

NA1
PROJECT
DOCUMENTATION
Fall 2015

TRIVIKRAM REDDY GURUKUNTA 16212362

PRUDHVI RAJ MUDUNURI 16208160

SNEHAL VANTASHALA 16206488

Part I.

GENI/Socket programming Warm-up (50%)

Develop and deploy a simple TCP client and server programs on GENI (refer GENI LabZero slide and sample socket programs). Show the screenshots of simple message exchanges. Start from client message 'Hello from Client-your names' and server responses with 'Hello from Server-your names'. Then messages from each side are echoed to each other. The program quit the program with typing 'Bye from Client-your name' and 'Bye from Server-your name'.

SOL:-

TCP Server:-

```
import java.io.*;
import java.net.*;

class TCPServer {

    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {

            Socket connectionSocket = welcomeSocket.accept();

            BufferedReader inFromClient =
                new BufferedReader(new
                    InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient =
                new DataOutputStream(connectionSocket.getOutputStream());
```

```

        clientSentence = inFromClient.readLine();

        capitalizedSentence = clientSentence.toUpperCase() + '\n';

        outToClient.writeBytes(capitalizedSentence);
    }
}

```

TCP Client

```

import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("server.Prudhvi.ch-geni-
net.instageni.iu.edu", 6789);

        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer =
            new BufferedReader(new
                InputStreamReader(clientSocket.getInputStream()));

        sentence = inFromUser.readLine();
    }
}

```

```

        outToServer.writeBytes(sentence + '\n');

        modifiedSentence = inFromServer.readLine();

        System.out.println("FROM SERVER: " + modifiedSentence);

        clientSocket.close();
    }
}

```

Output:-

```

tgwgd@client: ~
tgwgd@client:~/latest$ pwd
/users/tgwgd/latest
tgwgd@client:~/latest$ cd ..
tgwgd@client:~$ pwd
/users/tgwgd
tgwgd@client:~$ javac TCPClient.java
tgwgd@client:~$ java TCPClient
hi
FROM SERVER: HI
tgwgd@client:~$ █

tgwgd@server: ~
tgwgd@server:~/new$ ls
ChatServer.class  ChatServer.java  clientThread.class
tgwgd@server:~/new$ ls
ChatServer.class  ChatServer.java  clientThread.class
tgwgd@server:~/new$ pwd
/users/tgwgd/new
tgwgd@server:~/new$ cd ..
tgwgd@server:~$ pwd
/users/tgwgd
tgwgd@server:~$ ls
ChatServer.class      First.java          TCPServer2.java
ChatServer.java       new                 TCPServer.class
ChatServerThread.class phase2              TCPServer.java
ChatServerThread.java TCPServer2.class
tgwgd@server:~$ javac TCPServer.java
tgwgd@server:~$ java TCPServer
█

```

b) Server disconnects when client sends bye.

TCP Server

```
import java.io.*;
```

```
import java.net.*;
```

```
class TCPServer2 {
```

```
public static void main(String argv[]) throws Exception
{
    String clientSentence;
    String capitalizedSentence;

    ServerSocket welcomeSocket = new ServerSocket(6789);

    while(!capitalizedSentence.equals("BYE"))
    {

        Socket connectionSocket = welcomeSocket.accept();

        BufferedReader inFromClient =
            new BufferedReader(new
                InputStreamReader(connectionSocket.getInputStream()));
        DataOutputStream outToClient =
            new DataOutputStream(connectionSocket.getOutputStream());

        clientSentence = inFromClient.readLine();

        capitalizedSentence = clientSentence.toUpperCase() + '\n';

        outToClient.writeBytes(capitalizedSentence);
```

```

        connectionSocket.close();

    }

}

}

```

TCP Client:-

```

import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("server.Prudhvi.ch-geni-
net.instageni.iu.edu ", 6789);

        while(!capitalizedSentence.equals("BYE"))
        {

            DataOutputStream outToServer =
                new DataOutputStream(clientSocket.getOutputStream());
            BufferedReader inFromServer =
                new BufferedReader(new
                    InputStreamReader(clientSocket.getInputStream()));

            sentence = inFromUser.readLine();

```

```

        outToServer.writeBytes(sentence + '\n');

        modifiedSentence = inFromServer.readLine();

        System.out.println("FROM SERVER: " + modifiedSentence);
    }
    clientSocket.close();

}
}

```

Output:-



The image shows two terminal windows side-by-side. The left window is the server terminal, showing the output 'Hi..' and 'BYE' from the client, and the prompt 'tgwgd@server:~\$'. The right window is the client terminal, showing the output 'Hi..' and 'BYE' from the server, and the prompt 'tgwgd@client:~/new/updated\$'.

Part II.

Simple Chat socket program (50%)

Develop a simple chat program (similar to google hangout and skype chat), and show the screenshots of the execution of the below. Extend the first program to chat client-server program following these steps.

a) A chat server will accept a single client connection and display everything the client types. If the client user types 'quit', both client and server will end the program.

Sol:-

TCP Server:-

```
import java.io.*;
```

```
import java.net.*;
```

```
class TCPServer2 {
```

```
    public static void main(String argv[]) throws Exception
```

```
    {
```

```
        String clientSentence;
```

```
        String capitalizedSentence;
```

```
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

```
        while(!capitalizedSentence.equals("QUIT"))
```

```
        {
```

```
            Socket connectionSocket = welcomeSocket.accept();
```

```
            BufferedReader inFromClient =
```

```
                new BufferedReader(new
```

```
                    InputStreamReader(connectionSocket.getInputStream()));
```



```

        DataOutputStream outToClient =
            new DataOutputStream(connectionSocket.getOutputStream());

        clientSentence = inFromClient.readLine();

        capitalizedSentence = clientSentence.toUpperCase() + '\n';

        outToClient.writeBytes(capitalizedSentence);

        ServerSocket.close();

    }
}
}

```

TCPClient:-

```

import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("server.Prudhvi.ch-geni-
net.instageni.iu.edu ", 6789);
    }
}

```

```

        while(!capitalizedSentence.equals("QUIT"))
        {

DataOutputStream outToServer =
    new DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer =
    new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

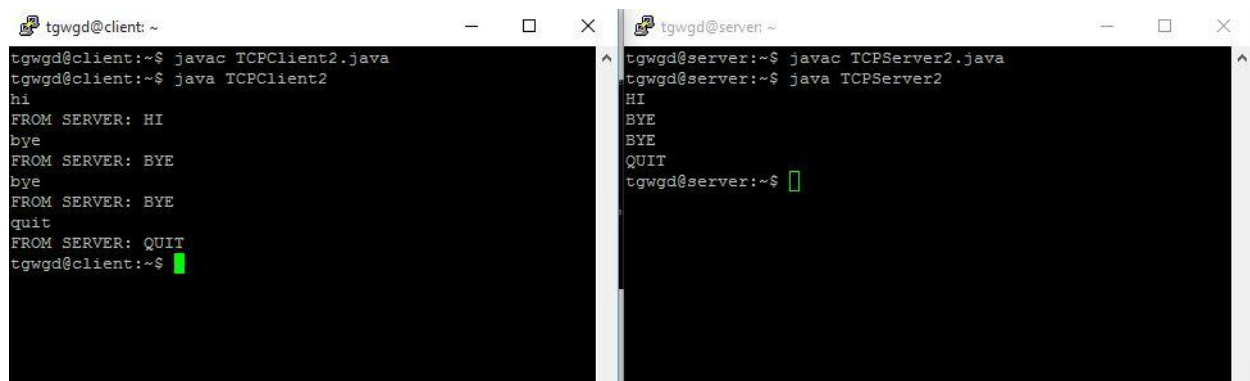
System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
}

```

Output:-



```

tgwgd@client: ~
tgwgd@client:~$ javac TCPClient2.java
tgwgd@client:~$ java TCPClient2
hi
FROM SERVER: HI
bye
FROM SERVER: BYE
bye
FROM SERVER: BYE
quit
FROM SERVER: QUIT
tgwgd@client:~$

tgwgd@server: ~
tgwgd@server:~$ javac TCPServer2.java
tgwgd@server:~$ java TCPServer2
HI
BYE
BYE
QUIT
tgwgd@server:~$

```

2

b) A server now remains 'open' for additional connection once a client quits. The server can handle at most one connection at a time.

Sol:-

Given in question that server at most handles one client at a time. Thus in the program the variable maxClients is initialized to 1 so that server handles one client at a time.

Program :

```
import java.io.*;
import java.io.PrintStream;
import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;

public class ChatServer {

    private static ServerSocket sSocket = null;
    private static Socket cSocket = null;
    private static final int maxClients = 1;
    private static final clientThread[] threads = new clientThread[maxClients];

    public static void main(String args[]) {
        int portNumber=0;
        if (args.length < 1) {
            System.out
```

```

        .println("Usage: java ChatServer portNumber\n");
        System.exit(0);
    } else {
        portNumber = Integer.valueOf(args[0]).intValue();
    }

    try {
        sSocket = new ServerSocket(portNumber);
    } catch (IOException e) {
        System.out.println(e);
    }

    while (true) {
        try {
            cSocket = sSocket.accept();
            int i = 0;
            for (i = 0; i < 1; i++) {
                if (threads[i] == null) {
                    (threads[i] = new clientThread(cSocket, threads)).start();
                    break;
                }
            }
        }

        if (i == maxClients) {
            PrintStream os = new PrintStream(cSocket.getOutputStream());
            os.println("limit exceeded.");
            os.close();
        }
    }
}

```

```

        cSocket.close();
    }
} catch (IOException e) {
    System.out.println(e);
}
}
}
}

```

```

class clientThread extends Thread {

```

```

    private BufferedReader inData = null;
    private PrintStream outData = null;
    private Socket clientSocket = null;
    private final clientThread[] threads;
    private int maxClients;

```

```

    public clientThread(Socket clientSocket, clientThread[] threads) {
        this.clientSocket = clientSocket;
        this.threads = threads;
        maxClients = threads.length;
    }

```

```

    public void run() {
        int maxClients = this.maxClients;

```

```

clientThread[] threads = this.threads;

try {
    inData = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
    outData = new PrintStream(clientSocket.getOutputStream());
    outData.println("Please Input your name:");
    String name = inData.readLine().trim();
    for (int i = 0; i < maxClients; i++) {
        if (threads[i] != null && threads[i] != this) {
            threads[i].outData.println("New connection from " + name);
        }
    }
    while (true) {
        String line = inData.readLine();
        System.out.println(line);
        if (line.startsWith("quit")) {
            break;
        }
        for (int i = 0; i < maxClients; i++) {
            if (threads[i] != null) {
                threads[i].outData.println(name + ":" + line);
            }
        }
    }
    for (int i = 0; i < maxClients; i++) {

```

```
        if (threads[i] != null && threads[i] != this) {
            threads[i].outData.println("connection closed!");
        }
    }
    outData.println("Bye " + name);
    for (int i = 0; i < maxClients; i++) {
        if (threads[i] == this) {
            threads[i] = null;
        }
    }
    inData.close();
    outData.close();
    clientSocket.close();
} catch (IOException e) {
}
}
}
```

```
tgwgd@client: ~/latest
tgwgd@client:~/latest$ pwd
/users/tgwgd/latest
tgwgd@client:~/latest$ ls
ChatClient.class  ChatClient.java
tgwgd@client:~/latest$ java ChatClient.java
Error: Could not find or load main class ChatClient
tgwgd@client:~/latest$ javac ChatClient.java
tgwgd@client:~/latest$ java ChatClient server.Prudhvi.ch-geni-net.instageni.iu.edu 2580
Please Input your name:
prudhvi
New connection from snehal
connection closed!

tgwgd@client: ~/new/updated
tgwgd@client:~$ pwd
/users/tgwgd
tgwgd@client:~$ cd new
tgwgd@client:~/new$ ls
ChatClient.class  ChatClient.java  updated
tgwgd@client:~/new$ cd updated
tgwgd@client:~/new/updated$ javac ChatClient.java
tgwgd@client:~/new/updated$ java ChatClient
Usage: java ChatServer hostname portNumber
tgwgd@client:~/new/updated$ java ChatClient server.Prudhvi.ch-geni-net.instageni.iu.edu 2580
Exception host not found
tgwgd@client:~/new/updated$ java ChatClient server.Prudhvi.ch-geni-net.instageni.iu.edu 2580
Please Input your name:
New connection from prudhvi
snehal
quit
Bye snehal

tgwgd@server: ~/new
tgwgd@server:~/new$ java ChatServer 2580
```

2)

c) A server now can handle multiple clients at the same time. The output from all the connected clients will appear on the server's screen.

Program:

```
import java.io.*;
import java.io.PrintStream;
import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;

public class ChatServer {

    private static ServerSocket sSocket = null;
    private static Socket cSocket = null;
    private static final int maxClients = 10;
```



```

private static final clientThread[] threads = new clientThread[maxClients];

public static void main(String args[]) {
    int portNumber=0;
    if (args.length < 1) {
        System.out
        .println("Usage: java ChatServer portNumber\n");
        System.exit(0);
    } else {
        portNumber = Integer.valueOf(args[0]).intValue();
    }

    try {
        sSocket = new ServerSocket(portNumber);
    } catch (IOException e) {
        System.out.println(e);
    }

    while (true) {
        try {
            cSocket = sSocket.accept();
            int i = 0;
            for (i = 0; i < 10; i++) {
                if (threads[i] == null) {
                    (threads[i] = new clientThread(cSocket, threads)).start();
                    break;
                }
            }
        }
    }
}

```

```

    }
}
if (i == maxClients) {
    PrintStream os = new PrintStream(cSocket.getOutputStream());
    os.println("limit exceeded.");
    os.close();
    cSocket.close();
}
} catch (IOException e) {
    System.out.println(e);
}
}
}
}

```

```

class clientThread extends Thread {

    private BufferedReader inData = null;
    private PrintStream outData = null;
    private Socket clientSocket = null;
    private final clientThread[] threads;
    private int maxClients;

    public clientThread(Socket clientSocket, clientThread[] threads) {
        this.clientSocket = clientSocket;
    }
}

```

```

        this.threads = threads;
        maxClients = threads.length;
    }

    public void run() {
        int maxClients = this.maxClients;
        clientThread[] threads = this.threads;

        try {
            inData = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            outData = new PrintStream(clientSocket.getOutputStream());
            outData.println("Please Input your name:");
            String name = inData.readLine().trim();
            for (int i = 0; i < maxClients; i++) {
                if (threads[i] != null && threads[i] != this) {
                    threads[i].outData.println("New connection from "+ name);
                }
            }
        } while (true) {
            String line = inData.readLine();
            System.out.println(line);
            if (line.startsWith("quit")) {
                break;
            }
            for (int i = 0; i < maxClients; i++) {

```

```

        if (threads[i] != null) {
            threads[i].outData.println(name + ":" + line);
        }
    }
}

for (int i = 0; i < maxClients; i++) {
    if (threads[i] != null && threads[i] != this) {
        threads[i].outData.println("connection closed!");
    }
}

outData.println("Bye " + name);
for (int i = 0; i < maxClients; i++) {
    if (threads[i] == this) {
        threads[i] = null;
    }
}

inData.close();
outData.close();
clientSocket.close();
} catch (IOException e) {
}
}
}

```



```

    public static void main(String args[]) {
        int portNumber=0;
        if (args.length < 1) {
            System.out
                .println("Usage: java ChatServer portNumber\n");
            System.exit(0);
        } else {
            portNumber = Integer.valueOf(args[0]).intValue();
        }

        try {
            sSocket = new ServerSocket(portNumber);
        } catch (IOException e) {
            System.out.println(e);
        }

        while (true) {
            try {
                cSocket = sSocket.accept();
                int i = 0;
                for (i = 0; i < 10; i++) {
                    if (threads[i] == null) {
                        (threads[i] = new clientThread(cSocket, threads)).start();
                        break;
                    }
                }
            }
        }
    }

```

```

    }
    if (i == maxClients) {
        PrintStream os = new PrintStream(cSocket.getOutputStream());
        os.println("limit exceeded.");
        os.close();
        cSocket.close();
    }
} catch (IOException e) {
    System.out.println(e);
}
}
}
}
}

```

```

class clientThread extends Thread {

```

```

    private BufferedReader inData = null;
    private PrintStream outData = null;
    private Socket clientSocket = null;
    private final clientThread[] threads;
    private int maxClients;

```

```

    public clientThread(Socket clientSocket, clientThread[] threads) {
        this.clientSocket = clientSocket;
        this.threads = threads;
    }

```

```
    maxClients = threads.length;
}
```

```
public void run() {
    int maxClients = this.maxClients;
    clientThread[] threads = this.threads;

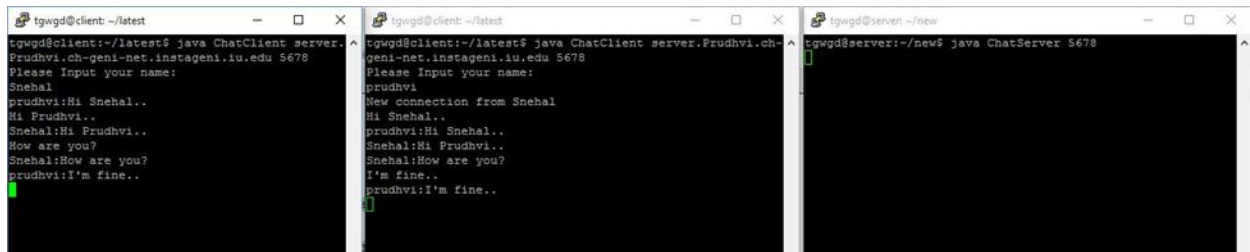
    try {
        inData = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        outData = new PrintStream(clientSocket.getOutputStream());
        outData.println("Please Input your name:");
        String name = inData.readLine().trim();
        for (int i = 0; i < maxClients; i++) {
            if (threads[i] != null && threads[i] != this) {
                threads[i].outData.println("New connection from " + name);
            }
        }
        while (true) {
            String line = inData.readLine();
            if (line.startsWith("quit")) {
                break;
            }
            for (int i = 0; i < maxClients; i++) {
                if (threads[i] != null) {
                    threads[i].outData.println(name + ":" + line);
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



```

        }
    }
}
for (int i = 0; i < maxClients; i++) {
    if (threads[i] != null && threads[i] != this) {
        threads[i].outData.println("connection closed!");
    }
}
outData.println("Bye " + name);
for (int i = 0; i < maxClients; i++) {
    if (threads[i] == this) {
        threads[i] = null;
    }
}
inData.close();
outData.close();
clientSocket.close();
} catch (IOException e) {
}
}
}

```



The image shows three terminal windows side-by-side, illustrating a chat application. The left window is the client terminal (tgwgd@client: ~/latest) running 'java ChatClient server.geni-net.instageni.iu.edu 5678'. It shows a conversation where Snehal connects, says 'Hi Prudhvi..', 'How are you?', and 'I'm fine..'. The middle window is the server terminal (tgwgd@server: ~/new) running 'java ChatServer 5678'. It shows the server receiving the connection and the messages from Snehal. The right window is another client terminal (tgwgd@client: ~/latest) running 'java ChatClient server.Prudhvi.ch-geni-net.instageni.iu.edu 5678', which shows the same conversation from Prudhvi's perspective.

```
tgwgd@client: ~/latest
tgwgd@client:~/latest$ java ChatClient server.
Prudhvi.ch-geni-net.instageni.iu.edu 5678
Please Input your name:
Snehal
prudhvi:Hi Snehal..
Hi Prudhvi..
Snehal:Hi Prudhvi..
How are you?
Snehal:How are you?
prudhvi:I'm fine..

tgwgd@client:~/latest$ java ChatClient server.Prudhvi.ch-
geni-net.instageni.iu.edu 5678
Please Input your name:
prudhvi
New connection from Snehal
Hi Snehal..
prudhvi:Hi Snehal..
Snehal:Hi Prudhvi..
Snehal:How are you?
I'm fine..
prudhvi:I'm fine..

tgwgd@server:~/new$ java ChatServer 5678
```

Team Contributions:-

- 1) **Trivikram Reddy: 16212362**
 - Created Slice
 - Generated ssh key
 - Downloaded putty
 - Executed phase 1
 - Project documentation
- 2) **Snehal Vantashala:16206488**
 - Putty keygen downloaded
 - Created private key
 - Login server and client machine
 - Executed phase 1
- 3) **Prudhvi Raj: 16208160**
 - Initiaized nodes
 - Phase 1 execution on server
 - Client server chat using threads
 - Phase 2 execution on server