School of Science, Computing and Engineering Technologies

Object Oriented Programming

Hurdle Task 1: Semester Test

Overview

Note: This hurdle task is a time-bound test. You have a 48 hour window during week 8 to complete it.
- **If you receive a Complete grade**, you have passed the hurdle and can include the test as evidence of that in your portfolio.
- **If you receive a Fix grade**, you must correct all issues and get your test signed off as Complete to meet the hurdle requirements. If you do not get it marked as Complete during the teaching period, you must include a copy of your corrections in your portfolio to demonstrate that you have addressed the issues raised. Failure to do this will result in an overall fail grade for the unit.
- **If you receive a Redo grade**, you must pass a re-sit test that will occur in week 12 in order to meet the hurdle requirements.

In this unit, you have been using object-oriented programming to implement all of your programs. For this task, you will need to show your understanding of the OO principles.

Purpose:	Demonstrate your understanding of object-oriented programming and the core concepts of object-oriented design.
Task:	You must complete two tasks. The first is a coding task, and the second is a series of short answer questions.
Time:	This task should be completed during week 8 — see Canvas for the assignment window.

Submission Details

You must submit the following files, formatted using formatmytask.com:

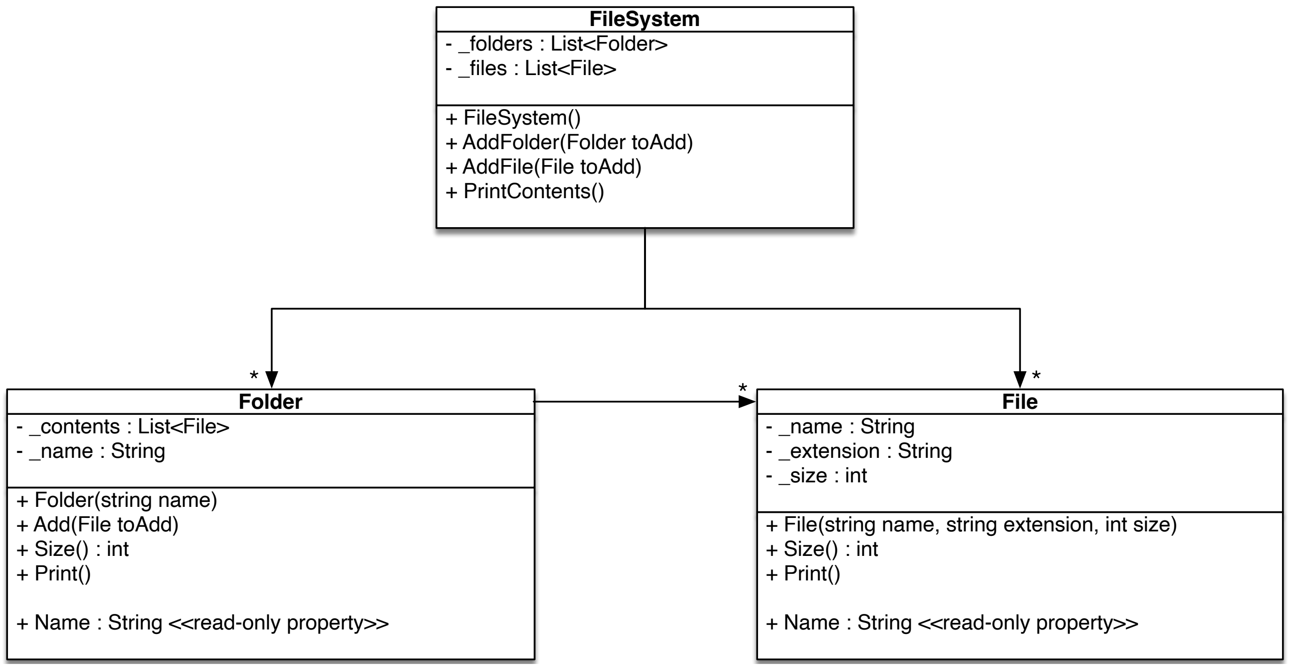
- For Task 1:
  - C# code files of the classes created
  - An image file showing your modified design as a UML class diagram
  - A screenshot of the program output
- For Task 2:
  - A PDF document with your answer

Make sure that you submit code that is readable, follows the universal task requirements, and is appropriately documented.

SWINBURNE UNIVERSITY OF TECHNOLOGY

Task 1

Consider the following program design:



FileSystem is a class that contains knowledge of folders and files within a system. Individual folders in this system are represented by instances of the **Folder** class, and can be added to the file system using **AddFolder**. Individual files in this system are represented by instances of the **File** class, and can be added to the file system using **AddFile**. Every folder and file has:

- A name
- A method which returns its size in bytes
  - For files this method simply returns the size of the file
  - For folders this method returns the sum of the sizes of all of the files it contains
- A method to print a description of itself
  - For files this description is the file's name, extension, and size
  - For folders this description is the folder name, followed by a description of each of the files it contains

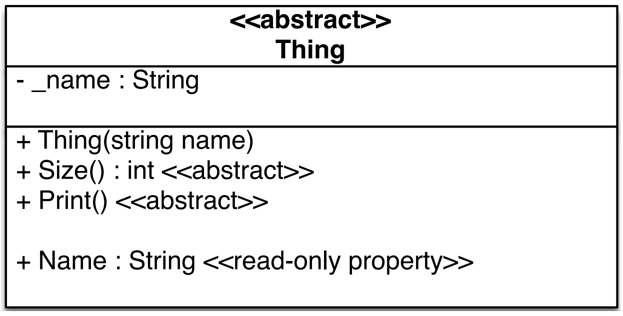
A file also has an extension, and a folder has the ability to add files to its collection.

A summary of all of the folders and files in the file system can be printed to the terminal using the **PrintContents** method of the FileSystem class. The following image is an example output of calling PrintContents:

```
This file system contains:
The folder 'Save files' 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 'New folder' is empty!
File 'AnImage.jpg' -- 5342 bytes
File 'SomeFile.txt' -- 832 bytes
```

You may have noticed that there is a big issue with the abstraction in this program design. Currently, folders can only contain files! In a real file system, folders can contain folders and/or files.

Your task is to redesign this program to represent a more accurate abstraction of a file system. To do this, you should restructure the program to use an **abstract Thing class**. This class should adhere to the following UML design:



```
<<abstract>>
Thing
- _name : String

+ Thing(string name)
+ Size() : int <<abstract>>
+ Print() <<abstract>>

+ Name : String <<read-only property>>
```

To implement this new class and integrate it with our existing design, use the following steps:

1. Implement the **Thing** abstract class according to the above design.
2. Update **Folder** so that it now stores instances of **Thing** instead of **File**.
3. Redesign and implement the **Folder** and **File** classes to be child classes of the new Thing class.
4. Modify **FileSystem** to have only one collection, called **_contents**.
5. Replace the **AddFolder** and **AddFile** methods in **FileSystem** with a single method called **Add**.
6. Implement a **PrintContents** method in FileSystem.
7. Write a simple **Main** method to demonstrate how your new design works. Make sure you clearly show:
   a) Creating a **FileSystem**.
   b) Adding files to the file system.
   c) Adding a folder that contains files to the file system.
   d) Adding a folder that contains a folder that contains files to the file system.
   e) Adding an empty folder to the file system.
   f) Calling the **PrintContents** method.

You are required to:

a) Provide a new UML class diagram for your updated design (hand drawn is fine).

b) Write the code for **all classes**, including the **Thing** abstract class, and all methods/fields/constructors required.

c) Write a simple **Main** method as described above.

d) Provide a screenshot showing the output of your program.

Task 2

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

Tip: Do not get distracted by "ad hoc" or "parametric" polymorphism, which are not specific to object oriented programming.

2. 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.
3. What is wrong with the class name **Thing**? Suggest a better name for the class, and explain the reasoning behind your answer.
4. Define the principle of **abstraction**, and explain how you would use it to design a class to represent a Book.

Tip: In object-oriented programming we have talked about the concepts of **abstraction** and **abstract classes**. Remember that they are different, and we are asking you to explain the first one!

Note: Write your answers **in your own words**. You can use as many reference materials as you like, but you must not directly copy text from anywhere.

Assessment Criteria

Outcome	Requirements
Pass	All parts of the submission are correct.
Fix and Resubmit	The submission clearly demonstrates that the author understands the key OO concepts and how to apply them in code, but there are some issues.
Redo	The submission does not clearly demonstrate that the author understands the key OO concepts and how to apply them in code. There are likely clear mismatches between the provided design and the code submitted, and one or more of the written answers are incorrect. UML may not match the code submitted. OR The submission was not in the correct format.