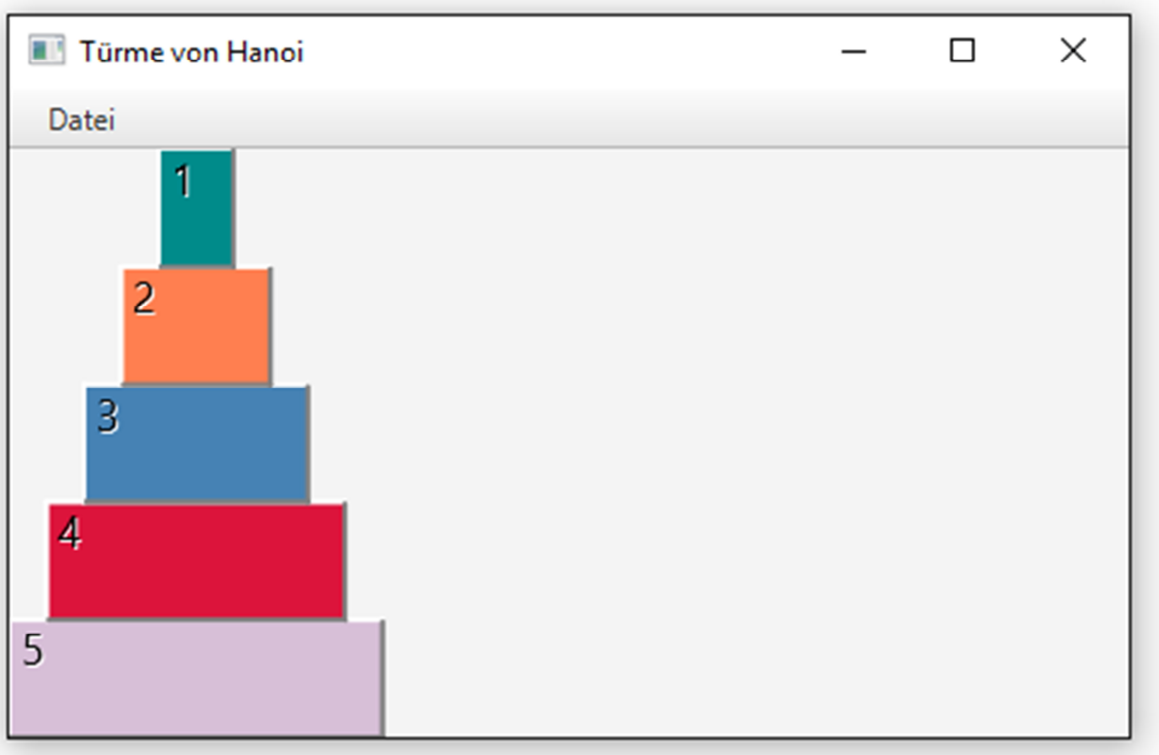
# Übungsblatt 17 JavaFx

**1. Türme von Hanoi**

Bei dieser Aufgabe haben wir 3 Stäbe und n gelochte Scheiben, die auf die Stäbe passen. Die Scheiben haben unterschiedliche Größen und liegen am Anfang der Größe nach geordnet auf einem der Stäbe, wobei die größte Scheibe (Scheibe n) unten liegt und die kleinste (Scheibe 1) oben. Die Aufgabe besteht darin, alle n Scheiben unter Einhaltung der folgenden Regeln auf einen anderen Stab zu versetzen:

- Es darf bei jedem Schritt immer nur eine Scheibe bewegt werden
- Es darf nie eine größere Scheibe auf eine kleinere gelegt werden.

Die vorgegebenen Klassen HanoiApp.java, MainPane.java, HanoiPane.java und Scheibe.java realisieren bereits die Benutzeroberfläche.




Der Benutzer kann von Hand die Scheiben umsetzen und über die Auswahl Datei→Neues Spiel… die Anzahl der Scheiben festlegen. Mit Hilfe einer timergesteuerten Animation soll der Umsetzvorgang visualisiert werden.

Entwerfen Sie eine Klasse DiskMover, die auf Anfrage jeweils den nächsten Zug generiert. Testen Sie die Klasse in einer eigenen main-Methode:

```
public static void main(String[] arg) {
      DiskMover dm = new DiskMover(0, 2, 3);
      while (dm.hasMoreMoves()) {
            System.out.println(dm.nextMove());
      }
}
```

Die Stäbe seien von links nach rechts durchnummeriert, also Stab 0,Stab 1, Stab 2. Obiger DiskMover bewegt 3 Scheiben von links nach rechts. Ein DiskMover mit nur einer Scheibe braucht nur einen Zug vom Start zum Ziel. Ein DiskMover mit mehr als einer Scheibe erzeugt sich einen Helfer, der das um eine Scheibe reduzierte Problem löst. Die Methode hasMoreMoves lässt sich einfach realisieren, wenn man dem DiskMover ein Attribut movesLeft gibt und dieses bei jedem Zug dekrementiert.

Es ist hilfreich für einen DiskMover drei Zustände zu unterscheiden: BEFORE_LARGEST, LARGEST und AFTER_LARGEST. Zeichnen Sie ein Zustandsdiagramm und überlegen Sie welche Aktionen die Transitionen auslösen.

**2. Das Maximum nach dem Teile-und-herrsche-Schema suchen**

Gegeben sei ein Array von Ganzzahlen:

```
int a[] = {15, 91, 3, 123, 5, 83, 7, 64, 69, 101};
```

Das Maximum soll rekursiv gefunden werden. Dazu wird das Array in zwei Hälften zerlegt. Das Maximum ist der größere Wert des Maximums der Hälften. Die Halbierung wird solange fortgesetzt bis die Hälften jeweils nur noch ein Element beinhalten. Verwenden sie folgenden Funktionskopf:

```
int max(int a[], int links, int rechts)
```

Zeigen Sie mit Hilfe eines Aufrufbaumes welche Parameter an die Methode max übergeben werden und welche Rückgabewerte sie erzeugt.

```
[ 15, 91, 3, 123, 5, 83, 7, 64, 69, 101 ]
max(0,9)
  max(0,4)
    max(0,2)
      max(0,1)
        max(0,0)
        return 15
        max(1,1)
        return 91
      return 91
      max(2,2)
      return 3
    return 91
    max(3,4)
      max(3,3)
      return 123
      max(4,4)
      return 5
    return 123
  return 123
  max(5,9)
    max(5,7)
      max(5,6)
```

```
        max(5,5)
        return 83
        max(6,6)
        return 7
      return 83
      max(7,7)
      return 64
    return 83
    max(8,9)
      max(8,8)
      return 69
      max(9,9)
      return 101
    return 101
  return 101
return 123
```

### 3. Das Maximum für Teilsummen eines Array finden

Schreiben Sie ein Programm zur Berechnung des Maximums der Teilsummen eines Arrays. Es sollen auch die Indizes für den Anfang und das Ende der Teilserie ausgegeben werden.

z.B.: double a[] = { 5, -11, 2, 4, -1};

Hat die maximale Teilsumme = 6, der Anfangsindex beträgt 2 und der Endindex ist 3.

double a[] = { 0, 5, -11, 2, 3, -1, 4, -2, 2, 23, -65, 1, -6, 8, 7, -13 };

Hat die maximale Teilsumme = 31, der Anfangsindex beträgt 3 und der Endindex ist 9.

a) Iterative Variante mit geschachtelten Schleifen.

```
[5.0, -11.0 ,2.0, 4.0, -1.0]
(0,0)5.0    (0,1)-6.0   (0,2)-4.0   (0,3)0.0    (0,4)-1.0
(1,1)-11.0  (1,2)-9.0   (1,3)-5.0   (1,4)-6.0
(2,2)2.0    (2,3)6.0    (2,4)5.0
(3,3)4.0    (3,4)3.0
(4,4)-1.0
Max: 6.0 Anfang: 2.0 Ende: 3.0
```

b) Teile und Herrsche Ansatz

```
[5.0, -11.0, 2.0, 4.0, -1.0]
maxSub(0,4)
  maxSub(0,2)
    maxSub(0,1)
      maxSub(0,0)
      return Max: 5.0 Anfang: 0 Ende: 0
      maxSub(1,1)
      return Max: -11.0 Anfang: 1 Ende: 1
    return Max: 5.0 Anfang: 0 Ende: 0
    maxSub(2,2)
    return Max: 2.0 Anfang: 2 Ende: 2
  return Max: 5.0 Anfang: 0 Ende: 0
  maxSub(3,4)
    maxSub(3,3)
    return Max: 4.0 Anfang: 3 Ende: 3
    maxSub(4,4)
    return Max: -1.0 Anfang: 4 Ende: 4
  return Max: 4.0 Anfang: 3 Ende: 3
return Max: 6.0 Anfang: 2 Ende: 3
Max: 6.0 Anfang: 2 Ende: 3
```

c) Scan Ansatz

Jedes Datenfeld wird besucht und zur Zwischensumme aufaddiert. Falls die Zwischensumme negativ wird, so wird sie auf 0 gesetzt. Es wird immer geprüft, ob die Zwischensumme ein neues Maximum darstellt.

**4. Selection-Sort**

Ein Array von Buchstaben soll in aufsteigender Reihenfolge sortiert werden.

- Die Position des Minimums im Array wird bestimmt.
- Dann wird es mit dem Element an erster Stelle getauscht.
- Für das Teilarray ab der nächsten Position wird dasselbe durchgeführt, bis man an der vorletzten Stelle aufhören kann.

Schätzen Sie die Laufzeit ab.

Visualisieren Sie den Sortiervorgang: