ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERIA EN ELECTRICIDAD Y COMPUTACION

**Taller de Principios SOLID – Diseño de Software**

**MSc. David Jurado**

# 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 grupos de máximo 5 integrantes.
* Debe utilizarse un repositorio de Github para este nuevo grupo.
* En el paquete desacargado hay un proyecto compuesto por 10 paquetes (2 por cada principio, mal y bien).
  + Los paquetes que terminan con **‘violation’** 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.
* Dentro de este documento, deben indicar como cada subpaquete está violando el principio SOLID correspondiente. Todos deben estar de acuerdo en el problema y como solucionarlo.
* Cada integrante debe corregir por lo menos un subpaquete, colocando la solución en el subpaquete ‘**solution’** respectivo y subirlo al repositorio remoto de Github con su propio usuario.

# Conteste

Explique brevemente como cada principio es violado en el código analizado y cuál sería la posible solución.

* SRP:
* OCP:
* LSP: Este principio es violado cuando la interface vehículo llama al método escuchar, la clase carro implementa el método de escuchar(), pero no se sabe si la radio esta prendida o apagada, la solución sería manejar la excepción, implementando el método encender.
* ISP: EL problema que viola este principio es que existe una sola interfaz llamada “Maquinaria” la cual contiene métodos, donde algunos no tienen nada de relación. El problema surge al momento de crear una clase que implementa la interfaz, por lo que se ve obligada de implementar cada método, aunque no los necesite. La posible solución es agregar interfaces que implementaran métodos en relación, por lo que al momento de crear una clase, esta implementará las interfaces sólo necesarias.
* DIP: Este principio es violado cuando los carros dependen de las partes específicas en vez de una abstracción de las partes. Igualmente es violado cuando la fábrica general (Interfaz Maquinaria) fabrica todas las partes por separado. La solución implementada sería darle la responsabilidad a los carros de conocer cómo se construyen y pedírselo a la Maquinaria. Para esto se usó la interfaz Ifabricable, que se encarga de agrupar todas las cosas fabricables. De esta manera, si es necesario un “carro tuneado” sólo es necesario crear una nueva clase y describir las partes que utiliza.

# Entregable

* El enlace al repositorio de GitHub con las soluciones y este archivo Word con el análisis grupal.
* Subir el enlace a Sidweb.