Physikalisch–Technische Lehranstalt Wedel
Prof. Dr. Uwe Schmidt

8. August 2003

Aufgaben zur Klausur **Objektorientierte Programmierung** im SS 2003 (IA 252)

Zeit: 75 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 10 Seiten

---

Aufgabe 1:

Einfach verkettete Ringe werden ähnlich wie verkettete Listen implementiert. Die Zeiger (oder Referenzen) in den Datenstrukturen werden nur auf andere Art interpretiert. Es wird in den Operationen immer zwischen leeren und nicht leeren Listen unterschieden. Diese Unterscheidung wird im objektorientierten Ansatz vernünftigerweise durch Unterklassenbildung realisiert.

Für die leere Liste wird immer eine Spezialreferenz genutzt.

Eine nicht leere Liste wird durch eine Referenz auf den letzten Knoten in der Liste repräsentiert. In diesem Knoten steht nicht die Referenz auf die leere Liste als Endekennung, sondern die Referenz auf den Anfang der Liste. So wird es möglich, sowohl auf den Anfang als auch auf das Ende einer Liste in konstanter Zeit zuzugreifen. Diese Eigenschaft ist wichtig für die drei Operationen ein Element an den Anfang einer Liste anhängen (cons), ein Element an das Ende einer Liste anhängen (append) und zwei Listen konkatenieren (concat). Diese besitzen eine konstante Laufzeit.

Der indizierte Zugriff und das Einfügen und Löschen an einer bestimmten Position läuft weiterhin in einer Zeit proportional zur Größe der Position.

Das Löschen und Einfügen an einer Position können einfach realisiert werden mit Hilfe einer Operation, die eine Liste an einer bestimmten Stelle in zwei Listen aufteilt. Diese Operation (splitAt) besitzt zwei Resultatwerte. Diese werden in diesem Beispiel aus Effizienzgründen über globale Variablen realisiert. Die Listen werden immer vor dem Element geteilt, auf das der Index verweist.

Indiziert wird immer ab 0. Bei der Verwendung von illegalen Indizes sollen Ausnahmen ausgelöst werden, wie sie in einigen Implementierungen vorgegeben sind.

Die Operationen, die allgemein auf Listen zulässig sein sollen, werden mit einem Interface festgelegt.

```
interface List {

    boolean isEmpty();
    Object head();
    List tail();
    int length();
    Object at(int i);

    List cons(Object e);
    List concat(List l2);
    List append(Object e);

    List insertAt(int i, Object e);
    List removeAt(int i);
}
```

Die Implementierung, die hier entwickelt werden soll, wird mit einer Klasse **ListAsRing** und zwei geschachtelten Klassen **Empty** und **Ring** realisiert.

In diesen Java–Klassen sind einige Programmteile vorgegeben. Es sind aber Methodenrümpfe oder Teile davon offen gelassen. Füllen sie diese Stellen mit den notwendigen Codestücken. Der Platz für die fehlenden Teile ist so gewählt, dass er für die notwendigen Anweisungen ausreichend ist.

Arbeiten Sie vor dem Lösen den gesammten Programmtext durch, da in den Java–Klassen Vorwärtsreferenzen enthalten sind.

```
public abstract class ListAsRing implements List {

    // construction of the empty list

    public static List mkEmpty() {

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

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

    // construction of a single element list

    public static List mkOne(Object e) {

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

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

    // access to the 1. element of a list

    public Object head() {

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

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

```
// preprend a list with a single element

public List cons(Object e) {

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

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

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

// append a single element to a list

public List append(Object e) {

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

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

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

// insert an element into a list in front of index i

public List insertAt(int i, Object e) {
      splitAt(i);
      return

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

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

// remove element at position i from a list

public List removeAt(int i) {
      splitAt(i);
      return

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

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

```
// none public stuff

// the representation for the empty list

private static final List empty = new Empty();

// result variables for splitAt

protected static List list1,list2;

protected abstract void splitAt(int i);


// local class Empty


private static final class Empty extends ListAsRing {

    public boolean isEmpty() {
        return
            true;
    }

    public int length() {
        return 0;
    }

    public Object at(int i) {
        throw
            new IndexOutOfBoundsException("in at");
    }

    public List tail() {
        throw
            new IndexOutOfBoundsException("in tail");
    }

    public List concat(List l2) {

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

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

```
    protected void splitAt(int i) {
        if (i != 0)
            throw
                new IndexOutOfBoundsException("in splitAt");
        list1 = list2 = empty;
    }

    public String toString() {
        return
            "[]";
    }

} // end local class Empty


// local class Ring for a single ring element


private static final class Ring extends ListAsRing {

    // the data fields for info and next reference

    private Object info;
    private Ring next;

    // constrcution of a single element ring

    Ring(Object e) {
        info = e;
        next = this;
    }

    public boolean isEmpty() {

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

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

```
public int length() {
    int res;
    Ring r1;

     ..........

     ..........

    while ( .......... ) {

        ..........

        ..........
    }
    return res;
}

public Object at(int i) {
    Ring r1 = ..........;
    while ( i != 0 ) {

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........
    }
    return

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

```
public List tail() {
    // single element list

    ..........

    ..........

    ..........

    // length > 1

    ..........

    ..........

    ..........

    ..........
}

public List concat(List l2) {
    if ( .......... ) {
        // l2 empty

        ..........

        ..........
    } else {
        // l2 not empty

        ..........

        ..........

        ..........

        ..........

        ..........

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

```
protected void splitAt(int i) {
    if (i == 0) {
        // 1. result is the empty list
        list1 = ..... ;
        list2 = ..... ;
        return;
    }
    Ring r1 = ..... ;
    while (i != 1) {
        .....
        .....
        .....
        .....
    }
    if ( ..... ) {
        // 2. result is empty list
        list1 = ..... ;
        list2 = ..... ;
    } else {
        // 2 none empty lists as result
        .....
        .....
        .....
        .....
        .....
        list1 = ..... ;
        list2 = ..... ;
    }
}
```

```
        public String toString() {
            Ring r1 = next;
            String res = "[" + r1.info;

            while (r1 != this) {
                r1 = r1.next;
                res += "," + r1.info;
            }

            res += "]";
            return
                res;
        }

    } // end local class Ring

} // end class List
```
