

Ejercicios SOLID y Patrones

[Descargar estos ejercicios](#)

Índice

1.  [Ejercicio 1](#)
2. [Ejercicio 2](#)

✓ Ejercicio 1

Vamos a modificar el ejercicio de **DABRadioCD** (debes partir de la propuesta de solución del bloque anterior), para que cumpla alguno de los principios SOLID como son el 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 2

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.....