LocalDate

This lesson introduces the LocalDate class.

We'll cover the following

- a) Getting the current date
- b) Getting a specific date using of() method
- c) Getting a specific date using parse() method
- d) Adding days and months to a given date.
- e) Getting day of week
- f) Checking if a date is before or after a given date.

The new Date and Time API is moved to the `java.time` package and the Joda time format is followed.

The classes in the new API are immutable and, hence, thread-safe. The new API contains lots of classes that allow us to have more fine-grained control over our date and time representation.

Below is the list of all the classes in the `java.time` package.

Class Summary

Class	Description
Clock	A clock providing access to the current instant, date and time using a time-zone.
Duration	A time-based amount of time, such as '34.5 seconds'.
Instant	An instantaneous point on the time-line.
LocalDate	A date without a time-zone in the ISO-8601 calendar system, such as `2007-12-03`.
LocalDateTime	A date-time without a time-zone in the ISO-8601 calendar system, such as `2007-12-03T10:15:30`.
LocalTime	A time without a time-zone in the ISO-8601 calendar system, such as `10:15:30`.
MonthDay	A month-day in the ISO-8601 calendar system, such as `--12-03`.
OffsetDateTime	A date-time with an offset from UTC/Greenwich in the ISO-8601 calendar system, such as `2007-12-03T10:15:30+01:00`.
OffsetTime	A time with an offset from UTC/Greenwich in the ISO-8601 calendar system, such as `10:15:30+01:00`.
Period	A date-based amount of time in the ISO-8601 calendar system, such as '2 years, 3 months and 4 days'.
Year	A year in the ISO-8601 calendar system, such as `2007`.
YearMonth	A year-month in the ISO-8601 calendar system, such as `2007-12`.
ZonedDateTime	A date-time with a time-zone in the ISO-8601 calendar system, such as `2007-12-03T10:15:30+01:00 Europe/Paris`.
ZoneId	A time-zone ID, such as `Europe/Paris`.
ZoneOffset	A time-zone offset from Greenwich/UTC, such as `+02:00`.

In this lesson, we will look at the `LocalDate` class of the `java.time` package. This class holds only the date part without a time-zone in the ISO-8601 calendar system

It represents a date in ISO format (yyyy-MM-dd).

Let's look at some of the common use cases that can be solved through this class.

a) Getting the current date

We can get the current date by using the static `now()` method in the `LocalDate` class.

```
import java.time.LocalDate;

class DateTimeDemo {
    public static void main( String args[] ) {
        // now() method will return the current date.
        LocalDate date = LocalDate.now();
        System.out.println(date);
    }
}
```

b) Getting a specific date using of() method

We can get a specific date by using the static `of()` method in the `LocalDate` class. This method has two overloaded versions.

Each of them is shown in the example below.

```
import java.time.LocalDate;
import java.time.Month;

class DateTimeDemo {
    public static void main( String args[] ) {

        // of(int year, int month, int dayOfMonth)
        LocalDate date = LocalDate.of(2019, 05, 03);
        System.out.println(date);

        // of(int year, Month month, int dayOfMonth)
        date = LocalDate.of(2019, Month.AUGUST, 03);
        System.out.println(date);
    }
}
```

c) Getting a specific date using parse() method

We can get a specific date by using the static `parse()` method in the `LocalDate`

class. This method has two overloaded versions.

Each of them is shown in the example below.

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

class DateTimeDemo {
    public static void main( String args[] ) {

        // parse(CharSequence text)
        LocalDate date = LocalDate.parse("2015-02-12");
        System.out.println(date);

        // parse(CharSequence text, DateTimeFormatter formatter)
        date = LocalDate.parse("12/02/2012", DateTimeFormatter.ofPattern("MM/dd/yyyy"));
        System.out.println(date);
    }
}
```

d) Adding days and months to a given date.

We can use a whole range of addition operation methods that can be used for adding days, weeks, and months to a given date.

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;

class DateTimeDemo {
    public static void main( String args[] ) {

        // Adding 4 days to the given date.
        LocalDate date = LocalDate.parse("2015-02-12").plusDays(4);
        System.out.println(date);

         // Adding 4 months to the given date.
        date = LocalDate.parse("2015-02-12").plus(4, ChronoUnit.MONTHS);
        System.out.println(date);

    }
}
```

e) Getting day of week

We can get the day of the week using `getDayOfWeek()` method.

```
import java.time.DayOfWeek;
import java.time.LocalDate;


class DateTimeDemo {
    public static void main( String args[] ) {

        DayOfWeek dayOfWeek = LocalDate.parse("2017-04-06").getDayOfWeek();

        System.out.println(dayOfWeek);
    }
}
```

f) Checking if a date is before or after a given date.

We can check if a date comes before or after another given date by using the `isBefore()` and `isAfter()` methods.

```
import java.time.LocalDate;

class DateTimeDemo {
    public static void main( String args[] ) {

        // Using isBefore() to check if the date is before a given date.
        boolean isBefore = LocalDate.parse("2020-03-12")
                .isBefore(LocalDate.parse("2018-06-14"));
        System.out.println(isBefore);

        // Using isAfter() to check if the date is after a given date.
        boolean isAfter = LocalDate.parse("2020-03-12")
                .isAfter(LocalDate.parse("2018-06-14"));
        System.out.println(isAfter);

    }
}
```

This has been a basic introduction to the `LocalDate` class and its utilities. In the next lesson, we will look at `LocalTime`.