Java 8 : API Date/Time

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Java

L'API Date-Time (disponible depuis Java 8)

- Nouvelle API pour la manipulation de date
- Simplifiant la manipulation, la recherche et la comparaison des dates
- Offrant une possibilité de travailler
  - soit sur le temps machine (Timestamp ou le nombre de secondes écoulées depuis 01/01/1970) : **Instant**
  - soit sur le temps humain (jour, mois, année, heure, minute, seconde...) : **LocalDateTime**, **LocalDate**, **LocalTime**...

Java

L'API Date-Time (disponible depuis Java 8)

- Nouvelle API pour la manipulation de date
- Simplifiant la manipulation, la recherche et la comparaison des dates
- Offrant une possibilité de travailler
  - soit sur le temps machine (Timestamp ou le nombre de secondes écoulées depuis 01/01/1970) : **Instant**
  - soit sur le temps humain (jour, mois, année, heure, minute, seconde...) : **LocalDateTime**, **LocalDate**, **LocalTime**...

Les classes de l'API Time se trouvent dans

```
java.time.*;
```

Java

Avantages de l'API Date-Time

- Richesse en terme de fonctionnalité
- Clarté
- Simplicité

Java

Plusieurs classes pour le temps humain

- `Year` : année
- `Month` : mois
- `YearMonth` : année + mois
- `MonthDay` : mois + jour
- `DayOfWeek` : jour
- `LocalDateTime` : date + heure
- `LocalDate` : date
- `LocalTime` : heure
- `ZonedDateTime` : date + heure + fuseau horaire
- `...`

Java

Pour obtenir une date, on fait appel à

- `now()` : pour obtenir une instance date et/ou l'heure courante
- `of()` : pour obtenir une instance à partir des données passées en paramètres
- `parser()` : pour obtenir une instance à partir d'une chaîne de caractère
- `from()` : pour obtenir une instance en convertissant des données passées en paramètres
- `...`

Java

Pour obtenir un objet `Year` contenant l'année du système

```
System.out.println(Year.now());
// affiche 2022
```

Java

Pour obtenir un objet `Year` contenant l'année du système

```
System.out.println(Year.now());
// affiche 2022
```

Pour obtenir un objet `Year` contenant l'année du système

```
System.out.println(Year.now().getValue());
// affiche 2022
```

Java

Pour obtenir un objet `Year` contenant l'année du système

```
System.out.println(Year.now());
// affiche 2022
```

Pour obtenir un objet `Year` contenant l'année du système

```
System.out.println(Year.now().getValue());
// affiche 2022
```


Pour construire un objet `Year` à partir d'un nombre passé en paramètre

```
System.out.println(Year.of(2018));
// affiche 2018
```

Pour savoir s'il s'agit d'une année bissextile

```
System.out.println(Year.of(2018).isLeap()).
// affiche false
```

Pour savoir s'il s'agit d'une année bissextile

```
System.out.println(Year.of(2018).isLeap()).
// affiche false
```

Ou aussi

```
System.out.println(Year.isLeap(2018));
// affiche false
```

Pour savoir s'il s'agit d'une année bissextile

```
System.out.println(Year.of(2018).isLeap()).
// affiche false
```

Ou aussi

```
System.out.println(Year.isLeap(2018));
// affiche false
```

On peut ajouter des années

```
System.out.println(Year.now().plusYears(3));
// affiche 2025
```

Pour savoir s'il s'agit d'une année bissextile

```
System.out.println(Year.of(2018).isLeap()).
// affiche false
```

Ou aussi

```
System.out.println(Year.isLeap(2018));
// affiche false
```

On peut ajouter des années

```
System.out.println(Year.now().plusYears(3));
// affiche 2025
```

Ou soustraire

```
System.out.println(Year.now().minusYears(3));
// affiche 2019
```

Java

Pour obtenir le mois selon son indice

```
System.out.println(Month.of(2));
// affiche FEBRUARY
```

Java

Pour obtenir le mois selon son indice

```
System.out.println(Month.of(2));
// affiche FEBRUARY
```

Pour obtenir le mois en français

```
System.out.println(Month.of(2).getDisplayName(TextStyle.FULL, Locale.
   FRANCE));
// affiche février
```

Java

Pour obtenir le nombre de jours max d'un mois

```
System.out.println(Month.of(2).maxLength());
// affiche 29
```

Java

Pour obtenir le nombre de jours max d'un mois

```
System.out.println(Month.of(2).maxLength());
// affiche 29
```

Pour obtenir le nombre de jours min d'un mois

```
System.out.println(Month.of(2).minLength());
// affiche 28
```

Java

Pour obtenir le nombre de jours max d'un mois

```
System.out.println(Month.of(2).maxLength());
// affiche 29
```

Pour obtenir le nombre de jours min d'un mois

```
System.out.println(Month.of(2).minLength());
// affiche 28
```

Pour obtenir le nombre de jours exact d'un mois

```
System.out.println(Month.of(2).length(Year.of(2022).isLeap()));
// affiche 28
System.out.println(Month.of(2).length(Year.of(2020).isLeap()));
// affiche 29
```

Java

Pour obtenir le mois suivant selon le paramètre

```
System.out.println(Month.of(2).plus(3));
// affiche MAY
```

Java

Pour obtenir le mois suivant selon le paramètre

```
System.out.println(Month.of(2).plus(3));
// affiche MAY
```

Pour obtenir le mois précédent selon le paramètre

```
System.out.println(Month.of(2).minus(2));
// affiche DECEMBER
```

Java

Pour obtenir le jour de la semaine selon l'indice

```
System.out.println(DayOfWeek.of(1));
// affiche MONDAY
```

Java

Pour obtenir le jour de la semaine selon l'indice

```
System.out.println(DayOfWeek.of(1));
// affiche MONDAY
```

Pour obtenir le jour de la semaine selon l'indice en français

```
System.out.println(DayOfWeek.of(1).getDisplayName(TextStyle.FULL,
   Locale.FRANCE));
// affiche lundi
```

Java

Pour obtenir le jour de la semaine selon l'indice

```
System.out.println(DayOfWeek.of(1));
// affiche MONDAY
```

Pour obtenir le jour de la semaine selon l'indice en français

```
System.out.println(DayOfWeek.of(1).getDisplayName(TextStyle.FULL,
   Locale.FRANCE));
// affiche lundi
```

Les méthodes `plus` et `minus` sont aussi présentes dans `DayOfWeek`

```
System.out.println(DayOfWeek.of(1).plus(2));
// affiche WEDNESDAY
System.out.println(DayOfWeek.of(1).minus(2));
// affiche SATURDAY
```

Java

Pour obtenir le mois et le jour du système

```
System.out.println(MonthDay.now());
// affiche --01-12
```

Java

Pour obtenir le mois et le jour du système

```
System.out.println(MonthDay.now());
// affiche --01-12
```

Pour construire un objet `MonthDay` avec des valeurs passées en paramètre

```
System.out.println(MonthDay.of(7, 30));
// affiche --07-30
```

Java

Pour obtenir le mois et l'année du système

```
System.out.println(YearMonth.now());
// affiche 2022-01
```

Java

Pour obtenir le mois et l'année du système

```
System.out.println(YearMonth.now());
// affiche 2022-01
```

Pour construire un objet `YearMonth` avec les valeurs passées en paramètre

```
System.out.println(YearMonth.of(1985, 7));
// affiche 1985-07
```

Java

Pour obtenir la date et l'heure actuelle

```
LocalDateTime localDateTime = LocalDateTime.now();
System.out.println("Date et heure actuelle : " + localDateTime);
// affiche Date et heure courante : 2018-07-30T03:09:01.874
```

Java

Pour obtenir la date et l'heure actuelle

```
LocalDateTime localDateTime = LocalDateTime.now();
System.out.println("Date et heure actuelle : " + localDateTime);
// affiche Date et heure courante : 2018-07-30T03:09:01.874
```

Pour obtenir la date (sans l'heure) à partir d'un `LocalDateTime`

```
LocalDate localDate = localDateTime.toLocalDate();
System.out.println("Date actuelle : " + localDate);
// affiche Date actuelle : 2018-07-30
```

Java

Pour obtenir la date et l'heure actuelle

```
LocalDateTime localDateTime = LocalDateTime.now();
System.out.println("Date et heure actuelle : " + localDateTime);
// affiche Date et heure courante : 2018-07-30T03:09:01.874
```

Pour obtenir la date (sans l'heure) à partir d'un `LocalDateTime`

```
LocalDate localDate = localDateTime.toLocalDate();
System.out.println("Date actuelle : " + localDate);
// affiche Date actuelle : 2018-07-30
```

Pour obtenir l'heure (sans la date) à partir d'un `LocalDateTime`

```
LocalTime localTime = localDateTime.toLocalTime();
System.out.println("Heure actuelle : " + localTime);
// affiche Heure actuelle : 03:14:40.495
```

Java

Comme pour `LocalDateTime`, on peut appeler `now` pour avoir la date actuelle (sans l'heure)

```
LocalDate localDate = LocalDate.now();
System.out.println("Date actuelle : " + localDate);
// affiche Date actuelle : 2022-01-12
```

Java

Comme pour `LocalDateTime`, on peut appeler `now` pour avoir la date actuelle (sans l'heure)

```
LocalDate localDate = LocalDate.now();
System.out.println("Date actuelle : " + localDate);
// affiche Date actuelle : 2022-01-12
```

Idem pour l'heure

```
LocalTime localTime = LocalTime.now();
System.out.println("Heure actuelle : " + localTime);
// affiche Heure actuelle : 21:02:49.025696300
```

Java

Comme pour `LocalDateTime`, on peut appeler `now` pour avoir la date actuelle (sans l'heure)

```
LocalDate localDate = LocalDate.now();
System.out.println("Date actuelle : " + localDate);
// affiche Date actuelle : 2022-01-12
```

Idem pour l'heure

```
LocalTime localTime = LocalTime.now();
System.out.println("Heure actuelle : " + localTime);
// affiche Heure actuelle : 21:02:49.025696300
```

On peut aussi construire `LocalDateTime` à partir d'un `LocalDate` et `LocalTime`

```
LocalDateTime localDateTime = LocalDateTime.of(localDate, localTime);
System.out.println("Date et heure actuelle : " + localDateTime);
// affiche Date et heure courante : 2022-01-12T21:02:49.025696300
```

Java

Il est possible de construire une date en passant les différents paramètres : année, mois, jour, heure, minute... (plusieurs surcharges possibles pour cette méthode)

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, 07, 30, 01, 30, 20);
System.out.println(dateHeureNaissance);
// affiche 1985-07-30T01:30:20
```

Java

Il est possible de construire une date en passant les différents paramètres : année, mois, jour, heure, minute... (plusieurs surcharges possibles pour cette méthode)

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, 07, 30, 01, 30, 20);
System.out.println(dateHeureNaissance);
// affiche 1985-07-30T01:30:20
```

Ou en utilisant les énumérations

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, Month.JULY, 30, 01, 30, 20);
System.out.println(dateHeureNaissance);
// affiche 1985-07-30T01:30:20
```

Java

Il est possible de construire une date en passant les différents paramètres : année, mois, jour, heure, minute... (plusieurs surcharges possibles pour cette méthode)

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, 07, 30, 01, 30, 20);
System.out.println(dateHeureNaissance);
// affiche 1985-07-30T01:30:20
```

Ou en utilisant les énumérations

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, Month.JULY, 30, 01, 30, 20);
System.out.println(dateHeureNaissance);
// affiche 1985-07-30T01:30:20
```

Il est possible de construire que la date (ou que l'heure)

```
LocalDate dateNaissance = LocalDate.of(1985, Month.JULY, 30);
System.out.println(dateNaissance);
// affiche 1985-07-30
```

Java

Pour construire une date avec un motif particulier, on utilise `DateTimeFormatter`

```
DateTimeFormatter format = DateTimeFormatter.ofPattern("d-MMMM-yyyy");
LocalDate formattedLocalDate = LocalDate.parse("01-mars-2020", format);
System.out.println(formattedLocalDate);
// affiche 2020-03-01
```

Java

Pour construire une date avec un motif particulier, on utilise `DateTimeFormatter`

```
DateTimeFormatter format = DateTimeFormatter.ofPattern("d-MMMM-yyyy");
LocalDate formattedLocalDate = LocalDate.parse("01-mars-2020", format);
System.out.println(formattedLocalDate);
// affiche 2020-03-01
```

Pour spécifier la langue utilisée dans la construction de la date

```
DateTimeFormatter format = DateTimeFormatter.ofPattern("d-MMMM-yyyy", Locale.ENGLISH);
LocalDate formattedLocalDate = LocalDate.parse("01-July-2020", format);
System.out.println(formattedLocalDate);
```

Java

Il est possible de construire une date à partir d'une chaîne de caractère

```
String dateString = "1985-07-30";
LocalDate dateFormString = LocalDate.parse(dateString);
System.out.println(dateFormString);
// affiche 1985-07-30
```

Java

Il est possible de construire une date à partir d'une chaîne de caractère

```
String dateString = "1985-07-30";
LocalDate dateFormString = LocalDate.parse(dateString);
System.out.println(dateFormString);
// affiche 1985-07-30
```


On peut récupérer un élément de la date

```
System.out.println(dateFormString.getDayOfMonth());
// affiche 30

System.out.println(dateFormString.getDayOfYear());
// affiche 211

System.out.println(dateFormString.getDayOfMonth());
// affiche 30

System.out.println(dateFormString.getMonthValue());
// affiche 7

System.out.println(dateFormString.getYear());
// affiche 1985
```

Java

Pour obtenir le mois ou le jour en toutes lettres

```
System.out.println(dateFormString.getDayOfWeek());
// affiche TUESDAY

System.out.println(dateFormString.getMonth());
// affiche JULY
```

Java

Pour obtenir le mois ou le jour en toutes lettres

```
System.out.println(dateFormString.getDayOfWeek());
// affiche TUESDAY

System.out.println(dateFormString.getMonth());
// affiche JULY
```

Pour obtenir le mois ou le jour en toutes lettres en français

```
System.out.println(dateFormString.getDayOfWeek().getDisplayName(TextStyle.FULL, Locale.
   FRANCE));
// affiche mardi

System.out.println(dateFormString.getMonth().getDisplayName(TextStyle.FULL, Locale.
   FRANCE));
// affiche juillet
```

Java

Pour obtenir le mois ou le jour en toutes lettres

```
System.out.println(dateFormString.getDayOfWeek());
// affiche TUESDAY

System.out.println(dateFormString.getMonth());
// affiche JULY
```

Pour obtenir le mois ou le jour en toutes lettres en français

```
System.out.println(dateFormString.getDayOfWeek().getDisplayName(TextStyle.FULL, Locale.
   FRANCE));
// affiche mardi

System.out.println(dateFormString.getMonth().getDisplayName(TextStyle.FULL, Locale.
   FRANCE));
// affiche juillet
```

Pour le jour, on peut faire aussi

```
System.out.println(DayOfWeek.from(dateFormString).getDisplayName(TextStyle.FULL, Locale
   .FRANCE));
// affiche mardi
```

Java

On peut aussi incrémenter une date en ajoutant un nombre de jours, mois, années...

```
System.out.println(dateFormString.plus(2, ChronoUnit.DAYS));
// affiche 1985-08-01

System.out.println(dateFormString.plus(1, ChronoUnit.WEEKS));
// affiche 1985-08-06

System.out.println(dateFormString.plus(30, ChronoUnit.YEARS));
// affiche 2015-07-30

System.out.println(dateFormString.plus(1, ChronoUnit.MONTHS));
// affiche 1985-08-30
```

Java

On peut aussi incrémenter une date en ajoutant un nombre de jours, mois, années...

```
System.out.println(dateFormString.plus(2, ChronoUnit.DAYS));
// affiche 1985-08-01

System.out.println(dateFormString.plus(1, ChronoUnit.WEEKS));
// affiche 1985-08-06

System.out.println(dateFormString.plus(30, ChronoUnit.YEARS));
// affiche 2015-07-30

System.out.println(dateFormString.plus(1, ChronoUnit.MONTHS));
// affiche 1985-08-30
```


On peut aussi décrémenter une date en retirant un nombre de jours, mois, années...

```
System.out.println(dateFormString.minus(2, ChronoUnit.DAYS));
// affiche 1985-07-28

System.out.println(dateFormString.minus(1, ChronoUnit.WEEKS));
// affiche 1985-07-23
```

Java

On peut aussi utiliser la classe `TemporalAdjusters` pour aller à un jour précis

```
LocalDate lundiSuivant = dateFormString.with(TemporalAdjusters.
   next(DayOfWeek.MONDAY));
System.out.println(lundiSuivant);
// affiche 1985-08-05
```

Java

On peut aussi utiliser la classe `TemporalAdjusters` pour aller à un jour précis

```
LocalDate lundiSuivant = dateFormString.with(TemporalAdjusters.
   next(DayOfWeek.MONDAY));
System.out.println(lundiSuivant);
// affiche 1985-08-05
```

Ou aussi

```
LocalDate lundiPrecedent = dateFormString.with(
   TemporalAdjusters.previous(DayOfWeek.MONDAY));
System.out.println(lundiPrecedent);
// affiche 1985-07-29
```

Considérons les deux dates suivantes

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, Month.JULY, 30, 01, 30, 20);
LocalDateTime dateDuJour = LocalDateTime.of(2018, Month.JULY, 30, 03, 59, 20);
```

Considérons les deux dates suivantes

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, Month.JULY, 30, 01, 30, 20);
LocalDateTime dateDuJour = LocalDateTime.of(2018, Month.JULY, 30, 03, 59, 20);
```

Pour comparer deux dates, on peut utiliser la méthode `compareTo()` qui commence par comparer les années, puis les mois...

```
System.out.println(dateHeureNaissance.compareTo(dateDuJour));
// affiche -33 (différence d'année) car dateHeureNaissance < dateDuJour

System.out.println(dateDuJour.compareTo(dateHeureNaissance));
// affiche 33 car dateHeureNaissance > dateDuJour

System.out.println(dateHeureNaissance.compareTo(dateHeureNaissance));
// affiche 0 car dateHeureNaissance = dateDuJour
```

Considérons les deux dates suivantes

```
LocalDateTime dateHeureNaissance = LocalDateTime.of(1985, Month.JULY, 30, 01, 30, 20);
LocalDateTime dateDuJour = LocalDateTime.of(2018, Month.JULY, 30, 03, 59, 20);
```

Pour comparer deux dates, on peut utiliser la méthode `compareTo()` qui commence par comparer les années, puis les mois...

```
System.out.println(dateHeureNaissance.compareTo(dateDuJour));
// affiche -33 (différence d'année) car dateHeureNaissance < dateDuJour

System.out.println(dateDuJour.compareTo(dateHeureNaissance));
// affiche 33 car dateHeureNaissance > dateDuJour

System.out.println(dateHeureNaissance.compareTo(dateHeureNaissance));
// affiche 0 car dateHeureNaissance = dateDuJour
```

On peut aussi utiliser les méthodes `isBefore()`, `isAfter()` ou `isEqual()`

```
System.out.println(dateHeureNaissance.isAfter(dateDuJour));
// affiche false

System.out.println(dateDuJour.isBefore(dateHeureNaissance));
// affiche false

System.out.println(dateHeureNaissance.isEqual(dateHeureNaissance));
// affiche true
```

Java

Pour les comparaisons, on peut aussi utiliser

- `ChronoUnit` pour comparer les `LocalDateTime`, les `LocalDate` et les `LocalTime`
- `Period` pour comparer seulement les `LocalDate`
- `Duration` permet de
  - construire une durée en secondes `LocalTime`
  - comparer deux `LocalTime` ou deux `LocalDateTime` mais pas deux `LocalDate`

Pour connaître la différence entre deux dates

```
System.out.println(ChronoUnit.CENTURIES.between(dateHeureNaissance, dateDuJour));
// affiche 0

System.out.println(ChronoUnit.DECADES.between(dateHeureNaissance, dateDuJour));
// affiche 3

System.out.println(ChronoUnit.YEARS.between(dateHeureNaissance, dateDuJour));
// affiche 33

System.out.println(ChronoUnit.MONTHS.between(dateHeureNaissance, dateDuJour));
// affiche 396

System.out.println(ChronoUnit.WEEKS.between(dateHeureNaissance, dateDuJour));
// affiche 1721

System.out.println(ChronoUnit.DAYS.between(dateHeureNaissance, dateDuJour));
// affiche 12053

System.out.println(ChronoUnit.HALF_DAYS.between(dateHeureNaissance, dateDuJour));
// affiche 24106

System.out.println(ChronoUnit.HOURS.between(dateHeureNaissance, dateDuJour));
// affiche 289274

System.out.println(ChronoUnit.MINUTES.between(dateHeureNaissance, dateDuJour));
// affiche 17356469

System.out.println(ChronoUnit.SECONDS.between(dateHeureNaissance, dateDuJour));
// affiche 1041388140
```

Java

Remarque

Si nous comparons des `LocalTime`, nous ne pourrons pas utiliser `ChronoUnit.CENTURIES`, `ChronoUnit.DECADES`... Nous pourrons seulement utiliser `ChronoUnit.HOURS`, `ChronoUnit.MINUTES`...

Java

Ou aussi en utilisant la méthode `between` de la classe `Period`

```
Period periode = Period.between(dateHeureNaissance.toLocalDate(), dateDuJour.
   toLocalDate());

System.out.println(periode.getYears());
// affiche 33

System.out.println(periode.getMonths());
// affiche 0

System.out.println(periode.getDays());
// affiche 0
```

Java

Ou aussi en utilisant la méthode `between` de la classe `Period`

```
Period periode = Period.between(dateHeureNaissance.toLocalDate(), dateDuJour.
   toLocalDate());

System.out.println(periode.getYears());
// affiche 33

System.out.println(periode.getMonths());
// affiche 0

System.out.println(periode.getDays());
// affiche 0
```


`Period`

- permet de comparer les `LocalDate`
- ne retourne pas d'heure ni minutes, ni secondes

Java

Sans convertir en `LocalTime`, le résultat n'est pas le même car on compare deux dates et non seulement deux heures

```
Duration duree1 = Duration.between(dateHeureNaissance, dateDuJour);

System.out.println(duree1.getSeconds());
// affiche 1041388140

Duration duree2 = Duration.between(dateHeureNaissance.toLocalTime(),
   dateDuJour.toLocalTime());

System.out.println(duree2.getSeconds());
// affiche 8940
```

Java

Pour connaître le fuseau horaire actuel

```
System.out.println("Fuseau horaire par défaut : " + ZoneId.systemDefault());
// affiche Fuseau horaire par défaut : Europe/Paris
```

Java

Pour connaître le fuseau horaire actuel

```
System.out.println("Fuseau horaire par défaut : " + ZoneId.systemDefault());
// affiche Fuseau horaire par défaut : Europe/Paris
```

Pour connaître les règles appliquées aux heures

```
System.out.println("Règles appliquées aux heures : " + ZoneId.systemDefault().getRules
   ());
// affiche Règles appliquées aux heures : ZoneRules[currentStandardOffset=+01:00]
```

Java

Pour obtenir une date avec le fuseau horaire

```
ZonedDateTime zonedDateTime = ZonedDateTime.now();
System.out.println(zonedDateTime);
// affiche 2019-09-22T10:33:18.496+02:00[Europe/Paris]
```

Java

Pour obtenir une date avec le fuseau horaire

```
ZonedDateTime zonedDateTime = ZonedDateTime.now();
System.out.println(zonedDateTime);
// affiche 2019-09-22T10:33:18.496+02:00[Europe/Paris]
```

Pour obtenir une date selon un motif défini

```
DateTimeFormatter dateTimeFormatter = DateTimeFormatter.
   ofPattern("'Le' dd '/' MMMM '/' yyyy");
System.out.println(zonedDateTime.format(dateTimeFormatter));
// affiche Le 22 / septembre / 2019
```

Java

Comment connaître tous les fuseaux horaires ?

```
ZoneId.getAvailableZoneIds().forEach(System.out::println);
// affiche une liste longue de fuseaux horaires
```

Java

Comment connaître tous les fuseaux horaires ?

```
ZoneId.getAvailableZoneIds().forEach(System.out::println);
// affiche une liste longue de fuseaux horaires
```

Pour obtenir la date et l'heure de Paris, on peut faire

```
ZoneId paris = ZoneId.of("Europe/Paris");
ZonedDateTime dateHeureParis = ZonedDateTime.of(LocalDateTime.now(), paris);
System.out.println(dateHeureParis);
// affiche 2019-09-22T20:09:27.949+02:00[Europe/Paris]
```

Java

Comment connaître tous les fuseaux horaires ?

```
ZoneId.getAvailableZoneIds().forEach(System.out::println);
// affiche une liste longue de fuseaux horaires
```

Pour obtenir la date et l'heure de Paris, on peut faire

```
ZoneId paris = ZoneId.of("Europe/Paris");
ZonedDateTime dateHeureParis = ZonedDateTime.of(LocalDateTime.now(), paris);
System.out.println(dateHeureParis);
// affiche 2019-09-22T20:09:27.949+02:00[Europe/Paris]
```


Comment connaître la date et l'heure exacte de Michigan aux États-Unis ?

```
ZoneId michigan = ZoneId.of("US/Michigan");
ZonedDateTime dateHeureMichigan = dateHeureParis.withZoneSameInstant(michigan);
System.out.println(dateHeureMichigan);
// affiche 2019-09-22T14:09:27.949-04:00[US/Michigan]
```

Java

Comment connaître le décalage horaire (Offset) par rapport à l'heure universelle

```
System.out.println(dateHeureParis.getOffset());
// affiche +02:00
```

Java

Comment connaître le décalage horaire (Offset) par rapport à l'heure universelle

```
System.out.println(dateHeureParis.getOffset());
// affiche +02:00
```

Pour construire une date avec le décalage horaire, on utilise la classe `OffsetDateTime`

```
OffsetDateTime dateAvecOffset = OffsetDateTime.of(LocalDateTime
   .now(), ZoneOffset.ofHours(6));
System.out.println(dateAvecOffset);
// affiche 2019-09-22T20:34:34.138+06:00
```

Java

Pour obtenir l'heure universelle

```
Instant maintenant = Instant.now();
System.out.println(maintenant);
// affiche 2019-09-22T05:12:06.030Z
```

Java

Pour obtenir l'heure universelle

```
Instant maintenant = Instant.now();
System.out.println(maintenant);
// affiche 2019-09-22T05:12:06.030Z
```

Pourobtenir le timestamp (nombre de secondes écoulées depuis 01/01/1970)

```
System.out.println(maintenant.getEpochSecond());
// affiche 1569135019
```

Java

Pour obtenir l'heure universelle

```
Instant maintenant = Instant.now();
System.out.println(maintenant);
// affiche 2019-09-22T05:12:06.030Z
```

Pourobtenir le timestamp (nombre de secondes écoulées depuis 01/01/1970)

```
System.out.println(maintenant.getEpochSecond());
// affiche 1569135019
```

Pour ajouter du temps à notre instant

```
Instant demain = maintenant.plus(1, ChronoUnit.DAYS);
System.out.println(demain);
// affiche 2019-09-23T06:56:22.998Z
```

Pour soustraire du temps à notre instant

```
Instant hier = maintenant.minus(1, ChronoUnit.DAYS);
System.out.println(hier);
// affiche 2019-09-21T07:00:16.265Z
```

Pour soustraire du temps à notre instant

```
Instant hier = maintenant.minus(1, ChronoUnit.DAYS);
System.out.println(hier);
// affiche 2019-09-21T07:00:16.265Z
```

Remarque

Seules les constantes `ChronoUnit.DAYS` et `ChronoUnit.HALF_DAYS` fonctionnent avec les instants

Pour soustraire du temps à notre instant

```
Instant hier = maintenant.minus(1, ChronoUnit.DAYS);
System.out.println(hier);
// affiche 2019-09-21T07:00:16.265Z
```

Remarque

Seules les constantes `ChronoUnit.DAYS` et `ChronoUnit.HALF_DAYS` fonctionnent avec les instants

Pour comparer deux instants, on peut utiliser la méthode `compareTo()`

```
System.out.println(maintenant.compareTo(hier));
// affiche 1 car maintenant > hier
System.out.println(hier.compareTo(maintenant));
// affiche -1 car maintenant < hier
System.out.println(maintenant.compareTo(maintenant));
// affiche 0 car maintenant = hier
```

Java

Pour comparer deux instants, on peut aussi utiliser les méthodes `isBefore()`, `isAfter()` **ou** `isEqual()`

```
System.out.println(maintenant.isAfter(hier));
// affiche true

System.out.println(maintenant.isBefore(hier));
// affiche false

System.out.println(maintenant.isEqual(maintenant));
// affiche true
```

Java

Date-time classes in Java	Modern class	Legacy class
Moment in UTC	`java.time.Instant`	`java.util.Date` ~~`Date`~~ `java.sql.Timestamp` ~~`Timestamp`~~
Moment with offset-from-UTC (hours-minutes-seconds)	`java.time.OffsetDateTime`	(lacking)
Moment with time zone (`Continent/Region`)	`java.time.ZonedDateTime`	`java.util.` ~~`GregorianCalendar`~~
Date & Time-of-day (no offset, no zone) **Not a moment**	`java.time.LocalDateTime`	(lacking)

Les dates après et avant Java 8 (Source : Stack OverFlow)