LAB SESSION 3

CONCURRENCY & PARALLELISM

ABSTRACT

To understand Socket Communication by implementing a server and a client & also with a Multithreaded Server handled via a Thread Implementing Client Manager

Cyril Naves & Ponathipan Jawahar

Exercise 3: Client/Server

1/ Write a class *Server* that:

- opens a socket on a port number given as an argument,
- waits for a connection,
- prints a message when the connection is accepted.

2/ Try to connect to the server using i/ the *Telnet* command and ii/ using a browser

3/ Write a class *Client* that connects to the server and prints a message when it is done. The address and port of the server are given as an argument.

4/ Improve your application so that the client sends successively 5 messages and receives an acknowledgment including the original message from the server. When the client emits the keyword **stop**, the connection are stopped at both end and the program is shut down.

Code:
Client.java:

```
package masterInt.CandP.exo3;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

/*
 * This is a client thread, the user can send message to server through this client.
 */
public class Client {

private final String host;

private final int port;

public Client(String serverHost, int serverPort) {
	this.host = serverHost;
	this.port = serverPort;
}

public void execute() {
	// initialize the resources
	Socket socket = null;
	try {
		socket = new Socket(this.host, this.port);
	} catch (UnknownHostException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}

	String line;
	BufferedReader reader = null;
	PrintWriter printer = null;
	BufferedReader socketReader = null;

	try {
		System.out.println("Client built up...");
		reader = new BufferedReader(new InputStreamReader(System.in));
		printer = new PrintWriter(socket.getOutputStream());
```

```
                socketReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        } catch (IOException e) {
                e.printStackTrace();
                System.out.println("reader or printer initialize failed.");
        }
        while (true) {
                try {
                        //wait for the user input from console and send it to server
                        line = reader.readLine();
                        printer.println(line);
                        printer.flush();
                        System.out.println("Client: message has been sent.");
                        //wait for the server ack
                        System.out
                                        .println(String.format("Client: receive server acknowledgement <%s>",
socketReader.readLine()));

                        if (line.equals("stop")) {
                                break;
                        }

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        //if the client send a stop message, release the resources
        try {
                reader.close();
                printer.close();
                socketReader.close();
                socket.close();
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        System.out.println("client closed...");
}

public static void main(String[] args) {
        if (args.length != 2) {
                System.err.println("usage: java " + Client.class.getCanonicalName() + " serverHost serverPort");
                System.exit(1);
        }

        try {
                new Client(args[0], Integer.parseInt(args[1])).execute();
        } catch (NumberFormatException e) {
                throw new IllegalArgumentException("Invalid port number: " + args[0]);
        }
}

}
```

Server.java:

```
package masterInt.CandP.exo3;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;

/*
 * This a single thread server.
 * Question 2
 * 1)Connect the server by Telnet:
 *   Type the command : telnet 127.0.0.1 55555 in windows cmd
 * 2)Connect the server by browser:
 *   Open any browser and enter 127.0.0.1:55555 as address
 */
public class Server {

// port number to listen on
private final int port;

public Server(int port) {
	this.port = port;
}

public void execute() {
	// ...
	System.out.println("begin server initializing...");
	ServerSocket serverSocket = null;
	Socket socket = null;

	//initialize the resources
	try {
		serverSocket = new ServerSocket(this.port);
		socket = serverSocket.accept();
	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}

	String line;
	//the index is responsible for counting the number of messages
	int index = 0;

	//read inputstream from socket
	BufferedReader reader = null;
	//write to socket outputstream
	PrintWriter printer = null;
	try {
		reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
		printer = new PrintWriter(socket.getOutputStream());
	} catch (IOException e) {
		e.printStackTrace();
		System.out.println("reader or printer initialize failed.");
	}
	System.out.println("server initialize successfully...");
	while (true) {
		try {
			//read message from client and return ack to client
			line = reader.readLine();
			System.out.println(String.format("Server: Message received from client: %s", line));
			printer.println(String.format("Message %s : %s", index++, line));
			printer.flush();
			if (line.equals("stop")) {
```

```
                                        //when receive a "stop" message, the server will release all the resources and shutdown
                                        printer.close();
                                        reader.close();
                                        socket.close();
                                        serverSocket.close();
                                        break;
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
                System.out.println("server closed...");
        }

        public static void main(String[] args) {
                if (args.length != 1) {
                        System.err.println("usage: java " + Server.class.getCanonicalName() + " <serverPort>");
                        System.exit(1);
                }

                // On unix systems you can check that the server is running
                // by executing the following command:
                // lsof -Pi | grep 9999
                try {
                        new Server(Integer.parseInt(args[0])).execute();
                } catch (NumberFormatException e) {
                        throw new IllegalArgumentException("Invalid port number: " + args[0]);
                }
        }

        }
```

Output:

```
Client:
Client built up...
first message
Client: message has been sent.
Client: receive server acknowledgement <Message 0 : first message>
second message
Client: message has been sent.
Client: receive server acknowledgement <Message 1 : second message>
third message
Client: message has been sent.
Client: receive server acknowledgement <Message 2 : third message>
stop
Client: message has been sent.
Client: receive server acknowledgement <Message 3 : stop>
client closed...

Server:
begin server initializing...
server initialize successfully...
Server: Message received from client: first message
Server: Message received from client: second message
Server: Message received from client: third message
Server: Message received from client: stop
server closed...
```

Summary of the Client and Server Class Implementation:

1) Client Class contains the Buffered Reader and PrintWriter Objects which are created for input and output functionalities from the socket.
2) Then the client is made to wait in while loop for incoming requests and outgoing responses to the server.
3) Server Class utilizes the ServerSocket Library from java.net package to create the connection on a specific port
4) Then Buffered Reader and PrintWriter is used to obtain the required input and output stream from the Socket.
5) Server is made to open for connections in a while loop until it receives the message of "stop" then socket is closed with the connection terminated.

Exercise 4: Multi-client

Modify the server part of the application written in Exercise 1 so that the server can accept many client connections.

1/When the server accepts a connection, it delegates the management of this connection to an instance of the runnable class *ClientManager*, and then wait again for connection.

2/The class *ClientManager* discusses with the associated client similarly to the server in Exercise 1.

Code:

ClientManager.java

```
package masterInt.CandP.exo4;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

/*
 * This class is responsible for managing the connection between server and client
 */
public class ClientManager implements Runnable
{

   private final Socket socket;
   private int id;


   //construct function, obtain the socket accepted by multithreadedserver
   public ClientManager(Socket socket,int id)
   {
      this.socket = socket;
      this.id=id;
   }

   public void manage() throws IOException
   {
      //...
   }

   /**
    * {@inheritDoc}
    */
   @Override
   public void run()
   {
      //similar to a single thread server, processing the message sent from the connected client
            String line;
            int index = 0;

            BufferedReader reader = null;
            PrintWriter printer = null;
            try {
                  reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                  printer = new PrintWriter(socket.getOutputStream());
            } catch (IOException e) {
                  e.printStackTrace();
                  System.out.println("reader or printer initialize failed.");
            }
            while(true)
            {
                  try {
                        line = reader.readLine();
```

```
                System.out.println(String.format("Server: Message received from client %s : %s",this.id, line));
                printer.println(String.format("Message %s from client %s : %s", index++,this.id, line));
                printer.flush();
                if (line.equals("stop")) {
                        printer.close();
                        reader.close();
                        socket.close();
                        break;
                }
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        System.out.println("client "+ this.id +" closed...");
    }

}
```

MultiThreadedServer.java:

```
package masterInt.CandP.exo4;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/*
 * This is a multi-threaded server, the main thread is responsible for monitoring client request
 */
public class MultithreadedServer
{

    // port number to listen on
    private final int port;

    public MultithreadedServer(int port)
    {
        this.port = port;
    }

public void execute()
    {
        //...
        System.out.println("begin server initializing...");
        ServerSocket serverSocket = null;
        try {
                serverSocket = new ServerSocket(this.port);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        System.out.println("server initialize successfully...");
        int i=0;
        while(true)
        {
                //wait for the client request, and delegate it to a clientManager thread
                Socket socket = null;
                try {
                        socket = serverSocket.accept();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                Thread t = new Thread(new ClientManager(socket,i));
                t.start();
                System.out.println("Client "+ i++ + " connected.");
```

```
            }
    }

    public static void main(String[] args)
    {
        if (args.length != 1) {
            System.err.println("usage: java "
                    + MultithreadedServer.class.getCanonicalName()
                    + " serverPort");
            System.exit(1);
        }

        // On unix systems you can check that the server is running
        // by executing the following command:
        // lsof -Pi | grep 9999
        try {
            new MultithreadedServer(Integer.parseInt(args[0])).execute();
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("Invalid port number: "
                    + args[0]);
        }
    }

}
```

Output:

```
Client1:
Client built up...
second message
Client: message has been sent.
Client: receive server acknowledgement <Message 0 from client 0 : second message>
stop
Client: message has been sent.
Client: receive server acknowledgement <Message 1 from client 0 : stop>
client closed...

Client2:
Client built up...
second message
Client: message has been sent.
Client: receive server acknowledgement <Message 0 from client 0 : second message>
stop
Client: message has been sent.
Client: receive server acknowledgement <Message 1 from client 0 : stop>
client closed...

Client3:
Client built up...
second message
Client: message has been sent.
Client: receive server acknowledgement <Message 0 from client 0 : second message>
stop
Client: message has been sent.
Client: receive server acknowledgement <Message 1 from client 0 : stop>
client closed...
```

Multithreaded Server:

```
begin server initializing...
server initialize successfully...
Client 0 connected.
Client 1 connected.
Client 2 connected.
Server: Message received from client 2 : first message
Server: Message received from client 0 : second message
Server: Message received from client 1 : third message
Server: Message received from client 2 : stop
client 2 closed...
```

Server: Message received from client 0 : stop
client 0 closed...
Server: Message received from client 1 : stop
client 1 closed...

Summary of the Client Manager and MultiThreaded Server Class Implementation:

1) MultiThreaded Server is used to create a ServerSocket Library instantiation on a specific port in the localhost.
2) Then it is made to wait on a while loop for incoming connection in which after each connection established with the client a new thread is spawned to create a client manager based on the incoming connections
3) The Client Manager implements Runnable Interface where run() method is overridden to create the BufferedReader and PrintWriter to get the Input and Output Stream from the socket.
4) The delegation of the connection Socket to the ClientManager is handled effectively using thread and it print the incoming and outgoing response.
5) ClientManager is made to accept the incoming messages until the "stop" message and then finally connection of the socket is closed.