# Tech Test Thought Process – Benjamin Burn

## Main

In Main.java first I created the parseArgs() function to check the user input and exit on error. After I initialise the Converter class with the number given by the user. Next call methods from Converter and print out the corresponding number-to-word conversion.

## Converter

First, I created a HashMap to store number, string pairs for all possible numbers, for example: (1, "ONE"), (12, "TWELVE"), (40, "FORTY"), this way I can construct any number between 0 and 999999999, which I chose as my top limit though it can be increased easily. Next I created a sanitiseNum() function which truncates/appends 0's, along with other checks, to the input string to allow for consistent operations. To complete the task, I knew that some kind of iteration was required since I needed to detect each number and the position it was in.

Originally, I was going to use a While loop and iterate over the string and store the value + name as a tuple for each number for cents and dollars, for example 12.25 -> DOLLARS[(1, "TEN"), (2, "ONE)], CENTS[(2, "TEN), (5, "ONE")]. This gave me all the required information but was cumbersome and required a second loop over the information.

Instead, I decided to split the input string on '.' So I could operate on DOLLARS and CENTS in separate functions. This skipped the unnecessary initial loop and allowed me to start directly converting the number. For CENTS this was fine since there are only 2 positions with not much logic to check if the number is < 10, between 10 and 19 (inclusive) or >= 20. On the other hand, DOLLARS required much more complex logic since I needed to handle many more cases. For example, 0 – 100 had to be checked once for each 10^3 increase of the number so once for 123, twice for 123123 and three times for 123123123. This along with other cases led me to believe that this method, though possible, would be too time consuming and not an ideal or fast solution.

So far, I had only been operating on strings and since this is slow, I wondered if I could operate arithmetically instead. If you have a number and take the floor of its division by the lowest order version of that number, you get the number of 'orders' in that number (for example 225 / 100 = 2), furthermore if you take the modulus of the original number, you get what remains (for examples 225 % 100 = 25). Using these two equations one can calculate all required parts of the number, where modulus acts as the 'iterating' factor where it repeatedly shrinks the number until no remainder is left. Based on this I used the split string used above and constructed a function that checked the highest possible order (millions) to the lowest possible order (ones) and took the modulus of that order at each step. Since in the decimal system hundreds are used in each 10^3 when greater than 1000 I separated out the code that handled numbers < 1000 into a separate function and numbers < 100 into another to avoid repeating code. These functions are then used whenever necessary.

I do believe that there is a way to do this recursively by calling the function again with the argument as the number / order (integer division) but I did not have the time to figure it out.