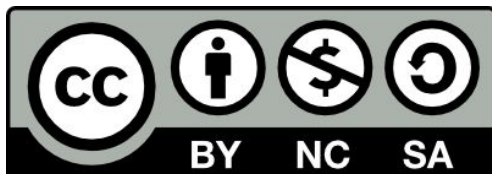
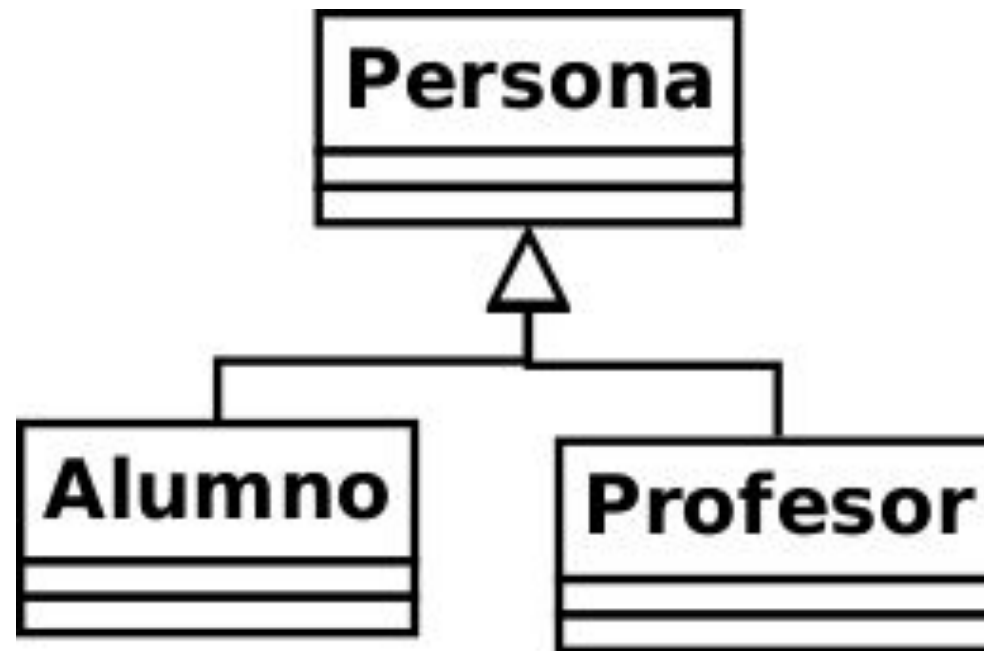


HERENCIA

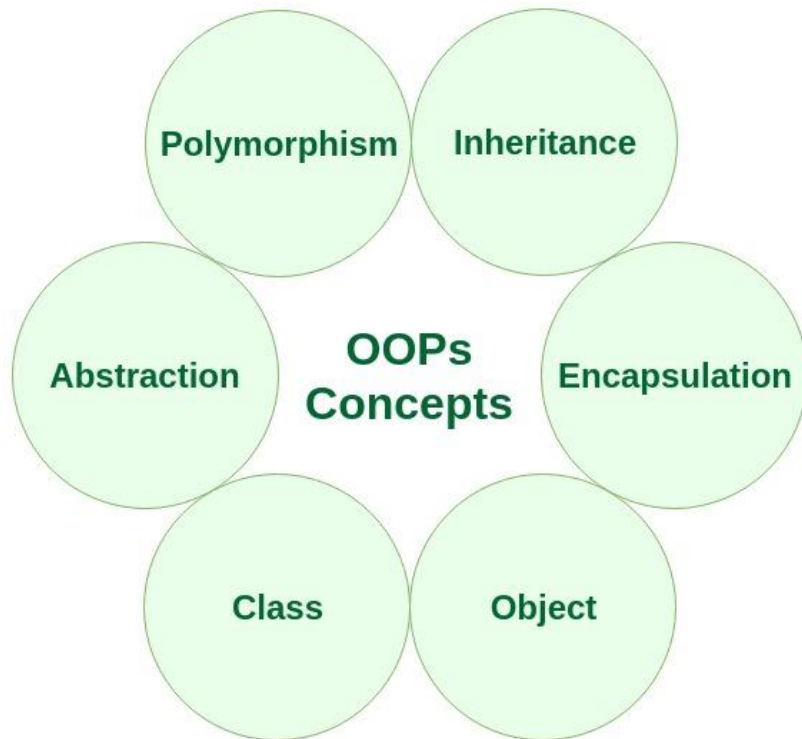


Aquesta obra està sota una llicència de Creative Commons Reconeixement-No comercial-Compartir igual 3.0 Espanya.

Institut Montilivi
1r DAM-DAW

Principis de la POO

- La OO inclou un conjunt de conceptes. Alguns dels quals són els següents:

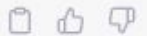


AJ

Principio de Liskov



El principio de Liskov es un principio fundamental de la programación orientada a objetos (POO), nombrado así en honor a Barbara Liskov, una informática pionera en la programación orientada a objetos y en la teoría de la computación.



El principio de Liskov establece que, en un programa orientado a objetos, los objetos de una clase derivada deben ser perfectamente sustituibles por objetos de su clase base, sin que esto afecte a la corrección del programa. En otras palabras, un objeto de una clase derivada debe ser capaz de responder a los mismos mensajes (métodos) que su clase base, y debe mantener el mismo comportamiento que la clase base cuando se utilice en el mismo contexto que la clase base.

Interfícies

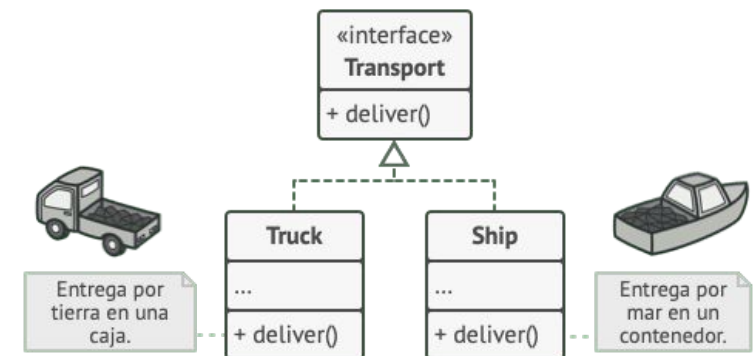
INTERFICIES → NO ÉS HERÈNCIA

```
public class Truck : IDeliverable
{
    private string name;
    private bool delivered;

    1 referencia
    public Truck(string name, bool delivered)
    {
        this.name = name;
        this.delivered = delivered;
    }

    3 referencias
    public bool Delivered {
        get
        { return delivered; }
        set { delivered = value; }
    }

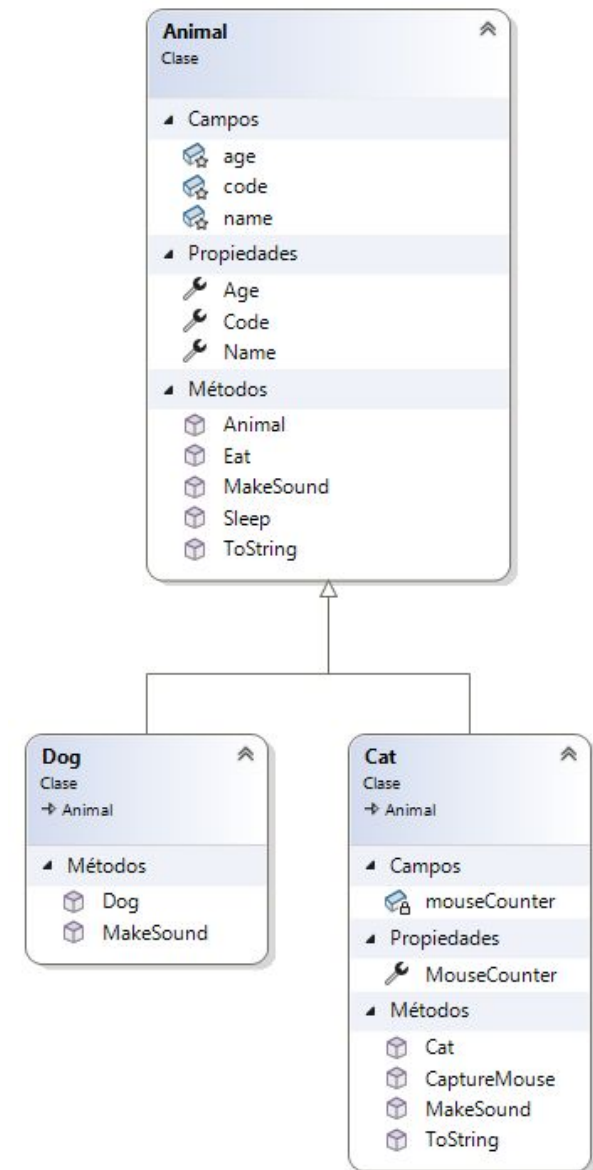
    2 referencias
    public void Deliver()
    {
        Console.WriteLine("-----INFORME DE LLIURAMENT DE MERCADERIES-----");
        Console.WriteLine("LLIURAMENT DE MERCADERIA PER TERRA CAMIÓ "+name);
        if (Delivered) Console.WriteLine("                MERCADERIA LLIURADA");
        else Console.WriteLine("                | MERCADERIA PENDENT DE LLIURAR");
        Console.WriteLine("-----");
    }
}
```



```
2 referencias
public interface IDeliverable
{
    4 referencias
    public void Deliver();
    4 referencias
    public bool Delivered
    {
        get;set;
    }
}
```

Herencia

- La classe Dog i la classe Cat hereten de la classe Animal.
- Animal és la superclasse. Dog i Cat són les subclasses. Les classes Dog i Cat són classes derivades de la classe Animal.
- Podem dir que:
 - Un Dog també és un Animal.
 - Un Cat també és un Animal.
 - Un Dog no és un Cat
 - Dog i Cat són Animals
 - Un Animal no té perquè ser un Dog, pot ser un Cat.



Classe Animal

```
public class Animal
{
    protected string code;
    protected string name;
    protected int age;

    1 referencia
    public string Code { get { return code; } }
    1 referencia
    public string Name { get { return name; } set { name = value; } }
    1 referencia
    public int Age { get { return age; } set { age = value; } }

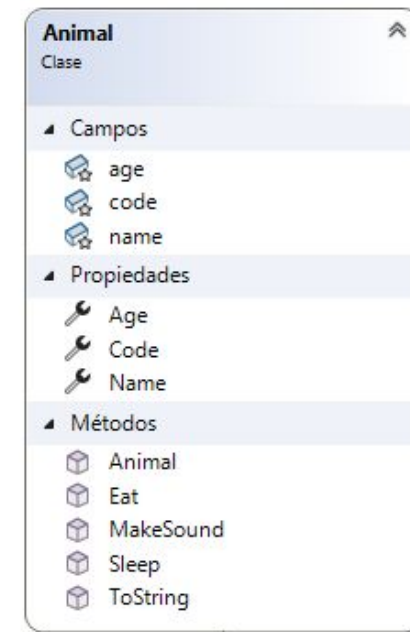
    1 referencia
    public Animal(string code, string name, int age, string specie)
    { this.code = code; this.name = name; this.age = age; }

    0 referencias
    public void Eat()
    { Console.WriteLine("Estic menjant."); }

    0 referencias
    public void Sleep()
    { Console.WriteLine("Estic dormint."); }

    3 referencias
    public virtual void MakeSound()
    { Console.WriteLine("I LOVE MY OWNER"); }

    2 referencias
    public override string ToString()
    {
        return $"{Code};{Name};{Age}";
    }
}
```



Analitza el codi que està subratllat.
Busca informació sobre cada paraula o símbol i explica per a què serveix.

Accessibilitat

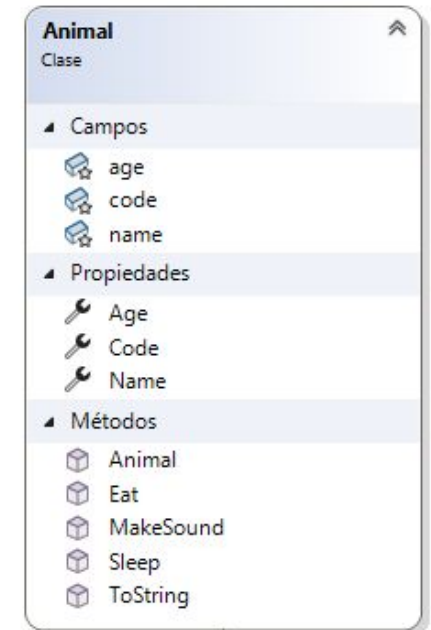
protected string code;

Atribut, propietat o mètode que pot ser heretat i accessible per les subclasses.

public string Code { get { return code; } }

Atribut, propietat o mètode que pot ser heretat i accessible des de qualsevol part del codi.

Normalment no utilitzarem **private**, doncs els atributs, propietats i mètodes private no són heretats per les subclasses.

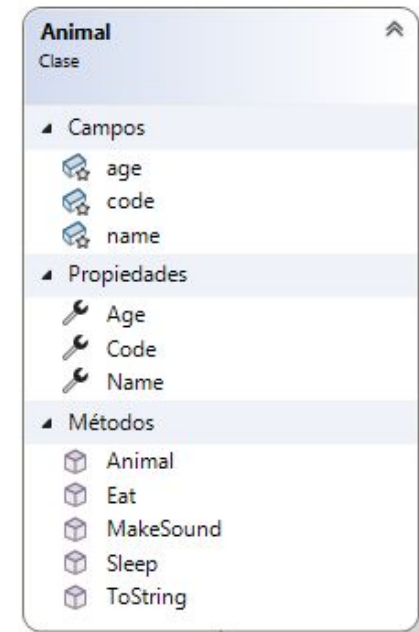


Propietats i mètodes virtuals

```
public virtual void MakeSound()  
{ Console.WriteLine("I LOVE MY OWNER"); }
```

La paraula virtual indica que un mètode (o propietat) pot ser sobreescrit per una subclasse.

Els mètodes Eat i Sleep no poden ser sobreescrits doncs no tenen la paraula virtual.



Classe Dog

1 referencia

```
public class Dog:Animal
```

```
{
```

0 referencias

```
public Dog(string code, string name, int age) : base(code, name  
{ }
```

2 referencias

```
public override void MakeSound()
```

```
{
```

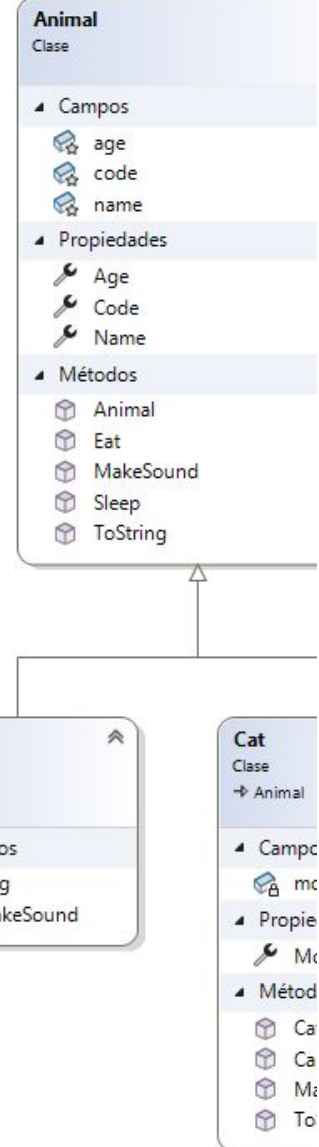
```
    base.MakeSound();
```

```
    Console.WriteLine("GUAU GUAU!");
```

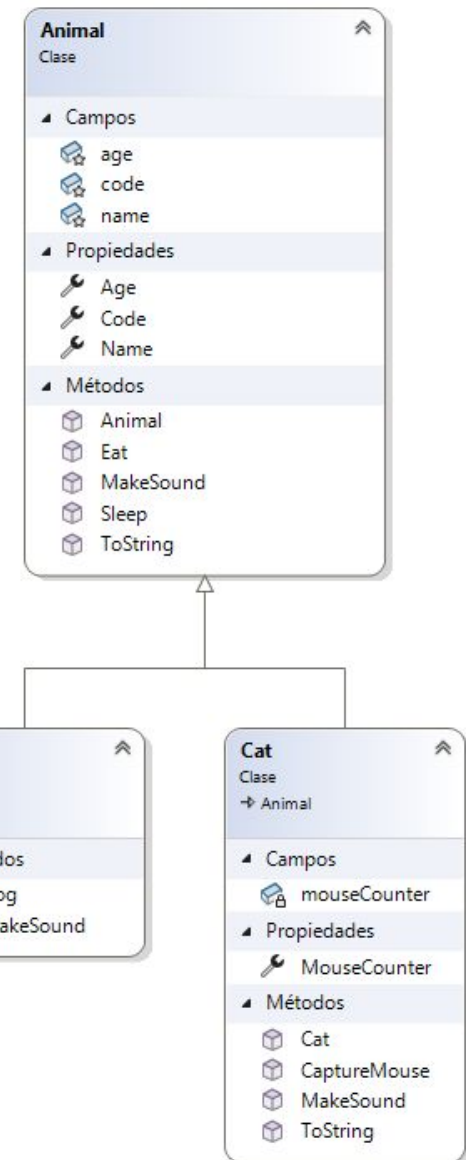
```
}
```

```
}
```

Analitza el codi que està subratllat.
Per a què serveix?



Classe Cat



```
public class Cat : Animal
{
    private int mouseCounter;
    2 referencias
    public Cat(string code, string name, int age) : base(code, name, age)
    { this.mouseCounter = 0; }

    5 referencias
    public override void MakeSound()
    {
        Console.WriteLine("MIAU MIAU!");
    }

    0 referencias
    public int MouseCounter
    {
        get { return mouseCounter; }
    }

    2 referencias
    public override string ToString()
    {
        return base.ToString() + ";" + mouseCounter;
    }

    3 referencias
    public void CaptureMouse()
    {
        mouseCounter++;
        Console.WriteLine("He caçat un ratolí.");
    }
}
```

Analitza el codi
que està subratllat.
Per a què serveix?

Tipus real i tipus aparent

- Un objecte “Cat” es pot assignar a una variable de tipus “Animal”, doncs un objecte “Cat” també és un “Animal”.

```
Animal a1 = new Cat("0001", "Felix", 80);
```

tipus aparent tipus real

```
a1.Sleep();  
a1.Eat();  
a1.MakeSound();  
//a1.CaptureMouse(); -> ERROR
```

Estic dormint
Estic menjant
MIAU MIAU!

De quina classe s'estan executant els diferents mètodes?

Test #1. Animals

HERENCIA

```
//Crear un gos, un gat i un animal
Dog dog1 = new Dog("0001", "Milú", 10);
Cat cat1 = new Cat("0002", "Gatsby", 15);
Animal a1 = new Animal("0003", "Turtle", 80);
```

```
//El gat caça tres ratolins
```

```
cat1.CaptureMouse();
cat1.CaptureMouse();
cat1.CaptureMouse();
Console.WriteLine("-----");
```

```
//Cridem el mètode Eat
```

```
cat1.Eat();
dog1.Eat();
a1.Eat();
Console.WriteLine("-----");
```

```
//Cridem el mètode MakeSound
```

```
cat1.MakeSound();
dog1.MakeSound();
a1.MakeSound();
Console.WriteLine("-----");
```

```
//Mostra el ToString de cada objecte
```

```
Console.WriteLine(dog1);
Console.WriteLine(cat1);
Console.WriteLine(a1);
```

De quina classe s'estan executant els diferents mètodes?

```
He caçat un ratolí.
He caçat un ratolí.
He caçat un ratolí.
```

```
-----
Estic menjant.
Estic menjant.
Estic menjant.
```

```
-----
MIAU MIAU!
I LOVE MY OWNER
GUAU GUAU!
I LOVE MY OWNER
```

```
-----
0001;Milú;10
0002;Gatsby;15;3
0003;Turtle;80
```

Test #2. Animals

POLIMORFISME

```
List<Animal> animals = new List<Animal>();
animals.Add(new Dog("0001", "Milú", 10));
animals.Add(new Cat("0002", "Gatsby", 15));
animals.Add(new Animal("0003", "Turtle", 80));

foreach (Animal a in animals)
{
    Console.WriteLine(a);
    a.MakeSound();
    Console.WriteLine();
}
```

```
0001;Milú;10
I LOVE MY OWNER
GUAU GUAU!
```

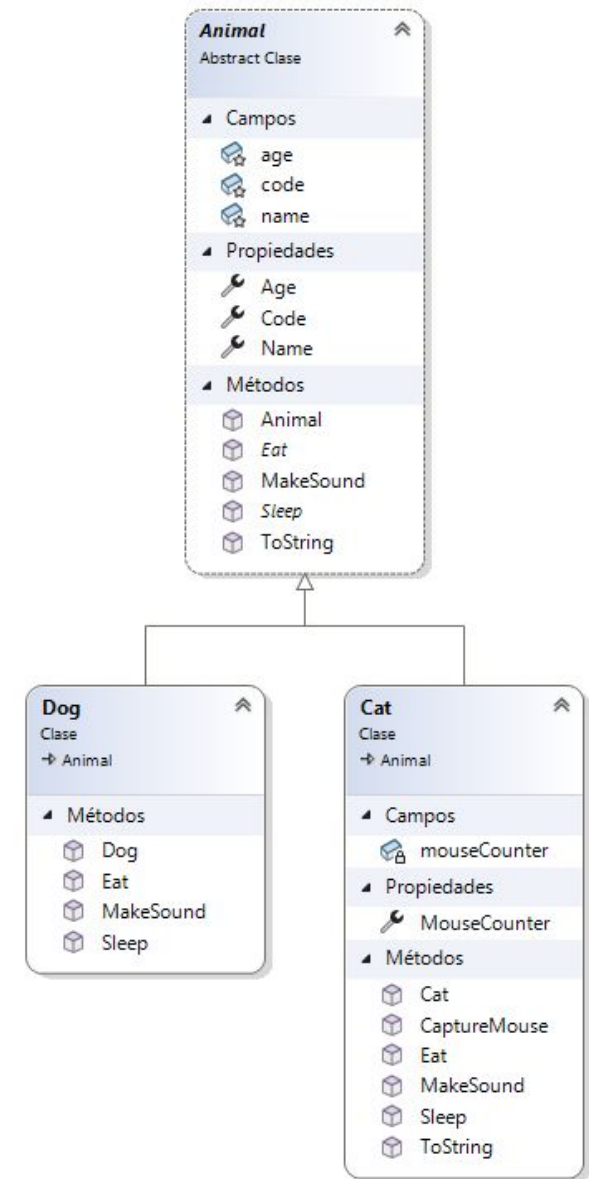
```
0002;Gatsby;15;0
MIAU MIAU!
```

```
0003;Turtle;80
I LOVE MY OWNER
```

- La llista d'animals conté objectes de tipus Animal (tipus aparent).
- En executar un mètode (o propietat) d'un animal, s'executa el mètode més pròxim al tipus real.

Classes abstractes

- Les classes abstractes tenen mètodes o propietats abstractes que han d'implementar les subclasses.
- No es possible crear un objecte d'una classe definida com a abstracte.
- La classe abstracte serveix com a base per implementar classes derivades (subclasse).



Classes abstractes

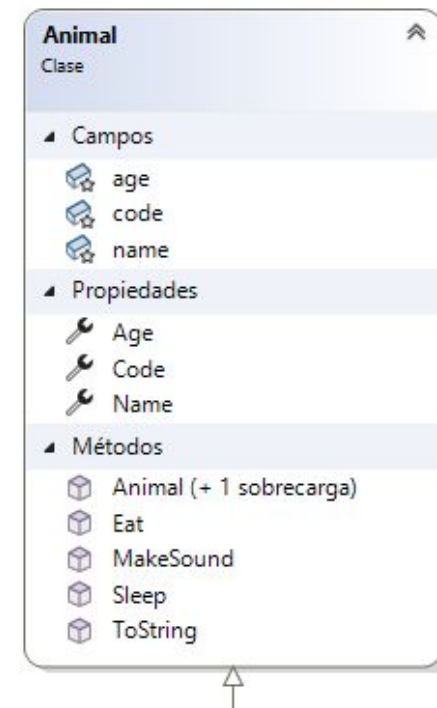
```
public abstract class Animal
{
    protected string code;
    protected string name;
    protected int age;

    1 referencia
    public string Code { get { return code; } }
    1 referencia
    public string Name { get { return name; } set { name = value; } }
    1 referencia
    public int Age { get { return age; } set { age = value; } }

    2 referencias
    public Animal(string code, string name, int age)
    { this.code = code; this.name = name; this.age = age; }

    2 referencias
    public abstract void Eat();
    2 referencias
    public abstract void Sleep();
    3 referencias
    public virtual void MakeSound()
    { Console.WriteLine("I LOVE MY OWNER"); }

    2 referencias
    public override string ToString()
    {
        return $"{Code};{Name};{Age}";
    }
}
```



Analitza el codi que està subratllat.
Busca informació sobre cada paraula o símbol i explica per a què serveix.

Classes abstractes

```
public class Cat : Animal
{
    private int mouseCounter;
    3 referencias
    public Cat(string code, string name, int age) ...
    2 referencias
    public override void MakeSound()...
    0 referencias
    public int MouseCounter...
    2 referencias
    public override string ToString()...
    0 referencias
    public void CaptureMouse()...

    4 referencias
    public override void Eat()
    {
        Console.WriteLine("Gat menjant.");
    }

    2 referencias
    public override void Sleep()
    {
        Console.WriteLine("Gat dormint.");
    }
}
```

```
1 referencia
public class Dog : Animal
{
    0 referencias
    public Dog(string code, string name, int age) ...
    2 referencias
    public override void MakeSound()...

    1 referencia
    public override void Eat()
    {
        Console.WriteLine("Gos menjant.");
    }

    1 referencia
    public override void Sleep()
    {
        Console.WriteLine("Gos dormint");
    }
}
```

Les subclasses han d'implementar tots els mètodes i propietats marcats com a abstract en la classe abstracta.

Classes abstractes

- Les classes abstractes no poden ser instanciades.

```
Animal a1 = new Animal("0003", "Turtle", 80);
```



**No es pot crear un
objecte sobre una classe
definida com a abstracte**

De les següents tres classes, quines poden ser instanciades?

- Animal?
- Dog?
- Cat?

Test #1. Animals (ABSTRACT)

```
//Crear un gos, un gat i un altre gos  
Dog dog1 = new Dog("0001", "Milú", 10);  
Cat cat1 = new Cat("0002", "Gatsby", 15);  
Animal a2 = new Dog("0003", "Ideafix", 80);
```

```
dog1.Eat();
```

```
cat1.Eat();
```

```
a2.Eat();
```

Gos menjant.

Gat menjant.

Gos menjant.

Observa que a2.Eat()
executa el mètode de
la classe Gos.

Test #2. Animals (ABSTRACT)

```
List<Animal> animals = new List<Animal>();  
animals.Add(new Dog("0001", "Milú", 10));  
animals.Add(new Cat("0002", "Gatsby", 15));  
animals.Add(new Dog("0003", "Ideafix", 80));  
  
for(int i=0;i<animals.Count;i++)  
{  
    Animal a = animals[i];  
    a.Eat();  
    a.Sleep();  
    Console.WriteLine();  
}
```

```
Gos menjant  
Gos dormint
```

```
Gat menjant  
Gat dormint
```

```
Gos menjant  
Gos dormint
```

Observa que a.Eat() executa el mètode de la classe Dog o Cat, en funció del tipus real de l'objecte.


Classes Sealed

- Una classe definida com Sealed no permet crear classes derivades a partir d'ella (tallem l'herència)

```
public sealed class Truck : IDeliverable
{
    private string name;
    private bool delivered;

    1 referencia
    public Truck(string name, bool delivered)
    {
```

```
0 referencias
public class CamioGran : Truck
{
    }
}
```

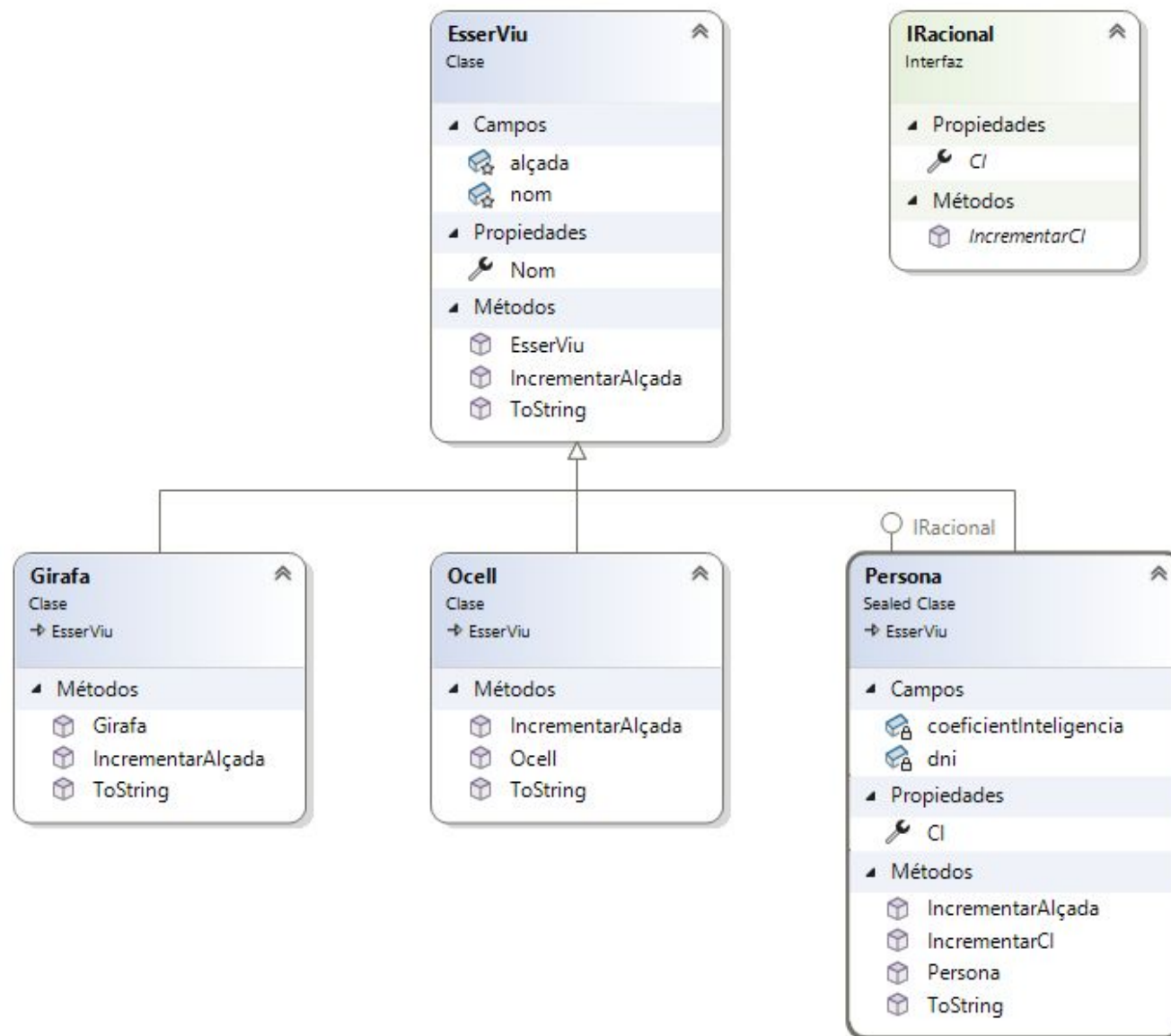
 class ExemplesHerència.Truck

CS0509: 'CamioGran': no puede derivar del tipo sellado 'Truck'

[Mostrar posibles correcciones](#) (Alt+Entrar o Ctrl+.)

DEMO ARCA DE NOE

Mètodes virtuals



DEMO LLOGUER VEHICLES

Classe abstracta i atribut estàtic

