Java Time API

Java Time Api

- Java 8 introduced a new API for date time.
- This is what java.time does better:
  - Thread Safety – due to being immutable (like String).
  - Easier to understand and more fluent due to immutable classes.
  - Centered around ISO standards.
  - More utility methods.
  - Better handling of timezone logic.

Java Time Api

- Most commonly used classes are:
  - LocalDate:
    - Dates in ISO format --> 2020-01-09
  - LocalTime:
    - Time without a date. --> 08:00
  - LocalDateTime:
    - A date and time combined. --> 2020-01-09T08:00

LocalDate

Creating a LocalDate with the current date:

```
LocalDate today = LocalDate.now();
```

Creating a LocalDate using parse:

```
LocalDate nextBirthDay = LocalDate.parse("2020-09-11");
```

Creating a LocalDate using LocalDate.of():

```
LocalDate nextBirthDay = LocalDate.of(2020,9,11);
```

You can add and subtract from a LocalDate:

```
LocalDate myBirthDate = nextBirthDay.minusYears(44);
```

When you change a LocalDate it returns a **new object.**

LocalDate

```java
LocalDate nextBirthDay = LocalDate.of(2020,9,11);

Month september = nextBirthDay.getMonth();            //SEPTEMBER

int monthOfYear = nextBirthDay.getMonthValue();       //9

DayOfWeek friday = nextBirthDay.getDayOfWeek();       //FRIDAY

int dayOfMonth = nextBirthDay.getDayOfMonth();        //11

int dayOfYear = nextBirthDay.getDayOfYear();          //255

int year = nextBirthDay.getYear();                    //2020

Year objYear = Year.of(year);                         //2020

boolean isLeapYear = nextBirthDay.isLeapYear();       //true
```

LocalTime

```
LocalTime currentTime = LocalTime.now();                                  //Current time with nanosecond precision

System.out.println(currentTime.truncatedTo(ChronoUnit.MINUTES));    //11:47
System.out.println(currentTime.truncatedTo(ChronoUnit.SECONDS));    //11:47:45
System.out.println(currentTime);                                          //11:47:45.285825200
```

ChronoUnit is a handy enum that can be used in various methods. Used for when you want to define a specific **time unit**.

```
LocalTime lunch = LocalTime.of(12,0);                                     //12:00
LocalTime startTime = LocalTime.parse("08:15");                           //08:15
LocalTime endTime = LocalTime.parse("16:0");                              //DateTimeParseException
```

LocalTime

Just like LocalDate you can add and remove. Each change return a **new LocalTime object**

```
// 13:35 + 1h = 14:35 -> 14:35 + 5min = 14:40 -> 14:40 - 5s = 14:39:55
LocalTime localTime = LocalTime.parse("13:35").plusHours(1).plusMinutes(5).minusSeconds(5);
```

You can use various **getters** to extract data from a LocalTime object

```
LocalTime localTime = LocalTime.parse("13:35:59");
int hour = localTime.getHour();     //13
int minute = localTime.getMinute(); //35
int second = localTime.getSecond(); //59
```

LocalTime objects also have a **min** and **max** value, useful for database ”between” queries

```
LocalTime min = LocalTime.MIN; //00:00
LocalTime max = LocalTime.MAX; //23:59:59.99999999
LocalTime noon = LocalTime.NOON; //12:00
LocalTime midnight = LocalTime.MIDNIGHT; //00:00
```

LocalDateTime

Sometimes we need to work with a **combination of LocalDate and LocalTime**.

```java
LocalDateTime now = LocalDateTime.now(); //2020-01-03T09:10:36.252309200
```

LocalDate LocalTime

```java
LocalDateTime endOfWorkDay = LocalDateTime.parse("2020-01-03T17:00"); //2020-01-03T17:00

LocalDateTime meetingAppointment = LocalDateTime.of(2020,1,7,8,0);    //2020-01-07T08:00

LocalDate date = LocalDate.parse("2020-03-15");
LocalTime time = LocalTime.parse("09:30");
LocalDateTime dateTime = LocalDateTime.of(date,time);                 //2020-03-15T09:30
```

LocalDateTime

```
LocalDateTime fiveMinutesToMidnight =
        LocalDateTime.of(2020,12,31,23,55);                //2020-12-31T23:55

LocalDateTime pizzaTime = fiveMinutesToMidnight.plusDays(1) //2021-01-01T14:00
        .minusHours(10)
        .plusMinutes(5);
```

LocalDateTime objects are quite simple to manipulate. It **returns new LocalDateTime object** for each method called.

DateTimeFormatter

Using DateTimeFormatter you can change the way a **LocalDate**, **LocalTime** and a **LocalDateTime is presented**.

```
LocalDate march25 = LocalDate.parse("2020-03-25");

String basicISODateString = march25.format(DateTimeFormatter.BASIC_ISO_DATE);    //20200325

String isoDateString = march25.format(DateTimeFormatter.ISO_DATE);               //2020-03-25

String custom = march25.format(DateTimeFormatter.ofPattern("eeee dd MMM YYYY")); //onsdag 25 mars 2020
```

Here, a custom format is defined using a pattern.
More information here.

Period

With the period class you can measure quantity of time in terms of **year, month and days.**
Works with date components only.

```
LocalDate originalMeetingDate = LocalDate.parse("2020-11-11");
LocalDate postponedMeetingDate = originalMeetingDate.plus(Period.ofMonths(2)); //2021-01-11
```

```
LocalDateTime lectureStart = LocalDateTime.of(2020,1,7,8,0);
LocalDateTime newStart = lectureStart.plus(Period.ofDays(1));                 //2020-01-08T08:00
```

Period

```
LocalDate myBirthDate = LocalDate.parse("1976-09-11");
LocalDate today = LocalDate.parse("2020-01-03");

Period period = Period.between(myBirthDate, today);

int years = period.getYears();
int months = period.getMonths();
int days = period.getDays();

//43 years, 3 months, 23 days.
System.out.println(years + " years, " + months + " months, " + days + " days.");
```

Period can also calculate years, months and days between two dates…

Duration

Similar to Period, Duration is used to deal with Time in terms of seconds.

```java
LocalTime start = LocalTime.MIDNIGHT;
LocalTime now = LocalTime.parse("15:23");

Duration durationSinceStart = Duration.between(start, now);

long seconds = durationSinceStart.getSeconds();
System.out.println(seconds); //55380
```

Later versions of Java (Java 9+) supports getting hours and minutes as well.

Questions?