Task 3: Implementing Heap Operations. Code a min-heap in JAVA with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation."

1. **CODE:**

```
public class MinHeap {
    private int[] heap;
    private int size;
    private int capacity;
       public MinHeap(int capacity) {
        this.capacity = capacity;
        this.size = 0;
        this.heap = new int[capacity];
    }
        private int parent(int index) {
        return (index - 1) / 2;
    }
    private int leftChild(int index) {
        return 2 * index + 1;
    }
        private int rightChild(int index) {
        return 2 * index + 2;
    }
        public void insert(int key) {
        if (size == capacity) {
            throw new IllegalStateException("Heap is full");
        }
                heap[size] = key;
        int current = size;
        size++;
```

```
            while (current != 0 && heap[parent(current)] > heap[current]) {
        swap(current, parent(current));
        current = parent(current);
    }    }
   public int extractMin() {
    if (size == 0) {
        throw new IllegalStateException("Heap is empty");
    }
    if (size == 1) {
        size--;
        return heap[0];
    }
            int root = heap[0];
    heap[0] = heap[size - 1];
    size--;
            heapify(0);
    return root;
}
public int getMin() {
    if (size == 0) {
        throw new IllegalStateException("Heap is empty");
    }
    return heap[0];
}
   private void heapify(int i) {
    int smallest = i;
    int left = leftChild(i);
    int right = rightChild(i);
    if (left < size && heap[left] < heap[smallest]) {
```

```
        smallest = left;
    }
    if (right < size && heap[right] < heap[smallest]) {
        smallest = right;
    }
    if (smallest != i) {
        swap(i, smallest);
        heapify(smallest);
    }   }
    private void swap(int i, int j) {
    int temp = heap[i];
    heap[i] = heap[j];
    heap[j] = temp;
}
   public static void main(String[] args) {
    MinHeap minHeap = new MinHeap(10);
    minHeap.insert(3);
    minHeap.insert(2);
    minHeap.insert(1);
    minHeap.insert(15);
    minHeap.insert(5);
    minHeap.insert(4);
    minHeap.insert(45);
    System.out.println("Minimum element: " + minHeap.getMin()); // Output: 1
    System.out.println("Extracted min: " + minHeap.extractMin()); // Output: 1
    System.out.println("Minimum element: " + minHeap.getMin()); // Output: 2
    System.out.println("Extracted min: " + minHeap.extractMin()); // Output: 2
    System.out.println("Minimum element: " + minHeap.getMin()); // Output: 3
}   }
```

Explanation:

❖ **MinHeap Class:**

1) **Attributes:**

- **heap:** Array to store heap elements.
- **size:** Current number of elements in the heap.
- **capacity:** Maximum capacity of the heap.
- **Constructor:** Initializes the heap with a given capacity.
- **parent(int index):** Returns the parent index of a given index.
- **leftChild(int index):** Returns the left child index of a given index.
- **rightChild(int index):** Returns the right child index of a given index.
- **insert(int key):** Inserts a new element into the heap and restores the heap property by bubbling up the new element.
- **extractMin():** Removes and returns the minimum element from the heap and restores the heap property by heapifying down the root element.
- **getMin():** Returns the minimum element without removing it.
- **heapify(int i):** Heapifies down a subtree rooted with node 'I' to restore the heap property.
- **swap(int i, int j):** Swaps two elements in the heap.
- **main(String[] args):** Tests the heap operations by inserting elements, extracting the minimum element, and printing the minimum element.