Bölüm 04

Kurucular ve Yokediciler

(Constructors and Destructors)

Kurucu Nedir?

C# dilinde iki tür kurucu vardır: dinamik (instance) kurucular, statik kurucular. Statik kurucu, öteki statik öğeler gibi davranır, ilk çağrıda sınıfa ait nesneyi yaratır. Kurucu deyince, genellikle dinamik (instance) kurucu kastedilir. Microsoft, C# kurucusunu şöyle tanımlar: *"Sınıfa ait bir nesne yaratan sınıf öğesidir."* Bu tanım güzeldir ama yeni başlayanların anlaması zor olabilir. Ama, aşağıdaki açıklamalar bu tanımı aydınlığa kavuşturacaktır.

new operatörü

Geçen bölümlerde bir sınıftan nesneler yaratmayı öğrendik, nesnelerle ilgili bazı işlemler yaptık. Bu bölümde, nesne yaratıcıların hünerlerini göreceğiz. Her zaman olduğu gibi, işin kuramsal yanını geriye bırakıp, pratik uygulamalarla kavramları açıklama yoluna gideceğiz.

Önce aşağıdaki programı derlemeyi deneyelim. Program `Ev` sınıfı içindeki `Yaz()` metodunu çağırmaktadır.

Kurucular01.cs

```
using System;
```

```
namespace Sınıflar
{
    class Uygulama
    {
        static public void Main()
        {
            Yaz();
        }
    }

    class Ev
    {
        public int kapıNo = 32;
        public string sokakAdı = "Papatya";
        public void Yaz()
        {
            Console.WriteLine("Ev {1} sokakta {0} numaralıdır.", kapıNo,
sokakAdı);
        }
    }
}
```

Program derlenirken şu hata iletisini verir.

```
Error    1    The name 'Yaz' does not exist in the current context ...
```

Buraya kadar öğrendiklerimize göre, hatanın ne olduğunu hemen anlıyoruz. `Yaz()` metodu, onu çağıran `Main()` 'in içinde bulunduğu `Uygulamalar` sınıfında değil, onun dışında olan `Ev` sınıfındadır. `Main()` metodu onu görememektedir.

`Main()` metoduna `Yaz()` metodunun `Ev` sınıfı içinde olduğunu göstermeyi deneyebiliriz. Bunun için `Main()` metodunun gövdesindeki `Yaz()` deyimi yerine

```
Ev.Yaz();
```

deyimini koyabiliriz.

Ancak, derleyicimiz, bu kez başka bir hata iletisi verecektir.

```
Error    1    An object reference is required for the non-static field, method, or property 'Sınıflar.Ev.Yaz()' ...
```

Şimdi hatayı daha iyi algılıyoruz. Derleyicimiz `Yaz()` metodunun içinde olduğu sınıfı değil, o sınıfa ait bir nesnenin işaretçisini (object reference) istemektedir. Bu isteğini de kolayca karşılayabiliriz. `yazlıkEv` adlı bir işaretçi bildirimi yapalım. Bu bildirimi yapmak için `Main()` 'in gövdesine şu deyimleri ekleyelim:

```
            Ev yazlıkEv;
            yazlıkEv.Yaz();
```

Bu kez, derleyicimiz şu hatayı iletecektir:

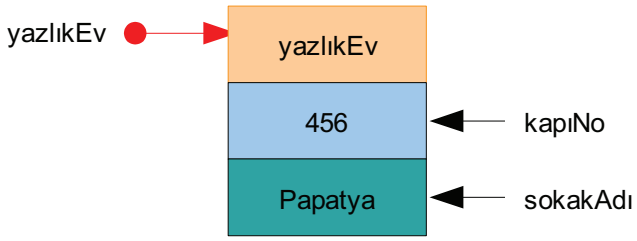
```
Error    1    Use of unassigned local variable 'yazlıkEv'    ...
```

Derleyicimizin ne dediğini artık kolayca anlıyoruz. `yazlıkEv` adlı bir işaretçi bildirimini yaptık, ama o `null` işaret ediyor; çünkü, onun işaret edeceği bir nesne yaratmadık.

Öyleyse, `new` oparatörünü kullanarak `yazlıkEv` referansının işaret edeceği bir nesne yaratalım.

```
            Ev yazlıkEv;
            yazlıkEv = new  Ev() ;
            yazlıkEv.Yaz();
```

Bu bildirimlerin etkisini aşağıdaki şekille temsil edebiliriz.



`Ev` sınıfında static olmayan `kapıNo` ve `sokakAdı` değişkenlerine verdiğimiz değerlerin, `yazlıkEv` nesnesine aynen gittiğine dikkat ediniz. Başka nesneler yaratsak, bu değerler o nesnelere de gidecektir.

İpucu

Sınıf değişkenlerine (veri alanı, field) atanan değerler, sınıfa ait her nesneye aynen gider.

Artık, derleyicimizin her istediğini yerine getirmiş oluyoruz. Aşağıdaki programımızı derleyip koşturabiliriz.

Kurucular02.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static public void Main()
        {
            Ev yazlıkEv ;
            yazlıkEv = new Ev();
            yazlıkEv.Yaz();
        }
    }

    class Ev
    {
        public int kapıNo = 456;
        public string sokakAdı = "Papatya";
        public void Yaz()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Papatya Sokak, No: 456
```

Sınıf öğelerinin değerlerinin nesnelere aynen gittiğini söylemiştik. Program koşarken, o değerleri nesne içinde değiştirmek mümkündür; ama nesne içindeki değişiklik, sınıf tanımındaki değerleri etkilemez. Bunu aşağıdaki örnekten görebiliriz.

Kurucular03.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static public void Main()
        {
            Ev  yazlıkEv = new Ev();
            yazlıkEv.Yaz();

            yazlıkEv.kapıNo = 789;
            yazlıkEv.sokakAdı = "Sümbül";
            yazlıkEv.Yaz();

            Ev kışlıkEv = new Ev() ;
            kışlıkEv.Yaz();
        }
    }

    class Ev
    {
        public int kapıNo = 456;
        public string sokakAdı = "Papatya";
        public void Yaz()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Papatya Sokak, No: 456
ADRES: Sümbül Sokak, No: 789
ADRES: Papatya Sokak, No: 456
```

Bu çıktıyı çözümlemek için `Main()` 'in gövdesindeki deyimlere bakalım.

```
            Ev  yazlıkEv = new Ev();
            yazlıkEv.Yaz();
```

Bu iki deyim, `yazlıkEv` adlı nesneyi yaratıyor ve onun `Ev` sınıfından aldığı değişken değerlerini konsola yazıyor:

```
ADRES: Papatya Sokak, No: 456
```

```
            yazlıkEv.kapıNo = 789;
            yazlıkEv.sokakAdı = "Sümbül";
            yazlıkEv.Yaz();
```

Bu üç deyim, `yazlıkEv` adlı nesneye `Ev` sınıfından giren değişken değerlerini değiştirip konsola yazıyor:

```
ADRES: Sümbül Sokak, No: 789
```

```
            Ev kışlıkEv = new Ev() ;
            kışlıkEv.Yaz();
```

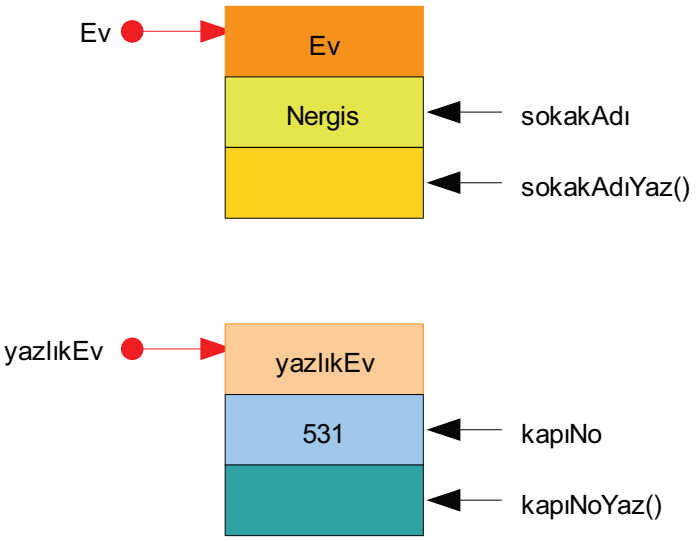
Bu son iki deyim, `kışlıkEv` adlı nesneyi yaratıyor ve onun `Ev` sınıfından aldığı değişken değerlerini konsola yazıyor.

Bu değerlerin sınıf değişkenlerinin değerleriyle aynı olduğunu görüyoruz. Bu demektir ki, ikinci grupta `yazlıkEv` nesnesi içinde yaptığımız değişiklikler sınıf değişkenlerine yansımıyor. Yansımış olsaydı, üçüncü grup çıktısı ikinci grup çıktısı ile aynı olurdu.

Static ve Dinamik Öğelere Erişim

Aşağıdaki programda `Main()` metodu `Ev` sınıfının iki metodunu çağırmaktadır. `KapıNoYaz()` metodu dinamiktir ve ona `yazlıkEv` nesnesi içinde erişilmektedir. Ama `SokakAdıYaz()` metodu static'tir; bir nesne yaratılmadan ona doğrudan erişilmektedir. Benzer şekilde, `Ev` sınıfı içinde `kapıNo` değişkeni dinamiktir, ona `yazlıkEv` nesnesi içinde erişilmektedir. Ama `sokakAdı` değişkeni static'tir; ona bir nesne yaratılmadan doğrudan erişilmektedir.

Dinamik değişken ve metotlar, sınıfa ait nesneler içinde bulunurlar. Dolayısıyla onlara nesne içinde erişilebilir. Static değişken ve metotlara, bir nesne yaratılmadan, doğrudan sınıfın adıyla erişilebilir.



Şekilde temsil edildiği gibi, `yazlıkEv` nesnesi içinde dinamik `kapıNo` değişkenine ve dinamik `KapıNoYaz()` metoduna erişilebilir. Static olan `sokakAdı` değişkenine ve `SokakAdıYaz()` metoduna nesne içinde değil, nesne dışında onlara ana bellekte ayrılan adreslerde erişilebilir. O adreslere ulaşmak için, referans olarak `Ev` kullanılır. Burada Ev, asıl sınıf değil, sözkonusu bellek adresini işaret eden referanstır.

Bu söylediklerimizi aşağıdaki programda görebilirsiniz.

Kurucular04.cs

```
using System;

namespace Kurucular
{
```

```
    class Uygulama
    {
        static public void Main()
        {
            Ev yazlıkEv = new Ev();
            yazlıkEv.KapıNoYaz();

            Ev.SokakAdıYaz();
        }
    }

    class Ev
    {
        public int kapıNo = 531;
        public static string sokakAdı = "Nergis";

        public void KapıNoYaz()
        {
            Console.WriteLine("Kapı No: {0} ", kapıNo);
        }

        public static void SokakAdıYaz()
        {
            Console.WriteLine("Sokak Adı: {0} ", sokakAdı);
        }
    }
}
```

Çıktı

```
Kapı No: 456
Sokak Adı: Papatya
```

Yukarıdaki programda static öğeleri dinamik, dinamik öğeleri static yaparak derleyicinin vereceği hata iletilerini görünüz.

Şimdi `Main()` metoduna neden `static` nitelemesi verildiğini anlıyor olmalısınız.

Kurucular (Constructors)

Önce aşağıdaki programı koşturalım, sonra çıktıyı çözümlemeye başlayalım.

Kurucular05.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev();
        }
    }

    class Ev
    {
        int kapıNo = 123;
        string sokakAdı = "Menekşe";
```

```
        public Ev()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
```

`Main()` metodu hiç bir fonksiyon çağırmadı, yalnızca `new` operatörü ile `Ev` sınıfına ait bir nesne yarattı. Nesneye bir ad verilmedi, onunla ilgili başka bir işlem yapılmadı.

`Ev` sınıfına gelince, sınıfın içinde iki değişkeni (veri alanı, field) ile görünüşü sanki bir metodu andıran

```
public Ev()
{
     Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo, sokakAdı);
}
```

bloku var. Bu blok bir metot bildirimini andırıyor, ama iki önemli farkı var:

1. Metot ait olduğu sınıfın adını taşıyor
2. Metodun bir veri tipi yok; void, int, bool gibi bir değer almayacak.

Artık kurucuyu tanımlayabiliriz.

Kurucu

Bir sınıfın kurucusu, sınıf içinde tanımlı olan, sınıfın adını alan ama değer kümesi olmayan özel bir metottur. Görevi sınıfa ait bir nesne yaratmaktır.

Yukarıdaki programa bakalım. `Main()` metodu `Ev` sınıfı içindeki `Ev()` metodunu çağırmadı; yalnızca `new` operatörü ile `Ev` sınıfına ait bir nesne yarattı. O nesneye bir ad vermedi, nesneyle ilgili başka bir iş yapmadı. Ama `Ev` sınıfı içindeki `Ev()` metodu (kurucu) kendiliğinden çalıştı ve konsola

```
ADRES: Menekşe Sokak, No: 123
```

yazdı. Bu nasıl olabildi?

Bu olgu, kurucudan beklenen önemli bir özeliktir. `Main()` metodundaki

```
new Ev() ;
```

deyimi `Ev` sınıfına ait nesneyi kurar kurmaz, onun içindeki

```
public Ev()
```

metodu *(kurucu, constructor)* kendiliğinden çalışır. O metodun (kurucu) ayrıca çağrılmasına gerek olmadığı gibi, çağrılması da mümkün değildir. Gerçekten onu çağırmayı denersek, derleyicimiz itiraz edecektir. `Main()` blokuna `new Ev()` deyimi yerine,

```
Ev aaa = new Ev();
aaa.Ev();
```

deyimlerini koyalım ve aaa adıyla bir nesne yaratalım ve o nesne içinden Ev() metodunu (kurucu) çağıralım. Derleyicinin şu hata iletisini verdiğini göreceğiz.

```
Error   1   'Kurucular.Ev' does not contain a definition for 'Ev' ...
```

Bu durum, şimdiye dek öğrendiklerimizle çelişiyor görünebilir. Ama unutmayalım ki, `Ev` sınıfı içinde tanımladığımız `Ev()` kurucusu genel anlamda bir metot değildir. O özel bir işlevi olan bir metottur, o bir

kurucudur. C# bir sınıf içinde o sınıfın adını alan bir metodu (kurucuyu) başkasının çalıştırmasına asla izin vermez. Sınıfa ait bir nesne kurulur kurulmaz, kurucu kendiliğinden çalışır ve üstüne yüklenen görevi sessizce yapar.

Peki ama, böyle gizemli bir metoda neden gerekseme duyarız? Onun yaptığı işi yapacak apaçık bir metot yazsak olmaz mı?

Elbette olur, ama daha zahmetli olur. Kendiliğinden çalışan kodlar bizim o an ilgilenmediğimiz ama yapılması gereken işleri yaparlar. Örneğin, programı açtığımızda sistemin giriş/çıkış birimlerinin çalışıp çalışmadığını denetleyebilir, uzaktaki bir makinaya bağlanıyorsak ağın çalışıp çalışmadığını denetleyebilir, bir veri tabanına bağlantının olup olmadığını denetleyebilir, v.b.

Kurucuyu biraz daha yakından tanımaya çalışalım. Yukarıdaki programda `Ev()` kurucusunun `public` erişim nitelemesini kaldıralım. Bunu kaldırdığımızda, `Ev()` kurucusu gendeğer (default) olarak `private` erişim belirtecini almış olacaktır. O belirteci ayrıca yazmamıza gerek yoktur.

Kurucular06.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev();
        }
    }

    class Ev
    {
        int kapıNo = 123;
        string sokakAdı = "Menekşe";

        Ev()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Bu programı derlemek istersek, şu hata iletisini alırız:

```
Error   1   'Kurucular.Ev.Ev()' is inaccessible due to its protection level   ...
```

Bu iletiden anlıyoruz ki, `private` nitelemesi olan kurucuyu başkası çalıştıramaz; yani kurucusu `private` olan bir nesneyi başkası yaratamaz.

Şimdi merak edilen bir soruya yanıt arayalım. *"Kurucu özel bir metottur, ama hiçbir değer alamaz"* dedik. Oysa, daha önce tanımladığımız metotlar `int, string, bool, void` gibi bilinen tiplerden birinden bir değer alıyordu; yani fonksiyonu çağırdığımızda bize bir değer veriyordu ((void olsa bile bir değer sayılır). Acaba kurucu bir değer alamaz mı? Alamayacağını göstermek için, yukarıdaki programda `public Ev()` başlığı yerine

```
public void Ev()
```

başlığını yazalım ve derlemeyi deneyelim. Şu hata iletisini alacağız.

```
Error   1   'Ev': member names cannot be the same as their enclosing type   ...
```

Başlıkta `void` yerine `short, int, bool, string` gibi istediğiniz veri tipini koyunuz. Her seferinde aynı hata iletisini alacaksınız. Bunun nedeni açıktır. `public void Ev()` deyimini, derleyici bir kurucu değil, bir metot bildirimi imiş gibi algılıyor. O metodun adı ile sınıfın adının aynı olmasını kabul etmiyor. Çünkü, bir sınıf içinde kurucu dışında hiçbir öğe sınıfın adını alamaz.

Peki kurucu başka ad alabilir mi? Alamayacağını deneyerek görebiliriz. Başlığa `public Ev()` yerine

```
public  Salon()
```

yazalım ve gövdesini aynen bırakalım. Derleyiciden şu iletiyi alırız:

```
Error    1        Method must have a return type ...
```

Böyle olması doğaldır, çünkü derleyiciye bir metot bildirimi yapıyoruz. Kurucu dışındaki her metodun bir değer kümesi (veri tipi) olmak zorundadır.

Parametreli Kurucular

Kurucular özel tip metotlar olduğuna göre, onların da parametrelere bağlı olarak tanımlanabileceklerini tahmin edebiliriz. Gerçekten böyle olduğunu aşağıdaki örnekten görebiliriz.

Kurucular07.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev(123);
        }
    }

    class Ev
    {
        int kapıNo ;
        string sokakAdı = "Menekşe";

        public Ev(int n)
        {
            kapıNo = n;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
```

Kurucu başlığını

```
public Ev(int n)
```

biçiminde yazdık. Bu sözdizimi metot bildirimindeki parametre tanımına benziyor. Bu başlık derleyiciye `Ev()` kurucusunun int tipinden n adlı bir parametresi olduğunu bildirir. `Main()` metodu, kurucuyu

```
   new Ev(123);
```

deyimiyle çağırıyor. Demek ki `n` parametresi yerine 123 değerini koydu. Bu tamamen metot çağırmaya benziyor. Kurucunun gövdesine bakarsak, `kapıNo` yerine 123 değerini atadığını ve onu konsola yazdığını görürüz.

İpucu

Kurucu başlığındaki `public Ev(int n)` yerine `public Ev()` yazılırsa, `Main()` 'in gövdesindeki `new Ev(123)` deyimi nesneyi kuramaz.

Derleyici şu hatayı iletir:

```
Error   1   'Kurucular.Ev' does not contain a constructor that takes '1' arguments   ...
```

Bu iletinin anlamı şudur. `Main()` metodu kurucuyu `Ev(123)` diye çağırdı; yani kurucunun 123 değerini alabilecek bir parametreye bağlı olduğunu varsaydı. Oysa, `Ev` sınıfı içindeki kurucu tanımı `public Ev()` şeklinde parametresizdir. `Main()` metodu 1 parametreli kurucu çağırıyor, ama öyle bir kurucu yok.

Tersine olarak, kurucu başlığında `public Ev(int n)` yazılı iken `Main()` 'in gövdesindeki `new Ev()` deyimi nesneyi kuramaz. Derleyici şu hatayı iletir.

```
Error   1   'Kurucular.Ev' does not contain a constructor that takes '0' arguments   ...
```

Bu iletinin de anlamı şudur. `Main()` metodu kurucuyu `Ev()` diye çağırdı; yani kurucunun parametresiz olduğunu varsaydı. Oysa, `Ev` sınıfı içindeki kurucu tanımı `public Ev(int n)` şeklinde 1 parametrelidir. `Main()` metodu parametresiz kurucu çağırıyor, ama öyle bir kurucu yok.

Böyle olduğunu deneyerek görünüz.

Metot çağırma kuralından biliyoruz ki, metodu çağırırken, onun bildiriminde belirtilen parametrelerin sayısı, sırası ve tipi çağrıda aynen yer almalıdır.

Şimdi iki parametreli bir kurucu tanımlayalım. Kurucu başlığına public `Ev(int n , string s)` ve çağrısına da new `Ev(123,"Menekşe")` yazalım.

Kurucular08.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev(123, "Menekşe");
        }
    }

    class Ev
    {
        int kapıNo;
        string sokakAdı;

        public Ev(int n, string s)
        {
            kapıNo = n;
            sokakAdı = s;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
```

```
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
```

Yukarıdaki kodları çözümlemek kolaydır.

Kurucu başlığını

```
public Ev(int n, string s)
```

biçiminde yazdık. Bu sözdizimi metot bildirimindeki parametre tanımına benziyor. Bu başlık derleyiciye Ev() kurucusunun int tipinden n adlı bir parametresi ile string tipinden s adlı bir parametresinin olduğunu bildirir. Main() metodu, kurucuyu

```
new Ev(123, "Menekşe");
```

deyimiyle çağırıyor. Demek ki `n` parametresi yerine 123 değerini, s parametresi yerine "Menekşe" değerini koydu. Bu iş, iki parametreli metot çağırmaya benziyor. Kurucunun gövdesine bakarsak, `kapıNo` yerine 123 değerini, `sokakAdı` değişkenine "Menekşe" değerini atadığını ve onları konsola yazdığını görürüz.

Şimdiye kadar parametresiz, bir parametreli ve iki parametreli kurucular tanımladık. Metotlardaki kurala uyarak, kurucunun parametre sayısını istediğimiz kadar artırabiliriz.

Aşkın Kurucular (overloaded constructors)

Şimdi yeni bir soruya yanıt arayalım. Acaba bir sınıfın birden çok kurucusu olabilir mi?

Bu sorudaki ince nokta şudur. Metot imzası (başlığı) metodu belirleyen bütün öğelere sahiptir: değer-tipi, adı ve parametreleri. Bir metodun kurucu olabilmesi için değeri olmayacak ve ait olduğu sınıfın adını alacak. O zaman, birden çok kurucu tanımlamak için, elimizde tek seçenek kalıyor: parametrelerin sayısını, sırasını ve tipini değiştirmek. Gerçekten, parametrelerin, sayısını, sırasını ve tiplerini değiştirerek bir sınıf içinde istediğimiz kadar kurucu tanımlayabiliriz. Bu işi yaparken parametrelere yüklendiğimiz için, farklı parametrelerle birden çok kurucu tanımlama işine *aşırı yükleme (overloading)* deniyor. Aşırı yüklenmiş kurucuya da *aşkın kurucu (overloaded constructor)* diyeceğiz. Aşırı yüklemeye basit bir örnek verebiliriz.

Kurucular09.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev();
            new Ev(456);
            new Ev(789, "Papatya");
        }
    }

    class Ev
    {
        int kapıNo = 123;
        string sokakAdı = "Menekşe";

        public Ev()
        {
```

```
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }

        public Ev(int n)
        {
            kapıNo = n;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }

        public Ev(int n, string s)
        {
            kapıNo = n;
            sokakAdı = s;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
ADRES: Menekşe Sokak, No: 456
ADRES: Papatya Sokak, No: 789
```

Programı çözümlemek kolaydır. `Ev` sınıfında parametresiz, 1 parametreli ve 2 parametreli olmak üzere üç kurucu tanımlandı. `Main()` metodu bu üçünü sırayla çağırdı. Her kurucu konsola bir adres yazdı. Parametresiz kurucu, adresteki `kapıNo` ve `sokakAdı` değişkenlerinin değerini sınıf değişkeninden aldı. Bir parametreli kurucu, `kapıNo` değişkeninin değerini çağrıdan, `sokakAdı` değişkeninin değerini sınıftan aldı. İki parametreli kurucu, `kapıNo` ve `sokakAdı` değişkenlerinin değerlerini çağrıdan aldı.

Aşağıdaki örnek kompleks sayı gösteren üç tane farklı kurucu tanımlamaktadır.

Kurucular10.cs

```
using System;
class Kompleks
{
    public Kompleks(int i, int j)
    {
        Console.WriteLine("{0} + i{1}" , i , j);
    }
    public Kompleks(double i, double j)
    {
        Console.WriteLine("{0} + i({1})", i, j);
    }
    public Kompleks()
    {
        int i = 5;
        int j = 8;
        Console.WriteLine("{0} + i{1}", i, j);
    }
}
class Uygulama
{
    public static void Main()
    {
```

```
        Kompleks c1 = new Kompleks(20, 25);
        Kompleks c2 = new Kompleks(2.5, 5.9);
        Kompleks c3 = new Kompleks();
    }
}
```

Statik Kurucular

Statik kurucu C# ile ortaya çıkan yeni bir kavramdır. Sınıfa ait ilk dinamik nesne yaratılmadan önce statik kurucu çağrılır. Sözdizimi şöyledir:

Kurucular11.cs

```
public class Deneme
{
    static Deneme()
    {
        // Başlatma kodları.
        // Yalnız statik öğelere erişilir
    }
    // Sınıfın öteki metotları
}
```

Özellikler:

1. Sınıfın bir tek statik kurucusu olabilir.
2. Statik kurucu parametresizdir.
3. Sınıfın ancak statik öğelerine erişebilir.
4. Statik kurucunun erişim belirteci olmaz.

Aşağıdaki program statik kurucuya basit bir örnektir.

Kurucular12.cs

```
using System;
class basitSınıf
{
    static long tikTak;
    static basitSınıf()
    {
        tikTak = DateTime.Now.Ticks;
    }
    public static void Main()
    {
        new basitSınıf();
        Console.WriteLine(basitSınıf.tikTak);
    }
}
```

Özet

Kurucu çağrılınca sınıfa ait bir nesne yaratılır; nesne yaratılır yaratılmaz kurucunun kodları kendiliğinden çalışır.

Eğer sınıfın bir kurucusu yoksa, CLR (Common Language Runtime) kendiliğinden sınıfa ait bir nesne yaratır. Buna *genkurucu* (default constructor) denilir.

Bir sınıfın istenildiği kadar kurucusu olabilir. Bir kurucunun parametrelerinin tipi, sayısı, sırası değişince farklı bir kurucu elde edilir *(Aşkın Kurucu)*.

- Kurucular değer almaz
- Kurucular aşırı yüklenebilirler.
- Bir sınıfta statik ve dinamik kurucular varsa, öncelik dinamik kuruculardadır.

Yokediciler (Destructors)

Birinci Bölümün sonunda açıkladığımız gibi, C# dili işi biten nesneleri Çöp Toplayıcı (GC- Garbage Collection) ile bellekten atar, boşalan bellek bölgesini Heap 'e ekler. Java ve C# dillerinde otomatik yapılan bu iş C++ dilinde programcı tarafından yapılır; yani programcı yarattığı bir nesneyi işi bitince bellekten silecek kodu da yazmak zorundadır.

Çöp toplayıcı bellekten işi biten nesneleri atmakta kusursuzdur, ama atma zamanını programcı belirleyemez; o derleyicinin işidir. Bu nedenle, çok özel durumlarda, C# da yaratılan bir nesnenin çöp toplayıcının keyfine bırakılmadan, işi biter bitmez bellekten silinmesi istenebilir. Bunu yapmak için kurucu'nun (constructor) tersine iş yapan yokedici (destructor) kullanılır.

Yokedici sınıf içinde sınıf adıyla parametresiz bir kurucu gibi tanımlanır, ancak önüne (~) simgesi konulur. Sözdizimi şöyledir:

```
class Ev
{
    ~ Ev()   // yokedici
    {
            // silme deyimleri
    }
}
```

Yokediciler yalnızca sınıflar içindir; yapılarda kullanılmaz.

Bir sınıfın yalnızca bir tane yokedicisi olabilir.

Yokedici aşırı yüklenemez, kalıtımı olamaz.

Yokedici çağrılamaz; o kendisi otomatik çalışır.

Yokedici değiştirilemez; parametresi yoktur.

Alıştırmalar

1. Aşağıdaki programdaki hatayı bulup düzeltiniz.

Kurucular13.cs

```
class Uygulama
{
    static void Main()
    {
        System.Console.WriteLine(birSınıf.i);
    }
}
```

```
class birSınıf
{
    public int i = 10;
}
```

2. Aşağıdaki programı koşturmadan çıktıyı tahmin ediniz.

Kurucular14.cs

```
class Uygulama
{
    static void Main()
    {
        birSınıf a = new birSınıf();
        birSınıf b = new birSınıf();
        a.j = 11;
        System.Console.WriteLine(a.j);
        System.Console.WriteLine(b.j);
        birSınıf.i = 30;
        System.Console.WriteLine(birSınıf.i);
    }
}
class birSınıf
{
    public static int i = 10;
    public int j = 10;
}
```

3. Aşağıdaki programı koşturmadan çıktıyı tahmin ediniz.

Kurucular15.cs

```
class Uygulama
{
    public static void Main()
    {
        birSınıf a;
        System.Console.WriteLine("Main");
        a = new birSınıf();
    }
}
class birSınıf
{
    public birSınıf()
    {
        System.Console.WriteLine("birSınıf");
    }
}
```

3. Aşağıdaki programı çözümleyiniz.

Kurucular16.cs

```
using System;
public class Üçgen
{
    private int _yükseklik;
    private int _taban;
```

```
    public int Yükseklik
    {
        get
        {
            return _yükseklik;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _yükseklik = value;
        }
    }

    public int Taban
    {
        get
        {
            return _taban;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _taban = value;
        }
    }

    public double Alan
    {
        get
        {
            return _yükseklik * _taban * 0.5;
        }
    }


    public Üçgen()
    {
        Console.WriteLine("Üçgen kurucusu çalıştı");

        _yükseklik = _taban = 1;
    }
}

class Uygulama
{
    static void Main(string[] args)
    {
        Üçgen üçgen = new Üçgen();

        Console.WriteLine("Yükseklik:\t{0}", üçgen.Yükseklik);
        Console.WriteLine("Taban:\t{0}", üçgen.Taban);
        Console.WriteLine("Alan:\t{0}", üçgen.Alan);
    }
}
```

4. Şimdi yukarıdaki kurucuyu parametreli hale getirelim.

Kurucular17.cs

```
using System;
public class Üçgen
{
    private int _yükseklik;
    private int _taban;

    public int Yükseklik
    {
        get
        {
            return _yükseklik;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _yükseklik = value;
        }
    }

    public int Taban
    {
        get
        {
            return _taban;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _taban = value;
        }
    }

    public double Alan
    {
        get
        {
            return _yükseklik * _taban * 0.5;
        }
    }


public Üçgen(int yükseklik, int taban)
{
    Console.WriteLine("Üçgen kurucusu çalıştı");

    this.Yükseklik = yükseklik;
    this.Taban = taban;
}
}

class Uygulama
{
```

```
    static void Main(string[] args)
    {
        Üçgen üçgen = new Üçgen(7,9);

        Console.WriteLine("Yükseklik:\t{0}", üçgen.Yükseklik);
        Console.WriteLine("Taban:\t{0}", üçgen.Taban);
        Console.WriteLine("Alan:\t{0}", üçgen.Alan);
    }
}
```

5. Aşağıdaki programı güneşin yedi gezegenini kapsayacak şekilde değiştiriniz.

Kurucular18.cs

```
using System;
class Gezegen
{

    public int yarıÇap;
    public double yerÇekim;
    private string ad;
    private static int sayaç;

    public Gezegen(int r, double g, string n)
    {
        yarıÇap = r;
        yerÇekim = g;
        ad = n;
        sayaç++;
    }

    public string adYaz()
    {
        return ad;
    }

    public static int GezegenSay()
    {
        return sayaç;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Gezegen earth = new Gezegen(6378, 9.81, "Earth");
        Gezegen saturn = new Gezegen(60268, 8.96, "Saturn");
        Console.WriteLine("Gezegen Sayısı: " + Gezegen.GezegenSay());
    }
}
```