Period and Duration classes

Period class representing a **date-based** unit and the Duration class representing a **time-based** unit.

Both of them belong to the java.time package.

Creating Period units

The Period class represents a period of time by **years**, **months,** and **days**.

If the unit includes all metrics, its unit format is P{year}Y{month}M{day}D, where {year}, {month} and {day} are placeholders for values.

- The first and probably the most common way of creating Period units is by using the of() static method.

```
Period period = Period.of(1995, 5, 23);
System.out.println(period); // P1995Y5M23D
```

- The next method allows us to get a Period unit in the form of a difference between two dates:

```
LocalDate startDate = LocalDate.of(1995, 5, 23);
LocalDate endDate = LocalDate.of(1995 , 8, 25);
Period period = Period.between(startDate, endDate);

System.out.println(period); // P3M2D
```

It will show negative values if the second argument is smaller, zero if they are equal, and a positive value if the second argument is larger.

- Another method accepts a text and parses it to a Period type.

```
// 23 days
System.out.println(Period.parse("P23D")); // P23D
// 5 months 23 days
System.out.println(Period.parse("P5M23D")); // P5M23D
// -1995 years 5 months 23 days
System.out.println(Period.parse("P-1995Y5M23D")); // P-1995Y5M23D
```

The same logic works for negative values. Here is a pattern you can use to mention that values are negative:

```
// -23 days
System.out.println(Period.parse("P-23D")); // P-23D
// -5 months -23 days
System.out.println(Period.parse("P-5M-23D")); // P-5M-23D
```

```
// -1995 years -5 months -23 days
System.out.println(Period.parse("P-1995Y-5M-23D")); // P-1995Y-5M-23D
// -1995 years -5 months -23 days
System.out.println(Period.parse("-P1995Y5M23D")); // P-1995Y-5M-23D
```

Extracting Period units

This class also provides us with some methods that are helpful when you want to extract a single date unit from a full date.

```
Period period  = Period.of(1995, 5, 23);

System.out.println(period.getYears()); // 1995
System.out.println(period.getMonths()); // 5
System.out.println(period.getDays()); // 23
```

Another method that performs the same operation is the get() method.

```
public static void main(String[] args) {
    Period period  = Period.of(1995, 5, 23);

    System.out.println(period.get(ChronoUnit.YEARS)); // 1995
    System.out.println(period.get(ChronoUnit.MONTHS)); // 5
    System.out.println(period.get(ChronoUnit.DAYS)); // 23
}
```

Note that ChronoUnit has fields for other units too, but the scope of this method only allows using the three of them mentioned above. If you use a unit such as ChronoUnit.WEEKS or any other, you will face an UnsupportedTemporalTypeException.

Operating with Period units

let's explore two pairs of methods designed for **adding** and **subtracting** date-based units, namely:

- addTo() and subtractFrom()
- plus() and minus()

```
Period period = Period.of(1, 1, 1);

System.out.println(period.addTo(LocalDate.of(1995, 5, 23))); // 1996-06-24
System.out.println(period.subtractFrom(LocalDate.of(1995, 5, 23))); //
1994-04-22
System.out.println(period); // P1Y1M1D
```

These pairs of methods perform similar operations but behave differently.

This means that the methods aren't designed to work with two Period units.

With their help, you will add or subtract a period to/from a Temporal variable which is a base interface of several classes
including LocalDate, LocalDateTime, ZonedDateTime, and so on.

n the other hand, the second pair plus() and minus(), is designed to operate with two Period type variables.
Period period = Period.of(1, 1, 1);

System.out.println(period.plus(Period.of(1995, 5, 23))); // P1996Y6M24D
System.out.println(period.minus(Period.of(1995, 5, 23))); // P-1994Y-4M-22D
System.out.println(period); // P1Y1M1D

The subtractFrom() method subtracts a period variable from a LocalDate while the minus() subtracts the second Period unit from the first one.

Creating Duration units

Duration represents a period of time storing the value
in **seconds** and **nanoseconds**.

They have similar methods that have similar roles.

Units of this class are created in the same way as the units of Period and have a similar unit format PT{hour}H{minute}M{second}S when the unit includes all metrics.

Duration durationOf = Duration.of(3, ChronoUnit.DAYS);
Duration durationOf1 = Duration.of(3, ChronoUnit.MINUTES);
Duration durationOf2 = Duration.of(3, ChronoUnit.NANOS);

System.out.println(durationOf); // PT72H
System.out.println(durationOf1); // PT3M
System.out.println(durationOf2); // PT0.000000003S

It accepts two arguments specifying the **amount** and the **time unit**.

The first parameter doesn't need an explanation, but you should be aware of an important limitation regarding the second one. Although ChronoUnit provides us with many units of time, here we can only use **accurate** units and **days**, which are counted as 24 hours (although the actual duration of a day is a bit longer).

All unsupported units will cause an UnsupportedTemporalTypeException. The

other two methods behave no differently from the same methods of the Period class.

```
LocalTime startTime = LocalTime.of(11, 45, 30);
LocalTime endTime = LocalTime.of(12, 50, 30);

System.out.println(Duration.between(startTime, endTime)); // PT1H5M
```

The parse() method also performs in the same way:

```
Duration duration1 = Duration.parse("PT1H20M");
Duration duration2 = Duration.parse("PT30M");

System.out.println(duration1); // PT1H20M
System.out.println(duration2); // PT30M
```

Unlike the Period class, there are fewer methods here to extract the desired unit from a Duration instance:

```
System.out.println(Duration.of(1, ChronoUnit.DAYS).getSeconds()); // 86400
System.out.println(Duration.of(1, ChronoUnit.HOURS).getSeconds()); // 3600
System.out.println(Duration.of(90, ChronoUnit.MINUTES).getSeconds()); //
5400
System.out.println(Duration.of(90, ChronoUnit.SECONDS).getSeconds()); // 90
System.out.println(Duration.of(90, ChronoUnit.SECONDS).getNano()); // 0
System.out.println(Duration.of(90, ChronoUnit.NANOS).getNano()); // 90
```

It has two methods operating on specified units: getSeconds() and getNano(), each returning its component in a unit. So, if the unit contains both seconds and nanoseconds, getSeconds() will return only seconds and the second one will return only nanoseconds.

```
Duration duration = Duration.ofSeconds(3675, 75);

System.out.println(duration); // PT1H1M15.000000075S
System.out.println(duration.getSeconds()); // 3675
System.out.println(duration.getNano()); // 75
```

Also, you can use the get() method if you don't want to specify the unit explicitly but pass the required unit as an argument when calling it:

```
Duration duration = Duration.of(10, ChronoUnit.MINUTES);
System.out.println(duration.get(ChronoUnit.SECONDS)); // 600
```

Like the similar method from the Period class, it will throw an exception if you pass an unsupported unit.

Operating with Duration units

The Duration class provides the same methods for adding and subtracting its units.

```
Duration duration = Duration.of(90, ChronoUnit.MINUTES);

System.out.println(duration.addTo(LocalTime.of(19, 5, 23))); // 20:35:23
System.out.println(duration.subtractFrom(LocalTime.of(19, 5, 23))); // 17:35:23
System.out.println(duration); // PT1H30M
```

If you take a closer look at the two code samples in this section and their results, you will see that the methods operate similarly to their Period equivalents.

```
Duration duration = Duration.of(90, ChronoUnit.MINUTES);

System.out.println(duration.plus(Duration.of(10, ChronoUnit.MINUTES))); // PT1H40M
System.out.println(duration.minus(Duration.of(10, ChronoUnit.MINUTES))); // PT1H20M
System.out.println(duration); // PT1H30M
```