Its high time!
JSR-310 – A new date and time API

Stephen Colebourne
Michael Santos

TS-6578

Java™ Champions

Java

Sun microsystems

Understand Java Specification Request (JSR) 310 - Date and Time API

GOAL

Agenda

- Problems today
- JSR-310 overview
- Continuous
- Human
- Integration
- Periods
- Miscellaneous

Spot the bugs...

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

TimeZone zone = TimeZone.getInstance("Asia/HongKong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

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

TimeZone zone = TimeZone.getInstance("Asia/HongKong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;

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

TimeZone zone = TimeZone.getInstance("Asia/HongKong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;

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

TimeZone zone = TimeZone.getInstance("Asia/HongKong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/HongKong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/HongKong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/Hong_Kong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/Hong_Kong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/Hong_Kong");
Calendar cal = new GregorianCalendar(zone);
cal.setTime(date);

DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/Hong_Kong");
Calendar cal = new GregorianCalendar(zone);
cal.setTime(date);

DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/Hong_Kong");
Calendar cal = new GregorianCalendar(zone);
cal.setTime(date);

DateFormat fm = new SimpleDateFormat("HH:mm Z");

Date calDate = cal.getTime();
String str = fm.format(calDate);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/Hong_Kong");
Calendar cal = new GregorianCalendar(zone);
cal.setTime(date);

DateFormat fm = new SimpleDateFormat("HH:mm Z");

Date calDate = cal.getTime();
String str = fm.format(calDate);
```

Spot the bugs...

```
int year = 2007 - 1900;
int month = 12 - 1;
Date date = new Date(year, month, 13, 16, 40);

TimeZone zone = TimeZone.getInstance("Asia/Hong_Kong");
Calendar cal = new GregorianCalendar(zone);
cal.setTime(date);

DateFormat fm = new SimpleDateFormat("HH:mm Z");
fm.setTimeZone(zone);
Date calDate = cal.getTime();
String str = fm.format(calDate);
```

Spot the bugs...

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

TimeZone zone = TimeZone.getInstance("Asia/HongKong");
Calendar cal = new GregorianCalendar(date, zone);


DateFormat fm = new SimpleDateFormat("HH:mm Z");


String str = fm.format(cal);
```

Existing API flaws

- Mutable
- January is 0, December is 11
- `Date` is not a date
- `Date` uses years from 1900
- `Calendar` cannot be formatted
- `DateFormat` not thread-safe
- SQL `Date/Time/Timestamp` extend Date

High time for a new API

- Time problems well known
  - Joda-Time - http://joda-time.sourceforge.net
- Desire for improvements in Java Platform, Standard Edition 7 (Java SE platform 7)*
- JSR-310 started February 2007
  - JSR-310 - http://jsr-310.dev.java.net

* No commitments have been made about inclusion

Agenda

- Problems today
- JSR-310 overview
- Continuous
- Human
- Integration
- Periods
- Miscellaneous

JSR-310 Overview

- Comprehensive model for date and time
- Type-safe
  - avoid primitives
  - self documenting
  - IDE friendly
- Interoperate with existing classes
- Consider XML and Database

Design principles

- Guide design
- Help decision making
- Derived from other libraries
- Derived from experience

Design principles – Immutable

- No change after construction
- Thread-safe
- Can be singletons

- Implementation considerations
  - classes and fields are final
  - construction typically by factory
  - 'with' methods instead of 'set'
    - `date.setDayOfMonth(12);` X
    - `date = date.withDayOfMonth(12);`

Design principles – Fluent

- Easy to read
- Easy to learn
- Like a sentence

- Implementation considerations
  - builder pattern
  - method names that 'flow'

Design principles – Clear, Explicit and Expected

- Each method is well-defined
- Javadoc easily explains what method does
- No coupling between methods

- Implementation considerations
  - few super/subclasses
  - no optional/pluggable state
  - long/complex javadoc → refactor

Design principles – Extensible

- Many weird ways to manipulate time
- JSR authors don't know everything
- Allow for extensions, but avoid confusion

- Implementation considerations
  - strategy pattern
  - default strategy for most use cases
  - clear javadoc for default

Analysing time

- 'Time' has many meanings
  - time-dimension
  - time-line
  - time-point
  - time-interval
  - time-duration
  - time-position
- How can we make sense of it?

Continuous and Human

- Two basic time-scales
  - way of 'counting' time
- Continuous
  - single incrementing number
  - designed for machines
- Human
  - field-based (year,month,day,hour,minute,second)
  - designed for humans

Agenda

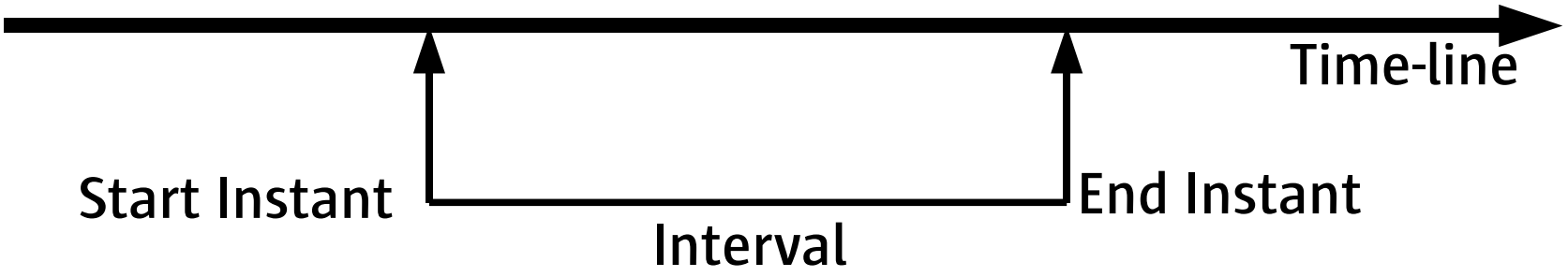
- Problems today
- JSR-310 overview
- Continuous
- Human
- Integration
- Periods
- Miscellaneous

Instant

- Single instantaneous point on the time-line
  - 53628746263276 nanoseconds after the epoch
- Used to store a timestamp
- Nanosecond precision for age of universe
  - problem – no suitable primitive – 96 bits
- Java class – **`Instant`**

Interval

- Interval of time on the time-line
  - from start instant to end instant
- Start inclusive, end exclusive
  - may support flexible inclusive/exclusive
- Java class – `InstantInterval`
  - or maybe `Interval<Instant>`

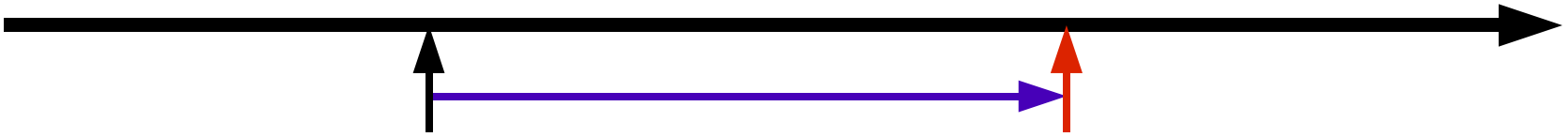


Duration

- Duration of time
  - 358753871581 nanoseconds
- Fundamental scientific quantity
  - not connected to the time-line
- Nanosecond precision
- Java class – `Duration`

Maths

- `Instant + Duration = Instant`



- `Duration + Duration = Duration`

- `Duration = 'Length' of an Interval`

Examples

```
// instant
Instant start = millisInstant(123450L);
Instant end = instant(223L, 450000000);
assert start.isBefore(end);
assert end.isAfter(start);

// interval
InstantInterval interval = interval(start,end);
assert interval.contains(start);

// duration
Duration duration = durationOf(interval);
Duration bigger = duration.multipliedBy(4);
Duration biggest = bigger.plus(duration);
Instant later = start.plus(duration);
```

Agenda

- Problems today
- JSR-310 overview
- Continuous
- Human
- Integration
- Periods
- Miscellaneous

Human-scale overview

- Human-scale dates and times
  - field based
  - year, month, day, hour, minute, second
- Requirements
  - Date and time
  - Date without time
  - Time without date
  - Time zone

ISO-8601

- Standard interchange format
- Basis for other standards
  - XML schema
- Includes
  - date
  - time
  - date-time
  - zone offset (from UTC)
  - period
- No support for time zone rules

ISO-8601

- **yyyy-MM-dd**
  - 2007-12-03
- **hh:mm:ss.SSSZ**
  - 11:05:30.123+01:00
- **yyyy-MM-dd'T'HH:mm:ss.SSSZ**
  - 2007-12-03T11:05:30.123+01:00

Analysis

- Start from ISO-8601
  - `yyyy-MM-dd'T'HH:mm:ss.SSSZ`
- Generalise
  - `{date}T{time}{offset}`
- Date – year, month, day
- Time – hour, minute, second, nanosecond
- Offset – from `+14:00` to `-12:00`

Classes

- `{date}T{time}{offset}`
- One class for each part
  - `LocalDate`
  - `LocalTime`
  - `ZoneOffset`
- One class for each combination
  - `LocalDateTime:  LocalDate + LocalTime`
  - `OffsetDate:  LocalDate + ZoneOffset`
  - `OffsetTime:  LocalTime + ZoneOffset`
  - `OffsetDateTime:  LocalDateTime + ZoneOffset`

Calendar fields

- `{year}-{month}-{day}`
- One class for each part
  - `Year`
  - `MonthOfYear` – Enum
  - `DayOfMonth`
- Extend to other related concepts
  - `DayOfWeek` — Enum
  - `DayOfYear`
  - and so on

Calendar fields

- `{hour}:{minute}:{second}`
- One class for each part
  - `HourOfDay`
  - `MinuteOfHour`
  - `SecondOfMinute`
- Extend to other related concepts
  - `NanoOfSecond`
  - and so on

Basic querying

- Query date/time using 'get' methods
- Method returns object/enum
  - no primitives

```
LocalDate date = date(2007, 12, 3);

Year year = date.getYear();

int yearValue = year.getValue();
boolean leap = year.isLeap();
Era era = year.getEra();
```

Matchers

- Need to query parts of a date/time
- Query can be simple or complex
  - is the year 2006 ?
  - is the date the last day of the year ?
- Strategy pattern – **`DateMatcher`**

```
public interface DateMatcher {
  boolean matchesDate(LocalDate input);
}

boolean matches = date.matches(year(2006));

boolean matches = date.matches(lastDayOfYear());
```

Adjusters

- Need to change a date/time
- Change can be simple or complex
  - change the year to 2006
  - change the date to the last day of the month
- Strategy pattern – **`DateAdjuster`**

```
public interface DateAdjuster {
  LocalDate adjustDate(LocalDate input);
}

LocalDate adjusted = date.with(year(2006));

LocalDate adjusted = date.with(lastDayOfMonth());
```

Resolvers

- Need to handle invalid date/time
  - February 30th
- Strategy pattern – **`DateResolver`**

```
public interface DateResolver {
  LocalDate resolveDate(
      Year year, MonthOfYear month, DayOfMonth day);
}

DateResolver res = DateResolvers.previousValid();
LocalDate date = date(2007, 2, 30, res);
// date = 2007-02-28
```

Time zones

- Rules for how zone offset changes
  - Daylight Savings
  - 'Permanent' offset changes
- Rules change frequently
  - Syria changed DST with 3 days notice
  - Western Australia has 3 year DST experiment
  - Brazil changes DST every year
- Rules based on Olson database
  - Represented by id – 'Europe/London'
- Complex

Time zones

- JDK™ software javadoc:
  - "TimeZone represents a time zone offset, and also figures out daylight savings"
- JSR-310 separates responsibility
  - `ZoneOffset` – from `+14:00` to `-12:00`
  - `TimeZone` – rules for switching zone offsets
- Only one additional date-time class
  - `ZonedDateTime`
  - (`ZonedDate` and `ZonedTime` are nonsense)

Zone resolvers

- Need to handle invalid time due to time zones
  - Spring Daylight Savings 'gap'
  - Autumn/Fall Daylight Savings 'overlap'
- Strategy pattern – **`ZoneResolver`**

```
// one day before DST ends (overlap of one hour)
TimeZone zone = timeZone("Europe/London");
ZonedDateTime dt = dateTime(2007,10,27,1,30,zone);
// dt = 2007-10-27 01:30 +01:00

ZoneResolver res = ZoneResolvers.retainOffset();
dt = dt.plus(days(1), res);
// dt = 2007-10-28 01:30 +01:00
```

Date/time class summary

- `LocalDate` `2007-12-03`
- `LocalTime` `11:05:30`
- `LocalDateTime` `2007-12-03T11:05:30`
- `OffsetDate` `2007-12-03+01:00`
- `OffsetTime` `11:05:30+01:00`
- `OffsetDateTime` `2007-12-03T11:05:30+01:00`
- `ZonedDateTime` `2007-12-03T11:05:30+01:00 Europe/Paris`

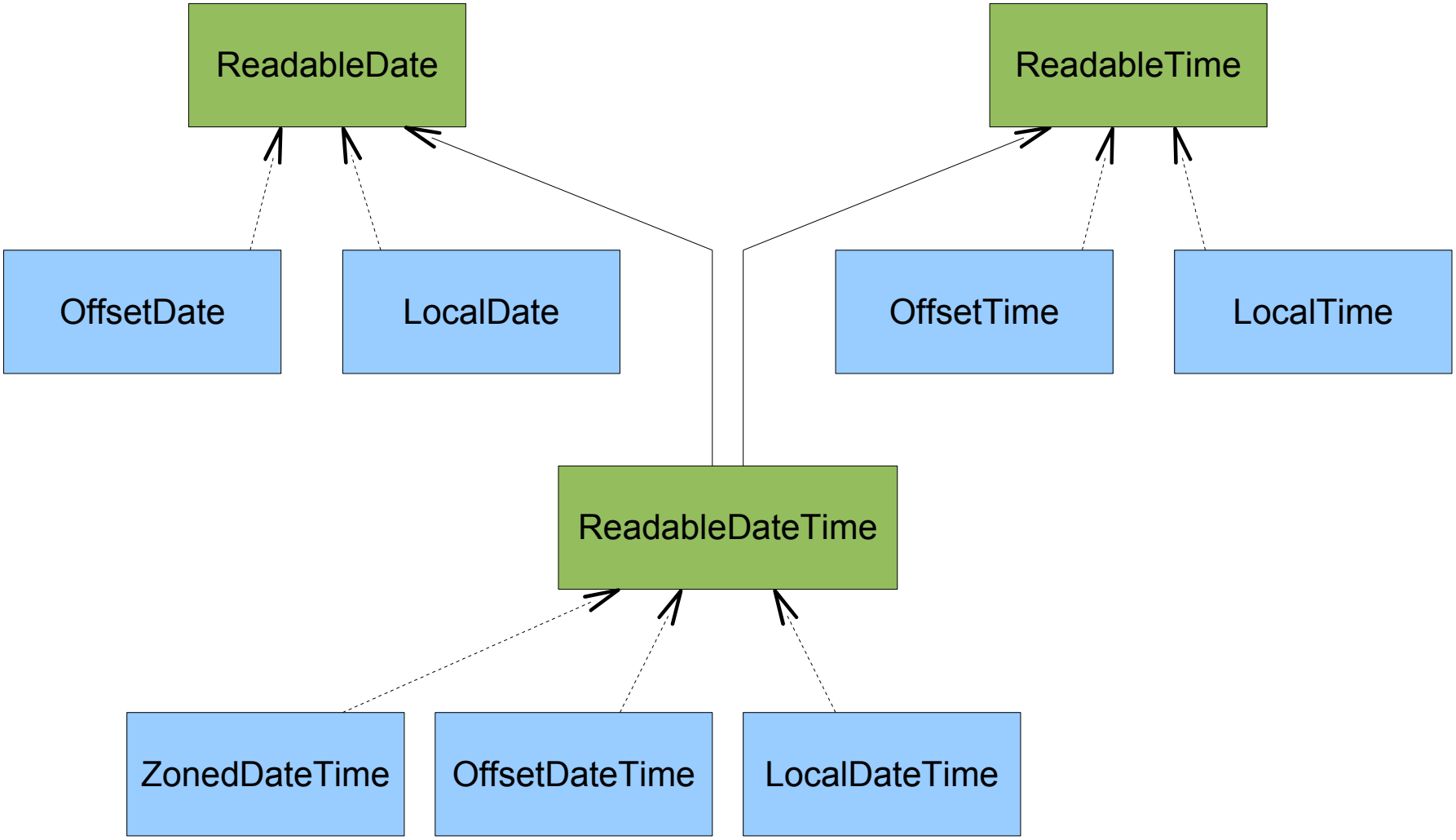
Agenda

- Problems today
- JSR-310 overview
- Continuous
- Human
- **Integration**
- Periods
- Miscellaneous

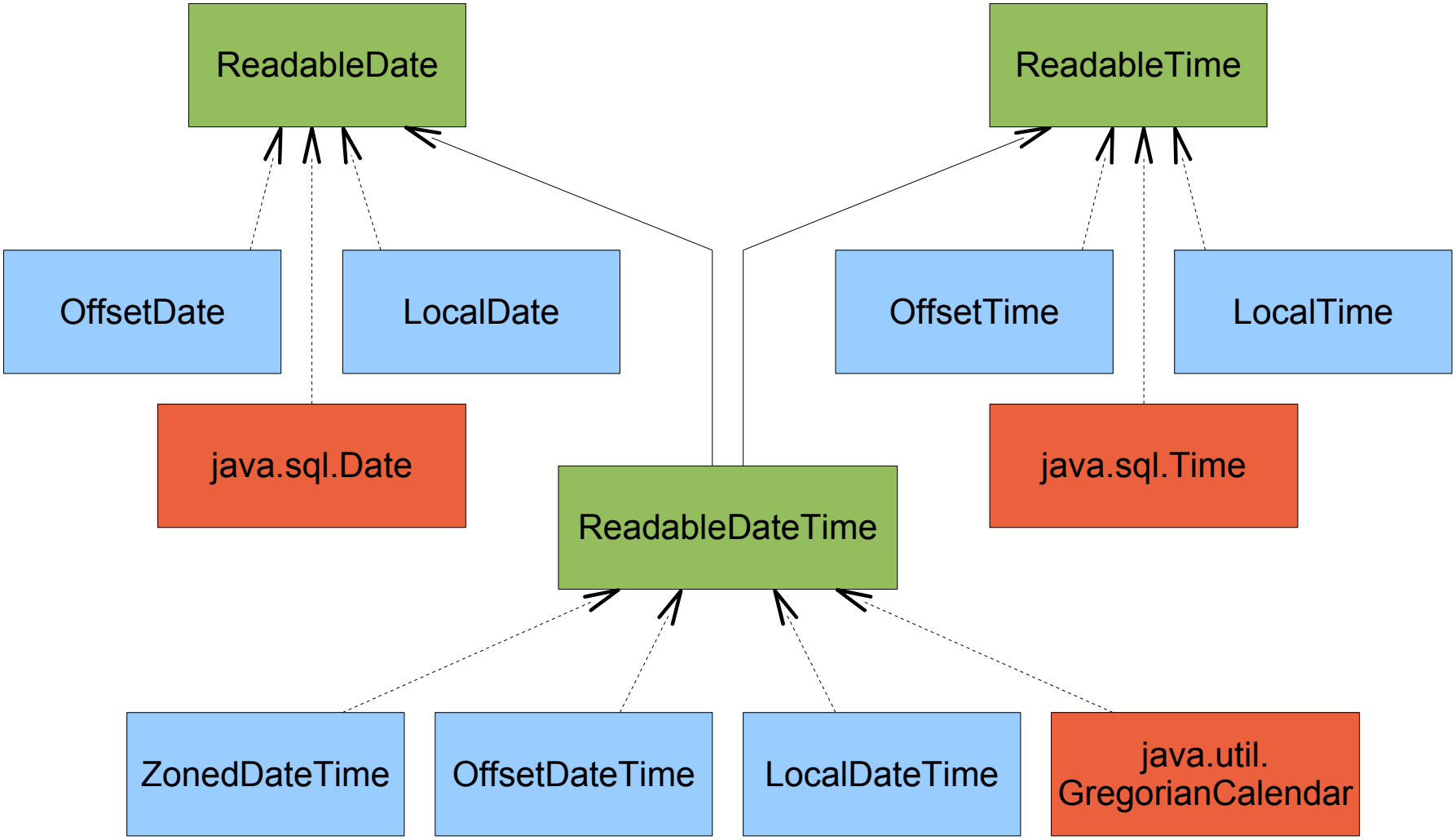
Integration via interfaces

- Simple interfaces link everything
  - `ReadableDate`
  - `ReadableTime`
  - `ReadableDateTime`
  - `ReadableInstant`
- Each provides one method
  - `toLocalDate()`
  - `toLocalTime()`
  - `toLocalDateTime()`
  - `toInstant()`

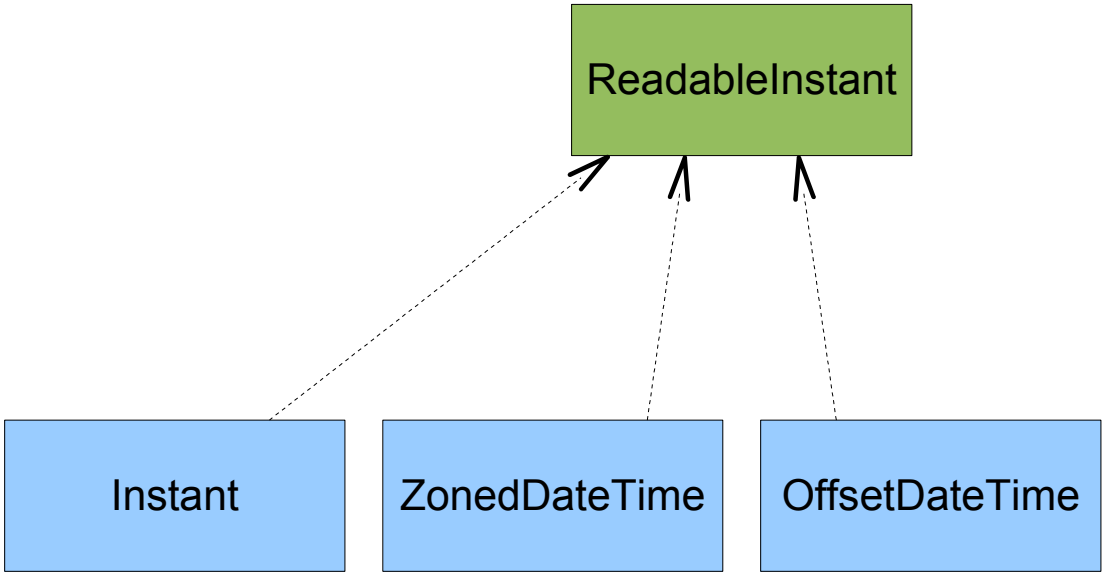
Date/time integration



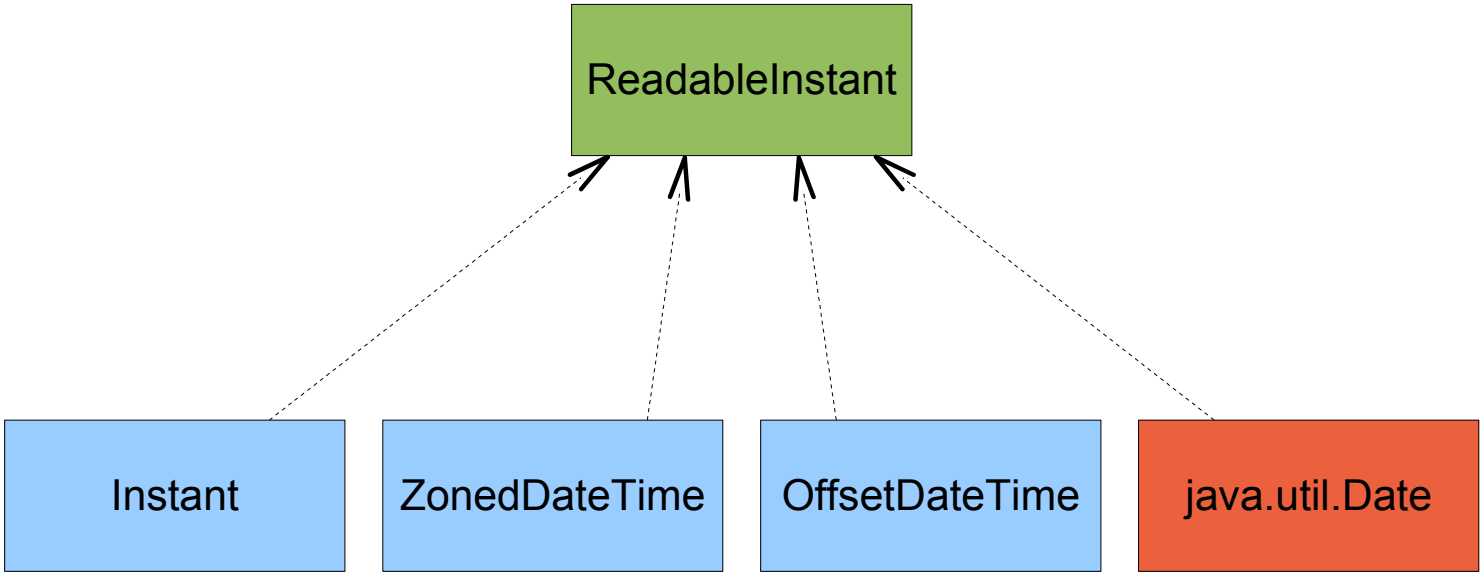
Date/time integration



Instant integration



Instant integration



Integration – Exising JDK classes

- All old JDK software date/time classes will:
  - Implement JSR-310 interfaces
  - Be constructable from JSR-310 interfaces
  - Not be deprecated
- All JSR-310 classes will:
  - Not reference the old JDK software date/time classes

Integration – Databases

- JDBC group represented on JSR-310
- Classes map onto SQL
  - `LocalDate` `DATE`
  - `LocalTime` `TIME WITHOUT TIME ZONE`
  - `LocalDateTime` `TIMESTAMP WITHOUT TIMEZONE`
  - `OffsetTime` `TIME WITH TIME ZONE`
  - `OffsetDateTime` `TIMESTAMP WITH TIME ZONE`
- Open issue on time zone mapping
  - DB time zone id differs from Java

Integration – XML

- XML opinions represented on JSR-310
- Classes map onto XML
  - `ReadableDate` `xs:date`
  - `ReadableTime` `xs:time`
  - `ReadableDateTime` `xs:datetime`
  - `YearMonth` `xs:gYearMonth`
  - `MonthDay` `xs:gMonthDay`
  - `Year` `xs:gYear`
  - `MonthOfYear` `xs:gMonth`
  - `DayOfMonth` `xs:gDay`

Agenda

- Problems today
- JSR-310 overview
- Continuous
- Human
- Integration
- Periods
- Miscellaneous

Periods

- Describe duration in human fields
  - 6 years, 2 months and 12 days
- Use cases
  - meeting length – 2 hours
  - conference length – 5 days
  - summer holiday – 6 weeks

Periods – Analysis

- Example: 6 years, 2 months and 12 days
- Generalise
  - `{years}{months}{days}`
- One class for each part
  - `Years`
  - `Months`
  - `Days`
  - and so on
- One class for combination
  - `Period`

Agenda

- Problems today
- JSR-310 overview
- Continuous
- Human
- Integration
- Periods
- Miscellaneous

Formatting and Parsing

- ISO-8601 returned by `toString()`
- Formatting
  - work in progress
- Parsing
  - work in progress
- Probably based on Joda-Time

Calendar systems

- Everything based on ISO-8601
  - current 'civil' calendar
  - not historically accurate
- Requirements
  - support common calendars in JDK software
  - not overcomplicate main use case
  - no ambiguity in API

Calendar systems

- Simple classes for other calendars
  - `HebrewDate`
  - `HinduDate`
  - `IslamicDate`
  - `JapaneseDate`
  - `ThaiDate`
  - and so on
- Subclass `Object`
- Implement `ReadableDate`
- Construct from `ReadableDate`

Current time

- Affected by time zone
  - often forgotten
- Requirements
  - stop time for test case
  - change to time in future/past
  - run time slowly

Current time

- Access current time using an object
  - avoid singleton
  - allows Inversion of Control
- Anyone can implement a subclass
  - you can control time

```
// system millis, default time zone
Instant instant = Clock.system().instant();
LocalDate date = Clock.system().today();

// system millis, specified time zone
TimeZone zone = timeZone("Europe/Moscow");
LocalDate date = Clock.system(zone).today();
```

Current time

- Supports Inversion of Control
  - inject `Clock`
  - could be a 'stop time' subclass for testing

```
public class MyForm {
  @Resource
  private Clock clock;  // injected

  public void validate(LocalDate date) {
    if (date.isBefore(clock.today())) {
      // error
    }
  }
}
```

Summary

- Current JDK date/time has problems
  - Joda-Time is the best current alternative
- JSR-310 underway
  - Continuous
    - timestamps
    - durations
  - Human
    - dates and times
    - periods
  - Formatting/parsing
  - Multiple calendar systems
  - Control over current time

Summary

- JSR-310 is open...
  - ...very open!
- Help make sure this date/time API works!
  - join the mailing list
  - comment on the wiki
  - review the API – javadoc or svn
- http://jsr-310.dev.java.net

THANK YOU

Stephen Colebourne
Michael Santos

TS-6578