

## Übungsblatt 7

Abgabe bis Dienstag, 12.12.06, 11 Uhr

**Hinweis:** Programmieraufgaben immer per Email (eine Email pro Blatt und Gruppe) an den zuständigen Tutor schicken (Java Quellcode und eventuell benötigte Datendateien). Bitte werfen Sie Ihre schriftlichen Lösungen in die Briefkästen in Geb. 051, Erdgeschoss ein. Für den Erhalt von Bonuspunkten müssen Sie in wenigstens 10 Übungen anwesend sein.

### Aufgabe 1

Um die Konkurrenz zwischen den Kindern in diesem Jahr zu erhöhen, hat sich Santa ein neues Bewertungssystem einfallen lassen. Anstatt die Kinder in “nice” und “naughty” einzustufen, hat jetzt jedes Kind ein *naughtiness*-Wert zwischen 0.0 (“nice”) und 1.0 (“naughty”), der angibt wie “naughty” dieses Kind war.

Sie sind nun auserwählt worden, dieses System in Java zu implementieren.

1. Schreiben Sie eine Klasse `Child`, die ein Kind repräsentiert. Jedes Kind ist durch seinen Vornamen, Nachnamen und einen *naughtiness*-Wert gekennzeichnet.
2. Schreiben Sie danach ein Java Programm, das einen Vektor von `Child`-Objekten gemäß *naughtiness*-Wert aufsteigend sortiert. Haben zwei `Child`-Objekte den gleichen *naughtiness*-Wert, müssen sie gemäß Nachname sortiert werden. Sind die Nachnamen auch identisch, sollen sie gemäß Vorname sortiert werden.
3. Das Programm soll am Ende eine Liste mit den ersten 25% der Kinder aus dem sortierten Vektor ausgeben.
4. Auf der Vorlesungswebseite finden Sie eine Liste mit Vornamen, Nachnamen und *naughtiness*-Werten von Kindern<sup>1</sup>. Ihr Programm soll diese Liste einlesen, um den Vektor von `Child`-Objekten zu erzeugen.

### Aufgabe 2

Betrachten Sie die Klasse `Set`, wie Sie in der Vorlesung vorgestellt wurde (s. Kapitel 7).

1. Führen Sie eine Best- und Worst-Case-Aufwandsabschätzung für die `contains`- und `addElement`-Methode in Abhängigkeit von der Anzahl der Elemente in der Menge durch.

---

<sup>1</sup>Fast alle Vor- und Nachnamen wurden von “The Random Name Generator” (<http://www.kleimo.com/random/name.cfm>) erzeugt. Jegliche Übereinstimmung mit real existierenden Personen und *naughtiness* Werten ist natürlich rein zufällig und unbeabsichtigt.

2. Führen Sie eine Aufwandsabschätzung für die `copy`-Methode in Abhängigkeit von der Anzahl der Elemente in der Menge durch.
3. Ist es sinnvoll bei der `copy` Methode über einen Worst-, Best- oder Average-Case-Fall zu reden?

### **Aufgabe 3**

Schreiben Sie eine Klasse mit folgenden Methode:

- `static String[] addElement(String[] array, String s)`
- `static String[] addElementAt(String[] array, String s, int i)`
- `static String[] removeElementAt(String[] array, int i)`

Die Methode `addElement` fügt die Referenz auf das `String`-Objekt `s` am Ende des Arrays ein. Die Methode `addElementAt` fügt die Referenz auf das `String`-Objekt `s` an Position `i` ein und die `removeElementAt` Methode löscht die Referenz auf das `String`-Objekt an Position `i`. Bei der Durchführung aller drei Methoden soll die Anordnung der Elemente im Array unverändert bleiben. Die Methoden liefern eine Referenz auf das neue Array zurück. Führen Sie anschließend eine Aufwandsabschätzung für jede einzelne Methode durch.