Using **`IDisposable`** Type Objects

The common language runtime's garbage collector (GC) reclaims the memory used by managed objects. Typically, types that use unmanaged resources implement the `IDisposable` or `IAsyncDisposable` interface to allow the unmanaged resources to be reclaimed. When you finish using an object that implements `IDisposable`, you call the object's `Dispose` or `DisposeAsync` implementation to explicitly perform cleanup. You can do this in one of two ways:

- With the C# `using` statement or declaration (Using in Visual Basic).
- By implementing a `try/finally` block, and calling the `Dispose` or `DisposeAsync` method in the finally.

Objects that implement `System.IDisposable` or `System.IAsyncDisposable` should always be properly disposed of, regardless of variable scoping, unless otherwise explicitly stated. Types that define a finalizer to release unmanaged resources usually call `GC.SuppressFinalize` from either their `Dispose` or `DisposeAsync` implementation. Calling `SuppressFinalize` indicates to the GC that the finalizer has already been run and the object shouldn't be promoted for finalization.

The **`using`** statement

The `using` statement in C# and the `Using` statement in Visual Basic simplify the code that you must write to cleanup an object. The `using` statement obtains one or more resources, executes the statements that you specify, and automatically disposes of the object. However, the `using` statement is useful only for objects that are used within the scope of the method in which they are constructed.

The following example uses the using statement to create and release a `System.IO.StreamReader` object.

Example:

```
using System.IO;
class UsingStatement
{
    static void Main()
    {
        var buffer = new char[50];
        using (StreamReader streamReader = new("file1.txt"))
        {
            int charsRead = 0;
            while (streamReader.Peek() != -1)
            {
                charsRead = streamReader.Read(buffer, 0, buffer.Length);
                //
                // Process characters read.
                //
            }
        }
    }
}
```

With C# 8, a `using` declaration is an alternative syntax available where the braces are removed, and scoping is implicit.

Example:

```
using System.IO;
class UsingDeclaration
{
    static void Main()
    {
        var buffer = new char[50];
        using StreamReader streamReader = new("file1.txt");

        int charsRead = 0;
        while (streamReader.Peek() != -1)
        {
            charsRead = streamReader.Read(buffer, 0, buffer.Length);
            //
            // Process characters read.
            //
        }
    }
}
```

`Try/finally` block

Instead of wrapping a `try/finally` block in a `using` statement, you may choose to implement the `try/finally` block directly. It may be your personal coding style, or you might want to do this for one of the following reasons:

- To include a `catch` block to handle exceptions thrown in the try block. Otherwise, any exceptions thrown within the `using` statement are unhandled.
- To instantiate an object that implements `IDisposable` whose scope is not local to the block within which it is declared.

Example:

```
using System;
using System.Globalization;
using System.IO;
class TryExplicitCatchFinally
{
    static void Main()
    {
        StreamReader? streamReader = null;
        try
        {
            streamReader = new StreamReader("file1.txt");
            string contents = streamReader.ReadToEnd();
            var info = new StringInfo(contents);
            Console.WriteLine($"The file has {info.LengthInTextElements} text 
elements.");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("The file cannot be found.");
        }
        catch (IOException)
        {
            Console.WriteLine("An I/O error has occurred.");
        }
        catch (OutOfMemoryException)
        {
            Console.WriteLine("There is insufficient memory to read the file.");
        }
        finally
        {
            streamReader?.Dispose();
        }
    }
}
```