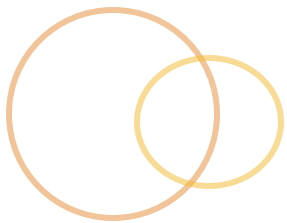
Java 8 Date / Time API

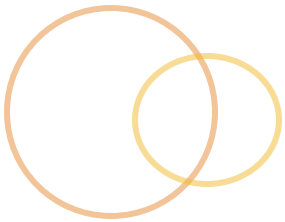
Tempus fugit!

Objectives



- Work with absolute points in time
- Work with periods of time
- Compare points in time
- Modify points in time based on time increments
- Modify points in time based on calendar aspects
- Convert dates and times to and from text
- Work with local dates and times

Representing Time



- The package `java.time` provides a comprehensive set of tools for handling time and date
  - Dates, potentially in many calendar systems
  - Dates and times, including timezones
  - Points in time without reference to any calendar system
  - Durations of time
  - Means for converting points in time between representations
  - Means for moving dates and times around by durations
  - Parsing and formatting times and dates

Absolute Points In Time

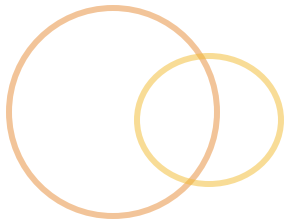
- `ZonedDateTime` and Instant represent an unambiguous point in time
  - Instant is just the moment in time
  - `ZonedDateTime` includes a notion about how this will be interpreted / presented, i.e. the time zone
- Both have nanosecond nominal accuracy

Creating ZonedDateTime

- `ZonedDateTime` can be constructed in several ways using static factory-type methods

now	Current instant, in the system default time zone
of(y, m, dom, h, m, s, ns, tz)	Instant specified by the arguments
of	Instant & timezone specified by arguments
ofInstant(inst, tz)	Instant & timezone specified by arguments
ofLocal	A LocalDate & Time, with a timezone
parse	Instant & timezone specified in text

Creating An Instant



- `Instant` can be created using static factory-type methods

now	Current instant
ofEpochSecond	Instant n seconds (& nanosecs) after Jan 1 1970 epoch
ofEpochMilli	Instant n milliseconds after Jan 1 1970 epoch
parse	Instant & timezone specified in text

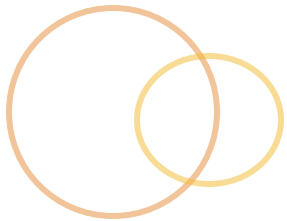
Representing Relative Time

- `Duration` and `Period` represent time differences:
  - 3 hours 15 minutes
  - 1 year and a day
  - 197.28 seconds
- `Duration` is “machine” based
  - 1 day is 24 hours
- `Period` is “calendar” based
  - 1 day might vary from 24 hours, depending on daylight saving etc.

Limitations Of Relative Time

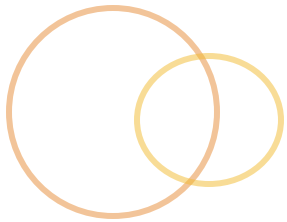
- Relative time represented by `Period` can be specified in terms of years, weeks, hours, etc.
- However, a period of 60 days cannot readily be converted to 2 months
  - Because a period, without reference to a starting point, doesn't know what calendar month(s) it covers
- Similarly, 365 days cannot be readily converted into 1 year as it might be a leap year
- `Period` has a `normalized` method, but this will only normalize elements up to days.

Time Units



- Several features of the Date Time API allow / require the use of time units to clarify a request
- `ChronoUnit` is an enumerated type that defines these, e.g.:
  - `ChronoUnit.CENTURIES`
  - `ChronoUnit.DAYS`
  - `ChronoUnit.HOURS`
  - Etc.
- These can be used, for example, in creating a `Duration`:
  - `Duration d = Duration.of(3, ChronoUnit.HOURS);`

Comparing Times

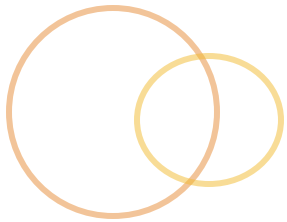


- Both `Instant` and `ZonedDateTime` (and others not yet introduced) implement `Temporal`
- The time difference between two `Temporal` values can be determined using `Duration.between(t1, t2)`
- Time "elements" between two points can be computed using `ChronoUnit`, e.g.:

```
long h = ChronoUnit.HOURS.between(t1, t2);
```

- Several other `between` methods exist in other classes, with more specific applications

Modifying Times



- `ZonedDateTime` can create a derived date using `plusXxx` and `minusXxx` methods

```
ZonedDateTime today = ZonedDateTime.now();
ZonedDateTime tomorrow = today.plusDays(1);
ZonedDateTime nextWeek = today.plusWeeks(1);
ZonedDateTime lastYear = today.minusYears(1);
Duration d4 = Duration.ofHours(77);
ZonedDateTime later = today.plus(d4);
```

- Note: ***most date / time API elements are immutable***, so modification behaviors create new objects; don't forget to store them!

More Time Modification

- ZonedDateTime also allows adjusting single elements of the represented time; e.g.:

```
ZonedDateTime here =
  ZonedDateTime.of(2015, 3, 8, 1, 55, 0, 0,
    ZoneId.of("America/Denver"));

ZonedDateTime ny =
  before.withZoneSameInstant(
    ZoneId.of("America/New_York"));
```

- Refers to the same moment, in a different time zone

More Time Modification

- The withXxx methods also allow changing the time

```
ZonedDateTime fiveAm = today.withHour(5);
```

Advanced Date Modification

- Humans often make date modification in less purely mathematical terms, e.g. “a week on Friday”, or “on the third Monday of the month”
- Changes such as these are supported by the `TemporalAdjuster` interface, along with utility implementations available from the `TemporalAdjusters` class

Advanced Date Modification

```
ZonedDateTime janOne =
  ZonedDateTime.of(2015, 1, 1, 0, 0, 0, 0,
    ZoneId.of("America/Denver"));

ZonedDateTime firstFriday = janOne.with(
    TemporalAdjusters.dayOfWeekInMonth(1,
                                       DayOfWeek.FRIDAY));

ZonedDateTime nextMonth = janOne.with(
    TemporalAdjusters.firstDayOfNextMonth());
```

Formatting And Parsing

- The `DateTimeFormatter` class is a configurable tool for formatting date/time objects as text, and parsing text into date/time objects
- Many ISO standard formats are supported directly as constants, but arbitrary formats can be created from template strings

Using a DateTimeFormatter

- `DateTimeFormatter` has several static methods for creating formatter objects suitable for different data / time object types

```
DateTimeFormatter dtf =
  DateTimeFormatter.ofLocalizedDateTime(
    FormatStyle.MEDIUM);
ZonedDateTime now = ZonedDateTime.now();
System.out.println("> " + dtf.format(now));
```

- Produces something like:

```
May 4, 2015 1:17:26 PM
```

Using a DateTimeFormatter

- Parse operations:

Parsing creates “internal” objects for data/time.

```
TemporalAccessor ta =
  dtf.parse("Jul 20, 1969 8:18:00 PM");

ZonedDateTime landing =
  LocalDateTime.from(ta).atZone(ZoneId.of("UTC"));

System.out.println("> " + landing);
```

If desired, convert to a specific date/time type

Arbitrary Date / Time Formats

- If the ISO standard date / time formats do not suit, `DateTimeFormatter` allows specification using template text (much like `printf`)

```
DateTimeFormatter dtf2 =
   DateTimeFormatter.ofPattern(
      "HH:mm:ss MMMM d, yyyy");
```

Format characters are defined in the API documentation

```
System.out.println(dtf2.format(now));
```

- Produces something like:

```
14:09:28 March 20, 2013
```

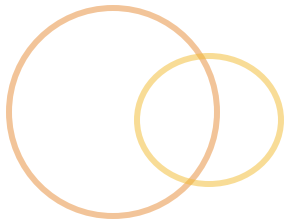
- This formatter can parse too

Arbitrary Date / Time Formats

- The format specification characters can typically be repeated. Repetitions are interpreted as changing the width and/or style of the representation
  - “yy” → 15
  - “yyyy” → 2015
  - “e” → 2
  - “ee” → 02
  - “eee” → Mon
  - “eeee” → Monday

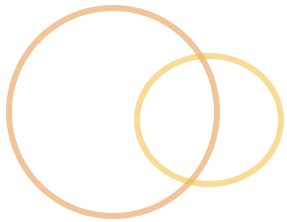
The API documentation indicates the format variations that are possible

Local Points In Time



- The date / time API can also describe dates and times in the local time zone
  - These might be simpler to work with, but might be inconvenient if the program is later modified to a global audience
- Three classes:
  - `LocalDate`
  - `LocalTime`
  - `LocalDateTime`

Local Points In Time



- Local time/dates support most of the conversions and adjustments that are applicable to `ZonedDateTime`
- They can be converted to `ZonedDateTime` given a time zone:

```
zdt = ldt.atZone(zoneId);
```

- They can be extracted from `ZonedDateTime` or `Instant`

```
ldt = LocalDateTime.ofInstant(inst, zoneId);
ldt1 = zdt.toLocalDateTime();
```

Limitations Of Local Date / Time

- Local date and time objects are missing some time information
  - They have no timezone
  - A `LocalDate` has no time
  - A `LocalTime` has no date
- Some processes, including data extraction, and formatting, might try to access these missing items
  - This will throw an exception
- Determine if an item is available using the `isSupported` methods

Using Legacy Date / Time Objects

- Several methods exist facilitating using the new date / time API with code already using older APIs

Conversion Class & Method Name
`Calendar.toInstant`
`GregorianCalendar.toZonedDateTime`
`GregorianCalendar.from(ZonedDateTime)`
`Date.from(Instant)`
`Date.toInstant`
`TimeZone.toZoneId`

Lab Exercise

- Suppose you are taking an airline flight from Los Angeles to New York. The flight is scheduled to leave at 8:24 am and arrive at 6:25 pm Calculate the total duration of the flight
- The flight has a layover in Denver. The first leg of the flight is scheduled to take 2 hours 25 minutes, and the layover is 41 minutes. What is the departure time of the second leg of the trip?

Lab Exercise

- Calculate the day of the week on which you were born
- Calculate the number of days you've been alive
- Your friend has scheduled a party next Thursday, calculate the date of that party