

## Introduction to Object-Oriented Programming (I)

### Objectives

At the end of this activity, you should be able to:

- Identify some advantages and disadvantages of Object-Oriented Programming.
- Define simple classes in C# with attributes and methods.
- Build libraries.
- Use a library from an application.

### Motivation

Any program is composed of the data and the set of operations or functions that can be done on these data. In *functional or structured programming*, the data and functions are defined separately. But in the real world, the objects are characterized by the actions that can be done with them. For example, what makes a screen interesting is that we can draw on it, clear it, set the background color, and so on. *Object-Oriented Programming (OOP)* uses the concept of class to better match the real world. A class allows connecting the data to the operations that can be done with them. An object (i.e., Peter's computer screen) is a particular instance of a class (i.e., the screen class) and can be accessed only through the operations defined for this class.

### What is modular programming?

Modular programming consists of splitting the application in different modules that can be programmed and compiled separately. Then, the application is built as a main program that uses the previously programmed modules.

In the case of large programs, it is necessary to split them into modules. When the application is developed by a team, each programmer can be in charge of one part and produce one or more modules. The different modules produced by all the programmers are joined later to build the full application.

Another interesting property of modular programming is that we can reuse parts of previous programs in a new one, avoiding having to rewrite them. This is what we do when we "reuse" some of the standard libraries (or modules) of the C# language. For example, we use the class `Console` and its methods `WriteLine` and `ReadLine` (already defined in the namespace `System`) to read and write on the screen.

Next, we summarize some of the main advantages of modular programming:

- helps to obtain a good organization of the code
- helps to distribute the work between several programmers (each programmer can work in his module independently)
- helps to do the maintenance of the application (we can improve one module by just modifying the code of that module)

## What is an object?

An object is a data structure (whose data fields are called *attributes*) and a set of functions and procedures (called *methods*) to operate with them. *The methods are the only way to operate on the information of the object.*

We have already used objects in the past, but they were very simple objects: they did not have methods, they only had attributes, and we were able to access these attributes because they were declared as “public”. *From now, the attributes will remain “private”, and we will implement as many public methods as needed to manipulate them.*

## My first object

To work with objects in C#, first, we have to define the **class**, that is, the template that specifies the characteristics (attributes and methods) of the object. Defining a class is like creating a new data type. Look at the following definition of the class `CPerson`:

```
using System;

namespace PeopleLib
{
    public class CPerson
    {
        // ATTRIBUTES

        int age;
        float height;
        string name;

        // METHODS

        public void SetAge(int a)
        {
            this.age = a;
        }

        public int GetAge()
        {
            return (this.age);
        }
    }
}
```

The class `CPerson` contains three **attributes** (age, height, and name) and two **methods** (SetAge and GetAge).

The “**this**” keyword refers to the current instance of the class.

Next, you have an example of the main program using the class `CPerson`:

```

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

using PeopleLib;

namespace PeopleConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            int ageIn, ageOut;

            // Create a new object named "myPerson" of class CPerson
            CPerson myPerson = new CPerson();

            // Ask the user for a value and set the age of "myPerson"
            Console.Write("Write the age of the person: ");
            ageIn = Convert.ToInt32(Console.ReadLine());
            myPerson.SetAge(ageIn);

            // Get the age of "myPerson" and write it on the screen
            ageOut = myPerson.GetAge();
            Console.WriteLine("He/she is " + ageOut + " years old.");

            Console.ReadKey();
        }
    }
}

```

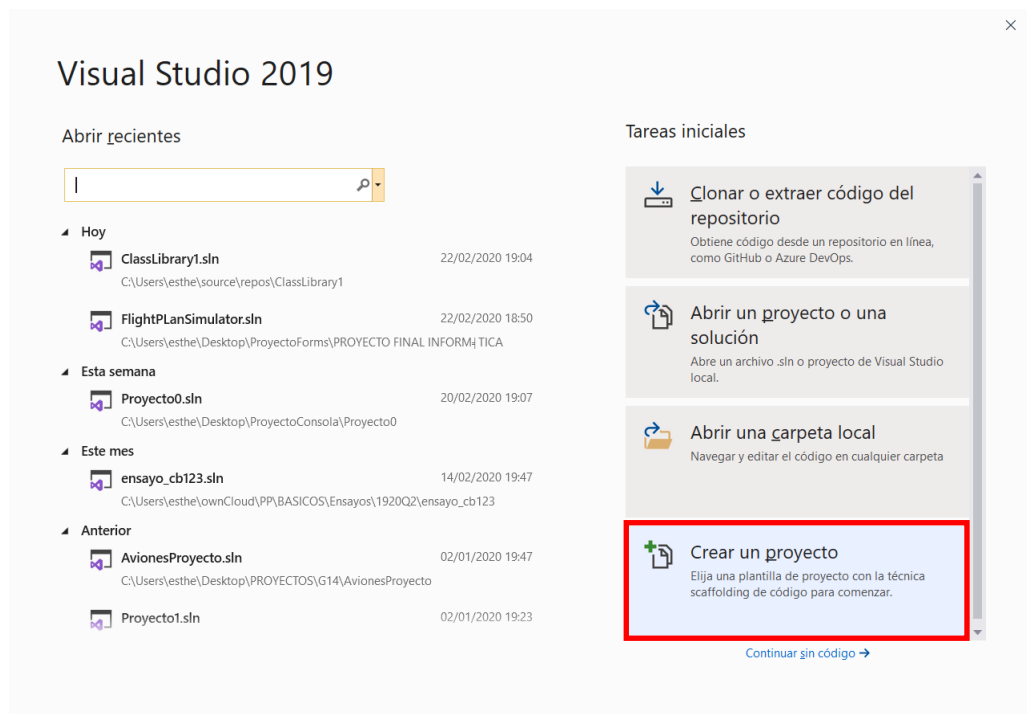
Note that to make use of the class `CPerson`, we must write the **using** directive "using `PeopleLib`" at the beginning of the program. This is because the class `CPerson` is defined in the namespace `PeopleLib` (you can check this in the class definition code).

Then, the main program can declare so many **objects** (or instances) of the class `CPerson` as needed. Instances of classes are created by using the **new** operator. In our example, `CPerson` is the class and `myPerson` is an object of that class.

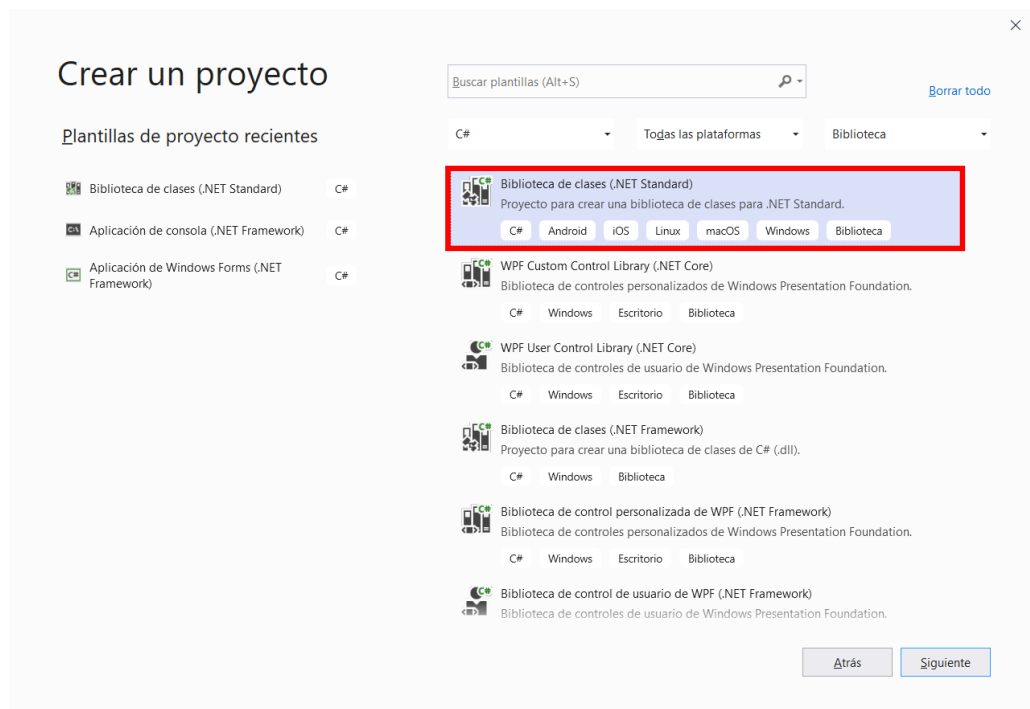
The program asks the user for a value (`ageIn`) and uses the method `SetAge` to assign the value introduced by the user (`ageIn`) to the attribute `age` of the object `myPerson`. Next, the program uses the method `GetAge` to get the value of the attribute `age` of the object `myPerson`, and writes it in the screen. Note the "dot" syntax to call a method on an object: `myPerson.SetAge(ageIn)` calls the method `SetAge` on the object `myPerson`.

1. Now, it is time to learn how to build a solution (we will name it *PracticaGuiadaPOO*) in Visual C# that includes two projects: a class library (*PeopleLib*) with the definition of the class `CPerson` and a console application (*PeopleConsole*) with the main program that uses it.

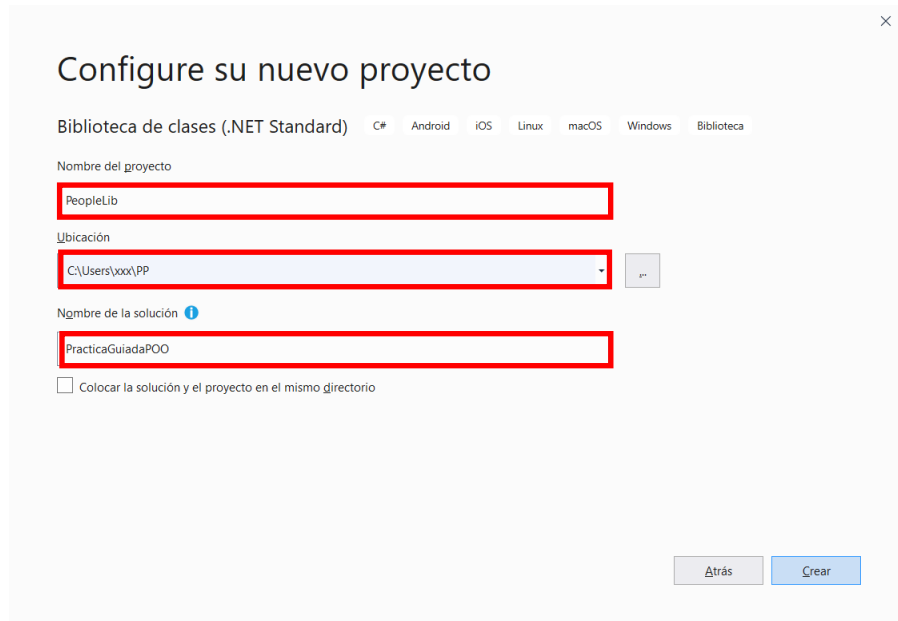
### 1.1. Open your **Visual Studio 2019** and select “Create new project”



### 1.2. Select the type of project that we want to add first. For example, to create the **class library**, select "Class Library (.NET Standard)".

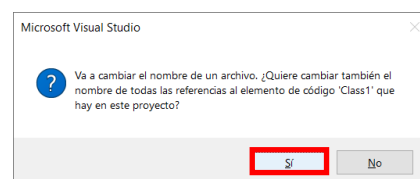
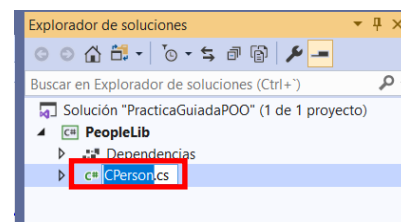
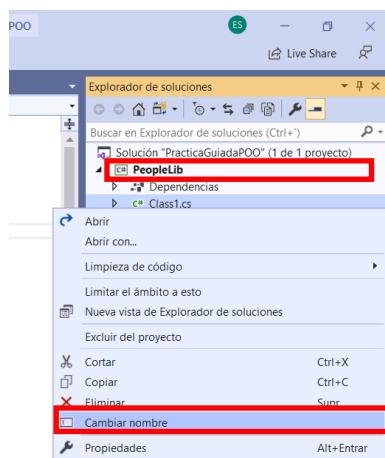


- 1.3. Indicate the name of the **project** (*PeopleLib*), the location (computer folder where you want the solution to be saved), and the name of the **solution** (*PracticaGuiadaPOO*).

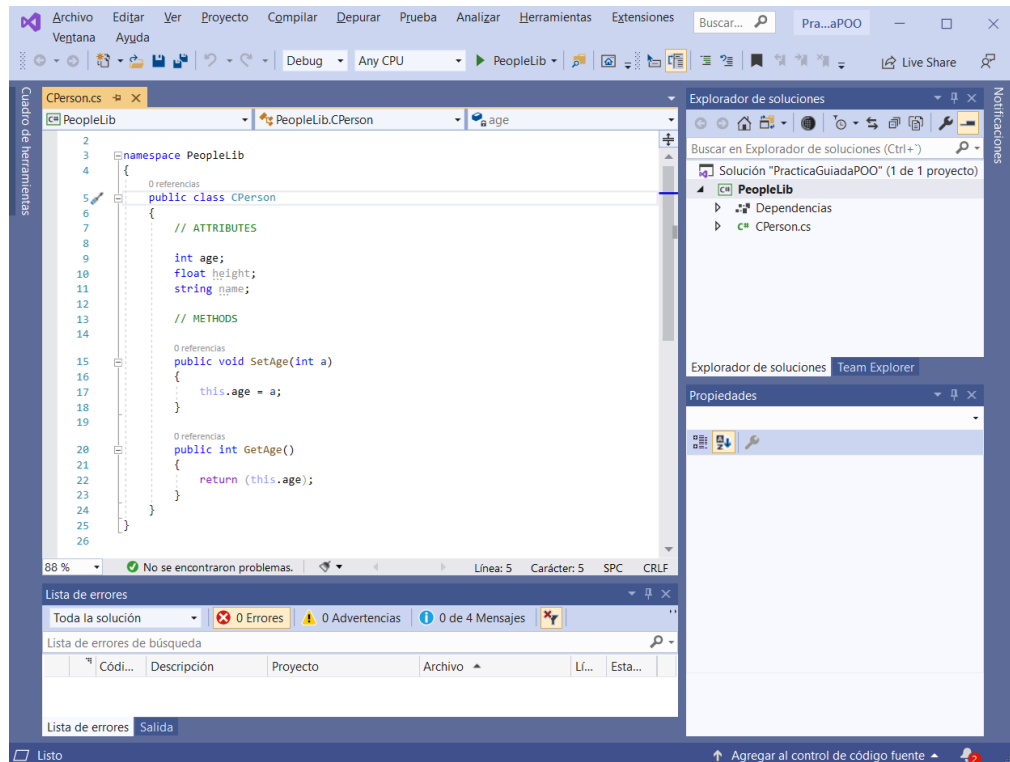


- 1.4. Notice the **Solution Explorer** on the right side of your screen (if you can't see the Solution Explorer, click its entry on the **View** menu at the top of Visual C#). The Solution Explorer allows you to see and manipulate the file structure of your application. In this case, you have one project called *PeopleLib* with a file *Class1.cs* that is the default template for the definition of a class.

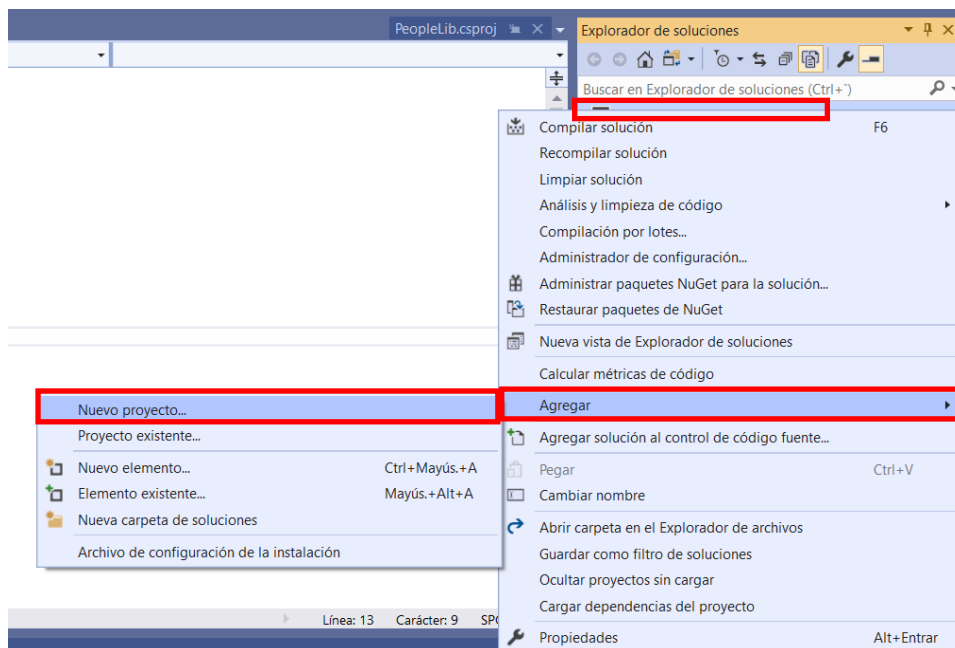
Right-click the default file *Class1.cs* and rename it *CPerson.cs*. The name of the class will be automatically renamed too.



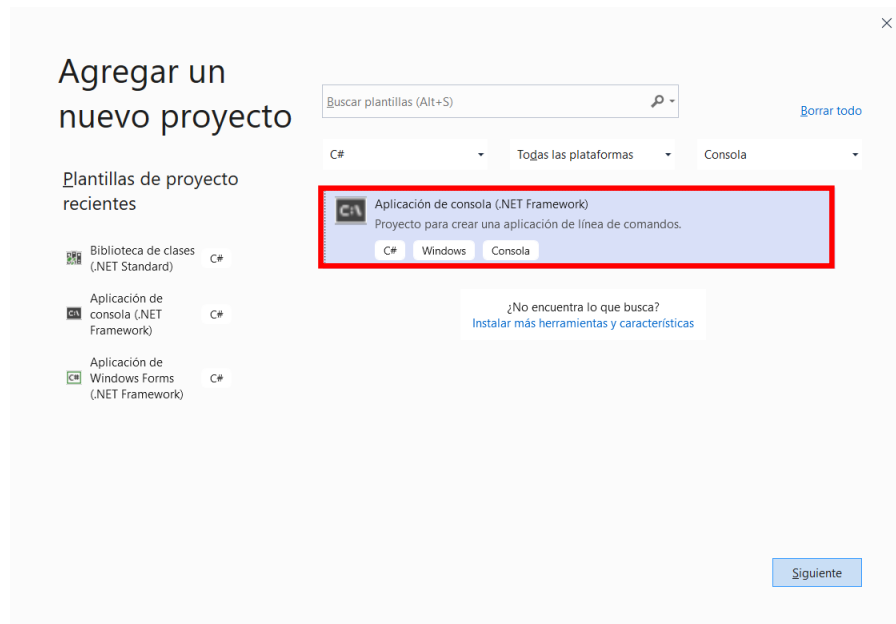
### 1.5. Copy the code of the class CPerson above and save the solution



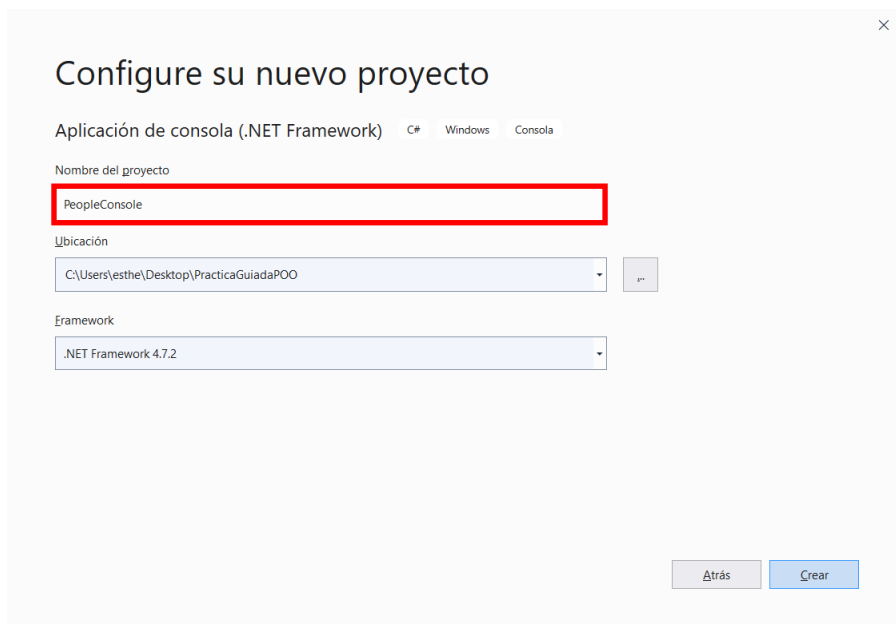
### 1.6. To add the main program to the solution, right-click *Solución 'PracticaGuiadaPOO'* in the Solution Explorer and select **Add->New Project...**



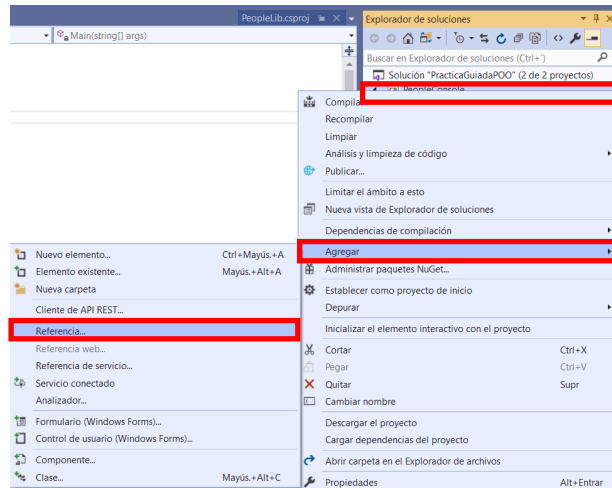
Pick the **Console Application** template (“Console Application (.NET Framework)”).



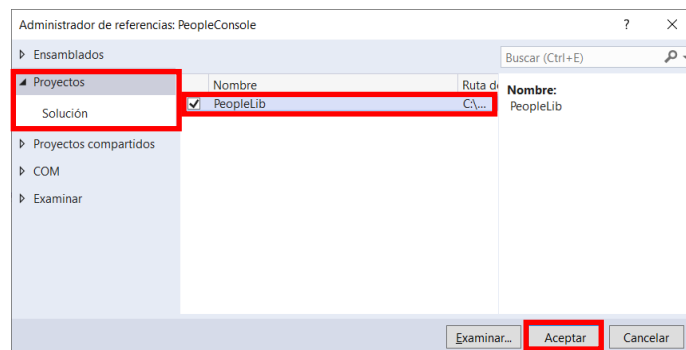
And name the project *PeopleConsole*:



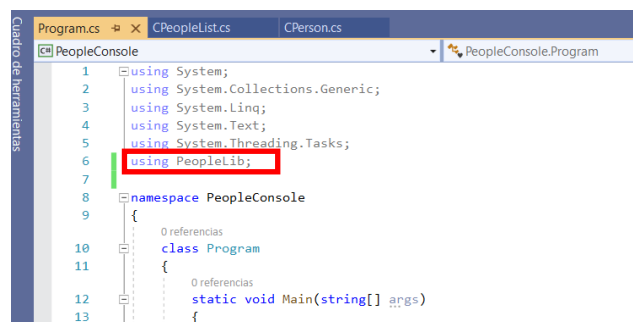
- 1.7. To indicate that the console project uses the class library project, we must add the reference. To do this, right-click on the project (*PeopleConsole*) in the solution explorer and select **Add->Reference...**



Go to the Projects/Solution tab and select the *PeopleLib* project.



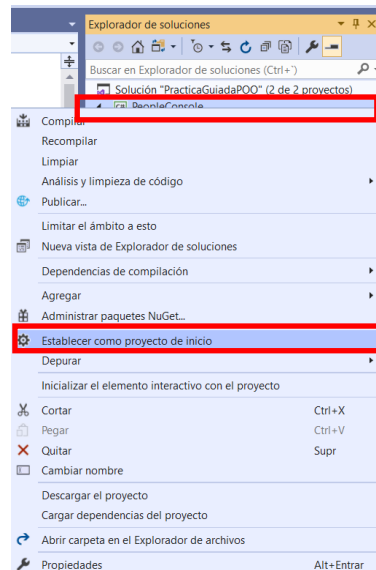
1.8. Copy the code of the example of the main program into the *Program.cs* file and save it. Do not forget to write the “*using PeopleLib*” directive.



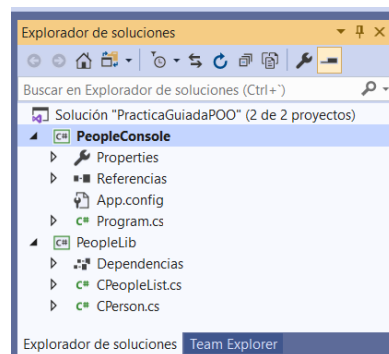
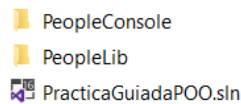
1.9. Finally, we must indicate the compiler which is the start Project (the one containing the main). In this case, there is only one main. But it is quite common to have one and more libraries to implement the functionality of the program and different interfaces that use them. For example, we can have a console application interface to test that the classes in the libraries work well, and a nice windows form application interface for the end-user application. Each one will have its own main, but we will like running only once at a time. Right-click the *PeopleConsole* project in the Solution Explorer and select the



**Set as StartUp Project** option. The name of the *PeopleConsole* project will be marked in bold.

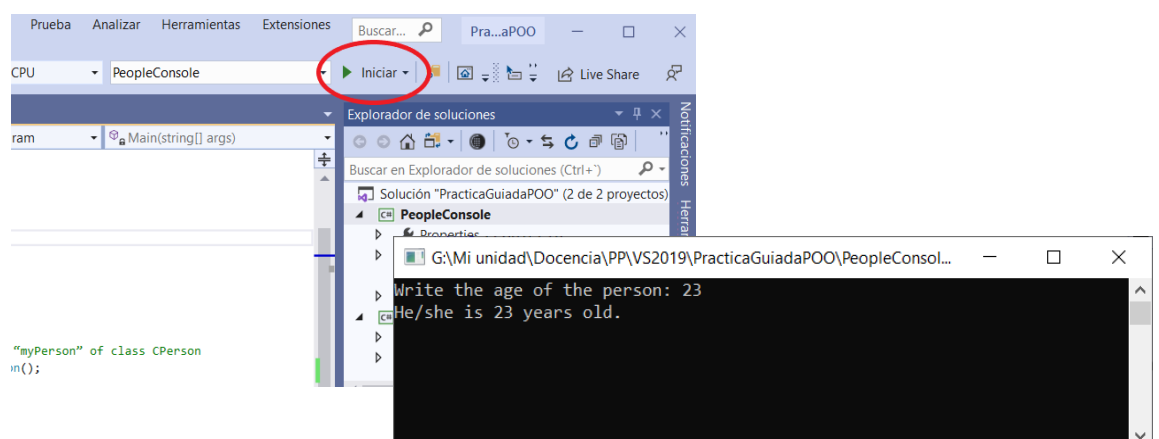


At the end of the process, the structure of the *PracticaGuiadaPOO* folder on disk and the appearance of the solution explorer should be as follows:



To reopen the application in the future, double-click the file *PracticaGuiadaPOO.sln*.

1.10. Now, you can build and run the application as usual.



Remember we said that the only way to work with an object is through the declared public methods. In this case, the class `CPerson` has only two public methods: `SetAge` and `GetAge`. Thus, the only things we can do with the object `myPerson` is to set or get its age using the `SetAge` and `GetAge` methods respectively.

2. Add the following line to the main program:

```
myPerson.height = 180;
```

Try to compile the program and look at the error message that appears. It means that you cannot access the `height` attribute directly, because it is not public. Definitely, **you can only use an object through the methods defined in the class.**

Classes often provide public methods to allow getting or setting attribute values. These methods are commonly called *setters* and *getters* because of the task they perform, but they are common methods and can have any name.

3. Add the following methods to the definition of the class `CPerson`, so that we can use them to set or get the height and name attribute values, as we do with the age:

```
SetHeight  
GetHeight  
SetName  
GetName
```

You must choose the right parameters and result for each method and write the code. Add the main program the sentences needed to check that the new methods work correctly.

## Common mistakes

Here you have some of the common mistakes we do when working with libraries:

- We miss the using directive in the program that uses the class.
- We miss the reference to the project or library that contains the class definition.
- There are several mains in the solution and we have not marked the right start-up project.

## Advantages and disadvantages of Object-Oriented Programming

Object-Oriented Programming tries to match the real world, which is made up of objects that can only (in general) be used in the way the designer has thought of. A washing machine, for example, consists of several elements (attributes) such as a motor, wires, tubes, a metallic box, and so on. To use the washing machine, nobody opens the metallic box and try to manipulate these elements. If we do not want to break the washing machine, we will use one of the established programs. It is the same for the objects: the fact of using the established methods (instead of accessing the attributes directly) reduces the probability of making mistakes.

Object-Oriented Programming also facilitates code modularity, as each class is in a different file. The applications are better organized and are easier to understand.

On the other hand, the maintenance of the applications is easier. We can change the definition of the class (for example, replace the algorithm used in one of the methods by a faster one) without modifying the rest of the program. Obviously, if we change the name, the parameters, or the result of the methods, then it would be necessary to change the parts of the program that uses these methods.

The above-mentioned advantages (modularity and easy maintenance) are especially relevant for large applications. In very small applications it will be likely not worth using Object-Oriented Programming. Note that, in the example code, there is a lot of code to do a very small task. We could have done it without objects, writing less code, and with the same degree of clearness and organization.

Finally, another important advantage of Object-Oriented Programming is the ease to reuse code. You could share your object `CPerson` with the other people in the world by just publishing its code in your web page. You could also improve it and publish new releases continuously. In fact, there are lots of objects codes (in C#, C++, Java, and other languages) at anybody's disposal on the internet.

### The constructor

A class constructor is a method that is automatically executed whenever an object of this class is created. This method has the same name as the class and it does not return any result.

4. Add the following method to the definition of your class `CPerson`:

```
public CPerson()  
{  
    this.age = 7;  
}
```

When we declare an object of the class `CPerson`, this constructor will automatically assign the value 7 to the age of the new person.

Add a new console application *PeopleConsole2* with the following main:

```
static void Main(string[] args)  
{  
    int myAge;  
  
    // Create "myPerson". The above constructor is executed.  
    CPerson myPerson = new CPerson();  
  
    // Get the age of "myPerson" and write it on the screen  
    myAge = myPerson.GetAge();  
    Console.WriteLine("The person is " + myAge + " years old.");  
}
```

The program only declares one object of class `CPerson` and writes its age on the console. The age is 7 because the constructor sets the attribute to this value automatically when the object was created.

Run the application **step by step (F11)** and check that the constructor is activated when the object is created.

The constructor can have parameters. For example:

```
public CPerson(int a, float h, string n)
{
    this.age = a;
    this.height = h;
    this.name = n;
}
```

Then, we can declare an object as follows:

```
CPerson myPerson = new CPerson(19, 180, "Juan");
```

In this way, we are setting the initial value of the three attributes at the moment of the declaration.

One class can have several constructors that differ in the type or number of parameters. When an object is created, the right constructor is executed depending on the arguments used in the declaration.

5. Add the new constructor above to your class `CPerson` and create a new console application *PeopleConsole3* with the following main code:

```
static void Main(string[] args)
{
    // Create two objects "p1" and "p2"
    // A different constructor is executed for each one.
    CPerson p1 = new CPerson();
    CPerson p2 = new CPerson(19, 180, "Juan");

    // Write the age of "p1" and "p2"
    Console.WriteLine("The first person is " + p1.GetAge());
    Console.WriteLine(p2.GetName() + " is " + p2.GetAge());
}
```

Run the application and check that it works well. Note that a different constructor is executed for objects `p` and `q` depending on the arguments used in their declarations.

## Methods

A method is a code block that contains a series of statements. A program causes the statements to be executed by calling the method. A method can receive arguments and can also return a value. Methods have access to the attributes of the class in which they are defined.

To call a method on an object, add a “dot” after the object name, the name of the method, and parentheses. Arguments are listed within the parentheses and are separated by commas.

## Parameter passing

The method definition specifies the names and types of any parameters that are required. When calling code calls the method, it provides concrete values called arguments for each parameter. The arguments must be compatible with the parameter type but the argument name used in the calling code does not have to be the same as the parameter name used in the method. The parameters can also be objects.

6. Add the following method `IsOlderThan` to your class `CPerson`:

```
public bool IsOlderThan(CPerson p)
{
    if (this.age > p.GetAge())
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

This method compares its age (the one of the object “before dot” in the caller) with the age of the person received as the parameter (object `p`).

Use this main program to check the new method:

```
static void Main(string[] args)
{
    // Create two objects "p1" and "p2"
    CPerson p1 = new CPerson(22, 165, "Ana");
    CPerson p2 = new CPerson(19, 180, "Juan");

    // Write a message showing who is the oldest one
    if (p1.IsOlderThan(p2))
    {
        Console.WriteLine(p1.GetName() + " is older than " +
            p2.GetName());
    }
    else
    {
        Console.WriteLine(p2.GetName() + " is older than " +
            p1.GetName());
    }
}
```

Objects are always passed by reference. This allows the method to change the value of the attributes and have that change persist.

7. Add the following method to your class `CPerson`:

```
public void CopyAgeTo(CPerson p)
{
    p.SetAge(this.age);
}
```

This method copies the value of its age (the one of the object “before dot” in the caller) to the age of the person received as the parameter (object p). Use this main program to check the new method:

```
static void Main(string[] args)
{
    // Create two objects "p1" and "p2"
    CPerson p1 = new CPerson();
    CPerson p2 = new CPerson(19, 180, "Juan");

    // Copy the age value of "p2" into "p1"
    p2.CopyAgeTo(p1);

    // Show the age of "p1" on the screen
    Console.WriteLine("p1 is " + p1.GetAge());
}
```

## Return values

Methods can return a value to the caller. If the return type (the type listed before the method name) is not `void`, the method can return the value by using the `return` keyword. A statement with the `return` keyword followed by a value that matches the return type will return that value to the caller. The result can also be an object. In that case, the type listed before the method name in the method definition is the class of the object.

**8.** Add the following method to your class `CPerson`:

```
public CPerson GetCopy ()
{
    // Create a copy of itself
    CPerson p = new CPerson();
    p.SetAge(this.age);
    p.SetHeight(this.height);
    p.SetName(this.name);

    // Return the new object (the copy) as a result
    return p;
}
```

This method creates a new person with the same attribute values (the value of `this` is the object “before dot”, the one used to call the method), and returns the new person. Use this main program to check the new method:

```
static void Main(string[] args)
{
    // Create two objects "p1" and "p2"
    CPerson p1 = new CPerson();
    CPerson p2 = new CPerson(19, 180, "Juan");

    // Copy the values of "p2" into "p1"
    p1 = p2.GetCopy();

    // Show "p1" on the screen
    Console.WriteLine(p1.GetName() + " is " + p1.GetAge());
}
```

## Recommendations

Here you have some recommendations extracted from *C# Coding Standards and Best Programming Practices*; by the dotnetspider team (latest version of this document can be downloaded from <http://www.dotnetspider.com/tutorials/BestPractices.aspx>):

- Use Pascal casing (first character of all words are Upper Case and other characters are lower case) for Class names and Method names.

Example: 

```
public int GetAge()
{
    ...
}
```

- Use Camel casing (first character of all words, except the first word are Upper Case and other characters are lower case) for variables and method parameters.

Example: 

```
int totalCount = 0;
```

- Use Meaningful, descriptive words to name variables. Do not use abbreviations.

Example: 

```
string name; // not string nam;
```

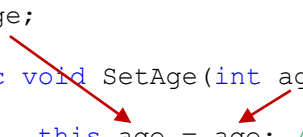
- Do not use underscores (\_) for local variable names.

- Use `this` to distinguish between parameter or local variables and attributes.

Example:

```
public class CPerson
{
    int age;

    public void SetAge(int age)
    {
        this.age = age; // attribute = parameter
    }
}
```



- Use one blank line to separate logical groups of code.

Example:

```
static void Main(string[] args)
{
    // Create two objects "p1" and "p2"
    CPerson p1 = new CPerson();
    CPerson p2 = new CPerson(19, 180, "Juan");

    // Write the age of "p1" and "p2"
    Console.WriteLine("Age of p1: " + p1.GetAge());
    Console.WriteLine("Age of p2: " + p2.GetAge());
}
```

- Comments should be in the same level of indentation as the code.
- Curly braces ({} ) should be on a separate line and not in the same line as if, for etc. Curly braces should be in the same level as the code outside the braces.

Example:

```
if (a > b)
{
    // Do something
    foo();
}
```