Bubble Sort

I. den Algorithmus simulieren

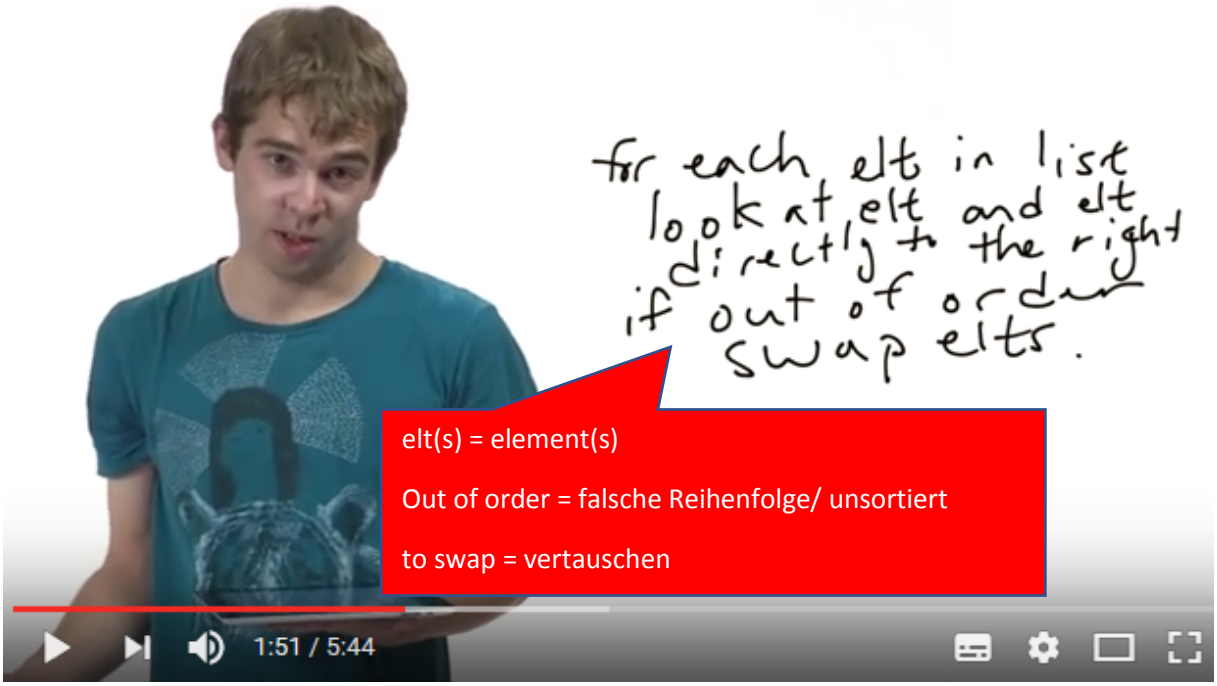
[Einzelarbeit]
Schau dir das youtube-Erklärvideo bis **1:44** zu dem Sortieralgorithmus *Bubble Sort* an.

[Partner-/ Gruppenarbeit]

- Sortiere das vor dir liegende Array genauso, wie es der Sprecher vormacht.
- Wie sieht das Array nach einem Durchlauf aus? Tragt hier die Zahlenfolge ein:

0	1	2	3	4

II. Teile des Pseudocodes entwickeln



[Einzelarbeit]
Notiere auf der Grundlage der Erklärungen von 1:44 – 1:51 den Pseudocode, indem du ….

- überlegst, welche Datenstrukturen (Array, Variable(n)) benötigt werden.
- die Sprachelemente SCHRITTWEISE…. WENN …. DANN…. verwendest.

[Partner-/Gruppenarbeit]
Vergleiche deine Ergebnisse zuerst mit denen deines Partners/ deiner Partnerin und dann mit der **→ Musterlösung** (bei Frau Anthes).

III. Anzahl der Durchläufe bestimmen

Worst-Case-Szenario

Das Worst-Case-Szenario bei einer unsortierten Zahlenfolge ist der, dass die Zahlen in absteigender Reihenfolge sortiert sind. Zum Beispiel: 9-8-7-6-5-4-3-2-1.

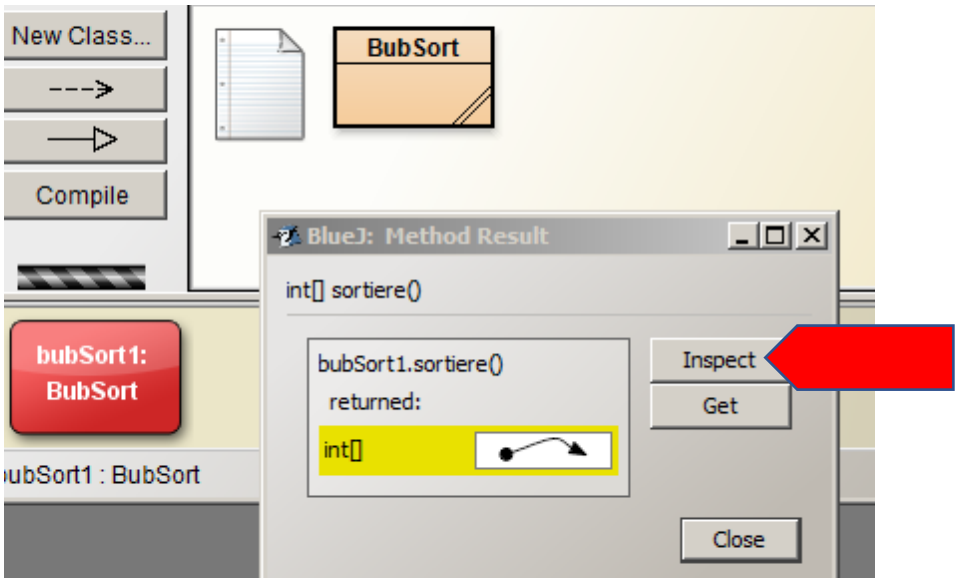
Das Best-Case-Szenario ist entsprechend, dass alle Elemente in aufsteigender Reihenfolge vorliegen – also bereits sortiert sind. Zum Beispiel: 1-2-3-4-5-6-7-8-9.

Wir nehmen das Worst-Case-Szenario an, um den **maximalen Aufwand** eines Sortieralgorithmus zu bestimmen.

Einzelarbeit:

Finde heraus, wie viele Durchläufe Bubble Sort benötigt, um ein maximal unsortiertes Array zu sortieren, indem du

- das Java-Programm bubbleSort.zip herunterlädst, entpackst und öffnest.
- ein Objekt der Klasse BubSort erstellst, ein Array mit zwei Elementen erstellst und die Methode `int[] sortiere()` sooft aufrufst, bis das Array sortiert ist. Schaue dir nach jedem Methodendurchlauf das Array mittels Inspect an (siehe unten):



Erstelle nun nacheinander zwei weitere BubSort-Objekte, deren Arrays jeweils unterschiedlich lang sind und trage die Werte in folgende Tabelle ein:

Länge des Arrays	Benötigte Durchläufe
2	
3	
4	
5	