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

polymorphism gives subclass objects the ability to override superclass methods and display particular behaviors. Because objects of subclasses can be handled using the superclass type without concern for the particular subclass, this promotes flexibility and code reuse. A key component of OOP, polymorphism encourages flexibility and maintainability while developing intricate programs. It makes code more reusable and reduces dependencies amongst system components, making development and extension simpler.

In Task 1, polymorphism was demonstrated through method overriding. Here's how it was used:

The abstract class Thing defines common behavior for all objects in the file system, such as calculating size and printing information. It declares abstract methods Size() and Print().

The File class extends Thing and provides specific implementations for Size() and Print(). By overriding these methods, File objects can exhibit behavior unique to files, such as displaying file extension and size.

Similarly, the Folder class extends Thing and overrides Size() and Print() methods to handle folders' specific behavior. It calculates the total size of all items in the folder and displays information about its contents.

The FileSystem class contains a list of Thing objects, including both files and folders. When invoking the PrintContents() method, it iterates through this list and calls the Print() method on each object. Despite being treated uniformly as Thing objects, polymorphism ensures that the appropriate Print() method is invoked based on the actual type of each object (either File or Folder), allowing each object to display its specific information.

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

Yes, since the FileSystem and Folder classes represent different parts of the file system and have different uses, we require both of them.

The file system as a whole, including all of its files and folders, is represented by the FileSystem class. It is responsible for maintaining the file system's general organization and functionality, including adding new files and directories and publishing the system's contents. It serves as a container for arranging and modifying different file system components.

Folder class: In the file system, this class represents individual folders. Related files and other folders can be grouped together and organized using folders. They can print information about themselves, calculate the overall size of their contents, and have a list of contents among other unique characteristics and activities.

Although it could appear that the Folder class could manage various FileSystem functions, including maintaining the file and folder hierarchy, breaking these responsibilities up into several classes adheres to the separation of concerns principle and improves the codebase's modularity and maintainability. Furthermore, having a distinct FileSystem class gives administrators more freedom to manage and expand the file system's capabilities in the future. As a result, both courses are required for a well-thought-out and structured file system solution.

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

A class named "Thing" is extremely general and gives no useful information about the class's objectives or behavior. To make code easier to read and maintain, class names should be clear and representative of their function inside the system.

The classes name "FileSystemItem" or "FileSystemEntity" would be more appropriate. These names reveal further information about the class's function and identify the file system entity that it represents. The terms "FileSystemItem" and "FileSystemEntity" make it apparent that the class represents any type of element that can be found in a file system, including files, folders, and other entities. This facilitates developers' comprehension of the function of the class and how it fits into the larger structure of the file system.

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

When creating a class to represent a book, abstraction means concentrating just on the essential elements of a book and ignoring any superfluous details. It entails modeling the Book class without non-essential features like color or scent, while including necessary features like title, author, release year, page count, and content. The Book class becomes more streamlined, understandable, and more appropriate for its intended use inside the software system by abstracting away these superfluous elements. This method guarantees that the class contains only the fundamental aspects needed to represent a book in an effective manner, reduces the codebase, and improves readability.