Name:	Peter Probe
Unterschrift:	

Fachrichtung:	Wirtschaftsinformatik
Kurs:	W WI 15 SEB
Studienhalbjahr:	1

Zur Vorlesung	Programmierung 1 (Systementwicklung)
Dozent:	
Datum:	
Bearbeitungszeit:	100 Minuten
Hilfsmittel:	keine

Aufgabenblock:	**1**	**2**	**3**	**4**	**5**	**6**	**Σ**
Punkte:	12	10	5	20	26	27	100
Erreicht:							

Weitere Hinweise:

- Bitte unterschreiben Sie die Klausur
- Bitte schreiben Sie leserlich
- Bitte unternehmen Sie keine Täuschungsversuche

Viel Erfolg !!!

Note:

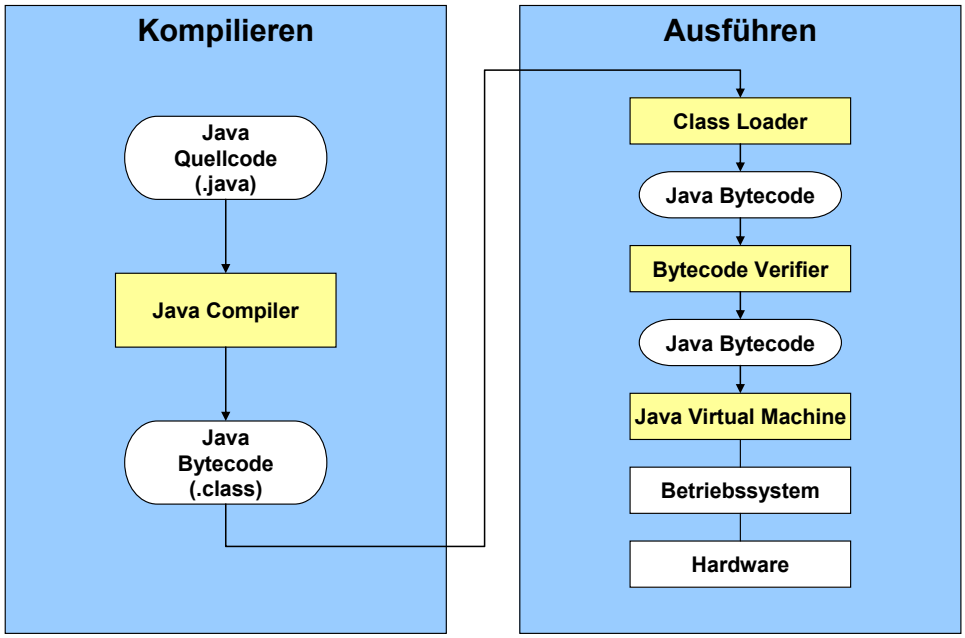
Aufgabenblock 1: Einführung

Aufgabe 1 (12 Punkte)
Beschreiben Sie die Darstellungsform „Pseudocode“!

- nahe an den Konstrukten verbreiteter Programmiersprachen
- Verwendung spezieller englischer Begriffe aus dem Alltag
- Begriffe haben eine festgelegte Bedeutung
- Sequenz
  - Alternative 1: Schritte werden durchnummeriert: 1, 2, 3, …
  - Alternative 2: Abschluss der Sequenz durch Semikolon
  - Vorteil Alternative 1: Verfeinerung einzelner Schritte: 2.1, 2.2, …
- Bedingte Anweisung
  - es werden Bedingungen auf Ihre Richtigkeit geprüft
  - Alternative 1: falls Bedingung dann Schritt
  - Alternative 2: falls Bedingung dann Schritt A sonst Schritt B
- Schleifen
  - kopfgesteuert: solange Bedingung wahr führe aus Schritte
  - fußgesteuert: wiederhole Schritte bis Bedingung wahr
  - Zählschleife: wiederhole für Zahlenbereich Arbeitsschritte

Aufgabenblock 2: Grundlagen von Java

Aufgabe 2 (7 Punkte)
Beschreiben Sie den Prozess von der Erstellung des Quellcodes bis zur Ausführung des Programms in der Programmiersprache Java mit Hilfe eines Diagramms!



Weiße Felder = ***0,5 P.***

Gelbe Felder = ***1 P.***

Aufgabe 3 (3 Punkte)
Wie wird in Java die Eigenschaft der Plattformunabhängigkeit sichergestellt?

- Compiler erstellt den Bytecode
- zu jedem Betriebssystem existiert eine JVM
- Bytecode wird von der jeweiligen JVM zur Laufzeit interpretiert

Aufgabenblock 3: Datentypen

Aufgabe 4 (5 Punkte)
Beschreiben Sie die wesentlichen Eigenschaften eines Arrays in der Programmiersprache Java!

- Arrays sind (mehrdimensionale) Feldvariablen, die aus mehreren Elementen bestehen
- alle Elemente eines Arrays gehören dem gleichen Datentyp an
- in Java sind Arrays semidynamisch, d.h. ihre Größe kann zur Laufzeit festgelegt, aber danach nicht mehr verändert werden
- Arrays in Java sind Objekte und bieten verschiedene Methoden
- die Elemente eines Arrays sind bei n Elementen von 0 bis n-1 durchnummeriert

Aufgabenblock 4: Ausdrücke und Anweisungen

Aufgabe 5 (6 Punkte)
Gegeben sind folgende Variablen mit ihren Werten:

```
short a = 11;
short b = 13;
short c = 5;
boolean z = false;
```

Füllen Sie folgende Wahrheitstabelle aus:

Ausdruck	**Wahrheitswert**
(a * c % b < 5)	True
(a < b) & (b > c)	True
(a < b) & (!(b > c) ^ !z)	True
((a >= b) | (b > c)) ^ !z	False
(++a % c * b) == (a-- + 15)	False
!(((-a == -b + b % a) && !z) ^ (3*c < 3f/2*b))	False

Aufgabe 6 (14 Punkte)
Folgende Ausgabe soll erzeugt werden:

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
```

Zeichnen Sie ein Struktogramm, das einen Algorithmus für diese Ausgabe abbildet.

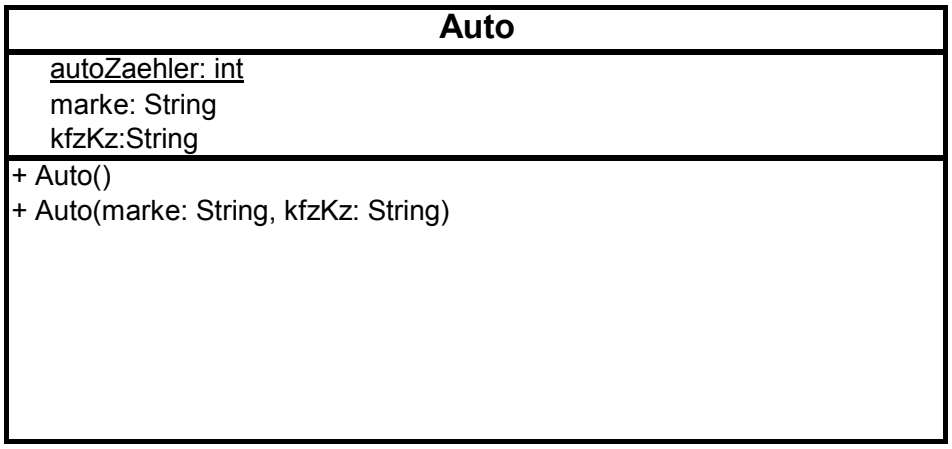
<table>
<tr><td colspan="3">Initialisiere ein zweidimensionales Array dreieck 1 P.</td></tr>
<tr><td colspan="3">gib die erste Dimension des Arrays mit 5 Zeilen vor 1 P.</td></tr>
<tr><td colspan="3">zaehle Zeilenindex i von 0 bis 4 2 P.</td></tr>
<tr><td></td><td colspan="2">gib die zweite Dimension des Arrays mit i+1 Spalten vor 2 P.</td></tr>
<tr><td></td><td colspan="2">zaehle Spaltenindex j von 0 bis i 2 P.</td></tr>
<tr><td></td><td colspan="2">j = 0 oder j = i 2 P.</td></tr>
<tr><td></td><td>setze dreieck[i][j] auf 1 1 P.</td><td>berechne dreieck[i][j] 1 P.</td></tr>
<tr><td></td><td colspan="2">gib dreieck[i][j] aus 1 P.</td></tr>
<tr><td></td><td colspan="2">erzeuge einen Zeilenumbruch 1 P.</td></tr>
</table>

Aufgabenblock 5: Objektorientierung

Aufgabe 7 (8 Punkte)
Worin unterscheiden sich Klassenattribute und –methoden von Instanzattributen und –methoden? Wie können Klassenmethoden auf den Instanzenkontext zugreifen? Beschreiben Sie zwei Lösungsansätze

- Klassenattribute und –methoden existieren unabhängig von einer Instanz einer Klasse
- ohne das ein Objekt existiert können diese verwendet werden
- sie werden durch den Modifier static als Klassenattribut bzw. –methode definiert
- Problem: static-Methoden können normalerweise nur auf static-Attribute zugreifen, und nicht auf Instanzattribute und -methoden
- Lösungen
    - Alternative 1
      Erzeugen einer statischen Referenzvariable, mit der auf den Instanzkontext zugegriffen werden kann
    - Alternative 2
      Erzeugen eine lokale Referenzvariable innerhalb einer statischen Methode, mit der auf den Instanzkontext zugegriffen werden kann

Aufgabe 8 (18 Punkte)
Gegeben ist folgendes UML-Diagramm:



a) Ergänzen Sie die Attribute im UML-Diagramm um den Sichtbarkeitsmodifier, um das Prinzip der Kapselung zu realisieren!
   - für den Modifier private ***2 P.***
b) Ergänzen sie das UML-Diagramm um die notwendigen Methoden, um das Prinzip der Kapselung zu vervollständigen!
   + getAutoZaehler(): int
   + setMarke(marke:String) :void
   + getMarke() :String
   + setKfzKz(kfzKz :String) :void
   + getKfzKz() :String
   Getter-/Setter-Methoden für die privaten Attribute ***2 P.***
c) Erweitern Sie das UML-Diagramm um den entsprechenden Destruktor der Klasse!
   # finalize(): void ***2 P.***
d) Implementieren Sie die Klasse Auto mit allen Attributen und Methoden (inkl. der von Ihnen ergänzten Methoden aus dem Teil b) der Aufgabe). Beim Erzeugen eines neuen Objektes soll das Attribut autoZaehler um 1 erhöht und beim Ausführen des Destruktors um 1 erniedrigt werden.

```
public class Auto {

1 P.  private static int autoZaehler = 0;
0,5 P.private String marke;
0,5 P.private String kfzKz;

      public Auto() {
1 P.        autoZaehler++;
      }

      public Auto(String marke, String kfzKz) {
1 P.        this();
0,5 P.      this.marke = marke;
0,5 P.      this.kfzKz = kfzKz;
      }

      public String getKfzKz() {
1 P.        return kfzKz;
      }

      public void setKfzKz(String kfzKz) {
1 P.        this.kfzKz = kfzKz;
      }

      public String getMarke() {
```

```
1 P.        return marke;
      }

      public void setMarke(String marke) {
1 P.        this.marke = marke;
      }

      public static int getAutoZaehler() {
1 P.        return autoZaehler;
      }

      protected void finalize() {
2 P.        autoZaehler--;
      }
}
```

Aufgabenblock 6: Vererbung

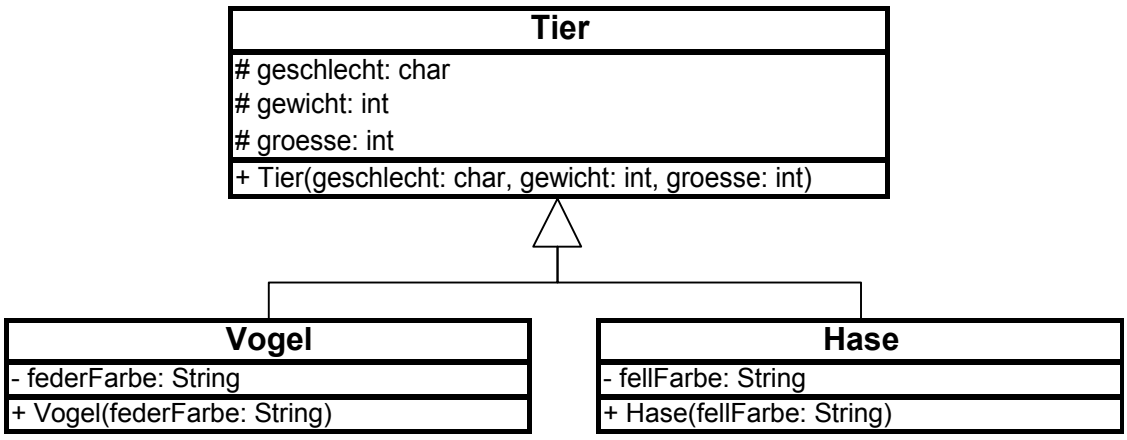
Aufgabe 9 (4 Punkte)

Beschreiben Sie das Prinzip des Überschreibens von Methoden! Wie wird entschieden, welche Methode zur Laufzeit ausgeführt wird?

- Methoden, die bereits in der Superklasse implementiert sind, können in Subklassen neu definiert werden (Überschreiben) ***2 P.***
- zur Laufzeit wird entschieden, welche Methode konkret ausgeführt wird (dynamisches Binden)
    - dabei sucht die JVM zunächst in der Klasse des Objektes nach einer passenden Methode ***1 P.***
    - wird dort keine passende Methode gefunden, wird die Vererbungshierarchie von unten nach oben nach einer passenden Methode durchsucht ***1 P.***

Aufgabe 10 (23 Punkte)

Gegeben ist das folgende Vererbungsdiagramm:



Sowie das folgende Klassendiagramm:

Käfig
- ccm: double - bewohner: Tier
+ Käfig(ccm: double) + setBewohner(bewohner: Tier) + getArtBewohner(): String

Folgende Aufgaben sind zu erledigen:

a) Implementieren Sie zunächst die Vererbungshierarchie. Rufen Sie dabei aus den Konstruktoren der Subklassen zunächst den Konstruktor der Super-Klasse

```
public class Tier {

0,5 P.protected char geschlecht;
0,5 P.protected int gewicht;
0,5 P.protected int groesse;

      public Tier(char geschlecht, int gewicht, int groesse) {
0,5 P.      this.geschlecht = geschlecht;
0,5 P.      this.gewicht = gewicht;
0,5 P.      this.groesse = groesse;
      }
}

public class Hase extends Tier {

0,5 P.private String fellFarbe;

      public Hase(String fellFarbe) {
1 P.        super('m', 3, 1);
0,5 P.      this.fellFarbe = fellFarbe;
      }
}

public class Vogel extends Tier {

0,5 P.private String federFarbe;

      public Vogel(String federFarbe){
1 P.        super('w', 1, 1);
0,5 P.      this.federFarbe = federFarbe;
      }
}
```

b) Implementieren Sie die Klasse Käfig. Die Methode getArtBewohner() soll die Art des Bewohners (Hase oder Vogel) als String zurückgeben.

```
public class Käfig {

0,5 P.private double ccm;
0,5 P.private Tier bewohner;

      public Käfig(double ccm) {
0,5 P.      this.ccm = ccm;
      }

1 P.  public void setBewohner(Tier bewohner) {
0,5 P.      this.bewohner = bewohner;
      }

1 P.  public String getArtBewohner() {
2 P.        if (bewohner instanceof Hase)
```

```
1 P.                return "Hase";
2 P.          if (bewohner instanceof Vogel)
1 P.                return "Vogel";
2 P.          return null;
        }
}
```

c) Implementieren Sie eine Testklasse KäfigTest, in der Sie
   1. ein Käfig-Objekt erzeugen
   2. wahlweise einen Bewohner (Hase oder Vogel) erzeugen
   3. dem Käfig-Objekt diesen Bewohner mit der Methode setBewohner() zuweisen
   4. die Art des Bewohners über die Methode getArtBewohner() herausfinden und auf der Konsole ausgeben

```
public class KäfigTest {

        public static void main(String[] args) {

1 P.          Käfig neuerKäfig = new Käfig(30.4);
1 P.          Hase bunny = new Hase("Grau");

1 P.          neuerKäfig.setBewohner(bunny);
1 P.          System.out.println(neuerKäfig.getArtBewohner());
        }
}
```