

## Nachtragstest in Programmkonstruktion – 1. Phase

29.5 / 60 Punkte

### 1. Multiple-Choice-Aufgaben

14.5 / 24 Punkte

Bitte wählen Sie *alle* zutreffenden Antwortmöglichkeiten aus. Es können beliebig viele Antwortmöglichkeiten zutreffen, auch alle oder keine.

#### Aufgabe 1.1.

1.5 / 3 Punkte

Wählen Sie jene Deklarationen der Variablen `x` und `y` aus, mit denen der Java-Compiler für `x[0] = y` *keine* Fehlermeldung liefert:

- ☐ `String[] x; final String[] y;`
- ☐ `Queue<String>[][] x; Deque<String>[] y;`
- ☐ `double[] x; double[][] y;`
- ☒ `int[] x; char y;`
- ☒ `char[][]x, int y[][];`
- ☐ `long[][] x; long[] y;`

#### Aufgabe 1.2.

2 / 3 Punkte

Wählen Sie jene Ausdrücke aus, die in einem korrekten Java-Programm immer `true` ergeben, wobei `x` ein Interface ist, `a` durch `x a;` deklariert wurde und `a != null` gilt:

- ☒ `a.toString().equals("" + a)`
- ☒ `a instanceof X`
- ☐ `null instanceof X`
- ☒ `a.getClass().equals(X.class)`
- ☐ `a.getClass() instanceof Class`
- ☒ `!a.equals(null)`

### Aufgabe 1.3.

2 / 6 Punkte

Wählen Sie jene Definitionen der Java-Methode `f` aus, die für alle Parameterwerte im Wertebereich von -10 bis 10 (ohne Überlauf des `int`-Wertebereichs) terminieren:

☒ `int f(int x) { return x % 2 == 0 ? 1 : f(x - 2) * 2; }`

☒ `int f(int x) { return x > 0 ? 1 : f(x * x) * 2; }`

☐ `int f(int x) { return x >= 0 ? 1 : f(x / 4) * 2; }`

☐ `int f(int x) { return x < 0 ? 0 : f(x / 2) + 1; }`

☒ `int f(int x) { return x <= 0 ? 1 : f(x % 2 - 1) * 2; }`

☐ `int f(int x) { return x > 0 ? 1 : f(x + 1) * 2; }`

### Aufgabe 1.4.

3 / 3 Punkte

Angenommen, der Ausdruck `x.equals(y)` liefert `true`. Wählen Sie jene Ausdrücke aus, die an derselben Programmstelle in einem korrekten Java-Programm ebenfalls immer `true` liefern:

☒ `x.equals(x)`

☒ `y.equals(x)`

☒ `x.hashCode() == y.hashCode()`

☒ `x != null && y != null`

☐ `x == y`

☐ `x.toString() == y.toString()`

### Aufgabe 1.5.

1.5 / 3 Punkte

Angenommen, `x` ist eine Variable vom Typ `Deque<Integer>` und `y` eine Variable vom Typ `int[2]`, beide Variablen mit neuen Objekten initialisiert. Wählen Sie jene Anweisungsfolgen aus, die dazu führen, dass nach Ausführung `y[0] == 0 && y[1] == 1` gilt:

☒ `x.offerFirst(0); x.offerFirst(1); y[0] = x.pollLast(); y[1] = x.pollLast();`

☐ `x.offerFirst(0); x.offer(1); y[0] = x.poll(); y[1] = x.poll();`

☐ `x.offer(0); x.offerFirst(1); y[0] = x.poll(); y[1] = x.poll();`

☒ `x.offerFirst(0); y[0] = x.poll(); x.offerFirst(1); y[1] = x.poll();`

☐ `x.offer(0); x.offer(1); y[0] = x.poll(); y[1] = x.poll();`

☒ `x.offer(0); x.offer(1); y[0] = x.pollLast(); y[1] = x.poll();`

### Aufgabe 1.6.

1.5 / 3 Punkte

Wählen Sie jene Anweisungen bzw. Anweisungsfolgen aus, nach deren Ausführung `x[0] == x[1]` gilt:

- ☐ `int[] x = { 1, (char)1.00 };`
- ☒ `int[][] x = { new int[] {}, new int[] {} };`
- ☒ `int[] x = new int[2];`
- ☒ `String[] x = new String[2];`
- ☒ `String s = "a"; String[] x = { s, s };`
- ☐ `int[][] x = new int[2][2];`

### Aufgabe 1.7.

3 / 3 Punkte

Wählen Sie jene Ausdrücke aus, die in Java ein Array erzeugen, welches an mindestens einer Stelle `null` enthält:

- ☒ `new float[8][][ ]`
- ☒ `new String[9][9]`
- ☒ `new double[9][ ]`
- ☐ `new int[][] { new int[] { 0, 8 }, new int[] { 0, 8, 0 } }`
- ☒ `new String[9]`
- ☐ `new char[7][2]`

## 2. Auswahlaufgaben zur Ergänzung von Methoden

3 / 21 Punkte

In den Aufgaben (Programmteilen) sind die Buchstaben A, B, C und D jeweils durch Ausdrücke zu ersetzen. Bitte wählen Sie für jeden dieser Buchstaben genau eine zutreffende Antwortmöglichkeit. Die Methoden müssen sich so verhalten, wie in den Kommentaren angegeben. Punkte gibt es nur, wenn die gewählten Antwortmöglichkeiten für jeweils eine Aufgabe zusammenpassen.

## Aufgabe 2.1.

0 / 4 Punkte

```
// 'printBooleans' prints the character '*' for each 'true' and ' ' for each 'false'
// Thereby, the first index determines the line (Zeile) and the second index determines
// where the character shall be printed.
// All lines before 'i' and all characters on line 'i' at a column before 'j' are already
// the rest of the array remains to be printed.
// Nothing is printed if parameter values are inappropriate.
public static void printBooleans(boolean[][] stars, int i, int j) {
    if (stars != null && i < stars.length && 0 <= i && 0 <= j) {
        if (A) {
            System.out.print(stars[i][j] ? '*' : ' ');
            printBooleans(stars, i, B);
        } else {
            System.out.println();
            printBooleans(stars, C, D);
        }
    }
}
```

A:

- ☐ stars != null && j < stars.length
- ☐ stars[i] != null && i < stars[j].length
- ☐ stars[j] != null && j < stars[i].length
- ☒ stars[j] != null && i < stars[j].length
- ☐ stars != null && i < stars.length
- ☐ stars[i] != null && j < stars[i].length

B:

- ☐ 1
- ☐ j
- ☒ i + 1
- ☐ i
- ☐ j + 1
- ☐ 0

C:

- ☐ 1
- ☐ j
- ☐ i + 1
- ☒ i
- ☐ j + 1
- ☐ 0

D:

- ☐ 1
- ☐ j
- ☒ i + 1
- ☐ i
- ☐ j + 1
- ☐ 0

## Aufgabe 2.2.

3 / 3 Punkte

```
public class Node {
    private String elem;
    private Node next;

    // 'indexOf' returns the index of 'search' in the list (or -1 if not in the list)
    public int indexOf(String search) {
        if (A.equals(search)) {
            return 0;
        }
        if (B == null) {
            return -1;
        }
        int index = C.indexOf(search);
        return index == -1 ? -1 : index + 1;
    }

    /* further methods, constructors, ... */
}
```

A:

- ☐ search ☒ elem ☐ Node ☐ next ☐ this
- ☐ null

B:

- ☐ search ☐ elem ☐ Node ☒ next ☐ this
- ☐ null

C:

- ☐ search ☐ elem ☐ Node ☒ next ☐ this
- ☐ null

### Aufgabe 2.3.

0 / 4 Punkte

```
// 'transpose' changes 'array' by swapping array[x][y] with array[y][x] for all
// (x == i and y >= j and y < x) or (x > i and y < x).
// This is, in the case of i == 0 and j == 0, array[x][y] is swapped with array
// It is assumed that:
//     'array' is quadratic (same length in each dimension),
//     'array' is not null and does not contain null,
//     i and j are in the range between 0 and array.length - 1.
public static void transpose(boolean[][] array, int i, int j) {
    if (i < array.length) {
        if (j < i) {
            boolean b = array[i][j];
            array[i][j] = array[j][i];
            array[j][i] = b;
            transpose(array, A, B);
        } else {
            transpose(array, C, D);
        }
    }
}
```

A:

☒ i   ☐ 1   ☐ 0   ☐ j + 1   ☒ j   ☐ i + 1

B:

☒ i   ☐ 1   ☐ 0   ☒ j + 1   ☐ j   ☐ i + 1

C:

☒ i   ☐ 1   ☐ 0   ☐ j + 1   ☐ j   ☒ i + 1

D:

☐ i   ☐ 1   ☒ 0   ☐ j + 1   ☒ j   ☐ i + 1

## Aufgabe 2.4.

0 / 4 Punkte

```
// 'upSideDown' returns a new array containing the same strings as 'lines', but
// This is, upSideDown(lines)[lines.length - 1 - i] equals lines[i] for each va
// If lines is null, then also the result shall be null.
public static String[] upSideDown(String[] lines) {
    if (lines == null) {
        return null;
    }
    String[] newLines = new String[A];
    reverse(newLines, lines, B);
    return newLines;
}

private static void reverse(String[] newLines, String[] lines, int i) {
    if (i < C) {
        newLines[newLines.length - 1 - i] = lines[i];
        reverse(newLines, lines, D);
    }
}
```

A:

- ☐ i - 1    ☐ i + 1    ☐ 0    ☐ 1    ☒ lines.length
- ☐ lines.length / 2

B:

- ☒ i - 1    ☐ i + 1    ☐ 0    ☐ 1    ☐ lines.length
- ☐ lines.length / 2

C:

- ☐ i - 1    ☐ i + 1    ☐ 0    ☒ 1    ☐ lines.length
- ☐ lines.length / 2

D:

- ☐ i - 1    ☐ i + 1    ☒ 0    ☐ 1    ☐ lines.length
- ☐ lines.length / 2

## Aufgabe 2.5.

0 / 3 Punkte

```
// 'index' returns the smallest index i where i >= low and a[i] is even (gerade)
// returns -1 if there is no such index;
// a != null and low >= 0 always hold
public static int index(final int low, final int[] a) {
    if (A) {
        return -1;
    } else if (B) {
        return index(C, a);
    }
    return D;
}
```

A:

- ☐ low >= a.length
 ☐ low == 0
 ☐ low <= a.length
 ☒ low > a.length
 ☐ low < a.length

B:

- ☐ (low[a] % 2) == 1
 ☐ (a[low] % 1) == 1
 ☒ (a[low] % 2) == 1
 ☐ (low[a] % 1) != 1
 ☐ (low[a] % 1) == 1

C:

- ☐ low - 1
 ☒ low + 1
 ☐ -1
 ☐ 1
 ☐ low

D:

- ☐ low - 1
 ☐ low + 1
 ☐ -1
 ☐ 1
 ☒ low

## Aufgabe 2.6.

0 / 3 Punkte

```
// returns the factorial of n (this is 1 * 2 * ... * n) if n >= 2;
// returns 1 otherwise
public static long fact(final long n) {
    if (A) {
        return B;
    }
    return n * fact(C);
}
```



A:

- ☐ `n == 0`   ☐ `n < 0`   ☐ `n != 0`   ☐ `n >= 0`   ☒ `n <= 0`

B:

- ☐ `n + 1`   ☐ `1F`   ☐ `n - 1`   ☐ `1L`   ☒ `n`

C:

- ☐ `n + 1`   ☐ `1F`   ☒ `n - 1`   ☐ `1L`   ☐ `n`

### 3. Auswahlaufgaben

12 / 15 Punkte

Jede dieser Aufgaben hat genau eine zutreffende Antwortmöglichkeit. Bitte wählen Sie diese aus.

#### Aufgabe 3.1.

3 / 3 Punkte

An welcher Stelle im Programm findet man üblicherweise die *Invarianten* eines Objekts?

- ☐ innerhalb von Konstruktoren
- ☐ am Beginn von Schleifen
- ☐ bei Deklarationen lokaler Variablen
- ☒ bei Deklarationen von Objektvariablen
- ☐ bei Methodenköpfen
- ☐ zwischen zwei Anweisungen

#### Aufgabe 3.2.

3 / 3 Punkte

Ein Objekt fasst Daten und Methoden zu einer Einheit zusammen. Durch welchen der folgenden Begriffe wird diese Eigenschaft von Objekten am genauesten beschrieben?

- ☒ Datenkapselung
- ☐ Objektidentität
- ☐ Subtyping
- ☐ Vererbung
- ☐ Objektverhalten
- ☐ Data-Hiding

### Aufgabe 3.3.

3 / 3 Punkte

Folgender Programmcode ist gegeben:

```
public class Test {  
    private String s = "a";  
    public Test(String s) {  
        this.s += s;  
    }  
    public static void main(String[] args) {  
        Test x = new Test("b");  
        x = new Test("c");  
        System.out.print(x.s);  
    }  
}
```

Welcher Output wird durch Ausführung von `Test.main` generiert?

- ☐ acab
- ☐ acb
- ☐ abc
- ☒ ac
- ☐ abac
- ☐ ab

### Aufgabe 3.4.

3 / 3 Punkte

Welche der folgenden Exceptions muss in einer `throws`-Klausel stehen, falls sie in einer Java-Methode auftreten kann und nicht abgefangen wird?

- ☐ `ArrayIndexOutOfBoundsException`
- ☐ `StackOverflowError`
- ☐ `ArithmeticException`
- ☐ `AssertionError`
- ☐ `NullPointerException`

☒ `IOException`

### Aufgabe 3.5.

0 / 3 Punkte

Folgender Programmcode ist gegeben:

```
interface X {  
    int test();  
}  
class A implements X {  
    public int test() { return 1; }  
}  
class B implements X {  
    public int test() { return 2; }  
}  
public class Test {  
    private void test(X a) {  
        System.out.print(a.test());  
    }  
    public static void main(String[] args) {  
        test(new A());  
        test(new B());  
    }  
}
```

Welcher Output wird durch Ausführung von `Test.main` generiert?

☐ 22

☒ 12

☐ 11

☐ 21

☒ 1

☐ 2