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: MaquinaHaceTodo, propiamente hace de todo, y no tiene una sola responsabilidad, más bien tiene varias, por lo tanto viola este principio. Para esto se utilizó la solución del principio ISP, cada interface de esta solución la implementa una nueva clase que cumple solo con esa responsabilidad. Como en el caso de Bicicleta implementa la interface de MaquinaBicileta y así con Carro y Usuario, dándoles solo una responsabilidad.
* OCP: La clase testeadorDeGiros posee métodos para hacer girar las llantas de un vehículo ya sea hacia la izquierda o derecha. Si se llegará a crear otra clase hija de Vehículo como Triciclo, tenemos que agregar nuevos métodos en testeadorDeGiros, es decir modificarlo.

Para evitar esto, creamos una clase abstracta VehiculosConRuedas con los métodos de girar a izquierda y girar a derecha, del cual extenderán Bicicleta y Carro donde se implementarán respectivamente. De esta forma dejamos cerrado para modificación a testeadorDeGiros y lo hemos abierto para extensión.

* LSP: En la clase testeadorDeAdicionales si se envía un carro y la radio no está encendida, obtendremos una respuesta no deseada, por lo que la clase carro no puede sustituir al padre sin tener un problema. Para solucionar el problema se crea la función encender en la clase radio y en la función escuchar de la clase carro se enciende la radio antes del código de escuchar.
* ISP: Este principio es violado ya que en una sola interfaz se implementan métodos que no todas las clases van a utilizar. Para solucionar este inconveniente se crean tres interfaces diferentes, cada una con métodos que las clases Carro y Bicicleta utilizarán en su totalidad. La interfaz sobrante puede ser utilizada por una clase Usuario ya que son acciones que sólo puede llevar a cabo el mismo.
* DIP: Este principio se ve alterado porque las clases dependen de detalles y no de abstracciones, el asiento de bicicleta y el asiento de carro tienen características y usos comunes (igual las llantas de bicicleta y las llantas de carro), por lo cual deberían heredar sus atributos de una clase padre o de una interfaz, INeumatico para las llantas e IAsiento para los asientos, de esta forma dependen de una abstracción (no de una concreción).

# Entregable

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