APPLICATION FOR MANAGING CLIENTS IN A BANK

Description:

The service provided by this application is that of managing clients operations in a bank. There will be CRUD operations, i.e. create, read, update and delete a client account. The main characteristic of our service is that it will be distributed and to do so we rely on zookeeper. Moreover, the interaction with the user will be a simple text-based interface where the client can execute any of the operations.

The three main aspects we are going to address in this project is listed below.

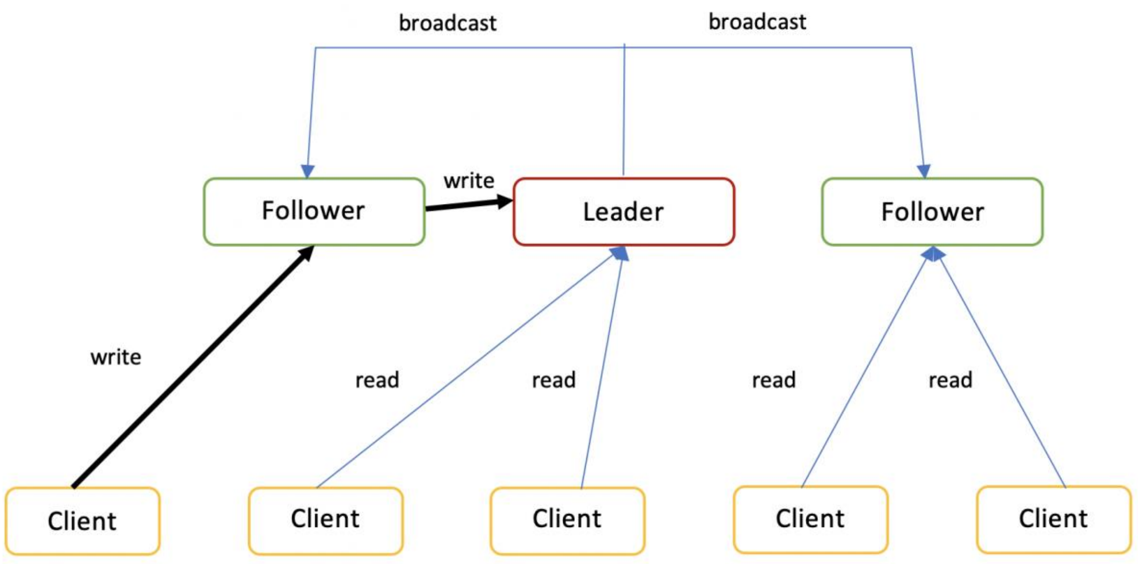
1. **Fault Tolerance** - in order to make the system fault tolerant, it will be distributed in several virtual server. Therefore, if one fails another server will replace it and handle the requests. This will be handled by the Zookeeper ensemble.

2. **Consistency** - As far as it goes for the consistency, Zookeeper guarantees it will have it. In fact, the two guarantees of a Zookeeper system are Consistency and Partition Tolerance. In particular, the atomic broadcast and the leader election through quorum will assure a consistent view of the system. Finally every, write operation on the database will have to go through the leader, which will broadcast it to the follower.

3. **Availability** - for the system to be able to guarantee its availability at all time, we will have three different server nodes running at all times. So, the client can access the service at any time even with a slight delay due to the priority given to the consistency.

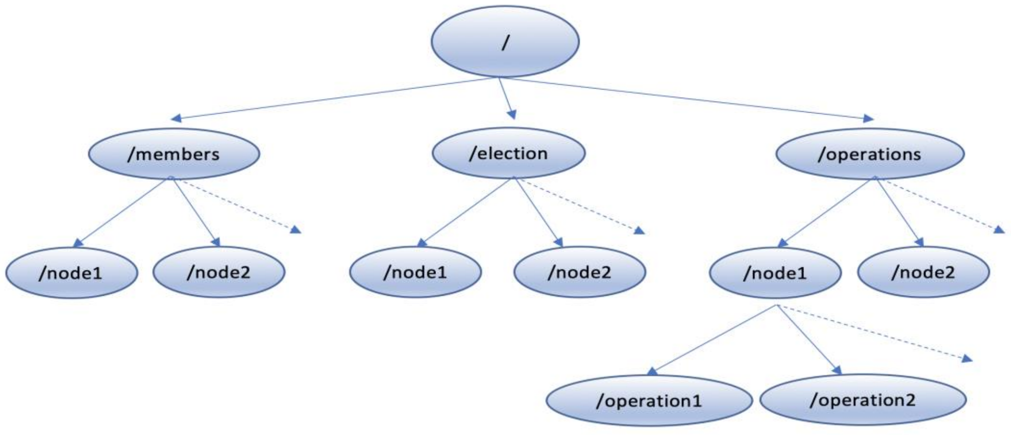
System Architecture and Software Design:

The system architecture diagram below shows how Zookeeper handles the requests from the clients. On top, we have three servers with one leader and two followers, as to guarantee consistency (leader handling the write requests), fault tolerance and availability (having three server running at the same time).



Data structure of the ZNodes of Zookeeper

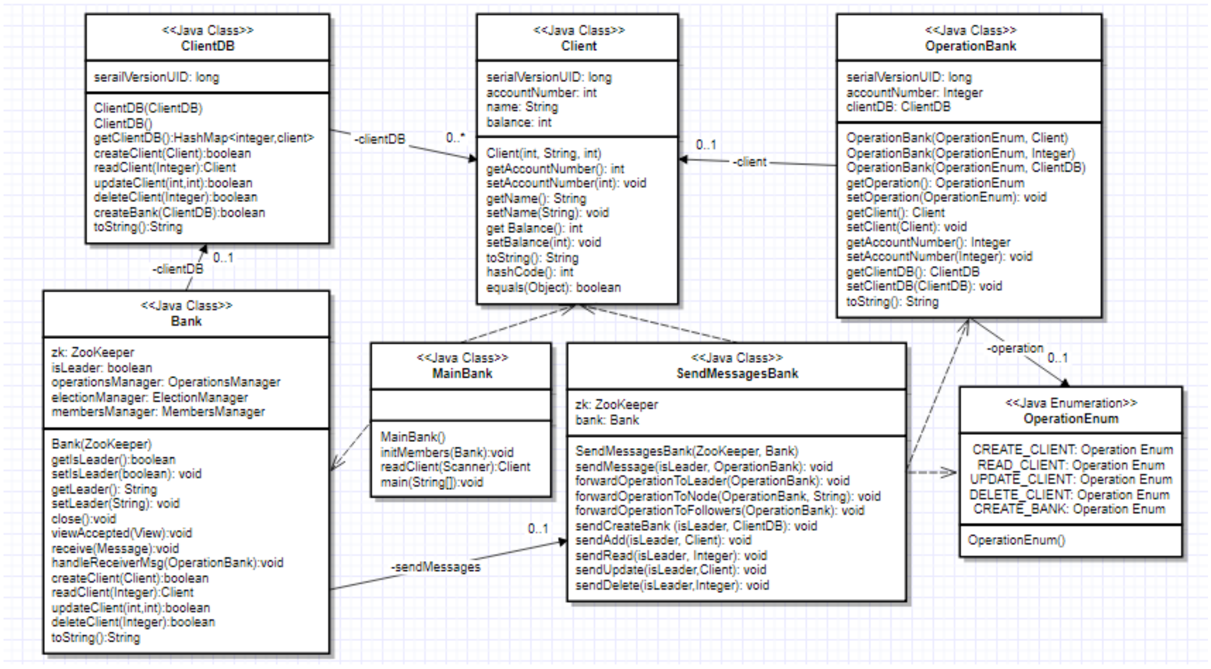
This is a basic data structure of Zookeeper and shows which kind of nodes we will have in our application. Members for each process running. Election, which will handle the leader election, most probably choosing the server with lowest id . Operation, which will contain all the information about the operation a client can execute on our system.



UML Diagrams

Below we present Class diagram and Sequence diagrams. Class diagram shows the structure of the system where as Sequence diagrams shows the interaction between various actors involved in CRUD operations of this application.

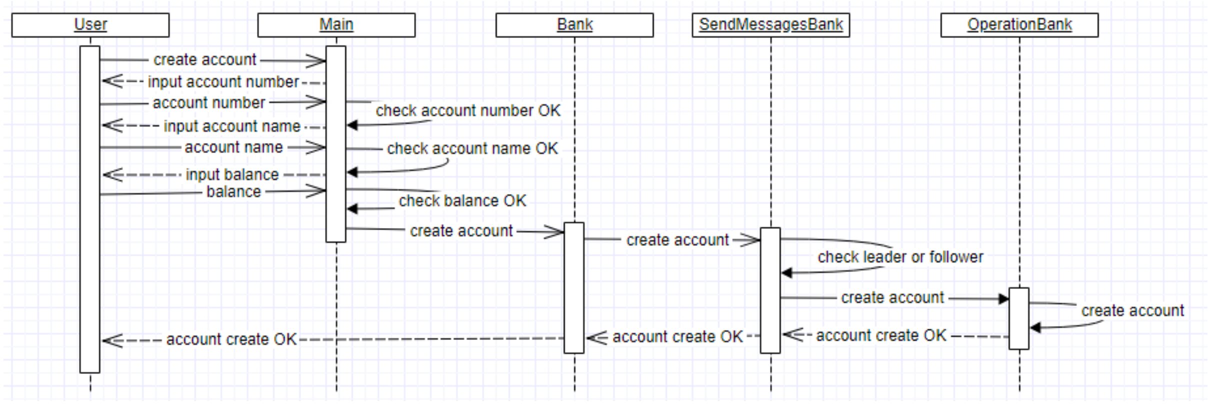
Class Diagram



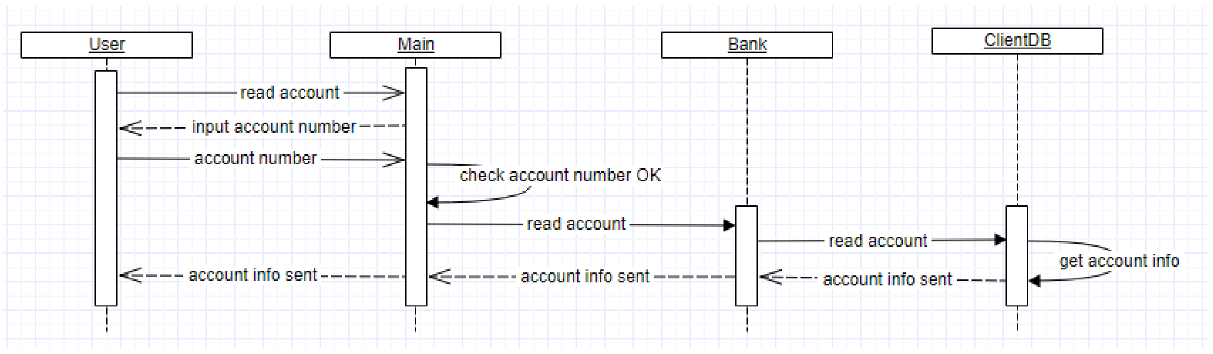
<<Java Class>>
ClientDB
serailVersionUID: long
ClientDB(ClientDB)
ClientDB()
getClientDB():HashMap<integer,client>
createClient(Client):boolean
readClient(Integer):Client
updateClient(int,int):boolean
deleteClient(Integer):boolean
createBank(ClientDB):boolean
toString():String
-clientDB
0..*
<<Java Class>>
Client
serialVersionUID: long
accountNumber: int
name: String
balance: int
Client(int, String, int)
getAccountNumber(): int
setAccountNumber(int): void
getName(): String
setName(String): void
get Balance(): int
setBalance(int): void
toString(): String
hashCode(): int
equals(Object): boolean
0..1
-client
<<Java Class>>
OperationBank
serialVersionUID: long
accountNumber: Integer
clientDB: ClientDB
OperationBank(OperationEnum, Client)
OperationBank(OperationEnum, Integer)
OperationBank(OperationEnum, ClientDB)
getOperation(): OperationEnum
setOperation(OperationEnum): void
getClient(): Client
setClient(Client): void
getAccountNumber(): Integer
setAccountNumber(Integer): void
getClientDB(): ClientDB
setClientDB(ClientDB): void
toString(): String
-clientDB
0..1
<<Java Class>>
Bank
zk: ZooKeeper
isLeader: boolean
operationsManager: OperationsManager
electionManager: ElectionManager
membersManager: MembersManager
Bank(ZooKeeper)
getIsLeader():boolean
setIsLeader(boolean): void
getLeader(): String
setLeader(String): void
close():void
viewAccepted(View):void
receive(Message):void
handleReceiverMsg(OperationBank):void
createClient(Client):boolean
readClient(Integer):Client
updateClient(int,int):boolean
deleteClient(Integer):boolean
toString():String
<<Java Class>>
MainBank
MainBank()
initMembers(Bank):void
readClient(Scanner):Client
main(String[]):void
<<Java Class>>
SendMessagesBank
zk: ZooKeeper
bank: Bank
SendMessagesBank(ZooKeeper, Bank)
sendMessage(isLeader, OperationBank): void
forwardOperationToLeader(OperationBank): void
forwardOperationToNode(OperationBank, String): void
forwardOperationToFollowers(OperationBank): void
sendCreateBank (isLeader, ClientDB): void
sendAdd(isLeader, Client): void
sendRead(isLeader, Integer): void
sendUpdate(isLeader,Client): void
sendDelete(isLeader,Integer): void
0..1
-sendMessages
-operation
0..1
<<Java Enumeration>>
OperationEnum
CREATE_CLIENT: Operation Enum
READ_CLIENT: Operation Enum
UPDATE_CLIENT: Operation Enum
DELETE_CLIENT: Operation Enum
CREATE_BANK: Operation Enum
OperationEnum()

Sequence Diagrams

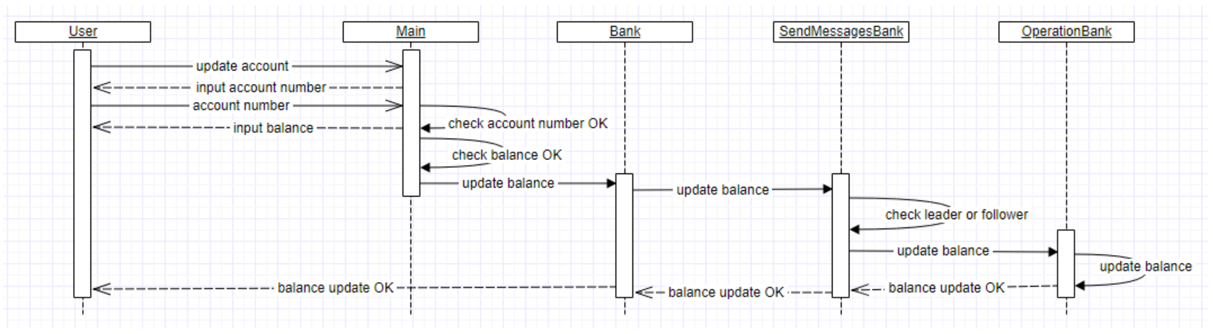
Create



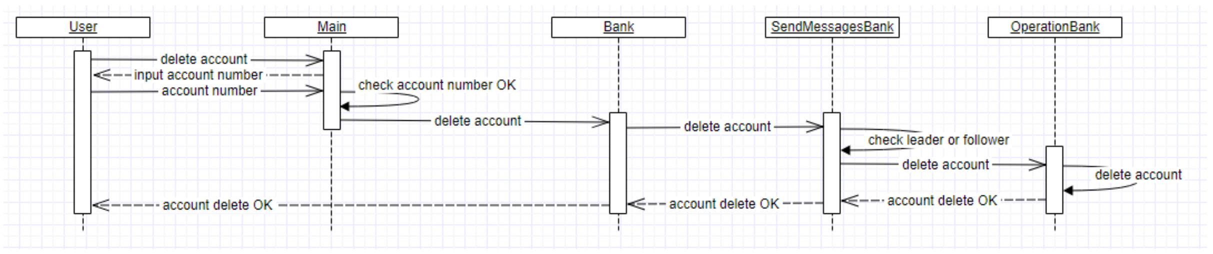
Read



Update



Delete


User
Main
Bank
SendMessagesBank
OperationBank
delete account
input account number
account number
check account number OK
delete account
delete account
check leader or follower
delete account
delete account
account delete OK
account delete OK
account delete OK