```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package hw06;

import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.NoSuchElementException;

/**
 *
 * @author said
 * @param <E>
 */
public class GITPriorityQueue<E> extends BinaryTree<E> implements
PriorityQueue<E>{
    Comparator<E> comparator;
    Node<E> lastLocalRoot;
    LinkedList<Node<E>> lastLocalRootList;
    Iterator<Node<E>> iter;
    private boolean insertReturn;

    public GITPriorityQueue() {
        super();
        lastLocalRootList = new LinkedList<>();
        lastLocalRoot = null;
    }

    /**
     * Creates a heap-based priority queue with the specified initial capacity
     * that orders its elements according to the specified comparator.
     *
     * @param comp The comparator used to order this priority queue
     * @throws IllegalArgumentException if cap is less than 1
     */
    public GITPriorityQueue(Comparator<E> comp) {
        super();
        comparator = comp;
        lastLocalRootList = new LinkedList<>();
        lastLocalRoot = null;
    }

    private int compare(E left, E right){

        if(comparator != null)
            return comparator.compare(left, right);
        else
            return ((Comparable<E>) left).compareTo(right);
    }

    private void swap(Node<E> i, Node<E> j){
        E temp;

        temp = i.data;
        i.data = j.data;
        j.data = temp;
    }
```

```
    @Override
    public boolean offer(E item) {
        if(root == null){
            root = new Node<>(item);
            lastLocalRootList.add(root);
            this.iter = lastLocalRootList.listIterator();
            lastLocalRoot = iter.next();
            return true;
        }else{
            if( (!iter.hasNext()) && isFull(lastLocalRoot)){
                initializeLocalRoots();
            }
            Node<E> newNode = new Node<>(item);

            if(isFull(lastLocalRoot)){
                lastLocalRoot = iter.next();
                lastLocalRoot.left = newNode;
                lastLocalRoot.left.parent = lastLocalRoot;
            }else{
                if(lastLocalRoot.left == null){
                    lastLocalRoot.left = newNode;
                    lastLocalRoot.left.parent = lastLocalRoot;
                }else{
                    lastLocalRoot.right = newNode;
                    lastLocalRoot.right.parent = lastLocalRoot;
                }

            }
            insertReturn = true;
            orderToUp(newNode);
        }

        if(!insertReturn){
            throw new IllegalStateException(item.toString() + " is already
existed");
        }
        return insertReturn;
    }

    private void orderToUp(Node<E> node) {
        if(node.parent != null){

            int cmp = compare(node.data, node.parent.data);

            if( cmp == 0){
                insertReturn = false;
            }else if( cmp < 0){
                swap(node, node.parent);
            }
            orderToUp(node.parent);
        }
    }

    private void initializeLocalRoots() {
        Iterator<Node<E>> localiter = lastLocalRootList.listIterator();
        LinkedList<Node<E>> newLocalRootList;
        newLocalRootList = new LinkedList<>();
        Node<E> node;

        while(localiter.hasNext()){
            node = localiter.next();
            newLocalRootList.add(node.left);
```

```
            newLocalRootList.add(node.right);
        }

        lastLocalRootList = newLocalRootList;
        this.iter = lastLocalRootList.listIterator();
        lastLocalRoot = iter.next();
    }

    @Override
    public E poll() {
        E value = root.data;

        if(root == null)
            return value;

        if(lastLocalRoot.right == null){
            root.data = lastLocalRoot.left.data;
            lastLocalRoot.left = null;
        }else{
            root.data = lastLocalRoot.right.data;
            lastLocalRoot.right = null;
        }
        orderToDown(root);
        return value;
    }

    private void orderToDown(Node<E> localRoot) {
        if( localRoot.left != null && localRoot.right != null ){
            Node<E> smallest;

            int cmp = compare(localRoot.left.data, localRoot.right.data);
            if(cmp < 0){
                smallest = localRoot.left;
            }else{
                smallest = localRoot.right;
            }

            cmp = compare(smallest.data, root.data);
            if(cmp < 0){
                swap(smallest, root);
                orderToDown(smallest);
            }
        }
    }

    @Override
    public E remove() {
        E removed = poll();

        if(removed == null){
            throw new NoSuchElementException();
        }

        return removed;
    }

    @Override
    public E peek() {
        return root.data;
    }

    @Override
```

```
    public E element() {
        E res = peek();
        if(res == null)
            throw new NoSuchElementException();
        else
            return res;
    }

    public boolean isFull(Node<E> localRoot){
        return (localRoot.left != null && localRoot.right != null);
    }




}
```