Westfälische Hochschule

Fachbereich Informatik und Kommunikation
Prof. Dr. Marcel Luis

OPR: Praktische Aufgabe „Permutation“

In dieser Aufgabe geht es darum, alle möglichen Anordnungen (Permutationen) von Zahlen zu generieren. Die Zahlen 0 bis 3 lassen sich z. B. als $0, 1, 2, 3$ oder $1, 3, 0, 2$ anordnen. (Es gibt noch 22 weitere Anordnungen dieser Zahlen. Allgemein gibt es für n verschiedene Zahlen $n!$ Anordnungen.)

Realisieren Sie im Paket `permutation` ein Klasse `Permutation` mit folgenden Methoden:

- Einen Konstruktor `Permutation(int)`, durch den für einen Parameter n ein Objekt erzeugt wird, das eine Anordnung der Zahlen $0, 1, \ldots n-1$ repräsentiert. Die Zahlen sollen in dem erzeugten Objekt aufsteigend angeordnet sein. Sie dürfen davon ausgehen, dass der Parameter größer als 0 ist.
- Eine Instanzmethode `String asText()`, die eine textuelle Darstellung einer Permutation erzeugt. In dieser Darstellung sollen die Zahlen entsprechend ihrer Anordnung hintereinander stehen, getrennt durch einen Strich.

  Beispiel: `(new Permutation(6)).asText()` soll `0-1-2-3-4-5` liefern.
- Eine Instanzmethode `boolean generateNext()`, die zu der aktuellen Anordnung der Zahlen die lexikografisch nächste Anordnung berechnet, sofern es noch eine gibt. Falls die aktuelle Anordnung bereits die lexikografisch letzte ist, soll die Methode `false` liefern, sonst `true`.

Die Methode `boolean generateNext()` soll zur aktuellen Anordnung die lexikografisch nächste berechnen. Wer den Begriff „lexikografische Ordnung“ nicht kennt, kann sich einfach vorstellen, die Zahlen wären Buchstaben eines Alphabets (0 der erste Buchstabe, 1 der zweite, 2 der dritte usw.). Dann ist eine Anordnung der Zahlen wie ein Wort aus diesen Buchstaben - und die lexikografische Reihenfolge der Wörter entspräche der Reihenfolge der Wörter in einem Lexikon.

Die folgende Tabelle zeigt die lexikografische Reihenfolge der Permutationen der Zahlen 0 bis 9. Die Permutationen sind der besseren Lesbarkeit wegen mit Strichen zwischen den Zahlen dargestellt, in Anlehnung an das Format, das `asText` erzeugt. (Auf die mit `(X)` markierten Permutationen wird später noch Bezug genommen.)

```
0-1-2-3-4-5-6-7-8-9
0-1-2-3-4-5-6-7-9-8
0-1-2-3-4-5-6-8-7-9
0-1-2-3-4-5-6-8-9-7
...
2-9-0-3-6-5-8-7-4-1  (X)
2-9-0-3-6-7-1-4-5-8  (X)
...
7-8-3-9-6-5-4-2-1-0
```

```
7-8-4-0-1-2-3-5-6-9
...
9-8-7-6-5-4-3-2-1-0
```

Realisieren Sie im Paket `permutation` außerdem eine Klasse `PermutationTest` zum Test der Methoden der Klasse `Permutation`. Realisieren Sie darin eine Methode `main` mit folgendem Testablauf.

Nr.	Operation	Sollergebnis
1	Objekt der Klasse `Permutation` erzeugen.	
2	Dieses Objekt textuell darstellen (`asText` aufrufen und Rückgabewert ausgeben).	`0-1-2-3-4-5-6-7-8-9`
3	nächste Permutation berechnen (`generateNext` aufrufen und Rückgabewert ausgeben)	`true`
4	Objekt textuell darstellen	`0-1-2-3-4-5-6-7-9-8`
5	362878 mal nächste Permutation berechnen	
6	Objekt textuell darstellen	`0-9-8-7-6-5-4-3-2-1`
7	nächste Permutation berechnen	`true`
8	Objekt textuell darstellen	`1-0-2-3-4-5-6-7-8-9`
9	3265918 mal nächste Permutation berechnen	
10	Objekt textuell darstellen	`9-8-7-6-5-4-3-2-0-1`
11	nächste Permutation berechnen	`true`
12	Objekt textuell darstellen	`9-8-7-6-5-4-3-2-1-0`
13	nächste Permutation berechnen	`false`
14	Objekt textuell darstellen	`9-8-7-6-5-4-3-2-1-0`

Hinweise

- Verwenden Sie nur den Vorlesungsstoff bis einschließlich zum Kapitel „Pakete".
- Diese Aufgabe erfordert *sorgfältiges Überlegen*, bevor Sie sich an den Rechner setzen. Wer gleich beginnt zu programmieren, kann hier nur scheitern!

  Der wesentliche Punkt ist die Berechnung der nächsten Permutation. Hierfür müssen Sie einen Algorithmus finden. Analysieren Sie dafür vor allem die mit `(X)` markierten Permutationen aus der obigen Tabelle. Bedenken Sie dabei folgende Punkte:

  - Von einer Permutation zur unmittelbar nächsten Permutation müssen soviele Zahlen wie möglich *am Anfang* unverändert bleiben, sonst würde nicht die *nächste* Permutation generiert (von 2-9-0-3-6-5-8-7-4-1 zu 2-9-0-3-6-7-1-4-5-8 bleibt der Anfang 2-9-0-3-6 unverändert). Anders ausgedrückt: Die Vertauschung der Zahlen, um von einer Permutation zur nächsten zu gelangen, muss in einem möglichst kleinen „Stück" *am Ende* erfolgen, im Beispiel innerhalb des Endstücks 5-8-7-4-1.
  - Überlegen Sie sich, warum man nicht durch Vertauschung innerhalb des noch kürzeren Endstücks 8-7-4-1 zur nächsten Permutation kommen kann. Wenn Ihnen dies klar ist, sind Sie auf der richtigen Spur.

- Erstellen Sie für die Prüfung durch ARCTERN im Abgabebereich der Aufgabe ein Verzeichnis für das Java-Paket und laden dorthin Ihre Klassen hoch.