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

**1. Queue**

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

### **1. Thao Tác Đối Tượng FIFO (First-In-First-Out)**

* Các phần tử trong Queue được xử lý theo thứ tự mà chúng được thêm vào, tức là phần tử được thêm vào trước sẽ được loại bỏ trước (FIFO).

### **2. Các Phương Thức Cơ Bản**

* add(E e): Thêm phần tử vào cuối hàng đợi. Nếu hàng đợi đầy (nếu giới hạn kích thước), nó sẽ ném IllegalStateException.
* offer(E e): Thêm phần tử vào cuối hàng đợi, nhưng không ném ngoại lệ nếu hàng đợi đầy. Thay vào đó, nó sẽ trả về false.
* remove(): Loại bỏ và trả về phần tử đầu tiên trong hàng đợi. Nếu hàng đợi rỗng, nó sẽ ném NoSuchElementException.
* poll(): Loại bỏ và trả về phần tử đầu tiên trong hàng đợi, hoặc trả về null nếu hàng đợi rỗng.
* peek(): Trả về phần tử đầu tiên mà không loại bỏ nó, hoặc trả về null nếu hàng đợi rỗng.

### **3. Các Loại Hàng Đợi**

* Queue có thể có nhiều loại khác nhau, như:
  + **LinkedList**: Sử dụng danh sách liên kết, cho phép thao tác nhanh chóng ở cả hai đầu.
  + **PriorityQueue**: Cho phép các phần tử được xử lý theo thứ tự ưu tiên thay vì theo thứ tự thêm.
  + **ArrayDeque**: Cung cấp một hàng đợi được triển khai bằng mảng, hỗ trợ thêm và xóa ở cả hai đầu.

### **4. Không Phải Lớp Thực Thi**

* Queue là một interface, do đó, nó không thể được khởi tạo trực tiếp. Thay vào đó, bạn sẽ sử dụng các lớp thực thi như LinkedList, PriorityQueue, hoặc ArrayDeque.

### **5. Hỗ Trợ Tính Năng Linh Hoạt**

* Một số hàng đợi có thể có giới hạn về kích thước, trong khi các hàng đợi khác có thể mở rộng. Điều này cung cấp cho lập trình viên tính linh hoạt trong việc quản lý các phần tử trong hàng đợi.

### **6. Đặc Điểm Khác**

* **Không đồng bộ**: Hầu hết các hàng đợi trong Java Collections Framework không đồng bộ. Nếu bạn cần hàng đợi đồng bộ, bạn có thể sử dụng java.util.concurrent package, như BlockingQueue.
* **Kiểm Soát Chất Lượng Dữ Liệu**: Queue có thể được sử dụng trong các kịch bản xử lý dữ liệu như hàng đợi tác vụ (task queue) trong ứng dụng đa luồng, nơi các tác vụ được xử lý theo thứ tự.

### **7. Thao Tác Nhanh Chóng**

* Queue cung cấp các thao tác nhanh chóng để thêm và xóa phần tử từ đầu hoặc cuối, giúp cải thiện hiệu suất cho các ứng dụng yêu cầu xử lý hàng đợi.

# **2. Nêu ra các đặc điểm PriorityQueue?**

PriorityQueue là một trong những cấu trúc dữ liệu trong Java Collections Framework. Dưới đây là các đặc điểm chính của PriorityQueue:

### **1. Thứ Tự Ưu Tiên**

* PriorityQueue tổ chức các phần tử theo thứ tự ưu tiên, nghĩa là phần tử có độ ưu tiên cao hơn sẽ được xử lý trước.
* Thứ tự ưu tiên có thể được xác định bởi một trình so sánh (comparator) hoặc theo thứ tự tự nhiên của các phần tử (nếu các phần tử implement Comparable).

### **2. Không Tuân Theo Thứ Tự Thêm**

* Phần tử trong PriorityQueue không được xử lý theo thứ tự mà chúng được thêm vào. Thay vào đó, phần tử với độ ưu tiên cao nhất sẽ được loại bỏ đầu tiên.

### **3. Cấu Trúc Dữ Liệu**

* PriorityQueue được triển khai bằng cách sử dụng một mảng động (dynamic array) hoặc một heap (thường là min-heap hoặc max-heap).
* Thời gian cho các thao tác thêm và loại bỏ phần tử là O(log n), nơi n là số lượng phần tử trong hàng đợi.

### **4. Khả Năng Thay Đổi**

* PriorityQueue có khả năng thay đổi độ ưu tiên của các phần tử khi chúng được thêm vào hoặc loại bỏ.
* Bạn có thể thay đổi độ ưu tiên của phần tử bằng cách xóa nó và thêm lại với độ ưu tiên mới.

### **5. Không Phải Là Một Hàng Đợi Đồng Bộ**

* PriorityQueue không đồng bộ, điều này có nghĩa là nó không an toàn khi sử dụng trong môi trường đa luồng. Nếu bạn cần một hàng đợi đồng bộ, bạn có thể sử dụng PriorityBlockingQueue trong java.util.concurrent.

### **6. Chấp Nhận Phần Tử Null**

* PriorityQueue không cho phép thêm phần tử null. Nếu bạn cố gắng thêm null, nó sẽ ném một NullPointerException.

### **7. Tính Năng Linh Hoạt**

* Bạn có thể khởi tạo một PriorityQueue mà không có trình so sánh, trong trường hợp đó, các phần tử sẽ phải implement Comparable.
* Nếu bạn cung cấp một Comparator, PriorityQueue sẽ sử dụng nó để xác định thứ tự ưu tiên.

### **8. Hỗ Trợ Các Kiểu Dữ Liệu Khác Nhau**

* PriorityQueue có thể chứa các kiểu dữ liệu nguyên thủy và đối tượng (object types), miễn là các phần tử có thể được so sánh hoặc bạn đã cung cấp một Comparator.

### **9. Không Giới Hạn Kích Thước**

* PriorityQueue không có giới hạn kích thước cố định. Khi số lượng phần tử vượt quá dung lượng hiện tại, nó sẽ tự động mở rộng.

### **10. Thao Tác Đơn Giản**

* Cung cấp các phương thức đơn giản để thao tác với hàng đợi ưu tiên, như offer(E e), poll(), peek(), và remove(Object o).

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

### **1. Thao Tác Hai Đầu**

* Deque cho phép thêm, xóa, và truy cập phần tử từ cả hai đầu, giúp linh hoạt trong việc quản lý dữ liệu.
* Các phương thức cơ bản:
  + addFirst(E e): Thêm phần tử vào đầu deque.
  + addLast(E e): Thêm phần tử vào cuối deque.
  + removeFirst(): Loại bỏ và trả về phần tử đầu tiên.
  + removeLast(): Loại bỏ và trả về phần tử cuối cùng.
  + peekFirst(): Trả về phần tử đầu tiên mà không loại bỏ nó.
  + peekLast(): Trả về phần tử cuối cùng mà không loại bỏ nó.

### **2. Có Thể Hoạt Động Như Hàng Đợi Hoặc Ngăn Xếp**

* Deque có thể được sử dụng như một hàng đợi (FIFO) hoặc một ngăn xếp (LIFO), tùy thuộc vào cách bạn thêm và loại bỏ các phần tử.
* Ví dụ: bạn có thể sử dụng addFirst() và removeFirst() để hoạt động như một ngăn xếp, hoặc sử dụng addLast() và removeFirst() để hoạt động như một hàng đợi.

### **3. Không Giới Hạn Kích Thước**

* Deque không có kích thước cố định. Khi số lượng phần tử vượt quá dung lượng hiện tại, nó sẽ tự động mở rộng.

### **4. Các Lớp Triển Khai**

* Các lớp triển khai phổ biến của Deque bao gồm ArrayDeque và LinkedList.
* ArrayDeque sử dụng mảng động và cung cấp hiệu suất cao hơn cho các thao tác thêm và loại bỏ.
* LinkedList sử dụng danh sách liên kết, cho phép thêm và xóa phần tử nhanh chóng ở cả hai đầu.

### **5. Thao Tác Nhanh Chóng**

* Deque cung cấp các thao tác nhanh chóng để thêm và xóa phần tử từ cả hai đầu, giúp cải thiện hiệu suất cho các ứng dụng yêu cầu thao tác thường xuyên.

### **6. Không Phải Là Một Hàng Đợi Đồng Bộ**

* Tương tự như Queue, Deque không đồng bộ, điều này có nghĩa là nó không an toàn khi sử dụng trong môi trường đa luồng. Nếu bạn cần một hàng đợi đồng bộ, có thể sử dụng BlockingDeque.

### **7. Khả Năng Chấp Nhận Phần Tử Null**

* Deque không cho phép thêm phần tử null. Nếu bạn cố gắng thêm null, nó sẽ ném một NullPointerException.

### **8. Hỗ Trợ Kiểu Dữ Liệu Khác Nhau**

* Deque có thể chứa các kiểu dữ liệu nguyên thủy và đối tượng (object types), miễn là các phần tử có thể được xử lý.

### **9. Thao Tác Đơn Giản**

* Cung cấp các phương thức đơn giản để thao tác với deque, như offerFirst(E e), offerLast(E e), pollFirst(), pollLast(), và các phương thức tương tự.

# **4. Trường hợp sử dụng ArrayDeque, PriorityQueue?**

### **1. PriorityQueue**

* **Trường hợp sử dụng**:
  + **Quản lý tác vụ theo độ ưu tiên**: Sử dụng trong các hệ thống cần xử lý các tác vụ theo độ ưu tiên, như trong lập lịch CPU, nơi các tác vụ quan trọng hơn được xử lý trước.
  + **Thuật toán tìm kiếm đường đi**: Trong các thuật toán như A\* hoặc Dijkstra, PriorityQueue được sử dụng để quản lý các nút cần khám phá, với các nút có chi phí thấp hơn được ưu tiên.
  + **Hệ thống thông báo**: Quản lý thông báo hoặc sự kiện trong các ứng dụng cần xử lý thông báo quan trọng hơn trước.
  + **Thuật toán Huffman coding**: Sử dụng PriorityQueue để xây dựng cây Huffman trong nén dữ liệu.

### **2. ArrayDeque**

* **Trường hợp sử dụng**:
  + **Ngăn xếp và hàng đợi**: ArrayDeque có thể được sử dụng như một ngăn xếp (LIFO) hoặc hàng đợi (FIFO), phù hợp cho các thao tác cần quản lý dữ liệu ở cả hai đầu.
  + **Thao tác nhanh chóng**: Khi cần thực hiện các thao tác thêm và loại bỏ phần tử nhanh chóng từ cả hai đầu, ArrayDeque là một lựa chọn tốt nhờ vào hiệu suất cao hơn so với LinkedList.
  + **Quản lý trạng thái**: Trong trò chơi hoặc ứng dụng đồ họa, ArrayDeque có thể được sử dụng để theo dõi trạng thái của các đối tượng, như các bước quay lại (undo) hoặc các thao tác trước đó.
  + **Quản lý buffer**: Sử dụng để quản lý buffer cho các tác vụ nhập/xuất (I/O), nơi các phần tử được thêm vào và loại bỏ từ hai đầu.