Rekursion Nachvollziehen

Aufgabe:

- Finden Sie heraus, was die folgende Applikation auf die Konsole schreibt:[1]
- Diskutieren Sie Ihre Lösung mit Ihrem Lernpartner.

```java
package ch.bbw.pr.rekursion;

/**
 * Rekursion App
 * @author Peter Rutschmann
 * @version 10.01.2018
 */
public class Applikation {

    public static void main(String[] args) {
        Rekursion myRekursion = new Rekursion();
        int value = 4;

        System.out.println("Beispiel für eine Rekursion:");
        System.out.println(myRekursion.function(value));
    }
}
```

```java
package ch.bbw.pr.rekursion;

/**
 * 10.01.2018 Rekursion
 *
 * @author Peter Rutschmann
 * @version
 */
public class Rekursion {

    public int function(int v) {
        if (v == 1)
        {
            return 1;
        }
        else
        {
            return function(v - 1) * v;
        }
    }
}
```

Output auf die Consloe

```
Beispiel für eine Rekursion:
```

Zusatzaufgabe:

➔ Was wäre das Resultat, wenn value = 10; wäre?[2]

[1] 24

[2] 3628800

Lösung zur Aufgabe ➔ Es ist die "Fakultät"

Die Fakultät ist in der Mathematikeine Funktion, die einer natürlichen Zahl das Produkt aller natürlichen Zahlen (ohne Null) kleiner und gleich dieser Zahl zuordnet. Sie wird durch ein dem Argument nachgestelltes Ausrufezeichen(„!") abgekürzt. Die Fakultät der Zahl 0 ist definitionsgemäss 1.

4! = 4*3*2*1 = 24

10! = 10*9*8*7*6*5*4*3*2*1 = 3628800

Das Berechnen der Fakultät kann mit einer Schleife gelöst werden.

```
public int byLoop(int value)
{
    int ergebnis = 1;
    for(int i=1; i<=value;i++)
    {
        ergebnis = ergebnis * i;
    }
    return ergebnis;
}
```

```
Fakultaet myFakultaet = new Fakultaet();
int value = 6;

System.out.println("Berechnen der Fakultät von: " + value);
System.out.println();
System.out.println("Berechnung mit Loop: " + myFakultaet.byLoop(value));
```

```
Berechnen der Fakultät von: 6

Berechnung mit Loop: 720
```

Oder man verwendet eine rekursive Funktion

```
20    public int byRekursion(int value)
21    {
22        if (value != 0)
23        {
24            return (value * byRekursion(value-1));
25        }
26        return 1;
27    }
```

Treppe ab und Treppe auf

```
4! = 4*3*2*1 = 24


value = 4


main -> byRekursion(4)
          4* byRekursion (3)                  return 4*6   ➔ 24
             3* byRekursion (2)             return 3*2
               2* byRekursion (1)       return 2*1
                 1* byRekursion (0) return 1
```

Main ruft die Methode *byRekursion(4)* das erste Mal mit dem Wert 4 auf.
Man gelangt zur Zeile 24 und ruft die Methode *byRekursion(4-1) erneut auf.*

Das wiederholt sich, bis der value 0 ist, dann gibt die Methode *byRekursion* ein return 1 zurück.

Es geht die Treppe hoch, wobei der zurückgegebene Wert immer mit dem Value der Treppenstufe multipliziert wird.

Bei der letzten Treppenstufe wird return 4* 6 zurückgegeben und damit 24 ausgegeben.

Weitere Informationen zu Rekursionen

(Quelle: https://www.baeldung.com/)

Vorteile rekursiver Algorithmen

- Kürzere Formulierung
- Leicht gewichtige Lösung
- Einsparung von Variablen
- Meist sehr effiziente Problemlösungen
- Bei rekursiven Datenstrukturen empfehlenswert

Nachteile rekursiver Algorithmen

- Weniger effizientes Laufzeitverhalten (Overhead durch Funktionsaufruf, Stack)
- Schwierig zu schätzendes Speicher-Laufzeitverhalten
- Verständnisprobleme bei ProgrammieranfängerInnen
- Konstruktion rekursiver Algorithmen ist "gewöhnungsbedürftig"
- Nicht bei allen Programmiersprachen möglich

Merkmale von rekursiven Algorithmen

- Methoden-**Signatur mit Parameter** `function(int v)`
- **Abbruchbedingung** `if (v == 1)`
- **Return-Wert** `public int function(`

```
11  public int function(int v) {
12      if (v == 1)
13      {
14          return 1;
15      }
16      else
17      {
18          return function(v - 1) * v;
19      }
20  }
```