README

Irsan Winarto

0415744603

irsanwinarto@tutanota.com

Contents

1. The Console

As per the specifications, a console has been provided. The file name is `NumberToWords.jar` and below is the usage in non-interactive mode:

```
java -jar NumberToWords.jar <your input real number>
```

The output will be printed on the next line. If more than one argument is supplied, then only the first argument will be used, and the others are ignored.

To start the console in interactive mode:

```
java -jar NumberToWords.jar
```

In interactive mode, the user can translate as many numbers as the user desires. To exit, simply issue “quit” to the prompt.

2. File Locations

This is a Maven project. The file `ConvertNumbersToWords.java` is located at the following:

```
src/main/java/irsan/winarto/solution/ConvertNumbersToWords.java
```

The test file is located at the following:

```
src/test/java/irsan/winarto/solution/ConvertNumbersToWordsTests.java
```

The console application is located at the project base directory (where the `pom.xml` file resides) with the filename `NumberToWords.jar`.

3. Modifications and Limitations

The API requirement as stated in the invitation email is

```
public string ConvertNumbersToWords(string NumberString)
```

I decided to modify the signature slightly:

```
public static String convertNumbersToWords(String numberString)
```

The case-sensitive modification here is that I use a lowercase letter as the first letter of the method name in accordance to Java naming convention.

Furthermore, the reason for declaring it `static` is that there does not seem to be a point in instantiating the class `ConvertNumbersToWords` for this exercise. Nonetheless, the class can still be instantiated.

Besides the modification above, please take note that I have chosen to spell 10^{53} as a "SEPTENDECILLION", instead of "SEPTDECILLION". There seems to be more resources online using the former term.

The limit of the size of the input number that my program accepts is 10^{63} or a "VIGINTILLION", which has 64 digits. I could not find an authoritative resource for naming scales beyond vigintillion. Hence, the limit. It is very trivial to extend my program to accept higher scales: simply add higher scales to `ConvertNumbersToWords.scaleNames` in the ascending order of the scales, and set `ConvertNumbersToWords.MAX_INT_PART_LENGTH` to the number of digits in the highest scale.

4. Algorithm

4.1 My Solution

To help explaining, this document refers to a string of **3 or less** consecutive digits as a "triple". My solution first breaks up a real number input into two parts: the integer part (before the decimal point) and the fractional part (after the decimal point).

For the integer part, it then parses every triple starting from the lease significant 3 digits by first naming it in terms of at most hundreds. For example, "2827008029" is parsed in terms of triples starting from the least significant triple digits as follows: "029" is parsed as

"TWENTY-NINE", then "008" is parsed as "EIGHT", "827" is parsed as "EIGHT HUNDRED AND TWENTY-SEVEN", and finally, "2" is parsed as "TWO".

Since every triple corresponds to a power of 1000, then appending to the tail of the triple the correct name for the power of 1000 (such as "THOUSAND", "MILLION", "SEPTENDECILLION", etc) gives the correct name for the triple.

Using the same example, "2827008029", the parse order is as follows: "029" is parsed as "TWENTY-NINE" and appended with nothing, "008" is parsed as "EIGHT" and appended with "THOUSAND" to become "EIGHT THOUSAND", "827" is "EIGHT HUNDRED AND TWENTY-SEVEN" appended with "MILLION" to become "EIGHT HUNDRED AND TWENTY-SEVEN MILLION", and "2" becomes "TWO BILLION".

Next, my program appends commas after every power of 1000, with one exception: if before the thousand, the input number does not contain hundreds, then instead of appending a comma after the thousand, it appends " AND ". Thus, for the example above, "8029" is "EIGHT THOUSAND AND TWENTY-NINE" instead of "EIGHT THOUSAND, TWENTY-NINE". Furthermore, either "DOLLAR" or "DOLLARS" are appended at the end of the translation. Therefore, the full name of the example number above is "TWO BILLION, EIGHT HUNDRED AND TWENTY-SEVEN MILLION, EIGHT THOUSAND AND TWENTY-NINE DOLLARS".

For the fractional part of the input, my program takes only the first three digits. It then rounds it to the nearest integer, with a tie (a number that ends with "5") being rounded up. After the rounding, it goes through the exact same process as the integer part, with the exception of using "CENT" or "CENTS" as the unit instead. The integer part is then combined with the fractional part using " AND " without any comma.

Cost analysis of my algorithm is as follows, with n being the number of digits:

1. First, breaking the input into the integer part and the fractional part takes $\theta(n)$ time.
2. Translating every triple, including appending the name of some power of 1000 and a comma to it, costs $\theta(1)$. Since the number of triples are either $n \bmod 3 \in \theta(n)$ or $n \bmod 3 + 1 \in \theta(n)$ (depending on whether n is divisible by 3), the total cost for translating all triples is $\theta(n)$.
3. Combining/joining the triples together costs $\theta(n)$.

Therefore, the total cost is $\theta(n)$ or linear.

4.2 Alternative Algorithms

Since every digit must be parsed to get the correct result, and no digit can be used to guess other digits in the input number, then there is simply no algorithm that is faster than $\theta(n)$. Therefore, different algorithms with the same running cost of $\theta(n)$ can differ only in the constant factors.

As described before, my algorithm parses “triples” in the order of starting from the least significant digits of the input number to the most significant digits. This is implemented using a loop. An alternative is to employ recursion. Although the recursion tree would simply be a linear graph (due to parsing in order of ascending significance) and the cost would still be $\theta(n)$, the overhead of allocating stack frames and popping them off the stack would be an extra constant cost for every “triple” in the input number. Therefore, I decided against using recursion.

5. Test Plan

As mentioned in section 2, my program can parse up to the scale of “VIGINTILLION”, which has 64 digits for the integer part. I have implemented more than 39 unit tests that cover scales from ones, tens, hundreds, thousands, tens of thousands, hundreds of thousands, millions, billions, trillions, and all other powers of 1000 up to vigintillions.

To generate proper test cases, I used a Python package called `num2words`, which is a popular package that can translate a numeral to words with the capability of parsing much higher scales than vigintillions in multiple languages, including English. However, `num2words` does not deal with zero cent properly. It even includes “zero cents” (yes, plural) in the output when the input number does not even have any decimal point. Thus, I wrote a Python script to randomly generate my test cases for all the scales mentioned on the previous page and to deal with the aforementioned quirk of `num2words`. The script has been included in my submission with the name `number_generator.py`.

Besides using num2words, I also added a few tests for edge cases such as dealing with numbers like “8029”, which should be parsed as “EIGHT THOUSAND AND TWENTY-NINE”, but “8129” is parsed as “EIGHT THOUSAND, ONE HUNDRED AND TWENTY-NINE”, i.e. the former uses the conjunction “AND” after the thousand but the later uses a comma because a hundred comes after it.

There are also other edge cases tested such as invalid input numbers, which are listed below:

1. null
2. empty string
3. a number that starts with a decimal point, e.g. “.82” (considered grammatically incorrect in English)
4. a number that ends with a decimal point, e.g. “8237823.”
5. multiple decimal points, e.g. “823782..832”
6. non-digit characters, e.g. the ‘e’ in “10e3” (this is acceptable in programming languages but is, otherwise, grammatically incorrect)
7. a number with more than 64 digits (vigintillion), which is the limit of my program

An `IllegalArgumentException` will be thrown if any of those is given as an input, with the error message displayed to the user. This is a Maven project. To run the tests, navigate to the project base directory (containing the `pom.xml` file) and issue the command `mvn test`.