Taller Principios SOLID

# Objetivos:

* Identificar violaciones a los principios SOLID.
* Corregir código que viole principios SOLID.
* Utilizar Git como herramienta de apoyo al trabajo colaborativo.

# Instrucciones

* Trabajar en los grupos del proyecto.
* Usted trabajará con el proyecto en un repositorio de Github. El proyecto está compuesto por 10 paquetes (2 por cada principio, mal y bien).
* Realizar un Fork al repositorio del grupo, desde el repositorio llamado: <https://github.com/djurado/solid>
* Cada integrante debe haber solucionado por lo menos uno de principios y debe verse reflejado en el usuario de los commits al repositorio.
* Los paquetes que contienen **‘mal’** en su nombre, contiene clases e interfaces que viola un principio SOLID. Las tres primeras letras del nombre del paquete le indican el principio que se está violando.
* Analice el código y reestructure las clases, de modo que el principio SOLID ya no sea violado. Agregue su propuesta de solución en sub-paquetes **‘bien’** respectivos.

# Conteste

Explique brevemente como cada principio es violado en el código analizado.

* SRP:

Según el principio SRP cada clase de tener una responsabilidad.

En el código, La clase “Pokemon” tiene dos responsabilidades:

* Manejar la info del Pokemon.
* Curarse.
* OCP:

Al tratar de agregar un nuevo tipo de ataque se debe modificar varias clases: Clase ataque y clase manejadorataque. Esto viola el principio al no extender las clases siendo posible hacerlo. Y también de esta manera se puede modificar viola la otra característica de este principio. De forma general no reutiliza el código.

* LSP:

Según el principio LSP si una función recibe un objeto como parámetro, de tipo X y en su lugar le pasamos otro de tipo Y, que hereda de X, dicha función debe proceder correctamente, entonces cambia la interface a una clase VehiculoAcuatico que tenga el método navegar y de esta superclase hereda Velero y MotoAcuatica, solo MotoAcuatica modifica el método navegar debido a que tiene que estar encendido, de esta manera nos aseguramos de que un mismo método funciona para todos de la manera adecuada.

* ISP:

Según ISP las clases no deben de depender de otras clases o interfaces que no necesitan usar.

En la interfaz usado tenia métodos innecesarios para los Pokemones de diferente tipo, por ejemplo un pokemon de agua no puede aprender un ataque de fuego pero tenia que implementarlo. La solución fue solo dejar los métodos necesarios en la interfaz y agregar los métodos únicos a cada clase correspondiente.

* DIP:

Según el principio DIP las clases mayores no deben depender de las clases menores.

El programador dependía de una computadora con un sistema operativo fijo.

Si queríamos agregar más de un sistema operativo o quitar el que ya estaba establecido, representaría muchas modificaciones en programador, o implementar un programador para cada sistema operativo.

# Entregable

* El enlace al repositorio de GitHub donde colocaron su solución a cada principio y este archivo Word con su análisis grupal.
* Subir el enlace a Sidweb.