Design 2

Observer Pattern

ObjectInterface is an abstract class which is further implemented by the class *Shape.* *ObjectControllerInterface* is basically an Observer Interface because it is controlling an Observervable (which is in this case ObjectInterface). ObjectInterface is an object which notifies ObjectControllerInterface about the changes in its state.
For instance, when any action in a class Shape was taken, like, an add/edit/delete/move, a controller gets a notification/update on exact changes and therefore ObjectController updates its Status.

Strategy Pattern

A class Shape is an abstract superclass which has 8 different impementations (basically, 8 shapes). Each time a constructor of a child was called, a constructor of a superclass was called, too. When you transform a shape from REST object into a string its a Strategy Draw method.
SVGContainer. As well, *ObjectControllerInterface* is an abstract class which is implemented in an *ObjectController.*

Iterator Pattern

In *ObjectController* all the modifications on my Objects will be implemenated through a usage of a ListArray, where while add/delete/edit I access certain Shape (through a key), since they all are saved in ListArray and have ID which is a key.
In a class *MyObjects* there is a method ListAll which goes through created shapes and list them with their keys – ids.

Proxy Pattern

The Proxy provides a surrogate or place holder to provide access to an object. In our case we decided to prevent unnecessary usage of a memory, so that in a class ShapeProxy a method will be called which is responsible for a creation of graphics through SVG. The method called createObject first checks if Object was already initialized or not. In first case its further creates a shape. Otherwise it first initializes the object and then creates a shape. This prevents initialization of an object twice.

Composite Pattern

Composite pattern will be mainly used for identification of a type of a Shape. For instance, user can search only triangles or so. For these purposes a method “classifyObject” will be used. In this case we are making a group of Objects answering certain parameters, so we

make a composition of those. This will be done through a class MyObjects, where there is a whole array of lists saved. Object controller is responsible for the group task as well. MyObjects basically represents all the created objects, also provides all the functionality that ObjectController provides.

Abstract Factory Pattern

In our class *ConrollerCreator* there will be few methods for the qualities of our Object: identShape, identColor, identPlace, and then it is all divided in different classes where those extentions of a shape are to be implemented: *AbstractShapeFactory*, *AbstractColorFactory*, AbstractPlacement.

Factory Method Pattern

There is a factory for creation of Objects – different shapes. There is an interface *BodyCreator*. In this class Objects of a shape will be not only created, but also checked for Correctness through many assertions to minimize a number of errors during creation of an Object. It is responsible for creation of objects without having to specify the exact class of the object that will be created.
Nextly, Software quality will be proven through Requirements list, which should make a procedure of creation most efficient and free from mistakes, minimize waiting times and make modifiability higher.
In order to create a Controller, there is a class *ControllerCreator*, which also minimized all possible mistakes is responsible for correct and consistent creation. Besides that, there is a a class *ShapeProxy*, where all the complexity/overhead of the target in the wrapper will be encapsulated.

Decorator Pattern

We do maintain the functionality of each interface. What is used here is that we came up to a conclusion it is not necessary to create “child” classes each time super class was created. And since our Shape has 8 implemenations, would be irrational to create a polygon when a triangle was called, as an example. As well, in an *ObjectController* we have done edit function which is used for all our shapes – despite Shape has its children – we do just extend different edits for different figures, but we don’t have on *ObjectController* 8 different edits, so its interchangeable. The Decorator attaches additional responsibilities to an object dynamically. So, whenever a function was called, it will only edit objects dynamically.