Memento Design Pattern

Used to save memento states of Editor so that we can rollback or undo a particular state.

Basically Editor class contains its own properties and functionalities, but with this, it also contains Editor memento composition which returns the object of EditorMemento with the current state stored inside this object

```
public class Editor {
    String text;
    int cursorX;
    int cursorY;
    int fs;
    String ff;
```

Editor class:-

Editor {

(composition)

```
EditorMemento getSnapshot() { //this is important.
    return new EditorMemento(text, cursorX, cursorY, fs, ff);
}
```

Constructor called

}

```
void restore (EditorMemento m) {
    this.text = m.getText();
    this.cursorX = m.getcursorX();
    this.cursorY = m.getcursorY();
    this.fs = m.getfs();
    this.ff = m.getff();
}

void print() {
    System.out.println("----------------------------");
    System.out.println(text);
    System.out.println(cursorX);
    System.out.println(cursorY);
    System.out.println(fs);
    System.out.println(ff);
}
```

to access the private feilds.

Test() Class {

```
EditorMemento em1 = e.getSnapshot(); //*Returns object of editorMemento through Editor.
```

}

```java
public class EditorMemento {
    private String text;
    private int cursorX;
    private int cursorY;
    private int fs;
    private String ff;
    private LocalDateTime moment;

    EditorMemento(String text, int cursorX, int cursorY, int fs, String ff) {
        this.text = text;
        this.cursorX = cursorX;
        this.cursorY = cursorY;
        this.fs = fs;
        this.ff = ff;
        this.moment = LocalDateTime.now();
    }
```

```java
public String getText() {
    return this.text;
}

public int getcursorX() {
    return this.cursorX;
}

public int getcursorY() {
    return this.cursorY;
}

public int getfs() {
    return this.fs;
}

public String getff() {
    return this.ff;
}
```

Memento object captures the state of editor object inside the editor class through get snapshot function and then this object with all the state at that time is stored inside the CareTaker list of memento objects.

```java
public class CareTaker {
    private Stack<EditorMemento> his = new Stack<>();
    //can be implemented through Arraylist if we want a specific index memento
    //Also can me implemented through Hashmap if we want a specific index corresponding
    //to a specific property like timpestamp.
    public void save(EditorMemento m) {
        his.push(m);
    }

    public EditorMemento undo() {
        return his.pop();
    }
}
```

CareTaker just registers/saves the memento Object inside it and exposes the function undo, where it pops these states whenever needed.

```
class Test {
    Run | Debug
    public static void main(String[] args) {
        Editor e = new Editor();
        CareTaker ct = new CareTaker();

        e.text = "hello";
        e.cursorX = 10;
        e.cursorY = 20;
        e.ff = "Devanagari";
        e.fs = 5;

        EditorMemento em1 = e.getSnapshot(); //*Returns object of editorMemento through Editor.
        ct.save(em1);
        e.print();

        e.text += "World ";
        e.cursorX = 30;
        e.print();

        EditorMemento em2 = e.getSnapshot();
        ct.save(em2);
```

```
        e.fs = 25;
        e.ff = "Comic Sans MS";
        e.print();

        EditorMemento em3 = e.getSnapshot();
        ct.save(em3);

        e.text += " ,hope all is well";
        e.print();

        e.restore(ct.undo());
        e.print();
        e.restore(ct.undo());
        e.print();
        e.restore(ct.undo());
        e.print();
```

```
------------------------------
hello
10
20
5
Devanagari
------------------------------
helloWorld
30
20
5
Devanagari
------------------------------
helloWorld
30
20
25
Comic Sans MS
------------------------------
helloWorld  ,hope all is well
30
20
25
Comic Sans MS
------------------------------
helloWorld
30
20
25
Comic Sans MS
------------------------------
helloWorld
30
20
5
Devanagari
------------------------------
hello
10
20
5
Devanagari
```

Facade :-

A layer added between client & subsystem of classes to make calling them easier.

Client

Facade

classes

```java
class Test {
    Run | Debug
    public static void main(String[] args) {
        Facade.work();
    }
}
```

I call for work

```java
public class Facade {
    static void work() {
        A o1 = new A();
        B o2 = new B();
        C o3 = new C();

        //work with particular order
        int x = o1.fun1();
        int y = o1.fun2(x);
        String a1 = o2.fun3();
        String a2 = o2.fun4();
        o3.fun5(y, a1);
        o3.fun6(a2);
        System.out.println(x: "Work Done!");
    }
}
```

automated work done.

```java
public class A {
    int fun1() {
        return 0;
    }

    int fun2(int x) {
        return 2 * x;
    }
}
```

```java
public class B {
    String fun3() {
        return "";
    }

    String fun4() {
        return "";
    }
}
```

```java
public class C {
    void fun5(int x , String y) {

    }

    void fun6(String z) {

    }
}
```