Lehrstuhl für Praktische Informatik IV
Holger Füßler
A5, 6, Raum B 219
68131 Mannheim
Telefon: (0621) 181–2605
Email: fuessler@informatik.uni-mannheim.de

Robert Schiele
B6, 29, Raum C0.04
68131 Mannheim
Telefon: (0621) 181–2214
Email: rschiele@uni-mannheim.de

UNIVERSITÄT MANNHEIM

Praktische Informatik I — 11. Übungsblatt
Wintersemester 2005/2006 — Abgabe: 25. Januar 2006

## Allgemeine Bearbeitungshinweise

Verwenden Sie für die Programmieraufgaben die in den Aufgaben angegebenen Klassennamen und geben Sie den Klassen-Quellcode sowohl auf Papier als auch per e-mail bei Ihrem Tutor ab. Falls nicht anders gefordert, lagern Sie sinnvoll Funktionalität in Methoden aus. Verwenden Sie Javadoc-Kommentare zur Dokumentation.
Aufgaben, bei denen die Punktzahl mit einem $*$ gekennzeichnet ist, zählen als Zusatzaufgaben zum Punktekonto, d.h. sie werden für die Berechnung der Zulassungsvoraussetzung im Zähler gewertet, aber nicht im Nenner.

## Aufgabe 1 — 10 Punkte

Im Folgenden geht es um die Laufzeit-Komplexität von gegebenen Java-Programmen. Geben Sie zunächst die exakte Formel für die Laufzeit und dann die zugehörige Komplexitätsklasse in O-Notation an.
Nehmen Sie hierfür folgende Laufzeitkosten an. (Die übrigen Kosten können vernachlässigt werden. Ferner können Sie annehmen, dass $n, m \geq 1$).

| | | |
|---|---|---|
| $c_z$ | : | Kosten für eine Zuweisung |
| $c_c$ | : | Kosten für einen Vergleich |
| $c_a$ | : | Kosten für eine arithmetische Operation |
| $c_i$ | : | Kosten für einen Methoden-Aufruf |

```
1  public static void methA(int n, int m) {
       int k = 0;
       for (int i = 9; i < n; i++)
           for (int j = 3; j < m; j++)
               k = (i*j)/3;
6  }

   public static void methB(int n, int m) {
       int k = 0;
       m = 150;
11     for (int i = 2; i < n; i++)
```

```
        for (int j = 2; j < m; j++)
            k = (i*j)/2 + 9;
}

public static void methC(int n, int m) {
    int k = 0;
    for (int i = 1; i < n; i++)
        for (int j = 1; j < m; j <<= 1)
            k = (i*j)/3 - 4;
}


public static void methD(int n, int m) {
    int k = 0;
    n = 150;
    for (int i = 1; i < m; i++)
        for (int j = i % n; j > 0; j--)
            k = (i*j)/3;
}

public static void methE(int n, int m) {
    int k = 0;
    n = 150;
    for (int i = 1; i < m; i++)
        for (int j = i; j > 0; j--)
            k = (i*(j+2))/3;
}
```

## Aufgabe 2 — 10 Punkte

### Aufgabe 2 a) — 5 Punkte

Entwickeln Sie einen deterministischen endlichen Automaten, der über dem Alphabet $\{0, 1\}$ genau diejenigen Wörter akzeptiert, die eine gerade Anzahl von Nullen und eine ungerade Anzahl von Einsen, mindestens von jedem Zeichen aber eines enthält.

### Aufgabe 2 b) — 5 Punkte

Entwickeln Sie einen deterministischen endlichen Automaten, der für ein beliebig langes Wort über dem Alphabet $\{a, b, c\}$ feststellt, ob das Teilwort *ababaca* enthalten ist.

## Aufgabe 3 — 10 Punkte

Für nichtnegative ganze Zahlen $n, k$ mit $0 \leq k \leq n$ und $n \neq 0$ wird der Binomialkoeffizient durch $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$ definiert. Für alle anderen ganzen Zahlen $n, k$ wird $\binom{n}{k} = 0$ definiert. Dabei bezeichnet $n!$ die Fakultät von $n$ ($0! = 1$).
Die folgende Klasse stellt eine Methode `binomial` zur Verfügung, welche den Bino-

mialkoeffizienten für zwei ganze Zahlen $n, k$ (zugegebenermaßen auf nicht sonderlich intelligente Weise) berechnen soll.

```
public class Binomial {
    public static long factorial(long n) {
        long r = (n == 0) ? 1 : n;
        while (--n > 1)
            r *= n;
        return r;
    }
    public static long binomial(long n, long k) {
        if (k < 0 || k > n)
            return 0;
        long r = factorial(n);
        r /= factorial(k);
        r /= factorial(n - k);
        return r;
    }
}
```

Im folgenden dürfen Sie davon ausgehen, dass es bei der Berechnung nicht zu Überläufen kommt und dass ausschließlich Werte aus dem Gültigkeitsbereich der jeweiligen Methode übergeben werden.

### Aufgabe 3 a) — 4 Punkte

Beweisen Sie zuerst die Korrektheit der angegebenen Implementation `factorial` zur Berechnung der Fakultät einer gegebenen Zahl $n$. Geben Sie hierzu eine aussagekräftige Schleifeininvariante an und zeigen Sie, dass diese gilt. Leiten Sie aus der Gültigkeit der Schleifeninvarianten direkt die Korrektheit des Algorithmusses her.
Hinweis: Es ist sinnvoll, zwischen dem inititalen Wert der Variablen `n` und der Variablen selbst durch Verwendung zweier Bezeichner zu unterscheiden. Geben Sie in diesem Fall aber auf jeden Fall an, welchen Bezeichner Sie wofür verwenden.

### Aufgabe 3 b) — 6 Punkte

Beweisen Sie die Korrektheit der angegebenen Implementation `binomial` zur Berechnung des Binomialkoeffizienten zweier gegebenen Zahlen $n, k$. Geben Sie hierzu vor und nach jeder Anweisung der Implementation eine Zusicherung (Assertion) an, die an dieser Stelle gilt und zeigen Sie dann jeweils, dass diese aus der vorhergehenden Zusicherung und der dazwischenliegenden Anweisung hergeleitet werden kann.
Hinweis: Wenn Sie zu schwache Zusicherungen wählen, wird es Ihnen unter umständen nicht gelingen, die Ableitung schlüssig durchzuführen.

## Aufgabe 4 * 10 Punkte

### Aufgabe 4 a) 4 Punkte

Eine Methode mit der Signatur `boolean isNMMatrix(int[][] matrix, int n, int m)` soll prüfen, ob eine übergebene `int[][]`-Variable eine gültige $n \times m$ Matrix ist. Analysieren Sie, was alles schief gehen kann, und spezifizieren Sie einen sinnvollen `black-box-test` für diese Methode in Form eines Java-Programms, das eine Fehlermeldung ausgibt, wenn die Methode nicht wie erwartet arbeitet.

### Aufgabe 4 b) 6 Punkte

Gegeben sei folgende Methode zur Umrechnung von Oktalziffern in eine Dezimalzahl. Testen Sie diese Methode als *white-box*, einmal als Anweisungs- und einmal als Wege-Test, d.h. definieren Sie eine Menge von Eingabedaten / Funktionsaufrufen, die die für diese Tests erforderlichen Kriterien erfüllen. Erstellen Sie hierfür zuerst ein Flussdiagramm. Wie hängt die Anzahl der Wege durch den Programmfluss von `octals.length` ab?

```
public static int getDecFromOct(int[] octals) {
    int retval = 0;
    if (octals == null)
        retval = -1;
    else {
        for (int i = 0; i < octals.length; i++) {
            if (octals[i] < 0)
                retval = -1;
            if (octals[i] > 7)
                retval = -1;
        }
        if (retval == 0) {
            for (int expo = 0; expo < octals.length; expo++)
                retval += (octals[i] << (3 * expo));
        }
    }
    return retval;
}
```