Queue Interface

1.Nêu các đặc điểm của **Queue Interface, Dequeue Interface**

**Queue** (hàng đợi) là một interface trong Java Collections Framework, đại diện cho một danh sách các phần tử được sắp xếp để xử lý. Nó tuân theo nguyên tắc **FIFO (First-In, First-Out)**, nghĩa là phần tử được thêm vào đầu tiên sẽ được lấy ra đầu tiên.

Các phương thức chính của Queue có hai phiên bản: một phiên bản ném ngoại lệ khi thất bại và một phiên bản trả về giá trị đặc biệt (null hoặc false).

* **Thêm phần tử**: add(e) (ném ngoại lệ) và offer(e) (trả về false).
* **Lấy phần tử và xóa**: remove() (ném ngoại lệ) và poll() (trả về null).
* **Xem phần tử**: element() (ném ngoại lệ) và peek() (trả về null).

**Deque** (double-ended queue) là một interface mở rộng từ Queue. Nó cung cấp chức năng của một hàng đợi hai đầu, cho phép thêm và xóa phần tử từ cả hai phía: đầu và cuối.

Các đặc điểm chính của Deque bao gồm:

* **Hỗ trợ LIFO (Last-In, First-Out)**: Mặc dù là một hàng đợi hai đầu, Deque cũng có thể được sử dụng như một **stack** (ngăn xếp), nơi phần tử được thêm vào cuối sẽ được lấy ra đầu tiên (**FIFO**).
* **Linh hoạt**: Deque cung cấp các phương thức để thao tác với cả hai đầu của hàng đợi.
* **Không giới hạn dung lượng** (đối với hầu hết các triển khai): Các lớp như ArrayDeque sẽ tự động mở rộng khi cần.
* **Đa năng hơn Queue:** Cung cấp các phương thức bổ sung như addFirst(), addLast(), peekFirst(), peekLast().
* **Chèn/Xóa hiệu quả ở cả hai đầu:** Hiệu suất thời gian không đổi cho các hoạt động thêm/xóa ở cả hai đầu trong hầu hết các lần triển khai.

Các phương thức chính của Deque thường bắt đầu với first hoặc last để chỉ rõ vị trí thao tác:

* **Thêm phần tử**: addFirst(e), addLast(e).
* **Lấy phần tử và xóa**: removeFirst(), removeLast().
* **Xem phần tử**: peekFirst(), peekLast().

2. Kể ra các **class**triển khai từ **Queue Interface, Dequeue Interface**, phân biệt trường hợp sử dụng tương ứng

**LinkedList**: Lớp này là lựa chọn phổ biến nhất để triển khai **Queue**. Nó hoạt động hiệu quả cho các thao tác thêm và xóa ở đầu và cuối danh sách.

**Trường hợp sử dụng**: Dùng khi cần một hàng đợi tiêu chuẩn (FIFO), ví dụ: xử lý các yêu cầu của người dùng theo thứ tự đến, hàng đợi in ấn.

**PriorityQueue**: Một hàng đợi đặc biệt, nơi các phần tử được sắp xếp theo một thứ tự ưu tiên nhất định, không tuân theo nguyên tắc FIFO.

**Trường hợp sử dụng**: Dùng khi các phần tử cần được xử lý dựa trên mức độ ưu tiên của chúng. Ví dụ: hàng đợi công việc trong hệ điều hành (các tác vụ có độ ưu tiên cao được xử lý trước), hoặc các thuật toán tìm đường đi ngắn nhất như Dijkstra.

**PriorityBlockingQueue** là một hàng đợi không có giới hạn, hỗ trợ nhiều luồng (thread-safe), và sắp xếp các phần tử theo thứ tự ưu tiên giống như PriorityQueue. Nó cũng hỗ trợ các thao tác "chặn" (blocking retrieval), nghĩa là một luồng sẽ chờ đợi nếu hàng đợi rỗng cho đến khi có phần tử mới được thêm vào.

**Trường hợp sử dụng:** khi cần một hàng đợi ưu tiên trong một ứng dụng đa luồng. Nó đặc biệt hữu ích khi nhiều luồng đang thêm các tác vụ với độ ưu tiên khác nhau, và một luồng làm việc cần phải xử lý các tác vụ có độ ưu tiên cao nhất trước. Ví dụ: Một hệ thống lên lịch tác vụ, nhiều luồng thêm các loại tác vụ khác nhau, nhưng chúng không có cùng mức độ quan trọng, cần một luồng làm việc để luôn xử lý các tác vụ khẩn cấp trước.

**ArrayDeque**: Lớp này được khuyến khích sử dụng thay cho LinkedList để làm queue hoặc stack thông thường, vì nó hiệu quả hơn (không an toàn khi đa luồng).

**Trường hợp sử dụng:** Dùng khi cần một cấu trúc dữ liệu vừa là queue (hàng đợi) vừa là stack (ngăn xếp). Ví dụ: lưu trữ lịch sử các trang web đã truy cập (Stack), hoặc xử lý các tác vụ từ hai đầu.

**LinkedList**: Giống như Queue, LinkedList cũng triển khai Deque và có thể hoạt động như một queue hai đầu.

**Trường hợp sử dụng:** Tương tự như ArrayDeque, nhưng có thể kém hiệu quả hơn một chút về hiệu suất trong các tác vụ queue/stack thông thường. LinkedList thích hợp hơn nếu bạn cần thêm hoặc xóa phần tử ở giữa danh sách, mặc dù Deque không tập trung vào việc này.