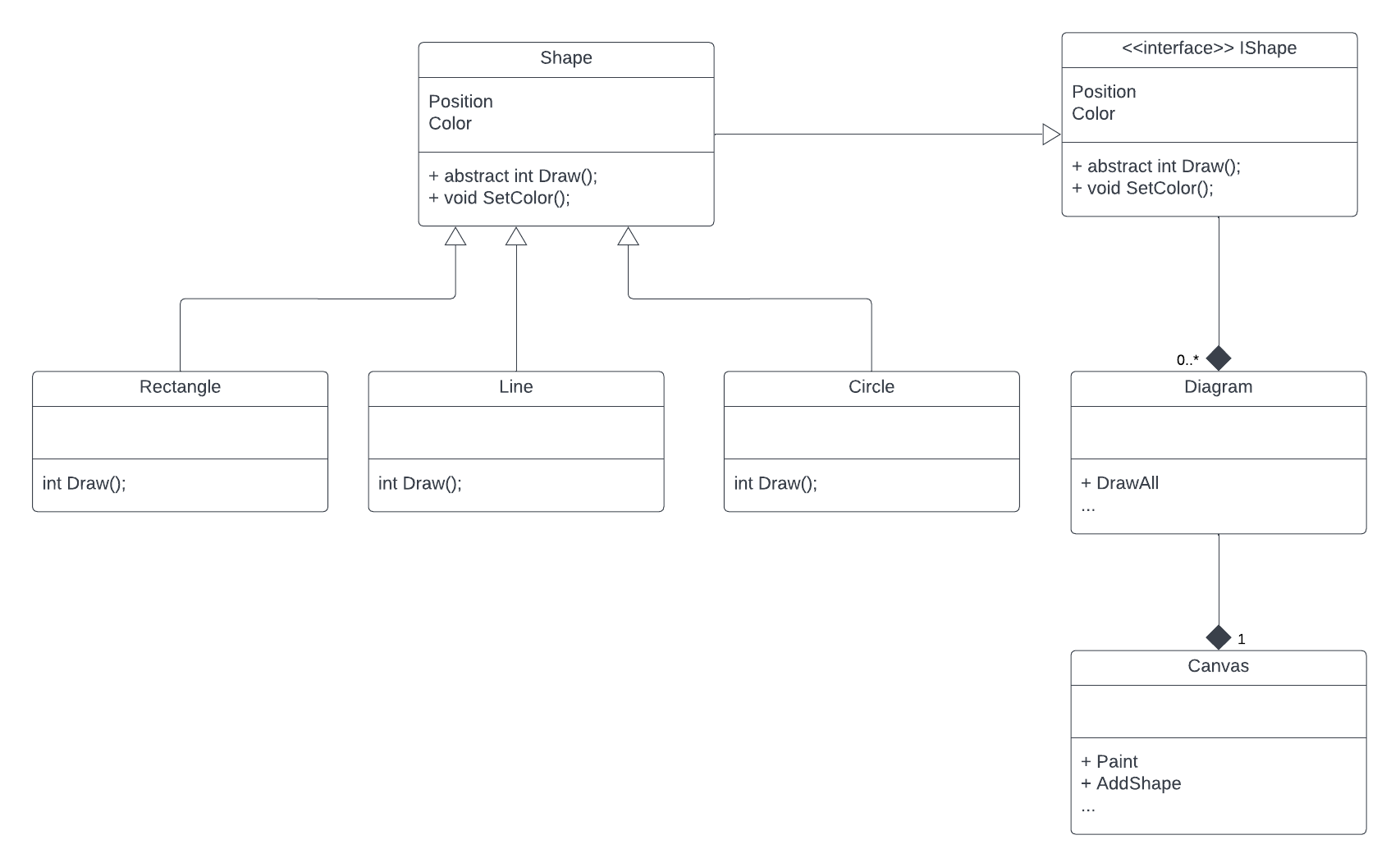
# Shape Diagram

The Diagram below shows a light version of the shape model.



# Design Patterns

Decorator Pattern was applied per shape; also, there is a main Decorator pattern named **CompositShapeDecorator; Decorators** Class like GreenLineDecorator and others probably are a redundancy since **CompositShapeDecorator**covers all scenarios for this exercise. But this **CompositShapeDecorator *may*** not have any value in a more complex environment.

For the API solution, the ImageService class behaves like a facade pattern.

# API Explanation

Notice the instance of the ImageService class is a singleton instance. This is not ideal but is a workaround to keep the state of the shapes since I did not implement a cache or database to have persistent data.

# Side notes

* With more time and more extensive scope, the canvas needs to be handled like a unique document/identifier. And should be part of the API route. For clarity and the scope of the exercise, I keep it simple, which carries on some limitations.
* There is no logic about drawing a shape per type, and only the placeholder or the method shell was declared.
* In order to have some logging or debug info, there is some ***console.writeline***, I am aware it is not ideal and affects the single responsibility of the class, is not an ideal solution, but needs it to have some testing and visibility on the class behaviors.