Practical

Classes, Object and Methods

Up until this point we have written all our code within the Main method. For small programs this is generally not an issue but once programs start to get more complex it is important that you modulise your code by placing common code or functionality into new class or methods. We will leave creating new classes for future units and concentrate more on the creating methods.

A method is a collection of code that performs a specific task and can accept variable inputs, making it useful and reusable.

There a 3 types of methods:

Example 1 : The following code prints out a heading of "----------" followed by the output of the for loop the reprints the heading of "----------".

```
using System;
namespace prac_Ex1
{
   class methods
   {
      static void Main(string[] args)
      {
         int i = 0;
         Console.WriteLine("----------");
         for (i = 0; i < 10; i++)
         {
            Console.WriteLine("i = " + i);
         }
         Console.WriteLine("----------");
         Console.ReadLine();
      }
   }
}
```

To determine what code should be placed in a method is not always clear. One rule is that where you have repeating code it is best placed in a method.

In this example we could place the `Console.WriteLine("----------");` code in a method as follows :

```
using System;
namespace prac_Ex1
{
```

```
class methods
{
   // Prints a series of characters
   static void print_lines()
   {
      Console.WriteLine("----------------------");
   }

   static void Main(string[] args)
   {
      int i = 0;

      print_lines();

      for (i = 0; i < 10; i++)
      {
         Console.WriteLine("i = " + i);
      }

      print_lines();

      Console.ReadLine();
   }
}
}
```

We have added a method **print_lines** and placed the code within the body of the method defined '{' and '}' characters. The **void** means the method does not return anything and nothing is placed inside the brackets '(' ')' of the method parameter list as it does not require any information passed to it. It just executes the code within it's body.

Then in the Main we have replaced the Console.WriteLine("----------"); with **print_lines();.**

When the program executes and gets to the first **print_lines();** code, the program jumps to the method, executes the body of the method and then returns to where it jumped from. Then when the program gets to the next **print_lines();** the process is repeated.

The main benefit of this type of programming is that I only have to change the contents of the method to change the way the programs runs rather than changing the Console.WriteLine("----------"); everywhere it was defined previously.

Example 2 : The next type of method is slightly more complex and provides more functionality to the method by returning a value to the

calling code. A method can return a bool, int, double, decimal and so on to suit the purpose of the method.

The following code contains a method that prompts the user for an integer number and returns the number (int) entered.

```
using System;

namespace prac_Ex2
{
   class methods
   {
      // Get an integer from the user
      static int get_whole_number()
      {
         string tempStr = "";
         int temp_number = 0;

         Console.Write("Enter a whole number >");
         tempStr = Console.ReadLine();
         temp_number = Int32.Parse(tempStr);
         return temp_number;
      }

      static void Main(string[] args)
      {
         int i = 0;
         int max = 0;
         max = get_whole_number();


         for (i = 0; i < max; i++)
         {
            Console.WriteLine("i = " + i);
         }

         Console.ReadLine();
      }
   }
}
```

When the programs reaches the **max = get_whole_number();** the program jumps to the **get_whole_number();** method, executes the code within the body of the method then returns the value entered and parsed by the code.

The number returned is then assigned to the variable **max** .

The for loop then loops **max** times according to what the user entered.

The method then can be used anytime you want the user to enter an integer within your program without having to reproduce the 6 lines of code contained within the method. Also at a later stage if we want to refine the data entry code, we only have to adjust the code in the method.

The **get_whole_number** function from **Example 2** contains a number of flaws with the method:

1. The prompt presented to the user is not very informative.
2. The method does not use TryParse and therefore cannot determine if the data entered is valid.

The following version addresses these flaws.

```
// Get an integer from the user
static int get_whole_number(string prompt)
{
   string tempStr = "";
   int temp_number = 0;
   // Set to false to ensure while loop loops at least once
   bool parseAttempt = false;
   while (parseAttempt == false)
   {
      // Use the prompt passed to the method
      Console.Write(prompt + " >");
      tempStr = Console.ReadLine();
      parseAttempt = Int32.TryParse(tempStr, out temp_number);
      if (parseAttempt == false)
         Console.WriteLine("Value entered is not a valid whole number");
   }
   return temp_number;
}
```

Note that we are passing a string to the method as the prompt.

Example 3. In this example we add another method that not only returns a value but we also accept data or pass data to the method that can be used within the method.

The program prompts the user for two integers, then uses a new method 'get_max' to determine which of the two values is greater, then assigns the values to max and loops the required number of times.

```
using System;
namespace prac_Ex3
{
   class methods
   {
      // Prints a series of characters
      static int get_whole_number()
      {
```

```
        string tempStr = "";
        int temp_number = 0;

        Console.Write("Enter a whole number >");
        tempStr = Console.ReadLine();
        temp_number = Int32.Parse(tempStr);
        return temp_number;
    }

    // Determine which value is greater and return that val
    static int get_max(int val1, int val2)
    {
        if (val1 >= val2)
            return val1;
        else
            return val2;
    }
    static void Main(string[] args)
    {
        int i = 0;
        int v1 = 0;
        int v2 = 0;
        int max = 0;
        // Get two values
        v1 = get_whole_number();
        v2 = get_whole_number();
        // Determine which is greater
        max = get_max(v1, v2);

        for (i = 0; i < max; i++)
        {
            Console.WriteLine("i = " + i);
        }

        Console.ReadLine();
    }
  }
}
```

The method **get_max** accepts two integers (copies) passed from the calling code **max = get_max(v1, v2);,** then uses those values within the body of the method to test which of the two is greater, then returns that value which is assigned to the variable max.

Example 4. The following example demonstrates how you can pass references to variables and further improve the functionality of your methods.

Consider the **TryParse** method we have been using in previous practicals.

```
parseAttempt = Int32.TryParse(tempStr, out temp_number);
```

You will note that we pass a string containing the characters entered by the user (**tempStr**) and a variable to put the converted data into if the function is successful (**out temp_number**). The method returns true or false if successful.

The key difference between the method we have looked at in the practical and the TryParse method is the use of the out keyword. TryParse uses a reference to a variable rather than accepting a copy of a variable, which enables the method to directly change the variable and return true or false at the same time.

Consider the following example. We need a method that accepts two variables, and swaps the variables values.

The method would be defined as follows:

```
static void swap(ref int val1, ref int val2)
{
    int temp;

    temp = val1;
    val1 = val2;
    val2 = temp;
}
```

'Ref' tells the compiler that the object is initialized before entering the function, while 'out' tells the compiler that the object will be initialised inside the function.

Question 1. Examine the following code from practical 6 and then re write the code to create and implement the following methods:

- print_menu : Prints the menu, prompts the user to enter an option and returns the option
- make_a_call : Prints "User selected to make a phone call"
- send_sms : Prints "User selected to send an SMS"
- find_contact : Prints "User selected to find a contact"

```
using System;

namespace prac_Q2
{
   class menu
   {
      static void Main(string[] args)
      {
         int response = 0;
         string tempVal = "";
         bool parseAttempt =false;
```

```
        while (response != 4)
        {
            // Display the menu
            Console .WriteLine("1. Make a phone call");
            Console .WriteLine( "2. Send an SMS" );
            Console.WriteLine("3. Find a contact" );
            Console.WriteLine( "4. Quit" );
            Console .Write("Enter your selection > ");

            // Get the users response
            tempVal = Console.ReadLine();
            parseAttempt =Int32 .TryParse(tempVal,out response);
            if (parseAttempt == false )
            {
                response = 0;
            }
            // Process the response
            switch (response)
            {
                case 1:
                    Console.WriteLine("User selected to make a phone call, option " +
response);
                    break;
                case 2:
                    Console.WriteLine("User selected to send an SMS, option " +
response);
                    break;
                case 3:
                    Console.WriteLine("User selected to find a contact, option " +
response);
                    break;
                case 4:
                    Console.WriteLine("User selected to Quit, option " + response);
                    break ;
                default:
                    Console.WriteLine("Unknown selection, option " + response);
                    break;
            }
        }
        Console.ReadLine();
    }
  }
}
```

Question 2 : Write a program that prompts for and gets two decimal numbers (val1 and val2) from the user then passes the two values to a method **subtract_negative** that performs the following:

- subtracts val2 from val1
- if result is negative returns true
- if result is positive returns false

The program should display a message to indicate if the outcomes was true or false.

Question 3 : Write a program that prompts for and gets two decimal numbers from the user (val1 and val2) then passes the two values to a method **max_swap** that determines which value is larger and swaps the value to ensure val1 is larger than val2.

Your program should print the value to the screen after calling the swap_max method to prove the swap took place.

You may use two methods, **max** and **swap**, to complete this program or you may combine the functionality into the one method **max_swap** if you wish.