Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
15. Januar 2018

11. Übung zur Vorlesung

Einführung in die Programmierung

A11-1 ***Generics I*** Gegeben sind Klassen `A`, `B`, `C`, `D` und die folgende Signatur: `public static A foo(A a, B b, C c, D d)` Ändern Sie diese schrittweise ab, so dass...

a) ...die Methode `foo` anstatt `A` als erstes Argument und Rückgabetyp einen beliebigen Typen akzeptiert, welcher das Interface `G` implementiert.

b) ...`foo` als 2. Argument eine `ArrayList` akzeptiert, deren Elemente Erben von `B` sind.

c) ...`foo` als 3. Argument alle Objekte eines Typen akzeptiert, welcher das parametrisches Interface `H<>` entweder direkt oder durch einen Vorfahren implementiert.

d) ...die Methode `foo` als 4. Argument nicht nur Objekte des Typs `D` akzeptiert, sondern auch alle Erben von `D`.

A11-2 ***Generics II*** Auf Vorlesungsfolie 13.44 ist eine Implementierung der binären Suche in einem sortiertem Array abgedruckt. Die gezeigte Methode hat folgende Signatur:
`public static boolean sucheVonBis(Comparable[] l, Object w, int i, int j)`

a) Auf der folgenden Folie wird dieser Typ zurecht bemängelt. Erklären Sie warum! Geben Sie ein Verwendungsbeispiel an, welches der Compiler akzeptiert, aber welches immer einen Laufzeitfehler liefert!

b) Geben Sie eine vernünftige, generische Signatur an! Überprüfen Sie auch, dass der Compiler ihr vorheriges Gegenbeispiel damit nicht mehr akzeptiert.

A11-3 ***Fakultätsfunktion*** Ein beliebtes einfaches Beispiel für Rekursion ist die Fakultät:

$$x! = \prod_{i=1}^{x} i = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (x-1) \cdot x$$

Gesucht ist eine Implementation für Signatur: `public static long fakultaet(int arg)`

a) Die mathematische Produktschreibweise könnten wir natürlich sofort als simple `for`-Schleife hinschreiben. Tun Sie dies!

b) Die Fakultät läßt sich natürlich auch rekursiv berechnen:

$$x! = \begin{cases} 1, & x = 0 \\ x \cdot (x-1)!, & x > 0 \end{cases}$$

Implementieren Sie die Fakultätsfunktion nun ohne jegliche iterative Schleifen, sondern nur unter Verwendung von Rekursion.

A11-4 ***Laufzeit I*** Welche Laufzeitkomplexität hat die folgende Methode:

```
public static int count(int[] a, int c) {
  int count = 0;
  for (int i=0; i < a.length; i++) { if (a[i] == c) count++; }
  return count;
}
```

H11-1 ***Laufzeit II*** (0 Punkte; Abgabe: `H11-1.txt` oder `H11-1.pdf`)
Angenommen ein Algorithmus benötigt 5 Sekunden um eine Eingabe mit 1000 Elementen zu verarbeiten. Berechnen Sie, wie lange der Algorithmus für andere Eingabegrößen ungefähr benötigt, unter Annahme folgender Laufzeitkomplexitäten:

	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$
1000	$5s$	$5s$	$5s$	$5s$	$5s$
2000					
3000			$45s$		
10000					

Beispiel: Wegen $\frac{3000^2}{1000^2} = 9$ benötigt der Algorithmus bei einer Eingabe der Größe 3000 und einer Laufzeitkomplexität von $O(n^2)$ also 9-mal so viel Zeit wie bei einer Eingabe der Größe 1000.

H11-2 ***Generics III*** (3 Punkte; BiSeCo.java)
Fortsetzung von Aufgabe A11-2, Teilaufgabe c):

Lesen Sie die Dokumentation des Interfaces `Comparator` aus der Standardbibliothek und implementieren Sie eine Variante der binären Suche mit folgender Signatur:

```
public static <T> boolean sucheVonBisC(Comparator<T> c, T[] l, T w, int i, int j)
```

H11-3 ***Duplikate*** (6 Punkte; NoDups.java)
Implementieren Sie folgende Methode:

```
public static <T> ArrayList<T> noDups(Comparator<T> cmp, ArrayList<T>)
```

welche als Rückgabewert eine Liste ohne Duplikate liefert, d.h. die Rückgabeliste enthält jeden Wert der Argumentliste genau einmal, wobei Gleichheit gemäß dem übergebenen `Comparator` entschieden wird. Geben Sie weiterhin die Laufzeit Ihrer Implementierung in O-Notation an und begründen Sie diese!

Die Punkte dieser Aufgabe werden für eine korrekte Implementierung und eine korrekt begründete Angabe der Laufzeit vergeben.

Jeweils 1 Bonuspunkt wird darauf vergeben, falls Ihre Implementierung eine Laufzeit besser als $O(n^2)$ hat, und wenn die Sortierung der ursprünglichen Argumentlist beibehalten wird (d.h. für die Eingabe [8, 2, 8, 3, 2, 8, 1, 2] wird [8,2,3,1] und nicht etwa [1,2,3,8] zurückgegeben).

Abgabe: Lösungen zu den Hausaufgaben können bis Sonntag, den 21.1.18, mit UniWorX nur als `.zip` abgegeben werden. Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (`www.tcs.ifi.lmu.de/lehre/ws-2017-18/eip/`).