# Testing Report

## Bank Sim Multi-thread program

CIS 3296 Section 5
Spring 2022

Team Members:

- **Arthur Kozhevnik**
- **Robert Stachurski**

Repository URL:

- **https://github.com/cis3296s22/banksim-05-kozhevnik-stachurski**

Trello:

- **https://trello.com/b/05dLGp96/bank-sim**

## Table of Contents

## Document Overview

In this lab, we were tasked with understanding the problems that could occur with multi-threading and finding solutions on how to fix these problems. From issues with race conditions to deadlocks, BankSim was riddled with issues common to multi-threading. In order for this BankSim to work properly, we needed to protect our critical sections using locks and semaphores. With these methods, we fixed issues that were weighing down the original code and we have described in detail how these fixes were implemented in the Report and Architecture sections as well in the Design API

## Report

Task 1: We were tasked with observing the race condition by forcing it using Thread. yield(), explaining why it occurs as well as drawing a UML sequence diagram to back up our explanation. We wrote that the race condition happens inside the transfer method of the Bank class. The race condition happens when two or more threads access the same account at the same time. If thread1 adds money to a balance and thread1 withdraws money from the account, when thread1 tries to save the new amount to the balance variable, it could be interrupted and thread2 saves its new balance to the variable. When thread1 wakes up, it will overwrite thread2's balance and it will be an incorrect amount even though the transactions happened and our first Sequence Diagram shows this as well.

Task 2: We were tasked with resolving the race condition and allowing the bank to transfer funds between multiple unrelated accounts by using locks. For this, we implemented synchronization for our *transfer* method in the Bank class and for the *withdraw* and *deposit* methods in the Account class.

Task 3: We were tasked with refactoring the method of testing into a separate thread. A tester thread is created and runs while the bank is still open.

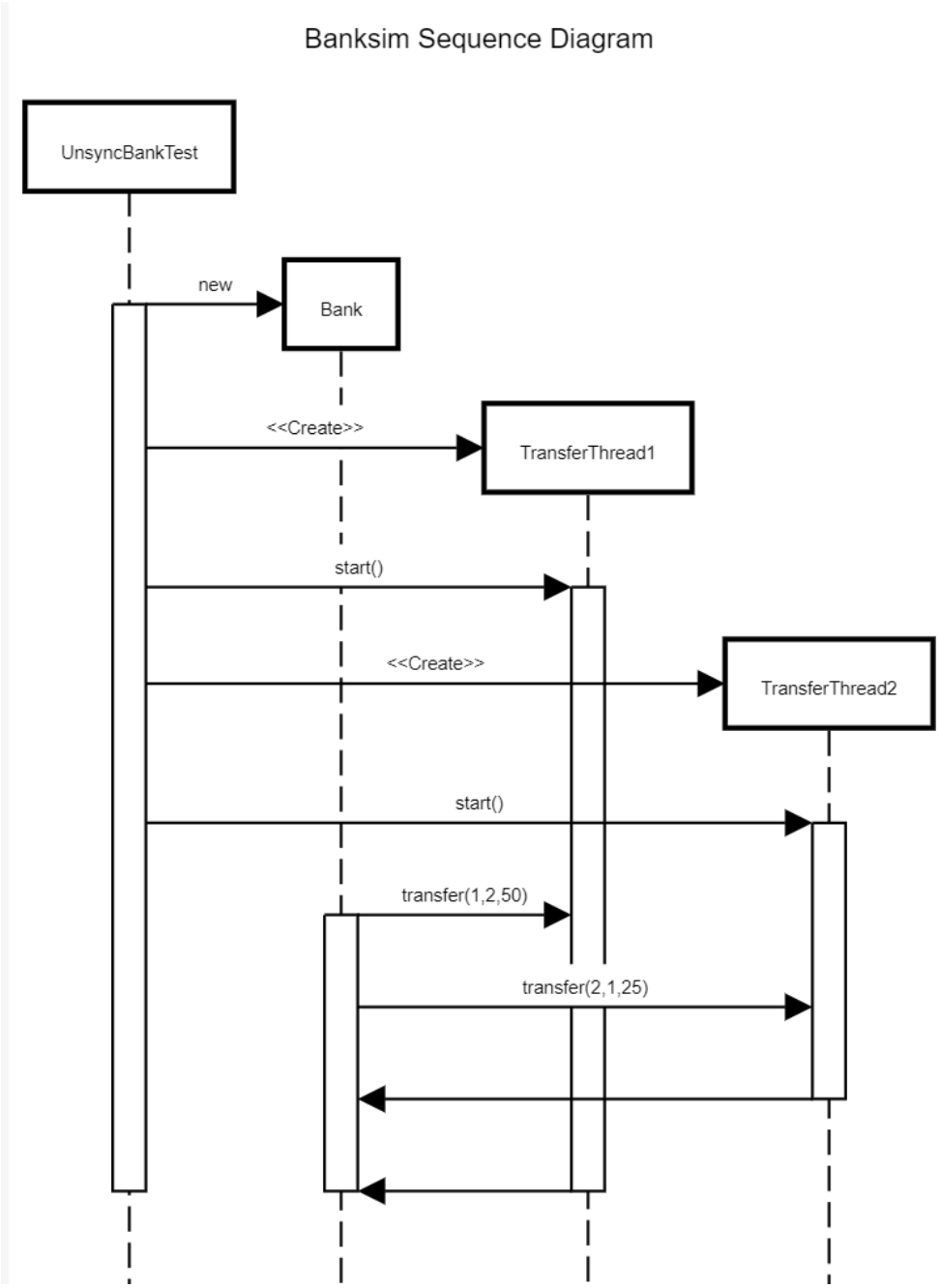
Task 4 and 5: We were tasked with providing a protection code for our transfer thread. In order to do this, we created a semaphore that would be acquired right before we called the different transfer methods of the Account class and released after all of the operations were complete. Testing thread wait to acquire all the permits available and make the transfer threads wait.

Task 6: We were tasked with implementing a wait/signal solution for the testing and transfer threads. To do this we implemented a *wait()* signal while we check for the funds in an account in the Account class. While the bank is open and the amount is greater than or equal to a balance a thread accessing the *checkFunds* method will wait until that condition is interrupted. We also added a *notifyAll()* signal at the end of the *deposit* method to notify any threads waiting to deposit and use checkFunds.

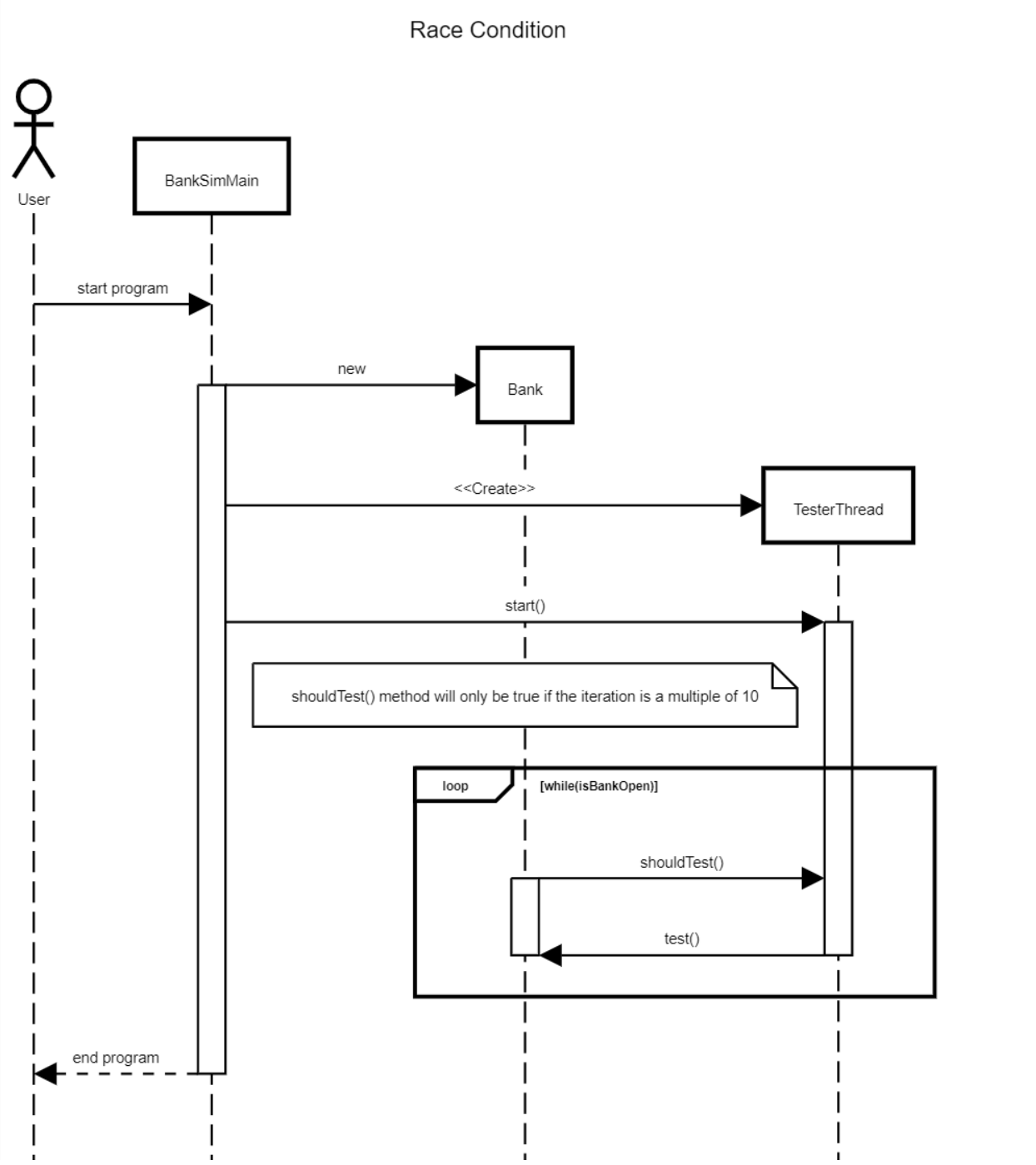
Task 7: We were tasked with stopping a deadlock once a thread finishes, to do this we implemented a close bank method. After the TransferThread is done with all of its processes a method will change our bankOpen boolean to false. This closes the bank because when a transfer happens, the bankOpen boolean is required to be true, or else the transfer would not go through. When the close bank method is called, it also notifies all the accounts so they could run and exit.

Problems Encountered: There were multiple problems involving the usage of semaphores and being able to make all transfer threads finish their transfer and wait for the testing thread to finish. We wanted to implement semaphores to keep mutual exclusion between threads. One of the problems with using semaphores was how we were going to make the transfer threads finish their transfer and how the testing thread was going to run while the transfer threads waited. We were not aware that a thread can acquire multiple permits but once we did know that as possible, the testing thread was made to acquire all permits. Also, another problem was when we implemented the funds check before the transfers, only around 30 transfers would take place and run in a loop. We solved this problem by removing the true parameter when creating the semaphore.

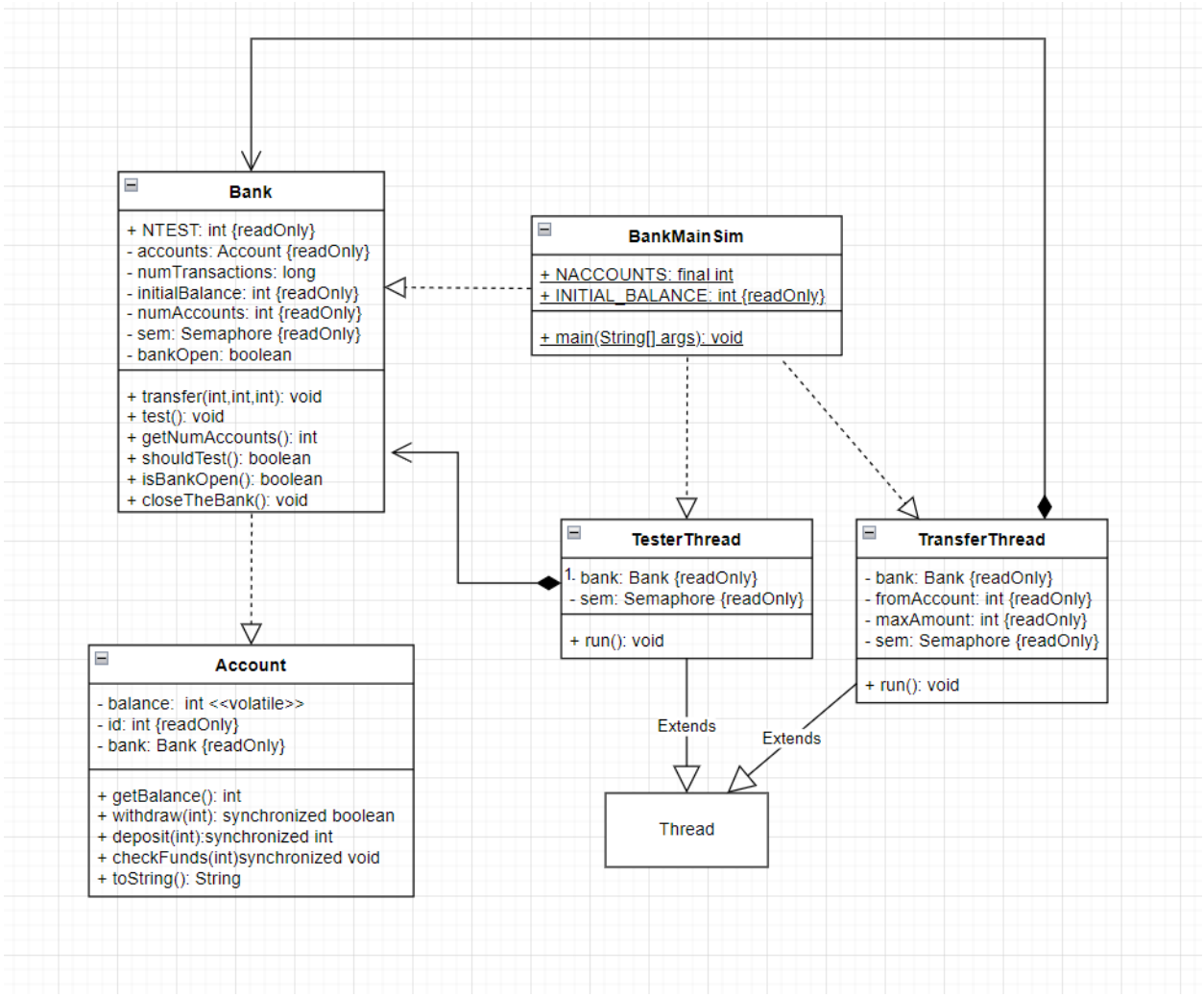
**Architecture:** (UML Class diagram and 2 UML Sequence diagrams)




Here we are depicting the race condition that was observed in Task 1. Both TransferThreads are started and both can begin a transfer without letting the other finish. This will cause each thread to have knowledge of only the current state of the bank accounts without knowing the state in which the bank accounts are in, in other threads. Two separate transfers having the same time will cause the incorrect information to be sent to the accounts. An account can have multiple withdraws of all of its funds which causes a race condition




Here we are depicting how the Tester Thread is invoked after the completion of task three. The BankSimMain will create the TesterThread, and it will start it right after. After doing so the TesterThread will enter a while loop constantly checking if the bank is open. While the case is upheld the TesterThread will receive an answer from the Bank if a test should be conducted ( if the iteration is a multiple of 10) and after confirming that it should in fact test the test() method will be run from the TesterThread class.




Here we are depicting the class diagram of the BankSim. We are showing different relationships between each of the classes. The BankMain is the beginning of the code so it has no incoming associations while having multiple outgoing associations. Both Tester and Transfer Thread extend the Thread class. They are also both created by BankSimMain and invoked by the Bank. The accounts are created in the Bank class.

## Detail Design API:

**Link:**

https://github.com/cis3296s22/banksim-05-kozhevnik-stachurski/tree/main/Documentation/Javadoc

## Coverage Report

**Link:**

https://github.com/cis3296s22/banksim-05-kozhevnik-stachurski/tree/main/Documentation/CoverageReport