# 7 - Date/Time API

< Previous I Next >

- All classess in java.time package such as classes for date, time, date and time combined, time zones, instants, duration, and clocks are immutable and thread-safe.
- Instant doesn't represent date. It represents an instantaneous point on the time-line in nanoseconds and is used to record event time-stamps in the application.
- The java.time package contains main API for dates, times, instants, and durations. The classes defined in this package represent the principle date-time concepts, including instants, durations, dates, times, time-zones and periods. All the classes of this package are immutable and thread-safe.
- Period has the prefix P, while Duration has the prefix PT.
- Duration (PT) containt H-M-S.
- Period (P) contains Y-M-D.
- With the `Period.between(date1, date2)` method, if the second date is before the first date, a minus sign is included in the output.
- Important thing to remember here is that Period is used to manipulate dates in terms of days, months, and years, while Duration is used to manipulate dates in terms of hours, minutes, and seconds. Therefore, Period doesn't mess with the time component of the date while Duration may change the time component if the date is close to the DST boundary. Durations and periods differ in their treatment of daylight savings time when added to ZonedDateTime. A Duration will add an exact number of seconds, thus a duration of one day is always exactly 24 hours. By contrast, a Period will add a conceptual day, trying to maintain the local time.
- Duration counts in terms of hours, minutes, and seconds. Therefore, days are converted into hours. That is why `System.out.println()Duration.ofDays(1))` will print PT24H and not PT1D.
- Duration does not convert hours into days. i.e. 25 hours will remain as 25 hours instead of 1 day and 1 hour.
- A Duration of 0 is printed as 0S and a Period of 0 is printed as 0D.
- Instant is a point on Java time line. This timeline start from from the first second of January 1, 1970 (1970-01-01T00:00:00Z) also called the EPOCH. Note that there is no time zone here. You may think of it as "Java Time Zone" and it matches with GMT or UTC time zone.
- Once created, an Instant cannot be modified. Its methods such as plus and minus return a new Instant object.
- Instant works with time (instead of dates), so you can use Duration instance to create new Instants.
- LocalDateTime is a time in a given time zone. (But remember that an instance of LocalDateTime itself does not store the zone information!). You can, therefore, use an Instant and a time zone to create a LocalDateTime object.
- Whenever you convert an Instant to a LocalDateTime using a time zone, just add or substract the GMT offset of the time zone i.e. if the time zone is GMT+2, add 2 hours and if the time zone is GMT-2, substract two hours.

- Instant class has a truncatedTo method that takes in a TemporalUnit and returns a new Instant with the fields smaller than the passed unit set to zero. For example, if you pass ChronoUnit.DAYS, hours, minutes, seconds, and nano-seconds will be set to 0 in the resulting Instant.
- TemporalUnit is an interface and ChronoUnit is a class that implements this interface and defines constants such as DAYS, MONTHS, and YEARS.
- FYI, any unit larger than ChronoUnit.DAYS causes the truncatedTo method to throw UnsupportedTemporalTypeException.

**#toString method of java.time.Duration:**

- It generates a string representation of the duration object using ISO-8601 seconds based representation, such as `PT8H6M12.345S`.
- The format of the returned string will be PTnHnMnS, where n is the relevant hours, minutes or seconds part of the duration. Any fractional seconds are placed after a decimal point i the seconds section. If a section has a zero value, it is omitted. The hours, minutes and seconds will all have the same sign.

**# `public static Duration between(Temporal startInclusive, Temporal endExclusive)` :**

1. Duration.between method computes the duration between two temporal objects. If the objects are of different types, then the duration is calculated based on the type of the first object. For example, if the first argument is a LocalTime then the second argument is converted to a LocalTime. (Not the case here because both arguments are of ZonedDateTime).
2. In this case, the difference between the two zones is 3 hours therefore the resulting duration will contain 3 hours.
3. The result of Duration.between method can be a negative period if the end is before the start.

- Duration.between method needs Temporal arguments that have a time component. (java.time.temporal.UnsupportedTemporalTypeException)

```
    LocalDateTime ld1 = LocalDateTime.of(2015, Month.NOVEMBER, 1, 2, 0);
    ZonedDateTime zd1 = ZonedDateTime.of(ld1, ZoneId.of("US/Eastern"));
    LocalDateTime ld2 = LocalDateTime.of(2015, Month.NOVEMBER, 1, 1, 0);
    ZonedDateTime zd2 = ZonedDateTime.of(ld2, ZoneId.of("US/Eastern"));
    long x = ChronoUnit.HOURS.between(zd1, zd2);
    System.out.println(x);      // It will print -2
```

- The time difference between two dates is simply the amount of time you need to go from date 1 to date 2. So if you want to go from 1AM to 2AM, how many hours do you need? On a regular day, you need 1 hour. That is, if you add 1 hour to 1AM, you will get 2AM. However, as given in the problem statement, at the time of DST change, 2 AM becomes 1AM. That means, even after adding 1 hour to 1AM, you are not at 2AM. You have to add another hour to get to 2 AM. In total, therefore, you have to add 2 hours to 1AM to get to 2AM.
- The answer can therefore be short listed to 2 or -2. Now, as per the JavaDoc description of the between method, it returns negative value if the end is before the start. In the given code, our end date

is 1AM, while the start date is 2AM. This means, the answer is -2.

< Previous | Next >