

# Theory: Comparing dates and time

🕒 22 minutes

0 / 5 problems solved

Skip this topic

Start practicing

660 users solved this topic. Latest completion was about 4 hours ago.

The classes `LocalDate`, `LocalTime`, `LocalDateTime` have methods for comparing their instances according to their position on the timeline. The methods compare instances as they go in chronological order (or the order of time).

## §1. The method compareTo

The method `compareTo` compares **this** instance and another one passed as the method's argument. It returns *0* if they are equal, a *negative* value if this instance is less than the other, and a *positive* value if this instance is greater.

Here is an example of comparing two instances of the class `LocalDate`:

```
1 | LocalDate date1 = LocalDate.parse("2017-01-02");
2 | LocalDate date2 = LocalDate.parse("2017-12-12");
3 |
4 | date1.compareTo(date1); // 0, date1 and date1 are equal
5 | date1.compareTo(date2); // -11, negative value => date1 is less than date2
6 |
date2.compareTo(date1); // 11, positive value => date2 is greater than date1
```

The class `LocalTime` has the same method, that returns either *0*, *1* or *-1*:

```
1 | LocalTime time1 = LocalTime.parse("15:30:10");
2 | LocalTime time2 = LocalTime.parse("17:50:30");
3 |
4 | time1.compareTo(time1); // 0, time1 and time1 are equal
5 | time1.compareTo(time2); // -1, time1 is less than time2
6 | time2.compareTo(time1); // 1, time2 is greater than time1
```

as well as `LocalDateTime`:

```
1 | LocalDateTime dateTime1 = LocalDateTime.parse("2017-01-01T20:30"); // 1 January 2017, 20:30
2 | LocalDateTime dateTime2 = LocalDateTime.parse("2017-01-02T23:00"); // 2 January 2017, 23:00
3 |
4 | dateTime1.compareTo(dateTime1); // 0, dateTime1 and dateTime are equal
5 | dateTime1.compareTo(dateTime2); // -1, dateTime1 is less than dateTime2
6 | dateTime2.compareTo(dateTime1); // 1, dateTime2 is greater than dateTime1
```

## §2. The methods isAfter, isBefore and isEqual

The classes have also some concise methods to compare dates and time on a timeline that return `boolean` value.

- The method `isAfter` returns `true`, only if this instance is strictly after another instance passed as the argument, otherwise, the method returns `false`.
- The method `isBefore` returns `true`, only if this instance is strictly before an instance passed as the argument, otherwise, the method returns `false`.
- The method `isEqual` returns `true`, if instances are equal, otherwise, the method returns `false`.

Here is an example of comparing two instances of the `LocalDate` class.

Current topic:

[Comparing dates and time](#) ...

Topic depends on:

✗ [LocalDateTime](#) ...

Table of contents:

[1 Comparing dates and time](#)

[§1. The method compareTo](#)

[§2. The methods isAfter, isBefore and isEqual](#)

[Feedback & Comments](#)

```
1 | LocalDate date1 = LocalDate.of(2017, 11, 30);
2 | LocalDate date2 = LocalDate.of(2017, 12, 1);
3 |
4 | date1.isEqual(date1); // true
5 | date1.isEqual(date2); // false
6 |
7 | date1.isBefore(date2); // true
8 | date1.isBefore(date1); // false
9 | date2.isBefore(date1); // false
10 |
11 |
12 | date2.isAfter(date1); // true
13 |
14 | date2.isAfter(date2); // false
15 |
16 | date1.isAfter(date2); // false
```

Here is an example of comparing two instances of the `LocalTime` class.

```
1 | LocalTime time1 = LocalTime.of(14, 20); // 14:20
2 | LocalTime time2 = LocalTime.of(15, 55); // 15:55
3 | LocalTime time3 = LocalTime.of(16, 40); // 16:40
4 |
5 | time1.isBefore(time2); // true
6 | time3.isBefore(time2); // false
7 | time3.isBefore(time3); // false
8 |
9 | time2.isAfter(time1); // true
10 |
11 | time2.isAfter(time3); // false
```

And here is an example of comparing two instances of the `LocalDateTime` class.

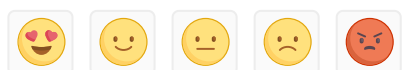
```
1 | LocalDateTime dateTime1 = LocalDateTime.parse("2017-12-01T21:30"); // 1 December 2017, 21:30
2 | LocalDateTime dateTime2 = LocalDateTime.parse("2017-12-02T21:30"); // 2 December 2017, 21:30
3 |
4 | dateTime1.isEqual(dateTime2); // false
5 | dateTime2.isEqual(dateTime2); // true
6 |
7 | dateTime1.isBefore(dateTime2); // true
8 | dateTime1.isAfter(dateTime2); // false
9 | dateTime2.isAfter(dateTime1); // true
```

Keep in mind, that the class `LocalTime` doesn't have the method `isEqual`. You can use the method `equals` instead.

We have learned now how to compare standard classes representing dates and time. Unsurprisingly, it can be done in almost the same unified way.

 Report a typo

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



Start practicing

[Comments \(5\)](#)

[Hints \(0\)](#)

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

[Show discussion](#)