Quick start guide

This is the quick introduction guide to Joda-Time and the features on offer. Its designed for those of you who are too impatient to read the full user guide.

Date and Time

Joda-Time includes these key datetime classes:

- `DateTime` - Immutable replacement for JDK `Calendar`
- `DateMidnight` - Immutable class representing a date where the time is forced to midnight
- `LocalDate` - Immutable class representing a local date without a time (no time zone)
- `LocalTime` - Immutable class representing a time without a date (no time zone)
- `LocalDateTime` - Immutable class representing a local date and time (no time zone)

Each datetime class provides a variety of constructors. These include the `Object` constructor. This allows you to construct, for example, `DateTime` from the following objects:

- `Date` - a JDK instant
- `Calendar` - a JDK calendar
- `String` - in ISO8601 format
- `Long` - in milliseconds
- any Joda-Time datetime class

This list is extensible. In other words Joda-Time sacrifices a little type-safety for extensibility. It does mean however, that converting from a JDK `Date` or `Calendar` to a Joda-Time class is easy - simply pass the JDK class into the constructor.

Each datetime class provides simple easy methods to access the datetime fields. For example, to access the month you can use:

```
DateTime dt = new DateTime();
int month = dt.getMonthOfYear();
```

All the main datetime classes are immutable (like String) and cannot be changed after creation. However, simple methods have been provided to alter field values in a newly created object. For example, to set the year, or add 2 hours you can use:

```
DateTime dt = new DateTime();
DateTime year2000 = dt.withYear(2000);
DateTime twoHoursLater = dt.plusHours(2);
```

In addition to the basic get methods, each datetime class provides property methods for each field. These provide access to the full wealth of Joda-Time functionality. For example, to access details about a month or year:

```
DateTime dt = new DateTime();
String monthName = dt.monthOfYear().getAsText();
String frenchShortName = dt.monthOfYear().getAsShortText(Locale.FRENCH);
boolean isLeapYear = dt.year().isLeap();
DateTime rounded = dt.dayOfMonth().roundFloorCopy();
```

Calendar systems and time zones

Joda-Time provides support for multiple calendar systems and the full range of time zones. The `Chronology` and `DateTimeZone` classes provide this support.

Joda-Time defaults to using the ISO calendar system (the calendar used by most of the business world) and the default time zone of your machine. These default values can be overridden whenever necessary. Please note that the ISO calendar system is historically inaccurate before 1583.

Joda-Time uses a pluggable mechanism for calendars. (The JDK uses subclasses such as `GregorianCalendar`.) To obtain a Joda-Time calendar, use one of the factory methods on `Chronology`.

```
Chronology coptic = CopticChronology.getInstance();
```

Time zones are implemented as part of the chronology. To obtain a Joda-Time chronology in the Tokyo time zone, you can use.

```
DateTimeZone zone = DateTimeZone.forID("Asia/Tokyo");
Chronology gregorianJuian = GJChronology.getInstance(zone);
```

Intervals and time periods

Joda-Time provides support for intervals and time periods.

An interval is represented by the `Interval` class. It holds a start and end datetime, and allows operations based around that range of time.

A time period is represented by the `Period` class. This holds a period such as 6 months, 3 days and 7 hours. You can create a `Period` directly, or derive it from an interval.

A time duration is represented by the `Duration` class. This holds an exact duration in milliseconds. You can create a `Duration` directly, or derive it from an interval.

Although a period and a duration may seem similar, they operate differently. For example, consider adding one day to a `DateTime` at the daylight savings cutover.

```
DateTime dt = new DateTime(2005, 3, 26, 12, 0, 0, 0);
DateTime plusPeriod = dt.plus(Period.days(1));
DateTime plusDuration = dt.plus(new Duration(24L*60L*60L*1000L));
```

Adding a period will add 23 hours in this case, not 24 because of the daylight savings change, thus the time of the result will still be midday. Adding a duration will add 24 hours no matter what, thus the time of the result will change to 13:00.