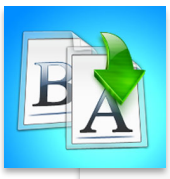
SINIF TÜRLERİ



İÇİNDEKİLER

- Giriş
- Soyut Sınıflar
- Mühürlü Sınıflar
- Statik Sınıflar ve Üyeleri
  - Statik Üyeler
  - Statik Alanlar
  - Statik Metotlar
  - Statik Yapıcılar
- Kısmi Sınıflar
- İç İçe Sınıflar

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Nesne yönelimli programlama içinde önemli bir yer tutan sınıflar ve sınıf türleri hakkında daha detaylı bilgilenecek,
  - soyut sınıflar, mühürlü sınıflar, statik sınıflar ve üyeleri konusunda kavramsal bilgiye sahip olacak ve C# içinde kullanımını örnek kodlar ile öğrenecek,
  - Kısmi sınıflar ve iç içe sınıfların nesne yönelimli programlama için önemini ve programlama sürecimizde nerelerde kullanılır örnek kodlar ile öğreneceksiniz.

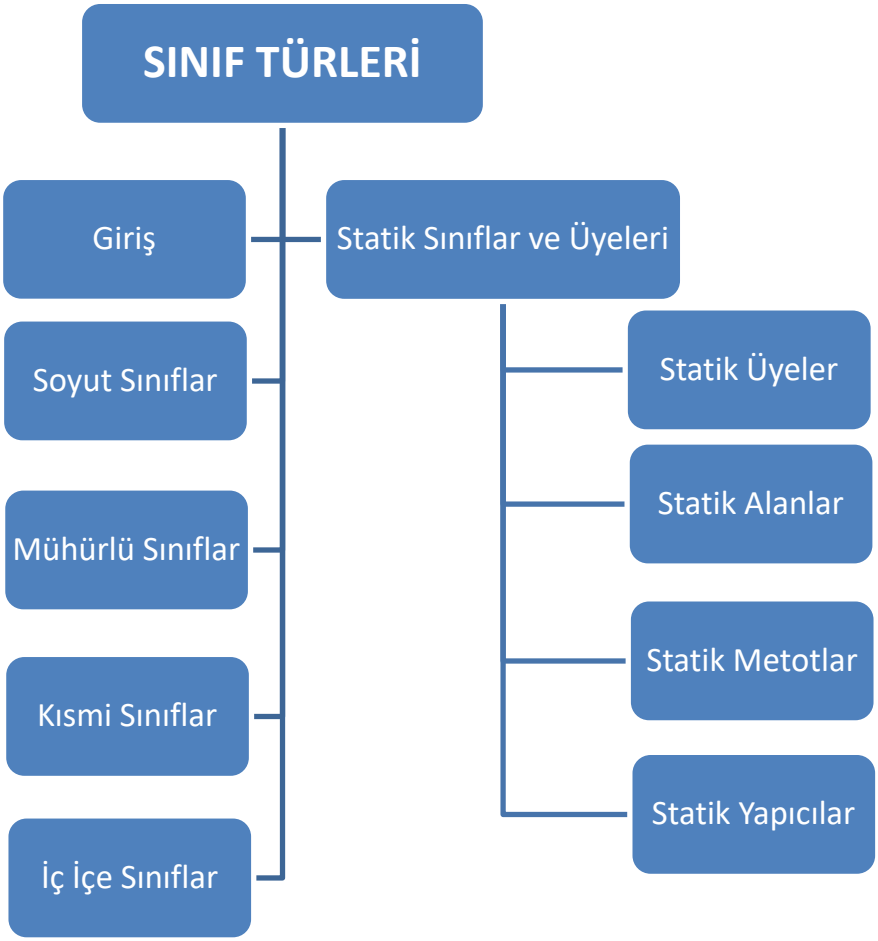
Atatürk Üniversitesi
Açıköğretim Fakültesi

NESNE TABANLI PROGRAMLAMA I

Dr. Muhammet DAMAR

ÜNİTE

3


SINIF TÜRLERİ
Giriş
Soyut Sınıflar
Mühürlü Sınıflar
Kısmi Sınıflar
İç İçe Sınıflar
Statik Sınıflar ve Üyeleri
Statik Üyeler
Statik Alanlar
Statik Metotlar
Statik Yapıcılar

GİRİŞ

Bir sınıf, bir referans türüdür, sınıfın bir nesnesi oluşturulduğunda, nesnenin atandığı değişken yalnızca o belleğe bir referans tutar.

Hızla dönüşen ve gelişen bilişim teknolojileri farklı donanımlar üzerinde çalıştırılabilen yazılım teknolojilerinin ve programlama tekniklerinin gelişip dönüşmesine sebep olmuştur. Visual Studio .NET birkaç programlama dilini aynı anda kullanmaya imkan veren bir altyapıya sahiptir. Bu özellik, bütünleşik yazılım geliştirme özelliği olarak tanımlanır. Bu sayede C#, Basic. NET, C++, F# gibi birçok programlama diline görsel ve konsol programlama geliştirmek için imkan sağlamaktadır.

Nesneye yönelimli programlama için sınıf yapısı bir önceki bölümde de detaylı olarak ortaya konulduğu gibi kritik önemdedir. Özellikle belleği etkin kullanmak, program içinde her şeyin sınıflar, sınıflara ait nesneler ve metotlar üzerinden yönlendirilmesi tüm bu süreç için bellek ortamından etkin bir şekilde faydalanılması nesne yönelimli programlamanın en önemli avantajı ve karakteristik özelliğidir.

public, protected, protected internal, internal ve private değiştiricileri sınıfının erişilebilirliğini denetler.

Sınıf veri üyeleri (sabitler ve alanlar), işlev üyeleri (yöntemler, özellikler, olaylar, dizin oluşturucular, işleçler, örnek oluşturucular, Yıkıcılar ve statik oluşturucular) ve iç içe türler içerebilen bir veri yapısıdır. Sınıf türleri, türetilmiş bir sınıfın bir temel sınıfı genişletebileceği ve özelleştirileceği bir mekanizma olan devralmayı destekler.

İzin verilen erişim değiştiricileri ve bir tür bildirimine yönelik varsayılan erişim, bildirimin gerçekleştiği bağlama göre değişir. Bu ünite kapsamında soyut sınıf, mühürlü sınıf, kısmi sınıf ve iç içe sınıf türleri incelenecektir. Söz konusu bu sınıf türlerine erişim ise erişim denetleyiciler ile sağlanmaktadır. Bu erişim denetleyiciler kod güvenliği ve kodların tekrarlı kullanımı hususunda kullanıcısına önemli avantajlar sağlar. C#'da beş farklı erişim değiştirici mevcuttur. Bunlar sırasıyla herkese açık kullanım (*public*), korumalı kullanım (*protected*), dahili kullanım (*internal*), korumalı dahili kullanım (*protected internal*) ve özel kullanım (*private*) şeklindedir. Sınıflarda belirtilen türlerin, *public protected, internal protected*, *internal* veya *private* erişimi olabilir. Varsayılan olarak *private erişim*' dir.

C#'da beş farklı erişim değiştirici mevcuttur. Bunlar sırasıyla herkese açık kullanım (public), korumalı kullanım (protected), dahili kullanım (internal), korumalı dahili kullanım (protected internal), özel kullanım (private).

Örnek

- *Bir yöntem için erişim değiştiricisi belirtmezseniz, varsayılan erişim değiştiricisi nedir?*
  - >>> Özel (Private)
- *Sınıflar ve yapılar kalıtımı destekler. Bu ifade doğru mudur, yanlış mıdır?*
  - >>> Yanlış, yalnızca sınıflar kalıtımı destekler. Struct kalıtımı desteklemez.
- Bir sınıf başka bir sınıftan türetilirse, türetilmiş sınıf otomatik olarak temel sınıfın tüm public, protected ve internal üyelerini içerecek mi?
  - >>> Evet, türetilmiş sınıf, yapıcıları ve yıkıcıları dışında temel sınıfın tüm public, protected ve internal üyelerini otomatik olarak içerecektir.

Bir sınıfı deklare etmek *(class_declaration)* , isteğe bağlı olarak bir sınıf değiştiricileri dizisi içerebilir. Aynı değiştiricinin bir sınıf bildiriminde birden çok kez görünmesi için derleme zamanı hatası vardır. Bu değiştiriciler şu şekilde ifade edilebilir:

Örnek

- ***: 'new'***
- ***| 'public'***
- ***| 'protected'***
- ***| 'internal'***
- ***| 'private'***
- ***| 'abstract'***
- ***| 'sealed'***
- ***| 'static'***
- ***| class_modifier_unsafe***

Public, *protected*, *protected internal*, *internal* ve *private* değiştiricileri sınıfının erişilebilirliğini denetler. Sınıf bildirimleri ve kullanımlar üzerinde çalıştırılan programın yapısına, kullanılan erişim denetleyici kurgusuna göre değişmektedir. Nesne yönelimli programlamanın bu yönü kod yazan programcı tarafından titizlikle hesaplanması ve kurgulanması gereken bir diğer unsur olduğu gözden kaçırılmamalıdır. Sınıf kavramı bir önceki bölümde detaylı olarak değerlendirildiğinden erişim denetleyicilerinin detayına bu bölümde girilmemektedir. Bu bölüm içinde sınıf türleri üzerinde değerlendirme yapılacak, örnek kodlar ile konu sizlere açıklanmaktadır.

Bu ünite kapsamında *statik sınıflar* ve sınıf üyeleri, *kısmi sınıflar* (*partial classes*) ve *iç içe sınıflar* (*nested classes*) açıklanmaktadır. Ayrıca *abstract*, *sealed* ve *static* değiştiricilere değinilecektir. Verilen örnekler Visual Studio 2019 Console ve Windows Forms uygulamalarında C# dili ile geliştirilmiştir.

Veri soyutlama, belirli ayrıntıları gizleme ve kullanıcıya yalnızca temel bilgileri gösterme işlemidir.

SOYUT SINIFLAR (ABSTRACT CLASS)

Soyutlama, yalnızca temel nitelikleri "gösteren" ve gereksiz bilgileri "gizleyen" nesne yönelimli programlama kavramıdır. Soyutlamanın temel amacı, gereksiz detayları kullanıcılardan gizlemektir. Soyutlama, nesnenin yalnızca ilgili ayrıntılarını kullanıcıya göstermek için daha büyük bir havuzdan veri seçmektir. Programlama karmaşıklığını ve çabalarını azaltmaya yardımcı olur. Nesne yönelimli programlamanın önemli kavramlarından biridir. Soyutlama, soyut sınıflar veya arayüzler ile gerçekleştirilebilir (Nesne Yönelimli Programlama II dersinizde detaylandırılacaktır.). *Abstract* değiştirici, bir sınıfın tamamlanmamış olduğunu ve yalnızca temel sınıf olarak kullanılması amaçlanan olduğunu göstermek için kullanılır. Soyut anahtar kelime, sınıflar ve yöntemler için kullanılır:

- *Soyut sınıf:* Nesneler oluşturmak için kullanılamayan kısıtlı bir sınıftır (ona erişmek için başka bir sınıftan miras alınması gerekir).
- *Soyut yöntem:* Yalnızca soyut bir sınıfta kullanılabilir ve bir gövdesi yoktur. Gövde, türetilmiş sınıf tarafından sağlanır (kalıtsal).

En az bir üye işlevi saf sanal işlev yapmak, soyut sınıf yapma yöntemidir. Soyut bir sınıf hem soyut hem de normal yöntemlere sahip olabilir. Bir sınıf birden çok soyut sınıftan miras alamaz. Soyut bir sınıf, soyut olmayan bir sınıftan aşağıdaki yollarla farklılık gösterir:

- Soyut bir sınıf doğrudan başlatılamaz ve bir derleme zamanı hatası, new anahtar kelimesi bir soyut sınıfta kullanılır.
- Derleme zamanı türleri soyut olan değişkenler ve değerler olması mümkün olsa da, bu tür değişkenler ve değerler, null soyut türlerden türetilmiş soyut olmayan sınıfların örneklerine başvuru ya da içermeli.
- Soyut bir sınıfa, soyut Üyeler içermesi için izin verilir (ancak gerekli değildir).
- Soyut bir sınıf korumalı olamaz.

```
abstract class Hayvan
    {
        public abstract void hayvanSesi();
        public void uykuModu()
        {
            Console.WriteLine("Horrrrr… Horrrrr…");
        }
    }
```

Soyut yöntem: Yalnızca soyut bir sınıfta kullanılabilir ve bir gövdesi yoktur.

Aşağıdaki kod örneği ile *Hayvan* sınıfının bir nesnesini oluşturmak mümkün değildir:

```
Hayvan benimTurum = new Hayvan();
// Bir hata oluşturacak (Bir örneği oluşturulamıyor
//soyut sınıf veya arayüz 'Hayvan')
```

Soyut sınıfa erişmek için başka bir sınıftan miras alınması gerekir. *hayvan* sınıfını bir *abstract* sınıfa aşağıdaki şekilde çevirebiliriz. Ekran çıktısı için ilk etapta

hayvan sınıfından kalıtlanan kedi sınıfındaki *hayvansesi()* metodu main içerisinde çalıştırılırken ikinci aşamada hayvan sınıfı içindeki *uyu()* metodu çalıştırılmaktadır.

```
// Soyut Sınıf Örneği
abstract class Hayvan
{
    // Soyut methot bir gövdeye sahip değildir.
    public abstract void HayvanSesi();
    // Uygun Method
    public void Uyu()
    {
        Console.WriteLine("Horrrr… Horrrr…");
    }
}
// Üretilen Ayi Sınıfı (hayvan sınıfından kalıtlıyoruz.)
class Kedi : Hayvan
{
    public override void HayvanSesi()
    {
        // HayvanSesi()'un gövdesi burada sağlanır
        Console.WriteLine("Kediler Uyur: marrrrr marrrr");
    }
}
class Program
{
    static void Main(string[] args)
    {
        Kedi datluKedim = new Kedi();
        // Burada bir kedi nesnesi oluturuyoruz
        datluKedim.HayvanSesi();
        // soyut sınıfın methodunu burada çağırıyoruz
        datluKedim.Uyu();
        // Uygun metodu burada çağırıyoruz.
    }
}
```

MÜHÜRLÜ SINIFLAR (SEALED CLASS)

Nesneye yönelimli programlamanın en önemli özelliklerinden birisi olan bir sınıftan başka sınıflar üretebilme durumu, güvenlik gibi çeşitli sebepler ile istenmeyebilir.

Bu sınıflardan türemenin istenmemesi durumunda sınıfın başına mühürlendiğini yani bu sınıftan türetme yapılamayacağını gösteren *sealed* anahtar sözcüğü kullanılır. Bu sayede de sabit özellikler ve metotlara sahip olan sınıflar elde edilebilir.

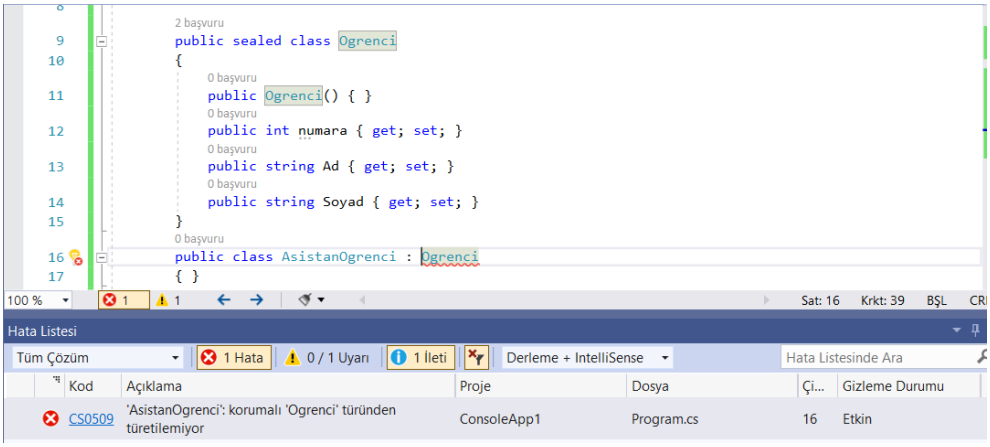
Mühürlü sınıflar (sealed class) hiçbir sınıfın kendisinden türetilemeyeceği açık ve net olan sınıflardır.

Bu duruma uyulmadığında yani bir *mühürlü* sınıf, başka bir sınıfın temel sınıfı olarak belirtilmişse, derleme zamanı hatası ile karşılaşılmaktadır. Ayrıca *mühürlü* bir sınıf de soyut bir sınıf olamaz. G*enellikle* istenmeden türetmenin önlenmesi için kullanılan mühürlü sınıflar kimi zaman program kodları için çalışma zamanı iyileştirmeleri de sağlamaktadır.

Ayrıca, korumalı bir sınıfın hiç türetilmiş sınıfa sahip olmadığı bilindiğinden, korumalı sınıf örneklerine sanal işlev üye çağırmaları sanal olmayan çağırmaları dönüştürmek mümkündür.

```
public sealed class Ogrenci
{
   public Ogrenci() { }
   public int Numara { get; set; }
   public string Ad { get; set; }
   public string Soyad { get; set; }
}
public class AsistanOgrenci : Ogrenci
{ }// türetme işlemi
```

Eğer *Ogrenci* sınıfından *LisansOgrenci* isminde yeni bir sınıf türetilmesi istenirse aşağıdaki şekilde tanımlanır. Fakat *Ogrenci* sınıfı mühürlü sınıf (*sealed class*) olduğu için aşağıdaki Resim 3.1.'deki hata mesajı alınır.



Resim 3.1. Mühürlü Sınıf Türetme İşleminde Karşılaşılan Hata Kodu

STATİK SINIFLAR VE ÜYELERİ

C#'da statik, somutlaştırılamayan bir şey anlamına gelir. Statik bir sınıfın nesnesini oluşturamazsınız ve bir nesne kullanarak statik üyelere erişemezsiniz.

Statik bir sınıfın nesnesi oluşturulamaz ve bir nesne kullanılarak statik üyelere erişilemez.

C# sınıfları, değişkenler, metotlar, özellikler, operatörler, olaylar ve yapıcılar, *static* değiştirici anahtar sözcüğü kullanılarak statik olarak tanımlanabilir.

Statik bir sınıf, temelde statik olmayan bir sınıfla aynıdır, ancak bir fark ile; statik bir sınıf somutlaştırılamaz. Başka bir deyişle, sınıf türünde bir değişken oluşturmak için *new* anahtar sözcüğünü kullanamazsınız. Statik bir sınıf, *sealed* veya *abstract* değiştiricisini içeremez. Ancak, statik bir sınıf tarafından örneklenemez veya türetilmediği için, hem korumalı hem de soyut gibi davranır.

Dizin oluşturucular (indexers) ve yıkıcılar (destructors) statik olamaz.

Statik Sınıf Kuralları

- Statik sınıflar bir örnekle temsil edilemez.
- Statik bir sınıfın tüm üyeleri statik olmalıdır; aksi halde derleyici hata verecektir.
- Statik bir sınıf, statik değişkenler, statik yöntemler, statik özellikler, statik operatörler, statik olaylar ve statik oluşturucular içerebilir.
- Statik bir sınıf, örnek üyeler ve kurucular içeremez.
- Dizin oluşturucular *(indexers)* ve yıkıcılar *(destructors)* statik olamaz.

- *var*, statik üyeleri tanımlamak için kullanılamaz. Statik anahtar kelimeden sonra açıkça bir üye türü belirtmelisiniz.
- Statik sınıflar mühürlü sınıflardır (mühürleme, sınıfların ve arayüzlerin izin verilen alt türleri üzerinde daha fazla kontrole sahip olmasını sağlar; bu, hem genel etki alanı modellemesi hem de daha güvenli platform kitaplıkları oluşturmak için kullanışlıdır.) ve bu nedenle miras alınamazlar.
- Statik bir sınıf diğer sınıflardan miras alamaz.
- Statik sınıf üyelerine *SinifAdi.SinifUyesininAdi* kullanılarak erişilebilir.
- Statik bir sınıf, programınızın bulunduğu uygulama etki alanının ömrü boyunca bellekte kalır.
- Statik bir sınıf, erişilebilirliği *protected* veya *protected internal* olan veya tarafından tanımlanan üyelere sahip olamaz.

Statik bir sınıf, erişilebilirliği protected veya protected internal olan veya tarafından tanımlanan üyelere sahip olamaz.

Statik bir kurucu yalnızca bir kez çağrılır ve statik bir sınıf, programınızın bulunduğu uygulama etki alanının ömrü boyunca bellekte kalır.

Statik bir sınıf, temelde statik olmayan bir sınıfla aynıdır, ancak bir farkla: Statik bir sınıf somutlaştırılamaz.

Statik sınıflar bir örnekle temsil edilemez.

Bu kısıtlamalardan herhangi birini ihlal etmek, yazdığımız kodlar için derleme hatası ile karşılaşmamız anlamına gelecektir. Kod yazarken bu kuraları gözden kaçırmamanız gerekmektedir.

Aşağıda örnek kurusunda da görüleceği gibi, *MetotX* adında bir genel yöntemi olan *TestSinifi* adında bir statik sınıfınız varsa, yöntemi aşağıdaki örnekte gösterildiği gibi çağırırsınız. Statik bir sınıf, yalnızca giriş parametreleri üzerinde çalışan ve herhangi bir dahili örnek alanı alması veya ayarlaması gerekmeyen yöntem kümeleri için uygun bir kapsayıcı olarak kullanılabilir.

Örnek

```
•TestSinifi.MetotX();
```

Visual Studio .NET sınıf kitaplığında, statik *System.Math* sınıfı, Math sınıfının belirli bir örneğine özgü verileri depolamak veya almak için herhangi bir gereksinim olmaksızın matematiksel işlemleri gerçekleştiren çeşitli yöntemler içerir. Aşağıda kod bloğunda da görüleceği üzere, sınıf adını ve yöntem adını belirterek sınıfın üyelerini uygulayabilirsiniz. Program çalıştırıldığında sırasıyla ekrana -3 ve -2 değerleri basılır.

```
using System;
public class Program
{
    public static void Main()
    {
        double notOrtalamam = -2.22;
        Console.WriteLine(Math.Floor(notOrtalamam));
        Console.WriteLine(Math.Round(notOrtalamam));
    }
}
```

Tüm sınıf türlerinde olduğu gibi, statik bir sınıfın tür bilgisi, sınıfa başvuran program yüklendiğinde .NET Framework ortak dil çalışma zamanı (*Common Language Runtime - CLR*) tarafından yüklenir. Program, sınıfın tam olarak ne zaman yüklendiğini belirleyemez. Ancak, programınızda sınıfa ilk kez başvurulmadan önce yüklenmesi ve alanlarının başlatılması ve statik oluşturucusunun çağrılması garanti edilir.

Statik bir kurucu yalnızca bir kez çağrılır ve statik bir sınıf, programınızın bulunduğu uygulama etki alanının ömrü boyunca bellekte kalır.

Aşağıdaki listede, statik bir sınıfın ana özellikleri ifade edilmektedir:

- Statik bir sınıf yalnızca statik üye(ler) içerir.
- Örneklenemez, yani statik sınıfın bir örneği oluşturulamaz.
- Mühürlüdür ve mühürleme, sınıfların ve arayüzlerin izin verilen alt türleri üzerinde daha fazla kontrole sahip olmasını sağlar. Bu, hem genel etki alanı modellemesi hem de daha güvenli platform kitaplıkları oluşturmak için kullanışlıdır.
- Statik bir sınıf örnek oluşturucuları *(Instance Constructors)* içeremez.

Statik bir sınıf oluşturmak, temelde yalnızca statik üyeler ve özel bir kurucu içeren bir sınıf oluşturmakla aynıdır. *Private Constructor* bir kurucu, yani özel erişime sahip bir kurucu, sınıfın örneğinin oluşturulmasını engeller. Statik bir sınıf kullanmanın avantajı, derleyicinin hiçbir örnek üyesinin yanlışlıkla eklenmediğinden emin olabilmesidir. Derleyici, bu sınıfın örneklerinin oluşturulamayacağını garanti eder.

HesapMakinesi sınıfı bir statiktir. Bunun tüm üyeleri de statiktir.

Bir sınıfı statik yapmak için statik değiştiriciyi sınıf adından önce ve erişim değiştiricisinden sonra uygulanabilir.

Statik sınıflar mühürlenir ve bu nedenle miras alınamaz. Nesne dışında herhangi bir sınıftan miras alamazlar. Statik sınıflar, bir örnek oluşturucu içeremez; ancak, statik bir kurucu (*static constructor*) içerebilirler.

Statik olmayan sınıflar, sınıf önemsiz olmayan başlatma gerektiren statik üyeler içeriyorsa, statik bir kurucu da tanımlamalıdır.

Bir sınıfı statik yapmak için statik değiştiriciyi sınıf adından önce ve erişim değiştiricisinden sonra uygulanabilir. Aşağıdaki, statik alanlar ve yöntemlerle statik bir sınıfı tanımlanmaktadır.

```
public static class HesapMakinesi
{
    private static int _sonucSaklanan = 0;
    public static string Tipi = "Aritmetik";
    public static int Topla(int sayi1, int sayi2)
    {
        return sayi1 + sayi2;
    }
    public static void Sakla(int sonuc)
    {
        _sonucSaklanan = sonuc;
    }
}
```

Yukarıda, *HesapMakinesi* sınıfı bir statiktir. Bunun tüm üyeleri de statiktir.

Statik sınıftan bir nesne oluşturulamaz. Bu nedenle statik sınıfın üyelerine, aşağıda gösterildiği gibi *SinifAdi.SinifUyesininAdi*, bir sınıf adı kullanılarak doğrudan erişilebilir.

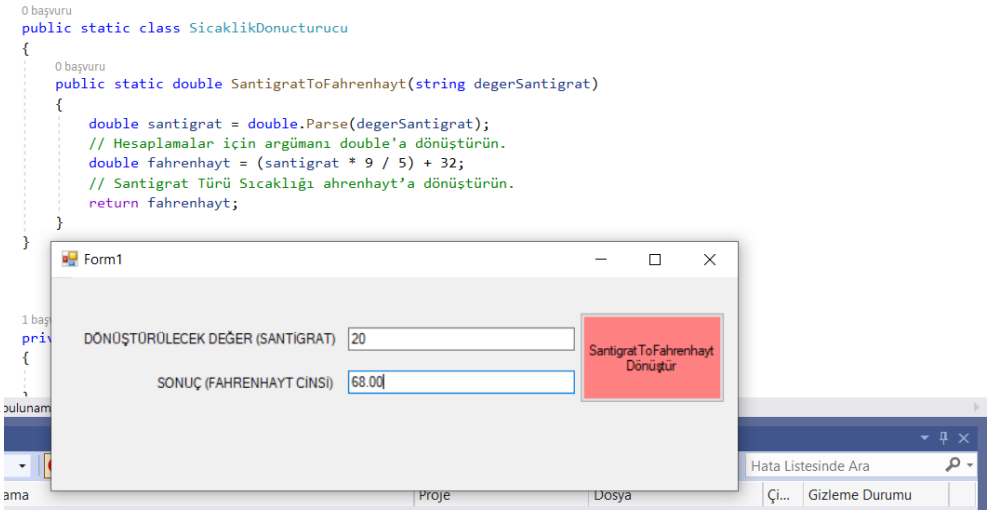
```
class Program
{
    static void Main(string[] args)
    {
        var sonuc = HesapMakinesi.Topla(10, 25);
       // statik yöntem çağırma
        HesapMakinesi.Sakla(sonuc);

        var hesapTuru = HesapMakinesi.Turu;
       // statik değişkene erişim
        HesapMakinesi.Turu = "Bilimsel";
       // statik değişkene değer atama
    }
}
```

Sıcaklığı santigrat sıcaklık türünden Fahrenhayta dönüştüren yöntem içeren bir statik sınıf örneği aşağıda gösterilmektedir.

```
using System;
public static class SicaklikDonucturucu
{
    public static double SantigratToFahrenhayt(string
degerSantigrat)
    {
        double santigrat = double.Parse(degerSantigrat);
        // Hesaplamalar için argümanı double'a dönüştürün.
        double fahrenhayt = (santigrat * 9 / 5) + 32;
        // Santigrat Türü Sıcaklığı ahrenhayt'a dönüştürün.
        return fahrenhayt;
    }
}
class TestSicaklikDonucturucu
{
   static void Main()
   {
     double FSicaklik = 0;
     Console.Write("Lütfen santigrat cinsinden sıcaklığını girin:");
      FSicaklik =
SicaklikDonucturucu.SantigratToFahrenhayt(Console.ReadLine());
      Console.WriteLine("Fahrenhayt cinsinden sıcaklık: {0:F2}", F);
    }
}
```

İlgili kod bloğu çalıştırıldığında Resim 3.2'deki ekran görüntüsü elde edilmektedir.



Resim 3.2. Bir Statik Sınıf Kod Bloğu Örneği ve Çalıştırıldığında Ekran Görüntüsü

C# statik yerel değişkenleri (yani, yöntem kapsamında belirtilen değişkenleri) desteklemez.

Statik Üyeler

Statik olmayan bir sınıf, statik yöntemler, alanlar, özellikler veya olaylar içerebilir. Statik üye, sınıfın hiçbir örneği oluşturulmamış olsa bile bir sınıfta çağrılabilir. Statik üyeye, örnek adıyla değil, her zaman sınıf adıyla erişilir. Sınıfın kaç örneğinin oluşturulduğuna bakılmaksızın, statik bir üyenin yalnızca bir kopyası vardır. Statik yöntemler ve özellikler, içerme türlerindeki statik olmayan alanlara ve olaylara erişemez ve bir yöntem parametresinde açıkça iletilmediği sürece herhangi bir nesnenin örnek değişkenine erişemezler.

Bir alan static const olarak bildirilemese de, const alanı davranışında esasen statiktir. Türün örneklerine değil, türe aittir. Bu nedenle const alanlarına, statik alanlar için kullanılan SinifAdi.SinifUyesininAdi gösteriminin aynısı kullanılarak erişilebilir.

Tüm bir sınıfı statik olarak bildirmektense, bazı statik üyelerle statik olmayan bir sınıf bildirmek daha tipiktir. Statik alanların iki yaygın kullanımı, somutlaştırılan nesnelerin sayısını tutmak veya tüm örnekler arasında paylaşılması gereken bir değeri depolamaktır. Statik yöntemler aşırı yüklenebilir ancak geçersiz kılınamaz. Çünkü bunlar sınıfa aittir ve sınıfın herhangi bir örneğine ait değildir.

Bir alan *static const* olarak bildirilemese de, *const* alanı davranışında esasen statiktir. Türün örneklerine değil, türe aittir. Bu nedenle *const* alanlarına, statik alanlar için kullanılan *SinifAdi.SinifUyesininAdi* gösteriminin aynısı kullanılarak erişilebilir. Nesne örneği gerekli değildir. Ayrıca C#, statik yerel değişkenleri (yöntem kapsamında bildirilen değişkenler) desteklemez.

Aşağıdaki örnekte gösterildiği gibi, üyenin dönüş türünden önce *static* anahtar sözcüğünü kullanarak statik sınıf üyelerini bildirebilirsiniz.

```
using System;

public class Otomobil
{
    public static int TekerSayisi = 4;
    public static int YakitTankiBuyuklugu
    {
        get
        {
            return 15;
        }
    }
    public static void Calistir() { }
    public static event OlayTuru BenzinBitti;
    // Diğer statik olmayan alanlar ve özellikler…
}
```

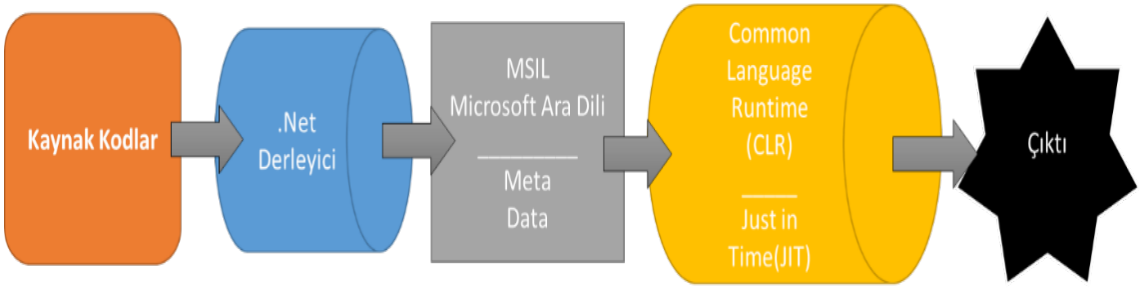
Statik olmayan sınıftaki statik üyeler; normal sınıf (statik olmayan sınıf), bir veya daha fazla statik yöntem, alan, özellik, olay ve diğer statik olmayan üyeler içerebilir.

Statik üyeler, statik üyeye ilk kez erişilmeden ve varsa statik kurucu çağrılmadan önce başlatılır. Statik bir sınıf üyesine erişmek için, aşağıdaki örnekte gösterildiği gibi üyenin konumunu belirtmek için değişken adı yerine sınıfın adını kullanılır.

Örnek

- Otomobil.Calistir();
- int i = Otomobil.TekerSayisi;

Sınıfınız statik alanlar içeriyorsa, sınıf yüklendiğinde bunları başlatan bir statik oluşturucu hazırlamanız gerekir. Statik bir yönteme yapılan bir çağrı, Microsoft ara dilinde (MSIL- Microsoft Intermediate Language) bir çağrı talimatı oluştururken, bir örnek yöntemine yapılan bir çağrı, bir boş nesne referanslarını da kontrol eden bir *callvirt* talimatı oluşturur. Ancak, çoğu zaman ikisi arasındaki performans farkı önemli değildir. Microsoft .NET'i çıkarmasının amacı Java'ya rakip olabilmektir. Bunda da MSIL çok önemli yer tutmaktadır. Genel kod yazım ve bu kodun derlenmesi ve bir çıktı üretmesi aşağıda şekil üzerinde gösterilmektedir. Microsoft Intermediate Language denilen kavram ise bu düşüncenin biraz daha geliştirilmiş halidir. Aşağıda **Şekil 3.1** üzerinde yazılan kodların çalıştırılma süreci aktarılmaktadır.



Şekil 3.1. .Net Ortamında Yazılan Kodların Çalıştırılma Süreci

Statik olmayan sınıftaki statik üyeler; normal sınıf (statik olmayan sınıf), bir veya daha fazla statik yöntem, alan, özellik, olay ve diğer statik olmayan üyeler içerebilir. Statik olmayan bir sınıfı bazı statik üyelerle tanımlamak, bütün bir sınıfı statik olarak bildirmekten daha pratiktir.

Statik Alanlar

Statik olmayan bir sınıftaki statik alanlar, *static* anahtar sözcüğü kullanılarak tanımlanabilir.

Statik olmayan bir sınıfın statik alanları tüm örnekler arasında paylaşılır. Böylece, bir örnek tarafından yapılan değişiklikler diğerlerine de yansıyacaktır.

```
using System;

public class SaymayiDurdur
{
    public static int OrnekSayimi = 0;
    // bir sınıfın örneğini oluştur
    public SaymayiDurdur()
    {
    }
}
class Program
{
    static void Main(string[] args)
    {
        SaymayiDurdur yeniornek1 = new SaymayiDurdur();
        Console.WriteLine(SaymayiDurdur.OrnekSayimi); //1
        SaymayiDurdur yeniornek2 = new SaymayiDurdur();
        SaymayiDurdur yeniornek3 = new SaymayiDurdur();
        SaymayiDurdur yeniornek4 = new SaymayiDurdur();
        SaymayiDurdur yeniornek5 = new SaymayiDurdur();
        SaymayiDurdur yeniornek6 = new SaymayiDurdur();
        SaymayiDurdur yeniornek7 = new SaymayiDurdur();
        SaymayiDurdur yeniornek8 = new SaymayiDurdur();
        Console.WriteLine(SaymayiDurdur.OrnekSayimi);//8
    }
}
```

Statik olmayan bir sınıfta bir veya daha fazla statik yöntem tanımlayabilirsiniz.

Statik olmayan bir sınıftaki statik alanlar, static anahtar sözcüğü kullanılarak tanımlanabilir.

Statik Metotlar

Statik olmayan bir sınıfta bir veya daha fazla statik yöntem tanımlayabilirsiniz. Statik yöntemler, bir nesne oluşturmadan çağrılabilir. Statik olmayan sınıfın bir nesnesini kullanarak statik yöntemleri çağıramazsınız.

Statik yöntemler yalnızca diğer statik yöntemleri çağırabilir ve statik üyelere erişebilir. Statik yöntemlerde sınıfın statik olmayan üyelerine erişemezsiniz.

```
using System;

class Program
{
    static int sayac = 0;
    string name = "Test Statik Method";

    static void Main(string[] args)
    {
        sayac++;
        // statik alanlara erişebilir
        Display("Selam C#!");
        // statik yöntemleri çağırabilir
        name = "Yeni Test Programım";
        // Hata: statik olmayan üyelere erişilemiyor
        SetRootFolder("C:\MyProgram");
        // Hata: statik olmayan yöntem çağrılamaz
    }

    static void Display(string text)
    {
        Console.WriteLine(text);
    }

    public void SetRootFolder(string path) { }
}
```

Statik yöntemler yalnızca diğer statik yöntemleri çağırabilir ve statik üyelere erişebilir.

Statik metotlar için kurallar şu şekilde ifade edilebilir:

- Statik yöntemler, bir dönüş türünden önce ve bir erişim değiştiricisinden sonra static anahtar sözcüğü kullanılarak tanımlanabilir.
- Statik yöntemler aşırı yüklenebilir ancak *override* yani üstüne yazılarak geçersiz kılma yapılamaz.
- Statik yöntemler yerel statik değişkenler içerebilir.
- Statik yöntemler, açıkça parametre olarak iletilmediği sürece statik olmayan değişkenlere erişemez veya bunları çağıramaz.

Statik olmayan bir sınıf, parametresiz bir statik kurucu içerebilir.

Statik Yapıcılar

Statik olmayan bir sınıf, parametresiz bir statik kurucu içerebilir. Statik anahtar kelime ile ve *public*, *private* ve *protected* gibi erişim değiştiricileri olmadan tanımlanabilir. Aşağıdaki örnek, statik oluşturucu ve örnek oluşturucu arasındaki farkı örnek kodlar üzerinden görülebilir.

Statik olmayan bir sınıf, parametresiz bir statik kurucu içerebilir. Parametreli statik oluşturuculara izin verilmez.

```
using System;
public class SaymayiDurdur
{
    static SaymayiDurdur()
    {    // statik yapıcılar
        Console.WriteLine("Statik yapıcı çağrıldı. ");
    }
    public SaymayiDurdur()
    {// örnek oluşturucu
        Console.WriteLine("Örnek oluşturucu çağrıldı.");
    }
    public static void CagirmaBilgisi()
    { // statik methot
        Console.WriteLine("Fonksiyon çağrıldı.");
    }
    public void Basla() { }      // örnek yöntem
    public void Durdur() { }     // örnek yöntem
}
```

Statik anahtar kelime ile ve public, private ve protected gibi erişim değiştiricileri olmadan tanımlanabilir.

Yukarıda, statik olmayan *SaymayiDurdur* sınıfı, statik bir kurucu ve ayrıca statik olmayan bir kurucu içerir. Statik yapıcı, statik yöntem kullanıldığında veya ilk kez bir örnek oluşturulurken yalnızca bir kez çağrılır. Aşağıdaki örnek, statik yöntem ilk kez çağrıldığında statik oluşturucunun çağrıldığını gösterir.

```
using System;
public class SaymayiDurdur
{
    static SaymayiDurdur()
    {    // statik yapıcılar
        Console.WriteLine("Statik yapıcı çağrıldı. ");
    }
    public SaymayiDurdur()
    {// örnek oluşturucu
        Console.WriteLine("Örnek oluşturucu çağrıldı.");
    }
    public static void CagirmaBilgisi()
    { // statik methot
        Console.WriteLine("Fonksiyon çağrıldı.");
    }
    public void Basla() { }    // örnek yöntem
    public void Durdur() { }    // örnek yöntem
}
public class Program
{
      public static void Main()
      {
          SaymayiDurdur.CagirmaBilgisi();  // statik yapıcı çağrıldı
          SaymayiDurdur.CagirmaBilgisi(); // yapıcı çağrılmadı
      }
}
```

Statik yapıcı, statik yöntem kullanıldığında veya ilk kez bir örnek oluşturulurken yalnızca bir kez çağrılır. Statik yöntemi ikinci kez çağırmak, statik bir kurucu çağırmaz.

Statik Yapıcı için kurallar şu şekilde ifade edilebilir; statik oluşturucu, *static* anahtar sözcüğü kullanılarak ve genel (*public*), özel (*private*) veya korumalı (*protected*) erişim değiştiricileri kullanılmadan tanımlanır.

- Statik olmayan bir sınıf, parametresiz bir statik kurucu içerebilir. Parametreli statik oluşturuculara izin verilmez.
- Statik kurucu, kullanım ömrü boyunca yalnızca bir kez yürütülecektir. Bu nedenle, bir sınıf birden çok yerde kullanılıyorsa, bir uygulamada ne zaman çağrılacağını belirleyemezsiniz.

- Statik bir kurucu yalnızca statik üyelere erişebilir. Sınıfın oluşturulan yeni örnek üyelerine erişilemez.

Örnek

•Statik üyeler, bellekte ***High-Frequency Heap*** adı verilen özel bir alanda depolanır. High-Frequency Heap Yüksek frekanslı yığın olarak Türkçeye tercüme edilebilir. Türlerin yöntem tablosu gibi yaygın olarak kullanılan dâhili veri yapılarını depolamak için kullanılır. Statik olmayan sınıfların statik üyeleri, sınıfın tüm örnekleri arasında paylaşılır. Böylece, bir örnek tarafından yapılan değişiklikler diğer tüm örneklere yansıtılacaktır.

Kısmı sınıf kullanım avantajı:
Büyük projeler üzerinde çalışırken, bir sınıfı ayrı dosyalara yaymak, birden fazla programcının aynı anda üzerinde çalışmasına olanak tanımaktadır.

Statik kurucu, kullanım ömrü boyunca yalnızca bir kez yürütülecektir. Bu nedenle, bir sınıf birden çok yerde kullanılıyorsa, bir uygulamada ne zaman çağrılacağını belirleyemezsiniz.

Kısmı sınıf, tanımı iki veya daha fazla dosyada bulunan bir sınıftır. Her kaynak dosya, sınıfın bir bölümünü içerir ve uygulama derlendiğinde tüm parçalar birleştirilir.

KISMİ SINIFLAR (PARTIAL CLASSES)

Kısmi sınıf (partial class), tanımı iki veya daha fazla dosyada bulunan bir sınıftır. Her kaynak dosya, sınıfın bir bölümünü içerir ve uygulama derlendiğinde tüm parçalar birleştirilir. Parçalara ayrılan sınıfı (*class*) derleme aşamasından tek bir sınıf (*class*) haline getirme işi derleme sırasında otomatik olarak yapılmaktadır. Partial yani kısmi sınıf kullanımı ile bizim ekstra bir şey gerçekleştirmeden bu işi yapmamıza imkan tanımaktadır. Şimdi asıl *partial* anahtar kelimesinin kullanım sebebine gelelim belirttiğimiz gibi *metod*, özellik, *event* gibi nesneleriniz normalde aynı sınıf (*class)* içerisinde yer alıyorsa bir yerden sonra karışıklıklara sebep oluyorsa yapmanız gereken *partial* anahtar kelimesini kullanmak ve parçalara ayırmaktır.

Aşağıda kod örneğinde örnek bir kullanımı görebilirsiniz. Özellik *(Property*) ve metotlarımızı ayrı sınıflar ile fakat aynı isimdeki sınıflarda bu kod bloğu aracılığı ile tutabiliriz.

```
using System;
namespace PartialClass
{
      public partial class Bisiklet
      {// sınıfın property dediğimiz özellikleri, değişkenleri...

          public int Numara { get; set; }
          public int JantSayisi { get; set; }
          public string Turu { get; set; }
      }
      public partial class Bisiklet
      {//sınıfa ait türler ki bu metotlar yukarıda olabilirdi.
          public void PedalCevir()
          {
          }
          public void FreneBas()
          {
          }
      }
}
```

C#'da kısmi sınıfları kullanmanın en önemli avantajları arasında; büyük projeler üzerinde çalışırken, bir sınıfı ayrı dosyalara yaymak, birden fazla

programcının aynı anda üzerinde çalışmasına olanak tanıması şeklinde ifade edilebilir.

Otomatik olarak oluşturulan bir kaynakla çalışırken, kaynak dosyayı yeniden oluşturmaya gerek kalmadan kod sınıfa eklenebilir. Bu çok önemli bir avantajdır. Visual Studio, Windows Forms, Web hizmeti sarmalayıcı kodu vb. oluştururken bu yaklaşımı kullanır. Visual Studio tarafından oluşturulan dosyayı değiştirmek zorunda kalmadan bu sınıfları kullanan kod oluşturabilmektedir.

Tüm bunlara ek olarak, kısmi sınıf oluşturduğumuz gibi kısmi yapılar, arayüzler ve yöntemler oluşturabilir. Fakat kısmi temsilciler ve numaralandırma oluşturulamaz. Kısmi bir sınıfın farklı bölümleri farklı arabirimlerden miras alabilir. İç içe sınıflar (*containing class*), içeren sınıf kısmi olmasa bile kısmi sınıflar olarak belirtilebilir.

Bir sınıf tanımını bölmek için aşağıdaki örnekte gösterildiği gibi kısmi anahtar sözcüğü olan *partial* (public partial class Ogrenci örneğin) kullanılmaktadır. Öğrenci sınıfı iki bölüme ayrılmıştır. İlk kısım, *Calis()* yöntemini, ikinci kısım ise *Oyna()* yöntemini tanımlar. Bu programı derlediğimizde her iki parça da birleştirilecek ve derlenecektir.

Tüm parçalar partial anahtar kelimeyi kullanmalıdır ve Final Class diye adlandırabileceğimiz son sınıfı oluşturmak için tüm parçalar derleme zamanında mevcut olmalıdır.

Partial - kısmi sınıflar oluştururken aşağıdaki noktaları akılda tutmak çok önemlidir.

- Tüm parçalar *partial* anahtar kelimeyi kullanmalıdır.
- Final Class diye adlandırabileceğimiz son sınıf, derleme zamanında tüm parçaların birleşimidir.
- Final Class diye adlandırabileceğimiz son sınıfı oluşturmak için tüm parçalar derleme zamanında mevcut olmalıdır.
- Partial - kısmi bir tanımda bildirilen herhangi bir sınıf üyesi, diğer tüm parçalar tarafından kullanılabilir.
- Tüm parçalar aynı erişim değiştiricilerine sahip olmalıdır - genel *(public)*, özel *(private)*, korumalı *(protected)* vb.

Her iki parçanın da kısmi anahtar kelime ve genel erişim değiştirici kullandığını unutmamak gerekir.

Oluşturulan kodu özelleştirmek için kısmi yöntemler (partial methods) kullanılabilir. Bir yöntem adının ve imzanın ayrılmasına izin verirler, böylece oluşturulan kod yöntemi çağırabilir.

```
using System;
namespace PartialClass
{
    public partial class Ogrenci
    {
        public void Calis()
        {
            Console.WriteLine("Ders Calışıyorum");
        }
    }
    public partial class Ogrenci
    {
        public void Oyna()
        {
            Console.WriteLine("Oyun oynuyorum.");
        }
    }
    public class Test
    {
        public static void Main()
        {
            Ogrenci OgrenciNesnesi = new Ogrenci();
            OgrenciNesnesi.Calis();
            OgrenciNesnesi.Oyna();
        }
    }
}
```

Kısmi bir metot oluşturmak için, kısmi sınıfın bir kısmında metodun bildirimini ve kısmi sınıfın diğer kısmında uygulamayı oluştururuz. Uygulama tamamen isteğe bağlıdır.

Uygulama sağlanmazsa, derleme zamanında yöntem ve yönteme yapılan tüm çağrılar kaldırılır. Bu nedenle, kısmi sınıftaki herhangi bir kod, uygulama sağlanmasa bile kısmi bir yöntemi serbestçe kullanabilir. Yöntem çağrıldığında ancak uygulanmadığında, derleme zamanı veya çalışma zamanı hatası oluşmaz.

Özetle, kısmi bir yöntem bildirimi, tanım ve uygulama olmak üzere iki bölümden oluşur. Bunlar kısmi bir sınıfın ayrı bölümlerinde veya aynı bölümde olabilir. Uygulama bildirimi yoksa, derleyici hem tanımlayıcı bildirimi hem de yönteme yapılan tüm çağrıları optimize eder.

Kısmi bir metot oluşturmak için, kısmi sınıfın bir kısmında metodun bildirimini ve kısmi sınıfın diğer kısmında uygulamayı oluştururuz. Uygulama tamamen isteğe bağlıdır.

Kısmi yöntemler oluştururken akılda tutulması gereken noktalar şu şekildedir:

- *Partial/Kısmi* yöntemler *extern/harici* olamaz.
- Kısmi yöntem bildirimleri, kısmi anahtar sözcükle başlamalıdır.
- Kısmi bir yöntemin dönüş türü *void* olmalıdır.
- Kısmi yöntemler *ref* parametresine sahip olabilir ancak çıkış parametreleri olmayabilir.
- Kısmi yöntemler dolaylı olarak *private'tir* ve bu nedenle *sanal-virtual* olamazlar.
- Kısmi yöntemler *extern* olamaz, çünkü gövdenin varlığı, bunların tanımlanıp tanımlanmayacağını veya uygulanıp uygulanmayacağını belirler.

Oluşturulan kodu özelleştirmek için kısmi yöntemler (*partial methods*) kullanılabilir. Bir yöntem adının ve imzanın ayrılmasına izin verirler, böylece oluşturulan kod yöntemi çağırabilir. Ancak geliştirici yöntemi uygulayıp uygulamamaya karar verebilir. Kısmi sınıflara (*partial class*) çok benzer şekilde, kısmi yöntemler, bir kod oluşturucu tarafından oluşturulan kodun ve bir insan tarafından geliştirilen kodun, çalışma zamanı maliyetleri olmadan birlikte çalışmasını sağlamaktadır.

İç içe sınıfların mantıksal gruplanması, daha iyi kapsülleme ve kod okunabilirliği yönünden üç temel faydası vardır.

İÇ İÇE SINIFLAR (NESTED CLASSES)

Nesneye yönelik programlama dilleri kullanıcıya bir sınıf içerisinde başka bir sınıf tanımlama olanağı sağlar. Bu tür sınıflara *nested class* (*gömülü sınıf* yani *iç içe sınıf*) adı verilir. Gömülü sınıflar statik ve statik olmayan gömülü sınıflar olmak üzere ikiye ayrılırlar. Statik olmayan gömülü sınıflara *Inner Class*(*iç sınıf*) adı verilir. İç içe sınıfların mantıksal gruplanması, daha iyi kapsülleme ve kod okunabilirliği yönünden üç temel faydası vardır.

İç içe türler, devralınan özel (private) veya korumalı (protected) üyeler dahil, içeren türün özel (private) veya korumalı (protected) üyelerine erişebilir.

İç içe sınıf (*nested class*) veya iç sınıf (*inner class*), aşağıdaki örnekte gösterildiği gibi içeren veya dış sınıfa erişebilir. İç içe türler, devralınan özel (*private*) veya korumalı (*protected*) üyeler dahil, içeren türün özel (*private*) veya korumalı (*protected*) üyelerine erişebilir. Aşağıda örnek kod bloğu ile iç içe sınıf örneği gösterilmektedir.

```
public class Container
{
    class Nested
    {
        Nested() { }
    }
}
```

Aşağıdaki diğer örneğimizde araba sınıfı içinde protected (korunan) ve public (genel) türünde sınıflar kullanılarak konunun daha iyi anlaşılması hedeflenmiştir.

```
public class Araba
    {
        protected Kapi OnSag;
        protected Kapi OnSol;
        protected Kapi ArkaSag;
        protected Kapi ArkaSol;
        protected class Motor
        {
            public int beygirGuc;

        }
        protected class Batarya
        {
            public int voltaj;
        }
        protected Motor Motorum;
        protected Batarya Bataryam;
        public Araba()
        {
            Motorum = new Motor();
            Bataryam = new Batarya();
        }
    }
    public class Kapi
    {
        public int pozisyon;
    }
    public class togg : Araba
    {
        public togg ()
        {
            Motorum.beygirGuc = 1500;
            Bataryam.voltaj = 123;
            OnSol.pozisyon = 2;
            OnSag.pozisyon = 1;
        }
    }
```

Başka bir sınıf veya yapı içinde tanımlanan bir türe (sınıf veya yapı), iç içe tür (*nested type*) denir. Aşağıda bir örnek gösterilmiştir. İç Sınıf, Konteyner Sınıfının (*container class*), içindedir. Dolayısıyla iç sınıfa (*inner class*) iç içe sınıf (*nested class*) denir.

Başka bir sınıf veya yapı içinde tanımlanan bir türe (sınıf veya yapı), iç içe tür (nested type) denir.

```
using System;
namespace IcIceSinifOrnegi
    {
        class KonteynarSinif
        {
            class IcSinif
            {
                public string str =" İç içe sınıfta bir dize değişkeni";
                public static void main()
                {
                    InnerClass IcIceSınıf.Nesnesi = new IcIceSinif();
                    Console.WriteLine(IcIceSınıf.Nesnesi.str);
                }
            }
        }
    }
```

Bir sınıf çağırılmak istenildiğinde isim alanı belirtilmek zorundadır. Aşağıda bunu örnek bir kod üzerinde ifade edilmektedir.

```
using System;
namespace MatFonk
{
    class Toplama
    {
        public Toplama(){ }
    }
    namespace MatFonk2
    {
        class Carpma
        {
            public Carpma() { }
        }
    }
}
class AnaSinif
{
    public static void Main()
    {
        MatFonk.Toplama toplar = new Mat.Toplam();
        MatFonk.MatFonk2.Carpma carpar = new MatFonk.MatFonk2.Carpma();
    }
}
```

Bireysel Etkinlik

- TestSinifi adında farklı dosyalarda oluşturduğunuz iki farklı sınıfın partial class ile işaretlenerek compile time'da birleştirilmesini sağlayanız. Bu şekilde iki ayrı dosyada tanımladığınız, yöntemlerin tek sınıfın üyeleriymiş gibi çalıştırılması için bir etkinlik gerçekleştirmeyi deneyiniz.

Özet

- Nesneye yönelimli programlama için sınıf yapısı bir önceki bölümde de detaylı olarak ortaya konulduğu gibi kritik önemdedir. Özellikle belleği etkin kullanmak, program içinde her şeyin sınıflar, sınıflara ait nesneler ve metotlar üzerinden yönlendirilmesi tüm bu süreç için bellek ortamından etkin bir şekilde faydalanılması nesne yönelimli programlamanın en önemli avantajı ve karakteristik özelliğidir.
- Sınıf veri üyeleri (sabitler ve alanlar), işlev üyeleri (Yöntemler, özellikler, olaylar, dizin oluşturucular, işleçler, örnek oluşturucular, yıkıcılar ve statik oluşturucular) ve iç içe türler içerebilen bir veri yapısıdır. Sınıf türleri, türetilmiş bir sınıfın bir temel sınıfı genişletebileceği ve özelleştirileceği bir mekanizma olan devralmayı destekler.
- C#'da beş farklı erişim değiştirici mevcuttur. Bunlar sırasıyla kamuya yani herkese açık kullanım (public), korumalı kullanım (protected), dahili kullanım (internal), korumalı dahili kullanım (protected internal), özel kullanım (private), şeklindedir.
- Derleme birimlerinde veya ad alanlarında belirtilen türler olabilir public veya internal erişebilir. Varsayılan olarak internal erişim'dir. Sınıflarda belirtilen türlerin, public protected, internal protected, internal veya private erişimi olabilir. Varsayılan olarak private erişim' dir. Yapılarda belirtilen türlerin public, internal veya private erişimi olabilir. Varsayılan olarak private erişim'dir. Bir sınıfı deklare etmek (class_declaration) , isteğe bağlı olarak bir sınıf değiştiricileri dizisi içerebilir. Aynı değiştiricinin bir sınıf bildiriminde birden çok kez görünmesi için derleme zamanı hatası vardır. Bu değiştiriciler şu şekilde ifade edilebilir: : 'new' | 'public'| 'protected'| 'internal' | 'private' | 'abstract' | 'sealed'| 'static'| class_modifier_unsafe.
- Konu kapsamında statik sınıflar ve sınıf üyeleri, kısmi sınıflar (partial classes) ve iç içe sınıflar (nested classes) açıklanmaktadır.
- **SOYUT SINIFLAR (ABSTRACT CLASS)**
- Veri soyutlama, belirli ayrıntıları gizleme ve kullanıcıya yalnızca temel bilgileri gösterme işlemidir. Soyutlama, soyut sınıflar veya arayüzler ile gerçekleştirilebilir (Nesne Yönelimli Programlama II dersinizde detaylandırılacaktır.). abstract değiştirici, bir sınıfın tamamlanmamış olduğunu ve yalnızca temel sınıf olarak kullanılması amaçlanan olduğunu göstermek için kullanılır. Soyut anahtar kelime, sınıflar ve yöntemler için kullanılır.
- *Soyut sınıf*: Nesneler oluşturmak için kullanılamayan kısıtlı bir sınıftır (ona erişmek için başka bir sınıftan miras alınması gerekir).
- *Soyut yöntem*: Yalnızca soyut bir sınıfta kullanılabilir ve bir gövdesi yoktur. Gövde, türetilmiş sınıf tarafından sağlanır (kalıtsal).
- Soyut bir sınıf doğrudan başlatılamaz ve bir derleme zamanı hatası, new işleci bir soyut sınıfta kullanılır.
- **MÜHÜRLÜ SINIFLAR (SEALED CLASS)**
- Nesneye yönelimli programlamanın en önemli özelliklerinden birisi olan bir sınıftan başka sınıflar üretebilme durumu, güvenlik gibi çeşitli sebepler ile istenmeyebilir. Bu sınıflardan türemenin istenmemesi durumunda sınıfın başına mühürlendiğini yani bu sınıftan türetme yapılamayacağını gösteren sealed anahtar sözcüğü kullanılır.
- Bu sayede de sabit özellikler ve metotlara sahip olan sınıflar elde edilebilir. Bu duruma uyulmadığında yani bir mühürlü sınıf, başka bir sınıfın temel sınıfı olarak belirtilmişse, derleme zamanı hatası ile karşılaşılmaktadır. Ayrıca mühürlü bir sınıf de soyut bir sınıf olamaz.
- **STATİK SINIFLAR VE ÜYELERİ**
- C#'da statik, somutlaştırılamayan bir şey anlamına gelir. Statik bir sınıfın nesnesini oluşturamazsınız ve bir nesne kullanarak statik üyelere erişemezsiniz.

Özet (devamı)

- C# sınıfları, değişkenler, metotlar, özellikler, operatörler, olaylar ve yapıcılar, static değiştirici anahtar sözcüğü kullanılarak statik olarak tanımlanabilir.
- Statik bir sınıf, temelde statik olmayan bir sınıfla aynıdır, ancak bir fark ile; statik bir sınıf somutlaştırılamaz. Başka bir deyişle, sınıf türünde bir değişken oluşturmak için new anahtar sözcüğünü kullanamazsınız. Statik bir sınıf, sealed veya abstract değiştiricisini içeremez. Ancak, statik bir sınıf tarafından örneklenemez veya türetilmediği için, hem korumalı hem de soyut gibi davranır. Statik bir kurucu yalnızca bir kez çağrılır ve statik bir sınıf, programınızın bulunduğu uygulama etki alanının ömrü boyunca bellekte kalır.
- Aşağıdaki listede, statik bir sınıfın ana özellikleri ifade edilmektedir:
  - Statik bir sınıf yalnızca statik üye(ler) içerir.
  - Örneklenemez, yani statik sınıfın bir örneği oluşturulamaz.
  - Mühürlüdür ve mühürleme, sınıfların ve arayüzlerin izin verilen alt türleri üzerinde daha fazla kontrole sahip olmasını sağlar. Bu, hem genel etki alanı modellemesi hem de daha güvenli platform kitaplıkları oluşturmak için kullanışlıdır.
  - Statik bir sınıf örnek oluşturucuları (Instance Constructors) içeremez.
- Statik sınıflar mühürlenir ve bu nedenle miras alınamaz. Nesne dışında herhangi bir sınıftan miras alamazlar. Statik sınıflar, bir örnek oluşturucu içeremez; ancak, statik bir kurucu (static constructor) içerebilirler.
- Statik Üyeler; statik olmayan bir sınıf, statik yöntemler, alanlar, özellikler veya olaylar içerebilir. Sınıfın hiçbir örneği oluşturulmamış olsa bile bir sınıfta çağrılabilir. Statik üyeler, statik üyeye ilk kez erişilmeden ve varsa statik kurucu çağrılmadan önce başlatılır.
- *Statik Alanlar;* statik olmayan bir sınıftaki statik alanlar, static anahtar sözcüğü kullanılarak tanımlanabilir. Statik olmayan bir sınıfın statik alanları tüm örnekler arasında paylaşılır.
- *Statik Metotlar;* statik olmayan bir sınıfta bir veya daha fazla statik yöntem tanımlayabilirsiniz ve bir nesne oluşturmadan çağrılabilir. Statik yöntemler yalnızca diğer statik yöntemleri çağırabilir ve statik üyelere erişebilir.
- *Statik Yapıcılar*; statik olmayan bir sınıf, parametresiz bir statik kurucu içerebilir. Statik anahtar kelime ile ve public, private ve protected gibi erişim değiştiricileri olmadan tanımlanabilir. Aşağıdaki örnek, statik oluşturucu ve örnek oluşturucu arasındaki farkı örnek kodlar üzerinden görülebilir.
- **KISMİ SINIF (PARTIAL CLASS)**
- Kısmi sınıf (partial class), tanımı iki veya daha fazla dosyada bulunan bir sınıftır. Her kaynak dosya, sınıfın bir bölümünü içerir ve uygulama derlendiğinde tüm parçalar birleştirilir. C#'da kısmi sınıfları kullanmanın en önemli avantajları arasında; büyük projeler üzerinde çalışırken, bir sınıfı ayrı dosyalara yaymak, birden fazla programcının aynı anda üzerinde çalışmasına olanak tanıması şeklinde ifade edilebilir.
- **İÇ İÇE SINIFLAR (NESTED CLASS)**
- Nesneye yönelik programlama dilleri kullanıcıya bir sınıf içerisinde başka bir sınıf tanımlama olanağı sağlar. Bu tür sınıflara Nested Class (gömülü sınıf yani iç içe sınıf) adı verilir. Gömülü sınıflar statik ve statik olmayan gömülü sınıflar olmak üzere ikiye ayrılırlar. Statik olmayan gömülü sınıflara Inner Class(İç Sınıf) adı verilir. İç içe sınıfların mantıksal gruplanması, daha iyi kapsülleme ve kod okunabilirliği yönünden üç temel faydası vardır.

DEĞERLENDİRME SORULARI

1. Soyut Sınıf (Abstract class) ile ilgili aşağıdakilerden hangisi yanlıştır?
   a) Soyut bir sınıf türetirsek ve tüm soyut yöntemleri uygulamazsak, türetilmiş sınıf da 'abstract' anahtar sözcüğü kullanılarak soyut olarak işaretlenmelidir.
   b) Soyut sınıfların kurucuları olabilir
   c) Bir sınıf, herhangi bir soyut yöntem olmadan soyut hale getirilebilir
   d) Bir sınıf birden çok soyut sınıftan miras alabilir
   e) Soyut sınıflar sayesinde kullanıcılardan gereksiz detaylar gizlenebilmektedir.

2. Aşağıdakilerden hangisi derlerken programcının karşısına hata mesajı verir?

```
class SinifA { }

abstract class SinifB { }

abstract class SinifC { abstract void method(); }
```

   a) SinifA
   b) SinifB, SinifC
   c) SinifA, SinifB
   d) SinifA, SinifB, SinifC
   e) Tüm Sınıflar Derlenir

3. Aşağıdakilerden hangisi derlerken programcının karşısına hata mesajı verir?

```
class SinifA { }

abstract class SinifB { }

class SinifC { abstract void method(); }
```

   a) SinifA
   b) SinifA, SinifB, SinifC
   c) SinifB
   d) SinifC
   e) Tüm Sınıflar Derlenir

Nesneye yönelik programlama dilleri kullanıcıya bir sınıf içerisinde başka bir sınıf tanımlama olanağı sağlar. Bu tür sınıflara …….. adı verilir.

4. Yukarıdaki boşluğa hangisi gelmelidir?
   a) İç İçe Sınıflar (Nested Class)
   b) Mühürlü Sınıf (Sealed Class)
   c) Soyut Sınıflar (Abstract Class)
   d) Statik Sınıflar (Static Class)
   e) Kısmi Sınıflar (Partial Class)

5. Statik sınıf ve üyeleri ile ilgili aşağıdakilerden hangisi yanlıştır?
   a) Statik sınıflar bir örnekle temsil edilemez.
   b) Statik bir sınıfın tüm üyeleri statik olmalıdır; aksi halde derleyici hata verecektir.
   c) Statik bir sınıf, örnek üyeler ve kurucular içeremez.
   d) Statik bir sınıf diğer sınıflardan miras alamaz.
   e) Statik sınıflar ve üyeleri için program çalıştıktan sonra istediğimiz gibi değiştirme hakkımız vardır.

Bir sınıfı deklare etmek, isteğe bağlı olarak bir sınıf değiştiricileri dizisi içerebilir. Aynı değiştiricinin bir sınıf bildiriminde birden çok kez görünmesi için derleme zamanı hatası vardır.

6. Bu değiştiriciler arasında aşağıdakilerden hangisi sayılamaz?
   a) static
   b) sealed
   c) abstract
   d) public
   e) external

Nesneye yönelimli programlamanın en önemli özelliklerinden birisi olan bir sınıftan başka sınıflar üretebilme durumu, güvenlik gibi çeşitli sebepler ile istenmeyebilir. Bu sınıflardan türemenin istenmemesi durumunda sınıfın başına mühürlendiğini yani bu sınıftan türetme yapılamayacağını gösteren …… anahtar sözcüğü kullanılır. Bu sınıflara da ……. denir.

7. Yukarıdaki boşluğa hangisi gelmelidir?
   a) nested, iç içe sınıflar
   b) static, mühürlü sınıflar
   c) abstract, mühürlü sınıflar
   d) sealed, soyut sınıflar
   e) sealed, mühürlü sınıflar

8. Kısmi sınıflar (partial class) ile ilgili aşağıdakilerden hangisi yanlıştır?
   a) İki veya daha fazla dosyada bulunan bir sınıf olarak tanımlanmaktadır.
   b) Her kaynak dosya, sınıfın bir bölümünü içerir ve uygulama derlendiğinde tüm parçalar birleştirilir.
   c) Parçalara ayrılan sınıfı derleme aşamasından tek bir sınıf haline getirme işi derleme sırasında otomatik olarak yapılmaktadır.
   d) Nested anahtar kelimesi ile tanımlanabilir.
   e) C#'da kısmi sınıfları kullanmanın en önemli avantajları arasında; büyük projeler üzerinde çalışırken, bir sınıfı ayrı dosyalara yaymak, birden fazla programcının aynı anda üzerinde çalışmasına olanak tanıması şeklinde ifade edilebilir.

```
using System;
public class SaymayiDurdur
{
    public static int OrnekSayimi = 0;
    // bir sınıfın örneğini oluştur
    public SaymayiDurdur()
    {
    }
}
class Program
{
    static void Main(string[] args)
    {
        SaymayiDurdur yeniornek1 = new SaymayiDurdur();
        Console.WriteLine(SaymayiDurdur.OrnekSayimi);
        SaymayiDurdur yeniornek2 = new SaymayiDurdur();
        SaymayiDurdur yeniornek3 = new SaymayiDurdur();
        SaymayiDurdur yeniornek4 = new SaymayiDurdur();
        Console.WriteLine(SaymayiDurdur.OrnekSayimi);
    }
}
```

9. Yukarıdaki programın çıktısı hangisi?
   a) 0
      0
   b) 1
      5
   c) 1
      2
   d) 1
      4
   e) Derleme hatası ile karşılaşırız

```
using System;
class TestProgramim
{
    static void Main(string[] args)
    {
        int numara = 2;
        testFonk1 (ref numara);
        Console.WriteLine(numara);
        Console.ReadLine();
    }
    static void testFonk1 (ref int numara)
    {
        numara = numara * numara * numara;
    }
}
```

10. Yukarıdaki programın çıktısı hangisi?

    a) Derleme hatası ile karşılaşırız
    b) 0
    c) 2
    d) 8
    e) 16

Cevap Anahtarı

1.d, 2.e, 3.d, 4.a, 5.e, 6.e, 7.e, 8.d, 9.a, 10.d

YARARLANILAN KAYNAKLAR

Alagić, S. (2015). Object-Oriented Technology. London: Springer.

Anggoro, W. (2016). Functional C#. England: Packt Publishing Ltd.

C# Klavuzu (2021). C# Belgeleri. Erişim Tarihi: 27/08/2021, https://docs.microsoft.com/tr-tr/dotnet/csharp/

C# Notes for Professionals (2021). GoalKicker.com, C# Notes for Professionals 105. Erişim Tarihi: 27.08.2021, https://goalkicker.com/CSharpBook

Dathan, B., & Ramnath, S. (2015). Object-Oriented Analysis, Design and Implementation. ABD: Springer.

DotNetTutorials (2021). Partial Class and Nested Types. Erişim Tarihi: 27/08/2021, https://dotnettutorials.net/

Gross, C. (2008). Beginning C# 2008: from novice to professional. New York: Dreamtech Press.

Gunnerson, E., & Wienholt, N. (2012). A Programmer's Guide to C# 5.0. New York: Apress.

İTÜBİDB (2019). Nesne Yönelimli Programlama (Object-Oriented Programming). Seyir Defteri. İTÜ Bilgi İşlem Daire Başkanlığı. Erişim Tarihi: 27/08/2021, https://bidb.itu.edu.tr/seyir-defteri/

Karaçay T. (2021). Erişim Belirteçleri, Bölüm 11. Erişim Tarihi: 27/08/2021, http://mail.baskent.edu.tr/~tkaracay/etudio/ders/prg/java/ch11/accessmods.htm

Keentpoint. (2021). C# - Sealed. Erişim Tarihi: 27/08/2021, https://www.keentpoint.com/tutorial/csharp/Sealed

Küçükkoç, İ. (2020). Algoritma ve Programlamaya Giriş, Ders Notları. Balıkesir Üniversitesi, Mühendislik Fakültesi Endüstri Mühendisliği Bölümü.

Nakov, S., & Kolev, V. (2013). Fundamentals of Computer Programming with C#: The Bulgarian C# Book. Sofia: Faber Publishing.

Nguyen P.(2020). Static Class. Erişim Tarihi: 27/08/2021, https://web.csulb.edu/~pnguyen/cecs475/pdf/note2.pdf

Shivakumar S. (2020). C# Tutorial. Erişim Tarihi: 27/08/2021, https://www.educba.com/partial-in-c-sharp/

TutorialsTeacher. (2021). C# - Partial Classes and Methods. Erişim Tarihi: 27/08/2021, https://www.tutorialsteacher.com/csharp/csharp-partial-class

W3Schools. (2021). C# Abstraction. Erişim Tarihi: 27/08/2021, https://www.w3schools.com/cs/cs_abstract.php