Physikalisch–Technische Lehranstalt Wedel
Prof. Dr. Uwe Schmidt

10. August 2001

Aufgaben zur Klausur **Objektorientierte Programmierung** im SS 2001 (IA 252)

Zeit: 60 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 9 Seiten

---

Aufgabe 1:

Gegeben sei eine Klassenhierarchie zur Verarbeitung von einstelligen reellwertigen Funktionen, d.h. reelwertige Funktionen werden durch Java–Objekte dargetellt.

Die Schnittstelle für die Funktionen wird als interface realisiert. Sie enthält eine Methode *at* zur Berechnung der Funktion an einer Stelle x. Außerdem ist eine Methode *derive* (= ableiten) zur Berechnug der 1. Ableitung zu implementieren. Drei häufig verwendete Funktionen werden beim Laden der Schnittstelle erzeugt. Diese sind global zugreifbar.

Die Schnittstelle:

```
public
interface Function {

  public
  double at(double x);

  public
  Function derive();

  public static final Function sin = new Sine();
  public static final Function cos = new Cosine();
  public static final Function zero = new ConstFunction(0.0);

}
```

Die Klasse für die Sinus–Funktion (*Sine*), die Klasse für die Kosinus–Funktion (*Cosine*) sei analog definiert.

```
public
final
class Sine implements Function {

  public
  double at(double x) {
    return
      Math.sin(x);
  }

  public
  Function derive() {
    return
      Function.cos;
  }
}
```

Die Klasse für konstante Funktionen (*ConstFunction*):

```
public
final
class ConstFunction implements Function {
  private
  double c;

  public
  ConstFunction(double c) {
    this.c = c;
  }

  public
  double at(double x) {
    return
      c;
  }

  public
  Function derive() {
    return
      Function.zero;
  }
}
```

Erweitern Sie diese Klassenhierarchie um eine Klasse *AddFunction* für Funktionen der Form $f(x) = f_1(x) + f_2(x)$. Hinweis: $f'(x) = f_1'(x) + f_2'(x)$

Erweitern Sie diese Klassenhierarchie um eine Klasse *MultFunction* für Funktionen der Form $f(x) = f_1(x) * f_2(x)$. Hinweis: $f'(x) = f_1'(x) * f_2(x) + f_1(x) * f_2'(x)$

Entwerfen Sie eine Schnittstelle *FindZero* für Algorithmen, die zu einer Funktion f in einem Intervall x_1 bis x_2 eine Nullstelle suchen.

Ist es sinnvoll, die konkreten Klassen in diesem Beispiel als **final** zu deklarieren?

ja ☐ nein ☐

Begründung:

.....

.....

.....

Kann das **final** Attribut vom Compiler im Allgemeinen zur Verbesserung des JVM–Codes genutzt werden?

ja ☐ nein ☐

Begründung:

.....

.....

.....

Die Schnittstelle *Function* ist als Java–interface deklariert. Ist diese Deklaration von Vorteil gegenüber einer Deklaration als abstrakte Klasse?

ja ☐ nein ☐

Begründung:

.....

.....

.....

Aufgabe 2:

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
    }
}
```
