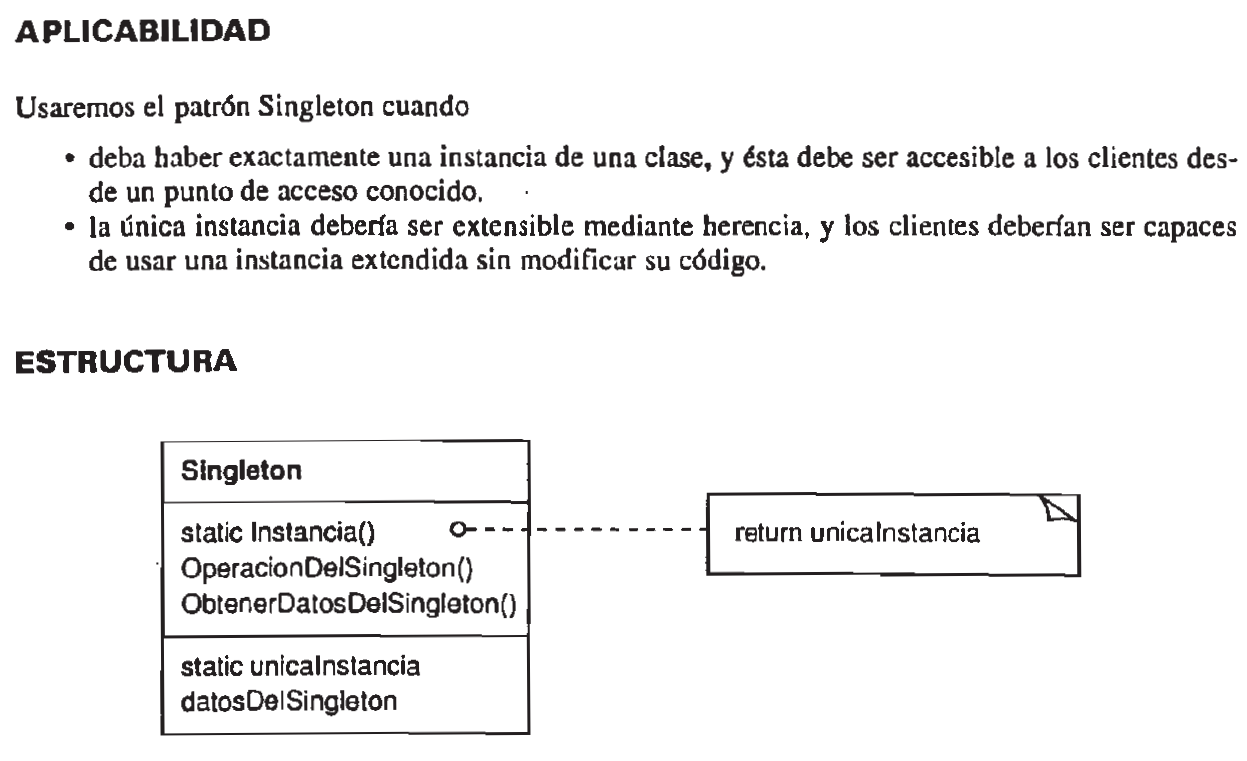
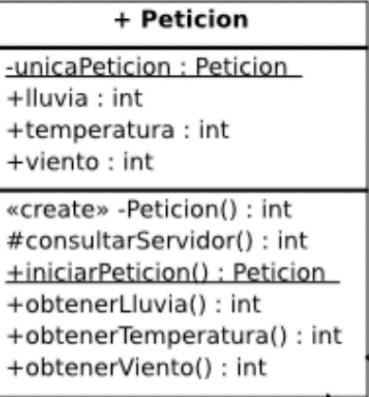
**PATRONES DE DISEÑO**

**La mayoría de los patrones esta en este link, de verdad todo esta muy muy bien explicado y se puede usar el traductor de Google, además contiene ejemplos de código en varios lenguajes**

<https://refactoring.guru/design-patterns/catalog>

**PATRON SINGLETON**



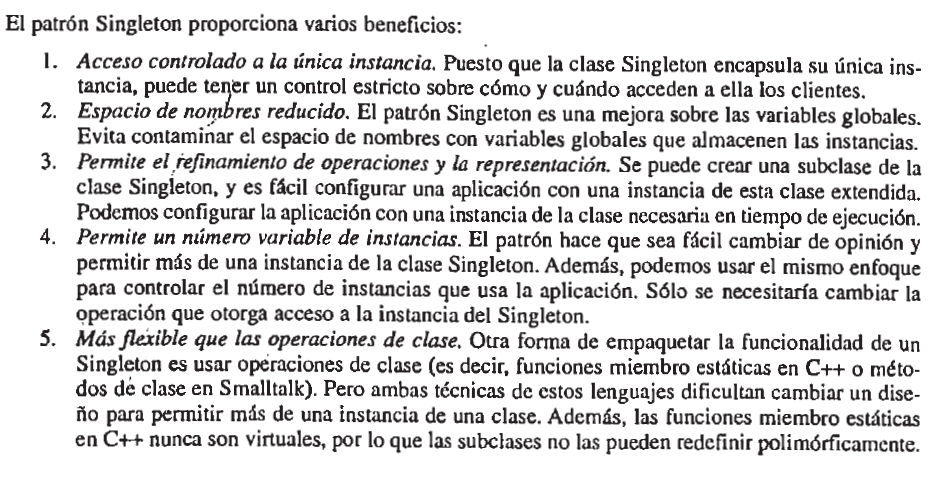
La clase debe de llamarse acorde al problema.

Tiene una propiedad del tipo del mismo nombre de la clase para la instanciación, después de pueden agregar las demás propiedades de la clase.

Debe tener la clase constructor que aca se ve como << Create>>Peticion. Hace referencia al constructor.

Después debe ir el método para “Obtener la instancia” de la clase, a este método es el que accederán todas las demás clases.

Después deben ir las funciones que necesita para la clase



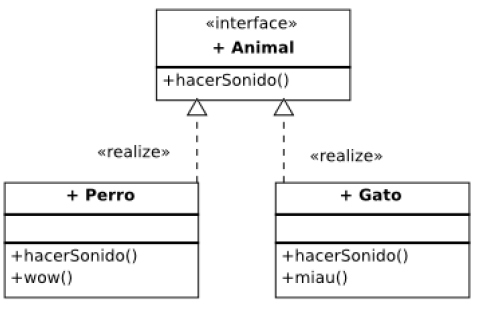
**INTERFACES EN JAVA**

<https://mundojava.net/interfaces.html?Pg=java_inicial_4_5.html>

Las interfaces básicamente son “Características o acciones” Que una clase debe realizar, esto se usa para clasificar el tipo de clases que tenemos, separando las características en común de las características que cada objeto puede tener por aparte

* **NO** debe existir: Propiedades en una interface
* Los métodos **NO** llevan código en la interface, el código se implementa en la clase. Y el método de la interface debe ser public
* Las clases solo pueden heredar o extender una clase a la vez (Por ejemplo, un perro puede heredar nada más la clase padre “Mascota” no podría heredar al mismo tiempo “Mascota” y “Cuadrúpedo”
* Las clases pueden implementar una o varias Interfaces.

**EN UML**

* **Se usa la flecha punteada con la palabra “realize” desde la clase hasta la interface**

**En programación**

En la programación se deben crear como una clase normal solo que se busca la opción de “Interface”.

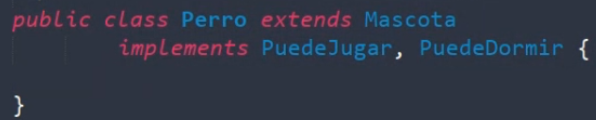
En programación los métodos que tenga la interface no deben tener código

No debe haber ninguna propiedad/ atributo en la interface

Puede haber VARIOS METODOS en una interface.

DECLARACION:

1. En la clase heredada se coloca primero el ***extends***. Este indica que hereda la clase padre
2. Luego se coloca ***implements*** seguido de las interfaces que implementa esa clase



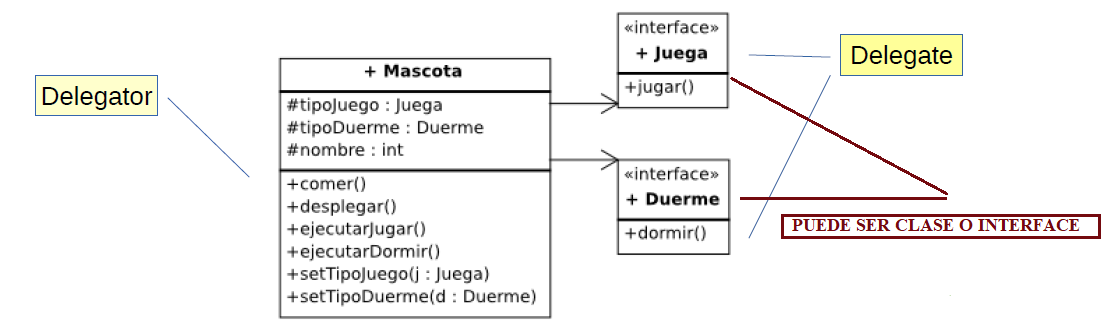
Esto quiere decir que extiende todas las características que tiene la clase padre mascota y sumadas a esas implementa las características de las dos interfaces.

1. Cuando marque el error de java, con el autocompletado implemente todos los métodos abstractos. Y adentro de los corchetes se configura los métodos de la interface

**Implementación de código en la Clase 10/06/20**

**PATRON DELEGATE**

* Delega sus tareas en otro objeto, algunas veces llamado “Helper”.
* Existe un ***Delegator*** (Es la clase principal) y un ***Delegate*** (La clase que hereda las “Responsabilidades” Atributos y métodos.)
* Puede tener una asociación de tipo composición o agregación.
* El ***Delegate*** normalmente es una clase, aunque también puede ser una interface. **OJO** (Es mas normal la clase porque la interface es para el patrón strategy).



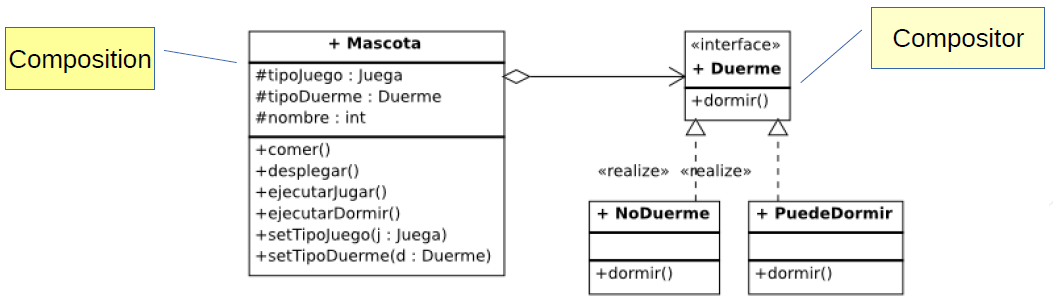
**PATRON STRATEGY**

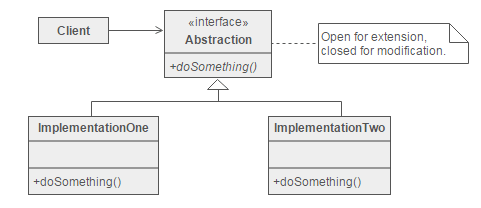
* Es un patrón de diseño de comportamiento que le permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer que sus objetos sean intercambiables
* El patrón de diseño de estrategia es un patrón de diseño de comportamiento en el que elegimos una implementación específica de algoritmo o tarea **en tiempo de ejecución**, de múltiples implementaciones para la misma tarea.
* Este patrón se basa en el principio abierto / cerrado. No necesitamos modificar el contexto [cerrado a modificación], pero podemos elegir y agregar cualquier implementación [abierto para extensión].
* Este patrón define un conjunto de algoritmos relacionados y los encapsula en clases separadas, y permite al cliente elegir cualquier algoritmo en tiempo de ejecución.
* Permite agregar un nuevo algoritmo sin modificar los algoritmos existentes o la clase de contexto, que utiliza algoritmos o estrategias

**Diseño**

En el patrón de strategy, primero creamos una abstracción del algoritmo. Esta debe ser una **interfaz** que tiene la operación abstracta. Luego creamos implementaciones de esta abstracción y estas se llaman *strategies*.

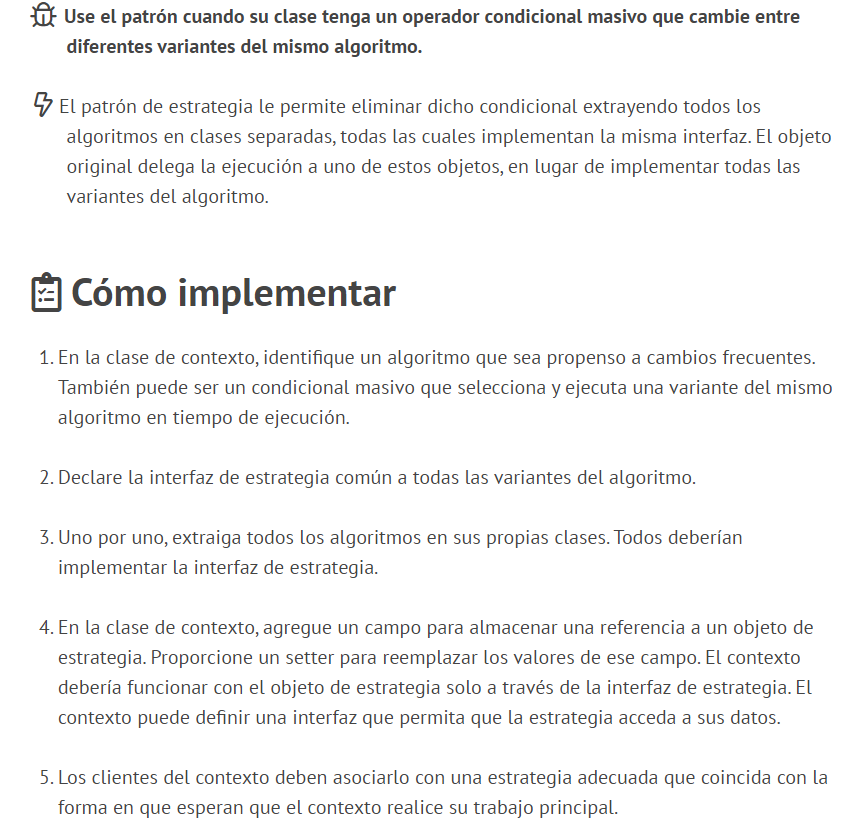
Un cliente siempre llamará a la abstracción (*strategies*) y pasará un objeto de contexto. Este objeto de contexto decidirá qué estrategia usar.





**NOTA**: Las interfaces se pueden cambiar por clases abstractas de manera que no se pueda instancear la clase base, si no solo las herencias de esa clase





Factory Metod