Exercise 1

Write a class `FloatAndDouble` with a method `computeBiggestEpsilon` that takes two `double` numbers x and x_0 as arguments, computes the biggest $\epsilon = 2^{-n}$, $n \in \mathbb{N}$ such that $|x - x_0| \geq \epsilon$ and returns n and ϵ. In a `main` method, let $x = 3 \cdot 0.1$ be of type `double` and $x_0 = 0.3$ be a `float`. Check that $x_0 \neq x$. Call the method above for x and x_0. Repeat the analysis by now declaring x_0 to be of type `double`.

Exercise 2

Write a class `McLaurinCosine` with a `static` method which approximates $\cos(x)$ for $x \in (0, \pi)$ by means of a McLaurin series expansion, i.e.,

$$\cos(x) \approx \sum_{k=0}^{n} \frac{(-1)^k x^{2k}}{(2k)!},$$

taking as arguments x itself and the order n. In the computation of the expansion, store here the factorial as an `int`.

Also write a method which takes as argument an integer value `maxOrder` and prints the absolute difference between your approximation and the value returned by `Math.cos(double x)`, for increasing order from 1 to `maxOrder`. In a `main` (this can also be part of the class `McLaurinCosine`) call this last method for $x = 3$ and maximum order 15. What does it happen here? Try to get what's the problem and to fix it.

Optional: you might consider to implement now the approximation of $\cos(x)$ also for values of x not in the interval $(0, \pi)$, basing on the implementation of the method you have written.

Exercise 3

Write two methods `floatHarmonicSumForward` and `doubleHarmonicSumForward` which compute the harmonic sum

$$1 + \frac{1}{2} + \cdots + \frac{1}{N},$$

using single precision (`float`) and double precision (`double`) respectively, and compare the result that you obtain for large values of N. Write two more methods `floatHarmonicSumBackward` and `doubleHarmonicSumBackward` that still compute the harmonic sums but going backwards, i.e.

$$\frac{1}{N} + \cdots + \frac{1}{2} + 1.$$

Compare the results with the ones obtained going forward, again for large values of N. What did it happen?