```
/*
 Client pseudo code

 General Data-Structures:
        enum Outcome { Processed, InconsistentWithHistory, InsufficientFunds }
        class Reply {
                string reqId;
                Outcome outcome;
                float bal;
        }

        enum Operation { GetBalance, Deposit, Withdraw, Transfer }

        class Request {
                string reqId,
                string bankId;
                string accountNum;
                float amount;
                string destBankId;
                string destAccountNum;
        }

        List reqList { reqId, Reply }                   // to maintain the history of requests
        Map bankServer { bankId, head, tail }           // map between bank and its servers


 Events:
        - Failure:
                - The client receives a response from the master that the head has failed.
                - Update the new head for corresponding bank.
                - Check to see if there is any pending request.
                  If yes, resend the request depending upon the config file flag.

        - Receive:
                - The client receives a response from the tail, it could be a success or failu
re.
                - Update the corresponding request in the reqList.

 Functions:
        - update:
                - deposit:
                        args:   reqId, accountNum, bankId, amount
                        return: {reqId, Outcome, bal}
                - withdraw:
                        args: reqId, accountNum, bankId, amount
                        return: {reqId, Outcome, bal}
                - transfer:
                        args: reqId, accountNum, bankId, amount, destAccountNum, destBankId
                        return: {reqId, Outcome, bal}
                        - This function will have a different signature, instead of passing on
e bank as in the above two cases, we have to pass two banks here.

        - Query:
                - getBalance:

        - checkTimeout:
                - probes the reqList to find out which request has not been catered yet.
                - send that request if the number of retries is smaller than the max retries p
ossible and updates the number of retries.
                - In case of update opeation, the client will query the tail if the reqId is p
resent in its history. If yes, then it won't resend otherwise it will resend the request.
*/
```

```
        /* Load the constants from the config file */

        // Callback function to handle the failure event notification from master
        event failure(bankId, serverId, flag):
                // update the head/tail depending upon the flag for the corresponding bank
                if(flag)
                        updateHead(bankId, serverId);
                else
                        updateTail(bankId, serverId);

                // invoke the request (query/update) corresponsing to the reqId from reqList
                invokeReq(reqId);
        end

        // callback function to handle the responses from server
        event receive(reply):
                // update the response in the reqList
                synchronize(reqList) {
                        updateReqList(reply.reqId, reply);
                }
        end

        // function to perform a query operation
        function query(bankId, accountNum):
                reqId = genUniqueReqId();
                tail = getTail(bankId);         // at least one server is always alive

                req = new Request(reqId, bankId, accountNum);
                initialize numRetries;
                repeat
                        send(Operation:GetBalance, req, tail);
                        pushReqList(reqId, new Reply());        // push the corresponding req
in the reqList
                                                                // there is no deletion from l
ist
                                                                // no synchronization needed

                        wait for TIMEOUT:
                                 synchronize(reqList) {
                                        if (checkReqList(reqId)):       // check to see if the
re is a response
                                                                // corresponding to the reqId
in the reqList
                                }
                                        return;
                        numRetries++;
                        until numRetries < MaxRetries
        end

        // The update function is separate from the query because the resend logic is differen
t.
        // The query operation is idempotent so the resend logic will be simple, but in case o
f update operations
        // we have explicitely ensure that the resend operation is idempotent
        function update(bankId, accountNum, Operation, amount, destBankID [optional], destAcco
untNum [optional]):
                reqId = getUniqueReqId();
                head = getHead(bankId);

                req = new Request(reqId, bankId, accountNum, amount, destBankId, destAccountNu
m);
                initialize numRetries;
```

```
                repeat
                        send(Operation, req, head);

                        pushReqList(reqId, new Reply());        // push the corresponding req
in the reqList
                                                                // there is no deletion from l
ist
                                                                // no synchronization needed

                        wait for TIMEOUT:
                                synchronize(reqList) {
                                        if(checkReqList(reqId)):
                                }
                                        return;
                                else
                                        flag = queryServer(reqIdi); // check to see if the req
 is processed and
                                                                    // response is lost
                                        if(flag)
                                                return;
                        numRetries++;
                        until numRetries < MaxRetries;
        end
```