Programming Assignment 1

**Task 1: Atomicity Bug Hunt**

Compile and run the Java app given to you as it is. Explain why the main requirement above (i.e. consistent state of the account array) is not met. What atomicity problem does it pose? Find the bug that causes it. In no more than three sentences, explain what went wrong.

In this case, the atomicity should be deposit() + withdraw(). When executing deposit() or withdraw(), it is not synchronized. If the threads are not synchronized, the tasks in either deposit() or withdraw() will be interrupted, which cause the shared data been miscalculated.

To fix the problem, we simply synchronize the threads, which locked the critical section in each thread, so the shared data ”account ” will not been modified by several threads concurrently.

**Task 2: Starting Order**

Explain, in about one sentence, what determines the start order of the threads. Also, very briefly, explain the lifetime of a thread: its creation, execution, and termination. Experiment with the start order of any of the threads. Is the consistency of the accounts preserved?

It’s the CPU scheduler determines the starting order of the threads.

Thread Creation: create a thread instance, this thread is in the “new” state and has its own memory space. However it has not been processed yet, it’s not alive.

Thread execution:

Runnable: the thread is executing in the java virtual machine, but it may be waiting for other resources from operating system such as processor.

Blocked: the thread is waiting for a different thread to release its lock in order to get the monitor lock.

Waiting: the thread is waiting by using a wait, join method.

Timed\_waiting: the thread is waiting by using a sleep, wait, join method. (The difference from waiting is that the maximum waiting time is specified by the method parameter, and waiting can be relieved by time as well as external changes.)

Thread termination: since the thread completed, it’s terminated.

The consistency of the accounts cannot be preserved. The thread can be interrupted during its execution.

**Task 3: Method level synchronization**

Create a package and name it task3 and copy the provided java files into that new package. Use synchronized technology (method level synchronization) in order to introduce a solution to the problem at hand.

Elapsed time in milliseconds 231803

Elapsed time in seconds is 231.803

**Task 4: Block level synchronization**

Create a package and name it task4 then copy the java files from task 3 into that new package. Use synchronized technology (block level synchronization) in order to improve the solution to the problem at hand.

Elapsed time in milliseconds 155660

Elapsed time in seconds is 155.66

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

Considering the results of task 3 and task 4, what is the advantage of synchronized block over synchronized method?

Comparing the elapsed time in task 3 and 4, we can see that task 3 takes more time to execute than task4. The synchronized block is more efficient than synchronized method.

Synchronized method includes more statements in the synchronized section than synchronized block. More statements in synchronized section extend the executing time for a thread in its synchronized section, it also means more waiting time for another thread which in the ready queue to enter critical section.  However the true critical section actually contains only one statement, which means synchronized method wastes a lot of time blocking other threads to enter uncritical section, this is the reason why synchronized method takes more time in execution.