



STRINGS

Zeichenketten



Zeichenketten

- Zeichenketten - Definition
- Literalschreibweise
- Plus – Operator
- Konstruktoren
- Umwandlung I: Primitiven Datentypen in Strings
- Umwandlung II: Strings in primitive Datentypen
- Methoden für Teilstring – Vergleiche
- Methoden für Manipulation von Strings

DEFINITION



Definition von Zeichenketten

- Zeichenketten bestehen aus einer Sequenz von Zeichen.
- Sie werden für die Repräsentation von textuellen Informationen (Wörter, Sätze, Aufzählungen etc.) verwendet.
- Zeichenketten sind keine primitiven Datentypen
- Zeichenketten sind in Java Instanzen der Klasse *java.lang.String*

LITERAL SCHREIBWEISE



Literal Schreibweise

- Die Literal-Schreibweise ermöglicht es einen String als einen Wert, ähnlich wie eine Zahl oder einen Wahrheitswert, hinzuschreiben.

```
String gruss = "Hallo Welt";
```

PLUS OPERATOR

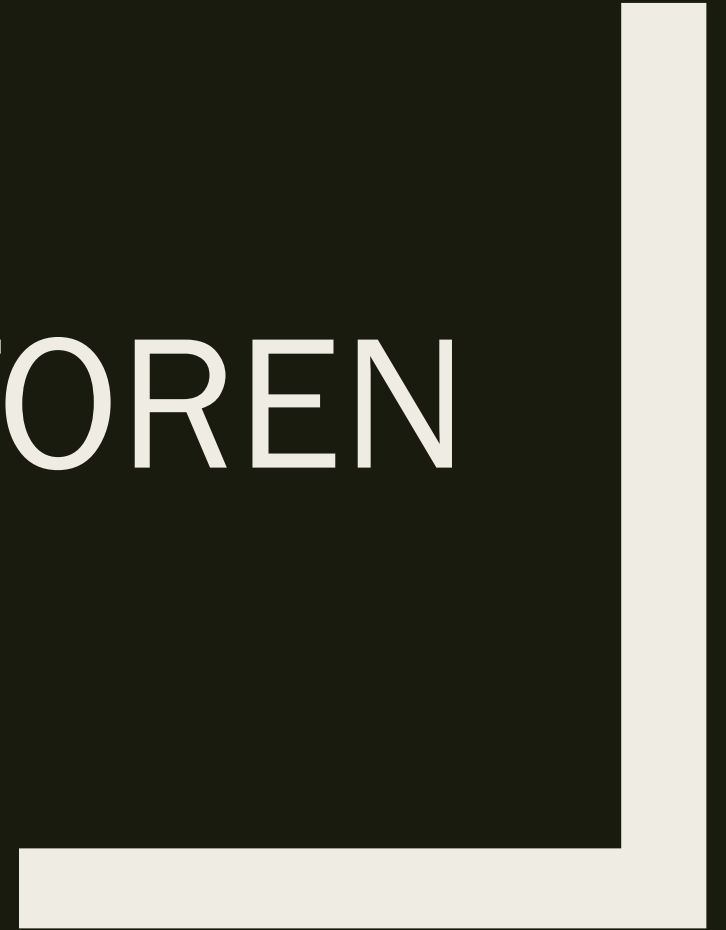


Plus Operator

- In Java ist der Plus-Operator für Zeichenketten definiert.
- So können Strings mit anderen Strings und primitiven Werten verknüpft werden.

```
String gruss2 = "Hallo Programmierer";  
gruss2 = gruss2 + ",";  
String ausgabe = gruss2 + " es wurden "  
                + args.length + " Parameter uebergeben.";   
System.out.println(ausgabe);
```


KONSTRUKTOREN



Konstrukturen

- Die Klasse „String“ stellt mehrere Konstrukturen zur Verfügung, mit denen sich String-Instanzen erzeugen lassen.
- Es gibt den **parameterlosen Konstruktor**, der eine Instanz erzeugt, die einen leeren String repräsentiert.

```
String leer = new String();  
System.out.println(leer);
```

- Der Copy-Konstruktor erzeugt eine Kopie eines übergebenen Strings.

```
String kopie = new String("Original");  
System.out.println(kopie);
```

Konstrukturen

- Es ist ebenfalls möglich, eine String-Instanz aus einem Char- oder Byte-Array von Bytes zu erzeugen.

```
char[] cArray = {'J', 'a', 'v', 'a'};  
String ausChars = new String(cArray);  
System.out.println(ausChars);
```

- Es stehen allerdings keine Konstrukturen für die Erzeugung von Strings aus Wahrheitswerten oder Zahlen zur Verfügung.

UMWANDLUNG



Umwandlung I

- Ein primitiver Wert kann mit einem Leerstring verkettet werden, um durch implizite Konvertierung eine String-Repräsentation zu erhalten.

```
int a = 12;  
float f = 1.45F;
```

```
String aString1 = "" + a;  
System.out.println(aString1);  
String fString1 = "" + f;  
System.out.println(fString1);
```

Umwandlung I

- Die Klasse „String“ enthält die statische Methode *valueOf*, die für alle primitiven Datentypen überladen ist und deren Werte als Zeichenkette interpretiert zurückgibt.

```
int a = 12;  
float f = 1.45F;
```

```
String aString2 = String.valueOf(a);  
System.out.println(aString2);  
String fString2 = String.valueOf(f);  
System.out.println(fString2);
```

Umwandlung II

- Für die Umwandlung von String-Repräsentationen in echte Werte verwendet man die statischen Methoden der Wrapper-Klassen für die primitiven Datentypen: Boolean, Byte, Short, Integer, Float und Double
- String muss gültiges Format besitzen, ansonsten wird bei der Umwandlung eine *NumberFormatException* geworfen
- String -> boolean

```
String bString = "true";  
boolean bool = Boolean.parseBoolean(bString);  
System.out.println(bool);
```

- String -> byte

```
String byteString = "127";  
byte byt = Byte.parseByte(byteString);  
System.out.println(byt);
```

Umwandlung II

- String -> short

```
String shortString = "255";  
short s = Short.parseShort(shortString);  
System.out.println(s);
```

- String -> int

```
String intString = "12345";  
int i = Integer.parseInt(intString);  
System.out.println(i);
```

- String -> long

```
String longString = "123456789";  
long l = Long.parseLong(longString);  
System.out.println(longString);
```


Umwandlung II

- String -> float

```
String floatString = "12.4";  
float f2 = Float.parseFloat(floatString);  
System.out.println(f2);
```

- String -> double

```
String doubleString = "1.125738999";  
double d = Double.parseDouble(doubleString);  
System.out.println(d);
```

Umwandlung II

- String (ungültiges Format) -> Zahl

```
String keineZahl = "ag4612";  
int zahl = Integer.parseInt(keineZahl);  
System.out.println(zahl);
```

- Fehlermeldung

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "ag4612"  
    at java.lang.NumberFormatException.forInputString(Unknown Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at java.lang.Integer.parseInt(Unknown Source)  
    at strings.StringsUndDates.main(StringsUndDates.java:155)
```

TEILSTRING VERGLEICH



Teilstring vergleiche

- Die Instanzen der Klasse „String“ bieten einige Instanz Methoden an, mit denen man den Inhalt der Strings analysieren und verarbeiten kann.
- Demo der Instanz Methoden mit „*String text*“

```
String text = "Initializes a newly created String object "  
    + "so that it represents the same sequence of "  
    + "characters as the argument; in other words, "  
    + "the newly created string is a copy of the argument string.";
```

Teilstring vergleiche

- *boolean contains (String s)*

true: String enthält den Teilstring im Parameter s

false: String enthält den Teilstring im Parameter s nicht

```
System.out.println("text.contains(\"Initializes\")");  
System.out.println(text.contains("Initializes"));
```

Teilstring vergleiche

- *boolean startsWith(String prefix)*

true: String beginnt mit dem Parameter prefix

false: String beginnt nicht mit dem String im Parameter prefix

```
System.out.println("text.startsWith(\"Initializes\")");  
System.out.println(text.startsWith("Initializes"));
```

- *boolean endsWith(String prefix)*

true: String endet mit dem Parameter suffix

false: String endet nicht mit dem String im Parameter suffix

```
System.out.println("text.endsWith(\"string.\")");  
System.out.println(text.endsWith("string."));
```

Teilstring vergleiche

- *int indexOf (char c) & int indexOf (String s)*

Gibt die Stelle im String zurück, an der das Zeichen bzw. der String im Parameter zum ersten Mal auftaucht.

Gibt -1 zurück, falls das Zeichen bzw. der String nicht auftaucht.

```
System.out.println("text.indexOf('Z')");  
System.out.println(text.indexOf('Z'));  
System.out.println("text.indexOf('c')");  
System.out.println(text.indexOf('c'));  
System.out.println("text.indexOf(\"so\")");  
System.out.println(text.indexOf("so"));
```

Teilstring vergleiche

- *int lastIndexOf (char c) & int lastIndexOf (String s)*

Gibt die Stelle im String zurück, an der das Zeichen bzw. der String im Parameter zum letzten Mal auftaucht.

Gibt -1 zurück, falls das Zeichen bzw. der String nicht vorkommt.

```
System.out.println("text.lastIndexOf('Z')");  
System.out.println(text.lastIndexOf('Z'));  
System.out.println("text.lastIndexOf('c')");  
System.out.println(text.lastIndexOf('c'));  
System.out.println("text.lastIndexOf(\"so\")");  
System.out.println(text.lastIndexOf("so"));
```


STRING MANIPULATION



String Manipulation

- *char charAt(int index)*

Gibt das Zeichen an der durch index bezeichneten Stelle im String zurück.

```
System.out.println("text.charAt(159)");  
System.out.println(text.charAt(159));
```

- *String substring(int anfang, int ende)*

Gibt den Teilstring zwischen der Stelle anfang (einschließlich) und ende (ausschließlich) zurück.

```
System.out.println(text.substring(0, 11));  
System.out.println(text.substring(0));
```

String Manipulation

- *String* `replace(char alt, char neu)`
String `replace(String alt, String neu)`

Gibt einen String zurück, in dem alle Vorkommnisse des Zeichens bzw. der Zeichenkette alt durch die Zeichenkette neu ersetzt wurden.

```
String alt = "Allocates a new string that contains the sequence of "  
            + "characters currently contained in the string buffer argument.";

alt.replace('a', '_');
System.out.println(alt);
String neu = alt.replace('a', '_');
System.out.println(alt.replace('a', '_'));
System.out.println(neu);
neu = alt.replace("string", "gnirts");
System.out.println(neu);
```

String Manipulation

- *String toLowerCase()*

Gibt einen String zurück, in dem alle Zeichen des Originalstrings klein geschrieben sind.

```
String alt = "Allocates a new string that contains the sequence of "  
            + "characters currently contained in the string buffer argument.";
```

```
alt.toLowerCase();  
System.out.println(alt);  
neu = alt.toLowerCase();  
System.out.println(neu);
```

String Manipulation

- *String toUpperCase()*

Gibt einen String zurück, in dem alle Zeichen des Originalstrings groß geschrieben sind.

```
String alt = "Allocates a new string that contains the sequence of "  
            + "characters currently contained in the string buffer argument.";
```

```
alt.toUpperCase();  
System.out.println(alt);  
neu = alt.toUpperCase();  
System.out.println(neu);
```

String Manipulation

- *String trim()*

Gibt einen String zurück, bei dem alle nichtdruckbaren Zeichen (Leerzeichen, Umbrüche, Tabulatoren etc.) am Anfang und am Ende des Originalstrings entfernt wurden.

```
String untrimmed = "    Huhu    ";  
untrimmed.trim();  
System.out.println(untrimmed);  
String trimmed = untrimmed.trim();  
System.out.println(trimmed);
```

String Manipulation

- Strings sind in Java unveränderlich.
- Es ist nicht möglich einen String direkt zu manipulieren, alle verändernden Methoden liefern immer ein neues String-Objekt zurück.
- Strings sind echte Objekte, deren interner Zustand (die einzelnen Zeichen, aus denen sie bestehen) aber nicht mehr geändert werden kann.

VERGLEICHEN VON STRINGS



Vergleichen von Strings

- `equals()`: gibt true zurück, wenn zwei Strings exakt dieselben Zeichen in derselben Reihenfolge enthalten. Dabei wird auch auf Groß- und Kleinschreibung geachtet.
- `equalsIgnoreCase()`: gibt true zurück, wenn zwei Strings exakt dieselben Zeichen in derselben Reihenfolge enthalten. Auf Groß- und Kleinschreibung wird nicht geachtet.

Vergleichen von Strings

- *boolean equals()*

```
String name1 = "Anna";  
String name2 = "Anna";  
String name3 = name2;
```

```
System.out.println("name1.equals(name2)");  
System.out.println(name1.equals(name2));
```

```
System.out.println("name1.equals(name3)");  
System.out.println(name1.equals(name3));
```

```
System.out.println("name2.equals(name3)");  
System.out.println(name2.equals(name3));
```

Vergleichen von Strings

- *boolean equalsIgnoreCase()*

```
String name1 = "Anna";  
String name2 = "Anna";  
String name3 = name2;
```

```
System.out.println("name1.equalsIgnoreCase(name2)");  
System.out.println(name1.equalsIgnoreCase(name2));
```

```
System.out.println("name1.equalsIgnoreCase(name3)");  
System.out.println(name1.equalsIgnoreCase(name3));
```

```
System.out.println("name2.equalsIgnoreCase(name3)");  
System.out.println(name2.equalsIgnoreCase(name3));
```

Vergleichen von Strings

- *compareTo()*: vergleicht zwei Strings anhand ihrer lexikografischen Ordnung. Funktioniert wie bei der Sortierung im Telefonbuch, wo ein Name vor einem eingeordnet wird, wenn er mit einem Buchstaben anfängt, der früher im Alphabet auftaucht
- *compareToIgnoreCase()*: Funktioniert wie *compareTo()*. Groß und Kleinschreibung wird ignoriert

Vergleichen von Strings

- *int compareTo()*

```
String name4 = "Anna";  
String name5 = "Bertha";  
String name6 = "Christa";
```

```
System.out.println("name4.compareTo(name5)");  
System.out.println(name4.compareTo(name5));
```

```
System.out.println("name5.compareTo(name4)");  
System.out.println(name5.compareTo(name4));
```

```
System.out.println("name6.compareTo(name6)");  
System.out.println(name6.compareTo(name6));  
System.out.println("");
```

```
System.out.println("name4.compareTo(name6)");  
System.out.println(name4.compareTo(name6));
```

```
System.out.println("name6.compareTo(name4)");  
System.out.println(name6.compareTo(name4));
```

Vergleichen von Strings

- *int compareToIgnoreCase()*

```
String name7 = "anna";  
String name8 = "Anna";  
String name9 = "bertha";  
String name10 = "Bertha";
```

```
System.out.println("name7.compareToIgnoreCase(name8)");  
System.out.println(name7.compareToIgnoreCase(name8));
```

```
System.out.println("name7.compareTo(name8)");  
System.out.println(name7.compareTo(name8));
```