Neue Themen für die 808

Februar 2025

SMART INDUSTRY CAMPUS

Plan für die Woche

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
• Selbsteinschätzung • Heap und Stack • Garbage Collector und Object Lifecycle	• Exceptions	• StringBuilder vs. String	• Calender API	• Wiederholung ArrayLists • Lambdas

SMART INDUSTRY CAMPUS

Plan für heute

- Wiederholung Strings
- Wiederholung StringBuilder
- Strings vs. StringBuilder

SMART INDUSTRY CAMPUS

Wiederholung Strings

BUCHSEITEN S.102-111

Creating and Manipulating Strings

- Initialisieren:

```
String s = "Hallo";
```

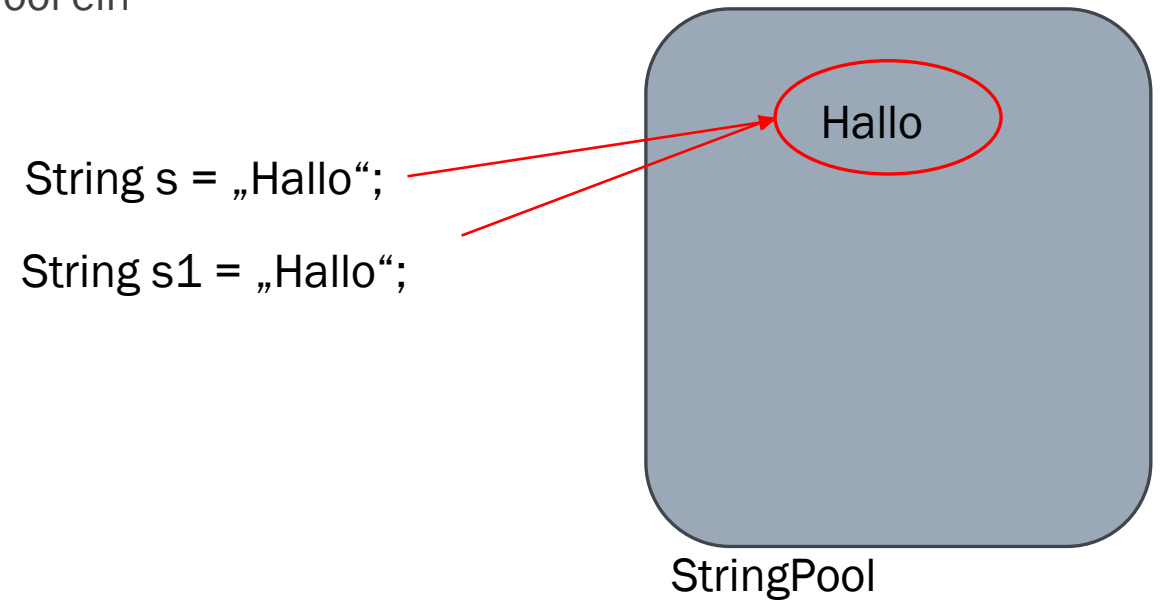
Landet im StringPool

```
String s1 = new String("Hallo");
```

Landet im StringPool und im Heap

StringPool

- ist eine Speicherstelle, in welcher Java String-Literale abspeichert
- bevor ein String-Literal in den StringPool landet, checkt Java, ob es nicht bereits existiert
  - Wenn es existiert, wird die Referenz auf das Bestehende Literal zurückgegeben
  - Wenn nicht, fügt Java das Literal in den StringPool ein



SMART INDUSTRY CAMPUS

StringPool

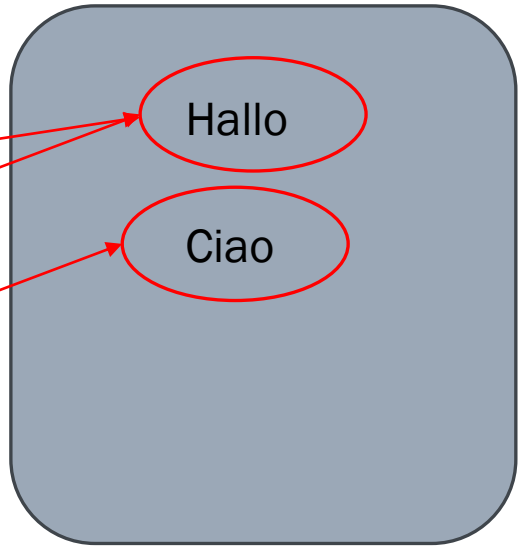
- ist eine Speicherstelle, in welcher Java String-Literale abspeichert
- bevor ein String-Literal in den StringPool landet, checkt Java, ob es nicht bereits existiert
  - Wenn es existiert, wird die Referenz auf das Bestehende Literal zurückgegeben
  - Wenn nicht, fügt Java das Literal in den StringPool ein

String s = „Hallo“;

String s1 = „Hallo“;

String s2 = „Ciao“;

-> s == s1 würde true ergeben



StringPool

String Heap

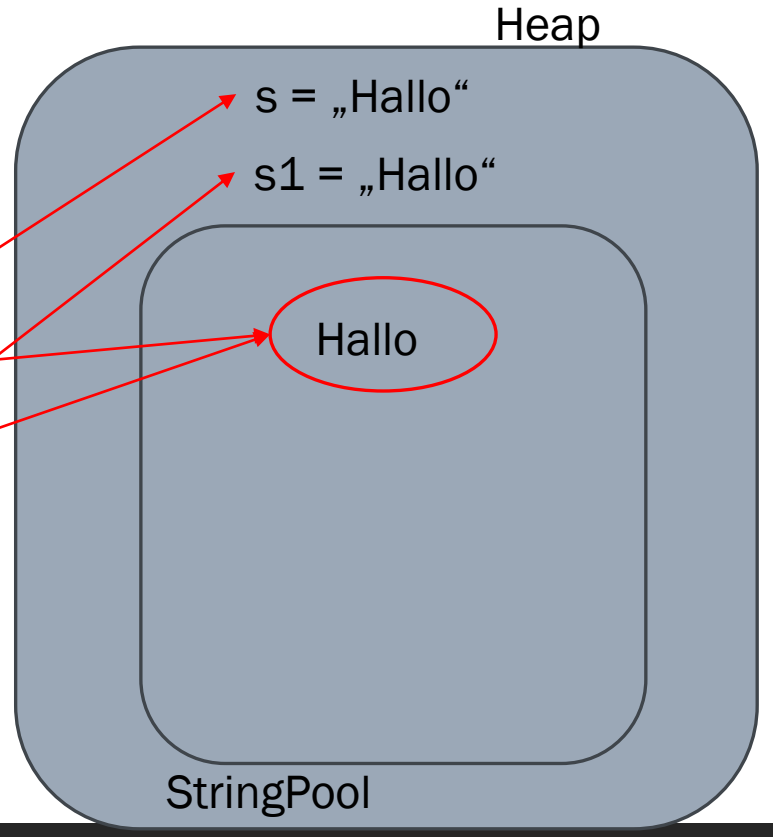
– erzeugt man ein String-Objekt mit new String() landet das Objekt im Heap

```
String s1 = new String("Hallo");
```

String s = new String(„Hallo“);

String s1 = new String(„Hallo“);

-> s == s1 würde false ergeben, da Referenzen unterschiedlich



OB LITERAL ODER OBJEKT, INHALTE MITTELS .equals() VERGLEICHEN

Concatenation (+)

- Strings kann man mittels „+“ verbinden
  - Die anderen Operatoren *, - , : ,... sind nicht kompatibel mit Strings
- das Resultat der Concatenation landet im Heap als Objekt

```
String str1 = "Hallo";
String str2 = "Welt";
String str3 = str1 + " " + str2;
```

→ Landet auf dem Heap

Concatenation (+)

- bei Concatenation muss man auf folgendes achten:
  - Numerische Operanden werden addiert, solange vor ihnen kein String steht

Was ist die Ausgabe?

- bei Concatenation muss man auf folgendes achten:
  - Numerische Operanden werden addiert, solange vor ihnen kein String steht

```
System.out.println(1 + 2);
System.out.println(1 + 2 + " Hallo");
System.out.println("Hallo" + " Tschüss");
System.out.println("Hallo" + 1 + 2);
```

A. 3
   3 Hallo
   Hallo Tschüss
   Hallo 3

B. 3
   12 Hallo
   Hallo Tschüss
   Hallo 3

C. 3
   3 Hallo
   Hallo Tschüss
   Hallo12

D. 3
   12Hallo
   Hallo Tschüss
   Halo12

SMART INDUSTRY CAMPUS

Was ist die Ausgabe?

- bei Concatenation muss man auf folgendes achten:
  - Numerische Operanden werden addiert, solange vor ihnen kein String steht

```
String s = "Ciao";
int i = 10;
System.out.println(1 + 2 + i + s);
```

A. 1 2 10 Ciao

B. 3 10 Ciao

C. 13 Ciao

D. 12 10 Ciao

SMART INDUSTRY CAMPUS

Methoden

- length():
  - Gibt die Länge der Characters zurück, die im String enthalten sind

```
String str1 = "Hallo";
System.out.println(str1.length());
```

str1.length() ergibt 5

SMART INDUSTRY CAMPUS

Methoden

- charAt(int index)
  - gibt den Character am spezifischen index aus
  - Index startet bei 0!

```
String str1 = "Hallo";
System.out.println(str1.charAt(4));
```

Ergibt o

```
String str1 = "Hallo";
System.out.println(str1.charAt(5));
```

Exception! StringIndexOutOfBoundsException

Methoden

- indexOf(String str)
  indexOf(int ch)

```
String str1 = "Hallo";
System.out.println(str1.indexOf("l"));
```

Ergibt 2

Leerzeichen werden mit gerechnet

- indexOf(String str, int fromIndex)
  indexOf(int ch, int fromIndex)

```
String str1 = "Hallo ma Frenda";
System.out.println(str1.indexOf(„a", 8));
```

Ergibt 14

Beim Nichtfinden gibt die Methode -1 zurück

SMART INDUSTRY CAMPUS

Methoden

- substring(int beginIndex)
  - Sucht nach Characters in einem String
  - Gibt einen Teil des Strings zurück
  - Index fängt bei 0 an!

```
String str1 = "Hallo";
System.out.println(str1.substring(3));
```

Ergibt lo

- substring(int beginIndex, int endIndex)
  - endIndex als exklusiv betrachten!

```
String str1 = "Hallo ma Frenda";
System.out.println(str1.substring(2, 10));
```

Ergibt llo ma F

SMART INDUSTRY CAMPUS

Methoden

- toLowerCase() and toUpperCase():
  - toLowerCase() : alle Characters des Strings werden klein geschrieben
  - toUpperCase() : alle Characters des Strings werden groß geschrieben

```
String str1 = "Hello ma Frenda";
System.out.println(str1.toLowerCase());       //hello ma frenda
System.out.println(str1.toUpperCase());       //HELLO MA FRENDA
```

SMART INDUSTRY CAMPUS

Methoden

- equals():
  - Gibt einen boolean zurück
  - Vergleicht die Inhalte von Strings
  - Case-sensitive: achtet auf die Groß-/Kleinschreibung des Strings

```
System.out.println("abc".equals("abc"));                    //true
System.out.println("abc".equals("ABC"));                    //false
System.out.println("abc".equalsIgnoreCase("ABC"));          //true
```

- equalsIgnoreCase():
  - Gibt einen boolean zurück
  - Vergleicht ebenfalls die Inhalte von Strings
  - Nicht case-sensitive: achtet NICHT auf die Groß-/Kleinschreibung des Strings

SMART INDUSTRY CAMPUS

Methoden

- startsWith(String prefix):
  - Gibt einen boolean zurück, wenn ein String mit dem übergebenen String anfängt
  - Case-sensitive

```
System.out.println("abc".startsWith("a"));          //true
System.out.println("abc".startsWith("A"));          //false
System.out.println("abc".endsWith("a"));            //false
System.out.println("abc".endsWith("c"));            //true
```

- endsWith(String suffix)
  - Gibt einen boolean zurück, wenn ein String mit dem übergebenen String endet
  - Case-sensitive

SMART INDUSTRY CAMPUS

Methoden

- contains(String str):
  - Überprüft, ob der übergebene String im ursprünglichen String enthalten
  - Case-sensitive

```
System.out.println("abc".contains("a"));          //true
System.out.println("abc".contains("A"));          //false
```

SMART INDUSTRY CAMPUS

Methoden

- replace(char oldChar, char newChar)
  replace(CharSequence oldChar, CharSequence newChar):
  - Sucht nach dem alten Character oder alten CharacaterSequence und ersetzt ihn mit einem neuen

```
System.out.println("ababcc".replace("a", "A"));          //AbAbcc
```

Methoden

- trim():
  - Löscht die Leerzeichen vor und nach einem String

```
System.out.println(„    HALLO    mein name ist Anita     ".trim());   //HALLO   mein name ist Anita
```

SMART INDUSTRY CAMPUS

Immutable

- sobald ein String-Objekt kreiert wurde, kann es nicht verändert werden
- Was heißt das?

```
public class ImmutableStringExample {
    public static void main(String[] args) {
        String original = "Hallo";
        String modified = original.concat(" Welt!");

        System.out.println("Modifiziert: " + modified);
        //Ausgabe: Hallo Welt!
    }
}
```

```
public class ImmutableStringExample {
    public static void main(String[] args) {
        String original = "Hallo";
        original.concat("Ola");

        System.out.println("Original: " + original);
        //Ausgabe : Hallo
    }
}
```

SMART INDUSTRY CAMPUS

Immutable

- sobald ein String-Objekt kreiert wurde, kann es nicht verändert werden
- man muss demnach einen neuen String erstellen
  - Bzw. den Pointer auf ein neues String-Objekt verweisen lassen
- die Anwendung von Methoden auf einen String resultieren in einer Erzeugung eines String-Objekts auf dem Heap

```
public class ImmutableStringExample {
    public static void main(String[] args) {
        String original = "Hallo";
        original.concat("Ola");

        System.out.println("Original: " + original);
        //Ausgabe : Hallo
    }
}
```

```
public class ImmutableStringExample {
    public static void main(String[] args) {
        String original = "Hallo";
        String modified = original.concat(" Welt!");

        System.out.println("Modifiziert: " + modified);
        //Ausgabe: Hallo Welt!
    }
}
```

SMART INDUSTRY CAMPUS

Method Chaining

- die Methoden müssen nicht einzeln aufgerufen werden
- die Methoden können verkettet werden

```
public class MethodChainingExample {
    public static void main(String[] args) {
        String result = "   Hallo, Java!   "
                .trim()
                .replace("Java", "Welt")
                .toUpperCase()
                .substring(0, 10);

        System.out.println(result); // Gibt "HALLO, WEL" aus
    }
}
```

SMART INDUSTRY CAMPUS

Aufgabe ca. 25-30 Minuten

Schreibe ein Java-Programm, das eine Benutzer-Eingabe verarbeitet und verschiedene String-Methoden darauf anwendet. Dein Programm soll:

1. Eingabe einlesen:
   - Frage den Benutzer nach einem beliebigen Satz.
   - Frage den Benutzer nach einem Wort, das im Satz vorkommen könnte.

2. Analysen und Operationen durchführen:
   - Prüfe, ob der Satz mit „Hallo“ beginnt oder mit „!“ endet.
   - Prüfe, ob der Satz das gesuchte Wort enthält (case-insensitive).
   - Ersetze alle Leerzeichen im Satz durch Unterstriche _ und gib den veränderten Satz aus.
   - Ermittle die Länge des Satzes und gib sie aus.
   - Falls das gesuchte Wort enthalten ist, gib die Position des ersten Auftretens aus. Falls nicht, gib eine entsprechende Meldung aus.
   - Schneide den Satz so zu, dass nur die ersten 10 Zeichen übrig bleiben, und gib das Ergebnis aus.
   - Entferne mögliche Leerzeichen am Anfang und Ende des Satzes und gib das bereinigte Ergebnis aus.
   - Gib das erste Zeichen des Satzes aus.

SMART INDUSTRY CAMPUS

StringBuilder

BUCHSEITEN S.111-117

SMART INDUSTRY CAMPUS

Wie viele Objekte resultieren von diesem Codeabschnitt? Wie viele landen direkt im Garbage Collector?

```
String s = "";
for (char current = 'a'; current <= 'z'; current++){
    s += current;
}
System.out.println(s);
```

A. Maximal 1
B. Maximal 20
C. Maximal 25
D. Maximal 27

SMART INDUSTRY CAMPUS

Grundlagen

- vorheriges Beispiel zeigt, dass Strings ineffizient werden können!
  - Jede Änderung erzeugt ein neues Objekt im Speicher, da String immutable (=unveränderbar) sind
  - Viele Änderungen führen zu Leistungsproblemen
- Lösung: StringBuilder
  - Ist **mutable** (=änderbar)
  - Änderungen werden **am selben Objekt** vorgenommen
  - Spart Speicher und verbessert die Performance

SMART INDUSTRY CAMPUS

Initialisierung

- Ein StringBuilder-Objekt mit einer initialen Zeichenkette „Hallo“

```
StringBuilder b = new StringBuilder("Hallo");
```

- Erstellt einen leeren StringBuilder
- Standardkapazität liegt bei 16, wird automatisch erweitert

```
StringBuilder c = new StringBuilder();
```

- Standardkapazität wird auf 5 gesetzt, wird automatisch erweitert

```
StringBuilder d = new StringBuilder(5);
```

SMART INDUSTRY CAMPUS

Methoden

- append(String s):
  - Erweitert das StringBuilder-Objekt um einen String s

```
public static void main(String[] args) {
    StringBuilder b = new StringBuilder("Hallo");
    b.append(" mein Name ist Anita");
    System.out.println(b);

    //Ausgabe: Hallo mein Name ist Anita
}
```

SMART INDUSTRY CAMPUS

Methoden

- length()
  - Gibt die Länge der Characters im StringBuilder zurück

```
public static void main(String[] args) {
    StringBuilder b = new StringBuilder("Hallo");
    b.append(" mein Name ist Anita");
    System.out.println(b.length());

    //Ausgabe: 25
}
```

SMART INDUSTRY CAMPUS

Methoden

- insert(int index, String str)
  - Fügt an die Index-Position den String str ein

```
public static void main(String[] args) {
    StringBuilder b = new StringBuilder("Hallo");
    b.append(" mein Name ist Anita");
    System.out.println(b.insert(19, „ immernoch"));

    //Ausgabe: Hallo mein Name ist immernoch Anita
}
```

SMART INDUSTRY CAMPUS

Methoden

- delete(int start, int end)
  - Löscht von start bis end-1 die Character
  - Wenn end-index nicht existiert, wird es trotzdem ausgeführt

```
public static void main(String[] args) {
   StringBuilder b = new StringBuilder("Hallo");
   b.append(" mein Name ist Anita");
   System.out.println(b.delete(20, 25));

   //Ausgabe: Hallo mein Name ist
}
```

- deleteCharAt(int index)
  - Löscht den Character an Position des indexes

```
public static void main(String[] args) {
   StringBuilder b = new StringBuilder("Hallo");
   b.append(" mein Name ist Anita");
   System.out.println(b.deleteCharAt(20));

   //Ausgabe: Hallo mein Name ist nita
}
```

SMART INDUSTRY CAMPUS

Methoden

- reverse():
  - Gibt die Character im StringBuilder von hinten nach vorne aus

```
public static void main(String[] args) {
    StringBuilder b = new StringBuilder("Hallo");
    System.out.println(b.reverse());

    //Ausgabe: ollaH
}
```

SMART INDUSTRY CAMPUS

Methoden

- toString():
  - Konvertiert den StringBuilder in einen String

```
public static void main(String[] args) {
    StringBuilder b = new StringBuilder("Hallo");
    System.out.println(b.reverse().toString());

    //Ausgabe: ollaH
}
```

SMART INDUSTRY CAMPUS

Mutability

– es wird an einem Objekt gearbeitet

```
public static void main(String[] args) {
    StringBuilder b = new StringBuilder("Hallo");
    b.append(" mein Name ist");
    StringBuilder c = b.append(" Anita");
    System.out.println(c);

    //Ausgabe: Hallo mein Name ist Anita
}
```

SMART INDUSTRY CAMPUS

Method Chaining

– auch hier ist Methodenverkettung möglich

```
StringBuilder b = new StringBuilder("Hallo");
StringBuilder c = b.append(" mein Name ist Anita").append(" !").reverse().deleteCharAt(10);
```

SMART INDUSTRY CAMPUS

Was ist die Ausgabe?

```
StringBuilder b = new StringBuilder("Hallo");
StringBuilder c = b.append(" mein Name ist");
b = b.append(" Anita").append(" !");
System.out.println(b);
System.out.println(c);
```

A. Hallo mein Name ist Anita !
   Hallo mein Name ist Anita !

B. Hallo Anita !
   Hallo mein Name ist

C. Hallo mein Name ist Anita !
   Hallo mein Name ist

SMART INDUSTRY CAMPUS

Aufgabe ca. 30 min

Du hast einen String und musst diesen so umwandeln, dass er in ein bestimmtes Zielformat passt. Du sollst dabei die Methoden der Klasse StringBuilder verwenden, um den String zu manipulieren.

String input = "ssapsthcamdnullotttsinereimmargorp";

1. Dreht den String um. Was steht da?

2. Formattiert den String so um, dass folgender String rauskommt:

Output : Programmieren ist toll und macht Spaß!

Achtet auf die Indexe! Wenn ihr etwas hinzufügt/wegnehmt, passen sich die Indexe mit an!

SMART INDUSTRY CAMPUS

String vs. StringBuilder

Merkmal	String	StringBuilder
Mutabilität	Unveränderlich (immutable)	Veränderlich (mutable)
Speicherverbrauch	Höher	Niedriger
Leistung bei Änderungen	Langsam	Schnell
Verwendung	Konstanten, unveränderliche Daten	Häufige Änderungen

SMART INDUSTRY CAMPUS

Equality

BUCHSEITEN S.117-118

== im Kontext von StringBuildern

SMART INDUSTRY CAMPUS

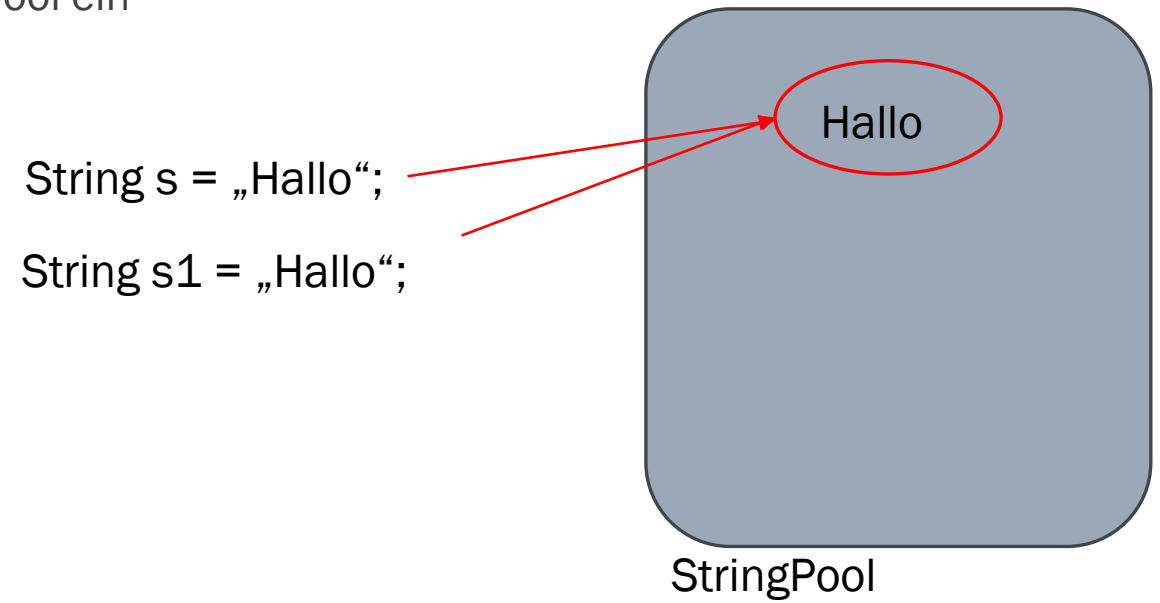
```
StringBuilder one = new StringBuilder();
StringBuilder thesecondone = new StringBuilder();
StringBuilder three = one.append("Hallo");
System.out.println(one == thesecondone);          //false
System.out.println(one == three);                 /true
```

== im Kontext von Strings

```
String s = "Hallo";
String s2 = "Hallo";
System.out.println(s == s2);              //true
```

StringPool

- ist eine Speicherstelle, in welcher Java String-Literale abspeichert
- bevor ein String-Literal in den StringPool landet, checkt Java, ob es nicht bereits existiert
  - Wenn es existiert, wird die Referenz auf das Bestehende Literal zurückgegeben
  - Wenn nicht, fügt Java das Literal in den StringPool ein



== im Kontext von Strings

```
String s = "Hallo";
String s2 = "Hallo  ".trim();
System.out.println(s == s2);              //false
```

SMART INDUSTRY CAMPUS

Immutable

- sobald ein String-Objekt kreiert wurde, kann es nicht verändert werden
- man muss demnach einen neuen String erstellen
  - Bzw. den Pointer auf ein neues String-Objekt verweisen lassen
- die Anwendung von Methoden auf einen String resultieren in einer Erzeugung eines String-Objekts auf dem Heap

```java
public class ImmutableStringExample {
    public static void main(String[] args) {
        String original = "Hallo";
        original.concat("Ola");

        System.out.println("Original: " + original);
        //Ausgabe : Hallo
    }
}
```

```java
public class ImmutableStringExample {
    public static void main(String[] args) {
        String original = "Hallo";
        String modified = original.concat(" Welt!");

        System.out.println("Modifiziert: " + modified);
        //Ausgabe: Hallo Welt!
    }
}
```

== im Kontext von Strings

```
String s = new String("Hallo");
String s2 = "Hallo";
System.out.println(s == s2);              //false
```

SMART INDUSTRY CAMPUS

.equals()

– vergleicht IMMER nur den Inhalt der Strings

```
String s = new String("Hallo");
String s2 = "Hallo";
System.out.println(s.equals(s2));                    //true
```

– StringBuilder hat equals() nicht überschrieben -> vergleicht also immer noch Referenzwerte (==)

SMART INDUSTRY CAMPUS

== vs. .equals()

Vergleich	String	StringBuilder
==	Vergleicht Referenzen (Speicheradressen)	Vergleicht Referenzen (Speicheradressen)
.equals()	Vergleicht den Inhalt der Strings	Hat keine .equals()-Methode überschrieben

SMART INDUSTRY CAMPUS

Aufgabe (ca. 5 min)

String str1 = "Hallo Welt";

String str2 = "Hallo Welt";

String str3 = new String("Hallo Welt");

Vergleiche die folgenden String-Objekte mit == und .equals() und gib jeweils das Ergebnis (true/false) aus:

str1 == str2
str1.equals(str2)
str1 == str3
str1.equals(str3)

Überprüfe dein Ergebnis in der IDE.

SMART INDUSTRY CAMPUS

Aufgabe (ca. 5 min)

StringBuilder sb1 = new StringBuilder("Hallo Welt");

StringBuilder sb2 = new StringBuilder("Hallo Welt");

Vergleiche die folgenden StringBuilder-Objekte mit == und .equals() und gib das Ergebnis (true/false) aus:

sb1 == sb2
sb1.equals(sb2)
sb1.toString().equals(sb2.toString())

Überprüfe dein Ergebnis in der IDE.