Distributed Banking System

Team Members:

Sneha Shukla(201917009)
Prince Mishra(201917007)

Report submitted for the
Final Project Review of

Course Code: IT559 - Distributed Systems

to

Professor: Prof. Kalyan Sasidhar

TABLE OF CONTENTS

1. Introduction

1.1 Theoretical background

In this age of technological advancements, individuals are moving towards Internet for trade and business. Individuals prefers web-based business applications over manual-based in order to satisfy their day-to-day needs. Internet Banking has made this possible by managing things remotely. Online Banking Applications are reliable platform for handling numerous users request on everyday basis. In this whole process of developing prototype of distributed banking system, we have followed a effective level of abstraction and modulation.

1.2 Objective

Main purpose to implement this project was to learn how to use RMI (Remote Method Invocation) and automate the Banking system being followed by banking company that deals in current account with or without check facility. The secondary objectives are as follows:

- To increase the number of accounts and customer. This will reduce the manual workload and give information instantly.
- The application is user friendly so that even a beginner can operate the application and thus maintain the status of account and balance status easily.

2. Software Requirements

- JDK (Java Development Kit)
- JRE (Java Runtime Environment)
- Windows Command Line (module of Kali Linux)

3. Proposed Work

3.1 Aim of the proposed Work

The aim of this work is to create a multithreaded client-server based program which will communicate via Java RMI (Remote Method Invocation). Here, the client i.e, ATM will initiate any bank related operation by calling a remote method on Bank server to execute some basic transaction functionalities in a concurrent way under an active session.

3.2 Problem Statement

The system will consist of two programs.

- One of these is the service component, and will represent the bank.
- The bank has a number of accounts; these accounts can be created(with an initial deposit value) and destroyed.In addition, accounts can have money deposited, can have their account balance queries, and can have money withdrawn.

The interface should be command-line based, and allow all (and only) the following commands:

- startingBalance(AcctNumber,name)
- destroy(AcctNumber)
- deposit(AcctNumber,Amount)
- withdraw(AcctNumber)
- inquiry(AcctNumber)
- Statement(AcctNumber,startDate,endDate)

The communication between different machine is done with RMI(Remote Method Invocation).

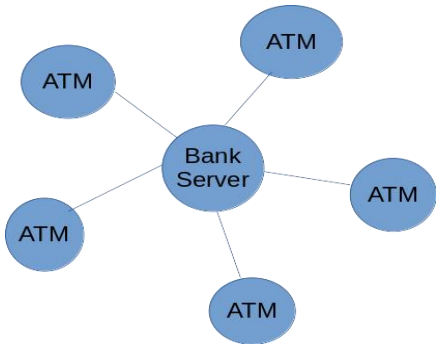

Fig 1. Distributed Banking System

4. **Proposed System Model**

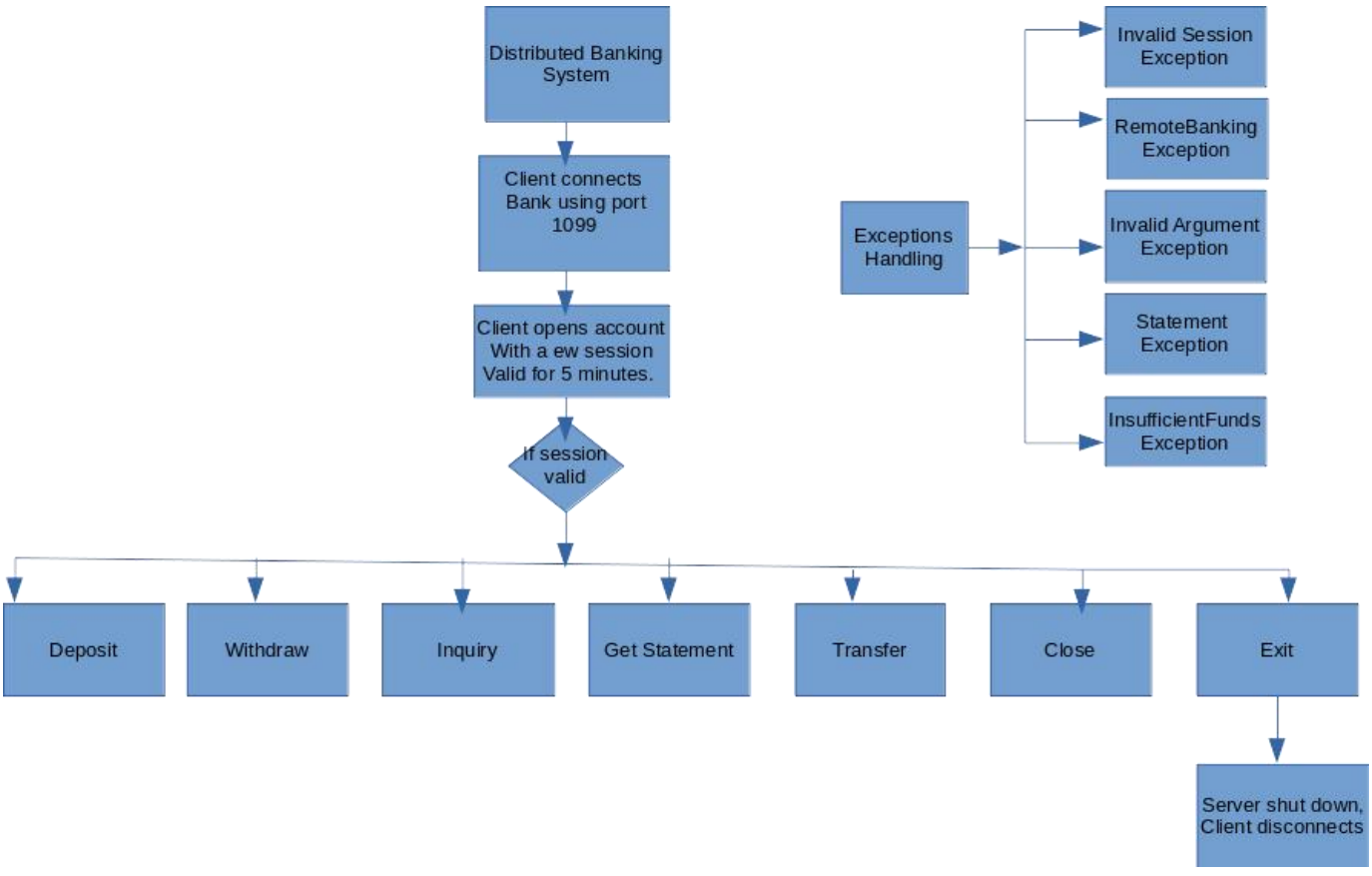

Fig2. Workflow Model

5. Code and Implementation details

5.1 Interface Module

5.1.1 AccountInterface Implementation

Following methods have been implemented in AccountInterface.java file:

1) *public void addTransaction(String type, double amount);*
   - This method takes transaction type as parameter which can deposit or withdraw, and amount as a second parameter which can be either added or deduced from their bank account.
   - This method is implemented in Account.java file that simply adds the transaction to the list of existing transactions.

2) *public List<Transaction> getTransactions();*
   - This method is implemented in Account.java which generates list of transactions for the indicated account number

3) *public List<Transaction> getTransactionsByDate(Date fromDate, Date toDate);*
   - This method takes fromDate and toDate as parameter .
   - This method is implemented in Account.java which returns list of transactions between the given dates , if transaction has occurred for specified dates.

4) *public int getAccountNum();*
   - This method is implemented in Account.java which returns account number, if queried.

5) *public void setAccountNum(int accountNum);*
   - This method is implemented in Account.java which set the account number when user opens their account with bank.

6) *public double getBalance();*
   - This method is implemented in Account.java which fetches the balance for the indicated account number.

7) *public void setBalance(double balance);*
   - This method is implemented in Account.java that takes balance as a parameter to set the balance for the indicated account number.

8) *public String getAccountName();*
   - This method is implemented in Account.java which fetches the user name for the queries account number.

9) *public void setAccountName(String accountName);*
   - This method is implemented in Account.java which takes account holder name as parameter and set the user name for the queried account number.

```java
package interfaces;

import server.*;
import client.*;
import java.util.Date;
import java.util.List;

/**
* Written by @Sneha and @Prince
* AccountInterface which has methods signatures
* which are implemented in Account class.
**/
public interface AccountInterface {

        public void addTransaction(String type, double amount);
        public List<Transaction> getTransactions();
        public List<Transaction> getTransactionsByDate(Date fromDate, Date toDate);
        public int getAccountNum();
        public void setAccountNum(int accountNum);
        public double getBalance();
        public void setBalance(double balance);
        public String getAccountName();
        public void setAccountName(String accountName);

}
```

5.1.2 **BankInterface Implementation**

This interface will declare all the methods that are overriden in Bank.java class file.

1) *public String startingBalance(int accountNum,String name) throws RemoteException, RemoteBankingException,InvalidArgumentException;*
    - This will throw remote exception if while opening account with Bank, Client applications loses connection with server.
    - This will throw RemoteBankingException if account already exists or doesn't exists.
    - This will throw InvalidArgument Exception if provided name is in invalid format.
    - It takes account Number and client name as parameter.
    - This method is overriden in Bank.java file which return balance along with message.

2) *public double closeAccount(int accountNum,long sessionID) throws RemoteException, RemoteBankingException,InvalidSessionException;*
    - This will throw remote exception if while closing account , Client losts connection with Bank(server).
    - This will throw RemoteBankingException if account already exists or doesn't exists.
    - This will throw InvalidSession Exception if the session is dead or invalid.
    - It takes account account Number, and session id.
    - This method is overriden in Bank.java file which returns positive balance if found, else return zero balance.

3) *public double deposit(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException, InvalidSessionException,RemoteBankingException;*

- This will throw remote as well as Invalid Session exception, if someone tries to use expired sessions for their transaction.
- This will throw RemoteBankingException if account already exists or doesn't exists.
- It takes an existing account number, amount to deposit, a valid session id, filename to log deposit entries, current thread executing deposit operation and number of times amount need to be deposited in the bank.
- This method is overriden in Bank.java file which returns new balance after deposit.

4) *public double withdraw(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException, InvalidSessionException,RemoteBankingException;*

- This will throw remote and invalid session exception, if Client losts its connection to server will withdrawing, if dead session is used.
- This will throw RemoteBankingException if account already exists or doesn't exists.
- It takes an existing account number, amount to withdraw if less than or equal to current balance, a valid session id, filename to log withdraw entries, current thread executing withdraw operation and number of times amount need to be withdrawn from the bank.
- This method is overriden in Bank.java which returns the updated balance after withdrawal.

5) *public double inquiry(int accountNum,long sessionID) throws RemoteException,InvalidSessionException,RemoteBankingException;*

- This will throw Remote, RemoteBanking and invalid session exception, if client losts its connection to server will inquiring the balance, or trying to access account with invalid account number, or if the session is dead.
- It takes an existing account number, and an active session id.
- This method is overriden in Bank.java which returns balance for an indicated account number.

6) *public java.util.List<Transaction> getStatement(int accountNum, Date from,Date to,long sessionID) throws RemoteException,InvalidSessionException,StatementException,RemoteBankingException;*

- This will throw statement exception apart from remote ,RemoteBanking and session exception, if transaction is not found for given dates, connection refused to connect with server,account doesn't exists and if session expires.
- It takes an existing account number, from date, to date and an active session id.
- This method is overriden in Bank.java which returns list of transactions between given data for an indicated account number.

7) *Public double getTotalBalance() throws RemoteException;*

- This will throw Remote exception, if connection refused to connect with server.
- It returns the total balance of all the accounts

.

8) *public void stopServer() throws RemoteException;*

- This will throw remote Exception, if connection refused to connect with server.
- It unbinds the registry that was used to create connection and unicasts remote object and unimport registry object to shut down the server when client requests “exit” operation.
- This method is overriden in Bank.java.

```
package interfaces;

import server.*;
import client.*;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;
import exceptions.*;


/*
Written by: @Sneha and @Prince
BankInterface has method signatures which
are implemented in Bank class.
**/
public interface BankInterface extends Remote {

        public String startingBalance(int accountNum,String name) throws RemoteException,RemoteBankingException,InvalidArgumentException;
        public double closeAccount(int accountNum,long sessionID) throws RemoteException,RemoteBankingException,InvalidSessionException;
        public double deposit(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException,
InvalidSessionException,RemoteBankingException;
        public double withdraw(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException,
InvalidSessionException,RemoteBankingException;
        public double inquiry(int accountNum,long sessionID) throws RemoteException,InvalidSessionException,RemoteBankingException;
        public double getTotalBalance() throws RemoteException;
        public void stopServer() throws RemoteException;
        public java.util.List<Transaction> getStatement(int accountNum, Date from, Date to,long sessionID) throws
RemoteException,InvalidSessionException,StatementException,RemoteBankingException;
}
```

5.2 **Server Module**

5.2.1 **Account Implementation**

Private data members declared are defined below:

1) private int accountNum;
2) private String accountName;
3) private double balance;
4) private List<Transaction> transactions;

Methods functionality has been already discussed above, are implemented as defined below:

1. public Account (int accountNum, String accountName, double openingBalance):
2. public void addTransaction(String type, double amount)
3. public List<Transaction> getTransactions()
4. public List<Transaction> getTransactionsByDate(Date fromDate, Date toDate)
5. public int getAccountNum()
6. public void setAccountNum(int accountNum)
7. public double getBalance()
8. public void setBalance(double balance)
9. public String getAccountName()
10. public void setAccountName(String accountName)

Code Screenshots

```java
/*
* Written by: @Sneha and @Prince
* Account class which implements AccountInterface and Serializable
* It holds user account number, name, transactions list
* and current balance.
**/

package server;

import interfaces.*;
import client.*;
import runnable.*;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Account implements AccountInterface,Serializable{

	private int accountNum;
	private String accountName;
	private double balance;

	// list of transaction associated with an account object
	private List<Transaction> transactions;

	// Account class constructor
	public Account (int accountNum, String accountName, double openingBalance) {

		this.setAccountNum(accountNum);
		this.setAccountName(accountName);
		this.setBalance(openingBalance);
		transactions = new ArrayList<Transaction>();
	}
```

```java
	// adding a transaction object to the list of transactions
	public void addTransaction(String type, double amount) {
		Transaction e = new Transaction(type, amount, getBalance());
		transactions.add(e);
	}

	// return all transactions
	public List<Transaction> getTransactions() {
		return transactions;
	}

	// return all transactions within a specified date range
	public List<Transaction> getTransactionsByDate(Date fromDate, Date toDate) {

		List<Transaction> statementList = new ArrayList<Transaction>();

		for (int i=0; i<transactions.size(); i++) {
			Transaction element = transactions.get(i);

			// check if the date value falls between the specified range
			if (element.getTransactionDate().after(fromDate) && element.getTransactionDate().before(toDate)) {
				statementList.add(element);
			}
		}
		return statementList;
	}
```

```java
	//setters and getters
	public int getAccountNum() {
		return accountNum;
	}

	public void setAccountNum(int accountNum) {
		this.accountNum = accountNum;
	}

	public double getBalance() {
		return balance;
	}
	public void setBalance(double balance) {
		this.balance = balance;
	}

	public String getAccountName() {
		return accountName;
	}

	public void setAccountName(String accountName) {
		this.accountName = accountName;
	}
```

5.2.2 **Transaction Implementation**

This module implements transaction related details of every account.

Code screenshots

```java
/*
* Written by: @Sneha and @Prince
* Transaction object that implements Serializable
* stores all list of transactions
* for every Bank account.
**/

package server;

import interfaces.*;
import client.*;
import runnable.*;
import java.io.Serializable;
import java.text.DecimalFormat;
import java.util.Date;

public class Transaction implements Serializable {

	private static final long serialVersionUID = -6841131027488692403L;

	// decimal formatting to 2 decimal places
	private DecimalFormat precision2 = new DecimalFormat("0.00");

	private String transactionType;
	private double transactionAmount;
	private double upToDateBalance;
	private Date transactionDate;

	public Transaction(String transactionType, double transactionAmount, double upToDateBalance){
		this.setTransactionType(transactionType);
		this.setTransactionAmount(transactionAmount);
		this.setUpToDateBalance(upToDateBalance);
		transactionDate = new Date();
	}
```

```java
	public String toString() {
		return "Type: " + transactionType +
				"\nAmount: €" + precision2.format(transactionAmount) +
			"\nBalance: €" + precision2.format(upToDateBalance) +
			"\nDate: " + transactionDate.toString() + "\n";
	}
	//setters and getters
	public Date getTransactionDate() {
		return transactionDate;
	}
	public void setTransactionDate(Date transactionDate) {
		this.transactionDate = transactionDate;
	}
	public String getTransactionType() {
		return transactionType;
	}
	public void setTransactionType(String transactionType) {
		this.transactionType = transactionType;
	}
	public double getTransactionAmount() {
		return transactionAmount;
	}
	public void setTransactionAmount(double transactionAmount) {
		this.transactionAmount = transactionAmount;
	}
	public double getUpToDateBalance() {
		return upToDateBalance;
	}
	public void setUpToDateBalance(double upToDateBalance) {
		this.upToDateBalance = upToDateBalance;
	}
```

Data members that have been declared under this class file:

1) private String transactionType;
2) private double transactionAmount;
3) private double upToDateBalance;
4) private Date transactionDate;

Methods implemented under this class file:

1) *public Transaction(String transactionType, double transactionAmount, double upToDateBalance)*:
    - This is constructor that takes transaction type (withdraw/deposit), transaction amount that can be either deduced or added, new balance after transaction has been performed.
    - It sets date of transaction also.

2) *public String toString()*

- It returns a transaction type, transaction amount, new balance and date of transaction as a string.

3) *public Date getTransactionDate()*

- It takes no parameter and returns transaction date.

4) *public void setTransactionDate(Date transactionDate)*

- It sets transaction Date for ongoing transaction.

5) *public String getTransactionType()*

- It returns the transaction type as a string

6) *public void setTransactionType(String transactionType)*

- It sets the transaction type for an indicated account number.

7) *public double getTransactionAmount()*

- It fetches the transaction amount to display to user after withdraw/deposit.

8) *public void setTransactionAmount(double transactionAmount)*

- It sets the transaction amount after withdraw/deposit.

9) *public double getUpToDateBalance()*

- It fetches the new balance after the transaction.

10) *public void setUpToDateBalance(double upToDateBalance)*

- It sets the new balance after the transaction.

5.2.3 **Session Implementation**

When new user joins bank he is allotted a fresh session id, which he can use to perform other bank operations.

Code screenshots

```
/*
* Written by: @Sneha and @Prince
* Session Object
* extends TimerTask, a thread that can be called at certain time intervals
* this allows the session to time to be incremented every second, and can be cancelled after 5mins (300s)
**/

package server;

import interfaces.*;
import client.*;
import runnable.*;
import java.io.Serializable;
import java.util.Date;
import java.util.Timer;
import java.util.TimerTask;

public class Session extends TimerTask implements Serializable{
    //Instance variables for each session object
    private int timeAlive;
    private Timer timer;
    private volatile boolean alive;
    private Account account;
    public long sessionId;

    //static variables to specify max session time, and timer delay
    private static final int MAX_SESSION_LENGTH = 60 * 5;
    private static final long DELAY = 1000;
```

```java
public Session(Account account) {
    //generate a random 6 digit sessionID
    this.sessionId = (int)(Math.random()*900000)+100000;
    this.account = account;
    this.alive = true;
    this.timeAlive = 0;
    //create timer object to allow the task to be scheduled to run every second
    this.timer = new Timer();
    this.startTimer();
    System.out.println(">> Session " + sessionId + " created\n");
}

private void startTimer() {
    //schedule timer to run every second
    this.timer.scheduleAtFixedRate(this, new Date(System.currentTimeMillis()), DELAY);
}

@Override
public void run() {
    //increment the time the session has been alive
    //updates once every second, so it represents the # of seconds the session has been alive for
    this.timeAlive++;
    //if session has been alive for 5 minutes
    if(this.timeAlive == MAX_SESSION_LENGTH) {
        //set alive to false and cancel the timer
        this.alive = false;
        this.timer.cancel();
        System.out.println("\n--------------------------\nSession " + this.sessionId + " terminated \n--------------------------");
        System.out.println(this);
        System.out.println("--------------------------");
    }
}
```

```java
//Getters and Setters
public boolean isAlive() {
    return this.alive;
}
public void setSessionclose() {
            this.timeAlive = MAX_SESSION_LENGTH ;
}

public long getClientId(){
    return this.sessionId;
}

public int getTimeAlive(){
    return this.timeAlive;
}

public int getMaxSessionLength(){
    return MAX_SESSION_LENGTH;
}

public Account getAccount(){
    return this.account;
}

@Override
public String toString() {
    return "Account: " + this.account.getAccountNum() + "\nSessionID: " +
            this.sessionId +"\nTime Alive: " + this.timeAlive + "\nAlive: " + this.alive;
}
```

Data members that have been declared under this class:

1) private int timeAlive;
2) private Timer timer;
3) private volatile boolean alive;
4) private Account account;
5) public long sessionId;

Modules that have been implemented under this class:

1) *public Session(Account account)*
    - This is constructor that generates random 6 digit session id.
    - It starts the timer for the active session id.

2) *private void startTimer()*
    - This method invokes scheduleAtFixedRate method which is used to schedulethe specified task for repeated fixed-rate execution beginning after the specified delay.

3) *public void run()*

- This method increments the static variable timeAlive until it reaches maximum session duration for an active session.

4) *public boolean isAlive()*

- It returns the boolean variable alive true if the session is still active.

5) *public long getClientId()*

- It returns the generated session Id for a transaction.

6) *public int getTimeAlive()*

- It returns the timeAlive value, how much time is remaining after we subtract this time alive from maximum session duration.

7) *public int getMaxSessionLength()*

- It retrieves the maximum session duration, fixed as 60*5 and delay is 1000 milliseconds.

8) *public Account getAccount()*

- It returns the Account object for an ongoing transaction.

9) *public String toString()*

- It returns account number of an existing user, an active session id, time spent in the form of a string.

5.2.4 **Bank(Server) Implementation**

In our project, this bank will act as server ,providing all the services requested by the clients. All the implemented methods are concurrency protected. Below are the modules that has been implemented as part of Bank application:

Code screenshots

```java
/**
* Written by: @Sneha and @Prince
* Bank class which acts as server,
* will accept the client connectio request
* and process all the operations requested by ATM class.
**/

package server;

import interfaces.*;
import client.*;
import runnable.*;
import java.rmi.Naming;
import exceptions.*;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.*;
import java.io.*;
import java.util.concurrent.locks.*;

// bank acts as the remote server that the client (ATM) connects to
public class Bank extends UnicastRemoteObject implements BankInterface {

	protected static final long serialVersionUID = -8317765732411101420L;
	protected Registry registry = null;


	// users accounts
	protected static List<Account> accounts = new ArrayList<Account>();
	protected List<Session> sessions,deadSessions;
```

```
    private Lock balanceChangeLock;
    private Condition sufficientFundsCondition;
    private static DecimalFormat df2 = new DecimalFormat("#.##");

    Map accounts_map= new HashMap();
    public Bank () throws RemoteException{

            super();

            registry = LocateRegistry.createRegistry(1099);
            System.setProperty("java.security.policy","file:test.policy");

            //initial accounts added
            sessions=new ArrayList<>();
            deadSessions=new ArrayList<>();
            accounts_map.put(1,"Prince");
            accounts_map.put(2,"Pranav");
            accounts_map.put(3,"Vishesh");
            accounts_map.put(4,"Chnadranshu");
            accounts_map.put(5,"Deepak");
            accounts_map.put(6,"Abhishek");

            //to ensure synchronization due to multithreading implementation
            balanceChangeLock = new ReentrantLock();
            sufficientFundsCondition = balanceChangeLock.newCondition();
    }
```

```
    @Override
    // to  open account of a client with provided accountNumber(which is optional) and Name of a new user.
    public synchronized String startingBalance(int accountNum,String name) throws RemoteException, RemoteBankingException,InvalidArgumentException
    {
            long session_id=9999;
            String acc_session=null;
            if(accountNum>100)
            {
                    acc_session="Invalid"+":"+String.valueOf(-2);
                    return acc_session;
            }
            if(accountNum>0 && accountNum<7){
                    acc_session="Invalid"+":"+String.valueOf(-1);
                    return acc_session;

            }
            else if(accountNum>6)
            {
                    if(accounts_map.get(accountNum)!=null)
                            throw new RemoteBankingException("Account already exists.");
                    if(isNumeric(name))
                            throw new InvalidArgumentException("Error:Invalid name. Failed attempt to open account.");
                    Account acc = new Account(accountNum,name,100);
                    accounts_map.put(accountNum,name);
                    accounts.add(acc);
                    Session s=new Session(acc);
                    sessions.add(s);
                    session_id=s.sessionId;
```

```
            else
            {
                        Random rand = new Random();
                        // Generate random accountNumbers in range 0 to 100
                        accountNum = rand.nextInt(101);
                        for (int i=0;i<accounts.size();i++) {
                                Account acc=accounts.get(i);
                                if (acc.getAccountNum()== accountNum) {
                                        accountNum = rand.nextInt(101);
                                }
                                else{

                                        if(accounts_map.get(accountNum)!=null)
                                                throw new RemoteBankingException("Account already exists.");
                                        if(isNumeric(name))
                                                throw new InvalidArgumentException("Error:Invalid name. Failed attempt to open account.");
                                        Account acct=new Account(accountNum,name,100);
                                        accounts_map.put(accountNum,name);
                                        accounts.add(acct);
                                        Session s=new Session(acct);
                                        sessions.add(s);
                                        session_id=s.sessionId;
                                        break;
                                }
                        }
            }
            acc_session=String.valueOf(session_id)+":"+String.valueOf(accountNum);
            return acc_session;
    }
```

```
    //to check whether the provided string contains digits
    public static boolean isNumeric(String str)
    {
        for (char c : str.toCharArray())
        {
            if (!Character.isDigit(c)) return false;
        }
        return true;
    }

    //to close account for a user with provided accountNumber and an active session ID
    @Override
    public synchronized double closeAccount(int accountNum,long sessionID) throws RemoteException, RemoteBankingException,InvalidSessionException{
            double balance=0;
            if(!checkSessionActive(sessionID))
                    throw new InvalidSessionException("Invalid Session!!!");
            if(accounts_map.get(accountNum)==null)
                    throw new RemoteBankingException("Account doesn't exists.");
            for (int i=0;i<accounts.size();i++) {
                    Account acc=accounts.get(i);
                    if (acc.getAccountNum()== accountNum){
                            if(acc.getBalance()==0) {
                                    accounts_map.remove(accountNum);
                                    accounts.remove(new Account(accountNum,acc.getAccountName(),acc.getBalance()));
                                    break;
                            }
                            else{
                                    balance=acc.getBalance();
                            }
                    }
            }
            return balance;
    }
```

```
     //to deposit amount into user account with provided account Number, amount, active session ID
     // filename to log deposit entries, thread which is running this method
     @Override
     public double deposit(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException, InvalidSessionException,
RemoteBankingException{

              balanceChangeLock.lock();
              double bal=0.00;
              try{
                      if(!checkSessionActive(sessionID))
                              throw new InvalidSessionException("Invalid Session!!!");

                      for (int i=0; i<accounts.size(); i++){
                              Account element = accounts.get(i);
                              if (element.getAccountNum() == accountNum){
                                              element.setBalance(element.getBalance() + amount);
                                              element.addTransaction("Deposit", amount);
                                              bal=inquiry(accountNum,sessionID);
                                              String mes=threadName+" "+String.valueOf(x)+" completed deposit ... Current balance after deposit: €"+String.valueOf(bal
+"\n";                                        try{
                                              FileWriter writer = new FileWriter(fileName, true);
                                              writer.write(mes);
                                              writer.close();
                                              sufficientFundsCondition.signalAll();
                                              }
                                              catch(Exception e){}
                                              return bal;
                              }
                      }
              }
              finally{
                      balanceChangeLock.unlock();
              }

              throw new RemoteBankingException("Account doesn't exists!!!");

      }
```

```
        //to withdraw amount from user account if sufficient balance founds, with provided account Number, amount, active session ID
        // filename to log withdraw entries, thread which is running this method
        @Override
        public double withdraw(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException,
InvalidSessionException,RemoteBankingException{

                balanceChangeLock.lock();
                double balance=0.09;
                try{
                // reduce the amount in the account with account number 'accountNum' by 'amount'
                        if(!checkSessionActive(sessionID))
                                        throw new InvalidSessionException("Invalid Session!!!");
                        for (int i=0; i<accounts.size(); i++){
                                Account element = accounts.get(i);
                                if (element.getAccountNum() == accountNum){
                                        balance=element.getBalance();
                                        while(balance<amount)
                                        {
                                                sufficientFundsCondition.await();
                                        }
                                        if(element.getBalance() > 0 && element.getBalance()-amount ≥ 0){
                                                element.setBalance(element.getBalance() - amount);
                                                element.addTransaction("Withdraw", amount);

                                                balance=inquiry(accountNum,sessionID);
                                                String message=threadName+" "+String.valueOf(x)+" completed withdrawal ... Current balance after withdrawal:
₹"+String.valueOf(balance)+"\n";

                                                FileWriter writer = new FileWriter(fileName, true);
                                                writer.write(message);
                                                writer.close();
                                                return balance;
                                        }
                                }
                        }
                }
                catch(InvalidSessionException e){
```

```
                catch(InvalidSessionException e){
                        throw new InvalidSessionException("Invalid Session!!!");
                }
                catch(Exception e)
                {System.out.println(e.getMessage());}
                finally{
                        balanceChangeLock.unlock();
                }
                throw new RemoteBankingException("Account doesn't exists.");
        }

        //to fetch total balance for entire account
        @Override
        public double getTotalBalance() throws RemoteException
        {
                balanceChangeLock.lock();
                double total=0.00;
                try
                {
                        for (int i=0; i<accounts.size(); i++){
                                Account element= accounts.get(i);
                                        total+=element.getBalance();
                        }
                        return total;

                }
                catch(Exception e){

                }
                finally{
                        balanceChangeLock.unlock();
                }
                return total;
        }
```

```
        // to fetch balance for an existing account provided account Number and an active session ID which is valid for ongoing transaction.
        @Override
        public synchronized double inquiry(int accountNum,long sessionID) throws RemoteException, InvalidSessionException,RemoteBankingException {
                // returns the balance of the account with account number 'accountNum'
                balanceChangeLock.lock();
                try{
                        if(!checkSessionActive(sessionID))
                                        throw new InvalidSessionException("Invalid Session!!!");

                        for (int i=0; i<accounts.size(); i++){
                                Account element = accounts.get(i);
                                if (element.getAccountNum() ==accountNum){
                                        return element.getBalance();

                                }
                        }
                }
                finally{
                        balanceChangeLock.unlock();
                }
                        throw new RemoteBankingException("Account doesn't exists.");
        }

        //to fetch the mini statement provided accountNumber, from Date(dd/mm/yyyy) , toDate(dd/mm/yyyy), and an active session ID.
        @Override
        public List<Transaction> getStatement(int accountNum, Date fromDate, Date toDate,long sessionID) throws RemoteException, InvalidSessionException,
tatementException,RemoteBankingException {
                List<Transaction> statementList = new ArrayList<Transaction>();

                if(!checkSessionActive(sessionID))
                        throw new InvalidSessionException("Invalid Session!!!");
                for (int i=0; i<accounts.size(); i++){
                        Account element = accounts.get(i);
                        if (element.getAccountNum() == accountNum){
                                        return element.getTransactionsByDate(fromDate, toDate);
                        }
                }
```

```
        throw new StatementException("Could not generate statement for given account and date");
    }

    //to stop server, if client uses exit command.
    @Override
    public void stopServer() throws RemoteException
    {
            System.out.println("Stopping server......");
            try
            {
                    registry.unbind("BankInterface");
                    UnicastRemoteObject.unexportObject(registry,true);
                    System.exit(0);

            }
            catch(RemoteException e){}
            catch(Exception e){}
    }

     // to check whether session is active or not.
     private boolean checkSessionActive(long sessID) throws InvalidSessionException{
     for(Session s : sessions){

        //Checks if the sessionID passed from client is in the sessions list and active
        if(s.getClientId() == sessID && s.isAlive()) {
            //Prints session details and returns true if session is alive
            System.out.println(">> Session " + s.getClientId() + " running for " + s.getTimeAlive() + "s");
            System.out.println(">> Time Remaining: " + (s.getMaxSessionLength() - s.getTimeAlive()) + "s");
            return true;
        }

        //If session is in list, but timed out, add it to deadSessions list
        //This flags timed out sessions for removeAll
        //They will be removed next time this method is called
        if(!s.isAlive()) {
            System.out.println("\n>> Cleaning up timed out sessions");
            System.out.println(">> SessionID: " + s.getClientId());
```

```
            System.out.println(">> SessionID: " + s.getClientId());
            deadSessions.add(s);
        }
    }
    System.out.println();

    // cleanup dead sessions by removing them from sessions list
    sessions.removeAll(deadSessions);

    //throw exception if sessions passed to client is not valid
    throw new InvalidSessionException("Invalid Session!!!");
    }


    public static void main(String[] args) throws Exception {
            try{

            @SuppressWarnings("unused")

            // Create an instance of the local object
            Bank bankServer = new Bank();
            //System.out.println("Bank Instance created!!!");

            // Put the server object into the Registry
            Naming.rebind("BankInterface", bankServer);
            System.out.println("-------------------------------------\nBank listening for incoming requests\n-------------------------------------\n");

            // setup some test accounts with various balances.
            Account account1 = new Account(1, "Prince", 11000);
            Account account2 = new Account(2, "Pranav", 15000);
            Account account3 = new Account(3, "Vishesh", 7500);
            Account account4 = new Account(4, "Chandranshu", 20000);
            Account account5 = new Account(5, "Deepak", 32900);
            Account account6 = new Account(6, "Abhishek", 3900);
```

```
            accounts.add(account1);
            accounts.add(account2);
            accounts.add(account3);
            accounts.add(account4);
            accounts.add(account5);
            accounts.add(account6);

            }
            catch (Exception e)
            {
                    System.out.println(e.getMessage());
            }

    }
```

1. *public **synchronized** String startingBalance(int accountNum,String name) throws RemoteException,RemoteBankingException,InvalidArgumentException*

- This module will create an account with the indicated starting balance (which is an integer) along with new session id.
- The AcctNumber argument (an integer) is optional, if it is supplied the account should be created with that value as the account number, if it is not create the account with an account number chosen by the bank.
- Account numbers will be in the range of 1-100. If client supplies any account Number with a value more than 100, then they are informed to enter the correct range of account Number ranging between 1-100. If client supplies any account Number which is already with the bank then, it will throw error to enter correct range of account Number.
- Account will be created only if session id is valid and entered name is in correct format, else it will throw error for invalid session id and InvalidArgument exception for invalid format. This ensures that security within a bank that a particular user with the active session id can only perform transactions along with their unique account Number.
- If successful, it will return the account number of the newly created account.

2. *public **synchronized** double closeAccount(int accountNum) throws RemoteException, , RemoteBankingException,InvalidSessionException*

- This module will close the account with the indicated account Number., only if the session Id is valid and if thread has unlocked lock object.
- This method will return the positive balance, if found and will prevent client from losing their money, in case they tried closing their account without prior knowledge of their balance in the account.
- If client balance is found to be 0.00 then only account will be closed successfully.

3. *public double deposit(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException, InvalidSessionException, RemoteBankingException*

- This module will **lock** the "*balanceChangeLock*" object and **unlocks** "*balanceChangeLock*" object in finally block when deposit is performed successfully, so that other thread can acquire it.
- This module will deposit the indicated amount to the indicated account number, only if session id is valid.
- It will return the balance after updating the amount to the indicated account Number.

4. *public double withdraw(int accountNum, double amount,long sessionID,String fileName,String threadName,int x) throws RemoteException, InvalidSessionException,RemoteBankingException*

- This module will withdraw the indicated amount from the indicated amount, if session is active and valid.
- This module will **lock** the "*balanceChangeLock*" object , **awaits** until amount is less than current balance and **unlocks** "*balanceChangeLock*" object in finally block when withdrawal is performed successfully, so that other thread can acquire it.
- It will return the updated balance if sufficient funds are found and will return the updated balance.
- Otherwise, it will throw insufficient balance exception.

5. *public **synchronized** double inquiry(int accountNum,long sessionID) throws RemoteException, InvalidSessionException,RemoteBankingException*

   - This module will return the current balance for the indicated account number forthe active and valid session id.
   - This module will ensure that while deposit/withdraw is being performed, no other thread should inquire about the balance, else incorrect balance will be displayed.
   - Otherwise, it will throw InvalidAccountException, if someone trying to access someone's else account, their actions will be forbidden.

6. *private boolean checkSessionActive(long sessID) throws InvalidSessionException*

   - This module will create a login session when user enters into Bank application.
   - If session is in list , but timed out then it will add it to deadSessions list
   - They will be removed next time when this method is invoked.
   - It returns true if session is alive, else will print remaining time .

7. public **synchronized** double getTotalBalance() throws RemoteException

   - This module will return total balance of all the accounts maintained by bank.

8. public **synchronized** List<Transaction> getStatement(int accountNum, Date fromDate, Date toDate,long sessionID) throws RemoteException, InvalidSessionException, StatementException,RemoteBankingException

   - This module will generate list of transactions for an existing account.
   - It will throw Statement Exception if provided dates are invalid .

9. public void stopServer() throws RemoteException

   - This module will shut down the server, unbind the registry object and unexport registry object.

5.3 **Client Module**

5.3.1 ATM(Client) Implementation

Now a days clients can access bank through ATM from any location, as it serves all the basic day-to-day financial needs of an individual in a lesser time. We have implemented ATM which will support bank operations so that a Client can avail bank services.

Following are the services offered by bank to client:

1. Open Account
2. Close Account
3. Deposit Amount
4. Withdraw Amount
5. Transfer Amount
6. Get Mini Statement
7. Inquiry
8. Exit

Code screenshots

```java
package client;

import interfaces.*;
import server.*;
import exceptions.*;
import runnable.*;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;
import java.util.Locale;
import java.io.*;

/**
 * ATM class is to represent a client process in a RMI banking application. The client
 * loads a set of account transactions from a account object and makes RMI calls on
 * corresponding remote account objects. The client is multi-threaded.
 */

public class ATM {

	private static BankInterface bankInterface;
	static long sessionID;

	// formatting decimal to decimal places
	private static DecimalFormat precision2 = new DecimalFormat("0.00");

	public static void main(String[] args) throws Exception{

		String option="0";
```

```java
		try{

			// connecting to remote server
			bankInterface = (BankInterface) Naming.lookup("rmi://127.0.0.1/BankInterface");
			System.out.println("\n------------------------------------------------------------\n Client
Connected"+"\n------------------------------------------------------------\n");

		//menu based banking
		while(Integer.parseInt(option)!=8)
		{
			System.out.println("------------------------------------------------------------");
			System.out.println("Please select one of the below operations to proceed further");
			System.out.println("\n1.Open Account  \n2.Deposit Cash  \n3.Withdraw Cash \n4.Transfer Cash \n5.Inquiry Balance \n6.Mini Statement \n7.Close
Account \n8.Exit");
			System.out.println("------------------------------------------------------------");
			Scanner in=new Scanner(System.in);
			option=in.nextLine();
			switch (option) {

			//open account
			case "1":
				try{
					System.out.println("Please provide your registration details. e.g.,Name, Account Number");
					System.out.println("If doesn't want to choose Account Number, please enter 0");
					String name=in.nextLine();

					int acc=in.nextInt();

					String[] account_str=bankInterface.startingBalance(acc,name).split(":");
					int accountNumber=Integer.parseInt(account_str[1]);

					if(accountNumber==-1)
						System.out.println("Please provide a valid account Number (6-100), as already customer exists");
					else if(accountNumber==-2)
						System.out.println("Please provide a valid account Number (6-100)");
					else
```

```
                                {

                                        sessionID=Long.parseLong(account_str[0]);
                                        System.out.println("Welcome to KBK Bank!!!");

                                        //Print account details
                                        System.out.println("-------------------------\nAccount Details:\n-------------------------\n" +
                                       "Account Number: " + accountNumber +
                                       "\nSessionID: " + account_str[0] +
                                       "\nUsername: " + name +
                                       "\nBalance: " +  100.00+
                                       "\n-------------------------\n");
                                        System.out.println("Session active for 5 minutes");
                                        System.out.println("Please use this session id " + sessionID + " for all other bank transactions");
                                }
                        }
                        catch(RemoteException e)
                        {
                                System.out.println(e.getMessage());
                        }
                        catch(InvalidArgumentException e)
                        {
                                System.out.println(e.getMessage());
                        }
                        catch(RemoteBankingException e)
                        {
                                System.out.println(e.getMessage());
                        }
                        break;

                //deposit amount
                case "2":
                        try{
                                System.out.println("Please provide deposit related details. e.g.,Account Number, Amount, SessionID");
                                String accNum=in.nextLine();
                                String amount=in.nextLine();
                                sessionID=Long.parseLong(in.nextLine());
```

```
                                SimpleDateFormat sdf = new SimpleDateFormat("ddMMyyyy");
                                Date curDate = new Date();
                                String strDate = sdf.format(curDate);
                                String fileName = "transactionLogs_" + accNum + "_" +strDate;
                                String path="/home/sneha/Downloads/DistributedBanking_DAIICT/DistributedBanking_DAIICT/src/Logs/"+fileName;

                                File newFile = new File(path);

                                DepositRunnable dt = new DepositRunnable(bankInterface,Integer.parseInt(accNum),Double.parseDouble(amount),sessionID,path,1);
                                Thread d=new Thread(dt);
                                d.setName("Prince.........(Deposit Thread)");
                                d.start();
                                Thread.sleep(2000);

                        }
                        catch(Exception e)
                        {
                                System.out.println(e.getMessage());
                        }
                        break;

                //withdraw amount
                case "3":
                        try
                        {
                                System.out.println("Please provide withdraw related details. e.g.,Account Number, Amount, SessionID");
                                String accN=in.nextLine();
                                String amt=in.nextLine();
                                sessionID=Long.parseLong(in.nextLine());

                                //SimpleDateFormat sdf = new SimpleDateFormat("ddMMyyyy_hhmmss");
                                SimpleDateFormat sdf = new SimpleDateFormat("ddMMyyyy");
                                Date curDate = new Date();
                                String strDate = sdf.format(curDate);
                                String fileName = "transactionLogs_" + accN + "_" +strDate;
                                String path="/home/sneha/Downloads/DistributedBanking_DAIICT/DistributedBanking_DAIICT/src/Logs/"+fileName;
                                File newFile = new File(path);
```

```
                                WithdrawRunnable wt=new WithdrawRunnable(bankInterface,Integer.parseInt(accN),Double.parseDouble(amt),sessionID,path,1);

                                Thread w=new Thread(wt);
                                w.setName("Sneha.........(Withdraw Thread)");
                                w.start();
                                Thread.sleep(2000);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e.getMessage());
                        }
                        break;

                //transfer amount
                case "4":
                                try{

                                        System.out.println("Please provide money transfer related details. e.g.,Account Number, Amount, SessionID");
                                        String acN=in.nextLine();
                                        String amnt=in.nextLine();
                                        sessionID=Long.parseLong(in.nextLine());

                                        SimpleDateFormat sdf = new SimpleDateFormat("ddMMyyyy_hhmmss");
                                        Date curDate = new Date();
                                        String strDate = sdf.format(curDate);
                                        String fileName = "transferlogFile_" + acN + "_" + strDate;
                                        String path="/home/sneha/Downloads/DistributedBanking_DAIICT/DistributedBanking_DAIICT/src/Logs/"+fileName;
                                        File newFile = new File(path);

                                        System.out.println("Total balance before transfer €"+precision2.format(bankInterface.getTotalBalance()));
                                        System.out.println("Transferring..... Please wait!!!");

                                        for(int i=1;i<=100;i++)
                                        {

                                                DepositRunnable dw = new DepositRunnable
(bankInterface,Integer.parseInt(acN),Double.parseDouble(amnt),sessionID,path,10);
```

```
                                                WithdrawRunnable wt=new
WithdrawRunnable(bankInterface,Integer.parseInt(acN),Double.parseDouble(amnt),sessionID,path,10);

                                                Thread one=new Thread(dw);
                                                Thread two=new Thread(wt);
                                                one.setName("Prince.....................................................(Deposit Thread)");
                                                two.setName("Sneha.....................................................(Withdraw Thread)");
                                                one.start();
                                                Thread.sleep(1000);
                                                two.start();
                                        }
                                        Thread.sleep(3000);
                                        System.out.println("Total balance after successful transfer €" +precision2.format(bankInterface.getTotalBalance()));

                                }
                                catch (Exception e)
                                {
                                        System.out.println(e.getMessage());
                                }
                        break;

                //inquiry bank balance
                case "5":
                        System.out.println("Please provide account related details. e.g.,Account Number, SessionID");
                        String AccountNum=in.nextLine();
                        sessionID=Long.parseLong(in.nextLine());
                        try{
                                double resultInquiry = bankInterface.inquiry(Integer.parseInt(AccountNum),sessionID);
                                System.out.println("Current balance: €" + precision2.format(resultInquiry));
                        }
                        //Catch exceptions that can be thrown from the server
                        catch (InvalidSessionException e)
                        {
                                System.out.println(e.getMessage());
                        }
```

```
                        catch (Exception e)
                        {
                                System.out.println(e.getMessage());
                        }
                        break;

                //get statement
                case "6":
                        try{
                                System.out.println("Please provide account related details to fetch statement. e.g.,Account Number, From Date(dd/mm/yyyy),
ToDate(dd/mm/yyyy), SessionID");
                                String actNumber=in.nextLine();
                                String fromD=in.nextLine();
                                String toD=in.nextLine();
                                sessionID=Long.parseLong(in.nextLine());

                                Date fromDate = new SimpleDateFormat("dd/MM/yy", Locale.ENGLISH).parse(fromD);
                                Date toDate = new SimpleDateFormat("dd/MM/yy", Locale.ENGLISH).parse(toD);

                                SimpleDateFormat sdf = new SimpleDateFormat("ddMMyyyy");
                                Date curDate = new Date();
                                String strDate = sdf.format(curDate);
                                String fileName = "transactionLogs_" + actNumber + "_" +strDate;
                                String path="/home/sneha/Downloads/DistributedBanking_DAIICT/DistributedBanking_DAIICT/src/Logs/"+fileName;

                                java.util.List<Transaction> statementList =
                                                bankInterface.getStatement(Integer.parseInt(actNumber), fromDate, toDate,sessionID);
                                String message="---------------------------------\nMINI STATEMENT\n---------------------------------\n";

                                FileWriter writer = new FileWriter(path, true);
                                writer.write(message);

                                for (int i=0; i<statementList.size(); i++) {
                                        Transaction element = statementList.get(i);
                                        writer.write(element.toString());
                                        writer.write("\n");
                                }
```

```
                                writer.close();
                                System.out.println("Balance Statement generated!!!");
                        }
                        catch(RemoteException e)
                        {
                                System.out.println(e.getMessage());
                        }
                        catch(InvalidSessionException e)
                        {
                                System.out.println(e.getMessage());
                        }
                        catch(StatementException e)
                        {
                                System.out.println(e.getMessage());
                        }
                        break;

                //close account
                case "7":
                        System.out.println("Please provide account related details to close account. e.g.,Account Number, SessionID");
                        String aN=in.nextLine();
                        sessionID=Long.parseLong(in.nextLine());
                        try{
                                double closed=bankInterface.closeAccount(Integer.parseInt(aN),sessionID);
                                if(closed>0){
                                        System.out.println("Error: Account can't be closed as positive Balance found amounts to €"+closed);
                                }
                                else
                                        System.out.println("ok. Account closed successfully!!!");
                                }
                        catch(RemoteBankingException e)
                        {
                                System.out.println(e.getMessage());
                        }
                        catch(InvalidSessionException e)
                        {
                                System.out.println(e.getMessage());
```

```
                                System.out.println(e.getMessage());
                        }
                        catch(Exception e)
                        {
                                System.out.println(e.getMessage());
                        }
                        break;

                //to start the menu
                case "0":
                        break;
                }
            }
            if(Integer.parseInt(option)==8){

                        bankInterface.stopServer();
            }
            System.exit(0);
        }
        catch(Exception e)
        {
                if(Integer.parseInt(option)==8)
                        System.out.println("Thanking you for visiting us!!!");
                else
                        System.out.println("Server shut down. Can't connect.");
        }

    }
```

The ATM is menu-based interface where user have to select any of the below operations:

1) Open :
   - It will read two parameters from the console namely: name of the user and an account number (which is optional).
   - It then converts all console-read inputs to their respective data type format.
   - It calls startingBalance method of Bank to open a new account for a user.
   - If the received response is a valid account number it will successfully shows the message to user that he has joined Bank with all necessary account details, otherwise will throw error to provide the valid account number.
   - If other thread tries to open the same account number again, it will throw error.

2) Close:
   - It will read two parameters from console namely: account number of an existing user and session Id.
   - It invokes closeAccount method of Bank to close the account if it doesn't have any positive balance.
   - If the received response is greater than zero then account close request will be declined, otherwise closed successfully with a message to client.

3) Deposit
   - It reads three parameters from console namely: an account number of an existing user, amount to deposit and an active session id.
   - It creates new instance of "***DepositRunnable***" and passes bankInterface object along with accountNumber, amount to be deposited, session ID, filename to log the deposit entries and number of times we want to perform deposit operation.
   - It starts the deposit thread every time user tries to deposit amount.
   - It will display current balance after deposit has been made successfully.

4) Withdraw
   - It reads three parameters from console namely: account number of an existing account holder, amount to be withdrawn and an active session id.
   - It invokes "***WithdrawRunnable***" and passes bankInterface object along with accountNumber, amount to be withdrawal, session ID, filename to log the withdraw entries and number of times we want to perform withdraw operation.
   - It throws InsufficientFunds error in case above case fails, else displays the updated balance to client.

5) Inquiry
    - It reads two parameters from console namely: account number of an existing user and an active session id.
    - It invokes inquiry method of bank to fetch the current balance if the indicated account number is valid, otherwise will throw error (“Account doesn’t exist”).

6) Statement
    - It reads four parameters from console namely: account number of an existing user, from date , to date and an active session id.
    - It invokes getStatement method of bank which will generate list of transactions if the provided date and indicated account number is valid.
    - Otherwise, it will throw statement error.

7) Transfer
    - This module is implemented to show how **deposit** and **withdraw thread** operates when they are called on **same** account concurrently.
    - It invokes “*DepositRunnable” and “WithdrawRunnable” thread.* Both thread are renamed as “Prince(Deposit Thread)” and “Sneha(Withdraw Thread)”.
    - The entire transfer log is saved in “transferLog_accNum_curDate” file.

8) Exit
    - It will shut down the server and will end the menu-based interface.

5.4 **Runnable Module**

In a multithreaded application, several threads can access the same data concurrently which may leave the data in inconsistent state (corrupted or inaccurate). It is well known that in distributed banking system, different users can access the same account simultaneously due to which the balance of an account can be changed frequently due to the transaction of deposit and withdrawal.

In this module implementation, we have ensured that no matter how many transactions are processed, the balance of the account should be reflected correctly, whenever any user tries to access his/her account. The shared data is protected which might get corrupted due to concurrent updates by multiple threads. The Java Concurrency API provides a synchronization mechanism that involves in locking/unlocking on a lock object. We have implemented two threads namely **Deposit Thread** and **Withdraw thread**. The mechanism works as follows:

- When a thread enters deposit/withdraw method of “*Bank*” class, it attempts to acquire the lock object and if the lock is not held by another thread, the concurrent threads gets exclusive ownership on the lock object.
- If the lock is currently held by another thread, then the current thread blocks and waits until the lock is released.
- Once the current thread successfully acquires the lock, it executes the code in the try block without worrying about intervention of other threads.
- Finally, the thread releases the lock and exists the deposit/withdraw method of “*Bank*” class.
- After that, the chance is given to other threads to acquire the lock. At any time, only one thread owns the lock and can execute the protected code.

To achieve this synchronization we have used “**ReentrantLock**” because it allows thread to acquire a lock , it already owns multiple times recursively which is implemented by two runnable’s as specified below:

5.4.1 **DepositRunnable Implementation**: *public class DepositRunnable implements Runnable*

1) *public DepositRunnable(BankInterface bankInterface,int accountNum,double amount,long sessionID,String fileName,int repetitions)*

- This constructor accepts the “bankInterface” object which is passed from client ATM class. This object ensures server methods accessibility.
- The other parameters are account Number of an account holder, amount to be deposited, an active session ID(which acts as entry token for any transaction), file Name in which all the deposit related transactions are written for future reference, and repetitions which ensures how many times deposit needs to be performed.
- This repetition variable is needed because we have one operation named “transfer” where we have ran a “deposit-withdraw” process numerous times on same account to show that our application handles concurrency efficiently.

2) *public void run()*

- This method is invoked when Deposit Thread is started from client (ATM class).
- It fetches the current thread name.
- It prepares a string “message” to entry the deposit logs in file for every account.
- ***bankInterface.deposit(accountNum,amount,sessionID,fileName,threadName,i); ,*** *this method is invoked from inside the Deposit thread to perform the deposit operation. The deposit method is already discussed under “Bank” module*

Code screenshot

```java
/*
* Written by: @Sneha and @Prince
* When user tries to deposit an amount
* this runnable is invoked from ATM class
**/

package runnable;

import interfaces.*;
import server.*;
import exceptions.*;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.io.*;
import java.rmi.*;

public class DepositRunnable implements Runnable
{
   /*
        private data members of thread
   */
   private int accountNum;
   private long sessionID;
   private double amount;
   private BankInterface bankInterface;
   private int repetitions;
   private String fileName;
   private static DecimalFormat precision2 = new DecimalFormat("#.##");
```

```java
/*
*    constructor to initialize accountNumber,
*    amount to be deposited,
*    an active sessionID,
*    fileName to log deposit entries
*    and number of times this thread to be called.
*/
public DepositRunnable(BankInterface bankInterface,int accountNum,double amount,long sessionID,String fileName,int repetitions)
{
     this.bankInterface=bankInterface;
     this.accountNum=accountNum;
     this.amount = amount;
     this.sessionID=sessionID;
     this.fileName=fileName;
     this.repetitions=repetitions;
}
@Override
/*
*    run method to invoke deposit method of Bank class using client object connected to server
*    using Java RMI.
*/
public void run()
{
   try
   {
            String message="---------------------------------\nDEPOSIT\n---------------------------------\n";
         for(int i=1;i<=repetitions;i++){
                String threadName = Thread.currentThread().getName();
                message+=threadName+" "+String.valueOf(i)+" is trying to deposit .... €"+String.valueOf(amount)+"\n";

                FileWriter writer = new FileWriter(fileName, true);
                writer.write(message);
                writer.close();

                double resultDeposit=bankInterface.deposit(accountNum,amount,sessionID,fileName,threadName,i);
```

```java
                if(repetitions==1)
                    System.out.println("Current balance after deposit: €" + precision2.format(resultDeposit));
                Thread.sleep(100);
        }
    }
    catch(InvalidSessionException e){System.out.println(e.getMessage());}
    catch (InterruptedException exception) {}
    catch(RemoteBankingException e){System.out.println(e.getMessage());}
    catch (RemoteException e){}
    catch (Exception e){}
 }
```

5.4.2 **WithdrawRunnable Implementation**: *public class WithdrawRunnable implements Runnable*

1. *public WithdrawRunnable(BankInterface bankInterface,int accountNum,double amount,long sessionID,String fileName,int repetitions))*

- This constructor accepts the “bankInterface” object which is passed from client ATM class. This object ensures server methods accessibility.
- The other parameters are account Number of an account holder, amount to be withdrawn, an active session ID(which acts as entry token for any transaction), file Name in which all the withdraw related transactions are written for future reference, and repetitions which ensures how many times withdraw needs to be performed.
- This repetition variable is needed because we have one operation named “transfer” where we have ran a “deposit-withdraw” process numerous times on same account to show that our application handles concurrency efficiently. This is already discussed in Client module.

2. *public void run()*

- This method is invoked when Withdraw Thread is started from client (ATM class).
- It fetches the current thread name.
- It prepares a string “message” to entry the withdraw logs in file for every account.
- ***bankInterface.withdraw(accountNum,amount,sessionID,fileName,threadName,j); ,*** *this method is invoked from inside the Withdraw thread to perform the withdraw operation. The withdraw method is already discussed under “Bank” module.*

Code screenshots

```
~/Downloads/DistributedBanking_DAIICT/DistributedBanking_DAIICT/src/runnable
/*
* Written by: @Sneha and @Prince
* When user tries to deposit an amount
* this runnable is invoked from ATM class
**/

package runnable;

import interfaces.*;
import server.*;
import exceptions.*;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.io.*;
import java.rmi.*;
public class WithdrawRunnable implements Runnable
{

   /*
        private data members of thread
   */
   private int accountNum;
   private long sessionID;
   private double amount;
   private BankInterface bankInterface;
   private String fileName;
   private int repetitions;
   private static DecimalFormat precision2 = new DecimalFormat("#.##");
```

```
 /*
 *    constructor to initialize accountNumber,
 *    amount to be withdrawn,
 *    an active sessionID,
 *    fileName to log deposit entries
 *    and number of times this thread to be called.
 */
 public WithdrawRunnable(BankInterface bankInterface,int accountNum,double amount,long sessionID,String fileName,int repetitions)
 {
      this.bankInterface=bankInterface;
      this.accountNum=accountNum;
      this.amount = amount;
      this.sessionID=sessionID;
      this.fileName=fileName;
      this.repetitions=repetitions;
 }

 @Override
 /*
 *    run method to invoke withdraw method of Bank class using client object connected to server
 *    using Java RMI.
 */
 public void run()
 {
    try
    {
                String message="---------------------------------\nWITHDRAW\n---------------------------------\n";
                for(int j=1;j<=repetitions;j++){
                        String threadName = Thread.currentThread().getName();
                          message+=threadName+" " +String.valueOf(j)+" is trying to withdraw..... €"+String.valueOf(amount)+"\n";

                          FileWriter writer = new FileWriter(fileName, true);
                          writer.write(message);
                          writer.close();
                          double resultWithdrawal=bankInterface.withdraw(accountNum,amount,sessionID,fileName,threadName,j);
                          if(repetitions==1)
                              System.out.println("Current balance after withdrawal: €" + precision2.format(resultWithdrawal));
```

```
                          if(repetitions==1)
                              System.out.println("Current balance after withdrawal: €" + precision2.format(resultWithdrawal));
                          if(resultWithdrawal==0.00)
                              throw new InsufficientFundsException();
                          Thread.sleep(200);
             }

    }
    catch (InterruptedException e) {System.out.println(e.getMessage());}
    catch (RemoteException e){}
    catch (RemoteBankingException e){System.out.println(e.getMessage());}
    catch (InvalidSessionException  e){System.out.println(e.getMessage());}
    catch (InsufficientFundsException e){System.out.println(e.getMessage());}
    catch(Exception e){}
 }
```

5.5 **Exception Handling Module**

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time, that disrupts the normal flow of the program, while an error indicates serious problem that a reasonable application doesn't find suitable to catch.

5.5.1 InvalidSessionException: *public InvalidSessionException()*

- It takes no parameter and displays "your session has timed out after 5 minutes of activity" if session expires.

Code screenshot

```
package exceptions;


/*
Written by: @Sneha and @Prince
when user tries to perform bank
operations using invalid session
which is dead now or belongs to
other ongoing transaction.
**/

public class InvalidSessionException extends Exception {
    public InvalidSessionException(String msg) {
        super(msg);
    }
}
```

5.5.2 RemoteBanking Exception: *public RemoteBankingException(String msg)*

- It takes message as parameter and displays "account doesn't exist" message if client tries to excess invalid account data.

Code screenshot

```
package exceptions;


/*
Written by: @Sneha and @Prince
when user tries to access account
which is not registered with Bank.
**/
public class RemoteBankingException extends Exception {
    public RemoteBankingException(String msg){
        super(msg);
    }
}
```

5.5.3 InsufficientFunds Exception: *public InsufficientFundsException()*

- It takes no parameter and displays "insufficient funds" if client tries to withdraw amount less than the current balance.

Code screenshot

```java
package exceptions;

/*
Written by: @Sneha and @Prince
when balance is 0.00 and
user is try to withdraw amount
**/

public class InsufficientFundsException extends Exception{
    public InsufficientFundsException(){
        super("Insufficient Funds");
    }
}
```

5.5.4 Statement Exception: public StatementException(String msg)

- It displays “Could not generate statement for given account and date” message if the provided date and account number is invalid.

Code screenshot

```java
package exceptions;


/*
Written by: @Sneha and @Prince
when date provided by user is
invalid for fetching mini
statement.
**/

public class StatementException extends Exception {
    public StatementException(String msg){
        super(msg);
    }
}
```

5.5.5 InvalidArgument Exception: public InvalidArgumentException(String msg)

- It displays “Err. Invalid name. Failed attempt to open account.” message if the provided name is in invalid format.

Code screenshots

```java
package exceptions;


/*
Written by: @Sneha and @Prince
when user tries to provide name in
invalid format.
**/
public class InvalidArgumentException extends Exception {
    public InvalidArgumentException(String msg){
        super(msg);
    }
}
```

6. **Project Structure and expected Output**

The following is folder structure using following command:

@kali:~$ tree /home/Downloads/DistributedBanking_DAIICT

```
/home/Downloads/DistributedBanking_DAIICT
└── DistributedBanking_DAIICT
    ├── src
    │   ├── client
    │   │   ├── ATM.class
    │   │   └── ATM.java
    │   ├── exceptions
    │   │   ├── InsufficientFundsException.class
    │   │   ├── InsufficientFundsException.java
    │   │   ├── InvalidAccountException.class
    │   │   ├── InvalidAccountException.java
    │   │   ├── InvalidArgumentException.class
    │   │   ├── InvalidArgumentException.java
    │   │   ├── InvalidSessionException.class
    │   │   ├── InvalidSessionException.java
    │   │   ├── RemoteBankingException.class
    │   │   ├── RemoteBankingException.java
    │   │   ├── StatementException.class
    │   │   └── StatementException.java
    │   ├── interfaces
    │   │   ├── AccountInterface.class
    │   │   ├── AccountInterface.java
    │   │   ├── BankInterface.class
    │   │   └── BankInterface.java
    │   ├── Logs
    │   │   ├── transactionLogs_7_28042020
    │   │   ├── transactionLogs_8_28042020
    │   │   └── transferlogFile_7_28042020_041859
    │   ├── runnable
    │   │   ├── DepositRunnable.class
    │   │   ├── DepositRunnable.java
    │   │   ├── WithdrawRunnable.class
    │   │   └── WithdrawRunnable.java
    │   ├── screenshots
    │   │   ├── Screenshot from 2020-04-28 16-21-02.png
    │   │   ├── Screenshot from 2020-04-28 16-21-04.png
    │   │   ├── Screenshot from 2020-04-28 16-28-40.png
    │   │   ├── Screenshot from 2020-04-28 16-29-34.png
    │   │   ├── Screenshot from 2020-04-28 16-29-36.png
    │   │   ├── Screenshot from 2020-04-28 16-29-39.png
    │   │   ├── Screenshot from 2020-04-28 16-29-41.png
    │   │   ├── Screenshot from 2020-04-28 16-29-44.png
    │   │   ├── Screenshot from 2020-04-28 16-29-46.png
    │   │   ├── Screenshot from 2020-04-28 16-29-47.png
    │   │   ├── Screenshot from 2020-04-28 16-29-50.png
    │   │   ├── Screenshot from 2020-04-28 16-29-52.png
    │   │   ├── Screenshot from 2020-04-28 16-29-55.png
    │   │   ├── Screenshot from 2020-04-28 16-29-57.png
    │   │   ├── Screenshot from 2020-04-28 16-29-59.png
    │   │   ├── Screenshot from 2020-04-28 16-30-01.png
    │   │   ├── Screenshot from 2020-04-28 16-30-03.png
    │   │   ├── Screenshot from 2020-04-28 16-30-05.png
    │   │   ├── Screenshot from 2020-04-28 16-30-19.png
```

```
    ├── Screenshot from 2020-04-28 16-30-26.png
    ├── Screenshot from 2020-04-28 16-30-32.png
    ├── Screenshot from 2020-04-28 16-30-40.png
    ├── Screenshot from 2020-04-28 16-30-45.png
    ├── Screenshot from 2020-04-28 16-30-59.png
    ├── Screenshot from 2020-04-28 16-31-05.png
    ├── Screenshot from 2020-04-28 16-31-13.png
    ├── Screenshot from 2020-04-28 16-31-19.png
    ├── Screenshot from 2020-04-28 16-31-24.png
    ├── Screenshot from 2020-04-28 16-31-35.png
    ├── Screenshot from 2020-04-28 16-31-40.png
    ├── Screenshot from 2020-04-28 16-32-27.png
    ├── Screenshot from 2020-04-28 16-32-34.png
    └── Screenshot from 2020-04-28 16-32-42.png
  └── server
    ├── Account.class
    ├── Account.java
    ├── Bank.class
    ├── Bank.java
    ├── Session.class
    ├── Session.java
    ├── Transaction.class
    └── Transaction.java
└── test.POLICY
```

9 directories, 67 files.

(Note*: Screenshots have been added in screenshot folder)

- For every account, log files are generated which is saved in "*Logs*" folder.
- For "deposit", "withdraw" and "get statement" operations, entry is logged into file in following format: transactionLogs_accountNumber_currentDate(ddmmyyyy).
- In order to show synchronization between threads, we have implemented transfer operation (as discussed earlier in "ATM" module). The results are stored in a file in following format: transferlogFile_accountNumber_currentDate(ddmmyyyy_hhmmss).

The output supports the objective of this project. Screenshots has been attached in next section.

7. **Screenshot of application**

Client Screenshot

```
$ java client/ATM
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

------------------------------------------------------------
 Client Connected
------------------------------------------------------------

------------------------------------------------------------
Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
```

```
7.Close Account
8.Exit
------------------------------------------------------------
1
Please provide your registration details. e.g.,Name, Account Number
If doesn't want to choose Account Number, please enter 0
100
100
Error:Invalid name. Failed attempt to open account.
------------------------------------------------------------
Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit
------------------------------------------------------------
1
Please provide your registration details. e.g.,Name, Account Number
If doesn't want to choose Account Number, please enter 0
Nidhi
7
Welcome to KBK Bank!!!
-------------------------
Account Details:
-------------------------
Account Number: 7
SessionID: 280478
Username: Nidhi
Balance: 100.0
-------------------------

Session active for 5 minutes
Please use this session id 280478 for all other bank transactions
------------------------------------------------------------
Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit
------------------------------------------------------------
1
Please provide your registration details. e.g.,Name, Account Number
If doesn't want to choose Account Number, please enter 0
Asha
5
Please provide a valid account Number (6-100), as already customer exists
------------------------------------------------------------
Please select one of the below operations to proceed further
```

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

1
Please provide your registration details. e.g.,Name, Account Number
If doesn't want to choose Account Number, please enter 0
Asha
0
Welcome to KBK Bank!!!

Account Details:

Account Number: 93
SessionID: 626929
Username: Asha
Balance: 100.0

Session active for 5 minutes
Please use this session id 626929 for all other bank transactions
---
Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

2
Please provide deposit related details. e.g.,Account Number, Amount, SessionID
7
1000
280478
Current balance after deposit: €1100

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

2
Please provide deposit related details. e.g.,Account Number, Amount, SessionID
7

```
1000
280472
Invalid Session!!!
---------------------------------------------------------------
Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit
---------------------------------------------------------------
2
Please provide deposit related details. e.g.,Account Number, Amount, SessionID
8
1000
280478
Account doesn't exists!!!
---------------------------------------------------------------
Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit
---------------------------------------------------------------
3
Please provide withdraw related details. e.g.,Account Number, Amount, SessionID
7
1000
280478
Current balance after withdrawal: €100
---------------------------------------------------------------
Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit
---------------------------------------------------------------
3
Please provide withdraw related details. e.g.,Account Number, Amount, SessionID
8
1000
280478
Account doesn't exists.
---------------------------------------------------------------
Please select one of the below operations to proceed further
```

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

3
Please provide withdraw related details. e.g.,Account Number, Amount, SessionID
7
100
280472
Invalid Session!!!

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

3
Please provide withdraw related details. e.g.,Account Number, Amount, SessionID
7
100
280478
Current balance after withdrawal: €0
Insufficient Funds

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

3
Please provide withdraw related details. e.g.,Account Number, Amount, SessionID
7
100
280478

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash

5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

5
Please provide account related details. e.g.,Account Number, SessionID
7
280478
Current balance: €0.00

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

6
Please provide account related details to fetch statement. e.g.,Account Number, From Date(dd/mm/yyyy), ToDate(dd/mm/yyyy), SessionID
7
27/04/2020
29/04/2020
280478
Balance Statement generated!!!

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

4
Please provide money transfer related details. e.g.,Account Number, Amount, SessionID
7
100
280478
Total balance before transfer €90400.00
Transferring..... Please wait!!!
Total balance after successful transfer €90400.00

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement

7.Close Account
8.Exit

5
Please provide account related details. e.g.,Account Number, SessionID
7
280478
Current balance: €0.00

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

7
Please provide account related details to close account. e.g.,Account Number, SessionID
7
280478
ok. Account closed successfully!!!

Please select one of the below operations to proceed further

1.Open Account
2.Deposit Cash
3.Withdraw Cash
4.Transfer Cash
5.Inquiry Balance
6.Mini Statement
7.Close Account
8.Exit

8
Thanking you for visiting us!!!
@kali:~/Downloads/DistributedBanking_DAIICT/DistributedBanking_DAIICT/src$

Server Screenshot

$ java server/Bank
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
---
Bank listening for incoming requests

>> Session 280478 created

>> Session 626929 created

>> Session 280478 running for 19s
>> Time Remaining: 281s
>> Session 280478 running for 19s
>> Time Remaining: 281s

>> Session 280478 running for 36s
>> Time Remaining: 264s
>> Session 280478 running for 44s
>> Time Remaining: 256s
>> Session 280478 running for 44s
>> Time Remaining: 256s
>> Session 280478 running for 53s
>> Time Remaining: 247s

>> Session 280478 running for 69s
>> Time Remaining: 231s
>> Session 280478 running for 69s
>> Time Remaining: 231s
>> Session 280478 running for 75s
>> Time Remaining: 225s
>> Session 280478 running for 81s
>> Time Remaining: 219s
>> Session 280478 running for 91s
>> Time Remaining: 209s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 102s
>> Time Remaining: 198s
>> Session 280478 running for 103s
>> Time Remaining: 197s
>> Session 280478 running for 103s
>> Time Remaining: 197s
>> Session 280478 running for 103s
>> Time Remaining: 197s
>> Session 280478 running for 103s
>> Time Remaining: 197s
>> Session 280478 running for 103s
>> Time Remaining: 197s
>> Session 280478 running for 103s
>> Time Remaining: 197s
>> Session 280478 running for 103s
>> Time Remaining: 197s
>> Session 280478 running for 103s
..
..
..
..
..

>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s

>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 200s
>> Time Remaining: 100s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s

>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 201s
>> Time Remaining: 99s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s

```
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 202s
>> Time Remaining: 98s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
```

>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 203s
>> Time Remaining: 97s
>> Session 280478 running for 204s
>> Time Remaining: 96s
>> Session 280478 running for 204s
>> Time Remaining: 96s
>> Session 280478 running for 204s
>> Time Remaining: 96s
>> Session 280478 running for 204s
>> Time Remaining: 96s
>> Session 280478 running for 204s
>> Time Remaining: 96s
>> Session 280478 running for 204s
>> Time Remaining: 96s
>> Session 280478 running for 214s
>> Time Remaining: 86s
>> Session 280478 running for 221s
>> Time Remaining: 79s
Stopping server.....
@kali:~/Downloads/DistributedBanking_DAIICT/DistributedBanking_DAIICT/src$

(Note* *Screenshots has been added in screenshots folder*)

8. Problems/Issues with RMI (Remote Method Invocation)

8.1 Design Issues with RMI

1) RMI invocation semantics
   - Invocation semantics depend upon implementation of Request-Reply protocol used by RMI.
   - Could be Maybe, At-least-once, At-most-once.

2) Transparency
   - Remote invocations are not fully transparent to programmer as there are likely chances of partial failure and higher latency.

- Access transparency : remote invocations should be made transparent in the way so that syntax of remote invocation is same as the syntax of local invocation, but programmer should be able to distinguish between remote and local objects by looking at their interfaces.
- Java RMI, remote objects implement the remote interface.

8.2 Implementation Issues with RMI

1) Parameter Passing
   - Representing a remote object reference takes 32 bits for IP, 32 bits for Port, 32 bits for time, 32 bits of object number, hence space issues.

2) Request/Reply Protocol
   - Handling failure at client and/or server is difficult.
   - Issues in marshalling of input, ouput parameters and results.
   - Handling failures in request-reply protocol.

3) Supporting persistent objects, object adapters, dynamic invocations etc.
   - Remote object references are passed by reference where local object references are passed by value.

9. Limitations

1) As of now our server is capable of storing user transaction details in form of list. It doesn't have any database storage to refer transactions in future, it is q 2-tier model.
2) Our application supports only few operations: open account, close account, deposit, withdraw, inquiry, transfer and fetching mini statement along with the support of concurrency.

10. Future Work

1) We can implement 3-tier model consisting of an application (implementing GUI), a layer for business logic and a database.
2) We can add many more functionalities like migration of account, once we have database, then linking of aadhaar with the accounts to make them more secure and less vulnerable.

Some of the above features can be added so that client can get real-time banking experience.

11. Roles and Responsibility

Student Name	Modules Implemented
Sneha Shukla	Bank(Server), BankInterface, DepositRunnable Thread, WithdrawRunnable Thread, Log file generation for each transfer/transaction, Transaction Implementation.
Prince Mishra	ATM(Client), Session Implementation, Exceptions Handling across modules, Account Implementation, AccountInterface.

12. **Project Github Link**

https://github.com/sgeek28/Distributed-Systems

13. **References**

[1] DISTRIBUTED SYSTEMS: Principles and paradigms by Andrew S. Tanenbaum and Maarten Van Steen

[2] Distributed Online Banking by Mahmood Akhtar and Kamyar Dezhgosha

[3] Maassen, Jason, et al. “Efficient RMI for parallel programming.” ACM Transactions on Programming Languages and Systems 23.6 (2001): 747-775.

[4] Buchmann, Alejandro P., et al. “A Transaction model for active distributed object systems.” (1992):123-158.

[5] Raman, Arun, et al. "Speculative parallelization using software multi-threaded transactions." ACM SIGARCH computer architecture news. Vol. 38. No. 1. ACM, 2010.

[6] Mohammad Tabrez Quasim. “An Efficient Approach For Concurrency Control in distributed Database System.” (2013)

[7] Kung, Hsiang-Tsung, and John T.Robinson, “On optimistic methods for concurrency control.” ACM Transactions on Database Systems (TODS) 6.2 (1981): 213-226.