



# **PROGRAMACIÓN ORIENTADA A OBJETOS**

## **INVESTIGACIÓN (INTERFACES EN C#)**

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES

**NOMBRE:**

Prado Maritza

**DOCENTE:**

Ing. Cinthia Anahí Mata Bravo

**TEMA 4:**

Herencia y polimorfismo

**SUBTEMA:**

- 4.1 Concepto de herencia y polimorfismo.
- 4.2 Definición de una clase base.
- 4.3 Definición de una clase derivada.
- 4.4 Clases abstractas.
- 4.5 Interfaces.
  - 4.5.1 Definición
  - 4.5.2 Implementación.
  - 4.5.3 Variable polimórficas
- 4.6 Reutilización de la definición de paquetes/librerías

**2º SEMESTRE**

**FECHA:**

19 de marzo de 2020



## Índice

<b>Interface C#</b> .....	3
Ejemplo 1/SampleMethod.....	4
Ejemplo 2/Implementación .....	5
Ejemplo 3/Implementación (primera parte).....	6
Ejemplo 4/Implementación (segunda parte).....	7
<b>Diferencias entre clase abstracta e interface</b> .....	7
Generalidades.....	7
<b>Clase abstracta</b> .....	7
<b>Interfaces</b> .....	7
<b>Conclusión</b> .....	8
<b>Bibliografía</b> .....	8



## Interface C#

Una interfaz contiene definiciones para un grupo de funcionalidades relacionadas que una clase o una estructura deben implementar. Una interfaz puede definir static métodos, que deben tener una implementación. Una interfaz puede proporcionar una implementación predeterminada para cualquiera o todos sus miembros de instancia declarados. Una interfaz no puede declarar datos de instancia como campos, propiedades implementadas automáticamente o eventos similares a propiedades.

Mediante el uso de interfaces, puede, por ejemplo, incluir el comportamiento de múltiples fuentes en una clase. Esa capacidad es importante en C # porque el lenguaje no admite la herencia múltiple de clases. Además, debe usar una interfaz si desea simular la herencia de estructuras, porque en realidad no pueden heredar de otra estructura o clase.

Una interfaz define un contrato. Cualquiera class o struct que implemente ese contrato debe proporcionar una implementación de los miembros definidos en la interfaz. A partir de C # 8.0, una interfaz puede definir una implementación predeterminada para los miembros. También puede definir static miembros para proporcionar una implementación única para una funcionalidad común.

En el siguiente ejemplo, la clase **ImplementationClass** debe implementar un método llamado SampleMethod que no tiene parámetros y retornos **void**.

## Ejemplo

```
C# Copiar

interface ISampleInterface
{
    void SampleMethod();
}

class ImplementationClass : ISampleInterface
{
    // Explicit interface member implementation:
    void ISampleInterface.SampleMethod()
    {
        // Method implementation.
    }

    static void Main()
    {
        // Declare an interface instance.
        ISampleInterface obj = new ImplementationClass();

        // Call the member.
        obj.SampleMethod();
    }
}
```

*Ejemplo 1/SampleMethod*

Una interfaz puede ser miembro de un espacio de nombres o una clase. Una declaración de interfaz puede contener declaraciones (firmas sin ninguna implementación) de los siguientes miembros:


- Métodos
- Propiedades
- Indexadores
- Eventos

Estas declaraciones de miembros anteriores generalmente no contienen un cuerpo. A partir de C # 8.0, un miembro de la interfaz puede declarar un cuerpo. Esto se llama *implementación predeterminada*. Los miembros con cuerpos permiten que la interfaz proporcione una implementación "predeterminada" para clases y estructuras que no proporcionan una implementación primordial. Además, comenzando con C # 8.0, una interfaz puede incluir:

- Constantes
- Operadores
- Constructor estático .
- Tipos anidados
- Campos estáticos, métodos, propiedades, indexadores y eventos.
- Declaraciones de miembros que usan la sintaxis de implementación de interfaz explícita.
- Modificadores de acceso explícito (el acceso predeterminado es public).

Las interfaces pueden no contener estado de instancia. Si bien los campos estáticos ahora están permitidos, los campos de instancia no están permitidos en las interfaces. Las propiedades automáticas de las instancias no son compatibles con las interfaces, ya que declararían implícitamente un campo oculto. Esta regla tiene un efecto sutil en las declaraciones de propiedad. En una declaración de interfaz, el siguiente código no declara una propiedad implementada automáticamente como lo hace en un `class` o `struct`. En cambio, declara una propiedad que no tiene una implementación predeterminada, pero debe implementarse en cualquier tipo que implemente la interfaz:

C#

 Copiar

```
public interface INamed
{
    public string Name {get; set;}
}
```

*Ejemplo 2/Implementación*

Una interfaz puede heredar de una o más interfaces base. Cuando una interfaz anula un método implementado en una interfaz base, debe usar la sintaxis explícita de implementación de la interfaz.

Cuando una lista de tipos base contiene una clase base e interfaces, la clase base debe ser la primera en la lista.

Una clase que implementa una interfaz puede implementar explícitamente miembros de esa interfaz. No se puede acceder a un miembro implementado explícitamente a través de una instancia de clase, sino solo a través de una instancia de la interfaz. Además, solo se puede acceder a los miembros predeterminados de la interfaz a través de una instancia de la interfaz.

Ejemplo

El siguiente ejemplo muestra la implementación de la interfaz. En este ejemplo, la interfaz contiene la declaración de propiedad y la clase contiene la implementación. Cualquier instancia de una clase que implemente `IPoint` tiene propiedades enteras `x` y `y`.



```
interface IPoint
{
    // Property signatures:
    int X
    {
        get;
        set;
    }

    int Y
    {
        get;
        set;
    }

    double Distance
    {
        get;
    }
}

class Point : IPoint
{
    // Constructor:
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    // Property implementation:
    public int X { get; set; }
}
```

*Ejemplo 3/Implementación (primera parte)*

```
// Property implementation
public double Distance =>
    Math.Sqrt(X * X + Y * Y);

}

class MainClass
{
    static void PrintPoint(IPoint p)
    {
        Console.WriteLine("x={0}, y={1}", p.X, p.Y);
    }

    static void Main()
    {
        IPoint p = new Point(2, 3);
        Console.Write("My Point: ");
        PrintPoint(p);
    }
}
// Output: My Point: x=2, y=3
```

*Ejemplo 4/Implementación (segunda parte)*

Cuando tenemos varias clases que tienen que implementar el mismo método, es aconsejable utilizar “interfaces”. Son como las clases (propiedades, métodos y eventos) pero no contienen código. La codificación se hará en la clase que implementa la interfaz.

## Diferencias entre clase abstracta e interface

### Generalidades

#### Clase abstracta

- Tiene jerarquías al igual de las interfaces
- Pueden tener estados (atributos) y comportamientos (métodos/funciones).
- No es posible crear un objeto a partir de una clase abstracta.

#### Interfaces

- Nos ayuda a crear jerarquías de clases.
- Una clase puede implementar muchas interfaces pero solo lo puede heredar de una clase.

Las interfaces nos ayudan a simular la herencia múltiple y son mucho más flexibles.



## Conclusión

De acuerdo a la información presentada anteriormente se muestra como se implementan las interfaces, que estas se pueden implementar de manera implícita o explícita ya depende de lo que sea requerido, así como la diferencia entre las clases abstractas que estas a su vez son clases en las cuales se declaran atributos y métodos los cuales no se harán uso en dicha clase, si no que esos serán heredados a sus clases hijas, pero por lo cual todos los métodos implementados en la clase principal deben ser utilizadas en sus clases hijas, pero una cosa curiosa es que en ellos no se escribe el algoritmo para resolver o darle una función, si no que estas ya serán adaptadas a las clases hijas de manera específica, es así como interface cumple casi con el mismo objetivo, ya que en la actualidad no existe mayor diferencia, pero estas son más flexibles, ya que en ellas se declaran los métodos para que después sean aplicadas en las clases y ya no las heredaran de una clase abstracta si no de un interface en el cual ya se le dará función en la clase.

El objetivo de recopilar la información ya antes expuesta de distintas fuentes es para que sea más comprensible a la hora de llevar a la práctica y sea un poco más comprensible con la ayuda de códigos y una pequeña guía.

## Bibliografía

Theme Grill, T. G. (s.f.). Interfaces – Tutorial de C#. Recuperado 20 marzo, 2020, de <https://csharp.com.es/interfaces/>

Bill Wagner, B. G. (2020, 17 enero). Interface - C# Reference. Recuperado 20 marzo, 2020, de <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface>

Bill Wagner, B. W. (2020, 20 febrero). Interfaces - C# Programming Guide. Recuperado 20 marzo, 2020, de <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/>