

## Hinweise zur Probeklausur

- Wir empfehlen, die Probeklausur unter so realistischen Bedingungen wie möglich zu schreiben. Das heißt, Sie sollten sich 120 Minuten möglichst **ungestört** mit der **ausgedruckten** Probeklausur beschäftigen und nur die für die echte Klausur erlaubten Hilfsmittel verwenden.
- Die Punkte pro Aufgabe entsprechen etwa dem Zeitaufwand in Minuten. Falls Sie bei einer Teilaufgabe nicht weiterkommen, überspringen Sie erstmal diese Teilaufgabe; ansonsten verlieren Sie in der echten Klausur unnötig Zeit.
- Sie dürfen bei jeder Aufgabe auf andere Methoden in der Aufgabe zurückgreifen, die sie selbst in der Aufgabe implementieren müssen oder vorgegeben sind. Ersteres ist auch dann erlaubt, falls sie diese Teilaufgabe nicht bearbeitet haben.
- Deckblatt, Hinweise usw. entsprechen voraussichtlich schon denen der echten Klausur. Sie können sich also bereits jetzt damit beschäftigen und alle Fragen dazu klären, dann müssen Sie das Deckblatt in der tatsächlichen Klausur nur noch überfliegen.
- Die Klausur wird doppelseitig gedruckt und oben links getackert. Zusammenhängende Aufgaben werden dabei, nach Möglichkeit (wie auch in der Probeklausur) beim Blättern nebeneinander zu liegen kommen.
- Die Probeklausur wird in der letzten Woche in den Übungen besprochen. Außerdem gibt es voraussichtlich ein Lösungs-Video.
- In der Probeklausur können teilweise Abwandlungen von alten Klausur- und Übungsaufgaben vorkommen. Dies kann in einer echten Klausur ebenfalls der Fall sein, muss es aber nicht.
- Bei jeder Klausur muss eine Auswahl aus den bearbeiteten Themen getroffen werden. Die Auswahl der Themen und die Verteilung der Punkte auf die einzelnen Themen kann in der echten Klausur anders aussehen. Insbesondere werden die echten Klausuren auch in kleinen Aufgaben die Themen der letzten beiden Vorlesungswochen aufgreifen.
  - Folgende Themen kommen beispielsweise in dieser Probeklausur nicht vor, sind aber klausurrelevant (unvollständige Aufzählung): I/O, Löschen in verketteten Listen, Interfaces
- Wir bemühen uns, dass die Probeklausur eher schwieriger und zeitlich knapper kalkuliert ist als Haupt- und Nachklausur, und dass Haupt- und Nachklausur gleich schwierig sind. Da der Schwierigkeitsgrad aber keine objektive Einschätzung ist, sondern von den eigenen Stärken und Schwächen abhängt, können wir das nicht garantieren. Die Klausuren orientieren sich an den im Semester bearbeiteten Materialien und den Lernzielen der Vorlesung.
- Zum weiteren Üben eignen sich insbesondere die Altklausuren von Prof. Schöttner aus den letzten Jahren, die Sie im Klausurarchiv der Fachschaft Informatik<sup>1</sup> finden können.

---

<sup>1</sup><https://fscs.hhu.de/klausur-archiv/>



Januar 2022

**Probeklausur**  
**Programmierung**  
**WS 21/22**

Nachname: \_\_\_\_\_ Vorname: \_\_\_\_\_

Matrikelnummer: \_\_\_\_\_ Sitzplatznummer: \_\_\_\_\_

Zusätzliche Blätter: \_\_\_\_\_ Unterschrift: \_\_\_\_\_

**Hinweise:**

- Diese Klausur enthält 21 nummerierte Klausurseiten. Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält. Sie dürfen die Heftung der Klausur nicht auftrennen.
- Sie erhalten außerdem von uns leere Blätter. Sollten Sie auf diesen Blättern für die Korrektur relevante Teile Ihrer Lösung notieren, markieren Sie dies deutlich an der entsprechenden Aufgabe und auf dem zusätzlichen Blatt. Schreiben Sie auf alle zusätzlichen Blätter Ihren Namen. Geben Sie außerdem oben auf dem Deckblatt an, wie viele zusätzliche Blätter Sie zur Korrektur abgeben. Wenn Sie weiteres Papier benötigen, melden Sie sich bitte.
- Alle Fachbegriffe in dieser Klausur werden wie in der Vorlesung definiert verwendet. Alle Fragen beziehen sich auf die in der Vorlesung vorgestellte Java-Version 11. Programmcode muss in Java geschrieben werden.
- Antworten dürfen auf Deutsch oder Englisch gegeben werden. Sofern keine ausformulierten Sätze verlangt sind, reichen nachvollziehbare Stichworte als Antwort.
- Zugelassene Hilfsmittel: eine beidseitig beschriebene oder bedruckte DIN-A4-Seite, Wörterbuch (Wörterbücher müssen vor Beginn der Klausur den Aufsichtspersonen zur Kontrolle vorgelegt werden.)
- Schalten Sie Ihr Mobiltelefon aus. Täuschungsversuche führen zum sofortigen Ausschluss von der Klausur. Die Klausur wird dann als nicht bestanden gewertet.
- Schreiben Sie **nicht** mit radierbaren Stiften und auch **nicht** mit rot!

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	$\Sigma$
Punktzahl	11	6	5	11	7	9	14	27	90
Erreicht									

**Aufgabe 1**

\_\_\_\_\_ / 11 Punkte

- (a) [2 Punkte] Ihr Terminal sieht gerade wie folgt aus:

```
● ● ●
/mnt/projects %
```

Sie wollen die Klasse `Hello.java` im Ordner `/mnt/projects/blatt01` kompilieren und die enthaltene `main`-Methode ausführen. Welche Befehle müssen Sie dazu eingeben?

---



---



---

- (b) [4 Punkte] Gegeben sei die folgende Klasse:

List.java

```
1 public class List<T> {
2     private class Node {
3         private T data;
4         private Node next;
5
6         private Node(E data, Node next) {
7             this.data = data;
8             this.next = next;
9         }
10    }
11
12    private Node head;
13
14    public void insert(T value) {
15        newElement = new Node(value, head);
16        head = newElement;
17    }
18 }
```

Java

Beim Compilieren der Klasse gibt es folgende Fehlermeldungen:

```
● ● ●
List.java:6: error: cannot find symbol
    private Node(E data, Node next) {
               ^
symbol:   class E
location: class List<T>.Node
where T is a type-variable:
    T extends Object declared in class List
List.java:15: error: cannot find symbol
        newElement = new Node(value, head);
                   ^
symbol:   variable newElement
location: class List<T>
where T is a type-variable:
    T extends Object declared in class List
List.java:16: error: cannot find symbol
        head = newElement;
                   ^
symbol:   variable newElement
location: class List<T>
where T is a type-variable:
    T extends Object declared in class List
3 errors
```

Geben Sie an, in welchen Zeilen die **Ursachen** für die Fehler sind, beschreiben Sie die Fehler jeweils kurz (max. 1 Satz) und geben Sie die korrigierte Codezeile **vollständig** an, sodass der Code das tut, was bei der Programmierung wahrscheinlich vorgesehen war. Geben Sie keine Folgefehler an, die durch Korrektur eines vorherigen Fehlers behoben werden.

Zeilennummer: \_\_\_\_\_

Fehlerbeschreibung: \_\_\_\_\_

Korrektur: \_\_\_\_\_

Zeilennummer: \_\_\_\_\_

Fehlerbeschreibung: \_\_\_\_\_

Korrektur: \_\_\_\_\_

Zeilennummer: \_\_\_\_\_

Fehlerbeschreibung: \_\_\_\_\_

Korrektur: \_\_\_\_\_

(c) [5 Punkte] Gegeben sei das folgende Programm **[Sequence]**.

```
Sequence.java Java
1 public class Sequence {
2
3     private static int numberAt(int index, int n0, int n1) {
4         if(index <= 0) {
5             return n0;
6         }
7         if(index == 1) {
8             return n1;
9         }
10        return numberAt(index - 1, n0, n1) + numberAt(index - 2, n0, n1);
11    }
12
13    public static void main(String[] args) {
14
15
16
17
18
19
20
21
22
23
24
25
26        int n0 = Integer.parseInt(args[0]);
27        int n1 = Integer.parseInt(args[1]);
28        int index = Integer.parseInt(args[2]);
29
30
31
32
33
34
35
36
37
38
39        System.out.println(numberAt(index, n0, n1));
40
41
42
43
44
45
46
47
48
49
50
51
52    }
53
54 }
```

- i) Vervollständigen Sie den folgenden Aufruf des Programms, sodass es ③ ausgeben würde:

`java Sequence 2 1 _____`

- ii) Das Programm stürzt ab, wenn es nicht mit Ganzzahlen aufgerufen wird (**NumberFormatException**) und auch dann, wenn weniger als 3 Argumenten angegeben werden. Passen Sie den Quellcode oben so an, dass in beiden Fällen die benutzerfreundlichere Fehlermeldung „3 Zahlen erwartet“ ausgegeben wird. Fangen Sie dabei Exceptions so genau wie möglich ab (fangen Sie also nicht alle möglichen Exceptions ab).

**Aufgabe 2**

---

 / 6 Punkte

Formen Sie die Kontrollstrukturen in den folgenden Codeausschnitten um, sodass sich die Semantik des Codes nicht ändert. Benutzen Sie in Ihrem Code jeweils nur die von der Aufgabe vorgegebene Art von Kontrollstruktur.

- (a) [2 Punkte] Formen Sie die folgende while-Schleife in eine for-each-Schleife um:

**Vorgabe****Java**

```
int[] numbers = {1, 1, 2, 3, 5, 7};  
int index = 0;  
while(index < numbers.length) {  
    System.out.print(numbers[index]);  
    index++;  
}
```

**Ihre Lösung****Java**

```
int[] numbers = {1, 1, 2, 3, 5, 7};
```

- (b) [2 Punkte] Formen Sie die folgende while-Schleife in eine do-while-Schleife um. Vermeiden Sie dabei unnötige Aufrufe von `Math.random()`.

**Vorgabe****Java**

```
double zahl;  
zahl = Math.random();  
while(zahl > 0.5) {  
    zahl = Math.random();  
}
```

**Ihre Lösung****Java**

```
double zahl;
```

- (c) [2 Punkte] Formen Sie die folgende if-Verzweigung in eine switch-Verzweigung mit minimaler Anzahl von **print**-Statements um:

Vorgabe

Java

```
int choice = 2;
if(choice == 1) {
    System.out.print("1 €");
} else if(choice == 2) {
    System.out.print("1,2 €");
} else if(choice == 3) {
    System.out.print("1 €");
} else {
    System.out.print("ungültig");
}
```

Ihre Lösung

Java

```
int choice = 2;
```

**Aufgabe 3**

---

 / 5 Punkte

Wir haben notiert, wer die Keksdose des Lehrstuhls benutzt hat. Wir wollen wissen, wer von uns diese als letzter, vorletzter usw. benutzt hat.

Schreiben Sie ein Java-Programm **Benutzung**, das beliebig viele Strings als Konsolenargumente entgegennimmt und diese dann in **umgekehrter** Reihenfolge wieder auf der Standardausgabe ausgibt. Dabei soll kein String doppelt ausgegeben werden (es wird also nur das letzte Vorkommen in der Argumentliste ausgegeben).

**Beispiel:**



```
% java Benutzung Dennis Fabian Dennis Markus Jan Markus
Markus Jan Dennis Fabian
```

Benutzung.java Java

**Aufgabe 4**

---

 / 11 Punkte

Gehen Sie in dieser Aufgabe davon aus, dass  $n \in \mathbb{N}$  und  $n \geq 1$  gilt.

Eine natürliche Zahl  $n$  ist eine abundante Zahl, wenn die Summe ihrer natürlichen Teiler ( $n$  ausgeschlossen) **größer** als  $n$  ist. Die kleinste abundante Zahl ist 12 (Teilersumme  $1 + 2 + 3 + 4 + 6 = 16$ ).

Unten ist ein Programm **AbundanteZahlen** mit einer Beispielausgabe vorgegeben. Erweitern Sie dieses Programm um folgende **private, statische** Methoden, damit das Programm wie im Beispiel funktioniert:

- (a) [4 Punkte] Schreiben Sie eine Methode `int teilersumme(int n)`, welche die Summe aller natürlichen Teiler von `n` berechnet, wobei  $n$  selbst **ausgeschlossen** ist.
- (b) [2 Punkte] Schreiben Sie eine Methode `boolean istAbundant(int n)`, die genau dann `true` zurückgibt, wenn `n` eine abundante Zahl ist.
- (c) [5 Punkte] Schreiben Sie eine Methode `int[] abundanteZahlen(int maxN)`, die ein Array zurückgibt, das genau alle abundanten Zahlen enthält, die kleiner oder gleich `maxN` sind.

**Beispiel:**

```
% java AbundanteZahlen
16
true
false
12 18 20
```

AbundanteZahlen.java Java

```
public class AbundanteZahlen {

    public static void main(String[] args) {
        System.out.println(teilersumme(12));
        System.out.println(istAbundant(12));
        System.out.println(istAbundant(13));

        for(int zahl: abundanteZahlen(20)) {
            System.out.print(zahl + " ");
        }
    }

    // Ergänzen Sie hier die Methoden teilersumme, istAbundant und abundanteZahlen.
}
```

AbundanteZahlen.java (Fortsetzung)

Java

}

**Aufgabe 5**

---

 / 7 Punkte

Aus der Vorlesung kennen Sie folgende Implementierung von Insertion Sort, die ein Array von Integern aufsteigend sortiert:

```
Java
public static void sort(int[] numbers) {
    for(int currentIndex = 0; currentIndex < numbers.length; currentIndex++) {
        int currentNumber = numbers[currentIndex];
        int insertionPosition = currentIndex;
        while(insertionPosition > 0 && numbers[insertionPosition - 1] > currentNumber) {
            numbers[insertionPosition] = numbers[insertionPosition - 1];
            insertionPosition--;
        }
        numbers[insertionPosition] = currentNumber;
    }
}
```

Zwei Strings können mithilfe der öffentlichen Instanzmethode `int compareTo(String other)` verglichen werden. Der Rückgabewert stellt hierbei das Ergebnis des Vergleichs wie folgt dar (**a** und **b** vom Typ `String`):

- |                                    |   |
|------------------------------------|---|
| <code>a.compareTo(b) == 0</code>   | → <b>a</b> und <b>b</b> sind gleich                 |
| <code>a.compareTo(b) &gt; 0</code> | → <b>a</b> ist lexikographisch größer als <b>b</b>  |
| <code>a.compareTo(b) &lt; 0</code> | → <b>a</b> ist lexikographisch kleiner als <b>b</b> |

Vervollständigen Sie die Methode `sorted`, die ein String-Array übergeben bekommt und dieses lexikographisch **absteigend** sortiert zurückgibt. Die Reihenfolge der Objekte im übergebenen Array soll dabei von der Methode **nicht** verändert werden. Falls ein Objekt im Array `null` ist, soll eine `IllegalArgumentException` geworfen werden. Sie dürfen davon ausgehen, dass der Methoden-Parameter `strings` ungleich `null` ist.

Ihre Lösung

Java

```
public static String[] sorted(String[] strings) {
```

Ihre Lösung (Fortsetzung)

Java

}

**Aufgabe 6**

---

 / 9 Punkte

Gegeben sei die folgende Klasse `SortedList`, die eine einfach verkettete Liste implementiert, die immer aufsteigend sortiert ist (doppelte Werte sind möglich):

```

SortedList.java                                         Java
1  public class SortedList {
2
3      private class Node {
4          private int data;
5          private Node next;
6
7          private Node(int data, Node next) {
8              this.data = data;
9              this.next = next;
10         }
11     }
12
13     private Node head;
14
15     public void mystery(int value) {
16         head = mystery(value, head);
17     }
18
19     public Node mystery(int value, Node current) {
20         if(current == null) {
21             return null;
22         }
23         if(current.data == value) {
24             return mystery(value, current.next);
25         } else {
26             return new Node(current.data, mystery(value, current.next));
27         }
28     }
29
30
31     // fügt value in die Liste ein, sodass die Liste aufsteigend sortiert bleibt
32     public void insert(int value) {
33         if (head == null || head.data > value) {
34             Node newElement = new Node(value, head);
35             head = newElement;
36             return;
37         }
38
39         Node current = head;
40         while (current.next != null) {
41             if(current.next.data > value) {
42                 Node newElement = new Node(value, current.next);
43                 current.next = newElement;
44                 return;
45             }
46             current = current.next;
47         }
48         current.next = new Node(value, null);
49     }

```

- (a) [1 Punkt] Welche Methoden in der Klasse sind rekursiv definiert? Woran erkennen Sie das?
- 
- 
- 

- (b) [2 Punkte] Was macht die Methode `mystery(int)`?
- 
- 
-

- (c) [6 Punkte] Vervollständigen Sie die Implementierung der Objektmethode `double median()`, die den Median der Liste berechnet. Sie dürfen davon ausgehen, dass die Liste nicht leer ist.

Der Median ist der mittlere Wert in der sortierten Liste. Bei einer geraden Anzahl von Listenelementen ist der Median gleich dem Mittelwert der beiden mittleren Elemente. Beispiel:  $-1, 1, 2, 5, 10, 12$ ; Median:  $\frac{2+5}{2}$

`SortedList.java (Fortsetzung)` Java

```

public double median() {

    int anzahlElemente = _____;

    Node current = _____;

    while(current _____ null) {

        _____;
        _____;
    }

    current = head;

    for(int i = 0; i < anzahlElemente / 2 - 1; i++) {

        _____;
    }

    if(_____) {
        return _____;
    } else {
        return _____;
    }
}

```

**Aufgabe 7**

---

 / 14 Punkte

Gegeben sei die Klasse **Tree** für einen binären Suchbaum mit Integern, wie Sie sie aus der Vorlesung kennen, mit zwei vorgegebenen Objektmethoden:

```

Tree.java Java
3  public class Tree {
4
5      private class BinaryNode {
6          private int element;
7          private BinaryNode left, right;
8
9          private BinaryNode(int element) {
10             this.element = element;
11         }
12     }
13
14     private BinaryNode root;
15
16     // fügt newNumber in den Baum ein
17     public void insert(int newNumber) {
18         // ... (Implementierung nicht abgedruckt)
19
20     }
21
22     // gibt alle Zahlen im Baum als Array zurück
23     public int[] toArray() {
24         // ... (Implementierung nicht abgedruckt)
25     }
26 }
```

- (a) [4 Punkte] Ergänzen Sie die Klasse um eine **öffentliche Objektmethode** `boolean contains(int needle)`, die genau dann `true` zurückgibt, wenn der Wert `needle` im Baum gespeichert ist. Nutzen Sie dabei die Eigenschaften eines binären Suchbaumes aus, um die Anzahl der Vergleiche minimal zu halten.

**Tree.java (Fortsetzung)**

Java

- (b) [5 Punkte] Mit dem Jaccard-Index kann die Ähnlichkeit zweier Menge  $A$  und  $B$  berechnet werden. Dabei wird die Anzahl der gemeinsamen Elemente von  $A$  und  $B$  durch die Gesamtzahl der Elemente in  $A$  und  $B$  (doppelte Elemente nur einfach gezählt) geteilt.

Beispiel:  $A = \{1,2,3\}$ ,  $B = \{3,4,5\}$ , Jaccard-Index:  $\frac{|A \cap B|}{|A \cup B|} = \frac{1}{5}$

Ergänzen Sie die Klasse um eine **öffentliche Klassenmethode**

`double jaccard(Tree a, Tree b)`, die den Jaccard-Index der beiden Bäume  $a$  und  $b$  berechnet. *Tipp: Nutzen Sie die vorgegebenen Methoden.*

Tree.java (Fortsetzung)

Java

}

- (c) [2 Punkte] Kim behauptet, dass die Suche nach einem Wert in einem binären Suchbaum immer schneller ist als in einer (nicht sortierten) verketteten Liste. Stimmen Sie dieser Aussage zu? Begründen Sie Ihre Antwort (z. B. unter Verwendung eines Gegenbeispiels oder eines Beweises).

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- (d) [3 Punkte] Eve möchte ein Programm schreiben, das über die Standardeingabe eine beliebige Anzahl von Unikennungen einlesen kann und anschließend alphabetisch sortiert ausgibt. Sie schlägt vor, die Unikennungen dafür in einem Array der Größe 200 zu speichern und anschließend zu sortieren.

Frank sagt, dass sie lieber einen binären Suchbaum verwenden sollte.

- i) Würden Sie eher Eves oder Franks Vorschlag umsetzen? Geben Sie einen nachvollziehbaren Grund in 1–3 ausformulierten Sätzen an.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

Eve erwidert, dass ein binärer Suchbaum nur mit Zahlen funktioniert.

- ii) Stimmen Sie dieser Aussage von Eve zu? Begründen Sie Ihre Antwort in 1–2 ausformulierten Sätzen.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Aufgabe 8**

---

 / 27 Punkte

In dieser Aufgabe sollen Sie Klassen und Methoden für das Kassensystem eines Buchladens implementieren.

**Hinweise:**

- Sie dürfen alle Variablennamen frei wählen.
- Wählen Sie sinnvolle Datentypen für Ihre Variablen.
- Alle Instanzvariablen müssen **privat** sein.
- Das genaue Format der Textausgaben ist Ihnen überlassen.
- Innerhalb dieser Aufgabe müssen Sie keine Exceptions abfangen. Sie müssen keine Parameter validieren.
- Lesen Sie sich die Aufgabenstellung vor Beginn der Implementierung **vollständig** durch, um einen besseren Überblick über das Gesamtbild zu erhalten.
- Gehen Sie davon aus, dass alle in dieser Aufgabe genannten Klassen im selben Package liegen.

## (a) [9 Punkte]

Schreiben Sie eine **abstrakte**, **öffentliche** Klasse `Produkt`, in welcher es eine **Instanzvariable** gibt, die den Netto-Preis eines Produkts beinhaltet. Ihre Klasse soll einen **Konstruktor** haben, dem der Netto-Preis als Parameter übergeben werden kann und **maximale Sichtbarkeit** hat.

Legen Sie in Ihrer Klasse eine **private**, **finale Klassenvariable** an, die den Steuersatz von 19 % speichert. Der Brutto-Preis des Produkts ( $\text{Netto-Preis} \times (1 + \text{Steuersatz})$ ) soll über eine **öffentliche** Methode `getPreis()` abgefragt werden können; verwenden Sie die Klassenvariable in dieser Methode.

Außerdem soll die `toString`-Methode überschrieben werden, sodass der Netto-Preis zurückgegeben wird.

Produkt.java

Java

(b) [10 Punkte]

Implementieren Sie zwei **nicht abstrakte, öffentliche** Klassen **Buch** und **Spiel**, die sinnvoll von **Produkt** erben. Ein Buch hat einen Preis, einen Titel und eine ISBN. Ein Spiel hat einen Preis und einen Namen. Die spezifischen Eigenschaften von Büchern und Spielen sollen in Instanzvariablen gespeichert werden und alle Eigenschaften über einen **öffentlichen** Konstruktor gesetzt werden können. Die **toString**-Methoden sollen in beiden Klassen überschrieben werden, sodass alle Eigenschaften von Büchern bzw. Spielen zurückgegeben werden.

Buch.java

Java

Spiel.java

Java

(c) [6 Punkte]

Vervollständigen Sie die Klasse **Buchladen**:

Die Methode **gesamtkosten** bekommt ein Array von beliebigen Produkten übergeben und soll den Gesamtpreis (Brutto) zurückgeben.

Ergänze Sie die **main**-Methode, sodass in ihr folgendes passiert:

- Ein **Buch**- und ein **Spiel**-Objekt mit folgenden Eigenschaften werden erstellt:
  - Buch: „Ana“, ISBN: 123, 12,34 €
  - Spiel: „Risk“, 2,45 €
- Die String-Repräsentationen der beiden Objekte werden auf der Standardausgabe ausgegeben.
- Die Gesamtkosten der beiden Produkte (berechnet mithilfe der Methode **gesamtkosten**) werden auf der Standardausgabe ausgegeben.

```
Buchladen.java  
Java  
public class Buchladen {  
  
    private static double gesamtkosten(Produkt[] produkte) {  
  
        }  
  
    public static void main(String[] args) {  
  
        }  
}
```

- (d) [1 Punkt] Die Klasse **Produkt** ist abstrakt, hat aber gar keine abstrakte Methode. Nennen Sie eine Auswirkung, die damit zusammenhängt, dass die Klasse abstrakt ist.

---

---

---

- (e) [1 Punkt] Kreuzen Sie alle Aussagen über die Klassen in dieser Aufgabe an, die richtig sind:

- Buch** ist eine Unterklasse von **Produkt**.
- Spiel** ist eine Oberklasse von **Produkt**.
- Produkt** ist ein Obertyp von **Buch**.
- Produkt** ist ein generischer Datentyp.