Noemi Lemonnier
#40001085

COMP 346 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.

The main requirement above explains that the balance in accounts must be the same before and after the operations are done. This requirement is not valid as when the code is run, the results in each account varies and changes. It is an atomicity problem of consistency, as the data does not respect the main constraint. It looks like all threads are started at the same time without being managed.

The cause of this risk condition problem is synchronization as none of the threads are manage correctly. As they are updating a shared variable without being able to view each other actions, they are not providing the hoped result. Threads are not communicating with each others.

Task 2: Starting Order

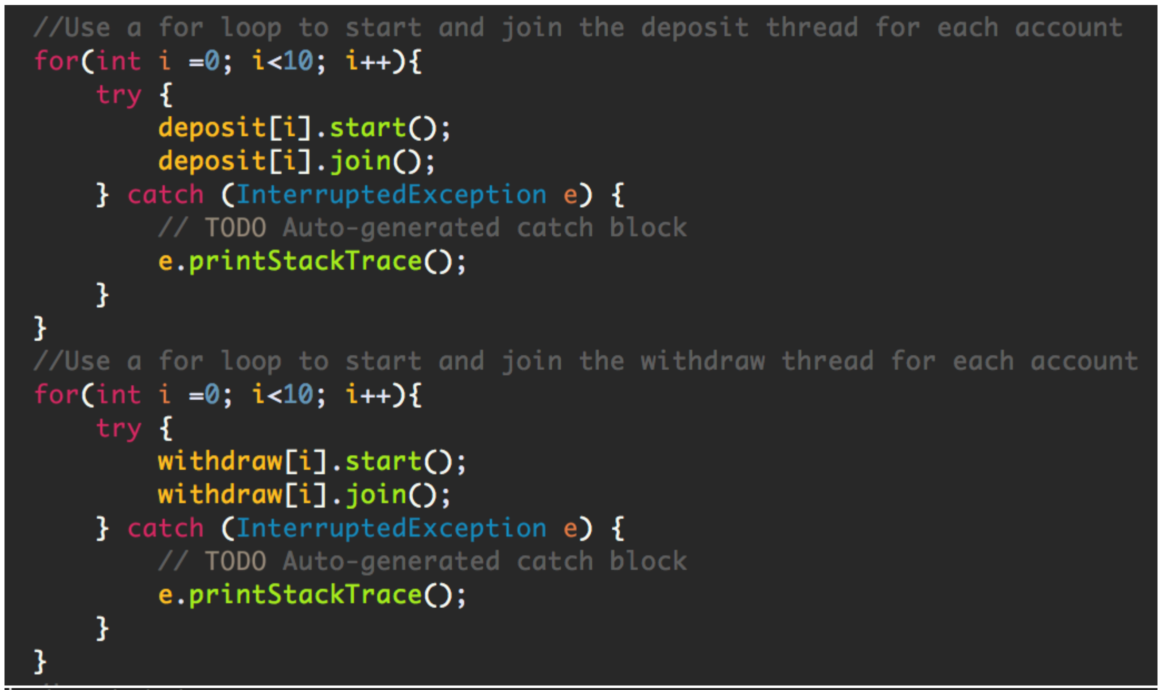
Explain, in about one sentence, what determines the start order of the threads. Also, very briefly, explain the lifetime of a thread: its creation, execution, and termination. Experiment with the start order of any of the threads. Is the consistency of the accounts preserved?

The start order of the threads is determine by how the programmer declared the threads' start and join methods. In this case, the programmer used a for loop to go through each account and start their deposit and withdraw threads and then used another for loop to join each deposit and withdraw threads.

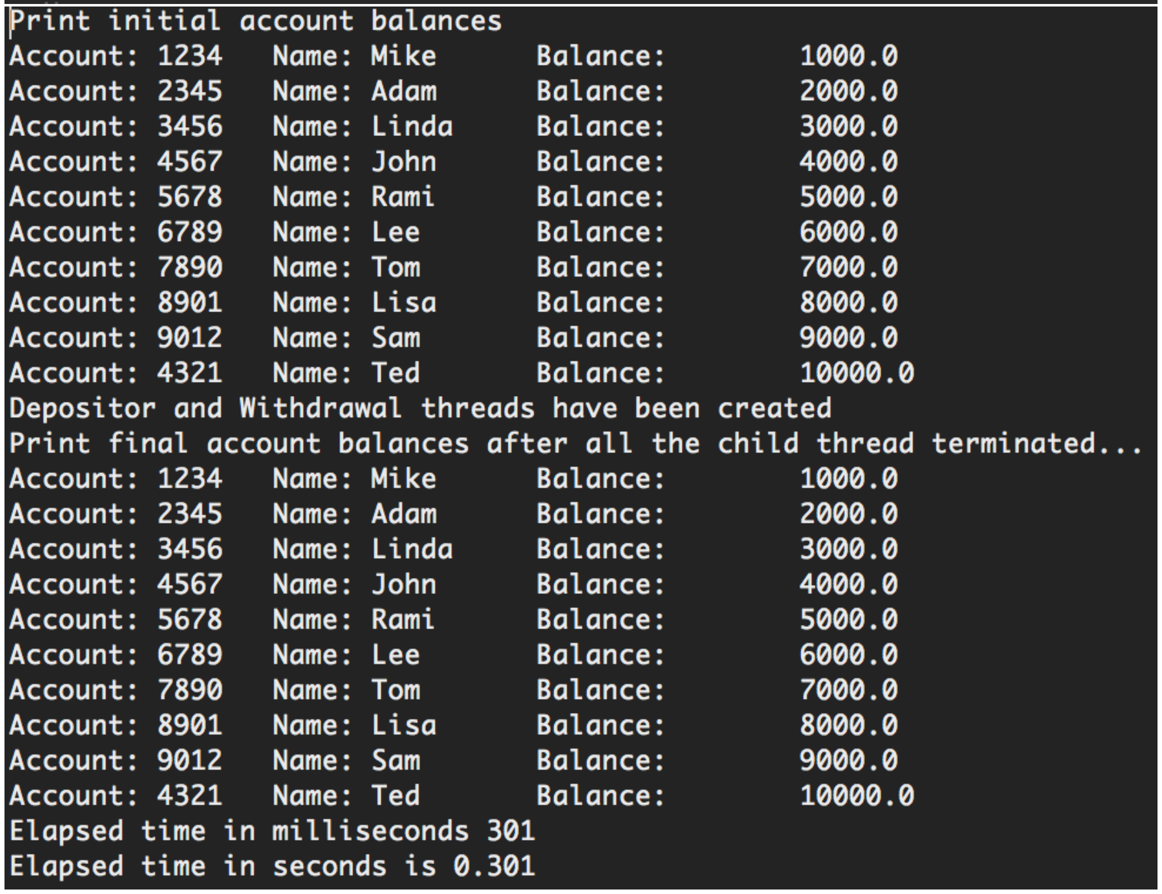
First, the threads are created and declared. Then, the threads are started one after the other by using a for loop that goes by accounts. For example, it will declare account[0] and start deposit[0] and withdraw[0] and it will do the same for the other accounts. Another for loop is then used to join threads of deposit[0] and of withdraw[0].

Synchronizing the threads for each can preserve the consistency of the accounts. Instead of starting the deposit and withdraw threads at the same time, it is needed to start the deposit thread then join it for all accounts and then start the withdraw thread and then join it for all accounts.

Noemi Lemonnier
#40001085



```
//Use a for loop to start and join the deposit thread for each account
for(int i =0; i<10; i++){
    try {
        deposit[i].start();
        deposit[i].join();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
//Use a for loop to start and join the withdraw thread for each account
for(int i =0; i<10; i++){
    try {
        withdraw[i].start();
        withdraw[i].join();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```



```
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:        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
Elapsed time in milliseconds 301
Elapsed time in seconds is 0.301
```

Noemi Lemonnier
#40001085

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?

The results for task3 with the Method level synchronization took 1.5 seconds and for task4 with the Block level synchronization it took 1.37 seconds. The main advantage of synchronized block over synchronized method is that it recudes the scope of lock. It means that it will allow the program to specifically synchronized a particular section of the program and it will increase the performance of the program.