**Group F4 Test Strategy Document**

In both the first and second part of the project our idea was to test using JUnit tests, the reasoning being that they give us the most clear indication of what goes wrong during actual, functional testing (rather than just displaying that the program works). While this was an effective solution for the first part of the project, it proved less so during the second part, due to the GUI-based input (particularly input dialogs), so instead we have elected to provide screenshots of some manual testing for the GUI, along with descriptions.

**Part A**

Part A covers 5 of 6 scenarios of use provided but more goes into testing the application that just that. For example, one of the subtypes of account we have provided is a kids’ account. This type of account can be deposited into, but cannot be withdrawn or transferred from. Thereby we need a test that shows that such a subtype works to those specifications. In order to do so, we:

* Created a kids’ account as well as a current account, and deposited 100 into each.
* Attempted to transfer 25 from the kids’ account to the current account, and attempted to transfer 50 from the current account to the kids’ account.
* Validated the values: the bank should have rejected the attempt to transfer from the kids’ account, but the money from the current should have arrived. Our expected values, then, were 50 in the current account and 150 in the kids’, and our actuals matched that.
* Finally, attempted to withdraw 100 and then deposit 100 into the kids’ account, and validated the values. As withdraw should have been rejected, 250 is the expected number in the account, and the actual matched.

Another of our accounts is the savings account. This one is similar to the kids’ account, whereby you cannot withdraw, but it is possible to transfer from it. The testing, thereby, is simple. We:

* Created a savings account as well as a current account, and deposited 100 into each.
* Attempted to withdraw 100 from the savings account and validated that the value was not 0. The application successfully rejects the attempt.
* Attempted to transfer 25 both to and from the savings account and validated that the account lost and then received the correct amount. The application allows the transfer to go through, as anticipated.

Through testing like this, we ensure every element of our application is thoroughly tested and works as intended.

The code for the tests can be found as a JUNit class inside the src folder.

**Part B**

For this part, the only way to test the GUI-based user input was to take screenshots of manual testing. The tabular display was actually very useful for testing purposes as it gives us a constant look into what is happening with and to each thread.

For each test, there was a very clear series of prompts or alterations to the table that should occur, which makes it visually quite easy to make sure everything is expected, especially for the simpler functions. We need only see the table be repopulated to tell that refresh works, for example. For more complicated functions, there’s more involved. For example, for “Search by Name”, we had to:

* Test that clicking the button correctly brought up an input dialog.
* Test that input dialog would only accept threads already contained in the table.
* Test that you could exit the input dialog without entering anything (and that the application wouldn’t crash when you attempted to do so)
* Test that the thread with name matching the user input was correctly found and displayed in the table.

For the full list of tests and screenshots, there is a separate document titled “Part B Tests” as part of the submission.