C Sharp Programlama Dili/Temel string işlemleri

Ders 19. Temel string işlemleri

String sınıfı

Yeni string oluşturma

Yeni bir string nesnesi şu yollarla oluşturulabilir.

```
string a="deneme";
char[] dizi={'a','b','c','d'};
String b=new String(dizi);
String c=new String(dizi,1,2); //dizi[1]'den itibaren 2 eleman stringe atandı.
String d=new String('x',50); //burada 50 tane x'in yan yana geldiği bir string
oluşuyor. yani d="xxxx..."
```

String metotları

```
static string Concat(params Array stringler)
```

İstenildiği kadar, istenilen türde parametre alır. Aldığı parametreleri birleştirip string olarak tutar.

```
static int Compare(string a,string b)
```

a ve b kıyaslanır, a büyükse pozitif bir sayı, b büyükse negatif bir sayı, eşitlerse 0 döndürür. Büyüklük-küçüklük mantığı sözlükte önde/sonda gelme mantığının aynısıdır. Sonda gelen daha büyük sayılır. Türkçe alfabesine uygundur.

```
static int Compare(string a,string b,bool c)
```

Birinci metotla aynı işi yapar. Tek fark eğer c true ise kıyaslamada büyük/küçük harf ayrımı gözetilmez. Eğer c false ise birinci metottan farkı kalmaz.

```
static int Compare(string a,int indeks1,string b,int indeks2)
```

Birinci metotla aynı mantıkta çalışır. Tek fark kıyaslamada ilk elemanların a[indeks1] ve b[indeks2] sayılmasıdır. Yani stringlerdeki bu elemanlardan önceki elemanlar yok sayılır.

```
static int Compare(string a,int indeks1,string b,int indeks2,bool c)
```

c true yapılırsa az önceki metodun büyük/küçük ayrımı gözetmeyeni kullanılmış olur.

```
int CompareTo(string str)
```

Compare metodunun tek parametre almış ve static olmayan hâlidir. Metodu çağıran veriyle parametre kıyaslanır. Metodu çağırana a, parametreye b dersek `static int Compare(string a,string b)` metodunun yaptığı işin aynısını yapar.

```
int IndexOf(string a)
int IndexOf(char b)
```

Kendisini çağıran string içinde parametredeki veriyi arar. Bulursa bulunduğu indeksi tutar. Eğer aranan birden fazla karaktere sahipse ilk karakterin indeksini tutar. Eğer arananı bulamazsa -1 değeri döndürür. Eğer stringin içinde aranandan birden fazla varsa ilk bulunanın indeksini döndürür.

```
int LastIndexOf(string a)
int LastIndexOf(char b)
```

IndexOf metoduyla aynı işi yapar. Tek fark arananın son bulunduğu yerin indeksini vermesidir. Örneğin aranan 'n' karakteri ise ve string "benim çantam var" ise 2 değil, 8 döndürür.

```
int IndexOfAny(char[] a)
int LastIndexOfAny(char[] b)
```

Birincisi a dizisinin herhangi bir elemanının ilk bulunduğu indeks ile geri döner. İkincisi ise b dizisinin herhangi bir elemanının son bulunduğu indeks ile geri döner. char dizisindeki elemanların hiçbiri bulunamazsa -1 ile geri dönülür.

```
bool StartsWith(string a)
bool EndsWith(string b)
```

Birincisi metodu çağıran string a ile başlıyorsa true, diğer durumlarda false değeri üretir. İkincisi metodu çağıran string b ile bitiyorsa true, diğer durumlarda false değeri üretir.

```
string Trim()
```

metodu çağıran stringin başındaki ve sonudaki boşluklar silinir.

```
string Trim(params char[] dizi)
```

metodu çağıran stringin başındaki ve sonundaki dizi dizisinde bulunan karakterler silinir. Örneğin string `ebebesdefbebe` ise ve dizi 'e' ve 'b' elemanlarından oluşuyorsa `sdef` değeri döndürülür.

```
string PadRight(int toplam)
string PadRight(int uzunluk,char c)
```

Birincisinde metodu çağıran stringin uzunluğu toplam olana kadar sağına boşluk eklenir. İkinci metotta ise aynı işlem boşluk ile değil c karakteri ile yapılır. Örneğin c karakterini '.' yaparak bir kitabın içindekiler bölümünü hazırlayabiliriz. Aynı işlemi stringin soluna yapmak için

```
string PadLeft(int toplam)
string PadLeft(int uzunluk,char c)
```

metotları kullanılır. Bu da sağa yaslı yazılar yazmak için uygundur.

```
string[] Split(params char[] ayirici)
string[] Split(char[] ayirici,int toplam)
string[] Split()
```

Birinci metotta metodu çağıran string ayirici dizisindeki karakter(ler)e göre parçalara ayrılır ve sonuç bir string dizisinde tutulur. İkincisinde bu işlem en fazla toplam kez tekrarlanır. Üçüncüsünde ayırıcı karakter olarak beyaz boşluk karakterleri kullanılır. Örnek:

```
string a="Ahmet,Mehmet,Osman,Ayşe";
string[] b=a.Split(',');
Console.WriteLine(b[0]);
```

NOT: Tek parametre alan Split() metodunda parametre params şeklinde tanımlanmıştır. Yani bu metoda birden fazla char parametresi atabiliriz. Ancak ikinci metodun ilk parametresi params olmayan bir dizidir. Bu parametreye sadece bir char dizisi verebiliriz.

NOT:: params olan tek parametreli Split() metoduna da parametre olarak bir char dizisi verebiliriz. Örnek:

```
string[] s = "ahmet,ozan.mustafa".Split(new char[]{ '.', ','});
string[] s2 = "ahmet,ozan.mustafa".Split(new char[]{ '.', ','},2);
foreach(string a in s)
    Console.WriteLine(a);
Console.WriteLine();
foreach(string b in s2)
    Console.WriteLine(b);
```

Bu programın ekran çıktısı şöyle olacaktır:

```
ahmet
ozan
mustafa

ahmet
ozan.mustafa
```

Burada ikinci örneğe 2 parça sınırı verildiği için program ilk parçalama karakteriyle karşılaştığında parçalama yapmış, sonuçta bu aşamada 2 parça oluştuğu için parçalamayı durdurmuştur.

Join() metodu ise Split() metodunun tam tersi şekilde işler.

```
static string Join(string ayirici,string[] yazilar)
static string Join(string ayirici,string[] yazilar,int baslangic,int toplam)
```

Birincisinde yazilar dizisinin elemanlarının arasına ayirici eklenerek tek bir string hâline getirilir. İkincisinde ise yazilar[baslangic]'ten itibaren toplam kadar eleman ayirici ile birleştirilip tek bir string olarak tutulur.

```
string ToUpper()
```

Kendisini çağıran stringin harflerini büyük yapar ve tutar.

```
string ToLower()
```

Kendisini çağıran stringin harflerini küçük yapar ve tutar.

```
string Remove(int indeks,int adet)
```

Yazıdan indeks nolu karakterden itibaren adet kadar karakteri yazıdan siler ve oluşan bu yeni stringi tutar.

```
string Insert(int indeks,string str)
```

Yazının indeks. elemanından sonrasına str stringini yerleştirir ve tutar.

```
string Replace(char c1,char c2)
```

Yazıda geçen bütün c1 karakterlerini c2 olarak değiştirir ve tutar.

```
string Replace(string s1,string s2)
```

Yazıda geçen bütün s1 yazılarını s2 olarak değiştirir ve tutar.

```
string Substring(int indeks)
```

Metodu çağıran stringin indeks. elemanından sonraki yazıyı tutar.

```
string Substring(int indeks,int toplam)
```

Metodu çağıran yazının indeks. elemanından sonraki toplam karakterlik yazıyı tutar.

Yazıları biçimlendirme

Şimdi isterseniz metotlardan başımızı kaldıralım ve yazıları biçimlendirmeye başlayalım. Biçimlendirme yazıların ekrana veya başka bir akıma istenilen formatta yazılması olayıdır. Ancak öncelikle yazılar üzerinde biçimlendirme yapabilmek için ilgili metodun biçimlendirmeyi destekliyor olması gerekir. Console.WriteLine(), String.Format() ve ToString() metotları biçimlendirmeyi destekler. İlk olarak Console.WriteLine() metodunu kullanarak yazıların nasıl biçimlendirilebileceğine bakalım.

```
Console.WriteLine("{0} numaralı atlet {1}. oldu.",no,sira);
```

Burada ekrana yazılacak yazıdaki {0} ifadesi no verisini, {1} ifadesi de sira verisini ifade ediyor. Bu verileri istediğimiz kadar artırabiliriz. Metin biçimlendirme taslağı:

```
{degisken_no,genislik:format}
```

şeklindedir. Farkındaysanız biz yukarıda sadece `degisken_no` 'yu kullandık. Diğerleri kullanılmadığı zaman varsayılan değerleri kullanılır. genislik yazının karakter cinsinden minimum genişliğidir. Eğer yazının büyüklüğü genislik değerinden küçükse kalan kısımlar boşlukla doldurulur. Eğer genislik pozitif bir sayıysa yazı sağa dayalı, negatif bir sayıysa sola dayalı olur. Örnek:

```
int a=54;
Console.WriteLine("{0,10} numara",a);
Console.WriteLine("{0,-10} numara",a);
```

Bu kodun konsol ekranındaki çıktısı şöyle olur.

```
        54 numara
54         numara
```

Dikkat ettiyseniz henüz `format` 'ı kullanmadık. Formatı kullanmadığımız için şimdiye kadar varsayılan değerdeydi.

```
Console.WriteLine("{0:X}",155);
```

Burada 155 sayısı ekrana 16'lık sayı sisteminde yazılır. Yani ekrana 9B yazılır. Şimdi bu örneği biraz daha geliştirelim:

```
Console.WriteLine("{0:X5}",155);
```

Bu kodla ekrana 0009B yazılır. Yani X'in yanındaki sayı, esas sayının kaç haneli olacağını belirliyor. Buradaki 5 sayısına duyarlıklık değeri denir ve her format belirleyicisi için farklı anlama gelir. Format belirleyiciler, duyarlılık değerleri ve açıklamaları:

Format belirleyici	Format belirleyici açıklaması	Duyarlılık anlamı
C veya c	Para birimi	Ondalık basamakların sayısı
D veya d	Tam sayı	En az basamak sayısı, soldaki basamaklar 0'la beslenir

E veya e	Bilimsel notasyon	Ondalık basamak sayısı
F veya f	Gerçek sayılar (float)	Ondalık basamak sayısı
G veya g	E veya F biçimlerinden kısa olanı	Ondalık basamak sayısı
N veya n	Virgül kullanılarak gerçek sayılar yazılır	Ondalık basamak sayısı
P veya p	Yüzde	Ondalık basamak sayısı
R veya r	Stringe dönüşen türün tekrar eski türe dönüşmesini sağlar	Yok
X veya x	On altılık düzende yazar	En az basamak sayısı, soldaki basamaklar 0'la beslenir

Şimdi bunlarla ilgili bir örnek yapalım:

```
using System;
class Formatlama
{
   static void Main()
   {
      float f=568.87f;
      int a=105;
      Console.WriteLine("{0:C3}",a);
      Console.WriteLine("{0:D5}",a);
      Console.WriteLine("{0:E3}",f);
      Console.WriteLine("{0:F4}",f);
      Console.WriteLine("{0:G5}",a);
      Console.WriteLine("{0:N1}",f);
      Console.WriteLine("{0:P}",a);
      Console.WriteLine("{0:X5}",a);
   }
}
```

Bu programın ekran çıktısı şöyle olur:

```
105,000 TL
00105
5,689E+002
568,8700
105
568,9
%10.500,00
00069
```

NOT: Eğer sayıların normal ondalıklı kısmı duyarlılık değerinden uzunsa yuvarlama yapılır.

String.Format() ve ToString() metotlarıyla biçimlendirme

String sınıfının Format() metodu tıpkı WriteLine() metodu gibi çalışmaktadır. Tek fark WriteLine() biçimlendirdiği yazıyı ekrana yazarken Format() metodu yazıyı bir string olarak tutar. Örnek:

```
string a=String.Format("{0:C3}",50);
```

ToString() metodunda ise biçimlendirme komutu parametre olarak girilir. Onun dışında Format metodundan farkı yoktur. Örnek:

```
string a=50.ToString("C3");
```

Tarih saat biçimlendirme

Örnek program:

```
using System;
class TarihZaman
{
   static void Main()
   {
      DateTime dt=DateTime.Now;
      Console.WriteLine("d--> {0:d}",dt);
      Console.WriteLine("D--> {0:D}",dt);
      Console.WriteLine();
      Console.WriteLine("t--> {0:t}",dt);
      Console.WriteLine("T--> {0:T}",dt);
      Console.WriteLine();
      Console.WriteLine("f--> {0:f}",dt);
      Console.WriteLine("F--> {0:F}",dt);
      Console.WriteLine();
      Console.WriteLine("g--> {0:g}",dt);
      Console.WriteLine("G--> {0:G}",dt);
      Console.WriteLine();
      Console.WriteLine("m--> {0:m}",dt);
      Console.WriteLine("M--> {0:M}",dt);
      Console.WriteLine();
      Console.WriteLine("r--> {0:r}",dt);
      Console.WriteLine("R--> {0:R}",dt);
      Console.WriteLine();
      Console.WriteLine("s--> {0:s}",dt);
      Console.WriteLine();
      Console.WriteLine("u--> {0:u}",dt);
      Console.WriteLine("U--> {0:U}",dt);
      Console.WriteLine();
      Console.WriteLine("y--> {0:y}",dt);
      Console.WriteLine("Y--> {0:Y}",dt);
   }
}
```

Programın ekran çıktısı şuna benzer olmalıdır.

```
d--> 04.01.2009
D--> 04 Ocak 2009 Pazar

t--> 16:25
T--> 16:25:50

f--> 04 Ocak 2009 Pazar 16:25
F--> 04 Ocak 2009 Pazar 16:25:50

g--> 04.01.2009 16:25
G--> 04.01.2009 16:25:50

m--> 04 Ocak
M--> 04 Ocak

r--> Sun, 04 Jan 2009 16:25:50 GMT
R--> Sun, 04 Jan 2009 16:25:50 GMT

s--> 2009-01-04T16:25:50

u--> 2009-01-04 16:25:50Z
U--> 04 Ocak 2009 Pazar 14:25:50

y--> Ocak 2009
Y--> Ocak 2009
```

Özel biçimlendirme oluşturma

Standart biçimlendirme komutlarının yanında bazı özel karakterler yardımıyla kendi biçimlerimizi oluşturabiliriz. Bu özel biçimlendirici karakterler aşağıda verilmiştir:

- # → Rakam değerleri için kullanılır.
- , → Büyük sayılarda binlikleri ayırmak için kullanılır.
- . → Gerçek sayılarda ondalıklı kısımları ayırmak için kullanılır.
- 0 → Yazılacak karakterin başına ya da sonuna 0 ekler.
- % → Yüzde ifadelerini belirtmek için kullanılır.
- E0, e0, E+0, e+0, E-0, e-0 → Sayıları bilimsel notasyonda yazmak için kullanılır.

Örnek bir program:

```
using System;
class OzelBicimlendirme
{
   static void Main()
   {
      Console.WriteLine("{0:#,###}",12341445);
      Console.WriteLine("{0:#.##}",31.44425);
      Console.WriteLine("{0:#,###E+0}",13143212);
      Console.WriteLine("{0:#%}",0.25);
   }
}
```

Bu programın ekran çıktısı şu gibi olmalıdır:

```
12.341.445
31,44
1.314E+4
25%
```

Düzenli ifadeler

Düzenli ifadeler değişken sayıda karakterden oluşabilecek ancak belirli koşulları sağlayan ifadelerdir. Örneğin e-posta adreslerini düşünebiliriz. Dünyada milyonlarca e-posta adresi olmasına ve farklı sayıda karakterden oluşabilmesine rağmen hepsi `kullaniciadi@domainismi.domaintipi` düzenindedir. Örneğin iletisim@microsoft.com bu düzenli ifadeye uymaktadır.

C#'taki düzenli ifade işlemleri temel olarak System.Text.RegularExpressions isim alanındaki Regex sınıfı ile yapılmaktadır. Bir karakter dizisinin oluşturulan düzenli ifadeye uyup uymadığını yine bu isim alanındaki Match sınıfıyla anlarız. Düzenli ifadeler başlı başına bir kitap olabilecek bir konudur. Burada sadece ana hatları üzerinde durulacaktır.

Düzenli ifadelerin oluşturulması

- Bir ifadenin mutlaka istediğimiz karakterle başlamasını istiyorsak ^ karakterini kullanırız. Örneğin ^9 düzenli ifadesinin anlamı yazının mutlaka 9 karakteri ile başlaması demektir. "9Abc" yazısı bu düzene uyarken "f9345" bu düzene uymaz.
- Belirli karakter gruplarını içermesi istenen düzenli ifadeler için \ karakteri kullanılır:

\D ifadesi ile yazının ilgili yerinde rakam olmayan tek bir karakterin bulunması gerektiği belirtilir.

\d ifadesi ile yazının ilgili yerinde 0-9 arası tek bir karakterin bulunması gerektiği belirtilir.

\W ifadesi ile yazının ilgili yerinde alfanumerik olmayan karakterin bulunması gerektiği belirtiliyor. Alfanumerik karakterler a-z, A-Z ve 0-9 aralıklarındaki karakterlerdir.

\w ifadesi ile yazının ilgili yerinde bir alfanumerik karakterin bulunması gerektiği belirtiliyor.

\S ifadesi ile yazının ilgili yerinde boşluk veya tab karakterleri haricinde bir karakterin olması gerektiği belirtiliyor.

\s ifadesi ile yazının ilgili yerinde yalnızca boşluk veya tab karakterlerinden biri bulunacağı belirtiliyor.

Bu öğrendiğimiz bilgiler ışığında 5 ile başlayan, ikinci karakteri rakam olmayan, üçüncü karakteri ise boşluk olan bir düzenli ifade şöyle gösterilebilir:

```
^5\D\s
```

Tahmin edersiniz ki aynı zamanda burada düzenli ifademizin yalnızca 3 harfli olabileceği belirttik. Yukarıdaki `^5\D\s` ifadesine filtre denilmektedir.

> Belirtilen gruptaki karakterlerden bir ya da daha fazlasının olmasını istiyorsak + işaretini kullanırız. Örneğin,

```
\w+
```

filtresi ilgili yerde bir ya da daha fazla alfanumerik karakterin olabileceğini belirtiyor. "123a" bu filtreye uyarken "@asc" bu filreye uymaz. + yerine * kullansaydık çarpıdan sonraki karakterlerin olup olmayacağı serbest bırakılırdı.

> Birden fazla karakter grubundan bir ya da birkaçının ilgili yerde olacağını belirtmek için | (mantıksal veya) karakteri kullanılır. Örnek:

```
m|n|s
```

ifadesinde ilgili yerde m, n ya da s karakterlerinden biri olmalıdır. Bu ifadeyi parantez içine alıp sonuna + koyarsak bu karakterlerden biri ya da birkaçının ilgili yerde olacağını belirtmiş oluruz:

```
(m|n|s)+
```

> Sabit sayıda karakterin olmasını istiyorsak {adet} şeklinde belirtiriz. Örnek:

```
\d{3}-\d{5}
```

filtresine "215-69857" uyarken "54-34567" uymaz.

> ? karakteri, hangi karakterin sonuna gelmişse o karakterden en az sıfır en fazla bir tane olacağı anlamına gelir. Örnek:

```
\d{3}B?A
```

Bu filtreye "548A" veya "875BA" uyarken "875BBA" uymaz.

> . (nokta) işareti ilgili yerde "\n" dışında bir karakterin bulunabileceğini belirtir. Örneğin

```
\d{3}.A
```

filtresine "123sA" ve "8766A" uyar.

> \b bir kelimenin belirtilen yazıyla sonlanması gerektiğini belirtir. Örnek:

```
\d{3}dır\b
```

filtresine "123dır" uyarken "123dırb" uymaz.

> \B ile bir kelimenin başında ya da sonunda bulunmaması gereken karakterler belirtilir. Örnek:

```
\d{3}dır\B
```

filtresine "123dır" veya "dır123" uymazken "123dır8" uyar.

> Köşeli parantezler kullanarak bir karakter aralığı belirtebiliriz. Örneğin ilgili yerde sadece büyük harflerin olmasını istiyorsak [A-Z] şeklinde, ilgili yerde sadece küçük harfler olmasını istiyorsak [a-z] şeklinde, ilgili yerde sadece rakamlar olmasını istiyorsak [0-9] şeklinde belirtebiliriz. Ayrıca sınırları istediğimiz şekilde değiştirebiliriz. Örneğin [R-Y] ile ilgili yerde yalnızca R ve Y arası büyük harfler olabileceğini belirtiriz.

Regex sınıfı

Regex sınıfı bir düzenli ifadeyi tutar. Bir Regex nesnesi şöyle oluşturulur:

```
Regex nesne=new Regex(string filtre);
```

Yani bu yapıcı metoda yukarıda oluşturduğumuz filtreleri parametre olarak veririz. Regex sınıfının Match metodu ise kendisine gönderilen bir yazının düzenli ifadeye uyup uymadığını kontrol eder ve uyan sonuçları Match sınıfı türünden bir nesne olarak tutar.

Match sınıfı

Match sınıfının NextMatch() metodu bir Match nesnesindeki bir sonraki düzenli ifadeyi döndürür. Yazının düzenli ifadeye uyup uymadığının kontrolü ise Match sınıfının Success özelliği ile yapılır. Eğer düzenli ifadeye uygun bir yapı varsa Success özelliğinin tuttuğu değer true olur.

MatchCollection sınıfı

MatchCollection sınıfı ile bir yazı içerisinde düzenli ifadeye uyan bütün Match nesneleri tutulur. Bir MatchCollection nesnesi şöyle oluşturulur:

```
MatchCollection mc=Regex.Matches(string yazi,string filtre)
```

Burada Regex sınıfının static Matches metodu kullanılmışır. Regex sınıfının Matches metodu iki parametre alır. İlk parametresi kontrol edilmek istenen yazı, ikincisi de filtredir. Bir MatchCollection nesnesi oluşturduktan sonra foreach döngüsü yardımıyla bu kolleksiyondaki bütün Match nesnelerine erişebiliriz. Match nesnesine eriştikten sonra düzenli ifadeye uyan karakter dizisinin orijinal yazıdaki yerini Index özelliğiyle ve yazının kendisini ToString() metoduyla elde edebiliriz. MatchCollection sınıfının Count özelliği ile düzenli ifadeye uyan alt karakter dizilerinin sayısını elde ederiz. Eğer Count özelliğinin tuttuğu değer 0 ise düzenli ifadeye uyan yazı bulunamadı demektir. Şimdi bu teorik bilgileri uygulamaya dökelim. Filtremiz şöyle olsun:

```
A\d{3}(a|o)+
```

Bu filtreyle düzenli ifademizin şöyle olduğunu söyleyebiliriz:

- İlk karakter A olacak.
- A'dan sonra üç tane rakam gelecek.
- Üç rakamdan sonra a ya da o karakterlerinden biri ya da birkaçı gelecek. Şimdi programımızı yazalım. Programımız kullanıcının girdiği yazıdaki filreye uyan kısımları kontrol etsin:

```
using System;
using System.Text.RegularExpressions;
class duzenli
{
   static void Main()
   {
      string filtre=@"A\d{3}(a|o)+";
      Console.Write("Yazı girin: ");
      string yazi=Console.ReadLine();
      MatchCollection mc=Regex.Matches(yazi,filtre);
      if(mc.Count==0)
      {
         Console.WriteLine("Yazıda filreye uyumlu kısım yok!");
         return;
      }
      foreach(Match bulunan in mc)
      {
         Console.WriteLine("Bulunan yer: "+bulunan.Index);
         Console.WriteLine("Bulunan yazı: "+bulunan.ToString());
      }
   }
}
```

Bu programda kullanıcının `A123aA657oA456oao` girdiğini varsayarsak ekran çıktısı şu şekilde olur.

```
Bulunan yer: 0
Bulunan yazı: A123a
Bulunan yer: 5
Bulunan yazı: A657o
Bulunan yer: 10
Bulunan yazı: A456oao
```

Programdan da anlayacağınız üzere Index ve ToString() üye elemanları Match sınıfına aittir ve static değildir. Bu programımızı MatchCollection sınıfıyla yapmıştık. Şimdi aynı programı Regex ve Match sınıflarıyla yapalım:

```
using System;
using System.Text.RegularExpressions;
class duzenli
{
    static void Main()
    {
        string filtre = @"A\d{3}(a|o)+";
        Console.Write("Yazı girin: ");
        string yazi = Console.ReadLine();
        Regex nesne = new Regex(filtre);
        Match a = nesne.Match(yazi);
        while (a.Success)
        {
            Console.WriteLine(a.ToString());
            Console.WriteLine(a.Index);
            a = a.NextMatch();
        }
    }
}
```

Bu programda kullanıcının `A123aA657oA456oao` girdiğini varsayarsak ekran çıktısı şöyle olur.

```
A123a
0
A657o
5
A456oao
10
```

Gördüğünüz gibi Match sınıfının NextMatch() metodu yazıdaki bir sonraki eşleşmeyi yine bir Match nesnesi olarak döndürüyor. Bir sonraki eşleşmenin başarılı olup olmadığını Match sınıfının Success özelliği ile anlıyoruz. Sıradaki eşleşme başarılı değilse döngüyü sonlandırıyoruz.

Düzenli ifadelerin içinden bölüm seçme

Bazen bir yazının bir düzenli ifadeye uyup uymadığının öğrenilmesi bizim için yeterli gelmez. Bazen bu uyan yazıların bazı kısımlarını ayrı ayrı görmek isteyebiliriz. Örnek bir program:

```
using System;
using System.Text.RegularExpressions;
class gruplama
{
   static void Main()
   {
      string filtre=@"asf(\d+)(\w+)";
      Console.Write("Yazı girin: ");
      string yazi=Console.ReadLine();
      Regex nesne=new Regex(filtre);
      Match a=nesne.Match(yazi);
      Console.WriteLine("Uyan yazı: "+a.ToString());
      Console.WriteLine("Birinci kısım: "+a.Groups[1].ToString());
      Console.WriteLine("İkinci kısım: "+a.Groups[2].ToString());
   }
}
```

Kullanıcının `asf31321edcve34` girdiğini varsayarsak bu program ekrana şunları yazar:

```
Uyan yazı: asf31321edcve34
Birinci kısım: 31321
İkinci kısım: edcve34
```

Gördüğünüz gibi filtreyi, istediğimiz kısmı parantez içine alarak parçalıyoruz, sonra da bu kısımlara Match sınıfına ait, static olmayan Groups özelliğine eklenen indeksleyici ile erişiyoruz, sonra da bu kısımları ToString() metodu yardımıyla ekrana yazdırıyoruz.

NOT: Sıfırıncı grup daima bulunan düzenli ifadenin tamamıdır.

Regex sınıfının önemli metotları

Split() metodu

Split() metodu bir yazıyı belirli bir düzenli ifadeye göre parçalara ayırır ve bütün parçaları bir string dizisi olarak tutar. Örnek program:

```
using System;
using System.Text.RegularExpressions;
class split
{
   static void Main()
   {
      string filtre=",+";
      string yazi="21,44,,34,,,332,21";
      Regex nesne=new Regex(filtre);
      string[] parcalar=nesne.Split(yazi);
      foreach(string a in parcalar)
        Console.WriteLine(a);
   }
}
```

Bu program ekrana alt alta 21, 44, 34, 332 ve 21 (virgüller olmadan) yazar. Filtremiz bir ya da daha fazla yan yana virgüldür ve bu virgüllere göre yazımız parçalanmıştır. Bunu String sınıfının Split() metoduyla yapamazdık.

Replace() metodu

Replace() metodu, bir yazının bir düzenli ifadeye uyan kısımlarını başka bir yazıyla değiştirmek için kullanılır. Örnek:

```
using System;
using System.Text.RegularExpressions;
class replace
{
   static void Main()
   {
      string filtre=@"\d+:\d+";
      string yazi="Saati belirtmek için : işareti kullanılır. Örnek: 12:35";
      Regex nesne=new Regex(filtre);
      string degistirilmis=nesne.Replace(yazi,"00:00");
      Console.WriteLine(degistirilmis);
   }
}
```

Bu program ekrana `Saati belirtmek için : işareti kullanılır. Örnek: 00:00` yazar. Bu şekilde stringdeki bütün saat bilgilerini değiştirebilirdik. Yine bu da String sınıfındaki Replace() metodunda olmayan bir özellik.