Bitte lösen Sie Aufgaben 7-9 bis zum 05.11.2018 **und geben Aufgabe 9 bis zum 1.11.2020 ab!**

Aufgabe 7

Schleifen, BorderLayout, JTextArea, JScrollPane

In dieser Aufgabe üben Sie verschiedene Schleifen, indem Sie folgende Zahlenfolgen durch eine ***for***- oder eine ***while***-Schleife realisieren:

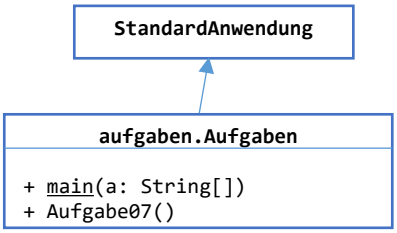
1 Die Potenzen von 2 bis 2^{10}. Verwenden Sie die Funktion *Math.pow* für die Potenzen
2 Die ungeraden Zahlen 1,3,5,7,…,51.
3 Die Quadratzahlen 1,2,4,9,16,…,1024.
4 Die Zahlen 1,2,4,7,11,16,22,29,37,…,191.

Die Anwendung besteht aus einem oberen Panel mit den *JButton* Elementen und einem unteren mit der Ausgabe als *JTextArea* innerhalb eines *JScrollPane* Objekts, ähnlich wie das Beispiel auf Seite 5 im ersten Übungsblatt.

Die Anwendung besteht aus zwei Klassen

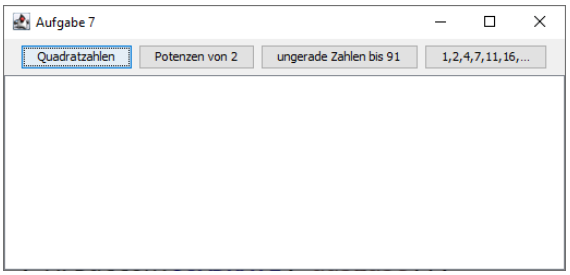
- Die von *StandardAnwendung* abgeleitete Klasse ***Aufgabe07*** enthält die *main*-Methode, welche lediglich *starteAnwendung* ruft, zusätzlich hat Sie einen Standardkonstruktor.
- Die von *JButton* abgeleitete Klasse ***A7Button*** implementiert *ActionListener*, sie besteht aus einem Konstruktor und der Methode *actionPerformed*.

Das Layout



Beginnen Sie mit der Klasse ***Aufgabe07***. Der Konstruktor ruft den Basisklassenkonstruktor, setzt sein Layout zu *BorderLayout* und erzeugt ein *JTextArea-* sowie ein *JPanel* Objekt, die er in den lokalen Variablen ***ausgabe*** und ***oben*** speichert.

Auf *oben* legen Sie vier *JButton* Objekte mit den Aufschriften „Quadratzahlen“, „Potenzen von 2“, „ungerade Zahlen bis 91“ sowie „1,2,4,7,11,16,…“. Später werden wir den Typ dieser *JButton* Objekte in *A7Button* ändern.

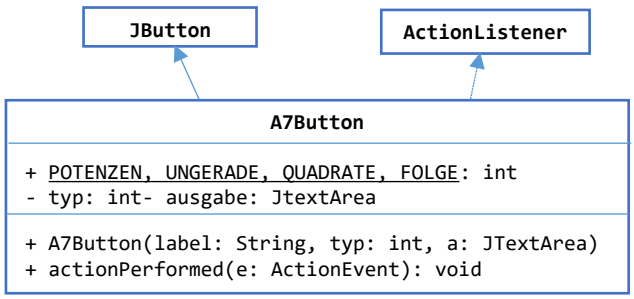


Dann legen Sie *oben* auf *this* an die Position *BorderLayout.NORTH*.

Auf die Position *BorderLayout.CENTER* legen Sie ein *JScrollPane* Objekt mit *ausgabe* als View.

Jetzt sollte die Anwendung schon ihr endgültiges Aussehen haben, wie rechts gezeigt, aber noch nicht funktionieren.

Die Funktionalität



Schreiben Sie von *JButton* abgeleitete Klasse ***A7Button***, welche *ActionListener* implementiert. Die Klasse kann in der Datei Aufgabe07.java hinter der Klasse Aufgabe07 stehen, darf dann aber nicht öffentlich sein.

Definieren Sie die öffentlichen *int*-Konstanten (*final*) *POTENZEN*, *UNGERADE*, QUADRATE und FOLGE mit beliebigen, aber **unterschiedlichen** Werten.

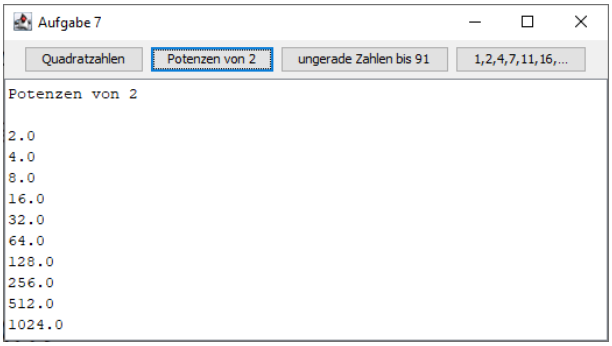
Der Konstruktor führt folgende Schritte durch:

1 Der erste Parameter (*label*) wird an den Basisklassenkonstruktor weitergegeben.
2 Der beiden anderen Parameter werden in den entsprechenden Instanzvariablen gespeichert.
3 Das *A7Button* Objekt registriert sich als sein eigener *ActionListener*.

In ***actionPerformed*** holen Sie sich den Titel des Buttons (Methode *getText*) und schreiben ihn in *ausgabe* (*setText*), wobei am Ende noch ein Zeilenumbruch (*\n*) angehängt wird.

Wenn Sie jetzt im Konstruktor von *Aufgabe07* die *JButton* Objekte durch *Aufgabe7Button* ersetzen, müsste beim Klick schon der entsprechende Text im Ausgabefeld erscheinen.

Dann prüfen Sie, ob *typ* den Wert *POTENZEN*, hat und wenn ja, programmieren Sie eine for-Schleife, welche die Werte berechnet und mit der Methode *append* auf die Ausgabe schreibt.



Falls *typ* nicht gleich *POTENZEN* ist, prüfen Sie mit *else if*, ob *typ* den Wert QUADRATE hat, dort nehmen Sie eine *while*-Schleife und berechnen die Quadratzahlen.

Für die beiden letzten Fälle nehmen Sie wieder jeweils eine *for*- und einmal eine *while* Schleife.

Aufgabe 8

Arrays

Übernehmen Sie in das Paket *tools* das Interface ***Sortierer*** sowie die Klasse ***ArrayTools*** mit Hilfsfunktionen für Arrays aus dem persönlichen Stundenplan in Ihr Projekt. Versuchen Sie, die Methoden in dieser Klasse genau zu verstehen.

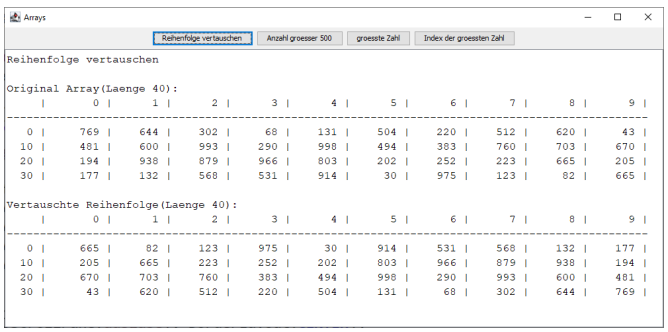
Erweitern Sie *ArrayTools* um folgende Methoden und testen Sie diese mit einer geeigneten Anwendung, die ähnlich aufgebaut ist wie Aufgabe 7:

tools.ArrayTools
+ zahlenAusgeben(ausgabe: JTextArea):void + fillRandom(n:int): int[]+ tausche(a: int[], i,j: int):void + istSortiert(a: int[]): boolean + sortiererBenchmark(s: Sortierer, a: JTextArea): void

- **`public static int anzahlElementeOberhalb(int[] x, int min)`**
  zählt die Elemente in *x*, die größer sind als *min*. Die Methode gibt das Ergebnis als *return*-Wert an das rufende Programm zurück. Verwenden Sie bei dieser Methode die verkürzte *for*-Schleife: `for(int k:x)…`
- **`public static int getMax(int[] x)`**
  findet die größte Zahl im Array, auch hier bietet sich die verkürzte for-Schleife an.
- **`public static int getIndexMax(int[] x)`**
  gibt den Index der größten Zahl zurück. In dieser Methode **müssen** Sie wieder eine for-Schleife mit einem Schleifenzähler verwenden.
- **`public static void tausche anzahlElementeOberhalb(int[] x, int i, int j)`**
  vertauscht die Elemente an den Indizes i und j.
  Prüfen Sie, ob `x` null ist oder i und j nicht zwischen `0` und `x.length` liegen, in diesen Fällen macht die Methode nichts.
- **`public static void vertauscheReihenfolge(int[] x)`**
  dreht die Reihenfolge der Elemente in einem *int*-Array um. Verwenden Sie die Methode *ArrayTools.tausche* um zwei Elemente zu vertauschen.
  Hinweis: programmieren Sie eine Schleife, die bis zur Hälfte der Länge des Arrays läuft und das erste Element mit dem letzten, das zweite mit dem vorletzten Element vertauscht u.s.w.

Schreiben Sie die von *StandardAnwendung* abgeleitete Klasse ***Aufgabe08*** sowie die von *JButton* abgeleitete Klasse ***A8Button*** – genau wie in Aufgabe 7.

Beim Klick auf einen der Buttons wird zunächst ein Array mit zufällig verteilten Zahlen erzeugt (Methode *ArrayTools.fillRandom*) und durch Aufruf von *ArrayTools.zahlenAusgeben* in der Textbox ausgegeben. Dann wird die entsprechende Methode gerufen (z.B. *vertauscheReihenfolge)* und danach das Ergebnis ausgegeben.



Aufgabe 9

Sortieralgorithmen

In dieser Aufgabe programmieren Sie den Selection- und Bubble-Sort Algorithmus.

Im Paket *tools* haben wir in Aufgabe 8 das Interface *Sortierer* übernommen, es fordert folgende Methode:

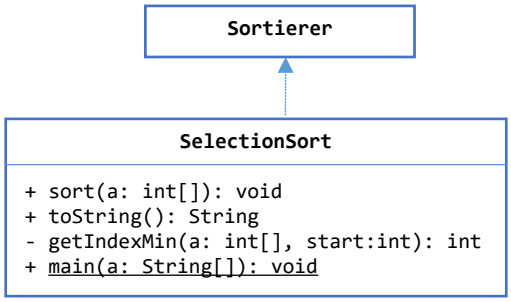
```
public void sort(int[] a) throws InterruptedException;
```

Ein Objekt vom Typ *Sortierer* muss also eine Methode *sort* besitzen, die einen *int*-Array als Parameter hat. (Die *InterruptedException* ermöglicht es, einen Sortiervorgang zu stoppen - Exceptions behandeln wir allerdings erst später, im Moment brauchen Sie die Details nicht zu verstehen).

Schauen Sie sich die Methode ***sortiererBenchmark*** in der Klasse *ArrayTools* genau an. Man sieht an diesem Beispiel den Vorteil eines Interface: wir konnten in *sortiererBenchmark* die Methode *Sortierer.sort* schon aufrufen ohne dass es eine Klasse gibt, die *Sortierer* implementiert.

Erstellen Sie das neue Paket ***sortieren*** und darin die Klasse ***SelectionSort,*** die *Sortierer* implementiert.
In der Methode ***sort*** programmieren Sie den in der Vorlesung behandelten Selection-Sort Algorithmus:

```
public void sort(int[] a){
  if(a!=null){
    for(int i=0;i<a.length-1;i++){
       …
```



Schreiben Sie in der Klasse *SelectionSort* die private Methode getIndex*Min, welche* den Index des kleinsten Elements im Array ***a*** **ab der Stelle *start*** zurückgibt.

Zusätzlich schreiben Sie die öffentliche Instanzmethode ***toString***, welche einfach nur den Text „Selection-Sort“ zurückgibt. Wenn ein Objekt eine solche Methode besitzt, kann man es wie einen String verwenden – das Objekt identifiziert sich dadurch mit diesem Text.
Die Methode *ArrayTools.sortiereBenchmark* nutzt diese Eigenschaft:

```
ausgeben(ausgabe, "Sortierer Vergleichstest für " + sortierer);
```

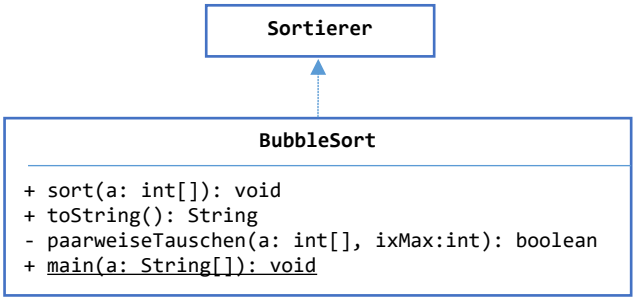
Fügen Sie *SelectionSort* eine ***main***-Methode hinzu, die *ArrayTools.sortiereBenchmark* ruft und als Parameter ein neues *SelectionSort* Objekt sowie *null* als zweiten Parameter übergibt.

Starten Sie die Anwendung *SelectionSort* und notieren Sie die Zeiten für die Sortiervorgänge. Die Komplexität $O(N^2)$ wird schnell offensichtlich, wie die Ausgabe im Kasten rechts zeigt.

```
Sortierer Vergleichstest fuer Selection-Sort
      10.000 |        0.040 sek.
      20.000 |        0.110 sek.
      40.000 |        0.409 sek.
      80.000 |        1.552 sek.
     160.000 |        6.036 sek.
     320.000 |       24.872 sek.
```

Schreiben Sie als nächstes die Anwendung ***BubbleSort***, welche den Bubble-Sort Algorithmus aus der Vorlesung implementiert. Auch hier zeigt sich das Verhalten von $O(N^2)$, er ist insgesamt etwas langsamer als der Selection-Sort, was hauptsächlich an der kurzen Distanz liegt, über welche Elemente vertauscht werden.

Sie verfügen jetzt über zwei Anwendungen, die Ihren Rechner gut auslasten!



Schätzen Sie ab, wie lange diese Algorithmen auf Ihrem Rechner brauchen, um ein Array der Länge 160.000.000 zu sortieren.

Das Wichtigste aus der aktuellen Semesterwoche

- **Arrays sind Objekte**, ihre Länge kann nicht geändert werden.
- Die ***for*-Schleife** ist meist die bequemste Art um alle Elemente eines Arrays zu besuchen: for(int i=0;i<a.length;i++)…
- Will man die Array-Elemente nicht ändern, kann man auch die **vereinfachte for-Schleife** verwenden: for(int k:a) …
- Die **while-**Schleife ist wie die for-Schleife eine vorgetestete Schleife.
- Die **do…while** Schleife ist eine nachgetestete Schleife.

- Die Komplexität eines Algorithmus, der sich aus zwei Teilen zusammensetzt, die seriell hintereinander ausgeführt werden, ist gleich der höchsten Komplexität der beiden Algorithmen.
- Ein **Sortierverfahren** heißt **stabil**, wenn Objekte mit gleichem Schlüssel ihre ursprüngliche Reihenfolge behalten.
- **Selection-Sort** ist ein instabiler in-situ Algorithmus mit der Komplexität $O(N^2)$
- **Bubble-Sort** ist ein stabiler in-situ Algorithmus mit der Komplexität $O(N^2)$