Exercise 1

Have a look at the interfaces

`MonteCarloEvaluationsInterface`

and

`MonteCarloEvaluationsWithExactResultInterface`.

you can find in the package

`com.andreamazzon.handout4.montecarloevaluations`.

The first interface provides methods to experiment with the Monte-Carlo implementation of a possibly general class of problems, from pricing to the computation of an integral. The second interface extends the first one. For this second interface, we suppose that we already know the exact value of the quantity we approximate by Monte-Carlo, so that we can compute the absolute errors of our approximations.

Write two classes `MonteCarloIntegrationPowerFunction` and `MonteCarloPi` implementing this second interface and computing the integral $\int_0^1 x^\alpha dx$, $\alpha \in (0, \infty)$, in the case of `MonteCarloIntegrationPowerFunction`, and π in the way you have seen in the lecture in the case of `MonteCarloPi`.

Test then your results by running the classes

`com.andreamazzon.handout4.montecarlopi.MonteCarloPiCheck`

and

`com.andreamazzon.handout4.montecarlointegration.MonteCarloIntegrationCheck`:

note that these two classes will give errors until you construct your classes.

Hints:

- Here you have total freedom about the choice of your design. A suggestion is: you can note that the implementation of almost all the methods of the interfaces does not depend on the specific Monte-Carlo computation, as they might use the vectors hosting the computed values. You can think to implement these methods in some abstract class / abstract classes implementing the correspondent interface(s), and then let `MonteCarloIntegrationPowerFunction` and `MonteCarloPi` extend this class / one of these classes. In this way, they would automatically implement the interface(s).
- Consider that, with respect to the methods of the interface

  `com.andreamazzon.handout3.MonteCarloExperiments`,

  the methods of the present interfaces do not take the number of Monte-Carlo computations as argument. So this should be a field of your classes. You can decide which access modifier to assign it according to the design of your solution (i.e., if you have or not the parent abstract classes, for example).
- You can find some methods to perform your experiments (for example, computing the standard deviation, or the average error, of the Monte-Carlo computations) in the class

  `com.andreamazzon.usefulmethodsmatricesandvectors.UsefulMethodsMatricesAndVectors`

in the exercise project (so you don't have to pull the Java project again).

Exercise 2

Write another class `MonteCarloPiFromHypersphere` implementing

`com.andreamazzon.handout4.montecarloevaluations.MonteCarloEvaluationsWithExactResultInterface`.

This class must provide approximations of π from the Monte-Carlo approximation of the volume of the unit hypersphere

$$\{(x_1, \dots, x_d) \in \mathbb{R}^d | x_1^2 + \cdots + x_d^2 \leq 1\}$$

for a given dimension d. The dimension must be a field of the class initialized in the constructor.
Hints: this is basically a generalization for higher dimensions of the Monte-Carlo implementation of π in the exercise above. The volume of the hypersphere is

$$V_d = \int_{-1}^{1} \cdots \int_{-1}^{1} \mathbf{1}_{\{x_1^2 + \cdots + x_d^2 \leq 1\}} dx_1 \dots dx_d = 2^d \int_{0}^{1} \cdots \int_{0}^{1} \mathbf{1}_{\{(2(x_1 - 0.5))^2 + \cdots + (2(x_d - 0.5))^2 \leq 1\}} dx_1 \dots dx_d. \quad (1)$$

It holds

$$V_{2k} = \frac{\pi^k}{k!}, \quad (2)$$

$$V_{2k+1} = \frac{2(4\pi)^k k!}{(2k+1)!}. \quad (3)$$

for $k \geq 1$ natural number.
Test this implementation taking inspiration from the classes for the tests above.