Datei „IntSortReihe1.java“

```
1  // Verschiedene einfache Sortierverfahren + BucketSort
2  // Der (Zeit-)Aufwand nimmt mit (reihung.length)^2 zu!
3  // Java jdk 1.1.18 und stiftUndCo -- R. Krell, 12.10.01
4
5  import stiftUndCo.*;
6
7  public class IntSortReihe1
8  {
9    // Deklaration global benötigter Objekte und der Variablen erzeugt
10   Bildschirm schirm;
11   Tastatur taste;
12   BuntStift stift;
13   int[] reihung, kopie;
14   boolean erzeugt = false;
15
16   public void erzeuge()
17   // Initalisierungsmethode: erzeugt alle benötigten Objekte einmal
18   {
19     if (! erzeugt) // kein neues Erzeugen bei versehentlichem Mehrfachaufruf
20     {
21       schirm = new Bildschirm(900,600);
22       taste = new Tastatur();
23       stift = new BuntStift();
24       reihung = new int[11];  // nimmt die zu sortierenden Zahlen auf
25       kopie = new int[11];    // Kopie der reihung, um gleiche Zahlen mit versch.
26 Methoden sortieren zu können
27       erzeugt = true;
28     }
29   }
30
31   public void fülle()
32   // Menü: erlaubt verschiedene Füllungen von reihung und kopie
33   {
34     char wunsch;
35     boolean gewählt;
36
37     schirm.löscheAlles();
38     stift.bewegeBis (10,20);
39     stift.schreibe ("Reihung von "+reihung.length+" ganzen Zahlen");
40     stift.bewegeBis (10,40);
41     stift.schreibe ("Füllung: A = Aufsteigend, B = aBsteigend, Z = Zufällig. (A,B,Z)? ");
42     gewählt = true;
43     do {
44       while (! taste.wurdeGedrueckt())
45         {};
46       gewählt = true;
47       wunsch = taste.zeichen();
48       switch (wunsch)
49       {
50         case 'a' : aufsteigendFüllen(); break;
51         case 'b' : absteigendFüllen(); break;
52         case 'z' : zufälligFüllen(); break;
53         default  : gewählt = false;
54       }
55     } while (! gewählt);
56     stift.schreibe (wunsch);
57     for (int i=0; i < reihung.length; i++)
58       {kopie[i] = reihung[i];} // kopie = reihung identifiziert beide Objekte, so dass sich
59      // künftige Änderungen auf reihung immer auch auf kopie auswirken -- was ja nicht gewollt ist
60   }
61
62   private void aufsteigendFüllen ()
63   // füllt die reihung so, dass zwischen Nachbarn <= gilt
64   {
```

```
65        reihung[0] = Hilfe.zufall(10,20);
66        for (int i=1; i < reihung.length; i++)
67        {
68          reihung[i] = reihung [i-1] + Hilfe.zufall(0,6);
69        }
70      }
71
72      private void absteigendFüllen ()
73      // füllt die reihung so, dass zwischen Nachbarn >= gilt
74      {
75        reihung[0] = Hilfe.zufall(80,100);
76        for (int i=1; i < reihung.length; i++)
77        {
78          reihung[i] = reihung [i-1] - Hilfe.zufall(0,6);
79        }
80      }
81
82      private void zufälligFüllen ()
83      // füllt die reihung mit völlig zufälligen Werten zwischen 10 und 99
84      {
85        for (int i=0; i < reihung.length; i++)
86        {
87          reihung[i] = Hilfe.zufall(10,100);
88        }
89      }
90
91      public boolean istSortiert()
92      // wird wahr, wenn die reihung aufsteigend sortiert ist
93      {
94        boolean okay = true;
95        int stelle = 0;
96        do
97        {
98          if (reihung[stelle] > reihung[stelle+1])
99          {
100           okay = false;
101         }
102         stelle = stelle + 1;
103       } while (okay && (stelle < reihung.length-1));
104       return (okay);
105     }
106
107     private void zeigeAlle ()
108     // schreibt die Elemente der reihung nach-/nebeneinander auf den Bildschirm
109     {
110       stift.bewegeBis (10, stift.vPosition() + 12);
111       stift.schreibe ("Reihung: ");
112       for (int i=0; i < reihung.length; i++)
113       {
114         stift.schreibe("  "+reihung[i]);
115       }
116     }
117
118     private void tausche (int i, int j)
119     // vertauscht innerhalb der reihung die beiden Inhalte an den Positionen i und j
120     // Methode wird von den meisten Sortierverfahren (s.u.) benutzt
121     {
122       int zwischen;  // hier int -- immer vom Typ der Komponenten der reihung!
123
124       zwischen = reihung[i];
125       reihung[i] = reihung[j];
126       reihung[j] = zwischen;
127     }
128
129     public void bubbleSort() // BubbleSort, 1. Version
130     // Sortieren durch Vertauschen unmittelbarer Nachbarn
131     // beim ersten Durchgang "perlt" die größte Zahl nach hinten, so dass
132     // spätere Durtchgänge immer ein Element früher enden können und so
```

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

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

```
271           {
272             aufhören = true;
273           }
274         } while ((aufhören==false)&&(stelle<reihung.length));
275
276         reihung[stelle-1] = zwischen; // Element an richtiger Stelle rein
277         sortAb = sortAb - 1; // sortierter Teil ist um eins nach vorne gewachsen
278
279       } while (sortAb > 0); // äußere do-Schleife
280     }
281
282   public void bucketSort() // Bucketsort
283     // Kein einfaches Sortierverfahren, da zusätzlicher Speicherplatz für konmtrolle
284     // nötig. Zählt in kontrolle, ob/wie oft ein Element der reihung auftrat und
285     // gibt dann in Schleife 3 die Sortierung aus.
286     // Wichtig: Verfahren funktioniert nur, wenn die in der Reihung auftretenden
287     // Werte im Indexbereich von kontrolle -- hier also zwischen 0 und 199 -- liegen!
288     // Dafür wächst der Zeitaufwand höchstens linear mit reihung.length!
289     {
290       int[] kontrolle = new int[200];
291       int stelle;
292
293       for (int index=0; index<kontrolle.length; index++)
294       {                              //Schleife 1: Initialisiert kontrolle
295         kontrolle[index] = 0;
296       }
297       for (stelle=0; stelle<reihung.length; stelle++)
298       {                              //Schleife 2: zählt Werte aus reihung
299         kontrolle [reihung[stelle]] = kontrolle [reihung[stelle]] + 1;
300       }
301       stelle = 0;
302       for (int index=0; index<kontrolle.length; index++)
303       {                              //Schleife 3: füllt Sortierung in die reihung
304
305         for (int zähler=1; zähler<=kontrolle[index]; zähler++)
306         {                                  //mehrfach gezählte Werte müssen auch
307           reihung[stelle] = index;         //wieder mehrfach in die reihung!
308           stelle = stelle + 1;
309         }
310       }
311     }
312
313
314   /* public void bucketsort() //falls Werte nicht mehrfach auftreten dürfen
315     {
316       boolean[] kontrolle = new boolean[100];
317       for (int index=0; index<100; index++)  // Schleife 1
318       {
319         kontrolle[index] = false;
320       }
321       for (int stelle=0; stelle<reihung.length; stelle++) // Schleife 2
322       {
323         kontrolle [reihung[stelle]] = true;
324       }
325       for (int index=0; index<100; index++)  // Schleife 3
326       {
327         if (kontrolle[index]==true)
328         {
329           stift.schreibe (" "+index);
330         }
331       }
332     }
333   */
334
335     public void demo ()
336     {
337       char wunsch;
338       boolean gewählt;
339
```

```
340       erzeuge();
341       fülle();
342       do {
343         zeigeAlle();
344         stift.bewegeBis (10,stift.vPosition()+20);
345         stift.schreibe ("1=BubbleSort, 2=BubbleSort2, 3=ShakerSort,
346           4=MinSort, 5=EinSort, 6=BucketSort.");
347         stift.bewegeBis (10,stift.vPosition()+10);
348         stift.schreibe ("Wunsch: 1..6=Sortiere, P=Prüfe, O=Orig., N=neu, E=Ende.
349           (1..6,P,O,N,E)? ");
350         do {
351           while (! taste.wurdeGedrueckt())
352             {};
353           gewählt = true;
354           wunsch = taste.zeichen();
355           stift.schreibe (wunsch);
356           switch (wunsch)
357           {
358             case '1' : bubbleSort(); break;
359             case '2' : bubbleSort2 (); break;
360             case '3' : shakerSort(); break;
361             case '4' : minSort(); break;
362             case '5' : einSort(); break;
363             case '6' : bucketSort(); break;
364             case 'p' : stift.bewegeBis(10, stift.vPosition()+10);
365                        stift.schreibe ("Die Prüfung ergab: sortiert="
366                         +istSortiert());
367                        break;
368             case 'o' : for (int i=0; i < reihung.length; i++)
369                        { reihung[i] = kopie[i]; }
370                        break;
371             case 'n' : fülle(); break;
372             case 'e' : ; break;
373             default  : gewählt = false;
374           }
375         } while (! gewählt);
376       } while (wunsch != 'e');
377       stift.gibFrei();
378       schirm.gibFrei();
379     }
380   }
```

Datei „IntSortReiheStart.java“

```
1   public class IntSortReiheStart
2   {
3     public static void main (String[] s)
4     {
5       IntSortReihe1 r;
6       r = new IntSortReihe1();
7       r.demo();
8     }
9   }
```