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 viola el principio de single responsibilty porque la clase ‘MaquinaHaceTodo’ debería enfocarse en hacer una sola responsabilidad y no todo; por lo que la solución seria crear una clase padre ‘MaquinaGenerica’ de la cual heredaran sus clases hijas las cuales tienen una responsabilidad especifica.
* MaquinaHaceCarro se encarga de crear carros
* MaquinaHaceBicicleta se encarga de crear bicicletas
* MaquinaControlaHabitacion gestiona una habitación, por ejemplo: la temperatura, luces.
* MaquinaOficinaUtil se encarga de imprimir informes y servir café.
* OCP: La violacion de este principio ocurre debido a que cuando se desea implementar un nuevo tipo de giro la clase “testeadorDeGiros” se modificara. Para esto es mejor crear una interfaz denominada “TiposGiros” la cual sera implementada por todas las clases de giros para cada vehiculo.
* LSP: La violación de este principio se da al tener una condición en el método escuchar en cuanto a la clase Carro se refiere, dependiendo si esta o no encendida la radio del mismo. A la hora del método testearAdicionalRadio donde se recorren todos los vehículos, el programa tendrá problemas en la clase hijo Carro debido a todo lo mencionado. Su solución debe ser implementar un método que encienda la radio para que así con la condición no se produzcan errores o excepciones. Así, la clase Carro puede comportarse como un vehículo e implementar el método escuchar sin ningún inconveniente.
* ISP: La violación de este principio se da en la interfaz Maquinaria, puesto que es una interfaz demasiado genérica, lo que provoca que las clases que implementan Maquinaria tengan que implementar métodos que no deberían tener. Para solucionar esto, en vez de tener una sola interfaz, se procede a dividir la interfaz genérica que se tiene por diferentes interfaces que son mucho más específicas, las cuales son:
  + IFBicicleta, contiene los métodos relacionados al objeto Bicicleta.
  + IFCarro, contiene los métodos relacionados al objeto Carro.
  + IHabitacion, contiene los métodos relacionados al objeto Habitación.
  + ISistemaElectrico, contiene los métodos relacionados al sistema eléctrico.
  + IUtil, contiene los métodos de servirCafe e imprimirInforme.
* DIP: La violación de este principio se encuentra al que ninguna clase se encuentra relacionada con otra, y el objetivo de este principio es el uso de abstracciones para conseguir que una clase interactúe con otras clases sin que las conozca directamente, por lo que recurrí a crear una clase abstracta padre Vehículo que tenga hijos auto y bici y ellas tendrán sus interfaces.

# Entregable

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