Lehrstuhl für Angewandte Informatik 1
Konzepte der Programmierung

Wintersemester 2017/18
06. Februar 2018

Prof. Dr. Bernhard Westfechtel

Konzepte der Programmierung (8 Leistungspunkte)

Probeklausur

Bearbeitungszeit:	120 Minuten
Hinweise:	Beschriften Sie jedes Blatt Ihrer Klausur mit Ihrem Namen und Ihrer Matrikelnummer.
	Die Klausur besteht aus 6 Aufgaben, die **alle** zu bearbeiten sind um die volle Punktzahl zu erreichen. Insgesamt sind 120 Punkte erreichbar.
	Wenn Annahmen gemacht werden müssen, sind diese unbedingt begründen. Schreiben Sie gut leserlich mit dokumentenechtem Stift, unleserliche Teile werden nicht gewertet.
	Die Programmieraufgaben sind in **Java** zu lösen. Wählen Sie für Objekt- und Klassenfelder sinnvolle Datentypen und für die Methoden geeignete Signaturen, sofern diese in der Aufgabenstellung nicht vollständig angegeben sind.
	Hilfsmittel sind nicht erlaubt.

Viel Erfolg!

Aufgabe 1 (Codeverständnis)

Betrachten Sie den nachfolgenden Quellcode.

```
1  public class MathUtil {
2      public static final double ZÄHLER = 0.1;
3      private static double a = multZähler(6);
4
5      public static int multZähler(double ZÄHLER) {
6          return (int) (0.5 * ZÄHLER);
7      }
8
9      public static void main(String[] args) {
10         System.out.println("Der Zähler ist " + ZÄHLER);
11         for (int i = 0; i < a; i += 2) {
12             int a = 5;
13             System.out.println("Berechne " + (a + berechne(3)));
14         }
15     }
16     public static int berechne(int a) {
17         if (a < 0) {
18             System.out.println("Negative Eingabe!"); return -1;
19         }
20         System.out.println(a);
21         double ZÄHLER = a;
22         if (a == 1) return 1;
23         else if (a == 0) return 0;
24         else return a * berechne((int) (ZÄHLER - 1));
25     }
26 }
```

(a) Geben Sie für alle Variablen unter Zeilenangabe an, ob sie **global** oder **lokal** sind.

Kommt es bei den globalen Variablen zu **Überdeckungen**? Falls ja, geben Sie den Bereich der Überdeckung in Zeilen an.

(b) Geben Sie die **Ausgabe** des Hauptprogramms an.

(c) Warum kann man die Methode `multZähler` in Zeile 3 mit einer Ganzzahl aufrufen, obwohl die Methode eine Gleitpunktzahl erwartet?

Warum muss hingegen der Ergebnistyp in Zeile 24 **explizit verengt** werden?

(4 + 6 + 2 = 12 Punkte)

Aufgabe 2 (Syntax)

Betrachten Sie die **kontextfreie Sprache** L_s, die den Aufbau eines *Supermarkt*s beschreibt. Der in der Realität dreidimensionale Markt wird dabei linear dargestellt. Dabei gelten die Konventionen „von Anfang nach Ende“, „von unten nach oben“ und „von links nach rechts“.

Direkt am Anfang der Verkaufsfläche befindet sich der Ein- und Ausgang (`E`) gefolgt von einer Kasse (`K`). Dann folgt der Regalbereich. Am Ende der Verkaufsfläche gibt es optional eine Frischetheke (`F`).

Im Regalbereich wechseln sich Regale und Gänge (`G`) ab. Der Regalbereich wird von Gängen umschlossen, sodass es einen Gang mehr als Regale gibt. Es gibt zudem mindestens zwei Regale.

Ein Regal hat immer ein Brett (`B`) am Boden. Darauf befindet sich der Regalinhalt, und es können beliebige weitere Bretter mit Inhalt folgen. Am oberen Ende des Regals muss allerdings kein Brett mehr sein.

Regalinhalt setzt sich wie folgt aus Dosen (`D`) und Tetrapacks (`T`) zusammen. Alle Dosen stehen immer links von allen Tetrapacks. Pro Regalfach gibt es genau doppelt so viele Tetrapacks wie Dosen. Der Inhalt kann auch leer sein.

Ein *Beispiel-Supermarkt* könnte folgendermaßen aussehen:

```
E K G B D T T B D D T T T T B B D T T G B B D D T T T T G F
```

Geben Sie eine **BNF (kontextfreie Grammatik)** G_S an, die alle Worte in der Sprache L_S erzeugt.

(**18 Punkte**)

Aufgabe 3 (Methoden, Arrays und Ausnahmen)

Ein *Einmaleins-Dreieck* der Größe n besteht aus n Zeilen mit je **bis zu** n Spalten. Die Einträge (i, j) enthalten das ganzzahlige Produkt $(n-i) \cdot (j+1)$ der Faktoren $(n-i)$ und $(j+1)$, wobei $i, j \in \{0, \ldots, n-1\}$. Die folgende Tabelle zeigt ein *Einmaleins-Dreieck* der Größe 7:

	$(j=0)$	(1)	(2)	(3)	(4)	(5)	(6)
$(i=0)$	7	14	21	28	35	42	49
(1)	6	12	18	24	30	36	
(2)	5	10	15	20	25		
(3)	4	8	12	16			
(4)	3	6	9				
(5)	2	4					
(6)	1						

Schreiben Sie in Java eine Klasse `Einmaleins` mit den folgenden öffentlichen **Klassenmethoden**. Verwenden Sie für Ganzzahlen den Datentyp `int`.

(a) `erzeugeDreieck(...)`: Ihr wird die Höhe des Dreiecks, n, übergeben. Sie soll ein 2D-Array mit **unterschiedlich langen** Zeilen und der entsprechenden Höhe initialisieren, indem die Werte an allen gültigen Positionen mit 0 belegt werden. Dieses Array soll von der Methode zurückgegeben werden.

(b) `setzeEintrag(...)`: Ihr werden zwei Faktoren i und j sowie ein entsprechendes Dreieck übergeben. Als Seiteneffekt schreibt die Methode das richtige Produkt an die entsprechende Stelle im übergebenen Dreieck.

Sie können davon ausgehen, dass erlaubte Werte für i und j übergeben werden. Sie brauchen **in dieser Teilaufgabe** also **keine Ausnahmebehandlung** zu implementieren.

(c) `berechneEinmaleins(...)`: Sie bekommt die Größe n übergeben, erstellt ein *Einmaleins-Dreieck* der Größe n mit gesetzten Werten und gibt es zurück.

(d) `gibEintrag(...)`: Ihr werden die zwei Faktoren i und j, sowie eine Dreiecksmatrix übergeben. Sie soll den in der Matrix gespeicherten Wert an der gegebenen Stelle, entsprechend dem Ergebnis $(n-i) \cdot (j+1)$, zurückgeben. Falls die Indizes ungültig sind, soll eine **Laufzeit-Ausnahme** der Klasse `IndexOutOfBoundsException` geworfen werden.

(e) Schreiben Sie ein **Hauptprogramm**, in dem das *Einmaleins-Dreieck* der Größe 5 erzeugt wird. Anschließend soll das Produkt zweier von der Konsole eingelesenen Integer-Zahlen (`In.readInt()`) aus dem Dreieck ausgelesen und auf der Standardausgabe ausgegeben werden. Fangen Sie eine etwaige in `gibEintrag(...)` geworfene Ausnahme ab, indem Sie eine **Fehlermeldung** ausgeben.

(5 + 3 + 6 + 6 + 6 = 26 Punkte)

Aufgabe 4 (Schleifen und Rekursion)

Die natürliche *Exponentialfunktion* e^z kann numerisch über eine Reihe angenähert werden:

$$e^z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \frac{z^3}{3!} + \dots$$

Die Anzahl der Glieder nach der 1 wird als Abbruchgenauigkeit bezeichnet.

Beispielsweise ergibt e^5, mit 4 als Abbruchgenauigkeit:

$$e^5 = 1 + \frac{5}{1!} + \frac{5^2}{2!} + \frac{5^3}{3!} + \frac{5^4}{4!} = 65.375$$

(a) Implementieren Sie als Erstes eine **Klassenmethode** `fakultät(...)`, die eine Ganzzahl übergeben bekommt und die Fakultät dieser Zahl als Wert des größtmöglichen Ganzzahldatentyps zurückgibt. Implementieren Sie die Methode **iterativ**.

(b) Implementieren Sie nun eine **rekursive Klassenmethode** `exponential(...)`. Ihr wird der Exponent und als zweiter Parameter die Genauigkeit übergeben. Sie gibt die Annäherung der Exponentialfunktion unter der Verwendung der gegebenen Genauigkeit zurück.

Hinweis: Zur Berechnung der Potenzen sollen Sie die Klassenmethode der `Math.pow(...)` benutzen. Der Methode wird als erster Parameter die Basis und als zweiter Parameter der Exponent übergeben. Sie liefert einen `double`-Wert zurück.

(c) **Kategorisieren** Sie Ihre rekursive Methode hinsichtlich der drei aus der Vorlesung bekannten Kriterien.

(4 + 12 + 3 = 19 Punkte)

Aufgabe 5 (Objektorientierte Programmierung und Vererbung)

In dieser Aufgabe implementieren Sie eine Vererbungshierarchie für motorisierte Fahrzeuge. Dabei soll der aktuelle Fahrzeugwert berechnet sowie die Anzahl der gefahrenen Kilometer verändert werden können. Betrachten Sie zunächst das folgende Interface, das die Grundfunktionalität für ein `Fahrzeug` deklariert.

```
1  public interface Fahrzeug {
2      /**
3       * @param aktJahr   Jahr, in dem der Wert bestimmt wird
4       * @return          aktueller Wert in Euro
5       */
6      public double gibAktWert(int aktJahr);
7      /**
8       * Addiert die gefahrenen Kilometer zur Gesamtkilometerleistung.
9       * @param km        gefahrene Kilometer
10      */
11     public void fahre(int km);
12 }
```

(a) Implementieren Sie eine **Klasse** `KFZ`, die das Interface `Fahrzeug` **vollständig implementiert**. Fügen Sie der Klasse außerdem folgende Komponenten hinzu:

- Eine **öffentliche Konstante** `STEUERSATZ`, die mit 0.19 initialisiert ist und den Mehrwertsteuersatz repräsentiert.
- Ein **nur für Subtypen sichtbares Objektfeld** `baujahr`, das im Konstruktor gesetzt wird und angibt wann das Fahrzeug gefertigt wurde.
- Ein **nur für Subtypen sichtbares Objektfeld** `listenPreis`, das den Neupreis laut Herstellerliste enthält. Es wird ebenfalls im Konstruktor gesetzt.
- Ein **nur für Subtypen sichtbares Objektfeld** `wertverlust`, das im Konstruktor gesetzt wird und den prozentualen Wertverlust pro Jahr eines Fahrzeugs speichert.
- Ein **nur für Subtypen sichtbares Objektfeld** `gefahreneKM`, das mit 0 initialisiert wird und die bereits gefahrenen Kilometer des Fahrzeugs speichert.

Der aktuelle Wert eines KFZ berechnet sich wie folgt:

$$(1 + STEUERSATZ) \cdot listenPreis \cdot (1 - wertverlust)^{alter}$$

(b) Geben Sie außerdem eine **Klasse** `Oldtimer` an, die von `KFZ` erbt, sich jedoch im Verhalten wie folgt unterscheidet:

- Oldtimer haben hier keinen Wertverlust.
- Der aktuelle Wert berechnet sich stattdessen wie folgt:

$$(1 + STEUERSATZ) \cdot listenPreis \cdot \frac{alter}{10}$$

(16 + 8 = 24 Punkte)

Aufgabe 6 (Datenstrukturen und Generizität)

Gegeben sei nachfolgende Implementierung einer *einfach verketteten generischen Liste*.

```
1  public class Liste<T> {
2
3      private class Eintrag {
4          T inhalt;
5          Eintrag nächster;
6
7          Eintrag(T inhalt){
8              this.inhalt = inhalt;
9          }
10     }
11
12     private Eintrag<T> erster;
13
14     public Liste(T t) {
15         if (t != null) {
16             erster = new Eintrag(t);
17         }
18     }
19 }
```

Erweitern Sie die Klasse `Liste<T>` um folgende Methoden:

(a) Eine **Objektmethode** `länge(...)`, die die Anzahl der in der Liste gespeicherten Elemente zurückgibt.

(b) Eine **Objektmethode** `fügeEin(...)`. Der Methode wird ein einzufügendes Element vom Typ `T` sowie die Position, an der das Element nach dem Einfügen stehen soll, übergeben. Für die Positionen gilt die übliche Annahme, dass bei 0 zu zählen begonnen wird.

Falls das Einfügen erfolgreich war, wird `true`, sonst `false` zurückgegeben.

Hinweis: Die Objekftfelder `erster` und `nächster` können zu jedem Zeitpunkt mit der Nullreferenz belegt sein.

(7 + 14 = 21 Punkte)