

HO GENT

OOSDII

Strings en reguliere expressies

Table of Contents

1. Doelstellingen	1
2. Inleiding	1
3. Fundamenten van karakters en Strings	1
4. Klasse String	2
4.1. String constructoren	2
4.2. String methodes	2
4.3. Strings vergelijken	3
4.4. Karakters en substrings uit Strings	4
4.5. Concateneren van Strings	4
4.6. Andere String-methodes	4
4.7. String methode valueOf	5
5. Klasse StringBuilder	5
5.1. StringBuilder constructors	5
5.2. StringBuilder methoden	6
5.3. Methodes voor karakterbewerkingen	6
5.4. Append methodes	6
5.5. Tussenvoeg- en verwijdermethodes	7
6. Klasse Character	7
7. Tokenizing Strings	7
8. Reguliere expressies, Class Pattern en Class Matcher	8
8.1. Stringmethodes met reguliere expressies	10
8.2. Klasse Pattern en klasse Matcher	10
9. Bijkomend leermateriaal	11

1. Doelstellingen

- Kan strings verwerken in Java

2. Inleiding

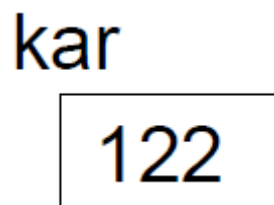
- In dit hoofdstuk bespreken we de mogelijkheden van de klassen :
 - String
 - StringBuilder
 - Character

Deze klassen vormen de basis voor string- en karakterbewerking in Java.

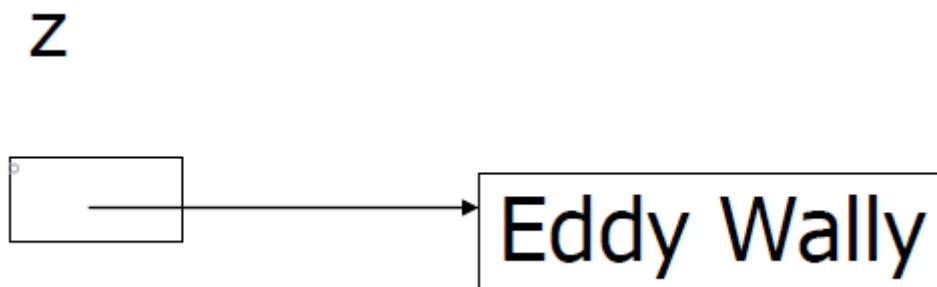
- En reguliere expressies
 - Met reguliere expressies kan je invoer valideren

3. Fundamenten van karakters en Strings

- Een karakter wordt intern opgeslagen door zijn overeenkomstige Unicode (zie Deitel - appendix B). Deze Unicode is een integerwaarde .
- Voorbeeld: `char kar = 'z';`
Intern:



- Een string is een reeks van karakters die als één geheel wordt beschouwd. Kan bestaan uit letters, cijfers en speciale karakters.
- Een string is een object van de klasse String.
- Voorbeeld : `String z = "Eddy Wally";`
Intern:

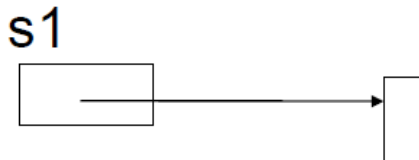


4. Klasse String

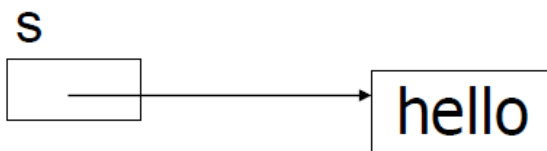
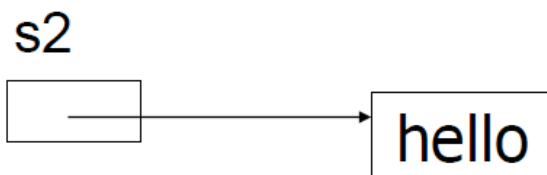
- De klasse String laat toe om Strings te creëren en te bewerken.
- Zie volledige klasse: <https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/String.html>
- Klasse bevat o.a.
 - meerdere constructoren
 - methodes om Strings te vergelijken, om karakters en substrings in Strings te localiseren / extraheren, om Strings te concateneren, ...

4.1. String constructoren

- `s1 = new String();` → lege string ("") met lengte 0



- `s2 = new String(s);`




4.2. String methodes

- De methode **length()** geeft het aantal karakters van een String weer

`s1.length()` ⇒ methode
NIET `s1.length` zoals bij arrays ⇒ attribuut

- Het eerste karakter in een String begint steeds bij positie 0

h a m b u r g e r
0 1 2 3 4 5 6 7 8


- De methode **charAt(pos)** geeft het karakter op de positie pos terug
- De methode **getChars(start, laatste, naar, vanafpos)** kopieert de karakters van een bepaalde String vanaf positie start t.e.m. laatste-1, in de array naar vanaf index vanafpos

4.3. Strings vergelijken

- Java voorziet een aantal methodes om String objecten met elkaar te vergelijken.
- Wanneer de computer twee strings met elkaar vergelijkt dan worden de interne numerieke codes van alle karakters in de strings met elkaar vergeleken.
 - String s1 is groter dan String s2
⇒ s1 komt alfabetisch na s2
- **equals()** test of de inhoud van twee strings identiek zijn (methode overgeërfd van de klasse Object)
 - Geeft true of false terug al naargelang alle karakters uit de String objecten gelijk zijn of niet (de Unicode waarden worden vergeleken).
- De **operator ==** test of twee referentie variabelen naar hetzelfde object in het geheugen wijzen.
 - De adressen in de referentie variabelen naar de Strings worden vergeleken, niet de inhoud van de Strings!
- Voorbeeld:
s1="hello"; ⇒ s1=="hello"?
s1=new String("hello"); ⇒ s1=="hello"?
- **equalsIgnoreCase()** test of twee String objecten identiek zijn, maar houdt geen rekening met hoofd en kleine letters
 - Geeft true of false terug
- **compareTo()**
 - geeft 0 terug als de Strings dezelfde zijn of een negatief/positief getal terug als ze verschillend zijn:
s1.compareTo(s2) = 0 als s1 en s2 gelijk zijn
> 0 als s1 > s2
< 0 als s1 < s2
- **regionMatches()** vergelijkt delen van twee Stringobjecten
Voorbeeld: s1.regionMatches(0, s4, 0, 5)
 - vergelijk de vijf eerste karakters uit s1 met die van s4
0 = start in s1

0 = start in s4
5 = aantal karakters

- Geeft true of false terug
- **startsWith(string)**
 - Geeft true of false terug, al naargelang het String object begint met string.
- **endsWith(string)**
 - Geeft true of false terug, al naargelang het String object eindigt op string.

4.4. Karakters en substrings uit Strings

- De methode **indexOf** zoekt het eerste voorkomen van een karakter in een String.
 - Voorbeeld: `letters.indexOf('c')`
 - geeft de index van het karakter c in de String letters weer, of
 - geeft -1 terug als het karakter niet in de String voorkomt
 - Voorbeeld: `letters.indexOf('c',7)`: idem, maar nu wordt pas gestart vanaf index 7 in letters
 - De methode `indexOf` kan ook gebruikt worden om een substring in een String te zoeken
- De methode **lastIndexOf** is analoog aan `indexOf`, alleen wordt nu het laatste voorkomen van een karakter weergegeven. (Zoekt van achter naar voor in de String)
- De methode **substring** maakt een nieuwe String uit een bestaande String.
 - Voorbeeld: `letters.substring(20)`
geeft een nieuwe String weer die een substring is van letters, beginnend vanaf index 20.
 - Voorbeeld: `letters.substring(start, last)`
geeft substring uit letters terug, kopie van karakters van index start t.e.m. index last 1
- De methode `substring` kan een `StringOutOfBoundsException` gooien

4.5. Concateneren van Strings

- De methode **concat** geeft een nieuwe String terug, die een concatenatie is van twee Strings.
 - Voorbeeld: `s1.concat(s2)`
geeft een nieuwe String die een samenvoeging is van s1 en s2.
s1 en s2 worden niet gewijzigd bij deze operatie

4.6. Andere String-methodes

- De methode **replace** geeft een nieuwe String terug, die een kopie is van de originele, maar waarin een bepaald karakter vervangen is door een ander karakter.
- De methode **toUpperCase** geeft een nieuwe String terug, die een kopie is van de originele, maar waarin alle karakters in hoofdletters staan.
- De methode **toLowerCase** geeft een nieuwe String terug, die een kopie is van de originele, maar waarin alle karakters in kleine letters staan.

- De methode **trim** geeft een nieuwe String terug, die een kopie is van de originele , maar waarin alle spaties vooraan en achteraan zijn verwijderd.
- De methode **toCharArray** kopieert alle karakters uit een String naar een array bestaande uit chars.

4.7. String methode valueOf

- De klasse String voorziet een aantal static klasse methodes, die argumenten van verschillende types kunnen converteren naar Strings.
 - De methode **valueOf** geeft een nieuwe String terug, die de String representatie voorstelt van een primitief datatype of ander object.

5. Klasse StringBuilder

- Nadat een String object gecreëerd is, kan de inhoud ervan nooit meer veranderen
- StringBuilder wordt gebruikt voor zogenaamde "dynamische strings" (dit is een aanpasbare versie van String)
- De capaciteit is het aantal karakters dat een StringBuilder kan bevatten
 - Als de capaciteit overtroffen wordt, dan breidt ze zich automatisch uit, om de bijkomende karakters te kunnen opvangen.
- Zie volledige klasse: <https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/StringBuilder.html>
- Gebruik + en += voor STRING concatenatie
 - ⇒ De klasse StringBuilder wordt gebruikt om deze operatoren te implementeren.

```
String string1 = "hello";
String string2 = "BC";
int value = 22;
String s = string1 + string2 + value;
⇒ new StringBuilder().append("hello").append("BC").append(22).toString();
```

5.1. StringBuilder constructors

- Er zijn meerdere constructoren in de klasse StringBuilder, enkele voorbeelden :
 - `buffer1 = new StringBuilder()`
Creëert een lege buffer met capaciteit van 16 karakters
 - `buffer2 = new StringBuilder(lengte)`
Creëert een lege buffer met capaciteit het aantal karakters dat door de integer lengte wordt aangegeven
 - `buffer 3 = new StringBuilder("tekst")`
Creëert een buffer met inhoud "tekst" en met capaciteit het aantal karakters in de string "tekst" + 16 dus in het vb.: 5 + 16 = 21

5.2. StringBuilder methoden

- Methode **length**
 - Geeft het aantal karakters in de StringBuilder terug
- Methode **capacity**
 - Geeft de capaciteit van de StringBuilder terug
 - capaciteit = aantal karakters dat kan opgeslagen worden zonder meer geheugenruimte te moeten alloceren
- Methode **setLength**
 - Verhoogt of verlaagt de lengte van de StringBuilder
- Methode **ensureCapacity**
 - Stelt de capaciteit van de StringBuilder in
 - Garandeert dat de StringBuilder een minimumcapaciteit heeft
 - Let op: als de originele capaciteit kleiner is dan de nieuwe, dan wordt de capaciteit ofwel het getal dat aangegeven wordt in het argument ofwel $2 * \text{de originele capaciteit} + 2$, naargelang wat groter is

5.3. Methodes voor karakterbewerkingen

- Methode **charAt**
 - Geeft het karakter uit de StringBuilder terug dat zich op de gespecificeerde index bevindt
 - Indien de index buiten de grenzen van de StringBuilder valt , dan krijg je een `StringIndexOutOfBoundsException`
- Methode **setCharAt**
 - Vult het opgegeven karakter in de StringBuilder in op de gespecificeerde index
 - Zie `charAt` voor indexwaarde die buiten de grenzen valt
- Methode **getChars**
 - Geeft een array van karakters terug die overeenkomt met de inhoud van de StringBuilder
 - 4 argumenten : startindex , index 1 positie voorbij laatste te kopiëren karakter , de array waarnaar moet gekopieerd worden en de beginpositie in de array
- Methode **reverse**
 - Keert de inhoud van de StringBuilder om

5.4. Append methodes

- Meerdere overloaded **append** methodes om waarden van verschillende datatypes aan het einde van een StringBuilder te kunnen plakken
 - Een versie voor elk van de primitieve datatypes plus een voor karakterarrays, een voor Strings en een voor Objects

5.5. Tussenvoeg- en verwijdermethodes

- Methode **insert**
 - meerdere overloaded methodes om de verschillende datatypes te kunnen tussenvoegen op een gegeven positie in een StringBuilder
 - Twee argumenten: index en het in te voegen gedeelte
 - StringIndexOutOfBoundsException bij verkeerde indexwaarde
- Methode **delete**
 - Wist een reeks karakters
 - Twee argumenten: startpositie en indexwaarde één positie voorbij het einde van de te wissen karakters
 - StringIndexOutOfBoundsException bij verkeerde indexwaarde
- Methode **deleteCharAt**
 - Wist één karakter
 - Eén argument: positie van het te wissen karakter
 - StringIndexOutOfBoundsException bij verkeerde indexwaarde

6. Klasse Character

- Primitieve variabelen als objecten behandelen
 - Klassen Boolean, Character, Double, Float, Byte, Short, Integer en Long
 - Behalve Boolean en Character worden deze klassen afgeleid van de klasse Number
 - Deze 8 klassen worden "**type wrappers**" genoemd en maken deel uit van java.lang
- Klasse Character: type wrapper voor karakters
 - Meeste methodes zijn static en testen of manipuleren een karakter
 - Constructor die aan de hand van een char argument een Character object maakt
 - Zie voorbeelden en Java API <https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/Character.html> documentatie voor meer informatie

7. Tokenizing Strings

- Token
 - Zie: Lezen van een zin
 - We delen de zin op in woorden en leestekens
 - Elk onderdeel (=token) heeft een betekenis voor ons
 - Compiler doet ook aan "tokenizing":
 - deelt statement op in keywords, identifiers, operators en andere elementen van de

programmeertaal

- 2 manieren:
 - Methode **split** van de klasse `String` verdeelt de zin in tokens en geeft een array van `Strings` terug.
 - Tokens worden gescheiden door **delimiters**
 - Dit zijn typisch whitespacekarakters zoals spatie, tab, newline, carriage return of een ander teken.
 - Gebruik van de klasse **StringTokenizer**
 - Deze manier van werken laat toe bij het teruggeven van de tokens **ook** hun scheidingsteken mee te geven.
- Voorbeelden:
 - Met `split`:

```
Scanner scanner = new Scanner(System.in);
System.out.println("Geef een zin en Enter");
String zin = scanner.nextLine();

String[] tokens = zin.split(" ");

System.out.printf("Aantal elementen: %d\nTokens:%n", tokens.length);
for(String token : tokens)
    System.out.println(token);
```

- Met `StringTokenizer`:

```
Scanner scanner = new Scanner(System.in);
System.out.println("Geef een zin en Enter");
String zin = scanner.nextLine();

StringTokenizer tokens = new StringTokenizer(zin);

System.out.printf("Aantal elementen: %d\nTokens:%n", tokens.countTokens());
while (tokens.hasMoreTokens())
    System.out.println(tokens.nextToken());
```

8. Reguliere expressies, Class Pattern en Class Matcher

- Een reguliere expressie is een `String`, een reeks van karakters en symbolen die een zoek “pattern” voorstelt om karakters te matchen in andere `Strings`.
- Dit is handig om:

- invoer te valideren
- na te gaan of data in een bepaald formaat staan
- de syntax van een programma te valideren
- Gebruik: methode **matches** van de klasse String:
 - De parameter is een reguliere expressie, waarmee de inhoud van een String object wordt vergeleken
 - De teruggeefwaarde is een boolean, die aangeeft of de "match" is gelukt
- Voorbeelden:
 - `if (postcode.matches("\\d{4}")) ...`
 - `if (userId.matches("[a-z0-9]{5,10}")) ...`

Reguliere expressie syntax

Expressie	matches
<code>\d</code>	elk cijfer
<code>\w</code>	elke letter, cijfer of underscore
<code>\.</code>	elk karakter, maar geen newline
<code>\D</code>	elk niet-cijfer
<code>\W</code>	elke niet-letter, niet-cijfer en geen underscore
<code>\S</code>	elke niet-witruimte
<code>[...]</code>	opsomming, <code>[abc]</code> \Rightarrow a of b of c
<code>[^...]</code>	\wedge negatie, <code>[^abc]</code> \Rightarrow alles behalve a, b en c
<code>[A-Z]</code>	- van tot en met \Rightarrow van A t.e.m. Z
<code>re*</code>	0 of meer
<code>re+</code>	1 of meer
<code>re?</code>	0 of 1
<code>re{n}</code>	precies n voorkomens
<code>re{n,}</code>	ten minste n voorkomens
<code>re{n,m}</code>	tussen n en m voorkomens
<code>a b</code>	of \Rightarrow a of b
<code>(re)</code>	groeperen van reguliere expressies
<code>^</code>	begin van de lijn
<code>\$</code>	einde van de lijn

8.1. Stringmethodes met reguliere expressies

- **replaceAll:**
 - vervangt in de String alle voorkomens van een bepaald stukje tekst door een nieuw stukje tekst
- **replaceFirst :**
 - vervangt alleen het eerste voorkomen van het stukje tekst door een nieuw stukje tekst
- **split:**
 - verdeelt een String in verscheidene substrings

8.2. Klasse Pattern en klasse Matcher

- Klasse **Pattern** stelt een reguliere expressie voor
- Klasse **Matcher** bevat
 - een reguliere expressie pattern
 - een **CharSequence** waarin gezocht wordt naar een pattern
- CharSequence is een interface ⇒ methodes charAt , length , subSequence en toString moeten worden gedeclareerd . De klassen String, StringBuilder , ... implementeren deze interface.
- Pattern klasse
 - Eénmaal reguliere expressie gebruiken ⇒ static Pattern methode **matches**
 - Meerdere keren reguliere expressie gebruiken ⇒ static Pattern methode **compile**
- Matcher klasse
 - matches, idem matches uit Pattern, maar ontvangt geen argumenten
 - find, lookingAt, replaceFirst en replaceAll
 - Voorbeeld :

```
public static void main(String args[]) {  
    String REGEX = "a*b";  
    String INPUT = "aabfooaabfoaabfoob";  
  
    Pattern p = Pattern.compile(REGEX);  
    Matcher m = p.matcher(INPUT); // get a matcher object  
    int count = 0;  
  
    while (m.find()) {  
        System.out.println("Match " + ++count);  
        System.out.println(m.group());  
    }  
}
```

Geeft als uitvoer:

```
Match 1  
aab  
Match 2  
aab  
Match 3  
ab  
Match 4  
b
```

9. Bijkomend leermateriaal

- <http://docs.oracle.com/javase/tutorial/java/data/strings.html>
- <http://docs.oracle.com/javase/tutorial/java/data/characters.html>
- <https://docs.oracle.com/javase/tutorial/essential/regex/index.html>