

*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

Bemerkungen:

Ein Java-Programm muss nach dem Prinzip der OOP gestaltet werden.

Insbeondere müssen dazu - falls möglich - verschiedene Konzepte wie Klassen, Vererbung, Polymorphie, Interface, usw. verwendet werden.

Es wird auch bewertet, wie gut dieses Prinzip der OOP umgesetzt wurde.

Hier speziell: Kein instanceof benutzen

1)

104P

Zur Information:

Frauen, Hündinnen und Männer sind Lebewesen. Hündinnen und Frauen können schwanger werden und müssen deshalb eine Fruchtwasseruntersuchung machen.

Die Klasse Hündin hat genau (d.h. keine weiteren) das Attribut "gewicht".

Die Klasse Frau hat genau das Attribut "alter".

Die Klasse Mann hat genau das Attribut "iq" (bedeutet Intelligenzquotient).

Die Klasse Lebewesen hat genau das Attribut "name".

Das Programm soll multipersonal entwickelt werden: Zu Testzwecken sollen die Entwickler (also diejenigen, die heute diese Klassenarbeit schreiben!) der einzelnen Klassen gezwungen werden, die Methode

`int getAttraktivität()`

zu entwickeln, die die "Attraktivität" eines Lebewesens berechnet.

Diese muß (bitte nur als Modell auffassen, hat nichts mit der Realität zu tun) wie folgt berechnet werden:

Bei einer Hündin:  $2 * \text{gewicht} + 50$

Bei einer Frau:  $2 * \text{alter}$

Beim Mann:  $100 - \text{iq}$

Die Dringlichkeit einer Fruchtwasseruntersuchung muß in der Methode

`int getfwUntersuchung()`

wie folgt berechnet werden:

Bei einer Hündin:  $\text{gewicht} - 10$

Bei einer Frau:  $\text{alter} - 35$

- a) 12P  
Erstellen Sie dazu ein UML-Diagramm, in dem nur die Klassennamen und alle Attribute vorkommen (und die nicht durch andere ergänzt werden dürfen).  
Alle zugehörigen Methoden müssen dargestellt werden (Ausnahme: get- und set-Methoden).  
Bitte vor Erstellung des UML-Diagramms die Klassenarbeit vollständig durchlesen.
- b) 49P  
Implementieren Sie dazu den nötigen Quellcode (mit korrekter Syntax, so daß das Programm lauffähig ist). Die entsprechenden Konstruktoren müssen erstellt werden.  
Es dürfen keine weiteren Attribute eingefügt werden.  
Die Konstruktoren müssen jeweils Parameter enthalten.  
Keine set- und get-Methoden implementieren. Diese werden als implementiert angenommen.
- c) 15P  
Erstellen Sie die Klasse "Familie" mit genau den Attributen "name" und "mitglieder"  
(Ein Feld der Länge 10 in dem Familienmitglieder abgespeichert werden können).  
c1) Erzeugen Sie einen Konstruktor mit genau 2 Parametern.  
c2) Erzeugen Sie die Methode add(...) mit der man an die i-te Stelle des Feldes ein Lebewesen abspeichern kann.  
c3) Implementieren Sie dort zusätzlich die Methode:  
... attraktivVergleich(...)  
Diese berechnet von 2 Lebewesen dasjenige mit der höheren Attraktivität und gibt dieses zurück.
- d) 28P  
Machen Sie folgendes in der Methode main()  
d1) Erzeugen Sie folgende Objekte:  
Hündin h1: (Name: Luna, Gewicht: 10),  
Frau f1: (Name: berta, Alter: 50),  
Mann m1: (Name: Hohlger, iq: 60),  
d2) Speichern Sie diese Lebewesen in dem Feld "lebewesen".  
d3) Berechnen Sie mit Hilfe einer Schleife und der Methode getAttraktivität()  
die Attraktivität der jeweiligen Lebewesen und geben diese auf dem Bildschirm aus.  
d4) In dem Feld "schwangereLW" (schwangere Lebewesen) müssen die im vorigen Programmteil erstellte Hündin und Frau abgespeichert werden und dann mit Hilfe einer Schleife und der Methode getfwUntersuchung() die Dringlichkeit einer Fruchtwasseruntersuchung des jeweiligen Lebewesens berechnet und auf dem Bildschirm ausgegeben werden.  
d5) Erzeugen Sie eine Familie (Variablenname: familie) mit dem Namen "lowinzig"  
Berechnen Sie, wer attraktiver ist: m1 oder h1.  
Speichern Sie das Ergebnis in der Variablen tempLebewesen ab.  
d6) h1 muß Mitglied der Familie "lowinzig" werden.

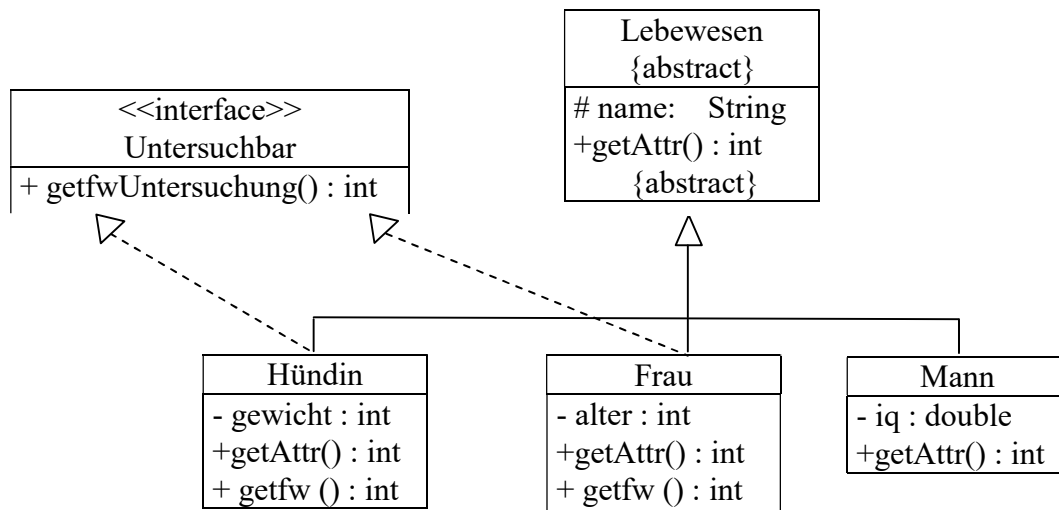
Lösung:

a)

12P

Bemerkung: Um Platz zu sparen wurde abgekürzt:

getfwUntersuchung durch getfw und getAttraktivität durch getAttr



b)

49P

```
// Interface erstellen 5P
public interface Untersuchbar {
    int getfwUntersuchung();
}

// 9P
abstract class Lebewesen{
    private String name;
    public Lebewesen(String name) {
        this.name = name;
    }

    public abstract int getAttraktivität();
}

// 13P
class Hündin extends Lebewesen implements Untersuchbar{
    private int gewicht;

    public int getGewicht() {
        return gewicht;
    }

    public void setGewicht(int gewicht) {
        this.gewicht = gewicht;
    }

    public Hündin(int gewicht, String name) {
        super(name);
        this.gewicht = gewicht;
    }

    public int getfwUntersuchung(){
        return gewicht-10;
    }

    public int getAttraktivität(){
        return (2*gewicht+50);
    }
}
```

```
// 13P
class Frau extends Lebewesen implements Untersuchbar{
    private int alter;

    public Frau(int alter, String name) {
        super(name);
        this.alter = alter;
    }

    public int getfwUntersuchung(){
        return alter-35;
    }

    public int getAttraktivität(){
        return (2*alter);
    }
}
```

```
// 9P
class Mann extends Lebewesen{
    private int iq;

    public Mann(int iq, String name) {
        super(name);
        this.iq = iq;
    }

    public int getAttraktivität(){
        return (100-iq);
    }
}
```

c)

15P

```
class Familie{
    private String name;
    private Lebewesen [] mitglieder;

    public Familie (String name){
        mitglieder = new Lebewesen[10];
        this.name=name;
    }

    public void add(Lebewesen l, int i){
        mitglieder[i]=l;
    }

    public Lebewesen attraktivVergleich(Lebewesen l1, Lebewesen l2){
        if(l1.getAttraktivität()<l2.getAttraktivität()){
            return l2;
        }
        else{
            return l1;
        }
    }
}
```

d)

28P

```
public class Startklasse {
    public static void main(String[] args) {
        int i;
        Hündin h1=new Hündin(10,"Luna");           // 2P
        Frau f1=new Frau(50,"Berta");              // 2P
        Mann m1=new Mann(60,"Hohlger");            // 2P
        Lebewesen [] lebewesen = new Lebewesen[3]; // 2P
        lebewesen[0]=h1;                           // 1P
        lebewesen[1]=f1;                           // 1P
        lebewesen[2]=m1;                           // 1P
        Untersuchbar [] schwangereLW = new Untersuchbar [2]; // 2P
        schwangereLW[0]=h1;                        // 1P
        schwangereLW[1]=f1;                        // 1P

        for(i=0;i<3;i++){                          // 3P
            System.out.println("Attraktionswert="
                               +lebewesen[i].getAttraktivität());
        }

        for(i=0;i<2;i++){                          // 3P
            System.out.println("Dringlichkeit="
                               +schwangereLW[i].getfwUntersuchung());
        }

        Familie familie = new Familie("lowinzig"); // 2P
        Lebewesen tempLebewesen;                  // 1P
        familie.add(h1, 0);                        // 2P
        tempLebewesen =familie.attraktivVergleich(m1, h1); // 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 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 zurückgegeben 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 Punktabzug !!!!

## AUFGABEN

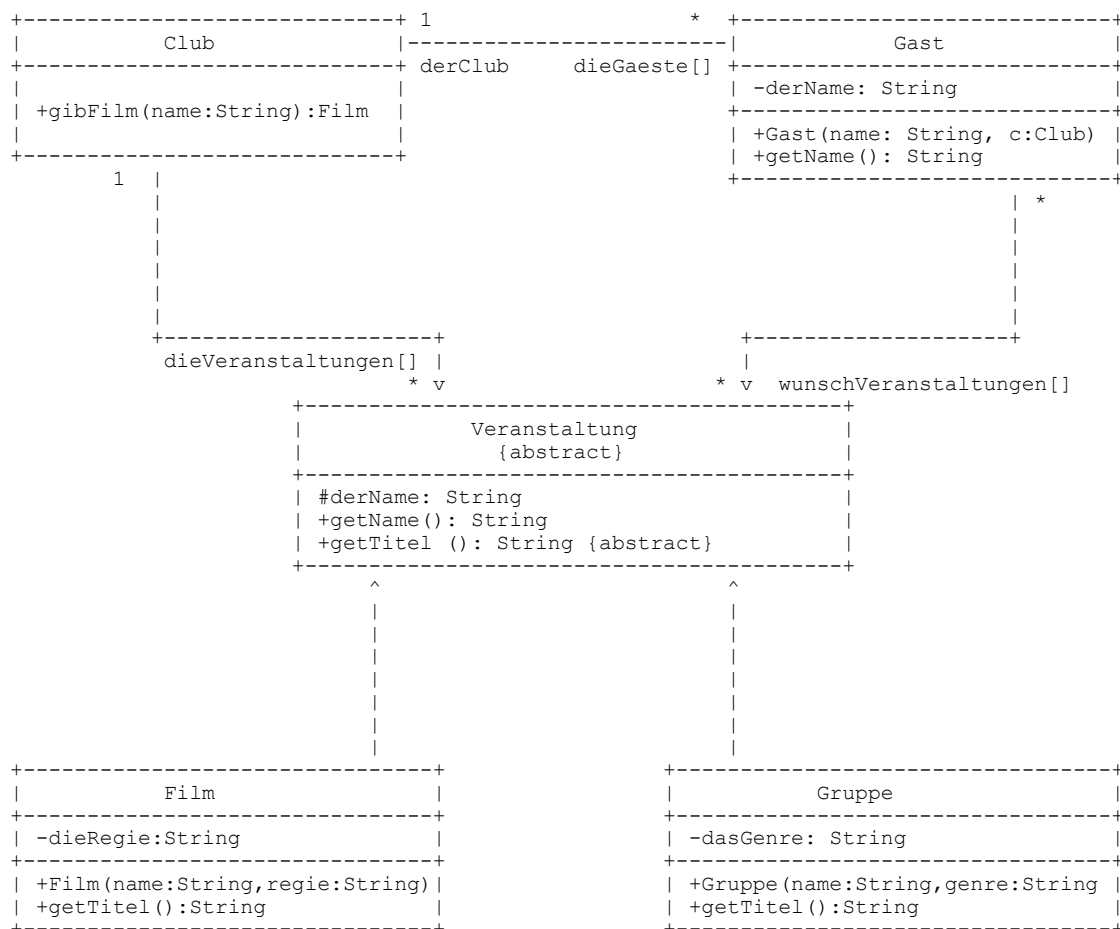
Um die Veranstaltungen besser auf die Gäste zuschneiden zu können, wird ein intelligentes Planungsmodul entwickelt.

Eine Klasse Club (Steuerungsschicht) verwaltet die Gäste, Filme, Musik- bzw.

Theatergruppen. Gäste können sich Filme und Gruppen wünschen.

Hier ein Vorentwurf des Klassendiagramms.

Wichtig: Es sind nicht alle Attribute und Methoden angegeben!



Beispiel:

dieGaeste[0] verweist auf das 1. Gastobjekt. Die Anzahl der verwalteten Gastobjekte kann durch dieGaeste.length ermittelt werden. Jeder Gast hat eine Wunschliste bzgl. der Veranstaltungen.

Bemerkung:

Wenn Sie Felder verwenden, benutzen Sie nur die Klasse ArrayList.

1) 9P

Erstellen Sie die abstrakte Klasse "Veranstaltung" mit genau einem Attribut und der zugehörigen get- und set-Methode.

Erstellen Sie genau einen Konstruktor mit genau einem Parameter.

Erstellen Sie die abstrakte Methode getTitel().

Bemerkung zu 2) und 3):

Die Methode Film.getTitel(): String gibt "<derName>, Regie: <dieRegie>" zurück, z.B: "Der kalte Winter, Regie: Leo Lego".

Die Methode Gruppe.getTitel(): String gibt "<derName>, <dasGenre>" zurück, z.B: "Sex Pistols, Punk".

2) 9P

Erstellen Sie die Klasse "Film" mit genau einem Attribut und der zugehörigen get-Methode.

Erstellen Sie genau einen Konstruktor mit genau zwei Parametern.

Erstellen Sie die Methode getTitel().

3) 9P

Erstellen Sie die Klasse "Gruppe" mit genau einem Attribut und der zugehörigen get-Methode.

Erstellen Sie genau einen Konstruktor mit genau zwei Parametern.

Erstellen Sie die Methode getTitel().

4) 17P

Erstellen Sie die Klasse Club mit genau 3 Attributen und den 3 zugehörigen get-Methoden.

Erstellen Sie genau einen Konstruktor mit genau einem Parameter.

Erstellen Sie zusätzlich noch die folgende Methode:

gibVeranstaltung (name: String) : Veranstaltung

Sie gibt den Verweis auf ein Veranstaltungsobjekt zurück, dessen Attribut derName mit dem Parameter name übereinstimmt. Wenn noch kein entsprechendes Objekt existiert, erzeugt sie ein neues Filmobjekt.

Implementieren Sie die Methode.

5) 10P

Machen Sie folgendes in der Methode main()

a) Erzeugen Sie das Objekt myClub, das einen Club mit dem Namen "Freizeit-Club" darstellt.

b) Erstellen Sie das Objekt myGast1 (Name: Alfred und Alfred ist Gast im "Freizeit-Club").

c) Erstellen Sie das Objekt myGruppe1 (Name: "TonSteineScherben", Genre: "Rockmusik").

d) Speichern Sie myGast1 in der Gästeliste vom "Freizeit-Club".

e) Speichern Sie myGruppe1 in der Veranstaltungsliste vom "Freizeit-Club".

f) Speichern Sie in den Wunschveranstaltungen von Alfred das 0. Element der Veranstaltungsliste vom "Freizeit-Club", wobei nicht vorausgesetzt werden darf, daß man weiß, welche Veranstaltung sich aktuell dort befindet.

Lösung:

1)

9P

```
abstract class Veranstaltung{
    protected String derName;

    public Veranstaltung(String pDerName){
        derName = pDerName;
    }

    public String getName(){
        return(derName);
    }

    public void setName(String pName){
        derName=pName;
    }

    abstract public String getTitel();
}
```

2)

9P

```
class Film extends Veranstaltung{
    private String dieRegie;

    public Film(String pDerName, String pDieRegie){
        super(pDerName);
        dieRegie = pDieRegie;
    }

    public String getRegie(){
        return(dieRegie);
    }

    public String getTitel(){
        return(getName()+" , "+getRegie());
    }
}
```

3)

9P

```
class Gruppe extends Veranstaltung{
    private String dasGenre;

    public Gruppe(String pDerName, String pDasGenre){
        super(pDerName);
        dasGenre = pDasGenre;
    }

    public String getGenre(){
        return(dasGenre);
    }

    public String getTitel(){
        return(getName()+" , "+getGenre());
    }
}
```

4)

17P

```

class Club{
    private String derName;
    private ArrayList <Gast> dieGaeste;
    private ArrayList <Veranstaltung> dieVeranstaltungen;

    public Club(String pDerName){
        derName = pDerName;
        dieGaeste = new ArrayList <Gast>();
        dieVeranstaltungen = new ArrayList <Veranstaltung>();
    }

    public String getName(){
        return(derName);
    }

    public ArrayList <Gast> getGaesteListe(){
        return(dieGaeste);
    }

    public ArrayList <Veranstaltung> getVeranstaltungListe(){
        return(dieVeranstaltungen);
    }

    public Veranstaltung gibVeranstaltung(String name){
        int i=0;
        while(i<dieVeranstaltungen.size()){
            if(dieVeranstaltungen.get(i).
                getName().equals(name)){
                return dieVeranstaltungen.get(i);
            }
            i++;
        }
        Film f=new Film(name,"");
        dieVeranstaltungen.add(f);
        return f;
    }
}

```

5)

10P

```

Club myClub1=new Club("Freizeit-Club");
Gast myGast1 = new Gast("Alfred", myClub1);
Gruppe myGruppel = new Gruppe("TonSteineScherben",
                                "Rockmusik");

myClub1.getGaesteListe().add(myGast1);
myClub1.getVeranstaltungListe().add(myGruppel);
myGast1.getWunschVeranstaltungListe().
    add(myClub1.getVeranstaltungListe().get(0));

```

*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:

Alle folgenden Teilaufgaben müssen (soweit sie die Programmierung betreffen) in genau **einem** einzigen Programm realisiert werden.

Zur Information:

Eine zweispurige Strasse wurde auf einer Teilstrecke von einem Kilometer Länge durch einen Bergrutsch so beeinträchtigt, daß nur noch eine Spur zu befahren ist. Um den Verkehr zu steuern wurden 2 Ampeln aufgestellt, so daß die Teilstrecke abwechselnd immer wieder in eine Richtung und dann in die andere Richtung befahren werden kann.

Wenn die Ampelanlage gestartet wird, ist jedem Zeitpunkt auf jeder Ampel genau eine der 3 Farben rot, gelb oder grün zu sehen

Jede Ampel hat die 4 Zustände 1 (rot) , 2 (grün) , 12 (rot-gelb) 21 (grün-gelb), die bekanntermaßen nach folgender Vorschrift auf einander folgen:

1 (rot) --> 12 (rot-gelb) --> 2 (grün) --> 21 (grün-gelb) --> 1 (rot) --> usw.

Außerdem hat jede Ampel 3 Lampen, von denen jede die 4 Farben rot, gelb, grün oder schwarz annehmen kann, wobei schwarz eine ausgeschaltete Lampe bedeutet.

Der Zustand 1 bedeutet: rote Lampe an, restliche Lampen schwarz.

Der Zustand 2 bedeutet: grüne Lampe an, restliche Lampen schwarz.

Der Zustand 12 bedeutet: gelbe Lampe an, restliche Lampen schwarz.

Der Zustand 21 bedeutet: gelbe Lampe an, restliche Lampen schwarz.

Es gibt die 3 Klassen "Lampe", "Ampel" und "Steuerung".

1) 9P  
Implementieren Sie die Klasse "Lampe" mit genau (und nur) dem Attribut farbe und dem Datentyp String. Zusätzlich müssen noch folgende Methoden erzeugt werden:

- a) ... getFarbe(...) :  
gibt die Farbe einer Lampe zurück.
- b) ... einschalten(...) :  
setzt eine Lampe auf eine bestimmte Farbe.
- c) ... ausschalten(...) :  
schaltet eine Lampe aus
- d) einen Konstruktor der Klasse

2) 32P  
Die Klasse Ampel hat als Attribute genau 3 Lampen und das Attribut "zustand". Implementieren Sie die Klasse "Ampel" mit genau (und nur) diesen Attributen. Zusätzlich müssen noch folgende Methoden erzeugt werden:

- a) ...setZustand(...) :  
Beachten Sie, daß beim Setzen eines Zustand zusätzlich noch die zugehörige Farbe der entsprechenden Lampe eingeschaltet werden muß.
- b) ... getZustand(...) :  
gibt den Zustand der Ampel zurück
- c) ...rotSchalten(...) :  
schaltet die rote Lampe auf rot und die anderen aus.
- d) ... gelbSchalten(...) :  
entsprechendes für die gelbe Lampe.
- e) ... grünSchalten(...) :  
entsprechendes für die grüne Lampe.
- f) ... weiterSchalten(...) :  
schaltet eine Ampel in den nächsten Zustand.
- g) einen Konstruktor der Klasse

3) 10P  
Die Klasse Steuerung hat als Attribute genau 2 Ampeln. Implementieren Sie die Klasse "Steuerung" mit genau (und nur) diesen Attributen. Zusätzlich müssen noch folgende Methoden erzeugt werden:

- a) einen Konstruktor der Klasse
- b) ...start(...) :  
beginnt den Steuerungsvorgang:  
Ampel1 rot, Ampel2 grün --> Ampel1 gelb, Ampel2 gelb --> Ampel1 grün, Ampel2 rot --> Ampel1 gelb, Ampel2 gelb ---> usw.

Lösung:

```
class Lampe{ // 9P
    private String farbe; // 1P

    public String getFarbe(){ // 2P
        return farbe;
    }

    public void einschalten(String pFarbe){ // 2P
        farbe=pFarbe;
    }

    public void ausschalten(){ // 2P
        farbe="schwarz";
    }

    public Lampe(){ // 2P
        farbe="schwarz";
    }
}

class Ampel{ // 32P
    private Lampe grünLampe; // 1P
    private Lampe gelbLampe; // 1P
    private Lampe rotLampe; // 1P
    private int zustand; // 1P

    public Ampel(){ // 3P
        grünLampe = new Lampe();
        gelbLampe = new Lampe();
        rotLampe = new Lampe();
    }

    public void setZustand(int pZustand){ // 8P
        if(pZustand == 1){ // rot
            rotSchalten();
        }
        else if(pZustand == 12){ // rot-gelb
            gelbSchalten();
        }
        else if(pZustand == 2){ // grün
            grünSchalten();
        }
        else if(pZustand == 21){
            gelbSchalten();
        }
        else { // Programmierfehler
            zustand=-1;
            return;
        }
        zustand=pZustand;
    }
}
```

```

public int getZustand(){                                // 2P
    return zustand;
}

private void rotSchalten(){                             // 3P
    grünLampe.ausschalten();
    gelbLampe.ausschalten();
    rotLampe.einschalten("rot");
}

private void gelbSchalten(){                           // 3P
    grünLampe.ausschalten();
    gelbLampe.einschalten("gelb");
    rotLampe.ausschalten();
}

private void grünSchalten(){                          // 3P
    grünLampe.einschalten("grün");
    gelbLampe.ausschalten();
    rotLampe.ausschalten();
}

public void weiterSchalten(){                         // 6P
    if(zustand == 1){
        setZustand(12);
    }
    else if(zustand == 12){
        setZustand(2);
    }
    else if(zustand == 2){
        setZustand(21);
    }
    else if(zustand == 21){
        setZustand(1);
    }
    else { // Programmierfehler
        zustand=-1;
    }
}
}

```



```
class Steuerung{ // 10P
    private Ampel ampel1; // 1P
    private Ampel ampel2; // 1P

    public Steuerung(){ // 2P
        ampel1 = new Ampel();
        ampel2 = new Ampel();
    }

    public void start(){ // 6P
        ampel1.setZustand(1);
        ampel2.setZustand(2);

        while (true){
            ampel1.weiterSchalten();
            ampel2.weiterSchalten();
        }
    }

    public void stopp(){
        ampel1.setZustand(1);
        ampel2.setZustand(1);
    }
}
```