
ORACLE®

JavaOne™

ORACLE®

The Java Date and Time API in Java SE 8

Introduction and Tips
CON3151

Roger Riggs
Consulting Member of Technical Staff
Java Products Group, Oracle
October 28, 2015

JavaOne
ORACLE

Program Agenda

1. Java Time Core API
2. Calendar Neutral API and Regional Calendars
3. Formatting and Localization
4. Tips and Techniques

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Background and History

- JSR 310 started in 2007 by Stephen Colebourne to build on Joda-Time value and experience
- Integrated into SE 8 Developer Releases in 2013
- SE 8 Final Release in March 2014
- Very successful project to provide a state of the art Calendar API to Java

Introduction to Java Time

- ISO 8601 Core Calendar – java.time
  - LocalTime, LocalDate, LocalDateTime, ZonedDateTime, …
  - Clock, Instant, Duration, Period, ZoneId, Month, DayOfWeek, …
- Parsing and Formatting – java.time.format
  - DateTimeFormat, DateTimeFormatBuilder, standard formats, patterns, styles, …
- TimeZone – java.time.zone
  - ZoneRules, transitions, etc.

Introduction to Java Time

- Regional Calendars – java.time.chrono
  - Chronology, Era, ChronoLocalDate, ChronoLocalDateTime, ChronoZonedDateTime, …
  - Japanese, ThaiBuddhist, Minguo, Hijrah calendars
- Framework – java.time.temporal
  - Units, Fields, Adjusters, Temporal, TemporalAccessor, TemporalAmount, TemporalQuery, …

ISO Calendar Types

```
LocalDate       2015-10-03
LocalTime                 11:05:30.987654321
LocalDateTime   2015-10-03T11:05:30
OffsetTime                11:05:30+01:00
OffsetDateTime  2015-10-03T11:05:30+01:00
ZonedDateTime   2015-10-03T11:05:30+01:00 Europe/Paris

Year            2010
YearMonth       2015-10
MonthDay            -10-03

Instant         2576458258.266 seconds since 1970-01-01
```

Java Time API Design vs. java.util Date and Calendar

- Fluent API
- Immutable instances
- Thread safe
- Strong types
- Fit for purpose types

- Not Fluent
- Mutable instances – clone needed
- Not Thread safe
- Weakly typed calendars
- One size fits all API

Date Time API Comparison

Fit for purpose Date Time types

Java.time ISO Calendar	Java.util Calendar
Instant	Date
LocalDate, LocalTime, LocalDateTime	Calendar
ZonedDateTime	Calendar
OffsetDateTime, OffsetTime,	Calendar
ZoneId, ZoneOffset, ZoneRules	TimeZone
Week Starts on Monday (1 .. 7) enum MONDAY, TUESDAY, … SUNDAY	Week Starts on Sunday (1 .. 7) int values SUNDAY, MONDAY, … SATURDAY
12 Months (1 .. 12) enum JANUARY, FEBRUARY, …, DECEMBER	12 Months (0 .. 11) int values JANUARY, FEBRUARY, … DECEMBER

Design Patterns of Method Names

Consistent naming improves API usability and approachability

Prefix	Method Type	Use
of	static factory	Creates an instance where the factory is primarily validating the input parameters, not converting them
from	static factory	Converts the input parameter to an instance of the target class, which may lose information from the input
date, **dateNow**	static factory	Creates a date from the arguments or from a clock or current time.
parse	static factory	Parses the input string to produce an instance of the target class
format	instance	Uses a formatter to format the values in the temporal object to a string
get	instance	Returns a part of the state of the target object
with	instance	Returns a copy of the target object with one element changed; the immutable equivalent to set
plus, **minus**	instance	Returns a copy of the target object with an amount of time added or subtracted
to	instance	Converts this object to another type
at	instance	Combines this object with another
isBefore, **isAfter**, **isEqual**	Instance	Compares this object with another on the timeline

Getting Date Time Fields

```
ZonedDateTime zdt = ...;

int nanos = zdt.getNano();
int millis = zdt.get(MILLIS_OF_SECOND);
int second = zdt.getSecond();
int minute = zdt.getMinute();
int hour = zdt.getHour();
int day = zdt.getDayOfMonth();
Month month = zdt.getMonth();
int monthval = zdt.getMonthValue();
int year = zdt.getYear();
```

```
GregorianCalendar cal = ...;


int millis = cal.get(Calendar.MILLISECOND);
int second = cal.get(Calendar.SECOND);
int minute = cal.get(Calendar.MINUTE);
int hour = cal.get(Calendar.HOUR);
int day = cal.get(Calendar.DAY_OF_MONTH);

int monthval = cal.get(Calendar.MONTH) + 1;
int year = cal.get(Calendar.YEAR);
```

Setting Date Time Fields

```
ZonedDateTime zdt = ...;

zdt = zdt.withNano(1);
Zdt = zdt.with(MILLI_OF_SECOND, 1);
zdt = zdt.withSecond(1);
zdt = zdt.withMinute(1);
zdt = zdt.withHour(1);
zdt = zdt.withDayOfMonth(1);
zdt = zdt.withMonth(1);
zdt = zdt.withYear(1);
```

```
GregorianCalendar cal = ...;


cal.set(Calendar.MILLISECOND, 1);
cal.set(Calendar.SECOND, 1);
cal.set(Calendar.MINUTE, 1);
cal.set(Calendar.HOUR, 1);
cal.set(Calendar.DAY_OF_MONTH, 1);
cal.set(Calendar.MONTH, 1 - 1);
cal.set(Calendar.YEAR, 1);
```

Date Time Field Arithmetic

```
ZonedDateTime zdt = ...;

zdt1 = zdt.plusNanos(1);
          .plus(1, MILLIS)
          .plusSeconds(1)
          .plusMinutes(1)
          .plusHours(1)
          .plusDays(1)
          .plusMonths(1)
          .plusYears(1);

zdt2 = zdt.minusNanos(1);
          .minus(1, MILLIS)
          .minusSeconds(1);
          .minusMinutes(1);
          .minusHours(1);
          .minusDays(1);
          .minusMonths(1);
          .minusYears(1);
```

```
GregorianCalendar cal = ...;

// Modify the fields
cal.add(Calendar.MILLISECOND, 1);
cal.add(Calendar.SECOND, 1);
cal.add(Calendar.MINUTE, 1);
cal.add(Calendar.HOUR, 1);
cal.add(Calendar.DAY_OF_MONTH, 1);
cal.add(Calendar.MONTH, 1);
cal.add(Calendar.YEAR, 1);


cal.add(Calendar.MILLISECOND, -1);
cal.add(Calendar.SECOND, -1);
cal.add(Calendar.MINUTE, -1);
cal.add(Calendar.HOUR, -1);
cal.add(Calendar.DAY_OF_MONTH, -1);
cal.add(Calendar.MONTH, -1);
cal.add(Calendar.YEAR, -1);
```

Calendar Neutral API and Regional Calendars

Calendar Neutral API

- The core java.time APIs are designed to avoid common errors
- For other calendars common assumptions may be invalid
  - Don't assume the number of months of year
  - Use the API to do all calendar arithmetic. Plus/minus days, months, years
  - Do not assume roll-over at particular numbers of days-per-month or months-per-year
  - Don't assume the week starts on Monday (ISO), use WeekFields.
  - Don't assume the month numbers are bound to specific months, use DateTimeFormatter to get names
- A Chronology is added to provide the correct semantics for the calendar

Calendar Neutral Dates

```
import static java.time.temporal.ChronoField.*;
import static java.time.temporal.ChronoUnit.*;

Locale locale = …;

Chronology chrono = Chronology.ofLocale(locale);
ChronoLocalDate date = chrono.dateNow();

int day   = date.get(DAY_OF_MONTH);
int month = date.get(MONTH_OF_YEAR);
int year  = date.get(YEAR_OF_ERA);
Era era   = date.get(ERA)

ChronoLocalDate nextmonth = date
    .with(DAY_OF_MONTH, 1)
    .plus(1, MONTHS);
```

```
import static java.util.Calendar.*;


Locale locale = …;

Calendar cal = Calendar.getInstance(locale);



int day   = cal.get(DAY_OF_MONTH);
int month = cal.get(MONTH) + 1;
int year  = cal.get(YEAR);
int era   = cal.get(ERA);

Calendar nextMonth = cal.clone();
nextMonth.set(DAY_OF_MONTH, 1);
nextMonth.add(MONTH, 1);
```

Calendar Neutral Dates, Times, and Time Zones

```
Chronology chrono = …;
ChronoLocalDate date = chrono.dateNow();

// Calendar neutral dates and times combinations.
ChronoLocalDateTime<?> cldt = date.atTime(LocalTime.NOON);

ChronoLocalDate d = cldt.toLocalDate();
LocalTime t = cldt.toLocalTime();

ZoneId zone = ZoneId.of(“Europe/Paris");
ChronoZonedDateTime<?> czdt = cldt.atZone(zone);
czdt = czdt.with(LocalTime.MIDNIGHT);


Output:
   2015-10-01T00:00+02:00[Europe/Paris]
```

Calendar Neutral API Mapping

Accessed by Fields and Units

Java.util.Calendar	Java.time.temporal.ChronoField	Java.time.temporal.ChronoUnits
Calendar.DAY_OF_MONTH	ChronoField.DAY_OF_MONTH	ChronoUnit.DAYS
Calendar.MONTH	ChronoField.MONTH_OF_YEAR	ChronoUnit.MONTHS
Calendar.YEAR	ChronoField.YEAR	ChronoUnit.YEARS
Calendar.HOUR	ChronoField.HOUR_OF_DAY	ChronoUnit.HOURS
Calendar.MINUTE	ChronoField.MINUTE_OF_HOUR	ChronoUnit.MINUTES
Calendar.SECOND	ChronoField.SECOND_OF_DAY	ChronoUnit.SECONDS
Calendar.MILLISECOND	ChronoField.MILLI_OF_SECOND	ChronoUnit.MILLIS

Regional Calendars

Listing, lookup, creating dates

```
Set<Chronology> chronologies =
  Chronology.getAvailableChronologies();
for (Chronology chrono : chronologies) {
     out.printf(" %s (%s)%n",
          chrono.getId(), chrono.getCalendarType());
}

// Lookup by name or CLDR type
Chronology chrono = Chronology.of("Hijrah");
chrono = Chronology.of("islamic-umalqura");

// Select calendar based on BCP47 locale language tags
Locale locale =
         Locale.forLanguageTag("ja-JP-u-ca-japanese");
chrono = Chronology.ofLocale(locale);

// Create date based on proleptic year, month, day
ChronoLocalDate date = chrono.date(2015, 10, 28);
```

```
ISO (iso8601)
Hijrah-umalqura (islamic-umalqura)
ThaiBuddhist (buddhist)
Japanese (japanese)
Minguo (roc)


Hijrah-umalqura (islamic-umalqura)


Japanese (japanese)


Japanese Heisei 27-10-28
```

Japanese Date, Eras and Chronology

```
static import java.time.chrono.JapaneseEra.HEISEI;
Locale locale =
      Locale.forLanguageTag("ja-JP-u-ca-japanese");
ZoneId zone = ZoneId.of("Asia/Tokyo");

JapaneseDate jdate =
   JapaneseDate.of(HEISEI, 27, 10, 28);

ChronoLocalDateTime<JapaneseDate> jldt =
    jdate.atTime(LocalTime.now());
ChronoZonedDateTime<JapaneseDate> jzdt =
    jldt.atZone(zone);

jzdt = jzdt.plus(2, ChronoUnit.DAYS);

jzdt.format(DateTimeFormatter
       .ofPattern("GGGGyy-mm-dd")
       .withLocale(locale))

Output: 平成27-10-30
```

```
Locale locale =
      Locale.forLanguageTag("ja-JP-u-ca-japanese");
ZoneId zone = ZoneId.of("Asia/Tokyo");

Calendar jcal = Calendar.getInstance(locale);
jcal.setTimeZone(TimeZone.getTimeZone(zone));
jcal.set(Calendar.YEAR, 28);
jcal.set(Calendar.MONTH, 10 - 1);
jcal.set(Calendar.DAY_OF_MONTH, 28);




jcal.add(Calendar.DAY_OF_MONTH, 2);

SimpleDateFormat sdf = (SimpleDateFormat)DateFormat
    .getDateInstance(SimpleDateFormat.SHORT, locale);
sdf.applyLocalizedPattern("GGGGyy-mm-dd");
sdf.format(jcal.getTime());

Output: 平成27-10-30
```

Hijrah Islamic Calendar

```
Locale locale = Locale.forLanguageTag(“ar-SA-u-ca-islamic-umalqura”);

ZoneId zone = ZoneId.of("Asia/Riyadh");
HijrahDate hdate = HijrahDate.now();
date = date.plus(1, ChronoUnit.DAYS);

ChronoLocalDateTime<HijrahDate> cldt = hdate.atTime(LocalTime.NOON);
ChronoZonedDateTime<HijrahDate> hzdt = cldt.atZone(zone);

hzdt = hzdt.plus(2, ChronoUnit.DAYS);
HijrahDate hdate2 = hzdt.toLocalDate();
LocalTime htime2 = hzdt.toLocalTime();

hzdt.format(DateTimeFormatter
            .ofLocalizedDate(FormatStyle.FULL)
            .withLocale(locale))

Output:
    2015 أكتوبر، 24 السبت،
```

Formatting and Localization

DateTimeFormatter Builder

Customizing parsing and formatting

- Factory used to build a template
  for a sequence of fields to be parsed or formatted
- Literals, for example “/”, “-”, “:”, or any string
- Numeric values with control of width, sign, leading zeros
- Text indexed by locale and style – short, narrow, full, standalone
- Patterns indexed by locale - date, time, date-time
- Fraction control, with control of width and decimal points
- Field padding to width and character
- ZoneId, ZoneOffset, Chronology, Instant specialized fields

DateTimeFormatter Builder II

- Case sensitive vs. in-sensitive parsing
- Optional fields
- Default values if not present in input
- Strict vs. Lenient parsing mode
- Concatenation of Formatters

Formatter Builder Example

Parse time with optional minutes and seconds

```
    DateTimeFormatter format = new DateTimeFormatterBuilder()
        .appendValue(HOUR_OF_DAY, 1, 2, SignStyle.NEVER)
        .optionalStart()
            .appendLiteral(":").appendValue(MINUTE_OF_HOUR, 2)
            .optionalStart()
                .appendLiteral(":").appendValue(SECOND_OF_MINUTE, 2)
            .optionalEnd()
        .optionalEnd()
        .parseDefaulting(MINUTE_OF_HOUR, 1)
        .parseDefaulting(SECOND_OF_MINUTE, 0)
        .toFormatter();
    LocalTime date = LocalTime.parse(s, format);
    out.printf(" Parsed %10s --- %7s%n", s, date.format(fmt));
Output:

 Parsed          9 → 9:01:00
 Parsed      09:05 → 9:05:00
 Parsed   09:30:59 → 9:30:59
 Parsed      23:00 → 23:00:00
```

Formatting Patterns

Format vs. DateTimeFormat vs. SimpleDateFormat

Formatter Letter %tx and %Tx	SDF Pattern Letter	CLDR Pattern Letters DateTimeFormatter	Date or Time Component	Examples
	G	G	Era designator	AD
Y, y, C	y	y	Year –of-Era	1996; 96
B, b, h, m	M	M	Month in year (context sensitive)	July; Jul; 07
B, b, h, m	L	L	Month in year (standalone form)	July; Jul; 07
	w	w	Week in year	27
	W	W	Week in month	2
j	D	D	Day in year	189
D, e	d	d	Day in month	10
A, a	E	E	Day name in week	Tuesday; Tue
	u	e	Day number in week (1 = Monday .. 7 = Sunday)	1
p	a	a	Am/pm marker	PM
H, k	H	H	Hour in day (0-23)	0
	k	k	Hour in day (1-24)	24
M	m	m	Minute in hour	30
S	s	s	Second in minute	55
L	S	S	Millisecond / Fraction of Second	978
Z, z	z , Z, X		Time zone	Pacific Standard Time; PST; GMT-08:00 ; -0800 ; -08; -0800; -08:00

java.util.Formatter supports Java Time types - String.format, PrintStream.format, …

```
Locale locale = Locale.getDefault();
Formatter fmt = new Formatter(out, locale);

Date date = new Date();
Calendar cal = Calendar.getInstance();

LocalDate ld = LocalDate.now();
LocalTime lt = LocalTime.now();
LocalDateTime ldt = LocalDateTime.now();
ZonedDateTime zdt = ZonedDateTime.now();

fmt.format("date:          %tc%n", date);
fmt.format(" cal:          %tc%n", cal);
fmt.format("ZonedDateTime: %tc%n", zdt);

fmt.format("    LocalDate: %1$ta %1$tb %1$td %1$tY%n", ld);
fmt.format("    LocalTime: %tT %n", lt);
fmt.format("LocalDateTime: %1$ta %1$tb %1$td %1$tT %1$tY%n", ldt);
```

```
date:          Wed Oct 21 10:30:49 EDT 2015
 cal:          Wed Oct 21 10:30:49 EDT 2015
ZonedDateTime: Wed Oct 21 10:30:49 EDT 2015

    LocalDate: Wed Oct 21 2015
    LocalTime: 15:16:33
LocalDateTime: Wed Oct 21 10:30:49 2015
```

Predefined Date Time Formats

```
ZonedDateTime now = ZonedDateTime.now();

String toString = now.toString();
// 2013-08-06T18:12:17.423-04:00[America/New_York]

DateTimeFormatter format =
  DateTimeFormatter.ISO_ZONED_DATE_TIME;

String s = now.format(format);
// 2015-10-21T10:34:36.954798-04:00[America/New_York]
```

Formatter	Example
ofLocalizedDateTime(dateStyle, timeStyle)	3 Jun 2015 11:05
BASIC_ISO_DATE	20151203
ISO_LOCAL_DATE	2015-12-03
ISO_OFFSET_DATE	2015-12-03+01:00
ISO_DATE	2015-12-03+01:00; 2015-12-03
ISO_LOCAL_TIME	10:15:30
ISO_OFFSET_TIME	10:15:30+01:00
ISO_TIME	10:15:30+01:00; 10:15:30
ISO_LOCAL_DATE_TIME	2015-12-03T10:15:30
ISO_OFFSET_DATE_TIME	2015-12-03T10:15:30+01:00
ISO_ZONED_DATE_TIME	2015-12-03T10:15:30+01:00[Europe/Paris]
ISO_INSTANT	2015-12-03T10:15:30Z
RFC_1123_DATE_TIME	Tue, 3 Jun 2015 11:05:30 GMT

DateTimeFormatter - Predefined and Customizable Patterns

```
import static
  java.time.format.DateTimeFormatter.*;

ZonedDateTime now = ZonedDateTime.now();

now.toString();                                  015-10-21T10:34:36.954798-04:00[America/New_York]

now.format(RFC_1123_DATE_TIME)                   Wed, 21 Oct 2015 10:34:36 -0400
now.format(ISO_DATE_TIME)                        2015-10-21T10:34:36.954798-04:00[America/New_York]
now.format(ofLocalizedDateTime(FormatStyle.FULL)) Wednesday, October 21, 2015 at 10:34:36 AM EDT

now.format(ofPattern("z zz zzz zzzz VV"))        EDT EDT EDT Eastern Daylight Time America/New_York

now.format(ofPattern("y'-W'w-e"))                2015-W43-4
now.format(ofPattern("h:ss a"))                  4:02 PM
now.format(ofPattern("'Q'q y"))                  Q4 2015
```

Locale based Date and Time formats – Use DateTimeFormatter instead of SimpleDateFormat

Immutable and thread safe formatting

```
Locale locale = …;
GregorianCalendar cal = new GregorianCalendar();

DateTimeFormatter dtf = DateTimeFormatter
   .ofLocalizedDateTime(FormatStyle.MEDIUM,
        FormatStyle.MEDIUM)
   .withLocale(locale);

ZonedDateTime zdt = cal.toZonedDateTime();
dtf.format(zdt);

Output:
   Oct 21, 2015, 10:38:32 AM
```

```
Locale locale = …;
Calendar cal = Calendar.getInstance();

SimpleDateFormat sdf
  = (SimpleDateFormat)DateFormat
  .getDateTimeInstance(SimpleDateFormat.MEDIUM,
        SimpleDateFormat.MEDIUM, locale);


Date date = cal.getTime();
sdf.format(date);

Output:
   Oct 21, 2015, 10:38:32 AM
```

DatePicker Control in JavaFX

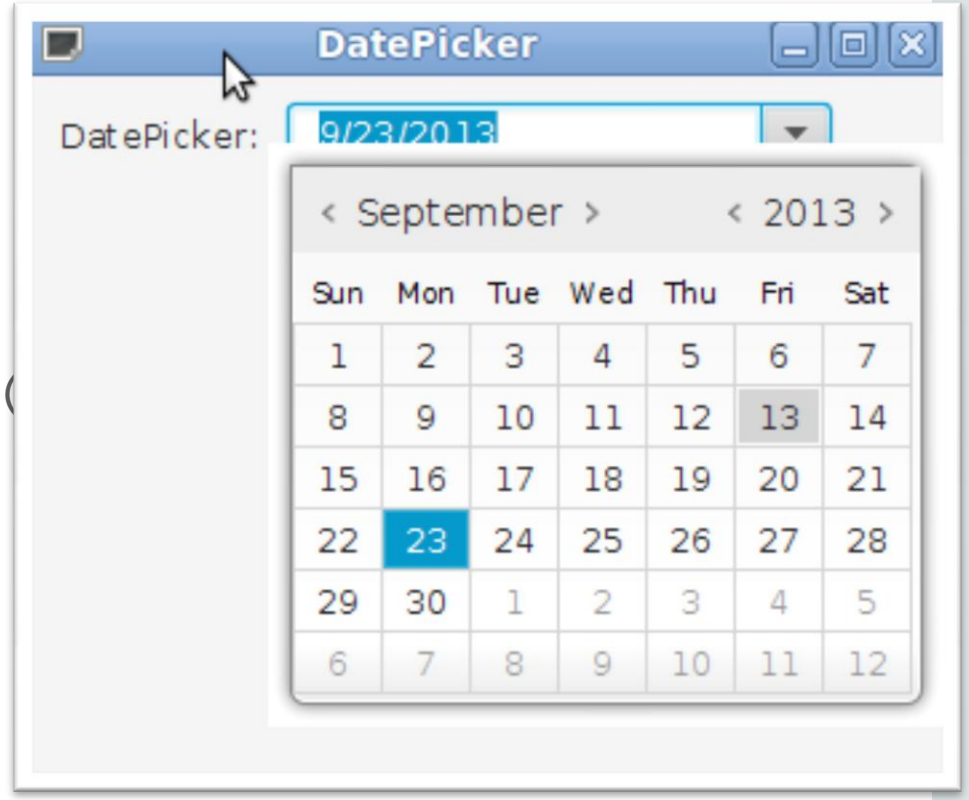
- Settable Chronology
- Supports Regional Formatting via Locale
- Customizable Cell Factory

```
import javafx.scene.control.DatePicker;

    datePicker = new DatePicker();

    datePicker.setOnAction(new EventHandler<ActionEvent>(
        @Override public void handle(ActionEvent t) {
            LocalDate isoDate = datePicker.getValue();
            println("Selected date: “ + isoDate);
        }
    });

Output:
    Selected date: 2013-09-23
```



Tips and Techniques

Tip: Converting Date or Instant to ZonedDateTime

Not equivalent – ZonedDateTime has a time zone and Instant and Date **do not**

```
ZonedDateTime is LocalTime, LocalDate, ZoneId
Instant is epoch-seconds, nano-seconds

long millis = System.currentTimeMillis();
Instant instant = Instant.ofEpochMilli(millis);

zdt = instant.atZone(ZoneOffset.UTC);

zdt = ZonedDateTime.ofInstant(instant, ZoneId.systemDefault());

zdt = ZonedDateTime.from(instant);
!! DateTimeException: Unable to obtain ZonedDateTime from TemporalAccessor: 2015-10-
  21T14:42:09.748Z of type java.time.Instant
!! No ZoneId or ZoneOffset
```

Tip: Converting GregorianCalendar and ZonedDateTime

Equivalent – both have date, time, and time zone

- ZonedDateTime is LocalDate, LocalTime, ZoneId
- GregorianCalendar has date, time, and time zone

```
GregorianCalendar calendar = new GregorianCalendar();
ZonedDateTime zdt = calendar.toZonedDateTime();

calendar = GregorianCalendar.from(zdt);
```

Tip: Formatting Duration

Default is ISO Duration syntax (PnTnHnMnS)

```
    Duration dur = Duration.ofMinutes(100);
    out.printf("  ISO Duration - Duration.toString(): %s%n", dur);

ISO Duration - Duration.toString(): PT1H40M

    // Convert to LocalTime
    LocalTime t = LocalTime.MIDNIGHT.plus(dur);
    out.printf(" Duration as LocalTime: %s%n", t);

As LocalTime: 01:40

    String pattern = "hh:mm:ss.SSS";
    DateTimeFormatter format = DateTimeFormatter.ofPattern(pattern);
    out.printf("  Formatted as %s: %s%n", pattern, t.format(format));

Formatted as hh:mm:ss.SSS: 01:40:00.000
```

Tip: Parsing with a choice of formats

If parsing should accept multiple unique formats – use optional components

```
String format = "[MM-dd-yyyy]"
              + " [yyyy/MM/dd]"
              + " [yyyyMMdd]";

String[] samples = {"09-25-2015",
                    "1999/01/01",
                    "19990109",
                   };

DateTimeFormatter format = DateTimeFormatter.ofPattern(format);
for (String s : samples) {
    LocalDate date = LocalDate.parse(s, format);
    out.printf("Parsed: %s as %s%n", s, date);
}
```

```
Parsed: 09-25-2015 as 2015-09-25
Parsed: 1999/01/01 as 1999-01-01
Parsed: 19990109   as 1999-01-09
```

Tip: Daylight Savings Lookup

Finding the next change

```
ZonedDateTime zdt = ZoneDateTime.now();
ZoneId zid = zdt.getZone();
ZoneRules rules = zid.getRules();
ZoneOffsetTransition transition = rules.nextTransition(zdt.toInstant());

Duration duration = transition.getDuration();
ZonedDateTime when = transition.getDateTimeBefore().atZone(zid);
String fmtString = duration.isNegative()
            ? " In the Fall on %s, fallback by %s%n"
            : " In the Spring on %s, spring forward by %s%n";
out.printf(fmtString,
            when.format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.FULL)),
            LocalTime.ofSecondOfDay(duration.abs().getSeconds()));
```

```
Output:
    In the Fall on Sunday, November 1, 2015 2:00:00 AM EST, fallback by 01:00
```

Tip: Comparing Date Time Values

When compareTo is not the same as timeline order

- The familiar `Comparable.compareTo` follows Object.equals semantics
  - Values in date time types are compared field by field; hour, minute, second, zone, etc.
  - For simple date time types, the results are the same
  - For ZonedDateTime and Calendar Neutral types, use time line order
- Time line order is equivalent to converting to Instant
  - Normalizes the effects of different ZoneId/ZoneOffsets
  - Normalizes the effects of different Chronologies
  - isBefore, isAfter, isEqual methods provide timeline order comparisons
  - ChronoLocalDateTime, ChronoZonedDateTime provide timeLineOrder `Comparator`

Tip: Importance of Time Line Order comparisons

```
    ZoneId zid = ZoneId.of("America/New_York");
    LocalDateTime now = LocalDateTime.now();
    ZonedDateTime zdt = now.atZone(zid);

    ZoneOffset offset = ZoneOffset.from(zdt);
    ZonedDateTime zdt1 = ZonedDateTime.of(zdt.toLocalDateTime(), offset);

    out.printf("zdt with ZoneId:     %s%nzdt with ZoneOffset: %s%n", zdt, zdt1);
    out.printf("  Object.equals:    %s%n", zdt.equals(zdt1));
    out.printf("  compareTo:        %s%n", zdt.compareTo(zdt1) == 0);
    out.printf("  Timeline isEqual: %s%n", zdt.isEqual(zdt1));
```

```
zdt with ZoneId:     2015-09-20T13:39:07.879-04:00[America/New_York]
zdt with ZoneOffset: 2015-09-20T13:39:07.879-04:00
  Object.equals:    false
  compareTo:        false
  Timeline isEqual: true
```

Tip: Temporal Adjusters

Apply a function to a Date Time Value

```
LocalDate date = …;

// Advance to Last Day of the Month
LocalDate endOfMonth = date
      .with(lastDayOfMonth());

// Advance to last Friday of the Month
LocalDate lastFriday = date
      .with(lastInMonth(DayOfWeek.FRIDAY));

// Go back to the previous Wednesday
LocalDate previousWed = date
      .with(previous(DayOfWeek.WEDNESDAY));

// Return the 4th Thursday of November(Thanksgiving)
LocalDate thanksgiving = date
      .with(Month.NOVEMBER)
      .with(dayOfWeekInMonth(4, DayOfWeek.THURSDAY));
```

TemporalAdjusters
firstDayOfMonth()
firstDayOfNextMonth()
firstDayOfNextYear()
firstDayOfYear()
firstInMonth(DayOfWeek dayOfWeek)
lastDayOfMonth()
lastDayOfYear()
lastInMonth(DayOfWeek dayOfWeek)
next(DayOfWeek dayOfWeek)
nextOrSame(DayOfWeek dayOfWeek)
previous(DayOfWeek dayOfWeek)
previousOrSame(DayOfWeek dayOfWeek)
dayOfWeekInMonth(int ordinal, DayOfWeek dayOfWeek)

Writing a Temporal Adjuster

Payday Function as TemporalAdjuster

```
// Paydays are the 15th and the last day of the month or
// the previous Friday if it lands on a weekend.

static Temporal nextPayday(Temporal currDate) {

    // Advance at least 1 day and skip Saturday and Sunday
    int pastFriday = currDate.get(ChronoField.DAY_OF_WEEK) - DayOfWeek.FRIDAY.getValue();
    int extraDays = (pastFriday < 0) ? 1 : 2 - pastFriday + 1;
    Temporal date = currDate.plus(extraDays, ChronoUnit.DAYS);

    // Payday is the 15th or the last day of the month
    int day = date.get(ChronoField.DAY_OF_MONTH);
    day = (day <= 15) ? 15 :  (int)date.range(ChronoField.DAY_OF_MONTH).getMaximum();
    Temporal payday = date.with(ChronoField.DAY_OF_MONTH, day);

    // If it falls on a Saturday or Sunday, backup to the previous Friday
    pastFriday = payday.get(ChronoField.DAY_OF_WEEK) - DayOfWeek.FRIDAY.getValue();
    if (pastFriday > 0) {
        payday = payday.minus(pastFriday, ChronoUnit.DAYS);
    }
    return payday;
}
```

Tip: Converting from Joda-Time

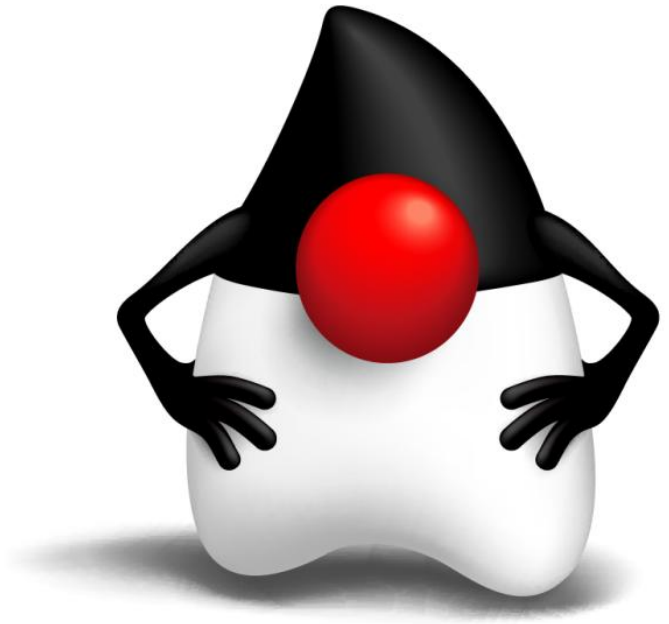
Designs are very similar but not every Joda-Time function is supported

- Mapping of class names
  - LocalDate, LocalTime, LocalDateTime are the same
  - Joda-Time DateTime is java.time.ZonedDateTime
  - Joda-Time DateTimeZone maps to java.time.ZoneId/ZoneRules/ZoneOffset
- Period and Duration
  - Joda-Time Period is years to seconds; Java.time.Period is simpler, only ISO date-based
  - Joda-Time Duration is same as java.time.Duration
- Java time is more open and flexible
  - Adjusters and queries are new
  - Ability to define own fields and units is essentially new
  - Adding calendar systems is simple rather than very hard

Summary

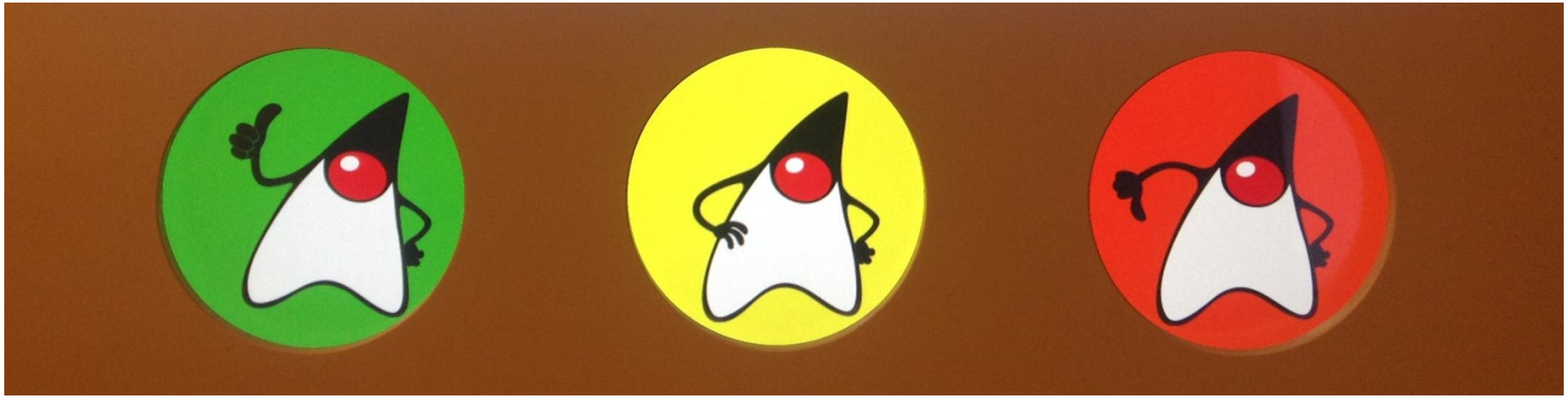
New Improved Date Time API

- Fluent, Immutable, Thread Safe, Easy to use
- Strong typing with fit for purpose types
- Easy to use formatting and parsing
- Extensible with Units, Fields, and Chronologies
- Interoperable with java.util.Calendar
- Supports Regional Calendars
- The essential ISO Calendar for global business



Q & A

- Threeten Articles and References
  - http://www.threeten.org/links.html
- Java™ Date Time Tutorials
  - http://docs.oracle.com/javase/tutorial/datetime
- Java™ Standard Edition API
  - http://docs.oracle.com/javase/8/docs/api



Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

JavaOne™

ORACLE®