Ta Quang Tung – 104222196 – COS20007 – T1 – Semester Test Task 2

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

Polymorphism is an OOP principle that allows objects of child classes to be used as objects of the parent class. In task 1, we have two child classes, AverageSummary and MinMaxSummary, both of which are derived from the abstract SummaryStrategy class. Polymorphism is used when the objects of both child classes can be stored as a SummaryStrategy inside DataAnalyser and used as though they are of type SummaryStrategy inside the Summarise method.

2. **Using an example, explain the principle of abstraction.**

In OOP, abstraction means identifying and presenting the essential features of objects while hiding the underlying implementation details from the user. Using task 1 as an example, we identify that any summary strategy has a method that receives a list of numbers and prints its summary to the screen, regardless of the implementation. From this knowledge, we create an abstract class SummaryStrategy that captures this essential feature. We then derive from this class to create child classes with the specific strategies we want to use. Using polymorphism to treat the child classes as the parent class, the user (DataAnalyser) now interacts only with the common PrintSummary method defined in each strategy without having to worry about the specific implementation.

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 we followed the original design of Task 1 to support 50 different summary strategies, we would have to create 50 fields corresponding to each strategy in DataAnalyser and add 50 if-statements to check for the right strategy to use. DataAnalyser would also have to know the method name of each strategy (e.g. PrintAverage, PrintMinMax, etc.), which can result in high coupling because if the names changed, the code inside DataAnalyser would have to be updated.

On the other hand, the new design forces all Strategy classes to derive from an abstract SummaryStrategy class, which requires them to have a common method PrintSummary that takes a list of numbers. Thanks to polymorphism, different strategies can be used interchangeably inside DataAnalyser, thus removing the need to create 50 different fields. DataAnalyser does not have to worry about the potentially different names or parameters of each strategy method, because everything is forced to be PrintSummary(List<int> numbers).