INTERFACES EN C#

PROGRAMACIÓN ORIENTADA A OBJETOS – HERENCIA Y POLIMORFISMO – TEMA 4

INSTITUTO TECNOLÓGICO DEL SUR DE NAYARIT
MAESTRA: CINTHIA ANAHÍ MATA BRAVO

EDUARDO MARMOLEJO ORNELAS - 191140006

INDICE

INTERFACES..... 2

RESUMEN PROPIO..... 3

INTERFACES

Describen un grupo de funciones relacionadas que pueden pertenecer a cualquier clase o estructura.

Sirve para resolver problemas de las herencias múltiples de la cuál no se puede hacer en C#, Java, etc.

Es parecido a una clase abstracta, solamente que esta no tiene implementado los métodos y que solamente los tiene definidos.

Una interfaz define un contrato. Cualquier clase o estructura que implemente ese contrato debe proporcionar una implementación de los miembros definidos en la interfaz. Una interfaz puede definir una implementación predeterminada para los miembros. También puede definir miembros estáticos para proporcionar una única implementación para la funcionalidad común.

Un miembro de interfaz puede declarar un cuerpo. Esto se denomina implementación predeterminada. Los miembros con cuerpos permiten que la interfaz proporcione una implementación "predeterminada" para clases y estructuras que no proporcionan una implementación de reemplazo. Además, a partir de la versión 8.0, una interfaz puede incluir:

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

Es posible que las interfaces no contengan el estado de la instancia. Mientras que los campos estáticos ahora están permitidos, los campos de instancia no se permiten en las interfaces. Las propiedades automáticas de instancia no se admiten en las interfaces, ya que declararían implícitamente un campo oculto. Esta regla tiene un efecto sutil en las declaraciones de propiedad.

Las interfaces sirven mucho para cuando estás programando junto con otros programadores, para decirles los nombres que uno quiere que se les llame, y que eso bueno ya que se le da calidad al código para las cosas que son parecidas en funcionalidad se llamen igual.

Resumen personal

Lo que entendí del tema, que la interfaz es una opción que sirve para solucionar las herencias múltiples, también que las interfaces se describen como grupo de funciones que puede pertenecer a cualquier clase.

Es parecido a una clase abstracta, pero en la interfaz solo tiene los métodos definidos.

También se me hizo interesante que lo asimilen a un contrato, y es una forma fácil de entender ya que es como un contrato, que en el cual las clases que se implementen deben de cumplir con las reglas que este tiene este contrato que en este caso son los métodos y atributos.

También que las interfaces son muy útiles al momento de programar con otros programadores, para así obligarlos a poner nombres u otras cosas como uno quiere (a lo que le entendí a un video), que eso es bueno para así darle calidad al código para que las cosas que sean de funcionalidad se llamen igual.

Dentro de estas interfaces pueden contener lo que son operadores, constructores estáticos, condiciones o tipos anidados (if, switch, entre otros.), propiedades, entre otras cosas más.

En pocas palabras, a lo que entendí que es parecido a una clase padre, ya que la interface les dice a las clases, que los nombres u otras cosas, ya sean atributos, métodos, entre otras cosas se llamen de una manera y que solamente muestra el comportamiento y no de lo que se está llevando a cabo.

También que pueden tener varias cosas dentro de ellos, operadores, constructores, que solamente están definidos es similar a una clase abstracta pero no son iguales.

También una clase base también puede implementar miembros de interfaz mediante el uso de clases abstractas, es decir clases abstractas con su contenido abstracto, por ejemplo:

```
abstrac ClassName1:InterfaceName

{

Public abstract void NombreDelMetodo();

}
```

En ese caso, una clase derivada puede cambiar el comportamiento de la interfaz reemplazando los miembros virtuales, implementando un override en el nuevo método de la nueva clase como se muestra a continuación.

```
ClassName2: ClassName1

{

Public override void NombreDelMetodo()

{

  Y en ese sitio colando las operaciones que va a realizar el método

}

}
```