

Übung 05: Generics, Innere Klassen, JUnit-Tests

Name: _____

Matrikelnummer: _____

 Informatik: G1 (Prähofer) G2 (Prähofer) G3 (Würthinger) G4 (Prähofer)

 WIN: G1 (Khalil) G2 (Khalil) G3 (Schwinger)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 4	24	Java-für PriorityQueue JUnit	Java-für PriorityQueue JUnit	<input type="checkbox"/>	

Übung 05: PriorityQueue (24 Punkte)

Prioritätswarteschlangen (`PriorityQueues`) dienen zur gereihten Verwaltung von Elementen nach einer Priorität. Das heisst, eine Prioritätswarteschlange soll Elemente gleichen Typs unter Berücksichtigung einer Priorität aufnehmen können. Wichtig ist, dass dabei Elemente gleicher Priorität vorkommen können.

Folgend (siehe auch Download von der Homepage) ist ein generisches Interface für Prioritätswarteschlangen gegeben. Der Typparameter `P` definiert dabei den Typ für die Priorität, der Typparameter `E` den Typ für die Elemente selbst. Beachten Sie, dass Typ `P` das Interface `Comparable<P>` erweitert und somit die Prioritätswerte über `compareTo` verglichen werden sollen. Dieser Vergleich definiert die Reihenfolge der Elemente.

Das Interface definiert Methoden für das Hinzufügen von Elementen (`insert`), den Zugriff auf das erste Element (= Element mit geringstem Prioritätswert) (`getFirstElement`) und der Priorität (`getFirstPriority`), das Löschen des ersten Elements (`deleteFirst`) und für den Test, ob die Prioritätswarteschlange leer ist (`isEmpty`). Die Methoden `getFirstPriority`, `getFirstElement` und `deleteFirst` werfen dabei `PriorityQueueExceptions`, die hier anzeigen soll, dass kein erstes Element vorhanden ist. Des Weiteren ist eine Methode `getIterator` definiert, die einen Iterator liefert, mit dem man über alle Elemente iterieren kann.

```

public interface PriorityQueue<P extends Comparable<P>, E> {
    public void insert(P priority, E element);
    public P getFirstPriority() throws PriorityQueueException;
    public E getFirstElement() throws PriorityQueueException;
    public void deleteFirst() throws PriorityQueueException;
    public boolean isEmpty();
    public PriorityQueueIterator<P, E> iterator();
}

public class PriorityQueueException extends Exception {
    private PriorityQueue priorityQueue;
    public PriorityQueueException(String message, PriorityQueue priorityQueue){
        super(message);
        this.priorityQueue = priorityQueue;
    }
    public PriorityQueue getPriorityQueue() {
        return priorityQueue;
    }
}

```

Des Interface `PriorityQueueIterator` definiert die Methoden wie folgt: Mit `hasNext` kann man überprüfen, ob weitere Elemente vorhanden sind, mit `next` springt man zum nächsten Element, mit `getPriority` und `getElement` kann auf die aktuellen Werte zugreifen. Die letzten Methoden werfen wiederum `PriorityQueueExceptions`, die anzeigen, dass kein aktuelles oder kein nächstes Element vorhanden ist.

```
public interface PriorityQueueIterator<P extends Comparable<P>, E> {
    public boolean hasNext();
    public void next() throws PriorityQueueException;
    public P getPriority() throws PriorityQueueException;
    public E getElement() throws PriorityQueueException;
}
```

Aufgabe 1: Implementierung von PriorityQueue

12 Punkte

Sie sollen nun `PriorityQueue` implementieren, wobei Sie intern eine Klasse des `Collection`-Package verwenden sollen. Dies beinhaltet auch eine Implementierung eines `PriorityQueueIterator`, den Sie als innere Klasse ausführen sollen.

Hinweis: `LinkedList` eignet sich gut, man muss nur eine Klasse für die Paare von Priorität und Elementwert implementieren. Am besten macht man das als statische innere Klasse.

Aufgabe 2: Testen Ihrer Implementierung mit dem JUnit-Framework

12 Punkte

Realisieren Sie dann einen umfangreichen JUnit-Test. Die Tests sollen die Methoden einzeln und umfassend testen und auch testen, dass die Exceptions richtig geworfen werden. Sie können dabei mit beliebigen Typen für Priorität und Element arbeiten. Achten Sie aber darauf, dass der Typ für die Priorität `Comparable` implementiert.

Hinweis: Mit Typ `Integer` für Priorität und `String` für die Elemente wird der Test einfach.