Exercise 1

Let $S^1 = (S^1_t)_{t\in[0,T]}$ and $S^2 = (S^2_t)_{t\in[0,T]}$ be two assets following the risk-neutral dynamics

$$dS^1_t = rS^1_t dt + \sigma_1 S^1_t dW^1_t, \quad 0 \le t \le T,$$
$$dS^2_t = rS^t_2 dt + \sigma_2 S^2_t dW^2_t, \quad 0 \le t \le T,$$

for constant volatilities σ_1, σ_2 and correlated Brownian motions $\langle W^1, W^2 \rangle_t = \rho t$, $\rho \in [-1, 1]$.
An exchange option for S^1 to S^2 with maturity T is a product that pays $(S^1_T - S^2_T)^+$ at time T. It can be seen that the value at time 0 of the exchange option is

$$C^{BS}(S^1_0, S^2_0, \sigma, T),$$

where $\sigma = \sqrt{\sigma_1^2 - 2\rho\sigma_1\sigma_2 + \sigma_2^2}$ and $C^{BS}(S, K, \sigma, T)$ is the Black Scholes value of a Call option of spot price S, strike K, volatility σ and maturity T.
Give the implementation of the class `ExchangeOption` that you find in

`com.andreamazzon.exercise10.products`

and that implements

`net.finmath.montecarlo.assetderivativevaluation.products.AbstractAssetMonteCarloProduct`.

Its

`getValue(double evaluationTime, AssetModelMonteCarloSimulationModel model)`

method returns the (discounted) payoff of an exchange option.
Complete the `JUnit` test class

`src/test/java/com.andreamazzon.exercise10.products`

which has two methods that check if the value you get is close enough to the analytical one. Follow the instructions in the code.

Exercise 2

Give the implementation of the class

`com.andreamazzon.exercise10.products.GeneralOption`,

which also extends `AbstractAssetMonteCarloProduct`. In this class we want to compute the value of an european option whose payoff is a general function of the terminal value S_T of a one-dimensional underlying. The function must be specified by the user when constructing an object of that class.
Complete the `JUnit` test class

`src/test/java/com.andreamazzon.exercise10.products.GeneralOptionTest`:

here we want to construct an object of type `GeneralOption` specifying a payoff function $f(x) = (x - K)^+$ for a given value of K, make it call the method `getValue(MonteCarloBlackScholesModel)` and test if the value we obtain is the same, up to a (very small) tolerance, as the one we get when you value the option by using the class

`net.finmath.montecarlo.assetderivativevaluation.products.EuropeanOption`.

Follow the instructions in the code.