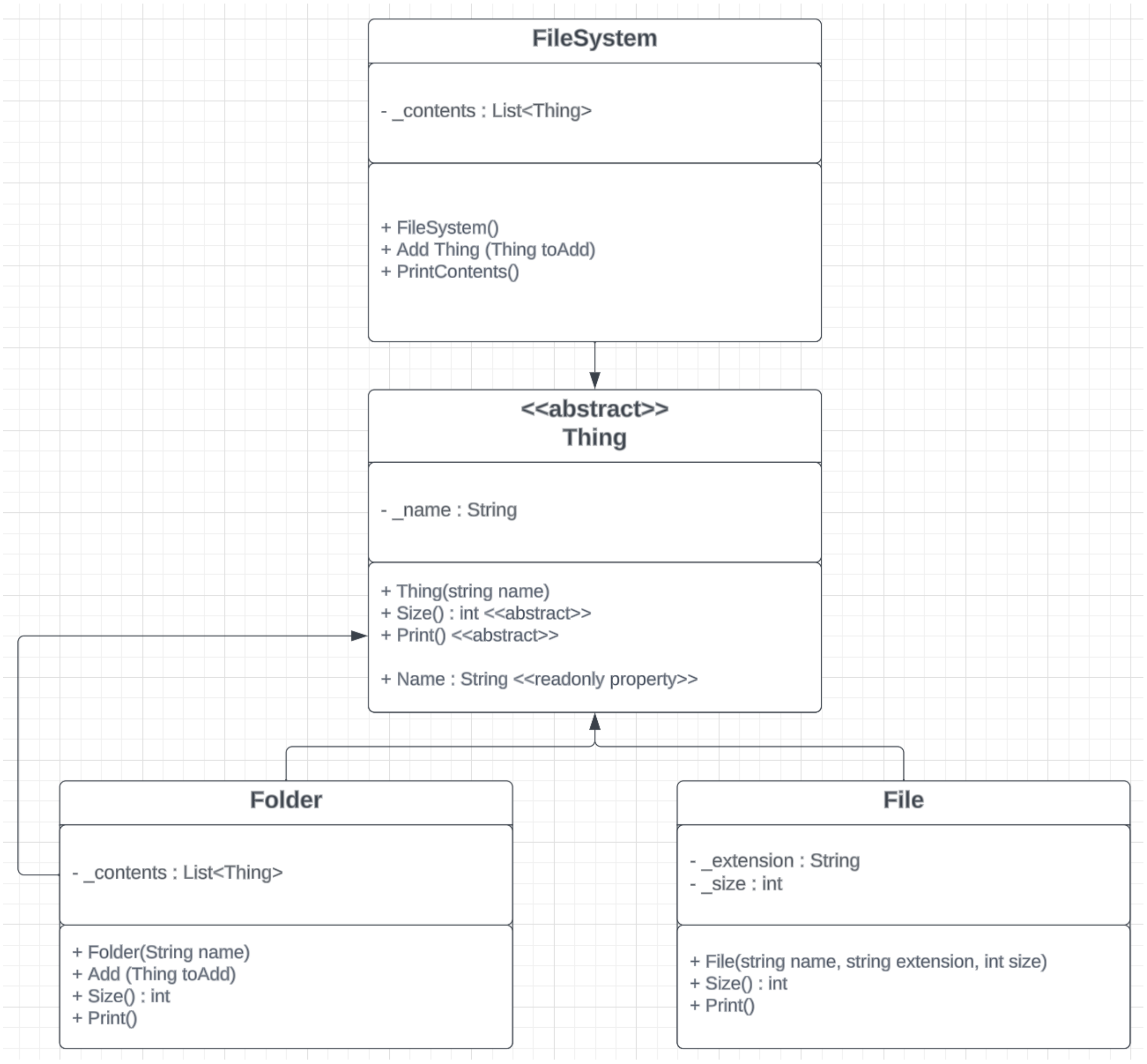
Swinburne University of Technology

COS20007 Object Oriented Programming

Semester test

PDF generated at 15:56 on Monday 16th October, 2023


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

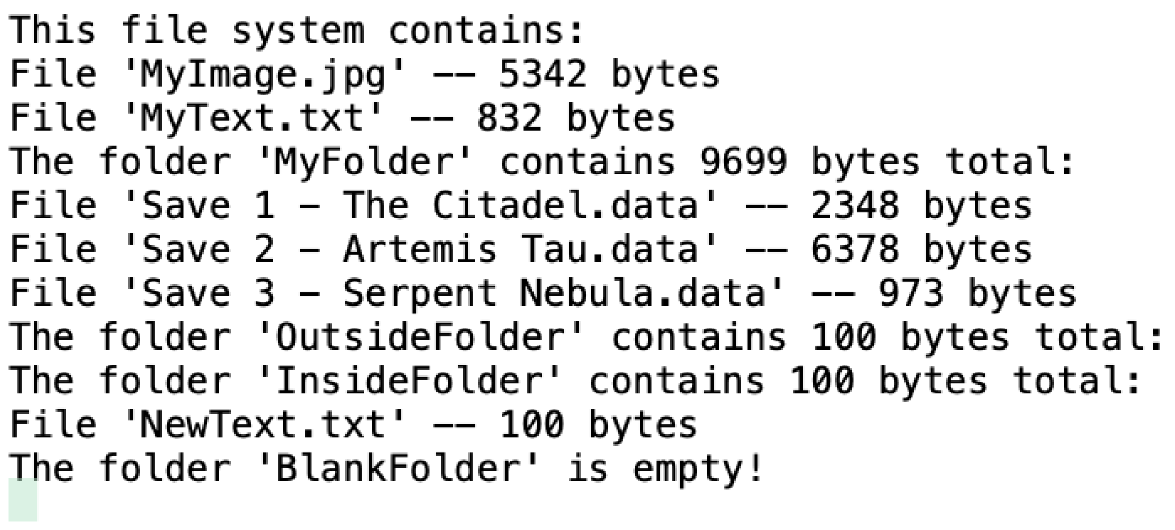
```
using System;
using System.Collections.Generic;

namespace HurdleTask
{
    class Program
    {
        static void Main(string[] args)
        {
            // File System
            FileSystem fileSystem = new FileSystem();

            // Files in File System (not within folder)
            fileSystem.Add(new File("MyImage", 5342, "jpg"));
            fileSystem.Add(new File("MyText", 832, "txt"));

            // Folder (contains files) in File System
            Folder fileFolder = new Folder("MyFolder");
            fileFolder.Add(new File("Save 1 - The Citadel", 2348, "data"));
            fileFolder.Add(new File("Save 2 - Artemis Tau", 6378, "data"));
            fileFolder.Add(new File("Save 3 - Serpent Nebula", 973, "data"));
            fileSystem.Add(fileFolder);

            // Folder within folder (contains files) in File System
            Folder outsideFolder = new Folder("OutsideFolder");
            Folder insideFolder = new Folder("InsideFolder");
            insideFolder.Add(new File("NewText", 100, "txt"));
            outsideFolder.Add(insideFolder);
            fileSystem.Add(outsideFolder);

            // Empty Folder (contains none) in File System
            Folder emptyFolder = new Folder("BlankFolder");
            fileSystem.Add(emptyFolder);

            //Print output
            fileSystem.PrintContents();
        }
    }
}
```

```
using System;
namespace HurdleTask
{
    class FileSystem
    {
        private List<Thing> _contents;

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

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

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

```

```
using System;
namespace HurdleTask
{
    abstract class Thing
    {
        protected string _name;

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

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

        public string Name
        {
            get
            {
                return _name;
            }
        }
    }
}

```

```csharp
using System;
namespace HurdleTask
{
    class Folder : Thing
    {
        private List<Thing> _contents;

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

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

        //Size
        public override int Size()
        {
            int size = 0;
            foreach (Thing item in _contents)
            {
                size += item.Size();
            }
            return size;
        }

        //Print structure
        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 item in _contents)
                {
                    item.Print();
                }
            }
        }
    }
}

```

```csharp
using System;
namespace HurdleTask
{
    class File : Thing
    {
        private int _size;
        private string _extension;

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

        //Size
        public override int Size()
        {
            return _size;
        }

        public string Extension
        {
            get { return _extension; }
        }


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

```



```
This file system contains:
File 'MyImage.jpg' -- 5342 bytes
File 'MyText.txt' -- 832 bytes
The folder 'MyFolder' contains 9699 bytes total:
File 'Save 1 - The Citadel.data' -- 2348 bytes
File 'Save 2 - Artemis Tau.data' -- 6378 bytes
File 'Save 3 - Serpent Nebula.data' -- 973 bytes
The folder 'OutsideFolder' contains 100 bytes total:
The folder 'InsideFolder' contains 100 bytes total:
File 'NewText.txt' -- 100 bytes
The folder 'BlankFolder' is empty!
```

COS20007 Object Oriented Programming
Hurdle Task 1: Semester Test

Dang Khoa Le
Student ID: 103844421
Major: Bachelor of Software Engineering

Abstract— **This document presenting Task 2 of the Semester Test, demonstrating the understanding of object-oriented programming and the core concepts of object-oriented design.**

I. Polymorphism:

Q: Describe the principle of polymorphism and how it was used in Task 1.

Polymorphism Principle: "Polymorphism comes from the Greek words 'polys' meaning much or many and 'morphe' (or morphism) meaning form or shape. Polymorphism is one of the core principles of Object-Oriented Programming (OOP). It is the concept of objects assuming various forms or, in OOP terms, different objects responding to the same message in distinct ways. It promotes code flexibility and extension.

Within this task, by introducing the Thing abstract class, both File and Folder classes are derived from it. This allows instances of file and folder to be treated as instances of Thing (class). A collection of Thing objects can be able to add both Files and Folders to it. Polymorphism principal enables code design that operates on the common attributes and methods of Thing within any specific derived types (whether it's a file or a folder).

II. FileSystem and Folder classes:

Q: 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.

Both of the FileSystem and Folder classes are compulsory and have to be implemented to satisfy Task 1 requirements.

The FileSystem class is the overall file system that works as a root directory (similar to a parent folder at high-level interfacing). It manages an entire file system with any folders and files to be stored within. The Folder class represents lower-directories or folders within the file system (similar to a child folder at low-level and individual interfacing). The two classes can be distincted that FileSystem manages the entire structure as a root directory, while Folder class has individual folders and their contents (folders or files) within the file system.

III Thing class:

Q: What is wrong with the class name Thing? Suggest a better name for the class, and explain the reasoning behind your answer.

The name (Thing) doesn't necessarily wrong, however, they don't generally have any meaning related to the topic nor could be able to represent or describe any concept. In OOP, it is also better to use class-name that can be descriptive and informative for the concept design, which support developers' cognition towards the code base, in terms of the readability, understandability, locatability and documentation.

Some such names that can be used instead of 'Thing' are 'Object', 'Item', 'Resource', 'Component' and as this class represent a direct - general implementation object to the FileSystem class, they could be named as specifically as 'FileSysyemObject", 'FileSysyem_Object" or in short, 'FSObject.

IV. Abstract:

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

Abstraction is a fundamental concept of OOP that simplifies complex systems by ignoring irrelevant details from users. Rather than delving into the inner workings, abstraction permits users to interact with a system through well-defined inputs. Additionally, abstraction involves representing an entity by filtering out unimportant specifics, emphasizing the essential aspects. In OOP, abstraction enables users to define classes or entities without the need to focus on excessive details.

To design a class representing a book, we need to implement a code structure that is similar to this:

+ Implement properties: relevant attributes such as the title, author, publish-date with the book types (audio book or reading book), genres (action, adventure, novel, documentary), and their target audience (for children, teenagers or adults).
+ Implement constructor: relevant constructors to the mentioned properties.
+ Implement classes: a class named as Book and additional classes that responsible to demonstrate the book structure's behaviours/functions, and storing the book's content itself.
+ Implement abstraction: neglect unnecessary specifics and focus on direct attributes to the book.

V. References:

[1]Wikipedia Contributors, "Polymorphism (computer science)," Wikipedia, Nov. 06, 2019. https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

[2]"How to Write Meaningful Variable Names? | Writing Clean Code," workat.tech. https://workat.tech/machine-coding/tutorial/writing-meaningful-variable-names-clean-code-za4m83tiesy0#:~:text=Classes%20should%20have%20descriptive%20names (accessed Sep. 27, 2023).

[3]Packtpub.com, 2022. https://subscription.packtpub.com/book/programming/9781788296540/47/ch47lvl1sec55/setting-up-the-book-class#:~:text=Let (accessed Sep. 27, 2023).