SPRINT #1

DESARROLLO DE ACTIVIDAD PATRONES DE DISEÑO

DANIEL ESTIVEN CUESTA NARANJO

C4 – G29

MIN TIC UNAB

UNIVERSIDAD AUTONOMA DE BUCARAMANGA

2/11/2021

Sprint #1

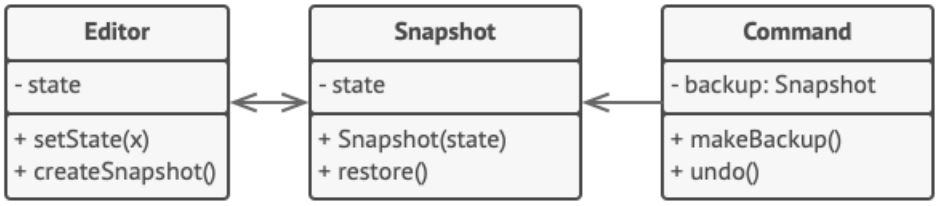
1.

1.1. Composite:

Este método permite al usuario restaurar y almacenar los elementos o el estado que haya realizado del objeto, con el fin de que este al querer renovar a una acción se dirija a donde se encuentra almacenado y traiga lo último que se realizó, de igual forma al guardarlo esté recorra todo el objeto y escanee sus elementos y los almacene

Ejemplos:

Ejem#1



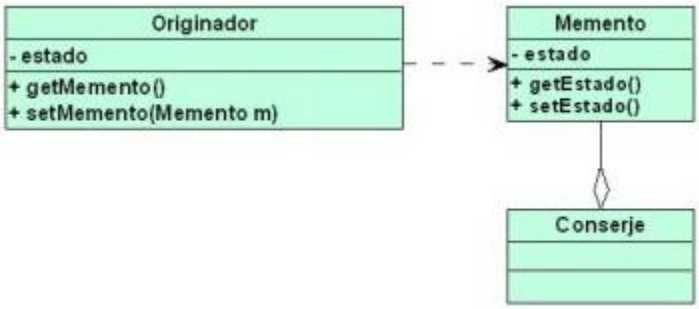

```
// El originador contiene información importante que puede
// cambiar con el paso del tiempo. También define un método para
// guardar su estado dentro de un memento, y otro método para
// restaurar el estado a partir de él.
class Editor is
    private field text, curX, curY, selectionWidth

    method setText(text) is
        this.text = text

    method setCursor(x, y) is
        this.curX = x
        this.curY = y

    method setSelectionWidth(width) is
        this.selectionWidth = width

    // Guarda el estado actual dentro de un memento.
    method createSnapshot():Snapshot is
        // El memento es un objeto inmutable; ese es el motivo
        // por el que el originador pasa su estado a los
        // parámetros de su constructor.
        return new Snapshot(this, text, curX, curY, selectionWidth)
```

```
// La clase memento almacena el estado pasado del editor.
class Snapshot is
    private field editor: Editor
    private field text, curX, curY, selectionWidth

    constructor Snapshot(editor, text, curX, curY, selectionWidth) is
        this.editor = editor
        this.text = text
        this.curX = x
        this.curY = y
        this.selectionWidth = selectionWidth

    // En cierto punto, puede restaurarse un estado previo del
    // editor utilizando un objeto memento.
    method restore() is
        editor.setText(text)
        editor.setCursor(curX, curY)
        editor.setSelectionWidth(selectionWidth)

// Un objeto de comando puede actuar como cuidador. En este
// caso, el comando obtiene un memento justo antes de cambiar el
// estado del originador. Cuando se solicita deshacer, restaura
// el estado del originador a partir del memento.
class Command is
    private field backup: Snapshot

    method makeBackup() is
        backup = editor.createSnapshot()
```

Los objetos de comando actúan como cuidadores. Buscan el memento del editor antes de ejecutar operaciones relacionadas con los comandos. Cuando un usuario intenta deshacer el comando más reciente, el editor puede utilizar el memento almacenado en ese comando para revertirse a sí mismo al estado previo.

La clase memento no declara ningún campo, consultor (getter) o modificador (setter) como público. Por lo tanto, ningún objeto puede alterar sus contenidos. Los mementos se vinculan al objeto del editor que los creó. Esto permite a un memento restaurar el estado del editor vinculado pasando los datos a través de modificadores en el objeto editor. Ya que los mementos están vinculados a objetos de editor específicos, puedes hacer que tu aplicación soporte varias ventanas de editor independientes con una pila centralizada para deshacer.

Ejem#2



Main.java:

```java
public class Main
{
    public static void main(String[] args)
    {
        // Crear el objeto originador/creador
        Originator creador = new Originator("Pedro", "Gil Mena");

        // Crear el objeto gestor/vigilante del Memento
        Caretaker vigilante= new Caretaker();

        // Crear el Memento y asociarlo al objeto gestor
        vigilante.setMemento( creador.createMemento() );

        // Mostrar los datos del objeto
        System.out.println("Nombre completo: [" + creador.getNombre() + " " + creador.getApellidos() + "]" );

        // Modificar los datos del objeto
        creador.setNombre("María");
        creador.setApellidos("Mora Miró");

        // Mostrar los datos del objeto
        System.out.println("Nombre completo: [" + creador.getNombre() + " " + creador.getApellidos() + "]" );

        // Restaurar los datos del objeto
        creador.setMemento( vigilante.getMemento() );

        // Mostrar los datos del objeto
        System.out.println("Nombre completo: [" + creador.getNombre() + " " + creador.getApellidos() + "]" );
    }
}
```

Originator.java (Originador

```java
package Memento;

public class Originator
{
    private String nombre;
    private String apellidos;

    // ---------------------------

    public Originator(String nombre, String apellidos) {
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    // ---------------------------

    public void setMemento(Memento m) {
        this.nombre = m.getNombre();
        this.apellidos = m.getApellidos();
    }

    // ---------------------------

    public Memento createMemento() {
        return new Memento(nombre, apellidos);
    }

    // ---------------------------

    public String getNombre() {
        return this.nombre;
    }

    // ---------------------------

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // ---------------------------

    public String getApellidos() {
        return this.apellidos;
    }

    // ---------------------------

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
```

Memento.java

```java
public class Memento
{
    private String nombre;
    private String apellidos;

    // ---------------------------

    public Memento(String nombre, String apellidos) {
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    // ---------------------------

    public String getNombre() {
        return this.nombre;
    }

    // ---------------------------

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // ---------------------------

    public String getApellidos() {
        return this.apellidos;
    }

    // ---------------------------

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
```

Caretaker.java (Conserje)

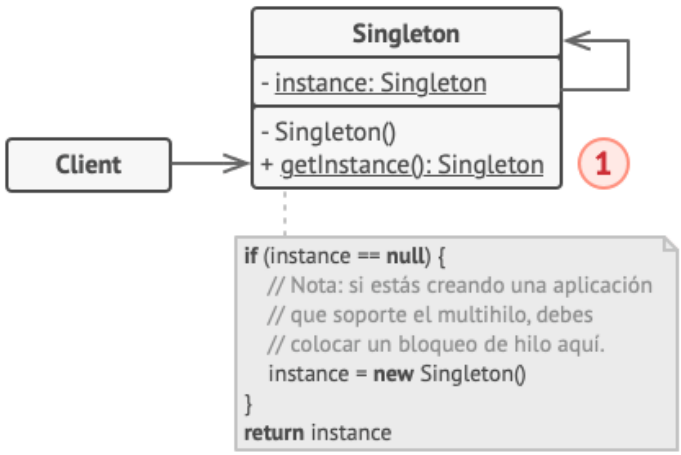
```java
package Memento;

public class Caretaker
{
    private Memento memento;

    // ---------------------------

    public void setMemento(Memento memento) {
        this.memento = memento;
    }

    // ---------------------------

    public Memento getMemento() {
        return this.memento;
    }
}
```

Al inicio del programa creamos un objeto de tipo Originator (que deseamos poder restaurar si se producen cambios) y otro de tipo Caretaker que utilizaremos a su vez para gestionar otro de tipo Memento (en el cual quedará registrado el estado anterior del primero de ellos, y que utilizaremos para restaurarlo).

A continuación, se muestran los datos iniciales del objeto de tipo Originator, y tras modificarlos se volverán a restaurar al estado inicial.

Singleton

Encapsula el proceso de creación de objetos a una función global para evitar que se creen más objetos y comprueba si el objeto existe y se tiene mayor control sobre las instancias que se generaron



En este ejemplo, la clase de conexión de la base de datos actúa como Singleton. Esta clase no tiene un constructor público, por lo que la única manera de obtener su objeto es invocando el método obtenerInstancia. Este método almacena en caché el primer objeto creado y lo devuelve en todas las llamadas siguientes.

Ejemplos:

Ejem#1

```
// La clase Base de datos define el método `obtenerInstancia`
// que permite a los clientes acceder a la misma instancia de
// una conexión de la base de datos a través del programa.
class Database is
    // El campo para almacenar la instancia singleton debe
    // declararse estático.
    private static field instance: Database

    // El constructor del singleton siempre debe ser privado
    // para evitar llamadas de construcción directas con el
    // operador `new`.
    private constructor Database() is
        // Algún código de inicialización, como la propia
        // conexión al servidor de una base de datos.
        // ...

    // El método estático que controla el acceso a la instancia
    // singleton.
    public static method getInstance() is
        if (Database.instance == null) then
            acquireThreadLock() and then
                // Garantiza que la instancia aún no se ha
                // inicializado por otro hilo mientras ésta ha
                // estado esperando el desbloqueo.
                if (Database.instance == null) then
                    Database.instance = new Database()
        return Database.instance

    // Por último, cualquier singleton debe definir cierta
    // lógica de negocio que pueda ejecutarse en su instancia.
    public method query(sql) is
        // Por ejemplo, todas las consultas a la base de datos
        // de una aplicación pasan por este método. Por lo
        // tanto, aquí puedes colocar lógica de regularización
        // (throttling) o de envío a la memoria caché.
        // ...

class Application is
    method main() is
        Database foo = Database.getInstance()
        foo.query("SELECT ...")
        // ...
        Database bar = Database.getInstance()
        bar.query("SELECT ...")
        // La variable `bar` contendrá el mismo objeto que la
        // variable `foo`.
```

Ejem#2

```
package main;

public class Main {

    public static void main(String[] args) {
        // TODO code application logic here
        InstitutoEducativo inst = InstitutoEducativo.getInstance();
        System.out.println("1:"+inst.getNombreInstituto());

        InstitutoEducativo inst2 = new InstitutoEducativo();
        System.out.println("2:"+inst2.getNombreInstituto());

        InstitutoEducativo inst3 = new InstitutoEducativo();
        System.out.println("3:"+inst3.getNombreInstituto());

        inst3.setNombreInstituto("capacitacion S.A.");

         System.out.println("1 bis:"+inst.getNombreInstituto());
    }

}
```

```java
package main;
public class InstitutoEducativo {
    private static InstitutoEducativo instance;
    private String nombreInstituto;

    InstitutoEducativo(){
    }

    public static InstitutoEducativo getInstance() {
        if (instance == null){
            instance = new InstitutoEducativo();
        }
        return instance;
    }

    public String getNombreInstituto() {
        return nombreInstituto;
    }

    public void setNombreInstituto(String nombreInstituto) {
        this.nombreInstituto = nombreInstituto;
    }
```