Klausur: Programmierkurs 1, Java, XML

A1.01, B6

26. Juli 2002

Name:		Vorname:	
Matrikelnummer:	Studiengang:	Lfd.Nr.:	Sitz-Nr.:

Hinweise zur Bearbeitung

a. Die Klausur besteht aus **5 Aufgaben**, für die insgesamt **66 Punkte** vergeben werden. Die Klausur gilt als bestanden, wenn davon **mindestens 33 Punkte** (d.h. 50 %) erreicht werden.

b. Eigenes Papier und sonstige Hilfsmittel (z.B. Taschenrechner, Palm, Notebook, Handy) sind nicht zugelassen. Die Rückseiten der Klausurbögen dürfen verwendet werden.

c. Alle Lösungswege und Rechengänge sind grundsätzlich zu dokumentieren. **Lösungen ohne Lösungsweg werden nicht gewertet.**

Viel Erfolg !

Unterschrift des Kandidaten:

Bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	Summe
Punkte						

Bestanden	Korrektor
ja | nein	

Aufgabe 1 (12 Punkte)

a. Schreiben Sie eine HTML-Tabelle (keine vollständige Seite) entsprechend den folgenden Vorgaben:

- Der Inhalt soll in zwei Zeilen und drei Spalten angeordnet sein und folgendes enthalten:

1	3	5
2	4	6

- Die Tabellenzellen sollen mit einem Rand versehen sein.
- Die Tabellenunterschrift soll '`Tabelle der 1. Aufgabe`' sein.

(6 Punkte)

b. Schreiben Sie die HTML-Tabelle XML-konform um, d.h., schreiben Sie sie als XHTML-Tabelle (wenn Sie es noch nicht getan haben). (2 Punkte)

c. Geben Sie CSS-Regeln für folgende Elemente und Darstellungen an:

- Ein `strong`-Element innerhalb von `blockquote` soll mit blauer, fetter Schrift dargestellt werden.
- Ein `strong` innerhalb von `em` soll mit kursiver, gelber Schrift dargestellt werden.

(4 Punkte)

Aufgabe 2 (10 Punkte)

a. Was versteht man unter dem Überladen von Methoden? (3 Punkte)

b. Was versteht man unter dem Überschreiben von Variablen oder Methoden? (3 Punkte)

c. Was ist der Inhalt der Zeichenkette `s` nach Ausführung des folgenden Programmfragments (bitte markieren) (2 Punkte)

```
int a = 7;
String s = a.toString();
```

- `7`,
- `a`,
- kein Inhalt, Kompilationsfehler.

d. Was ist der Inhalt der Zeichenkette `s` nach Ausführung des folgenden Programmfragments (bitte markieren) (2 Punkte)

```
Object b = new Long(7);
String s = b.toString();
```

- `7`,
- `b`,
- kein Inhalt, Laufzeit-Fehler.

Aufgabe 3 (14 Punkte)

a. Implementieren Sie den Konstruktor der Klasse `Summe`, so dass er die angegebenen Zahlen in dem Feld `betrag` speichert. Stellen Sie sicher, dass der Konstruktor auch ohne Angabe von Zahlen auf der Kommandozeile richtig funktioniert. (6 Punkte)

b. Implementieren Sie die Methode `toString`, so dass die Ausgabe von `main` wie unten ist. (4 Punkte)

c. Implementieren Sie die Methode `summe` zur Berechung der Summe aller Zahlen in `betrag` und der Rückgabe dieser Summe. (4 Punkte)

Die folgende Klasse `Summe` dient zur Berechung der Summe von beliebig vielen Zahlen, die auf der Kommandozeile angegeben werden.

```
public class Summe {

    private double[] betrag = null;

    public Summe(String[] args) {









    }

    public double summe() {











    }
```

```
    public String toString() {









    }

    public static void main(String[] args) {
        Summe alles = new Summe(args);
        System.out.print("Eingegeben wurde:\n" + alles);
        double s = alles.summe();
        System.out.println("Die Summe ist " + s);
    }
}
```

Die Ausgabe von `main` ist zum Beispiel wie folgt:

```
java Summe 3.3 -3.3 0.1 0.2 0.3 0.4
Eingegeben wurde:
Betrag 0 = 3.3
Betrag 1 = -3.3
Betrag 2 = 0.1
Betrag 3 = 0.2
Betrag 4 = 0.3
Betrag 5 = 0.4
Die Summe ist 1.0
```

Aufgabe 4 (16 Punkte)

a. Implementieren Sie die Methode `toString` der Klasse `Counter` mit Hilfe eines Iterators über die Schlüssel. (8 Punkte)

b. Implementieren Sie ein Programm `WordCount` mit Hilfe der Klasse `Counter` das zählt, wie häufig einzelne Worte in einer Datei vorkommen. (8 Punkte)

Die folgende Klasse `Counter` speichert wie in der letzten Klausur Zählerstande für verschiedene Zähler.

```
import java.util.Hashtable;
import java.util.Iterator;
public class Counter {
    private Hashtable db = null;
    public Counter() { db = new Hashtable();  }
    public long count(String key) {
        long cnt = 0;
        Object stand = db.get(key);
        if ( stand == null ) stand = new Long(cnt);
        if ( stand instanceof Long ) {
            cnt = ((Long)stand).longValue();
        }
        cnt++;
        db.put(key, new Long(cnt) );
        return cnt;
    }
    public String toString() {




    }
}
```

Das folgende Hauptprogramm

```
public static void main(String[] args) {
    Screen out = new Screen();
    String de = "Deutschland";
    String fr = "Frankreich";
    String gb = "Gross-Britanien";

    Counter zaehler = new Counter();
    zaehler.count(gb);  zaehler.count(de);
    zaehler.count(de);  zaehler.count(fr);
    out.println("Inhalt von zaehler: \n" + zaehler);
}
```

soll diese Ausgabe erzeugen:

```
Inhalt von zaehler:
Deutschland = 2
Gross-Britanien = 1
Frankreich = 1
```

Sie können zum Beispiel die Methoden `hasNext` und `next` von `Iterator`, die Methode `iterator` von `Set` und z.B. die Methode `keySet` von `Hashtable` verwenden.

```
//  Returns a Set view of the keys contained in this Hashtable.
    public Set keySet()

//  Returns an iterator over the elements in this set.
    Iterator iterator()

//  Returns true if the iteration has more elements.
    boolean hasNext()

//  Returns the next element in the iteration.
    Object next()
```

Der Rahmen für das Programm `WordCount`:

```
import java.io.BufferedReader;
import java.util.StringTokenizer;

public class WordCount {

    public static void main(String[] args) {
        Screen sc = new Screen();
        BufferedReader rein = null;
        if ( args.length > 0 ) {
           rein = new DateiEin(args[0]);
        }
        if ( rein == null ) rein = new KeyBoard();

        Counter zaehler = new Counter();
        StringTokenizer words = null;








































        sc.println(""+ein+" Zeilen gelesen");
        sc.println("word count:\n" + zaehler);
    }
}
```

Verwenden Sie zum Beispiel die Methoden `hasMoreTokens` und `nextToken` von `StringTokenizer`, um die Eingabezeilen in einzelne Worte aufzuspalten, die durch Leerzeichen getrennt sind.

```
//   Constructs a string tokenizer for the specified string. The
     public StringTokenizer(String str)

//   Tests if there are more tokens available from the string.
     public boolean hasMoreTokens()

//   Returns the next token from this string tokenizer.
     public String nextToken()
```

Die Ausgabe des Programms soll bei einem Dateiinhalt von "`rot blau gelb blau gelb gelb rot`" etwa wie folgt aussehen.

```
1 Zeilen gelesen
word count:
gelb = 3
blau = 2
rot = 2
```

Aufgabe 5 (14 Punkte)

Gegeben sei folgendes XML-Dokument:

```
<?xml version="1.0" ?>
<!DOCTYPE Sequences SYSTEM "sequences.dtd" >

<Sequences>
 <Sequence id="MIVN83300" ic-acckey="U83300"
           title="MIVN83300" length="946"
           comment="Veniliornis nigriceps ..."
           topology="linear" molecule="dna" >
   <Attribute name="version" content="U83300.1"/>
   <Attribute name="organism-species" content="woodpecker"/>
   <Attribute name="date-created" content="14-MAY-1997"/>
   <Attribute name="date-last-updated" content="4-MAR-2000"/>
   <Attribute name="database-xref" content="SPTRAFW:O03345"/>
   <Seq-data>
   aantttggatctctcctaggcatttgcctaataacacaaattgtcacagg
   ...
   </Seq-data>
</Sequence>
</Sequences>
```

Entwickeln Sie eine geeignete XML-DTD `sequences.dtd` für dieses Dokument, so dass es gültig (valid) bezüglich dieser DTD ist. Verlangen Sie, dass die Werte des `id`-Attributs von `Sequence` eindeutig sind, und das `molecule`-Attribut nur die Werte `dna` und `rna` annehmen kann. (14 Punkte)