# Assignment 2

DTU Compute
Technical University of Denmark
02102 Introductory Programming
14/04/2022

Group 34

Polly Clémentine Nielsen Boutet-Livoff - 214424


As I am the only member of the group, all the work was done by me.

# 1 Roman numerals

After running the main function the program will prompt the user to enter a positive integer. The program then converts it into roman numerals and prints a line showing the number in decimal and roman numerals. Roman numerals are closely related to the decimal system, so I decided to split the problem into figuring out what each decimal place would be in roman numerals. Instead of trying to convert 12 directly, I convert each decimal and then join them. 10 is X and 2 is II, so 12 is XII.

```
$ java RomanNumerals
Enter positive integer to convert: 3999
3999 = MMMCMXCIX
$ java RomanNumerals
Enter positive integer to convert: 1
1 = I
$ java RomanNumerals RomanNumerals.java && java RomanNumerals
Enter positive integer to convert: 123
123 = CXXIII
```

The program only supports positive numbers since roman numerals only represent positive numbers. Additionally it only supports numbers up to 3999, simply because Wikipedia didn't list how to write higher numbers[1].

# 2 Palindrome

The program asks for the user to input a string that it will check. The program then prints the line back to the user to ensure that there is no confusion as to what is being checked, and then states whether or not it's a palindrome. In order to check if some string is a palindrome we first sanitize the input by remove all non-letter characters (whitespaces, punctuation, etc.) and converting everything to lowercase. Afterwards, we simply check if each nth and nth last letter are the same until we've checked them all. If there's an uneven amount of letters we don't need to check the middle letter.

```
$ java Palindrome
Enter line to check: Racecar!
"Racecar!" is a palindrome!
$ java Palindrome
Enter line to check: Another one
"Another one" is not a palindrome!
$ java Palindrome
Enter line to check: Mr. Owl ate my metal worm
"Mr. Owl ate my metal worm" is a palindrome!
```

Our program works correctly for the English alphabet, however since it ignores everything that isn't a to z and assumes ASCII encoding[2]. So strings containing the danish vowels "æøå", or letters

[1] `https://en.wikipedia.org/wiki/Roman_numerals`

[2] I'm not totally sure it assumes ASCII, but regardless I can tell from testing that the regex I'm running doesn't care for unicode.

with accents are incorrectly[3] ignored. For example:

```
$ java Palindrome
Enter line to check: Dåsemad og go' dame sæd
"Dåsemad og go' dame sæd" is a palindrome!
```

The string is clearly not a palindrome as it uses different vowels in the second and second last position, however the program ignores the encoded bytes.

```
$ echo -n "Dåsemad og go' dame sæd" | hexyl
+--------+-------------------------+-------------------------+--------+--------+
|00000000| 44 c3 a5 73 65 6d 61 64 | 20 6f 67 20 67 6f 27 20 |D××semad| og go' |
|00000010| 64 61 6d 65 20 73 c3 a6 | 64                      |dame s××|d       |
+--------+-------------------------+-------------------------+--------+--------+
```

As we can see `c3 a5` and `c3 a6` are both outside of the standard ASCII definition as they are greater than `0x7f`. We could fix this issues in the software by comparing the nth and nth last unicode code point, however we would still have some edge cases such as combining diacritical marks. I believe the correct approach would be to gain a better understanding of the actual problem the function is trying to solve so that we can correctly limit it's functionality. Otherwise we might stumble into questions such as "What does a palindrome mean in Arabic script? Or using CJK characters? Or Emoji? Are Emoji punctuation? Is the Cyrillic o and the Latin o the same?", which are probably not questions we need to answer in order to correctly solve this problem.

# 3 Buffon's needles

The program asks for a number of iterations, runs a random simulation, and returns an estimate for $\pi$.

```
$ java BuffonsNeedle
Enter number of iterations: 10000000
Out of 10000000 throws, 3187584 hit an edge.
Which means we estimate pi to be 3.1371722282455927
$ java BuffonsNeedle
Enter number of iterations: 10
Out of 10 throws, 1 hit an edge.
Which means we estimate  to be 10.0
$ java BuffonsNeedle
Enter number of iterations: 2
Out of 2 throws, 0 hit an edge.
Which means we estimate  to be Infinity
```

The problem statement includes a partial description of how to simulate Buffon's needle problem that we implement. In order to simplify the code we write a helper function called `overlapsPoint` that checks if a line-segment with some width overlaps a point. We then simply generate the desired amount of needles by uniformly selecting a random double between 0 and 2, and an angle

[3]The program description doesn't specify anything with regards to encoding, so it's debatable if our program is incorrect, or if it's merely correct for a smaller set of input.

between 0 and $\pi$. It's a bit weird that we include $\pi$ in our program to guess $\pi$, but that's most straightforward way to write the program; In order to compute whether our needles intersect with the lines we use cosine, which needs the values to be in radians. Alternatively, we could've picked a float between 0 and 180 as the problem statement recommends, but that would entail converting it so radians (and thereby using pi) in order to use `Math.cos` or using an alternative cosine function. This in turn is possible but needlessly complicates the program and violates the KISS principle. If we're unlucky enough that 0 needles hit and edge, $\pi$ is esitmated to be "Infinity", which might not be the desired functionality. It is very hard to imagine a practical reason for this program to exist, so therefore I deem this functionality behavior to be reasonable and not a bug. In the extreme where every needle hits, $\pi$ is estimated to be one.