Interfaces

Video 48

Interfaz es un conjunto de directrices que deben cumplir las clases.

Una interfaz es parecida a una clase pero no es lo mismo. Heredan a las clases.

Tienen una declaración de un método que va a estar sin completar. Es una especie de plantilla para que lo usen las clases que heredan.

¿Si se quiere obligar a los nuevos desarrolladores que especifiquen un método para decir cuántas patas tienen los nuevos mamíferos que vayan agregando al fichero ProyectoHerencia?

Con una interfaz (es una opción). Puede crearse en cualquier lugar del fichero, incluso en otro fichero. La primer letra, por convención es una i mayúscula.

interface IMamíferosTerrestres

{

int NumeroPatas(); //los métodos de una interfaz no se codean, no se desarrollan, solo se declaran. Tampoco llevan modificador de acceso.

}

La clase ballena no debería implementar esa interfaz, ya que no tiene patas.

class Caballo : Mamiferos, IMamíferosTerrestres //las interfaces van despues de la clase. Al ppio esto marca error, hasta que se desarrolle el método que trae la interfaz.

{

public Caballo(string nombreCaballo) : base(nombreCaballo)

{

}

public void Galopar()

{

Console.WriteLine("Soy capaz de galopar");

}

public int NumeroPatas() //El nombre de este método debe coincidir exactamente con el de la interfaz, si no, seguirá dando error.

{

return 4;

}

Console.WriteLine("Número de patas de Babieca: " + Babieca.NumeroPatas());

Video 49

¿Para qué sirve una interfaz? El método patas podría crearse directamente en la clase, sin necesidad de crearse la interfaz.

Si uno crea el método en la clase, luego no es obligatorio crearlo o no, en otras clases.

Si uno crea el método en la interfaz, luego es obligatorio crear el método numeroPatas.

Dice que es obligatorio, pero después uno elige si tiene la clase que heredar la interfaz o no.

Una clase puede trabajar con varias interfaces. Pero de una sola clase.

Si la interfaz tiene mas de un método y vos hacés que una clase la herede, tenés que desarrollar todas en la clase, si no te seguirá marcando error. En ese sentido, si sería obligatorio completar todo, una vez que decidís heredar la interfaz.

interface IAnimalesYDeportes

{

string TipoDeDeporte();

Boolean EsOlimpico();

}

En la clase caballo:

public string TipoDeDeporte()

{

return ("Hípica");

}

public Boolean EsOlimpico()

{

return true;

}

Métodos iguales, con iguales parámetros, en interfaces diferentes ¿Cómo se resuelve esto?

Cuando se codea se le pasa el mismo valor a ambos métodos.

Yo puse 2 int NumeroPatas en métodos diferentes.

Se prescinde del modificador de acceso:Se le saca el public y…

Se le antepone el nombre de la interfaz+punto:

int IMamíferosTerrestres.NumeroPatas()

{

return 4;

}

int ISaltoConPatas.NumeroPatas()

{

return 2;

}

Ahora ahí da error:

Console.WriteLine("Número de patas de Babieca: " + Babieca.NumeroPatas());

Esto es difícil de resolver:

En el main codea:

static void Main(string[] args)

{

Caballo Babieca = new Caballo("Babieca");

IMamíferosTerrestres IBabieca = Babieca; //Luego se usa el objeto creado para llamar en el console.e

ISaltoConPatas IBabieca2 = Babieca;

Humano Juan = new Humano("Juan");

Gorila Copito = new Gorila("Copito");

Mamiferos[] almacenAnimales = new Mamiferos[3];

almacenAnimales[0] = Babieca;

almacenAnimales[1] = Juan;

almacenAnimales[2] = Copito;

for (int i = 0; i < 3; i++)

{

almacenAnimales[i].Religion();

}

Ballena myWally = new Ballena("Wally");

myWally.nadar();

Console.WriteLine("Número de patas de Babieca: " + IBabieca.NumeroPatas()); //Acá se llama al objeto recién creado

Console.WriteLine("Número de patas con las que Babieca salta: " + IBabieca2.NumeroPatas());





Ejercicio AvisosTrafico bastante piola. Repasa muchas cosas.

Diferenciar clase abstracta a interfaz

<https://www.youtube.com/watch?v=iRELmavR_eY>

La clase abstracta tiene tanto la declaración como la implementación. La interfaz solo las declara.

No hay herencia múltiple, pero si se pueden implementar varias interfaces.

Las clases abstractas pueden tener un método constructor, las interfaces no.

Una clase abstracta puede contener miembros estáticos, las interfaces no.

Clase abstracta puede tener modificadores de acceso, las interfaces siempre son públicas.

La clase abstracta tiene mejor rendimiento, es mas rápida.

Una interfaz solo puede contener métodos, propiedades, indexadores y eventos. Una clase abstracta, también campos y constantes.

Una clase abstracta puede ser implementada total o parcialmente. Con override se puede modificar e implementar. A la clase solo se le pone abstract, no virtual.