Christophe Savard
40017812

September 27th 2019
COMP-346 DD

Programming Assignment #1

Task 1.

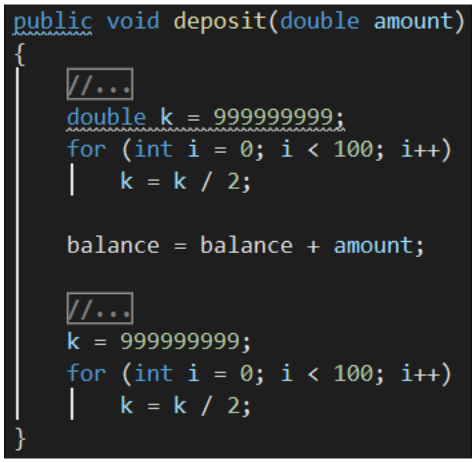
The starting and ending balance should be identical for everyone as the amount deposited is the same as the amount added for both threads. What is actually happening is that the threads are not given exclusive access to the balance of the account. This eventually results in a thread reading half written data from the other thread and interpreting it as the actual value, resulting in invalid numbers being taken as correct.

Task 2.

For each account, the deposit thread is first started, then the withdrawer thread is immediately started, then the process moves on to the next account. Swapping the order or starting all deposit then all withdraw threads does not affect the result either, as the atomicity problem mentioned above is still not fixed. Starting all of one time of thread, then waiting for them to all terminate could solve this problem but would get rid of the concurrency on the account level. Lifetime wise, the thread is first created by the main thread, then deposits or withdraws from its assigned account ten million times, and once this is complete, shuts itself off as there is no more code to execute.

Task 3.

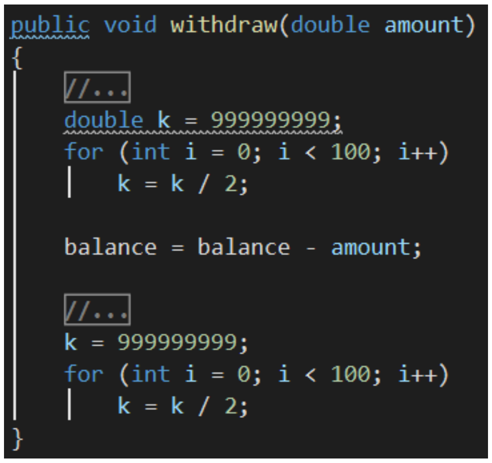
The following code is the sensitive code, specifically the "`balance = balance + amount`", Writing and reading the value stored in balance is not atomic and can lead to incorrect amounts being read.



```
public void deposit(double amount)
{
    //...
    double k = 999999999;
    for (int i = 0; i < 100; i++)
        k = k / 2;

    balance = balance + amount;

    //...
    k = 999999999;
    for (int i = 0; i < 100; i++)
        k = k / 2;
}
```



```
public void withdraw(double amount)
{
    //...
    double k = 999999999;
    for (int i = 0; i < 100; i++)
        k = k / 2;

    balance = balance - amount;

    //...
    k = 999999999;
    for (int i = 0; i < 100; i++)
        k = k / 2;
}
```

Task 6.

Instead of locking the entire method's execution, the synchronized block only locks at the critical locations where read/write operations on the object are performed, allowing for the rest of the code to proceed no matter if another thread is also doing the same. This allows for faster execution in theory. In practice, however, concurrency does not mean true parallelization. The OS may be running all the Java VM threads on a single core. This all comes down to how the OS decides to allocate the program to CPU time. This is why our times are so similar.