# I.Array

## 1. Nêu hiểu biết về kiểu mảng trong java

Mảng (array) trong Java là một cấu trúc dữ liệu được sử dụng để lưu trữ một tập hợp các giá trị cùng kiểu dữ liệu trong một vùng nhớ liên tiếp. Mỗi phần tử trong mảng được định danh bằng một chỉ số (index) duy nhất và có thể truy cập bằng cách sử dụng chỉ số đó.

-Đặc điểm :

 **Cùng kiểu dữ liệu**: Tất cả các phần tử trong mảng phải có cùng kiểu (như int, String, double, v.v.).

 **Kích thước cố định**: Khi tạo mảng, kích thước (số phần tử) phải được xác định và không thể thay đổi trong quá trình chạy chương trình.

 **Chỉ số (index)**: Phần tử đầu tiên của mảng có chỉ số là 0 và phần tử cuối cùng có chỉ số là n-1 (với n là số phần tử của mảng).

 **Lưu trữ tuần tự trong bộ nhớ**: Các phần tử của mảng được lưu liên tiếp trong bộ nhớ.

## 2.Ưu nhược điểm của kiểu mảng

### Ưu điểm

* Cung cấp phương tiện lưu trữ và truy xuất các phần tử cùng kiểu dữ liệu trong một vùng nhớ liên tiếp, giúp cho việc quản lý dữ liệu trở nên dễ dàng.
* Cho phép truy xuất các phần tử trong mảng theo chỉ số, giúp cho việc truy xuất dữ liệu nhanh chóng và dễ dàng.
* Dễ dàng tạo ra các mảng đa chiều (multidimensional array) để lưu trữ dữ liệu có cấu trúc phức tạp hơn.
* Hỗ trợ các thao tác tính toán trên mảng như sắp xếp, tìm kiếm, lọc, …

### Nhược điểm

* Mảng có kích thước cố định khi khai báo, nếu cần thêm phần tử mới, ta phải tạo một mảng mới với kích thước lớn hơn và sao chép dữ liệu từ mảng cũ sang mảng mới. Điều này có thể làm tốn thời gian và tài nguyên của hệ thống.
* Không thể chứa các phần tử khác kiểu dữ liệu trong cùng một mảng.
* Không có cơ chế tự động quản lý vùng nhớ của các phần tử trong mảng. Việc không giải phóng bộ nhớ tạo ra các vấn đề liên quan đến rò rỉ bộ nhớ (memory leak).
* Khó khăn trong việc thực hiện các thao tác chèn, xóa phần tử giữa mảng.

# II.List Interface

## Nêu ra các đặc điểm **List Interface**

1.  **Duy trì thứ tự chèn (insertion order):**  
   Các phần tử trong List được sắp xếp theo thứ tự mà chúng được thêm vào. Khi duyệt qua danh sách, thứ tự sẽ giống với thứ tự đã thêm.
2.  **Cho phép phần tử trùng lặp:**  
   Một List có thể chứa nhiều phần tử có cùng giá trị. Điều này khác với các interface như Set, không cho phép phần tử trùng lặp.
3.  **Hỗ trợ truy cập theo chỉ mục (index)**
4. Có thể chứa giá trị null

## 2. Kể ra các **class**triển khai từ **List Interface**

**-Các class triển khai từ list interface :**

ArrayList :

* Khi cần **truy cập phần tử theo chỉ mục** nhanh chóng.
* Khi **số lần thêm/xóa ít** và **đọc nhiều**.
* Không yêu cầu đồng bộ hóa:

LinkedList:

 Khi cần **thêm/xóa phần tử thường xuyên**, đặc biệt ở đầu/cuối.

 Khi **không cần truy cập ngẫu nhiên** nhiều.

Vector:

* Khi cần **danh sách đồng bộ** cho ứng dụng **đa luồng**.
* Tuy nhiên, hiện nay ít được sử dụng do hiệu suất thấp hơn.

Stack:

* Khi cần quản lý dữ liệu theo kiểu **LIFO** (ví dụ: Undo/Redo, trình duyệt web, xử lý đệ quy).

CopyOnWriteArrayList:

* Khi danh sách được **đọc nhiều** và **cập nhật ít** trong môi trường đa luồng.

# III.Queue Interface

## Nêu các đặc điểm của Queue Interface, Dequeue Interface

1. **Queue Interface** trong Java

Queue là một **interface** trong Java Collections Framework, mở rộng từ Collection. Nó mô tả một **hàng đợi (queue)**, nơi các phần tử được thêm vào cuối hàng và loại bỏ từ đầu hàng. Hàng đợi thường tuân theo cơ chế **FIFO (First-In-First-Out)**.

### **Đặc điểm chính của Queue Interface:**

* **Thứ tự FIFO:** Phần tử nào được thêm trước thì sẽ được lấy ra đầu tiên (trừ các biến thể như PriorityQueue).
* **Phương thức chính:**
  + **add(E e)**: Thêm một phần tử vào cuối hàng đợi. Nếu không thể thêm, nó sẽ ném ngoại lệ.
  + **offer(E e)**: Thêm phần tử vào hàng đợi, nhưng **không ném ngoại lệ** nếu không thêm được (trả về false).
  + **remove()**: Loại bỏ phần tử đầu tiên của hàng đợi, ném ngoại lệ nếu hàng rỗng.
  + **poll()**: Loại bỏ phần tử đầu tiên, trả về null nếu hàng rỗng.
  + **element()**: Trả về phần tử đầu tiên mà **không xóa**, ném ngoại lệ nếu hàng rỗng.
  + **peek()**: Trả về phần tử đầu tiên mà không xóa, trả về null nếu hàng rỗng.

2. **Deque Interface** trong Java

Deque (Double-Ended Queue) là một **interface** mở rộng từ Queue, cho phép chèn và loại bỏ phần tử từ **cả hai đầu** (đầu và cuối hàng). Điều này cung cấp cả hai cơ chế **FIFO** và **LIFO (Last-In-First-Out)**.

### **Đặc điểm chính của Deque Interface:**

* **Chèn và loại bỏ từ cả hai đầu:** Không chỉ thao tác theo thứ tự FIFO mà còn có thể sử dụng như một **stack** (ngăn xếp) với LIFO.
* **Phương thức chính:**
  + **Chèn ở đầu:**
    - addFirst(E e), offerFirst(E e): Thêm phần tử vào đầu hàng.
  + **Chèn ở cuối:**
    - addLast(E e), offerLast(E e): Thêm phần tử vào cuối hàng.
  + **Loại bỏ ở đầu:**
    - removeFirst(), pollFirst(): Loại bỏ phần tử đầu tiên.
  + **Loại bỏ ở cuối:**
    - removeLast(), pollLast(): Loại bỏ phần tử cuối cùng.
  + **Truy cập phần tử:**
    - getFirst(), peekFirst(): Lấy phần tử đầu tiên mà không xóa.
    - getLast(), peekLast(): Lấy phần tử cuối cùng mà không xóa.

# IV.Set Interface

## Nêu ra các đặc điểm **Set Interface**

1. **Không chứa phần tử trùng lặp**
   * Bất kỳ phần tử nào đã tồn tại trong tập hợp sẽ không được thêm lại.
   * So sánh phần tử dựa trên phương thức **equals()** (và đôi khi là hashCode() trong một số triển khai như HashSet).
2. **Không đảm bảo thứ tự**
   * Thứ tự lưu trữ phần tử có thể không được duy trì (như trong HashSet). Một số triển khai như TreeSet sắp xếp phần tử theo thứ tự tự nhiên hoặc tùy chỉnh.
3. **Phương thức chính của Set Interface:**
   * **add(E e)**: Thêm phần tử vào tập hợp, trả về true nếu phần tử được thêm, false nếu nó đã tồn tại.
   * **remove(Object o)**: Xóa phần tử khỏi tập hợp (nếu tồn tại).
   * **contains(Object o)**: Kiểm tra xem tập hợp có chứa phần tử đó không.
   * **size()**: Trả về số phần tử hiện tại trong tập hợp.
   * **isEmpty()**: Kiểm tra xem tập hợp có rỗng không.

## Kể ra các **class**triển khai từ **Set Interface,** Phân biệt rõ trường hợp sử dụng của từng **class**đó



# V.Map Interface

## Nêu ra các đặc điểm **Map Interface**

### **Đặc điểm chính của Map Interface**

1. **Cấu trúc cặp khóa-giá trị**
   * Mỗi phần tử trong Map là một cặp khóa và giá trị, trong đó mỗi khóa là duy nhất.
   * Ví dụ: Map<Integer, String> có thể lưu trữ các cặp như (1, "Một"), (2, "Hai").
2. **Không cho phép khóa trùng lặp**
   * Mỗi khóa chỉ có thể ánh xạ đến một giá trị duy nhất. Nếu thêm một khóa đã tồn tại, giá trị của khóa đó sẽ bị ghi đè.
   * Ví dụ: Nếu bạn thêm (1, "Một") và sau đó (1, "Hai"), giá trị cho khóa 1 sẽ là "Hai".
3. **Cho phép giá trị trùng lặp**
   * Nhiều khóa có thể ánh xạ đến cùng một giá trị.
   * Ví dụ: Cả khóa 1 và 2 có thể đều ánh xạ đến giá trị "Hai".
4. **Các phương thức chính trong Map Interface**
   * **put(K key, V value)**: Thêm cặp khóa-giá trị vào bản đồ.
   * **get(Object key)**: Truy xuất giá trị tương ứng với khóa.
   * **remove(Object key)**: Xóa cặp khóa-giá trị khỏi bản đồ.
   * **containsKey(Object key)**: Kiểm tra xem khóa có tồn tại trong bản đồ không.
   * **containsValue(Object value)**: Kiểm tra xem giá trị có tồn tại trong bản đồ không.
   * **size()**: Trả về số cặp khóa-giá trị trong bản đồ.
   * **isEmpty()**: Kiểm tra xem bản đồ có rỗng không.
5. **Không đảm bảo thứ tự**
   * Thứ tự của các cặp khóa-giá trị không được đảm bảo, trừ khi sử dụng một số triển khai đặc biệt.

### Kể ra các class triển khai từ Map Interface, Phân biệt rõ trường hợp sử dụng của từng class đó

