EP1 Programmiertest WS2021 27. Jänner 2022	**Matrikelnummer**	**Familienname**	**Unterschrift**

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder globalen Variablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **nicht** die Methoden `clone` und `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`, `isEmpty`
- Bis auf die Klasse `Math` dürfen **keine** weiteren Klassen der Java API verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = {{0}, {0, 1}, {0, 1, 2}, {0, 1, 2, 3}};
int[][] test2 = {{1, 2, 3, 4}, {0}, {1, 2}, {3, 4}, {5, 6}};
int[] seq = {0, 4, 3, 4, 4, 4, 8, 7};
```

Implementieren Sie folgende Methoden:

- `int[][] replicate(int[][] input, int[] iter)` liefert ein neues Array mit der selben Zeilenanzahl wie `input` zurück. Jede Zeile des neuen Arrays besteht aus den Einträgen der ursprünglichen Zeile in `input` und aus Wiederholungen dieser Zeile, wobei die Anzahl der Wiederholungen durch den jeweiligen Eintrag in `iter` bestimmt wird. Ein Eintrag von 0 in `iter` bedeutet, dass die Zeile nicht wiederholt wird und somit nur einmal elementweise in das neue Array kopiert wird. Bei einem negativen Eintrag in `iter` wird die Zeile rückwärts kopiert und auch rückwärts wiederholt, und zwar so oft, wie es dem Absolutbetrag des negativen Eintrags entspricht.
  Vorbedingung(en): `input != null`, `input.length > 0`, `input[i] != null` und `input[i].length > 0` für alle gültigen `i`, `iter != null`, `iter.length == input.length`.
  Wird die Methode z.B. mit den Parametern `test1` und `iter = new int[]{2, 1, -3, 0}` aufgerufen, entsteht folgendes Array:

0	0	0									
0	1	0	1								
2	1	0	2	1	0	2	1	0	2	1	0
0	1	2	3								

- `void move(int[][] input)` verändert das Array `input` auf folgende Weise: Das letzte Element der ersten Zeile (mit Index 0) wird entfernt und an das Ende der zweiten Zeile angehängt. Die selbe Operation wird dann mit der dritten und vierten Zeile gemacht, und so weiter, bis das Ende von `input` erreicht wird. Bei einer ungeraden Zeilenanzahl wird die letzte Zeile nicht verändert.
  Vorbedingung(en): `input != null`, `input.length > 0`, `input[i] != null` und `input[i].length > 0` für alle gültigen `i`.
  Wird die Methode z.B. mit `test2` aufgerufen, entsteht folgendes Array:

1	2	3
0	4	
1		
3	4	2
5	6	

- `boolean contains(int[] seq, int index, int n)` überprüft, ob in `seq` genau `n` Paare von direkt aufeinander folgenden Einträgen vorkommen, die richtig geordnet sind. Ein Paar gilt jeweils als richtig geordnet, wenn der erste Eintrag kleiner gleich dem zweiten Eintrag ist. Die Paare im Array dürfen sich auch überlappen (z.B. enthält das Array `{1, 2, 3}` zwei geordnete Paare, weil `1 <= 2` und `2 <= 3` ist). Die Zählung beginnt bei `index` (inklusive).
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `seq != null`, `seq.length > 0`, `index` beschreibt einen gültigen Index von `seq`, `n >= 0`.

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = replicate(test1, new int[]{2, 1, -3, 0})`	[[0, 0, 0], [0, 1, 0, 1], [2, 1, 0, 2, 1, 0, 2, 1, 0, 2, 1, 0], [0, 1, 2, 3]]
`result2 = replicate(test2, new int[]{1, 0, -1, 2, 0})`	[[1, 2, 3, 4, 1, 2, 3, 4], [0], [2, 1, 2, 1], [3, 4, 3, 4, 3, 4], [5, 6]]
`move(test1)`	[[], [0, 1, 0], [0, 1], [0, 1, 2, 3, 2]]
`move(test2)`	[[1, 2, 3], [0, 4], [1], [3, 4, 2], [5, 6]]]
`contains(seq, 0, 5)`	true
`contains(seq, 1, 4)`	true
`contains(seq, 3, 2)`	false
`contains(seq, 3, 4)`	false
`contains(seq, 6, 0)`	true

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	/ 1
	Testfälle korrekt implementiert	/ 2
`replicate`	Korrekte Länge der Zeilen	/ 2
	Korrekte Wiederholung der Zeilen	/ 3
	Korrekte umgedrehte Wiederholung der Zeilen	/ 3
	Korrektes Kopieren der Zeilen bei `iter[i] == 0`	/ 1
`move`	Korrekte Dimensionen	/ 2
	Korrektes Löschen des letzten Elements	/ 3
	Korrektes Einfügen in nächster Zeile	/ 3
	Korrekte Behandlung bei ungerader Zeilenanzahl	/ 1
`contains`	Korrekter Methodenansatz (Rückgabe vorhanden)	/ 1
	Basisfall/Basisfälle vorhanden	/ 1
	Basisfall/Basisfälle korrekt	/ 1
	Fortschritt der Rekursion vorhanden	/ 1
	Fortschritt der Rekursion korrekt	/ 1
	Korrekter Rückgabewert	/ 4
Gesamt		/ 30