Nachtragstest in Programmkonstruktion – 1. Phase

50.5 / 60 Punkte

1. Multiple-Choice-Aufgaben

21.5 / 24 Punkte

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

Aufgabe 1.1.

6 / 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:

- [x] `int f(int x) { return x >= 0 ? 1 : f(x / 4) * 2; }`
- [ ] `int f(int x) { return x % 2 == 0 ? 1 : f(x - 2) * 2; }`
- [ ] `int f(int x) { return x < 0 ? 0 : f(x / 2) + 1; }`
- [x] `int f(int x) { return x > 0 ? 1 : f(x + 1) * 2; }`
- [ ] `int f(int x) { return x > 0 ? 1 : f(x * x) * 2; }`
- [x] `int f(int x) { return x <= 0 ? 1 : f(x % 2 - 1) * 2; }`

Aufgabe 1.2.

2 / 3 Punkte

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

- [x] `new String[4]`
- [ ] `new int[][]{new int[]{0,2}, new int[]{0,0,9}}`
- [ ] `new float[9][][]`
- [ ] `new double[8][]`
- [ ] `new char[8][3]`
- [x] `new String[8][9]`

Aufgabe 1.3.

3 / 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.offer(0); x.offer(1); y[0] = x.pollLast(); y[1] = x.poll();`
- [ ] `x.offer(0); x.offerFirst(1); y[0] = x.poll(); y[1] = x.poll();`
- [x] `x.offerFirst(0); x.offerFirst(1); y[0] = x.pollLast(); y[1] = x.pollLast();`
- [x] `x.offer(0); x.offer(1); y[0] = x.poll(); y[1] = x.poll();`
- [x] `x.offerFirst(0); x.offer(1); y[0] = x.poll(); y[1] = x.poll();`
- [x] `x.offerFirst(0); y[0] = x.poll(); x.offerFirst(1); y[1] = x.poll();`

Aufgabe 1.4.

2.5 / 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] `x != null && y != null`
- [ ] `x == y`
- [x] `x.equals(x)`
- [x] `y.equals(x)`
- [ ] `x.hashCode() == y.hashCode()`
- [ ] `x.toString() == y.toString()`

Aufgabe 1.5.

2.5 / 3 Punkte

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

- [x] `String[] x = new String[2];`
- [ ] `int[][] x = new int[2][];`
- [x] `String s = "a"; String[] x = { s, s };`
- [x] `int[] x = new int[2];`
- [x] `int[] x = { 1, (char)1.40 };`
- [ ] `int[][] x = { new int[]{}, new int[]{} };`

Aufgabe 1.6.

3 / 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:

- [x] `a.getClass() instanceof Class`
- [ ] `a.getClass().equals(X.class)`
- [x] `a instanceof X`
- [x] `a.toString().equals("" + a)`
- [x] `!a.equals(null)`
- [ ] `null instanceof X`

Aufgabe 1.7.

2.5 / 3 Punkte

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

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

2. Auswahlaufgaben zur Ergänzung von Methoden

17 / 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.

3 / 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`
- ○ `1F`
- ○ `n - 1`
- ● `1L`
- ○ `n + 1`

C:

- ○ `n`
- ○ `1F`
- ● `n - 1`
- ○ `1L`
- ○ `n + 1`

Aufgabe 2.2.

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:

- ● `lines.length`
- ○ `0`
- ○ `i + 1`
- ○ `lines.length / 2`
- ○ `i - 1`
- ○ `1`

B:

- ○ `lines.length`
- ● `0`
- ○ `i + 1`
- ○ `lines.length / 2`
- ○ `i - 1`
- ○ `1`

C:

- ○ `lines.length`
- ○ `0`
- ○ `i + 1`
- ● `lines.length / 2`
- ○ `i - 1`
- ○ `1`

D:

- ○ `lines.length`
- ○ `0`
- ● `i + 1`
- ○ `lines.length / 2`
- ○ `i - 1`
- ○ `1`

Aufgabe 2.3.

3 / 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:

- ○ `(a[low] % 1) == 1`
- ○ `(low[a] % 2) == 1`
- ● `(a[low] % 2) == 1`
- ○ `(low[a] % 1) == 1`
- ○ `(low[a] % 1) != 1`

C:

- ● `low + 1`
- ○ `low`
- ○ `-1`
- ○ `low - 1`
- ○ `1`

D:

- ○ `low + 1`
- ● `low`
- ○ `-1`
- ○ `low - 1`
- ○ `1`

Aufgabe 2.4.

4 / 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`
- ○ `j`
- ○ `i + 1`
- ○ `1`
- ○ `j + 1`
- ○ `0`

B:

- ○ `i`
- ○ `j`
- ○ `i + 1`
- ○ `1`
- ● `j + 1`
- ○ `0`

C:

- ○ `i`
- ○ `j`
- ● `i + 1`
- ○ `1`
- ○ `j + 1`
- ○ `0`

D:

- ○ `i`
- ○ `j`
- ○ `i + 1`
- ○ `1`
- ○ `j + 1`
- ● `0`

Aufgabe 2.5.

4 / 4 Punkte

```
// 'printBooleans' prints the character '*' for each 'true' and ' ' for each 'f
// Thereby, the first index determines the line (Zeile) and the second index de
// where the character shall be printed.
// All lines before 'i' and all characters on line 'i' at a column before 'j' a
// 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 != null && i < stars.length`
- ○ `stars[j] != null && j < stars[i].length`
- ○ `stars[j] != null && i < stars[j].length`
- ● `stars[i] != null && j < stars[i].length`
- ○ `stars[i] != null && i < stars[j].length`

B:

○ `1` ○ `i + 1` ○ `0` ● `j + 1` ○ `i` ○ `j`

C:

○ `1` ● `i + 1` ○ `0` ○ `j + 1` ○ `i` ○ `j`

D:

○ `1` ○ `i + 1` ● `0` ○ `j + 1` ○ `i` ○ `j`

Aufgabe 2.6.

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
    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:

- ○ `next`
- ○ `this`
- ○ `null`
- ○ `Node`
- ○ `search`
- ● `elem`

B:

- ● `next`
- ○ `this`
- ○ `null`
- ○ `Node`
- ○ `search`
- ○ `elem`

C:

- ● `next`
- ○ `this`
- ○ `null`
- ○ `Node`
- ○ `search`
- ○ `elem`

3. Auswahlaufgaben

12 / 15 Punkte

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

Aufgabe 3.1. 3 / 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?

- [x] `12`
- [ ] `11`
- [ ] `2`
- [ ] `1`
- [ ] `21`
- [ ] `22`

Aufgabe 3.2.

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?

- [ ] `ab`
- [ ] `abc`
- [ ] `abac`
- [ ] `acb`
- [x] `ac`
- [ ] `acab`

Aufgabe 3.3.

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?

- [x] Datenkapselung
- [ ] Vererbung
- [ ] Data-Hiding
- [ ] Objektidentität
- [ ] Subtyping
- [ ] Objektverhalten

Aufgabe 3.4. 0 / 3 Punkte

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

- [ ] am Beginn von Schleifen
- [ ] bei Deklarationen von Objektvariablen
- [ ] bei Deklarationen lokaler Variablen
- [ ] bei Methodenköpfen
- [ ] zwischen zwei Anweisungen
- [x] innerhalb von Konstruktoren

Aufgabe 3.5. 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?

- [x] `IOException`
- [ ] `NullPointerException`
- [ ] `StackOverflowError`
- [ ] `ArrayIndexOutOfBoundsException`
- [ ] `AssertionError`
- [ ] `ArithmeticException`