Assignment 1: Help for Pet part

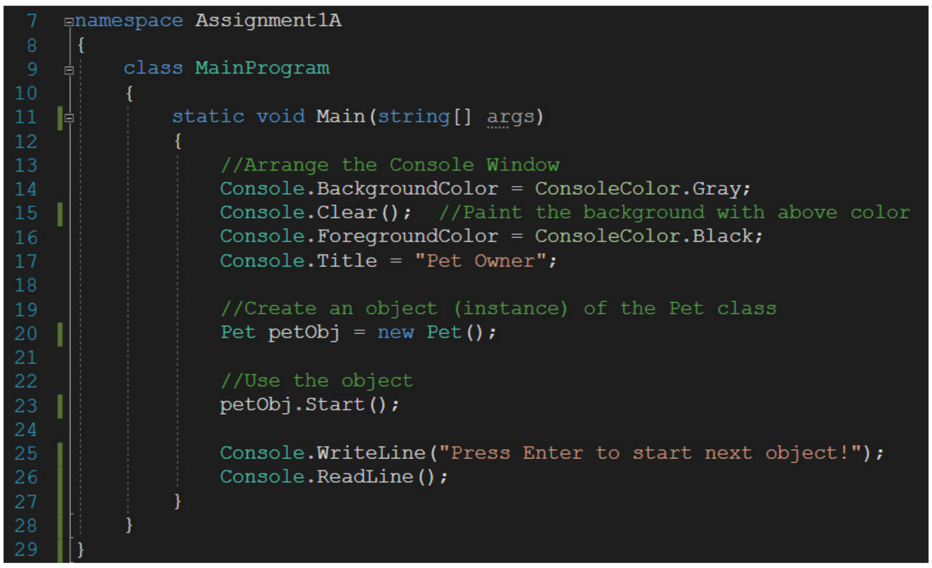
In this first assignment, many things come your way at the same time. Therefore, to help you get started quicker, a step-by-step guidance for Part Ia of the assignment is provided in this document. This is in addition to some instructions given in the assignment description already.

The identifiers in blue that occur in the code clips here in this document are all keywords in C#. Keywords are reserved by the language and should not be used as variable names or method names.

Note: *By Console Window, it is meant the Command Prompt window that comes with the Operating System. On the other hand, **Console** is also the name of the .NET class that contains the methods ReadLine (Console.ReadLine), Write and WriteLine and so on.*

The Program class:

Every C# program must have a class containing a method that is called **Main**. This method starts the program and the code that you write in the method is executed automatically line by line. In order to use another class and call its methods, we must create an object (instance) of that class. Here we are going to create an object of the **Pet** class and then calls the object's **Start**-method to start the Pet-class. Notice that the object, **petObj**, is declared and created on the same line (Line no 20).



```
 7  namespace Assignment1A
 8  {
 9      class MainProgram
10      {
11          static void Main(string[] args)
12          {
13              //Arrange the Console Window
14              Console.BackgroundColor = ConsoleColor.Gray;
15              Console.Clear();  //Paint the background with above color
16              Console.ForegroundColor = ConsoleColor.Black;
17              Console.Title = "Pet Owner";
18
19              //Create an object (instance) of the Pet class
20              Pet petObj = new Pet();
21
22              //Use the object
23              petObj.Start();
24
25              Console.WriteLine("Press Enter to start next object!");
26              Console.ReadLine();
27          }
28      }
29  }
```

In fact, the lines 20 and 23 were all we needed to do. Lines 14 to 17 are only extra code that make the output look nicer. Places starting with "//" marks the rest of the line as comments which are our own notes and will be ignored by the compiler.
The Console.**ReadLine** method on line 26 is used to let the Prompt window stay on the screen; otherwise it will be closed as soon as the execution comes back from the Start method of the **Pet** class, and you will never get a chance to see the window with all results..

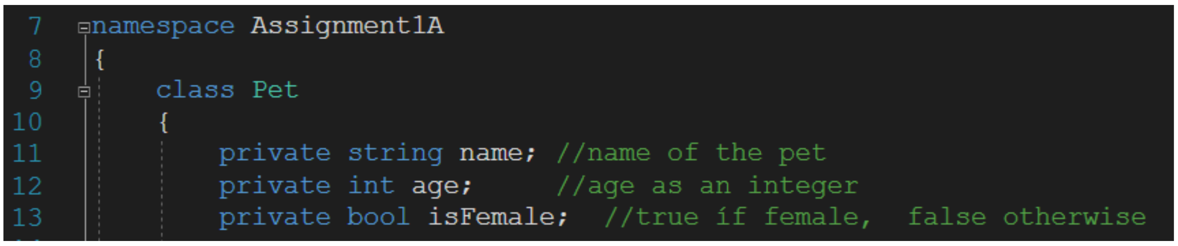
The class Pet:

As we discussed in our lessons that we have had so far, a class has fields and methods, to store values and manipulate the values. Every field (instance variable) should have a data type and an access modifier. We have said that fields should always be declared private but methods can be public or private depending on whether the method can be useful to other classes in which case it should be public. Methods that are used internally in a class can be declared private.

Fields: are variables needed to save values:

- that come from the user (or from other parts of an application),
- values that are needed between the methods for internal use, or
- for providing output.

In this class, we need variables only for storing user input: The **name** of a pet is obviously one or more words, and the data type for it should therefore be string data type. Age is a number, and as we usually use a whole number to specify age, we can use an int for the variable **age**. To find out whether the pet in question is a female or a male, we ask the user if the pet is a female and expect a yes or no answer. Then we interpret true as “yes” and false as “no” in our code.

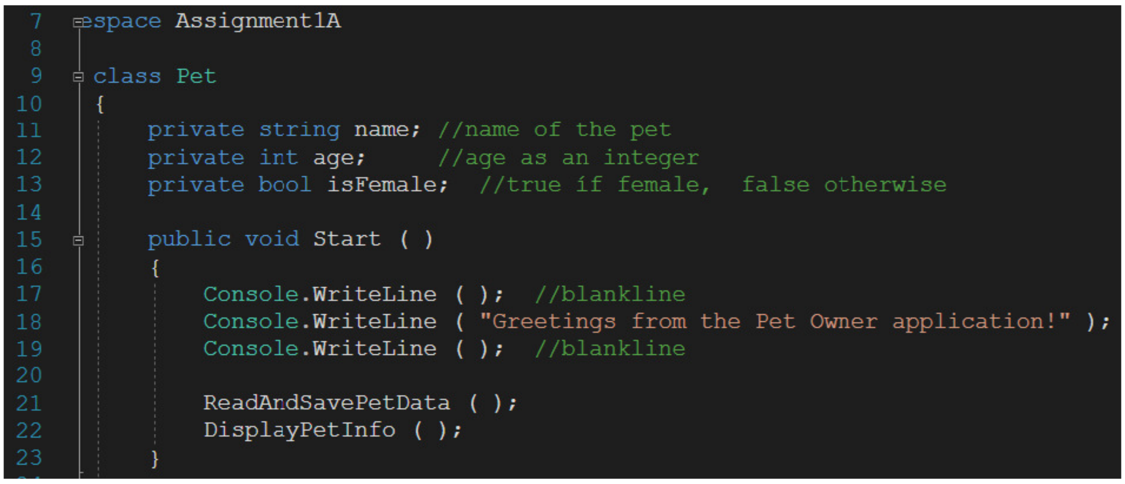


```
namespace Assignment1A
{
    class Pet
    {
        private string name; //name of the pet
        private int age;     //age as an integer
        private bool isFemale;  //true if female,  false otherwise
```

Notice that this class is encapsulated inside the same namespace as the Program class.

A namespace is used as a name for a group of related classes. In this way, different namespaces can have classes with same names. An application can have one namespace or more.

Now, to control the order in which the methods of the Pet class should run, we write a method and call it **Start**: We call other methods from this method.



```
 7  espace Assignment1A
 8
 9  class Pet
10  {
11      private string name; //name of the pet
12      private int age;     //age as an integer
13      private bool isFemale; //true if female,  false otherwise
14
15      public void Start ( )
16      {
17          Console.WriteLine ( );  //blankline
18          Console.WriteLine ( "Greetings from the Pet Owner application!" );
19          Console.WriteLine ( );  //blankline
20
21          ReadAndSavePetData ( );
22          DisplayPetInfo ( );
23      }
```

Lines 21 and 22 are calls to method that are written, and thus the compiler will complain if we try to compile. Let's write the methods.

The `ReadAndSavePetData` method:

This method is a `void` method and it is declared as `public` so it can be called from the Main method (just in case). Some of the methods should be `public` (in contrast to fields that should wholly be private) in order to be accessible for other classes. Methods that may not, or should not, be exposed to other classes, should be declared as `private`.

```
public void ReadAndSavePetData ( )
{

}
```

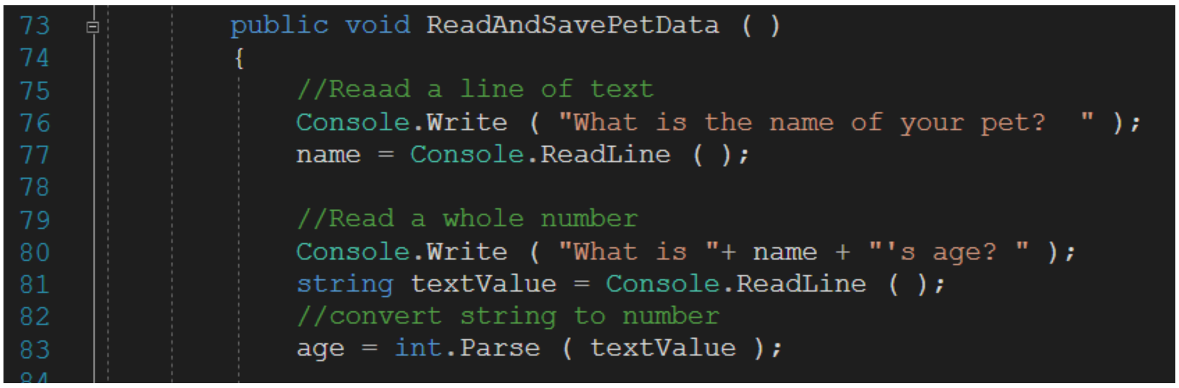
The next step is to write code (in the body of the method) to interact with the user and get values for the name, age, and gender of the pet to store in the instance variables (fields). We use Console.ReadLine to read a line of text from the Console Window. The user must of course receive some info about the type and purpose of the input expected by the program. This is achieved by using Console.WriteLine which write out a line of text to the Console Window.

To write a text to the Console window, use **Console.Write ( )** or **Console.WriteLine ( )** and send the text within the parentheses. The text has to be enclosed inside quotation marks. The two methods work quite the same except one difference. The **Console.Write ( )** method leaves the cursor at the position where the text ends while the **WriteLine ( )** method moves the curser to the beginning of the next line on the Console window after displaying the text it contains. The Console Window remembers the cursor position for the next write/read operation.

To fetch the input from the user to the program, we use the method **Console.ReadLine ().** This method cannot write anything to the Console window; it brings with it a line of text beginning from the cursor position to end of the line where the user's presses the **Return** key on the keyboard. The text can consist of one character, many characters (words) or no character if the user presses Enter directly with writing a text.

Everything the user writes, even numbers and symbols are considered as a sequence of chars, “135”, “Y”, “Hi, how are you” o “ 2 + 5”. It will be your job to convert the text to the desired type. For instance if the user is to feed in an integer, e.g. 135. ReadLine takes these chars as a string with three characters as“135” (chars 1, 3 and 5). Thus, “135” is not an integer; it represents an integer. It needs to be converted to the integer 135, before it can be saved in a variable of the type `int`. Other numerical types work in the same way. If we would like to get values of the types `char` or `string`, we do not have to convert to any type.

Console.Read() (returning an `int`) and **Console.ReadKey()** (returning a **key**) function are quite differently compared to **Console.ReadLine ()**, and they are complicated. Do not use these two methods to read values; the **Console.ReadLine ()** will suffice!



```
73      public void ReadAndSavePetData ( )
74      {
75          //Reaad a line of text
76          Console.Write ( "What is the name of your pet?  " );
77          name = Console.ReadLine ( );
78
79          //Read a whole number
80          Console.Write ( "What is "+ name + "'s age? " );
81          string textValue = Console.ReadLine ( );
82          //convert string to number
83          age = int.Parse ( textValue );
84
```

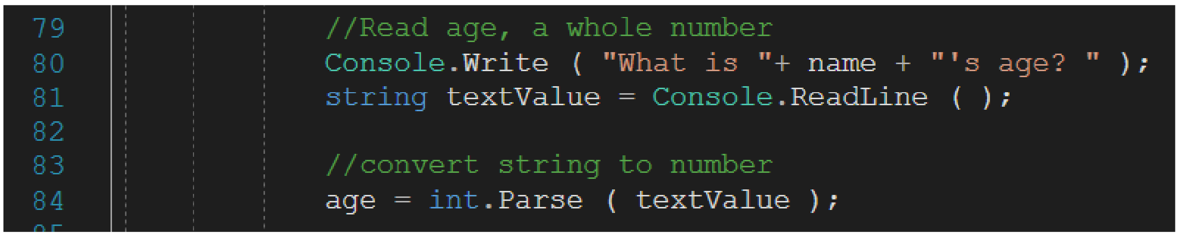
Here the user receives the message “*What is the name of your pet?*” followed by a couple of blank spaces when the code on Line 76 is executed. The **Console.Write()** method works such that the cursor remains at the same line where the message ends, after the two blank spaces that follow the question mark. The **Console.Write()** method is a `void` method, implying that its job is only to writ the text to the Console Window and then return back. The execution continues directly to the next line (77). **Console.WriteLine()** is also a void method.

What happens on Line 77? Here the **name** variable receives the results of the value from the method **Console.ReadLine ( ).** This is method is not a void method because it can hold the value it reads. It works similar to a `void` method but with one important difference, it returns a value after doing its job to the caller (line 77).

All return values have a data type, and the return value can be saved in a variable of that type only. The **Console.ReadLine**() does nothing but to bring a line of text from the cursor position on the Console window to the position where the user presses the **Enter** key on the keyboard. Whatever the user writes prior to pressing the Enter key is transported to the program by this method. If the user does not provide any text and presses the **Enter** key, **ReadLine** returns an “empty” string, i.e.**""**, a string with no character.

A very important thing to note here is that the variable receiving the result of the **ReadLine()** must be of the `string` data type. That is why we declared the variable **name** as a `string`. Otherwise, we must apply a data conversion (which we are going to do when reading the age value).

Reading **age** from the Console:



```
79            //Read age, a whole number
80            Console.Write ( "What is "+ name + "'s age? " );
81            string textValue = Console.ReadLine ( );
82
83            //convert string to number
84            age = int.Parse ( textValue );
```

In this part, we need to take a few steps to save an integer value in the variable **age**.

On Line 80, the message that **Console.Write** should display on the Console window, is a combination of three substrings. The first substring is "**What is** "; the second substring is the value currently stored in the variable **name**, so name will be replaced by its value, and that is why the variable name is not enclosed by quotation marks! . The third substring is **"'s age?** "

The three substrings are put together by the ‘+’ operator. If you wonder, the minus sign cannot be used to remove a part of a string. Putting strings together into one string is known as "string concatenation".

On Line 81, **Console.ReadLine** returns a line of text from the Console window, i.e. it brings a string representation of a number, provided the user writes a valid integer. The variable (**textValue**) that is intended to receive the return value of the ReadLine must be a `string`.

When the user writes numbers like "4", "+45" or "-99", all these are sequences of characters and thus each one is a string to the **ReadLine** method, but we need a number, an integer! Here we must convert the string to it corresponding number value, 4, -99, -99. The string values have quotation marks and numbers do not. When the user writes a number like 35 using the keyboard, ReadLine returns this input as "35"!

Conversion from one type to another can be done in many ways. For simple data types such as `int`, `double` and `bool`, you can proceed as we do it in case of **age**:

1. Save the result of ReadLine () in a string variable.
2. Convert the string value to a desired type using the types Parse method.
3. Or use the .NET object Conver.

Example to convert "string number" to `int` :

a. `int`.Parse as on Line 84 in above code
b. `int`.**TryParse** (later modules), or
c. The **.NET** object **Convert**, as follows:

```
age = Convert.ToInt32 ( textValue );
```

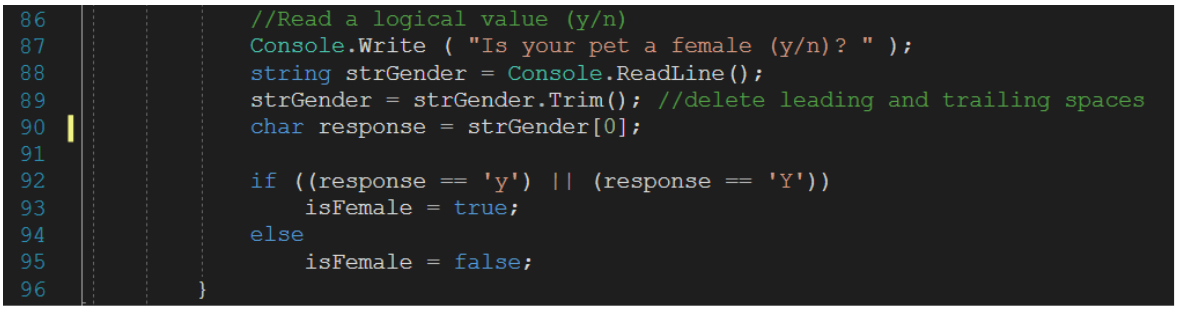
If you keep in mind and understand the above steps, you can combine them into one line and skip the need for the variable **textValue** (Line 81):

```
age = int.Parse ( Console.ReadLine ( ) );
```

Question: why do we need to declare **textValue** (Line 81) but not **age** (Line 84)?

That is because **age** is already declared as an instance variable (field) and is available for all methods in this class. If we had declared it again (int age = ….) on Line 84, we had created a new local variable! This would be a "bug" in the program. It will not then be the instance variable **age** receiving the user input.

Finally reading a `char` from the keyboard: Study the following code:



```
86              //Read a logical value (y/n)
87              Console.Write ( "Is your pet a female (y/n)? " );
88              string strGender = Console.ReadLine();
89              strGender = strGender.Trim(); //delete leading and trailing spaces
90              char response = strGender[0];
91
92              if ((response == 'y') || (response == 'Y'))
93                  isFemale = true;
94              else
95                  isFemale = false;
96          }
```

In the above code, the result of the **Console.ReadLine(),** a `string`.(Line 88) It is converted to the data type `char` by the method `char`.Parse (Line 90) and saved in a local variable **response** (by the '=' operator). Even if the user inputs a word like "Yes", it will be the first character ('Y') which will fit into the `char` variable (response in the above code). Thee method Trim() deletes all the trailing and ending blank spaces. The code **strGender[0]** picks the first character of the string saved in the **strGener**.

If the user writes a character 'y' or 'Y', or a sentence beginning with any of these chars, the answer is taken as a `true` value, according to the above code. Any other char given at the cursor position by the user will be taken as `false` (because of the `else` keyword on Line 94). The words `true` and `false` (lower case) are keywords (reserved words) in C#.

Put all the above code in the body of the method **ReadAndSavePetData** to complete the method.

Hint: As a good optimization step, you could put each reading operation in its own method.´ You could write a method **ReadName**, **ReadGender**, and a method **ReadAge**. The three methods can be then called from **ReadAndSavePetData**():

```
public void ReadAndSavePetData ( )
{
    ReadName ( );
    ReadAge ( );
    ReadGender ( );
}
```

This is a more structured way and more object-oriented.

The method: DisplayPetInfo:
The method frame:

```
public void DisplayPetInfo ( )
{

}
```

In this method, you will need to use **Console.WritLine**() to write information (output) as text to the Console window:

```
string textOut = "Name: " + name + "Age: " + age;
Console.WriteLine ( textOut );
```

To write a blank line:

```
Console.WriteLine ( );
```

I will let you write this method by yourself. Ask questions in Forum 1.

Good Luck!

Programming is fun. Never give up. Ask for help!

Farid Naisan,
Course Responsible and Instructor