File I/O and Serialization In C#

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Agenda

- File IO
  - Stream Architecture
  - FileStream
  - StreamReader and StreamWriter
  - Working with the fileSystem
- Serialization
- XML data files

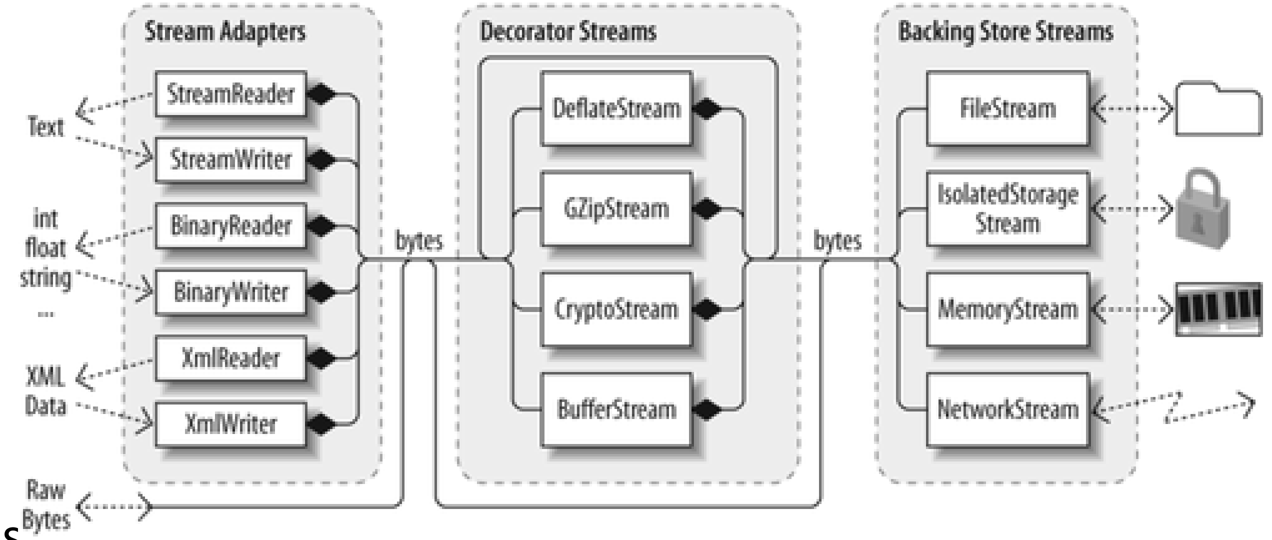
Læringsmål:

- Beskrive og anvende programmeringssproget C#.
- Anvende .Net frameworkets faciliteter til persistering af data i filer.

Files and Streams

- A *file* is a collection of data stored on a disk with a name and (often) a directory path
- A stream is something on which you can perform read and write operations
- When you open a file for reading or writing, it becomes a *stream*
- But streams are more than just open disk files:
  - Data coming over a network is a stream
  - And you can also create a stream to a buffer in memory
  - In a console application, keyboard input and text output are also streams

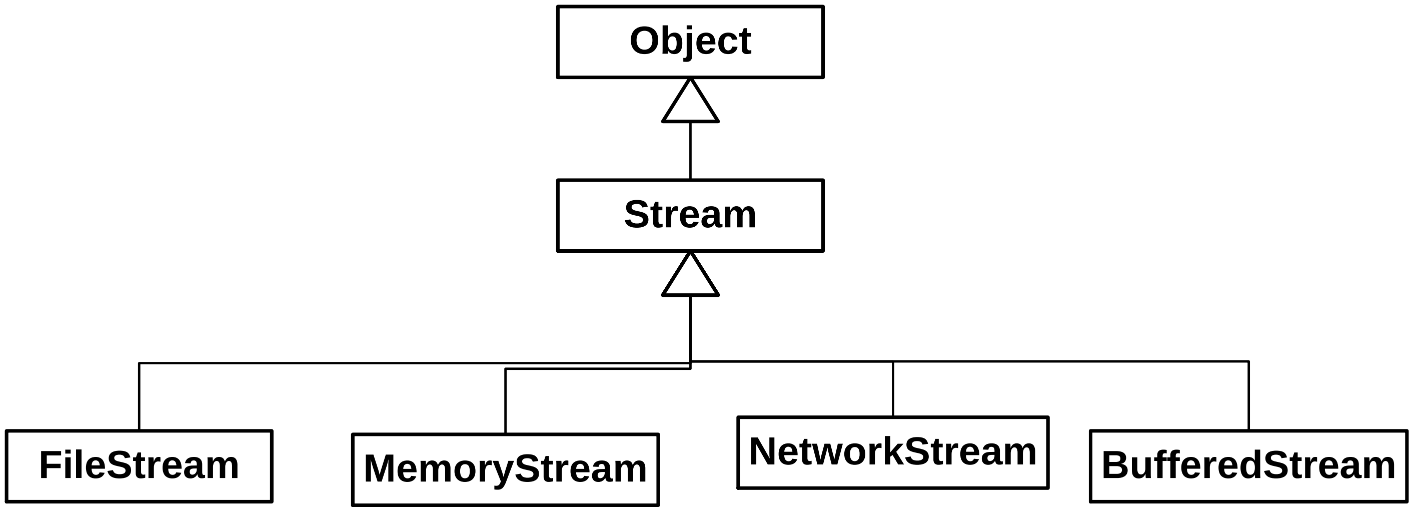
Stream Architecture



- Backing store streams
  - are the endpoint that makes input and output useful.
  - are hard-wired to a particular type of backing store, e.g. a file.
- Decorator streams
  - provide transparent binary transformations such as buffering or encryption.
  - You can chain decorators together.
- Stream Adapters
  - adapters offer typed methods for dealing in higher-level types such as strings and XML.
  - An adapter wraps a stream, just as a decorator. But an adapter is not itself a stream.

To compose a chain, you simply pass one object into another's constructor

Stream klasser



FileStream	A stream on a disk file.
MemoryStream	A stream that is stored in memory.
NetworkStream	A stream on a network connection.
BufferedStream	Implements a buffer on top of another stream.

The stream class provides raw functions to read and write at a byte level.

Kreering af en fil

På den ene måde:

```
FileInfo f = new FileInfo(@"C:\Temp\Test.txt");
FileStream fs = f.Create();
```

Og på den anden måde:

```
FileStream fs2 = new FileStream(@"C:\Temp\Test2.txt",
                                FileMode.Create);
```

Der findes et passende udvalg af overloads for begge metoder (FileMode, FileAccess, FileShare)

Åbning af en fil

På den ene måde:

```
FileInfo f2 = new FileInfo(@"C:\temp\Test.txt ");
FileStream  fs = f2.Open(FileMode.OpenOrCreate,
                         FileAccess.ReadWrite,
                         FileShare.None);
```

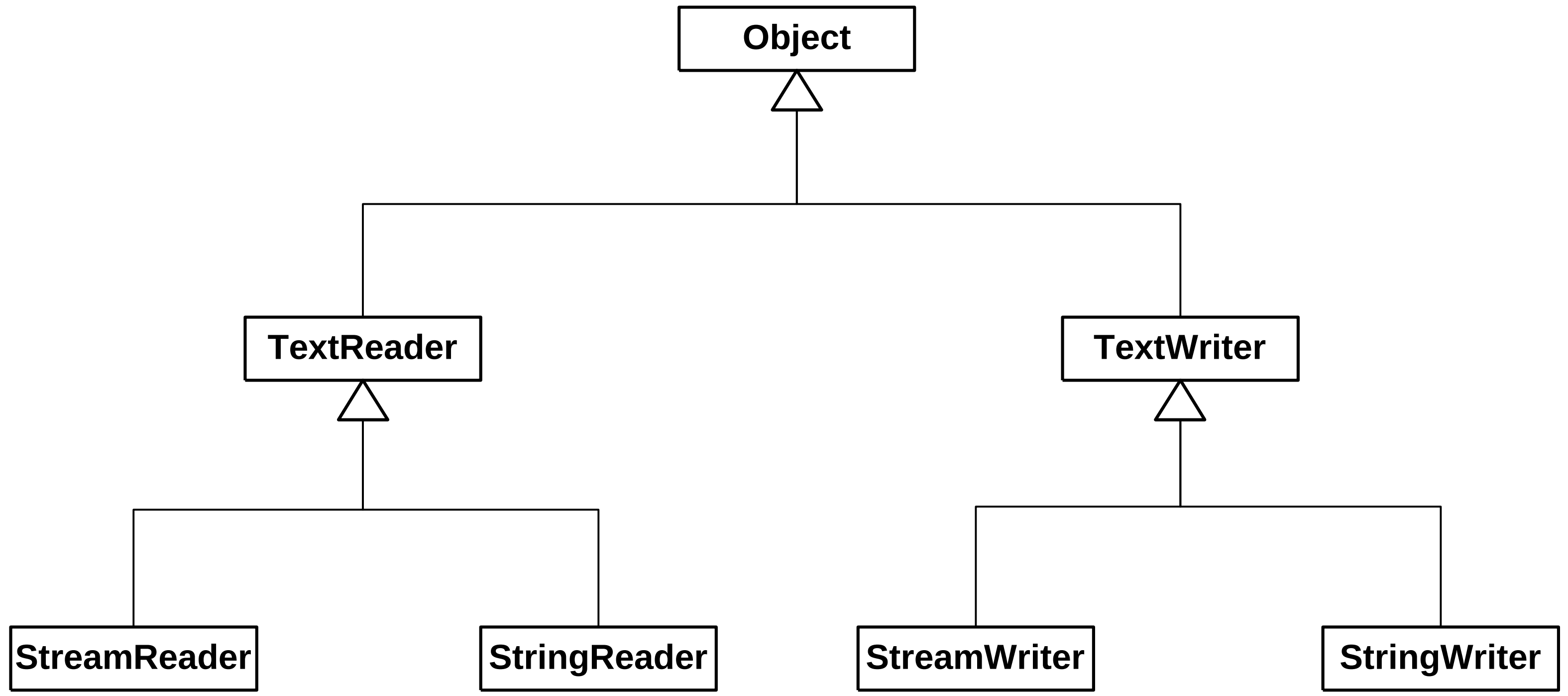
Og på den anden måde:

```
FileStream fs = new FileStream(@"C:\temp\Test.txt",
                  FileMode.OpenOrCreate,
                  FileAccess.ReadWrite,
                  FileShare.None);

// Or short version:
FileStream s = new FileStream(@"C:\Test.txt",
                  FileMode.Open);
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Readers og Writers



These classes provides a high level interface for reading and writing text files.

For binary file access at a higher level than the raw stream classes you can use the BinaryReader and the BinaryWriter classes.

Use of streamWriter And Reader

```
FileStream fs = new FileStream(@"C:\Temp\Test.txt",
                               FileMode.OpenOrCreate);
StreamWriter s = new StreamWriter(fs);
s.WriteLine("Test {0}", 55);
s.Close();
fs.Close();
```

```
FileStream fs = new FileStream("File.txt", FileMode.Open);
StreamReader s = new StreamReader(fs, Encoding.Default);
string line = "";

while ((line = s.ReadLine()) != null)
   Console.WriteLine(line);

s.Close();
fs.Close();
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

ReadAllLines

- Because reading from files is a very common operation Microsoft has created an easy shortcut:

```
string[] lines = File.ReadAllLines("file.txt");
foreach (var line in lines) {
  Console.WriteLine("Length={0}, Line={1}", line.Length, line);
}
```

But don't use it with big files as it has an inefficient use of memory!

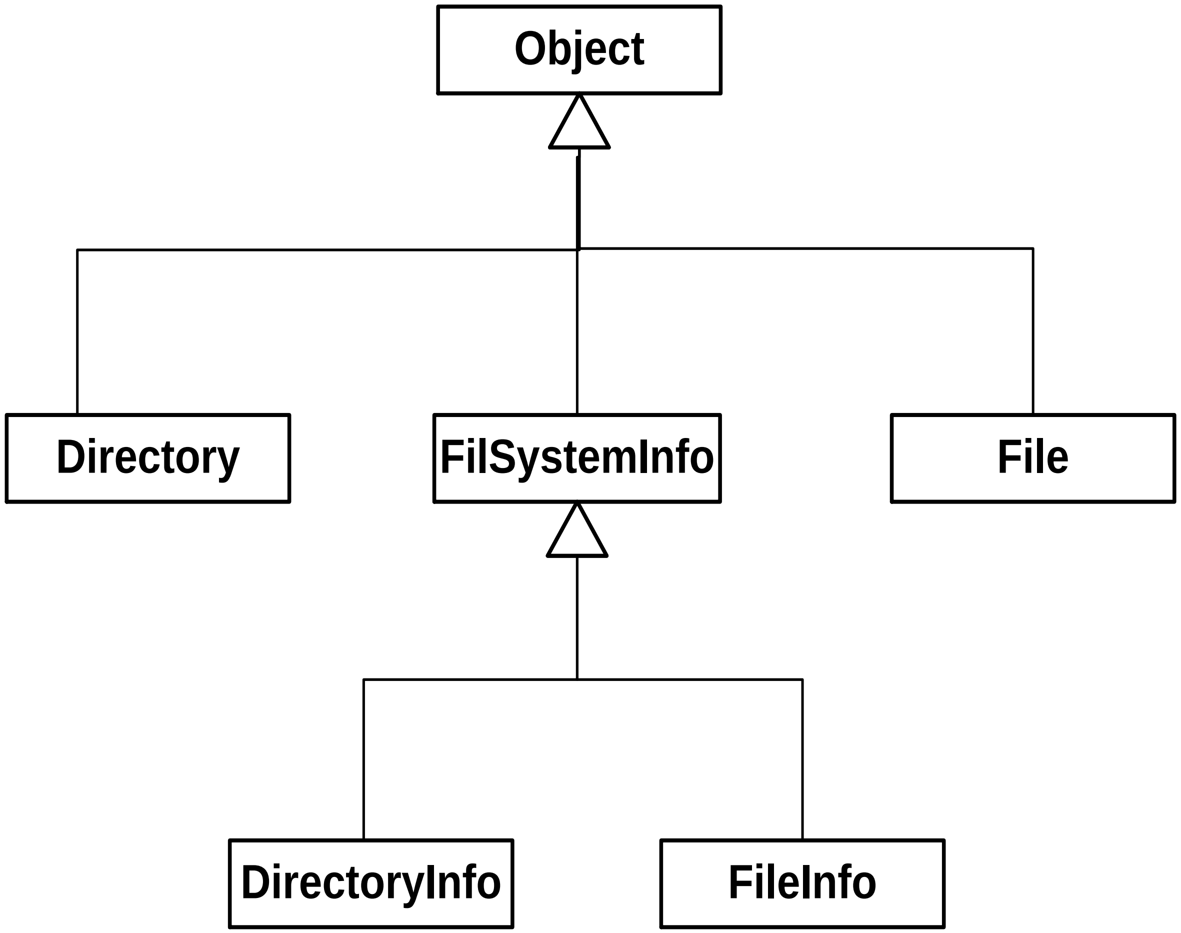
ReadLines

- This new method in .Net ver. 4.0 is much more efficient because it does not load all of the lines into memory at once; instead, it reads the lines one at a time

```
IEnumerable<string> lines = File.ReadLines("verylargefile.txt");
foreach (var line in lines) {
  Console.WriteLine("Length={0}, Line={1}", line.Length, line);
}
```

WORKING WITH THE FILESYSTEM

File og Directory klasserne



Directory og DirectoryInfo

- Disse klasser indeholder metoder, properties og opremsningstyper som kan bruges til at undersøge og ændre mappestrukturen (directories) på et disk drev.
- Directory klassen indeholder kun statiske metoder.
- DirectoryInfo klassen indeholder kun ikke statiske metoder og properties.
- Af hensyn til effektivitet er det nogle gange bedre at bruge metoderne i DirectoryInfo klassen frem for de tilsvarende i Directory klassen.

DirectoryInfo Properties

<u>De vigtigste ”Public Instance Properties”:</u>

• Attributes	Gets or sets the attributes of the current file.
• CreationTime	Gets or sets the creation time of the current file.
• Exists	Gets a value indicating whether the directory exists.
• Extension	The file name 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	Overridden. Gets the name of this DirectoryInfo instance.
• Parent	Gets the parent directory of a specified subdirectory.
• Root	Gets the root portion of a path.

DirectoryInfo Methods

De vigtigste ”Public Instance Methods”:

• **Create**	Creates a directory.
• **CreateSubdirectory**	Creates a subdirectory or subdirectories on the specified path. The specified path can be relative to this instance of the DirectoryInfo.
• **GetDirectories**	Returns the subdirectories of the current directory.
• **GetFiles**	Returns a file list from the current directory.
• **MoveTo**	Moves a DirectoryInfo and its contents to a new path.
• **Refresh**	Refreshes the state of the object.

DirectoryInfo eksempel

```
// Create a new directoryinfo object and use it.
DirectoryInfo dir = new DirectoryInfo(@"C:\Windows");




Console.WriteLine("FullName: {0}", dir.FullName);
Console.WriteLine("Name: {0}", dir.Name);
Console.WriteLine("Parent: {0}", dir.Parent);
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Directory eksempel

Indeholder statiske udgave af de fleste metoder fra DirectoryInfo.

Eks.:

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

FileInfo Methods

- **AppendText** Creates a StreamWriter that appends text to a file.
- **CopyTo** Copies an existing file to a new file.
- **Create** Creates a file.
- **CreateText** Creates a StreamWriter that writes a new text file.
- **Delete** Permanently deletes a file.
- **MoveTo** Moves a specified file to a new location, providing the option to specify a new file name.
- **Open** Opens a file with various read/write and sharing privileges.
- **OpenRead** Creates a read-only FileStream.
- **OpenText** Creates a StreamReader with UTF8 encoding that reads from an existing text file.
- **OpenWrite** Creates a read/write FileStream.
- **Refresh** Refreshes the state of the object.

FileInfo Eksempel

```
// Make a new FileInfo.
FileInfo f = new FileInfo(@"C:\Test.txt");
// Print some basic traits.
Console.WriteLine("Creation: {0}", f.CreationTime);
Console.WriteLine("Full name: {0}", f.FullName);
Console.WriteLine("Full atts: {0}", f.Attributes.ToString());
```

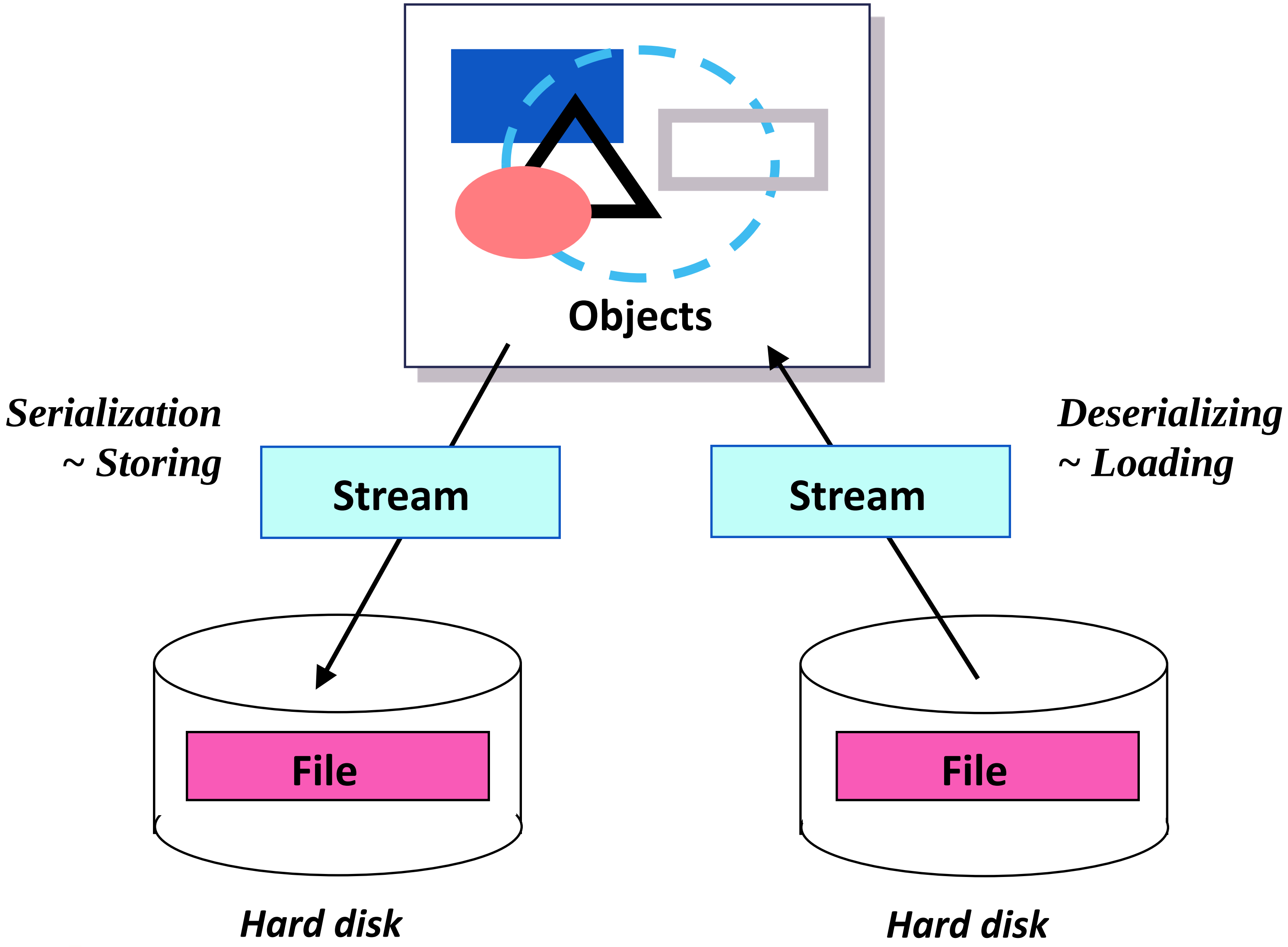
Se demo: BasicFileApp

SERIALIZATION

Serialization overview

Serialization Formatters: Binary, XML and JSON

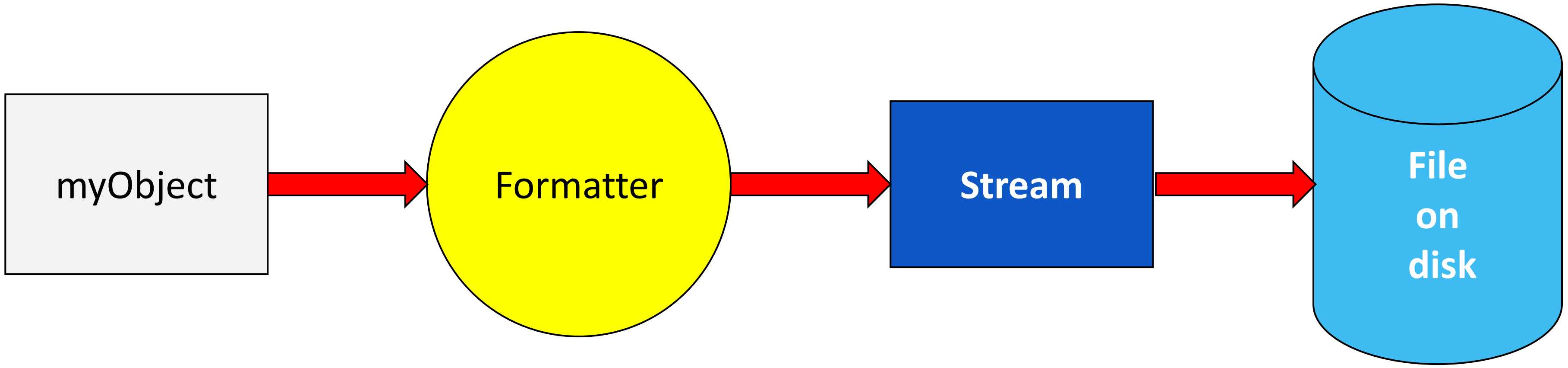
What Is Serialization?



Hvad er serialisation?

- Med “Serialization” menes den proces hvor man skriver et objekt til eller læser et objekt fra et persistent lager medie, som f.eks. en fil på harddisken.
- I C# gøres en klasse serialiserbar ved at tilføje attributten
  `[Serializable]`
  Dette er dog ikke nødvendigt ved brug af XML-serializer eller JSON-serializer.
- Den grundlæggende ide er at et objekt selv skal være i stand til at skrive dets interne tilstand (værdien af alle datamedlemmer) til en stream (et persistent lagermedie eller datakommunikationskanal).
- Senere er det så muligt at genskabe objektet ved at læse objektets tilstansdata fra en stream (et persistente lagermedie).
  - Denne proces kaldes for deserializing.
- **Hvis objektet har referencer til andre objekter, så serialiseres disse objekter også** (kræver at de også er erklæret Serializable, ved brug af den binære serializer).

The Serialization Process



```
using System.Runtime.Serialization.Formatters.Binary;
…
// Now save myObject to a binary stream.
FileStream myStream = File.Create(”MyFile.dat");
BinaryFormatter myBinaryFormat = new BinaryFormatter();
myBinaryFormat.Serialize(myStream, myObject);
myStream.Close();
```

Hvordan erklæres en klasse serializable ?

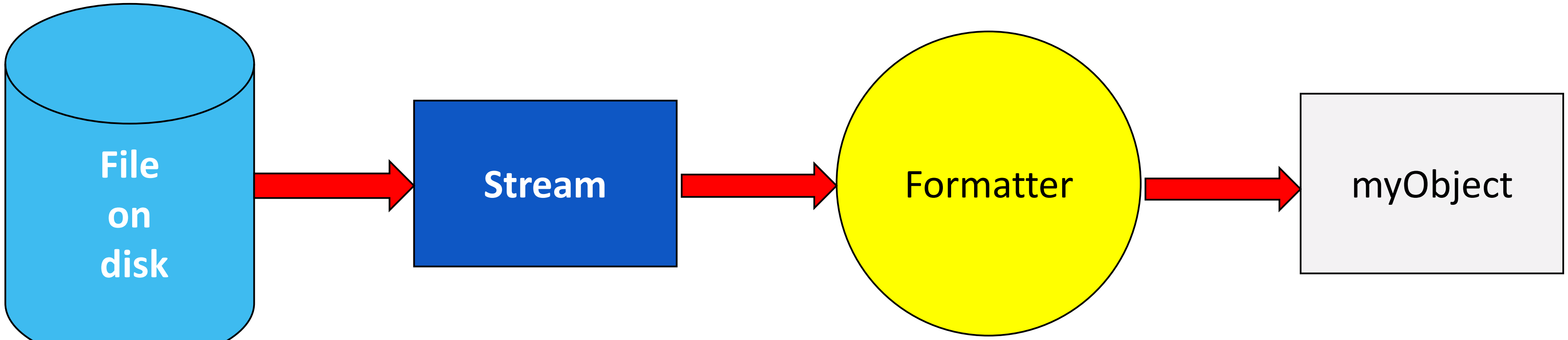
I C# gøres en klasse serialiserbar ved at tilføje attributten **[Serializable]** til klassen.

```
namespace CarToFileApp
{
// The Car class is serializable!
[Serializable]
public class Car
{
  protected string petName;
  protected Radio theRadio = new Radio();

  ...
```

Note: Put the NonSerialized attribute on fields you don’t want to serialize.

The Deserialization Process



```
using System.Runtime.Serialization.Formatters.Binary;
…
// Now read back myObject from a binary stream.
FileStream myStream = File.OpenRead(”MyFile.dat");
BinaryFormatter myBinaryFormat = new BinaryFormatter();
MyClass myObject = (MyClass)myBinaryFormat.Deserialize(myStream);
myStream.Close();
```

Custom Serialization

- If you want to customize the serialization process then you can implement the ISerializable interface in the classes you want to serialize.

SERIALIZATION FORMATTERS

BinaryFormatter

- Namespace: System.Runtime.Serialization.Formatters.Binary
- Serializes and deserializes an object, or an entire graph of connected objects, in binary format.
- PROS:
  - The output byte stream generated is **compact**
  - The serialization process is **faster** than using the other formatters.
  - This formatter **can serialize generic and non generic collections** (being the items within the collection is serializable)
  - Serializes public and private members (deep serialization)
- CONS:
  - Format **not readable by other techonolgies** (just .NET Framework)

XmlSerializer

- Namespace: System.Xml.Serialization
- Serializes and deserializes objects (just the public properties and fields) into and from XML documents.
- The XmlSerializer enables you to control how objects are encoded into XML.
- Requeriments:
  - Type must be public (public class person).
  - Must implement a parameterless constructor (in order to deserialize the object).
  - If you are serializing a non generic collection of items, you must pass the types that are stored in the collection as a parameter in the constructor of the **XmlSerializer** (see example code).

XmlSerializer

- PROS:
  - It can serialize **generic and non generic collections** (being the items within the collection is serializable)
  - Class doesn't need to be decorated with **[Serializable]** attribute.
  - Developer has a deep control about how each field is going to be serialized by using the attributes:
    - **[XmlAttribute]** : over a field, marks that the field will be serialized as attribute, instead of a node
    - **[XmlIgnore]** : won't serialize that field. The same as NonSerializable, but just for the XmlSerializer.
    - **[XmlElement (ElementName="NewName"]**: Allows you to rename the field when being serialized.
- **CONS**
  - It is more **verbose** (less efficient) than **BinaryFormatter**.
  - Only public members will be serialized! (shallow serialization).

JSON Serializer

- **Json.NET** is a popular open source JSON serializer.
  Info: http://james.newtonking.com/pages/json-net.aspx
- Flexible JSON serializer for converting between .NET objects and JSON.
- High performance, faster than .NET's built-in JSON serializers
- Write indented, easy to read JSON
- The serializer is a good choice when the JSON you are reading or writing maps closely to a .NET class.

```
Product product = new Product();
…
// Convert .Net object to JSON
string json = JsonConvert.SerializeObject(product);
// Convert JSON string to .Net object
Product deserializedProduct =
                 JsonConvert.DeserializeObject<Product>(json);
```

XML DATA FILES

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

XML Processing Options

XML serializing is just one way to work with XML data files – the .Net framework have several other technologies to process XML data (found in or below ns: System.Xml):

- XmlReader

  Fast forward-only access to XML data.
- XmlWriter

  Fast forward-only generate XML data.
- XmlDocument

  Implements the W3C DOM Level 1 Core and DOM Level 2 Core interfaces for reading and creating XML documents.
- XPathNavigator

  Provides several editing options and navigation capabilities over XML in an XmlDocument or an XPathDocument.
- LINQ to XML

XmlReader Example

```
using (XmlReader reader = XmlReader.Create("book3.xml"))
{
  // Parse the XML document. ReadString is used to
  // read the text content of the elements.
  reader.Read();
  reader.ReadStartElement("book");
  reader.ReadStartElement("title");
  Console.Write("The content of the title element: ");
  Console.WriteLine(reader.ReadString());
  reader.ReadEndElement();
  reader.ReadStartElement("price");
  Console.Write("The content of the price element: ");
  Console.WriteLine(reader.ReadString());
  reader.ReadEndElement();
  reader.ReadEndElement();
}
```

XmlDocument Example

```
using System;
using System.IO;
using System.Xml;

public class Sample
{
  public static void Main()
  {
     //Create the XmlDocument.
     XmlDocument doc = new XmlDocument();
     doc.LoadXml("<book genre='novel' ISBN='1-861001-57-5'>" +
               "<title>Pride And Prejudice</title>" + "</book>");
     //Create a new node and add it to the document.
     XmlNode elem = doc.CreateNode(XmlNodeType.Element, "price",
                                                       null);
    elem.InnerText = "19.95";
    doc.DocumentElement.AppendChild(elem);
    Console.WriteLine("Display the modified XML...");
    doc.Save(Console.Out);
  }
}
```

References

- XML Processing Options
  http://msdn.microsoft.com/en-us/library/bb669131.aspx
- XML serialization tutorial
  http://www.switchonthecode.com/tutorials/csharp-tutorial-xml-serialization
- Improving XML Performance
  http://msdn.microsoft.com/en-us/library/ff647804.aspx
- Json.NET
  Info: http://james.newtonking.com/pages/json-net.aspx