Quiz yourself: Working with date and time in Java

Java contains a rich set of methods for dealing with date and time operations. This quiz tests your knowledge of that functionality.

by Mikalai Zaikin and Simon Roberts

May 24, 2021

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

This quiz will test your knowledge of date- and time-based events, including situations where there’s a combination of date and time in a single object.

In the following scenario, a Java developer is traveling from the Chicago O’Hare airport (ORD) in the United States to the Warsaw airport (WAW) in Poland. She is trying to calculate the duration of the flight but thinks that Poland might be shifting to Daylight Saving Time on the weekend of her journey. The information printed on her ticket is as follows:

Depart Chicago O’Hare
Scheduled: March 27, 2021 5:30 PM (17:30)

Arrive Warsaw
Scheduled: March 28, 2021 9:35 AM (09:35)

She has written the following incomplete code:
What code does she insert at line n1 to calculate the elapsed time expected between departure and arrival? Choose one.

A. 

```java
long timeInPlane =
    ChronoUnit.HOURS.between(ord, waw);
```

The answer is A.

B. 

```java
Period timeInPlane =
    Period.between(ord, ZoneId.of("America/Chicago"),
                    waw, ZoneId.of("Europe/Warsaw");
```

The answer is B.

C. 

```java
Period timeInPlane =
    Period.between(ord, waw);
```

The answer is C.

D. 

```java
Duration timeInPlane =
    Duration.between(ord, waw);
```

The answer is D.

**Answer.** The correct answer is option D. The programmer is correct: Poland did move its clocks forward an hour at 02:00 on the morning of March 28, 2021. However, her task was easy because the `ZonedDateTime` class specifically addresses the representation of not only time zone but also of legally prescribed changes of time, such as Daylight Saving Time. Consequently, the various means of calculating the offset between two `ZonedDateTime` objects take any such shifts into account automatically.

Ever since Java 8 introduced the Date and Time API, Java contains a database of all the historical Daylight Saving Time
changes from governments around the world. This database is updated with new information as it becomes available.

Let's review the options and see what they produce.

Option A is incomplete; it will print **Time in plane is: 9** to the console, which is the correct number of full hours, but it will entirely ignore the minutes. The departure time ends with 30 minutes and the arrival time ends with 35 minutes, so the number of hours is not exact. If the arrival time were at the same number of minutes past the hour, the output might appear correct, but it would be by luck, not because of technical validity. Consider also what would happen if the developer’s flight had been to India or another country that has time zones that vary from those of other countries by an amount that is not a whole number of hours. Because of this, option A is incorrect.

You could improve this option by rewriting the code as follows, which would give a result that is a bit better—9.083333333333334—although the format is not very helpful:

```java
double timeInPlane =
    ChronoUnit.MINUTES.between(ord, waw) / 60
```

The **Period** class measures time differences in days, months, and years, and the **between** method accepts only two arguments, which must be **LocalDate** objects. Because of this, both options B and C are incorrect.

Option D accurately denotes the time spent in the plane and takes into account the time zones the flight passes through and the Daylight Saving Time change in Poland that occurs midflight.

The output is in the rather odd form **PT9H5M**. This cryptic presentation specifies a “period of time of 9 hours and 5 minutes,” which is, in fact, the scheduled time for the flight.

The “PT” part of the output is hardcoded in the **Duration.toString()** method, and as you might expect, it's possible to extract the hours, minutes, and so on for a nicer presentation. The Java Date and Time API is very full featured, and it’s worth spending a little time browsing Oracle’s tutorial on the topic as well as the Javadoc so you can get the best out of this powerful API.

There's a final observation to make here about the nature of test questions and what you need to answer them. While we were preparing this question, we pondered whether options B and C were bad because they appeared to require rote learning of the API.

Questions that test the simple learning of facts, particularly facts that an IDE will tell you, are a bad idea, although a few exist in the real test. You'll see many such questions on the free sample tests that are abundant on the web. However, we decided to
keep these options because although they can be rejected by rote knowledge, they are best answered from a position of understanding.

In that spirit, let’s explore how you would address options B and C from a position of understanding the API, rather than from simple learning of that API’s contents.

It’s a key feature of the API that the Duration and Period classes represent different concepts. Specifically, Duration represents “physics time” and Period represents a human-calendar type of time in days—strictly in days, months, and years, but expressly without hours, minutes, seconds, and the like. From that, it’s a simple deduction that Period cannot be appropriate to the task of extracting hours and minutes. Therefore, you can eliminate options B and C immediately, without concern about whether the specific methods exist or not.

With a little luck, this persuades you that understanding will beat, or at least greatly augment, knowledge, even in a multiple-choice test.

Conclusion. The correct answer is option D.

Mikalai Zaikin
Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous Sun Certified Programmer for Java study guides by Kathy Sierra and Bert Bates.

Simon Roberts
Simon Roberts joined Sun Microsystems in time to teach Sun’s first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O’Reilly Safari Books Online service). He remains involved with Oracle’s Java certification projects.