Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

11. Februar 2005

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im WS 2004/05 (WI h103, II h105, MI h353)

Zeit: 150 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg !

Diese Klausur besteht einschließlich dieses Deckblattes aus 14 Seiten

---

Aufgabe 1:

Gegeben sei das folgende Java Programm, bestehend aus zwei Schnittstellen und zwei Klassen. In der Methode *test* der Klasse *X* sind verschiedene Ausdrücke enthalten. Überprüfen Sie, ob die Ausdrücke erlaubte Java-Ausdrücke sind. Wenn dies nicht der Fall ist, kennzeichnen Sie die Ausdrücke mit dem Wort *error*, wenn die Ausdrücke wohlgeformt sind, bestimmen Sie den Typ und notieren diesen in der entsprechenden Zeile.

```
interface IF1 {
    IF1 if1(IF1 x);
}
interface IF2 {
    IF2 if2();
}
class Y implements IF1 {
    Y y1;
    public IF1 if1(IF1 x) {
        return x;
    }
}
class X extends Y implements IF2 {
    int i1;
    long l1;
    double d1;
    boolean b1;
    int [] ia1;

    Integer i2;
    Integer [] ia2;
    Double d2, d3;
    X x1;
    X [] xa1;
    X [] [] xm1;
    Y [] ya1;
    IF1 f1;
    IF2 f2;
    Object o1;
    Object [] oa1;

    public IF2 if2() {
        return this;
    }
    void test() {
        // int dieser Methode stehen die folgenden zu überprüfenden Ausdrücke
        // ...
    }
}
```

i1 + 1 .....

i1 + l1 .....

i1 + d1 .....

i1++ .....

i2++ .....

i1 + i2 .....

d2 + d3 .....

o1 = ia1 .....

oa1 = ia1 .....

o1 = ia1[i1] .....

oa1 = ia2 .....

o1 = xa1 .....

o1 = xa1[i1] .....

xa1 = oa1 .....

xa1 = (X[])oa1 .....

xa1[i1] = (X)oa1[i1] .....

x1 == i2 .....

x1 == y1 .....

x1 == null .....

y1.if1(**this**) .....

x1.if2() .....

x1.if2().if2() .....

x1.if2().if2().if1(x1) .....

if1(y1).test() .....

test() .....

f2 == y1 .....

f1 == f2 .....

f1 = y1 .....

f2 = y1 .....

(Y)if2 .....

Aufgabe 2:

Die folgenden Fragen beziehen sich alle auf Java Programme, die mit Threads arbeiten. Es wird dabei angenommen, dass kein Thread sich unbeschränkt lange in einem Monitor aufhält, zum Beispiel indem er in eine Endlosschleife läuft.

1. Threads brauchen nicht synchronisiert werden, wenn sie gemeinsame Variablen nur lesen.

   ja ☐ nein ☐

   Begründung:

   .....

2. Nur die Threads, die gemeinsame Variablen auch beschreiben, müssen beim Zugriff auf die Variablen synchronisiert werden. Die ausschließlich lesenden brauchen nicht synchronisiert werden.

   ja ☐ nein ☐

   Begründung:

   .....

3. Threads brauchen nicht synchronisiert werden, wenn sie gemeinsame Variablen nur schreiben.

   ja ☐ nein ☐

   Begründung:

   .....

4. Nichtdeterministische Thread–Programme sind grundsätzlich unbrauchbar.

   ja ☐ nein ☐

   Begründung:

   .....

5. Wenn es zur Berechnung einer Aufgabe ein deterministisches Programm und ein nichtdeterministisches Programm gibt, so ist das deterministische das effizientere.

   ja ☐ nein ☐

   Begründung:

   .....

6. Nichtdeterministische Programme liefern nur mit hoher Wahrscheinlichkeit die richtigen Ergebnisse.

   ja ☐ nein ☐

   Begründung:

   .....

7. Thread–Programme, die nur mit einer einzigen gemeinsamen synchronisierten Variablen arbeiten, sind immer deadlock–frei.

   ja ☐ nein ☐

   Begründung:

   .....

8. Bei Thread–Programmen, die mit geschachtelten Monitoren arbeiten, besteht immer die Gefahr eines deadlocks.

   ja ☐ nein ☐

   Begründung:

   .....

9. Bei Thread–Programmen müssen auch die Konstruktoraufrufe von gemeinsam genutzten Klassen synchronisiert werden.

   ja ☐ nein ☐

   Begründung:

   .....

10. Methoden die nur lesend auf Objekte zugreifen, brauchen nicht synchronisiert werden.

    ja ☐ nein ☐

    Begründung:

    .....

11. Container-Klassen sollten grundsätzlich so entwickelt werden, dass sie Thread–sicher sind.

    ja ☐ nein ☐

    Begründung:

    .....

12. Das Attribut *synchronized* wird auch bei der Typüberprüfung von Methodenaufrufen verwendet.

    ja ☐ nein ☐

    Begründung:

    .....

Aufgabe 3:

Die folgenden Klassen dienen zur Implementierung und Auswertung von aussagenlogischen Ausdrücken mit Java. Die Wurzelklasse ist **BoolExpr**. Von dieser sind zwei Unterklassen abgeleitet worden, **BoolExpr1** für einstellige und **BoolExpr2** für zweistellige Operationen.

```
public abstract class BoolExpr {
    public abstract boolean eval();
}

abstract class BoolExpr1 extends BoolExpr {
    protected BoolExpr operand;

    protected BoolExpr1(BoolExpr o) {
        operand = o;
    }
}

abstract class BoolExpr2 extends BoolExpr {
    protected BoolExpr left, right;

    protected BoolExpr2(BoolExpr l, BoolExpr r) {
        left = l;
        right = r;
    }
}
```

Entwickeln Sie eine konkrete Unterklasse **BoolConst** für die Repräsentation der beiden Wahrheitswerte **false** und **true**.

Entwickeln Sie eine konkrete Klasse **NotExpr** für die Negation.

Entwickeln Sie eine konkrete Klasse **AndExpr** für die Konjunktion (logisches UND). Die Auswertung der Teilausdrücke soll wie in Java von links nach rechts durchgeführt werden bis das Resultat feststeht.

Entwickeln Sie eine konkrete Klasse **ImpliesExpr** für die Implikation. Die Auswertung der Teilausdrücke soll nicht wie bei dem logischen UND nicht strikt durchgeführt werden, sondern strikt, d.h. beide Teilausdrücke sollen ausgewertet werden.

Aufgabe 4:

Gegeben sei das folgende C–Programm:

```
#include <stdio.h>

int h(int i) {
  printf("h");
  return i < 8 ? i : h( (i & 7) + (i >> 3));
}

int g(int i) {
  printf("g");
  return ( (i & 7) == 5) || (i >= 8 && g(i >> 3));
}

int f(int i) {
  printf("f");
  return (i % 5 == 0) || g(i) || (h(i) == 5);
}

int main(void) {
  printf(" %d\n",f(12));
  printf(" %d\n",f(13));
  printf(" %d\n",f(14));
  printf(" %d\n",f(15));

  return 0;
}
```

Welche vier Ausgabezeilen erzeugt dieses Programm:

1) .....

2) .....

3) .....

4) .....

Aufgabe 5:

Gegeben sei das folgende C–Programm:

```
#include <stdio.h>

#define einsMinus(x) 1 − x
#define zweiMinus(x) (2 − x)
#define dreiMinus(x) (3 − (x))

int main(void) {
  printf("r1 = %d\n", einsMinus(5+1));
  printf("r2 = %d\n", einsMinus(einsMinus(5)));
  printf("r3 = %d\n", zweiMinus(6+1));
  printf("r4 = %d\n", zweiMinus(zweiMinus(6)));
  printf("r5 = %d\n", dreiMinus(dreiMinus(7)));
  printf("r6 = %d\n", 2 ∗ einsMinus(5));
  printf("r7 = %d\n", einsMinus(5) ∗ 2);
  printf("r8 = %d\n", zweiMinus(6) ∗ 2);
  printf("r9 = %d\n", einsMinus(2<<3));
  printf("r10 = %d\n", zweiMinus(3<<2));
  printf("r11 = %d\n", dreiMinus(3<<2));
  printf("r12 = %d\n", 2 / einsMinus(1 + 2 ∗ 2));
  return 0;
}
```

Welche Ausgabezeilen erzeugt dieses Programm:

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....
