Institut für Informatik
Prof. Dr.-Ing. Elke Pulvermüller
Lars Huning, M.Sc.

Universität Osnabrück, 17.05.2021
Testat bis 02.06.2021, 18:00 Uhr

Übungen zu Einführung in die Software-Entwicklung

Sommersemester 2021

Blatt 6

Achtung! Dieses Aufgabenblatt erstreckt sich über 2 Wochen. Die Testate zu diesem Blatt finden vom 31.05.-02.06. statt. In der Woche vom 24.5.-26.5. finden keine Testate statt.

Aufgabe 6.1: `Cloneable` und `clone` - Implementierung (20 Punkte)

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

Erzeugen Sie auf Basis dieser Klassen eine generische und typsichere Liste `MyList`. 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` (`https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Cloneable.html`) und der Methode `Object.clone()` (`https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Object.html#clone()`). Implementieren Sie das Interface `Cloneable` in `MyList`.

Aufgabe 6.2: `Cloneable` und `clone` - Testen (15 Punkte)

Testen Sie automatisiert in einer eigenen Testklasse, ob Sie die Methode `clone()` innerhalb der Klasse `MyList` 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.

Hinweis 1: Sie müssen in dieser Aufgabe nur die korrekte Funktion der `clone()`-Methode testen. Die anderen Methoden der Klasse `MyList`, bzw. der (ggf. von Ihnen modifzierten) Klasse `Entry`, brauchen Sie nicht zu testen.

Hinweis 2: Die absoluten und nicht absoluten Anforderungen an die Implementierung von `clone()` finden Sie in der API von Java (`https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Object.html#clone()`).

Aufgabe 6.3: Heap (35 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, welche in der zip-Datei zu dieser Aufgabe mitgeliefert ist. 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.

Aufgabe 6.4: Vergleich von Java Collections (20 Punkte)

Beurteilen Sie welche der `Collection`-Implementationen `java.util.LinkedList`, `java.util.ArrayList` und `java.util.HashSet` hinsichtlich der Methoden `add(T)` `remove(T)` und `contains(T)` die besten Laufzeiteigenschaften hat. Führen Sie dazu eine rein quantitative Analyse durch. Bilden Sie also durchschnittliche Werte für die Laufzeit der drei Methoden über eine ausreichende Anzahl von Testfällen. Geben Sie die Ergebnisse in einer aussagekräftigen Tabelle aus. Erklären Sie Ihrem Tutor/ Ihrer Tutorin ob und warum die Ergebnisse Ihren Erwartungen entsprechen. Gestalten Sie Ihre Implementation derart, dass möglichst einfach neue Klassen in den Vergleich mit aufgenommen werden können, ohne viel Quellcode duplizieren zu müssen.

Hinweis: Die aktuelle Systemzeit kann in Java mit `System.nanoTime()` abgefragt werden.

Aufgabe 6.5: Fragen (10 Punkte)

Beantworten Sie Ihrer Tutorin / Ihrem Tutor Fragen zum Thema Generics und Polymorphismus.