SequencerImpl.java

```java
package com.distributed;
/*THIS IS THE SERVER.. IT SHOULD BE RUNNING FOR RMI TO WORK FOR SEQUNCER..*/
import java.io.IOException;
import java.net.InetAddress;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.*;
import java.rmi.*;
import java.rmi.registry.*;

/*UnicastRemoteObject makes the server object exportable to the client system
through Remote reference layer (here, client gets a reference of remote object
of server, or to say, server passes the remote object by reference (pass-by-reference) to client).
 */
public class SequencerImpl extends UnicastRemoteObject implements Sequencer {
     long seqnumber=0;
    final static String INET_ADDR = "224.0.0.3";
    InetAddress addr;
    SequencerJoinInfo infor;
    String sender;
    byte[] message;
    long messageID;
    long lastSeqReceived1;
    boolean leave= false;

    SequencerImpl()  throws RemoteException{
        super();
        try {
            addr = InetAddress.getByName(INET_ADDR);
        infor = new SequencerJoinInfo(addr, seqnumber);
    } catch (IOException e) {
        System.out.println(e.toString());
    }
}
    @Override
    public SequencerJoinInfo join(String sender) throws RemoteException, SequencerException {

            // join -- request for "sender" to join sequencer's multicasting service;
            // returns an object specifying the multicast address and the first sequence number to expect
            System.out.println(sender + " requesting to join sequencer's multicasting service");
            this.sender=sender;
        return infor;
    }

    @Override
    public void send(String sender, byte[] msg, long msgID, long lastSequenceReceived) throws RemoteException {
        // send -- "sender" supplies the msg to be sent, its identifier,
        // and the sequence number of the last received message
        this.sender=sender;
        seqnumber=seqnumber+1;
        message=msg;
        messageID=msgID;
        lastSeqReceived1=lastSequenceReceived;


    }
    @Override
    public void leave(String sender) throws RemoteException {
// leave -- tell sequencer that "sender" will no longer need its services

        leave=true;
    }

    @Override
    public byte[] getMissing(String sender, long sequence) throws RemoteException, SequencerException {
        // getMissing -- ask sequencer for the message whose sequence number is "sequence"

        return new byte[0];
    }
    @Override
    public void heartbeat(String sender, long lastSequenceReceived) throws RemoteException {
```

```
72  // heartbeat -- we have received messages up to number "lastSequenceReceived"
73
74          System.out.println("we have received messages up to number  "+lastSeqReceived1);
75      }
76      public static void main(String args[]) throws Exception {
77          //creates a remote object,links with an
78          // alias name and binds with the RMI registry,linked to RMI runtime mechanism.
79          Sequencer seql = new SequencerImpl();
80          Registry registry = LocateRegistry.createRegistry(1099);
81          registry.rebind("sequences", seql);  // sequences is alias name for seql and is used later by the client
82          System.out.println("Sequence  server is  ready");
83      }
84  }
85
```