Standard Calendar

Standard Calendar

There are two basic ways to represent time. One way represents time in human terms, referred to as *human time*, such as year, month, day, hour, minute and second. The other way, *machine time*, measures time continuously along a timeline from an origin, called the *epoch*, in nanosecond resolution. The Date-Time package provides a rich array of classes for representing date and time. Some classes in the Date-Time API are intended to represent machine time, and others are more suited to representing human time.

First determine what aspects of date and time you require, and then select the class, or classes, that fulfill those needs. When choosing a temporal-based class, you first decide whether you need to represent human time or machine time. You then identify what aspects of time you need to represent. Do you need a time zone? Date and time? Date only? If you need a date, do you need month, day, and year, or a subset?

> *Terminology: The classes in the Date-Time API that capture and work with date or time values, such as `Instant`, `LocalDateTime`, and `ZonedDateTime`, are referred to as temporal-based classes (or types) throughout this section. Supporting types, such as the `TemporalAdjuster` interface or the `DayOfWeek` enum, are not included in this definition.*

For example, you might use a `LocalDate` object to represent a birth date, because most people observe their birthday on the same day, whether they are in their birth city or across the globe on the other side of the international date line. If you are tracking astrological time, then you might want to use a `LocalDateTime` object to represent the date and time of birth, or a `ZonedDateTime`, which also includes the time zone. If you are creating a time-stamp, then you will most likely want to use an `Instant`, which allows you to compare one instantaneous point on the timeline to another.

The following table summarizes the temporal-based classes in the `java.time` package that store date and/or time information, or that can be used to measure an amount of time. A check mark in a column indicates that the class uses that particular type of data and the `toString()` Output column shows an instance printed using the `toString()` method. The Where Discussed column links you to the relevant page in the tutorial.

Class or Enum	Content	toString() Output	Where Discussed
`Instant`	Seconds (1)	`2013-08-20T15:16:26.355Z`	Instant Class
`LocalDate`	Year, Month, Day	`2013-08-20`	Date Classes
`LocalDateTime`	Year, Month, Day, Hour, Minutes, Seconds	`2013-08-20T08:16:26.937`	Date and Time Classes
`ZonedDateTime`	Year, Month, Day, Hour,	`2013-08-21T00:16:26.941+09:00[Asia/Tokyo]`	Time Zone and Offset Classes

Class or Enum	Content	toString() Output	Where Discussed
	Minutes, Seconds, Zone Offset, Zone ID		
`LocalTime`	Hour, Minutes, Seconds	`08:16:26.943`	Date and Time Classes
`MonthDay`	Month, Day	`--08-20`	Date Classes
`Year`	Year	`2013`	Date Classes
`YearMonth`	Year, Month	`2013-08`	Date Classes
`Month`	Month	`AUGUST`	DayOfWeek and Month Enums
`OffsetDateTime`	Year, Month, Day, Hour, Minutes, Seconds, Zone Offset	`2013-08-20T08:16:26.954-07:00`	Time Zone and Offset Classes
`OffsetTime`	Hour, Minutes, Seconds,	`08:16:26.957-07:00`	Time Zone and Offset Classes

Class or Enum	Content	toString() Output	Where Discussed
	Zone Offset		
`Duration`	Day (2), Hour (2), Minutes (2), Seconds	`PT20H` (20 hours)	Period and Duration
`Period`	Year, Month, Day (3)	`P10D` (10 days)	Period and Duration
``			

Notes:

(1): Seconds are captured to nanosecond precision.

(2): This class does not store this information, but has methods to provide time in these units.

(3): When a Period is added to a `ZonedDateTime`, daylight saving time or other local time differences are observed.

Last update: *January 27, 2022*

The Date Time API Overview ← **Standard Calendar** → DayOfWeek and Month Enums

Home > Tutorials > The Date Time API > Standard Calendar