



CS 2530

Serialization



Review: System.IO

- StreamReader
- StreamWriter
- File
- FileInfo
- Directory
- DirectoryInfo
- Path



Review: System.IO

- StreamReader . . derives from **TextReader**
- StreamWriter . . . derives from **TextWriter**
- File
- FileInfo
- Directory
- DirectoryInfo
- Path



Review: System.IO

- StreamReader . . . derives from **TextReader**
- StreamWriter . . . derives from **TextWriter**
- File **static class**
- FileInfo
- Directory **static class**
- DirectoryInfo
- Path **static class**



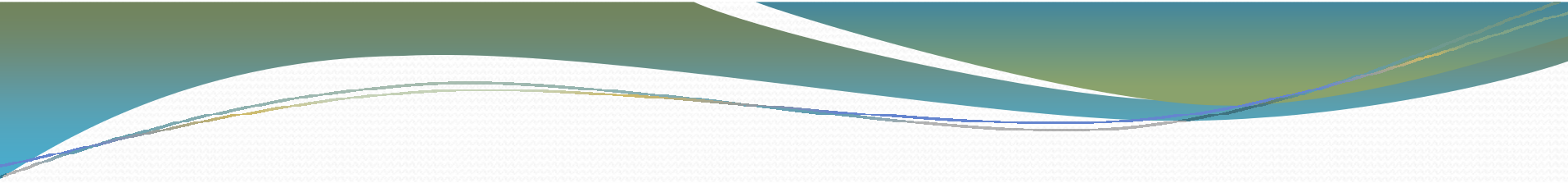
System.IO

- **Stream ..** abstract class that provides a generic view of a sequence of bytes..



System.IO

- **Stream ..** abstract class that provides a generic view of a sequence of bytes.
- **FileStream ...** Exposes a stream around a file, supporting synchronous and asynchronous read / write operations.



When should I use
StreamWriter and when
should I use FileStream ?

FileStream vs StreamWriter

- **FileStream** is a Stream

sequence of bytes

Streams only deal with `byte[]` data

- **StreamWriter** is a `TextWriter`

It converts `text` (string or char) to `byte[]` data for the underlying stream



Serialization

- **Serialization is the process of converting an object into a serial format (e.g. stream of bytes or XML) so it can be stored or transmitted.**



Serialization

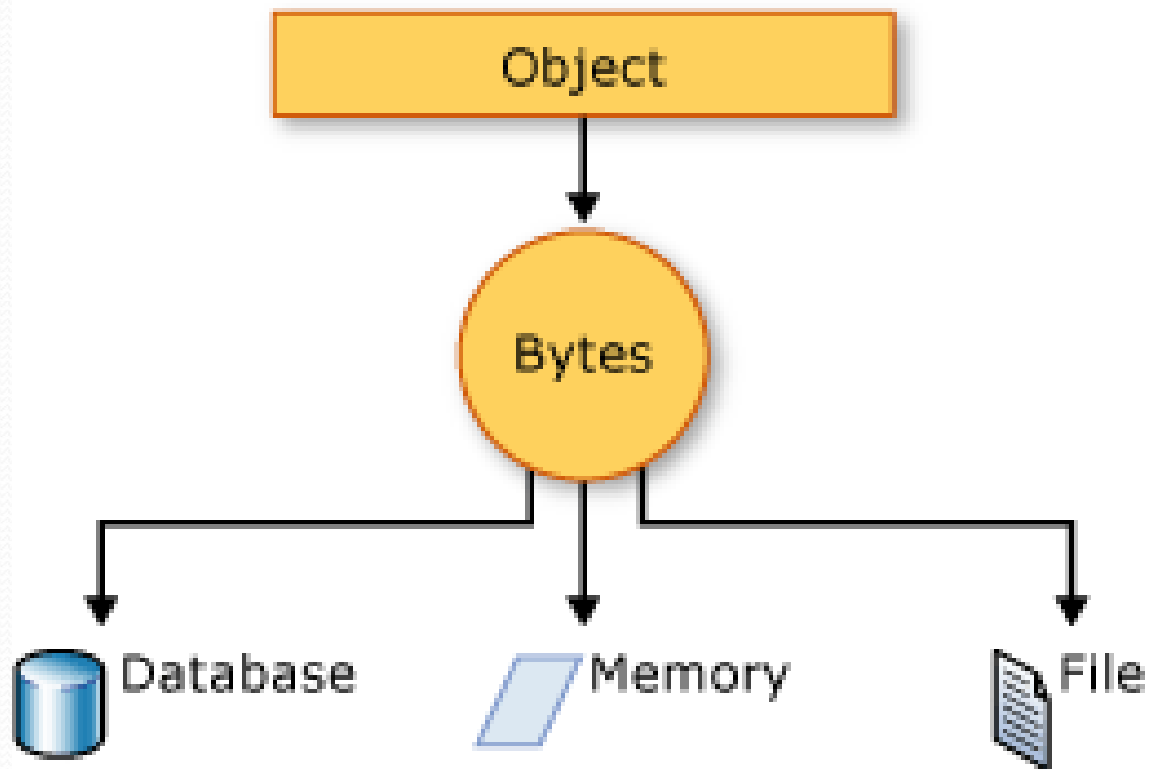
- Serialization is the process of converting an object into a serial format (e.g. stream of bytes or XML) so it can be stored or transmitted.
- Main purpose:
to save the state of an object in order to be able to recreate it when needed.
- The reverse process is called deserialization.



Serialization

- Serialized object:
 - is an **object** represented **in a serial format** (e.g. as **sequence of bytes**)
 - it includes the data and type of the object as well as the types of the data stored in the object.

Serialization





Serialization

To serialize an object you need 3 things:

1. The object to be serialized
2. A stream to contain the serialized object
3. A formatter

`System.Runtime.Serialization` contains the classes necessary for serializing and deserializing objects.



Attributes

- Attributes **associate meta-data** with the code
- In order for an object to be serializable the type needs to have the **SerializableAttribute** applied.
- If you do not want a field within your class to be serializable, apply the **NonSerializedAttribute** (e.g. used for data structures that are specific to a given environment like pointers or handles)



Binary vs XML Serialization

- Binary Serialization
 - Better performance than XML serialization
- XML Serialization
 - Serialized code is human readable
 - Greater flexibility for object sharing (interoperability)



Type (formatter)	Binary Formatter	Soap Formatter	XmlSerializer class
Best performance	Yes	No	No
Readable with other platforms (non .NET)	No	Yes	Yes
Serialize object private members	Yes (deep serialization)	Yes (deep serialization)	No (shallow serialization)
Serialize generic collections (*)	Yes	No	Yes
Easy control of how each member is serialized	No (**)	No (**)	Yes (by using attributes)
Saves metadata in output stream for deserialization	Yes	Yes	No

Serializes public and private members

Type (formatter)	Binary Formatter	Soap Formatter	XmlSerializer class
Best performance	Yes	No	No
Readable with other platforms (non .NET)	No	Yes	Yes
Serialize object private members	Yes (deep serialization)	Yes (deep serialization)	No (shallow serialization)
Serialize generic collections (*)	Yes	No	Yes
Easy control of how each member is serialized	No (**)	No (**)	Yes (by using attributes)
Saves metadata in output stream for deserialization	Yes	Yes	No

(**) ... possible but not easy

[Larger Comparison Table](#)



Class BinaryFormatter

- `System.Runtime.Serialization.Formatters.Binary`
- Serializes and deserializes an object, or an entire graph of connected objects, in binary format.
- The annotation `[Serializable]` needs to be put right above the type declaration that is going to be serialized



Example: Serialize

```
using (Stream fs = new FileStream("Stocks.dat", FileMode.Create))
{
    BinaryFormatter formatter = new BinaryFormatter();
    try
    {
        formatter.Serialize(fs, stocks);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Serialization failed: {0}", ex.Message);
    }
}
```



Example: Deserialize

```
using (Stream fs = new FileStream("Stocks.dat", FileMode.Open))
{
    BinaryFormatter formatter = new BinaryFormatter();
    try
    {
        stockData = formatter.Deserialize(fs) as List<Stock>;
    }
    catch (Exception ex)
    {
        Console.WriteLine("Deserialization failed: " +
            ex.Message);
    }
}
```




Class XMLSerialization

- `System.Xml.Serialization.XmlSerializer`
- Serializes and deserializes objects into and from XML documents.
- Some restrictions apply:
 - Only public properties and fields can be serialized
 - Properties must have public setter and getter
 - Class needs a default constructor