Group.java

```java
package com.distributed;
import java.net.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.registry.*;

public class Group implements Runnable {
    public static String host = null;
    public static String senderName = null;
    public static MulticastSocket s = null;
    public static InetAddress group = null;
    public static byte[] buf = null;
    public static Registry registry = null;
    public static Sequencer server;
    public static long lastSeqReceived1 = 0;
    public static int count = 0;
    public static MsgHandler handles = null;
    public static History hist;

    public Group(String host, MsgHandler handler, String senderName) throws GroupException {
        // contact Sequencer on "host" to join group,
        // create MulticastSocket and thread to listen on it,
        // perform other initialisations
        //this.handler =handler;
        this.host = host;
        this.senderName = senderName;
        handles = handler;
        hist = new History();
        try {

            int port = 6789;
            registry = LocateRegistry.getRegistry();
            server = (Sequencer) registry.lookup("sequences");
            SequencerJoinInfo object = server.join(senderName);
            InetAddress INET_ADDR = object.addr;
            group = INET_ADDR;
            s = new MulticastSocket(port);
            // s.setTimeToLive(1);
            s.joinGroup(group);

        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }

    public void send(byte[] msg) throws GroupException {
        // send the given message to all instances of Group using the same sequencer
        String messagelog = new String(msg, 0, msg.length);
        hist.storeHistory(messagelog); //store history
        byte[] m = msg;
        try {
            DatagramPacket messageOut = new DatagramPacket(m, m.length, group, 6789);
            s.send(messageOut);
        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
    }

    public void leave() {
        try {
            server.leave(senderName);
            s.leaveGroup(group);
            // leave group
        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
    }

    public void run() {
        // repeatedly: listen to MulticastSocket created in constructor, and on receipt
        // of a datagram call "handle" on the instance
        // of Group.MsgHandler which was supplied to the constructor
        buf = new byte[256];
        try {

            while (true) {
                // Receive the information and print it .
                DatagramPacket msgPacket = new DatagramPacket(buf, buf.length);
                lastSeqReceived1 = lastSeqReceived1 + 1;
                count = count + 1;
                s.receive(msgPacket);

                handles.handle(count, buf);

            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public interface MsgHandler {
        public void handle(int count, byte[] msg);
    }

    public class GroupException extends Exception {

        public GroupException(String s) {
            super(s);
        }

    }
    public class HeartBeater extends Thread {
        // This thread sends heartbeat messages when required
        public void HeartBeaters() throws Exception {
            server.heartbeat(senderName, lastSeqReceived1);
        }

    }
    public void viewhistory() {
        hist.viewHistory();

    }
}
```