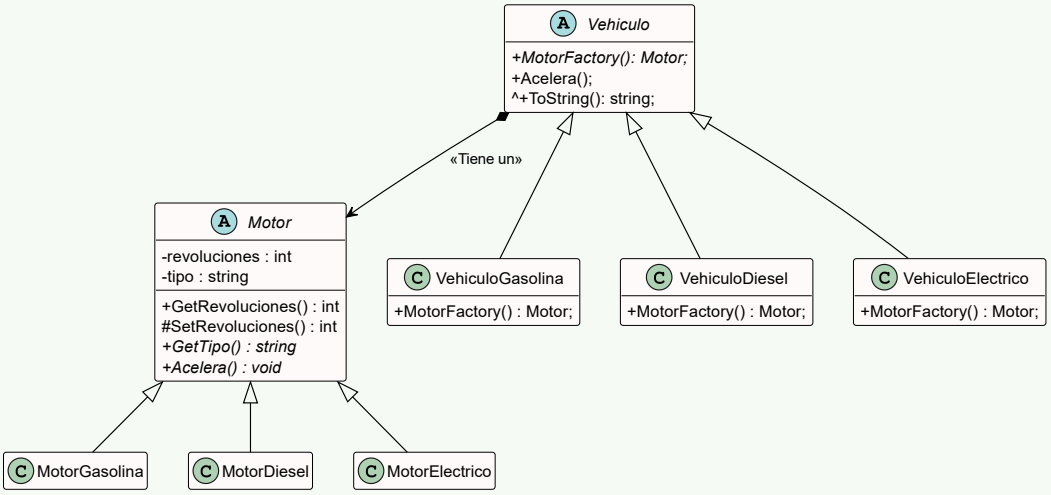
Ejercicios SOLID y Patrones

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**

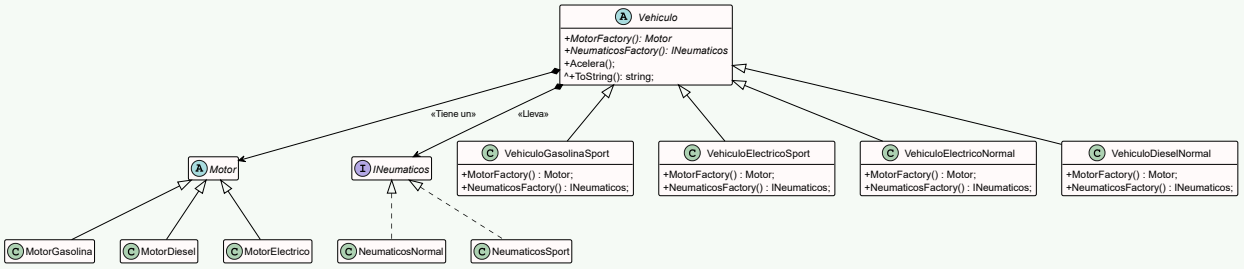
Para ello partiremos de la siguiente implementación de la abstracción de `Neumatico` que usamos en el ejemplo del patrón *'Fluid Builder'*.

```
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.

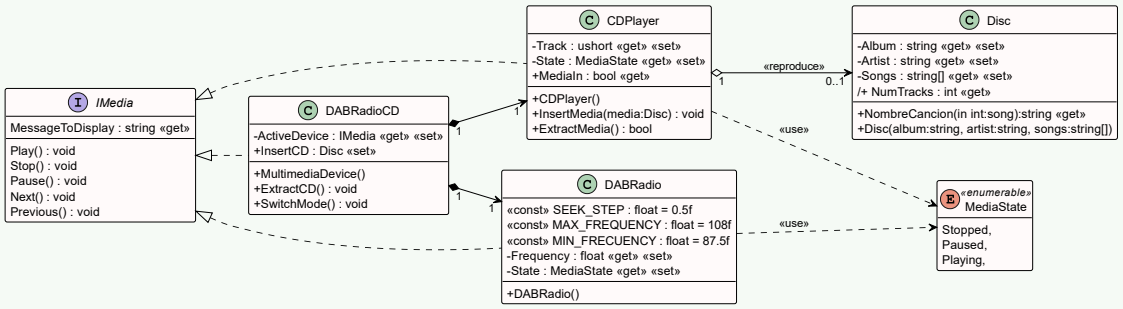
1. Modifica el programa principal del código de partida dado para que se creen de forma aleatoria cada una de estas concreciones.
2. Modifica el método `ToString` de vehículo para que además del tipo de motor, en una línea aparte, se muestre la propiedad `Descripcion` que implementa al abstracción de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

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

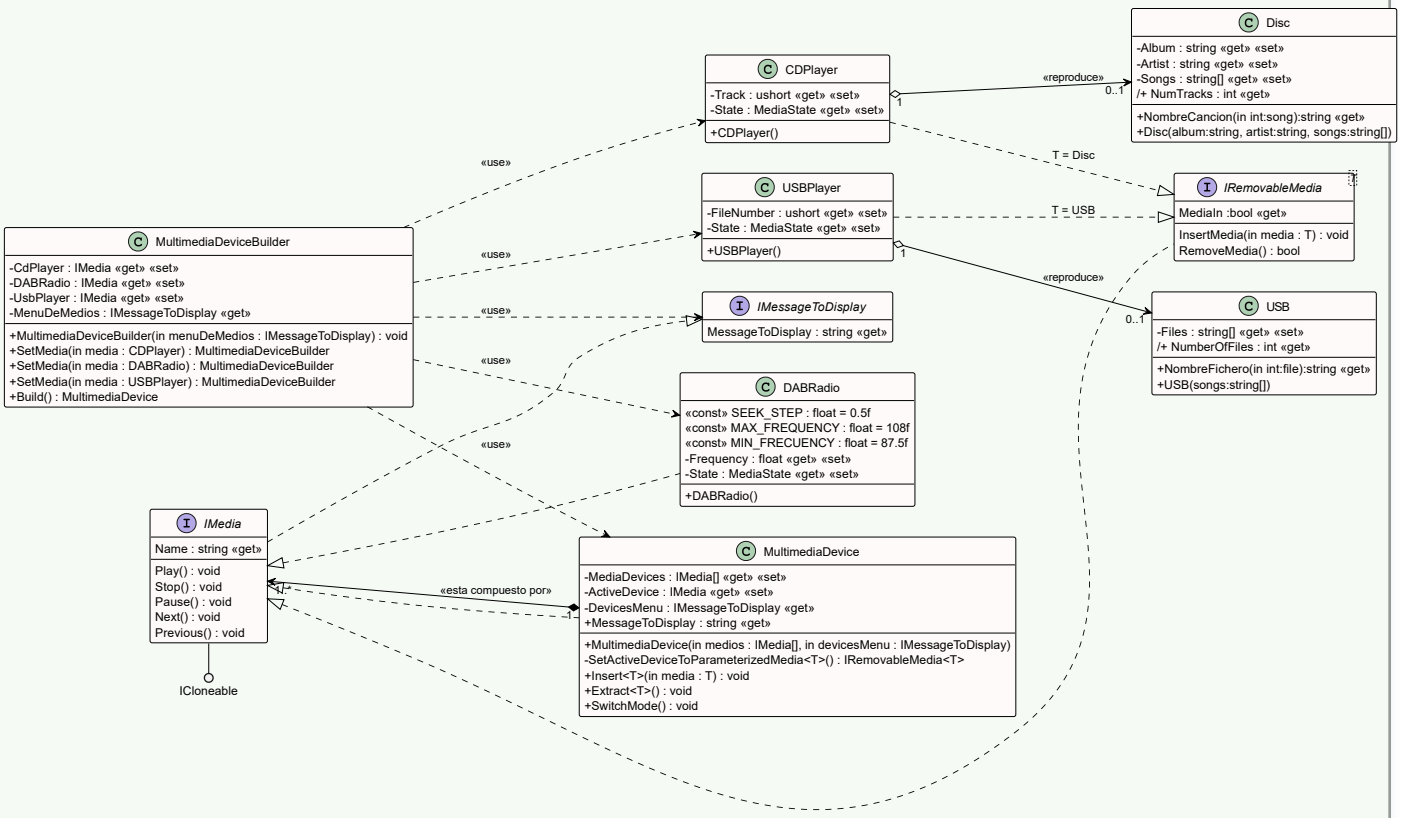
✅ Ejercicio 2

Vamos a modificar el ejercicio de **MultimediaDevice** (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.



Vamos a ampliar las especificaciones, refactorizando el código existente aplicando **SOLID**, para llegar al siguiente diagrama de clases UML (Global), que puedes consultar si no entiendes alguna de las especificaciones dada y donde hemos quitado las relaciones de `<<use>>` salvo en el caso de la clase `MultimediaDeviceBuilder`.



> ✋ **Importante:** Para la corrección de este ejercicio deberemos crear un repositorio de **git local** e ir haciendo un `commit` tras finalizar la propuesta en cada uno de los puntos numerados. En caso contrario no se evaluará su corrección.

Los pasos para hacer la refactorización pueden ser los siguientes:

1. Fíjate que hemos extraído el la propiedad `string MessageToDisplay { get; }` del interfaz `IMedia` a un nuevo interfaz denominado `IMessageToDisplay` siguiendo el *'Principio de Segregación de Interfaces'* y haremos que `IMedia` **herede de**

`IMessageToDisplay`. Además, hemos añadido en `IMedia` la propiedad `string Name { get; }` que simplemente me devolverá el nombre del medio de reproducción.

2. Vamos a extraer a un interfaz las características de que el medio de reproducción `CDPlayer` se pueda extraer, insertar un CD y también la propiedad de saber si hay un medio insertado o no. Pero..., no las podemos extraer a `IMedia` porque no todos los medios de reproducción las tienen ya que, por ejemplo, la DABRadio no las tiene. Por tanto, para cumplir de nuevo el *'Principio de Segregación de Interfaces'* definiremos el interfaz `interface IRemovableMedia<T> : IMedia` que heredará el interfaz de `IMedia` más las características de un reproductor de medios extradirles descritas en el diagrama.
   Además, por lógica, el método `Clone()` que implementemos en cada reproductor de medios extraibles no deberá hacer una copia del medio extraible que contenga. Esto es, si tenemos un reproductor de CD que tiene un CD metido, lo clonaremos sin el CD que contiene.

   > 📌 **Nota** Fíjate que es un interfaz parametrizado con `T`. Siendo `T` el tipo del medio extraible, en el caso de `CDPlayer` será un `CD`

3. Vamos ha hacer ahora que CDPLayer ya no implemente `IMedia` sino `IRemovableMedia<Disc>`.
4. Aprovechando que tenemos el nuevo interfaz de medios extraibles. Vamos a definir uno nuevo de forma análoga al `CDPlayer`. En este caso será un `USBPlayer` que implementará `IRemovableMedia<USB>` como se indica. Pero antes, debes definir la clase `USB` que contendrá un array de ficheros de audio reproducibles como se indica. Piensa cómo será la implementación teniendo en cuenta la de `Disc` y la definición del UML.
5. Vamos ahora a hacer cambios importantes en la clase que representa a **nuestro dispositivo** `DABRadioCD`.
   i. En primer lugar el nombre es **bastante 'desafortunado'** porque ahora debería llamarse `DABRadioCDUSB`. EN lugar de eso fíjate que ahora la hemos llamado `MultimediaDevice` de esta manera nos abstraemos de futuras ampliaciones.
   ii. Cómo teníamos una composición, creábamos los objetos en el constructor pero para cumplir el *'Principio de Inversión de Dependencias*' ahora los recibiremos como parámetro por el constructor. Además, para cumplir el *'Principio de Abierta-Cerrada*' y dejar la definición de la clase cerrada a futuras modificaciones. No pasaremos al constructor un `CDPlayer`, `DABRadio` y ahora al ampliar un `USBPlayer`. En su lugar pasaremos al constructor una `IMedia[] medios` que es una colección de la abstracción de medios con todos los que van a componer nuestro `MultimediaDevice` ahora y en futuras ampliaciones. Además, también recibirá una abstracción `IMessageToDisplay devicesMenu` con el menú de opciones que se corresponde con los medios que estamos recibiendo.
   iii. El cambios de 5.2 va a suponer replantearnos algunos métodos dentro de `MultimediaDevice` (fíjate bien en los interfaces descritos en el UML). Por ejemplo...
      - El constructor generará una excepción si tenemos menos de un medio en el array y marcará como `ActiveDevice` el primer medio del array.
      - `MessageToDisplay` usará la propiedad `Name` y `MessageToDisplay` de los `IMedia` para mostrar información del modo y el dispositivo activo y mostrará el `MessageToDisplay` del `DevicesMenu` recibido en el constructor.
      - 💡 Fíjate que hemos definido un método privado parametrizado `SetActiveDeviceToParameterizedMedia<T>` que pondrá o '*seteará*' como `ActiveDevice` el primer medio que encuentre en el array `IMedia[]` que tenga el medio extraible indicado en `T`. Esto es, el primer medio que cumpla que es un `ActiveDevice` es un `IRemovableMedia<T>`. En caso de no encontrarlo generaremos una excepción.
      - `void Insert<T>(T media)` Llamará a `SetActiveDeviceToParameterizedMedia` para cambiar al reproductor del medio que acabamos de insertar y como ya hacíamos generaremos una excepción si ya tenía un medio insertado. EN caso contrario lo insertaremos y activaremos el `Play()`.
      - `void Extract<T>()` Llamará a `SetActiveDeviceToParameterizedMedia` para cambiar al reproductor del medio y extraerá el medio que tuviese.
      - `SwitchMode()` tomará el siguiente medio al activo en el array de `IMedia[]` o el primero si el activo fuera el último.
6. Por último vamos a implementar el patrón creacional '*Fluent Builder*' para crear las instancias de nuestro `MultimediaDevice`. Fíjate en la definición y cómo tiene relaciones de uso con el resto de clases involucradas estando arriba en la jerarquía de dependencias.

   > ✋ **Importante:** EL método `Build()` hará un `Clone()` de cada uno de los `IMedia` antes de pasárselos al `MultimediaDevice` para asegurarnos de que se cumple la composición.

Si todo está bien implementado correctamente el siguiente programa principal debe ejecutar correctamente.
Si todo está bien implementado correctamente el siguiente programa principal debe ejecutar correctamente.

```
class Program
{
    private class MenuDispositivoMultimedia : IMessageToDisplay
    {
        public string MessageToDisplay =>
                "[1]Play \n" + "[2]Pause \n" +
                "[3]Stop \n" + "[4]Prev \n" + "[5]Next \n" +
                "[6]Switch \n" + "[7]Insert CD \n" +
                "[8]Extract CD \n" + "[9]Insert USB \n" +
                "[10]Extract USB \n" + "[13]Turn off";
    }

    public static void Main()
    {
        string[] cancionesEnCD = {
                "Wanna Be Startin' Somethin",
                "Baby Be Mine", "The Girl Is Mine", "Thriller", "Beat It",
                "Billie Jean", "Human Nature",
                "P.Y.T. (Pretty Young Thing)", "The Lady in My Life"
            };
        string[] ficherosEnUSB = {
                "One After 909_Let It Be_The Beatles.mp3",
                "Rocker_Let It Be_The Beatles.mp3",
                "Save the Last Dance for Me_Let It Be_The Beatles.mp3",
                "Don't Let Me Down_Let It Be_The Beatles.mp3",
                "Dig a Pony_Let It Be_The Beatles.mp3",
                "I've Got a Feeling_Let It Be_The Beatles.mp3",
                "Get Back_Let It Be_The Beatles.mp3"
            };

        Disc CD_DeThriller = new("Thriller", "Michael Jackson", cancionesEnCD);
        USB USB_DeThriller = new(ficherosEnUSB);
        MultimediaDevice dispositivoMultimedia =
                            new MultimediaDeviceBuilder(new MenuDispositivoMultimedia())
                            .SetMedia(new CDPlayer())
                            .SetMedia(new DABRadio())
                            .SetMedia(new USBPlayer())
                            .Build();

        int opcion = int.MaxValue;
        do
        {
            try
            {
                Console.WriteLine(dispositivoMultimedia.MessageToDisplay);
                Console.Write("Escoge una opción: ");
                opcion = int.Parse(Console.ReadLine() ?? "11");
                Console.Clear();
                switch (opcion)
                {
                    case 1: dispositivoMultimedia.Play(); break;
                    case 2: dispositivoMultimedia.Pause(); break;
                    case 3: dispositivoMultimedia.Stop(); break;
                    case 4: dispositivoMultimedia.Previous(); break;
                    case 5: dispositivoMultimedia.Next(); break;
                    case 6: dispositivoMultimedia.SwitchMode(); break;
                    case 7: dispositivoMultimedia.Insert(CD_DeThriller); break;
                    case 8: dispositivoMultimedia.Extract<Disc>(); break;
                    case 9: dispositivoMultimedia.Insert(USB_DeThriller); break;
```

```
                    case 10: dispositivoMultimedia.Extract<USB>(); break;
                    default:
                        break;
                }
            }
            catch (Exception e) { Console.WriteLine(e.Message); }
        } while (opcion > 0 && opcion < 11);
    }
}
```