Date & Time in Java 8

Richard Warburton
Raoul-Gabriel Urma

ITERATR LEARNING

Outline

1. The problem with existing date time libraries
2. The core classes LocalDate/LocalTime
3. Common API Patterns
4. Live Coding Example

The problem with existing date time libraries

Take 1

Java Date

- In the language since January 23, 1996
- Mutability
- Date is an instant in time
- Additional classes for SQL required
- No Timezones

Take 2

Calendar

- Still mutable
- Not thread safe
- Can’t format dates directly
- Arithmetic operations on dates tricky
- Heavyweight memory consumption

Highlighting the Problem

```
Date date = new Date(2007, 12, 13, 16, 40);


TimeZone zone =
        TimeZone.getTimeZone("Asia/HongKong");


Calendar cal = new GregorianCalendar(date, zone);
DateFormat fm = new SimpleDateFormat("HH:mm Z");
String str = fm.format(cal);
```

Highlighting the Problem

```
Date date = new Date(2007, 12, 13, 16, 40);


TimeZone zone =
        TimeZone.getTimeZone("Asia/Hong_Kong");


Calendar cal = new GregorianCalendar(date, zone);
DateFormat fm = new SimpleDateFormat("HH:mm Z");
String str = fm.format(cal);
```

Take 3

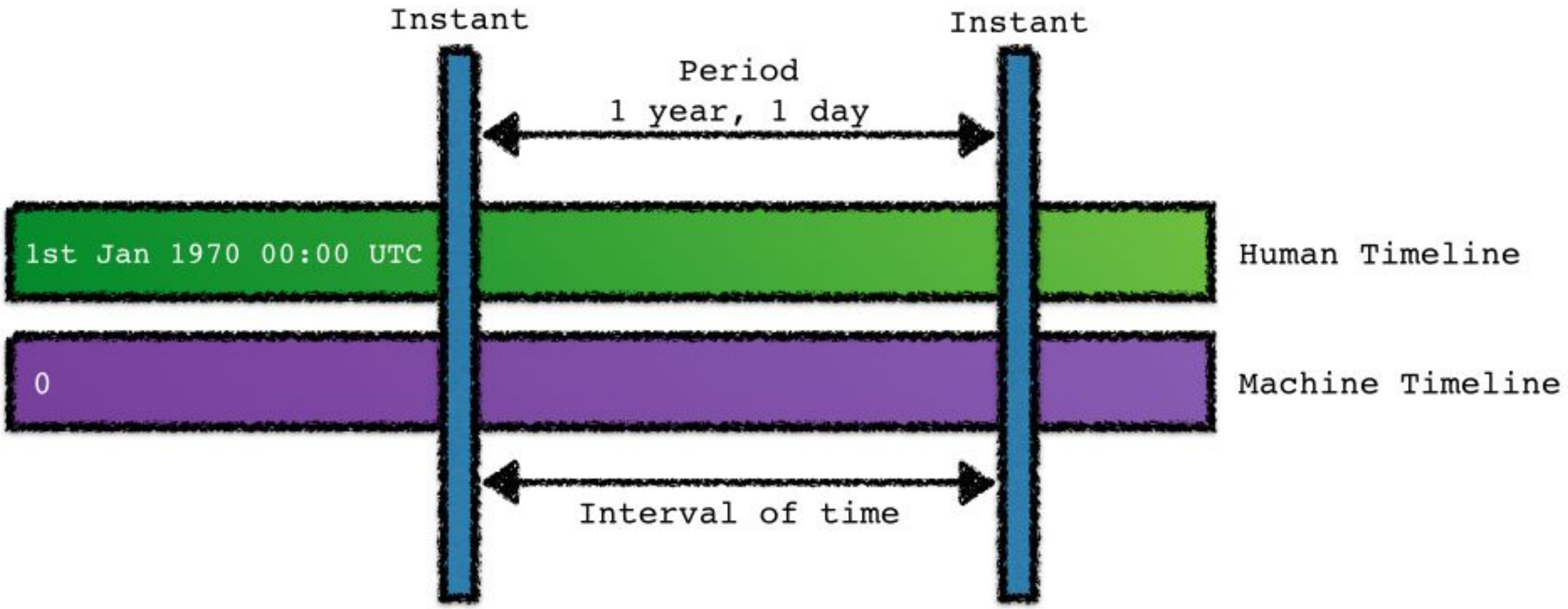
Joda Time

- Popular Open Source API
- Challenged the core Java APIs
- Introduced new required concepts

New Concepts

- Instant
  - Precise moment in time
  - Milliseconds since epoch

- Duration
  - Time in milliseconds between two points.


Instant
Instant
Period
1 year, 1 day
1st Jan 1970 00:00 UTC
Human Timeline
0
Machine Timeline
Interval of time

New Concepts

- Period
  - A period of time defined in terms of human readable fields, eg 1 Month
- Chronology
  - A calculation engine supporting rules of a specific calendar

Problems

- Nulls
  - Joda Time treats nulls as zero
  - Can lead to unexpected results from bugs
  - Easy to miss a problem

- Pluggable Chronologies
  - Powerful for multiple calendaring systems
  - How do you know what you are using?
  - How do you extend?

Problems

- Human/Machine Timeline
  - Machine counts from 0
  - Humans have logical dates
- Internal Implementation
  - Could be improved
- Weak Typing
  - Lots of constructors take `Object`

Quiz

What is the range of values returned by this method in Joda Time?

```
int month = dateTime.getMonthOfYear();
```

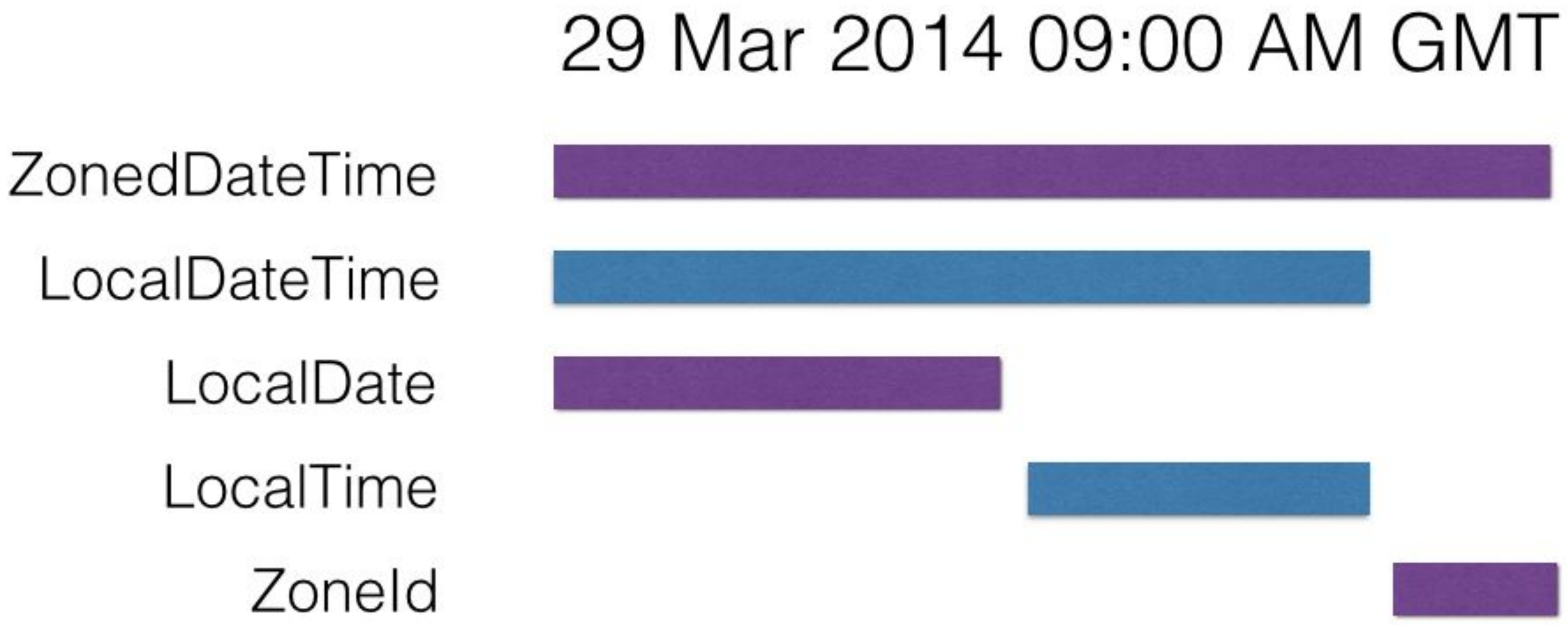
Answer

It depends on the chronology set, mostly we assume ISO.

```
//If chronology is ISO, month = 12
//If chronology is Coptic, month = 13
int month = dateTime.getMonthOfYear();
```

The core classes LocalDate/LocalTime

`java.time` classes



Zone Id vs Offset

- `java.time.ZoneId`
  - Identifier for a time zone
  - CEST or Europe/Amsterdam
  - Can resolve to an offset **at** a point in time

- `java.time.ZoneOffset`
  - Duration of time away from UTC
  - Many-to-many relationship with Ids

Duration vs Period

- `Duration`
    - Time based distance on the timeline
    - eg: 2 seconds + 3 nanoseconds

- `Period`
    - Date based distance on the timeline
    - eg: 3 weeks + 2 days

- NB: Getters return fields, don’t convert the period/duration to that field

Common API Patterns

Design Goals

- Domain Driven Design
- Immutable implementations
- Limited set of calendaring systems, but extendable to others using chronologies
- More expressive than `Date/Calendar` in order to better model the problem domain

Domain Models

- Many developers currently simply use Date
- Date is more representative of an instant
- Strange impact with daylight saving
- Move towards idea of having classes to represent each scenario

Immutability

- Once values are set they never change
- Mutability can have consequences in both multi threaded and single threaded code
- Existing formatters are not thread safe
- Developers using dates as part of their API can accidentally update values unexpectedly

Calendaring Systems

- API designed to meet what most developers require from a calendaring system
- Also has support for other major calendaring systems (provided in java.time.chrono)
- It is also possible to write your own calendaring systems

Framework Integration

- Not everything supports the Java 8 types yet

- Sometimes require extra hooks

```
org.hibernate.hibernate-java8
```

ANSI SQL Types

DATE	LocalDate
TIME	LocalTime
TIMESTAMP	LocalDateTime
TIME WITH TIMEZONE	OffsetTime
TIMESTAMP WITH TIMEZONE	OffsetDateTime

Live Coding: TripExample

Summary

Summary

- The existing Java classes exhibit poor usability characteristics
- Java 8 introduces a new Date & Time API
  - Immutable
  - Domain Driven Design
  - Supports different Calendaring Systems

(Optional) Exercise

Run the test at:

`com.java_8_training.problems.datetime.TestBirthdayDiary`

You will need to modify the diary at:

`BirthdayDiary`

The End

`java.time` overview

- `java.time`
  - The core classes and most frequently used
- `java.time.chrono`
  - Alternative calendaring systems
- `java.time.format`
  - Formatting and parsing tools

`java.time` overview

- `java.time.temporal`
  - Interfaces required for core classes

- `java.time.zone`
  - Underlying time zone rules, not often used