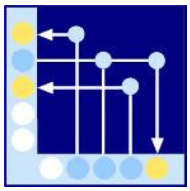
Hochschule Aalen
Fakultät Elektronik und Informatik
Studiengang Informatik



Programmieren 2

Vorlesung im Wintersemester 2014/2015
Prof. Dr. habil. Christian Heinlein

5. Übungsblatt (11. November 2014)

Aufgabe 5: Klassen

Implementieren Sie eine Java-Klasse `StringSet` zur Repräsentation von Mengen mit Elementen des Typs `String`, die die folgenden öffentlichen Konstruktoren und Methoden besitzt:

```
// Leere Menge mit Kapazität n >= 0 (d. h. Platz für n Elemente) erzeugen.
StringSet (int n)

// Leere Menge mit Kapazität n >= 0 erzeugen und anschließend
// Element s einfügen, falls dies möglich ist (vgl. insert).
StringSet (int n, String s)

// Kapazität der Menge (d. h. Wert des Konstruktorparameters n) liefern.
int capacity ()

// Kardinalität der Menge (d. h. tatsächliche Anzahl ihrer Elemente) liefern.
int card ()

// Menge in der Form "{ }", "{ a }", "{ a, b }" etc. ausgeben.
void print () {

// Enthält die Menge das Element s?
boolean contains (String s)

// Element s einfügen und true liefern, falls es noch nicht enthalten
// und nicht null ist und die Kapazität der Menge noch nicht erschöpft ist;
// andernfalls false liefern.
boolean insert (String s)

// Element s entfernen und true liefern, falls es enthalten ist;
// andernfalls false liefern.
boolean remove (Strings s)

// Schnittmenge der Mengen first und second als neue Menge mit
// geeigneter Kapazität liefern (first und second bleiben unverändert).
static StringSet intersection (StringSet first, StringSet second)
```

Speichern Sie die Elemente einer Menge mit Kapazität `n` und Kardinalität `m` in den ersten `m` Plätzen eines Arrays der Größe `n`!
Alle Objektvariablen der Klasse sowie eventuelle Hilfsmethoden (zur Vermeidung von Codeverdopplung!) sollen privat sein.
Außer der Methode `equals` der Klasse `String` sowie `System.out.print` und `System.out.println` sollen keinerlei Java-Bibliotheksmethoden verwendet werden!

Das folgende Programm kann zum interaktiven Testen der Klasse `StringSet` verwendet werden:

```
// Testprogramm für die Klasse StringSet.
class StringSetTest {
    public static void main (String [] args) {
        // Abbruch, wenn keine Konsole verfügbar ist (z. B. in Eclipse).
        if (System.console() == null) {
            System.out.println("console not available");
            return;
        }

        // Aktuelle Menge s.
        StringSet s = null;

        // Endlosschleife.
        while (true) {
            // Kommandozeile lesen und in Wörter zerlegen.
            // Abbruch bei Ende der Eingabe oder leerer Eingabezeile.
            String line = System.console().readLine("command: ");
            if (line == null || line.equals("")) return;
            String [] cmd = line.split(" ");

            // Fallunterscheidung anhand des ersten Zeichens des ersten Worts.
            switch (cmd[0].charAt(0)) {
            default: // new set
                int n = Integer.parseInt(cmd[0]);
                if (cmd.length == 1) s = new StringSet(n);
                else s = new StringSet(n, cmd[1]);
                break;
            case '+': // insert
                System.out.println(s.insert(cmd[1]));
                break;
            case '-': // remove
                System.out.println(s.remove(cmd[1]));
                break;
            case '?': // contains
                System.out.println(s.contains(cmd[1]));
                break;
            case '&': // intersection
                StringSet t = new StringSet(cmd.length - 1);
                for (int i = 1; i < cmd.length; i++) t.insert(cmd[i]);
                s = StringSet.intersection(s, t);
                break;
            }

            // Kardinalität, Kapazität und Inhalt der Menge s ausgeben.
            System.out.print(s.card() + " of " + s.capacity() + " element(s): ");
            s.print();
            System.out.println();
        }
    }
}
```

Lösung

```
// Menge von Strings.
class StringSet {
    // Array von Elementen.
    private String [] elems;

    // Tatsächliche Anzahl der Elemente.
    // Arraypositionen mit Index >= card sind unbenutzt.
    private int card;

    // Position von s im Array elems liefern, falls es enthalten ist.
    // Andernfalls -1 liefern.
    private int search (String s) {
        for (int i = 0; i < card; i++) {
            if (elems[i].equals(s)) return i;
        }
        return -1;
    }

    // Leere Menge mit Kapazität n >= 0 (d. h. Platz für n Elemente) erzeugen.
    public StringSet (int n) {
        elems = new String [n];
        card = 0;
    }

    // Leere Menge mit Kapazität n >= 0 erzeugen und anschließend
    // Element s einfügen, falls dies möglich ist (vgl. insert).
    public StringSet (int n, String s) {
        this(n);
        insert(s);
    }

    // Kapazität der Menge (d. h. Wert des Konstruktorparameters n) liefern.
    public int capacity () {
        return elems.length;
    }

    // Kardinalität der Menge (d. h. tatsächliche Anzahl ihrer Elemente) liefern.
    public int card () {
        return card;
    }

    // Menge in der Form "{ }", "{ a }", "{ a, b }" etc. ausgeben.
    public void print () {
        System.out.print("{");
        String comma = " ";
        for (int i = 0; i < card; i++) {
            System.out.print(comma + elems[i]);
            comma = ", ";
        }
        System.out.print(" }");
    }
```

```
// Enthält die Menge das Element s?
public boolean contains (String s) {
    return search(s) >= 0;
}

// Element s einfügen und true liefern, falls es noch nicht enthalten
// und nicht null ist und die Kapazität der Menge noch nicht erschöpft ist;
// andernfalls false liefern.
public boolean insert (String s) {
    // Wenn das Element s null ist, soll es nicht eingefügt werden.
    if (s == null) return false;

    // Wenn die Kapazität erschöpft ist,
    // kann das Element nicht eingefügt werden,
    // egal, ob es bereits enthalten ist oder nicht.
    // Da diese Überprüfung schneller geht als die nächste,
    // wird sie zuerst durchgeführt.
    if (elems.length == card) return false;

    // Wenn das Element s bereits enthalten ist,
    // soll es nicht nochmals eingefügt werden.
    if (search(s) >= 0) return false;

    // Element an Position card speichern und Kardinalität erhöhen.
    elems[card++] = s;
    return true;
}

// Element s entfernen und true liefern, falls es enthalten ist;
// andernfalls false liefern.
public boolean remove (String s) {
    // Ein null-Element ist niemals in der Menge enthalten.
    if (s == null) return false;

    // Nach dem Element s suchen.
    int i = search(s);

    // Wenn das Element s nicht enthalten ist, ist nichts zu tun.
    if (i == -1) return false;

    // Letztes Element an Position i verschieben
    // und Kardinalität erniedrigen.
    elems[i] = elems[--card];
    elems[card] = null;
    return true;
}
```

```
    // Schnittmenge der Mengen first und second als neue Menge mit
    // geeigneter Kapazität liefern (first und second bleiben unverändert).
    public static StringSet intersection (StringSet first, StringSet second) {
        // Leere Resultatmenge erzeugen.
        // Sie besitzt höchstens so viele Elemente
        // wie die kleinere der beiden Mengen.
        int n = first.card < second.card ? first.card : second.card;
        StringSet result = new StringSet(n);

        // Alle Elemente der Menge first, die auch in der Menge
        // second enthalten sind, zur Resultatmenge hinzufügen.
        for (int i = 0; i < first.card; i++) {
            String s = first.elems[i];
            if (second.contains(s)) result.insert(s);
        }
        return result;
    }
}
```