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:** Se le está delegando varias responsabilidades a una sola clase, en este caso sería a la clase “MaquinaHaceTodo”, lo cual es incorrecto desde el punto de vista de un programador, una posible solución es separar las responsabilidades(métodos) dentro de la clase en clases que ejecuten los mismos de una manera más específica y entendible, por lo cual se crearon las clases “servicios”, “fabrica” y ”ejecución” para que puedan ser llamadas cuando se las necesite.
* **OCP:** La clase testeadorDeGiros tenía implementado varios métodos para hacer girar carro y bicicleta por lo que, si queremos extender nuestro código y crear nuevos tipos de vehículos como un tractor, no podríamos ya que está definido para carro y bicicleta por lo que se procedió a separar en vehicle y la dirección para así poder crear cualquier tipo de vehículo y se pueda mover a cualquier dirección usando los métodos y sin afectar el código de nuestra clase principal.
* **LSP:** La clase Radio solo tenía método para comprobar que estaba encendido. Sin embargo, si la radio no estaba encendida, el carro no se podía escuchar y eso generaba un problema con el testeador de adicionales. Al hacer que Radio ya se pueda encender si no está encendida, ya funciona normalmente el testeador.
* **ISP:** En la interfaz existían métodos que no debían implementarse en las otras clases. Se creó 2 interfaces más para separar los métodos. Y si una clase necesita métodos de 2 interfaces simplemente implementa ambas.
* DIP: El principio es violado porque se crearon clases para características comunes que podían ser características de un padre y que pase a sus hijos. Se creo una clase padre que sea vehículo y luego se extendía a carro y bicicleta.

# Entregable

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