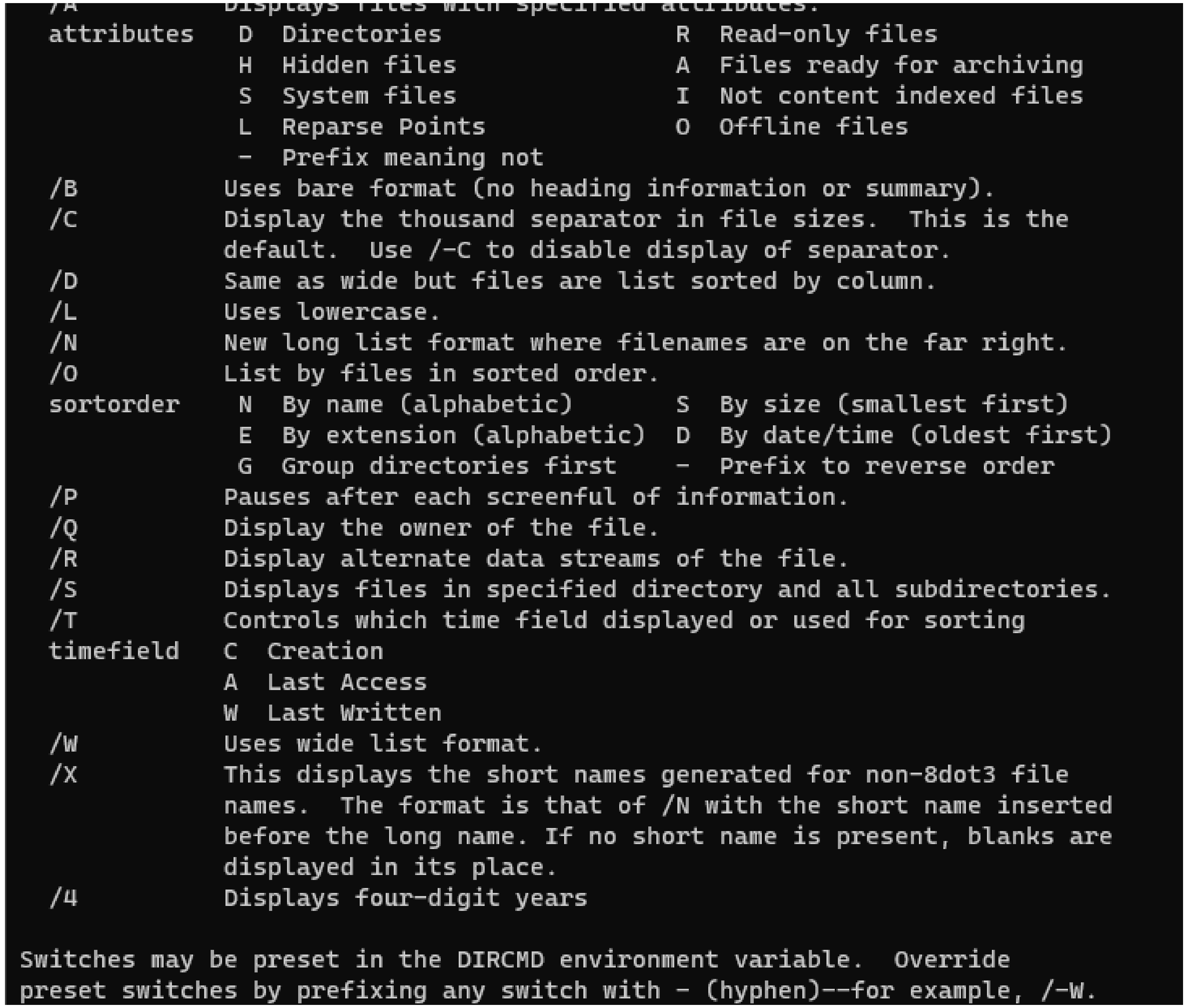
CLASSES AND LIBRARIES



```
  /A          Displays files with specified attributes.
  attributes   D  Directories                R  Read-only files
               H  Hidden files               A  Files ready for archiving
               S  System files               I  Not content indexed files
               L  Reparse Points             O  Offline files
               -  Prefix meaning not
  /B          Uses bare format (no heading information or summary).
  /C          Display the thousand separator in file sizes.  This is the
              default.  Use /-C to disable display of separator.
  /D          Same as wide but files are list sorted by column.
  /L          Uses lowercase.
  /N          New long list format where filenames are on the far right.
  /O          List by files in sorted order.
  sortorder    N  By name (alphabetic)       S  By size (smallest first)
               E  By extension (alphabetic)  D  By date/time (oldest first)
               G  Group directories first    -  Prefix to reverse order
  /P          Pauses after each screenful of information.
  /Q          Display the owner of the file.
  /R          Display alternate data streams of the file.
  /S          Displays files in specified directory and all subdirectories.
  /T          Controls which time field displayed or used for sorting
  timefield   C  Creation
              A  Last Access
              W  Last Written
  /W          Uses wide list format.
  /X          This displays the short names generated for non-8dot3 file
              names.  The format is that of /N with the short name inserted
              before the long name. If no short name is present, blanks are
              displayed in its place.
  /4          Displays four-digit years

Switches may be preset in the DIRCMD environment variable.  Override
preset switches by prefixing any switch with - (hyphen)--for example, /-W.
```

LET'S REVIEW DIR

Hypothetical Functions

- GetFilesInDir()
- GetSubDirsInDir()
- GetFileLastModifiedDate()
- GetFileCreateDate()
- GetFileOwner()
- GetFileSize()
- GetFileAttributes()
- GetCommandLineArguments()
- GetCurrentWorkingDirectory()
- IsFile()
- IsDirectory()

Pseudo-code

- Parse command line arguments
  - If none use the current working directory
  - Check if it is a directory
  - If not exit and tell user
- Get all files from the directory
- Get all directories from the directory
- For each file and directory:
  - output date modified, size, and name if file
  - output date modified, and name if a directory

What to put in a Shared Project

- Things that could be easily reused
- Not specific to a particular program
- Notice that console programs have much similarity
- Command line parsing
- Standard input reading
- File reading

```
public static class Utils
{
    public static DirectoryInfo GetCurrentWorkingDirectory()...

    public static string GetExePath()...

    public static string GetExeName()...

    public static bool HasOption(IEnumerable<string> args, string optionName)...

    public static bool HasOption(IEnumerable<string> args, params string[] optionNames)

    public static string GetOptionValue(IEnumerable<string> args, string optionName, st

    public static void WriteLine(string text, ConsoleColor color)...

    public static void Write(string text, ConsoleColor color)...

    public static long FileSize(FileSystemInfo fileOrDir)...

    public static IEnumerable<FileSystemInfo> FilterFilesBySize(IEnumerable<FileSystemI

    public static TimeSpan FileAge(FileSystemInfo file)...

    public static IEnumerable<FileSystemInfo> FilterFilesOrDirsByAge(IEnumerable<FileSy

    public static IEnumerable<FileInfo> GetFiles(DirectoryInfo dir, string searchPatter

    public static IEnumerable<DirectoryInfo> GetDirs(DirectoryInfo dir, string searchPa

    public static IEnumerable<FileSystemInfo> GetFilesAndDirs(DirectoryInfo dir, string
}
```

How to decide what is shared

- Imagine writing another similar application, would the function or class be useful?
- Would I be able to reuse it as-is, or would I have to change it if I reused it?
- Is there some hidden dependency on code in the main program?

Good Practice

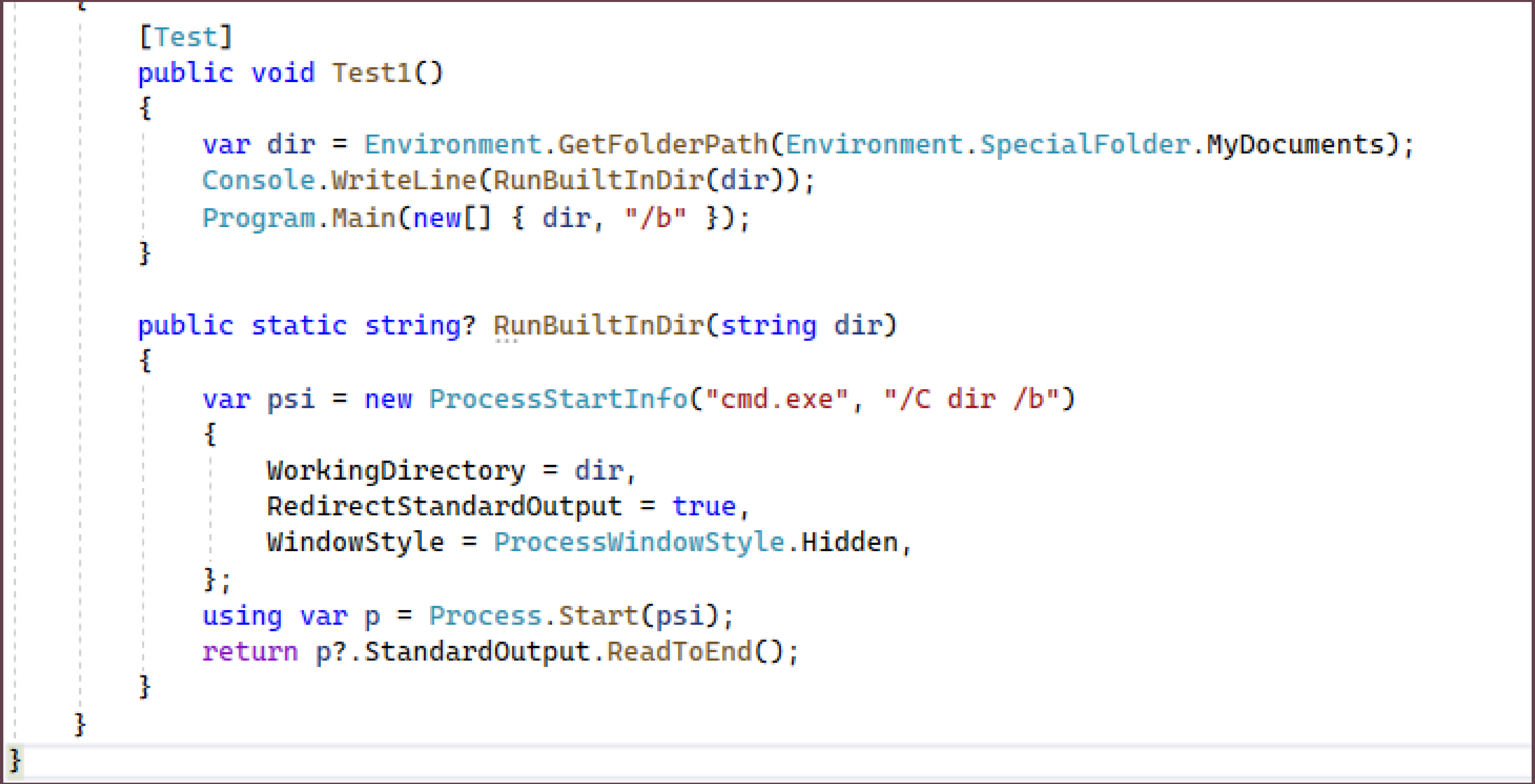
- Makes you think about the division of logic in your program
- This is what an AI will take a long time to get good at
- Helps you identify and reduce code coupling
- Overall reduces hidden complexity and decrease duplication

Code Coupling

- “One of the earliest indicators of design quality was coupling. There are several ways to describe coupling, but it boils down to this: If changing one module in a program requires changing another module, then coupling exists.”
- https://www.martinfowler.com/ieeeSoftware/coupling.pdf

What can we test?

- Does our find application work with a file?
- Does our find application work with standard input?
- Does our find algorithm find?
- Does our sort application sort?
- Does our sort application work with a file?
- Does our sort application work with standard input?
- Does our sort algorithm sort?
- Do our applications output help?

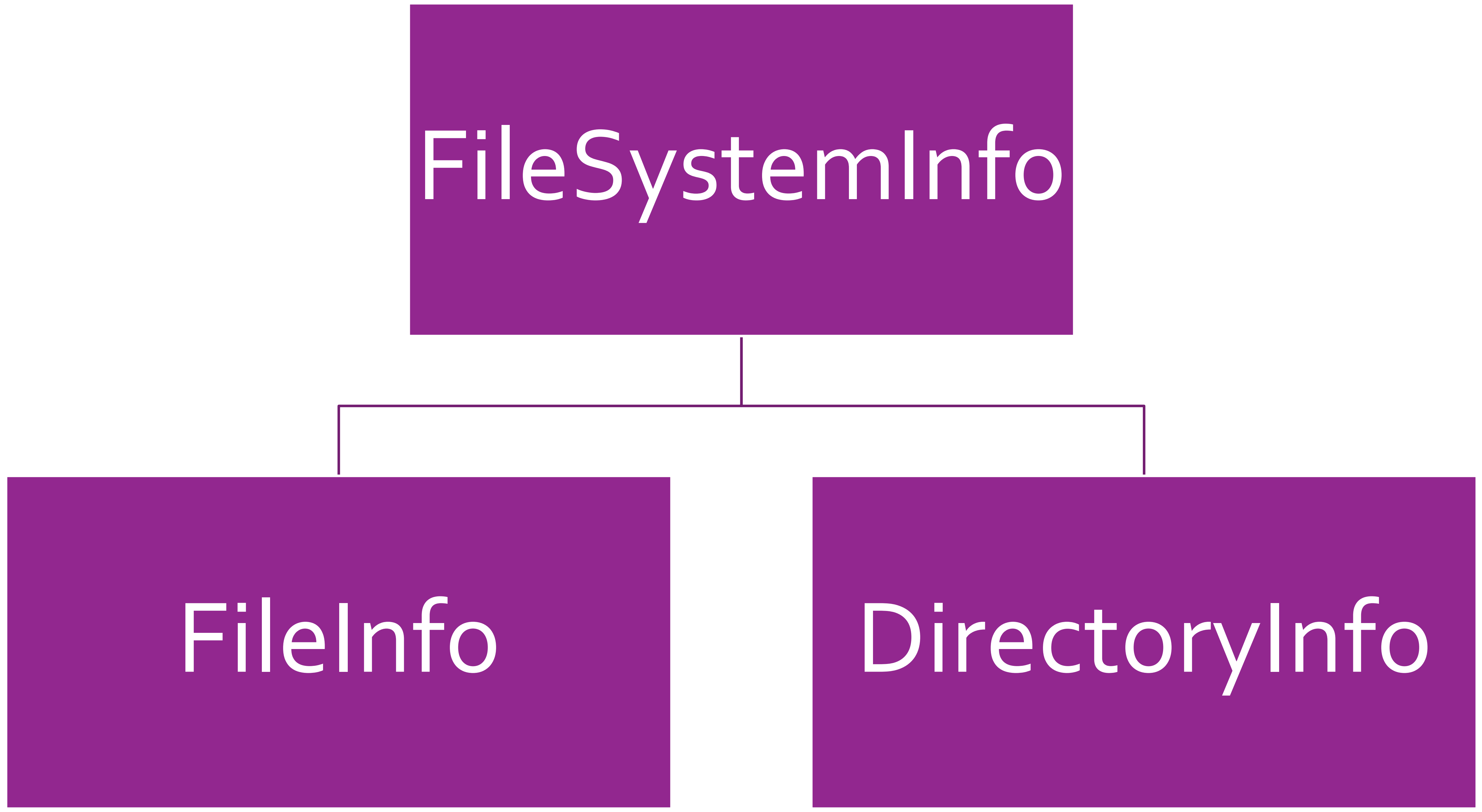


```
        [Test]
        public void Test1()
        {
            var dir = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            Console.WriteLine(RunBuiltInDir(dir));
            Program.Main(new[] { dir, "/b" });
        }

        public static string? RunBuiltInDir(string dir)
        {
            var psi = new ProcessStartInfo("cmd.exe", "/C dir /b")
            {
                WorkingDirectory = dir,
                RedirectStandardOutput = true,
                WindowStyle = ProcessWindowStyle.Hidden,
            };
            using var p = Process.Start(psi);
            return p?.StandardOutput.ReadToEnd();
        }
    }
}
```

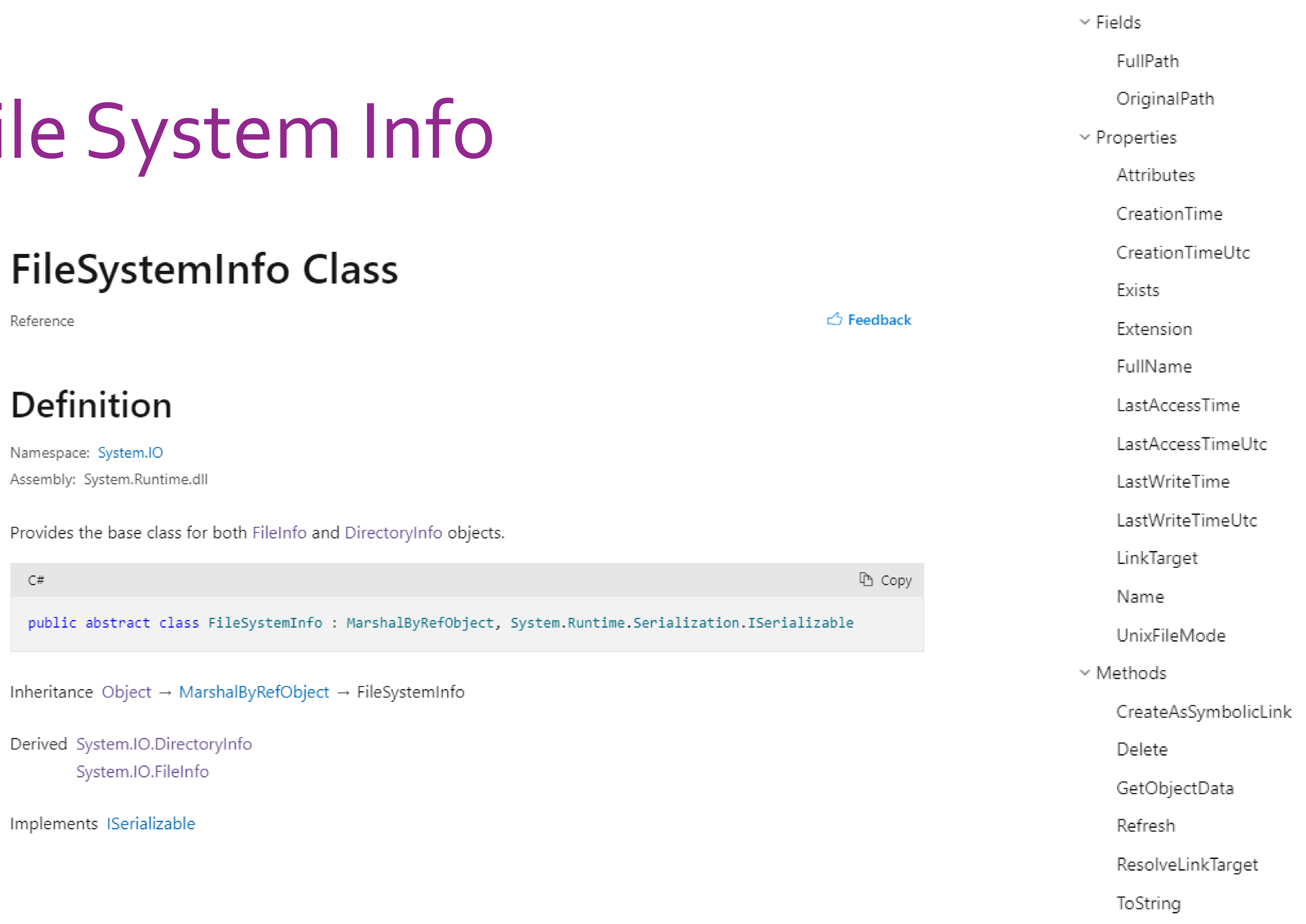
Representing Files and Directories

- We could use strings (for paths)
- However, how do we remember if looking at a file or directory
- One option is to use FileInfo and DirectoryInfo
- Both derive from FileSystemInfo
- A lot of the functionality we need is on these classes

Class Hierarchy



File System Info



FileSystemInfo Class

Reference

Feedback

Definition

Namespace: System.IO

Assembly: System.Runtime.dll

Provides the base class for both FileInfo and DirectoryInfo objects.

C# Copy

```csharp
public abstract class FileSystemInfo : MarshalByRefObject, System.Runtime.Serialization.ISerializable
```

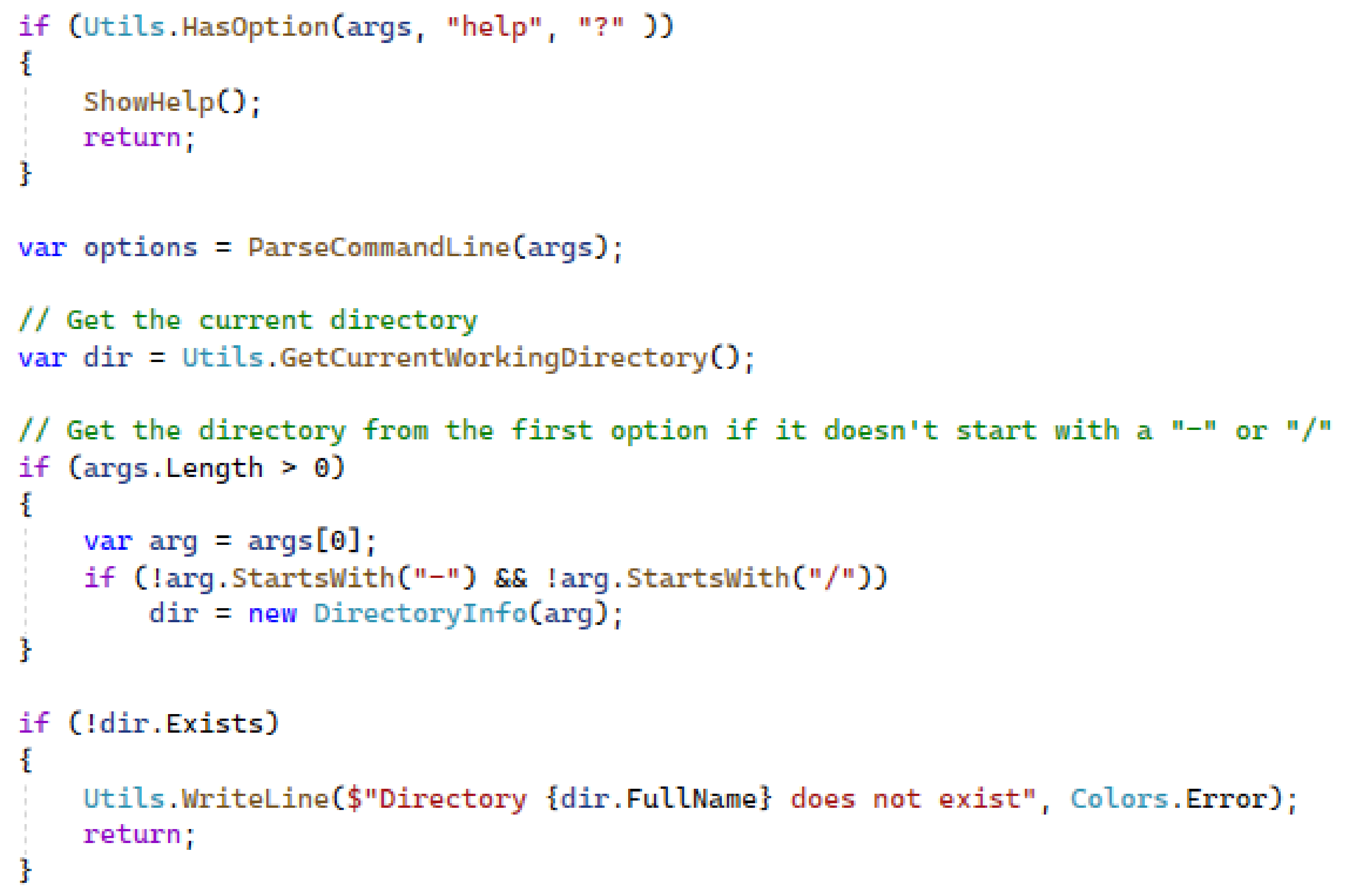
Inheritance Object → MarshalByRefObject → FileSystemInfo

Derived System.IO.DirectoryInfo
System.IO.FileInfo

Implements ISerializable

- Fields
  - FullPath
  - OriginalPath
- Properties
  - Attributes
  - CreationTime
  - CreationTimeUtc
  - Exists
  - Extension
  - FullName
  - LastAccessTime
  - LastAccessTimeUtc
  - LastWriteTime
  - LastWriteTimeUtc
  - LinkTarget
  - Name
  - UnixFileMode
- Methods
  - CreateAsSymbolicLink
  - Delete
  - GetObjectData
  - Refresh
  - ResolveLinkTarget
  - ToString

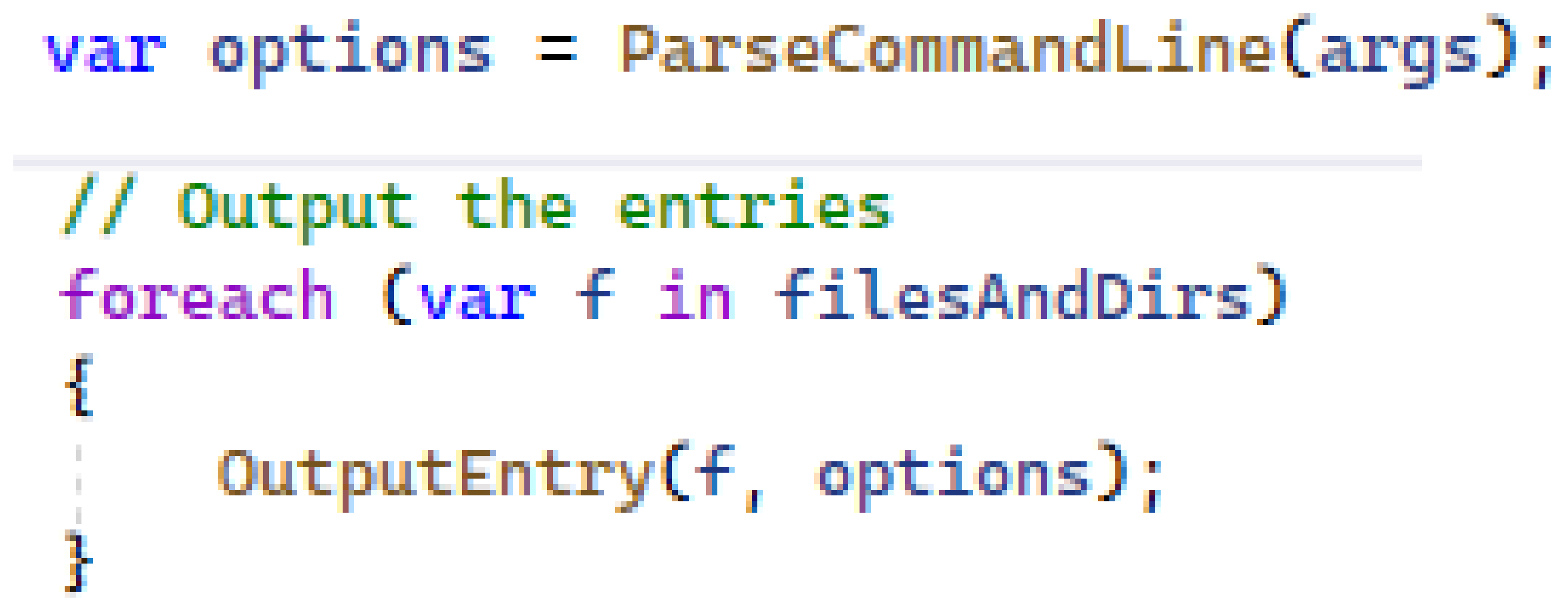
Parse Command Lines Arguments



```
if (Utils.HasOption(args, "help", "?" ))
{
    ShowHelp();
    return;
}

var options = ParseCommandLine(args);

// Get the current directory
var dir = Utils.GetCurrentWorkingDirectory();

// Get the directory from the first option if it doesn't start with a "-" or "/"
if (args.Length > 0)
{
    var arg = args[0];
    if (!arg.StartsWith("-") && !arg.StartsWith("/"))
        dir = new DirectoryInfo(arg);
}

if (!dir.Exists)
{
    Utils.WriteLine($"Directory {dir.FullName} does not exist", Colors.Error);
    return;
}
```

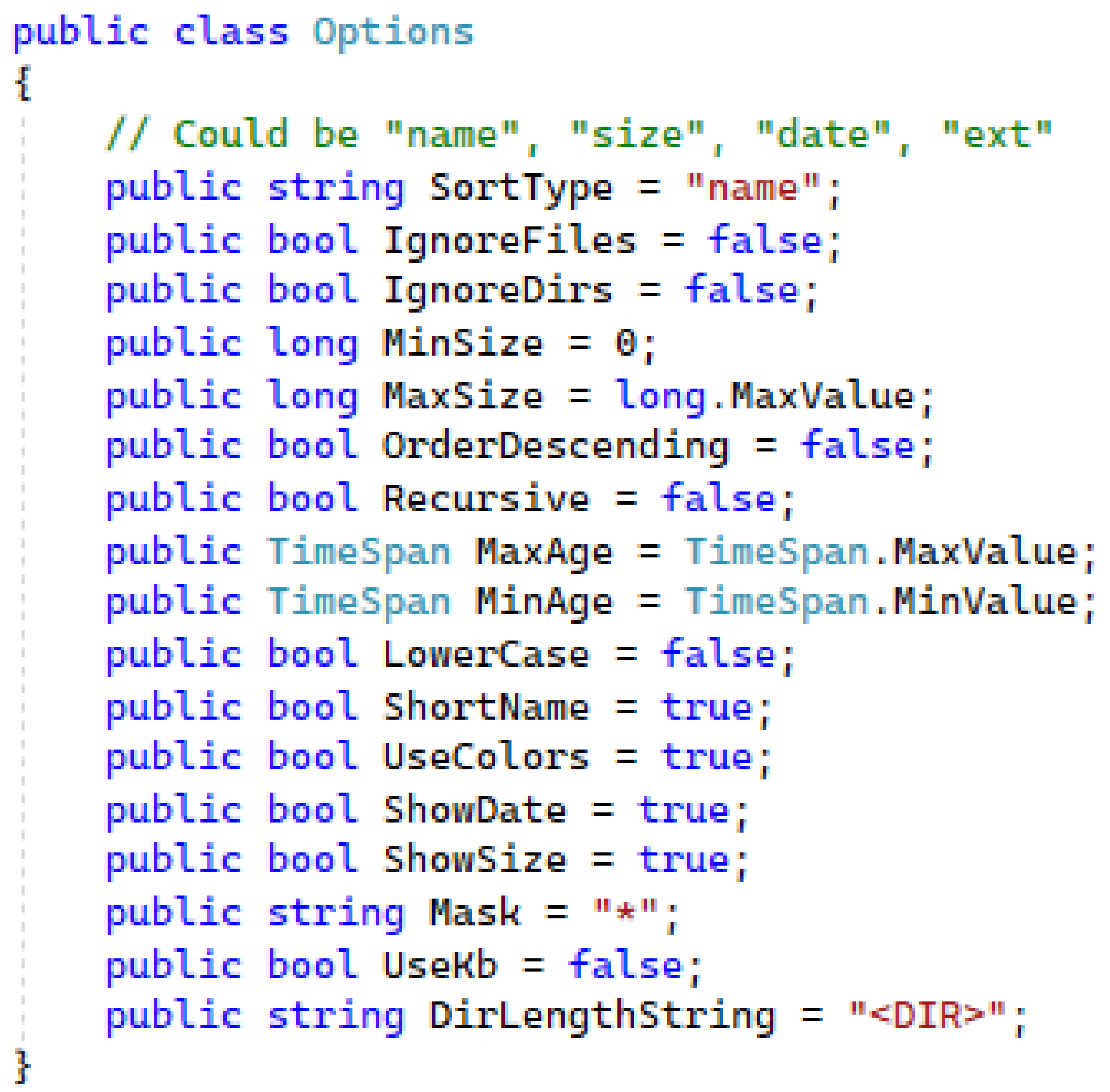
Classes are Useful for Grouping Data

- Consider the set of all options
- Rather than creating a variable for each option we can group it in one class
- We can then return it from a function and pass it as an argument



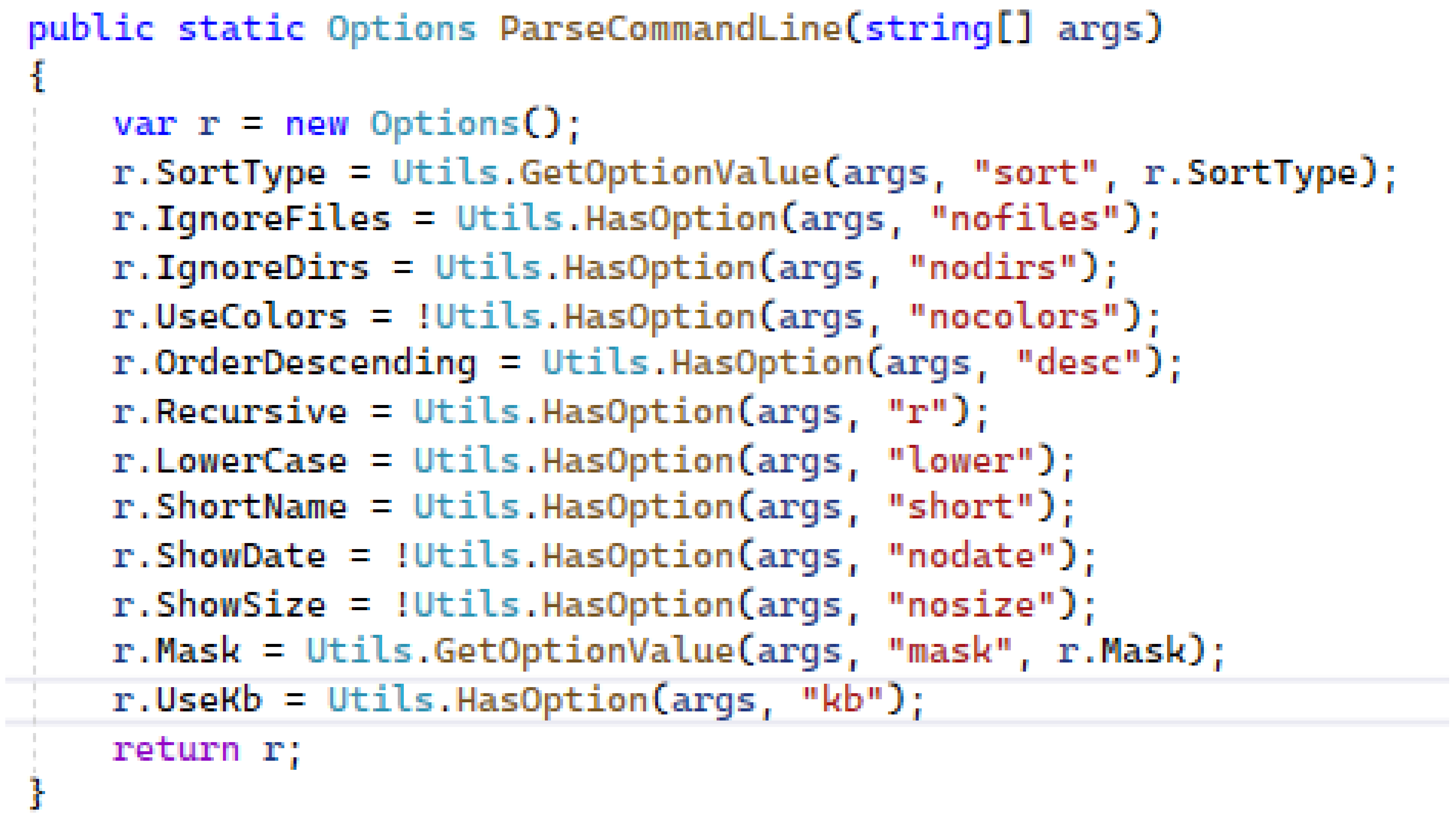
```
var options = ParseCommandLine(args);

// Output the entries
foreach (var f in filesAndDirs)
{
    OutputEntry(f, options);
}
```

Creating an options class



```
public class Options
{
    // Could be "name", "size", "date", "ext"
    public string SortType = "name";
    public bool IgnoreFiles = false;
    public bool IgnoreDirs = false;
    public long MinSize = 0;
    public long MaxSize = long.MaxValue;
    public bool OrderDescending = false;
    public bool Recursive = false;
    public TimeSpan MaxAge = TimeSpan.MaxValue;
    public TimeSpan MinAge = TimeSpan.MinValue;
    public bool LowerCase = false;
    public bool ShortName = true;
    public bool UseColors = true;
    public bool ShowDate = true;
    public bool ShowSize = true;
    public string Mask = "*";
    public bool UseKb = false;
    public string DirLengthString = "<DIR>";
}
```

Creating an Instance of the Options Class



```
public static Options ParseCommandLine(string[] args)
{
    var r = new Options();
    r.SortType = Utils.GetOptionValue(args, "sort", r.SortType);
    r.IgnoreFiles = Utils.HasOption(args, "nofiles");
    r.IgnoreDirs = Utils.HasOption(args, "nodirs");
    r.UseColors = !Utils.HasOption(args, "nocolors");
    r.OrderDescending = Utils.HasOption(args, "desc");
    r.Recursive = Utils.HasOption(args, "r");
    r.LowerCase = Utils.HasOption(args, "lower");
    r.ShortName = Utils.HasOption(args, "short");
    r.ShowDate = !Utils.HasOption(args, "nodate");
    r.ShowSize = !Utils.HasOption(args, "nosize");
    r.Mask = Utils.GetOptionValue(args, "mask", r.Mask);
    r.UseKb = Utils.HasOption(args, "kb");
    return r;
}
```