Functional Requirement:
We need to desing a class for text editor that support the following **
- [Type] Add the Text to our editor
- [Undo] User can Undo the last Operation
- [Delete] User can Delete some Text

Brute Force Design:

```
## class TextEditior:
        __text:str
        __history:stack

        def__init(self):
            self.__text = None
            self.__history = []

        def type(self, text):
            self.text += text
            self.__history.append(text)

        def undo():
            self.___history.pop()

        def delete(self, length):
            self.__text = self.__text[:len(self__length) - length +1]
```

Problem in a given solution

- [ ] Its Violates the SOLID Principcples
  - [x] ~~Single Responsibility Principles~~
  - [x] ~~Open Closed Principles~~
- [ ] Its very Hard extend the functionality
  - [ ] Supoose Delete needs to be extend there many ways to delete the text Between or From end and From Start

Command Design Pattern

 is a behavioral design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request’s execution, and support undoable operations.

Let's Dsign a class for Command Design Pattern

- [ ] Define a Command class as a Interace

```
    class Command(ABC):
        def excute()
        def undo()
```

- [ ] Now Type Command Class will implement the command class

```
    class Type(Comand):
        text_editor: TextEditor
        text:str

        def __init(self, text_editor, text):
            self.text_editor = text_editor
            self.text = text

        def execute():
            text_editor.type(text)

        def undo():
            text_editor.delete(len())
```

- [ ] Delete Command class will implement the command class

```
    class Delete(Command):
        text_editor: TextEditor
        text:str

        def execute():
        def udno():
```

- [ ] Define our Text editor class

```
class TextEditor:
    text:str

    def type(text):
        self.text += text

    def delete(self, length):
        self.text = text[:len(self.tex) - length]
```

- [ ] Define our commad Manager will be intract with our command class

```
class CommandManager:
    history:Stack = []


    def execute_command(self, command:Command):
        command.execute()

    def undo(self):
        if self.history:
            history.pop().undo()
```

```
command_obj = CommandManager()
text_editor = TextEditor()

heading_commad = Type("Create my Document", text_editor)
para_commad = Type("Its my first doucument", text_editor)

# Now Excute command
command_obj.execute(heading_commad)
command_obj.execute(para_commad)
```