Date and time

New classes for date and time management

From Java 8, we can use some additional classes to manage date and time. All these classes belong to `java.time` package. Depending on which part of the date we are interested in, Java provides three different classes to manage them all:

- `LocalDate` class lets us deal with dates made of day, month and year
- `LocalTime` class lets us focus on the *time* part, including hour, minute, second and even nanosecond.
- `LocalDateTime` class puts it all together, and lets us use a complete date, including the *time* part.

1. Creating dates and times

We can create a date or time of the current instant with a static method called `now` in these classes:

```
LocalDate dateNow = LocalDate.now();
LocalTime timeNow = LocalTime.now();
LocalDateTime fullNow = LocalDateTime.now();
```

We can also define a given date or time with a static method called `of`, specifying the date or time parts:

```
LocalDate myBirthday = LocalDate.of(1978, 4, 7);  // Year, month, day
LocalTime myTime = LocalTime.of(22, 30, 10);      // Hour, minute, second
LocalDateTime fullDate = LocalDateTime.of(1978, 4, 7, 22, 30, 10);
```

Keep in mind that month numbers start with 1 in this case, not with 0 as we have seen in *Calendar* class.

2. Getting parts of the date or time

Once we have our date or time created, we have some available methods to get the separate parts of this date or time, depending on the object we are managing (either *LocalDate*, *LocalTime* or *LocalDateTime*). For instance, if we are working with dates (either in *LocalDate* or *LocalDateTime* objects), we have methods such as `getYear`, `getMonthValue` or `getDayOfMonth` to get the year, month number or day, respectively:

```
int year = myBirthday.getYear();
int month = myBirthday.getMonthValue();
```

If we are managing times (either with *LocalTime* or *LocalDateTime*) we can use methods such as `getHour`, `getMinute` and `getSecond` to get these parts of the time:

```
int hour = myTime.getHour();
int second = myTime.getSecond();
```

3. Formatting dates and times

In order to format a date and show it with an appropriate format in the output, we can make use of `DateTimeFormatter` class, from `java.time.format` package. It has a static method called `ofPattern` that lets us specify the pattern of the date/time that we want to show. Then, we use an object of this type to call its `format` method, passing the date or time object as parameter. Let's see an example:

```
LocalDate myBirthday = LocalDate.of(1978, 4, 7);
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd - MMMM - yyyy");
System.out.println(formatter.format(myBirthday));
// Output: 07 - april - 1978
```

You can learn more about *DateTimeFormatter* class and the symbols that you can use in the patterns checking its official API.

4. Parsing dates and times from keyboard

If we want to read a date or time from the keyboard (or any text input, such as a text file or a text field in a JavaFX application) we can make use of `DateTimeFormatter` class to specify the pattern that we are expecting, along with `parse` method that lets us parse a string with the specified pattern. For instance, if we want the user to enter a dat with the format *dd/MM/yyyy*, we can do it this way:

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
Scanner sc = new Scanner(System.in);
LocalDate userDate = LocalDate.parse(sc.nextLine(), formatter);
```

If user input does not match the specified pattern, then a `DateTimeParseException` is thrown. Note that we can also use this same structure when dealing with *LocalTime* or *LocalDateTime*:

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy - H:m:s");
Scanner sc = new Scanner(System.in);
LocalDateTime userDateTime = LocalDateTime.parse(sc.nextLine(), formatter);
```

> **Exercise 1:**
>
> Create a program called **BirthdaySeason** that asks the user to enter his/her birth date with the format *day-month-year* and tells him/her if he/she was born in winter, spring, summer or autumn.

> **Exercise 2:**
>
> Create a program called **PoliteGreeting** that gets current time and tells *Good morning*, *Good afternoon*, *Good evening* or *Good night*. In order to detect which is the current day range, you can follow these rules:
>
> - *Good morning* if we are between 7 (included) and 12 (not included)
> - *Good afternoon* if we are between 12 (included) and 18 (not included)
> - *Good evening* if we are between 18 (included) and 21 (not included)
> - *Good night* otherwise

5. More information

You can read more information about the different methods and options available for each class in the official API:

- LocalDate
- LocalTime
- LocalDateTime