Prof. Dr. Uwe Schmidt

ptlwedel
PHYSIKALISCH-TECHNISCHE LEHRANSTALT

21. August 2007

Aufgaben zur Klausur **Objektorientierte Programmierung** im SS 2007 (IA 252)

Zeit: 75 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 7 Seiten

---

Aufgabe 1:

Generische ADTs werden in C++ durch sogenannte *templates* unterstützt, in Java ab 1.5 durch *generics*. Überprüfen Sie die folgenden Aussagen über generische ADTs.

1. Typparameter für Klasen werden zur Laufzeit des Programms in zusätzlichen Feldern in den Objekten gespeichert und führen so zu größerem Speicherbedarf zur Laufzeit.

   ja ☐ nein ☐

   Begründung:

   .....

2. Typparameter für Klassen sind ein Konzept, das die Flexibilität von Java– und C++–Programmen einschränkt.

   ja ☐ nein ☐

   Begründung:

   .....

3. In Java Programmen mit Typparametern lassen sich up– und down–casts zur Laufzeit vermeiden.

   ja ☐ nein ☐

   Begründung:

   .....

4. In Programmen mit Typparametern können Fehler in der Verwendung von Methoden und Variablen teilweise schon zur Übersetzungszeit entdeckt werden.

   ja ☐ nein ☐

   Begründung:

   .....

5. In Java Programmen mit Typparametern können zur Laufzeit noch mehr Überprüfungen gemacht werden als ohne.

   ja ☐ nein ☐

   Begründung:

   .....

6. In Programmen mit Typparametern können Laufzeit–Überprüfungen auf legale Feldzugriffe oder das Dereferenzieren von null-Referenzen teilweise eliminiert werden.

   ja ☐ nein ☐

   Begründung:

   .....

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

   ja ☐ nein ☐

   Begründung:

   .....

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

   ja ☐ nein ☐

   Begründung:

   .....

Aufgabe 3:

Gegeben sei das folgende Java–Programm:

```
class X {
    int i;

    X ref;

    X() { i = 0; ref = this; }

    void out() { System.out.println("i = " + i); }

    void f() { i = 1; }

    void g() { ++i; }

    void h() { f(); ref.g(); }
}

class Y extends X {
    void f() { i = 2; }
}

class Z extends Y {
    void g() { i += 3; }
}

class U extends X {
    void h() { super.h(); ref.h(); }
}

public class Virtual {
    public static void main(String [] argv) {
        X b = new Z();
        X c = new Y();
        X d = new X();
        X e = new U();

        b.h(); b.out();
        c.h(); c.out();
        d.h(); d.out();
        e.h(); e.out();
    }
}
```

Welche Ausgabezeilen erzeugt dieses Programm?

1) .....

2) .....

3) .....

4) .....

Aufgabe 4:

Gegeben sei das folgende Java–Programmstück. Es soll überprüft werden ob die Zuweisungen und Konversionen legal sind. Hierbei sind drei Fälle zu unterscheiden:

1. Die Korrektheit der Konversion kann statisch zur Übersetzungszeit überprüft werden und ist erlaubt.
   Dies ist zu kennzeichnen durch Ankreuzen von ☐ ct.
2. die Korrektheit der Konversion wird zur Laufzeit überprüft.
   Dies ist zu kennzeichnen durch Ankreuzen von ☐ rt.
3. es kann zur Übersetzungszeit festgestellt werden, daß die Konversion fehlerhaft ist.
   Dies ist zu kennzeichnen durch Ankreuzen von ☐ err.

```
interface J { }

interface I extends J { }

class A { }

class B extends A { }

class C extends B implements I { }

class D extends A implements J { }
```

```
class Conversion {
    Object o; Object [] oa;

    A a; B b; C c; D d; J j; I i;

    A [] aa; B [] ba;

    void f() {

        oa = (Object [])a;          ct  rt  err
        oa = (Object [])aa;         ct  rt  err
        oa = (Object [])o;          ct  rt  err

        d = (D)a;                   ct  rt  err
        d = (D)b;                   ct  rt  err
        d = (D)j;                   ct  rt  err
        d = (D)o;                   ct  rt  err

        b = (B)aa;                  ct  rt  err
        b = (B)(aa[0]);             ct  rt  err
        b = (B)c;                   ct  rt  err

        j = (J)a;                   ct  rt  err
        j = (J)ba;                  ct  rt  err
        j = (J)(ba[0]);             ct  rt  err
        j = (J)c;                   ct  rt  err
        j = (J)i;                   ct  rt  err
    }
}
```