Task 1: Atomicity Bug Hunt

The main requirement above is not met because the it doesn’t deal with the problem of synchronization. Both of depositor and withdrawer can run at the same time which may result in a mess. The atomicity problem it poses is that writing thread id and updating index. In the code of Withdrawer and Depositor, account.withdraw and account..deposit are not be protected so they may withdraw a lot of times without depositing, which may induce a mess. So I use the synchronized(account) to enclose the operation code. If we want two operations can write at the same time, we should use mutex to let withdrawer know whether the account still have enough money to be get. If putting X = 10 CAD and Y = 20CAD, the results are all negative because we let the depositor runs and it can’t be interrupted and withdrawer runs to withdrawal the money but withdrawal more than deposit a lot. If the accounts don’t have money, withdrawer shouldn’t withdraw money. In operating system, it likes a shared resources wanted to be read and wrote at the same time. It also likes the problem of ‘producer and consumer problem’.

Task 2: Starting Order is determined by JVM and Operating System. The lifetime: After thread created and getting the resources of CPU, it will be executed and at the end it will be terminated. But if the thread is interrupted by other threads or waiting some resources or lock, the execution will stop. For here, the consistency of the accounts isn’t preserved.

Task5: Synchronized block is more flexible because it can use any objects to lock and lock smaller critical section. The block we want to but synchronized method lock the entire method. The 4A# use less executing time than 3A#.