Programming Assignment 1 – Édouard Gagné 400161204

**Task 1: Atomicity Bug Hunt**

Each of the account objects have two threads, a deposit thread and a withdrawal thread, which attempt to modify the balance of the account. During the processing of one of these threads, it is possible that a context switch happens kicking out the thread from the running state to ready state to give the thread thread a chance to run. This leaves the deposit or withdrawal processes to be incompletes and not atomic, since they are stopped in the middle, and the final account balance to be inconsistent.

**Task 2: Starting order**

A thread is created when the program starts and is put into a runnable state. While in that state, it can be put into a waiting, time waiting or blocked state, and eventually be terminated when its task is completed.

Since threads are executed concurrently, their order of execution is not guaranteed and can change every time the application is launched, depending on how the operating system has scheduled the threads. For example, in the AccountManager class, if we start the withdrawal threads before the deposit threads, the results are completely different, so the consistency of the accounts is not preserved.

**Task 5: Synchronized block vs synchronized method**

Synchronized blocks have an advantage over synchronized methods, since they reduce the scope of the lock by only locking critical sections to increase performances instead of locking the whole method. Synchronized block also allows more flexibility since they can synchronize on objects other than “this”. However, since the scope of lock is not that much smaller using the synchronized block solution in the “Withdraw” and “Deposit” methods and that there are no other objects to synchronize with other than Accounts, the advantage of using synchronized block over synchronized method is marginal, which can be observed by the fact that task 3 and 4 executed in around the same time.