Softwaretechnik (WS 2005/2006) - Übungsblatt 11

Jan Hendrik Dithmar - Matrikelnummer 2031259
Übungsgruppe Mi, 8-10 Uhr
Dirk Heine - Matrikelnummer 2030941
Übungsgruppe Di, 14-16 Uhr

Rückgabe: Übungsgruppe Di, 14-16 Uhr

Aufgabe 1

1. `rec1.equal(rec1);`
   Aufruf der `equal`-Methode von `Rectangle`
   Begründung:
   Es wird nur die Klasse `Rectangle` verwendet. `Square` wird nicht genutzt, da man generell `Rectangle`-Objekte nicht in `Square`-Objekte umwandeln kann.

2. - `rec1.equal(rec2);`
     Aufruf der `equal`-Methode von `Rectangle`
   - `rec2.equal(rec1);`
     Aufruf der `equal`-Methode von `Rectangle`
   - `rec2.equal(rec2);`
     Aufruf der `equal`-Methode von `Rectangle`

   Begründung:
   Es werden jeweils Objekte vom Typ `Rectangle` instantiiert, welche die `equal`-Methode der `Rectangle`-Klasse verwenden. Wird an `equal` ein `Square`-Objekt übergeben, wird dieses zu einem `Rectangle` umgewandelt. Dies ist deshalb ohne weiteres möglich, da `Square` von `Rectangle` abgeleitet ist.

3. - `square.equal(rec1);`
     Aufruf der `equal`-Methode von `Rectangle`
   - `square.equal(rec2);`
     Aufruf der `equal`-Methode von `Rectangle`

   Begründung:
   Beim Aufrufen von `square.equal(...)` wird zunächst versucht, die `equal`-Methode von `Square` aufzurufen. Da als Parameter allerdings jeweils ein Objekt vom Typ `Rectangle` (`rec1` und `rec2` wurden als `Rectangle` instantiiert) übergeben wird, muss auf die `equal`-Methode der Oberklasse `Rectangle` zurückgegriffen werden.

   - `rec1.equal(square);`
     Aufruf der `equal`-Methode von `Rectangle`
   - `rec2.equal(square);`
     Aufruf der `equal`-Methode von `Rectangle`

   Begründung:
   Da `rec1` und `rec2` beide vom Typ `Rectangle` sind, wird die `equal`-Methode von `Rectangle` verwendet und das `Square`-Objekt wird in ein `Rectangle` umgewandelt und übergeben.

4. `square.equal(square);`
   Aufruf der `equal`-Methode von `Square`
   Begründung:
   Es wird nur die Klasse `Square` verwendet. Der Zugriff auf die Klasse

`Rectangle` ist nicht nötig, da das Objekt vom Typ `Square` problemlos (ohne Umwandlung) an die `equal`-Methode der Klasse `Square` übergeben werden kann.

Aufgabe 2

(a)

```
qsort []     = []
qsort (x:xs) = (qsort (filter (\y -> y<=x) xs)) ++ [x] ++
               (qsort (filter (\y -> y>x) xs))
```

(b)

Die Verkettung von zwei Listen hat Zeitaufwand von $\mathcal{O}(n)$, wobei n die Länge der ersten Liste ist ($n =$ `length(x::xs)`). Dies ist deshalb der Fall, da die erste Liste vorne an die zweite Liste Element für Element angefügt wird. Insgesamt kommt man so auf `length(x::xs)` Einfügeoperationen.

Aufgabe 3

```
public class Aufgabe3 {
  public class EmptyNode implements Comparable {
    public int compareTo(Object o) {
      // da leere Knoten immer gleich sind, geben
      // wir 0 zurück
      // man könnte auch eine Exception werfen
      return 0;
    }

    int height() {
      // die Höhe eines leeren Knotens ist auch 0
      return 0;
    }
  }

  public class Node extends EmptyNode {
    Comparable marke;

    public int compareTo(Node node) {
      return node.marke.compareTo(this.marke);
    }
  }

  public class Tree extends Node {
    Tree left = (Tree) new EmptyNode();
    Tree right = (Tree) new EmptyNode();

    int height() {
      return 1 + Math.max(left.height(), right.height());
    }

    Tree insert(Comparable o) {
      if (this instanceof EmptyNode) {
        this.marke = o;
      } else {
        switch(this.compareTo(o)) {
        case 0:
          break;
        case -1:
          this.left = this.left.insert(o);
          break;
        case 1:
          this.right = this.right.insert(o);
          break;
        default:
```

```
          // braucht man nur zur Vollständigkeit
          // man könnte auch eine Exception werfen
        }
      }
      return this;
    }
  }
}
```