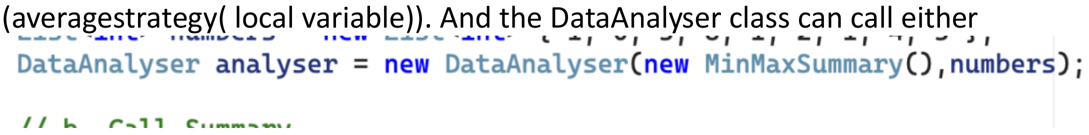
103812143_ Luan Nguyen

Task 2

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

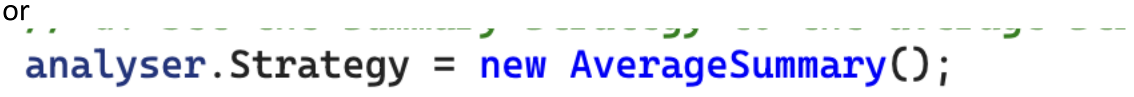
Polymorphism is one of the most important concepts of Object-Oriented Programming that allows objects of different types to be treated as they have the same category that make the code more **flexible** and **extensible.** This flexibility allows writing more generic and reusable code, as it can handle 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.

Abstraction in object-oriented programming is a fundamental concept that simplifies complex systems by focusing on the **roles** and **responsibilities** of properties while **ignoring** unnecessary implementation details. It enables the creation of models or representations of real-world objects or processes, including only the essential information and behaviours required for a specific context. By abstracting complexities, the system becomes more manageable and modular, while offering a clear and simplified interface for interacting with objects.
An example that illustrates the concept of abstraction is the design of a "Counter" in a Clock in Task 3.1. A Counter contains properties such as "seconds," "minutes," and "hours" for storing and tracking time, as well as functions like "Increment," "Tick"and "Reset." Through abstraction, the Counter hides the internal complexities of its implementation, allowing users to use the Clock class to interact with the Counter without needing to know the underlying details.
.

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