Les dates

Les dates et le temps sont représentés en Java par des classes. Cependant, au fil des versions de l'API standard, de nouvelles classes ont été proposées pour représenter les dates et le temps. Pour des raisons de compatibilité ascendante, les anciennes classes sont demeurées même si une partie de leurs méthodes ont été dépréciées. Ainsi, il existe plusieurs façons de représenter les dates et le temps en Java.

Date : la classe historique

La classe java.util.Date est la première classe a être apparue pour représenter une date. Elle comporte de nombreuses limitations :

- Il n'est pas possible de représenter des dates antérieures à 1900
- Elle ne supporte pas les fuseaux horaires
- Elle ne supporte que les dates pour le calendrier grégorien
- Elle ne permet pas d'effectuer des opérations (ajout d'un jour, d'une année…)

La quasi totalité des constructeurs et des méthodes de cette classe ont été dépréciés. Cela signifie qu'il ne faut pas les utiliser. Pourtant, la classe Date reste une classe largement utilisée en Java. Par exemple, la classe Date est la classe mère des classes java.sql.Date, java.sql.Time et java.sql.Timestamp qui servent à représenter les types du même nom en SQL. Beaucoup de bibliothèques tierces utilisent directement ou indirectement la classe Date.

```
package com.cgi.udev;

import java.time.Instant;
import java.util.Date;

public class TestDate {

  public static void main(String[] args) {
    Date aujourdhui = new Date(); // crée une date au jour et à l'heure d'aujourd'hui
    Date dateAvant = new Date(0); // crée une date au 1 janvier 1970 00:00:00.000

    System.out.println(aujourdhui.after(dateAvant)); // true
    System.out.println(dateAvant.before(aujourdhui)); // true
```

```
    System.out.println(aujourdhui.equals(dateAvant)); // false
    System.out.println(aujourdhui.compareTo(dateAvant)); // 1

    Instant instant = aujourdhui.toInstant();
  }

}
```

Actuellement, les méthodes permettant de comparer des instances de Date ne sont pas dépréciées. Pour construire une instance de Date, on peut utiliser le constructeur sans paramètre pour créer une date au jour et à l'heure d'aujourd'hui. On peut également passer un nombre de type **long** représentant le nombre de millisecondes depuis le 1er janvier 1970 à 00:00:00.000 (*l'epoch*). Il est également possible de modifier une instance de Date avec la méthode Date.setTime en fournissant le nombre de millisecondes depuis le 1er janvier 1970 à 00:00:00.000.

> Pour connaître le nombre de millisecondes écoulées depuis le 1er janvier 1970 à 00:00:00.000, il faut appeler la méthodes System.currentTimeMillis.

```
long nbMilliSecondes = System.currentTimeMillis();
```

> Depuis l'introduction en Java 8 de la nouvelle API des dates, il est possible de convertir une instance de Date en une instance de Instant avec la méthode Date.toInstant.

Calendar

Depuis Java 1.1, la classe java.util.Calendar a été proposée pour remplacer la classe java.util.Date. La classe Calendar pallie les nombreux désavantages de la classe Date :

- Il est possible de représenter toutes les dates (y compris avant notre ère)
- La classe Calendar supporte les fuseaux horaires (*timezone*)
- La classe Calendar est une classe abstraite pour laquelle il est possible de fournir des classes concrètes pour gérer différents calendriers (le JDK ne propose que la classe GregorianCalendar comme implémentation concrète pour le calendrier grégorien).

```
package com.cgi.udev;

import java.util.Calendar;
import java.util.Locale;
import java.util.TimeZone;

public class TestCalendar {

  public static void main(String[] args) {
    // Date et heure d'aujourd'hui en utilisant le fuseau horaire du système
    Calendar date = Calendar.getInstance();
    // Date et heure d'aujourd'hui en utilisant le fuseau horaire de la France
    Calendar dateFrance = Calendar.getInstance(Locale.FRANCE);
    // Date et heure d'aujourd'hui en utilisant le fuseau horaire GMT
    Calendar dateGmt = Calendar.getInstance(TimeZone.getTimeZone("GMT"));

    // On positionne la date au 8 juin 2005 à 12:00:00
    date.set(2005, 6, 8, 12, 0, 0);

    System.out.println(date.toInstant());

    // On ajoute 24 heures à la date et on passe au jour suivant
    date.add(Calendar.HOUR, 24);
    // On décale la date de 12 mois sans passer à l'année suivante
    date.roll(Calendar.MONTH, 12);
    System.out.println(date.toInstant()); // 9 juin 2005 à 12:00:00

  }

}
```

Comme pour les instances de Date, il est possible de comparer les instances de Calendar entre elles. Il est également possible de convertir une instance de Calendar en Date (mais alors on perd l'information du fuseau horaire puisque la classe Date ne contient pas cette information) grâce à la méthode Calendar.getTime. Enfin, on utilise la méthode Calendar.toInstant pour convertir une instance de Calendar en une instance de Instant.

Même si la classe Calendar est beaucoup plus complète que la classe Date, son utilisation est restée limitée car elle est également plus difficile à manipuler. Son API la rend assez fastidieuse

d’utilisation. Elle ne permet pas de représenter simplement la notion du durée. Et surtout, comme il s’agit d’une classe abstraite, il n’est pas possible construire une instance avec l’opérateur **new**. Il faut systématiquement utiliser une des méthodes de classes Calendar.getInstance.

L’API Date/Time

Depuis Java 8, une nouvelle API a été introduite pour représenter les dates, le temps et la durée. Toutes ces classes ont été regroupées dans la package java.time.

Les Dates

Les classes LocalDate, LocalTime et LocalDateTime permettent de représenter respectivement une date, une heure, une date et une heure.

```
package com.cgi.udev;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
import java.time.temporal.ChronoUnit;

public class TestTime {

  public static void main(String[] args) {
    LocalDate date = LocalDate.of(2005, Month.JUNE, 5); // 05/06/2005
    date = date.plus(1, ChronoUnit.DAYS); // 06/06/2005
    LocalDateTime dateTime = date.atTime(12, 00); // 06/06/2005 12:00:00
    LocalTime time = dateTime.toLocalTime(); // 12:00:00

    time = time.minusHours(2); // 10:00:00
  }

}
```

On peut facilement passer d’un type à une autre. Par exemple la méthode LocalDate.atTime permet d’ajouter une heure à une date, créant ainsi une instance de LocalDateTime. Toutes les instances de ces classes sont immutables.

Si on veut avoir l'information de la date ou de l'heure d'aujourd'hui, on peut créer une instance grâce à la méthode *now*.

```
LocalDate dateAujourdhui = LocalDate.now();
LocalTime heureMaintenant = LocalTime.now();
LocalDateTime dateHeureMaintenant = LocalDateTime.now();
```

Une instance de ces classes ne contient pas d'information de fuseau horaire. On peut néanmoins passer en paramètre des méthodes *now* un ZoneId pour indiquer le fuseau horaire pour lequel on désire la date et/ou l'heure actuelle.

```
LocalDate dateAujourdhui = LocalDate.now(ZoneId.of("GMT"));
LocalTime heureMaintenant = LocalTime.now(ZoneId.of("Europe/Paris"));
LocalDateTime dateHeureMaintenant = LocalDateTime.now(ZoneId.of("America/New_York"));
```

> Si vous avez besoin de représenter des dates avec le fuseau horaire, alors il faut utiliser la classe ZonedDateTime.

Les classe Year et YearMonth permettent de manipuler les dates et d'obtenir des informations intéressantes à partir de l'année ou du mois et de l'année.

```
package com.cgi.udev;

import java.time.LocalDate;
import java.time.Month;
import java.time.Year;
import java.time.YearMonth;

public class TestYear {

  public static void main(String[] args) {
    Year year = Year.of(2004);

    // année bissextile ?
    boolean isLeap = year.isLeap();

    // 08/2004
    YearMonth yearMonth = year.atMonth(Month.AUGUST);
```

```
    // 31/08/2004
    LocalDate localDate = yearMonth.atEndOfMonth();
  }

}
```

La classe Instant

La classe Instant représente un point dans le temps. Contrairement aux classes précédentes qui permettent de représenter les dates pour les humains, la classe Instant est adaptée pour réaliser des traitements de données temporelles.

```
package com.cgi.udev;

import java.time.Instant;

public class TestInstant {

  public static void main(String[] args) {
    Instant maintenant = Instant.now();
    Instant epoch = Instant.ofEpochSecond(0); // 01/01/1970 00:00:00.000

    Instant uneMinuteDansLeFuture = maintenant.plusSeconds(60);

    long unixTimestamp = uneMinuteDansLeFuture.getEpochSecond();
  }

}
```

> Les classes LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Year, YearMonth, Instant implémentent toutes les interfaces Temporal et TemporalAccessor. Cela permet d'utiliser facilement des instances de ces classes les unes avec les autres puisque beaucoup de leurs méthodes attendent en paramètres des instances de type Temporal ou TemporalAccessor.

Période et durée

Il est possible de définir des périodes grâce à des instances de la classe Period. Une période peut être construite directement ou à partir de la différence entre deux instances de type Temporal. Il est ensuite possible de modifier une date en ajoutant ou soustrayant une période.

```
package com.cgi.udev;

import java.time.LocalDate;
import java.time.Month;
import java.time.Period;
import java.time.Year;
import java.time.YearMonth;

public class TestPeriode {

  public static void main(String[] args) {
    YearMonth moisAnnee = Year.of(2000).atMonth(Month.APRIL); // 04/2000

    // période de 1 an et deux mois
    Period periode = Period.ofYears(1).plusMonths(2);

    YearMonth moisAnneePlusTard = moisAnnee.plus(periode); // 06/2001

    Period periode65Jours = Period.between(LocalDate.now(), LocalDate.now().plusDays(65));
  }

}
```

La durée est représentée par une instance de la classe Duration. Elle peut être obtenue à partir de deux instances de Instant.

```
package com.cgi.udev;

import java.time.Duration;
import java.time.Instant;

public class TestDuree {

  public static void main(String[] args) {
    Instant debut = Instant.now();
```

```
    // ... traitement à mesurer

    Duration duree = Duration.between(debut, Instant.now());
    System.out.println(duree.toMillis());
  }

}
```

Formatage des dates

Pour formater une date pour l'affichage, il est possible d'utiliser la méthode *format* déclarée dans les classes LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Year et YearMonth.

Le format de représentation d'une date et/ou du temps est défini par la classe DateTimeFormatter.

```
package com.cgi.udev;

import java.time.LocalDateTime;
import java.time.Month;
import java.time.format.DateTimeFormatter;
import java.util.Locale;

public class TestDuree {

  public static void main(String[] args) {
    // 01/09/2010 16:30
    LocalDateTime dateTime = LocalDateTime.of(2010, Month.SEPTEMBER, 1, 16, 30);

    // En utilisant des formats ISO de dates
    System.out.println(dateTime.format(DateTimeFormatter.BASIC_ISO_DATE));
    System.out.println(dateTime.format(DateTimeFormatter.ISO_WEEK_DATE));
    System.out.println(dateTime.format(DateTimeFormatter.ISO_DATE_TIME));

    DateTimeFormatter datePattern = DateTimeFormatter.ofPattern("dd/MM/yyyy");
    // 01/09/2010
    System.out.println(dateTime.format(datePattern));

    DateTimeFormatter dateTimePattern = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
```

```
    // 01/09/2010 16:30
    System.out.println(dateTime.format(dateTimePattern));

    // 1 septembre 2010
    DateTimeFormatter frenchDatePattern = DateTimeFormatter.ofPattern("d MMMM yyyy",
Locale.FRANCE);
    System.out.println(dateTime.format(frenchDatePattern));
  }

}
```

> Il est toujours possible d'utiliser la classe SimpleDateFormat pour formater une instance de la classe java.util.Date.

Revision #1
Created 10 February 2025 20:57:11 by Nicolas
Updated 10 February 2025 20:58:30 by Nicolas