Java 8 Date & Time API!

Example

How many bugs in this code?

```
Date date = new Date(2010, 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);
```

Example

6 bugs in the code!

```
Date date = new Date(2010, 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);
```

Overview

- Review the current date and time API
- Understand date and time concepts
- Take a look at the new date and time API

Problems Getting a Date

```
System.out.println(new Date(12, 12, 12));
// Sun Jan 12 00:00:00 EST 1913
```

Several problems here:

1. Which 12 is for which date field?
2. Month 12 is December, right? No. January
3. Year 12 is 12 CE, right? Wrong. 1913
4. Wait - there is a time in a date?
5. More than that, there is a time zone

A Sorry Implementation

- Conceptually an instant, not a date
- Properties have random offsets
  - Some zero-based, like month and hours
  - Some one-based, like day of the month
  - Year has an offset of 1900
- Mutable, not thread-safe
- Not internationalizable
- Millisecond granularity
- Does not reflect UTC

Backstory

- Date was the work of James Gosling and Arthur van Hoff
- Added in JDK 1.0, mostly deprecated in JDK 1.1, never removed
- IBM donated Calendar code to Sun

Revisited Examples

```
System.out.println(new
GregorianCalendar(12, 12, 12));
```

```
java.util.GregorianCalendar[time=?,areFieldsSet=false,areA
llFieldsSet=false,lenient=true,zone=sun.util.calendar.Zone
Info[id="America/New_York",offset=-
18000000,dstSavings=3600000,useDaylight=true,transitions=2
35,lastRule=java.util.Simpletime
zone[id=America/New_York,offset=-
18000000,dstSavings=3600000,useDaylight=true,startYear=0,s
tartMode=3,startMonth=2,startDay=8,startDayOfWeek=1,startT
ime=7200000,startTimeMode=0,endMode=3,endMonth=10,endDay=1
,endDayOfWeek=1,endTime=7200000,endTimeMode=0]],firstDayOf
Week=1,minimalDaysInFirstWeek=1,ERA=?,YEAR=12,MONTH=12,WEE
K_OF_YEAR=?,WEEK_OF_MONTH=?,DAY_OF_MONTH=12,DAY_OF_YEAR=?,
DAY_OF_WEEK=?,DAY_OF_WEEK_IN_MONTH=?,AM_PM=0,HOUR=0,HOUR_O
F_DAY=0,MINUTE=0,SECOND=0,MILLISECOND=?,ZONE_OFFSET=?,DST_
OFFSET=?]
```

Problems Getting a Date

```
System.out.println(dtFmt.format(new
GregorianCalendar(12,12,12).getTime()));
// January 12, 0013 12:00:00 AM EST
```

Several problems here:

1. Which 12 is for which date field?
2. Month 12 is December, right? No. January
3. They got the year right! Almost. 13 CE
4. Wait - there is a time in a calendar?
5. More than that, there is a time zone

Calendar

- “Calendar” represents a date, time and time-zone
- Defaults to Gregorian calendar
- In Thailand only, you get a Buddhist calendar
- You can ask specifically ask for a Japanese calendar Calendar

Not Much Improvement

- Conceptually an instant, not a calendar
- But, can’t create a Calendar from a Date
- Can’t format a Calendar
- Zero-based offsets
- Stores internal state in two different ways
  - milliseconds from epoch
  - set of fields
- Has bugs and performance issues
- Mutable, not thread-safe

JAVA 8 Date and Time API

- 2002 - Stephen Colebourne starts open source Joda-Time project
- 2005 - Release of Joda-Time 1.0
- 2007 - JSR 310, for inclusion in Java
- 2011 - Release of Joda-Time 2.0
- 2014 - Finally, the date and time API is in Java 8

No Problem Getting a Date

```
System.out.println(
LocalDate.of(12, 12, 12));
// 0012-12-12
```

No problems:

1. ISO 8601 order of fields - year, month, day.
2. Month 12 is December.
3. Year is 12 CE.
4. No time component.
5. No time zone

Bad Arguments

```
System.out.println(
LocalDate.of(13, 13, 13));
```

```
java.time.DateTimeException: Invalid value for MonthOfYear
(valid values 1 - 12): 13
       at java.time.temporal.ValueRange.checkValidValue(Unknown
Source)
       at
java.time.temporal.ChronoField.checkValidValue(Unknown Source)
       at java.time.LocalDate.of(Unknown Source)
…
```

Concepts

Most importantly, the Java 8 date and time API forces you to think carefully about what you are doing.

```
// Don't code like this again
Calendar calendar = new GregorianCalendar();
Date date = calendar.getTime();
```

Epoch

- Reference point to measure time
- May be based on religious or political milestones
- Divides the timeline into eras
- Start of a particular era

Computer System Epochs

- January 0, 0 - MATLAB
- January 1, 1 - Symbian, .NET
- January 1, 1601 - COBOL, Windows
- January 1, 1900 - LISP, SNTP
- January 1, 1904 – Old Mac OS
- **January 1, 1970** - Unix Epoch (Linux, Mac OS X), Java, C, JavaScript, Perl, PHP, Python, Ruby

Calendar System

- Organizes days for social, religious, commercial or administrative purposes
- Names periods like days, weeks, months, and years
- Periods may follow cycles of the sun or moon
- A date is a specific day in the system
- May be based on an epoch

UTC

- **GMT** is Greenwich Mean Time
- Mean solar time at the Royal Observatory in Greenwich

- **UTC** is Coordinated Universal Time
- Precisely defined with atomic time Does not change with seasons
- Replaced GMT as reference time scale on 1 January 1972

ISO 8601

- International standard for representation of dates and times
- Uses the Gregorian calendar system
- Ordered from most to least significant: year, month, day, hour, minute
- Each date and time value has a fixed number of digits with leading zeros
- Uses four-digit year at minimum, YYYY

Machine and Human Timelines

- Machines have one view of time
  - discrete points corresponding to the smallest measurement possible
  - a single, ever increasing number
- Humans have a different view of time
  - continuous timelines
  - calendar systems
  - arbitrary units like years, months, days, hours
  - time zones, and daylight savings rules

Design Principles

- Distinguish between machine and human views
- Well-defined and clear purpose
- Immutable, thread-safe
- Reject null and bad arguments early
- Extensible, by use of strategy pattern
- Fluent interface with chained methods

Instant

- Point on a discretized time-line
- Stored to nanosecond resolution
  - long for seconds since epoch, and
  - int for nanosecond of second
- Convert to any date time field using a Chronology
- Use for event time-stamps

Partial

- An indication of date or time that cannot identify a specific, unique instant
- Definition uses fields such as year, month, day of month, and time of day
- Commonly used partials, such as LocalDate and LocalTime are available
- Others like MonthDay, YearMonth (card expiration?) are also available

Duration

- Precise length of elapsed time, in nanoseconds
- Does not use date-based constructs like years, months, and days
- Can be negative, if end is before start

Period

- A length of elapsed time
- Defined using calendar fields - years, months, and days (not minutes and seconds)
- Takes time zones into account for calculation

Time Zone

- Region with uniform standard time for legal, commercial, social, and political purposes
- Some countries use daylight saving time for part of the year
- Offset from UTC (UTC-12 to UTC+14)
- UTC is sometimes denoted by Z (Zulu)
- JDK time zone data is updated with JDK releases

Clock

- Gets the current instant using a time-zone
- Use instead of System.currentTimeMillis()
- Use an alternate clock for testing Clock

```
public class SomeBean {
  @Inject private Clock clock;
  public void process() {
    LocalDate date = LocalDate.now(clock);
    ...
  }
}
```

Chronology

- Pluggable calendar system
- Provides access to date and time fields
- Built-in
  - ISO8601 (default): IsoChronology
  - Chinese: MinguoChronology
  - Japanese: JapaneseChronology
  - Thai Buddhist: ThaiBuddhistChronology
  - Islamic: HijrahChronology

New Packages

- java.time - instants, durations, dates, times, time zones, periods
- java.time.format - formatting and parsing
- java.time.temporal - field, unit, or adjustment access to temporals
- java.time.zone – support for time zones
- java.time.chrono - calendar systems other than ISO-8601

Commonly Used Classes

- LocalDate
  - ISO 8601 date without time zone and time
  - Corresponds to SQL DATE type
  - Example: birthdate or employee hire-date
- LocalTime
  - ISO 8601 time without time zone and date
  - Corresponds to SQL TIME type
  - Example: the time that an alarm clock goes off
- LocalDateTime
  - ISO 8601 date and time without time zone
  - Corresponds to SQL TIMESTAMP type

Commonly Used Classes

Class or Enum	Year	Month	Day	Hours	Minutes	Nanos	Zone Offset	Zone ID	toString Output
Instant						✓			2013-08-20T15:16:26.355Z
LocalDate	✓	✓	✓						2013-08-20
LocalDateTime	✓	✓	✓	✓	✓	✓			2013-08-20T08:16:26.937
ZonedDateTime	✓	✓	✓	✓	✓	✓	✓	✓	2013-08-21T00:16:26.941+09:00[Asia/Tokyo]
LocalTime				✓	✓	✓			08:16:26.943
MonthDay		✓	✓						--08-20
Year	✓								2013
YearMonth	✓	✓							2013-08
Month		✓							AUGUST
OffsetDateTime	✓	✓	✓	✓	✓	✓	✓		2013-08-20T08:16:26.954-07:00
OffsetTime				✓	✓	✓	✓		08:16:26.957-07:00
Duration						✓			PT20H
Period	✓	✓	✓						P10D

http://docs.oracle.com/javase/tutorial/datetime/iso/overview.html

Consistent Operations

- of - static factory, validates input
- from - static factory, converts to instance of target class
- get - returns part of the state is - queries the state
- with - immutable copy with elements changed
- to - converts to another object type
- plus, minus - immutable copy after operation

Staying Constant

- Day of week, for example DayOfWeek.SUNDAY
- Month , for example LocalDate.of(2014, Month.MAY, 20);
- Time units, for example Instant.now().plus(1, ChronoUnit.DAYS)
- Other useful constants
  - LocalTime.MIDNIGHT // 00:00
  - LocalTime.NOON // 12:00

Formatting

- Format with a DateTimeFormatter instance
- Internationalization is supported
- Custom formats can be used, including am/ pm for time

Parsing

- Parse with a DateTimeFormatter instance
- parse(…) methods return a temporal
- Use from(…) to convert to a known date or time type

Temporal Adjusters

```
TemporalAdjuster fourMinutesLater =
new TemporalAdjuster() {
  @Override
   public Temporal adjustInto(Temporal
   temporal) {
     return temporal.plus(4,
                          ChronoUnit.MINUTES);
   }
};
LocalTime time = LocalTime.of(12, 0, 0);
LocalTime later = time.with(fourMinutesLater);
```

Temporal Adjusters
Java 8 style

```
LocalTime time = LocalTime.of(12, 0, 0);
time.with(temporal ->
temporal.plus(4, ChronoUnit.MINUTES)));
```

Temporal Queries

- Strategy for extracting information from temporals
- Externalize the process of querying
- Examples
  - get the time zone in a temporal
  - check if date is February 29 in a leap year
  - calculate days until your next birthday
- **TemporalQueries** class has implementations of common queries

Summary

- Existing date-related APIs can be error- prone and tedious
- Separate concepts of computer-related times and human-related times

Need to manipulate dates and times? Use Joda-Time or the Java 8 date and time API