Banking System

(Experiment 2 - Clock Synchronization)

Yash Brid 2019130008

Abhishek Chopra 2019130009

Sumeet Halipur 2019130018

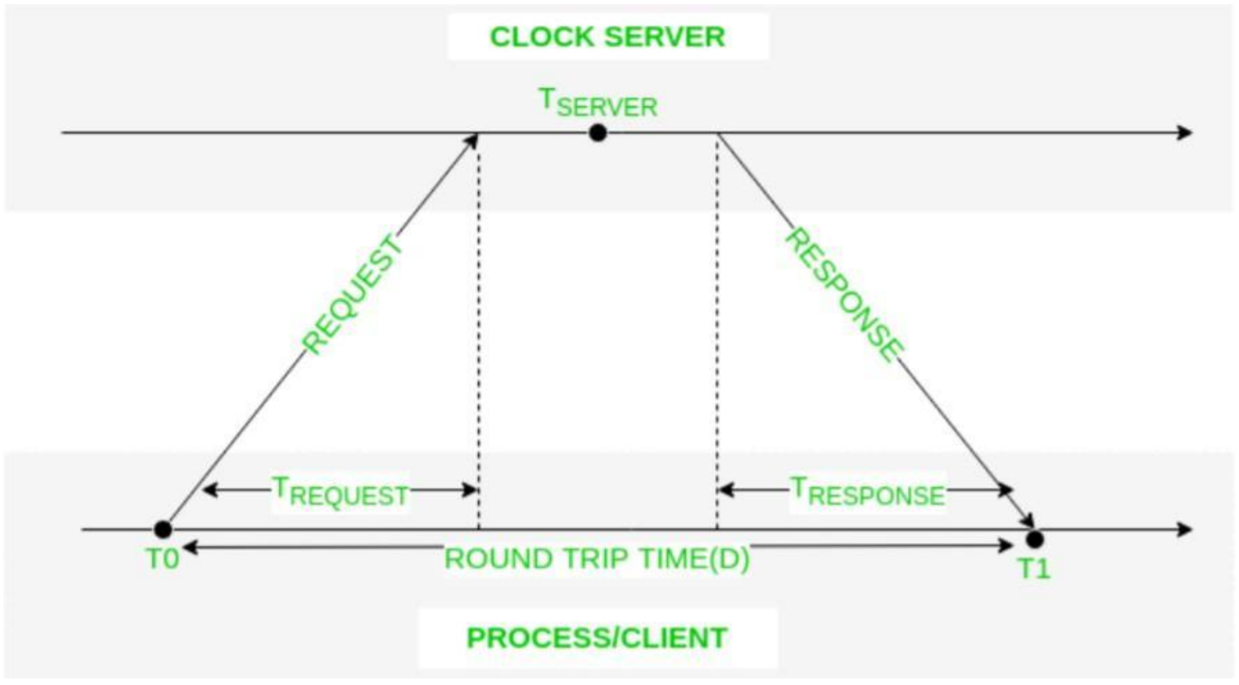
Aim: To implement Clock Synchronization for a Banking System.

Objective:

- To learn about Clock Synchronization in a distributed system.
- To implement Clock Synchronization for a Banking System.

Theory:

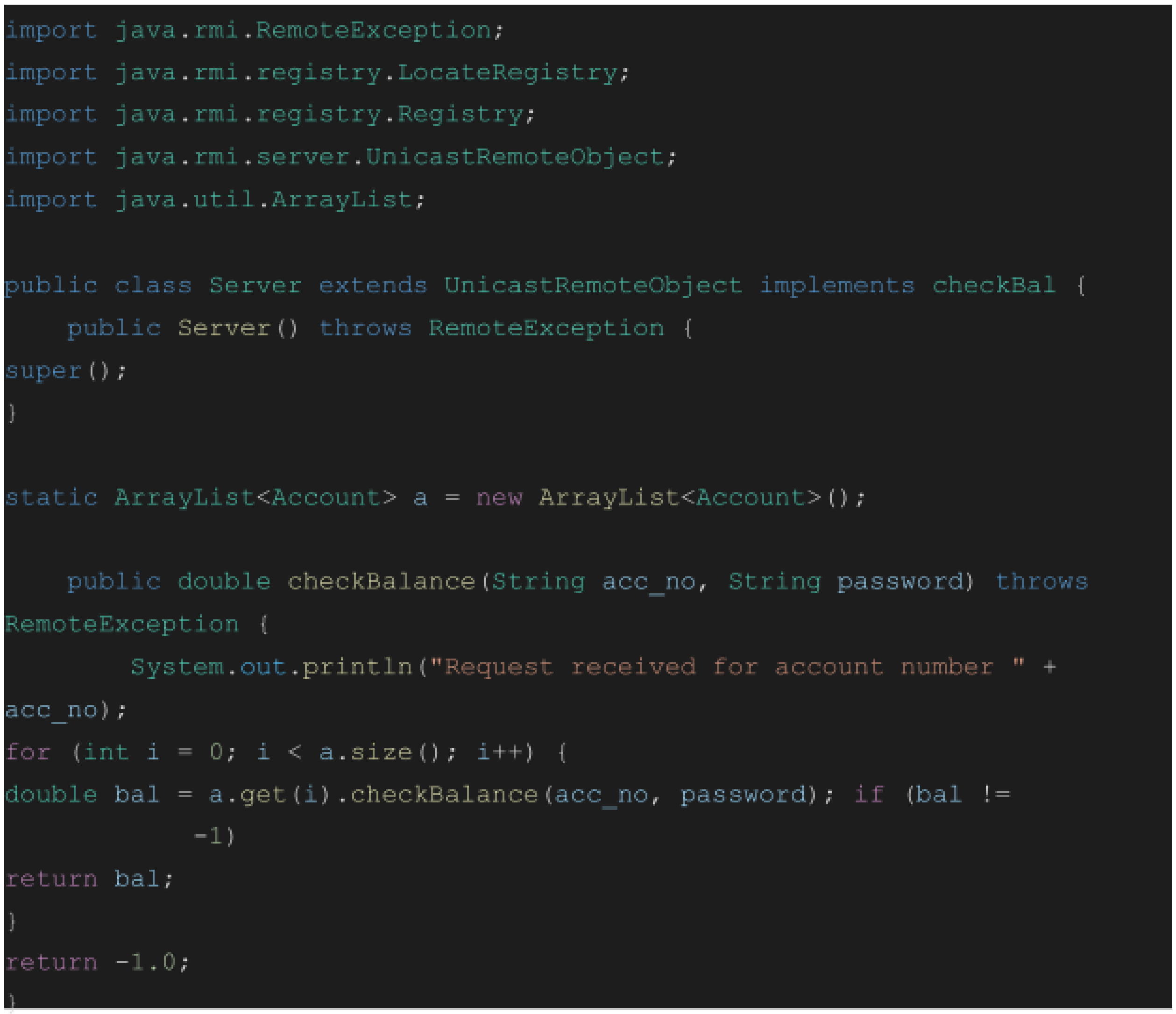
Cristian's Algorithm is a clock synchronization algorithm that is used to synchronize time with a time server by client processes. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy while redundancy-prone distributed systems/applications do not go hand in hand with this algorithm. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.



Algorithm:

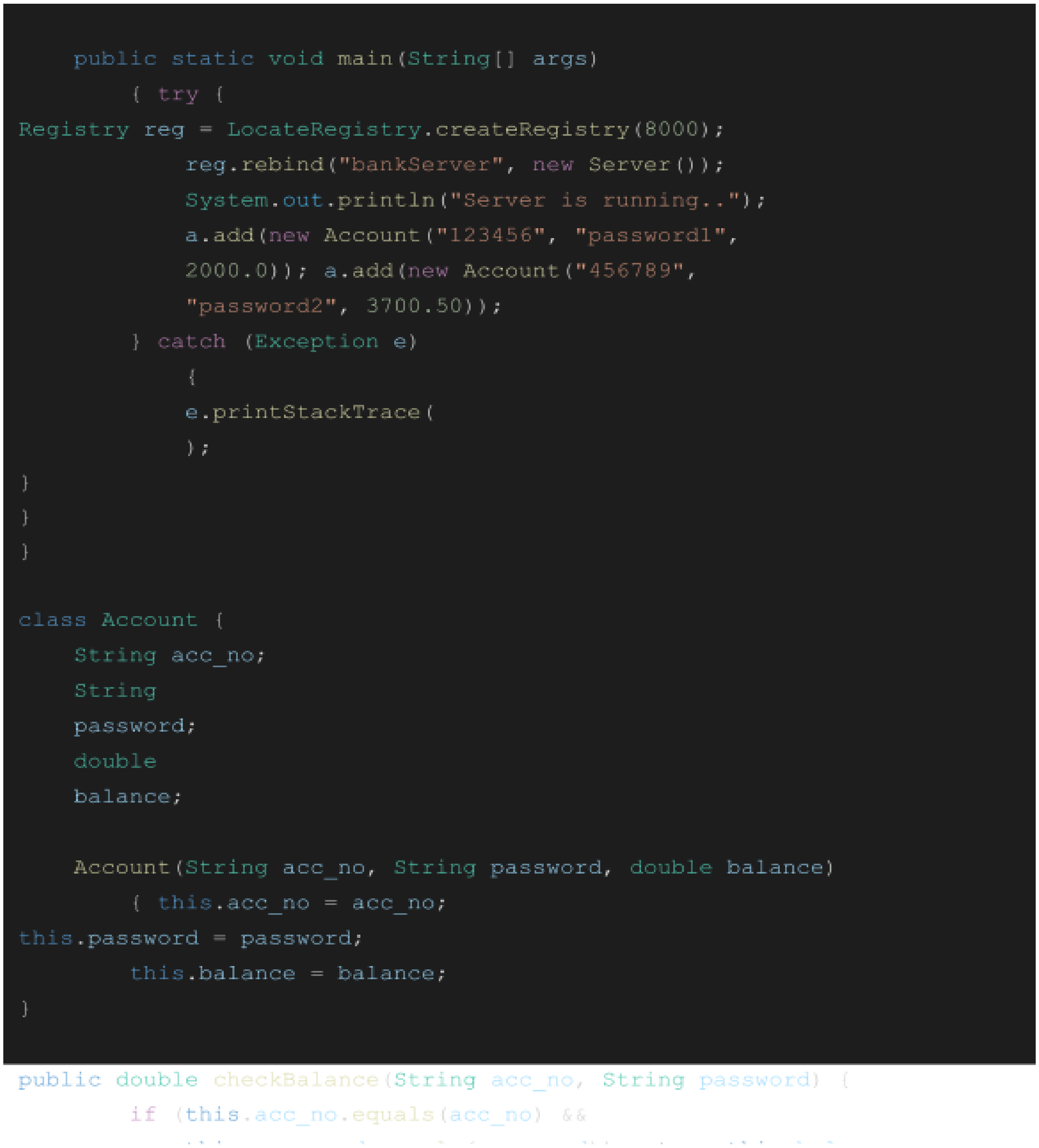
1) The process on the client machine sends the request for fetching clock time(time at server) to the Clock Server at time .
2) The Clock Server listens to the request made by the client process and returns the response in form of clock server time.
3) The client process fetches the response from the Clock Server at time T1 and calculates the synchronized client clock time using the formula given below. Tclient = Tserver + (T1-T0)/2 where Tclient refers to the synchronized clock time, Tserver refers to the clock time returned by the server, T0 refers to the time at which request was sent by the client process, T1 refers to the time at which response was received by the client process

Code:

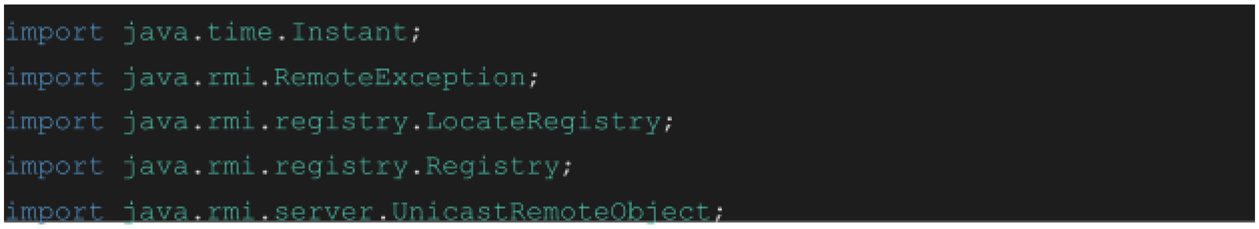


```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class Server extends UnicastRemoteObject implements checkBal {
    public Server() throws RemoteException {
super();
}

static ArrayList<Account> a = new ArrayList<Account>();

    public double checkBalance(String acc_no, String password) throws
RemoteException {
        System.out.println("Request received for account number " +
acc_no);
for (int i = 0; i < a.size(); i++) {
double bal = a.get(i).checkBalance(acc_no, password); if (bal !=
            -1)
return bal;
}
return -1.0;
}
```

Server.java

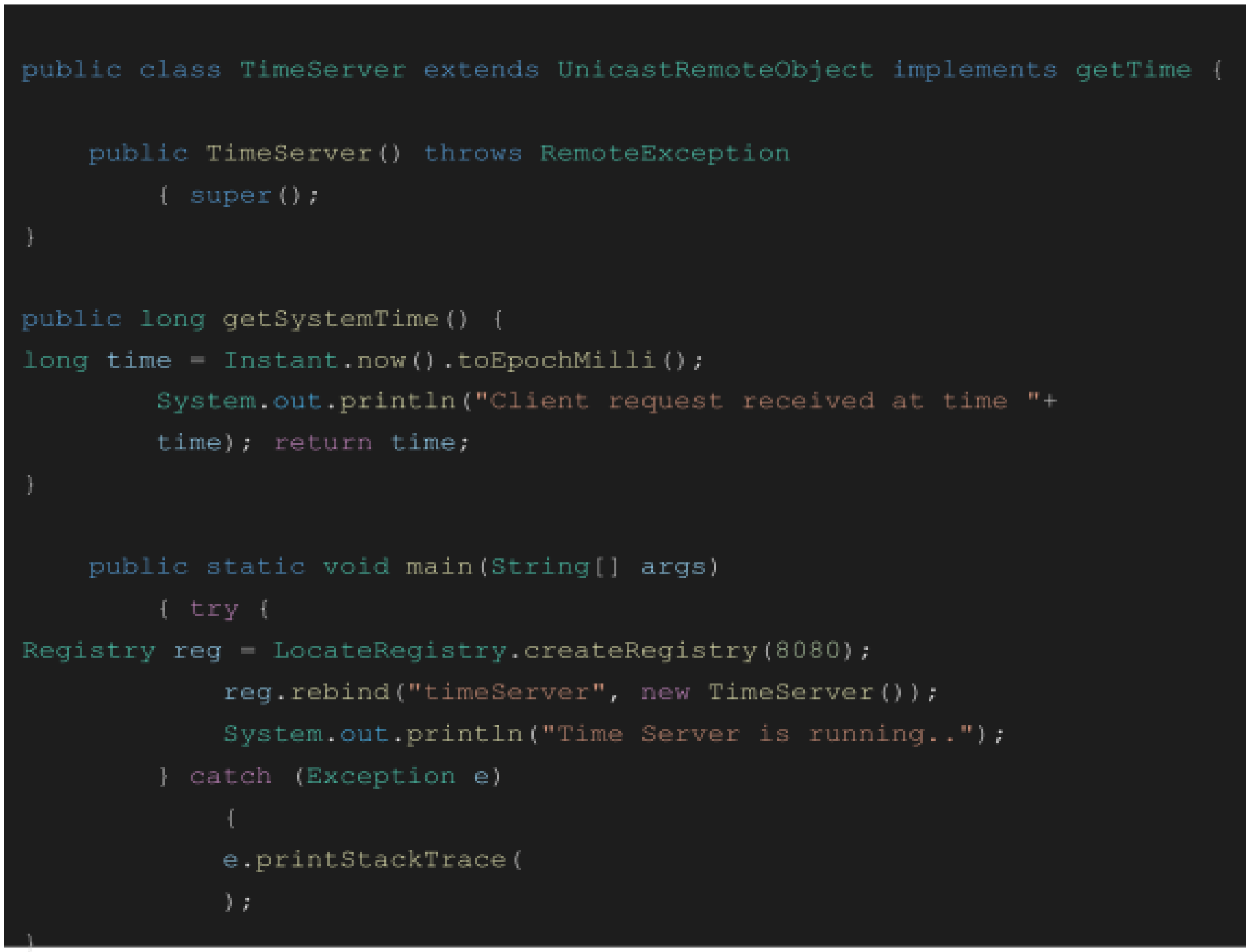


```
    public static void main(String[] args)
        { try {
Registry reg = LocateRegistry.createRegistry(8000);
            reg.rebind("bankServer", new Server());
            System.out.println("Server is running..");
            a.add(new Account("123456", "password1",
            2000.0)); a.add(new Account("456789",
            "password2", 3700.50));
        } catch (Exception e)
            {
            e.printStackTrace(
            );
}
}
}

class Account {
    String acc_no;
    String
    password;
    double
    balance;

    Account(String acc_no, String password, double balance)
        { this.acc_no = acc_no;
this.password = password;
        this.balance = balance;
}
public double checkBalance(String acc_no, String password) {
        if (this.acc_no.equals(acc_no) &&
```

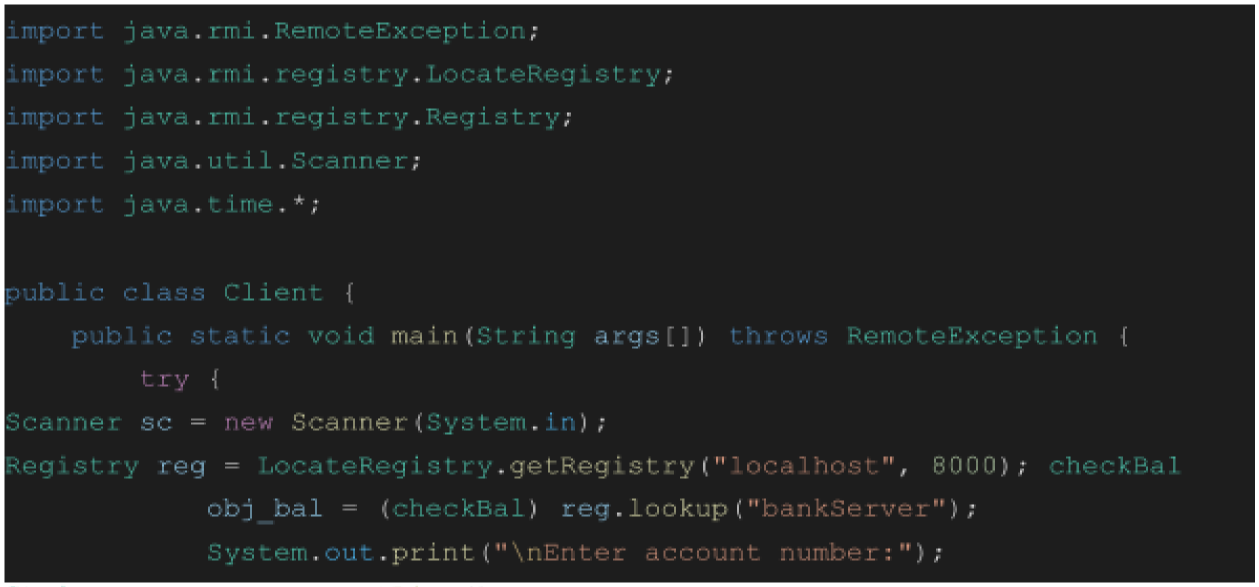


```
import java.time.Instant;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
```

TimeServer.java

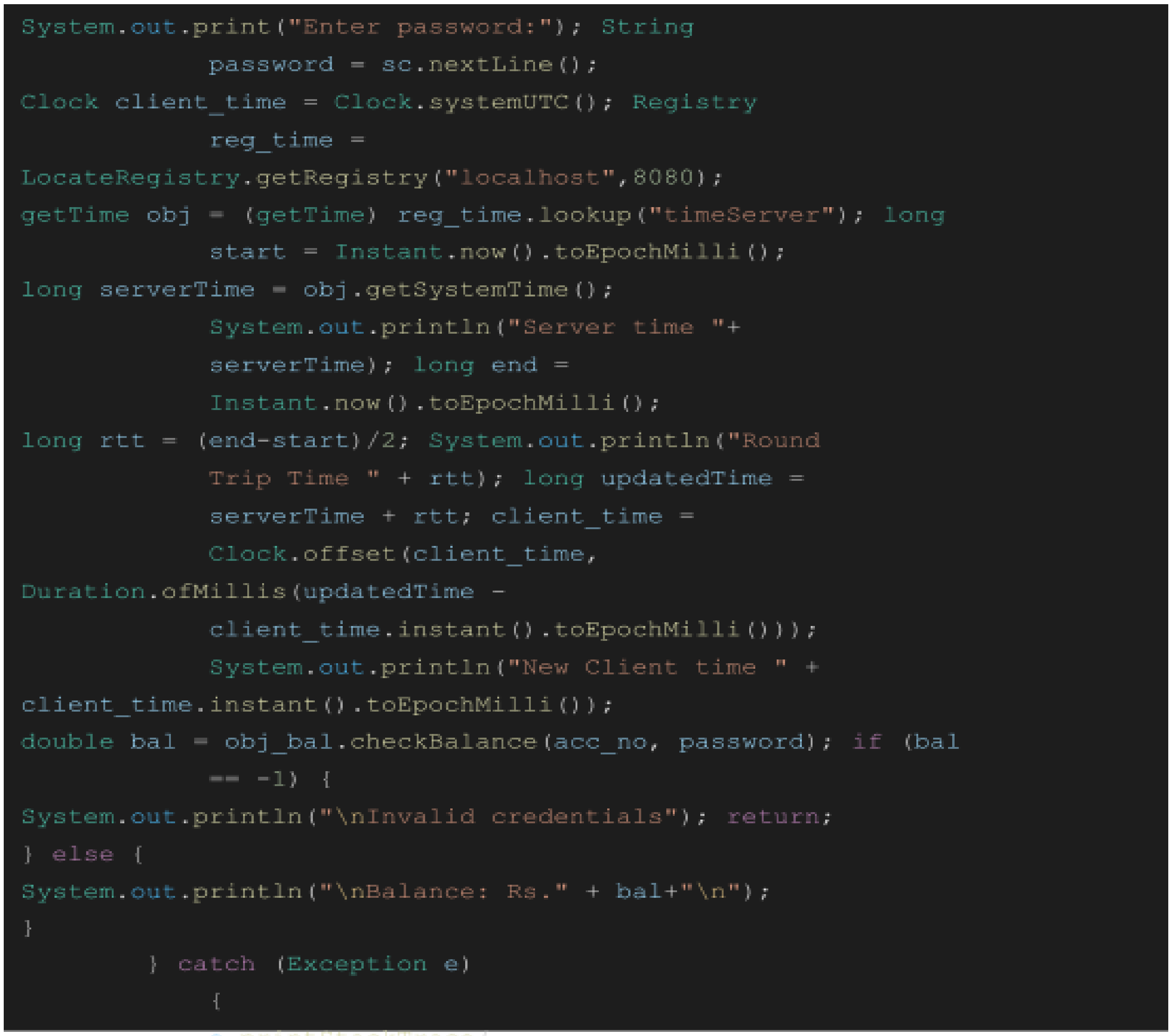


```
public class TimeServer extends UnicastRemoteObject implements getTime {

    public TimeServer() throws RemoteException
        { super();
}

public long getSystemTime() {
long time = Instant.now().toEpochMilli();
        System.out.println("Client request received at time "+
        time); return time;
}

    public static void main(String[] args)
        { try {
Registry reg = LocateRegistry.createRegistry(8080);
            reg.rebind("timeServer", new TimeServer());
            System.out.println("Time Server is running..");
        } catch (Exception e)
            {
            e.printStackTrace(
            );
}
}
}
```

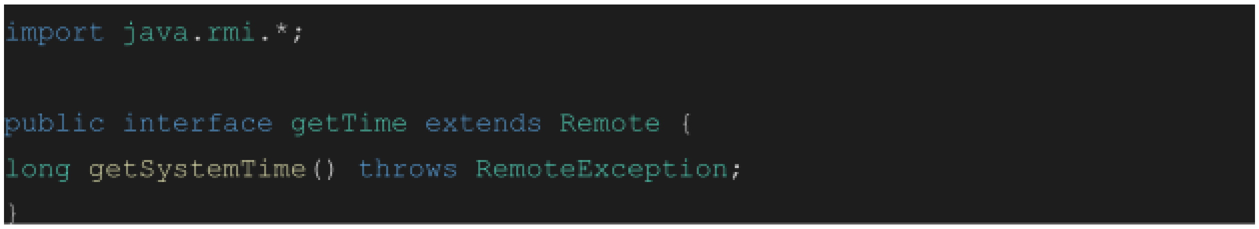


```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
import java.time.*;

public class Client {
    public static void main(String args[]) throws RemoteException {
        try {
Scanner sc = new Scanner(System.in);
Registry reg = LocateRegistry.getRegistry("localhost", 8000); checkBal
            obj_bal = (checkBal) reg.lookup("bankServer");
            System.out.print("\nEnter account number:");
String acc_no = sc.nextLine();
```

Client.java



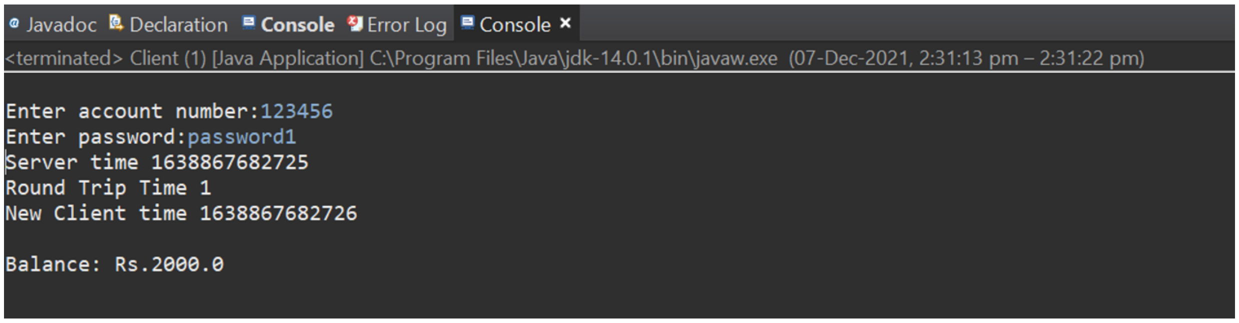
```
System.out.print("Enter password:"); String
            password = sc.nextLine();
Clock client_time = Clock.systemUTC(); Registry
            reg_time =
LocateRegistry.getRegistry("localhost",8080);
getTime obj = (getTime) reg_time.lookup("timeServer"); long
            start = Instant.now().toEpochMilli();
long serverTime = obj.getSystemTime();
            System.out.println("Server time "+
            serverTime); long end =
            Instant.now().toEpochMilli();
long rtt = (end-start)/2; System.out.println("Round
            Trip Time " + rtt); long updatedTime =
            serverTime + rtt; client_time =
            Clock.offset(client_time,
Duration.ofMillis(updatedTime -
            client_time.instant().toEpochMilli()));
            System.out.println("New Client time " +
client_time.instant().toEpochMilli());
double bal = obj_bal.checkBalance(acc_no, password); if (bal
            == -1) {
System.out.println("\nInvalid credentials"); return;
} else {
System.out.println("\nBalance: Rs." + bal+"\n");
}
        } catch (Exception e)
            {
            e.printStackTrace(
            );
}
}
```



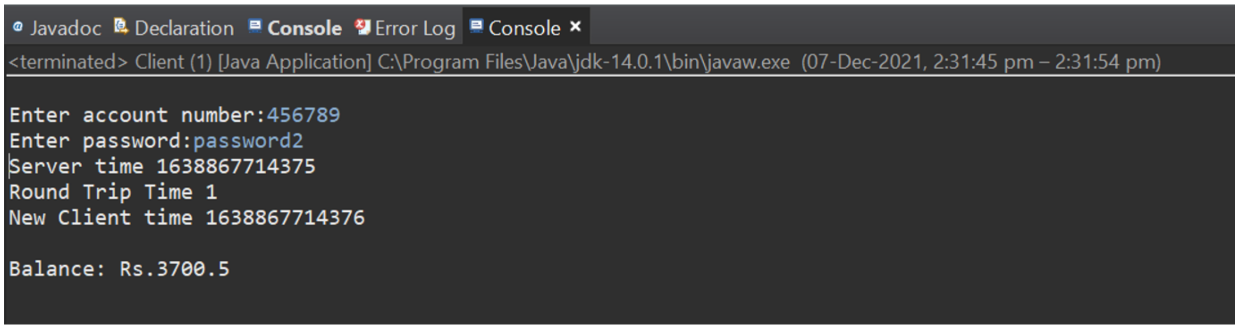
```
import java.rmi.*;

public interface getTime extends Remote {
long getSystemTime() throws RemoteException;
}
```

getTime.java

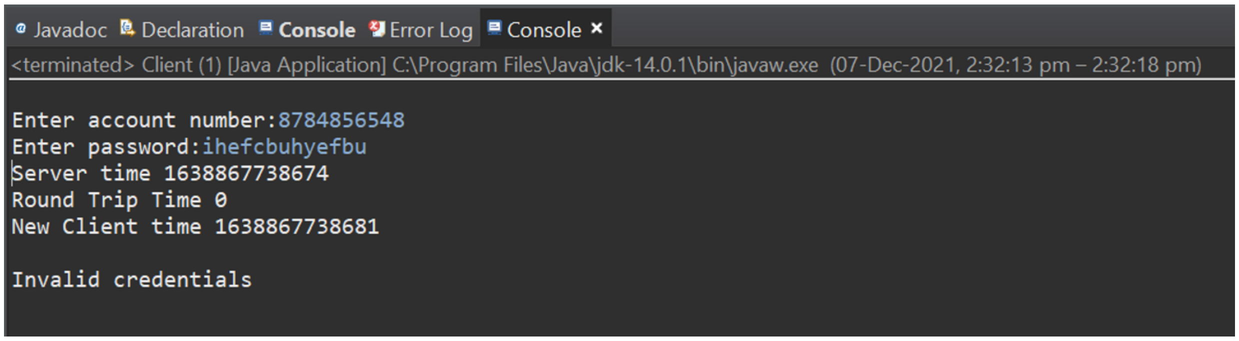
Output:



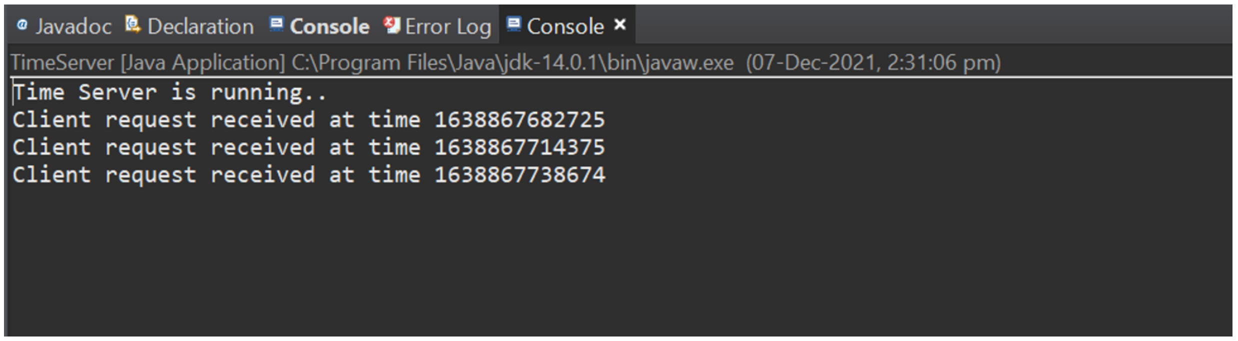
```
<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 2:31:13 pm – 2:31:22 pm)

Enter account number:123456
Enter password:password1
Server time 1638867682725
Round Trip Time 1
New Client time 1638867682726

Balance: Rs.2000.0
```



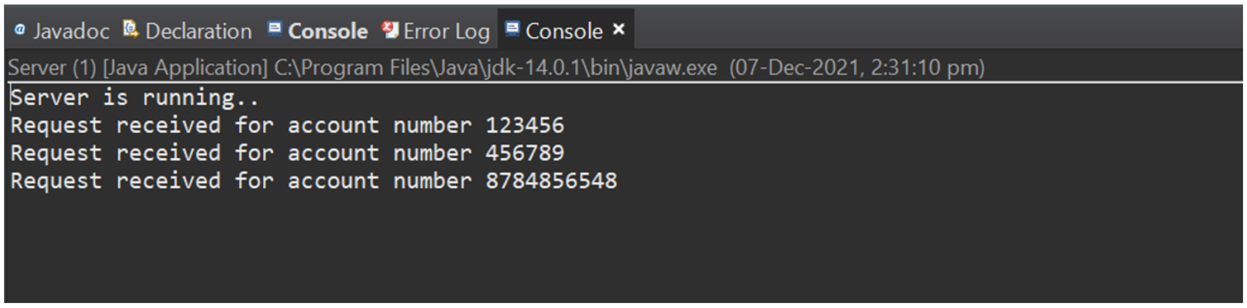
```
<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 2:31:45 pm – 2:31:54 pm)

Enter account number:456789
Enter password:password2
Server time 1638867714375
Round Trip Time 1
New Client time 1638867714376

Balance: Rs.3700.5
```



```
<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 2:32:13 pm – 2:32:18 pm)

Enter account number:8784856548
Enter password:ihefcbuhyefbu
Server time 1638867738674
Round Trip Time 0
New Client time 1638867738681

Invalid credentials
```



```
TimeServer [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 2:31:06 pm)
Time Server is running..
Client request received at time 1638867682725
Client request received at time 1638867714375
Client request received at time 1638867738674
```



```
Server (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 2:31:10 pm)
Server is running..
Request received for account number 123456
Request received for account number 456789
Request received for account number 8784856548
```

Conclusion

In this experiment, we have understood and implemented clock synchronization in the distributed system. Each time the client makes a request, the client side clock is adjusted according to the time returned by the server and this is also printed.