# 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

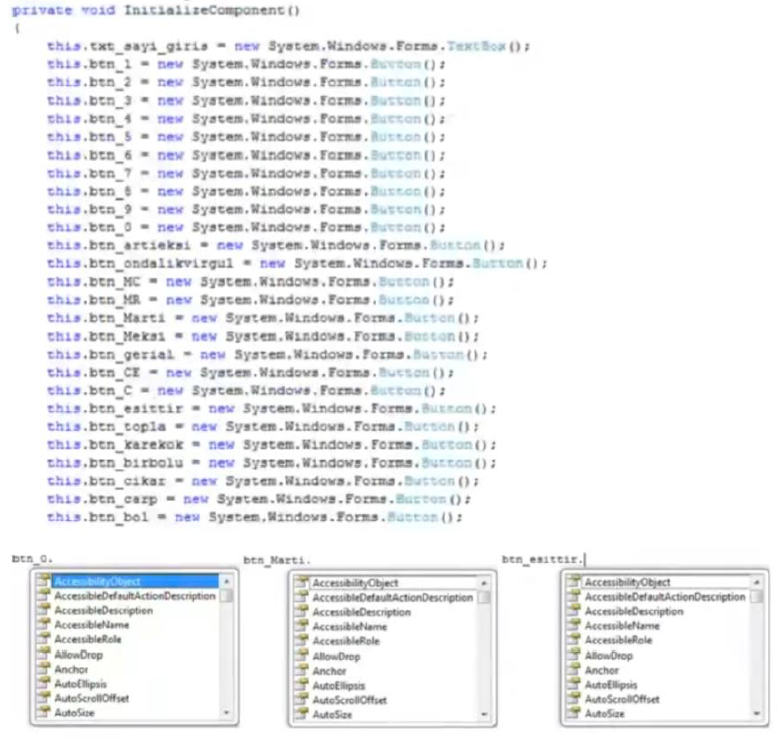
Nesnenin bir dizi gibi kullanılmasına izin verir.

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

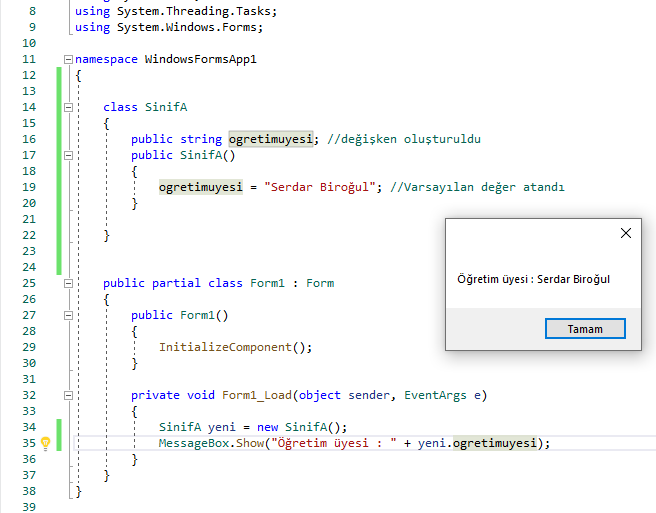
Bir sınıftan new komutu ile yeni nesneler yaratıldığı anda işletilecek olan kod blokları varsa Constructor yaratılmalıdır.

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.

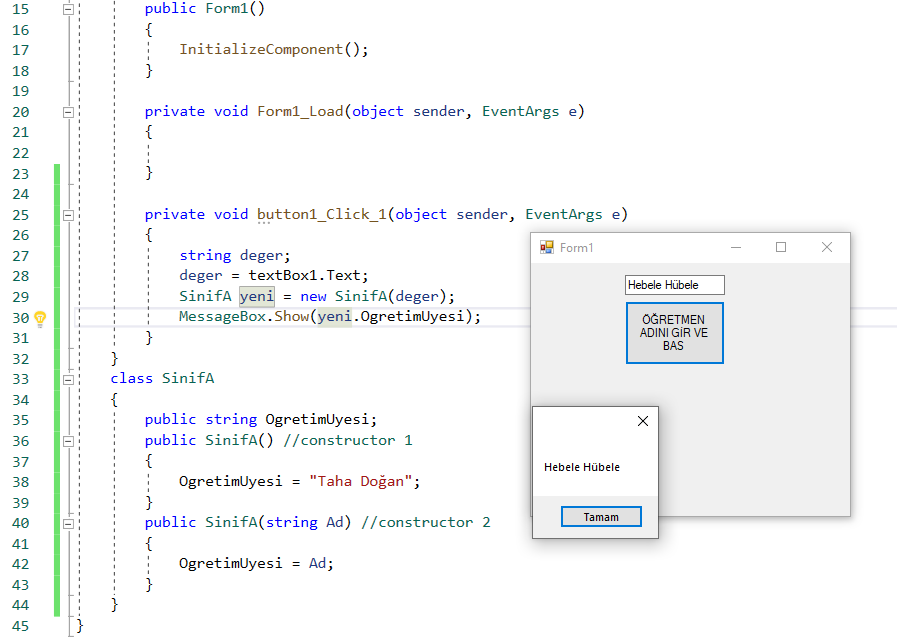
**(Visual Studio önce main fonksiyonu yazmamızı istedi. Class yapısını aşağı yazdık.)**



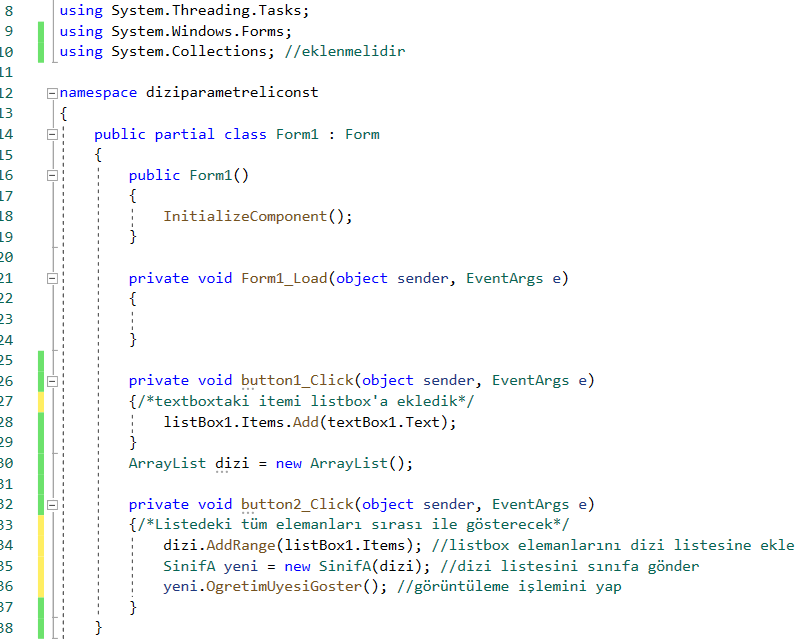
Yukarıdaki kodların işletilmesinde; SinifA classından **yeni** isimli bir değişken tanımlanmıştır. Ardından new komutu ile bu sınıfa ait yer bellekte otomatik olarak oluşturulmaktadır. new komutu sınıfın ismiyle aynı olan contruct bloğunu işlettiği için varsayılan değer olan OgretimUyesi değişken içeriği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

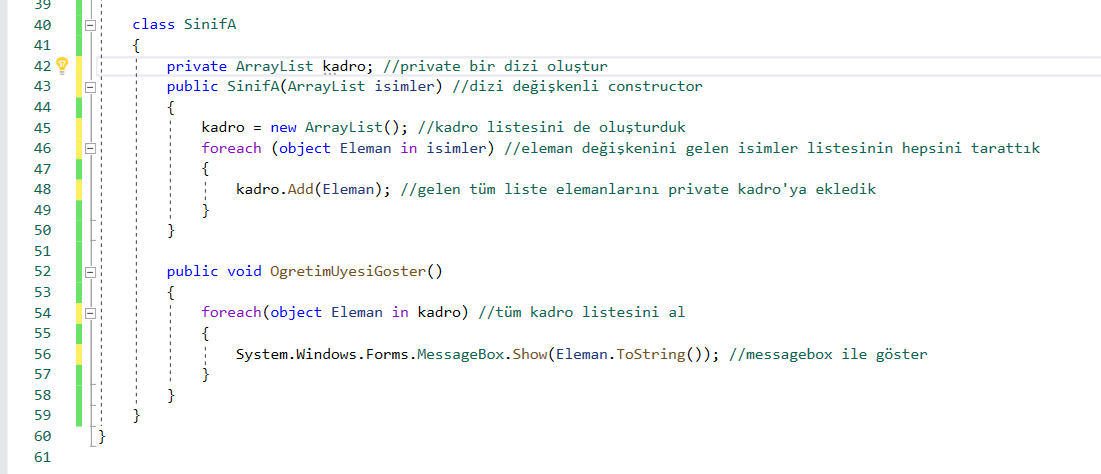
## Birden Fazla Constructor Oluşturmak

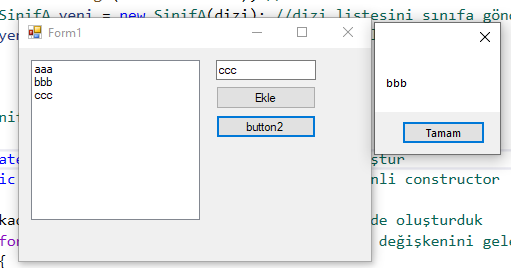
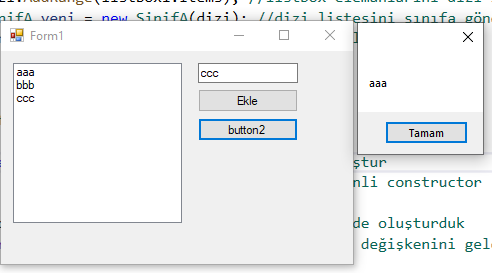
Class içerisinde birden fazla constructor yaratılabilir. Ancak constructor ismi class ile aynı isimde olacağından aynı isimli birden fazla constructor oluştururken parametrelerden faydalanacağız. Nesne yaratılırken kullanılacak parametreye göre hangi constructor kullanılacağı belirlenecektir.

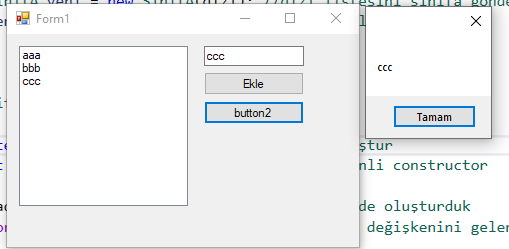


## Dizi Parametreli Constructor Oluşturmak

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



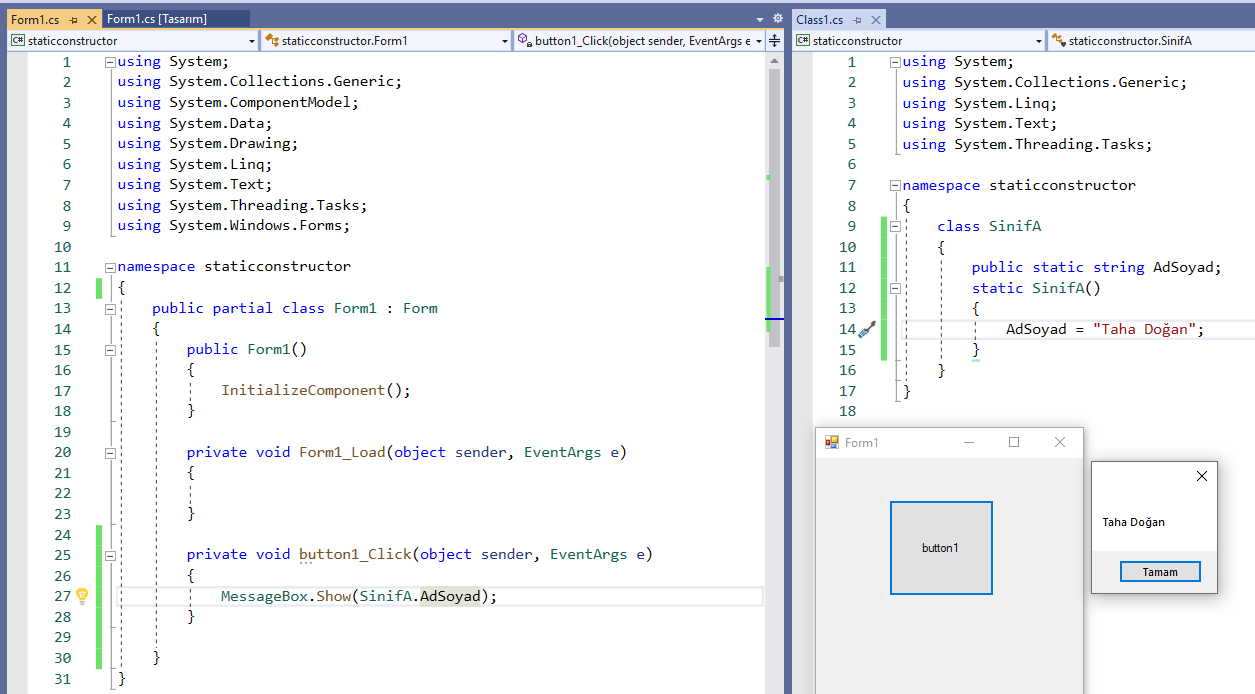




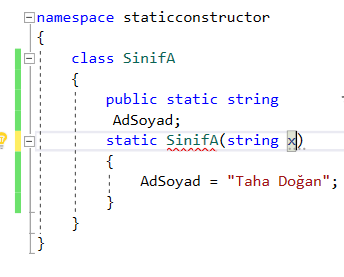
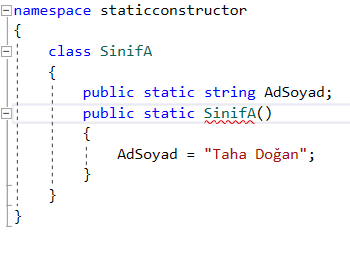
Tamam tuşuna bastıkça sırası ile aaa, bbb, ccc çıktısı geliyor.

## 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.



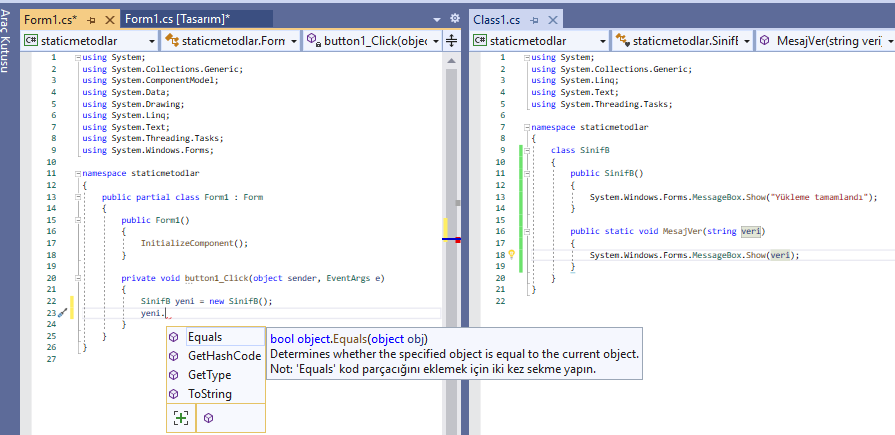
Static constructor yapılarına dışarıdan erişim yoktur. Yani public olarak tanımlanamazlar. Aynı şekilde dışarıdan erişimlerinin olmamasından dolayı içerisinde parametre bulunduramazlar. Aşağıdaki gibi kullanımlar hatalıdır.



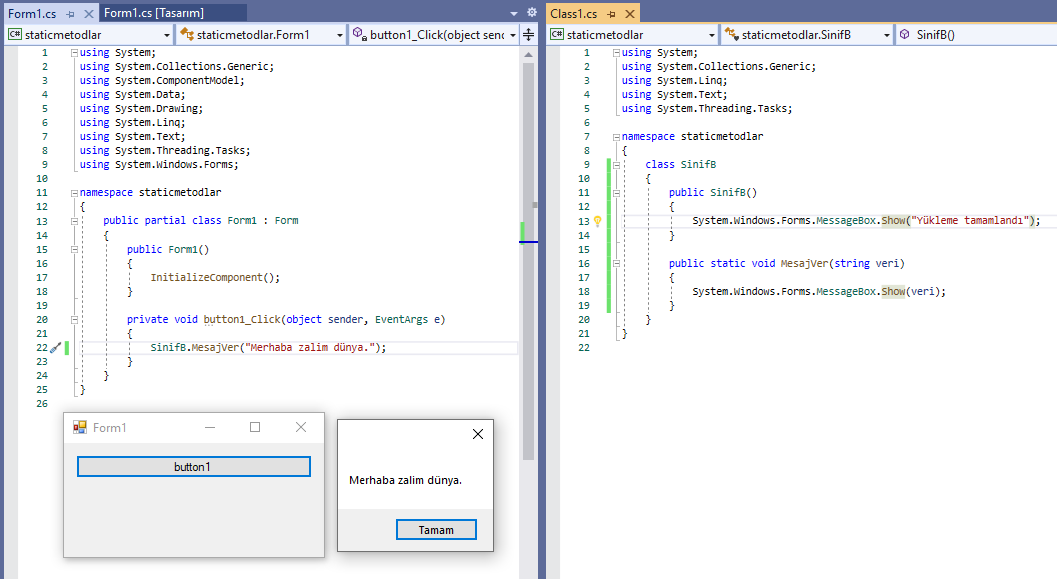
Static constructor oluşturulurken kullanılan tüm değişkenler de static olmalıdır.

## Static Metotlar

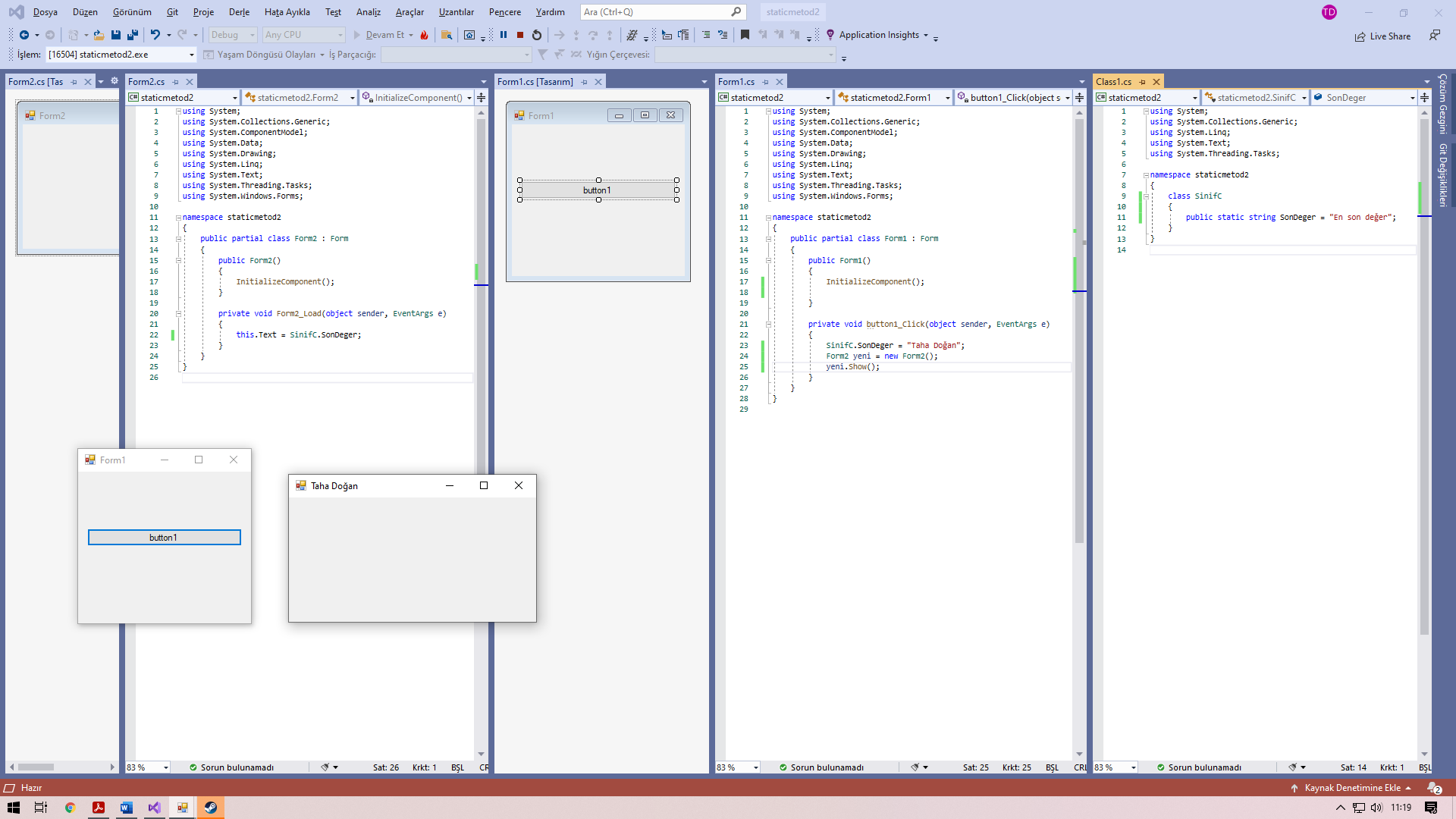
Sınıfa ait metot ve değişkenlerin çağırılma seçeneklerini belirleyen bildiridir. Metod veya değişkenin türetilecek olan yavru değişkenle mi yoksa direkt sınıfa ait isimle mi çağırılacağını belirler. Bir değişken/fonksiyon/prosedür static bildirisi ile tanımlanırsa o değişken türetilmeden sınıfın ismiyle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



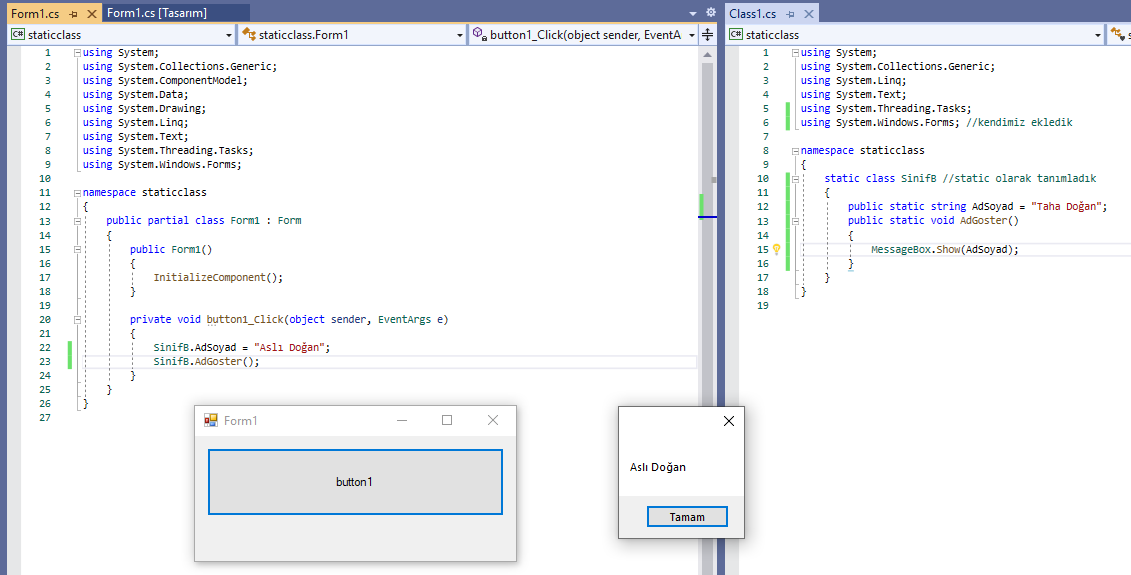
Static bildirisi eklenen metodlar işletilirken constructor kod bloğu (static değilse) kesinlikle çalıştırılmaz. new komutu ile kullanılmazlar. Static metodların tek değişikliği yukarıda bahsedildiği gibi sınıfın direkt ismi ile çağırılması durumu değildir. Bazen farklı sınıflarda üretilen değerlerin 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.

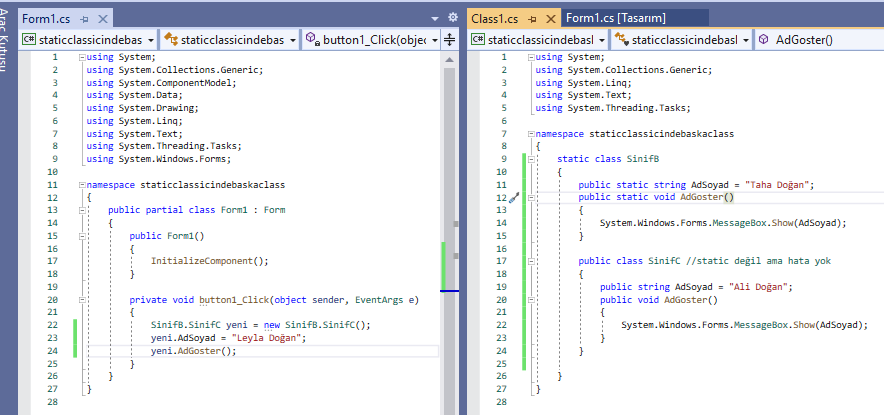
## Static Class Tanımlamak

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.

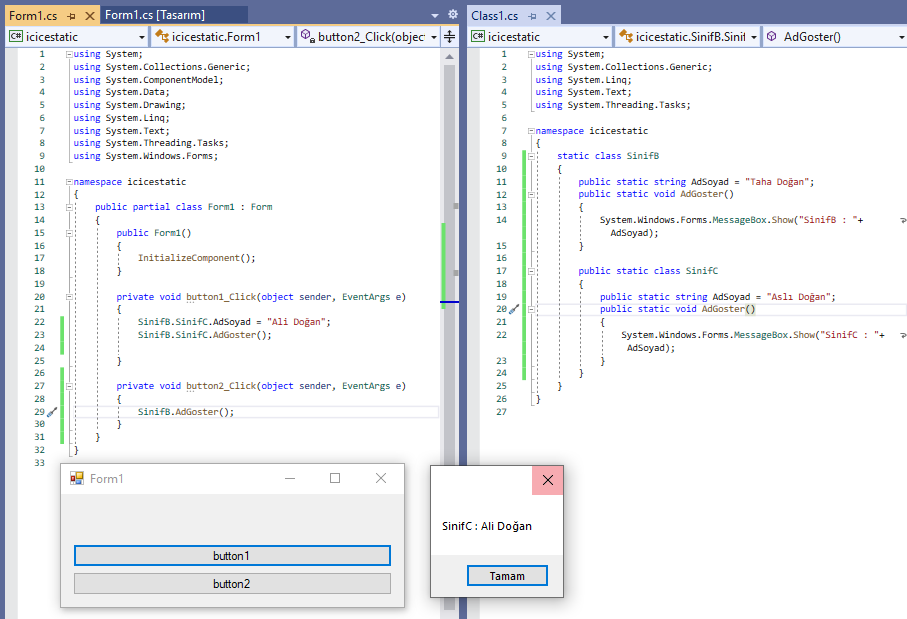


3 kere AdSoyad değişkenini değiştirerek static class içerisinde static olmayan sınıfın kullanımını göstermiş olduk. Static class içerisindeki static olmayan sınıfa ulaşmak için SinifB.SinifC dendiğine dikkat edilmelidir.

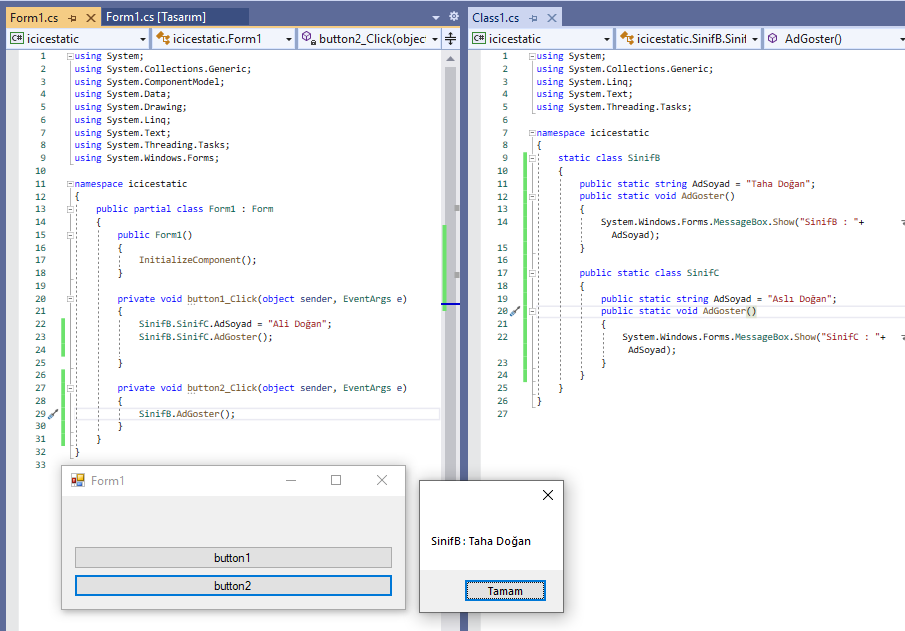
### İç İçe Static Class Yapıları

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 :

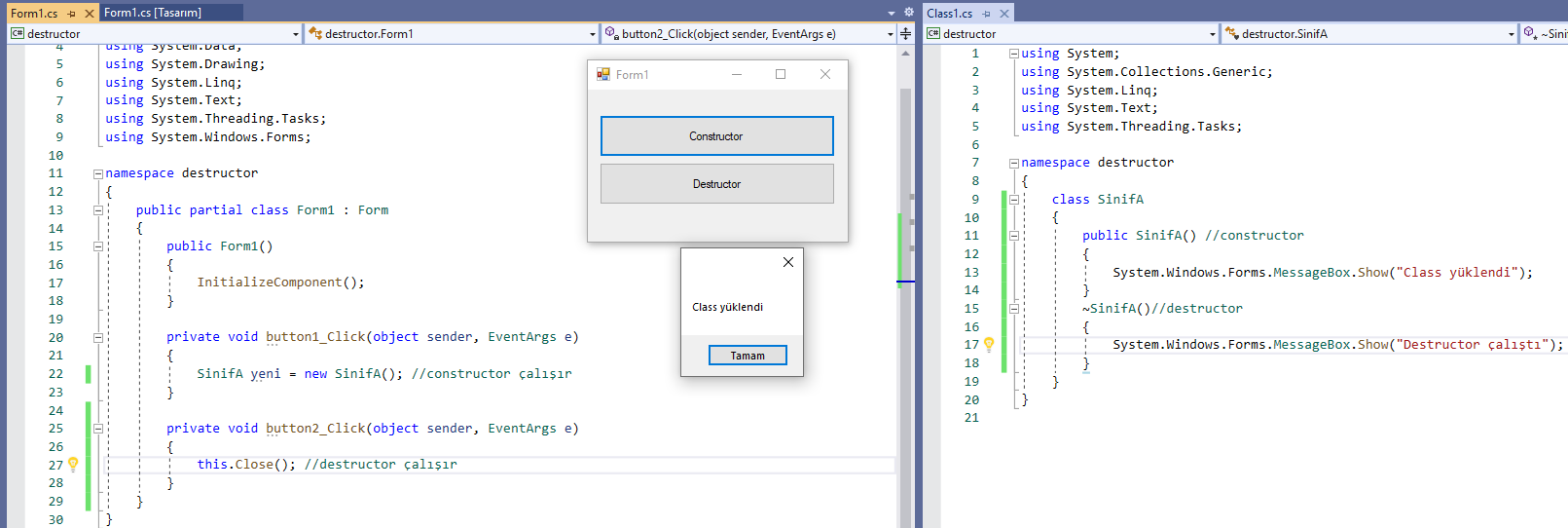


## Destructor Oluşturmak

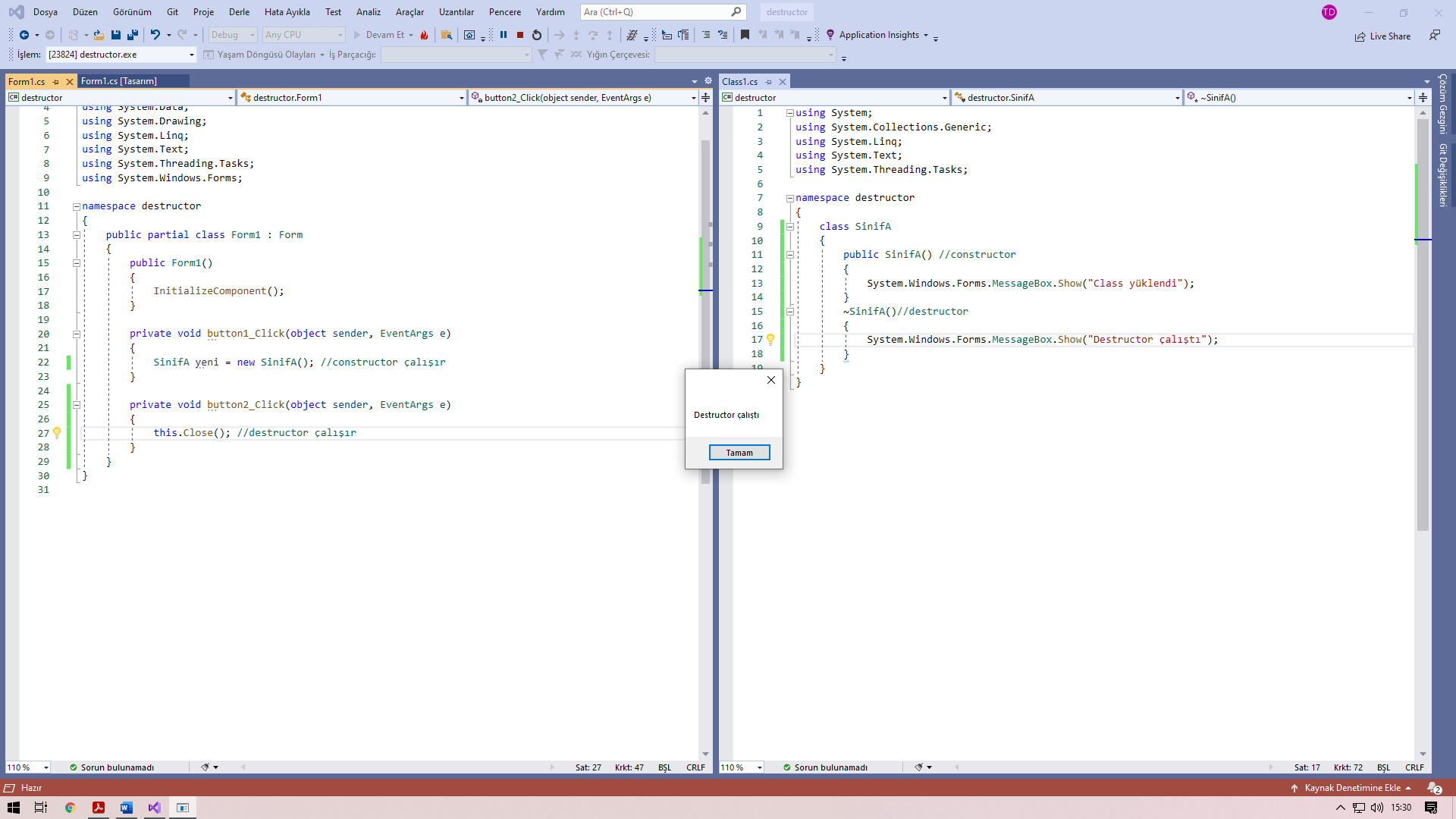
Bir class new komutu ile oluşturulduğunda constructor kod bloğu otomatik olarak işletilir. Aynı şekilde class bellekten atılırken de destructor bloğu işletilir. Benzer olarak Windows uygulamlarındaki kontrollere ait olaylardan biri olan **Close** olayının işletilmesi gibi düşünülebilir. Class bellekten atılırken işletilecek olan kodlar bu blok altına yazılır.

Kodlarla oluşturulan sınıfın görevi bittiğinde onun muhakkak bellekten de temizlenmesi gerekmektedir. Aksi taktirde o nesne kullanılmayacak olmasına rağmen bellekte kalacak ve uygulamanın performansının düşmesine neden olacaktır. Bu olay nesnel uygulamalarda sıkça yapılan bir hata olup ancak bilgisayar yeniden başlatılınca veya uygulama yeniden başlatılınca sorun giderilir.

Constructor butonuna basıldı:



Destructor butonuna basıldı:



3 saniye civarı bekleyip uygulama otomatikmen kapanıyor ancak constructor butonuna basmadan destructor butonuna basar isek tepkisiz direkt olarak kapanıyor.

## Sınıfları Bellekten Temizlemek

Bellek yönetimi için geliştirilen çöp toplama aracı (Garbage Collection - GC), yeni nesnelere yer açmak (allocate) için bellekteki uygun alanları serbest bırakan (deallocate) düzenektir. GC sistemi, uzun süre erişilmeyen, kullanılmayan nesneleri bellekten kaldırır, oluşturulan yeni nesneler için heap alanı üzerinde yer açar.

.NET Framework’ün sunduğu çöp toplama aracı programlarda kontrol etmekte zorluk çekilen bu bellek sızıntılarını otomatik olarak yok etmektedir (automatic object lifetime management). GC, hem zaman hem de maliyet açısından çöpleri bir başkasının toplaması gerektiğinin bir sonucu olarak ortaya çıkmıştır.

Garbage Collection mekanizmasının algoritmasını iyi anlamanın yolu bellek birimlerini tanımaktan geçer. Bir programın çalışmaya başlama ve devam etme sürecinde CPU ve RAM cephesinde bazı operasyonlar gerçekleşir. Program belleğe yüklendiğinde belleğin segment adı verilen üç farklı alanına yayılır.

* Code Segment (Programın kod alanı)
* Stack (Yığın)
* Heap

### Code Segment

Text segment olarak da bilinir. Programa ait makine kodunu yani programın CPU’ya göndereceği komutları tutar. Geri kalan stack ve heap alanında (Data Segment – DS) ise programın kullanıldığı veri ve kaynaklar tutulur. Bellekteki bu verilere ulaşmak için CPU tarafından register denen lokasyonlar kullanılır. İşlemci çekirdeğine gömülü özel bellek birimi olarak düşünebileceğimiz register alanları, genel olarak matematiksel işlemler ve bellekteki verilere ulaşma işlemlerini gerçekleştirir. Bir program yüklendiği zaman işletim sistemi, programdaki komut ve değişkenleri belleğe yükler ve ilgili segmentlerin adreslerini bahsi geçen registerlere aktararak programın ilk komutu ile işlemini programa bırakır.

### Yığın (Stack)

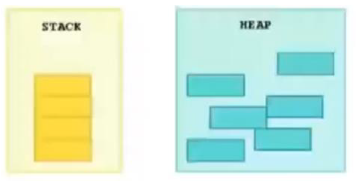
İşlemci tarafından verilerin geçici olarak saklandığı veya uygulamanın kullandığı değişkenlerin tutulduğu ve büyüklüğü işletim sistemine göre değişen bellek bölgesidir. Yığının büyüklüğü, yönetimi, programcının değil uygulamanın sorumluluğundadır. Derleyici veya çalışma ortamı, program içindeki değişkenleri ve uzunluklarını tarar ve yığın üzerinde ona göre ayrım yaparak yerleştirir. Bu yüzden derleyici program oluşturulmadan önce yığın üzerinde oluşturulacak verilerin boyut ve ömürlerini bilmek zorundadır. Bir değişkenin hangi bellek bölgesine yerleştirileceğini CPU üzerindeki Stack Pointer (SP) denen register belirler. SP yığın alanının en üst kısmının yani belleğin o anki yerinin adresini gösterir. Depolanacak verilerin eklenmesi veya silinmesi ile SP değeri azalır veya artar.

Bu bellek biriminin yığın olarak anılması LIFO (Last In First Out) ilkesine göre çalışmasından kaynaklanmaktadır. CPU tarafından verilerin yığın alanına konulması **Push,** alandan alınması **Pop** olarak tanımlanır. Bu ekleme ve çıkarma işlemlerinde bellek adresleri değişmez. Bunun yerine yığın işaretçisi (Stack Pointer – SP), aşağı yukarı hareket ederek ilgili veriye erişilir veya yeni veri eklenir.

Yığın, dinamik değişkenleri saklamanın yanında yordam çağrıları yaparken geri dönüş adresini saklamak, yerel değişkenleri depolamak, yordamlara parametre yollamak için de kullanılır. Yığın bellek alanı, program çalışmaya başladığı anda belirlenir ve daha sonra bu alanın boyutu değiştirilemez. Bu alan işletim sistemi tarafından genellikle kısıtlı şekilde belirlendiği için yığın üzerindeki çoğu ekleme işlemlerinde hafıza birimi taşması sorunu yaşanabilmektedir (stack overflow).

### Heap

İşletim sistemi tarafından programcının yönetimine bırakılmış, daha geniş alanlı ve kalıcı bir alandır. Genellikle ne kadar yer kaplayacağı belli olmayan değişkenlerin, nesnelerin veya büyük verilerin geçici olarak saklanması için kullanılır. Yığın bölgesinin tersine çalışma ortamı (Common Language Runtime - CLR), heap alanında ne kadar yer gerektiğini tayin eder. Heap alanının okunması veya yazması yığın alanı kadar hızlı değildir.



Bir süreç (process) başladığı anda boş bir bellek yeri bu süreç için ayrılır. Buna “Managed Heap (Heap Bellek)” denir. İşaretçiler aracılığıyla heap bellekten nesnelerin yerleri tutulur. Çalışma zamanında oluşturulan nesneler uygulama tarafından ihtiyaç duyulmadığı zamanlarda heap bellekten temizlenir. Bu işlem için Garbage Collector mekanizması kullanılır. Uygulama new operatörünü çalıştırdığı zaman heap belleğe gider ve yeterli bir yer olup olmadığına bakar. Yeterli yer varsa pointer heap bellekteki bu yeri gösterir. Nesnenin constructor metodu çalıştırılır ve adres döndürülür. İşte bu anda garbage collector devreye girer ve uygulamanın ihtiyaç duymadığı tüm objeler heap bellekten temizlenir. Temizlenecek obje yoksa ve yeterli yer hala sağlanmamışsa new operatörü “OutOfMemoryException” uyarısını verir.

Her uygulamanın başlarken bellekte bir yer ayırdığından bahsetmiştik. Burayı gösteren ilk adres uygulamanın başlangıç noktasıdır. Bir uygulama için birden çok başlangıç noktası mevcuttur. Static veriler için bir başlangıç noktası, CPU registerları için bir başlangıç noktası gibi.

Garbage collector, heap bellekteki her şeyi temizlenecek nesneler olarak algılar ve bir başlangıç noktasından başlayarak tüm nesneleri kontrol eder ve graph oluşturur. Bu başlangıç noktasındaki tüm nesneleri kontrol edip **graph** oluşturduktan sonra diğer başlangıç noktasına geçer ve oradaki tüm nesneleri de kontrol eder.

Garbage collector çok akıllıca bir algoritmaya sahiptir. Örneğin herhangi bir nesne iki başlangıç noktasında da var. Garbage collector bu nesneye geldiği zaman bu nesneyi zaten graph içerisine eklemiş olduğunu anlar ve bu nesnesin altındaki nesneleri kontrol etmez. Böylece kendisinin daha hızlı çalışmasını sağlar ve sonsuz döngüye girmesini engeller. Tüm başlangıç noktalarını ve nesneleri kontrol ettikten sonra oluşan graphı lineer şekilde kontrol ederek tek tek heap belleğe yeniden yerleştirir. Aralıklı olanları varsa bu aralığı da kaldırır. Nesnelerin yeni adresleri olmuşsa bu adresleri gösteren pointerların yeni adresleri göstermesini de sağlar. İşlem sonunda gereksiz olan nesneler heap bellekten temizlenmiş olur.

Garbage collector bu mantık ile bazı kurallar tanımlar:

* Yeni oluşturulan nesnelerin yaşam süreleri daha kısadır.
* Eskiden oluşturulmuş nesnelerin yaşam süreleri daha uzundur.
* Yeni oluşturulmuş nesneler birbirleri ile daha sıkı bir ilişki içindedir ve kullanım zamanları çok yakındır.
* Heap bellekteki bir bölüme ulaşmak, tüm heap belleği dolaşmaktan daha hızlıdır.

Kısaca çalışma mantığı:

* Uygulama çalıştırıldığında heap bellek boş olarak oluşturulur.
* Heap belleğin dolu olduğu varsayılır ise garbage collector gelip gereksiz nesneleri temizler. Bu ilk temizlemede temizlenmeyen yani sağ kalan tüm nesneleri “Generation 0” olarak tanımlar.
* Heap belleğin tekrardan dolması durumunda yine garbage collector çalıştırılır. Generation 0’dan sağ kalanları Generation 1 olarak tanımlar ve yenilerden sağ kalanları da Generation 0 olarak tanımlar.
* GC, çalıştırıldığı zaman sadece belli bir bölgeyi tarama işlemi yapabilir. Yani çalıştırıldığında tüm heap belleğini kontrol etmek yerine ilk önce Generation 0’a bakar.

Bazen uygulama kapatıldığı halde heap bellekteki bazı nesnelere erişim sağlanılabilir. Bu da, bu nesnenin bir “thread” ya da başka bir süreç tarafından kullanılması ile olabilir ya da uygulama kapanırken bazı nesneleri oluşturabilir. Bu nesnelerin “finalize” metotları çalıştırılmamış olacak ve istenilen temizleme işlemi gerçekleşmeyecektir.

### Dispose Metodu

Bu metod çağırıldığı zaman nesnenin tutmuş olduğu kaynaklar serbest bırakılır. Bu metodun birden fazla çağırılması “exception” hatasını döndürür.

Örneğin, zaten kaynakları serbest bırakılmıi bir nesneye tekrardan dispose metodunu kullandırmak gibi.

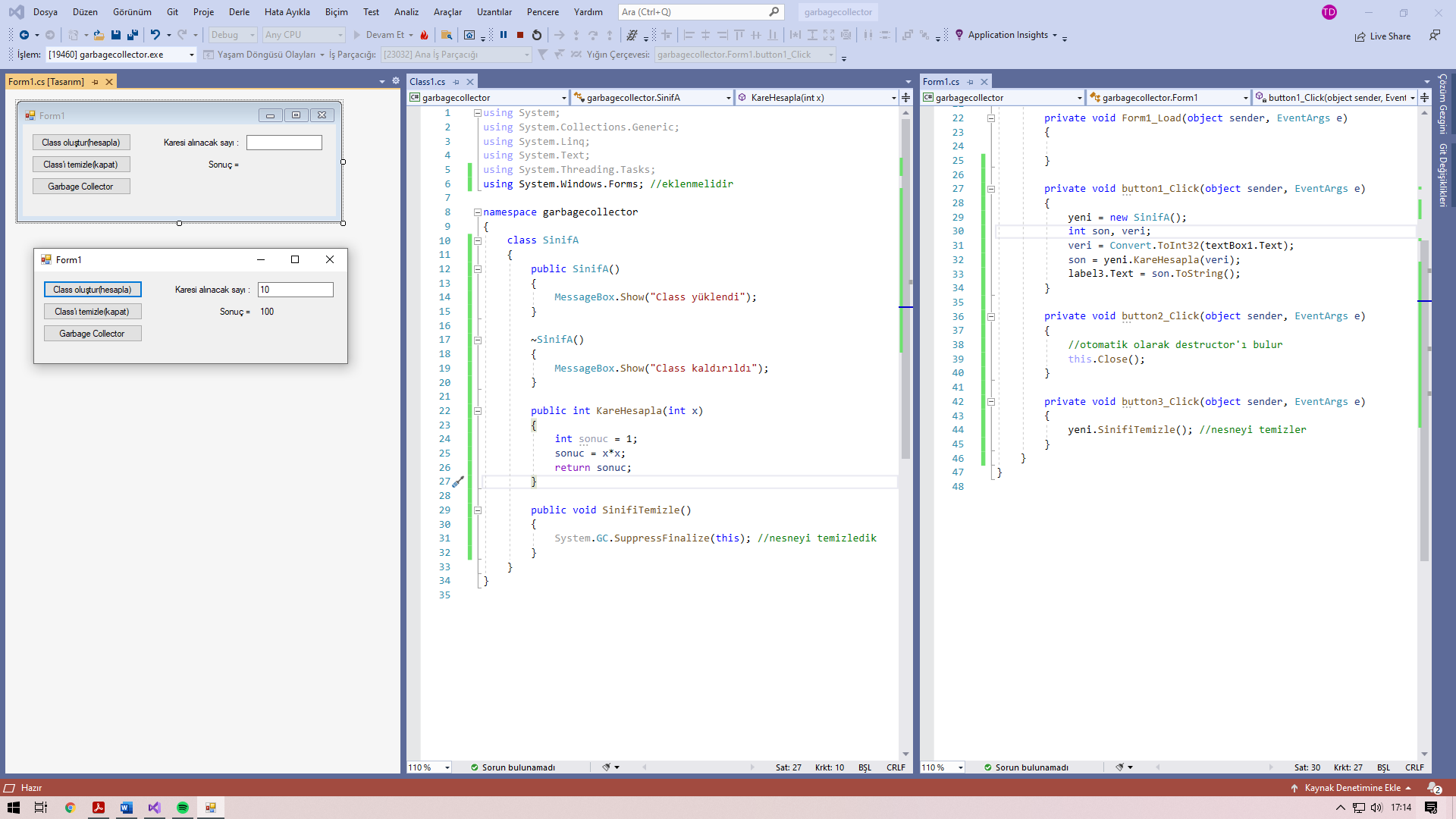
Program yazmada dikkat edilmesi gereken konulardan biri belleği en uygun şekilde kullanmaktır. Özellikle kurumsal uygulamalarda kullanılacak programların kaynakları ekonomik ve ergonomik kullanması her zaman tercih sebebi olmuştur. Bir programcı başta bellek olmak üzere sistem kaynaklarını daima iradeli kullanmalı ve meşgul ettiği alanı iş bittikten sonra serbest bırakmalıdır.

Yüksek düzeyli dillerde de sınıflardan üretilen nesnelerin işlemleri bittiğinde serbest bırakmaları gereklidir. Fakat bu süreçte her zaman bu kurala uyulmadığı için programlardan istenmeyen durumlar oluşabilmektedir.

* Bazı programcılar kullandıkları nesneleri silmedikleri için bellek sarfiyatı veya “memory leak” olarak tanımlanan bellek sızıntısı problemi yaşanabilmektedir.
* Bazı programcılar da nesneleri iki defa silmete çalıştıkları için programda hatalar oluşmaktadır.
* Başka bir programcı grubu da daha önce serbest bırakılmış nesneye erişmeye kalkışmaktadır.

## Garbage Collector

Bellekte oluşan atıl nesneleri (çöpleri) temizlemek için Garbage Collector sınıfı kullanılır. Class içerisindeki prosedür ile aynı sınıfın temizlenmesi, her kullanıcı tanımlı sınıfa eklenmesi çok önemlidir. Bununla ilgili örnek aşağıdadır.

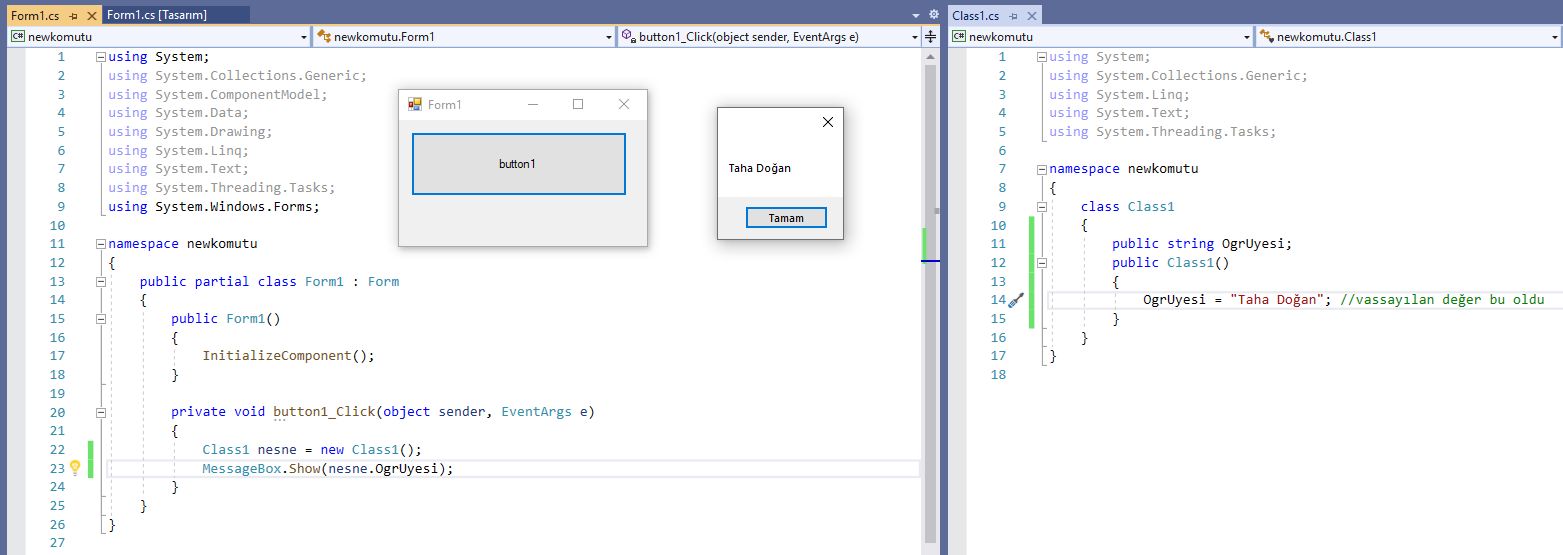


GC.Suppress.Finalize metodu ile aktif class bellekten temizlenir.

Yukarıda kodları verilmiş olan uygulama çalıştırılıp **Class oluştur** butonuna basıldığında girilen sayının karesi hesaplanarak label3 kontrolünde sonuç gösterilir. **Class’ı temizle** butonuna basıldıktan sonra **Garbage Collector** düğmesine basılır ise class bellekten alıtıyor mesajı göründükten sonra uygulama kapanır.

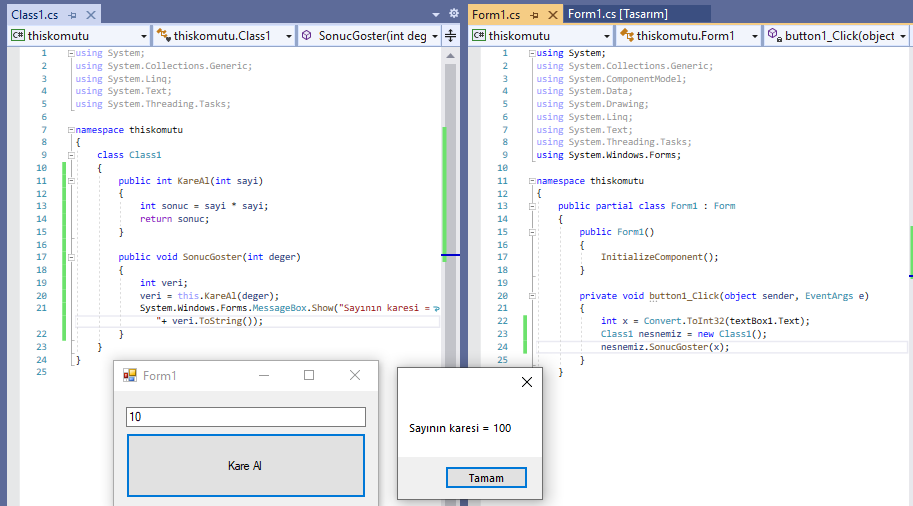
# New Komutu

Nesnel uygulamalar için sınıftan türetilen yavru değişkene bellekte yer ayırmak için kullanılan komuttur. Önemli özelliği de o sınıfa ait constructor kod bloğunu otomatik olarak işletmesidir.



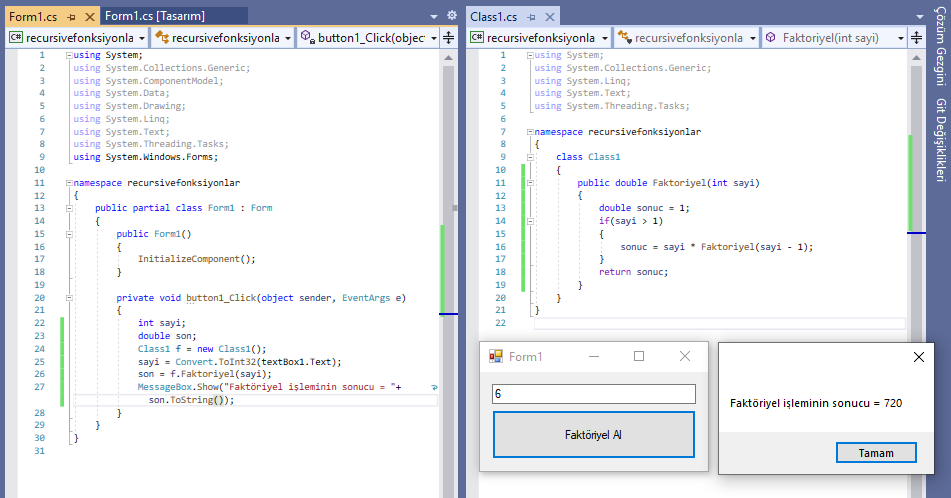
# This Komutu

Kullanıldığı sınıfta, sınıfa ait metod ve özellikleri işletmek-işaret etmek için kullanılan komuttur. Windows Forms uygulamalarında bulunan this komutu, kod hangi formda ise o formu temsil etmektedir.



SonucGoster metodu üzerinden işlem yaptık çünkü zaten this.KareAl kodu ile KareAl metodunu SonucGoster metodunun içerisinde çağırmış gibi olduk.

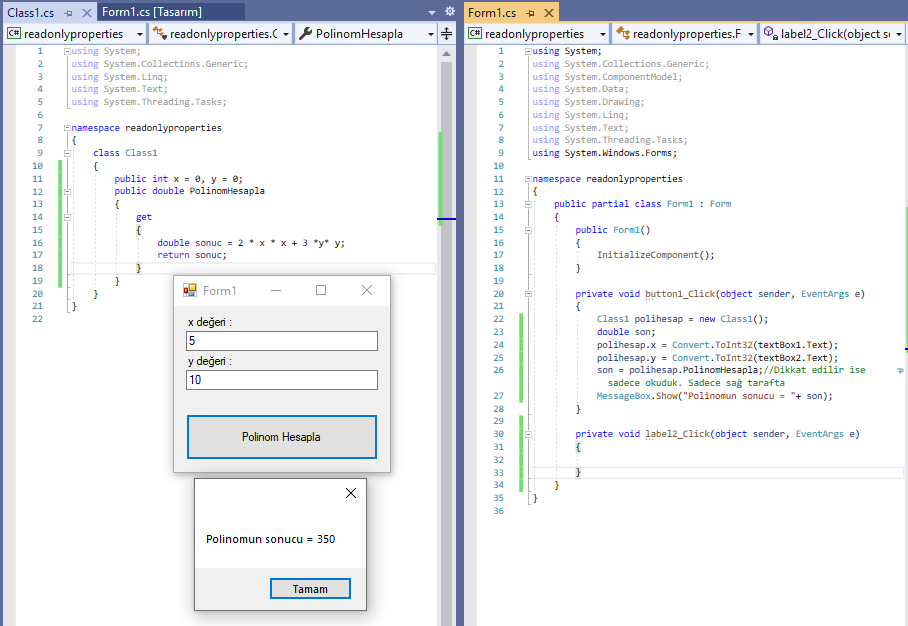
# Recursive Fonksiyonlar

Fonksiyonun kendi kod bloğu içerisinde kendi adını çağırarak iç içe kendini tekrarlaması-özyinemeli olması durumundaki fonksiyonlara denir. 

# Sınıflara Özellik Eklenmesi

Properties (özellikler) ifadesi görsel dillerde sınıflara ait oluşturulan en popüler yapılardır. Özellikle Windows Forms uygulamalarında çok kullanılan kontrollere ait birçok metot, propertie olarak ifade edilir. Bu da programcıların konuşma diline yakın işlem yapmasını sağlar. Özellik değerlerini muhakkak arka planda bir değişkende tutmak gereklidir. Amaç bu değerle ilgili bir kod bloğu oluşturularak sonucu hiçbir metodu çağırmadan kullanıcıya göstermek olacaktır.

## ReadOnly Properties Tanımlamak

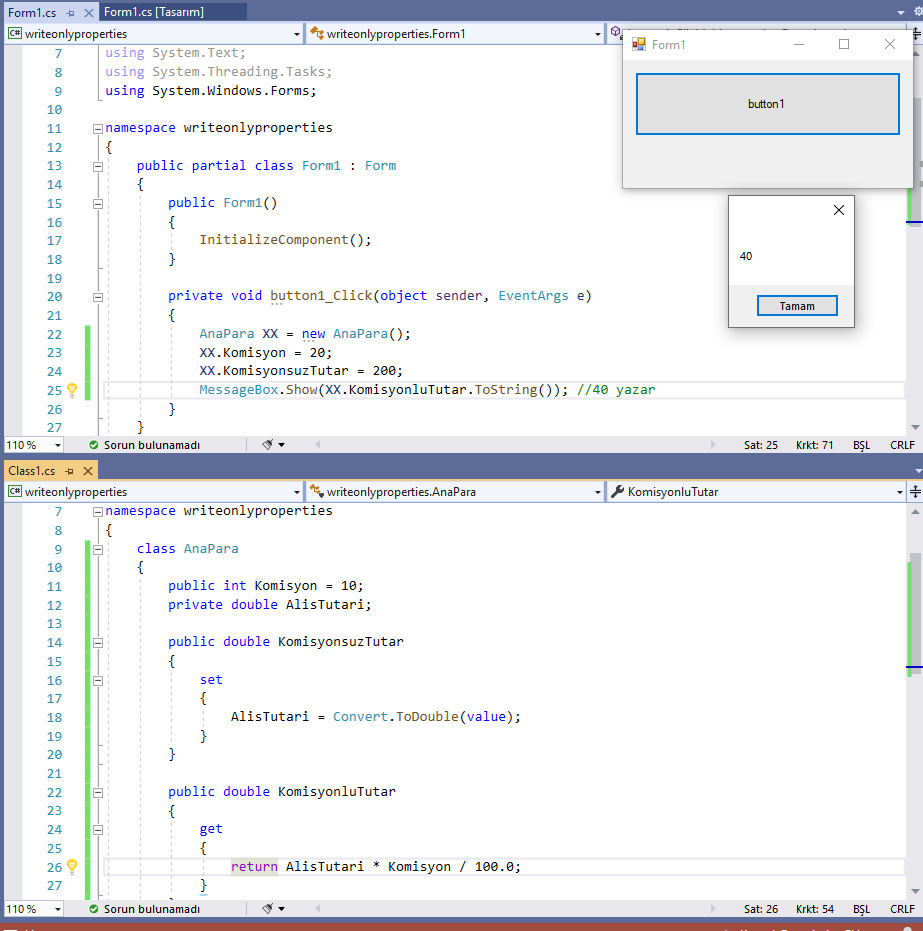
Bu tür özellikler sadece eşittir işaretinin sağında kullanılabilir. Yani değerleri başka değişkenlere aktarılabilir ancak asla değiştirilemezler. Sadece okunmak için vardır. 

Yukarıdaki örnekte uygulama için sadece get bloğu bulunan PolinomHesapla isimli bir özellik tanımlanmıştır. Bu sayede Class1 isimli sınıftan oluşturulacak olan yavru bir değişkenle PolinomHesapla isimli özellik istenilen double tipli bir değişkene aktarılabilir.

## WriteOnly Properties Tanımlamak

Bu tür özellikler sadece eşittir işaretinin solunda yer alabilir. Yani değerleri başka değişkenlere aktarılabilir fakat asla içeriği öğrenilemez.

Bu işlemde set bloğu bulunmaktadır. Bu bloğa yazılan kod sayesinde ilgili değişkene gerekli olan değerler kolayca gönderilebilir. Bloğun önemli özelliği value isimli parametredir. Bu parametredeki mantık eşittir işaretinin solunda kullanılacağı için atanan sağdaki değer direkt value’da tutulacaktır.

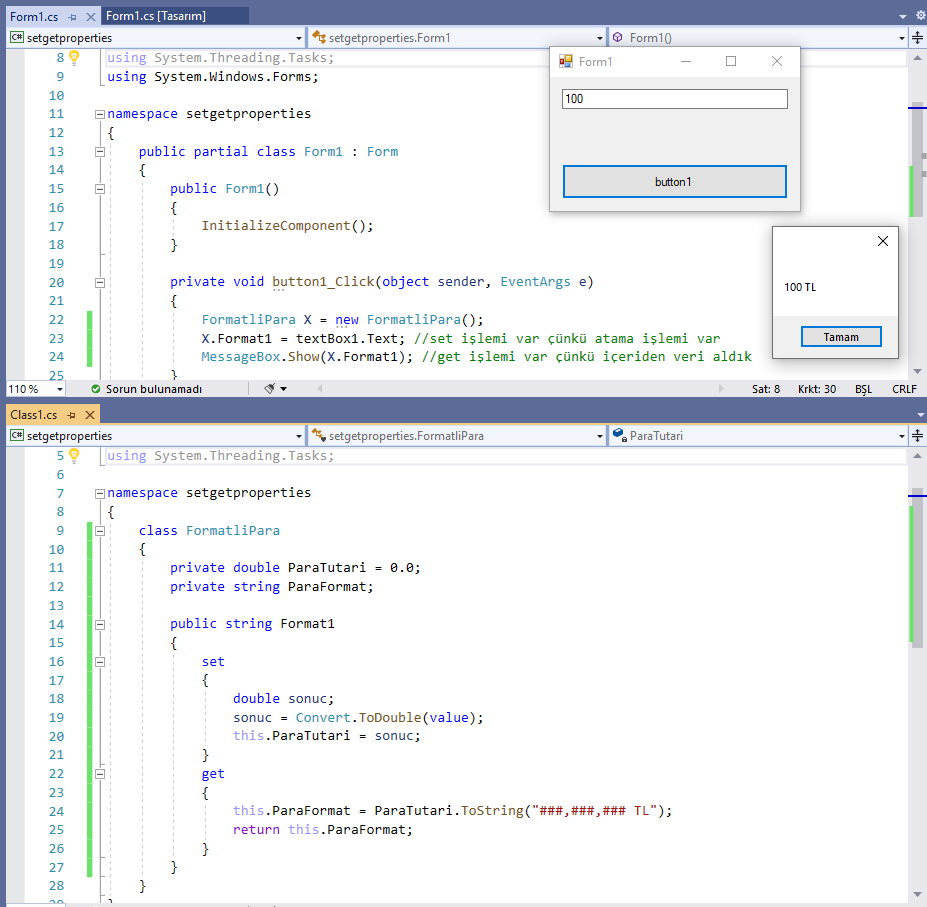


Yukarıdaki örnekte yer alan get ve set bloklarına dikkat edilmelidir. KomisyonsuzTutar özelliği set ifadesi ile tanımlandığı için bu blok içinde değer gönderme yapılabilir ve eşittir ifadesiin sağındaki değer set edilecek veri olarak kullanılır. Aynı benzer mantık KomisyonluTutar için de geçerlidir. Ancak bu özellik get bloğu ile tanımlandığı için bu özelliğe değer atama yapılamaz. Sadece bu özelliğin sahip olduğu değer başka bir değişkene aktarılabilir. Bu doğrultuda “int son = XX.KomiksyonsuzTutar” hatalı bir yazım olacaktır. Çünkü set ile tanımlı özelliklerin okunması yapılamaz. Bir değerin bir değişkene ataması yapılması için öncelikle okunması gereklidir.

## Yazılabilen ve Okunabilen Özellikler Tanımalamak

Hem set hem de get bloğunu **aynı anda** içeren özellik tanımlalarıdır. Yazma ve okuma işlemleri bir arada yapılabilir. Yani eşittir işaretinin hem sağında hem de solunda kullanılabilen özelliklerdir.

Bir başka bakış açısı ile, dışarıdan veri geldiğinde veya dışarıdan veri çekildiğinde hangi blokların çalışacaklarını ayırıyoruz ve bu işlemi get-set blokları ile yapıyoruz.



Yukarıdaki kod bloğunda string tipli bir özellik olan Format1 özelliğineTextBox kontrolüne ait bir içerik aktarılmaktadır. Özelliğe ait set bloğu çalışarak değer öncelikle double tipe dönüştürülmekte ardından ParaTutari isimli değişkene aktarılmaktadır. Özelliğe ait set bloğu çalışarak değer önce double tipine dönüştürülmekte, ardından ParaTutari isimli değişkene aktarılmaktadır. Form içinde messagebox ile yazdırma işleminde ise get bloğundaki kod işletilerek ParaTutari isimli double tip değişkenin değeri string tipine dönüştürülmekte ve formun başlığında formatlı olarak para bişiminde yazılmaktadır.

Buraya kadar anlatılanlar doğrultusunda konunun daha iyi anlaşılabilmesi için aşağıdaki örnek adım adım anlatılmıştır. Araba sürmek ve gaz pedalına basarak arabayı hızlandırmak istediği farzedilmektedir. Bunun yapılabilmesi için öncelikle birilerinin bu arabayı tasarlaması gerekir. Bu tasarım ev-inşaat planlarına benzeyen bir takım mühendislik çizimleridir aslında. Bu çizimler arabayı hızlandıracak olan gaz pedalını da içerir. Aslında arabayı hızlandıran geri planda karmaşık ve büyük bir mekanizma vardır ancak insanlar bu mekanizmadan habersiz, sadece gaz pedalını kullanarak arabayı hızlandırır. Aynı şekilde fren pedalı da arabayı yavaşlatan geri plandaki büyük ve karmaşık mekanizmanın detaylarını gizlerken, kullanımını kişilere sunar. Böylece motor vb. hakkında hiçbir bilgisi olmayan insanlar bile bu arabayı kullanabilir. Ancak arabanın kağıt üzerindeki mühendislik çizimleri gözardı edilemez. Öncelikle arabanın bu çizimlere göre inşa edilmesi gereklidir. Yapımı tamamlanmış bir araba ise elbette çizimlerdeki gibi bir hız pedalını da içerecektir. Yine de arabanın hızlanması için tek başına pedalın olması yeterli değil, sürücünün de bu pedalı kullanması gereklidir.

# Kodların Kısa Uygulanışları

Şimdi yukarıdaki örnek üzerinden nesneye yönelik programlama kavramlarını açıklamaya çalışalım. Programlamada metodlar belli görevlerin gerçekleşmesi için çağırıldığını şimdiye kadar olan süreçte öğrendik. Örneğin bir banka hesabını temsil eden bir sınıfta para yatırma, para çekme ve bakiye öğrenme gibi görevleri yerine getiren metodlar olabilir. Başka bir ifade ile metodlar sınıflara aittir.

Aşağıdaki Araba sınıfı incelenecek olursa;

public class Araba

{

private int hiz; //bu değişken sadece sınıf içerisinden erişilebilir.

public void hizlan()

{

hiz++;

}

}

Çizimlerden bir araba inşa edilmedikçe arabanın kullanılması mümkün olmadığı gibi tanımlanan bir sınıfta belirlenen görevlerin icra edilebilmesi için de o sınıfın bir nesnesinin inşa edilmesi gerekir. Yukarıdaki tanımlama arabanın konsept hali gibi düşünülür ise main bloğu vb. program kullanılırken-işletilirken yavru bir değişken ile tanımlanması da arabanın seri üretime geçmesidir.

public class Main{

public static void main(string [] args){

Araba anadol = new Araba(); //yavru değişken oluşturuldu

anadol.hizlan(); //hızlanması için komut verildi

}

}

Bu kod kesiminde ne Araba sınıfında hiz isimli bir değişken olduğu, ne de hizlan() metodunun içi bilinmemektedir. Aslında bilinmesi de gerekli değildir. Tek istenilen hızlan metodunun arabayı hızlandırmasıdır. Araba tasarımında bir eksiklik bulunmaktadır. Araba hızlanabilmekte ama anlık hızının ne olduğu bilinmemektedir. Tasarım biraz daha gelistirilir ise;

public class Araba

{

private int hiz;

public void hizlan()

{

hiz++;

}

public void hizGoster()

{

Console.Writeline("Hız :" + hiz);

}

}

Araba sınıfına eklenen yeni yeteneği bir anadol isimli nesne üzerinde test edelim:

public class Main{

public static void main(string [] args){

Araba anadol = new Araba(); //yavru değişken oluşturuldu

anadol.hizlan(); //hızlanması için komut verildi

anadol.hizGoster(); //hız gösterildi

}

}

Kodun çıktısı 1 olacaktır.

Diyelim ki anlık hızın 100km/h‘i geçip geçmediğinin kontrolü yapılmak istiyoruz. hizGoster() metodu hızı yalnızca ekrana bastırdığı için bu değeri kodun içinde integer olarak elde edip kullanma durumu söz konusu değildir. Bunu engellemek için tasarımda ufak bir değişikliğin yapılması gerekir. hizGoster() metodu hızı ekrana yazmakla beraber değerini de döndürsün. Aşağıdaki kodda hizGoster() metodunda yapılan değişikliğe dikkat edin.

public class Araba

{

private int hiz;

public void hizlan()

{

hiz++;

}

public int hizGoster()

{

Console.Writeline("Hız :" + hiz);

return hiz;

}

}

Artık hiz değeri kod içerisinde istenildiği gibi kullanabiliriz.

public class Main{

public static void main(string [] args){

Araba anadol = new Araba();

anadol.hizlan();

int hiz = anadol.hizGoster();

if(hiz > 100)

{

Console.Writeline("Aşırı hız!");

anadol.hizGoster();

}

}

}

Ancak bu tasarım yine de eksiktir. Bu şekilde hizlan() metodunu hızı elde etmek için her çağırdığımda ekrana da bir şeyler yazılacaktır. Oysa zaman zaman ekrana yazmak, zaman zaman da yalnızca hız değerini (value) elde etmek ve belli işlemlerin yapılması istenmektedir. Bunu yapmanın tek yolu iki işlevi birbirinden ayırmak, yani 2 ayrı metod yazmaktır.

public class Araba

{

private int hiz;

public void hizlan()

{

hiz++;

}

public void hizGoster()

{

Console.Writeline("Hız :" + hiz);

}

public int getHiz()

{

return hiz;

}

}

Araba sınıfının en son halinin gerçekleştirilmesi:

public class Main{

public static void main(string [] args){

Araba anadol = new Araba();

anadol.hizlan();

anadol.hizGoster();

if(anadol.getHiz()>100)

{

Console.Writeline("Aşırı hız!");

}

}

}

Araba tasarımına ihtiyaca yönelik pek çok değişken ve metod eklenebilir. Örneğin renk, anlık yakıt miktarı, fiyat gibi birçok değişkenlerden yanı sıra yavasla(), yakitAl(), vb. metodlar da sınıfa eklenebilir. Bu değişkenlerin sınıfın her bir nesnesi için ayrı ayrı olacağını, başka bir ifade ile her Araba nesnesinin kendi renk, hız fiyat vb. bilgilerini tutacağı unutulmamalıdır. Örneğin:

public class Main{

public static void main(string [] args){

Araba anadol = new Araba();

anadol.hizlan();

anadol.hizGoster();

Araba bmw = new Araba();

bmw.hizlan();

bmw.hizlan();

bmw.hizGoster();

}

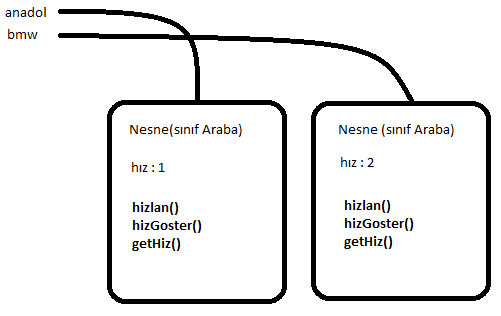
}

Programın çıktısı aşağıdaki gibi olur.

Hız :1

Hız :2

Anadol ve BMW nesnelerinin kendilerine ait hız değerleri bulunmaktadır. Bellekte (Memory/RAM) nesnelerin durumu aşağıdaki gibidir.



Şu ana kadar tasarlanan araba; hızlanma, hız gösterme gibi yeteneklere sahipti. Ancak arabayı yavaşlatmak mümkün değildi. Arabanın yavaşlamasını sağlayalım.

public class Araba

{

private int hiz;

public void hizlan()

{

hiz++;

}

public void hizGoster()

{

Console.Writeline("Hız :" + hiz);

}

public int getHiz()

{

return hiz;

}

public void yavasla(int nekadar)

{

hiz = hiz - nekadar;

}

}

public class Main{

public static void main(string [] args){

Araba anadol = new Araba();

anadol.hizlan();

anadol.hizlan();

anadol.hizlan();

anadol.hizGoster();

anadol.yavasla(2);

anadol.hizGoster();

}

}

Programın çıktısı aşağıdaki gibi olur.

Hız :3

Hız :1

## Public ve Private Erişim Belirteçleri (Access Modifiers)

Dikkat edilirse Araba sınıfındaki hız değişkeninin başında private, Araba sınıfının başında da public anahtar kelimeleri mevcuttur. Bu anahtar kelimeye erişim belirteçleri denmektedir. Yani bir değişkene nereden, nasıl erişilebileceğini belirten anahtar sözcüklerdir. Araba sınıfı public olduğundan her yerden erişilebilir bir sınıftır. Başka sınıflarda bu sınıf istenildiği gibi kullanılabilir ve araba nesneleri üretilebilir. Hız değişkeni ise private yani sınıfa özel bir değişkendir. Araba sınıfının metodları bu değişkene diledikleri gibi erişebilirken, sınıfın dışından bu değişkene erişmek mümkün değildir. Örneğin;

anadol.hiz = 5;

anadol.hiz = 200;

anadol.hiz = 100;

şeklindeki kullanımlar yanlış kullanımdır, geçersizdir.

Private tanımlama ile hiz değişkeni dışarıdan her türlü müdahaleye kapanmıştır. Arabayı ancak hızlan metodu ile birer birer hızlandırabiliriz. Tasarlanmış olan Araba sınıfından nesne oluşturulup kullanacak olan herkes nesneyi tasarıma uygun kullanmak zorundadır. Başka biri ifade ile benim sınıfım, benim kurallarım yapısı bunu ifade eder.

## Set ve Get Metodları

Yukarıda anlatılan sebeplerden dolayı programcılar, varsayılan olarak her değişkenin private tanımlanması gerektiğini ve bu değişkenlere erişim gerekiyorsa bunun yalnızca set ve get metotları ile sağlanmasını tavsiye eder. Şimdi tasarımı yapılan sınıfa bir değişken daha eklensin ve değişkene erişim sağlayacak set ve get metodları yazılsın.

public class Araba

{

private int hiz;

private string marka;

public void setMarka(string Marka)

{

this.marka = Marka;

}

public string getMarka()

{

return this.marka;

}

}

Bu kodu işletelim.

public class Main{

public static void main(string [] args){

Araba anadol = new Araba();

anadol.setMarka("Anadol");

Araba bmw = new Araba();

bmw.setMarka("BMW");

Console.Writeline("1. Araba : "+anadol.getMarka());

Console.Writeline("2. Araba : " + bmw.getMarka());

}

}

Set metodları **her zaman** ilgili değişken ile aynı tipte bir parametre aldığını ve dönüş tipinin **genelde** void olduğunu, get metodlarının ise **genelde** parametre almadığını ve dönüş tipinin **her zaman** igili değişken ile aynı olduğu unutulmamalıdır.

Dikkat edilir ise sınıf metodlarının başında da public anahtar kelimesi mevcuttur. Elbette buradan metodların da private olabileceği anlamı çıkmaktadır. Bu durumda dışarıdan erişilemeyecek metod ne işe yarar sorusu akla gelebilir. Ancak unutulmamalıdır ki sadece sınıf içerisinde kullanılacak, yani sınıf içi hizmet verecek metodlar da olabilir.

public class Araba

{

private int hiz;

private string marka;

private bool frenGecerliMi(int fren)

{

if(fren<0 )

{

return false;

}

else

{

return true;

}

}

public void yavasla(int miktar)

{

if(!frenGecerliMi(miktar))

{

return;

}

if(miktar < hiz)

{

hiz = hiz - miktar;

}

else

{

hiz = 0;

}

}

}

frenGecerliMi(int fren) metodu private olduğundan aşağıdaki gibi bir kod geçersiz olacaktır.

anadol.frenGecerliMi(-10)

Yukarıdaki kod yazılırken bile fark edilmelidir ki program böyle bir metodu görmez bile.

## Nesneleri Yapıcı (Constructor) Metodlar ile İlklendirme (Initialize)

Araba sınıfı son hali itibariyle eper ilerlemiş olsa da hala eksikler ve tutarsızlıklar mevcuttur. Örneğin sınıfın mevcut hali ile bir araba nesnesinin markasını belirtmeden oluşturabilir, verilen bir marka daha sonra istendiği gibi değiştirilebilir.

public class Main{

public static void main(string [] args){

Araba anadol = new Araba();

anadol.setMarka("Anadol");

Console.Writeline("Marka :"+ anadol.getMarka());

anadol.setMarka("BMW");

Console.Writeline("Marka :"+ anadol.getMarka());

}

}

Anadol marka oluşturulan araba bir satır kod ile BMW yapılmıştır. Bir araba nesnesi ilk oluşturulduğu anda markasını belirtmek zorunlu olsa, sonra da bu markanın bir daha değiştirilememesi daha uygun olurdu.

Yapıcı metodlar yalnızca nesne oluşturulurken bir defalığına çağırılan özel metodlardır. İsmiyle eşdeğer bu metod nesne oluşturulurken çağırılır ve o anda yapılması gereken görevleri icra eder. Her zaman yapıcı metod yazılması zorunlu değildir. Yukarıda belirtilenleri gerçekleştirecek olan bir yapıcı sınıf oluşturalım.

public class Araba

{

private int hiz;

private string marka;

public Araba(string marka)

{

this.marka = marka; //bu sınıf içerisindeki marka = parametre olan marka

}

public string getMarka()

{

return marka;

}

}

Aşağıdaki gibi geçersiz bir kodun yazılmamasına dikkat edilmelidir.

Araba anadol = new Araba();// geçersiz çünkü parametresi yok

anadol.setMarka("Anadol");//artık böyle bir metod yok

Doğru olan durum aşağıdaki gibidir.

Araba anadol = new Araba("Anadol"); //ilk ve son defa nesnenin markası belirlendi

Console.Writeline("Marka :"+anadol.getMarka()); //markayı her zaman öğrenebiliriz

Böylelikle bir araba nesnesi oluşturulurken markanın belirtilmesi zorunludur. Bu marka bir daha değiştirilemez çünkü setMarka() metodu artık yoktur. Ama getMarka() metodu ile her zaman arabanın markası öğrenilebilir.

## This Anahtar Sözcüğü

Sınıf kodu içerisinde sınıfın kendisini ifade edebilmek için this anahtar sözcüğü kullanılır. Aşağıdaki kodda Araba sınıfının yapıcı metodunda marka isimli iki adet değişken vardır. Bunlardan biri metoda parametre olarak gönderilen string’i, diğeri de sınıfın kendi string değişkenini temsil ediyor.

public class Araba

{

private int hiz;

private string marka;

public Araba(string marka)

{

this.marka = marka; //bu sınıf içerisindeki marka = parametre olan marka

}

}

Yukarıdaki kodda “this.marka = marka” yerine “marka = marka” yazılsaydı, sol taraftaki marka sınıfın değişkeni olan marka mı yoksa parametre olan marka mı olduğu ayırt edilemezdi. Bu yüzden this sözcüğü ile açıkça sol taraftaki markanın sınıfın bir değişkeni olduğu belirtilmiş olur. Eğer this anahtar sözcüğü kullanılmak istenmez ise değişkenlerin adları da değiştirilebilir ancak büyük projelerde teferruatlı bir yöntem olur. Ancak bu yöntem okunurluğu düşürür.