

# Programming281

Lecturer: Raymond Hood

## ASSIGNMENT 1

WERNER JANSE VAN RENSBURG (577930)

17/07/2023

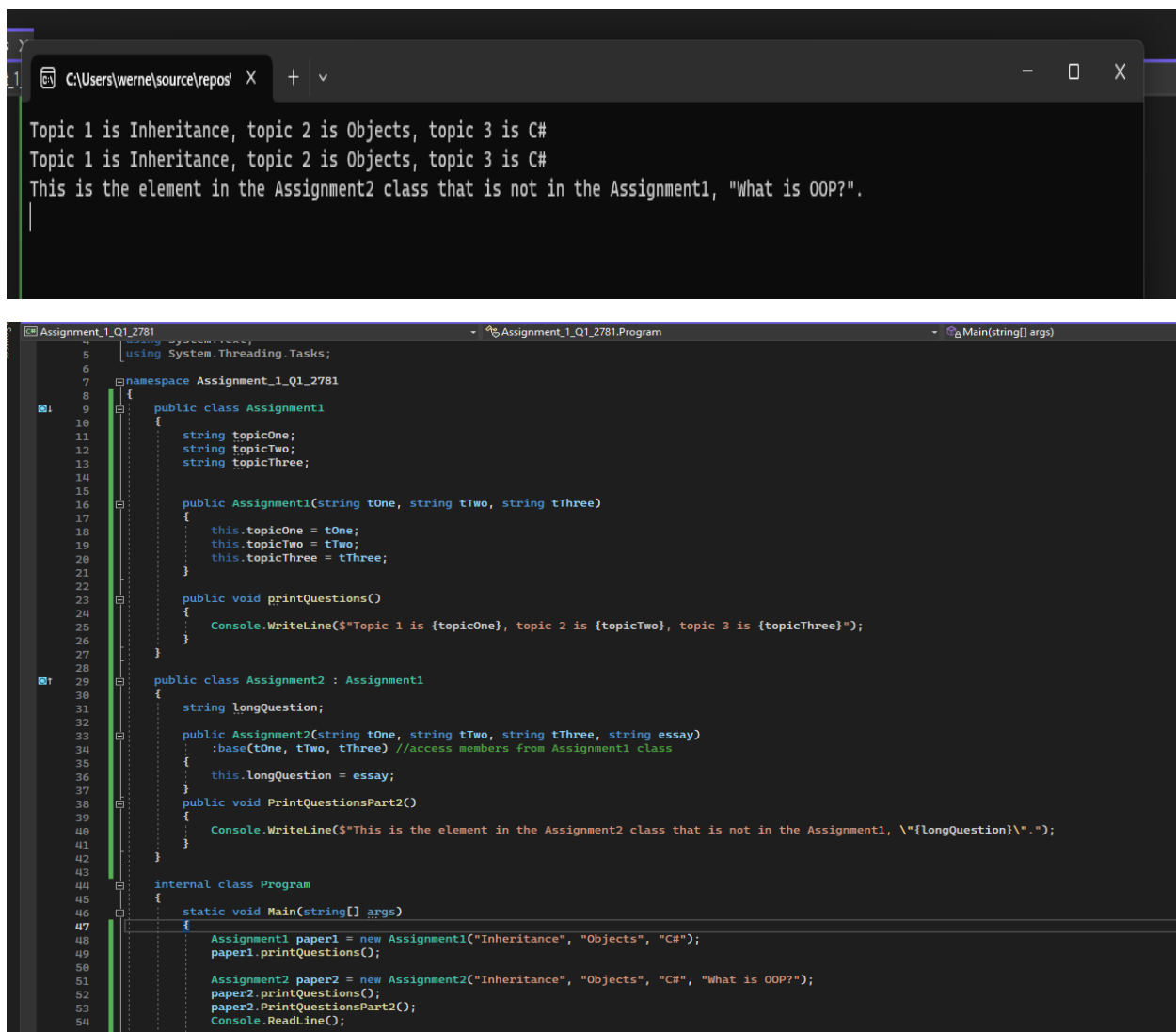
## Contents

Question 1:.....	2
a) Inheritance .....	2
b) Abstraction .....	3
c) Encapsulation .....	4
d) Polymorphism .....	6
Question 2:.....	6
Static Polymorphism: .....	6
Method Overloading: .....	6
Operator overloading:.....	7
Dynamic Polymorphism: .....	9
Virtual/ Overriding Method: .....	9
References.....	10
Figure 1 Inheritance Example.....	2
Figure 2 Abstraction Example .....	3
Figure 3 Creation of class Books using different encapsulation.....	4
Figure 4 Creating new object from the Book class and printing the name given .....	4
Figure 5 Output from class Program .....	5
Figure 6 Showing error due message .....	5
Figure 7 Method Overloading Example .....	6
Figure 8 Trying to get the total perimeter of obj 1 and obj 2 combined .....	7
Figure 9 Can get each one individually .....	7
Figure 10 Used operator overloading .....	8
Figure 11 Display total perimeter .....	8
Figure 12 Overriding method.....	9

## Question 1:

### a) Inheritance

According to (Taher, 2019) inheritance in general would be when something, this meaning an class in this case, gets something from someone else, meaning another class. (Taher, 2019) used the example when a child inherits his parents house one day. When can then use this knowledge to understand how inheritance in C#, using Object-Orientated-Programming (OOP), works. So when a class is created in C#, and we create another different class, but want to have the variables, methods or properties from the first class, we can use inheritance to allow the second class to access them. The first class would then be the parent class and the second class would then be the child class the receives from the parent. A concern that could be made is when you want a class only to inherit certain methods for example from another class. According to (Taher, 2019), the programmer can then make use of encapsulation to allow this. Encapsulation will be discussed later in this research assignment.



```
Topic 1 is Inheritance, topic 2 is Objects, topic 3 is C#
Topic 1 is Inheritance, topic 2 is Objects, topic 3 is C#
This is the element in the Assignment2 class that is not in the Assignment1, "What is OOP?".

using System.Threading.Tasks;

namespace Assignment_1_Q1_2781
{
    public class Assignment1
    {
        string topicOne;
        string topicTwo;
        string topicThree;

        public Assignment1(string tOne, string tTwo, string tThree)
        {
            this.topicOne = tOne;
            this.topicTwo = tTwo;
            this.topicThree = tThree;
        }

        public void printQuestions()
        {
            Console.WriteLine($"Topic 1 is {topicOne}, topic 2 is {topicTwo}, topic 3 is {topicThree}");
        }
    }

    public class Assignment2 : Assignment1
    {
        string longQuestion;

        public Assignment2(string tOne, string tTwo, string tThree, string essay)
            : base(tOne, tTwo, tThree) //access members from Assignment1 class
        {
            this.longQuestion = essay;
        }

        public void PrintQuestionsPart2()
        {
            Console.WriteLine($"This is the element in the Assignment2 class that is not in the Assignment1, \"{longQuestion}\".");
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            Assignment1 paper1 = new Assignment1("Inheritance", "Objects", "C#");
            paper1.printQuestions();

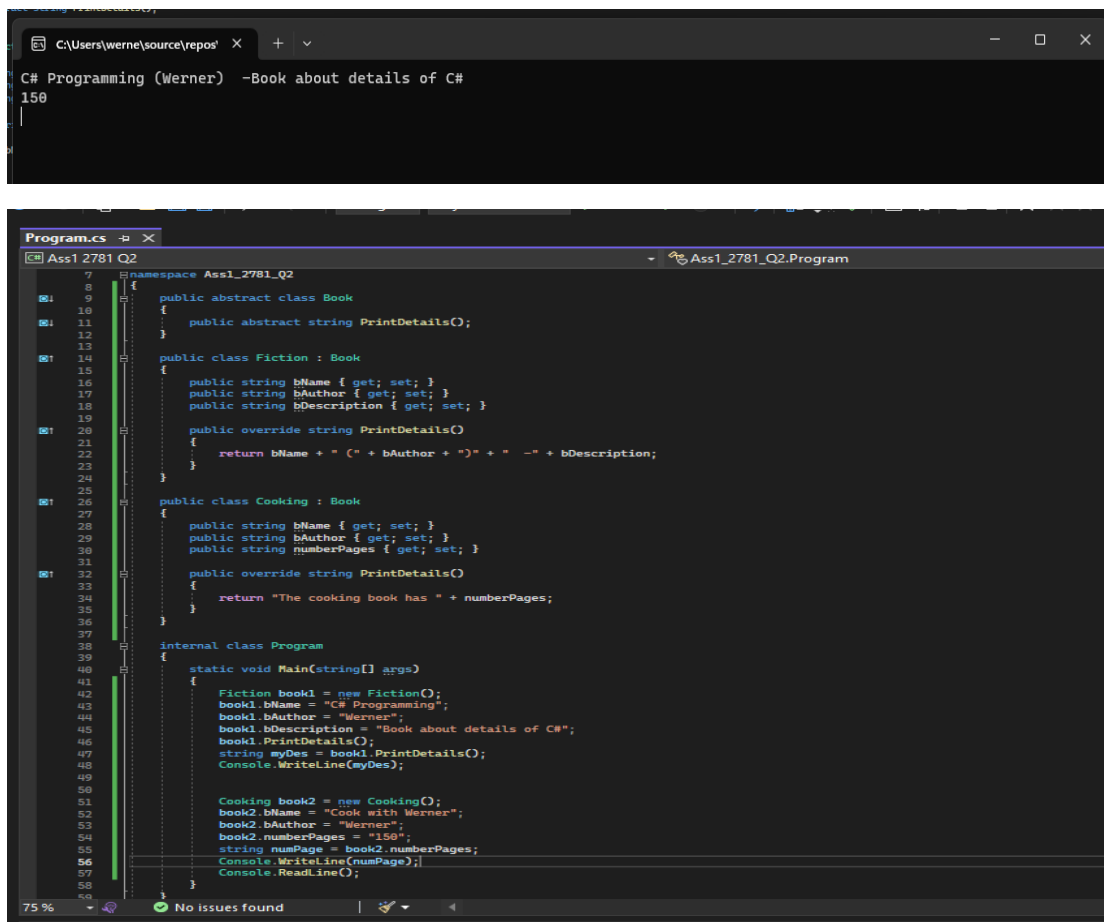
            Assignment2 paper2 = new Assignment2("Inheritance", "Objects", "C#", "What is OOP?");
            paper2.printQuestions();
            paper2.PrintQuestionsPart2();
            Console.ReadLine();
        }
    }
}
```

Figure 1 Inheritance Example

In the example created is a class called Assignment1 and gave it fields namely different topics for the an assignment. After it, a class called Assignment2 is created which inherits the topics from Assignment1 and then adds it own additional field, namely "longQuestion". We inherit from the class by first stating the class you are creating, then a ":" followed by the class it inherits from. To allow the Assignment 2 to access Assignment1's fields we use the "base:" keyword a Assignment2's constructor which fetches these fields above.

## b) Abstraction

According to (Taher, 2019) abstraction could be looked at as an abstract class in OOP. This means that the properties of this class for example methods are abstract as well and are declared with the abstract keyword. This class is only used by another class so it inherits from that certain class but overrides the abstract property or method. So the abstract method in the abstract method in the abstract class is declared but has no value. The programmer just uses the method in another class and is forced to if the parent class has a method marked as abstract. That is why the abstract class needs to be overwritten in the class the inherits the abstract class method. The reason to use abstract classes is to allow the use of one method in many classes that inherits from it.



```
C# Programming (Werner) -Book about details of C#
150

Program.cs
Ass1_2781_Q2
namespace Ass1_2781_Q2
{
    public abstract class Book
    {
        public abstract string PrintDetails();
    }

    public class Fiction : Book
    {
        public string bName { get; set; }
        public string bAuthor { get; set; }
        public string bDescription { get; set; }

        public override string PrintDetails()
        {
            return bName + " (" + bAuthor + ") " + "-" + bDescription;
        }
    }

    public class Cooking : Book
    {
        public string bName { get; set; }
        public string bAuthor { get; set; }
        public string numberPages { get; set; }

        public override string PrintDetails()
        {
            return "The cooking book has " + numberPages;
        }
    }

    internal class Program
    {
        static void Main(string[] args)
        {
            Fiction book1 = new Fiction();
            book1.bName = "C# Programming";
            book1.bAuthor = "Werner";
            book1.bDescription = "Book about details of C#";
            book1.PrintDetails();
            string myDes = book1.PrintDetails();
            Console.WriteLine(myDes);

            Cooking book2 = new Cooking();
            book2.bName = "Cook with Werner";
            book2.bAuthor = "Werner";
            book2.numberPages = "150";
            string numPage = book2.numberPages;
            Console.WriteLine(numPage);
            Console.ReadLine();
        }
    }
}
```

Figure 2 Abstraction Example

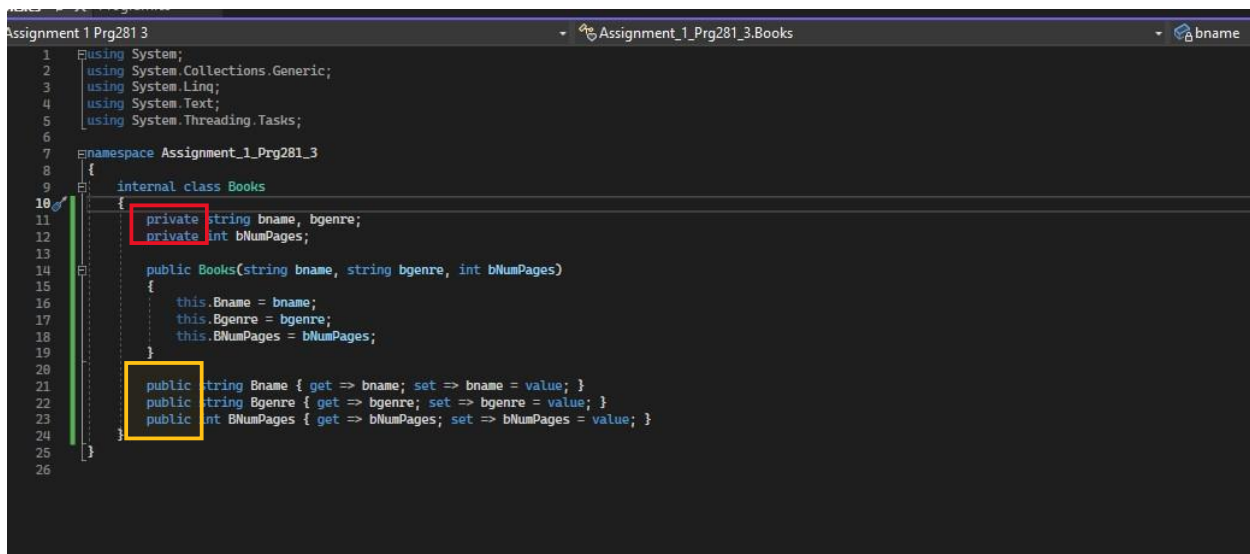
In the example above an abstract class *Book* is created with an abstract method called *PrintDetails()*. After this two classes of types of books are created namely, *Fiction* and *Cooking*. Both of these classes

inherit the abstract method from the book class, they are forced to. In each class the method is then overwritten to do something else. The method is then called in Main showing different outputs.

### c) Encapsulation

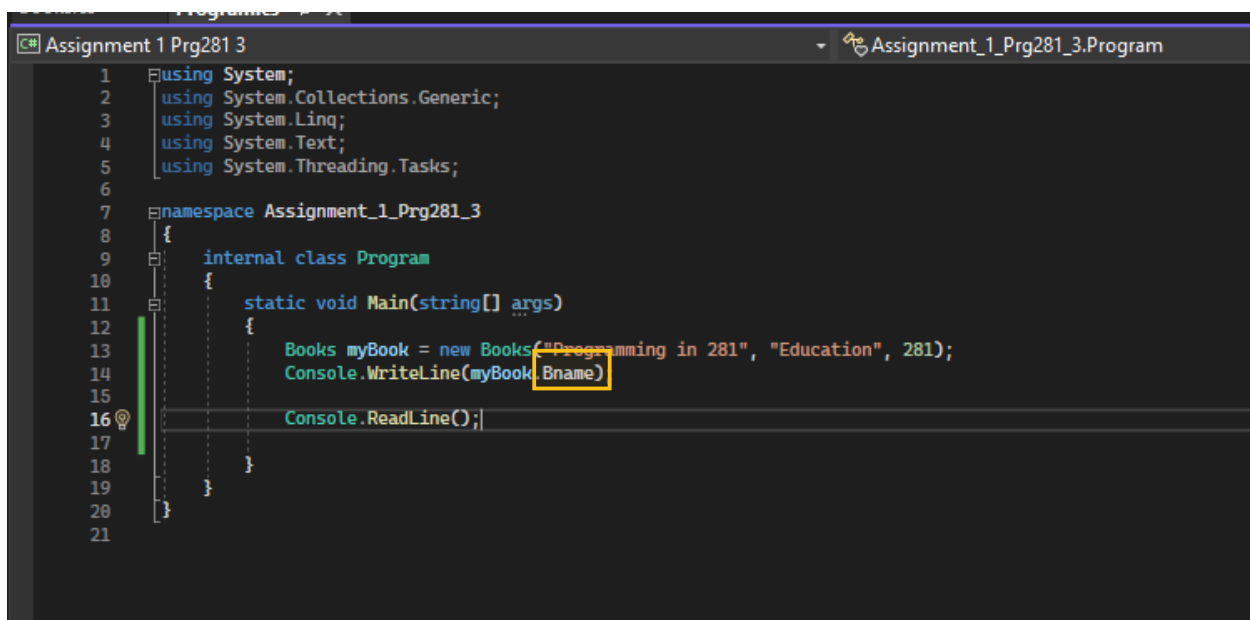
Encapsulation can be seen as when the visibility of something is chosen. In C# this can be done using access modifiers: Public, Private, Protected, Internal, Internal Protected.

We can use encapsulation to specify which classes can be seen by each other. This means which classes have access to another class. This a form of control and security in your code.



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Assignment_1_Prg281_3
8  {
9      internal class Books
10     {
11         private string bname, bgenre;
12         private int bNumPages;
13
14         public Books(string bname, string bgenre, int bNumPages)
15         {
16             this.Bname = bname;
17             this.Bgenre = bgenre;
18             this.BNumPages = bNumPages;
19         }
20
21         public string Bname { get => bname; set => bname = value; }
22         public string Bgenre { get => bgenre; set => bgenre = value; }
23         public int BNumPages { get => bNumPages; set => bNumPages = value; }
24     }
25 }
26
```

Figure 3 Creation of class Books using different encapsulation



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Assignment_1_Prg281_3
8  {
9      internal class Program
10     {
11         static void Main(string[] args)
12         {
13             Books myBook = new Books("Programming in 281", "Education", 281);
14             Console.WriteLine(myBook.Bname);
15
16             Console.ReadLine();
17         }
18     }
19 }
20
21
```

Figure 4 Creating new object from the Book class and printing the name given

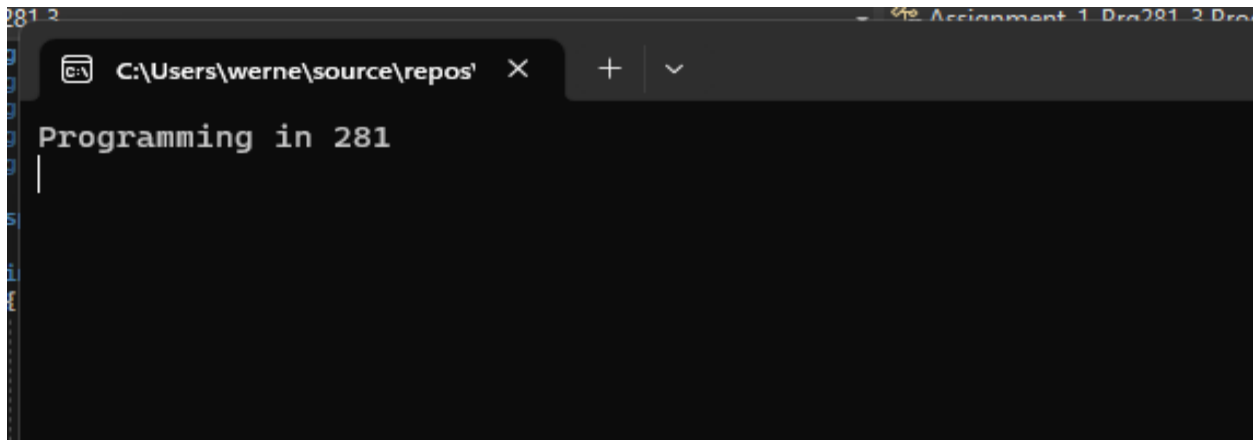


Figure 5 Output from class Program

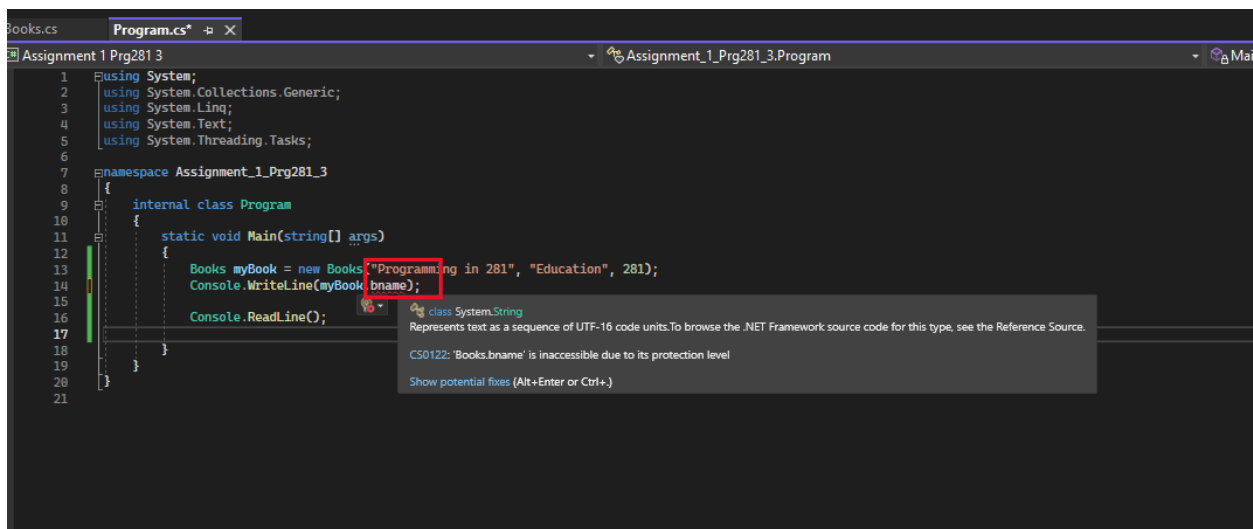


Figure 6 Showing error due message

In these examples above, encapsulation is explained. In figure 3 the red indicator shows that those fields are set to private. This means they are not accessible to the book object created in figure 6. That is why there is an error message displayed when trying to run the program. To give access to the inputs we create properties for each field in the class, as showed in figure 3. We make them public, given access to the Books constructor. The Books constructor is encapsulated to public allowing access to the Main to use this method to create new objects.

#### d) Polymorphism

The definition of polymorphism, according to (Turner, 2019) is anything that may take many different forms. An individual who plays several positions in a company is an example of polymorphism. As a result, one individual might take on several different responsibilities in this situation. Static and dynamic polymorphism are two separate forms of polymorphism that may be used in C# (Taher, 2019). In dynamic polymorphism, the role of a method would be defined at runtime as opposed to static polymorphism, where the role of a method (the form) is established by taking into account the compile time. An example of polymorphism would be in *Figure 2* when the *PrintDetails()* function is called exactly the same way in main but on different objects and display different outputs. So the function does multiple things, therefore we can say it has different roles.

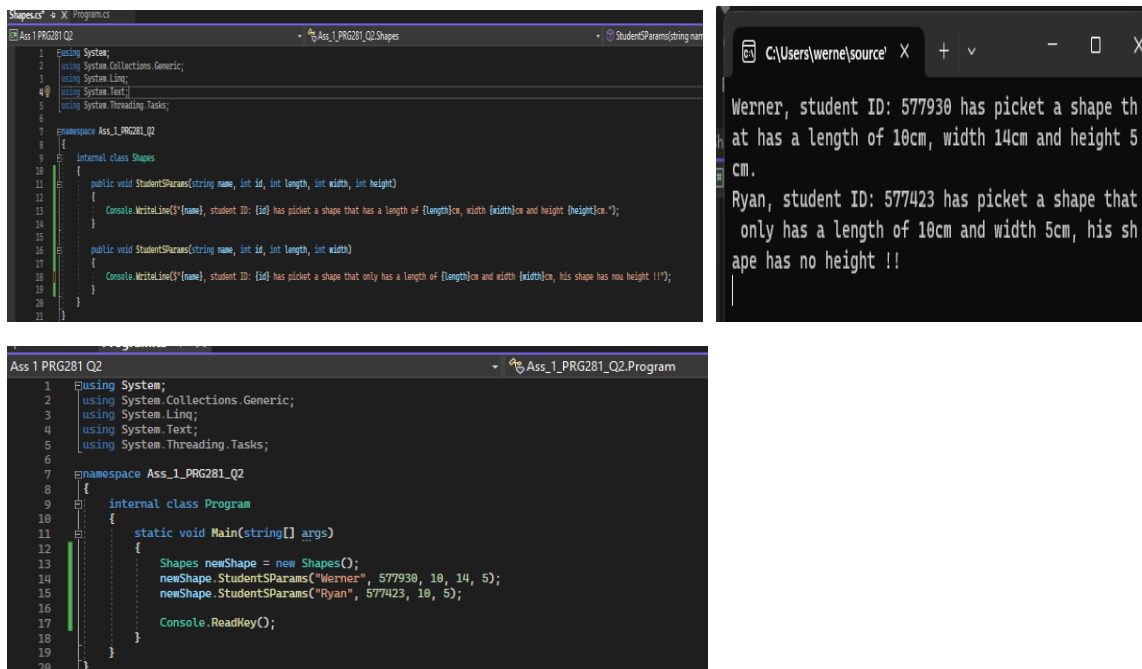
#### Question 2:

As (Taher, 2019) polymorphism can be derived into static-and dynamic polymorphism. It can then be divided into further sections to explain it even better (Trivedi, 2023). Static polymorphism can be divided into Method Overloading and Operator Overloading, while Dynamic polymorphism can be divided into Virtual/ Overriding Methods. We can then use these to implement it in OOP.

#### Static Polymorphism:

##### Method Overloading:

This is when a method has the same name in classes but the parameters of the method is different.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace Ass_1_PRG281_Q2
8 {
9     internal class Shapes
10     {
11         public void StudentSParams(string name, int id, int length, int width, int height)
12         {
13             Console.WriteLine($"{name}, student ID: {id} has picket a shape that has a length of {length}cm, width {width}cm and height {height}cm.");
14         }
15
16         public void StudentSParams(string name, int id, int length, int width)
17         {
18             Console.WriteLine($"{name}, student ID: {id} has picket a shape that only has a length of {length}cm and width {width}cm, his shape has no height !!");
19         }
20     }
21 }
22
```

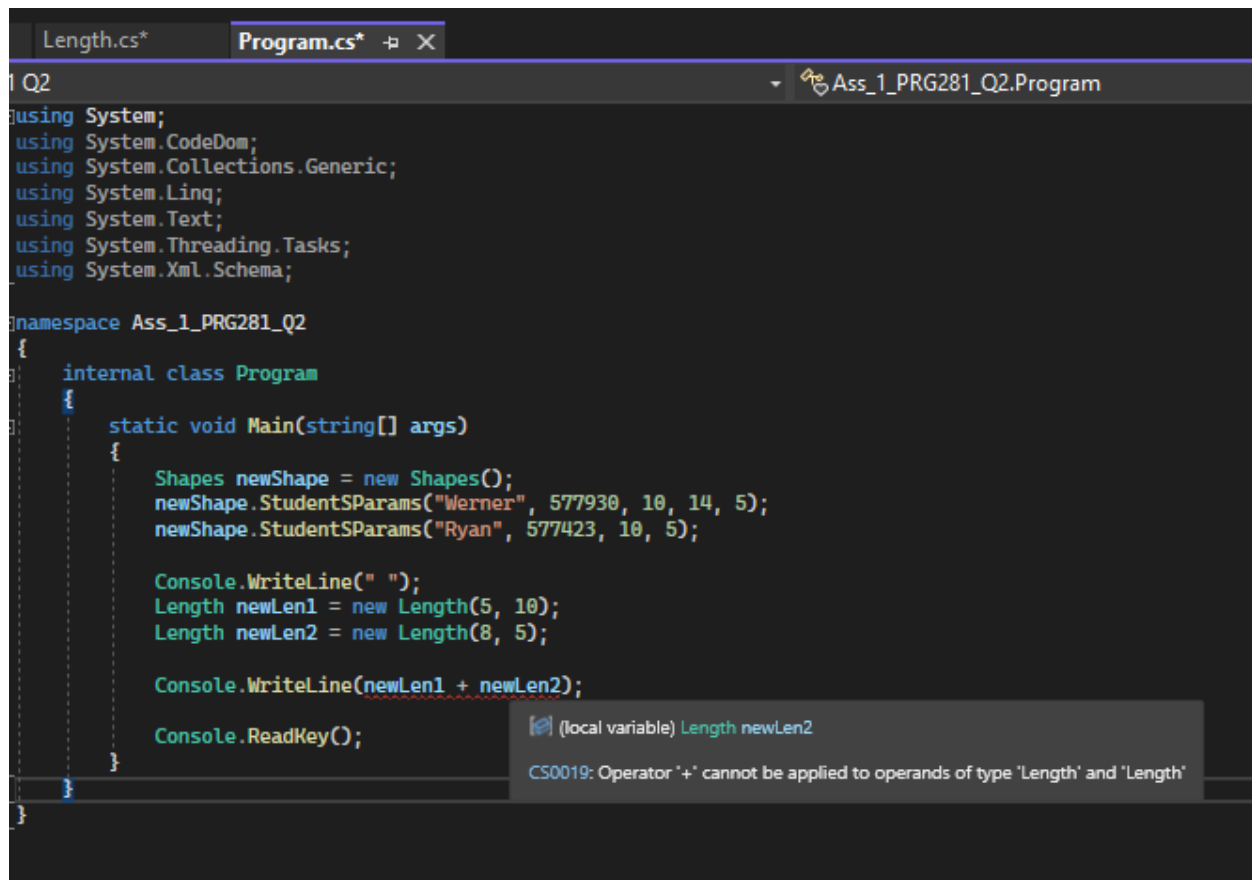
```
Werner, student ID: 577930 has picket a shape th
at has a length of 10cm, width 14cm and height 5
cm.
Ryan, student ID: 577423 has picket a shape that
only has a length of 10cm and width 5cm, his sh
ape has no height !!

```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace Ass_1_PRG281_Q2
8 {
9     internal class Program
10     {
11         static void Main(string[] args)
12         {
13             Shapes newShape = new Shapes();
14             newShape.StudentSParams("Werner", 577930, 10, 14, 5);
15             newShape.StudentSParams("Ryan", 577423, 10, 5);
16
17             Console.ReadKey();
18         }
19     }
20 }
21
```

Figure 7 Method Overloading Example

Operator overloading:



The screenshot shows a Visual Studio IDE with two tabs: 'Length.cs\*' and 'Program.cs\*'. The 'Program.cs' tab is active, displaying the following code:

```
1 Q2
using System;
using System.CodeDom;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Schema;

namespace Ass_1_PRG281_Q2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Shapes newShape = new Shapes();
            newShape.StudentSParams("Werner", 577930, 10, 14, 5);
            newShape.StudentSParams("Ryan", 577423, 10, 5);

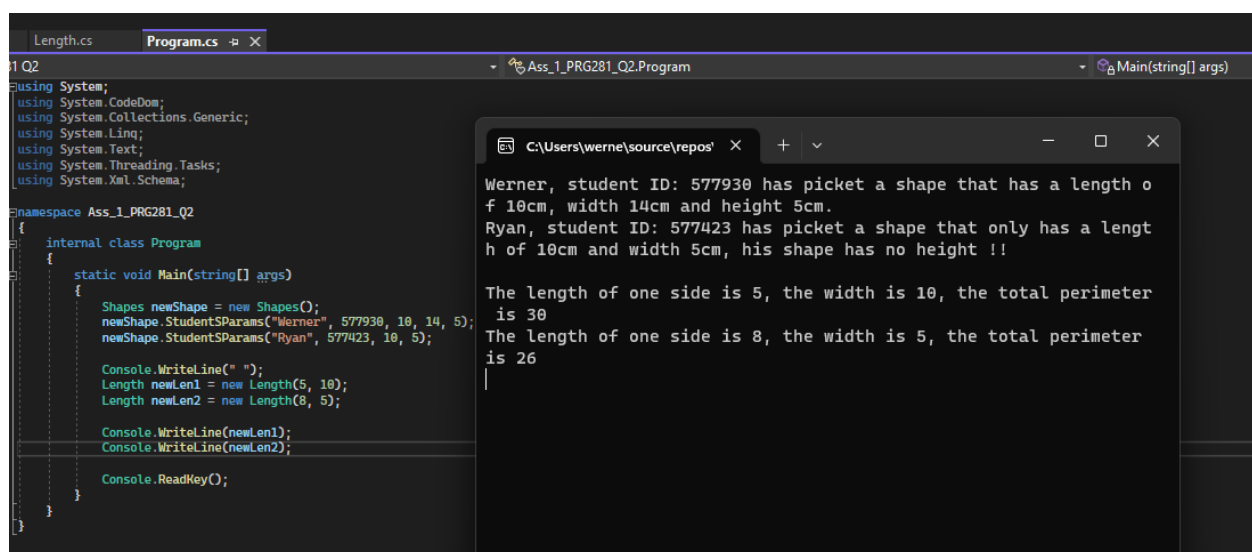
            Console.WriteLine(" ");
            Length newLen1 = new Length(5, 10);
            Length newLen2 = new Length(8, 5);

            Console.WriteLine(newLen1 + newLen2);

            Console.ReadKey();
        }
    }
}
```

A tooltip is visible over the '+' operator in the line `Console.WriteLine(newLen1 + newLen2);`, showing the text: `(local variable) Length newLen2`. Below the code, a red squiggly line under the '+' operator indicates an error. The error message at the bottom of the window reads: `CS0019: Operator '+' cannot be applied to operands of type 'Length' and 'Length'`.

Figure 8 Trying to get the total perimeter of obj 1 and obj 2 combined



The screenshot shows the same Visual Studio IDE as Figure 8, but with a console window open. The code is identical. The console window displays the following output:

```
Werner, student ID: 577930 has picket a shape that has a length o
f 10cm, width 14cm and height 5cm.
Ryan, student ID: 577423 has picket a shape that only has a lengt
h of 10cm and width 5cm, his shape has no height !!

The length of one side is 5, the width is 10, the total perimeter
is 30
The length of one side is 8, the width is 5, the total perimeter
is 26
```

Figure 9 Can get each one individually



```

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

namespace Ass_1_PRG281_Q2
{
    internal class Length
    {
        private int myLen, myWidth;
        private int perimeter;

        public Length(int myLen, int myWidth)
        {
            this.MyLen = myLen;
            this.MyWidth = myWidth;
            this.Perimeter = (myLen * 2) + (myWidth * 2);
        }

        public int MyLen { get => myLen; set => myLen = value; }
        public int MyWidth { get => myWidth; set => myWidth = value; }
        public int Perimeter { get => perimeter; set => perimeter = value; }

        public override string ToString()
        {
            return $"The length of one side is {myLen}, the width is {myWidth}, the total perimeter is {Perimeter}";
        }

        public static Length operator +(Length obj1, Length obj2)
        {
            int totLen = obj1.myLen + obj2.myLen;
            int totWidth = obj1.myWidth + obj2.myWidth;
            return new Length(totLen, totWidth);
        }
    }
}

```

Figure 10 Used operator overloading

```

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Ass 1 PRG281 Q2
[20756] Ass 1 PRG281 Q2.exe Lifecycle Events Thread: [8904] Main Thread Stack Frame: Ass 1 PRG281 Q2 Length.operator +
Program.cs
1 PRG281 Q2
1 using System;
2 using System.CodeDom;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7 using System.Xml.Schema;
8
9 namespace Ass_1_PRG281_Q2
10 {
11     internal class Program
12     {
13         static void Main(string[] args)
14         {
15             Shapes newShape = new Shapes();
16             newShape.StudentSParams("Werner", 577930, 10, 14, 5);
17             newShape.StudentSParams("Ryan", 577423, 10, 5);
18
19             Console.WriteLine(" ");
20             Length newLen1 = new Length(5, 10);
21             Length newLen2 = new Length(10, 5);
22
23             Console.WriteLine(newLen1 + newLen2);
24
25             Console.ReadKey();
26         }
27     }
28
29 }
30
31
Werner, student ID: 577930 has picket a shape that has a length of 10cm, w
idth 14cm and height 5cm.
Ryan, student ID: 577423 has picket a shape that only has a length of 10cm
and width 5cm, his shape has no height !!

The length of one side is 15, the width is 15, the total perimeter is 60

```

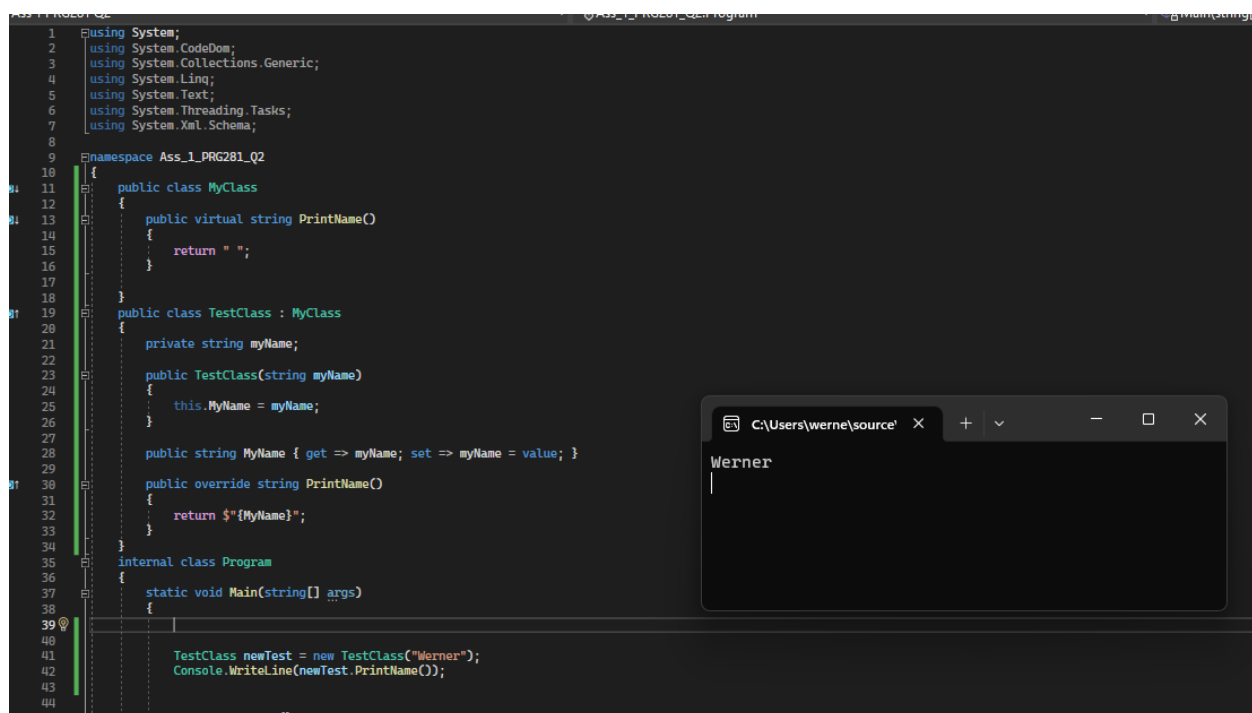
Figure 11 Display total perimeter

In the above example we want to get the total circumference of two shape objects created. Logically you want to add the circumference of object 1 and add it to object 2. As seen in figure 8, it is not possible to add objects together in main by just simply adding them together as if they are normal integers. That is why a Length operator is created. The parameters of the operator are obj 1 and object 2. There is a "+" sign in front of the brackets as well, as seen in figure 10. This whole operator, that performs like a function, basically replaces the whole expression in main when two Length objects need to be added together. So now when objects *newLen1* and *newLen2* need to be added together, the Length operator is called which overloads the normal "+" expression and returns a new object, which would be the total circumference of the two shapes created. The return value then replaces the Perimeter variable in the *ToString()* method.

### Dynamic Polymorphism:

#### Virtual/ Overriding Method:

According to (Trivedi, 2023) method overriding can be done using inheritance. This means the class which has a method and the class inheriting the method would use the same name for the method.



```
1 using System;
2 using System.CodeDom;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7 using System.Xml.Schema;
8
9 namespace Ass_1_PRG281_Q2
10 {
11     public class MyClass
12     {
13         public virtual string PrintName()
14         {
15             return " ";
16         }
17     }
18
19     public class TestClass : MyClass
20     {
21         private string myName;
22
23         public TestClass(string myName)
24         {
25             this.MyName = myName;
26         }
27
28         public string MyName { get => myName; set => myName = value; }
29
30         public override string PrintName()
31         {
32             return $"{MyName}";
33         }
34     }
35
36     internal class Program
37     {
38         static void Main(string[] args)
39         {
40
41             TestClass newTest = new TestClass("Werner");
42             Console.WriteLine(newTest.PrintName());
43
44             Console.ReadLine();
45         }
46     }
47 }
```

Figure 12 Overriding method

Above the *MyClass* that contains a virtual method called *PrintName()*. The *TestClass* inherits from this class meaning that it inherits the method as well. The method then gets overridden and returns something new as stated by the *TestClass*. The method is therefore a form of polymorphism due to the fact that they look the same but do different things.

## References

Taher, R., 2019. *Hands-On Object-Oriented Programming with C#*. Birmingham: Packt Publishing.

Trivedi, J., 2023. *C# Corner*. [Online]

Available at: <https://www.c-sharpcorner.com/UploadFile/ff2f08/understanding-polymorphism-in-C-Sharp/>

[Accessed 17 August 2023].

Turner, R., 2019. *C#: The Ultimate Beginner's Guide to Learn C# Programming Step by Step*. s.l.:Publishing Factory.