8. Übungsblatt „Programmiersprachen“ WS 97/98

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 8. Januar 1998 **Abgabe:** 19. Januar 1998

Hinweis: Für alle implementierten Klassen soll gelten, daß ausschließlich die in den Aufgaben angegebenen Mitglieder öffentlich sind. Außer den Klassen `Enumeration` und `NoSuchElementException` dürfen keine Klassen aus dem Paket `java.util` verwendet werden.

VIII.1: Implementierung von Schlangen in Java ($4 + 6$ Punkte)

In dieser Aufgabe sollen Schlangen mit Elementen des Typs `Object` in Java implementiert werden. Die in der Schlange befindlichen Elemente werden dabei durch eine einfach verkettete Liste repräsentiert.

a. Implementieren Sie die Klasse `Cell` mit folgenden öffentlichen Methoden:

```
class Cell {
  ...
  public Cell(Object o) { ... }
  public Object getElement() { ... }
  public Cell getNext() { ... }
  public Cell append(Object o) { ... }
}
```

Aus Objekten dieser Klasse wird die Liste mit den in der Schlange befindlichen Elementen aufgebaut. Ein `Cell`-Objekt stellt ein Paar aus einem Element und einem Next-Zeiger dar; diese Komponenten können über die `get`-Methoden erfragt werden. Der Konstruktor baut eine Zelle, deren Next-Zeiger `null` enthält. Verändert wird dieser Zeiger durch die Methode `append`: Diese erzeugt eine neue Instanz von `Cell` und setzt den Next-Zeiger des aktuellen Objekts destruktiv auf dieses neue Objekt.

b. Mithilfe der Klasse aus Teilaufgabe a soll nun die Schlange implementiert werden:

```
class Queue {
  ...
  public Queue() { ... }
  public boolean isEmpty() { ... }
  public void enqueue(Object o) { ... }
  public Object dequeue() throws EmptyQueueException { ... }
}
```

Der Konstruktor erzeugt eine leere Schlange. Die Methode `isEmpty` gibt `true` zurück genau dann, wenn die Schlange leer ist. Elemente können mittels `enqueue` an das Ende der Schlange angehängt werden und über `dequeue` vom Anfang der Schlange entfernt werden; das entfernte Element wird dabei zurückgegeben. Wird `dequeue` auf eine leere Schlange angewendet, so wird die Ausnahme `EmptyQueueException` geworfen, die Sie hierzu noch definieren müssen.

VIII.2: Implementierung von Listen in Java (14 Punkte)

Schreiben Sie eine abstrakte Klasse `List` mit Unterklassen `EmptyList` und `NonEmptyList`. Objekte dieser Klassen stellen Listen mit Elementen des Typs `Object` dar; `EmptyList` entspricht dem `nil` aus SML, `NonEmptyList` dem Konstruktor ::.

```
abstract class List {
  abstract public boolean isEmpty();
  abstract public Object hd() throws EmptyListException;
  abstract public List tl() throws EmptyListException;
  abstract public boolean member(Object o);
  abstract public List append(List l);
  abstract public List reverse();
  public java.util.Enumeration elements() { ... }
  public static List mkList(Object[] os) { ... }
}
```

Alle abstrakten Methoden operieren wie die bekannten Listenoperationen; sie verhalten sich nicht destruktiv, sondern bauen gegebenenfalls neue (Teil-)Listen auf. Durch Aufruf von `elements` erhält man einen Enumerator wie in der Vorlesung vorgestellt. Die statische Methode `mkList` liefert eine Liste zurück, die alle Elemente des übergebenen Arrays in der gegebenen Reihenfolge enthält.

`EmptyList` soll nur einen nullstelligen Konstruktor besitzen, `NonEmptyList` einen Konstruktor wie folgt, der ein Element vor eine bestehende Liste hängt:

```
public NonEmptyList(Object o, List l) { ... }
```

Beide Klassen sollen zusätzlich die Methoden

```
public String toString() { ... }
public boolean equals(Object o) { ... }
```

der Klasse `Object` überschreiben. Wählen Sie für `toString` eine geeignete externe Darstellung der Listen; `equals` soll Listen strukturell vergleichen.

Bemerkung: Diese Aufgabe dient im Wesentlichen didaktischen Zwecken. Anders als in funktionalen Sprachen werden Listen in Java nur selten verwendet oder anders realisiert.

VIII.3: Ströme in Java (9 Punkte)

Implementieren Sie eine Klasse `Primes` mit folgenden öffentlichen Methoden:

```
class Primes {
  public Primes() { ... }
  public int nextElement() { ... }
}
```

Die Methode `nextElement` liefere beim ersten Aufruf die 2, bei jedem nachfolgenden Aufruf die kleinste Primzahl, die größer als die zuletzt zurückgegebene ist.

Hinweis: Es empfiehlt sich, zur Implementierung die Listen aus der vorigen Aufgabe zuhilfe zu nehmen. Um `int`-Werte in Listen zu speichern, benötigen Sie dann die Wrapper-Klasse `java.lang.Integer`.