Informatik II (D-ITET)
FS 2009
Prof. Dr. F. Mattern
Philipp Bolliger (bolligph@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Verteilte Systeme
http://www.vs.inf.ethz.ch

Übungsserie Nr. 3

Ausgabe: 11. März 2009
Abgabe: 18./19. März 2009

1. Aufgabe: (5 Punkte) Sortieren von Strings

In Aufgabe 2 des letzten Aufgabenblattes haben Sie das Programm `SortArray.java` vervollständigt, das ein Array von zufallsgenerierten ganzen Zahlen absteigend sortiert.

(1a) *(3 Punkte)* Ändern Sie Ihre Lösung der Aufgabe 2 des letzten Aufgabenblattes so, dass Ihr Programm ein Array von `String` Objekten rekursiv lexikographisch sortiert. Anstatt der Zufallszahlen benutzen Sie nun die Kommandozeilen-Parameter als Eingabe für Ihre Sortierfunktion (einziges Argument der Methode `main`, welches eben ein String-Array ist). Wie müssen Sie den verwendeten Vergleichsoperator ersetzen?

Testen Sie Ihr Programm (nehmen wir an, es heisst `SortString.java`) mit der folgenden Eingabe, wobei alle Wörter Argumente des Programms sind.

```
java SortString Write once run everywhere ?
```

(1b) *(2 Punkte)* Was fällt Ihnen bei der Sortierung auf? Nehmen Sie eine Methode aus der Dokumentation der String-Klasse[1] zuhilfe, um eine von Gross-Kleinschreibung unabhängige Sortierung zu erreichen.

2. Aufgabe: (5 Punkte) Objekte, Strings und Referenzen

Strings gibt es in Java in zwei Varianten: unveränderlich (immutable) als Typ `String` und veränderlich (mutable) als `StringBuffer`. Unveränderlich bedeutet, dass der Speicherplatz des Objekts niemals mit einem anderen Wert überschrieben wird, was bei veränderlichen Objekten möglich ist.

Betrachten Sie das folgende Programm, welches sich von der Webseite herunterladen lässt:

[1] Java-API Dokumentation: `http://java.sun.com/j2se/1.5.0/docs/api/`

```
/**
 * Informatik II - FS2009 <br>
 * Uebungsserie 3, Aufgabe 2 <br>
 *
 *
 * @author Philipp Bolliger
 */
public class StringObject {

    void addString(String s1, String s2) {
        s1 = s1 + s2;
        // Markierung 3
    }

    public static void main(String[] args) {

        StringObject ref = new StringObject();
        String mystring = "Give ";
        StringBuffer mybuffer = new StringBuffer("Give ");
        // Markierung 1

        mystring = mystring + "me six ";
        mybuffer.append("me six ");
        // Markierung 2

        ref.addString(mystring, "lines");
        mybuffer.append("lines");
        // Markierung 4

        System.out.println(mystring);
        System.out.println(mybuffer);
    }
}
```

(2a) *(1 Punkt)* Was ist die Ausgabe des Programms?

(2b) *(4 Punkte)* Beschreiben Sie, welche Objekte an den jeweils markierten Stellen im Programm zur Laufzeit im Speicher existieren. Verwenden Sie dafür folgende informelle Notation: Objekte sind Tripel *Objekt-ID* $\times$ *Typ* $\times$ *Inhalt*, Referenzen sind Paare *Name* $\times$ *Objekt-ID*. Beispiel:

```
    String str = "foo";
    StringBuffer buf = new StringBuffer("foobuffer");
```

führt zu:

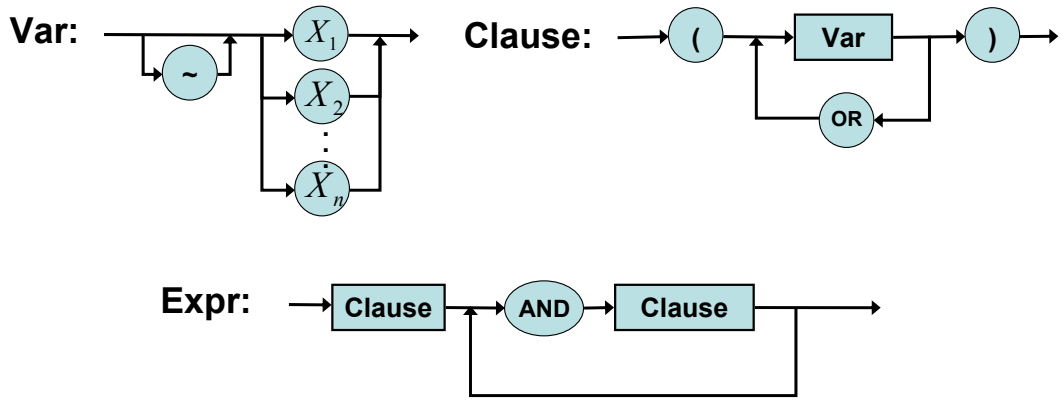
```
    (1, String, "foo")
    (2, StringBuffer, "foobuffer")

    (str, 1)
    (buf, 2)
```

Es existieren also zwei Objekte, eines vom Typ String mit Inhalt “foo” und ein zweites mit Inhalt “foobuffer”. Die Referenz `str` bezieht sich auf das Objekt mit der ID 1 (“foo”), die Referenz `buf` auf das Objekt mit der ID 2 (“foobuffer”).

Nehmen Sie an, dass nicht (mehr) erreichbare Objekte im Speicher erhalten bleiben, also keine Garbage Collection durchgeführt wird.

3. Aufgabe: (4 Punkte) Syntaxdiagramm

Betrachten Sie folgendes Syntaxdiagramm:



(3a) *(2 Punkte)* Geben Sie an, welche der folgenden Ausdrücke nach dem Diagramm 'Clause' korrekt erzeugt werden können.

	erzeugbar	**nicht erzeugbar**		**erzeugbar**	**nicht erzeugbar**
X_2	○	○	$\sim (X_1 OR \sim X_2)$	○	○
$(\sim X_1)$	○	○	$(X_2) OR (\sim X_1 OR X_2)$	○	○

(3b) *(2 Punkte)* Geben Sie an, welche der folgenden Ausdrücke nach dem Diagramm 'Expr' korrekt erzeugt werden können.

	erzeugbar	**nicht erzeugbar**
$(X_1 OR X_2) AND (\sim X_1)$	○	○
$(X_1) AND (\sim X_1 OR \sim X_2) AND (X_2)$	○	○

4. Aufgabe: (10 Punkte) Syntaxanalyse

In der Aufgabe 2 des letzten Übungsblattes haben Sie sich mit der Klammerdarstellung von Bäumen befasst. Nun sollen Sie die Syntax von *Binärbäumen* untersuchen. Zur Erinnerung: Ein Binärbaum ist entweder ein leerer Baum oder er besteht aus einem Knoten, welcher keinen, einen oder zwei Nachfolger hat, welche ebenfalls Binärbäume sind.

(4a) *(5 Punkte)* Stellen Sie nun ein Syntaxdiagramm auf, welches alle möglichen (gültigen und vollständigen) Klammerdarstellungen von *Binärbäumen* generiert. Die Knoten der generierten Bäume sollen als Attribute nur die Terminalsymbole 'A', 'B', ..., 'Z' enthalten. Codieren Sie den leeren Baum mit dem Terminalsymbol '-'.

(4b) *(5 Punkte)* Schreiben Sie ein Java-Programm, welches die Syntax einer (über die Kommandozeile eingegebenen) Klammerdarstellung überprüft und ggf. eine aussagekräftige Fehlermeldung ausgibt. Gehen Sie dabei wie in der Vorlesung besprochen vor. Benutzen Sie für diese Aufgabe das Code-Fragment `LKD.java`, welches Sie von der Vorlesungs-Webseite herunterladen können. Die main-Methode erstellt ein Objekt der Klasse LKD, wobei es den Kommandozeilenparameter als char-Array an den Konstruktor übergibt. Übergeben Sie Ihren Methoden den Index, an welchem die Prüfung zu beginnen hat, und geben Sie den Index zurück, bis zu welchem diese erfolgreich war.

Überlegen Sie sich:

- wie Ihr Programm das vorzeitige Ende der Eingabe erkennen kann,
- wie Ihr Programm das Vorhandensein überflüssiger Zeichen am Ende der Eingabe erkennen kann,
- ob Sie den leeren Baum als gültige Eingabe zulassen wollen.

Überprüfen Sie die Korrektheit Ihres Programms mit den folgenden Eingaben:

- "`-`" (Korrekte Darstellung: der leere Baum)
- "`A`" (Korrekte Darstellung: nur Wurzel)
- "`A(A(A))`" (Korrekte Darstellung)
- "`A(B,C(-,E))`" (Korrekte Darstellung)
- "`B()`" (Fehler: mindestens ein Nachfolger fehlt)
- "`A(B,)`" (Fehler: ein Nachfolger fehlt)
- "`A(B(C)`" (Fehler: schliessende Klammer fehlt)
- "`A(B)C`" (Fehler: überflüssiges `C`)

Beachten Sie, dass der Unix/Dos Kommandointerpreter (shell) gewisse Zeichen in der Kommandozeile interpretiert, wie z.B. Klammern oder den Stern. Umschliessen Sie Ihre Eingabe mit Anführungszeichen, um die Interpretation zu vermeiden.

Summe: 24 Punkte