**[JAVA COLLECTION](https://classroom.google.com/w/Njc0NDQwMTUzOTQ4/tc/Njc0NDQ3MDQ0NzIz)**

**5. Map**

# **1. Nêu ra các đặc điểm map Interface?**

**Lưu trữ theo cặp khóa-giá trị**:

* Mỗi mục trong Map được lưu trữ dưới dạng một cặp khóa-giá trị. Mỗi khóa là duy nhất, nhưng các giá trị có thể trùng lặp.
* Khóa là duy nhất, có nghĩa là không thể có hai cặp có cùng khóa trong một Map. Nếu bạn cố gắng chèn một cặp với khóa đã tồn tại, giá trị của khóa đó sẽ được ghi đè.

**Không tuân theo thứ tự**:

* Các kiểu Map như HashMap không duy trì thứ tự của các phần tử, tức là các cặp khóa-giá trị có thể không được lưu trữ theo thứ tự chèn vào.
* Tuy nhiên, LinkedHashMap duy trì thứ tự chèn và TreeMap duy trì thứ tự sắp xếp tự nhiên của các khóa.

**Các lớp triển khai phổ biến**:

* **HashMap**: Lớp HashMap sử dụng bảng băm để lưu trữ các phần tử và không đảm bảo thứ tự của các phần tử.
* **LinkedHashMap**: Kế thừa từ HashMap nhưng duy trì thứ tự chèn của các cặp khóa-giá trị.
* **TreeMap**: Triển khai NavigableMap, sắp xếp các cặp khóa-giá trị theo thứ tự tự nhiên của khóa hoặc theo bộ so sánh được chỉ định.

**Không chấp nhận khóa null trong TreeMap**:

* Trong HashMap và LinkedHashMap, khóa null được cho phép (một khóa null duy nhất), nhưng trong TreeMap thì không cho phép sử dụng khóa null vì nó cần sắp xếp các khóa.

**Các phương thức phổ biến**:

* put(K key, V value): Thêm một cặp khóa-giá trị vào bản đồ hoặc thay thế giá trị nếu khóa đã tồn tại.
* get(Object key): Lấy giá trị tương ứng với khóa được chỉ định.
* remove(Object key): Xóa cặp khóa-giá trị với khóa được chỉ định.
* containsKey(Object key): Kiểm tra xem Map có chứa khóa đó không.
* containsValue(Object value): Kiểm tra xem Map có chứa giá trị nào đó không.
* keySet(): Trả về một tập hợp chứa các khóa của Map.
* values(): Trả về một tập hợp chứa các giá trị của Map.
* entrySet(): Trả về một tập hợp chứa các cặp khóa-giá trị của Map.

**Hiệu suất**:

* HashMap có thời gian truy cập trung bình là O(1) cho các thao tác tìm kiếm, chèn, và xóa. Tuy nhiên, trong trường hợp xấu nhất (nếu có nhiều xung đột băm), thời gian truy cập có thể là O(n).
* TreeMap có thời gian truy cập O(log n) do cấu trúc cây đỏ-đen.

**Không đồng bộ (Non-synchronized)**:

* Các triển khai như HashMap và TreeMap không đồng bộ. Nếu bạn muốn sử dụng Map trong môi trường đa luồng, bạn có thể sử dụng Collections.synchronizedMap() để tạo ra một Map an toàn cho các thao tác đồng thời.

# **2. Nêu ra các đặc điểm HashMap?**

### **1. Lưu trữ theo cặp khóa-giá trị**

* HashMap lưu trữ dữ liệu dưới dạng các cặp khóa-giá trị. Mỗi khóa là duy nhất và ánh xạ tới một giá trị tương ứng.
* Nếu một khóa đã tồn tại trong HashMap, việc thêm một cặp khóa-giá trị mới với khóa đó sẽ ghi đè lên giá trị cũ.

### **2. Khóa không được trùng lặp, giá trị có thể trùng lặp**

* Trong HashMap, các khóa không được phép trùng lặp, nhưng các giá trị thì có thể trùng lặp. Một khóa chỉ có thể ánh xạ tới duy nhất một giá trị.

### **3. Cho phép khóa null và giá trị null**

* HashMap cho phép một khóa null duy nhất. Điều này có nghĩa là bạn có thể lưu trữ một cặp với khóa là null.
* Ngoài ra, nhiều giá trị null có thể tồn tại trong HashMap.

### **4. Không bảo toàn thứ tự các phần tử**

* HashMap không đảm bảo thứ tự của các phần tử (không tuân theo thứ tự chèn vào). Thứ tự của các cặp khóa-giá trị có thể thay đổi khi thêm hoặc xóa các phần tử.

### **5. Hiệu suất cao**

* HashMap cung cấp thời gian truy cập trung bình là O(1) cho các thao tác tìm kiếm, chèn và xóa. Điều này là nhờ vào cấu trúc bảng băm mà nó sử dụng.
* Trong trường hợp xấu nhất (nhiều xung đột băm), hiệu suất có thể giảm xuống O(n), nhưng điều này hiếm khi xảy ra nếu hàm băm tốt.

### **6. Không đồng bộ (Non-synchronized)**

* HashMap không phải là một lớp đồng bộ, có nghĩa là nó không an toàn trong môi trường đa luồng. Nếu nhiều luồng truy cập đồng thời vào HashMap, có thể xảy ra các vấn đề về dữ liệu.
* Nếu cần sử dụng HashMap trong môi trường đa luồng, có thể sử dụng Collections.synchronizedMap() để tạo ra phiên bản đồng bộ của HashMap.

### **7. Sử dụng bảng băm**

* HashMap sử dụng bảng băm (hash table) để lưu trữ các phần tử. Một giá trị băm (hash value) được tính toán từ khóa và giá trị này xác định vị trí của cặp khóa-giá trị trong bảng băm.
* Trong trường hợp xảy ra xung đột (nhiều khóa có cùng giá trị băm), HashMap sử dụng danh sách liên kết (linked list) hoặc cây đỏ-đen (từ Java 8 trở đi) để xử lý xung đột.

### **8. Load factor và capacity**

* **Capacity (Dung lượng)**: Đây là số lượng "bucket" trong HashMap. Khi số lượng phần tử vượt quá ngưỡng này, HashMap sẽ tăng dung lượng và sắp xếp lại các phần tử.
* **Load factor**: Đây là thước đo để xác định khi nào HashMap cần mở rộng (rehash). Mặc định, load factor là 0.75, nghĩa là khi bảng băm đạt 75% dung lượng thì sẽ tiến hành mở rộng.

### **9. Các phương thức quan trọng**

* put(K key, V value): Thêm hoặc cập nhật một cặp khóa-giá trị vào HashMap.
* get(Object key): Lấy giá trị tương ứng với khóa được chỉ định.
* remove(Object key): Xóa một cặp khóa-giá trị theo khóa được chỉ định.
* containsKey(Object key): Kiểm tra xem HashMap có chứa khóa đó không.
* containsValue(Object value): Kiểm tra xem HashMap có chứa giá trị đó không.
* keySet(): Trả về một tập hợp chứa tất cả các khóa trong HashMap.
* values(): Trả về một tập hợp chứa tất cả các giá trị trong HashMap.
* entrySet(): Trả về một tập hợp chứa tất cả các cặp khóa-giá trị trong HashMap.

### **10. Không hỗ trợ Iterator an toàn**

* Nếu bạn thay đổi HashMap trong khi sử dụng iterator mà không dùng các phương thức của iterator (ví dụ, thêm/xóa phần tử), sẽ gây ra ngoại lệ ConcurrentModificationException.

### **11. Ứng dụng thực tế**

* HashMap thường được sử dụng trong các tình huống yêu cầu tra cứu nhanh như lưu trữ bảng tra cứu, bộ nhớ đệm (cache), quản lý phiên (session), và nhiều ứng dụng khác trong lập trình.

# **3. Nêu ra các đặc điểm LinkedHashMap?**

LinkedHashSet là một lớp trong Java Collection Framework kế thừa từ HashSet và thực hiện giao diện Set. Dưới đây là một số đặc điểm chính của LinkedHashSet:

1. **Bảo toàn thứ tự chèn**:
   * LinkedHashSet duy trì thứ tự chèn của các phần tử. Điều này có nghĩa là khi bạn lặp qua LinkedHashSet, các phần tử sẽ được trả về theo thứ tự mà chúng đã được thêm vào.
2. **Không cho phép phần tử trùng lặp**:
   * Giống như HashSet, LinkedHashSet không cho phép các phần tử trùng lặp. Mỗi phần tử trong LinkedHashSet là duy nhất.
3. **Tốc độ truy cập nhanh**:
   * LinkedHashSet sử dụng một bảng băm để lưu trữ các phần tử, do đó thời gian truy cập (thêm, xóa, và kiểm tra phần tử) trung bình là O(1). Tuy nhiên, vì nó duy trì danh sách liên kết để bảo toàn thứ tự, nên việc lưu trữ có thể tốn nhiều bộ nhớ hơn so với HashSet.
4. **Cho phép một giá trị null**:
   * LinkedHashSet cho phép một giá trị null, giống như HashSet. Tuy nhiên, chỉ có thể có một phần tử null duy nhất trong tập hợp.
5. **Không đồng bộ**:
   * LinkedHashSet không đồng bộ. Nếu bạn muốn sử dụng nó trong môi trường đa luồng, bạn cần phải đồng bộ hóa nó thủ công hoặc sử dụng Collections.synchronizedSet(Set<T> s).
6. **Hỗ trợ các thao tác tập hợp**:
   * LinkedHashSet hỗ trợ các phép toán tập hợp như giao, hợp, và hiệu thông qua các phương thức như retainAll(Collection<?> c), addAll(Collection<? extends E> c), và removeAll(Collection<?> c).
7. **Khả năng mở rộng**:
   * Giống như HashSet, LinkedHashSet cũng có thể tự động mở rộng kích thước khi cần thiết.
8. **Hiệu suất tốt cho các tình huống cần duy trì thứ tự**:
   * LinkedHashSet là lựa chọn lý tưởng khi bạn cần một tập hợp duy nhất mà cần duy trì thứ tự của các phần tử trong quá trình thêm và xóa.

# **4. Nêu ra các đặc điểm TreeMap?**

TreeMap là một lớp trong Java, thuộc thư viện Collection Framework và là một phần của giao diện NavigableMap. TreeMap lưu trữ các cặp khóa-giá trị theo thứ tự sắp xếp và cung cấp các tính năng tìm kiếm, sắp xếp dữ liệu dựa trên khóa một cách hiệu quả. Dưới đây là các đặc điểm chính của TreeMap:

### **1. Lưu trữ theo cặp khóa-giá trị**

* Giống như các lớp khác trong họ Map, TreeMap lưu trữ dữ liệu dưới dạng các cặp khóa-giá trị, với mỗi khóa ánh xạ đến một giá trị duy nhất.
* Nếu một khóa đã tồn tại, giá trị tương ứng sẽ được ghi đè bởi giá trị mới.

### **2. Thứ tự tự nhiên hoặc theo comparator**

* TreeMap duy trì các phần tử theo **thứ tự sắp xếp tự nhiên** của khóa (thường là thứ tự tăng dần đối với các loại dữ liệu số hoặc thứ tự từ điển đối với chuỗi).
* Bạn cũng có thể chỉ định một bộ so sánh (custom comparator) khi khởi tạo TreeMap để thay đổi cách sắp xếp các phần tử theo các quy tắc tùy chỉnh.

### **3. Không cho phép khóa null**

* TreeMap không cho phép sử dụng **khóa null** vì nó cần thực hiện các phép so sánh để sắp xếp các phần tử. Tuy nhiên, giá trị có thể là null.

### **4. Sử dụng cấu trúc cây đỏ-đen (Red-Black Tree)**

* TreeMap được triển khai dựa trên **cấu trúc cây đỏ-đen**, một dạng cây nhị phân cân bằng. Nhờ vào cấu trúc này, TreeMap có thể thực hiện các thao tác như thêm, xóa và tìm kiếm các phần tử với thời gian trung bình là O(log n).

### **5. Duy trì thứ tự sắp xếp các phần tử**

* Khác với HashMap hay LinkedHashMap, TreeMap luôn sắp xếp các phần tử dựa trên khóa của chúng. Khi bạn lặp qua TreeMap, các phần tử sẽ xuất hiện theo thứ tự được sắp xếp (tự nhiên hoặc theo comparator đã cung cấp).

### **6. Không đồng bộ (Non-synchronized)**

* TreeMap không đồng bộ, tức là không an toàn cho các thao tác đa luồng. Nếu bạn muốn sử dụng TreeMap trong môi trường đa luồng, bạn có thể sử dụng Collections.synchronizedSortedMap() để tạo phiên bản đồng bộ của nó.

### **7. Các phương thức của NavigableMap và SortedMap**

* Là một phần của NavigableMap và SortedMap, TreeMap cung cấp nhiều phương thức bổ sung để thao tác với dữ liệu theo thứ tự:
  + firstKey(): Trả về khóa đầu tiên (nhỏ nhất) trong TreeMap.
  + lastKey(): Trả về khóa cuối cùng (lớn nhất) trong TreeMap.
  + ceilingKey(K key): Trả về khóa nhỏ nhất lớn hơn hoặc bằng key.
  + floorKey(K key): Trả về khóa lớn nhất nhỏ hơn hoặc bằng key.
  + higherKey(K key): Trả về khóa nhỏ nhất lớn hơn key.
  + lowerKey(K key): Trả về khóa lớn nhất nhỏ hơn key.
  + subMap(K fromKey, K toKey): Trả về một bản con của TreeMap chứa các phần tử nằm trong khoảng từ fromKey đến toKey.

### **8. Hiệu suất**

* TreeMap có thời gian thực hiện các thao tác cơ bản như chèn, tìm kiếm và xóa với độ phức tạp O(log n), nhờ sử dụng cấu trúc cây đỏ-đen.
* Do cần duy trì cấu trúc cân bằng và thứ tự sắp xếp của cây, TreeMap có thể chậm hơn HashMap hoặc LinkedHashMap trong các thao tác tìm kiếm hoặc thêm phần tử. Tuy nhiên, nếu yêu cầu sắp xếp dữ liệu hoặc các thao tác dựa trên thứ tự của khóa, TreeMap sẽ là một lựa chọn tốt.

### **9. Không hỗ trợ thứ tự chèn**

* TreeMap không duy trì thứ tự chèn các phần tử. Thay vào đó, thứ tự của các phần tử luôn tuân theo thứ tự khóa được sắp xếp.

### **10. Ứng dụng thực tế**

* TreeMap thường được sử dụng trong các tình huống cần duy trì dữ liệu theo thứ tự sắp xếp và cung cấp khả năng tìm kiếm hiệu quả dựa trên các phép so sánh khóa.
* Ví dụ, TreeMap có thể được sử dụng để lưu trữ bảng xếp hạng, lịch sự kiện, hoặc các hệ thống quản lý giao dịch yêu cầu dữ liệu luôn được sắp xếp.

### **11. Phương thức quan trọng**

* put(K key, V value): Thêm hoặc cập nhật một cặp khóa-giá trị vào TreeMap.
* get(Object key): Trả về giá trị tương ứng với khóa đã chỉ định.
* remove(Object key): Xóa một cặp khóa-giá trị với khóa đã chỉ định.
* containsKey(Object key): Kiểm tra xem TreeMap có chứa khóa đó hay không.
* containsValue(Object value): Kiểm tra xdem TreeMap có chứa giá trị nào đó không.
* firstEntry(), lastEntry(): Lấy cặp khóa-giá trị đầu tiên hoặc cuối cùng trong bản đồ.
* pollFirstEntry(), pollLastEntry(): Lấy và xóa cặp khóa-giá trị đầu tiên hoặc cuối cùng.
* descendingMap(): Trả về một bản đồ đảo ngược với thứ tự ngược lại của bản gốc.