Fachhochschule Ingolstadt SS 2007
Studiengang Elektro- und Informationstechnik/
Studiengang Informatik

Prüfung Praktische Informatik 2
Prüfung Grundlagen Informatik 2
Prüfung Softwareentwicklung 2
Prüfung Grundlagen der Programmierung 2

Aufgabensteller: Prof. Glavina, Prof. Grauschopf, Prof. Hahndel, Prof. Schmidt
Prüfungsdauer: 90 Minuten
Hilfsmittel: keine

Aufgabe 1: Verständnisfragen (25 %)

a) Welche Ausgabe erzeugt das folgende Programm?

```
class Auto {
    Auto() {
        System.out.println("Auto");
    }

    Auto(String name) {
        this();
        System.out.println("Auto mit Name");
    }

    public String toString() {
        return "Automobil";
    }
}

class Pkw extends Auto {
    Pkw() {
        this("unbekannt");
        System.out.println("Pkw");
    }

    Pkw(String name) {
        System.out.println(this + " mit Name");
    }
}
```

```
class Cabrio extends Pkw {
    Cabrio(int sitze) {
        System.out.println("Cabrio");
    }
}

class Programm {
    public static void main(String[] args) {
        System.out.println(new Cabrio(2));
    }
}
```

b) Welche Ausgabe erzeugt das folgende Programm?

```
class A {
    private static int i;

    A(int zahl) {i = zahl;}
    int get() {return i;}
}

class Programm {
    public static void main(String[] args) {
        A a1 = new A(47);
        A a2 = new A(11);
        System.out.println(a1.get());
        System.out.println(a2.get());
        System.out.println(a1.get() + a2.get());
    }
}
```

c) Welche Ausgabe erzeugt das folgende Programm?

```
interface X {
    public String x();
}

abstract class Y implements X {
    public String toString() {return "Y";}
}

class Z extends Y {
    public String x() {return "X";}
    public String toString() {return "Z";}
}
```

```
class Programm {
    static void f(X x) {System.out.println(x);}
    static void g(Y y) {System.out.println(y);}
    static void h(Z z) {System.out.println(z);}

    public static void main(String[] args) {
        f(new Z());
        g(new Z());
        h(new Z());
    }
}
```

d) Ergänzen Sie im folgenden Programm die Methode `main`, indem Sie
- ein Array aus mehreren Paprikas anlegen,
- dieses Array nach der Farbe sortieren (mit der Methode `Object[] java.util.Arrays.sort(Object[])`) und
- die sortierten Farben mit einer for-each-Schleife ausgeben.

```
enum Farbton {GRÜN, GELB, ORANGE, ROT};

class Paprika implements Comparable<Paprika> {
    private Farbton farbe;

    Paprika(Farbton farbe) {this.farbe = farbe;}

    public int compareTo(Paprika p) {
        return farbe.ordinal() - p.farbe.ordinal();
    }

    public static void main(String[] args) {

    }
}
```

e) Parametrieren Sie die folgende Klasse `C` mit einem generischen Typ `T`; dabei sollen alle Listenelemente und Methodenparameter vom Typ `T` sein.

```
import java.util.*;

class C implements Comparable {
    private LinkedList liste = new LinkedList();

    boolean add(Object o) {return liste.add(o);}
```

```
    public String toString() {
        String s = "";
        for (Object o : liste) s += o + "\n";
        return s;
    }

    public int compareTo(Object o) {
        return toString().compareTo(o.toString());
    }
}
```

Aufgabe 2: Klassenmodellierung und Polymorphie (ca. 25%)

Ein (logisches) Gatter verfügt über eine Reihe von booleschen Eingängen und berechnet daraus mit der Methode `output()` einen booleschen Wert, was durch folgende Attribute abgebildet werden kann:

- `input` Array von Wahrheitswerten (Eingänge des Gatters)
- `bezeichnung` Zeichenkette (Bezeichnung des Gatters)

Als konkrete Gatter betrachten wir nun `And`, `Or` und `Not`:

- `And` liefert genau dann `true`, wenn alle `input`-Werte `true` sind.
- `Or` liefert genau dann `true`, wenn mindestens ein Wert in `input` `true` ist.
- `Not` hat nur einen Eingangswert und negiert diesen.

a) Zeichnen Sie ein Klassendiagramm (inklusive Attribut- und Methodennamen) mit einer abstrakten Klasse `Gatter` und den drei davon abgeleiteten Klassen `And`, `Or` und `Not`. Die Klassen verfügen neben der Methode `output` über folgende Attribute und Methoden:

   - Jedes Gatter mit Ausnahme von `Not` hat einen Konstruktor mit zwei Parametern, wobei der erste die Bezeichnung des Gatters und der zweite die Anzahl der Eingänge darstellt.
   - Der Konstruktor von `Not` bekommt nur die Bezeichnung des Gatters.
   - Jedes Gatter verfügt über Getter für alle Attribute.
   - Jedes Gatter verfügt über ein Methode `setIndexedBit(int index, boolean value)`. Diese setzt den indizierten `input`-Wahrheitswert auf `value`.

b) Implementieren Sie die Klassen `Gatter`, `And` und `Not`. Achten Sie darauf, dass Konstruktoren und `setIndexedBit` den Wert von `output` korrekt belegen.

c) Schreiben Sie eine Anwendungsklasse, in der Sie zunächst ein `Gatter`-Array der Länge 3 anlegen. Belegen Sie anschließend das Array mit einem zweistelligen `Or`-Gatter, einem fünfstelligen `And`-Gatter und einem `Not`-Gatter.

Die Anwendungsklasse soll ausserdem eine Schleife enthalten, die für jedes Gatter den ersten Eingangswert auf `true` setzt und dann den Wert der Attribute und den Ausgangswert ausgibt.

Aufgabe 3: (ca. 25 %)

Geldbeträge brauchen nicht gerundet zu werden.

a) Schreiben Sie für eine Kassen-Software eine Klasse `Artikel`, die als Attribute eine ganzzahlige Artikelnummer, den Nettopreis in Cent und einen beschreibenden String hat.

Fügen Sie einen Konstruktor hinzu, der die Attribute mit als Parameter übergebenen Werten belegt.

Fügen Sie die Methode `getPreis` hinzu. Diese gibt den Nettopreis in Cent für diesen Artikel zurück, es sei denn die `boolean`-Konstante `Aktionen.MINUS_ZWANZIG` hat den Wert `true`. In diesem Fall wird der um 20% reduzierte Cent-Betrag zurückgegeben. (Die Klasse `Aktionen` muss nicht definiert werden.)

Fügen Sie eine Methode `getMwstBetrag` hinzu. Diese gibt den Mehrwertsteuerbetrag für diesen Artikel in Cent zurück. Der gewöhnliche Mehrwertsteuersatz ist 19%.

b) In den folgenden Aufgaben können Sie davon ausgehen, dass für jedes Attribut von `Artikel` Getter und Setter existieren.

Schreiben Sie eine von Artikel abgeleitete Klasse `Tiernahrung`, die die ererbten Methoden `getPreis` und `getMwstBetrag` wie folgt modifiziert: Der Wert von `Aktionen.MINUS_ZWANZIG` hat keinen Einfluss auf `getPreis`, der Mehrwertsteuersatz ist hier 7%.

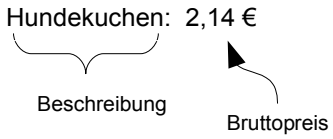
c) Schreiben Sie eine `main`-Methode, die eine `HashMap` namens `lager` anlegt. Diese soll so parametrisiert werden, dass sie den Zugriff auf einen `Artikel` über dessen Artikelnummer erlaubt.

Erzeugen Sie einen „Hundekuchen“ mit Artikelnummer 4711, Nettopreis 200 Cent, und fügen Sie ihn in `lager` ein.

Ermitteln Sie mit Hilfe von `lager` den Preis des Artikels mit der Artikelnummer 12345 und geben Sie diesen aus.

d) Wenn ein Kunde Waren an der Kasse bezahlt, ergibt dies eine `ArrayList<Integer> artikelListe`, die Artikelnummern enthält (kann als gegeben betrachtet werden).

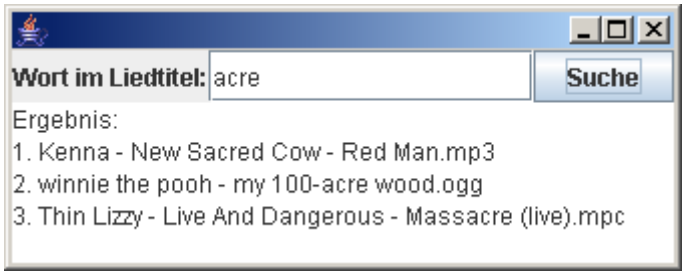
Schreiben Sie ein Programmstück, das basierend auf dieser Liste mit Hilfe von `lager` einen Kassenzettel ausgibt, der zeilenweise wie folgt aufgebaut ist:



Weiter soll der Gesamt-Bruttopreis und die darin enthaltene Mehrwertsteuer ausgegeben werden.

Aufgabe 4: GUI-Aufgabe (25 %)

Hinweis: Im Anschluss an die Aufgabenstellung finden Sie eine kurze Beschreibung einiger API-Methoden.



a) Schreiben Sie eine Klasse `Suchfenster`, die das oben abgebildete Fenster erzeugt. Das Fenster enthält als Attribute einen Suchbereich (`JTextField`) und einen Ausgabebereich (`JTextArea`). Diese sind anfangs leer.

Falls Sie die Swing-Bibliothek nicht kennen, dürfen Sie stattdessen die entsprechenden awt-Steuerelemente verwenden.

b) Deklarieren Sie in `Suchfenster` ein Attribut `playlist`, in dem die Namen von Audiodateien gespeichert werden sollen.

Schreiben Sie eine Methode

```
private void suche()
```

Diese Methode soll in dem Attribut `playlist` nach allen Dateinamen suchen, die

den im Suchbereich stehenden Suchstring enthalten. Die Ergebnisse der Suche sollen mit Überschrift und Nummerierung (siehe Bild) in den Ergebnisbereich geschrieben werden.

c) Modifizieren Sie die Klasse `Suchfenster` so, dass beim Drücken der Schaltfläche "Suche" die in Teilaufgabe b) programmierte Suchmethode gestartet wird. Bei Ereignissen, die nicht von „Suche“ ausgelöst wurden, soll eine Fehlermeldung auf der Konsole ausgegeben werden.

Auszug aus der API für die Klassen `String`, `JTextField` und `JTextArea`:

`String`:

`public boolean` **`contains`**`(String s)`	– Falls der String `s` enthält, wird true zurückgegeben.
`public int` **`indexOf`**`(String str)`	– Falls der String `str` enthält, wird ein Wert >=0 zurückgegeben, sonst -1.
`public char` **`charAt`**`(int index)`	– gibt das Zeichen des Strings mit gegebenem Index zurück.
`public int` **`length`**`()`	– gibt die Länge des Strings zurück.

`JTextField`:

`public String` **`getText`**`()`	– gibt den Inhalt des Textfeldes zurück.
`public void` **`setText`**`(String t)`	– setzt den Inhalt des Textfeldes.

`JTextArea`:

`public String` **`getText`**`()`	– gibt den Inhalt des Textbereichs zurück.
`public void` **`setText`**`(String t)`	– setzt den Inhalt des Textbereichs.
`public void` **`append`**`(String str)`	– fügt `str` am Ende des Textbereichs an.