INTRODUCTION TO THE JAVA 8 DATE AND TIME API

JULIET CHINYERE NKWOR

WEB DEVELOPER

PM GLOBAL TECHNOLOGY SERVICES

https://queencoder.herokuapp.com/

CLASSES TO CREATE AND MANIPULATE CALENDAR DATA

- LocalDateTime,
- LocalDate,
- LocalTime,
- DateTimeFormatter

METHOD PREFIXES, TYPES, AND THEIR USES IN THE DATE AND TIME API IN JAVA 8

Prefixes	Method Type	Uses
of	Static	Creates an instance where the factory is primarily validating the input parameters, not converting them.
parse	static	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.
plus	instance	Returns a copy of the target object with an amount of time added.

WORKING WITH CALENDAR DATA

Create and manipulate calendar data using classes from

- Java.time.LocalDateTime
- Java.time.LocalDate
- Java.time.LocalTime
- Java.time.format.DateTimeFormatter

LocalDate

To store dates like a birthday or anniversary, visiting a place, or starting a job, school, or college, you don't need to store the time. LocalDate can be used to store dates like 2015-12-27 without time or time zones. LocalDate instances are immutable.

❑CREATING LOCALDATE

The LocalDate constructor is marked private, so you must use one of the factory methods to instantiate it. The static method of() accepts year, month, and day of month:

```
LocalDate date1 = LocalDate.of(2015, 12, 27);
LocalDate date2 = LocalDate.of(2015, Month.DECEMBER, 27);
```

LocalDate

To get the current date from the system clock, use the static method now():

```
LocalDate date3 = LocalDate.now();
```

You can also parse a string in the format 2016-02-27 to instantiate LocalDate. Here's an example:

```
LocalDate date2 = LocalDate.parse("2025-08-09");
```

LocalDate

- QUERYING LOCALDATE

You can use instance methods like get*XX*() to query LocalDate on its year, month, and date values.

```
LocalDate date = LocalDate.parse("2020-08-30");
System.out.println(date.getDayOfMonth());
System.out.println(date.getDayOfWeek());
System.out.println(date.getMonthValue());
System.out.println(date.getYear());
```

Output: The output of the preceding code looks like this:

```
30, SUNDAY, 8, 2020
```

You can use the instance methods isAfter() or isBefore() to determine whether a date is chronologically before or after another date:

LocalDate

❑ MANIPULATING LOCALDATE

The LocalDate class defines methods with the names minus*XX*(), plus*XX*(), and with*XX*() to manipulate the date values.

```
LocalDate bday = LocalDate.of(2052,03,10);
System.out.println(bday.minusWeeks(30));
System.out.println(bday.minusYears(1));
```

Output of the preceding code: 2051-08-13, 2051-03-10

The plus*XX*():

```
LocalDate launchCompany = LocalDate.of(2016,02,29);
System.out.println(launchCompany.plusDays(1));
System.out.println(launchCompany.plusMonths(1));
```

Output of the preceding code: 2016-03-01, 2016-03-29

LocalDate

- MANIPULATING LOCALDATE

The **with*XX*()** methods return a copy of the date instance replacing the specified day,

```
LocalDate firstSex = LocalDate.of(2036,02,28);
System.out.println(firstSex.withDayOfMonth(1));
System.out.println(firstSex.withYear(1));
```

Output: of the preceding code looks like this:
2036-02-01,
0001-02-28

LocalTime

To store times like breakfast, conference talk start time, or in-store sale end time, you can use LocalTime. It stores time in the format hours-minutes-seconds (without a time zone) and to nanosecond precision. LocalTime is also immutable.

- CREATING LOCALTIME

The LocalTime constructor is private, so you must use one of the factory methods to instantiate it. The static method of() accepts hours, minutes, seconds, and nanoseconds:

```
LocalTime timeHrsMin = LocalTime.of(12, 12);
LocalTime timeHrsMinSec = LocalTime.of(0, 12, 6);
LocalTime timeHrsMinSecNano = LocalTime.of(14, 7, 10, 998654578);
```

LocalTime

❑ CREATING LOCALTIME

- The of() method uses a 24-hour clock to specify the hour value.
- LocalTime doesn't define a method to pass a.m. or p.m. Use values 0–23 to define hours.
- To get the current time from the system clock, use the static method now():

```
LocalDate date3 = LocalTime.now();
```

- You can parse a string to instantiate LocalTime by using its static method parse(). You can either pass a string in the format 15:08:23 (hours:minutes:seconds)

```
LocalTime time = LocalTime.parse("15:08:23");
```

LocalTime

❑USING LOCALTIME CONSTANTS

You can use constants from the LocalTime class to work with predefined times:

- LocalTime.MIN—Minimum supported time, that is, 00:00
- LocalTime.MAX—Maximum supported time, that is, 23:59:59.999999999
- LocalTime.MIDNIGHT—Time when the day starts, that is, 00:00
- LocalTime.NOON—Noontime, that is, 12:00

❑QUERYING LOCALTIME

You can use instance methods like get*XX*() to query LocalTime on its hour, minutes, seconds, and nanoseconds. All these methods return an int value:

```
LocalTime time = LocalTime.of(16, 20, 12, 98547);
System.out.println(time.getHour());
System.out.println(time.getNano());
```

Here's the output: 16,
98547

LocalTime

❑ QUERYING LOCALTIME

- You can use the instance methods isAfter() and isBefore() to check whether a time is after or before the specified time. The following code outputs true:

```
LocalTime shreyaFinishTime = LocalTime.parse("17:09:04");
LocalTime paulFinishTime = LocalTime.parse("17:09:12");
if(shreyaFinishTime.isBefore(paulFinishTime))
System.out.println("Shreya wins");
else
System.out.println("Paul wins");
```

Output: Shreya wins

LocalTime

- MANIPULATING LOCALTIME

You can use the instance methods minusHours(), minusMinutes(), minusSeconds(), and minusNanos() to create and return a copy of LocalTime instances with the specified period subtracted.

```
LocalTime movieStartTime = LocalTime.parse("21:00:00");
int commuteMin = 35;
LocalTime shreyaStartTime = movieStartTime.minusMinutes(commuteMin);
System.out.println("Start by " + shreyaStartTime + " from office");
```

Here's the output of the preceding code: Start by 20:25 from office

The following example uses the addSeconds() and isAfter() methods to add seconds to a time and compares it with another time:

```
int worldRecord = 10;
LocalTime raceStartTime = LocalTime.of(8, 10, 55);
```

LocalTime

- MANIPULATING LOCALTIME

```
LocalTime raceEndTime = LocalTime.of(8, 11, 11);
if (raceStartTime.plusSeconds(worldRecord).isAfter(raceEndTime))
    System.out.println("New world record");
else
    System.out.println("Try harder");
```

The output of the preceding code looks like this:

The withHour(), withMinute(), withSecond(), and withNano() methods accept an int value and return a copy of LocalTime with the specified value altered.

```
  LocalTime startTime = LocalTime.of(5, 7, 9);
        if (startTime.getMinute() < 30)
startTime = startTime.withMinute(0); System.out.println(startTime);
```

Here's the output: 05:00:09

LocalDateTime

If you want to store both date and time (without the time zone), use the class Local- DateTime. It stores a value like 2050-06-18T14:20:30:908765 (year-month-dayThours :minutes:seconds:nanoseconds).

Instead of discussing individual methods of this class, here's an example that covers the important methods of this class:

```
LocalDateTime prizeCeremony = LocalDateTime.parse("2050-06-05T14:00:00");
LocalDateTime dateTimeNow = LocalDateTime.now();
if (prizeCeremony.getMonthValue() == 6)
System.out.println("Can't invite president");
else
    System.out.println("President invited");
```

LocalDateTime

```
LocalDateTime chiefGuestDeparture = LocalDateTime.parse("2050-06-05T14:30:00");
if (prizeCeremony.plusHours(2).isAfter(chiefGuestDeparture))
System.out.println("Chief Guest will leave before ceremony completes");

LocalDateTime eventMgrArrival = LocalDateTime.of(2050, 6, 5, 14, 30, 0);
if (eventMgrArrival.isAfter(prizeCeremony.minusHours(3)))
      System.out.println("Manager is supposed to arrive 3 hrs earlier")
```

DateTimeFormatter

Defined in the package java.time.format, the class DateTimeFormatter can be used to format and parse date and time objects. Defined in the package java.time.format, the class DateTimeFormatter can be used to format and parse date and time objects.

❑ INSTANTIATE OR ACCESS DATETIMEFORMATTER

You can instantiate or access a DateTimeFormatter object in multiple ways:

- By calling a static of*XXX* method, passing it a FormatStyle value
- By access public static fields of DateTimeFormatter
- By using the static method ofPattern and passing it a string value

- **OfLocalizedDate**
- **OfLocalizedTime**
- **ofLocalizedDateTime**

DateTimeFormatter

- By calling a static of*XXX* method, passing it a FormatStyle value

```
DateTimeFormatter formatter1 =
DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM);
DateTimeFormatter formatter2 = DateTimeFormatter.ofLocalizedTime(FormatStyle.FULL);
DateTimeFormatter formatter3 =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG);
DateTimeFormatter formatter4 =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT,
                                      FormatStyle.SHORT);
```

DateTimeFormatter

Examples of how **FormatStyle** affects formatting of a date (say, August 11, 2057) or time (say, 14 hours, 30 minutes, and 15 seconds) object

FormatStyle	**Example**
FormatStyle.FULL	Saturday, August 11, 2057
FormatStyle.LONG	August 11, 2057
FormatStyle.MEDIUM	Aug 11, 2057
FormatStyle.SHORT	8/11/57
FormatStyle.MEDIUM	2:30:15 PM
FormatStyle.SHORT	2:30 PM

DateTimeFormatter

- By access public static fields of DateTimeFormatter

Predefined formatters in the class **DateTimeFormatter** and an example of how they format a date (say, August 11, 2057) or time (say, 14 hours 30 minutes, and 15 seconds) object

FormatStyle	**Example**
BASIC_ISO_DATE	20570811
ISO_DATE/ISO_LOCAL_DATE	2057-08-11
ISO_TIME/ISO_LOCAL_TIME	14:30:15.312
ISO_DATE_TIME/ISO_LOCAL_DATE_TIME	2057-08-11T14:30:15.312

DateTimeFormatter

■ By using the static method ofPattern and passing it a string value

DateTimeFormatter formatter6= DateTimeFormatter.ofPattern("yyyy MM dd");

❑ PARSE DATE OR TIME OBJECTS USING DATETIMEFORMATTER

To format a date or time object, you can use either the instance format method in date/time objects or the instance format method in the DateTimeFormatter class.

DateTimeFormatter formatter = DateTimeFormatter.ofLocalizedDate(FormatStyle.LONG);
LocalDate date = LocalDate.of(2057,8,11);
System.out.println(formatter.**format**(date));

Outputs: August 11, 2057

DateTimeFormatter

❑ PARSE DATE OR TIME OBJECTS USING DATETIMEFORMATTER

Following are examples that use different patterns:

```
LocalDate date = LocalDate.of(2057,8,11); LocalTime time = LocalTime.of(14,30,15);
DateTimeFormatter d1 = DateTimeFormatter.ofPattern("y");
DateTimeFormatter d2 = DateTimeFormatter.ofPattern("YYYY");
DateTimeFormatter d3 = DateTimeFormatter.ofPattern("Y M D");
DateTimeFormatter t1 = DateTimeFormatter.ofPattern("H h m s");
DateTimeFormatter t2 = DateTimeFormatter.ofPattern("'Time now:'HH mm a");
System.out.println(d1.format(date)); System.out.println(d2.format(date));
System.out.println(d3.format(date)); System.out.println(d4.format(date));
System.out.println(t1.format(time)); System.out.println(t2.format(time));
```

Here's the output of the preceding code: 2057, 2057, 2057 8 223, 7, 14 2 30 15, Time now:14 30 PM

DateTimeFormatter

- PARSE DATE OR TIME OBJECTS USING DATETIMEFORMATTER

Let's work with the method parse of LocalDate, LocalTime, or LocalDateTime to parse a string value using a DateTimeFormatter, producing a date or time object:

```
DateTimeFormatter d1 = DateTimeFormatter.ofPattern("yyyy-MM-dd");
LocalDate date = LocalDate.parse("2057-01-29", d1 );
```

Summary

The Date and Time API in Java 8 simplifies how you work with the date and time classes. We worked with LocalDate, which is used to store only dates of the format 2016-08-14. LocalTime stores time in the format 14:09:65:23 (hours:minutes:seconds :nanoseconds). The class LocalDateTime stores both date and time. The class Date- TimeFormatter is used to format date and time using a predefined or custom format.