

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 Punkteabzug !!!!

AUFGABEN

I)

Bemerkung:

Alle folgenden Teilaufgaben müssen in genau **einem** einzigen Programm realisiert werden.
Sichtbarkeits- und Zugriffsmodifizierer wie public, usw. dürfen nicht weggelassen werden.

Die (verkäuflichen) Waren (z.B. Wein, Brot, usw.) und die (nicht verkäufliche) Betriebsausstattung (z.B. Stühle, Tische, Geschäftsautos, usw.) bilden zusammen das Betriebsvermögen.

1) 9P

Zeichnen Sie ein UML-Diagramm, in dem die Klassen "Wein", "Brot", "Stuhl", "Tisch", "Betriebsvermögen" mit ihren Beziehungen untereinander vorkommen.

Von den Klassen muß im UML-Diagramm jeweils nur der Name dargestellt werden (ohne Attribute und Methoden).

2) 2P

Die Klasse Brot besitzt genau das Attribut Gewicht.

Der Preis des Brots berechnet sich aus dem doppelten seines Gewichts.

Die Klasse Wein besitzt genau die 2 Attribute Volumen und Alter.

Der Preis des Weins berechnet sich aus der Summe seines Volumens und seines Alters.

Was muß programmtechnisch gemacht werden, damit man genau die Entwickler der Klassen Wein und Brot zwingen kann, unbedingt die Methoden "... getPreis(...)" zu implementieren?
Verbale Beschreibung und Ergänzung des UML-Diagramms oben.

Die folgenden Aufgaben müssen alle in einem Programm erstellt werden

3) 8P

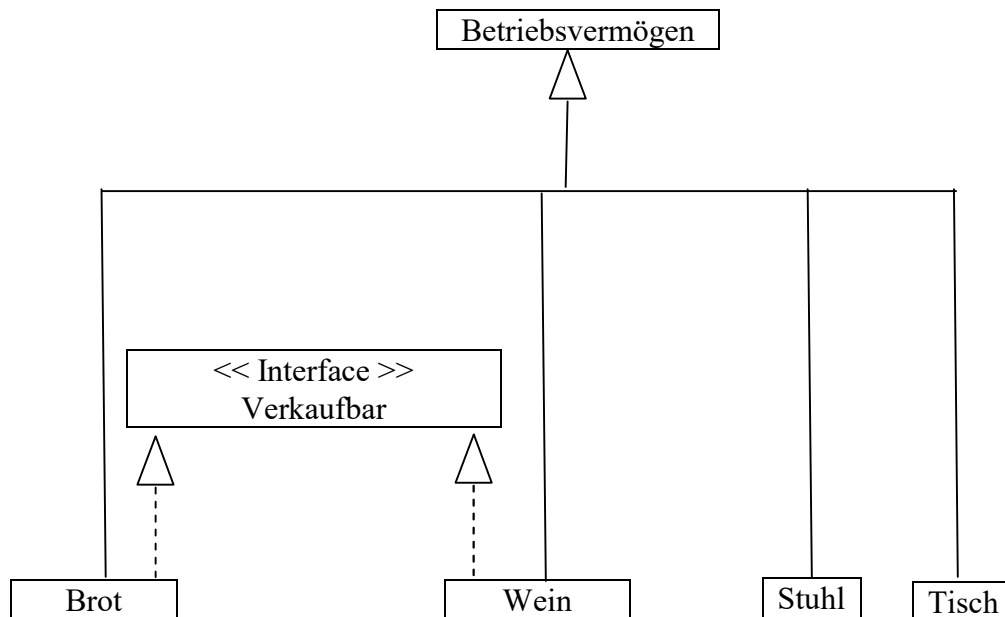
Erzeugen Sie die Klasse "Betriebsvermögen" mit genau dem Attribut "name" (und nur diesem Attribut), den dazugehörigen get- und set-Methoden und einem Konstruktor (mit genau einem Parameter)

- 4) 7P
Erzeugen Sie die Klasse "Brot" mit genau dem Attribut "gewicht" (und nur diesem Attribut), einem Konstruktor (mit genau zwei Parametern) und die Methode "...getPreis(...)", aber ohne die dazugehörigen get-und set-Methoden.
- 5) 7P
Erzeugen Sie die Klasse "Wein" mit genau den Attributen "volumen" und "alter" (und nur diesen Attributen), genau einem Konstruktor (mit genau drei Parametern) und die Methode "...getPreis(...)", aber ohne die dazugehörigen get-und set-Methoden.
- 6) 3P
Erzeugen Sie das Interface "Verkaufbar"
- 7) 2P
Erzeugen Sie in der Klasse "Betriebsvermögen" die statische Methode ...verkaufspreis(...).
- 8) 14P
Machen Sie folgendes in der Methode main()
a) Erstellen Sie ein Brot (mit Variable b bezeichnen) mit Gewicht 10
b) Erstellen Sie einen Wein (mit Variable w bezeichnen) mit Volumen 3 und Alter 4.
c) Speichern Sie diese 2 Objekte in dem Feld "waren" der Länge 2.
d) Mit einer Schleife muß der Preis dieser 2 Waren auf dem Bildschirm ausgegeben werden.
e) Alternativ soll zur Sicherheit zusätzlich noch der Preis des Brotes b mit Hilfe der statischen Methode ...verkaufspreis(...) auf dem Bildschirm ausgegeben werden.

Lösung:

1)

9P



2)

2P

Interface "Verkaufbar" erzeugen und dort den Methodenkopf `getPreis()` angeben.

3-8)

// 14P

```
public class Startklasse {
    public static void main(String[] args) {
        int i;
        Brot b1=new Brot("Dinkel",10);           //2P
        Wein w1=new Wein("Hirnhammer",3,4);       //2P
        Verkaufbar waren[]=new Verkaufbar[2];     //2P
        waren[0]=b1;                             //2P
        waren[1]=w1;                             //2P
        for(i=0;i<2;i++){                       //2P
            System.out.println(waren[i].getPreis());
        }
        System.out.println("Preis="+Betriebsvermögen.verkaufspreis(b1) ); //2P
    }
}
```

// 7P

```
class Brot extends Betriebsvermögen implements Verkaufbar{
    private double gewicht;

    public Brot(String name, double gewicht){
        super(name);
        this.gewicht=gewicht;
    }

    public double getPreis(){
        return 2*gewicht;
    }
}
```

```

// 7P
class Wein extends Betriebsvermögen implements Verkaufbar{
    private double alter;
    private double volumen;

    public Wein(String name, double alter, double volumen){
        super(name);
        this.alter=alter;
        this.volumen=volumen;
    }

    public double getPreis(){
        return volumen+alter;
    }
}

// 10P
abstract class Betriebsvermögen{
    private String name;

    public Betriebsvermögen(String name){
        this.name=name;
    }

    public static double verkaufspreis(Verkaufbar v){
        return v.getPreis();
    }

    public String getName() {
        return name;
    }

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

// 3P
public interface Verkaufbar {
    double getPreis();
}

```

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:

Zu der Sprache Java gehört (genauer im Java-API) die Klasse `Arrays`

Diese besitzt die static-Methode `sort(Object[] a)`

Mit dieser Methode lässt sich ein Feld sortieren.

(Die Klasse `Object` ist Oberklasse jeder Klasse in Java. Sie existiert automatisch und braucht deswegen nicht erstellt zu werden).

Zitat aus der Java-API:

“All elements in the array must implement the Comparable interface.”

Das Interface `Comparable` enthält den Methodenkopf

```
public int compareTo(Object o)
```

Zitat aus der Java-API:

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Bemerkung:

Alle folgenden Teilaufgaben müssen in genau **einem** einzigen Programm realisiert werden.

Sichtbarkeits- und Zugriffsmodifizierer wie public, usw. dürfen nicht weggelassen werden.

1) 10P

Erstellen Sie die Klasse `Hund` mit genau den 2 Attributen "name" und "gewicht", den zugehörigen get- und set-Methoden und genau einem Konstruktor mit genau zwei Parametern.

2) 5P

Erstellen Sie in `main()` das Feld "hunde", in dem die 3 folgenden Hunde abgespeichert werden: Bello mit 30 Kg, Schnappi mit 20 Kg und Rex mit 10 Kg.

3) 35P

a) Beschreiben Sie verbal, was und warum dazu gemacht werden muß.

b) Sortieren Sie mit Hilfe der Methode `sort(...)` dieses Feld bzgl. des Hundegewichts. (programmieren).

Lösung:

```
package sortierenmitinterfacel;
import java.util.Arrays;

public class Startklasse {
    public static void main(String[] args) {
        Hund[] hunde = new Hund[3];
        hunde[0]=new Hund("Bello",30);
        hunde[1]=new Hund("Schnappi",20);
        hunde[2]=new Hund("Rex",10);
        Arrays.sort(hunde);
        for(int i=0;i<3;i++)
            System.out.println(hunde[i].getName()+
                               ""+hunde[i].getGewicht());
    }
}

class Hund implements Comparable {
    private String name;
    private double gewicht;

    public Hund() {
        name = "Musterhund";
    }

    public Hund(String pName, double pGewicht) {
        name = pName;
        gewicht = pGewicht;
    }

    public void setName(String n) {
        name = n;
    }

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

    public void setGewicht(double pGewicht) {
        gewicht = pGewicht;
    }

    public double getGewicht() {
        return (gewicht);
    }

    public int compareTo(Object h){
        if(this.gewicht<((Hund)h).gewicht)
            return -1;
        else if(this.gewicht==((Hund)h).gewicht)
            return 0;
        else
            return 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

Bemerkung:

Alle folgenden Teilaufgaben müssen 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" 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)

Erstellen Sie mit Hilfe der OOP die Klassen "Kreis", "Spieler" und "Flasche".

2)

Die Methode spielAuswerten(...) gibt auf dem Bildschirm den aktuellen Gewinn und das Gesamtvermögen eines Spielers auf dem Bildschirm aus. Zusätzlich wird noch die Anzahl der Cents des entsprechenden Kreisausschnitts auf 0 gesetzt.

3)

Machen Sie folgendes in der Methode main():

Es sollen genau 2 Spieler erzeugt werden.

Diese sollen 30 mal spielen (bei jeweils einem Einsatz von 1 Cent auf den jeweils ersten.

Kreisausschnitt, also es soll 30 mal folgendes gemacht werden:

```
spieler1.geldSetzen(...);  
.... flasche.drehen();  
spieler1.spielAuswerten(...);  
spieler2.geldSetzen(...);  
.... flasche.drehen();  
spieler2.spielAuswerten(...);
```


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)

Eine einfache Geschwindigkeits-Messanlage besteht aus 2 Lichtschranken, die im Abstand d (in Meter) voneinander am Straßenrand aufgestellt sind. Die Geschwindigkeit v eines vorbeifahrenden KFZ wird berechnet aus den Zeitpunkten t_1 und t_2 (jeweils in Sekunden), in denen die Lichtschranken unterbrochen wurden, also: $v = 3,6 * d / (t_2 - t_1)$ wobei v die Dimension km/h hat.

Diese Berechnung erfolgt in der Basisklasse "Messanlage".

Am Ortseingang gibt es die eine Warntafel, die dem Autofahrer anzeigt, mit welcher Geschwindigkeit er in den Ort einfährt ("Sie fahren 55km/h"). Diese wird durch die Klasse "Warntafel" modelliert.

Im Ort steht ein Blitzer. Er blitzt zusätzlich bei Überschreitung einer Geschwindigkeit von 60 km/h und speichert Geschwindigkeit und Blitzsituation ab. Diese wird durch die Klasse "Blitzer" modelliert.

Methoden der Klasse Messanlage:

Messanlage(double abstand) :

Der Konstruktor hat als Parameter den Abstand der Lichtschranken (in Meter). Der Konstruktor kopiert den Parameter abstand in das Attribut d .

void messen(double t_1 , double t_2)

t_1 ist der Zeitpunkt der Unterbrechung der ersten Lichtschranke, t_2 der zweiten. Die Geschwindigkeit des durchfahrenden Fahrzeuges wird berechnet und im Attribut v gespeichert.

void ausgeben(void)

Diese Methode wird nur deklariert und nur in den Unterklassen ausprogrammiert.

Methoden der Klasse Warntafel:

Warntafel(double abstand)

Dieser Konstruktor ruft den Basiskonstruktor von Messanlage auf und übergibt ihm den Parameter abstand.

void ausgeben(void)

gibt die gemessene Geschwindigkeit aus (z.B: "Sie fahren 55km/h!").

Methoden der Klasse Blitzer:

Blitzer(String stdort, double abstand)

Dieser Konstruktor speichert einen Standort-Name im Attribut standort, ruft den Basiskonstruktor von Messanlage auf und übergibt ihm den Parameter abstand.

void messen(double t1, double t2)

überschreibt die gleichnamige Methode der Basisklasse Messanlage!

Die Berechnung der Geschwindigkeit erfolgt, indem die Methode der Oberklasse verwendet wird. Zusätzlich wird das Attribut blitz gleich 1 gesetzt, wenn die Geschwindigkeit v größer 60km/h beträgt, sonst ist blitz gleich 0.

void ausgeben(void)

gibt die gemessene Geschwindigkeit aus, zusätzlich den Standort des Blitzers und das Wort "GEBLITZT!", wenn geblitzt wurde (bei blitz = 1).

Beispiel: Der folgende Quelltext erzeugt die Bildschirm-Ausgabe:

```
public static void main(String[] args) {
    Warntafel w = new Warntafel(20);
    w.messen(170.1, 172.1);
    w.ausgeben();

    Blitzer b = new Blitzer("Parkstrasse", 20);
    b.messen(201.2, 203.2);
    b.ausgeben();
    b.messen(321.9, 322.9);
    b.ausgeben();
}
```

Bildschirmausgaben

// 1. Bildschirmausgabe

Sie fahren 36.0 km/h

// 2. Bildschirmausgabe

* Standort: Parkstrasse

* Geschwindigkeit: 36.0 km/h

// 3. Bildschirmausgabe

* Standort: Parkstrasse

* Geschwindigkeit: 72.0 km/h GEBLITZT!

AUFGABEN:

a) Erstellen Sie das zugehörige UML-Diagramm.

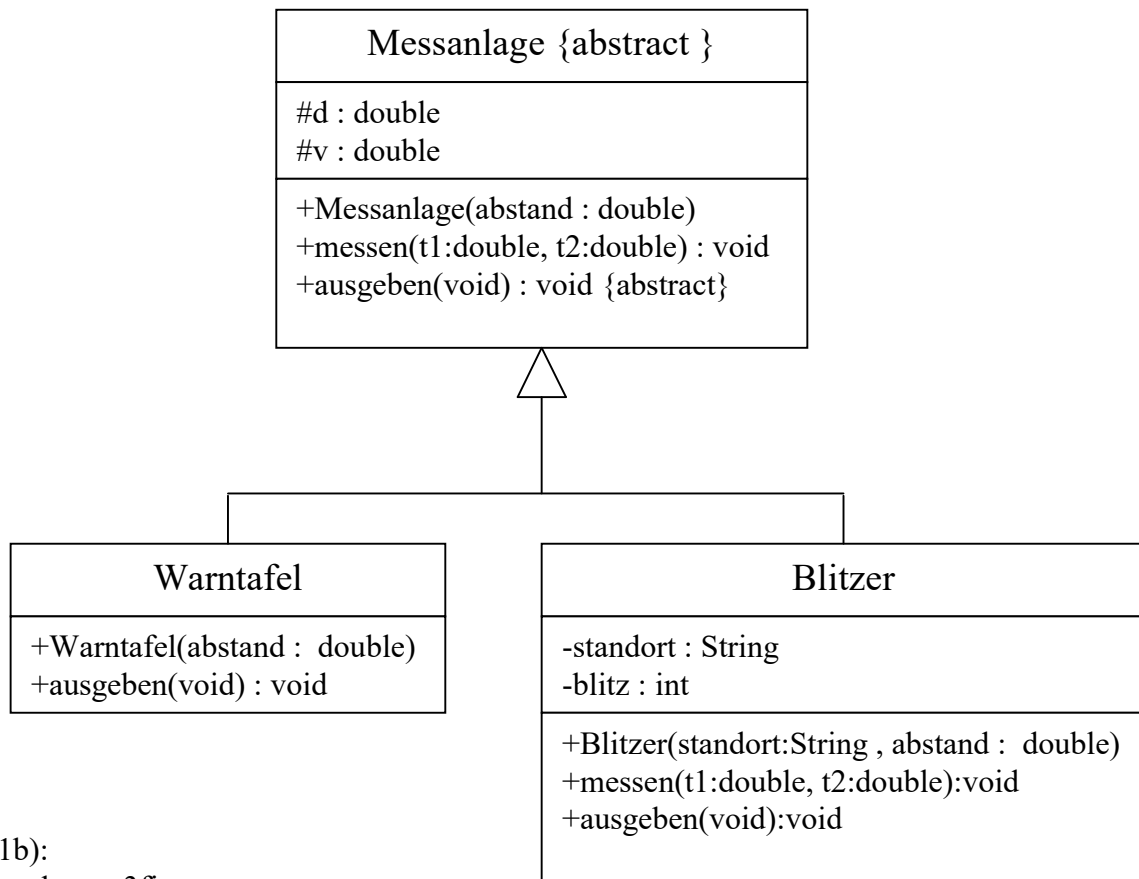
16P

b) Implementieren Sie die 3 Klassen mit den oben beschriebenen Methoden..

34P

Lösungen:

1a)



1b):

```
package e3fi;
```

```
public class MainE3FI{
```

```
    public static void main(String[] args) {
        Warntafel w = new Warntafel(20);
        w.messen(170.1, 172.1);
        w.ausgeben();
```

```
        Blitzer b = new Blitzer("Parkstrasse", 20);
        b.messen(201.2, 203.2);
        b.ausgeben();
        b.messen(321.9, 322.9);
        b.ausgeben();
```

```
    }
}
```

```
abstract class Messanlage{
```

```
    protected double d;
    protected double v;
```

```
    public Messanlage(double abstand){
        d=abstand;
    }
}
```

```
    public void messen(double t1, double t2) {
```

```

        v = (d/(t2 - t1))*3.6;
    }

    public abstract void ausgeben();
}

class Warntafel extends Messanlage{
    public Warntafel(double abstand){
        super(abstand);
    }

    public void ausgeben() {
        System.out.println("\nSie fahren " + v + " km/h");
    }
}

class Blitzer extends Messanlage{
    private String standort;
    private int blitz;

    public Blitzer(String ort, float abstand){
        super(abstand);
        standort = ort;
    }

    public void messen(double t1, double t2) {
        //v = (d/(t2 - t1))*3.6;
        super.messen(t1, t2);
        blitz = 0;
        if(v > 60.0) {
            blitz = 1;
        }
    }

    public void ausgeben(){
        System.out.println("\n*****");
        System.out.println("* Standort: " + standort);
        if(blitz==0)
            System.out.println("* Geschwindigkeit: "+v+ " km/h");
        else
            System.out.println("* Geschwindigkeit: "+v+ " km/h GEBLITZT! ");
        System.out.println("*****");
    }
}

```