Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

26. Januar 2001

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im WS 200/01 (WI h103, II h105, MI h353)

Zeit: 150 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 17 Seiten

---

Aufgabe 1:

Die folgenden Klassen, **XDoc** und deren Unterklassen, dienen zur Implementierung einer Baumstruktur für einfache XML–ähnliche Strukturen.

Es gibt in der XML–Struktur einfache Objekten, dieses sind das leere Objekt und das Textobjekt. Als zusammengesetzte Objekte gibt es Tags und Sequenzen.

Es gibt in den Klassen keine Methoden, die den Zustand der Objekte nach deren Erzeugung verändern. Dieses ist wegen der **final**–Attribute an den Datenfeldern auch gar nicht zugelassen.

Die Verarbeitung von solchen zusammengesetzten Objekten geschieht immer durch Traversierung (Durchlaufen) der Struktur und Verarbeitung der Knoten. Dieses sich immer wiederholende Schema ist hier in eine extra Klasse **XCmd** ausgelagert worden.

Unterschiedliche Verarbeitungsarten werden dann durch Unterklassenbildung von bf XCmd realisiert.

Die **XDoc**–Klasse und deren Unterklassen können als universelle Datenstruktur verwendet werden: beliebige Zeichenketten (Strings) können z.B. durch ein **XText**–Objekt repräsentiert werden.

Aus diesem Grunde liefern die Verarbeitungsfunktionen aus **XCmd** alle als Resultat ein **XDoc**–Objekt zurück.

In der **XDoc**–Klasse gibt es eine Vergleichsfunktion **isEqual**. Diese soll zwei beliebige Objekte überprüfen, ob diese die gleiche Struktur besitzen und ob an den Knoten die gleichen Strings gespeichert sind. Versuchen Sie, die Methodenrümpfe der Vergleichsfunktionen so zu schreiben, dass der Test für mehrfach genutzte Teilstrukturen effizient arbeitet, und das möglichst wenige String-Vergleiche ausgeführt werden.

Die **toString**–Methoden sind hier nur zum Verständnis der Funktionsweise und zu Testzwecken implementiert. Diese dürfen bei der Implementierung der Methoden nicht verwendet werden.

Teile der Implementierung sind vorgegeben, füllen sie die fehlenden Methodenrümpfe so, dass die oben geforderte Funktionsweise sichergestellt ist.

Die abstrakte Klasse für die XML-Dokumente und ihrer Ausprägungen:

```
abstract public class XDoc {

    public abstract XDoc process(XCmd c);

    public abstract String toString();

    public abstract boolean isEqual(XDoc d2);

} // end XDoc

class XEmpty extends XDoc {

    public static final XDoc xempty = new XEmpty();

    private XEmpty() {}

    public String toString() {
        return
            "";
    }

    public XDoc process(XCmd c) {
        return
            c.processXEmpty(this);
    }

    public boolean isEqual(XDoc d2) {

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

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

}
```

```
public final class XText extends XDoc {
    final String text;

    public XText(String text) {
        this.text = text;
    }

    public String toString() {
        return
            text;
    }

    public XDoc process(XCmd c) {
        return
            c.processXText(this);
    }

    public boolean isEqual(XDoc d2) {

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........
    }

} // end XText
```

```
public final class XTag extends XDoc{
    final String tag;
    final XDoc body;

    public XTag(String tag,
                XDoc body) {
        this.tag = tag;
        this.body = body;
    }

    public String toString() {
        return
            "<" + tag + ">"
            + body.toString()
            + "</" + tag + ">";
    }

    public XDoc process(XCmd c) {
        return
            c.processXTag(this);
    }

    public boolean isEqual(XDoc d2) {

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

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

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

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

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

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

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

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

} // end XTag
```

```
public final class XSeq extends XDoc {
    final XDoc first;
    final XDoc rest;

    public XSeq(XDoc first,
                XDoc rest) {
        this.first = first;
        this.rest = rest;
    }

    public String toString() {
        return
            first.toString()
            + rest.toString();
    }

    public XDoc process(XCmd c) {
        return
            c.processXSeq(this);
    }

    public boolean isEqual(XDoc d2) {

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

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

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

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

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

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

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

        ..........................
    }
} // end XSeq
```

Die abstrakte Klasse für die Verarbeitung der verschiedenen Knoten:

```
public abstract class XCmd {

    public abstract XDoc processXEmpty(XEmpty x);

    public abstract XDoc processXText(XText x);

    public abstract XDoc processXTag(XTag x);

    public abstract XDoc processXSeq(XSeq x);

} // end XCmd
```

Eine Klasse zum extrahieren aller Textelemente aus einem Dokument. Alle Tags sollen aus einem Dokument gelöscht werden. Wenn das so entstandene Dokument mit toString in einen Text verwandelt wird, sollen keine Tags mehr auftreten.

```
public class RemoveTagsCmd extends XCmd {

    public XDoc processXEmpty(XEmpty x) {

        ..........

        ..........
    }

    public XDoc processXText(XText x) {

        ..........

        ..........

        ..........
    }

    public XDoc processXTag(XTag x) {

        ..........

        ..........

        ..........
    }

    public XDoc processXSeq(XSeq x) {

        ..........

        ..........

        ..........

        ..........
    }
} // end XCmd
```

Eine Klasse zum extrahieren aller Tags aus einem Dokument. Alle Textelemente sollen aus einem Dokument gelöscht werden. Wenn das so entstandene Dokument mit toString in einen Text verwandelt wird, sollen nur noch die Tags auftreten.

```
public class RemoveTextCmd extends XCmd {

    public XDoc processXEmpty(XEmpty x) {

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

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

    public XDoc processXText(XText x) {

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

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

    public XDoc processXTag(XTag x) {

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

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

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

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

    public XDoc processXSeq(XSeq x) {

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

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

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

        ..........................
    }
} // end XCmd
```

Eine Klasse zum Transformieren des gesammten Objekts in ein einziges **XText**–Objekts, dass die String-Repräsentation des Objekts enthält, d.h. das in dem **XText**–Objekt der Text gespeichert ist, den man mit **toString** berechnen würde. Diese Klasse kann also zum Ersetzen der **toString**–Methoden dienen.

```
public class ToStringCmd extends XCmd {

    public XDoc processXEmpty(XEmpty x) {

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

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

    public XDoc processXText(XText x) {

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

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

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

    public XDoc processXTag(XTag x) {

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

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

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

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

    public XDoc processXSeq(XSeq x) {

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

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

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

        ..........................
    }
} // end XCmd
```

Eine einfache Klasse zur Verarbeitung ist die folgende:

```
public class IdentCmd extends XCmd {

    public XDoc processXEmpty(XEmpty x) {
        return
            x;
    }

    public XDoc processXText(XText x) {
        return
            x;
    }

    public XDoc processXTag(XTag x) {
        return
            new XTag(x.tag,
                    x.body.process(this));
    }

    public XDoc processXSeq(XSeq x) {
        return
            new XSeq(x.first.process(this),
                    x.rest.process(this));

    }
} // end XCmd
```

Hat diese Klasse irgendeinen Nutzen?

ja ☐ nein ☐

Begründung:

. . . . . .

. . . . . .

. . . . . .

Ein kleines Testprogramm (nur als Verständnishilfe):

```
public class Test {
    public static void main(String[] argv) {
        XDoc d = new XSeq(new XTag("head",
                                   XEmpty.xempty
                                   ),
                          new XTag("body",
                                   new XText("hello world")));

        String t = d.toString();

        // t = "<head></head><body>hello world</body>"

        XDoc d1 = d.process(new ToStringCmd());
        String t1 = ((XText)d1).text;

        // t1 = "<head></head><body>hello world</body>"

        XDoc d2 = d.process(new RemoveTagsCmd());
        String t2 = d2.toString();

        // t2 = "hello world"

        XDoc d3 = d.process(new RemoveTextCmd());
        String t3 = d3.toString();

        // t3 = "<head></head><body></body>"

        XDoc d4 = d.process(new IdentCmd());
        String t4 = d4.toString();

        // t4 = "<head></head><body>hello world</body>"

}
```

Diese Datenstruktur ist so entwickelt, dass Objekte nie nach ihrer Erzeugung verändert werden. Welche Vorteile besitzt dieser Ansatz gerade in Java gegenüber einem Ansatz, bei dem Datenfelder verändert werden.

1) .....

2) .....

3) .....

Welche Nachteile besitzt dieser Ansatz gegenüber einem, bei dem Datenfelder verändert werden.

1) .....

2) .....

3) .....

Ist sichergestellt, dass diese Eigenschaft, dass keine Objekte verändert werden, auch bei Erweiterung der Klassenhierarchie um neue Klassen immer erhalten bleibt?

ja ☐ nein ☐

Begründung:

.....

.....

Müssen Klassen modifiziert werden, wenn **XDoc** von weiteren Klassen beerbt wird?

ja ☐ nein ☐

Begründung:

.....

.....

Aufgabe 2:

Gegeben seien die folgenden Variablen:

```
int x;
unsigned int u;
long int s;
float f;
char *p1;
long int *p2;
void *p3;
```

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Vorsicht: Es kommen fehlerhafte und logisch falsche Ausdrücke von. Kennzeichnen Sie diese entsprechend

p2[*p2] .....

s || x .....

p1 ? x : f .....

++f .....

p1 == p3 .....

p1 = p3 .....

1 + ~p2 .....

p1 && p1 .....

x += f .....

! p3 .....

~s .....

! s .....

p3 ? x : f .....

p1 + 0x23 .....

s | x .....

Aufgabe 3:

Gegeben sei das folgende C–Programm zur Verarbeitung von Mengen als Bitstrings.

```
#include <stdio.h>

typedef unsigned char Menge;
#define MengeMax 8

void printMenge(Menge s) {
    unsigned int i = MengeMax;
    while ( i-- != 0 )
        printf("%1u", (unsigned int)((s >> i) & 1));
}

static unsigned int linecnt = 0;

#define PRINT(s) { printf("%2u) ", ++linecnt); printMenge(s); printf("\n"); }

#define einStueck(n,m) (dieErsten(m+1) ^ dieErsten(n))
#define dieErsten(n) (einElement(n) - 1)
#define einElement(i) ( (Menge)(1 << (i)) )

int main(void) {
    Menge s1;

    PRINT( einElement(1) );
    PRINT( einElement(MengeMax) );

    PRINT( (Menge)1 );
    PRINT( (Menge)14 );

    PRINT( einStueck(4,4) );
    PRINT( einStueck(2,1) );
    PRINT( einStueck(0,MengeMax-1) );

    PRINT( 021 & 020 );
    PRINT( 021 && 020 );

    s1 = einStueck(0,3) | ~einStueck(1,6); PRINT(s1);
    s1 = einStueck(1,4) ^ (Menge)((128 - 1) * 4); PRINT(s1);

    s1 = 8 + 48;
    s1 = s1 ^ (s1 & (~s1 + 1)); PRINT(s1);

    return 0;
}
```

Die Mengen sind in diesem Beispiel 8 Bits lang, können also die Elemente $0, 1, \ldots, 7$ enthalten. *printSet* gibt eine Menge im Binärformat aus. Die Menge, die nur die 1 enthält würde als 00000010 ausgegeben werden. Das *PRINT* Makro gibt jeweils eine Menge pro Zeile aus und numeriert die Zeilen durch.

Welche 12 Ausgabezeilen erzeugt dieses Programm?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....
