🔹 **What is a Heap?**

A **Heap** is a special **tree-based data structure** that satisfies the **Heap Property**:

- **Max Heap** → Parent node is always **greater than or equal** to its children.
- **Min Heap** → Parent node is always **smaller than or equal** to its children.

A Heap is usually implemented as a **binary tree** (more specifically a **complete binary tree**).

---

🔹 **Heap in Java**

Java doesn't provide a direct **Heap class**, but we can use **PriorityQueue** (from java.util) which internally implements a **Min Heap**.

👉 If you want a **Max Heap**, you can use a custom **Comparator**.

---

🔹 **Example 1: Min Heap (Default)**

```
import java.util.PriorityQueue;


public class MinHeapExample {
    public static void main(String[] args) {
        // By default, PriorityQueue is a Min Heap
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();


        // Adding elements
        minHeap.add(50);
        minHeap.add(20);
        minHeap.add(30);
        minHeap.add(10);
        minHeap.add(40);


        System.out.println("Min Heap: " + minHeap);


        // Removing elements (smallest comes first)
        while (!minHeap.isEmpty()) {
            System.out.println("Removed: " + minHeap.poll());
```

```
        }
    }
}
```

**Output:**

```
Min Heap: [10, 20, 30, 50, 40]
Removed: 10
Removed: 20
Removed: 30
Removed: 40
Removed: 50
```

---

- **Example 2: Max Heap**

```
import java.util.*;

public class MaxHeapExample {
    public static void main(String[] args) {
        // Custom comparator to create a Max Heap
        PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

        maxHeap.add(50);
        maxHeap.add(20);
        maxHeap.add(30);
        maxHeap.add(10);
        maxHeap.add(40);

        System.out.println("Max Heap: " + maxHeap);

        while (!maxHeap.isEmpty()) {
            System.out.println("Removed: " + maxHeap.poll());
        }
    }
```

}

**Output:**

Max Heap: [50, 40, 30, 10, 20]

Removed: 50

Removed: 40

Removed: 30

Removed: 20

Removed: 10

---

- **Example 3: Heap with Custom Objects**

You can also use **PriorityQueue with custom objects** by implementing Comparator.

```
import java.util.*;

class Student {
    String name;
    int marks;

    Student(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }
}

public class CustomHeapExample {
    public static void main(String[] args) {
        // Max Heap based on student marks
        PriorityQueue<Student> maxHeap = new PriorityQueue<>((a, b) -> b.marks - a.marks);

        maxHeap.add(new Student("Alice", 85));
        maxHeap.add(new Student("Bob", 95));
        maxHeap.add(new Student("Charlie", 75));
```

```
        while (!maxHeap.isEmpty()) {
            Student s = maxHeap.poll();
            System.out.println(s.name + " -> " + s.marks);
        }
    }
}
```

**Output:**

Bob -> 95

Alice -> 85

Charlie -> 75

---

✅ **Summary**

- **Heap** = complete binary tree with heap property.
- Java provides PriorityQueue → implements **Min Heap** by default.
- Use Collections.reverseOrder() or custom Comparator for **Max Heap**.
- Useful in algorithms like **Dijkstra's shortest path, Huffman coding, Top-K problems, scheduling**.