Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

3. Februar 2010

Aufgaben zur Klausur **Objektorientierte Programmierung** im WS 2009/10 (BInf 211, BTInf 211, BMInf 211, BWInf 211)

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 9 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 Schnittstellen:

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

interface Duplicate {
    Duplicate dup();
}
```

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. Zusätzlich soll es über die Schnittstelle Duplicate möglich sein, eine neue Zahlenfolge zu erzeugen, die die gleiche Folge liefert wie die Ausfolge, unabhängig davon, wieviele Elemente schon aus der Ausgangsfolge ausgelesen wurden.

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

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

    public Const() { this(0); }
    public Const(double value) { this.value = value; }

    public double next() { return value; }

    public Duplicate dup() {

        . . . . . .

        . . . . . .
    }
}
```

Vervollständigen Sie die Klasse.

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) {

        . . . . . .

        . . . . . .
    }
    public double next() {
        return cnt++;
    }
    public Duplicate dup() {

        . . . . . .

        . . . . . .
    }
}
```

Vervollständigen Sie diese Klasse.

Manchmal möchte man aus einer Zahlenfolge eine neue erzeugen, bei der die ersten n Folgeglieder gelöscht sind, also eine Art *Shift*-Operation. Vervollständigen Sie hierfür die Klasse *Drop*, mit der diese Operation realisiert werden kann. Für die Folge $0, 1, 2, 3, ...$ und $n = 1$ ergibt sich dann die Folge $1, 2, 3, ...$

```
public
class Drop implements Sequence {
    private Sequence s;

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

    public Drop(int n1, Sequence s1) {
        s = s1;

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

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

    public double next() {
        return
            s.next();
    }
    public Duplicate dup() {

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

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

Zahlenfolgen können durch Kombination zweier anderer Zahlenfolgen konstruiert werden, zum Beispiel durch paarweises Aufsummieren oder Subtrahieren der Folgeglieder. Vervollständigen hierzu Sie die Klasse *Sub* zum Subtrahieren zweier Folgen. Für die Folgen $1, 4, 9, 16, ...$ und $0, 1, 4, 9, ...$ ergibt sich die Resultatfolge $1, 3, 5, 7, ...$

```
public class Sub implements Sequence {
    private final Sequence s1, s2;

    public Sub(Sequence s1, Sequence s2) {
        this.s1 = s1;
        this.s2 = s2;
    }
    public double next() {

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

        ..........................................
    }
    public Duplicate dup() {

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

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

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

Im folgenden wird davon ausgegeangen, dass es für alle vier Grundrechenarten die analogen Klassen zu *Sub* gibt (*Add*, *Mult*, *Sub*, *Divide*).

Eine Klasse, die wenig sinnvoll erscheint, ist die folgende:

```
public class Id implements Sequence {
    protected Sequence s;

    public Id(Sequence s1) {
        s = s1;
    }
    public double next() {
        return
            s.next();
    }
    public Duplicate dup() {
        return
            s.dup();
    }
}
```

Gibt es außer Zeilenschinderei und Rechenzeitverschwendung noch einen Grund, diese Klasse zu realisieren?

ja ☐ nein ☐

Begründung:

. . . . . .

. . . . . .

In der Mathematik wird, um das Wachstum einer Folge zu untersuchen, häufig die Differenzenfolge gebildet. Im Beispiel über die *Sub*-Klasse ist so eine Folge berechnet worden.

Vervollständigen Sie die Klasse *Diff*, mit der zu einer gegebenen Folge die Differenzenfolge berechnet werden kann.

public
class Diff

```
    ..............................................
{

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

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

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

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

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

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

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

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

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

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

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

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

Die inverse Operation zur Bildung der Differenzenfolge ist das Aufsummieren einer Folge, zu einer Folge also die zugehörige Reihe zu berechnen. Zu einer Folge $1, 2, 4, 7, ...$ wird das Resultat $0, 1, 3, 7, 14, ...$ geliefert. Vervollständigen Sie auch diese Klasse *Sum*:

```
public class Sum implements Sequence {
    private Sequence s;

        ..........

    public Sum(Sequence s) {
        this.s = s;

        ..........

        ..........
    }
    public double next() {

        ..........

        ..........

        ..........
    }
    public Duplicate dup() {

        ..........

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

Als einfache Anwendung dieses Baukastens konstruieren Sie bitte eine Klasse *AritmMean*, die alleine durch die Nutzung der vorhandenen Klassen und ohne Neuimplementierung von *next* realisiert ist, und mit der die Folge der Mittelwerte der ersten n Folgeglieder eine Folge berechnet werden kann. Wenn die Ausgangszahlenfolge mit den Werten $1.0, 2.0, 3.0, 4.0, 10.0, ...$ beginnt, so soll die Folge $1.0, 1.5, 2.0, 2.5, 4.0, ...$ entstehen.

public class ArithmMean

. . . . . .
{

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .
}
