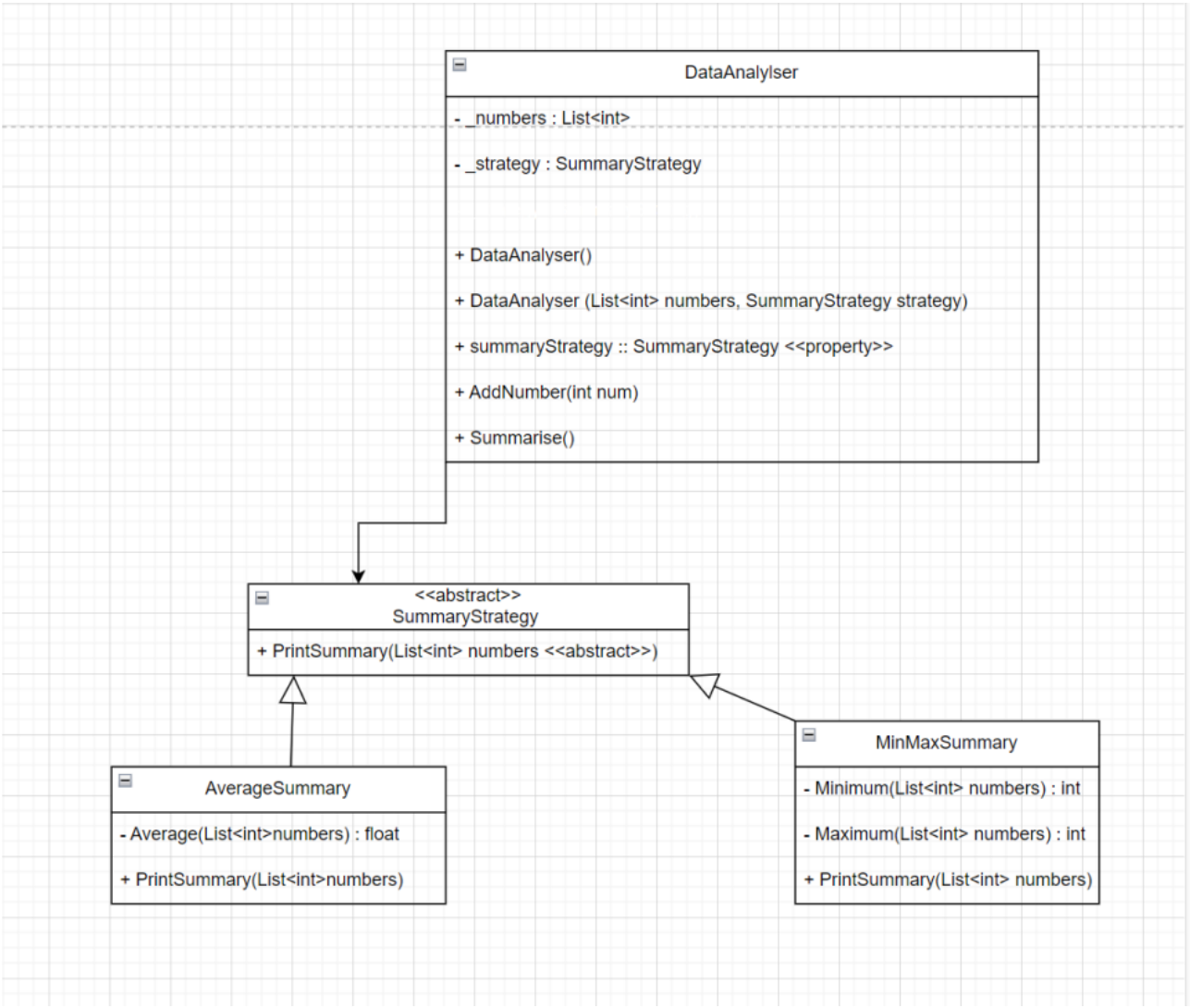
Swinburne University of Technology

COS20007 Object Oriented Programming

T1 - Semester Test

PDF generated at 16:01 on Wednesday 10th May, 2023


DataAnalyser
- _numbers : List<int>
- _strategy : SummaryStrategy
+ DataAnalyser()
+ DataAnalyser (List<int> numbers, SummaryStrategy strategy)
+ summaryStrategy :: SummaryStrategy <<property>>
+ AddNumber(int num)
+ Summarise()
<<abstract>>
SummaryStrategy
+ PrintSummary(List<int> numbers <<abstract>>)
AverageSummary
- Average(List<int>numbers) : float
+ PrintSummary(List<int>numbers)
MinMaxSummary
- Minimum(List<int> numbers) : int
- Maximum(List<int> numbers) : int
+ PrintSummary(List<int> numbers)

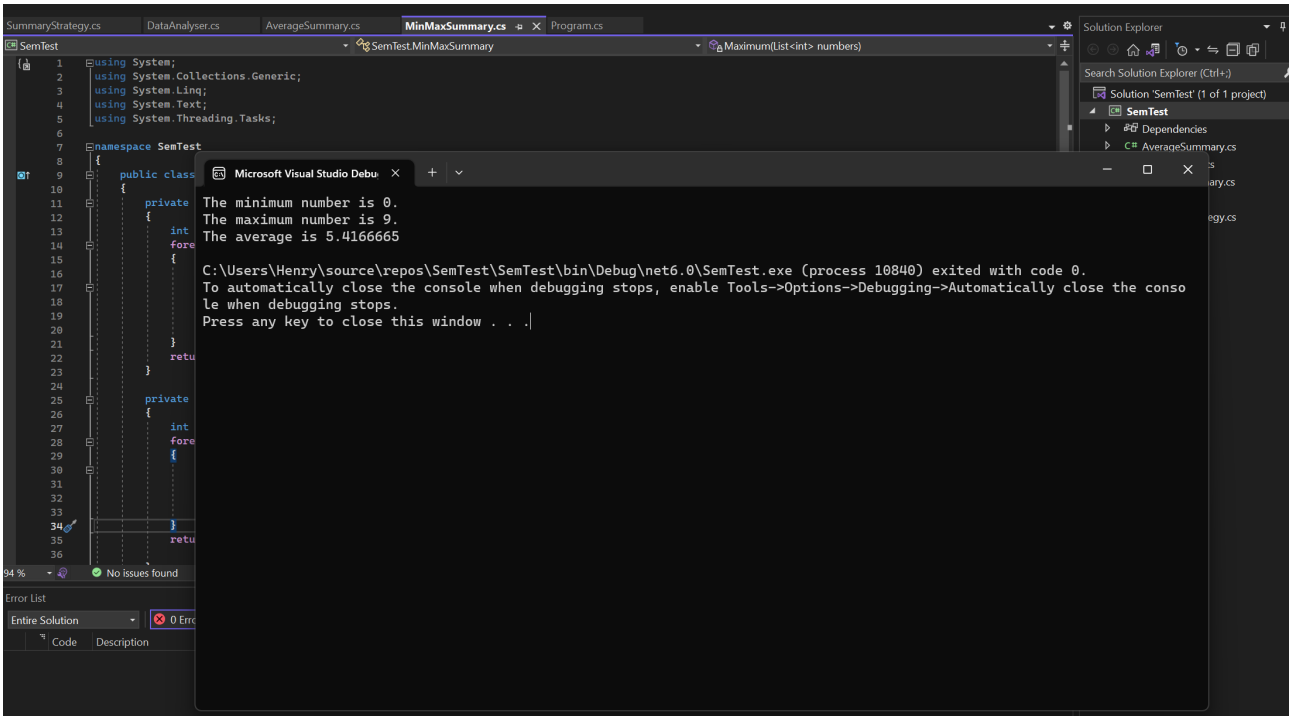
```
using SemTest;

public class Program
{
    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 0, 4, 2, 4, 9, 0, 8, 5 };
        MinMaxSummary minmax = new MinMaxSummary();
        AverageSummary average = new AverageSummary();

        DataAnalyser dataAnalyser = new DataAnalyser(numbers, minmax);

        dataAnalyser.Summarise();

        dataAnalyser.AddNumber(18);
        dataAnalyser.AddNumber(9);
        dataAnalyser.AddNumber(5);

        dataAnalyser.summaryStrategy = average;
        dataAnalyser.Summarise();
    }
}


```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SemTest
{
    public class DataAnalyser
    {
        private List<int> _numbers;
        private SummaryStrategy _strategy;

        public DataAnalyser(): this (new List<int>(), new AverageSummary())
        {

        }

        public DataAnalyser(List<int> numbers, SummaryStrategy strategy)
        {
            _numbers = numbers;
            _strategy = strategy;
        }


        public SummaryStrategy summaryStrategy
        {
            get
            {
                return _strategy;
            }
            set
            {
                _strategy = value;
            }
        }

        public void AddNumber(int num)
        {
            _numbers.Add(num);
        }

        public void Summarise()
        {
            _strategy.PrintSummary(_numbers);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SemTest
{
    public abstract class SummaryStrategy
    {
        public abstract void PrintSummary(List<int> numbers);
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SemTest
{
    public class MinMaxSummary : SummaryStrategy
    {
        private int Minimum(List<int> numbers)
        {
            int MinNum = numbers[0];
            foreach (int number in numbers)
            {

                if (number < MinNum)
                {
                    MinNum = number;
                }
            }
            return MinNum;
        }

        private int Maximum(List<int> numbers)
        {
            int MaxNum = numbers[0];
            foreach (int number in numbers)
            {
                if (number > MaxNum)
                {
                    MaxNum = number;
                }
            }
            return MaxNum;

        }

        public override void PrintSummary(List<int> numbers)
        {
            Console.WriteLine($"The minimum number is {Minimum(numbers)}.\n" +
                $"The maximum number is {Maximum(numbers)}.");
        }



    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SemTest
{
    public class AverageSummary : SummaryStrategy
    {
        private float Average(List<int> numbers)
        {
            int sum = 0;
            foreach (int number in numbers)
            {
                sum += number;

            }
            return (float)sum/numbers.Count;
        }

        public override void PrintSummary(List<int> numbers)
        {
            Console.WriteLine($"The average is {Average(numbers)}");
        }
    }
}
```


SummaryStrategy.cs
DataAnalyser.cs
AverageSummary.cs
MinMaxSummary.cs
Program.cs
SemTest
SemTest.MinMaxSummary
Maximum(List<int> numbers)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SemTest
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SemTest' (1 of 1 project)
SemTest
Dependencies
AverageSummary.cs
Microsoft Visual Studio Debu
The minimum number is 0.
The maximum number is 9.
The average is 5.4166665
C:\Users\Henry\source\repos\SemTest\SemTest\bin\Debug\net6.0\SemTest.exe (process 10840) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
No issues found
Error List
Entire Solution
Code
Description

OOP Task 2

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

Polymorphism refers to objects being able to take many forms. Through abstract classes and interfaces derived classes or classes that share an interface can allows objects to behave and function differently but also be implemented as if they were the same object type.

Interfaces define a functionality so that many classes can implemented by many classes and be used as the same type.

Abstract methods are able to override methods from its parent class, in this way it can be made to behave differently but still be implemented like it is from the same object.

Polymorphism allows for code to be reusable since common interfaces or methods can be shared among different classes it reduces the amount of code required for similar functions.

In task1 AverageSummary and MinMaxSummary classes inherited from the abstract class SummaryStrategy. Through polymorphism SummaryStrategy class is used to access its child classes and its PrintSummaries could be implemented as if they were the same type through the Summarise method.

2. Using an example, explain the principle of abstraction. In your answer, refer to how classes in OO programs are designed.

Abstraction is the concept of mapping out the essential features of an object, the specifics of how the objects are implemented are not included in the process in order to create a digital representation of the object behaves and its core features. OO programs are designed through digital representations like UML diagrams. These diagrams assist in building and designing the program by outlining the roles and collaborations objects may have with each other.

3. What was the issue in 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 there were 50 different summary approaches instead of 2 the program would require different ways of calling each summary approach making reusability very low. This would cause the program to have repetitive codes such as the summarise method which may cause duplication in codes for each approach and make the program inefficient.