LocalDate

The class LocalDate represents a single date in a **YYYY-MM-dd** format, such as 2017-11-25 or 2025-01-23 . It could be used to store any date: from your birthday to the day of the Apocalypse.

The class belongs to the java.time package.

Creating LocalDate and current time

After importing the class, an instance storing the current date can be created as below:
LocalDate now = LocalDate.now();

Of course, it is also possible to create an instance of LocalDate that represents a specific day of a year. It can be obtained by using either of the two special static methods: of and parse.

Here are two examples:
LocalDate date1 = LocalDate.of(2017, 11, 25); // 2017-11-25 (25 November 2017)
LocalDate date2 = LocalDate.parse("2017-11-25"); // 2017-11-25 (25 November 2017)

The numbering system is intuitive: the number of a month is a number from 1 to 12 inclusive, the first day in a month has the number 1.
The other useful way to create an instance of LocalDate is by indicating a year and the sequential number of a day in this year, like this:

LocalDate.ofYearDay(2017, 33); // 2017-02-02 (2 February 2017)

A number of a day in the year is an int number from 1 to 365-366 (depending on whether it is a leap year or not).
LocalDate.**ofYearDay**(2016, 365); // 2016-12-30 (30 December 2016)
LocalDate.**ofYearDay**(2017, 365); // 2017-12-31 (31 December 2017)

Be careful though, because an exception may occur when we deal with the 366th day:
LocalDate.**ofYearDay**(2016, 366); // 2016-12-31 (31 December 2016)
LocalDate.**ofYearDay**(2017, 366); // here an exception occurs, because the year is not a leap year

LocalDate: year, month, day and length

Let's now consider the following instance of LocalDate:

```
LocalDate date = LocalDate.of(2017, 11, 25); // 2017-11-25 (25 November 2017)
```

We can get the year, month, the day of a month, the day of a year:

```
int year = date.getYear(); // 2017
int month = date.getMonthValue(); // 11
int dayOfMonth = date.getDayOfMonth(); // 25
int dayOfYear = date.getDayOfYear();  // 329
```

It is also possible to get a length of the year and month:

```
int lenOfYear = date.lengthOfYear(); // 365
int lenOfMonth = date.lengthOfMonth(); // 30
```

Arithmetic methods of LocalDate

The class has other methods for adding, subtracting and altering a day, month and year. Let's create another LocalDate instance:

```
LocalDate date = LocalDate.of(2017, 1, 1); // 2017-01-01 (1 January 2017)
```

And take a look at how we can apply these methods:

```
LocalDate tomorrow = date.plusDays(1);    // 2017-01-02 (2 January 2017)
LocalDate yesterday = date.minusDays(1);  // 2016-12-31 (31 December 2016)
LocalDate inTwoYears = date.plusYears(2); // 2019-01-01 (1 January 2019)
LocalDate in2016 = date.withYear(2016);   // 2016-01-01 (1 January 2016)
```

Conclusion

As you can see, the LocalDate class provides plenty of helpful methods for working with dates. Don't hesitate to use them when needed!

The LocalDate class is immutable and its methods always return a new instance of the class.