Concurrency — Exercise 6
Rust II

Prof. Dr. Oliver Haase

Problem 1

Implement the banking example from the lecture in Rust. Follow these recommendations and directions:

- Model accounts with a struct with the methods `deposit` and `withdraw`.
- Outside the struct, implement an `execute` function as shown in the banking example.
- Write a `main` function that creates an account and that runs a series of threads each of which executes a command on this account.

Problem 2

Now implement the factorizer service from exercise 4 as a struct with functions, say `FactorizerService`. Follow these recommendations and directions:

- Move the variables `LAST_NUMBER` and `LAST_FACTORS` into the newly defined struct `FactorizerService`.
- Make the `service` method a method of the `FactorizerService`.
- Make the `factorizer` method an associated function of the `FactorizerService`.
- Get rid of the `unsafe` blocks.
- Run a similar test program as in exercise 4 to check whether a `service` call produces a cache hit or a cache miss.

In this version, the whole `service` method will be synchronized, because as an instance method of the `FactorizerService` it will be part of the shared data and hence guarded by its mutex.

Problem 3

Modify the solution from problem 2 such that the computation of the factors does not happen in the critical region, as seen in the lecture. Hint: To make this work, you need to convert the `service` method into an associated function. This will allow you to do the locking *inside* the `service` function.

Have fun and good luck!