**Hash table**

Hash table là một cấu trúc dữ liệu cung cấp khả năng thêm và tìm kiếm rất nhanh.

* Dù cho có nhiều mục dữ liệu như thế nào , chèn và tìm kiếm (Đôi khi xóa) có thể chỉ mất thời gian không đổi : O(1)

Nhược điển: Bảng dựa trên mảng và mảng khó mở rộng sau khi được tạo ra.

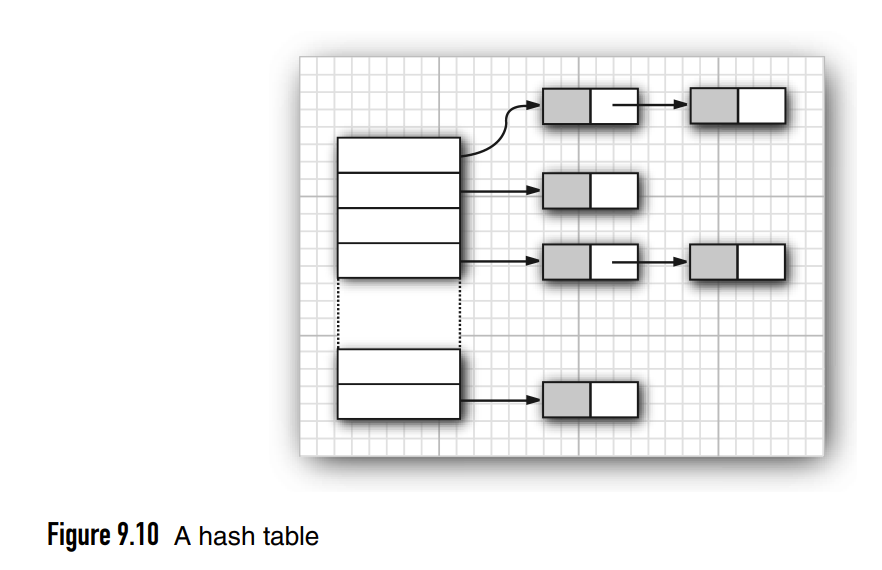
* Đối với một số loại Hash table , hiệu xuất có thể xuống cấp trầm trọng khi một bảng trở lên quá đầy đủ. Vì vậy người lập cần có ý tưởng tương đối chính xác có bao nhiêu mục dữ liệu cần được lưu trữ(Hoặc chuyển bị để chuyển dữ liệu sang một bảng băm lớn hơn, một quá trình tốn nhiều thời gian).
* Không có cách nào thuận tiện để truy cập các mục trong trong Hash table (oder)

**Introduction to Hashing**

* Một khái niệm quan trọng , cách chuyển đổi một dài giá trị thành giá trị chỉ số mảng.

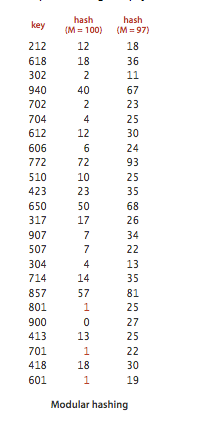
Hash table

* Trong java Hash table được triển khai dưới dạng linked lists. Từng danh sách được gọi là bucket.

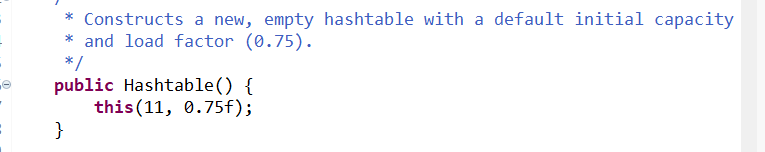


* Để tìm kiếm một đối tượng trong bảng, hãy tính hash code và giảm nó theo modulo của tổng buckets. Kết quả số là chỉ số của bucket chứa phần tử.
* VD: hash code = 76268, total Bucket: 128
* Index bucket = 76268 % 128.

Đôi khi kết quả băm sẽ trùng. Dẫn đến tình trạng xung đột băm (a hash collision)



* Nếu load factor(hệ số tải) is 0.75 lớn hơn 75% Thì sẽ nhân đôi tổng bucket trước đó. Và tiến hành băm lại.
* Mảng 1 chiều ban đầu có giá trị mặc định 11.

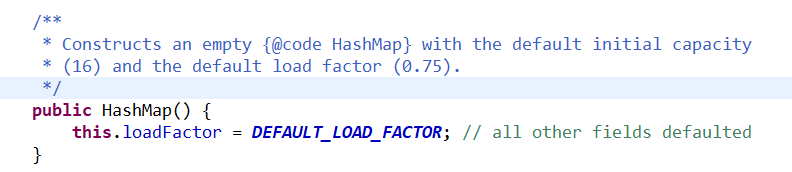


# Sự khác biệt giữa HashMap và Hashtable

HashMap và Hashtable đều được sử dụng để lưu trữ dữ liệu ở dạng khóa và giá trị. Cả hai đều đang sử dụng kỹ thuật băm để lưu trữ các khóa duy nhất

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap **không được đồng bộ hóa** . Nó không an toàn theo luồng và không thể được chia sẻ giữa nhiều luồng nếu không có mã đồng bộ hóa thích hợp. | Hashtable được **đồng bộ hóa** . Nó an toàn theo chủ đề và có thể được chia sẻ với nhiều chủ đề. |
| 2) HashMap **cho phép một khóa null và nhiều giá trị null** . | Hashtable **không cho phép bất kỳ khóa hoặc giá trị rỗng nào** . |
| 3) HashMap là một **lớp mới được giới thiệu trong JDK 1.2** . | Hashtable là một **lớp kế thừa** . |
| 4) HashMap **nhanh** . | Hashtable **chậm** . |
| 5) Chúng ta có thể làm cho HashMap được đồng bộ hóa bằng cách gọi mã này là Map m= Collections.synchronizedMap (hashMap); | Hashtable được đồng bộ hóa nội bộ và không thể không được đồng bộ hóa. |
| 6) HashMap được duyệt **qua bởi Iterator** . | Hashtable được duyệt **bởi Enumerator và Iterator** . |
| 7) Trình lặp lại trong HashMap **không nhanh** . | Điều tra viên trong Hashtable **không nhanh** . |
| 8) HashMap kế thừa lớp **AbstractMap** . | Hashtable kế thừa lớp **Từ điển** . |

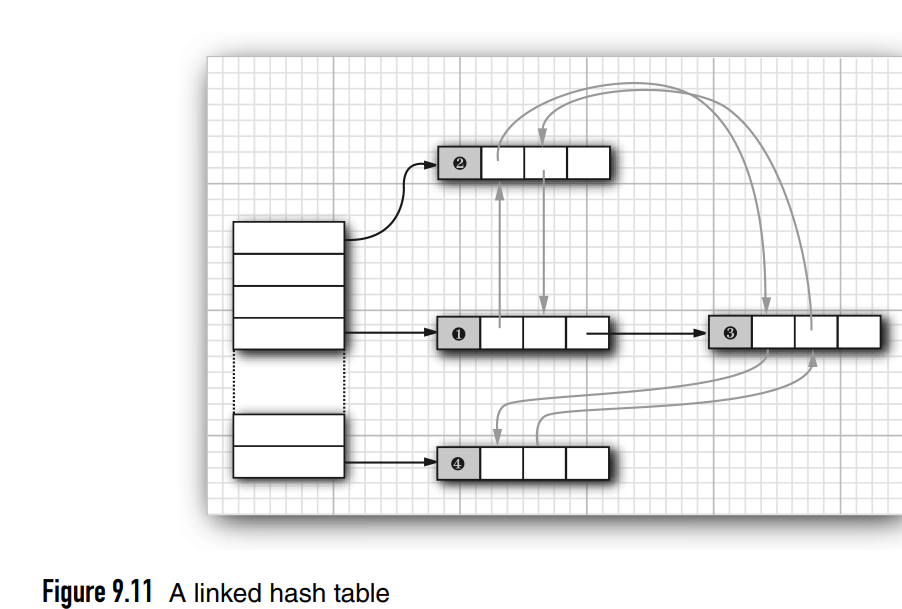
**HASH MAP**

* HashMap là một phần trong Collection của java. Được triển khai trên interface Map.
* HashMap được lưu trữ dữ liệu dưới dạng key, value.
* Khi chèn 1 giá trị chùng key, thì sẽ thay thế giá trị của key đó.
* **HashMap** tương tự như [HashTable](https://www.geeksforgeeks.org/hashtable-in-java/) , nhưng nó không được đồng bộ hóa. Nó cũng cho phép lưu trữ các khóa null, nhưng chỉ nên có một đối tượng khóa null và có thể có bất kỳ số giá trị null nào. Lớp này không đảm bảo về thứ tự của bản đồ
* Một trong các lớp con của HashMap là [LinkedHashMap](http://java.sun.com/javase/7/docs/api/java/util/LinkedHashMap.html), vì vậy trong trường hợp bạn muốn thứ tự lặp lại có thể dự đoán được (theo mặc định là thứ tự chèn), bạn có thể dễ dàng hoán đổi HashMap cho a LinkedHashMap. Điều này sẽ không dễ dàng nếu bạn đang sử dụng Hashtable.
* **HashMap**  khởi tạo mảng ban đầu có giá trị 16, load factor = 0.75
* 

**LinkedHashMap**

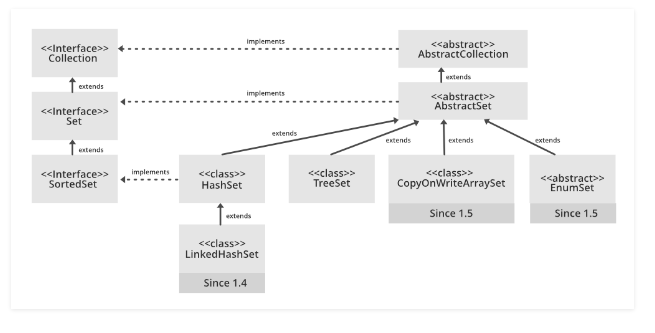
**Hệ thống phân cấp của LinkedHashMap**

* LinkedHashMap giống như HashMap vì LinkedHashMap extend HashMap , nhưng với tính năng bổ xung là có thể lưu thứ tự thêm vào của phần tử.
* LinkedHashMaplà bất đồng bộ.



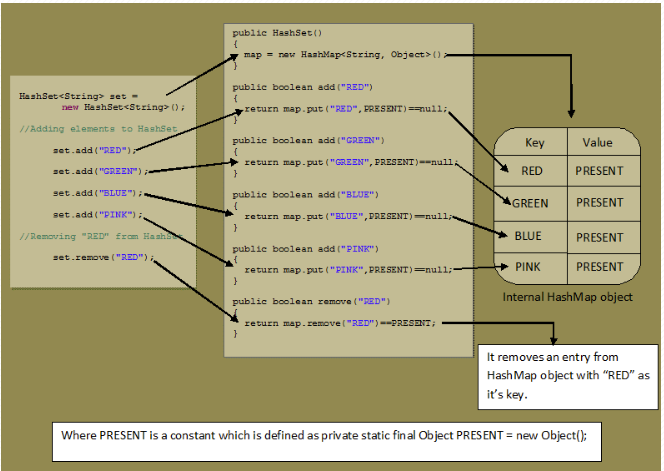
**HashSET**

Hệ thống phân cấp của HashSet

****

* Set Collection là tập các phần tử không trùng lặp.
* HashSet cũng triển khai các giao diện **Serializable** và **Cloneable** .
* Các phần tử NULL được phép trong HashSet.
* Java Collection cung cấp một lớp HashSet triển khai một bộ dựa trên một hash table.
* Khi thêm phần tử bằng method add. Method contains để thực hiện kiểm tra một phần tử đã có trong Set. Và sẽ chỉ kiểm tra phần tử trong bucket chứ không phải toàn bộ phần tử trong tập hợp.

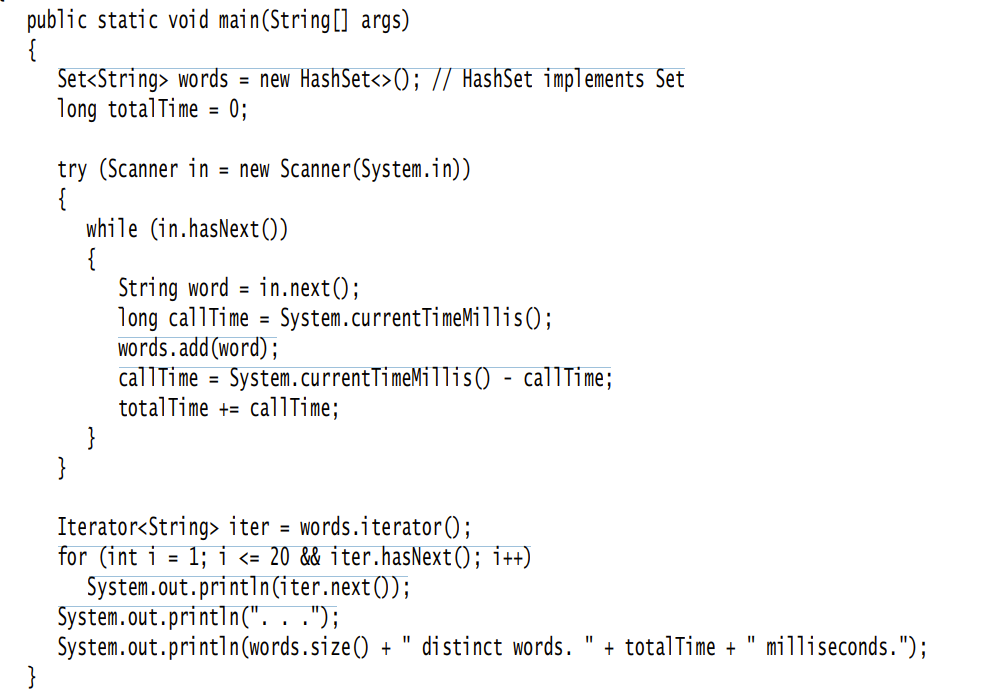
**Khởi tạo HashSet**



* HashSet được khởi tạo từ một HashMap nên bộ nhớ lưu dữ liệu tương tự HashMap.
* Khi sử dụng hàm add về bản chất là ta đang push 1 key và value với vs key là data thêm vào còn value là 1 đối tượng. Và đối tương đó giống nhau trên tất cả data của HashMap.
* **HashMap**  khởi tạo mảng ban đầu có giá trị 16, load factor = 0.75.

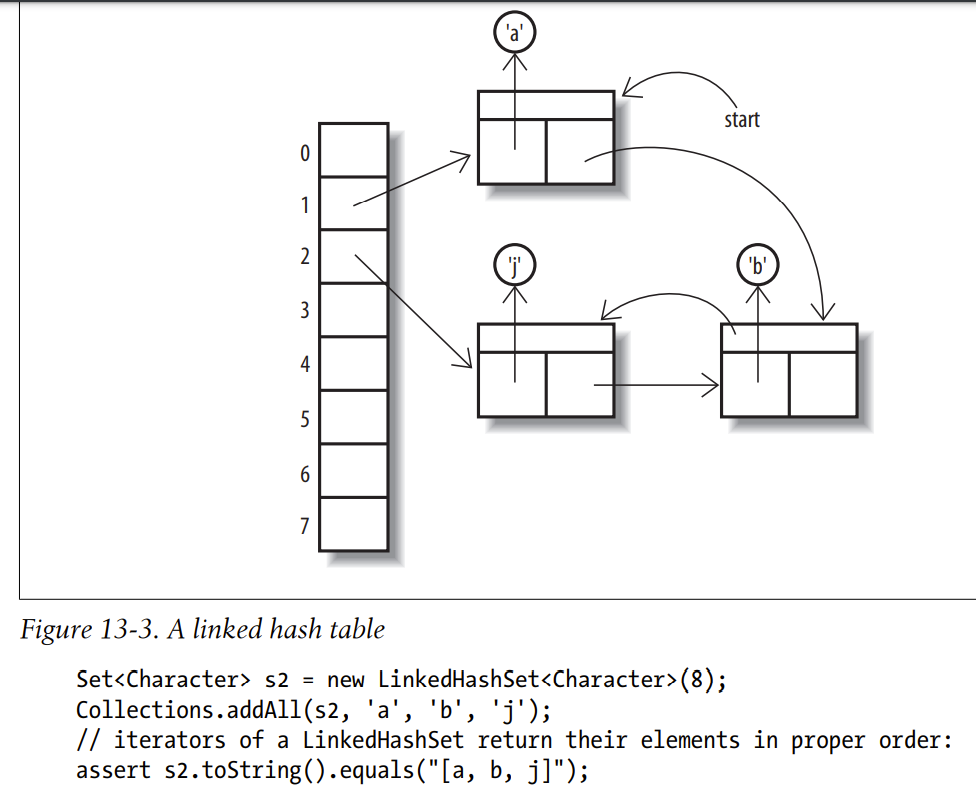
**Ưu và nhược điểm.**

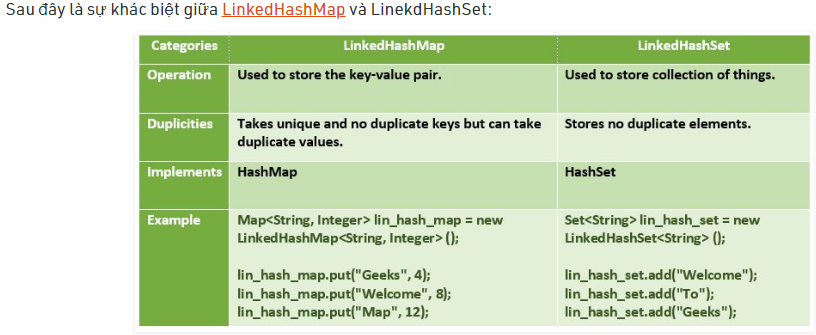
* HashSET được triển khai dựa trên Hash Table nên có những đặc điểm như, insert, search nhanh, và được sử dụng khi mong muốn các phần tử trong tập hợp không bị trùng lặp,
* Nhược điểm: không thuận tiện cho việc sắp xếp dữ liệu.



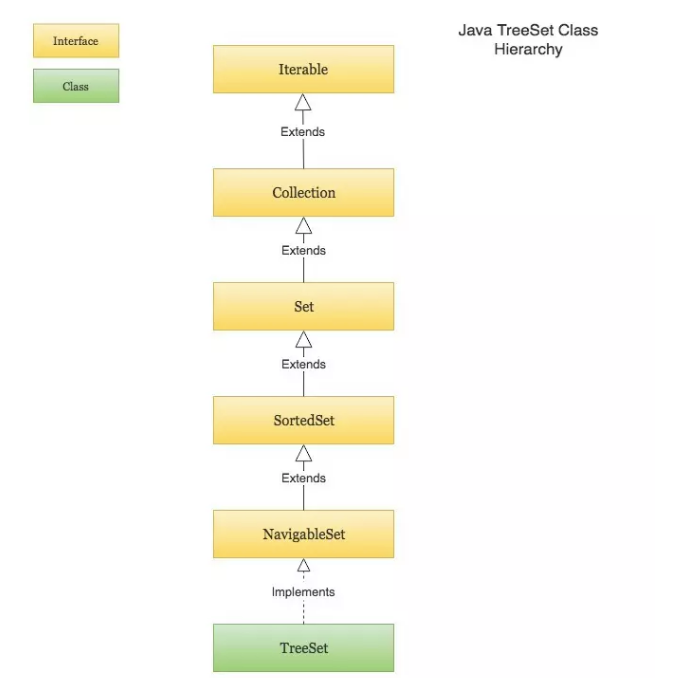
**LinkedHashSet**

* LinkedHashSet được extens HashSet và implement Set.
* LinkedHashSet được khởi tạo từ LinkedHashMap nên có thể lưu thứ tự của phần tử thêm vào.
* LinkedHashSet lưu các phần tử không chùng lặp





**TreeSet**

****

* Class [TreeSet](https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html#TreeSet--) là một phần của java collection framework. Nó implement từ interface **NavigableSet**, lớp này được kế thừa từ **SortedSet**.
* Lớp TreeSet trong nội bộ sử dụng TreeMap để lưu trữ các phần tử. Các phần tử trong một TreeSet được sắp xếp theo thứ tự tự nhiên của chúng.
* Bạn cũng có thể cung cấp một phương thức so sánh tùy chỉnh Comparator cho TreeSet tại thời điểm khởi tạo để cho phép nó sắp xếp các phần tử dựa trên bộ so sánh được cung cấp.

SortedSet interface cung cấp các hàm để giữ cho các phần tử được sắp xếp. Và NavigableSet interface cung cấp các hàm để điều hướng thông qua SortedSet.

Ví dụ, tìm phần tử lớn hơn hoặc nhỏ hơn phần tử đã cho, tìm phần tử đầu tiên và phần tử cuối cùng trong SortedSet.

Vì lớp TreeSet kế thừa từ NavigableSet interface, nên nó có những phương thức của cả NavigableSet cũng như SortedSet.

Một số đặc điểm quan trọng của TreeSet trong Java :

* TreeSet không chứa các phần tử trùng lặp.
* Các phần tử trong một TreeSet được sắp xếp theo thứ tự tự nhiên của chúng, hoặc dựa trên một bộ so sánh Comparator tùy chỉnh được cung cấp tại thời điểm khởi tạo TreeSet.
* TreeSet không thể chứa các giá trị null.
* TreeSet sử dụng TreeMap để lưu trữ các phần tử.
* TreeSet class không phải là một thread-safe. Bạn phải đồng bộ hóa rõ ràng quyền truy cập đồng thời vào TreeSet trong môi trường đa luồng.
* TreeSet tương tự như hashSet nhưng có một số cải tiến,Một treeSet là một tập hợp đã được sắp xếp.
* Việc sắp xếp được thực hiện dữa trên cấu trúc dữ liệu dạng tree.
* Việc add phần tử và TreeSet sẽ chậm hơn vào.

**\*HashCode:**

- Hash code là một số nguyên có nguồn ngốc từ một đối tượng.

- HashCode method được định nghĩa trong class Object. Do đó mọi đối tượng đều có hash code mặc định. Hash code đó có nguồn ngốc từ địa chỉ ô nhớ của object.

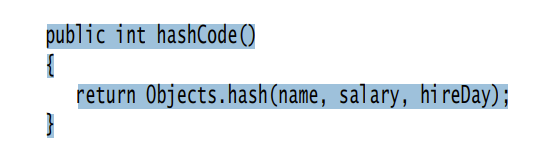
- Chú ý. Đối với String thì hash code có nguồn ngốc từ content của chúng

- nếu overive HashCode method thì cũng xẽ cần overive lại Equals method.

- Định nghĩa về equals method vs hashCode phải tương thíc với nhau.

+ Giả sử x.equsl(y) đúng thì x.hashcode phải bằng y.hashCode

+ Nếu xác định Employee.equals để so sánh ID nhân viên, sau đó phương thức hashCode cần băm ID, không phải tên nhân viên hoặc địa chỉ bộ nhớ.



**\*Equeal:**

- Equals method trong Object class dùng để kiểm tra xem 1 đối tượng có được coi là bằng nhau hay không. Equals method xác định liệu hai tham chiếu đối tượng có giống hệt nhau hay không.

