**LAB MANUAL**

**Subject :** Distributed Computing

**Subject Code :**  CS401

**Submission Date:** 20.9.2016 , 5:00 pm

**Instructions:** Submit your code along with screenshots of the outputs in a consolidated document. Points for each assignment is mentioned beside the question. Points for a question will be awarded only if you have correctly worked out the previous questions. No points will be awarded if you indulge in any form of plagiarism.

**ASSIGNMENT : 1**

]

A distributed banking system consists of a server and some Automated Teller Machines

(ATM). The server manages all users’ account information. A customer can invoke the

following operations at an ATM.

1. void deposit(int acnt, int amt): this operation increases the balance of user account acnt by

amt, and returns nothing;

2. void withdraw(int acnt, int amt): this operation decreases the balance of user account acnt

by amt, and returns nothing;

3. int inquiry(int acnt): this operation returns the balance of user account acnt

For simplicity, in this assignment you do not need to consider the synchronization

problem.

Programming Requirement

You are required to write this client-server (banking system) program which communicates via RPC. You are recommended to develop this system using JAVA/RMI (the functionality comes in the package java.rmi).

Specifically, your program should consist of two parts: one for the client and another for the server. The client (as the ATM) will initiate RPC by sending request message to the bank server to execute a specified procedure (e.g. deposit) with parameters. Recall the sequence of events during a RPC:

1. Client procedure calls client stub in normal way

2. Client stub builds message, calls local OS

3. Client's OS sends message to remote OS

4. Remote OS gives message to server stub

5. Server stub unpacks parameters, calls server

6. Server does work, returns result to the stub

7. Server stub packs it in message, calls local OS

8. Server's OS sends message to client's OS

9. Client's OS gives message to client stub

10. Client stub unpacks result, returns to client

Design Requirement

You can use the following interface and class definitions as a starting point for your

project. However, you are free to develop your own interface and class definitions.

**1.Interface**

public interface BankInterface extends Remote

{

public void deposit(int account, int amount)

throws RemoteException;

public void withdraw(int account, int amount)

throws RemoteException;

public int inquiry(int account)

throws RemoteException;

}

**2. Server Object**

public class Bank extends unicastRemoteObject implements BankInterface

{

private Vector userAccount; //users’ account

private Vector userBalance; //users’ balance

public Bank() throws RemoteException

{

}

public void deposit(int account, int amount) throws RemoteException

{

//implementations

}

public void withdraw(int account, int amount) throws RemoteException

{

//implementations

}

public int inquiry(int account) throws RemoteException

{

//implementations

}

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

{

//init Bank server

}

}

Initially, the Bank server should have the following data in its database (userAccount,

userBalance in the above example code)

Account Balance

100 $1000

The bank server has only one input parameter “server\_port”, which specifies the port of

rmiregistry. The default port of rmiregistry is 1099, but we may have to use other ports,

if 1099 has already been used by some other programs.

3. ATM Object

public class ATM

{

public static void (String args[]) throws Exception

{

//get user’s input, and perform the operations

}

}

The input parameters of an ATM must include:

* server\_address: the address of the rmiregistry
* server\_port: the port of the rmiregistry
* operation: one of “deposit”, “withdraw”, and “inquiry”
* account: the user account
* amount: only for “deposit” and “withdraw” operations

[20]