

생각하라! 표현하라! 코딩하라!

Inflearn

AVATPC  
생각 표현 코딩

## JavaThinkingPresentationCoding

### ☞ Check Point

- 클래스에 대한 정확한 이해가 필요한 사람
- 객체지향 개념의 이해가 필요한 사람
- 다형성 개념을 활용하고 싶은 사람
- 다양한 API를 활용해보고 싶은 사람
- JSON, XML Data를 핸들링 해보고 싶은 사람
- Open API를 활용해서 프로젝트를 해보고 싶은 사람

- 이해하기 쉬운 그림으로 설명
- 제작 실무교재 제공
- github 소스 제공

JavaTPC  
박매일 강사

### 강의경력

(현)한국전력공사 코딩 위탁 교육  
- 2019 In-House 코딩 위탁교육  
- 디지털변환 관련 기초 코딩(Python) 교육  
(현)광주소프트웨어마이스터고 산학협력교사  
- IoT실무프로젝트, 리눅스시스템프로그래밍  
조선대학교 산업기술융합대학원 외래강사  
한국산업인력공단 NCS기업활용 컨설턴트  
ICU 정보통신교육원, 조선대, 호남대 강의  
리눅스마스터 공인강사, 정보처리기사  
직업능력개발훈련교사(정보처리, 멀티미디어, 사무자동화)  
2015'창업경진대회 중소기업청장상(우수상)  
2016,17' 한전KDN 에너지SW경진대회수상

## PART -1

## 목차

- 수업 1 : 자바개발환경설치(JDK12, Eclipse IDE)
- 수업 2 : Java구동 방식
- 수업 3 : 이것만 알자! 프로그래밍의 3대 요소(변수,자료형,할당)
- 수업 4 : 관계를 이해하라(V. D. A)
- 수업 5 : 관계를 이해하라(실습)
- 수업 6 : 데이터를 이동하라(변수 VS 배열의 관계)
- 수업 7 : 데이터를 이동하라(실습)
- 수업 8 : 메서드는 변수다(변수 VS 메서드의 관계)
- 수업 9 : 메서드는 변수다(실습)
- 수업10 : JVM의 메모리 모델(JVM이 사용하는 메모리 영역 - 4가지)
- 수업11 : JVM의 메모리 모델(실습)
- 수업12 : 기본자료형(PDT) VS 사용자정의자료형(UDDT)
- 수업13 : 객체가 메모리에 어떻게 만들어지나! 객체생성과정(new, 생성자 메서드, this)
- 수업14 : 객체가 메모리에 어떻게 만들어지나(실습)
- 수업15 : private 생성자도 있어요?(static과 관계)
- 수업16 : 애매하다! class, object, instance 상호관계
- 수업17 : 잘 설계된 클래스(Model : DTO, DAO, Utility)
- 수업18 : 잘 설계된 클래스(실습)
- 수업19 : 메서드의 오버로딩(Overloading)
- 수업20 : 동일한 구조,이질적인 구조(배열 VS 클래스의 관계)
- 수업21 : 학습정리(우리가 사용하는 클래스의 종류들)

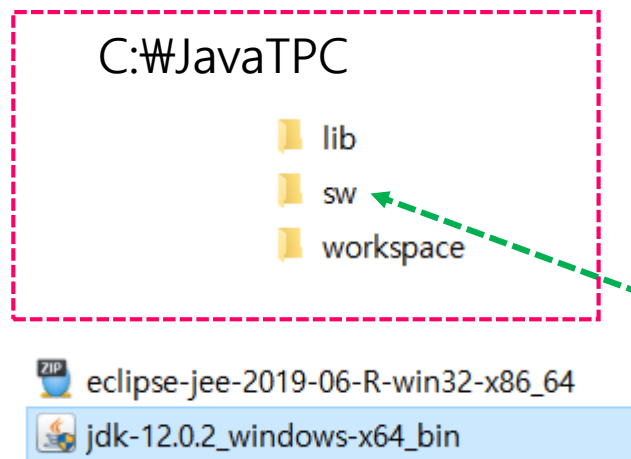
## 1. JavaSE 개발환경 구축(JDK 설치)

→Java 개발환경(플랫폼) → JavaSE, JavaEE, JavaME, Java **Android** 개발환경

자바를 설치한다는 것은 자바 JDK를 설치한다는 의미입니다. **JDK**는 Java Development Kit의 줄임 말로, JDK에는 개발하는데 필요한 라이브러리와 플랫폼이 포함되어 있습니다. 자바 프로그램을 실행하기 위해서는 자바 실행환경 **JRE**(Java Runtime Environment)만 있으면 되지만, 프로그램을 개발하기 위해서는 반드시 JDK가 필요합니다.

→자바는 오라클 사이트에서 다운로드 할 수 있습니다.

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>



Java SE Development Kit 12.0.2		
You must accept the <a href="#">Oracle Technology Network License Agreement for Oracle Java SE</a> to download this software.		
Thank you for accepting the Oracle Technology Network License Agreement for Oracle Java SE; you may now download this software.		
Product / File Description	File Size	Download
Linux	155.14 MB	<a href="#">jdk-12.0.2_linux-x64_bin.deb</a>
Linux	162.79 MB	<a href="#">jdk-12.0.2_linux-x64_bin.rpm</a>
Linux	181.68 MB	<a href="#">jdk-12.0.2_linux-x64_bin.tar.gz</a>
macOS	173.63 MB	<a href="#">jdk-12.0.2_osx-x64_bin.dmg</a>
macOS	173.98 MB	<a href="#">jdk-12.0.2_osx-x64_bin.tar.gz</a>
Windows	158.63 MB	<a href="#">jdk-12.0.2_windows-x64_bin.exe</a>
Windows	179.57 MB	<a href="#">jdk-12.0.2_windows-x64_bin.zip</a>

→JDK를 설치하면 **JavaSE개발환경이 구축** 되었다.  
내 PC에는 이런 것들이 설치됨

TOOL(컴파일러, 실행 명령어 들...)  
API(미리 만들어 놓은 class 들...)  
JVM(자바가상머신, 자바를 실행해주는 엔진)

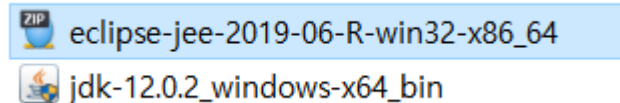
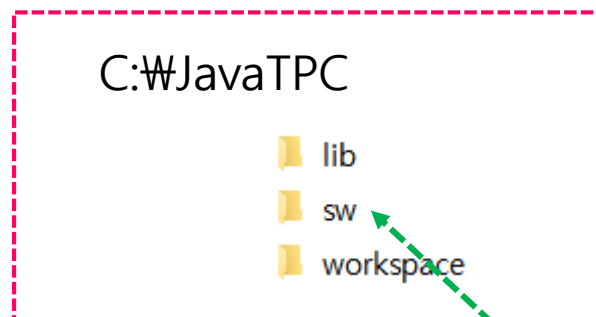
➤ 개발해볼까?

## 2. Eclipse IDE 다운로드

자바 프로그래밍을 하는데 필요한 통합 개발 환경(IDE : Integrated Development Environment)인 이클립스를 설치하도록 하겠습니다.

→이클립스 사이트에 접속합니다.

<https://www.eclipse.org/downloads/packages/>

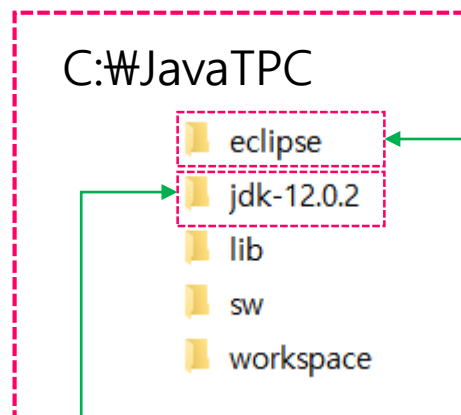
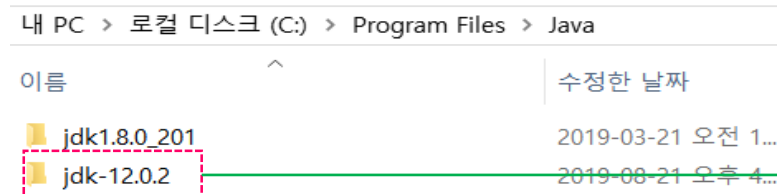
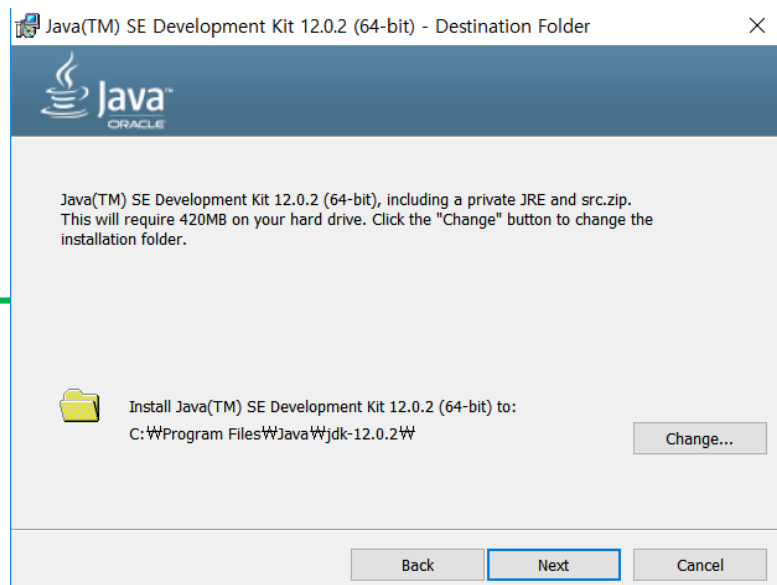


## 3. Java, Eclipse IDE 설치하기

eclipse-jee-2019-06-R-win32-x86\_64

jdk-12.0.2\_windows-x64\_bin

install

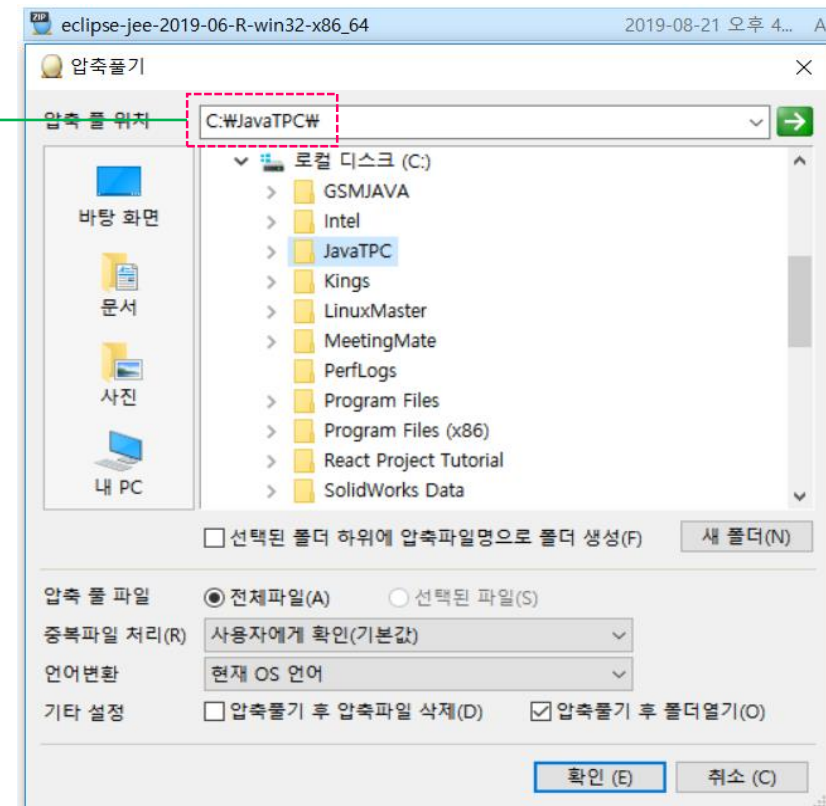


copy

eclipse-jee-2019-06-R-win32-x86\_64

압출풀기

jdk-12.0.2\_windows-x64\_bin



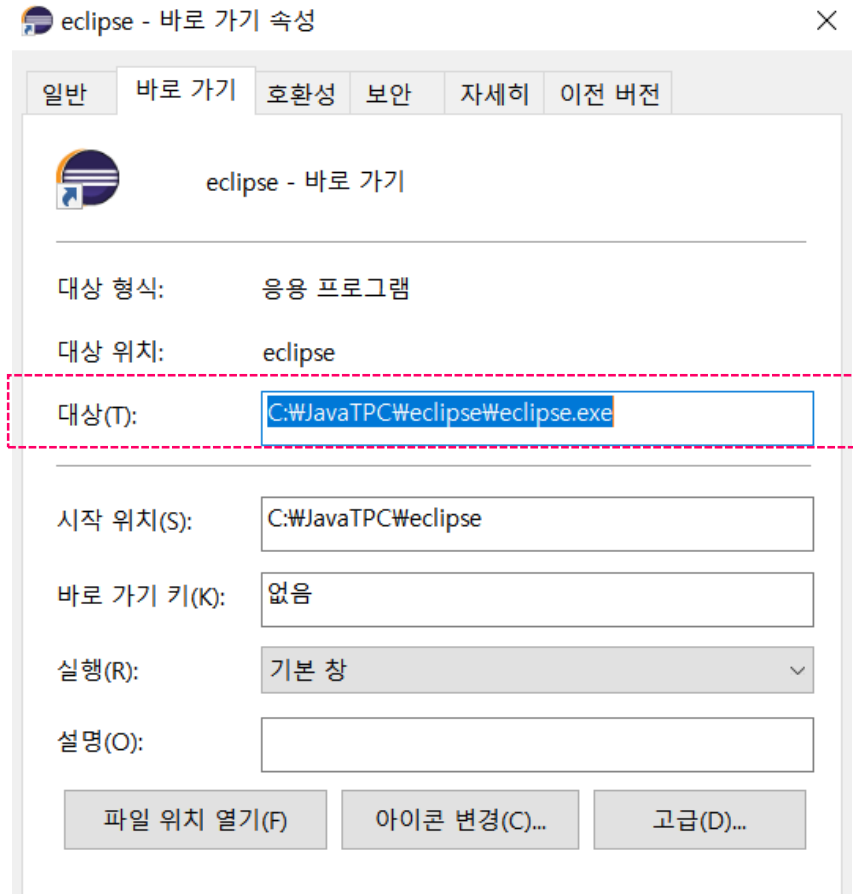
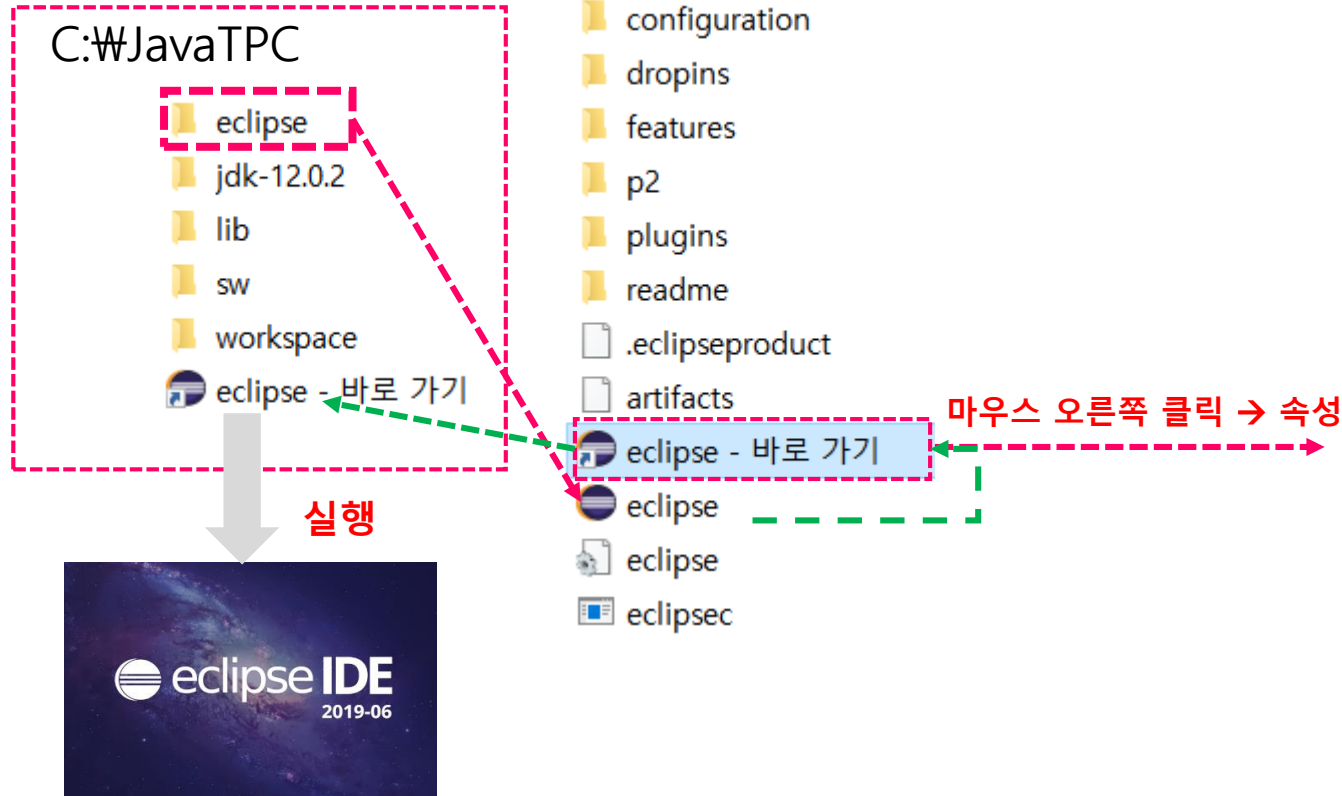
JDK를 설치하면 JavaSE개발환경이 구축 되었다.

TOOL(컴파일러, 실행 명령어 등...)  
API(미리 만들어 놓은 class 등...)  
JVM(자바가상머신)

## 4. Eclipse IDE 실행환경 구성

## 최종 디렉토리 구성

C:\JavaTPC



```
C:\JavaTPC\workspace>weclipse.exe -vm C:\JavaTPC\jdk-12.0.2\bin\javaw.exe -data C:\JavaTPC\workspace
```

```
-vm C:\JavaTPC\jdk-12.0.2\bin\javaw.exe
```

```
-data C:\JavaTPC\workspace
```

## 5. Java구동 방식

C:\JavaTPC\workspace\JavaTPC

.settings

bin

TPC01.class(실행가능한 파일:byte code)

src

TPC01.java(소스파일)

.classpath

.project

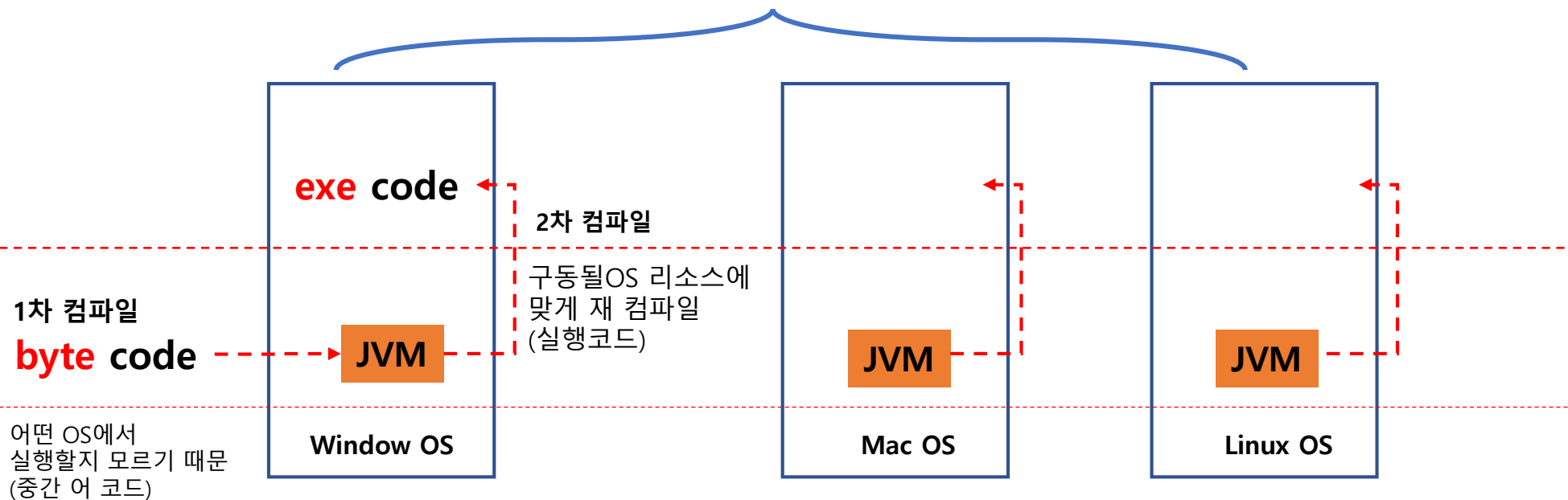
C:\JavaTPC\workspace\JavaTPC\src\java.exe TPC01

실행

C:\JavaTPC\workspace\JavaTPC\src\javac.exe TPC01.java

컴파일

Java 프로그램은 OS에 독립적으로 실행할 수 있다.  
JVM이라는 가상머신이 구동하기 때문이다.



## 1. 변수, 자료형, 할당

## 1. 변수(Variable)

- 데이터를 저장할 메모리 공간의 이름(symbol)

## 2. 자료형(Data Type)

- 변수의 크기와 변수에 저장될 데이터의 종류를 결정하는 것

## 3. 할당(Assign)

- 변수에 값을 저장(대입, 할당)하는 것

## 자료형(DataType)

→ **기본자료형(PDT)** : 컴파일러에서 기본적으로 제공하는 자료형

종류	자료형	크기(byte)	예시
정수	short, int, long	2, 4, 8	10, 20
실수	float, double	4, 8	23.4f, 34.567
문자	char	2	'A', 'a'
불	boolean	1	true(참), false(거짓)

VS

→ **사용자정의자료형(UDDT)** : 객체 자료형(Object DataType)

- 필요에 의해서 새롭게 만들어 사용하는 자료형
- 만드는 도구, 설계하는 도구, 모델링하는 도구 가 필요하다. : **class**

종류	자료형	예시
책	BookDTO	자바의정석(제목, 가격, 출판사)
회원	MemberVO	김길동(이름, 주소, 전화번호)
문자열	String	"APPLE"

객체(Object)  
책



class를 이용하여 객체를 설계

Java API  
java.lang.\*

```
public class BookDTO{
    public String title;
    public int price;
    public String company;
    public int page;
}
```

→ 우리가 만드는 객체는 프로젝트에 따라 다양하므로 **class**로 언제든지 만들어 사용하면 된다 !!!



## 2. 변수선언과 할당

## 1. 변수(Variable)

- 데이터를 저장할 메모리 공간의 이름(symbol)

## 변수선언

메모리에 변수(기억공간)를 만드는 것

**DataType + Variable**

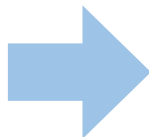
변수가 선언되면 **ST(변수테이블)**에 등록이 된다.

## 2. 자료형(Data Type)

- 변수의 크기와 변수에 저장될 데이터의 종류를 결정하는 것

```
int a;
float b;
```

```
a=10;
b=34.5f;
```

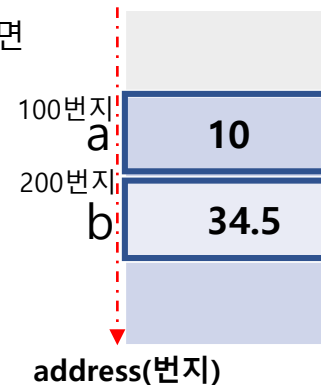


Symbol Table(변수목록표)

변수이름(key)	번지(value)
a	100번지
b	200번지

symbol table을 거쳐서  
memory에 접근  
symbol table에 변수가 없으면  
**can not find symbol** 에러

Memory



※ Symbol Table(변수목록표)

변수가 기억공간을 할당 받으면 변수의 번지가 등록되는 테이블

## 3. 할당, 대입(Assign, =) : 변수에 값을 대입하는 것

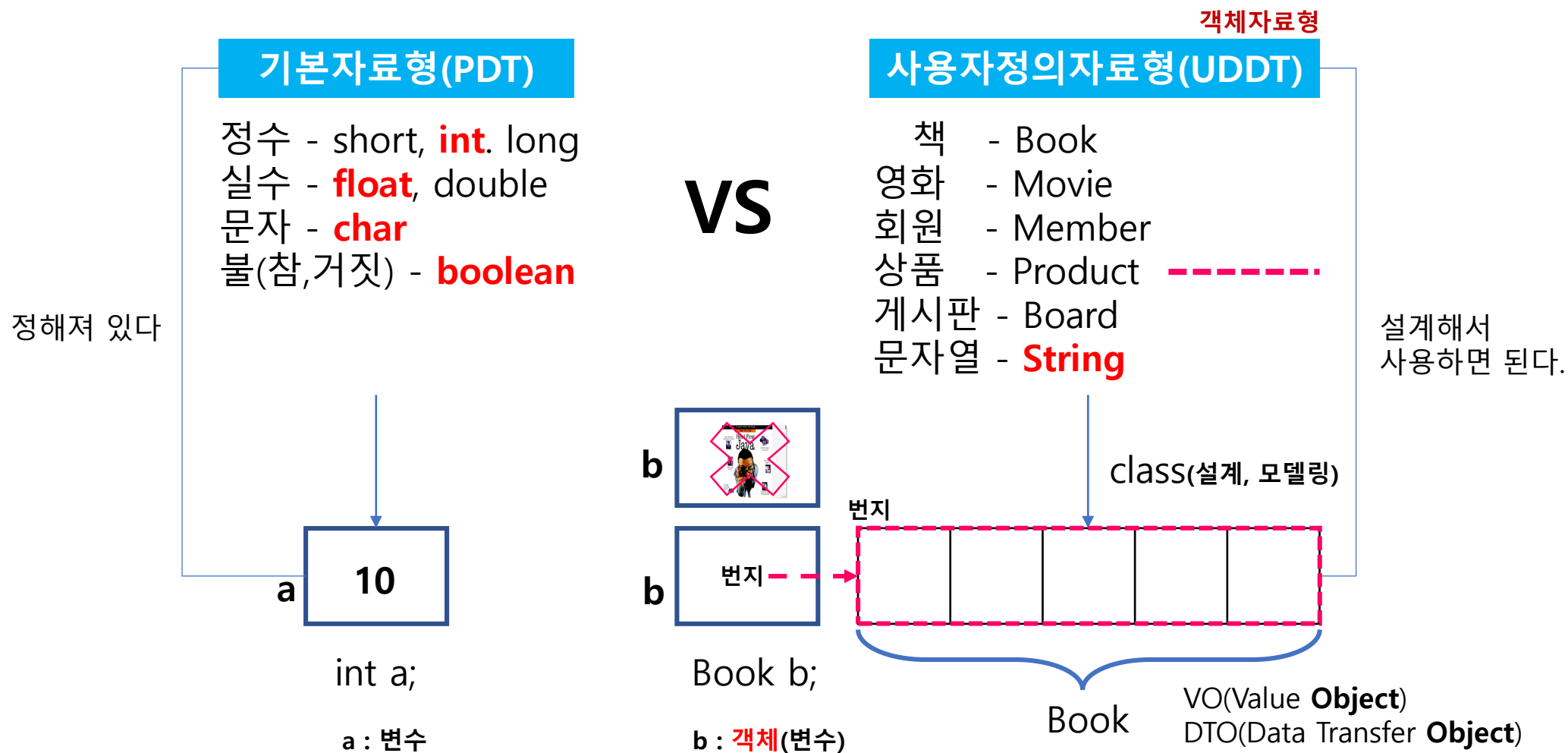
=  
해석에 주의

L-Value = R-Value;

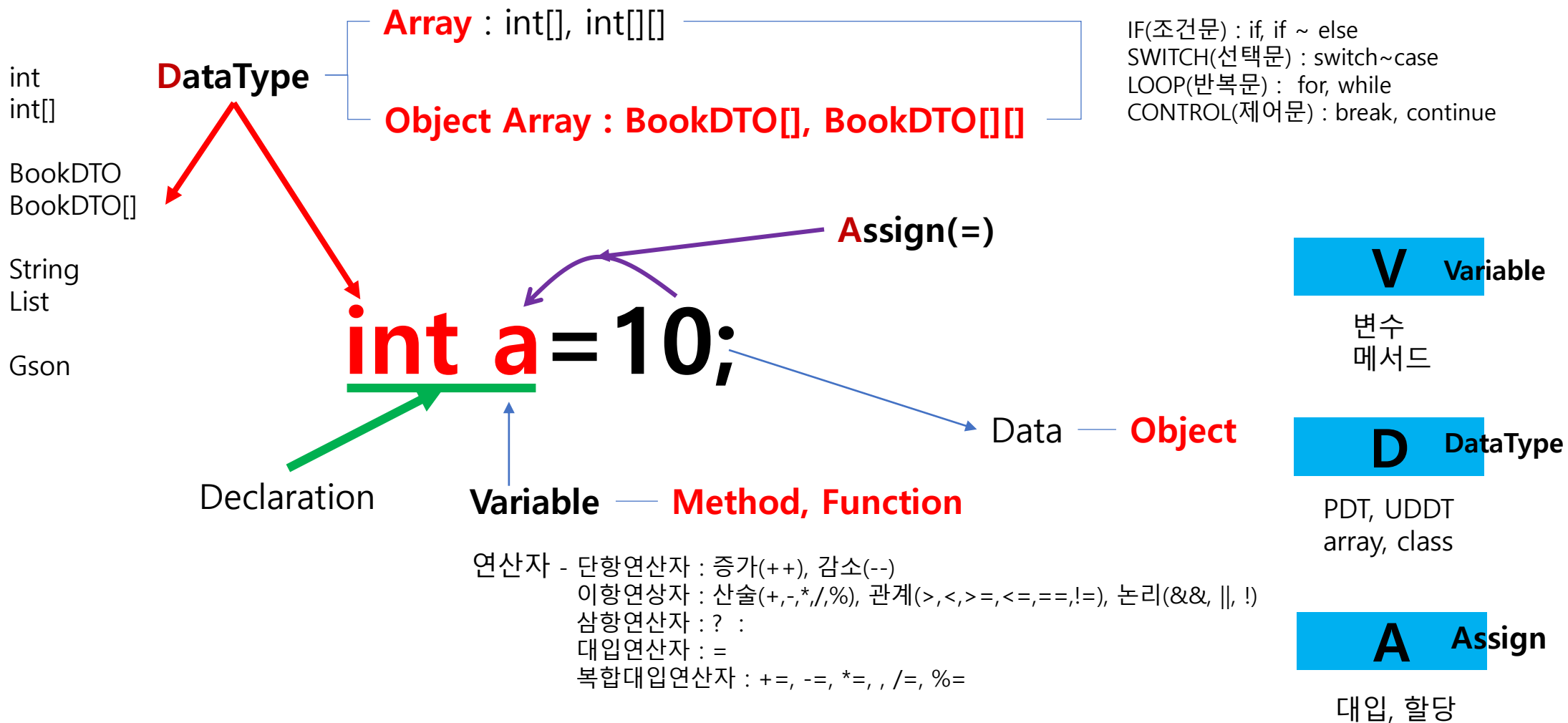
변수 = 값, 변수, 수식, 메서드 호출 문

```
a=10;
a=b;
a=b+20;
a=sum(b, c);
```

## 1. 기본자료형 vs 사용자정의자료형

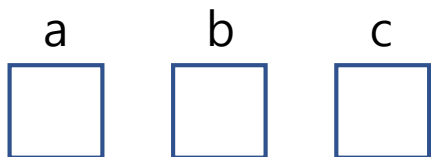


## 2. 관계를 이해하라.(Relational)

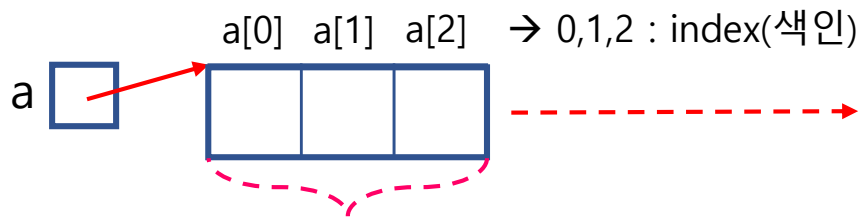


## 1. 변수와 배열(Array)

Q) 3개의 정수를 저장하기 위해서 변수 3개를 만드는 방법

변수를 **개별적(불연속)**으로 만드는 방법

```
int a, b, c;
a=10;
b=20;
c=30;
```

변수를 **연속적**으로 만드는 방법(Array, 객체)

[ ] 배열기호  
[ ] : 1차원  
[ ][ ] : 2차원

int[ ] : int가 여러 개 인 구조

```
int[] a;
a=new int[3];
or
int[] a=new int[3];
```

```
a[0]=10;
a[1]=20;
a[2]=30;
a.length=>3
```

배열의 길이?

→데이터 처리가 복잡하다.  
→데이터 이동이 어렵다.  
→데이터를 한 개만 저장가능 하다.

Array(배열) 특징

class

→많은 수의 변수를 만들기가 용이하다.  
→기억공간 접근이 쉽다(반복 문 사용 가능)  
→데이터 이동이 쉽다.  
(데이터를 하나의 형태로 담아서 이동 가능)

→ 서로 다른 데이터 타입(이질적인 구조, 객체)을 저장 할 수 없다  
( Array단점)

Array(배열)

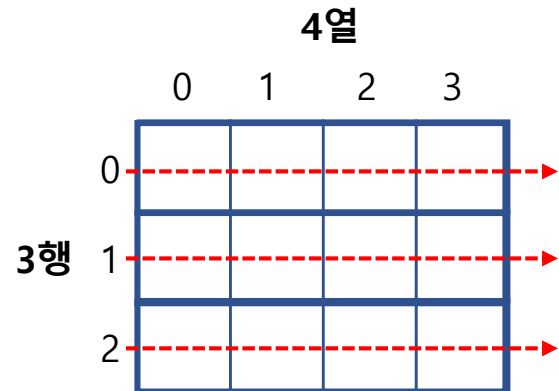
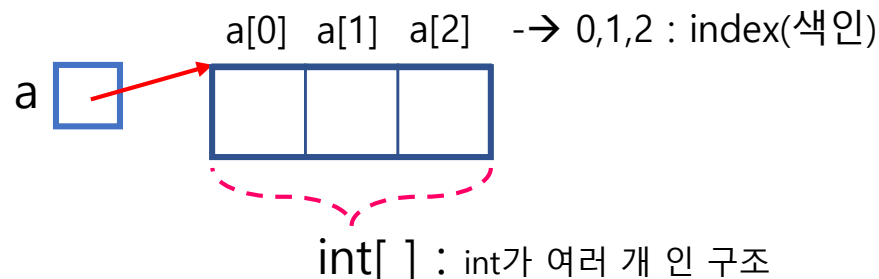
동일한 타입의 데이터를 여러 개 저장하기위한 연속적인 메모리 구조 - Array(배열)

## 2. 1차원, 2차원 배열(Array)

## 1차원 Array

```
int[] a;
a=new int[3];
or
```

```
int[] a=new int[3];
```



row major

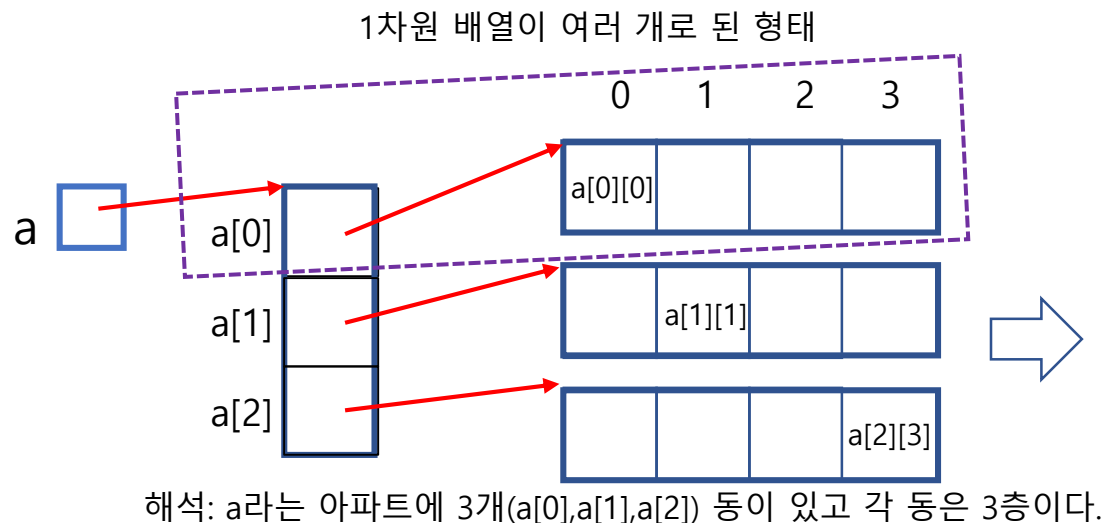
## 2차원 Array

```
int[][] a;
a=new int[3][4];
or
```

```
int[] a=new int[3][4];
```

```
a.length=>3
a[0].length=>4
a[1].length=>4
a[2].length=>4
```

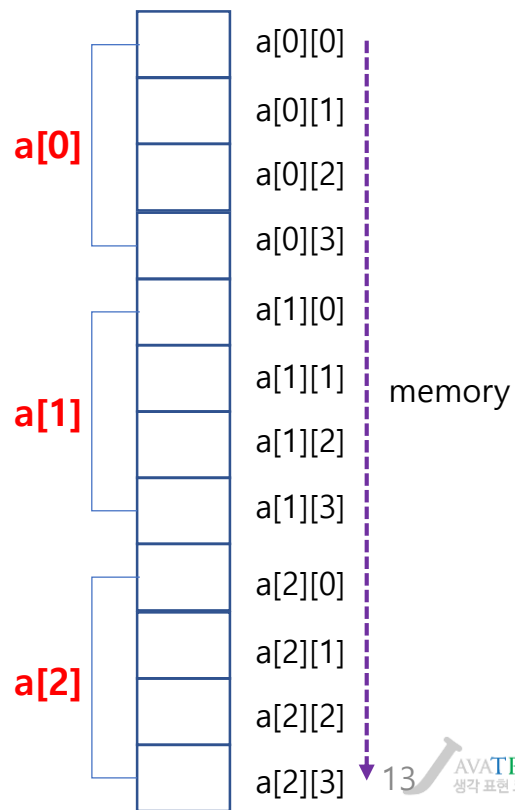
[3][4]  
=3행4열



[가변길이 배열]

```
int[] a=new int[3][];
```

```
a[0]=new int[3]; a[1]=new int[4]; a[2]=new int[5];
```



## 1. 변수와 메서드(method)

변수(variable)

변수(Variable) : 데이터를 **한 개 만(한 개의 형태)** 저장 가능하다.  
→ 저장만 한다.

메서드(method) → 메서드 이름이 변수 역할을 한다

메서드(method) : 동작을 한 후에 데이터를 **한 개 만** 만들어 낸다.  
→ 동작 후 저장한다.

DataType

**int** a=10;

return DataType

**int** sum=a+b;

변수와 메서드는  
결론적으로 데이터를 한 개만  
저장하므로 비슷하다.

↑ 함축적인 표현

메서드에서 리턴 하는 값을 메서드 이름에 저장한다.  
(메서드 이름이 **변수 역할**을 한다)

method 호출문

**int v=sum(10,30);**

method Call(호출)

method 선언문

```
public int sum(int a, int b){
    return a+b;
}
```

**sum=40**

return

외부로부터 데이터를 받을 때

method 선언  
(정의 부 + 구현 부)

접근제어자 리턴타입 메서드이름(매개변수 리스트){  
// 처리부분 ....  
// 리턴여부 → return  
}

## 2. 메서드의 매개변수 전달기법(parameter passing)

## Call By Value(값 전달 기법)

기억공간 개별

```
int a=10; int b=20;
```



```
int v=sum(a, b);
```

method 호출 부

a,b : value(값)

Call By Value

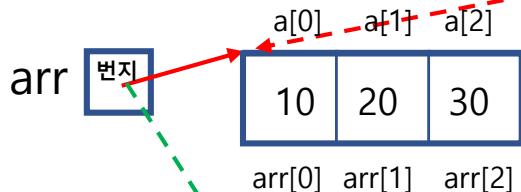
method 정의 부

```
public int sum(int a, int b){  
    int v=a+b;  
    return v;  
}
```

## Call By Reference(번지전달 기법)

기억공간 공유

```
int[] arr={10, 20, 30};
```



```
int v=sum(arr);
```

method 호출 부

arr : reference(번지)

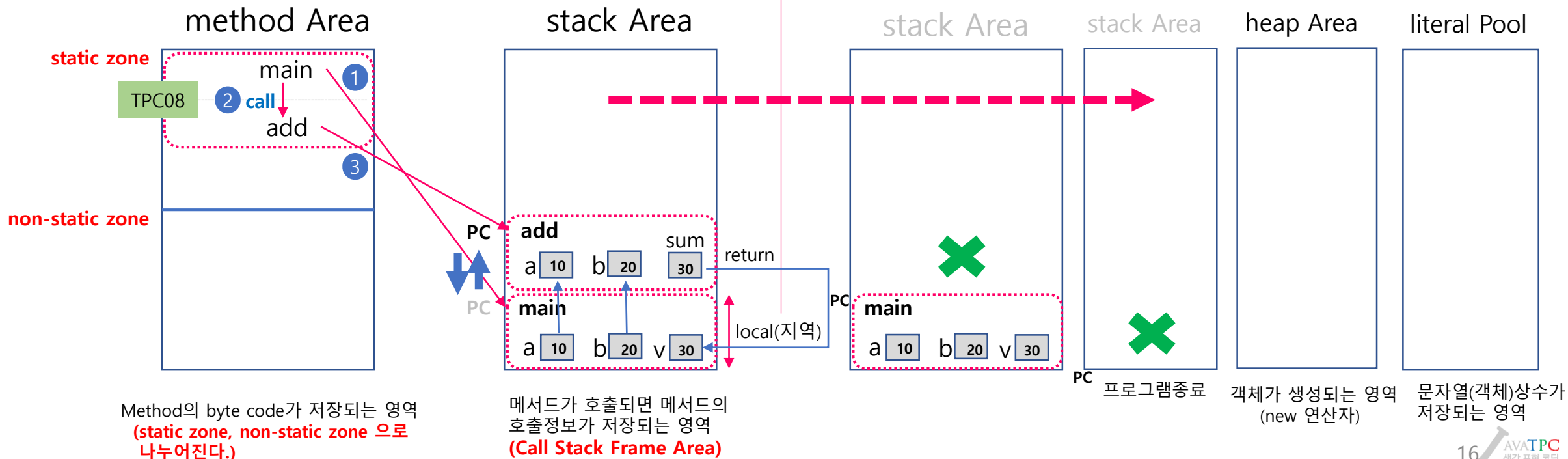
Call By Reference

```
public int sum(int[] a){  
    int v=0;  
    for(int i=0;i<a.length;i++){  
        v+=a[i];  
    }  
    return v;  
}
```

## 1. JVM Memory Model 1

[JVM이 TPC08 class(실행클래스)를 실행하는 절차]

1. 해당클래스를 현재 디렉토리에서 찾는다.
2. 찾으면 클래스 내부에 있는 **static 키워드**가 있는 메서드를 메모리로 로딩 한다.  
- method Area의 static zone에 로딩 한다. **main()** , **add()** method
3. static zone에서 main() 메소드를 실행한다(호출, 시작)  
- main() method가 호출되면 main() method의 호출정보가 Stack Area에 들어간다(push)  
- 프로그램이 시작되는 부분이다.(PC의 위치가 현재 동작되고 있는 메서드다.)
4. Stack Area가 비어 있으면 프로그램이 종료된 것이다.



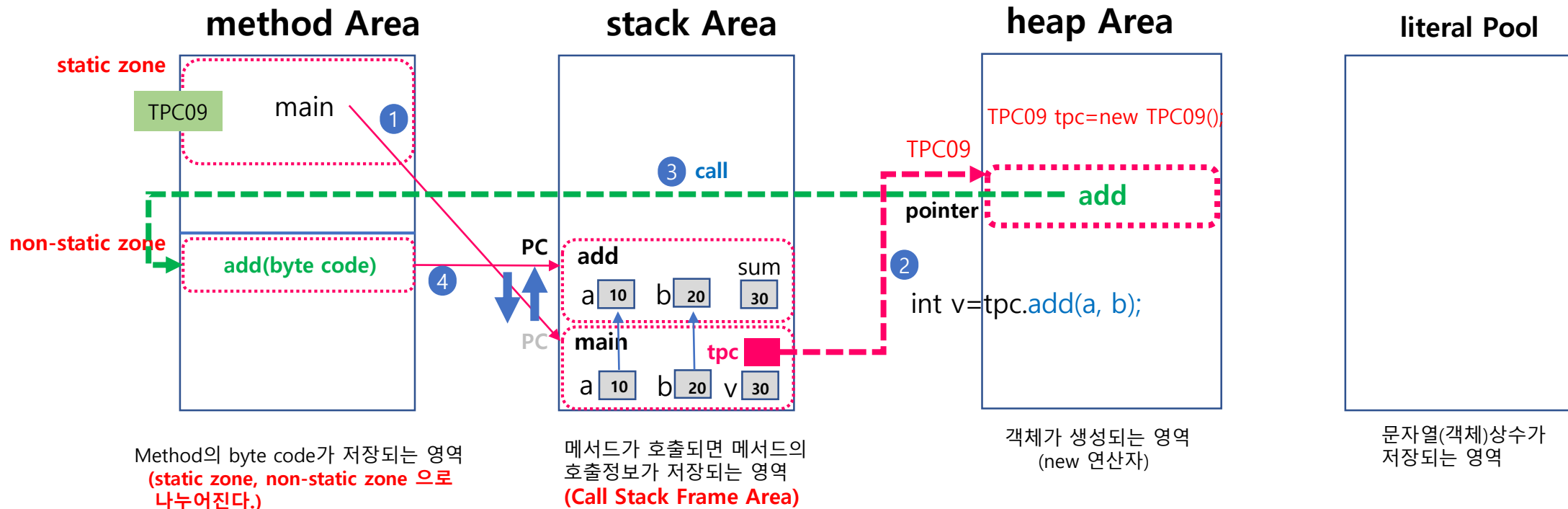


## 2. JVM Memory Model 2

[JVM이 TPC09 class(실행클래스)를 실행하는 절차]

1. 해당클래스를 현재 디렉토리에서 찾는다.
2. 찾으면 클래스 내부에 있는 **static 키워드**가 있는 메서드를 메모리로 로딩 한다.  
- method Area의 static zone에 로딩 한다. main() method
3. static zone에서 main() 메서드를 실행한다(호출, 시작)  
- main() method가 호출되면 main() method의 호출정보가 Stack Area에 들어간다(push)  
- 프로그램이 시작되는 부분이다.(PC의 위치가 현재 동작되고 있는 메서드다.)
4. Stack Area가 비어 있으면 프로그램이 종료된 것이다.

```
public class TPC09{
    public static void main(String[] args){
        int a=10; int b=20;
        TPC09 tpc=new TPC09();
        int v=tpc.add(a, b);
        System.out.println(v);
    }
    public int add(int a, int b){
        int sum= a+b;
        return sum;
    }
}
```



## 1. 기본자료형(PDT) VS 사용자정의자료형(UDDT)

## 이것만 알면 !

1. **기본자료형(PDT) VS 사용자정의자료형(UDDT)**
  - DataType을 확실히 이해하자
2. **class, object, instance 상호관계**
  - 객체생성과정(new 연산자, 생성자 메서드, this)
3. **잘 설계된 클래스**
  - DTO(VO), DAO, Utility

## 기본자료형(PDT) VS 사용자정의자료형(UDDT)

정수(10)

**int** a;

DataType

**기본자료형(PDT)**  
컴파일러에서 기본적으로  
제공해주는 자료형

책(제목,가격,출판사,페이지수)

**BookDTO** b;

**사용자정의자료형(UDDT)**  
사용자가 직접 만들어서  
사용하는 자료형

class로 새로운 자료형을 만든다.

```
public class BookDTO {
    public String title;
    public int price;
    public String company;
    public int page;
}
```

```
public BookDTO(){
    super();
}
```

default constructor  
(기본생성자)

stack Area

b

this

heap Area

객체생성과정?

BookDTO b=new BookDTO();

new  
생성자 메서드

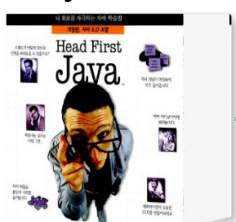
title price company page

BookDTO

## 2. 객체생성과정

## 객체생성과정 BookVO b=new BookVO(); // new 연산자와 생성자 메서드 호출

Object(책)



1. 상태정보(변수) : attribute, property, member

: 제목, 출판사, 저자, 가격, 페이지수, ISBN, 이미지, 무게, 두께, 재질.....

Modeling

필요한 속성만 뽑아내는 과정

제목, 가격, 출판사, 페이지수 -&gt; Head First Java, 30000, 한빛 미디어, 700

2. 행위정보 : 동작(method), 기능(function)

```
public class BookVO {
    public String title;
    public int price;
    public String company;
    public int page;
}
```

객체생성

b

this

Memory

Head First Java

30000

한빛 미디어

700

title

price

company

page

BookVO

Point

b.

title  
price  
company  
page.(dot)연산자  
접근, 참조연산자

public member만 접근가능

객체생성 후 접근 방법 .(dot) 연산자

```
b.title = "Head First Java";
b.price = 30000;
b.company = "한빛 미디어";
b.page = 700;
```

객체의 상태정보를 직접 접근하면  
잘 못된 데이터가 저장될 수 있다.

설계를 잘 해야 된다.

정보은닉 필요  
Information Hiding  
private

### 3. 생성자 메서드(Constructor)

#### 이것만 알면 !

1. 객체를 생성할 때 사용되는 메서드
2. 객체 생성 후 객체의 초기화를 하는 역할 수행
3. 특징
  - 클래스이름과 동일한 메서드
  - 메서드의 return type이 없다(void 아님)
  - public 접근 권한을 가진다.(단, private 생성자도 있음)
  - 생성자가 없을 때는 기본 생성자가 만들어 진다.

#### 생성자 중복정의(Overloading)

**BookVO b=new BookVO();**

**BookVO b=new BookVO("자바",20000,"길벗",790);**

생성자 메서드를 활용하여 객체를 적절하게 초기화 하라. 중복정의(Overloading)

```
public class BookVO {
    private String title;
    private int price;
    private String company;
    private int page;

```

**BookVO b=new BookVO();**

```
public BookVO(){
    super();
}
```

BookVO() 호출

default constructor : 초기화 작업 없음  
(기본생성자)



```
public class BookVO {
    private String title;
    private int price;
    private String company;
    private int page;

```

**BookVO b=new BookVO("자바",20000,"길벗",790);**

초기값

```
public BookVO(String title, int price, String company, int page){
    this.title=title;
    this.price=price;
    this.company=company;
    this.page=page;
}
```

초기화

→ overloading constructor : 초기화를 위해서(중복 정의된 생성자)  
→ 생성자를 중복정의 하면 기본생성자는 자동으로 만들어지지 않는다.

## 4. private 생성자 메서드(Constructor)

- 객체생성에 관여하는 생성자 메서드가 private 접근제어를 가지면 객체를 생성할 수 없다는 뜻이 된다.
- 그러므로 객체를 생성하지 않고도 사용가능 해야 된다.(모든 클래스의 멤버가 static 멤버가 되어야 한다.)

## → non-static 멤버인 경우(인스턴스 메서드)

객체생성 후 접근 가능

```
Inflearn inf=new Inflearn(); // 생성자가 public인 경우
inf.tpc();
```

## → static 멤버인 경우(클래스 메서드)

객체생성 없이 접근가능(클래스 이름으로 접근)

```
Inflearn.java();
```

↓  
클래스를 사용하는 시점에서 static 멤버는 먼저  
자동으로 메모리에 로딩이 된다. 그 이후에 호출이 된다

생성자가 private이므로 객체 생성 불가  
Inflearn inf=new Inflearn(); // X

- ※ Java API 중에서도 생성자가 private인 클래스도 많이 있다.  
System, Math.... System sys=new System(); X  
→ 자주 사용하는 객체나, 동작은 static 멤버로 만든다.

```
public class Inflearn {
    private Inflearn() {
    }
}
```

모든 멤버가 static멤버이면 인위적으로 private  
생성자로 만들어 객체생성을 막을 수도 있다.

// 인스턴스 메서드

```
public void tpc(){
    System.out.println("TPC강의 너무 재미있다.");
}
```

// 클래스 메서드

```
public static void java(){
    System.out.println("Java강의 너무 재미있다.");
}
```

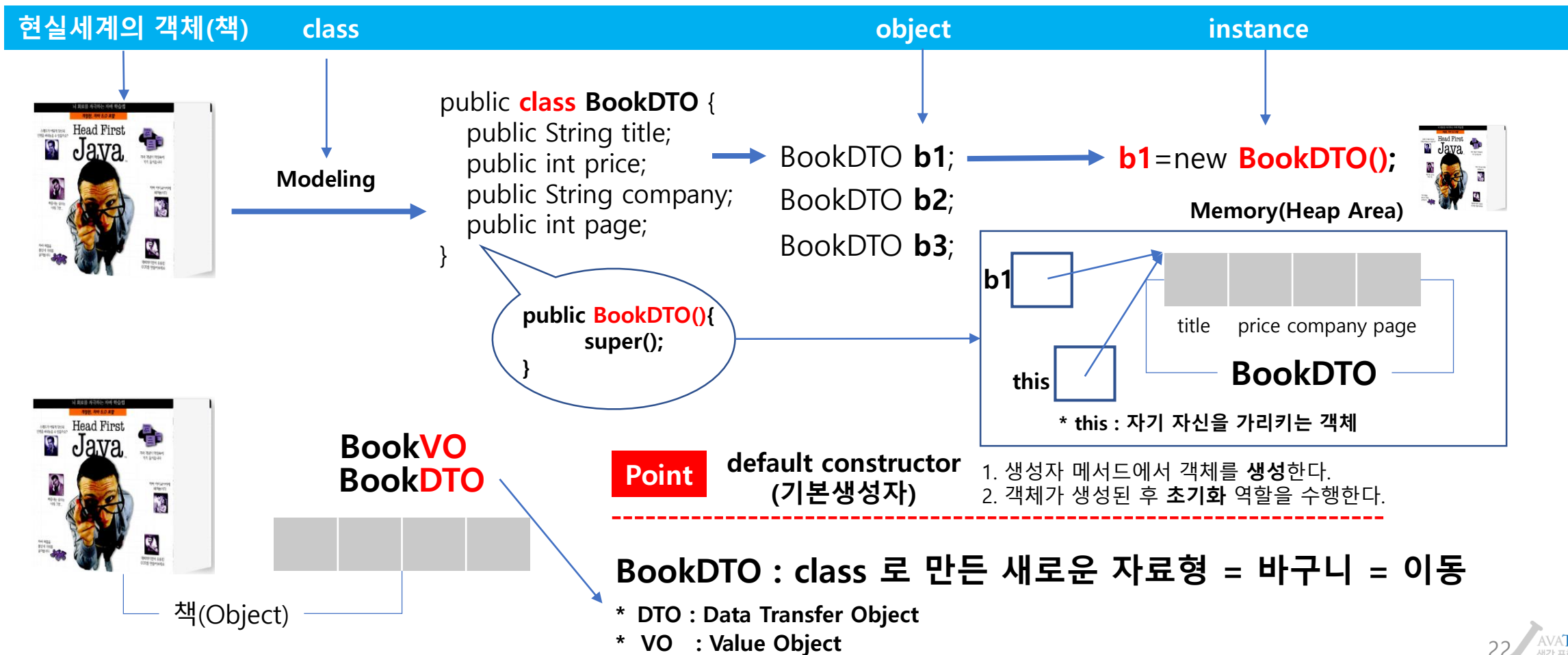
어떤 객체에 생성자 메서드가 private이면  
모든 멤버는 static가 붙은 멤버가 되어야 한다.

※ static 멤버 접근방법

클래스이름.클래스메서드(static 메서드)

## 1. class, object, instance 상호관계

객체 생성과정 **BookDTO b=new BookDTO();** // new 연산자와 생성자메서드 호출



## 1. 정보은닉(private)

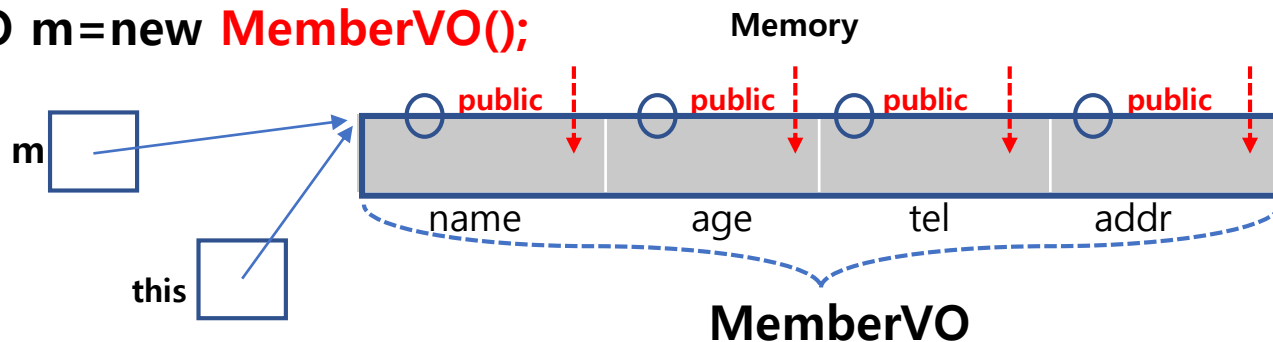
→ 정보은닉(private) : 다른 객체(class)로부터 접근을 막는 것(private)

```
public class MemberVO {
    private String name;
    private int age;
    private String tel;
    private String addr ;

    public MemberVO(){

    }
}
```

MemberVO m=new MemberVO();



Point

m

name  
age  
tel  
addr

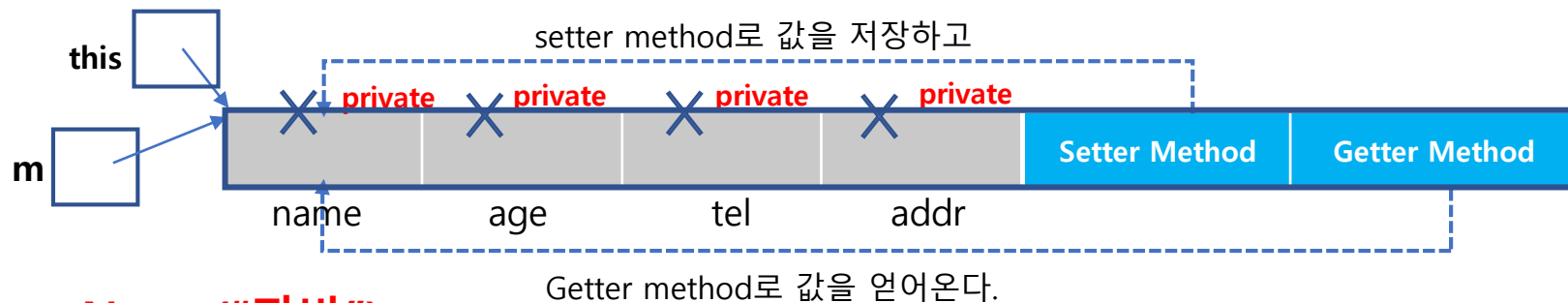
접근할 수 없다

m.name = "홍길동";  
m.age=30;  
m.tel="010-0000-0000";  
m.addr="서울";

private 멤버변수를 접근할 때  
setter, getter method를 활용하라!

setter, getter method

```
public void setName(String name){
    this.name=name;
}
public String getName(){
    return name;
}
```



m.setName("자바");  
m.getName();

## 2. 잘 설계 된 DTO, VO 클래스

```
public class MemberVO {  
    private String name;  
    private int age;  
    public MemberVO() { }  
    public MemberVO(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getAge() {  
        return age;  
    }  
    public void setAge(int age) {  
        this.age = age;  
    }  
    @Override  
    public String toString() {  
        return "MemberVO [name=" + name + ", age=" + age + "];"  
    }  
}
```

1

private 으로 객체의 상태를 보호한다.  
정보은닉(information hiding)

2

디폴트 생성자를 명시적으로 만든다.  
오버로딩 생성자를 만들어 적절하게 초기화 한다.  
• 객체를 생성하는 작업은 생성자 내부에서 JVM이  
자동으로 처리한다.

3

Private으로 만들어진 멤버변수를 접근하기 위해서  
setter, getter method를 만든다.

- DI(Dependency Injection : 종속객체 주입)  
- setter method의 역할

4

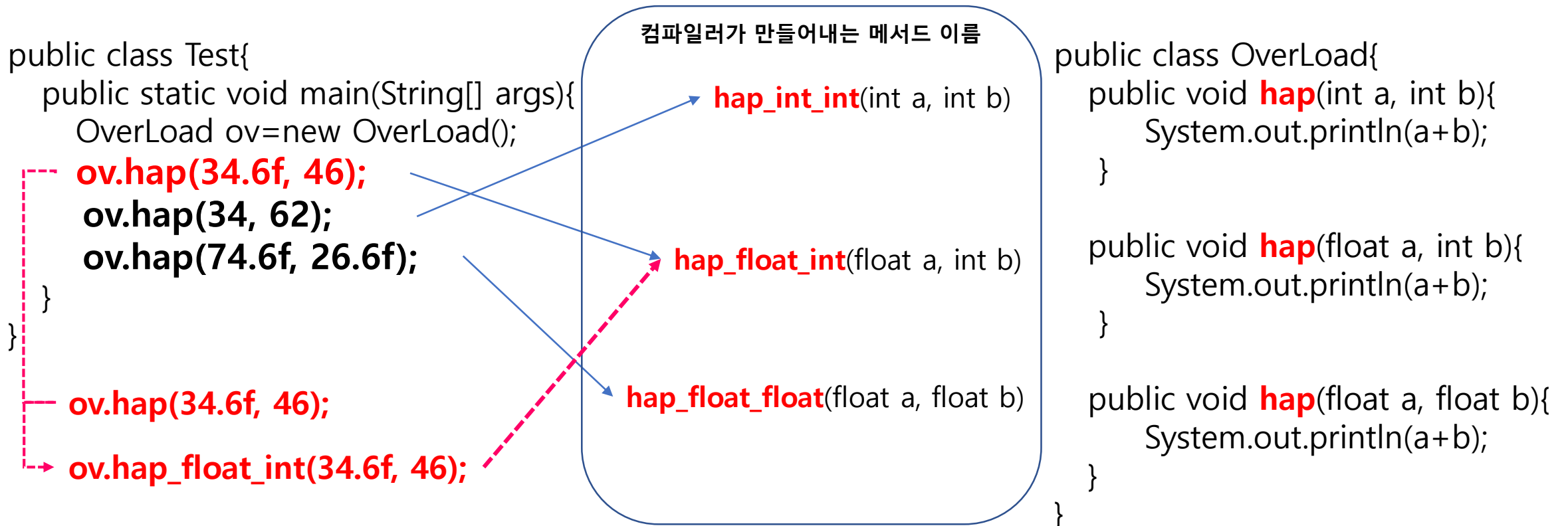
객체가 가지고 있는 값 전체를 출력하기위한  
toString() method를 재정의 한다.



## 1. 메서드 오버로딩(Method Overloading)

## 메서드 오버로딩(Method Overloading)

같은 이름의 메소드를 여러 개 가지면서 매개변수의 유형과 개수가 다르도록 하는 기술  
 ->메서드의 signature 가 다르면 된다(signature : 매개변수의 타입, 개수)



## Point

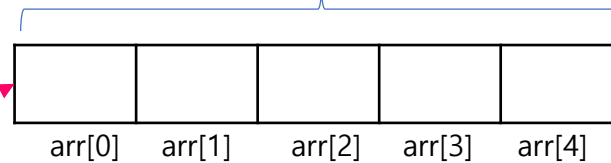
오버로딩 (Overloading) : 정적 바인딩(컴파일 시점에서 호출될 메서드가 이미 결정되어 있는 바인딩)→속도와는 관계 없다.

## 1. 배열 VS 클래스의 관계

`int[] arr=new int[5]`

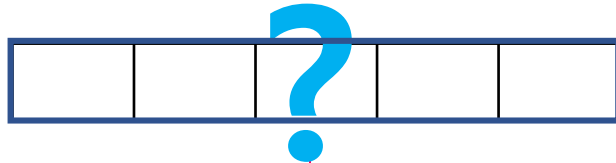
정수, 정수, 정수, 정수, 정수

구조만들기

구조의 이름: `int[]`

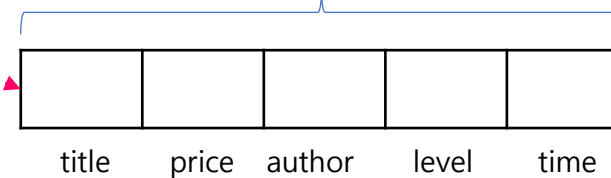
Array

동일한 데이터 구조



객체(Object)

구조만들기

구조의 이름: `MovieVO`

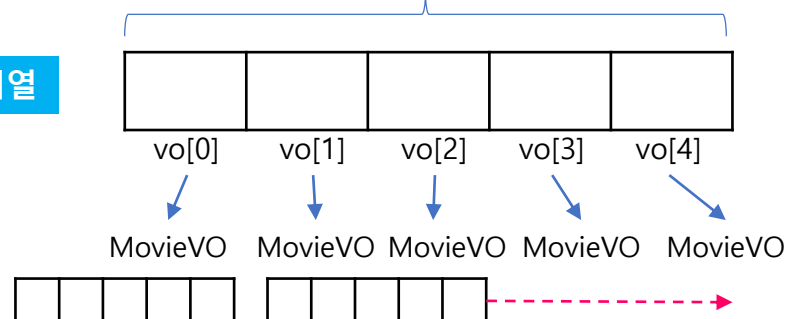
class

서로 다른(이질적인)한 데이터 구조

```
public class MovieVO{
    private String title;
    private int price;
    private String author;
    private int level;
    private float time;
}
```

`MovieVO vo=new MovieVO();`

Object 배열

구조의 이름: `MovieVO[]`

// 객체배열

```
MovieVO[] vo=new MovieVO[5];
vo[0]=new MovieVO();
vo[1]=new MovieVO();
vo[2]=new MovieVO();
vo[3]=new MovieVO();
vo[4]=new MovieVO();
```

## 학습정리

- ▷ **class** - DataType 측면 : 새로운 자료형을 만드는(설계하는) 도구 = **모델링도구**
  - OOP(객체지향)측면 : 객체의 상태정보와 행위정보를 추출하여 **캡슐화** 하는 도구
- ▷ **Model** : **class**를 **Model**이라고도 부른다.(역할이 정해지므로)
- ▷ **우리가 만드는 Model의 종류(3가지는 거의 대부분 만들게 되어있다)**
  1. DTO(Data Transfer Object) : 데이터 구조, 데이터를 담는 역할, 이동하기위해서 데이터를 담는다.
    - VO(Value Object) : 객체를 담아서 하나의 값(덩어리)으로 취급한다는 의미로
  2. DAO(Data Access Object) : 데이터를 처리하는 역할(비즈니스 로직), 데이터베이스와 CRUD하는 역할
  3. Utility(Helper Object) : 도움을 주는 기능을 제공하는 역할(날짜, 시간, 통화, 인코딩 등)
- ▷ **우리가 앞으로 사용하게 될 class들** : API(Application Programming Interface)
  1. Java에서 제공해주는 class 들
    - String, System, Integer, ArrayList, Map 등
  2. 만들어 사용하는 class 들(DTO, DAO, Utility)
  3. 1, 2번이 아닌 다른 사람이 만들어서 제공해주는 class 들
    - Gson, Jsoup, POI, iText 등



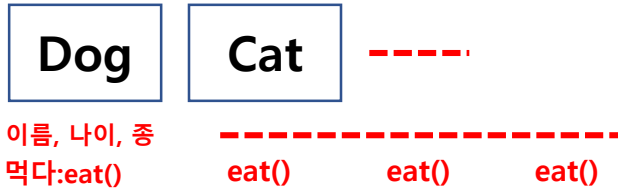
## PART -2

## 목차

- 수업 1 : Inheritance(수평적구조VS수직적구조)
- 수업 2 : 수평적구조VS수직적구조(실습)
- 수업 3 : 재정의(Override)
- 수업 4 : Override(실습)
- 수업 5 : 나보다 부모가 먼저야!
- 수업 6 : 나보다 부모가 먼저야(실습)
- 수업 7 : 부모 자식간 형변환이 된다.!
- 수업 8 : 리모콘 너무 좋은데( 다형성 이론)
- 수업 9 : 너무 좋아 좋아! 다형성의 활용
- 수업10 : 너무 좋아 좋아! 다형성의 활용(실습)
- 수업11 : 추상클래스(일부 다형성 보장)
- 수업 12 : 추상클래스(실습)
- 수업13 : 인터페이스(100% 다형성 보장)
- 수업 14 : 인터페이스(실습)
- 수업15 : 부모가 있어서 너무 좋아!
- 수업16 : 인터페이스와 JDBC의 관계
- 수업17 : 인터페이스와 JDBC의 관계 (실습)
- 수업18 : 인터페이스의 상속관계
- 수업19 : Object 클래스는 신이야!
- 수업21 : Object class의 활용
- 수업21 : 학습정리(객체지향의 3대 특징)

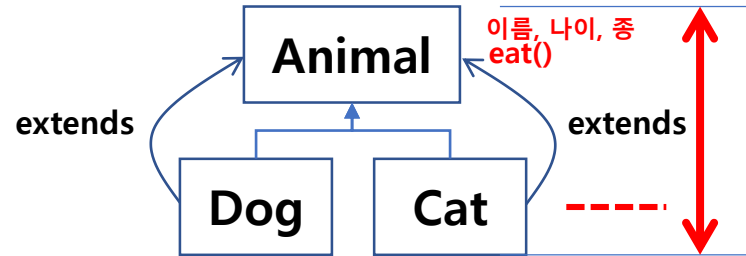
## 1. 상속→클래스의 설계(행위적인 측면)

### 수평적 설계



- 코드의 중복이 발생
- 새로운 요구사항에 대한 코드의 수정이 불가피하다.
- 관리하기가 어렵다.(부모와 자식의 관계를 생각해보라)

### 수직적 설계(계층화, 상속구조)



- 수평적 설계의 단점을 극복할 수 있다.
- 확장을 쉽게 할 수 있다.
- 코드가 복잡해 진다.(이점이 많다)

추상화  
보편화,  
일반화,  
개념화

super class(상위, 부모)

세분화,  
상세화,  
구체화,  
구상화

sub class(하위, 자식)

## 2. 상속 개념

```

public class Animal extends Object {
    public String name;
    public int age;
    public String part;
    public void eat(){
        System.out.println("?");
    }
    public Animal(){
        super();
    }
}
    
```

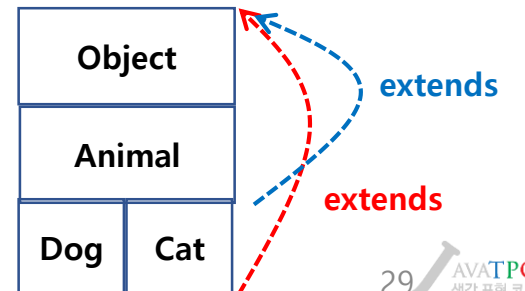
```

public class Dog extends Animal {
    public void eat(){
        System.out.println("개 처럼 먹다.");
    }
    public Dog(){
        super();
    }
}

public class Cat extends Animal {
    public void eat(){
        System.out.println("고양이 처럼 먹다.");
    }
    public Cat(){
        super();
    }
}
    
```

→상속에서 부모와 자식에 연결되는 방법  
**super()** : 자신의 생성자에서 부모의 생성자를 호출

상속 memory



## 1. Override(재정의) → 상속관계에서 상속받은 하위 클래스가 상위 클래스의 동작을 수정하는 것

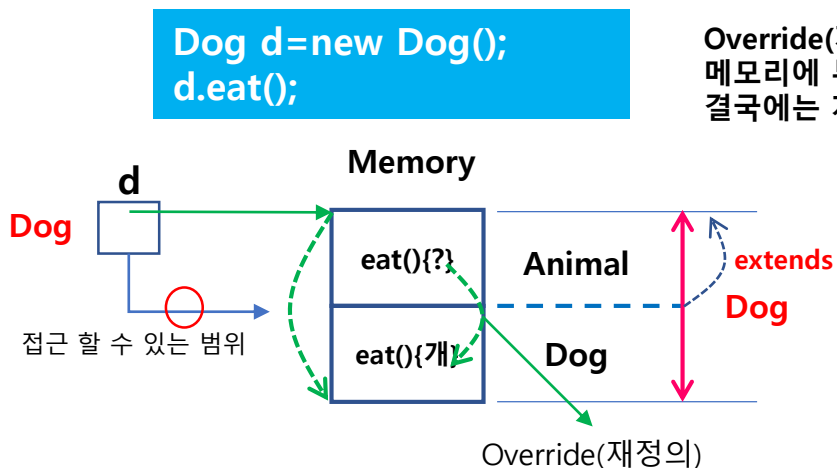
```

public class Animal {
    public void eat() {
        System.out.println("?"); // 포괄적, 추상적
    }
}

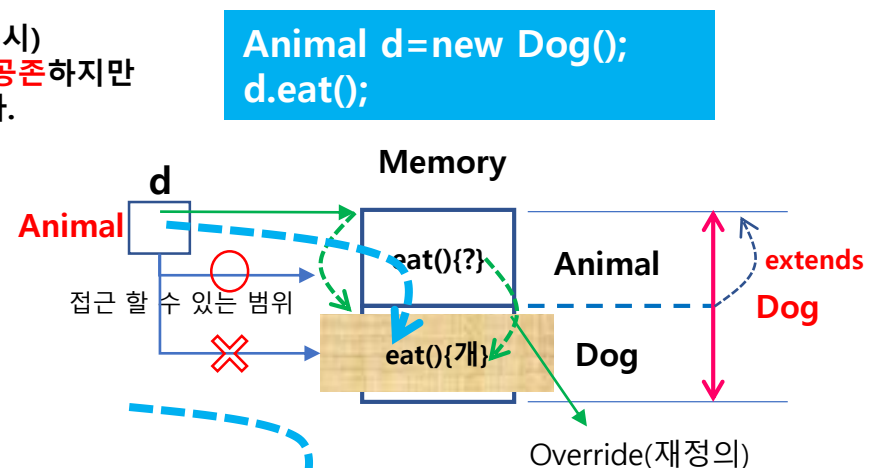
public class Dog extends Animal {
    public void eat() {
        System.out.println("개처럼 먹다.");
    }
}

public class Cat extends Animal {
    public void eat() {
        System.out.println("고양이 처럼 먹다.");
    }
    public void night() {
        System.out.println("밤에 눈에서 빛이 난다.");
    }
}
  
```

Override  
(재정의)



Override(재정의=보모 메서드 무시)  
메모리에 부모와 자식 메서드가 공존하지만  
결국에는 자식 메서드가 실행된다.

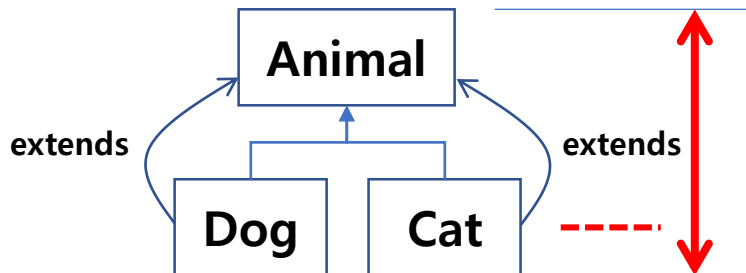


→ Override(재정의) : 동적 바인딩(호출될 메서드가 실행시점에서 결정되는 바인딩)  
프로그램의 속도가 떨어지는 원인이 되지만 이점이 더 많기때문에 사용 한다.

**Point**

Override를 통해 하위 클래스를 접근 할 수 있다.

## 1. 상속관계에서 객체생성 방법

상위클래스를(부모를)  
활용하라!`Animal d=new Dog();`

간접

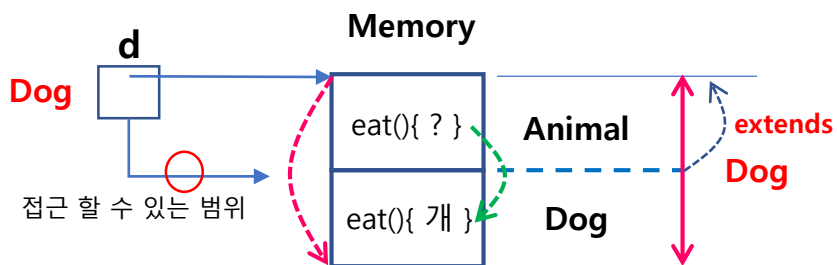
`Dog d=new Dog();`

직접

부모 클래스를 이용하지 않는 방식(**직접이용**)

```

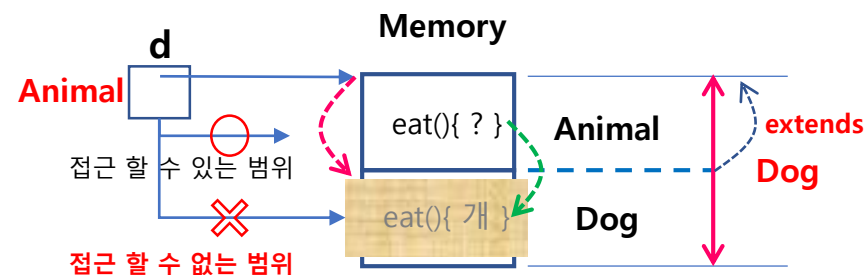
Dog d=new Dog();
Cat c=new Cat();
  
```

`Dog d=new Dog();`부모 클래스를 이용하는 방식(**하위 클래스의 동작 방식을 모를 때, 간접이용**)

→ .class(실행) 파일만 있고 .java(소스)파일이 없는 경우

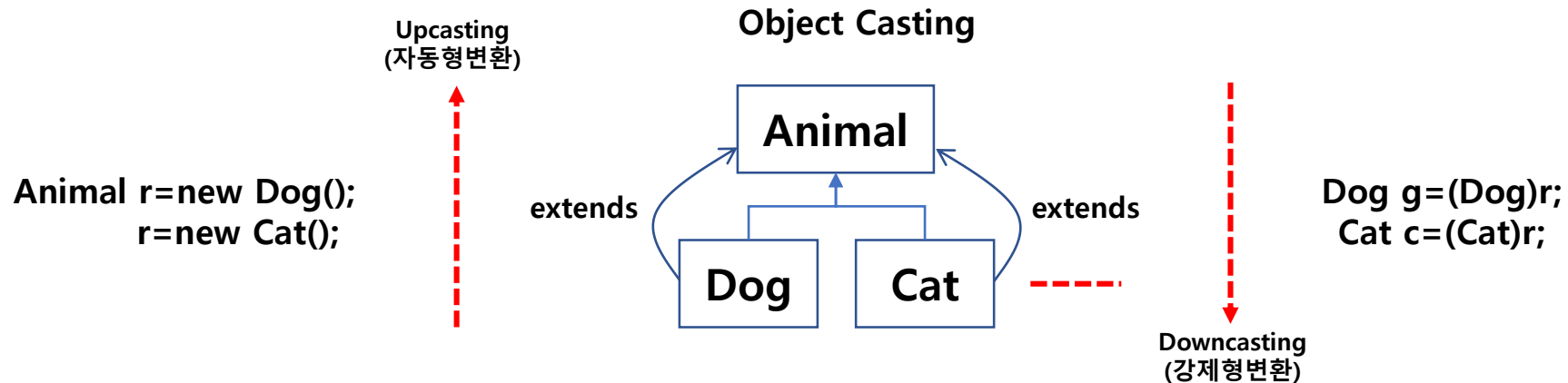
```

Animal d=new Dog();
Animal c=new Cat();
  
```

`Animal d=new Dog();`→ 하위 클래스를 접근 할 수 없다. 가능 하게 하는 방법 ? **Override(재정의)**

## 1. Object Casting(객체 형 변환)

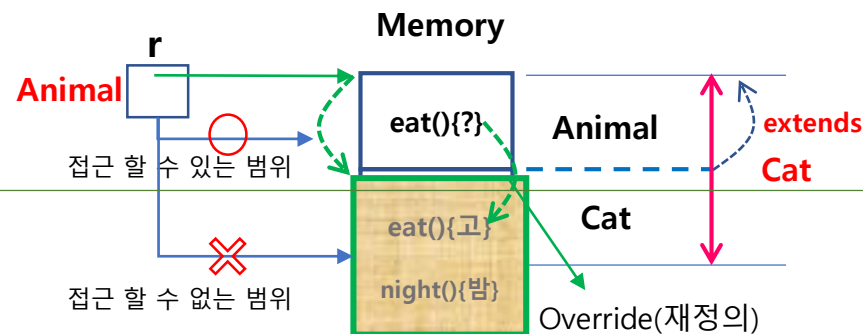
- 상속관계에 있는 클래스들 간의 형(DataType)을 바꾸는 것



**Animal** r=new Cat(); // upcasting  
r.eat();

**Animal** r=new Cat();  
// r.night();  
Cat c=(Cat)r; // downcasting  
c.night();

고양이처럼 먹다.

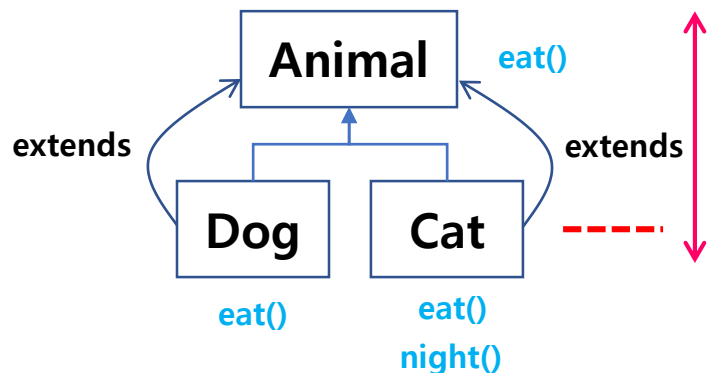


밤에 눈에서 빛이 난다.



## 1. message polymorphism(다형성)

- 상속관계에 있는 클래스에서 상위클래스가 동일한 메시지로 하위클래스들을 서로 다르게 동작시키는 객체지향 원리(개념)



### message polymorphism(다형성)

상위클래스인 Animal이 하위클래스인 Dog와 Cat에게 동일하게 먹어라(eat)라고 메시지를 보냈을 때 하위 클래스인 Dog와 Cat의 eat()메서드가 **서로 다르게 동작되는 객체지향 원리**

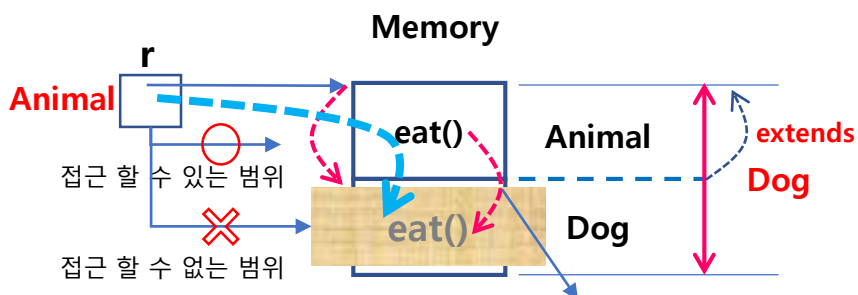
**Point**

개처럼 먹다.  
고양이처럼 먹다.

동작은 하지만 결과는 다르다.

```
Animal r=new Dog();
r.eat();
r=new Cat();
r.eat();
```

하위클래스의 동작방식을  
알 수 없어도 상위클래스를 통해  
하위클래스를 구동 시킬 수 있다.



Override(재정의)

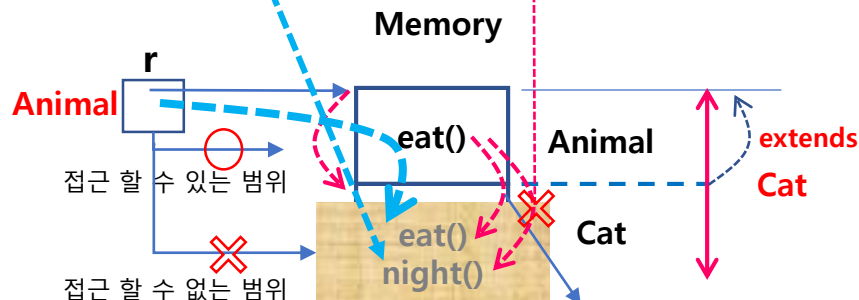
**Point**

Override를 통해 하위 클래스를 접근 할 수 있다.

```
Animal r=new Cat();
```

~~r.night();~~ night() method는 재정의가 되지 않음(접근불가)

```
Cat c=(Cat)r; // downcasting
c.night();
```



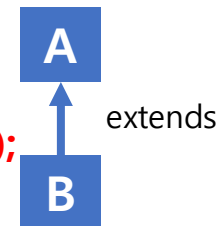
Override(재정의)

**Point**

Downcasting으로 하위클래스의 night()를 접근할 수 있다.

## 1. 다형성 이론의 전제조건(부모 클래스를 잘 활용하라)

- 상속관계가 되어야 한다.
- 객체생성을 upcasting으로 할 것(상위클래스가 하위클래스에게 메시지를 보내야 하므로)  
(upcasting이 되면 downcasting을 할 수 있다.)
- 하위클래스가 반드시 재정의(override)해야 한다.(다형성이 보장되기 위해서는)
- 동적 바인딩을 통해 실현된다.  
(동적 바인딩 : 실행시점에서 사용될 메서드가 결정되는 바인딩, 프로그램의 속도를 떨어뜨리는 원인이 된다.)



**A a=new B();**

## 2. ★ 다형성활용 방법 ★

부모 클래스를 잘 활용하라

### 1. 다형성인수(데이터 이동)

```
Dog d=new Dog();
display(d);
Cat c=new Cat();
display(c);
```

```
public static void display(Animal r){
    r.eat();
}
```

다형성인수

(Animal의 모든 하위클래스를 받을 수 있다.)

### 2. 다형성 배열(서로 다른 객체를 담을 수 있다)

**Animal[] r=new Animal[2];**

다형성배열

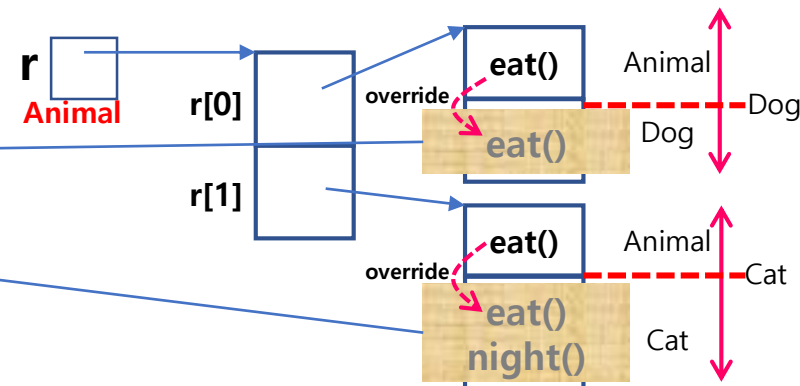
서로 다른 하위클래스를 담을 수 있다.

```
r[0]=new Dog();
r[1]=new Cat();
```

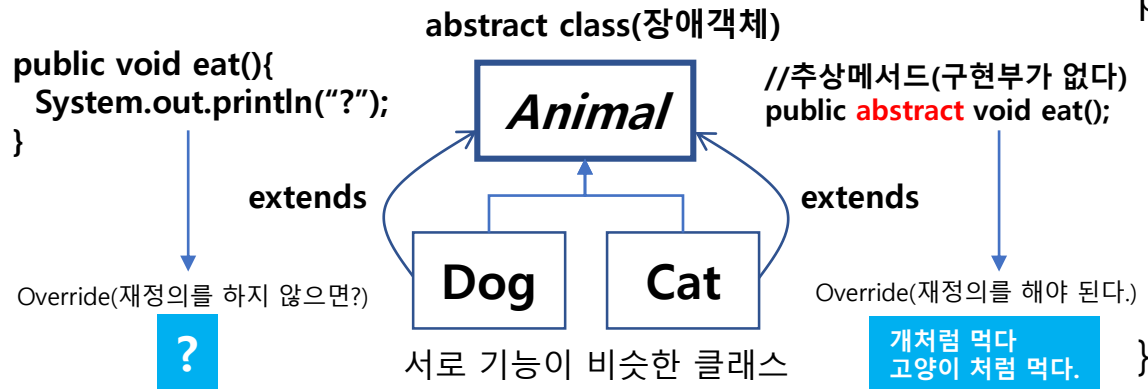
```
r[0].eat();
```

```
r[1].eat();
```

```
((Cat)r[1]).night();
```



## 1. 추상클래스 활용(일부 다형성 보장)



```
Animal r=new Dog();
r.eat();
```

```
Animal r=new Cat();
r.eat();
```

반드시 동작됨

```
public abstract class Animal {
    추상 메서드.....
}
```

```
override public class Dog extends Animal {
    추상 메서드를 반드시 구현
}
```

```
override public class Cat extends Animal {
    추상 메서드를 반드시 구현
}
```

추상클래스(불완전한 객체)

```
public abstract class Animal{
    public abstract void eat(); // 추상메서드
    public void move(){
        System.out.println("무리를 지어서 이동한다.");
    }
}
```

불완전한 메서드

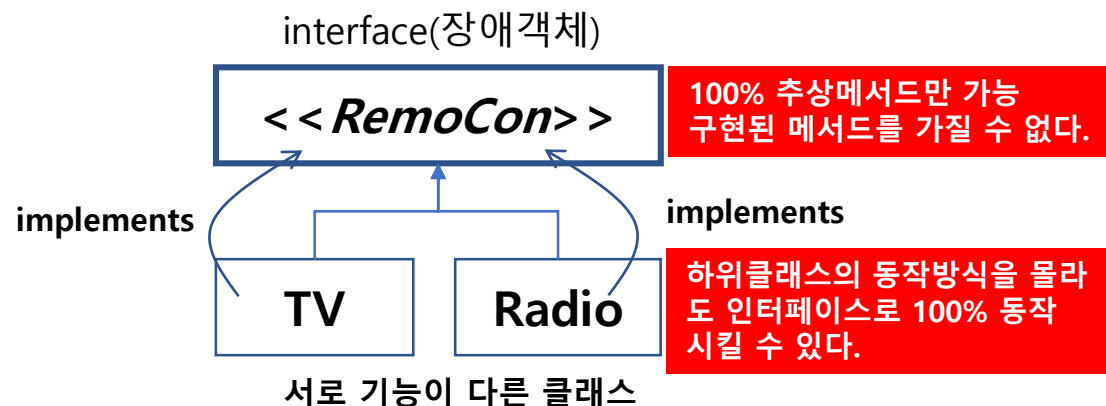
구현메서드

**override**

```
public class Dog extends Animal {
    public void eat(){
        System.out.println("개처럼 먹다.");
    }
}
```

```
public class Cat extends Animal {
    public void eat(){
        System.out.println("고양이 처럼 먹다.");
    }
    public void night(){
        System.out.println("밤에 눈에서 빛이 난다.");
    }
}
```

## 1. 인터페이스 활용(100% 다형성 보장)



<pre>RemoCon r=new TV(); r.chUp(); r.chDown(); r.internet();</pre>	반드시 동작됨	<pre>RemoCon r=new Radio(); r.chUp(); r.chDown(); r.internet();</pre>	반드시 동작됨
--	---------	---	---------

```

public interface RemoCon {
    추상 메서드.....
}

public class TV implements RemoCon {
    추상 메서드를 반드시 구현
}

public class Radio implements RemoCon {
    추상 메서드를 반드시 구현
}
  
```

## 인터페이스(불완전한 객체)

```

public interface RemoCon{
    public abstract void chUp();
    public void chDown();
    public abstract void internet();
    public void internet(){
        System.out.println("인터넷이 된다.");
    }
}
  
```

불완전한 메서드

구현된 메서드를 가질 수 없다.

```

public class TV implements RemoCon {
    public void chUp(){
        System.out.println("TV 채널이 올라간다.");
    }
    public void chDown(){
        System.out.println("TV 채널이 내려간다.");
    }
    public void internet(){
        System.out.println("인터넷이 된다.");
    }
}
  
```

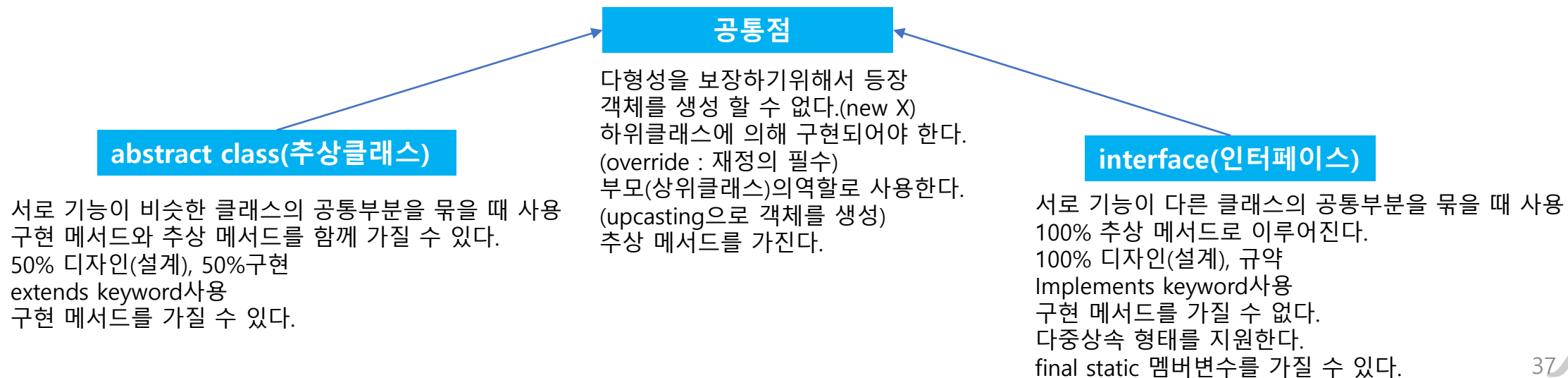
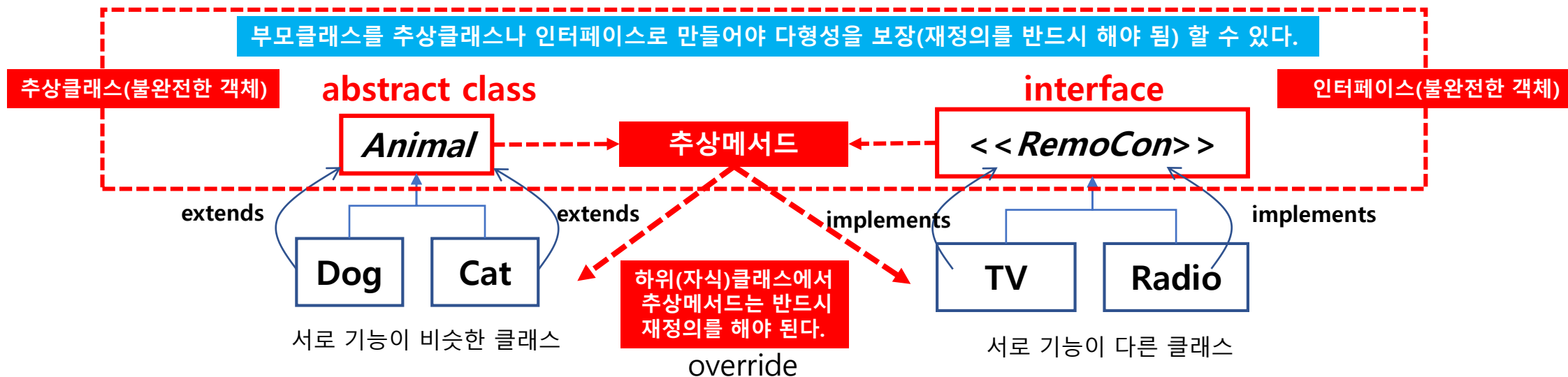
override

```

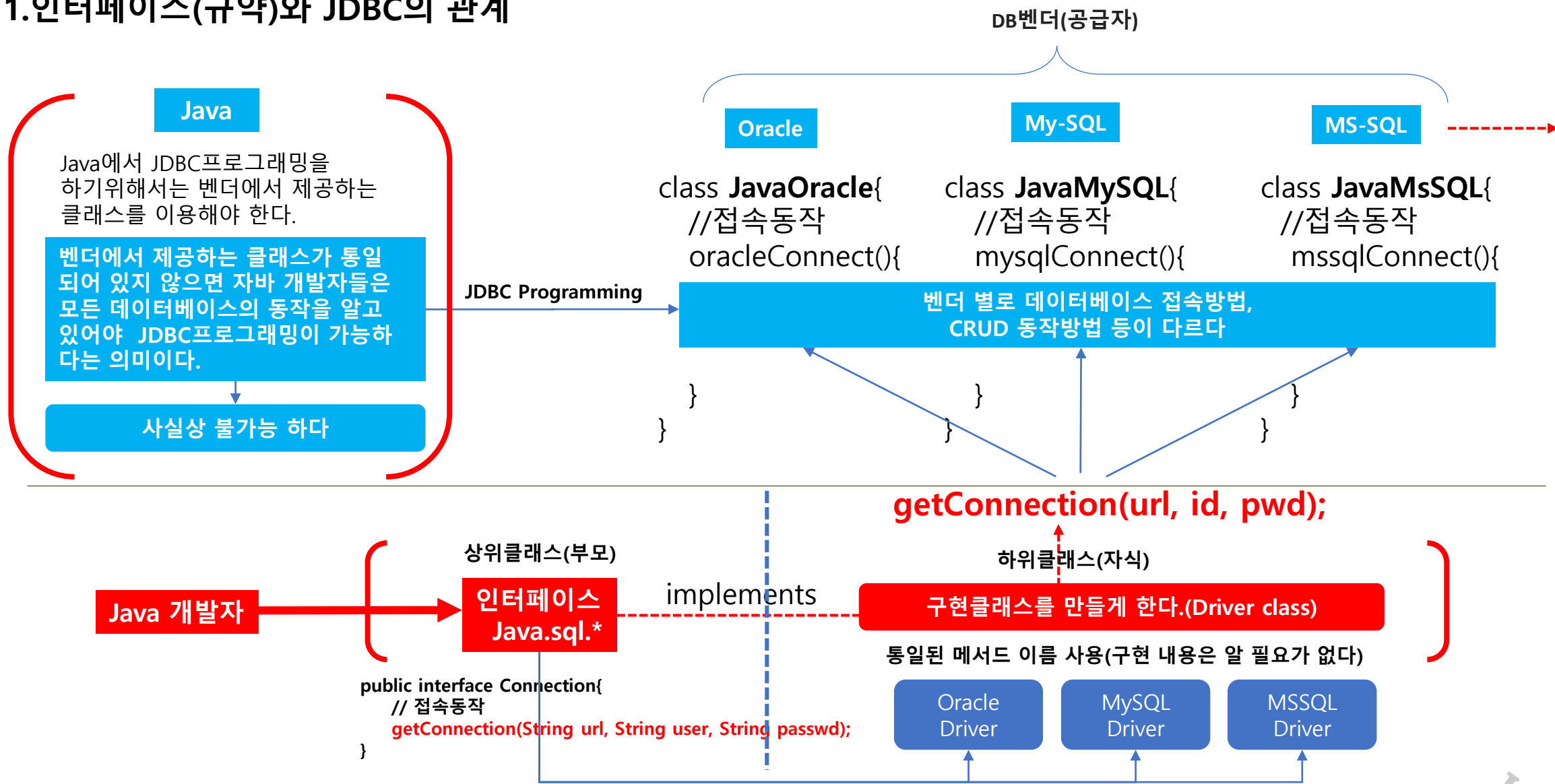
public class Radio implements RemoCon {
    public void chUp(){
        System.out.println("Radio 채널이 올라간다.");
    }
    public void chDown(){
        System.out.println("Radio 채널이 내려간다.");
    }
    public void internet(){
        System.out.println("인터넷이 지원되지 않는다.");
    }
}
  
```

override

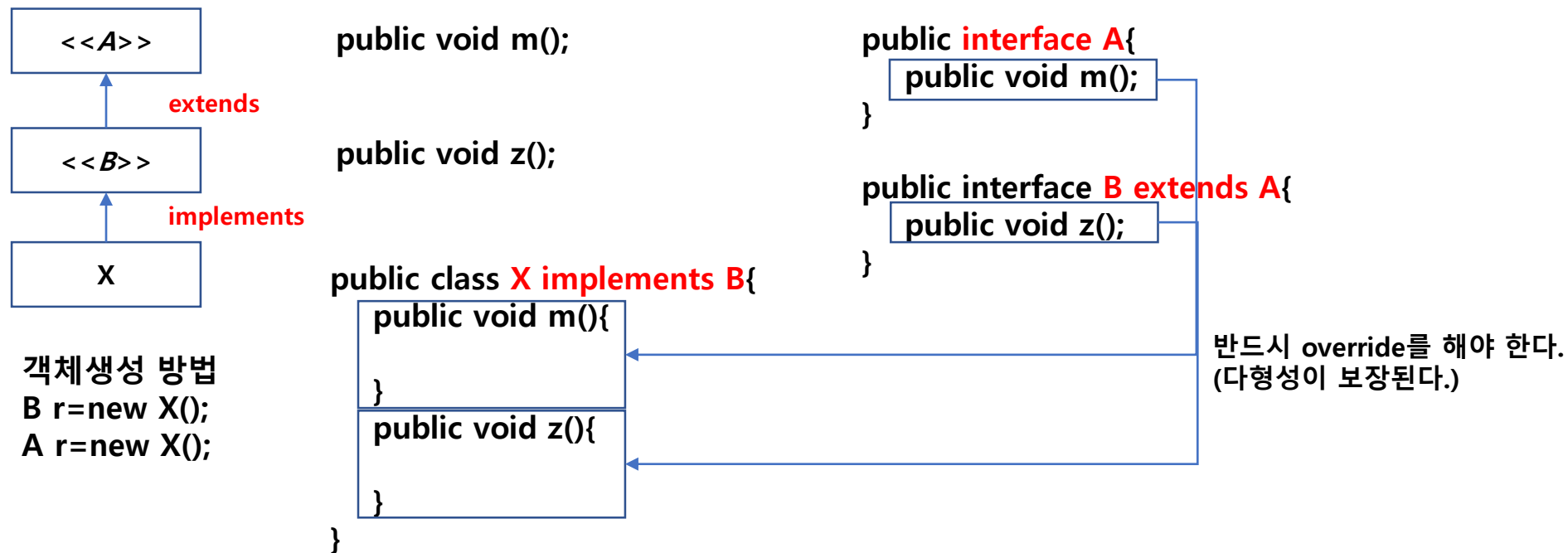
## 1. 추상클래스와 인터페이스(다형성을 보장하기 위함)



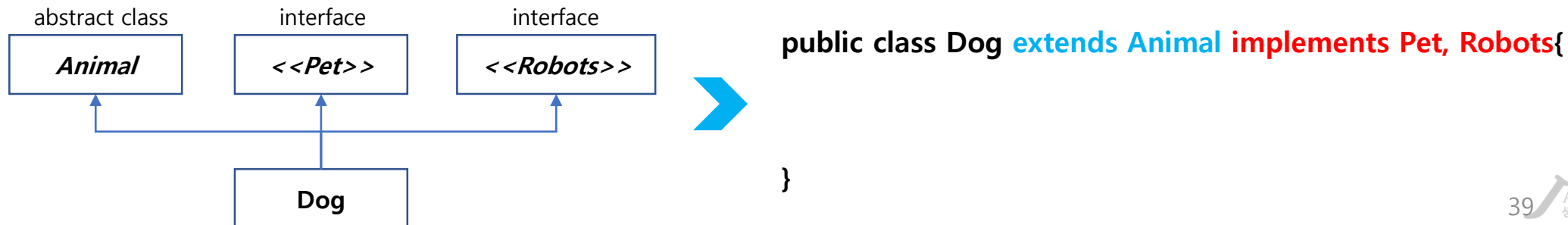
## 1. 인터페이스(규약)와 JDBC의 관계



## 1. 인터페이스와 인터페이스의 상속관계



## 2. 다중상속관계에 있는 클래스 구조



## 1. Object class, toString()

클래스를 설계할 때 기본적으로 생략된 코드(3곳)

```

import java.lang.*;
public class A extends Object {

    public A(){
        super();
    }
    public void display(){
        System.out.println("나는 A이다.");
    }
    @Override
    public String toString(){
        return "재정의 메서드 입니다.";
    }
}

```

Object  
toString()  
Override

Object class를 이용한 객체 생성

```

Object obj=new A();
obj.display();

```

```

A a=(A)obj;
a.display();

```

```

System.out.println(obj.toString());

```

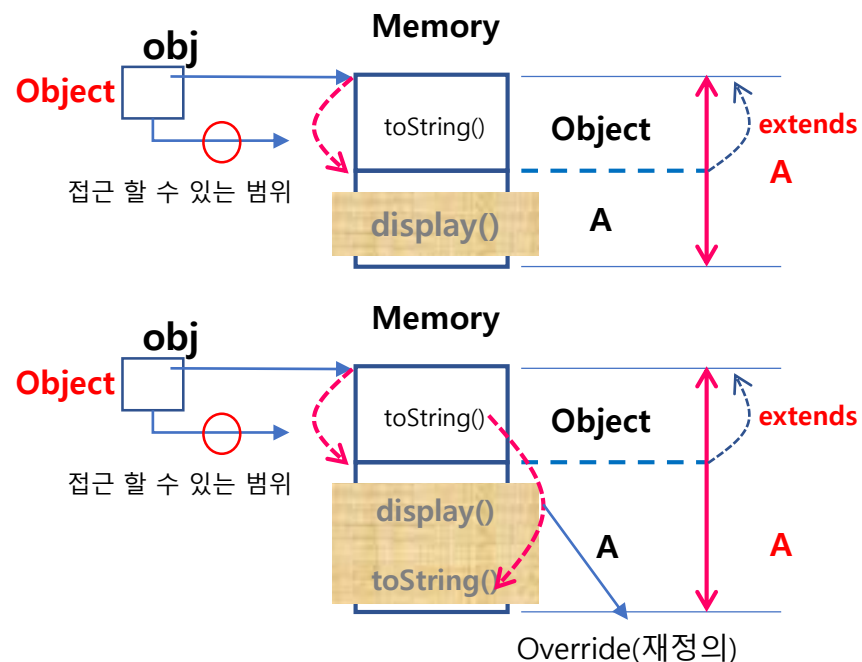
toString()

1. 재정의 안 했을 경우(번지 출력)
2. 재정의 했을 경우(재정의된 메서드 실행)

## Object class

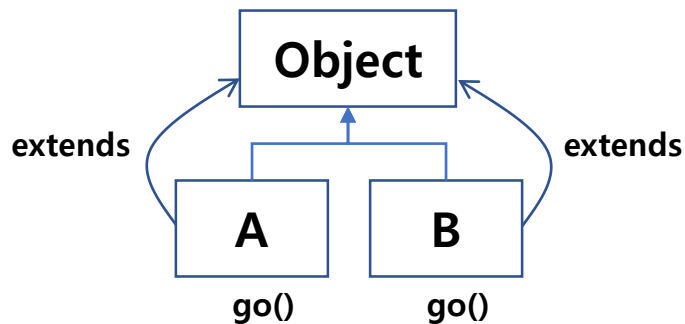
→ 모든 클래스의 root 클래스

- 최상위 클래스(상속관계에서)
- Object 클래스를 잘 활용하면 프로그램을 유연하게 만들 수 있다.





## 1.Object class의 활용(다형성 인수, 다형성 배열)



## 다형성 인수

```

public class ObjectTest{
    public static void main(String args){
        display(new A());
        display(new B());
    }
    public void display(Object o){
        if(o instanceof A){
            ((A)o).go();
        }else{
            ((B)o).go();
        }
    }
}
  
```

다형성인수

```

Object obj1=new A();
Object obj2=new B();
  
```

upcasting

```

A a=(A)obj1;
B b=(B)obj2;
  
```

downcasting

## 다형성 배열

서로 다른 객체를(A,B) 배열에 저장 하려면 부모객체(Object)를 활용하라

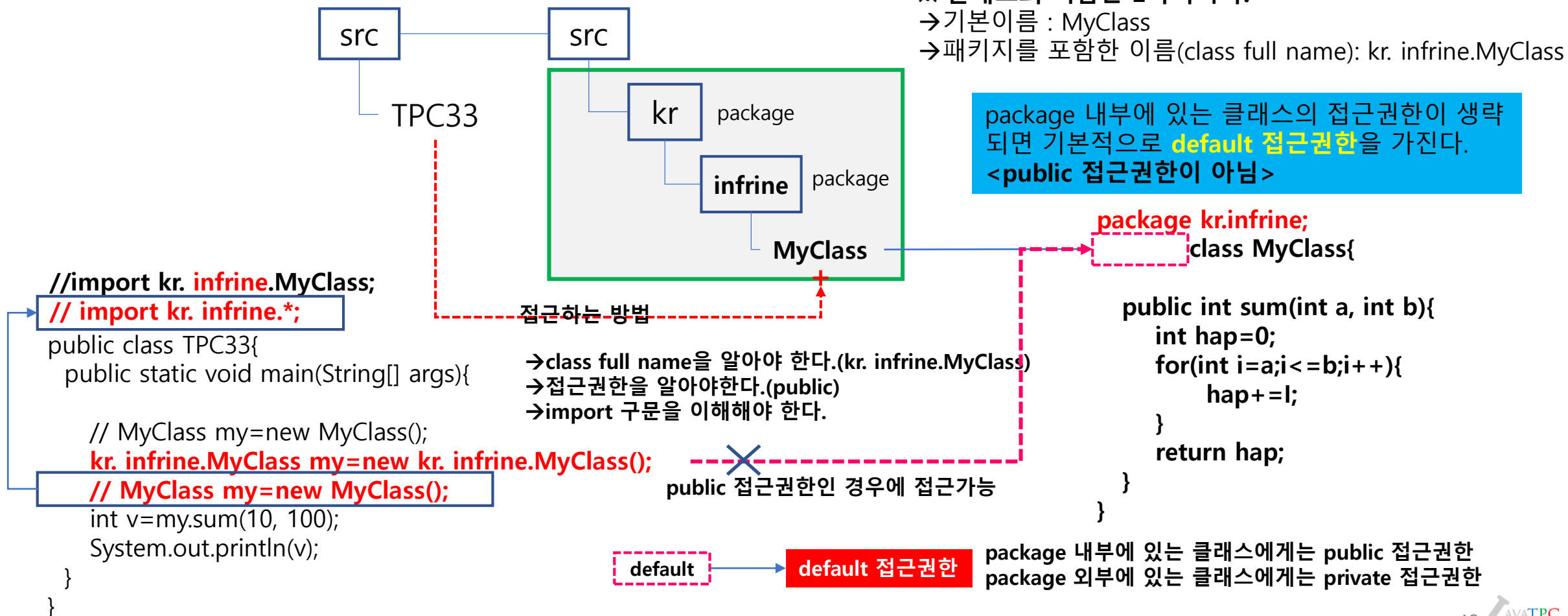
```

public class ObjectTest{
    public static void main(String args){
        Object[] o=new Object[2];
        o[0]=new A();
        o[1]=new B();
        for(int i=0;i<o.length;i++){
            if(o[i] instanceof A){
                ((A)o[i]).go();
            }else{
                ((B)o[i]).go();
            }
        }
    }
}
  
```

## 1.package(패키지)

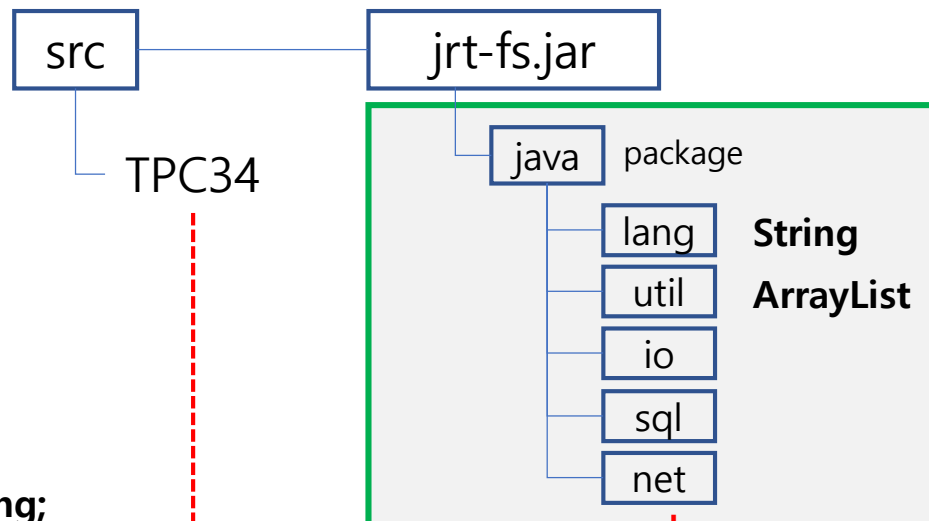
→ 기능이 비슷한 클래스를 모아서 관리하기 쉽게 하기 위해 / package 외부에서 접근하는 것을 막을 때 사용

### package 외부(outside)에서 접근하는 방법



## 1. Java API 접근하기(Java API : 2단계 package로 구성됨)

Java API 中 String 클래스를 접근하는 방법



접근하는 방법

※ String 클래스의 이름은 2가지이다.

→ 기본이름 : String

→ 패키지를 포함한 이름(class full name): java.lang.String

java.lang package는 디폴트 package로 클래스를 만들면 자동으로 import 된다.  
**import java.lang.\*;**

```
//import java.lang.String;
// import java.lang.*;
public class TPC34{
    public static void main(String[] args){
        // String str=new String("APPLE");
        java.lang.String str=new java.lang.String("APPLE");
        // String str=new String("APPLE");
        String v=str.toLowerCase();
        System.out.println(v);
    }
}
```

```
package java.lang;
public class String{
```

```
    public String toLowerCase(){
    }
    public int length(){
    }
    public char charAt(int pos){
    }
}
```

## 1.String 클래스

- 자바에서 문자열은 쌍 따옴표 " " 로 감싸면 된다.
- 자바에서 문자열을 저장하는 기본 자료형(**Data Type**)은 없다.
- **문자열은 여러가지 조작**을 할 수 있기 때문에 별도의 클래스로 자료형(`java.lang.String`)을 만들어 두었다.
- 그래서 자바에서 **문자열**은 책, 영화, 회원 처럼 **객체**로 취급된다.

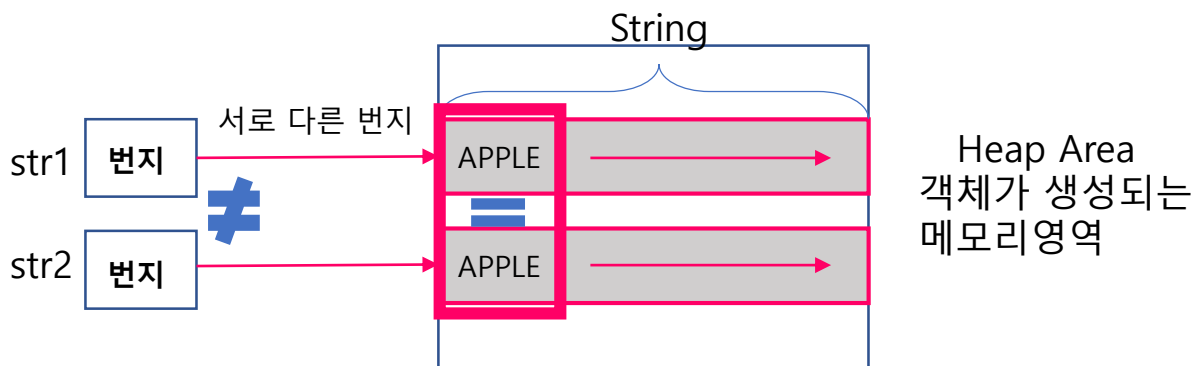
"APPLE", "홍길동"

### 문자열 생성방법

#### 1. new로 생성

```
String str1=new String("APPLE");
```

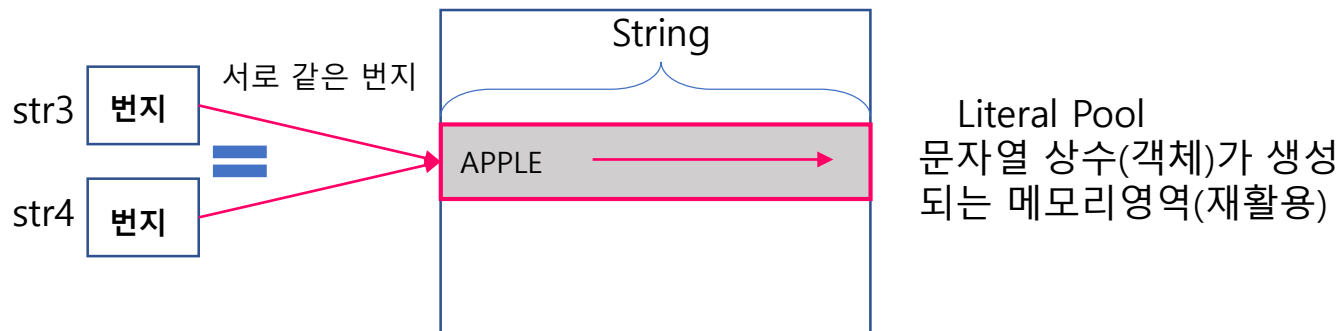
```
String str2=new String("APPLE");
```



#### 2. 문자열 상수로 생성

```
String str3= "APPLE";
```

```
String str4= "APPLE";
```



## 1. 배열처럼 동작하는 클래스 만들기(IntArray)

## 배열 기본 동작의 이해

```
public class TPC36{
```

```
    public static void main(String[] args){
```

```
        int[] a=new int[3];
```

1

배열생성동작

```
        a[0]=10;
```

```
        a[1]=20;
```

```
        a[2]=30;
```

2

추가 동작

```
        int v=a[1];
```

```
        int len=a.length;
```

3

읽는 동작

4

배열크기

```
        for(int i=0;i<a.length;i++){
            System.out.println(a[i]);
        }
    }
}
```

## 클래스(API)로 만들어 놓음

```
public class IntArray{
```

```
    private int count;
```

```
    private int[] arr;
```

```
    public IntArray(){
```

```
        this(10);
```

```
    }
```

```
    public IntArray(int init){
```

```
        arr=new int[init];
```

```
    }
```

```
    public void add(int data){
```

```
        arr[count++] = data;
```

```
    }
```

```
    public int get(int index){
```

```
        return arr[index];
```

```
    }
```

```
    public int size()
```

```
        return count;
```

```
    }
```

```
}
```

```
public class TPC36{
```

```
    public static void main(String[] args){
```

```
        IntArray arr=new IntArray(3);
```

```
        arr.add(10);
```

```
        arr.add(20);
```

```
        arr.add(30);
```

```
        for(int i=0;i<arr.size();i++){
```

```
            System.out.println(arr.get(i));
```

```
        }
```

```
    }
```

```
}
```

## 학습정리

## ▷ 객체지향 프로그래밍의 3대 특징

- 정보은닉(Information Hiding)
- 상속(Inheritance)
- 다형성(polymorphism)

## ▷ message polymorphism(다형성)

- 상속관계에 있는 클래스에서 상위클래스가 동일한 메시지로 하위클래스들을 서로 다르게 동작시키는 객체지향 원리(개념)

## ▷ 다형성 이론의 전제조건

- 상속관계가 되어야 한다.
- 객체생성을 upcasting으로 할 것(상위클래스가 하위클래스에게 메시지를 보내야 하므로)  
(upcasting이 되면 downcasting을 할 수 있다.)
- 하위클래스가 반드시 재정의(override)해야 한다.(다형성이 보장되기 위해서는)
- 동적 바인딩을 통해 실현된다.  
(동적 바인딩 : 실행시점에서 사용될 메서드가 결정되는 바인딩, 프로그램의 속도를 떨어뜨리는 원인이 된다.)

## ▷ 추상클래스와 인터페이스의 공통점

- 다형성을 보장하기위해서 등장
- 객체를 생성 할 수 없다.(new X)
- 하위클래스에 의해 구현되어야 한다. (override : 재정의 필수)
- 부모(상위클래스)의역할로 사용한다. (upcasting으로 객체를 생성)
- 추상 메서드를 가진다.



수고하셨습니다.