

Objektorientierte Programmierung (OOP)

Aufgabenblätter zur Klausur, WS 2017/18

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

Die Klausur enthält eine Bonusaufgabe mit max. 25 Punkten. Sie haben damit mehr Möglichkeiten, um die maximal benötigten 100 Punkte zu erreichen. Ab 50 Punkten ist die Klausur bestanden.

Fragenteil (33 Punkte für 22 Fragen, 1.5 Punkte/Frage)

Hinweis: Pro Frage kann mehr als eine Antwort korrekt sein!

1. Eine Klasse, die als `class A {}` deklariert wird, wird vom Compiler als `class A extends Object {}` aufgefasst. Zu welcher Definition wird `class B extends A {}` erweitert?
 - ☐ `class B extends A, Object {}`
 - ☐ `class B extends A extends Object {}`
 - ☐ `class B extends class A extends Object {}`
 - ☐ Sie wird gar nicht erweitert. Das ist nur nötig bei Klassen, die kein `extends` besitzen
2. Welche Definition der abstrakten Klasse `AbstractCar` ist syntaktisch erlaubt?
 - ☐ `class AbstractCar { void drive(float distance); }`
 - ☐ `abstract class AbstractCar { void drive(float distance); }`
 - ☐ `abstract class AbstractCar { abstract void drive(float distance); }`
 - ☐ `class AbstractCar { abstract void drive(float distance); }`
3. Rundungsfehler machen einen exakten Vergleich von zwei Fließkommazahlen schwierig. Welche Überprüfung hilft unter Berücksichtigung dieser Problematik festzustellen, ob eine Variable `a` vom Typ `float` „gleich null“ ist?
 - ☐ `Math.abs(a) <= 1E-6`
 - ☐ `Math.round(a) == 0f`
 - ☐ `a == 0.0f`
 - ☐ `a ~ 0.0f`
4. Die Methode `isEven(int n)` ermittelt, ob der übergebene, positive Integer gerade ist oder nicht. Welche Implementierung ist korrekt für den Rumpf der Methode?
 - ☐ `return (n && 1) = 0;`
 - ☐ `return (n & 1) == 0;`
 - ☐ `return (n && 1) == 1;`
 - ☐ `return (n & 1) = 1;`

5. Wie ändert man die Größe von Java-Arrays nachträglich?

- ☐ Arrays sind verkleinerbar, indem man der Variable `length` einen kleineren Wert zuweist
- ☐ Arrays sind vergrößerbar, indem man der Variable `length` einen größeren Wert zuweist
- ☐ Arrays sind nur insofern veränderbar, indem man die Variable `length` auf 0 setzt
- ☐ Man kann die Größe von Arrays nicht verändern.

6. Was sagen Sie zu dem folgenden Code?

```
class Foo<T> { int getLength(T value) { return value.length(); } }
```

- ☐ Code wäre ok, wenn es am Anfang `class Foo<String> { ... }` heißen würde
- ☐ Code wäre ok, wenn es am Anfang `class Foo { ... }` heißen würde
- ☐ Code ist nicht ok: `T` fordert nicht ein, dass es eine `length`-Methode gibt
- ☐ Code ist ok, da `T` ein Interface mit `length`-Methode sein kann

7. Kann man eine Klasse derart in einer Methode deklarieren und nutzen?

```
void foo() { class Strange { }; System.out.println(new Strange()); }
```

- ☐ Nein, die Klasse müsste `private` sein
- ☐ Nein, die Klasse müsste `abstract` sein
- ☐ Ja, es handelt sich hier um eine lokale Klassendefinition
- ☐ Ja, aber nur deshalb, weil die Klasse paketsicher ist

8. Sie sehen die folgende Codezeile: Was kann man dazu sagen?

```
AutoCloseable a = new AutoCloseable() { public void close() { } };
```

- ☐ Da ist ein Fehler im Code, die runden Klammern hinter `AutoCloseable` müssen weg
- ☐ Innerhalb der geschweiften Klammern fehlt ein Semikolon
- ☐ `a instanceof AutoCloseable` ergibt `false` (weil es ein anonymes Interface ist)
- ☐ `a instanceof AutoCloseable` ergibt `true` (weil es eine anonyme Klasse ist)

9. Welche der folgenden Aussagen zu den Operatoren `&` und `&&` sind richtig?

- ☐ `7 & 3` ist kein erlaubter Java-Ausdruck
- ☐ `7 && 3` ist kein erlaubter Java-Ausdruck
- ☐ bei `false && testCondition()` wird `testCondition()` nie aufgerufen
- ☐ bei `false & testCondition()` wird `testCondition()` nie aufgerufen

10. Unter den Schritten, um eine Zahl von der Konsole einzulesen, ist was eine gültige Codezeile?

- ☐ `java.util.Scanner read = new java.util.Scanner(System.in);`
- ☐ `java.util.System read = new java.util.System(Scanner.in);`
- ☐ `java.util.Scanner scan = new java.util.Scanner(System.in);`
- ☐ `java.util.System scan = new java.util.System(Scanner.in);`

11. Welche der folgenden try-Anweisungen sind syntaktisch erlaubt?

- ☐ try { ... } catch (NumberFormatException nfe) { ... }
- ☐ try { ... }
- ☐ try { ... } catch (EventException ee) { ... } catch (IOException ioe) { ... }
- ☐ try { ... } catch (IOException e) { ... } finally { ... }

12. In welchem Fall *kann* (!) eine Methode in ihrem Rumpf eine return-Anweisung enthalten?

- ☐ Das ist eine Fangfrage: Eine Methode muss immer ein return enthalten
- ☐ Wenn der Rückgabotyp nicht void ist
- ☐ Wenn der Rückgabotyp void ist
- ☐ Im Konstruktor kann ein return stehen, was faktisch nie gemacht wird

13. Was ist ein gültiger Ausdruck zur Instanziierung eines char-Arrays?

- ☐ new char[]()
- ☐ new char[9]
- ☐ new char[]{ }
- ☐ new char[9]{ }

14. Warum kann man keine Instanz der Klasse `java.lang.Math` erzeugen?

- ☐ Weil `Math` keinen Konstruktor hat
- ☐ Weil der Konstruktor von `Math` privat ist
- ☐ Weil `Math` als `static` deklariert ist
- ☐ Weil die Klasse `Math` abstrakt ist

15. Welche der folgenden Definitionen von generischen Klassen sind syntaktisch korrekt?

- ☐ class Foo<> { Object o; }
- ☐ class Gen<Eric> { Eric e; }
- ☐ class Test<A,B> { }
- ☐ class <T> MyGeneric { T myGenericVariable; }

16. Die Option `-cp` bei `java` bzw. `javac` steht für welches Java-Konzept?

- ☐ den „codepoint“, an dem die Ausführung des Programms starten soll
- ☐ den „classpath“, in dem nach anderen Klassen und Paketen gesucht wird
- ☐ die „C procedures“, mit denen Code der Sprache C in Java eingebunden werden kann
- ☐ die „character precision“, die angibt wieviel Bytes für einen char verwendet werden

17. `Long.parseLong("11",5)` — Autsch! Warum?

- ☐ Kein Autsch, die Zeile ergibt $1 \times 5 + 1 = 6$
- ☐ Es muss `Int.parseInt(11, 5)` heißen
- ☐ Es gibt kein Zeichen 11 im 5er-System, das gibt einen Fehler
- ☐ Es wird ein Long geparkt, richtig ist `Long.parseLong("11L",5)`

18. Kann ich die Klasse `java.awt.Color` und eine Klasse `de.thm.mni.Color` zusammen im gleichen Stück Code verwenden?
- ☐ Ja, man kann beide unqualifiziert als `Color` bezeichnen, die Typinferenz macht es eindeutig
 - ☐ Ja, wenn man stets den vollqualifizierten Namen verwendet
 - ☐ Ja, wenn man die eine Klasse importiert und die andere vollqualifiziert verwendet
 - ☐ Nein, das geht nicht, man muss Unterklassen mit eigenen Namen einführen
19. Angenommen, Sie haben eine Map namens `book` vom Typ `HashMap<ISBN, Book>`. Wie sieht der Kopf einer `for`-Schleife aus, die über alle ISBNs iteriert?
- ☐ `for(book.keySet(): ISBN isbn)`
 - ☐ `for(ISBN isbn: book.keySet())`
 - ☐ `for(ISBN isbn.keySet(): book)`
 - ☐ `for(book: ISBN isbn.keySet())`
20. Was bedeuten (a) `import de.thm.mni;` und (b) `import de.thm.mni.*;`?
- ☐ (a) importiert den Typ `mni` aus `de.thm`
 - ☐ (a) importiert alle Typen aus `de.thm.mni`
 - ☐ (b) importiert alle Typen aus `de.thm.mni`
 - ☐ (b) importiert den Typ `mni` und alle Untertypen aus `de.thm`
21. Welchen Wert hat `new ArrayList<Integer>(10).size()`?
- ☐ 0
 - ☐ 1
 - ☐ 10
 - ☐ Die Arrayliste ist initial null, so dass der `size`-Aufruf eine `NullPointerException` wirft
22. In welchen Fällen wird bei der folgenden try-with-resources-Anweisung `sc.close()` aufgerufen?
- ```
try (SomeClass sc = new SomeClass()) { foo(sc); } catch (IOException e) {}
```
- ☐ wenn der Aufruf von `foo(sc)` keine Ausnahme wirft
  - ☐ wenn der Aufruf von `foo(sc)` eine `IOException` wirft
  - ☐ wenn der Aufruf von `foo(sc)` eine `NullPointerException` wirft
  - ☐ wenn der Aufruf von `foo(sc)` das Interface `Closeable` implementiert

## 1. Programmieraufgabe: Größenvergleich (10 Punkte)

Die Methode `isGreater` wird aufgerufen mit zwei positiven Ganzzahlen und ermittelt, ob die erste Zahl größer als die zweite Zahl ist; ist dem so, wird ein `true`, ansonsten ein `false` zurückgegeben.

Implementieren Sie eine rekursive Lösung gemäß den folgenden Vorgaben:

- `isGreater(i, j)` ergibt `false`, wenn `i == 0` ist
- `isGreater(i, j)` ergibt `true`, wenn `j == 0` und `i != 0` ist
- ansonsten gilt `isGreater(i, j) == isGreater(i-1, j-1)`

Setzen Sie die Vorgaben um, verzichten Sie aber zum Zwecke der Performanz auf unnötige Vergleiche.

## 2. Programmieraufgabe: Punkt-Distanz (20 Punkte)

Ein Punkt im kartesischen Koordinatensystem ist definiert durch die Angabe der  $x$ - und der  $y$ -Koordinate. Die Distanz zweier Punkte berechnet sich gemäß des Satzes von Pythagoras aus der Wurzel der Summe von  $(x_1 - x_2)^2$  und  $(y_1 - y_2)^2$ . Hierbei sind  $x_1$  und  $y_1$  die Koordinaten des ersten Punkts,  $x_2$  und  $y_2$  die des zweiten.

### 2.1 Konstruktor implementieren

Implementieren Sie eine Klasse `Point`, mit deren Hilfe Punkt-Instanzen erzeugt werden können. Die Werte der zwei Koordinaten (Kommazahlen) werden beim Konstruktoraufbau übergeben.

### 2.2 Distanzberechnung implementieren

- Ergänzen Sie den Code um die Importierung der Methoden von `java.lang.Math`.
- Implementieren Sie die Methode `dist`, die die Distanz zu einem übergebenen Punkt errechnet und zurückgibt. Verwenden Sie geeignete Methoden des `Math`-Pakets.

### 2.3 Repräsentation bereitstellen

Implementieren Sie die Methode `toString`. Ein Punkt mit den beispielhaften Koordinatenwerten 3 und 4 soll sich als `P(3.0,4.0)` darstellen.

## 3. Programmieraufgabe: Schranke (37 Punkte)

In dieser Aufgabe geht es um die Nachbildung einer Schranke (*gate*) in einem Parkhaus. An der Schranke fordert man beim *Check-in* eine Karte (*ticket*) an und erhält eine Karte und Einlass, wenn im Parkhaus freie Plätze verfügbar sind. Bei der Ausfahrt, dem *Check-out*, muss die Karte wieder vorgelegt werden, damit sich die Schranke für den Auslass öffnet.

*Hinweis: Verwenden Sie in Ihrem Code die im Text angegebenen englischen Namen und die vorgegebenen Bezeichnungen für Klassen, Methoden etc. Statische Felder und Methoden sind ausdrücklich verboten. Halten Sie den Code in den Ausdrucksmitteln so kompakt wie möglich und berücksichtigen Sie alle Angaben! Gehen Sie davon aus, dass die Klasse `Ticket` bereits existiert; der parameterlose Default-Konstruktor der Klasse steht zur Verfügung.*

### 3.1 Geprüfte Ausnahme erstellen

Erstellen Sie eine geprüfte (*checked*) Ausnahme namens `GateException`, die – wie üblich für Ausnahmen – eine Nachricht als Zeichenkette entgegennehmen kann. Nutzen Sie die Vererbung von einer geeigneten Ausnahme-Klasse, und verwenden Sie `super` zur Weitergabe des Arguments an die Oberklasse.

### 3.2 Schranke implementieren

Vorbemerkung: In der Aufgabe ist das Interface `Set` zu verwenden. Nutzen Sie ausschließlich die Methoden `add`, `remove` und `size` von `Set`.

Implementieren Sie eine Klasse namens `Gate` (*Schranke*) mit den folgenden Eigenschaften:

- a) Eine Schranke verwaltet
  1. eine Menge von Karten (verwende den Namen `tickets` und das Interface `Set`), die initial leer ist; die Karten sind nur innerhalb der Klasse sicht- und nutzbar
  2. eine Kapazitätsgröße (`capacity`), die die maximale Anzahl an verwalteten Karten definiert; die Kapazität ist zwar öffentlich lesbar aber nicht veränderbar.
- b) Der Konstruktor der Klasse `Gate` nimmt die Kapazitätsangabe des Parkhauses als Ganzzahl entgegen. Stellen Sie per `assert` sicher, dass die Kapazität mindestens 0 ist; ansonsten gibt es die Meldung „*specify appropriate capacity*“.
- c) Die Methode `checkIn` wird ohne Argumente aufgerufen und wirft die Ausnahme `GateException` mit der Nachricht „*capacity exceeded*“, sofern die Anzahl der `tickets` die Kapazität erreicht hat. Ansonsten wird eine Karte (`Ticket`) produziert, in die Kartenverwaltung aufgenommen und von der Methode zurückgegeben.
- d) Die Methode `checkOut` bekommt eine Karte (`Ticket`) übergeben und signalisiert über einen booleschen Wert, ob der Auslass gewährt (`true`) oder verweigert wird (`false`). Die Karte wird bei Auslass aus der Karten-Verwaltung entfernt. Wichtig: Im Rumpf der Methode darf nur ein einziges Semikolon verwendet werden, siehe dazu auch den Folgepunkt.
- e) Geben Sie an, wie die Methode `remove` von `Set` definiert sein muss, damit der voranstehende Aufgabenpunkt mit nur einem Semikolon realisierbar ist.

### 3.3 Interface aufstellen und nutzen

Leiten Sie aus den Methoden der Klasse `Gate` das Interface `Passable` (*passierbar* im Sinne von „Durchlassmöglichkeit“) ab.

- a) Geben Sie den Code für das Interface `Passable` an.
- b) Geben Sie ausschließlich die veränderten Codezeilen der Klasse `Gate` an, die sich aus der Anforderung ergeben, dass die Klasse `Gate` das Interface `Passable` implementieren soll.

### 3.4 Szenario mit JShell-Eingaben

Der vollständige und korrekte Code sei frisch in der JShell geladen. Geben Sie für das nachstehende Szenario die JShell-Eingaben an, die den einzelnen Aktionen entsprechen; pro Punkt wird genau eine Anweisung (ein „Einzeiler“) an die JShell übergeben. Folgender Ablauf ist abzubilden:

- a) Eine Instanz von `Gate` wird erzeugt mit der Kapazität 1
- b) Es gibt einen ersten Check-in
- c) Es gibt einen zweiten Check-in
- d) Es gibt einen Check-out mit einer ungültigen Karte
- e) Es gibt einen Check-out mit der Karte aus dem ersten Check-In

### Bonus-Aufgabe: Palindrom-Erkennung (max. 15 + 10 Punkte)

Implementieren Sie eine Methode `isPalindrom`, die eine Zeichenkette entgegennimmt und einen booleschen Wert zurückgibt; `true`, wenn die Zeichenkette ein Palindrom ist, ansonsten `false`.

Zur Erinnerung: Ein Palindrom ist ein Wort, das vorwärts wie rückwärts gelesen die gleiche Buchstabenfolge aufweist; Beispiele sind die Wörter „Abba“ und „Reittier“. Das funktioniert nur, wenn man übergebene Wörter z.B. mittels `toLowerCase()` im Methodenrumpf vor dem Vergleich in Kleinschreibung umwandelt – tun Sie das bitte auch.

Arbeiten Sie im Methodenrumpf statt mit einer Zeichenkette mit einem Array aus Einzelzeichen namens `word`, nutzen Sie dazu `toCharArray()`. Die Berechnung ist iterativ durchzuführen mit Hilfe zweier Indizes namens `i` und `j`, mit Hilfe derer vom Anfang und vom Ende des Wortes her die Einzelzeichen verglichen werden.

*Für eine funktionsfähige Lösung, die die gestellten Anforderungen erfüllt, erhalten Sie 15 Punkte. 10 weitere Punkte erhalten Sie, wenn es Ihnen darüber hinaus gelingt, eine syntaktisch kompakte Lösung anzugeben, die mit nicht mehr als vier(!) Semikolons im Programmcode auskommt.*

– Ende des Aufgabenblatts –