ID: 65011338
Lab 9 Name: Ko-Kwan Mongkholtham

Task 1: Create directory named pack8_Trees with package code inside. Study MyMinHeap's insert(int d) and remove() methods. Test L8_PQ_Main's demo1().

- **Project Structure**

```
└── tree
.
└── pack8_Trees
    ├── L8_PQ_Main.java
    └── code
        ├── MyMinHeap.java
        ├── MyPriorityQueue.java
        └── MyQueueInterface.java

3 directories, 4 files
```

- **MyMinHeap.java code**: continue at page 2

```java
package code;

public class MyMinHeap {
    int MAX_SIZE = 100;
    int heap[] = new int[MAX_SIZE];
    int size = 0;

    public void insert(int d) {
        if (isFull()) {
            throw new RuntimeException(message:"Heap is full");
        }

        heap[size] = d;
        int current = size;

        while (current != 0 && heap[current] < heap[parent(current)]) {
            swap(current, parent(current));
            current = parent(current);
        }

        size++;
    }

    public int remove() {
        if (isEmpty()) {
            throw new RuntimeException(message:"Heap is empty");
        }

        int popped = heap[0];
        heap[0] = heap[--size];

        heapify(pos:0);
        return popped;
    }
```

```java

    public int peek() {
        if (isEmpty()) {
            throw new RuntimeException(message:"Heap is empty");
        }
        return heap[0];
    }

    public boolean isFull() {
        return size == MAX_SIZE;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < size; i++) {
            sb.append(heap[i]).append(str:", ");
        }
        return sb.toString();
    }

    private int parent(int pos) {
        return (pos - 1) / 2;
    }

    private int leftChild(int pos) {
        return (2 * pos) + 1;
    }

    private int rightChild(int pos) {
        return (2 * pos) + 2;
    }

```

```java
    private void swap(int first, int second) {
        int tmp = heap[first];
        heap[first] = heap[second];
        heap[second] = tmp;
    }

    private void heapify(int pos) {
        if (!isLeaf(pos)) {
            if (heap[pos] > heap[leftChild(pos)] ||
                (rightChild(pos) < size && heap[pos] > heap[rightChild(pos)])) {

                if (rightChild(pos) < size && heap[leftChild(pos)] > heap[rightChild(pos)]) {
                    swap(pos, rightChild(pos));
                    heapify(rightChild(pos));
                } else {
                    swap(pos, leftChild(pos));
                    heapify(leftChild(pos));
                }
            }
        }
    }

    private boolean isLeaf(int pos) {
        return pos >= (size / 2) && pos < size;
    }
}
```

- Test L8_PQ_Main's demo1().

```
-demo1---
heap strucutre is 11, 13, 16, 15, 17, 18,
least 3 value is [11, 13, 15]
```

========== END TASK 1 ==========

Task 2: Given an abstract class MyQueueInterface.java, implement MyPriorityQueue.java from MyMinHeap's capabilities.

- MyPriorityQueue.java code

```java
package code;

public class MyPriorityQueue implements MyQueueInterface {
    private MyMinHeap heap;

    public MyPriorityQueue() {
        this.heap = new MyMinHeap();
    }

    public String toString() {
      return heap.toString();
    }

    @Override
    public void enqueue(int d) {
        if (isFull()) {
            throw new RuntimeException(message:"Priority Queue is full");
        }
        heap.insert(d);
    }

    @Override
    public int dequeue() {
        if (isEmpty()) {
            throw new RuntimeException(message:"Priority Queue is empty");
        }
        return heap.remove();
    }
```

```java

    @Override
    public int front() {
        return heap.peek();
    }

    @Override
    public boolean isFull() {
        return heap.isFull();
    }

    @Override
    public boolean isEmpty() {
        return heap.isEmpty();
    }
}
```

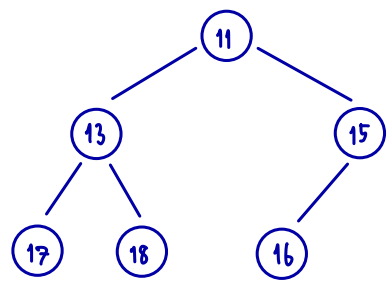
- Test L8_PQ_Main's demo2().

```
-demo2---
pq structure is 11, 13, 16, 15, 17, 18, 19,
least 3 value is [11, 13, 15]
```

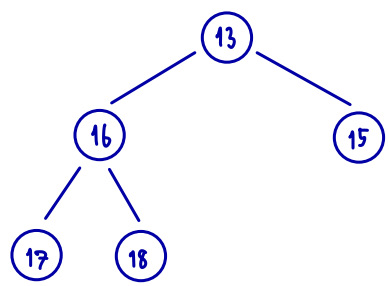
========== END TASK 2 ==========

Task 3: draw / write the heap snapshot during each dequeue() was performed.

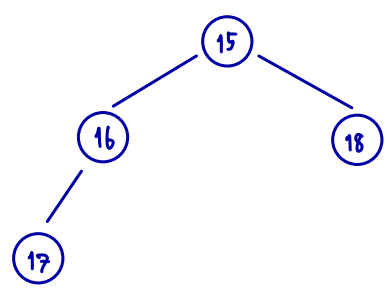
pq.enqueue(11); pq.enqueue(15);
pq.enqueue(16); pq.enqueue(13);
pq.enqueue(17); pq.enqueue(18);



After first dequeue() it will remove 11



After second dequeue() it will remove 13



lastly, third dequeue() will remove 15