Java

제 13 강

인터페이스 & 객체지향개념 II-3

날짜와 시간 & 형식화

1. 상속
2. 오버라이딩
3. package와 import

→ 객체지향개념 II-1

4. 제어자
5. 다형성

→ 객체지향개념 II-2

6. 추상클래스
7. 인터페이스

→ 객체지향개념 II-3

7. 인터페이스(interface)

7. 인터페이스(interface)

7.1 인터페이스(interface)란?

- 일종의 추상클래스. 추상클래스(미완성 설계도)보다 추상화 정도가 높다.
- 실제 구현된 것이 전혀 없는 기본 설계도.(알맹이 없는 껍데기)
- 추상메서드와 상수만을 멤버로 가질 수 있다.
- 인스턴스를 생성할 수 없고, 클래스 작성에 도움을 줄 목적으로 사용된다.
- 미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는 데 사용된다.

7.2 인터페이스의 작성

- 'class'대신 'interface'를 사용한다는 것 외에는 클래스 작성과 동일하다.

```
interface 인터페이스이름 {
    public static final 타입 상수이름 = 값;
    public abstract 메서드이름(매개변수목록);
}
```

- 하지만, 구성요소(멤버)는 추상메서드와 상수만 가능하다.

- 모든 멤버변수는 public static final 이어야 하며, 이를 생략할 수 있다.
- 모든 메서드는 public abstract 이어야 하며, 이를 생략할 수 있다.

```
interface PlayingCard {
    public static final int SPADE = 4;
    final int DIAMOND = 3;      // public static final int DIAMOND = 3;
    static int HEART = 2;       // public static final int HEART = 2;
    int CLOVER = 1;             // public static final int CLOVER = 1;

    public abstract String getCardNumber();
    String getCardKind();  // public abstract String getCardKind();
}
```

7.3 인터페이스의 상속

- 인터페이스도 클래스처럼 상속이 가능하다.(클래스와 달리 다중상속 허용)

```
interface Movable {
     /** 지정된 위치(x, y)로 이동하는 기능의 메서드 */
     void move(int x, int y);
}

interface Attackable {
     /** 지정된 대상(u)을 공격하는 기능의 메서드 */
     void attack(Unit u);
}

interface Fightable extends Movable, Attackable { }
```

- 인터페이스는 Object클래스와 같은 최고 조상이 없다.

7.4 인터페이스의 구현

- 인터페이스를 구현하는 것은 클래스를 상속받는 것과 같다.
  다만, 'extends' 대신 'implements'를 사용한다.

```
class 클래스이름 implements 인터페이스이름 {
     // 인터페이스에 정의된 추상메서드를 구현해야한다.
}
```

- 인터페이스에 정의된 추상메서드를 완성해야 한다.

```
class Fighter implements Fightable {
     public void move() { /* 내용 생략*/ }
     public void attack() { /* 내용 생략*/ }
}

interface Fightable {
     void move(int x, int y);
     void attack(Unit u);
}
```

```
abstract class Fighter implements Fightable {
     public void move() { /* 내용 생략*/ }
}
```

- 상속과 구현이 동시에 가능하다.

```
class Fighter extends Unit implements Fightable {
     public void move(int x, int y) { /* 내용 생략 */}
     public void attack(Unit u) { /* 내용 생략 */}
}
```

7.5 인터페이스를 이용한 다형성

- 인터페이스 타입의 변수로 인터페이스를 구현한 클래스의 인스턴스를 참조할 수 있다.

```
class Fighter extends Unit implements Fightable {
     public void move(int x, int y)  { /* 내용 생략 */ }
     public void attack(Fightable f) { /* 내용 생략 */ }
}
```

```
Fighter   f = new Fighter();
Fightable f = new Fighter();
```

- 인터페이스를 메서드의 매개변수 타입으로 지정할 수 있다.

```
void attack(Fightable f) {   // Fightable인터페이스를 구현한 클래스의 인스턴스를
    //...                    // 매개변수로 받는 메서드
}
```

- 인터페이스를 메서드의 리턴타입으로 지정할 수 있다.

```
Fightable method() { // Fightable인터페이스를 구현한 클래스의 인스턴스를 반환
    // ...
    return new Fighter();
}
```

7.6 인터페이스의 장점

1. 개발시간을 단축시킬 수 있다.

일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능하다. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문이다.

그리고 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행할 수 있다.

2. 표준화가 가능하다.

프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능하다.

3. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.

서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다.

4. 독립적인 프로그래밍이 가능하다.

인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제구현에 독립적인 프로그램을 작성하는 것이 가능하다.

클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능하다.

7.6 인터페이스의 장점 – 예제

```
interface Repairable {}

class GroundUnit extends Unit {
    GroundUnit(int hp) {
        super(hp);
    }
}

class AirUnit extends Unit {
    AirUnit(int hp) {
        super(hp);
    }
}

class Unit {
    int hitPoint;
    final int MAX_HP;
    Unit(int hp) {
        MAX_HP = hp;
    }
}
```

```
class Tank extends GroundUnit implements Repairable {
    Tank() {
        super(150);      // Tank의 HP는 150이다.
        hitPoint = MAX_HP;
    }

    public String toS
        return "Tank"
    }
}
```

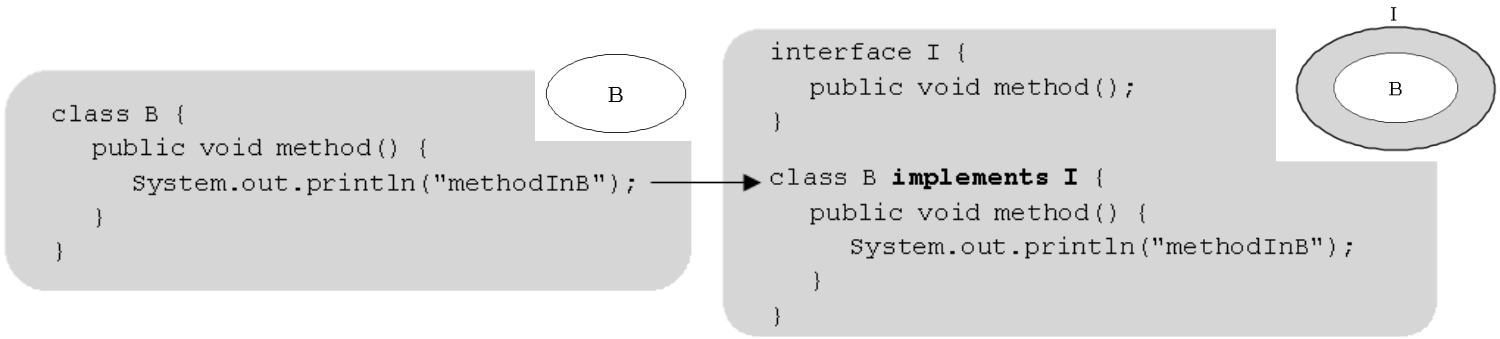
```
class Marine extends GroundUnit {
    Marine() {
        super(40);
        hitPoint = MAX_HP;
    }
}
```

```
class SCV extends GroundUnit implements Repairable{
    SCV() {
        super(60);
        hitPoint = MAX_HP;
    }

    void repair(Repairable r) {
        if (r instanceof Unit) {
            Unit u = (Unit)r;
            while(u.hitPoint!=u.MAX_HP) {
                u.hitPoint++;  // Unit의 HP를 증가시킨다.
            }
        }
    } // repair(Repairable r) {
}
```

```
public static void main(String[] a
    Tank tank = new Tank();
    Marine marine = new Marine();
    SCV scv = new SCV();

    scv.repair(tank);   // SCV가 Tank를 수리한다.
 // scv.repair(marine); // 에러!!!
}
```

7.7 인터페이스의 이해(1/3)

- 인터페이스는…
  - 두 대상(객체) 간의 '연결, 대화, 소통'을 돕는 '중간 역할'을 한다.
  - 선언(설계)와 구현을 분리시키는 것을 가능하게 한다.



- 인터페이스를 이해하려면 먼저 두 가지를 기억하자.
  - 클래스를 사용하는 쪽(User)과 클래스를 제공하는 쪽(Provider)이 있다.
  - 메서드를 사용(호출)하는 쪽(User)에서는 사용하려는 메서드(Provider)의 선언부만 알면 된다.

7.7 인터페이스의 이해(2/3)

▶ 직접적인 관계의 두 클래스(A-B)

```
class A {
	public void methodA(B b) {
		b.methodB();
	}
}
```

```
class B {
	public void methodB() {
		System.out.println("methodB()");
	}
}
```

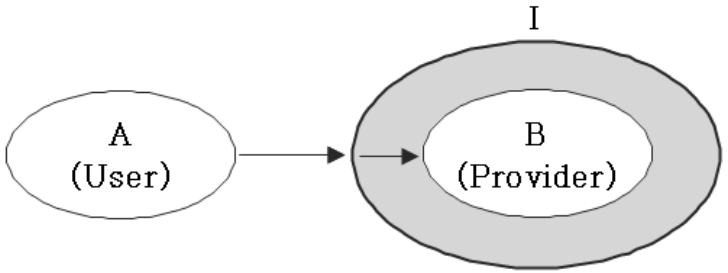
```
class InterfaceTest {
	public static void main(String args[]) {
		A a = new A();
		a.methodA(new B());
	}
}
```

▶ 간접적인 관계의 두 클래스(A-I-B)

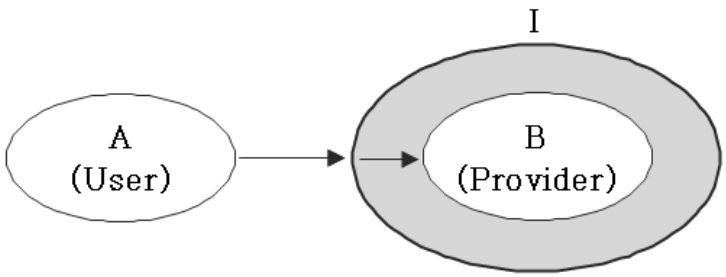
```
class A {
	public void methodA(I i) {
		i.methodB();
	}
}
```

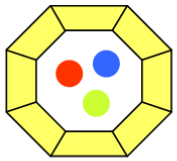
```
interface I { void methodB(); }

class B implements I {
	public void methodB() {
		System.out.println("methodB()");
	}
}
```

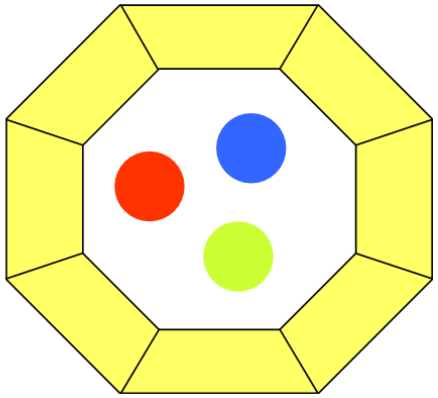
```
class C implements I {
	public void methodB() {
		System.out.println("methodB() in C");
	}
}
```



7.7 인터페이스의 이해(3/3)







```
public class Time {
      private int hour;
      private int minute;
      private int second;

      public int getHour() { return hour; }
      public void setHour(int h) {
            if (h < 0 || h > 23) return;
            hour=h;
      }
      public int getMinute() { return minute; }
      public void setMinute(int m) {
            if (m < 0 || m > 59) return;
            minute=m;
      }
      public int getSecond() { return second; }
      public void setSecond(int s) {
            if (s < 0 || s > 59) return;
            second=s;
      }
}
```

```
public interface TimeIntf {
      public int  getHour();
      public void setHour(int h);

      public int  getMinute();
      public void setMinute(int m);

      public int  getSecond();
      public void setSecond(int s);
}
```

1. 날짜와 시간 & 형식화

1.날짜와 시간

- 대표적인 2개의 Api
  - java.util.Date (Deprecated)
  - java.util.Calendar
- 사용하기 매우 불편해 대체되는 오프소스 라이브러리
  - Joda-Time
- 결국, jdk8에서 joda-time 라이브러리를 수용

> Joda-Time provides a quality replacement for the Java date and time classes. Joda-Time is the de facto standard date and time library for Java prior to **Java SE 8**. Users are now asked to migrate to java.time (JSR-310).

- http://d2.naver.com/helloworld/645609

Calendar

```
Calendar cal = Calendar.getInstance(); //Singleton&Factory pattern
```

13강 인터페이스 & 객체지향개념 II

- 구현체
  - GregorianCalendar
  - JapaneseImperialCalendar
  - BuddhistCalendar

```
private static Calendar createCalendar(TimeZone zone, Locale aLocal
        ...
        ...

        if (aLocale.hasExtensions()) {
            String caltype = aLocale.getUnicodeLocaleType("ca");
            if (caltype != null) {
                switch (caltype) {
                case "buddhist":
                cal = new BuddhistCalendar(zone, aLocale);
                    break;
                case "japanese":
                    cal = new JapaneseImperialCalendar(zone, aLocal
                    break;
                case "gregory":
                    cal = new GregorianCalendar(zone, aLocale);
                    break;
                }
            }
        }
        ...
        ...

        return cal;
    }
```

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

// (CAL.get(Calendar.MONTH) + 1)) 이런 형식으로 하면 다음월을 받아 올 수
// 있다. CAL.get(Calendar.MONTH) + 1로 하면 이상한 값이 나온다. (괄호유무)
System.out.println("월(0~11, 0:1월): " + (CAL.get(Calendar.MONTH) + 1));

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

// DATE와 DAY_OF_MONTH는 같다.
System.out.println("이 달의 몇 일: " + CAL.get(Calendar.DATE));
System.out.println("이 달의 몇 일: " + CAL.get(Calendar.DAY_OF_MONTH));
System.out.println("이 해의 몇 일: " + CAL.get(Calendar.DAY_OF_YEAR));

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

// 천분의 1초를 시간으로 표시하기 위해 3600000으로 나누었다.(1시간 = 60 * 60초)
System.out.println("TimeZone(-12~+12): " +
(CAL.get(Calendar.ZONE_OFFSET)/(60*60*1000)));

// 이 달의 마지막 일을 찾는다.
System.out.println("이 달의 마지막 날: " + CAL.getActualMaximum(Calendar.DATE) );
```

날짜 설정

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

//2016.3.7 로 설정 month   the month between 0-11.
//Mutable
date.set(2016, 2, 7);

date.add(Calendar.YEAR, 2); //date.add(Calendar.JUNE, 2) compile ok
```

Date <-> Calendar

```
//1.Calenter -> Date
Calendar cal = Calendar.getInstance();
...
Date d = new Date(cal.getTimeInMillis());

//2.Date -> Calendar
Date d = new Date();
...
Calendar cal = Calendar.getInstance();
cal.setTime(d);
```

```
Calendar CAL = Calendar.getInstance();
Date d = new Date(CAL.getTimeInMillis());

String s = d.toString();
System.out.println("현재날짜 : "+ s);
```

Apache Common Lang (DateUtils)

```
Date now = new Date();
Date tomorrow = DateUtils.addDays(now, 1);
Date tomorrowAnd2Min = DateUtils.addSeconds(tomorrow, 120);
```

Extra

- 본인(Clint)는 주로 날짜/시간을 저장할 경우 Long 타입으로 저장합니다.

```
public class Connection {
    ...

    @Column(updatable = false)
    private Long created;

    ...

    @PrePersist
    public void onCreate() {
        this.created = System.currentTimeMillis();
    }
}
```

- 이유는 Zone, Parsing 등 편하기 때문입니다.

```
long today = System.currentTimeMillis(); // long 형의 현재시간

DateFormat df = new SimpleDateFormat("HH:mm:ss"); // HH=24h, hh=12h
String str = df.format(today);

Date date = new Date(today);
```

- api 제공시, javascript에서 변환이 편함.

```
var date = new Date(1324339200000);

date.toString("MMM dd");
```

```
<!-- AngularJS -->
<td>{{ conn.created | date : 'yyyy.MM.dd HH:mm:ss' }}</td>
```

2.형식화 클래스

- DecimalFormat
- SimpleDateFormat
- ChoiceFormat
- MessageFormat

DeciamlFormat

숫자를 형식화

```
DecimalFormat df = new DecimalFormat("#,###.##");

Number num = df.parse("1,234,567.89");
//1234567.89

df.format(1234567.89);
//1,234,567.89
```

```
int radius;   double area, circumference;
Scanner scan = new Scanner(System.in);

// 원의 반지름을 입력받는다.
System.out.print("원의 반지름을 입력하세요 : ");
radius = scan.nextInt();
// 원의 넓이 : 반지름의 제곱 * 파이
area = Math.PI * Math.pow(radius, 2);
// 원의 길이 : 지름(반지름*2) * 파이
circumference = 2 * Math.PI * radius;

// 원의 넓이와 길이 값을 출력한다.
System.out.println("원의 넓이 : " + area);
System.out.println("원의 길이 : "+circumference);
System.out.println("");

// DecimalFormat으로 "0.###" 패턴을 생성한다.
DecimalFormat fmt = new DecimalFormat("0.###");

System.out.println("Format 적용 후 (0.###)");
// 원의 넓이와 길이에 "0.###" 패턴을 적용하여 출력한다.
System.out.println("원의 넓이(Format적용) : " + fmt.format(area));
System.out.println("원의 길이(Format적용) : " + fmt.format(circumference));
System.out.println("");

// DecimalFormat 패턴을 "000.#" 으로 변경한다.
fmt.applyPattern("000.#");
System.out.println("Format 변경 후 (000.#)");
// 원의 넓이와 길이에 "000.#" 패턴을 적용하여 출력한다.
System.out.println("원의 넓이(Format 변경 후) : " + fmt.format(area));
System.out.println("원의 길이(Format 변경 후) : " + fmt.format(circumference));
```

SimpleDateFormat

날짜를 형식화

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-mm-dd");

String result = sdf.format(new Date());

SimpleDateFormat sdf2 = new SimpleDateFormat("yyyy년MM월dd일");
Date result2 = sdf2.parse("2016년3월8일");
```

```
SimpleDateFormat df = new SimpleDateFormat("yyyy년 MM월 dd일 hh시 mm분 ss초");

Calendar CAL = Calendar.getInstance();
String today = df.format(CAL.getTime());
System.out.println(today);
```

Symbol	설명	사용 예
y	year : 년도	yy : 14 , yyyy : 2014
M	month in year : 월	M : 1, MM : 01, MMM : 1월
d	day in month : 일	d : 1, dd : 01, ddd : 001, dddd : 0001
h	hour in am/pm(1-12) : 시	h : 1 , hh : 01, hhh : 001, hhhh : 0001
a	am/pm marker	a : 오후 (AM/PM 으로 표시하려면 format에서 Locale.US를 지정)
k	hour in day (1-24) : 시	k : 13, kk : 13, kkk : 013, kkkk : 0013
m	minute in hour : 분	m : 20, mm: 20, mmm : 020, mmmm : 0020
s	second in minute : 초	s : 10, ss : 10, sss : 010. ssss: 0010
w	week in year : 주(년기준)	w : 1, ww : 01, www :001, wwww : 0001
D	day in year : 일(년기준)	D : 1, DD : 01, DDD: 001, DDDD : 0001
E	day of week : 요일	E : 수
F	day of week in month : 월 기준 주간 요일 순번	F : 1, FF : 01, FFF:001, FFFF:0001
G	era designator : 시대	AD
H	hour in day (0-23) : 시	H: 13, HH : 13, HHH : 013 , HHHH:0013
K	hour in am/pm(0-11) : 시	K: 1 , KK : 01, KKK :001, KKKK: 0001
S	fractional seconds : 초	S : 402
W	week in month : 주 (월기준)	W : 1, WW : 01, WWW : 001, WWWW : 0001
z	time zone	z : KST, zzzz: 한국표준시
Z	time zone (RFC 822)	Z : +0900

ChoiceFormat

특정범위에 속하는 값을 형식화

```
String pattern = "60#D|70#C|80<B|90#A";
int[] scores = {91, 90, 80, 88, 70, 52, 60};

ChoiceFormat cf = new ChoiceFormat(pattern);

for(int score : scores){
  System.out.println(cf.format(score));
}
```

MessageFormat

데이터를 정해진 양식에 맞게 형식화

```
String format = "Name : {0}, Tel : {1}, Loc : {2}";
String[] params = {"Clint.cho", "010.1234.5789", "PanGyo"};

MessageFormat messageFormat = new MessageFormat(format);
String result = messageFormat.format(params);

System.out.println(String.format("Hello %s", "Java Study"));
```

3. java.time패키지

- JDK1.8 부터 추가 되었으며 다음과 같이 4개의 하위 패키지를 가지고 있다.

파키지	설명
java.time	날짜와 시간을 다루는데 필요한 핵심 클래스들을 제공
java.time.chrono	표준(ISO)가 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

3.1 java.time패키지의 핵심 클래스

클래스	설명
LocalTime	시간
LocalDate	날짜
LocalDateTime	날짜 + 시간
ZoneDateTime	시간대 + 날찌 + 시간
Instant	시간을 나노초로 표현
Period	두 날짜간의 차이
Duration	두 시간의 차이

객체 생성

- 객체 생성은 now(), of() 두개의 static mehtod를 사용한다.

Temporal과 TemporalAmount

- 날짜와 시간을 표한하기 위한 클래스 들은 모두 Temporal, TemporalAccessor, TemporalAdjuster 인터페이스를 구현 Temporal 이 TemporalAccessor를 상속
- 날짜외 시간의 간격을 표현하기 위한 클래스 들은 TemporalAmount를 구현

TemporalUnit과 TemporalField

- 날짜와 시간의 단위를 정해 놓은 것이 TemporalUnit인터페이스 이고 이것을 구현한 것이 열거형 ChronoUnit이다
- 날짜와 시간의 필드를 정해 놓은 것이 TemporalField인터페이스 이고 이것을 구현한 것이 열거형 ChronoField이다.

3.2 LocalDate와 LocalTime

- LocalDate와 LocalTime은 java.time패키지의 가장 기본이 되는 클래스이며 나머지 클래스들은 이들의 확장이다.
- 객체 생성 : now(), of()
- 특정 필드의 값 가져오기 get(), getXXX(), 매개변수 등은 p.556 참조
- 필드의 값 변경하기 with(), plus(), minus()
- LocalTime 의 tuncatedTo()는 지정된 필드보단 작은 단위의 필드 값을 0으로 변경.

```
LocalTime time = LocalTime.of(12, 34, 56); // 12시 34분 56
time = time.truncatedTo(ChronoUnit.HOURS); // 시(hour)보다
System.out.println(time);
```

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

3.3 Instant

- Instant는 에포크 타임(EPOCH TIME, 1970-01-01 00:00:00 UTC)

```
쉬어가기
협정 세계시(協定世界時, 프랑스어: Temps Universel Coordonné, 영어: Co
국제 전기 통신 연합은 협정 세계시에 대한 통일된 약자를 원했으나, 영어권의 사
"UTC"는 보통 "Universal Time Code"이나 "Universal Time Convention
대한민국의 시간대(Korea Standard Time, KST)는 UTC +9에 속합니다.
```

- Instant를 생성할때는 now()와 ofEpochSecond()를 사용
- Instant는 기존의 java.util.Date를 대체하기 위한 것이며 변환 메서드가 추가 되었다.

3.4 LocalDateTime과 ZonedDateTime

```
LocalDate + LocalTime -> LocalDateTime
LocalDateTime + 시간데 -> ZoneDateTime
```

- 기본적인거는 LocalDate와 LocalTime과 동일
- 시간대는 기존에는 TimeZone클래스를 사용했으나 ZoneId라는 클래스가 생김
- ZoneId는 일광 절약시간(DST, Daylight Saving Time)을 자동적으로 처리해 주므로 더 편리하다. 일광 절약시간 = summer time 낮 시간이 길어지는 봄부터 시곗바늘을 1시간 앞당겼다가 낮 시간이 짧아지는 가을에 되돌리는 제도
- ZoneOffset은 UTC로부터 얼마만큼 떨어져 있는지를 표현한다. 서울은 '+9'이다.
- OffsetDateTime은 시간대를 시간의 차이로만 구분하며 서로 다른 시간대에서 데이터를 주고받을때 시간을 표현하기에 적합하다.

3.5 TemporalAdjusters

- 자주 쓰일만한 날짜 계산들을 대신해주는 메서드를 정의해 놓은 클래스

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

- TemporalAdjuster의 adjustInto()를 구현함으로써 직접 만들 수 있으나 사용은 클래스의 with()를 통해서 사용한다 with() 내부에서 adjustInto() 호출

```
@Override
public LocalDate with(TemporalAdjuster adjuster) {
    // optimizations
    if (adjuster instanceof LocalDate) {
        return (LocalDate) adjuster;
    }
    return (LocalDate) adjuster.adjustInto(this);
}
```

3.6 Period와 Duration

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

- Period.between()

```
public static Period between(LocalDate startDateInclusive, Loca
    return startDateInclusive.until(endDateExclusive);
}

@Override
public Period until(ChronoLocalDate endDateExclusive) {
    LocalDate end = LocalDate.from(endDateExclusive);
    long totalMonths = end.getProlepticMonth() - this.getProlep
    int days = end.day - this.day;
    if (totalMonths > 0 && days < 0) {
        totalMonths--;
        LocalDate calcDate = this.plusMonths(totalMonths);
        days = (int) (end.toEpochDay() - calcDate.toEpochDay()
    } else if (totalMonths < 0 && days > 0) {
        totalMonths++;
        days -= end.lengthOfMonth();
    }
    long years = totalMonths / 12;  // safe
    int months = (int) (totalMonths % 12);  // safe
    return Period.of(Math.toIntExact(years), months, days);
}
```

- between과 until을 위에 있는거 처럼 같은일을 하나 between()은 static메서드 이고, until()은 인스턴스 메서드라는 차이가 있다.
- Duration.between()

```
public static Duration between(Temporal startInclusive, Tempora
    try {
        return ofNanos(startInclusive.until(endExclusive, NANOS
    } catch (DateTimeException | ArithmeticException ex) {
        long secs = startInclusive.until(endExclusive, SECONDS
        long nanos;
        try {
            nanos = endExclusive.getLong(NANO_OF_SECOND) - star
            if (secs > 0 && nanos < 0) {
                secs++;
            } else if (secs < 0 && nanos > 0) {
                secs--;
            }
        } catch (DateTimeException ex2) {
            nanos = 0;
        }
        return ofSeconds(secs, nanos);
    }
}
```

- of(), with(), plus(), minus(), negate(), abs(), toXXX()

3.7 파싱과 포맷

- 형식화와 관련된 클래스들은 java.time.format패키지에 들어있는데 그중에서 DataTimeFormatter가 핵심이다 p.572참고

로케일에 종속된 형식화

- DateTimeFormatter의 static메서드 ofLocalizeDate(), ofLocalizedTime(), ofLocalizedDateTime()은 로케일(locale)에 종속적인 포맷터를 생성한다.

```
DateTimeFormatter formatter = DateTimeFormatter.ofLocalized
String shortFormat = formatter.format(LocalDate.now());
```

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

출력형식 직접 정의하기

```
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern('
```

문자열을 날짜와 시간으로 파싱하기

```
public static LocalDate parse(CharSequence text) {
    return parse(text, DateTimeFormatter.ISO_LOCAL_DATE);
}

public static LocalDate parse(CharSequence text, DateTimeFormat
    Objects.requireNonNull(formatter, "formatter");
    return formatter.parse(text, LocalDate::from);
}

public static DateTimeFormatter ofPattern(String pattern) {
    return new DateTimeFormatterBuilder().appendPattern(patter
}
```

감사합니다.