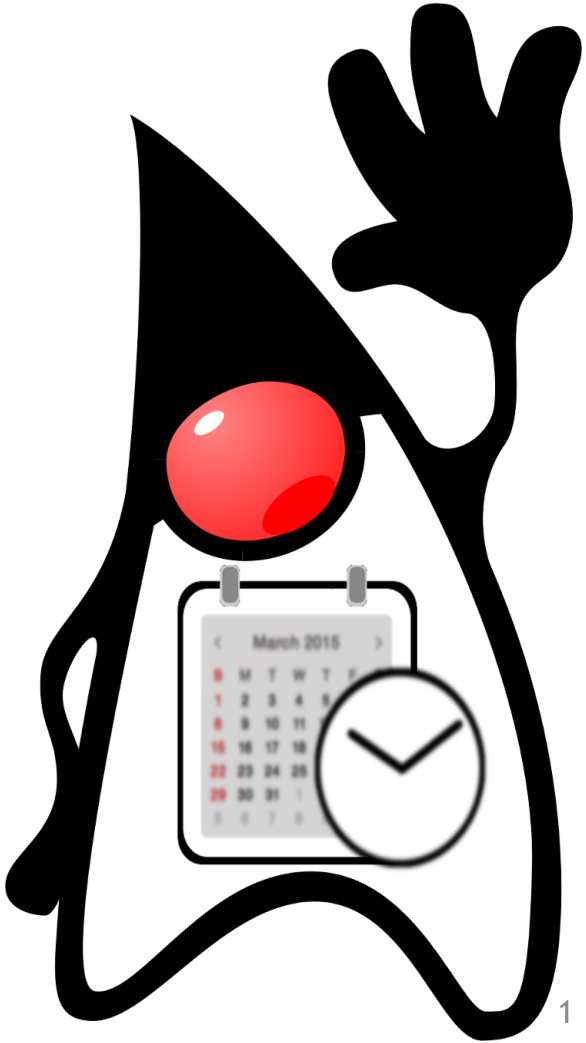
Java 8 Time

Chandra Guntur

@CGuntur



Prerequisites

1. JDK 8u51 or above (installed, path setup).

2. IDE setup with JDK8.

3. Working knowledge of Java and
   - `java.util.Date`
   - `java.util.Calendar`

Background : Issues With the Date

`java.util.Date`:

- **is mutable-only** - Date instances are not immutable.
- **has concurrency problems** - Date instances are not threadsafe.
- **has incorrect naming** - Date is not a "date" but a "timestamp".
- **lacks convention** - Days start with 1, months with 0 and yearswith 1900.
- **lacks fluency** - Cannot create durations (a quarter, 5 minutes) OR combos(year+month, date without seconds) etc.

Additional observations:

- `System.currentTimeInMillis()` is not accurate and can return same value for multiple sequential calls.
- `java.util.Date` vs. `java.sql.Date` - SQL flavor is only a DATE with no time.
- `java.sql.Timestamp` - SQL flavor replicating java.util.Date but additionally storing nanoseconds.

Background : Issues With Calendar

`java.util.Calendar`:

- **lacks clarity** - Mix of dates and times.
- **has confusing timezone support** - Not very easy to switch timezones, offsets etc.
- **has severe formatting hurdles** - SimpleDateFormat and Calendar do not interoperate well.
- **presents extension hardships** - New calendar systems are created by extending Calendar.

*"Calendar is the **Ravenous Bugblatter Beast of Traal** for Date APIs"* - Chandra Guntur (circa 2000).

What was proposed to fix this?

JSR-310: Date and Time API
Excerpts from JSR-310 (https://jcp.org/en/jsr/detail?id=310)

"The main goal is to build upon the lessons learned from the first two APIs (Date and Calendar) in Java SE, providing a more **advanced and comprehensive model for date and time manipulation.**"

"The new API will be targeted at all applications needing a data model for dates and times. This model will go beyond classes to replace Date and Calendar, to **include representations of date without time, time without date, durations and intervals.**"

"The new API will also tackle related date and time issues. These include **formatting and parsing, taking into account the ISO8601 standard** and its implementations, such as XML."

"The final goal of the new API is to be **simple to use**. The API will need to contain **some powerful features**, but these must **not be allowed to obscure the standard use cases**. Part of being easy to use includes interaction with the existing Date and Calendar classes ..."

Origins

JSR-310 is “inspired from” the very popular **Joda-Time** library by ***Stephen Colebourne***, who is also the lead on JSR-310.

JSR-310 was a means to both overcome the shortcomings as well as refactor portions of the **Joda-Time**. http://blog.joda.org/2009/11/why-jsr-310-isn-joda-time_4941.html.

Check out the cheatsheet from converting from **Joda-Time** to java.time apis: http://blog.joda.org/2014/11/convertingfrom-joda-time-to-javatime.html

Also check out ***Meno Hochschild***’s (author of the Time4J library) response at stack overflow: http://stackoverflow.com/questions/24631909/differences-between-java-8-date-time-apijava-time-and-joda-time

Java time API

Dates and Times

Simple Date and Time ‘containers’

`Instant` stores a **numeric timestamp from Java epoch**, + nanoseconds.

`LocalDate` stores a **date without a time portion** (calendar date).

`LocalTime` stores a **time without a date portion** (wall clock).

`LocalDateTime` stores a **date and time** (LocalDate + Local Time).

`ZonedDateTime` stores a **date and time with a time-zone.**

`OffsetTime` stores a **time and offset from UTC without a date.**

`OffsetDateTime` stores a **date with time and offset from UTC.**

Java time API - (continued)

Ranges and Partials

Spans and ranges of temporality

`Duration` models **time in nanoseconds** for time intervals. (e.g. 5 mins)

`Period` models amount of **time in years, months and/or days.** (e.g. 2 Days)

`Month` stores a **month on its own.** (e.g. MARCH)

`MonthDay` stores a **month and day without a year or time** (e.g. date of birth)

`Year` stores a **year on its own.** (e.g. 2015)

`YearMonth` stores a **year and month without a day or time.** (e.g. credit card expiry)

`DayOfWeek` stores a **day-of-week on its own.** (e.g. WEDNESDAY)

Java time API - (continued)

Chronology

A calendar system to organize and identify dates

`Chronology` is a factory to create or fetch pre-built calendar systems.
Default is `IsoChronology` (e.g. `ThaiBuddhistChronology`).

`ChronoLocalDate` stores a **date without a time** in an arbitrary chronology.

`ChronoLocalDateTime` stores a **date and time** in an arbitrary chronology.

`ChronoZonedDateTime` stores a **date, time and timezone** in an arbitrary chronology.

`ChronoPeriod` models a **span on days/time** for use in an arbitrary chronology.

`Era` stores a **timeline**
[typically two per Chronology, but some have more eras].

Date and Time Usage

	Type	Year	Month	Date	Hour	Minute	Second (with nano)	Zone Offset	Zone ID	toString()
Date and Time	Instant									1999-01-12T12:00:00.747Z
	LocalDate									1999-01-12
	LocalTime									12:00:00.747
	LocalDateTime									1999-01-12T12:00:00.747
	ZonedDateTime									1999-01-12T12:00:00.747-05:00 [America/New_York]
	OffsetTime									12:00:00.747-05:00
	OffsetDateTime									1999-01-12T12:00:00.747-05:00
Ranges	Duration									P22H
	Period									P15D
Partials	*Month*		*							*JANUARY*
	MonthDay									-01-12
	Year									1999
	YearMonth									1999-01
	DayOfWeek			*						*TUESDAY*

Method name conventions

Prefix	Method Type	Description	Example
`of`	static factory	Factory method to obtain an instance with provided parameters - validates and builds with no conversion.	`LocalDate.of(…)` or `Instant.ofEpochSecond(…)`
`from`	static factory	Factory method to obtain an instance with provided parameters - validates, converts and builds.	`LocalDateTime.from(…)` or `OffsetTime.from(…)`
`parse`	static factory	Factory method to obtain an instance with provided CharSequence parameters, by parsing the content.	`LocalDate.parse(…)` or `OffsetDateTime.parse(…)`
`format`	instance	Formats the instance with provided formatter.	`localDate.format(formatter)`
`get`	instance	Return part of the state of the target temporal object.	`localDate.getDayOfWeek()`
`is`	instance	Queries a part of the state of the target temporal object.	`localTime.isAfter(…)`
`with`	instance	Returns a copy of the immutable temporal object with a portion altered. Alternate for a `set` operation.	`offsetTime.withHour(…)`
`plus`	instance	Return a copy of the temporal object with an added time.	`localDate.plusWeeks(…)`
`minus`	instance	Return a copy of the temporal object with subtracted time.	`localTime.minusSeconds(…)`
`to`	instance	Convert the temporal object into a new temporal object of another type.	`localDateTime.toLocalDate(…)`
`at`	instance	Combine the temporal object into a new temporal object with supplied paraeters.	`localDate.atTime(…)`

JDBC Support

No public JDBC API changes - call `setObject` and `getObject` to directly use Java 8 time.

Following conversions are implicit with JDBC 4.2:

ANSI SQL	Java SE 8
DATE	`LocalDate`
TIME	`LocalTime`
TIMESTAMP	`LocalDateTime`
TIMESTAMP	`Instant`
TIME WITH TIMEZONE	`OffsetTime`
TIMESTAMP WITH TIMEZONE	`OffsetDateTime`

Workshop time