Institut für Informatik
Prof. Dr.-Ing. Elke Pulvermüller
Dipl.-Systemwiss. Mathias Menninghaus

Universität Osnabrück, 16.05.2017
http://www-lehre.inf.uos.de/~binf
Testat bis 24.05.2017, 14:00 Uhr

# Übungen zu Informatik B

*Sommersemester 2017*

**Blatt 5**

### Aufgabe 5.1: Sequenzdiagramme (25 Punkte)

Betrachten Sie das Programm `UseLibrary`, das die Implementation des Klassendiagrammes vom vorherigen Aufgabenblatt benutzen soll. Geben Sie den Programmablauf in einem Sequenzdiagramm wieder. Beschränken Sie sich dabei auf die Darstellung der aufrufenden Klasse sowie der Instanzen `lib`, `li`, `searchResult` und der `List inventory` aus `lib`. Zum Zeichnen können Sie ein beliebiges Programm, wie beispielsweise das auf den CIP-Pool-Rechnern installierte DIA (`http://www.dia-installer.de`) oder Visual Paradigm (`https://ap.visual-paradigm.com/university-of-osnabrueck`) verwenden. Geben Sie Ihre Lösung auch schriftlich ab.

### Aufgabe 5.2: `Cloneable` und `clone` (30 Punkte)

Betrachten Sie die Klassen `List` und `Entry`, mit denen der ADT Liste implementiert wurde.

Erzeugen Sie auf Basis dieser Klassen eine typsichere Liste. Begründen Sie Ihrem Tutor, ob es sinnvoller ist, von der bestehenden Klasse zu erben und nur die nötigsten Methoden zu überschreiben, oder die Liste vollständig neu zu implementieren.

Lesen Sie außerdem die Java - Dokumentation zu dem Interface `Cloneable` und der Methode `Object.clone()`. Implementieren Sie das Interface `Cloneable` in Ihrer generischen Liste.

Testen Sie anschließend automatisiert in einer eigenen Testklasse, ob Sie die Methode `clone()` korrekt implementiert haben. Achten Sie darauf, auch auf die nicht absoluten Anforderungen an eine `clone()` - Implementation zu testen. Begründen Sie, wenn Ihre Implementation bewusst einige dieser Anforderungen nicht erfüllt.

### Aufgabe 5.3: Heap (45 Punkte)

Ein Heap ist ein binärer Baum, der bis zur vorletzten Ebene voll besetzt ist. Auf der Blatt-Ebene ist er von links beginnend bis zum letzten Element vollständig besetzt. Jeder Knoten eines Heaps enthält einen Schlüssel. Der Schlüssel eines Knotens ist kleiner oder gleich der Schlüssel seiner Kind-Knoten.

Implementieren Sie einen typsicheren Heap. Die Ordnung der eingefügten Schlüssel-Elemente soll entweder dadurch gegeben sein, dass alle Schlüssel das Interface `java.lang.Comparable` implementieren oder dass beim Erzeugen eines Heaps ein passender `java.util.Comparator` mitgegeben wird. Mit Ihrer Implementation sollen beide Varianten abgedeckt werden. Wird der Heap also

mit einem `Comparator` erzeugt, wird die Ordnung anhand dieses `Comparator` definiert. Wurde kein `Comparator` mit angegeben, soll die Heap-Implementation davon ausgehen, das alle eingefügten Instanzen vom Typ `Comparable` sind. Mit den Methoden eines Heaps soll man Objekte in einen Heap einfügen, das kleinste Objekt löschen oder das kleinste Objekt ermitteln können. Nachdem eine von außen sichtbare Methode abgearbeitet wurde, darf der Heap in keinem ungültigen Zustand sein. Achten Sie darauf, das Ihre Implementation auch tatsächlich die Laufzeiteigenschaften eines Heap hat, es genügt nicht, eine sortierte Liste zu implementieren.

Schreiben Sie anschließend eine Testklasse, die die beschriebenen Funktionalitäten automatisch testet. Sie können dafür auch die Klasse `HeapSort` benutzen. Testen Sie auch, ob zu den entsprechenden Parametereingaben eventuell angekündigte Exceptions geworfen werden.

**Hinweise**

- Benutzen Sie passende Exceptions, wenn es zu Fehlern kommt.
- Es empfiehlt sich, den Heap mit Hilfe eines Arrays zu implementieren.
- Für die Manipulation von Arrays können Sie die Klasse `java.util.Arrays` nutzen.
- Es ist zulässig, wenn der Compiler Warnmeldungen liefert, solange Ihnen die darin beschriebenen Fehlerquellen bewusst sind und Sie diese entweder eliminieren oder mindestens dokumentieren.