Sınıflar, Yapılar ve Numaralandırmalar(Enumeration)

- *Sınıflar, bildirimleri*
- *Yapıcı Metotlar*
- *Yıkıcı Metotlar*
- *Statik üye Elemanlar*
- *const ve readonly elemanları*
- *Numaralandırmalar (Enumeration)*

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Yapıcı Metotlar(Constructors)**

  Bir nesnenin dinamik olarak yaratıldığı anda otomatik olarak çalıştırılan metotlardır. Bu metotlar sayesinde nesnenin üye elemanlarına ilk değer ataması yapılabilir.

  Daha önceden bahsettiğimiz gibi bir nesne oluşturulduğunda sayısal değerler için 0,referanslar için null, bool türü için false atanır, bu atama bir yapıcı metot ile gerçekleştirilir. Yapıcıların dönüş değeri yoktur, isimleri sınıf ile aynı isimdir.

Yapıcılar, Statik Üyeler, Yıkıcılar

```
class Zaman
 {
  public int sa;
  public int dk;
  public int sn;
 public Zaman(int st,int d,int s)
  {
   sa=st;  dk=d; sn=s;
  }
}
```

```
 static void Main()
 {
Zaman Z=new Zaman(3,4,5);
 }
```

Zaman sınıfı içerisinde 3 adet üye değişken tanımlanmış. Zaman isimli yapıcı nesne oluşturulduğunda çalıştı.

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Varsayılan Yapıcı Metotlar(Default Constructors)**

  Varsayılan metot üye elemanların varsayılan değerlerini belirler ki Zaman örneğinde saat,dakika,saniye üye değişken değerleri varsayılan yapıcı tarafından 0 olarak atanmıştır..

```
static void Main()
{
Zaman Z=new Zaman(); // default constructor çalışır
Z.Yaz();
}
```

Yapıcılar, Statik Üyeler, Yıkıcılar

- Varsayılan yapıcılar aşırı yüklenebilirler. Farklı parametreler alan varsayılan yapıcılar tanımlanabilir.

```
public Zaman(int s,int d,int sn)

public Zaman(int s,int d)

public Zaman(int s)
```

Yapıcılar, Statik Üyeler, Yıkıcılar

```
class Zaman
 {
  public int sa;
  public int dk;
  public int sn;
 public Zaman(int s,int d,int s)
  {
 sa=s;
 dk=d;
 sn=s;
  }
```

```
public Zaman(int s,int d): this(s,d,0)
{
}
public Zaman(int s): this(s,0,0)
{
}
public void yaz()
{
Console.WriteLine("saat:{0}",s);
Console.WriteLine("dk:{0}",dk);
Console.WriteLine("saniye:{0}",sn);
} }
```

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Kopyalayıcı Yapıcı Metotlar(Copy Constructors)**

  Bir Zaman nesnesini yine bir Zaman nesnesini kullanarak oluşturabiliriz. Burada yapıcı metodun giriş parametresi bir nesne olacaktır.

```
public Zaman(Zaman YeniZaman)
{
 sa=YeniZaman.sa;
 dk=YeniZaman.dk;
sn=YeniZaman.sn;
}
```

Yapıcılar, Statik Üyeler, Yıkıcılar

Zaman z1= new Zaman(5,7,45);

Zaman z2= new Zaman(z1); //kopyalayıcı çalıştı

veya

Zaman z2=new Zaman(5,6,20);

z2=z1; //z2'nin elemanlarına ulaşamayız artık

C++ dilinde bir sınıf başka bir sınıfa aktarılırken kopyalayıcı metot çağrılır ancak C#'ta sadece referanslar aktarılır. Kopyalayıcı metot z1 nesnesinin referansı z2 nesnesine aktarılmış oluyor.

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Yıkıcı Metotlar(Destructors) ve Dispose() metodu**

  Nesnelere erişim mümkün olmadığı durumlarda bu nesnelerin heap'ten silinmesi gerekir, çünkü bu nesneler bellek bölgesi işgaline sebep olur. C++' ta bu işlemi kullanıcı kendi yapmak zorundadır.

  C#' ta bu işlem için Gereksiz Nesne Toplayıcısı(Garbage Collector) mekanizması mevcuttur. Bu mekanizma gereksiz nesnelerin tuttuğu referans bölgelerini iade eder, ancak bu nesnelerin faaliyet alanlarının ne zaman iade edileceği kesin olarak bilinmez.

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Yıkıcı Metotlar(Destructors) ve Dispose() metodu**

  Bu yüzden bir nesnenin kapladığı bellek alanlarının ne zaman iade edileceği kesin olarak bilinmez fakat C#' ta bildirilen yıkıcı metotların bu iade işleminden hemen önce çalıştırılacağı kesindir.

  C#'ta bir nesnenin kaynaklarını iade etmenin iki yolu vardır biri Dispose() metodunu kullanmaktır. Diğeri de yıkıcı metod bildirmektir.

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Yıkıcı Metotlar(Destructors) ve Dispose() metodu**

Yıkıcılar C++' ta olduğu gibidir. Yıkıcıların isimleri sınıf ile aynı olmalıdır sadece yıkıcı başına ~ (tilda) işareti eklenir. Geri dönüş değerleri yoktur, public, private gibi erişim denetimleri yoktur.

```
class Yikici
{
~Yikici()
    {
    Console.WriteLine("Yikici çalişti…");
    }
}
```

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Yıkıcı Metotlar(Destructors) ve Dispose() metodu**

```
class Yikici
{
~Yikici()
    { Console.WriteLine(“Yikici çalişti…”);
    }
}
class Ana {
{
Yikici y=new Yikici();
}
Console.WriteLine("son satir....");  }
```

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Yıkıcı Metotlar(Destructors) ve Dispose() metodu**

Yukarıdaki örnek C++ dilinde yazılmış olsaydı ilk önce yıkıcı metot çalışırdı ardından ekrana "son satır" yazdırılırdı. Yikicinin tanımlı olduğu bloğun sonunda "y" nesnesine erişmek mümkün değildir ancak bloğun dışına gelindiğinde ilk yapılan yıkıcı metodu çağırmak olmamıştır. Önce ekrana son satir yazılmış sonra Garbage Collector devreye girmiştir, nesne silinmeden yikici metot ile ekrana "yikici" yazdırılmıştır.

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Statik Üye Elemanları**

C#'ta bütün metotlara sınıflardan ulaşılır ve çoğu durumda erişim için bu sınıflardan nesneler oluşturulur. Bazı durumlarda buna ihtiyaç olamayabilir. Bu durumlarda metotlar statik tanımlanır.

Statik metotlar olaileceği gibi statik değişkenlerde olabilir.

- Statik Metotlar

Örneğin; Math.Sqrt(sayi); Math sınıfından nesne oluşturmadan Sqrt metodunu kullanmış oluyoruz. Yine main yazarken statik tanımlı olması main' in çalışması için herhangi bir sınıf nesnesine ihtiyaç olmamasıdır.

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Statik Metotlar (devam)**

Bir statik metot içinden sınıfın diğer statik metotları çağrılabilir. Ancak normal bir üye metot çağrılamaz. Çünkü normal metotlar nesneler üzerinde işlem yaparlar. Dolayısıyla nesnelerin adresleri metoda this referansı ile gönderilir. Statik metotlar global metotlar olduklarından dolayı this referansları yoktur.

```
class statik
{ public static void yaz()
  {   }
public void yaz1()
{
yaz(); //geçersiz!!!!
} }
```

Yapıcılar, Statik Üyeler, Yıkıcılar

- **Statik Değişkenler**

herhangi bir nesne ile static değişkenlere ulaşılamaz. Bu değişkenlere ancak sınıfın adı ile ulaşabiliriz.

```
class statik
{
    public static int a;
    static public int b;
}
class Ana
{ static void Main()
{
  Console.WriteLine(Statik.a);
}}
```

```
class Statik
  {
   public int sa;
   static public void deneme()
    {
     sa=123; //HATA!!!!
    }
  }
```

Yapıcılar, Statik Üyeler, Yıkıcılar

- **const ve readonly**

  Bir değişken program boyunca sabit tanımlanmak isteniyorsa. const kullanılır. Derleme esnasında değeri bilinmelidir. Çalışma zamanında belirlenmez.

  public const Pl=3.14;

  const Sınıf s=new Sınıf(); // HATA!!!!new var.çalışma zamanında.

- **Statik değişkenlerde olduğu gibi const değişkenlere nesne üzerinden ulaşılamaz, bağlı olduğu sınıf üzerinden ulaşılır.**
- **Sabit ifadeleri referans tipinden olamaz. Çünkü referans tiplerinin değerleri çalışma zamanında belirlenir. Fakat public const string a="denem"; geçerlidir, string referans olmasına rağmen. Fakat burada new operatörü kullanılmamıştır.**

Yapıcılar, Statik Üyeler, Yıkıcılar

- **const ve readonly (devam)**

  referans tipleri sabit tanımlamak için readonly anahtar sözcüğü kullanılır. Statik elemanları readonly tanımlarsak const'un referans tipindeki bir versiyonunu elde etmiş oluruz.

```
class Ana
    {
     static readonly Deneme d= new Deneme();
    }
```

Yapıcılar, Statik Üyeler, Yıkıcılar

const : Türkçe'ye Sabit olarak çevrilebilir. Tanımlanma anında değeri verilmek zorundadır. Sonradan değeri değiştirilemez.

readonly : Sadece-Okunabilir anlamına gelir. Class seviyesinde tanımlanır. Tanımlandığı anda değeri verilebilir veya Class Constructor'ında değeri verilebilir. Sonradan değeri değiştirilemez.

```
public class Matematik
{
    private readonly int PI; /// PI değişkeninin değerini burada da
verebilirdim.

    public Matematik()
    {
        PI = 22 / 7;
    }
}
```

C# Dilinde "enum" Yapısı

Enum (Enumaration) Bu yapı yazılım dilinde enum, enumaration ya da enum sabitleri olarak adlandırılır. Değişkenlerin alabileceği değerlerin sabit (belli) olduğu durumlarda programı daha okunabilir hale getirmek için kullanılır.

```
public enum SehirPlakalari
{
   Ankara = 6, Kocaeli = 41, İzmir = 35, Adana = 1, Hatay = 31
}
```

C# Dilinde "enum" Yapısı

Enum yapısının içindeki verilere erişmek için ‘GetNames’metodu kullanılır. Bu metot sayesinde bir "enum" sabiti içerisindeki değerler alınıp string (karakter katarı) formatında bir diziye atanabilir.

GetNames( ) Metodunun Kullanımı :

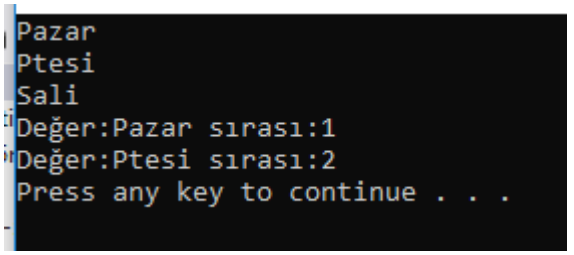
```
string[] SehirPlakalariDizisi = Enum.Getnames(typeof(SehirPlakalari));
```

ToString() Methodunun Kullanımı :

ToString() metoduna aşağıda belirtilen parametreler verilerek Enum sabiti içerisindeki istenilen değerlere ulaşılabilir.

Parametre	Görevi
G veya g	Değeri genel formatta yazar.
D veya d	Değerin sayısal karşılığını yazar.
X veya x	Değerin Hexadecimal karşılığını yazar.

C# Dilinde "enum" Yapısı




```
public enum Gunler
    {
        Pazar=1,Ptesi=2,Sali=3
    }
    class Program
    {
        static void Main(string[] args)
        {
            Gunler gun = new Gunler();

            string[] strGunler = Enum.GetNames(typeof(Gunler));
            foreach (var item in strGunler)
            {
                Console.WriteLine(item);
            }
            Gunler g;
           for(g=Gunler.Pazar;g<Gunler.Sali;g++)
            {
                Console.WriteLine("Değer:" + g + " sırası:" + (int)g);
            }
        }
    }
```

Operator overloading,kalıtım haftaya