Inhalt

1. String
2. Methoden
3. Formatierungen
4. Uhrzeit-Formate

String

- Aneinanderkettung von chars (String = Zeichenkette)
- Komplexer Datentyp für Texte

```
String example = "Hello";
```

- Aneinanderkettung von Strings durch `+`

```
String example = "Hello" + " World";
```

- An Strings können Strings, Integer, Double, ... mittels + angehängt werden.

Methoden von Strings

- **Verändern Originalstring NICHT**
- Original-String bleibt unverändert
- Kopie wird erstellt.

```
String meinStr = "abc";
String kopieVonStr = meinStr.toUpperCase();

System.out.println(meinStr); // abc
System.out.println(kopieVonStr); // ABC
```

Methoden von Strings

```
String meinStr = "abc";
String leerStr = "";
```

Länge

```
int length = meinStr.length(); // 3
length = leerStr.length(); // 0
```

Zeichen an bestimmter Stelle (1. Zeichen = Index 0)

```
int c = meinStr.charAt(0); // 'a'
c = meinStr.length(1); // 'b'
```

Boolsche Abfragen

Ist String leer?

```
String str = "";
boolean istLeer = str.isEmpty() // true
```

Beinhaltet str b?

```
String str = "abc";
boolean hatB = str.contains("b") // true
```

Startet String mit a?

```
String str = "abc";
boolean hatB = str.startsWith("b") // true
```

Boolsche Abfragen

Endet String mit a?

```
String str = "abc";
boolean hatB = str.endsWith("c") // true
```

Letztes Zeichen bestimmen

```
String meinStr = "abc";
String leerStr = "";
```

- Länge ist 3
- Letzte Position ist 2 (0='a', 1='b', 2='c')

```
int lastCharPos = meinStr.length() - 1;
int lastChar = meinStr.charAt(lastCharPos); // 'c'
```

Erster Index von Substring

0	1	2	3	4	5	6	7	8	9	10
'H'	'a'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'

```java
String myStr = "Hallo World";
// Suche nach String
System.out.println(myStr.indexOf("Wo")); // 46
System.out.println(myStr.indexOf("o")); // 4
// Suche nach String, ab Index
System.out.println(myStr.indexOf("o", 5)); // 7

// Suche nach char
System.out.println(myStr.indexOf('o')); // 4

// Suche nach char, ab Index
System.out.println(myStr.indexOf('o', 5)); // 7
```

Teilstring/Substring

```
String meinStr = "abc";

String bc = meinString.substring(1, 2); // begin index, end index
```

Leerzeichen vorne und hinten entfernen

```
String meinStr = "     abc      ";

String ohneLeerzeichen = meinString.trim(); // "abc"
```

in Klein-/Großbuchstaben umwandeln

```
String meinStr = "Abc";

String klein = meinString.toLowerCase(); // "abc"
String gross = meinString.toUpperCase(); // "ABC"
```

Zeichen ersetzen: replace

```
String meinStr = "abc";

String newStr = meinStr.replace('a', 'x'); // alt, neu => "xbc"
```

Verkettung

- Methodenaufrufe (`.toLowerString(), .replace(), ...`) können aneinandergehängt werden.

```
String o1 = "BBRZ";
String o2 = o1.replace('Z', 'z').substring(0, 2)+"w"; System.out.println(o2+" & "+o1); // Ausgabe: BBRw & BBRZ
```

Vergleiche

- Strings sind **Referenzdatentypen**. Daher wird bei Vergleich nicht der Wert, sondern die Adresse verglichen
- => Daher muss mit `.equals()` verglichen werden.

```
String a = "abc";
String b = "abc";

if(a==b){ /* wird nie betreten */
}else{ /* immer false*/ }
```

- **equals**

```
if(a.equals(b)){
  // wenn gleich
}
```

Strings formatieren

- Werte werden immer lang genau gespeichert (viele Nachkommastellen, Uhrzeit mit Millisekunden)
- Für Ausgabe ist oft Formatierung erforderlich
- Hierzu werden die Werte in **formatierte Strings** umgewandelt
- Beispiele:
  - Gleitkommazahlen
  - Datum/Uhrzeit
  - Währungen

Strings formatieren

- formatiert auf Konsole ausgeben

```
System.out.printf(...);
```

- formatiert in neuen String speichern

```
String.format(...)
```

Beispiel

```
double temperatur = 35.0595;
System.out.printf("%.2f", temperatur);
String formatiert = String.format("%.2f", temperatur);
```

String format

```
String formattedString = String.format(Locale, format, arguments);
```

- `Locale` : (optional) Legt die Spracheinstellungen für die Formatierung fest.
- `format` : Die Zeichenkette mit Platzhaltern.
- `arguments` : Die Werte, die in die Platzhalter eingesetzt werden.

Platzhalter

- `%s` : String
- `%d` : Dezimalzahl (Integer)
- `%f` : Gleitkommazahl (Float)
- `%n` : Neue Zeile
- `%%` : Prozentzeichen

Beispiel `String.format`

```java
String name = "Tommi";
int age = 30;
double salary = 50000.5;

String formattedText = String.format("Name: %s, Alter: %d, Gehalt: %.2f€", name, age, salary);
```

Ausgabe

```
Ergebnis: "Name: Tommi, Alter: 30, Gehalt: 50000.50€"
```

Nummern formatieren

```
double val = 1323.34212345;
String formatiert = String.format("#.00", val)
```

Wert	Beschreibung
0	Eine einzelne Ziffer
#	Eine einzelne Ziffer. Wird ausgelassen, falls führende Null.
.	Dezimaltrennzeichen
,	Tausendertrennzeichen
E	Aktiviert Exponentialdarstellung
%	Ausgabe als Prozentwert

Beispiel

```
double value = 1768.3518;
System.out.printf(print(value, "#0.0"));
System.out.printf(print(value, "#0.000"));
System.out.printf(print(value, "000000.000"));
System.out.printf(print(value, "#.000000"));
System.out.printf(print(value, "#,###,##0.000"));
System.out.printf(print(value, "0.000E00"));
```

Ausgabe

```
1768,4
1768,352
001768,352
1768,351800
1.768,352
1,768E03
```

Nachkommastellen definieren

- Anzahl der Nachkommastellen : `%.1f`
- Vorne mit Leerzeichen auf auffüllen und eine Nachkommastelle : `%10.1f`

```
System.out.printf("Das ist ein %.1f!%n", 2.200002340000234); // Das ist ein 2,2!
System.out.printf("Das ist ein %.2f!%n", 2.200002340000234); // Das ist ein 2,20!
System.out.printf("Das ist ein %.3f!%n", 2.200002340000234); // Das ist ein 2,200!
System.out.printf("Das ist ein %.4f!%n", 2.200002340000234); // Das ist ein 2,2000!

// auf eine Länge von 10 (vor Komma) mit Leerzeichen füllen
System.out.printf("Das ist ein %10.1f!%n", 2.200002340000234); // Das ist ein        2,2!
// auf eine Länge von 10 (vor Komma) mit 0 füllen

System.out.printf("Das ist ein %010.1f!%n", 2.200002340000234); // Das ist ein 00000002,2!
```

Fixe Längen (String)

```java
String myString = "test";
// %s ist Platzhalter für String
// %n steht für Zeilenumbruch
System.out.printf("Das ist ein %s!%n", myString);
// auffüllen von auf 10 Zeichen (rechtsbündig)
System.out.printf("Das ist ein %10s!%n", myString);
// auffüllen von auf 10 Zeichen (linksbündig)
System.out.printf("Das ist ein %-10s!%n", myString);
```

Mögliche Fehler von String.format

MissingFormatArgumentException

- Wenn zu wenig Argumente angegeben werden

```
String message = String.format("Name: %s, Alter: %d", "John");
```

- Zu viele Argumente werden jedoch ignoriert

```
String message = String.format("Name: %s, Alter: %d", "John", 30,
"Zusätzlicher Text");
```

IllegalFormatException

- Wenn die Argumente nicht zum erwarteten Format passen

```
String message = String.format("Name: %s, Alter: %d", 30, "John");
```

Prozentzeichen anzeigen

- Um ein Prozentzeichen selbst anzuzeigen, musst es es doppelt geschrieben sein: `%%`

```
String message = String.format("Der Rabatt beträgt 10%% auf ausgewählte Artikel.");
```

Kommatrennung/Local

```
String formattedText = String.format(Locale.GERMAN, "Preis: %.2f Euro", 19.99);
```

Parsen von Strings

- Das Parsen von Strings in Java bezieht sich auf die Konvertierung von Zeichenketten in andere Datentypen wie Zahlen, Datum oder benutzerdefinierte Objekte.
- **Casten** `int x = (int) 123.45` => `x` wird zu `123`
  - Umwandlung eines Datentyps in einen anderen innerhalb einer bestimmten Klassenhirarchie
- **Parsen**
  - Umwandlung einer Zeichenkette in einen anderen Datentyp

```java
String numberStr = "42";
int parsedNumber = Integer.parseInt(numberStr);
// Achtung auf Formatierung.
String decimalStr = "3.14";
double parsedDecimal = Double.parseDouble(decimalStr);
```

Mit Formatierung

```
NumberFormat nf = NumberFormat.getInstance(Locale.GERMAN);
try {
    nf.parse("123,444");
} catch (ParseException e) {
    throw new RuntimeException(e);
}
```

Fehlerbehung bei Parsen

```
try {
    int parsedNumber = Integer.parseInt("asdf");
} catch (NumberFormatException e) {
    System.err.println("Fehler beim Parsen: " + e.getMessage());
}
```

Uhrzeit formatieren

- `LocalDateTime` ist eine Klasse in der Java Standardbibliothek (java.time), die Datum und Uhrzeit ohne Berücksichtigung von Zeitzonen darstellt.

```java
// imports ganz oben
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
// ...

LocalDateTime currentTime = LocalDateTime.now();

// Defintion von Format
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss");

// Formatieren und Ausgabe der aktuellen Uhrzeit
String formattedTime = currentTime.format(formatter);
System.out.println("Aktuelle Uhrzeit: " + formattedTime);
```

Formatierungen

- **H**: Stunde des Tages im 24-Stunden-Format (00-23).
- **h**: Stunde des Tages im 12-Stunden-Format (01-12).
- **m**: Minute (00-59).
- **s**: Sekunde (00-59).
- **S**: Millisekunde (0-999).
- **a**: Vormittag (AM) oder Nachmittag (PM).
- **k**: Stunde des Tages im 1-basierten 24-Stunden-Format (1-24).
- **K**: Stunde des Tages im 1-basierten 12-Stunden-Format (1-12).
- **z**: Zeitzone-Abkürzung (z.B. "PST" für Pacific Standard Time).
- **Z**: Zeitzone-Offset (z.B. "+0800" für 8 Stunden vor UTC).

LocalDateTime

Aktuelles Datum/Uhrzeit anlegen

```
LocalDateTime currentDateTime = LocalDateTime.now();
LocalDateTime firstDayYear = LocalDateTime.of(2023, 1, 1);
```

Manipulation

- Monate, Tage, Stunden, Minuten, Sekunden, ... addieren/subtrahieren

```
LocalDateTime futureDateTime = currentDateTime.plusHours(2);
LocalDateTime pastDateTime = currentDateTime.minusHours(3);
```

Achtung

Beachten Sie, dass LocalDateTime keine Zeitzoneninformationen speichert. Verwenden von `ZonedDateTime`, um Zeitzonen zu berücksichtigen.