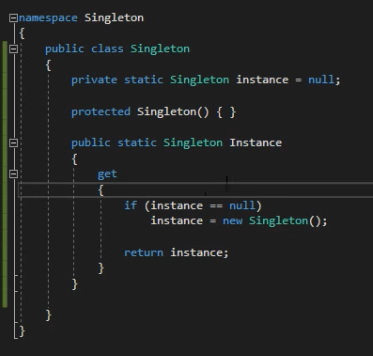
**Patrones de diseño**

**SINGLETON**



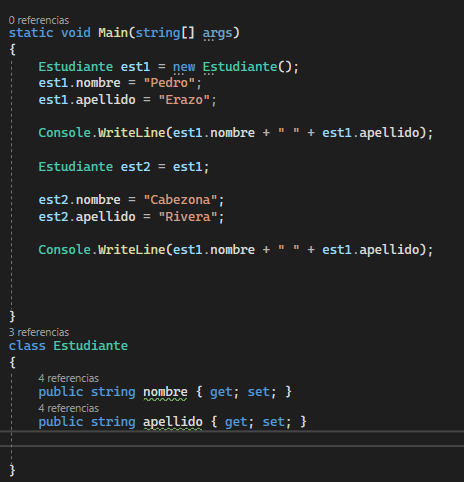
Creamos una clase la cual lleva por nombre el patrón de diseño que estamos manejando, creamos un atributo de la misma con el valor de NULL, creamos un constructor vacío, finalmente creamos una propiedad del mismo tipo de la clase sobre la que estamos trabajando y dentro de la misma creamos un GET. El GET lo que devuelve en caso de que la instancia sea NULL es la creación de la instancia y finalmente retorna la instancia.

El propósito de este patrón de diseño es que solo se deba crear una única instancia de la clase y que con esta se puedan realizar todos los propósitos que el usuario considere.

**PROTOTYPE**

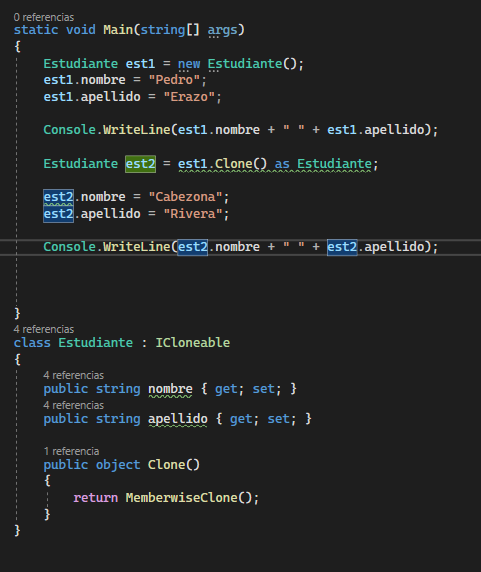
Patrón de diseño que sirve para clonar un objeto.

Tenemos una clase normal, desde MAIN creamos una instancia y le asignamos valores, cada propiedad, finalmente imprimimos los valores, posteriormente creamos otro objeto de la clase y le asignamos el valor del primer objeto al segundo, posteriormente cambiamos los valores al segundo. El problema es que, al cambiar los valores del segundo objeto, también se cambian los del primero. Lo que esperamos de PROTOTYPE es que esto no pase y que se mantengan bien los dos valores.



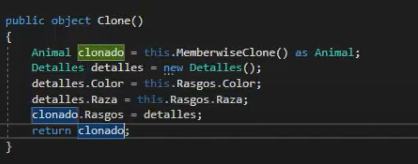
Como clonar

En este caso lo que hemos hecho es utilizar el método CLONE de la clase Estudiante y como tal lo que se hace es un clon del primer objeto, por lo que si cambiamos el segundo objeto, no se va a alterar el primero.



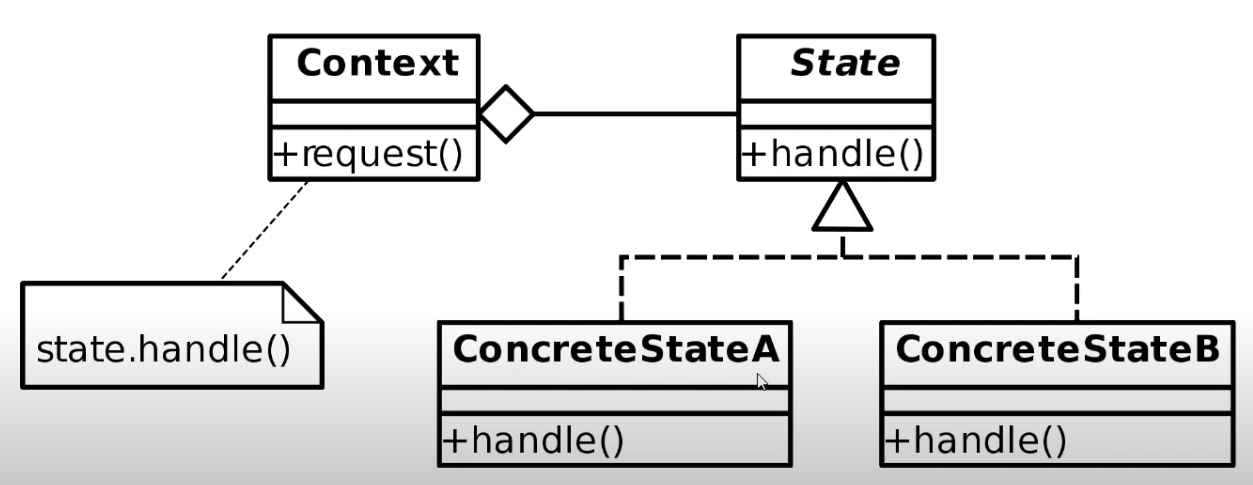
**PROTOTYPE PROFUNDO**

Cuando clonamos, pero el objeto que queremos clonar cuenta con atributos de tipo objeto, la forma anterior de clonar no va a funcionar, por lo que necesitamos aplicar otra forma.

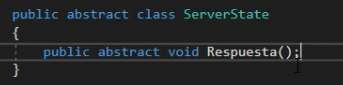


**STRATEGY – MEDIATOR (PENDIENTES)**

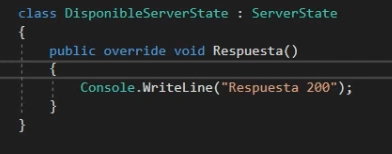
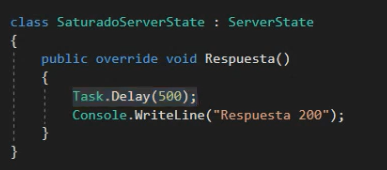
**STATE**

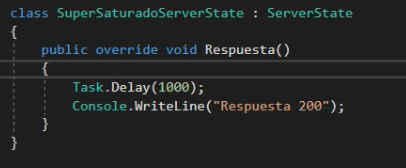
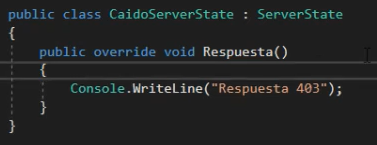


Creamos una clase abstracta

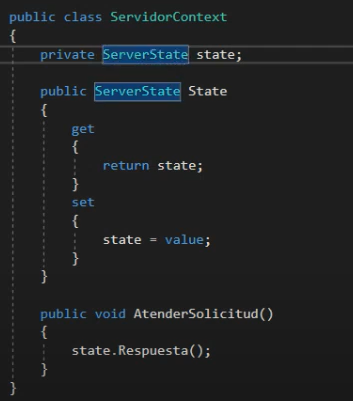


Ahora creamos diferentes clases que simulan el estado del estado

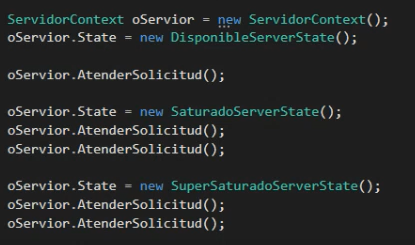
 

Ahora creamos la clase del contexto del servidor



Tenemos un atributo de tipo SERVERSTATE y posteriormente creamos una propiedad para modificar su estado. Creamos un método que tiene como función llamar el método respuesta que tiene el STATE.

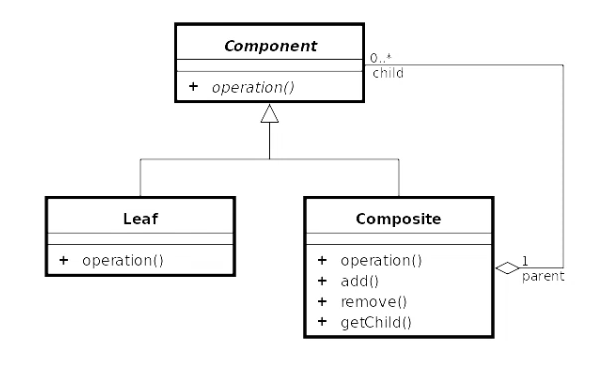
Desde el MAIN



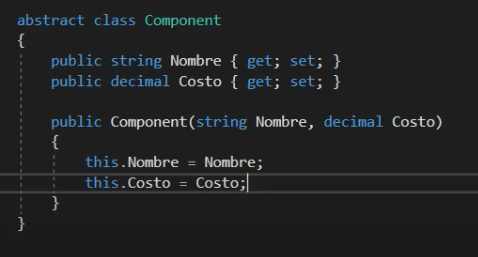
Lo que hacemos es crear un objeto de tipo del contexto del servidor y posteriormente le asignamos distintas instancias de las clases hijas que heredan de esa clase abstracta y finalmente accedemos al método que llama a la respuesta de cada estado.

Se puede notar que el mismo objeto se ajusta para cada estado.

**COMPOSITE**



Creamos una clase abstracta con sus propiedades y su respectivo constructor



Creamos una clase llamada ingredientes la cual hereda de COMPONENT

