JAVA 8 FEATURE

DATE 12 NOV 2021

1. EXPLORE DATE AND TIME API

Java provide the date and time functionality with the help of two packages java.time and java.util. The package java.time is introduced in Java 8, and the newly introduced classes tries to overcome the shortcomings of the legacy java.util.Date and java.util.Calendar classes.

Classical Date Time API Classes

The primary classes before Java 8 release were:

**Java.lang.System:** The class provides the currentTimeMillis() method that returns the current time in milliseconds. It shows the current date and time in milliseconds from January 1st 1970.

**java.util.Date:** It is used to show specific instant of time, with unit of millisecond.

**java.util.Calendar:** It is an abstract class that provides methods for converting between instances and manipulating the calendar fields in different ways.

**java.text.SimpleDateFormat:** It is a class that is used to format and parse the dates in a predefined manner or user defined pattern.

**java.util.TimeZone:** It represents a time zone offset, and also figures out daylight savings.

Drawbacks of existing Date/Time API's

1. **Thread safety:** The existing classes such as Date and Calendar does not provide thread safety. Hence it leads to hard-to-debug concurrency issues that are needed to be taken care by developers. The new Date and Time APIs of Java 8 provide thread safety and are immutable, hence avoiding the concurrency issue from developers.
2. **Bad API designing:** The classic Date and Calendar APIs does not provide methods to perform basic day-to-day functionalities. The Date and Time classes introduced in Java 8 are ISO-centric and provides number of different methods for performing operations regarding date, time, duration and periods.
3. **Difficult time zone handling:** To handle the time-zone using classic Date and Calendar classes is difficult because the developers were supposed to write the logic for it. With the new APIs, the time-zone handling can be easily done with Local and ZonedDate/Time APIs.

### New Date Time API in Java 8

The new date API helps to overcome the drawbacks mentioned above with the legacy classes. It includes the following classes:

**java.time.LocalDate:** It represents a year-month-day in the ISO calendar and is useful for representing a date without a time. It can be used to represent a date only information such as a birth date or wedding date.

**java.time.LocalTime:** It deals in time only. It is useful for representing human-based time of day, such as movie times, or the opening and closing times of the local library.

**java.time.LocalDateTime:** It handles both date and time, without a time zone. It is a combination of LocalDate with LocalTime.

**java.time.ZonedDateTime:** It combines the LocalDateTime class with the zone information given in ZoneId class. It represent a complete date time stamp along with timezone information.

**java.time.OffsetTime:** It handles time with a corresponding time zone offset from Greenwich/UTC, without a time zone ID.

**java.time.OffsetDateTime:** It handles a date and time with a corresponding time zone offset from Greenwich/UTC, without a time zone ID.

**java.time.Clock :** It provides access to the current instant, date and time in any given time-zone. Although the use of the Clock class is optional, this feature allows us to test your code for other time zones, or by using a fixed clock, where time does not change.

**java.time.Instant :** It represents the start of a nanosecond on the timeline (since EPOCH) and useful for generating a timestamp to represent machine time. An instant that occurs before the epoch has a negative value, and an instant that occurs after the epoch has a positive value.

**java.time.Duration :** Difference between two instants and measured in seconds or nanoseconds and does not use date-based constructs such as years, months, and days, though the class provides methods that convert to days, hours, and minutes.

**java.time.Period :** It is used to define the difference between dates in date-based values (years, months, days).

**java.time.ZoneId :** It states a time zone identifier and provides rules for converting between an Instant and a LocalDateTime.

**java.time.ZoneOffset :** It describe a time zone offset from Greenwich/UTC time.

**java.time.format.DateTimeFormatter :** It comes up with various predefined formatter, or we can define our own. It has parse() or format() method for parsing and formatting the date time values.