1.Tìm hiều về thuật toán sorting và search

+Tăng dần

[99, 88, 55, 77, 1, 66]

#1 vòng lặp số 1.

#1.1 -> [**99**, **88**, 55, 77, 1, 66] -> [**88**, ***99***, 55, 77, 1, 66] (So sánh phần tử đầu tiên với phần tử liền kề sau nó, nếu phần tử đầu tiên lớn hơn thì đổi chỗ 2 phần tử)

#1.2 -> [88, **99**, **55**, 77, 1, 66] -> [88, **55**, ***99***, 77, 1, 66]

1.3 -> [88, 55, **99**, **77**, 1, 66] -> [88, 55, **77**, ***99***, 1, 66] (Tương tự bước trên)

#1.4 -> [88, 55, 77, **99**, **1**, 66] -> [88, 55, 77, **1**, ***99***, 66] (Tương tự bước trên)

#1.5 -> [88, 55, 77, 1, **99**, **66**] -> [88, 55, 77, 1, **66**, ***99***] (Tương tự bước trên)

#2 vòng lặp số 2.

#2.1 -> [**88**, **55**, 77, 1, 66, 99] -> [**55**, ***88***, 77, 1, 66, 99] (So sánh phần tử đầu tiên với phần tử liền kề sau nó, nếu phần tử đầu tiên lớn hơn thì đổi chỗ 2 phần tử)

#2.2 -> [55, **88**, **77**, 1, 66, 99] -> [55, **77**, ***88***, 1, 66, 99] (So sánh phần tử đầu tiên với phần tử liền kề sau nó, nếu phần tử đầu tiên lớn hơn thì đổi chỗ 2 phần tử)

#2.3 -> [55, 77, **88**,\*\* 1\*\*, 66, 99] -> [55, 77,\*\* 1\*\*, ***88***, 66, 99] (Tương tự bước trên)

#2.4 -> [55, 77, 1, **88**, **66**, 99] -> [55, 77, 1, **66**, ***88***, 99] (Tương tự bước trên)

Sau vòng lặp số 2, hai phần tử 88 và 99 được xếp cuối mảng dữ liệu, điều này xảy ra tương tự với các vòng lặp tiếp theo

#3 vòng lặp số 3.

#3.1 -> [**55**, **77**, 1, 66, 88, 99] -> [55, 77, 1, 66, 88, 99] {Không thay đổi vị trí}

#3.2 -> [55, **77**, **1**, 66, 88, 99] -> [55,\*\* 1\*\*, ***77***, 66, 88, 99]

#3.3 -> [55, 1, **77**, **66**, 88, 99] -> [55, 1, **66**, ***77***, 88, 99]

#4 vòng lặp số 4.

#4.1 -> [**55**, **1**, 66, 77, 88, 99] -> [**1**, ***55***, 66, 77, 88, 99]

#4.2 -> [**1**, **55**, 66, 77, 88, 99] -> [1, 55, 66, 77, 88, 99] {Không thay đổi vị trí}

#5 vòng lặp số 5.

#5.1 -> [**1**, **55**, 66, 77, 88, 99] -> is\_sorted = true, break;

Code mẫu

private static void sort(boolean ascending) {

int input[] =[99, 88, 55, 77, 1, 66];

int temp;

for (int i = 0; i < input.length; i++) {

for (int j = 1; j < (input.length - i); j++) {

if (ascending) {

if (input[j - 1] > input[j]) {

temp = input[j - 1];

input[j - 1] = input[j];

input[j] = temp;

}

} else {

if (input[j - 1] < input[j]) {

temp = input[j - 1];

input[j - 1] = input[j];

input[j] = temp;

}

}

}

+Sắp xếp

private int getIndexOf(String s, String[] arr)

{

for (int i = 0; i < arr.length; i++){

if (arr[i].equals(s){

return i; //i là vị trí của s

}

}

return -1; // không có trả về -1 vì 0 là vị trí đầu tiên trong mảng

}

2. Multithread

**Thread** (luồng) về cơ bản là một tiến trình con (sub-process). Một đơn vị xử lý nhỏ nhất của máy tính có thể thực hiện một công việc riêng biệt. Trong Java, các luồng được quản lý bởi máy ảo Java (JVM).

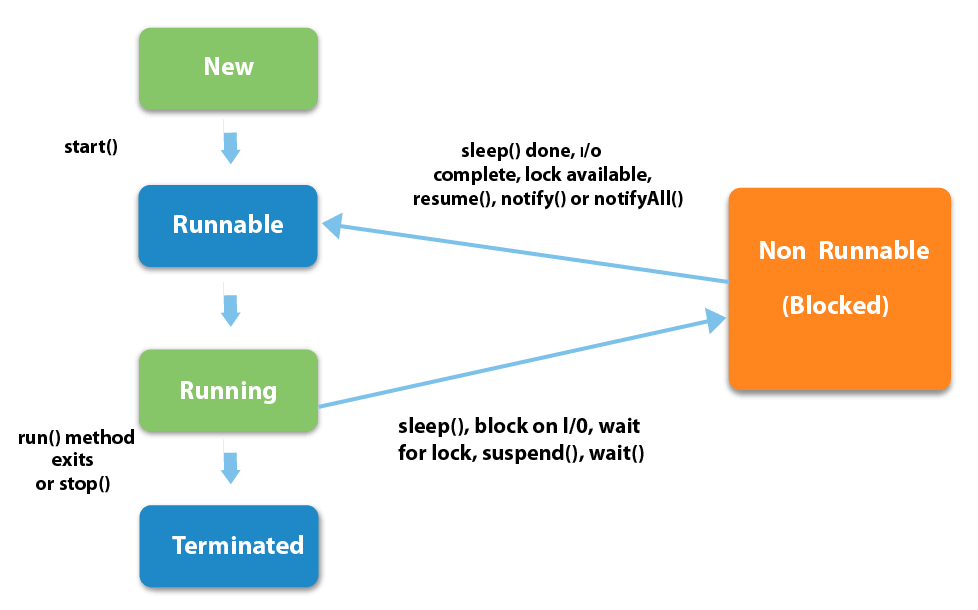
**Multi-thread** (đa luồng) là một tiến trình thực hiện nhiều luồng đồng thời. Một ứng dụng Java ngoài luồng chính có thể có các luồng khác thực thi đồng thời làm ứng dụng chạy nhanh và hiệu quả hơn.

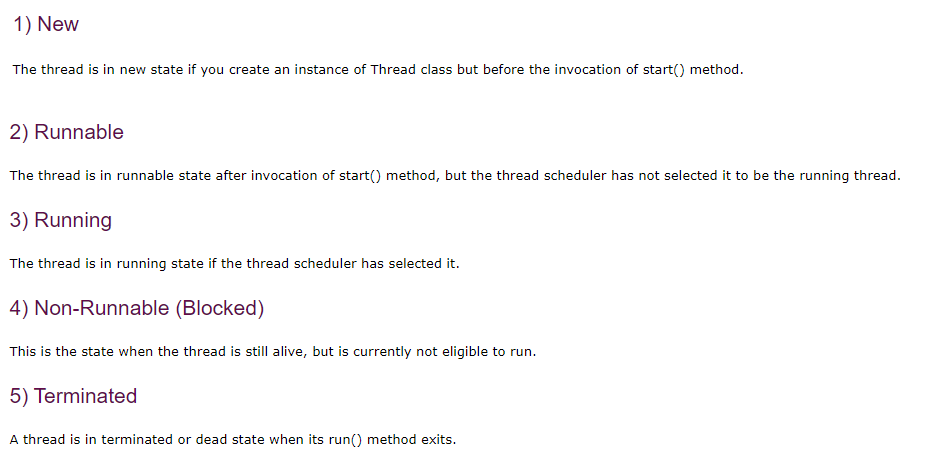
VD: Trình duyệt web hay các chương trình chơi nhạc là 1 ví dụ điển hình về đa luồng.

+ Khi duyệt 1 trang web, có rất nhiều hình ảnh, CSS, javascript… được tải đồng thời bởi các luồng khác nhau.

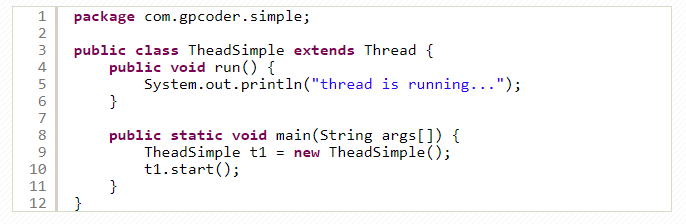
+ Khi play nhạc, chúng ta vẫn có thể tương tác được với nút điều khiển như: Play, pause, next, back … vì luồng phát nhạc là luồng riêng biệt với luồng tiếp nhận tương tác của người dùng.

Vòng đời của 1 thread

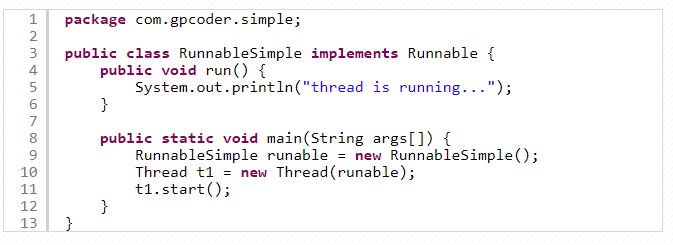




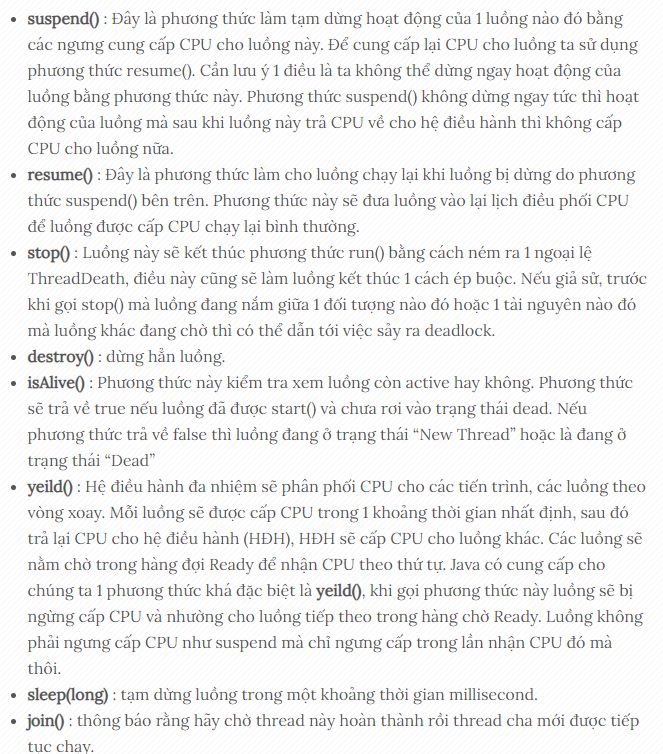
**Tạo luồng bằng cách extend từ lớp Thread**

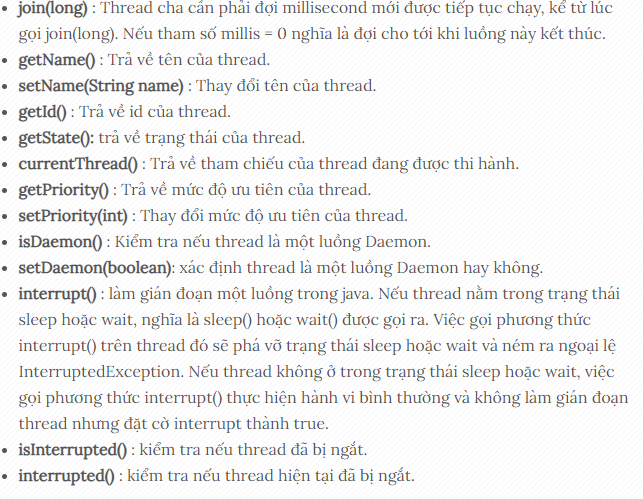


**Tạo luồng bằng cách implement từ Interface Runnable**



**Các hàm thường dung**

****



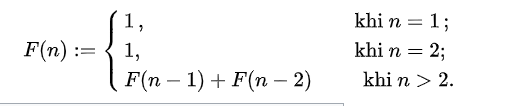
Tham khảo tại:

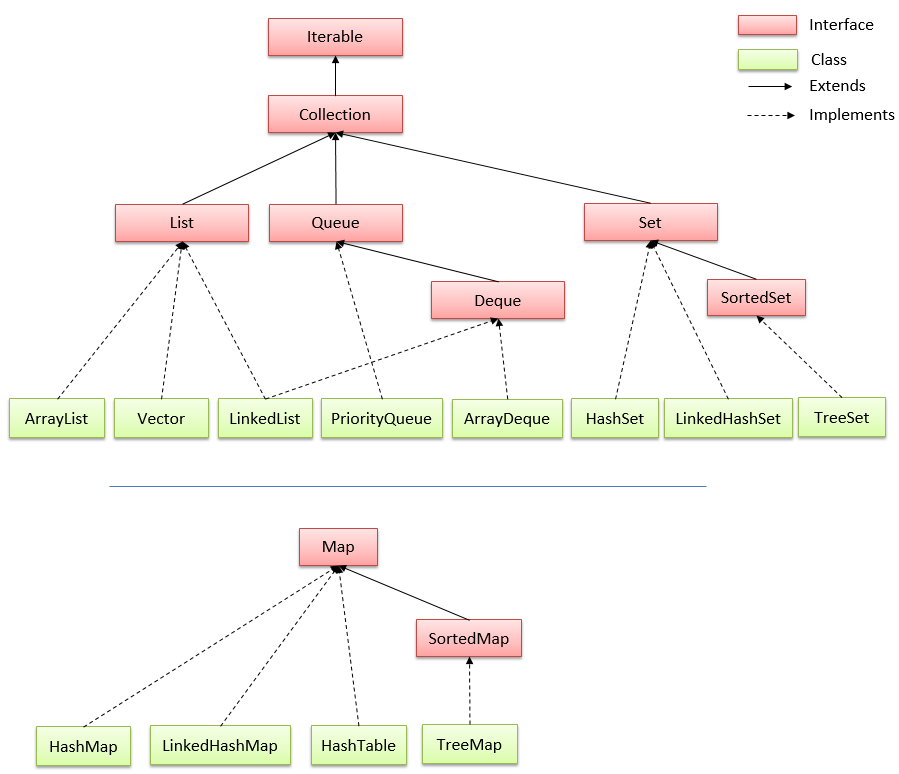
<https://gpcoder.com/3484-lap-trinh-da-luong-trong-java-java-multi-threading/#Tao_luong_bang_cach_implement_tu_Interface_Runnable>

<https://www.javatpoint.com/threadgroup-in-java>

3.Dãy số fibonacci

**Dãy Fibonacci** là [dãy](https://vi.wikipedia.org/wiki/D%C3%A3y_(to%C3%A1n_h%E1%BB%8Dc)) [vô hạn](https://vi.wikipedia.org/wiki/V%C3%B4_t%E1%BA%ADn) các [số tự nhiên](https://vi.wikipedia.org/wiki/S%E1%BB%91_t%E1%BB%B1_nhi%C3%AAn) bắt đầu bằng hai phần tử 0 và 1 hoặc 1 và 1, các phần tử sau đó được thiết lập theo quy tắc *mỗi phần tử luôn bằng tổng hai phần tử trước nó*. [Công thức truy hồi](https://vi.wikipedia.org/w/index.php?title=C%C3%B4ng_th%E1%BB%A9c_truy_h%E1%BB%93i&action=edit&redlink=1) của dãy Fibonacci là:





{\displaystyle F(n):=\left\{{\begin{matrix}1\,,\qquad \qquad \qquad \quad \,\ \ \,&&{\mbox{khi }}n=1\,;\ \ \\1,\qquad \qquad \qquad \qquad \,&&{\mbox{khi }}n=2;\ \ \,\\F(n-1)+F(n-2)&&{\mbox{khi }}n>2.\end{matrix}}\right.}

+ Queue

**Queue interface** của Java Collection Framework cung cấp chức năng của cấu trúc dữ liệu hàng đợi.

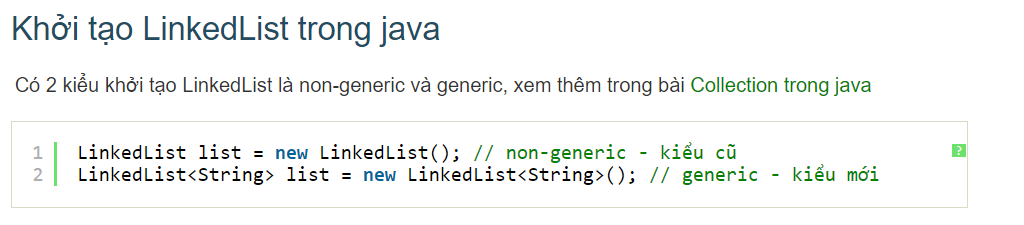
Vì Queue là một interface, nên chúng ta không thể triển khai trực tiếp interface đó.  
  
Để sử dụng các tính năng của cấu trúc dữ liệu Queue, chúng ta cần sử dụng các class triển khai nó:

* ArrayDeque
* LinkedList
* PriorityQueue

**Queue** có nghĩa là "**Hàng đợi**" (bạn có thể liên tưởng đến việc xếp hàng vậy)  
Trong hàng đợi (trong Queue), các phần tử được lưu trữ và truy cập theo cách nhập trước, xuất trước.

Có nghĩa là:

* Khi thêm phần tử thì sẽ xếp vào phía sau
* Khi loại bỏ phần tử thì sẽ loại bỏ từ phía trước



5.Stack

Lớp Stack là một lớp phụ của lớp Vector trong Java mà triển khai một last-in-first-out (LIFO) stack. Bạn có thể nghĩ về Stack như là một ngăn xếp thẳng đứng.

Stack chỉ định nghĩa constructor mặc định, mà tạo một stack trống.

Lớp Stack bao gồm tất cả phương thức được định nghĩa bởi lớp Vector, và một số phương thức khác của riêng nó.

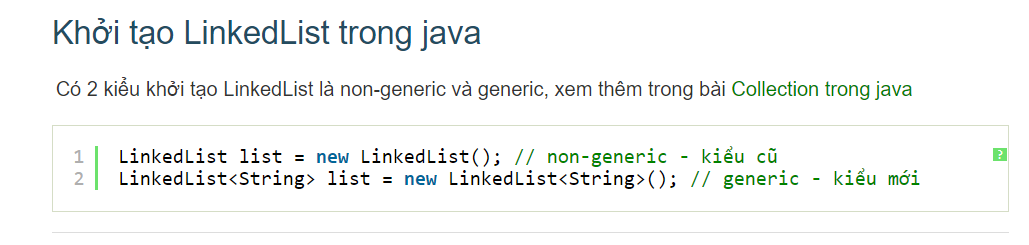


**LinkedList**

Lớp LinkedList trong java là một lớp kế thừa lớp AbstractSequentialList và triển khai của List, Queue Interface trong Collections Framework nên nó sẽ có một vài đặc điểm và phương thức tương đồng với List, Queue. Lớp LinkedList trong java sử dụng cấu trúc danh sách liên kết kép Doubly để lưu trữ các phần tử.

Những điểm cần ghi nhớ về lớp LinkedList:

* Lớp LinkedList trong java có thể chứa các phần tử trùng lặp.
* Lớp LinkedList duy trì thứ tự của phần tử được thêm vào.
* Lớp LinkedList là không đồng bộ (non-synchronized).
* Trong java lớp LinkList, thao tác nhanh vì không cần phải dịch chuyển nếu bất kỳ phần tử nào bị xoá khỏi danh sách.
* Lớp LinkedList trong java có thể được sử dụng như list (danh sách), stack (ngăn xếp) hoặc queue (hàng đợi).



**Lớp HashSet trong Java**

Lớp **HashSet trong java** là một lớp kế thừa lớp AbstractSet và triển khai của Set Interface trong Collections Framework nên nó sẽ có một vài đặc điểm và phương thức tương đồng với Set. HashSet được sử dụng để tạo một bộ sưu tập sử dụng bảng băm để lưu trữ. Nó kế thừa lớp AbstractSet và triển khai [Set interface](https://viettuts.vn/java-collection/set-trong-java).

**Đặc điểm của lớp HashSet trong Java**

Các điểm quan trọng về lớp HashSet trong java là:

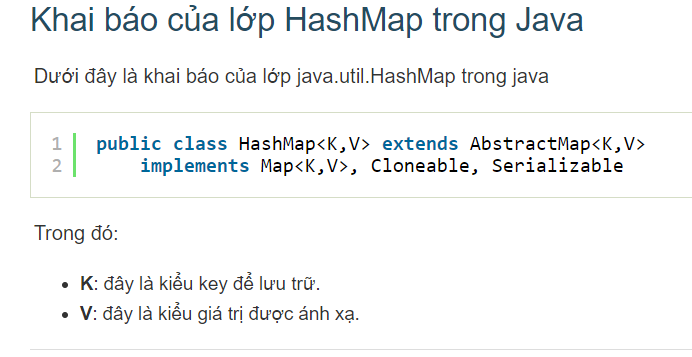
HashSet chỉ chứa các phần tử duy nhất.

HashSet lưu trữ các phần tử bằng cách sử dụng một cơ chế được gọi là **băm** (hash table).



**HashMap**

Lớp HashMap extends lớp AbstractMap và implements giao diện Map.



[**Lớp LinkedHashSet trong Java**](javascript:void(0))

Lớp LinkedHashSet trong Java kế thừa HashSet.

Lớp LinkedHashSet duy trì một linked list của các entry trong Set, theo thứ tự chúng đã được chèn. Điều này cho phép tính lặp lại theo thứ tự chèn qua Set đó.

Đó là, khi tuần hoàn qua một LinkedHashSet bởi sử dụng một Iterator, các phần tử sẽ được trả về theo thứ tự như khi chúng đã được chèn.

Hash code sau đó được sử dụng như là index, tại đó dữ liệu mà được liên kết với key đã được lưu trữ. Phép biến đổi của key này vào trong hash code được thực hiện tự động.

**Cấu trúc cây:**

<https://vietjack.com/cau-truc-du-lieu-va-giai-thuat/cau-truc-du-lieu-cay.jsp>

<https://nmlt.files.wordpress.com/2013/01/5-tree1.pdf>

Bài toán liên quan tới đồ thị (Graph)

https://viblo.asia/p/cau-truc-du-lieu-va-giai-thuat-cau-truc-du-lieu-do-thi-graph-djeZ1V6YlWz