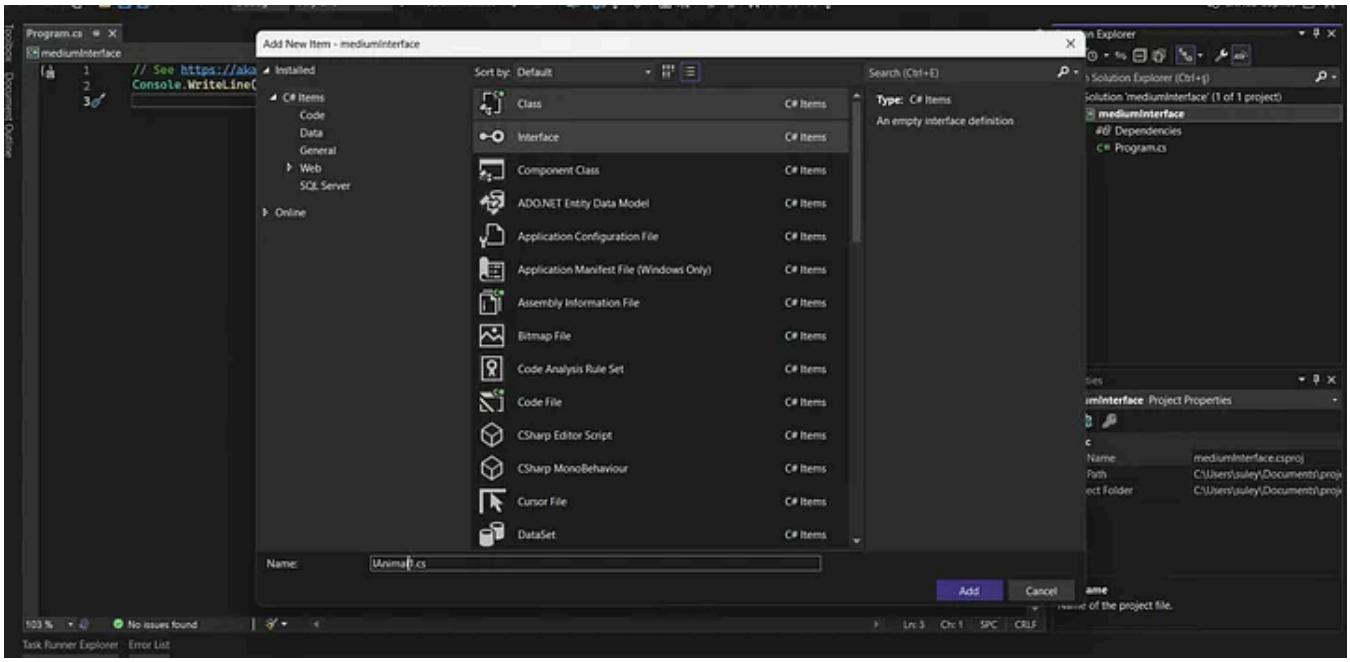
C# Interface Nedir? Nasıl Kullanılır? Örnekli Anlatım

Süleyman Meral
3 min read · Just now

Önceki yazımda **abstract class**'ların soyutlamayı nasıl sağladığından bahsetmiştim. Ancak soyutlamanın bir diğer önemli yöntemi de **Interface** kullanımıdır. Ve bu yöntem projelerde çokça kullanılır.

Interface OOP dillerinde kullanılan ve metotlara ait imzaları(signutare) içeren bir yapıdır. Yani içerisinde gövdesi olmayan metotlar tanımlarız ve bu yapıdan türetilen sınıflar metotların içeriğini belirler. İlgili interfaceden miras alan sınıflar bu Interface'de tanımlanan tüm metotları override etmeleri gerekmektedir. Abstract Classdan farklı olarak sadece gövdesiz metotları içerirler. Abstract classda ise hem gövdeli hem gövdesiz metotlar yazabiliriz. Abstract classda temel farkı ise bir sınıf sadece bir abstract classdan kalıtım alabiliyorken , birden fazla interface'den kalıtım alabilir.

Şimdi bir interface örneği yaparak anlattıklarımızı pekiştirelim.



Projemize yeni bir item ekleyip Interface'i seçiyoruz. Interfaceler isimlendirilirken genelde isminin başına "I" koyup isimlendirilir. Bizde ILivingThings adında canlılar alemini temsilen bir interface oluşturalım.

```
public interface ILivingThings
{
    void Live();

}
```

Kodda gördüğümüz gibi gövdesi olmayan fakat bu interfaceden türetilen tüm sınıflarda override edilmesi zorunlu olan bir metod tanımladık. Live metodu tüm canlılarda ortaktır. Şimdi ise hayvanlara ait özellikleri içeren bir interface daha tanımlayalım. Bunun ismi ise "IAnimal" olsun.

Hayvanlara ait 2 ortak özellik belirleyelim ve bunları gövdesiz metod olarak tanımlayalım.

```
public  interface IAnimal1
{
    void AnimalSound(); // No Body
    void Eat();         // No Body
}
```

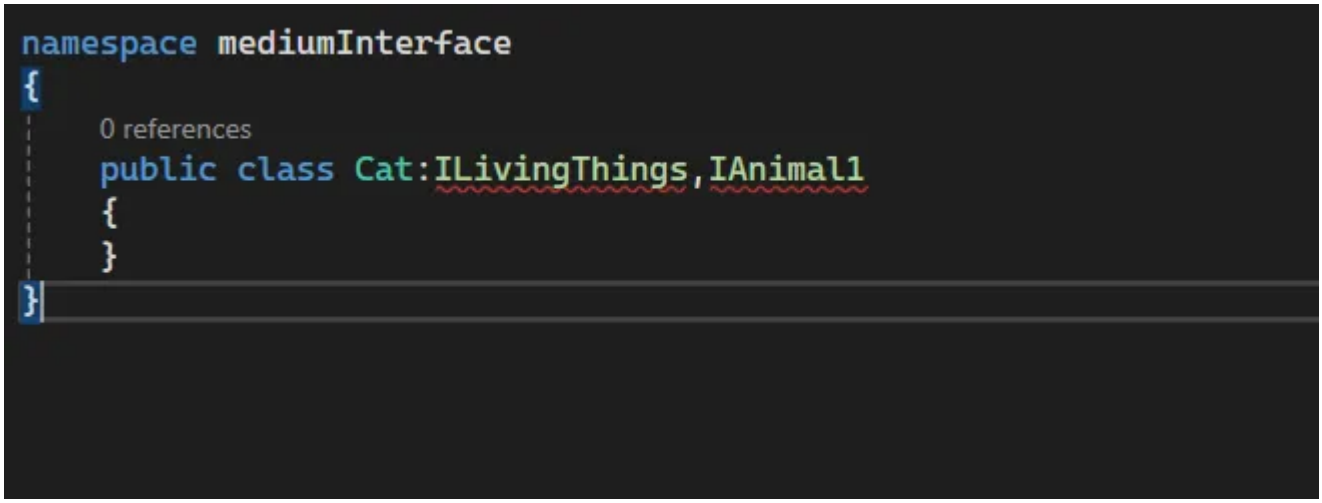
Tüm hayvanlar ses çıkarır ve yemek yer. Fakat bunu farklı şekillerde yapabilirler bu yüzden metodun içeriğini bu interfaceden miras alan classlardan girmesini isteyeceğiz.

Şimdi ise insanlara ait özellikleri tutacak bir interface daha tanımlayalım.

```
public interface IHuman
{
    void Work();

}
```

Tüm insanlar çalışabilir. Bu yüzden work adında bir fonksiyon tanımladık.

Şimdi Cat isimli bir class oluşturalım ve ilgili interfacelerden miras alalım.



Cat isimli classın 2 adet interfaceden miras almasını istedik. Kedi bir canlıdır ve aynı zamanda bir hayvandır. Bu yüzden bu interfacede tanımlanan özellikleri içermek zorundadık. Interface özelliklerini classımıza implement edelim.

```
0 references
public class Cat : ILivingThings, IAnimal1
{
    1 reference
    public void AnimalSound()
    {
        throw new NotImplementedException();
    }

    1 reference
    public void Eat()
    {
        throw new NotImplementedException();
    }

    1 reference
    public void Live()
    {
        throw new NotImplementedException();
    }
}
```

Interfacelerin üzerine gelip CTRL + . yapıp implement dediğimizde ilgili interface'e ait fonksiyonlar otomatik olarak oluşacaktır. Görüldüğü gibi interfacelerde tanımladığımız fonksiyonlar oluştu. Şimdi bunların içeriğini Cat classına göre kendimiz belirleyeceğiz.

```
public class Cat : ILivingThings, IAnimal1
{
    1 reference
    public void AnimalSound()
    {
        Console.WriteLine("Meow");
    }

    1 reference
    public void Eat()
    {
        Console.WriteLine("Cat Food");
    }

    1 reference
    public void Live()
    {
        Console.WriteLine("Living..");
    }
}
```

İçeriklerimizi kendimiz belirledik. Şimdi bir insan için aynı şekilde bir class tanımlayalım. Classımızın ismi Suleyman olsun. Bu class canlı ve insan nolduug için ILivingThings ve IHuman interfacelerinden miras alabilir.

```
0 references
public class Suleyman : ILivingThings, IHuman
{
    1 reference
    public void Live()
    {
        Console.WriteLine("Eat-Sleep-Coding-Repeat");
    }

    1 reference
    public void Work()
    {
        Console.WriteLine("Work As Software Engineer");
    }
}
```

İlgili interafcelerden miras alıp içeriklerini belirledik. Gördüğümüz üzere classlar birden fazla interfaceden miras alabiliyor. Burada abstract classdan miras alsaydık sadece bir adet classdan miras alabilirdik.

Örnekleri pekiştirmek açısından bir hayvan classı daha oluşturalım ve ilgili interfacelerden miras alalım.

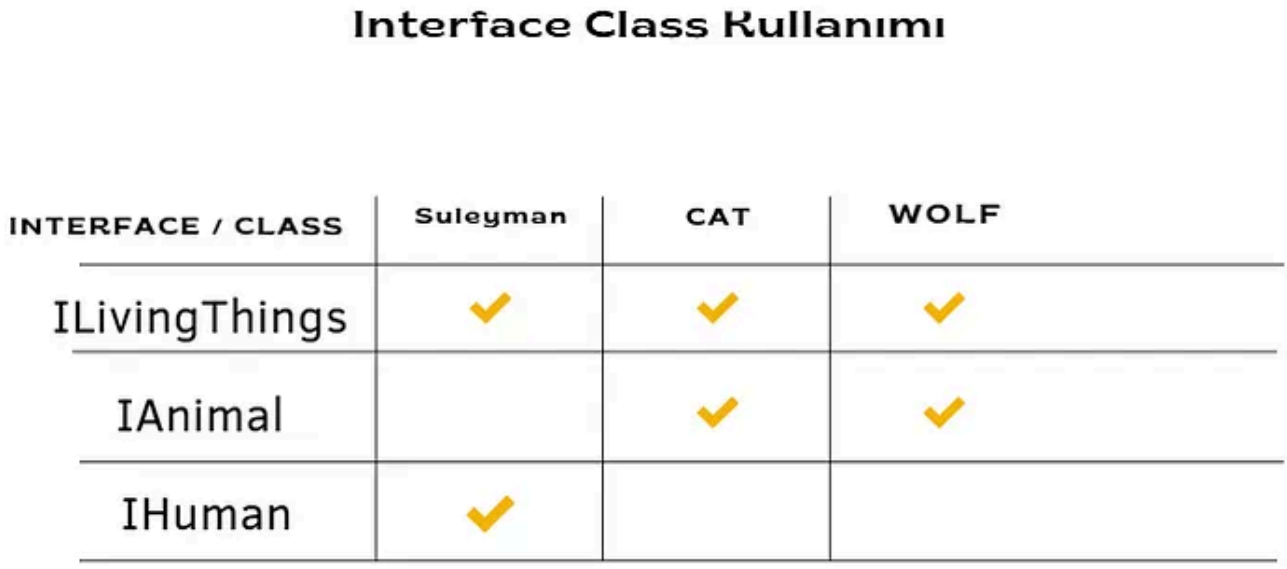
```
public class Wolf : IAnimal1, ILivingThings
{
    1 reference
    public void AnimalSound()
    {
        Console.WriteLine("Auu");
    }

    1 reference
    public void Eat()
    {
        Console.WriteLine("Eating meat");
    }

    1 reference
    public void Live()
    {
        Console.WriteLine("Live in forest");
    }
}
```

```
using mediumInterface;

Suleyman suleyman = new Suleyman();
Cat cat = new Cat();
Wolf wolf = new Wolf();

suleyman.Work();
suleyman.Live();
cat.Eat();
cat.AnimalSound();
wolf.Eat();
wolf.AnimalSound();
```

Classlarımızdan nesne türettik ve ilgili metotları çağırdık. Şimdi çıktılarını inceleyelim.

```
Work As Software Engineer
Eat-Sleep-Coding-Repeat
Cat Food
Meow
Eating meat
Auu
```

Görüldüğü üzere içerikler classlarda tanımladığımız şekilde oluştu.



Tabloda görüğümüz gibi her class kendi türüne göre olan interfacelerden miras aldı. Bu şekilde daha rahat ve daha sürdürülebilir bir kod yazmış olduk. Yeni bir insan veya bir hayvan tanımladığımız zaman içereceği fonksiyonları tekrardan düşünüp yazmak yerine ilgili interfacelerden miras alıp kolayca implement edebiliyoruz.

Yazilim Gelistirme | Nesne Tabanlı Programlama | Oop Concepts | C Sharp Programming | Software Engineering

Edit profile

Written by Süleyman Meral

0 Followers · 1 Following

Software Engineering Student/ .Net Developer