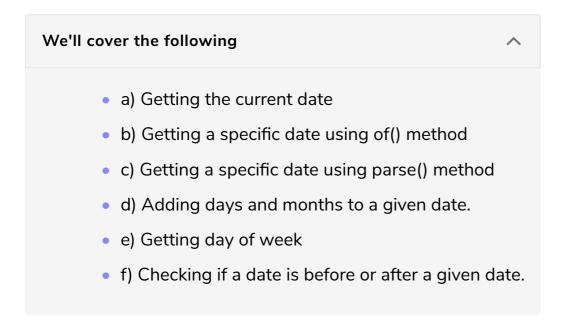
LocalDate

This lesson introduces the LocalDate class.



The new Date and Time API is moved to the <code>java.time</code> 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.

Summary	
	Description
	A clock providing access to the current instant, date and time using a time-zone.
ion	A time-based amount of time, such as '34.5 seconds'.
nt	An instantaneous point on the time-line.
Date	A date without a time-zone in the ISO-8601 calendar system, such as 2007-12-03.
DateTime	A date-time without a time-zone in the ISO-8601 calendar system, such as $2007 - 12 - 03T10 : 15 : 30$.
Time	A time without a time-zone in the ISO-8601 calendar system, such as 10:15:30.
hDay	A month-day in the ISO-8601 calendar system, such as12-03.
tDateTime	A date-time with an offset from UTC/Greenwich in the ISO-8601 calendar system, such as 2007-12-03T10:15:30+01:00.
tTime	A time with an offset from UTC/Greenwich in the ISO-8601 calendar system, such as $10:15:30+01:90$.
d	A date-based amount of time in the ISO-8601 calendar system, such as '2 years, 3 months and 4 days'.
	A year in the ISO-8601 calendar system, such as 2007.
lonth	A year-month in the ISO-8601 calendar system, such as 2007-12.
dDateTime	A date-time with a time-zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30+01:00 Europe/Paris.
d	A time-zone ID, such as Europe/Paris.
Offset	A time-zone offset from Greenwich/UTC, such as $+\theta 2:\theta \theta$.

In this lesson, we will look at the LocalDate class of the java.time package. This

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 <code>now()</code> method in the <code>LocalDate</code> class.



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.

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