Series 08 — 08.11.2017 – v1.0
Condition Objects

Exercise 1: General Questions (2 Points)

Answer the following questions:

a) Why are servers (e.g. web servers) usually structured as thread-per-message gateways?

b) What are condition objects? Name at least one advantage and one disadvantage of using condition objects.

c) Why does the *SimpleConditionObject* from the lecture not need any instance variables?

d) What are "permits" and "latches"? When it is natural to use them?

Exercise 2: Questions about Futures (3 points)

Consider the sample code *FutureTaskDemo.java* and *EarlyReplyDemo.java* which you both find in *Series-08-ConditionObjects-Code.zip*[1] in the package *asynchrony*. In both cases, several client threads request a server to compute fibonacci numbers.

a) Which implementation would you prefer for this kind of problem? Is there any considerable difference at all? Justify your answer!

b) Write a new class *FutureTaskExecDemo.java* that uses an executor service to compute the future task and to execute the clients, instead of creating explicit new threads. What is the benefit of using executors?

c) Add a time constraint such that the client thread waits for at most a given amount of time for the result.

Exercise 3: Nested Monitor (2 points)

Farmer Napoleon owns a magic chicken called Clarissa who is supposed to lay infinitely many eggs. Napoleon has hoped to dispose an endless source of eggs to build up his egg-imperium. But there is a serious deadlock problem hidden behind the story. Your task is to resolve this problem in order to let Napoleon continously retrieve eggs from Clarissa. Be careful that your solution is data race free.

You can find the code in the file *CP-Series8.zip* of the previous task in the package *eggFarm*.

Exercise 4: Thread Speed Evaluation (3 points)

We provide you with an implementation of a Pi approximation tool. This tool uses the Leibniz formula for Pi with infinite series for approximation. Your task is to observe the characteristics of the threads as well as their calculation speed. You can simply modify the values of the variables *concurrentThreads* and *rounds* in the class *ThreadedPiCalculator* to change the amount of allowed concurrent threads,

[1] http://scg.unibe.ch/download/lectures/cp-exercises-2017/Series-08-ConditionObjects-Code.zip

respectively the accuracy of the result, i.e., number of iterations performed. It is very important that you perform multiple runs before answering the questions below, because the outputs each run can vary up to a degree. You can find the sample code on GitHub at `https://github.com/pgadient/concurrency_e08t04.git`. Please answer the following questions:

a) What amount of processing cores does the CPU in your notebook have and what's the model / manufacturer of it?

b) Does the implementation scale well, i.e., more concurrent threads help greatly to reduce the overall calculation time? Please provide concrete runtimes you experienced!

c) Depending on your results, why or why not does the solution scale well?

d) How would you improve the runtime with respect to faster calculations (without changing the algorithm)?

e) Which algorithm would you recommend as drop-in replacement for the Leibniz formula for faster calculation?

f) Why do the runtimes with identical parameters vary so much?