

# Algorithmen und Programmierung III

## Nachklausur WS 04/05 - Musterlösung

### Aufgabe 1

a) Der Quelltext enthält keine Fehler. Die Ausgabe lautet:

```
6
-1
```

b)

<pre>E[] a = new E[8];</pre>	<p>Hier wird versucht, ein Feld zu erzeugen, dessen Elementtyp durch die Typvariable E bestimmt ist. Dies ist in Java 5 nicht möglich (generic array creation).</p>
<pre>boolean[] b = new boolean[8]; static class D {     boolean m(int i) {         return !b[i];     } }</pre>	<p>Hier wird aus der statischen inneren Klasse D versucht, auf ein nicht-statisches Element der umschließenden Klasse C zuzugreifen. Dies ist nicht möglich.</p>
<pre>void m(byte b) {...} static void test() { ...     int x = 8;     c.m(x); ...}</pre>	<p>Der Typ des Parameters b der Methode m ist byte, aufgerufen wird m mit einem int, int ist jedoch nicht verträglich mit byte.</p>

### Aufgabe 2

```
public class Graph {
    static class Node {
        Set<Node> adj = new TreeSet<Node>(); // Nachbarn
        boolean mark = false; // zwecks Markierung
        int size() {
            if (mark) return 0;
            else mark = true;
            int result = 1;
            mark = true;
            for (Node n : adj)
                result += n.size();
        }
    }
}
```

```

        return result;
    }
}
private Set<Node> nodes;           // Knotenmenge des Graphen
public Vector<Integer> componentSizes() {
    Vector<Integer> result = new Vector<Integer>();
    for (Node n : nodes)           // alte Markierungen löschen
        n.mark = false;
    for (Node n : nodes)
        if (!n.mark)               // falls noch nicht besucht,
            result.add(n.size()); // Komponente von n betrachten
    return result;
}
}

```

### Aufgabe 3

```

BinTree(E val, BinTree<E> left, BinTree<E> right) {
    this.val = val;
    this.left = left;
    this.right = right;
}
public BinTree<E> duplicate() {
    return new BinTree<E>(val == null ? null : val.duplicate(),
                          left == null ? null : left.duplicate(),
                          right == null ? null : right.duplicate());
}

```

### Aufgabe 4

#### a) Repräsentation

```

private String[] stacks;
private int x_top, y_top; // zeigen auf oberstes Element
private final int max;

public StackPair(int size) {
    stacks = new String[size];
    x_top = -1; y_top = size;
    max = size;
}

```

#### b) konkrete Invariante

```

x_top < y_top &&           // keine Überschneidung

```

```
x_top >= -1 && x_top < max && // gültiger Indexbereich
y_top >= 0 && y_top <= max // für x_top und y_top
```

## c) Abstraktionsfunktion

```
abst(stacks, x_top, y_top, max) = (a, b, max)
where a = [stacks[i] | i <- [0..x_top]]
      b = reverse [stacks[i] | i <- [y_top..max-1]]
```

## d) konkrete Spezifikation

```
void pushX(String s)
  pre: x_top+1 < y_top, sonst Overflow
  post: (x_top' = x_top+1) ^
        (all(\i -> stacks'[i] == if i==x_top' then s else stacks[i])
 [0..max-1])
```

```
String popY()
  pre: y_top < max, sonst Underflow
  post: (result = stacks[y_top]) ^ (y_top' = y_top+1)
```