Grundlagen der Programmkonstruktion
Übungsblatt 4 (zu lösen bis 10./11./12. Juni 2013)

1 Termination (0.6 Punkte)

Gegeben ist die folgende Methode `year(int days)`

```
public static boolean isLeapYear(int year) {
    // true, wenn year ein Schaltjahr ist, sonst false.
}

public static int year (int days) {
    // Gibt die aktuelle Jahreszahl zurueck, wenn
    // seit dem 1.1.1980 inklusive "days" Tage vergangen sind.
    // D.h. 1 entspricht dem 1. Jaenner 1980,
    // 366 entspricht dem 31. Dezember 1980 (1980 war ein Schaltjahr),
    // 367 ist der 1. Jaenner 1981, ...

    // days ist immer > 0

    int year = 1980;

    while (days > 365) {
        if (IsLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }

    return year;
}
```

1.1 Termination (0.2 Punkte)

Die Methode `years(int days)` ist fehlerhaft: Für bestimmte Eingaben terminiert die `while`-Schleife nicht. Finden Sie den Fehlerfall und geben Sie ein Beispiel an.

1.2 Korrektur (0.4 Punkte)

Korrigieren Sie die Methode (oder schreiben Sie sie komplett neu), sodass der Fehler nicht auftritt. Beachten Sie, dass die Methode ein richtiges Ergebnis liefern muss, d.h., `year(366)` muss 1980 zurückliefern, `year(367)` 1981.

2 Türme von Hanoi (0.5 Punkte)

Die Türme von Hanoi (siehe Wikipedia) sind ein beliebtes Beispiel zur Demonstration von Rekursion.

2.1 Nachvollziehen (0.2 Punkte)

Versuchen Sie die rekursive Methode nachzuvollziehen. Ermitteln Sie alle Aufrufe von `hanoi(3, ‘‘A’’, ‘‘B’’, ‘‘C’’);` und die Ausgabe der Methode in der richtigen Reihenfolge. Sie sollen in der Lage sein, genau zu erklären, warum wann welcher Aufruf erfolgt!

```
public static void hanoi(int n, String a, String b, String c) {
    if(n == 0) {
        return;
    } else {
        hanoi(n - 1, a, c, b);
        System.out.println("Von " + a + " nach " + c);
        hanoi(n - 1, b, a, c);
    }
}
```

2.2 Laufzeit bestimmen (0.3 Punkte)

Bestimmen Sie die Laufzeit (in Form der ausgegeben Zeilen) der Methode bei einem gegebenen n und geben Sie die algorithmische Komplexität in der O-Notation an. Begründen Sie Ihre Antwort.

3 Bubble Sort (1.4 Punkt)

Bubble Sort ist ein einfacher Sortieralgorithmus und funktioniert folgendermaßen: Es werden benachbarte Elemente in einem Array vertauscht, wenn Sie in der falschen Reihenfolge sind. Sollten keine Elemente in der falschen Reihenfolge sein, ist das Array sortiert.
Beispiel:

```
{"A", "D", "B", "C"}   ==> array[1] und array[2] werden vertauscht
{"A", "B", "D", "C"}   ==> array[2] und array[3] werden vertauscht
{"A", "B", "C", "D"}   ==> fertig.
```

3.1 Methode erstellen (1 Punkte)

Erstellen Sie die Methode `static void sort(String[] array)`. Diese soll ein String-Array mit Bubble-Sort sortieren. Sie sollen diese Methode für beliebige Beispiele nachvollziehen können! Zum Vergleichen von Zeichenketten benutzen Sie die `compareTo`-Methode.

3.2 Laufzeit ermitteln (0.4 Punkte)

Bestimmen Sie die Laufzeit ihrer Implementierung in Abhängigkeit von $n =$ `array.length`. Geben Sie zusätzlich die Laufzeit im best-case und im worst-case an und geben Sie String-Arrays (mit mindestens 4 Elementen) an, bei denen dieses Verhalten erreicht wird.