ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Zusatzbeispiele Formatierung

Formatierungen

- Seit JDK 1.5 neue Methoden in der Klasse `String`:

```
public static String format(Locale l,
                            String format,
                            Object... args)
// Returns a formatted string using the specified
// locale, format string, and arguments.

public static String format(String format,
                            Object... args)
// Returns a formatted string using the specified format
// string and arguments.
```

Locale

- Zitat Java API:
  - ⇨ *A Locale object represents a specific geographical, political, or cultural region.*
  - ⇨ *An operation that requires a Locale to perform its task is called locale-sensitive and uses the Locale to tailor information for the user.*
- Locale können angewendet werden bei der Ausgabe von Zahlen, sowie Datums- und Zeitangaben
- Klasse `java.util.Locale`

LocaleTest (1)

```
import java.io.*;
import java.util.*;

public class LocaleTest {
    private PrintStream out = System.out;

    public void start() {
        out.println("\nDefault Locale:");
        Locale l = Locale.getDefault();
        info(l);
        l = Locale.FRANCE;
        out.println("\nLocale.FRANCE:");
        info(l);
        l = Locale.US;
        out.println("\nLocale.US:");
        info(l);

        System.out.println("Verfügbare Locales");
        Locale[] ltab = Locale.getAvailableLocales();
        for (Locale l1 : ltab)
            System.out.println(l1);
    }
```

LocaleTest (2)

```
public void info(Locale l) {
        out.println("DisplayCountry : " + l.getDisplayCountry());
        out.println("DisplayLanguage: " + l.getDisplayLanguage());
        out.println("DisplayName    : " + l.getDisplayName());
        out.println("DisplayScript  : " + l.getDisplayScript());
        out.println("DisplayVariant : " + l.getDisplayVariant());
        out.println("ISO3Country    : " + l.getISO3Country());
        out.println("ISO3Language   : " + l.getISO3Language());
        out.println("Language       : " + l.getLanguage());
        out.println("Variant        : " + l.getVariant());
    }

    public static void main(String args[]) {
        new LocaleTest().start();
    }
}
```


Format-String

- Format-Strings in Java sind ähnlich aufgebaut wie in C/C++
  - ⇨ Siehe dazu Beschreibung der Klasse **`java.util.Formatter`**
- Ein Format-String kann Text und sogenannte Format-Spezifizierer enthalten
- Format-Spezifizierer:

  `%[argument_index$][flags][width][.precision]conversion`
  - ⇨ **`argument_index`** : optionale Dezimalzahl, Position des Arguments in der Argumentliste, z. B. 1$ oder 2$
  - ⇨ **`flags`**: optionale Menge an Zeichen, die das Ausgabeformat beeinflussen
  - ⇨ **`width`**: optionale positive ganze Zahl für die Ausgabebreite
  - ⇨ **`precision`**: optionale Angabe von z. B. Anzahl Nachkommastellen
  - ⇨ **`conversion`**: Angabe der Ausgabekonvertierung

Format-Spezifizierer

conversion	
b	boolescher Wert
c	einzelner Character
d	Ganzzahl in Dezimaldarstellung
o	Ganzzahl in Oktaldarstellung
x	Ganzzahl in Hexadezimaldarstellung
f	Fließkommazahl
e	Fließkommazahl mit Exponent
t	Prefix für Datums-/Zeitangaben
s	Strings und andere Objekte

flags	
-	Linksbündige Ausgabe
+	Vorzeichen immer ausgegeben
0	Zahlen werden mit Nullen aufgefüllt
,	Zahlen werden mit Tausenderpunkten ausgegeben
(	Negative Zahlen werden in Klammern eingeschlossen

Formatierungen

- Beispiele:

```
int i = 31;
double x = Math.PI;
String s = String.format("Zahl: %d\n", i);
System.out.println(s);
s = String.format(
   "Zahl: dezimal:%1$d oktal:%1$o hex:%1$x\n", i);
System.out.println(s);

s = String.format("Zahl: |%8d|\n", i);
// 8 Stellen breit
System.out.println(s);
s = String.format("Zahl: |%-8d|\n", i);
// linksbündig
System.out.println(s);
s = String.format("PI: |%8.4f|\n", x);
// 8 Stellen, 4 Nachkommastellen
System.out.println(s);
```

```
Zahl: 31

Zahl: dezimal:31
oktal:37 hex:1f


Zahl: |      31|


Zahl: |31      |


PI: |  3,1416|
```

Formatierungen

- Beispiele:

```
s =
String.format(Locale.US, "PI: |%8.4f|\n", x);
// 8 Stellen, 4 Nachkommastellen
System.out.println(s);

s =
String.format(Locale.FRANCE, "PI: |%8.4f|\n", x);
// 8 Stellen, 4 Nachkommastellen
System.out.println(s);

Calendar c = Calendar.getInstance();
s = String.format("Heute: %1$td.%1$tm.%1$tY", c);
System.out.println(s);
```

```
PI: |  3.1416|


PI: |  3,1416|


Heute: 06.12.201
```

Veraltete Datumsklasse

Formatierungen

- Ab JDK 1.5 neue Methoden in der Klasse `PrintStream`:

```
public PrintStream format(Locale l, String format,
                          Object... args)
public PrintStream printf(Locale l, String format,
                          Object... args)
// A convenience method to write a formatted string to
// this output stream using the specified format string
// and arguments.

public PrintStream format(String format,
                          Object... args)
public PrintStream printf(String format,
                          Object... args)
// Writes a formatted string to this output stream using
// the specified format string and arguments.
```

Formatierungen

- Beispiele:

```
int i = 31;
double x = Math.PI;
System.out.printf("Zahl: %d\n", i);
System.out.printf("Zahl: dezimal:%1$d
            oktal:%1$o hex:%1$x\n", i);
System.out.printf("Zahl: |%8d|\n", i);
// 8 Stellen breit

System.out.printf("Zahl: |%-8d|\n", i);
// linksbündig

System.out.printf("PI: |%8.4f|\n", x);
// 8 Stellen, 4 Nachkommastellen

System.out.printf(Locale.US, "PI: |%8.4f|\n", x);
// 8 Stellen, 4 Nachkommastellen
```

```
Zahl: 31
Zahl: dezimal:31
oktal:37 hex:1f
Zahl: |      31|


Zahl: |31      |


PI: |  3,1416|

PI: |  3.1416|
```


Formatierungen

- Beispiele:

```
Calendar c = Calendar.getInstance();
System.out.printf("Heute: %1$td.%1$tm.%1$tY\n", c);
System.out.printf("Text: |%-20s|\n","Informatik");
```

```
Heute: 10.12.2014
Text: |Informatik          |
```

java.time

- Seit Java 8 gibt es eine komplett neue Datum/Zeit-API

`java.time`	Basispaket mit grundlegenden Klassen für Datum, Zeit, Zeitzonen, Zeitdauern und Uhren
`java.time.format`	Klassen zum Formatieren und Parsen von Datums- und Zeitangaben

- Details dazu im Oracle-Tutorial:
  https://docs.oracle.com/javase/tutorial/datetime/

java.time

- Wesentliche Klassen

`java.time`	
`LocalDate`	Datum ohne Zeitzone
`LocalDateTime`	Datum und Uhrzeit ohne Zeitzone
`LocalTime`	Uhrzeit ohne Zeitzone
`ZonedDateTime`	Datum und Uhrzeit unter Berücksichtigung der Zeitzone

`java.time.format`	
`DateTimeFormatter`	Klasse zum Formatieren und Parsen von Datums- und Zeitangaben

Beispiel: TimeFormat (1)

```
public class TimeFormat {
    public static void test1() {
        LocalDate today = LocalDate.now(); // heutiger Tag
        LocalDate date  = LocalDate.of(2017, Month.JANUARY, 6);
                          // Datum aus Datumsangabe konstruieren
        LocalDate soon = today.plusMonths(6);
                         // sechs Monate dazuaddieren
        System.out.println("today: " + today +
            " date: " + date  +
            " soon: " + soon);
        // klassisches Formatieren
        // 1$ bezieht sich auf den 1. Parameter
        // %td Tag des Monats 2-stellig dargestellt
        // %tm Monat als 2-stellige Zahl
        // %tY Jahr als 4-stellige zahl
        System.out.printf("Heute: %1$td.%1$tm.%1$tY\n", today);
```


Beispiel: TimeFormat (2)

```
        // Formatieren mit java.time.format
        DateTimeFormatter formatter =
                          DateTimeFormatter.ofPattern("dd.MM.yyyy");
        String todayFormat = today.format(formatter);
        System.out.println("Heute: " + todayFormat);

    }
```

```
today: 2016-12-29 date: 2017-01-06 soon: 2017-06-29
Heute: 29.12.2016
Heute: 29.12.2016
```