LocalDateTime

CHAPTER 12 새로운 날짜와 시간 API

개요

java.time 패키지 내 존재하는 날짜,시간과 관련된 클래스 스펙에 대한 정리

LocalDate & LocalTime & LocalDateTime

LocalDate (feat. java8 doc)

사실상 LocalDate 와 LocalTime 은 대동소이한 점이 많기에, LocalTime 은 생략하도록 합니다.

- Immutable 한 객체이며, 별도의 '시간' 정보는 저장 & 표현하지 않는다. (타임존 포함)
- value-based 클래스이므로, ==, hashcode, synchronization 으로 인한 비교는 예측 불가능한 결과를 가지므로, 객체값 비교를 하고자 할 때에는 equals 메소드를 사용해라!

조금 더 아는 척을 하고 싶다면 : ISO-8601 캘린더 시스템을 이용한다고 한다.

인스턴스 생성 및 날짜값 반환

LocalDate

```
LocalDate date = LocalDate.of(2019,8,23); // 2019-08-23
LocaalDate now = LocalDate.now();
int year = date.getYear();
Monty month = date.getMonth();
int day = date.getDayOfmonth();


int year = date.get(ChronoField.YEAR);
//..........이하 생략
```

- enum 객체인 ChronoField 를 arg 로 하여 날짜값 반환을 받는 것도 가능함. (상기 코드블럭 참조)
  - ChronoField implements TemporalField
  - https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html#get-java.time.temporal.TemporalField-

parse

```
LocalDate date = LocalDate.parse("2019-08-23");
```

- 정형화된 YYYY-MM-DD 포멧 뿐만 아니라, DateTimeFormatter 를 arg 로 하여 정형화되어있지 않은 날짜 포멧팅의 경우로 처리도 가능.
  - 포멧팅이 맞지 않는 경우, DateTimeParseException 을 발생시킴. (implements RuntimeException)

LocalTime

- LocalDate 의 시간 버전

- 시간 버전이라는 점만 제외하면 LocalDate 의 spec 과 같다고 보면 된다.
- 초의 경우 nanosecond 단위까지 저장

LocalDateTime

- LocalDate + LocalTime
- 공식 문서의 내용도 LocalDate 의 특징과 LocalTime 의 특징을 합한 내용으로 기술되어있다.

Instant

- LocalDateTime 은 사람 관점에서 보기 편한 시간단위(년,월,일,시,분,초)
- Instant 는 컴퓨터 관점에서 보기 편한 시간단위.
  - 익히 알고 있는 Unix timestamp. 기준시간 : 1970/01/01 00:00:00
- 나노초 단위까지 저장

인스턴스 생성 및 날짜값 반환

```
Instant.ofEpochSecond(3); // 1970/01/01 00:00:03
Instant.ofEpochSecond(4, 1000000000); // 1970/01/01 00:00:05 <-- 기준시간에서 4 초를 더한 후, 1 억 나노초를 더하여 5 초
```

당연히 day,month 값과 같은, human readable 한 값을 얻고 싶을 땐

```
int day = Instant.now().get(ChronoField.DAY_OF_MONTH);
```

feat. java8 doc

- 나노초 단위 spec 까지 저장하기 위해서는, 현재 primitive data type 의 long 변수 하나만으로는 시간 정보를 다 저장할 수가 없음
  - private final long seconds;
  - private final int nanos;
- 이렇게 초/나노초 단위를 각각 long, int 타입으로 저장하여, 시간의 표현 범위는 무한대에 가깝게 늘렸다.

    - -1000000000-01-01T00:00Z 부터, 1000000000-12-31T23:59:59.999999999Z 까지. (기원전 10 억년 ~ 서기 10 억년)
- 기타... 윤년과 같은 개념으로 지구의 자전시간과 초 단위의 보정을 위한 spec 이 Time-scale 라는 항목으로 문서에 기술 되어있으니 뭔가 더 아는 척을 하고 싶다면 참고하시면 될 듯

Duration & Period

- 특정 시간이 아닌, '기간' 내지는 지속시간 단위를 표현하고 싶은 경우에는 Duration & Period 를 이용한다
- Temporal 인터페이스를 구현한 LocalDate,LocalTime,LocalDateTime,Instant 객체의 기간 정보를 관리하게 해 주는 클래스이다.

Duration

- Instant 클래스와 같은 저장 방식을 갖고 있다. long 변수 하나만으로는 시간 정보를 다 저장할 수 없기에, long(초) & int(나노초) 두개의 인스턴스 변수를 내장한 채로 기간을 표현한다.
    - **주의사항 : 그래서 '초' 정보를 갖고 있는 Temporal 객체만 사용 가능**

사용법

```
Duration d1 = Duration.between(time1, time2);    // LocalTime
Duration d2 = Duration.between(dateTime1, dateTime2);  // LocalDateTime
Duration d3 = Duration.between(instant1, instant2);  // Instant


Duration d4 = Duration.between(date1, date2);  // LocalDate --> throws
Exception
Duration d5 = Duration.between(time1, instant2) // throws Exception
```

- 분명 Duration 의 between 메서드는 Temporal 인터페이스를 구현한 객체를 파라미터로 받는데,
  파라미터로 LocalDate 를 넣는 경우 Exception 을 throw 한다.
- LocalDate 는 왜 안되는가?
    - Date 에는 초 단위가 없기 때문. Duration 은 '초' 정보를 갖고 있는 Temporal 객체만 사용 가능하다고 하였다!
- 그럼 반쪽짜리 기간 관리인 것 아닌가?

- Period 를 쓰자.

Period

- Duration 의 '날짜 이상 단위' 버전
- Duration 이 시,분,초 단위의 기간을 표현하였다면, Period 는 날짜 이상의 단위의 기간을 표현한다.

사용법

```
Period p1 = Period.between(LocalDate.of(2020,2,28), LocalDate.of(2020,3,1));
// 2 (윤년까지 계산해 준다!)
```

날짜 조정,파싱,포매팅

- LocalDate 등과 같은 값을 조정하고자 할 때(년,월,일 변경)
  - withXXXX 메소드를 통해 LocalDate 를 변경할 수 있다.
    - 하지만, 여태 학습한 객체는 다 immutable 하기에, 객체의 값이 변경되는 것이 아니라 새로운 객체를 생성하여 값을 리턴해 주는 구조이므로 사용시에 참고.

예제

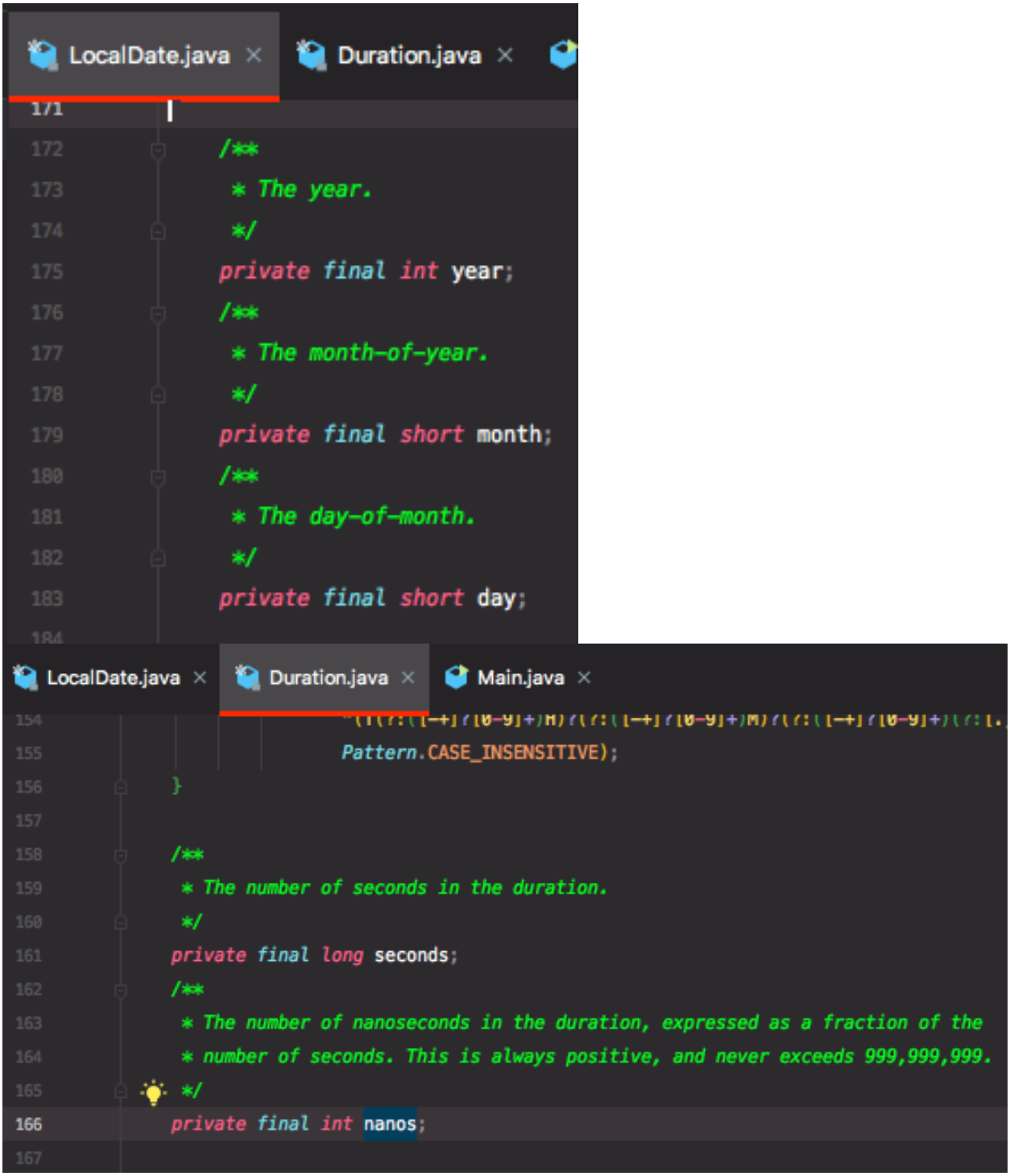
```
LocalDate date1 = LocalDate.of(2019,8,23);  // 2019-08-23
LocalDate date2 = date1.withYear(2011);    // 2011-08-23
LocalDate date3 = date2.withDayOfMonth(25); // 2011-08-25


LocalDate date4 = date3.with(ChronoField.MONTH_OF_YEAR, 2); //2011-02-25
```

- 값이 적용은 되나, date1,2,3,4 모두 다 다른 레퍼런스를 갖고 있는 새로운 객체변수임을 자각하자

증거

- class 파일을 까 보면, 내장 객체의 변수들이 죄다 final 로 설정되어 있음을 알 수 있다.



그러므로

이런 방법으로는 날짜가 변경되지 않는다..

```
LocalDate date = LocalDate.of(2019,8,23);
date.withYear(2011);


sysout(date);     // 2019-08-23. 2011-08-23 이 출력되지 않는다!
```

TemporalAdjusters

- 특정 날짜가 아니라, '이번달 마지막 날', 내지는, '6 월의 첫 수요일 날' 과 같은, 계산식으로는 조금 복잡하게 생각을 해야 하는 상황에서 유용하게 쓰일 수 있다.

default 로 구현되어있는 메소드 리스트(feat. java8 document)

https://docs.oracle.com/javase/8/docs/api/java/time/temporal/TemporalAdjusters.html

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static TemporalAdjuster	dayOfWeekInMonth(int ordinal, DayOfWeek dayOfWeek) Returns the day-of-week in month adjuster, which returns a new date in the same month with the ordinal day-of-week.
static TemporalAdjuster	firstDayOfMonth() Returns the "first day of month" adjuster, which returns a new date set to the first day of the current month.
static TemporalAdjuster	firstDayOfNextMonth() Returns the "first day of next month" adjuster, which returns a new date set to the first day of the next month.
static TemporalAdjuster	firstDayOfNextYear() Returns the "first day of next year" adjuster, which returns a new date set to the first day of the next year.
static TemporalAdjuster	firstDayOfYear() Returns the "first day of year" adjuster, which returns a new date set to the first day of the current year.
static TemporalAdjuster	firstInMonth(DayOfWeek dayOfWeek) Returns the first in month adjuster, which returns a new date in the same month with the first matching day-of-week.
static TemporalAdjuster	lastDayOfMonth() Returns the "last day of month" adjuster, which returns a new date set to the last day of the current month.
static TemporalAdjuster	lastDayOfYear() Returns the "last day of year" adjuster, which returns a new date set to the last day of the current year.
static TemporalAdjuster	lastInMonth(DayOfWeek dayOfWeek) Returns the last in month adjuster, which returns a new date in the same month with the last matching day-of-week.
static TemporalAdjuster	next(DayOfWeek dayOfWeek) Returns the next day-of-week adjuster, which adjusts the date to the first occurrence of the specified day-of-week after the date being adjusted.
static TemporalAdjuster	nextOrSame(DayOfWeek dayOfWeek) Returns the next-or-same day-of-week adjuster, which adjusts the date to the first occurrence of the specified day-of-week after the date being adjusted unless it is already on that day in which case the same object is returned.
static TemporalAdjuster	ofDateAdjuster(UnaryOperator<LocalDate> dateBasedAdjuster) Obtains a TemporalAdjuster that wraps a date adjuster.
static TemporalAdjuster	previous(DayOfWeek dayOfWeek) Returns the previous day-of-week adjuster, which adjusts the date to the first occurrence of the specified day-of-week before the date being adjusted.
static TemporalAdjuster	previousOrSame(DayOfWeek dayOfWeek) Returns the previous-or-same day-of-week adjuster, which adjusts the date to the first occurrence of the specified day-of-week before the date being adjusted unless it is already on that day in which case the same object is returned.

사용법

```
LocalDate localDate = LocalDate.of(2019,8,23);
LocalDate getFirstDay = localDate.with(TemporalAdjusters.firstDayOfMonth());
// 2019-08-01
LocalDate lastDayOfYear = localDate.with(TemporalAdjusters.lastDayOfYear());
// 2019-12-31
```

사내 시스템에서 유용하게 사용할 수 있는 영역

- 배송도착안내 시스템
- 금 or 토요일 주문시, 영업일이 아닌 토요일이나 일요일의 경우 배송이 도착하지 않음.
- TemporalAdjusters 를 이용하면, 주문을 한 날 D-Day 를 기준으로 소요되는 영업일만 계산하여 배송도착시점을 알 수가 있음.

그래서 직접 간단하게 만들어 봄

```
public class ExpectedDeliveredDay implements TemporalAdjuster {

    private final int nextDayShipDueHour = 17;
    /**
     * 17 시 이후는 익일 발송, 17 시 이전은 당일 발송
     */
    @Override
    public Temporal adjustInto(Temporal temporal) {

        int shipPeriod = 1; //배송소요일은 발송일 기준으로 다음날(+1 일)

        //발송일 정의
        DayOfWeek shipDayOfWeek =
DayOfWeek.of(temporal.get(ChronoField.DAY_OF_WEEK)); //기본은 당일 발송
        if (temporal.get(ChronoField.HOUR_OF_DAY) > nextDayShipDueHour) {
//17 시를 넘길 시 익일 발송
            shipDayOfWeek = shipDayOfWeek.plus(1);
        }
        //배송일이 토 -> 이틀이 더해져 월요일 발송
        //배송일이 일 -> 하루가 더해져 월요일 발송
        if (shipDayOfWeek == DayOfWeek.SATURDAY || shipDayOfWeek ==
DayOfWeek.SUNDAY) {
            shipDayOfWeek = DayOfWeek.MONDAY;
        }

        Temporal shipTemporalDay =
temporal.with(TemporalAdjusters.nextOrSame(shipDayOfWeek));
        return shipTemporalDay.plus(shipPeriod, ChronoUnit.DAYS);
    }
}
```

DateTimeFormatter

- 말 그대로 Temporal 을 상속받아 사용하는 LocalDateTime 류의 날짜 표시를 특정 포메팅에 의거, 이쁘게 포메팅 해 주는 클래스

사용법

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMdd");
LocalDate dt = LocalDate.of(2019,8,23);
String result = dt.format(formatter); // 20190823  LocaalDate to String
LocalDate.parse(result, formatter); // String to LocalDate
```

Builder 패턴을 이용하여 직접 DateTimeFormatter 객체를 만들 수도 있다.

```
DateTimeFormatter myFormatter = new DateTimeFormatterBuilder()
        .appendText(ChronoField.YEAR)
        .appendLiteral("   ")
        .appendText(ChronoField.MONTH_OF_YEAR)
        .appendLiteral(". ")
        .appendText(ChronoField.DAY_OF_MONTH)
        .parseCaseInsensitive()
        .toFormatter()
```

```
LocalDate dt = LocalDate.of(2019,8,23);

dt.format(myFormatter);       // 2019 8 월. 23
```