103812143_ Luan Nguyen

Task 2

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

Polymorphism is one of the most important concepts of Object-Oriented Programming that allows objects of different types to be treated as they have the same category that make the code more **flexible** and **extensible.** This flexibility allows writing more generic and reusable code, as it can handle different objects without needing to be modified or rewritten for each type while extensibility allows adding new functionality to the code by introducing new classes without having to modify the existing code that relies on the superclass or interface

In Task 1, MinMaxSummary and the AverageSummary are two different strategies of the SummaryStrategy class or, in other words, Summary Strategies have 2 different forms that are the MinMaxSummary (minmax strategy (local variable)) and the AverageSummary (averagestrategy(local variable)). And the DataAnalyser class can call either

```
DataAnalyser analyser = new DataAnalyser(new MinMaxSummary(),numbers);
```

or

```
analyser.Strategy = new AverageSummary();
```

And the Summarise are executed differently based on the given strategy

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

The Clock is responsible for time management and features like alarms, made accessible through a simplified interface. By abstracting complexities, the Clock can collaborate with components like user interfaces and alarm systems seamlessly. This promotes code modularity and reuse, allowing the Clock to be integrated flexibly into various systems while maintaining its functionality and interactions.

Taking designing a Clock as an example. A “Clock ” can **encapsulate** properties such as “seconds”,” minutes”, and” hours ” that are for storing and tracking time and functions such as “set alarm”, “start”, “stop”,…

Through **abstraction**, the Clock hides all the complicated properties and functions that we don’t need to know but understand the interface and how to interact with it. Not only hiding stuff but abstraction also allows **reusability** and **modularity**. Abstraction allows us to create a base clock class that encapsulates common properties and functionalities required by all types of clocks. This base clock class can serve as a reusable blueprint from which you can derive different types of clocks, such as a digital clocks or an analog clock. Moreover, Abstraction also allows us to encapsulate and organize related functionalities within separate classes or modules. Each clock type, such as the digital clock or analog clock, can be

implemented in its own class, encapsulating the specific properties and behaviours relevant to that type.

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.

In the original design, the code is lack of OOP structure which makes the program insufficient and may become more and more complex when having more approaches. If there were 50 different summary approaches instead of just 2, at first, more **memory** will be used compared to the one that applies the OOP structure. Moreover, the programmer has to repeat the same if, or else functions many times which might make the code **unreadable** that make it **hard for updating and debugging**. However, by applying OOP structure(concepts such as **abstraction**, **inheritance**, **polymorphism** and **encapsulation**) as we have done in Task1, common functions are reusable and make it easier for extension and modification without being afraid of code readability