**Queue Hierarchy in Java**

**Introduction**

* **Queue Interface**: Introduced in Java SE 5 as part of the Collections Framework.
* **Deque Interface**: Extended the Queue interface in Java SE 6.
* **Hierarchy**: Queue extends Collection, and Deque extends Queue.

**Stack vs Queue**

* **Stack (LIFO - Last In, First Out)**
  + push(element): Adds an element to the stack.
  + pop(): Removes the most recently added element.
* **Queue (FIFO - First In, First Out)**
  + offer(element) / add(element): Adds an element to the queue.
  + poll(): Removes the oldest element.
  + peek(): Views the next element to be removed without removing it.

**Why These Structures Matter?**

* **Simplicity**: Easy to implement, even in early computing.
* **Usefulness**: Many algorithms rely on stacks (e.g., recursion, parsing) and queues (e.g., scheduling, BFS).

**Modeling Queues and Stacks in Java**

**Interfaces in Collections Framework**

* **Queue Interface**: Models a FIFO queue.
* **Deque Interface**: Models a double-ended queue (supports push, pop, poll, and peek from both ends, acting as both a queue and a stack).

**Queues & Stacks in Concurrent Programming**

* Extended by **BlockingQueue, BlockingDeque, and TransferQueue** for concurrency control (outside this scope).

**Handling Edge Cases in Queues**

A queue can be:

1. **Full** (Cannot accept more elements).
2. **Empty** (Cannot return an element).

**Handling Corner Cases in the Queue Interface**

| **Operation** | **Method** | **Behavior When Full/Empty** |
| --- | --- | --- |
| **Push** | add(element) | Throws IllegalStateException when full |
|  | offer(element) | Returns false when full |
| **Poll** | remove() | Throws NoSuchElementException when empty |
|  | poll() | Returns false when empty |
| **Peek** | element() | Throws NoSuchElementException when empty |
|  | peek() | Returns null when empty |

**Modeling LIFO Stacks and FIFO Queues with Deque**

**Introduction**

* **Deque Interface** (Java SE 6): Extends Queue and introduces a new naming convention.
* Provides methods for **both FIFO and LIFO** operations.

**FIFO Operations in Deque**

| **Operation** | **Method** | **Behavior When Full/Empty** |
| --- | --- | --- |
| **Push** | addLast(element) | Throws IllegalStateException when full |
|  | offerLast(element) | Returns false when full |
| **Poll** | removeFirst() | Throws NoSuchElementException when empty |
|  | pollFirst() | Returns null when empty |
| **Peek** | getFirst() | Throws NoSuchElementException when empty |
|  | peekFirst() | Returns null when empty |

**LIFO Operations in Deque**

| **Operation** | **Method** | **Behavior When Full/Empty** |
| --- | --- | --- |
| **Push** | addFirst(element) | Throws IllegalStateException when full |
|  | offerFirst(element) | Returns false when full |
| **Pop** | removeFirst() | Throws NoSuchElementException when empty |
|  | pollFirst() | Returns null when empty |
| **Peek** | getFirst() | Throws NoSuchElementException when empty |
|  | peekFirst() | Returns null when empty |

**Key Differences Between Queue and Deque Naming**

* **Queue**: element() for peeking.
* **Deque**: getFirst() and getLast() for peeking.

**Deque Additional Methods**

* push(element): Adds element to the head.
* pop(): Removes and returns the element at the head.
* poll(): Removes the element at the tail.
* peek(): Views the element at the tail without removal.
* Returns null when no element is available.

**Implementing Queue and Deque in Java**

**Available Implementations in the Collections Framework**

| **Implementation** | **Interface** | **Backing Structure** | **Key Features** |
| --- | --- | --- | --- |
| **ArrayDeque** | Queue, Deque | Dynamic Array | Grows automatically, always accepts new elements |
| **LinkedList** | Queue, Deque | Doubly Linked List | Efficient access to first & last elements, always accepts new elements |
| **PriorityQueue** | Queue | Heap-backed Array | Maintains elements in sorted order (natural or custom Comparator), always keeps the smallest element at the head |

**Avoid Using the Stack Class**

* **Why?**
  + Stack extends Vector, which is thread-safe but outdated.
  + The Collections Framework provides better alternatives (Deque and ArrayDeque).
* **When to use an alternative?**
  + **For a non-thread-safe stack** → Use Deque (ArrayDeque).
  + **For a thread-safe stack** → Consider BlockingQueue implementations.