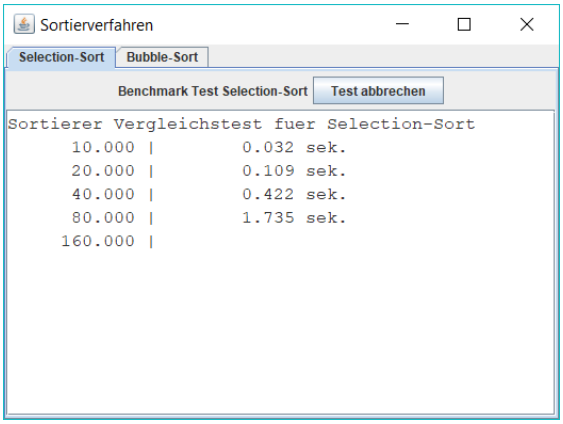
Bitte geben Sie ***Aufgabe11.java***, ***SortiererTestPanel.java*** sowie ***QuickSort.java*** bis zum 08.11.2020 ab!

Aufgabe 10 Swing-App zum Vergleich von Sortierverfahren

Wir implementieren eine Swing-App, mit der wir das Laufzeitverhalten unserer Sortieralgorithmen studieren können. Nebenbei sehen Sie, wie in einer Anwendung mehrere parallele Threads laufen können.

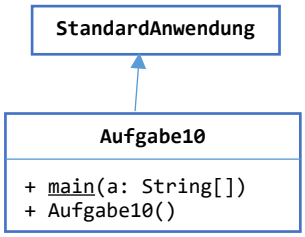
Die Swing-Anwendung *Aufgabe10* hat folgende Merkmale:

- Das Hauptpanel besteht aus mehreren Registerkarten (Klasse *JTabbedPane*).
- Die Anwendung erlaubt es, einen Benchmark-Test stoppen und neu zu starten während andere Sortierungen im Hintergrund laufen, d.h man kann in verschiedenen Registern mehrere Tests parallel laufen lassen. Im Task-Manager kann man feststellen, dass tatsächlich mehrere Kerne der CPU voll belastet werden. Dies wird dadurch erreicht, dass jede Sortierung in einem eigenen ***Thread*** läuft.



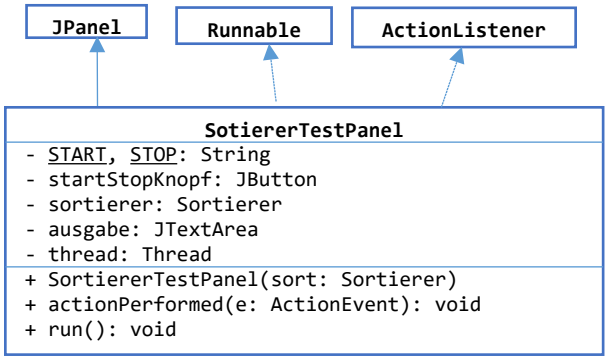
Das Layout der Anwendung

Schreiben Sie die von *StandardAnwendung* abgeleitete Klasse ***Aufgabe10***. Die Methode *main* ruft wie immer *starteAnwendung*.



Im **Konstruktor** von *Aufgabe10* geben Sie der Anwendung ein *BorderLayout* und legen darauf ein Objekt vom Typ *JTabbedPane,* für das Sie zwei Mal die Methode *add* rufen und ihr jeweils einen Text sowie zunächst ein neues (leeres) *JPanel* Objekt übergeben.

Dann schreiben Sie anhand des UML Diagramms die von *JPanel* abgeleitete öffentliche Klasse ***SortiererTestPanel***, welche *ActionListener* sowie *Runnable* implementert.



START und *STOP* bekommen die Werte „Test starten“ sowie „Test abbrechen“, *startStopKnopf* bekommt START als Aufschrift, *ausgabe* wird mit dem Standardkonstruktor initialisiert. Die Werte von *thread* und *sortierer* bleiben *null*.

Der **Konstruktor** speichert seinen Parameter in *sortierer* und registriert sich als *ActionListener* von *startStopKnopf*. Sein eigenes Layout setzt er zu einem *BorderLayout*.

Dann erzeugt er das neue *JPanel*-Objekte *oben,* auf das er *startStopKnopf* legt, danach wird *oben* auf *this* an *BorderLayout.NORTH* und ein neues *JScrollPane* Objekt mit *ausgabe* als View auf *CENTER* gelegt.

Zum Schluss schreiben Sie in *ausgabe* den Text „Der Test wurde noch nicht gestartet“ und machen *ausgabe* nicht-editierbar (Methode *setEditable*).

Die Methoden *actionPerformed* und *run* bleiben zunächst leer.

Die Anwendung *Aufgabe10* hat jetzt ihr endgültiges Layout, aber noch keine Funktionalität.

Sortier-Test unterbrechbar machen

Jetzt erweitern wir ***SelectionSort.sort*** um folgende Zeile unmittelbar vor dem Aufruf von *tausche*:

```
Thread.sleep(0);
```

Bei jedem Durchgang der Schleife gibt *sleep* die Kontrolle kurz an das Betriebssystem ab und dabei wird geprüft, ob der Thread ein Interrupt-Signal bekommen hat. Deshalb ist das *throws InterruptedException* hinter der Parameterliste der Methode *sort* notwendig. Schauen Sie sich die Methode *ArrayTools.sortiereBenchmark* an – sie fängt in der *catch*-Anweisung diese Exception ab (*throw-catch* lernen wir später noch kennen).

Die gleiche Erweiterung nehmen Sie in ***BubbleSort*** direkt vor dem Aufruf von *paarweiseVertauschen* vor.

Funktionalität

Wir erweitern die Klasse *SotiererTestPanel* wie folgt:

- Die Methode ***run*** ruft *ArrayTools.sortiererBenchmark* mit den Parametern *sortierer* und *ausgabe*.
- Die Methode ***actionPerformed*** prüft, ob *thread null* ist.
  - Wenn ja, speichert sie in der Variable *thread* ein neues *Thread* Objekt, dessen Konstruktor sie *this* übergibt, startet den Thread (Methode start) und wechselt die Aufschrift von *startStopKnopf* auf *STOP* (Methode setText).
  - Falls *thread* nicht den Wert *null* hat, ruft sie die Methode *thread.interrupt()*, setzt *thread* auf *null* und wechselt die Aufschrift von *startStopKnopf* auf *START*.

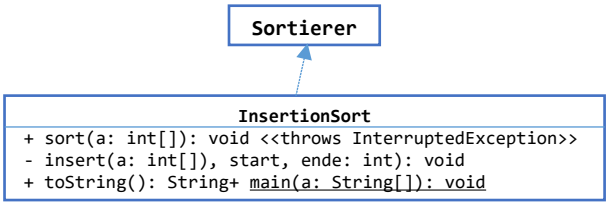
Aufgabe 11

Mehr Sortieralgotithmen

In dieser Aufgabe erweitern wir unsere Anwendung *Aufgabe10* um weitere Sortieralgorithmen.

Insertion-Sort

Schreiben Sie die Klasse ***InsertionSort***, welche den Algorithmus aus der Vorlesung implementiert. Erstellen Sie eine Kopie der Klasse *Aufgabe10* als ***Aufgabe11*** und fügen ein weiteres Panel mit dem Insertion-Sort Algorithmus ein.

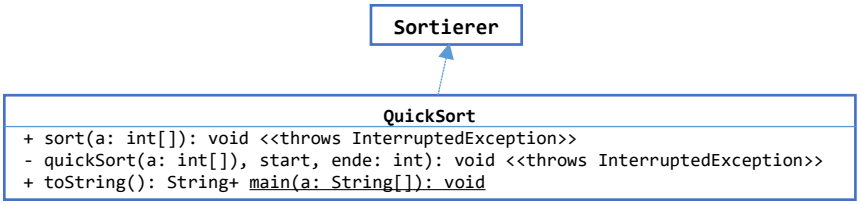


Shell-Sort und Merge-Sort

Im Online Kurs finden Sie die Klassen ***ShellSort*** und ***MergeSort***, fügen Sie zwei weitere Panel für diese Sortierverfahren hinzu. Schauen Sie sich die Klasse an und vollziehen Sie den Algorithmen soweit wie möglich nach. Sie können auch drei Panel für den Shell-Sort erzeugen, welche die verschiedenen Distanz-Sequenzen verwenden.

Quick-Sort

Implementieren Sie die Klasse ***QuickSort***, welche *Sortierer* implementiert entsprechend dem nebenstehenden Diagramm.

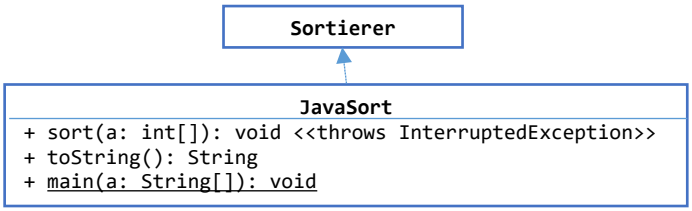


Die *sort*-Methode ruft die Methode *quickSort* für den ganzen Array.

Die Methode ***quickSort*** implementiert den rekursiven Algorithmus aus der Vorlesung. Rufen Sie nach der äußeren *while*-Schleife *Thread.sleep(0)*, damit die Ausführung unterbrochen werden kann.

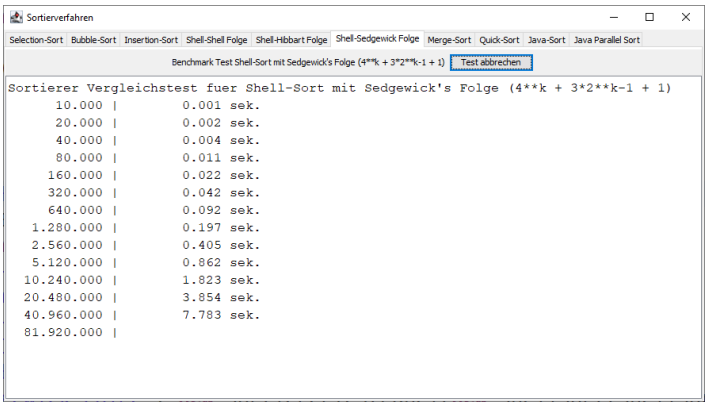
Vergleichen Sie den Aufwand der verschiedenen Sortieralgorithmen indem Sie die Laufzeiten für verschiedene Array-Längen in eine Tabelle eintragen. Schätzen Sie ab, wie lang das Sortieren von 160 Millionen Zahlen jeweils brauchen würde.

Java-Sort und Java-Parallel-Sort



Schreiben Sie zum Vergleich die Klasse ***JavaSort***, welche *Sortierer* implementiert. In der Methode ***sort*** rufen Sie die in Java standardmäßig vorhandene Methode *Arrays.sort* auf. Vergleichen Sie die Performance des Java-Sortierers mit ihrem eigenen Quick-Sort.
Der Unterschied resultiert hauptsächlich aus einer besseren Pivot-Suche. Fügen Sie der Klasse *Aufgabe11* ein Panel mit dem Java-Sort hinzu.

Schreiben Sie zusätzlich die Klasse ***JavaParallelSort***, welche die Methode *Arrays.parallelSort* verwendet, die einen parallelisierten Merge-Sort implementiert.



Je nach der Anzahl Threads, welche Ihr Prozessor gleichzeitig ausführen kann, sehen Sie nochmals eine Verbesserung gegenüber dem normalen Java Sort.

Fügen Sie der Klasse *Aufgabe11* ein weiteres Panel mit dem parallelen Java Sort hinzu. Schauen Sie sich die Beschreibung der Klasse Arrays genauer an, dort gibt es viele nützliche Methoden!

Mit dieser Anwendung können Sie Ihren Rechner gut auslasten!

Typische Fragen für die mündliche Prüfung

Schreiben Sie folgende Methode(n):

```
+istVorhanden(a:int[], key:int):boolean

+wieOftVorhanden(a:int[], key:int):int

+ungeradeZahlen(a:int[]):int[]
```
liefert Array mit allen ungeraden Zahlen aus a (Arrays dürfen die Länge 0 haben).

```
+zahlenOberhalb(a:int[], w:int):int[]
```
liefert Array mit allen Zahlen aus a, die größer als w sind (Arrays dürfen die Länge 0 haben).

Geben Sie die Komplexität der folgenden Programmfragmente in Landau Notation an:

```
int summe = 0;
int[] a = new int[5000];   // Hinwis; Array wird mit 0 gefüllt
================================================================================
for(int i=1;i<N;i++){
      Summe = summe + 10*i;
}
================================================================================
for(int k:a){
    for(inti=0;i<k;i++) …
}
================================================================================
for(int i=0;i<a.length;i++){
    for(int j=I;j<a.length;j++) …
}
================================================================================
for(int i=0;i<a.length;i++){
    for(int j=a.length;j>0;j/=2) …
}
```