**Solution Idea:**

Input constraints:

* Input the number of nodes (boundary) of the polygon.
* Input each of the nodes as a 2D point on a XY plane. The polygon can be convex or concave but no intersecting edges. The input of the points should be in a counter-clockwise order with respect to a XY plane.

Y

4

3

2

1

X

* Input a color for each of the boundary nodes among Red, Blue or Yellow. (Red = r, Blue = b, Yellow = y).
* Input the number of nodes inside the polygon.
* Input each inside node same as a 2D point on a XY plane and the point must not be outside or on the boundary of the polygon.

Sample Input:

4  
2 2 r  
5 1 y  
7 6 b  
3 5 y  
1  
5 4

Solution:

Classes:

* A class named PointCoordinate containing variables x (x value), y (y value), color (color of the node) represents a node of the polygon as an object.
* A Main class to take input and, apply logic to get the output.
* A supportive class FindIntersection to do all the geometry.

Algorithm:

* Create an adjacency matrix (two-dimensional array) G of size total-nodes to represent the graph (including boundary and inside vertices).
* Update G by connecting boundary nodes as given input.
* Fill the array for connections between the boundary vertices (1 represent a connection, 0 represents no-connection)
* Now, the algorithm creates the Triangulating Polygon with non-overlapping triangles.
  + For a node ‘i’, the algorithm connects it with each node ‘j’. If the x value of j is smaller than the x value of i, and
    - the edge between i and j doesn’t contain any of the other nodes on it
    - the edge has no intersection point on it with any of the other established edges of the polygon.
* Now, the algorithm does color the inside nodes of the polygon with a recursive method with each of the possible combination of given number of colors.
* After assigning a color to all the inside nodes by the recursive approach, the algorithm checks by traversing the triangulatingPolygon graph whether the solution exists for two and no more complete triangle or not.
* Whenever the algorithm finds a solution for a combination of colors it returns true (Solution exists) otherwise returns false (solution doesn’t exist).

Executing instruction:

* The project is developed in IntelliJ IDEA. If possible you can open the cloned project in the IDE and run.
* The Ant build script is provided in the project repository, so you can use that file to run the project in any Ant build compatible IDE (I hope ☺).
* JDK 1.8 is used
* Java language level 8

Resources:

I have opened plenty of websites or forums or pages. I am mentioning a few which are closely related.

<https://en.wikipedia.org/wiki/Delaunay_triangulation>

<https://en.wikipedia.org/wiki/Point_set_triangulation#cite_note-6>

<https://en.wikipedia.org/wiki/Art_gallery_problem>

<https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/>

<https://en.wikipedia.org/wiki/Polygon_triangulation>

<https://www.geeksforgeeks.org/check-whether-a-given-point-lies-inside-a-triangle-or-not/>

<https://www.geeksforgeeks.org/program-for-point-of-intersection-of-two-lines/>

<https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/>