Date-Time

Overview

Time seems to be a simple subject; even an inexpensive watch can provide a reasonably accurate date and time. However, with closer examination, you realize the subtle complexities and many factors that affect your understanding of time. For example, the result of adding one month to January 31 is different for a leap year than for other years. Time zones also add complexity.

For example, a country may go in and out of daylight saving time at short notice, or more than once a year or it may skip daylight saving time entirely for a given year.

The Date-Time API uses the calendar system defined in ISO-8601 as the default calendar. This calendar is based on the Gregorian calendar system and is used globally as the de facto standard for representing date and time. The core classes in the Date-Time API have names such as LocalDateTime, ZonedDateTime, and OffsetDateTime. All of these use the ISO calendar system. If you want to use an alternative calendar system, such as Hijrah or Thai Buddhist, the java.time.chrono package allows you to use one of the predefined calendar systems. Or you can create your own.

The Date-Time API uses the Unicode Common Locale Data Repository (CLDR). This repository supports the world's languages and contains the world's largest collection of locale data available. The information in this repository has been localized to hundreds of languages. The Date-Time API also uses the Time-Zone Database (TZDB). This database provides information about every time zone change globally since 1970, with history for primary time zones since the concept was introduced.

Date-Time

Design Principles

The Date-Time API was developed using several design principles.

Clear

The methods in the API are well defined and their behavior is clear and expected. For example, invoking a Date-Time method with a null parameter value typically triggers a NullPointerException.

Fluent

The Date-Time API provides a fluent interface, making the code easy to read. Because most methods do not allow parameters with a null value and do not return a null value, method calls can be chained together and the resulting code can be quickly understood. For example:

```
LocalDate today = LocalDate.now();
LocalDate payday = today.with(TemporalAdjusters.lastDayOfMonth()).minusDays(2);
```

Immutable

Most of the classes in the Date-Time API create objects that are immutable, meaning that, after the object is created, it cannot be modified. To alter the value of an immutable object, a new object must be constructed as a modified copy of the original. This also means that the Date-Time API is, by definition, thread-safe. This affects the API in that most of the methods used to create date or time objects are prefixed with of, from, or with, rather than constructors, and there are no set methods. For example:

```
LocalDate dateOfBirth = LocalDate.of(2012, Month.MAY, 14);
LocalDate firstBirthday = dateOfBirth.plusYears(1);
```

Extensible

The Date-Time API is extensible wherever possible. For example, you can define your own time adjusters and queries, or build your own calendar system.

Date-Time

Packages

The Date-Time API consists of the primary package, java.time, & 4 subpackages:

`java.time`

The core of the API for representing date and time. It includes classes for date, time, date and time combined, time zones, instants, duration, and clocks. These classes are based on the calendar system defined in ISO-8601, and are immutable and thread-safe.

`java.time.chrono`

The API for representing calendar systems other than the default ISO-8601. You can also define your own calendar system. This tutorial does not cover this package in any detail.

`java.time.format`

Classes for formatting and parsing dates and times.

`java.time.temporal`

Extended API, primarily for framework and library writers, allowing interoperations between the date and time classes, querying, and adjustment. Fields (TemporalField and ChronoField) and units (TemporalUnit and ChronoUnit) are defined in this package.

`java.time.zone`

Classes that support time zones, offsets from time zones, and time zone rules. If working with time zones, most developers will need to use only ZonedDateTime, and ZoneId or ZoneOffset.

Date-Time

Method Naming Conventions

The Date-Time API offers a rich set of methods within a rich set of classes. The method names are made consistent between classes wherever possible. For example, many of the classes offer a now method that captures the date or time values of the current moment that are relevant to that class. There are from methods that allow conversion from one class to another.

There is also standardization regarding the method name prefixes. Because most of the classes in the Date-Time API are immutable, the API does not include setmethods. (After its creation, the value of an immutable object cannot be changed. The immutable equivalent of a set method is with.) The following table lists the commonly used prefixes:

PREFIX	Method Type	Use
of	static factory	Creates an instance where the factory is primarily validating the input parameters, not converting them.
from	static factory	Converts the input parameters to an instance of the target class, which may involve losing information from the input.
parse	static factory	Parses the input string to produce an instance of the target class.
format	instance	Uses the specified formatter to format the values in the temporal object to produce a string.
get	instance	Returns a part of the state of the target object.
is	instance	Queries the state of the target object.
with	instance	Returns a copy of the target object with one element changed; this is the immutable equivalent to a set method on a JavaBean.
plus	instance	Returns a copy of the target object with an amount of time added.
minus	instance	Returns a copy of the target object with an amount of time subtracted.
to	instance	Converts this object to another type.
at	instance	Combines this object with another.