Exercise 1

Write a `JUnit test case` called `EnhancedValueTestWithLoop`, possibly along the lines of `EnhancedValueDoubleDifferentiableTest`, with a method `TestAdditionWithLoop`. Here you test the class `EnhancedValueDifferentiable` we have seen last time by computing the derivative with respect to x of the function $f(x) = \sum_{i=1}^{n} x$, for n fixed: construct it as a `for` loop, then check that the value is nx and that the derivative is n, both up to a tolerance equal to the machine precision.

Exercise 2

This exercise (or some points of this exercise) may be done during the Tutorium

Write another `JUnit test case` that you now call `StochasticAutomaticDifferentiationTest`, whose goal is to test the implementation of the backward and forward Stochastic algorithmic differentiation in the `Finmath` library. We want to see how these approaches apply to objects of type `RandomVariable`, test the result they provide when computing the derivative of an operation and also test their performances. In particular, this test will be based on the classes `net.finmath.montecarlo.automaticdifferentiation.backward.RandomVariableDifferentiableAD` and `net.finmath.montecarlo.automaticdifferentiation.backward.RandomVariableDifferentiableAAD`, both implementing the interface

`net.finmath.montecarlo.automaticdifferentiation.RandomVariableDifferentiable`.

Do the following:

(a) Write a `private` method `constructAndReturnRandomVariable()` that returns a `RandomVariable` object, which wraps an array of `doubles` whose length is given by a field of the class and whose entries are random numbers, uniformly distributed between 0 and 2.

(b) Call this method in order to construct two objects `xValue` and `yValue` of type `RandomVariable`, with such random entries. Create an object of type `RandomVariableDifferentiableAAD`, representing the random variable `xValue`, by typing `RandomVariableDifferentiable xBackward = new RandomVariableDifferentiableAAD(new RandomVariableDifferentiableAAD(xValue))`.
Same thing for `yValue`, represented by an object `RandomVariableDifferentiable yBackward`. Do then the same for the forward differentiation, by creating the objects `xForward` and `yForward`. Have a look at how these objects are created in the respective classes.

(c) Create a tree representing the operation $f(x, y) = e^{x^2+xy^2}$, first for the backward and then for forward case: for the backward case the leaves will be `xBackward` and `yBackward`, for the forward case `xForward` and `yForward`. The highest nodes of the trees will be called `resultBackward` and `resultForward`, respectively. Also create two fields `Map<Long, RandomVariable> backwardGradient` and `Map<Long, RandomVariable> forwardGradient`;[1]

(d) Write two `private` methods `getBackwardGradient()` and `getForwardGradient()`, which assign to `backwardGradient` and `forwardGradient` the gradient computed with the backward and with the forward method, respectively, and return a `long` representing the time needed for computing the gradient.

(e) Write a method `public void testRepeatedly()`, where you call both the methods above 1000 times, and you count and print how many times `getBackwardGradient()` returned a time smaller than `getForwardGradient()`, and print the average time returned by both the methods.

[1] Note that point 2 and point 3 can be done by coding outside any specific method

(f) Write two `void` methods `testForwardDifferentiation()` and `testBackwardDifferentiation()`, where you test the derivative computed with the forward and backward implementation, respectively. In particular, get the derivative of $f(x, y) = e^{x^2+xy^2}$ with respect to x (you have to see which method of teh `Finmath` classes to use), print its average (it is a `RandomVariable`!) and check if all its realizations are equal (again, up to a very small tolerance) to the realizations of the `RandomVariable` object representing the analytic derivative.