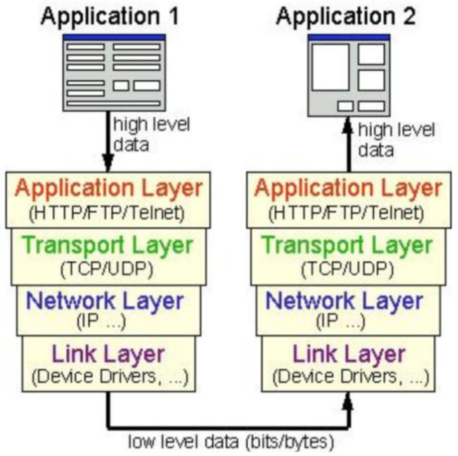
Network Programming

Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

Input & Output Introduction

1. Introduction to Input/Output

- I/O applies to network data transfers, as well as saving and loading to computer’s hard disk.
- The underlying I/O operations that are common to both types of transfers:
  - How to read and write to disk, using streams.
  - How to convert complex objects into a format that can be written to a stream.



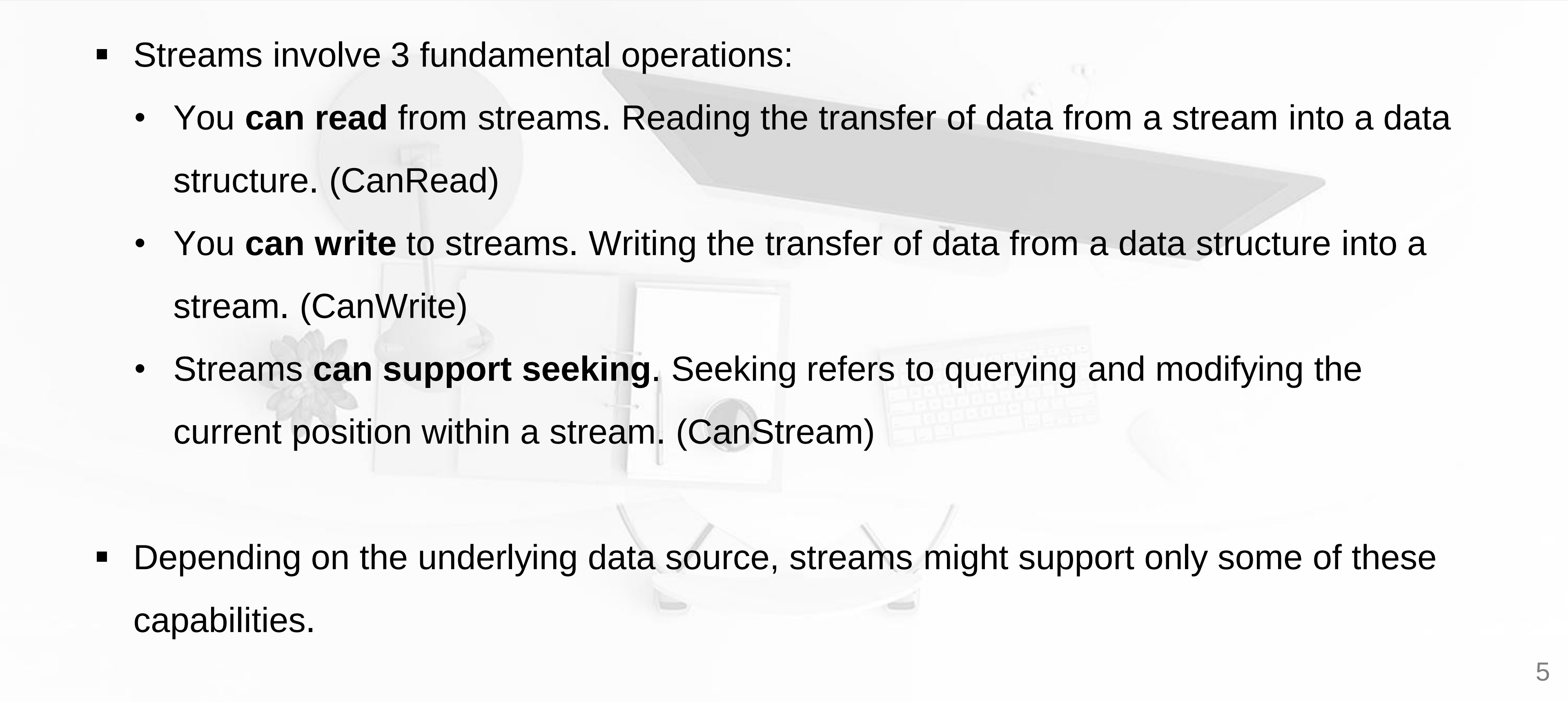
A large part of what **network programs do** is **simple input and output**

2. Streams

- In order to provide similar programmatic interfaces to the broad range of I/O devices with which a programmer has to contend, a stream-based architecture was developed. I/O devices can be anything from printers to hard disks to network interfaces.

- Not all devices support the same functions, for example: read a file vs read (download) a web page.

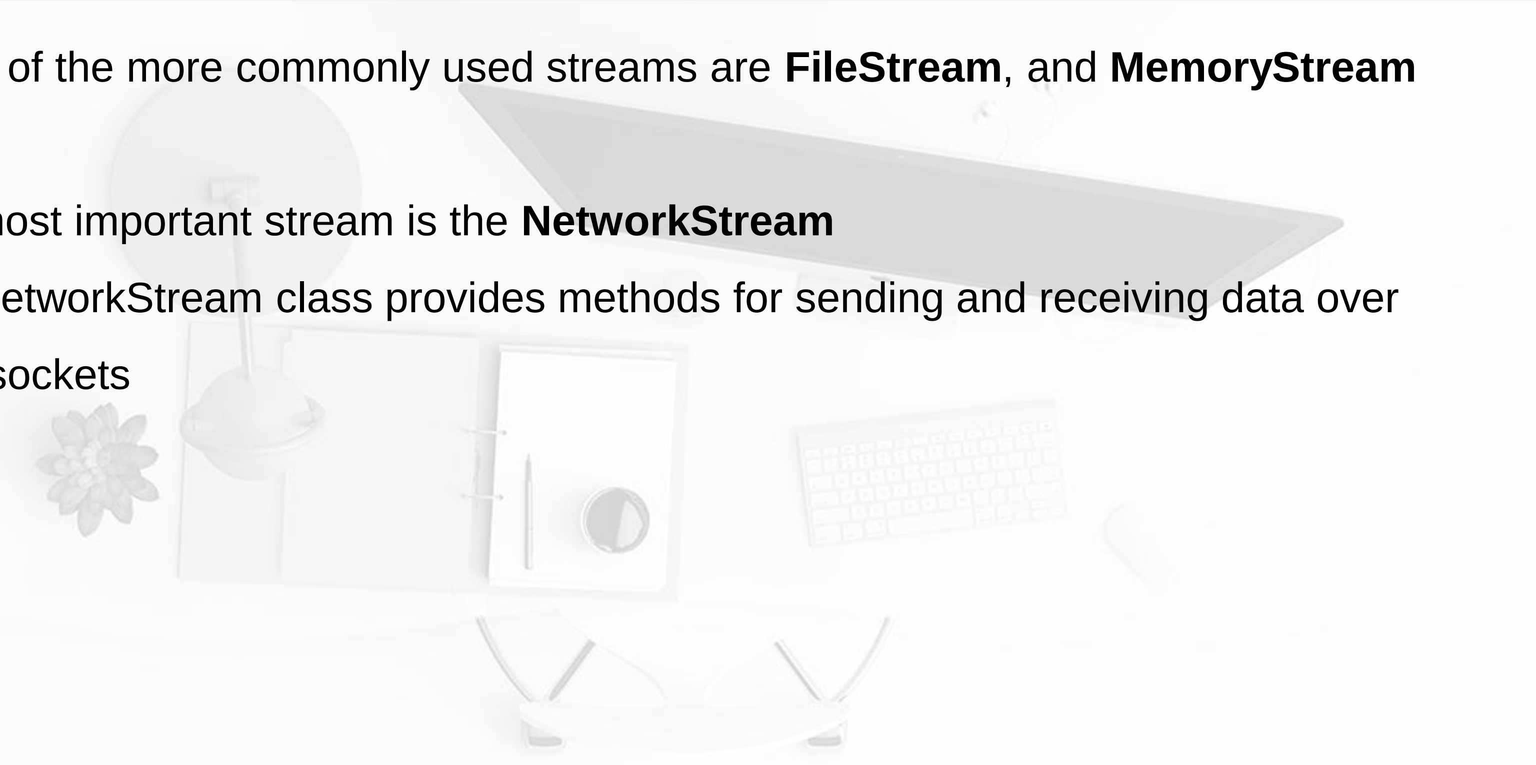
2. Streams



- Streams involve 3 fundamental operations:
  - You **can read** from streams. Reading the transfer of data from a stream into a data structure. (CanRead)
  - You **can write** to streams. Writing the transfer of data from a data structure into a stream. (CanWrite)
  - Streams **can support seeking**. Seeking refers to querying and modifying the current position within a stream. (CanStream)

- Depending on the underlying data source, streams might support only some of these capabilities.

2. Streams



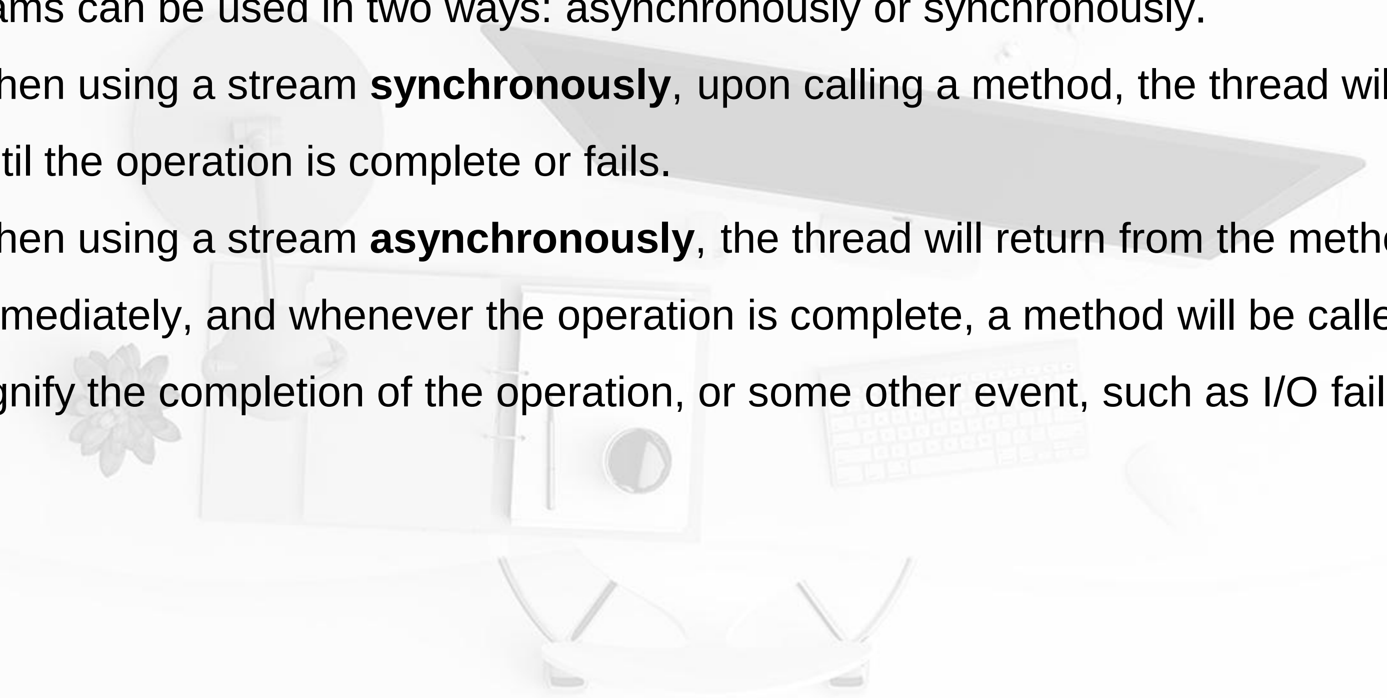
- Some of the more commonly used streams are **FileStream**, and **MemoryStream**

- The most important stream is the **NetworkStream**

  The NetworkStream class provides methods for sending and receiving data over

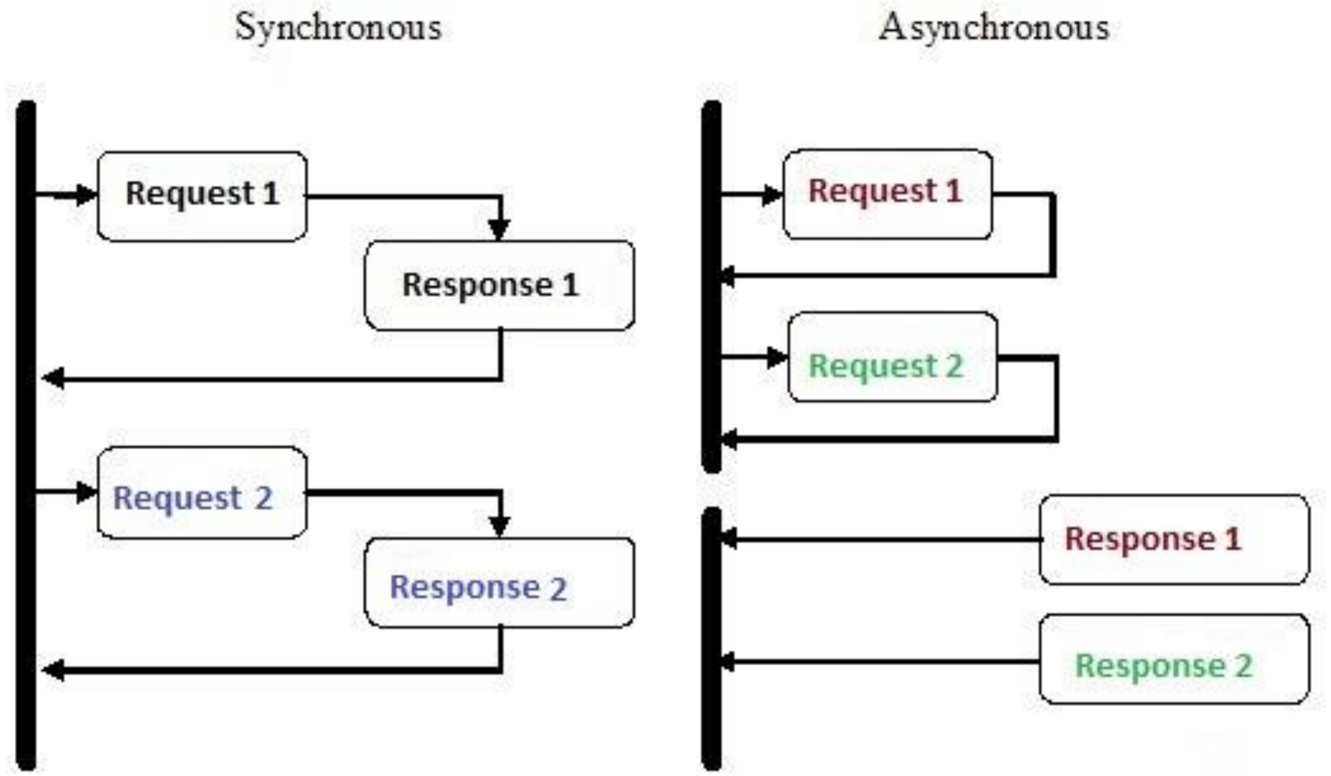
Stream sockets

2. Streams

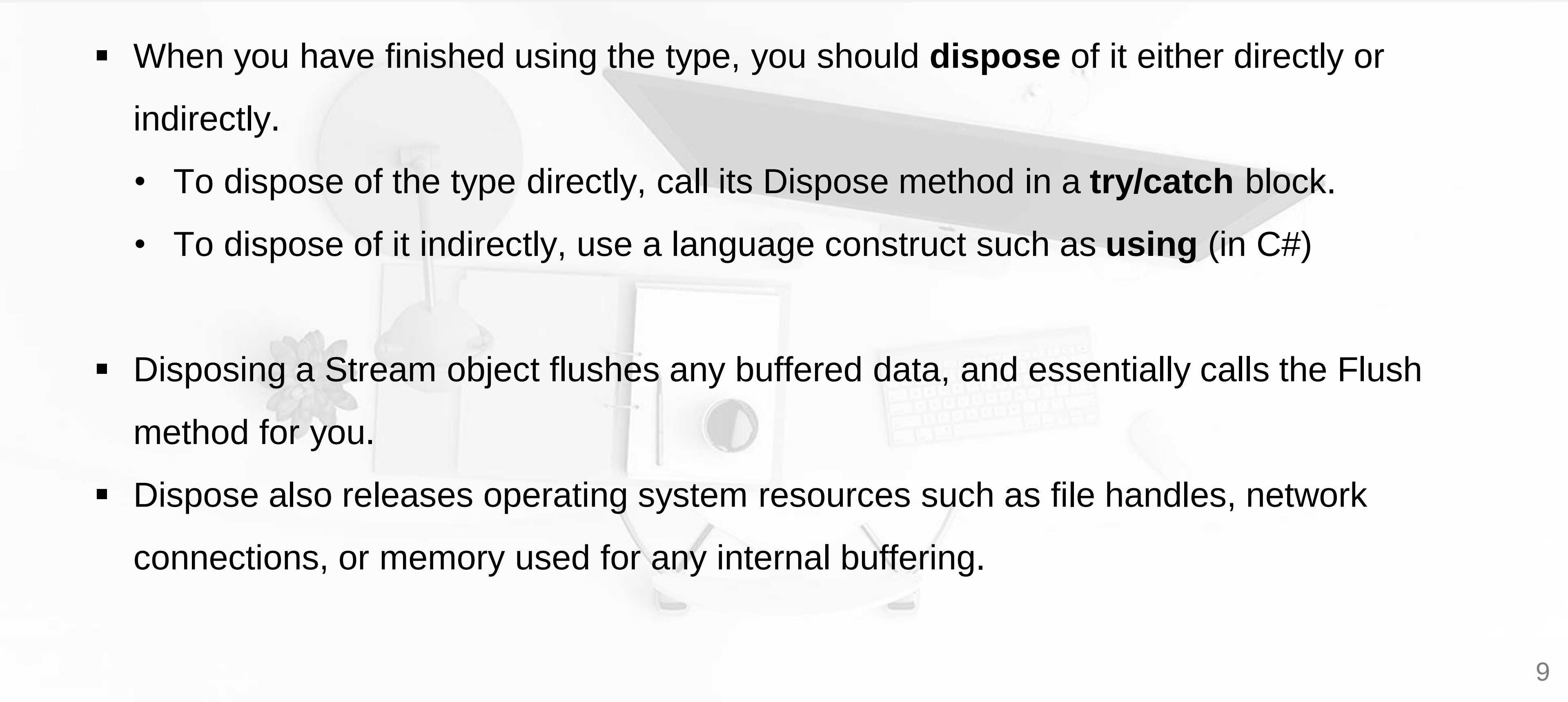


- Streams can be used in two ways: asynchronously or synchronously.
  - When using a stream **synchronously**, upon calling a method, the thread will halt until the operation is complete or fails.
  - When using a stream **asynchronously**, the thread will return from the method call immediately, and whenever the operation is complete, a method will be called to signify the completion of the operation, or some other event, such as I/O failure.

2. Streams


Synchronous
Asynchronous
Request 1
Response 1
Request 2
Response 2
Request 1
Request 2
Response 1
Response 2

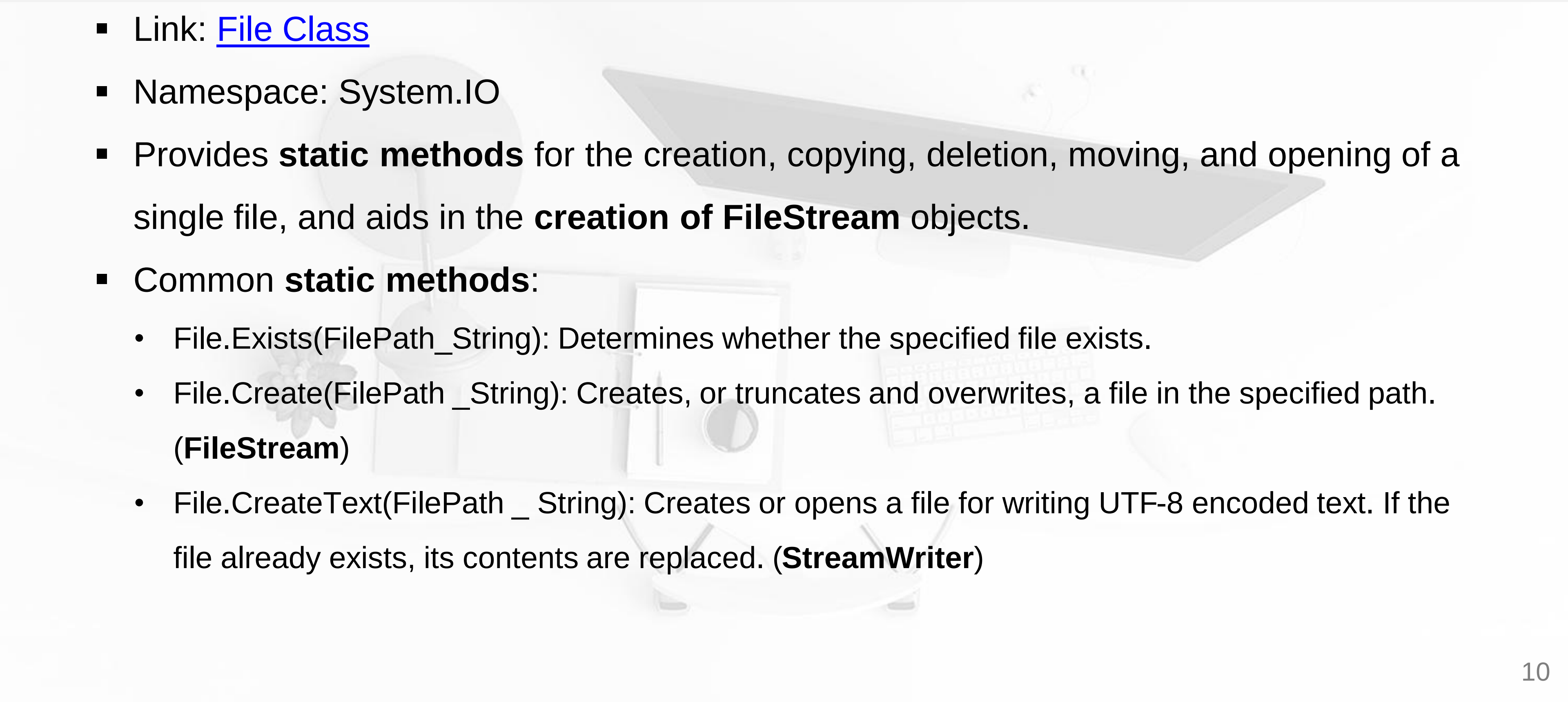
2. Streams



- When you have finished using the type, you should **dispose** of it either directly or indirectly.
  - To dispose of the type directly, call its Dispose method in a **try/catch** block.
  - To dispose of it indirectly, use a language construct such as **using** (in C#)

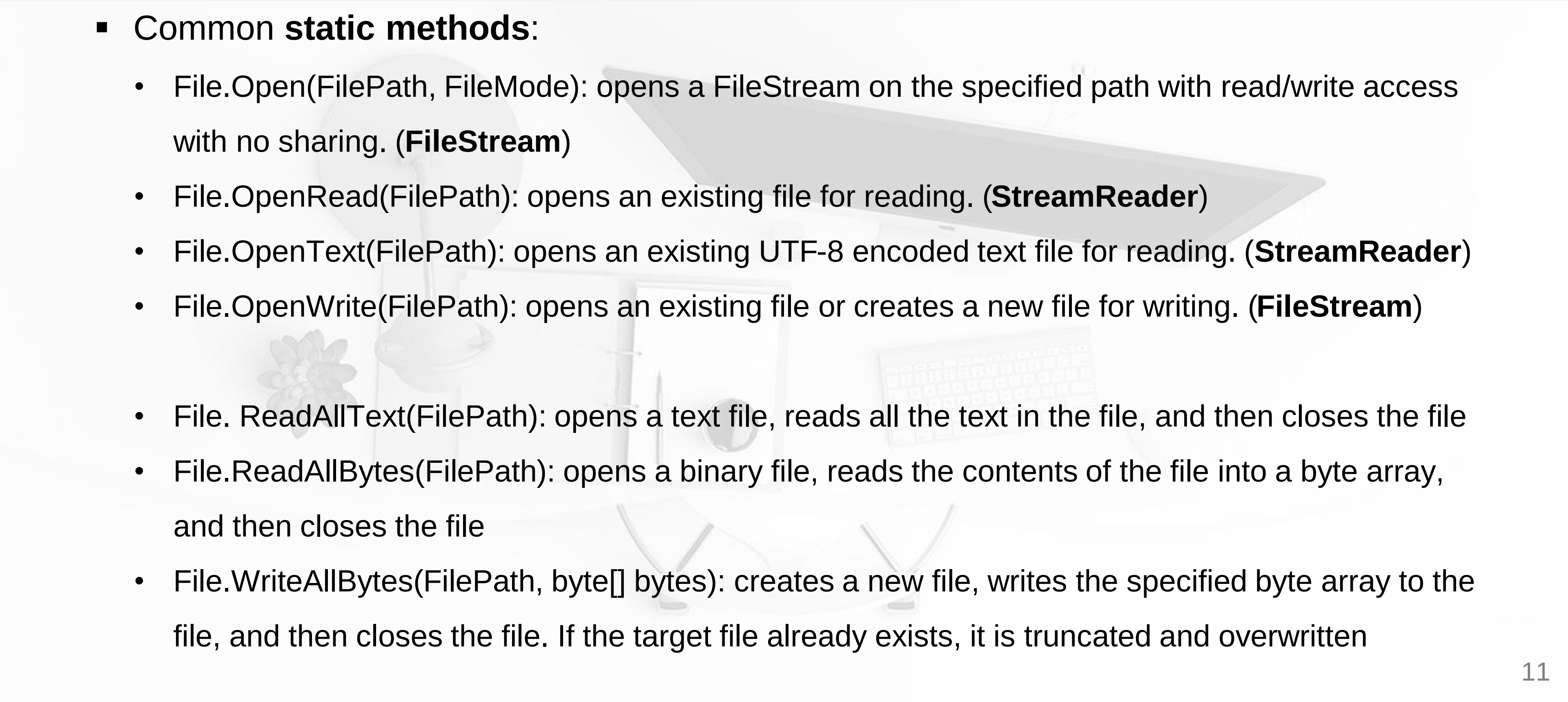
- Disposing a Stream object flushes any buffered data, and essentially calls the Flush method for you.
- Dispose also releases operating system resources such as file handles, network connections, or memory used for any internal buffering.

3. File Class



- Link: File Class
- Namespace: System.IO
- Provides **static methods** for the creation, copying, deletion, moving, and opening of a single file, and aids in the **creation of FileStream** objects.
- Common **static methods**:
  - File.Exists(FilePath_String): Determines whether the specified file exists.
  - File.Create(FilePath _String): Creates, or truncates and overwrites, a file in the specified path. (**FileStream**)
  - File.CreateText(FilePath _ String): Creates or opens a file for writing UTF-8 encoded text. If the file already exists, its contents are replaced. (**StreamWriter**)

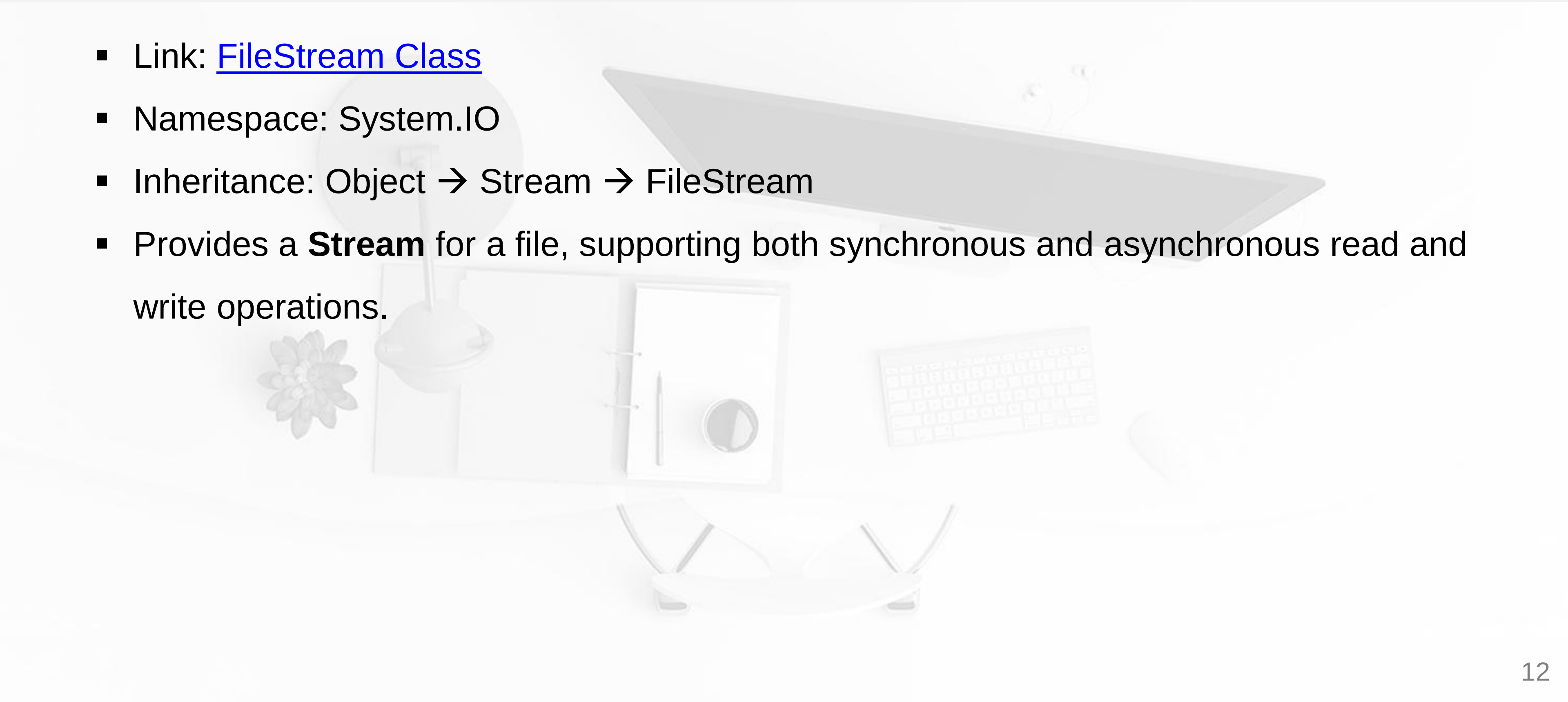
3. File Class



- Common **static methods**:
  - File.Open(FilePath, FileMode): opens a FileStream on the specified path with read/write access with no sharing. (**FileStream**)
  - File.OpenRead(FilePath): opens an existing file for reading. (**StreamReader**)
  - File.OpenText(FilePath): opens an existing UTF-8 encoded text file for reading. (**StreamReader**)
  - File.OpenWrite(FilePath): opens an existing file or creates a new file for writing. (**FileStream**)

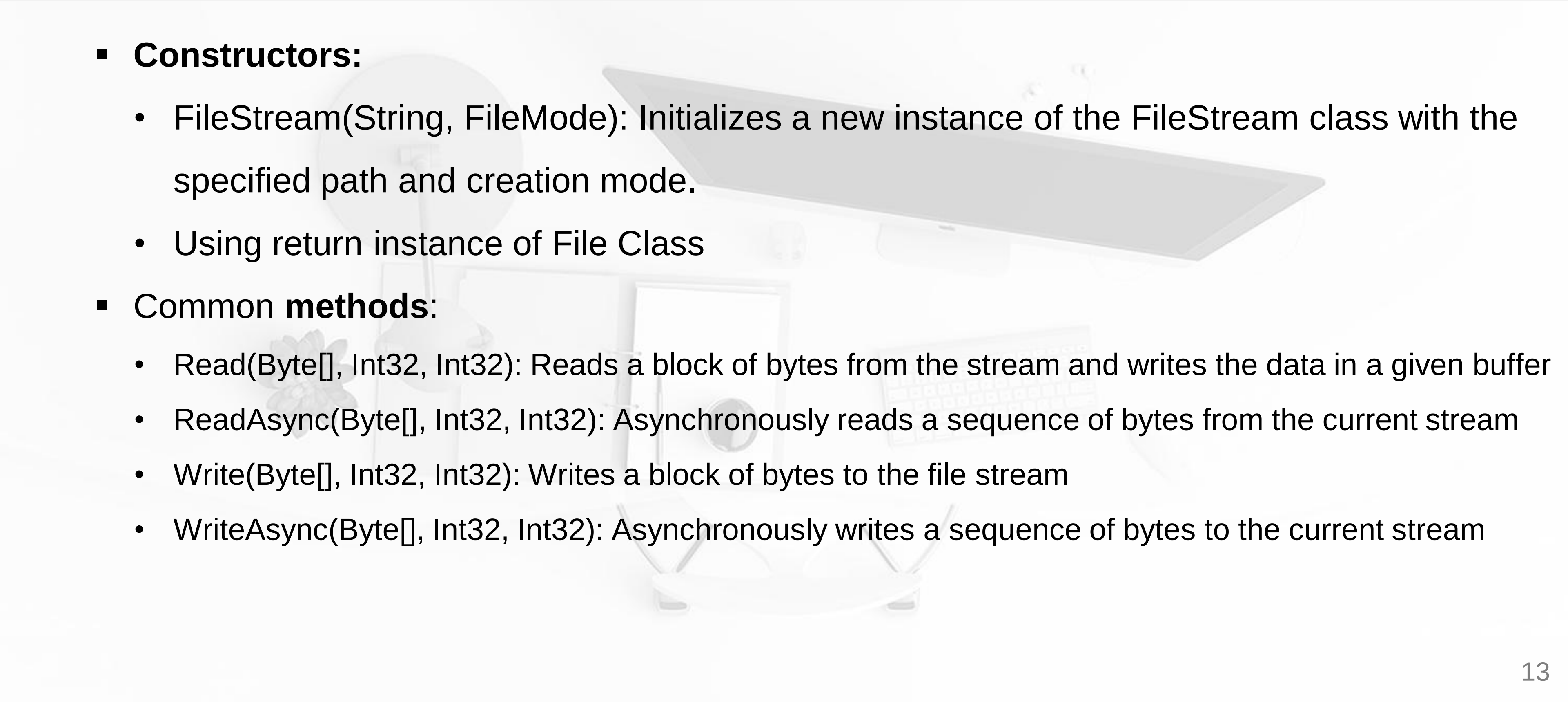
  - File. ReadAllText(FilePath): opens a text file, reads all the text in the file, and then closes the file
  - File.ReadAllBytes(FilePath): opens a binary file, reads the contents of the file into a byte array, and then closes the file
  - File.WriteAllBytes(FilePath, byte[] bytes): creates a new file, writes the specified byte array to the file, and then closes the file. If the target file already exists, it is truncated and overwritten

4. File Stream



- Link: FileStream Class
- Namespace: System.IO
- Inheritance: Object → Stream → FileStream
- Provides a **Stream** for a file, supporting both synchronous and asynchronous read and write operations.

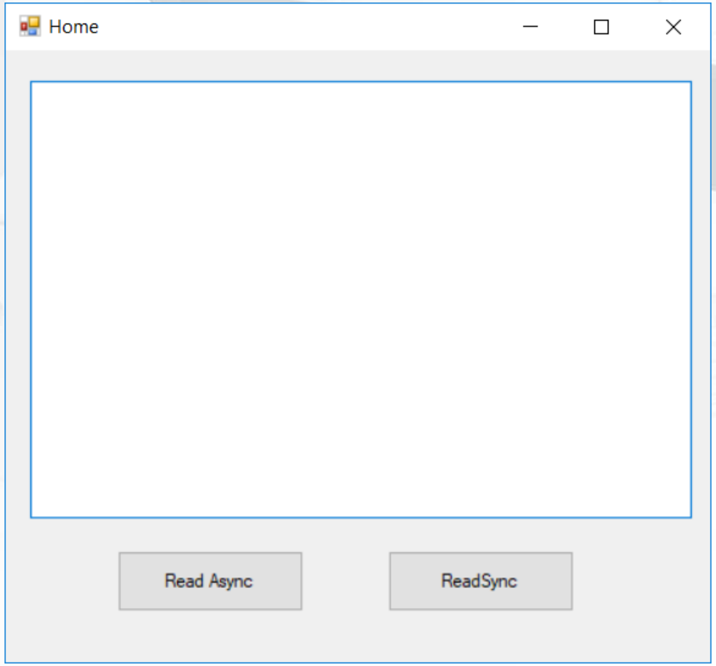
4. File Stream



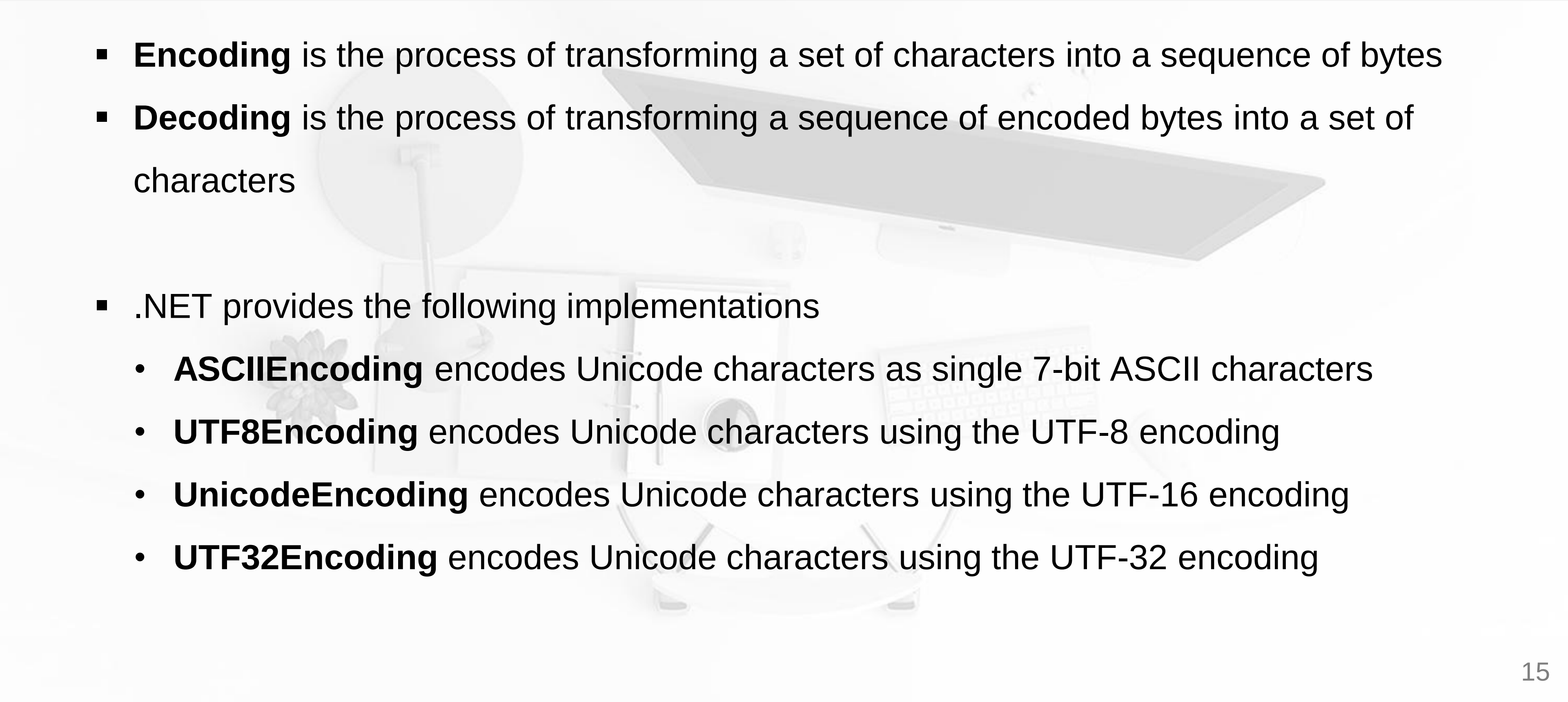
- **Constructors:**
  - FileStream(String, FileMode): Initializes a new instance of the FileStream class with the specified path and creation mode.
  - Using return instance of File Class
- Common **methods**:
  - Read(Byte[], Int32, Int32): Reads a block of bytes from the stream and writes the data in a given buffer
  - ReadAsync(Byte[], Int32, Int32): Asynchronously reads a sequence of bytes from the current stream
  - Write(Byte[], Int32, Int32): Writes a block of bytes to the file stream
  - WriteAsync(Byte[], Int32, Int32): Asynchronously writes a sequence of bytes to the current stream

4. File Stream

Exercise: Write a program to read, write data using File Stream



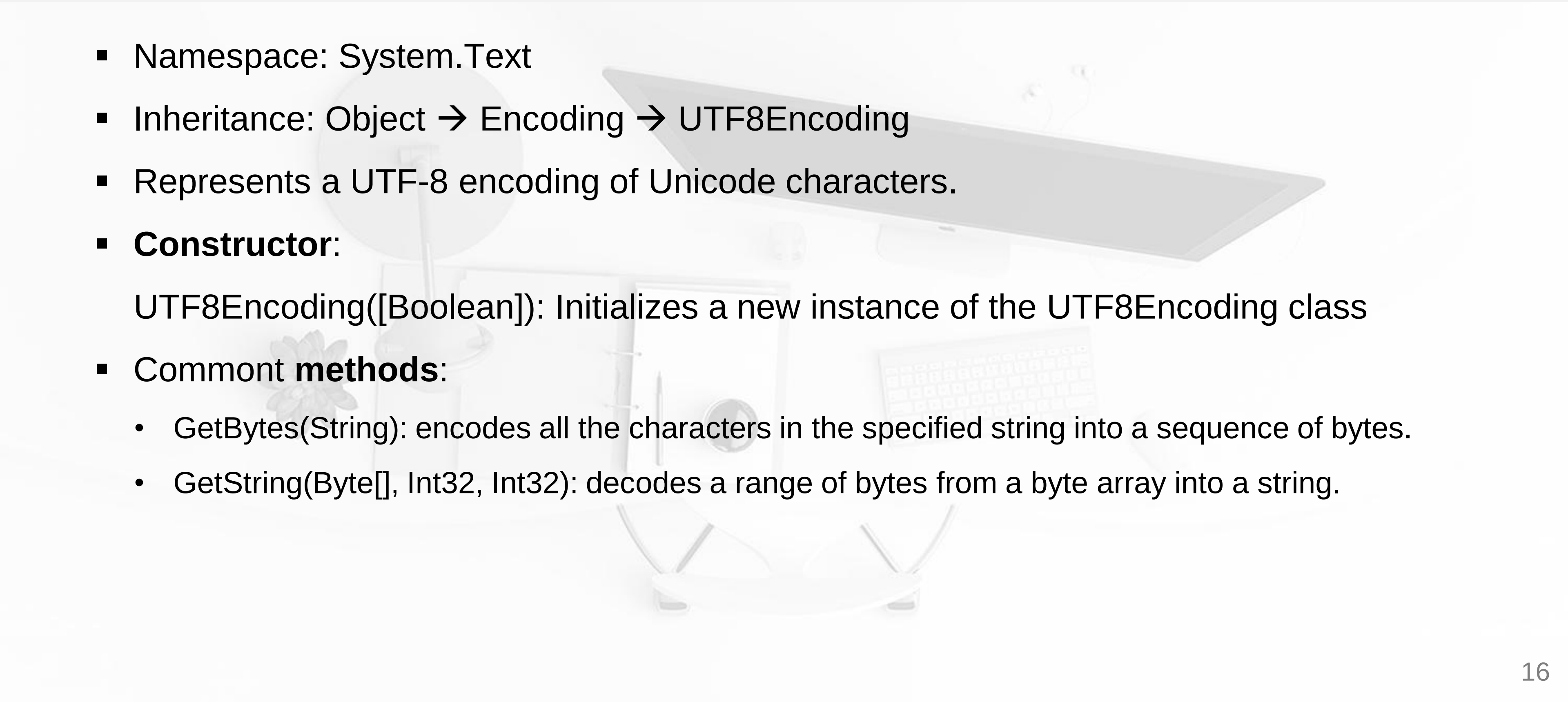
5. Encoding / Decoding data



- **Encoding** is the process of transforming a set of characters into a sequence of bytes
- **Decoding** is the process of transforming a sequence of encoded bytes into a set of characters

- .NET provides the following implementations
  - **ASCIIEncoding** encodes Unicode characters as single 7-bit ASCII characters
  - **UTF8Encoding** encodes Unicode characters using the UTF-8 encoding
  - **UnicodeEncoding** encodes Unicode characters using the UTF-16 encoding
  - **UTF32Encoding** encodes Unicode characters using the UTF-32 encoding

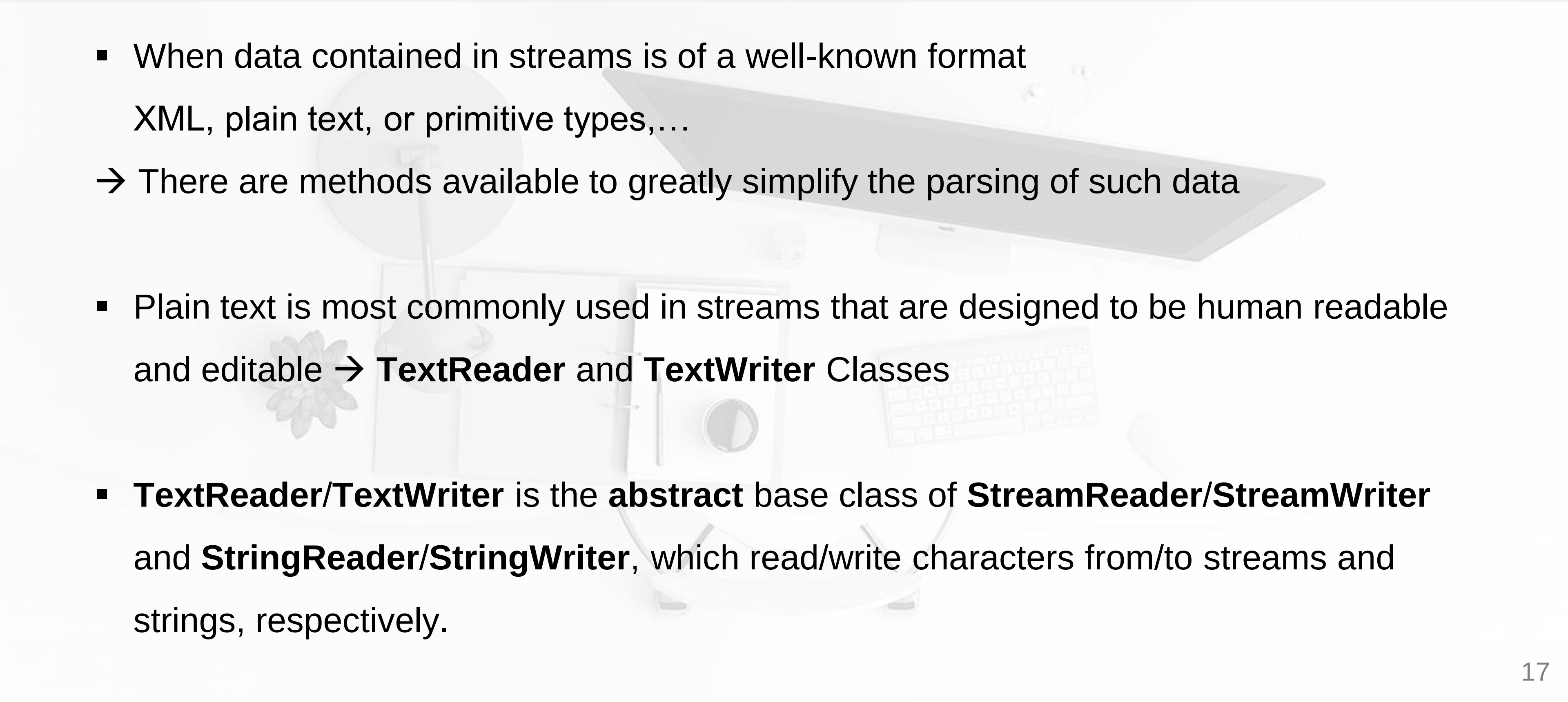
6. UTF8Encoding Class



- Namespace: System.Text
- Inheritance: Object → Encoding → UTF8Encoding
- Represents a UTF-8 encoding of Unicode characters.
- **Constructor**:

  UTF8Encoding([Boolean]): Initializes a new instance of the UTF8Encoding class
- Commont **methods**:
  - GetBytes(String): encodes all the characters in the specified string into a sequence of bytes.
  - GetString(Byte[], Int32, Int32): decodes a range of bytes from a byte array into a string.

7. Binary and Text Stream



- When data contained in streams is of a well-known format
  XML, plain text, or primitive types,…

→ There are methods available to greatly simplify the parsing of such data

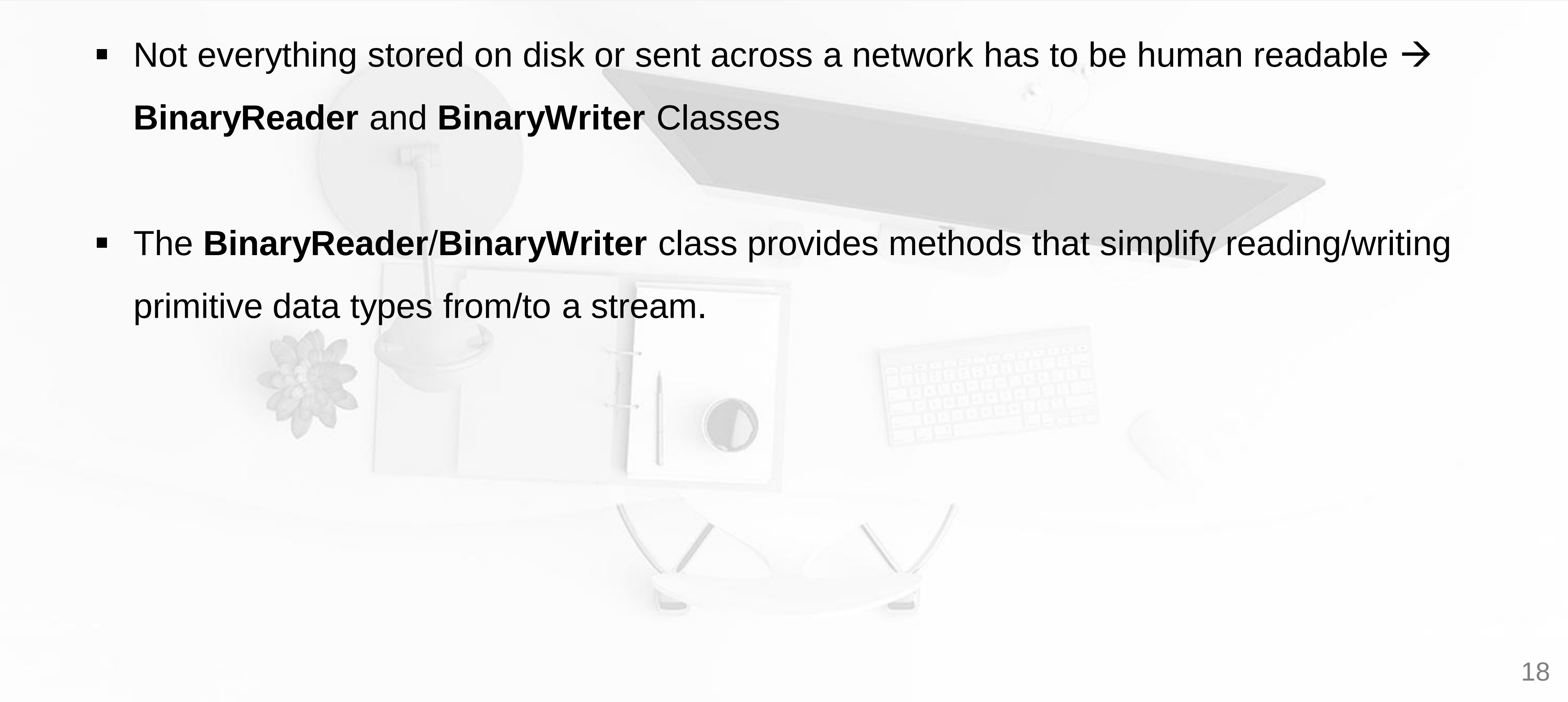
- Plain text is most commonly used in streams that are designed to be human readable and editable → **TextReader** and **TextWriter** Classes

- **TextReader**/**TextWriter** is the **abstract** base class of **StreamReader**/**StreamWriter** and **StringReader**/**StringWriter**, which read/write characters from/to streams and strings, respectively.

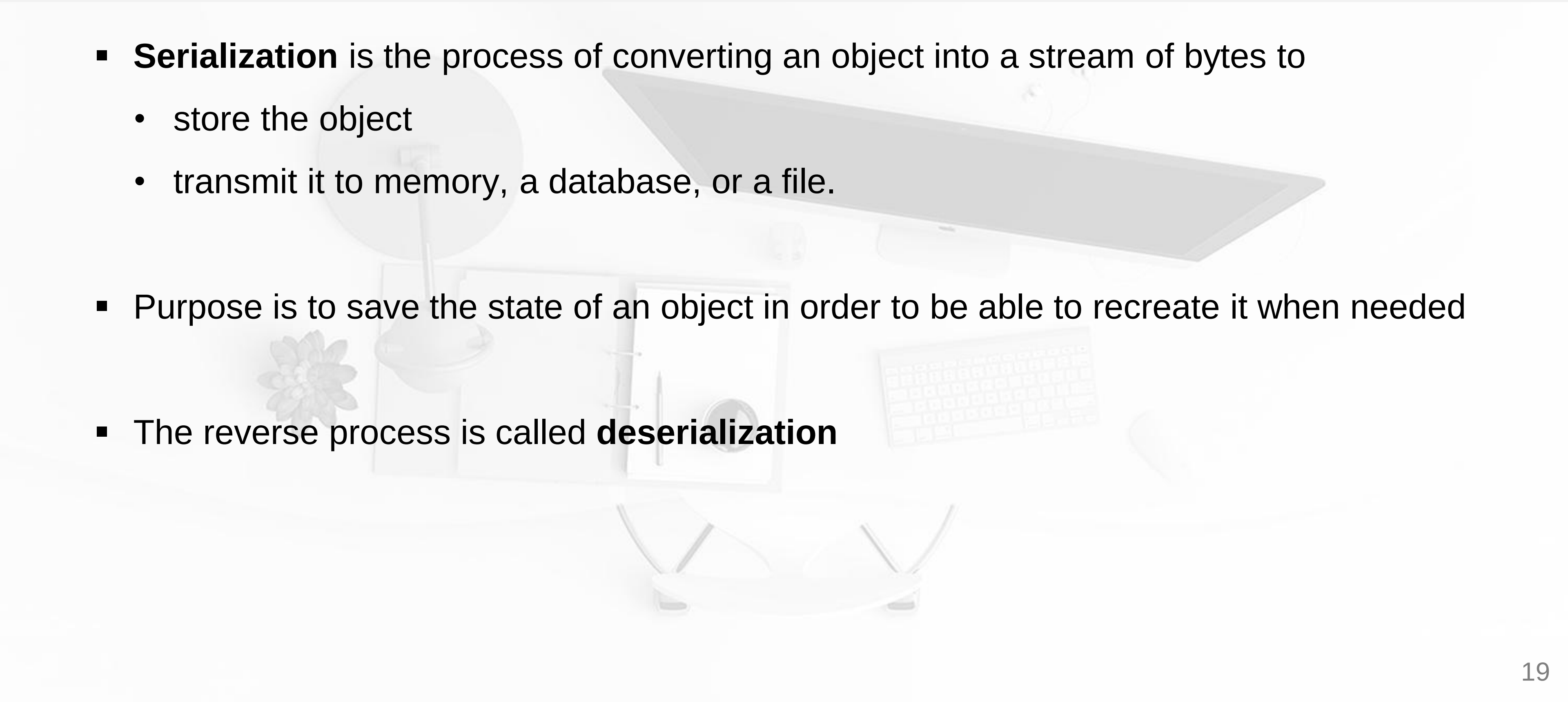
7. Binary and Text Stream

- Not everything stored on disk or sent across a network has to be human readable → **BinaryReader** and **BinaryWriter** Classes

- The **BinaryReader**/**BinaryWriter** class provides methods that simplify reading/writing primitive data types from/to a stream.



8. Serialization



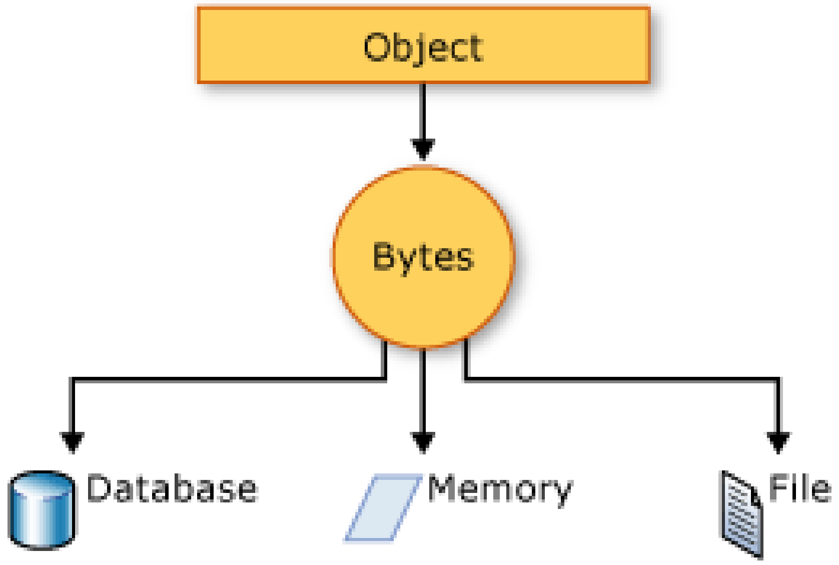
- **Serialization** is the process of converting an object into a stream of bytes to
  - store the object
  - transmit it to memory, a database, or a file.

- Purpose is to save the state of an object in order to be able to recreate it when needed

- The reverse process is called **deserialization**

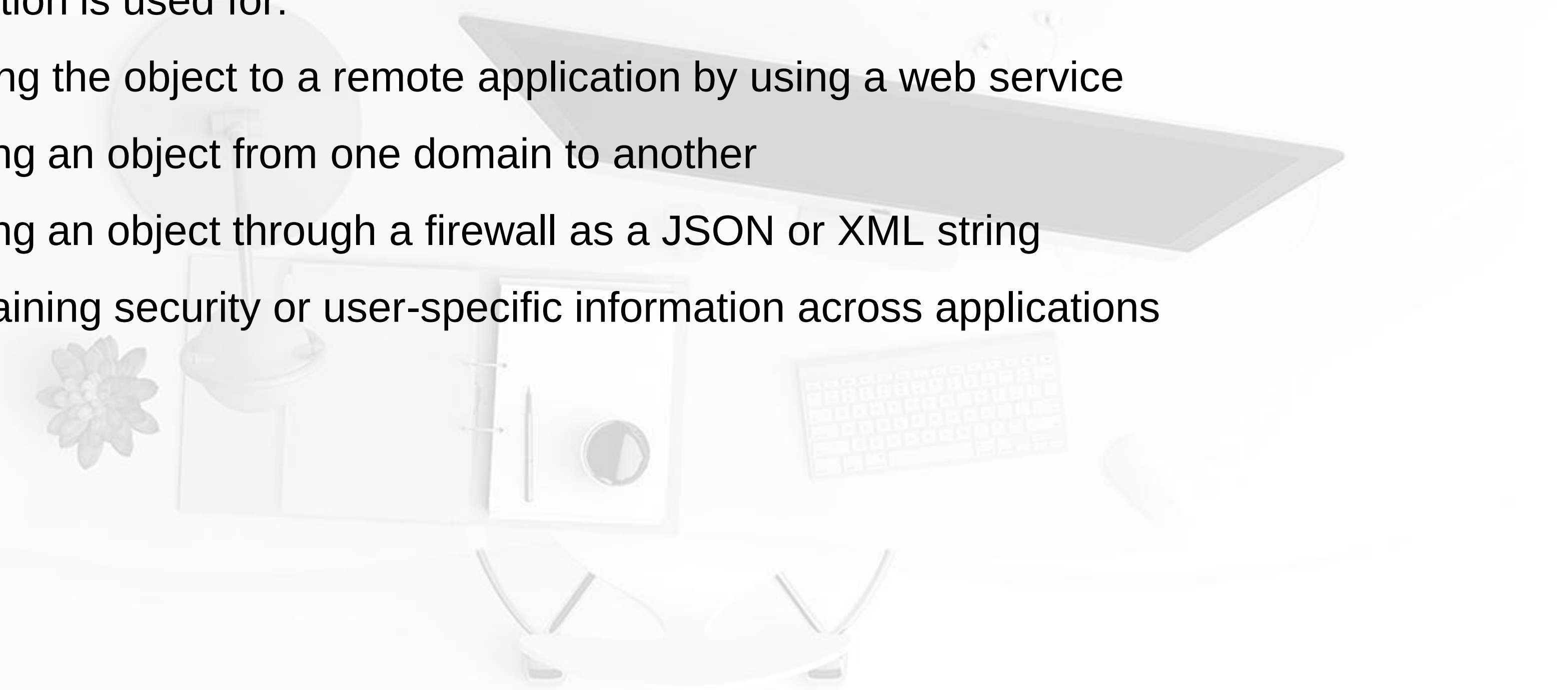
8. Serialization

How serialization works?



The overall process

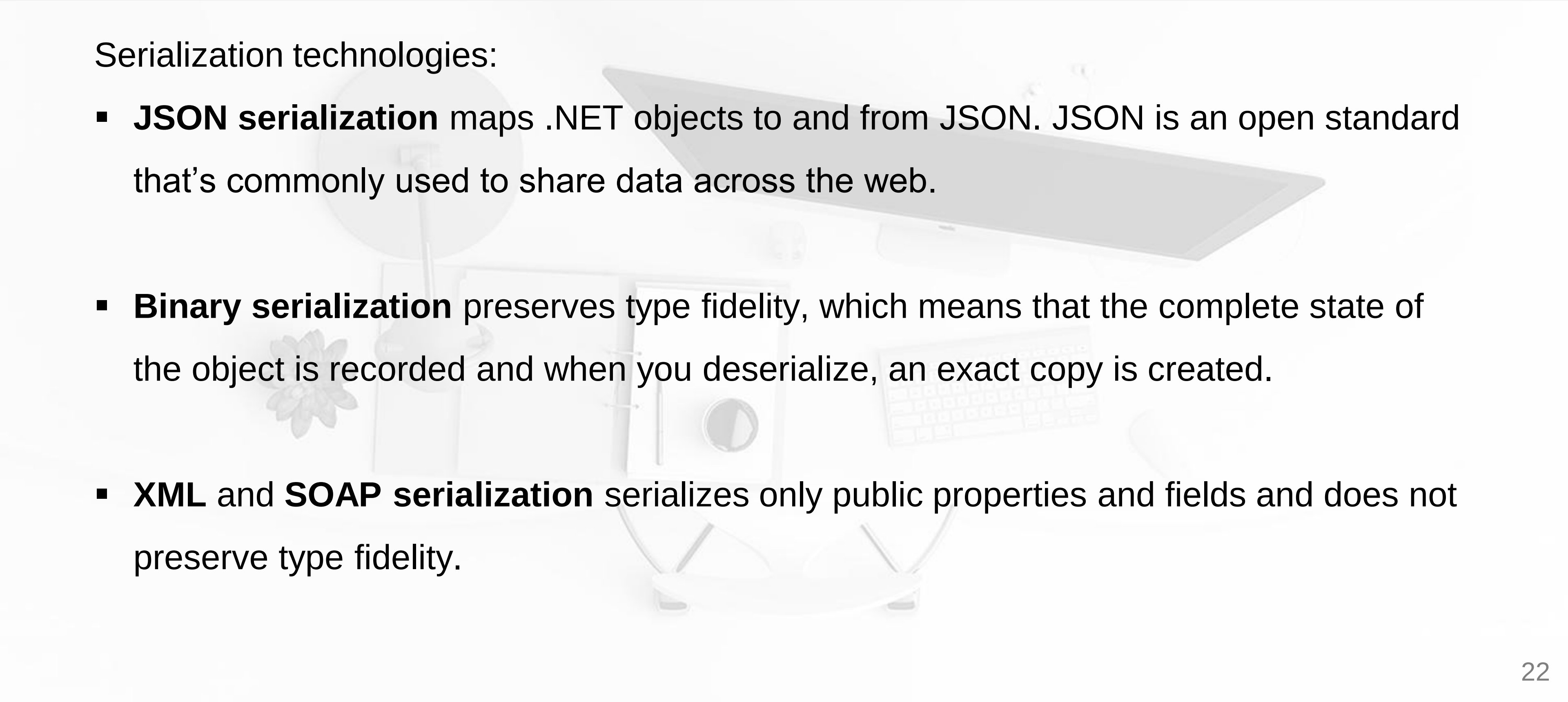
8. Serialization



Serialization is used for:

- Sending the object to a remote application by using a web service
- Passing an object from one domain to another
- Passing an object through a firewall as a JSON or XML string
- Maintaining security or user-specific information across applications

8. Serialization



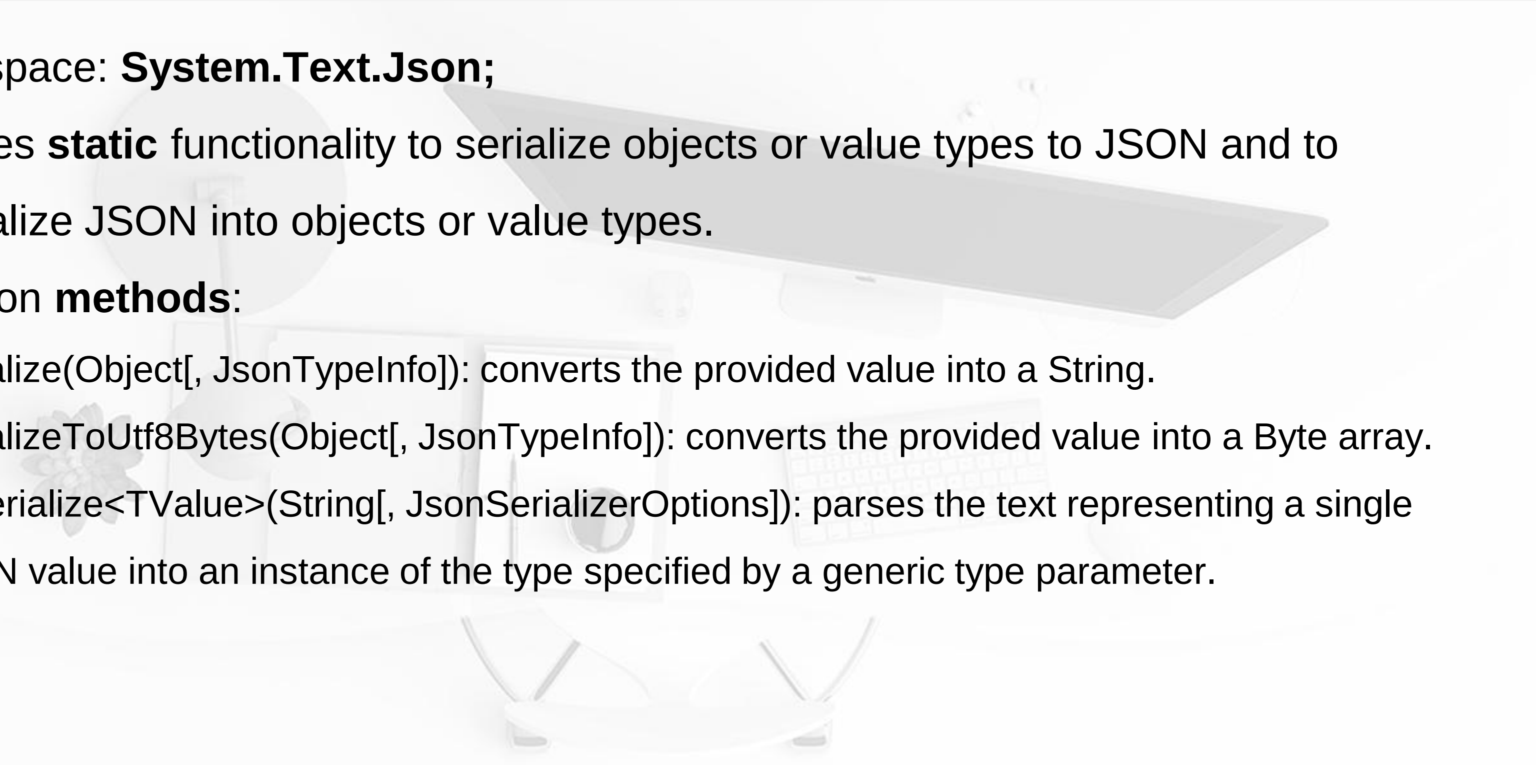
Serialization technologies:

- **JSON serialization** maps .NET objects to and from JSON. JSON is an open standard that's commonly used to share data across the web.

- **Binary serialization** preserves type fidelity, which means that the complete state of the object is recorded and when you deserialize, an exact copy is created.

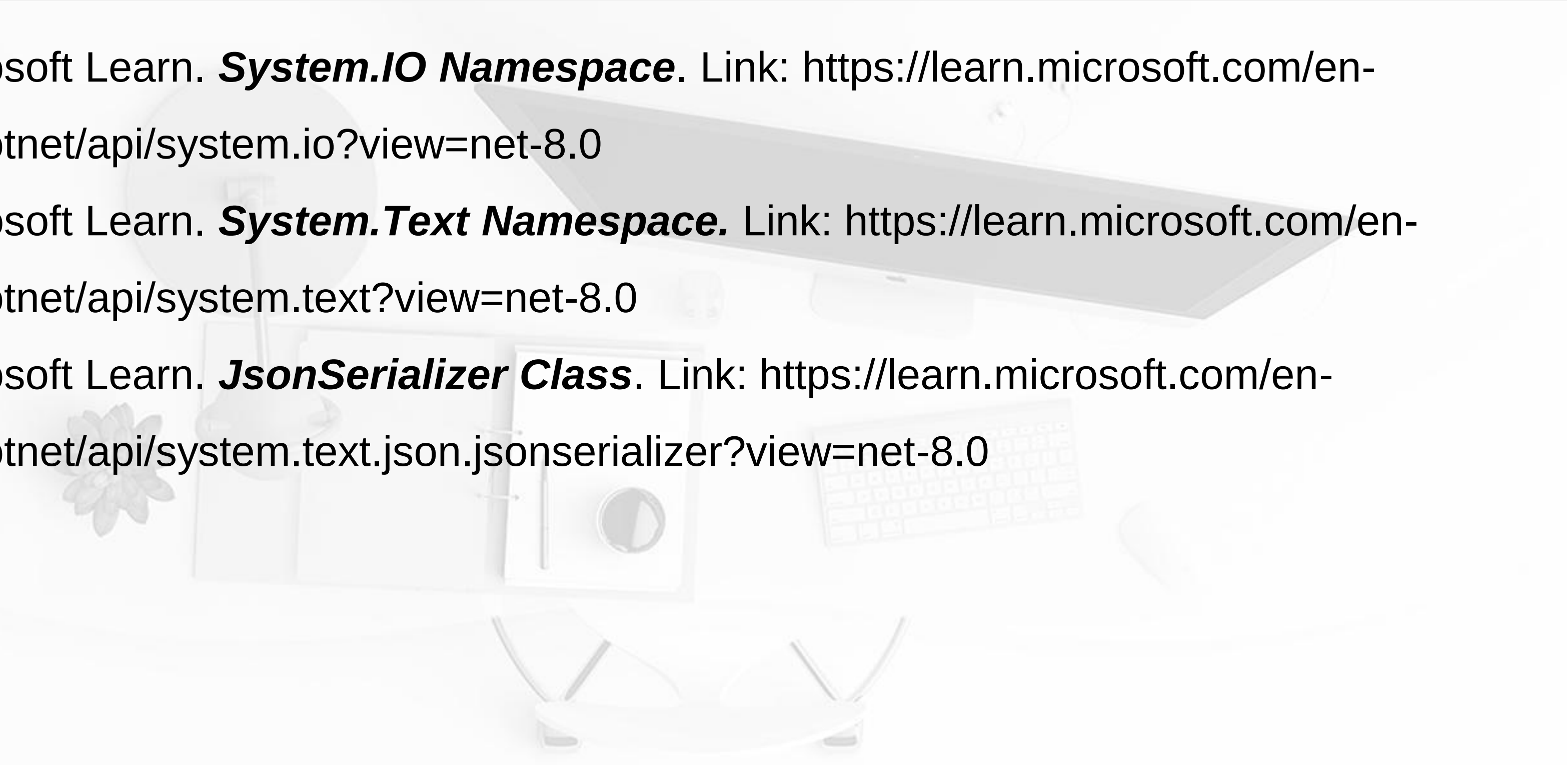
- **XML** and **SOAP serialization** serializes only public properties and fields and does not preserve type fidelity.

9. JsonSerializer Class



- Namespace: **System.Text.Json;**
- Provides **static** functionality to serialize objects or value types to JSON and to deserialize JSON into objects or value types.
- Common **methods**:
  - Serialize(Object[, JsonTypeInfo]): converts the provided value into a String.
  - SerializeToUtf8Bytes(Object[, JsonTypeInfo]): converts the provided value into a Byte array.
  - Deserialize<TValue>(String[, JsonSerializerOptions]): parses the text representing a single JSON value into an instance of the type specified by a generic type parameter.

Reference



- Microsoft Learn. ***System.IO Namespace***. Link: https://learn.microsoft.com/en-us/dotnet/api/system.io?view=net-8.0
- Microsoft Learn. ***System.Text Namespace.*** Link: https://learn.microsoft.com/en-us/dotnet/api/system.text?view=net-8.0
- Microsoft Learn. ***JsonSerializer Class***. Link: https://learn.microsoft.com/en-us/dotnet/api/system.text.json.jsonserializer?view=net-8.0

Q&A