(English below)

Második gyakorlat

Tematika

- Szálak létrehozása Java-ban
- Műveletek a szálakkal – join, sleep, interrupt
- Szálak életciklusa
- Aszinkron metódusok

Példák

A concurent.labs.example.ExampleHub fájlban a fenti tematika bemutatására szánt kódok vannak összegyűjtve, kommentekkel ellátva.

Feladat

Az órai feladat a concurent.labs.task package alatt található fájlok kiegészítése. A TODO kommentekkel jelölt helyekre a hallgató feladata, hogy működőképes implementációt készítsen.

A feladat egy szimuláció létrehozása, ahol két féle dolgozó (ezeket szálak testesítik meg) próbál felépíteni 5 házat – a bányászok (miner) az aranybányából bányásznak aranyat, amiből az építészek (builder) tudnak házakat építeni.

- **Configuration**: Ez az osztály a szimuláció konfigurációját tartalmazza, mint hogy meddig tartson egy építkezés, hány dolgozónk legyen, stb. A feladat során ehhez **nem** kell hozzányúlni. Miután kész vagyunk a feladattal, egy jó teszt lehet ezeket kedvünk szerint állítgatni és meggyőződni, hogy nem töri el a megoldásunkat, ha más paraméterekkel kell dolgoznia.
- **Resources**: Ez az osztály felel azért, hogy nyomon kövesse, mennyi aranyat lehet még bányászni a bányából, mennyi arannyal rendelkeznek a dolgozók, illetve hány házat építettek már fel. A feladat ebben az osztályban meggyőződni, hogy a különböző szálak probléma nélkül, szálbiztosan férnek hozzá az adatokhoz, tudják azokat módosítani
- **ThreadCraft**: Itt található a main metódus, illetve a dolgozók implementációja. A feladat a következőkből áll:
    - Elindítani a dolgozók (bányászok és építészek) szálait (mineAction() és buildAction())

- Elindítani egy logoló szálat, ami bizonyos időközönként kiírja a szimuláció haladtát (loggingAction())
- A main-en belül megvárni, amíg a dolgozók szálai befejezik a munkát, majd kiírni, hogy vége a szimulációnak
- Implementálni a sleepForMsec(int msec) metódust, ami azért felel, hogy az azt meghívó szálat “sleepeltesse” a megszabott ideig

Lab 2

Syllabus

- Thread creation in Java
- Operating with threads – join, sleep, interrupt
- Lifecycle of threads
- Asynchronous methods

Examples

Examples for the aforementioned topics can be found in the concurent.labs.example.ExampleHub class.

Task

The task to do during lab 2 can be found in the concurent.labs.task package. There are multiple segments in the code that has only a "TODO" comment with instructions on what should be done. The aim is to provide an implementation based on those comments and verify that the simulation is running correctly.

The goal of this simulation is to have two types of workers (represented by threads) who are trying to build 5 houses – the miners are mining gold from the goldmine, while the builders, once they have enough gold to do so, build houses.

- **Configuration**: This class contains the configurations for the simulation, such as duration of building a house, number of workers, etc. During completion this class should **not** be modified. After achieving a working simulation, the quality of the solution could be tested by changing some of the parameters, ensuring that the code does not only work for the predefined config.
- **Resources**: This class keeps track of all the resources used by the simulation – goldmine capacity, gold owned by the workers and number of houses built. The task is to ensure thread safety on the methods of this class since multiple threads will try to modify or read the data.
- **ThreadCraft**: This is the entry point of the simulation, containing the main method and behaviour of the workers. These implementations need to be provided:
    - Starting the workers' threads (miners and builders) with the provided action (mineAction() and buildAction())

- Start a thread responsible for periodically logging the state of the simulation (`loggingAction()`)
- In the main function before printing “Simulation over” the workers’ threads need to finish
- Implement the `sleepForMsec(int msec)` method which is used to invoke sleep on a thread for a given time