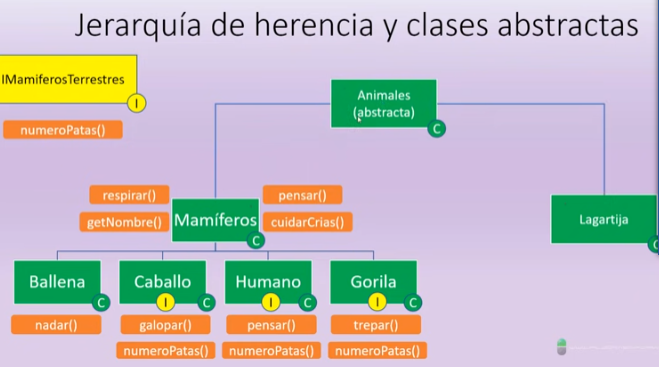
Clase abstracta

Video 52 y 53

“Si en una aplicación tenés código repetido, es que algo se hizo mal”



De esa forma se podría integrar la clase lagartija, tomando lo que le es común con los mamíferos. Codeando los métodos comunes, ya no en mamíferos sino en animales.

Mamíferos heredaría respirar y getNombre, que antes eran parte de si mismo.

Método abstracto es similar a lo que se hace con las interfaces. Solo se le declara, no se desarrolla(Eso no es así, se puede solo declarar, o también desarrollar). Eso para hacer que varíe en sus clases herederas.

Cuando se hereda un método abstracto, se está obligado a desarrollarlo.

Cuando en una clase se declara un método abstracto (al menos)G, la clase debe ser abstracta.

Se le dice abstracta a las que está mas arriba en la jerarquía. Las que menos cosas pueden hacer, ya que no heredan de nadie. Las que heredan de muchas, tienen sus métodos.

En cualquier parte se puede crear dicha clase:

abstract class Animales

{

public void respirar()

{

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

}

public abstract void getNombre();

}

Ahí se ve que desarrolla el método respirar. Hereda también getNombre.

class Mamiferos:Animales //Al poner :animales da error, ya que uno está obligado a desarrollar getNombre(); ya que solo está declarado.

public override void getNombre()

{

Console.WriteLine("El nombre del mamífero es: " + nombreSerVivo);

} //ESTO ES LO QUE SE ESTA OBLIGADO A HACER

public void respirar() //Aquí también aparece una advertencia. Por lo que hay que eliminarlo, ya que es exactamente igual.

{

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

}

Luego, al poner:

class Lagartija : Animales

Da error, y se abre una ventana mostrando soluciones posibles. Seleccionas eso, y aparece esto: //EL ERROR ES POR LO MISMO, POR QUE HAY QUE DESARROLLAR GETNOMBRE

class Lagartija : Animales

{

public override void getNombre()

{

throw new NotImplementedException();

}

}

Así quedaría la clase lagartija:

class Lagartija : Animales

{

public Lagartija (String nombre)

{

nombreReptil = nombre;

}

public override void getNombre()

{

Console.WriteLine("El nombre del reptil es: "+ nombreReptil);

}

private string nombreReptil; //para hacer esto crea el constructor de la clase.

}

Se podría poner la clase getNombre en mamíferos y en lagartija. Aunque ahí se estaría repitiendo código. Además, dice que si creas la clase abstracta, obligás a que se haga algo con la clase abstracta.

Clase selladas (Sealed)

Deja en claro que muchas de las cosas últimas, cuando se trata de obligar a tal o cual cosa, se piensa como jefe de proyecto, con un diseño ya bien pensado.

Si se tienen en claro que una clase no herede nada a otras, entra en juego la sealed class.

class Chimpance:Gorila

{

public Chimpance (string nombreChimpance) : base("nombreChimpance") { }

} //Si se hubiera querido que gorila no herede a chimpance, había que:

sealed class Gorila:Mamiferos, IMamíferosTerrestres

Tras poner eso, en la declaración de la clase chimpancé aparecerá un error.

También hay métodos sealed.

class Adolescente : Humano

{

public Adolescente(string nombreAdolescente):base( nombreAdolescente) { }

public override void Religion()

{

Console.WriteLine("Dios ha muerto, nosotres le matames");

}

}

Acá creo la clase adolescente y sobreescribo el método heredado de humano Religion.

public sealed override void Religion()

Cuando uno pone eso en el método humano, luego dará error en el método adolescente.