20-date-and-time

Contents

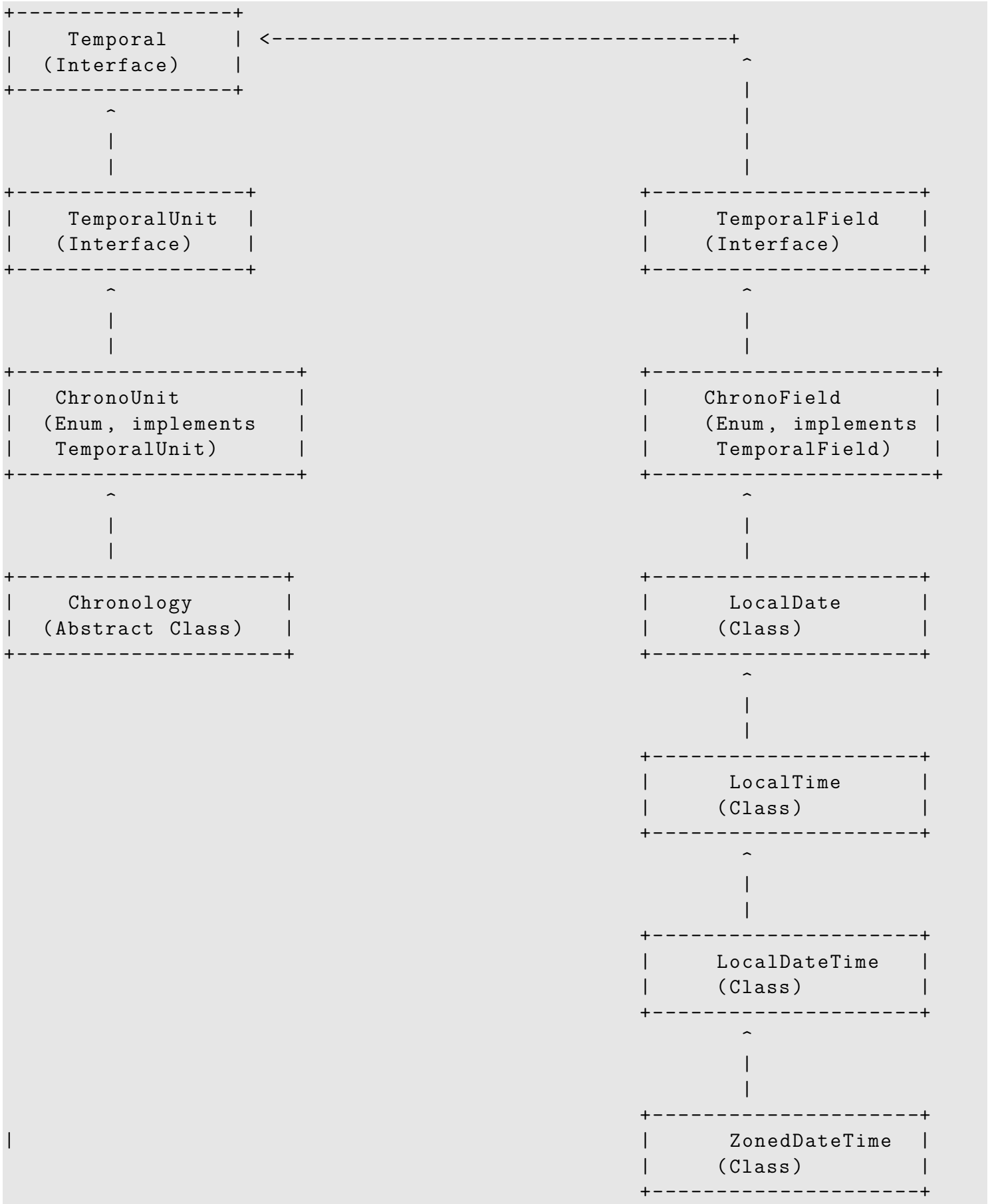
- Date & Time
  - TemporalUnit
    - LocalDate
    - LocalTime
    - LocalDateTime
    - DateTimeFormatter
    - ZonedDateTime
  - TemporalAmount
    - Period
    - Duration
    - Instant
    - Usage

Date & Time

With java 8 the standard introduced several new top level classes for dealing with dates and date transformations. They are called `LocalDate`, `Localime`, `LocalDateTime`, from the name it is easy to infer what those classes are meant to represent and interface with. Those are located at the top level package `java.time`, there are however more packages under the java time umbrella which are used to fine tune and provide fine grained access to modify and transform dates and time formats

`None of the new date & time related class types define a public constructor, they are meant to be constructed through factory static final methods, provided by the runtime, they follow this concept to be consistent with the rest of the new types provided by the run-time`

The diagram below shows the general relationship between the classes and interfaces in the new `java.time` package provided by the `JDK 8` specification



TemporalUnit

A unit of date-time, such as Days or Hours. Measurement of time is built on units, such as years, months, days, hours, minutes and seconds. Implementations of this interface represent those units. The various implementations are listed below

LocalDate

The local date time is internally represented using the `Gregorian calendar`, under the `ISO8601`. By default the format for the date of the is `LocalDate YYYY-MM-DD`

Method Name/Signature	Description
LocalDate.now()	Returns the current date based on the system clock.
LocalDate.of(int year, int month, int dayOfMonth)	Creates an instance of `LocalDate` for a specified year, month, and day.
LocalDate.ofYearDay(int year, int dayOfYear)	Returns a `LocalDate` from a given year and the day of the year (1 to 365/366).
LocalDate.parse(CharSequence text)	Parses a `LocalDate` from a text string (e.g., "2023-10-26").
LocalDate.parse(CharSequence text, DateTimeFormatter formatter)	Parses a date using the specified `DateTimeFormatter`.
plusDays(long daysToAdd)	Returns a copy of this `LocalDate` with the specified number of days added.
plusMonths(long monthsToAdd)	Returns a copy of this `LocalDate` with the specified number of months added.
plusYears(long yearsToAdd)	Returns a copy of this `LocalDate` with the specified number of years added.
minusDays(long daysToSubtract)	Returns a copy of this `LocalDate` with the specified number of days subtracted.
minusMonths(long monthsToSubtract)	Returns a copy of this `LocalDate` with the specified number of months subtracted.
minusYears(long yearsToSubtract)	Returns a copy of this `LocalDate` with the specified number of years subtracted.
isBefore(ChronoLocalDate otherDate)	Checks if this `LocalDate` is before the specified date.
isAfter(ChronoLocalDate otherDate)	Checks if this `LocalDate` is after the specified date.
isLeapYear()	Checks if the year of this `LocalDate` is a leap year.
getDayOfWeek()	Returns the day of the week for this `LocalDate` (e.g., MONDAY, TUESDAY).
getDayOfMonth()	Returns the day of the month (1-31) of this `LocalDate`.
getMonth()	Returns the `Month` enum for the month of this `LocalDate` (e.g., JANUARY, FEBRUARY).
getMonthValue()	Returns the month of this `LocalDate` as an `int` (1-12).
lengthOfMonth()	Returns the length of the month of this `LocalDate` in days.
lengthOfYear()	Returns the length of the year of this `LocalDate` in days (365 or 366).

```
LocalDate curDate = LocalDate.now(); // each of the new Local* classes
   implement a human readable toString()
System.out.println(curDate); // that would print the date in the format
   mentioned above, yyyy-mm-dd
```

LocalTime

Method Name/Signature	Description
LocalTime.now()	Returns the current time based on the system clock.
LocalTime.of(int hour, int minute)	Creates an instance of `LocalTime` for a specified hour and minute.
LocalTime.of(int hour, int minute, int second)	Creates an instance of `LocalTime` for a specified hour, minute, and second.
LocalTime.of(int hour, int minute, int second, int nanoOfSecond)	Creates an instance of `LocalTime` with hour, minute, second, and nanosecond.
LocalTime.parse(CharSequence text)	Parses a `LocalTime` from a text string (e.g., "12:34:56").
LocalTime.parse(CharSequence text, DateTimeFormatter formatter)	Parses a time using the specified `DateTimeFormatter`.
plusHours(long hoursToAdd)	Returns a copy of this `LocalTime` with the specified number of hours added.
plusMinutes(long minutesToAdd)	Returns a copy of this `LocalTime` with the specified number of minutes added.
plusSeconds(long secondsToAdd)	Returns a copy of this `LocalTime` with the specified number of seconds added.
plusNanos(long nanosToAdd)	Returns a copy of this `LocalTime` with the specified number of nanoseconds added.
minusHours(long hoursToSubtract)	Returns a copy of this `LocalTime` with the specified number of hours subtracted.
minusMinutes(long minutesToSubtract)	Returns a copy of this `LocalTime` with the specified number of minutes subtracted.
minusSeconds(long secondsToSubtract)	Returns a copy of this `LocalTime` with the specified number of seconds subtracted.
minusNanos(long nanosToSubtract)	Returns a copy of this `LocalTime` with the specified number of nanoseconds subtracted.
isBefore(LocalTime otherTime)	Checks if this `LocalTime` is before the specified time.
isAfter(LocalTime otherTime)	Checks if this `LocalTime` is after the specified time.
getHour()	Returns the hour part of this `LocalTime` (0-23).
getMinute()	Returns the minute part of this `LocalTime` (0-59).
getSecond()	Returns the second part of this `LocalTime` (0-59).
getNano()	Returns the nanosecond part of this `LocalTime` (0-999,999,999).
toSecondOfDay()	Returns the number of seconds since midnight for this `LocalTime`.
toNanoOfDay()	Returns the number of nanoseconds since midnight for this `LocalTime`.

```
LocalTime curTime = LocalTime.now(); // similarly to the LocalDate, LocalTime
    also implements a human readable toString()
System.out.println(curTime); // that would print the time in the format
   mentioned above, hh:mm:ss:mmmm
```

LocalDateTime

Method Name/Signature	Description
LocalDateTime.now()	Returns the current date-time based on the system clock.
LocalDateTime.of(int year, int month, int dayOfMonth, int hour, int minute)	Creates an instance with specified date and time.
LocalDateTime.of(int year, int month, int dayOfMonth, int hour, int minute, int second)	Creates an instance with date, time, and seconds.
LocalDateTime.of(LocalDate date, LocalTime time)	Combines a `LocalDate` and `LocalTime` to create a `LocalDateTime`.
LocalDateTime.parse(CharSequence text)	Parses a `LocalDateTime` from a text string (e.g., "2023-10-26T10:15:30").
LocalDateTime.parse(CharSequence text, DateTimeFormatter formatter)	Parses using a specified `DateTimeFormatter`.
plusDays(long daysToAdd)	Returns a copy of this `LocalDateTime` with the specified number of days added.
plusHours(long hoursToAdd)	Returns a copy of this `LocalDateTime` with the specified number of hours added.
plusMinutes(long minutesToAdd)	Returns a copy of this `LocalDateTime` with the specified number of minutes added.
plusSeconds(long secondsToAdd)	Returns a copy of this `LocalDateTime` with the specified number of seconds added.
plusNanos(long nanosToAdd)	Returns a copy of this `LocalDateTime` with the specified number of nanoseconds added.
minusDays(long daysToSubtract)	Returns a copy of this `LocalDateTime` with the specified number of days subtracted.
minusHours(long hoursToSubtract)	Returns a copy of this `LocalDateTime` with the specified number of hours subtracted.
minusMinutes(long minutesToSubtract)	Returns a copy of this `LocalDateTime` with the specified number of minutes subtracted.
minusSeconds(long secondsToSubtract)	Returns a copy of this `LocalDateTime` with the specified number of seconds subtracted.
minusNanos(long nanosToSubtract)	Returns a copy of this `LocalDateTime` with the specified number of nanoseconds subtracted.
isBefore(ChronoLocalDateTime<?> otherDateTime)	Checks if this `LocalDateTime` is before the specified date-time.
isAfter(ChronoLocalDateTime<?> otherDateTime)	Checks if this `LocalDateTime` is after the specified date-time.
toLocalDate()	Extracts the `LocalDate` part from this `LocalDateTime`.
toLocalTime()	Extracts the `LocalTime` part from this `LocalDateTime`.
getYear()	Returns the year part of this `LocalDateTime`.
getMonth()	Returns the `Month` enum for the month part of this `LocalDateTime`.
getDayOfMonth()	Returns the day of the month of this `LocalDateTime`.
getHour()	Returns the hour part of this `LocalDateTime` (0-23).
getMinute()	Returns the minute part of this `LocalDateTime` (0-59).
getSecond()	Returns the second part of this `LocalDateTime` (0-59).

Method Name/Signature	Description
getNano()	Returns the nanosecond part of this `LocalDateTime`.

```
LocalDateTime curDateTime = LocalDateTime.now(); // similarly to the
   LocalDate, LocalDateTime also implements a human readable toString()
System.out.println(curDateTime); // that would print the time in the format
   mentioned above, yyyy-mm-ddThh:mm:ss:mmmm
```

DateTimeFormatter

In general along with the core date time classes the default library provides means of changing or defining the format of a date when it is printed out to string, or to construct an instance of the `LocalDate, Time and DateTime` classes from a given pattern

Method Name/Signature	Description
static DateTimeFormatter ofPattern(String pattern)	Creates a `DateTimeFormatter` using the specified pattern string.
static DateTimeFormatter ofPattern(String pattern, Locale locale)	Creates a `DateTimeFormatter` using the specified pattern and locale.
static DateTimeFormatter ISO_LOCAL_DATE()	Returns a formatter for ISO-8601 local date format (yyyy-MM-dd).
static DateTimeFormatter ISO_LOCAL_TIME()	Returns a formatter for ISO-8601 local time format (HH:mm:ss).
static DateTimeFormatter ISO_LOCAL_DATE_TIME()	Returns a formatter for ISO-8601 local date-time format (yyyy-MM-dd'T'HH:mm:ss).
String format(TemporalAccessor temporal)	Formats the specified temporal object into a string.
T parse(CharSequence text, ResolverStyle resolverStyle)	Parses the text to produce a temporal object based on the formatter.
T parse(CharSequence text)	Parses the text to produce a temporal object based on the formatter's pattern.

The table below lists all the different indicators that can be used in the construction of a pattern, the rule of thumb is that based on the number of repretitions of the letter in the pattern that is how a certain element will be represented, that is not always true, but it is an approximate, for example `M MM MMM MMMM`, will give output of `4, 04, Apr, April`.

Indicator	Description
a	AM/PM indicator
d	Day in month
E	Day in week
h	Hour, 12-hour clock
H	Hour, 24-hour clock
M	Month
m	Minutes
s	Seconds
y	Year

Below one can simply see how the `formatter` can be used to parse any type of `LocalDate`, Time and `DateTime` into a String, or do the inverse operation of that by using the `parse` method to convert a String to an instance of Local*

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy"); //
   Define a pattern for formatting

LocalDate date = LocalDate.of(2023, 10, 26); // Create a LocalDate instance
String formattedDate = date.format(formatter); // Format the LocalDate to a
   string

String dateString = "26-10-2023"; // Define a local date as a String
LocalDate parsedDate = LocalDate.parse(dateString, formatter); // Parse a
   string to create a LocalDate instance
```

ZonedDateTime

This is a special type of `LocalDateTime`, which allows one to represent an actual `time zone` along with the rest of the information about a date and time. The `ZonedDateTime` class allows one to convert from regular `LocalDate, Time and DateTime` to `ZonedDateTime` and back.

```
LocalDate localDate = LocalDate.of(2024, 10, 26);
LocalTime localTime = LocalTime.of(14, 30); // 2:30 PM

// Combine LocalDate and LocalTime to create LocalDateTime
LocalDateTime localDateTime = LocalDateTime.of(localDate, localTime);
System.out.println("LocalDateTime: " + localDateTime);

// Convert LocalDateTime to ZonedDateTime
ZonedDateTime zonedDateTime = localDateTime.atZone(ZoneId.of("America/
   New_York"));
System.out.println("ZonedDateTime (America/New_York): " + zonedDateTime);

// Convert ZonedDateTime back to LocalDateTime
LocalDateTime convertedLocalDateTime = zonedDateTime.toLocalDateTime();
System.out.println("Converted LocalDateTime from ZonedDateTime: " +
   convertedLocalDateTime);

// Create ZonedDateTime directly
ZonedDateTime zonedDateTimeNow = ZonedDateTime.now(ZoneId.of("Europe/London")
   );
System.out.println("Current ZonedDateTime (Europe/London): " +
   zonedDateTimeNow);
```

The string representation of a ZonedDateTime is very similar to the regular `LocalDateTime` string representation however a zone information is included at the end of the string, the zone information is enclosed in a square brackets - `YYYY-MM-DDTHH:MM:SS.NNNNNNNNN[ZoneId]`. That is due to the fact that the zone information is very sensitive to changes in the date time, obviously, therefore a very accurate representation of the date and time information has to be available

`Note that the default printout for ZonedDateTime is always using nanos unlike the LocalDateTime which is has the format of YYYY-MM-DDTHH:MM:SS which does not include any nano seconds information`

TemporalAmount

Framework-level interface defining an amount of time, such as “6 hours”, “8 days” or “2 years and 3 months”. This is the base interface type for amounts of time. An amount is distinct from a date or time-of-day in that it is not tied to any specific point on the time-line. The various implementations of this interface are listed below.

Period

A period is defined based on date amount, measured in years, months or days. It is useful for representing a period of time based on dates. There are several methods which are provide means of inter operating with `LocalDate` or `LocalDateTime`, which are the relevant `TemporalUnits` which can support math with Period type, which is based on years, months and days. One such example is the method called `between` with the following signature - `Period.between(LocalDate start, LocalDate end)` - to calculate the period between two dates, that period instance can then be converted to any scale which the period class type supports, such as days, months or years

Duration

This class models a quantity or amount of time in terms of seconds and nanoseconds. It can be accessed using other duration-based units, such as minutes and hours. In addition, the `ChronoUnit.DAYS` unit can be used and is treated as exactly equal to 24 hours, thus ignoring daylight savings effects.

Instant

The Instant class represents a moment on the timeline, typically used to capture a timestamp. It is the closest equivalent to an absolute point in time in Java and is often measured in milliseconds (or nanoseconds) since the Unix epoch (1970-01-01T00:00:00Z). Commonly used when you need an exact, machine-based time representation (e.g., for timestamps or logging). Instant is immutable and thread-safe.

Usage

The table below represents the basic api of the temporal amount implementations such as Period, Duration, Instant - these are designed in such a way that they can easily inter operate between the other implementations of `TemporalAmount (Period, Duration, Instant)` and `TemporalUnit (LocalTime, LocalDate, LocalDateTime)`

`Note that most TemporalUnit implementation classes implement methods which work and inter operate with the TemporalAmount interface.`

Method	Description
LocalDate.atStartOfDay()	Converts a `LocalDate` to a `LocalDateTime` at the start of the day.
LocalDate.minus(Period period)	Returns a `LocalDate` that is the result of subtracting a `Period` from this `LocalDate`.
LocalDate.plus(Period period)	Returns a `LocalDate` that is the result of adding a `Period` to this `LocalDate`.
LocalDateTime.toInstant(ZoneOffset offset)	Converts a `LocalDateTime` to an `Instant` using a specified `ZoneOffset`.
Duration.between(Instant start, Instant end)	Calculates the duration between two `Instant` objects.

Method	Description
Duration.ofDays(long days)	Obtains a `Duration` representing a specified number of days.
Duration.ofHours(long hours)	Obtains a `Duration` representing a specified number of hours.
Duration.ofMinutes(long minutes)	Obtains a `Duration` representing a specified number of minutes.
Duration.ofSeconds(long seconds)	Obtains a `Duration` representing a specified number of seconds.
Duration.toHours()	Converts this `Duration` to a number of hours.
Duration.toMillis()	Converts this `Duration` to a number of milliseconds.
Instant.isAfter(Instant other)	Checks if this `Instant` is after the specified `Instant`.
Instant.isBefore(Instant other)	Checks if this `Instant` is before the specified `Instant`.
Instant.minus(Duration duration)	Returns an `Instant` that is the result of subtracting a `Duration` from this `Instant`.
Instant.minus(Duration duration)	Returns an `Instant` that is the result of subtracting a `Duration` from this `Instant`.
Instant.now()	Obtains the current instant from the system clock.
Instant.ofEpochMilli(long epochMilli)	Obtains an `Instant` from a timestamp in milliseconds since the epoch (1970-01-01T00:00:00Z).
Instant.ofEpochSecond(long epochSecond)	Obtains an `Instant` from a timestamp in seconds since the epoch.
Instant.plus(Duration duration)	Returns an `Instant` that is the result of adding a `Duration` to this `Instant`.
Instant.plus(Duration duration)	Returns an `Instant` that is the result of adding a `Duration` to this `Instant`.
Instant.toEpochMilli()	Converts this `Instant` to a number of milliseconds since the epoch.
Instant.toEpochSecond()	Converts this `Instant` to a number of seconds since the epoch.
Period.between(LocalDate start, LocalDate end)	Calculates the period between two `LocalDate` objects.
Period.minus(Period period)	Returns a `Period` that is the result of subtracting the specified `Period` from this `Period`.
Period.ofDays(int days)	Obtains a `Period` representing a specified number of days.
Period.ofMonths(int months)	Obtains a `Period` representing a specified number of months.
Period.ofYears(int years)	Obtains a `Period` representing a specified number of years.
Period.plus(Period period)	Returns a `Period` that is the result of adding the specified `Period` to this `Period`.
Period.toString()	Returns a string representation of the `Period`.
Period.toTotalMonths()	Converts this `Period` to a total number of months.