Gestion du temps en Java

La JSR 310, un instant, une date, une heure, une date/heure, une time-zone, un intervalle de temps

Voir aussi les urls : https://blog.ippon.fr/2014/03/19/java-8-gestion-du-temps/
https://www.baeldung.com/java-8-date-time-intro, https://www.baeldung.com/java-period-duration

1 Introduction

La gestion du temps est un problème est complexe. Travailler en base 60 pour les secondes et les minutes puis en base 24 pour les heures n'est pas très simple. Mais la palme revient à la gestion des mois du calendrier qui n'ont pas tous le même nombre de jours, voire pire puisque certains mois ont un nombre de jours variable suivant les années. Les ordinateurs utilisent une technique différente, en ne travaillant pas directement avec des dates et heures mais en nombre de secondes ou de millisecondes depuis une date de référence (généralement le 1er janvier 1970 à 0 heure). Ce mode de représentation n'est cependant pas très pratique pour un humain : la valeur 61380284400000L n'est pas très évocatrice, par contre 25/12/2019 est beaucoup plus parlant.

2 La JSR 310 depuis Java 8

En Java7, avec les classes java.util.Date, java.text.DateFormat, Calendar, GregorianCalendar, d'importantes fonctionnalités sont possibles mais leur utilisation relève parfois du casse-tête pour le développeur. D'où la JSR310 a été implémentée dans le Java SE depuis Java 8. La gestion du temps a été complètement repensée avec l'apparition de plus nombreuses classes qu'auparavant dans des nouveaux packages.

3 Les packages pour le temps

nom du package	explications
java.time	The main API for dates, times, instants, and durations.
java.time.chrono	Generic API for calendar systems other than the default ISO.
java.time.format	Provides classes to print and parse dates and times.
java.time.temporal	Access to date and time using fields and units, and date time adjusters.
java.time.zone	Support for time-zones and their rules.

4 Quelques classes du package java.time

nom de la classe	**explications**
LocalDate	Représente une date (jour/mois/année) sans heure
LocalDateTime	Représente une date et une heure sans prise en compte du fuseau horaire.
LocalTime	Représente une heure sans prise en compte du fuseau horaire.
OffsetDateTime	Représente une date-heure avec le décalage /UTC.
OffsetTime	Représente une heure avec le décalage /UTC.
ZonedDateTime	Représente une date-heure avec le fuseau horaire correspondant.
Duration	Représente une durée exprimée en heures minutes secondes.
Period	Représente une durée exprimée en jours mois années.
MonthDay	Représente un jour et un mois sans année.
YearMonth	Représente un mois et une année sans jour.
…	…

5 Quelques méthodes

Toutes ces classes proposent une série de méthodes permettant la manipulation de leurs éléments. Ces méthodes respectent une convention de nommage facilitant l'identification de leur usage.

5.1 of

of retourne une instance de la classe initialisée avec les différentes valeurs passées comme paramètres.

```
LocalDate noel;
noel = LocalDate.of(2019, 12,25);
```

5.2 parse

parse transforme la chaîne de caractères passée comme paramètre vers le type correspondant.

```
LocalTime horloge;
horloge = LocalTime.parse("22:45:03");
```

5.3 withXxxx

withXxxx retourne une nouvelle instance en **imposant** la composante indiquée par Xxxx par la valeur passée comme paramètre : withDayOfMonth, withMonth, withYear, withHour, ...

```
LocalTime horloge;
horloge = LocalTime.parse("22:45:03");

LocalTime nouvelleHeure;
nouvelleHeure = horloge.withHour(9);
```

5.4 plusXxxx et minusXxxx

plusXxxx et minusXxxx retournent une nouvelle instance de la classe après **ajout** ou **retrait** du nombre d'unités indiqué par le paramètre.
Xxxx indique ce qui est ajouté ou retranché.

```
LocalDate paques;
paques = LocalDate.of(2019,4,20);

LocalDate ascension;
ascension = paques.plusDays(39);
```

5.5 from

from permet la **conversion** entre les différents types. En cas de conversion vers un type moins complet, il y a perte d'informations.

```
LocalDateTime maintenant;
maintenant = LocalDateTime.now();

// transformation en LocalDate avec perte de l'heure
LocalDate aujourdhui;
aujourdhui = LocalDate.from(maintenant);
```

5.6 atXxxx

atXxxx : fusionne l'objet reçu comme paramètre avec l'objet courant et retourne le résultat de cette association. On peut par exemple fusionner un LocalDate et un LocalTime pour obtenir un objet LocalDateTime.

```
LocalDate jourMatch;
```

```
jourMatch = LocalDate.of(2019,7,14);

LocalTime heureMatch;
heureMatch = LocalTime.of(21,00);

LocalDateTime fin;
fin = jourMatch.atTime(heureMatch);
```

5.7 Calcul sur les dates

```
Period dif = Period.between(date1,date2);
```

5.8 Un exemple complet

Dans l'exemple suivant, on compte le nombre de jours fériés tombant un samedi ou un dimanche.

```
MonthDay[] fetes;
        fetes=new MonthDay[8];
        fetes[0]=MonthDay.of(1,1);
        fetes[1]=MonthDay.of(5,1);
        fetes[2]=MonthDay.of(5,8);
        fetes[3]=MonthDay.of(7,14);
        fetes[4]=MonthDay.of(8,15);
        fetes[5]=MonthDay.of(11,1);
        fetes[6]=MonthDay.of(11,11);
        fetes[7]=MonthDay.of(12,25);

        int nbJours;
        int annee;
        LocalDate jourTest;
        for (annee=2015;annee<2030;annee++)
        {
             nbJours=0;
             for(MonthDay test:fetes)
             {
                  jourTest=test.atYear(annee);
                  if (jourTest.getDayOfWeek() == DayOfWeek.SATURDAY
||jourTest.getDayOfWeek()== DayOfWeek.SUNDAY)
                  {
                        nbJours++;
                  }
             }
             System.out.println("en " + annee + " il y a " + nbJours
+ " jour(s) ferie(s) un samedi ou un dimanche");
         }
```