

*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

1) 2P

Welche Zuweisungen sind korrekt bzw. nicht korrekt ?

- a) "Objekt einer Unterklasse" = "Objekt der Oberklasse"
- b) "Objekt einer Oberklasse" = "Objekt der Unterklasse"

2)

Herr Ripig will für seinen Fuhrpark die Jahressteuer seiner einzelnen Fahrzeuge berechnen. Dazu werden die Klassen "Auto" und "Lkw" erstellt.

Die jährliche Steuer berechnet sich wie folgt:

Jahressteuer Auto = 2 \* Hubraum Auto

Jahressteuer Lkw = 3 \* Leistung Lkw

a) 20P

Implementieren Sie die Klassen "Auto" mit (mit Attribut "hubraum") und "Lkw" (mit dem Attribut "leistung"). Implementieren Sie dazu jeweils einen Konstruktor (mit jeweils genau einem Parameter) und die set- und get-Methoden.

Keine weiteren Attribute einfügen!

In den einzelnen Klassen muß jeweils noch die Methode "... getSteuer (...)" implementiert werden, die die Jahressteuer berechnet.

b)

Das Programm soll multipersonal entwickelt werden:

In einem Feld sollen verschiedene Objekte (Auto, Lkw) seines Fuhrparks abgespeichert werden.

Danach soll jeweils die Jahressteuer der einzelnen Objekte auf dem Bildschirm ausgegeben werden.

b1) 13P

Erstellen Sie ein UML-Diagramm (mit allen Attributen und Methoden, außer den Konstruktoren)

b2) 3P  
Was muß programmtechnisch gemacht werden, damit man die Entwickler der einzelnen Klassen zwingen kann, unbedingt die obige Methode "... getSteuer (...)" zu implementieren? Verbale Beschreibung !

b3) 3P  
Ergänzen Sie dazu die bisherige Implementierung (Unbedingt sinnvolle Namen geben!!).  
Bitte keine Attribute der bereits bestehenden Klassen verändern oder neue einfügen!

b4) 9P  
Machen Sie folgendes in der Methode main()

b41)  
Erzeugen Sie das Auto "auto1" mit dem Hubraum 1000 ccm und den Lkw "lkw1" mit der Leistung 300 PS.

b42)  
Speichern Sie diese 2 Objekte in dem Feld.

b43)  
Geben Sie mit Hilfe einer Schleife und der entsprechenden Methode und der Ausgabemethode System.out.println(...) die jeweilige Jahressteuer der einzelnen Fahrzeuge auf dem Bildschirm aus.

b5)  
Aus Sicherheitsgründen will H. Ripig die Daten des Feldes "löschen".  
Wie kann dies programmtechnisch erreicht werden?  
Ergänzen Sie dazu die bisherige Implementierung

## Lösungen:

1)

2P

2a)

20P

```
class Auto extends Fahrzeug { // 1P
    private int hubraum; // 1P

    public Auto(int pHubraum) { // 2P
        setHubraum(pHubraum);
    }

    public void setHubraum(int pHubraum) { // 2P
        hubraum = pHubraum;
    }

    public int getHubraum() { // 2P
        return (hubraum);
    }

    public int getSteuer(){ // 2P
        return(2*hubraum);
    }
}

class Lkw extends Fahrzeug { // 1P
    private int leistung; // 1P

    public Lkw(int pLeistung) { // 2P
        setLeistung(pLeistung);
    }

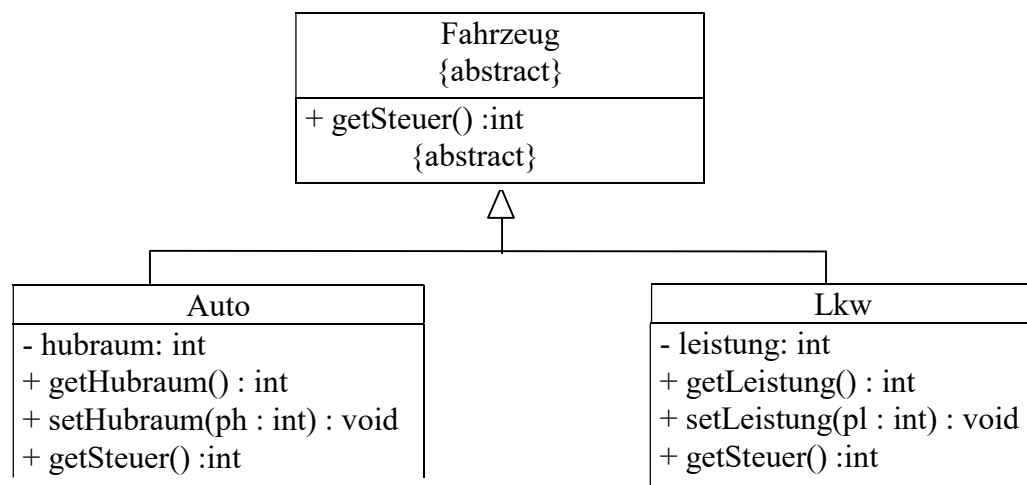
    public void setLeistung(int pLeistung) { // 2P
        leistung = pLeistung;
    }

    public int getLeistung() { // 2P
        return (leistung);
    }

    public int getSteuer(){ // 2P
        return(3*leistung);
    }
}
```

b1)

14P



b2) 3P  
Die Klasse Fahrzeug und die Methode getSteuer () abstract machen.  
Bei den Unterklassen jeweils extends hinzufügen.

b3) 3P

b)  
abstract class Fahrzeug { // 1P  
 abstract public int getSteuer(); // 2P  
}

b4+b5) 9P

```
public class Startklasse {  
    public static void main(String[] args) {  
        int i;  
        Fahrzeug[] feld; // 1P  
        feld = new Fahrzeug[2]; // 1P  
  
        feld[0]=new Auto(1000); // 1P  
        feld[1]=new Lkw(300); // 1P  
        for (i = 0; i < 2; i++) { // 3P  
            System.out.println("Steuer des "+i+"-ten Objekts=  
                                "+feld[i].getSteuer());  
        }  
        for (i = 0; i < 2; i++) { // 2P  
            feld[i]=null;  
        }  
    }  
}
```

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:

Das Spiel "Flaschen drehen" funktioniert wie folgt beschrieben:

Ein Kreis wird in 10 durchnummerierte Kreisausschnitte unterteilt. In der Mitte des Kreises befindet sich eine Flasche, die jeder Spieler, der am "Zug" ist drehen muß und die danach auf genau einen Kreisausschnitt zeigt. Jeder Spieler, der am "Zug" ist, muß vor jedem Drehvorgang mindestens einen Cent auf genau einen Kreisausschnitt setzen (auf dem auch schon mehrere Cents liegen können). Sein Vermögen vermindert sich dann um diesen gesetzten Betrag,

Nach jedem Drehvorgang darf der drehende Spieler alle Cents von dem Kreisausschnitt nehmen, auf den die Flaschenöffnung zeigt. Dann muß der nächste Spieler die Flasche drehen.

Bei 2 Spielern ergibt sich folgende dynamische Entwicklung des Spiels:

Spieler 1 setzt auf einen Kreisausschnitt seiner Wahl eine bestimmte Anzahl Cents

Spieler 1 dreht die Flasche, die danach auf einen bestimmten Kreisausschnitt zeigt.

Für Spieler 1 wird die Gewinnauswertung durchgeführt.

Spieler 2 setzt auf einen Kreisausschnitt seiner Wahl eine bestimmte Anzahl Cents

Spieler 2 dreht die Flasche, die danach auf einen bestimmten Kreisausschnitt zeigt.

Für Spieler 2 wird die Gewinnauswertung durchgeführt.

weiter mit Spieler 1...

1) 11P  
Erzeugen Sie die Klasse "Kreis" mit genau dem Array "dieWerte" als Attribut, einem Konstruktor (ohne Parametern), der alle Kreisausschnitte mit 0 Cents vorbelegt und die folgenden Methoden:

...getWert(...)

gibt die Belegung (Geldwert) eines Kreisausschnitts zurück.

... setWert(...)

belegt einen Kreisausschnitt mit einem bestimmten Einsatz.

Der Geldbetrag auf diesem Kreisausschnitt wird um diesen Einsatz vermehrt.

... reset(...)

belegt einen Kreisausschnitt mit dem Einsatz 0

2) 5P  
Erzeugen Sie die Klasse "Flasche" mit genau dem Attribut "kreis" als Attribut, einem Konstruktor (mit genau einem Parameter) und die Methode

...drehen(), die einen Wert zwischen 0 und 9 zurückgibt.

Dazu dürfen Sie die Methode MyMath.random() verwenden, die Ihnen ein arbeitsloser Mathematiker spendiert hat und die ganze Zahlen zwischen 0 und 9 zurückgibt.

3) 27P  
Erzeugen Sie die Klasse "Spieler" mit genau den Attributen "name", "vermögen", "kreis" als Attribut, einem Konstruktor (mit genau 3 Parametern), den get- und set-Methoden und den folgenden, weiteren Methoden:

... geldSetzen(...)

setzt einen Einsatz auf einen bestimmten Kreisausschnitt. Das gesamte Vermögen wird um diesen Einsatz vermindert.

... spielAuswerten(...)

gibt den aktuellen Gewinn und zusätzlich noch das Vermögen auf dem Bildschirm aus.

4) 9P  
Erstellen Sie ein UML-Diagramm ohne Attribute und ohne Methoden  
Navigierbarkeit, Multiplizität jeweils angeben.

## Lösung:

```
// 11 P
class Kreis{
    private int[] dieWerte; //2P

    public Kreis(){ //2P
        dieWerte = new int[10];
    }

    public int getWert(int i) { //2P
        return dieWerte[i];
    }

    public void setWert(int einsatz, int i) { //3P
        dieWerte[i]=dieWerte[i]+ einsatz;
    }

    public void reset (int i) { //2P
        dieWerte[i]=0;
    }
}

// 5P
class Flasche{
    private Kreis kreis; //1P

    public Flasche(Kreis kreis){ //2P
        this.kreis = kreis;
    }

    public int drehen(){ //2P
        double zufall;
        int erg;
        zufall = Math.random();
        erg=(int) (10*zufall);
        if(erg>9)
            erg=9;
        //System.out.println("gedreht= "+erg);
        return erg;
        //kreis.setNr(erg);
    }
}

// 27P
class Spieler{
    private String name; //1P
    private int vermögen; //1P
    private Kreis kreis; //1P

    //3P
    public Spieler(String name, int vermögen, Kreis kreis){
        this.name = name;
        this.vermögen = vermögen;
        this.kreis = kreis;
    }
}
```

```

public String getName() {                                     //2P
    return name;
}

public void setName(String name) {                           //2P
    this.name = name;
}

public int getVermögen() {                                    //2P
    return vermögen;
}

public void setVermögen(int vermögen) {                      //2P
    this.vermögen = vermögen;
}

public void geldSetzen(int einsatz, int nr) {                //4P
    kreis.setWert(einsatz, nr);
    vermögen = vermögen - einsatz;
}

public Kreis getKreis() {                                    //2P
    return kreis;
}

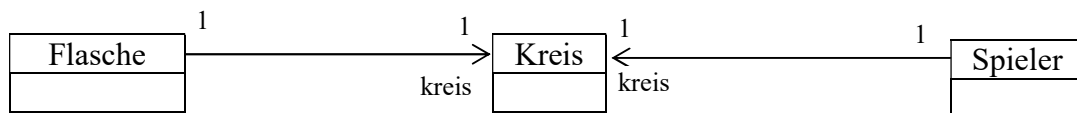
public void setKreis(Kreis kreis) {                          //2P
    this.kreis = kreis;
}

// Spiel auswerten für Kreisausschnitt i
public void spielAuswerten(int i){                           //5P
    int wert;
    wert=kreis.getWert(i);
    vermögen = vermögen + wert;
    // Wert zurücksetzen
    kreis.reset(i);
    System.out.print("Spieler "+name);
    System.out.print(" hat gewonnen: "+wert);
    System.out.println(" hat Gesamtvermögen: "+vermögen);
}
}

```

4)

9P





*Name, Vorname:*

Hilfsmittel:

selbstverfasste, handgeschriebene, schriftliche Unterlagen

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

Zur Information:

Das Spiel "Black Jacket" ("17 + 4") funktioniert wie folgt beschrieben:

Spieler 1 "würfelt" so lange mit einem Spezialwürfel (mit dem genau eine der folgenden Zahlen 2, 3, 4, 7, 8, 9, 10, 11 erwürfelt werden kann), mit dem Ziel als Summe nicht über 21 zu kommen. Die einzelnen Würfe sind verdeckt und vom Gegenspieler nicht zu sehen.

Dann macht der Gegenspieler die gleiche Prozedur.

Wer die größere Summe hat, hat gewonnen.

Wer eine Summe  $> 21$  hat, hat verloren (wenn die Summe des Gegeners  $< 22$ ).

Bei gleicher Summe ist der Spielausgang unentschieden.

Hat jeder Spieler eine Summe  $> 21$ , ist der Spielausgang ebenfalls unentschieden.

Wenn ein Spieler nicht mehr würfeln will oder eine Summe  $> 21$  hat, wird das Spiel sofort beendet und auf dem Bildschirm die entsprechenden Infos ausgegeben (Gewinner und Verlierer mit jeweiliger Punktzahl).

Bemerkung:

Die Methode `würfeln()` darf benutzt werden. Diese liefert genau eine der Zahlen 2, 3, 4, 7, 8, 9, 10, 11

Die Methode `eingabe()` darf benutzt werden. Mit ihr kann man Werte über Tastatur eingeben.

1)

Implementieren Sie die Klasse "Spiel" mit den entsprechenden Attributen und Methoden (mit Hilfe der OOP).

2)

2 Spieler sollen gegeneinander Spielen.

Implementieren Sie das zugehörige Programm in `main()`.

Geben Sie jeweils den Gewinner bzw. Verlierer (mit der jeweiligen Summe) auf dem Bildschirm aus.

*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);
    }
}
```

*Name, Vorname:*

Hilfsmittel:

selbstverfasste, handgeschriebene, schriftliche Unterlagen

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

Ein "geklammerter Term" soll hier eine Zeichenfolge sein, die aus Kleinbuchstaben a - z, den Rechenzeichen +, -, \*, / besteht und geklammert ist (außer die Zeichenfolge besteht genau aus einem Kleinbuchstaben, also a - z).

Beispiele für "geklammerte Terme":

b	(x*y)
((e*(f*f))	((a+b)+(c-d))

Beispiele für keine "geklammerten Terme" (fehlgeformt):

A	(x)
(a- -b)	(a*/b)+c

Erstellen Sie ein Struktogramm nach DIN 66261 für eine Funktion `letzteOeffnendeKlammer`, die als Parameter ein Array von Zeichen hat.

Die Funktion liefert den Index (Stelle) der letzten öffnenden Klammer zurück (falls so eine Klammer existiert), sonst -1.

Erstellen Sie ein Struktogramm nach DIN 66261 für eine Funktion `checkTerm`, die als Parameter ein Array von Zeichen hat. Die Funktion liefert zurück, ob die Zeichenfolge ein "geklammerter Term" ist oder nicht.

Bemerkung:

Die Funktion `ersetze(...)`, die in einer Zeichenfolge eine Zeichenfolge durch ein Zeichen ersetzt, wird als bekannt vorausgesetzt und darf verwendet werden.

Hilfestellung:

Suchen Sie in der Zeichenkette die am weitesten rechts stehende, öffnende Klammer. Falls es diese gibt und die nächsten darauffolgenden Zeichen von der Form: Kleinbuchstabe, Rechenzeichen, Kleinbuchstabe, schließende Klammer sind, wird diese Zeichenfolge durch den Kleinbuchstaben t ersetzt.

Beispiel:

Die am weitesten rechts stehende, öffnende Klammer wird hier durch die Klammer [ kenntlich gemacht. Danach kommt die Ersetzung.

Zu prüfende Zeichenkette:  $((((a+b)*c)-[f+x))+d)$

1. Ersetzung:  $((([a+b]*c)-t)+d)$

2. Ersetzung:  $(([t*c)-t)+d)$

3. Ersetzung:  $([t-t)+d)$

4. Ersetzung:  $[t+d)$

t



## Lösung:

Funktion: letzteOeffnendeKlammer Parameter: string : Array von Zeichen  Rückgabewert: index der letzten öffnenden Klammer -1 : Es gibt keine öffnende Klammer					
last = -1 i = 0					
	<table> <tr> <td></td><td>string[i] == '('</td></tr> <tr> <td>W</td><td>F</td></tr> </table>		string[i] == '('	W	F
	string[i] == '('				
W	F				
last = i					
i ++					
solange String-Ende nicht erreicht					
return last					

Funktion: checkTerm Parameter: string : Array von Zeichen  Rückgabewert: 1: string ist Term 0 : string ist kein Term							
last = letzteOeffnendeKlammer(string);							
last == -1							
<table> <tr> <td>string[0] ist Kleinb. und string[1]!='\0'</td><td>die nächsten 4 Zeichen sind von der Form <b>B</b>uchst., <b>R</b>echnez., <b>B</b>uchtab, )</td></tr> <tr> <td>W</td><td>F</td></tr> </table>	string[0] ist Kleinb. und string[1]!='\0'	die nächsten 4 Zeichen sind von der Form <b>B</b> uchst., <b>R</b> echnez., <b>B</b> uchtab, )	W	F	<table> <tr> <td>W</td><td>F</td></tr> </table>	W	F
string[0] ist Kleinb. und string[1]!='\0'	die nächsten 4 Zeichen sind von der Form <b>B</b> uchst., <b>R</b> echnez., <b>B</b> uchtab, )						
W	F						
W	F						
istTerm = 1 break	istTerm = 0 break Die Zeichenkette <b>'B' 'R' 'B' '</b> durch Zeichen <b>'t'</b> ersetzen						
istTerm = 0 break							
solange true							
return istTerm							