Date et heure

La gestion de date et d'heure a longtemps été la bête noire des développeurs Java. La classe GregorianCalendar était disponible pour répondre aux problèmes de manipulation de date et d'heure. De nombreuses fonctionnalités étaient prévues mais leur utilisation relevait parfois du casse-tête. Il est vrai que le 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 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 61380284400000 n'est pas très évocatrice, par contre 25/12/2014 est beaucoup plus parlant. C'est pourquoi de nombreuses fonctions permettent le passage d'un format à l'autre.

Depuis la version 8 de Java, la gestion des dates et des heures a été complètement repensée. Au lieu de n'avoir qu'une ou deux classes dédiées à cette gestion et avec lesquelles il fallait jongler, de nombreuses classes spécialisées ont fait leur apparition.

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 et une heure avec le décalage UTC.
OffsetTime	Représente une heure avec le décalage UTC.
ZonedDateTime	Représente une date et une 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.

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.

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

```
LocalDate noel;
noel=LocalDate.of(2014, 12,25);
```

- from : 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);
```

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

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

- withxxxxxx : retourne une nouvelle instance en modifiant la composante indiquée par xxxxx par la valeur passée comme paramètre.

```
LocalTime horloge;
horloge=LocalTime.parse("22:45:03");
LocalTime nouvelleHeure;
nouvelleHeure=horloge.withHour(9);
```

- plusxxxxx et minusxxxx : retourne une nouvelle instance de la classe après ajout ou retrait du nombre d'unités indiqué par le paramètre. xxxxxx indique ce qui est ajouté ou retranché.

```
LocalDate paques;
paques=LocalDate.of(2014,4,20);
LocalDate ascension;
ascension=paques.plusDays(39);
```

- atxxxxxx : combine l'objet reçu comme paramètre avec l'objet courant et retourne le résultat de cette association. On peut par exemple combiner un objet LocalDate et un objet LocalTime pour obtenir un objet LocalDateTime.

```
LocalDate jourMatch;
jourMatch=LocalDate.of(2014,7,13);

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

Support exclusif ne peut être utilisé autrement que par du personnel NEEDEMAND

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

Le petit exemple de code ci-dessous illustre quelques opérations sur les dates en comptant 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=2014;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");
        }
```

Support exclusif ne peut être utilisé autrement que par du personnel NEEDEMAND