1. Các đặc điểm Map Interface**.**

* Map lưu trữ dữ liệu dưới dạng cặp key - value , trong dó key là duy nhất và ánh xạ đến 1 value duy nhất.
* Key là duy nhất và không có key trùng lặp trong 1 map .
* Không quan tâm đến thứ tự khi lưu trữ phần tử.
* Có thể lưu giá trị null cho key và value.

1. Class triển khai Map Interface và trường hợp sử dụng.

2.1. HashMap

* Mỗi key trong hashmap là duy nhất , nếu lưu cùng key thì map sẽ overide lại.
* Cho phép lưu giá trị null ở cả key và value.
* Không an toàn trong môi trường đa luồng .
* Cơ chế triển khai :
  + Thêm phần tử :
    - Bước 1: Khi gọi phương thức put(key, value), HashMap sẽ sử dụng hàm băm (hashCode()) của đối tượng khóa để tính toán chỉ số (index) trong mảng bucket của HashMap.
    - Bước 2: Nếu keyđã tồn tại trong bucket, nó sẽ ghi đè giá trị cũ.
    - Bước 3: Phần tử được thêm vào mảng bucket tại vị trí tương ứng với chỉ số băm.
      * Nếu không có xung đột, phần tử được thêm trực tiếp vào bucket.
      * Nếu có xung đột, phần tử sẽ được thêm vào danh sách liên kết hoặc cây nhị phân.
  + Xoá phần tử :
    - Bước 1: HashMap sử dụng hashCode của khóa để tính chỉ số của bucket chứa khóa đó.
    - Bước 2: Tìm phần tử trong bucket: HashMap sẽ duyệt qua danh sách liên kết hoặc cây nhị phân trong bucket để tìm phần tử có khóa cần xóa.
    - Bước 3: Xóa phần tử: Sau khi tìm thấy phần tử có khóa khớp, phần tử sẽ được xóa khỏi danh sách liên kết hoặc cây nhị phân tại vị trí bucket tương ứng.
  + Tìm kiếm phần tử :
    - Bước 1: HashMap sử dụng hashCode để tính chỉ số của bucket chứa khóa cần tìm kiếm.
    - Bước 2: HashMap sẽ duyệt qua danh sách liên kết hoặc cây nhị phân trong bucket tương ứng. Nếu tìm thấy phần tử có khóa khớp, giá trị của phần tử đó sẽ được trả về.
    - Bước 3: Nếu tìm thấy phần tử có khóa khớp, phương thức get() sẽ trả về giá trị tương ứng. Nếu không, trả về null
  + Duyệt mảng :
    - Để duyệt tất cả các phần tử trong HashMap, Java cung cấp các phương thức như entrySet(), keySet() và values():
    - keySet(): Trả về tập hợp tất cả các khóa trong HashMap.
    - entrySet(): Trả về tập hợp tất cả các cặp khóa-giá trị trong HashMap.
    - values(): Trả về tập hợp tất cả các giá trị trong HashMap.
    - Duyệt qua toàn bộ HashMap thường sẽ duyệt từng bucket một, sau đó duyệt danh sách liên kết hoặc cây nhị phân trong mỗi bucket.
* Trường hợp sử dụng : HashMap rất phù hợp trong các tình huống cần hiệu suất cao và không yêu cầu đồng bộ hóa hoặc thứ tự phần tử.

2.2. LinkedHashMap

* LinkedHashMap duy trì thứ tự chèn của các phần tử, giúp dễ dàng truy xuất theo thứ tự các phần tử đã thêm vào.
* LinkedHashMap còn có tùy chọn để sắp xếp các mục dựa trên thứ tự truy cập.
* Hiệu suất tốt vì được xây dựng dựa trên hashCode.
* Làm cache hiệu quả: Như đã đề cập, với cơ chế dựa trên thứ tự truy cập, LinkedHashMap có thể được sử dụng để xây dựng bộ nhớ đệm (cache) .
* Hiệu năng chậm hơn hash map vì phải duy trì sắp xếp và truy cập .
* Cơ chế hoạt động
  + Thêm phần tử
    - LinkedHashMap sử dụng cơ chế băm giống như HashMap.
    - Mỗi mục (entry) được lưu trong bảng băm cũng là một nút của danh sách liên kết kép, chứa liên kết đến phần tử trước và sau. Điều này giúp duy trì thứ tự thêm hoặc truy cập .
  + Xoá phần tử
    - Tương tự như hashMap.
    - Sau khi xoá phần tử cũng xoá liên kết giữa các phần tử.
  + Tìm kiếm phần tử
    - LinkedHashMap sử dụng bảng băm để tìm kiếm phần tử dựa trên khóa.
  + Duyệt mảng
    - Việc duyệt qua các phần tử của LinkedHashMap tuân theo thứ tự của danh sách liên kết kép. Điều này có nghĩa là các phần tử sẽ được duyệt theo thứ tự thêm hoặc thứ tự truy cập (access order), tuỳ thuộc vào cách LinkedHashMap được cấu hình.
* Trường hợp sử dụng : khi cần vừa truy xuất nhanh vừa duy trì thứ tự chèn hoặc truy cập

2.3. TreeMap

* TreeMap lưu trữ các phần tử theo thứ tự tự nhiên của các khóa (dựa trên việc sắp xếp tự nhiên hoặc dựa trên Comparator được cung cấp).
* TreeMap thích hợp khi bạn cần lấy dữ liệu theo thứ tự hoặc cần các thao tác phạm vi (ví dụ: lấy tất cả các khóa trong một phạm vi nhất định.
* Tiêu tốn nhiều bộ nhớ .
* không hỗ trọ kiểu dữ liêuh null.
* Cơ chế hoạt động
  + Thêm phần tử
    - Bước 1: Tìm vị trí chèn: TreeMap thực hiện việc tìm kiếm nhị phân dựa trên khóa. Bắt đầu từ root node, nó so sánh khóa cần thêm với khóa hiện tại ở mỗi nút:
    - Nếu khóa mới nhỏ hơn khóa nút hiện tại, di chuyển về nhánh con trái.
    - Nếu khóa lớn hơn, di chuyển về nhánh con phải.
    - Nếu tìm thấy một khóa tương ứng (khóa đã tồn tại), nó sẽ ghi đè giá trị mới vào vị trí của giá trị cũ.
    - Bước 2: Chèn vào vị trí thích hợp: Khi đến một nút lá (nơi không còn nút con nào), nút mới sẽ được chèn.
    - Bước 3: Cân bằng lại cây: Sau khi chèn, TreeMap thực hiện các thao tác cân bằng lại tree.
  + Xoá phần tử
    - **Bước 1: Tìm phần tử cần xóa**: Bắt đầu từ gốc của cây, TreeMap sử dụng tìm kiếm nhị phân để tìm khóa tương ứng. Quá trình này giống với việc tìm kiếm trong cây nhị phân tìm kiếm.
    - **Bước 2: Xóa phần tử**: Có ba trường hợp cần xử lý:
      * **Node không có con**: **Node** được xóa trực tiếp khỏi cây.
      * **Node có một con**: **Node** này được thay thế bằng con của nó.
      * **Node có hai con**: **Node** cần xóa được thay thế bằng nút nhỏ nhất trong cây con bên phải (hoặc nút lớn nhất trong cây con bên trái).
    - **Bước 3: Cân bằng lại cây**: Sau khi xóa, các thao tác xoay cây và đổi màu được thực hiện để đảm bảo tree vẫn cân bằng.
  + Tìm kiếm phần tử
    - TreeMap thực hiện tìm kiếm nhị phân để tìm khóa tương ứng:
    - Bắt đầu từ gốc, nó so sánh khóa cần tìm với khóa của mỗi nút.
    - Nếu khóa nhỏ hơn, tiếp tục tìm kiếm ở nhánh trái.
    - Nếu khóa lớn hơn, tiếp tục tìm kiếm ở nhánh phải.
    - Nếu khóa trùng khớp, trả về giá trị tương ứng. Nếu không tìm thấy, trả về null.
  + Duyệt phần tử trong mảng :
    - TreeMap sử dụng cách duyệt **in-order traversal** để trả về các phần tử theo thứ tự sắp xếp tự nhiên của các khóa.
    - Các phương thức entrySet(), keySet(), và values() trả về danh sách các phần tử trong cây theo thứ tự in-order:
      * **entrySet()**: Trả về tập hợp các cặp khóa-giá trị dưới dạng Map.Entry<K, V>.
      * **keySet()**: Trả về tập hợp các khóa.
      * **values()**: Trả về danh sách các giá trị.
* Trường hợp sử dụng :
  + Khi cần duy trì thứ tự tự nhiên hoặc sắp xếp tùy chỉnh của các khóa.
  + Khi bạn cần thao tác với một phạm vi của các phần tử, ví dụ như tìm kiếm trong một khoảng khóa nhất định.
  + Khi các thao tác truy xuất theo thứ tự khóa quan trọng đối với bài toán của bạn.