Sathurthikan Saththyvel 40213455
Team Partner: Nicholas Werugia 40131956

Task 1: 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. Find the bug(s) that cause(s) it and in a few sentences explain how it can be fixed.

Since, there is no restriction placed on the threads there is nothing preventing them from accessing a shared resource at the same time which leads to bugs. In other words, this makes context switches occur at any point in time without them being prevented from occurring.The easiest way to fix this thread error is to synchronize the methods that modify the account values. In this case, we can simply fix the code by having both ‘deposit’ and ‘withdraw’ methods in the Account.java class have the ‘synchronize’ modifier.

Task 2: Explain in a few sentences what determines the starting order of the threads. Can the consistency of the accounts be preserved by changing the starting order? Explain.

In this, the starting order of the threads depends on how long it took for its previous action to end. So, if for example, a thread completes twice before another one is done, it will be able to do “deposit” or “withdraw” twice in a row. This is why errors in balance happen.The consistency of the accounts will not be preserved by simply changing the starting order as synchronization is crucial in allowing the operating system to know how to determine the access of these threads to the shared resource.

Task 3: Identify the critical sections of the given code. You can “cut and paste” the relevant pieces of code to your answer.

The critical sections of the given code as can be seen below are the "deposit" and "withdraw" methods within the 'Account.java' file as these use threads that if not synchronized could potentially access a shared resource at time if instantiated one after the other. So, this makes them the critical section of the code given.

```
/**
* A method that allows a customer to deposit money into this account
* @param amount A double that represents a deposit amount
*/
public void debosit(double amount){
// Waste some time doing fake computations
// do not remove or modify any of the following 3 statements
double k = 999999999;
for(int i=0;i<100;i++)
k = k / 2;

balance = balance + amount;
```

Sathurthikan Saththyvel 40213455
Team Partner: Nicholas Werugia 40131956

```
// Waste some time doing fake computations
// do not remove or modify any of the following 3 statements
k = 999999999;
for(int i=0;i<100;i++)
k = k / 2;

}

/**
* A method that allows a customer to withdraw money from this account
* @param amount A double that represents a withdrawal amount
*/
public void withdraw(double amount){

// Waste some time doing fake computations
// do not remove or modify any of the following 3 statements
double k = 999999999;
for(int i=0;i<100;i++)
k = k / 2;

balance = balance - amount;

// Waste some time doing fake computations
// do not remove or modify any of the following 3 statements
k = 999999999;
for(int i=0;i<100;i++)
k = k / 2;
}
```

Task 6: Considering the results of Task 4 and Task 5, what is(are) the advantage(s) of synchronized block over synchronized method, or vice versa? Explain.

Block synchronization is faster than method synchronization as can be observed by the time elapsed being much faster for Task 5. This is because block synchronization can be much more specific in what it will synchronize within the given code over having to synchronize parts of the code that don't need to be involved in the synchronization of these tasks and won't affect access to the shared resource. Block synchronization is more efficient.