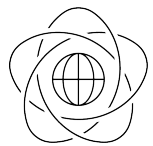
# Programming 3 Advanced

## Streams, Filesystem, Serialization

Tomasz Herman

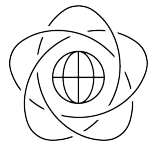
Faculty of Mathematics and Information Science
Warsaw University of Technology

Lecture 10, 8 grudnia 2024



# Outline





# Streams architecture

**Stream** abstract base class for reading and writing bytes

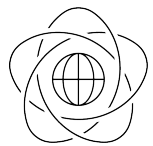
**Backing store streams** provide access to underlying data storage

**Decorator streams** wrap other streams, transforming the data

**Stream adapters** wrap the stream in a class with a specialized methods

## Documentation

`https://learn.microsoft.com/en-us/dotnet/standard/io/#streams`



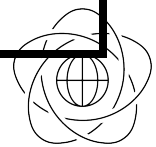
# Stream operations

## Streams involve three operations:

- Reading
- Writing
- Seeking

## Checking capabilities:

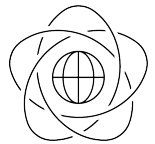
```
1  using Stream stream = new FileStream("lorem.txt",
2                                       FileMode.Open);
3
4  Console.WriteLine($"Can read: {stream.CanRead}");
5  Console.WriteLine($"Can write: {stream.CanWrite}");
6  Console.WriteLine($"Can Seek: {stream.CanSeek}");
```



# Stream

## Reading byte by byte

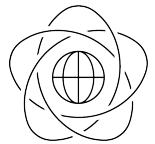
```
1  using Stream stream = new FileStream("lorem.txt",
2                                       FileMode.Open);
3
4  int charRead;
5  while ((charRead = stream.ReadByte()) != -1)
6  {
7      Console.WriteLine(charRead);
8  }
```



# Stream

## Reading into buffer

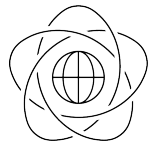
```
1  using Stream stream = new FileStream("lorem.txt",
2                                       FileMode.Open);
3
4  byte[] buffer = new byte[256];
5  int read;
6  while ((read = stream.Read(buffer, 0, buffer.Length)) > 0)
7  {
8      Console.WriteLine($"Read {read} bytes");
9  }
```



# Stream

## Writing byte by byte

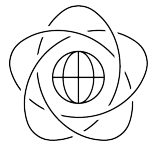
```
1  using Stream stream = new FileStream("lorem.txt",
2                                       FileMode.Create);
3
4  for (byte c = (byte)'a'; c < 'z'; c++)
5  {
6      stream.WriteByte(c);
7  }
```



# Stream

## Writing from buffer

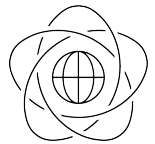
```
1  using Stream stream = new FileStream("lorem.txt",
2                                       FileMode.Create);
3
4  byte[] buffer = new byte[256];
5  for (int i = byte.MinValue; i <= byte.MaxValue; i++)
6  {
7      buffer[i - byte.MinValue] = (byte)i;
8  }
9  stream.Write(buffer, 0, buffer.Length);
```



# Stream

## Seeking

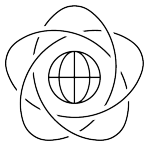
```
1  using Stream stream = new FileStream("lorem.txt",
2                                       FileMode.Open);
3  stream.Position = 0;
4  stream.Position = stream.Length;
5  stream.Seek(offset: 10, SeekOrigin.Begin);
6  stream.Seek(offset: -10, SeekOrigin.End);
7  stream.Seek(offset: -10, SeekOrigin.Current);
```



Façade Provides a simplified interface

Adapter Converts one interface to another so that it matches what the client is expecting

Decorator Dynamically adds responsibility to the interface by wrapping the original code



# Backing Store Streams

FileStream: Files

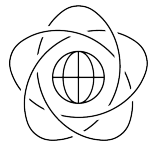
MemoryStream: In-memory byte array buffer

PipeStream: Pipes (IPC communication)

NetworkStream: Sockets (IPC communication)

## Flushing

The `Flush` method forces any internally buffered data to be written immediately to the backing store. `Flush` is called automatically when a stream is closed (or disposed).



# FileStream

```
1  // Façade methods:
2  FileStream fs1 = File.Create("file1.txt");
3  FileStream fs2 = File.OpenRead("file2.txt");
4  FileStream fs3 = File.OpenWrite("file3.txt");
5  FileStream fs4 = File.Open("file4.txt",
6      FileMode.OpenOrCreate, FileAccess.ReadWrite);
7  // Directly creating a FileStream:
8  FileStream fs5 = new FileStream("file5.txt",
9      FileMode.OpenOrCreate, FileAccess.ReadWrite);
```

## FileMode and FileAccess

FileMode: CreateNew, Create, Open, OpenOrCreate, Truncate, Append

FileAccess: Read, Write, ReadWrite

## Documentation

https://learn.microsoft.com/en-us/dotnet/fundamentals/runtime-libraries/system-io-filestream

**FileMode:**

CreateNew Throws exception if file exists

Create Truncates if file exists

Open Throws exception if file doesn't exist

OpenOrCreate

Truncate Throws exception if file doesn't exist

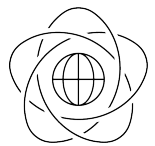
Append Opens at the end of file

**FileAccess:**

Read

Write

ReadWrite



# File

## Other façade methods

**Reads an entire file into memory:**

File.ReadAllText returns a string

File.ReadAllLines returns an array of strings

File.ReadAllBytes returns an array of bytes

**Lazily reads a file:**

File.ReadLines returns an `IEnumerable<string>`

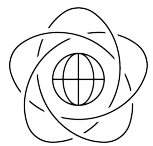
**Write an entire file:**

File.WriteAllText

File.WriteAllLines

File.WriteAllBytes

File.AppendAllText



# MemoryStream

```
1 using Stream stream = new FileStream("test.txt",
2     FileMode.Create);
3 var ms = new MemoryStream();
4 stream.CopyTo(ms);
```

## FileMode and FileAccess

FileMode: CreateNew, Create, Open, OpenOrCreate, Truncate, Append

FileAccess: Read, Write, ReadWrite

## Documentation

https://learn.microsoft.com/en-us/dotnet/api/system.io.memorystream?view=net-8.0

# PipeStream

## Pipes

PipeStream provides a simple means by which one process can communicate with another through the operating system's pipes protocol.

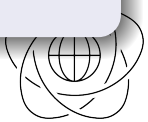
PipeStream an abstract class with four concrete subtypes:

AnonymousPipeServerStream and AnonymousPipeClientStream used for communication between parent and children processes

NamedPipeServerStream and NamedPipeClientStream used for communication between processes through 'name'

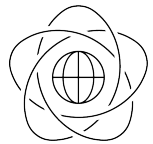
## Documentation

`https://learn.microsoft.com/en-us/dotnet/standard/io/pipe-operations`



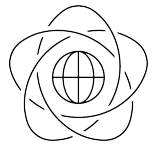
## Server Process:

```
1  using var pipe = new NamedPipeServerStream("p3_pipe");
2  pipe.WaitForConnection();
3
4  using var fs = File.OpenRead("lorem.txt");
5  byte[] buffer = new byte[BufferSize];
6  int read;
7  while ((read = fs.Read(buffer, 0, BufferSize)) > 0)
8  {
9      pipe.Write(buffer, 0, read);
10 }
```



## Client Process:

```
1  using var pipe = new NamedPipeClientStream("p3_pipe");
2  pipe.Connect();
3
4  using var fs = File.OpenWrite("lorem.txt");
5  byte[] buffer = new byte[BufferSize];
6  int read;
7  while ((read = pipe.Read(buffer, 0, BufferSize)) > 0)
8  {
9      fs.Write(buffer, 0, read);
10 }
```



# Decorator Steams

## Buffering Stream

BufferedStream Adds buffering capabilities to a stream

## Compression Streams

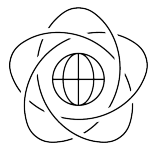
DeflateStream

GZipStream

BrotliStream Better compression ratio, slower compression

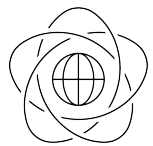
## Cryptography Stream

CryptoStream Adds encryption/decryption to a stream



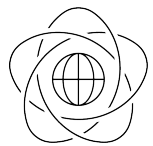
# BufferedStream

```
1  // Write 100K bytes to a file:
2  File.WriteAllBytes("file.bin", new byte [100000]);
3
4  using FileStream fs = File.OpenRead("file.bin");
5  // Add 20k bytes buffering
6  using BufferedStream bs = new BufferedStream(fs, 20000);
7
8  bs.ReadByte();
9  Console.WriteLine(fs.Position); // 20000
```



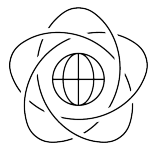
# Compressing

```
1  using FileStream fsIn = File.OpenRead("file.txt");
2  using FileStream fsOut = File.Create("file.txt.gz");
3  using var ds = new GZipStream(fsOut,
4                                CompressionMode.Compress);
5
6  byte[] buffer = new byte[4096];
7  int read;
8
9  while ((read = fsIn.Read(buffer, 0, buffer.Length)) > 0)
10 {
11     ds.Write(buffer, 0, read);
12 }
```



# Decompressing

```
1  using var fsIn = File.OpenRead("file.txt.gz");
2  using var ds = new GZipStream(fsIn,
3                                CompressionMode.Decompress);
4  using var fsOut = File.Create("file.txt");
5
6  byte[] buffer = new byte[4096];
7  int read;
8
9  while ((read = ds.Read(buffer, 0, buffer.Length)) > 0)
10 {
11     fsOut.Write(buffer, 0, read);
12 }
```



# Stream Adapters

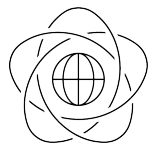
Text Adapters adapts stream to work with string and character data

Binary Adapters adapts stream to work with primitive types such as int, bool, string, and float

Xml Adapters adapts stream to work with reading/writing xml

## Adapters

An adapter wraps a stream, just like a decorator. Unlike a decorator, however, an adapter is not itself a stream; it typically hides the byte-oriented methods completely.



## Text Adapters

TextReader and TextWriter abstract base classes

StreamReader and StreamWriter translates the stream's bytes into characters or strings

StringReader and StringWriter uses in-memory strings

### StreamReader

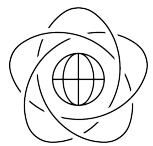
`https://learn.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=net-8.0`

### StreamWriter

`https://learn.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-8.0`

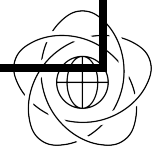
# StreamReader

```
1  using FileStream fs = File.OpenRead("lorem.txt");
2  using StreamReader sr = new StreamReader(fs);
3
4  while (sr.ReadLine() is { } line)
5  {
6      Console.WriteLine(line);
7  }
```



# StreamWriter

```
1  using FileStream fs = File.OpenWrite("fizzbuzz.txt");
2  using StreamWriter sw = new StreamWriter(fs);
3
4  for (int i = 1; i <= 100; i++)
5  {
6      sw.Write($"{i} : ");
7      if (i % 3 == 0 && i % 5 == 0)
8          sw.WriteLine("FizzBuzz");
9      else if (i % 3 == 0)
10         sw.WriteLine("Fizz");
11     else if (i % 5 == 0)
12         sw.WriteLine("Buzz");
13     else
14         sw.WriteLine(i);
15 }
```



# Binary Adapters

## Binary Adapters

BinaryReader and BinaryWriter Converts primitive data types to binary values

### BinaryReader

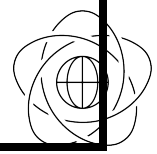
https://learn.microsoft.com/en-us/dotnet/api/system.io.binaryreader?view=net-8.0

### BinaryWriter

https://learn.microsoft.com/en-us/dotnet/api/system.io.binarywriter?view=net-9.0

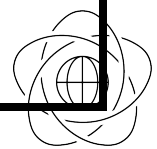
# BinaryReader

```
1  using FileStream fs = File.OpenRead("player.bin");
2  using BinaryReader br = new BinaryReader(fs);
3
4  string name = br.ReadString();
5  int health = br.ReadInt32();
6  long experience = br.ReadInt64();
7  long money = br.ReadInt64();
8  float posX = br.ReadSingle();
9  float posY = br.ReadSingle();
10
11 Player player = new Player(name, health, experience,
12     money, new Vector2(posX, posY));
13 Console.WriteLine(player);
14
15
16 public record Player(string Name, int Health,
17     long Experience, long Money, Vector2 Position);
```



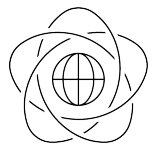
# BinaryWriter

```
1  using FileStream fs = File.OpenWrite("player.bin");
2  using BinaryWriter bw = new BinaryWriter(fs);
3
4  Player player = new Player("Bob", 100, 2500,
5      10, new Vector2(48.5f, 32.5f));
6
7  bw.Write(player.Name);
8  bw.Write(player.Health);
9  bw.Write(player.Experience);
10 bw.Write(player.Money);
11 bw.Write(player.Position.X);
12 bw.Write(player.Position.Y);
13
14
15 public record Player(string Name, int Health,
16     long Experience, long Money, Vector2 Position);
```



# Working with Zip files

```
1  ZipFile.CreateFromDirectory(".", "../output.zip");
2  ZipFile.ExtractToDirectory("../output.zip", "../output");
3
4  using ZipArchive zip = ZipFile.Open("../output.zip",
5      ZipArchiveMode.Read);
6  foreach (ZipArchiveEntry entry in zip.Entries)
7      Console.WriteLine(
8          $"{entry.FullName,-32} size: {entry.Length}");
```



# Interacting with a filesystem

## Files

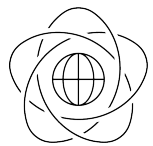
- File (static class)
- FileInfo

## Directories

- Directory (static class)
- DirectoryInfo

## Partitions

- DriveInfo



# File class

Checking if exists: Exists

Manipulating file: Delete, Copy, Move, Replace, CreateSymbolicLink

Checking/setting attributes: GetAttributes, SetAttributes

Encrypting/decrypting (Windows only): Encrypt, Decrypt

Checking/updating timestamps: GetCreationTime, GetLastAccessTime, GetLastWriteTime, SetCreationTime, SetLastAccessTime, SetLastWriteTime

Checking/updating permissions: GetUnixFileMode, SetUnixFileMode

## Documentation

`https://learn.microsoft.com/en-us/dotnet/api/system.io.file?view=net-8.0`

# Directory class

Similar method set as in File: Exists, Delete, Move, etc.

Creating new directory: CreateDirectory

Getting content: GetFiles, GetDirectories, GetFileSystemEntries,

Getting content (lazy evaluation): EnumerateFiles, EnumerateDirectories, EnumerateFileSystemEntries

Working with current directory GetCurrentDirectory, SetCurrentDirectory

Utilities: GetParent, GetDirectoryRoot

Getting Unix mount points: GetLogicalDrives

## Documentation

`https://learn.microsoft.com/en-us/dotnet/api/system.io.directory?view=net-8.0`

# Path class

Manipulating paths: IsPathRooted, GetPathRoot, GetDirectoryName, GetFileName, GetFullPath, **Combine**

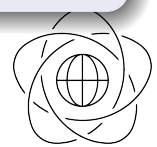
Working with extensions: HasExtension, GetExtension, GetFileNameWithoutExtension, ChangeExtension

Utils: DirectorySeparatorChar, AltDirectorySeparatorChar, PathSeparator, VolumeSeparatorChar, GetInvalidPathChars, GetInvalidFileNameChars

Working with temporary files: GetTempPath, GetRandomFileName, GetTempFileName

## Documentation

`https://learn.microsoft.com/en-us/dotnet/api/system.io.path?view=net-8.0`



# FileInfo and DirectoryInfo

If you are performing multiple operations on the same file/directory, it can be more efficient to use FileInfo/DirectoryInfo instance methods instead of the corresponding static methods of the File/Directory class.

## FileInfo documentation

`https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo?view=net-8.0`

## DirectoryInfo documentation

`https://learn.microsoft.com/en-us/dotnet/api/system.io.directoryinfo?view=net-8.0`

# DriveInfo

```
1  DriveInfo c = new DriveInfo("C");
2  long totalSize = c.TotalSize;
3  long freeBytes = c.TotalFreeSpace;
4  long availableBytes = c.AvailableFreeSpace;
5  Console.WriteLine($"Total size: {totalSize}");
6  Console.WriteLine($"Free size: {freeBytes}");
7  Console.WriteLine($"Available size: {availableBytes}");
8
9  foreach (DriveInfo d in DriveInfo.GetDrives())
10 {
11     Console.WriteLine(d.Name);
12 }
```

## Documentation

https://learn.microsoft.com/en-us/dotnet/api/system.io.driveinfo?view=net-8.0

# Filesystem events

```
1  using var watcher = new FileSystemWatcher(path, filter);
2  watcher.Created += OnCreated;
3  watcher.Changed += OnChanged;
4  watcher.Deleted += OnDeleted;
5  watcher.Renamed += OnRenamed;
6  watcher.Error += OnError;
7  watcher.IncludeSubdirectories = includeSubDirs;
8  watcher.EnableRaisingEvents = true;
9  Console.WriteLine("Press enter to end");
10 Console.ReadLine();
```

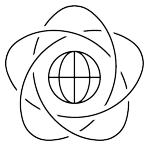
## Documentation

https://learn.microsoft.com/en-us/dotnet/api/system.io.filesystemwatcher?view=net-9.0

# Serialization

Serialization and deserialization are typically used to do the following:

- Transmit objects across a network or application boundary
- Store representations of objects within a file or database

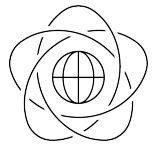


# Serialization engines

- Xml serializer
- Json serializer
- Data contract serializer (Xml and Json)
- Binary serializer (obsolete)

## Binary serialization

The BinaryFormatter type is dangerous and is not recommended for data processing:
`https://aka.ms/binaryformatter`



# XmlSerializer

## Serialization

```
1  var weatherForecast = new WeatherForecast
2  {
3      // ...
4  };
5
6  var stringWriter = new StringWriter();
7  var xmlSerializer = new XmlSerializer(
8      typeof(WeatherForecast));
9  xmlSerializer.Serialize(stringWriter, weatherForecast);
10
11 Console.WriteLine(stringWriter.ToString());
```

### Serialization

https://learn.microsoft.com/en-us/dotnet/standard/serialization/how-to-serialize-an-object

# XmlSerializer

## Deserialization

```
1  string xml = """
2               // ...
3               """;
4
5  var stringReader = new StringReader(xml);
6  var xmlSerializer = new XmlSerializer(
7      typeof(WeatherForecast));
8  WeatherForecast? wf = (WeatherForecast?)xmlSerializer
9      .Deserialize(stringReader);
10
11 Console.WriteLine($"Date: {wf?.Date}");
12 Console.WriteLine($"Summary: {wf?.Summary}");
```

### Deserialization

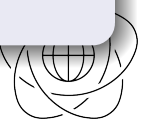
https://learn.microsoft.com/en-us/dotnet/standard/serialization/how-to-deserialize-an-object

## Example

```
1  public class Person
2  {
3      [XmlElement("FirstName")]
4      public string Name;
5      [XmlIgnore()]
6      public int Age;
7  }
```

## Documentation

https://learn.microsoft.com/en-us/dotnet/standard/serialization/controlling-xml-serialization-using-attributes



# JsonSerializer

## Serialization

```
1  var weatherForecast = new WeatherForecast
2  {
3      // ...
4  };
5
6  var options = new JsonSerializerOptions
7      { WriteIndented = true };
8  string jsonString = JsonSerializer
9      .Serialize(weatherForecast, options);
```

### Serialization

https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to

# JsonSerializer

## Deserialization

```
1  string jsonString =
2      """
3      {
4        // ...
5      }
6      """;
7
8  WeatherForecast? wf =
9      JsonSerializer.Deserialize<WeatherForecast>(jsonString);
10
11 Console.WriteLine($"Date: {wf?.Date}");
12 Console.WriteLine($"Summary: {wf?.Summary}");
```

### Deserialization

https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/deserialization

## Example

```
1  public class Person
2  {
3      [JsonPropertyName("FirstName")]
4      public string Name { get; set; }
5      [JsonInclude()]
6      public DateTime DateOfBirth;
7      [JsonIgnore()]
8      public int Age { get; set; }
9  }
```