A Simple Interactive Program

This lesson will discuss using command line to pass input to a program in C#

Accepting Command-Line Input

In the example, in the previous lesson you simply ran the program and it produced an output. However, many programs are written to accept **command-line** input.

Down below is an example of a more interactive sample program which takes input using the command line.

In this example, you'll be required to enter your **name** which will then be *displayed* when you run the code.

> **Note:** Before you run the code, click on the `>_STDIN` button present beside the `RUN` button in the code widget. Write your **name** in the input bar that appears and then click `RUN` button to see the output.

```
// Namespace Declaration
using System;

// Program start class
class Name
{
    // Main begins program execution.
    static void Main()
    {
      // Write to console
      Console.WriteLine("Please enter your name:");
      // Read from console
      string name = Console.ReadLine();
      // writing the string and additional argument to console
```

```
        Console.WriteLine("Hello, {0}", name);
        //keep program running till user clicks run
        Console.ReadLine();

    }
}
```

The `WriteLine` *method* is actually a bit more complicated than it may seem at first. The `WriteLine` method actually works similar to the printf function from C++.

- It takes a *string* and a *variable* number of additional arguments.
- The *first* argument is known as the *format* string.
- The `WriteLine` method searches the *format* string for tokens of the form **{index}**, such as **“{0}”**, known as *format* items (format items may actually have a more elaborate syntax than what is indicated here, but that is ignored for now to keep things simple).
- The `WriteLine` function replaces each format item with the argument specified by the index of the format item. So, when `Console.WriteLine("Hello, {0}", name)` is called, the **“{0}”** is replaced by the name entered by the user.
- The final line of the method is a bit of a hack that ensures that the program stays running until the user hits the **“Enter”** key.

> **IMPORTANT NOTE:** All statements end with a `;`, semi-colon. Classes and methods begin with `{`, **left** curly brace, and end with a `}`, **right** curly brace. Any statements within and including `{` and `}` define a block. **Blocks** define the *scope* (or lifetime and visibility) of program elements.

Now you know the basic structure of a C# program lets delve into more details in the upcoming chapter!