*\*Method isInside()* 🡪 overriding method from super class Shape, checks if the point entered by user is indeed inside the shape. Initially discovers all the vertices of regular triangle on the coordinate system. Then makes conversion into Barycentric coordinates in which returns true if the inequalities

0 <= s <= 1 and 0 <= t <= 1 and s + t <= 1 hold true.

**#6 Class RegularHexagon:** Extends the class Polygon by having same attributes and overriding abstract methods. It consists of two private fields *side(double)* and center *Point(double ,double).*

*\*Method draw()* 🡪 overriding method from super class Shape , illustrates the kind of object by name returning String value “RegularHexagon”.

*\*Method isInside()* 🡪 overriding method from super class Shape, checks if the point entered by user is indeed inside the shape. Initially calculates the inscribed circle’s radius, and returns true, if the distance is less than the radius of inscribed circle with a slight inaccuracy.

**#7 Class Square:** Extends the class Polygon by having same attributes and overriding abstract methods. It consists of two private fields *side(double)* and center *Point(double ,double).*

*\*Method draw()* 🡪 overriding method from super class Shape , illustrates the kind of object by name returning String value “Square”.

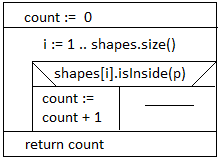
*\*Method isInside()* 🡪 overriding method from super class Shape, checks if the point entered by user is indeed inside the shape. Initially discovers the bottom left and top right vertices of square on the coordinate system. And returns true, if the point lies in the border of vertical and horizontal slicing.

**#8 Class Database:** stores collection of reference type Shapes to the ArrayList and reserves private member given point of type *Point(double ,double).*

*\*Method read()* 🡪 reads the data from the file using class Scanner, and after initializes the objects and stores them into collection.

*\*Method report()* 🡪 outputs the properties of derived classes stored in collection.

*\*Method countPointHolder()* 🡪 counts the total number of a given point occurrence in the shapes and returns the integer type of count using the standard count algorithm.



**#9 Class Main**: Launches the program in the main method, accessing other classes through the Database class. Illustrates the result of program, and errors in case of any exceptions.

Exception Handling:

*\*****Ex FileNotFoundException*** *🡪* Thrown whenthe file with the specified pathname does not exist with warning message.

*\*****Ex InputMismatchException*** *🡪* Thrown by a Scanner to indicate that the token retrieved does not match the pattern for the expected type, or out of range of expected type.

Testing:

**#1 Case:** Scanner reads an empty file, not any shapes exist in the collection, expected output: **0**

**Source:** test1.txt

**#2 Case:** All the shapes contain the given point, expected output: **5**

**Source:** test2.txt

**#3 Case:** There exists only one shape, and does not contain the given point, expected output: **0**

**Source:** test3.txt

**#4 Case:** Not any shape exists containing the given point, expected output: **0**

**Source:** test4.txt

**#5 Case:** Among all, only regular hexagon and square contain the point, expected output: **2**

**Source:** test5.txt