**TALLER 5**

Unity-Dessign-Pattern es un repositorio que contiene implementaciones de diversos patrones en Unity3D C#, en este caso la implementación y estructura del patrón de comportamiento Memento.

Por motivos de practicidad, vamos a enfocarnos en el ejemplo 2[[1]](#footnote-1). El proyecto funciona similar a un editor de texto que guarda artículos que son ingresados. Sin embargo ¿Qué pasa si el usuario desea recuperar una versión anterior de un artículo?

**Implementación:**

Observando el siguiente fragmento del ejemplo, muestra la creación de la clase MementoExample2, la cual cuenta con los atributos: caretaker, originator, savedFiles y currentArticle. Adicionalmente, la clase cuenta con tres métodos, Save, Undo y Redo, los cuales actúan virtualmente como botones.

**namespace** **MementoExample2**

{

**public** **class** **MementoExample2** : MonoBehaviour

{

Caretaker caretaker = **new** Caretaker();

Originator originator = **new** Originator();

**int** savedFiles = **0**, currentArticle = **0**;

**void** **Start**()

{

// here we do some virtual typing and saving texts:

Save("Tex1: Hello World, this is text example 1");

Save("Text2: Ok here comes example number 2.");

Save("Text3: And example number 3. Just testing.");

Save("Text4: ....");

// Here we do some virtual button pressing

Debug.Log("Pressing Undo");

Undo();

Debug.Log("Pressing Undo");

Undo();

Debug.Log("Pressing Undo");

Undo();

Debug.Log("Pressing Redo");

Redo();

}

// these methods below might get called when someone is pressing a button

// you could easily implement it with unitys new ui system :)

**public** **void** **Save**(**string** text)

{

originator.Set(text);

caretaker.Add(originator.StoreInMemento());

savedFiles = caretaker.GetCountOfSavedArticles();

currentArticle = savedFiles;

}

**public** **string** **Undo**()

{

**if** (currentArticle > **0**)

currentArticle -= **1**;

Memento prev = caretaker.Get(currentArticle);

**string** prevArticle = originator.RestoreFromMemento(prev);

**return** prevArticle;

}

**public** **string** **Redo**()

{

**if** (currentArticle < savedFiles)

currentArticle += **1**;

Memento next = caretaker.Get(currentArticle);

**string** nextArticle = originator.RestoreFromMemento(next);

**return** nextArticle;

}

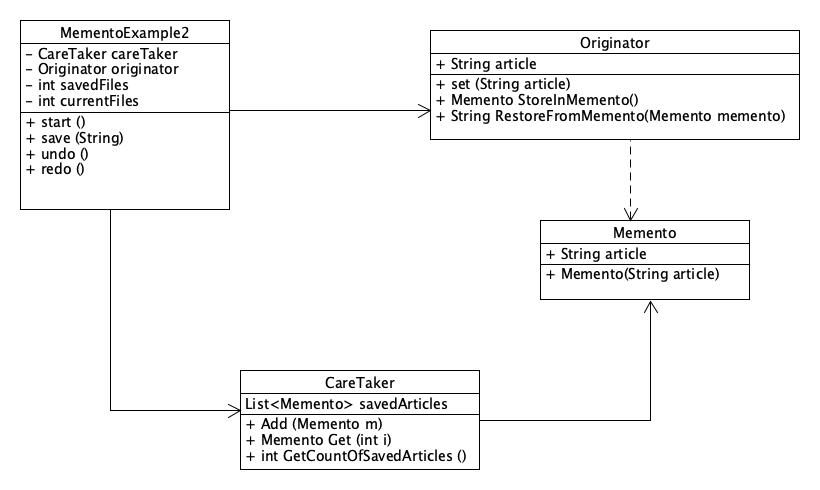
}

}

La implementación del patrón Memento se hace evidente con la existencia de atributos y objetos como Originator, Caretaker y Memento.

**¿Cómo funciona?**

Memento, es un patrón de diseño que se enfoca en recuperar versiones previas de un objeto sin revelar los detalles de su implementación, problema que surge por fallas en el encapsulamiento. La responsabilidad de guardar las versiones es delegada al Originator, el cual es el único objeto que tiene total acceso al estado actual, de esta forma otros objetos no tienen que intentar copiar el estado de originator al que no tienen acceso completo. Estas copias son guardadas en otro objeto llamado Menento, restringe el acceso a la información guardada a cualquier clase que no sea originator. Memento solo conoce a dos objetos, aquel que lo produce y Caretaker, que tiene un acceso limitado a almacenamiento. Caretaker se encarga de mantener la información segura, además puede conocer datos sobre la versión guardada como su nombre u hora de creación, pero nunca va a tener acceso al contenido almacenado.



Para este caso, en el método Save() originator recibe un String al cual se le hace una copia de estado usando un Memento y dicha copia es almacenada en caretaker. Además, actualiza los parámetros savedFiles y currentArticle. Para recuperar una versión se usa Undo() y para deshacer un cambio hecho por Undo() se utiliza Redo (), la implementación de estos métodos es similar y lo que hay que notar es como originator es quien finalmente realiza los cambios sobre las versiones y careTaker solo se encarga de guardar los mementos y administrarlos.

**Ventajas y Desventajas:**

Aunque la aplicación del patrón se puede ver trivial en este proyecto, Memento tiene buena utilidad en diversos proyectos donde se maneje y altere constantemente información y se requiera tener registro de esto. Su aplicabilidad está asociada a editores de texto que desean recuperar una versión anterior, sistemas de negocios o bancarios que necesiten revisar operaciones, desarrollo de juegos que necesitan recuperar puntos de partida, entre otros.

Sin embargo, este patrón cuenta con algunas desventajas. Primero, su implementación puede llegar a ser potencialmente costosa, esto se debe a que Originator podría eventualmente encontrarse con un evento en el que tenga que copiar grandes cantidades de información o porque el usuario constantemente está recuperando y guardando versiones. Además, dado que Caretaker no conoce a la versión, tiende a usar grandes espacios de almacenamiento para guardar un memento aun cuando no es necesario. Otro problema con el que se encuentra Memento es la dificultad de su implementación en algunos lenguajes, problema que como consecuencia no se puede garantizar la seguridad de acceso a las versiones.

**Alternativas:**

Otra forma de solucionar los problemas abarcados por este patrón podría ser mantener un registro de los cambios realizados a un objeto, aunque esto hace la restauración más compleja, ayuda a reducir la cantidad de almacenamiento utilizado.

1. [Assets/Behavioral Patterns/Memento Pattern/Example2/MementoExample2.cs](https://github.com/QianMo/Unity-Design-Pattern/blob/master/Assets/Behavioral%20Patterns/Memento%20Pattern/Example2/MementoExample2.cs) [↑](#footnote-ref-1)