Mitrais Developer Test - Timothy Ryan S

The first reason why I chose this algorithm is easy to understand. The level of easy means that any experience level developer can understand it. By writing easy understandable code, We can take less time to understand the code logic. As a result, if there is a bug it can be easier to locate the bug cause and fix it. It applies when we want to do some enhancement too. The reason I rejected the other approach is it takes more time to understand the flow and higher risk of bugs occured when developing the code due to comprehensive flow.

The main approach is by grouping the functions by the following division of number per unit, tens,hundreds, and more than hundreds (thousands,millions,etc). By grouping the function into this, we can be more specific to develop the code on each function. At the end, those number division functions will be collected into a function. The function will collect input parameters, which are numbers that will be converted into words, currency, and penny. All of those functions are written in one class object named MoneyWords. Of course by its name, this class has one specific purpose: to convert numbers into words.

A function that was first developed is for units, because the unit contains unique numbers. For example: 1 is ‘one’, 2 is ‘two’ and so on until 19 (nineteen). After that, the next one is for tens. In this function, a number needs to be divided by 10 to find it’s tens. If the result is below or an equal one, we call the unit function. If the result is more than one and below or equal to 9, we separate each result into conditionals to generate the words. For example: 2 is ‘twenty-’, 3 'thirty-’ and so on.

After that, we need to check if the inputted parameter has a unit remaining or not. If there is none remaining, we need to remove the “-”. It is useful to anticipate if there are none remaining. For example: 20 will become ‘twenty’ instead of ‘twenty-’. 21 will become ‘twenty-one’. But, if there is remaining, we need to call unit function to convert the remaining into words. Here, we can be 100% sure that the remaining is below 10 due to mod.

The next one is hundreds function. In this function, inputted parameters will be divided by 100 and also moded by 100. If the divided number is more than 0, it means the number has hundred, so we need to call unit function and add the “hundred and”. After that. We check the modded number if there is, we call tens functions. The last one if the divided number is more than 0 but there is no remaining then we need to remove the word “and”.

The next one is all functions, this will be the first called function before calling the other functions. In this function, we created 2 arrays of number scale value and name (e.g: value = 1000, name=”thousand”, value=1000000, name=”million”). After that, we do a loop to find the current digit of the number by dividing inputted parameter with a value in an array value. If the result word output is more than 0, we can append the word “,” into result word output. Then, we call hundreds function and add the array name of the current scale. For example: inputted parameter is 12345; we do a loop to find if the inputted parameter matches with some scale in the array. In that case, we find that it matches the “thousand” scale, so we get the number of thousand first (12345/1000=12). So we call hundreds function to get the word of “12” and add the current scale name on it (“thousand”). After that, we subtract the inputted parameter with the current number of thousand times current scale, and find the word of the remaining numbers.

The next one is currency function. In this function, we check if the numbers are negative, then we add “minus” word in the result and make the numbers into positive by multiplying by -1. After that, we divided the number to find dollars and cents and call all functions for each of them. If there are any cents then we append the word result from penny to the main result one. In certain

conditions if the number of dollars is not 1, then we change the word dollar into dollars. That applies to cents too.