Name: Keegan Vaz
SEC: TE CMPN-B
Roll No.: 28
PID: 192120

EXPERIMENT NO.6

AIM: Java program for Socket Programming

THEORY:

Java Socket Programming

- Java Socket programming is used for communication between the applications running on different JRE.
- Java Socket programming can be connection-oriented or connection-less.
- Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while another socket reaches out to the other to form a connection. Server forms the listener socket while the client reaches out to the server.

Classes of Socket Programming: Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Important methods of Socket class:

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

Important methods of ServerSocket class:

Method	Description

1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket

The client in socket programming must know two information:

a. IP Address of Server, and
b. Port number.

IP address:

An Internet Protocol address (IP address) is a numerical label such as 192.0.2.1 that is connected to a computer network that uses the Internet Protocol for communication. An IP address serves two main functions: host or network interface identification and location addressing.

Internet Protocol version 4 (IPv4) defines an IP address as a 32-bit number. However, because of the growth of the Internet and the depletion of available IPv4 addresses, a new version of IP (IPv6), using 128 bits for the IP address, was standardized in 1998. IPv6 deployment has been ongoing since the mid-2000s.

Port No:

In computer networking, a port is a communication endpoint. At the software level, within an operating system, a port is a logical construct that identifies a specific process or a type of network service. A port is identified for each transport protocol and address combination by a 16-bit unsigned number, known as the port number. The most common transport protocols that use port numbers are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

A port number is always associated with an IP address of a host and the type of transport protocol used for communication. It completes the destination or origination network address of a message. Specific port numbers are reserved to identify specific services so that an arriving packet can be easily forwarded to a running application. For this purpose, port numbers lower than 1024 identify the historically most commonly used services and are called the well-known port numbers. Higher-numbered ports are available for general use by applications and are known as ephemeral ports.

Different types of Port numbers:

Number	Assignment
20	File Transfer Protocol (FTP) Data Transfer
21	File Transfer Protocol (FTP) Command Control
22	Secure Shell (SSH) Secure Login

23	Telnet remote login service, unencrypted text messages
25	Simple Mail Transfer Protocol (SMTP) email delivery
80	Hypertext Transfer Protocol (HTTP) used in the World Wide Web
443	HTTP Secure (HTTPS) HTTP over TLS/SSL

The dynamic or private ports are those from 49152 through 65535. One common use for this range is for ephemeral ports.

Here, we are going to make two-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Then the server can reply back and the client will read the message and print it. Here, two classes are being used: Socket and ServerSocket.

The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.

Advantages:

1)Sockets are flexible and sufficient.

2)Efficient socket based programming can be easily implemented for general communications.

3)It causes low network traffic.

Disadvantages:

1)Socket based communications allows only to send packets of raw data between applications.

2)Both the client-side and server-side have to provide mechanisms to make the data useful in any way.

#Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

#ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

```
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection and waits for the client
```

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

```
Socket s=new Socket("localhost",6666);
```

Code:

Client_1:

```
import java.net.*;
import java.io.*;
class MyClient_1{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",6666);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}
dout.close();
s.close();
}}
```

Client_2:

```
import java.net.*;
import java.io.*;
class MyClient_2{ public static void main(String args[])throws Exception{
Socket sa=new Socket("localhost",6666);
DataInputStream dins=new DataInputStream(sa.getInputStream());
DataOutputStream douts=new DataOutputStream(sa.getOutputStream());
```

```
BufferedReader brs=new BufferedReader(new InputStreamReader(System.in));

String str3="",str4="";
while(!str3.equals("stop")){
str3=brs.readLine();
douts.writeUTF(str3);
douts.flush();
str4=dins.readUTF();
System.out.println("Server says: "+str4);
}
douts.close();
sa.close();
}}
```

Server:

```
import java.net.*;
import java.io.*;
class MyServer1{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();
Socket sa=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
DataInputStream dins=new DataInputStream(sa.getInputStream());
DataOutputStream douts=new DataOutputStream(sa.getOutputStream());
BufferedReader brs=new BufferedReader(new InputStreamReader(System.in));
String str="",str2="",str3="",str4="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client1 says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
```

```
str3=dins.readUTF();
System.out.println("client2 says: "+str3);
str4=brs.readLine();
douts.writeUTF(str4);
douts.flush();
}
din.close();
s.close();
dins.close();
sa.close();
ss.close();
}}
```

Output:

```
C:\Users\keega\Desktop\Keegan\SFIT\SEM 5\Computer Network>javac MyClient_1.java
C:\Users\keega\Desktop\Keegan\SFIT\SEM 5\Computer Network>java MyClient_1
Hello Server, I am Client 1
Server says: Hello Client 1
```

```
C:\Users\keega\Desktop\Keegan\SFIT\SEM 5\Computer Network>javac MyClient_2.java
C:\Users\keega\Desktop\Keegan\SFIT\SEM 5\Computer Network>java MyClient_2
Hello Server, I am Client 2
Server says: Hello Client 2
```

```
C:\Users\keega\Desktop\Keegan\SFIT\SEM 5\Computer Network>javac MyServer1.java
C:\Users\keega\Desktop\Keegan\SFIT\SEM 5\Computer Network>java MyServer1
client1 says: Hello Server, I am Client 1
Hello Client 1
client2 says: Hello Server, I am Client 2
Hello Client 2
```

Conclusion: We learnt how to use Java Socket programming to connect two different applications and to be able to communicate. We learnt how to create the server and the client using serversocket and socket classes respectively. Hence we learnt how efficient socket based programming can be easily implemented for general communications.