TCP Sockets

- A communication link created via TCP/IP sockets is a connection-oriented link. This means that the connection between the server and the client remains open throughout the duration of the dialogue between the two and is only broken (under normal circumstances) when one end of the dialogue formally terminates the exchanges (via an agreed protocol).
- Since there are two separate types of a process involved (a client and a server); we shall examine them separately, taking the server first.

A. Setting up a Server process requires five steps:

1. **Create a ServerSocket object**
   - The ServerSocket constructor requires a port number (1024 – 65535, for non – reserved ones) as an argument. For example:

     ```
     ServerSocket serverSocket = new ServerSocket(1234);
     ```
   - The server will wait ('listen for') a connection from a client on port 1234.
2. **Put the server into a waiting state**
   - The server waits indefinitely ('blocks') for a client to connect. It does this by calling the method accept of the class ServerSocket, which returns a Socket object. For example:

     ```
     Socket link = serverSocket.accept();
     ```
3. **Set up input and output streams**
   - The methods getInputStream and getOutputStream of the class Socket are used to get references to streams associated with the Socket object returned in Step 2. These streams will be used for communication with the client that has just made a connection.
   - For a non – GUI application, we can wrap a Scanner object around the InputStream object returned by the method getInputStream, in order to obtain string – oriented input (just as we would do with input from standard input stream, System.in). For example:

     ```
     Scanner input = new Scanner(link.getInputStream());
     ```
   - Similarly, we can wrap a PrintWriter object around the OutputStream object returned by the method getOutputStream. Supplying the PrintWriter constructor with a second argument of true will cause the output buffer to the flushed for every call of the println method (which is usually desirable). For example:

     ```
     PrintWriter output = new
                          PrintWriter(link.getOutputStream(), true);
     ```
4. **Send and receive data**
   - Having set up our Scanner and PrintWriter objects, send and receiving data is very straight forward. We simply use method nextLine for receiving data and method println for sending data, just as we might do for console I/O. For example:

     ```
     output.println("Awaiting data ... ");
     String input = input.nextLine();
     ```

5. **Close the connection (after completion of the dialogue)**
    - This is achieved via method close of the class Socket. For example:

```
link.close();
```

```
//Server that echoes back clients messages
//At the end of a dialogue, sends a message indicating
//the number of messages received. Uses TCP

import java.io.*;
import java.net.*;
import java.util.*;

public class TCPEchoServer{
	private static ServerSocket serverSocket;
	private static final int PORT = 1234;

	public static void main(String [] args){
		System.out.println("Opening port ... \n");

		try{
			serverSocket = new ServerSocket(PORT);			//Step 1
		}
		catch(IOException ioEx){
			System.out.println("Unable to attach to port!");
			System.exit(1);
		}

		do{
			handleClient();
		}while(true);
	}

	private static void handleClient(){
		Socket link = null;						//Step 2

		try{
			link = serverSocket.accept();				//Step 2

			Scanner input = new
					Scanner(link.getInputStream());		//Step 3
			PrintWriter output = new
				PrintWriter(link.getOutputStream(), true);	//Step 3

			int numMessages = 0;
			String message = input.nextLine();			//Step 4

			while(!message.equals("***CLOSE***")){
				System.out.println("Message received.");
				numMessages++;
				output.println("Message "
					+ numMessages + ": " + message);		//Step 4
				message = input.nextLine();
			}

			output.println(numMessages
```

```
                                        + " messages received.");          //Step 4
            }
            catch(IOException ioEx){
                ioEx.printStackTrace();
            }
            finally{
                try{
                    System.out.println("\n * Closing connection ... *");
                    link.close();
                }
                catch(IOException ioEx){
                    System.out.println("Unable to disconnect !");
                    System.exit(1);
                }
            }
        }//end of handleClient
}
```

B. Setting up the corresponding client process involves four steps:

1. **Establish a connection to the server**
   - We create a Socket object, supplying its constructor with the following two arguments:
     i. server's IP address (of type InetAddress), and
     ii. the appropriate port number for the service.

     (The port number for the server and client programs must be the same, of course!).
   - For simplicity sake, the client and server are placed on the same host, which then will allow is to retrieve the IP address by calling the static method getLocalHost of class InetAddress. For example:

     ```
     Socket link = new Socket(InetAddress.getLocalHost(), 1234);
     ```
2. **Set up input and output streams**
   - These are set up in exactly the same way as the server streams were set up (by calling methods getInputSream and getOutputStream of the Socket object that was created in Step2).
3. **Send and receive data**
   - The Scanner object at the client end will receive messages sent by PrintWriter object at the server end, while the PrintWriter object at the client end will send messages that are received by the Scanner object at the server end (using methods nextLine and println respectively).
4. **Close the connection**
   - This is exactly the same as for the server process (using method close of class Socket).

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

public class TCPEchoClient{
        private static InetAddress host;
```

```
        private static final int PORT = 1234;

        public static void main(String [] args){
                try{
                        host = InetAddress.getLocalHost();
                }
                catch(UnknownHostException unEx){
                        System.out.println("Host ID not found!");
                        System.exit(1);
                }
                accessServer();
        }
        private static void accessServer(){
                Socket link = null;

                try{
                        link = new Socket(host, PORT);                          //Step 1

                        Scanner input = new
                                        Scanner(link.getInputStream());         //Step 2
                        PrintWriter output = new
                                PrintWriter(link.getOutputStream(), true);      //Step 2

                        //set up a stream for keyboard entry
                        Scanner userEntry = new Scanner(System.in);
                        String message, response;

                        do{
                                System.out.println("Enter a message: ");
                                message = userEntry.nextLine();

                                output.println(message);                        //Step 3
                                response = input.nextLine();                    //Step 3

                                System.out.println("\n SERVER> " + response);
                        }while(!message.equals("***CLOSE***"));
                }
                catch(IOException ioEx){
                        ioEx.printStackTrace();
                }
                finally{
                        try{
                                System.out.println("\n Closing
                                                        connection .. *");      //Step 4
                                link.close();
                        }
                        catch(IOException ioEx){
                                System.out.println("Unable to disconnect !");
                                System.exit(1);
                        }
                }
        }

}
```