Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

31. Januar 2003

---

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im WS 2002/03 (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:**

Gegeben sei die folgende Java-Klasse.

```
public class Buffer {
    private boolean empty = true;
    private Data value = null;

    public void put(Data d) {
        value = d;
        empty = false;
    }

    public Data get() {
        Data d = value;

        value = null;
        empty = true;

        return d;
    }
}
```

Diese Klasse implementiert einen Puffer für ein Exemplar aus der Klasse *Data*. Es soll dabei sicher gestellt sein, dass der Puffer entweder leer ist, angezeigt durch die Variable *empty*, oder voll, also eine Referenz auf ein *Data*-Objekt enthält. Diese Eigenschaft wird in der Variablen *empty* gespeichert.

Diese Klasse ist nicht *Thread*-sicher. Außerdem wird nicht sichergestellt, dass die *put*- und *get*-Operationen immer genau wechselseitig aufgerufen werden, so dass alle mit *put* geschriebenen Daten auch genau einmal mit *get* gelesen werden.

Erweitern Sie die *get*- und *put*-Methoden so, dass diese *Thread*-sicher sind und dass die zusätzlichen Bedinungen für den Einsatz in einem Erzeuger-Verbraucher-Muster für die *Buffer*-Klasse erfüllt sind.

Hinweis: in Java gibt es die Methoden *wait()* und *notify()*. *wait()* kann eine überprüfte Ausnahme *InterruptedException* auslösen.

Die modifizierte *put()*-Methode:

Die modifizierte *get()*-Methode:

**Aufgabe 2:**

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 $X$:

Der übrige Programmcode:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

**Aufgabe 3:**

Die folgenden Klasse **NTree** und einige Hilfsklassen und Schnittstellen dienen zur Implementierung von beliebigstelligen Bäumen.

In diesen Klassen sind einige Methodenrümpfe zu entwickeln, und zwar zur einheitlichen Verarbeitung aller Knoten eines Baumes.

Die Methode **toString** dient hier zur Testausgabe. Sie ist bei der Entwicklung der anderen Methoden nicht zu verwenden.

Die Methode **noOfElements** soll die Anzahl der Knoten in einem Baum berechnen.

Die Methode **map** soll aus einem Baum durch Anwenden einer Funktion von der Art **Function** auf alle Knoten einen neuen Baum berechnen. Es soll hierbei der Knoten selbst als erstes verarbeitet werden, anschließend die Kinder in der Reihenfolge, in der sie im Feld abgespeichert sind.

Die Methode **copy** soll für eine komplette Baumstruktur eine physikalische Kopie anlegen, es sollen also alle Exemplare von **NTree** dupliziert werden. Die Knoteninformation selbst soll unverändert bleiben.

Tipp: Bitte lesen Sie alle Programmteile einschließlich des Testprogramms sorgfältig durch, bevor Sie mit der Bearbeitung der Aufgabe beginnen.

Tipp: Bitte vermeiden Sie die Verdopplung von Algorithmen(–teilen).

Die Schnittstelle für die Representation von einstelligen Funktionen:

```
interface Function {
    public Object apply(Object o);
}
```

**Die Klasse für die Baumstruktur:**

```
public class NTree {
    private Object node;
    private NTree [] children;

    private NTree(Object n,
                  NTree [] cs) {
        node = n;
        children = cs;
    }
```

NTree (cont.): Die erzeugenden Funktionen und die Testausgabe

```
    private static final NTree [] ecs = new NTree[0];

    public static NTree mkLeave(Object n) {
        return
            mkTree(n, ecs);
    }

    public static NTree mkTree(Object n,
                               NTree [] cs) {

        return
            new NTree(n, cs);
    }

    public static NTree mkTree1(Object n,
                                NTree c) {
        return
            mkTree(n, new NTree []{c});
    }

    public static NTree mkTree2(Object n,
                                NTree c1,
                                NTree c2) {
        return
            mkTree(n, new NTree []{c1, c2});
    }

    public String toString() {
        String cs = "";
        for (int i = 0; i < children.length; ++i) {
            cs += children[i].toString();
            if (i < children.length -1) {
                cs += ",";
            }
        }
        return
            node.toString() + "(" + cs + ")";
    }
```

NTree (cont.): Die Methode zur Berechnung der Anzahl der Knoten

```
public int noOfNodes() {

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

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

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

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

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

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

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

NTree (cont.): Die Verarbeitung aller Knoten

```
public NTree map(Function f) {
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
    ..........................
}
```

NTree (cont.): Das Klonen

```
    public NTree copy() {
        ..........
        ..........
        ..........
        ..........
        ..........
        ..........
        ..........
        ..........
        ..........
        ..........
    }

}
```

Ende von **NTree**

**Eine Klasse für eine Funktion zur Berechnung der Anzahl Zeichen eines String–Objekts.**

Hier soll angenommen werden, dass die Klasse von Anwendern immer *vernünftig* verwendet wird.

```
public class StringLengthFunction
    implements Function {

    public Object apply(Object o) {

        ..........

        ..........

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

**Eine Klasse zum Durchnummerieren aller Knoten eines Baumes.**

Hierbei soll jedes Objekt auf eine Zahl abgebildet werden, und zwar auf die Stelle an der das Objekt beim Durchlauf verabeitet wird. Es soll beim Zählen mit 1 begonnen werden, der Wurzelknoten wird also auf die 1 abgebildet.
Tipp: Bitte vorher das Testprogramm genau durcharbeiten.

```
public class NumberNodesFunction
    implements Function {

        ..........

        ..........
    public Object apply(Object o) {

        ..........

        ..........

        ..........

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

**Ein einfaches Testprogramm:**

```
public class Test {
    public static void main(String [] argv) {
        NTree t1 =
            NTree.mkTree2("tooMuch",
                          NTree.mkTree1("ham",
                                        NTree.mkLeave("and")),
                          NTree.mkLeave("eggs")
                          );

        System.out.println(t1.noOfNodes());

        System.out.println(t1.copy());

        System.out.println(t1.map(new StringLengthFunction()));

        System.out.println(t1.map(new NumberNodesFunction()));

    }
}
```

Welche 4 Zeilen gibt dieses Testprogramm aus?

1) .....

2) .....

3) .....

4) .....

**Fragen:**

1. Ist es sinvoll, den Konstruktor für **NTree** private zu deklarieren?

   ja ☐ nein ☐

   Begründung:

   .....

2. Ist es sinnvoll, dass die Funktionen **mkTree1** und **mkTree2** ihre Arbeit an **mkTree** delegieren, und nicht direkt den Konstruktor aufrufen?

   ja ☐ nein ☐

   Begründung:

   .....

3. Ist es sinnvoll, die Konstante **ecs** zu verwenden, obwohl sie nur an einer Stelle verwendet wird?

   ja ☐ nein ☐

   Begründung:

   .....

4. Ist die **copy**–Methode für die Verwendung der Datenstruktur notwendig, wenn nur mit den gegebenen Methoden gearbeitet wird?

   ja ☐ nein ☐

   Begründung:

   .....

5. Ist es sinnvoll, dass die Funktionen **mkLeave** ihre Arbeit an **mkTree** delegiert, und nicht direkt den Konstruktor aufruft?

   ja ☐ nein ☐

   Begründung:

   .....

---

**Aufgabe 4:**

Gegeben ist das folgende C Programmstück. Dieses soll in eine header Datei *p.h* und eine Implementierungsdatei *p.c* aufgeteilt werden, so daß es von mehreren anderen Modulen verwendet werden kann. Kennzeichnen Sie durch Ankreuzen des *.h* Feldes für die header Datei oder *.c* für die Implementierungsdatei, in welche Datei die einzelnen Codestücke übertragen werden müssen.

```
static int nochEineZahl;                                    .h  .c

#include <stdio.h>                                          .h  .c

typedef unsigned long Myint;                                .h  .c

const Myint maximum = 8888;                                 .h  .c

extern Myint eineZahl;                                      .h  .c

typedef struct X * Px;                                      .h  .c

Myint eineZahl = 42;                                        .h  .c

static int nochEineZahl = 4711;                             .h  .c

static int f (Px x1);                                       .h  .c

int g (Px x1);                                              .h  .c

struct X {                                                  .h  .c
  Myint d;
} X;

Myint read (struct X);                                      .h  .c

Myint read (struct X t) {                                   .h  .c
  return t.d + nochEineZahl;
}

#define minimum 1111                                        .h  .c

int check (Px t) {                                          .h  .c
  return t→d ≥ minimum;
}

void outX (FILE * out, Px t);                               .h  .c
```

```
void outX (FILE * out, Px t) {
  fprintf(out, "%lu\n", t→d);
}
```
.h .c

```
extern Myint einPaarZahlen[];
```
.h .c

```
int g (Px x1) {
  return f(x1) + 2;
}
```
.h .c

```
static int f (Px x1) {
  return read (*x1);
}
```
.h .c

```
#define init_X(x) ((x).d = 0)
```
.h .c

```
struct s { int a[minimum]; };
```
.h .c

```
extern struct s einStruct;
```
.h .c

---