Objektorientierte Programmierung (OOP)
Lösungsblätter zur Klausur, WS 2018/19

WICHTIG

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

Vor- und Nachname	Matrikelnummer	Note

Prüfungsdauer: 90 Minuten

Wertung: Insgesamt können 100 Punkte erreicht werden. Ab 50 Punkten ist die Klausur „bestanden“. Die pro Aufgabe erzielbaren Punkte sind angegeben.

Hilfsmittel: Es sind keine Hilfsmittel erlaubt.

Hinweise:

- Bitte verwenden Sie keinen Rotstift oder Bleistift.
- Tragen Sie in das obige Feld Ihren Namen und Ihre Matrikelnummer ein. Unterschreiben Sie das Deckblatt unten.
- Sie erhalten je einen getrennten Satz an Aufgaben- und Lösungsblättern.
- Schreiben Sie Ihre Antworten bitte ausschließlich auf die Lösungsblätter und dort in die dafür vorgesehenen Abschnitte.
- Geben Sie dieses Deckblatt zusammen mit allen Lösungsblättern ab. Die Aufgabenblätter sind für den Verbleib bei Ihnen bestimmt.
- Überprüfen Sie die Klausur auf Vollständigkeit. Bitte beanstanden Sie Mängelexemplare umgehend.
- Es werden nur leserliche Klausurlösungen bewertet!
- Programmcode hat sich in Art und Form an den in der Veranstaltung verwendeten Konventionen zu orientieren. Schreiben Sie syntaktisch korrekten Code.

Datum	Unterschrift
8. Februar 2019	

1. Fragen (16 x 2 = 32 Punkte)

Tragen Sie hier bitte durch deutliches Ankreuzen Ihre Lösungen ein. Die Spalte „Punkte“ bitte leer lassen.

	1.	2.	3.	4.	*Punkte*		1.	2.	3.	4.	*Punkte*
A	☑	☑	☐	☑		I	☐	☑	☐	☐	
B	☑	☑	☑	☐		J	☑	☐	☐	☑	
C	☐	☑	☐	☐		K	☐	☑	☐	☐	
D	☐	☑	☑	☐		L	☑	☐	☐	☐	
E	☐	☐	☑	☐		M	☐	☐	☐	☑	
F	☑	☐	☐	☑		N	☑	☐	☐	☑	
G	☐	☑	☑	☐		O	☑	☐	☑	☑	
H	☑	☐	☑	☐		P	☐	☐	☑	☐	

Sollten Sie eine Lösung revidieren wollen, streichen Sie die betreffende Reihe durch und notieren Sie hier neben der Tabelle Ihre korrigierte Lösung im Stil von Z13 (die hypothetische Frage Z, Antwort 1 und 3).

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

2.1 Implementiere einen Durchlauf (22 P.)

```
List<Integer> bubble(List<Integer> nums) {
    ArrayList<Integer> result = new ArrayList<>();    // 2
    if (nums.isEmpty()) return result;                // 2
    Integer max = nums.get(0);                        // 2
    for(Integer num : nums.subList(1,nums.size())) { // 2
        if (num <= max) {                             // 2
            result.add(num);                          // 2
            continue;                                 // 2
        }
        result.add(max);                              // 2
        max = num;                                    // 2
    }
    result.add(max);                                  // 2
    return result;                                    // 2
}
```

2.2 Das Ergebnis eines Durchlaufs (3 P.)

```
jshell> bubble(List.of(5,3,7,1,0,4))
$27 ==> [3, 5, 1, 0, 4, 7]           // 3
```

2.3 Formuliere Testfälle (3 P.)

```
assert bubble(List.of()).equals(List.of());              // 1, leere Liste
assert bubble(List.of(1)).equals(List.of(1));            // 1, ein Element
assert bubble(List.of(3,4)).equals(List.of(3,4));        // 1, zwei Elemente
assert bubble(List.of(4,3)).equals(List.of(3,4));        // 1, ... in drei
assert bubble(List.of(4,4)).equals(List.of(4,4));        // 1, ... Beziehungen
assert bubble(List.of(1,2,3)).equals(List.of(1,2,3));    // 1, drei Elemente
assert bubble(List.of(2,1,3)).equals(List.of(1,2,3));
assert bubble(List.of(2,3,1)).equals(List.of(2,1,3));    //    ... besser(!)
assert bubble(List.of(1,3,2)).equals(List.of(1,2,3));
assert bubble(List.of(3,1,2)).equals(List.of(1,2,3));
assert bubble(List.of(3,2,1)).equals(List.of(2,1,3));    //    ... besser(!)
```

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

Tragen Sie bitte nur den fehlenden Code aus dem Methodenrumpf hier ein.

```
List<Integer> bubbleSort(List<Integer> nums) {
    List<Integer> out, in = nums;
    // while(!(out = bubble(in)).equals(in)) in = out;      // dafür: 12 P
    out = bubble(in);                                       // 2
    while(!out.equals(in)) {                                // 2
        in = out;                                           // 2
        out = bubble(in);                                   // 2
    }
    return out;
}
```

3. Ringstruktur (insgesamt max. 32 P.)

3.1 Erstelle Aufzählungstyp für Richtungsangaben (2 P.)

```
enum Side { LEFT, RIGHT; } // 2 Punkte
```

3.2 Implementiere die Klasse (26 P.)

```
class Ring<T> implements Ringable<T> {     // 2
    private Ring<T> right;                 // 2
    private T value;                       // 2

    public Ring(T value) {
        this.value = value;                // 2
        right = this;                      // 2
    }
    public T getValue() {
        return value;                      // 2
    }
    public void add(T value) {
        Ring<T> r = new Ring<>(value);     // 2
        r.right = right;                   // 2
        right = r;                         // 2
    }
    public Ring<T> move(Side s) {
        if (s == Side.RIGHT) return right;             // 2
        Ring<T> next = this;                           // 2
        while(next.right != this) next = next.right;   // 2
        return next;                                   // 2
    }
}
```

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

```
    default void add(Side s, T value) {
        if (s == Side.RIGHT) add(value);  // 2
        else move(Side.LEFT).add(value);  // 2
    }
```