

Übung 1 (6 Punkte)

Ermitteln Sie eine mathematische Formel, die die Abhängigkeit der Suchzeit von der Anzahl der Zahlen N angibt und berechnen Sie mit Ihrer Formel die durchschnittliche Suchzeit für 32 und 500 Zahlen (2 Punkte).

Übung 2 (10 Punkte)

Erstellen Sie mit BlueJ eine neue Klasse Zahlen, welche die Natürlichen Zahlen repräsentieren soll. Dazu enthält die Klasse ein Attribut namens liste, welches wiederum ein Array von 32 int-Zahlen ist.

Der Konstruktor dieser Klasse sollte so aussehen:

```
public Zahlen()
{
    liste = new int[32];
    rand = new Random();

    for (int i=0; i<32; i++)
        liste[i] = rand.nextInt(100)+1;
}
```

Nun Ihre eigentliche Aufgabe:

Neben dem Konstruktor benötigt die Klasse Zahlen noch eine Methode zum Anzeigen: `public void anzeigen()` für die Konsolenausgabe.

Programmieren Sie die Methode so, dass jeweils 16 Zahlen nebeneinander angezeigt werden! Für 32 Zahlen benötigt Ihre Ausgabe also 2 Zeilen.

Übung 3 (10 Punkte)

Verändern Sie die Klasse Zahlen so, dass beim Erzeugen der Zufallszahlen jede Zahl aus 1..100 höchstens einmal vorkommt. Außerdem sollen jetzt nicht 32, sondern 64 Zahlen erzeugt werden.

Übung 4 (6 Punkte)

Erweitern Sie die `anzeigen()`-Methode der Klasse Zahlen so, dass jetzt mehr als 64 Zahlen angezeigt werden können, z.B. 100 oder 150 (Wiederholungen sind nicht zugelassen). Es sollen allerdings immer 16 Zahlen pro Reihe angezeigt werden. Bei 150 Zahlen müsste die `anzeigen()`-Methode also 10 Reihen von Zahlen anzeigen, 9 Reihen mit 16 Zahlen und die restlichen 6 Zahlen in der zehnten Reihe (4 Punkte).

Übung 5 (10 Punkte)

Erweitern Sie die Klasse um eine Sortiermethode `feldSortieren()`.

Übung 6 (15 Punkte)

Man programmiere eine lineare Methode `suchen(int key, int anfang, int ende)` die nach Eingabe der Suchzahl die Zahl im Array sucht und die Position ausgibt oder „Nicht gefunden“ meldet, wenn die Zahl nicht im Array vorhanden ist.

Aufgabe 7 (15P)

Ergänzen Sie die Klasse Zahlen um ein rekursives binäres Suchverfahren. Eingaben sind der Suchschlüssel, der Feldanfang und das Feldende. Ausgaben sind die Indexnummer des Feldelementes, das mit dem Schlüssel übereinstimmt und die Anzahl der erforderlichen Vergleiche. Testen Sie dann das binäre Suchverfahren mehrmals mit verschiedenen im Array enthaltenen und auch nicht enthaltenen Suchzahlen, ob es funktioniert.

Hilfen und Anleitungen zum Algorithmus "binäre Suche"**Schritt 1**

Die Mitte des Arrays wird berechnet.

Seien links und rechts die Indices des ersten und des letzten Arrayelementes, so berechnet sich die Mitte nach folgender Gleichung:

$$\text{mitte} = (\text{links} + \text{rechts}) / 2$$

Da mitte, links und rechts int-Zahlen sind, findet eine ganzzahlige Division statt. Ein Array mit 32 Elementen hat folgende Daten:

$$\text{links} = 0, \text{ rechts} = 31.$$

Daraus berechnet sich mitte mit $31/2 = 15$. Die Zahl mit dem Index 15 ist also jetzt das mittlere Element.

Schritt 2

Jetzt wird das mittlere Element mit der Suchzahl verglichen. Ist die Suchzahl größer als das mittlere Element, so muss rechts weiter gesucht werden (Denken Sie daran: der Array ist sortiert!). Ist die Suchzahl kleiner als das mittlere Element, so wird links weitergesucht. Trifft keine der beiden Bedingungen zu, so ist die mittlere Zahl die gesuchte Zahl, und der Algorithmus kann abbrechen. Als Flussdiagramm sieht der Algorithmus so aus:

Man kann diesen Algorithmus rekursiv programmieren oder in eine while-Schleife einbauen. Auf jeden Fall ist sicherzustellen, dass das mittlere Element immer wieder neu berechnet wird.

Eine mögliche rekursive Funktion könnte folgenden Funktionskopf haben:

```
int sucheRekursiv(int such, int l, int r)
```

such ist die zu suchende Zahl, l und r die linke und die rechte Grenze des Arrays.

Wird die Funktion zum ersten Mal aufgerufen, ist $l = 0$ und $r = N-1$, wobei N die Zahl der Arrayelemente ist. Bei einem aus 32 Zahlen bestehenden Array würde also

folgender Aufruf stattfinden: `ergebnis = sucheRekursiv(suchzahl, 0, 31);`

ergebnis ist eine int-Variable, die das Funktionsergebnis von `sucheRekursiv()` speichert.

Die rekursive Methode sollte als private-Funktion deklariert werden. Nach dem Prinzip der Datenkapselung sollten die Parameter einer Methode keine Rückschlüsse über interne Implementierungen der Methode erlauben. Würde man nun zwei Arraygrenzen als Parameter übergeben, so würde man dieses Prinzip verletzen.

Nach dem Prinzip der Datenkapselung darf man der Suchmethode nur die Suchzahl übergeben. Also schreiben wir eine zweite Methode, diesmal eine öffentliche (public), welche nichts anderes macht, als die private rekursive Methode aufzurufen:

```
public int sucheBinaer(int suchzahl){
    return sucheRekursiv(suchzahl,0,31); }
```

Bevor man das binäre Suchen ausführt, muss man dafür sorgen, dass der Array sortiert ist. Benutzen Sie dafür den in der Hausaufgabe 5 einzubauenden Algorithmus.

Das Java-Programm ist bis zum 31.01.08 an andreas.pohl@igfsek2 komprimiert zu schicken.

Viel Erfolg!

Lösung zur Suchen-Aufgabe 1

Wie lange muss man höchstens suchen?

- Gegeben: Liste mit N Einträgen
- Gesucht: Anzahl der höchstens benötigten Suchschritte

Anders herum:

- Gegeben: k Suchschritte (Vergleiche)
- Gesucht: Anzahl der Einträge, die durchsucht werden können:
 - mit 1 Vergleich: 2 Einträge*
 - mit 2 Vergleichen: 4 Einträge*
 - mit 3 Vergleichen: 8 Einträge*
 - mit k Vergleichen: $2 \cdot 2 \cdot \dots \cdot 2 = 2^k$ Einträge*

Nochmal: Wie viele Vergleiche für N Einträge?

- Mit k Vergleichen lassen sich $N = 2^k$ Einträge durchsuchen
- Wie lässt sich k aus N bestimmen?

Umkehrfunktion zur Zweier-Potenz: Zweier-Logarithmus \log_2

Damit: $k = \log_2 N$

Beispiel:

Wie viele Vergleiche k für $N = 10.000$ Einträge?

$$k = \log_2 10.000 \approx 13,29 \Rightarrow 14 \text{ Vergleiche}$$

Ratespiel:

Ich denke mir eine Zahl zwischen 1 und 1.000.

Finde mit 10 Ja/Nein-Fragen die Zahl heraus!

Klappt mit binärer Suche: $\log_2 1.000 \approx 9,97 \Rightarrow 10$ Vergleiche

/**

*** Lösungen der Aufgaben 2, 3 und 4**

* Es werden 64 Zufallszahlen aus dem Bereich 0..100

* ohne Wiederholungen erzeugt. Die Ausgabe erfolgt in Zeilen

* mit jeweils 16 Zahlen

* **Aufg4:** führen Sie als globale Variable int max=100 (oder 150) ein

* @author (Andreas Pohl)

* @version (21.1.2008)

*/

import java.util.Random;

public class Zahlen{

int[] liste; // Instanzvariablen

Random rand;

int max=100;

public Zahlen() { //Konstruktor für Objekte der Klasse Zahlen

liste = new int[max];

rand = new Random();

for(int i = 0; i < max; i++)

{

liste[i] = rand.nextInt(100) + 1;

for(int j = 0; j < i; j++) //keine doppelten Elemente

{

if(liste[i] == liste[j])

i = i - 1;

}

}

}

public void listenAusgabe(){

//Listenausgabe - Aufgabe 2 - mit Zeilenumbruch

System.out.println("Die Liste enthaelt die folgenden Elemente: ");

int j = 0;

for(int i = 0; i < max; i++)

{

System.out.print(liste[i]);

System.out.print(" ");

j = j + 1;

if(j == 16)

{

System.out.println();

j = 0;

}

}

System.out.println();

}

public void listFuellen() //Neue Werte fuer das Array

{

liste = new int[max];

rand = new Random();

for(int i = 0; i < max; i++)

{

liste[i] = rand.nextInt(100) + 1;

for(int j = 0; j < i; j++) //Innere for-Schleife bis zum jeweilig gueltigen i

{

if(liste[i] == liste[j])

i = i - 1; //Prüfung auf Wiederholung

}

}

}

```
/**
```

```
* Lösungen der Aufgaben 5
```

```
* Wir verwenden z.B. den Algorithmus BubbleSort mit der dazugehörigen Methode vertausche( ).
```

```
* @author (Andreas Pohl)
```

```
* @version (21.1.2008)
```

```
*/
```

```
public void feldSortieren()
{
    int grenze = 1;
    int m = 0;
    int n = 0;
    while(grenze < maxIndex)
    {
        n = n + 1; //Anzahl der Durchgaenge
        int merke = maxIndex; //
        for(int i = 0; i < maxIndex - 1; i++)
        {
            if(sortFeld[i] > sortFeld[i + 1])
            {
                vertausche(i, i + 1);
                merke = i;
                m = m + 1; //Anzahl der Vertauschungen
            }
        } // for i
        grenze = merke;
    } // while grenze
    System.out.println("Vertauschungen: "+m);
    System.out.println("Durchgaenge: "+n);
} // bubblesort

public void vertausche(int a, int b)
{
    int ablage = sortFeld[a];
    sortFeld[a] = sortFeld[b];
    sortFeld[b] = ablage;
} // vertausche
```

```
/**
 * Lösungen der Aufgaben 6
 * Methode lineare Suche
 * @author (Andreas Pohl)
 * @version (21.1.2008)
 */

public void lineareSuche(int key)
{
    boolean gefunden = false;
    int position = 0;
    int j = 1;
    for(int i = 0; i < maxIndex; i++)
    {
        if(key == sortFeld[i])
        {
            gefunden = true;
            System.out.println("Gefunden an Pos ("+(i + 1) + ")"+j + "-mal");
            position = i;
            j = j + 1;
        }
    }
    if (!gefunden)System.out.println("Nicht gefunden");
}
```

```
/**
```

```
* Lösungen der Aufgaben 7
```

```
* rekursive Methode binäreSuche
```

```
* @author (Andreas Pohl)
```

```
* @version (21.1.2008)
```

```
*/
```

```
public void sucheRekursiv(int key, int erstes, int letztes)
{
    boolean gefunden = false;
    if(letztes <= erstes)
    {
        gefunden = false;
        System.out.println("Keine Chance, "+key+" ist nicht dabei!");
    }
    else
    {
        if(key == sortFeld[erstes])
        {
            gefunden = true;
            int nr=erstes+1;
            System.out.println(key+" an Platz Nr.: "+nr);
            return;
        }
        if(key == sortFeld[letztes])
        {
            gefunden = true;
            int nr=letztes+1;
            System.out.println(key+" an Platz Nr.: "+nr);
            return;
        }
        int mitte =(erstes + letztes) / 2;
        if(key < sortFeld[mitte])binSuchen(key, erstes, mitte);
        else binSuchen(key, mitte, letztes);
    }
}
```

Vollständiges Programm zu den Aufgaben 2-7

```
/**
 * Lösungen der Aufgaben 1-7
 *
 * @author (Andreas Pohl)
 * @version (08.02.2008)
 */
import java.util.*;
public class Zahlen{
    // Anfang Globale Variablen
    int[] sortFeld;
    int maxIndex = 100;
    Random rand;
    // Ende Globale Variablen
    //Konstruktor initialisiert das Feld sortFeld
    public Zahlen(){
        sortFeld = new int[maxIndex];
        rand = new Random();
        for(int i = 0; i < maxIndex; i++)
        {
            sortFeld[i] = rand.nextInt(1000) + 1;
            for(int j = 0; j < i; j++)
            {
                if(sortFeld[i] == sortFeld[j])
                    i = i - 1;
            }
        }
    }

    // Anfang Ereignisprozeduren
    public void feldFuellen(){
        sortFeld = new int[maxIndex];
        rand = new Random();
        for(int i = 0; i < maxIndex; i++)
        {
            sortFeld[i] = rand.nextInt(1000) + 1;
        }
    }
    public void feldAusgabe(){
        System.out.println("Das Feld enthaelt die folgenden Elemente: ");
        int j = 0;
        for(int i = 0; i < maxIndex; i++)
        {
            System.out.print(sortFeld[i]);
            System.out.print(" ");
            j = j + 1;
            if(j == 16)
            {
                System.out.println();
                j = 0;
            }
        }
        System.out.println();
    } //feldAusgabe
}
```

```
public void sort(){
    int grenze = 1;
    int m = 0;
    int n = 0;

    while(grenze < maxIndex)
    {
        n = n + 1; //Anzahl der Durchgaenge
        int merke = maxIndex; //
        for(int i = 0; i < maxIndex - 1; i++)
        {
            if(sortFeld[i] > sortFeld[i + 1])
            {
                vertausche(i, i + 1);
                merke = i;
                m = m + 1; //Anzahl der Vertauschungen
            }
        } //for i
        grenze = merke;
    } //while grenze
    System.out.println("Vertauschungen: "+m);
    System.out.println("Durchgaenge: "+n);
} //bubblesort

public void vertausche(int a, int b){
    int ablage = sortFeld[a];
    sortFeld[a] = sortFeld[b];
    sortFeld[b] = ablage;
} //vertausche

public void fuegeEin(int inhalt, int i){
    while((i >= 0) &&(sortFeld[i] > inhalt))
    {
        sortFeld[i + 1] = sortFeld[i];
        i = i - 1;
    }
    sortFeld[i + 1] = inhalt;
}

public void lineareSuche(int key){
    boolean gefunden = false;
    int position = 0;
    int j = 1;
    for(int i = 0; i < maxIndex; i++)
    {
        if(key == sortFeld[i])
        {
            gefunden = true;
            System.out.println("Gefunden an Pos ("+(i + 1) + ")"+j + "-mal");
            position = i;
            j = j + 1;
        }
    }
    if (!gefunden)System.out.println("Nicht gefunden");
}
```

```
public void binSuchen(int key, int erstes, int letztes){
    boolean gefunden = false;
    if(letztes <= erstes)
    { gefunden = false;
      System.out.println("Keine Chance, "+key+" ist nicht dabei!");
    }
    else
    { if(key == sortFeld[erstes])
      { gefunden = true;
        int nr=erstes+1;
        System.out.println(key+" an Platz Nr.: "+nr);
        return;
      }
      if(key == sortFeld[letztes])
      { gefunden = true;
        int nr=letztes+1;
        System.out.println(key+" an Platz Nr.: "+nr);
        return;
      }
      int mitte =(erstes + letztes) / 2;
      if(key < sortFeld[mitte])binSuchen(key, erstes, mitte);
      else binSuchen(key, mitte, letztes);
    }
  }
}

public void sucheBinaer(int key){
    binSuchen(key,0,maxIndex-1);
}
}
```

1. Schreibe für das Programm vom letzten Freitag eine Methode `reHash(int code)`, der bei einer Kollision der zuerst bestimmte Hashcode übergeben und um 1 erhöht zurückgeliefert wird.
2. In Java gibt es die Klasse `HashTable`, die über die Methode `int hashCode()` der Klasse `Object` und eine Kollisionsbehandlung verfügt. Teste die Beispiellasse mit BlueJ und beschreibe die Funktionsweise!

NACH BLUEJ ÜBERTRAGEN!

```
import java.util.*;

public class HashTableTest{

    public static void main(String[] args)
    {
        Hashtable h = new Hashtable();

        //Pflege der Aliase
        h.put("Fritz", "f.mueller@test.de");
        h.put("Franz", "fk@b-blabla.com");
        h.put("Paula", "user0125@mail.uofm.edu");
        h.put("Lissa", "lb3@gateway.fhdto.northsurf.dk");

        //Ausgabe
        Enumeration e = h.keys();
        while (e.hasMoreElements()) {
            String alias = (String)e.nextElement();
            System.out.println(
                alias + " --> " + h.get(alias)
            );
        }
    }
}
```

Die Klasse `Hashtable` ist eine Konkretisierung der abstrakten Klasse `Dictionary`.

Diese stellt einen assoziativen Speicher dar, der Schlüssel auf Werte abbildet und über den Schlüsselbegriff einen effizienten Zugriff auf den Wert ermöglicht.

Ein `Dictionary` speichert also immer zusammengehörige Paare von Daten, bei denen der Schlüssel als Name des zugehörigen Wertes angesehen werden kann.

Über den Schlüssel kann später der Wert leicht wiedergefunden werden.

Das Einfügen von Elementen erfolgt durch Aufruf der Methode `put`:

```
public Object put(Object key,
Object value)
```

Alle Klassen erben von der Klasse `Object`. `Object` ist die Superklasse aller anderen Klassen. Die Klasse `Object` definiert u.a. die elementare Methode: `int hashCode()`

Das Interface `Enumeration` besitzt die Methoden `hasMoreElements` und `nextElement`.

Nach der Initialisierung zeigt ein `Enumeration`-Objekt auf das erste Element der Aufzählung. Durch Aufruf von `hasMoreElements` kann geprüft werden, ob weitere Elemente in der Aufzählung enthalten sind, und `nextElement` setzt den internen Zeiger auf das nächste Element

Hausaufgaben zu Freitag, 15. 2.2008 (je 10 Punkte)

Für Ein- und Ausgaben sollen die Klassen `In` und `Out` benutzt werden!

- 1) Gegeben seien drei Zahlenvariablen `a`, `b` und `c`. Schreiben Sie einen Algorithmus *sort*, der die Variableninhalte so umordnet, dass $a \leq b \leq c$ ist.
- 2) Schreiben Sie ein Java-Programm, das die `x`- und `y`-Koordinaten zweier Punkte einliest und den Abstand zwischen ihnen berechnet und ausgibt.
Hinweis: Die ganzzahlige Wurzel einer Zahl `x` können Sie als `(int)Math.sqrt(x)` berechnen.
- 3) Herr Meyer und Herr Schmidt sind Angestellte in derselben Firma. Herr Meyer erhält 2400€ Gehalt, Herr Schmidt 2750€. Bei einer Personalversammlung haben sie sich zwischen einer Gehaltserhöhung prozentual 4,5% und einem Festbetrag von 100€ zu entscheiden.
Planen und schreiben Sie ein Javaprogramm, das nach Eingabe von Gehalt und Prozentsatz die Erhöhung ausgibt.

Lösungen

- 1) Gegeben seien drei Zahlvariablen a, b und c. Schreiben Sie einen Algorithmus *sort*, der die Variableninhalte so umordnet, dass $a \leq b \leq c$ ist.

Lösung (Da nur ein Algorithmus gefordert ist, werden werden alle richtigen Lösungen, gleich in welcher Form, akzeptiert)

```
Lies a, b, c, h
wenn a>b, dann    h ← a      h ← a gleichwertig mit h=a
                  a ← b
                  b ← h
wenn b>c, dann    h ← b
                  b ← c
                  c ← h
```

- 2) Schreiben Sie ein Java-Programm, das die x- und y-Koordinaten zweier Punkte einliest und den Abstand zwischen ihnen berechnet und ausgibt.
Hinweis: Die ganzzahlige Wurzel einer Zahl x können Sie als `(int)Math.sqrt(x)` berechnen.

```
public class abstand {
    public static void main(String[] arg) {
        int x1=In.readInt();
        int y1=In.readInt();
        int x2=In.readInt();
        int y2=In.readInt();
        int abstand;

        abstand=(int) Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))

        Out.println("Abstand von P1-P2: "+abstand)
    }
}
```

- 3) Herr Meyer und Herr Schmidt sind Angestellte in derselben Firma. Herr Meyer erhält 2400€ Gehalt, Herr Schmidt 2750€. Bei einer Personalversammlung haben sie sich zwischen einer Gehaltserhöhung prozentual 4,5% und einem Festbetrag von 100€ zu entscheiden. Planen und schreiben Sie ein Javaprogramm, das nach Eingabe von Gehalt und Prozentsatz die Erhöhung ausgibt.

Planung:

Eingaben: Gehalt, Prozentsatz → int gehalt, float prozentsatz

Verarbeitung: $erhoehung = \text{gehalt} * \text{prozentsatz} / 100$

Ausgabe: Erhöhung → float erhoehung

```
public class gehaltserhoehung{
    public static void main(String[] arg) {
        int gehalt =In.readInt();
        float prozentsatz=In.readFloat();
        float erhoehung;

        erhoehung=gehalt*prozentsatz/100;

        Out.println("Die Erhoehung betraegt = "+erhoehung);
    }
}
```