**TASK 1, 2 AND 5**

**Output of the given file:**

Print initial account balances

Account: 1234 Name: Mike Balance: 1000.0

Account: 2345 Name: Adam Balance: 2000.0

Account: 3456 Name: Linda Balance: 3000.0

Account: 4567 Name: John Balance: 4000.0

Account: 5678 Name: Rami Balance: 5000.0

Account: 6789 Name: Lee Balance: 6000.0

Account: 7890 Name: Tom Balance: 7000.0

Account: 8901 Name: Lisa Balance: 8000.0

Account: 9012 Name: Sam Balance: 9000.0

Account: 4321 Name: Ted Balance: 10000.0

Depositor and Withdrawal threads have been created

Print final account balances after all the child thread terminated...

Account: 1234 Name: Mike Balance: 2.892219E7

Account: 2345 Name: Adam Balance: -6.388762E7

Account: 3456 Name: Linda Balance: -5.751443E7

Account: 4567 Name: John Balance: -6.425807E7

Account: 5678 Name: Rami Balance: -6.353291E7

Account: 6789 Name: Lee Balance: -7.574007E7

Account: 7890 Name: Tom Balance: -6.010931E7

Account: 8901 Name: Lisa Balance: -8.016319E7

Account: 9012 Name: Sam Balance: -5.318203E7

Account: 4321 Name: Ted Balance: -9.434647E7

Elapsed time in milliseconds 104

Elapsed time in seconds is 0.104

**TASK 1**

All the threads are accessing the balance at the same time without any order causing an atomicity problem when the threads access or modify the balance value of the method, causing it to end up with an unexpected value. The bug is the lack of atomicity in the code from not having a synchronization method.

**TASK 2**

There is no way of determining the starting order of threads. To ensure specific order, explicit synchronization must be used.

**Life cycle:**

* **New:** A new thread is created
* **Runnable:** After calling the start() function on a thread, its state changes to runnable. The control is given to the thread scheduler which decides what thread is to run.
* **Running:** when a thread is executing its state changes to running. A thread can change state back to runnable, waiting (for a resource) or terminated.
* **Waiting:** A thread’s state changes to waiting when waiting for another thread to finish executing (join()) or for a resource to be available. Once the thread’s wait state is over it changes back to a runnable state.
* **Terminated:** Once a thread has finished executing its state is terminated.

**Is the consistency preserved?**

No, without having the synchronised method all threads are accessing the method simultaneously affecting its consistency.

**TASK 5**

The real advantage of synchronized block over synchronized method is that you can chose which object or part of the method is to be synchronized instead of synchronizing the whole method of the object or class.