



HÖGSKOLAN I BORÅS

Input och Output (I/O)

Troelsen Kapitel 20

Grundläggande applikationsutveckling med C#

Agenda

- Directory, DirectoryInfo, File, FileInfo och DriveInfo.
- IDisposable och *using*.
- Stream, FileStream och MemoryStream.
- TextReader och TextWriter.
- StreamReader och StreamWriter.
- StringReader och StringWriter.
- StringBuilder.
- BinaryReader och BinaryWriter.
- BinaryFormatter.
- [Serializable] och [NonSerialized].

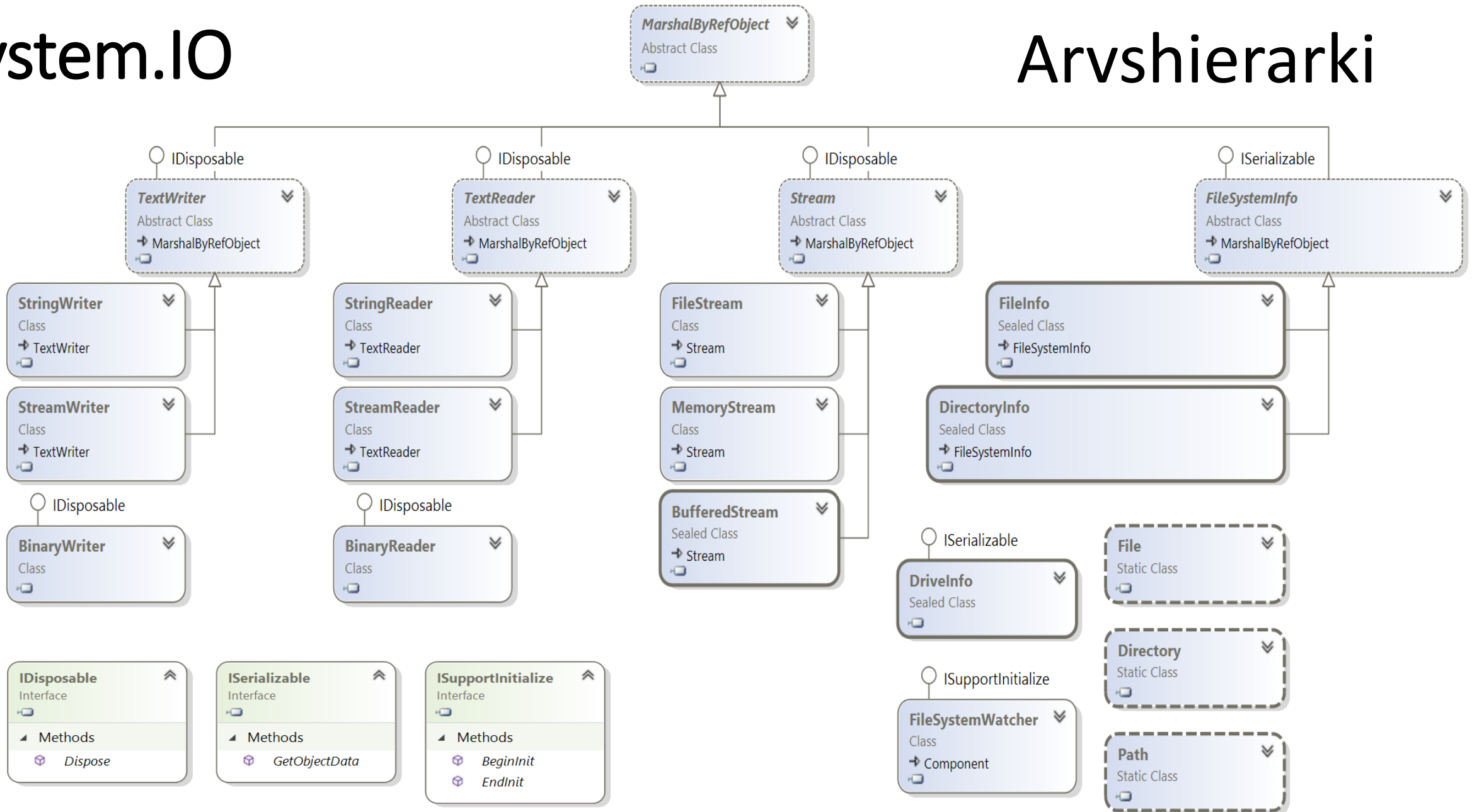
Namespace System.IO

- **System.IO** innehåller typer för att:
 - Manipulera diskenheter, foldrar och filer.
 - Läsas/skriva till/från
 - Filer
 - Minne
 - Strängar
- För att använda typerna från *namespace* **System.IO** måste **System.IO** inkluderas längst upp i källkodsfilen med nyckelordet **using**:

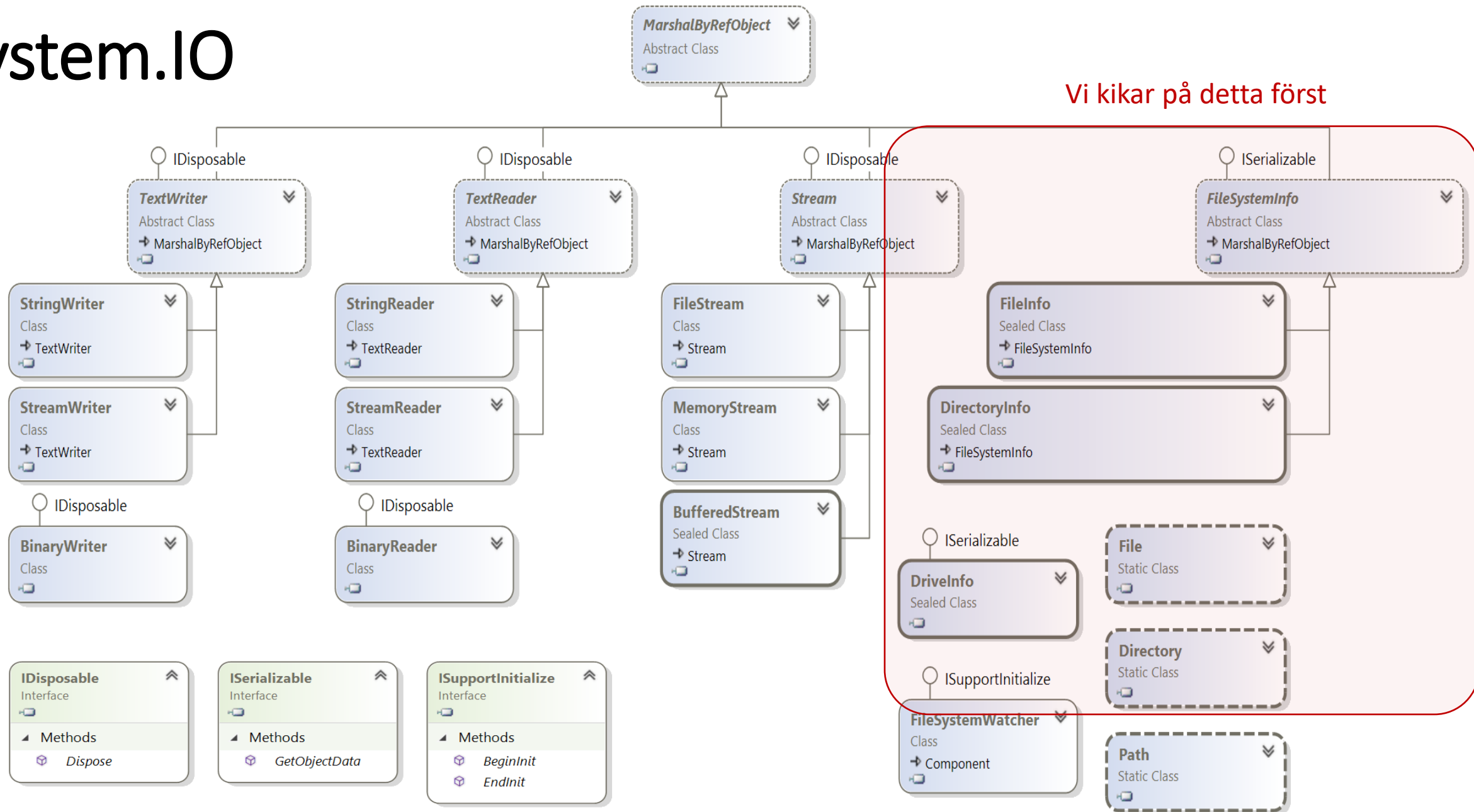
```
using System.IO;
```

System.IO

Arvshierarki

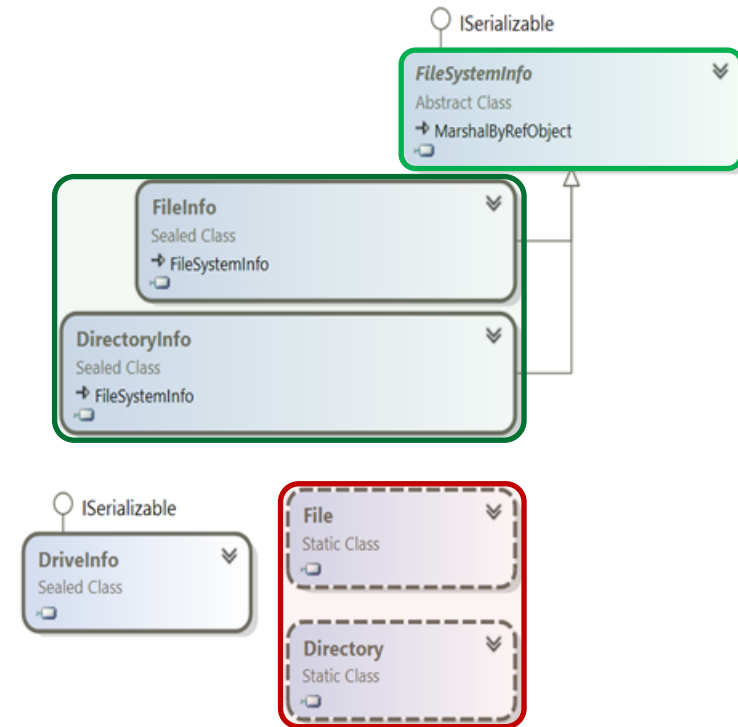


System.IO



Directory, DirectoryInfo, File och FileInfo

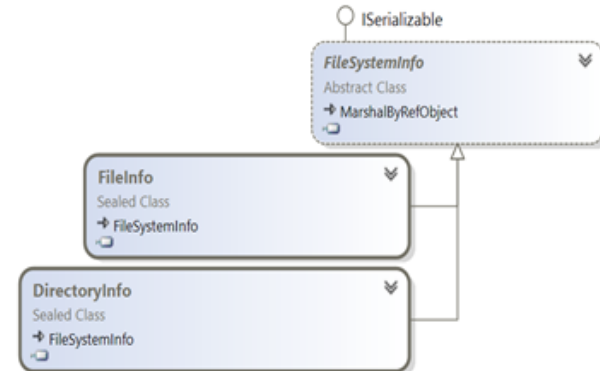
- **System.IO** innehåller fyra klasser (**Directory**, **File**, **DirectoryInfo**, **FileInfo**) för att manipulera filer samt interagera med datorns filsystem.
- **Directory** och **File** (ärver från **Object**) är **statiska** klasser och innehåller **statiska medlemmar** för att **skapa, ta bort, kopiera** och **flytta foldrar/filer**.
- **DirectoryInfo** och **FileInfo** (som ärver från **FileSystemInfo**) måste instansieras och har **liknande funktionalitet** som **Directory** och **File** via **instansmedlemmar**.
- Medlemmarna i **DirectoryInfo** och **FileInfo** returnerar **starkt typade objekt** med mycket information om foldrar/filer (skapelsetid, läs-/skrivbarhet, osv) medan de statiska medlemmarna i **Directory** och **File** returnerar **strängar**.



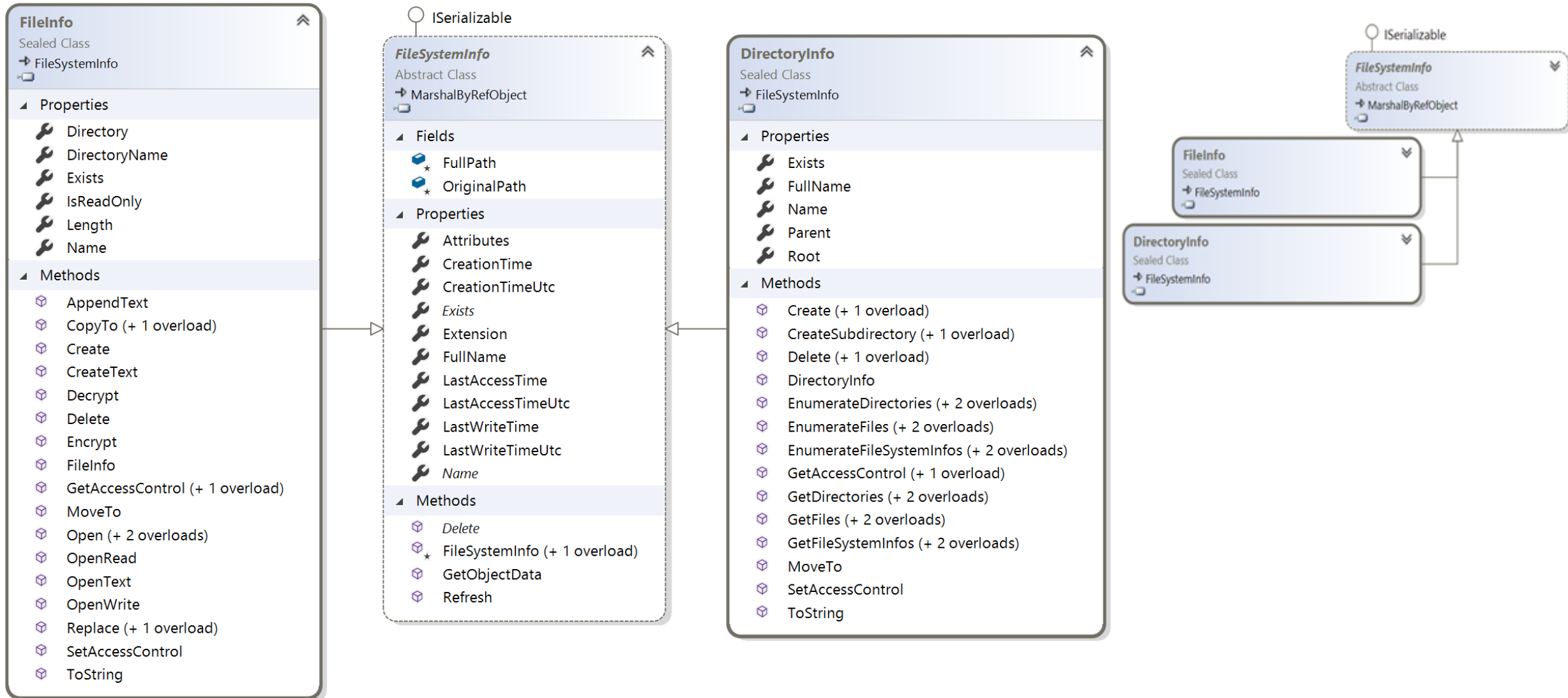
FileSystemInfo

- **DirectoryInfo** och **FileInfo** ärver funktionalitet från **FileSystemInfo**, som innehåller medlemmar för att bl.a. manipulera **filattribut**, **skapelse-/access-/skrivtid**, för att ta reda på en folders/fils **namn och sökväg**, och en **fils filändelse**, samt för att ta reda på om **foldern/filen finns**.

Property	Meaning in Life
Attributes	Gets or sets the attributes associated with the current file that are represented by the FileAttributes enumeration (e.g., is the file or directory read-only, encrypted, hidden, or compressed?)
CreationTime	Gets or sets the time of creation for the current file or directory
Exists	Determines whether a given file or directory exists
Extension	Retrieves a file's extension
FullName	Gets the full path of the directory or file
LastAccessTime	Gets or sets the time the current file or directory was last accessed
LastWriteTime	Gets or sets the time when the current file or directory was last written to
Name	Obtains the name of the current file or directory



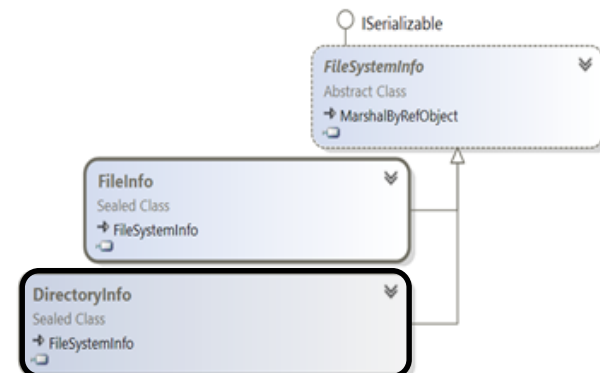
FileSystemInfo, FileInfo och DirectoryInfo



DirectoryInfo

- Utöver den ärvda funktionaliteten från **FileSystemInfo** innehåller **DirectoryInfo** funktionalitet för att bl.a. **skapa, ta bort och flytta foldrar**, **lista subfoldrar och filer**, samt **hämta en folders förälderfolder och rotfolder**.

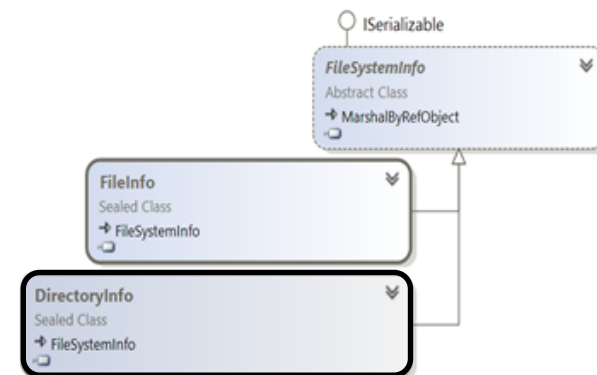
Member	Meaning in Life
Create() CreateSubdirectory() Delete()	Creates a directory (or set of subdirectories) when given a path name Deletes a directory and all its contents
GetDirectories() GetFiles()	Returns an array of DirectoryInfo objects that represent all subdirectories in the current directory Retrieves an array of FileInfo objects that represent a set of files in the given directory
MoveTo()	Moves a directory and its contents to a new path
Parent Root	Retrieves the parent directory of this directory Gets the root portion of a path



DirectoryInfo

- **DirectoryInfo** konstruktorn tar en **sökväg till en folder** som in-parameter.
- Sökvägen till **aktuell folder** (som applikationen exekverar från) **anges med en punkt .**
- När man jobbar med sökvägar kan man använda en **verbatim-sträng**, som är en sträng som föregås av **@** tecknet, t.ex. **@“sträng”**.
- En **verbatim-sträng** tolkar alla **specialtecken som vanliga tecken**, t.ex. kan man skriva “C:\Windows” istället för “C:\\Windows” med en verbatim-sträng, annars måste *backslash* tecknet \ “escape:as” med “escape tecknet” (en *backslash* i C#).

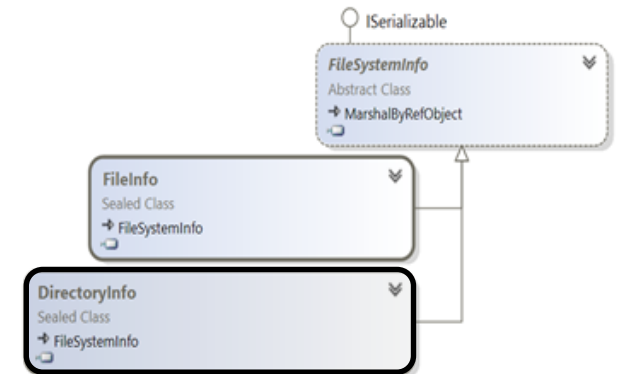
```
// Bind to the current working directory.  
DirectoryInfo dir1 = new DirectoryInfo(".");  
  
// Bind to C:\Windows using a verbatim string  
DirectoryInfo dir2 = new DirectoryInfo(@"C:\Windows");
```



DirectoryInfo.Create()

- Om man försöker att interagera med en folder som inte finns, fås undantaget **System.IO.DirectoryNotFoundException**.
- För att skapa en **ny folder** anges **sökvägen** när konstruktorn anropas, och därefter anropas metoden **Create()** på instansen.

```
// Bind to a nonexistent directory, then create it.  
DirectoryInfo dir3 = new DirectoryInfo(@"C:\MyCode\Testing");  
dir3.Create();
```



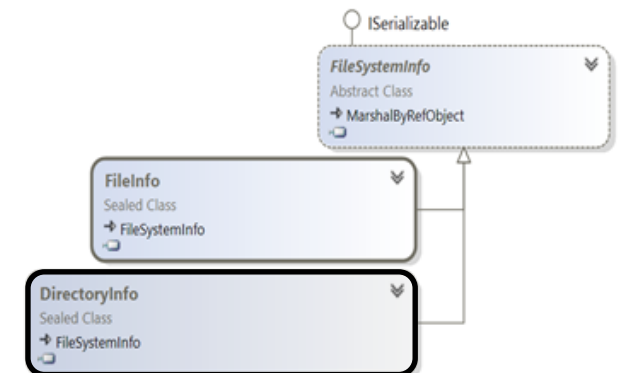
DirectoryInfo egenskaper

- När man har skapat en **DirectoryInfo instans**, kan man använda de **ärvda medlemmarna** från **FileSystemInfo** för att undersöka folderns innehåll.

```
// Dump directory information.
DirectoryInfo dir = new DirectoryInfo(@"C:\Windows");
Console.WriteLine("***** Directory Info *****");
Console.WriteLine("FullName: {0}", dir.FullName);
Console.WriteLine("Name: {0}", dir.Name);
Console.WriteLine("Parent: {0}", dir.Parent);
Console.WriteLine("Creation: {0}", dir.CreationTime);
Console.WriteLine("Attributes: {0}", dir.Attributes);
Console.WriteLine("Root: {0}", dir.Root);
```

```
***** Fun with Directory(Info) *****

***** Directory Info *****
FullName: C:\Windows
Name: Windows
Parent:
Creation: 2018-04-11 23:04:33
Attributes: Directory
Root: C:\
*****
```

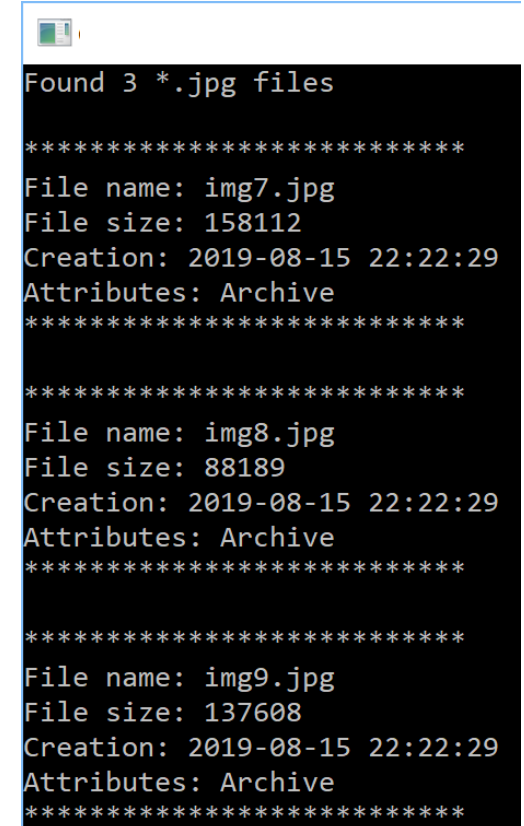


DirectoryInfo.GetFiles()

- **GetFiles()** metoden tar ett **sökmönster** (t.ex. *.jpg) och ett värde från enumen **SearchOptions** (t.ex. **SearchOptions.AllDirectories**) som in-parametrar och returnerar en array med **FileInfo** objekt med information om matchande filer i en folder.

```
DirectoryInfo dir = new DirectoryInfo(@"..\images");  
  
// Get all files with a *.jpg extension.  
FileInfo[] imageFiles = dir.GetFiles("*.jpg", SearchOption.AllDirectories);  
  
// How many were found?  
Console.WriteLine("Found {0} *.jpg files\n", imageFiles.Length);  
  
// Now print out info for each file.  
foreach (FileInfo f in imageFiles)  
{  
    Console.WriteLine("*****");  
    Console.WriteLine("File name: {0}", f.Name);  
    Console.WriteLine("File size: {0}", f.Length);  
    Console.WriteLine("Creation: {0}", f.CreationTime);  
    Console.WriteLine("Attributes: {0}", f.Attributes);  
    Console.WriteLine("*****\n");  
}
```

Punkten motsvarar foldern som innehåller programmens assembly



```
Found 3 *.jpg files  
  
*****  
File name: img7.jpg  
File size: 158112  
Creation: 2019-08-15 22:22:29  
Attributes: Archive  
*****  
  
*****  
File name: img8.jpg  
File size: 88189  
Creation: 2019-08-15 22:22:29  
Attributes: Archive  
*****  
  
*****  
File name: img9.jpg  
File size: 137608  
Creation: 2019-08-15 22:22:29  
Attributes: Archive  
*****
```

DirectoryInfo.CreateSubdirectory()

- Metoden **CreateSubdirectory()** tar en **relativ sökväg** (från aktuell folder) som inparameter och skapar alla subfoldrar i **sökvägen**.
- **CreateSubdirectory()** returnerar ett nytt **DirectoryInfo objekt** som representerar den nya subfoldern.

```
DirectoryInfo dir = new DirectoryInfo(".");

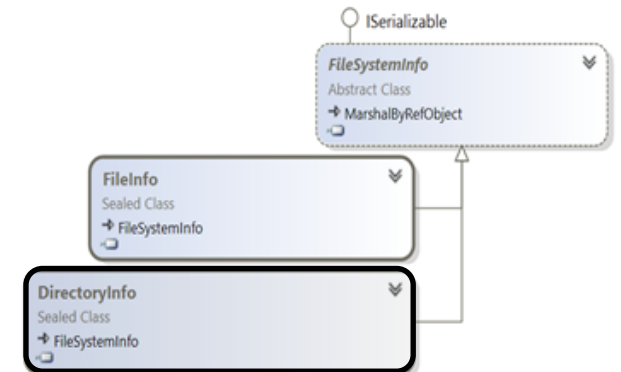
// Create \MyFolder off initial directory.
dir.CreateSubdirectory("MyFolder");

// Capture returned DirectoryInfo object.
DirectoryInfo myDataFolder =
dir.CreateSubdirectory(@"MyFolder2\Data");

// Prints path to ..\MyFolder2\Data.
Console.WriteLine("New Folder is: {0}", myDataFolder);
```



```
New Folder is: C:\F3\F3Code\DirectoryApp\bin\Debug\MyFolder2\Data
```



Directory

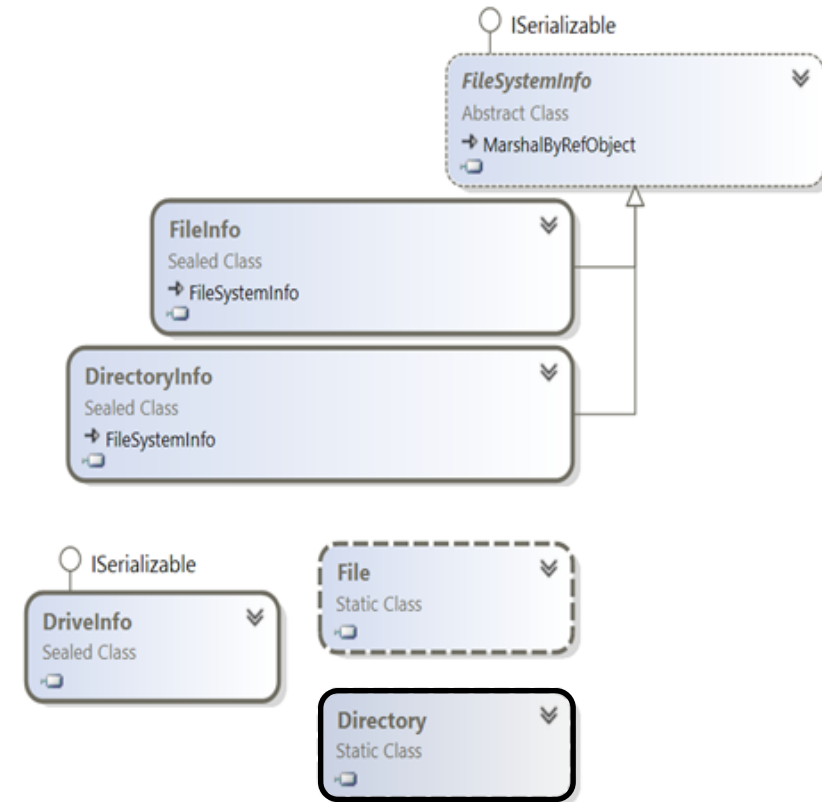
- Den **statiska** klassen **Directory** innehåller liknande funktionalitet som **DirectoryInfo**, fast innehåller **statiska medlemmar** som **returnerar strängar**.
- Metoden **Directory.GetLogicalDrives()** returnerar en lista med datorns diskenheter.
- Metoden **Directory.Delete()** tar en **sökväg till en folder** (samt en optionell **boolesk parameter**), som in-parameter och tar bort foldern (och alla subfoldrar om den booleska parametern är **true**).

```
// List all drives on current computer.
string[] drives = Directory.GetLogicalDrives();
Console.WriteLine("Here are your drives:");
foreach (string s in drives)
    Console.WriteLine("--> {0} ", s);

Directory.Delete(@"C:\MyFolder");
// The second parameter specifies whether you
// wish to destroy any subdirectories.
Directory.Delete(@"C:\MyFolder2", true);
Console.WriteLine("Directories deleted.");
```

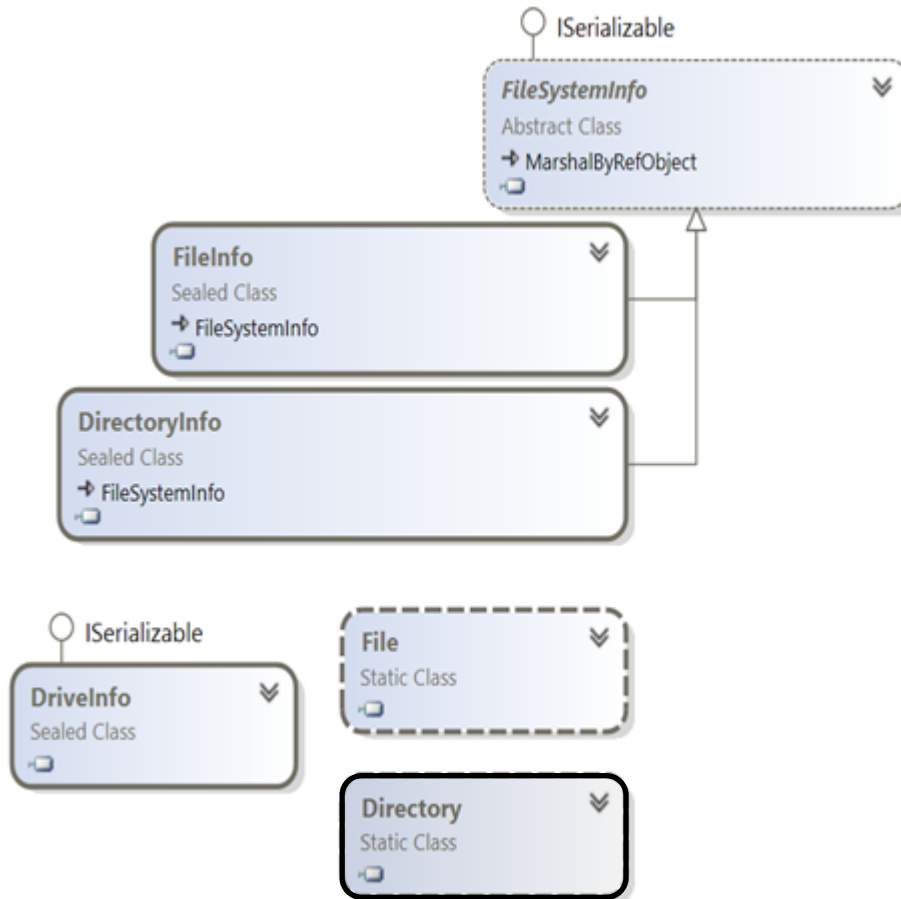


```
Here are your drives:
--> C:\
Directories deleted.
```



Directory

- **Directory** klassen innehåller många fler användbara statiska medlemmar.



Directory
Static Class

Methods

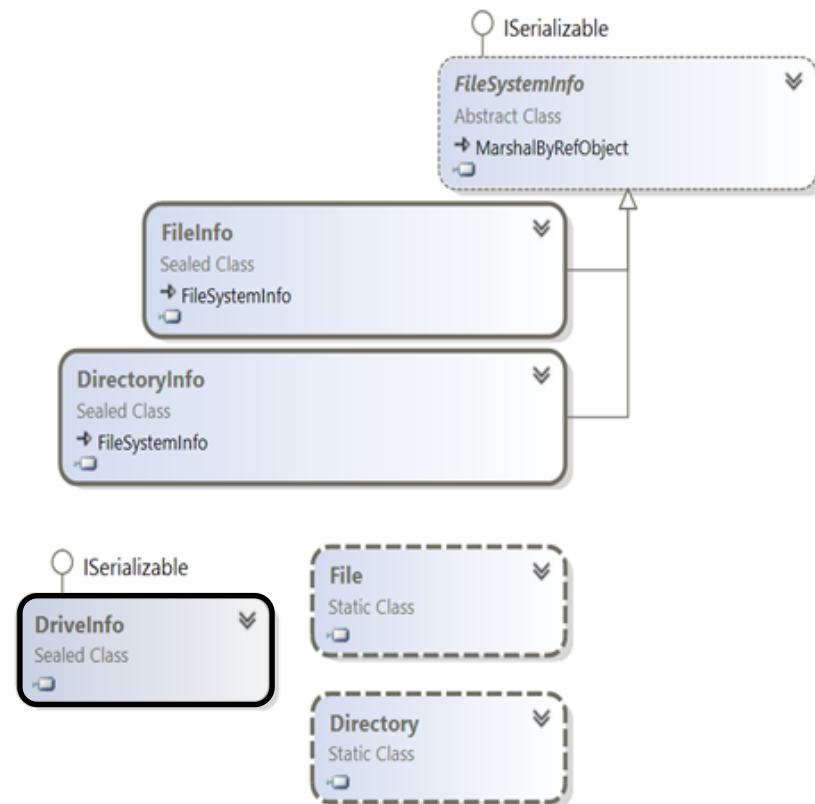
- CreateDirectory (+ 1 overload)
- Delete (+ 1 overload)
- EnumerateDirectories (+ 2 overloads)
- EnumerateFiles (+ 2 overloads)
- EnumerateFileSystemEntries (+ 2 overloads)
- Exists
- GetAccessControl (+ 1 overload)
- GetCreationTime
- GetCreationTimeUtc
- GetCurrentDirectory
- GetDirectories (+ 2 overloads)
- GetDirectoryRoot
- GetFiles (+ 2 overloads)
- GetFileSystemEntries (+ 2 overloads)
- GetLastAccessTime
- GetLastAccessTimeUtc
- GetLastWriteTime
- GetLastWriteTimeUtc
- GetLogicalDrives
- GetParent
- Move
- SetAccessControl
- SetCreationTime
- SetCreationTimeUtc
- SetCurrentDirectory
- SetLastAccessTime
- SetLastAccessTimeUtc
- SetLastWriteTime
- SetLastWriteTimeUtc

DriveInfo

- **DriveInfo** klassen innehåller en **statisk** metod **DriveInfo.GetDrives()** som också listar datorns enheter, men returnerar **DriveInfo objekt** med **detaljerad information** om enheterna, t.ex. enhetstyp, ledigt utrymme och namnet på enheten (volymen).

```
// Get info regarding all drives.
DriveInfo[] myDrives = DriveInfo.GetDrives();

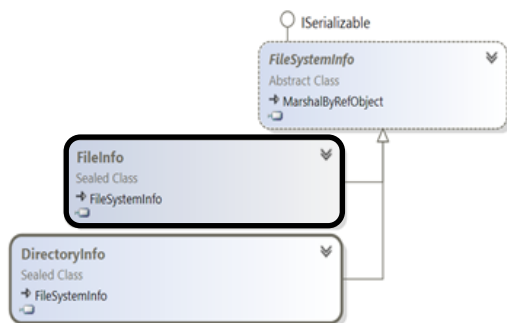
// Now print drive stats.
foreach(DriveInfo d in myDrives)
{
    Console.WriteLine("Name: {0}", d.Name);
    Console.WriteLine("Type: {0}", d.DriveType);
    // Check to see whether the drive is mounted.
    if(d.IsReady)
    {
        Console.WriteLine("Free space: {0}", d.TotalFreeSpace);
        Console.WriteLine("Format: {0}", d.DriveFormat);
        Console.WriteLine("Label: {0}", d.VolumeLabel);
    }
}
```



```
Name: C:\
Type: Fixed
Free space: 17018589184
Format: NTFS
Label: Local Disk
```

FileInfo

- I exemplet med **DirectoryInfo.GetFiles()** returnerades en **FileInfo** instans för varje fil i en folder, där **FileInfo** objekten innehöll information om bl.a. namn, längd och skapelsetid.
- **FileInfo** ärver funktionalitet från **FileSystemInfo** och innehåller ytterligare medlemmar för att bl.a. **skapa**, **öppna**, **kopiera**, **flytta** och **ta bort** filer, samt för att ta reda på en fils **egenskaper**.
- Många metoder returnerar **I/O specifika typer** (t.ex. **FileStream**, **StreamWriter** och **StreamReader**) som används för att **skriva/läsa till/från en fil**.

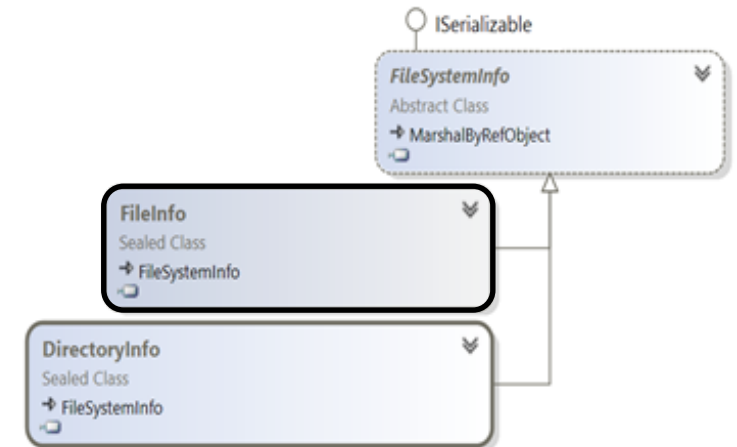


Member	Meaning in Life
AppendText()	Creates a StreamWriter object (described later) that appends text to a file
CopyTo()	Copies an existing file to a new file
Create()	Creates a new file and returns a FileStream object (described later) to interact with the newly created file
CreateText()	Creates a StreamWriter object that writes a new text file
Delete()	Deletes the file to which a FileInfo instance is bound
Directory	Gets an instance of the parent directory
DirectoryName	Gets the full path to the parent directory
Length	Gets the size of the current file
MoveTo()	Moves a specified file to a new location, providing the option to specify a new file name
Name	Gets the name of the file
Open()	Opens a file with various read/write and sharing privileges
OpenRead()	Creates a read-only FileStream object
OpenText()	Creates a StreamReader object (described later) that reads from an existing text file
OpenWrite()	Creates a write-only FileStream object

FileInfo.Create()

- **FileInfo** konstruktorn tar en **sökväg till en fil** som in-parameter.
- Om inte filen finns, måste den först skapas med instansmetoden **Create()**, som returnerar ett **FileStream** objekt.
- **FileStream** objektet (en så kallad **ström**) innehåller **metoder** för att **läsa/skriva från/till filen**
- När man är klar med **FileStream** objektet, **måste den "stängas"** med metoden **Dispose()**.

```
// Make a new file on the C drive.  
FileInfo f = new FileInfo(@"C:\Test.dat");  
FileStream fs = f.Create();  
  
// Use the FileStream object...  
  
// Close down file stream.  
fs.Dispose();
```

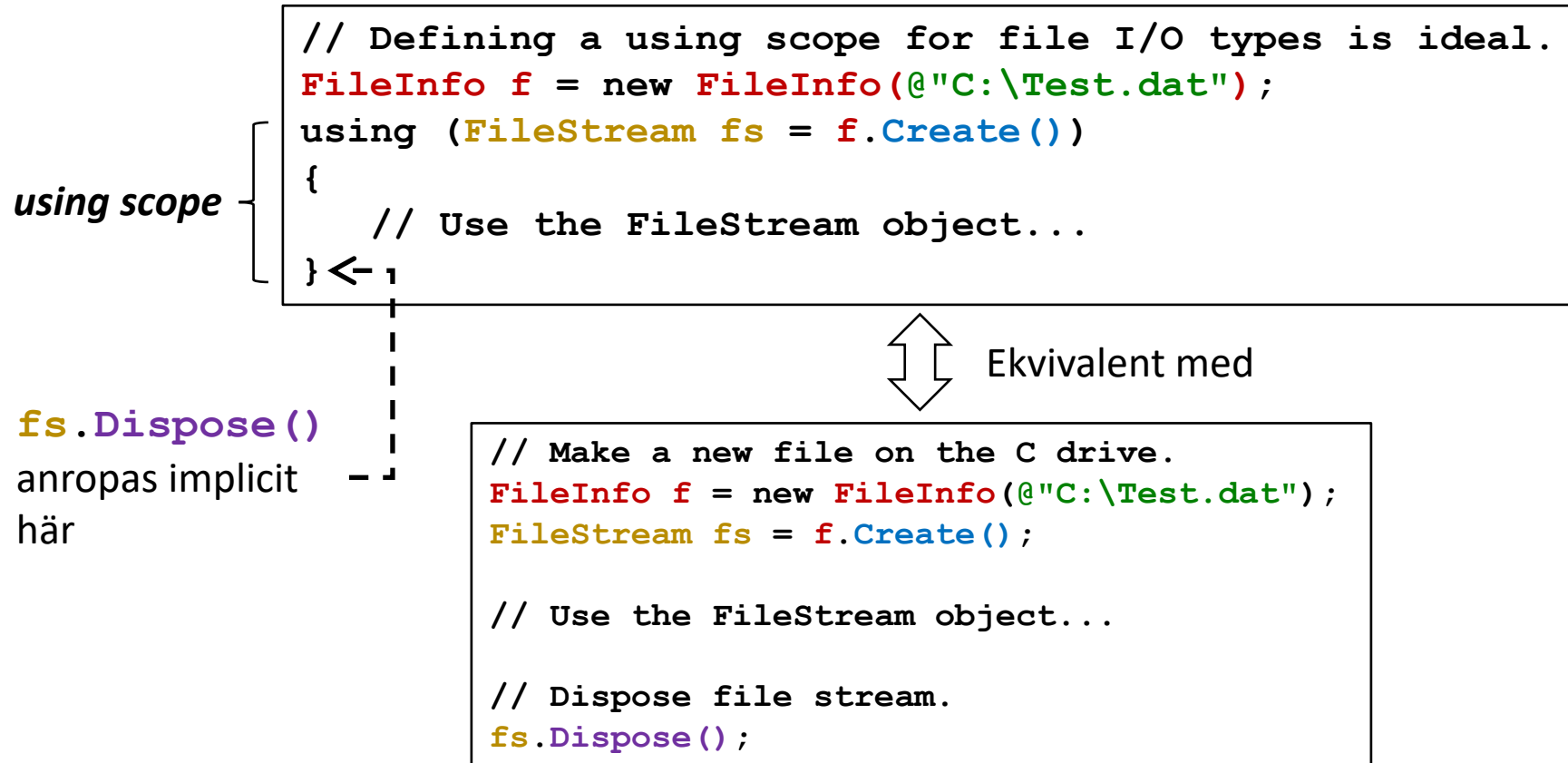


IDisposable och *using*

- Alla typer som använder operativsystemets resurser (t.ex. strömmar, sockets, och databaskopplingar) måste “stänga” dessa när de är färdiga med dem, så att resurserna kan “återlämnas” till operativsystemet.
- Dessa typerna implementerar interfacet **IDisposable** som innehåller metoden **Dispose()**.
- **FileStream** implementerar **IDisposable** och måste stängas via metoden **Dispose()**.
- Dock finns det en konstruktion i C#, ett så kallat *using scope*, som **automatiskt (implicit)** **anropar Dispose()** på ett **IDisposable objekt** när objektet lämnar *using scopet*.
- Ett *using scope* skapas med syntaxen:

```
using( /* create IDisposable instance here */ )  
{  
  
    /* use IDisposable instance here */  
  
} // <-- IDisposable.Dispose() is implicitly called here
```

IDisposable och using



FileInfo.Open()

- **Open()** är en mycket bättre metod för att både skapa nya filer och öppna existerande filer.
- Metoden är överlagrad, där en av dessa tar tre enumar som in-parametrar:

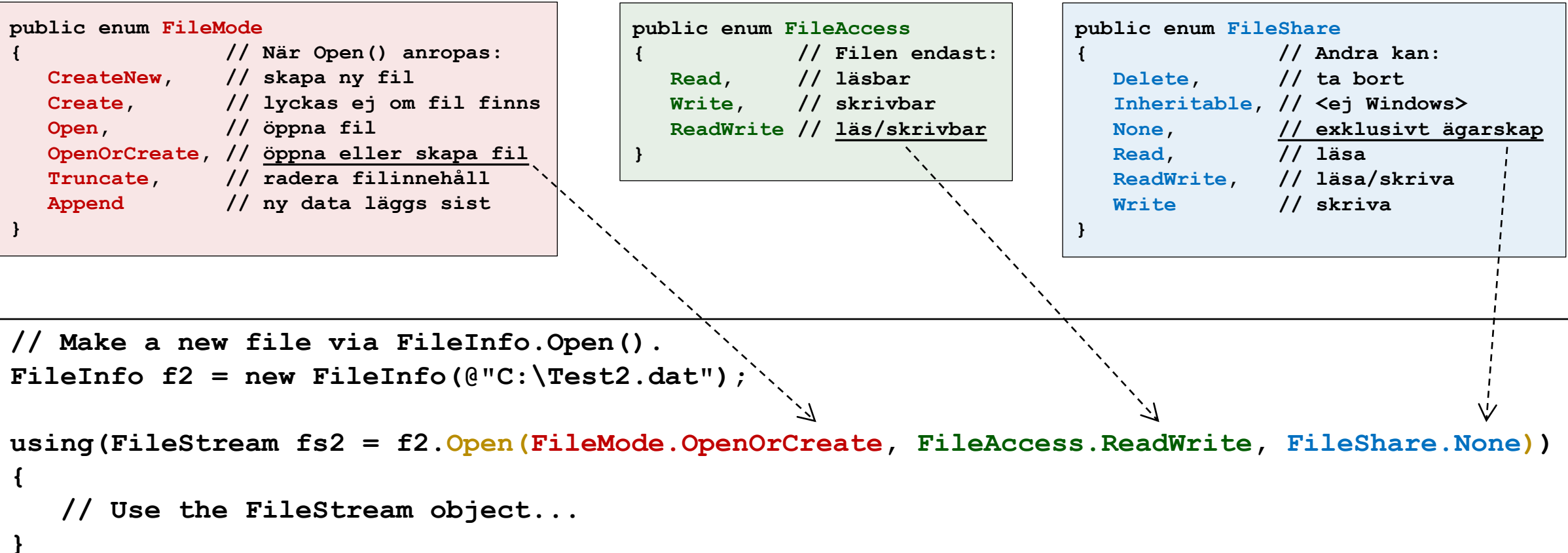
```
public enum FileMode
{
    CreateNew,    // När Open() anropas:
                 // skapa ny fil
    Create,       // lyckas ej om fil finns
    Open,         // öppna fil
    OpenOrCreate, // öppna eller skapa fil
    Truncate,     // radera filinnehåll
    Append        // ny data läggs sist
}
```

```
public enum FileAccess
{
    Read,    // Filen endast:
             // läsbar
    Write,    // skrivbar
    ReadWrite // läs/skrivbar
}
```

```
public enum FileShare
{
    Delete,    // Andra kan:
              // ta bort
    Inheritable, // <ej Windows>
    None,        // exklusivt ägarskap
    Read,        // läsa
    ReadWrite,   // läsa/skriva
    Write        // skriva
}
```

```
// Make a new file via FileInfo.Open().
FileInfo f2 = new FileInfo(@"C:\Test2.dat");

using(FileStream fs2 = f2.Open(FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.None))
{
    // Use the FileStream object...
}
```

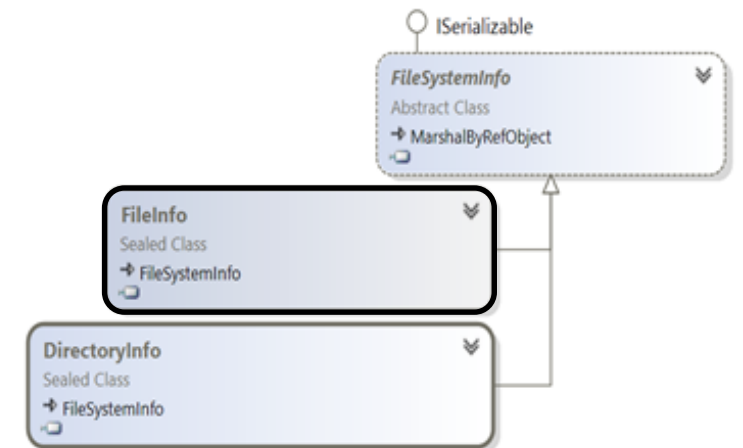


FileInfo.OpenRead() och FileInfo.OpenWrite()

- För att slippa ange enum-värden när man öppnar en fil för läsning eller skrivning, finns även metoderna **OpenRead()** och **OpenWrite()** tillgängliga.
- **OpenRead()** returnerar en **endast läsbar filström (FileStream)**.
- **OpenWrite()** returnerar en **endast skrivbar filström (FileStream)**.

```
// Get a FileStream object with read-only permissions.
FileInfo f3 = new FileInfo(@"C:\Test3.dat");
using(FileStream readOnlyStream = f3.OpenRead())
{
    // Use the FileStream object...
}

// Now get a FileStream object with write-only permissions.
FileInfo f4 = new FileInfo(@"C:\Test4.dat");
using(FileStream writeOnlyStream = f4.OpenWrite())
{
    // Use the FileStream object...
}
```



OpenText(), CreateText() och AppendText()

- Till skillnad mot **Create()**, **Open()**, **OpenRead()** och **OpenWrite()** som returnerar en instans av typ **FileStream**, returnerar **OpenText()** en **StreamReader**, samt **CreateText()** och **AppendText()** en **StreamWriter**.
- En **FileStream** används för att **läsa/skriva bytes**.
- En **StreamReader** används för att **läsa tecken/text** (*character data*).
- En **StreamWriter** används för att **skriva tecken/text** (*character data*).

```
FileInfo f5 = new FileInfo(@"C:\boot.ini");
using(StreamReader sreader = f5.OpenText())
{
    // Use the StreamReader object to read text ...
}

FileInfo f6 = new FileInfo(@"C:\Test6.txt");
using(StreamWriter swriter = f6.CreateText())
{
    // Use the StreamWriter object to write text ...
}

FileInfo f7 = new FileInfo(@"C:\FinalTest.txt");
using(StreamWriter swriterAppend = f7.AppendText())
{
    // Use the StreamWriter object to append text ...
}
```


File

- **File** innehåller liknande funktionalitet som **FileInfo**, fast innehåller **statiska medlemmar** för **Create()**, **Open()**, **OpenRead()**, **OpenWrite()**, **OpenText()**, **CreateText()** och **AppendText()**.

```
// Obtain FileStream object via File.Create().
using(FileStream fs = File.Create(@"C:\Test.dat")) {}

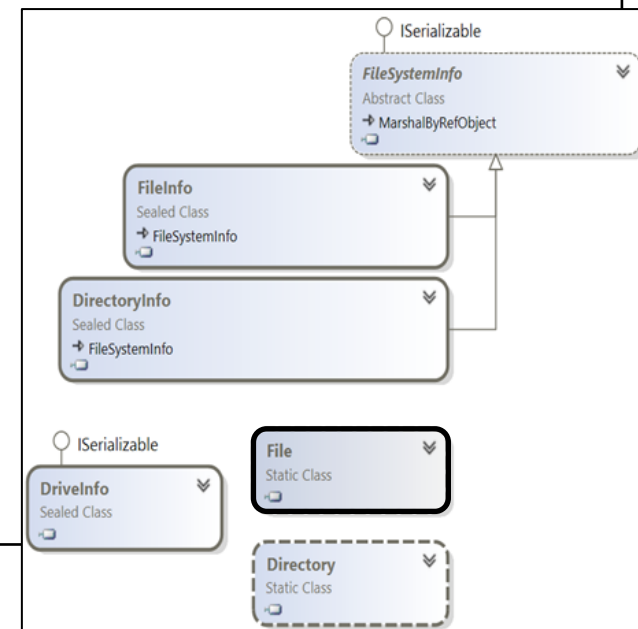
// Obtain FileStream object via File.Open().
using(FileStream fs2 = File.Open(@"C:\Test2.dat", FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.None)) {}

// Get a FileStream object with read-only permissions.
using(FileStream readOnlyStream = File.OpenRead(@"Test3.dat")) {}

// Get a FileStream object with write-only permissions.
using(FileStream writeOnlyStream = File.OpenWrite(@"Test4.dat")) {}

// Get a StreamReader object.
using(StreamReader sreader = File.OpenText(@"C:\boot.ini")) {}

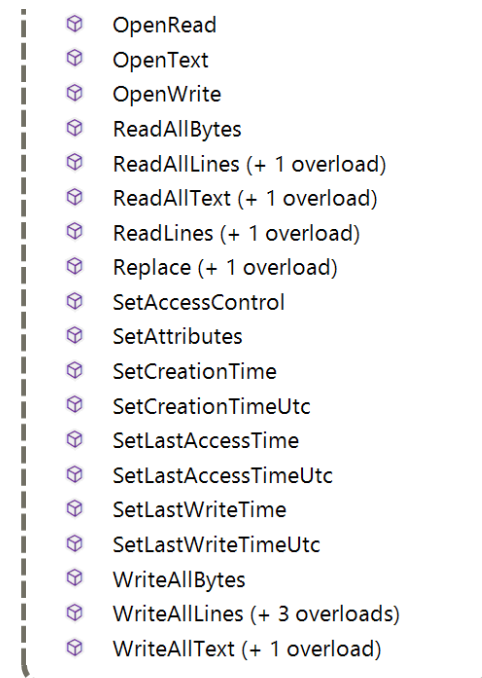
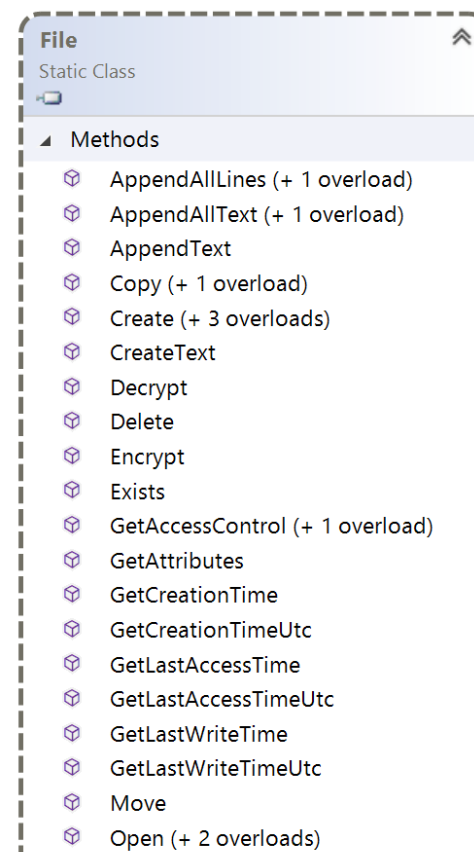
// Get some StreamWriters.
using(StreamWriter swriter = File.CreateText(@"C:\Test6.txt")) {}
using(StreamWriter swriterAppend = File.AppendText(@"C:\FinalTest.txt")) {}
```



File

- **File** klassen innehåller många fler användbara statiska medlemmar.
- **Följande statiska metoder stänger även filen automatiskt efter anropet.**

Method	Meaning in Life
<code>ReadAllBytes()</code>	Opens the specified file, returns the binary data as an array of bytes, and then closes the file
<code>ReadAllLines()</code>	Opens a specified file, returns the character data as an array of strings, and then closes the file
<code>ReadAllText()</code>	Opens a specified file, returns the character data as a <code>System.String</code> , and then closes the file
<code>WriteAllBytes()</code>	Opens the specified file, writes out the byte array, and then closes the file
<code>WriteAllLines()</code>	Opens a specified file, writes out an array of strings, and then closes the file
<code>WriteAllText()</code>	Opens a specified file, writes the character data from a specified string, and then closes the file



File

- Exempelvis, stänger metoderna **WriteAllLines()** och **ReadAllLines()** filen automatiskt.
- **WriteAllLines()** öppnar en fil, skriver en **array av strängar** till den, samt stänger den.
- **ReadAllLines()** öppnar en fil, läser innehållet till en **array av strängar**, samt stänger filen.

```
string[] myTasks = {"Fix bathroom sink", "Call Dave", "Call Mom and Dad", "Play Xbox One"};

// Write out all data to file on C drive.
File.WriteAllLines(@"tasks.txt", myTasks);

// Read it all back and print out.
foreach (string task in File.ReadAllLines(@"tasks.txt"))
{
    Console.WriteLine("TODO: {0}", task);
}
```



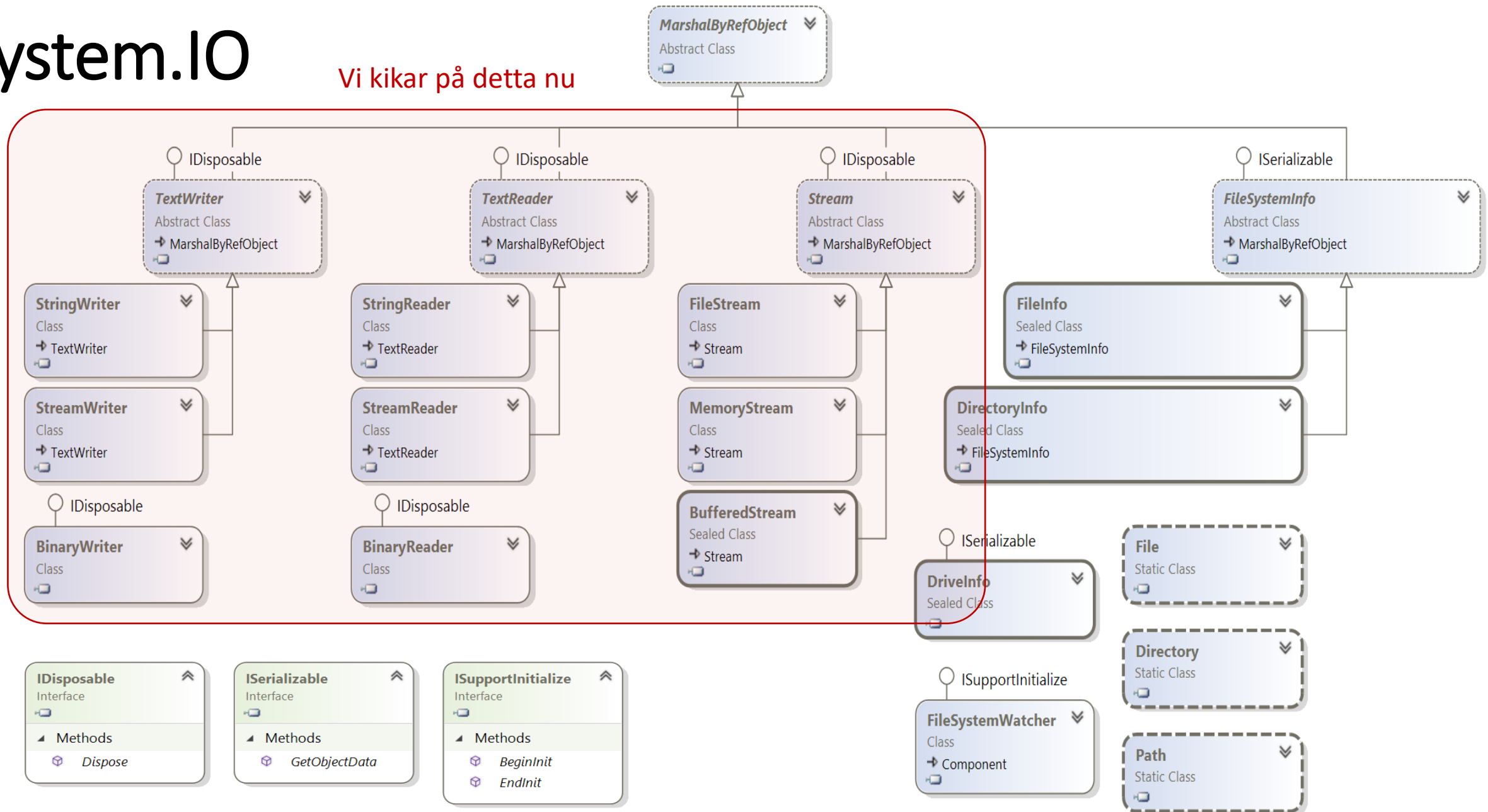
```
***** Simple IO with the File Type *****

TODO: Fix bathroom sink
TODO: Call Dave
TODO: Call Mom and Dad
TODO: Play Xbox 360

Press <Enter> to exit ...
```

System.IO

Vi kikar på detta nu

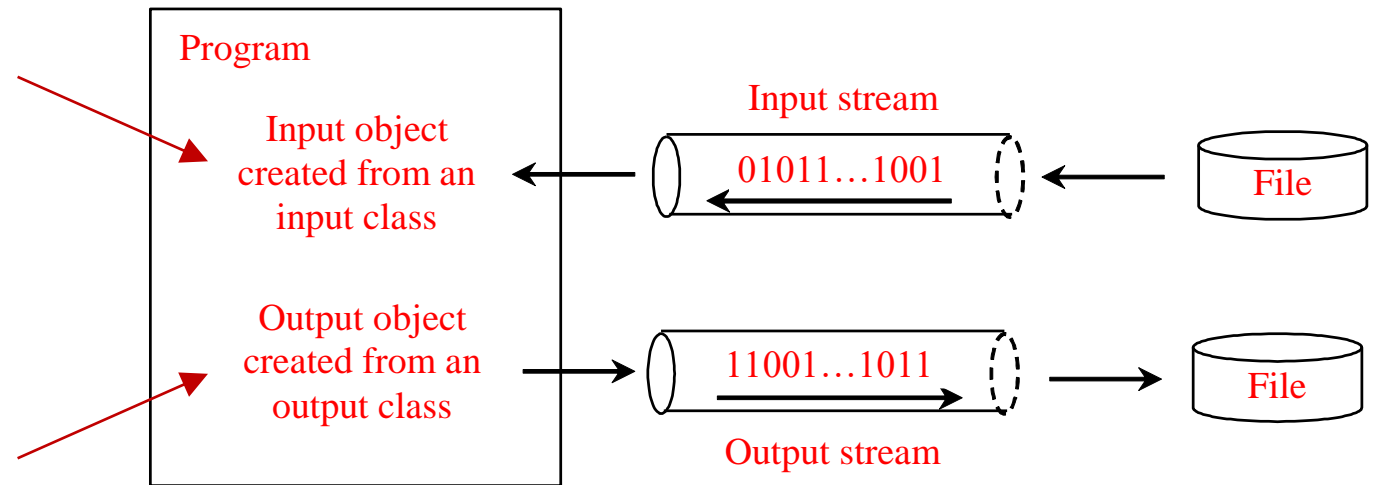


Input/Output (I/O) i .NET

- **FileInfo** objekt innehåller information om en fil, men innehåller inga metoder för att läsa/skriva data från/till en fil. För att kunna utföra I/O (**läs/skriv**) operationer, måste en lämplig instans av .NETs **Strömmar** skapas.

```
FileInfo file = new FileInfo("temp.txt");  
StreamReader input = file.OpenText();  
String s = input.ReadLine();
```

```
FileInfo file = new FileInfo("temp.txt");  
StreamWriter output = file.CreateText();  
output.WriteLine("C# 101");
```

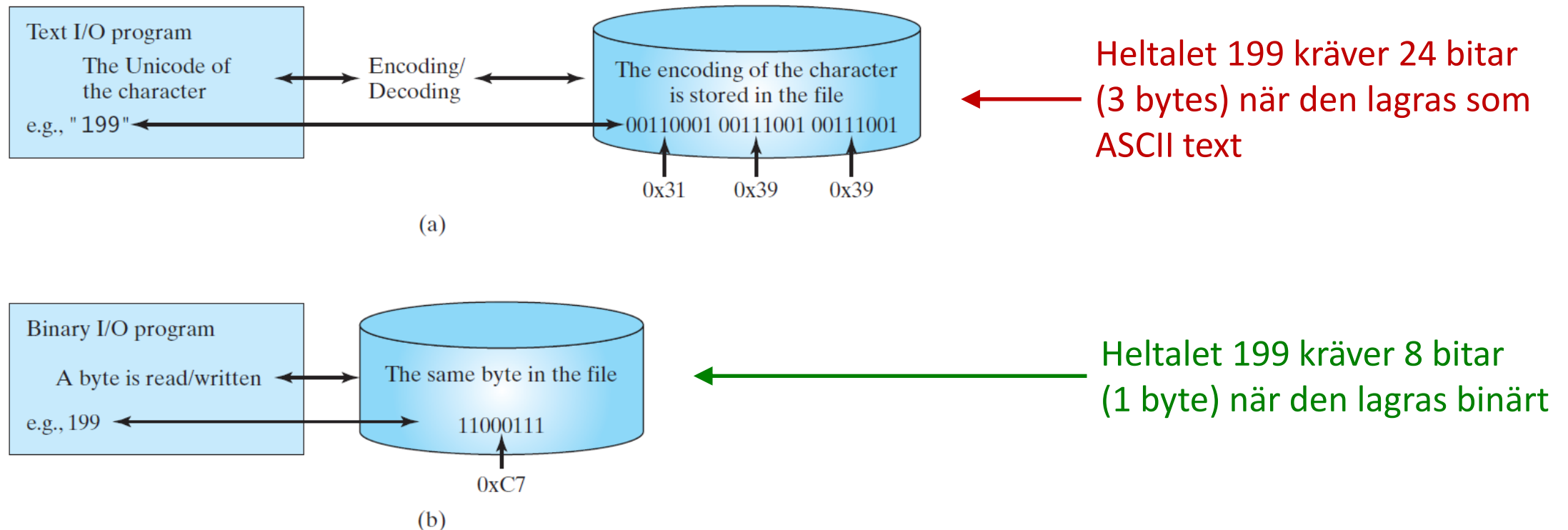


Textfiler och binärfiler

- Data som lagras som **textfiler kan läsas av en människa** via en texteditor.
- Data som lagras som **binärfiler kan inte läsas av en människa** via en texteditor.
- **Binärfiler** är designade att läsas av ett program, och **processas** därför mycket **effektivare (snabbare)** än textfiler.
- Exempelvis lagras C# källkod som **textfiler** (.cs) och kan därför läsas av en människa via en källkodseditor (t.ex. editorn i Visual Studio), medan C# kompilerad kod lagras som bytekod i **binärfiler** (.dll, .exe) och kan därför processas mycket effektivare av *CLR (Common Language Runtime)*.
- Exempelvis lagras det decimala heltalet 199 som tre tecken '1', '9', '9' i en **textfil** och som en byte (0xC7, 11000111b) i en **binärfil**.

Character (text) och binär I/O i .NET

- **Text I/O** innebär att CLR måste konvertera mellan **Unicode** och ett filspecifikt format (t.ex. ASCII, UTF8) vid skrivning till en textfil, och från ett filspecifikt format till **Unicode** vid läsning från en textfil (ett 16-bitars **Unicode** tecken i .NET kan lagras i en **char**).
- **Binär I/O** innebär ingen konvertering (en byte läses/skrivs direkt från/till en binärfil).



Character (text) och binär I/O Klasser

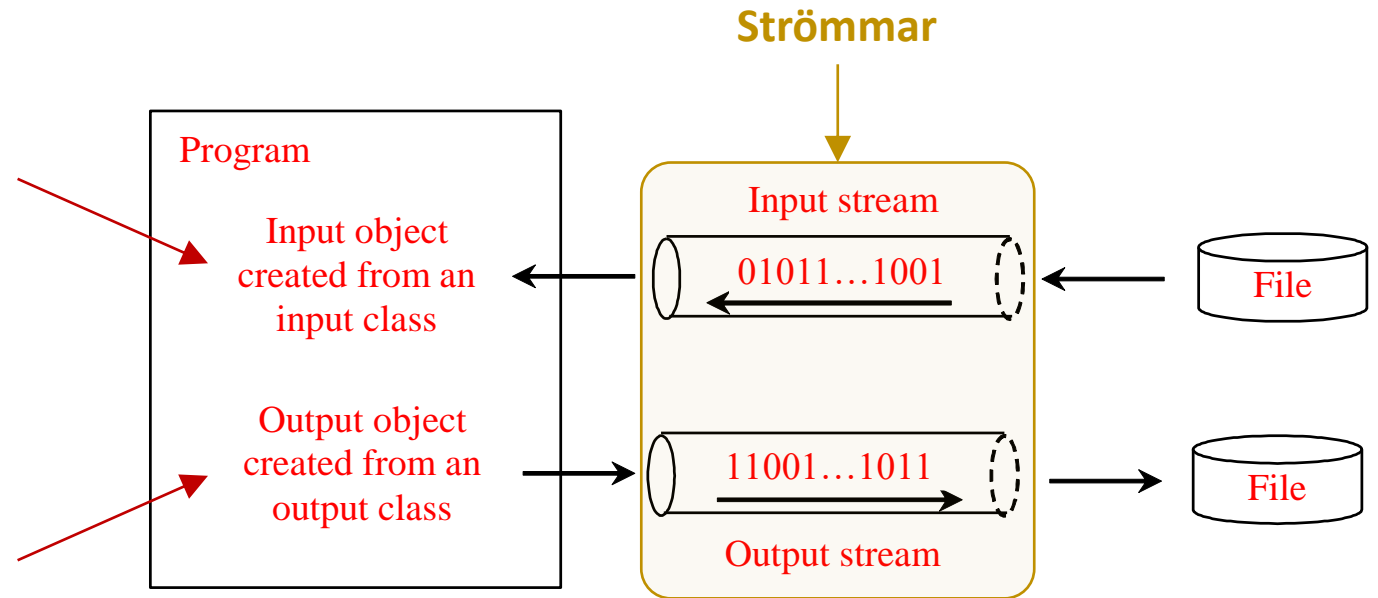
- .NET använder abstraktionen **ström (stream)** för att beskriva sina I/O klasser.
- En **ström** en abstraktion som kan **läsa från en källa** och/eller **skriva till en sänka**.
- En källa/sänka kan t.ex. vara en fil, en array, en nätverks-socket, mm.
- I .NET finns två typer av strömmar:
 - **Text-baserade strömmar** (character streams) som **läser/skriver tecken**.
 - **Binärströmmar** (binary streams) som **läser/skriver bytes**.
- För att läsa från en källa eller skriva till en sänka, **kopplar** man en lämplig **instans av en ström till källan eller sänkan**.

Character (text) och binär I/O Klasser

- För att läsa text från källan `FileInfo("temp.txt")`, kan t.ex. **StreamReader** användas.
- För att skriva text till säkan `FileInfo("temp.txt")`, kan t.ex. **StreamWriter** användas.

```
FileInfo file = new FileInfo("temp.txt");  
StreamReader input = file.OpenText();  
String s = input.ReadLine();
```

```
FileInfo file = new FileInfo("temp.txt");  
StreamWriter output = file.CreateText();  
output.WriteLine("C# 101");
```



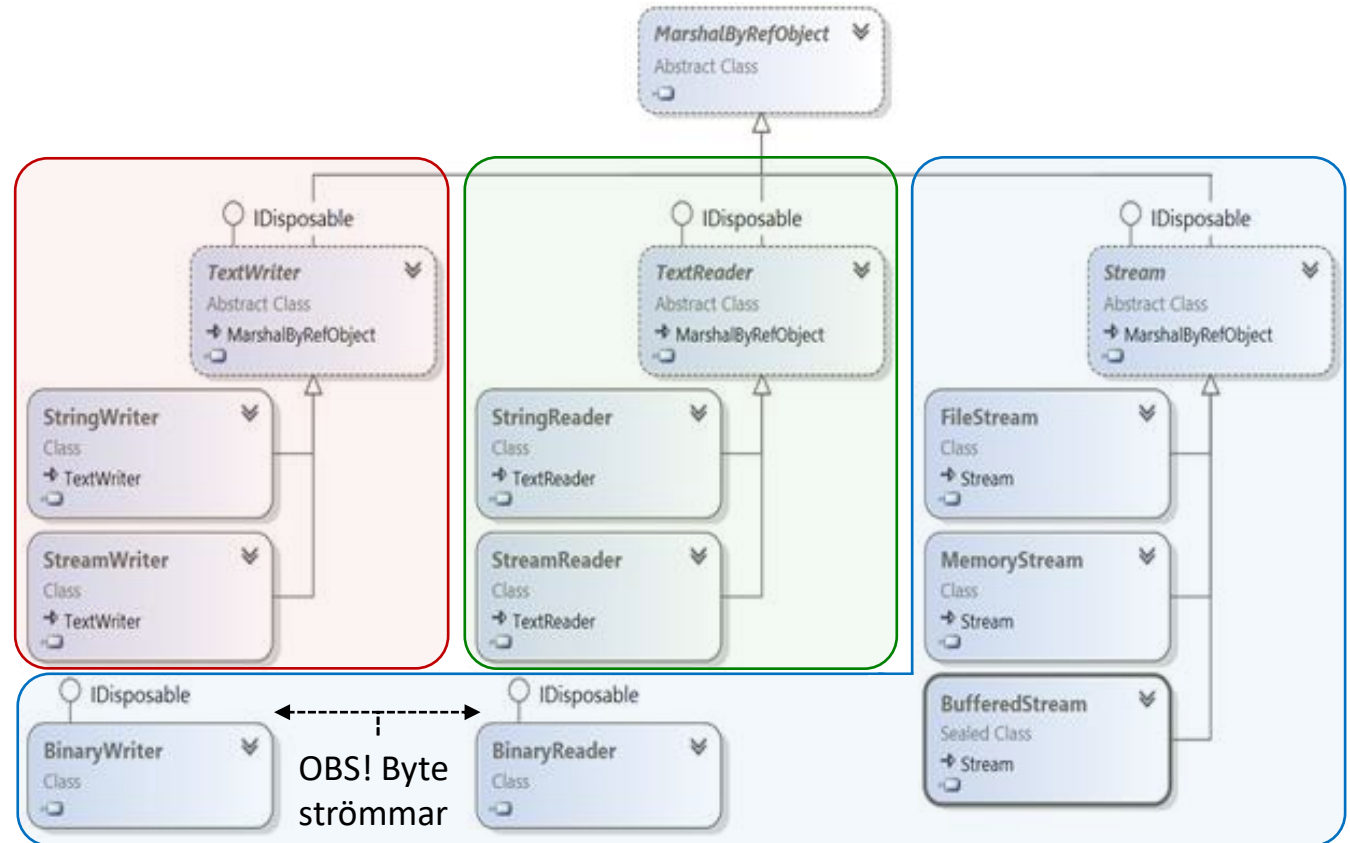
Character (text) och binär I/O Klasser

- .NETs I/O klasser (strömmar) finns i namespace **System.IO**.
- .NETs **text-baserade** I/O klasser slutar generellt med ...**Reader** eller ...**Writer**
 - En **Reader** är en **text-baserad input ström** (kan läsa tecken från en källa, t.ex. en fil).
 - En **Writer** är en **text-baserad output ström** (kan skriva tecken till en sänka, t.ex. en fil).
- .NETs **binära** I/O klasser slutar generellt med ...**Stream**
 - En **Stream** är en **binär input eller output ström** (kan läsa/skriva bytes från/till en källa/sänka).
- Det finns motsvarande klasser för binära och text-baserade strömmar (t.ex. **StreamReader** som är **text-baserad** kontra **FileStream** som är **binär**).

.NETs I/O klasshierarki

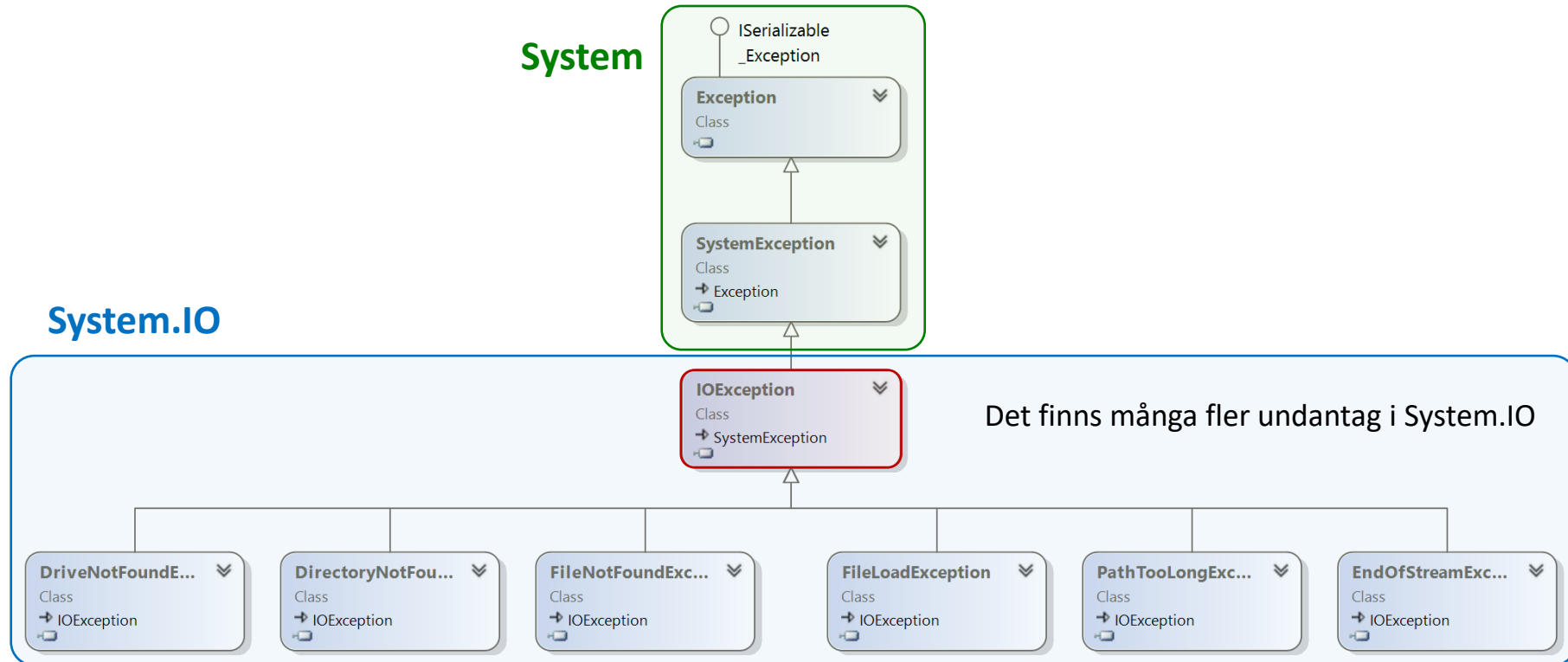
Samtliga I/O klasser finns i **System.IO**.

- **TextReader**
basklass för text-baserad input.
- **TextWriter**
basklass för text-baserad output.
- **Stream**
basklass för binär input & output.



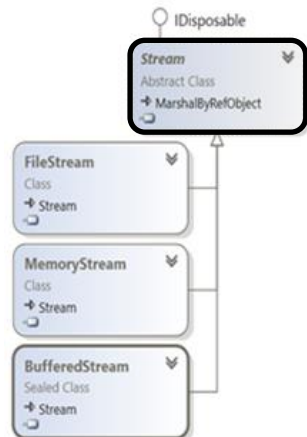
Klasshierarkin för .NETs I/O undantag

- De flesta I/O undantag finns i **System.IO**, där **IOException** utgör basklassen.



Stream

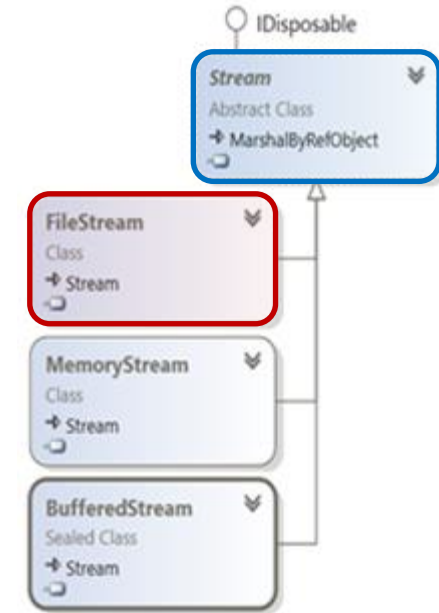
- Den **abstrakta** klassen **System.IO.Stream** utgör **basklassen för binära strömmar**.
- **System.IO.Stream** och dess subklasser representerar en **ström av bytes** som strömmar mellan en källa och en sänka.
- Subklasser ärver ett antal medlemmar från **System.IO.Stream**, t.ex. metoderna **Read()** och **Write()** för att **läsa/skriva** bytes och **Seek()** för att **söka** i en ström.



Member	Meaning in Life
CanRead CanWrite CanSeek	Determines whether the current stream supports reading, seeking, and/or writing.
Close()	Closes the current stream and releases any resources (such as sockets and file handles) associated with the current stream. Internally, this method is aliased to the Dispose() method; therefore, <i>closing a stream</i> is functionally equivalent to <i>disposing a stream</i> .
Flush()	Updates the underlying data source or repository with the current state of the buffer and then clears the buffer. If a stream does not implement a buffer, this method does nothing.
Length	Returns the length of the stream in bytes.
Position	Determines the position in the current stream.
Read() ReadByte() ReadAsync()	Reads a sequence of bytes (or a single byte) from the current stream and advances the current position in the stream by the number of bytes read.
Seek()	Sets the position in the current stream.
SetLength()	Sets the length of the current stream.
Write() WriteByte() WriteAsync()	Writes a sequence of bytes (or a single byte) to the current stream and advances the current position in this stream by the number of bytes written.

FileStream

- **FileStream** ärver från **Stream** och tillför funktionalitet för att läsa/skriva binärfiler.
- **FileStream** kan läsa/skriva **enstaka bytes** eller **bytearrayer**.
- Eftersom en **FileStream** endast arbetar med bytes, måste t.ex. en sträng (som innehåller ett antal **chars** i 16-bitars Unicode format) kodas till en motsvarande byte array vid skrivning till en binärfil, samt avkodas när strängen återskapas från filen.
- Namespace **System.Text** innehåller en typ **Encoding** som kan koda/avkoda strängar.
- När strängen har **kodats**, skrivs byte arrayen till binärfilen med metoden **Write()**.
- För att läsa tillbaka bytes används metoden **Read()**, som **avkodas** till en sträng.
- Om man använder samma **FileStream** instans för att skriva och sedan läsa samma fil, måste **filpekaren** (positionen i byteströmmen) återställas till första positionen med propertyn **Position**.



FileStream (exempel)

```
// Obtain a FileStream object.
using(FileStream fStream = File.Open(@"myMessage.dat", FileMode.Create))
{
    // Encode a string as an array of bytes.
    string msg = "Hello!";
    byte[] msgAsByteArray = Encoding.Default.GetBytes(msg);

    // Write byte[] to file.
    fStream.Write(msgAsByteArray, 0, msgAsByteArray.Length);

    // Reset internal position of stream.
    fStream.Position = 0;

    // Read the bytes from file and display to console.
    Console.WriteLine("Your message as an array of bytes: ");
    byte[] bytesFromFile = new byte[msgAsByteArray.Length];
    for (int i = 0; i < msgAsByteArray.Length; i++)
    {
        bytesFromFile[i] = (byte)fStream.ReadByte();
        Console.WriteLine(bytesFromFile[i]);
    }

    // Display decoded messages.
    Console.WriteLine("\nDecoded Message: ");
    Console.WriteLine(Encoding.Default.GetString(bytesFromFile));
}
```



Your message as an array of bytes: 7210110810811133
Decoded Message: Hello!

Erhåll en **FileStream fStream** via **File.Open()**

Koda strängen med **Encoding.Default.GetBytes()**

Skriv bytearrayen till filen med **fStream.Write()**

Återställ filpekaren med **fStream.Position**

Läs en byte i taget från filen med **fStream.ReadByte()** och spara i en bytearray

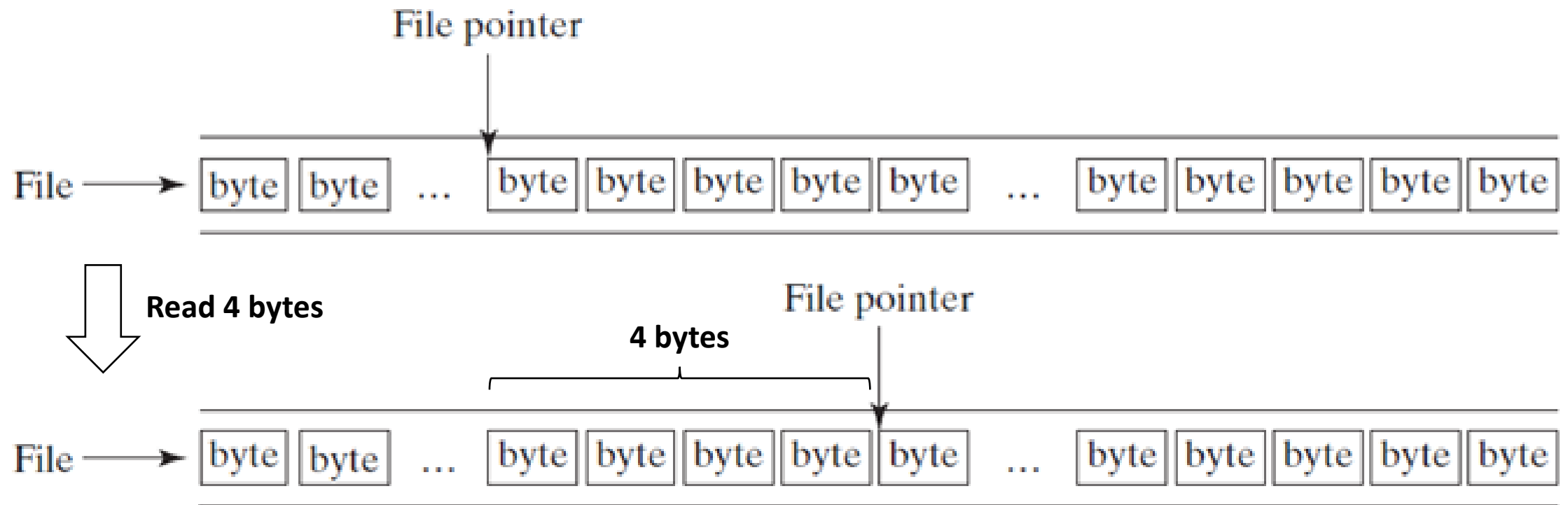
Avkoda bytearrayen till en sträng med **Encoding.Default.GetString()**

Filpekare

- En speciell pekare, kallad ***filpekaren***, pekar alltid på en viss byte i filen.
- En läs-/skrivoperation utförs alltid för den byte som filpekaren pekar på.
- När en fil öppnas, pekar filpekaren på första byten i filen.
- När en läs-/skrivoperation utförs, flyttas filpekaren automatiskt fram i filen.

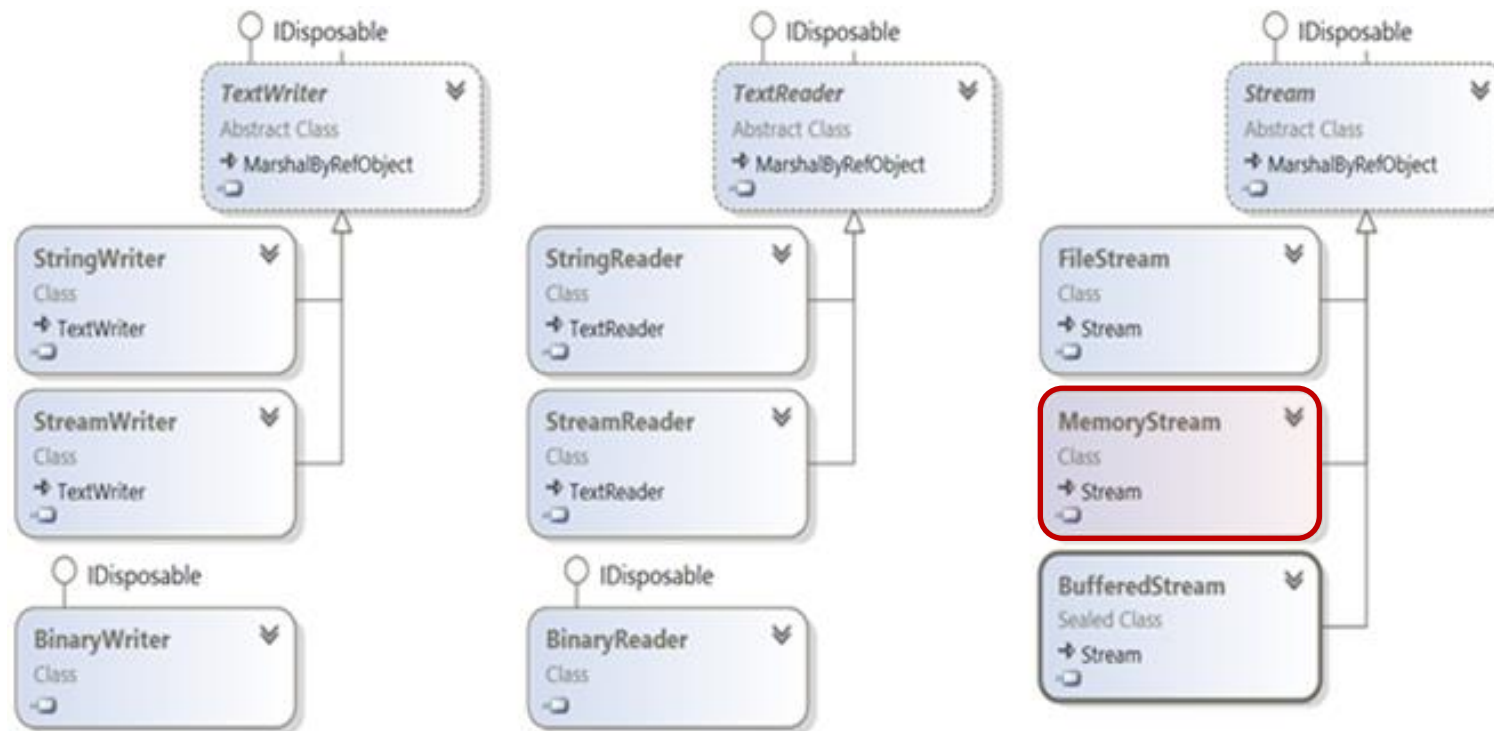
Filpekare

- Exempelvis om 4 bytes läses, med start från filpekarens nuvarande position, har filpekaren flyttats fram 4 bytes.



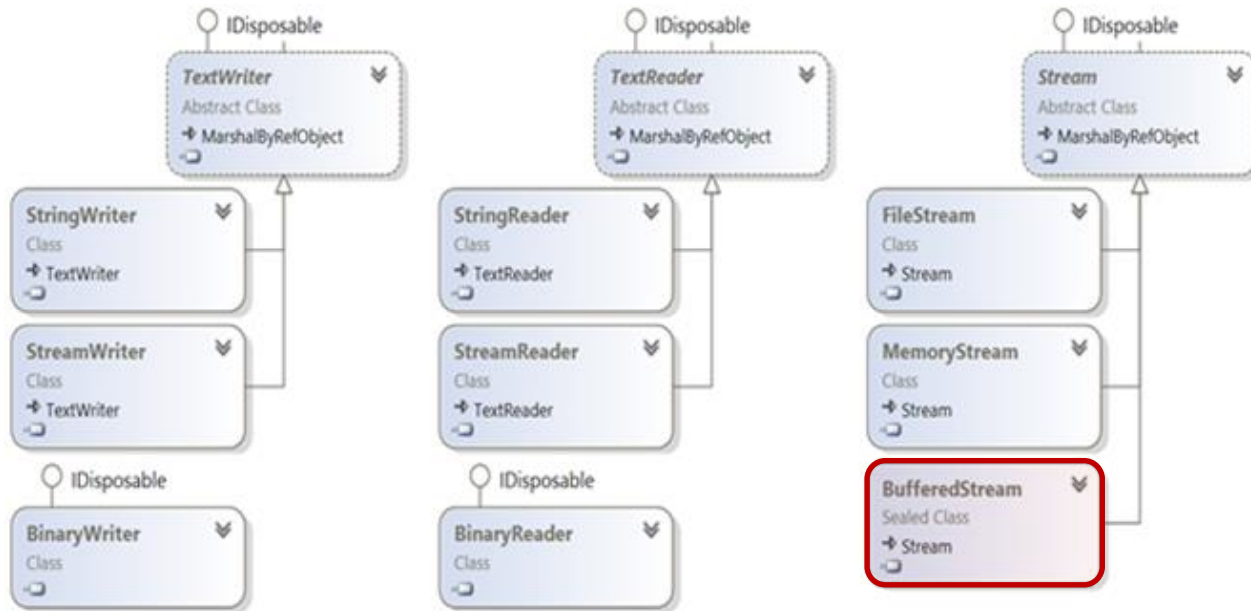
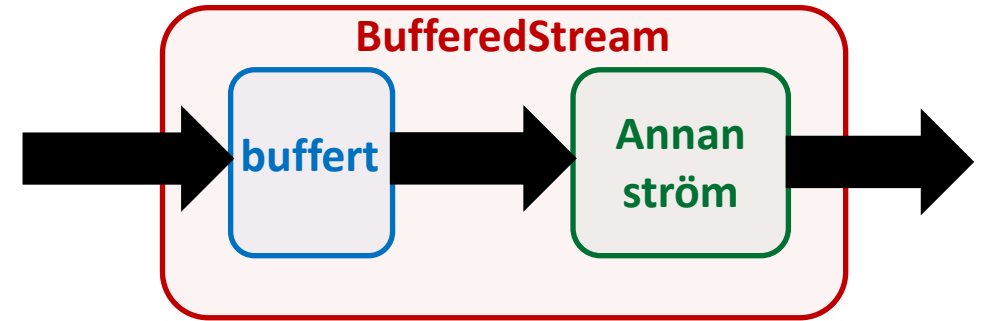
Andra byte-baserade I/O typer

- Samma teknik används för byte-baserade strömmar till/från andra källor, t.ex.
 - Läsa/skriva till/från minne (**MemoryStream**).
 - Läsa/skriva till/från en socket (via typer i **System.Net.Sockets**).



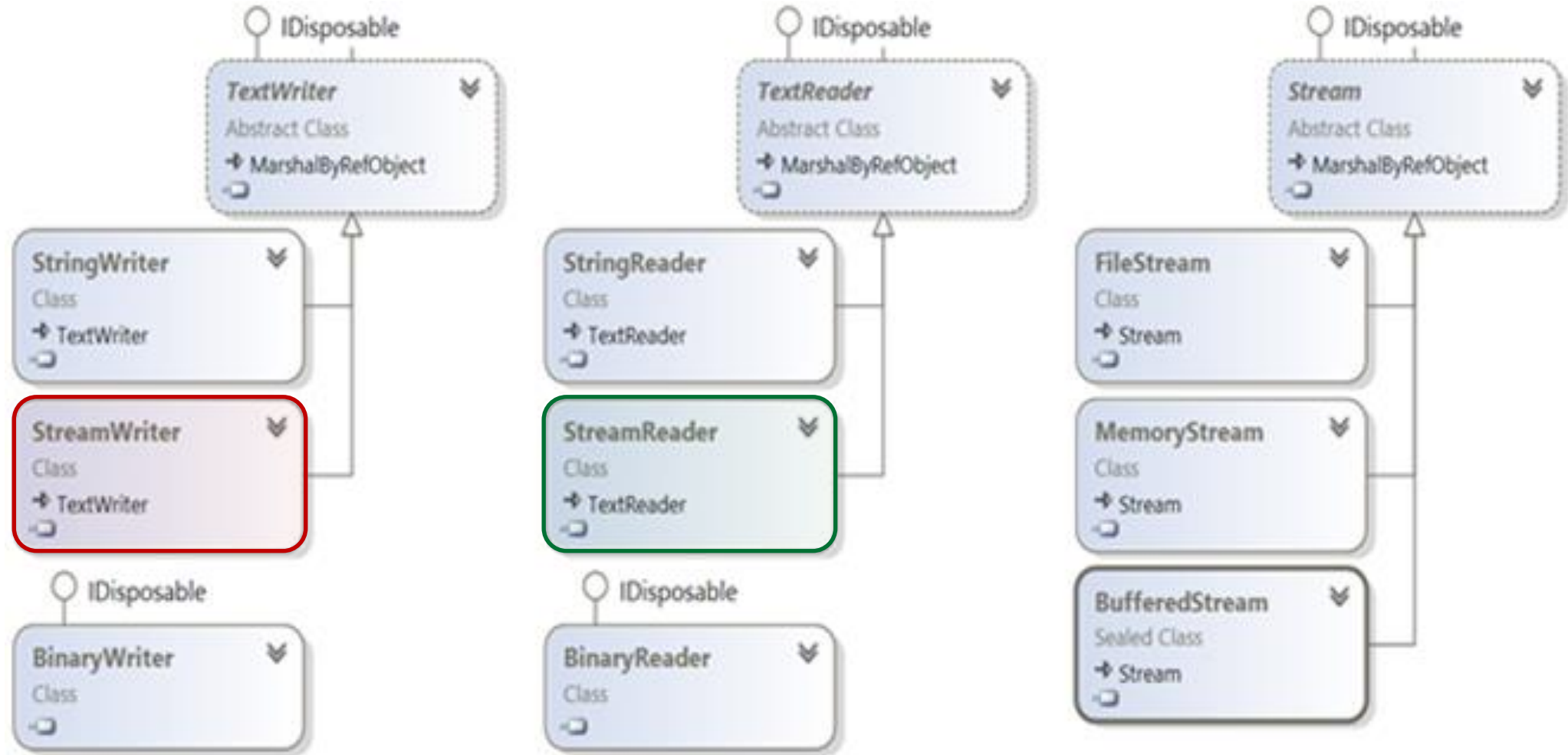
Andra byte-baserade I/O typer

- **BufferedStream** innehåller en intern **buffert** och används för att skapa “buffrade strömmar”, vilket kan snabba upp skriv- och läsoperationer.
- **BufferedStream** “omsluter” en godtycklig **innesluten ström** och tillhandahåller **bufferten**.



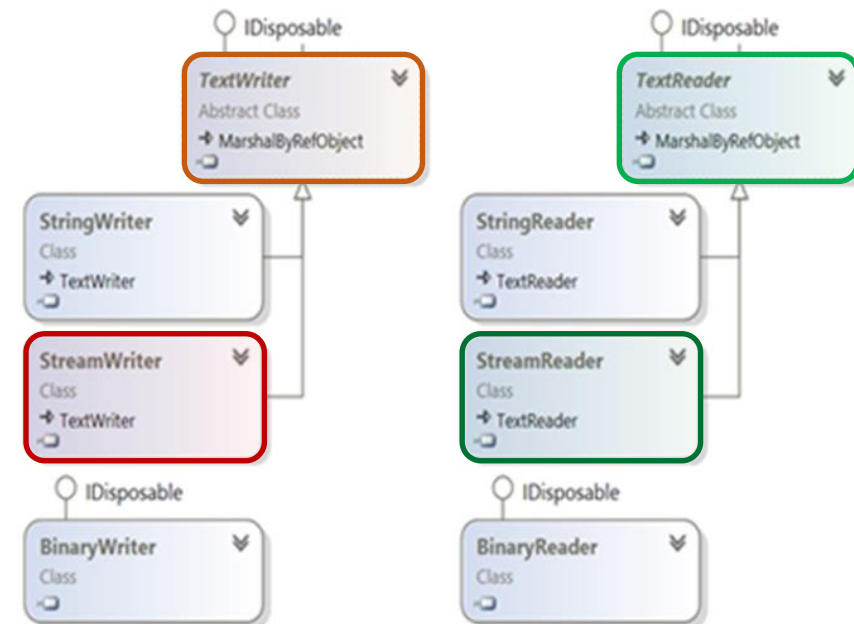
- Istället för att skriva bytes direkt till den **inneslutna strömmen**, skrivs bytesen till **bufferten**.
- När **bufferten** är full, töms dess innehåll genom att skicka bytesen till den **inneslutna strömmen**.
- Metoden **Flush()** kan användas för att “spola ut” (tömma) **buffertens innehåll** till den **inneslutna strömmen**.
- Man får inte glömma att anropa **Flush()** en sista gång innan den **inneslutna strömmen** stängs, annars kan det finnas bytes kvar i **bufferten** som inte har skickats till den **inneslutna strömmen**.
- Om property **AutoFlush** sätts till **true**, anropas **Flush()** efter varje skriv- eller läsoperation.

StreamWriter och StreamReader



StreamWriter och StreamReader

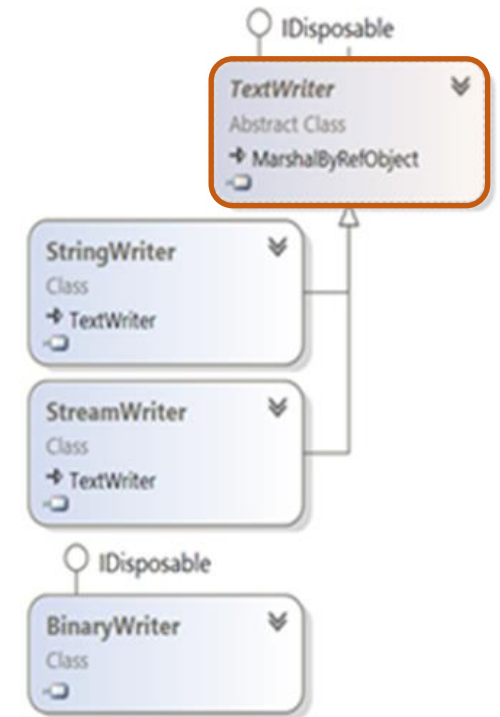
- **StreamWriter** och **StreamReader** är **text-baserade (character) strömmar** och kan användas för att skriva/läsa till/från textfiler.
- Båda strömmarna **använder Unicode som default**, men detta kan ändras genom att skicka in ett lämpligt **System.Text.Encoding** objekt via konstruktorn.
- **StreamWriter** ärver från den abstrakta klassen **TextWriter**, som innehåller funktionalitet för att **skriva textuell data** till en **text-baserad ström**.
- **StreamReader** ärver från den abstrakta klassen **TextReader**, som innehåller funktionalitet för att **läsa och titta i (peek)** en **text-baserad ström**.



TextWriter

- Den abstrakta klassen **TextWriter** innehåller bl.a. metoderna **Flush()**, **Write()** och **WriteLine()**.
- **Flush()** tömmer alla tecken i skrivbufferten till sänkan (vilket görs automatiskt vid varje skrivning om property **AutoFlush** sätts till **true**).
- **Write()** och **WriteLine()** skriver text (**utan** respektive **med** ett nyradstecken). Den statiska klassen **System.Console** innehåller en **TextWriter** (i propertyn **out**).

Member	Meaning in Life
Close()	This method closes the writer and frees any associated resources. In the process, the buffer is automatically flushed (again, this member is functionally equivalent to calling the Dispose() method).
Flush()	This method clears all buffers for the current writer and causes any buffered data to be written to the underlying device; however, it does not close the writer.
NewLine	This property indicates the newline constant for the derived writer class. The default line terminator for the Windows OS is a carriage return, followed by a line feed (\r\n).
Write() WriteAsync()	This overloaded method writes data to the text stream without a newline constant.
WriteLine() WriteLineAsync()	This overloaded method writes data to the text stream with a newline constant.

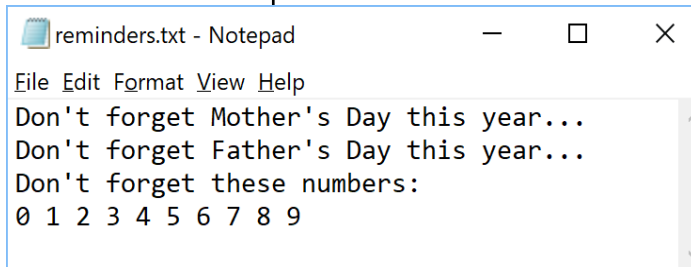


Skriva till en fil med StreamWriter

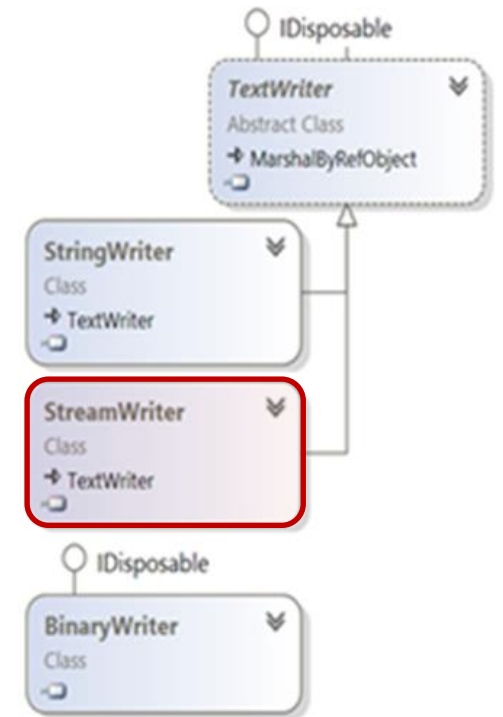
- **File.CreateText()** returnerar en **StreamWriter** som kan användas för att skriva text till en textfil via metoderna **Write()** och **WriteLine()**.
- Metoderna **Write()** och **WriteLine()** fungerar som *printf* funktionen i C, som kan ta en **formatsträng** med **placeholders {x}**, där x anger positionen i den **efterföljande variabellistan**.

```
// Get a StreamWriter and write string data.
using(StreamWriter writer = File.CreateText("reminders.txt"))
{
    writer.WriteLine("Don't forget Mother's Day this year...");
    writer.WriteLine("Don't forget Father's Day this year...");
    writer.WriteLine("Don't forget these numbers:");
    for(int i = 0; i < 10; i++)
        writer.Write(i + " ");

    // Insert a new line.
    writer.Write(writer.NewLine);
}
Console.WriteLine("Created file and wrote some thoughts...");
```



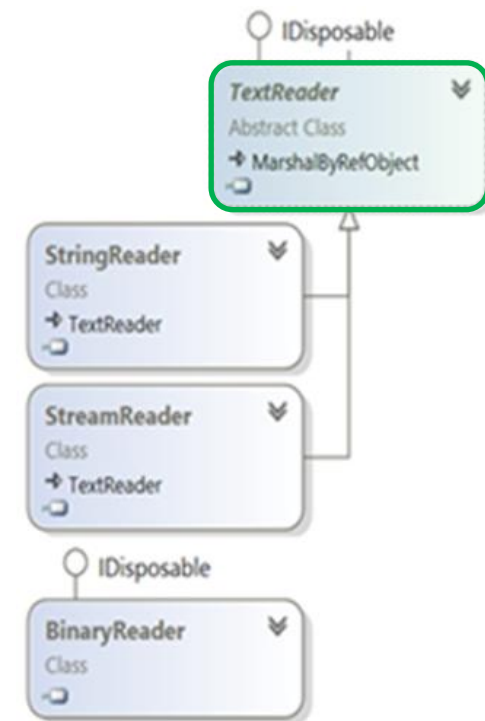
```
reminders.txt - Notepad
File Edit Format View Help
Don't forget Mother's Day this year...
Don't forget Father's Day this year...
Don't forget these numbers:
0 1 2 3 4 5 6 7 8 9
```



TextReader

- Den abstrakta klassen **TextReader** innehåller bl.a. metoderna **Read()**, **ReadLine()** och **ReadToEnd()**.
- **ReadToEnd()** läser in all text från källan som returneras som en sträng.
- **Read()** och **ReadLine()** läser **ett tecken** respektive **en rad** från källan. Den statiska klassen **System.Console** innehåller en **TextReader** (i propertyn **in**).
- Om metoderna returnerar **null** har slutet av källan (strömmen) nåtts.

Member	Meaning in Life
Peek()	Returns the next available character (expressed as an integer) without actually changing the position of the reader. A value of -1 indicates you are at the end of the stream.
Read() ReadAsync()	Reads data from an input stream.
ReadBlock() ReadBlockAsync()	Reads a specified maximum number of characters from the current stream and writes the data to a buffer, beginning at a specified index.
ReadLine() ReadLineAsync()	Reads a line of characters from the current stream and returns the data as a string (a null string indicates EOF).
ReadToEnd() ReadToEndAsync()	Reads all characters from the current position to the end of the stream and returns them as a single string.

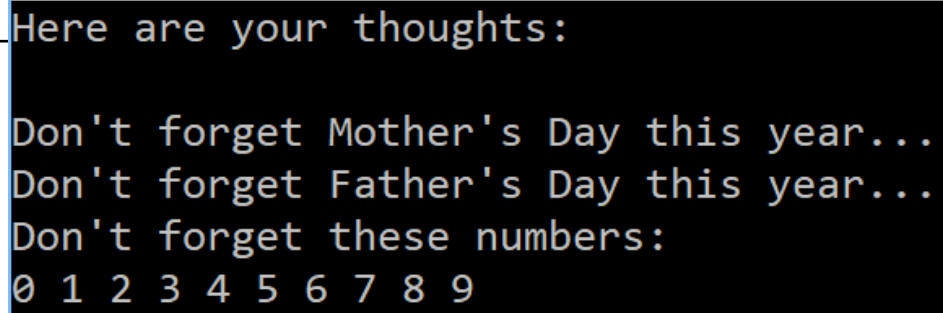


Läsa från en fil med StreamReader

- **File.OpenText()** returnerar en **StreamReader** som kan användas för att läsa text från en textfil via metoderna **Read()** och **ReadLine()**.

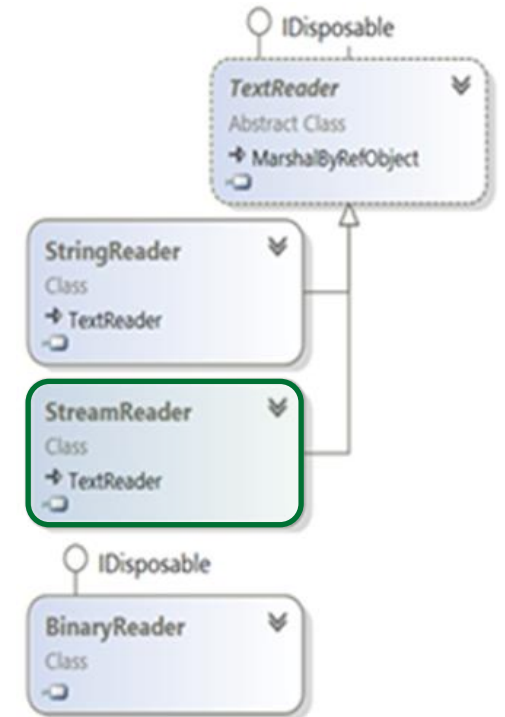
```
// Now read data from file.
Console.WriteLine("Here are your thoughts:\n");

using(StreamReader sr = File.OpenText("reminders.txt"))
{
    string input = null;
    while ((input = sr.ReadLine()) != null)
    {
        Console.WriteLine(input);
    }
}
```



```
Here are your thoughts:

Don't forget Mother's Day this year...
Don't forget Father's Day this year...
Don't forget these numbers:
0 1 2 3 4 5 6 7 8 9
```

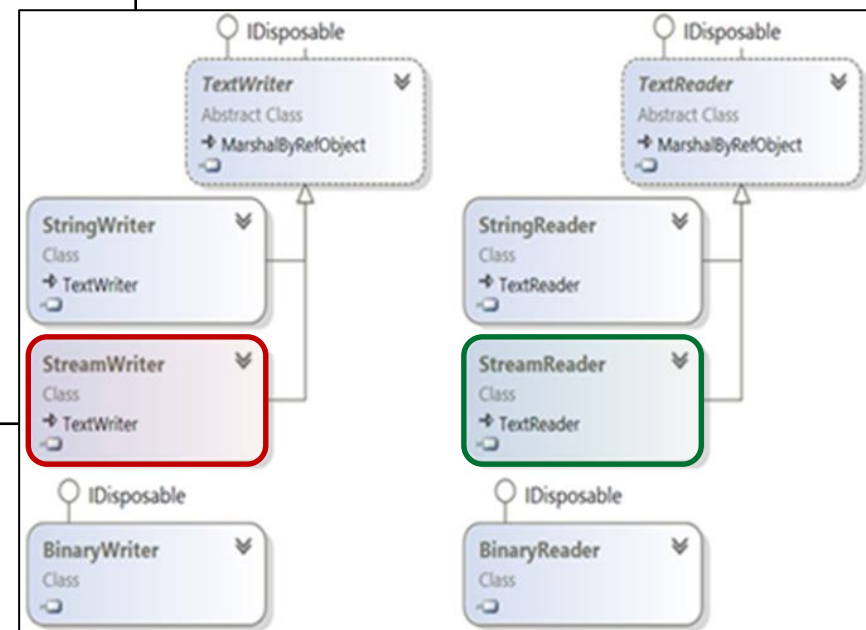


StreamWriter och StreamReader

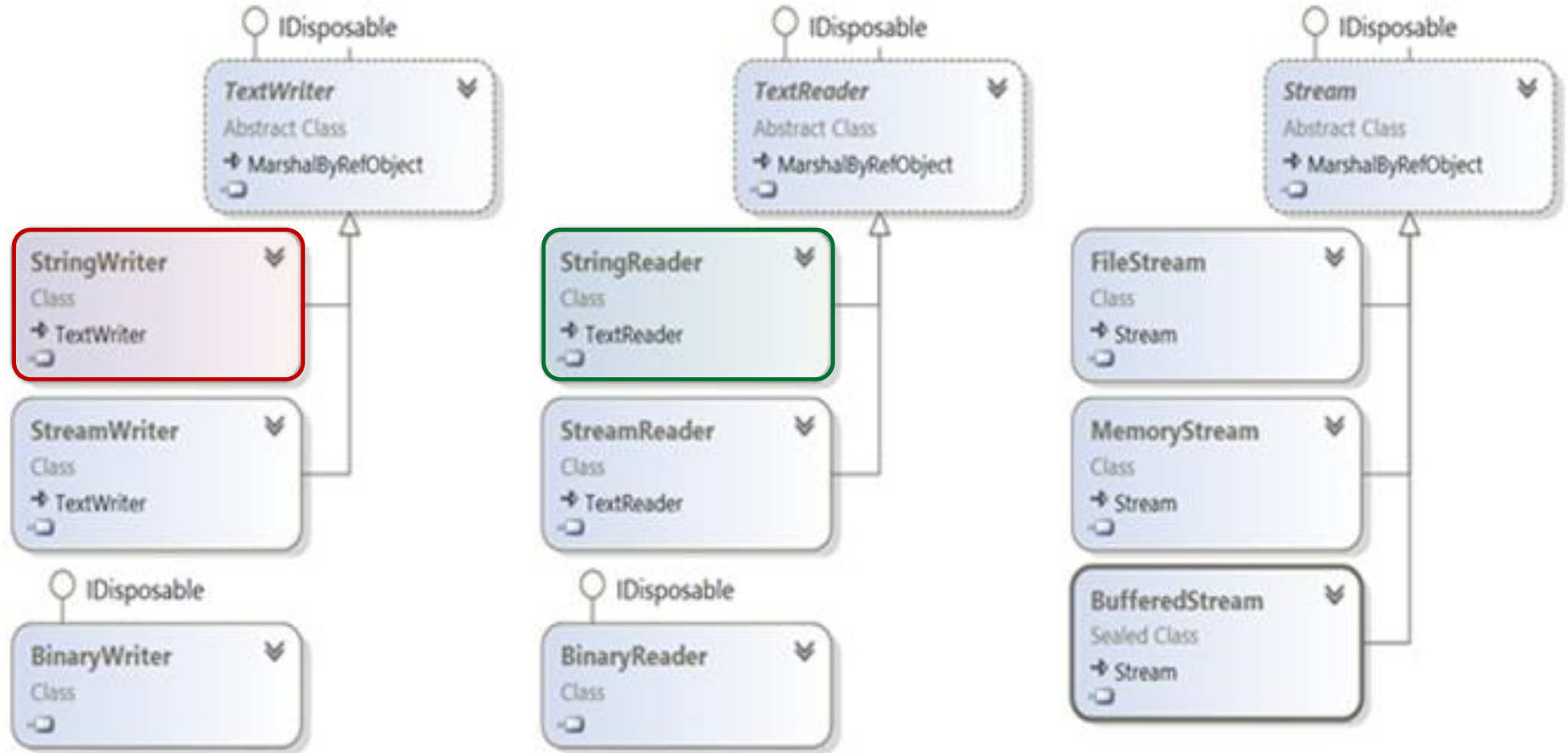
- Man kan även skapa **StreamWriter** och **StreamReader** instanser utan att använda **File.CreateText()** respektive **File.OpenText()** genom att ange sökvägen till filen direkt i konstruktorn.

```
// Get a StreamWriter and write string data.
using(StreamWriter writer = new StreamWriter("reminders.txt"))
{
    ...
}

// Now read data from file.
using(StreamReader sr = new StreamReader("reminders.txt"))
{
    ...
}
```



StringWriter och StringReader



StringWriter och StringReader

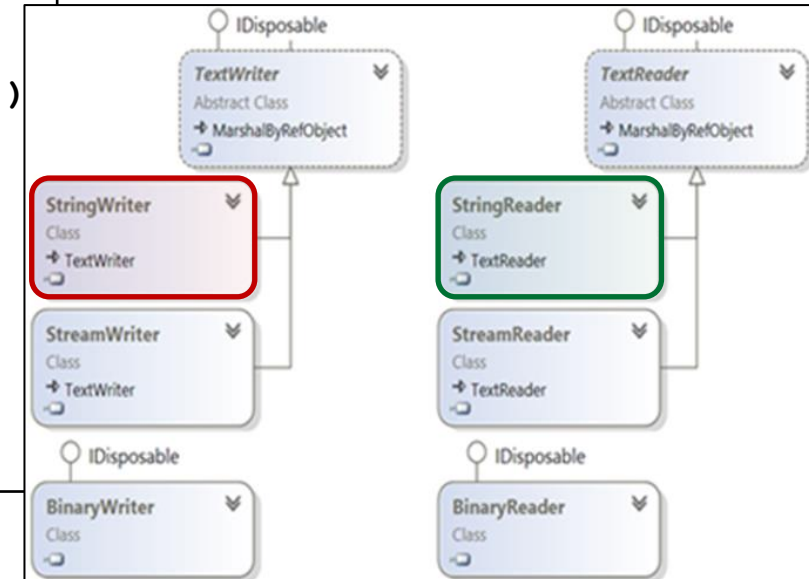
- **StringWriter** och **StringReader** klasserna kan användas för att **skriva/läsa** strängar till/från minnet istället för till/från en textfil på hårddisken.

```
// Create a StringWriter and emit character data to memory.
using (StringWriter strWriter = new StringWriter())
{
    strWriter.WriteLine("Don't forget Mother's Day this year...");
    // Get a copy of the contents (stored in a string) and dump to console.
    Console.WriteLine("Contents of StringWriter:\n{0}", strWriter);

    // Read data from the StringWriters string with a StringReader.
    using (StringReader strReader = new StringReader(strWriter.ToString()))
    {
        string input = null;
        while ((input = strReader.ReadLine()) != null)
        {
            Console.WriteLine(input);
        }
    }
}
```



Contents of StringWriter:
Don't forget Mother's Day this year...
Don't forget Mother's Day this year...

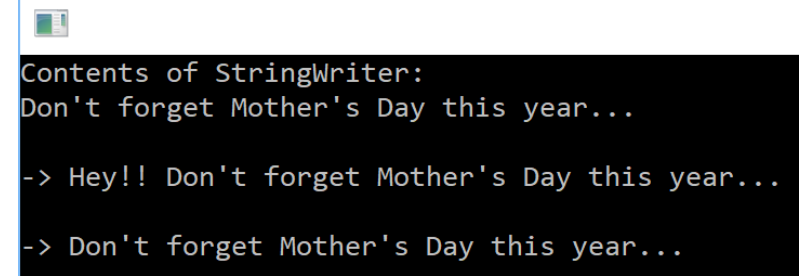


StringBuilder

- Instansmetoden **GetStringBuilder()** i **StringWriter** returnerar en **StringBuilder** instans.
- **StringBuilder** klassen är ingen ström, men är användbar när man vill bygga strängar.
- Typen **System.String** i .NET är **omodifierbar** (*immutable*), dvs varje gång man ändrar i en **System.String** returneras en ny sträng-instans, till skillnad mot en **StringBuilder** som jobbar med en **intern tecken buffert** (dvs inte skapar nya strängar). För att hämta den färdigbyggda strängen, används metoden **ToString()**.

```
using (StringWriter strWriter = new StringWriter())
{
    strWriter.WriteLine("Don't forget Mother's Day this year...");
    Console.WriteLine("Contents of StringWriter:\n{0}", strWriter);

    // Get the internal StringBuilder.
    StringBuilder sb = strWriter.GetStringBuilder(); ←
    sb.Insert(0, "Hey!! ");
    Console.WriteLine("-> {0}", sb.ToString());
    sb.Remove(0, "Hey!! ".Length);
    Console.WriteLine("-> {0}", sb.ToString());
}
```



```
Contents of StringWriter:
Don't forget Mother's Day this year...

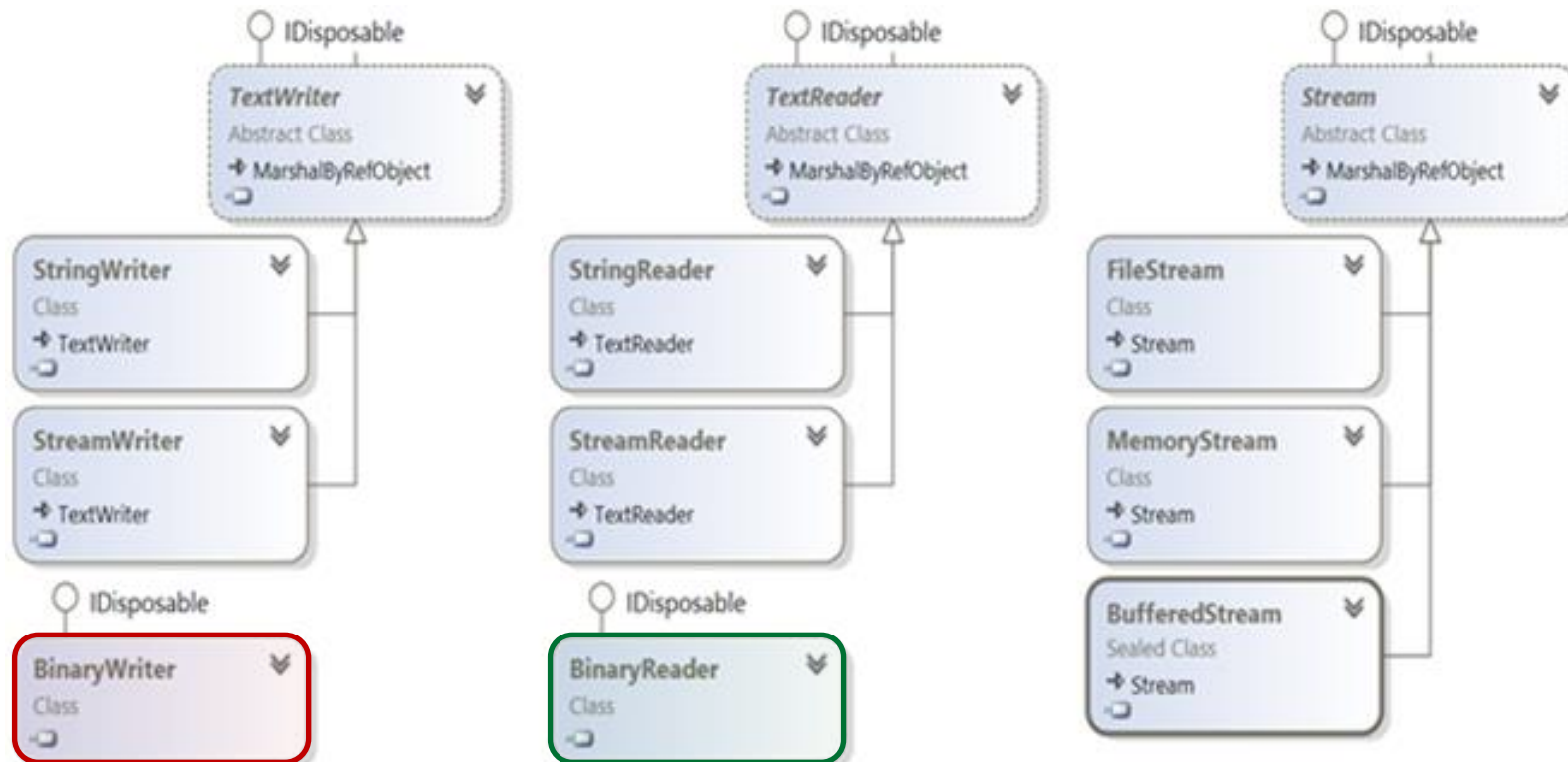
-> Hey!! Don't forget Mother's Day this year...

-> Don't forget Mother's Day this year...
```

StringBuilder klassen finns
i namespace **System.Text**

BinaryWriter och BinaryReader

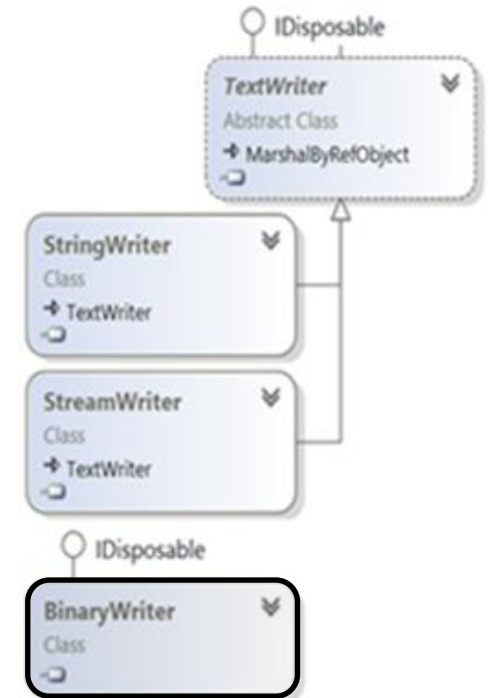
- **BinaryWriter** och **BinaryReader** ärver direkt från **System.Object**.
- Trots orden **Writer** och **Reader** i namnen kan de skriva olika **datatyper** till en underliggande ström i ett kompakt **binärformat**.



BinaryWriter

- **BinaryWriter** innehåller en **Write()** metod som har överlagrats för olika **datatyper**, fungerar ihop med andra strömmar (t.ex. **FileStream**, **MemoryStream**, **BufferedStream**) som skickas in via konstruktorn (som kan accessas via propertyn **BaseStream**), samt erbjuder **slumpvis access** (*random access*) till datan i strömmen.

Member	Meaning in Life
BaseStream	This read-only property provides access to the underlying stream used with the BinaryWriter object.
Close()	This method closes the binary stream.
Flush()	This method flushes the binary stream.
Seek()	This method sets the position in the current stream.
Write()	This method writes a value to the current stream.



BinaryReader

- **BinaryReader** innehåller ett antal **ReadXXXX()** metoder som matchar de överlagrade **Write()** metoderna i **BinaryWriter** (där **XXXX** är namnet på **datatypen**), fungerar ihop med andra strömmar som skickas in via konstruktorn (som kan accessas via propertyn **BaseStream**), samt erbjuder **slumpvis access (random access)** till datan.

Member	Meaning in Life
BaseStream	This read-only property provides access to the underlying stream used with the BinaryReader object.
Close()	This method closes the binary reader.
PeekChar()	This method returns the next available character without advancing the position in the stream.
Read()	This method reads a given set of bytes or characters and stores them in the incoming array.
ReadXXXX()	The BinaryReader class defines numerous read methods that grab the next type from the stream (e.g., ReadBoolean(), ReadByte(), and ReadInt32()).

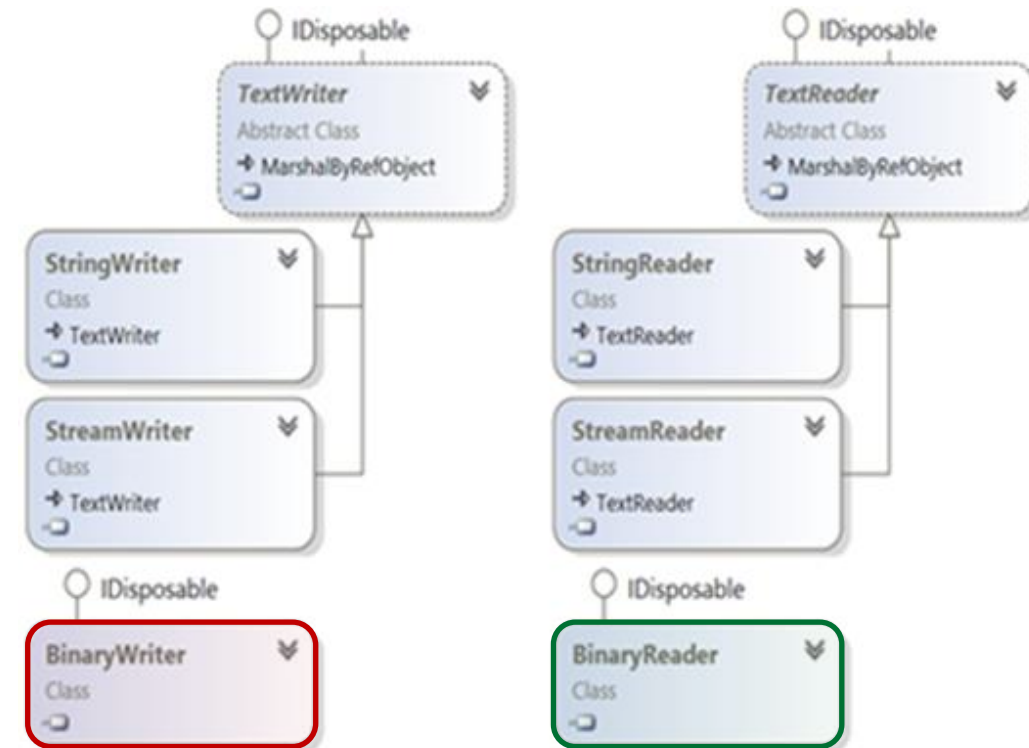


Pipeline av I/O strömmar

- **BinaryWriter**, **BinaryReader** och andra strömmar har propertyn **BaseStream**.
- Propertyen **BaseStream** innehåller en **underliggande (innesluten) ström**.

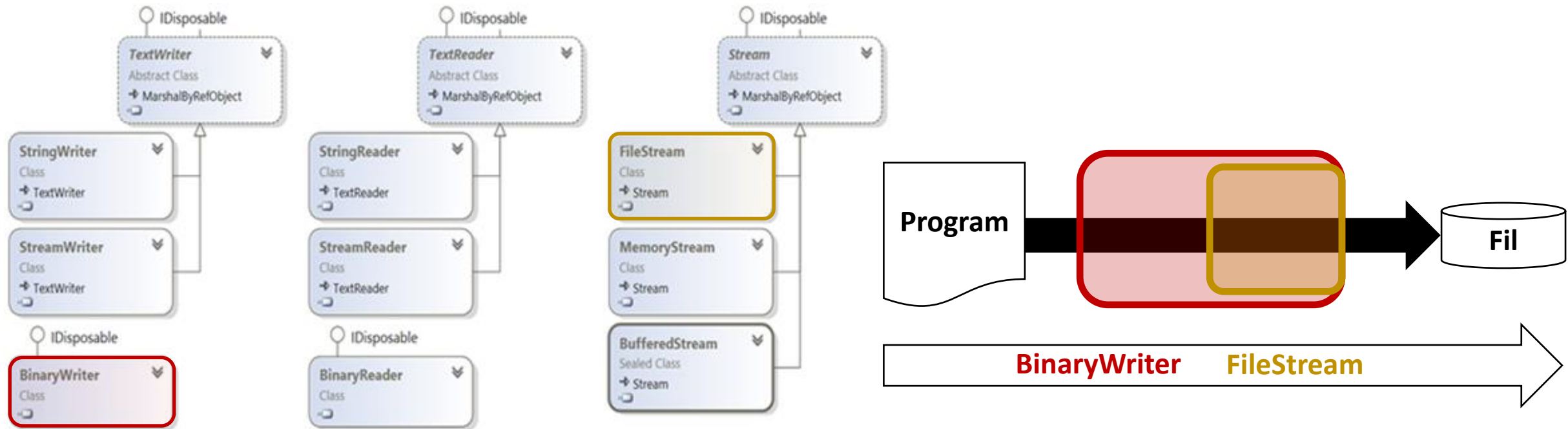
Member	Meaning in Life
BaseStream	This read-only property provides access to the underlying stream used with the BinaryWriter object.
Close()	This method closes the binary stream.
Flush()	This method flushes the binary stream.
Seek()	This method sets the position in the current stream.
Write()	This method writes a value to the current stream.

Member	Meaning in Life
BaseStream	This read-only property provides access to the underlying stream used with the BinaryReader object.
Close()	This method closes the binary reader.
PeekChar()	This method returns the next available character without advancing the position in the stream.
Read()	This method reads a given set of bytes or characters and stores them in the incoming array.
ReadXXXX()	The BinaryReader class defines numerous read methods that grab the next type from the stream (e.g., ReadBoolean(), ReadByte(), and ReadInt32()).



Pipeline av I/O strömmar

- En **pipeline** av strömmar kan skapas genom att innesluta strömmar i varandra, så att datan flödar genom en kedja av strömmar.
- Exempelvis kan en **FileStream** inneslutas i en **BinaryWriter** så att primitiva datatyper kan skrivas till en fil.



BinaryWriter, BinaryReader och FileStream

```
// Open a binary writer for a file.  
FileInfo f = new FileInfo("BinFile.dat");  
using(BinaryWriter bw = new BinaryWriter(f.OpenWrite()))  
{
```

```
    // Print out the type of BaseStream.  
    // (System.IO.FileStream in this case).  
    Console.WriteLine("Base stream is: {0}", bw.BaseStream);
```

```
    // Create some data to save in the file.  
    double aDouble = 1234.67;  
    int anInt = 34567;  
    string aString = "A, B, C";
```

```
    // Write the data.  
    bw.Write(aDouble);  
    bw.Write(anInt);  
    bw.Write(aString);
```

```
}  
Console.WriteLine("Done!");
```

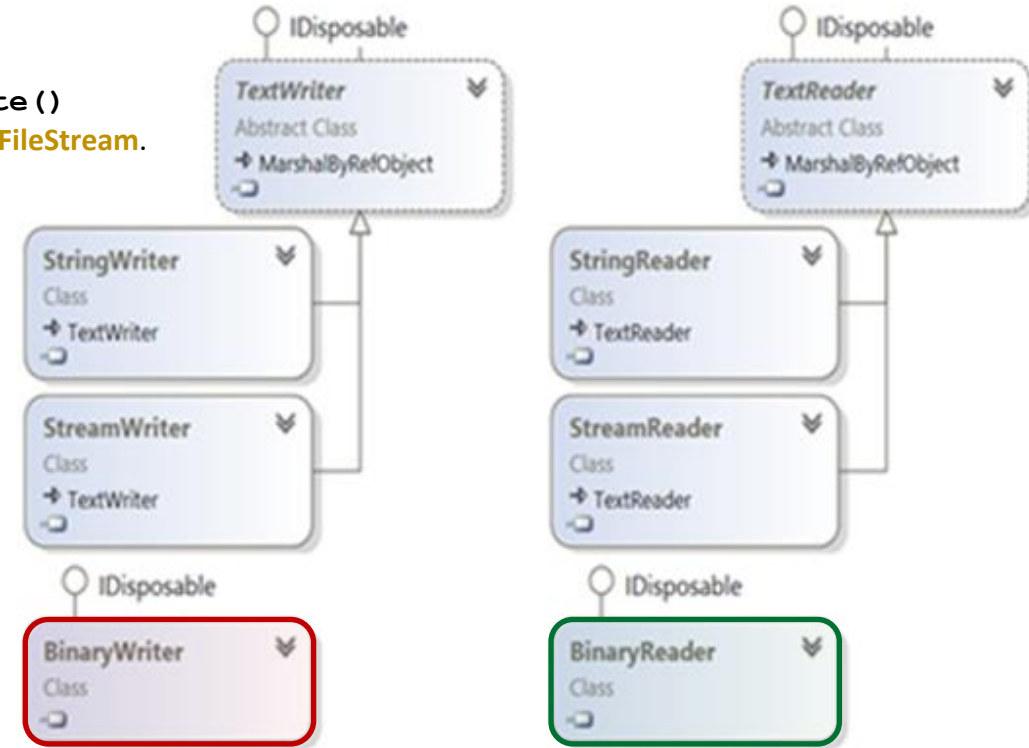
```
// Open a binary reader for a file.  
FileInfo f = new FileInfo("BinFile.dat");  
using(BinaryReader br = new BinaryReader(f.OpenRead()))  
{
```

```
    // Read the binary data from the stream.  
    Console.WriteLine(br.ReadDouble());  
    Console.WriteLine(br.ReadInt32());  
    Console.WriteLine(br.ReadString());
```

```
}
```

f.OpenWrite()
returnerar en **FileStream**.

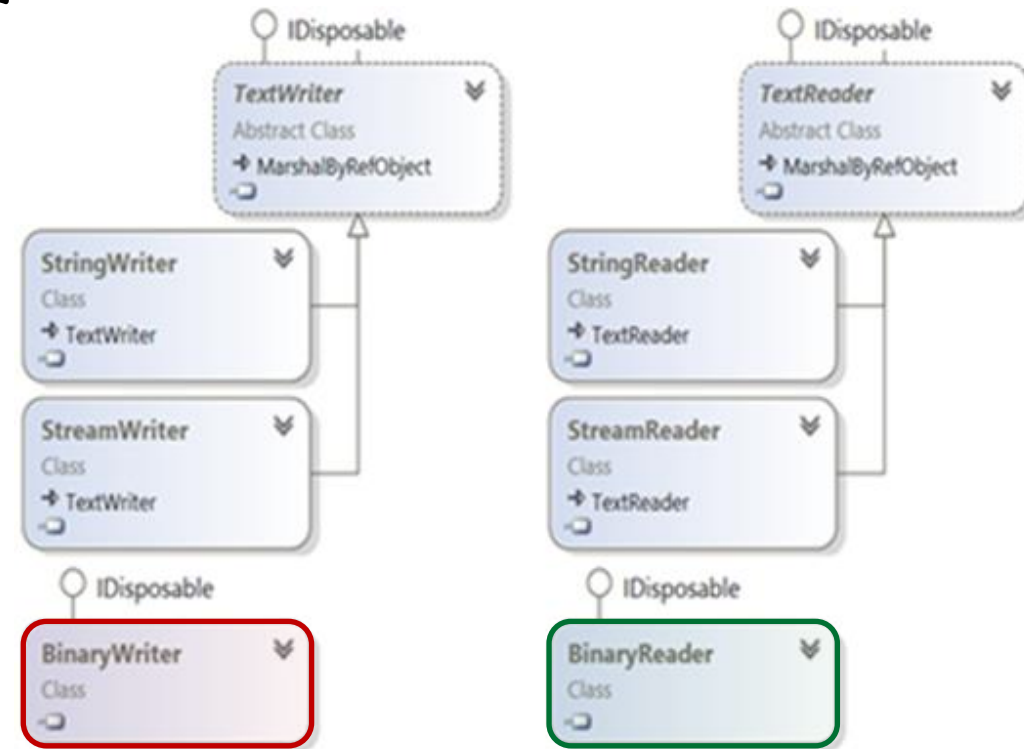
```
Base stream is: System.IO.FileStream  
Done!  
1234.67  
34567  
A, B, C
```



Anropen till **BinaryReaders ReadXXXX()** metoder matchar anropen till **BinaryWriters Write()** metoder, t.ex. för att skriva en **double aDouble** används **BinaryWriters Write(aDouble)** och för att läsa en **double** används **BinaryReaders ReadDouble()**.

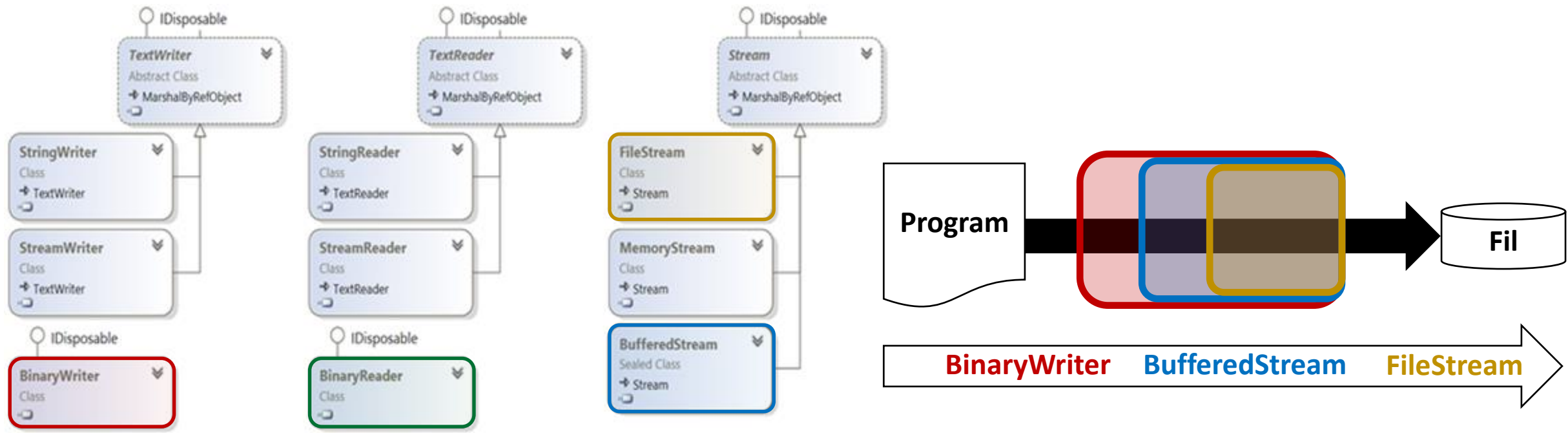
BinaryWriter och BinaryReader

- Data måste läsas i samma ordning och i samma format som när datan skrevs, t.ex.
 - Om **Write(3)** efterföljt av **Write("2")** skrivs till en fil, måste datan läsas in med **ReadInt()** efterföljt av **ReadString()**.
- Om man försöker läsa från en inputström när ingen mer data finns tillgänglig, kastas undantaget **EndOfStreamException**.
 - Strömmens **EndOfStream** property kan användas för att kolla ifall mer data finns tillgänglig innan en läsning görs.



Pipeline av I/O strömmar

- Ytterligare ett exempel där en **FileStream** inneslutas i en **BufferedStream** för att förbättra prestandan genom att buffra bytes tillfälligt för att sedan “spola ut dem” (**Flush**) till en fil i en operation via den inneslutna **FileStream** instansen.
- Därefter innesluts samma **BufferedStream** instans, i sin tur, i en **BinaryWriter** så att primitiva datatyper kan skrivas till filen.



BinaryWriter, BufferedStream och FileStream

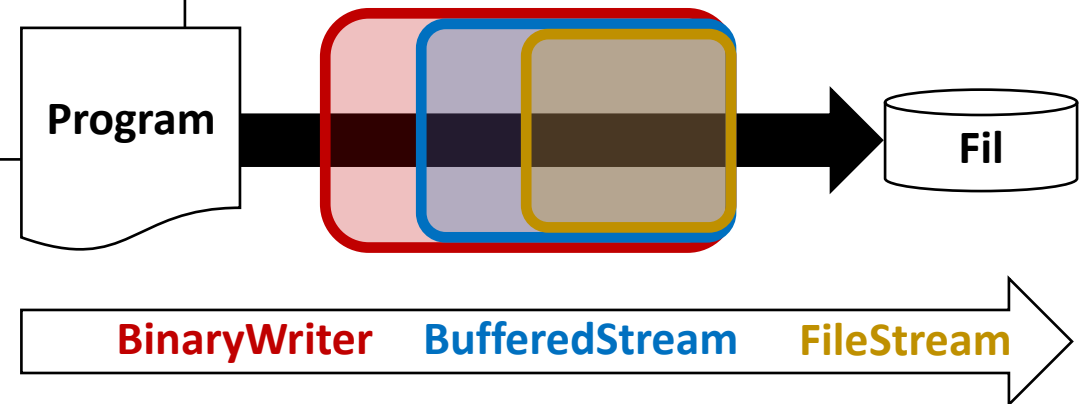
```
// Open a binary writer for a file.
FileInfo f = new FileInfo("BinFile.dat");
using(BinaryWriter bw = new BinaryWriter(new BufferedStream(f.OpenWrite())))
{
    // Print out the type of BaseStream.
    // (System.IO.FileStream in this case).
    Console.WriteLine("Base stream is: {0}", bw.BaseStream);

    // Create some data to save in the file.
    double aDouble = 1234.67;
    int anInt = 34567;
    string aString = "A, B, C";

    // Write the data.
    bw.Write(aDouble);
    bw.Write(anInt);
    bw.Write(aString);
    bw.Flush(); <-----
}
Console.WriteLine("Done!");
```

f.OpenWrite()
returnerar en FileStream.

Måste anropa Flush() på BinaryWriter instansen för att tömma
BufferedStream instansens buffert till filen via FileStream instansen.



Tecken och strängar i binär I/O

- **Unicode** består av 2 bytes, dvs 16 bitar (**Extended Unicode** består av 3 bytes, 24 bitar).
- Metoden **Write(char c)** skriver **Unicode**n för tecknet **c** till output strömmen.
- Metoden **Write(String s)** skriver **Unicode**n för varje tecken i strängen **s** till output strömmen.
- **ASCII** är en delmängd (de lägsta 8 bitarna) av **Unicode**.
- Eftersom de flesta programmen endast behöver använda **ASCII**, är det ineffektivt att representera ett 8-bitars **ASCII** tecken med ett 16-bitars **Unicode** tecken.
- **UTF-8** är en kod som tillåter system att arbeta effektivt med både **ASCII** och **Unicode** (de flesta operativsystemen använder **ASCII**, C# använder **Unicode**).
- **UTF-8** lagrar ett tecken som 1, 2 eller 3 byte(s);
 - **ASCII** värden (mindre än 0x7F) lagras som 1 byte.
 - **Unicode** värden (mindre än 0x7FF) lagras som 2 bytes.
 - **Extended Unicode** värden (större eller lika med 0x7FF) lagras som 3 bytes.

Serialisering och deserialisering

- **Serialisering (*serialization*)** beskriver processen att **konvertera ett objekt till en sekvens av bytes** (inklusive arrayer och samlingsklasser) som kan skickas till en ström.
- **Deserialisering (*deserialization*)** beskriver processen att **återskapa ett objekt från en sekvens av bytes** (inklusive arrayer och samlingsklasser) som kan tas emot från en ström.
- **All nödvändig information för att återskapa ett objekt serialiseras**, inklusive superklasser och inneslutna objekt.
- **Serialisering leder oftast till mindre kod** jämfört med användning av readers/writers från namespace **System.IO**.
- För att en **klass (eller struct)** skall vara ***serialiserbar*** måste klassen **märkas med attributet `[Serializable]`** (alldeles ovanför klassnamnet):

```
[Serializable]  
class MyClass { }
```


BinaryFormatter (Serialisering)

- Namespace **System.Runtime.Serialization.Formatters.Binary** innehåller klassen **BinaryFormatter** som används för att serialisera/deserialisera instanser av serialiserbara klasser (och structar) i ett kompakt binärformat.
- Instansmetoden **Serialize()** tar en godtycklig **ström** samt ett **objekt** som in-parametrar.

```
static void Main(string[] args)
{
    UserPrefs userData = new UserPrefs();
    userData.WindowColor = "Yellow";
    userData.FontSize = 50;

    // The BinaryFormatter persists state data in a binary format.
    // You would need to import System.Runtime.Serialization.Formatters.Binary
    // to gain access to BinaryFormatter.
    BinaryFormatter binFormat = new BinaryFormatter();

    // Store object in a local file.
    using(Stream fStream = new FileStream("user.dat", FileMode.Create, FileAccess.Write, FileShare.None))
    {
        binFormat.Serialize(fStream, userData);
    }
}
```

```
[Serializable]
public class UserPrefs
{
    public string WindowColor;
    public int FontSize;
}
```

BinaryFormatter (Deserialisering)

- För att återskapa ett serialiserat objekt används metoden **Deserialize()** i **BinaryFormatter**.
- Instansmetoden **Deserialize()** tar en godtycklig **ström** som in-parametrar och returnerar en **System.Object** som måste explicit typkonverteras till ett **objekt** av aktuell **typ**.

```
[Serializable]
public class UserPrefs
{
    public string WindowColor;
    public int FontSize;
}
```

```
BinaryFormatter binFormat = new BinaryFormatter();

// Read the UserPrefs from the binary file.
using(Stream fStream = new FileStream("user.dat", FileMode.Open, FileAccess.Read, FileShare.None))
{
    UserPrefs userData = (UserPrefs)binFormat.Deserialize(fStream);
    Console.WriteLine("WindowColor: {0}, FontSize: {1}", userData.WindowColor, userData.FontSize);
}
```

Attributet [NonSerialized]

- Om man inte vill att en **medlem** skall serialiseras (t.ex. om medlemmen innehåller ett slumpvärde som inte skall sparas) markeras **medlemmen** med attributet **[NonSerialized]** (alldeles ovanför namnet).
- När objektet deserialiseras får medlemmen sitt defaultvärde beroende på typ (t.ex. får en **int** värdet **0** och en innesluten **klass** värdet **null**).

```
[Serializable]
class MyClass
{
    [NonSerialized]
    int myAttribute;
}
```

Attributet [NonSerialized]

```
static void Main(string[] args)
{
    UserPrefs userData = new UserPrefs();
    userData.WindowColor = "Yellow";
    userData.FontSize = 50;

    // The BinaryFormatter persists state data in a binary format.
    // You would need to import System.Runtime.Serialization.Formatters.Binary
    // to gain access to BinaryFormatter.
    BinaryFormatter binFormat = new BinaryFormatter();

    // Store object in a local file.
    using(Stream fStream = new FileStream("user.dat", FileMode.Create, FileAccess.Write, FileShare.None))
    {
        binFormat.Serialize(fStream, userData);
    }

    // Read the UserPrefs from the binary file.
    using(Stream fStream = new FileStream("user.dat", FileMode.Open, FileAccess.Read, FileShare.None))
    {
        UserPrefs userData = (UserPrefs)binFormat.Deserialize(fStream);
        Console.WriteLine("WindowColor: {0}, FontSize: {1}", userData.WindowColor, userData.FontSize);
    }
}
```

```
[Serializable]
public class UserPrefs
{
    public string WindowColor;
    [NonSerialized]
    public int FontSize;
}
```

Attributet **FontSize** serialiseras inte här

Sammanfattning Input/Output (I/O)

- .NETs I/O klasser finns i namespace **System.IO**.
- **DriveInfo** används för att erhålla information om **diskenheter**.
- **Directory** och **File** innehåller **statiska metoder** för att **manipulera filsystemet**.
- **DirectoryInfo** och **FileInfo** ärver från **FileSystemInfo** och innehåller **instansmetoder**.

Sammanfattning Input/Output (I/O)

- En **FileStream** (byte-baserad ström) kan erhållas från:
 - `FileInfo.Create()`
 - `FileInfo.Open()`
 - `FileInfo.OpenRead()`
 - `FileInfo.OpenWrite()`
- En **StreamReader** (text-baserad ström) kan erhållas från:
 - `FileInfo.OpenText()`
- En **StreamWriter** (text-baserad ström) kan erhållas från:
 - `FileInfo.CreateText()`
 - `FileInfo.AppendText()`
- Ett **using scope** kan användas för att automatiskt stänga **IDisposable** objekt.

Sammanfattning Input/Output (I/O)

- .NETs ström-baserade I/O klasser finns i namespace **System.IO**.
- Strömmar kan kopplas till ***källor*** och ***sänkor***.
- Det finns två kategorier av strömmar; ***text-baserade*** (*character*) och ***binära***.
 - **Stream** är den abstrakta basklassen för **binära operationer**.
 - **TextReader** är den abstrakta basklassen för **text-baserade läsoperationer**.
 - **TextWriter** är den abstrakta basklassen för **text-baserade skrivoperationer**.

Sammanfattning Input/Output (I/O)

- **FileStream** kan användas för att läsa/skriva data från/till en **binärfil**.
- **MemoryStream** kan användas för att läsa/skriva binärdata från/till **RAM**.
- **BufferedStream** kan användas för att skapa **buffrade strömmar**.

- **StreamReader** kan användas för att **läsa från en textfil**.
- **StreamWriter** kan användas för att **skriva till en textfil**.

- **StringReader** kan användas för att **läsa text från en sträng** (i RAM).
- **StringWriter** kan användas för att **skriva text till en sträng** (i RAM).
- **StringBuilder** har en intern buffert och är effektivare för strängmanipulationer.

Sammanfattning Input/Output (I/O)

- **BinaryReader** kan användas för att **läsa primitiva datatyper och strängar** från en **binärström**.
- **BinaryWriter** kan användas för att **skriva primitiva datatyper och strängar** till en **binärström**.
- **Pipelines** av strömmar skapas genom att innesluta strömmar i varandra.
 - Glöm inte att tömma en **BufferedStreams** buffert via metoden **Flush()**.

Sammanfattning Input/Output (I/O)

- **BinaryFormatter** kan användas för att **skriva/läsa** (serialisera/deserialisera) **objekt** från/till en **binärström**.
- Endast instanser av klasser som markeras med **[Serializable]** attributet går att serialisera.
- Ett attribut som markeras med **[NonSerialized]** serialiseras inte av CLR.