

Umgang mit Zeichenketten

((itanius informatik))

Java Foundation Track by Carsten Bokeloh

Zeichenketten

- Die Klasse Character
- String und seine Methoden
- StringBuffer & StringBuilder

Strings

- ❖ Einzelne Zeichen: Die Character-Klasse
- ❖ Ansammlung von Zeichen: Strings in Java
- ❖ Stringklasse String
 - ❖ Konstruktoren
 - ❖ Methoden für den Umgang mit Strings
- ❖ Stringklassen StringBuffer und StringBuilder
 - ❖ Konstruktoren
 - ❖ spezielle Methoden für StringBuffer
- ❖ Ein-undAusgabe von Strings
- ❖ Ausblick, weitere Klassen

Die Character Klasse

- ❖ Klasse Character ist im Kernpaket java.lang enthalten
- ❖ Zeichen werden im Unicode-Format abgelegt
- ❖ Jedes Zeichen benötigt 16Bit(2Byte)!
- ❖ Sonderzeichen können direkt dargestellt werden
- ❖ Die Charakter-Klasse enthält statische Methoden zum Behandeln einzelner Zeichen
- ❖ Alle zeichenuntersuchenden Methoden beginnen mit is

Die Character Klasse

- ❖ Handelt es sich um
 - ❖ eine Ziffer zwischen 0 und 9?
 - ❖ einen Buchstaben?
 - ❖ einen Groß- oder Kleinbuchstaben?
 - ❖ Ein Leerzeichen, Zeilenvorschub, Tabulator oder Return?
 - ❖ `toUpperCase(char value)` wandelt einen kleinen in einen großen Buchstaben
 - ❖ `toLowerCase(char value)` wandelt einen großen in einen kleinen Buchstaben

Strings

- ❖ Ein String ist eine Sammlung von Zeichen
- ❖ Zeichen im Unicode-Format(16Bit pro Zeichen)
- ❖ über Umwege auch Unicode4(32Bit pro Zeichen)
- ❖ Methoden zum einfachen Umgang mit Strings sind in drei Klassen enthalten:
 - ❖ String
 - ❖ StringBuffer, StringBuilder
- ❖ Alle Klassen setzen die objektorientierte Idee um!

String

- ❖ Stringobjekte müssen nicht immer mit `new` angelegt werden.
- ❖ Beispiel: `"Hallo,String!".length()` liefert den Wert 14 (Leer- und Sonderzeichen zählen mit!)
- ❖ Anführungszeichen deuten das Stringobjekt an
- ❖ Java erzeugt automatisch für jedes Zeichenketten-Literal ein Stringobjekt
- ❖ konstante Zeichenketten werden nur einmal im sog. Konstantenpool (Speicherbereich) abgelegt, unabhängig vom Vorkommen im Programm

Stringklasse String

- ❖ Nicht änderbare Zeichenketten(immutable)
- ❖ fast alle Stringoperationen sind möglich
- ❖ die Länge steht fest und kann nicht verändert werden
- ❖ der Inhalt steht fest und kann nicht verändert werden
- ❖ scheinbare Veränderungen erzeugen ein neues Stringobjekt
 - ❖ Beispiel: 2 Strings zusammenfügen→ein neues Stringobjekt wird erzeugt

Konstruktoren

- ❖ Es gibt drei Konstruktoren zum Anlegen eines
 - ❖ Stringobjektes mit new:
 - ❖ `String()` erzeugt ein leeres Stringobjekt
 - ❖ `String(String value)` erzeugt ein Stringobjekt mit dem Inhalt eines anzugebenden Strings
 - ❖ `String(char[] value)` erzeugt ein neues Stringobjekt aus einem Charakter-Array
- ❖ Im Folgenden werden die wichtigsten Methoden der Stringklasse vorgestellt.

Methoden

- ❖ *length()* liefert die Länge eines Strings als Integer
- ❖ Um einen leeren String zu erkennen, kann die Methode *isEmpty()* verwendet werden.

```
"".isEmpty(); // true
```

- ❖ Alternativ kann ein leerer String auch durch eine Abfrage mit *length()* untersucht werden:

```
"".length() == 0; // true
```


Methoden

- ❖ Strings sind Referenzvariablen
- ❖ spezielle Methoden zum Vergleichen von Strings nötig
- ❖ Beispiel:
 - ❖ `String s1 = new String("Hallo");`
 - ❖ `String s2 = new String("Hallo");`
 - ❖ `s1 == s2; // liefert false!!!`
- ❖ Grund: s1 und s2 werden an verschiedenen Speicherstellen neu angelegt. Die Adressen werden verglichen!

Suche

- ❖ Suchen eines Zeichens in einem String:
indexOf(char value, int start)
- ❖ Beginnt mit der Suche am Stringanfang
- ❖ Gibt das erste Vorkommen des Zeichens zurück
- ❖ unterscheidet Groß- und Kleinschreibung!
- ❖ Rückgabe von -1, wenn das Zeichen nicht enthalten ist

Suche

- ❖ Suchen eines Teilstrings in einem String:

contains(String value)

- ❖ Kann nur dann nach einzelnen Zeichen suchen, wenn value nur ein Zeichen enthält
- ❖ Rückgabewert true, wenn der Teilstring enthalten ist
- ❖ unterscheidet Groß- und Kleinschreibung!
- ❖ Liefert nicht die Position des Teilstrings!

Suche

- ❖ Suchen eines Teilstrings in einem String:

indexOf(String value , int start)

- ❖ Beginnt mit der Suche am Stringanfang
- ❖ Gibt das erste Vorkommen des Teilstrings zurück
- ❖ unterscheidet Groß- und Kleinschreibung!
- ❖ Rückgabe von -1, wenn der Teilstring nicht enthalten ist

Suche

- ❖ Suchen eines Teilstrings bzw. Zeichens in einem String:
lastIndexOf(String value, int start)
lastIndexOf(char value, int start)
- ❖ Gibt das letzte Vorkommen des Teilstrings zurück
- ❖ unterscheidet Groß- und Kleinschreibung!
- ❖ Rückgabe von -1, wenn der Teilstring nicht enthalten ist

Vergleich

- ❖ Vergleich von zwei kompletten Strings:
equals(String value)
- ❖ Untersucht die absolute Übereinstimmung zweier Strings
- ❖ unterscheidet Groß- und Kleinschreibung!
- ❖ Sind beide Strings absolut identisch, wird true zurückgegeben

Weitere Vergleichsmethoden

- ❖ *equalsIgnoreCase(String value)*
- ❖ *startsWith(String value, int offset)*
- ❖ *endsWith(String value, int offset)*
- ❖ *regionMatches(boolean caseignore, int offset, String value, int offset2, int len)*
- ❖ *compareTo(String value) -> **lexikographischer Vergleich***
- ❖ *compareToIgnoreCase(String value)*

Methoden zum Extrahieren

- ❖ *charAt(int index)* // **Zeichen an angegebener Position**
- ❖ *substring(int start, int ende)* // neues Stringobjekt vom startindex bis zum endindex-1
- ❖ *split(String trennzeichen, int anz)* // Zerlegung in Teilstrings in in Stringarray.

Methoden zum Konvertieren

- ❖ *getChars(int start, int ende, char[] array)*
 - ❖ schreibt den angegebenen Teilstring in das zu übergebene Char-Array
- ❖ *valueOf(Variable value)*
 - ❖ Konvertiert eine übergebene Zahl oder ein übergebenes Datum in einen String

Methoden zum Konvertieren

- ❖ String in eine Primitive konvertieren
 - ❖ `parseBoolean(String s)`
 - ❖ `parseByte(String s)`
 - ❖ `parseShort(String s)`
 - ❖ `parseInt(String s)`
 - ❖ `parseLong(String s)`
 - ❖ `parseDouble(String s)`
 - ❖ `parseFloat(String s)`

Methoden zum Suchen & Ersetzen

- ❖ `replace(char alt, char neu)`
- ❖ `replace(String alt, String neu)`
- ❖ `replaceAll(String alt, String neu)`
- ❖ `replaceFirst(String alt, String neu)`

weitere Methoden

- ❖ *concat(String value)* // Anhängen des Strings
- ❖ *toLowerCase()* // alles in Kleinbuchstaben
- ❖ *toUpperCase()* // alles in Großbuchstaben
- ❖ *trim()* // Entfernen von Leerzeichen

Stringklassen StringBuffer & StringBuilder

- ❖ Änderbare Zeichenketten
- ❖ dynamisch änderbare Länge
- ❖ dynamisch änderbarer Inhalt
- ❖ Unterschied zwischen StringBuffer und StringBuilder:
 - ❖ StringBuffer schützt vor nebenläufigen Operationen
 - ❖ StringBuilder schützt nicht vor nebenläufigen Operationen
- ❖ Im Folgenden wird nur von StringBuffer gesprochen, StringBuilder ist hierzu identisch!

Konstrukturen

- ❖ Es gibt drei Konstrukturen zum Anlegen eines Stringobjektes mit new:
 - ❖ *StringBuffer()* erzeugt ein leeres StringBufferobjekt
 - ❖ *StringBuffer(int length)* erzeugt ein StringBufferobjekt mit einer bestimmten Länge
 - ❖ *StringBuffer(String value)* erzeugt ein neues StringBufferobjekt mit dem Inhalt und der Länge des übergebenen Strings
- ❖ *StringBuffer(CharSequence)* wie oben, aber auch aus Char-Array oder StringBuffer

Länge vom StringBuffer

- ❖ Die Länge kann über *length()* festgestellt werden.
- ❖ *capacity()* gibt die Puffergröße des StringBuffer an
- ❖ die Puffergröße gibt die Länge des Strings und die noch freien Feldelemente des StringBuffer an

Länge vom StringBuffer

- ❖ Ändern der StringBufferlänge:
 - ❖ *trimToSize()*
 - ❖ *setLength(int laenge)*
- ❖ *trimToSize()* verkleinert den StringBuffer auf die Länge des enthaltenen Strings
- ❖ *setLength()* setzt den StringBuffer auf eine neue Länge, ein längerer String wird abgeschnitten, ein kürzerer durch `\u0000` aufgefüllt

Methoden zum Anhängen von Daten

- ❖ `append(boolean b)`
- ❖ `append(char c)`
- ❖ `append(char[] carray)`
- ❖ `append(char c)`
- ❖ `append(double d)`
- ❖ `append(float f)`
- ❖ `append(int i)`
- ❖ `append(long l)`
- ❖ `append(String str)`
- ❖ `append(StringBuffer str)`

Besondere Methoden

- ❖ Löschen von Zeichenketten oder Zeichen:
 - ❖ *deleteCharAt(int index)*
 - ❖ *delete(int start, int ende)*
- ❖ Löscht ein einzelnes Zeichen bzw. einen durch Start- und Endposition angegebenen Teilstring

Besondere Methoden

- ❖ Umdrehen eines StringBuffer:
 - ❖ *reverse()*
- ❖ Einfügen eines StringBuffer:
 - ❖ *insert(int offset, Variable typ)*
- ❖ fügt an die Stelle *offset* den Wert *typ* ein (*typ* kann eine beliebige Angabe sein!)

Vergleichsmethoden

- ❖ Vergleichen eines Strings mit einem StringBuffer:
 - ❖ *contentEquals(StringBuffer value)*
- ❖ **Muss als Methode des Strings aufgerufen werden!!!**
- ❖ Vergleicht auf die bisher bekannte Art

Vergleichsmethoden

- ❖ Es existieren keine Vergleichsmethoden für StringBuffer und StringBuilder!
- ❖ Ein Vergleich ist nur durch die vorherige Konvertierung in einen String möglich:
 - ❖ *toString()*
- ❖ Die Funktion konvertiert einen StringBuffer oder einen StringBuilder in einen String
- ❖ Das Ergebnis kann wie ein normaler String behandelt werden!

Übung

- ❖ Schreiben Sie ein Programm, dass einen String rückwärts ausgibt, also aus „Ich mag Java“ wird „avaJ gam hcl“. Und zwar auf drei Arten.
- ❖ 1. Durch Benutzung der Klasse StringBuffer
- ❖ 2. Durch Benutzung der Klasse StringBuilder
- ❖ 3. Nur durch Benutzung der Klasse String

Übung 2

Übung 2 siehe github.