# List

* Giới thiệu chung
* List là các giao diện kế thừa từ giao diện Collection
* Lưu trữ các phần tử một cách tuần tự, các phần tử có thể giống nhau
* Cấp phát động 🡪 Không cần khai báo giới hạn số lượng phần tử .
* Index xuất phát từ 0 và không thể truy cập tới đối tượng của list thông qua cách gọi index giống mảng mà phải thông qua phương thức get() ( ~~arrayList[3]~~ )
* Các lớp ArrayList, LinkedList, Vector

Sự giống nhau :

* Đều là các lớp được triển khai gián tiếp từ giao diện List – thừa kế các tính chất của List
* Đều được lập trình tổng quát nên kiểu dữ liệu của các lớp này là kiểu dữ liệu nguyên thủy: String, Integer, Double. Được khai báo thông qua cặp <> sau tên lớp ta định sử dụng.
* Đều có thể sử dụng iterator để duyệt tiến các phần tử trong list.
* Đặc trưng riêng của ArrayList, LinkedList, Vector

|  |  |  |
| --- | --- | --- |
| ArrayList | LinkedList | Vector |
| - Kế thừa từ lớp trừu tượng AbstractList triển khai giao diện List  - Các phương thức không đồng bộ (non-synchronized) 🡪 đồng bộ thủ công. Tuy nhiên sẽ chạy nhanh hơn khi không đồng bộ.  - Được sử dụng nhiều  - Search operation: độ phức tạp tính toán O(1), delete tồi nhất O(n)  - Duyệt các phần tử thông qua iterator , chỉ duyệt tiến | - Kế thừa lớp trừu tượng AbstractSequentialList-lớp kế thừa của lớp trừu tượng AbstractList. Ngoài ra còn triển khai từ giao diện Queue  - Các phương thức không đồng bộ (non-synchronized) 🡪 đồng bộ thủ công. Tuy nhiên sẽ chạy nhanh hơn khi không đồng bộ.  - Được sử dụng nhiều  - Search operation: độ phức tạp tính toán O(n), delete tồi nhất O(1).  - Duyệt các phần tử thông qua iterator, có thể duyệt tiến [hoặc lùi](http://javahungry.blogspot.com/2015/04/difference-between-arraylist-and-linkedlist-in-java-example.html) | - Kế thừa từ lớp trừu tượng AbstractList triển khai giao diện List.  - Các phương thức được đồng bộ 🡪 Hỗ trợ truy cập đa luồng. Chạy chậm hơn nhưng an toàn hơn.  - Hiện nay ít dùng do nhiều vấn đề #.  - Chậm hơn ArrayList  - Duyệt các phần tử thông qua iterator hoặc [enumeration](http://www.javatpoint.com/difference-between-arraylist-and-vector) |

* Khi nào nên dùng ArrayList, LinkedList, Vector:
* ArrayList: phù hợp với dữ liệu chỉ đọc, vì có tốc độ get() nhanh . ( quá trình nhập dữ liệu vào có thể được nhập từ trước, chạy ngầm nên khi users sử dụng sẽ nhanh hơn )
* LinkedList: phù hợp với việc thao tác trực tiếp dữ liệu do tính linh hoạt của nó
* Vector: ít được sử dụng.

# Map + HashSet

## Giống nhau :

* Các phần tử tồn tại duy nhất
* Thêm phần tử vào đầu hoặc cuối theo thứ tự giá trị value
* Làm việc với các kiểu lớp bao nguyên thủy

## [Khác nhau](http://simplecodecjava.blogspot.com/2016/01/java-su-khac-nhau-giua-hashmap-va.html)

|  |  |  |
| --- | --- | --- |
| Hashtable | HashMap | HashSet |
| - Quản lý dữ liệu theo kiểu bảng băm  Key 🡪Value, các value có thể bằng nhau  - Phần tử mới thêm vào theo thứ tự và sắp xếp theo thứ tự tăng dần của khóa .  - Các phương thức được đồng bộ tự động nên an toàn hơn nhưng chậm.  - Không cho lưu khóa và giá trị là NULL  - Triển khai từ giao diện MAP | - Quản lý dữ liệu theo kiểu bảng băm  Key 🡪Value, các value có thể bằng nhau  - Phần tử mới thêm vào không theo thứ tự và không sắp xếp theo thứ tự tăng dần của khóa .  - Các phương thức non-synchronized nên có thể đa luồng, chạy nhanh hơn  - Khóa và giá trị có thể là NULL  - Triển khai từ giao diện MAP  - HashMap cải thiện một số thiếu sót của HashTable. VD phương thức containValue(Object value). | - Quản lý dữ liệu theo kiểu tập hợp Set các giá trị là duy nhất  - Phần tử mới thêm vào theo thứ tự tăng dần của giá trị.  - Các phương thức non-synchronized nên có thể đa luồng, chạy nhanh hơn.  - Cho lưu giá trị là NULL  - Triển khai từ giao diện SET |

## Khi nào sử dụng

* Hashtable: Khi yêu cầu dữ liệu an toàn, không cần thiết nhanh chóng
* HashMap: Khi sử dụng nhiều luồng 🡪 Tối ưu tốc độ. Sử dụng khi gặp các Key có tính duy nhất nhưng các key lại có cùng value. Khi sử dụng giá trị có tính ánh xạ 1 .. n
* HashSet: Khi duy trì một danh sách duy nhất, sắp xếp có thứ tự tăng dần.

# Stack và Queue

## Giống nhau:

* Lưu trữ các phần tử theo trật tự liên tiếp, được cấp phát động
* Làm việc với các lớp bao nguyên thủy
* Có thể truy cập xóa phần tử ở mọi vị trí trong danh sách.

## Khác nhau

|  |  |
| --- | --- |
| Stack | Queue |
| - Kế thừa từ lớp Vector, nên có thẻ chèn được các phần tử vào mọi vị trí trong danh sách và các phương thức là synchronized.  - Hoạt động theo quy tắc LIFO : Last in First Out, vào trước ra sau  - Có hỗ trọ tìm kiếm các phần tử trong danh sách | - Là một interface, được triển khai bởi các lớp con: Deque và LinkedList  - Hoạt động theo nguyên tắc FIFO: First In First Out, vào trước ra trước kiểu xếp hàng.  -Không hỗ trợ tìm kiếm các phần tử, muốn tìm kiếm phải thông qua iterator. |

## Khi nào sử dụng :

* Stack: Đối với dữ liệu cần tính an toàn cao và có thể linh hoạt trong việc cập nhật. Đối với kiểu dữ liệu nhập vào dạng FIFO. Có thể sử dụng cho các tác vụ trực tiếp với NSD.
* Queue: Đối với trường hợp sử dụng cần tốc độ nhanh, kiểu dữ liệu dạng FIFO. Sử dụng cho tác vụ chạy ngầm