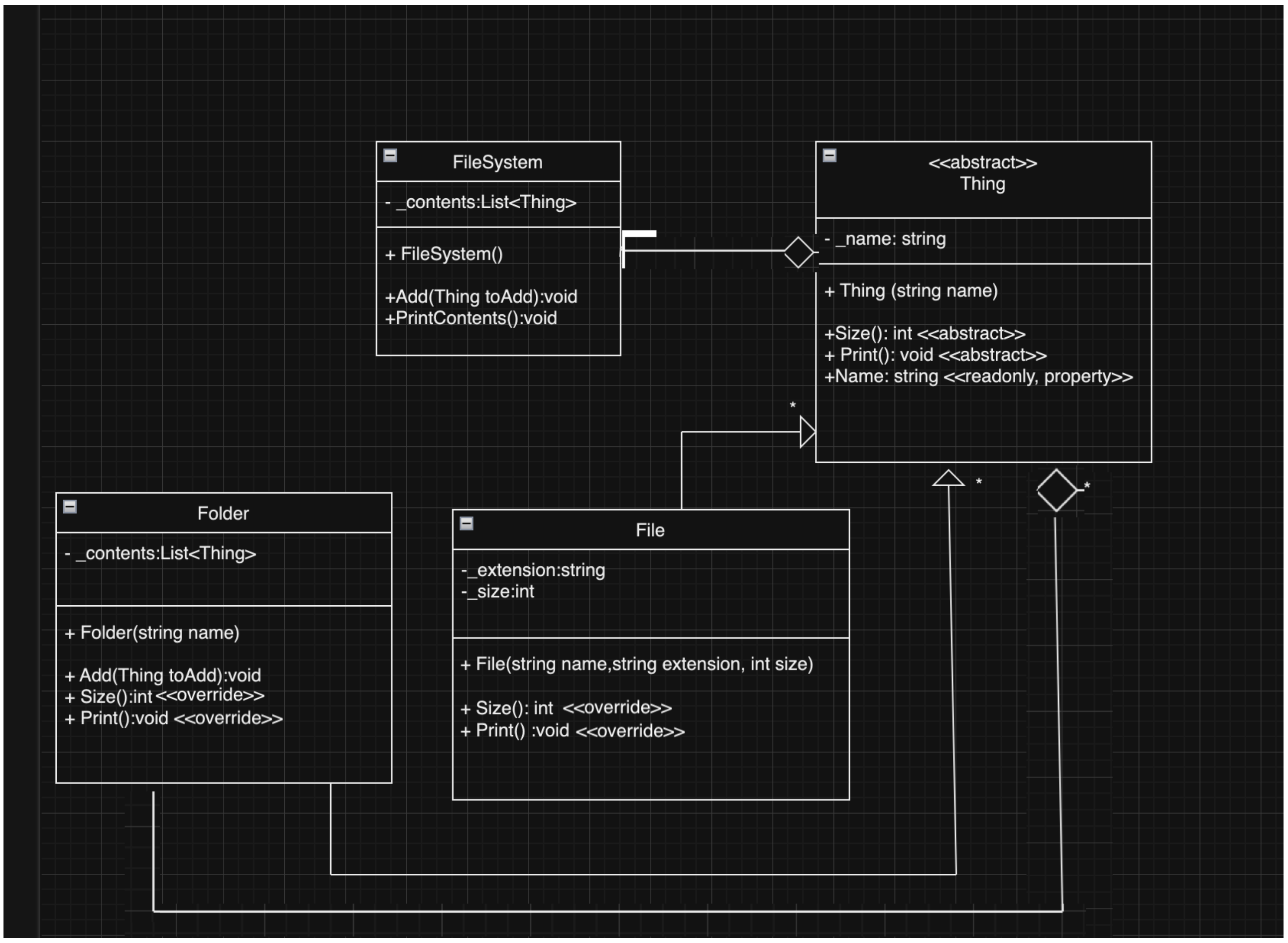
Swinburne University of Technology

COS20007 Object Oriented Programming

Semester test

PDF generated at 17:07 on Monday 16th October, 2023


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

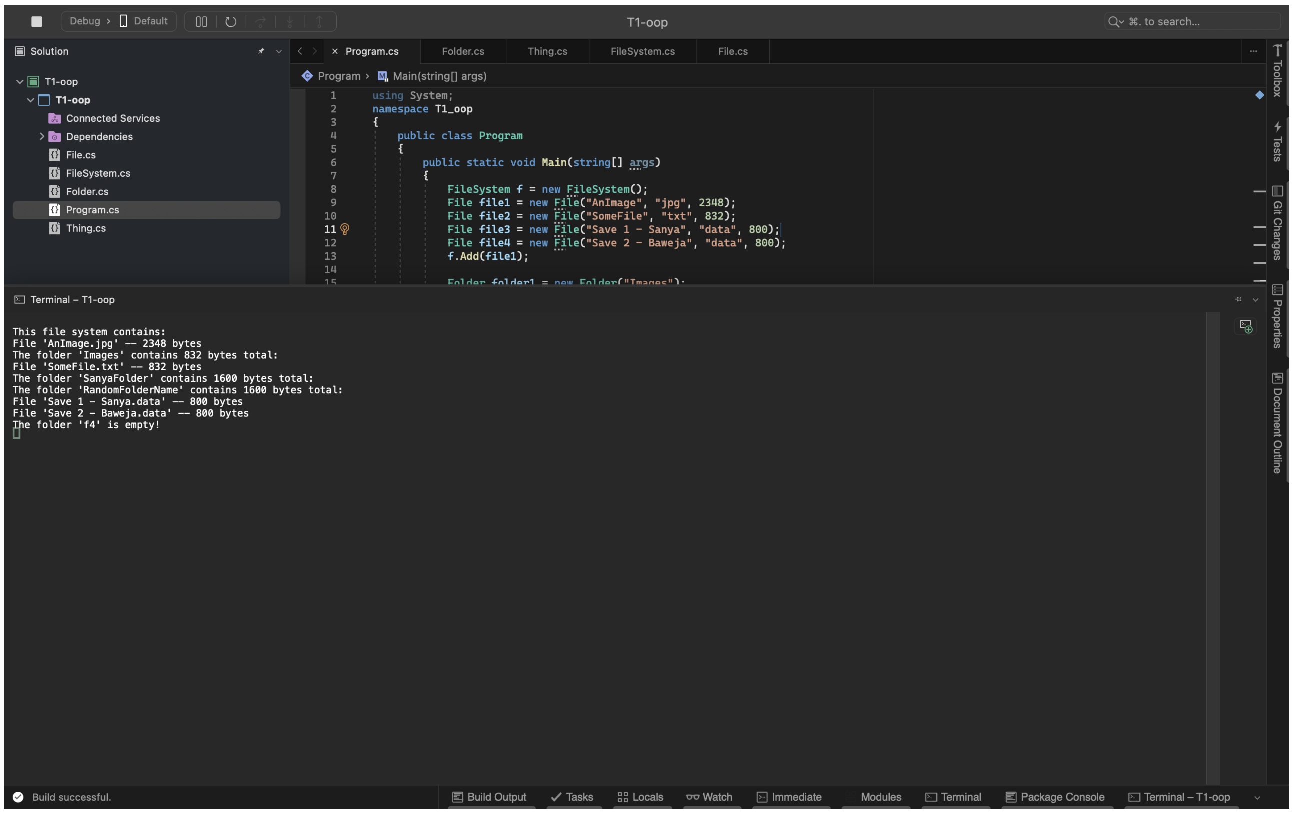
```
using System;
namespace T1_oop
{
    public class Program
    {
        public static void Main(string[] args)
        {
            FileSystem f = new FileSystem();
            File file1 = new File("AnImage", "jpg", 2348);
            File file2 = new File("SomeFile", "txt", 832);
            File file3 = new File("Save 1 - Sanya", "data", 800);
            File file4 = new File("Save 2 - Baweja", "data", 800);
            f.Add(file1);

            Folder folder1 = new Folder("Images");
            folder1.Add(file2);
            f.Add(folder1);

            Folder folder2 = new Folder("RandomFolderName");
            Folder folder3 = new Folder("SanyaFolder");
            folder2.Add(file3);
            folder2.Add(file4);
            folder3.Add(folder2);
            f.Add(folder3);


            Folder folder4 = new Folder("f4");
            f.Add(folder4);

            f.PrintContents();



            Console.ReadLine();
        }
    }
}

```

```
using System;
namespace T1_oop
{
    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 t in _contents)
            {

                t.Print();
            }
        }
    }
}

```

```csharp
using System;
namespace T1_oop
{
    public abstract class Thing
    {
        private string _name;
        public Thing(string name)
        {
            _name = name;
        }
        public abstract int Size();

        public abstract void Print();

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

```

```
using System;
namespace T1_oop
{
    public class Folder:Thing
    {
        private List<Thing> _contents;

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

        }

        public void Add(Thing toAdd)
        {
            _contents.Add(toAdd);
        }
        public override int Size()
        {
            int i = 0;

            foreach (Thing file in _contents)
            {
                i += file.Size();
            }
            return i;
        }

        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 t in _contents)
                {
                    t.Print();
                }

            }
        }
    }
}

```

```
using System;
namespace T1_oop
{
    public class File : Thing
    {
        private string _extension;
        private int _size;

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

        public override int Size()
        {
            return _size;
        }
        public override void Print()
        {

            string filedesc = "File '" + Name + "." + _extension + "' -- " + _size +
↪   " bytes";
            Console.WriteLine(filedesc);


        }

    }
}

```



```csharp
using System;
namespace T1_oop
{
    public class Program
    {
        public static void Main(string[] args)
        {
            FileSystem f = new FileSystem();
            File file1 = new File("AnImage", "jpg", 2348);
            File file2 = new File("SomeFile", "txt", 832);
            File file3 = new File("Save 1 - Sanya", "data", 800);
            File file4 = new File("Save 2 - Baweja", "data", 800);
            f.Add(file1);

            Folder folder1 = new Folder("Images");
```

```
This file system contains:
File 'AnImage.jpg' -- 2348 bytes
The folder 'Images' contains 832 bytes total:
File 'SomeFile.txt' -- 832 bytes
The folder 'SanyaFolder' contains 1600 bytes total:
The folder 'RandomFolderName' contains 1600 bytes total:
File 'Save 1 - Sanya.data' -- 800 bytes
File 'Save 2 - Baweja.data' -- 800 bytes
The folder 'f4' is empty!
```

Test 1 – OOP

Task 2

1) Polymorphism:
   My idea of polymorphism is 'existing in many forms. It really basically knows to implement child classes methods when their instance is created and where instances of parent are expected. The example below signifies this. It defines that methods can be used in different forms and each class can implement it based on its own way. It is like different classes respond to a same method in their own suited way. For example, In task 1, we created methods like Size and Print which are declared in parent class Thing but are overridden, or in simple terms, their implementation is varied in child classes. Each of the classes: Folder and file implements these methods to reach the desired output.
   Out of the two concepts of polymorphism, we use method overriding which was implemented in the print and size methods of the child classes which define their own unique implementation. Now polymorphism is in action when I define the PrintContents method in FileSystem which checks the contents in the list of things which can be anything out of folder and file and decides the implementation of the size and print methods only at runtime based on what the thing is. This way methods can have the same name but can be implemented differently.

2) In the modified design I think removing the FileSystem class won't cause any problem because now using the Thing class we can manage with adding both files and folders to a folder. It is just that FileSystem gave a centralized approach to manage overall storage of files, folders, files within folders and folders within folders. Hence, these can be merged into single entity which can store all types of things irrespective of what their functionalities are. On a broader view, FileSystem is offering quite similar responsibility as the Folder. So removing the former would make the code even simpler.

3) Nah, I don't really think that Thing is a good name for the abstract class because it does not give any real description of what the class is about and what purpose it offers. It is very generalised and anything could be named like that. According to me a good name could be Entity or FileSystemComponents or Contents which we used as an attribute too. These somehow point that we are talking about something that is a part of the program. Like an entity which makes up folders and subfolders or components which are a part of the storage system.

4) Abstraction: It is like removing the details which may not be necessary at that point of time but maybe significant at some point. Abstraction helps identify classification,

roles, responsibilities and collaboration. It is more about getting rid of complexity in the representation, idea or design.

This can be seen as creating the overall template for some actual classes which share common attributes. It can also be seen as a way to create certain methods which is necessary for the actual classes to define hence giving them responsibilities. For collaboration, the objects work in a team by fulfilling their own roles along with common objective.

In case of a Book class, abstraction can be achieved by identifying certain attributes which are essential for a Book like ‘Author’, ‘ISBN No.’, ‘Genre’, ‘Title’, ‘Page No.’ etc. Also, certain methods can be declared which may have different implementation for each real book. There can be methods like Description which include the above attributes but may not show all of them for all books. Like an electronic book may not have ISBN or price. And audio books may not have page no. In these cases abstraction comes as a saviour. So, we can declare the abstract method in the class Book and each child class have to mandatory implement these methods.

Reference:

1) Week 7 Online Lecture Recording

2) The Object Oriented Thought Process by Matt Weisfeld.

3) https://www.techtarget.com/whatis/definition/abstraction#:~:text=In%20object%2Doriented%20programming%2C%20abstraction,reduce%20complexity%20and%20increase%20efficiency.