Concurrency — Exercise 2
Interaction through Cooperation

Prof. Dr. Oliver Haase

In this exercise your task is to implement the concurrency examples from the lecture in Java.

Problem 1

Implement example 1, i.e. the banking example with a shared `Account` class and concurrent execute tasks. Run a set of `deposit` and `withdraw` operations concurrently and produce race conditions.

Problem 2

Similarly, implement example 2, i.e. the factorizer service that writes but does not use a simple cache. Run multiple concurrent service operations and try to produce a multi-attribute state race condition. You may implement simple mock-ups for the types `Request` and `Response`.

Problem 3

Now extend your factorizer service from problem 2 such that it uses its cache, just like example 3 in the lecture. Produce a check-then-act race condition.

Problem 4

Implement the proof-of-work example (example 4) in the lecture in Java. Because it is very difficult to produce the race condition described in this example, this problem is not about producing the race condition. For the implementation, consider the following hints and advices:

- For the computation of SHA256, you can use the class `java.security.MessageDigest` and in particular the factory method `MessageDigest.getInstance("SHA-256")`.

- Class `java.security.MessageDigest` is not thread-safe. To avoid synchronization, equip each `checkHash` task with its own instance.
- Use a `BigInteger` to represent the `target` value that the hash must be smaller than.
- Use some arbitrary string to represent the block for which the proof-of-work is computed, such as `"new block"`.

Now experiment with different values for `target` such that the computation of the proof-of-work takes a time span in the order of seconds or a few minutes. Print the winner nonce that also indicates the number of `checkHash` computations as well as the execution time for the `proveWork` method.

Have fun and good luck!