Studiengänge: Elektro- und Informationstechnik, Mechatronik, Informatik, Flug- und Fahrzeuginformatik, Wirtschaftsinformatik

Prüfung
Grundlagen der Programmierung 2
Objektorientierte Programmierung

Prüfer: Glavina, Hahndel, Regensburger, U. Schmidt, Windisch

Prüfungsdauer: 90 Minuten

Hilfsmittel: keine

Studiengang	Dozent	Matrikelnummer	Semester	Raum	Platz

Aufgabe	1	2	3	4	Σ	Note
Punkte						

Bitte beachten:

Tragen Sie Ihre persönlichen Angaben auf dieses Deckblatt ein.

Schreiben Sie Ihre Antworten direkt in die dafür vorgesehenen freien Stellen dieser Prüfung.

Geben Sie alle Blätter wieder ab, auch wenn einzelne Seiten nicht beschrieben sein sollten.

Viel Erfolg!

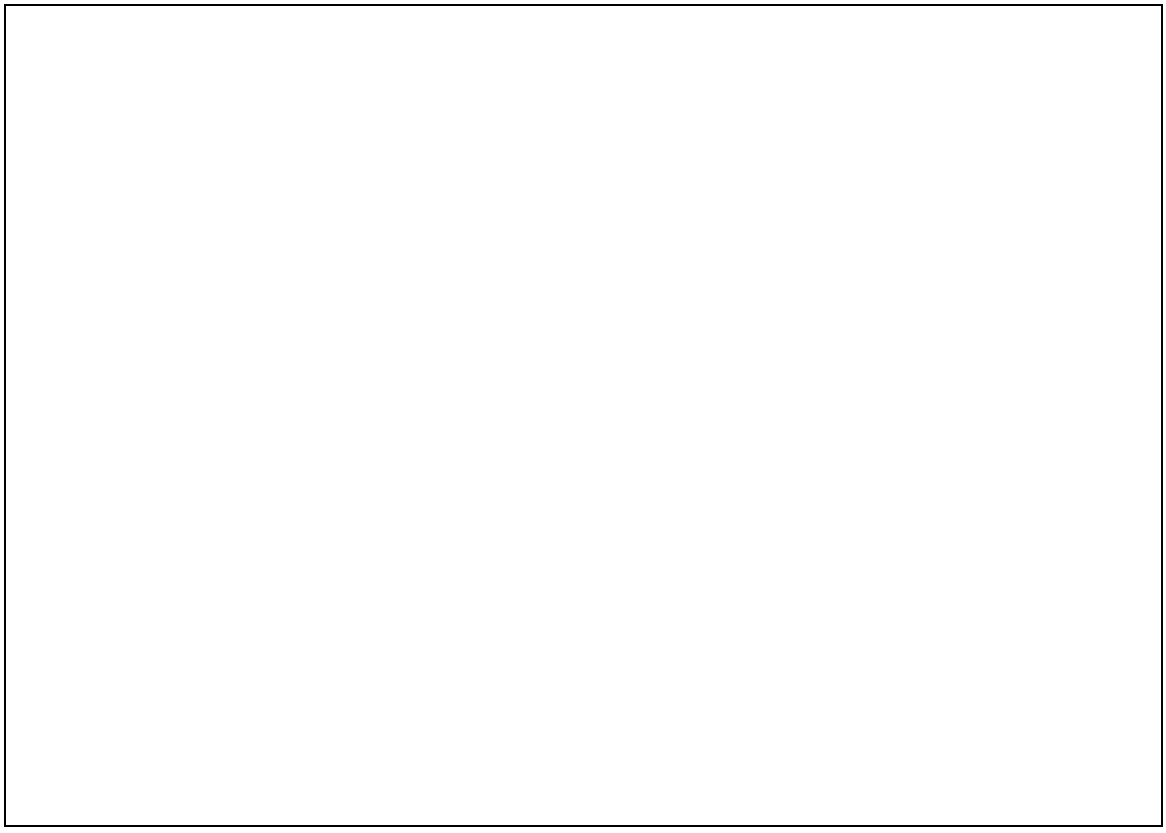
Aufgabe 1 (Verständnisfragen; ca. 25 %)

a) Welche Ausgaben erzeugt das folgende Programm?

```
class Tier {
    Tier() {
        System.out.println("Tier");
    }
    public String toString() {return "Animal";}
}

class Säugetier extends Tier {
    Säugetier() {
        System.out.println(super.toString());
    }
    public String toString() {return "Mammal";}
}

public class Löwe extends Säugetier {
    public Löwe() {
        this("Leo");
        System.out.println(this);
        System.out.println("Löwe");
    }
    public Löwe(String s) {
        System.out.println(s);
    }
    public String toString() {return "Lion";}

    public static void main(String[] args) {
        new Löwe();
    }
}
```

b) Welche Ausgaben erzeugt das folgende Programm?

```
class A {
    public String toString() {return "A";}
}

class B extends A {
    public String toString() {return "B";}
}

class C extends A {}

class D extends C {
    public String toString() {return "D";}
}

class E extends B {}

public class Klasse {
    public static void main(String[] args) {
        A[] liste = new A[] {new A(), new B(), new C(),
                             new D(), new E()};
        for (A a : liste) System.out.println(a);
    }
}
```



c) Ergänzen Sie die folgende Klasse durch

- Implementierung des Interfaces `Comparable<Rechteck>`
- Anlegen eines Arrays von 3 verschieden großen Rechtecken in `main`
- Sortieren des Arrays mit Hilfe von `java.util.Arrays.sort`
- Ausgabe des sortierten Arrays

Rechteck x ist größer als Rechteck y, wenn die Fläche von x größer ist als die Fläche von y; bei gleicher Fläche entscheidet der Umfang.

```
public class Rechteck implements Comparable<Rechteck> {
    private int länge, breite;

    public Rechteck(int länge, int breite) {
        this.länge = länge;
        this.breite = breite;
    }
    public int fläche() {return länge * breite;}
    public int umfang() {return 2 * (länge + breite);}
    public String toString() {
        return "F: " + fläche() + ", U: " + umfang();
    }









    public static void main(String[] args) {









   }
}
```

Aufgabe 2 (Interfaces; ca. 25 %)

Gegeben sei folgendes Interface:

```
interface Addition<T> {
    void addiere(T o); // this += o;
}
```

Die Methode `addiere` soll den übergebenen Parameter zu dem aktuellen Objekt hinzuaddieren. Dieses Interface kann von allen Klassen implementiert werden, die der Addition fähig sind, z.B. komplexe Zahlen, Vektoren, Matrizen u.a.

a) Ergänzen Sie die Klassen `Complex` und `Vector3D` durch geeignete Konstruktoren und implementieren Sie das Interface `Addition` gemäß den jeweils geltenden Rechenregeln für die Addition komplexer Zahlen bzw. die Vektoraddition.

Zur Erinnerung:

Zwei komplexe Zahlen $z_1 = x_1 + i \cdot y_1$ und $z_2 = x_2 + i \cdot y_2$ werden addiert, indem man deren Real- und Imaginärteile addiert:

$$z_1 + z_2 = (x_1 + i \cdot y_1) + (x_2 + i \cdot y_2) = (x_1 + x_2) + i \cdot (y_1 + y_2)$$

Zwei Vektoren $\vec{v}_1 = (x_1, y_1, z_1)^T$ und $\vec{v}_2 = (x_2, y_2, z_2)^T$ werden addiert, indem man ihre Komponenten addiert:

$$\vec{v}_1 + \vec{v}_2 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \\ z_1 + z_2 \end{pmatrix}$$

Der Betrag einer komplexen Zahl ist wie folgt definiert:

$$|z| = |x + i \cdot y| = \sqrt{x^2 + y^2}$$

```
class Complex implements Addition<Complex> {
    private double re, im; // Real- und Imaginärteil









}


class Vector3D implements Addition<Vector3D> {
    private double x, y, z; // Komponenten des 3D-Vektors









}
```

b) Fügen Sie der Klasse `Vector3D` eine Methode `summe3D` hinzu, die alle Vektoren einer übergebenen `liste` aufsummiert und den Ergebnisvektor zurückgibt.

```
static Vector3D summe3D(java.util.List<Vector3D> liste) {









}
```

c) Die Objekte der Klasse `Complex` sollen zusätzlich paarweise vergleichbar sein. Implementieren Sie das Interface `Comparable`. Dabei soll gelten, dass eine komplexe Zahl a genau dann größer ist als eine komplexe Zahl b, wenn der Betrag von a größer ist als der Betrag von b.

(In dieser Teilaufgabe sind nur die Comparable-Methoden zu implementieren.)

```
class Complex implements Addition<Complex>,
                         Comparable<Complex> {









}
```

Aufgabe 3 (Collections; ca. 25 %)

Realisieren Sie eine Klasse `Video` für den Schnitt von Videodateien und ergänzen Sie diese Klasse um die fehlenden Methoden.

Das nachstehende Anwendungsprogramm der Methode `main` soll durch Aufruf dieser Methoden folgende Ausgaben erzeugen:

```
Szenen   : [Bier, Szene A, Kaffee, Zahnpasta, Szene B, Szene
C, Waschmittel]
Werbefrei: [Szene A, Szene B, Szene C]
Werbung  : [Bier, Kaffee, Zahnpasta, Waschmittel]
Werbeeinnahmen: 90 €
```

a) Implementieren Sie eine Methode `add`, die dem Attribut `szenen` einen neuen Eintrag hinzufügt.

b) Implementieren Sie eine Methode `schneiden`, die alle Werbeblöcke in `werbung` und alle werbefreien Blöcke in `werbefrei` zusammenfasst (ein Werbeblock ist jeder Block, der nicht das Wort "Szene" enthält).

c) Implementieren Sie eine Methode `werbeeinnahmen`, die alle Werbeeinnahmen aufsummiert. Jeder Werbeblock bringt eine Werbeeinnahme, und zwar 10 € für jede angefangenen 4 Zeichen der Werbeaussage (z.B. 10 € für `"Bier"`, 30 € für `"Zahnpasta"` usw.)

```
import java.util.*;

public class Video {
    private List<String>
        szenen    = new LinkedList<String>(),
        werbefrei = new LinkedList<String>(),
        werbung   = new LinkedList<String>();

    public static void main(String[] args) {
        Video video = new Video();
        video.add("Bier");     video.add("Szene A");
        video.add("Kaffee");   video.add("Zahnpasta");
        video.add("Szene B");  video.add("Szene C");
        video.add("Waschmittel");
        video.schneiden();
        System.out.println("Szenen   : " + video.szenen);
        System.out.println("Werbefrei: " + video.werbefrei);
        System.out.println("Werbung  : " + video.werbung);
        System.out.println("Werbeeinnahmen: " +
                           video.werbeeinnahmen() + " €");
    }
```

```
    // Implementieren Sie hier die geforderten Methoden


}
```

Aufgabe 4 (GUI und Events, ca. 25 %)

Realisieren Sie die nebenstehende Anwendung zur Ermittlung von Kartenpreisen.
Der Anwender kann sowohl den Einzelpreis als auch die gewünschte Kartenzahl eingeben; die Anwendung ermittelt dann den Gesamtpreis.
Im gezeigten Beispiel wurden 3 Karten à 45 € gewünscht; der Gesamtpreis beträgt also 135 €.

Für beide Textfelder gilt, dass bei Eingabe der Enter-Taste der Gesamtpreis als Produkt von Einzelpreis und Kartenzahl neu berechnet und an der vorgesehenen Stelle ausgegeben wird.

Hinweise

`JTextField` sendet einen `ActionEvent`, wenn die Enter-Taste gedrückt wird.

Die Beschriftung von `JLabel` bzw. der Text von `JTextField` lassen sich lesen und schreiben mit

```
String getText()
void setText(String s)
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Kartenpreise
    extends JFrame implements ActionListener {

    private JTextField einzelpreis, kartenzahl;
    private JLabel gesamtpreis;

    // Ergänzen Sie hier Konstruktor und Callback-Methode
```

```
    public static void main(String[] args) {
        new Kartenpreise();
    }
}
```