Exercise 1

This exercise might be done during the Tutorium.

This exercise gives an example of the computations of prices of options via Monte-Carlo. The idea is that, exploiting the results you have seen in the lecture, one can approximate the price of an option as

$$\frac{1}{n}\sum_{i=1}^{n} p_i,$$

where $p_i = p(\omega_i)$, $i = 1, \dots, n$ for large n, are different realizations of the payoff of the option.

Write a class `DigitalOption` implementing the interface

`com.andreamazzon.exercise2.EuropeanTypeOptionMonteCarlo`.

There you find two methods, i.e., `getPayoff(StochasticProcessSimulatorInterface underlyingProcess)` and `getPrice(StochasticProcessSimulatorInterface underlyingProcess)`, whose description you find in the interface, that return the realizations of the payoff at maturity time and the Monte-Carlo price (i.e., the average of the payoff realizations), respectively, of an European option whose underlying is described by the object `underlyingProcess`.

Note that `StochasticProcessSimulatorInterface` is the interface you can find in the Java course project in the package

`com.andreamazzon.session5.abstractclasses.simulators`.

You have here to implement these two methods for a digital option, i.e., an option that for an underlying S at maturity time i has payoff 1 if $S(i) - K > 0$ and 0 vice versa, where $K > 0$ is a given strike.

Test your code by creating an object of this class with maturity 7 and strike 100, and let it call the method `getPrice` by giving it an object of type

`com.andreamazzon.session5.abstractclasses.simulators.BinomialModelSimulator`,

with initial value 100, $u = 1.5$, $d = 0.5$, interest rate 0, final time 7, number of simulations 100000 and seed of your choice. The price you get should be approximately equal to 0.22.

Hint: what you can do to implement `getPayoff` is basically to see which method(s) of the interface `StochasticProcessSimulatorInterface` might help you, and make `underlyingProcess` call it/them. Remember that, due to polymorphism, `underlyingProcess` can be of whatever class implementing `StochasticProcessSimulatorInterface`. The method `getPrice` might rely on the call of `getPayoff`.

Exercise 2

In this exercise, we want to do some experiments regarding the Monte-Carlo computation of the prices of the digital option implemented above when the underlying is a binomial model.

Have a look at the interface

`com.andreamazzon.exercise2.EuropeanTypeOptionMonteCarlo`.

This interface has two methods, namely `getMinAndMax(int numberOfPriceComputations)` and `getHistogram(int numberOfBins, int numberOfPriceComputations)`.

For both the methods, `numberOfPriceComputations` is the length of an array of Monte-Carlo prices of a general option. The values of this array represent all the prices we obtain when we give to the option an underlying which describes always the same process but simulated with different seeds. Indeed, every time we simulate the underlying with a different seed, we obtain different realizations of the underlying, then different realizations of the payoff of the option, and then a different price.

The first method computes the minimum and maximum values of the array, whereas the second one returns an object of a container class representing an histogram which describes the distribution of the prices. This class is `com.andreamazzon.exercise2.HistogramData`, you can have a look at it to see a description of its fields and its `getters` and `setters`.

Write a class `MonteCarloExperimentsWithBinomialModel` implementing this interface, in the case when the underlying is a binomial model, represented by an object of type

`com.andreamazzon.session5.abstractclasses.simulators.BinomialModelSimulator`,

and the option is the digital option we have implemented above.

Have then a look to the class

`com.andreamazzon.exercise2.MonteCarloTest`

and run it. The only thing you have to do here is to call the constructor of your class `MonteCarloExperimentsWithBinomialModel` when indicated.

Hints:

- You can construct the array where you store the prices of the option inside the methods you have to implement, for example calling the method `getPrice` of `DigitalOption` inside a `for` loop, giving it at every iteration of the loop an object of type

  `com.andreamazzon.session5.abstractclasses.simulators.BinomialModelSimulator`,

  with always same parameters (which might be given in the constructor of your class) but *random* seed, given for example by the `nextInt()` method of an object of type `java.util.Ramdom`.

- In order to get minimum, maximum and histogram of the array, you can use the methods `getMin(double[] vector)`, `getMax(double[] vector)` and `buildHistogram(double[] realizations, double minBin, double maxBin, int binsNumber)` that you can find in

  `com.andreamazzon.session4.usefulmatrices.UsefulMethodsMatricesVectors`.

  In order to do that, you have to pull the project of the Java course.