Aufgabenteil zur OOP-Klausur, WS2018

Prüfer: Prof. Dr. Dominikus Herzberg, Fachbereich MNI

Bitte nutzen Sie für Ihre Antworten ausschließlich die separat ausgehändigten Lösungsblätter! Sie geben am Ende der Klausur nur die Lösungsblätter ab!

Ihre Lösungen müssen alle Angaben aus der Aufgabenstellung berücksichtigen. Lesen Sie deshalb die Aufgaben gründlich durch.

1 Fragen (16 × 2 = 32 Punkte)

Hinweis: Pro Frage kann mehr als eine Antwort korrekt sein. Die Fragen sind so formuliert, dass in der Regel nicht ableitbar ist, ob eine oder mehr Antworten zutreffen.

Frage A: Welche Deklaration des Kopfs einer `for`-Schleife ist gültig?

1. `for(int i = 0, j = 1; i < 10; i++, j++)`
2. `for(int i = 0, j = 1;; i++, j++)`
3. `for(int i = 0, float j = 1; ; i++, j++)`
4. `for(String s = ""; s.length() < 10; s += '!')`

Frage B: Welche Zeile repräsentiert ein Literal?

1. `false`
2. `"Hello World"`
3. `712`
4. `boolean`

Frage C: Welche Ausnahme (Exception) muss „abgesichert“ werden?

1. `RuntimeException`
2. Eine Exception, die weder eine `RuntimeException` noch ein `Error` ist
3. `AssertionError`
4. `Error` außer `AssertionError`

Frage D: Welche Anweisung wird von Java beanstandet?

1. `byte b = 3;`
2. `float f = 1024.0;`
3. `byte b = 1024;`
4. `int c = 1_024;`

Frage E: Welchen Wert hat `s` nach der Zuweisung `short s = (byte)1024`?

1. `1024`
2. `1`
3. `0`
4. `24`

Frage F: Gegeben sei die folgende Klasse; der Rumpf der angegebenen Methoden ist nicht von Interesse.

```
class Test1 {
    float aMethod(float a, float b) { /* ... */ }
    // Zeile 3
}
```

Welche aufgeführte Methode kann Zeile 3 ersetzen?

1. `int aMethod(int a, int b) { /* ... */ }`
2. `float aMethod(float a, float b) { /* ... */ }`
3. `float aMethod(float c, float d) { /* ... */ }`
4. `float aMethod(int a, int b, int c) { /* ... */ }`

Vorspann zu Frage G und H: Gegeben sei die folgende Klassendeklaration:

```
class Test extends Base {
    Test(int j) { }
    Test(int j, int k) { super(j,k); }
}
```

Frage G: Um eine Instanz von `Test` zu erzeugen ist welcher Aufruf gültig?

1. `Test t = new Test();`
2. `Test t = new Test(1);`
3. `Test t = new Test(1,2);`
4. `Test t = new Test(1,2,3);`

Frage H: Welcher Konstruktor *muss* ausdrücklich in der Deklaration der Klasse `Base` vorhanden sein?

1. `Base() { }`
2. `Base(int j) { }`
3. `Base(int j, int k) { }`
4. `Base(int j, int k, int m) { }`

Frage I: Was ergibt `-50 >> 1`?

1. `-100`
2. `-25`
3. `-5`
4. Keinen der angegebenen Werte

Frage J: Welche nachfolgende Aussage ist zutreffend? Gegeben sei der folgende Code:

```
Class Xxx {
    int[] ages;
    int[] heights = new int[10];
}
```

1. `ages` wird mit `null` initialisiert.
2. `ages` wird mit einer Referenz auf ein leeres Array initialisiert.
3. `heights` wird mit einer Referenz auf ein leeres Array initialisiert.
4. `heights` wird mit einer Referenz auf ein Array mit zehn Werten initialisiert.

Frage K: Welche Aussage ist zutreffend für die folgende Methode?

```
void callMe(String... names) { /* some code */ }
```

1. Die Methode ist fehlerhaft.
2. Innerhalb der Methode ist `names` ein `String`-Array.
3. Innerhalb der Methode ist `names` eine Liste von `String`s.
4. Die Methode darf ausschließlich innerhalb der umgebenden Klasse aufgerufen werden.

Frage L: Was ist der Wert von `x` nach der Ausführung der folgenden Programmzeile?

```
x = 32 * (31 - 10 * 3);
```

1. Der Wert ist `32`
2. Der Wert ist `31`
3. Der Wert ist `3`
4. Der Wert ist `2016`

Frage M: Welche Aussage zum folgenden Code-Fragment ist gültig?

```
int[] a = [1,2,3,4,5,6,7,8,9,0];
```

1. Der Wert von `a[2]` ist `2`
2. Der Wert von `a[2]` ist `3`
3. Der Wert von `a.length` ist `10`
4. Die Codezeile ist ungültig

Frage N: Welcher Typ ist mit 16 Bits kodiert?

1. `char`
2. `int`
3. `byte`
4. `short`

Frage O: Welcher Lambda-Ausdruck ist syntaktisch korrekt?

1. `x -> x++`
2. `x, y -> x + y`
3. `(x, y) -> x ^ y`
4. `(x) -> ++x`

Frage P: Nach der Ausführung der beiden Anweisungen haben die Variablen welche Werte?

```
int x, a = 6, b = 7;
x = a++ + b++;
```

1. x = 15, a = 7, b = 8
2. x = 15, a = 6, b = 7
3. x = 13, a = 7, b = 8
4. x = 13, a = 6, b = 7

2 BubbleSort (insgesamt max. 36 P. + 4 Bonuspunkte)

Zur kurzen Erinnerung an den BubbleSort: In einer Zahlenfolge wird bei einem Durchlauf von links nach rechts die größere zweier benachbarter Zahlen nach rechts getauscht; wir nennen die Methode, die einen solchen Durchlauf realisiert, `bubble`. Es werden so viele Durchläufe, d.h. `bubble`-Aufrufe gemacht, wie nötig; die dafür verantwortliche Methode nennen wir `bubbleSort`.

2.1 Implementiere einen Durchlauf (22 P.)

Sie sollen die Methode `List<Integer> bubble(List<Integer> nums)` implementieren. Diese Methode nimmt eine Zahlenfolge als Liste entgegen und liefert das Ergebnis eines Durchlaufs in einer neuen(!) Liste zurück. Sie dürfen die Eingangsliste `nums` nicht verändern. Die folgende Variante eines BubbleSort-Durchgangs ist umzusetzen:

1. Lege eine leere Liste namens `result` an
2. Wenn `nums` leer ist, gibt es nichts zu tun
3. Das erste Element von `nums` ist der vorläufig größte Wert `max`
4. Durchlaufe die restlichen Elemente von `nums`. Beim Durchlauf gilt Folgendes:
    - Ist die betrachtete Zahl kleiner/gleich `max`, füge sie `result` hinzu;
    - ansonsten wird `max` zu `result` hinzugefügt und auf den aktuellen Zahlenwert gesetzt
5. Füge nach dem Durchlauf `max` zu `result` hinzu und beende den Durchlauf

Beachten Sie:

- Im Rumpf dürfen max. zwei `if` verwendet werden; ein `else` ist nicht zulässig
- Die einzige Schleife ist eine `for`-Schleife, zu verwenden in der *foreach*-Version
- Sie dürfen insgesamt nicht mehr als neun (9) Semikolons (`;`) verwenden
- Benutzen Sie für Listen ausschließlich die folgenden Methoden: `add(E e)`, `get(int index)`, `isEmpty()` und `subList(int fromIndex, int toIndex)` (wobei `fromIndex` inklusiv und `toIndex` exklusiv ist)

2.2 Das Ergebnis eines Durchlaufs (3 P.)

Welches Ergebnis sehen Sie in der JShell, wenn Sie Folgendes eingeben?

```
jshell> bubble(List.of(5,3,7,1,0,4))
```

2.3 Formuliere Testfälle (3 P.)

Formulieren Sie für die obige `bubble`-Methode sechs (6) Tests. Die Tests sind nicht willkürlich ausgesucht, sondern testen systematisch verschiedene, grundlegende Fälle durch. Verwenden Sie zur Erzeugung von Listen die Methode `List.of` — wenn, dann verwenden Sie ausschließlich positive Ganzzahlen.

Hinweis: Diese Aufgabe kann unabhängig von der Lösung zu `bubble` bearbeitet werden.

2.4 Implementiere mehrere Durchläufe (8 P., 4 Bonuspunkte)

Vervollständigen Sie die Methode `bubbleSort` unter Verwendung von `bubble`.

- Verwenden Sie im Rumpf eine `while`-Schleife
- Ihre Lösung darf nicht mehr als drei (3) Semikolons haben
- Für eine korrekte Lösung, die nur ein (1) Semikolon benötigt, erhalten Sie 4 Bonuspunkte

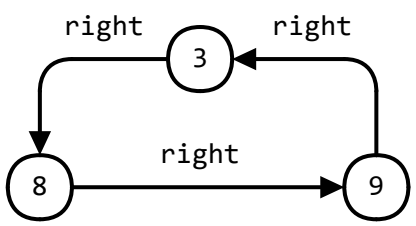
```
List<Integer> bubbleSort(List<Integer> nums) {
    List<Integer> out, in = nums;
    // Ergänzen Sie Ihre Lösung




    return out;
}
```

Hinweis: Diese Aufgabe kann unabhängig von der Lösung zu `bubble` bearbeitet werden.

3 Ringstruktur (insgesamt max. 32 P.)



Die generische Klasse `Ring` dient dazu, um eine Ringstruktur zu realisieren. Sie hat einzig die privaten Variablen `right` und `value`. Im Bild sind drei Instanzen von `Ring` zu sehen, wobei mit `right` auf die jeweils „rechts“ liegende Instanz verwiesen wird. Ein Ring kann nur entlang dieser Verkettung durchlaufen werden. Auf den von einer `Ring`-Instanz beheimateten Wert verweist `value` — im Bild sind es Beispielwerte vom Typ `Integer`.

3.1 Erstelle Aufzählungstyp für Richtungsangabe (2 P.)

Implementieren Sie den Aufzählungstyp `Side` mit den Werten `LEFT` und `RIGHT`.

3.2 Implementiere die Klasse (26 P.)

Gegeben sei das generische Interface `Ringable`, welches von der Klasse `Ring` implementiert wird.

```
interface Ringable<T> {           // Max. Anzahl Semikolons in Implementierung
        void add(T value);        // 3x ";"
        T getValue();             // 1x ";"
        Ringable<T> move(Side s); // 4x ";"
}
```

Mit Blick auf die Implementierung erklären sich die Methoden wie folgt:

- Die Methode `add` fügt eine neue `Ring`-Instanz mit dem übergebenen `value` „rechts“ hinzu
- Die Methode `getValue` liefert den `value` der `Ring`-Instanz zurück
- Die Methode `move` liefert für den Wert `Side.RIGHT` die `Ring`-Instanz „rechts“ zurück und für den Wert `Side.LEFT` die `Ring`-Instanz „links“; hier ist die Herausforderung, über die `right`-Verweise die „linke“ Instanz zurückzugeben.

Implementieren Sie die generische Klasse `Ring`.

- Die Rümpfe der Methoden dürfen die maximal angegebene Anzahl an Semikolons nicht überschreiten.
- Es sind ausdrücklich keine zusätzlichen Instanz- oder Klassenvariablen oder Methoden erlaubt
- Bitte implementieren Sie *nicht* die `toString`-Methode.

Studieren Sie zum Verständnis außerdem den JShell-Dialog. Die `toString`-Methode zeigt den eigenen beheimateten `value` der `Ring`-Instanz an und alle Werte der über `right` verketteten `value`-Werte der `Ring`-Instanzen; der Stern `*` zeigt an, dass nun wieder der Anfangswert in der Ringstruktur folgen würde.

```
jshell> Ring<Integer> r = new Ring<>(8)
r ==> 8 -> *

jshell> r.add(9)

jshell> r
r ==> 8 -> 9 -> *

jshell> r.move(Side.RIGHT).add(3)

jshell> r
r ==> 8 -> 9 -> 3 -> *

jshell> r.move(Side.LEFT)
$44 ==> 3 -> 8 -> 9 -> *

jshell> $44.add(5)

jshell> $44
$44 ==> 3 -> 5 -> 8 -> 9 -> *

jshell> r
r ==> 8 -> 9 -> 3 -> 5 -> *
```

Sollten Sie Probleme mit generischen Typen haben, dann implementieren Sie bitte alternativ die nicht-generische Klasse `Ring` und gehen Sie von dem nicht-generischen Interface `Ringable` in folgender Version aus:

```
interface Ringable {          // Max. Anzahl Semikolons in Implementierung
    void add(int value);      // 3x ";"
    int getValue();           // 1x ";"
    Ringable move(Side s);    // 4x ";"
}
```

Beachten Sie, dass Sie in dieser Aufgabe dann max. 16 Punkte erhalten.

3.3 Implementiere Default-Methode im Interface (4 P.)

Ergänzen Sie das Interface um die Methode `default void add(Side s, T value)`. Mit `s` wird angegeben, ob eine `Ring`-Instanz mit diesem `value` links oder rechts im Ring eingefügt wird.

- Es sind maximal zwei (2) Semikolons erlaubt.

Es ist das Interface `default void add(Side s, int value)` zu ergänzen, wenn Sie sich für die nicht-generische Implementierung von `Ring` entschieden haben. Punktabzüge gibt es für diese Aufgabe keine.