

OOP assignment

by Katarzyna Pindel

FILE	12033_KATARZYNA_PINDEL_OOP_ASSIGNMENT_226878_1403433182.D OCX (155.68K)		
TIME SUBMITTED	17-FEB-2020 10:00AM (UTC+0000)	WORD COUNT	1707
SUBMISSION ID	120213860	CHARACTER COUNT	9311

2020

Object Oriented Programming

KATARZYNA PINDEL

WEB DEVELOPMENT

CONTENTS

1. Orientated Programming	2
2. The difference between a class and an object	2
2.1 A class.....	2
2.2 An object	3
3. Data encapsulation	5
4. Interfaces	5
Bibliography	8

1. ORIENTATED PROGRAMMING

The concept of object-oriented programming appeared as early as in the 1960s thanks to the Simula 67 [1] language, designed by scientists from Oslo to run a simulation. However, the idea of object-oriented programming owes its popularization to the SmallTalk language [2]. The mid-1980s was a time when object-oriented programming became the dominant technique - mainly due to C++. At that time, it was also possible to create objects in many other languages. C# is a language that allows you to write applications only object-oriented [3]. This is the most advantage - object-oriented programming is much more logical and allows for better organization of the code, which is very important in large projects.

The idea of object-oriented programming is becoming more and more common, as applications are becoming more and more complicated and complex. The division of applications into modules is no longer enough, such code should be divided into individual classes. Older programming languages, such as C, do not allow object-oriented programming - this possibility only appeared in the successor of the said language - C++. The C# language, is fully object-oriented, so for a programmer it is even necessary to know what objectivity is and how to create your own classes.

By writing the program in a manner consistent with OOP, we define abstract objects that have attributes (i.e. data) and methods (i.e. functions / procedures). An object-oriented program is one that involves the creation of such objects and then mechanisms that enable communication between these objects. Generally, such programming is to imitate the operation of the natural world. The object-oriented approach is intended to fulfil the overarching goal - to create a higher abstraction that can be practically used.

2. THE DIFFERENCE BETWEEN A CLASS AND AN OBJECT

1.1 2.1 A CLASS

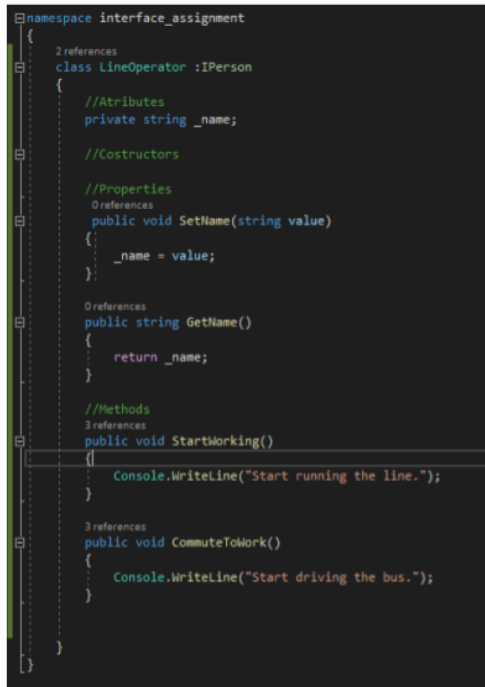
Writing a program in accordance with the OOP philosophy involves defining and implementing appropriate classes, as well as creating objects from them and manipulating them. So, class is a key concept. A class is a complex type of variables, consisting of fields that store data and have methods that perform programmed actions. The class defines all its elements, i.e. determines how its methods work and what its individual fields mean. These functions, (methods) belong to some module (because only inside the modules you can place functions). The class has to be declared only once [4]. Its definition specifies the fields it contains and the method declarations that the class will have. The primary purpose of a class is to hold data/information. This is achieved with attributes which are also known as data members [5]. The attributes define many operations on a data that is in form of an object.

The class divided into two parts:

- definition, in which we define the fields of the class and enter prototypes of its methods,
- implementation, placed in the module, which is simply the code of previously defined methods

This system not only works extremely well, but also implements one of the postulates of object-oriented programming, which is hiding unnecessary details. These details will be the code of each method, which knowledge is not needed to use the class. The class groups certain variables and functions together. The class is equivalent to an object with certain features. When creating a class, we also create a new data type.

The class definition syntax looks like this:



```
namespace interface_assignment
{
    2 references
    class LineOperator : IPerson
    {
        //Attributes
        private string _name;

        //Constructors

        //Properties
        0 references
        public void SetName(string value)
        {
            _name = value;
        }

        0 references
        public string GetName()
        {
            return _name;
        }

        //Methods
        3 references
        public void StartWorking()
        {
            Console.WriteLine("Start running the line.");
        }

        3 references
        public void CommuteToWork()
        {
            Console.WriteLine("Start driving the bus.");
        }
    }
}
```

Figure 1- Creating a class LineOperator

1.2 2.2 AN OBJECT

Objects are usable instances of classes [6]. The object is a physical representative of the class in memory [7]. All operations are performed on objects. Each of these objects has its own fields and methods (many objects of one class can have the same fields storing different values). The object is a physical representative of the class in memory. He is a copy of the class. We create it like a variable because really it is a variable [4]. All operations are performed on objects. The object is an instance of the class. What does this mean in practice? The class is total, and the object is a specific case. The class should contain fields (information) and methods (actions that can be performed). The object belongs to a given class and contains more specific information and methods. To illustrate this with an example - a class is, e.g., Person. We know about her that she has first and last name fields, but in the case of a class we are not talking about the values of these fields - we just have such fields. The class is human as a species, a being. An object, in turn, is a specific person - one who has a full name. It should be noted that you can create multiple objects that are instances of the same class. Then each of these objects has its own fields and methods (many objects of one class can have the same fields storing different values).

The object definition syntax looks like this:

```
LineOperator lineOperator=new LineOperator();
```

Or

Object personList and new person added to that list:

```
namespace interface_assignment
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            LineOperator lineOperator = new LineOperator();
            Manager manager = new Manager();

            List<IPerson> personList = new List<IPerson>();
            personList.Add(lineOperator);
            personList.Add(manager);

            foreach(IPerson person in personList)
            {
                person.CommuteToWork();
                person.StartWorking();
            }

            Console.ReadLine();
        }
    }
}
```

Figure 2- Creating an object

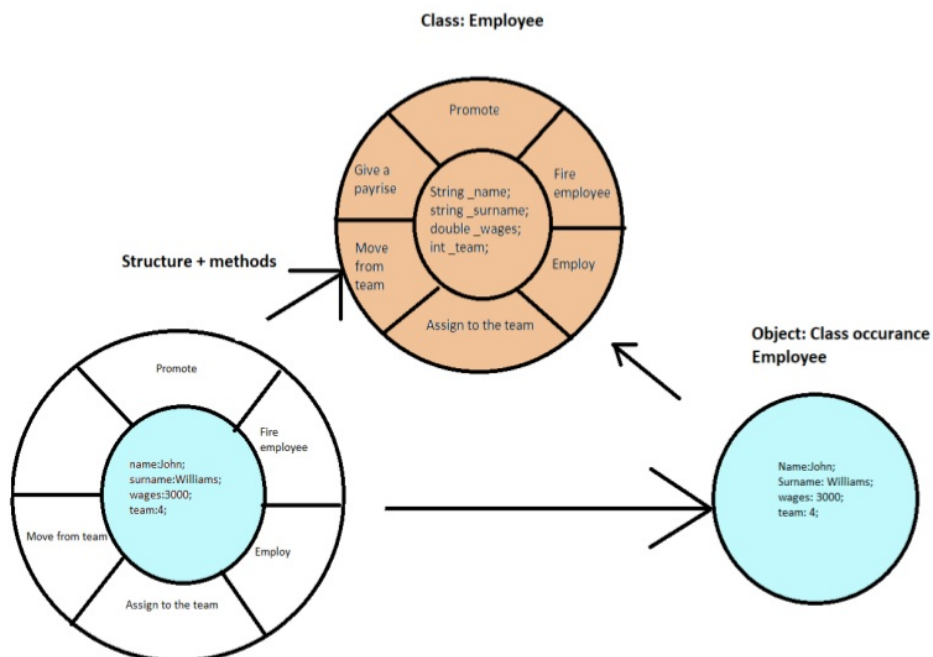


Figure 3- Structure of a class and an object

3. DATA ENCAPSULATION

The process of combining code and manipulating into single unit is commonly referred to as encapsulation. The security is provided to protect data stored in these fields against unauthorized or at least unreasonable access. [8]. Encapsulation is a tool enabling (or "taking") access rights to structure components. So, it's hiding the implementation, encapsulation. Ensures that an object cannot change the internal state of other objects in an unexpected way. Only the internal methods of the object are entitled to change its state, so we have three sections: private (invisible to other classes), public (will allow you to change and read the values in the fields (then other classes can use these methods)) and protected (makes the class element visible, but for classes that you inherit it) [9]. Let's assume that an application for managing people. When programming such a system, we do not want to share fields of the Person class with other classes in the project, but we want to enable these fields to be changed by these classes. For this purpose, encapsulation was introduced:

```
class Person
{
    //Attributes
    private int _staffNo;
    private string _name;
    private string _surname;
    private string _gender;
}
```

Figure 4- Data encapsulation

Why we limit access to data at all? :

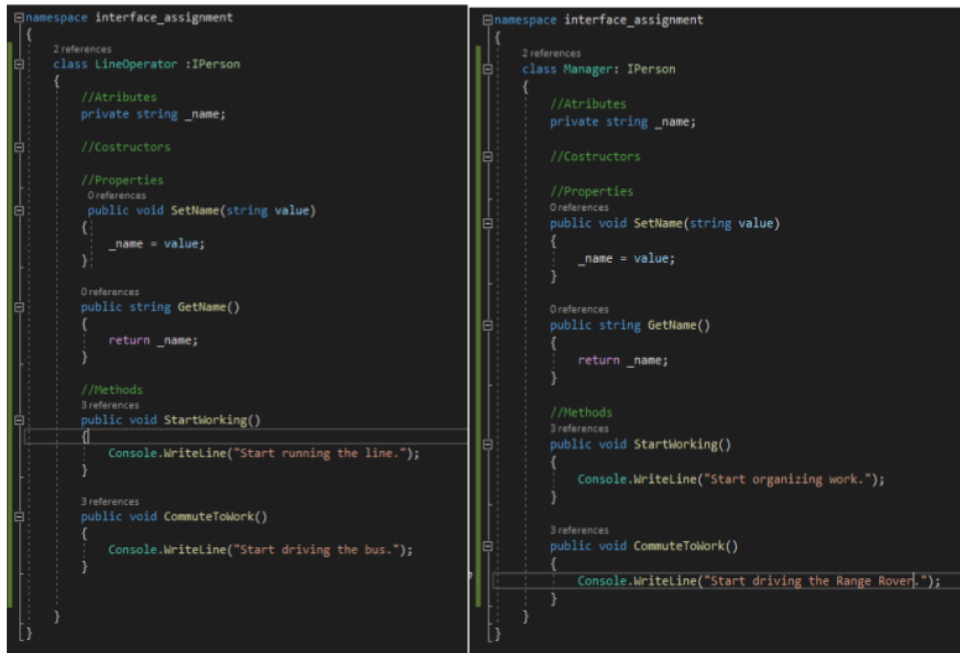
- We have more control over fields and methods,
- If we want to enter a new logic for a given field or change the old one, then we change it only within one class. We change the code of methods operating on these fields, and all places in the project using these methods remain unchanged,
- Classes can be easily implemented as read-only - if they have only the read method, or write-only (if write only) - if they only have the write method,
- We improve data security.

4. INTERFACES

The interface is defined as a kind of pattern that all implementing classes must follow, include behaviour from multiple sources in a class [10]. The interface specifies "what" should be done and the inheriting class "how" should be done. The interface defines the properties, methods, and events (with empty bodies) [11] that are components of this interface. It is worth remembering that the interface contains only member declarations, not their definition. The definition is the role of the class that the interface implements. This usually helps maintain a consistent structure that inheriting classes must represent. "Abstractive classes" (as the interface can be called) can serve the same purpose to some extent, however they are used mainly when only a few methods have to be declared in the base class and derived classes implement these methods. Interfaces can actually slightly resemble classes and certainly connect strongly with classes. The easiest interface in C # will

probably be explained by comparing it to the computer ports e.g. USB port. The USB port itself determines how the data is transferred, what the socket looks like, etc. But what specific device we will connect the USB port doesn't care, nor does it care what happens next with the transferred data, it only says how they are to be transferred between computer and device. It is only important that the plug fits. And the interfaces work in a similar way - they determine the way of communication between classes, but what exactly are the classes of the interface itself no longer cares. In addition, just as a device can support several different ports, the same class can implement several interfaces.

Interface construction in C #:



```
namespace interface_assignment
{
    2 references
    class LineOperator : IPerson
    {
        //Attributes
        private string _name;

        //Constructors

        //Properties
        0 references
        public void SetName(string value)
        {
            _name = value;
        }

        0 references
        public string GetName()
        {
            return _name;
        }

        //Methods
        3 references
        public void StartWorking()
        {
            Console.WriteLine("Start running the line.");
        }

        3 references
        public void CommuteToWork()
        {
            Console.WriteLine("Start driving the bus.");
        }
    }
}

namespace interface_assignment
{
    2 references
    class Manager : IPerson
    {
        //Attributes
        private string _name;

        //Constructors

        //Properties
        0 references
        public void SetName(string value)
        {
            _name = value;
        }

        0 references
        public string GetName()
        {
            return _name;
        }

        //Methods
        3 references
        public void StartWorking()
        {
            Console.WriteLine("Start organizing work.");
        }

        3 references
        public void CommuteToWork()
        {
            Console.WriteLine("Start driving the Range Rover.");
        }
    }
}
```

Figure 5-Class LineOperator with interface IPerson

Figure 6- Class Manager with interface IPerson

```

namespace interface_assignment
{
    5 references
    interface IPerson
    {
        3 references
        void StartWorking();
        3 references
        void CommuteToWork();
    }
}

```

Figure 5-Interface with 2 functions

```

namespace interface_assignment
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            LineOperator lineOperator = new LineOperator();
            Manager manager = new Manager();

            List<IPerson> personList = new List<IPerson>();
            personList.Add(lineOperator);
            personList.Add(manager);

            foreach(IPerson person in personList)
            {
                person.CommuteToWork();
                person.StartWorking();
            }

            Console.ReadLine();
        }
    }
}

```

Figure 6-Interface list

```

Start driving the bus.
Start running the line.
Start driving the Range Rover.
Start organizing work.

```

Figure 7-Program output

As it be observed the word class has been replaced by the word interface. Inside the interface we have method declarations. Declarations without implementation because the interface is only used to communicate which method, we can use but does not specify what it will do next. In addition, no keyword that specifies the visibility of the method is used. This is because by definition all methods in the interface will be implemented as public, it makes no sense that the method used to communicate with the world would be private and unavailable outside of a particular class.

2 BIBLIOGRAPHY

- [1] O.-J. Dahl, "The Roots of Object-Oriented Programming:," in *Software Pioneers*, Berlin, 2002, pp. 78-90.
- [2] S. Cook, "Languages and object-oriented programming," *Software Engineering Journal*, vol. 1, no. 2, 1986.
- [3] E. International, C# Language, 5th ed., Geneva, 2017.
- [4] W3Schools, "What is the difference between Classes and objects," [Online]. Available: <https://www.w3schools.in/java-questions-answers/difference-between-classes-objects/>.
- [5] W3schools, "Java Objects and classes," [Online]. Available: <https://www.w3schools.in/java-tutorial/object-classes/>.
- [6] Microsoft, "Object-Oriented Programming (C#)," 2015. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/object-oriented-programming>.
- [7] adobe, "OBJECT-ORIENTED PROGRAMMING CONCEPTS: OBJECTS AND CLASSES," [Online]. Available: <https://www.adobe.com/devnet/actionscript/learning/oop-concepts/objects-and-classes.html>.
- [8] S. Deshpande, "Collaboration of Object Oriented Programming and SoftwareDevelopment," *International Research Journal of Engineering and Technology*, vol. 03, no. 09, 2016.
- [9] M. Yaiser, "OBJECT-ORIENTED PROGRAMMING CONCEPTS: ENCAPSULATION," 31 October 2011. [Online]. Available: <https://www.adobe.com/devnet/actionscript/learning/oop-concepts/encapsulation.html>.
- [10] Microsoft, "Interfaces (C# Programming Guide)," 08 August 2018. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/>.
- [11] w3schools, "C# Interface," [Online]. Available: https://www.w3schools.com/cs/cs_interface.asp.

OOP assignment

GRADEMARK REPORT

FINAL GRADE

81 / 100

GENERAL COMMENTS

Instructor

Part 1 :

Class (10): 10

Encapsulation (10): 10

Interface (10): 10

Part 2:

Class (10): 10

Logic (10): 10

Polymorphism (30): 29

Part 3:

Written (10): 0

Code (10): 2

Comments

=====

Very nice discussion of concepts.

Part 2:

Excellent coding. Could have provided a few more comments but you have added comments where you have interesting code.

I think you slightly overcomplicated the implementation by using a number of clever techniques. A simple brute force approach would have been OK for this assignment. But I find it hard to criticise you for doing the research and discovering these techniques in the first place.

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8



Comment 1

next?

PAGE 9
