School of Science, Computing and Engineering Technologies

Object Oriented Programming - May 2025

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. **Late submissions will incur a penalty to your final grade**. The test will automatically close and stop accepting submissions ten hours after the due time.

- If you receive an **Acknowledgment** grade, you have passed the hurdle and can include the test as evidence of that in your portfolio.
- If you receive a **Revision** grade, you must correct all issues within **two weeks** and get your test signed off as Submission Acknowledgment to meet the hurdle requirements. Failure to do this will result in an overall fail grade for the unit.

If you have special consideration, i.e., medical certificates, **you must send to the convenor at least one day before this hurdle test is announced.**

If you have an Education Access Plan issued by Swinburne Accessibility team and would like to request an extension, **you must send to the convener at least two days before the test**.

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. **The task contains personalized requirements.**
Time:	This task should be completed during week 8 — see Canvas for the assignment window.

Submission Details

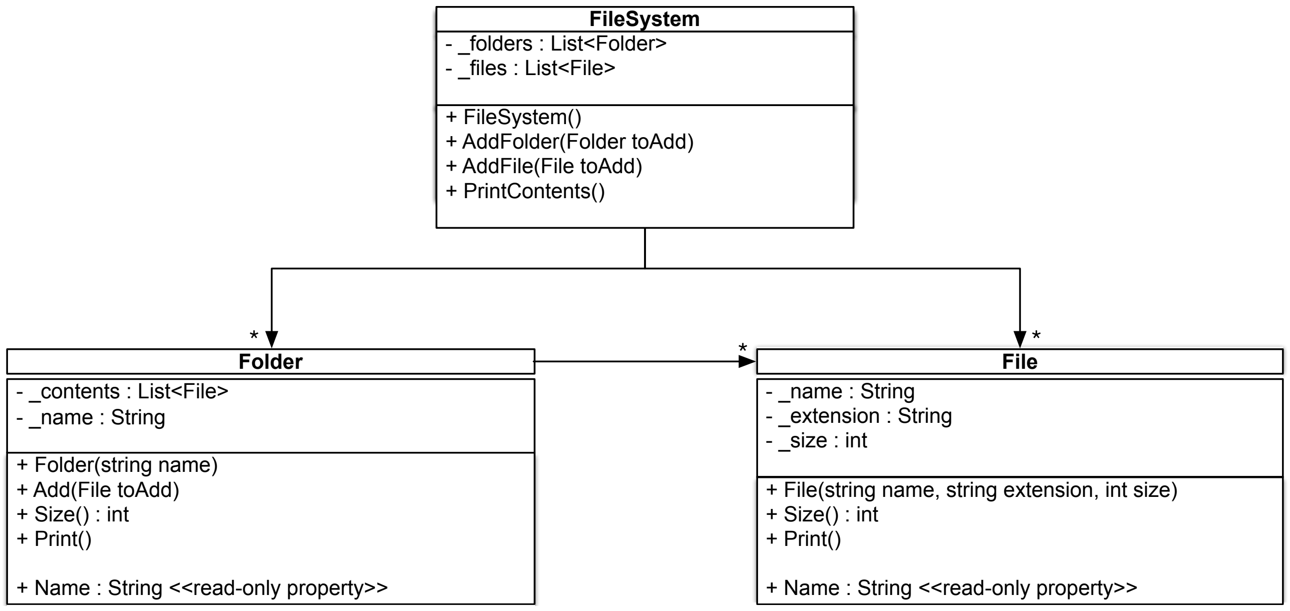
Please print your solutions to PDF and combine it with the screenshot taken for this test.

- 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.

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 and/or subordinate folders it contains.

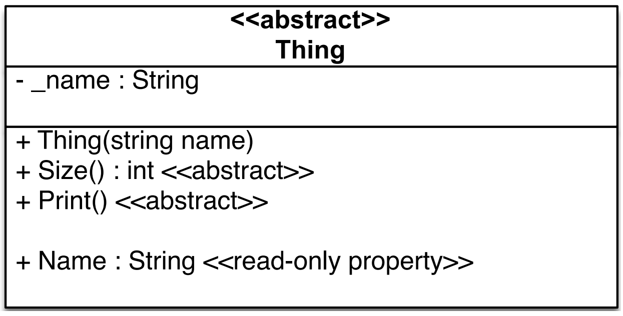
A folder has the ability to add files to its collection.

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 content is an example output of calling PrintContents:

```
This File System contains:
The Folder: 'Saved Files' contains 5 files totalling 23780 bytes:
File 'Martin The Warrior by Brian JAcques.text' Size: 3720 bytes
File 'Northern Lights by Philip Pullman.text' Size: 3990 bytes
File 'The Escapement by K. J. Parker.text' Size: 4070 bytes
File 'Snow Crash by Neal Stephenson.text' Size: 4400 bytes
File 'Renegade's Magic by Robin Hobb.text' Size: 7600 bytes
The Folder: 'New Project' is empty!
File 'ArtWork.jpg' Size: 5342 bytes
File 'PriceList.xls' Size: 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:



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. Let A be an array of the first 10 prime numbers, e.g. 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, where A[index$_0$] = 2, A[index$_1$] = 3, A[index$_2$] = 5, A[index$_3$] = 7, A[index$_4$] = 11, A[index$_5$] = 13, A[index$_6$] = 17, A[index$_7$] = 19, A[index$_8$] = 23, and A[index$_9$] = 29.
8. Use the last four digits of your student ID as indices, manually identify your new array called B where each element in B is the index into the array A. For example, a student ID ending '9270' will have the array of A[index$_9$] A[index$_2$] A[index$_7$] A[index$_0$], then B = [29,5,19,2].
9. Write a **Main** method to demonstrate how your new design works. Make sure you clearly show:
   a) Creating a **FileSystem**.
   b) Adding a number of **B[0]** files to the file system. For example, if **B[0]** = 29, 29 files will be added. Each file should be named with the format "YourStudentID-index.txt". For example, '1119270-00.txt', '1119270-01.txt', ..., '1119270-28.txt'
   c) Adding a folder that contains a number of **B[1]** files to the file system. The files have above format name.
   d) Adding a folder that contains a folder that contains **B[2]** files to the file system.
   e) Adding a number of **B[3]** empty folders 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 the **FileSystem** and **Folder** classes
5. Which Pass (and Credit) tasks you have submitted to Canvas utilize both the principles of **polymorphism** and **abstraction**? You can list two examples.

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.