

Name:

Department: Computer Engineering

Class & Semester: B.E (Final Year), SEM VIII

Subject: Distributed Computing Lab (DCL)

Expt. No. 6

Title: Write a Program to Demonstrate Group Communication using Java.

Date:

Subject In-charge Sign:

.....

Experiment No. 06

Aim: Write A Program to Demonstrate Group Communication using Java.

Theory:

Java Groups is a group communication toolkit written entirely in Java. It is based on IP multicast, but extends it with

1. reliability and
2. Group membership.

Reliability includes

- lossless transmission of a message to all recipients (with retransmission of missing messages)
- fragmentation of large messages into smaller ones and reassembly at the receiver's side
- ordering of messages, e.g. messages m1 and m2 sent by P will be received by all receivers in the same order, and not as m2, m1 (FIFO order)
- Atomicity: a message will be received by all receivers, or none.

Group Membership includes

- Knowledge of who the members of a group are and
- Notification when a new member joins, an existing member leaves, or an existing member has crashed

The table below shows where Java Groups fits in. In unicast communication, where one sender sends a message to one receiver, there is UDP and TCP. UDP is unreliable, packets may get lost, duplicated, may arrive out of order, and there is a maximum packet size restriction. TCP is also fragments packets that are too big and present messages to the application in the order in which they were sent. In the multicast case, where one sender sends a message to many receivers, IP Multicast extends UDP: a sender sends messages to a multicast address and the receivers have to join that multicast address to receive them. Like in UDP, message transmission is still unreliable, and there is no notion of membership (who has currently joined the multicast address).

	Unreliable	Reliable
Unicast	UDP	TCP
Multicast	IP Multicast	Java Groups

Java Groups extends reliable unicast message transmission (like in TCP) to multicast settings. As described above it provides reliability and group membership on top of IP Multicast. Since every application has different reliability needs, Java Groups provides a flexible protocol stack

architecture that allows users to put together custom-tailored stacks, ranging from unreliable but fast to highly reliable but slower stacks. As an example, a user might start with a stack only containing IP Multicast. To add loss-less transmission, he might add the NAKACK protocol (which also weeds out duplicates). Now messages sent from a sender are always received by the recipients, but the order in which they will be received is undefined. Therefore, the user might choose to add the FIFO layer to impose per/sender ordering. If ordering should be imposed over all the senders, then the TOTAL protocol may be added. The Group Membership Service (GMS) and FLUSH protocols provide for group membership: they allow the user to register a callback function that will be invoked whenever the membership changes, e.g. a member joins, leaves or crashes. In the latter case, a failure detector (FD) protocol is needed by the GMS to announce crashed members. If new members want to obtain the current state of existing members, then the STATE_TRANSFER protocol has to be present in this custom-made stack. Finally, the user may want to encrypt messages, so he adds the CRYPT protocol (which encrypts/decrypts messages and takes care of re-keying using a key distribution toolkit).

Using composition of protocols (each taking care of a different aspect) to build reliable multicast communication has the benefit that

- users of a stack only pay for the protocols they use and
- Since protocols are independent of each other, they can be modified, replaced or new ones can be added, improving modularity and maintainability.

The Java Groups API

The API of JavaGroups is very simple (similar to a UDP socket) and is always the same, regardless of how the underlying protocol stack is composed. To be able to send/receive messages, a *channel* has to be created. The reliability of a channel is specified as a string, which then causes the creation of the underlying protocol stack. The example below creates a channel and sends/receives 1 message:

To join a group, **Connect ()** is called. It returns when the member has successfully joined the group named "MyGroup", or when it has created a new group (if it is the first member).

Then a message is sent using the **Send ()** method. A message contains the receiver's address ('null' = all group members), the sender's address ('null', will be filled in by the protocol stack before sending the message) and a byte buffer. In the example, the String object "Hello world" is set to be the message's contents. It is serialized into the message's byte buffer.

Since the message is sent to all members, the sender will also receive it (unless the socket option 'receive own multicasts' is set to 'true'). This is done using the **Receive ()** method.

Finally, the member disconnects from the channel and then closes it. This results in a notification being sent to all members who are registered for membership change notifications.

Program:

Server.java

```
import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Vector;

public class Server

{

    private static Vector<PrintWriter> writers=new Vector<PrintWriter>();

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

    {

        ServerSocket listener=new ServerSocket(9001);

        System.out.println("The server is running at port 9001.");

        while(true)

            new Handler(listener.accept()).start();

    }

    private static class Handler extends Thread

    {

        private Socket socket;

        public Handler(Socket socket)

        {
```

```
        this.socket=socket;

    }

    public void run()

    {

        try

        {

BufferedReader in=new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out=new PrintWriter(socket.getOutputStream(), true);

out.println("SUBMITNAME");

String name=in.readLine();

System.out.println(name+ "joined");

writers.add(out);

while(true)

{

    String input=in.readLine();

    for (PrintWriter writer:writers)

        writer.println("MESSAGE" +name+ ":" +input);

}

}

    catch(Exception e)

    {

        System.err.println(e);

    }

}

}
```

```
    }  
}
```

master.java

```
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.Socket;  
import java.util.Scanner;  
public class master  
{  
    public static void main(String args[]) throws Exception  
    {  
        Scanner sc=new Scanner(System.in);  
        Socket socket=new Socket("localhost",9001);  
        BufferedReader in=new BufferedReader(new InputStreamReader(socket.getInputStream()));  
        PrintWriter out=new PrintWriter(socket.getOutputStream(), true);  
        System.out.print("Enter your name:");  
        String name=sc.nextLine();  
        while(true)  
        {  
            String line=in.readLine();  
            if(line.startsWith("SUBMITNAME")) out.println(name);  
            else if (line.startsWith("MESSAGE"))  
                System.out.println(line.substring(8));  
            if(name.startsWith("master"))
```

```

        {
            System.out.print("Enter a message:");
            out.println(sc.nextLine());
        }
    }
}
}

```

slave1.java

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class slave1
{
    public static void main(String[] args) throws Exception
    {
        Scanner sc= new Scanner(System.in);

        Socket socket=new Socket("localhost",9001);

        BufferedReader in=new BufferedReader(new InputStreamReader(socket.getInputStream()));

        PrintWriter out=new PrintWriter(socket.getOutputStream(), true);

        System.out.print("Enter your name: ");

        String name =sc.nextLine();

        while (true)
        {
            String line=in.readLine();

```

```

        if(line.startsWith("SUBMITNAME")) out.println(name);
        else if (line.startsWith("MESSAGE"))
            System.out.println(line.substring(8));
        if(name.startsWith("master"))
        {
            System.out.print("Enter a message: ");
            out.println(sc.nextLine());
        }
    }
}
}

```

slave2.java

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class slave2
{
    public static void main(String [] args) throws Exception
    {
        Scanner sc=new Scanner(System.in);
        Socket socket=new Socket("localhost",9001);
        BufferedReader in=new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }
}

```



```
PrintWriter out=new PrintWriter(socket.getOutputStream() , true);

System.out.print("Enter your name:");

String name=sc.nextLine();

while(true)
{
    String line=in.readLine();

    if(line.startsWith("SUBMITNAME"))

        out.println(name);

    else if (line.startsWith("MESSAGE"))

        System.out.println(line.substring(8));

    if(name.startsWith("master"))

        {

            System.out.print("Enter a message:");

            out.println(sc.nextLine());

        }

}

}
```

Output:

```
C:\>java Server
The server is running at port 9001.
```

```
C:\ C:\WINXP\system32\cmd.exe - java Server
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\

C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"

C:\>d:

D:\>java Server
The server is running at port 9001.
masterjoined
```

```
C:\ C:\WINXP\system32\cmd.exe - java master
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\

C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"

C:\>d:

D:\>java master
Enter your name:master
Enter a message:_
```

```
C:\WINXP\system32\cmd.exe - java Server
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\
C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"
C:\>d:
D:\>java Server
The server is running at port 9001.
masterjoined
slave1joined
slave2joined
-
```

```
C:\WINXP\system32\cmd.exe - java master
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\
C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"
C:\>d:
D:\>java master
Enter your name:master
Enter a message: _
```

```
C:\WINXP\system32\cmd.exe - java slave1
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\
C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"
C:\>d:
D:\>java slave1
Enter your name: slave1
-
```

```
C:\WINXP\system32\cmd.exe - java slave2
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\

C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"

C:\>d:

D:\>java slave2
Enter your name:slave2
```

```
C:\WINXP\system32\cmd.exe - java Server
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\

C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"

C:\>d:

D:\>java Server
The server is running at port 9001.
masterjoined
slave1joined
slave2joined
_
```

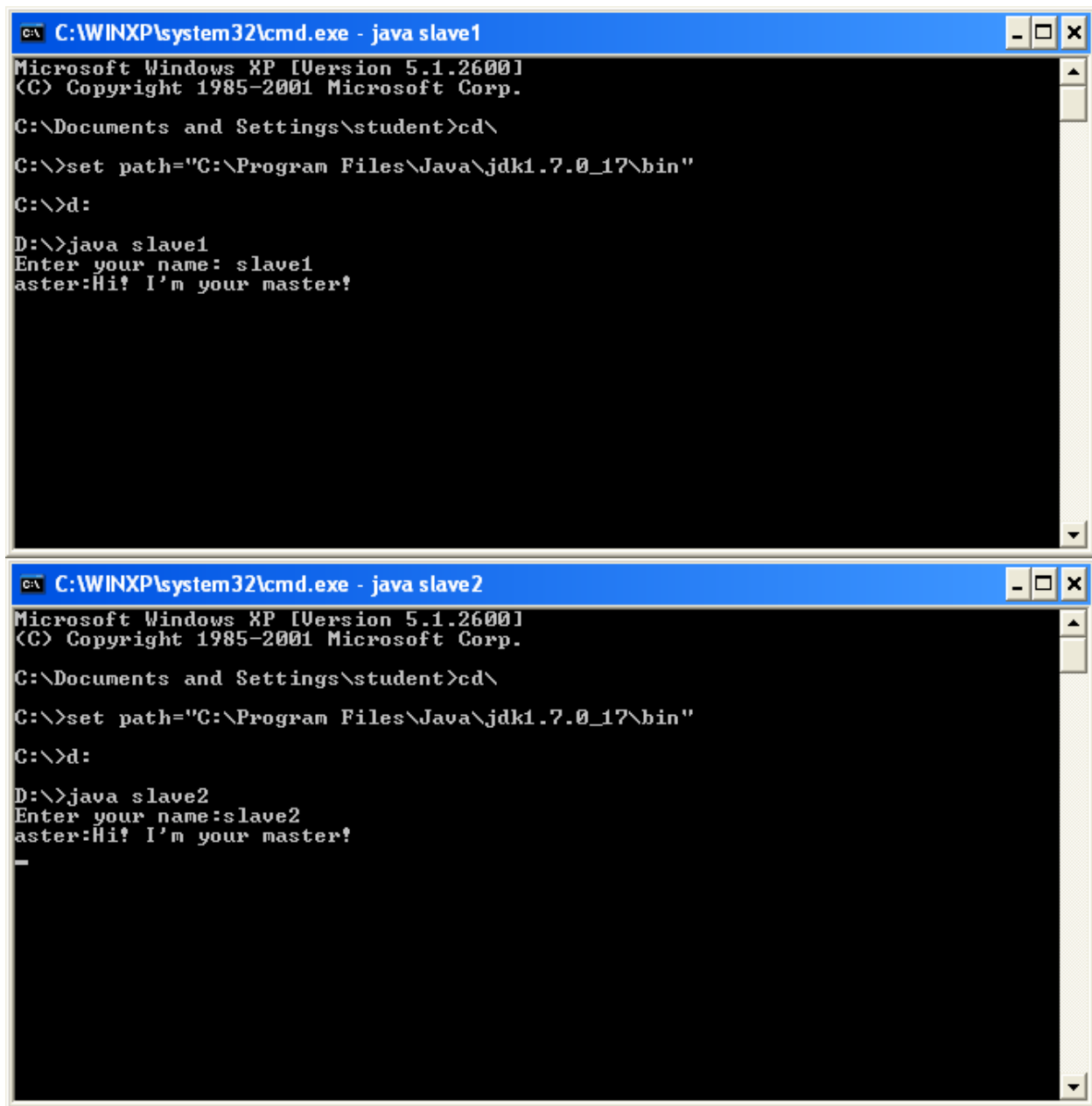
```
C:\WINXP\system32\cmd.exe - java master
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\

C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"

C:\>d:

D:\>java master
Enter your name:master
Enter a message:Hi! I'm your master!
aster:Hi! I'm your master!
Enter a message:_
```



The image displays two separate Windows XP command prompt windows, each titled "C:\WINXP\system32\cmd.exe - java slave1" and "C:\WINXP\system32\cmd.exe - java slave2". Both windows show the same sequence of commands and output. The first window shows the execution of "java slave1", where the user enters "slave1" and receives the response "aster:Hi! I'm your master!". The second window shows the execution of "java slave2", where the user enters "slave2" and receives the same response "aster:Hi! I'm your master!". The command prompts show the user's current directory as "C:\Documents and Settings\student\" and the path to the Java JDK bin directory is added to the system path.

```
C:\WINXP\system32\cmd.exe - java slave1
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\
C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"
C:\>d:
D:\>java slave1
Enter your name: slave1
aster:Hi! I'm your master!

C:\WINXP\system32\cmd.exe - java slave2
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\student>cd\
C:\>set path="C:\Program Files\Java\jdk1.7.0_17\bin"
C:\>d:
D:\>java slave2
Enter your name:slave2
aster:Hi! I'm your master!
_
```

Conclusion: Thus we have demonstrated Group communication using Java.