Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

1. Februar 2000

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im WS 99/00 (WI h103, II h105, MI h353)

Zeit: 120 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 16 Seiten

---

Aufgabe 1:

Die folgenden Klassen dienen zur Implementierung einer allgemeinen dynamischen Datenstruktur für Listen und Bäume. Die Bezeichnungen sind an die in LISP üblichen Bezeichnungen angelehnt. In dieser einfachsten universellen Datenstruktur gibt es drei Ausprägungen:

Atome sind die elemetaren Objekte, diese werden durch einen Namen identifiziert.

Ein spezielles elementares (atomares) Objekt wird mit nil bezeichnet und wird z.B. für die Darstellung einer leeren Liste verwendet.

Zusammengesetzte Objekte werden mit Hilfe von Paaren aufgebaut, wobei die beiden Teile wieder beliebig komplexe Werte sein dürfen.

Listen werden üblicherweise als eine Folge von Paaren dargestellt, wobei die Elemente der Liste über den cdr-Verweis verkettet werden und das Ende der Liste durch eine Referenz auf das nil-Objekt gekennzeichnet wird.

Mit Prädikaten kann man Eigenschaften von einem Wert erfragen:

- isAtom soll gelten für nicht zusammengesetzte Objekte
- isNil nur für den speziellen nil–Wert
- isPair nur für zusammengesetzte Werte
- isList soll gelten für die leere Liste, repäsentiert durch nil, und für Paare, deren zweiter Teil (cdr) eine Liste ist
- isEqual soll zwei beliebige Werte auf Gleichheit überprüfen. Zu implementieren ist dieser Test mit möglichst wenigen Vergleichsoperationen.

Die append-Methode soll eine neue Liste aufbauen, bei der der Wert am Ende der Liste steht, also über die Referenz in car aus dem letzten Paar zugreifbar ist.

Die length Methode berechnet die Anzahl der Paare, die über die cdr-Referenzen verkettet sind.

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

Die toString–Methoden sind als reine Testmethoden zu behandeln, keine der anderen Methoden sollte diese in ihrer Implementierung nutzen.

```
public
    abstract
    class Value {

        //---------
        // Prädikate

        public
            boolean isAtom() {
            return
                false;
        }
        public
            boolean isNil() {
            return
                false;
        }
        public
            boolean isPair() {
            return
                false;
        }
        public
            boolean isList() {
            return
                false;
        }
        public
            boolean isEqual(Value v2) {
            return
                false;
        }

        //---------
        // Selektoren

        public
            Value car() {
            throw
                new RuntimeException("car not supported");
        }

        public
            Value cdr() {
            throw
                new RuntimeException("cdr not supported");
        }
```

```
//---------

public
    Value append(Value v2) {
    return
        new Pair(v2,this);
}


public
    int length() {
    return
        0;
}

//---------

public static final
    Value nil = new Nil();

public static
    Value atom(String name) {
    return
        new Atom(name);
}

public static
    Value pair(Value car, Value cdr) {
    return
        new Pair(car,cdr);
}
```

```
//innere Klasse Nil

private static final
    class Nil extends Value {

        public
            boolean isAtom() {

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

                ..........................
        }
        public
            boolean isNil() {

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

                ..........................
        }
        public
            boolean isList() {

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

                ..........................
        }
        public
            boolean isEqual(Value v2) {

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

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

        public
            String toString() {
            return
                "nil";
        }
    }

// Ende Klasse Nil
```

```
// innere Klasse Atom

private static final
    class Atom extends Value {

        final
            String name;

        Atom(String name) {
            this.name = name;
        }

        public
            boolean isAtom() {

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

        public
            boolean isEqual(Value v2) {

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

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

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

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

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

        public
            String toString() {
            return
                name;
        }
    }

// Ende Klasse Atom
```

```
// innere Klasse Pair

private static final
    class Pair extends Value {

        final
            Value car, cdr;

        Pair(Value car, Value cdr) {
            this.car = car;
            this.cdr = cdr;
        }

        public
            boolean isPair() {

             ..........................
        }
        public
            boolean isList() {

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

        public
            Value car() {
            return
                car;
        }

        public
            Value cdr() {
            return
                cdr;

        public
            String toString() {
            return
                "( " + car.toString() + "  .  " + cdr.toString() + " )";
        }
```

```
        public
            boolean isEqual(Value v2) {

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

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

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

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

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

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

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

            public
                Value append(Value v2) {

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

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

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

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

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

            public
                int length() {

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

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

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

                ..........................
            }
        }
    // Ende Klasse Pair
}
```

Fragen:

1. Ist das static Attribut vor den Klassen Nil, Atom und Pair wichtig?

   ja ☐ nein ☐

   Begründung:

   .....

2. Würden diese Klassen auch noch korrekt arbeiten, wenn man das static Attribut weglässt?

   ja ☐ nein ☐

   Begründung:

   .....

3. Hätte das fehlende static Attribut Einfluss auf die Größe der Objekte?

   ja ☐ nein ☐

   Begründung:

   .....

4. Ist die Funktion atom in der Klasse Value notwendig, oder ist es vernünftiger, den sehr kurzen Funktionsrumpf in die Anwendungen direkt einzukopieren.
   Notwendig?

   ja ☐ nein ☐

   Begründung:

   .....

5. Was müßte man ändern, damit man diese atom Funktion einsparen kann.

   .....

6. Ist diese Änderung aus softwaretechnischer Sicht vernünftig?

   ja ☐ nein ☐

   Begründung:

   .....

7. Wäre eine entsprechende Funktion public static $Valuenil()\{$return $nil;\}$ aus softwaretechnischen Gründen sinnvoll?

   ja ☐ nein ☐

   Begründung:

   .....

8. Ist das final Attribut für die static Variable nil wichtig?

   ja ☐ nein ☐

   Begründung:

   .....

9. Ist das final Attribut für die inneren Klassen Nil, Atom und Pair wichtig?

   ja ☐ nein ☐

   Begründung:

   .....

10. Ist es aus softwaretechnischen Gründen sinnvoll, die Klassen Nil, Atom und Pair als innere Klassen zu realisieren?

    ja ☐ nein ☐

    Begründung:

    .....

    .....

11. Wieviele Objekte von der Klasse Nil werden in einem Programm, das diese Klassen verwendet, erzeugt?

    .....

12. Mit welcher Zeitkomplexität arbeitet die append-Methode?

    .....

13. Mit welcher Speicherplatzkomplexität arbeitet die append Methode?

    .....

14. Mit welcher Speicherplatzkomplexität arbeitet die statische pair Methode aus Value?

    .....

15. Mit welcher Zeitkomplexität arbeitet die isEqual-Methode?

    .....

Aufgabe:

Die Atom Objekte werden nie verändert, so dass für Atome mit gleichem Name auch die gleichen Objekte verwendet werden können. Schreiben sie für diese Speicherplatzoptimierung die Routine atom aus der Klasse Value neu.

Aufgabe:

Gegeben sei die folgende Schnittstelle für eine Listenklasse:

```
public
    interface List {

        // hängt ein Element an die Liste vorne an.
        public
            void prepend(Value v);

        // hängt ein Element an die Liste hinten an.
        public
            void append(Value v);

        // berechnet die Länge einer Liste
        public
            int length();

        // greift auf das i-te Element der Liste
        // zu, das Element mit dem Index 0
        // ist der Kopf der Liste
        public
            Value get(int i);

        // testet ob die Liste leer ist
        public
            boolean isEmpty();
    }
```

Für diese Schittstellen ist eine Implementierung zu entwickeln, die die Value–Klasse verwendet. Der Rahmen für diese Klasse ist wieder vorgegeben. Entwickeln Sie die fehlenden Teile.

```
public
    class ListAsValueList implements List {
        private Value l;

        public ListAsValueList() {

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

        public
            void prepend(Value v) {

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

        public
            void append(Value v) {

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

        public
            int length() {

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

        public
            Value get(int i) {

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

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

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

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

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

        public
            boolean isEmpty() {

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

```
    public
        String toString() {
        String res = "";
        int len = length();

        if (len != 0) {

            res = get(0).toString();

            for (int i = 1;
                  i < length();
                  ++i) {
                res += ", " + get(i).toString();
            }
        }
        return
            res;
    }
}
```

Aufgabe 2:

Gegeben seien die folgenden Variablen und Funktionen:

```
unsigned int x;
long int s;
float f;
unsigned char *p1;
int *p2;
void *p3;
int (*pf)(void);
void (*pf2)(double);
int f1(void);
int f2(int x1,int x2);
void f3(double x1);
```

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

*(p2+x) .....

p3+x .....

pf=f1() .....

*(p3+x) .....

p1 == p3 ? f1 : pf .....

~p2 .....

! p2 .....

*p1 && p1 .....

*p1 & p1 .....

s || x .....

pf2==f3 .....

s | x .....

(*pf)(f1()) .....

pf=f1 .....

pf2=f3 .....

f=f3(f) .....

*p3 .....