



Neue Themen für die 808

Februar 2025

Plan für die Woche

Montag

- Selbsteinschätzung
- Heap und Stack
- Garbage Collector und Object Lifecycle

Dienstag

- Exceptions

Mittwoch

- StringBuilder vs. String

Donnerstag

- Calender API

Freitag

- Wiederholung ArrayLists
- Lambdas

Plan für heute

- LocalDate
- LocalTime
- LocalDateTime
- Period
- DateTimeFormatter

LocalDate, LocalDateTime, LocalTime

BUCHSEITEN S.138-144

LocalDate

- beinhaltet nur das Datum
 - keine Zeitzonen oder Uhrzeit
- **Beispiel:**
 - Geburtsdatum dieses Jahr -> ganzer Tag kann zum Feiern genutzt werden

LocalDate – Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `now()`:
 - Das heutige Datum
 - `LocalDate.now()`

```
public static void main(String[] args) {  
    LocalDate heute = LocalDate.now();  
  
    System.out.println("Heutiges Datum: " + heute);  
}
```

Heutiges Datum: 2025-02-13



ISO_DATE - Format

LocalDate – Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `of(int year, int month, int dayOfMonth)`
`of(int year, Month month, int dayOfMonth)`


```
public static void main(String[] args) {  
    LocalDate geburtsdatum = LocalDate.of(1895, 5, 23);  
  
    System.out.println("Geburtsdatum: " + geburtsdatum);  
}
```

Geburtsdatum: 1895-05-23

LocalDate – Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `of(int year, int month, int dayOfMonth)`
`of(int year, Month month, int dayOfMonth)`

```
public static void main(String[] args) {  
    LocalDate geburtsdatum = LocalDate.of(1895, Month.MAY, 23);  
  
    System.out.println("Geburtsdatum: " + geburtsdatum);  
}
```



Konstante

Geburtsdatum: 1895-05-23

LocalTime

- beinhaltet nur die Zeit
 - keine Zeitzonen oder Datum
- **Beispiel:**
 - 12 Uhr mittags

LocalTime – Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `now()`:
 - Die jetzige Uhrzeit

```
public static void main(String[] args) {  
    LocalTime zeit = LocalTime.now();  
  
    System.out.println("Zeit: " + zeit);  
}
```

Zeit: 17:23:13.887

→ Systemuhr in der lokalen Zeit

LocalTime – Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `of(int hour, int minute)`
 - `of(int hour, int minute, int second)`
 - `of(int hour, int minute, int second, int nanosecond)`

```
public static void main(String[] args) {  
    LocalTime zeit = LocalTime.of(12, 12, 12);  
  
    System.out.println("Zeit: " + zeit);  
}
```

Zeit: 12:12:12

LocalDateTime

- beinhaltet Datum und Zeit
 - keine Zeitzonen
- Beispiel:
 - 12 Uhr mittags am 12.02.2025

LocalDateTime - Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `now()`:
 - Die jetzige Uhrzeit und heutiges Datum

```
public static void main(String[] args) {  
    LocalDateTime zeit = LocalDateTime.now();  
  
    System.out.println("Zeit: " + zeit);  
}
```

Zeit: 2025-02-12T17:39:34.779

T signalisiert, wo die Zeit anfängt

LocalDateTime - erzeugende Methoden

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen

`of(int year, int month, int dayOfMonth, int hour, int minute)`

Obtains an instance of `LocalDateTime` from year, month, day, hour and minute, setting the second and nanosecond to zero.

`of(int year, int month, int dayOfMonth, int hour, int minute, int second)`

Obtains an instance of `LocalDateTime` from year, month, day, hour, minute and second, setting the nanosecond to zero.

`of(int year, int month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond)`

Obtains an instance of `LocalDateTime` from year, month, day, hour, minute, second and nanosecond.

`of(int year, Month month, int dayOfMonth, int hour, int minute)`

Obtains an instance of `LocalDateTime` from year, month, day, hour and minute, setting the second and nanosecond to zero.

`of(int year, Month month, int dayOfMonth, int hour, int minute, int second)`

Obtains an instance of `LocalDateTime` from year, month, day, hour, minute and second, setting the nanosecond to zero.

`of(int year, Month month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond)`

Obtains an instance of `LocalDateTime` from year, month, day, hour, minute, second and nanosecond.

`of(LocalDate date, LocalTime time)`

Obtains an instance of `LocalDateTime` from a date and time.

LocalDateTime – Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `of(int year, int month, int day, int hour, int minute)`

```
public static void main(String[] args) {  
  
    LocalDateTime meeting = LocalDateTime.of(2025, 3, 15, 10, 30);  
    System.out.println("Meeting geplant für: " + meeting);  
}
```

Meeting geplant für: 2025-03-15T10:30

LocalDateTime – Methoden (1)

- statische Methoden!
 - Beim Aufruf der Methode muss die Klasse dabei stehen
- `of(LocalDate date, LocalTime time)`

```
LocalDate date = LocalDate.now();  
LocalTime time = LocalTime.now();  
LocalDateTime secondMeeting = LocalDateTime.of(date, time);  
System.out.println("Meeting geplant für: " + secondMeeting);
```

Meeting geplant für: 2025-02-12T17:52:04.658

Good To Know

- von den vorherig genannten Klassen kann KEIN Objekt erstellt werden!
 - Die Klasse haben nämlich einen privaten Konstruktor
 - Beim Aufruf eines Konstruktors erfolgt ein Compilerfehler
- Versucht man invalide Nummern zu übergeben, wird eine Exception geworfen

```
LocalDate.of(2015, Month.JANUARY, 32);
```



Aufgabe

Schreibe ein Java-Programm, das:

- Das aktuelle Datum und die aktuelle Uhrzeit ausgibt.
- Den Benutzer nach einem zukünftigen Termin (Datum und Uhrzeit) fragt.
- Mache Konsolenausgaben



Manipulieren der Daten

- Die Date-Klassen sind immutable (=unveränderbar)
 - Deswegen muss man das Ergebnis einer Variablen zuweisen, sonst wird das Ergebnis ignoriert!

```
public static void main(String[] args) {  
    LocalDate date = LocalDate.now();  
    date.plusDays(3);  
    System.out.println(date); //Ausgabe: 2025-02-12  
  
    date = date.plusDays(3);  
    System.out.println(date); //Ausgabe: 2025-02-15  
}
```

Manipulieren der Daten

- Die Date-Klassen sind immutable (=unveränderbar)
 - Deswegen muss man das Ergebnis einer Variablen zuweisen, sonst wird das Ergebnis ignoriert!

```
public static void main(String[] args) {  
    LocalDate date = LocalDate.now();  
    date.plusDays(3);  
    System.out.println(date); //Ausgabe: 2025-02-12  
  
    date = date.plusDays(3);  
    System.out.println(date); //Ausgabe: 2025-02-15  
}
```

Es gibt weitere Methoden

Methodenverkettung
funktioniert hier auch!

Jede Klasse hat auch
getter-Methoden für jedes
einzelne Element

Methoden (2)

- es gibt eine weitere Vielzahl an Methoden, die man auf die Date-Klassen anwenden kann
- sind nicht statisch

	↓ LocalDate	↓ LocalTime	LocalDateTime
plusYears/minusYears	✓	X	✓
plusMonths/minusMonths	✓	X	✓
plusWeeks/minusWeeks	✓	X	✓
plusDays/minusDays	✓	X	✓
plusHours/minusHours	X	✓	✓
plusMinutes/minusMinutes	X	✓	✓
plusSeconds/minusSeconds	X	✓	✓
plusNanos/minusNanos	X	✓	✓

Aufgabe

Schreibe ein Java-Programm, das folgende Schritte ausführt:

- Erstelle ein `LocalDate`-Objekt mit dem heutigen Datum und gib es aus.
 - Berechne das Datum in 3 Jahren, 2 Monaten und 10 Tagen und gib es aus.
- Erstelle ein `LocalTime`-Objekt mit der aktuellen Uhrzeit.
 - Berechne die Uhrzeit in 5 Stunden und 30 Minuten und gib sie aus.
- Erstelle ein `LocalDateTime`-Objekt mit dem aktuellen Datum und der aktuellen Uhrzeit.
 - Berechne das Datum und die Uhrzeit 1 Woche in der Vergangenheit und gib es aus.



Period

BUCHSEITEN S.145-147

Period

- repräsentiert eine Zeitspanne
 - Ein Jahr, ein Monat, ein Tag
 - Beispiel: Ein Fußballspiel dauert 90 Minuten.
- beispielsweise um Zeitspannen zwischen zwei Daten zu berechnen
- kann man auf das Datum der LocalDate, LocalDateTimes mittels .plus(Period p) addieren

`of(int years, int months, int days)`

Obtains a `Period` representing a number of years, months and days.

`ofDays(int days)`

Obtains a `Period` representing a number of days.

`ofMonths(int months)`

Obtains a `Period` representing a number of months.

`ofWeeks(int weeks)`

Obtains a `Period` representing a number of weeks.

`ofYears(int years)`

Obtains a `Period` representing a number of years.

Period

- repräsentiert eine Zeitspanne
 - Ein Jahr, ein Monat, ein Tag
 - Beispiel: Ein Fußballspiel dauert 90 Minuten.
- beispielsweise um Zeitspannen zwischen zwei Daten zu berechnen
- kann man auf das Datum der LocalDate, LocalDateTimes mittels .plus(Period p) addieren

```
public static void main(String[] args) {  
    LocalDate start = LocalDate.of(2020, 1, 1);  
    LocalDate ende = LocalDate.now();  
  
    Period periode = Period.between(start, ende);  
    System.out.println("Verstrichene Zeit: " + periode.getYears() + " Jahre, " + periode.getMonths() + " Monate, " + periode.getDays() + " Tage");  
}
```

Verstrichene Zeit: 5 Jahre, 1 Monate, 11 Tage

Aufgabe

Erstelle ein `LocalDate`-Objekt mit dem aktuellen Datum.

Erstelle ein weiteres `LocalDate`-Objekt für den 1. Januar 2028.

Berechne die Differenz zwischen diesen beiden Daten mit `Period` und gib aus, wie viele Jahre, Monate und Tage dazwischen liegen.

Erstelle ein `LocalTime`-Objekt mit der aktuellen Uhrzeit.

Erstelle ein weiteres `LocalTime`-Objekt, das um 3 Stunden und 45 Minuten später liegt.

Erstelle ein `LocalDateTime`-Objekt für Silvester (31. Dezember 2024, 23:59 Uhr) und überprüfe, ob es nach der aktuellen Zeit liegt.



`LocalDateTime` hat eine
Methode
`isAfter(LocalDate dt);`



DateTimeFormatter

BUCHSEITEN S.148-151

DateTimeFormatter

- wird genutzt, um ein Datumformat in ein anderes Datumsformat umzuwandeln
- ISO_DATE
 - Internationaler Standard für Datum
 - Zeit darf nicht mitgegeben werden
- ISO_TIME
 - Internationaler Standard für Zeit
 - Datum darf nicht mitgegeben werden
- ISO_DATE_TIME
 - Internationaler Standard, achtend für Datum und Zeit
 - Zeit und Datum MÜSSEN mitgegeben werden

DateTimeFormatter

– format(DateTimeFormatter.Konstante)

```
LocalDate date = LocalDate.now();  
LocalTime time = LocalTime.now();  
LocalDateTime localDateTime = LocalDateTime.now();  
  
System.out.println(date.format(DateTimeFormatter.ISO_DATE));  
System.out.println(time.format(DateTimeFormatter.ISO_TIME));  
System.out.println(localDateTime.format(DateTimeFormatter.ISO_DATE_TIME));
```

2025-02-12

18:57:56.077

2025-02-12T18:57:56.077

DateTimeFormatter

- man kann auch sein eigenes Pattern erstellen, falls keine passend sind
- ofPattern(String s):
 - M für Monat (1,2,3,4....)
 - MM für Monat (01,02,03,...)
 - MMM für Monat (Jan, Feb,...)
 - MMMM für Monat (January, February,...)
 - d für Tag (1,2,3,...)
 - dd für Tag (01,02,03,...)
 - yy für Jahr (25, 26, 27,...)
 - yyyy für Jahr (2025, 2026,...)

DateTimeFormatter

- man kann auch sein eigenes Pattern erstellen, falls keine passend sind
- ofPattern(String s):
 - h für Stunden (1, 2, 3,...)
 - hh für Stunden (02,03,04,...) //0-12
 - HH für Stunden(01,02,03,...23) //0-23
 - mm für Minuten
 - ss für Sekunden

DateTimeFormatter

- `format(DateFormat):`
 - Konvertiert ein übergebenes Datumsformat in das Gewünschte

```
public static void main(String[] args) {  
  
    LocalDateTime localDateTime = LocalDateTime.now();  
  
    DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("dd.MM.yyyy hh:mm:ss");  
    System.out.println(dateTimeFormatter.format(localDateTime));  
}
```


Parsing Date or Times

- man kann auch einen String in ein Datumsformat konvertieren
- jede Date-Klasse hat eine parse()-Methode
- parse(String s)

– parse(String s, DateTimeFormatter f)

```
DateTimeFormatter f =  
DateTimeFormatter.ofPattern("dd.MM.yyyy");  
  
LocalDate date2 = LocalDate.parse("02.08.2000", f);  
System.out.println(date2);
```

2000-08-02

```
LocalDate date = LocalDate.parse("2015-12-12");  
LocalTime time = LocalTime.parse("11:20");  
System.out.println(date);  
System.out.println(time);
```

2015-12-12
11:20

Aufgabe

Erstelle ein `LocalDate`-Objekt mit dem aktuellen Datum und formatiere es, sodass beispielsweise 12.02.2025 rauskommt.

Erstelle ein `LocalTime`-Objekt mit der aktuellen Uhrzeit und formatiere es, sodass 12:32:50 rauskommt.

Erstelle ein `LocalDateTime`-Objekt mit dem aktuellen Datum und der aktuellen Zeit und formatiere es , sodass beispielsweise 12 Februar 2025 - 14:30 rauskommt.

Gib die formatierten Strings in der Konsole aus.



Aufgabe

Gegeben sind drei Strings:

- "28.02.2025" (ein Datum)
- "14:45:30" (eine Uhrzeit)
- "28.02.2025 14:45" (Datum und Zeit kombiniert)

Schreibe ein Programm, das diese Strings mit `DateTimeFormatter` in `LocalDate`, `LocalTime` und `LocalDateTime` umwandelt.

Gib die geparsten Werte in der Konsole aus.

