P3.2 - Transaction Server

Part 2: Implementation

Conrad Murphy

Harshith Shakelli

Zachary Wilson-Long

server.log

- **Open Transactions:**

```
1  Transaction server initiated
2  Accepting incoming transactions
3  ==============================
4
5  [TransactionManagerWorker] Transaction #0 - OPEN_TRANSACTION
6  [TransactionManagerWorker] Transaction #2 - OPEN_TRANSACTION
7  [TransactionManagerWorker] Transaction #9 - OPEN_TRANSACTION
8  [TransactionManagerWorker] Transaction #0 - READ_REQUEST > Account #6
9  [TransactionManagerWorker] Transaction #7 - OPEN_TRANSACTION
10 [TransactionManagerWorker] Transaction #5 - OPEN_TRANSACTION
11 [TransactionManagerWorker] Transaction #4 - OPEN_TRANSACTION
12 [TransactionManagerWorker] Transaction #3 - OPEN_TRANSACTION
```

The first few lines consist of mostly the server opening transactions. We start with brackets showing which class is conducting the printed operation. Then we show which transaction is being conducted upon by showing the transaction's id preceded by a #.

- **Closed Transactions:**

```
60 [TransactionManagerWorker] Transaction #6 - WRITE_REQUEST > Account #9, new ba
61 [TransactionManager] Transaction #7 - Failed Validation
62 [TransactionManagerWorker] Transaction #7 - CLOSED_TRANSACTION - ABORTED
63 [TransactionManager] Transaction #0 - Successfully Validated
64 [TransactionManagerWorker] Transaction #0 - CLOSED_TRANSACTION - COMMITTED
65 [TransactionManager] Transaction #4 - Failed Validation
```

Once a transaction has completed its validation (successfully or unsuccessfully) we display whether it was aborted (if validation is unsuccessful) or if it was committed (if validation is successful). On the client side which will be shown in the client log, if a transaction is aborted it is reattempted by the client but as far as the server is concerned the transaction is dead and the new attempt will come back as a new transaction.

- **Read Request:**

```
13 [TransactionManagerWorker] Transaction #4 - READ_REQUEST > Account #6
```

If a read request is made we simply state the transaction id and which account they are trying to read, pretty simple.

- **Write Request:**

```
15 [TransactionManagerWorker] Transaction #3 - WRITE_REQUEST > Account #7, new balance $1
```

Similarly to the read request, the write request is simple as well. We show which transaction id is writing on what account number. Additionally however we show what the new balance of the account is after the write.

- **Failed Validation:**

```
48  [TransactionManager] Transaction #9 - Failed Validation
49  [TransactionManagerWorker] Transaction #1 - READ_REQUEST > Account #9
50  [TransactionManagerWorker] Transaction #2 - WRITE_REQUEST > Account #1, new balance $15
51  [TransactionManagerWorker] Transaction #8 - WRITE_REQUEST > Account #5, new balance $13
52  [TransactionManagerWorker] Transaction #6 - WRITE_REQUEST > Account #7, new balance $0
53  [TransactionManagerWorker] Transaction #1 - WRITE_REQUEST > Account #9, new balance $12
54  [TransactionManagerWorker] Transaction #9 - CLOSED_TRANSACTION - ABORTED
```

When a transaction fails we show the transaction id and say it failed as such. Which determines that later on when the transaction is closed, that it is aborted.

- **Successful Validation:**

```
63  [TransactionManager] Transaction #0 - Successfully Validated
64  [TransactionManagerWorker] Transaction #0 - CLOSED_TRANSACTION - COMMITTED
```

Much like with failed validation we state which transaction succeeded followed later by a log on that same transaction closing but this time being committed.

clientdriver.log

- **Final Balance Check:**

```
215  [Transferring $6 from account #3 to #2]
216  Transaction failed, restarting...
217  Transaction successful
218  Transaction successful
219
220  Valid data confirmed! Accounts started and ended with 100 total money.
221
222
223  Valid data confirmed! Accounts started and ended with 100 total money.
224
225
226  Valid data confirmed! Accounts started and ended with 100 total money.
```

After a client has successfully completed all *n* of its transactions, it submits one final transaction which simply reads from every account and adds up the total. If the total matches what is expected (number of accounts times starting balance in each), the output is “Valid data confirmed!...” Note that we also have an output statement for the total failing to match, shown below.

```
else
{
    System.out.printf("\nINVALID DATA! Accounts started with %d money, "
            + "ended with %d.\n\n", totalMoney, totalSoFar);
}
```