

TP JSR 310: Date and Time API

Objectifs

- Utiliser la bonne norme de calendrier ISO
- Différencier temps humain et temps machine
- Différencier les classes locales et de zone
- Différencier les périodes et la durée
- Créer des dates locales
- Calculer des dates dans le futur et le passé
- Calculer le temps entre deux dates
- Formater les dates
- Convertir les anciens objets Date et GregorianCalendar

Introduction

L'API Java Date-Time utilise la norme ISO 8601 pour représenter les dates et les heures. ISO 8601 vise à normaliser les différences dans les formats de date/heure à travers le monde et est basé sur le calendrier grégorien qui a été introduit en 1582. Le format de base est le suivant:

YYYY-MM-DDThh:mm:ss

L'heure est représentée au format 24 heures. Bien que ISO 8601 soit le format par défaut, l'API Date-Heure nous donne plusieurs façons de formater les dates et les heures selon les besoins de notre application. Nous en verrons des exemples ici.

L'API Java Date-Time a deux manières de représenter l'heure : l'heure humaine et l'heure machine.

Le temps humain nous est familier. Il se compose d'unités telles que les années, les mois, les jours, les heures, les minutes et secondes. Comme son nom l'indique, le temps humain est l'heure normale du calendrier et de l'horloge.

Le temps machine est un peu différent. Le temps machine est une chronologie (mesurée en nanosecondes) qui commence le 1er janvier 1970, qui est connu comme **epoch**. Nombres positifs représentent le temps après epoch, et les nombres négatifs représentent le temps avant epoch.

Il existe des méthodes utilitaires qui convertissent le temps machine en temps humain, et vice versa.

LocalDate

Les objets `LocalDate` sont créés à l'aide de l'une des méthodes de factory fournies plutôt qu'à l'aide du constructeur `LocalDate`. Une factory method est simplement une méthode qui crée un nouvel objet pour nous.

Les deux factory méthodes `LocalDate` les plus couramment utilisées sont **`now()`** et **`parse()`**.


Créons un nouvel objet `LocalDate` contenant la date d'aujourd'hui.

Créer un nouveau projet java nommé `DateAndTime` comme suit

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.



Project name:

☒ Use default location

Location:

JRE

☐ Use an execution environment JRE:

☐ Use a project specific JRE:

☒ Use default JRE 'jdk-16.0.1' and workspace compiler preferences

[Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files


☒ Create separate folders for sources and class files

[Configure default...](#)

Working sets

☐ Add project to working sets

Working sets:



Ne pas créer de module
Cliquer sur don't create ou ne pas créer

New module-info.java

Create module-info.java

Enter a module name.

Module name:

☐ Generate comments (configure templates and default value [here](#))

Don't Create


Create

Cliquer droit sur src et créer un package launcher

New Java Package

Java Package

Create a new Java package.



Creates folders corresponding to packages.

Source folder:

DateAndTime/src


Browse...

Name:

launcher

☐ Create package-info.java

☐ Generate comments (configure templates and default value [here](#))



Cancel


Finish

Cliquer droit sur le package et créer une classe DateTester avec une méthode main

New Java Class

Java Class

Create a new Java class.



Source folder:

DateAndTime/src

Browse...

Package:

launcher

Browse...

☐ Enclosing type:

Browse...

Name:

DateTester

Modifiers:

☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

java.lang.Object

Browse...

Interfaces:

Add...

Remove

Which method stubs would you like to create?

☒ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments



Cancel

Finish

Le projet est prêt on va pouvoir commencer

Date courante et date d'après une String: now() et parse()

Ajouter le code suivant

```
LocalDate ld = LocalDate.now();  
System.out.println(ld);
```

Comme suit

```
1 package launcher;  
2  
3 import java.time.LocalDate;  
4  
5 public class DateTester {  
6  
7     public static void main(String[] args) {  
8         LocalDate ld = LocalDate.now();  
9         System.out.println(ld);  
10  
11     }  
12  
13 }
```

Nous avons fait appel à `LocalDate.now()`,

Ajoutons le code suivant:

```
12         ld = LocalDate.parse("2021-03-01");  
13         System.out.println(ld);
```

Ajoutons deux nouveaux exemples. Dans cet exemple, nous créons un nouvel objet `LocalDate` contenant la date représentée par une chaîne ISO 8601 bien formée, qui est passée dans la méthode `parse()` du `LocalDate`.

Avec une date non ISO

```
16         ld = LocalDate.parse("02/07/2021", DateTimeFormatter.ofPattern(  
17             "MM/dd/yyyy"));  
18         System.out.println(ld);
```

Notez que cette version de la méthode `parse()` nécessite à la fois la chaîne de date et un paramètre spécifiant le modèle de la date d'arrivée.

Pour ce faire, nous utilisons la méthode statique **`ofPattern()`** sur le **`DateTimeFormatter`**.

Parce que le code `DateTimeFormatter` est dans un package différent de `LocalDate`, nous devons importer `java.time.format.DateTimeFormatter` pour l'utiliser.

Lorsque nous exécutons la liste, vous pouvez voir à partir de la sortie qui a été affichée en ISO alors que notre date saisie était 02/07/2021
2021-02-07.

Pour convertir une date en `String`, il suffit d'appeler la méthode `toString()` sur l'objet `LocalDate`. En utilisant nos valeurs de la liste précédente, ce serait aussi aussi simple que ça :

```
String isoDate = ld.toString();
```

Formatage des dates

Bien que ISO 8601 soit le format de date par défaut pour l'API Java Date-Heure, les dates peuvent être affichées dans une variété de formats. Pour ce faire, nous utilisons une combinaison de **`LocalDate.format()`** et la classe **`DateTimeFormatter`**.

Vous avez vu la classe `DateTimeFormatter` utilisée pour afficher une date depuis le format `MM/jj/aaaa` à l'aide de la méthode `parse()` de l'objet `LocalDate`. Ce qui suit aussi formate une `LocalDate` en une chaîne avec le modèle de date `MM/dd/yyyy` :

```
22     DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yyyy");
23     LocalDate ld2 = LocalDate.parse("02/07/2021", formatter);
24     String formatted = ld.format(formatter);
25     System.out.println(ld2);
26     System.out.println(formatted);
```

On utilise un `DateTimeFormatter` est appelé ici *formatter*. Il s'agit d'une mise en forme ou d'un modèle qui peut être utilisé pour formater notre date.

Le `DateTimeFormatter` sera ensuite transmis à la méthode **`parse()`** de `LocalDate` avec une valeur de date pour créer un `LocalDate` dans la deuxième ligne. Dans la dernière ligne, nous affectons la valeur renvoyée par l'appel de la méthode `format()` sur notre objet `LocalDate` (*ld2*) à notre chaîne formatée, *formatted*.

En conséquence, la valeur de formaté ressemble à ceci :
02/07/2021

Jouer avec les formats

A vous de jouer: donner les sorties suivantes:

Starting date: 2020-12-25

12=25=2020+=+=+=

=> 12/2020 <==

2020-25-12-25-2020

Utilisation de la localisation

Un autre exemple qui mérite d'être examiné, implique **ofLocalizedDate ()** du **DateTimeFormatter** que nous utilisons. Cette méthode utilise la localisation informations du système pour déterminer comment la date doit être formatée. le le format d'utilisation de la méthode est le suivant :

```
Id.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL));
```

Dans ce cas, un style prédéfini est passé à la méthode ofLocalizedData(). le le style prédéfini est une énumération **FormatStyle**. Dans ce cas, FormatStyle.FULL. Lorsqu'elle est appelée, la ligne précédente donnera une valeur de date qui ressemble à ceci (selon les informations de localisation sur votre ordinateur, cela peut sembler différent-ent pour vous)

Tester les scenarii suivants avec FULL MEDIUM et SHORT pour avoir la sortie suivante:

```
vendredi 25 décembre 2020
25 décembre 2020
25 déc. 2020
25/12/2020
```

```
11      LocalDate ld = LocalDate.parse("2020-12-25");
12      System.out.println("Starting date: " + ld);
13
14      String formatted = ld.format(
15      DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL));
16      System.out.println(formatted);
17
18      System.out.println(
19      ld.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.LONG)));
20
21      System.out.println(
22      ld.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.
23      MEDIUM)));
24
25      System.out.println(
26      ld.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT)));
27
```

Mois, jour année

Les différentes méthode permettent d'extraire chaque type d'information d'une date voir le tableau suivant:

now, of	These static methods construct a <code>LocalDate</code> , either from the current time or from a given year, month, and day.
plusDays, plusWeeks, plusMonths, plusYears	Adds a number of days, weeks, months, or years to this <code>LocalDate</code> .
minusDays, minusWeeks, minusMonths, minusYears	Subtracts a number of days, weeks, months, or years from this <code>LocalDate</code> .
plus, minus	Adds or subtracts a Duration or Period.
withDayOfMonth, withDayOfYear, withMonth, withYear	Returns a new <code>LocalDate</code> with the day of month, day of year, month, or year changed to the given value.
getDayOfMonth	Gets the day of the month (between 1 and 31).
getDayOfYear	Gets the day of the year (between 1 and 366).
getDayOfWeek	Gets the day of the week, returning a value of the <code>DayOfWeek</code> enumeration.
getMonth, getMonthValue	Gets the month as a value of the <code>Month</code> enumeration, or as a number between 1 and 12.
getYear	Gets the year, between -999,999,999 and 999,999,999.
until	Gets the <code>Period</code> , or the number of the given <code>ChronoUnits</code> , between two dates.
isBefore, isAfter	Compares this <code>LocalDate</code> with another.
isLeapYear	Returns true if the year is a leap year—that is, if it is divisible by 4 but not by 100, or divisible by 400. The algorithm is applied for all past years, even though that is historically inaccurate. (Leap years were invented in the year -46, and the rules involving divisibility by 100 and 400 were introduced in the Gregorian calendar reform of 1582. The reform took over 300 years to become universal.)

Amusons-nous à extraire le jour ou le mois de naissance de Beyoncé
 Nous pouvons l'
 afficher et italien ou en allemand

Écrire le code suivant:

```

146     LocalDate dateNaisBeyonce=LocalDate.of(1981,9,4);
147     System.out.println(dateNaisBeyonce);
148     System.out.println(dateNaisBeyonce.getMonth());
149     System.out.println(dateNaisBeyonce.getMonth().getDisplayName(TextStyle.FULL, Locale.GERMAN));
150 //
151 //
152     LocalDate isNow =LocalDate.now();
153     System.out.println(isNow);
154     System.out.println(isNow.getDayOfWeek().getDisplayName(TextStyle.FULL, Locale.ITALIAN));

```

Cela donne

```

1981-09-04
SEPTEMBER
September
2021-06-02
mercoledì

```

LocalDateTime

Nous nous sommes concentrés sur LocalDate jusqu'à présent ; cependant, parfois, vous voudrez aussi avoir besoin à la fois la date et l'heure en utilisant un LocalDateTime objet. Pour la plupart, cet objet fonctionne comme l'objet LocalDate, sauf que vous obtenir également des informations sur l'heure.

```

10     LocalDateTime ldt = LocalDateTime.now();
11     System.out.println(ldt);
12
13     String formatted =
14         ldt.format(DateTimeFormatter.ofPattern("yyyy-MM-dd hh:mm:ss"));
15     System.out.println(formatted);

```

Si vous ne vouliez que l'heure, sans la date, alors vous pouvez utiliser la date-TimeFormatter avec un modèle tel que hh:mm:ss.

Calcul de dates

LocalDate fournit un certain nombre de méthodes qui vous permettent d'avancer une date dans le futur ou en arrière dans le passé. Cela peut être fait en ajoutant ou en soustrayant des années, des mois, semaines ou jours à une date. La variété de méthodes fournies par LocalDate inclue:

```
plusYears()  
plusMonths()  
plusWeeks()  
plusDays()  
minusYears()  
minusMonths()  
minusWeeks()  
minusDays()
```

Pour utiliser ces méthodes, transmettez simplement le nombre d'années, de mois ou de jours que vous voulez ajouter ou soustraire de la date existante, et la méthode renverra un nouveau LocalDate. C'est aussi simple qu'il y paraît.

Exemple : soustraire (déplace remonter dans le temps) huit jours à compter de la date *ld*.

```
LocalDate past = ld.minusDays(8);
```

Essayer

A vous de jouer

Avec utilisation simple de la méthode `plusYear()` afficher quel jour de la semaine le jour de l'An sera pour chacune des 10 prochaines années.

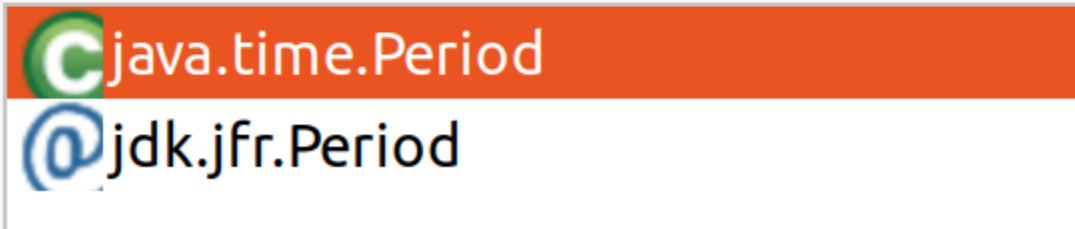
La différence entre deux dates

LocalDate fournit également un moyen facile de déterminer la durée entre deux dates via la méthode **until()**.

La méthode jusqu'à **until()** compare deux dates et renvoie un objet représentant la différence de temps entre les deux dates.

Testons un exemple d'utilisation d'une variable `LocalDate` et de comparaison avec une autre date en utilisant la Classe **period** pour stocker la différence. Pour utiliser la classe `Period`, vous devez l'importer

Choose type to import:



The image shows an IDE's 'Choose type to import' dialog. It features a search bar at the top. Below it, two suggestions are listed: 'java.time.Period' with a green 'C' icon, and 'jdk.jfr.Period' with a blue '@' icon. The 'java.time.Period' option is highlighted with an orange background.

Saisissez le code suivant

```
22      LocalDate ld5 = LocalDate.now();
23      LocalDate otherDate = LocalDate.parse("2022-01-01");
24      Period diff = ld5.until(otherDate);
25      System.out.println("Starting date: " + ld5);
26      System.out.println("Other date:" + otherDate);
27      System.out.println("=====");
28      System.out.println("Difference: " + diff );
29
```

vous pouvez voir que la différence est de `PansYmoisMjoursD` ;
cependant, ces informations sont présentées d'une manière un peu cryptique. Heureusement, la classe `Period` a des méthodes `getter` pour les années, mois et jours qui composent le décalage horaire entre les deux dates.

```
getYears()
getMonths()
getDays()
```

Ajouter le code suivant:

```

31 System.out.println("Years: " + diff.getYears());
32 System.out.println("Months: " + diff.getMonths());
33 System.out.println("Days: " + diff.getDays());

```

Convertir les anciennes dates

Dans la dernière section de ce TP, nous examinerons la conversion de l'héritage Date et GregorianCalendar aux objets LocalDate. Ces deux types d'objets date ont été utilisés dans les anciennes versions de Java, nous pourrions donc les rencontrer.

La conversion d'un objet Date hérité en un objet LocalDate implique deux étapes. D'abord, vous devez convertir la Date en un objet ZonedDateTime et de là en un LocalDate.

La première étape (conversion de la date en ZonedDateTime) se compose de plusieurs éléments.

1. Nous convertissons la Date en Instant. Essentiellement, nous convertissons la date de temps humain en temps machine.
2. Nous convertissons ensuite l'instant dérivé de la date héritée en un ZonedDateTime objet en utilisant la méthode statique **ofInstant()**.

Nous devons également passer un identifiant de fuseau horaire dans la méthode ofInstant(), nous utilisons donc le système par défaut de la machine sur laquelle le code s'exécute.

La deuxième étape est plus simple. Nous appelons simplement la méthode toLocalDate() de notre objet ZonedDateTime.

Coder comme suit

```

12 LocalDate ld;
13 Date legacyDate = new Date();
14 // Step 1
15 ZonedDateTime zdt = ZonedDateTime.ofInstant(
16     legacyDate.toInstant(), ZoneId.systemDefault());
17 // Step 2
18 ld = zdt.toLocalDate();
19 System.out.println(legacyDate);
20 System.out.println(ld);

```

Pour finir on peut convertir GregorianCalendar

```
24    LocalDate ld2;  
25    GregorianCalendar legacyCalendar = new GregorianCalendar();  
26    // Step 1  
27    ZonedDateTime zdt2 = legacyCalendar.toZonedDateTime();  
28    // Step 2  
29    ld2 = zdt2.toLocalDate();  
30    System.out.println(legacyCalendar);  
31    System.out.println(ld2);  
32
```

Bravo!