Banking System

(Experiment 4 -Mutual Exclusion)

Yash Brid - 2019130008
Abhishek Chopra - 2019130009
Sumeet Haldipur - 2019130018

Aim

To implement a Mutual Exclusion Algorithm for a Banking System.

Objective –

● To implement a Mutual Exclusion Algorithm in a Distributed System using Java. ● To learn about various Mutual Exclusion Algorithms and implement one of the algorithms for our problem statement, i.e.Banking System.

Problem Statement

To implement a Banking System where a user can transfer money and can check updated balance.

Theory

Mutual exclusion: Mutual exclusion is a concurrency control property that is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

Mutual exclusion in single computer system Vs. distributed system: In a single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variables (For example Semaphores) mutual exclusion problem can be easily solved.

Requirements of Mutual exclusion Algorithm:

● No Deadlock: Two or more sites should not endlessly wait for any message that will never arrive.
● No Starvation: Every site who wants to execute critical sections should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute a critical section while other sites are repeatedly executing critical sections.
● Fairness: Each site should get a fair chance to execute critical sections. Any request to execute critical sections must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
● Fault Tolerance: In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Solution to distributed mutual exclusion: As we know shared variables or a local kernel can not be used to implement mutual exclusion in distributed systems. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

1. Token Based Algorithm: - A unique token is shared among all the sites. - If a site possesses the unique token, it is allowed to enter its critical section - This approach uses sequence numbers to order requests for the critical section. - This approach ensures Mutual exclusion as the token is unique - Example: Suzuki-Kasami's Broadcast Algorithm.

2. Non-token based approach: - This approach uses timestamps instead of sequence numbers to order requests for the critical section. - Whenever a site makes a request for a critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests. - All algorithms which follow a non-token based approach maintain a logical clock. Logical clocks get updated according to Lamport's scheme - Example: Lamport's algorithm, Ricart–Agrawala algorithm

Suzuki Kasami Algorithm

Suzuki–Kasami algorithm is a token-based algorithm for achieving mutual exclusion in distributed systems.This is modification of Ricart–Agrawala algorithm, a permission based (Non-token based) algorithm which uses REQUEST and REPLY messages to ensure mutual exclusion.

Algorithm:

To enter Critical section:
1. When a site Si wants to enter the critical section and it does not have the token then it increments its sequence number RNi[i] and sends a request message REQUEST(i, sn) to all other sites in order to request the token.
2. Here sn is update value of RNi[i]
3. When a site Sj receives the request message REQUEST(i, sn) from site Si, it sets RNj[i] to maximum of RNj[i] and sn i.e RNj[i] = max(RNj[i], sn).
4. After updating RNj[i], Site Sj sends the token to site Si if it has a token and RNj[i] = LN[i] + 1
To execute the critical section: Site Si executes the critical section if it has acquired the token. To release the critical section:

1. After finishing the execution Site Si exits the critical section and does the following: 2. sets LN[i] = RNi[i] to indicate its critical section request RNi[i] has been executed 3. For every site Sj, whose ID is not present in the token queue Q, it appends its ID to Q if RNi[j] = LN[j] + 1 to indicate that site Sj has an outstanding request.
4. After the above update, if the Queue Q is non-empty, it pops a site ID from the Q and sends the token to the site indicated by the popped ID.
5. If the queue Q is empty, it keeps the token.

Code

Token.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Token extends UnicastRemoteObject implements TokenInterface {
    int token[];
    int queue[];
    int owner;
    int head;
    int tail;

    public Token() throws RemoteException {
        token = new int[3];
        queue = new int[100];
        owner = -1;
        head = 0;
        tail = 0;
    }

    public int[] getToken() throws RemoteException {
        return token;
    }

    public int[] getQueue() throws RemoteException {
        return queue;
    }
    public int getOwner() throws RemoteException {
        return owner;
    }

    public int getHead() throws RemoteException {
        return head;
    }

    public int getTail() throws RemoteException {
        return tail;
    }

    public void setToken(int index, int value) throws RemoteException {
        this.token[index] = value;
    }
```

```
    public void setQueue(int[] queue) throws RemoteException {
        this.queue = queue;
    }

    public void setOwner(int owner) throws RemoteException {
        this.owner = owner;
    }

    public void setHead(int head) throws RemoteException {
        this.head = head;
    }

    public void setTail(int tail) throws RemoteException {
        this.tail = tail;
    }

    public static void main(String args[]) throws RemoteException {
        try {
            Registry reg = LocateRegistry.createRegistry(8082);
            reg.rebind("tokenServer", new Token());
            System.out.println("Token server is running..");
        } catch (Exception e) {
            System.out.println("Exception" + e);
        }
    }
}
```

TokenInterface.java

```
import java.rmi.*;

public interface TokenInterface extends Remote {
    public int[] getToken() throws RemoteException;

    public int[] getQueue() throws RemoteException;

    public int getOwner() throws RemoteException;

    public int getHead() throws RemoteException;

    public int getTail() throws RemoteException;

    public void setToken(int index, int value) throws RemoteException;
```

```
    public void setQueue(int[] queue) throws RemoteException; public

    void setOwner(int owner) throws RemoteException; public void

    setHead(int head) throws RemoteException;

    public void setTail(int tail) throws RemoteException;
}
```

Server.java

```
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(int serverNo) throws RemoteException {
        super();
        RN = new int[3];
        no_of_requests = 0;
        critical = false;
        this.serverNo = serverNo;
        try {
            Registry reg = LocateRegistry.getRegistry("localhost", 8082);
            TokenInterface token = (TokenInterface)
reg.lookup("tokenServer");
            this.token = token;
        } catch (Exception e) {
            System.out.println("Exception occurred : " + e.getMessage());
        }
    }

    static ArrayList<Account> a = new ArrayList<Account>() {
        {
            add(new Account("123456", "password1", 2000.0));
            add(new Account("456789", "password2", 3000.0));
            add(new Account("234567", "password3", 4000.0));
            add(new Account("345678", "password4", 5000.0));
        }
    };
    int RN[];
    boolean critical;
    int no_of_requests;
```

```
    TokenInterface token;
    int serverNo;

    public double checkBalance(String acc_no, String password) throws
RemoteException {
        System.out.println("Balance 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;
    }

    public boolean transfer(String d_acc_no, String cred_acc_no, String
password, double amt) throws RemoteException {
        System.out.println("Transfer request received for account number "
+ d_acc_no);
        System.out.println("Transfer to credit account number " +
cred_acc_no);
        boolean isValid = false;
        for (int i = 0; i < a.size(); i++) {
            isValid = a.get(i).checkValid(d_acc_no, password);
            if (isValid) {
                break;
            }
        }
        if (!isValid) {
            return false;
        } else {
            if (token.getOwner() == -1) {
                token.setOwner(serverNo);
                System.out.println("No owner");
                no_of_requests++;
                RN[serverNo]++;
            } else {
                sendRequest();
            }
            while (token.getOwner() != serverNo)
                ;
            System.out.println("Got token");
            critical = true;
```

```
            boolean b = critical_section(d_acc_no, cred_acc_no, password,
amt);
            critical = false;
            releaseToken();
            return b;
        }

    }

    public void sendRequest() throws RemoteException {
        no_of_requests++;
        for (int i = 0; i < 1; i++) {
            try {
                Registry reg = LocateRegistry.getRegistry("localhost",
8000+i);
                checkBal server = (checkBal) reg.lookup("bankServer"+i);
                server.receiveRequest(serverNo, no_of_requests);
            } catch (Exception e) {
                System.out.println("Exception occurred : " +
e.getMessage());
            }
        }
    }

    public boolean critical_section(String d_acc_no, String cred_acc_no,
String password, double amt) {
        int deb_ind = 0;
        int cred_ind = 0;
        for (int i = 0; i < a.size(); i++) {
            if (a.get(i).acc_no.equals(d_acc_no) &&
a.get(i).password.equals(password)) {
                deb_ind = i;
            }
            if (a.get(i).acc_no.equals(cred_acc_no)) {
                cred_ind = i;
            }
        }
        if (a.get(deb_ind).balance < amt)
            return false;
        else {
            a.get(deb_ind).balance -= amt;
            a.get(cred_ind).balance += amt;
            return true;
```

```
    }
}

public void receiveRequest(int i, int n) throws RemoteException {
        System.out.println("Received request from " + i);
    if (RN[i] <= n) {
        RN[i] = n;
        if (token.getToken()[i] + 1 == RN[i]) {
            if (token.getOwner() == serverNo) {
                if (critical) {
                    System.out.println("Add to queue");
                    token.getQueue()[token.getTail()] = i;
                    token.setTail(token.getTail() + 1);
                } else {
                    System.out.println("Queue empty, setting owner");
                    token.setOwner(i);
                }
            }
        }
    }
}

public void releaseToken() throws RemoteException {
    token.setToken(serverNo, RN[serverNo]);
    if (token.getHead() != token.getTail()) {
        System.out.println("Release token");
        token.setOwner(token.getQueue()[token.getHead()]);
        System.out.println("New owner" + token.getOwner());
        token.setHead(token.getHead() + 1);
    }
}

public static void main(String[] args) {
    try {
        Registry reg = LocateRegistry.createRegistry(8000);
        reg.rebind("bankServer0", new Server(0));

        Registry reg1 = LocateRegistry.createRegistry(8001);
        reg1.rebind("bankServer1", new Server(1));

        Registry reg2 = LocateRegistry.createRegistry(8002);
        reg2.rebind("bankServer2", new Server(2));
```

```
            System.out.println("3 servers are running now ");
        } 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) && this.password.equals(password))
        return this.balance;
        else
            return -1.0;
    }

    public boolean checkValid(String acc_no, String password) { if
        (this.acc_no.equals(acc_no) && this.password.equals(password))
        return true;
        else
            return false;
    }
}
```

Client.java

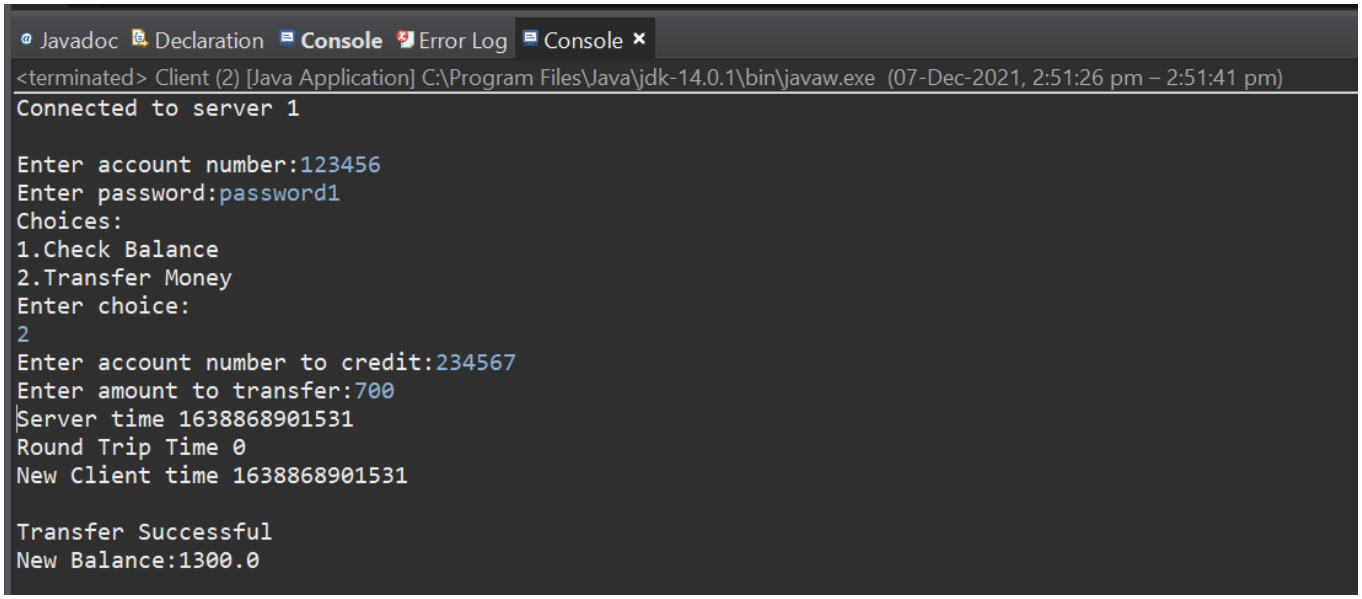
```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
import java.time.*;

public class Client {
    public static void main(String args[]) throws RemoteException {
        try {
```

```
            Scanner sc = new Scanner(System.in);
            Registry reg = LocateRegistry.getRegistry("localhost", 8081);
            loadBalancerInterface lb = (loadBalancerInterface)
reg.lookup("loadBalancer");
            System.out.println("Connected to server " +
lb.getServerName());
            checkBal obj_bal = lb.getServer();
            Clock client_time;
            Registry reg_time;
            getTime obj;
            long start;
            long serverTime;
            long end;
            long rtt;
            long updatedTime;
            System.out.print("\nEnter account number:");
            String acc_no = sc.nextLine();
            System.out.print("Enter password:");
            String password = sc.nextLine();
            System.out.println("Choices:\n1.Check Balance\n2.Transfer
Money\nEnter choice:");
            int ch = sc.nextInt();
            switch (ch) {
                case 1:
                client_time = Clock.systemUTC();
                reg_time = LocateRegistry.getRegistry("localhost", 8080);
                obj = (getTime) reg_time.lookup("timeServer");
                start = Instant.now().toEpochMilli();
                serverTime = obj.getSystemTime();
                System.out.println("Server time " + serverTime);
                end = Instant.now().toEpochMilli();
                rtt = (end - start) / 2;
                System.out.println("Round Trip Time " + rtt);
                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 {
```

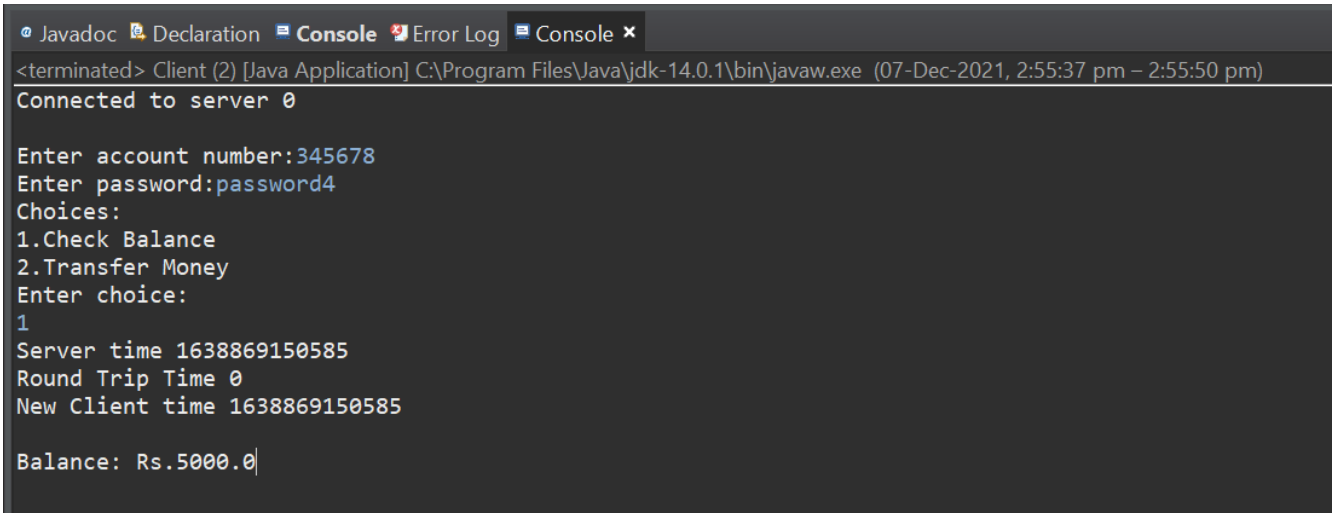
```
                    System.out.println("\nBalance: Rs." + bal + "\n");
                }
                break;

                case 2:
                sc.nextLine();
                System.out.print("Enter account number to credit:");
                String cred_acc_no = sc.nextLine();

                System.out.print("Enter amount to transfer:");
                double amt = sc.nextDouble();

                client_time = Clock.systemUTC();
                reg_time = LocateRegistry.getRegistry("localhost", 8080);
                obj = (getTime) reg_time.lookup("timeServer");
                start = Instant.now().toEpochMilli();
                serverTime = obj.getSystemTime();
                System.out.println("Server time " + serverTime);
                end = Instant.now().toEpochMilli();
                rtt = (end - start) / 2;
                System.out.println("Round Trip Time " + rtt);
                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());
                boolean status = obj_bal.transfer(acc_no, cred_acc_no,
password, amt);
                if (status) {
                    System.out.println("\nTransfer Successful");
                    System.out.println("New
Balance:"+obj_bal.checkBalance(acc_no, password));
                } else {
                    System.out.println("\nError occured");
                }
                break;
                default:
                System.out.println("Wrong choice entered");
            }
            return;

        } catch (Exception e) {
            e.printStackTrace();
```

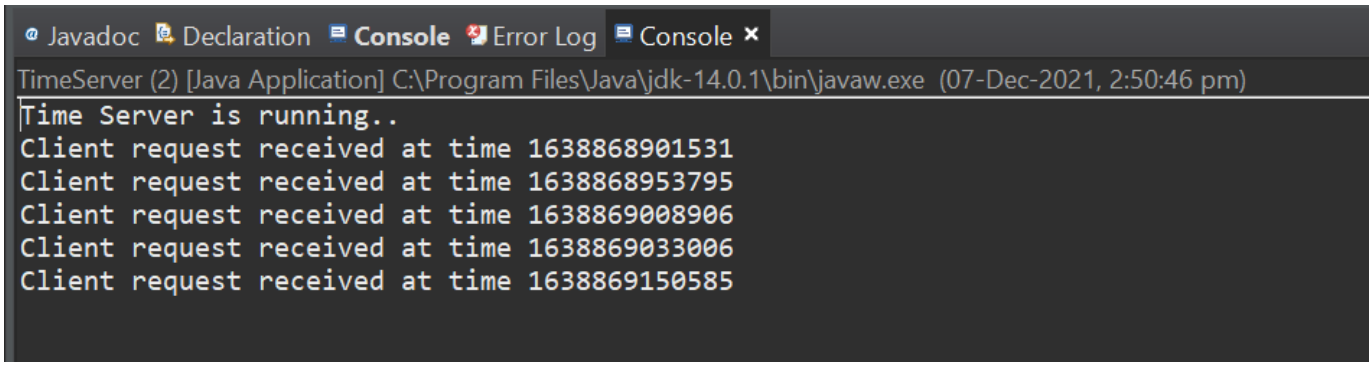
```
        }
    }
}
```

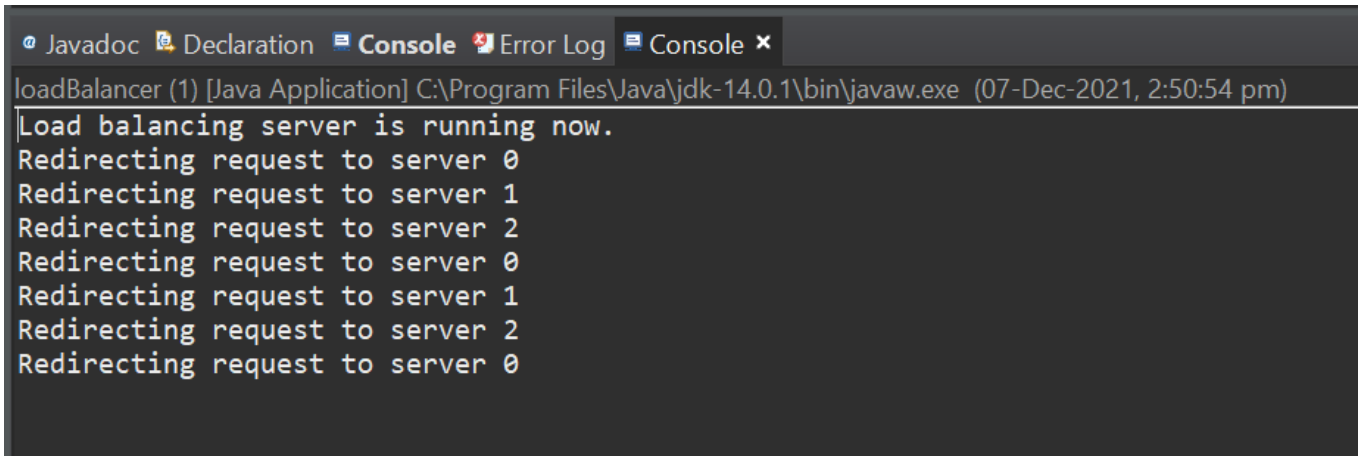
Output

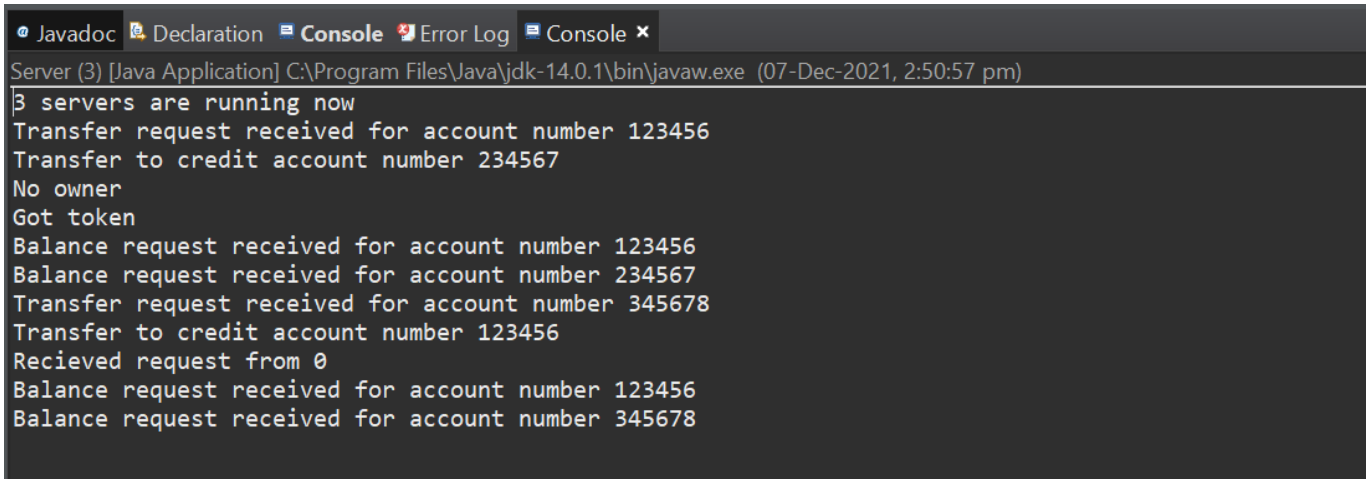

Javadoc Declaration Console Error Log Console
<terminated> Client (2) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (07-Dec-2021, 2:51:26 pm – 2:51:41 pm)
Connected to server 1
Enter account number:123456
Enter password:password1
Choices:
1.Check Balance
2.Transfer Money
Enter choice:
2
Enter account number to credit:234567
Enter amount to transfer:700
Server time 1638868901531
Round Trip Time 0
New Client time 1638868901531
Transfer Successful
New Balance:1300.0

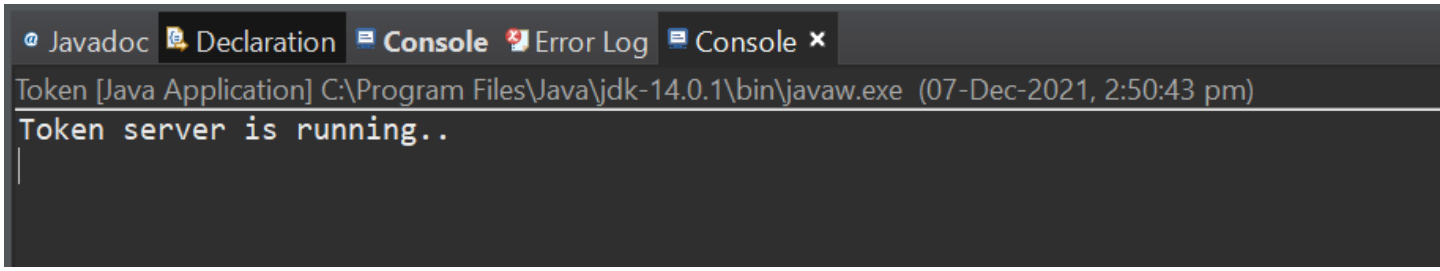

Javadoc Declaration Console Error Log Console
<terminated> Client (2) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (07-Dec-2021, 2:52:02 pm – 2:52:33 pm)
Connected to server 2
Enter account number:234567
Enter password:password3
Choices:
1.Check Balance
2.Transfer Money
Enter choice:
1
Server time 1638868953795
Round Trip Time 0
New Client time 1638868953795
Balance: Rs.4700.0


Javadoc Declaration Console Error Log Console
<terminated> Client (2) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (07-Dec-2021, 2:55:37 pm – 2:55:50 pm)
Connected to server 0
Enter account number:345678
Enter password:password4
Choices:
1.Check Balance
2.Transfer Money
Enter choice:
1
Server time 1638869150585
Round Trip Time 0
New Client time 1638869150585
Balance: Rs.5000.0


Javadoc Declaration Console Error Log Console
TimeServer (2) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (07-Dec-2021, 2:50:46 pm)
Time Server is running..
Client request received at time 1638868901531
Client request received at time 1638868953795
Client request received at time 1638869008906
Client request received at time 1638869033006
Client request received at time 1638869150585


Javadoc Declaration Console Error Log Console
loadBalancer (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (07-Dec-2021, 2:50:54 pm)
Load balancing server is running now.
Redirecting request to server 0
Redirecting request to server 1
Redirecting request to server 2
Redirecting request to server 0
Redirecting request to server 1
Redirecting request to server 2
Redirecting request to server 0





Conclusion:

From this experiment, we learned about the importance of mutual exclusion in Distributed Systems. We used the Suzuki Kasami Mutual Exclusion Algorithm to ensure that only one server is able to execute the critical section at a time while others wait.