Hochschule Aalen
Fakultät Elektronik und Informatik
Studiengang Informatik

Programmieren 2

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

7. Übungsblatt (28. November 2014)

Aufgabe 7: Überschreiben von Standardmethoden

Transformieren Sie die Klasse `StringSet` (Musterlösung von Aufgabe 5) in eine Klasse `ObjectSet`, die anstelle von Strings beliebige Objekte als Elemente enthalten kann!

Fügen Sie dann die folgenden drei Standardmethoden hinzu (vgl. Skript, § 5.10):

- `boolean equals (Object other)`
  Liefert genau dann `true`, wenn das Objekt `other` ebenfalls ein `ObjectSet` ist und die gleichen Elemente wie die aktuelle Menge enthält. (Zum Vergleich der Elemente soll wiederum `equals` verwendet werden.)

- `int hashCode ()`
  Liefert als Ergebnis die Summe der Hashwerte aller Elemente der Menge. (Die Hashwerte der einzelnen Elemente sollen wiederum mittels `hashCode` bestimmt werden. Für eine leere Menge erhält man die Summe 0.)

- `String toString ()`
  Liefert die gleiche Zeichenfolge, die die Methode `print` ausgibt. (Zur Umwandlung der einzelnen Elemente in Zeichenfolgen soll wiederum `toString` verwendet werden.)

Vermeiden Sie Code-Verdopplungen!

Verwenden Sie zum Testen u. a. folgendes Programm und erklären Sie, wie bei einem Aufruf der Art

```
java ObjectSetTest 1 2 3 4 5 6 7 8
```

die einzelnen Ausgabezeilen genau zustandekommen!

```
// Punkt im zweidimensionalen Raum.
class Point {
  // Koordinaten des Punkts.
  public final double x, y;

  // Punkt mit Koordinaten x und y konstruieren.
  public Point (double x, double y) { this.x = x; this.y = y; }

  // Zeichenketten-Darstellung des aktuellen Punkts liefern.
  public String toString () { return "(" + x + ", " + y + ")"; }
```

```
  // Vergleich des aktuellen Punkts mit einem anderen Objekt other.
  public boolean equals (Object other) {
    if (!(other instanceof Point)) return false;
    Point that = (Point)other;
    return this.x == that.x && this.y == that.y;
  }

  // Hashwert für den aktuellen Punkt liefern.
  public int hashCode () { return (int)(x + y); }
}

// Testprogramm für die Klasse ObjectSet.
class ObjectSetTest {
    public static void main (String [] args) {
        int n = args.length;

        // Je zwei Kommandozeilenargumente definieren einen Punkt p.
        // Jeder solche Punkt wird in die Menge s1 eingefügt.
        ObjectSet s1 = new ObjectSet(n);
        for (int i = 0; i < n; i += 2) {
            double x = Double.parseDouble(args[i]);
            double y = Double.parseDouble(args[i+1]);
            Point p = new Point(x, y);
            s1.insert(p);
        }
        System.out.println(s1 + " " + s1.hashCode());

        // In die Menge s2 werden die gleichen Punkte eingefügt
        // wie in s1, aber in umgekehrter Reihenfolge.
        ObjectSet s2 = new ObjectSet(n/2);
        for (int i = n - 2; i >= 0; i -= 2) {
            double x = Double.parseDouble(args[i]);
            double y = Double.parseDouble(args[i+1]);
            Point p = new Point(x, y);
            s2.insert(p);
        }
        System.out.println(s2 + " " + s2.hashCode());

        // Vergleich von s1 und s2.
        System.out.println(s1.equals(s2));

        // Da s1 und s2 selbst wieder Objekte sind, können sie in eine
        // dritte Menge s3 eingefügt werden.
        ObjectSet s3 = new ObjectSet(2);
        s3.insert(s1);
        s3.insert(s2);
        System.out.println(s3);
    }
}
```

Lösung

```
// Menge von beliebigen Objekten.
class ObjectSet {
    // Array von Elementen.
    private Object [] elems;

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

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

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

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

    // 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(toString());
    }

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

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

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

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

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

    // Wenn das Element x 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 ObjectSet intersection (ObjectSet first, ObjectSet 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;
    ObjectSet result = new ObjectSet(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++) {
        Object x = first.elems[i];
        if (second.contains(x)) result.insert(x);
    }
    return result;
}

// Aktuelle Menge mit dem Objekt other vergleichen.
// Resultatwert true genau dann, wenn other ebenfalls ein
// ObjectSet ist und die gleichen Elemente wie this enthält.
public boolean equals (Object other) {
    // Wenn other kein ObjectSet ist, kann es nicht gleich this sein.
    if (!(other instanceof ObjectSet)) return false;

    // Andernfalls kann other als ObjectSet that verwendet
    // und mit this verglichen werden.
    ObjectSet that = (ObjectSet)other;

    // Wenn this und that unterschiedliche Kardinalität besitzen,
    // sind sie nicht gleich.
    if (this.card != that.card) return false;

    // Wenn this und that dieselbe Kardinalität besitzen,
    // genügt es zu überprüfen, ob jedes Element von this
    // auch in that enthalten ist.
    for (int i = 0; i < this.card; i++) {
        Object x = this.elems[i];
        if (!that.contains(x)) return false;
    }
    return true;
}

// Hashwert als Summe der Hashwerte der Elemente berechnen.
public int hashCode () {
    int sum = 0;
    for (int i = 0; i < card; i++) {
        sum += elems[i].hashCode();
    }
    return sum;
}
```

```
    // Zeichenkettendarstellung der Form "{ }", "{ a }", "{ a, b }" etc.
    // liefern.
    public String toString () {
        String s = "{";
        String comma = " ";
        for (int i = 0; i < card; i++) {
            s += comma + elems[i];  // Hier wird implizit
                                    // elems[i].toString() ausgeführt.
            comma = ", ";
        }
        s += " }";
        return s;
    }
}
```