

Introduction à la programmation en Java

Cours 6

Stéphane Airiau

Université Paris-Dauphine

Date & Time

Le temps **absolu**

- il y a 86 400 secondes dans une journée
- l'horloge doit correspondre à l'horloge officielle (exactement à 12h, extrêmement proche le reste du temps !)
- extrêmement proche est défini d'une façon précise
- un `Instant` est un moment donné
- une origine a été fixée de manière arbitraire (1ier janvier 1970) (the epoch)
- un `Instant` est donc défini par le temps qui s'est écoulé depuis l'origine jusqu'au moment donné (avec une précision de l'ordre de la nano seconde).
- on peut utiliser `equals` et `compareTo` pour comparer deux instants.
- la méthode `static Instant.now()` donne l'instant présent

exemple : mesurer le temps d'un algorithme

- `Duration` mesure le temps écoulé entre deux instants.
- On peut obtenir la durée avec les unités usuelles en appelant les méthodes
 - `toNanos`
 - `toMillis`
 - `toSeconds`
 - `toMinutes`
 - `toHours`
 - `toDays`

```
1 Instant start = Instant.now();
2 runAlgorithm();
3 Instant end = Instant.now();
4 Duration timeElapsed = Duration.between(start, end);
5 long millis = timeElapsed.toMillis();
```

En interne, une durée utilise

- le nombre de secondes est stocké dans un **long**
 - le nombre de nanosecondes est stocké dans un **int**
- ➡ si on n'a pas besoin de la précision à la nano seconde, on peut juste appeler `toNanos` et faire les calculs à l'aide de **long**.

Méthode	Description
<code>plus, minus</code>	ajoute une durée à un instant ou une autre durée (idem pour soustraire)
<code>plusNanos, plusMillis, plusSeconds, etc..</code>	ajoute un certain nombre d'unités de temps à un instant ou une durée
<code>minusNanos, minusMillis, minusSeconds, etc..</code>	soustrait un certain nombre d'unités de temps à un instant ou une durée
<code>multipliedBy, negated</code>	multiplier/diviser une durée (pas un instant)
<code>isZero, isNegative</code>	vérifie si la durée est zéro ou si elle est négative

Le temps "local"

- le temps peut être défini avec un fuseau horaire ou non.

LocalDate API

Le temps "local"

- le temps peut être défini avec un fuseau horaire ou non.
- Est-ce que July 16, 1969, 09 :32 :00 définit un moment précis dans l'histoire ?

LocalDate API

Le temps "local"

- le temps peut être défini avec un fuseau horaire ou non.
- Est-ce que July 16, 1969, 09 :32 :00 définit un moment précis dans l'histoire ?
- Apollo 11 a décollé, mais c'est imprécis ! cela indique 24 moments différents !

LocalDate API

Le temps "local"

- le temps peut être défini avec un fuseau horaire ou non.
- Est-ce que July 16, 1969, 09 :32 :00 définit un moment précis dans l'histoire ?
- Apollo 11 a décollé, mais c'est imprécis ! cela indique 24 moments différents !
- July 16, 1969, 09 :32 :00 EDT précise le fuseau horaire, on sait exactement quand ça s'est passé

LocalDate API

Le temps "local"

- le temps peut être défini avec un fuseau horaire ou non.
- Est-ce que July 16, 1969, 09 :32 :00 définit un moment précis dans l'histoire ?
- Apollo 11 a décollé, mais c'est imprécis ! cela indique 24 moments différents !
- July 16, 1969, 09 :32 :00 EDT précise le fuseau horaire, on sait exactement quand ça s'est passé
- `LocalDate` `LocalTime` temps sans fuseaux horaires

`LocalDate` API

Le temps "local"

- le temps peut être défini avec un fuseau horaire ou non.
- Est-ce que July 16, 1969, 09 :32 :00 définit un moment précis dans l'histoire ?
- Apollo 11 a décollé, mais c'est imprécis ! cela indique 24 moments différents !
- July 16, 1969, 09 :32 :00 EDT précise le fuseau horaire, on sait exactement quand ça s'est passé
- `LocalDate` `LocalTime` temps sans fuseaux horaires

```
1 | LocalDate today = LocalDate.now(); // aujourd'hui
2 | LocalDate bastilleDay = LocalDate.of(1789, 7, 14);
3 | bastilleDay = LocalDate.of(1903, Month.JULY, 14);
```

`LocalDate` API

La durée entre deux instants "locaux" `Period`

```
LocalDate d1 = birthday.plus(Period.ofYears(1))  
LocalDate d2 = birthday.plus(Duration.ofDays(365))
```

Il est possible que `d1` soit différent de `d2`

```
independenceDay.until(christmas, ChronoUnit.DAYS) // 174 days
```

Il existe aussi les classes `MonthDay`, `YearMonth` et `Year` pour représenter des dates "partielles".

Si on veut représenter le 25 décembre sans mentionner l'année, on peut utiliser la classe `YearMonth`.

On peut demander la date du prochain

- jour de la semaine `next (weekday)`, `previous (weekday)`
- quel est le jour du jour n dans le mois
`dayOfWeekInMonth (n, weekday)`
- `firstDayOfMonth ()`, `firstDayofNextMonth ()`,
`firstDaysOfNextYear () ...`

Temps avec fuseau horaire

Base de données officielle des fuseaux horaires www.iana.org/time-zones

chaque fuseau horaire en Java possède un identifiant

`ZoneId.getAvailableIds()`

```
ZonedDateTime apollo11launch =  
    ZonedDateTime.of(1969, 7, 16, 9, 32, 0, 0,  
                    ZoneId.of("America/New_York"));  
//1969-07-16T09:32-04:00[America/New_York]
```

les méthodes de `ZonedDateTime` vont prendre en compte les heures d'été.

ZonedDateTime API

Utiliser les normes pour afficher un temps

`DateTimeFormatter`

`DateTimeFormatter`

Ces normes sont utilisées pour que des machines/programmes puissent lire des données

Pour les humains, on a 4 styles SHORT, MEDIUM, LONG, FULL.

Style	Date	Time
SHORT	7/16/69	9:32 AM
MEDIUM	Jul 16, 1969	9:32:00 AM
LONG	July 16, 1969	9:32:00 AM EDT
FULL	Wednesday, July 16, 1969	9:32:00 AM EDT

```
DateTimeFormatter formatter =  
    DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG);  
String formatted = formatter.format(apollo11launch);  
// July 16, 1969 9:32:00 AM EDT
```

Et comment écrire mercredi 16 juillet 1969 pour les français ?

↳ Locale

- on spécifie le langage (chaque pays a un code zh, du, en, fr, it, tr, es, etc ...)
- on peut spécifier un script en option
 - Latn pour latin
 - Cyril pour cyrillic
 - Hant pour les caractères chinois
- on peut spécifier le pays, en utilisant les mêmes codes mais en majuscule
- et deux autres options

```
Locale usEnglish = Locale.forLanguageTag("en-US");
```


Locale prédéfinies

Locale.CANADA
Locale.CANADA_FRENCH
Locale.CHINA
Locale.FRANCE
Locale.GERMANY
Locale.ITALY
Locale.JAPAN
Locale.KOREA
Locale.PRC
Locale.TAIWAN
Locale.UK
Locale.US

Certaines spécifient juste le langage

Locale.CHINESE
Locale.ENGLISH
Locale.FRENCH
Locale.GERMAN
Locale.ITALIAN
Locale.JAPANESE
Locale.KOREAN
Locale.SIMPLIFIED_CHINESE
Locale.TRADITIONAL_CHINESE

formater des nombres : NumberFormat

```
Locale loc = Locale.GERMANY;  
NumberFormat formatter = NumberFormat.getCurrencyInstance(loc);  
double amt = 123456.78;  
String result = formatter.format(amt);
```

Le résultat est

123.456,78 €

```
String input = ...;  
NumberFormat formatter = NumberFormat.getNumberInstance();  
// Get the number formatter for default format locale  
Number parsed = formatter.parse(input);  
double x = parsed.doubleValue();
```

La méthode lancera l'exception `ParseException` si le nombre n'est pas écrit suivant le bon format.

Les monnaies

Currency class

Monnaie	Identifiant
US Dollar	USD
Euro	EUR
Livre Sterling	GBP
Yen	JPY

```
NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.US);  
formatter.setCurrency(Currency.getInstance("EUR"));  
System.out.println(formatter.format(euros));
```