Temel I/O ve String İşlemleri

- Programlama dilleri içersinde yer alan önemli yapılardan biri de I/O sistemidir.

- I/O sistemi bilgisayarın çeşitli kaynaklarına erişmek için kullanılacak yollar anlamına gelir.

- C# dilinde de I/O sistemi ile ilgili tüm sınıflar System.IO isim alanı altında bulunmaktadır.

I/O Sistemi

- C# I/O sistemi stream (akımlar) üzerine kuruludur. Akımlar bir girdi ya da çıktı sistemi üzerinden byte düzeyinde bilgiyi okuyan sanal(soyut) birimlerdir.

- Akımlar ile dosyalardan bilgi okuma/yazma işlemleri dışında klavyeden bilgi okunup, ekrana da bilgi yazılabilir.

- Console.Out (standart çıkış), Console.In (standart giriş) ve Console.Error (standart hata) standart akımlardan bazılarıdır.

- Akımlar ile ilgili işlemler System.IO isim alanı içersinde yer alan Stream sınıfı ile gerçekleştirilir.

- Stream sınıfı akımlar için gerekli temel metot ve özellikleri barındırır. Diğer özel akımlar bu sınıftan türetilerek daha özel hale getirilmiştir. FileStream, MemoryStream ve BufferedStream bunlardan bazılarıdır.

- Temel Stream metotları şunlardır:

Temel Stream metotları şunlardır:

- int Read(byte[] buf, int index, int bytesayisi)
  - Bu metot ile buf[index]'den başlayarak byte sayisi kadar bilgi akımdan okunarak diziye aktarılır.
- int ReadByte()
  - Akımdan bir byte okunur ve int olarak döndürülür.
- void Write(byte[] buf, int index, int bytesayisi)
  - buf[index]'ten itibaren bytesayisi kadar bilgiyi akıma yazar.
- void WriteByte(byte b)
  - Akıma b ile gelen bilgiyi yazar.
- Long Seek (long ilerleme_sayisi, SeekOrigin konum)
  - Bu metot ile akımın konumu, SeekOrigin(End, Begin, Current) numaralandırması ile belirtilmiş konumdan ilerleme sayısı kadar ileriye ötelenir.
- void Flush()
  - Akımla ilgili tampon bellekteki bilgiler silinir ve fiziksel depolama ortamına gönderilir.
- void Close()
  - Akım kapatılarak kaynakları iade edilir.

Temel Stream özellikleri şunlardır:

- bool CanRead: Akımdan okuma yapılabiliyorsa true değerini alır.
- bool CanSeek: Akım konumlandırmayı destekliyorsa true olur.
- bool CanWrite: Akım yazma yapabiliyorsa true değerine sahip olur.
- long Length: Akımın uzunluğunu verir.
- long Position: Akımın o anki konumunu verir.

Dosya ve Klasör İşlemleri

- Dosyalar ile ilgili işlemleri yaparken Stream sınıfını kullanacağız.

- Dosyalar ile ilgili genel bir takım işlemleri gerçekleştirmek için de bazı yardımcı sınıflar bulunur:
  - Directory, File, Path, FileInfo, DirectoryInfo

- Bu sınıfları kullanarak kopyalama, isim değiştirme gibi temel dosya ve klasör işlemlerini gerçekleştirebiliriz.

- FileInfo ve File sınıfları dosya sistemindeki dosyaları, Directory ve DirectoryInfo sınıfları dosya sistemindeki klasörleri, Path sınıfı da dosya, klasör yol (path) bilgilerini temsil eder.

- Directory ve File sınıfları static metotlar içerir. Info eklentili sınıflar ise belirli bir dosya grubu ile işlem yapmaya olanak tanır. Info eklentili sınıf metotları nesne üzerinden kullanılabilirler.

I) Directory Sınıfı

- System.IO isim alanı içersinde yer alan ve sahip olduğu statik metotlar ile klasörlerle ilgili işlemler yapılmasını sağlayan sınıftır.

- Tüm üye elemanları metottur. Hiçbir özelliği yoktur. Önemli metotları ve kullanımları örnektedir.

DirectoryInfo CreateDirectory(string yol)
void Delete(string yol) void Delete (string yol,bool a)
bool Exists(string yol)
DateTime GetCreationTime(string yol)
string GetCurrentDirectory()
string [ ] GetDirectories (string yol)
string GetDirectoryRoot(string yol)
string[ ] GetFiles(string yol)
string GetFileSystemEntries(string yol)
DateTime GetLastAccessTime(string yol) DateTime GetLastWriteTime(string yol)
string[ ] GetLogicalDrives();

DirectoryInfo GetParent(string yol)
void Move (string kaynak_yol, string hedef_yol)
DateTime SetLastAccessTime(string yol,DateTime zaman) DateTime SetLastWriteTime(string yol,DateTime zaman) DateTime SetLastCreationTime(string yol,DateTime zaman)
void SetCurrentDirectory(string yol)

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Klasor";
        Directory.CreateDirectory(yol);
        Console.WriteLine("Klasör Oluşturuldu...");
        Console.ReadLine();
        Console.Clear();

        Console.WriteLine("Klasör Var mı: " + Directory.Exists(yol));
        Console.WriteLine("Oluşturulma Zamanı: " + Directory.GetCreationTime(yol));
        Console.WriteLine("Son Erişim Zamanı : " + Directory.GetLastAccessTime(yol));
        Console.WriteLine("Son Yazma Zamanı  : " + Directory.GetLastWriteTime(yol));

        Console.WriteLine("Şu anki klasör: " + Directory.GetCurrentDirectory());
        Console.ReadLine();
        Console.Clear();

        Console.WriteLine("C Klasör Listesi:");
        string[] klasorler = Directory.GetDirectories(@"C:\");

        foreach (string s in klasorler)
            Console.WriteLine(s);

        Console.ReadLine();
        Console.Clear();
```

```
Console.WriteLine("Kök Klasör:");
Console.WriteLine(Directory.GetDirectoryRoot(yol));
Console.ReadLine();
Console.Clear();

Console.WriteLine("C Dosya Listesi");
string[] dosyalar = Directory.GetFiles(@"C:\");

foreach (string s in dosyalar)
    Console.WriteLine(s);

Console.ReadLine();
Console.Clear();

Console.WriteLine("C Dosya ve Klasör Listesi");
string[] tum = Directory.GetFileSystemEntries(@"C:\");

foreach (string s in tum)
    Console.WriteLine(s);

Console.ReadLine();
Console.Clear();

Console.WriteLine("Mantıksal Sürücüler:");
string[] mantiksalsuruculer = Directory.GetLogicalDrives();

foreach (string s in mantiksalsuruculer)
    Console.WriteLine(s);

Console.ReadLine();
Console.Clear();
```

```
        Console.WriteLine("Bir üst klasör:");
        Console.WriteLine(Directory.GetParent(@"C:\Windows\system32"));

        Console.ReadLine();
        Console.Clear();

        Directory.Move(@"C:\Klasor", @"C:\Windows\Klasor");
        Console.WriteLine("Klasör Taşındı...");
        Console.ReadLine();
        Console.Clear();

        Directory.SetCurrentDirectory(@"C:\Windows");

        Console.WriteLine("Klasör Silindi...");
        Directory.Delete("Klasor");
    }
}
```

II) File Sınıfı

Bazı önemli File Sınıfı metotları ise:

StreamWriter AppendText(string yol)
void Copy (string kaynak, string hedef) void Copy(string kaynak,string hedef,bool ustuneyaz)
FileStream Create(string yol) FileStream Create(string yol,tampon miktarı)
StreamWriter CreateText(string yol)
FileAttributes GetAttributes(string yol)

GetAttributes ile geri dönen değer (FileAttributes) aşağıdakilerden biri olabilir:

Archive,Compressed, Device, Directory, Encrypted, Hidden, Normal, NotContentIndexed, Offline, ReadOnly, ReparsPoint, SparseFile, System, Temporary

FileStream Open (string yol, FileMode A, FileAccess B, FileShare C)

A

Append, Create, CreateNew,Open,OpenorCreate,Trunccate

Append: Açılan dosyanın sonuna ekleme yapmak için kullanılır.

Create: Yeni bir dosya oluşturmak için kullanılır.Eğer ki dosya var ise üzerine yazılır.

CreateNew: Yeni bir dosya oluşturur. Eğer belirtilen dosya var ise hata verir.

Open: Dosya açar.

OpenOrCreate: Belirtilen dosya varsa açılır yoksa yenisi oluşturulur.

Truncate: Belirtilen dosya açılır ve içi tamamen silinir.

B

Read: Dosya okumak için açılır.

ReadWrite:Dosya okumak ve yazılmak için açılır.

Write:Dosya sadece yazmak için açılır.

C

Inheritable: Dosyanın child prosesler tarafından türetilebilmesini sağlar.

None: Dosyanın başka prosesler tarafından açılmasını engeller.

Read: Dosyanın başka prosesler tarafından açılabilmesini sağlar.

ReadWrite:Dosyanın başka prosesler tarafından açılıp okunabilmesini ve üzerine yazılabilmesini sağlar.

Write: Dosyaya başka proseslerin de yazılabilmesini sağlar.

FileStream OpenRead(string yol)
StreamReader OpenText (string yol)
FileStream OpenWrite(string yol)

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string[] tumdosyalar = Directory.GetFileSystemEntries(@"C:\");

        foreach (string s in tumdosyalar)
        {
            Console.Write(s + " ==> ");
            Console.WriteLine(File.GetAttributes(s));
        }
    }
}
```

III) DirectoryInfo Sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Windows\System32";

        DirectoryInfo d = new
                    DirectoryInfo(yol);

        Console.Write("Özellikler:");
        Console.WriteLine(d.Attributes);

        Console.Write("Oluşturma Tarihi:");
        Console.WriteLine(d.CreationTime);

        Console.Write("Klasör var mı?");
        Console.WriteLine(d.Exists);

        Console.Write("Uzantı:");
        Console.WriteLine(d.Extension);
```

```
        Console.Write("Tam Yol:");
        Console.WriteLine(d.FullName);

        Console.Write("Son Erişim Zamanı:");
        Console.WriteLine(d.LastAccessTime);

        Console.Write("Son Yazma Zamanı:");
        Console.WriteLine(d.LastWriteTime);

        Console.Write("Klasör Adı:");
        Console.WriteLine(d.Name);

        Console.Write("Üst Klasör:");
        Console.WriteLine(d.Parent);

        Console.Write("Kök Klasör:");
        Console.WriteLine(d.Root);
    }
}
```

DirectoryInfo sınıfının önemli metotları:

- void Create()
  - Metot ile klasör oluşturur.
- DirectoryInfo CreateSubdirectory(string yol)
  - Bir alt dizin oluşturur.
- void Delete(bool a)
  - Klasörün silinmesini sağlar. bool kontrol ile dolu dahi olsa silinir.
- DirectoryInfo[] GetDirectories()
  - İstenen konumdaki tüm klasörleri DirectoryInfo dizisi haline getirir.

- FileInfo[] GetFiles()
  - İlgili klasördeki tüm dosyalar için FileInfo dizisi oluşturur.
- FileSystemInfo[] GetFileSystemInfos()
  - Tüm dosya ve klasörler için FileSystemInfo dizisi olurur.
- void MoveTo(string hedef)
  - İlgili dizin içeriğiyle birlikte hedefe taşınır.
- void Refresh()
  - İlgili klasörün özelliklerini dosya sisteminden (disk vs.) yeniden yükler.

IV) FileInfo Sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Windows\clock.avi";

        FileInfo f = new FileInfo(yol);

        Console.Write("Özellikler:");
        Console.WriteLine(f.Attributes);

        Console.Write("Oluşturma Tarihi:");
        Console.WriteLine(f.CreationTime);

        Console.Write("Dosya var mı?");
        Console.WriteLine(f.Exists);

        Console.Write("Uzantı:");
        Console.WriteLine(f.Extension);

        Console.Write("Tam Ad:");
        Console.WriteLine(f.FullName);

        Console.Write("Son Erişim Zamanı:");
        Console.WriteLine(f.LastAccessTime);

        Console.Write("Son Yazma Zamanı:");
        Console.WriteLine(f.LastWriteTime);

        Console.Write("Boyut:");
        Console.WriteLine(f.Length);

        Console.Write("Dosya Adı:");
        Console.WriteLine(f.Name);

        Console.Write("Bulunduğu Klasör:");
        Console.WriteLine(f.DirectoryName);
    }
}
```

- FileInfo sınıfı da önceki anlatılan sınıfların metotlarının bir çoğuna sahiptir. Farklı olanlarından bazıları ise:
  - FileInfo CopyTo(string hedef, bool a)
    - İlgili dosyayı hedefe kopyalar, bool ifade true ise üzerine yazar.
  - void MoveTo(string hedef)
    - Dosya hedef olarak belirtilen yere taşınır.
  - void Refresh()
    - İlgili dosya bilgileri dosya sisteminden tekrar okunur.

V) Path Sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\windows\clock.avi";

        Console.Write("Uzantı: ");
        Console.WriteLine(Path.GetExtension(yol));

        Console.Write("Klasör: ");
        Console.WriteLine(Path.GetDirectoryName(yol));

        Console.Write("Dosya Adı: ");
        Console.WriteLine(Path.GetFileName(yol));

        Console.Write("Uzantısız Dosya Adı:");
        Console.WriteLine(Path.GetFileNameWithoutExtension(yol));

        Console.Write("Tam Yol: ");
        Console.WriteLine(Path.GetFullPath(yol));

        Console.Write("Kök Dizin: ");
        Console.WriteLine(Path.GetPathRoot(yol));
```

```
        Console.Write("Geçici Dosya Adı: ");
        Console.WriteLine(Path.GetTempFileName());

        Console.Write("Geçici Dosya Dizini: ");
        Console.WriteLine(Path.GetTempPath());

        Console.Write("Uzantı var mı? : ");
        Console.WriteLine(Path.HasExtension(yol));

        Console.Write("Alternatif Alt Dizin Ayracı: ");
        Console.WriteLine(Path.AltDirectorySeparatorChar);

        Console.Write("Dizin Ayracı: ");
        Console.WriteLine(Path.DirectorySeparatorChar);

        Console.Write("Geçersiz Karakterler: ");
        Console.WriteLine(Path.GetInvalidPathChars());

        Console.Write("Yol Ayracı: ");
        Console.WriteLine(Path.PathSeparator);

        Console.Write("Sürücü Ayracı: ");
        Console.WriteLine(Path.VolumeSeparatorChar);
    }
}
```

Dosya Yazma ve Okuma İşlemleri

- C# dilinde dosya işlemleri de akımlar (stream) sayesinde gerçekleştirilir.

- System.IO isim alanı içinde bir çok dosya okuma yazma ile ilgili bir çok sınıf bulunur.

- BinaryReader, BinaryWriter, TextReader, TextWriter ve Stream sınıfları en önemlilerindendir.

- Metin temelli dosyalar üzerinde çalışmak için TextReader ve TextWriter kullanılır.
- İkilik dosyalarla ilgili işlemler için BinaryReader ve BinaryWriter sınıfları kullanılır.
- FileStream ise daha genel dosya işlemleri yapmak için hem metin hem de ikilik dosyalar için kullanılabilir.

FileStream Sınıfı

- Depolama ortamlarında bulunan dosyalar ile ilgili işlemler yapmak için kullanılır.
- StreamWriter ve StreamReader sınıfları da bu sınıfı kullanarak çalışırlar.
- Dosya üzerinde byte bazında işlemler yapılabileceği gibi metin temelli işlemler de gerçekleştirilebilir.

FileStream bir çok yol ile oluşturulabilir:

```
string yol = @"C:\dosya.txt";

FileStream fs1 = new FileStream(yol, FileMode.OpenOrCreate);
FileStream fs2 = new FileStream(yol, FileMode.OpenOrCreate, FileAccess.Write);
FileStream fs3 = new FileStream(yol, FileMode.OpenOrCreate, FileAccess.Write,
    FileShare.None);

FileInfo fi = new FileInfo(yol);

FileStream fs4 = fi.OpenRead();
FileStream fs5 = fi.OpenWrite();
FileStream fs6 = fi.Create();
FileStream fs7 = fi.Open(FileMode.OpenOrCreate);
```

Görüldüğü gibi FileInfo sınıfını ve bazı metotlarını kullanarak da dosya akımı oluşturma şansı vardır. Dosyalarla ilgili işlemlerimiz bittiğinde FileStream sınıfı Close() metodu ile kapatılmalıdır.

FileStream ile Okuma ve Yazma

- FileStram nesnesi ile bilgi okumak ve yazmak için dört temel metot kullanılır:
  - int ReadByte();
  - int Read(byte[] dizi, int başlangıç, int adet);
  - void WriteByte(byte veri);
  - void Write(byte[] dizi, int başlangıç, int adet);

- ReadByte() metodu akımdan okuma yapamadığı zaman geriye -1 değerini döndürür. Okunan byte değeri int türüne dönüştürülür ve bu değer ile geri dönülür.

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        string yol = @"Dosya.txt";
        FileStream fsw = new FileStream(yol, FileMode.Create, FileAccess.Write);

        byte[] veri = new byte[20];

        for (int i = 0; i < veri.Length; i++) veri[i] = (byte)(i + 1);

        fsw.Write(veri, 0, veri.Length);
        Console.WriteLine("{0} byte yazıldı...", veri.Length);

        fsw.Close();

        FileStream fsr = new FileStream(yol, FileMode.Open, FileAccess.Read);

        byte[] okunan = new byte[fsr.Length];

        fsr.Read(okunan, 0, (int)fsr.Length);

        foreach (byte b in okunan) Console.Write(b + " ");

        fsr.Close();
    }
}
```

- Dosya akımına yazılan veriler dosya sistemindeki dosyaya tamponlama mekanizması olduğundan dolayı hemen aktarılmaz. Verilerin dosyaya aktarılması için ya flush() metodu kullanılmadı ya da dosya close() metodu kullanılarak tamamen kapatılmalıdır.

- FileStream dosya akımı, dosyaların türünden bağımsız olarak çalışmaktadır. Bahsedilen bazı metotların yanı sıra önemli özelliklere de sahiptir.

- Bu sınıfın sahip olduğu önemli özellikler CanRead, CanSeek, CanWrite, Position, Length'dir.

- Önemli Metotları ise
  - void Lock(long konum, long uzunluk);
  - long Seek(long ofset, SeekOrigin s);
    - SeekOrigin numaralandırması Begin,Current ve End elemanlarına sahiptir.

Dosya Akımı ile Text İşlemleri Yapmak

- FileStream byte bazlı işlem yaptığı için metin dosyalarda pek kullanışlı olmayabilir.
- Metin dosyalarda işlem yapmak için ayrıca StreamReader ve StreamWriter isimli iki sınıf bulunur.
- Var olan bir dosya akımı StreamReader sınıfı ile okunur ya da StreamWriter sınıfı ile akıma yeni birşeyler yazılır.

StreamReader Sınıfı

StreamReader nesnesi de farklı yollarla oluşturulabilir.

```
string yol = @"C:\dosya.txt";

FileStream fs = new FileStream(yol, FileMode.Open);
StreamReader sr1 = new StreamReader(fs);

StreamReader sr2 = new StreamReader(yol);

FileInfo fi = new FileInfo(yol);
StreamReader sr3 = new StreamReader(fi);
```

Bu nesneler de Close() metoduna sahiptir.

Önemli Metotları:

- string ReadLine();

  Akımdan bir satırlık veriyi okur ve string türü olarak geri döner.Eğer veri okunamazsa dönen değer null'dur. Okunan satır sonuna \n karakteri eklenmemektedir.

- string ReadToEnd();

  Akımdaki verilerin tamamı string olarak geri döndürülür. Okuma yapılmazsa null değer yerine boş string ile geri dönülür.

- int Read();

  Akımdan 1 karakterlik bilgi okur ve bu karakterin int'e dönüşmüş hali geri döndürülür.

Önemli Metotları:

- int Read(char[] dizi, int indeks, int adet);

  Akımdan adet kadar karakteri, dizi[indeks] elemanından itibaren diziye yerleştirir.

- int Peek();

  Akımdan bir karakterlik bilgi okur, bu karakterin int'e dönüşmüş hali ile geri döner.İşlem başarısız olursa -1 döndürür. En önemli nokta ise akımın konum göstericisinin değişmemesidir.

StreamWriter Sınıfı

Benzer olarak StreamWriter da bir çok yöntemle oluşturulabilir.

```
string yol = @"C:\dosya.txt";

FileStream fs = new FileStream(yol, FileMode.Open);
StreamWriter sw1 = new StreamWriter(fs);

StreamWriter sr2 = new StreamWriter(yol);

FileInfo fi = new FileInfo(yol);
StreamWriter sr3 = new StreamWriter(fi);
```

Bu nesneler de Close() metoduna sahiptir.

Önemli Metotları:

- void Write(string str);

  Akıma str yazısını ekler. Yazının sonuna herhangi bir sonlandırıcı karakter koyulmaz. Aşırı yüklenmiş bir çok şekli vardır.

- void WriteLine(string str);

  Bu metot akıma yazarken eklediği yazının sonuna \n satır atlama karakterini ekler.

- void Flush();

  Tampondaki bilgilerin boşaltılmasını ve dosya sistemindeki dosyanın güncellenmesini sağlar.

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        string yol = @"C:\Dosya.txt";

        FileStream fs = new FileStream(yol, FileMode.Append, FileAccess.Write);
        StreamWriter sw = new StreamWriter(fs);

        while (true)
        {
            string giris = Console.ReadLine();
            if (giris.ToLower() == "x") break;
            sw.WriteLine(giris);
        }
        sw.Flush(); sw.Close();

        Console.Clear();

        StreamReader sr = new StreamReader(yol);

        string satir; int i=1;

        while ((satir = sr.ReadLine()) != null)
        {
            Console.WriteLine("{0}:{1}",i,satir);
            i++;
        }
        sr.Close(); fs.Close();
    }
}
```

BinaryReader ve BinaryWriter Sınıfları

- Şu ana kadar anlatılan sınıflar ile byte, char ve string olarak okuma/yazma işlemleri yapılabiliyordu.
- Diğer temel veri türlerinin ikilik kodlama olarak yazılmasını ve okunmasını BinaryReader ve BinaryWriter sınıfları sağlar.
- Bu iki sınıf StreamReader ve StreamWriter sınıfına çok benzerdir.
- Bu sınıflardan farklı olarak Write() metotları int, char, decimal … gibi parametreler alabilir ve ReadInt32, ReadChar, ReadDecimal … gibi metotlar içerir.

Console I/O İşlemleri

- I/O işlemlerinde System.IO isim alanı içinde olmayan tek sınıf Console'dir.

- Konsol Giriş/Çıkış işlemleri için önceden tanımlanmış 3 tane standart akım bulunur.
  - Console.Out ve Console.Error TextWriter sınıfının bir çeşididir.
  - Console.In ise TextReader türündendir.

- Console.Out.WriteLine ile Console.WriteLine aynı işi yapar.

- Cosole.In ile de Read işlemleri yapılabilir. Bunların dışında Console.In içersinde TextReader sınıfının bazı metotları da bulunur:
  - int Peek()
  - int ReadBlock(char dizi[], int index, int adet)
  - string ReadToEnd();

- C# da bütün I/O işlemleri akımlar üzerine kuruludur ve akımlar başka akımlara yönlendirilebilir.

- Örneğin Console.In bir NetworkStream'e yönlendirilerek ağ üzerinden bilgi okunması için Read metotları kullanılabilir.

- Console sınıfı ile standart akımları yönlendirmek için SetOut, SetIn ve SetError metotları kullanılabilir.

String İşlemleri

- Programlarda kullanılan verilerin büyük bir çoğunluğu string türündendir.

- C# da string işlemleri System.String sınıfı içersinde yer alan üye metot ve özelliklerle yapılır.

- String veriler için indeksleyici de tanımlanmıştır. Yani bir karakter dizisi gibi işlem görebilir. Fakat karakterler salt okunurdur.

- Stringler değişik şekillerde tanımlanabilirler:
  - string a = “C# Programlama Dili”;
  - char[] dizi = {’1’,’2’,’3’,’4’,’5’};
    string s = new string(dizi);
  - string s = new string(dizi,1,2);
  - string s = new string(‘w’,10);

String sınıfının metotları oldukça fazladır. Önemli metotlarından bazıları şunlardır:

- String.Concat()
  - String verilerin ardarda eklenmesini sağlar. + operatörü ile eşdeğerdir.
- String.Compare()
  - İki string değeri karşılaştırı. == ve != operatörleri ile benzer işlem gerçekleştirir. Fakat Compare metodunun bazı aşırı yüklemeleri ile daha gelişmiş (büyük küçük harf duyarlılığı gibi.) karşılaştırmalar yapılabilir.

– stringnesne.IndexOf()

  - string içersinde alt stringlerin aranmasını sağlar. Geriye aranan alt stringin bulunduğu konumu ya da bulunamadı (-1) bilgisini döndürür.

– stringnesne.LastIndexOf()

  - IndexOf ile aynı çalışır. Farkı ise aranan karakterin en son nerede görüldüğünün indeksini geri döndürür.

- stringnesne.Trim()
  - Bir string ifadenin başındaki ve sonundaki boşlukları ya da belirlenmiş karakterleri atar.

- PadRight, PadLeft
  - Bir stringin sağına ya da soluna yeni karakterler ilave etmek için kullanılır.

- stringnesne.Split()
  - Belli bir biçime sahip olan toplu string verileri, beliritilen ayırıcı karakterlerden ayırıp ayrı bir string dizisi üretir.

- string.Join()
  - Split metodunun tersi gibi çalışır. Ayrı stringleri belli bir ayırıcı karakter ile birleştirip tek bir string üretir.

- stringnesne.ToUpper()
  - Stringin karakterlerinin hepsini büyük harfe çevirip geri döndürür.

- stringnesne.ToLower()
  - Stringin karakterlerinin hepsini küçük harfe çevirip geri döndürür.

- stringnesne.Remove()
  - String içinden belli sayıda karakterin atılmasını sağlar.

- stringnesne.Insert()
  - String içine belirli indeksten itibaren yeni bir string eklemek için kullanılır.

- stringnesne.Replace()
  - String içindeki belirlenen karakter ya da karakterleri başkalarıyla değiştirir.

- stringnesne.SubString()
  - String ifadenin belli bir kısmının elde edilmesini sağlar

Biçimlendirme

- Program çıktılarının belli bir düzende olması oldukça önemlidir. Bazı zamanlar standart çıktıların anlaşılması zor olabilir.

- Bu problemi çözmek için biçimlendirme teknikleri kullanılır. Bu teknikler yalnızca biçimlendirmeyi destekleyen komutlar tarafından kullanılabilir.

- Console.WriteLine(), String.Format() ve ToString metotları biçimlendirmeyi destekleyen metotlardır.

- Bu metotlarda kullanılan {} parantezleri arasındaki ifadeler belli değişkenlerin belli bir düzende biçim metnine aktarılmasını sağlıyordu:
  - Console.WriteLine(“Merhaba {0}!”,isim)

- Biçimlendirme bu {} işaretleri arasında eklenir. En genel kullanımı
  - {değişken_no, genişlik : format}

- Sadece değişken verildiğinde değişkenin türüne göre varsayılan ayarlar kullanılır.

- Genişlik yazılacak olan verinin diğer verilerle olan mesafesini ve hangi yöne hizalanması gerektiğini belirler. Format ise verinin türüne göre değişik biçimlendirilmesini sağlar.

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        int a = 2063;
        float f = 0.789F;
        double d = 1234.6789;

        Console.WriteLine("Para Birimi   : {0:C2}", a);
        Console.WriteLine("Tamsayı       : {0:D6}", a);
        Console.WriteLine("Gerçek Sayı   : {0:F5}", d);
        Console.WriteLine("Bilimsel      : {0:E3}", d);
        Console.WriteLine("Kısa Gösterim : {0:G5}", d);
        Console.WriteLine("Binlik Ayracı : {0:N5}", d);
        Console.WriteLine("Yüzde         : {0:P2}", f);
        Console.WriteLine("16'lık        : {0:X5}", a);

        Console.WriteLine("\nToString() Metodu ile:");
        Console.WriteLine(a.ToString("N2"));

        Console.WriteLine("\nFormat() Metodu ile:");
        Console.WriteLine(String.Format("{0:X4}",a));
    }
}
```

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        DateTime dt = DateTime.Now;

        Console.WriteLine("d : {0:d}", dt);
        Console.WriteLine("D : {0:D}", dt);
        Console.WriteLine("t : {0:t}", dt);
        Console.WriteLine("T : {0:T}", dt);
        Console.WriteLine("f : {0:f}", dt);
        Console.WriteLine("F : {0:F}", dt);
        Console.WriteLine("g : {0:g}", dt);
        Console.WriteLine("G : {0:G}", dt);
        Console.WriteLine("m : {0:m}", dt);
        Console.WriteLine("M : {0:M}", dt);
        Console.WriteLine("r : {0:r}", dt);
        Console.WriteLine("R : {0:R}", dt);
        Console.WriteLine("s : {0:s}", dt);
        Console.WriteLine("u : {0:u}", dt);
        Console.WriteLine("U : {0:U}", dt);
        Console.WriteLine("y : {0:y}", dt);
        Console.WriteLine("Y : {0:Y}", dt);
    }
}
```

- Standart biçimlendirmelerin dışında özel biçimlendirmeler de tasarlanabilir.

- “#” “,” “.” “0” “E” ve “%” karakterleri ile özel biçimlendirmeler oluşturulabilir.
  - {0:#,###.##}
  - {0:#%}
  - {0:#,###E+0}