Die: https://o7planning.org/de/10217/anleitung-java-string-stringbuffer-und-stringbuilder
bearbeitet von J. Rau 08.2019

Java String, StringBuffer und StringBuilder

1- Grundlegendes

Java hat 3 Klassen zum Umgang mit Zeichenketten **String**, **StringBuffer** und **StringBuilder**, die sich im Wesentlichen ähneln.

2- Zeichenfolge

2.1- String ist eine besondere Klasse

String ist eine wichtige Java-Klasse. Sobald du den Befehl **System.out.println ()** zur Ausgabe benutzt, wird ein **String** zur Ausgabe auf dem Bildschirm erzeugt.

Sie wird häufig im Programm benutzt, deshalb muss sie effizient und flexibel sein. Das ist der Grund warum **String** ein **Objekt und (!) ein primitiver Datentyp** ist.

(1) Primitiv - String literal

String literal wird im Stack gespeichert. Es erfordert nur geringen Speicher.

```
String literal = "Hello World";
```

- sehr einfacher Operator + für die Verbindung von 2 Strings

- **String literals** mit gleichem Inhalt belegen eine gemeinsame Speicherzelle, das spart Speicher.

(2) Objekteigenschaft - String I

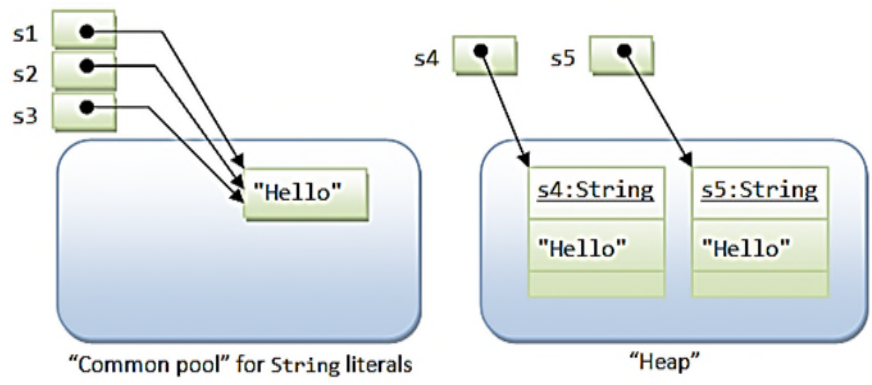
String ist eine Klasse, ein String-Objekt wird deshalb durch den Operator **new** erstellt

```
String-Objekt = new String("Hello World");
```

Die **String-Objekte** werden wie alle Objekte im **Heap** gelagert, dies erfordert ein komplexes Speichermanagement und nimmt mehr Speicher ein. Zwei **String-Objekte** mit dem gleichen Inhalt liegen in zwei unterschiedlichen Speicherzellen des Heap.

2.2- String-Literal vs. String-Objekt

```
String s1 = "Hello";              // String literal
String s2 = "Hello";              // String literal
String s3 = s1;                   // same reference
String s4 = new String("Hello");  // String object
String s5 = new String("Hello");  // String object
```



Die **String literal** mit gleichem Inhalt werden in einem Speicherraum im String pool gelagert.
String-Objekte werden im Heap gelagert und benutzen keinen gemeinsamen Speicher, obwohl die beiden String-Objekte den gleichen Inhalt haben.

Die: https://o7planning.org/de/10217/anleitung-java-string-stringbuffer-und-stringbuilder
bearbeitet von J. Rau 08.2019

Die Methode **equals ()** vergleicht den **Inhalt zweier String-Objekte**.

Der **== Operator** vergleicht den **benutzten Speicherraum**.

Da einfache Datentypen (int, double, char…) mit gleichem Inhalt gleiche Speicherräume nutzen, ist er für deren Vergleich tauglich. In der Praxis sollten Sie **string literal** statt des **new-Operators** benutzen.

```
Zeichenfolge s1 = "Hello";              String literal
Zeichenfolge s2 = "Hello";              String literal
Zeichenfolge s3 = s1;                   gleiche Referenz/Zeiger
String s4 = new String("Hello");        String-Objekt
String s5 = new String("Hello");        String-Objekt
---------------------------------------------------------------------
s1 == s1;               true, gleicher Zeiger
s1 == s2;               true, s1 und s2 teilen sich Speicher im gemeinsamen Pool
s1 == s3;               true, s3 wird derselbe Zeiger wie s1 zugewiesen
s1 == s4;               false, verschiedene Zeiger
s4 == s5;               false, verschiedene Zeiger im Heap

s1.equals(s3);          wahr, gleicher Inhalt
s1.entspricht(s4);     wahr, gleicher Inhalt
s4.equals(s5);          wahr, gleicher Inhalt
```

3- Methoden der Klasse String

Methode	**Beschreibung**
`char charAt (int index)`	Gibt das Zeichen am angegebenen Index zurück.
`int compareTo (String anotherStr)`	Vergleicht zwei Zeichenfolgen lexikographisch.
`int compareToIgnoreCase (String str)`	Vergleicht zwei Zeichenfolgen lexikographisch und ignoriert Groß-/Kleinschreibung.
`String concat (String str)`	Verkettet die angegebene Zeichenfolge mit dem Ende dieser Zeichenfolge.
`boolean endsWith (String suffix)`	Testet, ob diese Zeichenfolge mit dem angegebenen Suffix endet.
`boolean equals (String str)`	Vergleicht diese Zeichenfolge mit dem angegebenen Objekt.
`boolean equalsIgnoreCase (String anotherStr)`	Vergleicht mit einer anderen Zeichenfolge, wobei Groß-/Kleinschreibung ignoriert werden.
`int indexOf (char z)`	Gibt den Index innerhalb dieser Zeichenfolge des ersten Vorkommens des angegebenen Zeichens zurück.
`int indexOf (char z, int fromIndex)`	Gibt den Index innerhalb dieser Zeichenfolge des ersten Vorkommens des angegebenen Zeichens

	zurück, wodurch die Suche am angegebenen Index gestartet wird.
`int indexOf (String str)`	Gibt den Index des ersten Vorkommens der angegebenen Teilzeichenfolge zurück.
`int indexOf (String str, int fromIndex)`	Gibt den Index des ersten Vorkommens der angegebenen Teilzeichenfolge zurück, beginnend mit dem angegebenen Index.
`int lastIndexOf (char z)`	Gibt den Index innerhalb dieser Zeichenfolge des letzten Vorkommens des angegebenen Zeichens zurück.
`int lastIndexOf (char z, int fromIndex)`	Gibt den Index des letzten Vorkommens des angegebenen Zeichens zurück und sucht rückwärts beginnend vom angegebenen Index.
`int lastIndexOf (String str)`	Gibt den Index des letzten Vorkommens der angegebenen Teilzeichenfolge zurück.
`int lastIndexOf (String str, int fromIndex)`	Gibt den Index des letzten Vorkommens der angegebenen Teilzeichenfolge zurück, wobei rückwärts beginnend mit dem angegebenen Index gesucht wird.
`int length ()`	Gibt die Länge dieser Zeichenfolge zurück.
`boolean matches (String regex)`	Gibt an, ob die Zeichenfolge mit dem angegebenen regulären Ausdruck übereinstimmt.
`boolean regionMatches (int start1, String other, int start2, int len)`	Testet, ob zwei Zeichenfolgenbereiche gleich sind.
`String replace (char oldChar, char newChar)`	Gibt eine neue Zeichenfolge zurück, die sich aus dem Ersetzen aller Vorkommen von oldChar in dieser Zeichenfolge durch newChar ergibt.
`String replaceAll (String regex, String replacement)`	Ersetzt jede Teilzeichenfolge dieser Zeichenfolge, die dem angegebenen regulären Ausdruck entspricht, mit der angegebenen Ersetzung.
`String replaceFirst (String regex, String replacement)`	Ersetzt die erste Teilzeichenfolge dieser Zeichenfolge, die dem angegebenen regulären Ausdruck entspricht, mit der angegebenen Ersetzung.
`String [] split (String regex)`	Teilt diese Zeichenfolge um Übereinstimmungen des angegebenen regulären Ausdrucks auf.

`boolean startsWith (String prefix)`	Testet, ob diese Zeichenfolge mit dem Präfix beginnt.
`boolean startsWith (String prefix, int index)`	Testet, ob diese Zeichenfolge mit dem angegebenen Präfix beginnt, das einen angegebenen Index beginnt.
`String substring (int beginIndex)`	Gibt eine neue Zeichenfolge zurück, die eine Teilzeichenfolge dieser Zeichenfolge ist.
`String substring (int beginIndex, int endIndex)`	Gibt eine neue Zeichenfolge zurück, die eine Teilzeichenfolge dieser Zeichenfolge ist.
`String toLowerCase ()`	Konvertiert alle Zeichen in dieser Zeichenfolge mithilfe der Regeln des Standardgebietsschemas in Kleinbuchstaben.
`String toUpperCase ()`	Konvertiert alle Zeichen in dieser Zeichenfolge in Großbuchstaben mithilfe der Regeln des Standardgebietsschemas.
`String trim ()`	Gibt eine Kopie der Zeichenfolge zurück, wobei vorn und hinten Leerzeichen weggelassen werden.

3.1- Länge ()

```
public class LengthDemo {
       String str = "Dies ist Text";
public LenghtDemo  ()  {
       int len = str.length();
       System.out.println("String Length is : " + len);   }}
```

Ergebnis:

3.2- concat (String)

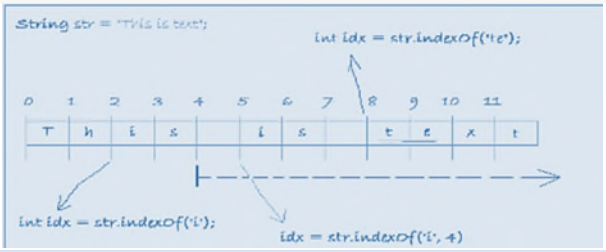
```
public class ConcatDemo {
        String s1 = "Eins";
        String s2 = "Zwei";
        String s3 = "Drei";
public ConcatDemo ()  {
//s1.concat(s2) wie s1 + s2
        String s = s1.concat(s2);
        System.out.println("s1.concat(s2) = " + s);
//s1.concat(s2).concat(s3) wie s1 + s2 + s3;
        s = s1.concat(s2).concat(s3);
        System.out.println("s1.concat(s2).concat(s3) = " + s);    }}
```

Ergebnis:

Die: https://o7planning.org/de/10217/anleitung-java-string-stringbuffer-und-stringbuilder
bearbeitet von J. Rau 08.2019

3.3- indexOf (..)

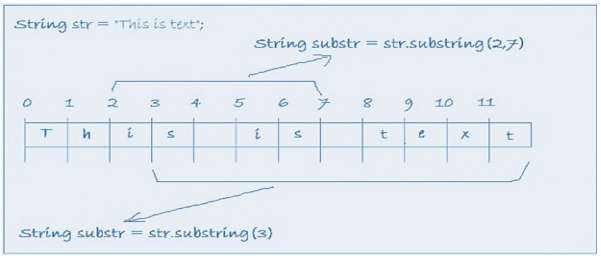
```
public class IndexOfDemo {
        String str = "Dies ist Text";
public IndexOfDemo  ()  {
        int idx = str.indexOf('i');
        System.out.println("- indexOf('i')
= " + idx);
        idx = str.indexOf('i', 4);
        System.out.println("- indexOf('i',4) = " + idx);
        idx = str.indexOf("te");
        System.out.println("- indexOf('te') = " + idx);    }}
```



Ergebnis:

3.4- Teilzeichenfolge (..)

```
public class SubstringDemo {
        String str = "Dies ist Text";
public SubstringDemo ()  {
        String substr = str.substring(3);
        System.out.println("- substring(3)=" +substr);
        substr = str.substring(2, 7);
        System.out.println("- substring(2, 7) =" + substr);    }}
```



Ergebnis:

3.5- ersetzen

```
public class ReplaceDemo {
        String str = "Dies ist Text";
public ReplaceDemo ()  {
        String s2 = str.replace('i', 'x');
        System.out.println("- s2=" + s2);
        String s3 = str.replaceAll("is", "abc");
        System.out.println("- s3=" + s3);
        String s4 = str.replaceFirst("is", "abc");
        System.out.println("- s4=" + s4);
//is|te": bedeutet "ist" oder "te" ersetzt durch "+".
       String s5 = str.replaceAll("is|te", "+");
       System.out.println("- s5=" + s5);    }}
```

Ergebnis:

Die: https://o7planning.org/de/10217/anleitung-java-string-stringbuffer-und-stringbuilder
bearbeitet von J. Rau 08.2019

3.6- weitere Beispiele

```
public class StringOtherDemo {
        String str = "Dies ist Text";
public StringOthertDemo  ()  {
        System.out.println("- str=" + str);
        String s2 = str.toLowerCase();
        System.out.println("- s2=" + s2);
        String s3 = str.toUpperCase();
        System.out.println("- s3=" + s3);
        boolean swith = str.startsWith("This");
        System.out.println("- 'str' startsWith This ? " + swith);
//Eine Zeichenfolge mit Leerzeichen am Anfang und am Ende.
//Hinweis: \t ist ein Tabulatorzeichen,    \n ist ein Zeilenumbruch
        str = " \t Java ist heiß!  \t \n ";
        System.out.println("- str=" + str);
        String s4 = str.trim();
        System.out.println("- s4=" + s4);
}
```

Ergebnis: