

자바객체지향프로그래밍

패스트캠퍼스X아놀자: 백엔드 개발 부트 캠프

2023. 7.

박매일 강사

bitcocom@empas.com

강의영역

Java, Python
DataBase, Data Modeling
Web, MVC Framework
Spring, Spring Boot, JPA
React.js, Vue.js

강의경력

한국전력공사 In-House DT 실무자 교육
한양대학교 ERICA 자바 온라인 강의
정보통신산업진흥원 SW융합 채용연수사업
(현)인공지능 사관학교 자바, 웹, 스프링,DB 강의
(현)소프트웨어마이스터고 소프트웨어공학 교육
(현)인프런 자바,웹,스프링,IoT 온라인 콘텐츠 제공
(현)패스트캠퍼스 자바 온라인 콘텐츠 제공

목록

00_자바객체지향프로그래밍.....	1
01_Java(특수문자).....	2
02_Java(기본자료형).....	4
03_Java(연산자).....	5
04_Java(구동 방식).....	6
05_Java(변수, 자료형, 선언).....	7
06_Java(변수와 메서드).....	9
07_Java(기본 자료형과 객체 자료형).....	11
08_Java(변수와 배열).....	14
09_Java(class, object, instance의 차이).....	16
10_Java(메서드 오버로딩(Method Overloading)).....	18
11_Java(private constructor).....	19
12_Java(정보은닉(Information Hiding)).....	20
13_Java(배열과 객체).....	22
14_Java(Class 정리).....	23
15_Java(Inheritance(상속)).....	24
16_Java(Override(재정의)).....	27
17_Java(Override(Object Casting(객체 형 변환)).....	29
18_Java(message polymorphism(다형성)).....	30
19_Java(abstract class).....	32
20_Java(abstract class VS interface).....	34
21_Java(Interface).....	35
22_Java(Interface & JDBC).....	38
23_Java(Object).....	39
24_Java(Package).....	40
25_Java(기본 API & String Class).....	42
26_Java(나만의 API 만들기(IntArray)).....	44
27_Java(ObjectList 만들기).....	46
28_Java(ArrayList).....	48
29_Java(Wrapper Class).....	49
30_Java(Json & Gson).....	51
31_Java(org.json).....	55
32_Java(Geocoding(경도, 위도 추출)).....	58
33_Java(Book Search).....	64

목차

- 특수문자
- 서식문자
- 서식문자의 정렬과 소수점 제한 기능

특수문자

특수문자란 특수한 상황에서 사용되는 문자라고 한다. 주로 '\ ' 이녀석을 사용하는 경우를 말한다. '\ '는 escape라는 이름을 가졌다. 뜻대로 무언가를 탈출시키는 기능을 한다. 원래의 기능에서 탈출시킨다. 즉, 기능을 바꿔준다고 이해된다.

기능을 바꿔준다는 의미를 자세히 살펴보자. 특수문자는 \t, \n, '(작은 따옴표), "(큰 따옴표)가 있다. 이 문자들은 각자 기능이 있다. t, n은 문자열이고, ', "은 char나 String을 나타낸다.

이스케이프()를 사용하면 저 문자들이 원래 기능이 아니라 다른 기능이 된다. \t : tab, \n : enter, ' : 작은따옴표, " : 큰 따옴표, \ : \ 이다.

서식문자

서식문자란 서식에 사용하는 문자이다. 서식의 의미는 글의 형식이다.

%d : decimal, 10진수

%o : octal, 8진수

%h : hexa decimal, 16진수

%c : char

%s : String

%f : float, double

```
int num1 = 37;
System.out.printf("오늘의 기온은 %d입니다.\n", num1);
System.out.println("오늘의 기온은 " + 10 + "입니다");
System.out.printf("37의 10진수는 %d입니다.\n", 37);
System.out.printf("37의 8진수는 %o입니다.\n", 37);
System.out.printf("37의 16진수는 %h입니다.\n", 37);
System.out.printf("%c %s\n", 'p', "KIM YOUNG MO");
float num2 = 1.23f;
double num3 = 1.34343d;
```

```
System.out.printf("%f %f\n", num2, num3);
```

왜 쓰는가?

숫자(상수)를 직접 적어도 되지만 복잡하게 서식문자를 사용하는 이유는 무엇일까? 때로는 다른 서버에서 정보를 입력받을 때가 있다. 그 때 서식문자를 활용하면 자유자재로 사용가능하다.

예) 오늘 날씨를 알려주는 어플이라면 매일 기상청에서 정보를 받을 텐데, 매일 달라진 기온을 입력하려면 귀찮으니까 시간이 걸린다. 서식문자 사용하면 입력받은 기온이 자동으로 출력된다.

중요!

- java에서 String과 변수는 '+'라는 연산자로 합칠 수 있다. 정수(상수)는 그냥 String에 들어간다.

```
int num1 = 10;
System.out.println("오늘의 날씨는 " + num1 + "입니다.");
System.out.println("오늘의 날씨는 10도 입니다.);
```

- double 자료형도 서식문자는 %f 사용한다.

서식문자의 정렬과 소수점 제한 기능

정수와 실수의 자릿수를 정할 수 있다.

```
System.out.printf("오늘의 숫자는 %5d입니다.\n", num1);
System.out.printf("오늘의 숫자는 %4d입니다.\n", num2);
float f = 1.23f;
System.out.printf("float is %.1f\n", f);
```

'%' 뒤에다 표현할 자릿수를 적으면 된다. 정수는 정수를 실수는 '%.정수' (%.8)라고 적는다.

목차

- 기본자료형과 객체자료형
- 자바의 기본자료형
- 형변환

기본자료형

기본자료형과 객체자료형의 차이

기본자료형은 변수에 직접 데이터가 저장된다.

객체자료형은 객체가 변수에 저장되지 않고 객체의 주소를 저장한다. 크기는 4바이트로 고정된다.

배운 점: 파이썬은 기본자료형도 객체이기 때문에 예를 들어, int라도 그 주소가 변수에 저장된다. 데이터의 값이 저장되지 않는다.

자바의 기본자료형

자바 속 기본자료형은 char, int, double, boolean 등이 있다. String이 기본자료형이 아니라 객체라는 점이 새로웠다. 그래서 대문자로 시작하는 건가.. 파이썬과 비슷하면서 원리에서 다른 부분이 있어 차이를 구분하며 배우고 있다.

~~약간 헷갈린다~~

주로 사용하는 자료형은 char(2), int(4), double(8), boolean(1)라고 한다.

형변환

형변환이란 (자료)형을 변환한다는 의미다. 각 자료형은 크기가 정해져 있기에 변환할 땐 주의해야 한다. 작은 자료형이 큰 자료형으로 변환하는 건 문제가 안되지만, 큰 자료형이 작은 자료형으로 바뀔 땐 문제가 생긴다.

큰 자료형에서 작은 자료형으로 변환하면 큰 자료형의 데이터가 손실될 수 있다.

목차

- 대입연산자
- 산술연산자
- 관계연산자
- 증감연산자

대입연산자

값을 대입할 때 사용하는 연산자이다. 기호는 '='이다

수학에서는 두 값이 같을 때 사용하지만 프로그래밍에서는 대입할 때 사용한다.

산술연산자

사칙연산할 때 사용하는 연산자

주의 : '/'(나누기) 사용하면 값은 int다. 뒤에 나머지를 지운다. 파이썬과 차이점

관계연산자

피연산자 두개의 관계를 비교해서 True / False을 알려주는 연산자이다.

증감연산자

++, -- : 1을 더한다. 1을 빼다는 의미를 가진다.

원래 $x = x + 1$ 의 식을 간단하게 표현한 것이다.

중요

++가 전위(피연산자 전에 위치)하면 피연산자에 1을 더하고 피연산자를 출력한다.

++가 후위(피연산자 후에 위치)하면 피연산자를 출력하고 메모리에서 1을 더한다.

```
int x = 1;
System.out.println(++x); 하면 x = 2
System.out.println(x++); 하면 x = 1이 나오지만 메모리에선 1이 더해져있다.
```

목차

- byte code
- JVM
- 자바의 불편함

byte code, 중간어, .class

인텔리J에서 코드를 실행하면 먼저 java파일은 class파일로 컴파일된다. 이때 컴파일을 1차 컴파일이라 하자.이 때 자바로 작성된 코드는 byte code로 바뀐다. 하지만 완벽한 기계어는 아니기 때문에 중간어라고도 한다고 함.

왜?

그럼 왜? 중간어로 한번 바꿀까? 오히려 번거로울 텐데,, 그 이유는 어떤 OS에서 프로그램이 작동될지 모르기 때문이다.어떤 OS든 자바 프로그램이 운영될 수 있게 1차적으로 byte code로 준비시킨다고 이해하면 된다.

예시) 어디에서나 먹을 수 있는 즉석조리식품 느낌. 바로 먹을 수 있는 요리라 할 순 없지만 약간의 조리만 필요한 상태

JVM(Java Virtual Machine)

컴파일을 하는 객체는 JVM이다. 가상 머신인데, 이 친구가 자바에서 핵심적인 역할을 한다. 명령이 없으면 OS 위에 그냥 있다가 실행하라는 명령을 받으면 1차 컴파일을 하고 이후 각 OS에 맞게 2차 컴파일을 한 뒤 메모리에 로딩해 실행시킨다. 브로커 역할

그래서 자바의 불편함은?

딱 봐도 여러 과정을 거치니 자바의 단점? 이 파악된다.

타언어보다 속도가 느릴 수 있다

~~어떤 상황에 따라 다르다~~

또한, 자바를 실행하려면 무조건 JVM이 있어야 한다.

나는 개발은 안 하지만 자바만 실행할 거야!라고 해도 JVM을 설치해야 한다.

목차

- 변수
- 자료형
- 선언

변수

메모리 속 기억공간이라 한다. 컴퓨터에게 1+1 연산을 시키려면 어떻게 해야 할까? 먼저 알려줘야 한다. 컴퓨터에게 알려주려면 메모리에 1+1을 넣어야 한다.

기억공간이라 했으니 공간의 특징이 있다.

- 1.크기가 있다
- 2.들어가는 데이터의 종류가 있다.

자료형(Data type)

기억 공간의 크기와 들어가는 데이터의 종류를 결정하는 자료형이다. 자바에는 int, char, String, boolean 등 자료형이 있다. 자료형과 그 기억공간의 이름을 붙이면 기억공간이 만들어진다.

아파트도 평 수가 나눠져있는 것처럼

자료형의 크기

- int : 4byte, 32bit, 2^{32} 개 정수 표현 가능. 42억 개
- long : 8byte, 64bit, 매우 큼
- float : 4byte, 32bit
- double : 8byte, 32bit

자료형의 종류

- primitive(기본자료형) : boolean, int, long, float, double
- non-primitive(객체 자료형) : String, array... 기본자료형으로는 부족할 땐? 컴파일러에서 자동으로 제공해주는 자료형도 유용하지만 때로는 다른 자료형이 필요할 때도 있다. 예를 들어) 책을 기억공간에 넣고 싶다. 근데 책을 넣을 수 있는 자료형이 자바에는 없다? 그럼 어찌지?? 만들면 된다. 뭐로? Class로 만든다.

내가 원하는 자료형은 설계해서 직접 만든다. 이게 객체지향 프로그램의 기본이다.

선언

자료형과 그 기억공간에 이름을 붙이는 작업을 선언이라 한다. 자료형과 변수가 만나는 걸 선언이라 한다!!

```
""int a; // 선언, int형 크기와 int형이 들어갈 수 있는 기억 공간이 만들어짐
```

""변수가 선언되면 컴파일러는 변수 목록표를 만든다. (Symbol table). 이 목록표는 key와 value로 구성된다. key 값에서는 각 변수가 들어가고 value에는 각 변수의 메모리 주소(address)가 있다. 그래서 key 값인 변수를 호출하면 value에 있는 메모리 주소를 찾아가서 변수를 가져온다.

결국, 기본 자료형은 Symbol table에서 변수의 주소가 들어간다. 값의 주소가 아니다.

목차

- 메서드
- 매개변수 전달 기법(parameter passing)

핵심

메서드는 매개변수를 받아 동작한 후 결과값을 변수처럼 가지고 있다

메서드

변수는 하나의 데이터를 담고 있다. 2개 이상의 데이터를 담을 땐 그 데이터의 주소를 담지, 데이터를 담는 것이 아니다. 메서드도 비슷한 맥락이다. 인자 값을 받아서 기능을 수행한다. 그다음 결과 값을 메서드 이름에 저장한다.

```
public int sum(int a, int b) {
    return a + b;
}
int c = sum(3+5); // 3 + 5의 결과값이 sum에 담기고 그 값이 c에 할당된다.
```

매개변수 전달 기법(parameter passing)

메서드를 호출할 때 메서드의 선언부로 매개변수를 전달하는 방법은 크게 2가지가 있다. 총 5가지 정도 있다고 한다.

한 번쯤은 들어봤을 두 친구들 자바 처음 배울 때 혼란에 빠트리는

- Call by Value
- Call by reference

Call by value

메서드 호출할 때 값(value)을 보낸다.

~~다 값으로 하자 안 할 수가 있나?~~

변수(메모리 공간)에 저장된 값을 직접 메서드 선언부로 보낸다. 그럼 메서드 선언부는 새롭게 변수(메모리 공간)를 만들고 기능을 한 다음 결과값을 반환한다.

결론 : 메서드 호출부의 매개변수와 선언부의 매개변수는 기억공간(메모리 공간)을 공유하지 않는다

```
//호출부
int a = 3;
int b= 5;
int result = sum(a, b);
//선언부
public int sum(int c, int d) { //c, d는 a, b와 기억공간을 공유하지 않는다.
    return c + d;
}
```

Call by reference

메서드 호출할 때 매개 변수의 주소를 보낸다. 값을 직접 보내지 않고 주소를 메서드 선언부로 보낸다. 그럼 메서드 선언부는 받은 주소를 참조하는 새로운 변수를 만들어 작업한다. 그 뒤 결과값을 반환한다.

결론 : 메서드 호출부의 매개변수와 선언부의 매개변수는 기억공간을 공유한다.

```
int[] arr = {10, 20};
int[] result = sum(arr);
int sum = 0;
public int[] sum(arr2){ //arr2는 arr과 같은 주소를 공유한다.
    for(int i : arr2) {
        sum += i;
    }
}
```

목차

- 기본 자료형과 객체 자료형
- 객체는 덩어리다
- 객체는 어떻게 할당하지?
- 객체의 다른 이름

핵심

class는 새로운 자료형을 만드는(모델링하는) 도구이다

기본 자료형과 객체 자료형

자바에서 제공하는 기본 자료형만 잘 쓰면 되지. 왜 객체 자료형이 필요할까? 그 이유는 기본 자료형만으로는 프로그램을 제대로 만들 수 없기 때문이다.

간단

예를 들면,

- 정수 1개를 담는 자료형을 선언하세요!
- 책 1개를 담는 자료형을 선언하세요!

정수는 담을 수 있는데 책은 어떻게 담지? 정수는 int라는 자료형이 있어서 메모리의 크기와 종류를 정할 수 있다. 하지만 책이라는 자료형은 없기 때문에 설계해야 한다.

이때 필요한 도구가 Class이다.

객체란 덩어리다.

책을 담기 위해선 'Book'이라는 자료형이 필요하다. 하지만 자바는 제공하지 않기 때문에 우리가 만들어야 한다. Class로 말이다. 책에는 제목, 출판사, 페이지수 등 다양한 속성(상태 정보)이 있다. 이것을 컴퓨터 안에서는 어떻게 표현할까?

바로 아래처럼

```
public class Book{
    public String title;
    public int price;
    public String company;
    public int pages;
}
```

컴퓨터에서 책은 위에 코드처럼 표현할 수 있다. 마치 여러 변수가 덩어리를 이루고 있다. 이게 객체다. 그러면 이 객체를 컴퓨터에게 알려주려면 메모리에 저장해야 한다. 이 과정을 객체 생성이라고 한다.

객체는 어떻게 할당하지?

그럼 이제 문제는 거의 해결됐다. 책을 넣고 싶어서 'Book'이라는 자료형을 만들었다. 그럼 이제 선언해 보자.

```
Book b; // 책을 담을 수 있는 기억공간(변수) 생성
```

선언은 완료됐다. 그럼 이제 변수 b에 객체를 넣으면 될까? 안된다. 변수는 하나의 데이터 타입만 넣을 수 있기 때문에 다양한 자료형이 뭉친 덩어리인 객체는 들어갈 수 없다.

~~1안실 교사원에 10명이 들어가려는 상황~~

그럼 이 문제를 어떻게 해결할까? 바로 객체의 주소를 변수에 넣는다. 주소를 알면 객체를 찾아갈 수 있으니 객체를 넣지 않고 객체의 주소를 넣는다.

객체의 다른 이름

- VO(Value Object) : 객체가 비록 여러 자료형이 합쳐져 있지만 하나로 인식하기 때문에 Value라고 부른다.
- DTO(Data Transfer Object) : 데이터를 이동시키는 역할을 한다는 의미. 결국 객체는 만드는데서 끝나지 않고 이동해서 사용하는 게 중요하기 때문이다.

객체 생성

그럼 메모리에 우리가 설계한 책의 자료형(Book)으로 어떻게 책을 만들까?
코드를 보자

```
Book b; // 기억공간(변수)를 만듦. 담기진 않음.  
b = new Book(); // 메모리에 객체를 생성함.
```

아~ new라는 도구를 통해서 객체를 메모리에 생성하는구나. 그럼 메모리가 실제일까? 아니다. 메모리에 담긴 객체가 실체이다. 그래서 메모리에 만들어진 객체를 인스턴스라고 한다. 그리고 이런 인스턴스를 담는 변수 b를 인스턴스 변수, 객체 변수라고 한다. 심지어 b를 객체라고도 함. 왜? 객체를 담고 있으니까.

상태 정보, 속성, 멤버 변수

다 같은 말이다. 영어로는 attributes, member variable이라고 한다. 우리가 만든 class에는 여러 자료형이 섞여있다. int, String, boolean 등등.. 그래서 하나의 class에 속해진 멤버와 같으니 그들을 멤버 변수라 말한다.

또는 객체는 사물이니까 사물의 속성이라고도 말한다.

목차

- 배열이란?
- 2차원 배열은 아파트다.
- 2차원 배열 생성 및 출력

핵심

배열은 여러 개의 변수를 쉽게 이동시키기 위한 자료형이다

배열이란?

만약 정수 5개를 사용해야 한다면 int 변수를 5개를 만들어야 한다. 그런데 이 변수들을 이동시켜야 하는 상황이 발생할 수 있다.

예) 메서드에 인자로 들어갈 때

그럼 이 5개 변수를 하나 하나씩 옮겨야 한다. 이런 불편함을 줄이기 위해 배열을 쓸 수 있다.

배열은 여러 개의 기억공간이 하나로 합쳐진 구조이다. 같은 자료형만 사용할 수 있다. 아까 말한 5개의 정수도 하나의 배열에 넣어두면 이 배열 하나만 들고 다니면 된다. ~~장바구니에 들어간 과일들처럼~~

여러 개의 변수가 모여있다고 하니 객체랑 비슷하지 않은가? 전에 객체를 덩어리라고 표현했다. 그럼 객체처럼 배열을 담는 변수는 배열을 담고 있는 게 아니라 배열의 주소를 담고 있다. 객체와의 차이점은 다른 자료형은 담을 수 없다는 점. ~~상황에 따라서 단점일 수도, 장점일 수도~~

```
int[] array; // 배열 선언, 메모리에 배열을 담을 수 있는 기억공간이 만들어짐
array = new int[3]; // 배열 객체 생성. 객체 주소를 array에 담는다.
```

2차원 배열은 아파트다.

2차원 배열은 아파트다. 보통 2차원 배열을 어려워 한다. ~~네-아야가..~~

아파트에는 동이 있고, 동에는 층이 있다.

```
int[][] a = new int[3][4] // 3개 동이 있고 각 동에는 4개 층이 있다.
```

맨 처음 a에 담긴 배열의 주소는 int [3]의 주소이다. 그래서 a.length()하면 3이 나온다.

- int[0]이 int[4]의 주소를,
- int[1]이 int[4]의 주소를,
- int[2]이 int[4]의 주소를,
- int[3]이 int[4]의 주소를 가지고 있다.

a가 2차원 배열의 주소를 한 번에 가지고 있지 않다는 말!

그래서 행을 아파트의 동, 열을 아파트 동의 층 수라고 표현했다.

가변 배열

행의 길이만 지정하고 열의 길이는 따로 지정할 수 있다. 왜냐? 각 행이 그 열의 주소를 가지고 있기 때문이다.

```
int[][] arr = new int[3][]; // 3행만 지정
arr[0] = new int[4]; // 0동에 4층 생성
arr[1] = new int[5]; // 1동에 5층 생성
arr[2] = new int[6]; // 2동에 6층 생성
```

2차원 배열 생성 및 출력

- 2차원 배열이 메모리에서 어떻게 구조를 이루는지 알고 있으면 생성이 어렵지 않다.

```
// Two Dimentional array
int[][] arr = new int[5][4];
int element;
for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j < arr[i].length; j++) {
        element = arr[i][j];
        element = 4;
        System.out.printf("%d\t", element);
    }
    System.out.println();
}
String[][] arr = new String[4][];
// Create variable array
for (int i = 0; i < arr.length; i++){
    arr[i] = new String[i+1];
}
// Assign value and print elements
for (int i = 0; i < arr.length; i++){
    for (int j = 0; j < arr[i].length; j++){
        arr[i][j] = "*";
        System.out.printf("%s", arr[i][j]);
    }
    System.out.println();
}
```


Java - class, object, instance의 차이

09

목차

- Class
- Object
- Instance
- 그럼 객체가 왜 필요할까?

핵심

- Class: 객체를 설계하는 도구
- Object: 선언된 객체 변수
- Instance: 생성되고 값이 있는 객체
- 객체를 만드는 이유는 다양한 자료형을 한 번에 이동하기 위함이다.

Class

Java를** 배우면 자주 등장하는 용어가 있다. Class, Object, Instance이다. 이 용어들이 객체와 관련되어 있다는 건 알겠는데, 정확하게 어떤 차이가 있는지 모르는 경우가 있다. 한번 알아보자.

먼저 Class다. Class는 객체를 설계하는 도구이다. 내가 Book이라는 객체를 만드려고 할 때 먼저 Book이라는 객체의 속성을 설계해야 한다. 아래 코드처럼

```
public class Book {
    public String title; // 제목
    public int price; // 가격
}
```

책이 가지는 속성(제목, 가격)으로 책을 설계했다. 물론 현실 세계에서 책은 저렇게 단순하지는 않지만 우리는 컴퓨터에서 책을 정의해야 하기 때문에 필요한 속성만 가지고 책을 만들 수 있다. 이제 컴퓨터에서는 책이라는 건 저 Class를 의미한다. 이와 같이 Class는 객체를 설계하는 도구이다. Class가 만들어졌다고 해서 객체가 생성된 것은 아니다.

Object

Object는 선언된 객체 변수이다. Class로 객체를 설계하고 객체를 사용하기 위해선 객체를 생성해야 한다. 생성하기 전에 객체 변수를 선언해야 하는데, 그 객체 변수를 Object라 한다. 변수라는 이름을 생략한 것이다. Object는 구체적이지 않다. 책 예시처럼 책은 큰 범주일 뿐이지 구체적인 책을 지칭하지 않는다. 이처럼 Object는 객체를 담을 수 있는 변수를 말한다. 여기서 특정한 책이 만들어지면 그것을 Instance라고 한다.

```
Book book; // 객체 변수를 선언했다. 저 book변수를 객체(변수)라 한다.
```

Instance

Instance는 메모리에 로딩된 구체적인 객체를 말한다. 객체를 사용하려면 무조건 메모리에 객체를 생성해야 한다. 그때 new 연산자와 생성자 메서드를 사용한다. 객체를 생성하면 메모리에 객체가 만들어지는데 이 객체를 Instance라고 한다. 이전 객체 변수보다 훨씬 구체적인 객체가 된다. 왜냐하면 값이 들어가 있기(초기화) 됐기 때문이다. 예를 들어, 책이라 해도 'java의 정석'이라는 구체적인 객체가 된 것이다.

```
Book book = new Book(); // 이렇게 생성된 객체를 Instance라고 한다.
```

하지만 Object와 Instance를 무 자르듯 딱 자를 수 없다. 객체가 생성되면 이제 객체 변수도 인스턴스 변수라고 이름이 변경된다. 그래서 Object를 책이라고 한다면 Instance는 'java의 정석'과 같이 구체적인 책 한 권을 말한다.

그럼 객체가 왜 필요할까?

객체가 필요한 이유는 현실 세계의 사물을 컴퓨터에 표현한다는 목적도 있지만 결국엔 데이터를 이동하기 위함이다. 다양한 자료형을 하나로 묶어 객체로 표현하면 그 여러 개 자료형은 바구니(객체)에 담겨서 이동하기가 편해진다.

Java - 메서드 오버로딩(Method Overloading)

10

목차

- 메서드 오버로딩(Method Overloading)이란?
- 이름이 같으면 어떤 함수인지 찾는데 속도가 느리지 않을까?

핵심

- 메서드 오버로딩은 메서드의 이름이 같지만 매개변수의 타입과 개수를 다르게 하는 기능이다.

메서드 오버로딩(Method Overloading)이란?

메서드 오버로딩은 메서드의 이름이 같지만 매개변수의 타입과 개수를 다르게 하는 기능이다. 매개변수의 타입과 개수를 Signature라고 한다. 즉, 이름이 같더라도 매개변수의 **Signature**가 다르다면 에러가 발생하지 않는다. 그럼 왜 사용할까?

당연히 편리하기 때문이다. 덧셈 기능을 하는 메서드가 있다. 그런데 매개변수의 타입과 개수가 다르다고 메서드의 이름을 다 다르게 하면 생산성이 떨어진다. 그래서 같은 이름을 쓰되, 매개변수의 개수와 타입을 다르게 하는 것이다.

이름이 같으면 어떤 함수인지 찾는데 속도가 느리지 않을까?

그럼 이와 같은 의문이 들 수 있다. 컴퓨터가 같은 이름의 메서드가 100개 있다면 현재 호출된 메서드가 무엇인지 100번을 찾아야 하는 것 아닐까? 아니다. 단호 왜냐하면 객체의 메서드가 heap area(정확하게는 Method area의 non-static zone)에 저장될 때는 컴퓨터가 각 메서드의 이름을 매개변수의 타입과 개수에 맞춰서 변경한다. 그렇기에 소스 코드에서는 같은 메서드 이름이지만 컴파일러가 인식할 때는 각각 다른 이름이다. 이를 **정적 바인딩**이라고도 한다. 그래서 속도와는 상관이 없다.

정적 바인딩

컴파일할 때 호출되는 메서드를 정하는 바인딩, 소스 코드에선 이름이 같은 메서드가 자기만의 주소가 정해져 있는 것.

목차

- 객체를 생성하지 않고 메서드를 사용하는 법
- 인스턴스 메서드와 클래스 메서드

핵심

- private 생성자 메서드를 사용하려면 모든 메서드에 static 키워드가 있어야한다.

객체를 생성하지 않고 메서드를 사용하는 법

생성자는 public과 private 둘 다 사용 가능하다. public 키워드가 있으면 다른 클래스에서 객체를 생성할 수 있다. 하지만 private이면 객체 생성을 할 수 없다. 왜냐면 생성자에 접근할 수 없기 때문이다. 그러면 private 생성자가 있는 객체의 메서드를 사용하려면 어떻게 해야할까? 이 말은 객체를 생성하지 않고 메서드를 어떻게 사용할까?와 같은 말이다.

인스턴스 메서드와 클래스 메서드

메서드는 인스턴스 메서드와 클래스 메서드로 나뉜다. 둘의 차이는 static이 있느냐 없느냐의 차이이다. 인스턴스 메서드는 static이 없기 때문에 반드시 new 연산자로 객체를 생성해야 사용할 수 있다. 하지만 클래스 메서드는 static이 있기 때문에 new 연산자로 객체를 생성하지 않더라도 메모리에 로딩될 수 있다. 즉, 사용할 수 있다. 방법은 클래스의 이름으로 접근하는 방법이다. 클래스의 이름으로 접근하는 그 순간, static이 있는 메서드는 메모리에 로딩이 되고 호출이 된다.

결국, 객체를 생성하지 않고 메서드를 사용하는 방법은 모든 멤버함수에 static을 붙여준다.

```
Math.random() // Math도 private 생성자가 있지만 멤버함수가 static이기에 (클래스명.메소드)로 사용 가능하다.
System.out.println() // System도 마찬가지로 멤버함수에 static이 있어서 (클래스명.메소드)로 사용 가능하다.
```

자주 사용하는 클래스면 모든 멤버함수에 static을 붙이면 빠르게 사용할 수 있다.

Java - 정보은닉(Information Hiding)	12
---------------------------------	----

목차

- 정보 은닉(Information Hiding)은 왜 필요한가?
- 정보 은닉(Information Hiding)은 어떻게 하는가?
- 결국 잘못된 값이 들어가는 건 똑같지 않나?
- 잘 설계된 클래스

핵심

- 정보 은닉은 클래스의 상태 정보를 안전하게 보호하기 위한 기능이다

정보 은닉(Information Hiding)은 왜 필요한가?

정보 은닉은 다른 클래스로부터 접근을 막는 행동이다. 그럼 왜 정보 은닉을 해야할까? 여기서 정보는 객체의 상태정보를 말한다. 정보 은닉을 하지 않으면 객체의 상태정보에 누구나 접근이 가능해진다. 이 의미는 잘못된 데이터가 들어갈 수 있다는 의미다. 잘못된 데이터가 들어가는 것을 막기 위해 정보 은닉을 한다.

예를 들면, 사람의 위는 누구나 접근할 수 있게 개방되어 있지 않다. 뼈와 근육으로 감추어져 있다. 왜 그런가? 어떤 사람이 내 위에 직접적으로 음식물을 전달하면 내 위는 위험할 수 있다. 돌이 울지, 사과가 울지 모르기 때문이다. 그래서 사람은 입이 존재한다. 입에서 음식물을 판단하고 위에 갈 수 있는 음식이면 삼키지만 아니면 뱉는다.

정보 은닉(Information Hiding)은 어떻게 하는가?

정보 은닉을 하기 위해선 `private` 키워드를 사용한다. 객체 상태정보에 `private`을 사용해서 안전하게 보호한다. 그럼 어떻게 객체 상태정보에 데이터를 입력할까? 아까 예시처럼 입과 같은 기능이 필요하다. Setter, Getter 메서드이다.

Setter, Getter method

객체의 상태 정보를 `private`으로 설정하면 어떻게 상태 정보를 입력하고 출력할 수 있을까? 이 때 setter, getter 메서드를 사용한다. 상태 변수 하나에 setter, getter 메서드를 하나씩 만든다.

```

public class Book{
    private String title;
    private int price;
}

public void setName(String title){
    this.title = title;
}

public String getName() {
    return title;
}

... // price도 마찬가지로 구현한다.

```

결국 잘못된 값이 들어가는 건 똑같지 않나?

그럼 이런 생각이 든다. 메서드로 하더라도 내가 잘못된 데이터를 넣을 수 있지 않을까? 물론 넣을 수 있다. 하지만 메서드이기 때문에 잘못된 데이터를 걸러낼 수 있다. 바로 if문이나 while 문 등 다양한 기능을 활용할 수 있다. 메서드를 사용하지 않고 상태 정보를 직접 수정할 수 있었다면 잘못된 데이터를 걸러낼 방법이 없다.

잘 설계된 클래스

잘 설계된 클래스는 어떤 구조일까? 아무나 객체의 데이터를 접근하거나 사용하는 것을 막는 구조이다 단계 별로 살펴보자.

1. 객체의 상태 정보가 은닉되어 있어야 한다. (private 사용)
2. 기본 생성자가 명시적으로 표현되고, 오버로딩 생성자를 만들어 적절하게 초기화한다.
3. Setter & Getter 메서드가 존재한다.
4. 객체의 전체 정보를 출력하는 toString 메서드가 존재한다.

목차

- 배열과 객체(Array vs Object)의 공통점
- 배열과 객체의 차이점
- 객체 배열

핵심

- 배열과 객체는 여러개의 데이터를 담는 바구니다

배열과 객체(Array vs Object)의 공통점

배열과 객체의 공통점은 여러 개의 데이터를 담는 자료형이라는 점이다. 바구니 그렇기에 변수에는 값이 아니라 주소가 참조된다.

배열과 객체의 차이점

배열과 객체의 차이점은 담겨진 데이터들이 동일한 자료형인지 아닌지로 나뉜다. 모든 데이터가 동일한 자료형이라면 배열을 사용해야 한다. 이에 반해 데이터가 서로 다른 자료형이라면 객체를 설계해 사용해야 한다. 배열은 자바에서 기본으로 제공하는 자료형이기 때문에 따로 설계할 필요없다. 하지만 객체는 class를 통해 직접 설계를 하고 객체를 생성해야 한다. ~~내 입맛대로 만드는 객체~~

객체 배열

좀 더 응용해보자. 배열은 데이터가 같은 자료형일 때 사용한다고 했다. 그러면 그 데이터의 자료형이 객체라면? 배열에 담을 수 있지 않을까? 담을 수 있다. 배열에 담긴 데이터의 자료형이 객체라면 그 배열은 객체 배열이라고 할 수 있다. 마치 2차원 배열과 비슷하다.

```
Book[] book = new Book[3]; // 객체 배열 생성
book[0] = new Book(); // 0번 자리에 객체 생성
book[1] = new Book(); // 1번 자리에 객체 생성
book[2] = new Book(); // 2번 자리에 객체 생성
```

목차

- Class를 보는 관점 2가지
- 우리가 만드는 Class의 종류 3가지
- 우리가 사용하게 될 Class의 종류

핵심

- Class는 무궁무진하다. 내가 어떻게 쓰느냐에 따라 활용 범위가 달라진다

Class를 보는 관점 2가지

지금까지 클래스에 대해서 배웠다. 2가지 관점에서 클래스를 바라봤다.

1. Data type 측면 : 새로운 자료형(객체)를 설계하는 도구 -> 모델링
2. OOP(객체 지향) 측면 : 객체의 상태 정보와 행위정보를 추출하여 캡슐화 시키는 도구

그래서 클래스를 모델이라고도 한다. 왜냐하면 역할이 정해져있기 때문이다.

우리가 만드는 Class의 종류 3가지

지금까지는 DTO기능을 하는 클래스에 대해서 주로 배웠다. DTO는 객체를 바꾸니 역할로 사용하는 것이다. 다양한 데이터를 담아서 이동하는 역할이다. 하지만 이외에도 클래스의 기능은 더 있다.

1. DTO(Data Transfer Object) : 데이터를 이동하는 역할, 담는 역할
2. DAO(Data Access Object) : 데이터를 처리하는 역할, 데이터베이스와 연결해 CRUD를 하기도 한다.
3. Utility(Helper Object) : 다양한 기능을 가지고 있어서 돕는 역할(인코딩, 날짜, 시간, ...)

우리가 사용하게 될 Class의 종류

1. 자바에서 기본으로 제공하는 클래스 : Array, Hash, System ...
2. 직접 설계해서 사용하는 클래스 : DTO, DAO, Utility
3. 외부 회사나 사람이 만들어서 배포하는 클래스 : Gson, Jsoup, POI, iText ...

목차

- 수평적 설계와 수직적 설계
- 상속 체이닝
- 상속과 Override의 활용 : TV와 리모콘
- 부모 클래스에 상태 정보가 있는 경우

핵심

- 상속은 자식이 부모의 기능을 사용할 수 있다는 의미이다.(물려 받는다는 의미가 아니다. 확장 의미이다.)

Class를 행위 정보 위주로 바라보자.

그 동안 클래스를 바라보는 관점은 다양한 데이터를 담는 자료형으로 살펴봤다. ~~바구니 역할~~ 즉, 상태 정보에 초점을 맞춘 관점이었다. 하지만 이제는 객체의 행위 정보에 초점을 맞춰서 이야기를 해보자. 먼저 상속에 대해서 알아보자.

수평적 설계와 수직적 설계

클래스는 객체를 설계하는 도구라는 의미는 변함없다. 그 중 설계하는 방법으로 크게 2가지가 있다. 수평적 설계와 수직적 설계이다. 수평적 설계는 각 클래스를 독립적 설계하는 방법을 말한다. 독립적이라는 말은 좋아보이지만 코드를 관리하는 입장에선 그닥 좋진 않다. 만약 100개의 클래스가 비슷한 상태 정보과 행위 정보를 가지고 있다면 100개 클래스에 같은 코드를 입력해야한다. 너무나 비생산적이다. ~~개발자는 이런 광경을 눈 뜨고 못 본다.~~

수평적으로 클래스를 설계하면

1. 코드의 중복이 발생하고
2. 코드 관리가 어려워진다.

반면에, 수직적 구조는 독립적으로 클래스를 설계하지 않고 계층을 두고 설계하는 방법이다. 마치 부모, 자식처럼 말이다. 만약 개, 고양이 클래스를 만들어야 할 때 이 둘을 포괄하는 동물이라는 클래스를 만들어서 개, 고양이 클래스가 그 정보를 상속 받으면 되지 않을까?

수평적 설계는

1. 코드의 중복을 줄일 수 있고
2. 코드 관리가 수평적 설계보다 쉬워진다.

이 때 자식 클래스인 개, 고양이는 기존 기능에서 확장을 했기 때문에 extends라는 키워드로 상속 받는다. 그리고 부모 클래스를 super class라고 부르고 자식 클래스를 sub class라 부른다.

```
public class Dog extends Animal {
    public static void main(String[] args) {
    }
}
```

상속 체이닝(Inheritance Chaining)

상속 체이닝은 가장 상위 클래스부터 객체가 생성된 뒤 연쇄적(Chaining)으로 하위 클래스로 객체가 생성되는 방법을 말한다. 왜 이 방법이 쓰이냐면 실생활 예시를 들자면, 내가 존재하기 위해선 반드시 부모님이 먼저 태어나야 한다. 만약 개라는 객체를 만들기 위해선 상위 클래스는 동물이라는 클래스가 만들어져야 한다. 이 과정은 super()라는 키워드로 이루어진다.

```
public class Animal{ // super 클래스
    private String name;
    private int weight;
}
public class Dog extends Animal {
    public void eat() {
        System.out.println("와구와구");
    }
    public Dog() {
        super() // Dog 객체를 생성할 때 생성자 메서드 안에 부모 객체 생성자 super()가
        // 있어서 상위 클래스인 Animal이 메모리에 먼저 생성된다. 그 다음 Dog가 생성된다.
    }
}
```

상속과 Override의 활용 : TV와 리모콘

상속을 할 경우 여러가지 장점이 있지만 다른 관점에서 살펴보자. 지금까지는 내가 설계한 클래스를 사용했기 때문에 어떤 메서드가 있는지 알고 쓸 수 있다. 하지만 다른 사람이 내 객체

를 사용할 때는 어떤 기능이 있는지 전부 알지 못한다. 왜? 그들에게 배포할 때 java파일(소스 코드)을 배포하면 보안의 위험이 있으니 byte code로 컴파일된 파일을 배포한다. 그럼 상대방은 소스코드가 없으니 내 객체를 제대로 쓸 수가 없다. 그러면 이 때 필요한 건 내 객체를 사용할 수 있는 도구가 필요하다. 리모콘

우리가 TV를 살 때 리모콘을 같이 사는 이유는 TV를 사용하기 위해서다. 리모콘이 없이 TV를 사용하려면 TV 속 모든 기능을 다 알아야한다. 하지만 회사에서 TV 기능이 담긴 설계도를 줄 수는 없는 노릇이다. 대신에 회사에는 리모콘을 제공한다. 사용자는 TV가 속에서 어떻게 작동하는지 알 수는 없지만 리모콘이 있다면 TV를 사용할 수 있다.

다시 본문으로 돌아와서 소스 코드 없이도 내 객체를 다른 사람이 쓸 수 있게 하려면 리모콘 역할이 필요하다. 이 때 Override 기능이 활용된다. 하위 메서드가 Override(재정의) 되면 상위 객체로 하위 메서드에 접근할 수 있다. 만약 사용자가 하위 메서드의 소스 코드를 보지 못해도 상위 객체를 통해서 리모콘처럼 사용 가능하다. 단, 메서드가 override(재정의)가 되어 있어야 한다.

자동 형변환(upCasting), 강제 형변환(downCasting)

자식 객체를 부모 객체 변수에 담는 것을 자동 형변환(upCasting)이라 한다. 부모와 자식은 상속 관계이기 때문에 자동으로 형변환이 된다.

```
Animal ani = new Dog();
```

부모 객체 변수를 자식 객체로 바꾸는 것을 강제 형변환이라고 한다. 부모를 자식으로 바꾸는 일이니 자동으로 이루어지지 않고, 강제적으로 이루어지는 것이다.

```
((Cat)ani).night();
```

부모 클래스에 상태 정보가 있는 경우

부모 클래스에서 상태 정보가 선언되면 부모의 기능을 이어받는 자식 클래스도 생성자에서 부모의 생성자를 초기화 해줘야 한다. 그래야 객체를 생성할 때 부모 객체도 올바르게 생성된다.

```
public class Dog extends Animal {
    public Dog(String name, int weight) {
        super(name, weight);
    }
}
```

목차

- 재정의(Override)
- 상속 관계에서 객체 생성하는 2가지 방법

핵심

- Override(재정의)는 항상 하위 클래스의 메서드 실행을 도와준다.
- 상속 개념은 메모리 안에서 어떻게 작동되는 지 이해해야 한다.

재정의(Override)

재정의란 상속 관계에서 하위 클래스가 상위 클래스의 기능(메서드)을 수정하는 것을 말한다. 이 기능이 필요한 이유는 부모의 메서드가 항상 자식에게 필요한 메서드가 될 수는 없기 때문이다.

상속 관계에서 객체 생성하는 2가지 방법

1.자료형을 하위 클래스로 하는 법

```
Dog d = new Dog();
d.eat();
```

위와 같은 코드에서는 인스턴스 변수 d가 가르키는 인스턴스는 Dog지만 메모리가 확장(extends)됐기 때문에 상위 클래스인 Animal 영역까지 가르킨다. 그래서 메서드를 호출하면 Override(재정의)된 하위 클래스의 메서드가 호출된다.

2.자료형을 상위 클래스로 하는 법(upCasting)

```
Animal d = new Dog() // 객체는 Dog지만 자료형이 Animal로 형변환된다.(upCasting)
d.eat();
```

위와 같은 코드에서는 인스턴스 변수 d가 가르키는 인스턴스 Animal이다. 그러면 d.eat()을 호출하면 Animal의 eat() 호출되어야 한다. 하지만 하위 클래스가 eat()을 Override(재정의)를 했다면 하위 클래스의 eat()이 호출된다. 재밌는 점이 인스턴스 변수 d가 가르키는 객체는 Animal뿐이라서 하위 클래스인 Dog에 갈 수 없는데 Override(재정의)가 됐다면 하위 클래스에 접근할 수 있다. 이게 프로그램 실행될 때 정해지기 때문에 정적 바인딩이라 한다.

Override의 강력함

Override는 강력하다. 왜냐하면 메모리에 하위, 상위 메서드가 공존할 때 Override(재정의)가 되어 있으면 무조건 하위 메서드가 실행된다. 이는 하위 클래스의 구동 방식(소스 코드)를 모르더라도 상위 클래스를 통해 하위 클래스의 기능을 실행 시킬 수 있다. ~~아까 말한~~ 라모콘처럼

동적바인딩

Override는 동적바인딩이다. 동적바인딩은 프로그램이 실행될 때 호출되는 메서드의 주소가 결정되는 방법이다. 컴파일되는 시점에 호출되는 메서드가 정해지지 않고 프로그램이 시작하면서 호출되는 메서드가 즉석으로 결정된다. 이렇게 미리 정하지 않고 하다보니 속도가 느려지는 원인이 되기도 하지만 유용하다.

일전에 Method Overloading에 대해 배울 때 모르면 여기로 오버로딩은 정적 바인딩이라 말했다. 이제 둘의 차이가 확 느껴진다. 정적 바인딩은 프로그램 시작 전에 미리 호출될 메서드가 정해지는 것. 그래서 속도에 영향이 없다. 하지만 동적 바인딩은 프로그램이 시작하면서 호출될 메서드가 정해지는 것. 그래서 속도가 느릴 수 있다.

Java - Object Casting(객체 형 변환)

17

목차

- Object Casting(객체 형 변환)이란?
- 그럼 상속이 중요한 이유

핵심

- upCasting을 사용하면 부모 객체로 자식 객체를 자유자재로 활용할 수 있다.

Object Casting(객체 형 변환)이란?

객체 형 변환이란 상속 관계에 있는 클래스끼리 자료형을 바꾸는 기능을 말한다. 객체 형 변환에는 upCasting과 downCasting이 있다.

upCasting은 자식 객체를 부모 객체로 바꾸는 것을 말한다. 왜 바꿀까? 일전에 말한대로 만약 자식 객체의 기능(메서드)를 사용할 수 없는 상황일 때 부모 객체를 통해 자식 객체의 기능에 접근할 수 있다. (다만, Override가 된 전제 안에서)

그럼 downCasting은 언제 사용할까? 자식 객체의 메서드 중 override가 안된 메서드를 사용할 때 필요하다. override가 안된 메서드라면 부모 객체에서 접근할 수 없다. 그래서 upCasting한 부모 객체를 다시 자식 객체로 형 변환을 해서 사용해야 한다.

```
Animal ani = new Dog(); // upCasting
ani.eat() //컴파일 시점에서는 Animal의 eat()이지만 실행하면 Dog의 eat()으로 실행
Dog d = (Dog) ani; // downCasting
```

그럼 상속이 중요한 이유

상속이 중요한 이유는 무엇일까? 부모 객체로 자식 객체를 핸들링할 수 있다는 점은 알겠다. 그럼 이게 실생활에서 어떻게 사용될까? 바로 API를 쓸 때 중요하다. 대부분의 API는 상속으로 이루어져있기 때문에 이 원리를 안다면 똑똑하게 사용할 수 있다.

Java - message polymorphism(다형성)

18

목차

- message polymorphism(다형성)이란?
- 다형성 이론의 전제 조건
- 다형성 이론의 활용 방법 2가지

핵심

- 다형성은 부모 클래스를 적절하게 이용하는 것이다.

Message polymorphism(다형성)이란?

다형성이란 상속 관계에 있는 클래스에서 부모 클래스가 동일한 메세지로 자식 클래스들을 서로 다르게 동작시키는 원리이다. 같은 메서드를 호출해도 출력되는 결과 값이 달라진다. 객체 지향의 핵심이다.

예를 들어, 부모가 3명의 자식에게 꽃을 사오라고 말하면 3명의 자식은 각자 자신의 방법으로 꽃을 사올 것이다. 첫 째는 꽃 배달 업체를 이용한다. 둘 째는 직접 산다. 셋 째는 친구에게 부탁한다. 이처럼 꽃 사오라는 동일한 메세지지만 자식마다 동작이 다르다.

다형성 이론의 전제 조건

1. 상속 관계가 되어야 한다.
2. 객체 생성을 upCasting으로 해야 한다.
3. 자식 클래스에서 override가 되어야 한다.

다형성 이론의 활용 방법 2가지

1. 다형성 인수(데이터 이동)

메서드 선언부에서 매개변수 타입을 부모 객체로 해놓으면 서로 다른 자식 객체가 매개변수로 들어와도 함수가 작동된다.

```
Dog d = new Dog();
display(d); // 왈왈
```

```
Cat c = new Cat();
display(c); // 야옹
public static void display(Animal r) {
    r.eat();
}
```

2. 다형성 배열(서로 다른 객체를 담을 수 있다)

배열은 같은 자료형인 데이터들만 담을 수 있다. 하지만 다형성 이론을 이용하면 배열에도 서로 다른 자료형(객체)를 담을 수 있다. 상속 관계에서는 자식 객체를 부모 객체로 자유롭게 형 변환할 수 있기 때문이다. 그래서 부모 객체 배열을 만들고 각 element에 서로 다른 자식 객체를 담으면 된다.

```
Animal[] r = new Animal[2]; // Animal 객체만 담을 수 있는 배열
r[0] = new Dog(); // Animal 변수에 Dog 객체를 담는 것, upCasting
r[1] = new Cat();
r[0].eat();
r[1].eat();
```


목차

- Abstract class(추상 클래스)란?

핵심

- 추상 클래스는 다형성을 일부 보장하기 위해 사용한다. 마차-빈껍데기

Abstract class(추상 클래스)란?

추상 클래스는 추상 메서드가 있는 클래스를 의미한다. 추상 메서드는 구현부가 정의되어 있지 않은 불완전한 메서드를 의미한다. 그럼 왜 추상 메서드가 필요할까? 다형성을 보장하기 위해서이다. 다형성을 이용하면 부모 객체의 메서드는 실행될 일이 없다. 그렇기 때문에 구현부를 없애고 추상적으로(실제 기능하지 않는) 메서드로 정의해도 된다. 이렇게 하면 장점은 무엇일까? 자식 클래스에게 반드시 재정의의를 시킨다. 반드시 재정의의를 하면 다형성을 무조건 이용할 수 있다.

하지만 추상 클래스에는 추상 메서드만 있는 것이 아니라 구현 메서드도 존재한다. 그럼 이 구현 메서드는 자식 클래스 전체에 해당되는 메서드이다. 왜냐하면 부모 객체에 메서드를 정의한다는 건 자식 객체가 굳이 override하지 않아도 활용할 수 있는 메서드이기 때문이다.

추상 클래스는 구현 메서드를 구현할 수 있기 때문에 서로 기능이 비슷한 자식 클래스를 상속할 때 사용한다.

예시) TV 리모콘은 그 자체만으로는 기능이 없다. 반드시 TV와 함께 작동해야 한다. 그게 아니라면 다른 사물에는 사용할 수 없다. 즉, 그 자체만으로는 빈껍데기이다. 비어있다는 의미. 그러면 자식 클래스인 TV에서 리모콘이 명령하면 동작할 수 있도록 반드시 준비를 해야한다. 이 준비가 프로그래밍에서는 override를 하는 것이다.

-2019.12.31-

예시

```
// abstract class
public abstract class Animal {
```

```
public abstract void eat();
public void move() {
    System.out.println("Animal is moving...");
}
} // sub class
public class Cat extends Animal {
    @Override
    public void eat() {
        System.out.println("Sound of cat is ...");
    }
}
```

목차

- Abstract class와 interface의 공통점
- Abstract class와 interface의 차이점

핵심

- 둘 다 다형성을 보장하기 위해 만들어졌다.

Abstract class와 interface의 공통점

1. 다형성을 보장하기 위해 이용된다.
2. 자식 객체에서 override를 해야 한다.
3. upCasting으로 객체를 생성한다.
4. 부모(리모콘)의 역할을 한다.

Abstract class와 interface의 차이점

1. 구현 메서드가 있느냐 없느냐의 차이
2. 다중 상속이 불가능하냐 가능하냐의 차이
3. extends or implements 키워드 차이

목차

- 인터페이스와 추상클래스의 차이
- 상수가 올 수 있다.
- 인터페이스와 인터페이스의 상속 관계
- 다중 상속 관계 구현하기

핵심

- 인터페이스는 다형성을 100% 보장하기 위해 사용한다.
- 인터페이스는 자식 클래스를 100% 핸들링할 수 있다

인터페이스와 추상클래스의 차이

인터페이스는 모든 메서드가 추상 메서드인 클래스를 말한다. 추상 클래스와의 차이는 모든 메서드가 추상 메서드라는 점이다. 즉, 구현 메서드는 절대 정의할 수 없다. 그럼 왜? 인터페이스와 추상 클래스는 나뉘을까?

인터페이스는 서로 다른 기능을 가진 자식 클래스를 모아 다형성을 이용할 때 필요하다. 왜냐하면 구현 메서드가 없기 때문에 모든 자식 클래스에 적용되는 기능이 없다는 말이다. 각자 자식 클래스에서 부모의 메서드를 자기에 맞게 override하면 된다.

추상 클래스는 기능이 비슷한 자식 클래스를 모아 다형성을 이용할 때 필요하다. 추상 클래스는 추상 메서드를 가질 수 있지만 모든 메서드가 100% 추상 메서드는 아니다. 구현 메서드로 자식 클래스 전체에 해당하는 기능은 정의할 수 있다.

-2019.12.31- TV class 코드 추가

예시

```
public interface RemoCon {
    int MAXCH =100;
    int MINCH = 1;
    void chUp();
    void chDown();
    void internet();
}
```

```

public class TV implements RemoCon {
    int currCH = 70;
    @Override
    public void chUp() {
        if (currCH < RemoCon.MAXCH) {
            currCH++;
        } else {
            currCH = 1;
        }
        System.out.printf("TV 채널이 올라간다. : %d\n", currCH);
    }
}

```

상수가 올 수 있다.

인터페이스는 모든 메서드가 추상 메서드이기 때문에 객체 생성이 불가능하다. 그래서 상수를 사용하려면 인터페이스이름.상수이름 이렇게 표기해야한다. 이와 관련해서는 private생성자 <-여기로

```
RemoCon.MAXCH // 인터페이스 RemoCon의 상수 MAXCH
```

인터페이스와 인터페이스의 상속 관계

인터페이스끼리도 상속을 할 수 있다. 그 때 extends 키워드를 사용한다. 만약 다른 인터페이스를 상속받은 인터페이스가 있다고 하자. 이 인터페이스 implements하는 클래스는 모든 인터페이스의 메서드를 override 해야한다.

```

public interface A {
    public void x();
}
public interface B extends A {
    public void y();
}
public class C implements B {
    public void x() {
        재정의
    }
    public void y() {
        재정의
    }
}

```

```

    }
}
B r = new c();

```

다중 상속 관계 구현하기

자바에서 클래스는 한 번의 상속만 가능하다. 하지만 인터페이스가 있다면 다중 상속이 가능하다. 그럼 왜 **다중 상속**이 필요할까?

예를 들어, Dog라는 클래스를 설계하려고 할 때 Dog는 Animal도 되고, Pet도 되고, Robot도 가능하다. 그렇기 때문에 하나의 종류만 확정할 수 없다. 각 종류에 맞게 인터페이스를 구현하면 훨씬 정교하게 Dog를 설계할 수 있다.

```

public class Dog extends Animal implements Pet, Robots {
}

```

목차

- JDBC(Java Database connectivity) programming
- JDBC에 인터페이스를 활용하는 방법

핵심

- 인터페이스는 리모콘이다.

JDBC(Java Database connectivity) programming

JDBC programming은 자바를 이용해 데이터베이스를 조작하는 프로그래밍을 말한다. 데이터베이스 공급자는 여러 명이기 때문에 각 회사에서 만든 API는 다 다를 것이다. 기능은 유사해도 메서드의 이름이나 구현 방법은 다를 것이다. 그러면 자바에서 데이터베이스를 사용하려면 각 데이터베이스의 조작 방법을 전부 알아야 사용할 수 있다는 말이다.

근데 이 방법밖에 없을까? 아니다. 인터페이스를 활용하면 조금 더 편리하게 사용할 수 있다. 그럼 어떻게 인터페이스를 활용할까?

JDBC에 인터페이스를 활용하는 방법

자바 개발자가 각 공급사 별 데이터베이스 문법을 다 알고 쓸 수 없으니 할 수는 있지만 상당히 힘들기 때문이다. 개발자는 이런 걸 못 참는다. 자바에서 데이터베이스와 관련된 인터페이스(리모콘)를 만들어 각 공급사에 제공하는 건 어떨까? 각 공급사는 전달 받은 인터페이스에 맞게 override를 하고 클래스 파일을 만들 수 있다. 이 클래스 파일을 Driver라고 한다. 우리가 컴퓨터 설치할 때 말하는 드라이버 파일이 여기서 비롯된 것 같다.

이제 자바 개발자는 인터페이스로 모든 데이터베이스를 쉽게 관리할 수 있다. 비록 각 공급사 별 드라이버 클래스에서 구현되는 방식은 다르겠지만 개발자 입장에선 기능만 돌아가면 된다.

목차

- Object Class
- toString()

핵심

- Object는 부모의 부모의 부모다.

Object Class

Object 클래스는 모든 클래스의 root클래스이다 부모의 부모의 부모 기본적으로 하나의 클래스가 생성되면 자연스럽게 Object의 상속을 받는다. 즉, Object 클래스가 가진 기능을 활용할 수 있다. 많지는 않다

하지만 상속, 다형성 이론 배운 상황에서는 Object 클래스를 활용해서 조금 더 확장해서 사용할 수 있다.

처음 클래스가 생성되면 생략되는 코드가 3곳이 있다.

1. import java.lang.* -> Object 클래스가 있는 패키지
2. public class A extends Object { -> extends 키워드
3. public A() { super() } -> Object 생성자를 호출하는 생성자 메서드

toString()

모든 클래스는 Object 클래스를 상속받기 때문에 toString()을 사용할 수 있다. 만약 클래스에서 toString() 메서드를 정의하지 않으면 Object의 toString()을 사용하게 되는데, 이 때는 객체의 메모리 주소(address)가 나온다.

하지만 자체적으로 toString()을 정의했다면 그 클래스의 toString()이 구현된다. Object 것이 아닌.

목차

- Package(패키지)란?
- 외부에서 package 내부 클래스에 접근하는 방법
- Default 접근권한

핵심

- Package는 여러 클래스를 모아 놓은 가방이다.

Package(패키지)란?

패키지는 여러 클래스를 모아 놓은 단위를 말한다. 그럼 왜 패키지를 사용할까? 여러가지 이유가 있지만 간추려서 2가지가 있다.

1. 기능이 비슷한 클래스를 모아서 관리를 쉽게 하기 위함
2. 외부로 부터 접근을 막기 위함 (1번에 비해 빈도 수가 떨어진다.)

마치 가방이라고 생각하면 된다. 가방은 비슷한 물건끼리 넣을 수 있고, 외부로 부터 물체를 보호할 수 있다. 패키지도 이와 마찬가지로. 하지만 외부로 부터 접근을 막는 목적으로 자주 사용되지 않는다. 왜냐하면 클래스를 만드는 목적이 외부와 공유하기 위해 만든 목적일 가능성이 크기 때문이다. 그런데 공유가 안된다면 기능이 제대로 발휘하지 못하는 것이다.

또한, 패키지가 다르면 메서드의 이름이 같아도 된다. 오버로딩이 허용된다.

외부에서 package 내부 클래스에 접근하는 방법

- class full name을 알아야 한다. (root package부터 경로가 적인 이름)
- 접근 권한을 알아야 한다. (public 인지 private, default ...)
- import를 활용하면 편하다.

Default 접근권한

package가 명시적으로 선언되어 있고 클래스 앞에 접근자가 생략되면 public이 아니라 default 접근 권한을 가진다. 이 의미는 패키지 바깥에서 아무나 클래스에 접근할 수 없다는 말이다. 왜냐하면 패키지의 기능 중 하나가 외부로부터 클래스를 보호하는 역할도 있기 때문이다.

```
package kr.tpc;  
class A { // default 접근 권한을 가진다.  
  
}
```

목차

- API(Application Programming Interface)란?
- 자바의 기본 API
- String class

핵심

- API는 여러 클래스가 합쳐져 기능을 하는 도구이다

API(Application Programming Interface)란?

API란 프로그래밍 기초를 지나면 자주 만나게 되는 용어이다. 그럼 API는 무엇일까? 내 언어로 정리하자면 여러 클래스가 합쳐져 기능을 하는 도구이다. 프로그래밍은 절대 하나의 클래스, 프로그램으로만 이루어지지 않는다. 수십, 수백 개의 클래스들끼리 연결되어 하나의 프로그램이 만들어진다.

이 때 여러 클래스들은 뭉쳐져서 하나의 기능을 수행할 수 있다. 마치 레고 조각이 뭉쳐져서 큰 성이 만들어지는 것처럼 말이다. 그런데 이 API는 내가 만들 수도 있지만 누군가 뛰어난 기능으로 만들었을 수도 있다. 다른 사람이 만든 걸 굳이 내가 만들 필요는 없다. 그 사람꼐 쓰면 된다.

자바의 기본 API

자바에도 물론 API가 있다. JDK 속에 설치되어 있다. 그래서 자바의 기본 API도 충분히 유용하고 잘 사용할 수 있어야 한다. 경로는 jrt-fs.jar에 있으며 2단계 구조로 패키지가 구성되어 있다. java.lang, java.util, java.sql, ... 등. 기본 API를 사용하려면 패키지 경로를 통해 접근해 사용하면 된다.

String class

자바에서 문자열을 담는 자료형 String은 객체이다. 객체라는 의미는 String을 사용하려면 메모리에 String을 생성해야 한다. 객체를 생성할 땐 new 연산자와 생성자 메서드를 사용한다.

하지만 우리는 String을 선언할 때 new와 생성자 연산자를 사용하지 않는다. 무슨 일일까?

String을 생성하는 방법은 2가지가 있기 때문이다.

1. 일반적인 객체 생성처럼 new 연산자와 생성자 메서드를 사용하는 방법.
2. 문자열 상수로 생성하는 방법이다. ** 이 둘의 차이는 무엇일까?**

이 둘의 차이는 객체의 재활용 여부이다.

첫 번째 방법은 heap area에 생성된다. 그리고 만약 같은 값의 문자열이 또 만들어지면 이전 객체의 주소를 재활용하지 않고 새롭게 heap area에 객체를 생성한다.

두 번째 방법은 literal pool에 생성된다. 이곳은 문자열 상수가 생성되는 메모리 영역이다. 특징은 주소를 재활용한다. 그래서 만약 같은 값의 문자열이 또 만들어지면 객체를 따로 생성하는 것이 아니라 이전 객체 주소를 재활용한다. 즉, 객체 주소가 같다.

예 제

```
String str1 = new String("hello");
String str2 = new String("hello");
if (str1.equals(str2)) {
    System.out.println("YES");
} else {
    System.out.println("NO");
}
```

목차

- 나만의 API 만들기
- IntArray 만들기

핵심

- API는 이해하고 사용하기에 조금 더 편하다.

나만의 API 만들기

우리가 배열을 사용하고 싶으면 자바에서 기본으로 제공하는 배열을 사용하면 된다. 하지만 배열을 사용하기 위한 기본 문법은 다 알고 있어야 한다. 그걸 알지 못하면 사용하지 못한다.

이 때 배열과 비슷한 기능을 하는 API를 만들면 조금 더 직관적으로 배열을 사용할 수 있다. 메서드를 활용하기 때문에 조금 더 기능에 부합하게 메서드 이름을 만들 수 있다.

IntArray 만들기

```
// 정수형 배열 기능하는 API만들기
public class IntArray {
    private int count;
    private int[] arr;
    // 배열 생성
    public IntArray() {
        this(10);
    }
    public IntArray(int init) {
        arr = new int[init];
    }
    // 배열에 값 추가 : add
    public void add(int data) {
        System.out.println(count);
        arr[count++] = data;
    }
}
```

```
// 배열에 값 얻기 : get
public int get(int index) {
    return arr[index];
}
// 배열의 길이 구하기 : size
public int size() {
    return count;
}
}
```

목차

- ObjectArray
- 실습
- 이미 ArrayList는 있다.

핵심

- Object를 이용하면 upCasting, downCasting을 생각해라

ObjectArray

지난 시간에 만든 intArray는 int 자료형 데이터만 넣을 수 있다. 이러면 아쉬움이 남는다. 그럼 모든 자료형을 담을 수 있는 배열을 어떻게 만들 수 있을까? 모든 자료형을 upCasting으로 받을 수 있는 Object 배열로 만들면 어떨까?

또한, Object 객체로 배열을 만든다는 것은 데이터가 들어갈 때는 upCasting으로 나올 때는 downCasting으로 나온다는 의미다. 이걸 자유자재로 사용할 수 있어야 한다.

실습

```
public class ObjectArray {
    private int count;
    private Object[] arr;
    // 배열 생성
    public ObjectArray() {
        this(10);
    }
    public ObjectArray(int init) {
        arr = new Object[init];
    }
    // 배열에 값 추가 : add
    public void add(Object data) {
        arr[count++] = data;
    }
}
```

```
// 배열에 값 얻기 : get
public Object get(int index) {
    return arr[index];
}
// 배열의 길이 구하기 : size
public int size() {
    return count;
}
}
```

이미 ArrayList는 있다.

우리가 구현한 ObjectArray보다 기능이 더 많은 ArrayList가 자바에선 기본 API로 제공하고 있다. 똑같이 Object 배열로 이루어져있지만 크기가 제약이 없기 때문에 더 유용하다. 하지만 어떤 구조 이루어져 있는지 이해하는 것도 중요하다.

목차

- ArrayList

핵심

- 제네릭으로 자료형을 정하면 downCasting 안해도 된다.

ArrayList

ArrayList는 Array의 단점을 보완한 API다. 길이의 제약이 없기 때문에 훨씬 편리하게 사용할 수 있다. ArrayList는 원래 Object[]로 이루어져 있다. 그래서 원소가 들어갈 때는 upCasting을 하고 꺼내서 사용할 때는 downCasting을 해야 한다.

하지만 <> : generic으로 자료형을 지정하면 Object[]에서 지정한 자료형으로 바뀐다. 그러면 원소를 꺼낼 때 downCasting할 필요가 없다.

만약 제네릭을 사용하지 않으면 코드에 경고창이 뜬다.

```
public static void main(String[] args) {
    ArrayList<BookDTO> arr = new ArrayList<>();
    arr.add(new BookDTO("java", 15000, "google"));
    arr.add(new BookDTO("python", 20000, "amazon"));
    for (BookDTO obj : arr) {
        System.out.println(obj.toString());
    }
}
```

목차

- Wrapper Class(포장 클래스)란?
- Boxing & UnBoxing

핵심

- 자바는 자동으로 포장도 해주고 풀어 주기도 한다.

Wrapper Class(포장 클래스)란?

Wrapper Class는 기본 자료형을 객체 자료형으로 포장하는 클래스이다. 즉, 바꿔 준다는 의미이다. 왜 바꿔야 할까? 기본 자료형을 객체 자료형으로 만들면 상속, 다형성 등 객체를 유용하게 사용할 수 있다.

예시

- int -> Integer
- float -> Float
- char -> Character
- boolean -> Boolean

Boxing & UnBoxing

만약, Object[]에 int를 넣고 싶으면 어떻게 넣을 수 있을까? 원래는 못 넣는다. 자료형이 다르기 때문이다. 하지만 int를 Integer로 포장해서 객체로 만들면 Object[]에 들어갈 수 있다. 그렇다면 아래 코드처럼 써야 한다.

```
Object[] obj = new Object[2];
obj[0] = new Integer(10);
obj[1] = new Integer(10);
```


하지만 자바에서는 int로 적어도 자동으로 Integer로 포장(Boxing)해서 배열에 넣어 준다.


```
java
obj[0] = 1; // 1을 자동으로 Boxing(포장)해서 Integer로 바꿔 준다.
```


```

```
""

```

그래서 Boxing은 int 자료형을 자동으로 Integer로 바꿔 주는 기능을 말한다.<br>

Unboxing은 Integer 객체를 자동으로 int 자료형으로 바꿔 주는 기능이다. (포장을 풀다)<br>

```
""java
```

```
Integer i = 3; // Boxing
```

```
int i = new Integer(3); // unboxing
```

## 목차

- 객체를 표현하는 3가지 방법
- Gson API 다운로드
- Gson API로 JSON 다루기

## 핵심

- Json으로 데이터 정보를 편리하게 주고받을 수 있다.

## 객체를 표현하는 3가지 방법

서로 다른 프로그램끼리 객체를 주고 받을 땐 문자열 형태로 주고 받는 것이 편하다. 왜냐하면 문자열은 각 언어 별 형태가 거의 비슷하기 때문이다. "" or " 문자열 데이터를 주고 받는 형식은 크게 3가지가 있다.

### 1. 의미는 없고 구분은 되어 있는 경우

```
String textMember = "홍길동,광주,010-1111-1111#나길동,서울,010-3212-3213"
```

데이터 표현은 단순하지만, 의미가 명확하지 않다. 우리는 사람이기 때문에 저 데이터가 어떤 기준으로 나뉘어졌는지 알 수 있지만, 컴퓨터는 정확히 파악하기 어렵다. 또한, 데이터를 분리하는 과정, 전 처리 과정이 복잡해지는 단점이 있다.

### 2. 의미도 있고 구분도 되어 있는 경우 : XML(Extensible Markup Language)

```
<member>
 <name>홍길동</name>
 <address>광주</address>
 <phone>010-2222-3333</phone>
</member>
```

확실히 이전 버전보다 의미도 있고 데이터 간 구분도 되어 있다. 태그로 구분되어 있기 때문에 데이터 처리는 쉽다. 하지만 데이터의 크기가 커지는 단점이 있다.

### 3. 의미도 있고 구분도 되어 있는 경우 : JSON(JavaScript Object Notation)

- 한 개의 JSON : {"name" : "홍길동", "address" : "광주"}
- 여러 개의 JSON(배열)

```
String jsonMembers = "[{\"name\" : \"홍길동\", \"address\" : \"광주\"},
 {\"name\" : \"나길동\", \"address\" : \"서울\"}]\";
```

- <http://json.org> 참고

XML보다 간결하다. 하지만 표현이 제한적인 단점이 있다. 하지만 간결함이 가장 큰 무기가 되서 최근엔 다양하게 사용되는 표기법이다.

### Gson API 다운로드

- <https://mvnrepository.com/> 에서 Gson 검색 후 API 다운로드
- 2.8.6 버전 다운로드(jar)

Version	Vulnerabilities	Repository	Usages	Date
2.9.x				
2.9.0		Central	1,185	Feb, 2022
2.8.9		Central	1,667	Oct, 2021
2.8.8	1 vulnerability	Central	780	Aug, 2021
2.8.7	1 vulnerability	Central	753	May, 2021
2.8.6	1 vulnerability	Central	3,687	Oct, 2019
2.8.5	1 vulnerability	Central	4,587	May, 2018
2.8.x				
2.8.4	1 vulnerability	Central	293	May, 2018



## Gson » 2.8.6

Gson

License	Apache 2.0
Categories	JSON Libraries
Date	(Oct 04, 2019)
Files	jar (234 KB) View All
Repositories	Central WSO2 Public
Ranking	#11 in MvnRepository (See Top Artifacts)

## Gson API로 JSON 다루기

```
// Gson으로 객체를 json형식 문자열로 바꾸기
BookDTO b = new BookDTO("java", 5000, "한빛미디어");
Gson g = new Gson();
String json = g.toJson(b);
System.out.println(json);
// Gson으로 json을 객체로 바꾸기
BookDTO book = g.fromJson(json, BookDTO.class);
System.out.println(book.toString());
// Gson으로 List를 Json으로 바꾸기
List<BookDTO> list = new ArrayList<>();
list.add(new BookDTO("python", 1000, "note"));
list.add(new BookDTO("c", 10000, "please"));
String json2 = g.toJson(list);
System.out.println(json2);
// Gson을 활용하여 Json을 List로 바꾸기
List<BookDTO> list2 =
 g.fromJson(json2, new TypeToken<List<BookDTO>>().getType());
for (BookDTO v : list2) {
 System.out.println(v);
}
```

Json을 List를 바꿀 땐 객체로 바꿀 때보다 복잡하다. 그 이유는 객체는 하나의 자료형이기 때

문에 BookDTO.class라고 적으면 된다. 하지만 배열은 속에 객체라는 자료형이 ~~있기 때문에~~ 있어서 총 2개가 된다. 그래서 자바에게 확실하게 알려줘야 한다.

## 목차

- Json-java(org.json) 사용법
- org.json API 다운로드
- json 파일 읽어서 값 추출하는 실습

## 핵심

- org.json API를 사용하면 객체를 바로 만들어 사용할 수 있다.

## org.json(JSON-java) 사용법

json을 만드는 다른 API이다. Gson과 다른 점은 객체를 바로 만들어서 사용할 수 있다. JSONObject라는 객체가 json형태로 객체를 생성한다.

```
// 1. student json 객체 만들기
JSONObject student = new JSONObject();
student.put("name", "son");
student.put("phone", "010-1111-1111");
student.put("address", "광주");
JSONObject student2 = new JSONObject();
student2.put("name", "jun");
student2.put("phone", "010-2222-2222");
student2.put("address", "서울");
// 2. students json 배열 만들기
JSONArray students = new JSONArray();
students.put(student);
students.put(student2);
// 3. object json 객체 만들기
JSONObject object = new JSONObject();
object.put("students", students);
System.out.println(object.toString(2));
```



## org.json API 다운로드

- <https://mvnrepository.com/> 에서 org.json 검색 후 API 다운로드
- json-20220320.jar 파일 다운로드

**MVNREPOSITORY**

**Repository**

- Central 19
- Sonatype 13
- Spring Plugins 6
- Liferay Public 5
- ICM 4
- Spring Lib M 4
- Mulesoft 2
- Spring Lib Release 2

**Found 39 results**

Sort: **relevance** | popular | newest

 1. **JSON In Java** 4,862 usages [JSON](#)

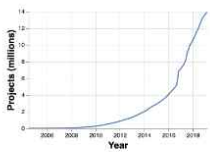
org.json » json

JSON is a light-weight, language independent, data interchange format. See <http://www.JSON.org/> The files in this package implement JSON encoders/decoders in Java. It also includes the capability to convert between JSON and XML, HTTP headers, Cookies, and CDL. This is a reference implementation. There is a large number of JSON packages in Java. Perhaps someday the Java community will standardize on one. Until then, choose carefully. The ...

Version	Vulnerabilities	Repository	Usages	Date
20220320		Central	172	Mar, 2022
20211205		Central	172	Dec, 2021
20210307		Central	315	Mar, 2021
20201115		Central	224	Nov, 2020
20200518		Central	246	May, 2020

**MVNREPOSITORY**


**Indexed Artifacts (28.9M)**



**Popular Categories**

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries

Home » org.json » json » 20220320

 **JSON In Java » 20220320**

JSON is a light-weight, language independent, data interchange format. See <http://www.JSON.org/> The files in this package implement JSON encoders/decoders in Java. It also includes the capability to convert between JSON and XML, HTTP headers, Cookies, and CDL. This is a reference implementation. There is a large number of JSON packages in Java. Perhaps someday the Java community will standardize on one. Until then, choose carefully. The ...

License	JSON
Categories	JSON Libraries
HomePage	<a href="https://github.com/douglascrockford/JSON-java">https://github.com/douglascrockford/JSON-java</a>
Date	(Mar 20, 2022)
Files	<a href="#">pom (7 KB)</a> <a href="#">bundle (69 KB)</a> <a href="#">View All</a>

## json 파일 읽어서 값 추출하는 실습

```
// 1. json 파일에서 json 데이터 불러오기
String src = "info.json";
InputStream is = Project01_C.class.getResourceAsStream(src);
if (is == null) {
```

```

 throw new NullPointerException("Cannot find resource file");
 }
 // 2. 읽어온 json 데이터를 메모리에 로딩하기 : JSONTokener 객체 사용
 JSONTokener tokenizer = new JSONTokener(is);
 // 3. JSONTokener 객체를 JSONObject 객체로 만들기
 JSONObject students = new JSONObject(tokenizer);
 // 4. Json 객체에 있던 jsonArray 꺼내기
 JSONArray studentArr = students.getJSONArray("students");
 // 5. jsonArray 안에 있는 student 객체 출력하기.
 for (int i = 0; i < studentArr.length(); i++) {
 JSONObject student = studentArr.getJSONObject(i);
 System.out.println(student.get("address"));
 }

```

## Gson과 org.json의 차이

Gson과 org.json의 차이는 객체를 직접 만드느냐의 차이이다. Gson은 내가 만든 객체를 json으로 바꾸고, org.json은 객체를 통해 바로 json객체를 만든다. org.json은 javascript 객체를 바로 java에서 만드는 느낌.

-2019.12.30 추가-

## JsonObject에서 값을 꺼낼 땐 자료형이 Object?

JsonObject의 value 값은 Object 자료형인 것 같다?

```

// 객체 생성
JSONObject jsonObject = new JSONObject();
jsonObject.put("name" , "gildong");
jsonObject.put("address" , "광주");
// 배열 생성
JSONArray arr = new JSONArray();
arr.put(jsonObject);
for (int i = 0; i < arr.length(); i++) {
 JSONObject obj = arr.getJSONObject(i); // 배열 속 객체
 // 객체 속 "address"의 값을 Object로 upCasting 해야함
 Object string = obj.get("address");
 System.out.println(string.toString());
}

```

## 목차

- Geocoding이란
- 위도, 경도 추출하는 실습(Geocoding)
- 전체 코드

## 핵심

- Geocoding이란 주소를 입력하면 위도와 경도를 얻는 코딩을 말한다.

## Geocoding이란?

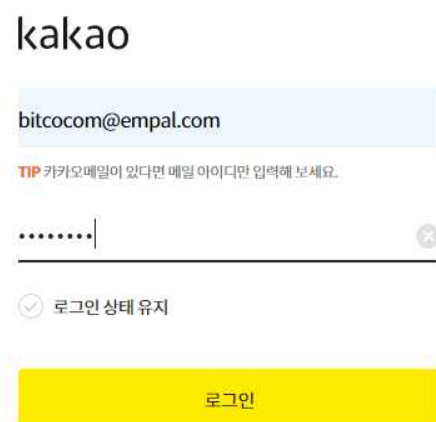
Geocoding이란 주소를 입력하면 위도와 경도를 얻는 코딩을 말한다. 위도와 경도를 얻기 위해선 위도와 경도를 제공하는 웹사이트에 요청을 해야 한다.

## 요청 시에 필요한 정보가

- 요청하는 웹사이트 URL
- client ID : 웹사이트에서 나를 식별하는 ID
- client secret : Password

## 위도, 경도 추출하는 실습(Geocoding)

- 카카오 로컬API URL 가져오기 : <https://apis.map.kakao.com/> -> 로컬API -> 주소 검색하기 -> APP KEY 발급하기(로그인 필수) -> 애플리케이션 추가하기



전체 애플리케이션 (9)



## 애플리케이션 추가하기

앱 아이콘

이미지  
업로드

파일 선택

JPG, GIF, PNG  
권장 사이즈 128px, 최대 250KB

앱 이름

ERICAJAVA

사업자명

한양대학교ERICA

- 입력된 정보는 사용자가 카카오 로그인을 할 때 표시됩니다.
- 정보가 정확하지 않은 경우 서비스 이용이 제한될 수 있습니다.

취소

저장



## 앱 키

네이티브 앱 키

REST API 키

JavaScript 키

Admin 키

- BufferedReader : String 정보를 읽어 온다.
- InputStreamReader : 사용자에게 입력 받은 값은 byte이기에 BufferedReader가 읽을 수 있게 도와주는 객체

URLConnection : 웹사이트(서버)와 프로그램을 연결 시켜주는 객체

```
URLConnection con=(URLConnection) url.openConnection();
con.setRequestMethod("GET");
con.setRequestProperty("Authorization", "KakaoAK 80519c0647680c4a850971314626217");
```

서버에서 보낸 코드 한 줄씩 읽는 코드

```
String line;
StringBuffer response = new StringBuffer();
```

```
while ((line = br.readLine())!=null) {
 response.append(line);
}
```

## 예 시

### • Request

```
curl -v -X GET "https://dapi.kakao.com/v2/local/search/address.json" \
-H "Authorization: KakaoAK ${REST_API_KEY}" \
--data-urlencode "query=전북 삼성동 100"
```

### • Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
 "meta": {
 "total_count": 4,
 "pageable_count": 4,
 "is_end": true
 },
 "documents": [
 {
 "address_name": "전북 익산시 부송동 100",
 "y": "35.97664845766847",
 "x": "126.99597295767953",
 "address_type": "REGION_ADDR",
 "address": {
 "address_name": "전북 익산시 부송동 100",
 "region_1depth_name": "전북",
 "region_2depth_name": "익산시",
 "region_3depth_name": "부송동",
 "region_3depth_h_name": "삼성동",
 "h_code": "4514069000",
 "b_code": "4514013400",
 "mountain_yn": "N",
 "main_address_no": "100",
 "sub_address_no": "",
 "x": "126.99597295767953",
```

```

 "y": "35.97664845766847"
 },
 "road_address": {
 "address_name": "전북 익산시 망산길 11-17",
 "region_1depth_name": "전북",
 "region_2depth_name": "익산시",
 "region_3depth_name": "부송동",
 "road_name": "망산길",
 "underground_yn": "N",
 "main_building_no": "11",
 "sub_building_no": "17",
 "building_name": "",
 "zone_no": "54547",
 "y": "35.976749396987046",
 "x": "126.99599512792346"
 }
},
...
]
}

```

## 소스 코드

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.*;

import org.json.JSONArray;
import org.json.JSONObject;
import org.json.JSONTokener;

public class ERICA02 {
 public static void main(String[] args) {
 BufferedReader io=new BufferedReader(new InputStreamReader(System.in));
 try {
 System.out.print(" 주소를입력 하세요:");
 String address=io.readLine();
 String addr=URLEncoder.encode(address, "UTF-8");
 String reqUrl="https://dapi.kakao.com/v2/local/search/address.json?query="+addr

```

```

URL url=new URL(reqUrl);
URLConnection con=(URLConnection) url.openConnection();
con.setRequestMethod("GET");
con.setRequestProperty("Authorization", "KakaoAK
80519c0647680e4e850971314626277");

BufferedReader br
int responseCode=con.getResponseCode(); // 200
if(responseCode==200) {
 br=new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
}else {
 br=new BufferedReader(new InputStreamReader(con.getErrorStream()));
}
String line
StringBuffer response=new StringBuffer(); // JSON
while((line=br.readLine())!=null) {
 response.append(line);
}
br.close();
//System.out.println(response.toString());

JSONTokener tokenener=new JSONTokener(response.toString());
JSONObject object=new JSONObject(tokenener);
//System.out.println(object.toString());

JSONArray arr=object.getJSONArray("documents");
JSONObject subJobj =(JSONObject) arr.get(0);
JSONObject roadAddress =(JSONObject)subJobj.get("road_address");
String value=null
if(roadAddress == null){
 JSONObject subsubJobj = (JSONObject) subJobj.get("address");
 value = (String) subsubJobj.get("address_name");
}else{
 value = (String) roadAddress.get("address_name");
}
if(value.equals("") || value==null){
 subJobj = (JSONObject) arr.get(1);
 subJobj = (JSONObject) subJobj.get("address");
 value =(String) subJobj.get("address_name");
}

```

```

 }
 String x = (String) roadAddress.get("x");
 String y = (String) roadAddress.get("y");
 System.out.print(value);
 String building_name=(String) roadAddress.get("building_name");
 System.out.println(" "+building_name);
 System.out.println(x+":"+y);
} catch (Exception e) {
 e.printStackTrace();
}
}
}

```

주소를 입력하세요: **경기도 안산시 상록구 한양대학로 55**  
**경기 안산시 상록구 한양대학로 55**  
**126.834393833262:37.2980047345996**



## 예시

## • Request

```
curl -v -X GET "https://dapi.kakao.com/v3/search/book?target=title" \
--data-urlencode "query=미움받을 용기" \
-H "Authorization: KakaoAK ${REST_API_KEY}"
```

## • Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
 "meta": {
 "is_end": true,
 "pageable_count": 9,
 "total_count": 10
 },
 "documents": [
 {
 "authors": [
 "기시미 이치로",
 "고가 후미타케"
],
 "contents": "인간은 변할 수 있고, 누구나 행복해 질 수 있다. 단 그러기 위해서는 ‘용기’가 필요하다고 말한 철학자가 있다. 바로 프로이트, 융과 함께 ‘심리학의 3대 거장’으로 일컬어지고 있는 알프레드 아들러다. 『미움받을 용기』는 아들러 심리학에 관한 일본의 1인자 철학자 기시미 이치로와 베스트셀러 작가인 고가 후미타케의 저서로, 아들러의 심리학을 ‘대화체’로 쉽고 맛깔나게 정리하고 있다. 아들러 심리학을 공부한 철학자와 세상에 부정적이고 열등감 많은",
 "datetime": "2014-11-17T00:00:00.000+09:00",
 "isbn": "8996991341 9788996991342",
 "price": 14900,
 "publisher": "인플루엔셜",
 "sale_price": 13410,
 "status": "정상판매",
 "t h u m b n a i l " :
 }
]
}
```

```

"https://search1.kakaocdn.net/thumb/R120x174.q85/?fname=http%3A%2F%2Ft1.daumcdn.net%2Flib
ook%2Fimage%2F1467038",
 "title": "미움받을 용기",
 "translators": [
 "전경아"
],
 "u": "r",
 "l": "":
"https://search.daum.net/search?w=bookpage&bookId=1467038&q=%EB%AF%B8%EC%9B%80%
EB%B0%9B%EC%9D%84+%EC%9A%A9%EA%B8%B0"
},
...
]
}

```

## 소스코드

```

import java.io.BufferedReader;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;

import org.json.JSONArray;
import org.json.JSONObject;
import org.json.JSONTokener;

public class ERICA01 {
 public static void main(String[] args) {
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
 try {
 System.out.print("책 제목:");
 String title=br.readLine();
 getIsbnImage(title);
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}

```

```

 }
}
private static void getIsbnImage(String title) {
 try {
 String openApi="https://dapi.kakao.com/v3/search/book?target=title&query="
 + URLEncoder.encode(title, "UTF-8");
 URL url=new URL(openApi);
 HttpURLConnection con=(HttpURLConnection) url.openConnection();
 con.setRequestMethod("GET");
 con.setRequestProperty("Authorization", "KakaoAK
 0519c0647680e4e850971314626274");
 int responseCode=con.getResponseCode();
 BufferedReader br1=null;
 if(responseCode==200) {
 br1=new BufferedReader(new InputStreamReader(con.getInputStream(),"UTF-8"));
 }else {
 br1=new BufferedReader(new InputStreamReader(con.getErrorStream()));
 }

 String inputLine;
 StringBuffer response=new StringBuffer();
 while((inputLine=br1.readLine())!=null) {
 response.append(inputLine);
 }
 br1.close();
 //System.out.println(response.toString());
 JSONTokener tokenizer=new JSONTokener(response.toString());
 JSONObject object=new JSONObject(tokenizer);
 String img=null;
 JSONArray documents=object.getJSONArray("documents");
 for(int i=0; i<documents.length();i++) {
 JSONObject book=(JSONObject)documents.get(i);
 System.out.print(book.get("title")+"\t");
 System.out.print(book.get("datetime").toString().split("T")[0]+"\t");
 System.out.print(book.get("price")+"\t");
 System.out.print(book.get("isbn")+"\t");
 System.out.print(book.get("thumbnail")+"\t");
 System.out.println(book.get("publisher")+"\t");
 }
 }
}

```

```

 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}

```

책제목: **미움받을 용기**

미움받을 용기	2014-11-17	14900	8996991341	9788996991342	h
미움받을 용기(특별 합본호)(한정판)(양장본 Hardcover)	2018-03-02	19800	1		
작가의 문장수업(미움받을 용기 고가 후미타케)	2015-09-02	3600	896952093		
미움받을 용기 2	2016-05-02	14900	1186560126	9791186560129	h
아들러 심리학을 읽는 밤	2015-01-15	13800	8952230647	9788952230645	
미움받을 용기	2014-12-11	11900	9788996991359	<a href="https://search1.k">https://search1.k</a>	
아들러 선생님 고민있어요!(양장본 Hardcover)	2018-08-31	12000	895582465		
피토염적용기 被討厭的勇氣 미움받을 용기 (日本原作)	2017-09-01	33500	7		
엄마를 위한 미움받을 용기	2015-06-22	14000	1157950531	9791157950539	
오늘을 살아갈 용기 아들러 심리학	2015-10-20	16000	1157790380	979115	