# Giriş

Nesne tabanlı programlama, programlarda sınıflar tanımlamaya imkan veren ve bu sınıflardan yenilerinin türetilmesini sağlayan bir programlama tekniğidir.

Bu teknik yazılım geliştiren insanlara büyük avantajlar sağlamaktadır. Karmaşık projelerin üretmini ve bakımını kolaylaştırır, program kodunun tekrarlı kullanımına olanak sağlar (code reuseability). Bu da maliyeti azaltır, ulaşılabiliriliği artırır.

Bu teknikte esas olan gerçek hayatta var olan olguların programlamaya aktarılmasıdır. Burada iki önemli birim vardır: veri ve veriyi işleyen metodlar.

Mesela ütü sınıfımız olsun. Marka, model, renki çalıştığı elektrik voltajı, ütüleyebildiği kumaş türleri bu ütünün verilerindendir. Aynı zamanda ütü ısınabilir, ütüleme işleminde kullanılır ve soğumaya bırakılabilir. Bunlar ise ütünün metotlarıdır.

# Kapsülleme(Encapsulation)

İşlemlerin ve verilerin gizlenmesine veya bir grup içerisinde toplanmasına kapsülleme denir.

Kapsülleme sayesinde sınıf içerisinde kullanılan alanların dışarıdan herhangi bir etki ile doğrudan değiştirilmelerini engellenmiş olunur. Bu durum istek dışı atamaların önüne geçer ve eğer bu alanların değerleri değişecek ise belirlenen bir çizgide olması sağlanır. Bu sayede program üzerindeki kontrol artar.

Kapsüllemenin bir diğer işlevi ise veri korumadır. Bir sınıf tasarımında doğru olan nesnelerin durum verilerini **private** erişim belirleyicisi ile koruma altına almaktır. Bu yolla dış dünya alanın değerini elde etmek ya da değiştirmek istediğinde bu protokolden geçmek zorunda kalır. Dış dünyaya açık, yani **public** erişim belirleyicisine sahip alanların sorunu, ait oldukları sınıfın iş mantıklarını algılayabilme yeteneklerinin olmamasıdır. Dolayısıyla atanan değerin iş mantığına uygun olup olmadığına dair herhangi bir kontrolün sınıf içerisinde yapılabilmesi mümkün değildir.

Nesne kullanıcısı genelde sınıfı yazan kişi olabilir de olmayabilir de. Dolayısıyla böyle bir kontrolü nesne kullanıcısı tarafında yapmak verimli değildir. Sınıfların durum verileri olarak anılan alanlar, private erişim belirleyicisi alarak dış dünyaya kapatıldıklarında başlangıç değerleri, sınıf içerisinde tanımlandığı yerde ya da yapıcı metot yardımıyla verilebilir. Böyle bir kapsülleme tercih edilebilir ancak bazı durumlarda alanın değerinin dışarıdan okunması, değerinin sadece içsel olarak sınıf üyeleri tarafından atanması, durum verileri üzerinde yapılacak değişikliklerin dışarıya açılması gerekebilir. Burada değişiklik ile birlikte gelen verinin kontrol edilerek istenmeyen durumların önüne geçilmesi gerekliliği ortadadır.

Kapsülleme, durum verilerinin tutarlıulığını sağlamak için bir yol sunmaktadır. Public erişim belirtecine sahip alanlar tanımlamaktansa, alan verilerini private olarak tanımlamak alışkanlık haline getirilmelidir (herhangi bir iş kuralı olamasa da). Bu durumda alanlar nesne kullanıcısına iki teknikle açılabilir.

* Nesne yönelimli diğer diller için geleneksel yol olan Erişen ve Değiştiren (Accessor and Mutator) metotlar tanımlamak.
* Bir özelllik (property) tanımlamak.

Hangi teknik kullanılırsa kullanılsın, iyi kapsüllenmiş bir sınıf ham verisini (alanlarını) ve onu nasıl yönettiğinin ayrıntılarını dış dünyanın kem gözlerinden gizlemelidir. Bu yaklaşımın güzel ve faydalı olan yanı sınıfı yazan kişinin çalışan kodu bozmadan kapalı kapılar ardından metodun ya da özelliğin uygulanışını değiştirmekte serbest olmasıdır.

.NET Framework temel sınıf kütüphanesi, geleneksel Erişen/Değiştiren metotlar yerine tip özelliklerini tercih eder. Bu yüzden sınıf kütüphanesi ile sağlıklı bir şekilde etkileşen programlar için kendi tiplerimizi yazarken özelliklerden faydalanmak doğru bir davranış olacaktır. Ayrıca özelliklerin bir alanın değerini işaret edip alana değer atamaları zorunlu değildir. Bir alan olmadan da özelliklerden faydalanılabilinir.

# Kalıtım (Inheritence)

İnsanlardaki kalıtım gibi pragramlama dillerinde de kalıtım vardır. En basit tanımı ile kalıtım, bir sınıftan yeni sınıflar türetmektir. Kalıtım ile türetilen sınıflar, türetildiği sınıfın özelliklerinin tamamını da devralır. Buradan türetilmiş sınıfın türediği sınıfa ait üyelere erişebileceği sonucu da çıkarılır. Tüm sınıflarda ortak olan özellikleri tek bir sınıf içerisinde toplamak için bir sınıftan başka sınıflar türetilir. Her sınıfın özelliklerinin tekrar yazılması engellenir ve sınıflar arası hiyerarşi ortaya çıkar.

# Çok Biçimlilik (Polymorphism)

Bir metodun farklı nesnelerde veya farklı ortamlarda farklı sonuçlar üretmesidir.

C# dili tam anlamıyla nesne tabanlı bir dildir. Bu dil içerisinde sınıf kavramının önemli bir yeri vardır. Bu kavramı iyi anlamak, her türlü teknikte sınıfların avantajlarından yararlanılması ve kişiye özgü nesnelere sahip olunması sağlanabilir. Zaten .NET teknolojisinde yer alan her nesne mutlaka sınıflardan türetilmektedir.

Bir temel sınıf, içerisindeki kodların kendisinden türeyen sınıflar tarafından değiştirilebilmesini istiyorsa bu üye **virtual** anahtar kelimesi ile işaretlenmelidir. Türeyen sınıf, virtual anahtar kelimesi ile işaretli bir üyenin uygulanışını kendi sınıfına ait bir iş mantığıyla değiştirmek isteyebilir ancak zorunda değildir. Üyenin başına **override** anahtar kelimesi yazılarak yeniden kodlanması ile böyle bir değişiklik mümkün olmaktadır. Ayrıca ezilen (overriden) her üye, ihtiyaç duyulması halinde temel sınıfta yer alan uygulanışı yeniden çağırmakta serbesttir. Kod içerisinde istenen herhangi bir yerde temel sınıfın bir üyesi **base** anahtar kelimesi ile çağırılabilir.

Kapsülleme , kalıtım, çok biçimlilik neye yarar sorusuna cevap:

1. Temel sınıfta yer alan bir üyenin, türeyen sınıf tarafından nasıl değiştirileceğine dair bir yol sunmaktadır.
2. Bir türetilen sınıfın özellik ve davranışlarını, bir türeyen sınıf miras alarak sanki kendi üyesiymiş gibi kullanmaktadır.
3. Dilin nesne kullanıcısından gereksiz uygulama ayrıntılarını saklayabilmektedir.
4. Dilin var olan sınıf tanımlamalarının üzerine yeni bir sınıf tanımlaması inşa edilmesine izin verebilmesidir.

# Sınıf Üyeleri

Sınıf üyeleri temelde iki kısma ayrılmaktadır. Veri üyeleri ve fonksiyon üyeleridir.

## Veri Üyeleri

### Alanlar

Nesneye ait bilgilerin tutulduğu üye değişkenlerdir.

### Sabitler

Nesnenin değiştirilemeyen kısımlarıdır.

### Olaylar

Belirli bir tetilemenin meydana gelmesi sonucunda çağırılacak olan fonksiyonların adres bilgilerini tutmakla sorumludur.

Sınıfın veri üyeleri değer türlerinde olabileceği gibi referans türlerinde de olabilir. Her nesnenin sahip olduğu data diğer nesnelerden ayrılmıştır. Yani her nesnenin datası kendisine aittir ve hiçbir şekilde karışmaz.

## Fonksiyon üyeleri

Yöntemler (methods), özellikler (properties), yapıcı fonksiyonlar (constructors), yıkıcı fonksiyonlar (destructors), operatörler ve dizinleyiciler (indexer)

### Yöntemler

Nesnenin davranış şekilleridir. Nesne kullanıcısına üye veriler üzerinde işlem yapmasına izin verir ve nesnenin dışa kapalı üyelerine ulaşılmasını sağlar.

### Özellikler

Alanlar gibidir. Hatta bunlara akıllı alanlar da denilmektedir. Nesnenin üye verilerine değer atanmasının ve değer çekilmesinin kontrollü olarak yapılmasını sağlar.

### Yapıcı ve Sonlandırıcı Yöntemler

Nesnenin özel yöntemleridir. Yapıcı yöntemler nesne ilk oluşturulduğunda çağırılırken sonlandırıcı yöntemler nesnenin ömrü tamamlandığında çağırılır.

### Operatörler

Nesnelerin daha esnek kullanılmasına izin vermektedir.

### Dizinleyiciler

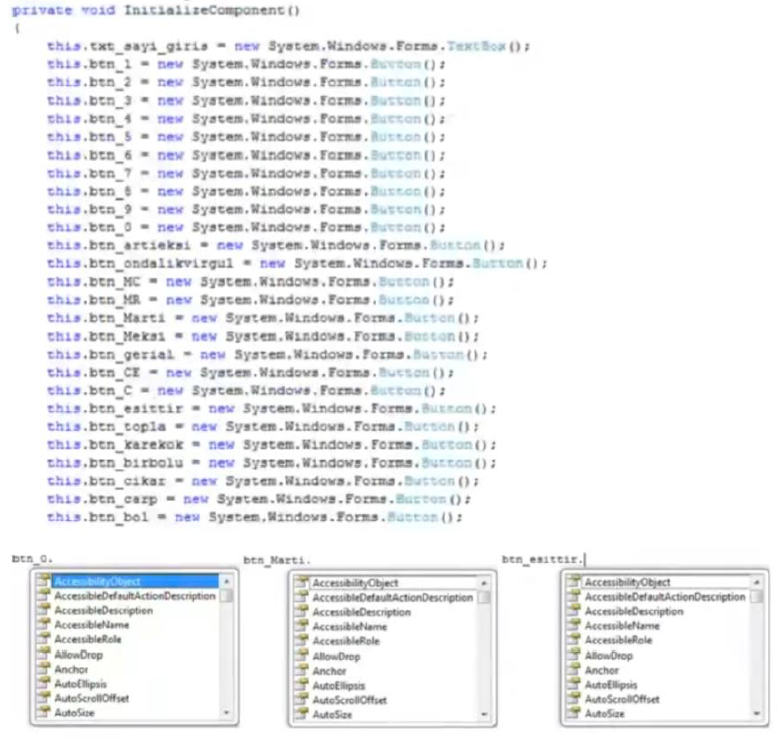
Nesnenin bir dizi gibi kullanılmasına izin verir.

Nesne yönelimli programlamada her sınıf kendine ait özelliklere, metotlara ve olaylara sahiptir. Bir sınıftan başka nesneler de oluşturulabilir. Nesneleri oluşturan temel yapı **sınıf** (class) yapısıdır. Sınıf, o nesnenin yapacağı işleri ve özelliklerini barındıran kod bloğudur. Bu sınıftan tanımlanmış değişkenler birer nesnedir. Bir sınıftan “**New**” komutu ile bellekte bir kopya oluşturulduğunda bu kopyaya nesne denir. Bir sınıftan aynı anda birbirinden bağımsız çok sayıda nesne oluşturulabilir. Aynı sınıftan oluşturulmuş olunan bu nesneler **properties** sayesinde farklı özelliklere de sahip olabilir. Sınıflar başka bir sınıfın özelliklerini alabilir. Daha önce tanımlanmış olan bir sınıftan yeni bir sınıf tanımlanıp bu sınıfa ek özellikler kazandırılabilir. Aşağıdaki hesap makinesi uygulamasında 27 tane buton kullanılmıştır. Form üzerine yerleştirilen her bir yeni buton aslında, buton sınıfından türetilerek onun özellik ve metotlarını beraberinde sahiplenir. Sonra yeni türetilmiş olan bu kontrolün özellikleri değiştirilse de ana unsurları diğer buton nesneleri ile aynıdır.



Yukarıdaki uygulamaya ait **Form1.Designer.cs** dosyasına bakıldığında forma ait her kontrolün o kontrole ait sınıftan türetildiği görülebilir.

**System.Windows.Form.Button()** şeklindeki ifade ile btn\_1 kontrolünün Button isimli sınıftan türetilmesi sağlanır.



Yukarıda görüldüğü gibi bir sınıftan başka sınıflar türetildiğinde hepsi bu ana sınıfına ait özellik ve metotları da beraberinde alırlar. Visual C# programında form üzerine yerleştirilen her kontrol arkaplanda bu kontrolü ifade eden ana sınıftan üretilir.

Bu durum bir örnek üzerinden açıklanırsa; **Motorlu\_Arac** isimli bir sınıf var olsun ve bu sınıftan yola çıkarak **Otomobil** sınıfı türetilirse otomobil sınıfı araç sınıfının tüm özelliklerini sahiplenir. Motorlu\_Arac isimli bir sınıf olsun ve aşağıdaki gibi özellikleri olsun.

class Motorlu\_Arac

Özellikleri:

Renk

Depo\_hacmi

Yolcu\_kapasitesi

Metotlar:

Çalışmak

Durmak

Gitmek

Yukarıdaki gibi bir sınıf tasarımında her motorlu aracın belli başlı özellikleri veya değerlendirme kriterleri vardır. Renk, depo hacmi vb. özellikleri (properties) oluştururken metotlardan kasıt yaptığı işlerdir (events).

Sınıf yapısında soyutlama işlemi de yapılabilir. Yani sınıf oluşturulduğunda yapacağı işler için gerekli olan kodlar yazıldığında bu sınıfı kullanmak isteyen başka bir programcının o sınıfın nasıl yazıldığını bilmesi gerekmez. Sadece o sınıfa ait özelliklere değer ataması, metotları çağırması ve olaylara kod yazması yeterli olacaktır. C# programında yüzlerce sınıf vardır ancak hiçbirinin içerisindeki kodların bilinmesine gerek yoktur. Bilinmesi gereken o sınıfa ait metotların ve özelliklerin nasıl kullanıldığıdır.

Türkçe kullanımı Nesneye Dayalı Programlama olan bu yöntem son dönemde popüler tüm diller tarafından imkan verdiği ölçüde kullanılmaktadır. Bu tür programlama tekniklerinde amaç işi yapan metodu bir yerde tanımlamak şayet diğer sınıflar da aynı işi yaptıracaksa içerisinde ikinci kez aynı kodları yazmak yerine önceki sınıfa başvurarak o metodu kullanmak istediğini belirtmektir. Bu sayede aynı kod bloğunun tekrar yazılmasından doğacak olan hata riski azaltılır. Ayrıca projenin çok daha az yer kaplaması sağlanarak veriden tasarruf edilir. NDP yönteminin en çok kullanıldığı projeler dll uzantılı uygulamalardır. Bu uygulama referans gösterilerek istenilen projede kullanılabilir.

# Sınıf Tanımlarında Kullanılan İfadeler

## : operatörü

Bir sınıf tasarlanırken başka bir sınıfın özelliklerini devralması için kullanılır.

**public class ceptelefon:telefonlar**

ceptelefon sınıfının telefonlar sınıfından özelliklerini alması sağlanır.

## sealed

Bir sınıf tanımlanırken bu deyim kullanılırsa bu sınıftan başka sınıfların türetilmeyeceği bildirilir.

**public sealed class ceptelefon:telefonlar**

Bu tanımlama ile ceptelefon sınıfının özellikleri alınarak başka bir sınıf türetilemez.

## abstract

Bir sınıf tanımlanırken bu ifade kullanılır ise bu sınıfın mutlaka kalıtım edilmesi gerektiği vurgulanır. Ancak bu tür bir sınıftan nesne oluşturulamaz. Sınıftan yeni sınıflar türetilip o sınıftan nesneler türetilebilir.

**public acstract class telefonlar**

Bu ifade ile bu classtan yeni bir nesne oluşturulamayacağı anlaşılır. Ancak başka bir sınıf oluşturulup bu yeni sınıftan nesne oluşturulabilir.

## public

Bir nesne veya sınıf bu anahtar kelime ile tanımlanır ise o nesneye diğer tüm sınıflardan da ulaşılabilir.

## private

Bir nesne veya sınıf private olarak tanımlanır ise o nesneye diğer sınıflardan ulaşılamaz. Sadece bulunduğu sınıf içerisinde kullanılabilir.

## protected

Bir nesne veya sınıf protected olarak tanımlanır ise o nesneye sadece o sınıf içerisinden ve o sınıftan türetilmiş olan diğer sınıflardan erişilebilir.

## static

Bir değişken static olarak tanımlanmış ise sınıfın bütün nesneleri-kopyaları tarafından ortak olarak kullanılabilir.

## Overload

Aynı isimde farklı parametrelere sahip prosedürlerin yazılmasına denir.

## Override

Temel sınıf içinde tanımlanmış olan bir prosedürün yerine türetilmiş sınıftan aynı isimde bir prosedürün yazılması durumunda override ile diğer prosedürün pasif hale getirilmesi (ezilmesi) olarak tanımlanabilir.

## virtual

Bir prosedürün görevi pasifleştirilip aynı isimde bir prosedür yazılacak ise bu prosedür virtual olarak tanımlanır. Türetilmiş sınıflar bu prosedürü ezip kendi prosedürlerini yazabilirler.

## base

Bu deyim türetilmiş sınıf içerisinde temel sınıfa erişmek için kullanılır.

## this

Bulunulan sınıfı tanımlamak, ait olunan sınıfa erişmek için kullanılır. Örneğin bir prosedürün içerisinde x isimli yerel bir değişken varsa ve aynı isimde bir değişken sınıfın içerisinde varsa x=5 ifadesiyle prosedürün içerisindeki değişkene atama yapılır. this.x=5 ifadesi ile classın içerisindeki değişkene atama yapılır.

## constructor

Bir sınıftan yeni bir nesne oluşturulduğunda otomatik olarak çalışan prosedürlere denir. C#’da constructor sınıf ile aynı isme sahip prosedürlerdir.

## destructor

Sınıftan oluşturulan nesne yok edildiğinde otomatik olarak çalışan prosedürlere denir. Bu prosedür sınıf ile aynı isme sahip “~” ile başlayan prosedürdür.

## Garbage Collector

Bu ifade ile program içerisinde üretilen nesneleri takip ederek onlara gerek kalmadığı anda o nesne için ayrılan bellek bölgesi serbest bırakılır. Bir sınıftan yeni bir nesne oluşturulduğunda sınıftaki değişkenlerin bir kopyası bu nesneye verilir ve bu nesne hafızada bir alan kaplar. Oluşturulan nesne yok edilmese bile Garbage Collector belirli sürelerle kontroller yaparak ihtiyaç duyulmayan nesneleri yok eder. Böylece hafızada gereksiz yer işgali engellenmiş olur.

## Dispose

Bellekte çok fazla yer işgal eden nesneler Gargabe Collector’ü beklemeden kullanıcı tarafından yok edilebilir. Bir nesnenin dispose metodu kullanılarak bellekte o nesne için ayrılmış olunan yer serbest bırakılır.

## Namespace

Bu deyim ile sınıfların gruplandırılması yapılır. Benzer işi yapan çok sayıda sınıfı bir namespace içinde toplayarak hem sınıflar derli toplu tutulur hem de bu sınıflar kullanılmak istendiğinde o sınıfların bulunduğu namespace’ler using anahtar kelimesi ile koda dahil edilebilir. .Net Framework’e ait bir çok sınıf namespace ile gruplandırılmıştır. Çizimle ilgili sınıflar Drawing, disk işlemleri ile ilgili sınıflar IO namespaceleri içerisinde gruplandırılmıştır.

namespace Okul

{

class ogrenci

{

}

class ogretmen

{

}

class memur

{

}

}

ogrenci sınıfından bir nesne tanımlanacağı-oluşturulacağı zaman:

**okul.ogrenci ogrenci1 = new okul.ogrenci();**

Eğer namespace olarak tanımlanmış olan Okul grubu using ifadesi ile programa eklenir ise sadece sınıf isimleri ile tanımlamalar yapılabilir.

**using Okul;**

C# programı ekranında en üste kod yazım sahasının tepesinde yer alan alana yazılır.



Yukarıdaki gibi Okul namespace’i programa eklendiğinde içindeki sınıflara erişmek için şu şekilde kod yazımı yapılabilir.

**ogrenci ogrenci2 = new ogrenci();**

Sınıfların uygulamaya dahil edilmesi için **Project/Add** sınıf menüsü adımından sonra aşağıdaki resimde gösterildiği gibi bir pencere açılır. Name kısmına sınıfın adı verildikten sonra Add düğmesine basılır.





# Constructor

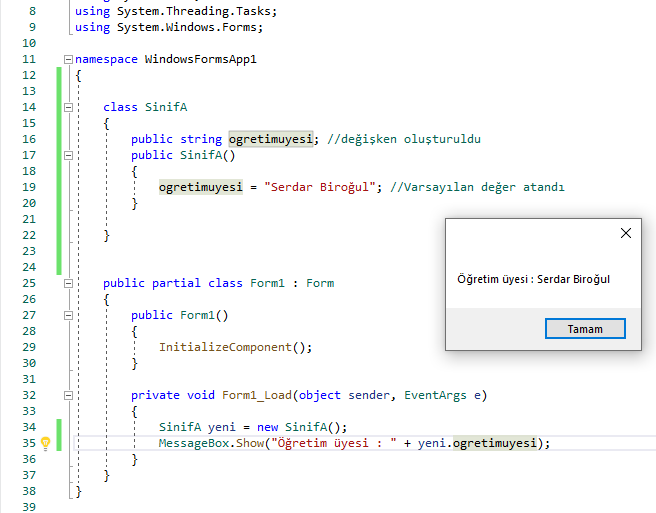
Bir sınıftan new komutu ile yeni nesneler yaratıldığı anda işletilecek olan kod blokları varsa Constructor yaratılmalıdır.

Windows uygulamalarında form yüklenirken kullanıcı henüz bir işlem yapmadan Load yordamı işlenir. Constructors da aynı bu yordam gibi çalışır. Burada Load yordamı ile constructor arasındaki fark yaratılan her classın load yordamı olması beklenmez. Ayrıca işleyiş zamanı ve bazı ufak teknik farklılıklar da vardır.

Constructor yani kurucu, herhangi bir sınıfın nesnesi oluşturulacağı esnada o sınıf içerisinde en başta çalıştırılacak kod parçasıdır. Oluşturulan sınıflarda birer kurucu oluşturmak zorunda olunmasa da c# için arka planda o sınıfı oluşturmak adına bir kuruya ihtiyacı vardır. Dolayısıyla program boş bir kurucu oluşturur. Faakt bir kurucu oluşturulduğunda c# arkaplanda ikinci bir kurucuyu asla oluşturmaz. Sınıftan nesne oluşturulurken ilk çalışacak olan kısmın kurucu olduğu göz önüne alındığında belirli bir kod parçasının sınıf kurulduğu esnada çalışması istenirse bu kod parçası sınıf içerisinde bulunan constructor içerisine yazılır.

C#’da constructor bloğu class ismi ile aynı olmak zorunda ve dışarıdan erişilebilecekleri için bu yordamın public olarak tanımlanması gereklidir. Aşağıda görüldüğü gibi constructor bloğu sınıf içerisinde oluşturulmuştur.

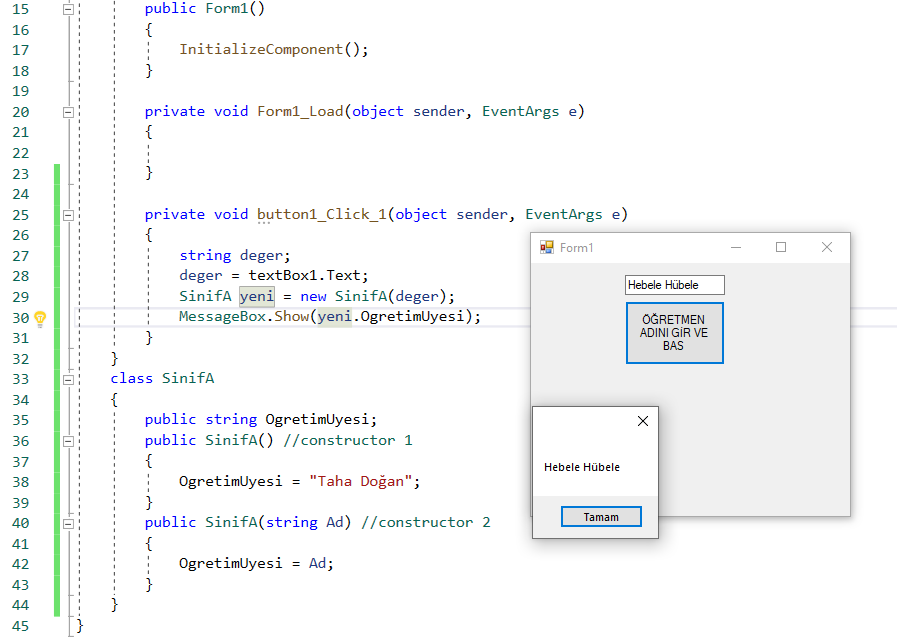
**(Visual Studio önce main fonksiyonu yazmamızı istedi. Class yapısını aşağı yazdık.)**



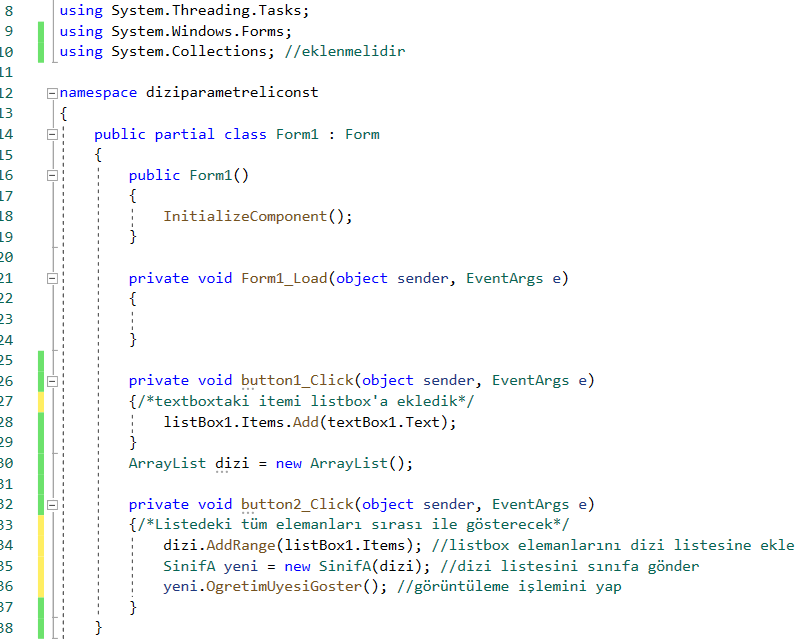
Yukarıdaki kodların işletilmesinde; SinifA classından **yeni** isimli bir değişken tanımlanmıştır. Ardından new komutu ile bu sınıfa ait yer bellekte otomatik olarak oluşturulmaktadır. new komutu sınıfın ismiyle aynı olan contruct bloğunu işlettiği için varsayılan değer olan OgretimUyesi değişken içeriği formda messagebox olarak geri döndürülür. MessageBox program açılır açılmaz karşımıza çıkar.

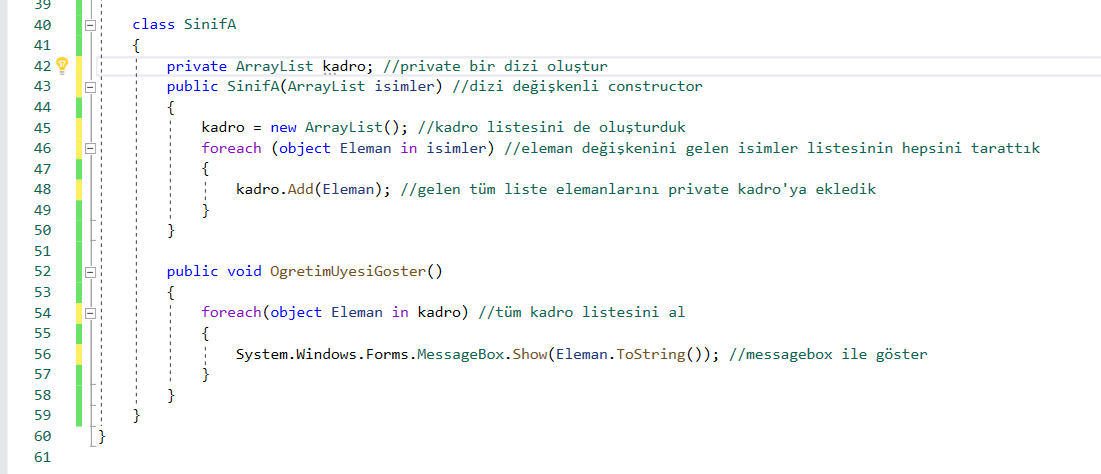
## Birden Fazla Constructor Oluşturmak

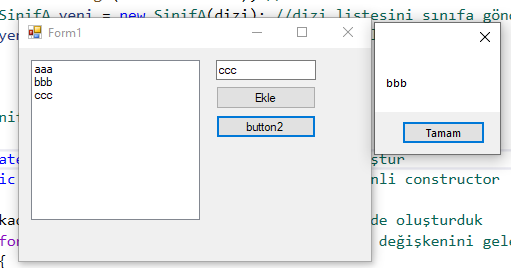
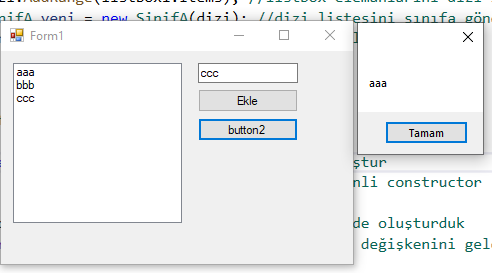
Class içerisinde birden fazla constructor yaratılabilir. Ancak constructor ismi class ile aynı isimde olacağından aynı isimli birden fazla constructor oluştururken parametrelerden faydalanacağız. Nesne yaratılırken kullanılacak parametreye göre hangi constructor kullanılacağı belirlenecektir.

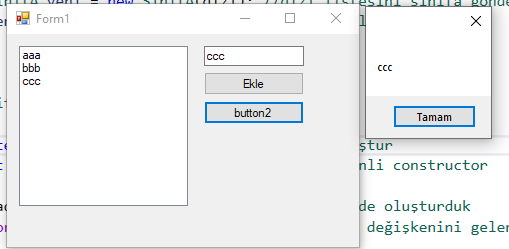


## Dizi Parametreli Constructor Oluşturmak

Bazı durumlarda class oluşturulurken dizi değişken kullanmak zorunda kalınabilir. Bu durumda **ArrayList** dizi sınıfı değişkeni kullanılarak class yaratılabilir. (.AddRange() listenin sonuna ekleme yapar.) 



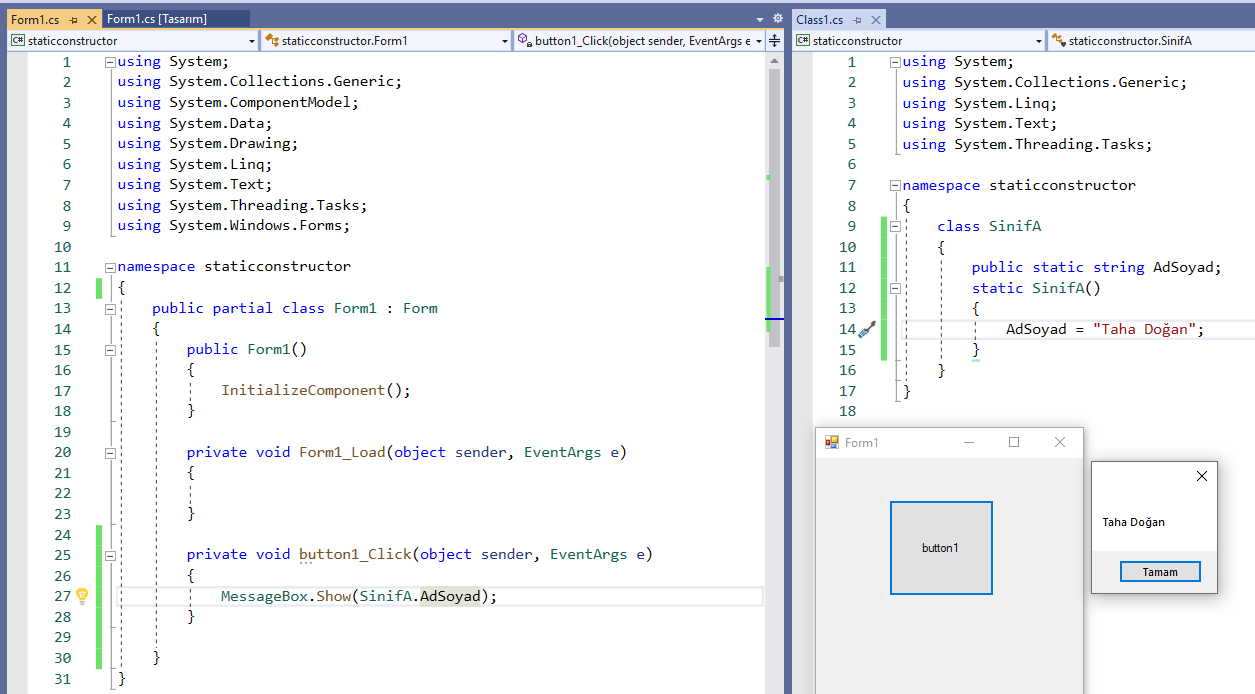




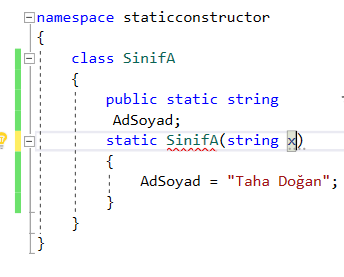
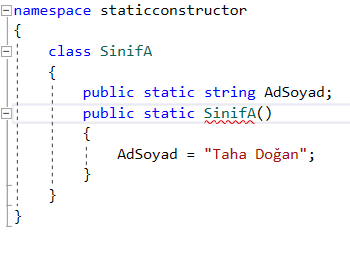
Tamam tuşuna bastıkça sırası ile aaa, bbb, ccc çıktısı geliyor.

## Static Constructor

Class içinde oluşturulan constructor kod bloğu sadece new komutu ile yaratılan yavru değişken tarafından işlenmektedir. Static olarak tanımlanan bir değişken uygulamada kullanılırken constructor bloğunu işletmez.



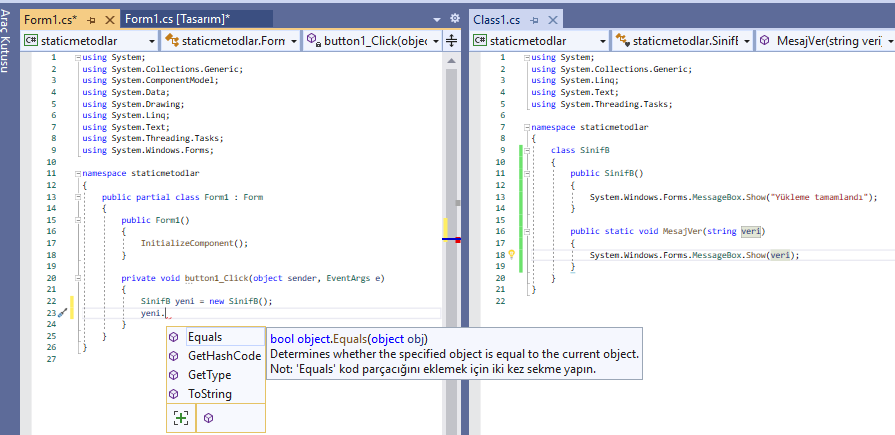
Static constructor yapılarına dışarıdan erişim yoktur. Yani public olarak tanımlanamazlar. Aynı şekilde dışarıdan erişimlerinin olmamasından dolayı içerisinde parametre bulunduramazlar. Aşağıdaki gibi kullanımlar hatalıdır.



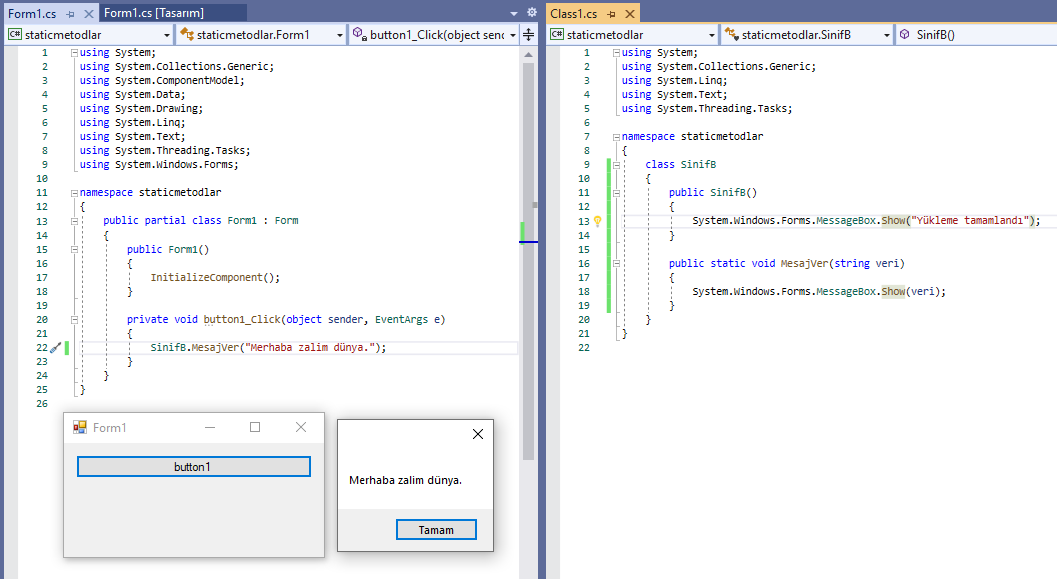
Static constructor oluşturulurken kullanılan tüm değişkenler de static olmalıdır.

## Static Metotlar

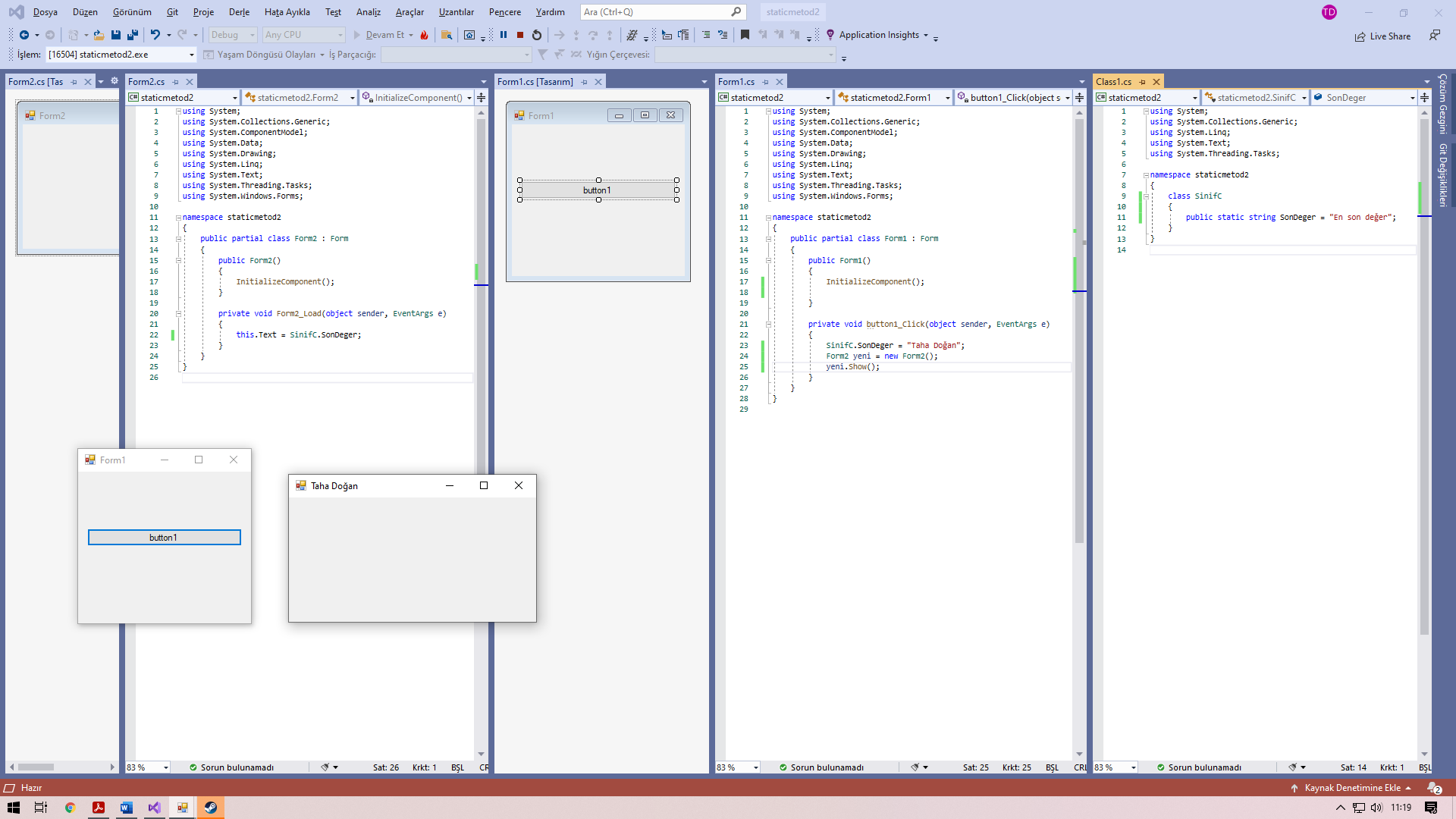
Sınıfa ait metot ve değişkenlerin çağırılma seçeneklerini belirleyen bildiridir. Metod veya değişkenin türetilecek olan yavru değişkenle mi yoksa direkt sınıfa ait isimle mi çağırılacağını belirler. Bir değişken/fonksiyon/prosedür static bildirisi ile tanımlanırsa o değişken türetilmeden sınıfın ismiyle çağırılmak zorundadır. Aşağıdaki uygulamada **MesajVer** prosedürüne yavru değişken tanımlanarak ulaşılamaz.



Görüldüğü gibi MesajVer prosedürüne erişmek için yavru değişken tanımlaması yaparak ulaşamayız. Tanımlama yapmadan, aşağıdaki gibi bir kullanım gereklidir.



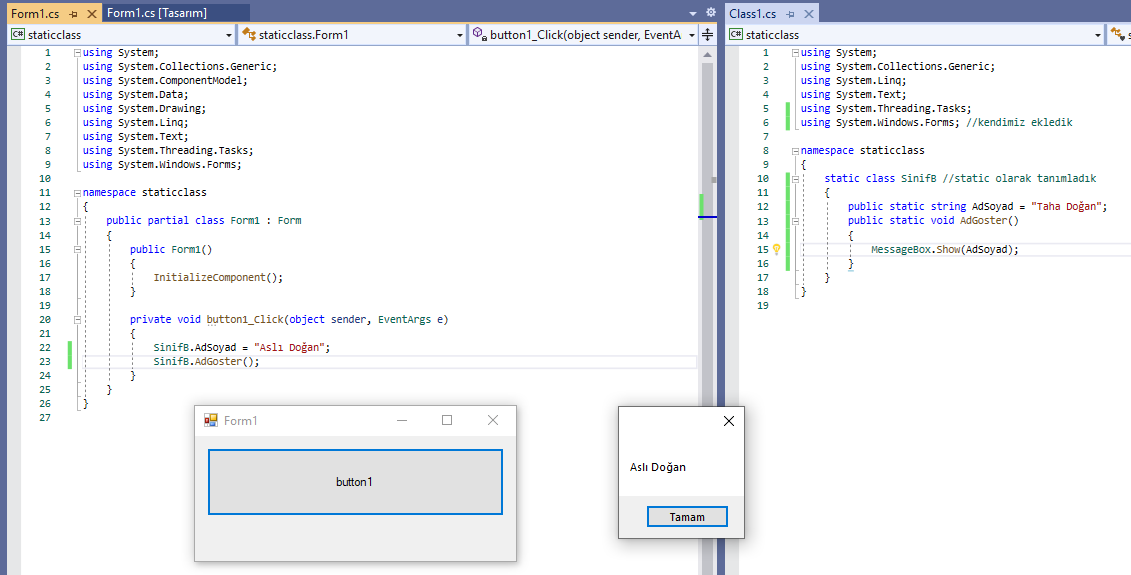
Static bildirisi eklenen metodlar işletilirken constructor kod bloğu (static değilse) kesinlikle çalıştırılmaz. new komutu ile kullanılmazlar. Static metodların tek değişikliği yukarıda bahsedildiği gibi sınıfın direkt ismi ile çağırılması durumu değildir. Bazen farklı sınıflarda üretilen değerlerin diğer sınıfın/sınıfların kullanması gerekebilir. Bu gibi durumlarda bu değişken static olarak tanımlanır. Basit olarak bir formda kullanılan değişkenin en son değerinin başka bir formda kullanılması istenirse static bildiriden faydalanılır. Bununla ilgili bir örnek aşağıda verilmiştir.



Yukarıdaki uygulamada SinifC sınıfına ait static değişkenin değeri “En son değer” olarak belirlenmiştir. Ardından buton kontrolüne basıp 2. form açıldığında SonDeger isimli değişkenin değeri “Taha Doğan” olacaktır. 2. form açıldığında değişmiş olan SonDeger değişkeninin içeriği yazdırılır. Bu veya buna benzer mantıkla bir sınıf içerisindeki değer, diğer bir sınıf içerisinde son hali ile kullanılır.

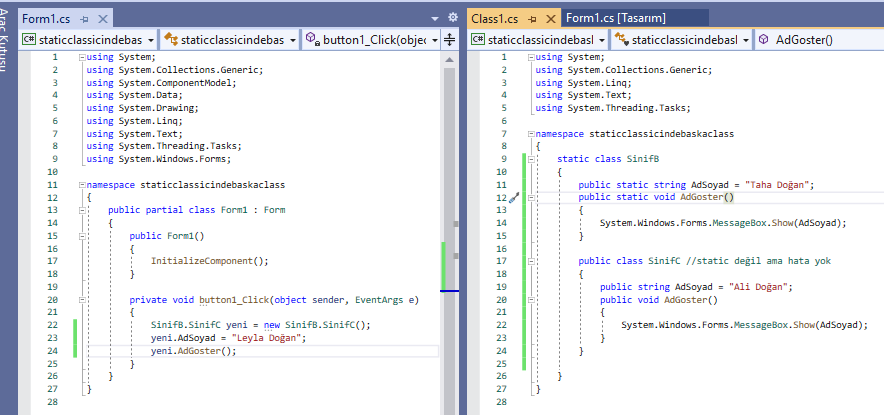
## Static Class Tanımlamak

Static class içerisine sadece static değerler tanımlanabilir. Ayrıca static olan sınıftan kesinlikle yavru bir değişken üretilip kullanılamaz. Aşağıdaki örnek incelendiğinde static sınıfta yer alan metod ve değişkenlere doğrudan ulaşılır.



### Static Class İçerisinde Başka Class Yapıları Oluşturmak

Static sınıflar içerisinde static olmayan bir metod veya değişken kullanılamaz. Ancak bu durum sınıflar için geçerli değildir. Aşağıdaki gibi bir kod bloğu doğru çalışacaktır.

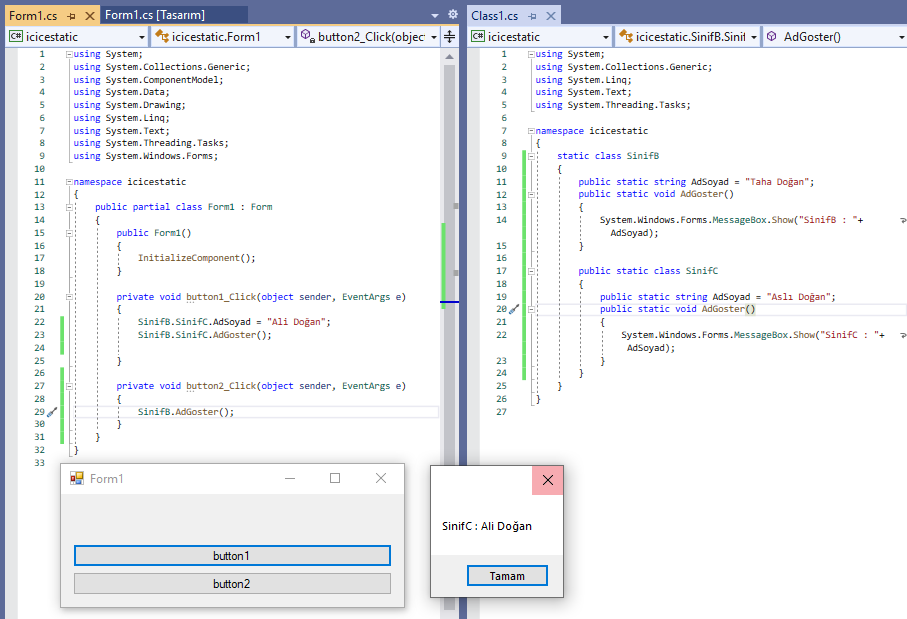


3 kere AdSoyad değişkenini değiştirerek static class içerisinde static olmayan sınıfın kullanımını göstermiş olduk. Static class içerisindeki static olmayan sınıfa ulaşmak için SinifB.SinifC dendiğine dikkat edilmelidir.

### İç İçe Static Class Yapıları

İç içe static sınıf tanımlamaları istendiği sayıda yapılabilir. Burada dikkat edilmesi gereken nokta içerlerinde yer alan metod veya değişken tanımlamalarında static tanımlama yapılmasıdır.

button1 :



button2 :

