

## Assignment 0 – Useful methods

To read data for your application (like a name or an age) you always need to write the same code. In this assignment you will create a few methods that will make reading data a bit easier.

You will also use these methods in assignment 1 and 2. This means that you want to write these methods as clear as possible, and to make them work as good as possible.

This week we will copy the methods to other assignments, but later we will create our own code-library.

- a) Create a method with signature:

```
int ReadInt(string question)
```

that prints the given question and subsequently reads an integer value. This value is returned by the method.

→ Test the method (with the code below).

- b) Also create a variant of ReadInt that checks if the given integer value is within a certain range. This means that you'll have 2 methods ReadInt. We call this 'method overloading'.

The signature of this method is:

```
int ReadInt(string question, int min, int max)
```

When the entered value is not within the range [min..max] the question is asked again and a new value is read (use a loop).

→ To implement this 2<sup>nd</sup> ReadInt-method, make use of the 1<sup>st</sup> ReadInt-method.

→ Test the method (with the code below).

- c) Also create a method with signature:

```
string ReadString(string question)
```

This method asks a question and returns the answer.

→ Test the method (with the code below).

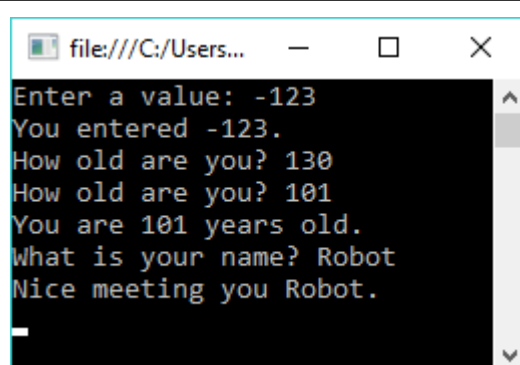
```
static void Main()
{
    Program myProgram = new Program();
    myProgram.Start();
}

void Start()
{
    int value = ReadInt("Enter a value: ");
    Console.WriteLine("You entered {0}.", value);

    int age = ReadInt("How old are you? ", 0, 120);
    Console.WriteLine("You are {0} years old.", age);

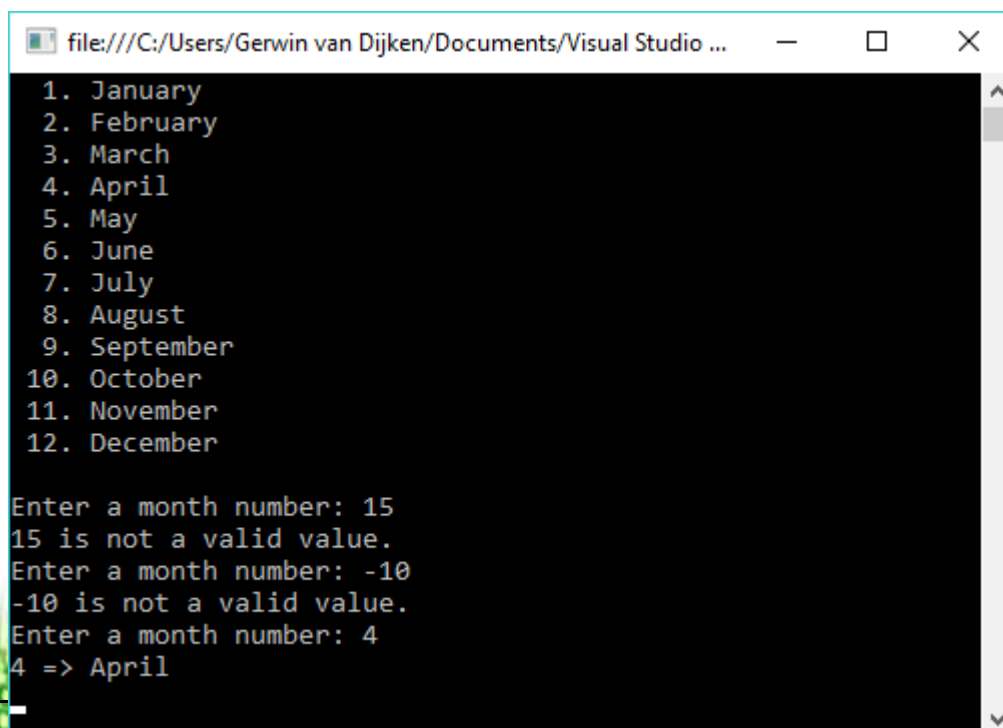
    string name = ReadString("What is your name? ");
    Console.WriteLine("Nice meeting you {0}.", name);

    Console.ReadKey();
}
```



## Assignment 1 – Enumerations – Month

- a) Create an enumeration type with name Month (store this enumeration in a separate file 'Month.cs') and give the enum Month 12 options: January, February, ..., November en December. Make sure that the first option (January) has value 1.
- b) Create a method with signature:  
`void PrintMonth(Month month)`  
that prints the given month (just the name).  
→ Call method PrintMonth (from the Start method) to test it.
- c) Create a method with signature:  
`void PrintMonths()`  
that uses method PrintMonth (see item b) to print all enum options of Month below each other (use a loop). Make sure that the numbers (1..12) are 'right-aligned'<sup>1</sup> zijn (see example below).  
→ Call method PrintMonths from the Start method.
- d) Create a method with signature:  
`Month ReadMonth(string question)`  
that reads a month number and converts this number from an int-value to an enum-value. This enum value is returned.  
→ Call this method (from the Start method) and subsequently print the returned value (ofcourse with method PrintMonth, see item b).
- e) What happens if an invalid number is entered ( $1 > \text{number} > 12$ )?  
Modify the ReadMonth method to print a message when an invalid number has been entered; for testing invalid enum values use method `Enum.IsDefined(typeof(Month), ...)` that returns `true` or `false`. Continue reading a month number until `Enum.IsDefined` returns `true`.



```
file:///C:/Users/Gerwin van Dijken/Documents/Visual Studio ...
1. January
2. February
3. March
4. April
5. May
6. June
7. July
8. August
9. September
10. October
11. November
12. December

Enter a month number: 15
15 is not a valid value.
Enter a month number: -10
-10 is not a valid value.
Enter a month number: 4
4 => April
```

<sup>1</sup>Some examples of right-aligned can be found at (e.g.) <http://www.csharp-examples.net/align-string-with-spaces/>

## Assignment 2 – Structures - Person

In this assignment you will read integer and string values. Reuse your Read methods (of assignment 0) by copying them to Program.cs (of assignment 2).

### The struct Person

- Create a struct `Person` (in a separate file) with fields `FirstName`, `LastName`, `Age` and `City`. Declare (in the `Start`-method) a variable of type `Person` and fill all fields of this struct.
- Create a method with signature  
`Person ReadPerson()`  
 for reading a (complete) `Person`.  
 → Use methods `ReadInt` and `ReadString` (copied from assignment 0).
- Create a method with signature  
`void PrintPerson(Person p)`  
 that prints the given `Person`.

### The enum GenderType

- Add to struct `Person` a field `Gender` of type `GenderType` (options: `Male` and `Female`).
- Create a method with signature  
`GenderType ReadGender(string question)`  
 for reading a gender. Use this in method `ReadPerson` to read the gender.
- Also create a method with signature  
`void PrintGender(GenderType gender)`  
 that prints 'm' of 'f' depending on the given gender.  
 → Use this in method `PrintPerson` to print the gender.

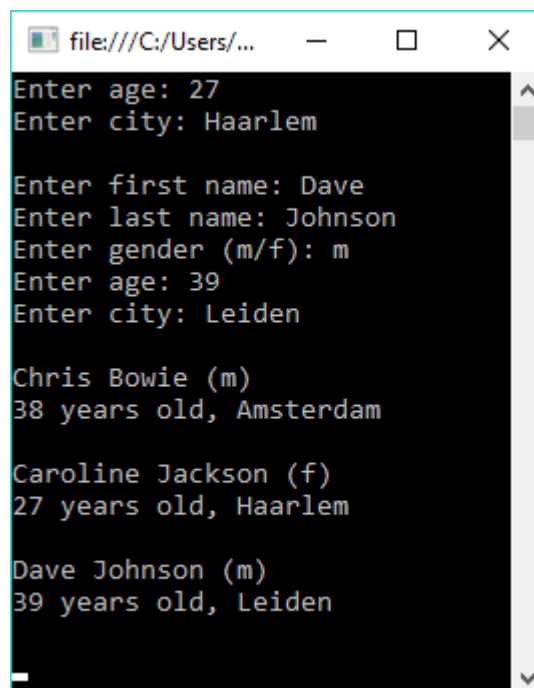
### Persons in an array

- Create a `Person` array with length 3. Fill this array (in a loop) with information entered by the user. Make sure to use method `ReadPerson`. Subsequently print all persons (again in a loop) with the `PrintPerson` method.  
 → Output of the application must be (more or less) the same as the screenshot below.

### Celebrate birthday

- Create a method with signature  
`void CelebrateBirthday(...)`  
 that receives a `Person` and increases the age of this person.  
 → Test this method by printing a person before and after calling `CelebrateBirthday` (from the `Start` method).

N.B. Probably there is problem with celebrating (the age is not increased). What could be the reason for this? Search a solution for this, but incrementing the age should stay inside the method, and printing the age should stay in the `Start` method!



```

file:///C:/Users/...
Enter age: 27
Enter city: Haarlem

Enter first name: Dave
Enter last name: Johnson
Enter gender (m/f): m
Enter age: 39
Enter city: Leiden

Chris Bowie (m)
38 years old, Amsterdam

Caroline Jackson (f)
27 years old, Haarlem

Dave Johnson (m)
39 years old, Leiden
  
```

## Assignment 3 – Classes - Yahtzee

### The class Dice

- Create a class Dice (in a separate file).
- Add a method with signature:  
`public void Throw()`  
 inside the class Dice (yep, that's possible). In this Throw method a new random value is generated between 1 and 6. Store it in field value.
- Also add to class Dice a method with signature:  
`public void DisplayValue()`  
 This method displays the current value of the dice.
- Test the class Dice by creating a Dice variable and by throwing this dice a few times (display all values).

```
class Dice
{
    public int value;
    static Random rnd = new Random();
}
```

```
void Start()
{
    Dice d1 = new Dice(); // dice 1
    d1.Throw();
    d1.DisplayValue();
}
```

### The class YahtzeeGame

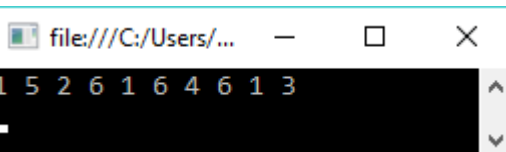
- Create a class YahtzeeGame (in a separate file) containing a Dice array 'dices'.  
 Add a method `public void Init()` that creates all dices (with new Dice).
- Add a method to class YahtzeeGame with signature  
`public void Throw()`  
 In this Throw method all 5 dices are thrown (by calling the method Throw of the class Dice).
- Also add to class YahtzeeGame a method with signature  
`public void DisplayValues()`  
 that displays the value of all dices (see example). Of course method DisplayValue of class Dice is used for this.
- Now test class YahtzeeGame with the code below:

```
static void Main(string[] args)
{
    Program myProgram = new Program();
    myProgram.Start();
}

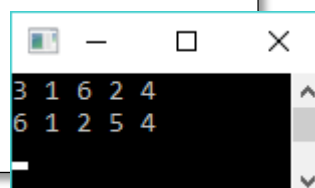
void Start()
{
    YahtzeeGame yahtzeeGame = new YahtzeeGame();
    yahtzeeGame.Init();

    yahtzeeGame.Throw(); // throw dices
    yahtzeeGame.DisplayValues(); // display result

    yahtzeeGame.Throw();
    yahtzeeGame.DisplayValues();
}
```



```
file:///C:/Users/...
1 5 2 6 1 6 4 6 1 3
```



```
3 1 6 2 4
6 1 2 5 4
```

Check for Yahtzee, ThreeOfAKind, FourOfAKind etc.

- i) Create a method inside class YahtzeeGame with signature:

```
public bool Yahtzee()
```

This Yahtzee method determines if Yahtzee has been thrown (5 dices with the same value); if this is the case return **true**, otherwise return **false**.

- j) Use the code printed below to check if you have implemented class YahtzeeGame correctly. A possible output of the program is displayed below.

How many throws (needed) did you expect (more or less) as result?

*(how many yahtzee options are there, and how many different throws with 5 dices?)*

- k) [optional] Expand the class with the following Yahtzee options:

- ThreeOfAKind
- FourOfAKind

- l) [optional] Expand the class with the following Yahtzee options:

- FullHouse
- SmallStreet
- BigStreet

```
6 4 3 2 4
3 5 3 3 1
5 4 4 1 6
2 5 6 4 6
5 4 2 5 6
2 3 3 3 2
3 3 3 3 3
Number of attempts needed (for Yahtzee): 1733
```

```
static void Main(string[] args)
{
    Program myProgram = new Program();
    myProgram.Start();
}

void Start()
{
    YahtzeeGame yahtzeeGame = new YahtzeeGame();
    yahtzeeGame.Init();
    PlayYahtzee(yahtzeeGame); // play the game
}

void PlayYahtzee(YahtzeeGame game)
{
    int nrOfAttempts = 0;

    do
    {
        game.Throw();           // throw all dices
        game.DisplayValues();    // display the thrown

        nrOfAttempts++;

    } while (!game.Yahtzee());

    Console.WriteLine("Number of attempts needed (Yahtzee): {0}", nrOfAttempts);
}
```