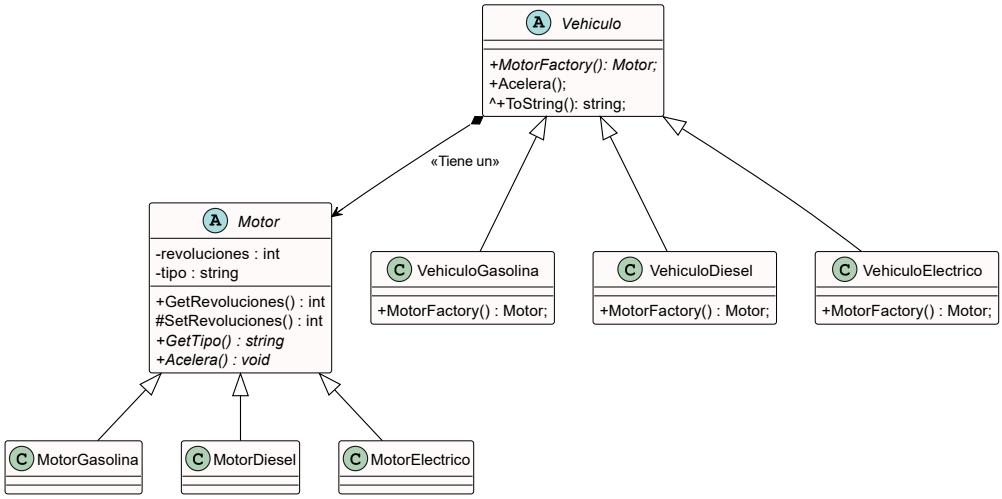
# Ejercicios SOLID y Patrones

Descargar estos ejercicios

## Índice

## Ejercicio 1

Partiendo del siguiente programa en este enlace descarga con el código de ejemplo ampliado de implementación de la factoría de vehículos que implementa el siguiente diagrama.




Donde cada concreción de la factoría de motor `VehiculoGasolina`, `VehiculoDiesel` y `VehiculoElectrico` creaba su motor correspondiente.

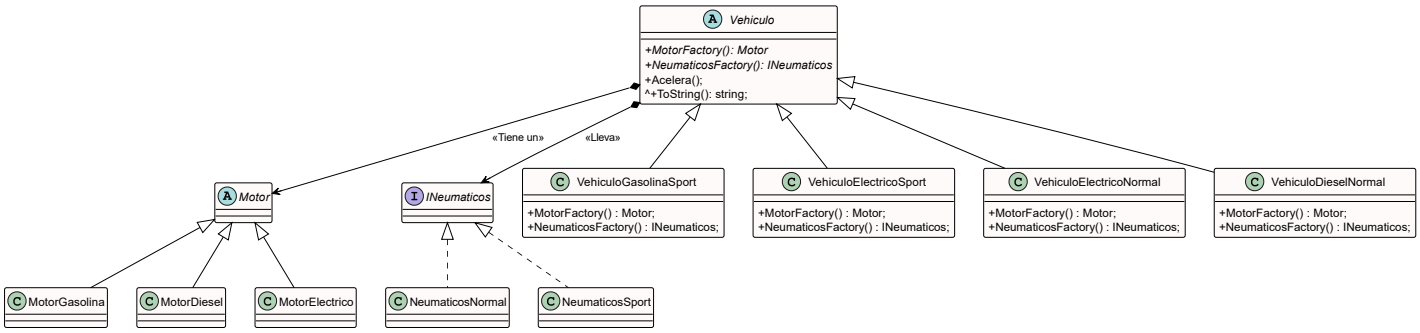
Vamos a modificar el código para que `Vehiculo` tenga dos tipos de factoría:

1. Una para Motores (Lo que ya está implementado)
2. **Otra para Neumáticos**

Para ello partiremos de la siguiente implementación de la abstracción de `Neumatico` que usamos en el ejemplo del patrón *'Fluid Builder'*.

```
public interface INeumaticos
{
    string Descripcion
    => $"{Tipo} {Ancho}/{Perfil} R{Radio} {IndiceCarga}{IndiceVelocidad}";
    string Tipo { get; }
    string IndiceVelocidad { get; }
    int IndiceCarga { get; }
    int Radio { get; }
    int Perfil { get; }
    int Ancho { get; }
}
public class NeumaticosNormal : INeumaticos
{
    public string IndiceVelocidad => "H";
    public int IndiceCarga => 88;
    public int Radio => 16;
    public int Perfil => 55;
    public int Ancho => 205;
    public string Tipo => "Normales";
}
public class NeumaticosSport : INeumaticos
{
    public string IndiceVelocidad => "Y";
    public int IndiceCarga => 92;
    public int Radio => 18;
    public int Perfil => 40;
    public int Ancho => 225;
    public string Tipo => "Sport";
}
```

Al final deberemos llegar al siguiente diagrama...




Ahora tendremos nuevas concreciones de factoría donde se crearán vehículos con una combinación de motor y neumáticos determinada.

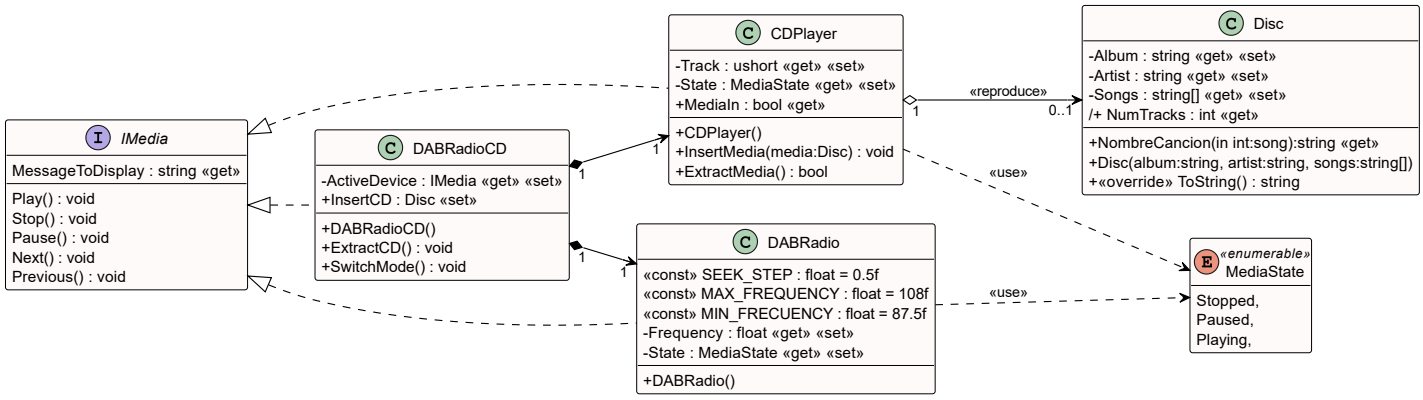
1. Modifica el programa principal del código de partida dado para que se creen de forma aleatoria cada una de estas concreciones.
2. Modifica el método `ToString` de vehículo para que además del tipo de motor, en una línea aparte, se muestre la propiedad `Descripcion` que implementa al abstracción de INeumaticos.

   > 📌 **Nota:** Fíjate que aunque `INeumaticos` es una interfaz la propiedad `Descripcion` tiene una **implementación por defecto**.

## ✅ Ejercicio 2

Vamos a modificar el ejercicio de **DABRadioCD** (debes partir de la propuesta de solución de entrega de interfaces), para que cumpla alguno de los principios SOLID como son el de de **segregación de interfaces** y el de **inversión de dependencias**.

> 📌 **Nota:** Puesto que todos los tipos en nuestra propuesta de solución están en el mismo fuente. Deberás separar cada tipo en un fuente diferente con el nombre apropiado.




Las modificaciones propuestas serán las siguientes:

1. Tenéis el diagrama de clases, será la solución a la que tenéis que llegar. Como podéis ver hay una clase nueva que sirve de Factoría (Será la clase que servirá para inyectar las dependencias) y os la podéis dejar para el final.
2. Las Interfaces se han segregado, de forma que si eres un medio al que se le pueda insertar o extraer información (CD, USB, etc), habrá que heredar de IExtraibleMedia que a su vez hereda de IMeda
3. IExtraibleMedia es una interfaz genérica, de forma que cuando yo herede de ella deberemos indicar que tipo somos (Disc en este primer caso)
4. Uno de los cambios importantes, será que en la clase DABRADIOCD, ahora le llegará toda la información de tipo IMedia (Cumpliendo el principio de Inversión de dependencia), y además vamos a generalizar de forma que en vez de tener dos variables radioCD y dabRadio (como pone en el diagrama) tendremos un array de IMedia, y el activeDevice (este último como siempre), esto será así para poder crearnos DAB con distinto tipo y cantidad de elementos.
   Vale, con esto tendréis que modificar el ejercicio para que se quiten todos los errores y que funcione.
   Recordar que para crear un DABRadioCD utilizaremos la clase Factory, dentro de ella se creará un array de tipo IMedia (con un cd y una radio) y este array será el que se inyecte a la DABRadioCD.
5. Probar que funcione con el programa que ya existe, pero usando la Factory
6. Lo último será añadir una clase USBPlayer, y probar su funcionamiento con un DABRAdio con 3 elementos

## Ejercicio 3

Posible ejercicio de patrón Decorator
Tenemos las clases

Class Tarrina

```
enumerado Tamaño { Pequeña, Mediana, Grande, Cubo }
enumerado Sabor { Vainilla, Chocolate }
enumerado Añadido { Lacasitos, Conguitos }
```

Class Barquillo

```
enumerado Tamaño { Pequeño, Mediano, Grande }
enumerado Sabor { Vainilla, Fresa }
enumerado Cubierta { Chocolate, Caramelo }
```

Si tuviéramos que añadir nuevos sabores o añadidos a la Tarrina parece que definiendo el enumerado fuera de la clase cumpliríamos OCP, aunque no es así ya estaríamos modificando el propio tipo enumerado que debería estar 'cerrado'...

Peor aún, no podríamos añadir cubierta como a los barquillos porque tendríamos que definir una nueva propiedad en la clase Tarrina.

Lo mismo nos sucedería para los barquillos.

Una posible solución sería hacer utilizar el patrón decorator que acabamos de ver pero nos solucionaría el tema de la Tarrina pero tendríamos que crear componentes muy parecidos para el Barquillo.

IMPLEMENTAR SOLUCION CON DECORATOR.....