

## Sortieren sowie Suchen mit dem Halbierungsverfahren, programmiert mit der Java-AWT

Funktion und Oberfläche sind in getrennten Klassen untergebracht. Die ganze Anwendung besteht damit aus drei Dateien:

### Sortier- und Such-Routinen

```
1 // Anwendung "SchönSort"
2 // Klasse SchoenSort_Funktion nur mit Sortierfunktionalität;
3 // Ein-/Ausgabe ausgelagert in eigene, andere Klasse
4 // Java jdk 1.1.18 -- R. Krell, 13.11.01
5
6 import stiftUndCo.*; // für Zufallsgenerator
7
8 public class SchoenSort_Funktion
9 {
10     int[] feld; // Deklaration...
11
12     public void zufälligFüllen (int länge)
13     // erzeugt feld in der gewünschten Länge und
14     // füllt feld mit zufälligen Werten zwischen 10 und 99
15     {
16         feld = new int[länge];
17         for (int i=0; i < feld.length; i++)
18         {
19             feld[i] = Hilfe.zufall(10,100);
20             // oder -- ohne stiftUndCo.*, aber mit import java.util.*;
21             // und lokaler Definition Random zufall = new Random();
22             // dann feld[i] = 10 + Math.abs (zufall.nextInt()) % 90;
23         }
24     }
25
26     public boolean istSortiert()
27     // wird wahr, wenn das feld aufsteigend sortiert ist
28     {
29         boolean okay = true;
30         int stelle = 0;
31         do
32         {
33             if (feld[stelle] > feld[stelle+1])
34             {
35                 okay = false;
36             }
37             stelle = stelle + 1;
38         } while (okay && (stelle < feld.length-1));
39         return (okay);
40     }
41
42     public String zeigeReihe ()
43     // erzeugt eine Zeichenkette mit allen Elementen vom feld.
44     // Schreibt nicht mehr direkt auf den Bildschirm!
45     {
46         String zchnkette = new String();
47         zchnkette = "";
48         for (int i=0; i < feld.length; i++)
49         {
50             zchnkette = zchnkette + " " +feld[i];
51         }
52     }
53 }
```

```

52     return (zchnkette);
53 }
54
55 private void tausche (int i, int j)
56 // vertauscht innerhalb vom feld die beiden Inhalte an den
57 // Positionen i und j. Wird von minSort benutzt.
58 {
59     int zwischen = feld[i];
60     feld[i] = feld[j];
61     feld[j] = zwischen;
62 }
63
64 public void minSort() // MinSort
65 // Sortieren durch Auswahl des jeweils kleinsten Elements im Rest
66 // und Tausch dieses Elements nach vorne. Nach dem 0. Durchgang steht
67 // also das kleinste Element ganz links auf Platz 0, beim nächsten Durchgang
68 // kommt das nächstkleinere Element auf Platz 1 usw., so dass die Sortierung
69 // von links nach rechts im feld wächst. Aufwand ~ (feld.length)^2
70 {
71     int minStelle, durchgang, stelle;
72
73     for (durchgang=0; durchgang<feld.length-1; durchgang++)
74     {
75         minStelle = durchgang; // vorderstes El. im unsortierten Teil zum Vergleich
76         for (stelle=durchgang+1; stelle<feld.length; stelle++)
77         {
78             if (feld[stelle] < feld[minStelle])
79             {
80                 // wird ein kleineres Element gefunden, so wird dessen Index gemerkt
81                 minStelle = stelle;
82             }
83         }
84         // das gefundene kleinste Element wird nach vorne (auf den Platz mit
85         // dem Index durchgang) getauscht, sofern es nicht schon dort steht:
86         if (minStelle > durchgang)
87         {
88             tausche (durchgang, minStelle);
89         }
90     }
91 }
92
93 public int finde (int zahl)
94 // sucht mit dem Halbierungsverfahren. Ist die gesuchte Zahl im
95 // feld, wird ihre Position genannt. Gibt's die Zahl nicht, ist der
96 // Funktionswert -1. Gesucht wird immer zwischen links und rechts;
97 // dieser Bereich wird ständig halbiert. Aufwand ~ log2 (feld.length).
98 {
99     int links = 0; //Immer: feld[links] <= zahl <= feld[rechts]
100    int rechts = feld.length - 1; //Anfangs mit ganzem feld
101    int mitte;
102    do {
103        mitte = (links + rechts + 1) / 2;
104        if (zahl < feld[mitte])
105        {
106            rechts = mitte - 1; // zahl liegt in linker "Hälfte"
107        }
108        else
109        {
110            links = mitte; // zahl liegt in rechter "Hälfte"
111        }
112    } while (links<rechts);
113    if (zahl==feld[links]) // oder feld[rechts]; nur mitte ist nicht aktuell

```

```

114     {
115         return (links); // zahl gefunden: Index als Funktionswert übergeben
116     }
117     else
118     {
119         return (-1); // zahl nicht gefunden: -1 als Fkts.-wert
120     }
121 }
122 }
123 }
124 }
125 }
126 }

```

---

## Oberfläche

```

1 // Anwendung "SchönSort"
2 // Klasse SchoenSort_Oberflaeche ohne Sortierfunktionalität;
3 // benutzt bzw. deklariert und erzeugt dafür Objekt reihung
4 // Java jdk 1.1.18 -- R. Krell, 13.11.01
5
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class SchoenSort_Oberflaeche
10     extends Frame // Frame ist Fensterklasse
11 {
12     SchoenSort_Funktion reihung = new SchoenSort_Funktion();
13     int anzahl = 10;
14     int gesucht;
15
16     Label lbÜberschrift, lbAnzahl, lbSuche, lbSuchergebnis;
17     Button btErzeuge, btSortiere;
18     TextField tfAnzahl, tfSuchzahl;
19     TextArea taAusgabe;
20
21     public void richteFensterEin() // Fenster initialisieren und beschreiben
22     {
23         //WindowListener hinzufügen, damit Schließknopf funktioniert
24         addWindowListener (
25             new WindowAdapter ()
26             {
27                 public void windowClosing (WindowEvent ereignis)
28                 { //ersetzt bisher leere Methode
29                     setVisible (false);
30                     dispose();
31                     System.exit(0);
32                 }
33             }
34         ); // runde Klammer vom Windowlistener geschlossen;
35
36         setTitle("Sortieren und Finden ganzer Zahlen"); // Fenster betiteln
37         setSize (420,400); // Fenstergröße festlegen
38     }
39
40     public void beschrifteLabel() // Anfangsbeschriftungen
41     {
42         lbÜberschrift = new Label("Binäre Suche in einer sortierten Reihung");

```

```

43     lbAnzahl = new Label("Anzahl der Zahlen (5..20):");
44     lbSuche = new Label("Suche nach:");
45     lbSuchergebnis = new Label(" ??? gefunden an Stelle ??");
46 }
47
48 public void richteKnöpfeEin()
49 {
50     // Knöpfe definieren und beschriften
51     btErzeuge = new Button ("Zufallszahlen");
52     btSortiere = new Button ("Sortieren");
53
54     //Funktion der Knöpfe festlegen
55     btErzeuge.addActionListener (
56         new ActionListener ()
57         {
58             public void actionPerformed (ActionEvent e)
59             {
60                 reihung.zufälligFüllen (anzahl);
61                 taAusgabe.append("Zufällig:\n"+reihung.zeigeReihe()+"\n");
62             }
63         }); // runde Klammer (von addActionListener)
64
65     btSortiere.addActionListener (
66         new ActionListener ()
67         {
68             public void actionPerformed (ActionEvent e)
69             {
70                 reihung.minSort();
71                 taAusgabe.append("Sortiert:\n"+reihung.zeigeReihe()+"\n");
72             }
73         }); // runde Klammer (von addActionListener)
74 }
75
76 public void richteEingabezeilenEin ()
77 {
78     tfAnzahl = new TextField(""+anzahl,2);
79     tfSuchzahl = new TextField("",3);
80
81     tfAnzahl.addActionListener (
82         new ActionListener ()
83         {
84             public void actionPerformed (ActionEvent e)
85             {
86                 int anz = Integer.parseInt (tfAnzahl.getText());
87                 if ((anz>=5) && (anz<=20))
88                 {
89                     anzahl = anz;
90                 }
91                 else
92                 {
93                     tfAnzahl.setText (""+anzahl);
94                 }
95             }
96         }); // runde Klammer (von addActionListener)
97
98     tfSuchzahl.addActionListener (
99         new ActionListener ()
100        {
101            public void actionPerformed (ActionEvent e)
102            {
103                int gesucht = Integer.parseInt (tfSuchzahl.getText());
104                int index = reihung.finde(gesucht);

```

```

105         if (index<0)
106         {
107             lbSuchergebnis.setText(" "+gesucht+" nicht gefunden!");
108         }
109         else
110         {
111             lbSuchergebnis.setText(" "+gesucht+" gefunden an Stelle "+index);
112         }
113         tfSuchzahl.setText("");
114     }
115     }); // runde Klammer (von addActionListener)
116 }
117
118 public void richteAusgabeEin ()
119 {
120     taAusgabe = new TextArea("Anzeige der Reihung:\n\n",100,6);
121 }
122
123 public void ordneBildschirmobjekteAn ()
124 {
125     Panel p1 = new Panel();
126     Panel p2 = new Panel();
127     setLayout (new GridLayout(4,0)); // innere Unterteilung des Fensters
128     add (lbÜberschrift);
129     // p1.setLayout (new FlowLayout());
130     p1.add (lbAnzahl);
131     p1.add (tfAnzahl);
132     p1.add (btErzeuge);
133     p1.add (btSortiere);
134     add (p1);
135     add (taAusgabe);
136     // p2.setLayout (new FlowLayout());
137     p2.add (lbSuche);
138     p2.add (tfSuchzahl);
139     p2.add (lbSuchergebnis);
140     add (p2);
141     setVisible(true);
142 }
143
144 public void führeAus()
145 {
146     richteFensterEin();
147     beschrifteLabel();
148     richteKnöpfeEin();
149     richteEingabezeilenEin();
150     richteAusgabeEin();
151     ordneBildschirmobjekteAn();
152 }
153 }

```

---

#### Startdatei

```

1 public class SchoenSort_Start
2 {
3     public static void main (String[] s)
4     {
5         SchoenSort_Oberflaeche so;
6         so = new SchoenSort_Oberflaeche();
7         so.führeAus();

```

8 }  
9 }

Die Anwendung hat dann folgendes Aussehen:

