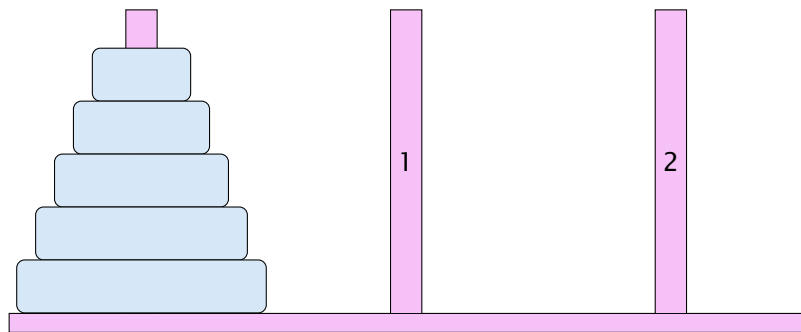


9 Türme von Hanoi



- ▶ Bewege Stapel von links nach rechts.
- ▶ In jedem Zug darf genau ein Ring bewegt werden.
- ▶ Es darf nie ein größerer auf einen kleineren Ring gelegt werden.

9 Türme von Hanoi

Idee

- ▶ Für Turm der Höhe $h = 0$ ist das Problem trivial.
- ▶ Falls $h > 0$ zerlegen wir das Problem in drei Teilprobleme:
 1. Versetze oberen $h - 1$ Ringe auf freien Platz
 2. Bewege die unterste Scheibe zum Ziel
 3. Versetze die zwischengelagerten Ringe zum Ziel
- ▶ Versetzen eines Turms der Höhe $h > 0$ erfordert also zweimaliges Versetzen eines Turms der Höhe $h - 1$.

Es gibt keine andere Möglichkeit!!!

Implementierung

```
1 public static void move(int h, byte a, byte b) {  
2     if (h > 0) {  
3         byte c = free(a,b);  
4         move(h-1,a,c);  
5         write("move "+a+" to "+b+"\n");  
6         move(h-1,c,b);  
7     }  
8 }
```

... bleibt die Ermittlung des freien Rings

Beobachtung

Offenbar hängt das Ergebnis nur von der Summe der beiden Argumente ab...

	0	1	2
0		2	1
1	2		0
2	1	0	

$\text{free}(x,y)$

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

$\text{sum}(x,y)$

Implementierung

Um solche Tabellen **leicht** implementieren zu können stellt **Java** das **switch**-statement zur Verfügung:

```
1 public static byte free(byte a, byte b) {  
2     switch (a + b) {  
3         case 1: return 2;  
4         case 2: return 1;  
5         case 3: return 0;  
6         default: return -1;  
7     }  
8 }
```

Allgemeines Switch-Statement

```
switch (expr) {  
    case const0: (ss0)? (break;)?  
    case const1: (ss1)? (break;)?  
        ...  
    case constk-1: (ssk-1)? (break;)?  
    (default: ssk)?  
}
```

- ▶ `expr` sollte eine ganze Zahl/`char`, oder ein `String` sein.
- ▶ Die `consti` sind Konstanten des gleichen Typs.
- ▶ Die `ssi` sind alternative Statement-Folgen.
- ▶ `default` ist für den Fall, dass keine der Konstanten zutrifft
- ▶ Fehlt ein `break`-Statement, wird mit den Statement-Folgen der nächsten Alternative fortgefahren!

Beispiel

Dies dient nur als Beispiel für die `switch`-Anweisung. Im vorliegenden Fall wäre es übersichtlicher für jeden Monat einen eigenen `case` einzuführen (d.h., kein `default`), und den „fall-through“ in den jeweils nächsten `case` zu vermeiden.

```
1 int numOfDays;  
2 boolean schaltjahr = true;  
3 switch (monat) {  
4     case "April":  
5     case "Juni":  
6     case "September":  
7     case "November":  
8         numOfDays = 30;  
9         break;  
10    case "Februar":  
11        if (schaltjahr)  
12            numOfDays = 29;  
13        else  
14            numOfDays = 28;  
15        break;  
16    default:  
17        numOfDays = 31;  
18 }
```

`monat` darf nicht `null` sein;
man kann nicht mithilfe eines `switch`-statements gegen `null` testen.

Der Bedingungsoperator

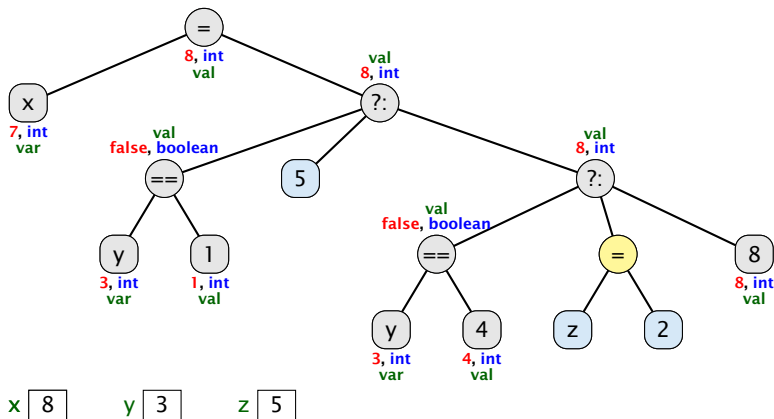
Eine Alternative zu einem `switch` ist der **Bedingungsoperator**:

`condition ? expr1 : expr2`

Der Bedingungsoperator

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
<code>? :</code>	Bedingungsoperator	boolean, 2*typ	rechts	14

Beispiel: $x = y == 1 ? 5 : y == 4 ? z = 2 : 8$



Beispiel

`String` ist ein Referenzdatentyp. Ein Vergleich `monat == "Januar"` vergleicht nur die Referenzen, der `Strings` `monat` und `"Januar"`. Die sind im allgemeinen unterschiedlich, auch wenn `monat` und `"Januar"` den gleichen `Inhalt` haben.

```
numOfDays =
    "Januar".equals(monat) ? 31 :
    "Februar".equals(monat) ? (schaltjahr ? 29 : 28) :
    "Maerz".equals(monat) ? 31 :
    "April".equals(monat) ? 30 :
    "Mai".equals(monat) ? 31 :
    "Juni".equals(monat) ? 30 :
    "Juli".equals(monat) ? 31 :
    "August".equals(monat) ? 31 :
    "September".equals(monat) ? 30 :
    "Oktober".equals(monat) ? 31 :
    "November".equals(monat) ? 30 :
    "Dezember".equals(monat) ? 31 :
    -1 ;
```

Implementierung

Für unseren Fall geht das viel einfacher:

```
1 public static byte free(byte a, byte b) {  
2     return (byte) (3-(a+b));  
3 }
```

9 Türme von Hanoi

Bemerkungen:

- ▶ `move()` ist rekursiv, aber nicht end-rekursiv.
- ▶ Sei $N(h)$ die Anzahl der ausgegebenen Moves für einen Turm der Höhe $h \geq 0$. Dann ist

$$N(h) = \begin{cases} 0 & \text{für } h = 0 \\ 1 + 2N(h-1) & \text{andernfalls} \end{cases}$$

- ▶ Folglich ist $N(h) = 2^h - 1$.
- ▶ Bei genauerer Analyse des Problems lässt sich auch ein nicht ganz so einfacher nicht-rekursiver Algorithmus finden.

Hinweis: Offenbar rückt die kleinste Scheibe in jedem zweiten Schritt eine Position weiter. . .