

Formation Java 11

Date & Time API

Date Time API

<code>java.time.LocalDate</code>	Une date (ex. 2016-03-24)
<code>java.time.LocalTime</code>	Une heure (ex. 10:02:03)
<code>java.time.LocalDateTime</code>	Une date heure (ex. 2016-03-24 10:02:03)
<code>java.time.Period</code>	Durée en année/mois/jour (ex. 1a3m2j)
<code>java.time.Instant</code>	Un point instantané dans le temps (Fuseau)
<code>java.time.Duration</code>	Durée en heure/minute/seconde (ex. 02:03:02)
<code>java.time.ZonedDateTime</code>	Une date heure avec le fuseau horaire
<code>java.time.ZoneId</code>	Identifiant d'une zone de fuseau horaire

java.time.LocalDate (1)

```
LocalDate result = LocalDate.of(2017, 2, 24);
```

```
result.getYear() // 2017;
```

```
result.getMonth() // Month.FEBRUARY
```

```
result.getDayOfMonth() // 24
```

```
result.getDayOfWeek() // DayOfWeek.FRIDAY
```

```
result.getDayOfYear() // 55
```

```
result.get(ChronoField.YEAR), // 2017
```

```
result.get(ChronoField.MONTH_OF_YEAR) // Month.FEBRUARY.getValue()
```

```
result.get(ChronoField.DAY_OF_MONTH), // 24
```

```
result.get(ChronoField.DAY_OF_WEEK), //
```

```
DayOfWeek.FRIDAY.getValue()
```

```
result.get(ChronoField.DAY_OF_YEAR), // 55
```

java.time.LocalDate (2)

```
// String => LocalDate
LocalDate result = LocalDate.parse("2017-02-24");

result.getYear() // 2017;
result.getMonth() // Month.FEBRUARY
result.getDayOfMonth() // 24
result.getDayOfWeek() // DayOfWeek.FRIDAY
result.getDayOfYear() // 55
```

java.time.LocalDate (3)

```
LocalDate localDate = LocalDate.of(2017, 3, 24);
```

```
// Modifier une date consiste à créer une nouvelle date
```

```
// "2016-03-24"
```

```
LocalDate newLocalDate1 = localDate.withYear(2016);
```

```
// "2017-03-22"
```

```
LocalDate newLocalDate2 = localDate.with(ChronoField.DAY_OF_MONTH, 22);
```

```
// "2017-05-24"
```

```
LocalDate newLocalDate3 = localDate.plus(2, ChronoUnit.MONTHS);
```

```
// Pour avoir la date courante
```

```
LocalDate newLocalDate4 = localDate.now();
```

java.time.LocalDateTime

```
LocalTime time1 = LocalDateTime.of(20, 53, 01);
```

```
time1.getHour() // 20  
time1.getMinute() // 53  
time1.getSecond() // 01
```

```
LocalTime time2 = LocalDateTime.parse("20:53:01");
```

```
time2.getHour() // 20  
time2.getMinute() // 53  
time2.getSecond() // 01
```

```
LocalTime time3 = time.withHour(21); // 21:53:01
```

java.time.LocalDateTime

```
// 24/02/2017 20:53:01  
LocalDateTime dt1 = LocalDateTime.of(2017, 2, 24, 20, 53, 1);
```

```
// 24/02/2017 20:53:01  
LocalDate localDate = LocalDate.of(2017, 2, 24);  
LocalTime localTime = LocalTime.of(20, 53, 1);  
LocalDateTime dt3 = LocalDateTime.of(localDate, localTime);
```

```
// Création à partir d'une date  
LocalDateTime dt4 = localDate.atTime(20, 53, 1);
```

```
// Création à partir d'une heure  
LocalDateTime dt5 = localTime.atDate(localDate);
```

java.time.Duration & java.time.Period

```
// Duration
LocalTime time1 = LocalTime.of(20, 53, 1);
LocalTime time2 = LocalTime.of(20, 54, 1);

Duration duration = Duration.between(time1, time2);
duration.toMinutes() // 1L
duration.toNanos() // 60_000_000_000L

// Period
Period period = Period.between(LocalDate.of(2017, 3, 24), LocalDate.of(2017, 3, 25));
assertThat(period.getDays(), is(1));

Period period = Period.of(1, 2, 3);
period.toString() // "P1Y2M3D"
```


java.time.Instant

// 3 secondes depuis 1970. L'instant prend en compte le fuseau horaire.

```
Instant instant1 = Instant.ofEpochSecond(3);  
// nanosecond precision  
Instant instant2 = Instant.ofEpochSecond(3, 0);  
Instant instant3 = Instant.ofEpochSecond(2, 1_000_000_000);  
Instant instant4 = Instant.ofEpochSecond(4, -1_000_000_000);
```

toInstant()

- La méthode `toInstant()` a été ajoutée aux classes `java.util.Date` et `java.util.Calendar`.

```
Date currentDate = new Date();  
Instant now = currentDate.toInstant();  
ZoneId currentZone = ZoneId.systemDefault();
```

```
LocalDateTime localDateTime = LocalDateTime.ofInstant(now, currentZone);  
ZonedDateTime zonedDateTime = ZonedDateTime.ofInstant(now, currentZone);
```

ZonedDateTime

```
// 2016-01-15T15:14:15.662+01:00[Europe/Paris]  
ZonedDateTime.now();
```

```
// 2016-01-15T15:14:15.662+02:00[Europe/Paris]  
ZonedDateTime.now(ZoneId.of("Europe/Paris"));
```

```
// java.time.zone.ZoneRulesException:  
// Unknown time-zone ID: Europe/Nantes  
ZonedDateTime.now(ZoneId.of("Europe/Nantes"));
```

```
//2016-07-02T16:52:25.374-04:00[America/Indiana/Indianapolis]  
ZonedDateTime.now(ZoneId.of("America/Indiana/Indianapolis"));
```

```
LocalDateTime aLocalDateTime = LocalDateTime.of(2016, 01, 11, 17, 30);
```

```
// 2016-01-11T17:30+01:00[Europe/Paris]  
ZonedDateTime e = ZonedDateTime.of(aLocalDateTime, ZoneId.of("Europe/Paris"));
```

java.time.format.DateTimeFormatter

```
LocalDate result = LocalDate.parse("2017-02-24");
```

```
// 20170224
```

```
result.format(DateTimeFormatter.BASIC_ISO_DATE);
```

```
// 2017-02-24
```

```
result.format(DateTimeFormatter.ISO_LOCAL_DATE);
```

```
// 24/02-2017
```

```
result.format(DateTimeFormatter.ofPattern("dd/MM-  
yyyy"));
```

Classe TemporalAdjusters

- Permet des opérations usuelles sur des dates.
- Exemple : trouver le prochain lundi du mois suivant.

```
LocalDate localDate = LocalDate.of(2017, 1, 1);  
  
localDate.with(TemporalAdjusters.firstDayOfNextMonth());  
localDate.with(TemporalAdjusters.firstDayOfNextYear());  
localDate.with(TemporalAdjusters.lastDayOfMonth());  
localDate.with(TemporalAdjusters.next(DayOfWeek.FRIDAY));
```

Travaux Pratiques