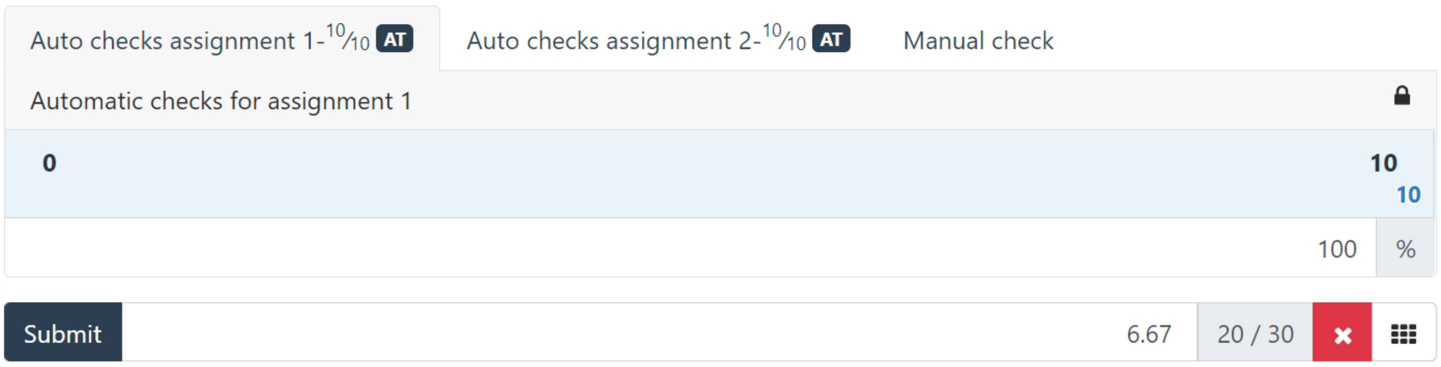
Week 1 assignments

When creating the program code, you must apply the following basic principles:

- create a separate project for each assignment;
- use name ‘assignment1’, ‘assignment2’, etcetera for the projects;
- create one solution for each week containing the projects for that week;
- make sure the output of your programs are the same as the given screenshots;

CodeGrade auto checks

Make sure all CodeGrade auto checks pass (10/10) for your assignments. The manual check will be done by the practical teacher.

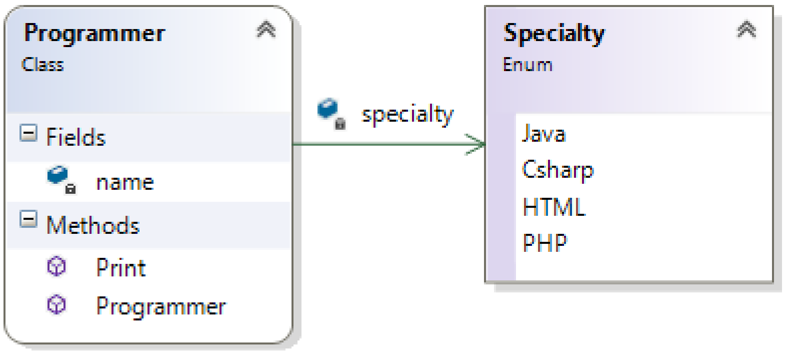

Auto checks assignment 1- 10/10 AT
Auto checks assignment 2- 10/10 AT
Manual check
Automatic checks for assignment 1
0
10
10
100 %
Submit
6.67
20 / 30

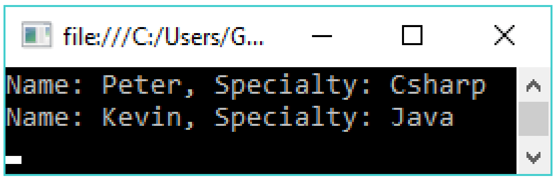
Assignment 1 – Software company

The software company Blue Pigeon has grouped a portion of its staff into teams. Each team consists of four programmers. Each programmer has a specialty, such as Java, C#, HTML or PHP.

a) Create a Specialty enumeration that contains these specialties, and store this enum in a separate file.

b) Create a *Programmer* class (in a separate file) that contains a name and a member of the type *Specialty*. Give this class a constructor in which you can include the name of the programmer and his/her specialty. Also give the class a method Print() that prints the fields of the object concerned on the screen. Test this Print method by creating and printing a few programmers.

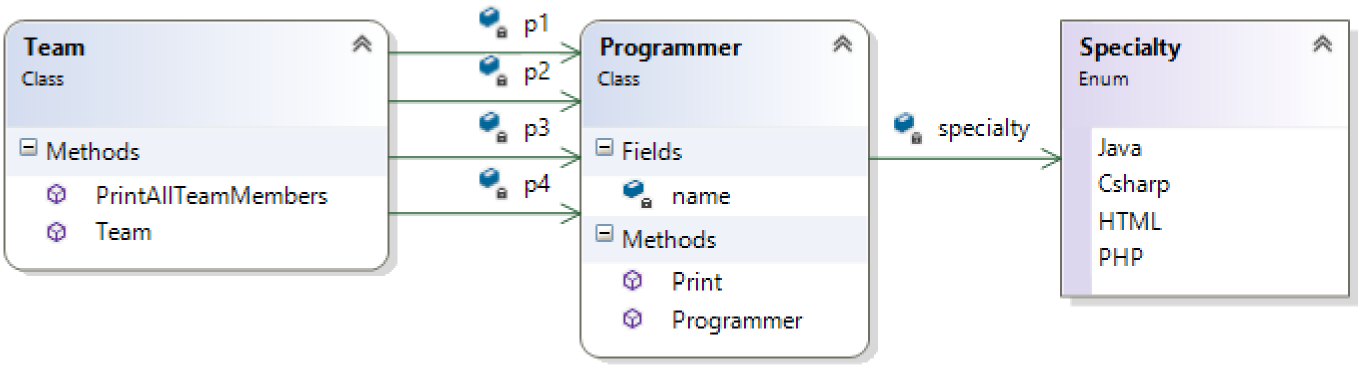


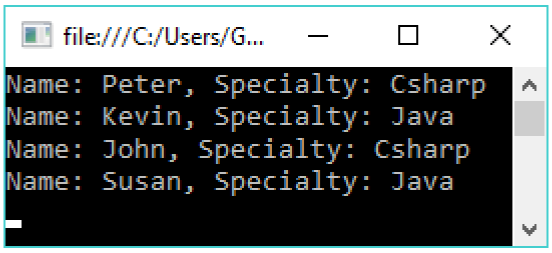


```
Name: Peter, Specialty: Csharp
Name: Kevin, Specialty: Java
```

Example of Print (class Programmer)

c) Create a class *Team* (in a separate file) that creates, in the contructor, four new instances of *Programmer* with made-up names and specialties. Give the class a method *PrintAllTeamMembers*() that prints the names of the programmers and their specialties on the screen.
Call this method from the Start method to show the result (first create a Team object for this).





```
Name: Peter, Specialty: Csharp
Name: Kevin, Specialty: Java
Name: John, Specialty: Csharp
Name: Susan, Specialty: Java
```

Example of PrintAllTeamMembers (class Team)

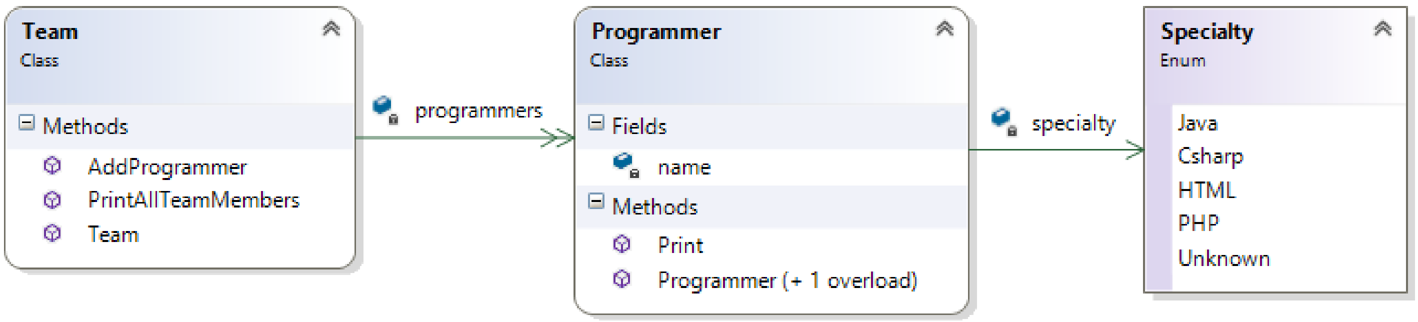
Think about this design; how can it be improved?

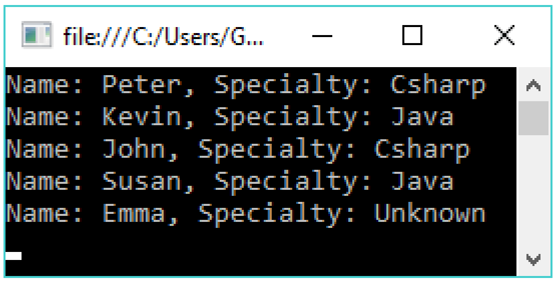
d) Modify class Team: add a list of programmers, modify the constructor (create the list of programmers), and add a method *AddProgrammer* with the following signature:

```
public void AddProgrammer(Programmer p)
{
    // add 'programmer' to the list
    // ...
}
```

Change the Start-method: the creation of the Programmer-objects (instances) will no longer be done in class Team, but can now be done in the Start method. This means that class Team will no longer contain the same (hardcoded) programmers, the composition of a team can now be made by the 'user' of the class.

e) Add a 2nd constructor to class Programmer that only receives the name of the programmer, and sets the specialty (hardcoded) to 'Unknown'. You can use the other constructor (with 2 parameters), by calling it from the constructor with 1 parameter (see paragraph 4.7.4, Yellow book). Test this 2nd constructor by creating a programmer without specialty, add it to the team, and then print the team again.





```
Name: Peter, Specialty: Csharp
Name: Kevin, Specialty: Java
Name: John, Specialty: Csharp
Name: Susan, Specialty: Java
Name: Emma, Specialty: Unknown
```

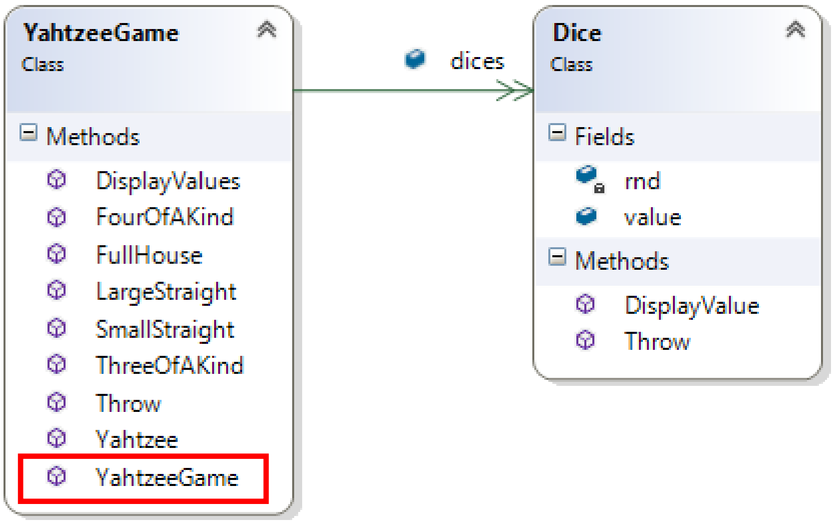
Example of PrintAllTeamMembers (class Team), where a programmer without specialty is created (and added to the team)

f) Check if the classdiagram of your own VisualStudio-project is the same as the one displayed above. To create a classdiagram, choose project-option "View Class Diagram" in Visual Studio. To get the arrows, click on a fieldname and select popup menu-item "Show as Association" (for normal fields like *specialty*) or select "Show as Collection Association" (for lists like *programmers*).

Assignment 2 – Yahtzee game

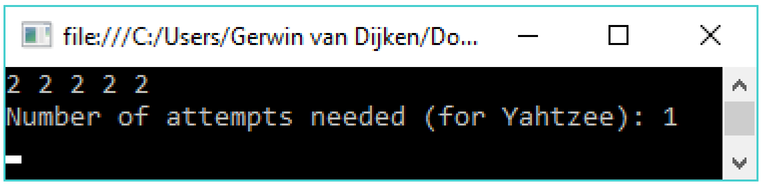
Now that we know there's a special method for initialising an object (called 'constructor'), we can modify class YahtzeeGame (see week 1 of Programming 2).

a) Remove the Init method and add a constructor that creates the 5 dices. Of course, you must also remove the call to the Init method (is done in the Start method).
→ Check if the Yahtzee application is still working (with the loop that waits until Yahtzee was thrown, 5 equal dices).



As you can see, class YahtzeeGame no longer has an Init method (but a constructor)

b) Class 'Dice' has a <u>static</u> random generator, we want to get rid of this. What happens if you remove the word 'static' and run the program again? Do you also get Yahtzee immediately?

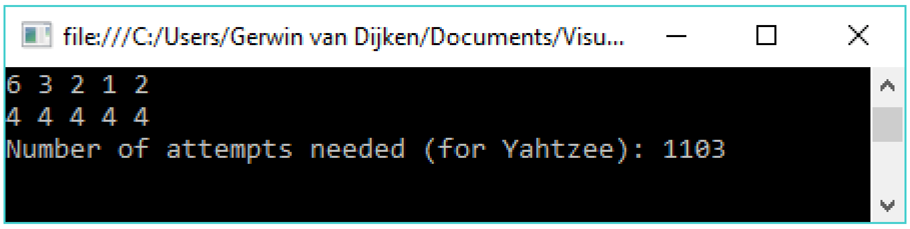


```
2 2 2 2 2
Number of attempts needed (for Yahtzee): 1
```

This effect will only happen in (the old) .NET Framework, not in .NET Core !!

This happens because without 'static' each Dice-object creates its own Random-object, and since the (5) Dice-objects are created close together, the Random-objects are also created close together. A Random-object uses the current time to create 'pseudo-random' numbers, and since the current time for all (5) object will be the same, the generated numbers will also be the same everytime.

c) Give class Dice a constructor with a Random-object <u>as parameter</u>, and make sure this object is stored 'internally' (in a member), so method 'Throw' can access it. Modify the constructor of class YahtzeeGame: create <u>one</u> Random-object, and pass this one to all Dice-objects (while creating them).
→ Check if the application is working normally again.



```
6 3 2 1 2
4 4 4 4 4
Number of attempts needed (for Yahtzee): 1103
```