yonggyo1981 동영상 강의 URL 추가 2 years ago

Name	Name	Last commit date
..		
day12	java.time 패키지 강의 작성	2 years ago
README.md	동영상 강의 URL 추가	2 years ago
data.txt	형식화 클래스 강의 작성	2 years ago

README.md

강의 동영상 링크

동영상 링크

형식화 클래스

DecimalFormat

- 형식화 클래스 중에서 숫자를 형식화 하는데 사용된다.
- 숫자 데이터를 정수, 부동소숫점, 금액 등의 다양한 형식으로 표현할 수 있으며, 반대로 일정한 형식의 텍스트 데이터를 숫자로 쉽게 변환할 수 있다.
- 형식화 클래스에서는 원하는 형식으로 표현 또는 변환하기 위해서 패턴을 정의한다.

DateFormat의 패턴에서 사용되는 기호

기호	의미	패턴	결과
0	10진수(값이 없을 때는 0)	0 0.0 0000000000.0000	1234568 1234567.9 1234567.8900
#	10진수	# #.# ##########.####	1234568 1234567.9 1234567.89
.	소수점	#.#	1234567.9
-	음수부호	#.#- -#.#	1234567.9- -1234567.9
,	단위 구분자	#,###.##	1,124,567.89 123,4567.89
;	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

day12/format/DecimalFormatEx1.java - 숫자를 형식화된 문자열로 변환

```
package day12.format;

import java.text.*;

public class DecimalFormatEx1 {
	public static void main(String[] args) {
		double number = 1234567.89;
		String[] pattern = {
			"0",
			"#",
			"0.0",
			"0000000000.0000",
			"##########.####",
			"#.#-",
```

```
                        "-#.#",
                        "#,###.##",
                        "#,####.##",
                        "#,###.##+;#,###.##-",
                        "#.#%",
                        "#.#\u2030",
                        "\u00A4 #,###",
                        "'#'#.###",
                        "''#,###"
                };

                for (int i = 0; i < pattern.length; i++) {
                        DecimalFormat df = new
DecimalFormat(pattern[i]);
                        System.out.printf("%19s : %s\n", pattern[i],
df.format(number));
                }
        }
}
```

실행결과

```
                  0 : 1234568
                  # : 1234568
                0.0 : 1234567.9
    0000000000.0000 : 0001234567.8900
    ##########.#### : 1234567.89
               #.#- : 1234567.9-
               -#.# : -1234567.9
           #,###.## : 1,234,567.89
          #,####.## : 123,4567.89
#,###.##+;#,###.##- : 1,234,567.89+
               #.#% : 123456789%
               #.#‰ : 1234567890‰
            ¤ #,### : ? 1,234,568
            '#'#.### : #1234567.89
             ''#,### : '1,234,568
```

day12/format/DecimalFormatEx2.java - 패턴이 있는 문자열을 숫자로 변환

```
package day12.format;

import java.text.*;

public class DecimalFormatEx2 {
        public static void main(String[] args) {
                DecimalFormat df = new DecimalFormat("#,###.##");
                DecimalFormat df2 = new DecimalFormat("0.0000");

                try {
                        Number num = df.parse("1,234,567.89");
                        System.out.print("1,234,567.89 -> ");
```

```
                        double d = num.doubleValue();
                        System.out.print(d + " -> ");

                        System.out.println(df2.format(num));
                } catch (ParseException e) {
                        e.printStackTrace();
                }

        }
}
```

실행결과
```
1,234,567.89 -> 1234567.89 -> 1234567.8900
```

SimpleDateFormat

- Date와 Calendar를 사용해서 날짜 데이터를 원하는 형태로 다양하게 출력하는 것은 불편하고 복잡하나, SimpleDateFormat을 사용하면 이러한 문제들이 간단하게 해결된다.
- DateFormat은 추상클래스로 SimpleDateFormat의 상위 클래스이다. DateFormat는 추상클래스이므로 인스턴스를 생성하기 위해서는 getDateInstance()와 같은 static메서드를 이용해야 한다. getDateInstance()에 의해서 반환되는 것은 DateFormat을 상속받아 완전하게 구현한 SimpleDateFormat인스턴스이다.

SimpleDateFormat의 패턴에 사용되는 기호

기호	의미	보기
G	연대(BC,AD)	AD
y	년도	2022
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(1~365)	100
d	월의 몇 번째 일(1~31)	15
F	월의 몇 번째 요일(1~5)	1
E	요일	월
a	오전/오후(AM, PM)	PM

기호	의미	보기
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
`	escape 문자(특수문자를 표현하는데 사용)	없음

day12/format/DateFormatEx1.java - Date 인스턴스를 형식화된 날짜 문자열로 변환

```
package day12.format;

import java.util.*;
import java.text.*;

public class DateFormatEx1 {
	public static void main(String[] args) {
		Date today = new Date();

		SimpleDateFormat sdf1, sdf2, sdf3, sdf4;
		SimpleDateFormat sdf5, sdf6, sdf7, sdf8, sdf9;

		sdf1 = new SimpleDateFormat("yyyy-MM-dd");
		sdf2 = new SimpleDateFormat("''yy년 MMM dd일 E요일");
		sdf3 = new SimpleDateFormat("yyyy--MM-dd HH:mm:ss.SSS");
		sdf4 = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss a");

		sdf5 = new SimpleDateFormat("오늘은 올 해의 D번째 날입니
다.");
		sdf6 = new SimpleDateFormat("오늘은 이 달의 d번째 날입니
다.");
		sdf7 = new SimpleDateFormat("오늘은 올 해의 w번째 주입니
다.");
		sdf8 = new SimpleDateFormat("오늘은 이 달의 W번재 주입니
다.");
		sdf9 = new SimpleDateFormat("오늘은 이 달의 F번째 E요일 입
니다.");
```

```
                System.out.println(sdf1.format(today));
                System.out.println(sdf2.format(today));
                System.out.println(sdf3.format(today));
                System.out.println(sdf4.format(today));
                System.out.println();
                System.out.println(sdf5.format(today));
                System.out.println(sdf6.format(today));
                System.out.println(sdf7.format(today));
                System.out.println(sdf8.format(today));
                System.out.println(sdf9.format(today));

        }
}

실행 결과
2022-05-04
'22년 5월 04일 수요일
2022--05-04 23:17:44.147
2022-05-04 11:17:44 오후

오늘은 올 해의 124번째 날입니다.
오늘은 이 달의 4번째 날입니다.
오늘은 올 해의 19번째 주입니다.
오늘은 이 달의 1번재 주입니다.
오늘은 이 달의 1번째 수요일 입니다.
```

day12/format/DateFormatEx2.java - Calendar클래스를 이용한 Date인스턴스를 형식화된 날짜 문자열로 변환

```
package day12.format;

import java.util.*;
import java.text.*;

public class DateFormatEx2 {
        public static void main(String[] args) {
                Calendar cal = Calendar.getInstance();
                cal.set(2022, 4, 4); // 2022년 5월 4일 - Month는 0~11의
범위를 갖는다.

                Date day = cal.getTime();

                SimpleDateFormat sdf1, sdf2, sdf3, sdf4;
                sdf1 = new SimpleDateFormat("yyyy-MM-dd");
                sdf2 = new SimpleDateFormat("yy-MM-dd E요일");
                sdf3 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
                sdf4 = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss a");

                System.out.println(sdf1.format(day));
                System.out.println(sdf2.format(day));
                System.out.println(sdf3.format(day));
```

```
                System.out.println(sdf4.format(day));
        }
}
```

실행결과
2022-05-04
22-05-04 수요일
2022-05-04 23:23:24.437
2022-05-04 11:23:24 오후

day12/format/DateFormatEx3.java - 형식화된 문자열을 Date 인스턴스로 변환

```
package day12.format;

import java.util.*;
import java.text.*;

public class DateFormatEx3 {
        public static void main(String[] args) {
                DateFormat df = new SimpleDateFormat("yyyy년 MM월 dd일");
                DateFormat df2 = new SimpleDateFormat("yyyy/MM/dd");

                try {
                        Date d = df.parse("2022년 05월 04일");
                        System.out.println(df2.format(d));
                } catch (ParseException e) {
                        e.printStackTrace();
                }
        }
}
```

실행결과
2022/05/04

ChoiceFormat

특정 범위에 속하는 값을 문자열로 변환해준다.

day12/format/ChoiceFormatEx1.java

```
package day12.format;

import java.text.*;

public class ChoiceFormatEx1 {
        public static void main(String[] args) {
                double[] limits = {60, 70, 80, 90}; // 낮은 값부터 큰 값
```

```
의 순서로 적어야한다.
                // limits, grades간의 순서와 개수를 맞추어야 한다.
                String[] grades = {"D","C", "B", "A"};

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

                ChoiceFormat form = new ChoiceFormat(limits, grades);

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

실행결과
100:A
95:A
88:B
70:C
52:D
60:D
60:D
```

day12/format/ChoiceFormatEx2.java

```
package day12.format;

import java.text.*;

public class ChoiceFormatEx2 {
        public static void main(String[] args) {
                String pattern = "60#D|70#C|80<B|90#A";
                int[] scores = { 91, 90, 80, 88, 70, 52, 60 };

                ChoiceFormat form = new ChoiceFormat(pattern);

                for(int i=0; i<scores.length; i++) {

System.out.println(scores[i]+":"+form.format(scores[i]));
                }
        }
}

실행결과
91:A
90:A
80:C
88:B
70:C
```

```
52:D
60:D
```

MessageFormat

- 데이터를 정해진 양식에 맞게 출력할 수 있도록 도와준다.(format)
- 지정된 양식에서 필요한 데이터만 손쉽게 추출할 수 있다.(parse)

day12/format/MessageFormatEx1.java

```
package day12.format;

import java.text.*;

public class MessageFormatEx1 {
	public static void main(String[] args) {
		String msg = "Name: {0} \nTel: {1} \nAge: {2}
\nBirthday: {3}";

		Object[] arguments = {
			"고애신", "02-123-1234", "27", "07-09"
		};

		String result = MessageFormat.format(msg,  arguments);
		System.out.println(result);
	}
}

실행결과
Name: 고애신
Tel: 02-123-1234
Age: 27
Birthday: 07-09
```

day12/format/MessageFormatEx2.java

홑따옴표(')는 MessageFormat의 양식에 escape문자로 사용되기 때문에 문자열 msg내에서 홑따옴표를 사용하려면 홑따옴표를 연속으로 두 번 사용해야 한다.

```
package day12.format;

import java.text.*;

public class MessageFormatEx2 {
	public static void main(String[] args) {
		String tableName = "CUST_INFO";
		String msg = "INSERT INTO " + tableName + " VALUES
```

```
(''{0}'', ''{1}'', {2}, ''{3}'');";

                Object[][] arguments = {
                                {"최유진", "02-123-1234", "27", "07-09"},
                                {"고애신", "032-333-1234", "33", "10-07"}
                };

                for(int i = 0; i < arguments.length; i++) {
                        String result = MessageFormat.format(msg,
arguments[i]);
                        System.out.println(result);
                }
        }
}
```

실행결과

```
INSERT INTO CUST_INFO VALUES ('최유진', '02-123-1234', 27, '07-09');
INSERT INTO CUST_INFO VALUES ('고애신', '032-333-1234', 33, '10-07');
```

day12/format/MessageFormatEx3.java

parse(String source)를 이용해서 출력된 데이터로 부터 필요한 데이터를 추출하는 방법

```
package day12.format;

import java.text.*;

public class MessageFormatEx3 {
        public static void main(String[] args) throws ParseException {
                String[] data = {
                                "INSERT INTO CUST_INFO VALUES ('최유진',
'02-123-1234', 27, '07-09');",
                                "INSERT INTO CUST_INFO VALUES ('고애신',
'032-1234-1234', 33, '10-07');"
                };

                String pattern = "INSERT INTO CUST_INFO VALUES ({0},
{1}, {2}, {3});";
                MessageFormat mf = new MessageFormat(pattern);

                for(int i = 0; i < data.length; i++) {
                        Object[] objs = mf.parse(data[i]);
                        for(int j = 0; j < objs.length; j++) {
                                System.out.print(objs[j] + ",");
                        }
                        System.out.println();
                }
        }
}
```

실행결과

```
'최유진','02-123-1234',27,'07-09',
'고애신','032-1234-1234',33,'10-07',
```

day12/format/MessageFormatEx4.java

```
package day12.format;

import java.util.*;
import java.text.*;
import java.io.*;

public class MessageFormatEx4 {
        public static void main(String[] args) throws ParseException,
IOException {
                String tableName = "CUST_INFO";
                String fileName = "data.txt";
                String msg = "INSERT INTO " + tableName + " VALUES ({0},
{1},{2},{3});";

                Scanner s = new Scanner(new File(fileName));
                String pattern = "{0},{1},{2},{3}";
                MessageFormat mf = new MessageFormat(pattern);

                while(s.hasNextLine()) {
                        String line = s.nextLine();
                        Object[] objs = mf.parse(line);
                        System.out.println(MessageFormat.format(msg,
objs));
                }
        }
}

data.txt
'최유진','02-123-1234',27,'07-09'
'고애신','032-333-1234',33,'10-07'

실행결과
INSERT INTO CUST_INFO VALUES ('최유진','02-123-1234',27,'07-09');
INSERT INTO CUST_INFO VALUES ('고애신','032-333-1234',33,'10-07');
```

날짜와 시간

Calendar와 Date

- Date는 날짜와 시간을 다룰 목적으로 JDK1.0부터 제공되어온 클래스이다.

- Date클래스의 기능이 부족했기 때문에 Calendar라는 새로운 클래스를 그 다음 버전인 JDK1.1부터 제공하기 시작하였다.
- Calendar는 Date보다는 낳았지만 몇가지 단점들이 발견 되어 JDK1.8부터 **java.time 패키지**로 기존의 단점을 개선한 새로운 클래스들이 추가 되었다.

Calendar와 GregorianCalendar

- Calendar는 추상 클래스이기 때문에 직접 객체를 생성할 수 없고, 메서드를 통해서 완전히 구현된 클래스의 인스턴스를 얻어야 한다.

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

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

- Calendar를 상속받아 완전히 구현한 클래스로는 GregorianCalendar와 BuddhistCalendar가 있다.
- getInstance()는 시스템의 국가와 지역설정을 확인해서 태국인 경우에는 BuddhistCalendar의 인스턴스를 반환하고, 그 외에는 GregorianCalendar와 인스턴스를 반환한다.
- GregorianCalendar는 Calendar를 상속받아 오늘날 전세계 공통으로 사용하고 있는 그레고리력에 맞게 구현한 것으로 태국을 제외한 나머지 국가에서는 GregorianCalendar를 사용하면 된다.
- 인스턴스를 직접 생성해서 사용하지 않고 메소드를 통해서 인스턴스를 반환받게 하는 이유는 최소한의 변경으로 프로그램이 동작할 수 있도록 하기 위한 것이다.

```
class MyApplication {
        public static void main(String[] args) {
                Calendar cal = new GregorianCalendar(); // 경우에 따라 이 부분을 변경해야 한다.
                ...
        }
}
```

- 상기 코드와 같이 특정 인스턴스를 생성하도록 프로그램이 작성되어 있다면, 다른 종류의 역법(calendar)을 사용하는 국가에서 실행되거나, 새로운 역법이 추가된다면, 즉 다른 종류의 인스턴스를 필요로 하는 경우에 MyApplication을 변경해야 한다.

```
class MyApplication {
        public static void main(String[] args) {
                Calendar cal = Calendar.getInstance();
```

```
                ...
        }
}
```

- getIntance()를 사용하면 구현되는 내용은 달라지겠지만 MyApplication이 변경되지 않아도 된다.
- getInstance()메서드가 static인 이유는 메서드 내의 코드에서 인스턴스 변수를 사용하거나 인스턴스 메서드를 호출하지 않기 때문
- Calenda는 추상 클래스 이므로 객체를 생성할 수 없다, 따라서 객체를 생성하기 위한 static 메서드은 getInstance()가 필요하다.

Date와 Calendar간의 변환

- Calendar가 새로 추가되면서 Date는 대부분의 메서드가 'dreprecated' 되었으므로 잘 사용되지 않는다.
- 그러나 여전히 Date를 필요로 하는 메서드들이 있기 때문에 Calendar를 Date로 또는 그 반대로 변환할 일이 있다.
    - Calendar를 Date로 변환

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

    - Date를 Calendar로 변환

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

day12/date_calendar/CalendarEx1.java

```
package day12.date_calendar;

import java.util.*;

public class CalendarEx1 {
        public static void main(String[] args) {
                // 기본적으로 현재날짜와 시간으로 설정된다.
                Calendar today = Calendar.getInstance();

                System.out.println("이 해의 년도 : " +
today.get(Calendar.YEAR));
                System.out.println("월(0~11, 0:1월) : " +
today.get(Calendar.MONTH));
```

```
                System.out.println("이 해의 몇 째 주 : " +
today.get(Calendar.WEEK_OF_YEAR));
                System.out.println("이 달의 몇 째 주 : " +
today.get(Calendar.WEEK_OF_MONTH));

                // DATE와 DAY_OF_MONTH와는 같다.
                System.out.println("이 달의 몇 일 : "+
today.get(Calendar.DATE));
                System.out.println("이 달의 몇 일 : "+
today.get(Calendar.DAY_OF_MONTH));
                System.out.println("이 해의 몇 일 : "+
today.get(Calendar.DAY_OF_YEAR));
                System.out.println("요일(1~7), 1:일요일  : "+
today.get(Calendar.DAY_OF_WEEK)); // 1. 일요일, 2. 월요일 ... 7.토요일
                System.out.println("이 달의 몇 째 요일 : "+
today.get(Calendar.DAY_OF_WEEK_IN_MONTH));
                System.out.println("오전_오후(0:오전, 1:오후) : "+
today.get(Calendar.AM_PM));
                System.out.println("시간(0~11) : "+
today.get(Calendar.HOUR));
                System.out.println("시간(0~23) : "+
today.get(Calendar.HOUR_OF_DAY));
                System.out.println("분(0~59) : "+
today.get(Calendar.MINUTE));
                System.out.println("초(0~59) : "+
today.get(Calendar.SECOND));
                System.out.println("1000분의 1초(0~999)  : "+
today.get(Calendar.MILLISECOND));

                System.out.println("TimeZone(~12~+12): " +
(today.get(Calendar.ZONE_OFFSET) / (60 * 60 * 1000)));
                System.out.println("이 달의 마지막 날:" +
today.getActualMaximum(Calendar.DATE));
        }
}
```

실행결과

```
이 해의 년도 : 2022
월(0~11, 0:1월) : 4
이 해의 몇 째 주 : 19
이 해의 몇 일 : 125
요일(1~7), 1:일요일  : 5
이 달의 몇 째 요일 : 1
오전_오후(0:오전, 1:오후) : 0
시간(0~11) : 9
시간(0~23) : 9
분(0~59) : 56
초(0~59) : 58
1000분의 1초(0~999)  : 207
```

```
TimeZone(~12~+12): 9
이 달의 마지막 날:31
```

- getInstance(),를 통해서 얻은 인스턴스는 기본적으로 현재 시스템의 날짜와 시간에 대한 정보를 담고 있다.
- 원하는 날짜나 시간으로 설정하려면 **set 메서드 ** 사용
- 원하는 필드의 값을 얻어오려면 **int get(int field)**를 사용
- get(Calendar.MONTH)로 얻어오는 값의 범위는 0~11이다. 즉 0은 1월, 11은 12월을 의미한다.

day12/date_calendar/CalendarEx2.java

```
package day12.date_calendar;

import java.util.Calendar;

public class CalendarEx2 {
	public static void main(String[] args) {
		// 요일은 1부터 시작하기 때문에, DAY_OF_WEEK[0]은 비워두었
다.
		final String[] DAY_OF_WEEK = {"", "일", "월", "화", "수",
"목", "금", "토" };

		Calendar date1 = Calendar.getInstance();
		Calendar date2 = Calendar.getInstance();

		// month의 경우 0부터 시작하기 때문에 8월인 경우 7로 지정해
야 한다.
		// date1.set(2021, Calendar.AUGUST, 15); 와 같이 할 수 도
있다.
		date1.set(2021, 7, 15); // 2021년 8월 15일
		System.out.println("date1은 " + toString(date1) +
DAY_OF_WEEK[date1.get(Calendar.DAY_OF_WEEK)] + "요일 이고, ");
		System.out.println("오늘(date2)은 " + toString(date2) +
DAY_OF_WEEK[date2.get(Calendar.DAY_OF_WEEK)] + "요일 이고, ");

		// 두 날짜 사이의 차이를 얻으려면, getTimeInMillis() 천분
의 일초 단위로 변환해야 한다.
		long difference = (date2.getTimeInMillis() -
date1.getTimeInMillis()) / 1000;
		System.out.println("그 날(date1)부터 지금(date2)까지 " +
difference + "초가 지났습니다.");
		System.out.println("월(day)로 계산하면 " + difference /
(24*60 * 60) + "일 입니다."); // 1일 - 24 * 60 * 60
	}

	public static String toString(Calendar date) {
		return date.get(Calendar.YEAR) + "년 " +
(date.get(Calendar.MONTH) + 1) + "월 " + date.get(Calendar.DATE) + "일";
```

```
        }
}
```

```
실행결과
date1은 2021년 8월 15일일요일 이고,
오늘(date2)은 2022년 5월 5일목요일 이고,
그 날(date1)부터 지금(date2)까지 22723200초가 지났습니다.
월(day)로 계산하면 263일 입니다.
```

- 날짜와 시간을 원하는 값으로 변경하려면 set메서드를 사용

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

- 시간상 전후를 알고 싶을 때는 두 날짜간의 차이가 양수인지 음수인질을 판단하면 된다.
- 또는 간단히 boolean after(Object when)과 boolean before(Object when)을 사용해도 된다.

day12/date_calendar/CalendarEx3.java

```
package day12.date_calendar;

import java.util.*;

public class CalendarEx3 {
        public static void main(String[] args) {
                Calendar date = Calendar.getInstance();
                date.set(2021, 7, 31); // 2021년 8월 31일

                System.out.println(toString(date));
                System.out.println("= 1일 후 =");
                date.add(Calendar.DATE, 1);
                System.out.println(toString(date));

                System.out.println("= 6달 전 =");
                date.add(Calendar.MONTH, -6);
                System.out.println(toString(date));

                System.out.println("= 31일 후(roll) =");
                date.roll(Calendar.DATE, 31);
                System.out.println(toString(date));

                System.out.println("= 31일 후(add) =");
                date.add(Calendar.DATE, 31);
```

```
                System.out.println(toString(date));
        }

        public static String toString(Calendar date) {
                return date.get(Calendar.YEAR) + "년 " +
(date.get(Calendar.MONTH) + 1) + "월 " + date.get(Calendar.DATE) + "일";
        }
}

실행결과
2021년 8월 31일
= 1일 후 =
2021년 9월 1일
= 6달 전 =
2021년 3월 1일
= 31일 후(roll) =
2021년 3월 1일
= 31일 후(add) =
2021년 4월 1일
```

- add(int field, int amount)를 사용하면 지정한 필드의 값을 원하는 만큼 증가 또는 감소 시킬수 있다.
- roll(int field, int amount)도 지정한 필드의 값을 증가 또는 감소시킬 수 있는데, add 메서드와의 차이점은 다른 필드에 영향을 미치지 않는다는 것 예를 들어 add메서드로 날짜 필드(Calendar.MONTH)의 값을 31만큼 증가시켰다면 다음 달로 넘어가므로 월 필드(Calendar.MONTH)의 값도 1이 증가하지만, roll 메서드는 같은 경우에 월 필드의 값은 변하지 않고 일 필드의 값만 바뀐다.

day12/date_calendar/CalendarEx4.java

```
package day12.date_calendar;

import java.util.*;

public class CalendarEx4 {
        public static void main(String[] args) {
                Calendar date = Calendar.getInstance();

                date.set(2021, 0, 31); // 2021년 1월 31일
                System.out.println(toString(date));
                date.roll(Calendar.MONTH, 1);
                System.out.println(toString(date));
        }

        public static String toString(Calendar date) {
                return date.get(Calendar.YEAR) + "년 " +
(date.get(Calendar.MONTH) + 1) + "월 " + date.get(Calendar.DATE) + "일";
        }
}
```

```
실행결과
2021년 1월 31일
2021년 2월 28일
```

java.time 패키지

- Date와 Calendar가 가지고 있는 단점들을 해소하기 위해 JDK1.8부터 'java.time 패키지'가 추가되었다.
- 이 패키지는 다음과 같이 4개의 하위 패키지를 가지고 있다

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)을 위한 클래스들을 제공
java.time.zone	시간대(time-zone)와 관련된 클래스들을 제공

- 상기 패키지들에 속한 클래스들의 가장 큰 특징은 String클래스 처럼 불변(immutable)이라는 것이다.
- 그래서 날짜나 시간을 변경하는 메서드들은 기존의 객체를 변경하는 대신 항상 변경된 새로운 객체를 반환한다.
- 기존 Calendar 클래스는 변경이 가능하므로, 멀티 쓰레드 환경에서 안전하지 못하다.

java.time 패키지의 핵심 클래스

- 날짜와 시간을 하나로 표현하는 Calendar 클래스와 달리, java.time 패키지에서는 날짜와 시간을 별도 클래스로 분리해 놓았다.
- LocalTime 클래스 : 시간을 표현
- LocalDate 클래스 : 날짜를 표현
- LocalDateTime 클래스 : 날짜와 시간을 모두 표현

```
LocalDate(날짜) + LocalTime(시간) -> LocalDateTime(날짜 & 시간)
```

- ZonedDateTime 클래스 : LocalDateTime + 시간대

- Instant 클래스 : 날짜와 시간을 초 단위(정확히는 나노초)로 표현한다.
  날짜와 시간을 초단위로 표현한 값을 타임스탬프(time-stamp)라고 부르는데, 이 값은 날짜와 시간을 하나의 정수로 표현할 수 있으므로 날짜와 시간의 차이를 계산하거나 순서비를 비교하는데 유리하다.
- 기타 : Year, YearMonth, MonthDay 클래스

Period와 Duration

- Period 클래스 - 두 날짜 간의 차이를 표현하기 위한 것
- Double 클래스 - 시간의 차이를 표현하기 위한 것

```
날짜 - 날짜 = Period
시간 - 시간 = Duration
```

객체 생성하기 - now(), of()

- now() : 현재 날짜와 시간을 저장하는 객체를 생성한다.

```
LocalDate date = LocalDate.now(); // 현재 날짜
LocalTime time = LocalTime.now(); // 현재 시간
LocalDateTime dateTime = LocalDateTime.now(); // 현재 날짜와 시간
ZonedDateTime dateTimeInKr = ZonedDateTime.now(); // 현재 날짜와 시간 +
시간대
```

- of() : 지정된 날짜, 시간, 시간대의 객체를 생성한다.

```
LocalDate date = LocalDate.of(2021,11,23); // 2021년 11월 23일
LocalTime time = LocalTime.of(23,59,59); // 23시 59분 59초

LocalDateTime dateTime = LocalDateTime.of(date, time);
ZonedDateTime zDateTime = ZonedDateTime.of(dateTime,
ZoneId.of("Asia/Seoul"));
```

Temporal과 TemporalAmount

- LocalDate, LocalTime, LocalDateTime, ZonedDateTime등 날짜와 시간을 표현하기 위한 클래스들은 모두 **Temporal, TemporalAccessor, TemporalAdjuster 인터페이스**를 구현하였다.
- Duration과 Period는 **TemporalAmount 인터페이스**를 구현하였다.

```
Temporal, TemporalAccessor, TemporalAdjuster를 구현한 클래스
        - LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Instant 등
```

```
TemporalAmount를 구현한 클래스
        - Period, Duration
```

TemporalUnit과 TemporalField

- TemportalUnit 인터페이스 : 날짜와 시간의 단위를 정의해 놓은 것, 이 인터페이스를 구현한 것이 **열거형 ChronoUnit**이다.
- TemporalFIeld 인터페이스 : 년, 월, 일 등의 날짜와 시간의 필드를 정의해 놓은 것, 이 인터페이스를 구현한 것이 **열거형 ChronoField**이다.

```
LocalTime now = LocalTime.now(); // 현재 시간
int minute = now.getMinute(); // 현재 시간에서 분(minute)만 추출
int minute = now.get(ChronoField.MINUTE_OF_HOUR); // now.getMinute()와 동
일
```

- 날짜와 시간에서 틀정 필드의 값만 얻을 때는 get()이나 get으로 시작하는 이름의 메서드를 이용한다.
- 특정 날자의 시간에서 지정된 단위의 값을 더하거나 뺄 때는 plus() 또는 minus()에 값과 함께 열거형 ChonoUnit을 사용한다.

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

```
LocalDate today = LocalDate.now(); // 오늘
LocalDate tomorrow = today.plus(1, ChronoUnit.DAYS); // 오늘에 1일을 더한
다.
LocalDate tomorrow = today.plusDays(1); // today.plus(1,
ChronoUnit.DAYS)와 동일
```

- isSupported() : TemporalField나 TemporalUnit을 사용할 수 있는지 확인하는 메서드

```
boolean isSupported(TemporalUnit unit)  // Temporal에 정의
boolean isSupported(TemporalField field) // TemporalAccessor에 정의
```

LocalDate와 LocalTime

- LocalDate와 LocalTime은 java.time 패키지의 가장 기본이 되는 클래스이다.
- 나머지 클래스 들은 이 두 클래스의 확장이다.
- now() : 현재의 날짜와 시간을 LocalDate와 LocalTime으로 각각 반환한다.

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

LocalDate birthDate = LocalDate.of(2021, 12, 31); // 2021년 12월 31일
LocalTime birthTime = LocalTime.of(23, 59, 59); // 23시 59분 59초
```

- of() : 지정된 날짜와 시간으로 LocalDate와 LocalTime 객체를 생성한다.

```
static LocalDate of(int year, Month month, int dayOfMonth)
static LocalDate of(int year, int month, int dayOfMonth)

static LocalTime of(int hour, int min)
static LocalTime of(int hour, int min, int sec)
static LocalTime of(int hour, int min, int sec, int nanoOfSecond)
```

- 일 단위 또는 초 단위로 시간을 지정할 수 있다.

```
// 1999년의 365번째 날
LocalDate birthDate = LocalDate.ofYearDay(1999, 365); // 1999년 12월 31일

// 오늘 0시 0분 0초 부터 86399초(하루는 86400초)가 지난 시간
LocalTime birthTime = LocalTime.ofSecondDay(86399); // 23시 59분 59초
```

- parse() : 문자열을 날짜와 시간으로 변환할 수 있다.

```
LocalDate birthDate = LocalDate.parse("1999-12-31");
LocalTime birthTime = LocalTime.parse("23:59:59");
```

특정 필드의 값 가져오기 - get(), getXXX()

- 주의할 점 : Calendar와 달리 월(month)의 범위가 1~12이고, 요일은 월요일이 1, 화요일이 2, ..., 일요일은 7이라는 것이다.

- LocalDate

메서드	설명(1999-12-31 23:59:59)
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)

메서드	설명(1999-12-31 23:59:59)
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()	윤년 여부 확인(false)

- LocalTime

메서드	설명(1999-12-31 23:59:59)
int getHour()	시(23)
int getMinute()	분(59)
int getSecond()	초(59)
int getNano()	나노초(0)

- get(), getLong() : 원하는 필드를 직접 지정할 수 있다
  대부분의 필드는 int타입의 범위에 속하지만, 몇몇 필드는 int타입의 범위를 넘어서는데, 그때에는 get()대신 getLong()을 사용한다. 하기 표는 ChronoField에 정의된 상수 목록이며 getLong을 사용해야 하는 경우는 *표 표기 하였다.

TemporalField(ChronoField)	설명
ERA	시대
YEAR_OF_ERA, YEAR	년
MONTH_OF_YEAR	월
DAY_OF_WEEK	요일(1:월요일, 2:화요일, ... 7:일요일)
DAY_OF_MONTH	일
AMPM_OF_DAY	오전/오후
HOUR_OF_DAY	시간(0~23)
CLOCK_HOUR_OF_DAY	시간(1~24)
HOUR_OF_AMPM	시간(0~11)
CLOCK_HOUR_OF_AMPM	시간(1~12)
MINUTE_OF_HOUR	분
SECOND_OF_MINUTE	초

TemporalField(ChronoField)	설명
MILLI_OF_SECOND	천분의 일초
MICRO_OF_SECOND *	백만분의 일초
NANO_OF_SECOND *	10억분의 일초
DAY_OF_YEAR	그 해의 몇번째날
EPOCH_DAY *	EPOCH(1970.1.1)부터 몇번째 날
MINUTE_OF_DAY	그 날의 몇 번째 분(시간을 분으로 환산)
SECOND_OF_DAY	그 날의 몇 번째 초(시간을 초로 환산)
MILL_OF_DAY	그날의 몇번째 몇 번째 밀리초
MICRO_OF_DAY *	그날의 몇 번째 마이크로 초
NANO_OF_DAY *	그날의 몇 번째 나노초
ALIGNED_WEEK_OF_MONTH	그 달의 n번째 주
ALIGNED_WEEK_OF_YEAR	그 해의 n번째 주
ALIGNED_DAY_OF_WEEK_IN_MONTH	요일(그 달의 1일을 월요일로 간주하여 계산)
ALIGNED_DAY_OF_WEEK_IN_YEAR	요일(그 해의 1월 1일을 월요일로 간주하여 계산)
INSTANT_SECONDS	년월일을 초단위로 환산(1970-01-01 00:00:00 UTC를 0초로 계산) Instant에만 사용가능
OFFSET_SECONDS	UTC와 시차. ZoneOffset에만 사용가능
PROLEPTIC_MONTH	년월을 월단위로 환산(2015년11월 = 2015*12+11)

- 상기 목록은 ChonoField에 정의된 모든 상수를 나열하였으나, 사용할 수 있는 필드는 클래스마다 다르다(예 : LocalDate는 날짜를 표현하기 위한 것이므로, MINUTE_OF_HOUR와 같이 시간에 관련된 필드는 사용할 수 없다)
- 해당 클래스가 지원하지 않는 필드를 사용하면 UnsupportedTemporalTypeException이 발생한다.
- range() : 특정 필드가 가질 수 있는 값의 범위를 조회

```
System.out.println(ChronoField.CLOCK_HOUR_OF_DAY.range()); // 1 - 24
System.out.println(ChronoField.HOUR_OF_DAY.range()); // 0 - 23
```

필드의 값 변경하기 - with(), plus(), minus()

with()

- 날짜와 시간에서 특정 필드 값을 변경하려면, 하기와 같이 with로 시작하는 메서드를 사용한다.

```
LocalDate withYear(int year)
LocalDate withMonth(int month)
LocalDate withDayOfMonth(int dayOfMonth)
LocalDate withDayOfYear(int dayOfYear)

LocalTime withHour(int hour)
LocalTime withMinute(int minute)
LocalTime withSecond(int second)
LocalTime withNano(int nanoSecond)
```

- with()를 사용하면 원하는 필드를 직접 지정할 수 있다.

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

```
date = date.withYear(2000); 년도를 2000년으로 변경
time = time.withHour(12); // 시간을 12시로 변경
```

plus(), minus()

- 특정 필드에 값을 더하거나 뺄때

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

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

- plus()로 만든 메서드

```
LocalDate plusYears(long yearsToAdd)
LocalDate plusMonths(long monthToAdd)
LocalDate plusDays(long daysToAdd)
LocalDate plusWeeks(long weeksToAdd)
```

```
LocalTime plusHours(long hoursToAdd)
LocalTime plusMinutes(long minutesToAdd)
LocalTime plusSeconds(long secondsToAdd)
LocalTime plusNanos(long nanoToAdd)
```

LocalTime의 truncatedTo()

- 지정된 것보다 작은 단위의 필드를 0으로 만든다.

```
LocalTime time = LocalTime.of(12, 34, 56); // 12시 34분 56초
time = time.truncatedTo(ChronoUnit.HOURS); // 시(hours)보다 작은 단위를 0으로 만든다
System.out.println(time); // 12:00
```

- LocalDate에 truncatedTo()는 없다. LocalDate의 필드인 년, 월, 일은 0이 될수 없다.

ChronoUnit에 정의된 상수 목록

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
MICROS	백만분의 일초

TemporalUnit(ChronoUnit)	설명
NANOS	10억분의 일초

날짜와 시간의 비교 - isAfter(), isBefore(), isEqual()

- LocalDate와 LocalTime도 compareTo()가 적절하게 재정의 되어 있어, 하기와 같이 compareTo()로 비교할 수 있다.

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

- 그러나 보다 편하하게 비교할 수 있는 메서드들이 추가로 제공된다.

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

day12/time/NewTimeEx1.java

```
package day12.time;

import java.time.*;
import java.time.temporal.*;

public class NewTimeEx1 {
	public static void main(String[] args) {
		LocalDate today = LocalDate.now(); // 오늘의 날짜
		LocalTime now = LocalTime.now(); // 현재의 시간

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

		System.out.println("today=" + today);
		System.out.println("now=" + now);
		System.out.println("birthDate=" + birthDate);  // 1999-12-31
		System.out.println("birthTime=" + birthTime); // 23:59:59

		System.out.println(birthDate.withYear(2000)); // 2000-12-31
		System.out.println(birthDate.plusDays(1)); // 2000-01-01
		System.out.println(birthDate.plus(1, ChronoUnit.DAYS)); // 2000-01-01
```

```
                // 23:59:59 -> 23:00

System.out.println(birthTime.truncatedTo(ChronoUnit.HOURS));

                // 특정 ChronoField의 범위를 알아내는 방법

System.out.println(ChronoField.CLOCK_HOUR_OF_DAY.range()); // 1-24
                System.out.println(ChronoField.HOUR_OF_DAY.range()); //
0-23
        }
}
```

실행결과

```
today=2022-05-05
now=18:51:40.110864700
birthDate=1999-12-31
birthTime=23:59:59
2000-12-31
2000-01-01
2000-01-01
23:00
1 - 24
0 - 23
```

Instant

- Instant는 에포크 타입(EPOCH TIME, 1970-01-01 00:00:00 UTC)부터 경과된 시간을 나노초 단위로 표현한다.
- Instant를 생성할 때는 위와 같이 now()와 ofEpochSecond()를 사용한다.

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

- 필드에 저장된 값을 가져올때는 하기와 같이 한다.

```
long epochSec = now.getEpochSecond();
int nano = now.getNano();
```

- 타임스탬프(timestamp)처럼 밀리초 단위의 EPOCH TIME을 필요로 하는 경우

```
long toEpochMilli()
```

- Instant는 항상 UTC(+00:00)를 기준으로 하기 때문에 LocalTime과 차이가 있을 수 있다.
  (예 : 한국은 시간대가 +09:00이므로 Instant와 LocalTime간에는 9시간의 차이가 있다.)
  > UTC는 'Coordincated Universal Time'의 약어로 '세계 협정시'이라고 하며, 1972년 1월 1일부터 시행된 국제 표준시이다. 이전에 사용되던 GMT(Greenwich Mean Time)와 UTC는 거의 같지만, UTC가 좀 더 정확하다.

Instant와 Date간의 변환

Instant는 기존의 java.util.Date를 대체하기 위한 것이며, JDK1.8부터 Date에 Instant로 변환할 수 있는 새로운 메서드가 추가되었다.

```
static Date from(Instant instant)  // Instant -> Date
Instant toInstant()  // Date -> Instant
```

LocalDateTime과 ZonedDateTime

- LocalDate에 LocalTime을 합쳐 놓은 것이 LocalDateTime이다.
- LocalDateTime에 시간대(time zone)을 추가한 것이 ZonedDateTime이다.

```
LocalDate + LocalTime -> LocalDateTime
LocalDateTime + 시간대 -> ZonedDateTime
```

LocalDate와 LocalTime으로 LocalDateTime만들기

- LocalDate와 LocalTime을 합쳐서 만들기

```
LocalDate date = LocalDate.of(2021, 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 dt3 = date.atTime(12, 34, 56);
LocalDateTime dt4 = time.atDate(LocalDate.of(2021, 12, 31));
LocalDateTime dt6 = date.atStartOfDay(); // dt6 = date.atTime(0, 0, 0);
```

- of() - 날짜와 시간을 직접 지정, now() - 현재 날짜와 시간

```
// 2021년 12월 31일 12시 34분 56초
LocalDateTime dateTime = LocalDateTime.of(2021, 12, 31, 12, 34, 56);
```

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

LocalDateTime의 변환

```
LocalDateTime dt = LocalDateTime.of(2021, 12, 31, 12, 34, 56);
LocalDateTime date = dt.toLocalDate(); // LocalDateTime -> LocalDate
LocalTime time = dt.toLocalTime(); // LocalDateTime -> LocalTime
```

LocalDateTime으로 ZonedDateTime 만들기

- LocalDateTIme에 시간대(time-zone)을 추가하면, ZonedDateTime이 된다. 기존에는 TimeZone클래스로 시간대를 다뤘지만 새로운 시간 패키지에서는 ZoneId라는 클래스를 사용한다.
- ZoneId는 일광 절약시간(DST, Daylight Saving Time)을 자동적으로 처리해 주므로 편리하다.
- LocalDateTime에 atZone()으로 시간대 정보를 추가하면, ZonedDateTime을 얻을 수 있다.

> 사용가능한 ZoneId의 목록은 ZoneId.getAvailableZoneIds()로 얻을 수 있다.

```
ZoneId zid = ZoneId.of("Asia/Seoul");
ZonedDateTime zdt = dateTime.atZone(zid);
```

- LocalDate의 asStartOfDate() 메서드에 매개변수로 ZoneId를 지정하는 방법

```
ZoneId zid = ZoneId.of("Asia/Seoul");
ZonedDateTime zdt = LocalDate.now().atStartOfDay(zid);
```

- 특정 시간대의 시간을 알고 싶을 경우

```
예: 뉴욕의 현재 시간을 알고 싶은 경우

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

- 상기 코드에서 now() 대신 of를 사용하면 날짜와 시간을 지정할 수 있다.
```

ZonedOffset

UTC로 부터 얼마만큼 떨어져 있는지를 ZoneOffset으로 표현한다.

```
ZoneOffset krOffset = ZonedDateTime.now().getOffset();
ZoneOffset krOffset = ZoneOffset.of("+9");
int krOffsetInSec = krOffset.get(ChronoField.OFFSET_SECONDS); // 32400초
```

OffsetDateTime

- ZonedDateTime은 ZonedId로 구역을 표현하는데, ZoneId가 아닌 ZoneOffset을 사용하는 것이 OffsetDateTime이다.
- ZoneId는 일광절약시간처럼 시간대와 관련된 규칙들을 포함하고 있는데, ZoneOffset은 단지 시간대를 시간의 차이로만 구분한다.
- 서로 다른 시간대에 존재하는 컴퓨터간의 통신에는 OffsetDateTime이 필요하다.

```
ZoneId zid = ZoneId.of("Asia/Seoul");
ZoneOffset krOffset = ZoneOffset.of("+9");

ZonedDateTime zdt = ZonedDateTime.of(date, time, zid);
OffsetDateTime odt = OffsetDateTime.of(date, time, krOffset);

// ZonedDateTime -> OffsetDateTime
OffsetDateTime odt = zdt.toOffsetDateTime();
```

ZonedDateTime의 변환

ZonedDateTime도 LocalDateTime처럼 날짜와 시간에 관련된 다른 클래스로 변환하는 메서드를 가지고 있다.

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

// ZonedDateTime -> GregorianCalendar
GregorianCalendar from(ZonedDateTime zdt)

// GregorianCalendar -> ZonedDateTime
ZonedDateTime toZonedDateTime()
```

day12/time/NewTimeEx2.java

```
package day12.time;

import java.time.*;
```

```
public class NewTimeEx2 {
        public static void main(String[] args) {
                LocalDate date = LocalDate.of(2021,  12, 31); // 2021년
12월 31일
                LocalTime time = LocalTime.of(12, 34, 56); // 12시 34분
56초

                // 2021년 12월 31일 12시 34분 56초
                LocalDateTime dt = LocalDateTime.of(date,  time);

                ZoneId zid = ZoneId.of("Asia/Seoul");
                ZonedDateTime zdt = dt.atZone(zid);
                //String strZid = zdt.getZone().getId();
                //System.out.println(strZid); // Asia/Seoul

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

                // ZonedDateTime -> OffsetDateTime
                OffsetDateTime odt = zdt.toOffsetDateTime();

                System.out.println(dt);
                System.out.println(zid);
                System.out.println(zdt);
                System.out.println(seoulTime);
                System.out.println(nyTime);
                System.out.println(odt);
        }
}
```

실행결과

```
2021-12-31T12:34:56
Asia/Seoul
2021-12-31T12:34:56+09:00[Asia/Seoul]
2022-05-05T19:40:28.799906600+09:00[Asia/Seoul]
2022-05-05T06:40:28.802895400-04:00[America/New_York]
2021-12-31T12:34:56+09:00
```

TemporalAdjusters

자주 쓰일만한 날짜 계산들을 대신 해주는 메서드를 정의해 놓은 것이 TemporalAdjusters클래스이다.

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

TemporalAdjusters의 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 마 지막 ?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

TemporalAdjuster 직접 구현하기

- 보통은 TemporalAdjusters에 정의된 메서드로 충분하지만, 필요하다면 자주 사용되는 날짜계산을 새주는 메서드를 직접 만들 수도 있다.
- TemporalAdjuster인터페이스는 추상 메서드 하나만 정의되어 있으며, 이 메서드만 구현하면 된다.

```
@FunctionalInterface
public interface TemporalAdjuster {
        Temporal adjusterInto(Temporal temporal);
}
```

- 실제로 구현해야 하는 것은 adjustInfo()지만, TemporalAdjuster와 같이 사용해야 하는 메서드는 with()이다.

- adjustInfo()는 내부적으로 사용할 의도로 작성된 것이므로 with()를 사용
- 날짜와 시간에 관련된 대부분의 클래스는 Temporal 인터페이스를 구현하였으므로 adjustInfo()의 매개변수가 될 수 있다.

특정 날짜로부터 2일 후의 날짜 계산하는 예

```
class DayAfterTomorrow implements TemporalAdjuster {
        @Override
        public Temporal adjustInfo(Temporal temporal) {
                return temporal.plus(2, ChronoUnit.DAYS); // 2일을 더한
다.
        }
}
```

day12/time/NewTimeEx3.java

```
package day12.time;

import java.time.*;
import java.time.temporal.*;
import static java.time.DayOfWeek.*;
import static java.time.temporal.TemporalAdjusters.*;

class DayAfterTomorrow implements TemporalAdjuster {

        @Override
        public Temporal adjustInto(Temporal temporal) {
                return temporal.plus(2, ChronoUnit.DAYS);
        }
}

public class NewTimeEx3 {
        public static void main(String[] args) {
                LocalDate today = LocalDate.now();
                LocalDate date = today.with(new DayAfterTomorrow());

                p(today);
                p(date);
                p(today.with(firstDayOfNextMonth())); // .다음 달의 첫 날
                p(today.with(firstDayOfMonth())); // 이 달의 첫 날
                p(today.with(lastDayOfMonth())); // 이 달의 마지막 날
                p(today.with(firstInMonth(TUESDAY))); // 이 달의 첫번째
화요일
                p(today.with(lastInMonth(TUESDAY))); // 이 달의 마지막 화
요일
                p(today.with(previous(TUESDAY))); // 지난 주 화요일
                p(today.with(previousOrSame(TUESDAY))); // 지난 주 화요일
(오늘 포함)
                p(today.with(next(TUESDAY))); // 다음 주 화요일
                p(today.with(nextOrSame(TUESDAY))); // 다음 주 화요일(오
```

```
늘 포함)
                p(today.with(dayOfWeekInMonth(4, TUESDAY))); // 이달의 4
번째 화요일
        }

        public static void p(Object obj) {
                System.out.println(obj);
        }
}
```

```
실행결과
2022-05-05
2022-05-07
2022-06-01
2022-05-01
2022-05-31
2022-05-03
2022-05-31
2022-05-03
2022-05-03
2022-05-10
2022-05-10
2022-05-24
```

Period와 Duration

period는 날짜의 차이, Duration은 시간의 차이를 계산

```
날짜 - 날짜 = Period
시간 - 시간 = Duration
```

between()

두 날짜의 차이를 나타내는 Period 객체를 반환

Period - 날짜의 차이

```
LocalDate date1 = LocalDate.of(2020, 1, 1);
LocalDate date2 = LocalDate.of(2021, 12, 31);

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

date1이 date2보다 날짜 상으로 이전이면 양수로, 이후면 음수로 Period에 저장된다.

Duration - 시간의 차이

```
LocalTime time1 = LocalTime.of(00, 00, 00);
LocalTime time2 = LocalTime.of(12, 34, 56);  // 12시 34분 56초

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

get() - Period, Duration에서 특정 필드의 값을 얻을 때

```
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()
long nano = du.get(ChronoUnit.NANOS); // long getNano()
```

- Duration에서는 ChonoUnit.SECONDS, ChonoUnit.NANOS 또는 long getSeconds(), long getNanos() 밖에 사용할 수 없다.
- 만약 Durationdptj 시, 분, 초 형태로 변환하는 방법은 다음과 같다.

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

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

between()과 until()

- until()과 between()은 거의 같은 일을 한다.
- between()은 static 메서드이고, until()은 인스턴스 메서드라는 차이가 있다.

```
Period pe = Period.between(today, myBirthDay);
Period pe = today.until(myBirthDay);
long dday = today.until(myBirthDay, ChronoUnit.DAYS);
long sec = LocalTime.now().until(endTime, ChronoUnit.SECONDS);
```

of(), with()

of() - 지정한 값으로 Period 또는 Duration의 인스턴스 생성

- Period에는 of(), ofYears(), ofMonths(), ofWeeks(), ofDays()가 있다.
- Duration에는 of(), ofDays(), ofHours(), ofMinutes(), ofSeconds()가 있다.

```
Period pe = Period.of(1, 12, 31); // 1년 12개월 31일
Duration du = Duration.of(60, ChonoUnit.SECONDS); // 60초
Duration du = Duration.ofSecond(60); // Duration.of(60,
ChonoUnit.SECONDS)과 동일
```

with() - 특정 필드의 값을 변경

```
pe = pe.withYears(2); // 1년에서 2년으로 변경. withMonths(), withDays()
du = du.withSeconds(120); // 60초에서 120초로 변경. withNanos()
```

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

Period와 Duration을 다른 단위의 값으로 변환하는데 사용된다.

Period와 Duration의 변환 메서드

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 반환해서 반환
	long toNanos()	나노초 단위로 반환해서 반환

LocalDate의 toEpochDay()

- Epoch Day인 '1970-01-01'부터 날짜를 세어서 반환
- 이 메서드를 이용하면 Period를 사용하지 않고도 두 날짜간의 일수를 계산할 수 있다.

```
LocalDate date1 = LocalDate.of(2021, 11, 28);
LocalDate date2 = LocalDate.of(2021, 11, 29);

long period = date2.toEpochDay() - date1.toEpochDay();
```

LocalTime의 toSecondOfDay(), toNanoOfDay()

Duration을 사용하지 않고도 뺄샘으로 시간 차이를 계산할 수 있다.

```
int toSecondOfDay()
long toNanoOfDay()
```

day12/time/NewTimeEx4.java

```
package day12.time;

import java.time.*;
import java.time.temporal.*;

public class NewTimeEx4 {
	public static void main(String[] args) {
		LocalDate date1 = LocalDate.of(2020, 1, 1);
		LocalDate date2 = LocalDate.of(2021, 12, 31);

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

		System.out.println("date1=" + date1);
		System.out.println("date2=" + date2);
		System.out.println("pe=" + pe);

		System.out.println("YEAR=" + pe.get(ChronoUnit.YEARS));
		System.out.println("MONTH=" +
pe.get(ChronoUnit.MONTHS));
		System.out.println("DAY=" + pe.get(ChronoUnit.DAYS));

		LocalTime time1 = LocalTime.of(0, 0, 0);
		LocalTime time2 = LocalTime.of(12, 34, 56); // 12시간 23
분 56초

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

		System.out.println("time1=" + time1);
		System.out.println("time2=" + time2);
		System.out.println("du=" + du);

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

		System.out.println("HOUR=" + tmpTime.getHour());
		System.out.println("MINUTE=" + tmpTime.getMinute());
		System.out.println("SECOND=" + tmpTime.getSecond());
		System.out.println("NANO=" + tmpTime.getNano());
	}
}

실행결과
date1=2020-01-01
date2=2021-12-31
pe=P1Y11M30D
```

```
YEAR=1
MONTH=11
DAY=30
time1=00:00
time2=12:34:56
du=PT12H34M56S
HOUR=12
MINUTE=34
SECOND=56
NANO=0
```

파싱과 포맷

- 날짜와 시간을 원하는 형식으로 출력하고 해석(파싱, parsing)하는 방법
- 형식화(formatting)와 관련된 클래스들은 java.time.format 패키지에 들어가 있다.
- DateTimeFormatter

출력 형식 정의하기

DateTimeFormatter의 ofPattern()으로 원하는 출력형식을 작성할 수 있다.

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

DateTimeFormatter와 패턴에 사용되는 기호

기호	의미	보기
G	연대(BC,AD)	서기 또는 AD
y 또는 u	년도	2021
M 또는 L	월(1~~12 또는 1월~~12월)	11
Q 또는 q	분기(quarter)	4
w	년의 몇 번째 주(1~53)	48
W	월의 몇 번째 주(1~5)	4
D	년의 몇 번째 일(1~366)	332
d	월의 몇 번째 일(1~31)	28
F	월의 몇 번째 요일(1~5)	4
E 또는 e	요일	토 또는 7
a	오전/오후(AM, PM)	오후

기호	의미	보기
H	시간(0~23)	22
k	시간(1~24)	22
K	시간(0~11)	10
h	시간(1~12)	10
m	분(0~59)	12
s	초(0~59)	35
S	천분의 일초(0~999)	7
A	천분의 일초(그 날의 0시 0분 0초 부터의 시간)	80263808
n	나노초	475000000
N	나노초(그 날의 0시 0분 0초 부터의 시간)	
V	시간대 ID(VV)	Asia/Seoul
z	시간대(time-zone) 이름	KST
O	지역화된 zone-offset	GMT+9
Z	zone-offset	+0900
X 또는 x	zone-offset(Z는 +00:00를 의미)	+09
'	escape문자(특수문자를 표현하는데 사용)	없음

day12/time/format/DateFormatterEx1.java

```
package day12.time.format;

import java.time.*;
import java.time.format.*;

public class DateFormatterEx1 {
	public static void main(String[] args) {
		ZonedDateTime zdateTime = ZonedDateTime.now();

		String[] patternArr = {
			"yyyy-MM-dd HH:mm:ss",
			"''yy년 MMM월 dd일 E요일",
			"yyyy-MM-dd HH:mm:ss.SSS Z VV",
			"yyyy-MM-dd hh:mm:ss s",
			"오늘은 올 해의 D번째 날입니다.",
			"오늘은 이 달의 d번째 날입니다.",
			"오늘은 이 달의 w번째 주입니다.",
```

```
                        "오늘은 이 달의 W번째 주입니다.",
                        "오늘은 이달의 W번째 E요일입니다."
                };

                for(String p : patternArr) {
                        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern(p);
                        System.out.println(zdateTime.format(formatter));
                }
        }
}
```

```
실행결과
2022-05-05 21:29:31
'22년 5월월 05일 목요일
2022-05-05 21:29:31.962 +0900 Asia/Seoul
2022-05-05 09:29:31 31
오늘은 올 해의 125번째 날입니다.
오늘은 이 달의 5번째 날입니다.
오늘은 이 달의 19번째 주입니다.
오늘은 이 달의 1번째 주입니다.
오늘은 이달의 1번째 목요일입니다.
```

문자열을 날짜와 시간으로 파싱하기

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

- 자주 사용되는 기본적인 형식의 문자열은 ISO_LOCAL_DATE와 같은 형식화 상수를 사용하지 않고도 파싱이 가능하다.

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

- ofPattern()을 이용해서 파싱을 할 수 있다.

```
DateTimeFormatter pattern = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");
LocalDateTime endOfYear = LocalDateTime.parse("2021-12-31 23:59:59",
pattern);
```

day12/time/format/DateFormatterEx2.java

```
package day12.time.format;

import java.time.*;
import java.time.format.*;

public class DateFormatterEx2 {
        public static void main(String[] args) {
                LocalDate newYear = LocalDate.parse("2016-01-01",
DateTimeFormatter.ISO_LOCAL_DATE);

                LocalDate date = LocalDate.parse("2001-01-01");
                LocalTime time = LocalTime.parse("23:59:59");
                LocalDateTime dateTime = LocalDateTime.parse("2001-01-
01T23:59:59");

                DateTimeFormatter pattern =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                LocalDateTime endOfYear = LocalDateTime.parse("2021-12-
31 23:59:59", pattern);

                System.out.println(newYear);
                System.out.println(date);
                System.out.println(time);
                System.out.println(dateTime);
                System.out.println(endOfYear);
        }
}
```

실행결과

```
2016-01-01
2001-01-01
23:59:59
2001-01-01T23:59:59
2021-12-31T23:59:59
```