Assignment 1: Help

In this first module, you will encounter many new concepts simultaneously. To assist you in getting started more efficiently, this document provides step-by-step guidance for Task 1 with some hints and guidance for Task 2 of the assignment, complementing the instructions already given in the assignment description.

In the code snippets provided, the words marked in blue are all keywords in C#. These, along with variable names and other words or combinations of words used in coding, are known as identifiers. Keywords are reserved by the language and should not be used as names for variables or methods.

Follow the steps below:

1. Create a Console Application:

Begin by starting Visual Studio and creating a new Console Application. Visual Studio will set up the solution and create the project structure, including a startup class named Program. This class contains a definition for the Main method, where your coding starts.

Name your application and your project appropriately. Right-click on each item and select the **"Rename"** option.

Before proceeding further in this class, create your first class, Pet, referromg to the class diagram presented in the assignment description.

2. Create the Pet class

Right-click on the Project name in Visual Studio, select **Add**, then **New Item** (or directly choose the **Class** item from the menu). If you select **'New Item'**, you will find the **Class** option among the templates. Name the class Pet.cs, and click **Add** to create the class.

Complete the class as follows:

a. Declare fields of class. Fields are variables used to:
   - Save values from the user (or other parts of an application).
   - Hold values needed for internal use between methods.
   - Provide output.

b. Define the methods and write code inside them to perform the operations expected when the methods are executed at runtime.

As we have discussed in our lessons so far, a class consists of fields and methods for storing and manipulating values. Every field (instance variable) should have a data type and an access modifier. It has been emphasized that fields should always be declared private. However, methods can be public or private, depending on their usefulness to other classes (public) or if they are meant for internal use within the class (private).

In the Pet class, we need variables to store user input, which include a name for a certain pet (object), the age, and the pet's gender: The name of a pet, being one or more words, should have the data type string. Age, typically a whole number, can be represented using an int for the variable **age.**

To ascertain the pet's gender, we ask the user if the pet is female, expecting a "yes" or "no" (or y/n) answer. The user's response is then converted into a Boolean value and stored in the **isFemale** variable: true for a "yes" response and false for "no". It's important to note that there are alternative ways to handle this, such as using an 'isMale' variable, or representing gender with integral constants, like using 0 and 1, or 1 and 2, to denote male and female types, respectively.

To ascertain the pet's gender, we ask the user if the pet is female, expecting a 'yes' or 'no' answer (y/n). The user's response is then converted into a Boolean value and stored in the **isFemale** variable: : true for a "yes" response and false for "no". It is important to note that there are alternative ways to handle this, such as using an **isMale** variable, or representing gender with integral constants, like using 0 and 1, or 1 and 2, to denote male and female types, respectively. We will later learn about using an **enum**, another type in C#, which is used to denote a group of related constants.

Fields: The code shown in the image lists the types of attributes we intend to store for a particular animal (pet). Consider these similar to columns in a table (form), which will be populated with specific values for each animal. If you wish to include additional attributes, you can declare more fields.

Methods: The provided code snippet defines a Start method, which is public and therefore accessible by other objects. Methods can be categorized as either void or non-void. The key distinction is that non-void methods return a value.

The first three statements inside the Start method prompt the user for information. The two methods called towards the end of the Start method are methods that we have planned to write for communication with the user.

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

        1 reference
        public void Start()
        {
            Console.WriteLine();  //blankline
            Console.WriteLine("Greetings from the Pet class!");
            Console.WriteLine();  //blankline

            ReadAndSavePetData();
            DisplayPetInfo();
        }
```

The user is someone who interacts with the program through the Console window, e.g. you when testing your program. Remember the names of the methods can be anything you like, as long as you follow the language's naming rules. It is however crucial to use names that are informative and provide a gesture of the purpose of the method.

The code snippet presented below shows how you may implement the above-mentioned two methods. Change the code if you would like to bring modifications. The methods are both used internally and could be declared as private, although they are defined as public in the code provided below.

3. The method ReadAndSavePetData:

The purpose of this method is to read user input, specifically the pet's name, age, and whether the pet is female, and then save these values into their corresponding instance variables (fields)..

```
26  public void ReadAndSavePetData()
27  {
28      //Reaad a line of text
29      Console.Write("What is the name of your pet?  ");
30      name = Console.ReadLine();
31
32      //Read age, a whole number
33      Console.Write("What is " + name + "'s age? ");
34      string textValue = Console.ReadLine();
35
36      //convert string to number
37      age = int.Parse(textValue);
38
39      //Read a logical value (y/n)
40      Console.Write("Is your pet a female (y/n)? ");
41
42      string strGender = Console.ReadLine();
43
44      //strGender = strGender.Trim(); //delete leading and trailing spaces
45      char response = strGender[0];
46
47      if ((response == 'y') || (response == 'Y'))
48          isFemale = true;
49      else
50          isFemale = false;
51  }
```

This method is a void method and is declared as public, enabling it to be called from the Main method, if necessary. However, since the method is invoked from the Start method, it could be declared as private to restrict its access to within the class. Some methods need to be public (unlike fields, which should always be private) to allow accessibility to other classes. Methods that are not intended or required to be exposed to other classes should be declared as private.

What happens on Lines 34 and 37? On these lines, the variable **name** receives the value through the Console.ReadLine() method. ReadLine is not a void method, as it returns the value it reads.

The process for obtaining the user input for the age value is similar, but the value returned by Console.ReadLine on Line 34 cannot be directly assigned to the variable **age**, or any other variable type than a string. This is because the value is text, composed of digits. The text needs

to be converted into an integer before being stored in the **age** variable, which is of an integer type. This conversion occurs on Line 37.

A few notes:

- The reason we don't declare age again (e.g., int age = ...') on Line 37 is that age is already declared as an instance variable (field) and is available to all methods in the class. Declaring it again would create a new local variable, leading to a programming error. It would mean that the instance variable **age** is not the one receiving the user input. The same is true for the variables **name** and **isFemale**.
- The variable **name** is a string, and so is the value returned by Console.ReadLine(). Therefore, no conversion is required.
- The variable **age** is an integer (int), while Console.ReadLine() returns a string representation of the numerical value entered by the user. This string must be converted to an integer before it can be saved in the **age** variable. Variables can only store values of their own type.

- The variable **isFemale** is a Boolean, which can only store true or false. The program, however, allows the user to input a 'y' or 'n' character (instead of true and false), which then needs to be translated into true or false, respectively.

  In the example, the user's response is taken as a string value instead of a char, even though only one character is needed and expected. Reading a string is easier than reading a char from the Console Window. A string can be empty, have one character, or many characters. In all cases, a string contains a sequence of chars represented by an array (a subject to be covered in a later module). The first character of the string is fetched using the index [0] as shown in the code below, to determine if it can be considered a 'y' or 'n', i.e., a true or false value.

```
string strGender = Console.ReadLine();
 // strGender = strGender.Trim(); // delete leading and trailing
spaces
 char response = strGender[0];
 if ((response == 'y') || (response == 'Y'))
     isFemale = true;
 else
     isFemale = false;
```

- The local variable **strGender** is a string that contains the text entered by the user at the cursor position in the Console window, up to the point when the Enter key on the keyboard is pressed. The text may consist of one or more characters. The first character is retrieved using the statement:

```
char response = strGender[0];
```

- Sometimes, the user might accidentally include extra spaces at the beginning or end of a string. These can be removed by using the **Trim** method of a string variable, as shown in the code below:

```
strGender = strGender.Trim();
```

  However, this line is commented out in the above code. If you wish to use it, you need to uncomment the line..

- The if statement checks if the resulting character (**response**) is a "y", in which case **isFemale** is set to true. All other values (else) of **response** are considered false. While you could handle cases where **response** is "n" or something else, we apply this simplification since we have not yet covered conditional statements in detail.

An alternative approach to handling the control statement is to first convert the string **strGender** to lowercase and then compare it with "y" (lowercase y), as shown in the code below:

```
string strGender = Console.ReadLine();
strGender = strGender.ToLower();

char response = strGender[0];
if (response == 'y')
    isFemale = true;
else
    isFemale = false;
```

Alternatively, you could convert strGender to uppercase and compare it with a capital "Y".

4. The method DisplayPetInfo:

This method is responsible for displaying the values entered by the user back to them as text in the Console window presenting the information in a formatted manner.

```
57  public void DisplayPetInfo()
58  {
59      Console.WriteLine();
60      Console.WriteLine("++++++++++++++++++++++++++++++++");
61
62      string textOut = "Name: " + name + " Age: " + age;
63      Console.WriteLine(textOut);
64      if (isFemale == true)
65          Console.WriteLine(name + ": She's such a wonderful pup!");
66      else
67          Console.WriteLine(name + ": He's such a wonderful pup!");
68
69      Console.WriteLine("++++++++++++++++++++++++++++++++");
70      Console.WriteLine();
71  }
72
```

- Programmers often use an alternative syntax to express if an expression is true by omitting the "== true" part. The statement on line 64 can be commonly written in this simplified form, conveying the same meaning:

```
if (isFemale)  //is the same as if (isFemale == true)
```

- To combine (concatenate) two or more substrings into one string, the '+' operator is used. The statement on line 62 produces a single string that includes literals enclosed within quotation marks, like "Name: ", along with spaces, symbols, etc., and values taken from variables. It is important to note that variables are not enclosed in quotation marks; if they were, they would be treated as literals. On line 62, four substrings are concatenated and stored as a single string in the variable **textOut**

After completing the Pet class, proceed to the **Program** class and write code to instantiate an object of the Pet class, then call the Start method to test the functionality of the class.

Optimization Suggestion:

As an important optimization step forward, the above method could be broken down into smaller methods, each handling a specific instance variable. This would organize the code in a more object-oriented manner:

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

The reason this is not implemented in the previous example is to avoid making it more complex for beginners.

The Program class:

Every C# program must have a class that contains a method named **Main**. This method starts the program, and the code written within it is executed automatically, line by line.

```
3  namespace ConsoleApps
4  {
       0 references
5      class Program
6      {
           0 references
7          static void Main(string[] args)
8          {
9              SetupConsoleWindow();
10
11             Console.Title = "My Favorite Pet";
12             //Create a pet object
13             Pet petObj = new Pet();
14
15             //Call a method of the object to run
16             petObj.Start();
17
18             //Continue with other classes as in above
19
20             Console.WriteLine("Press Enter to start next part!");
21             Console.ReadLine();
22         }
23
           1 reference
24         static void SetupConsoleWindow()
25         {
26             //Arrange the Console Window
27             Console.BackgroundColor = ConsoleColor.White;
28             Console.Clear();  //Paint the background with above color
29             Console.ForegroundColor = ConsoleColor.Black;
30             Console.Title = "My Console Classes";
31         }
32     }
33 }
```

In order to be able to use another class and call its methods, we must create an object (instance) of that class. In the example code above, an object of the Pet class is created on Line 13, and then the object's Start-method is called on Line 16. The statements written in the Start method

will then execute from top to bottom when the program is executed. After the execution of the Start class is complete, the execution returns and continues with the next statement on Line 20.

At the start of the Main method, there's a call to the **SetupConsoleWindow** method, which contains code for modifying the appearance of the Console Window (the output window). You have the option to either modify the implementation of this method or choose not to use it, sticking with the default settings. This method is defined as static because the Main method, which is a static method, can only call other static methods. If you had declared an instance variable (field) in the Program class, it would also need to be defined as static to be used in static methods. As a general rule, static methods can only interact with static fields and methods. However, this rule does not apply to local variables, hence the **petObj** variable.

The class TicketSeller:

Using the pattern and procedure detailed in this help document so far, you should be able to create this class, declare its fields, and write the necessary methods, and therefore no further guidance is provided for this class. However, you are welcome to use the forum for the module to ask questions and discuss any problems you encounter.

- Note that both the **Pet** and **TicketSeller** classes are encapsulated within the same namespace as the **Program** class. A namespace serves as a designation for a group of related classes, allowing different namespaces to contain classes with the same names. An application can consist of one or multiple namespaces.

- As a final point regarding this class, consider defining all methods of the **Pet** class, except for the **Start** method, as private.

- Be consistent in declaring classes either with the modifier public or without it (or internal) for all your classes. For instance, use either public class Pet or simply class Pet. If you use public with the **Pet** class, then the **Program** class must also be declared public. If you choose not to use public with your class names, the default access modifier internal will be applied by the compiler. This makes the class accessible to all classes in the same application.

Now, following the above guidelines, you should be able to complete the next task independently without needing further assistance.

Good luck.