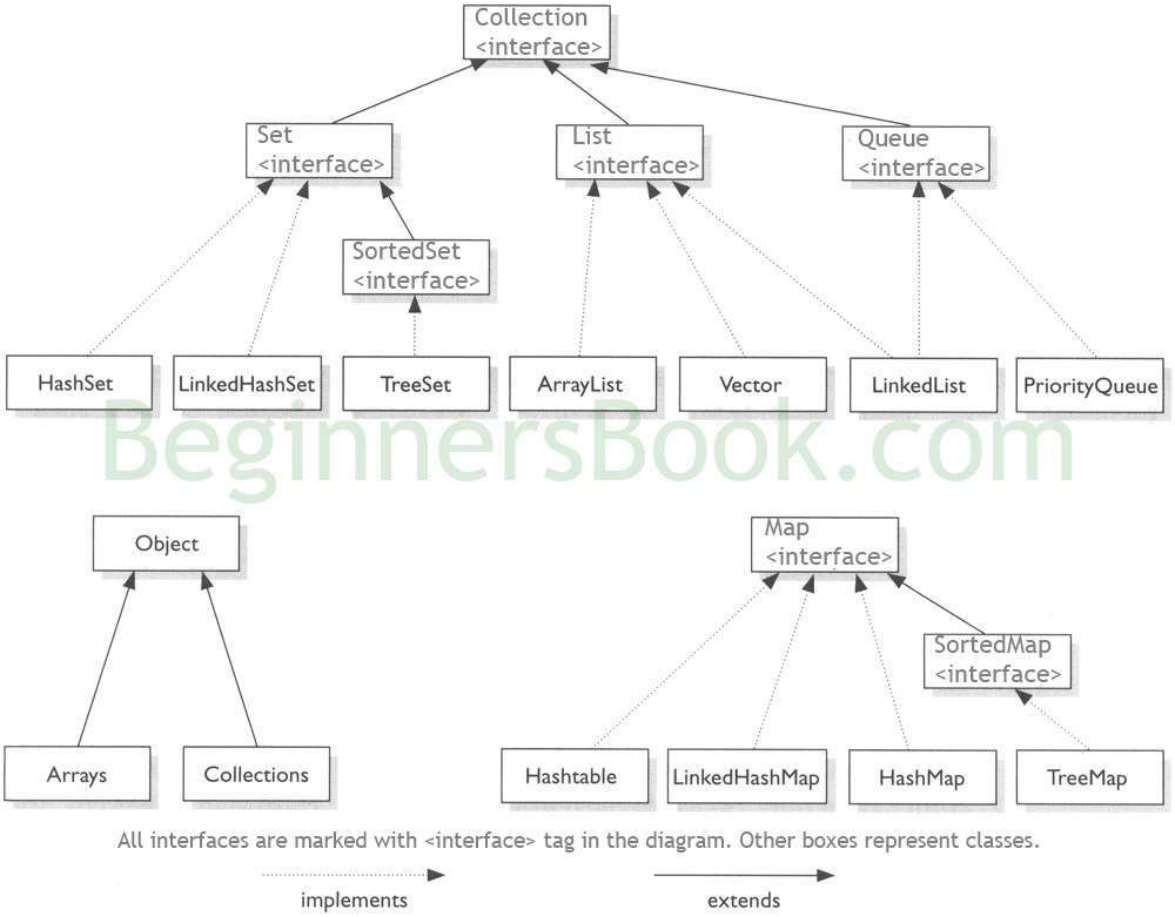
Hướng dẫn sử dụng Java Collection Framework



1. Tổng quan về các Collection:

1.1. Nhóm Collection lưu trữ các đối tượng, dữ liệu là các kiểu tham chiếu:

- Có 3 nhánh con trong nhóm Collection: **Queue, List, Set**.
- Các phần tử có thể giống nhau hoặc không phụ thuộc vào thuộc nhánh nào trong 3 nhánh kể trên

1.2. Nhóm Map lưu trữ các cặp key/value:

- Các cặp **key/value** chứa trong Map (bản đồ) là luôn có key khác nhau giữa các cặp
- Nếu biết key có thể lấy ra giá trị value trong Map ứng với key này

2. Sự khác nhau giữa các Collection:

java.util.Queue	*java.util.List*	*java.util.Set*
Cho phép chứa các phần tử trùng lặp	Cho phép chứa các phần tử trùng lặp	Không cho phép chứa các phần tử trùng lặp
Không cho phép chứa các phần tử null	Cho phép chứa nhiều phần tử null	Tùy theo class thi hành Set hỗ trợ chứa phần tử null hay không. Nếu có hỗ trợ thì chỉ chứa nhiều nhất 1 phần tử null nếu có.

Một danh sách (**List**) là một danh sách tuần tự các đối tượng, nơi mà các đối tượng giống nhau có thể xuất hiện một hoặc nhiều lần. Ví dụ: [1, 7, 1, 3, 1, 1, 1, 5]. Và bạn có thể nói về phần tứ "thứ N" trong danh sách. Bạn có thể thêm một phần tử vào bất kỳ một vị trí nào trong danh sách, thay đổi một phần tử nào tại một vị trí nào đó trong danh sách, hoặc xóa một phần tử tại một vị trí bất kỳ trong danh sách.

Hàng đợi (**Queue**) cũng là một tập hợp tuần tự, nhưng bạn chỉ có thể chạm vào phần tử đứng ở đầu hàng đợi. Tất cả các phần tử được chèn vào cuối của hàng đợi và xóa phần tử đầu tiên của hàng đợi. Bạn có thể biết được có bao nhiêu phần tử trong hàng đợi, nhưng bạn không thể tìm ra hoặc nói về phần tử thứ N, bạn chỉ có thể thấy nó khi nó đứng lên đầu tiên của hàng đợi.

Set là một tập hợp không tuần tự, và nó không cho phép trùng lặp. Bất cứ một phần tử nào hoặc nằm trong tập hợp hoặc không nằm trong tập hợp. {7, 5, 3, 1} chính xác là giống với {1, 7, 1, 3, 1, 1, 1, 5}. Bạn không thể nói về phần tử thứ N thậm chí là phần tử đầu tiên, vì nó không có sự tuần tự. Bạn có thể thêm hoặc xóa các phần tử, và có thể tìm ra nếu thực sự nó tồn tại (Ví dụ "7 có nằm trong tập hợp này không?"). **Chú ý**: **SortedSet** là một interface con của **Set** nó có thể chứa các phần tử có thứ tự.

2.1. Sự khác nhau giữa TreeSet, LinkedHashSet và HashSet

a. Giới thiệu sơ lượt:

- Chức năng chính: TreeSet là sorting, LinkedHashSet theo thứ tự insert và HashSet chỉ lưu trữ object.
- HashSet implement HashMap, TreeSet implement TreeMap.

b. Giống nhau:

- Duplicates: cả ba đều implement Set nên đều không cho phép insert phần tử giống nhau.
- Thread safety: đều không thread-safe, nếu sử dụng trong môi trường mutil-threading phải sử dụng synchronize bên ngoài nếu cần.
- Fail-fast iterator: cả ba đều trả về fail-fast iterator.

c. Khác nhau:

- Performance và Speed: HashSet nhanh nhất, LinkedHashSet nhanh nhì(gần như tương tự HashSet), TreeSet chậm hơn một tí vì phải sắp xếp cho mỗi lần insert.
- Ordering: HashSet không thực hiện, LinkedHashSet theo thứ tự insert (insertion-order) giống như List interface, TreeSert sắp xếp cho mỗi phần tử.
- Internal implement: HashSet sử dụng HashMap, LinkedHashSet sử dụng HashMap và LinkedList, TreeSet sử dụng TreeMap.
- Null: Cả 2 HashSet và LinkedHashSet cho phép phần tử null, nhưng TreeSet không cho phép và throws NullPointerExeption nếu insert null.

2.2. Sự khác nhau giữa LinkedList và ArrayList

a. Giống nhau:

- Đều implement List interface, nên đều chứa các method của List interface.
- Đều không synchronized.
- Ordered inserted.
- Cho phép phần tử giống nhau và null.
- Fail-fast iterator.

b. Khác nhau:

- Tất cả sự khác nhau giữa ArrayList và LinkedList nguyên nhân chính là do sự khác nhau về data-structure, Array và LinkedList.
- Array searching và getting phần từ rất nhanh, dựa vào chỉ số(index). Nhưng khi remove thì tốn kém cho việc sắp xếp lại tất cả các phần tử. LinkedList không cho truy cập dựa theo index, nếu muốn lấy 1 phần tử phải duyệt qua list.

- Insert và remove trong LinkedList nhanh hơn ArrayList vì ArrayList cần phải resizing (nếu insert vào ArrayList đã đầy thì phải copy toàn bộ vào một array mới có kích thước lớn hơn và sau đó insert vào).
- LinkedList thì có nhiều memory overhead hơn ArrayList(Array chỉ cần index và data, còn LinkedList cần data, địa chỉ của con trỏ next và pre).

*** Lưu ý:**

- Sử dụng ArrayList khi ứng dụng sử dụng nhiều truy cập theo index, và ít thao tác insert và delete.
- LinkedList ngược lại.

2.3. Sự khác nhau giữa Vector và ArrayList

a. Giống nhau:

- Vector và ArrayList đều sử dụng nền tảng index và hỗ trợ bởi array bên trong.
- Cả 2 đều thực hiện thứ tự insert (insert-order).
- Cùng return Iterator và ListIterator fail-fast.
- Đều cho phép phần tử null và giống nhau(duplicate elements).

b. Khác nhau:

- Synchronization và thread-safety: Vertor là synchronize, còn ArrayList thì không. Các method trong Vector là synchronized sử dụng nhiều trong multi-threaded và môi trường xử lý đồng thời(concurrent).
- Speed và Performance: Vì vector là synchronized nên chậm hơn 1 tí so với ArrayList.
- Enumeration: Vector có thể trả về enumeration, ArrayList thì không.

*** Lưu ý:** Nên sử dụng ArrayList hơn là Vector nếu không cần synchronize.

2.4. PriorityQueue: lưu trữ các phần tử trong nội bộ theo trật tự tự nhiên của các phần tử (nếu các phần tử này là kiểu **Comparable**), hoặc theo một **Comparator** (bộ so sánh) được cài đặt cho **PriorityQueue**.

2.5. Sự khác nhau giữa HashMap, TreeMap, Hashtable và LinkedHashMap

- HashMap được thực hiện như một bảng băm, và không có thứ tự về các khóa hoặc giá trị.
- TreeMap được thực hiện dựa trên cấu trúc cây và được sắp xếp theo khóa.
- LinkedHashMap giữ nguyên lệnh chèn
- Hashtable được đồng bộ, ngược với HashMap