Lab 6

**Q1**

package Queue;

public class ArrayQueue

{

private int maxSize;

private int[] queueArray;

private int front;

private int rear;

private int nItems;

// creating Queue

public ArrayQueue(int size)

{

maxSize = size;

queueArray = new int[maxSize];

front = 0;

rear = -1;

nItems = 0;

}

// inserting element

public void enqueue(int item)

{

if (rear == maxSize - 1)

{

System.out.println("Queue is full. Cannot enqueue.");

}

else

{

queueArray[++rear] = item;

nItems++;

}

}

// removing element from queue

public int dequeue()

{

if (isEmpty())

{

System.out.println("Queue is empty. Cannot dequeue.");

return -1;

}

else

{

int temp = queueArray[front];

front++;

nItems--;

return temp;

}

}

// checking the queue is empty

public boolean isEmpty()

{

return (nItems == 0);

}

// size of queue

public int size()

{

return nItems;

}

public static void main(String[] args)

{

ArrayQueue queue = new ArrayQueue(5);

// checking queue is empty

System.out.println("Queue is empty: " + queue.isEmpty());

// inserting element in queue

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

// removing element

System.out.println("Dequeued: " + queue.dequeue());

System.out.println("Dequeued: " + queue.dequeue());

// printing size

System.out.println("Queue size: " + queue.size());

// checking queue is empty

System.out.println("Queue is empty: " + queue.isEmpty());

}

}

**Output**

**Queue is empty: true**

**Dequeued: 10**

**Dequeued: 20**

**Queue size: 1**

**Queue is empty: false**

**Q2.**

package Queue;

import java.util.Arrays;

public class MinHeapPriorityQueue

{

private int capacity = 10;

private int size = 0;

private int[] items = new int[capacity];

// creating minheap

private int getLeftChildIndex(int parentIndex)

{

return 2 \* parentIndex + 1;

}

private int getRightChildIndex(int parentIndex)

{

return 2 \* parentIndex + 2;

}

private int getParentIndex(int childIndex)

{

return (childIndex - 1) / 2;

}

private boolean hasLeftChild(int index)

{

return getLeftChildIndex(index) < size;

}

private boolean hasRightChild(int index)

{

return getRightChildIndex(index) < size;

}

private boolean hasParent(int index)

{

return getParentIndex(index) >= 0;

}

private int leftChild(int index)

{

return items[getLeftChildIndex(index)];

}

private int rightChild(int index)

{

return items[getRightChildIndex(index)];

}

private int parent(int index)

{

return items[getParentIndex(index)];

}

private void swap(int indexOne, int indexTwo)

{

int temp = items[indexOne];

items[indexOne] = items[indexTwo];

items[indexTwo] = temp;

}

// checking capacity

private void ensureCapacity()

{

if (size == capacity)

{

items = Arrays.copyOf(items, capacity \* 2);

capacity \*= 2;

}

}

// inserting elements in minheap

public void insert(int item)

{

ensureCapacity();

items[size] = item;

size++;

heapifyUp();

}

// delete minimum element from minheap

public int deleteMin()

{

if (size == 0) throw new IllegalStateException();

int minItem = items[0];

items[0] = items[size - 1];

size--;

heapifyDown();

return minItem;

}

// checking the minheap is empty

public boolean isEmpty()

{

return size == 0;

}

// size of min heap

public int size()

{

return size;

}

private void heapifyUp()

{

int index = size - 1;

while (hasParent(index) && parent(index) > items[index])

{

swap(getParentIndex(index), index);

index = getParentIndex(index);

}

}

private void heapifyDown()

{

int index = 0;

while (hasLeftChild(index))

{

int smallerChildIndex = getLeftChildIndex(index);

if (hasRightChild(index) && rightChild(index) < leftChild(index))

{

smallerChildIndex = getRightChildIndex(index);

}

if (items[index] < items[smallerChildIndex])

{

break;

} else {

swap(index, smallerChildIndex);

}

index = smallerChildIndex;

}

}

public static void main(String[] args)

{

// creating the priority min heap

MinHeapPriorityQueue pq = new MinHeapPriorityQueue();

// inserting element in minheap

pq.insert(10);

pq.insert(5);

pq.insert(20);

pq.insert(1);

pq.insert(22);

pq.insert(3);

//size of the minheap

System.out.println("Size of the priority queue: " + pq.size());

// delete the minimum element among minheap

System.out.println("Deleted min item: " + pq.deleteMin());

// printing size of minheap

System.out.println("Size after deletion: " + pq.size());

// checking the minheap is empty

System.out.println("priority Queue is empty: " + pq.isEmpty());

}

}

**Output**

**Size of the priority queue: 6**

**Deleted min item: 1**

**Size after deletion: 5**

**priority Queue is empty: false**