Prof. Dr. Uwe Schmidt

ptlwedel
PHYSIKALISCH-TECHNISCHE LEHRANSTALT

17. August 2006

Aufgaben zur Klausur **Objektorientierte Programmierung** im SS 2006 (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 12 Seiten

---

Aufgabe 1:

Bitte lesen Sie diese Aufgabe vollständig durch, bevor Sie beginnen, die einzelnen Lösungsteile zu entwickeln.

In dieser Aufgabe soll ein einfacher *Baukasten* für die Verarbeitung von Zahlenfolgen entwickelt werden. Zahlenfolgen besitzen folgende Schnittstelle:

```
interface Sequence {
    double next();
}
```

Die Zahlenfolgen werden wie Datenströme verarbeitet, es gibt eine Methode zur Berechnung des nächsten Elements in der Folge. Die hier verwendete Technik findet sich zum Beispiel in der Klasse Reader im Paket java.io und deren Unterklassen wieder, insbesondere in den Filterklassen. Da hier mit unbeschränkt langen Zahlenfolgen gearbeitet wird, gibt es im Gegensatz zu Eingabe–Datenströmen keinen Test auf Ende der Zahlenfolge.

Um Zahlenfolgen zu erzeugen, gibt es einige einfache Klassen. Mit der Klasse *Const* können konstante Zahlenfolgen generiert werden.

```
public
class Const implements Sequence {
  private
  double value;

  public Const() { this(0); }

  public Const(double value) { this.value = value; }

  public double next() { return value; }
}
```

Mit *Count* kann gezählt werden $(0, 1, 2, ...$ oder $1, 2, 3, ...)$

```
public
class Count implements Sequence {
  private
  double cnt;

  public Count() { this(0); }

  public Count(double start) { cnt = start; }

  public double next() { return cnt++; }
}
```

Entwickeln Sie eine Klasse *Fibonacci* für die Fibonacci-Zahlenfolge ab 0: $0, 1, 1, 2, 3, 5, 8, 13, ...$
Tipp: Um unnötige Verzweigungen zu vermeiden berechnen Sie immer einen Folgewert im Voraus.

```
public
class Fibonacci implements Sequence {
  private
  double x0,x1;

  public Fibonacci() {

    .....

    .....
  }

  public double next() {
    double res;

    .....

    .....

    .....

    .....

    .....
    return res;
  }
}
```

Aus einer Zahlenfolge können neue Zahlenfolgen erzeugt werden. Eine einfache Art ist die, jeden Folgenwert mit einem Faktor zu multiplizieren. Entwickeln Sie hierfür eine Klasse *Scale*:

```
public
class Scale implements Sequence {
  private
  double factor;

  private

  ..........................................

  public Scale( ..........................................  ) {

    ..........................................

    ..........................................
  }

  public double next() {

    ..........................................

    ..........................................

    ..........................................

    ..........................................
  }
}
```

Eine weitere in der Mathematik häufig verwendete Operation für das Erzeugen einer neuen Zahlenfolge ist das Aufsummieren der ersten n Folgeglieder. Dieses soll hier mit Hilfe der Klasse *Sum* realisiert werden. Der Algorithmus soll so arbeiten, dass für die Zahlenfolge $1, 1, 1, ...$ (*new Const(1.0)* die Zahlenfolge $0, 1, 2, 3, ...$ entsteht.

```
public class Sum implements Sequence {
  private

   ..............................................

  private
  Sequence s;

  public Sum( ..............................................) {

     ..............................................

     ..............................................
  }

  public double next() {
    double res;

     ..............................................

     ..............................................

     ..............................................
    return res;
  }
}
```

Zahlenfolgen können auch durch Kombination zweier anderer Zahlenfolgen konstruiert werden, zum Beispiel durch Aufsummieren der Folgeglieder. Hier soll eine Klasse zum Mischen zweier Zahlenfolgen entwickelt werden. Das Mischen soll so arbeiten, dass immer der kleinere Wert als erstes genommen wird. Aus den Folgen $1, 2, 4, 8, 16, ...$ und $0, 1, 1, 2, 3, 5, 8, 13, ...$ soll damit die Zahlenfolge $0, 1, 1, 1, 2, 2, 3, 4, 5, 8, 8, 13, 16, ...$ entstehen. Tipp: Man muss in den Ausgangszahlenfolgen immer einen Schritt vorausschauen.

```
public class Merge implements Sequence {
  private

  .....
  private

  .....
  public Merge( ..... ) {

    .....

    .....
  }

  public double next() {
    double res;

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....
    return res;
  }
}
```

Aufgabe 2:

Gegeben sei das folgende Java Programm, bestehend aus zwei Schnittstellen und zwei Klassen. In der Methode *test* der Klasse *Y* 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();
}
interface IF2 {
    IF2 if2(IF2 x);
}
class X implements IF2 {
    X x1;
    public IF2 if2(IF2 x) {
        return x;
    }
}
class Y extends X implements IF1 {
    int i1;
    int [] ia1;

    Integer [] ia2;
    Y y1;
    Y [] ya1;
    Y [] [] ym1;
    X [] xa1;
    IF2 f2;
    IF1 f1;
    Object o1;
    Object [] oa1;

    public IF1 if1() {
        return this;
    }
    void test() {
        // in dieser Methode stehen die folgenden zu überprüfenden Ausdrücke

        // ...
    }
}
```

o1 = ia1 .....

oa1 = ia1 .....

o1 = ia1[i1] .....

oa1 = ia2 .....

o1 = ya1 .....

o1 = ya1[i1] .....

ya1 = oa1 .....

ya1 = (Y[])oa1 .....

ya1[i1] = (Y)oa1[i1] .....

y1 == x1 .....

y1 == null .....

x1.if2(**this**) .....

y1.if1() .....

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

y1.if1().if1().if2(y1) .....

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

f1 == x1 .....

f1 == f2 .....

f2 = x1 .....

(X)f2 .....

Aufgabe 3:

Gegeben sei die folgende Klasse:

```
class X {
    int x1;

    void reset() {
        x1 = 0;
    }

    Y f() {
        return
            new Y();
    } // end f


    Y g() {
        return
            new Y() {
                void f() {
                    --y1;
                    --x1;
                }
            };
    } // end g


    class Y {
        int y1;

        void f() {
            reset();
            ++y1;
            ++x1;
        }

    } // end Y

} // end X
```

In diesem Beispiel werden geschachtelte Klassen genutzt. Transformieren Sie dieses Programmstück in ein gleichwertiges, in dem ausschließlich mit *toplevel*–Klassen gearbeitet wird.

Die neue Klasse für X:

Die neue(n) *toplevel*–Klasse(n):