Ejercicios Interfaces y Genericos

Descargar estos ejercicios

Índice

Ejercicio 1

> Para entender mejor las **Interfaces**, a continuación se muestra un ejemplo sencillo. Fíjate, sobre todo, en el programa principal para comprender mejor su utilidad.

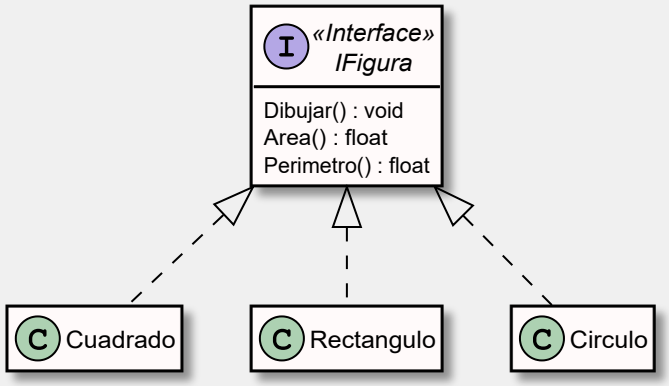
```
interface IVisualiza
{
   void Visualiza();
}
class Triangulo : IVisualiza
{
   private double @base;
   private double altura;
   public Triangulo(double base_, double altura)
   {
       @base = base_;
       this.altura = altura;
   }
   private double area
   {
       get { return @base * altura / 2; }
   }
   public void Visualiza()
   {
       Console.WriteLine($"Base del triángulo: {@base}");
       Console.WriteLine($"Altura del triángulo: {altura}");
       Console.WriteLine($"Área del triángulo: {area}");
   }
}
class Proveedor : IVisualiza
{
   private string nombre;
   private string apellidos;
  public Proveedor(string nombre, string apellidos)
   {
       this.nombre = nombre;
       this.apellidos = apellidos;
   }
   public void Visualiza()
   {
       Console.WriteLine($"Nombre: {nombre}");
       Console.WriteLine($"Apellidos: {apellidos}");
   }
}
class EjemploInterfacesApp
{
   static void VerDatos(IVisualiza oVisualizable)
   {
       oVisualizable.Visualiza();
   }
   static void Main()
   {
       Triangulo t = new Triangulo(10, 5);
       VerDatos(t);
       Proveedor p = new Proveedor("Erik", "Erik otra vez");
```

```
            VerDatos(p);
        }
    }
```

Nota: Crea una librería de interfaces llamada **MisInterfaces** a la que le irás añadiendo todas las interfaces que implementes en los ejercicios. Para poder usarla deberás incluirla en tus proyectos.

Ejercicio 2

A partir del siguiente UML crea las clases e Interfaces necesarias para implementar un editor de figuras geométricas. Crea además, un programa principal que te permita probarlo correctamente.



Ejercicio 3

Tendremos una clase **Estudios** que implementará los interfaces **IEstudios** e **IVisualiza**. La interfaz IEstudios deberá asegurar que las clases que deriven de ella, implementen un método que muestre por pantalla la edad mínima para empezar esos estudios.
Al implementar la interfaz IVisualiza (del ejercicio 1), en el método se mostrará por pantalla la información relativa a los estudios en curso.
La clase Estudios **no podrá ser instanciada** y derivarán de ella las clases **Superior, Medio y Elemental**: los estudios superiores, medios y elementales, tendrán como edades de acceso 18, 16 y 12 años respectivamente.

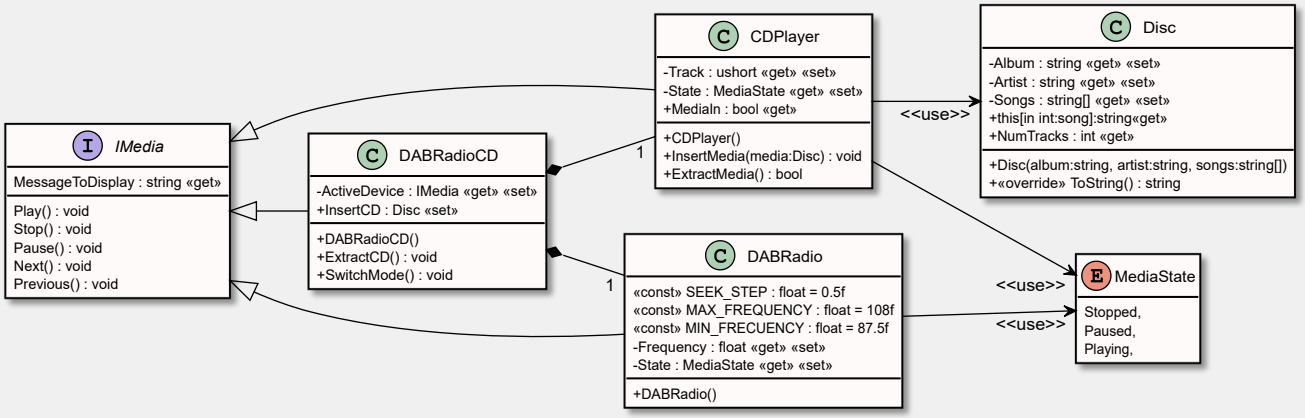
- Para todos los estudios, nos interesará ver en pantalla su nombre y duración.
- Para los estudios superiores, nos interesará el lugar donde se realizan.
- Para los Medios nos interesa ver por pantalla el nombre de un estudio superior al que den acceso.

Crea al menos un objeto de cada una de las clases y comprueba su funcionamiento. Comprueba si las clases derivadas de Estudios son a su vez derivadas de **IEstudios y IVisualiza**.

✅ Ejercicio 4

Vamos a diseñar las clases para un posible sistema operativo de una radio con DAB (Digital Audio Broadcasting) y un reproductor de CD.

Para ello, seguiremos el modelado propuesto en diagrama de clases del ejemplo teniendo en cuenta las siguientes especificaciones funcionales.



Tendremos una clase **DABRadioCD** que estará compuesta por dos dispositivos un reproductor de **CD** y un sintonizador **DAB**.
En el reproductor de CD además podremos tener un Compat Disc ® representado por la clase **Disc**.
La clase Disc tendrá un indizador que me permitirá acceder al título de cada canción y una sobreescritura de ToString que me permitirá ver el nombre del álbum y el artista de la canción.

Nota: Recuerda que en C# los Get (Accesores), Set (Mutadores) son Propiedades y que los campos se pueden implementar a través de **Propiedades Autoimplementadas**.

El reproductor de CD implementa la interfaz **IMedia** con la funcionalidad:

- **MessageToDisplay**: Propiedad que devuelve un mensaje para el Display del DABRadioCD con el estado del reproductor. Devolviendo NO DISC si no hay un disco en su interior. Además, en este caso el resto de opciones de reproducción deberían devolver el mismo mensaje, no teniendo efecto.

- **Play**: Que reproducirá el disco desde la pista `1` , si el reproductor está parado o desde la pista `correspondiente` si está pausado. Devolviendo en MessageToDisplay el estado, la información del CD y la pista que está sonando…
  PLAYING... Album: Thriller Artist: Michael Jackson Track 1 - Wanna Be Startin' Somethin
- **Stop**: Parará la reproducción. Devolviendo MessageToDisplay...
  STOPPED... Album: Thriller Artist: Michael Jackson
- **Pause**: Pausará la reproducción si está sonando y la reanudará si está pausada. Si pasa a pausado, MessageToDisplay devolverá…
  PAUSED... Album: Thriller Artist: Michael Jackson. Track 1 - Wanna Be Startin' Somethin
- **Next/Previous**: Si esta sonando, buscará la anterior o siguiente pista a reproducir de forma cíclica. Esto es, si llega al final irá al principio y viceversa. Además, si está pausado empezará a reproducir la nueva pista.

El sintonizador de DAB implementa el interfaz IMedia con la funcionalidad, empezará parada.

- **MessageToDisplay**: Propiedad que devuelve un mensaje para el Display del DABRadioCD con el estado de la radio.
- **Play**: Que sintonizará la primera frecuencia de la banda de FM (MIN_FREQUENCY) si estaba apagada (OFF) o continuará con el streaming almacenado en el buffer si estaba pausada. Devolviendo MessageToDisplay…
  HEARING... FM – 87,5 MHz
- **Stop**: Parará el streamig. Devolviendo MessageToDisplay… RADIO OFF
- **Pause**: Pausará la reproducción si está sonando la radio y la reanudará si está pausada. Si pasa a pausado se almacenará todo el streaming en un buffer para poder reanudar la emisión donde se quedó y MessageToDisplay devolverá…
  PAUSED - BUFFERING... FM – 87,5 MHz
- **Next/Previous**: Si esta sonando moverá el dial a la anterior o siguiente frecuencia, con saltos de 0,5 MHz cada vez que se pulse. Si llega al final de la banda (MAX_FREQUENCY) irá al principio de la misma y viceversa. Además, si está pausada empezará a reproducir desde la nueva frecuencia.

Nuestro DABRadioCD implementa el interfaz IMedia con la funcionalidad:
Para los métodos de IMedia, llamará a los respectivos del dispositivo activo en ese momento.

- **MessageToDisplay**: Devolverá una cadena con el dispositivo activo, el estado devuelto por el correspondiente método del dispositivo activo y el menú de opciones para manejar nuestro DABRadioCD.
  PAUSED - MODO: CD
  STATE: PLAYING... Album: Thriller Artist: Michael Jackson. Track 1 - Wanna Be Startin' Somethin

[1]Play [2]Pause [3]Stop [4]Prev [5]Next [6]Switch [7]Insert CD [8]Extract CD, [ESC]Turn off

- **Insertar un CD**: Devolverá una excepción si ya hay un CD dentro del reproductor. Si no lo hay, pasaremos a modo CD y empezará la reproducción automáticamente.
- **Extraer un CD**: Retirará el CD del reproductor y pasará a modo DAB.
- **Intercambiar modo**: Pasará de CD a DAB o viceversa. Teniendo en cuenta que si pasamos a CD este empezará a reproducir donde se quedó.

Otras funcionalidades u operaciones sobre los objetos puedes deducirlas del siguiente programa principal de ejemplo y de la propuesta de diagrama de clases UML del ejercicio.

```
  public static void Main()
  {
       string[] canciones = {
          "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"};
       Disc thriller = new Disc("Thriller", "Michael Jackson", canciones);
       DABRadioCD radioCD = new DABRadioCD();
       ConsoleKeyInfo tecla = new ConsoleKeyInf();
      do
      {
           try
           {
               Console.WriteLine(radioCD.MessageToDisplay);
               tecla = Console.ReadKey(true);
               Console.Clear();
               switch (tecla.KeyChar)
               {
                   ...
                   case '7':
                       radioCD.InsertCD = thriller;
                       break;
                   ...
               }
           }
           catch (Exception e)
           {
               Console.WriteLine(e.Message);
           }
       } while (tecla.Key != ConsoleKey.Escape);
   }
}
```

Ejercicio 5

Crea una clase parametrizada **Fraccion**, que tendrá dos propiedades públicas de solo lectura **Numerador** y **Denominador** y un constructor para inicializar los datos. Prueba el siguiente programa con la clase creada:

```
class Program
{
 public static void Main()
 {
   Fraccion<int> fraccionEnteros = new Fraccion<int>(4, 5);
   Console.Write($"La fracción es:{fraccionEnteros.Numerador}/{fraccionEnteros.Denominad
   Fraccion<float> fraccionDecimales = new Fraccion<float>(4.7f, 5.2f);
   Console.Write($"La fracción es:{fraccionDecimales.Numerador}/{fraccionDecimales.Denom
 }
}
```

Ejercicio 6

Partiendo de la siguiente definición de clase parametrizada…

```
class A<T, U>
{
  private T clave;
  private U valor;
  …
}
```

- Define un constructor que reciba los dos atributos como parámetro.
- Crea 2 propiedades que te permitirán devolver los dos atributos.
- Prueba la clase en la Main con una clave de tipo entero y un valor de tipo cadena.

Nota: Sin usar el código *'autogenerado'* por el IDE.

Ejercicio 7

Crea una clase estática genérica llamada **Comparador** que posea a su vez dos métodos de utilidad estáticos llamados Mayor y Menor. Ambos recibirán dos parámetros del tipo genérico, y devolverán true o false en el caso de que el primer parámetro sea mayor que el segundo y viceversa, **¿qué problemas has encontrado?**.
La mejor forma de solucionarlo, es obligando a que el parámetro genérico implemente la interface IComparable.
Crea una clase programa que te permita probar estos métodos, mandándo diferentes elementos enteros, strings, float, etc.

✅ Ejercicio 8

Partiendo del ejercicio anterior, crea una clase **Persona** que tenga solo dos propiedades: Nombre y Edad.
Comprueba si funcionan los métodos Mayor y Menor con ella, **¿qué ocurre?**. Ahora haz que la clase derive de IComparable y de ICloneable y que invalide el ToString().
Crea un programa que te permita saber, de dos objetos Persona distintos, cual es el mayor.
Clona una persona y prueba los clones con el método estático Menor.

✅ Ejercicio 9

Crea una clase genérica **Lista** que tenga un array parametrizado y privado. Un constructor para inicializar el array a 0, y un método **Add** al que le llegue un dato de tipo parametrizado, redimensione el array y lo añada al final de este.
Además crearemos un **indizador público** para poder acceder y modificar el elemento correspondiente del array. Prueba la clase creada con el siguiente programa:

```
class Program
{
        public static void Main()
        {
            Lista<int> lista=new Lista<int>();
            lista.Add(5);
            Console.WriteLine(lista[0]);
            lista.Add(8);
            Console.WriteLine(lista[1]);
            lista[1]=10;
            Console.WriteLine(lista[1]);
            //Prueba la lista con string
        }
}
```

Ejercicio 10

Crea una clase con un parámetro genérico **ParOrdenado** con dos atributos denominados primero y segundo del mismo tipo **T**. Crea el correspondiente constructor, propiedades y el método ToString.
Además, añade un **indizador** de solo lectura que dependiendo de si el índice es 0 te devolverá el valor de atributo primero y si es 1 el valor de atributo segundo (o una excepción en cualquier otro caso). Crea un programa que te permita probar esta clase usando enteros y cadenas.

✅ Ejercicio 11

A partir de la clase del ejercicio anterior, vamos a crear una clase **NumeroComplejo** que derivare de **ParOrdenado < double >**.
Como ya sabemos, los números complejos constan de dos partes una real y una imaginaria (compuesta por el número y el sufijo i), por eso vamos a utilizar la superclase ParOrdenado que ya posee los dos elementos que necesitamos.

- Tendremos un constructor al que le pasaremos un string con la forma binomial de un número complejo y se encargará de comprobarlo con una expresión regular.
- Además, también tendrás que redefinir los operadores +, - , *, ==, != y el cast explícito.
- Anula el método ToString que te devolverá el número de forma correcta con elsufijo y el signo + añadido.

Crea los elementos necesarios para que la clase quede completa y el programa para probar su funcionamiento.
💡 **Pista:** Un número complejo se representa en forma binomial como:

$$z = a + bi$$

Las operaciones que se piden en el programa son…

$$Suma => (a,b) + (c,d) = (a + c, b + d)$$

$$Multiplicación => (a,b) \cdot (c,d) = (a \cdot c - b \cdot d, a \cdot d - c \cdot b)$$

$$Resta => (a,b) - (c,d) = (a - c, b - d)$$

$$Igualdad => (a,b) = (c,d) \Leftrightarrow a = c \wedge b = d$$