

## Datei „IntSortReihe2.java“

```
1 // Verschiedene einfache Sortierverfahren + BucketSort
2 // Der (Zeit-)Aufwand nimmt mit (reihung.length)^2 zu!
3 // Java jdk 1.1.18 -- R. Krell, 12.10.01
4 // ohne stiftUndCo -- Ein-/Ausgabe per Konsole bzw. Dos-Box;
5 // ** Start aus Dos-Box mit "java IntSortReiheStart" <Eingabe> **
6
7 import java.io.*; // für Ein-/Ausgabe
8 import java.util.*; // für Zufallszahlengenerator
9
10 public class IntSortReihe2
11 {
12     // Deklaration global benötigter Objekte und der Variablen erzeugt
13
14     int[] reihung, kopie;
15     boolean erzeugt = false;
16     Random zufall = new Random(); // neuer Zufallszahlengenerator
17
18     public void erzeuge()
19     // Initialisierungsmethode: erzeugt alle benötigten Objekte einmal
20     {
21         if (! erzeugt) // kein neues Erzeugen bei versehentlichem Mehrfachaufruf
22         {
23             reihung = new int[11]; // nimmt die zu sortierenden Zahlen auf
24             kopie = new int[11]; // Kopie der reihung, um gleiche Zahlen
25                                 // mit versch. Methoden sortieren zu können
26             erzeugt = true;
27         }
28     }
29
30     public void fülle()
31     // Menü: erlaubt verschiedene Füllungen von reihung und kopie
32     {
33         BufferedReader tastatur = new BufferedReader(new InputStreamReader(System.in));
34         //^ Vorbereiten der Lesemöglichkeit von der Tastatur
35         String eingabe = new String();
36         char wunsch;
37         boolean gewählt;
38
39         System.out.println ("Reihung von "+reihung.length+" ganzen Zahlen");
40         System.out.print("Füllung: A = Aufsteigend, B = absteigend, Z = Zufällig. (A,B,Z + <Eingabe>)? ");
41         gewählt = true;
42         do {
43             gewählt = true;
44             try //try-catch = Konstruktion zur Fehlerbehandlung
45             {
46                 eingabe = tastatur.readLine(); // Eingabestring lesen (alle Zeichen vor <Eingabe>-Taste)
47             }
48             catch (Exception dummy) // falls Eingabe nicht gelesen werden konnte
49             {
50                 System.out.print (" ??");
51             }
52             wunsch = (eingabe+" ").charAt(0); //erstes Zeichen der Eingabe verwenden
53
54             switch (wunsch)
55             {
56                 case 'a' : aufsteigendFüllen(); break;
57                 case 'b' : absteigendFüllen(); break;
58                 case 'z' : zufälligFüllen(); break;
59                 default : gewählt = false;
60             }
61         } while (! gewählt);
62         for (int i=0; i < reihung.length; i++)
63             {kopie[i] = reihung[i];} // kopie = reihung identifiziert beide Objekte, so dass sich
```

```

64     // künftige Änderungen auf reihung immer auch auf kopie auswirken -- was ja nicht gewollt ist
65 }
66
67 private void aufsteigendFüllen ()
68 // füllt die reihung so, dass zwischen Nachbarn <= gilt
69 {
70     reihung[0] = 10 + Math.abs (zufall.nextInt()) % 10; // Zufallszahl zw. 10 und 19
71     for (int i=1; i < reihung.length; i++)
72     {
73         reihung[i] = reihung [i-1] + Math.abs (zufall.nextInt()) % 6;
74     }
75 }
76
77 private void absteigendFüllen ()
78 // füllt die reihung so, dass zwischen Nachbarn >= gilt
79 {
80     reihung[0] = 90 + Math.abs (zufall.nextInt()) % 10; // Zufallszahl zw. 90 und 99
81     for (int i=1; i < reihung.length; i++)
82     {
83         reihung[i] = reihung [i-1] - Math.abs (zufall.nextInt()) % 6;
84     }
85 }
86
87 private void zufälligFüllen ()
88 // füllt die reihung mit völlig zufälligen Werten zwischen 10 und 99
89 {
90     for (int i=0; i < reihung.length; i++)
91     {
92         reihung[i] = 10 + Math.abs (zufall.nextInt()) % 90; // Zufallszahl zw. 10 und 99
93     }
94 }
95
96 public boolean istSortiert()
97 // wird wahr, wenn die reihung aufsteigend sortiert ist
98 {
99     boolean okay = true;
100    int stelle = 0;
101    do
102    {
103        if (reihung[stelle] > reihung[stelle+1])
104        {
105            okay = false;
106        }
107        stelle = stelle + 1;
108    } while (okay && (stelle < reihung.length-1));
109    return (okay);
110 }
111
112 private void zeigeAlle ()
113 // schreibt die Elemente der reihung nach-/nebeneinander auf den Bildschirm
114 {
115     System.out.println ("Reihung: ");
116     for (int i=0; i < reihung.length; i++)
117     {
118         System.out.print(" "+reihung[i]);
119     }
120     System.out.println("");
121 }
122
123 private void tausche (int i, int j)
124 // vertauscht innerhalb der reihung die beiden Inhalte an den Positionen i und j
125 // Methode wird von den meisten Sortierverfahren (s.u.) benutzt
126 {
127     int zwischen; // hier int -- immer vom Typ der Komponenten der reihung!
128
129     zwischen = reihung[i];
130     reihung[i] = reihung[j];
131     reihung[j] = zwischen;
132 }

```

```

133
134 public void bubbleSort() // BubbleSort, 1. Version
135 // Sortieren durch Vertauschen unmittelbarer Nachbarn
136 // beim ersten Durchgang "perlt" die größte Zahl nach hinten, so dass
137 // spätere Durchgänge immer ein Element früher enden können und so
138 // der sortierte Teil von hinten nach vorne wächst
139 {
140     int durchgang, stelle;
141
142     for (durchgang=1; durchgang<reihung.length; durchgang++)
143         // hier mit fester Durchgangszahl = Elementzahl - 1
144         {
145             for (stelle=0; stelle<reihung.length-durchgang; stelle++)
146                 // Durchgänge beginnen immer bei 0, enden aber immer früher
147                 {
148                     if (reihung[stelle] > reihung[stelle+1])
149                     {
150                         // Nachbarn in falscher Reihenfolge werden vertauscht
151                         tausche (stelle, stelle+1);
152                     }
153                 }
154         }
155     }
156
157 public void bubbleSort2() // BubbleSort, verbesserte Version
158 // Sortieren durch Vertauschen unmittelbarer Nachbarn
159 // wie vor, allerdings: wurde in einem Durchgang nichts mehr vertauscht,
160 // so sind keine weiteren Durchgänge nötig und es ist alles sortiert.
161 // NB: War anfangs hinten die kleinste Zahl, sind alle Durchgänge nötig!
162 {
163     int durchgang, stelle;
164     boolean vertauscht;
165
166     durchgang=1;
167     do {
168         vertauscht = false; // in diesem Durchgang wurde noch nicht getauscht
169         for (stelle=0; stelle<reihung.length-durchgang; stelle++)
170             {
171                 if (reihung[stelle] > reihung[stelle+1])
172                 {
173                     tausche (stelle, stelle+1);
174                     vertauscht = true; // jetzt hat ein Tausch stattgefunden
175                 }
176             }
177         durchgang = durchgang + 1;
178     } while (vertauscht && (durchgang<reihung.length));
179     // höchstens so viele Durchgänge wie beim normalen bubbleSort;
180     // aber vorzeitiges Ende, wenn zuletzt nichts mehr vertauscht wurde
181 }
182
183
184 public void shakerSort() // ShakerSort
185 // wie BubbleSort, aber mit Durchgängen abwechselnd in beide Richtungen:
186 // beim Durchgang nach rechts "perlt" die größte Zahl nach oben
187 // beim Durchgang nach links "perlt" jetzt die kleinste Zahl nach unten,
188 // so dass möglicherweise noch mehr Durchgänge gespart werden können
189 {
190     int durchgang, stelle;
191     boolean vertauscht;
192
193     durchgang=1;
194     do {
195         vertauscht = false;
196         // Durchgang nach rechts:
197         for (stelle=durchgang-1; stelle<reihung.length-durchgang; stelle++)
198             {
199                 if (reihung[stelle] > reihung[stelle+1])
200                 {
201                     tausche (stelle, stelle+1);

```

```

202         vertauscht = true;
203     }
204 }
205 if (vertauscht)
206 {
207     // falls noch nötig, Durchgang nach links
208     vertauscht = false;
209     for (stelle=reihung.length-durchgang-2; stelle>=durchgang-1; stelle--)
210     {
211         if (reihung[stelle] > reihung[stelle+1])
212         {
213             tausche (stelle, stelle+1);
214             vertauscht = true;
215         }
216     }
217 }
218 durchgang = durchgang + 1; // Hin- und Zurück = 1 (Doppel-)Durchgang
219 } while (vertauscht && (durchgang<=reihung.length/2));
220 }
221
222 public void minSort() // MinSort
223 // Sortieren durch Auswahl des jeweils kleinsten Elements im Rest
224 // und Tausch dieses Elements nach vorne. Nach dem 0. Durchgang steht
225 // also das kleinste Element ganz links auf Platz 0, beim nächsten Durchgang
226 // kommt das nächstkleinere Element auf Platz 1 usw., so dass die Sortierung
227 // von links nach rechts wächst
228 {
229     int minStelle, durchgang, stelle;
230
231     for (durchgang=0; durchgang<reihung.length-1; durchgang++)
232     {
233         minStelle = durchgang; // vorderstes Element im unsortierten Teil zum Vergleich
234         for (stelle=durchgang+1; stelle<reihung.length; stelle++)
235         {
236             if (reihung[stelle] < reihung[minStelle])
237             {
238                 // wird ein kleineres Element gefunden, so wird dessen Index gemerkt
239                 minStelle = stelle;
240             }
241         }
242         // das gefundene kleinste Element wird nach vorne (auf den Platz mit
243         // dem Index durchgang) getauscht, sofern es nicht schon dort steht:
244         if (minStelle > durchgang)
245         {
246             tausche (durchgang, minStelle);
247         }
248     }
249 }
250
251 public void einSort() // EinSort
252 // in den hinteren, sortierten Teil der reihung wird jeweils das letzte Element
253 // davor richtig eingefügt, wodurch der sortierte Teil nach vorne wächst.
254 // Damit Platz zum Einfügen ist, werden die sortierten Elemente solange
255 // um eins nach vorne gezogen, bis das neue Element in die Lücke gehört.
256 {
257     int sortAb, stelle;
258     int zwischen; // hier int -- immer vom Typ der Komponenten der Reihung
259     boolean aufhören;
260
261     sortAb = reihung.length-1; // anfangs gilt nur das letzte Element als sortiert
262     do
263     {
264         zwischen = reihung[sortAb-1]; // letztes unsortiertes Element rausnehmen
265
266         aufhören = false;
267         stelle = sortAb;
268         do
269         {
270             if (zwischen > reihung[stelle])

```

```

271     {
272         reihung[stelle-1] = reihung[stelle]; // Nachfolgende Inhalte nach vorne schieben
273         stelle = stelle + 1;
274     }
275     else
276     {
277         aufhören = true;
278     }
279 } while ((aufhören==false)&&(stelle<reihung.length));
280
281 reihung[stelle-1] = zwischen; // Element an richtiger Stelle rein
282 sortAb = sortAb - 1; // sortierter Teil ist um eins nach vorne gewachsen
283
284 } while (sortAb > 0); // äußere do-Schleife
285 }
286
287 public void bucketSort() // Bucketsort
288 // Kein einfaches Sortierverfahren, da zusätzlicher Speicherplatz für Kontrolle
289 // nötig. Zählt in Kontrolle, ob/wie oft ein Element der reihung auftrat und
290 // gibt dann in Schleife 3 die Sortierung aus.
291 // Wichtig: Verfahren funktioniert nur, wenn die in der Reihung auftretenden
292 // Werte im Indexbereich von Kontrolle -- hier also zwischen 0 und 199 -- liegen!
293 // Dafür wächst der Zeitaufwand höchstens linear mit reihung.length!
294 {
295     int[] kontrolle = new int[200];
296     int stelle;
297
298     for (int index=0; index<kontrolle.length; index++)
299     {
300         kontrolle[index] = 0; //Schleife 1: Initialisiert Kontrolle
301     }
302     for (stelle=0; stelle<reihung.length; stelle++)
303     {
304         kontrolle [reihung[stelle]] = kontrolle [reihung[stelle]] + 1;
305     }
306     stelle = 0;
307     for (int index=0; index<kontrolle.length; index++)
308     {
309         //Schleife 3: füllt Sortierung in die reihung
310         for (int zähler=1; zähler<=kontrolle[index]; zähler++)
311         {
312             reihung[stelle] = index; //mehrfach gezählte Werte müssen auch
313             stelle = stelle + 1; //wieder mehrfach in die reihung!
314         }
315     }
316 }
317
318
319 /* public void bucketsort() //falls Werte nicht mehrfach auftreten dürfen
320 {
321     boolean[] kontrolle = new boolean[100];
322     for (int index=0; index<100; index++) // Schleife 1
323     {
324         kontrolle[index] = false;
325     }
326     for (int stelle=0; stelle<reihung.length; stelle++) // Schleife 2
327     {
328         kontrolle [reihung[stelle]] = true;
329     }
330     for (int index=0; index<100; index++) // Schleife 3
331     {
332         if (kontrolle[index]==true)
333         {
334             stift.schreibe (" "+index);
335         }
336     }
337 }
338 */

```

```

339
340 public void demo ()
341 {
342     BufferedReader tastatur = new BufferedReader(new InputStreamReader(System.in));
343     String eingabe = " ";
344     char wunsch;
345     boolean gewählt;
346
347     erzeuge();
348     fülle();
349     do {
350         zeigeAlle();
351         System.out.println ("");
352         System.out.println ("1=BubbleSort, 2=BubbleSort2, 3=ShakerSort, 4=MinSort,
353         5=EinSort, 6=BucketSort.");
354         System.out.print  ("Wunsch: 1..6=Sortiere, P=Prüfe, O=Orig., N=neu, E=Ende.
355         (1..6,P,O,N,E)? ");
356     do {
357         gewählt = true;
358         try // Kommentar s.o. bei fülle!
359         {
360             eingabe = tastatur.readLine();
361         }
362         catch (Exception dummy)
363         {
364             System.out.print (" ??");
365         }
366         wunsch = (eingabe+" ").charAt(0);
367
368         switch (wunsch)
369         {
370             case '1' : bubbleSort(); break;
371             case '2' : bubbleSort2 (); break;
372             case '3' : shakerSort(); break;
373             case '4' : minSort(); break;
374             case '5' : einSort(); break;
375             case '6' : bucketSort(); break;
376             case 'p' : System.out.println ("Die Prüfung ergab: sortiert="
377                 +istSortiert());
378                 break;
379             case 'o' : for (int i=0; i < reihung.length; i++)
380                 { reihung[i] = kopie[i]; }
381                 break;
382             case 'n' : fülle(); break;
383             case 'e' : ; break;
384             default : gewählt = false;
385         }
386     } while (! gewählt);
387 } while (wunsch != 'e');
388 }
389 }

```

---

#### Datei „IntSortReiheStart.java“

```

1 public class IntSortReiheStart
2 {
3     public static void main (String[] s)
4     {
5         IntSortReihe2 r;
6         r = new IntSortReihe2();
7         r.demo();
8     }
9 }

```