

CMSC 132:

OBJECT-ORIENTED PROGRAMMING II



PriorityQueue in JCF

Department of Computer Science

University of Maryland, College Park

What is a Priority Queue?

- A **Priority Queue** is a special type of queue where **elements are dequeued in order of priority**, not just FIFO (first-in-first-out).
- Each element has a **priority**, and the element with the **highest or lowest priority** is removed first.
- Internally, most implementations use a **heap** data structure.
- **Key Concepts:**
 - The priority queue arranges elements so that the one with the highest (or lowest) priority is always accessible at the front — regardless of when it was added.
 - Common operations: insert, peek, remove
 - Not designed for keeping the entire queue sorted at all times—it's optimized for **fast access to the highest (or lowest) priority element**, not for sorted iteration.

How Priority Queues Work

- Think of a **to-do list** where each task has urgency:
 - "Finish project" (high priority)
 - "Watch TV" (low priority)
- A priority queue ensures "Finish project" is handled first, even if "Watch TV" was added earlier.
- **Under the hood:**
 - Typically implemented as a **binary heap**.
 - **Min-heap**: smallest element dequeued first.
 - **Max-heap**: largest element dequeued first (via custom comparator).

PriorityQueue in Java

- `java.util.PriorityQueue<E>`
 - Part of the **Java Collections Framework**
 - Based on a **min-heap** by default (natural ordering of elements)
 - Requires elements to be Comparable **or** use a custom Comparator
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/PriorityQueue.html>

- **Constructors:**

```
PriorityQueue<> ();
```

```
PriorityQueue<> (Comparator<? super E> comparator);
```

```
PriorityQueue<> (Collection<? extends E> c);
```

- **Common methods:**

```
add(E e), offer(E e) – insert
```

```
peek() – view head
```

```
poll() – remove head
```

```
remove() – remove head
```

Behavior and Use Cases

- **Default Behavior:**

- Orders using compareTo() method (natural ordering)
- For objects, you must implement Comparable, or provide a Comparator

- **Use Cases:**

- Scheduling jobs by urgency
- Handling tasks in simulation engines
- Dijkstra's shortest path algorithm
- Event-driven systems (e.g., event queues)

Customizing Priority

Custom comparator examples:

- **Max-heap of integers:**

```
PriorityQueue<Integer> maxHeap = new PriorityQueue<>((a, b) -> b - a);
```

- **Custom object (e.g., Task with priority):**

Provide a `Comparator<Task>` to sort by priority field.

See Code Examples In this order:

PriorityQueueDemo, MaxHeapDemo, TaskManager,
ERQueue