Übung zu Algorithmen und Programmieren III, WS 2001/2

Übung 9

Ausgabe: 11.12.01
Abgabe: 20.12.01 bis 14.00 Uhr

Aufgabe 1 (3 + 1 + 4 = 8 P)

1) Schreiben Sie eine Java-Klasse `TwoStacks`, in der Sie zwei Stacks Stack1 und Stack2 in einem einzigen Array implementieren. Die Elemente der beiden Stacks sollen von einem Typ sein, der von der Klasse `Object` erbt. Wenn die Stacks zusammenstoßen, wird ein Fehler erzeugt. Schreiben Sie darüber hinaus eine Testklasse, die

a) zufällig einen der Stacks auswählt,
b) eine push- oder pop-Operation (zufallsgesteuert) ausführt.

2) Messen Sie die Zeit bis Sie in einem 200-elementigen Array eine Kollision haben. Dasselbe bei einem 300-, 400- und 500-elementigen Array.

3) Schreiben Sie ein Java-Programm, das den 2-Stack-Array als zweifarbigen Balken in einem Fenster darstellt und das Füllen des Arrays in der Testklasse anzeigt. Für eine kurze Einführung zum Thema "Fensterprogrammierung unter Java" siehe `http://www.inf.fu-berlin.de/lehre/SS01/alp2/uebungsaufgaben/intro-gui1.html`.

Aufgabe 2 (7 P)

Es gibt Situationen, in denen das Durchlaufen eines Stacks Sinn macht. Z. B. beim Debuggen sind wir als Entwickler sehr dankbar, wenn das Laufzeitsystem uns die Information liefert, wie der gesamte Stack der Methodenaufrufe im Augenblick aussieht.

Das Navigieren durch irgendeine Datenstruktur ist immer implementationsabhängig. Daher ist es sinnvoll, wenn die Datenstruktur selbst einen Iterator zur Verfügung stellt, der die für das Durchlaufen notwendigen Operationen mitbringt. Diesen Iterator kann man dann nutzen, ohne die interne Datenrepräsentation zu kennen. Weiter ist es nützlich, wenn die Datenstruktur beliebig viele Iteratoren anzufordern erlaubt, die ab verschiedenen Positionen die Datenstruktur unabhängig von einander durchlaufen. Das Interface `java.util.Iterator` legt eine Schnittstelle für Klassen fest, die Iteratoren implementieren.

Erweitern Sie Ihre Klasse `TwoStacks` so, dass zwei interne Klassen `ForwardIterator` für Stack1 (also ein Iterator, der am „bottom“ des Stacks1 zu iterieren anfängt) und `BackwardIterator` für Stack2 (ein Iterator, der am „top“ des Stacks2 zu iterieren anfängt) mit den folgenden Eigenschaften zur Verfügung stehen:

```
class ForwardIterator implements java.util.Iterator {
  private Object current = null; // merkt sich das aktuelle Element
  ...
  public ForwardIterator () { ... }
  public boolean hasNext() { ... }
  public Object next() { ... }
  public void remove() { // macht beim Stack keinen Sinn, deswegen
                         // nur eine geeignete Exception werfen ... }
}

class BackwardIterator implements java.util.Iterator {
  private Object current = null; // merkt sich das aktuelle Element
  ...
  public BackwardIterator () { ... }
  public boolean hasNext() { ... }
  public Object next() { ... }
  public void remove() { // macht beim Stack keinen Sinn, deswegen
                         // nur eine geeignete Exception werfen ... }
}
```

Bitte wenden

Sorgen Sie weiter dafür, dass Ihre Klasse `TwoStacks` zwei Methoden zur Verfügung stellt:

```
public java.util.Iterator forwardIterator() {
   return new ForwardIterator();
}

public java.util.Iterator backwardIterator() {
   return new BackwardIterator();
}
```

Schreiben Sie ein Testprogramm, das den Inhalt der beiden Stacks von einem Objekt vom Typ `TwoStacks` mit zwei Iteratoren (Stack1 vorwärts, Stack2 rückwärts) ausgibt. Eine Methode `show` könnte in der Testklasse so aussehen:

```
public static void show(java.util.Iterator iter) {
 while (iter.hasNext())
System.out.println(iter.next());
}
```

Aufgabe 3 (5 P)

Schreiben Sie eine Klasse namens `boolexp.PrintTraverser,` die das Interface `boolexp.Traverser` implementiert und einen booleschen Ausdruck auf dem Bildschirm als Text ausgibt (s. Aufgabe 2 des Übungsblattes 7 und die Musterlösung dieser Aufgabe in `http://www.inf.fu-berlin.de/lehre/WS01/ALP3/uebungen/uebungen.html`). Die Klasse `boolexp.PrintTraverser` soll mit dem folgenden Testprogramm funktionieren.

```
package boolexp;

public class Test {
  public static void main(String argv[]) {
    BoolExpression boolexpr =//entspricht true OR (true OR false) AND true
      new Or(new BoolLiteral(true),
            new And(new Or(new BoolLiteral(true),
                        new BoolLiteral(false)),
                    new BoolLiteral(true)));

 // Die zu schreibenden Klassen zum Durchlaufen (traversieren) der Baeume
    CalculateTraverser c = new CalculateTraverser();
    PrintTraverser p = new PrintTraverser();
    boolexpr.accept(c);
    boolexpr.accept(p);
    System.out.println(" = " + c.getResult());

  }
}
```

Das Testprogramm soll als Ausgabe den Ausdruck:

```
true OR (true OR false) AND true = true
```

haben.

Aufgabe 4 (3 P)

Geben Sie ein Java-Programm an, in dem Sie mindestens 3 fehlerhafte Objektrefernz-Zuweisungen (mit Kommentaren) haben. Lesen Sie dazu bitte den Paragraph „5.2 Assignment Conversion“ der Java-Spezifikation (s. `http://java.sun.com/docs/books/jls/second_edition/html/conversions.doc.html#184206`).