Ähnlich wie logische Operatoren, werden aber auf ganzzahlige Werte angewendet.

Hier werden Hexadezimalzahlen genutzt.
0123456789ABCDEF (0-15)
=
4 Bit 0. 2^4

int bitmaske = 0x000F; = 0000 0000 0000 1111
↑
Hexadezimal

Bei den Bitweisen Operatoren wird also jedes einzelne Bit mit dem bitweisen Und-Operator (&) oder dem bitweisen Oder-Operator (|) verglichen.

Bitweisen Und-Operator (&):

int bitmaske = 0x000F;
int wert = 0xAAAA;
int ergebnis = wert & bitmaske;

1010 1010 1010 1010
0000 0000 0000 1111
0000 0000 0000 1010

int ergebnis = 0x000A;

Bitweiser Oder-Operator (|):

```
int bitmaske = 0x000F;
int wert = 0xAAAA;
int ergebnis = wert | bitmaske;
```

```
1010 1010 1010 1010
0000 0000 0000 1111
1010 1010 1010 1111
```

```
int ergebnis = 0xAAAF;
```

Sowohl bei den logischen als auch bei den bitweisen Operatoren gibt es auch noch den entweder Oder-Operator (XOR: ^) der immer dann wahr zurück gibt, falls nur einer der beiden Werte wahr ist.
Also falls beide Werte gleich sind wird hier immer falsch zurück gegeben.

Bitweiser Entweder(oder-Operator:

```
int bitmaske = 0x000F;
int wert = 0xAAAA;
int ergebnis = wert ^ bitmaske;
```

```
1010 1010 1010 1010
0000 0000 0000 1111
1010 1010 1010 0101
```

```
int ergebnis = 0xAA5;
```

Es gibt auch noch das sogenannt Bitshiften. Hiermit kappt man einen vorgegeben Teil der Bitreihe vorne oder hinten ab, rückt verbleibende Bitreihe auf und füllt die dann leeren Plätze mit dem selben Bitwert auf den der naheliegenste Bit hat.

```
int wert = 0xAAAA;
1010 1010 1010 1010
int ergebnis = wert >> 3;
1111 0101 0101 0101
int bitmaske = 0x000F;
0000 0000 0000 1111
int ergebnis = bitmaske >>3;
0000 0000 0000 0001
```

Mit dem erweiterten Bitshift-Operator (>>>) kann man auch festlegen, das die leeren Plätze immer mit 0 aufgefüllt wird.

```
int wert = 0xAAAA;
1010 1010 1010 1010
int ergebnis = wert >>> 3;
1111 0101 0101 0101
int ergebnis = 0x1555;
```

Das funtioniert aber nur in die Rechte-Richtung

Das gleiche Funktioniert auch in die andere Richtung, nur wird hier immer mit 0 aufgefüllt.

```
int wert = 0xAAAA;
1010 1010 1010 1010
int ergebnis = wert << 3;
0101 0101 0101 0000
intergebnis = 0x5550;
```