

# PCP — Lecture 06

Fall 2020 September 12, 2020

## Last Time - Operations, conversions and reading from a user

- It is possible to read *a number* from the user.
- What is a class, what is an object.
- How to define and use a class.
- Some novel keywords: **new** (to create an object), **return** (to send a value back from a method), **private** (to prevent access to the attributes from the “outside world”), **public** (to allow using the methods from the “outside world”).

## 1 Unified Modeling Language

UML is a *specification language* with multiple benefits:

- It is cross-language (you can use it to describe a class written in C#, Java, Python, ...),
- It represents only the “surface”, and the implementations details are left to programmers,
- It is a language to interact with non-programmer, or with programmers that simply want to use the class without knowing all of its details.

A class is represented as follows:

ClassName
- attribute: <b>int</b>
+ SetAttribute(attributeParameter: <b>int</b> ): <b>void</b>
+ GetAttribute(): <b>int</b>

Note that **void** is optionnal. For our Rectangle class, this gives:

Rectangle
- width: <b>int</b>
- length: <b>int</b>
+ SetLength(lengthParameter : <b>int</b> ): <b>void</b>
+ GetLength(): <b>int</b>
+ Setwidth(widthParameter: <b>int</b> ): <b>void</b>
+ GetWidth(): <b>int</b>
+ ComputeArea(): <b>int</b>

## 2 More Methods for the Rectangle Class

Last time, in lab, you were asked to write additional methods. They look like this:

```
public int ComputePerimeter()
{
    return length*2 + width*2;
}

public void DoubleRectangle()
{
    width *= 2;
    length *= 2;
}

public void Swap()
{
    int temp;
    temp = length;
    length = width;
    width = temp;
}
```

We could also write a method that multiply the length and width of a rectangle by a particular factor given in argument:

```
public void MultiplyRectangle(int factor)
{
    width *= factor;
    length *= factor;
}
```

Note that this method is more general than `DoubleRectangle`, which can be “emulated” using `MultiplyRectangle(2)`.

## 3 Variables and Methods Name and Conventions

### 3.1 Variable Scope

A variable exists at a particular *time* and place in a program, that defines *its scope*.

#### 3.1.1 Time

You cannot use a variable *before* declaring it! The following would return an error:

```
a = 3;
int a;
```

### 3.1.2 Space:

- One project can not access the variable of another project!
- `Rectangle.cs`'s variables are not directly accessible in `Program.cs` (you have to use accessors).
- The variable in a method are not accessible from the other methods.

## 3.2 Renaming

Identifiers can be uniformly renamed.

```
int a = 3;  
a += 2;
```

is the same as

```
int myVar = 3;  
myVar += 2;
```

We will use this example to discuss the scope:

```
class MyClass{  
    private int attribute;  
    public void SetAttribute(int attributeParameter){...}  
}
```

`MyClass`, `attribute`, `SetAttribute` and `attributeParameter` can be changed, those are the identifiers in this class.

## 3.3 Conventions

We can change all the identifiers in the classes if we want: class names, method names, etc. But it's good to have conventions.

“Hard” convention:

- Variable (including instance variables and parameters) names start with a lower case.
- Classes and Method names start with an upper case.
- You must be consistent.

Variations / Conventions for instances variables and argument names:

- `mAttributes` or `_Attributes`
- `SetAttributes(int aAttribute)`, or `(int value)`
- Names of accessors are up to you.

## 4 Named Constant

A constant is a variable whose value cannot change.

```
const int MONTHS = 12;
const double AVOGADRO = 6.0220e23; // Avogadro Number. Units 1/mol
const double PI = 3.14159265358979;
const double MILES_TO_KM = 1.60934;
```

- Value at to be fixed at declaration (= can only be initialized), and cannot change.
- Name is often ALL CAPS.

For instance,  $\pi$  is defined in the `Math` class and can be accessed as follows:

```
Console.WriteLine(Math.PI);
```

## 5 Format Specifiers

We can use interpolation to display more nicely numerical values. There are four important format specifiers in C#.

Format specifier	Description
N or n	Formats the string with a thousands separator and a default of two decimal places.
E or e	Formats the number using scientific notation with a default of six decimal places.
C or c	Formats the string as currency. Displays an appropriate currency symbol (\$ in the U.S.) next to the number. Separates digits with an appropriate separator character (comma in the U.S.) and sets the number of decimal places to two by default.
P or p	Print percentage

```
Console.WriteLine(
    "\n" + $"{1234.567:N}" // 1,234.57
    + "\n" + $"{1234.5:N}" // 1,234.50
    + "\n" + $"{1234.567:E}" // 1.234567E+003
    + "\n" + $"{1234.567:C}" // $1,234.57
    + "\n" + $"{1234.5:C}" // $1,234.50
    + "\n" + $"{.5:P}" // 50.00%
);
```