Test 2 in Programmkonstruktion

27 / 40 Punkte

Alle Aufgaben beziehen sich auf Java.

1. Single-Choice Fragen zu Datenstrukturen

12 / 15 Punkte

```
Deque<String> queue = new LinkedList<>();
Map<String, String> map = new HashMap<>();
queue.offer("Alice");
queue.offerFirst("Matthew");
queue.offer("Emily");
map.put("Matthew","Jimmy");
map.put(queue.peek(),"Nancy");
queue.offer(map.get("Alice"));
String x = queue.peek();
map.put("Rose",queue.poll());
String y = map.get(queue.poll());
```

Welche Aussagen sind nach Ausführung der obigen Anweisungen (in dieser Reihenfolge) korrekt?

Aufgabe 1.1.

3 / 3 Punkte

`x` liefert

- ○ null
- ○ "Alice"
- ○ "Emily"
- ● "Matthew"
- ○ einen anderen Wert
- ○ einen Laufzeitfehler

Aufgabe 1.2.

3 / 3 Punkte

`map.get("Matthew")` liefert

- ○ null
- ○ "Alice"
- ○ "Jimmy"
- ○ "Matthew"
- ● einen anderen Wert
- ○ einen Laufzeitfehler

Aufgabe 1.3.

3 / 3 Punkte

`y` liefert

- ● null
- ○ "Alice"
- ○ "Matthew"
- ○ "Nancy"
- ○ einen anderen Wert
- ○ einen Laufzeitfehler

Aufgabe 1.4.

3 / 3 Punkte

`map.get("Rose")` liefert

- ○ null
- ○ "Emily"
- ○ "Jimmy"
- ● "Matthew"
- ○ einen anderen Wert
- ○ einen Laufzeitfehler

Aufgabe 1.5.

0 / 3 Punkte

`queue.poll()` liefert

- ● "Alice"
- ○ "Emily"
- ○ "Nancy"
- ○ einen anderen Wert
- ○ einen Laufzeitfehler

2. Single-Choice Fragen zu Arrays und Rekursion

15 / 15 Punkte

Folgende Implementierungen der Methode `sad` sind syntaktisch korrekt. Die Methode ermittelt die Summe der absoluten Differenzen zwischen benachbarten Elementen des Teilarrays von `a` ab (inklusive) Index `0` bis (inklusive) Index `to`. Vorbedingungen: `0 < to` und `to < a.length`. Beispiel: für `a = { 1, 3, 2, 5 }` und `to = 3` ist das Ergebnis 2 + 1 + 3 = 6. Geben Sie an, welche Aussage auf die jeweilige Implementierung zutrifft.

Aufgabe 2.1.

3 / 3 Punkte

```
public static int sad(int[] a, int to) {
    int sum = 0;
    for (; to > 0; to--) {
        sum += Math.abs(a[to-1] - a[to]);
    }
    return sum;
}
```

Welche Aussage trifft hier zu?

- ○ Laufzeitfehler (erzeugt einen Laufzeitfehler bei bestimmten gültigen Eingaben)
- ○ falscher Wert (bei allen gültigen Eingaben kein Laufzeitfehler aber liefert einen falschen Wert bei bestimmten gültigen Eingaben)
- ● korrekt (liefert für alle gültigen Argumente das korrekte Ergebnis)

Aufgabe 2.2.

3 / 3 Punkte

```
public static int sad(int[] a, int to) {
    int sum=0;
    while (0 < to) {
        int d = a[to] - a[--to];
        sum += (d > 0 ? d : -d);
    }
    return sum;
}
```

Welche Aussage trifft hier zu?

- ○ Laufzeitfehler
- ○ falscher Wert
- ● korrekt

Aufgabe 2.3.

3 / 3 Punkte

```
public static int sad(int[] a, int to) {
    int i = a[to-1] > a[to] ? a[to-1] - a[to] : a[to] - a[to-1];
    return i + ( to == 0 ? 0 : sad(a,to-1) );
}
```

Welche Aussage trifft hier zu?

- ● Laufzeitfehler
- ○ falscher Wert
- ○ korrekt

Aufgabe 2.4.

3 / 3 Punkte

```
public static int sad(int[] a, int to) {
    int n = Math.abs(a[to-1] - a[to]);
    if (to == 1)
        return n;
    else
        return sad(a,to-1) + n;
}
```

Welche Aussage trifft hier zu?

- ○ Laufzeitfehler
- ○ falscher Wert
- ● korrekt

Aufgabe 2.5.

3 / 3 Punkte

```
public static int sad(int[] a, int to) {
    if (to == 1) {
        return Math.abs(a[0] - a[1]);
    }
    return sad(a,to-1) + Math.abs(a[to-1] - a[to]);
}
```

Welche Aussage trifft hier zu?

- ○ Laufzeitfehler
- ○ falscher Wert
- ● korrekt

3. Auswahlaufgaben

0 / 10 Punkte

In den Methoden sind die Buchstaben A, B, C und D jeweils durch einen der vorgeschlagenen Programmteile 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 zusammenpassen.

Aufgabe 3.1.

0 / 5 Punkte

```
// positives gibt eine Deque zurück, die genau die Elemente von q mit Wert > 0,
// in der originalen Reihenfolge, enthält.
// (q wird dabei entleert.)
// Vorbedingung: q != null
//
public static Deque<Integer> positives(Deque<Integer> q) {
    Deque<Integer> qn;
    Integer h = A;
    if (h != null) {
        qn = B;
        if (h > 0) {
            C;
        }
    } else {
        qn = D;
    }
    return qn;
}
```

A:

- ○ `new LinkedList<Integer>()`
- ○ `positives(q)`
- ● `q.poll()`
- ○ `qn.offerFirst(h)`

B:

- ● `new LinkedList<Integer>()`
- ○ `positives(q)`
- ○ `q.poll()`
- ○ `qn.offerFirst(h)`

C:

- ○ `new LinkedList<Integer>()`
- ● `positives(q)`
- ○ `q.poll()`
- ○ `qn.offerFirst(h)`

D:

- ○ `new LinkedList<Integer>()`
- ○ `positives(q)`
- ○ `q.poll()`
- ● `qn.offerFirst(h)`

Aufgabe 3.2.

0 / 5 Punkte

```
// Die Methode 'search' liefert den letzten in 'm' gefundenen
// Eintrag zurück, der nicht der Wert 0 ist.
// D.h., gibt es mehrere Einträge ungleich 0 in 'm' wird unter diesen Einträgen
// mit größtem ersten Index der mit dem größten zweiten Index geliefert.
// Wird kein Eintrag ungleich 0 gefunden, wird 0 zurückgeliefert.
// Vorbedingung: m != null und m.length != 0 sowie
// m[i] != null und m[i].length != 0 für alle gültigen i.
//
// Beispiel:
// search(m) liefert 6, wenn
// m = {{0, 0, 0, 0},
//      {0, 0, 0, 3, 4},
//      {0, 1, 2, 0, 5}
//      {7, 6, 0, 0}}
public static int search(int[][] m) {
    return go(m,0,0);
}

public static int go(int[][] m, int x, int y) {
    int result = 0;
    if(A) {
        result = B;
        if(result != 0) {
            return result;
        }
    }
    if(C) {
        result = D;
        if(result != 0) {
            return result;
        }
    }
    return m[x][y];
}
```

A:

- [ ] `x!=m.length-1`
- [x] `y!=m[x].length-1`
- [ ] `x!=m[y].length-1`
- [ ] `y!=m.length-1`

B:

- [ ] `go(m,x,y-1)`
- [x] `go(m,x,y+1)`
- [ ] `go(m,0,y+1)`
- [ ] `go(m,x+1,0)`

C:

- [x] `x!=m.length-1`
- [ ] `y!=m[x].length-1`
- [ ] `x!=m[y].length-1`
- [ ] `y!=m.length-1`

D:

- [ ] `go(m,x,y-1)`
- [ ] `go(m,x,y+1)`
- [ ] `go(m,0,y+1)`
- [x] `go(m,x+1,0)`