Lehrstuhl für Praktische Informatik IV
Prof. Dr. Wolfgang Effelsberg
A5, 6, Raum B 223
68131 Mannheim
Telefon: (0621) 181–2600
Email: effelsberg@informatik.uni-mannheim.de

Holger Füßler
Email: fuessler@informatik.uni-mannheim.de

Robert Schiele
Email: rschiele@uni-mannheim.de

UNIVERSITÄT MANNHEIM

Praktische Informatik I — Modulklausur
Wintersemester 2005/2006 — 9. März 2006

Hinweise

1. Überprüfen Sie bitte Ihr Klausurexemplar auf Vollständigkeit (14 Seiten).
2. Bearbeiten Sie die Aufgaben *ausschließlich* auf dem Aufgabenblatt der jeweiligen Aufgabe.
3. Schreiben Sie auf jedes Blatt, das bewertet werden soll, oben ihren Namen und ihre Matrikelnummer.
4. Verwenden Sie nur dokumentenechte Stifte (z. B. keinen Bleistift) und keine roten Stifte.
5. Es sind keine Hilfsmittel zugelassen.
6. Bearbeitungszeit: 100 Minuten.

Korrekturzeile — Bitte *nicht* ausfüllen!

1	2	3	4	5	gesamt
22	17	17	19	25	100

Name: Matrikelnummer:

Name: Matrikelnummer:

Aufgabe 1 — 22 Punkte

Aufgabe 1 a) — 2 Punkte

Erklären Sie kurz, was man in Java unter dem Überladen (nicht dem Überschreiben) von Methoden versteht und was man dabei beachten muss.

Name: Matrikelnummer:

Aufgabe 1 b) 4 Punkte

Gegeben sei der folgende Java-Code:

```
1  public class Position {
       double x;
       double y;
       public Position(){}
       public Position(double x, double y) {
6          this.x = x;
           this.y = y;
       }
       public static void main(String[] args) {
           Position p1 = new Position(3.0, 5.0);
11         Position p2 = new Position(9.0, 2.0);
           p1.y = 10.0;
           p1 = p2;
           p2.x = 17.2;
           System.out.printf("%f:%f %f:%f\n", p1.x, p1.y, p2.x, p2.y);
16     }
   }
```

Was wird in Zeile 15 ausgegeben?

Aufgabe 1 c) 4 Punkte

Ergänzen Sie die Klasse Position um eine Methode `Position copy()`, die eine Kopie der eigenen Instanz zurückgibt.

Aufgabe 1 d) 3 Punkte

Welche Art von Typangleichungen führt Java bei Zuweisungen/Methodenaufrufen implizit aus? (Konkrete Beispiele sind nicht verlangt.) Wie führt man eine explizite Typangleichung durch? Was muss man hierbei beachten?

Name: Matrikelnummer:

Aufgabe 1 e) 9 Punkte

Gegeben sei der folgende Java-Code:

```
public interface Ernten { /* ... */ }

public class Getreide implements Ernten { /* ... */ }

public class Roggen extends Getreide {
    public static void test(Roggen rog, Getreide getr, Ernten ernten,
                            int i, long l, byte b, short s,
                            double d, float f, boolean bool) {
        ernten = rog;          // Zeile A
        rog = getr;            // Zeile B
        getr = rog;            // Zeile C
        f = d;                 // Zeile D
        i = (int) l;           // Zeile E
        bool = (i == 0);       // Zeile F
        b = (bool ? 0 : 1);    // Zeile G
        i = s;                 // Zeile H
    }
}
```

Beschreiben Sie für jede der mit *A* - *H* gekennzeichneten Zeilen, ob die dort aufgeführte Zuweisung zulässig ist. Begründen Sie Ihre Ausssagen.

Name: Matrikelnummer:

Aufgabe 2 — 17 Punkte

Aufgabe 2 a) — 2 Punkte

Sie möchten eine Menge von `int` Werten speichern. Als Datenstrukturen stehen ein Binärbaum oder eine einfach verkettete Liste zur Auswahl. Beschreiben und begründen Sie kurz, wann Sie die eine und wann Sie die andere Struktur verwenden würden.

Aufgabe 2 b) — 10 Punkte

Gegeben sei die folgende Listenimplementierung. Das letzte „Element“ zeigt auf `null` und enthält 0 als Wert, d.h. eine Liste, die n Integer enthält, hat immer $n + 1$ Listenobjekte.

Implementieren Sie eine *iterative* Instanz-Method `int count()`, die die Anzahl der in der Liste gespeicherten `int`s zurückgibt. Implementieren Sie eine *rekursive* Instanz-Methode `int sum()`, die die Summe der gespeicherten `int`s zurückgibt.

```
class Liste {
  int wert;
  Liste rest;
  Liste() { wert = 0; rest = null; }
  // ... restliche Methoden
}
```

Aufgabe 2 c) 5 Punkte

Dem gleichen Muster folgt die Binärbaum-Implementierung.

```
class BinBaum {
  int wert;
  BinBaum kleinerGleich;
  BinBaum groesser;
  BinBaum() { wert = 0; kleinerGleich = null; groesser = null; }
  // ... restliche Methoden
}
```

Implementieren Sie eine *rekursive* Instanz-Methode `int countLeaves()`, die die Anzahl der Blätter eines solchen Baumes zurück gibt.

Name: Matrikelnummer:

Aufgabe 3 — 17 Punkte

Aufgabe 3 a) — 4 Punkte

Erklären Sie kurz, wozu man Syntaxdiagramme benutzt, welche Arten von Symbolen man unterscheidet, und wie diese definiert sind.

Aufgabe 3 b) — 3 Punkte

Entscheiden Sie, ob die folgenden Sprachelemente in Java gültige Literalkonstanten sind. Wenn gültig, geben Sie jeweils den (kleinsten) Typ an. Wenn ungültig, den Grund für den Syntaxfehler. *(Hinweis: Der Buchstabe „O“ wie Oskar kommt in den Elementen nicht vor.)*

Element	Gültig	Nicht Gültig	Typ	Warum ungültig
`´ab´`	☐	☐		
`09833539`	☐	☐		
`.32345E+2f`	☐	☐		
`0x32BASL`	☐	☐		
`"h"`	☐	☐		
`0xFFFF`	☐	☐		

Name: Matrikelnummer:

Aufgabe 3 c) 10 Punkte

Entwickeln Sie ein Syntaxdiagramm, das alle Ausdrücke der Art Berechnung `(<operator> <arg1> ... <argN>)` akzeptiert. Operatoren können $\{+, -, *, /\}$ sein. Die Argumente sind einfache Integer-Zahlen oder wiederum Berechnungen. Es muss mindestens ein Argument angegeben sein.

Beispiele: `(+ 1 3 52)`, `(+ (- 3 9) 2 4)`.

Name: Matrikelnummer:

Aufgabe 4 — 19 Punkte

Im Folgenden betrachten wir die Methode `binomial`:

```
public static long binomial(long n, long k) {
    if (k < 0 || k > n)
        return 0;
    long r = 1;
    if (2 * k > n)
        k = n - k;
    long i = n - k;
    while (++i <= n)
        r *= i;
    while (k > 1)
        r /= k--;
    return r;
}
```

Aufgabe 4 a) — 9 Punkte

Zeichnen Sie ein Flussdiagramm für die angegebene Methode.

Aufgabe 4 b) 4 Punkte

Geben Sie eine sinnvolle Grenze für die Laufzeitkomplexität für die Methode in der O-Notation an. Begründen Sie Ihre Wahl kurz.

Aufgabe 4 c) 6 Punkte

Geben Sie eine minimale Menge von Testfällen für einen vollständigen Verzweigungstest an.

Name: Matrikelnummer:

Aufgabe 5 — 25 Punkte

In der Regel können heutige Prozessoren ganze Zahlen zwar direkt addieren oder multiplizieren, allerdings nicht potenzieren. Will man also Berechnungen der Art a^n durchführen, so muss dafür ein eigener Algorithmus entwickelt werden. Ein einfacher, aber sehr langsamer Algorithmus für ganze Zahlen wäre zum Beispiel, die Zahl a n-mal auf die 1 als Startwert zu multiplizieren.

Aufgabe 5 a) — 5 Punkte

Beschreiben Sie diesen einfachen Algorithmus mit Hilfe eines Struktogramms.

Aufgabe 5 b) — 5 Punkte

Implementieren Sie den Algorithmus in der Methode
`static long exp(long a, long n)`.

Aufgabe 5 c) 10 Punkte

Der obige Algorithmus ist allerdings ineffizient, weshalb Sie im folgenden einen Algorithmus aus der folgenden Idee entwickeln sollen:

Der Ausdruck a^n kann auch geschrieben werden als $a^n = \prod_i a^{(2^{n_i})}$, mit beliebigen ganzen Zahlen n_i, für die gilt $n = \sum_i 2^{n_i}$.

Entwickeln Sie also einen nicht-rekursiven Algorithmus, der die Zahl n in eine Summe von Zweierpotenzen zerlegt und damit dann die einzelnen Komponenten gemäß obiger Formel aufmultipliziert.

Hinweise:

- Beachten Sie, dass gilt: $a^{(2^{n_i})} = \left(a^{\left(2^{n_i-1}\right)}\right)^2$
- Wenn Sie die genannten Formeln nicht verstehen, so versuchen Sie, sich diese durch einfache Rechenbeispiele mit kleinen Zahlen klar zu machen.
- Eine Ganzzahl n kann auf einem Rechner sehr leicht in eine Summe von Zweierpotenzen zerlegt werden, indem jedes Bit der Zahl einzeln betrachtet wird.

Beschreiben Sie Ihren Algorithmus mit Hilfe eines Struktogramms.

Aufgabe 5 d) 5 Punkte

Implementieren Sie mit Ihrem Algorithmus die Methode
`static long fastexp(long a, long n)`.