

```

/*
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 failure.
    - Update the corresponding request in the reqList.

Functions:
- update:
    - deposit:
        args: reqId, accountNum, bankId, amount
        return: reply:{reqId, Outcome, bal}
    - withdraw:
        args: reqId, accountNum, bankId, amount
        return: reply:{reqId, Outcome, bal}
    - transfer:
        args: reqId, accountNum, bankId, amount, destAccountNum, destBankId
        return: reply:{reqId, Outcome, bal}
        - This function will have a different signature, instead of passing one
        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 possible
    and updates the number of retries.
    - In case of update operation, the client will query the tail if the reqId is present
    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) corresponding 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) { // take lock on the 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 list
ist // no synchronization needed

        wait for TIMEOUT:
            synchronize(reqList) { // take lock on the reqList
                flag = checkReqList(reqId) // check to see if the
re is a response // corresponding to the reqId
in the reqList
            }
            if(flag) // if true then return else continue
                return;
            numRetries++;
            until numRetries < MaxRetries
end

// The update function is separate from the query because the resend logic is different.
// The query operation is idempotent so the resend logic will be simple, but in case of update operations
// we have explicitly ensure that the resend operation is idempotent
function update(bankId, accountNum, Operation, amount, destBankID [optional], destAccountNum [optional]):
    reqId = genUniqueReqId();
    head = getHead(bankId);

    req = new Request(reqId, bankId, accountNum, amount, destBankId, destAccountNum);

```

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

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

        wait for TIMEOUT:
            synchronize(reqList) {
                flag1 = checkReqList(reqId)
            }
            if(flag1)           // if true then return else continue
                return;
            else
                flag2 = queryServer(reqId); // check to see if the req
is processed and
                                                       // response is lost
                if(flag2)       // if true then return else continue
                    return;
        numRetries++;
        until numRetries < MaxRetries;

    end
```