Mr. Bellavia

LESSON 01 – INTRODUCTION TO METHODS

In this lesson we will explore the creation and use of custom methods containing our own blocks of code that can be called for execution.

Sections:

I. WHAT IS A METHOD?

A **method** (aka function) is a block of code that is only executed when called. So far, we have designed most of our programs into blocks of code. For example, consider the following simple program that outputs a message to the user and then generates a random number:

```
// welcome message
Console.WriteLine("Hello There World!");
Console.WriteLine("------------------");

// generate random number
Console.Write("Your lucky number for today is: ");
Random rnd = new Random();
int num = rnd.Next(1, 10);
Console.Write(num);
```

Sample **Output:**

```
Hello There World!
------------------
Your lucky number for today is: 8
```

Let's now reconstruct the above program putting these two blocks of code into separate **methods:**

```
void WelcomeMessage()
{
    // welcome message
    Console.WriteLine("Hello There World!");
    Console.WriteLine("------------------");
}

void GenerateLuckyNumber()
{
    // generate random number
    Console.Write("Your lucky number for today is: ");
    Random rnd = new Random();
    int num = rnd.Next(1, 10);
    Console.Write(num);
}
```

The above shows two methods named **'WelcomeMessage'** and **'GenerateLuckyNumber'.**

Let's breakdown the construction of a method. To create a method, we start with the keyword **void** followed by a **method name** of our choice with **two brackets ( ).** Then we have a set of **braces { }** which encloses the block of code that will be executed by our method:

```
void WelcomeMessage()
{
    // welcome message
    Console.WriteLine("Hello There World!");
    Console.WriteLine("------------------");
}
```

If we run the code above, nothing will happen. This is because we must **call** the method to be executed.

To call a method we simply type the **method name** followed by two **brackets ( ):**

```
void WelcomeMessage()
{
    // welcome message
    Console.WriteLine("Hello There World!");
    Console.WriteLine("------------------");
}

void GenerateLuckyNumber()
{
    // generate random number
    Console.Write("Your lucky number for today is: ");
    Random rnd = new Random();
    int num = rnd.Next(1, 10);
    Console.Write(num);
}

// call both methods
WelcomeMessage();
GenerateLuckyNumber();
```

Sample **Output:**

```
Hello There World!
------------------
Your lucky number for today is: 8
```

A final note about methods. Methods can be called as many times as needed. For example:

```
.
.
.
WelcomeMessage();
WelcomeMessage();
GenerateLuckyNumber();
GenerateLuckyNumber();
WelcomeMessage();
```

Sample **Output:**

```
Hello There World!
------------------
Hello There World!
------------------
Your lucky number for today is: 6
Your lucky number for today is: 4
Hello There World!
------------------
```

II. WHY USE METHODS?

So far, we have written huge blocks of code to run our programs. This is great for small programs, but what if our programs grow larger? How will they be maintained? Consider the following scenarios:

Scenario 1: Imagine you had a block of code to determine the winner of a tic-tac-toe game. This same block of code would need to be executed once after every player's turn. You could put this piece of code into a loop, but the more complex you make your game, the larger the blocks of code in your loop are going to become. Your loop could take up hundreds or even thousands of lines of code! It would be best to put this block of code inside a method that can be called after every player's turn.

Scenario 2: Imagine you had written multiple programs in which each program reads data from a csv file. For each program, you will need to write a piece of code that reads in a csv file. You could write this code multiple times for each program, or you can write this code once inside a method that can be called by any one of your programs. If any changes must be made to the way you read in csv files, you will only have to change this one method.

Both scenarios above demonstrate advantages of using methods in your program. Here is a list of the most common benefits for using methods/functions in your program:

- **Organization** - As programs grow in complexity, having all the code exist in one giant block becomes increasingly complicated. A method/function is almost like a mini-program that we can write separately from the main program, without having to think about the rest of the program while we write it. This allows us to divide complicated tasks into smaller, simpler ones, and drastically reduces the overall complexity of our program.
- **Reusability** - Once a method/function is written, it can be called multiple times from within the program. This avoids duplicated code and minimizes the probability of copy/paste errors. Methods/functions can also be shared with other programs, reducing the amount of code that must be written from scratch (and retested) each time.
- **Testing** - Because methods/functions reduce code redundancy, there's less code to test in the first place. Also, because methods/functions are self-contained, once we've tested a method/function to ensure it works, we don't need to test it again unless we change it. This reduces the amount of code we must test at one time, making it much easier to find bugs (or avoid them in the first place).
- **Extensibility** -- When we need to extend our program to handle a case it didn't handle before, methods/functions allow us to make the change in one place and have that change take effect every time the method/function is called.
- **Abstraction** - To use a method/function, you only need to know its name, inputs, outputs, and where it lives. You don't need to know how it works, or what other code it's dependent upon to use it. This is super-useful for making another person's code accessible.

(Taken from www.learncpp.com)

III. NESTED METHOD CALLS?

It is also possible to call one method from another method (i.e., nested method calls). For example:

```
void PrintTime()
{
    Console.Write("\nCurrent Time: ");
    Console.Write(DateTime.Now);
}
void GenerateLuckyNumber()
{
    PrintTime();

    Console.Write("\nYour lucky number for today is: ");
    Random rnd = new Random();
    int num = rnd.Next(1, 10);
    Console.Write(num);
}
void MainProgram()
{
    GenerateLuckyNumber();
}
MainProgram();
```

Sample **Output:**

```
Current Time: 2022-05-30 8:25:23 PM
Your lucky number for today is: 6
```

Notice that we first call **MainProgram().** Then MainProgram() calls **GenerateLuckyNumber().** Finally, GenerateLuckNumber() calls **PrintTime().**