Date and time

Operations with dates and times

From the classes seen in previous document to manage dates and times (*LocalDate*, *LocalTime* and *LocalDateTime*) we can do some specific operations with them. For instance, we can work with different time zones, so we can convert the date or time of a given zone of the planet into another, different zone. Also, we can make some particular operations, such as adding, substracting or comparing dates or times.

1. Working with time zones

In order to work with different time zones, we first need to identify the zone(s) we are interested in. `ZoneId` class provides a static method to list all the available zone identifiers. Also, it has a static `of` method to choose a specific zone:

```
Set<String> zones = ZoneId.getAvailableZoneIds();
for(String s: zones)
    System.out.println(s);
ZoneId madrid = ZoneId.of("Europe/Madrid");
```

Then, we can use `ZonedDateTime` class to convert a date or time given in a specific zone to another different zone. We use `of` and `withZoneSameInstant` methods from this class. For instance, this piece of code transform the current time in the zone *Europe/Madrid* into zone *Europe/Bucharest*:

```
ZonedDateTime dateZone =
    ZonedDateTime.of(LocalDateTime.now(), ZoneId.of("Europe/Madrid"));
ZonedDateTime anotherZone =
    dateZone.withZoneSameInstant(ZoneId.of("Europe/Bucharest"));
System.out.println("Now in Bucharest: " + anotherZone);
```

> **Exercise 1:**
>
> Create a program that asks the user to enter the zone and time where he/she was born, and transform this time into some other zone, such as Chicago ("America/Chicago") or Tokio ("Asia/Tokyo").

2. Operations with dates and times

Once we have our date or time objects (either *LocalDate*, *LocalTime* or *LocalDateTime*) we can make some additional operations with them, such as:

- Adding/Substracting parts of the date. For instance, add N days to a given date.
- Compare dates to determine which one is earlier or later
- Define a period between two dates. For instance, check how many days have passed since a given date.

2.1. Adding or substracting parts of a date

We have some methods available in *LocalDate*, *LocalTime* or *LocalDateTime* objects to add or substract an amount to/from them. For instance, we can use methods such as `plusDays`, `minusYears`, `plusHours` ...

```
LocalDate today = LocalDate.now();
LocalDate future = today.plusDays(15);
System.out.println("In 15 days it will be " + future);
```

Also, we can use the general methods `plus` and `minus`. In this case, we need to specify the quantity to be added or substracted, along with the unit. We can make use of `ChronoUnit` class from `java.time.temporal` package to specify these units.

```
LocalDateTime now = LocalDateTime.now();
LocalDateTime past = now.minus(3, ChronoUnit.HOURS);
System.out.println("3 hours ago it was " + past);
```

Note that all these operations return an object of the same type of the original that we took to add or substract.

2.2. Comparing dates

As *LocalDate*, *LocalTime* and *LocalDateTime* classes implement *Comparable* interface, we can easily compare objects of these classes by calling `compareTo` method:

```
LocalDate date1 = LocalDate.now();
LocalDate date2 = LocalDate.of(2022, 6, 10);

if (date1.compareTo(date2) < 0)
    System.out.println("Today is not 2022-6-10 yet");
```

2.3. Getting periods

We can use `Instant` and `Duration` classes to set the duration between two time instants. This can be particularly useful to measure the time it takes to complete a task.

```
Instant start = Instant.now();

// ... Task to be executed

Instant end = Instant.now();
Duration duration = Duration.between(start, end);
System.out.println("Task completed in " + duration.toMillis() + "ms");
```

Also, we can use `Period` class to determine the period between two different dates or times, in the desired unit. For instance, this piece of code determines the number of years and days between two dates, using different versions of `until` method from `Period` object:

```
LocalDate date1 = ...
LocalDate date2 = ...

Period period = date1.until(date2);
System.out.println(period.getYears());
System.out.println(date1.until(date2, ChronoUnit.DAYS));
```

> **Exercise 2:**
>
> Create a program called **NextBirthday** that asks the user his/her birth date, and show how old he/she is, and how many days are left until his/her next birthday.