Between assignment 3 and 4 I found 3 to be much easier to implement the undo-redo feature, using the memento design pattern.

If I compare how I implemented each (then I'll explain how I implemented the undo and redo), when the user added a shape, the memento originator created a shape object, representing the current added shape while the commando had to call in intermediate invoker to actually run the shape adding command. The originator then created a memento object ("snapshot") of the shape and stored it in a "caretaker" which was much easier when compared with my command pattern, needing to not only store the commands in the invoker but also requiring the command class itself have an execute and unexecute command in order to implement the undo and redo feature (I do understand that execute has to exist regardless of whether an undo or redo feature are being implemented).

My command pattern had more complexity as I needed a receiver class to serve as my actual canvas where the shapes are added and removed using the commands, this canvas receiver can be compared to the caretaker but adding a shape to the caretaker is simple compared to adding to the receiver.

That was my biggest problem I found with command, to just store the shapes on the canvas it took three associated classes, invoker to run a command, command to create the shape object and the receiver to actually store the shape.

In terms of implementing undo and redo I did the same thing between both patterns, two stacks, undo which stores all the current shapes on the canvas and redo which stores all the shapes undone off of the canvas. To undo remove from pop from the undo stack and push to the redo stack, to redo pop from the redo stack and push to the undo stack.

I found this simple logic is suited towards the memento pattern which stores the all the shapes (mementos) just in the caretaker alone, therefore the caretaker is the class that handles undoing and redoing using two stacks (in my implementation I used lists) which is straightforward.

I thought the receiver would handle undoing and redoing as it was my canvas, it stored the shapes, but with command the commands themselves are what needs to be undone and redone therefore extra complexity is added needing the invoker to store the two (undo, redo) stacks and store the logic for the undo redo methods who call execute and unexecute methods (respectively) in the command class to undo and redo shapes stored in the receiver class, making such simple logic cumbersome and awkward as three classes have to interact for a simple undo, compared to the simplicity of the originator restoring a shape by calling the caretakers undo or redo methods which return a shape.

Therefore, I found memento as a pattern is pretty straight forward, store states, which for the assignment of storing shapes on a canvas translated over easily, using a design command pattern added complexity by storing commands, removing the simplicity of just storing the shape objects themselves.
Memento being simple to implement doesn't mean it will scale well, I found when I added loads and loads of shapes my computer got slower, this is probably because the program is storing whole objects while when I did the same with command it wasn't as bad since only commands are stored.

Memento was better to use for this assignment but that because this assignment was a simple enough application, simple as in not needing many other features, if that was the case I say command would have been more suited. Refactoring guru showed an example of command being used when different features to the same task, such as three buttons all save the file, our assignment was mainly just adding shapes using one command line interface, there was no other interfaces in the program which could add shapes, if that was the case the command pattern would be ideal as it accounts for these different ways of invoking a command.