Advanced Software Engineering

Laboratory Exercise

Learning outcomes.

* Familiarity with Visual Studio .NET IDE environment
* Creating simple applications containing data and methods.
* Display message boxes and receive user input.
* Converting data types.
* Instantiating objects and calling methods.

***This is formative assessment.***

**Task 1. Familiarity with Visual C#.**

[**I have used Community Edition for these examples, as it is free!**](https://www.visualstudio.com/vs/community/)

The labs have VS Professional. The only practical difference is the Professional version has multiple languages, which we won’t use.

Create a new project and select a Console Application. You can give your project a name and change its location at this point. This will generate a bare-bones application for you (use File->New->Project), that will compile and run. As it stands it won’t display anything. The code it generates should look something like:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

}

}

}

// Code ends

Namespaces

I was very lazy and didn’t even change the name of the project, that’s why I get “namespace ConsoleApplication1”, if you called your project anything else then your namespace will be called what you put.

A namespace is the .net equivalent of a Java package. Namespace/packages are to isolate our classes from the rest of the code universe. The goal of OO is to be able to just plug pre-existing classes together and not have to write anything (we are not there yet). However, what if I write a class called “queue” and so do you? How will we tell them apart? The answer is we put them in packages/namespaces. My queue will be in my namespace and yours will be in your namespace. Technically we use our ip address as our namespace (backwards), so mine would be:

uk.ac.leedsbeckett.mullier.<namespace>

yours would be

uk.ac.leedsbeckett.<your\_student\_id>.<namespace>

or if you owned your own domain

uk.mullier.<namespace>

We aren’t going to be unleashing our ASE examples on the world, so we don’t need to go this far now, but you could if you wanted to.

**Code**

Each time you create a new console based application, you would be advised to modify the code to look like:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Starting....");

Program app = new Program();

app.go();

}

public void go()

{

Console.WriteLine("in go");

}

}

}

An analysis of this program will tell you that a method called ***Go*** resides within the class. This method has a ***void*** return type (for those of you familiar with C/C++/Java), and no parameters. This is one place in which to add code. The reason why we have done it like this is because all programs must have an entry point, in most languages this is the “main” method. However, this is a “static” method. Static methods can be called without having to first creating an object of the class it is in. That sounds good, but this is what is termed “a static context” and you can’t do everything in a static context. So the first thing we need to do is get out of the static context.

We do this by creating an object of the class:

Program app = new Program();

This created an object of “Program” called app. (In actual fact it creates a reference called app which is attached to a nameless object but….). Also, this is why the “constructor” (a special method with the same name as the class which is used to “instantiate” an object of the class) has to have a void return type. Void means “doesn’t return anything”, such as an integer and in the case of a constructor it is already returning something, the reference to the object (literally the address of it in memory).

We then call one of the methods inside the app object. There is only one, “go”. Execution then goes inside the app object and we are no longer in a static context.

It is worth mentioning that I have written the go() method specifically to make things clearer, as there is a clear and explicit call to a go() method of an object we have created in the main() method. It would, however, be more usual to simply put the code of our go() method inside the constructor of the class. This is called when the object is made, so the effect is the same and a professional programmer is more likely to do it this way.

Next, compile and run the program using the Build and Debug menus.

**Task 2. Gaining familiarity with external libraries of code.**

**T**he next task is to gain familiarity with calling external code resident in libraries or **assemblies.** An assembly in the .net environment is simply a body of code that can be found in various forms, and which you can call (and therefore not have to design and implement). We will gain familiarity with this by using the MessageBox class, a pre-written library of code available to us.

A message box allows simple GUI’s to be created that can display or receive messages from the user.

**<versions of Visual Studio prior to 2019>**

To add the facility to display MessageBoxes in C#, using the Visual Studio IDE, take the following steps:

Right click on the project in the project explorer and select “add->reference->assemblies” (On older versions this is done by Go to the menu option project->Add Reference) and select the System.Windows.Forms.dll library. Any file ending with .dll is a dynamically linked library of code (i.e. a library that is linked to your executable file at runtime). It may be that this assembly is already added to your project (ticked) but that will be because you are using a form project, this is to add it to a console project.

**<versions of Visual Studio 2019 and after>**

This is what being a software engineer is like, things are constantly changing. Now you don’t add a reference but istead change the xml setup file of your project.

Double click on your project name in the project explorer and the xml file <project>.csproj will pop up. The first line should read:

<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">

You should then add into the <proprtyGroup> the line <UseWindowsForms>true</UseWindowsForms>

Save everything.

The equivalent in Java would have been to have added a reference to a Jar file. In both cases the library is a previously compiled set of resources, without an entry point to execute, that can be called by our program.

**Task 3. Adding program statements to create a message box.**

You will need to add the following statements to your code in an appropriate place. Hint – design your program first - think about the order of the statements by creating a flowchart of the necessary steps.

MessageBox.Show("Message", "Title",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

Since the program also makes use of an additional library, you will also need to add a using statement at the beginning of the program.

using System.Windows.Forms;

**Task 4** – Create a body mass index program. Body mass index is a measure of health and can be calculated using the following formula:

BMI =             Weight in Pounds               
 ((Height in inches x Height in inches) x 703)

or

BMI =             Weight in Kilograms               
 (Height in Meters x Height in Meters)

When run, the program should produce banded output as follows:

|  |  |  |
| --- | --- | --- |
| Adults | Women | Men |
| anorexia | < 17.5 | |
| underweight | <19.1 | <20.7 |
| in normal range | 19.1-25.8 | 20.7-26.4 |
| marginally overweight | 25.8-27.3 | 26.4-27.8 |
| overweight | 27.3-32.3 | 27.8-31.1 |
| very overweight or obese | >32.3 | >31.1 |
| severely obese | 35 – 40 | |
| morbidly obese | 40 – 50 | |
| super obese | 50 – 60 | |

HINT –You will need to make use of the following program statements to achieve this, taken from this weeks lecture notes.

Reading a string from the keyboard.

string s = Console.ReadLine();

Converting the string to a floating point number:

float f = Single.Parse(s);

Converting a floating point number back to a string:

s = f.ToString();

You can perform the same operations on integer numbers.

You will need to make use of conditional (if-else) statements to make this program work.

***Finally, make use of a loop to make the program repeatedly ask for new input.***