A blue logo with a black background

Description automatically generated

BÁO CÁO THỰC HÀNH

**Bài thực hành số 04**

**Môn học:** Lập trình ứng dụng mạng

**Lớp:** NT109.O21.MMCL

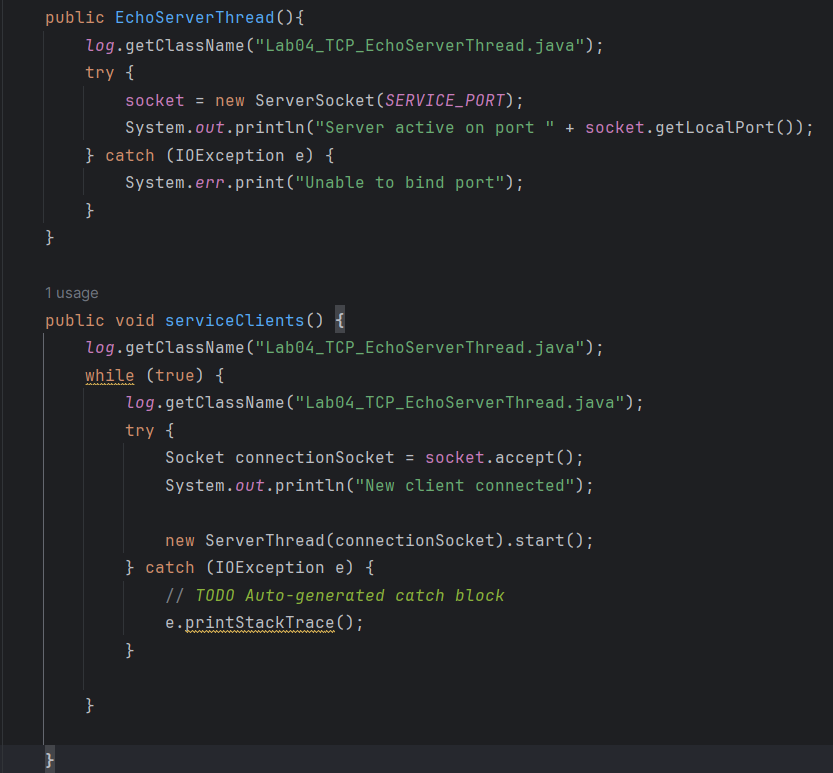
**SINH VIÊN THỰC HIỆN:**

|  |  |
| --- | --- |
| **Họ và tên** | **MSSV** |
| Nguyễn Văn Huấn | 21522108 |

# BÁO CÁO CHI TIẾT

## Nội dung 1

### Extend Threads cho TCP chat



(file EchoServerThread.java)

Constructor **EchoServerThread**:

* + Constructor này khởi tạo một **ServerSocket** trên một cổng cụ thể (**SERVICE\_PORT**).
  + Nếu việc khởi tạo thất bại do một **IOException**, nó in một thông báo lỗi cho biết không thể ràng buộc với cổng.

Phương thức **serviceClients**:

* + Phương thức này đảm nhận việc xử lý các kết nối từ các client.
  + Nó chứa một vòng lặp vô hạn liên tục chấp nhận các kết nối client đến bằng cách sử dụng phương thức **accept()** của **ServerSocket**.
  + Khi một client mới kết nối, nó tạo một đối tượng **ServerThread** mới (có thể xử lý giao tiếp với client) và khởi động nó như một luồng riêng biệt.
  + Nếu một **IOException** xảy ra trong quá trình **accept()**, nó in ra dấu vết ngăn xếp.

A computer screen shot of a program code

Description automatically generated

(file ServerThread.java)

**Log class name (Ghi tên lớp)**:

* + Dòng đầu tiên trong phương thức **run()** gọi phương thức **getClassName()** từ đối tượng **log** để ghi lại tên của lớp, có thể để ghi nhật ký về hoạt động của lớp.

**Đọc từ client và ghi đến client (Read from client and write to client)**:

* + Trong vòng lặp **do-while**, một **BufferedReader** được sử dụng để đọc dữ liệu từ client thông qua **socket.getInputStream()**, và một **PrintWriter** được sử dụng để ghi dữ liệu đến client thông qua **socket.getOutputStream()**.
  + Dữ liệu được đọc từ client dưới dạng chuỗi bằng cách sử dụng phương thức **readLine()** của **BufferedReader**.
  + Sau khi đọc dữ liệu từ client, chuỗi đó được in ra màn hình và ghi vào nhật ký với mức độ ghi nhật ký là "debug".
  + Sau đó, dữ liệu được gửi trở lại client sau khi được bổ sung với một chuỗi " (Server accepted!)" và kết thúc dòng.

**Kiểm tra điều kiện kết thúc (Check termination condition)**:

* + Vòng lặp sẽ tiếp tục chạy cho đến khi chuỗi từ client là "bye".

**Đóng kết nối (Close connection)**:

* + Khi client gửi chuỗi "bye", kết nối sẽ được đóng bằng cách gọi phương thức **close()** trên **socket**.
  + Nếu có lỗi xảy ra trong quá trình đọc hoặc ghi dữ liệu, một **IOException** sẽ được ném và thông báo lỗi sẽ được in ra màn hình cùng với dấu vết ngăn xếp.

A computer screen shot of a program

Description automatically generated

A computer screen with text and images

Description automatically generated

(file EchoClient.java)

**Ghi tên lớp vào log**:

* + Trong mỗi lần lặp, dòng đầu tiên ghi tên lớp vào log, có thể để ghi nhật ký về hoạt động của lớp.

**Nhập dữ liệu từ người dùng**:

* + Một **BufferedReader** được sử dụng để đọc dữ liệu từ người dùng thông qua **System.in** (bàn phím).
  + Dữ liệu nhập từ người dùng được đọc dưới dạng chuỗi bằng cách sử dụng phương thức **readLine()** của **BufferedReader**.

**Gửi dữ liệu tới máy chủ và nhận phản hồi từ máy chủ**:

* + Một **PrintWriter** được sử dụng để gửi dữ liệu tới máy chủ thông qua **clientSocket.getOutputStream()**.
  + Dữ liệu được gửi tới máy chủ thông qua phương thức **println()**, được kết thúc bằng dấu xuống dòng **'\n'**.
  + Sau đó, một **BufferedReader** được sử dụng để đọc dữ liệu từ máy chủ thông qua **clientSocket.getInputStream()**.
  + Dữ liệu phản hồi từ máy chủ được đọc dưới dạng chuỗi bằng cách sử dụng phương thức **readLine()** của **BufferedReader**.

**In kết quả ra màn hình và kiểm tra điều kiện kết thúc**:

* + Kết quả từ máy chủ được in ra màn hình.
  + Vòng lặp tiếp tục chạy cho đến khi người dùng nhập chuỗi "bye".

**Đóng kết nối socket**:

* + Khi người dùng nhập "bye", kết nối socket sẽ được đóng bằng cách gọi phương thức **close()** trên **clientSocket**.

A screen shot of a computer

Description automatically generated

(file log của phần TCP chat)

### b. MultiThreadDemo

A screen shot of a computer program

Description automatically generated

(file MyThread.java)

2 hàm sort: một hàm sort đơn giản và hàm sort sử dụng thuật toán bubbleSort()

A computer screen shot of a program code

Description automatically generated

(file Init.java)

Khởi tạo mảng 2000 phần tử và hoán đổi vị trí trong mảng để thành mảng random và log.

A screen shot of a computer

Description automatically generated

(file App.java)

Sử dụng switch case, với case 1 thì tạo single thread và chạy thuật toán sắp xếp mảng 2000 phần tử, case 2 thì tạo multithreads (5 threads) để sắp xếp một mảng

A screen shot of a computer screen

Description automatically generated

(file log của MultiThreadDemo.java)

### c. Trả lời câu hỏi

1. Khi nào ta sử dụng extend thread và implement từ interface Runnable

**- Nếu muốn đơn giản, dễ sử dụng, dễ tạo ra một luồng mới thì dùng extend thread.**

**- Nếu muốn linh hoạt hơn trong việc quản lý code thì implement từ interface Runnable**

Kết hợp câu 2 và 3. Chạy chương trình sắp xếp mảng tăng dần 2000 phần tử bằng single thread va so sánh thời gian chạy với 4 thread chạy song song + chạy bằng thuật toán bubble sort()

A black background with many small squares

Description automatically generated with medium confidence

(khi chạy single thread với đơn vị thời gian là nanosecond)

- Single thread chạy giải thuật sắp xếp với 5671300 nanosecond

A black background with many small squares

Description automatically generated

(khi chạy multithread với đơn vị là nanosecond)

- Tổng thời gian các threads chạy là 81379400 nanosecond

Single thread chạy nhanh hơn multithread: lý do theo em nghĩ là đối với những bài toán đơn giản như sắp xếp mảng 2000 phần tử thì single thread nhanh hơn còn với bài toán phức tạp chẳng hạn mảng 2000000 phần tử thì multithread sẽ nhanh hơn.

## Khó khăn

- Multithread không dễ, cần nhiều kiến thức và kĩ thuật lập trình để sử dụng.

# TÀI LIỆU THAM KHẢO