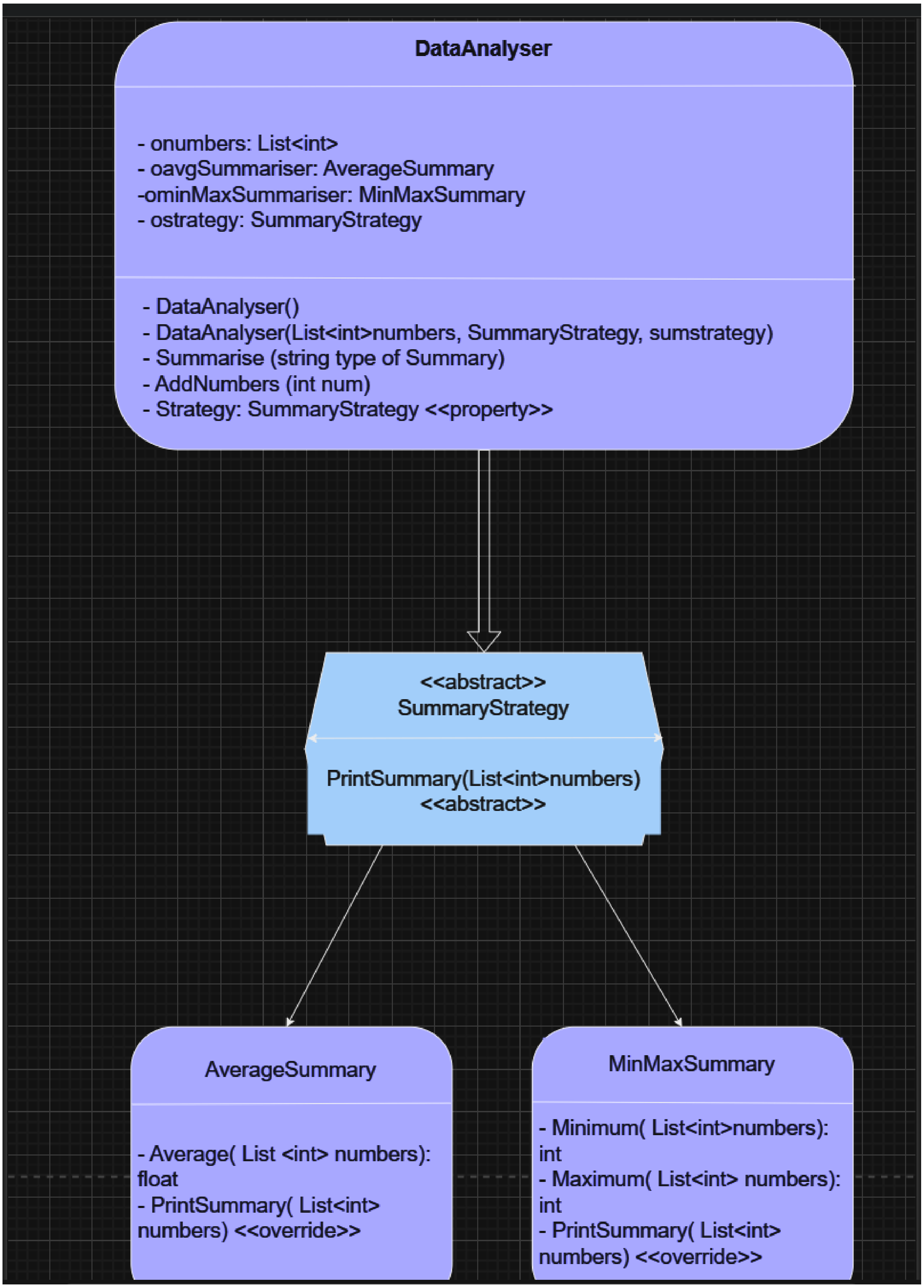
Swinburne University of Technology

COS20007 Object Oriented Programming

Semester test

PDF generated at 18:18 on Thursday 16th November, 2023


DataAnalyser
- onumbers: List<int>
- oavgSummariser: AverageSummary
-ominMaxSummariser: MinMaxSummary
- ostrategy: SummaryStrategy
- DataAnalyser()
- DataAnalyser(List<int>numbers, SummaryStrategy, sumstrategy)
- Summarise (string type of Summary)
- AddNumbers (int num)
- Strategy: SummaryStrategy <<property>>
<<abstract>>
SummaryStrategy
PrintSummary(List<int>numbers)
<<abstract>>
AverageSummary
- Average(List <int> numbers): float
- PrintSummary(List<int> numbers) <<override>>
MinMaxSummary
- Minimum(List<int>numbers): int
- Maximum(List<int> numbers): int
- PrintSummary(List<int> numbers) <<override>>

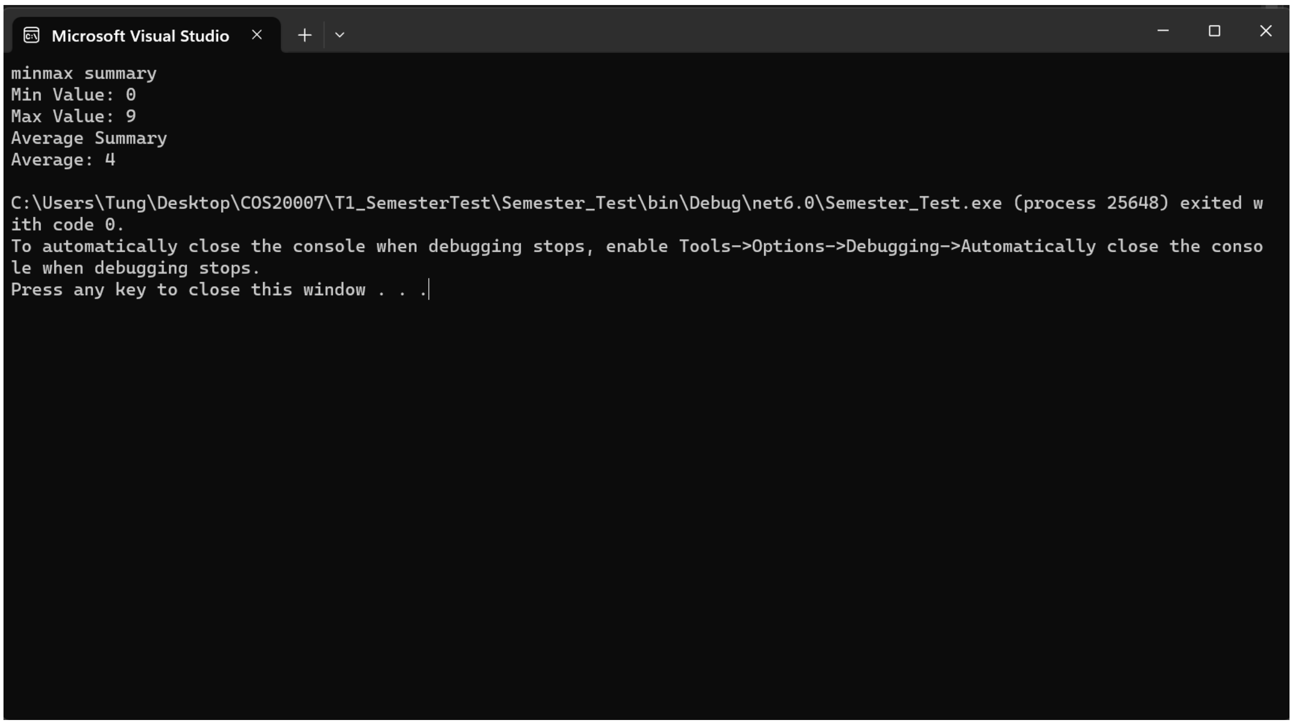
```
namespace Semester_Test
{
    public class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("minmax summary");
            List<int> studentID = new List<int>() { 1, 0, 4, 1, 7, 5, 9, 1, 5 };
            DataAnalyser dataAnalyser = new(studentID, new MinMaxSummary());
            dataAnalyser.Summarise("MinMax");
            studentID.Add(10);
            studentID.Add(7);
            studentID.Add(2);

            Console.WriteLine("Average Summary");
            dataAnalyser.Strategy = new AverageSummary();
            dataAnalyser.Summarise("average");
        }
    }
}
```

```
using Semester_Test;

public class DataAnalyser
{
    private List<int> onumber;
    private AverageSummary oavgSumariser;
    private MinMaxSummary ominMaxSummariser;
    private SummaryStrategy ostrategy;

    public SummaryStrategy Strategy
    {
        get { return ostrategy; }
        set { ostrategy = value; }
    }

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

    public DataAnalyser(List<int> numbers, SummaryStrategy sumstrategy)
    {
        onumber = numbers;
        ostrategy = sumstrategy;
    }

    // New constructor
    public DataAnalyser(List<int> numbers)
    {
        onumber = numbers;
        ostrategy = new AverageSummary();
    }

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

    public void Summarise(string typeOfSummary)
    {
        if (Strategy is MinMaxSummary)
        {
            ostrategy.PrintSummary(onumber);
        }
        else if (Strategy is AverageSummary)
        {
            ostrategy.PrintSummary(onumber);
        }
    }
}
```

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

namespace Semester_Test
{
    public class AverageSummary : SummaryStrategy
    {
        private float Average(List<int> onumber)
        {
            int total = 0;
            foreach (int number in onumber)
            {
                total += number;
            }
            float result = total / onumber.Count;
            return result;
        }
        public override void PrintSummary(List<int> onumber)
        {
            Console.WriteLine("Average: " + Average(onumber));
        }
    }
}
```

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

namespace Semester_Test
{
    public class MinMaxSummary : SummaryStrategy
    {
        private int Minimum(List<int> onumber)
        {
            int small = onumber[0];
            foreach (int i in onumber)
            {
                if (small > i)
                {
                    small = i;
                }
            }
            return small;
        }
        private int Maximum(List<int> onumber)
        {
            int large = onumber[0];
            foreach (int i in onumber)
            {
                if (large < i)
                {
                    large = i;
                }
            }
            return large;
        }
        public override void PrintSummary(List<int> onumber)
        {
            Console.WriteLine("Min Value: " + Minimum(onumber));
            Console.WriteLine("Max Value: " + Maximum(onumber));
        }
    }
}
```

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

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



```
minmax summary
Min Value: 0
Max Value: 9
Average Summary
Average: 4

C:\Users\Tung\Desktop\COS20007\T1_SemesterTest\Semester_Test\bin\Debug\net6.0\Semester_Test.exe (process 25648) exited w
ith 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 . . .
```

Semester Test

Name: Le Ba Tung
ID: 104175915

Task 2

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

In object-oriented programming, the ability of objects of various kinds to react differently to the same method call is known as polymorphism. This is implemented in Task 1 using the abstract base class SummaryStrategy. The Strategy object's PrintSummary function is called by the DataAnalyser's Summarize function. AverageSummary and MinMaxSummary implement PrintSummary differently, which leads to differing behavior, even though it is the same method call.

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

When something is abstracted, extraneous implementation details are hidden and only the most important aspects of the item are revealed. This aids in handling complexity. In object-oriented design, the overall behavior of objects is defined by abstract classes and interfaces, which do not describe how they are implemented. The actual implementations are then supplied by the child classes. For instance, child classes such as AverageSummary are required to implement the PrintSummary function, which is declared by SummaryStrategy. This permits various summary implementations while enforcing a uniform interface for summarizing.

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.

The main problem was that the summarization logic was embedded directly in the DataAnalyser class. This violates the single responsibility principle - the DataAnalyser class was responsible for both storing the data and performing the summarization. Additionally, there was tight coupling between the DataAnalyser and the specific summarization strategies (AverageSummary and MinMaxSummary). The DataAnalyser needed to know about these concrete classes in order to instantiate and delegate to them. This meant that adding a new summarization strategy would require changes to the DataAnalyser class - we would have to add a new if/else branch to handle the new strategy. If there were 50 different strategies instead of just 2, the DataAnalyser class would become very large and unwieldy. It would have a huge if/else statement or switch statement to pick the right strategy. This violates the open/closed principle - the DataAnalyser class is not closed for modification. By using an abstract SummaryStrategy class and polymorphism, we decouple the summarization logic from the DataAnalyser. The DataAnalyser just needs to know how to interact with the SummaryStrategy abstract interface. It doesn't need to know about concrete implementations. This means we can add new summarization strategies freely without ever needing to modify the DataAnalyser. I also create a new class that implements the SummaryStrategy interface. This is a more flexible, extensible object-oriented design.