* **Level 3 (Intermediate Programs)**

**Coding Program (FibonacciSeries)**

**Pi Match**

A Pi Match score is defined by how close the digits in a number are to the approximation of Pi, 3.14. In order to calculate the Pi Match score for a number, we drop the decimal from the Pi approximation, leaving us with 314 and we start from the leftmost digits of the number and calculate the difference between those three digits and 314. We keep shifting over one digit to the right and performing the same calculation until we do not have 3 digits left in our number. The total score is the average of each 3 digit score. If a number has less than 3 digits it should have a score of 0. You can assume that the number will have at most 12 digits.

**Example 1:** For the number 3149 the Pi Match score is calculated as follows (314 - 314 = 0) + (149 - 314 = -165) / 2 = -82.5 **Example 2:** For the number 357878 the Pi Match score is calculated as follows (357 - 314 = 43) + (578 - 314 = 264) + (787 - 314 = 473) + (878 - 314 = 564) / 4 = 336

**Word score**

You are tasked with writing the scoring algorithm for a simple word game. In this game, players will submit their word and receive back the score of their word. The goal of the game is for the players to try and figure out how to make the highest scoring words based on the scores they are getting back. In order to score a word, you will add up the point value of each letter in the word. The word doesn't have to be a real word, just a string of characters. ***The scoring system for the game is:*** *F = 3, J = 6, X = 12, A, I, E, O = 2, T = 5 and all other letters are 0.* It does not matter if the letter is upper or lower case. You can ignore any non letter input in the string and assume the string will be at most 50 characters long. **Example 1:** For the string "XRay Machine" The score would be: 12 + 0 + 2 + 0 + 0 + 0 + 2 + 0 + 0 + 2 + 0 + 2 = 20 **Example 2:** For the string "Jabbt" The score would be: 6 + 2 + 0 + 0 + 5 = 13

**Title Capitalization** Many writers are often confused by the different methods of capitalizing titles. There are several forms of capitalization rules, but one of the most popular is called "title case" or "up style." Implement a function that will take a title in the form of a string and return the string with the correct capitalization for a title according to these rules. **Always capitalize the first word in a title. Always capitalize the last word in a title. Lowercase the following words, unless they are the first or last word of the title: "a", "the", "to", "at", "in", "with", "and", "but", "or." Uppercase any words not in the list above.** *(This is not the exact set of rules, but pretty close)* A word is defined as a series of non-space characters. **Example 1:** "i love solving problems and it is fun" Would return "I Love Solving Problems and It Is Fun" **Example 2:** "wHy DoeS A biRd Fly?" Would return "Why Does a Bird Fly?"

**Basic Calculator**

**Windows service that pops up on the bottom right hand corner on my birthday**

**Compare directories**

**Generate an Xml File and Validate it**

**Send an automatic email**

**Generate Password**

**Contact list**

**Converter (seconds to minutes, hours to days etc)**

**Notepad**

**Get IP Address**

**Create a login screen**

**Create a rock pater scissors**

**Voice recorder application**

**StopWatch:** Design a Stopwatch Design a class called Stopwatch. The job of this class is to simulate a stopwatch. It should provide two methods: Start and Stop. We call the start method first, and the stop method next. Then we ask the stopwatch about the duration between start and stop. Duration should be a value in TimeSpan. Display the duration on the console. We should also be able to use a stopwatch multiple times. So we may start and stop it and then start and stop it again. Make sure the duration value each time is calculated properly. We should not be able to start a stopwatch twice in a row (because that may overwrite the initial start time). So the class should throw an InvalidOperationException if its started twice. 1 Educational tip: The aim of this exercise is to make you understand that a class should be always in a valid state. We use encapsulation and information hiding to achieve that. The class should not reveal its implementation detail. It only reveals a little bit, like a blackbox. From the outside, you should not be able to misuse a class because you shouldn’t be able to see the implementation detail.

**StackOverflow** Design a class called Post. This class models a StackOverflow post. It should have properties for title, description and the date/time it was created. We should be able to up-vote or down-vote a post. We should also be able to see the current vote value. In the main method, create a post, up-vote and down-vote it a few times and then display the the current vote value. In this exercise, you will learn that a StackOverflow post should provide methods for up-voting and down-voting. You should not give the ability to set the Vote property from the outside, because otherwise, you may accidentally change the votes of a class to 0 or to a random number. And this is how we create bugs in our programs. The class should always protect its state and hide its implementation detail. Educational tip: The aim of this exercise is to help you understand that classes should encapsulate data AND behaviour around that data. Many developers (even those with years of experience) tend to create classes that are purely data containers, and other classes that are purely behaviour (methods) providers. This is not object-oriented programming. This is procedural programming. Such programs are very fragile. Making a change breaks many parts of the code.

**Design a Stack** A Stack is a data structure for storing a list of elements in a LIFO (last in, first out) fashion. Design a class called Stack with three methods. void Push(object obj) object Pop() void Clear() The Push() method stores the given object on top of the stack. We use the “object” type here so we can store any objects inside the stack. Remember the “object” class is the base of all classes in the .NET Framework. So any types can be automatically upcast to the object. Make sure to take into account the scenario that null is passed to this object. We should not store null references in the stack. So if null is passed to this method, you should throw an InvalidOperationException. Remember, when coding every method, you should think of all possibilities and make sure the method behaves properly in all these edge cases. That’s what distinguishes you from an “average” programmer. The Pop() method removes the object on top of the stack and returns it. Make sure to take into account the scenario that we call the Pop() method on an empty stack. In this case, this method should throw an InvalidOperationException. Remember, your classes should always be in a valid state and used properly. When they are misused, they should throw exceptions. Again, thinking of all these edge cases, separates you from an average programmer. The code written this way will be more robust and with less bugs. The Clear() method removes all objects from the stack. We should be able to use this stack class as follows: var stack = new Stack(); stack.Push(1); stack.Push(2); stack.Push(3); Console.WriteLine(stack.Pop()); Console.WriteLine(stack.Pop()); Console.WriteLine(stack.Pop()); The output of this program will be 3 2 1 Note: The downside of using the object class here is that if we store value types (eg int, char, bool, DateTime) in our Stack, boxing and unboxing occurs, which comes with a small performance penalty. In my C# Advanced course, I’ll teach you how to resolve this by using generics, but for now don’t worry about it. 2 Real-world use case: Stacks are very popular in real-world applications. Think of your browser. As you navigate the web, the address of each page you visit is stored in a stack. As you click the Back button, the most recent address is popped. This is because of the LIFO behaviour of stacks.