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: La clase MáquinaHaceTodo cumple con muchas tareas por lo que tendrá muchas razones para ser cambiada, violando el principio de responsabilidad única. La solución consiste en separar a la clase MáquinaHaceTodo la cual está enfocada en muchas tareas y podría cambiar por muchas razones en otras clases para que realicen una tarea específica y se cumpla el SRP.
* OCP: Dentro de la clase testeadora de giros encontramos varios métodos de giros correspondientes a dos tipos de vehículos, esto rompe el principio de ocp, por lo que se procede a la creación de una interfaz que contenga dichos métodos y unna clase que los implemente por medio de polimorfismo, haciendo de esta clase una general por el cual lleva el nombre de “vehículo” ya que no sabemos el tipo de transporte a usar.
* LSP: En el método escuchar de la clase Carro, existe una condición que verifica si la radio está encendida, pero no hay una procedimiento que siga el programa en caso de que no lo esté. Por ello la solución sería de implementar un método que se encargue de encender la radio, así el método escuchar verifica sin ningún problema el hecho de si está o no encendida la radio.
* ISP: Podemos observar que se crea una interfaz global (Maquinaria), la cual contiene varios métodos que podrían ocasionar conflictos con las clases que los implementen, es decir, si una clase implementa esta interfaz deberá implementar todos los métodos o declararse abstracta pero al hacer esto dicha clase no podrá instanciarse. Una solución a este problema, consiste en crear arias interfaces que me permitan escoger de forma conveniente la que debería implementar para mi clase, asegurándonos de un eficiente trabajo y evitando los posibles conflictos.
* DIP: Los vehículos en este caso bicicleta y carro dependen de sus respectivas llantas para moverse y también tiene sus respectivos asientos, como hay dependencia entre las diferentes clases estamos violando este principio ya que nuestro código no debe depender de implementaciones sino de abstracciones.

Una solución es implementar interfaces que se encarguen de los métodos que se relaciones con todas la clases y cambiar el código para que la dependencia sea desde afuera para desacoplar cualquier clase de cualquier implementación de otro clase diferente.

# Entregable

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