

# Java의 정석

## 제 10 장

## 날짜와 시간 & 형식화

2017. 8. 23

남궁성 강의

castello@naver.com

## 1.1 Calendar와 Date

### ▶ java.util.Date

- 날짜와 시간을 다룰 목적으로 만들어진 클래스(JDK1.0)
- Date의 메서드는 거의 deprecated되었지만, 여전히 쓰이고 있다.

### ▶ java.util.Calendar

- Date클래스를 개선한 새로운 클래스(JDK1.1). 여전히 단점이 존재

### ▶ java.time패키지

- Date와 Calendar의 단점을 개선한 새로운 클래스들을 제공(JDK1.8)

## 1.2 Calendar

### ▶ java.util.Calendar

- 추상 클래스이므로 getInstance()를 통해 구현된 객체를 얻어야 한다.

```
Calendar cal = new Calendar(); // 에러!!! 추상클래스는 인스턴스를 생성할 수 없다.
```

```
// OK, getInstance() 메서드는 Calendar 클래스를 구현한 클래스의 인스턴스를 반환한다.  
Calendar cal = Calendar.getInstance();
```

### ▶ Date와 Calendar간의 변환

#### 1. Calendar를 Date로 변환

```
Calendar cal = Calendar.getInstance();  
...  
Date d = new Date(cal.getTimeInMillis()); // Date(long date)
```

#### 2. Date를 Calendar로 변환

```
Date d = new Date();  
...  
Calendar cal = Calendar.getInstance();  
cal.setTime(d)
```

## 1.3 Calendar의 주요 메서드(1/3) – get()

### ▶ get()으로 날짜와 시간 필드 가져오기 – int get(int field)

```
Calendar cal = Calendar.getInstance(); // 현재 날짜와 시간으로 셋팅됨
int thisYear = cal.get(Calendar.YEAR); // 올해가 몇년인지 알아낸다.
int lastDayOfMonth = cal.getActualMaximum(Calendar.DATE); // 이 달의 마지막날
```

### ▶ Calendar에 정의된 필드

필드명	설 명
YEAR	년
MONTH	월(0부터 시작)
WEEK_OF_YEAR	일
WEEK_OF_MONTH	그 달의 몇 번째 주
DATE	일
DAY_OF_MONTH	그 달의 몇 번째일
DAY_OF_YEAR	그 해의 몇 번째일
DAY_OF_WEEK	요일
DAY_OF_WEEK_IN_MONTH	그 달의 몇 번째 요일

필드명	설 명
HOURL	시간(0~11)
HOURL_OF_DAY	시간(0~23)
MINUTE	분
SECOND	초
MILLISECOND	천분의 일초
ZONE_OFFSET	GMT기준 시차(천분의 일초 단위)
AM_PM	오전/오후

## 1.4 Calendar의 주요 메서드(2/3) – set()

### ▶ set()으로 날짜와 시간지정하기

```
void set(int field, int value)
void set(int year, int month, int date)
void set(int year, int month, int date, int hourOfDay, int minute)
void set(int year, int month, int date, int hourOfDay, int minute, int second)
```

- 날짜 지정하는 방법. 월(MONTH)이 0부터 시작한다는 점에 주의

```
Calendar date1 = Calendar.getInstance();
date1.set(2017, 7, 15); // 2017년 8월 15일 (7월 아님)
// date1.set(Calendar.YEAR, 2017);
// date1.set(Calendar.MONTH, 7);
// date1.set(Calendar.DATE, 15);
```

- 시간 지정하는 방법.

```
Calendar time1 = Calendar.getInstance();
time1.set(Calendar.HOUR_OF_DAY, 10); // time1을 10시 20분 30초로 설정
time1.set(Calendar.MINUTE, 20);
time1.set(Calendar.SECOND, 30);
```

## 1.5 Calendar의 주요 메서드(3/3) – clear()

### ▶ clear()와 clear(int field)로 Calendar객체 초기화 하기

- clear()는 Calendar객체의 모든 필드를 초기화

```
Calendar dt = Calendar.getInstance();

// Tue Aug 29 07:13:03 KST 2017
System.out.println(new Date(dt.getTimeInMillis()));

dt.clear(); // 모든 필드를 초기화
// Thu Jan 01 00:00:00 KST 1970
System.out.println(new Date(dt.getTimeInMillis()));
```

- clear(int field)는 Calendar객체의 특정 필드를 초기화

```
Calendar dt = Calendar.getInstance();

// Tue Aug 29 07:13:03 KST 2017
System.out.println(new Date(dt.getTimeInMillis()));

dt.clear(Calendar.SECOND); // 초를 초기화
dt.clear(Calendar.MINUTE); // 분을 초기화
dt.clear(Calendar.HOUR_OF_DAY); // 시간을 초기화
dt.clear(Calendar.HOUR); // 시간을 초기화

// Tue Aug 29 00:00:00 KST 2017
System.out.println(new Date(dt.getTimeInMillis()));
```

# 2.1 DecimalFormat – 숫자의 형식화(1/2)

- 숫자를 다양한 형식(패턴)으로 출력할 수 있게 해준다.

```
double number = 1234567.89;
DecimalFormat df = new DecimalFormat("#.##E0");
String result = df.format(number); // "1.2E6"
```

기호	의미	패턴	결과(1234567.89)
0	10진수(값이 없을 때는 0)	0 0.0 0000000000.0000	1234568 1234567.9 0001234567.8900
#	10진수	# #.# #####.####	1234568 1234567.9 1234567.89
.	소수점	##	1234567.9
-	음수부호	##- -##	1234567.9- -1234567.9
,	단위 구분자	###,## ####,##	1,234,567.89 123,4567.89
E	지수기호	#E0 0E0 ##E0 00E0 ###E0 0000E0 #.#E0 0.0E0 0.000000000E0 #####E0	.1E7 1E6 1.2E6 12E5 123.5E4 1235E3 1.2E6 1.2E6 1.234567890E6 1.23456789E6

## 2.1 DecimalFormat – 숫자의 형식화(2/2)

- 특정 형식으로 되어 있는 문자열에서 숫자를 뽑아낼 수도 있다.

```
DecimalFormat df = new DecimalFormat("#,###.##");
Number num = df.parse("1,234,567.89");

double d = num.doubleValue();
```

[참고] Integer.parseInt()는 콤마(,)가 포함된 문자열을 숫자로 변환 못함

기호	의미	패턴	결과(1234567.89)
;	패턴구분자	#,###,##+;#,###,##-	1,234,567.89+ (양수일 때) 1,234,567.89- (음수일 때)
%	퍼센트	#,##%	123456789%
\u2030	퍼밀 (퍼센트 x 10)	#,#\u2030	1234567890%
\u00A4	통화	\u00A4 #,###	₩ 1,234,568
'	escape문자	'##,### "##,###	#1,234,568 '1,234,568



## 2.2 SimpleDateFormat – 날짜의 형식화(1/2)

- 날짜와 시간을 다양한 형식으로 출력할 수 있게 해준다.

```
Date today = new Date();
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");

// 오늘 날짜를 yyyy-MM-dd형태로 변환하여 반환한다.
String result = df.format(today);
```

기호	의미	보기
G	연대(BC, AD)	AD
y	년도	2006
M	월(1~12 또는 1월~12월)	10 또는 10월, OCT
w	년의 몇 번째 주(1~53)	50
W	월의 몇 번째 주(1~5)	4
D	년의 몇 번째 일(1~366)	100
d	월의 몇 번째 일(1~31)	15
F	월의 몇 번째 요일(1~5)	1
E	요일	월

## 2.2 SimpleDateFormat – 날짜의 형식화(2/2)

- 특정 형식으로 되어 있는 문자열에서 날짜와 시간을 뽑아낼 수도 있다.

```
DateFormat df = new SimpleDateFormat("yyyy년 MM월 dd일");  
DateFormat df2 = new SimpleDateFormat("yyyy/MM/dd");  
Date d = df.parse("2015년 11월 23일"); // 문자열을 Date로 변환  
String result = df2.format(d);
```

기호	의미	보기
a	오전/오후 (AM, PM)	PM
H	시간(0~23)	20
k	시간(1~24)	13
K	시간(0~11)	10
h	시간(1~12)	11
m	분(0~59)	35
s	초(0~59)	55
S	천분의 일초(0~999)	253
z	Time zone (General time zone)	GMT+9:00
Z	Time zone (RFC 822 time zone)	+0900
'	escape문자(특수문자를 표현하는데 사용)	없음

## 2.3 ChoiceFormat – 범위의 형식화

- ChoiceFormat은 특정 범위에 속하는 값을 문자열로 변환해준다.
- if문이나 switch문으로 처리하기 복잡한 경우에 유용하다.
- 패턴 구분자 '#'는 경계값을 포함하고 '<'는 포함하지 않는다.

```
class ChoiceFormatEx1 {
    public static void main(String[] args) {
        double[] limits = {60, 70, 80, 90}; // 오름차순으로 정렬된 double값들
        String[] grades = {"D", "C", "B", "A"}; // limits와의 순서와 개수를 맞추어야 한다.


        ChoiceFormat form = new ChoiceFormat(limits, grades);

        int[] scores = { 100, 95, 88, 70, 52, 60, 70};

        for(int i=0;i<scores.length;i++) {
            System.out.println(scores[i]+":"+form.format(scores[i]));
        }
    } // main
}
```

```
double[] limits = {60, 70, 80, 90};
String[] grades = {"D", "C", "B", "A"};

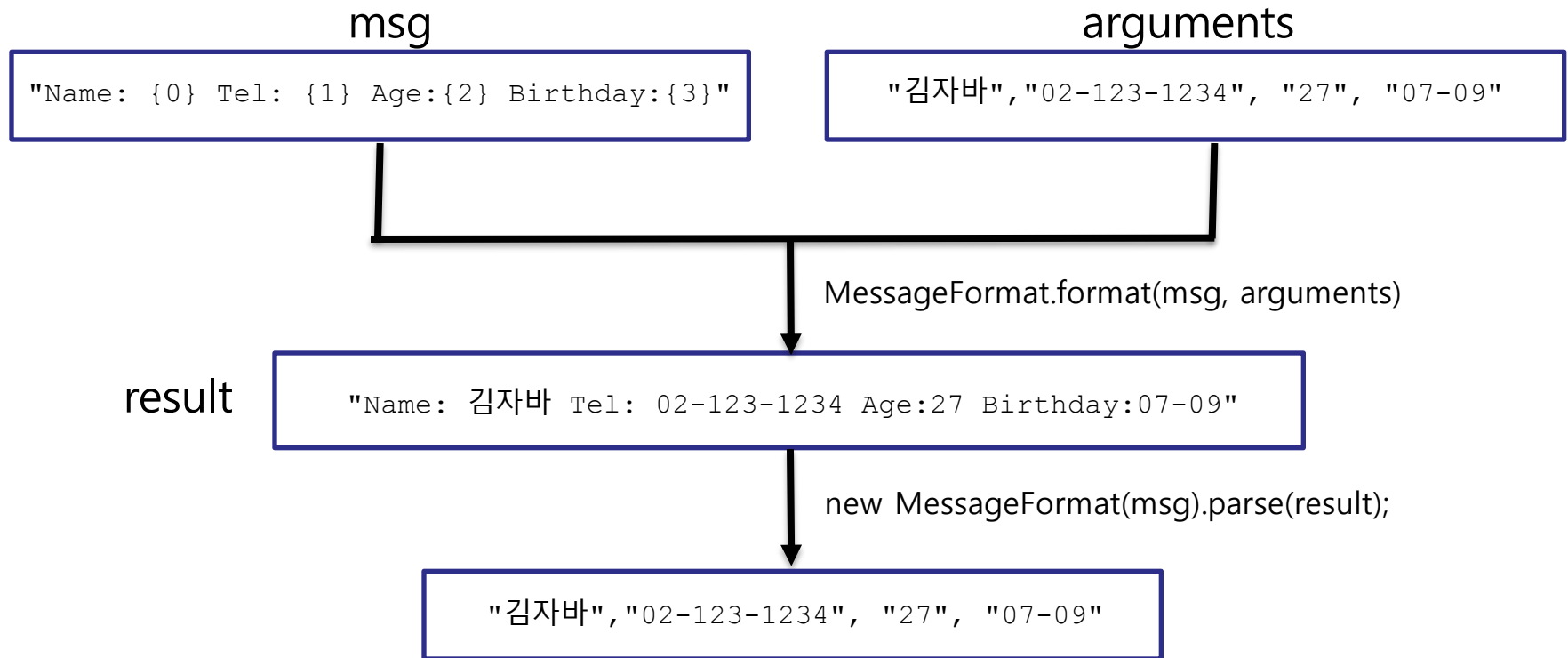
ChoiceFormat form = new ChoiceFormat(limits, grades);
```



```
String pattern = "60#D|70#C|80#B|90#A";
ChoiceFormat form = new ChoiceFormat(pattern);
```

## 2.4 MessageFormat – 텍스트의 형식화

- 데이터를 정해진 양식에 맞춰 출력할 수 있도록 도와준다.
- 특정 형식을 가진 문자열에서 데이터를 뽑아낼 때도 유용하다.



## 3.1 java.time 패키지

- Date, Calendar의 단점을 보완하기 위해 추가된 패키지(JDK1.8부터)
- 이 패키지에 속한 클래스들은 모두 '불변(immutable)'이다.

패키지	설명
java.time	날짜와 시간을 다루는데 필요한 핵심 클래스들을 제공
java.time.chrono	표준(ISO)이 아닌 달력 시스템을 위한 클래스들을 제공
java.time.format	날짜와 시간을 파싱하고, 형식화하기 위한 클래스들을 제공
java.time.temporal	날짜와 시간의 필드(field)와 단위(unit)를 위한 클래스들을 제공
java.time.zone	시간대(time-zone)와 관련된 클래스들을 제공

▲ 표10-3 java.time패키지와 서브 패키지들

## 3.2 java.time패키지의 핵심 클래스(1/3)

- 날짜를 표현할 때는 LocalDate, 시간을 표현할 때는 LocalTime을 사용
- 날짜와 시간을 같이 표현할 때는 LocalDateTime을 사용

LocalDate + LocalTime → LocalDateTime  
날짜 시간 날짜 & 시간

- 시간대(time-zone)까지 다뤄야 할 때는 ZonedDateTime을 사용

LocalDateTime + 시간대 → ZonedDateTime

- Period는 날짜간의 차이를, Duration은 시간의 차이를 표현할 때 사용

날짜 - 날짜 = Period  
시간 - 시간 = Duration

## 3.2 java.time 패키지의 핵심 클래스(2/3)

- Temporal : 날짜와 시간을 표현하는 클래스들이 구현
- TemporalAmount : 날짜와 시간의 차이를 표현하는 클래스가 구현

**Temporal, TemporalAccessor, TemporalAdjuster를 구현한 클래스**

– LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Instant 등

**TemporalAmount를 구현한 클래스**

– Period, Duration

- Temporal로 시작하는 인터페이스들은 매개변수 타입으로 많이 사용되며, TemporalAmount인지 아닌지만 구별하면 된다.

## 3.2 java.time패키지의 핵심 클래스(3/3)

- TemporalUnit : 날짜와 시간의 단위를 정의해 놓은 인터페이스
- TemporalField : 년, 월, 일 등 날짜와 시간의 필드를 정의해 놓음.

```
int get(TemporalField field)
LocalDate plus(long amountToAdd, TemporalUnit unit)
```

- ChronoUnit은 TemporalUnit을, ChronoField는 TemporalField를 구현

```
LocalDate today      = LocalDate.now(); // 오늘
LocalDate tomorrow = today.plus(1, ChronoUnit.DAYS); // 오늘에 +1일
// LocalDate tomorrow = today.plusDays(1); // 위의 문장과 동일

LocalTime now = LocalTime.now(); // 현재 시간
int minute = now.getMinute();    // 현재 시간에서 분 (minute) 만 뽑아낸다.
// int minute = now.get(ChronoField.MINUTE_OF_HOUR); // 위의 문장과 동일
```



## 3.3 LocalDate와 LocalTime

- java.time패키지의 핵심. 이 두 클래스를 잘 이해하면 나머지는 쉬움.
- now()는 현재 날짜 시간을, of()는 특정 날짜 시간을 지정할 때 사용

```
LocalDate today = LocalDate.now(); // 오늘의 날짜
LocalTime now = LocalTime.now(); // 현재 시간
```

```
LocalDate birthDate = LocalDate.of(1999, 12, 31); // 1999년 12월 31일
LocalTime birthTime = LocalTime.of(23, 59, 59); // 23시 59분 59초
```

- 일 단위나 초 단위로도 지정가능(1일은 24\*60\*60=86400초)

```
LocalDate birthDate = LocalDate.ofYearDay(1999, 365); // 1999년 12월 31일
LocalTime birthTime = LocalTime.ofSecondDay(86399); // 23시 59분 59초
```

- parse()로 문자열을 LocalDate나 LocalTime으로 변환할 수 있다.

```
LocalDate birthDate = LocalDate.parse("1999-12-31"); // 1999년 12월 31일
LocalTime birthTime = LocalTime.parse("23:59:59"); // 23시 59분 59초
```

### 3.3 LocalDate와 LocalTime - 필드값 가져오기(1/2)

- LocalDate와 LocalTime에서 특정 필드의 값 가져오는 메서드

클래스	메서드	설명(1999-12-31 23:59:59)
LocalDate	int getYear( )	년도 (1999)
	int getMonthValue( )	월 (12)
	Month getMonth( )	월 (DECEMBER) getMonth( ).getValue( ) = 12
	int getDayOfMonth( )	일 (31)
	int getDayOfYear( )	같은 해의 1월 1일부터 몇번째 일 (365)
	DayOfWeek getDayOfWeek( )	요일 (FRIDAY) getDayOfWeek( ).getValue( ) = 5
	int lengthOfMonth( )	같은 달의 총 일수 (31)
	int lengthOfYear( )	같은 해의 총 일수 (365), 윤년이면 366
	boolean isLeapYear( )	윤년여부 확인 (false)
LocalTime	int getHour( )	시 (23)
	int getMinute( )	분 (59)
	int getSecond( )	초 (59)
	int getNano( )	나노초 (0)

### 3.3 LocalDate와 LocalTime – 필드값 가져오기(2/2)

- LocalDate와 LocalTime에서 특정 필드 값 가져오기 - get(), getLong()

```
int    get    (TemporalField field)
long   getLong(TemporalField field)
```

- get(), getLong()에 사용할 수 있는 필드의 목록(\*는 getLong()사용)

TemporalField(ChronoField)	설명	TemporalField(ChronoField)	설명
ERA	시대	EPOCH_DAY *	EPOCH(1970.1.1)부터 몇번째 날
YEAR_OF_ERA, YEAR	년	MINUTE_OF_DAY	그 날의 몇 번째 분(시간을 분으로 환산)
MONTH_OF_YEAR	월	SECOND_OF_DAY	그 날의 몇 번째 초(시간을 초로 환산)
DAY_OF_WEEK	요일(1:월, 2:화, ... 7:일)	MILLI_OF_DAY	그 날의 몇 번째 밀리초( $=10^{-3}$ 초)
DAY_OF_MONTH	일	MICRO_OF_DAY *	그 날의 몇 번째 마이크로초( $=10^{-6}$ 초)
AMPM_OF_DAY	오전/오후	NANO_OF_DAY *	그 날의 몇 번째 나노초( $=10^{-9}$ 초)
HOUR_OF_DAY	시간(0~23)	ALIGNED_WEEK_OF_MONTH	그 달의 n번째 주(1~7일 1주, 8~14일 2주, ...)
CLOCK_HOUR_OF_DAY	시간(1~24)	ALIGNED_WEEK_OF_YEAR	그 해의 n번째 주(1월 1~7일 1주, 8~14일 2주, ...)
HOUR_OF_AMPM	시간(0~11)	ALIGNED_DAY_OF_WEEK_IN_MONTH	요일(그 달의 1일을 월요일로 간주하여 계산)
CLOCK_HOUR_OF_AMPM	시간(1~12)	ALIGNED_DAY_OF_WEEK_IN_YEAR	요일(그 해의 1월 1일을 월요일로 간주하여 계산)
MINUTE_OF_HOUR	분	INSTANT_SECONDS	년월일을 초단위로 환산(1970-01-01 00:00:00 UTC를 0초로 계산) Instant에만 사용가능
SECOND_OF_MINUTE	초	OFFSET_SECONDS	UTC와의 시차. ZoneOffset에만 사용가능
MILLI_OF_SECOND	천분의 일초( $=10^{-3}$ 초)	PROLEPTIC_MONTH	년월을 월단위로 환산(2015년11월=2015*12+11)
MICRO_OF_SECOND *	백만분의 일초( $=10^{-6}$ 초)		
NANO_OF_SECOND *	10억분의 일초( $=10^{-9}$ 초)		
DAY_OF_YEAR	그 해의 몇번째 날		

## 3.3 LocalDate와 LocalTime – 필드값 변경하기

- with(), plus(), minus()로 특정 필드의 값을 변경(새로운 객체가 반환됨)

```
LocalDate with(TemporalField field, long newValue)
```

```
LocalTime plus(TemporalAmount amountToAdd)
```

```
LocalTime plus(long amountToAdd, TemporalUnit unit)
```

```
LocalDate plus(TemporalAmount amountToAdd)
```

```
LocalDate plus(long amountToAdd, TemporalUnit unit)
```

TemporalUnit(ChronoUnit)	설명
FOREVER	Long.MAX_VALUE초 (약 3천억년)
ERAS	1,000,000,000년
MILLENNIA	1,000년
CENTURIES	100년
DECADES	10년
YEARS	년
MONTHS	월
WEEKS	주
DAYS	일
HALF_DAYS	반나절
HOURS	시
MINUTES	분
SECONDS	초
MILLIS	천분의 일초 ( $=10^{-3}$ )
MICROS	백만분의 일초 ( $=10^{-6}$ )
NANOS	10억분의 일초 ( $=10^{-9}$ )

### 3.3 LocalDate와 LocalTime – 날짜와 시간의 비교

- 날짜와 시간을 비교할 때, isAfter(), isBefore(), isEqual()를 사용

```
boolean isAfter (ChronoLocalDate other)
boolean isBefore(ChronoLocalDate other)
boolean isEqual (ChronoLocalDate other)    // LocalDate에만 있음
```

- compareTo()로도 비교할 수 있다.

```
int result = date1.compareTo(date2); //같으면 0, date1이 이전이면 -1, 이후면 1
```

- 대부분의 경우, isEqual() 대신 equals()를 사용해도 된다.

```
LocalDate    kDate = LocalDate.of(1999, 12, 31);
JapaneseDate jDate = JapaneseDate.of(1999, 12, 31);

System.out.println(kDate.equals(jDate));    // false    연대가 다름
System.out.println(kDate.isEqual(jDate));    // true
```

## 3.4 Instant – java.util.Date를 대체

- 에포크 타임(1970-01-01 00:00:00 UTC)부터 경과된 시간을 표현
- 날짜와 시간과 달리 단일 진법(10진법)이라 계산에 편리

```
Instant now = Instant.now();  
Instant now2 = Instant.ofEpochSecond(now.getEpochSecond());  
Instant now3 = Instant.ofEpochSecond(now.getEpochSecond(),  
                                       now.getNano());
```

- 나노초 단위가 아니라 밀리초 단위의 에포크 타임이 필요할 때

```
long toEpochMilli()
```

- Instant와 Date간의 변환에 사용하는 메서드

```
static Date    from(Instant instant)    // Instant → Date  
Instant toInstant()                    // Date → Instant
```

## 3.5 LocalDateTime과 ZonedDateTime(1/3)

- LocalDateTime은 LocalDate와 LocalTime을 합쳐놓은 것
- ZonedDateTime은 LocalDateTime에 시간대(time zone)를 추가한 것

```
LocalDate    + LocalTime  → LocalDateTime
LocalDateTime + 시간대    → ZonedDateTime
```

- now(), of()로 LocalDateTime만들기. 다양한 종류의 of()가 있음

```
// 2015년 12월 31일 12시 34분 56초
```

```
LocalDateTime dateTime = LocalDateTime.of(2015, 12, 31, 12, 34, 56);
LocalDateTime today    = LocalDateTime.now();
```

- LocalDate와 LocalTime으로 LocalDateTime만들기

```
LocalDate date = LocalDate.of(2015, 12, 31);
LocalTime time = LocalTime.of(12, 34, 56);
```

```
LocalDateTime dt  = LocalDateTime.of(date, time);
LocalDateTime dt2 = date.atTime(time);
LocalDateTime dt3 = time.atDate(date);
LocalDateTime dt4 = date.atTime(12, 34, 56);
LocalDateTime dt5 = time.atDate(LocalDate.of(2015, 12, 31));
LocalDateTime dt6 = date.atStartOfDay(); // dt6 = date.atTime(0,0,0);
```

## 3.5 LocalDateTime과 ZonedDateTime(2/3)

- LocalDateTime을 LocalDate 또는 LocalTime으로 변환하기

```
LocalDateTime dt = LocalDateTime.of(2015, 12, 31, 12, 34, 56);  
LocalDate date = dt.toLocalDate(); // LocalDateTime → LocalDate  
LocalTime time = dt.toLocalTime(); // LocalDateTime → LocalTime
```

- LocalDateTime으로 ZonedDateTime만들기

```
ZoneId      zid = ZoneId.of("Asia/Seoul");  
ZonedDateTime zdt = dateTime.atZone(zid);  
System.out.println(zdt); // 2015-11-27T17:47:50.451+09:00[Asia/Seoul]
```

[참고] 사용가능한 ZoneId의 목록은 ZoneId.getAvailableZoneIds()로 얻을 수 있다.

- 현재 특정시간대(예를 들어 뉴욕)의 시간을 알고 싶을 때

```
ZoneId      nyId  = ZoneId.of("America/New_York");  
ZonedDateTime nyTime = ZonedDateTime.now().withZoneSameInstant(nyId);
```



## 3.6 ZonedDateTime의 변환

- ZonedDateTime을 변환하는데 사용되는 메서드

LocalDate	toLocalDate()
LocalTime	toLocalTime()
LocalDateTime	toLocalDateTime()
OffsetDateTime	toOffsetDateTime()
long	toEpochSecond()
Instant	toInstant()

- ZonedDateTime을 GregorianCalendar(Calendar)로 변환하는 방법

```
// ZonedDateTime → GregorianCalendar
GregorianCalendar from(ZonedDateTime zdt)

// GregorianCalendar → ZonedDateTime
ZonedDateTime toZonedDateTime()
```

## 3.7 ZoneOffset과 OffsetDateTime

- ZoneOffset은 UTC(표준시)로부터 얼마만큼 떨어져있는지 표현에 사용  
한국은 UTC+9 즉 UTC보다 9시간 빠르다.

```
// 현재 위치(서울)의 ZoneOffset을 얻은다음, offset을 초단위로 구한다.  
ZoneOffset krOffset = ZonedDateTime.now().getOffset();  
int krOffsetInSec = krOffset.get(ChronoField.OFFSET_SECONDS); // 32400초(9시간)
```

- OffsetDateTime은 ZoneOffset으로 시간대를 표현  
(ZonedDateTime은 ZoneId로 시간대를 표현. ZoneId는 시간대 관련 규칙 포함)
- OffsetDateTime는 서로 다른 시간대의 컴퓨터간의 통신에 유용

```
ZonedDateTime zdt = ZonedDateTime.of(date, time, zid);  
OffsetDateTime odt = OffsetDateTime.of(date, time, krOffset);  
  
// ZonedDateTime → OffsetDateTime  
OffsetDateTime odt = zdt.toOffsetDateTime();
```

## 3.8 TemporalAdjusters

- plus(), minus()로 계산하기에 복잡한 날짜계산을 도와준다.

// TemporalAdjusters로 다음주 월요일을 알아낸다.

```
LocalDate today = LocalDate.now();
```

```
LocalDate nextMonday = today.with(TemporalAdjusters.next (DayOfWeek.MONDAY));
```

- TemporalAdjusters가 제공하는 메서드

메서드	설명
firstDayOfNextYear()	다음 해의 첫 날
firstDayOfNextMonth()	다음 달의 첫 날
firstDayOfYear()	올 해의 첫 날
firstDayOfMonth()	이번 달의 첫 날
lastDayOfYear()	올 해의 마지막 날
lastDayOfMonth()	이번 달의 마지막 날
firstInMonth (DayOfWeek dayOfWeek)	이번 달의 첫 번째 ?요일
lastInMonth (DayOfWeek dayOfWeek)	이번 달의 마지막 ?요일
previous (DayOfWeek dayOfWeek)	지난 ?요일(당일 미포함)
previousOrSame (DayOfWeek dayOfWeek)	지난 ?요일(당일 포함)
next (DayOfWeek dayOfWeek)	다음 ?요일(당일 미포함)
nextOrSame (DayOfWeek dayOfWeek)	다음 ?요일(당일 포함)
dayOfWeekInMonth(int ordinal, DayOfWeek dayOfWeek)	이번 달의 n번째 ?요일

## 3.9 Period와 Duration(1/4)

- Period는 날짜의 차이를, Duration은 시간의 차이를 계산하기 위한 것

날짜 - 날짜 = Period  
시간 - 시간 = Duration

- 두 날짜 또는 시간의 차이를 구할 때는 between()을 사용

```
// 두 날짜의 차이 구하기
LocalDate date1 = LocalDate.of(2014, 1, 1);
LocalDate date2 = LocalDate.of(2015, 12, 31);

Period pe = Period.between(date1, date2);

// 두 시각의 차이 구하기
LocalTime time1 = LocalTime.of(00, 00, 00);
LocalTime time2 = LocalTime.of(12, 34, 56); // 12시 34분 56초

Duration du = Duration.between(time1, time2);
```

## 3.9 Period와 Duration(2/4)

- 특정 필드의 값을 얻을 때는 get()을 사용

```
long year  = pe.get(ChronoUnit.YEARS);    // int getYears()
long month = pe.get(ChronoUnit.MONTHS);    // int getMonths()
long day   = pe.get(ChronoUnit.DAYS);      // int getDays()

long sec   = du.get(ChronoUnit.SECONDS);    // long getSeconds()
int nano   = du.get(ChronoUnit.NANOS);     // int getNano()
```

- getHours(), getMinute()같은 메서드가 없다.

```
System.out.println(pe.getUnits()); // [Years, Months, Days]
System.out.println(du.getUnits()); // [Seconds, Nanos]
```

- 시분초를 구할 때는 Duration을 LocalTime으로 변환하는 것이 편리

```
LocalTime tmpTime = LocalTime.of(0,0).plusSeconds(du.getSeconds());

int hour = tmpTime.getHour();
int min  = tmpTime.getMinute();
int sec  = tmpTime.getSecond();
int nano = du.getNano();
```

## 3.9 Period와 Duration(3/4)

- until()은 between()과 거의 같다.(between()은 static메서드)
- Period는 년월일을 분리해서 저장하므로, D-day구할 때는 until()이 낫다.

```
// Period pe = Period.between(today, myBirthDay);  
Period pe = today.until(myBirthDay);  
long dday = today.until(myBirthDay, ChronoUnit.DAYS);
```

- LocalTime에도 until()이 있지만, Duration을 반환하는 until()은 없다.

```
long sec = LocalTime.now().until(endTime, ChronoUnit.SECONDS);
```

- of(), ofXXX(), with(), withXXX()

```
Period pe      = Period.of(1, 12, 31); // 1년 12개월 31일  
Duration du    = Duration.of(60, ChronoUnit.SECONDS); // 60초  
// Duration du = Duration.ofSeconds(60); // 위의 문장과 동일.  
  
// withYears(), withMonths(), withDays()  
pe = pe.withYears(2); // 1년에서 2년으로 변경.  
du = du.withSeconds(120); // 60초에서 120초로 변경. withNanos()
```

## 3.9 Period와 Duration(4/4)

- plus(), minus()외에도 곱셈과 나눗셈하는 메서드도 있다.

```
pe = pe.minusYears(1).multipliedBy(2); // 1년을 빼고, 2배를 곱한다.  
du = du.plusHours(1).dividedBy(60);   // 1간을 더하고 60으로 나눈다.
```

- isNegative()와 isZero()를 사용하면 날짜와 시간의 순서를 확인가능

```
boolean sameDate = Period.between(date1, date2).isZero();  
boolean isBefore = Duration.between(time1, time2).isNegative();
```

- 다른 단위로 변환. toTotalMonths(), toDays(), toHours(), toMinutes()

클래스	메서드	설명
Period	long toTotalMonths()	년월일을 월단위로 변환해서 반환(일 단위는 무시)
Duration	long toDays()	일단위로 변환해서 반환
	long toHours()	시간단위로 변환해서 반환
	long toMinutes()	분단위로 변환해서 반환
	long toMillis()	천분의 일초 단위로 변환해서 반환
	long toNanos()	나노초 단위로 변환해서 반환

## 3.10 날짜와 시간의 형식화(formatting)

- java.time.format패키지 : 형식화와 관련된 클래스를 제공
- DateTimeFormatter의 format()를 사용해서 날짜와 시간을 형식화

```

LocalDate date = LocalDate.of(2016, 1, 2);
String yyyymmdd = DateTimeFormatter.ISO_LOCAL_DATE.format(date);
String yyyymmdd = date.format(DateTimeFormatter.ISO_LOCAL_DATE);
System.out.println(yyyymmdd); // "2016-01-02"

```

DateTimeFormatter	설명	보기
ISO_DATE_TIME	Date and time with ZoneId	2011-12-03T10:15:30+01:00[Europe/Paris]
ISO_LOCAL_DATE	ISO Local Date	2011-12-03
ISO_LOCAL_TIME	Time without offset	10:15:30
ISO_LOCAL_DATE_TIME	ISO Local Date and Time	2011-12-03T10:15:30
ISO_OFFSET_DATE	ISO Date with offset	2011-12-03+01:00
ISO_OFFSET_TIME	Time with offset	10:15:30+01:00
ISO_OFFSET_DATE_TIME	Date Time with Offset	2011-12-03T10:15:30+01:00
ISO_ZONED_DATE_TIME	Zoned Date Time	2011-12-03T10:15:30+01:00[Europe/Paris]
ISO_INSTANT	Date and Time of an Instant	2011-12-03T10:15:30Z
BASIC_ISO_DATE	Basic ISO date	20111203
ISO_DATE	ISO Date with or without offset	2011-12-03+01:00 2011-12-03
ISO_TIME	Time with or without offset	10:15:30+01:00 10:15:30
ISO_ORDINAL_DATE	Year and day of year	2012-337
ISO_WEEK_DATE	Year and Week	2012-W48-6
RFC_1123_DATE_TIME	RFC 1123 / RFC 822	Tue, 3 Jun 2008 11:05:30 GMT



## 3.11 로케일(locale)에 종속된 형식화

- ofLocalizedDate(), ofLocalizedTime(), ofLocalizedDateTime()

```
DateTimeFormatter formatter
    = DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT);
String shortFormat = formatter.format(LocalDate.now());
```

- FormatStyle의 종류에 따른 출력형태는 다음과 같다.

FormatStyle	날짜	시간
FULL	2015년 11월 28일 토요일	N/A
LONG	2015년 11월 28일 (토)	오후 9시 15분 13초
MEDIUM	2015. 11. 28	오후 9:15:13
SHORT	15. 11. 28	오후 9:15

## 3.12 출력형식 직접 정의하기

- DateFormatter의 ofPattern()으로 직접 출력형식 작성하기

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy/MM/dd");
```

- 출력형식의 작성에 사용되는 기호의 목록

기호	의미
G	연대(BC, AD)
y 또는 u	년도
M 또는 L	월(1~12 또는 1월~12월)
Q 또는 q	분기(quarter)
w	년의 몇 번째 주(1~53)
W	월의 몇 번째 주(1~5)
D	년의 몇 번째 일(1~366)
d	월의 몇 번째 일(1~31)
F	월의 몇 번째 요일(1~5)
E 또는 e	요일
a	오전/오후(AM, PM)
H	시간(0~23)
k	시간(1~24)
K	시간(0~11)

기호	의미
h	시간(1~12)
m	분(0~59)
s	초(0~59)
S	천분의 일초(0~999)
A	천분의 일초(그 날의 0시 0
n	나노초(0~999999999)
N	나노초(그 날의 0시 0분 0
V	시간대 ID(VV)
z	시간대(time-zone) 이름
O	지역화된 zone-offset
Z	zone-offset
X 또는 x	zone-offset (Z는 +00:00
,	escape문자(특수문자를 표현하는데 사용)

## 3.13 문자열을 날짜와 시간으로 파싱하기

- parse()를 이용하면 문자열을 날짜와 시간으로 파싱할 수 있다.

```
static LocalDateTime parse(CharSequence text)
static LocalDateTime parse(CharSequence text, DateTimeFormatter formatter)
```

- DateTimeFormatter에 정의된 형식을 사용할 때는 다음과 같이 한다.

```
LocalDate date =
    LocalDate.parse("2016-01-02", DateTimeFormatter.ISO_LOCAL_DATE);
```

- 자주 사용되는 형식은 ISO\_LOCAL\_DATE 등을 사용하지 않고 파싱 가능

```
LocalDate      newDate      = LocalDate.parse("2001-01-01");
LocalTime      newTime      = LocalTime.parse("23:59:59");
LocalDateTime  newDateTime  = LocalDateTime.parse("2001-01-01T23:59:59");
```

- ofPattern()으로 파싱할 수도 있다.

```
DateTimeFormatter pattern =
    DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
LocalDateTime endOfYear =
    LocalDateTime.parse("2015-12-31 23:59:59", pattern);
```