Priority Queue with Named Nodes in Java

If you want to store named nodes in a priority queue in Java, you can do this by defining a custom class for the node (e.g., with a name and a priority field), and then using a PriorityQueue with a custom comparator or by implementing Comparable.

Step-by-step Example:

```
import java.util.PriorityQueue;

class Node implements Comparable<Node> {
    String name;
    int priority;

    public Node(String name, int priority) {
        this.name = name;
        this.priority = priority;
    }

    @Override
    public int compareTo(Node other) {
        return Integer.compare(this.priority, other.priority);
    }

    @Override
    public String toString() {
        return name + " (priority: " + priority + ")";
    }
}

public class NamedPriorityQueueExample {
    public static void main(String[] args) {
        PriorityQueue<Node> queue = new PriorityQueue<>();

        queue.offer(new Node("A", 3));
        queue.offer(new Node("B", 1));
        queue.offer(new Node("C", 2));

        while (!queue.isEmpty()) {
            Node node = queue.poll();
            System.out.println("Processing: " + node);
        }
    }
}
```

Expected Output:

```
Processing: B (priority: 1)
Processing: C (priority: 2)
Processing: A (priority: 3)
```

Explanation of Comparable<Node>:

Comparable<Node> is a Java interface that allows objects of the Node class to be compared to each other. This is used in sorting and ordering in collections like PriorityQueue or TreeSet.

The compareTo(Node other) method returns:
- A negative number if 'this' is less than 'other'
- Zero if they are equal
- A positive number if 'this' is greater than 'other'

This defines the natural ordering. In this example, lower priority values come out first (min-heap behavior).

To reverse the order (max-heap), simply reverse the comparison:
return Integer.compare(other.priority, this.priority);

Why Queue.poll() returns names and not priorities:

Queue.poll() returns the Node object with the highest priority based on compareTo. It does not return the name or priority directly.

When printing the Node object with:
System.out.println(node);

Java automatically calls the toString() method of that object behind the scenes.

So if your toString() is defined like this:
```
@Override
public String toString() {
    return name + " (priority: " + priority + ")";
}
```

You'll see the name and priority printed.

Why toString() is called without explicitly calling it:

When you pass an object to System.out.println(), Java implicitly calls the object's toString() method.

System.out.println(node);

is internally translated to:

System.out.println(node.toString());

If you don't override toString(), you get a default representation like:
Node@6bc7c054

Overriding toString() provides a human-readable format when printing objects.