Aufgabe 19 - Zu erstellen sind zwei Klassen: *Sortieren* & *Test*

Klasse *Sortieren*:

Sortieren
- zahlen : float[]
+ Sortieren(groesse : int , von : float, bis : float) + Sortieren(zahlenarray : float[]) - generiereZufallszahlen(von : float, bis : float) +bubbleSortAufsteigend() : int + bubbleSortAbsteigend() : int + bubbleSort(aufsteigend : boolean) : int // optional - tauschen(pos1 : int, pos2 : int) + ausgeben() + shuffle(anzahl : int) + …Sort() : int // weiterer Sortieralgorithmus

zahlen: ist ein Array von ***float***-Werten.

1. Konstruktor: Erzeugt ein Array in der angegeben Größe und initialisiert es mit Zufallszahlen (Gleitkommazahlen) im angegebenen Bereich.

2. Konstruktor: Erstellt ein Array das gleich groß ist wie das übergebene und kopiert die Werte.

generiereZufallszahlen(von : float, bis : float) befüllt das Array *zahlen* mit zufällig generierten Werten im angegeben Bereich.

Im Unterricht haben wir den BubbleSort Algorithmus besprochen und auch eine erste Optimierung (jeder Druchlauf endet ein Feld früher) durchgeführt. Hier der **nicht** optimierte Quelltext:

```
public void bubbleSortAufsteigend()
{
    float temp=0;
    for (int d=0; d<zahlen.length-1; d++)    // Durchläufe
    {
        // je Durchlauf:
        for (int i=0; i<zahlen.length-1; i++)
        {
            // benachbarte Felder vergleichen:
            if (zahlen[i] > zahlen[i+1])
            {
                // wenn nicht in richtiger Reihenfolge, tauschen:
                temp = zahlen[i];
                zahlen[i] = zahlen[i+1];
                zahlen[i+1] = temp;
            }
        }
    }
}
```

bubbleSortAufsteigend() : int sortiert die Werte aufsteigend mittels BubbleSort-Algorithmus und zählt die für das Sortieren durchgeführten Zahlenvergleiche (= Aufwand … Rückgabewert).

<u>Realisiere folgende Optimierungen:</u>

- Obiger Algorithmus ist nicht optimal, weil bei jedem Durchlauf immer alle Felder verglichen werden. Ergänze die fehlende Optimierung! (wurde im Unterricht gezeigt)
- Wenn in einem Durchlauf kein Tausch zweier Zahlen mehr stattfindet, dann ist es nicht notwendig weitere Durchläufe auszuführen, die Zahlen sind bereits vollständig sortiert.

bubbleSortAbsteigend() : int sortiert die Werte absteigend mittels BubbleSort-Algorithmus und zählt wiederum die für das Sortieren durchgeführten Zahlenvergleiche (= Rückgabewert)…

Wer möchte, versucht bitte beide Methoden zu einer zusammenzufassen:
bubbleSort(aufsteigend : boolean) : int (*aufsteigend==false* → es wird absteigend sortiert).

***ausgeben()*:** dient zur Kontrolle und gibt das Array in übersichtlicher Form auf die Konsole aus.

shuffle(anzahl : int): Mischt die (sortierten) Zahlen. Wie stark die Zahlen vermischt werden bestimmt *anzahl*. Mischen bedeutet *anzahl* Vertauschungen von jeweils zwei zufällig gewählten Elementen des Arrays.

Klasse *Test*:

Test
+ testen()

testen()*:* soll eine Instanz der Klasse *Sortieren* erzeugen und für unterschiedlich große Arrays die Anzahl der Vergleichsoperationen beim Sortieren und die benötigte Zeit ausgeben - z.B.: für ein Array mit 100, 1000 und 10000 zufällig belegten Feldern.
Wie stark steigt der Aufwand? Ist BubbleSort ein effizienter Sortieralgorithmus?

Nach dem Sortieren werden die Zahlen mehr oder weniger stark gemischt (*shuffle()*).
Wie ändert sich der Aufwand? Ist er bei fast vollständig sortierten Zahlen erkennbar geringer?

Hinweise:

Ermitteln der benötigten Rechenzeit: Dazu holt man sich unmittelbar vor dem Aufruf einer Sortiermethode die aktuelle Zeit mittels *System.currentTimeMillis()* und ebenso unmittelbar danach. Die Differenz der beiden *long*-Werte ergibt die benötigte Rechenzeit in Millisekunden.

Debugging: Versuche auch einen „Breakpoint“ an geeigneter Stelle im Sortieralgorithmus zu setzen und sieh Dir den Ablauf mit Hilfe des Debuggers und des Inspektorfensters an.

Optional:

Wer Zeit hat, versucht einen weiteren Sortieralgorithmus zu realisieren
z.B.: Insertionsort: http://de.wikipedia.org/wiki/Insertionsort oder
Shakersort: http://de.wikipedia.org/wiki/Shakersort
und vergleicht den Aufwand mit Bubblesort durch entsprechnde Aufrufe in der Testklasse.