Java Threads →

Deadlocks, Livelocks & Starvation

LABS

Lab sources: <gitlab address>

ada.diaconescu@telecom-paris.fr

Deadlock

- Use java.util.concurrent.locks.ReentrantLock to implement a shared Resource with mutual exclusion
- Define a class called DeadlockExample
  - Declare two static variables: lock1 and lock2 of type ReentrantLock
  - Use lock() and unlock() methods to acquire and release the shared Resource, respectively
  - Define operation1(): acquire lock1, sleep(50), acquire lock2, <do something>, unlock lock2 & lock1
  - Define operation2(): acquire lock2, sleep(50), acquire lock1, <do something>, unlock lock1 & lock2

- Create two threads T1 and T2
  - T1 calls operation1() in its run() method
  - T2 calls operation2() in its run() method

- The main(...) method creates an instance of T1 and an instance of T2, then starts them both

- Questions:
  - What happens? Why?
  - What happens if the two variables lock1 and lock2 are no longer class variables (static)? Why?

Livelock

- Use java.util.concurrent.locks.ReentrantLock to implement a shared Resource with mutual exclusion
- Define a class called LivelockExample
  - Declare two static variables: lock1 and lock2 of type ReentrantLock
  - Use trylock() and trylock(long timeout, TimeUnit unit) methods to try to acquire the shared Resource

    → acquires the lock unless it is already held by another thread; and returns ‘true’

    → returns ‘false’ otherwise (if the lock is already held by another thread)
  - Define operation1(): in a while loop:

    →try to acquire lock1, with a timeout of 50 ms; sleep(50); try to acquire lock2

    → if lock2 acquired: <do something> and break out of the while loop; otherwise: unlock lock1 and go to the next while iteration (using continue)
  - Define operation2(): in a while loop:

    →try to acquire lock1, with a timeout of 50 ms; sleep(50); try to acquire lock2

    → if lock2 acquired: <do something> and break out of the while loop; otherwise: unlock lock1 and go to the next while iteration (using continue)
- Create two threads T1 and T2 as before and a main method that creates an instance of each and starts them
- Questions: what happens? Why? How does the timeout value impact the result?