Sınıf Türleri ve farklılıkları

- *abstract class*
- *sealed class*
- *static class*
- *partial class*
- *interface*

Sınıf Türleri ve farklılıkları

- **abstract class**

Abstract Class, ortak özellikli Class'lara Base(taban) Class olma görevini üstlenir. Özellikle sınıftan örnek (nesne) türetilmesi istenmiyorsa abstract class kullanılır.

Örneğin bir Hayvan sınıfını temel alarak diğer hayvan türlerini tanımlamak isteyelim ama temel sınıf olan Hayvan sınıfı sadece türetme için kullanılıp bu sınıftan nesne türetilmesi istenmiyorsa temel sınıfı *abstract* olarak tanımlamalıyız.

Sınıf Türleri ve farklılıkları

- **abstract class**

Senaryo-1

```
abstract class Hayvan
{
    int Boy { get; set; }
    int Agirlik { get; set; }
    public Hayvan()...
}
class Kedi:Hayvan
{
   public Kedi()...
}
class Program
{
    static void Main(string[] args)
    {
        Kedi k1 = new Kedi();
        Hayvan h1 = new Hayvan();
    }
}
```

Hayvan.Hayvan()

Cannot create an instance of the abstract class or interface 'Hayvan'

Senaryo-2

```
class Hayvan
{
    int Boy { get; set; }
    int Agirlik { get; set; }
    public Hayvan()...
}
abstract class Kedi:Hayvan
{
   public Kedi()...
}
class yavruKedi:Kedi
{

}
class Program
{
    static void Main(string[] args)
    {
        Kedi k1 = new Kedi();
        yavruKedi yK1 = n
        Hayvan h1 = new H
    }
}
```

Kedi.Kedi()

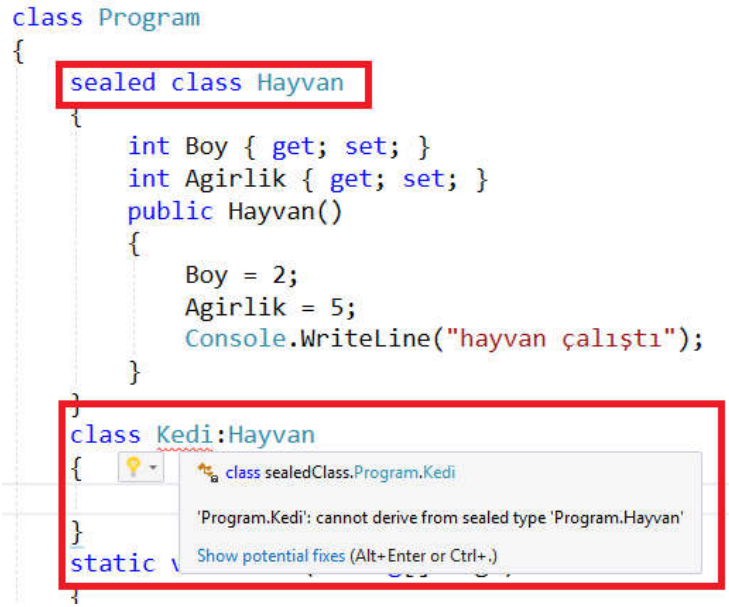
Cannot create an instance of the abstract class or interface 'Kedi'

Sınıf Türleri ve farklılıkları

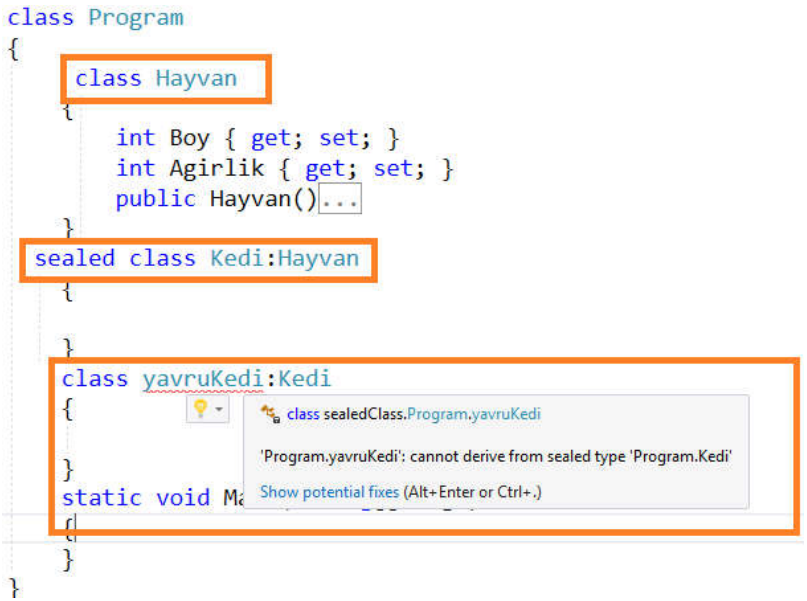
- **sealed class**

Sealed, sınıflara ve üyelere(metot ve property) uygulanabilen bir anahtar sözcüktür. Anlamı; kapalı, mühürlü şeklinde çevrilebilir. Sınıflar için kalıtımı, üyeler için ise override edilmeyi önler. Sealed classları bir çeşit güvenlik önlemi olarak düşünebiliriz. Yanlışlıkla türetilmeleri engellenmiş oluyor.

Senaryo-1



Senaryo-2



Sınıf Türleri ve farklılıkları

- **static class**

Statik üyelerdeki durumu şöyle kısaca bir hatırlarsak, statik bir üye oluşturulduktan sonra ilgili class altındaki statik bir üyeye erişebilmek için ilgili class için bir nesne almadan ulaşabileceğimizi, hatta nesne aldıktan sonra bu üyelere ulaşamadığımızı anımsayabiliriz.

İçinde sadece statik üye olan bir class için hiç bir zaman bu class'tan türeyecek nesneye ihtiyacımızın olmayacağını söyleyebiliriz o halde ama bu durum şu demek değildir, bu class'tan nesne oluşturamayacağımız anlamına gelmemelidir.

Peki hangi durumlarda static class tercih edilebilir? Bu sorunun cevabı nesneyebağımlı olmayan işlemlerin yapıldığı durumlar olarak açıklanabilir. Static Class'lar sadece statik üye içerebilirler, diğerleri tanımlanamazlar ve mutlaka üyelerin static keyword'ü eklenmelidir.

Sınıf Türleri ve farklılıkları

- **static class**

Örneğin Hayvan sınıfı içinde ortalama() isimli bir fonksiyon kullanılacak bu fonksiyon Hayvan sınıfı ile direk ilintili bir metot değildir. Bu sebepten bu metotu direk sınıf içinden çağırmak için statik tanımlayabiliriz.

Lakin nesne tanımladığınızda bu metotu kullanma gereği duymayacağız/kullanamayacağız fakat bellek alanı işgal edecektir. Bundan kaçınmak bu metot için statik bir sınıf tanımlanıp kullanılabilir.

Sınıf Türleri ve farklılıkları

- **static class**

```
static class Ortalama
    {
        public static void ort()
        {
            return;
        }
    }
    class Hayvan
    {
        int Boy { get; set; }
        int Agirlik { get; set; }
        public Hayvan()
        {
            Boy = 2;
            Agirlik = 5;
            Console.WriteLine("hayvan çalıştı");
        }
    }
```

```
class Program
    {
        static void Main(string[] args)
        {
            Hayvan h1 = new Hayvan();
            Ortalama.ort();
        }
    }
```

Sınıf Türleri ve farklılıkları

- **partial class**

Büyük projelerde oluşturduğumuz class' lar zamanla okunması zor hale gelebilecek kod satırları ile dolabilmektedir. Partial class ise bir class' ı birden fazla class olarak bölmemize olanak sağlar. Fiziksel olarak birden fazla parça ile oluşan partial class' lar, Çalıştığı zaman tek bir class olarak görev yapar.

Partial class ile fiziksel olarak parça class'ların birleşmesi için class isimlerinin aynı olması gerekmektedir.

Sınıf Türleri ve farklılıkları

- **partial class**

```
public partial class Personeller
    {
        public Personeller()
        {
        }

public Personeller(Personeller personel)
        {
            PersonelEkle(personel);
        }
    }
  public partial class Personeller
     {
         private string adi;
         private string soyadi;
         private int yas;
         private List<Personeller> personel = new List<Personeller>();
     }
```

SINIF TÜRLERİ

- **abstract class**

Should be used when there is a IS-A relationship and no instances should be allowed to be created from that abstract class. Example: An Animal is an abstract base class where specific animals can be derived from, i.e. Horse, Pig etc. By making Animal abstract it is not allowed to create an Animal instance.

- **interface**

An interface should be used to implement functionality in a class. Suppose we want a horse to be able to Jump, an interface IJumping can be created. By adding this interface to Horse, all methods in IJumping should be implemented. In IJumping itself only the declarations (e.g. StartJump and EndJump are defined), in Horse the implementations of these two methods should be added.

- **sealed class**

By making Horse sealed, it is not possible to inherit from it, e.g. making classes like Pony or WorkHorse which you like to be inheriting from Horse.

- **static class**

Mostly used for 'utility' functions. Suppose you need some method to calculate the average of some numbers to be used in the Horse class, but you don't want to put that in Horse since it is unrelated and it also is not related to animals, you can create a class to have this kind of methods. You don't need an instance of such a utility class.

- **partial class**

A partial class is nothing more than splitting the file of a class into multiple smaller files. A reason to do this might be to share only part of the source code to others. If the reason is that the file gets too big, think about splitting the class in smaller classes first.