

SWINBURNE UNIVERSITY OF TECHNOLOGY

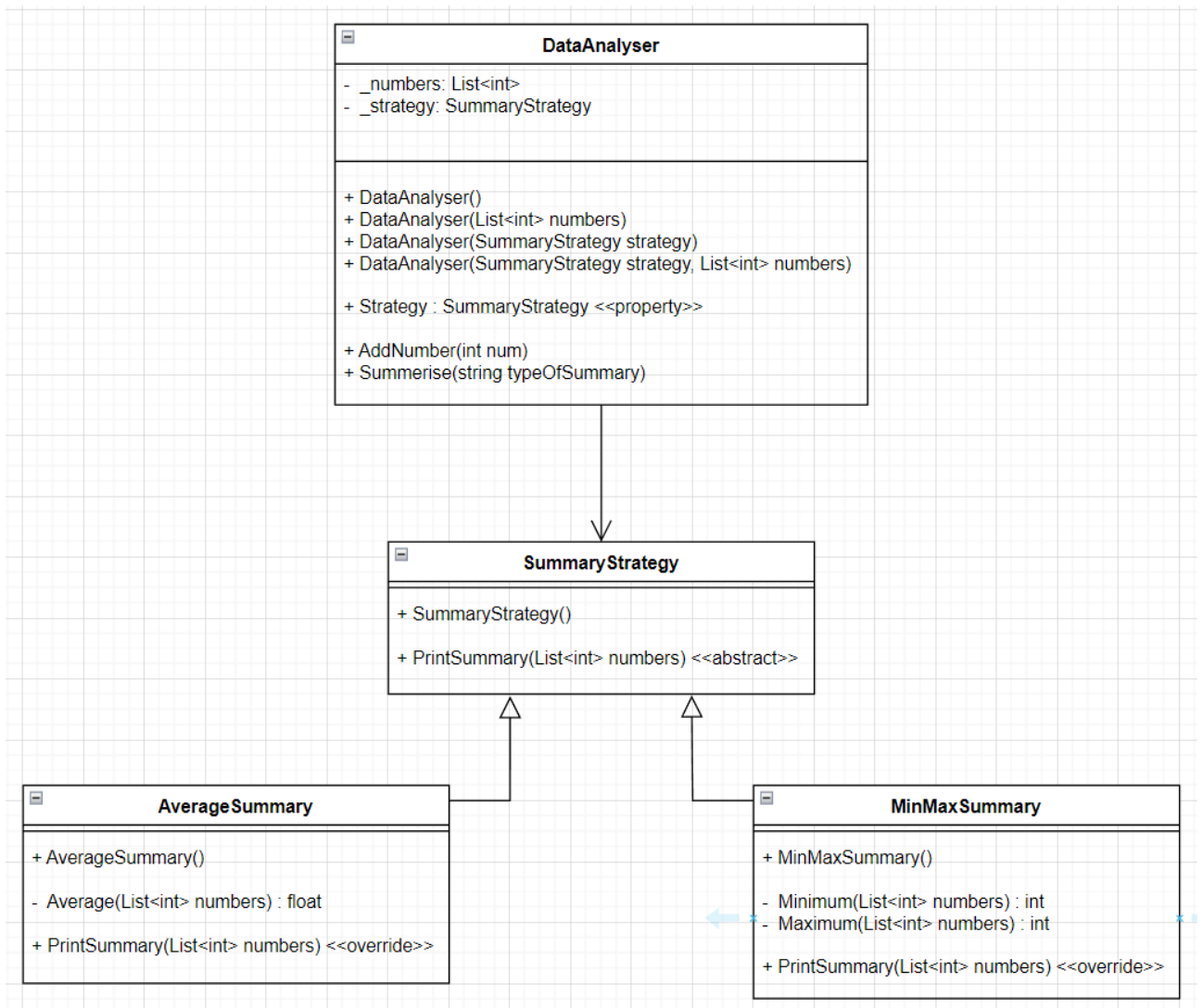
COS20007 OBJECT ORIENTED PROGRAMMING

---

## T1 - Semester Test

---

PDF generated at 01:35 on Friday 28<sup>th</sup> April, 2023



```
1  using System.Collections.Generic;
2
3  namespace HurdleTest
4  {
5      public class Program
6      {
7          // 7. Write a simple Main method to demonstrate how your new design works:
8          public static void Main(string[] args)
9          {
10             // a) Create a DataAnalyser object with a list containing the individual
11             ↪ digits of your student ID and the minmax summary strategy.
12             SummaryStrategy summaryStrategy = new MinMaxSummary();
13             List<int> numbers = new List<int>() { 4, 1, 8, 6, 7 };
14
15             DataAnalyser dataAnalyser = new DataAnalyser(summaryStrategy, numbers);
16
17             // b) Call the Summarise method.
18             dataAnalyser.Summarise();
19
20             // c) Add three more numbers to the data analyser.
21             dataAnalyser.AddNumber(2);
22             dataAnalyser.AddNumber(5);
23             dataAnalyser.AddNumber(9);
24
25             // d) Set the summary strategy to the average strategy.
26             summaryStrategy = new AverageSummary();
27             dataAnalyser.Strategy = summaryStrategy;
28
29             // e) Call the Summarise method.
30             dataAnalyser.Summarise();
31         }
32     }
```

```

1  using System.Reflection.Metadata;
2
3  namespace HurdleTest
4  {
5      public class DataAnalyser
6      {
7          private List<int> _numbers;
8          // 3. Modify DataAnalyser to have a private variable, _strategy, that is of
↪ the type SummaryStrategy
9          private SummaryStrategy _strategy;
10
11          // 5. Modify the DataAnalyser constructors to:
12
13          //      a) allow the strategy to be set through a parameter
14          public DataAnalyser(SummaryStrategy strategy, List<int> numbers)
15          {
16              _strategy = strategy;
17              _numbers = numbers;
18          }
19          public DataAnalyser(SummaryStrategy strategy) : this(strategy, new
↪ List<int>())
20          {
21          }
22
23          //      b) by default (i.e., if there are no parameters), set the strategy
↪ to the average strategy.
24          public DataAnalyser(List<int> numbers) : this(new AverageSummary(), numbers)
25          {
26          }
27          public DataAnalyser() : this(new AverageSummary(), new List<int>())
28          {
29          }
30
31          // 4. Add a public property for this new private variable.
32          public SummaryStrategy Strategy
33          {
34              get
35              {
36                  return _strategy;
37              }
38              set
39              {
40                  _strategy = value;
41              }
42          }
43
44          public void AddNumber(int num)
45          {
46              _numbers.Add(num);
47          }
48
49          // 6. Modify DataAnalyser Summarise method to use the currently stored
↪ strategy instead of relying on a string parameter

```

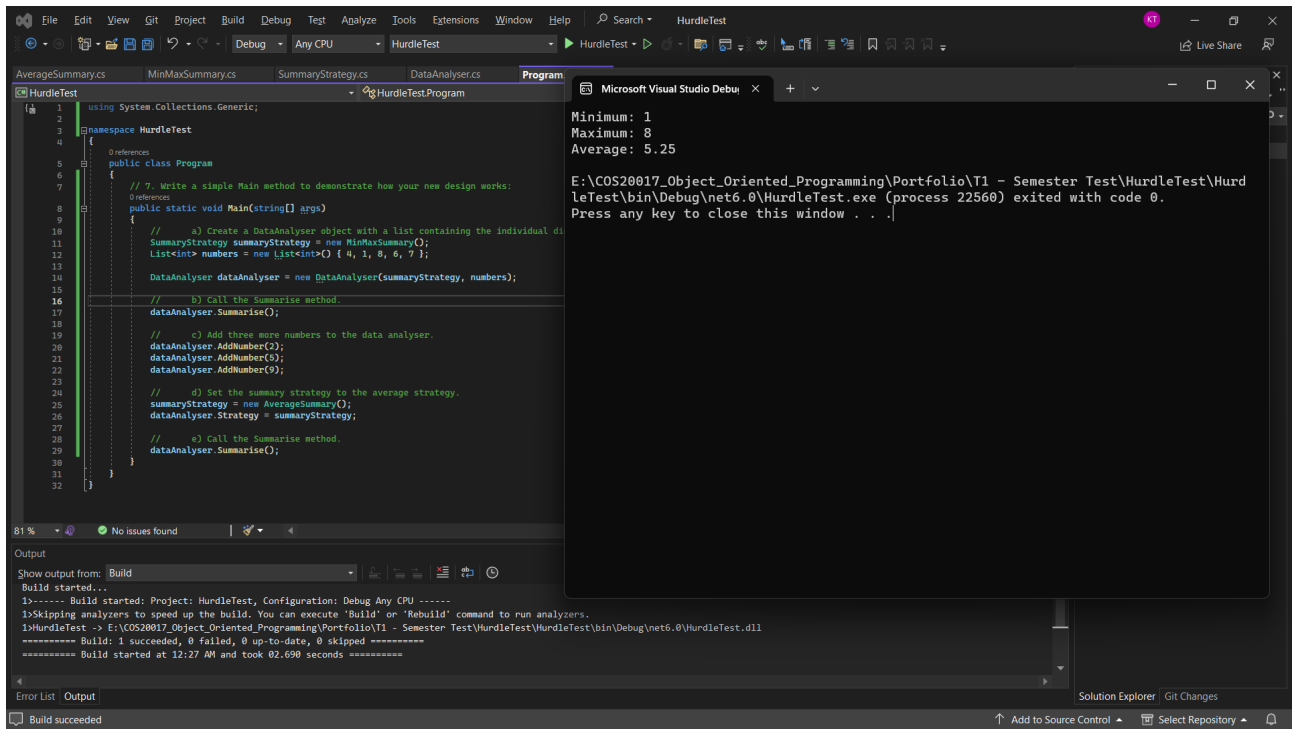
```
50     public void Summarise()  
51     {  
52         this.Strategy.PrintSummary(_numbers);  
53     }  
54 }  
55 }
```

```
1 namespace HurdleTest
2 {
3     // 1. Implement the SummaryStrategy abstract class according to the above design.
4     public abstract class SummaryStrategy
5     {
6         public SummaryStrategy()
7         {
8         }
9
10        public abstract void PrintSummary(List<int> numbers);
11    }
12 }
```

```
1 namespace HurdleTest
2 {
3     // 2. Redesign and implement the AverageSummary and MinMaxSummary classes to be
4     ⇨ child classes of the new SummaryStrategy class.
5     public class MinMaxSummary : SummaryStrategy
6     {
7         public MinMaxSummary()
8         {
9
10
11
12         private int Minimum(List<int> numbers)
13         {
14             int minNumber = numbers[0];
15             for (int i = 1; i < numbers.Count; i++)
16             {
17                 if (numbers[i] < minNumber)
18                 {
19                     minNumber = numbers[i];
20                 }
21             }
22
23             return minNumber;
24         }
25         private int Maximum(List<int> numbers)
26         {
27             int maxNumber = numbers[0];
28             for (int i = 1; i < numbers.Count; i++)
29             {
30                 if (numbers[i] > maxNumber)
31                 {
32                     maxNumber = numbers[i];
33                 }
34             }
35
36             return maxNumber;
37         }
38         public override void PrintSummary(List<int> numbers)
39         {
40             Console.WriteLine("Minimum: " + Minimum(numbers) + "\nMaximum: " +
41             ⇨ Maximum(numbers));
42         }
43     }
44 }
```

```
1 namespace HurdleTest
2 {
3     // 2. Redesign and implement the AverageSummary and MinMaxSummary classes to be
4     ⇐ child classes of the new SummaryStrategy class.
5     public class AverageSummary : SummaryStrategy
6     {
7         public AverageSummary()
8         {
9
10
11
12         private float Average(List<int> numbers)
13         {
14             float sum = 0;
15             foreach (int number in numbers)
16             {
17                 sum += number;
18             }
19
20             return (sum / (numbers.Count));
21
22         public override void PrintSummary(List<int> numbers)
23         {
24             Console.WriteLine("Average: " + this.Average(numbers));
25         }
26     }
```





Name: Trung Kien Nguyen

Student ID: 104053642

## TASK 2

1.

In general, Polymorphism is one of the fundamental principles of object-oriented programming that allows objects of different classes to be treated as if they were objects of the same class, through the use of a parent "abstract" class or an "interface". In other words, Polymorphism means "many forms", so it relates to any programming aspects where one thing can be used with many different types. In addition, Polymorphism is useful because it enables code reuse, flexibility, and extensibility in OOP programs.

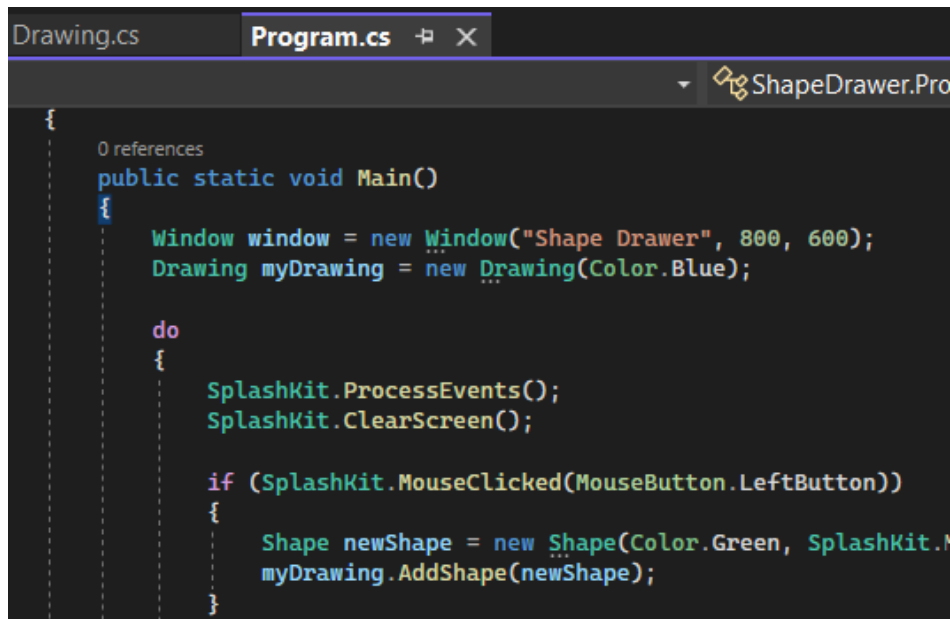
In the first task, this principle of Polymorphism was used in the two classes "AverageSummary" and "MinMaxSummary", which are inherited from the abstract class "SummaryStrategy". In the class "DataAnalyser", it is much simpler to use the private variable "\_strategy", which is of the type SummaryStrategy, to implement the functions of both the classes "AverageSummary" and "MinMaxSummary", instead of using two separate variables "\_avgSummariser", "\_minMaxSummariser" respectively as in the original UML Diagram.

2.

In Object-oriented programming, objects are the building blocks of a program. The principle of abstraction involves the process of hiding the internal details (e.g, some properties and methods) of an object from the external world, while the programmers can also provide access only for required properties and methods to the other classes, or other programs. Alternatively, classes in OOP are designed to encapsulate data and behavior, providing a high level of abstraction that enables programmers to build complex systems by combining simple and reusable components.

Abstraction principle can be used to create a boundary between the application and the client program. It allows programmers to create modular and reusable code that is much easier to update, maintain and extend.

**Example:** In the task 3.3 - Drawing Program – A Drawing Class: Considering the class "Drawing" containing the field List<Shape> "\_shapes" and the method "AddShape", in the "Main" method of the "Program" class, to add a new shape to draw when left-clicking, we only need to call "myDrawing.AddShape()", instead of implementing change directly in the field "\_shapes" of the object "myDrawing", which is unnecessary.



```
0 references
public static void Main()
{
    Window window = new Window("Shape Drawer", 800, 600);
    Drawing myDrawing = new Drawing(Color.Blue);

    do
    {
        SplashKit.ProcessEvents();
        SplashKit.ClearScreen();

        if (SplashKit.MouseClicked(MouseButton.LeftButton))
        {
            Shape newShape = new Shape(Color.Green, SplashKit.M
            myDrawing.AddShape(newShape);
        }
    }
}
```

3.

According to the original design as in the given UML diagram, if I do not create the abstract class “SummaryStrategy” as a parent class of the classes “AverageSummary” and “MinMaxSummary” (not using Polymorphism and Inheritance principles of OOP), it may come up with the duplication of code, making the whole program harder to update, maintain and extend, since I have to update the same code in multiple places if I want to make a change.

Moreover, if there are about 50 or more different summary approaches to choose from, the problem of duplication can become much worse. I would need to create 50 separate classes for each shape, which can lead to a lot of duplicated code and make my codebase much harder to read and understand, and much more complicated to work with.

In conclusion, by creating the abstract class “SummaryStrategy” as the parent class, along with defining the common properties and methods that all summary approaches should have in one place, which, in this case, is "PrintSummary()" method, I can avoid this duplication issue and save a great deal of time. This makes it easier to update, maintain and extend my codebase, even as the number of summary approaches increases.

## REFERENCES

[1] Rohan Vats. “Polymorphism In OOPS: What is Polymorphism [Detailed Explanation]” - UpGrad.

<https://www.upgrad.com/blog/polymorphism-in-oops/#:~:text=Polymorphism%20is%20the%20method%20in%20an%20object%2Doriented%20programmin%20language,the%20properties%20of%20the%20class>. September 22, 2022

[2] Pankaj. “Tutorial - What is Abstraction in OOPS?” - Digital Ocean.

<https://www.digitalocean.com/community/tutorials/what-is-abstraction-in-oops>. August 4, 2022