Time

New Austrian Coding School

Programming - memo

The Time API provides a **wide variety** of classes and methods that solve all **temporal problems**. No need to calculate anything any more by ourselves.

Continuous time

The concept of **continuous** time is based on the Unix time and it's represented as a **single incrementing number**.

- The *Instant* class represents a given **point of time** in nanoseconds from January the 1st of 1970, the **epoch** time.
- The *Duration* class represents an **amount of time** measured in nanoseconds.

```
System.out.println("--- INSTANT");

Instant now = Instant.now();
System.out.println("Instant: " + now);

Instant fiftySecondsLater = now.plusSeconds(50);
System.out.println("Instant after 50 seconds: " + fiftySecondsLater);

boolean isAfter = fiftySecondsLater.isAfter(now);
System.out.println("Is after: " + isAfter);


System.out.println("--- DURATION");

Instant begin = Instant.now();
Instant end = begin.plus(50, ChronoUnit.DAYS);
Duration duration = Duration.between(begin, end);
System.out.println("Duration: " + duration);
System.out.println("Duration in days: " + duration.toDays());
System.out.println("Duration in hours: " + duration.toHours());
System.out.println("Duration in minutes: " + duration.toMinutes());
```

Human time

The human time is based on concepts that we use in our **everyday life** such as day, hour or minute.

- We can represent days, months and years with *LocalDate*.
- We can represent hours, minutes and seconds with *LocalTime*.
- We can represent both together with *LocalDateTime*.
- These three time representations are **specific** to the **local time zone** wherever we are.
- The *Period* class represents a **descriptive amount of time** as understood by humans and not machines.

```
System.out.println("---LOCAL DATE");

LocalDate today = LocalDate.now();
System.out.println("Today: " + today);
LocalDate fiveWeeksAgo = today.minusWeeks(5);
System.out.println("Five weeks ago: " + fiveWeeksAgo);
LocalDate someDayThisYear = LocalDate.parse("2019-06-06");
System.out.println("This year is a leap year: " + someDayThisYear.isLeapYear());
System.out.println("Last year was a leap year: " +
someDayThisYear.minusYears(1).isLeapYear());
System.out.println("Next year will be a leap year: " +
someDayThisYear.plusYears(1).isLeapYear());

System.out.println("---LOCAL TIME");

LocalTime time = LocalTime.now();
System.out.println("Time: " + time);
LocalTime someTime = LocalTime.parse("13:37:33");
System.out.println("Some time: " + someTime);
System.out.println("Hours: " + someTime.getHour());
System.out.println("Minutes: " + someTime.getMinute());
System.out.println("Seconds: " + someTime.getSecond());

System.out.println("---LOCAL DATE TIME");

System.out.println("Right now: " + LocalDateTime.now());
LocalDateTime someMoment = LocalDateTime.of(someDayThisYear, someTime);
System.out.println("Some moment: " + someMoment);

System.out.println("---PERIOD");
Period period = Period.between(today, someDayThisYear);
System.out.println("Period: " + period);
System.out.println("Period in years: " + period.getYears());
System.out.println("Period in months: " + period.getMonths());
System.out.println("Period in days: " + period.getDays());
```

Zoned human time

Unfortunately, humans made the decision to have **different time zones**.

- The *ZoneId* class represents any one of the possible **available zones**.
- The *ZonedDateTime* class represents one *LocalDateTime* in a **particular zone**.

```
System.out.println("--- ZONED HUMAN TIME");

LocalDateTime now = LocalDateTime.now();
System.out.println("Now: " + now);
ZoneId here = ZoneId.systemDefault();
System.out.println("Here: " + here);
ZonedDateTime nowHere = now.atZone(here);
System.out.println("Now here: " + nowHere);
System.out.println("Available zones: " + ZoneId.getAvailableZoneIds());
ZoneId newYork = ZoneId.of("America/New_York");
ZonedDateTime nowInNewYork = nowHere.withZoneSameInstant(newYork);
System.out.println("Now in New York: " + nowInNewYork);
```

Time adjustments

There are many ways in which we can **modify** these **time measurements**. Their own methods and the classes *TemporalAdjusters* and *ChronoUnit* provide everything we need.

```
System.out.println("---CHRONO UNIT");

LocalDate today = LocalDate.now();
LocalDate after = today.plus(123456789, ChronoUnit.DAYS);
```

```
Period period = Period.between(today, after);
System.out.println("Period in years: " + period.getYears());
System.out.println("Period in months: " + period.getMonths());
System.out.println("Period in days: " + period.getDays());

long days = ChronoUnit.DAYS.between(today, after);
System.out.println("Period in days: " + days);

System.out.println("---TEMPORAL ADJUSTER");

LocalDate firstDayOf2020 = LocalDate.of(2019, 06,
06).with(TemporalAdjusters.firstDayOfNextYear());
System.out.println("First day of next year: " + firstDayOf2020);
System.out.println("First day of next year will be a: "+ firstDayOf2020.getDayOfWeek());
System.out.println("First Sunday of next year will be the: "+
firstDayOf2020.with(TemporalAdjusters.next(DayOfWeek.SUNDAY)));
```

Time formatting

Displaying the different **time measurements** in **different formats** can be done with the *DateTimeFormatter*.

```
System.out.println("---TIME FORMATTING");
LocalDateTime now = LocalDateTime.now();
System.out.println("Now in standard formatting: " + now);
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy HH|mm|ss");
String formatted = now.format(formatter);
System.out.println("Now formatted: " + formatted);
```