(English below)

Harmadik gyakorlat

Tematika

- Szálak létrehozása és elinditása
- Szinkronizált blokkok
- Synchronized Collections és iterálás rajtuk

Példák

A concurent.labs.example. package-ben a szinkronizált gyűjtemények 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 az előző órái szimuláció átirása úgy, hogy a megosztott erőforrásokat granulárisan lockoljuk a teljes osztály lockolása helyett (szinkronizáláshoz a szinkronizációs blokkokat fogjuk használni).

- **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.
- **ThreadCraft**: Itt található a main metódus, illetve a dolgozók implementációja.

A feladat a következőkből áll:

Resources osztály:

- Megoldani, hogy a különböző szálak probléma nélkül, szálbiztosan férnek hozzá az adatokhoz, tudják azokat módosítani

ThreadCraft osztály:

- 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 3

Syllabus

- Creating and starting Java Threads
- Synchronized blocks
- Synchronized Collections and iteration over them

Examples

Examples on synchronized collections and their usage (also iteration) have been collected in concurent.labs.example package.

Task

During lab 3 tasks under concurent.lab.task package have to be solved.. 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.
Task is to enhance the solution from lab 2 in a way that resources will be locked granularly instead of locking the whole class (we will use synchronized blocks to achieve this).

- **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.
- **ThreadCraft**: This is the entry point of the simulation, containing the
- main method and behaviour of the workers.

Task contains the follwoing:

Resources class:

- Ensure thread safety on the methods of this class since multiple threads will try to modify or read the data

ThreadCraft class:

- 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