BÁO CÁO THỰC HÀNH

Bài thực hành số 04: Lập trình đa luồng

Môn học: Lập trình ứng dụng mạng

Lớp: NT109.O21.MMCL

THÀNH VIÊN THỰC HIỆN:

STT	Họ và tên	MSSV
1	Lê Hoàng Khánh	21522205

ĐÁNH GIÁ KHÁC:

Tổng thời gian thực hiện	1 tuần
Ý kiến *(nếu có)* + Khó khăn + Đề xuất, kiến nghị	Không có

MỤC LỤC

A. BÁO CÁO CHI TIẾT

1. Chương trình sắp xếp mảng.

a. Implement thuật toán sắp xếp.

Thuật toán sắp xếp ở đây là QuickSort một thuật toán nổi tiếng. Thuật toán này được đánh giá cao nhờ tốc độ sắp xếp nhanh chóng, hiệu quả trên nhiều kích thước mảng khác nhau.

```java
package io.github.tuilakhanh.Lab04;

public class QuickSort {

    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);
            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }

    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);

        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return (i + 1);
    }

    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

Lớp QuickSort bao gồm 2 phương thức chính:

quickSort(int[] arr, int low, int high):

- **Mục đích:** Sắp xếp mảng arr theo thứ tự tăng dần trong khoảng từ low đến high.
- **Cách thức:**
    - Điều kiện dừng: Nếu low >= high, mảng con đã được sắp xếp.

- Phân chia: Gọi hàm partition để chia mảng con thành hai phần, đặt phần tử pivot vào vị trí chính xác.
- Sắp xếp đệ quy: Gọi đệ quy quickSort để sắp xếp hai phần con còn lại bên trái và bên phải pivot.

partition(int[] arr, int low, int high):

- **Mục đích:** Chia mảng con từ low đến high thành hai phần xung quanh phần tử pivot, đảm bảo phần tử pivot ở vị trí chính xác.
- **Cách thức:**
  - Chọn pivot: Sử dụng phần tử cuối cùng (arr[high]) làm pivot.
  - Đặt chỉ mục: Khởi tạo i = low - 1.
  - Duyệt mảng: Duyệt từ low đến high - 1.
    - So sánh: Nếu arr[j] <= pivot, hoán đổi arr[i] và arr[j], đồng thời tăng i.
  - Đặt pivot: Hoán đổi arr[i + 1] và arr[high], trả về i + 1 (vị trí mới của pivot).

b. Warpper SingleThread

```java
package io.github.tuilakhanh.Lab04;

public class SingleThreadQuickSort implements Runnable {
    private final int[] arr;
    private final int low;
    private final int high;

    public SingleThreadQuickSort(int[] arr, int low, int high) {
        this.arr = arr;
        this.low = low;
        this.high = high;
    }

    @Override
    public void run() {
        QuickSort.quickSort(arr, low, high);
    }
}
```

Lớp SingleThreadQuickSort là một warpper bao bọc xung quanh class Quicksort, nhằm mục đích thực thi thuật toán này trong một luồng duy nhất.

SingleThreadQuickSort bao gồm:

Constuctor (SingleThreadQuickSort(int[] arr, int low, int high)):

- Khởi tạo các biến arr, low và high với giá trị tương ứng được truyền vào.

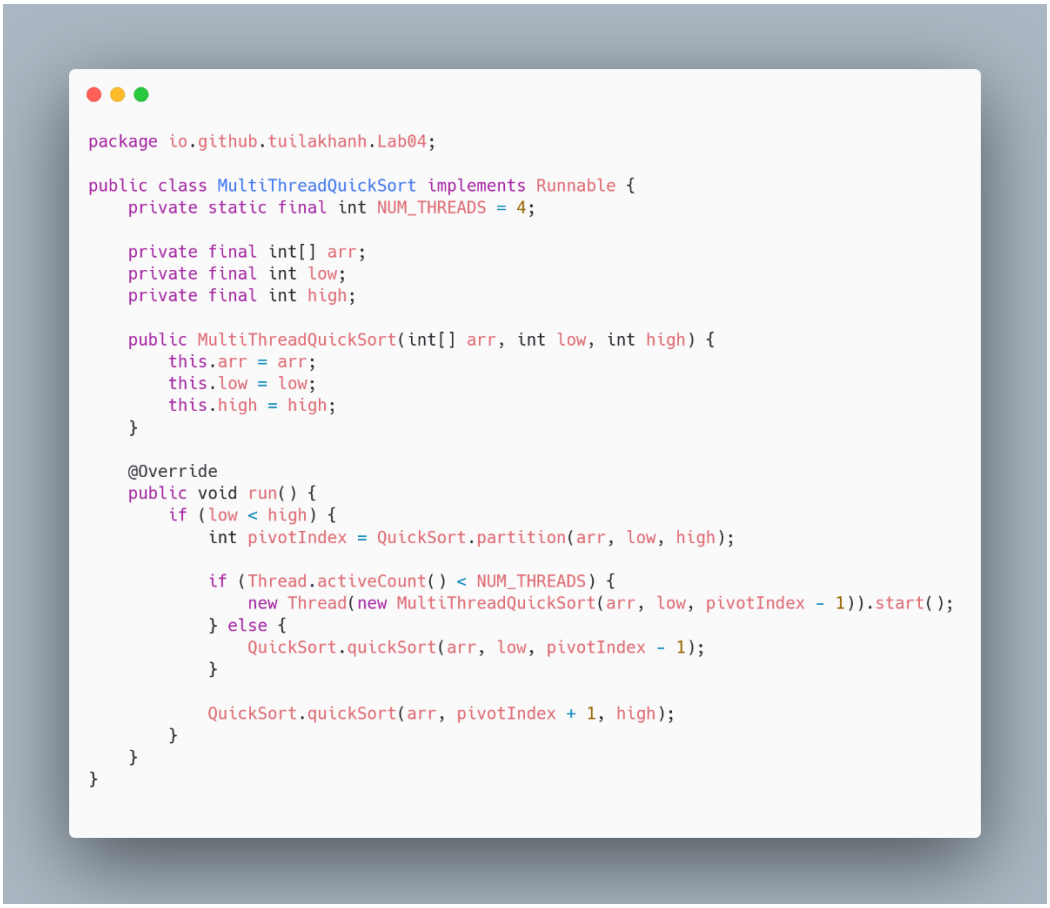
run():

- Triển khai giao diện Runnable của Java.
- Gọi hàm QuickSort.quickSort để thực hiện thuật toán Quicksort cho phần mảng con được chỉ định.

Cách thức hoạt động:

- **Bước 1:** Khởi tạo, tạo một đối tượng SingleThreadQuickSort với mảng cần sắp xếp, chỉ số bắt đầu và kết thúc của phần mảng con.
- **Bước 2:** Tạo luồng, tạo một luồng mới và truyền đối tượng SingleThreadQuickSort vào làm tham số cho phương thức start().
- **Bước 3:** Thực thi Quicksort, khi luồng được khởi chạy, phương thức run() được gọi, thực hiện thuật toán Quicksort cho phần mảng con được chỉ định.
- **Bước 4:** Hoàn thành, sau khi thuật toán Quicksort hoàn thành, việc sắp xếp phần mảng con được hoàn tất.

c. Warpper MultiThread



```java
package io.github.tuilakhanh.Lab04;

public class MultiThreadQuickSort implements Runnable {
    private static final int NUM_THREADS = 4;

    private final int[] arr;
    private final int low;
    private final int high;

    public MultiThreadQuickSort(int[] arr, int low, int high) {
        this.arr = arr;
        this.low = low;
        this.high = high;
    }

    @Override
    public void run() {
        if (low < high) {
            int pivotIndex = QuickSort.partition(arr, low, high);

            if (Thread.activeCount() < NUM_THREADS) {
                new Thread(new MultiThreadQuickSort(arr, low, pivotIndex - 1)).start();
            } else {
                QuickSort.quickSort(arr, low, pivotIndex - 1);
            }

            QuickSort.quickSort(arr, pivotIndex + 1, high);
        }
    }
}
```

Lớp MultiThreadQuickSort là warrpper của class QuickSort để tận dụng sức mạnh của lập trình đa luồng, nhằm tăng tốc độ sắp xếp mảng bằng thuật toán Quicksort.

Cấu trúc Lớp:

- NUM_THREADS: Hằng số xác định số lượng luồng tối đa được phép sử dụng.

Constructor (MultiThreadQuickSort(int[] arr, int low, int high)):

- Khởi tạo các biến arr, low và high với giá trị tương ứng được truyền vào.

run():

- Triển khai giao diện Runnable của Java.
- Kiểm tra điều kiện dừng: Nếu low >= high, phần mảng con đã được sắp xếp.
- Phân chia mảng con: Gọi hàm QuickSort.partition để chia mảng con thành hai phần, xác định vị trí pivot.
- Quản lý luồng:
    - Kiểm tra số lượng luồng: Nếu Thread.activeCount() < NUM_THREADS, tạo một luồng mới để sắp xếp phần mảng con bên trái pivot.

- Sắp xếp tuần tự: Nếu số lượng luồng tối đa đạt, thực hiện sắp xếp phần mảng con bên trái pivot tuần tự.

- Sắp xếp phần mảng con bên phải pivot: Gọi đệ quy MultiThreadQuickSort để sắp xếp phần mảng con bên phải pivot.

Cách thức hoạt động:

- **Bước 1:** Khởi tạo: Tạo đối tượng MultiThreadQuickSort với mảng cần sắp xếp, chỉ số bắt đầu và kết thúc của phần mảng con.
- **Bước 2:** Tạo luồng: Tạo một luồng mới và truyền đối tượng MultiThreadQuickSort vào làm tham số cho phương thức start().
  - Thực thi Quicksort đa luồng: Khi luồng được khởi chạy, phương thức run() được gọi.
  - Thuật toán Quicksort được thực hiện cho phần mảng con được chỉ định.
  - Nếu có thể, các phần mảng con bên trái được chia nhỏ và sắp xếp đồng thời trong các luồng riêng biệt.
- **Bước 3:** Hoàn thành: Sau khi thuật toán Quicksort hoàn thành, việc sắp xếp mảng được hoàn tất.

d. Thực thi các các lớp trên.

```java
package io.github.tuilakhanh.Lab04;

import io.github.tuilakhanh.log.Log;

import java.util.Arrays;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        int[] data = new Random().ints(2000, 0, 3000).toArray();
        int[] singleData = data.clone();
        int[] multiData = data.clone();

        Log log = new Log("configs/config.txt");

        long startTime = System.nanoTime();
        Thread singleThread = new Thread(new SingleThreadQuickSort(singleData, 0, data.length - 1));
        singleThread.start();
        try {
            singleThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        long endTime = System.nanoTime();
        long singleThreadTime = endTime - startTime;

        startTime = System.nanoTime();
        Thread multiThread = new Thread(new MultiThreadQuickSort(multiData, 0, data.length - 1));
        multiThread.start();
        try {
            multiThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        endTime = System.nanoTime();
        long multiThreadTime = endTime - startTime;

        // Print results
        log.writeLog(String.format("Array size: %s", data.length));
        log.writeLog(String.format("Single-threaded Quicksort Time: %sns\n", singleThreadTime));
        System.out.println(Arrays.toString(singleData));
        log.writeLog(String.format("Multi-threaded Quicksort Time: %sns\n", multiThreadTime));
        System.out.println(Arrays.toString(multiData));
    }
}
```

Các bước của chương trình.

- **Bước 1:** Tạo dữ liệu, tạo một mảng số ngẫu nhiên để sử dụng cho việc sắp xếp.
- **Bước 2:** Thực hiện Sắp xếp Quicksort, thực hiện cả thuật toán Quicksort đơn luồng và đa luồng trên các bản sao của mảng được tạo.
- **Bước 3:** Đo thời gian thực thi, đo thời gian cần thiết cho mỗi phương pháp sắp xếp.
- **Bước 4:** Xuất kết quả, ghi kết quả sắp xếp và thời gian thực thi vào một file log.

Các thành phần chính:

Khởi tạo dữ liệu:

- int[] data = new Random().ints(2000, 0, 3000).toArray();: Tạo một mảng data gồm 2000 số nguyên ngẫu nhiên trong phạm vi từ 0 đến 3000.
- int[] singleData = data.clone();, int[] multiData = data.clone();: Tạo các bản sao của mảng data để sắp xếp bằng các phương pháp đơn luồng và đa luồng.

Sắp xếp đơn luồng:

- Đo thời gian bắt đầu (startTime)
- Tạo luồng singleThread với đối tượng SingleThreadQuickSort.
- Khởi động luồng (singleThread.start())
- Chờ luồng hoàn thành (singleThread.join()).
- Đo thời gian kết thúc (endTime).
- Ghi kết quả vào file log và in kết quả ra màn hình.

Sắp xếp đa luồng:

- Lặp lại quá trình tương tự như sắp xếp đơn luồng, sử dụng luồng multiThread với đối tượng MultiThreadQuickSort.

e. Kết quả.

Thử nghiệm với mảng có kích thước nhỏ.

```
28-04-2024 02:34:57 - Array size: 20

28-04-2024 02:34:57 - Single-threaded Quicksort Time: 914642ns


[597, 648, 728, 1095, 1333, 1378, 1429, 1573, 1751, 1760, 1892, 2270, 2340, 2410, 2433, 2472, 2707, 2804, 2875, 2880]
28-04-2024 02:34:57 - Multi-threaded Quicksort Time: 523251ns


[597, 648, 728, 1095, 1333, 1378, 1429, 1573, 1751, 1760, 1892, 2270, 2340, 2410, 2433, 2472, 2707, 2804, 2875, 2880]
```

Thử nghiệm với mảng có kích thước lớn. (Kết quả mảng đã sắp xếp sẽ không được log lại).

```
28-04-2024 02:37:04 - Array size: 2000

28-04-2024 02:37:04 - Single-threaded Quicksort Time: 1494814ns


28-04-2024 02:37:04 - Multi-threaded Quicksort Time: 577099ns
```

Kết quả, qua 2 thực nghiệm ở trên có thể sử dụng MultiThread với số Thread là 4 sẽ giúp thuật toán nhanh hơn được khoảng 50%.

2. Extend Thread và Implement Runnable.

Điểm chung:

Cả extends Thread và implements Runnable đều là các phương pháp phổ biến để tạo luồng trong Java.

Điểm khác biệt:

Kiến trúc:

- **extends Thread**: Sử dụng mô hình kế thừa hướng đối tượng, biến lớp con của bạn thành một lớp con của Thread. Lớp con của bạn sẽ thừa hưởng các phương thức và thuộc tính của Thread, bao gồm cả khả năng quản lý trạng thái luồng, ưu tiên luồng và các thao tác I/O.
- **implements Runnable:** Tách biệt logic thực thi luồng khỏi lớp hiện tại. Lớp của bạn sẽ triển khai giao diện Runnable, cung cấp phương thức run() để xác định hành vi khi luồng được thực thi.

Kiểm soát luồng:

extends Thread: Cung cấp nhiều kiểm soát hơn đối với luồng. Có thể thay đổi tên luồng, thiết lập mức độ ưu tiên luồng, xử lý các ngoại lệ luồng. Sử dụng các phương thức luồng được xác định trước như sleep(), interrupt() và join().

implements Runnable: Ít kiểm soát hơn so với extends Thread. Người sử dụng chỉ có thể xác định logic thực thi luồng trong phương thức run(). Việc quản lý trạng thái luồng, ưu tiên và các thao tác I/O cần được xử lý thủ công hoặc bằng các thư viện hỗ trợ.

Sử dụng:

extends Thread: Thích hợp khi:

- Cần kiểm soát chặt chẽ hành vi của luồng, bao gồm tên, mức độ ưu tiên và trạng thái.
- Lớp của cần sử dụng các phương thức luồng được xác định trước.
- Cần tạo ra một luồng đơn lẻ với logic phức tạp.

implements Runnable: Ưu tiên khi:

- Cần tập trung vào logic thực thi luồng mà không cần quan tâm đến các chi tiết quản lý luồng.
- Lớp đã kế thừa từ một lớp khác, không cho phép kế thừa thêm.
- Cần tạo ra nhiều luồng nhỏ, đơn giản với logic tương tự.

Kết luận:

Lựa chọn giữa extends Thread và implements Runnable phụ thuộc vào nhu cầu sử dụng cụ thể.

extends Thread phù hợp cho các tình huống cần kiểm soát chi tiết luồng và sử dụng các phương thức luồng sẵn có.

implements Runnable linh hoạt hơn, dễ sử dụng và phù hợp cho các luồng đơn giản hoặc khi muốn kế thừa từ một lớp khác.

B. TÀI LIỆU THAM KHẢO