Design of Triangle Problem

Types :-

The three type of triangles share one state, which are the sides of the triangle. Therefore, I have decided to use an *Abstract Class* " Triangle ", which holds a *List<Double>* represents the sides of the triangle.

Each concrete triangle extends the Triangle class, and consequently inherits the state, i.e. *List<Double>*. Moreover, the equilateral triangle is a **special case** of the isosceles one, so it makes sense to design the first as a *subclass* the latter.

Each concrete class contains a *Final Static String* holds represents it type. It's better to be in the concrete type, to avoid explicit strings in the code. This improves the code readability and avoid potential misspelling bugs. The *toString()* method is overridden in each concrete class to identify its type, and validate the output.

Factory :-

To decouple the code, the **Factory** design pattern is utilized. This guarantees that the logic of determining the type of the rectangle based on it's sides, is not coupled with the instantiation of the concrete triangle objects itself. Certainly this makes the code easier to maintain and to change the design later on.

It would be easier if the Factory is implemented using conditional statements, each of which return one of the triangle concrete types. Though, this also couple the code which is a bad programming practice. OOP design should be open-close, means it can be modified with minimal changes. To achieve such goal, I have used **Reflection** to instantiate the required concrete triangle type. Using a *Hash Table*; in which the key is a *String* contains the concrete type name, and the value is it's *Class object*. Consequently, adding more triangle types does not requires editing the Factory anymore. Each new Triangle type **registers** itself in the Hash Table, and once the client asks for it the Factory return the required object. The code in the Factory is the same for all the legacy and new types.

Logic :-

TriangleTypeDeterminer class is responsible for deciding which type of rectangles is required, based on the length of it's side. It is possible to iterate over the sides and compare each other in a *brute force* way, or even use a conditional statements in the Logic class to determine the concrete type. But I have a use a *HashSet*, to determine the type of the triangle, by its size after filling all the sides into the HashSet.

Recall that the Set interface does not allow duplication, so if all the sides are the same, the size of the HashSet will be equals to "1", and it two sides are similar the size is equal to "2", the final case where all are different the size will be "3". Based on the size of the set, the logic request the concrete triangle type from the Factory, and return it.

Also two overloaded methods were implemented, the first takes the sides as variables, and the second take it as a List<Double>. Since the Logic is the same, it is a mandate to call the second method from the first one in order to avoid **code duplication**.

Client :-

A tester class which invokes the Logic class and display the result.