Aufgabenteil zur OOP-Klausur, WS2019

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 nachfolgende Aussage ist zutreffend für die Variable einer Instanz der gegebenen Klasse?

```
1  Class Xxx {
2      int[] ages;
3      int[] heights = new int[0];
4  }
```

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

Frage B: 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 `Strings`.
4. Die Methode darf ausschließlich innerhalb der umgebenden Klasse aufgerufen werden.

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

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

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

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

```
int[] a = new int[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 E: Welcher Typ ist mit 16 Bits kodiert?

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

Frage F: Welche Wertangaben sind nach der Eingabe dieser Zeile korrekt?

```
jshell> int a = 3, b; a = b = a - b = b = 2;
```

1. `a = 3, b = 2`
2. `a = 1, b = 1`
3. `a = 2, b = 0`
4. Die Zeile ist syntaktisch fehlerhaft

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

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

1. `x = 13, b = 8, a = 7`
2. `x = 14, b = 8, a = 7`
3. `x = 13, b = 7, a = 6`
4. `x = 14, b = 7, a = 6`

Frage H: 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 I: Welche Zeile repräsentiert ein Literal?

1. `012`
2. `true`
3. `interface`
4. `"Hello '2' You"`

Frage J: 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 K: Welche Anweisung wird von Java beanstandet?

1. `byte b = 3;`
2. `byte b = 1024;`
3. `int c = 1_024;`
4. `float f = 01024;`

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

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

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

```
1  class Test1 {
2      float aMethod(float a, float b) { /* ... */ }
3      // Zeile 3
4  }
```

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 N und O: Gegeben sei die folgende Klassendeklaration:

```
1  class Test extends Base {
2      Test(int j) { }
3      Test(int j, int k) { super(j,k); }
4  }
```

Frage N: 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 O: 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 P: Was ergibt `-50 >> 2`?

1. `-200`
2. `-100`
3. `-25`
4. Keinen der angegebenen Werte

2 Syntaxbaum (8 Punkte)

Zeichnen Sie den Syntaxbaum zu dem Codefragment. Es gelten ausdrücklich die Darstellungskonventionen aus der OOP-Veranstaltung.

```
int nextDay = day == daysPerMonth(year)[month - 1] ? 1 : day + 1;
```

3 Vererbung (12 Punkte)

Die Klasse `Repeater` erbt von `Speaker`. Ein `Repeater` wiederholt einen übergebenen Text lediglich, wobei ein Leerzeichen die Textwiederholung trennt, siehe Beispiel:

```
jshell> new Speaker("Hi there!")
$30 ==> Hi there!

jshell> new Repeater("Hi there!")
$31 ==> Hi there! Hi there!
```

Implementieren Sie beide Klassen mit dem gezeigten Verhalten. Die Klasse `Repeater` nutzt maximal die Eigenschaften der Vererbung aus und nimmt in keiner Weise Bezug auf Variablen der Klasse `Speaker`. Fügen Sie an geeigneter Stelle die Zusicherung ein, die einen `null`-Wert für die Zeichenkette verbietet.

4 Boolscher Datentyp selbst gemacht (19 Punkte)

4.1 Interface vervollständigen (2 Punkte)

Ergänzen Sie das Interface um eine Standard-Implementierung für die Methode `MyBool nand(MyBool b)`, die das `nand` mit einem `and` und einem anschließenden `not` umsetzt. Im Rumpf der Implementierung ist nur *ein* Semikolon erlaubt.

```
interface Boolable {
    MyBool not();
    MyBool and(MyBool b);



}
```

4.2 Wahrheitswerte als Aufzählungstyp (7 Punkte)

Legen Sie einen Aufzählungstyp namens `MyBool` an, der die zwei Wahrheitswerte `TRUE` und `FALSE` hat und das Interface `Boolable` implementiert.

- Implementieren Sie die Methode `not`. Verwenden Sie den ternären Operator; der Rumpf kommt mit *einem* Semikolon aus.

- Implementieren Sie die Methode `and`. Verwenden Sie ein einziges `if` und kein `else`.

4.3 Klasse `Claim` (10 Punkte)

- Legen Sie eine Klasse `Claim` an, die im Konstruktor eine beliebige Anzahl an Argumenten des Typs `MyBool` entgegennimmt. Die zur Speicherung der Argumente intern verwendete Variable heiße `bools`.
- Legen Sie eine Methode `MyBool andAll()` an, die die in `bools` abgelegten Werte per `and` der Reihe nach verknüpft und das Ergebnis zurückgibt; es wird also gleichermaßen das Ergebnis der „Ver-UND-ung“ aller Werte in `bools` zurückgegeben. Gehen Sie davon aus, dass in `bools` mindestens ein Wert vorliegt. Sichern Sie die Methode entsprechend ab. Verwenden Sie als einziges Schleifenkonstrukt ein einziges `for`.

5 Bereichszähler (29 Punkte + 11 Bonuspunkte)

5.1 Zahlenbereich (leicht, 10 Punkte)

Erstelle eine Klasse namens `Range` zur Repräsentation eines Zahlenbereichs (man könnte auch von einem Zahlenintervall sprechen), das von einer Zahl `from` (inklusive) bis zu einer Zahl `to` (exklusive) reicht. Es handelt sich um Zahlen vom Typ `double`.

- Der Konstruktor der Klasse erwartet zwei `double`-Zahlen `from` und `to`. Wenn `from` größer als `to` ist, soll eine Ausnahme vom Typ `IllegalArgumentException` geworfen werden mit der Nachricht `"from <= to"`.
- Es gibt eine statische Methode `Range of(double from, double to)`, die nichts anderes macht, als den Konstruktor aufzurufen.
- Es gibt eine boolsche Methode `isIn`, die überprüft, ob eine übergebene Zahl `d` innerhalb des Zahlenbereichs liegt (Rückgabe `true`) oder nicht.

5.2 Bereichszähler, 1. Teil (mittelschwer, 13 Punkte)

Ein Bereichszähler bekommt ein Array von Zahlenbereichen übergeben. Der Bereichszähler zählt mit, wie oft eine registrierte Zahl in einem Zahlenbereich liegt.

- Ein Bereichszähler (Klasse `RangeCounter`) bekommt bei der Initialisierung ein Array mit Instanzen von `Range` übergeben; dieses Array wird in `ranges` abgespeichert. Außerdem wird im Konstruktor ein `long`-Array namens `counter` initialisiert. Zum Zahlenbereich `ranges[0]` gehört der Zähler `counter[0]`, zu `ranges[1]` der `counter[1]` usw.

- Die Methode `boolean register(double d)` geht alle Zahlenbereiche durch (verwende eine `for`-Schleife) und inkrementiert all die Counter zu den Zahlenbereichen, zu denen die `isIn`-Methode ein `true` liefert. Wenn mindestens ein Counter erhöht wurde, liefert die Methode ein `true` zurück, ansonsten ein `false`.

Zum Beispiel werde die Variable `rc` wie folgt angelegt:

```
RangeCounter rc = new RangeCounter(
                    new Range[]{Range.of(0,0.25),
                                Range.of(0.25,0.5),
                                Range.of(0.5,0.75),
                                Range.of(0.75,1.0)});
```

Implementieren Sie `RangeCounter`.

5.3 Bereichszähler, 2. Teil (komplizierter, 6 Punkte)

Die `toString`-Methode von `RangeCounter` gibt eine Übersicht über die Zählergebnisse aus in der Reihenfolge der Zahlenbereiche, so wie sie initialisiert wurden. Für den maximalen Zahlenwert (sofern von 0 verschieden), werden 40 „Sterne“ horizontal aufgetragen, gefolgt von der Anzahl der „Treffer“. Die anderen Zahlenwerte werden mit einer entsprechend anteiligen Anzahl an „Sternen“ dargestellt.

Hier ist ein Beispiel für die Ausgabe der Variable `rc`. Hierfür wurden vier Bereiche angelegt, wie oben gezeigt, die mit einer Million zufälligen Zahlenwerten für `register` weitgehend gleichmäßig „getroffen“ wurden.

```
jshell> rc
rc ==>
**************************************** 249715
**************************************** 249937
**************************************** 250209
**************************************** 250139
```

Vervollständigen Sie den nachstehenden Code. Nutzen Sie die *foreach*-Variante der `for`-Schleife. Im zu vervollständigenden Codeabschnitt werden nur exakt *zwei* Semikolons benötigt.

```
public String toString() {
    String s = "\n";
    long max = Arrays.stream(counter).max().getAsLong();



}
```

Die Beschränkung auf zwei Semikolons werden Sie nur einhalten können, wenn Sie die `repeat`-Methode verwenden. Ein Beispiel:

```
jshell> "abc".repeat(3)
$1 ==> "abcabcabc"
```

5.4 Bereichszähler, 3. Teil (anspruchsvoll, Bonus: 4 + 7 Punkte)

1. Vervollständigen Sie die `run`-Methode der Klasse `RangeCounter`. Die Methode registriert `n` von einem `DoubleGenerator` gelieferten Zahlen bei der `register`-Methode. Verwenden Sie eine `while`-Schleife und nur genau ein Semikolon.

```
void run(long n, DoubleGenerator dg) {

}
```

Der Typ `DoubleGenerator` ist wie folgt deklariert:

```
interface DoubleGenerator {
    double generate();
}
```

2. Vervollständigen Sie den Aufruf von `rc.run`. Es sollen eine Millionen Zahlen registriert werden, wobei ein Zufallszahlengenerator übergeben wird für Fließkommazahlen von 0 (inklusiv) bis 1 (exklusiv).

```
rc.run(




)
```