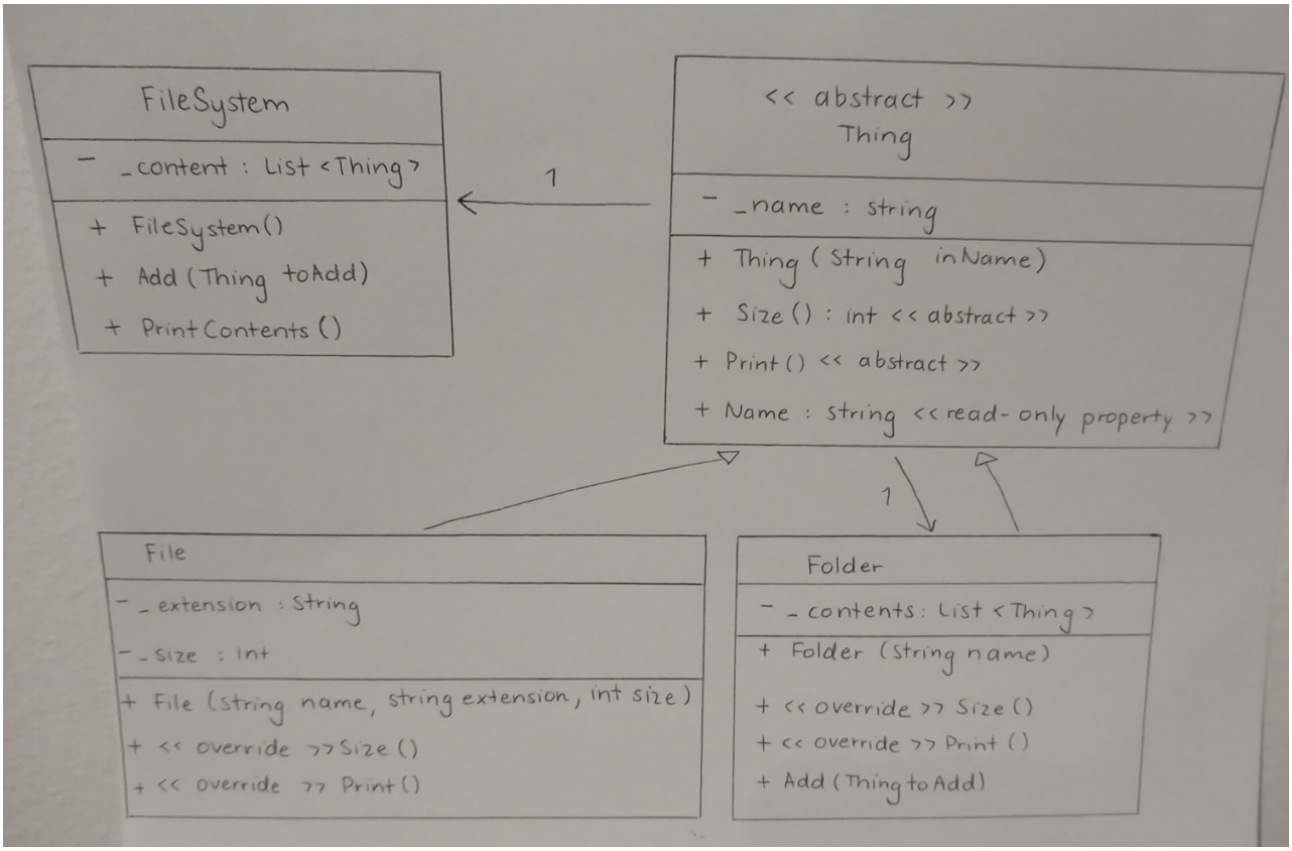
Swinburne University of Technology

COS20007 Object Oriented Programming

Semester test

PDF generated at 12:54 on Monday 16th October, 2023


FileSystem
- _content : List<Thing>
+ FileSystem()
+ Add(Thing toAdd)
+ PrintContents()
1
<< abstract >>
Thing
- _name : string
+ Thing(String inName)
+ Size() : int << abstract >>
+ Print() << abstract >>
+ Name : string << read-only property >>
1
File
- _extension : String
- _size : int
+ File(string name, string extension, int size)
+ << override >> Size()
+ << override >> Print()
Folder
- _contents : List<Thing>
+ Folder(String name)
+ << override >> Size()
+ << override >> Print()
+ Add(Thing toAdd)

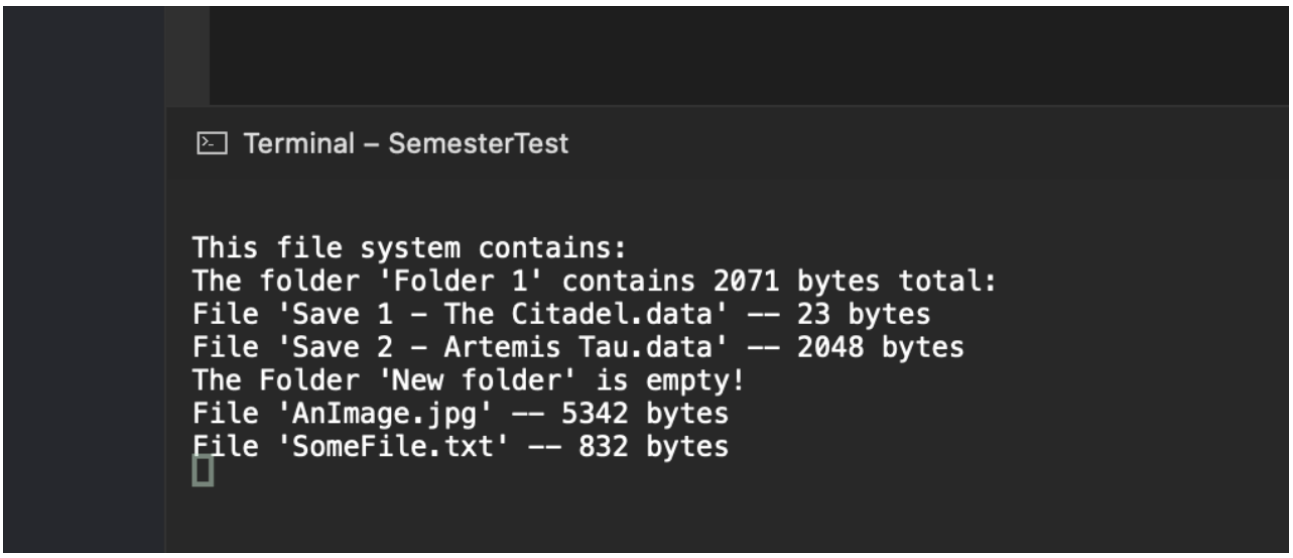
```
using System;
using System.Collections.Generic;

namespace SemesterTest
{
    class Program
    {
        static void Main(string[] args)
        {
            // Creating a FileSystem.
            FileSystem fileSystem = new FileSystem();

            // Adding a folder that contains files to the file system.
            Folder folder1 = new Folder("Folder 1");
            File file1 = new File("Save 1 - The Citadel", "data", 23);
            File file2 = new File("Save 2 - Artemis Tau", "data", 2048);
            folder1.Add(file1);
            folder1.Add(file2);
            fileSystem.Add(folder1);

            // Adding an empty folder to the file system.
            Folder emptyFolder = new Folder("New folder");
            fileSystem.Add(emptyFolder);

            // Adding files to the file system.
            File file3 = new File("AnImage", "jpg", 5342);
            File file4 = new File("SomeFile", "txt", 832);
            fileSystem.Add(file3);
            fileSystem.Add(file4);

            // Calling the PrintContents method.
            fileSystem.PrintContents();
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace SemesterTest
{
    public class FileSystem
    {
        private List<Thing> _contents;

        public FileSystem()
        {
            _contents = new List<Thing>();
        }

        public void Add(Thing toAdd)
        {
            _contents.Add(toAdd);
        }

        public void PrintContents()
        {
            Console.WriteLine("This file system contains:");
            foreach (Thing thing in _contents)
            {
                thing.Print();
            }
        }
    }

}

```

```
using System;
using System.Collections.Generic;

namespace SemesterTest
{
    public abstract class Thing
    {
        private string _name;

        public Thing(string inName)
        {
            _name = inName;
        }

        public string Name
        {
            get { return _name; }
        }

        public abstract int Size();
        public abstract void Print();

    }
}

```

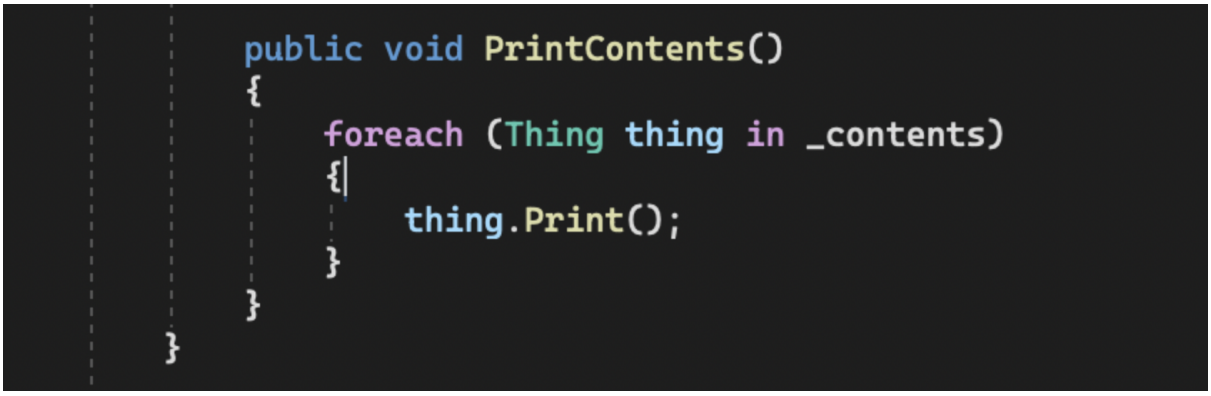
```
using System;
using System.Collections.Generic;

namespace SemesterTest
{
    public class Folder : Thing
    {
        private List<Thing> _contents;

        public Folder(string name) : base(name)
        {
            _contents = new List<Thing>();
        }

        public override int Size()
        {
            int totalSize = 0;
            foreach (Thing thing in _contents)
            {
                totalSize += thing.Size();
            }
            return totalSize;
        }

        public override void Print()
        {
            if (_contents.Count == 0)
            {
                Console.WriteLine($"The Folder '{Name}' is empty!");
            }
            else
            {
                Console.WriteLine($"The folder '{Name}' contains {Size()} bytes
↪  total:");
                foreach (Thing thing in _contents)
                {
                    thing.Print();
                }
            }
        }
        public void Add(Thing toAdd)
        {
            _contents.Add(toAdd);
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace SemesterTest
{
    public class File : Thing
    {
        private string _extension;
        private int _size;

        public File(string name, string extension, int size) : base(name)
        {
            _extension = extension;
            _size = size;
        }

        public override int Size()
        {
            return _size;
        }

        public override void Print()
        {
            Console.WriteLine($"File '{Name}.{_extension}' -- {_size} bytes");
        }
    }
}
```


Terminal – SemesterTest
This file system contains:
The folder 'Folder 1' contains 2071 bytes total:
File 'Save 1 - The Citadel.data' -- 23 bytes
File 'Save 2 - Artemis Tau.data' -- 2048 bytes
The Folder 'New folder' is empty!
File 'AnImage.jpg' -- 5342 bytes
File 'SomeFile.txt' -- 832 bytes

Task 2

Describe the principle of polymorphism and how it was used in Task 1.
Polymorphism is a concept that relates to a single object with many forms; one way to achieve this is using inheritance.This means the child class can also be handled as its parent type in the program, and programmers can use it in place where functions accept the parent type.
An example of Polymorphism is implemented in the PrintContents Method. In the FileSystem class, this method iterates through the _contents list, which contains objects of the Thing class. The thing.Print() method is called for each item in the list. Polymorphism comes into play as Thing can be an instance of both File or Folder.



Consider the FileSystem and Folder classes from the updated design in Task 1. Do we need both of these classes? Explain why or why not.

No, we don't need both of these classes because 'FileSystem' and 'Folder' share similar methods and functions. In my code, I generate a new object called 'fileSystem'. 'fileSystem' is an instance of the 'FileSystem' class, which serves as the root or entry point of the 'file system'. Folder objects like 'folder1' and 'emptyFolder' are added to the 'fileSystem'. Therefore, 'FileSystem' exists as a whole system that contains all the folders and files within it, while 'Folder' exists as a directory that contains all the individual sub folders and files. Although, in this context, the 'Folder' class duplicates the functionality of 'FileSystem' so Folder can replace FileSystem.

What is wrong with the class name Thing? Suggest a better name for the class, and explain the reasoning behind your answer.
'Thing' is a rather generic name, and does not provide much information about its intended purpose. A more suitable name would be to change it to 'ItemType' as it describes that instances of this class are types of items stored in the '_contents' lists, which can include both 'File' and 'Folder' types. It's generally better to give a descriptive name that can make the code more readable.

Define the principle of abstraction, and explain how you would use it to design a class to present a Book.

Abstraction conceals the complexity from the users and presents essential information. For instance, if you are using a TV remote control, it isn't necessary to understand the electronic components or the software running inside the TV. It is only needed to know how to use the remote to change channels, adjust the volume, etc. An abstract class is limited in that you cannot create objects directly from it. On the other hand, an Abstract method lacks implementation in the base class, but it can be inherited and overridden by classes derived from it. Now, In terms of a "Book" class, the base method can be named "Book", and one of the abstract methods can be "Content". The other two classes, say, "ScienceBook" and "TravelBook". In the case of the "ScienceBook", the overridden "Content" method can use string for providing scientific theories and experiments with explanations, and integers to represent numerical data. On the other hand, for the "TravelBook", the overridden method could rely on strings to convey information about travel destinations, and recommendations. This way, abstraction allows the creation of a general "Book" while enabling implementation for various book types like science and travel.