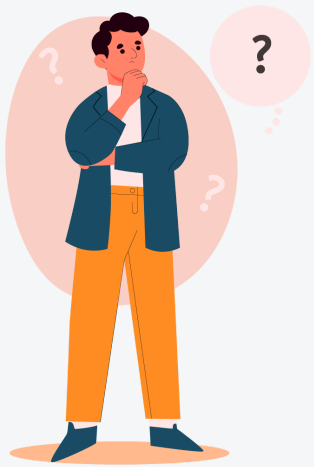
Interfaces

Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1° D
Libro: Interfaces

Imprimido por: Eduardo Moreno
Día: martes, 27 de mayo de 2025, 01:37

Tabla de contenidos

Preguntas orientadoras



- ¿Qué es una interfaz?
- ¿Cómo se crea una interfaz?
- ¿Cómo se implementa una interfaz?
- ¿Cómo se obliga a que diferentes clases que no comparten la misma jerarquía implementen un método específico?
- ¿Cómo se hace para definir un "contrato" de métodos que tiene que sí o sí implementar una clase?

Introducción

En este módulo aprenderemos a utilizar **interfaces** (interfaz, en singular).

Una interfaz de una clase es todo lo que se puede hacer con ella. Hay veces que diferentes jerarquías de clases necesitan tener una misma interfaz. Por ejemplo: un automóvil y una radio pueden tener métodos compartidos como "encender" y "apagar", y no necesariamente son de la misma jerarquía de clases.

¿Qué establece? Una interfaz establece una declaración de un contrato que debe ser implementado por una clase o estructura. Define un conjunto de métodos que una clase o estructura debe tener, pero no proporciona ninguna implementación de esos métodos. La implementación de los métodos es responsabilidad de la clase o estructura que implementa la interfaz.

En suma, las interfaces se utilizan para añadir funcionalidad a las clases o estructuras sin tener que modificar el código de las clases o estructuras existentes.

Comencemos con el recorrido.

¿Qué es una interfaz en POO?

En Programación de Orientación a Objetos (POO) se denomina interfaz de una clase a todo lo que se puede hacer con la clase. A efectos prácticos: todos los métodos y propiedades que no son privadas de la clase conforman su interfaz.

¡Veamos un ejemplo concreto!

Dada la siguiente clase llamada Caja que almacena valores enteros y sus métodos son:

- agregar (agrega un valor entero a la caja)
- quitar (quita un valor entero de la caja)
- estaRepetido (nos devuelve verdadero o falso si un determinado valor recibido por parámetro se encuentra en la caja)

```
class Caja
{
    public void Agregar(int v){...};
    public int Quitar(){...};
    private bool EstaRepetido(int v) {...};
}
```

Su interfaz está formada por los métodos agregar y quitar.

El método EstaRepetido no forma parte de la interfaz de dicha clase, ya que es privado.

En POO se dice que la interfaz de una clase define el comportamiento de dicha clase, señalando qué se puede hacer con sus objetos; dado un objeto de la clase Caja se puede llamar al método agregar() y al método quitar() pero no se puede llamar al método estaRepetido().

Por esta razón, resulta importante señalar que interfaces idénticas no significa que sean clases intercambiables.

Trabajemos con otro ejemplo

En las dos clases siguientes se tiene la misma interfaz, pero son dos clases diferentes:

```
class Caja
{
    public void agregar (int v) { ... }
    public int quitar() { ... }
}
class Bolsa
{
    public void agregar (int v) { ... }
    public int quitar() { ... }
}
```

Imaginemos que en otra parte del programa tenemos un método como el siguiente:

```
public void embalar(Caja c)
{
    // Hacer cosas con c ...
    int i = c.quitar();
    c.agregar(10);
}
```

El método recibe una Caja y opera con ella.

Ahora dado que las interfaces de Caja y Bolsa son iguales, sería deseable que lo siguiente funcione usando polimorfismo:

```
...
Bolsa b = new Bolsa();
embalar(b);
...
```

Pero esto no va a compilar ya que para el compilador Caja y Bolsa son dos clases totalmente distintas sin ninguna relación entre ellas. Por lo tanto, un método que espera una Caja no puede aceptar un objeto de la clase Bolsa.

Profundicemos sobre la implementación

Implementación de Interfaces

En el ejemplo anterior (class Caja; class Bolsa), observamos algo que es muy común que suceda: tener dos clases que hacen lo mismo pero de diferente manera.

¿Y por qué sucede? Estas clases están relacionadas por una parte de su interfaz pública, no necesariamente toda; y no tienen una superclase o clase padre en común.

	Desarrollemos esta idea

Imaginemos que estamos desarrollando un videojuego en el que tendremos personajes, objetos fijos y objetos móviles. Todos estos elementos tendrán una funcionalidad para "aparecer" y "desaparecer" en la pantalla del videojuego. Esta funcionalidad es la misma para todos, lo que varía es la implementación.

En POO, se dice que las interfaces son funcionalidades (el qué hace) o también se los conoce como "contratos públicos" y las clases representan o realizan las implementaciones (el cómo lo hace).

Para que se pueda intercambiar las distintas clases que tienen la misma funcionalidad con implementaciones diferentes, los lenguajes de POO permiten explicitar la interfaz, es decir, separar la declaración de la interfaz de su implementación (de su clase). Para ello se usa la palabra clave interface:

interface IComportamientoEnPantalla

```
{
    void aparecer();
    void desaparecer();
}
```

Este código declara una interfaz IComportamientoEnPantalla que declara los métodos aparecer y desaparecer. Es importante observar que:

- Los métodos no se declaran como public (en una interfaz la visibilidad no tiene sentido, ya que todo es public).
- Los métodos no se implementan (no se coloca código).
- No se pueden instanciar interfaces (no se puede hacer new IComportamientoEnPantalla()).

Continuemos con esta idea ...

Continuando con el ejemplo

Volviendo a nuestro desarrollo del videojuego, debemos ahora definir las clases a las que les podemos indicar explícitamente que implementen una interfaz, es decir, proporcionen implementación (código) a todos y cada uno de los métodos declarados en la interfaz:

```
class Personaje : IComportamientoEnPantalla
{
          public void aparecer() { ...<codigo>... }
    public void desaparecer() { ...<codigo>... }

        }
```

```
class objetoFijo : IComportamientoEnPantalla
{
    public void aparecer() { ...<codigo>... }
    public void desaparecer() { ...<codigo>... }
}
class objetoMovil : IComportamientoEnPantalla
{
    public void aparecer() { ...<codigo>... }
    public void desaparecer() { ...<codigo>... }
}
```

La clase Personaje declara explícitamente que implementa la interfaz IContenedor. Por lo tanto, la clase debe proporcionar implementación para todos los métodos de la interfaz.

El siguiente código no compilaría:

```
class Personaje : IComportamientoEnPantalla
{
    public void aparecer() { ...<codigo>... }
}
//  Error:  Falta  el  método  desaparecer()!!!
```

Por esta razón, en POO, **se dice que las interfaces son contratos**, porque cuando creamos la clase la interfaz nos obliga a implementar ciertos métodos y si usamos la clase, la interfaz nos dice qué métodos podemos llamar.
La ventaja principal de trabajar con interfaces es que si dos clases implementan la misma interfaz, entonces son intercambiables. Por lo tanto, en cualquier lugar donde se espere una instancia de la interfaz, puede pasarse una instancia de cualquier clase que implemente dicha interfaz.

Podríamos declarar nuestro método actualizarPantalla como sigue:

```
void actualizarPantalla(IComportamientoEnPantalla unObjeto)
// observar que el parámetro unObjeto es del tipo IComportamientoEnPantalla
{
// unObjeto puede ser un personaje, un objetoFijo o un objetoMovil
    unObjeto.aparecer();
    unObjeto.desaparecer();
}
```

Al indicar que el parámetro es de tipo IComportamientoEnPantalla (en vez de poner Personaje, ObjetoFijo, etc) estamos indicando que el método trabaja con cualquier clase que implemente IComportamientoEnPantalla.

A continuación, veamos cuándo es necesario utilizar interfaz.

¿Cuándo usar interfaces?

En general, siempre que tengamos más de una clase para hacer lo mismo deberíamos usar interfaces. En este caso, para nuestro desarrollo, es mejor pecar de exceso que de defecto ya que no hay penalizaciones de rendimiento por hacer esto.

No necesariamente toda clase debe implementar una interfaz obligatoriamente, ya que muchas clases internas no lo implementaran, pero en el caso de las clases públicas (visibles desde el exterior) debería pensarse bien.

Además, pensar en la interfaz antes que en la clase en sí, es pensar en lo que debe hacerse en lugar de pensar en cómo debe hacerse.

Algo para resaltar: Usar interfaces permite a posteriori cambiar una clase por otra que implemente la misma interfaz y poder integrar la nueva clase de forma mucho más fácil (solo debemos modificar donde instanciamos los objetos pero el resto del código queda igual).

Dividiendo las interfaces

Imaginemos que tenemos otro videojuego que debe trabajar con varios vehículos, entre ellos aviones y autos, así que declaramos la siguiente interfaz:

```
interface IVehiculo
{
    void acelerar(int kmh);
    void frenar();
    void girar(int angulos);
    void despegar();
    void aterrizar();
}
```

Luego implementamos la clase avión:

```
class Avion : IVehiculo
{
    public void acelerar(int kmh) { ... }
    public void frenar() { ... }
    public void girar (int angulos) { ... }
    public void despegar() { ... }
    public void aterrizar() { ... }
}
```

Y luego la clase auto, pero.....:

```
class Auto : IVehiculo
{
    public void acelerar(int kmh) { ... }
    public void frenar() { ... }
    public void girar (int angulos) { ... }
    public void despegar() { ??? los autos no vuelan }
    public void aterrizar()   { ??? los autos no vuelan }
}
```

La interfaz IVehiculo tiene demasiados métodos y no define el comportamiento de todos los vehículos, dado que no todos los vehículos despegan y aterrizan. En este caso es mejor dividir la interfaz en dos:

```
interface IVehiculo
{
    void acelerar(int kmh);
    void frenar();
    void girar (int angulos);
}
```

interface IVehiculoVolador : IVehiculo // observar que deriva de IVehiculo

{

void despegar();

void aterrizar();

```
}
```

Al definir la relación de herencia entre IVehiculoVolador y IVehiculo significa que una clase que implemente IVehiculoVolador debe implementar también IVehiculo forzosamente. Por lo tanto, podemos afirmar que todos los vehículos voladores son también vehículos.

Ahora sí que la clase Auto puede implementar IVehiculo y la clase Avion puede implementar IVehiculoVolador (y por lo tanto también IVehiculo). Si un método desplazar() recibe un objeto IVehiculoVolador puede usar métodos tanto de IVehiculoVolador como de IVehiculo:

```
void desplazar  (IVehiculoVolador  vv)
{
    vv.Despegar();      // Ok. Despegar es de IVehiculoVolador
    vv.Acelerar(10);    // Ok. Acelerar es de IVehiculo y IVehiculoVolador deriva de
IVehiculo
}
```

¡Al revés no! Si un método desplazar recibe un IVehiculo no puede llamar a métodos de IVehiculoVolador. Todos los vehículos voladores son vehículos pero al revés no: ¡no todos los vehículos son vehículos voladores!

Siempre que haya desglose no tiene por qué haber herencia de interfaces. Imaginemos el caso que además de vehículos debemos trabajar con Elevadores. Tenemos otra interfaz:

```
interface IElevador
{
    void subir();
    void bajar();
}
```

Ahora, podrían existir clases que implementen IElevador cómo Ascensor:

class Ascensor : IElevador

```
{
    public void subir() { ... }
    public void bajar() { ... }
}
```

Pero, también, tenemos vehículos que pueden ser a la vez elevadores, por ejemplo, una grúa.

Esto no es ningún problema ya que una clase puede implementar más de una interfaz a la vez. Para ello debe implementar todos los métodos de todas la interfaces:

```
class Grua : IVehiculo, IElevador
{
    public void acelerar(int kmh) { ... }
    public void frenar() { ... }
    public void girar (int angulos) { ... }
    public void subir() { ... }
    public void bajar() { ... }
}
```

Ahora, si un método recibe un IVehiculo le puedo pasar una Grua y si otro método recibe un IElevador también le puedo pasar una Grua. O dicho de otro modo, las grúas se comportan tanto como vehículos como elevadores a la vez.

Recomendaciones

Es importante desglosar bien nuestras interfaces porque en caso contrario vamos a tener dificultades a la hora de implementarlas.

Algunos Tips:

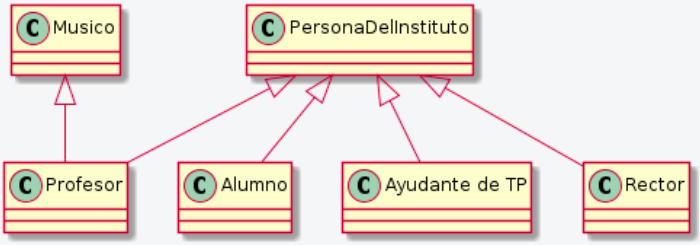
Para implementar un miembro de interfaz, el miembro correspondiente de la clase de implementación debe ser público, no estático y tener el mismo nombre y firma que el miembro de interfaz (la firma es la cantidad y tipo de parámetros y el tipo de dato que devuelve).

Una clase que implementa una interfaz debe proporcionar obligatoriamente una implementación para todos los métodos declarados en la interfaz.

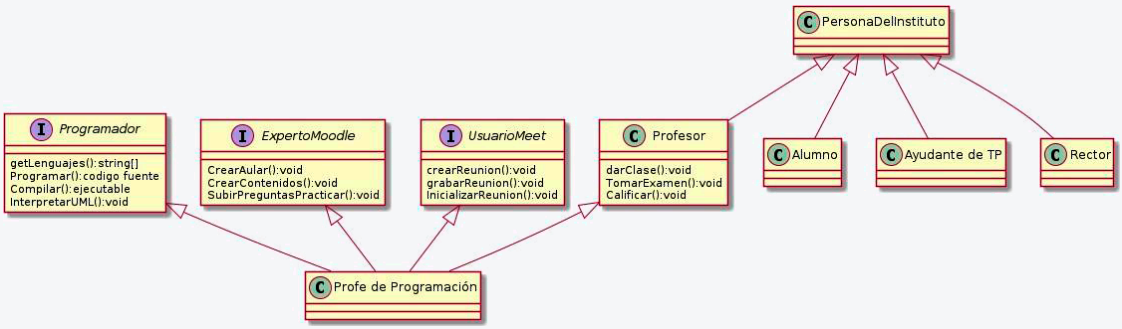
Si una clase base implementa una interfaz, cualquier clase que se derive de la clase base hereda esta implementación.

Herencia múltiple de interfaces

En semanas anteriores, al trabajar herencia, se remarcó que C# no soporta herencia múltiple (una clase no puede heredar de varias clases a la vez), ya que en general este tipo de herencia suele generar errores y vuelve más complejo el código.



En cambio sí se puede usar interfaces múltiples en una misma clase. Esto le indica al compilador que esta clase implementa todos esos métodos que se declaran en cada interfaz.



En este caso, la clase Profe de Programacion hereda de Profesor quien además hereda de PersonaDelInstituto.

Por otra parte, el Profe de Programacion implementa varias "interfaces" como la de Programador, ExpertoMoodle, UsuarioMeet.

Es decir, a un objeto del tipo ProfeDeProgramacion se le puede pedir todos los métodos de todas las clases que hereda y además todos los métodos de las interfaces que declara.

```
class ProfeDeProgramacion : Profesor, Programador, ExpertoMoodle, UsuarioMeet
{
    ...
// Programador
    public void programar(){..}
    public void compilar() { ...}
    public void interpretarUML() {...}
   ...
// UsuarioMeet
    public  crearAula(){..}
    public grabarReunion() {...}
    ...
//ExpertoMoodle
    public crearAula()  {...}
    public crearContendio() {...}
   ...
}
void Main() {
    var profe = new ProfeDeProgramacion();
    ...
    profe.crearAula();
    profe.crearContenido();
    profe.SubirPreguntasPracticar();
    ...
    profe.crearReunion();
    ...
    profe darClase();
    ...
    profe.grabarReunion;
    ...
    profe.interpretarUML();
    ...
    profe.programar();
    profe.compilar();
    ...
    profe.tomarExamen();
    profe.calificar();
    ...
}
```

En resumen

En esta semana aprendimos **Interfaces y Herencia de Interfaces en POO**. Aquí están los puntos claves para recordar:

- Una interfaz es un tipo que define un contrato para que lo implementen las clases.
- Una clase que implementa una interfaz debe proporcionar una implementación para todos los miembros de la interfaz.
- La herencia de interfaces es una forma de que una clase herede de múltiples interfaces.
- Las interfaces se pueden utilizar para hacer que el código sea más flexible y reutilizable.