

*Name, Vorname:*

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Sichtbarkeits- und Zugriffsmodifizierer wie public, usw. dürfen nicht weggelassen werden.
- In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe- oder Eingabe Methoden.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

1) 4P  
Was ist der Unterschied zwischen einer Klasse und einem Objekt?

2) 10P  
a)

Welche Werte haben die folgenden Variablen in der Methode main ?(Begründen Sie)  
h1 , i , h2 , alter von h2

```
public class Startklasse{
    public static void main(String[] args) {
        Hund h1;                // <---
        int i;                   // <---
        Hund h2=new Hund ();     // <---
    }
}

class Hund{
    private int alter;
    public Hund(int pAlter){
    }
}
```

b)  
Würde eine Fehlermeldung erzeugt werden, wenn der Konstruktor in der Klasse Hund entfernt werden würde? Begründen Sie.

- 3) (Alle Teilaufgaben müssen in genau **einem** einzigen Programm realisiert werden) 37P
- a) 12P  
Erstellen Sie die Klasse Bruch, (mit genau den 2 Attributen zaehler und nenner, genau 2 set-Methoden, genau 2 get-Methoden und genau einem Konstruktor mit 2 Parametern), die eine mathematische Bruchzahl mit einem ganzzahligen Zähler und einem ganzzahligen Nenner repräsentieren soll.
- Zusätzlich sollen genau noch folgende Methoden (und nur diese) erzeugt werden:
- b) 3P  
... kuerzen(int teiler) :  
kürzt den Bruch durch den wert teiler  
Beispiel: 8 / 40 --- Teiler 4 --> 2 / 10
- c) 3P  
... alsKommazahl( ... ) :  
stellt den Bruch durch eine Kommazahl dar.  
Beispiel: 1/2 ---> 0.5
- d) 3P  
... void quadrieren( ) :  
bildet das Quadrat eines Bruchs.  
Beispiel: 2 / 3 ---> 4 / 9
- e) 5P  
... Bruch multiplizieren(Bruch bruch) :  
multipliziert 2 Brüche miteinander  
Beispiel: 3 / 4 \* 5 / 6 ---> 15 / 24  
Tipp:  $\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$
- f) 3P  
Legen Sie die Brüche b0 (Wert: 7/8) , b1 (Wert: 2/3) und b2 (Wert : 4/5) an.
- g) 3P  
Speichern Sie die Kommazahl (mit der entsprechenden Methode) von b0 in der Variablen wert ab.
- h) 2P  
Berechnen Sie (mit der entsprechenden Methode) das Quadrat von b0.  
Wo wird das Ergebnis gespeichert?
- i) 3P  
Berechnen Sie (mit den entsprechenden Methoden) b1 \* b2 und speichern Sie das Ergebnis in der Variablen b4 (mit dem Datentyp Bruch) ab.

Lösungen:

1) 4P  
Eine Klasse ist ein Bauplan, nachdem ein Objekt gebastelt und im Arbeitsspeicher angelegt wird.

2) 10P  
a) 8P  
h1 , i sind undefiniert (dem Programmierer unbekannt)  
h2 zeigt auf ein Objekt im Arbeitsspeicher, alter von h2 hat den Wert 0

b) 2P  
Nein, da der Konstruktor automatisch vom Compiler angelegt wird.

3)

```
package bruch2;
public class Bruch2 {
    public static void main(String[] args) {
        Bruch b0=new Bruch(7,8); // 1P
        Bruch b1=new Bruch(2,3); // 1P
        Bruch b2=new Bruch(4,5); // 1P
        Bruch b4;
        double wert;

        wert=b0.alsKommazahl(); // 3P
        b0.quadrieren(); // 2P
        b4=b1.multiplizieren(b2); // 3P
    }
}

class Bruch{
    private int zaehler; // 1P
    private int nenner; // 1P

    public Bruch(int zaehler, int nenner) { // 2P
        this.zaehler = zaehler;
        this.nenner = nenner;
    }

    public int getZaehler() { // 2P
        return zaehler;
    }

    public void setZaehler(int zaehler) { // 2P
        this.zaehler = zaehler;
    }

    public int getNenner() { // 2P
        return nenner;
    }

    public void setNenner(int nenner) { // 2P
        this.nenner = nenner;
    }
}
```

```

public void kuerzen(int teiler) {                                // 3P
    nenner = nenner/teiler;
    zaehler = zaehler/teiler;
}

public double alsKommazahl() {                                    // 3P
    return ((double) zaehler / (double) nenner);
}

public void quadrieren() {                                       // 3P
    zaehler=zaehler*zaehler;
    nenner = nenner * nenner;
}

public Bruch multiplizieren(Bruch bruch) {                       // 5P
    int z, n;
    z=zaehler*bruch.zaehler;
    n=nenner*bruch.nenner;
    Bruch ergBruch=new Bruch(z,n);
    return ergBruch;
}

public void printBruch() {
    System.out.print(zaehler+"/"+nenner);
}
}

```

## NACHTERMIN

x)

5P

gegeben sei folgender Programmausschnitt (Hund ist eine gegebene Klasse mit den entsprechenden Attributen und Methoden):

```
...  
Hund dopermann;  
dopermann.setGewicht(10);  
dopermann.bellen();  
...
```

- a) Warum gibt es während der Laufzeit eine Fehlermeldung?
- b) Welchen Wert hat dopermann?

X)

Signaturbeispiele machen  
UML erstellen lassen

Name, Vorname:

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Sichtbarkeits- und Zugriffsmodifizierer wie public, usw. dürfen nicht weggelassen werden.
- In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe- oder Eingabe Methoden.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punktabzug !!!!

## AUFGABEN

I) 52P

Bemerkung:

In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe-Methoden.

Alle Teilaufgaben müssen in **einem** Programm realisiert werden

1) 7P

Erzeugen Sie die Klasse Quadrat mit genau dem Attribut (und nur dem Attribut) "laenge", den zu " laenge " gehörigen get-und set-Methoden und einem Konstruktor (kein Standardkonstruktor).

2) 34P

Quadrate sollen zu statistischen Auswertungen analysiert werden (z.B. Output einer Maschine, die Abdeckungen mit quadratischer Grundfläche herstellt). Dazu wird ein Entwickler beauftragt die Klasse "**Verwaltung**" zu implementieren.

In der Klasse Verwaltung gibt es genau die 2 folgenden Attribute:

- Das Attribut (integer-Feld) "**quadrate**", in dem das jeweilige Quadrat abgespeichert wird.

Dieses Feld ist kein dynamisches Feld.

- Das Attribut "**gesamtLaenge**" , in dem die Länge des Feldes "quadrate" abgespeichert wird. (beim Anlegen eines Objekts der Klasse Verwaltung).

Die Klasse Verwaltung muß genau die folgenden Methoden besitzen:

a)

... Verwaltung( ... ) :

Konstruktor, der die Gesamtlänge des Feldes festlegt und Speicher für das Feld quadrate im Arbeitsspeicher reserviert.

b)

... setQuadrat(int i, Quadrat pQuadrat) :

belegt das Feld quadrate an der i-ten Zelle mit einem Quadrat

Dazu muß auch überprüft werden, ob sich die Zelle (deren Inhalt geändert werden soll) innerhalb des Feldes befindet.

c)

... getQuadrat( ... ) :

gibt einen Verweis auf das Feld "quadrate" zurück.

d)

... getQuadrat( ... ) :

gibt das Quadrat an einer Stelle i des Feldes zurück.

Dazu muß auch überprüft werden, ob sich die Zelle (deren Inhalt geändert werden soll) innerhalb des Feldes befindet.

e)

... updateQuadrat(int i, double laenge)

verändert an der Zelle i des Feldes die Länge des Quadrats auf einen neuen Wert.

Beachten Sie, daß an der Stelle i noch kein Quadrat sein kann (null) und sich die zu verändernde Zelle innerhalb der Feldgrenzen befinden muß.

f)

... ausgabe( ... )

Gibt alle Quadrate (genauer die Länge und die entsprechende Stelle im Feld) auf dem Bildschirm aus.

Beachten Sie, daß an bestimmten Stellen des Feldes noch kein Quadrate gespeichert sein können (null).

3)

11P

a)

Erzeugen Sie das Objekt "**verwalten**" der Klasse Verwaltung, so daß das Feld (Attribut) "quadrate" die Länge 10 hat.

b)

Fügen Sie im Feld "quadrate" (mit Hilfe der entsprechenden Methode der Klasse Verwaltung) in der 0-ten Zelle ein Quadrat der Länge 30 ein.

c)

Die Länge 30 dieses Quadrats soll nun (mit Hilfe der entsprechenden Methode der Klasse Verwaltung) auf die Länge 40 abgeändert werden.

d)

Das Quadrat in der 0-ten Zelle soll mit Hilfe einer Schleife und mit Hilfe der entsprechenden Methode der Klasse Verwaltung in die restlichen Zellen "kopiert" werden.

e)

Geben Sie die Länge aller Quadrate und deren Index (Stelle) im Feld - mit Hilfe der entsprechenden Methode der Klasse Verwaltung - auf dem Bildschirm aus.

Die Länge des Feldes "quadrate" muß mit Hilfe der entsprechenden Methoden ermittelt werden.

## Lösungen:

1)

7P

```
class Quadrat{
    private double laenge;

    public Quadrat(double laenge) {
        this.laenge = laenge;
    }

    public double getLaenge() {
        return laenge;
    }

    public void setLaenge(double laenge) {
        this.laenge = laenge;
    }
}
```

2)

30P

```
class Verwaltung{
    private int gesamtLaenge; //1P
    private Quadrat[] quadrate; //1P

    public Verwaltung(int gesamtLaenge){ //4P
        this.gesamtLaenge=gesamtLaenge;
        quadrate = new Quadrat[gesamtLaenge];
    }

    public void setQuadrat(int i, Quadrat pQuadrat){ //4P
        if(i>=0 && i<gesamtLaenge){
            quadrate[i]=pQuadrat;
        }
    }

    public Quadrat[] getQuadrat(){ //2P
        return quadrate;
    }

    public Quadrat getQuadrat(int i){ //6P
        if(i>=0 && i<gesamtLaenge){
            return quadrate[i];
        }
        else{
            return null;
        }
    }

    public void updateQuadrat(int i, double laenge){ //6P
        if(quadrate[i]!=null && i>=0 && i<gesamtLaenge){
            quadrate[i].setLaenge(laenge);
        }
    }
}
```



```

public void ausgabe() {                                     //6P
    int i;
    for(i=0;i<gesamtLaenge;i++){
        if(quadrate[i]!=null){
            System.out.println("Quadrat["+i+"]=
                                "+quadrate[i].getLaenge());
        }
    }
}
}

```

3) 15P

```

public class Startklasse_E2FI_13_1_2014_nr2 {
    public static void main(String[] args) {
        int i;
        Verwaltung verwaltung = new Verwaltung(10);         //2P
        verwaltung.setQuadrat(0, new Quadrat(30));          //4P
        verwaltung.updateQuadrat(0, 40);                    //2P
        for(i=1;i<verwaltung.getQuadrat().length;i++){      //5P
            verwaltung.setQuadrat(i, verwaltung.getQuadrat(0));
        }
        verwaltung.ausgabe();                                //2P
    }
}

```

Name, Vorname:

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Sichtbarkeits- und Zugriffsmodifizierer wie public, usw. dürfen nicht weggelassen werden.
- In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe- oder Eingabe Methoden.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punktabzug !!!!

## AUFGABEN

I) 50P  
1)

Ein Spielsüchtiger will auf Anraten seines Freundes seine Wetten verwalten. Dazu wird ein Entwickler beauftragt die Klasse "**Verwaltung**" zu implementieren.

In der Klasse Verwaltung gibt es das Attribut (integer-Feld) "**wetten**", in dem der Gewinn (bzw. Verlust) bei einer Wette abgespeichert wird. Dieses Feld ist kein dynamisches Feld.

Die Länge des Feldes wird in dem Attribut "**gesamtLaenge**" abgespeichert und beim Anlegen eines Objekts der Klasse Verwaltung festgelegt.


Das Ende des Feldes wetten wird immer größer, je mehr Wetten abgespeichert werden. Dieses Ende wird in dem integer-Attribut "**ende**" gespeichert und gibt den Index an, wo der nächste neue Wettgewinn abgespeichert wird.

Bemerkung:

Der Gewinn bzw. Verlust bei einer Wette wird mit Wettgewinn bezeichnet. Dieser Begriff wird auch benutzt, wenn die Wette mit einen Verlust endet (dann ist der Wettgewinn eben negativ).

Beispiel:

|    |     |    |  |  |  |
|----|-----|----|--|--|--|
| 10 | -20 | 30 |  |  |  |
|----|-----|----|--|--|--|



gesamtLaenge = 6

ende = 3

Gesamtgewinn =  $10 - 20 + 30 = 20$

Wenn nun ein neuer Wettgewinn (z.B. 100 Euro verloren) angefügt wird, bedeutet dies, daß daß die der mit dem Wert 30 folgenden Zelle mit dem -100 belegt wird. Insbesondere ändert sich dadurch der Wert des Attributs ende.

Erstellen Sie in der Klasse Verwaltung genau (und nur) folgende Methoden:

a) 6P

... Verwaltung( ... ) :

Konstruktor, der die Gesamtlänge des Feldes festlegt und Speicher für das Feld wetten im Arbeitsspeicher reserviert.

b) 6P

... anfüegen( ... ) :

fügt einen Wettgewinn an das Ende der bisherigen Wettgewinne an.

Wenn das Feld voll ist, werden keine neuen Wettgewinne angefügt.

c) 6P

... gibErgebnis ( ... ) :

gibt den Wettgewinn einer Wette zurück, der in einer bestimmten Zelle abgespeichert wurde.

Dazu muß auch überprüft werden, ob sich die Zelle (deren Inhalt geändert werden soll)

innerhalb der bis jetzt angelegten Wettgewinne befindet.

d) 6P

... aendern(...)

Falls ein falscher Wettgewinn in eine Zelle eingetragen wird, kann man diesen nachträglich

abändern. Dazu muß auch überprüft werden, ob sich die Zelle (deren Inhalt geändert werden

soll) innerhalb der bis jetzt angelegten Wettgewinne befindet.

e) 6P

... loeschen( ... )

Löscht den letzten Wettgewinn aus dem Feld wetten. Das bedeutet nicht, daß der letzte

Wettgewinn 0 wird, sondern daß der letzte Eintrag getilgt wird.

f) 7P

... bilanzieren( ... )

Bildet die Summe aller Wettgewinne und zeigt damit dem Spieler effektiv an, was er in

Wirklichkeit insgesamt gewonnen bzw. verloren hat.

g) 6P

... ausgabe( ... )

Gibt alle Wettgewinne auf dem Bildschirm aus.

2)

Realisieren Sie (mit Hilfe der entsprechenden Methode der Klasse Verwaltung) Folgendes in der Methode main(..) der Startklasse:

a) 1P

Erzeugen Sie das Objekt "**verwalten**" der Klasse Verwaltung, so daß das Feld (Attribut)

"wetten" die Länge 10 hat.

b) 4P

Fügen Sie im Feld "wetten" die Wettgewinne -100 , -200, +30, +1000 hintereinander ein.

c) 1P

Löschen Sie den letzten Wettgewinn aus der Wettgewinnliste.

d) 1P

Bilden Sie die Bilanz ihrer Wettgewinne.

## Lösungen:

```
class Verwaltung{
    private int ende;
    private int gesamtLaenge;
    private int[] wetten;
    // 6P
    public Verwaltung(int gesamtLaenge){
        this.gesamtLaenge=gesamtLaenge;
        ende=0;
        wetten = new int[gesamtLaenge];
    }
    // 6P
    public int gibErgebnis(int index){
        if(index >=0 && index < ende){
            return wetten[index];
        }
        else{
            return -1;
        }
    }
    // 6P
    public void anfüegen(int wert){
        if(ende<=gesamtLaenge-1){
            wetten[ende]=wert;
            ende++;
        }
    }
    // 6P
    public void ändern(int index, int wert){
        if(index>=0 && index < ende){
            wetten[index]=wert;
        }
    }
    // 6P
    public void löschen(){
        if(ende>0){
            ende--;
        }
    }
    // 7P
    public int bilanzieren(){
        int i;
        int summe=0;
        for(i=0;i<ende;i++){
            summe=summe+wetten[i];
        }
        return summe;
    }
    // 6P
    public void ausgabe(){
        int i;
        for(i=0;i<ende;i++){
            System.out.println("Wette["+i+"]="+wetten[i]);
        }
    }
    // 7P
}

public class Startklasse {
    public static void main(String[] args) {
        int i;
        Verwaltung verwaltung = new Verwaltung(10);
        verwaltung.anfüegen(-100);
        verwaltung.anfüegen(-200);
        verwaltung.anfüegen(+30);
        verwaltung.anfüegen(+1000);
        verwaltung.löschen();
        verwaltung.bilanzieren();
    }
}
```

Name, Vorname:

Hilfsmittel:  
keine

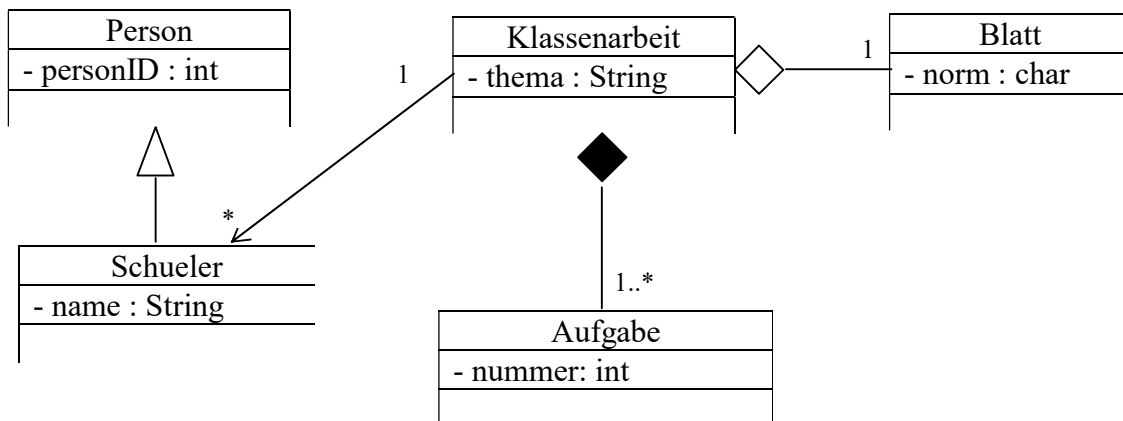
Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Sichtbarkeits- und Zugriffsmodifizierer wie public, usw. dürfen nicht weggelassen werden.
- In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe- oder Eingabe Methoden.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

I) 50P

Gegeben ist das folgende unvollständige UML-Diagramm (nur die Attribute der Klasse Klassenarbeit sind unvollständig). In allen Klassen fehlen die entsprechenden Methoden.



Bemerkung:

In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe-Methoden.

Alle Teilaufgaben müssen in **einem** Programm realisiert werden

- 1) 4P  
Was bedeuten die 2 verschiedenen Pfeile (offene bzw. geschlossene Pfeilspitze) und die Rauten (weiß und schwarz). Jeweils nur ein Stichwort angeben.
- 2) 7P  
Erzeugen Sie die Klasse Blatt mit genau dem im UML angegebenen Attribut (und nur diesem Attribut) und den dazugehörigen get-und set-Methoden und genau einem Konstruktor (mit genau einem Parameter).
- 3) 7P  
Erzeugen Sie die Klasse "Person" mit genau dem im UML angegebenen Attribut (und nur diesem Attribut) und den dazugehörigen get-und set-Methoden und genau einem Konstruktor (mit genau einem Parameter).
- 4) 9P  
Erzeugen Sie die Klasse "Schueler" mit genau dem im UML angegebenen Attribut (und nur diesem Attribut) und den dazugehörigen get-und set-Methoden und genau einem Konstruktor (mit genau zwei Parametern).
- 5) 7P  
Erzeugen Sie die Klasse "Aufgabe" mit genau dem im UML angegebenen Attribut (und nur diesem Attribut) und den dazugehörigen get-und set-Methoden und genau einem Konstruktor (mit genau einem Parameter).
- 6) 4P  
a) Erzeugen Sie die Klasse "Klassenarbeit" mit allen nötigen (genau) 4 Attributen.
- b) 4P  
Erzeugen Sie genau einen Konstruktor (mit genau zwei Parametern). Falls ein Array benötigt wird, soll dieses nicht dynamisch sein und im Konstruktor das Feld die Länge 10 erhalten).
- c) 4P  
Erzeugen Sie die folgende Methode:  
... insertAufgabe( ... ) :  
fügt eine Aufgabe an eine bestimmte Stelle des Feldes ein.  
Beachten Sie dazu, daß eine Komposition und eine Aggregation anders implementiert werden.
- d) 4P  
Erzeugen Sie die folgende Methode:  
... insertBlatt ( ... ) :  
fügt ein Blatt ein.  
Beachten Sie dazu, daß eine Komposition und eine Aggregation anders implementiert werden.

## Lösungen:

1) 4P  
geschlossene Pfeilspitze: Vererbung, geöffnete Pfeilspitze: Assoziation,  
schwarze Raute: Komposition, weiße Raute: Aggregation

2) 7P

```
class Blatt {
    private char norm;

    public Blatt(char pNorm) {
        norm = pNorm;
    }

    public char getNorm() {
        return norm;
    }

    public void setNorm(char norm) {
        this.norm = norm;
    }
}
```

3) 7P

```
class Person{
    private int personID;

    public Person(int personID) {
        this.personID = personID;
    }

    public int getPersonID() {
        return personID;
    }

    public void setPersonID(int personID) {
        this.personID = personID;
    }
}
```

4) 9P

```
class Schueler extends Person {
    private String name;

    public Schueler(int personID, String pName){
        super(personID);
        name=pName;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

5) 7P

```
class Aufgabe{
    private int nummer;

    public int getNummer() {
        return nummer;
    }

    public void setNummer(int nummer) {
        this.nummer = nummer;
    }

    public Aufgabe(int pNummer){
        nummer=pNummer;
    }
}
```

6)

```
class Klassenarbeit{
    private String thema;
    private Aufgabe[] dieAufgaben;
    private Schueler[] dieSchueler;
    private Blatt blatt;

    public Klassenarbeit(String pThema, Blatt pBlatt){
        thema=pThema;
        blatt=pBlatt;
        dieAufgaben=new Aufgabe[10];
        dieSchueler=new Schueler[10];
    }

    public void insertAufgabe(int pNummer, int index){
        dieAufgaben[index]=new Aufgabe(pNummer);
    }

    public void insertSchuler(Schueler pSchueler, int index){
        dieSchueler[index]=pSchueler;
    }

    public void insertBlatt(Blatt pBlatt ){
        blatt=pBlatt;
    }
    ...
}
```



*Name, Vorname:*

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Sichtbarkeits- und Zugriffsmodifizierer wie public, usw. dürfen nicht weggelassen werden.
- In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe- oder Eingabe Methoden.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punktabzug !!!!

## AUFGABEN

- I) 51P  
Die Verwaltung eines Bauernhofs soll mit Hilfe der OOP auf EDV umgestellt werden.  
In einem Gespräch zwischen der EDV-Firma „Makall“ und den Besitzern des Bauernhofs wurde folgendes festgehalten:  
Der Bauernhof gehört genau 2 Besitzern d.h. Personen und besteht aus genau einem Bauernhaus und einem Kuhstall, in dem maximal 10 Kühe untergebracht werden können..  
Die Klasse Kuh wurde schon implementiert (mit den Attributen name und alter und den entsprechenden Methoden).  
Aus Gründen der Vereinfachung besitzen die Klassen "Besitzer", "Person" und "Bauernhaus" jeweils genau ein Attribut
- 1) 16P  
Erstellen Sie ein UML-Diagramm (mit den entsprechenden Attributen, aber ohne Methoden), das diesen Fall modelliert.
- 2) 7P  
Implementieren Sie die Klasse "Person" mit den üblichen Attributen und Methoden.
- 3) 9P  
Implementieren Sie die Klasse "Besitzer" mit den üblichen Attributen und Methoden.
- 4) 4P  
a)  
Erzeugen Sie die Klasse "Bauernhof" mit allen nötigen Attributen.
- b) 5P  
Erzeugen Sie genau einen Konstruktor (mit genau 4 Parametern). Falls ein Array benötigt wird, soll dieses nicht dynamisch sein und die Länge 10 haben.

c)

10P

Erstellen Sie in der Klasse Bauernhof eine Methode

... anfüegenKuh (Kuh pKuh) :

fügt eine Kuh an das Ende der bisherigen gespeicherten Kühe an.

Wenn das Feld voll ist, werden keine neuen Kühe angefügt.

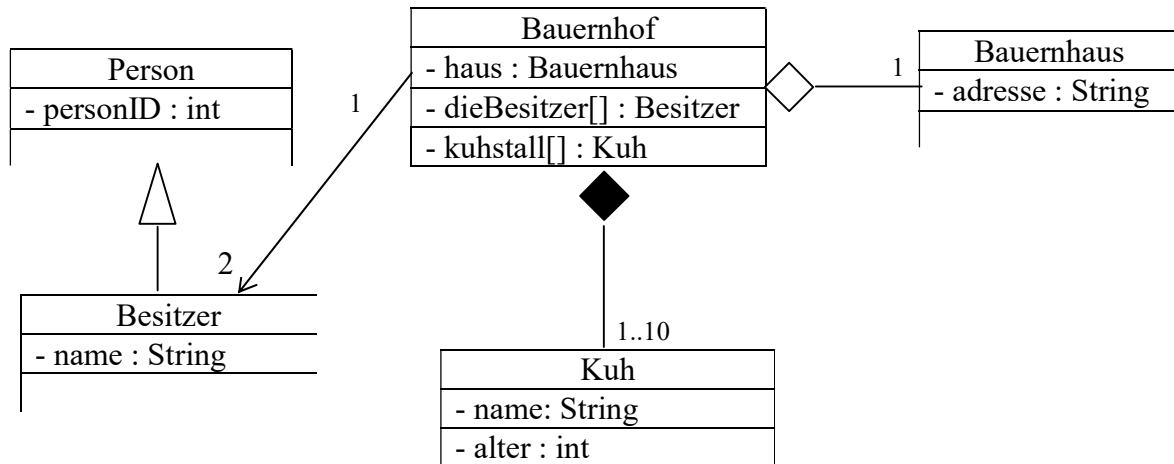
Bemerkung:

In allen Klassen dürfen in keiner Methode Ein- oder Ausgaben (auf dem Bildschirm) vorkommen, außer es handelt sich um reine Ausgabe-Methoden.

Alle Teilaufgaben - sofern es sich um Programmieraufgaben handelt - müssen in **einem** Programm realisiert werden

Lösung:

1)



2)

7P

```
class Person{
    private int personID;

    public Person(int personID) {
        this.personID = personID;
    }

    public int getPersonID() {
        return personID;
    }

    public void setPersonID(int personID) {
        this.personID = personID;
    }
}
```

3)

9P

```
class Besitzer extends Person {
    private String name;

    public Besitzer(int personID, String pName){
        super(personID);
        name=pName;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

4)

```
class Bauernhof{
    private String adresse;
    private Kuh[] kuhstall;
    private Besitzer[] dieBesitzer;
    private Bauernhaus haus;

    public Bauernhof(String adresse, Besitzer b1, Besitzer b2,
                      Bauernhaus haus){
        this.adresse = adresse;
        dieBesitzer=new Besitzer[2];
        dieBesitzer[0]=b1;
        dieBesitzer[1]=b2;
        this.haus=haus;
        kuhstall = new Kuh[10];
    }

    public void anfuegenKuh(Kuh k){
        for(int i=0;i<10;i++){
            if(kuhstall[i]==null){
                kuhstall[i]=k;
                return;
            }
        }
    }

    public void printKuhstall(){
        for(int i=0;i<10;i++){
            if(kuhstall[i]!=null){
                sout ("Kuhname="+kuhstall[i].getName());
                sout ("Kuhgewicht="+kuhstall[i].getGewicht());
            }
        }
    }
}
```