```
//---------------------------------UDPServer.java
//home/tovantran/Ctest/UDPServer.java --> 2015-03-14 by ./.tv  owner: tovantran

//https://systembash.com/a-simple-java-udp-server-and-udp-client/
//ip=localhost port=9876
//package socket.udpserver;

//import java.io.*;
import java.net.*;

class UDPServer
{
        public static void main(String args[]) throws Exception
        {
                DatagramSocket serverSocket = new DatagramSocket(9876);
                byte[] receiveData = new byte[1024];
                byte[] sendData = new byte[1024];
                while(true)
                {
                        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                        serverSocket.receive(receivePacket); //Wait here
                        String sentence = new String( receivePacket.getData());
                        // convert byte to String
                        System.out.println("RECEIVED: " + sentence);
                        InetAddress IPAddress = receivePacket.getAddress();
                        int port = receivePacket.getPort();
                        String capitalizedSentence = sentence.toUpperCase();
                        sendData = capitalizedSentence.getBytes();
                        DatagramPacket sendPacket =
                                new DatagramPacket(sendData, sendData.length, IPAddress, port);
                        serverSocket.send(sendPacket);
                }
        }
}
//---------------------------------UDPClient.java
//home/tovantran/Ctest/UDPClient.java --> 2015-03-14 by ./.tv  owner: tovantran

//https://systembash.com/a-simple-java-udp-server-and-udp-client/
//ip=localhost port=9876

//package socket.udpclient;

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

class UDPClient
{
        public static void main(String args[]) throws Exception
        {
                BufferedReader inFromUser =
```

```
                        new BufferedReader(new InputStreamReader(System.in));
                DatagramSocket clientSocket = new DatagramSocket(); //Socket
                InetAddress IPAddress = InetAddress.getByName("localhost");
                byte[] sendData = new byte[1024];
                byte[] receiveData = new byte[1024];
                for (int i = 0; i < sendData.length; i++){
                        sendData[i] = 0x41;
                }
                String sentence = inFromUser.readLine();
                sendData = sentence.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
                clientSocket.send(sendPacket); //Send Packet
                DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                clientSocket.receive(receivePacket);
                String modifiedSentence = new String(receivePacket.getData());
                System.out.println("FROM UDP SERVER:" + modifiedSentence);
                clientSocket.close();
        }
}

//---------------------------------TCPServer.java
//home/tovantran/Ctest/TCPServer.java --> 2015-03-14 by ./.tv  owner: tovantran

//https://systembash.com/a-simple-java-tcp-server-and-tcp-client/
//package mythread.tcpserver;

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);  // ServerSocket only port
                while(true)
                {
                        Socket connectionSocket = welcomeSocket.accept();
                        BufferedReader inFromClient =     //Connect & Read
                                new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
                        DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
                        clientSentence = inFromClient.readLine();
                        System.out.println("Received: " + clientSentence);
                        capitalizedSentence = clientSentence.toUpperCase() + '\n';
                        outToClient.writeBytes(capitalizedSentence); //Write
                }
        }
}
```

```
//---------------------------------TCPClient.java
//home/tovantran/Ctest/TCPClient.java --> 2015-03-14 by ./.tv  owner: tovantran

//https://systembash.com/a-simple-java-tcp-server-and-tcp-client/
//package mythread.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("localhost", 6789); //Socket(client) host and port
                DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream()); //Write & Read
                BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))
                sentence = inFromUser.readLine();
                outToServer.writeBytes(sentence + '\n');
                modifiedSentence = inFromServer.readLine();
                System.out.println("FROM TCP SERVER: " + modifiedSentence);
                clientSocket.close();
        }
}
//---------------------------------AppSingleThreadedServer.java
//package mythread.tcp.runnable;

import java.net.ServerSocket;
import java.net.Socket;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;

class SingleThreadedServer implements Runnable{

        protected int          serverPort   = 6789;
        protected ServerSocket serverSocket = null;
        protected boolean      isStopped    = false;
        protected Thread       runningThread= null;

        public SingleThreadedServer(int port){
                this.serverPort = port;
        }

        public void run(){
                synchronized(this){
```

```
                this.runningThread = Thread.currentThread();
            }
            openServerSocket();

            while(! isStopped()){
                Socket clientSocket = null;
                try {
                    clientSocket = this.serverSocket.accept();
                } catch (IOException e) {
                    if(isStopped()) {
                        System.out.println("Server Stopped.") ;
                        return;
                    }
                    throw new RuntimeException(
                                "Error accepting client connection", e);
                }
                try {
                    processClientRequest(clientSocket); //Server Loop R/W
                } catch (IOException e) {
                    //log exception and go on to next request.
                }
            }

            System.out.println("Server Stopped.");
        }

        private void processClientRequest(Socket clientSocket) throws IOException {
            String clientSentence;
            InputStream  input  = clientSocket.getInputStream();
            OutputStream output = clientSocket.getOutputStream();
            BufferedReader inFromClient =  //Second methods
                    new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            long time = System.currentTimeMillis();

            clientSentence = inFromClient.readLine();
            System.out.println("FROM CLIENT: " + clientSentence);

            output.write(("HTTP/1.1 200 OK\n\n<html><body>" +
                          "Singlethreaded Server: " +
                          time +
                          "</body></html>").getBytes());
            output.close();
            input.close();
            System.out.println("Request processed time: " + time);
        }

        private synchronized boolean isStopped() {
            return this.isStopped;
        }
```

```
        public synchronized void stop(){
                this.isStopped = true;
                try {
                        this.serverSocket.close();
                } catch (IOException e) {
                        throw new RuntimeException("Error closing server", e);
                }
        }

        private void openServerSocket() {
                try {
                        this.serverSocket = new ServerSocket(this.serverPort);
                } catch (IOException e) {
                        throw new RuntimeException("Cannot open port 6789", e);
                }
        }
}

public class AppSingleThreadedServer {
        public static void main(String args[]){
                SingleThreadedServer server = new SingleThreadedServer(6789);
                new Thread(server).start();

                try {
                        Thread.sleep(10 * 10000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                System.out.println("Stopping Server");
                server.stop();
        }
}


//---------------------------------AppMultiThreadedServer.java
//package mythread.tcp.multithreaded.runnable;

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

//import mythread.tcp.runnable.SingleThreadedServer;

class MultiThreadedServer implements Runnable{

        protected int          serverPort   = 6789;
        protected ServerSocket serverSocket = null;
        protected boolean      isStopped    = false;
        protected Thread       runningThread= null;
```

```
	public MultiThreadedServer(int port){
		this.serverPort = port;
	}

	public void run(){
		synchronized(this){
			this.runningThread = Thread.currentThread();
		}
		openServerSocket();
		while(! isStopped()){
			Socket clientSocket = null;
			try {
				clientSocket = this.serverSocket.accept();
			} catch (IOException e) {
				if(isStopped()) {
					System.out.println("Server Stopped.") ;
					return;
				}
				throw new RuntimeException(
						"Error accepting client connection", e);
			}
			new Thread(
					new WorkerRunnable(
						clientSocket, "Multithreaded Server")
				).start();
		}
		System.out.println("Server Stopped.") ;
	}


	private synchronized boolean isStopped() {
		return this.isStopped;
	}

	public synchronized void stop(){
		this.isStopped = true;
		try {
			this.serverSocket.close();
		} catch (IOException e) {
			throw new RuntimeException("Error closing server", e);
		}
	}

	private void openServerSocket() {
		try {
			this.serverSocket = new ServerSocket(this.serverPort);
		} catch (IOException e) {
			throw new RuntimeException("Cannot open port 6789", e);
		}
	}
```

```
}
public class AppMultiThreadedServer {
        public static void main(String args[]){
                SingleThreadedServer server = new SingleThreadedServer(6789);
                new Thread(server).start();

                try {
                        Thread.sleep(10 * 10000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                System.out.println("Stopping Server");
                server.stop();
        }
}

//---------------------------------AppPoolThreadedServer.java
//package mythread.threadpooled.runnable;

import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class ThreadPooledServer implements Runnable {

        protected int serverPort = 6789;
        protected ServerSocket serverSocket = null;
        protected boolean isStopped = false;
        protected Thread runningThread = null;
        protected ExecutorService threadPool = Executors.newFixedThreadPool(10);

        public ThreadPooledServer(int port) {
                this.serverPort = port;
        }

        public void run() {
                synchronized (this) {
                        this.runningThread = Thread.currentThread();
                }
                openServerSocket();
                while (!isStopped()) {
                        Socket clientSocket = null;
                        try {
                                clientSocket = this.serverSocket.accept();
                        } catch (IOException e) {
                                if (isStopped()) {
                                        System.out.println("Server Stopped.");
```

```
                                        break;
                                }
                                throw new RuntimeException("Error accepting client connection",
                                                e);
                        }
                        this.threadPool.execute(new WorkerRunnable(clientSocket,
                                                "Thread Pooled Server"));
                }
                this.threadPool.shutdown();
                System.out.println("Server Stopped.");
        }

        private synchronized boolean isStopped() {
                return this.isStopped;
        }

        public synchronized void stop() {
                this.isStopped = true;
                try {
                        this.serverSocket.close();
                } catch (IOException e) {
                        throw new RuntimeException("Error closing server", e);
                }
        }

        private void openServerSocket() {
                try {
                        this.serverSocket = new ServerSocket(this.serverPort);
                } catch (IOException e) {
                        throw new RuntimeException("Cannot open port 8080", e);
                }
        }
}

public class AppPoolThreadedServer {
        public static void main(String args[]) {
                ThreadPooledServer server = new ThreadPooledServer(6789);
                new Thread(server).start();

                try {
                        Thread.sleep(20 * 100000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                System.out.println("Stopping Server");
                server.stop();
        }
}
//---------------------------------WorkerRunnable.java
//home/tovantran/Ctest/JavaThread/WorkerRunnable.java --> 2015-03-17 by ./.tv  owner: tovantran
```

```
//package mythread.threadpooled.runnable;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;

public class WorkerRunnable implements Runnable {

        protected Socket clientSocket = null;
        protected String serverText = null;
        String clientSentence;

        public WorkerRunnable(Socket clientSocket, String serverText) {
                this.clientSocket = clientSocket;
                this.serverText = serverText;
        }

        public void run() {
                try {
                        InputStream input = clientSocket.getInputStream();
                        OutputStream output = clientSocket.getOutputStream();
                        BufferedReader inFromClient = //Second method: echo
                                new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                        long time = System.currentTimeMillis();

                        clientSentence = inFromClient.readLine();
                        System.out.println("FROM CLIENT: " + clientSentence);

                        output.write(("HTTP/1.1 200 OK\n\nWorkerRunnable: "
                                        + this.serverText + " - " + time + "").getBytes());
                        output.close();
                        input.close();
                        System.out.println("Request processed time: " + time);
                } catch (IOException e) {
                        // report exception somewhere.
                        e.printStackTrace();
                }
        }
}
//home/tovantran/Ctest/JavaThread/ServerEcho.java --> 2015-03-17 by ./.tv  owner: tovantran
//---------------------------------ServerEcho.java
//package mythread.tcp.runnable;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;


public class ServerEcho {

	public static void main(String[] args) {

		new ServerEcho().startServer();
	}

	public void startServer() {
		final ExecutorService clientProcessingPool = Executors.newFixedThreadPool(10);

		Runnable serverTask = new Runnable() {
			@Override
				public void run() {
					try {

						ServerSocket serverSocket = new ServerSocket(6789);
						System.out.println("Waiting for clients to connect...");
						while (true) {
							Socket clientSocket = serverSocket.accept();
							clientProcessingPool.submit(new ClientTask(clientSocket));
						}
					} catch (IOException e) {
						System.err.println("Unable to process client request");
						e.printStackTrace();
					}
				}
		};
		Thread serverThread = new Thread(serverTask);
		serverThread.start();

	}

	private class ClientTask implements Runnable {
		private final Socket clientSocket;
		String clientSentence;
		String capitalizedSentence;
		private ClientTask(Socket clientSocket) {
			this.clientSocket = clientSocket;
		}

		@Override
		public void run() {
```

```
                        try {

                                System.out.println("Got a client !!!");
                                DataOutputStream outToClient = new DataOutputStream(clientSocket.getOutputStream());
                                BufferedReader inFromClient =
                                        new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                                clientSentence = inFromClient.readLine();
                                System.out.println("TCP Server Thread Received: " + clientSentence);
                                capitalizedSentence = clientSentence.toUpperCase() + '\n';
                                outToClient.writeBytes(capitalizedSentence);
                                // Do whatever required to process the client's request

                                clientSocket.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }

}
```