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

    - **Polymorphism** is a principle of OO Programming that allows objects of different classes to be treated as they were of the same class. This means that different objects can respond to the same message or method call in different ways. It is typically achieved through interfaces and method overriding.

    - The principle of **polymorphism** is used in the implementation of the '*SummaryStrategy*' base class and its derived classes, *'AverageSummary'* and *'MinMaxSummary'.* The '*DataAnalyser'* class uses the '*SummaryStrategy'* base class as its type, allowing it to use objects of both derived classes in a polymorphic way. This means that *'DataAnalyser'* class can call the *'PrintSummary'* method on an object of either the *'AverageSummary'* or *'MinMaxSummary'* class, depending on the current strategy set for the object.

2. Using an example, explain the principle of **abstraction**. In you answer, refer to how classes in OO programs are designed.

    - **Abstraction** involves identifying the essential features of a system or object and representing them in a simplified form, while hiding or removing the non-essential details. This is achieved by creating classes and objects that provide a public interface that exposes only the essential features and hides the implementation details.

      To determine what to hide or show in OOP, we need to analyse the problem domain and identify the key concepts and entities that are relevant to the problem. We then represent these concepts as classes and objects in our program, focusing on the essential attributes and behaviours.

      For example: If we are creating a program to manage a library, we might identify the key concepts as books, borrowers, and the library itself. We would then create classes for each of these concepts, focusing on the essential attributes and behaviours of each class. For the Book class, this might include attributes such as checking out or returning the book. Similarly, for the Borrower class, we might include attributes such as the borrower's name and contact information, and behaviours such as checking out or returning books.

      Once we have identified the essential features of each class, we can then hide the implementation details by making them private or protected. This ensures that users of the class can only interact with the essential features and are shielded from the internal workings of the class. This simplifies the programming process and makes it easier to maintain and extend the program over time.

3. What was the issue with the original design in Task 1? Consider what would happen if we had 50 different summary approaches to choose from instead of just 2.

    - The issue with the original design in Task 1 was that it relied on string parameters to determine the type of summary approach to use, which makes it harder to maintain as new summary approaches are added. If we had 50 different summary approaches to choose from, we would need to add 50 different if statements to check the value of the parameter and implement the corresponding summary approach. By using the strategy pattern, we can encapsulate each summary approach in its own class and easily swap them in and out without having to modify the DataAnalyser class.

Reference:

Trivedi, J. (2023). *Understanding Polymorphism In C#.* [online] www.c-sharpcorner.com. Available at: https://www.c-sharpcorner.com/UploadFile/ff2f08/understanding-polymorphism-in-C-Sharp/.

www.knowledgehut.com. (n.d.). *Abstraction in C# Tutorial with Examples*. [online] Available at: https://www.knowledgehut.com/tutorials/csharp/csharp-abstraction#:~:text=Abstraction%20in%20C%23-.