**Architectural Design**

The whole system is based on the MVC (Model-View-Controller) architectural design pattern that allows us to separate the business logic (Controller), data management (Model) and how the data are presented to the user (View).

We have chosen this pattern because it uses an Object-Oriented approach which is very suitable for our context, the representation of our data as models.

**Architectural Decisions**

**Singleton Design Pattern**

The application will implement a GUI(Graphical User Interface) so the user will trigger events interacting with it. The events will be captured by the MVC controller that will use an unique instance of the class DrawingEditor to perform the related actions.

**State Design Pattern**

Given that the user can perform several operations on different shapes, it has been decided to use the State Design Pattern so that the actions triggered by the user aren't directly dependent from the DrawingEditor object but from its state.

**Command Design Pattern**

All the operations that change the drawing must be undoable. This means that there is an operation which works on other operations. In order to implement this behavior it has been decided to use the Command Design Pattern. In this way, operations will be transformed into objects(Commands) and the undo operation could work on them.

**Object Adapter Design Pattern**

In order to reuse an existing library (java.scene.shape.Shape) that by its nature it’s not compatible with the Canvas object, used to draw the shapes, it has been decided to use the Object Adapter Design Pattern that allow us to reuse the shape properties and method of the existing classes using a different interface suitable with our aims.

**Attached:** UML diagram

<https://drive.google.com/file/d/1zRVuKbnuFxyu-Vz8oMoQ--eq_r3aeDnp/view?usp=share_link>