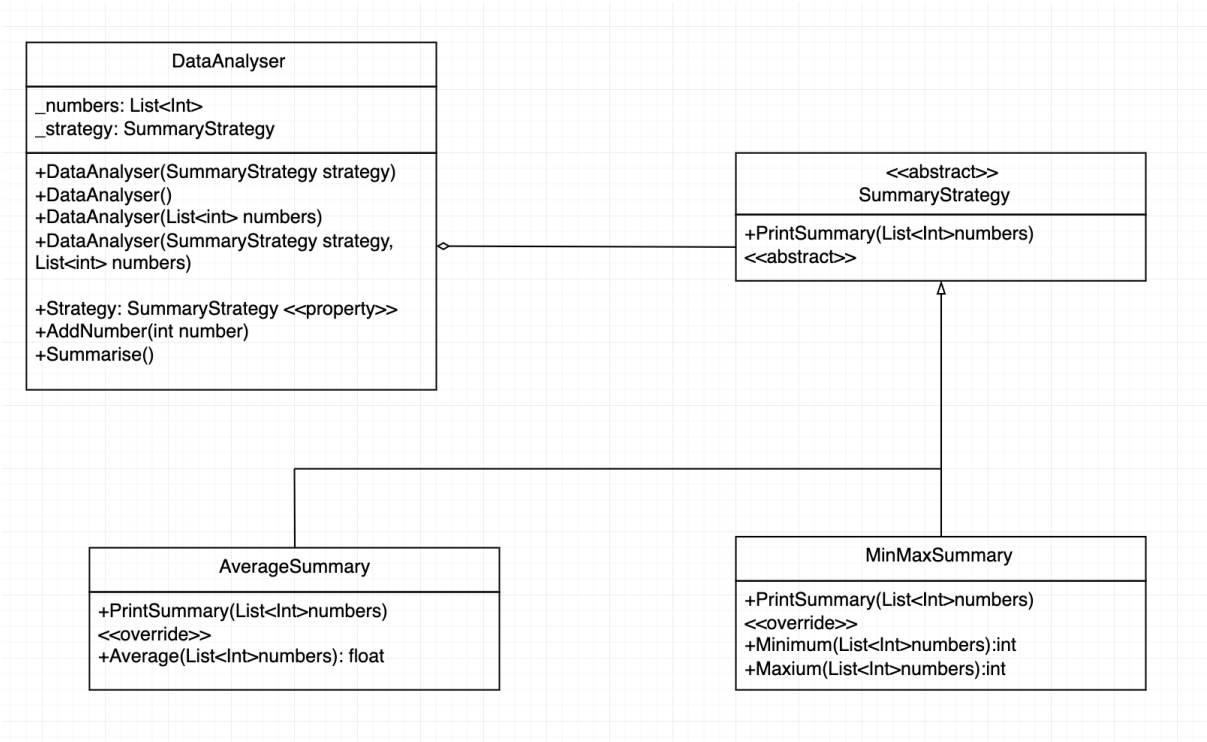
Name: Tran Duc Anh Dang Student ID: 103995439

UML:



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

- Consider a programme with a parent class and several child classes that receive inheritance from the parent class's properties. We will require a notion known as polymorphism if we wish to implement a method from the parent in the child without mixing types. Polymorphism offers a solution in which each child will be able to retain their individual methods and a method may be utilised in the child classes in precisely the same way as their parent class. We must provide an interface for reuse in order for this to function. The offspring will use their own iterations of the parent's procedures, which the parent will have established.

For reference, polymorphism has been implemented in Task 1, specifically property of **Strategy** and **Summarise()** methods. Furthermore, **PrintAverage(List<int>numbers)** and **PrintMinMax(List<int>numbers)** as known as a print function can be used for **AverageSummary** and **MinMaxSummary** through **SummaryStrategy**, even they are in different classes.

Question 2: Using an example, explain the principle of abstraction.

- Programs created using OOP often consist of large code bases with interconnected components. As a result, managing the complexity of updating several code files is a difficult process. Abstraction is the answer to this. Basically, the idea is to hide unnecessary information and just look at an object's key characteristics.

For reference, abstraction has been implemented in this task is a data abstraction which the object data is not visible to the outer world, creating data abstraction, and can only access to the object's data is provided through some methods. Specifically **in** this task, the only way to add a new number to dataanalyser is to call **dataanalyser.AddNumbers(...)** instead of worry about the **_numbers** field that adds new items to the **List<int>.** Furthermore, the property of **Strategy** is another instance of the abstraction concept. To change the strategy that we want to use (Average or MinMax), we need to call **dataanalyser.Strategy** and set it to either **AverageSummary** or **MinMaxSummary.**

Question 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.

- If OOP structure hasn't been applied (original design) in Task 1, if we would have add 50 different summary approaches to choose, we would also have to add 50 different subjects and making the program extremely complex as it's unreadable as we not only miss out the reusability, readability but also the control of its complexity of the code which a huge advantage of OOP. Furthermore, we couldn't hide its implementation details behind some interface therefore with 50 objects all 50 implementations will be public.