universität
innsbruck

Zeichenketten

Programmiermethodik

Lukas Kaltenbrunner, Simon Priller
Universität Innsbruck

Strings

- Die Klasse `String` ist in Java vordefiniert.
- String-Variablen enthalten Referenzen auf String-Objekte.
- Besonderheiten:
  - String-Objekte können aus String-Literalen und Textblöcken erzeugt werden.
  - Konkatenationsoperator + (und auch +=)
  - String Umwandlung (String Conversion)
  - String-Objekte sind nach dem Anlegen **nicht** mehr veränderbar! (engl. Immutable)
    → Jede Änderung erzeugt implizit ein neues Objekt.
  - Stringpool zur Vermeidung redundanter Speicherung (außer bei Erzeugung mit `new`).
- Die Klasse `String` implementiert das Interface `CharSequence`.

String-Klasse (1)

- Auswahl einiger Konstruktoren:
  - `String()`
    - Erzeugt ein neues Objekt ohne Zeichen (leerer String)
  - `String(String string)`
    - Erzeugt eine Kopie von `string`; unnötig, da Strings immutable sind
  - `String(char[] value)`
  - `String(char[] value, int offset, int count)`
  - `String(StringBuffer buffer)`
  - `String(StringBuilder builder)`

```
char[] charArray = {'V','O',' ','P','M'};

String s1 = "String-Literal";
String s2 = """
            Textblock""";                // s2 is equal to "Textblock"
String s3 = new String();                // s3 is equal to ""
String s4 = new String(charArray);       // s4 is equal to "VO PM"
String s5 = new String(s4);              // s5 is equal to "VO PM"
String s6 = new String(charArray, 3, 2); // s6 is equal to "PM"
```

String-Klasse (2)

- Die Klasse `String` verwaltet einen Stringpool, welcher zu Beginn leer ist.
  - Jeder String, welcher das Ergebnis eines konstanten Ausdrucks ist, wird nur einmal erzeugt und im Stringpool abgelegt.
  - Mit der Methode `intern()` kann ein String in den Stringpool eingefügt werden.
- Die Methode `equals()` gibt `true` zurück, wenn die Strings gleichlang sind und alle Zeichen übereinstimmen.

```
String s1 = "Programmiermethodik";
String s2 = "Programmiermethodik";
String s3 = new String(s1);

System.out.println(s1 == s2);                              // true
System.out.println(s1 == ("Programmier" + "methodik"));    // true
System.out.println(s1 == s3);                              // false

System.out.println(s1.equals(s3));                         // true
System.out.println(s1.equals("PROGRAMMIERMETHODIK"));      // false
```

String-Klasse (3)

- Auswahl einiger objektbezogener Methoden:

`int compareTo(String anotherString)`

- Lexikografischer Vergleich der beiden Strings.

`boolean equalsIgnoreCase(String anotherString)`

- Gibt `true` zurück, wenn die Strings gleichlang sind und alle Zeichen übereinstimmen. Unterschiede in der Groß- und Kleinschreibung werden ignoriert.

`int indexOf(int ch)`

- Index des ersten Vorkommnis des Zeichens `ch`.

`int length()`

- Länge des Strings.

`String replace(char oldChar, char newChar)`

- Ersetzt alle Vorkommen von `oldChar` mit `newChar`.

`String startsWith(String prefix)`

- Überprüft, ob das String-Objekt mit dem String `prefix` beginnt.

`String toLowerCase()`

- Wandelt alle Buchstaben in Kleinbuchstaben um.

`String toUpperCase()`

- Wandelt alle Buchstaben in Großbuchstaben um.

String-Klasse (4)

- Auswahl einiger Klassenmethoden:

```
static String join(CharSequence delimiter,
                   CharSequence... elements)
```

  - Verbindet mehrere Strings, wobei zwischen einzelnen Teilen `delimiter` eingesetzt wird.

```
static String valueOf(object)
```

  - Wandelt ein Objekt in einen String um.
  - Diese Methode ist überladen und kann auch für primitive Datentypen verwendet werden.

Formatierung von Strings

- Jede Methode, welche einen Text formatiert, erfordert einen Formatstring und entsprechende Argumentliste.
- Der Formatstring ist ein String, welcher einen einen fixen Text und beliebig viele Formatspezifizierer enthält.
- Folgende Methoden werden in der Klasse `String` für die Formatierung bereitgestellt:
  - `static String format(String format, Object... args)`
  - `static String format(Local l, String format, Object... args)`
  - `String formatted(Object... args)`
- Die Methode `System.out.printf()` kann für die formatierte Ausgabe auf dem Standarausgabestream `System.out` verwendet werden.

Formatspezifizierer

- Formatspezifizierer sind wie folgt aufgebaut:

  `%[argument_index][flags][width][.precision]converter`

  `[argument_index]`
  - Position in der Argumentenliste (erstes Arg mit `1$`, zweites Arg mit `2$`).
  - Das vorherige Argument kann mit `<` referenziert werden.

  `[flags]`
  - Menge von Zeichen zur Modifikation der Ausgabe (beispielsweise `0`: führende 0en, `+`: Vorzeichen).

  `[width]`
  - Minimale Anzahl der Zeichen, die ausgegeben werden.

  `[.precision]`
  - Maximale Anzahl der Zeichen oder Genauigkeit bei Gleitkommazahlen.

  `converter`
  - Zeichen, das die Formatierung des Arguments bestimmt - erlaubte Formatierungen hängen vom jeweiligen Datentyp ab.
  - Ist als einziger Teil nicht optional.

- Mehr Details in der Dokumentation

Converter bei Formatspezifizierern

Converter	Beschreibung	Zulässige Typen
`'b','B'`	`true` oder `false` bei `Boolean`; `false` bei `null`; sonst `true`	Beliebig
`'h','H'`	Hexadezimale Repräsentation des Hashcodes	Beliebig
`'s','S'`	Stringrepräsentation	Beliebig
`'c','C'`	Unicodezeichen	`Character`, `Byte` und `Short`
`'d'`	Dezimalzahl	`Byte`, `Short`, `Integer`, `Long`, und `BigInteger`
`'o'`	Oktalzahl	`Byte`, `Short`, `Integer`, `Long`, und `BigInteger`
`'x','X'`	Hexadezimalzahl	`Byte`, `Short`, `Integer`, `Long`, und `BigInteger`
`'e','E'`	Exponentialdarstellung	`Float`, `Double` und `BigDecimal`
`'f'`	Dezimalzahl	`Float`, `Double` und `BigDecimal`
`'g','G'`	Exponentialdarstellung oder Dezimalzahl	`Float`, `Double` und `BigDecimal`
`'a','A'`	Hexadezimale Gleitkommazahl	`Float`, `Double`
`'t','T'`	Präfix zur Datums- und Zeitumwandlung	`Long`, `Calendar`, `Date` und `TemporalAccessor`
`'%'`	Prozentzeichen	
`'n'`	Plattformspezifischer Zeilenumbruch	

- Sofern nicht anders angegeben ist das Ergebnis bei `null` der String null.
- Bei Convertern mit Großbuchstaben werden keine Kleinbuchstaben im Resultat verwendet.
- Statt den Wrapper-Typen können bei den zulässigen Typen auch die entsprechenden primitiven Datentypen verwendet werden.

Erzeugung/Ausgabe langer Strings

```
public class Lecture {
    private final String title;
    private List<String> topics;
    private LocalDate lastUpdate;

    …

    public String getDescription() {
        String month = String.valueOf(lastUpdate.getMonthValue());
        String formattedMonth = month.length() < 2 ? "0" + month : month;
        String day = String.valueOf(lastUpdate.getDayOfMonth());
        String formattedDay = day.length() < 2 ? "0" + day : day;
        return "Die Vorlesung " + title + " befasst sich mit " + topics +
                " (letztes Update: " + lastUpdate.getYear() + "-" + formattedMonth +
                "-" + formattedDay + ")";
    }

    …
}
```

Favor Format over Concatenation

```java
public class Lecture {
    private final String title;
    private List<String> topics;
    private LocalDate lastUpdate;

    …

    public String getDescription() {
        return "Die Vorlesung %s befasst sich mit %s (letztes Update: %tF)"
                .formatted(title, topics, lastUpdate);
    }

    …
}
```

- Vorher:
  - Methode `getDescription` ist nicht komplex, aber trotzdem schwer zu lesen
- Nachher:
  - Wir trennen das Layout des Strings (wie wird der String ausgegeben) von den Daten (was wird ausgegeben).
  - Leichter lesbar und übersichtlicher

StringBuilder/StringBuffer

- Verhält sich wie String, kann aber editiert werden.
  - String-Objekte sind nach dem Erzeugen nicht mehr veränderbar.
  - Jede Konkatenation bei Strings erzeugt neue String-Objekte.
  - Viele Objekte anzulegen ist schlecht für die Performance!
- Kann dazu verwendet werden Zeichenketten aufzubauen.
- Nach dem Editieren kann ein `StringBuilder` bzw. `StringBuffer` in einen String umgewandelt werden.
- Beide Klassen besitzen eine identische API.
  - Die Methoden der Klasse `StringBuffer` sind synchronisiert.
- Die Klassen `StringBuilder` und `StringBuffer` implementierten das Interface `CharSequence`.

StringBuilder (1)

- Auswahl einiger Konstruktoren:

  `StringBuilder()`
  - Erzeugt einen `StringBuilder` ohne Zeichen.

  `StringBuilder(String str)`
  - Erzeugt einen `StringBuilder`, welcher mit dem Inhalt des übergeben Strings initialisiert wird.

- Die Klasse `StringBuilder` überschreibt die Methode `equals` nicht.
- Die Methode `toString` wird überschrieben und liefert eine String-Repräsentation der Zeichenkette.

```
StringBuilder sb1 = new StringBuilder("Programmiermethodik");
StringBuilder sb2 = new StringBuilder("Programmiermethodik");

System.out.println(sb1.equals(sb2));                        // false
System.out.println(sb1.toString().equals(sb2));             // false
System.out.println(sb1.toString().equals(sb2.toString()));  // true
System.out.println(sb1.toString().contentEquals(sb2));      // true
```

StringBuilder (2)

- Auswahl einiger Methoden:
  - `StringBuilder append(Object obj)`
    - Hängt die String-Repräsentation des Objekts an das Ende der Zeichenkette.
    - Diese Methode ist überladen und kann auf für primitive Datentypen verwendet werden.
  - `StringBuilder delete(int start, int end)`
    - Löscht die Zeichen des Substrings.
  - `StringBuilder insert(int offset, Object obj)`
    - Fügt die String-Repräsentation des Objekts ein.
    - Diese Methode ist überladen und kann auf für primitive Datentypen verwendet werden.
  - `StringBuilder replace(int start, int end, String str)`
    - Ersetzt die Zeichen im Substring der Zeichenkette mit den Zeichen aus `str`.
  - `int length()`
    - Ermittelt die Länge der Zeichenkette.
  - `StringBuilder reverse()`
    - Dreht die Zeichenfolge um.
  - `String toString()`
    - Gibt eine String-Repräsentation der Zeichenkette zurück.

Quellen

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, Gavin Bierman: **The Java® Language Specification (***Java SE 17 Edition***)**, Oracle, 2021
- Michael Inden: **Der Weg zum Java-Profi: Konzepte und Techniken für die professionelle Java-Entwicklung**, dpunkt.verlag, 5. Auflage, 2021
- Christian Ullenboom: **Java ist auch eine Insel: Einführung, Ausbildung, Praxis**, Rheinwerk Verlag, 16. Auflage, 2022 (Java 17)
- Simon Harrer, Jörg Lenhard, Linus Dietz: **Java by Comparison: Become a Java Craftsman in 70 Examples**, The Pragmatic Programmers, LLC, 2018