# Final Report Programming Assignment 2

## Design Report:

### Multithreading strategy:

The multithreading approach implemented in the webServer employs a mix of thread pools, semaphores, and a blocking queue to effectively manage numerous client requests.

- The use of a fixed-size thread pool of 150 threads is responsible for handling the individual client requests. It helps manage and reuse threads efficiently.
- A semaphore was used to control the number of concurrent requests that can be processed. It was initialized with a maximum of 200 users. Before handling a client request, the server acquires a permit from the semaphore using “requestSemaphore.acquire()” function and after the request is handled, the permit is released using “requestSemaphore.release()”

- A blockingQueue is used to queue incoming client sockets for processing. It is initialized with a capacity of 10,000 elements
- A lock for synchronizing transfers was used to synchronize access to the “processTransfer” method. This ensures that only one thread at a time can execute the critical section of code within the method. The lock is acquired in the beginning of the method and is unlocked at the end.
- The implementation of a “handleRequestsFromQueue” method which runs in a separate thread and continuously checks the Blocking Queue for incoming requests.

## Synchronization strategy:

In addition to some synchronization strategies stated above since they are also part of the multithreading strategy that was implemented to have a thread safe running webServer, In the Account.java class:

- “AtomicInteger” was used to ensure atomicity for balance operations within the class to prevent race conditions from happening in the Account.java class.

The reason behind using the “AtomicInteger” is to prevent race conditions and ensure the balance related operations are done atomically providing thread safety. The reason why would the race condition happen is if several threads are trying to access the critical section which is the read and update the balance operations of two accounts in that example.

## Testing strategy:

Two strategies were used to test the multithreaded webServer that was Implemented:

1) - The first test was running a simple web client that would send requests in a loop within the main method to test the webServer.

   - It includes a fixed sleep time between requests being done.

2) - The second test was running another simple web client that would create and start 1000 threads simultaneously, each representing a client.

   - It includes a random sleep time between requests, generated randomly which stimulates a more realistic scenario where clients might not send requests at a constant rate.

In conclusion, both web clients are effective ways to test the implemented webServer. The second web client would still be more realistic since it simulates the presence of multiple clients that are interacting with the webServer.