Calculating due date counting only working days

by Benny Bottema - Monday, February 18, 2019

https://www.bennybottema.com/2019/02/18/calculating-due-date-counting-only-working-days/

Recently I was required to calculate an expected due date based on office working. Shockingly, this is not readily available in the SDK or even as a well-known Open Source library.

Jollyday to the rescue! And some fancy Java Streams.

Contents

Introducing Jollyday

Jollyday is a small and a little bit outdated library, that still does a fine job if you use it straightforwardly.

It has all kinds of rules with which you can have a dynamically calculated holiday in case it moves if it is in the weekend (like Holland's King's Day, which moves to Saturday if it is on a Sunday). It also has validity directives so you can specify different rules for different years of the same holiday, such as when our Queen's Day moved when we got a new Queen.

It even includes Easter, which requires insane math!

Let's calculate the due date!

Configure the weekends

We'll use JDK date time API for defining the Weekend.

```
Set<DayOfWeek> WEEKEND_DAYS = newHashSet(DayOfWeek.SATURDAY, DayOfWeek
.SUNDAY);
```

with the above, you can now do something like

WEEKEND_DAYS.contains(myLocalDate.getDayOfWeek()).

There are other ways to determine if a given date lies in the Weekend, but it’s not as readable:

localDate.getDayOfWeek().getValue() >= DayOfWeek.SATURDAY.getValue())

Or using a TemporalQuery:

localDate.query(t -> t.get(ChronoField.DAY_OF_WEEK) >= 5).

Configure the holidays

Either pick a country ready to go or, if you want to configure it further, take one of the country XML’s from the library and create your own. We’ll do just that and create a new Holidays config named creatively: “MyList”.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Configuration hierarchy="ml" description="MyList" xmlns:tns="http
://www.example.org/Holiday" xmlns:xsi="http://www.w3.org/2001/XMLSchem
a-instance" xsi:schemaLocation="http://www.example.org/Holiday /Holida
y.xsd">
    <tns:Holidays>
        <tns:Fixed month="JANUARY" day="1" validFrom="2019" validTo="2
019" descriptionPropertiesKey="Nieuwjaar"/>
        <tns:Fixed month="APRIL" day="19" validFrom="2019" validTo="20
19" descriptionPropertiesKey="Goede vrijdag"/>
        etc..
    </tns:Holidays>
</tns:Configuration>
```

Now to get these holidays in Java:

```
HolidayManager HOLIDAY_MANAGER = HolidayManager.getInstance(ManagerPar
ameters.create("MyList", null));
```

Determine due date

With the weekends and holidays set, we can use Java streams to find the due date rather elegantly:

```
public LocalDate determineProcessingDueDate(LocalDate startingDate, in
t operationalDaysNeeded) {
 return Iterables.getLast(collectOperationalDaysStartingFromDate(start
ingDate, operationalDaysNeeded));
}

private List<LocalDate> collectOperationalDaysStartingFromDate(LocalDa
te startingDate, int numberOfOperationalDays) {
 return Stream.iterate(startingDate, date -> date.plusDays(1))
   .filter(localDate -> !WEEKEND_DAYS.contains(localDate.getDayOfWeek(
)))
   .filter(localDate -> !HOLIDAY_MANAGER.isHoliday(localDate))
   .limit(numberOfOperationalDays)
   .collect(Collectors.toList());
}
```

Here’s what happens:

- Generate an infinite list starting with a date and adding one day each iteration.
- Filter out any holiday or weekend days
- Keep the other days until the stream limit was reached, indicated by the number of working days we want to count
- Using Guava’s Iterables.getLast we return the date found after counting operational working days.
