



4 min read · Feb 24, 2025



Süleyman Meral



Share



More

C# Abstract Class Nedir? Nasıl Kullanılır?

Herkese merhaba! 🌈 Medium üzerindeki ilk yazımda C#'ta Abstract Class yapısını ele almak istedim. Bu yazıda “Abstract Class Nedir?”, “Neden Kullanılır?” ve “Nasıl Kullanılır?” gibi sorulara yanıt vererek, konuyu genel bir bakış açısıyla inceleyeceğiz.

Abstract Class Nedir?

Abstract class, nesne yönelimli programlamada (OOP) kullanılan, doğrudan örneği oluşturulamayan ancak inheritance (miras) alınarak genişletilebilen bir sınıftır. İçerisinde **gövdesiz (abstract) metotlar** ve **gövdesi olan metotlar** barındırabilir. Abstract metotlar, alt sınıflar tarafından mutlaka **override edilmelidir**, yani her alt sınıf bu metotları kendine özgü şekilde tanımlamak zorundadır. Abstract class'lar, ortak davranışları belirleyerek kod tekrarını önler ve **tutarlı bir miras yapısı oluşturmayı sağlar**. Özellikle, **birden fazla alt sınıfın ortak özellik ve davranışlara sahip olması gereken durumlarda** kullanılır.

Şimdi visual studio üzerinden bir örnek yaparak konuyu pekiştirelim.

```
public abstract class Animal
{
    public abstract void AnimalSound();
    public void sleep()
    {
        Console.WriteLine("Zzz");
    }
}
```

Animal isimli bir Abstract sınıfı oluşturduk. Bu sınıfımızın içinde AnimalSound adında bir adet abstract fonksiyonu bulunuyor. Görüldüğü üzere bu fonksiyonumuzun bir gövdesi yok. Bu yüzden bu classdan miras alan classlar kendi içinde bu fonksiyonu override etmelidirler. Aslında temel amacımız bu sınıftan türetilen sınıflara zorunlu olarak bir metot sağlamak. Bu classda AnimalSound metodu bu görevi üstlenmektedir.

Ayrıca classımızda void türünde (geri dönüş değeri olmayan) bir metot daha tanımlanmış. Bu metot isteğe bağlı şekilde override edilebilir fakat bunun zorunluluğu yoktur direkt olarak da kullanılabilir.

“Cat” isimli bir class oluşturup bu classın abstract classımız olan Animal classından inheritance(miras) almasını sağlayalım.

```
namespace mediumAbstract
{
    2 references
    public class Cat : Animal // Animal classından miras alıyoruz
    {
    }
}
```

Cat classımız Animal classından miras alıyor. Yani bu classa ait özellikleri ve metotları kullanabilecek.

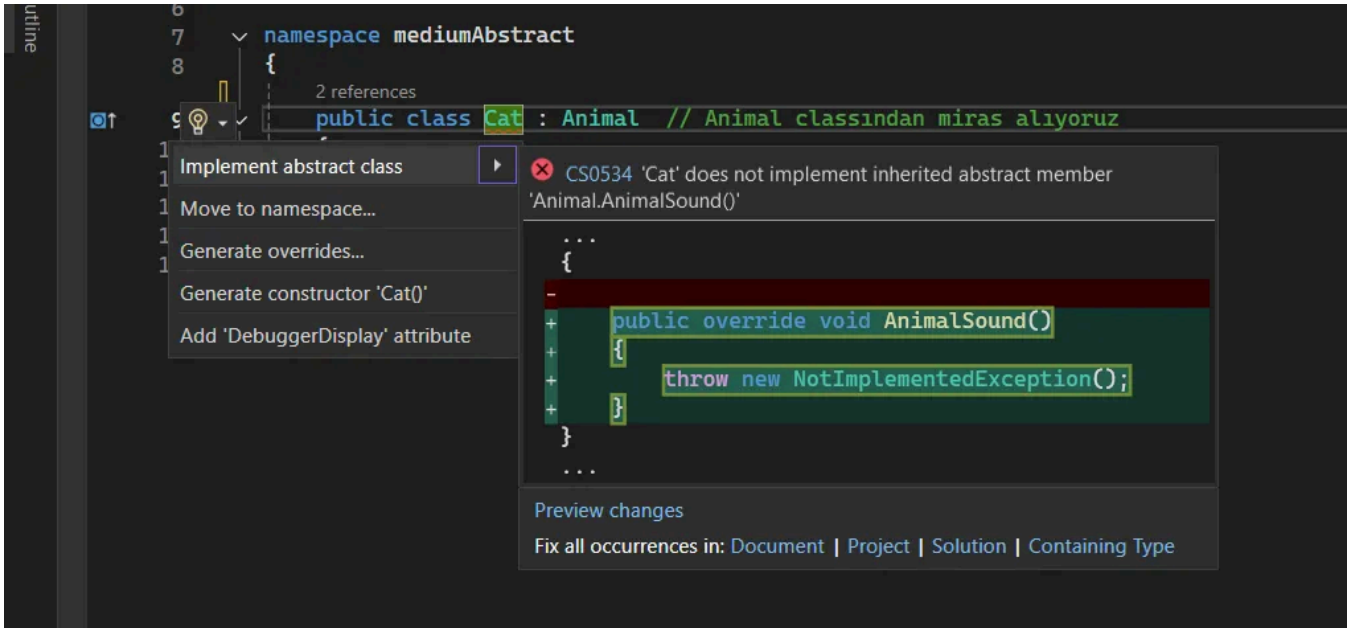
```
namespace mediumAbstract
{
    2 references
    public class Cat : Animal // Animal classından miras alıyoruz
    {
    }
}
```

class mediumAbstract.Cat

CS0534: 'Cat' does not implement inherited abstract member 'Animal.AnimalSound()'

Show potential fixes (Alt+Enter or Ctrl+.)

Inheritance uyguladığımız zaman Cat classımız yukarıda gördüğümüz uyarıyı veriyor. Cat classının üzerine gelip ctrl + . işlemi yaptığımızda abstract classımızı implement ediyoruz.



```
2 references
public class Cat : Animal // Animal classından miras alıyoruz
{
    1 reference
    public override void AnimalSound()
    {
        throw new NotImplementedException();
    }
}
```

İmplementasyon işleminden sonra override edilecek metodumuz otomatik olarak oluşuyor. Bu fonksiyonun gövdesini dilediğimiz gibi doldurabiliriz.

```
2 references
public class Cat : Animal // Animal classından miras alıyoruz
{
    1 reference
    public override void AnimalSound()
    {
        Console.WriteLine("Miaw");
    }
}
```

Metodumuzun gövdesinde basit bir komut yazdık.

Aynı işlemlere “Dog” adında başka bir class oluşturarak devam ediyoruz.

```
public class Dog : Animal
{
    public override void AnimalSound()
    {
        Console.WriteLine("woof");
    }
}
```

Dog classımızda farklı olarak Animal classımızda tanımladığımız Sleep adlı metodu da override edelim. Bu metodun isteğe bağlı şekilde override edilebileceğini söylemiştik.

İlk olarak Animal classımızda bulunan Sleep metodumuzu virtual olarak tanımlamamız gerekiyor. Virtual olarak tanımlanan metotlar alt sınıflar tarafından isteğe bağlı şekilde override edilebilir.

```
public abstract class Animal
{
    public abstract void AnimalSound();
    public virtual void Sleep()
    {
        Console.WriteLine("Zzz");
    }
}
```

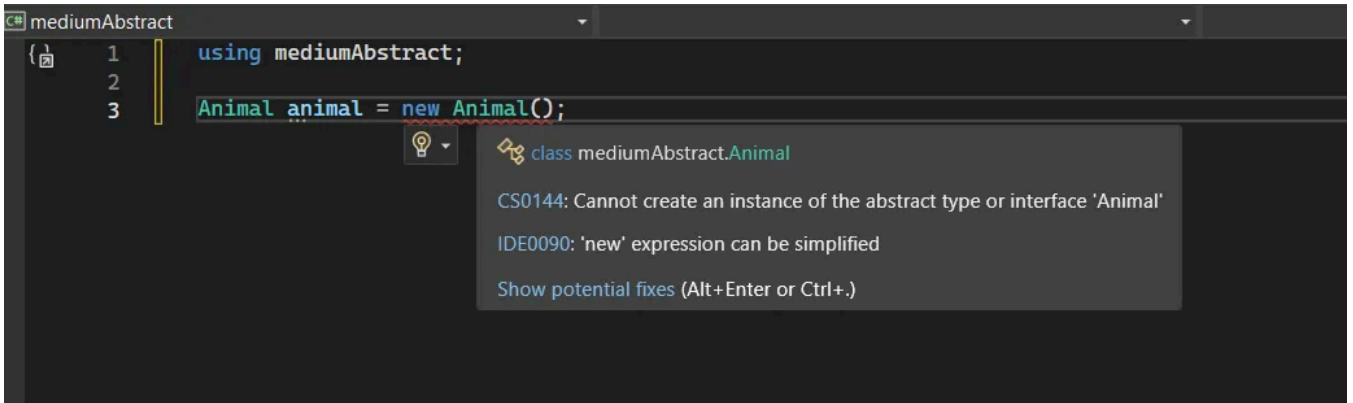
Animal classımızı güncelledik. Şimdi Dog isimli classımızda bu metodu da override edelim.

```
public class Dog : Animal
{
    public override void AnimalSound()
    {
        Console.WriteLine("woof");
    }
    public override void Sleep()
    {
        Console.WriteLine("sleeping..");
    }
}
```

```
}
```

Sleep metodumuzu override ederek içeriğini değiştirdik. Fakat Cat classımızda bu içerik Animal classında tanımlandığı gibi korundu.

Şimdi bu metodları çağırarak çıktılarına bakalım.



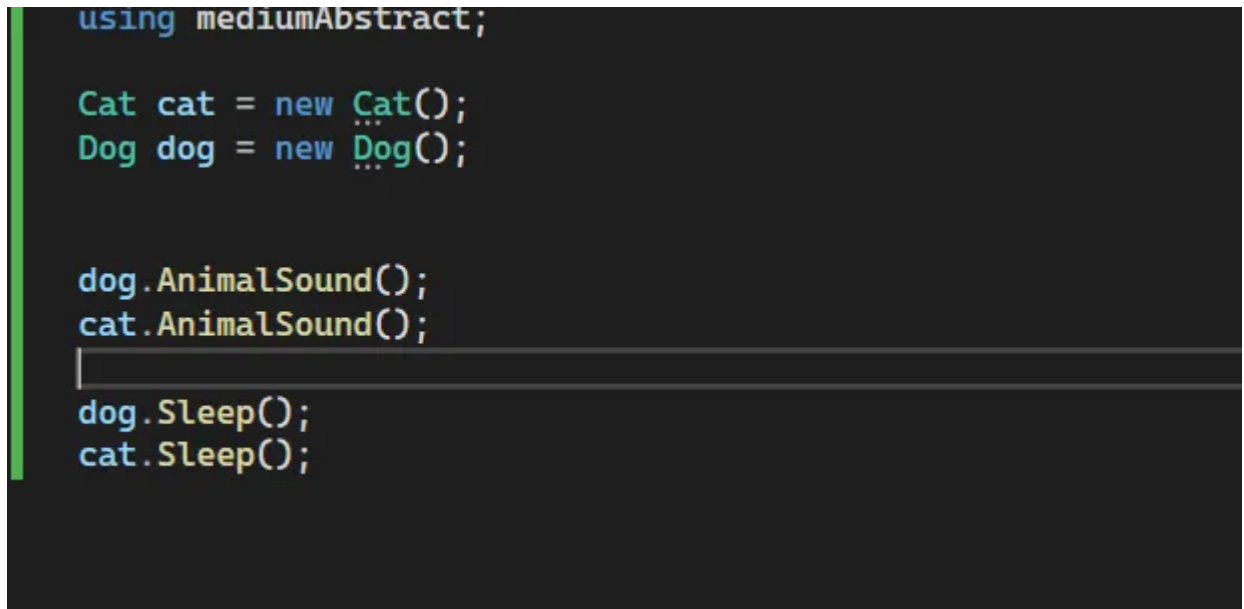
The screenshot shows a C# code editor with a file named 'mediumAbstract'. The code contains the following lines:

```
1 using mediumAbstract;  
2  
3 Animal animal = new Animal();
```

An error message is displayed below the code, indicating that 'Animal' is an abstract type and cannot be instantiated. The message includes the following details:

- class mediumAbstract.Animal
- CS0144: Cannot create an instance of the abstract type or interface 'Animal'
- IDE0090: 'new' expression can be simplified
- Show potential fixes (Alt+Enter or Ctrl+.)

Görselde gördüğümüz gibi Abstract sınıflardan bir instance türetilemez. Sadece ilgili classlara miras verir.



```
using mediumAbstract;  
  
Cat cat = new Cat();  
Dog dog = new Dog();  
  
dog.AnimalSound();  
cat.AnimalSound();  
  
dog.Sleep();  
cat.Sleep();
```

Cat ve Dog classlarımızdan birer instance türeterek içeriklerinde yer alan fonksiyonları çağırdık. Şimdi programımızı çalıştırarak çıktılarını inceleyelim.

```
Microsoft Visual Studio Debu x + v
woof
Miaw
sleeping..
Zzz
```

Görüldüğü üzere AnimalSound metodu içeriklerini belirlediğimiz gibi geldi. Sleep fonksiyonu ise Dog classında override edilip değiştirildiği için orada belirtilen şekilde geldi

Cat classında ise direkt olarak Animal classındaki içeriğe göre geldi. Çünkü o metodu Cat classında override etmemiştik.

Abstract class, diğer sınıflar için bir temel yapı sunar ve belirli metodların alt sınıflar tarafından mutlaka implement edilmesini sağlar. Bu sınıflar, doğrudan örneklenemezler, yani bir nesne oluşturulamaz. Soyut sınıflar, bir uygulamanın genel yapısını tanımlamak için kullanılır ve uygulamanın her alt sınıfının belirli metodları sağlamasını zorunlu kılar. Böylece, ortak özellikleri ve metodları bir arada tutarak, daha sürdürülebilir ve tutarlı bir kod yapısı sağlar. Abstract Class kullanarak, aynı zamanda kod tekrarını azaltabilir, farklı sınıfların belirli işlevleri tutarlı bir şekilde yerine getirmesini sağlayabilirsiniz. Alt sınıflar, soyut sınıfın içeriğini kendi gereksinimlerine göre özelleştirirken, ortak bir yapı üzerinden çalışarak, yazılımın bakımını ve genişletilmesini kolaylaştırır.

Software Development

Yazılım

Backend Development

Programlama



Edit profile

Written by Süleyman Meral

6 followers · 6 following

Backend Developer(.NET Core)