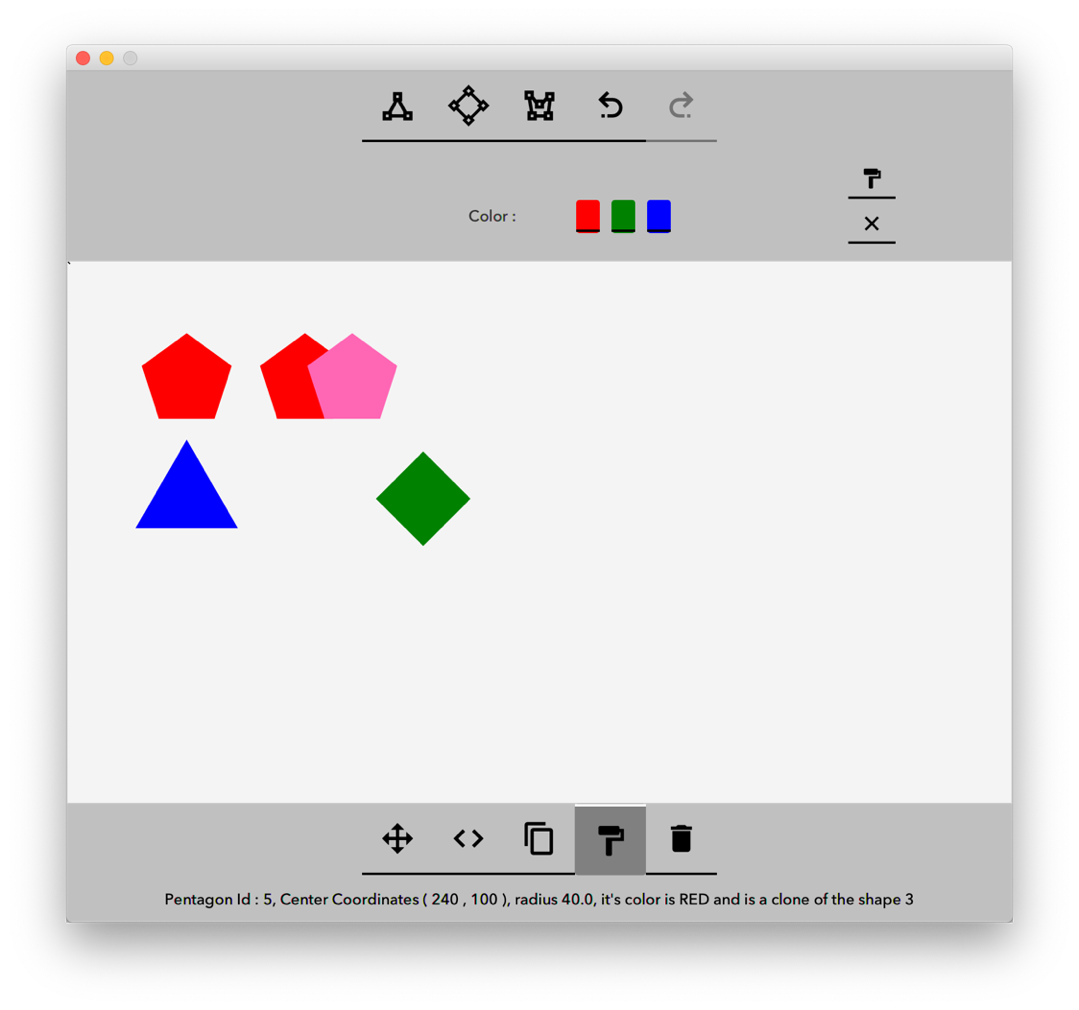
Multiple patterns

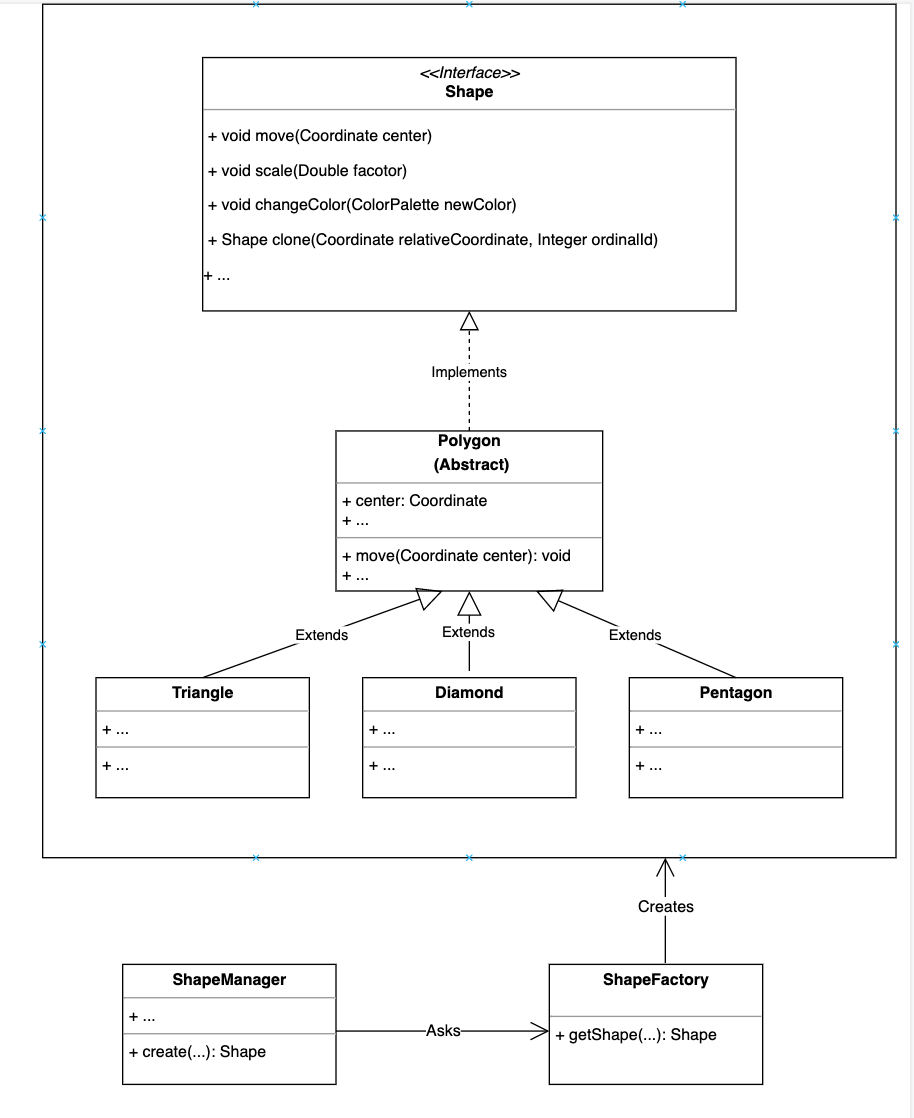
# Points of attention

* The shapes that are chosen in this project are polygons that have equidistant points from the center of the shape.
* The X, Y coordinates are integers
* The radius the represents the distance between the center of the shape and the shape’s points is a double strictly greater than zero.
* The scaling factor is a double, the shape’s size increases if this factor is greater than 1 and the shape decreases if this factor is between zero and one.
* The redo action won’t be possible until an undo action is executed.
* If a new action is done after an undo, the redo stack will be cleared.



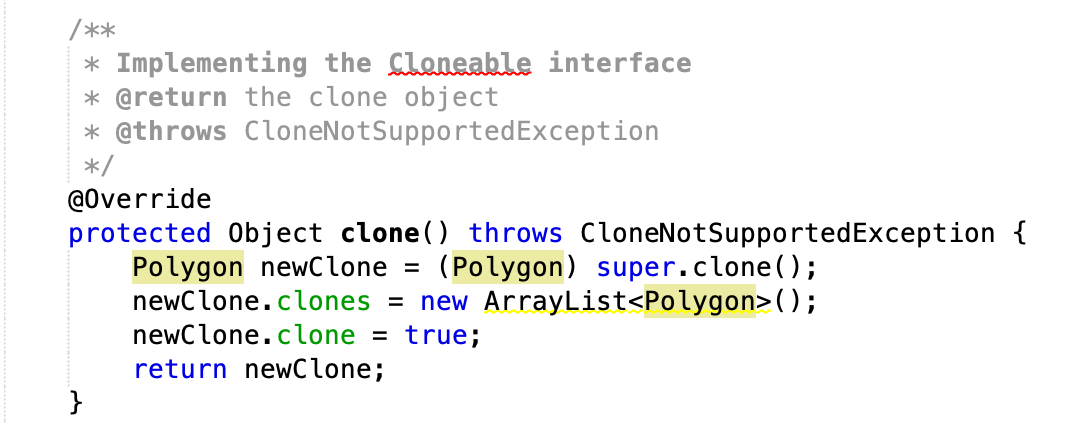
# Shape creation

To be able to manipulate multiple shapes, I used the Factory design pattern to get the right implementation of the shape depending on its type. For this purpose, a common interface is created that represents the tools that a shape can have. This interface is implemented by all the shapes that we will manipulate. An abstract class is created (Polygon) to aggregate the common behavior of the polygon shapes, it also gives the opportunity for child classes to override those methods if needed. In this case, only the String representation of the shapes has changed from the super class.

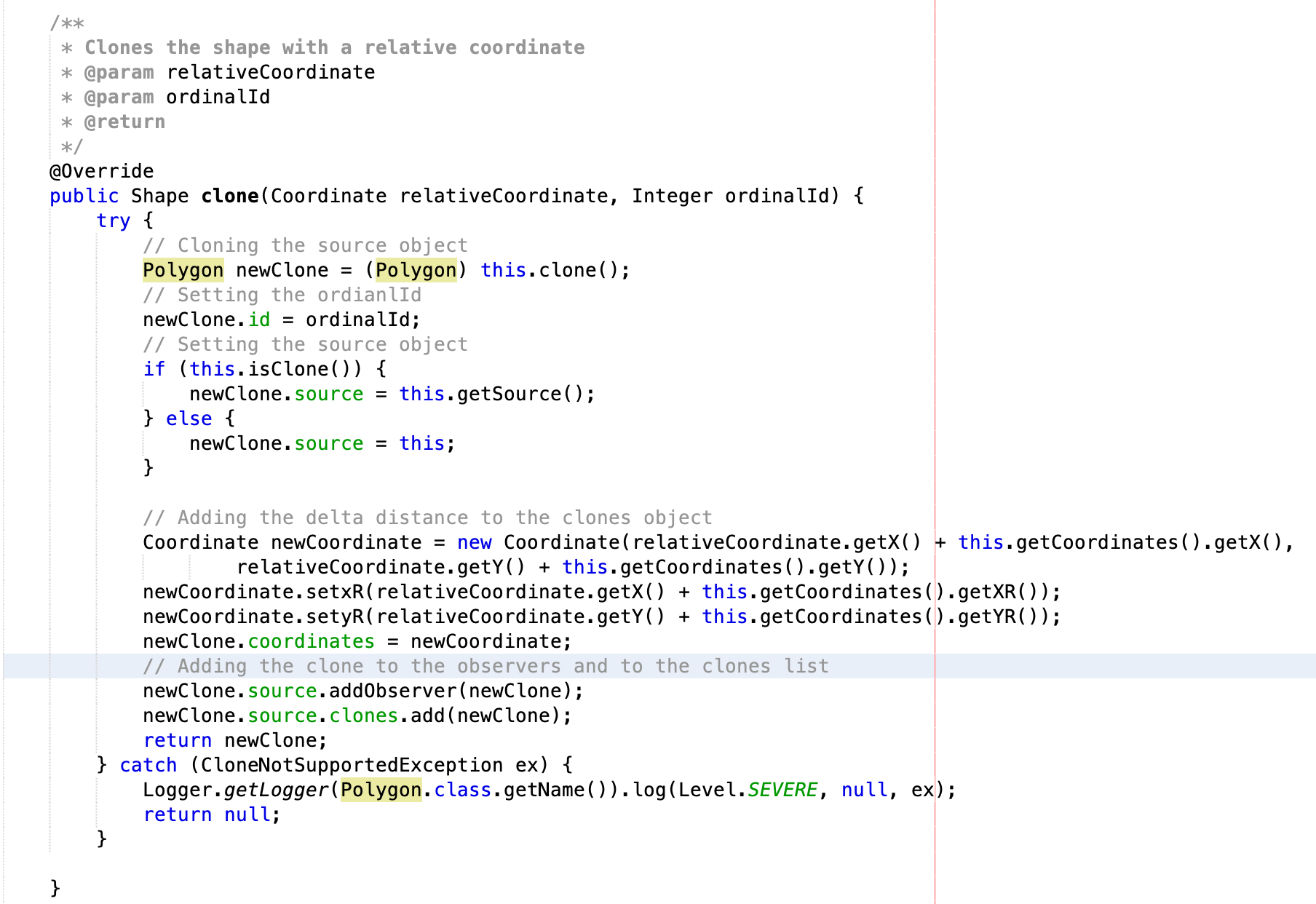


# Clones management

The cloning of the shapes is implemented through the Prototype design pattern. The Polygon object implements the Cloneable interface given by Java, and the clone method is implemented in the following way:



To add the delta distance from the source shape, the clone method is marked as protected and the actual method to be exposed is a clone method that takes a delta coordinates, this method calls the clone method first before to continue its job:

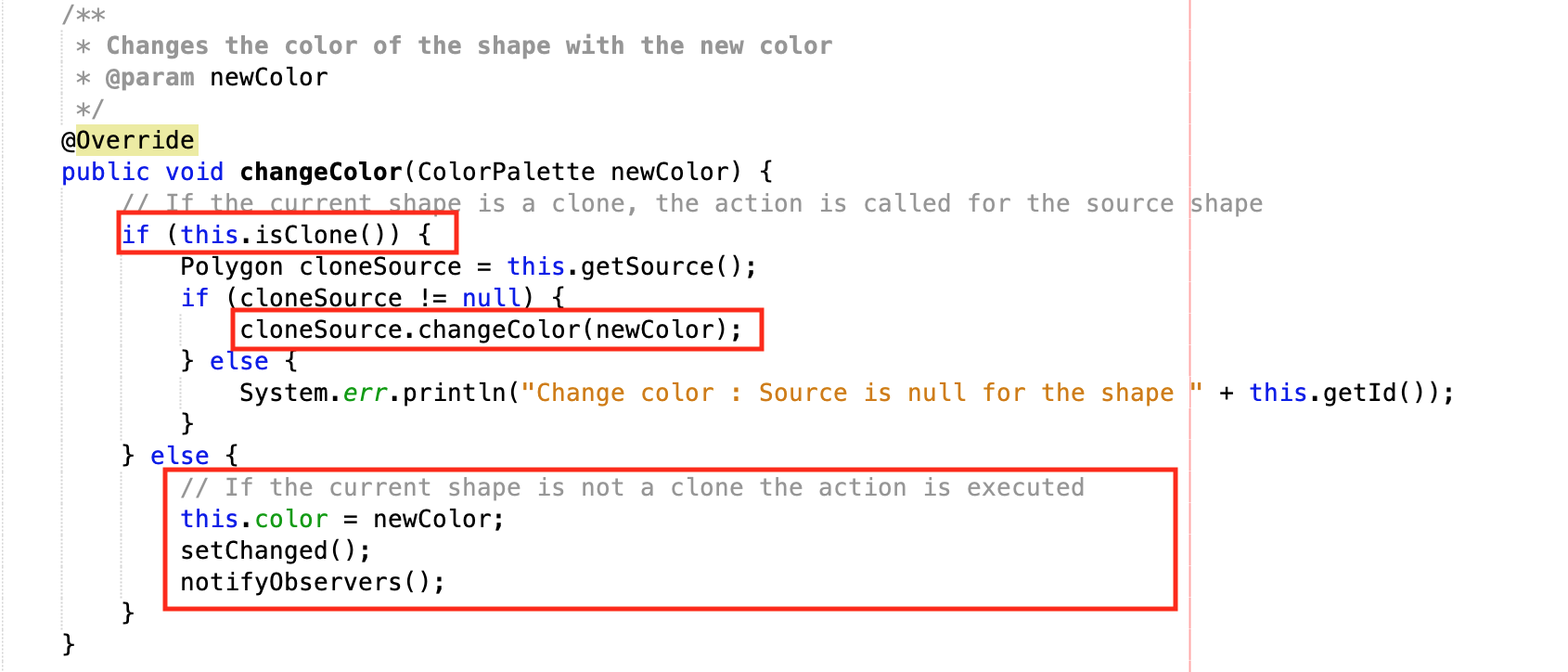


To transmit the action between clones and the source shapes, the Observer design pattern is used. Since the source object and the clones are of the same class, the polygon class extends the Observable class and implements the Observer interface.

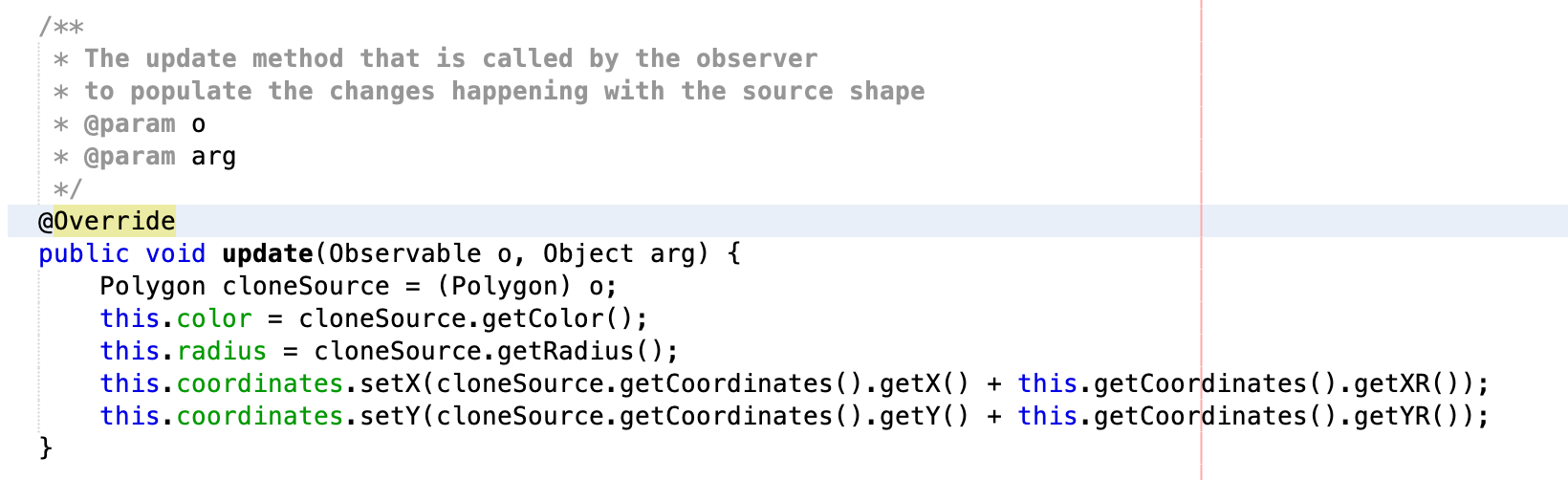
When a clone is created, it is added to the list of observers of the source object. This action has two cases: the object to clone is not a clone and then the clone is added directly to the source object, the second case is when the object to clone is a clone, and in this case the clone is added as an observer of the source of the object to clone:



The next step is to notify the observers when an action happens, this action is performed as follows: The notification of the observers is only executed when the current object is not a clone, otherwise the action is delegated to the source object.

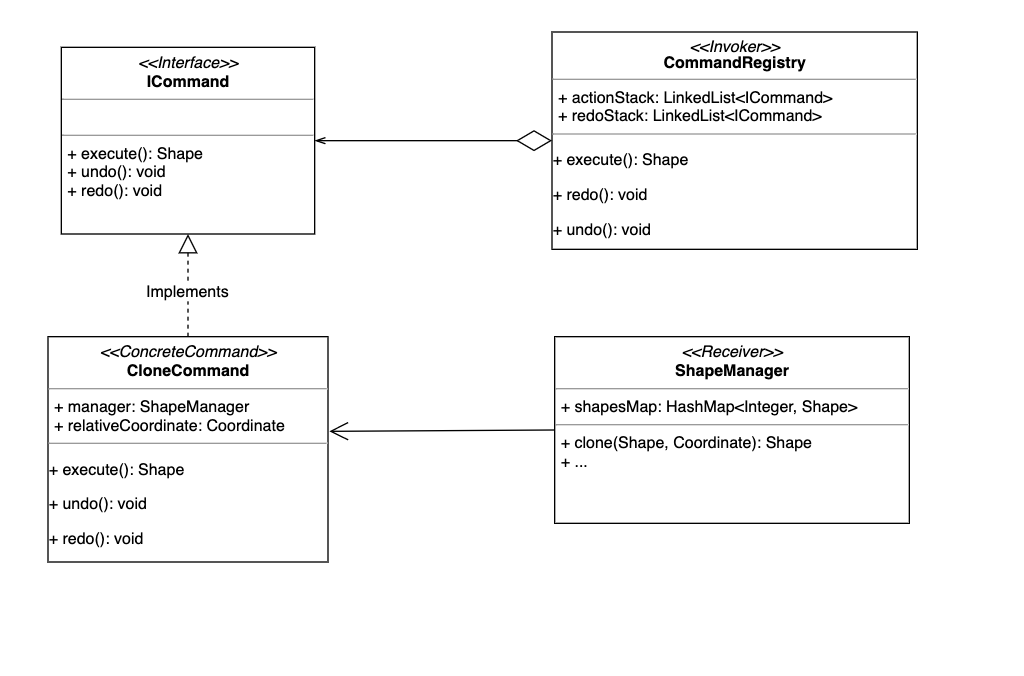


The update method of the observer is executed when this shape is notified of a change, in this case the properties of the observer are updated with the new values:



# Undo / Redo

Since the actions that can be performed are not very complex, the Command design pattern has been chosen with a compensation mechanism to execute the undo actions.



The best structure for managing the undo and redo mechanism is the linked list, that’s why it is used in the CommandRegistry class. It saves the command to be executed with the status of the important parameters. The undo process is done by compensation, if the action to be executed is f(), the undo method has to implement the f-1() action. The ShapeManager class has a HashMap to manipulate the shapes to give the possibility to access its elements using only the id.

# Iterating through the elements

To have the possibility to go through the different shapes created, the Iterator design pattern id used. The ShapeManager offers an iterator that has two main methods: hasNext() that checks if the list contains more elements, and the next() method to get the current shape. This Iterator is represented by the ShapeIteratorImplClass:



# Decoupling GUI from logic

The logic is presented as a library that offers an API that can be used by any form of GUI. The GUI uses this API as a library.