The task:

A closed 2D shape is represented as an array of connected 2D points.
Each point have x and y coordinates.
For 1m square it can be like this [{0,0}, {1,0}, {1,1} {0,1},{0,0}]

A valid 2D shape must have no sub-contours or interior points.

Any points should be used only once, with the exception of the first/last point of the figure.

A valid shape can be a convex polygon or a non-convex polygon
(https://en.wikipedia.org/wiki/Convex_polygon)

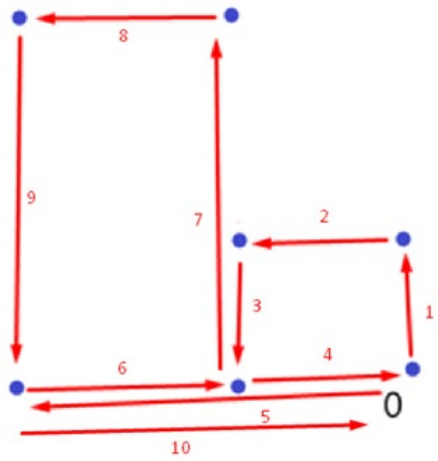
Invalid 2D shape in real practice occurs when a new shape is created as a direct concatenation of two or more simple valid shapes.

The Task is as follows: (Also please an additional clarification below)

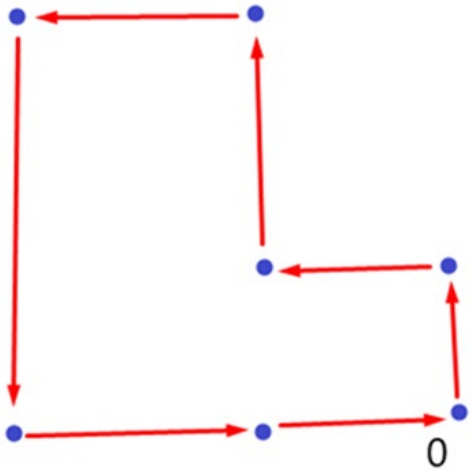
Implement (in Java) TheShapeFixer class,
that implements shape validation and correction with two required methods:

isValid(Shape2D): boolean - returns true if a shape is valid
repair(Shape2D): Shape2D - takes an invalid shape and creates a valid version of it

(for example, shape 1 is invalid, while shape 2 is its valid version)



1



2

Clarification:

Perhaps I should clarify the task. In the resulted shape the correct points order is important. If we try to analyze the examples, we will see that only removing double points and self-intersections from left shape cannot give ua a valid right shape. The points must be rearranged.

We must get closed counter-clockwise shape as possible close to convex polygon around all our points (see “Convex hull algorithms” https://en.wikipedia.org/wiki/Convex_hull_algorithms).

But we cannot use hull algorithms directly because our polygon can be non-convex polygon (our example is not convex), and points in result polygon can be connected only if they are connected on the original or if a point lays on the line between the other two points.

Let’s see an additional example. Hull algorithms can create the following convex shape, but it is incorrect for us, because points 1 and 3 do not connected directly in the original shape.

Can use segments 1-2, because points 1 and 2 connected on source, and 2-3, because point 2 lies on line (7) between points 6 and 3