

---

Aufgaben zur Klausur **Objektorientierte Programmierung** im WS 99/00 (IA 252)

Zeit: 60 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:

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 elementaren 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, repräsentiert 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 Methodenrumpfe 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:

.....