Prof. Dr. Uwe Schmidt

ptlwedel
PRIVATE BERUFSFACHSCHULE

5. Februar 2015

Aufgaben zur Klausur **Objektorientierte Programmierung** im WS 2014/15 (IA 252)

Zeit: 90 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.

Nutzen Sie die Rückseiten der Klausur zur Entwicklung der Lösungen und übertragen die fertigen Lösungen in das Aufgabenblatt.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Tipp: Bei der Entwicklung der Lösung können kleine Skizzen manchmal hilfreich sein.

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 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 eines Werts 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.

Nutzen Sie an keiner Stelle im zu entwickenden Code den instanceof-Operator.

Die abstrakte öffentliche Klasse *Value*

```
public abstract
    class Value {
    // Praedikate

    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 String name() {
        throw
            new RuntimeException("name not supported");
    }
    public Value append(Value v2) {
        return
            new Pair(v2,this);
    }
```

```
    public int length() {
        return
            0;
    }
    // intelligente Konstruktoren

    public static final
        Value nil = new Nil();

    private static
        java.util.Dictionary<String,Atom> atoms
        = new java.util.Hashtable<String,Atom>();

    public static Value atom(String name) {
        // trace output

        System.err.println("Value.atom(" + name + ")");

        Atom a = atoms.get(name);
        if (a == null) {
            // trace output

            System.err.println("new Atom(" + name + ")");

            a = new Atom(name);
            atoms.put(name, a);
        }
        return
            a;
    }
    public static Value pair(Value car, Value cdr) {
        return
            new Pair(car, cdr);
    }
}
```

Die Klasse für den Spezialwert *Nil*

```
final
    class Nil extends Value {

    public
        boolean isAtom() {

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

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

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

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

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

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

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

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

        ..........................................................
    }
    public
        String name() {
        return
            "nil";
    }
    public
        String toString() {
        return
            name();
    }
}
```

Die Klasse für die anderen Atome *Atom*

```
final
    class Atom extends Value {

    private final
        String name;

    Atom(String name) {
        this.name = name;
    }
    public
        boolean isAtom() {

            . . . . . .

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

            . . . . . .

            . . . . . .

            . . . . . .

            . . . . . .

            . . . . . .

            . . . . . .
    }
    public String name() {

            . . . . . .

            . . . . . .
    }
    public
        String toString() {
        return
            name();
    }
}
```

Die Klasse für zusammengesetzte Werte *Pair*

```
final
    class Pair extends Value {

    private 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
        int length() {

        . . . . . .

        . . . . . .

        . . . . . .
    }
```

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

    public
        Value append(Value v2) {
            ..........
            ..........
            ..........
            ..........
    }

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

Fragen:

1. 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:

   . . . . . .

2. Ist es aus Software-technischen Gründen sinnvoll in den Funktionen *car* und *cdr* mit der Klasse *RuntimeException* anstatt mit einer eigenen von *Exception* abgeleiteten Klasse zu arbeiten?

   ja ☐ nein ☐

   Begründung:

   . . . . . .

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

   ja ☐ nein ☐

   Begründung:

   . . . . . .

4. Wie viele Objekte aus der Klasse *Nil* werden in einem Programm, das diese Klassen verwendet, erzeugt?

   . . . . . .

5. Mit welcher Zeitkomplexität arbeitet die append-Methode?

   . . . . . .

6. Ist die umständliche Implementierung der statischen Methode *atom* in *Value* sinnvoll?

   . . . . . .

7. Mit welcher Zeitkomplexität arbeitet die isEqual-Methode?

   . . . . . .