

Theory: LocalDate

🕒 23 minutes

0 / 5 problems solved

Skip this topic

Start practicing

1704 users solved this topic. Latest completion was about 2 hours ago.

Time is a fundamental notion not only in human life but in programming as well. Of course, Java provides some tools for working with temporal entities. We'll start with such units of time measurement as days (or dates).

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.

§1. Creating LocalDate and current time

After importing the class, an instance storing the current date can be created as below:

```
1 | 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:

```
1 | LocalDate date1 = LocalDate.of(2017, 11, 25); // 2017-11-25 (25 November 2017)
2 | 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:

```
1 | 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).

```
1 | LocalDate.ofYearDay(2016, 365); // 2016-12-30 (30 December 2016)
2 | LocalDate.ofYearDay(2017, 365); // 2017-12-31 (31 December 2017)
```

Be careful though, because an exception may occur when we deal with the 366-th day:

```
1 | LocalDate.ofYearDay(2016, 366); // 2016-12-31 (31 December 2016)
2 |
LocalDate.ofYearDay(2017, 366); // here is an exception occurs, because the year is not leap
```

§2. LocalDate: year, month, day and length

Let's now consider the following instance of `LocalDate` :

```
1 | 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:

Current topic:

LocalDate

...

Topic depends on:

✓

Objects

Stage 3

...

Topic is required for:

LocalDateTime

...

Table of contents:

1

LocalDate

§1. Creating LocalDate and current time

§2. LocalDate: year, month, day and length

§3. Arithmetic methods of LocalDate

Feedback & Comments

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

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

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

§3. 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:

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

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

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

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.

 Report a typo

195 users liked this theory. **1** didn't like it. What about you?



Start practicing

[Comments \(9\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)