

SWINBURNE UNIVERSITY OF TECHNOLOGY

COS20007 OBJECT ORIENTED PROGRAMMING

T1 - Semester Test

PDF generated at 23:58 on Tuesday 9th May, 2023



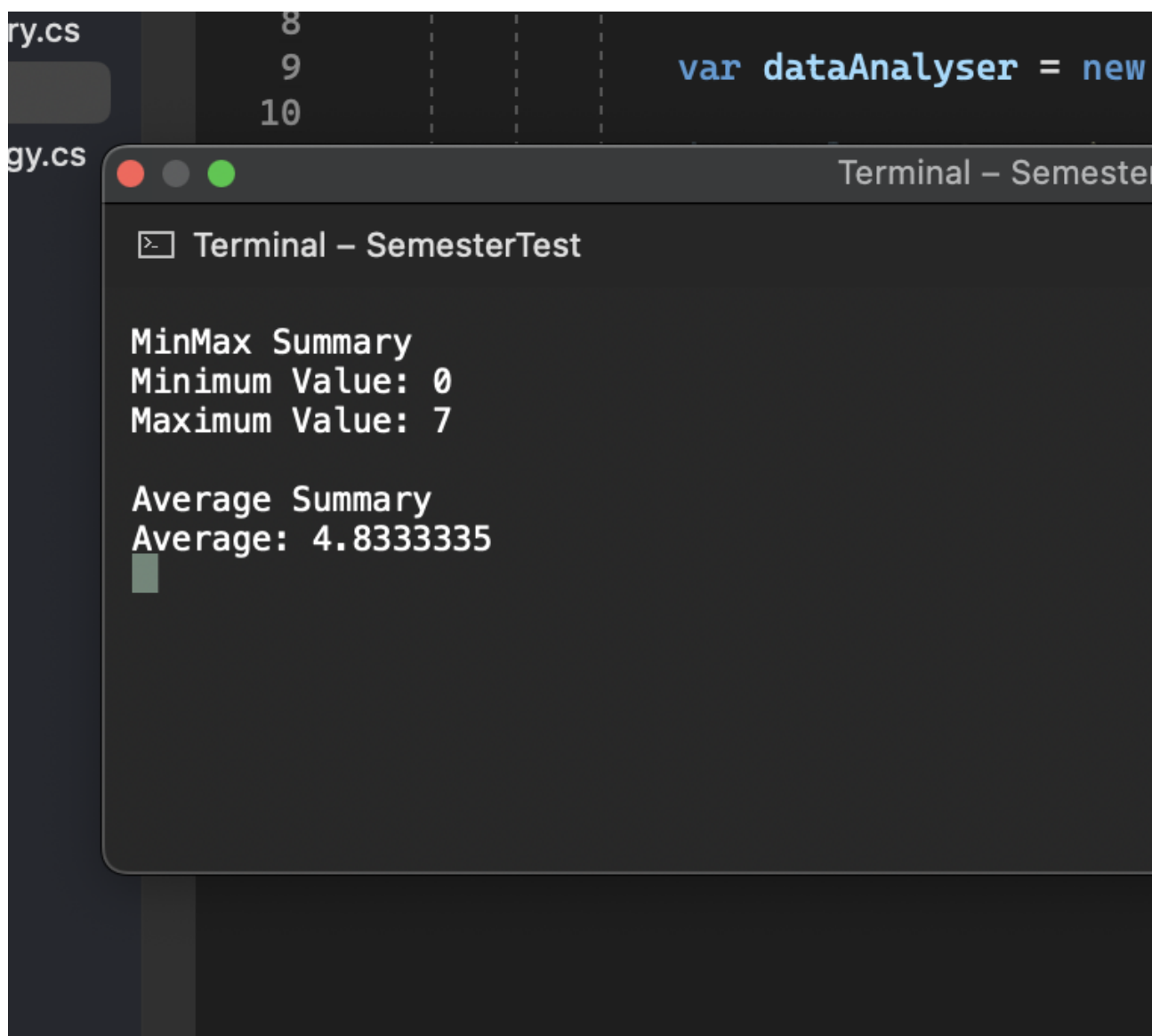
```
1 namespace SemesterTest
2 {
3     internal class Program
4     {
5         static void Main(string[] args)
6         {
7             Console.WriteLine("MinMax Summary");
8
9             DataAnalyser dataAnalyser = new DataAnalyser(new List<int>()
↪ {1,0,4,0,7,1,4,5,3}, new MinMaxSummary());
10
11             dataAnalyser.Summarise();
12
13             dataAnalyser.AddNumber(7);
14             dataAnalyser.AddNumber(5);
15             dataAnalyser.AddNumber(21);
16
17             Console.WriteLine("\nAverage Summary");
18             dataAnalyser.Strategy = new AverageSummary();
19
20             dataAnalyser.Summarise();
21         }
22     }
23 }
```

```
1  using System;
2  namespace SemesterTest
3  {
4      public class DataAnalyser
5      {
6          private List<int> _numbers;
7          private SummaryStrategy _strategy;
8
9          public SummaryStrategy Strategy
10         {
11             get
12             {
13                 return _strategy;
14             }
15             set
16             {
17                 _strategy = value;
18             }
19         }
20     }
21
22     public DataAnalyser() : this(new List<int>(), new AverageSummary()) { }
23
24     public DataAnalyser(SummaryStrategy strategy) : this(new List<int>(),
↪ strategy)
25     { }
26
27     public DataAnalyser(List<int> numbers, SummaryStrategy strategy)
28     {
29         _numbers = numbers;
30         _strategy = strategy;
31     }
32
33     public void AddNumber(int num)
34     {
35         _numbers.Add(num);
36     }
37
38     public void Summarise()
39     {
40         _strategy.PrintSummary(_numbers);
41     }
42
43 }
44 }
45
```

```
1  using System;
2
3  namespace SemesterTest
4  {
5      public abstract class SummaryStrategy
6      {
7          public abstract void PrintSummary(List<int> numbers);
8      }
9  }
```

```
1  using System;
2  namespace SemesterTest
3  {
4      public class MinMaxSummary : SummaryStrategy
5      {
6          private int Minimum(List<int> numbers)
7          {
8              int min = numbers[0];
9              foreach (int i in numbers)
10             {
11                 if (min > i)
12                 {
13                     min = i;
14                 }
15             }
16             return min;
17         }
18
19         private int Maximum(List<int> numbers)
20         {
21             int max = numbers[0];
22             foreach (int i in numbers)
23             {
24                 if (max < i)
25                 {
26                     max = i;
27                 }
28             }
29             return max;
30         }
31
32         public override void PrintSummary(List<int> numbers)
33         {
34             Console.WriteLine("Minimum Value: " + Minimum(numbers));
35             Console.WriteLine("Maximum Value: " + Maximum(numbers));
36         }
37     }
38 }
39
```

```
1  using System;
2
3  namespace SemesterTest
4  {
5      public class AverageSummary : SummaryStrategy
6      {
7          private float Average(List<int> numbers)
8          {
9              int total = 0;
10             foreach (int number in numbers)
11             {
12                 total += number;
13             }
14             float result = (float)total / numbers.Count;
15             return result;
16         }
17
18         public override void PrintSummary(List<int> numbers)
19         {
20             Console.WriteLine("Average: " + Average(numbers));
21         }
22     }
23 }
24
```



The image shows a code editor window in the background with a dark theme. It contains a C# file named `ry.cs` with line numbers 8, 9, and 10 visible. Line 9 contains the code `var dataAnalyser = new`. In the foreground, there is a terminal window titled "Terminal - SemesterTest". The terminal displays the output of a program, showing a "MinMax Summary" with a minimum value of 0 and a maximum value of 7, followed by an "Average Summary" with an average value of 4.8333335. A cursor is visible on the line following the average output.

```
ry.cs      8  
           9      var dataAnalyser = new  
           10  
gy.cs  
Terminal - SemesterTest  
Terminal - SemesterTest  
MinMax Summary  
Minimum Value: 0  
Maximum Value: 7  
  
Average Summary  
Average: 4.8333335  
█
```


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

- **Polymorphism** is a principle of OO Programming that allows objects of different classes to be treated as they were of the same class. This means that different objects can respond to the same message or method call in different ways. It is typically achieved through interfaces and method overriding.
- The principle of **polymorphism** is used in the implementation of the '*SummaryStrategy*' base class and its derived classes, '*AverageSummary*' and '*MinMaxSummary*'. The '*DataAnalyser*' class uses the '*SummaryStrategy*' base class as its type, allowing it to use objects of both derived classes in a polymorphic way. This means that '*DataAnalyser*' class can call the '*PrintSummary*' method on an object of either the '*AverageSummary*' or '*MinMaxSummary*' class, depending on the current strategy set for the object.

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

- **Abstraction** involves identifying the essential features of a system or object and representing them in a simplified form, while hiding or removing the non-essential details. This is achieved by creating classes and objects that provide a public interface that exposes only the essential features and hides the implementation details.

To determine what to hide or show in OOP, we need to analyse the problem domain and identify the key concepts and entities that are relevant to the problem. We then represent these concepts as classes and objects in our program, focusing on the essential attributes and behaviours.

For example: If we are creating a program to manage a library, we might identify the key concepts as books, borrowers, and the library itself. We would then create classes for each of these concepts, focusing on the essential attributes and behaviours of each class. For the Book class, this might include attributes such as checking out or returning the book. Similarly, for the Borrower class, we might include attributes such as the borrower's name and contact information, and behaviours such as checking out or returning books.

Once we have identified the essential features of each class, we can then hide the implementation details by making them private or protected. This ensures that users of the class can only interact with the essential features and are shielded from the internal workings of the class. This simplifies the programming process and makes it easier to maintain and extend the program over time.

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 issue with the original design in Task 1 was that it relied on string parameters to determine the type of summary approach to use, which makes it harder to maintain as new summary approaches are added. If we had 50 different summary approaches to choose from, we would need to add 50 different if statements to check the value of the parameter and implement the corresponding summary approach. By using the strategy pattern, we can encapsulate each summary approach in its own class and easily swap them in and out without having to modify the DataAnalyser class.

Reference:

Trivedi, J. (2023). *Understanding Polymorphism In C#*. [online] www.c-sharpcorner.com. Available at: <https://www.c-sharpcorner.com/UploadFile/ff2f08/understanding-polymorphism-in-C-Sharp/>.

www.knowledgehut.com. (n.d.). *Abstraction in C# Tutorial with Examples*. [online] Available at: <https://www.knowledgehut.com/tutorials/csharp/csharp-abstraction#:~:text=Abstraction%20in%20C%23->.