```
/*
 Server pseudo code

 General Data-Structures

        ServertType type;
        Server successor;
        Server predecessor;

        class SyncMsgContext  {
                string seqNum;
                Request req;
                Operation opr;
                Reply reply;
        }

        List sentReq {  SyncMsgContext }  // we can use it as a sent updates which are sent to
 the successor
                                          // but are not received and processed by the tail
        List historyReq { RequestContext }

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

        enum Outcome { Processed, InconsistentWithHistory, InsufficientFunds }

        class Reply {
                string reqId;
                Outcome outcome;
                float bal;
        }

        enum Operation { GetBalance, Deposit, Withdraw, Transfer }
        enum serverRelation { successor, predecessor }
        enum serverType { Head, Internal, Tail }

 Events:

        - sync
                - receive sync requests from predecessor server
                - apply the update
                - propagate the update to the successor

        - receive
                - event to receive the requests from the clients.
                - The requests can have four kind of operations.
                - Update request will be sent to the head server. Head will send sync message
to next server in the chain.
                - Query request will be sent to the tail server. The tail will send the reply
back to the client.
                - assign a unique sequence number to each received request.

        - failure
                - event to handle the failure of any adjacent server in the chain
                - this will just update the current value of the successor/predecessor
                - if the received server relation is predecessor
                - it will also send the master the sequence number of the last request it rece
```

```
ived
                - If the received server relation is successor,
                - then send all the updates greater than the received seqNum to the new succes
sor
                - the predecessor will halt until it sends all the pending updates to the succ
essor

        - failureHeadTail
                - event to handle the failure of head or tail
                - update the serverType if you are the new head/tail
                - If tail, check for the pending response for transfer operation

        - ack
                - will receive the ack from the next server in the chain (successor)
                - delete all the req from the pendingReq list smaller than the seqNum received
                - send the ack to the predecessor

        - checkReq
                - check to see if the req with the corresponding reqId
                - is present in the history

 Functions:
        - failure
                - If the server has exceeded its sent or receive count then it should terminat
e
                - It has to check whether the configuration is set to "unbound", under which i
t won’t terminate
                - This function will do nothing but exit(0)


        - sendHeartBeat
                - send the heart beat signal to the master

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

        // callback function to receive sync request from the predecessor server in the chain
        event sync(SyncContextMsg msg):
                if(serverType == Tail && msg.opr == Transfer) {
                        // fetch req details from msg context
                        req = new Request(reqId, bankId, accountNum, amount, null, null);
                        head = getHead(req.destBankId); // send deposit to dest bank
                        // wait for reply from dest
                        msg.reply  = sendReq(Operation.Deposit, req, head);
                        // since query is done on tail server
                        sendResponse(msg.reply); // send response to client
                        sendAck(seqNum, predecessor); // send ack to the predecessor
                }
                applyUpdates(msg);      // update local history with msg context

                if(successor != null)
                        sendSyncUpdate(successor, msg);
                else
                        sendResponse(msg.reply); // if tail then send response to client
                        sendAck(seqNum, predecessor); // send ack to the predecessor
        end

        // callback function to receive the request from clients
        event receive(Operation opr, Request req):
                switch(opr):
                        case GetBalance:
                                bal = retrieveBal(req);
                                reply = new Reply(req.reqId, Outcome.Processed, bal);
                                sendResponse(reply);    // since query is done on tail server
                                break;

                        case Deposit:
                                flag = checkIfAlreadyProcessed(req);    // req already present
 in history
                                if(!flag):
                                        bal  = applyDeposit(req);
                                        reply = new Reply(req.reqId, Outcome.Processed, bal);
                                else {
                                        bal = retrieveBal(req);
                                        reply = new Reply(req.reqId, Outcome.InconsistentWithH
istory, bal);
                                }
                                seqNum = generateSeqNum();
                                msg = new SyncContextMsg(seqNum, req, opr, reply);
                                sendSyncUpdate(successor, msg); // since update is done on hea
d
                                break;

                        case Withdraw:
                                flag = checkIfAlreadyProcessed(req);    // req already present
 in history
                                if(!flag):
                                        bal  = applyWithdraw(req);
                                        if(bal < 0) {
                                                bal = retrieveBal(req);
                                                reply = new Reply(req.reqId, Outcome.Insuffici
entFunds, bal);
                                        }
                                        else {
                                                reply = new Reply(req.reqId, Outcome.Processed
, bal);
                                        }
                                else {
                                        bal = retrieveBal(req);
                                        reply = new Reply(req.reqId, Outcome.InconsistentWithH
istory, bal);
                                }
                                seqNum = generateSeqNum();
                                msg = new SyncContextMsg(seqNum, req, opr, reply);
                                sendSyncUpdate(successor, msg); // since update is done on hea
d
                                break;

                        case Transfer:
                                flag = checkIfAlreadyProcessed(req);    // req already present
 in history
                                if(!flag):
                                        bal = applyWithdraw(req);       // withdraw the amount
 from source
                                        if(bal < 0) {
                                                bal = retrieveBal(req);
                                                reply = new Reply(req.reqId, Outcome.Insuffici
entFunds, bal);
                                        }
                                else {
                                        bal = retrieveBal(req);
                                        reply = new Reply(req.reqId, Outcome.InconsistentWithH
istory, bal);
                                }
                                seqNum = generateSeqNum();
                                msg = new SyncContextMsg(seqNum, req, opr, reply);
                                sendSyncUpdate(successor, msg); // since update is done on hea
d
                                break;
                // check for failure condition after every req
                failure();
        end

        // callback function to handle the failure updates from master
        // Master notifies the server with ServerId about the new predecessor/successor becaus
e of a failed server
        event failure(serverId, serverRelation, seqNum):
                switch(serverRelation):
                        case Successor:
                                successor = serverId;   // new successor of the server which l
istens to this event
                                sendUpdates(seqNum, successor); // send all the updates >seqNu
m to the new successor
                                break;
                        case predecessor:
                                predecessor = serverId; // new predecessor of the server which
 listens to this event
                                seqNum = retrieveLastSeqNum();  // Retrieve the seqNum of the
last request
                                                                // handled by this server whic
h has received this request
                                sendAckMaster(seqNum);
                                break;
        end

        // callback to handle the failure of head/tail server
        event failureHeadTail(serverType):
                updateServerType();     // update its own server type
                if(serverType == Tail) {
                        flag1 = checkTransferReq(sentReq);      // check if there’s any transf
er req
                                                                // without any response
                        repeat until flag1
                                flag2 = queryServer(reqId);     // if req is there in dest his
tory then continue
                                        if(flag2)
                                                flag1 = false
                                        else
                                                // wait to get the reply
                                                msg.reply = sendReq();  // else resend the req
                                                // since query is done on tail server
                                                sendResponse(msg.reply); // send response to c
lient
                                                sendAck(seqNum, predecessor); // send ack to t
he predecessor

                        }
                }
        end

        // callback function to handle the ack from the successor server in the chain
        // and sending the ack to the predecessor
        event ack (seqNum):
                deleteReq(seqNum);      // delete all the requests from the sentReq List small
er than seqNum
                sendAck(seqNum, predecessor);
        end

        event checkReq(reqId):
                if reqId in historyReq
                        sendResponse(TRUE)
                else
                        sendResponse(FALSE)
        end

        // function to check for the failure condition
        // called after every receive event
        function failure:
                if(currSendCnt >= MaxSendCnt || currReceiveCnt >= MaxReceiveCnt)
                        exit(0)
        end

        // function to send the heartbeat signal to master
        // called every second
        function sendHeartBeat:
                sendAckMaster(ownServerId);
        end
```