

PCP — Lecture 08

Fall 2020 September 24, 2020

Last Time - UML, Methods, Scope, Conventions, Constants and Format Specifier

- A UML diagram is a convenient way to represent the “surface” of a class, that is, its description, without including the actual code.
- We revisited some methods for the Rectangle class, and enriched it with the `ComputePerimeter` or `MultiplyRectangle` methods.
- We defined the scope of a variable as the place in space and time where a variable can be accessed.
- We discussed some of the conventions on the naming of methods and variables.
- We studied the role and importance of constant variables.
- We saw how string interpolation could use format specifiers to display numerical information in a “formatted” way.

1 A Class for Classroom

1.1 UML - Specification

ClassRoom
- name: <code>string</code>
- number: <code>int</code>
+ <code>SetName(nameParameter : string): void</code>
+ <code>GetName(): string</code>
+ <code>SetNumber(numberParameter: int): void</code>
+ <code>GetNumber(): int</code>

1.2 Implementation

```

1  using System;
2
3  class Classroom
4  {
5      private string name;
6      private int number;
7
8      public void SetName(string nameParameter)
9      {
10         name = nameParameter;
11     }
12     public string GetName()

```

```

13     {
14         return name;
15     }
16
17     public void SetNumber(int numberParameter)
18     {
19         number = numberParameter;
20     }
21     public int GetNumber()
22     {
23         return number;
24     }
25 }

```

1.3 Default Values

What if we display the values of the instance variables before setting them?

```

ClassRoom english = new ClassRoom();
Console.WriteLine(english.GetName()); // Nothing!
Console.WriteLine(english.GetNumber()); // 0

```

Indeed, instance variables are different from “usual” variables in that sense that they receive a “default” value when created. This value depends of the variable datatype:

Type	Default
numerical value	0
char	'\x0000'
bool	false
string	null

- Note how different it is from the variables we have been using so far, that could not be for instance displayed if their value had not been set.
- We can set a different default value, using, in the class declaration,

```

private string name = "Unknown";
private int number = -1;

```

1.4 Constructors

1.4.1 Custom

A constructor is a method used to create an object. It has to have the same name as the class, and doesn't have a return type.

```

public ClassRoom(string nameParameter, int numberParameter)
{
    name = nameParameter;

```

```

        number = numberParameter;
    }

```

We use it as follows:

```
ClassRoom math = new ClassRoom("Bertrand", 5);
```

Note:

- the order of the arguments matter,
- the variables, as usual, have a particular scope,
- constructor do not have a return type (not even `void`)

In the UML diagram, we would add:

```
+ «constructor» ClassRoom(nameParameter: string, numberParameter: int)
```

Note that we could skip the «constructor» part, can you tell why?

1.4.2 Default

If we implement this constructor, then we lose the “No args”, default constructor

```
public ClassRoom() { }
```

We can re-define it, using something like:

```
public ClassRoom() {
    name = "Unknown";
    int = -1;
}
```

2 Signature and Overloading

Every method has a signature made of - its name, - its parameters types (but not the parameter names).

Note that the return type is not part of the method signature in C#.

In a class, all the methods need to have a different signature. You cannot, for example, have these two methods in the same class:

```
int DoSomething(int a, int b);
string DoSomething(int c, int d);
```

It is possible, however, to have two methods with the same name, as long as they have different signatures. If we are in such a situation, then we say that we are *overloading*. We will look at examples of overloading in lab.

3 ToString

A particular method can be used to display information about our objects. It is called `ToString`, and can be defined as follows:

```
public override string ToString()
{
    return "Person: " + Name + " " + Age;
}
```

We will look at examples and usage in class and lab.