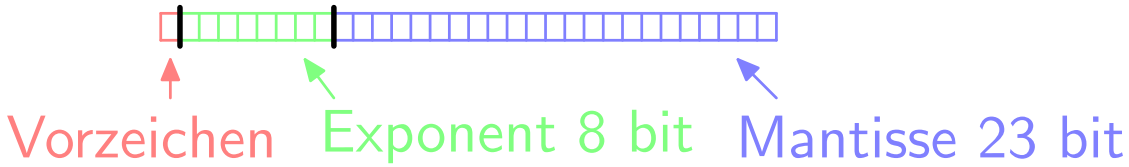
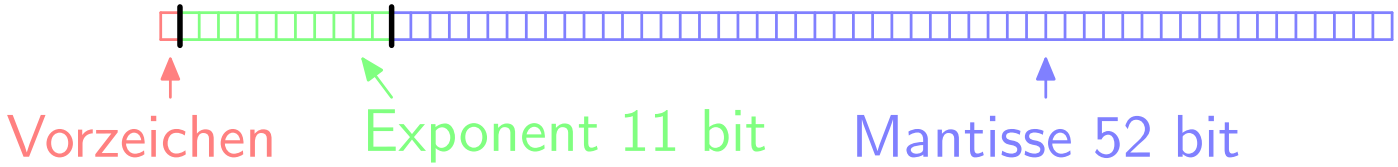
Fließkommazahlen

- **float**: Fließkommazahl nach IEEE 754–Standard mit 32 bit



- **double**: Fließkommazahl nach IEEE 754–Standard mit 64 bit



Die Mantisse ist eine binär kodierte natürliche Zahl, aber die führende Eins wird *nicht* gespeichert.

Der Exponent wird als binär kodierte natürliche Zahl gespeichert und daher um 127 bzw. 1023 erhöht. Vorteil: Fließkommazahlen können wir normale Binärzahlen verglichen werden.

Zuweisungen

Einer Variable können neue Werte zugewiesen werden:

```
int x;
```

⋮

```
x = 5;
x = y;
x = y + z;
x = (42 * y + z) - y * y + 2 * x;
```

Arithmetische Ausdrücke

In absteigender Priorität:

- $*$, $/$ Multiplikation, Division
- $+$, $-$ Addition, Subtraktion

$x + y * z$ bedeutet $x + (y * z)$

$x - y - z$ bedeutet $(x - y) - z$

Beispiel

Kugeloberfläche und -inhalt

```
class Kugel {
  private double radius;
  public Kugel(double r) {
    radius = r;
  }
  public double getSurface() {
    return 4.0 * Math.PI * radius * radius;
  }
  public double getVolume() {
    return 4.0/3.0 * Math.PI * radius * radius * radius;
  }
}
```

Wahrheitswerte

Primitiver Typ: **boolean**

Vergleichsoperationen liefern als Ergebnis **boolean**.

- $x == y$ Gleichheit
- $x != y$ Ungleichheit
- $x < y$ Kleiner
- $x <= y$ Kleiner oder gleich

Es gibt die Konstanten *true* und *false*.

```
int x, y, z;
...
boolean b = (5 * x <= y + z);
```

Wahrheitswerte

Wahrheitswerte können mit *logischen Operationen* verknüpft werden.

- $!a$ Negation
- $a \;\&\&\; b$ logisches Und
- $a \;||\; b$ logisches Oder

Logische Operatoren haben niedrigere Priorität als Vergleichsoperatoren.

Beispiel

```
boolean b = (0 <= x && x < 10);
```

Überblick

Verzweigungen

```
public int maximum(int x, int y) {
  int result;
  if(x < y) {
    result = y;
  }
  else {
    result = x;
  }
  return result;
}
```

Mit einer **if**-Anweisung können wir eine Bedingung prüfen und dann den **then**- oder **else**-Zweig ausführen lassen. Der **else**-Zweig ist optional.

Verzweigungen

```
public int sign(int x) {
  int result;
  if(x < 0) {
    result = -1;
  }
  else if(x > 0) {
    result = +1;
  }
  else {
    result = 0;
  }
  return result;
}
```

Es darf mehrere **else if**-Teile geben.

Schleifen

```
public int dreiecksZahl(int n) {
    int sum = 0;
    int k = 0;
    while(k <= n) {
        num = num + k;
        k = k + 1;
    }
    return sum;
}
```

Wir berechnen hier die Summe

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

Schleifen

```
while(bedingung) {
    anweisung1;
    anweisung2;
    ...
}
```

Solange *bedingung* erfüllt ist, wird der Rumpf der Schleife ausgeführt. Ist *bedingung* anfangs nicht erfüllt, wird der Rumpf gar nicht ausgeführt.

for-Schleifen

```
for(anweisung1; bedingung; anweisung2) {
    rumpf;
}
```

ist gleichbedeutend mit

```
anweisung1;
while(bedingung) {
    rumpf;
    anweisung2;
}
```

for-Schleifen

Beispiel

```
public int dreiecksZahl(int n) {
   int sum = 0;
   for(int k = 1; k <= n; k = k + 1) {
      sum = sum + k;
   }
   return sum;
}
```

Wieder berechnen wir

$$n \mapsto \sum_{k=0}^{n} k = \frac{n(n+1)}{2}.$$

for-Schleifen

```
public int tetraederZahl(int n) {
  int sum = 0;
  for(int k = 1; k <= n; k = k + 1) {
    for(int j = 1; j <= k; j = j + 1) {
      sum = sum + j;
    }
  }
  return sum;
}
```

Wir berechnen

$$n \mapsto \sum_{k=1}^{n} \sum_{j=1}^{k} j = \frac{n(n+1)(n+2)}{6}.$$

Überblick

Arrays

Wir wollen häufig viele Dinge gemeinsam speichern.

```
int a[ ] = new int[10]; // Array der Größe 10
a[0] = 7;
a[4] = 3;
a[9] = 10;
a[10] = 12; // nicht erlaubt!
a[3] = a[0] + a[4];
```