Multithreaded Client/Server Application and Proxy Server. (10 points)

This homework consists of two parts: *i*) implement a multithreaded server (allow a server to handle multiple clients simultaneously) and *ii*) implement a fault tolerance and load balancing with a proxy server, both parts by using plain Java Sockets.

Please read the details about sockets and get an example classes in Java:

https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

The solution for this homework must be uploaded to OLAT until Monday, **09.04.2018**, at **08:00**, and will be evaluated on 10th of April!

Part 1: A Simple Client / (Multithreaded) Server Application.

The server should offer a simple service that has a single operation *String* ***sort****(inputString)*, which returns the sorted *inputString*. The server application processes the request and replies back the result to the client (Do not waste time and resources on special cases – empty string, too big string length, etc, just use the test case you like that works).

Implement the following components using Java (TCP) Sockets:

a. Implement a utility class *Protocol* defining a constant for a server port number as well as two methods *request* and *reply*. The *request* method should accept a socket connected to the server as an argument and should use the associated input / output streams to manage the client side of the communication. Each request should be processed within its own (new) thread. Hence, whenever *accept()* returns a server socket, a new request handler should be created, which is processing the request within its own thread. The *reply* method simply accepts a socket connected to the client and conducts the server's end of the communication protocol.

b. Implement the *Server*, which opens a new socket and waits for client connections. Clients' requests shall be processed according to your protocol by forwarding it to the *reply* method of the *Protocol* class.

c. Implement a proper remotely triggered server-shutdown. The server should stop accepting new requests and finishing all already started requests. Also describe the necessary protocol modifications. (You must not use System.exit(…); *Note:* when closing a server socket, any thread currently blocked by an invocation of the accept() method will throw a SocketException – you may exploit this fact.)

d. Demonstrate that your new server can handle multiple (at least two) clients simultaneously.

For all implementations, ensure proper exception and resource handling (e.g. close all streams + consider the possibility of I/O Exceptions).

(5 points)

Tip: Let your request be handled using an Extension of the Runnable interface and use an ExecutorService (see Executors) to maintain all threads. For the shutdown, close the server socket first and wait until all jobs within the executor are completed.

Part 2: A Proxy Server.

Extend the solution of Part 1 such that you create a proxy server or dispatcher between clients and end-point servers. Namely, clients should not be changed and they should submit their requests to the proxy server. The proxy server should distribute the requests to the end-point servers, which will conduct the sorting and return the sorted string to the proxy server. After this, the proxy should return the result to the client.

a) Develop the end-point server.

b) Should the end-point server be multi-threaded?

c) Develop an appropriate proxy server. The proxy server should return within the reply which server has done the sorting.

d) Demonstrate that your application is scalable with at least two clients, one proxy and two end-point servers.

e) Demonstrate that your application is fault tolerant by gracefully shutting down the second end-point server. Present that after the failure, clients will be served by the first end-point server.

(5 points)

Part 3 Bonus:

Implement a solution with an arbitrary number of end-point servers in which they can register to the proxy by sending a message to the well-known socket of the proxy. Similarly, they can abandon the proxy by sending a message to the proxy server. **(3 points)**