SWE4	Übung zu Softwareentwicklung mit modernen Plattformen 4	SS 2022, Übung 3

☐ Gruppe 1 (J. Heinzelreiter)

☒ Gruppe 2 (P. Kulczycki) Name: Angelos Angelis Aufwand [h]: 11

☐ Gruppe 3 (M. Hava) Peer-Review von: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (70 + 30 P)	100	100	100

Prioritätswarteschlangen (`src/priority-queue`)

In der ersten Übung zum Themengebiet Java haben wir uns mit der Heap-Datenstruktur beschäftigt, welche eine effiziente Realisierung von Prioritätswarteschlangen ermöglicht. In dieser Übungsaufgabe sollen Sie auf Basis dieser Datenstruktur eine Prioritätswarteschlange mit Java implementieren.

a) Implementieren Sie die Klasse `PriorityQueue` (im Paket `swe4.collections`), die vergleichbare Java-Objekte vom Typ T verwalten kann. `PriorityQueue` weist folgende Schnittstelle auf:

```
public class PriorityQueue<T> implements Iterable<T> {
  public PriorityQueue();
      // Erzeugt eine leere PriorityQueue.
  public PriorityQueue(Iterable<T> collection);
      // Erzeugt eine PriorityQueue, in der alle Elemente von collection enthalten sind.
  public void add(T element);
      // Fügt ein neues Element zur PriorityQueue hinzu. Elemente können auch mehrfach
      // in der PriorityQueue enthalten sein.
  public void add(Iterable<T> collection);
      // Fügt alle Elemente von collection zur PriorityQueue hinzu.
  public int removeAll(T element);
      // Entfernt alle Vorkommen von element. Gibt zurück, wie viele Elemente gelöscht
      // worden sind.
  public boolean removeMax();
      // Entfernt ein Vorkommen des größten Elements. Gibt zurück, ob ein Element gelöscht wurde.
  public T max();
      // Gibt das größte Element von PriorityQueue zurück. Ist die Queue leer, wird eine
      // NoSuchElementException geworfen.
  public int size();
      // Gibt die Anzahl der Elemente in PriorityQueue zurück.
  public Iterator<T> iterator();
      // Gibt einen Iterator zurück, mit dem durch alle zum Zeitpunkt des Aufrufs diese Methode
      // gespeicherten Elemente iteriert werden kann. Der Iterator liefert die Elemente in
      // absteigender Reihenfolge.
  public String toString();
      // Liefert alle Elemente von PriorityQueue als Zeichenkette. Die Reihenfolge entspricht der
      // internen Repräsentierung der Daten.
  private void heapify();
      // Interne Methode, welche das Feld mit den Datenelementen in einen Heap umwandelt.
```

```
}
```

Funktionale Anforderungen:

- Implementieren Sie alle Operationen auf effiziente Art und Weise, indem Sie die Heap-Datenstruktur verwenden.
- Wenn das interne Feld zur Speicherung weiterer Elemente nicht mehr ausreicht, ist die Feldgröße zu verdoppeln.
- Achten Sie auch auf eine effiziente Implementierung des Konstruktors und der Methode `add`, welche einen Behälter von Elementen übergeben bekommen. Gehen Sie so vor, dass Sie zunächst alle Elemente zu ihrem internen Feld hinzufügen und anschließend durch Aufruf der Methode `heapify` die Heap-Eigenschaft des internen Feldes wieder herstellen.

  Hinweis: Implementieren Sie `heapify`, indem Sie für alle Elemente, die zumindest einen Nachfolger haben, eine downHeap-Operation durchführen. Beginnen Sie mit dem am weitesten rechts stehenden Element, das noch einen Nachfolger hat, und durchlaufen Sie von dort weg alle Elemente des Heaps bis zur Wurzel.
- Der Iterator liefert die Elemente in absteigender Reihenfolge. Veränderungen an der `PriorityQueue` dürfen auf einen bestehenden Iterator keinen Einfluss haben. Erstellen Sie dazu beim Erzeugen des Iterators einen Schnappschuss der zu diesem Zeitpunkt in der Warteschlange enthalten Elemente, den Sie im Iterator speichern.
- Sie dürfen bei dieser Übung (noch) nicht das JDK-Behälter-Framework verwenden.
- Sie können voraussetzen, dass die Elemente von `PriorityQueue<T>` vergleichbar sind. Zwei Elemente `t1` und `t2` vom Datentyp T können so miteinander verglichen werden:

  ```
  int r = (((Comparable<T>)t1).compareTo(t2));
  // r == 0 => t1 == t2
  // r < 0 => t1 < t2
  // r > 0 => t1 > t2
  ```

  Datentypen wie `Integer`, `String`, `LocalDate` etc. erfüllen diese Eigenschaft.

b) Erstellen Sie zum Test der Klasse `PriorityQueue` eine möglichst umfangreiche Testsuite, die folgenden Anforderungen genügt:

- Die Testfälle sind voneinander unabhängig.
- Jeder Testfall testet nach Möglichkeit nur einen oder wenige Aspekte von `PriorityQueue`.
- Die Testfälle decken den Code von `PriorityQueue` vollständig ab (jede Quelltextzeile wird von mindestens einem Test zur Ausführung gebracht). Es ist anzustreben, dass der Quelltext mehrfach abgedeckt wird.
- Erstellen Sie ein möglichst engmaschiges Netz an Assertionen. Ein Qualitätsmerkmal eines Unittests ist auch die Anzahl der darin enthaltenen Assertionen.
- Wenden Sie im Zuge des Feedbacks Ihre Testsuite auf die Implementierung Ihrer Kollegin bzw. Ihres Kollegen an und dokumentieren Sie die Ergebnisse.

Geben Sie dem Testen in dieser Übung einen besonderen Stellenwert. Im Zuge des Feedbacks müssen Sie Ihre Testsuite auch auf die Implementierung Ihres Feedback-Partners anwenden. Achten Sie daher darauf, dass Ihr Programm mit dem in der Übung zur Verfügung gestellten Ant-Script übersetzbar ist und die Unit-Tests damit ausgeführt werden können.

Lösungsidee:

Meine PriorityQueue besteht aus einem Object array und einer int variable size. Ich habe Object als datentyp ausgewählt weil dieser es ermöglich ein generic Array zu implementieren. Zu beachten is folgendes:

public void add(Iterable<T> collection): Es wird an dem Array alle Elemente hinzugefügt und dann geheapified weil das effizienter ist.

private void heapify(): Der einzige Fall wo diese Methode benötigt wird ist nach dem public void add(Iterable<T> collection) ansonsten wird im array mittels upheap die elemente an der richtigen Stelle hinzugefügt. Abgesehen davon wurde heapify ähnlich wie in der Angabe beschrieben implementiert nur dass dann die restliche Elemente bis zur Wurzel rekursiv durchlaufen werden

public Iterator<T> iterator(): Zu beacten hier ist dass der Iterator ein seperates array besitzt. Dieses ist die sortierte Version des Heaps in der Priority Queue.

Quellcode:

```
package swe4.collections;

import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;

public final class PriorityQueue<T> implements Iterable <T> {
    public static Object[] heap = null;
    private static int size = 0;
    private static int left(int i) {  //static = Klassenmethode
        return 2*i+1;
    }

    private static int right(int i) {
        return 2*i+2;
    }

    private static int parent(int i) {
        return (i-1) / 2;
    }
    public PriorityQueue(){
        heap = new Object[11];
    }

    public PriorityQueue(Iterable<T> collection){
        heap = new Object[11];
        for (T element : collection){
            add(element);
        }
    }

    private void upHeap(int k) {
        if(size <= 1) return;
```

```
        Object e = heap[k];
        int comp = (((Comparable<T>) heap[parent(k)]).compareTo((T) e));
        while(k > 0 && comp < 0) {
            heap[k] = heap[parent(k)];
            k = parent(k);
        }

        heap[k] = e;
    }

    public void add(T element){
        size++;
        if (size == heap.length) increaseCapacity();
        heap[size-1] = element;
        upHeap(size-1);
    }

    public void add (Iterable<T> collection){
        for (T element : collection){
            size++;
            if (size == heap.length) increaseCapacity();
            heap[size-1] = element;
        }
        buildHeap();
    }

    public int removeAll(T element){
        int count = 0;
        for (int i =0; i < size; i++){
            if (heap[i] == element){
                System.arraycopy(heap, i + 1, heap, i, heap.length - i - 1);
                count++;
                i--;
                size--;
            }
        }
        return count;
    }

    private void downHeap(int k) {
        Object e = heap[k];

        while(k <= parent(size-1)) {
            int j = left(k);
            int comp = (((Comparable<T>) heap[j]).compareTo((T) heap[j+1]));
            if (j < size-1 && comp < 0) j++;
            int comp2 = (((Comparable<T>) e).compareTo((T) heap[j]));
            if (comp2 >= 0) //Element muss nicht mehr nach unten
                break;

            heap[k] = heap[j];  //siehe Zeichnung
            k = j;
        }

        heap[k] = e;
    }
```

```
    public boolean removeMax(){
        if (size == 0) return false;
        heap[0] = heap[size-1]; //schiebe letztes Element nach ganz oben
        size--;
        downHeap(0);
        return true;
    }

    public T max(){
        if (size == 0) throw new ArrayIndexOutOfBoundsException("Empty
Array");
        return (T)heap[0];
    }

    public int size(){
        return size;
    }

    private void heapify(Object arr[], int n, int i)
    {
        int largest = i;
        int l = (2 * i) + 1;
        int r = (2 * i) + 2;

        if (l < n) {
            int comp = (((Comparable<T>) heap[l]).compareTo((T)
arr[largest]));
            if (comp > 0) {
                largest = l;
            }
        }

        if (r < n){
            int comp2 = (((Comparable<T>)heap[r]).compareTo((T)
arr[largest]));
            if (comp2 > 0) {
                largest = r;
            }
        }

        if (largest != i) {
            Object swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }

    public void buildHeap()
    {

        int startIdx = (size / 2) - 1;

        for (int i = startIdx; i >= 0; i--) {
            heapify(heap, size, i);
        }
    }
```

```
private void increaseCapacity(){
    System.out.println("Length of initial array = " + heap.length);
    int len = heap.length;
    Object newHeap[] = new Object[len*2];
    System.arraycopy(heap, 0, newHeap, 0, len);
    heap = newHeap;
    System.out.println("Length of new array = "+heap.length);
}

private final class ArrayIterator <T> implements Iterator <T>
{
    private Object array[];
    private int    pos = 0;

    public ArrayIterator()
    {
        array = heap;
        Arrays.sort(array,0,size(),Collections.reverseOrder());
    }

    private void updateArray(){
        array = heap;
        Arrays.sort(array,Collections.reverseOrder());
    }

    public boolean hasNext()
    {
        return pos < size;
    }

    public T next() throws InvalidIteratorException
    {
        if ( hasNext() )
            return (T) array[pos++];
        else
            throw new InvalidIteratorException("No next");
    }
}

@Override
public Iterator<T> iterator() {
        return new ArrayIterator();
}

@Override //Annotation
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("[");

    for (int i=0; i<size; i++) {
        if (i > 0) sb.append(", ");
        sb.append(heap[i]);
    }
    sb.append("]");
    return sb.toString();
}
```

```
}
```

Tests:

```
package swe4.collections;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

public class PriorityQueueTestMain {
    //Constructor Tests
    @Test
    @DisplayName("Create New and empty PQ")
    void newEmptyPQ(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        System.out.println("size: " + pq.size());
    }

    @Test
    @DisplayName("Create New PQ with a String list of size 11")
    void newPQFromIntList(){
        List<Integer> list = new ArrayList<Integer>();
        list.add(1);
        list.add(3);
        list.add(5);
        list.add(4);
        list.add(6);
        list.add(13);
        list.add(10);
        list.add(9);
        list.add(8);
        list.add(15);
        list.add(17);
        PriorityQueue<Integer> pq = new PriorityQueue<>(list);
        assertEquals(11,pq.size());
    }

    @Test
    @DisplayName("Create New PQ with a String list of size 11")
    void newPQFromStringList(){
        List<String> list = new ArrayList<String>();
        list.add("1");
        list.add("3");
        list.add("5");
        list.add("4");
        list.add("6");
        list.add("13");
        list.add("10");
```

```
        list.add("9");
        list.add("8");
        list.add("15");
        list.add("17");
        PriorityQueue<String> pq = new PriorityQueue<>(list);
        assertEquals(11,pq.size());
    }

    //Iterator Tests
    @Test
    @DisplayName("Create Iterator on empty PQ")
    void newItOnEmptyPQ(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        Iterator i = pq.iterator();
    }

    @Test
    @DisplayName("Create Iterator on filled PQ")
    void newItOnFilledPQ(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(10);
        pq.add(22);
        pq.add(45);
        Iterator i = pq.iterator();
    }

    @Test
    @DisplayName("Create Iterator, iterate and return every Item in PQ until
the last one (sorted)")
    void newItOnFilledPQSortedOutput(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(10);
        pq.add(22);
        pq.add(45);
        Iterator i = pq.iterator();
        assertEquals(45,i.next());
        assertEquals(22,i.next());
        assertEquals(10,i.next());
    }

    @Test
    @DisplayName("Create Iterator on empty PQ iterate through it, Exception
expected")
    void newItOnEmptyPQIterate(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        Iterator i = pq.iterator();
        assertThrows(InvalidIteratorException.class,() -> { i.next(); });
    }

    @Test
    @DisplayName("Create Iterator, Check if iterates through duplicate
values")
    void newItOnFilledPQIterateDuplicates(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(10);
        pq.add(10);
        pq.add(10);
```

```
        Iterator i = pq.iterator();
        assertEquals(10,i.next());
        assertEquals(10,i.next());
        assertEquals(10,i.next());
    }

    @Test
    @DisplayName("Create Iterator, Has no next on empty PQ")
    void newItOnEmptyPQHasNoNext(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        Iterator i = pq.iterator();
        assertEquals(false,i.hasNext());
    }

    //add() Tests
    @Test
    @DisplayName("Create PQ, add Interger, Test toString")
    void addIntPQ(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(10);
        assertEquals("[10]",pq.toString());
    }

    @Test
    @DisplayName("Create PQ, add String, Test toString and heap properties")
    void addStringPQ(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Erster String");
        pq.add("Zweiter String");
        assertEquals("Zweiter String",pq.max());
    }

    @Test
    @DisplayName("Create PQ, add String List, Test toString and heap
properties")
    void addStringListPQ(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Erster String");
        pq.add("Zweiter String");
        List<String> list = new ArrayList<String>();
        list.add("1");
        list.add("3");
        list.add("5");
        list.add("4");
        pq.add(list);
        assertEquals("Zweiter String",pq.max());
    }

    //removeAll() Tests
    @Test
    @DisplayName("Create PQ, Remove all occurances of String")
    void RemoveStringPQ(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Erster String");
        pq.add("Zweiter String");
        pq.add("Zweiter String");
        pq.add("Zweiter String");
```

```
        pq.add("Zweiter String");
        List<String> list = new ArrayList<String>();
        list.add("4");
        list.add("3");
        list.add("4");
        list.add("4");
        pq.add(list);
        pq.removeAll("Zweiter String");
        assertEquals("Erster String",pq.max());
    }

    @Test
    @DisplayName("Create PQ, Remove all occurances of Int also check size")
    void RemoveIntPQCheckSize(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(1);
        pq.add(3);
        pq.add(3);
        pq.add(3);
        pq.add(2);
        pq.add(3);
        pq.removeAll(3);
        assertEquals(2,pq.size());
        assertEquals(2,pq.max());
    }

    //Heapify() Tests
    @Test
    @DisplayName("Create PQ, Heapify Int PQ")
    void HeapifyIntPQ(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(1);
        pq.add(2);
        pq.add(2);
        pq.add(2);
        pq.add(3);
        pq.add(2);
        pq.buildHeap();
        assertEquals(3,pq.max());
    }

    @Test
    @DisplayName("Create PQ, Heapify String PQ")
    void HeapifyStringPQ(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Erster String");
        pq.add("Zweiter String");
        List<String> list = new ArrayList<String>();
        list.add("4");
        list.add("3");
        list.add("4");
        list.add("4");
        pq.add(list);
        assertEquals("Zweiter String",pq.max());
    }

    //removeMax() Tests
```

```
    @Test
    @DisplayName("Create PQ, remove max Int from PQ")
    void removeMaxIntPQ(){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(1);
        pq.add(2);
        pq.add(2);
        pq.add(2);
        pq.add(3);
        pq.add(2);
        int prevSize = pq.size();
        pq.removeMax();
        assertEquals(2,pq.max());
        assertEquals(prevSize-1,pq.size());
    }

    @Test
    @DisplayName("Create PQ, remove max String from PQ")
    void removeMaxStringPQ(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Erster String");
        pq.add("Zweiter String");
        List<String> list = new ArrayList<String>();
        list.add("4");
        list.add("3");
        list.add("4");
        list.add("4");
        pq.add(list);
        int prevSize = pq.size();
        pq.removeMax();
        assertEquals("Erster String",pq.max());
        assertEquals(prevSize-1,pq.size());
    }

    @Test
    @DisplayName("Create PQ, remove max String 2x from PQ")
    void removeMaxString2xPQ(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        pq.add("Erster String");
        pq.add("Zweiter String");
        List<String> list = new ArrayList<String>();
        list.add("4");
        list.add("3");
        list.add("4");
        list.add("4");
        pq.add(list);
        int prevSize = pq.size();
        pq.removeMax();
        pq.removeMax();
        assertEquals("[4, 4, 4, 3]",pq.toString());
        assertEquals(prevSize-2,pq.size());
    }

    //Other Tests

    //size when empty
    @Test
```

```
    @DisplayName("Create empty PQ, get size 0 Expected")
    void emptyPQSize(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        assertEquals(0,pq.size());
    }
    //size before and after deleting
    @Test
    @DisplayName("Create PQ, get size add get size")
    void PQSize(){
        PriorityQueue<String> pq = new PriorityQueue<>();
        assertEquals(0,pq.size());
        pq.add("12");
        pq.add("asdae");
        assertEquals(2,pq.size());
    }
    //ToString LocalDate
    @Test
    @DisplayName("Create PQ, LocalDate ToString")
    void PQLocalDate(){
        PriorityQueue<LocalDate> pq = new PriorityQueue<>();
        assertEquals(0,pq.size());
        pq.add(LocalDate.parse("2007-12-03"));
        pq.add(LocalDate.parse("2008-12-03"));
        assertEquals(2,pq.size());
        assertEquals("[2008-12-03, 2007-12-03]",pq.toString());
    }

    //max when empty
    @Test
    @DisplayName("Create empty PQ, try max error expected")
    void PQMaxLocalDate(){
        PriorityQueue<LocalDate> pq = new PriorityQueue<>();
        assertThrows(ArrayIndexOutOfBoundsException.class,() -> { pq.max();
});
    }

}
```