Revision Summary:

My first Comment was about the mistakes in UML Diagram of the program, I realized that relationships between classes were not correct and most of them were messy, So I fixed them with proper diagram and resubmitted it. The Second Comment was that the Name property was defined in the thing class and also repeated in the children classes which is against the best practices in OOP so I Fixed them and Presented a neat and clean UML diagram. The comments about the code mentioned that Name was being repeated and needed to be abstracted so I fixed that and Removed all the repetitions of Name. Also , The filesystem class contained an incorrect constructor so I included a list of _contents to fix it, The final comment mentioned that Name property was missing in the parent class so I included that and got the program running, I also figured out why the program was running without the name property in the thing class because I had added a redundant property called name to every class which is again against the best practices of OOP.

Revision Summary For PortFolio:

I acknowleged all the feedbacks and edited the code and uml accordingly

```
1
2  namespace test
3  {
4      public class file1: Thing
5      {
6
7          private string _extension;
8          private int _size;
9
10
11         public file1 (string name , string extension, int size) : base
             (name)
12         {
13
14             this._extension = extension;
15             this._size = size;
16         }
17         public override int Size()
18         {
19             return _size;
20         }
21
22         public override void Print()
23         {
24             Console.WriteLine($"File '{Name}{_extension} ' Size: {_size}
                 bytes");
25
26         }
27
28
29
30
31
32     }
33 }
34
```

```
 1
 2
 3  namespace test
 4  {
 5      public class FileSystem
 6      {
 7
 8          private List<Thing> _contents;
 9          public FileSystem()
10          {
11              _contents = new List<Thing> ();
12          }
13
14
15
16          public void Add(Thing thing)
17          {
18              _contents.Add(thing);
19          }
20
21          public void PrintContents()
22          {
23              Console.WriteLine($"This File System contains: ");
24              foreach(var item in  _contents)
25              {
26                  item.Print();
27              }
28          }
29      }
30  }
31
```

```
1
2
3
4
5  namespace test
6  {
7      public class Folder : Thing
8      {
9          List<Thing> _contents = new List<Thing>();
10
11
12         public Folder(string name) : base(name)
13         {
14             _contents = new List<Thing>();
15         }
16
17
18
19
20
21         public void Add(Thing thing)
22         {
23             _contents.Add(thing);
24         }
25
26         public override int Size()
27         {
28             int size = 0;
29             foreach(var item in _contents)
30             {
31                 size += item.Size();
32             }
33             return size;
34         }
35
36         public override void Print()
37         {
38             if (_contents.Count == 0)
39             {
40                 Console.WriteLine($"The Folder: {Name} is empty!");
41             }
42             else
43             {
44                 Console.WriteLine($"The Folder: '{Name}' Contains
                       {_contents.Count} files totalling {Size()} bytes");
45
46
47                 foreach(var item in  _contents)
48                 {
```

```
49                      item.Print();
50                  }
51              }
52
53
54          }
55
56
57      }
58  }
59
```

```
1  namespace test
2  {
3      internal class Program
4      {
5          static void Main(string[] args)
6          {
7              FileSystem filesystem = new FileSystem();
8
9              for (int i = 0; i < 5; i++)
10             {
11                 filesystem.Add(new file1($"104762184-{i}",".txt",100 +
                       i*10));
12             }
13
14             Folder folder = new Folder("Folder WITH 3 files");
15             for (int i = 0; i < 3; i++)
16             {
17                 folder.Add(new file1($"104762184-{i}", ".txt", 150 + i *
                       10));
18             }
19             filesystem.Add(folder);
20
21
22             Folder folder1 = new Folder("Folder with subfolder");
23             Folder subfolder = new Folder("subfolder with 23 files");
24             for (int i = 0;i<23;i++)
25             {
26                 subfolder.Add(new file1($"104762184-{i}", ".txt", 200 + i *
                       10));
27             }
28
29             folder1.Add(subfolder);
30             filesystem.Add(folder1);
31
32
33             for (int i = 0; i < 11; i++)
34             {
35                 filesystem.Add(new Folder($"'Empty Folder {i}'"));
36             }
37             filesystem.PrintContents();
38         }
39     }
40 }
41
```

```
 1
 2
 3  namespace test
 4  {
 5      public abstract class Thing
 6      {
 7          private string _name;
 8
 9          public string Name { get { return _name; } }
10
11
12          public Thing(string name)
13          {
14              _name = name;
15          }
16
17          public abstract int Size();
18          public abstract void Print();
19
20      }
21  }
22
```

<<abstract>>
Thing

- _name : string

+ Name : string <<read-only property
+ thing (string name)
+ Size () : int <<abstract>>
+ Print () <<abstract>>

Composition

Inheritance

File System

- _ Contents : List<<thing>>

+ FileSystem ()
+ Add (Thing thing)
+ Print Contents ()

Folder

- _ Contents : List<<thing>>

+ Folder (string name)
+ Add ( Thing thing)
+ Size : Int
+ Print()

File1

- _ extension : string
- _ size : int

+ File1(string name, string extension, int size)
~~+ Size()~~
+ size () : int
+ Paint ()

**Explain the concept of polymorphism and how it was applied in Task 1:**

- **Polymorphism allows objects from different classes to be treated as instances of a common superclass. This concept makes it possible to use one interface for a wide range of actions, with the specific action being determined by the object performing it.**
- **In Task 1, polymorphism is applied through the Thing abstract class, which defines abstract methods like Size() and Print(). The File and Folder classes implement these methods with their own specific logic. As a result, the FileSystem class can invoke Size() or Print() on any Thing object (whether it's a File or Folder) without needing to know the exact type of the object.**

**Are both the FileSystem and Folder classes necessary?:**

- **Not necessarily. The Folder class already functions as a container for both files and folders, which is similar to the role of a file system. Thus, the FileSystem class could be seen as redundant since its purpose could be fulfilled by a top-level Folder. In this design, the entire file system could simply be represented as a root folder, making a separate FileSystem class unnecessary.**

**What's wrong with the class name Thing?:**

- **The name Thing is overly vague and doesn't provide any meaningful insight into the purpose of the class. A more appropriate name would be something like FileSystemItem, FileComponent, or FileEntity, as these names better reflect the role of the class in representing files and folders within the file system.**

**Define the concept of abstraction and describe how it can be applied to designing the FileSystem and Folder classes:**

- **Abstraction refers to the practice of concealing complex details while exposing only the essential features of an object or function. In the context of designing the FileSystem and Folder classes, abstraction is applied by using a common interface (Thing in this case) to hide the differences between files and folders. This approach allows users to interact with the file system uniformly, without needing to be aware of whether an item is a file or a folder, as both are abstracted through the Thing interface.**

Microsoft Visual Studio Debu

```
This File System contains:
File '104762184-0.txt ' Size: 100) bytes
File '104762184-1.txt ' Size: 110) bytes
File '104762184-2.txt ' Size: 120) bytes
File '104762184-3.txt ' Size: 130) bytes
File '104762184-4.txt ' Size: 140) bytes
The Folder: 'Folder WITH 3 files' Contains 3 files totalling 480 bytes
File '104762184-0.txt ' Size: 150) bytes
File '104762184-1.txt ' Size: 160) bytes
File '104762184-2.txt ' Size: 170) bytes
The Folder: 'Folder with subfolder' Contains 1 files totalling 7130 bytes
The Folder: 'subfolder with 23 files' Contains 23 files totalling 7130 bytes
File '104762184-0.txt ' Size: 200) bytes
File '104762184-1.txt ' Size: 210) bytes
File '104762184-2.txt ' Size: 220) bytes
File '104762184-3.txt ' Size: 230) bytes
File '104762184-4.txt ' Size: 240) bytes
File '104762184-5.txt ' Size: 250) bytes
File '104762184-6.txt ' Size: 260) bytes
File '104762184-7.txt ' Size: 270) bytes
File '104762184-8.txt ' Size: 280) bytes
File '104762184-9.txt ' Size: 290) bytes
File '104762184-10.txt ' Size: 300) bytes
File '104762184-11.txt ' Size: 310) bytes
File '104762184-12.txt ' Size: 320) bytes
File '104762184-13.txt ' Size: 330) bytes
File '104762184-14.txt ' Size: 340) bytes
File '104762184-15.txt ' Size: 350) bytes
File '104762184-16.txt ' Size: 360) bytes
File '104762184-17.txt ' Size: 370) bytes
File '104762184-18.txt ' Size: 380) bytes
File '104762184-19.txt ' Size: 390) bytes
File '104762184-20.txt ' Size: 400) bytes
File '104762184-21.txt ' Size: 410) bytes
File '104762184-22.txt ' Size: 420) bytes
The Folder: 'Empty Folder 0' is empty!
The Folder: 'Empty Folder 1' is empty!
The Folder: 'Empty Folder 2' is empty!
The Folder: 'Empty Folder 3' is empty!
The Folder: 'Empty Folder 4' is empty!
The Folder: 'Empty Folder 5' is empty!
The Folder: 'Empty Folder 6' is empty!
The Folder: 'Empty Folder 7' is empty!
The Folder: 'Empty Folder 8' is empty!
The Folder: 'Empty Folder 9' is empty!
The Folder: 'Empty Folder 10' is empty!
```