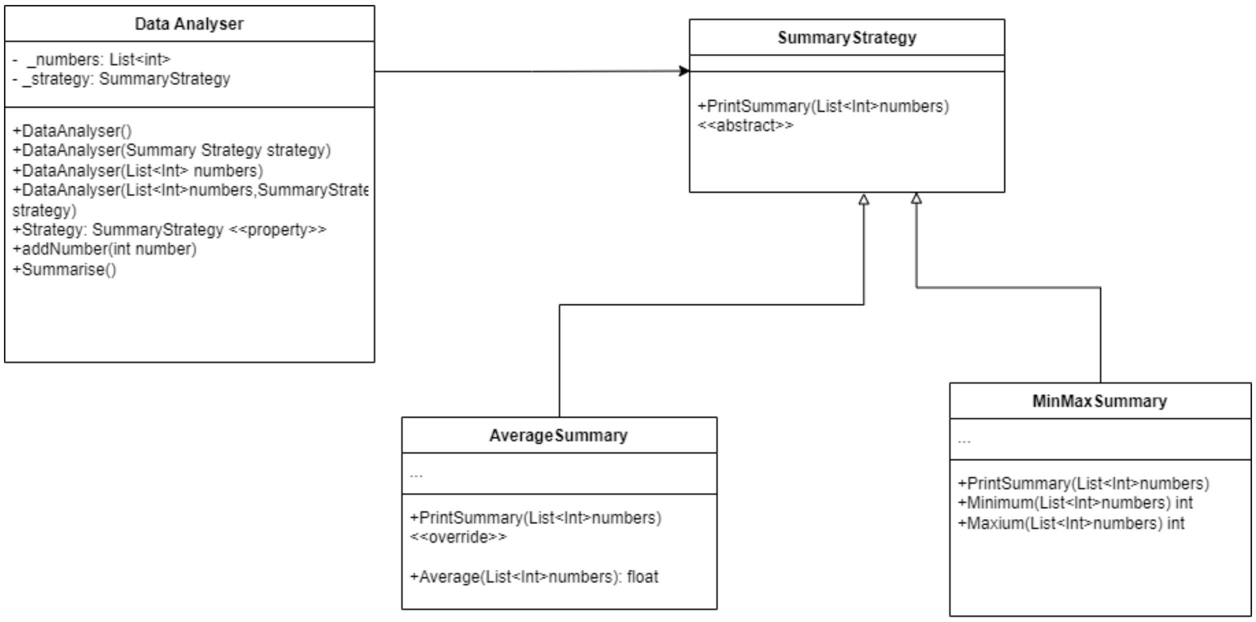
Data Analyser
- _numbers: List<int>
- _strategy: SummaryStrategy
+DataAnalyser()
+DataAnalyser(Summary Strategy strategy)
+DataAnalyser(List<Int> numbers)
+DataAnalyser(List<Int>numbers,SummaryStrate strategy)
+Strategy: SummaryStrategy <<property>>
+addNumber(int number)
+Summarise()
Summary Strategy
+PrintSummary(List<Int>numbers) <<abstract>>
AverageSummary
...
+PrintSummary(List<Int>numbers) <<override>>
+Average(List<Int>numbers): float
MinMax Summary
...
+PrintSummary(List<Int>numbers)
+Minimum(List<Int>numbers) int
+Maxium(List<Int>numbers) int

1.

-The capacity of objects of various classes to be treated as objects of a common base class is known as the object-oriented programming principle of polymorphism. It gives programmers the ability to design code that can operate with objects of different types without having to know the exact type at compile time by enabling objects to be utilized interchangeably.

- In Task 1, the classes "AverageSummary" and "MinMaxSummary" employed polymorphism.

They derive from the abstract "SummaryStrategy" class. Consequently, "SummaryStrategy" provides access to "AverageSummary" and "MinMaxSummary". Additionally, the "SummaryStrategy" class exhibits polymorphism in that it can take on numerous forms.

- As a result, publishing the values from "DataAnalyzer" will be considerably easier because you just need to call out "_strategy."To express "SummaryStrategy" for either AverageSummary or MinMaxSummary in this task, print Summary(_numbers).

2.

- The practice of breaking complicated systems down into smaller, more manageable parts called classes is known as the principle of abstraction in object-oriented programming. Programmers can concentrate on the key characteristics and actions of an item or system by masking non-essential aspects through abstraction.

Classes in an object-oriented program are meant to represent concepts or real-world objects. Every class contains information about an entity, such as its characteristics and methods. The process of abstraction entails determining the traits and actions that a collection of objects have in common and building a class to represent them.

- For example, let's consider a program that models different shapes, such as circles, rectangles, and triangles. Each shape has properties like area and perimeter, and behaviors like calculating those values. By using abstraction, the program can define an abstract base class called "Shape" that declares common properties and methods for all shapes. Specific shape classes like "Circle", "Rectangle", and "Triangle" can then inherit from the "Shape" class and provide their own implementations for the abstract methods.

Abstraction allows the program to treat all shapes uniformly, regardless of their specific type. It simplifies the design and implementation of the program by providing a common interface and hiding the internal details of each shape class. This way, the program can work with shapes without needing to know the specific implementation details of each shape class.

3.

If we have 50 summary methods, the program heavy coupling issue will be more evident and its consequences will scale with the quanity of methods. This also limits code reusability and flexibility, as you would need to write separate code for each derived class to handle specific operations. Individual classes are still able to override PrintSummary as needed, allowing flexibility. But changes are localized rather than requiring updates across many classes.

The abstract class implementation provides a uniform interface for related classes while still enabling customization where needed.

Reference:

https://www.w3schools.com/cs/cs_abstract.php