Fachhochschule Ingolstadt WS 2006/2007
Studiengang Elektro- und Informationstechnik/
Studiengang Informatik

Prüfung Praktische Informatik 2
Prüfung Softwareentwicklung 2

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

Aufgabe 1: (Klassenmodellierung und Polymorphie, ca. 30%)

a) Definieren Sie ein Interface `Rechnung`, die folgende Methoden vorgibt:

- `int getRechnungsbetrag()`
- `String getRechnungstext()`

(Hinweis für die nachfolgenden Teilaufgaben: `getRechnungstext()` soll in den jeweiligen Implementierungen der Klasse einen Text zurückgeben, der Höhe und Zusammensetzung des Preises erläutert, z.B. „`20 Kubikmeter zu 160 Cent ergeben 3200 Cent`“)

b) Im Folgenden sollen die Verbrauchskosten `Stromkosten` und `Wasserkosten` als Klassen modelliert und implementiert werden, die von einer ebenfalls zu implementierenden abstrakten Klasse `Verbrauchskosten` abgeleitet werden, in der alle gemeinsamen Bestandteile der beiden erstgenannten Klassen zusammengefasst werden. Strom wird in der Einheit Kilowattstunden und Wasser in der Einheit Kubikmeter abgerechnet, dabei wird hier zur Vereinfachung grundsätzlich in Cent gerechnet.

- Es wird das unter a) definierte Interface `Rechnung` implementiert.
- Beide Klassen verfügen über die beiden ganzzahligen Attribute `anzahl` und `preisProEinheit`
- Sie enthalten außerdem das Attribut `bezeichnungEinheit` als Zeichenkette.
- Wasserkosten hat ein zusätzliches ganzzahliges Attribut `wasserhärte`.
- Es gibt bei beiden Klassen einen geeigneten Konstruktor, der alle Attribute mit übergebenen Werten initialisiert.
- Die Kosten in Cent ergeben sich jeweils aus der Anzahl der verbrauchten Einheiten multipliziert mit dem Preis für eine Einheit. Eine Spezialität von Wasserkosten: Ist die Wasserhärte > 4, so wird der Rechnungsbetrag um 10% verringert.

c) Implementieren Sie (außerhalb der unter b) aufgebauten Klassenhierarchie) eine Klasse `Kabelanschluss`, die ebenfalls das Interface Rechnung implementiert und zudem über folgende Eigenschaften verfügt:

- Es gibt das ganzzahlige Attribut `grundgebühr`.
- Es gibt einen Konstruktor, der das Attribut mit einem übergebenen Wert belegt.

d) Schreiben Sie eine Anwendungsklasse `Nebenkostenrechnung` mit folgender Funktionalität der `main`-Methode:

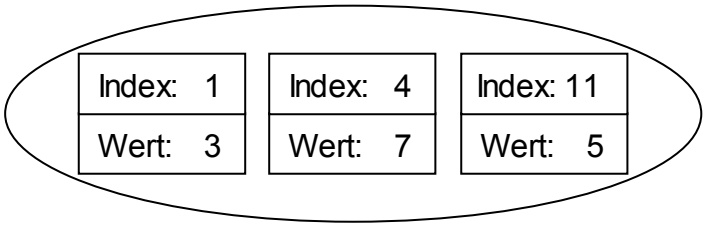
- Zunächst soll ein Array `januar` vom Typ Rechnung für vier Elemente angelegt werden
- Im Array werden anschließend folgende Instanzen abgelegt: Ein Objekt `Stromkosten` (180 Kilowattstunden zu 20 Cent), ein Objekt `Wasserkosten` (20 Kubikmeter zu 160 Cent, die Wasserhärte sei 5), ein Objekt `Kabelanschluss` für 1450 Cent.
- Im letzten Teil soll in einer Schleife sämtliche Rechnungstexte ausgegeben und der Gesamtbetrag der Rechnung berechnet werden, der anschließend ebenfalls ausgegeben wird.

Aufgabe 2: (Collection-Klassen, ca. 35%)

Bei vielen Aufgaben der numerischen Mathematik kommt es im Lösungsverlauf zu sehr großen, aber "dünn besetzten" Matrizen; das sind Matrizen, die nur sehr wenige Elemente haben, die von 0 verschieden sind. Alle Nullen einer dünn besetzten Matrix abzuspeichern, wäre ineffizient, deshalb werden nur die von 0 verschiedenen Elemente in einer Tabelle gespeichert. Beispiel:

$$\begin{pmatrix} 0 & 3 & 0 & 0 \\ 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

wird gespeichert als



Jedes von 0 verschiedene Element wird als Index-Wert-Paar in einer `HashMap` gespeichert. Der Index eines Matrixelements errechnet sich dabei aus seiner Zeilen- und Spaltenadresse (siehe Methode `setElement`).

Gegeben sei das folgende Java-Rahmenprogramm, das dünne Matrizen beliebiger Größe verwalten soll; die Matrixelemente werden mit `setElement` hinzugefügt:

```
import java.util.HashMap;

class DünneMatrix {
    private int zeilen, spalten;
    private HashMap<Integer, Double> elemente;

    DünneMatrix(int zeilen, int spalten) {
        this.zeilen = zeilen;
        this.spalten = spalten;
        elemente = new HashMap<Integer, Double>();
    }

    void setElement(int zeile, int spalte, double wert) {
        if (wert != 0.0)
```

```
            elemente.put(zeile * spalten + spalte, wert);
    }

    int füllgrad() {
        // gibt einen Wert zwischen 0 und 100 zurück
    }

    void drucke() {
        // gibt die Matrix zeilen- und spaltenweise aus
    }

    DünneMatrix transponiere() {
        // gibt die Transponierte der Matrix zurück
    }

    public static void main(String[] args) {
        // Anwendungsprogramm
    }
}
```

a) Implementieren Sie die Methode `füllgrad`, welche das Verhältnis zwischen der Zahl der von 0 verschiedenen Elemente zur Zahl aller Elemente angibt. Es soll dabei ein korrekt gerundeter Prozentsatz zwischen 0 und 100 zurückgegeben werden. (Im obigen Beispiel würde 25 zurückgegeben werden.)

b) Implementieren Sie die Methode `drucke`. Diese Methode soll eine dünne Matrix in ihrer ursprünglichen Gestalt, also einschließlich der Nullelemente, zeilen- und spaltenweise auf dem Bildschirm ausgeben.

c) Implementieren Sie die Methode `transponiere`. Diese Methode soll eine neue Matrix zurückgeben, die entsteht, wenn man Zeilen und Spalten der Originalmatrix vertauscht (die Originalmatrix bleibt unverändert).

d) Schreiben Sie als Anwendung eine Methode `main`, die
- eine dünne Matrix gemäß dem obigen Beispiel erzeugt
- den Füllgrad der Matrix wie folgt ausdruckt: "Füllgrad = … %"
- die Matrix ausdruckt
- die Transponierte der Matrix ausdruckt

Einige wichtige Methoden der Klasse `HashMap<K, V>`:

`public V` **`put`**`(K key, V value)`
assoziiert (paart) den Wert `V` mit dem Schlüssel `K`

`public V` **`get`**`(Object key)`
liefert den Wert zurück, der mit dem angegebenen Schlüssel assoziiert ist, oder `null`, falls dieses Schlüssel-Wert-Paar nicht existiert

`public int` **`size`**`()`
liefert die Anzahl der Schlüssel-Wert-Paare

Aufgabe 3 (Stringmanipulation, GUI; ca. 35%)

Bücher werden eindeutig identifiziert durch die ISBN (International Standard Book Number). Diese Nummer besteht aus neun bedeutungsvollen Ziffern und einer Prüfziffer (das ist die letzte, zehnte Ziffer). Beispiele für ISBNs sind "3-86640-001-2", "3-446-19313-8" oder "0-7475-5100-6" (die Bindestriche können an verschiedenen Stellen stehen und trennen die Bestandteile Land, Verlag, Titel und Prüfziffer).

a) Schreiben Sie eine Methode `pruefziffer` (als Hilfsfunktion für eine noch zu erstellende Klasse), die eine ISBN als Parameter übergeben bekommt und als Ergebnis (`char`) die Prüfziffer zurück liefert.

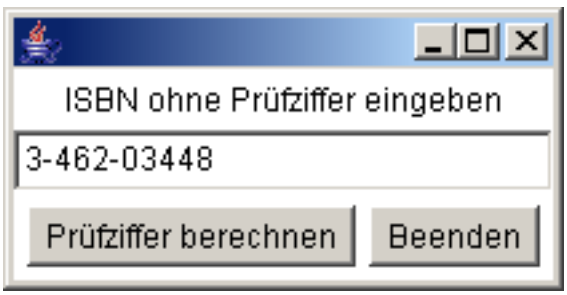
Der Parameter darf nur Ziffern, Leerzeichen und Bindestriche enthalten; Leerzeichen und Bindestriche werden bei der Bearbeitung einfach ignoriert. Die Anzahl der Ziffern muss genau neun sein. In allen anderen Fällen – unerlaubte Zeichen, zu viel oder zu wenig Ziffern – wird einfach ein Fragezeichen als Ergebnis geliefert.

Die Prüfziffer ergibt sich wie folgt:

- aus den ersten neun Ziffern wird eine gewichtete Quersumme gebildet: erste Ziffer mal 1, plus zweite Ziffer mal 2, plus dritte Ziffer mal 3, usw. bis, neunte Ziffer mal neun.
- von dieser Quersumme wird der Rest genommen, der bei der Division durch 11 entsteht.
- falls der Rest 10 ist, wird ein großes X geliefert, ansonsten die entsprechende Ziffer.

Sie dürfen bei Ihren Berechnungen davon ausgehen, dass bereits die Methoden `digitToInt` und `intToDigit` (siehe nachfolgender Programmrahmen) vorliegen, die eine Ziffer (`char`) in eine Zahl, bzw. eine (einstellige) Zahl in ein Ziffer-Zeichen umwandeln (Beispiele: `digitToInt('5') → 5, intToDigit(5) → '5'`).

b) Wir betrachten nun eine Klasse `IsbnCheck`, die ein Fenster wie abgebildet öffnet, und beim Drücken auf den Knopf "Prüfziffer berechnen" eine Meldung
"Die Prüfziffer für die ISBN ... ist ..." auf der Konsole ausgibt (die Auslassungen „...“ sind dabei durch die jeweils aktuellen Werte zu ersetzen!).



Beim Drücken auf "Beenden" wird das Programm beendet.

Implementieren Sie den Konstruktor `IsbnCheck` sowie die Methoden `main` und `actionPerformed`. (S. Programmrahmen nächste Seite.)

```
// Programmrahmen für die vorstehende Aufgabe:

import java.awt.*;
import java.awt.event.*;

public class IsbnCheck
extends Frame implements ActionListener
{
  private TextField isbn;
  // TextField hat u.a. die Methoden
  // public String getText()
  // public void setText(String s)

  public IsbnCheck()
  {
    ... // fehlt noch!
  }

  public static void main(String[] args)
  {
    ... // fehlt noch!
  }

  public void actionPerformed(ActionEvent e)
  {
    String buttonCommand = e.getActionCommand();
    ... // fehlt noch!
  }

  private char pruefziffer(String is)
  {
    ... // fehlt noch!
  }

  private int digitToInt(char c)
  {
    // darf als vorgegeben angenommen werden
  }

  private char intToDigit(int i)
  {
    // darf als vorgegeben angenommen werden
  }

} // end class IsbnCheck
```