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

Polymorphism is a concept of object-oriented programming where objects from different classes can be treated like objects from a common superclass. It provides a way to perform one single action in various forms. In practical terms, polymorphism allows methods to have some different implementations but with the same call.

Usage in Task 1: In Task 1, polymorphism is implemented using the abstract class `Thing`. The classes `File` and `Folder` both inherit from the class `Thing`. The classes implement both methods, `Size`, and `Print`, each differently. This allows for a collection of `Thing` objects to accept a method call such as `Size` or `Print` without knowing ahead of time which objects are `File` and which are `Folder`.

Question 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.

Yes, we need both the `FileSystem` and `Folder` classes in the updated design for the following reasons:

- **FileSystem Class:**
  - Represents the entire file system.
  - Provides a method to add `Thing` objects (either `File` or `Folder`) to the system.
  - Manages a single collection (`_contents`) of `Thing` objects and provides a `PrintContents` method to display the structure of the file system.
- **Folder Class:**
  - Represents a directory within the file system that can contain both `File` and `Folder` objects.
  - Provides methods to add `Thing` objects to the folder and to calculate the total size of its contents.
  - Implements the `Print` method to display its contents, including nested folders and files.

The `FileSystem` class manages the top-level structure, while the `Folder` class manages hierarchical organization within the file system. Removing either class would result in loss of functionality or complicate the design.

Question 3: 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` is too vague and does not convey any specific meaning about its purpose or role within the file system. A better name would be `FileSystemItem`.

Reasoning:

- **Descriptive:** `FileSystemItem` clearly indicates that the class represents an item within the file system.
- **Contextual:** It provides context to its role as a common base class for `File` and `Folder`.
- **Understandable:** It improves code readability and understanding for anyone maintaining or extending the code in the future.

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

Abstraction is an object-oriented programming principle that involves hiding the complex implementation details and showing only the essential features of an object. It allows focusing on what an object does instead of how it does it.

Designing a Book Class Using Abstraction

When we design a `Book` class using abstraction, we aim to show only the essential details that are important to understand and use a `Book` object, while hiding the complex details of how these features work internally.

Key Steps:

1. **Identify Essential Attributes and Behaviors:**
    - **Attributes (What a Book Has):** These are the basic details about a book that you would care about, such as:
        - Title
        - Author
        - ISBN (a unique identifier for the book)
        - Page Count
        - Publisher
    - **Behaviors (What a Book Does):** These are actions or functionalities related to the book, like:
        - Printing the book's details
2. **Create a Simple, Clean Interface:**
    - **Properties:** These allow us to get the values of the book's attributes without worrying about how they are stored or managed inside the class.
    - **Methods:** These perform actions related to the book.

Example in Code:

```
1 reference
public class Book
{
    // Private attributes
    2 references
    private string _title;
    2 references
    private string _author;
    2 references
    private string _isbn;
    2 references
    private int _pageCount;
    2 references
    private string _publisher;
    // Constructor
    0 references
    public Book(string title, string author, string isbn, int pageCount, string publisher)
    {
        _title = title;
        _author = author;
        _isbn = isbn;
        _pageCount = pageCount;
        _publisher = publisher;
    }
    // Public properties
    1 reference
    public string Title => _title;
    1 reference
    public string Author => _author;
    1 reference
    public string ISBN => _isbn;
    1 reference
    public int PageCount => _pageCount;
    1 reference
    public string Publisher => _publisher;
    // Methods
    0 references
    public void PrintDetails()
    {
        Console.WriteLine($"Title: {Title}");
        Console.WriteLine($"Author: {Author}");
        Console.WriteLine($"ISBN: {ISBN}");
        Console.WriteLine($"Page Count: {PageCount}");
        Console.WriteLine($"Publisher: {Publisher}");
    }
}
```

This design hides the complexity of storing and managing attributes, exposing only what is necessary to interact with a `Book` object.