Faculty of Science, Engineering and Technology

Object Oriented Programming

Pass Task 8.1: Semester TEST (Time-limited Evaluation of Skills Task)

Overview

> **Note**: This task is a time-bound test, replacing the usual Semester Test. You have 5 days to complete it (within Week 8 of semester), and only **one** opportunity to resubmit the task after receiving feedback. The test is not a hurdle requirement, but counts as a pass task to be included in your final portfolio.

In this unit, you have been using object-oriented programming to implement all of your programs. For this task, you will need to show your understanding of the principles associated with this programming paradigm.

Purpose:	Demonstrate your understanding of object-oriented programming and the core concepts of object-oriented design.
Task:	You must complete two tasks. The first is a coding task, to be submitted as C# source code files, a UML diagram, and a screenshot showing your program's output. The second task asks for a written response, to be submitted as a PDF.
Time:	This task should be completed during week 8.

Submission Details

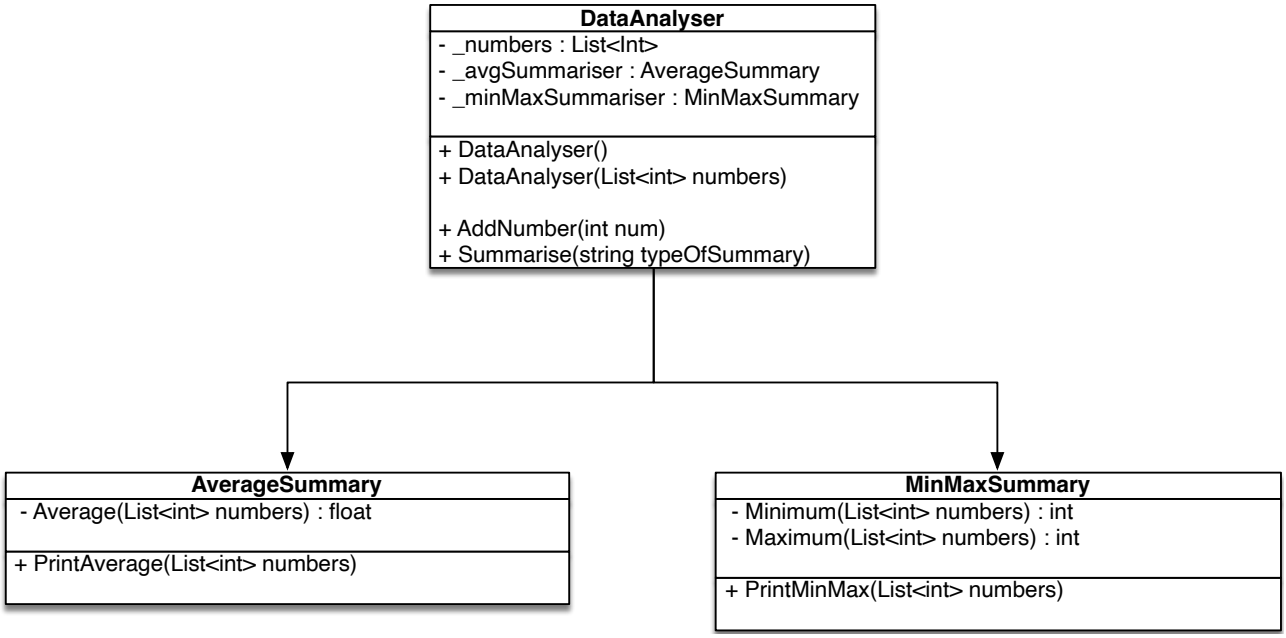
You must submit the following files to Doubtfire:

- For Task 1:
  - C# code files of the classes created
  - An image file showing your modified design as a UML class diagram
  - A screenshot of the program output
- For Task 2:
  - A PDF document with your answer

Make sure that you submit code that is readable and is appropriately documented.

Task 1

Consider the following program design:

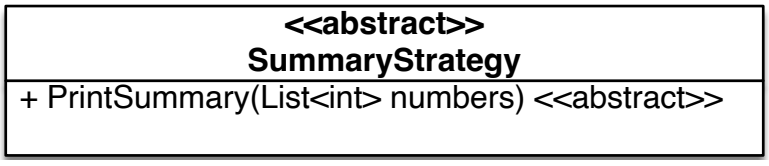


DataAnalyser is a class that can print a summary of a list of numbers. Those numbers can be initialised through the constructor with a parameter, or alternatively be added using the AddNumber method after object initialisation.

Currently there are two ways of summarising the data. These two ways are implemented in the AverageSummary and MinMaxSummary classes. As their names suggest, AverageSummary calculates the average value of the list, and MinMaxSummary prints out the minimum and maximum values of the list.

At the moment, the DataAnalyser can be asked to summarise its list of numbers with a call to its Summarise method. This method accepts a single parameter, which indicates which summary method should be used. If the parameter's value is "average", it should use the average summary. If the value is "minmax", it should use the minmax summary. The casing of the value should not matter (e.g., passing in the value of "aVeraGE" should result in the same behaviour as passing in the value "average").

Your task is to redesign this program to better follow the principles of object-oriented programming, focusing on the concept of **polymorphism**. To do this, you should restructure the program to use an **abstract SummaryStrategy class**. This class should adhere to the following UML design:



To implement this new class and integrate it with our existing design, the following changes need to be made:

1. Implement the **SummaryStrategy** abstract class according to the above design.
2. Redesign the **AverageSummary** and **MinMaxSummary** classes to be child classes of the new SummaryStrategy class.
3. Modify **DataAnalyser** to have a private variable, "**_strategy**", that is of the type SummaryStrategy.
4. Add a public property for this new private variable.
5. Modify the **DataAnalyser** constructors to:
   a) allow the strategy to be set through a parameter
   b) by default (i.e., if there are no parameters), set the strategy to the average strategy.
6. Modify DataAnalyser's **Summarise** method to use the currently stored strategy instead of relying on a string parameter.
7. Write a simple **Main** method to demonstrate how your new design works:
   a) Create a DataAnalyser object with a list of 10 numbers and the **minmax** summary strategy.
   b) Call the **Summarise** method.
   c) Add three more numbers to the data analyser.
   d) Set the summary strategy to the **average** strategy.
   e) Call the **Summarise** method.

Tip: Subclasses need to adhere to the requirements of the parent, but they can still add their own additional functionality. Consider how the code can be broken up so that the methods in your classes have just one purpose and no side effects.

You are required to:

a) Provide a new UML class diagram for your updated design (hand drawn is fine).

b) Write the code for **all classes**, including the SummaryStrategy abstract class, and all methods/fields/constructors required.

c) Write a simple **Main** method as described above.

Submit a photo or image of your UML, all source code files, and a screenshot showing your program working to Doubtfire.

Task 2

1. Describe the principle of **polymorphism** and how it was used in Task 1.
2. Using an example, explain the principle of **abstraction**.
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.

Tip: In object-oriented programming we have talked about the concepts of **abstraction** and **abstract classes**. Remember that they are different, and we are asking you to explain the first one!

Note: Write your answer **in your own words**. You can use as many reference materials as you like, but you must not directly copy text from anywhere.

Submit your answer as a PDF document to Doubtfire.