**Het werken met datums en tijden in de standaard Java-API is niet echt een genot. De `java.util.Date` is bijna helemaal deprecated, de `GregorianCalendar` is niet hét antwoord en voor zaken die je in ieder project wel een keer nodig hebt, zoals 'valt tijdstip x voor, na, of in de periode tussen tijden y en z?', ben je op eigen code aangewezen.**

*thema*

# Open source: Joda-Time

## *Java datetime library*

Joda-Time is een java datetime library gehost op Sourceforge. De mensen die het project hebben opgezet menen dat er niets te redden is aan de JDK date, en zijn dus maar opnieuw begonnen. De library heeft de volgende kenmerken:

- Tijdstippen worden uitgedrukt in milliseconden sinds 1970 (`long` integer). Met behulp van `Chronology` objecten (kalendersystemen) wordt dit getal on-demand herleid tot maanden, dagen en uren.
- De default chronology is `ISO8601` (voor XML), daarnaast zijn onder andere de `Gregorian` en meer exotische zoals de `Buddhist` en `Coptic` chronology's beschikbaar. Helaas zijn er geen stardates.
- Er is onderscheid tussen instants en partials. Instants zijn tijdstippen met een exacte tegenwaarde in milliseconden (bijvoorbeeld 30-12-2004, 17:00:00.000 UTC). Partials zijn tijdstippen waarvan slechts een deel van de velden is ingevuld (bijvoorbeeld 25-1-2005) en die dus geen exacte tegenwaarde hebben.
- Een tijdstip (instant) is gemodelleerd in de `DateTime` class. Met twee instants in de tijd (of met een instant en een duration) kun je een `Interval` aanmaken. Beide classes zijn rijkelijk voorzien van comparators en aanverwante methoden.
- De partials zijn `YearMonthDay`, `DateMidnight` en `TimeOfDay`.
- De objecten in de library werken zoals de Java `String`: ze hebben een constante waarde, hun mutator methodes leveren nieuwe instanties op. Gedrag onder multithreading is daarmee heel overzichtelijk. In de javadocs staat trouwens steeds duidelijk aangegeven of een class threadsafe is of niet.
- Analoog aan de `StringBuffer` zijn er voor de verschillende date/time-objecten ook `MutableXxxxx` versies beschikbaar. Wanneer er op een date/time-instantie veel objectmanipulatie moet plaatsvinden kunnen die sneller zijn.
- De library voorziet ook in local-aware `Formatter` classes.
- De jarfile heeft geen dependency's buiten de JDK (1.3 of hoger).

**In de praktijk** Hier volgen enkele voorbeelden in code-vorm, om enig gevoel te krijgen voor hoe JodaTime er uitziet en wat het kan doen.

Het werken met instants:

```
final long KWARTIER = 15 * 60 * 1000; //mil-
liseconden
DateTime dt0, dt1, dt2, dt3, dt4;
//diverse constructors, met en zonder time-
zone, chronology params
dt0 = new DateTime(2004, 12, 30, 17, 58, 24,
033);
dt1 = new DateTime(
DateTimeZone.getInstance("Europe/Amsterdam")
);
dt2 = new DateTime(
System.currentTimeMillis() - 1000,
Chronology.getBuddhist()
);
//diverse mutators, accessors en comparators
dt3 = dt2.minus( KWARTIER );
dt4 = dt3.withZone( DateTimeZone.UTC )
.withFieldAdded( DurationFieldType.weeks(), -
5)
.withField( DateTimeFieldType.hourOfDay(),
16);
System.out.println( dt0.isBeforeNow() );
System.out.println( dt1.isAfter( dt2 ));
System.out.println( dt2.dayOfWeek().getAsText(
Locale.GERMANY ) );
```

Het werken met tijdspannes:

```
boolean b0, b1, b2, b3, b4;
//Interval loopt VANAF een instant TOT een
ander instant
Interval iv0 = new Interval( dt0, dt1 );
b0 = iv0.contains( dt0 ); //true
b1 = iv0.contains( dt1 ); //false
//Interval is eqv. een start instant plus een
aantal milliseconden
Duration dur = new Duration( KWARTIER );
Interval iv1 = new Interval( dt0, dur );
//diverse comparators
b2 = iv1.overlaps( iv0 ); //interval ->
interval
b3 = iv1.isAfter( iv0 ); //interval -> inter-
val
b3 = iv1.isAfter( dt2 ); //interval ->
instant
b4 = iv1.isBeforeNow();
```

De partials:

```
DateMidnight dmt = new DateMidnight( 2005,
12, 25 ); //25-12-05 0:00u
YearMonthDay ymd = new YearMonthDay( 2005,
12, 25 ); //25-12-05
TimeOfDay tod = new TimeOfDay( 15, 30 );
//15:30u
//DateMidnight is (ook) een instant
iv1.isBefore( dmt );
dm1.isEqualNow();
//ToD en YMD kun je echter alleen uitlezen of
converteren...
int i = tod.hourOfDay().get();
int j = ymd.year().get();
DateTime dt5 = ymd.toDateTime( tod );
DateTime dt6 = ymd.toDateMidnight();
//...comparators zouden handig zijn maar zijn
er niet (want zijn op de
//grensgevallen niet altijd definieerbaar).
tod.isBefore( SOME_OTHER_ToD ); //DOES NOT
COMPILE
iv1.contains( ymd ); //DOES NOT COMPILE
```

De formatters in actie:

```
//Formatters worden geïnstantieerd via
DateTimeFormatterBuilder
Locale LNL = new Locale("NL");
DateTimeFormatterBuilder builder = new DateTi
meFormatterBuilder(LNL);
//Twee manieren om formatters te bouwen: via
een pattern string...
DateTimeFormatter dtf0a = builder.
appendPattern("dd MMMM YYYY")
.toFormatter();
builder.clear(); //(niet vergeten als je
dezelfde builder gebruikt)
//...of via chaining. dtf0b krijgt hetzelfde
pattern als dtf0a
DateTimeFormatter dtf0b = builder.appendDay-
OfMonth(2)
.appendLiteral(' ')
.appendMonthOfYearText()
.appendLiteral(" ")
.appendYear(4, 4)
.toFormatter();
//De formatter kan schrijven naar Strings,
StringBuffers en Writers
String output = dtf0a.print( new
DateMidnight(2005, 1, 10) );
//en doubleert als parser:
DateTime read = dtf0b.parse( output );
```

## Geschikt voor productie?

• *Bugs?* Er staan op dit moment geen open bugs in de tracker. Op de gesloten bugs die vermeld staan, is steeds dezelfde dag nog gereageerd. Er is een grote verzameling unit-tests.

• *Stabiele API?* De naamgeving is heel consistent. Het ziet er niet uit alsof die nog eens overhoop gehaald hoeft te worden. Op de website staat bovendien: *'The main API has been firmed up and is not intended to change beyond now'.*

• *Bruikbare documentatie?* Ja. De javadocs zijn compleet en tamelijk gedetailleerd.

• *Portable?* Waar dit relevant is, beschikken de classes over `toDate()` en `toCalendar()` methodes voor conversie naar JDK-objecten. Let bij de conversie naar `java.util.Date` wel op de tijdzone. Als je niet oplet kun je ongewenst een paar uur verspringen.

• *Licentie?* Het project gebruikt een Apache/BSD-stijllicentie, veel vriendelijker dan de GPL. Wel eisen ze dat de mededeling *'This product includes software developed by the Joda project (http://www.joda.org/)'* in de software en/of in de documentatie wordt opgenomen.

**Download** De nieuwste versie is op het moment van schrijven van dit artikel v0.98: joda-time-0.98-src.zip (1,4 MB).

**Conclusie** Joda-time is een prima datum/tijd-library die de huidige lappendeken aan JDK-*time+fixes* kan vervangen. Het project is volwassen genoeg om in productie te nemen en de licentie staat dit ook toe. Zie ook: http://joda-time.sourceforge.net/index.html.

*Barend Garvelink is werkzaam bij Sogeti Nederland (barend.garvelink@sogeti.nl). Het Java Competence Network is een onderdeel van de divisie Distributed Software Engineering (DSE) van Sogeti Nederland B.V. (e-mail: java@sogeti.nl).*