

Java Programming

Basic and Advanced

장민창

mcjang@hucloud.co.kr

0. Java Programming Dev Env

0. Java Programming Dev Env

- <https://jdk.java.net/>

jdk.java.net

Production and Early-Access OpenJDK Builds, from Oracle

Ready for use: [JDK 20](#), [JDK 19](#), [JavaFX 20](#), [JMC 8](#)

Early access: [JDK 21](#), [Generational ZGC](#), [JavaFX 21](#),
[Jextract](#), [Loom](#), [Metropolis](#), [Panama](#), & [Valhalla](#)

Looking to learn more about Java? Visit [dev.java](#) for the latest Java developer news and resources.

Looking for Oracle JDK builds and information about Oracle's enterprise Java products and services? Visit the [Oracle JDK Download page](#).

0. Java Programming Dev Env

- <https://jdk.java.net/20/>

[jdk.java.net](#)

GA Releases

JDK 20

JDK 19

JavaFX 20

JMC 8

Early-Access
Releases

JDK 21

Generational ZGC

JavaFX 21

Jextract

Loom

Metropolis

Panama

Valhalla

Reference

Implementations

Java SE 20

Java SE 19

Java SE 18

Java SE 17

Java SE 16

Java SE 15

Java SE 14

Java SE 13

Java SE 12

OpenJDK JDK 20 General-Availability Release

This page provides production-ready open-source builds of the Java Development Kit, version 20, an implementation of the Java SE 20 Platform under the GNU General Public License, version 2, with the Classpath Exception.

Commercial builds of JDK 20 from Oracle, under a non-open-source license, can be found [here](#).

Documentation

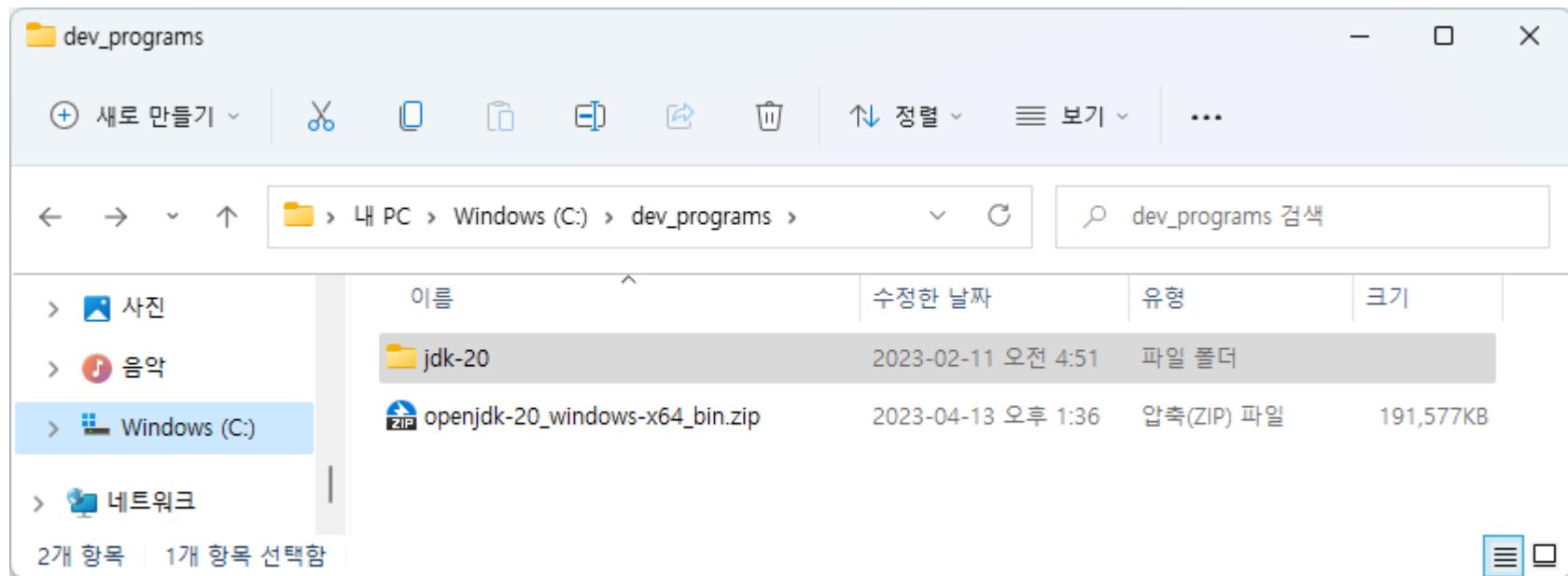
- [Features](#)
- [Release notes](#)
- [API Javadoc](#)

Builds

Linux/AArch64	tar.gz (sha256)	195952958 bytes
Linux/x64	tar.gz (sha256)	197562076
macOS/AArch64	tar.gz (sha256)	191881541
macOS/x64	tar.gz (sha256)	194328775
Windows/x64	zip (sha256)	196174504

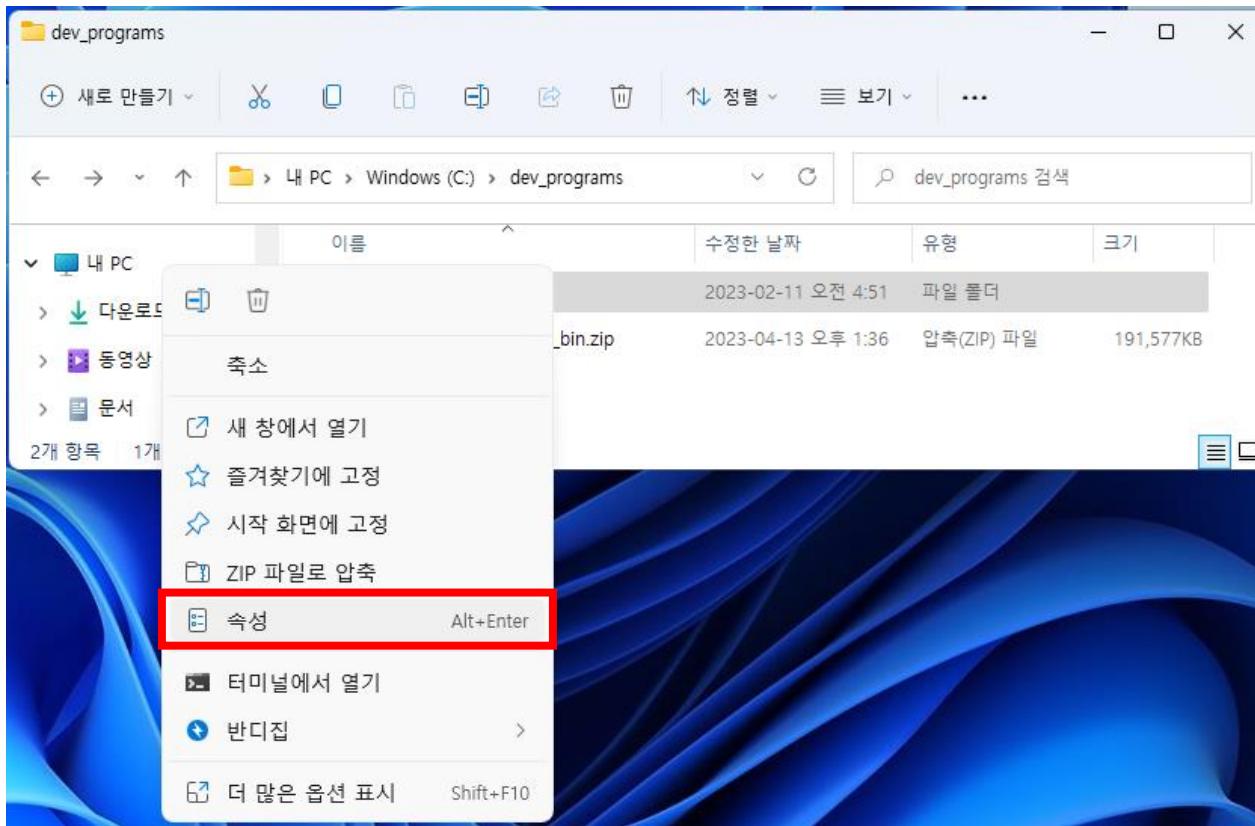
0. Java Programming Dev Env

- C:\dev_program 에 복사 후 압축 풀기



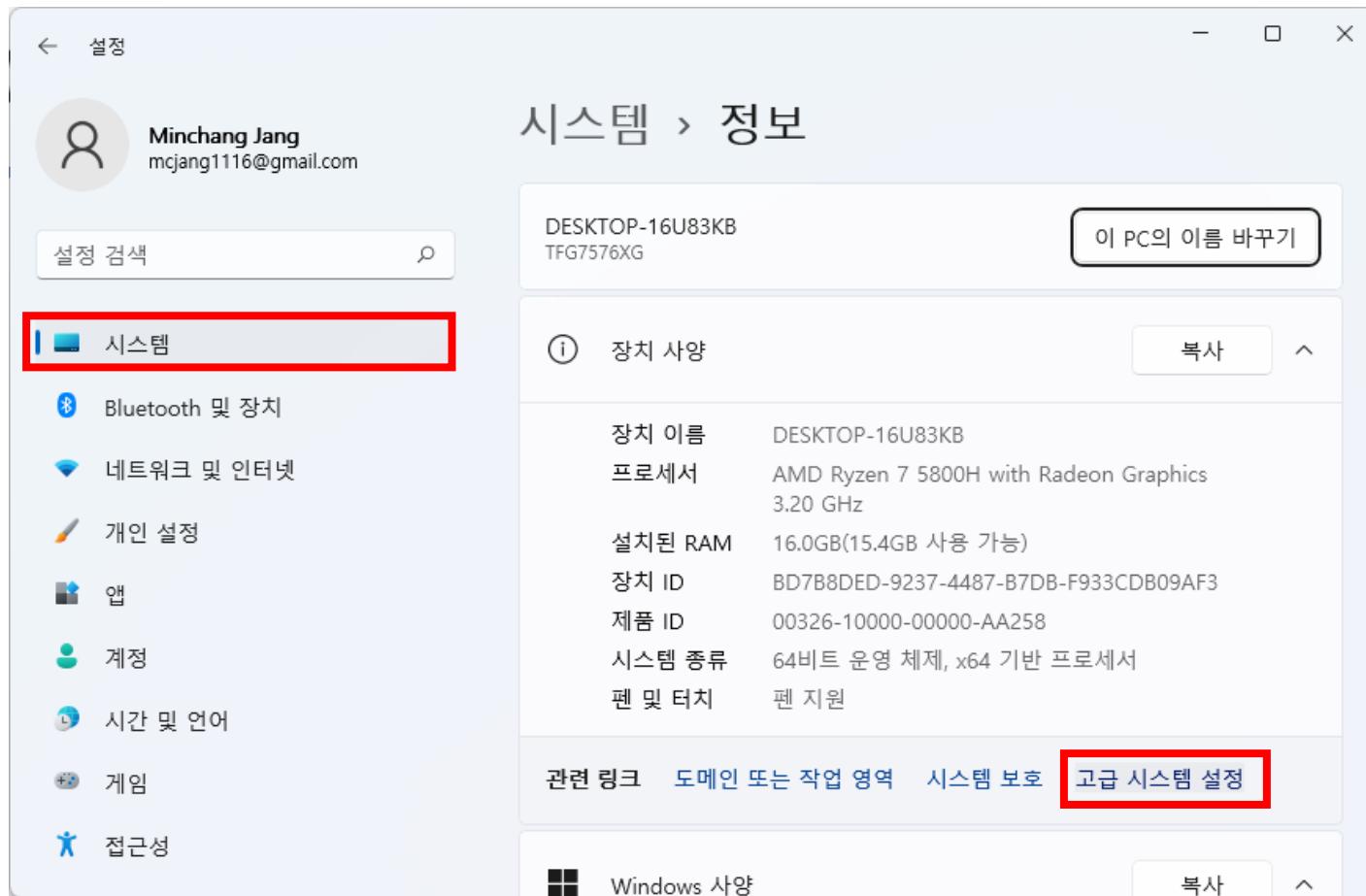
0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 내 PC 오른쪽클릭 → 속성



0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 시스템 → 고급 시스템 설정



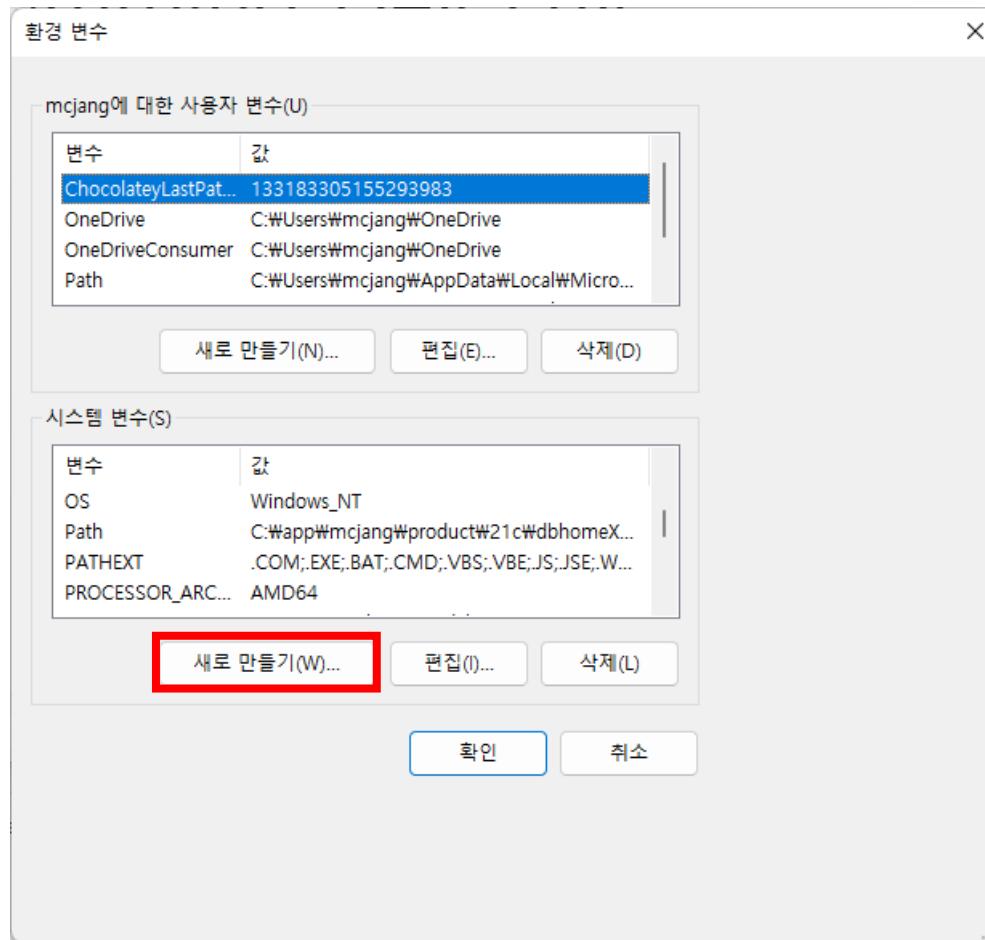
0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 고급 → 환경 변수(N)...



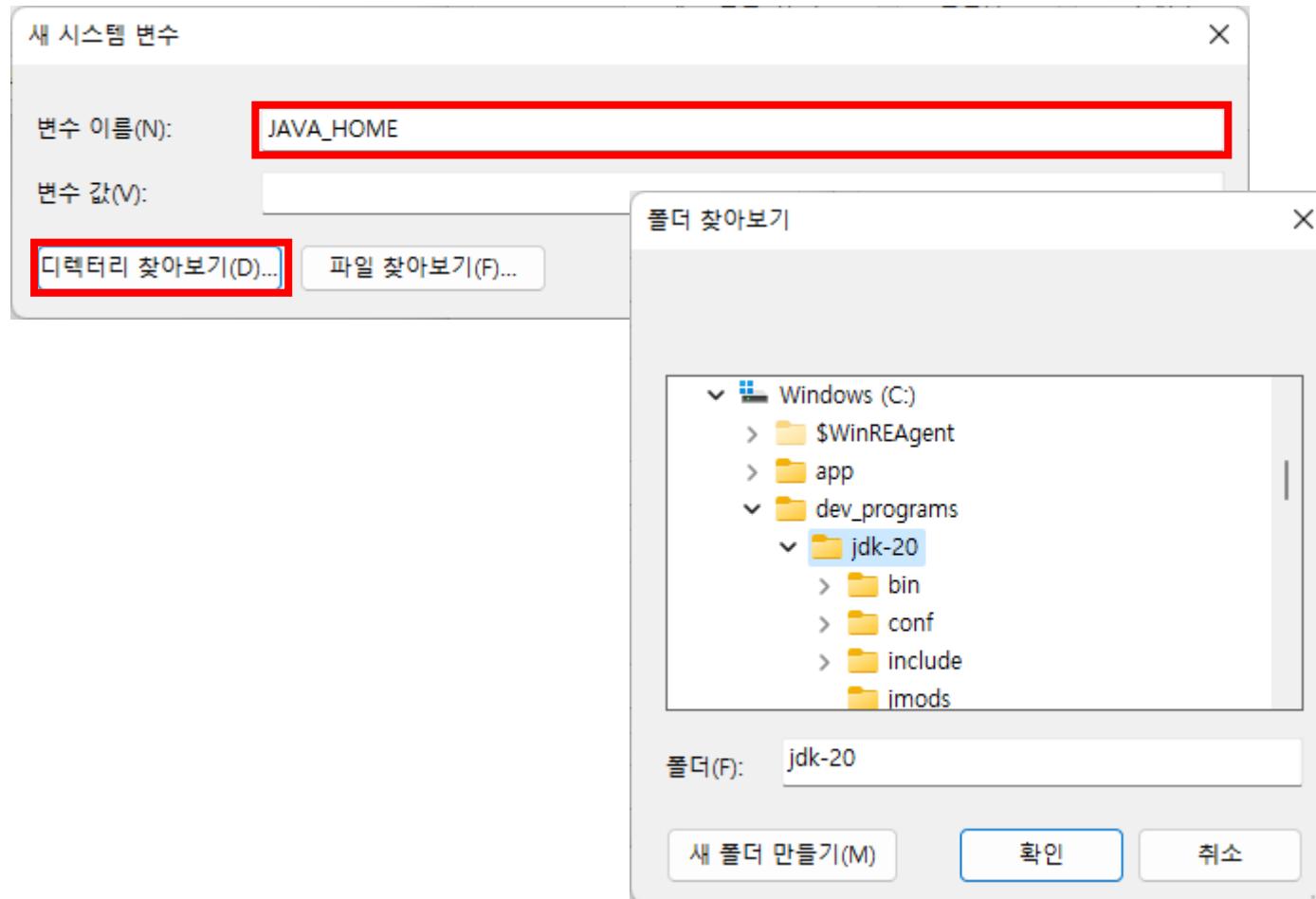
0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 고급 → 환경 변수(N)...



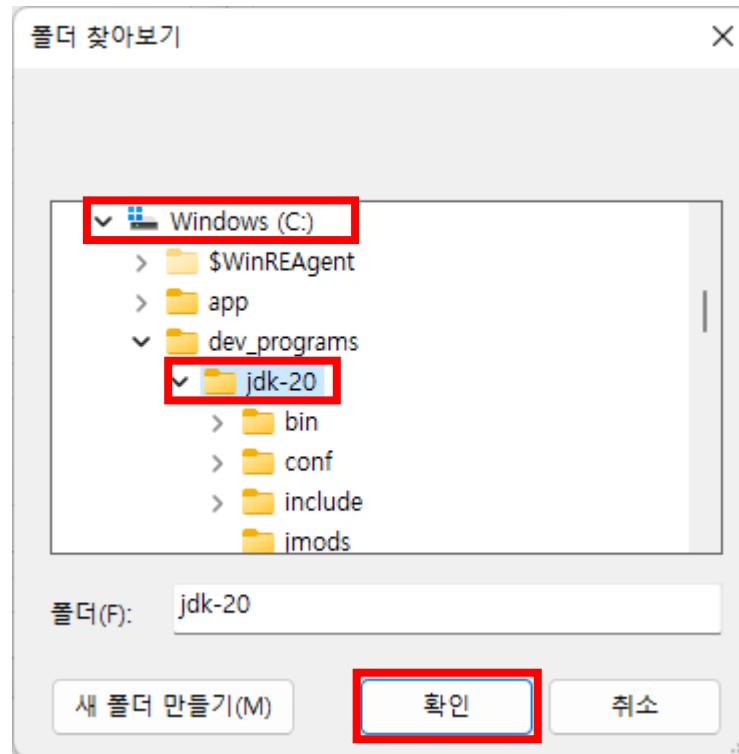
0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 변수 이름(N): JAVA_HOME



0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 변수 이름(N): JAVA_HOME



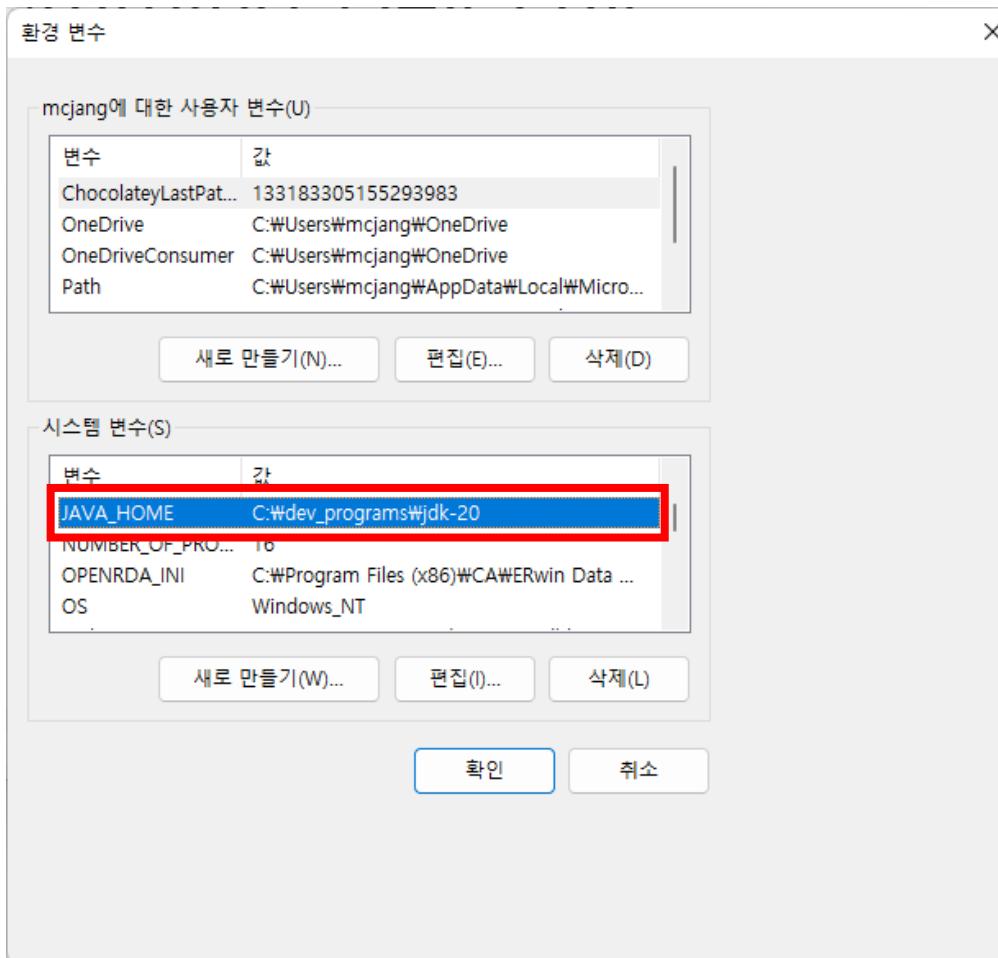
0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 변수 값(V)에 입력된 것 확인 후 “확인” 버튼 클릭



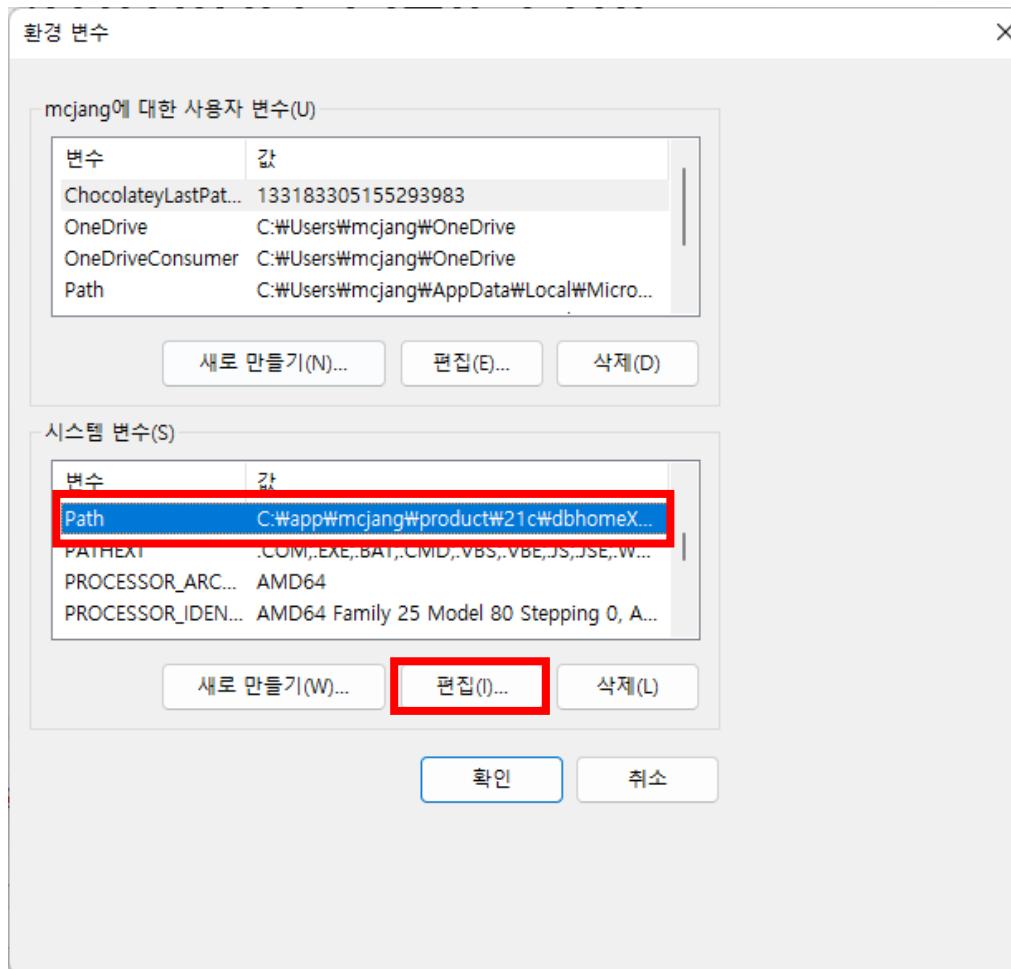
0. Java Programming Dev Env

- 시스템 변수에 JAVA_HOME Path 등록하기
 - 시스템 변수 확인



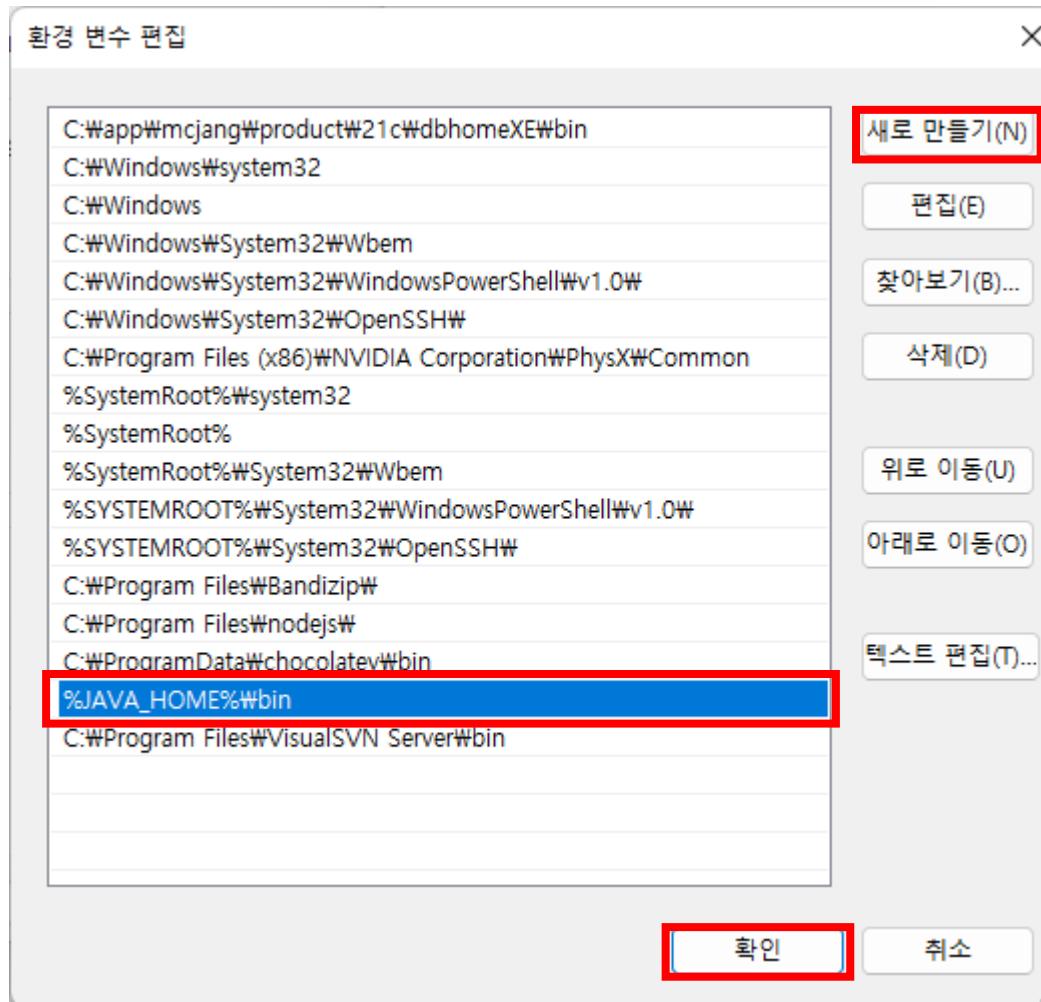
0. Java Programming Dev Env

- 시스템 변수에 PATH에 JAVA_HOME 위치 등록하기
 - 시스템 변수 확인



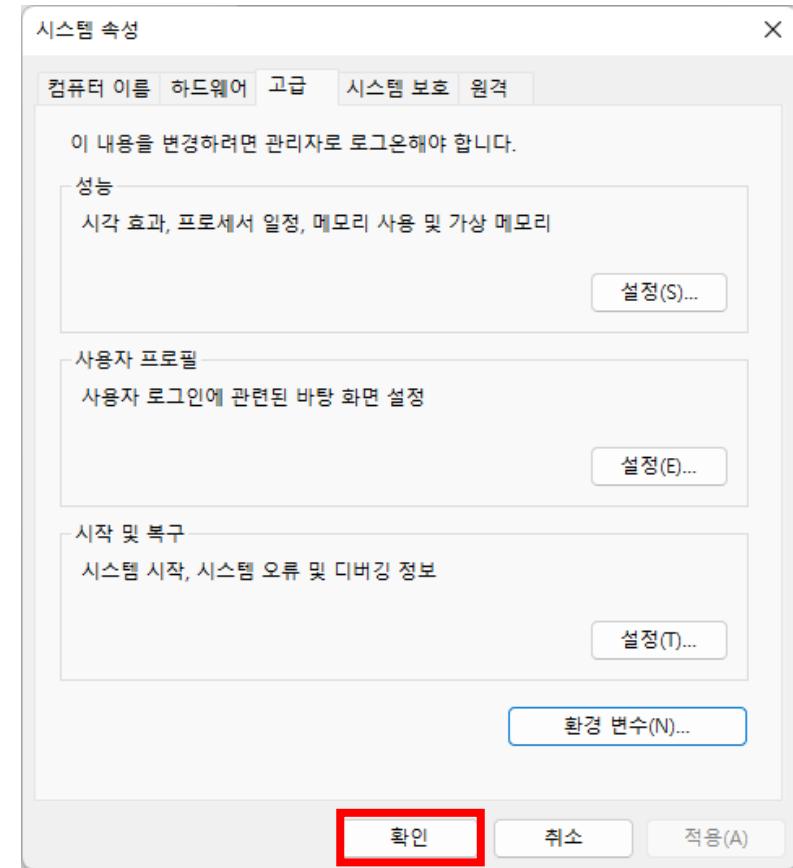
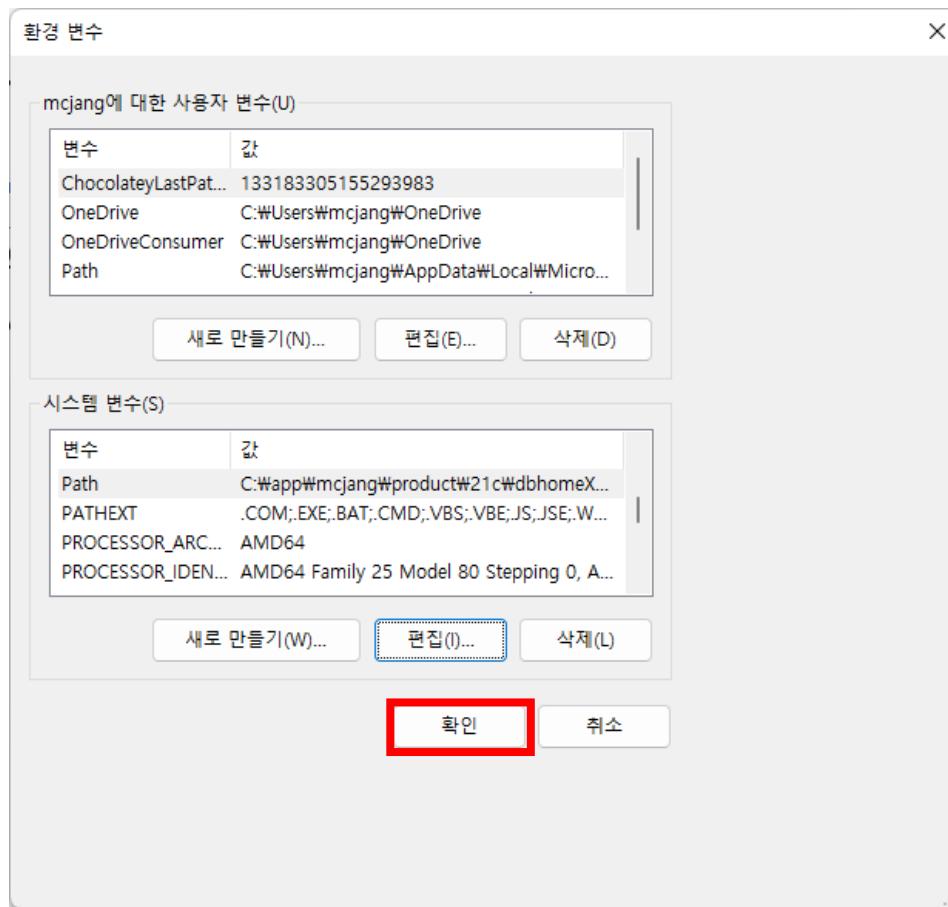
0. Java Programming Dev Env

- 시스템 변수에 PATH에 JAVA_HOME 위치 등록하기
 - 새로 만들기(N) 클릭 후 %JAVA_HOME%\bin 작성하고 확인



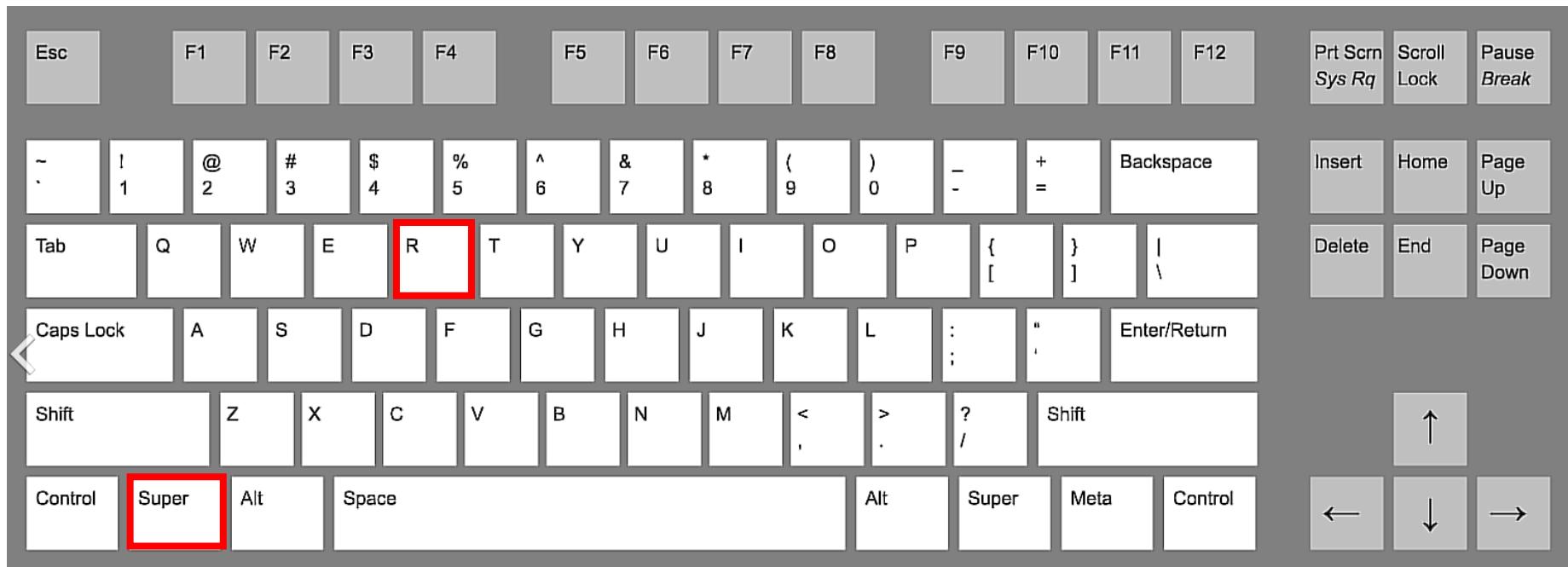
0. Java Programming Dev Env

- 시스템 변수에 PATH에 JAVA_HOME 위치 등록하기
 - 확인 버튼 클릭해서 시스템 설정 창 닫기



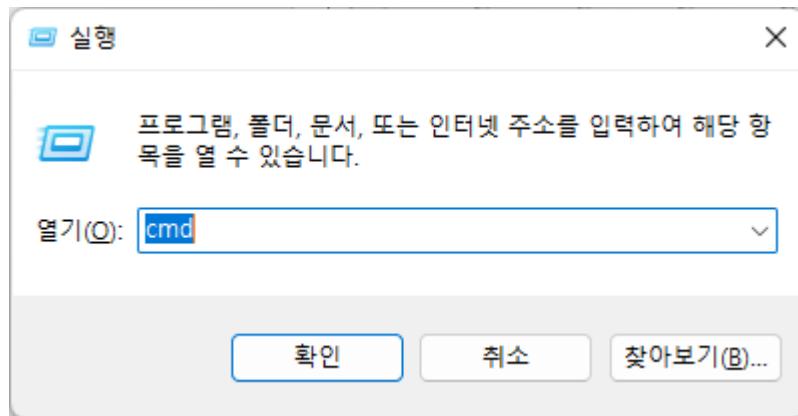
0. Java Programming Dev Env

- 자바 설치 버전 확인.
 - Windows Key + R



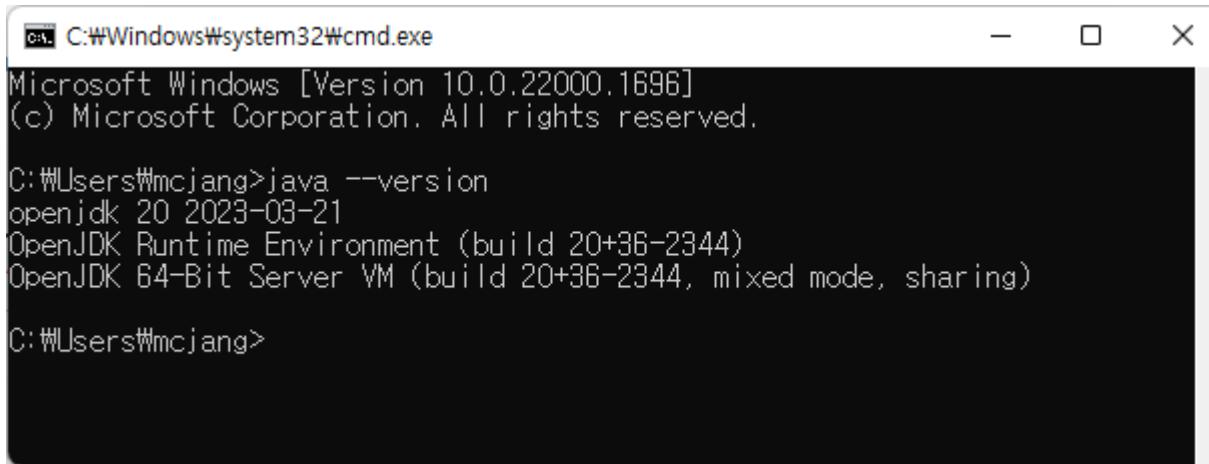
0. Java Programming Dev Env

- 자바 설치 버전 확인.
 - 열기(O): cmd 입력하고 엔터



0. Java Programming Dev Env

- 자바 설치 버전 확인.
 - java --version 입력 후 엔터



A screenshot of a Windows Command Prompt window titled "cmd.exe". The window shows the following text:

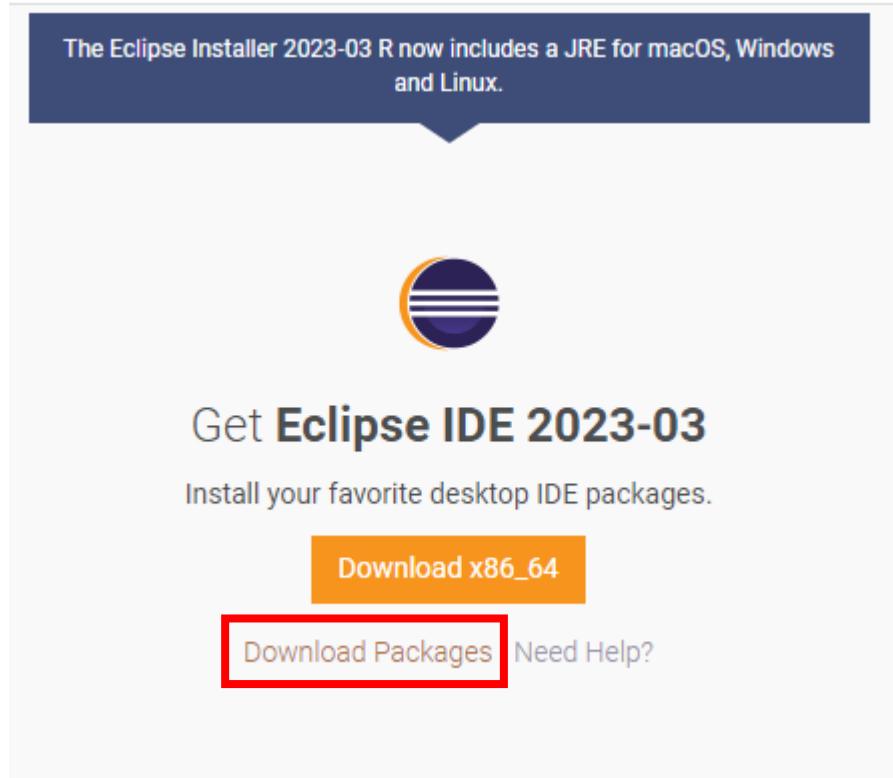
```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.22000.1696]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mcjang>java --version
openjdk 20 2023-03-21
OpenJDK Runtime Environment (build 20+36-2344)
OpenJDK 64-Bit Server VM (build 20+36-2344, mixed mode, sharing)

C:\Users\mcjang>
```

0. Java Programming Dev Env

- 자바 Editor 설치 (Eclipse)
 - <https://www.eclipse.org/downloads/>



0. Java Programming Dev Env

- 자바 Editor 설치 (Eclipse)
 - <https://www.eclipse.org/downloads/>

Eclipse IDE 2023-03 R Packages

Eclipse IDE for Java Developers



306 MB 130,731 DOWNLOADS

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

Eclipse IDE for Enterprise Java and Web Developers



518 MB 103,964 DOWNLOADS

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.

Click here to open a bug report with the Eclipse Web Tools Platform.
Click here to raise an issue with the Eclipse Platform.
Click here to raise an issue with Maven integration for web projects.
Click here to raise an issue with Eclipse Wild Web Developer (incubating).

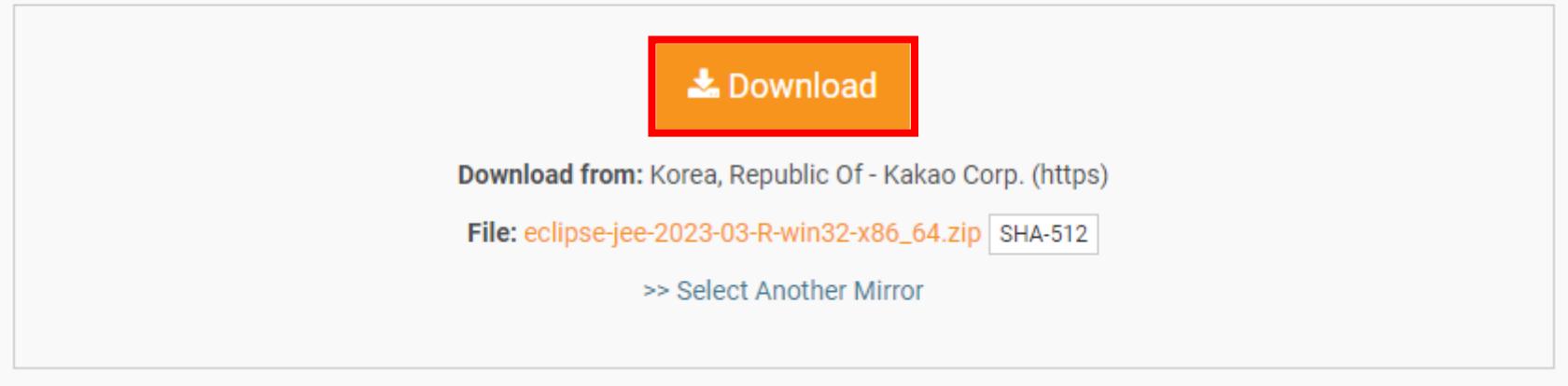
 Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64

 Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64

0. Java Programming Dev Env

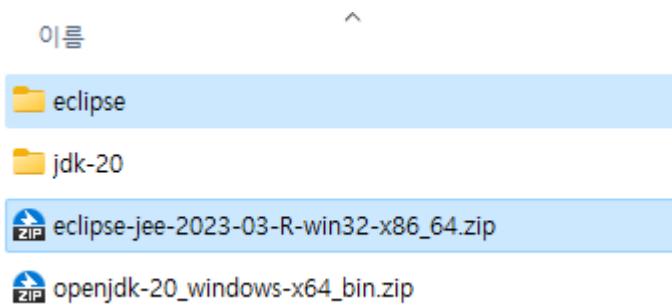
- 자바 Editor 설치 (Eclipse)
 - <https://www.eclipse.org/downloads/>

All downloads are provided under the terms and conditions of the Eclipse Foundation Software User Agreement unless otherwise specified.



0. Java Programming Dev Env

- C:\dev_programs 에 압축 풀기

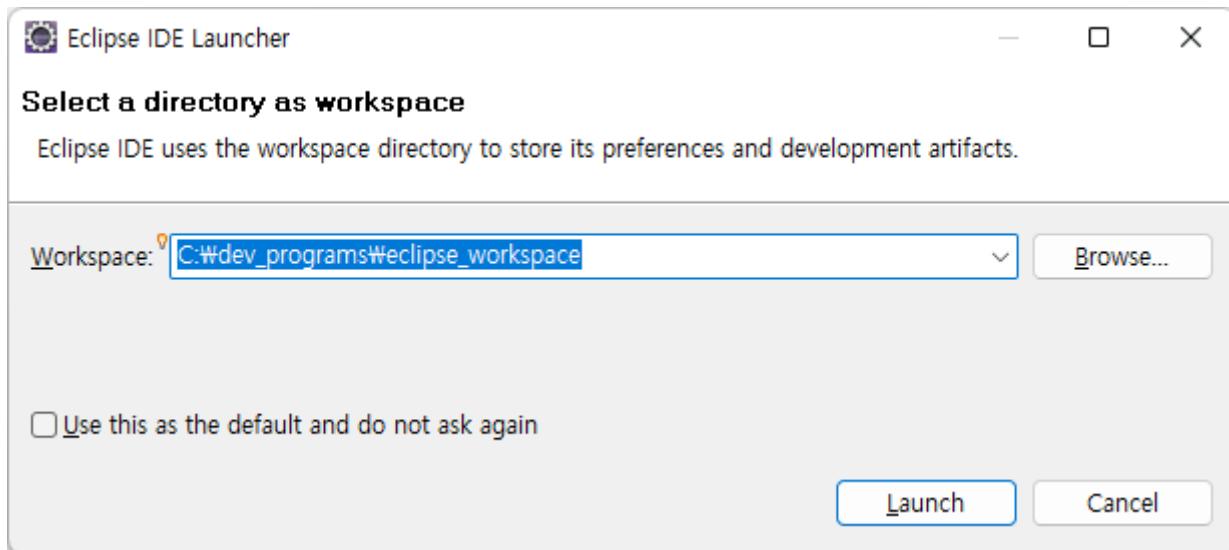


- eclipse 폴더의 eclipse.exe 파일 실행

이름	수정한 날짜	유형	크기
configuration	2023-03-09 오후 4:22	파일 폴더	
dropins	2023-03-09 오후 4:22	파일 폴더	
features	2023-03-09 오후 4:22	파일 폴더	
p2	2023-03-09 오후 4:21	파일 폴더	
plugins	2023-03-09 오후 4:22	파일 폴더	
readme	2023-03-09 오후 4:22	파일 폴더	
.eclipseproduct	2023-03-02 오전 8:14	ECLIPSEPRODUCT...	1KB
artifacts.xml	2023-03-09 오후 4:22	XML 문서	595KB
eclipse.exe	2023-03-09 오후 4:28	응용 프로그램	521KB
eclipse.ini	2023-03-09 오후 4:22	구성 설정	1KB
eclipsec.exe	2023-03-09 오후 4:28	응용 프로그램	233KB

0. Java Programming Dev Env

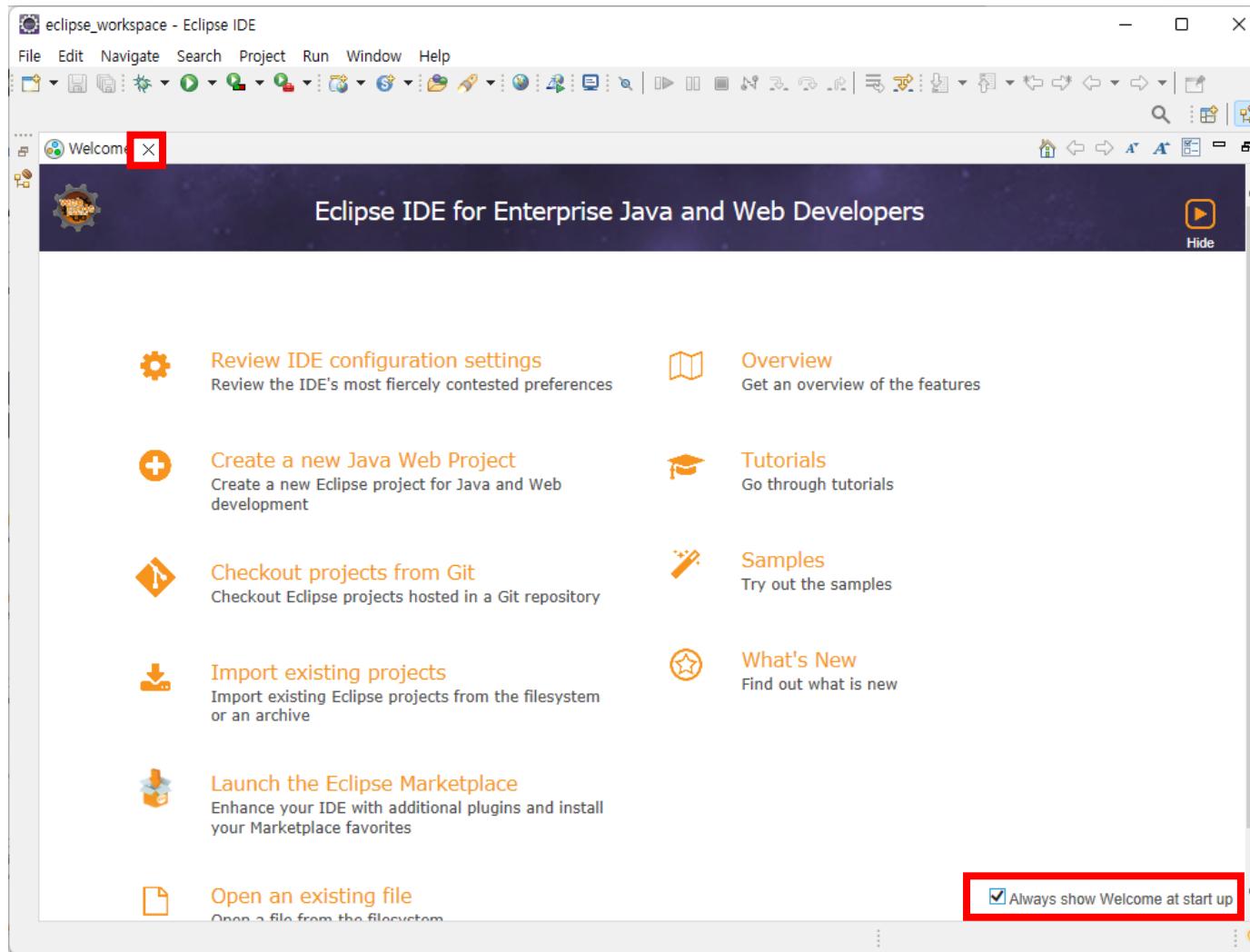
- 이클립스 실행



- workspace를 "C:\dev_programs\workspace"로 지정 후 Launch 클릭

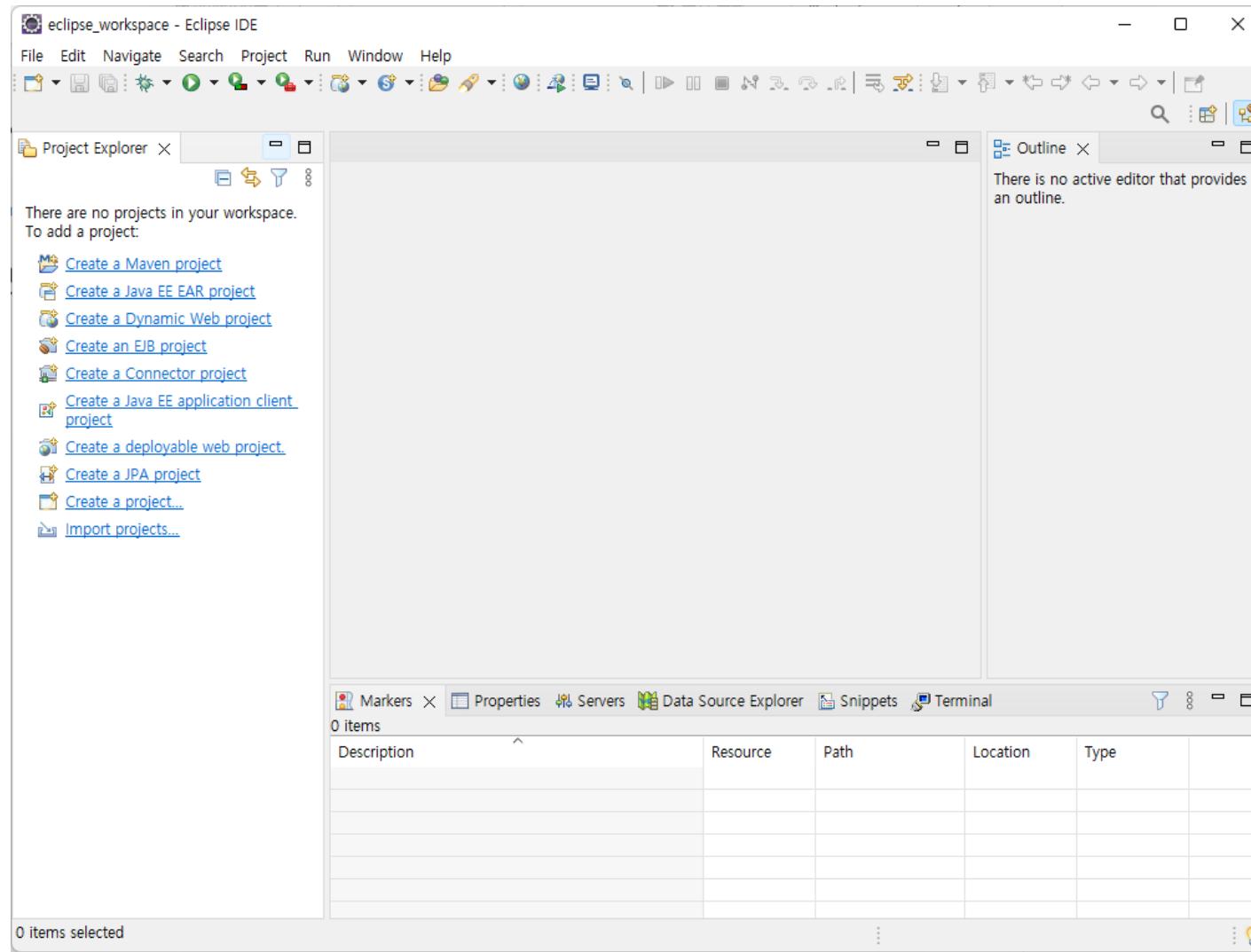
0. Java Programming Dev Env

- Welcome Page 제거

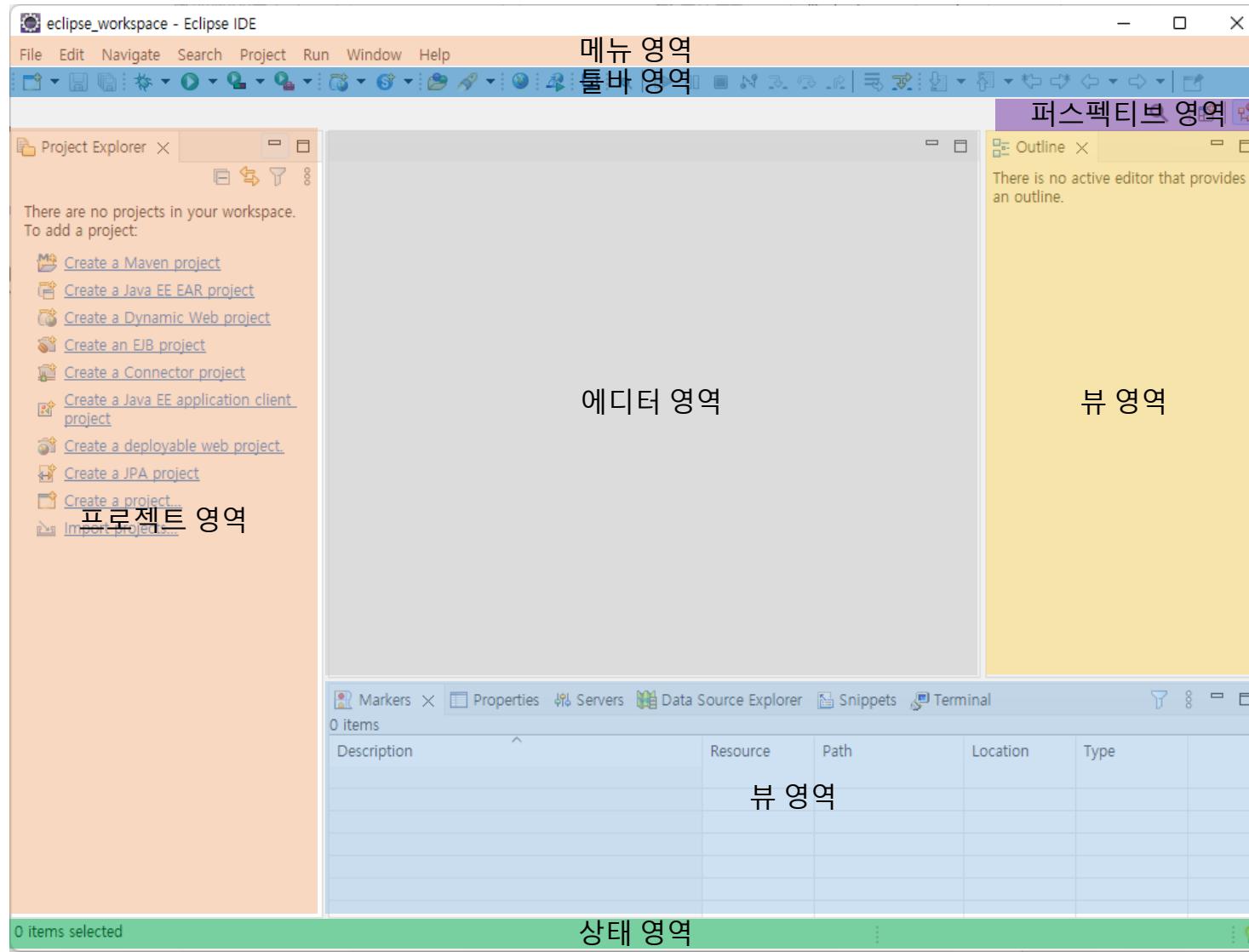


0. Java Programming Dev Env

- Welcome Page 제거 후 레이아웃

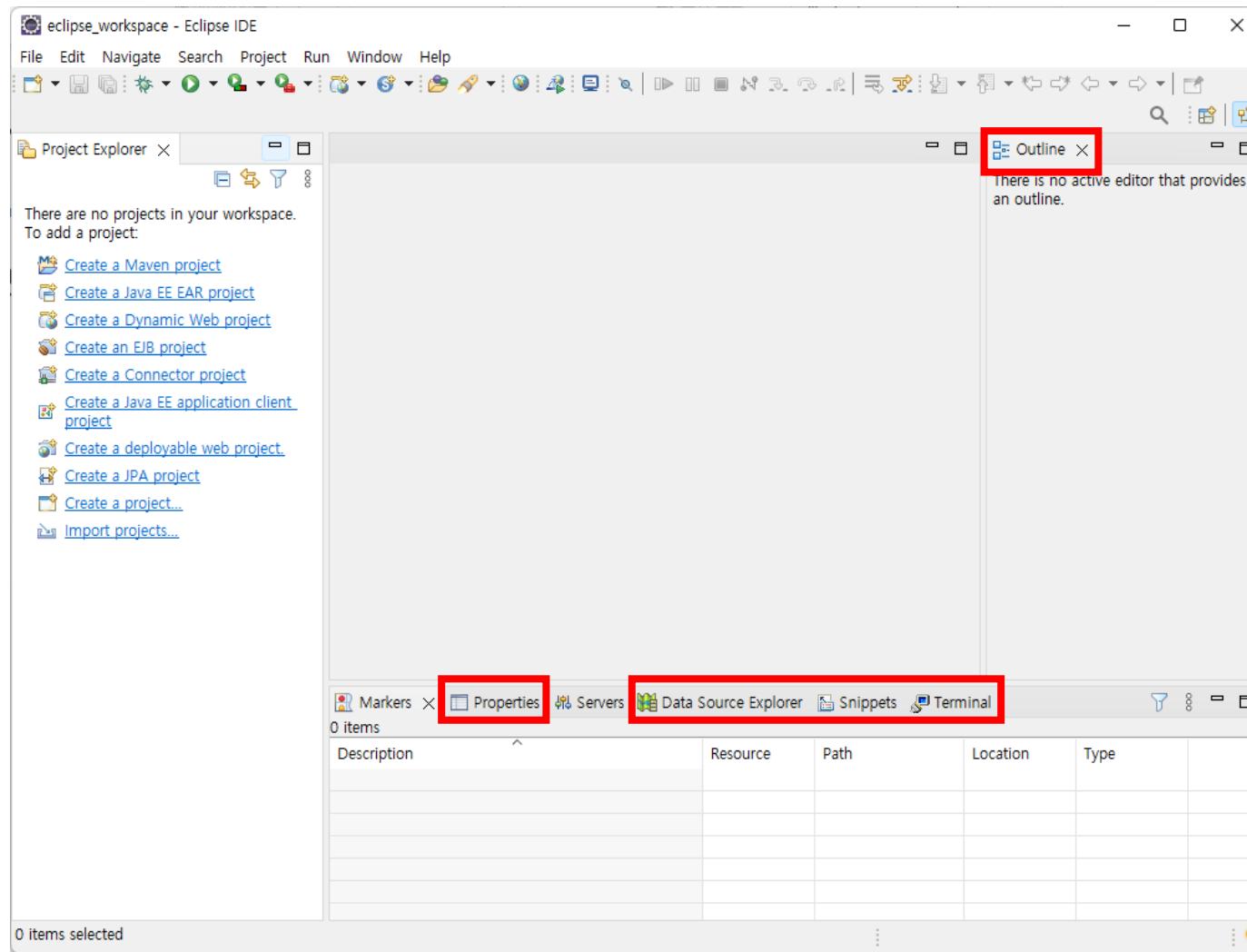


0. Java Programming Dev Env



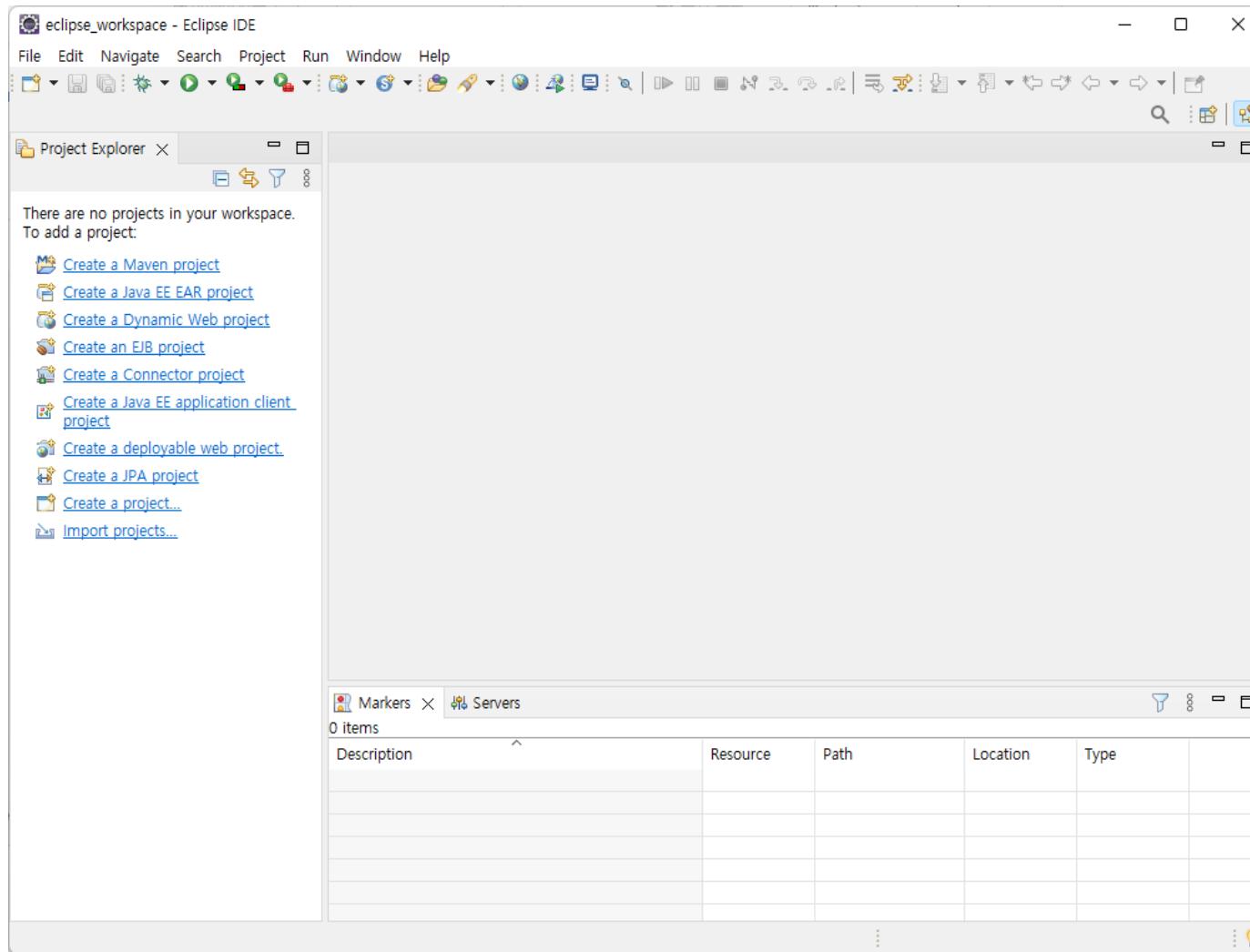
0. Java Programming Dev Env

- 필요없는 View 제거



0. Java Programming Dev Env

- 필요없는 View 제거 후 레이아웃



0. Java Programming Dev Env

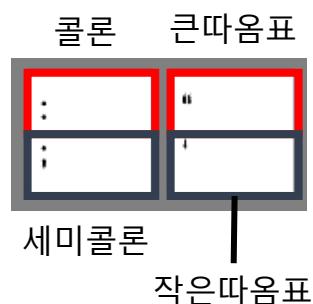
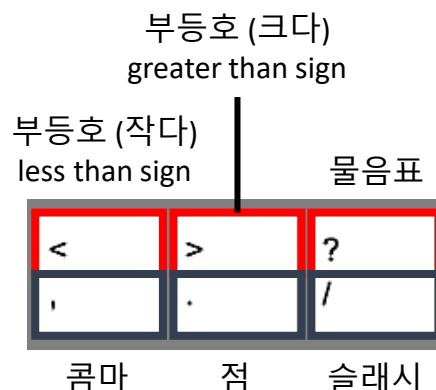
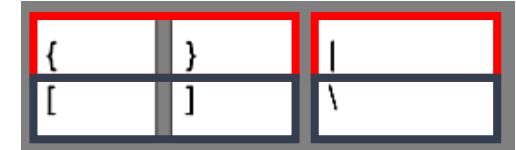
- 키 명칭



백틱

대수
하이픈
는
equal sign

중괄호
Curly Brackets
파이프



대괄호
Square Brackets
역슬래시
백슬래시

0. Java Programming Dev Env

- 자주 사용하는 이클립스 단축키

파일관련 단축키

설명	단축키	설명	단축키
저장	<code>Ctrl + S</code>	모든 파일 저장	<code>Ctrl + Shift + S</code>

에디터 탭 관련 단축키

설명	단축키	설명	단축키
파일 탭 이동	<code>Ctrl + PgUp, Ctrl + PgDn</code>	에디터 탭 닫기	<code>Ctrl + W</code>
최대화 / 되돌리기	<code>Ctrl + M</code>	텍스트 크기 조절	<code>Ctrl + +, Ctrl + -</code>
파일나누어 보기 / 해제	<code>Ctrl + Shift + -</code>		

검색관련 단축키

설명	단축키	설명	단축키
파일 내용 찾기	<code>Ctrl + F, Ctrl + H</code>	파일 찾기	<code>Ctrl + Shift + R</code>
같은 단어 찾기 (Forward)	<code>Ctrl + K</code>	같은 단어 찾기 (Backward)	<code>Ctrl + Shift + K</code>
호출관계 추척하기	<code>Ctrl + Alt + H</code>		

0. Java Programming Dev Env

- 자주 사용하는 이클립스 단축키

코드 복사/이동 관련

설명	단축키	설명	단축키
라인복사(위로)	<code>Ctrl + Alt + ↑</code>	라인복사(아래로)	<code>Ctrl + Alt + ↓</code>
코드라인 이동	<code>Alt + ↑, Alt + ↓</code>	잘라내기	<code>Ctrl + X</code>
복사	<code>Ctrl + C</code>	붙여넣기	<code>Ctrl + V</code>

커서 이동 관련

설명	단축키	설명	단축키
줄 끝으로 이동	<code>End</code>	줄 앞으로 이동	<code>Home</code>
파일 끝으로 이동	<code>Ctrl + End</code>	파일 처음으로 이동	<code>Ctrl + Home</code>
단어별 건너뛰기	<code>Ctrl + →, Ctrl + ←</code>	지정 라인으로 이동	<code>Ctrl + L</code>

0. Java Programming Dev Env

- 자주 사용하는 이클립스 단축키

코드 편집 관련

설명	단축키	설명	단축키
모든 내용 선택	<code>Ctrl + A</code>	여러 줄 선택	<code>Shift + ↑, Shift + ↓</code>
한 줄 전체 선택	<code>Shift + End</code>	한 줄 전체 선택	<code>Shift + Home</code>
현재 커서에서 파일 끝까지 선택	<code>Ctrl + Shift + End</code>	현재 커서에서 파일 처음까지 선택	<code>Ctrl + Shift + Home</code>
단어별 선택하기	<code>Ctrl + Shift + → Ctrl + Shift + ←</code>	여러 라인 동시 편집 시작/종료	<code>Alt + Shift + A</code>
뒤로 되돌리기	<code>Ctrl + Z</code>	앞으로 되돌리기	<code>Ctrl + Y</code>
대문자로 바꾸기	<code>Ctrl + Shift + X</code>	소문자로 바꾸기	<code>Ctrl + Shift + Y</code>
들여쓰기	<code>Tab</code>	내어쓰기	<code>Shift + Tab</code>
줄맞춤 자동 정렬	<code>Ctrl + Shift + F</code>	Package Import	<code>Ctrl + Shift + O</code>
Quick Fix	<code>Ctrl + 1</code>	컨텐츠 어시스트	<code>Ctrl + Space</code>
이름 일괄 변경	<code>Alt + Shift + R</code>	변수/메소드 찾기	<code>Ctrl + O</code>
라인 삭제	<code>Ctrl + D</code>		
싱글라인 주석 및 해제	<code>Ctrl + /</code>	멀티라인 주석	<code>Ctrl + Shift + /</code>

0. Java Programming Dev Env

- 자주 사용하는 이클립스 단축키

실행 및 디버그 관련

설명	단축키	설명	단축키
Java 코드 실행	<code>Ctrl + F11</code>	Java 코드 Debug 실행	<code>F11</code>
(Debug) 다음 스텝	<code>F6</code>	(Debug) 들어가기	<code>F5</code>
(Debug) 나가기	<code>F7</code>	(Debug) 다음 Breakpoint 까지 진행하기	<code>F8</code>

Java Programming

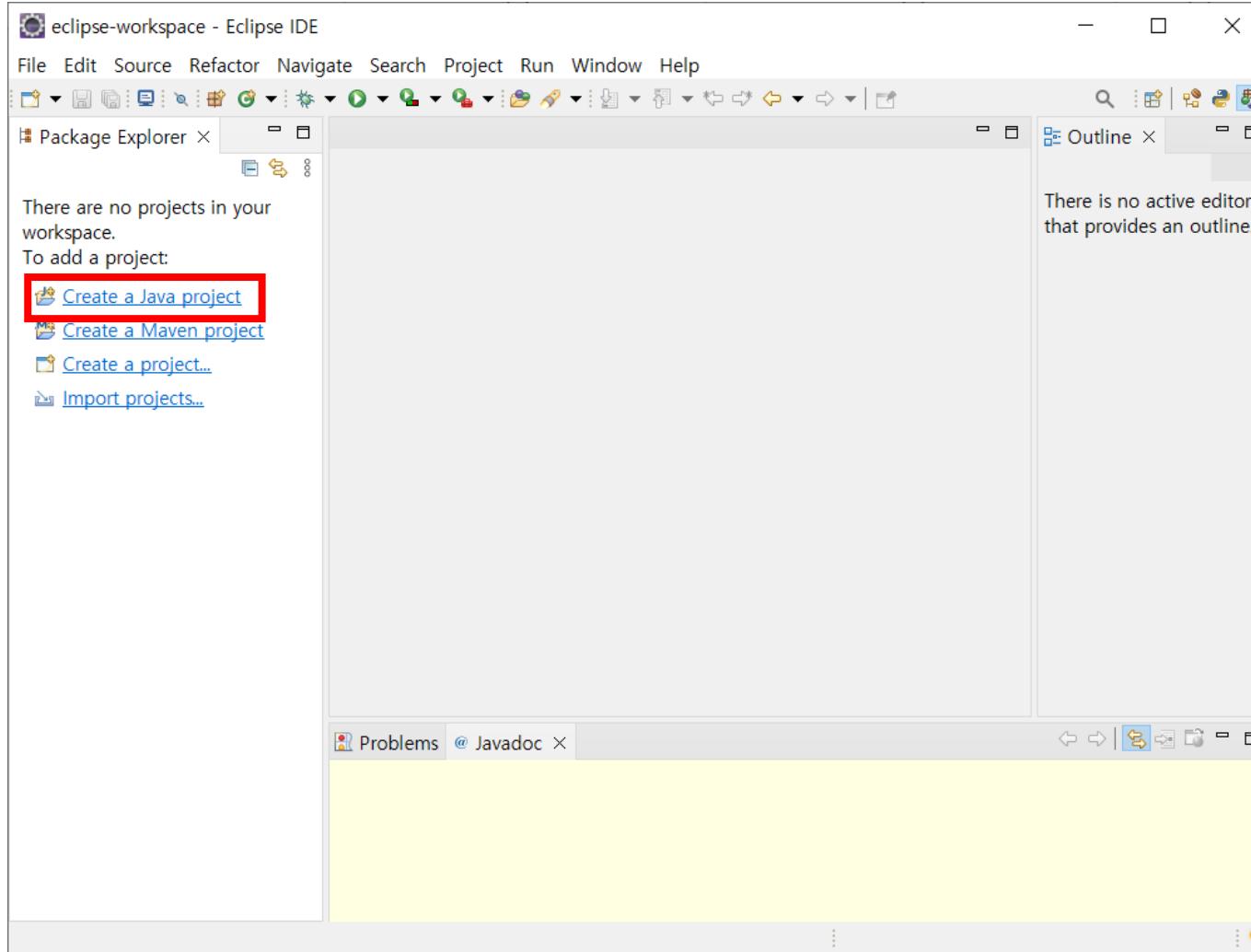
기초

1. Hello Java Programming
2. 자료형
3. 변수와 변수의 범위
4. 상수
5. 형 변환
6. 연산자
7. 실행흐름 제어
8. 메소드
9. 클래스 / 인스턴스/생성자

1. Hello Java Programming

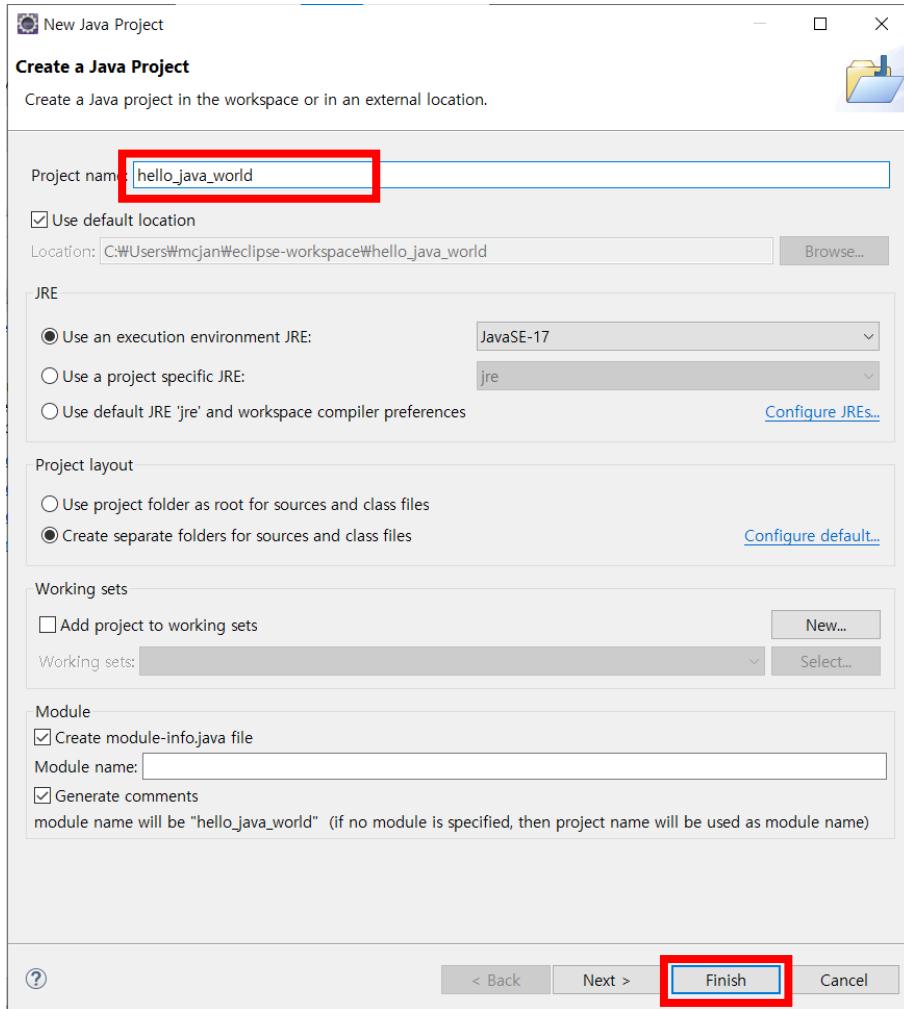
1. Hello Java Programming

- Create a Java project 클릭



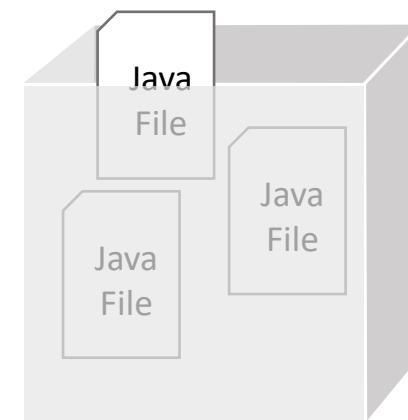
1. Hello Java Programming

- Project name: hello_java_world



Project?

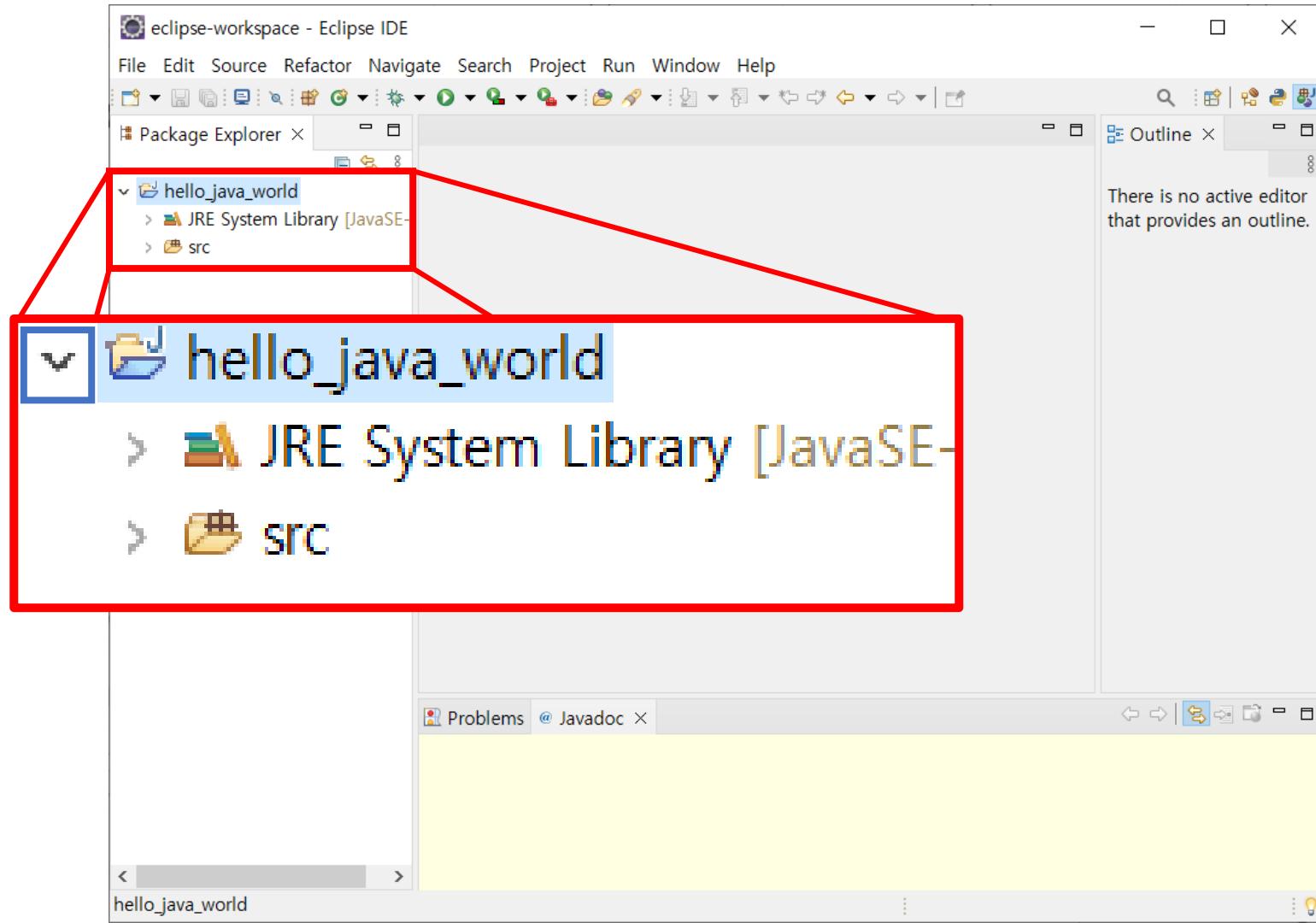
여러 개의 Java 파일로
하나의 Program으로 만드는 일종의 폴더



Java project

1. Hello Java Programming

- Project name: hello_java_world

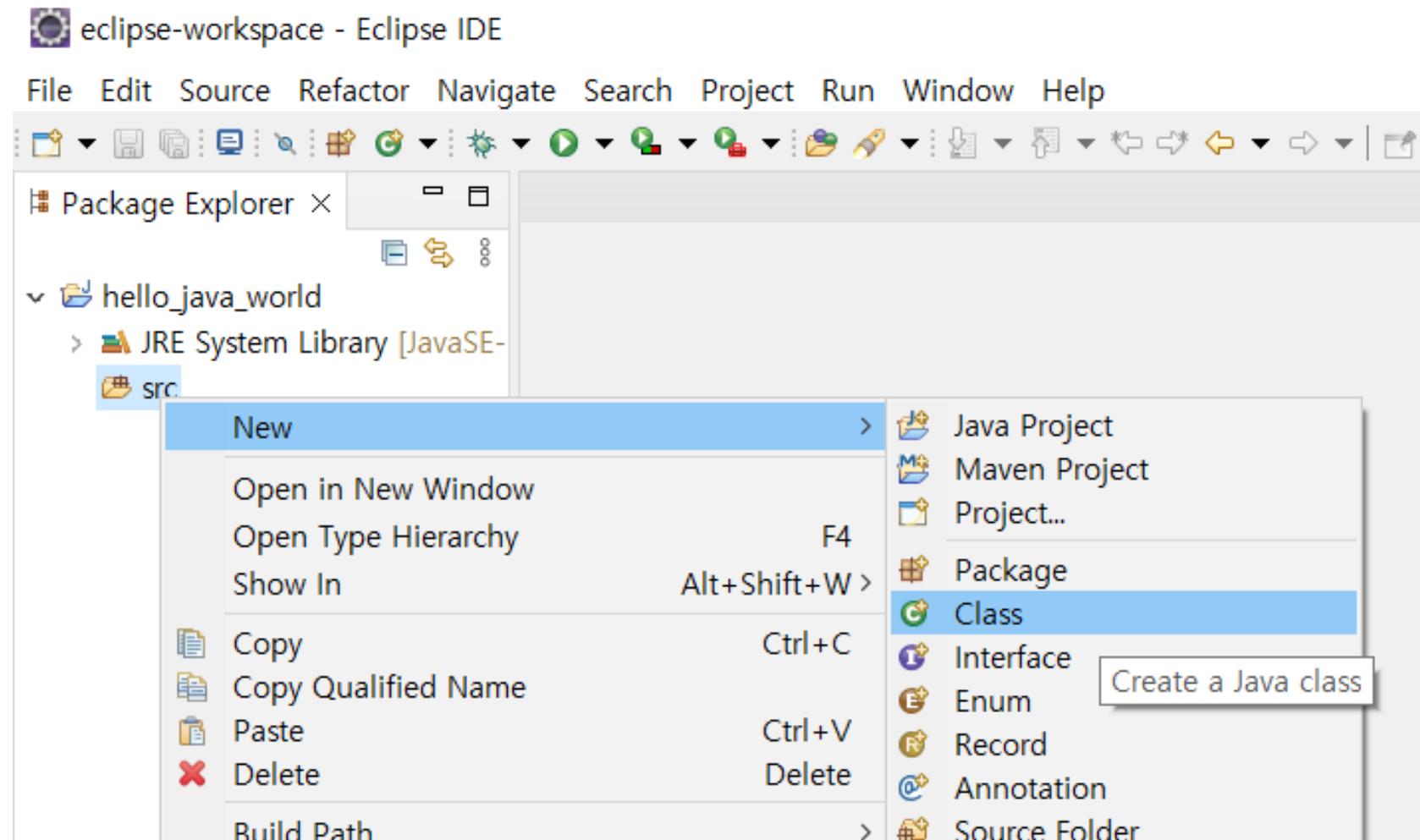


1. Hello Java Programming

- src/module-info.java 파일 삭제
 - ✓  hello_java_world
 -  JRE System Library [JavaSE]
 - ✓  src
 -  module-info.java

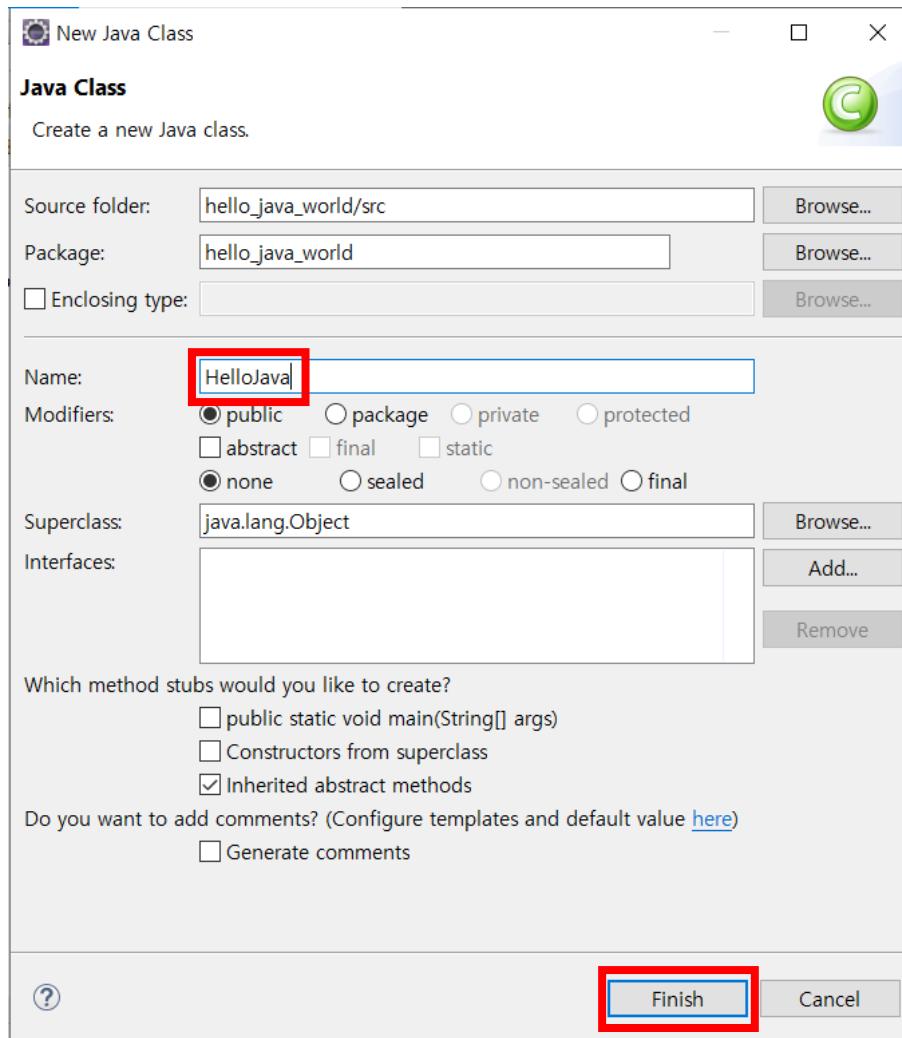
1. Hello Java Programming

- src 오른쪽 클릭 > New > Class 클릭



1. Hello Java Programming

- Name: HelloJava



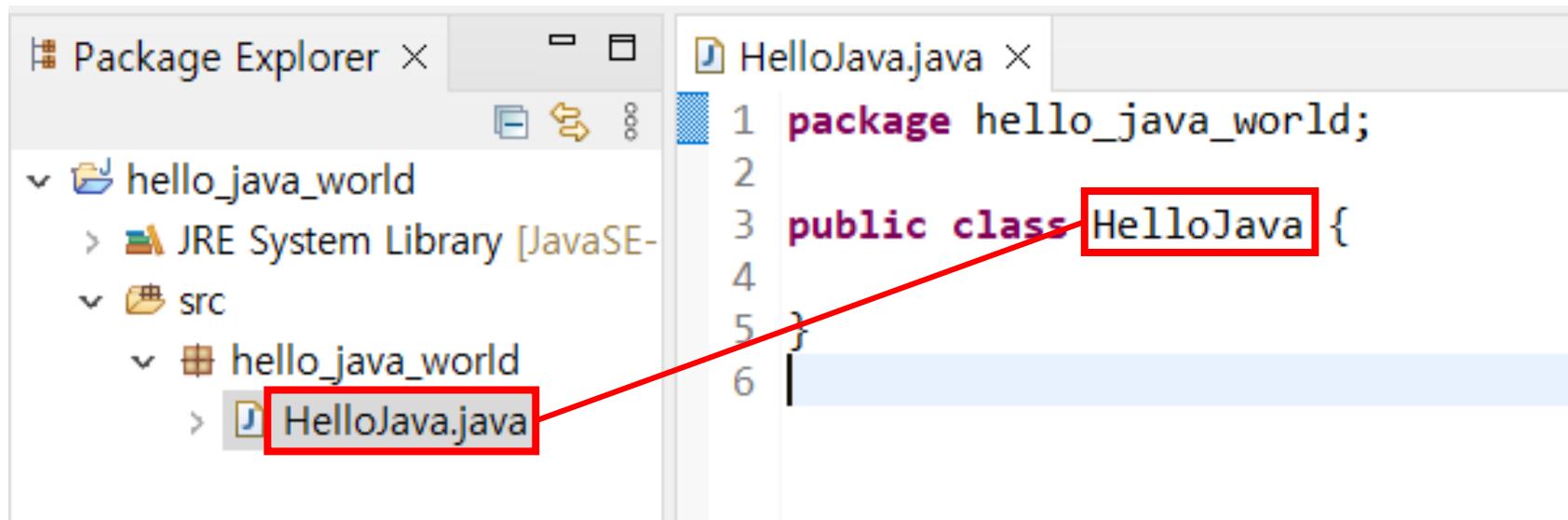
Class?

Java Program에서 하나 이상의 정보를 표현하는 묶음.

보통 Java 파일은 Class를 의미합니다.

1. Hello Java Programming

- HelloJava.java
 - 클래스의 이름과 파일이름은 동일해야 합니다.
 - 만약, 클래스의 이름과 파일명이 다르면 에러가 발생합니다.



1. Hello Java Programming

- 노란영역 작성

```
package hello_java_world;

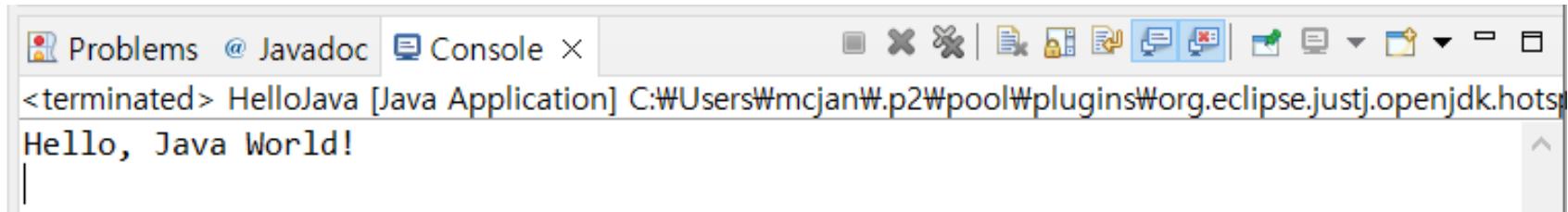
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("Hello, Java World!");
    }
}
```

세미콜론은 명령을 구분짓습니다.
하나의 명령이 종료되면 반드시 세미콜론을 작성해야 합니다.

중괄호 {} 사이는 Tab을 한 번 눌러 반드시 들여쓰기를 해야합니다.
들여쓰지 않으면 가독성이 나빠져 수정하기가 매우 어렵게 됩니다.

1. Hello Java Programming

- HelloJava 완성
- Ctrl + F11을 눌러 실행.



public static void main(String[] args) { ... }

HelloJava 파일을 실행시키는 코드입니다.

이 코드가 없거나 오탈자가 있다면 HelloJava는 실행되지 않습니다.

System.out.println("...");

큰 따옴표 안의 내용을 “콘솔”에 출력시키는 코드입니다.

숫자, 문자, 불린

2. 자료형

2. 자료형

- 자료형이란.
 - 메모리(Ram)에 데이터를 할당하기 위한 타입.
- Java의 자료형은 크게 두 가지로 분류됩니다.

Primitive type (기본 자료형)	Reference type (참조 자료형)
숫자	문자열
문자 (한 글자)	배열
true / false	등 Primitive Type을 제외한 모든 것.
값을 저장합니다.	메모리를 참조합니다.

2. 자료형

- Primitive Type의 구분

Primitive type (기본 자료형)	자료형	Literal	크기		설명
숫자 (정수)	byte	없음	1 byte	8 bit	2^8 -128 ~ 127 사이의 숫자
	short	없음	2 byte	16 bit	2^{16} -32,768 ~ 32,767 사이의 숫자
	int	없음	4 byte	32 bit	2^{32} -2,147,483,648 ~ 2,147,483,647 사이의 숫자
	long	L	8 byte	64 bit	2^{64} -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 사이의 숫자
숫자 (부동소수점)	float	f, F	4 byte	32 bit	2^{32} 1.4E-45 ~ 3.4028235E38 사이의 숫자
	double	d, D	8 byte	64 bit	2^{64} 4.9E-324 ~ 1.7976931348623157E308 사이의 숫자
불린	boolean	없음	1 byte	8 bit	2^8 true 혹은 false
문자 (한 글자)	char	없음	1 byte	8 bit	2^8 한 글자. 예) A, B, C

숫자 타입의 지정 방법

byte, short 타입은 잘 사용되지 않습니다.

일반적으로 10억 미만의 수는 int를, 10억 이상의 수는 long을 사용합니다.

부동소수점 float, double의 차이는 소수점이하의 길이의 차이만 존재합니다.

메모리 주소의 별칭 (Alias)

3. 변수와 변수의 범위

3. 변수

- 변수란.
 - 데이터가 할당된 메모리 주소의 별칭.
 - 변수의 이름으로 메모리 주소에 접근이 가능합니다.
 - 변수는 정의, 값 할당 부분으로 구성됩니다.

변수의 정의

아래와 같은 포맷으로 정의합니다. (같은 이름의 변수는 정의할 수 없습니다.)

자료형 변수명:

예> **int number;**

변수 값 할당

이미 정의된 변수에 값을 할당할 때에 아래와 같이 합니다.

변수명 = 값;

예> **number = 10;**

변수의 정의와 할당을 동시에 할 수도 있습니다.

자료형 변수명 = 값;

예> **int number = 5;**

3. 변수 – 숫자(정수)형

- 정수형 변수의 선언과 할당.

... 생략 ...

```
public static void main(String[] args) {  
    byte byteNumber = 1;  
    System.out.println(byteNumber);
```

```
    short shortNumber = 10;  
    System.out.println(shortNumber);
```

```
    int intNumber = 20;  
    System.out.println(intNumber);
```

```
    long longNumber = 30L;  
    System.out.println(longNumber);  
}
```

... 생략 ...

3. 변수 – 숫자(정수)형

- 메모리에서는 무슨 일이 벌어졌을까?
 - 임의의 메모리에 바이트 확보 후 값 할당.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	20	17	18 int 4byte 할당	19
21	22	23	24	25	26	27	28	29	30
31	32	33	1	34	35	36	37	38	39
41	42	43	byte 1byte 할당	44	45	10	46	47	48
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	30	83	84	85	86 long 8byte 할당	87	88	89
									90

3. 변수 – 숫자(정수)형

- 메모리에서는 무슨 일이 벌어졌을까?
 - 각 메모리 주소에 변수명 매칭. (별칭 부여)



3. 변수 – 숫자(정수)형

- 메모리에서는 무슨 일이 벌어졌을까?
 - 실제로는 2진수가 저장된다.

1	2	3	4	5	6	7	8	9	10
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

메모리 주소 33의 별칭: byteNumber
byte byteNumber = 1;

byte 1byte 할당

메모리 주소 16의 별칭: intNumber
int intNumber = 20;

int 4byte 할당

메모리 주소 45의 별칭: shortNumber
short shortNumber = 10;

short 2byte 할당

메모리 주소 82의 별칭: longNumber
long longNumber = 30L;

long 8byte 할당

3. 변수 – 숫자(정수)형

- 정수형 변수의 재할당 (값 수정).

... 생략 ...

```
public static void main(String[] args) {  
    byte byteNumber = 1;  
    byteNumber = 2;  
    System.out.println(byteNumber);  
    short shortNumber = 10;  
    shortNumber = 11;  
    System.out.println(shortNumber);  
    int intNumber = 20;  
    intNumber = 21;  
    System.out.println(intNumber);  
    long longNumber = 30L;  
    longNumber = 31L;  
    System.out.println(longNumber);  
}
```

... 생략 ...

3. 변수 – 숫자(정수)형

- 메모리에서는 무슨 일이 벌어졌을까?
 - 변수가 가리키고 있는 메모리 주소의 데이터가 변경된다.



3. 변수 – 숫자(정수)형

- 메모리에서는 무슨 일이 벌어졌을까?
 - 실제로는 2진수가 저장된다.

1	2	3	4	5	6	7	8	9	10
21	22	23	24	25	26	27	28	29	30
31	32	33 00000010	34	35	36	37	38	39	40
41	42	43 byte 1byte 할당	44	45 00000000	46 00001011	47	48	49	50
51	52	53	54	55	56 short 2byte 할당	57	58	59	60
61	62	63			64 메모리 주소 82의 별칭: longNumber	65	66	67	68
71	72	73			74 longNumber = 31L;	75	76	77	78
81	82 00000000	83 00000000	84 00000000	85 00000000	86 00000000	87 0000000000	88 0000000000	89 00011111	90

메모리 주소 33의 별칭: byteNumber
byteNumber = 2;

메모리 주소 16의 별칭: intNumber
intNumber = 21;

메모리 주소 45의 별칭: shortNumber
shortNumber = 11;

메모리 주소 82의 별칭: longNumber
longNumber = 31L;

int 4byte 할당

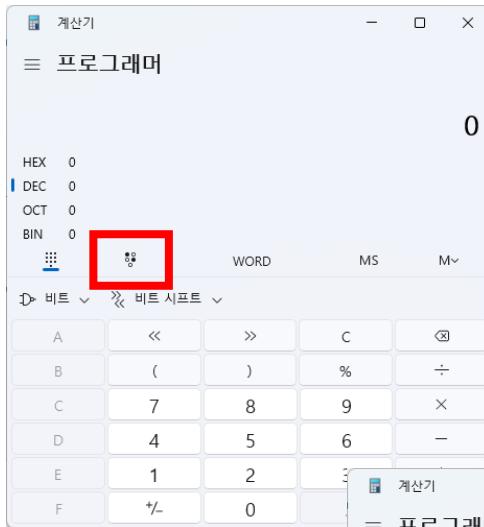
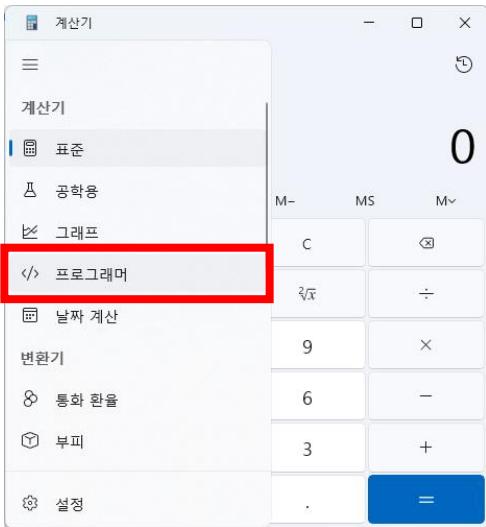
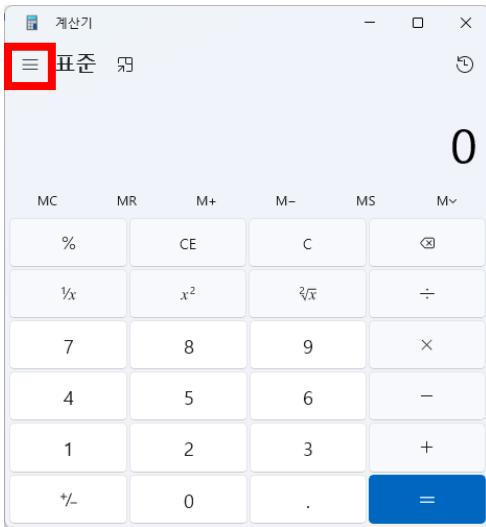
byte 1byte 할당

short 2byte 할당

long 8byte 할당

3. 변수 – 숫자(정수)형

- 2진수 계산기
 - 계산기 프로그램을 실행.



A screenshot of a hex dump application window titled '계산기' (Calculator). The title bar says '프로그래머'. The display shows '0'. The hex dump table has columns for HEX, DEC, OCT, and BIN. The first row shows values 00000000, 00000000, 00000000, and 00000000. The second row shows values 60, 56, 52, and 48. The third row shows values 00000000, 00000000, 00000000, and 00000000. The fourth row shows values 44, 40, 36, and 32. The fifth row shows values 00000000, 00000000, 00000000, and 00000000. The sixth row shows values 28, 24, 00, and 16. The seventh row shows values 00000000, 00000000, 00000000, and 00000000. The eighth row shows values 12, 8, 4, and 0. The value at address 12 is highlighted with a yellow box and labeled '1 byte'. The value at address 4 is highlighted with a red box.

HEX	DEC	OCT	BIN
00000000	0	0	00000000
60	56	52	48
00000000	0	0	00000000
44	40	36	32
00000000	0	0	00000000
28	24	00	16
00000000	0	0	00000000
12	8	4	0

3. 변수 – 숫자(부동소수점)형

- 부동소수점형 변수의 선언과 할당

... 생략 ...

```
public static void main(String[] args) {  
    float floatNumber = 10.55f;  
    System.out.println(floatNumber);  
  
    double doubleNumber = 11.556;  
    System.out.println(doubleNumber);  
}  
... 생략 ...
```

3. 변수 – 숫자(부동소수점)형

- 메모리에서는 무슨 일이 벌어졌을까?
 - 임의의 메모리에 바이트 확보 후 값 할당 후 변수명 매칭

메모리 주소 12의 별칭: floatNumber

1	2	3	4	5	6	7	8	9	10
11	12	10.55	13	14	15	16	17	18	19
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

float floatNumber = 10.55f;

메모리 주소 63의 별칭: doubleNumber

double doubleNumber = 11.556;

double 8byte 할당

3. 변수 – 문자형

- 문자형 변수의 선언과 할당
 - 문자형은 작은 따옴표로 표현.
 - 작은 따옴표 안의 숫자는 문자로 처리됩니다.

```
public static void main(String[] args) {  
    char letter = 'A';  
    System.out.println(letter);  
  
    letter = 'B';  
    System.out.println(letter);  
  
    letter = '1';  
    System.out.println(letter);  
}
```

char

한 글자를 표현하는 char 타입의 값은 작은따옴표로 감싸야 합니다.
작은 따옴표에는 한 글자만 적을 수 있습니다.

3. 변수 - 불린형

- boolean(불린, 불리언)
 - boolean형은 true 혹은 false만 존재하는 타입입니다.
- ```
public static void main(String[] args) {
 boolean areYouStudent = true;
 System.out.println(areYouStudent);

 boolean areYouDesigner = false;
 System.out.println(areYouDesigner);
}
```
- 연산자, 제어문, 반복문 등에서 많이 사용됩니다.

### 3. 변수의 범위

---

- 변수는 선언된 위치에 따라 사용할 수 있는 범위가 달라집니다.
- 변수는 중괄호 – { } 의 범위를 가집니다.

```
public class App {
 public static void main(String[] args) {
 int number = 10;
 System.out.println(number); — number 변수의 사용 가능 범위
 }
}
```

절대 변하지 않을 빛의 속도 = 299,792,458m/s

## 4. 상수

# 4. 상수

- 변수가 정의되고 최초 한번 값이 할당되고나면
- 다시는 재할당(수정/업데이트)할 수 없는 변수를 상수라고 합니다.

## 상수의 정의와 할당

변수의 정의와 할당과 동일합니다.

어떤 변수의 타입 앞에 **final**만 붙이면 상수가 됩니다.

```
public static void main(String[] args) {
 final int speedOfLight = 299_792_458;
 System.out.println(speedOfLight);

 speedOfLight = 10; ← 에러!
}
```

# 4. 상수

---

- 상수를 정의하고 값을 나중에 할당할 수도 있습니다.
- 할당이 되고나면 이 값은 변경할 수 없습니다.

```
public static void main(String[] args) {
 final int speedOfLight;
 speedOfLight = 299792458;
 System.out.println(speedOfLight);
}
```

Java에는 여러 상수들이 이미 정의되어 있습니다.

원주율: Math.PI

자연상수: Math.E

int 변수의 최대 값: Integer.MAX\_VALUE

## 5. 형 변환

# 5. 형변환

- Primitive Type을 다시 볼까요.

| Primitive type<br>(기본 자료형) | 자료형 명   | 자료형  | Literal | 크기     |          | 설명                                                            |
|----------------------------|---------|------|---------|--------|----------|---------------------------------------------------------------|
| 숫자 (정수)                    | byte    | 없음   | 1 byte  | 8 bit  | $2^8$    | -128 ~ 127 사이의 숫자                                             |
|                            | short   | 없음   | 2 byte  | 16 bit | $2^{16}$ | -32,768 ~ 32,767 사이의 숫자                                       |
|                            | int     | 없음   | 4 byte  | 32 bit | $2^{32}$ | -2,147,483,648 ~ 2,147,483,647 사이의 숫자                         |
|                            | long    | l, L | 8 byte  | 64 bit | $2^{64}$ | -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 사이의 숫자 |
| 숫자 (부동소수점)                 | float   | f, F | 4 byte  | 32 bit | $2^{32}$ | 1.4E-45 ~ 3.4028235E38 사이의 숫자                                 |
|                            | double  | d, D | 8 byte  | 64 bit | $2^{64}$ | 4.9E-324 ~ 1.7976931348623157E308 사이의 숫자                      |
| 布尔                         | boolean | 없음   | 1 byte  | 8 bit  | $2^8$    | true 혹은 false                                                 |
| 문자 (한 글자)                  | char    | 없음   | 1 byte  | 8 bit  | $2^8$    | 한 글자. 예) A, B, C                                              |

- 자바에서 지원하는 기본 자료형은 각각 고유한 바이트 크기를 가지고 있습니다.

# 5. 형변환

---

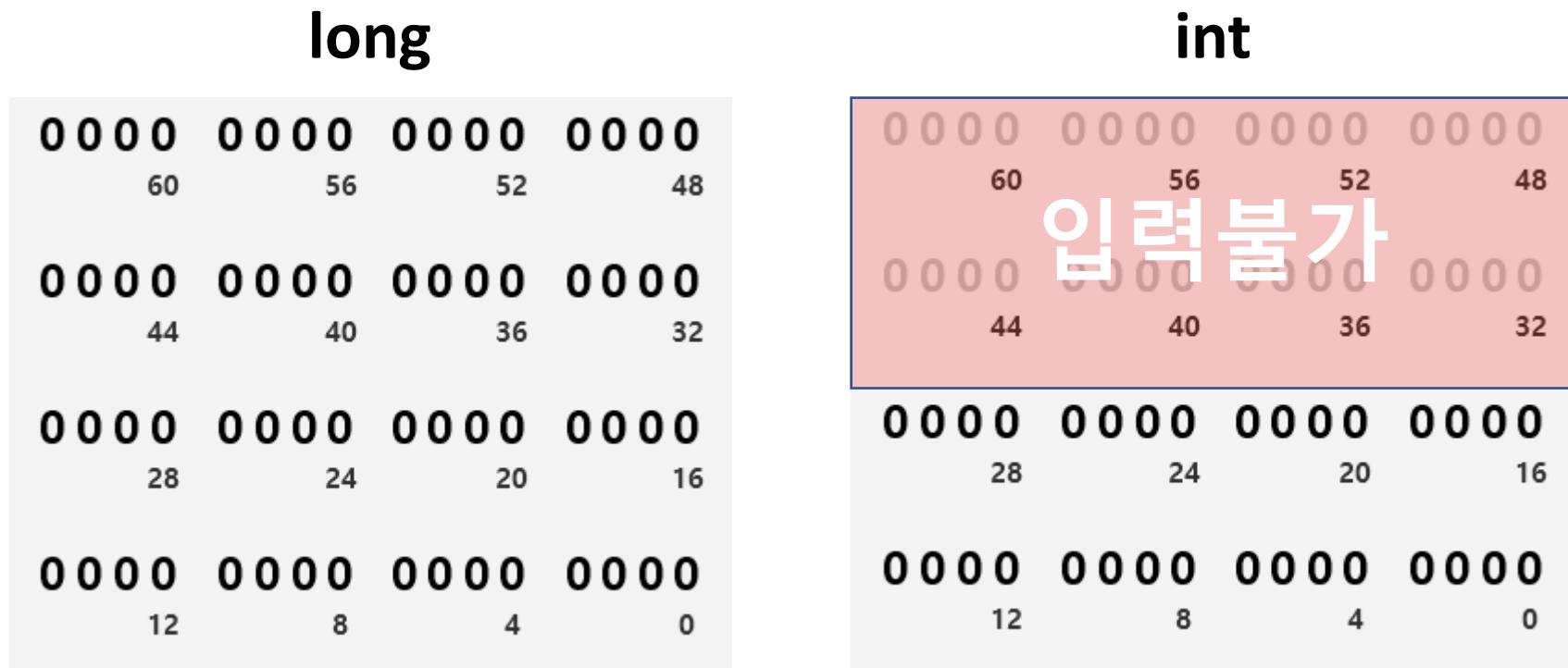
- 4 byte 크기의 int 변수 값을 8 byte 크기의 long 변수에 할당하면 어떤 일이 일어날까요.

```
public static void main(String[] args) {
 int normalNumber = Integer.MAX_VALUE;
 long bigNumber = normalNumber;
 System.out.println(normalNumber);
 System.out.println(bigNumber);
}
```

- 에러 없이 잘 동작합니다.
- 잘 동작하는 것은 메모리 크기 때문인데요.

# 5. 형변환

- long 형과 int 형의 2진수 범위를 보면  
long 형이 int형보다 두 배 많은 정보를 담을 수 있게되어있습니다.



# 5. 형변환

- long의 범위가 훨씬 크기 때문에 int 값을 할당하는데 문제가 없습니다.  
이런 특징을 “**목시적 형변환**”이라고 합니다.

## int의 최대 범위

| 2,147,483,647                   |                |     |                                         |
|---------------------------------|----------------|-----|-----------------------------------------|
| HEX                             | 7FFF FFFF      | DEC | 2,147,483,647                           |
| OCT                             | 17 777 777 777 | BIN | 0111 1111 1111 1111 1111 1111 1111 1111 |
| DWORD                           | 60 56 52 48    | MS  | M▼                                      |
| 0000 0000 0000 0000             |                |     |                                         |
| 60                              | 56             | 52  | 48                                      |
| 0000 0000 0000 0000             |                |     |                                         |
| 44                              | 40             | 36  | 32                                      |
| 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                |     |                                         |
| 28                              | 24             | 20  | 16                                      |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                |     |                                         |
| 12                              | 8              | 4   | 0                                       |

## long의 최대 범위

| 9,223,372,036,854,775,807       |                                 |     |                                                                                 |
|---------------------------------|---------------------------------|-----|---------------------------------------------------------------------------------|
| HEX                             | 7FFF FFFF FFFF FFFF             | DEC | 9,223,372,036,854,775,807                                                       |
| OCT                             | 777 777 777 777 777 777 777 777 | BIN | 0111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 |
| QWORD                           | 60 56 52 48                     | MS  | M▼                                                                              |
| 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                 |     |                                                                                 |
| 28                              | 24                              | 20  | 16                                                                              |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                 |     |                                                                                 |
| 44                              | 40                              | 36  | 32                                                                              |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                 |     |                                                                                 |
| 28                              | 24                              | 20  | 16                                                                              |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                 |     |                                                                                 |
| 12                              | 8                               | 4   | 0                                                                               |

# 5. 형변환

- byte → short → int → long 은 모두 “**묵시적 형변환**” 을 합니다.

묵시적 형변환 가능

| 계산기                 |           | -  | □  | × |
|---------------------|-----------|----|----|---|
| 프로그램                |           |    |    |   |
| 127                 |           |    |    |   |
| HEX                 | 7F        |    |    |   |
| DEC                 | 127       |    |    |   |
| OCT                 | 177       |    |    |   |
| BIN                 | 0111 1111 |    |    |   |
| BYTE                |           | MS | M▼ |   |
| 0000 0000 0000 0000 |           |    |    |   |
| 60   56   52   48   |           |    |    |   |
| 0000 0000 0000 0000 |           |    |    |   |
| 44   40   36   32   |           |    |    |   |
| 0000 0000 0000 0000 |           |    |    |   |
| 28   24   20   16   |           |    |    |   |
| 0000 0000 0111 1111 |           |    |    |   |
| 12   8   4   0      |           |    |    |   |

| 계산기                 |                     | -  | □  | × |
|---------------------|---------------------|----|----|---|
| 프로그램                |                     |    |    |   |
| 32,767              |                     |    |    |   |
| HEX                 | 7FFF                |    |    |   |
| DEC                 | 32,767              |    |    |   |
| OCT                 | 77 777              |    |    |   |
| BIN                 | 0111 1111 1111 1111 |    |    |   |
| WORD                |                     | MS | M▼ |   |
| 0000 0000 0000 0000 |                     |    |    |   |
| 60   56   52   48   |                     |    |    |   |
| 0000 0000 0000 0000 |                     |    |    |   |
| 44   40   36   32   |                     |    |    |   |
| 0000 0000 0000 0000 |                     |    |    |   |
| 28   24   20   16   |                     |    |    |   |
| 0111 1111 1111 1111 |                     |    |    |   |
| 12   8   4   0      |                     |    |    |   |

| 계산기                 |                                         | -  | □  | × |
|---------------------|-----------------------------------------|----|----|---|
| 프로그램                |                                         |    |    |   |
| 2,147,483,647       |                                         |    |    |   |
| HEX                 | 7FFF FFFF                               |    |    |   |
| DEC                 | 2,147,483,647                           |    |    |   |
| OCT                 | 17 777 777 777                          |    |    |   |
| BIN                 | 0111 1111 1111 1111 1111 1111 1111 1111 |    |    |   |
| DWORD               |                                         | MS | M▼ |   |
| 0000 0000 0000 0000 |                                         |    |    |   |
| 60   56   52   48   |                                         |    |    |   |
| 0000 0000 0000 0000 |                                         |    |    |   |
| 44   40   36   32   |                                         |    |    |   |
| 0000 0000 0000 0000 |                                         |    |    |   |
| 28   24   20   16   |                                         |    |    |   |
| 0111 1111 1111 1111 |                                         |    |    |   |
| 28   24   20   16   |                                         |    |    |   |

| 계산기                             |                                                                                 | -  | □  | × |
|---------------------------------|---------------------------------------------------------------------------------|----|----|---|
| 프로그램                            |                                                                                 |    |    |   |
| 9,223,372,036,854,775,807       |                                                                                 |    |    |   |
| HEX                             | 7FFF FFFF FFFF FFFF                                                             |    |    |   |
| DEC                             | 9,223,372,036,854,775,807                                                       |    |    |   |
| OCT                             | 777 777 777 777 777 777                                                         |    |    |   |
| BIN                             | 0111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 |    |    |   |
| QWORD                           |                                                                                 | MS | M▼ |   |
| 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                                                                 |    |    |   |
| 60   56   52   48               |                                                                                 |    |    |   |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                                                                 |    |    |   |
| 44   40   36   32               |                                                                                 |    |    |   |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                                                                 |    |    |   |
| 28   24   20   16               |                                                                                 |    |    |   |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                                                                 |    |    |   |
| 12   8   4   0                  |                                                                                 |    |    |   |

묵시적 형변환 불가능

# 5. 형변환

- 반면, 크기가 큰 long 형의 값을 비교적 작은 int형으로 자연스러운 변환은 불가능합니다.

```
public static void main(String[] args) {
 long bigNumber = Long.MAX_VALUE;
 int normalNumber = bigNumber; ← 에러!
 System.out.println(bigNumber);
 System.out.println(normalNumber);
}
```

- 위 코드에서 에러를 제거하려면 “명시적 형변환”을 해야만 합니다.
- “명시적 형변환”이란 강제로 형을 변경하는 것을 말합니다.

```
public static void main(String[] args) {
 long bigNumber = Integer.MAX_VALUE + 1L;
 int normalNumber = (int) bigNumber;
 System.out.println(bigNumber);
 System.out.println(normalNumber);
}
```

# 5. 형변환

---

- 범위를 벗어난 수를 넣었을 때,
  - 할당 값이 이상해지는 경우가 종종 발생할 수 있습니다.
  - 할당할 수 있는 범위를 초과하는 버그가 발생하기 때문입니다.
- 아래의 코드를 생각해봅니다.

```
public static void main(String[] args) {
 System.out.println(Integer.MAX_VALUE);
 int normalNumber = Integer.MAX_VALUE + 1;
 System.out.println(normalNumber);
}
```

# 5. 형변환

## Integer.MAX\_VALUE

| 계산기                             |                                         | -  | □  | × |
|---------------------------------|-----------------------------------------|----|----|---|
| 프로그래머                           |                                         |    |    |   |
| 2,147,483,647                   |                                         |    |    |   |
| HEX                             | 7FFF FFFF                               |    |    |   |
| DEC                             | 2,147,483,647                           |    |    |   |
| OCT                             | 17 777 777 777                          |    |    |   |
| BIN                             | 0111 1111 1111 1111 1111 1111 1111 1111 |    |    |   |
|                                 | DWORD                                   | MS | M▼ |   |
| 0000 0000 0000 0000             |                                         |    |    |   |
| 60 56 52 48                     |                                         |    |    |   |
| 0000 0000 0000 0000             |                                         |    |    |   |
| 44 40 36 32                     |                                         |    |    |   |
| 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                         |    |    |   |
| 28 24 20 16                     |                                         |    |    |   |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |                                         |    |    |   |
| 12 8 4 0                        |                                         |    |    |   |

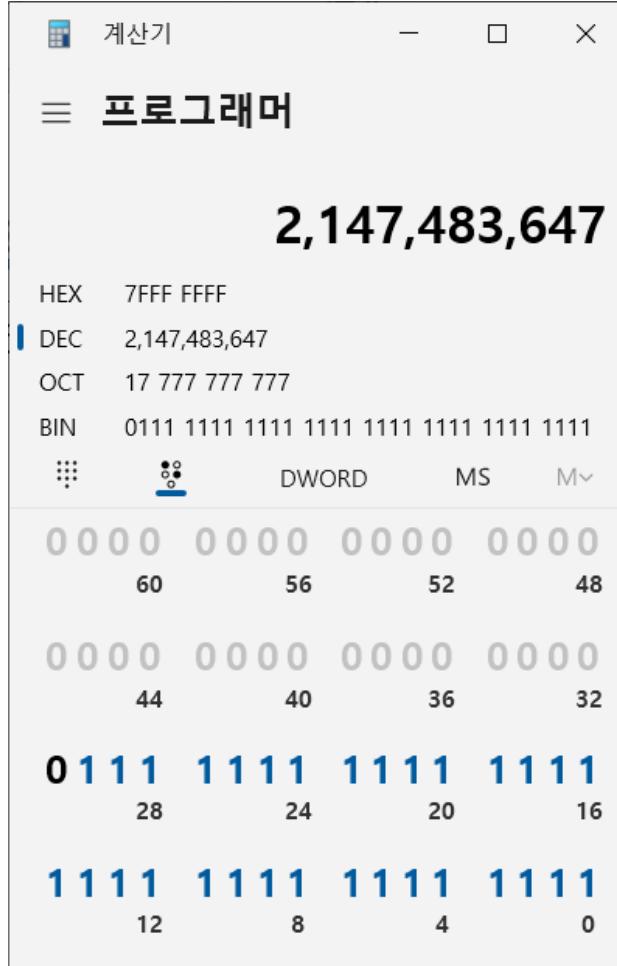
- 2진계산법
- $0 + 1 = 1$
- $1 + 1 = 10$
- $10 + 1 = 11$
- $11 + 1 = 100$

|   |   |   |   |   |
|---|---|---|---|---|
|   | 1 | 1 | 1 |   |
|   | 1 | 1 | 1 | 1 |
| + |   |   |   | 1 |
| 1 | 0 | 0 | 0 | 0 |

- int 최대값
- 01111111\_11111111\_11111111\_11111111 에 1을 더 하면
- 10000000\_00000000\_00000000\_00000000 이 되어
- 10진수 -2,147,483,648 가 됩니다.
- 이런 현상을 정수오버플로우라고 합니다.

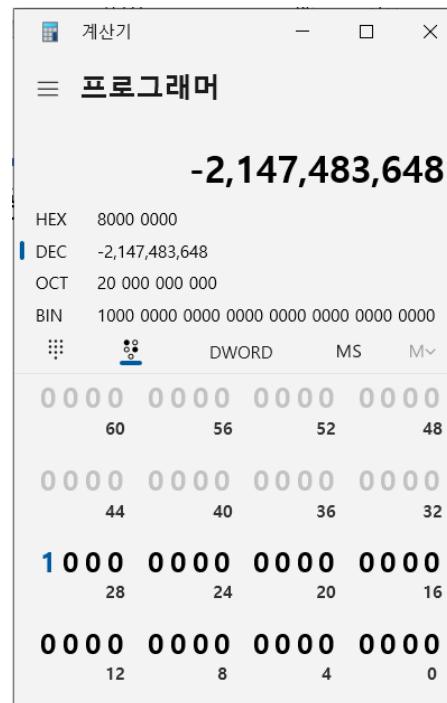
# 5. 형변환

## Integer.MAX\_VALUE

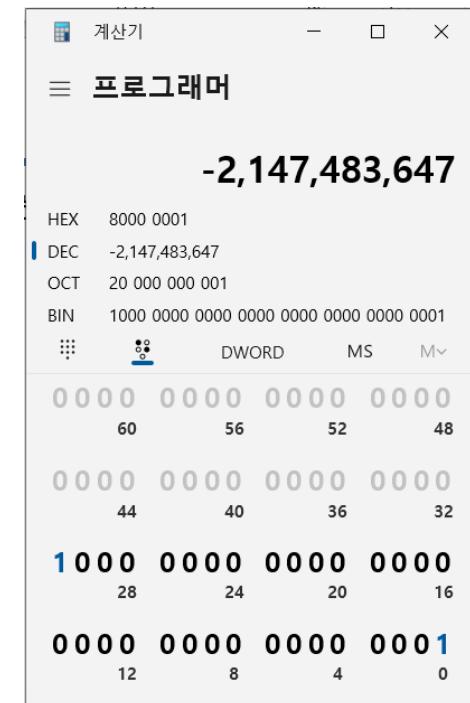


- 만약 int 최대값에 2를 더하면 어떤 일이 일어날까요?

+ 1



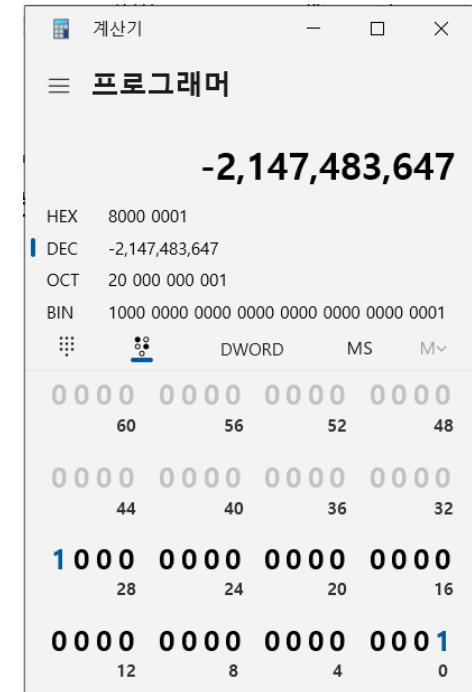
+ 1



# 5. 형변환

- 따라서 아래 코드에서 normalNumber의 값은
- 2,147,483,647이 됩니다.

```
public static void main(String[] args) {
 long longNumber = 2147483649L; ← int 최대 값보다 2큽니다.
 System.out.println(longNumber);
 int normalNumber = (int) longNumber;
 System.out.println(normalNumber);
}
```



## 명시적 형변환의 주의점

명시적 형변환을 할 때, 항상 정수 오버플로우가 발생하지 않도록 주의를 기울어야 합니다.

# 5. 형변환

- 정수 <-> 부동소수점의 변환
  - 정수를 부동소수점으로 변환 (명시적 형변환)

```
public static void main(String[] args) {
 int num = 10;
 float fnum = num;
 double dnum = num;

 System.out.println(num); // 10
 System.out.println(fnum); // 10.0
 System.out.println(dnum); // 10.0
}
```

## 정수 → 부동소수점 변환에서 명시적 변환이 가능한 이유?

부동소수점은 정수보다 표현하는 범위가 더 큽니다. (같은 byte를 사용하더라도)  
부동소수점 = 정수 + 소수점

# 5. 형변환

- 정수 <-> 부동소수점의 변환
  - 부동소수점을 정수로 변환 (묵시적 형변환)

```
public static void main(String[] args) {
 float fnum = 10.9f;
 int num = (int) fnum;
 System.out.println(fnum); // 10.5
 System.out.println(num); // 10

 double dnum = 11.15;
 num = (int) dnum;
 System.out.println(dnum); // 11.15
 System.out.println(num); // 11
}
```

## 부동소수점 → 정수 변환시 소수점 밑 데이터가 사라지는 이유

정수는 소수점 이하의 자릿수들을 표현하지 못합니다.  
즉, 소수점 이하 자릿수는 모두 버려집니다.

# 5. 형변환

---

- 부동소수점 소수점 처리

```
public static void main(String[] args) {
 double dnum = 10.1;
 // 소수점 올림.
 dnum = Math.ceil(dnum);
 System.out.println(dnum); // 11.0
 // 소수점 버림.
 dnum = 10.9;
 dnum = Math.floor(dnum);
 System.out.println(dnum); // 10.0
 // 소수점 반올림.
 dnum = 15.18;
 dnum = Math.round(dnum);
 System.out.println(dnum); // 15.0
 dnum = 10.56;
 dnum = Math.round(dnum);
 System.out.println(dnum); // 11.0
}
```

# 5. 형변환

- 부동소수점 소수점 처리

```
public static void main(String[] args) {
 // 29.37 을 29.4로 올림 처리하기
 double dnum = 29.37;
 double dnum2 = dnum * 10;
 System.out.println(dnum2); // 293.7

 dnum2 = Math.round(dnum2);
 System.out.println(dnum2); // 294.0

 double dnum3 = dnum2 / 10;
 System.out.println(dnum3); // 29.4
}
```

**Java는 소수점 자리 처리가 유연하지 못합니다.**

| Java에서 소수점 자리 처리를 하려면 원하는 자릿수만큼 곱한 후 처리해야 합니다.

# 5. 형변환

---

- 문자열을 byte, short, int, long, float, double, boolean으로 변환
- Java는 문자열을 Primitive Type으로 변경이 가능합니다.

## 문자열

처음 문자열을 소개합니다.

문자열은 여러 개의 글자들을 나열한 것을 말합니다.

문자열 변수의 타입은 **String**으로 사용하고 값은 **큰 따옴표**로 감싸야합니다.

**String** 변수명 = "Hello, World!";

형식으로 사용합니다.

# 5. 형변환

- 문자열을 정수로 변환하려면 타입.parseType(문자)를 사용합니다.

```
public static void main(String[] args) {
 String numberString = "10";
 byte byteNumber = Byte.parseByte(numberString);
 System.out.println(byteNumber);

 numberString = "10000";
 short shortNumber = Short.parseShort(numberString);
 System.out.println(shortNumber);

 numberString = "1000000";
 int intNumber = Integer.parseInt(numberString);
 System.out.println(intNumber);

 numberString = "1000000000";
 long longNumber = Long.parseLong(numberString);
 System.out.println(longNumber);
}
```

# 5. 형변환

- 정수형이 아닌 문자열을 변환하려고 하면 에러가 발생합니다.

```
public static void main(String[] args) {
 String numberString = "A";
 byte byteNumber = Byte.parseByte(numberString);
 System.out.println(byteNumber);
}
```

Exception in thread "main" java.lang.NumberFormatException: For input string: "A"

- 부동소수점을 정수형으로 변환하려고 해도 에러가 발생합니다.

```
public static void main(String[] args) {
 String numberString = "10.5";
 byte byteNumber = Byte.parseByte(numberString);
 System.out.println(byteNumber);
}
```

Exception in thread "main" java.lang.NumberFormatException: For input string: "10.5"

# 5. 형변환

---

- 문자열을 부동소수점으로 변환하려면 **타입.parseType(문자)** 를 사용합니다.

```
public static void main(String[] args) {
 String numberString = "10.5";
 float floatNumber = Float.parseFloat(numberString);
 System.out.println(floatNumber);

 numberString = "11.577777777";
 double doubleNumber = Double.parseDouble(numberString);
 System.out.println(doubleNumber);

 numberString = "12";
 double doubleNumber2 = Double.parseDouble(numberString);
 System.out.println(doubleNumber2);
}
```

# 5. 형변환

---

- 문자열을 불린으로 변환하려면 **Boolean.parseBoolean(문자)** 를 사용합니다.

```
public static void main(String[] args) {
 String str = "true";
 boolean bool = Boolean.parseBoolean(str);
 System.out.println(bool); // true

 str = "TRUE";
 bool = Boolean.parseBoolean(str);
 System.out.println(bool); // true

 str = "tRue";
 bool = Boolean.parseBoolean(str);
 System.out.println(bool); // true
}
```

# 5. 형변환

---

- 문자열을 불린으로 변환하려면 **Boolean.parseBoolean(문자)** 를 사용합니다.

```
public static void main(String[] args) {
 String str = "false";
 boolean bool = Boolean.parseBoolean(str);
 System.out.println(bool); // false

 str = "";
 bool = Boolean.parseBoolean(str);
 System.out.println(bool); // false

 str = "anystring";
 bool = Boolean.parseBoolean(str);
 System.out.println(bool); // false
}
```

# 6. 연산자

# 6. 연산자

- Java가 지원하는 연산자의 종류
  - 할당 연산자

| 할당연산자  |     |         |                  |                  |
|--------|-----|---------|------------------|------------------|
| 연산자 기호 | 설명  | 예제      | 예제 설명            | 결과               |
| =      | 더하기 | num = 1 | num 변수에 1을 할당한다. | num 변수에 1이 할당된다. |

- 산술 연산자

| 산술연산자  |          |          |                     |                        |
|--------|----------|----------|---------------------|------------------------|
| 연산자 기호 | 설명       | 예제       | 예제 설명               | 결과 ( $a = 20, b = 3$ ) |
| +      | 더하기      | $a + b$  | a에서 b를 더한다.         | 23                     |
| -      | 빼기       | $a - b$  | a에서 b를 뺀다           | 17                     |
| *      | 곱하기      | $a * b$  | a에서 b를 곱한다.         | 60                     |
| /      | 나누기의 몫   | $a / b$  | a에서 b를 나눈 몫을 구한다    | 6                      |
| %      | 나누기의 나머지 | $a \% b$ | a에서 b를 나눈 나머지를 구한다. | 2                      |

# 6. 연산자

---

- 산술연산자

```
public static void main(String[] args) {
 int number = 10;
 int addedNumber = number + 2;
 System.out.println(addedNumber);

 int subtractedNumber = number - 3;
 System.out.println(subtractedNumber);

 int multipliedNumber = number * 3;
 System.out.println(multipliedNumber);

 int devideNumber = number / 3;
 System.out.println(devideNumber);

 int devideRemainNumber = number % 3;
 System.out.println(devideRemainNumber);
}
```

# 6. 연산자

---

- 복합 산술연산자

```
public static void main(String[] args) {
 int number = 10;
 int computedNumber1 = number + 3 - 1 * 3 / 3 % 3;
 System.out.println(computedNumber1); // 12

 int computedNumber2 = (number + (3 - 1)) * 3 / 5 % 3;
 System.out.println(computedNumber2); // 1
}
```

# 6. 연산자

- 연산 우선 순위 (PEMDAS)

| PEMDAS |                |                 |
|--------|----------------|-----------------|
| 우선순위   | 이름             | 설명              |
| 1      | Parenthesis    | 괄호 ( )          |
| 2      | Exponents      | 제곱 (Java 지원X)   |
| 3      | Multiplication | 곱하기 *           |
| 3      | Division       | 나누기 /, %        |
| 4      | Addition       | 더하기 +           |
| 4      | Subtraction    | 빼기 -            |
| 5      | Left to right  | 그 외 원쪽에서 오른 쪽으로 |

# 6. 연산자

---

- 산술연산자 – 스스로에게 연산하기

```
public static void main(String[] args) {
 int number1 = 10;
 number1 = number1 + 2;
 System.out.println(number1); // 12

 int number2 = 10;
 number2 = number2 - 2;
 System.out.println(number2); // 8

 int number3 = 10;
 number3 = number3 * 2;
 System.out.println(number3); // 20

 int number4 = 10;
 number4 = number4 / 4;
 System.out.println(number4); // 2

 int number5 = 10;
 number5 = number5 % 4;
 System.out.println(number5); // 2
}
```

# 6. 연산자

- Java가 지원하는 연산자의 종류
  - 단항 연산자

| 단항연산자           |             |                     |                                |            |
|-----------------|-------------|---------------------|--------------------------------|------------|
| 연산자 기호          | 설명          | 예제                  | 예제 설명                          | 결과 (a = 9) |
| <code>+=</code> | 더해서 할당      | <code>a += 2</code> | <code>a</code> 에 2를 더해서 할당     | 11         |
| <code>-=</code> | 빼서 할당       | <code>a -= 2</code> | <code>a</code> 에 2를 빼서 할당      | 7          |
| <code>*=</code> | 곱해서 할당      | <code>a *= 2</code> | <code>a</code> 에 2를 곱해서 할당     | 18         |
| <code>/=</code> | 나누어 뜯을 할당   | <code>a /= 2</code> | <code>a</code> 를 2로 나눈 뜯을 할당   | 4          |
| <code>%=</code> | 나누어 나머지를 할당 | <code>a %= 2</code> | <code>a</code> 를 2로 나눈 나머지를 할당 | 1          |
| <code>++</code> | 1을 더함       | <code>a++</code>    | <code>a</code> 에 1을 더해서 할당     | 10 (후 연산)  |
| <code>--</code> | 1을 뺄        | <code>a--</code>    | <code>a</code> 에 1을 빼서 할당      | 8 (후 연산)   |
| <code>++</code> | 1을 더함       | <code>++a</code>    | <code>a</code> 에 1을 더해서 할당     | 10 (선 연산)  |
| <code>--</code> | 1을 뺄        | <code>--a</code>    | <code>a</code> 에 1을 빼서 할당      | 8 (선 연산)   |

# 6. 연산자

---

- 산술연산자 – 스스로에게 연산하기

```
public static void main(String[] args) {
 int number1 = 10;
 number1 = number1 + 2;
 System.out.println(number1);
}
```

- 위의 코드는 아래 코드와 완전하게 일치하는 코드입니다.

```
public static void main(String[] args) {
 int number1 = 10;
 number1 += 2;
 System.out.println(number1);
}
```

# 6. 연산자

---

- 산술연산자 – 스스로에게 연산하기를 단항연산자로 하기

```
public static void main(String[] args) {
 int num = 10;
 num += 2; // num = num + 2;
 num -= 2; // num = num - 2;
 num *= 2; // num = num * 2;
 num /= 2; // num = num / 2;
 num %= 2; // num = num % 2;
}
```

# 6. 연산자

---

- 산술연산자 – 스스로에게 1을 더하거나 뺄 때

```
public static void main(String[] args) {
 int num = 10;
 num += 1;
 num -= 1;
}
```

- 아래처럼 줄일 수 있습니다.

```
public static void main(String[] args) {
 int num = 10;
 num++;
 num--;
}
```

- 단 ++, -- 연산자는 위치에 따라 선/후 수행 처리를 합니다.

# 6. 연산자

---

- `++, --` 는 위치에 따라 증감을 먼저 하거나 나중에 합니다.

```
public static void main(String[] args) {
 int num1 = 10;
 int num2 = 10;
 // 10을 먼저 출력한 다음 1 더하기
 System.out.println(num1++); // 10
 // 1을 먼저 더한 다음 출력하기
 System.out.println(++num2); // 11
}
```

```
public static void main(String[] args) {
 int num1 = 10;
 int num2 = 10;
 // 10을 먼저 출력한 다음 1 빼기
 System.out.println(num1--); // 10
 // 1을 먼저 뺀 다음 출력하기
 System.out.println(--num2); // 9
}
```

# 6. 연산자

---

- 산술 연산자 연습 문제

```
public class ArithmaticProblem1 {

 public static void main(String[] args) {
 int minutes = 5;
 int seconds = 50;
 int time = 0;

 /*
 * 산술연산자를 이용해
 * minutes와 seconds의 값을 초로 변환해
 * time 변수에 할당하고 출력해보세요.
 */
 }
}
```

# 6. 연산자

---

- 산술 연산자 연습 문제

```
public class ArithmaticProblem2 {

 public static void main(String[] args) {
 int processTime = 145;
 int minutes = 0;
 int seconds = 0;
 /*
 * 산술 연산자를 이용해
 * processTime을 분(Minute), 초(Second)
 * 를 구한다음 minutes, seconds 변수에 할당하고
 * 출력해보세요.
 */
 }
}
```

# 6. 연산자

---

- 산술 연산자 연습 문제

```
public class ArithmaticProblem3 {

 public static void main(String[] args) {
 int celsius = 30;
 int fahrenheit = 0;
 /*
 * 섭씨온도를 나타내는 celsius 변수와
 * 화씨온도를 나타내는 fahrenheit 변수가 있습니다.
 * celsius 변수에는 30 이 할당되어 있습니다.
 * 섭씨 30도를 화씨온도로 변경하면
 * 화씨 86도가 됩니다.
 * 섭씨온도를 화씨온도로 변경해
 * fahrenheit 변수에 할당하고 출력해보세요.
 * 변경공식: (섭씨 × 9/5) + 32 = 화씨
 */
 }
}
```

# 6. 연산자

- Java가 지원하는 연산자의 종류
  - 비교 연산자

| 비교연산자  |        |           |                     |                           |
|--------|--------|-----------|---------------------|---------------------------|
| 연산자 기호 | 설명     | 예제        | 예제 설명               | 결과<br>$a = 10, b = 5$ 일 때 |
| $= =$  | 같다     | $a == b$  | $a$ 와 $b$ 가 같다      | false                     |
| $>$    | 크다     | $a > b$   | $a$ 가 $b$ 보다 크다     | true                      |
| $> =$  | 크거나 같다 | $a > = b$ | $a$ 가 $b$ 보다 크거나 같다 | true                      |
| $<$    | 작다     | $a < b$   | $a$ 가 $b$ 보다 작다     | false                     |
| $< =$  | 작거나 같다 | $a < = b$ | $a$ 가 $b$ 보다 작거나 같다 | false                     |
| $! =$  | 다르다    | $a != b$  | $a$ 와 $b$ 가 다르다     | true                      |

# 6. 연산자

---

- 비교연산자는 값과 값을 비교할 때 사용되며 그 결과는 항상 불린(boolean)입니다.

```
public static void main(String[] args) {
 int num1 = 10;
 int num2 = 5;
 boolean isEqual = num1 == num2;
 boolean isNum1GreaterThanNum2 = num1 > num2;
 boolean isNum1GreaterOrEqualsThanNum2 = num1 >= num2;
 boolean isNum1LessThanNum2 = num1 < num2;
 boolean isNum1LessOrEqualsThanNum2 = num1 <= num2;
 boolean isNotEqual = num1 != num2;

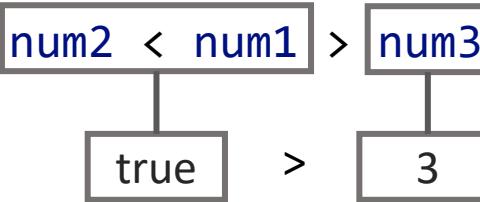
 System.out.println(isEqual);
 System.out.println(isNum1GreaterThanNum2);
 System.out.println(isNum1GreaterOrEqualsThanNum2);
 System.out.println(isNum1LessThanNum2);
 System.out.println(isNum1LessOrEqualsThanNum2);
 System.out.println(isNotEqual);
}
```

# 6. 연산자

- 아래와 같은 비교는 할 수 없습니다.

```
public static void main(String[] args) {
 int num1 = 10;
 int num2 = 5;
 int num3 = 3;

 boolean isGreaterThanAll = num2 < num1 > num3;
}
```



The diagram illustrates the evaluation of the expression `num2 < num1 & num3 > num1`. It shows two separate conditions in boxes: `num2 < num1` (which evaluates to `true`) and `num3 > num1` (which evaluates to `false`). A horizontal arrow between the boxes indicates the logical AND operation, resulting in a final value of `false`.

# 6. 연산자

- Java가 지원하는 연산자의 종류
  - 논리 연산자
  - 논리 연산자의 피연산자는 항상 불린이며
  - 논리 연산의 결과도 불린 입니다.

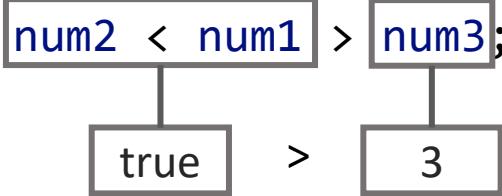
| 논리연산자                   |     |                             |                                                |                                                                |
|-------------------------|-----|-----------------------------|------------------------------------------------|----------------------------------------------------------------|
| 연산자 기호                  | 설명  | 예제                          | 예제 설명                                          | 결과                                                             |
| <code>&amp;&amp;</code> | 그리고 | <code>a &amp;&amp; b</code> | <code>a</code> 와 <code>b</code> 가 모두 true다     | <code>a</code> 와 <code>b</code> 가 모두 true일 때 true              |
| <code>  </code>         | 또는  | <code>a    b</code>         | <code>a</code> 와 <code>b</code> 중 하나 이상이 true다 | <code>a</code> 와 <code>b</code> 중 하나라도 true일 때 true            |
| <code>!</code>          | 부정  | <code>!a</code>             | <code>a</code> 의 반대                            | <code>a</code> 가 true일 때 false, <code>a</code> 가 false일 때 true |

# 6. 연산자

- 아래와 같은 연산은 비교/논리연산이 함께 사용되어야 합니다.

```
public static void main(String[] args) {
 int num1 = 10;
 int num2 = 5;
 int num3 = 3;

 boolean isGreaterThanAll = num2 < num1 > num3;
}
```

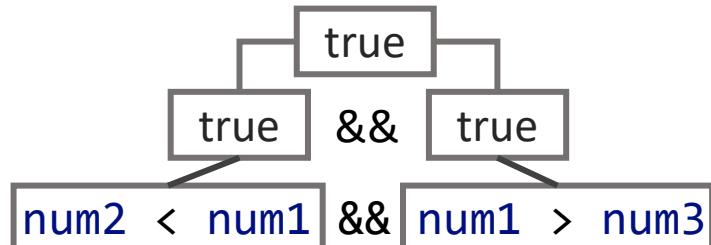


# 6. 연산자

- num1이 num2 보다 크다. 그리고 num3보다 커야 한다.

```
public static void main(String[] args) {
 int num1 = 10;
 int num2 = 5;
 int num3 = 3;

 boolean isGreaterThanAll = num2 < num1 && num1 > num3;
 System.out.println(isGreaterThanAll);
}
```



# 6. 연산자

- 논리 연산자 상세 비교

| 논리연산자 | 값1    | 값2    | 연산          | 결과    |
|-------|-------|-------|-------------|-------|
| &&    | true  | true  | 값1 && 값2    | true  |
| &&    | true  | false | 값1 && 값2    | false |
| &&    | false | true  | 값1 && 값2    | false |
| &&    | false | false | 값1 && 값2    | false |
|       | true  | true  | 값1    값2    | true  |
|       | true  | false | 값1    값2    | true  |
|       | false | true  | 값1    값2    | true  |
|       | false | false | 값1    값2    | false |
| !     | true  | true  | !(값1 && 값2) | false |
| !     | true  | false | !(값1 && 값2) | true  |
| !     | false | true  | !(값1 && 값2) | true  |
| !     | false | false | !(값1 && 값2) | true  |
| !     | true  | true  | !(값1    값2) | false |
| !     | true  | false | !(값1    값3) | false |
| !     | false | true  | !(값1    값2) | false |
| !     | false | false | !(값1    값3) | true  |

# 6. 연산자

---

- and 연산 간단한 예제

```
public static void main(String[] args) {
 boolean and = true && true;
 System.out.println(and); // true

 and = true && false;
 System.out.println(and); // false

 and = false && true; // dead code
 System.out.println(and); // false

 and = false && false; // dead code
 System.out.println(and); // false
}
```

# 6. 연산자

---

- dead code
  - 실행되지 않는 무의미한 코드를 말합니다.
  - 아래의 노란 블럭이 dead code 입니다.

```
public static void main(String[] args) {
 boolean and = true && true;
 System.out.println(and); // true

 and = true && false;
 System.out.println(and); // false

 and = false && true; // dead code
 System.out.println(and); // false

 and = false && false; // dead code
 System.out.println(and); // false
}
```

# 6. 연산자

---

- and연산에서 dead code가 발생되는 원인.
  - and는 피연산자 모두 true일 때 true를 반환합니다.
  - 피연산자 중 하나라도 false면 결과는 무조건 false가 됩니다.
- Case1 **true && true**
  - 첫 번째 피연산자가 true 이기 때문에
  - 두 번째 피연산자가 true인지 검증이 필요합니다.
- Case2 **false && true**
  - 첫 번째 피연산자가 false 이기 때문에
  - 두 번째 피연산자를 검증하지 않아도 결과는 false로 확정이 됩니다.
  - 즉, 두 번째 피연산자는 dead code가 됩니다.

# 6. 연산자

---

- || 연산 간단한 예제

```
public static void main(String[] args) {
 boolean or = true || true; // dead code
 System.out.println(or); // true

 or = true || false; // dead code
 System.out.println(or); // true

 or = false || true;
 System.out.println(or); // true

 or = false || false;
 System.out.println(or); // false
}
```

# 6. 연산자

---

- || 연산에서 dead code가 발생되는 원인.
- || 는 피연산자 중 하나라도 true면 결과는 무조건 true가 됩니다.
- Case1    **true** || **true**
  - 첫 번째 피연산자가 true 이기 때문에
  - 두 번째 피연산자를 검증하지 않아도 결과는 true로 확정이 됩니다.
  - 즉, 두 번째 피연산자는 dead code가 됩니다.
- Case2    **false** || **true**
  - 첫 번째 피연산자가 false 이기 때문에
  - 두 번째 피연산자가 true인지 검증이 필요합니다.

# 6. 연산자

---

- ! 연산 간단한 예제

```
public static void main(String[] args) {
 boolean not = !true;
 System.out.println(not); // false

 not = !false;
 System.out.println(not); // true
}
```

- ! 연산자는 불린의 결과를 부정하는 연산자 입니다.
- 나중에 배열 String 연산에서 자주 사용됩니다.

# 6. 연산자

---

- 비교연산자와 논리연산자의 동시 사용
  - 논리연산자만 이용해서 비교하는 경우는 없습니다.
  - 연산의 결과가 불린인 경우는 모두 논리연산이 가능합니다.

```
public static void main(String[] args) {
 int a = 10;
 // 변수 a의 값이 10보다 크고 20보다 작은가?
 boolean is11To19 = a > 10 && a < 20;
 System.out.println(is11To19);

 int b = 345795;
 // 변수 b는 짝수인가?
 boolean isEven = b % 2 == 0;
 System.out.println(isEven);

 int c = -345796;
 // 변수 c는 0보다 큰 짝수인가?
 boolean isEven2 = c > 0 && c % 2 == 0;
 System.out.println(isEven2);
}
```

# 6. 연산자

---

- 비교연산자와 논리연산자의 동시 사용

```
public static void main(String[] args) {
 int d = 35;
 // 변수 d의 값은 2 또는 5의 배수인가?
 boolean isMultiple2Or5 = d % 2 == 0 || d % 5 == 0;
 System.out.println(isMultiple2Or5);

 int son = 7;
 int parent = 40;
 // son은 부모님과 함께 12세 이상 관람과의 영화를 볼수 있나?
 boolean isAvailable = son >= 12 || parent >= 12;
 System.out.println(isAvailable);
}
```

# 6. 연산자

---

- 비교연산자와 논리연산자의 동시 사용

```
public static void main(String[] args) {
 int e = 35;
 // 변수 e의 값은 3의 배수가 아닌가?
 boolean isNotMultiple3 = e % 3 != 0;
 System.out.println(isNotMultiple3);

 isNotMultiple3 = !(e % 3 == 0);
 System.out.println(isNotMultiple3);
}
```

if – else if – else : 경우의 수 중 하나.

switch : 실행 흐름의 분리

while : ~ 동안 반복

for : ~ 까지 반복

## 7. 실행흐름 제어

## **7. 실행흐름 제어: if – else if – else**

# 7. 실행흐름 제어: if – else if – else

---

- 변수의 값에 따라 실행의 흐름을 바꾸어야 할 때 사용할 수 있는
- if - else if – else
  - 여러 경우의 수 중 단 하나의 경우만 실행합니다.

```
public static void main(String[] args) {
 if (첫 번째 비교연산 또는 논리연산) {
 // 첫 번째 비교연산 또는 논리연산의 결과가 true일 때 실행될 코드
 }
 else if (두 번째 비교연산 또는 논리연산) {
 // 첫 번째 비교연산 또는 논리연산의 결과가 false이고
 // 두 번째 비교연산 또는 논리연산의 결과가 true일 때 실행될 코드
 }
 else {
 // 첫 번째 비교연산 또는 논리연산의 결과가 false이고
 // 두 번째 비교연산 또는 논리연산의 결과가 false일 때 실행될 코드
 }
}
```

# 7. 실행흐름 제어: if – else if – else

---

- 변수의 값에 따라 실행의 흐름을 바꾸어야 할 때 사용할 수 있는
- if - else if – else
  - 여러 경우의 수 중 단 하나의 경우만 실행합니다.

```
public static void main(String[] args) {
 int number = 5;
 if (number == 5) {
 // number가 5와 같을 경우 실행될 코드.
 }
 else {
 // number가 5가 아닐 경우 실행될 코드.
 }
}
```

# 7. 실행흐름 제어: if – else if – else

---

- 어떤 문장이 출력될까요?

```
public static void main(String[] args) {
 int number = 5;
 if (number == 5) {
 // number가 5와 같을 경우 실행될 코드.
 System.out.println("숫자가 5와 같습니다.");
 }
 else {
 // number가 5가 아닐 경우 실행될 코드.
 System.out.println("숫자가 5가 아닙니다.");
 }
}
```

# 7. 실행흐름 제어: if – else if – else

- 변수의 범위

```
public static void main(String[] args) {
 int number = 5;

 if (number == 5) {
 int number2 = 1;
 System.out.println(number);
 System.out.println(number2);
 }
 else {
 int number2 = 2;
 System.out.println(number);
 System.out.println(number2);
 }

 System.out.println(number);
 System.out.println(number2);
}
```

— number2 변수의 사용 가능 범위

— number 변수의 사용 가능 범위

— number2 변수의 사용 가능 범위

# 7. 실행흐름 제어: if – else if – else

---

- 여러 조건을 사용할 수도 있습니다.
- 어떤 문장이 출력될까요?

```
public static void main(String[] args) {
 int number = 7;
 if (number == 5) {
 // number가 5와 같을 경우 실행될 코드.
 System.out.println("숫자가 5와 같습니다.");
 }
 else if (number == 7) {
 // number가 7과 같을 경우 실행될 코드.
 System.out.println("숫자가 7과 같습니다.");
 }
 else {
 // number가 5, 7이 아닐 경우 실행될 코드.
 System.out.println("숫자가 5, 7이 아닙니다.");
 }
}
```

# 7. 실행흐름 제어: if – else if – else

---

- 난수를 이용해 간단한 Up/Down 게임 만들어 보기

```
public static void main(String[] args) {
 // double 타입의 난수
 double randomNumber = Math.random();

 // double타입의 난수를 정수로 변환 0 ~ 99
 int answer = (int) (randomNumber * 100);
 int value = 60;

 if (answer == value) {
 System.out.println("정답입니다!");
 }
 else if (answer > value) {
 System.out.println("UP!");
 }
 else if (answer < value) {
 System.out.println("DOWN!");
 }

 System.out.println("정답은 " + answer + "입니다.");
}
```

# 7. 실행흐름 제어: if – else if – else

---

- if – else if – else 연습 문제

```
public static void main(String[] args) {
 int korScore = 90;
 int engScore = 88;
 int mathScore = 70;
 int progScore = 80;

 int sum = 0;
 int average = 0;

 // 국어, 영어, 수학, 프로그래밍의 합계와 평균을 각각 계산해 할당하고
 // 아래 기준에 따라 성적을 출력해보세요.
 // 평균점수 95점 이상: A+
 // 평균점수 90점 이상: A
 // 평균점수 85점 이상: B+
 // 평균점수 80점 이상: B
 // 평균점수 70점 이상: C
 // 평균점수 70점 미만: F
}
```

# 7. 실행흐름 제어: if – else if – else

---

- if – else if – else 연습 문제

```
public static void main(String[] args) {
 int money = 1_000_000;

 int father = 40;
 int mother = 36;
 int daughter = 11;

 int adultOneWayFlightFare = 300_000;
 int kidOneWayFlightFare = 120_000;

 // 3인 가족이 100만원으로 비행기를 타고 편도 여행을 가려합니다.
 // 부모님의 나이는 각각 40, 36세입니다.
 // 딸의 나이는 11세입니다.
 // 성인의 비행요금은 30만원
 // 아동의 비행요금은 12만원입니다.
 // 3인 가족은 여행을 떠날 수 있을까요?
 // 여행을 떠날 수 있다면 "여행가자!"
 // 여행을 떠날 수 없다면 "다음에가자 ㅠㅠ"
 // 를 출력해보세요.
}
```

## 7. 실행흐름 제어: switch

# 7. 실행흐름 제어: switch

- 변수 값에 따라 코드 제어가 가능한 switch
  - 코드의 시작 위치 종료 위치 등을 제어할 수 있고
  - 보통 상태를 제어할 때 사용합니다.
  - case에서 break를 실행하면 흐름이 끊어지면서 switch가 종료됩니다.

```
int value = 0;
switch(value) {
 case 0:
 value가 0일 때 실행할 코드 + case 1 실행
 case 1:
 value가 1일 때 실행할 코드
 break; ← switch 종료
 case 2:
 value가 2일 때 실행할 코드 + default 실행
 default:
 위 조건이 모두 맞지 않거나 가장 마지막에 실행할 코드
}
```

# 7. 실행흐름 제어: switch

- if vs. switch

|       | if                | switch         |
|-------|-------------------|----------------|
| 실행 구문 | 여러 경우의 수 중 하나     | 여러 경우의 수 실행 가능 |
| 값 비교  | 모든 비교/논리연산자 사용 가능 | equals 비교만 가능  |
| 반환 가능 | 불가능               | 가능             |

# 7. 실행흐름 제어: switch

---

- Java 1.6 switch 예제 \_ 1 / 3

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

 final int READY = 0;
 final int UPDATE_REQUEST = 1;
 final int UPDATE_PROCESSING = 2;
 final int UPDATE_SUCCESS = 3;
 final int UPDATE_FAILURE = 4;
 final int UPDATE_COMPLETE = 5;

 int nowVersion = 1;
 int nextVersion = 2;
```

# 7. 실행흐름 제어: switch

---

- Java 1.6 switch 예제 \_ 2 / 3

```
int updateState = READY;
switch (updateState) {
 case READY:
 updateState = UPDATE_REQUEST;
 System.out.println("업데이트 준비가 완료됐습니다.");
 case UPDATE_REQUEST:
 updateState = UPDATE_PROCESSING;
 System.out.println("업데이트 요청을 했습니다.");
 case UPDATE_PROCESSING:
 if (nowVersion < nextVersion) {
 updateState = UPDATE_SUCCESS;
 System.out.println("업데이트 진행중입니다.");
 }
 else {
 updateState = UPDATE_FAILURE;
 System.out.println("업데이트 할 수 없습니다.");
 break;
 }
```

# 7. 실행흐름 제어: switch

---

- Java 1.6 switch 예제 \_ 2 / 3

```
case UPDATE_SUCCESS:
 updateState = UPDATE_COMPLETE;
 System.out.println("업데이트를 성공했습니다.");
case UPDATE_COMPLETE:
 System.out.println("업데이트를 완료했습니다.");
 break;
case UPDATE_FAILURE:
 System.out.println("업데이트를 실패했습니다.");
default:
 System.out.println("업데이트를 완료했습니다.");
}

System.out.print("마지막 업데이트 상태: ");
System.out.println(updateState);
}
```

# 7. 실행흐름 제어: switch

- Java 1.7 switch 기능 추가.
  - Java 1.6에서 switch에 사용가능한 타입은 정수가 유일.
  - Java 1.7부터 String 사용 가능.

```
public static void main(String[] args) {
 final String READY = "ready";
 final String UPDATE_REQUEST = "request";
 final String UPDATE_PROCESSING = "processing";
 final String UPDATE_SUCCESS = "success";
 final String UPDATE_FAILURE = "failure";
 final String UPDATE_COMPLETE = "complete";

 int nowVersion = 1;
 int nextVersion = 2;

 String updateState = READY;
 ... 생략 ...
}
```

# 7. 실행흐름 제어: switch

---

- Java 13 switch 리턴을 위한 yield 기능 추가
  - if와 유사한 기능을 지원

```
public static void main(String[] args) {
 String memberId = "admin";
 int memberGrade = switch(memberId) {
 case "admin":
 case "root":
 yield 1;
 case "sysopr":
 yield 2;
 default:
 yield 3;
 };
 System.out.println(memberId);
 System.out.println(memberGrade);
}
```

# 7. 실행흐름 제어: switch

---

- Java 14 switch 표현식 기능 추가.
  - switch에서 반환 가능
  - if와 유사한 기능을 지원

```
public static void main(String[] args) {
 String memberId = "admin";
 int memberGrade = switch(memberId) {
 case "admin", "root" -> 1;
 case "sysopr" -> 2;
 default -> 3;
 };
 System.out.println(memberId);
 System.out.println(memberGrade);
}
```

## 7. 실행흐름 제어: while

# 7. 실행흐름 제어: while

---

- while 반복문.
  - 조건이 false가 될 때까지 반복을 수행하는 키워드입니다.

```
while(조건) {
 반복될 코드 블럭
}
```

# 7. 실행흐름 제어: while

---

- while 반복문.
  - 예> 1부터 10까지 출력하는 코드 만들기

```
public static void main(String[] args) {
 System.out.println(1);
 System.out.println(2);
 System.out.println(3);
 System.out.println(4);
 System.out.println(5);
 System.out.println(6);
 System.out.println(7);
 System.out.println(8);
 System.out.println(9);
 System.out.println(10);
}
```

# 7. 실행흐름 제어: while

---

- while 반복문.
  - 예> 1부터 10까지 출력하는 코드 만들기 (단항 연산자 이용)

```
public static void main(String[] args) {
 int i = 0;
 System.out.println(++i);
 System.out.println(++i);
 System.out.println(++i);
 System.out.println(++i);
 System.out.println(++i);
 System.out.println(++i);
 System.out.println(++i);
 System.out.println(++i);
 System.out.println(++i);
}
```

# 7. 실행흐름 제어: while

---

- while 반복문.
  - 예> 1부터 10까지 출력하는 코드 만들기 (while 이용)

```
public static void main(String[] args) {
 int i = 0;
 while(i < 10) {
 System.out.println(++i);
 }
}
```

# 7. 실행흐름 제어: while

---

- while 반복문은 종료조건이 없는 무한반복을 위해 사용됩니다.
  - 무한반복 예제

```
public static void main(String[] args) {
 int i = 0;
 while(true) { // 반복 조건이 true이므로 무한히 반복
 System.out.println(++i);
 }
}
```

# 7. 실행흐름 제어: while

- while 무한 반복으로 Up/Down 게임 만들어보기 \_ 1 / 2

**Scanner keyboard = new Scanner(System.in);**

이 코드는 콘솔에 입력한 값을 변수로 사용할 수 있도록 해주는 코드입니다.

```
import java.util.Scanner;

public class App {

 public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 double randomNumber = Math.random();
 int answer = (int) (randomNumber * 100);
 int value = 0;
```

# 7. 실행흐름 제어: while

---

- while 무한 반복으로 Up/Down 게임 만들어보기 \_ 2 / 2

```
while(true) {
 System.out.print("숫자를 입력하세요. ");
 value = keyboard.nextInt();
 if (answer == value) {
 System.out.println("정답입니다!");
 break; // 정답일 경우 무한 반복 종료
 }
 else if (answer > value) {
 System.out.println("Up!");
 }
 else {
 System.out.println("Down!");
 }
}
}
```

## 7. 실행흐름 제어: for

# 7. 실행흐름 제어: for

---

- for 반복문.
  - 지정한 범위까지만 반복을 수행하는 반복문입니다.
  - 나중에 학습할 Array 또는 List에서 사용합니다.

```
for(초기값; 반복조건; 증감식) {
 반복할 코드
}
```

# 7. 실행흐름 제어: for

- for 반복문.
  - 예> 1부터 10까지 출력하는 코드 만들기
    - i 값은 1부터 시작
    - i 값은 반복 할 때마다 1씩 증가
    - i 값이 11보다 작을 동안 반복

```
public static void main(String[] args) {
 for(int i = 1; i < 11; i++) {
 System.out.println(i);
 }
}
```

| 반복 횟수 | i 값 | 조건     | 조건결과  | 반복 여부 |
|-------|-----|--------|-------|-------|
| 1     | 1   | i < 11 | true  | true  |
| 2     | 2   | i < 11 | true  | true  |
| 3     | 3   | i < 11 | true  | true  |
| 4     | 4   | i < 11 | true  | true  |
| 5     | 5   | i < 11 | true  | true  |
| 6     | 6   | i < 11 | true  | true  |
| 7     | 7   | i < 11 | true  | true  |
| 8     | 8   | i < 11 | true  | true  |
| 9     | 9   | i < 11 | true  | true  |
| 10    | 10  | i < 11 | true  | true  |
| 11    | 11  | i < 11 | false | false |

# 7. 실행흐름 제어: for

---

- for 반복문.
  - 예> 0부터 100 중 짝수만 출력하는 코드 만들기
    - i 값은 0부터 시작
    - i 값은 반복 할 때마다 2씩 증가
    - i 값이 101보다 작을 동안 반복

```
public static void main(String[] args) {
 for(int i = 0; i < 101; i += 2) {
 if (i > 0) {
 System.out.println(i);
 }
 }
}
```

- 변수(i)의 범위 생각해보기

# 7. 실행흐름 제어: for

---

- for 반복문.
  - 예> 1부터 100 중 짝수만 출력하는 코드 만들기
    - i 값은 1부터 시작
    - i 값은 반복 할 때마다 1씩 증가
    - i 값이 101보다 작을 동안 반복

```
public static void main(String[] args) {
 for(int i = 1; i < 101; i++) {
 if (i % 2 == 0) {
 System.out.println(i);
 }
 }
}
```

# 7. 실행흐름 제어: for

---

- for 반복문으로 구구단을 출력해보기

```
public static void main(String[] args) {
 for(int i = 2; i < 10; i++) {
 System.out.println(i + " X " + 1 + " = " + i * 1);
 System.out.println(i + " X " + 2 + " = " + i * 2);
 System.out.println(i + " X " + 3 + " = " + i * 3);
 System.out.println(i + " X " + 4 + " = " + i * 4);
 System.out.println(i + " X " + 5 + " = " + i * 5);
 System.out.println(i + " X " + 6 + " = " + i * 6);
 System.out.println(i + " X " + 7 + " = " + i * 7);
 System.out.println(i + " X " + 8 + " = " + i * 8);
 System.out.println(i + " X " + 9 + " = " + i * 9);
 }
}
```

# 7. 실행흐름 제어: for

- for 반복문으로 구구단을 출력해보기

```
public static void main(String[] args) {
 for(int i = 2; i < 10; i++) {
 System.out.println(i + " X " + 1 + " = " + i * 1);
 System.out.println(i + " X " + 2 + " = " + i * 2);
 System.out.println(i + " X " + 3 + " = " + i * 3);
 System.out.println(i + " X " + 4 + " = " + i * 4);
 System.out.println(i + " X " + 5 + " = " + i * 5);
 System.out.println(i + " X " + 6 + " = " + i * 6);
 System.out.println(i + " X " + 7 + " = " + i * 7);
 System.out.println(i + " X " + 8 + " = " + i * 8);
 System.out.println(i + " X " + 9 + " = " + i * 9);
 }
}
```

같은 출력내용에 숫자만 1씩 증가하고 있습니다.  
이런 코드들은 반복문을 이용해 간단히 줄일 수 있습니다.

# 7. 실행흐름 제어: for

---

- 이중 for 반복문으로 구구단을 출력해보기

```
public static void main(String[] args) {
 for(int i = 2; i < 10; i++) {
 for(int j = 1; j < 10; j++) {
 System.out.println(i + " X " + j + " = " + i * j);
 }
 }
}
```

- 변수(i, j)의 범위 생각해보기

# 7. 실행흐름 제어: for

---

- for 반복문 연습문제
  - 1. 1 부터 100 까지의 합을 구해 출력해보세요.
  - 2. 1 부터 100 중 홀수의 합을 구해 출력해보세요.
  - 3. 1 부터 100 중 3, 5, 6의 배수만 출력해보세요.
  - 4. 아래 결과가 나오게 반복문을 작성해보세요.

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

## **8. 메소드**

# 8. 메소드

---

- 메소드란 하나의 기능을 하는 코드의 묶음 단위입니다.
  - 메소드가 하는 일에 따라 파라미터(인자)가 필요할 수도 있습니다.
  - 또한, 처리 결과를 반환하거나 하지 않을 수도 있습니다.
- 아래 코드 처럼 1과 3을 더해 출력하는 코드가 있습니다.

```
public static void main(String[] args) {
 System.out.println(1 + 3);
}
```

- 숫자를 더하는 코드가 여러 번 있습니다.

```
public static void main(String[] args) {
 System.out.println(1 + 3);
 System.out.println(4 + 7);
 System.out.println(2 + 2);
 ...
}
```

# 8. 메소드

---

- 동일한 기능의 코드가 여러 번 중복되어 있을 경우,
- 해당 코드를 수행하는 기능을 한 번만 만들어두면 편리합니다.

```
public class App {
 public static void add() {
 System.out.println(1 + 3);
 }

 public static void main(String[] args) {
 //System.out.println(1 + 3);
 add();
 add();
 add();
 add();
 }
}
```

# 8. 메소드

---

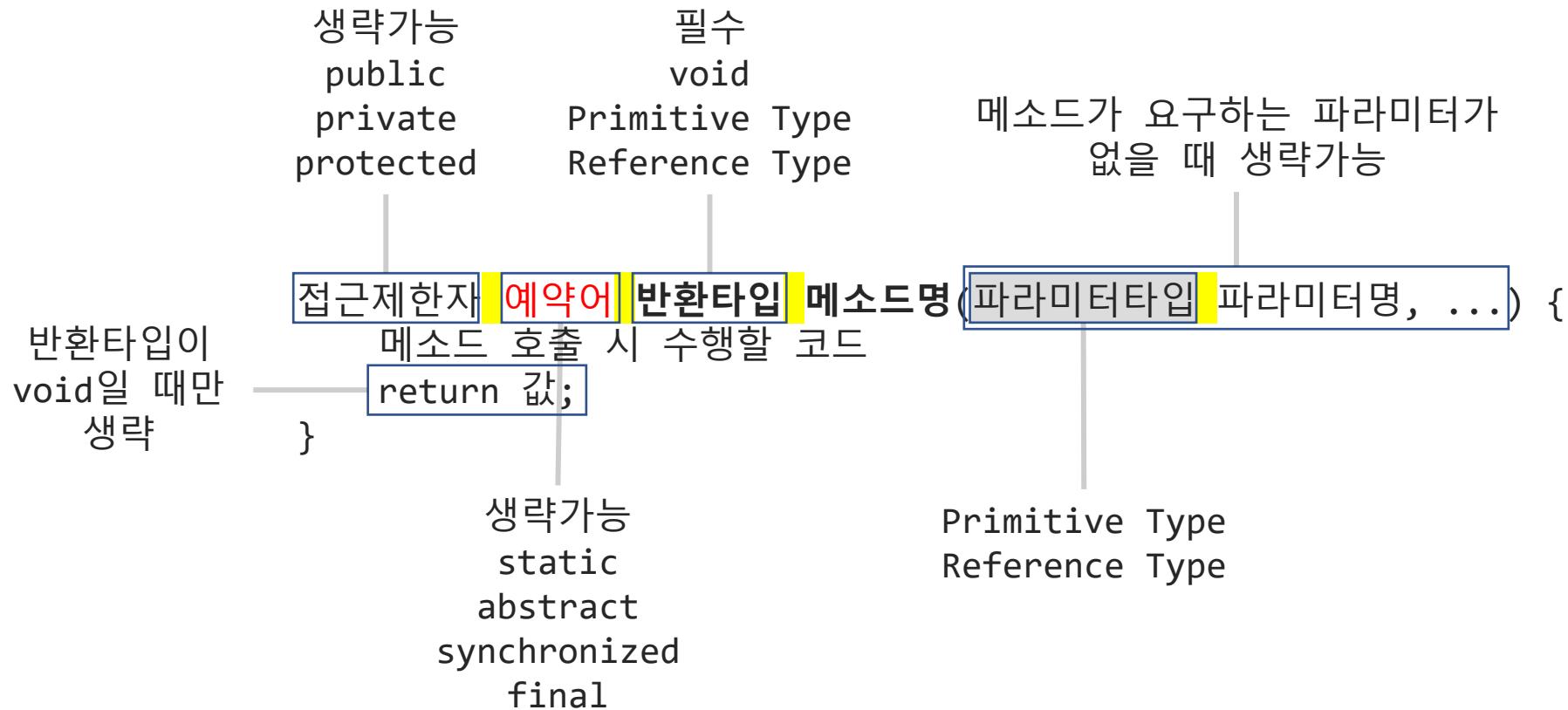
- 조금 전 "add"라는 코드를 만들어 "호출(실행)"하는 프로그램을 만들었습니다.
- 이렇게 기능단위로 코드를 분리해 두는 것을 "메소드"라고 합니다.

```
// add 메소드 정의
public static void add() {
 // add 메소드가 하는 일
 System.out.println(1 + 3);
}

public static void main(String[] args) {
 //System.out.println(1 + 3);
 add(); // add 호출(실행)
 add(); // add 호출(실행)
 add(); // add 호출(실행)
 add(); // add 호출(실행)
}
```

# 8. 메소드

- 메소드 정의 규칙



# 8. 메소드

---

- “안녕하세요?” 만 출력하는 메소드 만들고 여러 번 호출해보기

```
public class App {
 public static void sayHello() {
 System.out.println("안녕하세요?");
 }

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

# 8. 메소드

---

- 1 + 1만 계산하는 메소드를 만들고 여러 번 호출(실행)해보기

```
public class App {
 public static void printAddResult() {
 System.out.println(1 + 1);
 }

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

# 8. 메소드

---

- 메소드 실행에 필요한 데이터(파라미터)가 있을 경우,
- 메소드를 호출(실행)할 때 데이터를 반드시 전달해야 합니다.
- 만약, 이름을 전달받아 “이름씨, 안녕하세요?” 라고 출력하는 메소드를 만들어야 한다면 아래 코드처럼 작성해야 합니다.

```
public static void printHello(String name) {
 System.out.println(name + "씨, 안녕하세요?");
}
```

- String name 이라는 부분이 메소드 실행에 필요한 파라미터

# 8. 메소드

- 주어진 문자와 안녕하세요? 를 함께 출력하는 메소드를 만들고 여러 번 호출해보기
  - 예> 주어진 문자: 스타크
  - 출력결과> 스타크씨, 안녕하세요?

```
public class App {
 public static void printHello(String name) {
 System.out.println(name + "씨, 안녕하세요?");
 }

 public static void main(String[] args) {
 printHello("스타크"); // String name에 "스타크" 할당
 printHello("헐크"); // String name에 "헐크" 할당
 printHello("그루트"); // String name에 "그루트" 할당
 }
}
```

- name 변수의 범위 생각해보기

# 8. 메소드

---

- 아래 문제를 각 메소드로 만들고 호출하는 코드를 작성해보세요.
- for 반복문 연습문제
  - 1. 1 부터 100 까지의 합을 구해 출력해보세요.
  - 2. 1 부터 100 중 홀수의 합을 구해 출력해보세요.
  - 3. 1 부터 100 중 3, 5, 6의 배수만 출력해보세요.
  - 4. 아래 결과가 나오게 반복문을 작성해보세요.

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

# 8. 메소드 – 반환하는 메소드

---

- 메소드의 반환타입이 void 일 경우
  - 아무것도 반환하지 않는 메소드가 됩니다.
- 반환타입이 int인 경우
  - 정수형(int)을 반환하는 메소드가 되며, return이라는 키워드가 메소드 가장 마지막에 작성되어야 합니다.
- 반환타입이 double인 경우
  - 부동소수점형(double)을 반환하는 메소드가 되며, return이라는 키워드가 메소드 가장 마지막에 작성되어야 합니다.
- 즉, 반환타입이 void가 아닌 경우
  - 항상 작성된 자료형으로 반환하는 메소드가 되며 반드시 return 키워드가 메소드 가장 마지막에 작성 되어야 합니다.

# 8. 메소드 – 반환하는 메소드

---

- 메소드 반환은 메소드를 호출(실행)한 라인에 결과를 돌려준다는 의미입니다.

```
public class App {

 public static int getMultiply(int num, int time) {
 return num * time;
 }

 public static void main(String[] args) {
 getMultiply(10, 4);
 }
}
```

- 위 코드에서 노란색으로 표시된 영역이 메소드를 호출하므로
- 하늘색으로 표시된 영역에서 반환하는 값은
- 노란색 영역으로 40이라는 데이터를 반환하게 됩니다.

# 8. 메소드 – 반환하는 메소드

---

- 즉, `getMultiply(10, 4)`의 값을 알고 싶다면 노란색 영역처럼 값을 할당 받는 코드가 반드시 필요합니다.

```
public class App {

 public static int getMultiply(int num, int time) {
 return num * time;
 }

 public static void main(String[] args) {
 int result = getMultiply(10, 4);
 System.out.println(result);
 }
}
```

# 8. 메소드 – 반환하는 메소드

---

- 산술 연산자 연습 문제를 분, 초를 파라미터로 받으면 초를 반환하는 메소드로 만들어 호출하고 결과를 출력해보세요.

```
public class ArithmaticProblem1 {

 public static void main(String[] args) {
 int minutes = 5;
 int seconds = 50;
 int time = 0;

 /*
 * 산술연산자를 이용해
 * minutes와 seconds의 값을 초로 변환해
 * time 변수에 할당하고 출력해보세요.
 */
 }
}
```

# 8. 메소드 – 반환하는 메소드

---

- 산술 연산자 연습 문제를 processTime을 파라미터로 받아  
분을 반환, processTime을 파라미터로 받아  
초를 반환하는 메소드로 만들어 호출하고 결과를 출력해보세요.

```
public class ArithmaticProblem2 {

 public static void main(String[] args) {
 int processTime = 145;
 int minutes = 0;
 int seconds = 0;
 /*
 * 산술 연산자를 이용해
 * processTime을 분(Minute), 초(Second)
 * 를 구한다음 minutes, seconds 변수에 할당하고
 * 출력해보세요.
 */
 }
}
```

# 8. 메소드 – 반환하는 메소드

---

- 산술 연산자 연습 문제를 섭씨를 파라미터로 받아 화씨를 반환하는 메소드로 만들어 호출하고 결과를 출력해보세요.

```
public class ArithmaticProblem3 {

 public static void main(String[] args) {
 int celsius = 30;
 int fahrenheit = 0;
 /*
 * 섭씨온도를 나타내는 celsius 변수와
 * 화씨온도를 나타내는 fahrenheit 변수가 있습니다.
 * celsius 변수에는 30 이 할당되어 있습니다.
 * 섭씨 30도를 화씨온도로 변경하면
 * 화씨 86도가 됩니다.
 * 섭씨온도를 화씨온도로 변경해
 * fahrenheit 변수에 할당하고 출력해보세요.
 * 변경공식: (섭씨 × 9/5) + 32 = 화씨
 */
 }
}
```

# 8. 메소드 – 반환하는 메소드

---

- if – else if – else 연습 문제에서 아래 메소드 만들고 호출 실습
  - 과목별 점수를 파라미터로 받아 합계를 구해 반환하는 메소드
  - 합계와 4를 파라미터로 받아 평균을 구해 반환하는 메소드
  - 평균을 파라미터로 받아 등급을 반환하는 메소드

```
public static void main(String[] args) {
 int korScore = 90;
 int engScore = 88;
 int mathScore = 70;
 int progScore = 80;
 int sum = 0;
 int average = 0;

 // 평균점수 95점 이상: A+
 // 평균점수 90점 이상: A
 // 평균점수 85점 이상: B+
 // 평균점수 80점 이상: B
 // 평균점수 70점 이상: C
 // 평균점수 70점 미만: F
}
```

# 8. 메소드 – 반환하는 메소드

---

- if – else if – else 연습 문제에서 아래 메소드 만들고 호출 실습
  - 나이를 파라미터로 받아 편도 비행요금을 계산해 반환하는 메소드

```
public static void main(String[] args) {
 int money = 1_000_000;

 int father = 40;
 int mother = 36;
 int daughter = 11;

 int adultOneWayFlightFare = 300_000;
 int kidOneWayFlightFare = 120_000;

 // 3인 가족이 100만원으로 비행기를 타고 편도 여행을 가려합니다.
 // 부모님의 나이는 각각 40, 36세이며, 딸의 나이는 11세입니다.
 // 성인의 비행요금은 30만원
 // 아동의 비행요금은 12만원입니다.
 // 3인 가족은 여행을 떠날 수 있을까요?
 // 여행을 떠날 수 있다면 "여행가자!"
 // 여행을 떠날 수 없다면 "다음에가자 ㅠㅠ" 를 출력해보세요.
}
```

# 9. 클래스 / 인스턴스 / 생성자

# 9. 클래스와 인스턴스

# 9. 클래스와 인스턴스

---

- 클래스는 객체지향프로그래밍에서 가장 중요한 개념입니다.
- 객체지향프로그래밍 (OOP)
  - 수행 과제나 문제를 실세계의 객체로 표현해
  - 사람이 알기 쉽게 풀어나갈 수 있는 방법을 말합니다.
- 클래스란,
  - 존재할 수 있는 사물을 추상화 시켜놓은 개념
  - Java에서는 Reference Type의 자료형에 해당합니다.
- 객체란,
  - 클래스를 변수로 만든 것을 의미
  - 일반적으로 객체는 “인스턴스” 라고 부릅니다.
  - 클래스를 인스턴스로 만드는 과정을 “인스턴스화” 한다라고 표현.
- 클래스를 인스턴스로 만드는 인스턴스화의 과정은
  - “생성자 메소드” 를 호출하는 것과 동일합니다.

# 9. 클래스와 인스턴스

---

- 실세계에 존재하는 모든 질량이 있는 사물들은
  - 아래 두 가지로 표현이 가능합니다.
    - 속성
    - 기능(행동)
- 속성이란, 사물이 가지는 여러 특성/특징 들을 말합니다.
  - 예> 자동차의 속성
    - 시동 여부
    - 속도

# 9. 클래스와 인스턴스

---

- 기능이란, 클래스(객체)가 할 수 있는 기능(행동)을 말합니다.
  - 예> 자동차의 기능
    - 시동버튼을 눌러 시동을 건다.
      - 엔진에 기본 동력이 제공되며 자동차가 출발할 준비를 한다.
    - 가속 페달을 밟는다.
      - 가속 페달을 밟는 만큼 엔진에 동력이 제공된다.
    - 브레이크 페달을 밟는다.
      - 브레이크 페달을 밟으면 바퀴가 서서히 멈춘다.
    - 시동버튼을 눌러 시동을 끈다.
      - 엔진에 필요한 기본 동력을 제공하지 않아 자동차가 완전히 멈춘다.

# 9. 클래스와 인스턴스

---

- 클래스의 속성
  - “변수”로 정의
  - “멤버변수” 혹은 “인스턴스 필드”라고 부릅니다.
- 클래스의 기능
  - “메소드”로 정의
  - “인스턴스 메소드”라고 부릅니다.
- 멤버변수의 대상
  - 인스턴스 메소드의 영향을 받는 변수들.
  - 영향을 받지 않는 변수는 제거하거나 상수로 변경해야 합니다.
- 일부 멤버변수는 메소드의 실행상태를 제어하기도 합니다.
  - 시동이 꺼져 있을 때 가속페달을 누르면 아무 일도 일어나지 않아야 합니다.

# 9. 클래스와 인스턴스

- 자동차 클래스의 설계 \_ 1 / 3

```
public class Car {
 boolean isEngineStart;
 int speed;

 public void pressEngineStartButton() {
 if (isEngineStart) {
 System.out.println("시동을 끕니다.");
 isEngineStart = false;
 speed = 0;
 }
 else {
 System.out.println("시동을 켭니다.");
 isEngineStart = true;
 speed = 10;
 }
 }
}
```

멤버변수에 값을 할당하지 않았어요!

일반적인 지역변수는 반드시 값을 할당을 해야만 합니다.

하지만, 멤버변수는 값을 미리 할당하지 않아야 합니다. (인스턴스마다 다른 값을 가지기 때문)

# 9. 클래스와 인스턴스

---

- 자동차 클래스의 설계 \_ 2 / 3

```
public void pressGasolinPedal(int press) {
 if (!isEngineStart) {
 System.out.println("먼저 시동을 켜주세요.");
 return;
 }

 speed += press;
 if (speed > 240) {
 speed = 240;
 }
 System.out.println("속도: " + speed);
}
```

# 9. 클래스와 인스턴스

---

- 자동차 클래스의 설계 \_ 3 / 3

```
public void pressBrakePedal(int press) {
 if (!isEngineStart) {
 System.out.println("먼저 시동을 켜주세요.");
 return;
 }

 speed -= press;
 if (speed < 0) {
 speed = 0;
 }
 System.out.println("속도: " + speed);
}

public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
 car.pressEngineStartButton();
}
}
```

# 9. 클래스와 인스턴스

---

- main 메소드 다시 보기 \_ 1

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
 car.pressEngineStartButton();
}
```

- 클래스를 변수로 사용하려면, 인스턴스로 만들어야 합니다.

`new Car(); // Car 클래스를 인스턴스로 생성합니다.`

- 인스턴스가 만들어지면 변수에 할당해야 사용할 수 있습니다.

`Car car = new Car(); // 인스턴스를 만들어 Car 타입의 car 변수에 할당합니다.`

- Reference Type의 변수는 "인스턴스" 라고 부르는데,

- Car 클래스는 Reference Type이므로 car 변수는 인스턴스입니다.

# 9. 클래스와 인스턴스

---

- main 메소드 다시 보기 \_ 2

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
 car.pressEngineStartButton();
}
```

- Reference Type의 인스턴스는 Primitive Type에는 없는 특별한 점 연산자 (" . " )를 사용할 수 있습니다.

```
car.pressEngineStartButton();
```

- 위 코드는 car 인스턴스의 pressEngineStartButton 메소드를 실행해라 라고 하는 명령입니다.

# 9. 클래스와 인스턴스

---

- Reference Type?
  - Reference Type은 “메모리 참조타입”을 말합니다.
  - 여러 변수들의 메모리 공간을 종류별로 묶어 그 주소를 참조
- Primitive Type과 Reference Type의 차이점.
  - Primitive Type
    - 정해진 Byte크기 만큼 메모리를 할당
    - 항상 값을 참조
  - Reference Type
    - 여러 개의 Primitive Type과 여러 개의 Reference Type,
    - 여러 개의 메소드를 사용하므로, 하나의 메모리로 사용할 수 없습니다.
    - 항상 메모리 주소를 참조

# 9. 클래스와 인스턴스

- Reference Type 메모리 참조 예

```
public static void main(String[] args) {
 Car car = new Car();
}
```

- car 인스턴스의 메모리 주소는 32가 됩니다.

|    |               |           |    |    |    |    |    |    |    |
|----|---------------|-----------|----|----|----|----|----|----|----|
| 1  | 2             | 3         | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| 11 | car           | 13        | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 |               | 23        | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32<br>boolean | 33<br>int | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42            | 43        | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52            | 53        | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62            | 63        | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72            | 73        | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82            | 83        | 84 | 85 | 86 | 87 | 88 | 89 | 90 |

# 9. 클래스와 인스턴스

- Reference Type 메모리 참조 예

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
}

public void pressEngineStartButton() {
 if (isEngineStart) {
 System.out.println("시동을 끕니다.");
 isEngineStart = false;
 speed = 0;
 }
 else {
 System.out.println("시동을 켭니다.");
 isEngineStart = true;
 speed = 10;
 }
}
```

|    |         |       |    |    |    |    |    |
|----|---------|-------|----|----|----|----|----|
| 1  | 2       | 3     | 4  | 5  | 6  | 7  | 8  |
| 11 |         | 13    | 14 | 15 | 16 | 17 | 18 |
| 21 |         | 23    | 24 | 25 | 26 | 27 | 28 |
| 31 | 32 true | 33 10 | 34 | 35 | 36 | 37 | 38 |
| 41 | 42      | 43    | 44 | 45 | 46 | 47 | 48 |
| 51 | 52      | 53    | 54 | 55 | 56 | 57 | 58 |
| 61 | 62      | 63    | 64 | 65 | 66 | 67 | 68 |
| 71 | 72      | 73    | 74 | 75 | 76 | 77 | 78 |
| 81 | 82      | 83    | 84 | 85 | 86 | 87 | 88 |

# 9. 클래스와 인스턴스

- Reference Type 메모리 참조 예

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
}

public void pressGasolinPedal(int press) {
 if (!isEngineStart) {
 System.out.println("먼저 시동을 켜주세요.");
 return;
 }

 speed += press;
 if (speed > 240) {
 speed = 240;
 }
 System.out.println("속도: " + speed);
}
```

|    |            |          |    |    |    |    |
|----|------------|----------|----|----|----|----|
| 1  | 2          | 3        | 4  | 5  | 6  | 7  |
| 11 |            | 13       | 14 | 15 | 16 | 17 |
| 21 |            | 23       | 24 | 25 | 26 | 27 |
| 31 | 32<br>true | 33<br>20 | 34 | 35 | 36 | 37 |
| 41 | 42         | 43       | 44 | 45 | 46 | 47 |
| 51 | 52         | 53       | 54 | 55 | 56 | 57 |
| 61 | 62         | 63       | 64 | 65 | 66 | 67 |
| 71 | 72         | 73       | 74 | 75 | 76 | 77 |
| 81 | 82         | 83       | 84 | 85 | 86 | 87 |

# 9. 클래스와 인스턴스

- Reference Type 메모리 참조 예

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
}

public void pressBrakePedal(int press) {
 if (!isEngineStart) {
 System.out.println("먼저 시동을 켜주세요.");
 return;
 }

 speed -= press;
 if (speed < 0) {
 speed = 0;
 }
 System.out.println("속도: " + speed);
}
```

|    |            |          |    |    |    |    |
|----|------------|----------|----|----|----|----|
| 1  | 2          | 3        | 4  | 5  | 6  | 7  |
| 11 |            | 13       | 14 | 15 | 16 | 17 |
| 21 |            | 23       | 24 | 25 | 26 | 27 |
| 31 | 32<br>true | 33<br>10 | 34 | 35 | 36 | 37 |
| 41 | 42         | 43       | 44 | 45 | 46 | 47 |
| 51 | 52         | 53       | 54 | 55 | 56 | 57 |
| 61 | 62         | 63       | 64 | 65 | 66 | 67 |
| 71 | 72         | 73       | 74 | 75 | 76 | 77 |
| 81 | 82         | 83       | 84 | 85 | 86 | 87 |

# 9. 클래스와 인스턴스

- Reference Type 메모리 참조 예

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
 car.pressEngineStartButton();
}

public void pressEngineStartButton() {
 if (isEngineStart) {
 System.out.println("시동을 끕니다.");
 isEngineStart = false;
 speed = 0;
 }
 else {
 System.out.println("시동을 켭니다.");
 isEngineStart = true;
 speed = 10;
 }
}
```

|    |             |         |    |    |    |    |    |
|----|-------------|---------|----|----|----|----|----|
| 1  | 2           | 3       | 4  | 5  | 6  | 7  | 8  |
| 11 |             | 13      | 14 | 15 | 16 | 17 | 18 |
| 21 |             | 23      | 24 | 25 | 26 | 27 | 28 |
| 31 | 32<br>false | 33<br>0 | 34 | 35 | 36 | 37 | 38 |
| 41 | 42          | 43      | 44 | 45 | 46 | 47 | 48 |
| 51 | 52          | 53      | 54 | 55 | 56 | 57 | 58 |
| 61 | 62          | 63      | 64 | 65 | 66 | 67 | 68 |
| 71 | 72          | 73      | 74 | 75 | 76 | 77 | 78 |
| 81 | 82          | 83      | 84 | 85 | 86 | 87 | 88 |

# 9. 클래스와 인스턴스

- Reference Type 메모리 참조 예

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
 car.pressEngineStartButton();
}
```

- 메소드가 종료되면 지역변수(인스턴스 포함)는 사라집니다.

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |

# 9. 클래스와 인스턴스

- JVM(Java Virtual Machine)의 메모리 구조

## Method (Static)

상수(final) 명과 타입,  
멤버변수 명과 타입, 생성자, 인스턴스 메소드,  
클래스 변수(static) 명과 타입, 클래스 메소드(static),  
클래스 또는 인터페이스의 여부, 클래스 이름(타입), 슈퍼클래스의 이름(타입)

접근 가능

접근 불가능

접근 가능

접근 불가능

## Stack

파라미터, 지역변수, 반환 값,  
연산에 사용된 임시 변수 등

## Heap

new 키워드로 생성된  
객체 또는 배열 등의 Reference Type 객체  
(Method 영역에 적재된 클래스만 생성이 가능)

# 9. 클래스와 인스턴스

- Car 클래스의 구조와 메모리

## Method (Static)

### Class

Name: Car, Type: Class, Super class: Object

### Field

Name: isEngineStart, Type: boolean

Name: speed, Type: int

### Method

Name: pressEngineStart, Return type: void

Name: pressGasolinPedal, Return type: void, Parameter type: int

Name: pressBrakePedal, Return type: void, Parameter type: int

Name: main, Return type: void, Parameter type: String[], static

# 9. 클래스와 인스턴스

- Car 클래스의 구조와 메모리

```
public static void main(String[] args) {
 Car car = new Car();
}
```

Stack

Name: args, Type: String[]  
Name: car, Type: Car

Heap

Car

isEngineStart: false  
speed: 0

# 9. 클래스와 인스턴스

- Car 클래스의 구조와 메모리

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
}

public void pressEngineStartButton() {
 if (isEngineStart) {
 System.out.println("시동을 끍니다.");
 isEngineStart = false;
 speed = 0;
 }
 else {
 System.out.println("시동을 켭니다.");
 isEngineStart = true;
 speed = 10;
 }
}
```

## Stack

Name: args, Type: String[]  
Name: car, Type: Car

## Heap

### Car

isEngineStart: false → true  
speed: 0 → 10

# 9. 클래스와 인스턴스

- Car 클래스의 구조와 메모리

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
}

public void pressGasolinPedal(int press) {
 if (!isEngineStart) {
 System.out.println("먼저 시동을 켜주세요.");
 return;
 }

 speed += press;
 if (speed > 240) {
 speed = 240;
 }
 System.out.println("속도: " + speed);
}
```

## Stack

Name: args, Type: String[]  
Name: car, Type: Car  
Name: press, Type: int, value: 10  
Name: temp: Type: int, value: 20

## Heap

### Car

isEngineStart: true  
speed: 10 → 20

# 9. 클래스와 인스턴스

- Car 클래스의 구조와 메모리

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
}
```

```
public void pressBrakePedal(int press) {
 if (!isEngineStart) {
 System.out.println("먼저 시동을 켜주세요.");
 return;
 }

 speed -= press;
 if (speed < 0) {
 speed = 0;
 }
 System.out.println("속도: " + speed);
}
```

## Stack

Name: args, Type: String[]  
Name: car, Type: Car  
Name: press, Type: int, value: 10  
Name: temp: Type: int, value: 10

## Heap

### Car

isEngineStart: true  
speed: 20 → 10

# 9. 클래스와 인스턴스

- Car 클래스의 구조와 메모리

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
 car.pressEngineStartButton();
}

public void pressEngineStartButton() {
 if (isEngineStart) {
 System.out.println("시동을 끕니다.");
 isEngineStart = false;
 speed = 0;
 }
 else {
 System.out.println("시동을 켭니다.");
 isEngineStart = true;
 speed = 10;
 }
}
```

## Stack

Name: args, Type: String[]  
Name: car, Type: Car

## Heap

### Car

isEngineStart: true → false  
speed: 10 → 0

# 9. 클래스와 인스턴스

- Car 클래스의 구조와 메모리

```
public static void main(String[] args) {
 Car car = new Car();
 car.pressEngineStartButton();
 car.pressGasolinPedal(10);
 car.pressBrakePedal(10);
 car.pressEngineStartButton();
}
```



# 9. 클래스와 인스턴스

- 학생 평균 및 학점 구하기 실습

| Student                   |
|---------------------------|
| java: int                 |
| python: int               |
| cpp: int                  |
| csharp: int               |
| getSumAllScores(): int    |
| getAverage(): double      |
| getCourseCredit(): double |
| getABCDE(): String        |

멤버변수

인스턴스 메소드

- getSumAllScores(): int
  - 멤버변수를 모두 더해 int 타입으로 반환
- getAverage(): double
  - 합계점수 / 4의 결과를
  - 소수점 아래 두 자리 까지만 반환.
- getCourseCredit(): double
  - (평균점수 – 55) / 10의 결과를
  - 소수점 아래 두 자리 까지만 반환
- getABCDE(): String
  - 학점을 아래 기준에 맞춰 반환
    - 4.1 ~ 4.5: A+
    - 3.6 ~ 4.0: A
    - 3.1 ~ 3.5: B+
    - 2.6 ~ 3.0: B
    - 1.6 ~ 2.5: C
    - 0.6 ~ 1.5: D
    - 0.1 ~ 0.5: F

# 9. 클래스와 인스턴스

---

- 학생 평균 및 학점 구하기 실습
  - (다 풀어본 후 다음 슬라이드를 보세요.)
  - 호출 코드 및 결과

```
public static void main(String[] args) {
 Student student = new Student();
 student.java = 100;
 student.python = 97;
 student.cpp = 81;
 student.csharp = 99;

 int sum = student.getSumAllScores();
 double average = student.getAverage();
 double courseCredit = student.getCourseCredit();
 String abcde = student.getABCDE();

 System.out.println("합계: " + sum); //377
 System.out.println("평균: " + average); // 95.25
 System.out.println("학점: " + courseCredit); // 3.92
 System.out.println("등급: " + abcde); // A
}
```

# 9. 클래스와 인스턴스

---

- Student 클래스 내용 \_ 1 / 3

```
public class Student {
 int java;
 int python;
 int cpp;
 int csharp;

 public int getSumAllScores() {
 return java + python + cpp + csharp;
 }

 public double getAverage() {
 int average = (int) (getSumAllScores() / 4.0 * 100);
 return average / 100.0;
 }

 public double getCourseCredit() {
 int courseCredit = (int) ((getAverage() - 55) / 10.0 * 100);
 return courseCredit / 100.0;
 }
}
```

# 9. 클래스와 인스턴스

---

- Student 클래스 내용 \_ 2 / 3

```
public String getABCDE() {
 double courseCredit = getCourseCredit();
 if (courseCredit >= 4.1) {
 return "A+";
 } else if (courseCredit >= 3.6) {
 return "A";
 } else if (courseCredit >= 3.1) {
 return "B+";
 } else if (courseCredit >= 2.6) {
 return "B";
 } else if (courseCredit >= 1.6) {
 return "C";
 } else if (courseCredit >= 0.6) {
 return "D";
 } else {
 return "F";
 }
}
```

# 9. 클래스와 인스턴스

---

- Student 클래스 내용 \_ 3 / 3

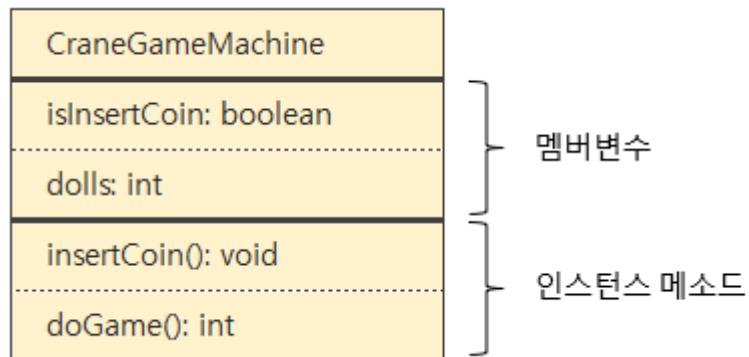
```
public static void main(String[] args) {
 Student student = new Student();
 student.java = 100;
 student.python = 97;
 student.cpp = 81;
 student.csharp = 99;

 int sum = student.getSumAllScores();
 double average = student.getAverage();
 double courseCredit = student.getCourseCredit();
 String abcde = student.getABCDE();

 System.out.println("합계: " + sum); //377
 System.out.println("평균: " + average); // 95.25
 System.out.println("학점: " + courseCredit); // 3.92
 System.out.println("등급: " + abcde); // A
}
```

# 9. 클래스와 인스턴스

- 인형뽑기 게임 흉내내기 실습
  - (다 풀어본 후 다음 슬라이드를 보세요.)



- `insertCoin()`
  - 인형뽑기 기계에 동전을 넣습니다.
  - 동전을 넣지 않으면 `doGame()`은 실행되지 않아야 합니다.
  - 동전을 넣으면 `isInsertCoin` 변수는 `true`가 되어야 합니다.
  - `dolls`의 값이 0보다 작거나 같을 경우 `isInsertCoin`은 `true`로 변하지 않습니다.
- `doGame()`
  - `isInsertCoin`이 `true` 일 때만 게임을 진행합니다.
  - 만약, 인형을 뽑았다면 1을 반환하고
  - 뽑지 못했다면 0을 반환합니다.
  - 반환하는 숫자는 랜덤하게 선정합니다.
  - 1을 반환했을 경우, `dolls` 변수에서 1을 빼야 합니다.
  - 값을 반환하기 직전에 `isInsertCoin`을 `false`로 변경해야 합니다.

# 9. 클래스와 인스턴스

---

- 인형뽑기 게임 흉내내기 실습 \_ 1 / 3

```
import java.util.Random;

public class CraneGameMachine {
 boolean isInsertCoin;
 int dolls;

 public void insertCoin() {
 if (dolls > 0) {
 isInsertCoin = true;
 }
 }
}
```

# 9. 클래스와 인스턴스

---

- 인형뽑기 게임 흉내내기 실습 \_ 2 / 3

```
public int doGame() {
 if (isInsertCoin) {
 Random random = new Random();
 // 0 ~ 1 중 하나가 랜덤하게 추출됩니다.
 int result = random.nextInt(2);
 dolls -= result;
 isInsertCoin = false;
 return result;
 }
 return 0;
}
```

# 9. 클래스와 인스턴스

---

- 인형뽑기 게임 흉내내기 실습 \_ 3 / 3

```
public static void main(String[] args) {
 CraneGameMachine cgm = new CraneGameMachine();
 cgm.isInsertCoin = false;
 cgm.dolls = 10;

 cgm.insertCoin();
 int result = cgm.doGame();

 System.out.println("isInsertCoin: " + cgm.isInsertCoin);
 System.out.println("남은 인형개수: " + cgm.dolls);
 System.out.println("뽑은 인형개수: " + result);
}
}
```

## 9. 생성자

# 9. 생성자

- 생성자
  - 클래스를 인스턴스로 만들어주는 특별한 **메소드**입니다.
  - 반드시 **new**라는 키워드로 호출해야 합니다.
  - 생성자 이름은 인스턴스로 만드려는 클래스의 이름과 동일해야 함
- 아래 코드 모두 생성자를 호출해 인스턴스를 생성합니다.

```
Car car = new Car();
CraneGameMachine cgm = new CraneGameMachine();
Student student = new Student();
```

# 9. 생성자

---

- 생성자는 인스턴스를 생성하는 메소드 입니다.
- 생성자를 정의하지 않았는데, 호출이 가능하다?
  - 생성자가 없는 클래스를 작성하면 “기본생성자” 가 자동 추가됩니다.

- ConstructorTest.java

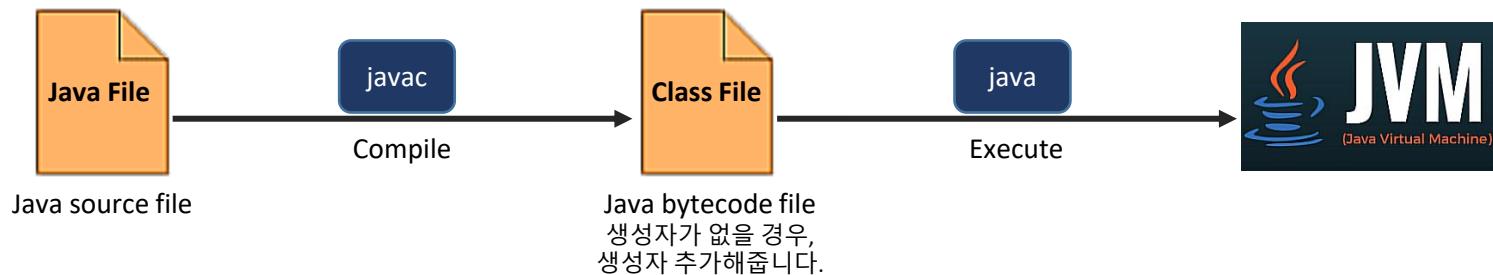
```
public class ConstructorTest {
}
```

- ConstructorTest.class

```
public class ConstructorTest {
 public ConstructorTest() {
 }
}
```

# 9. 생성자

- Java 파일이 실행되는 과정



- Java 파일을 실행하려면 바이트코드로 변환이 되어야 합니다.
- 바이트코드로 변환 할 때, 생성자 정의가 안되어 있다면
- 기본생성자를 자동으로 생성해 줍니다.

## 바이트코드

| java 파일을 바이트코드로 변경 한 후 변경된 파일을 JVM에서 실행합니다.  
| java 파일을 바이트코드로 변경하는 과정을 컴파일이라고 합니다.

# 9. 생성자

- 생성자 만들어 호출해보기.

```
public class ConstructorTest {
 public ConstructorTest() {
 System.out.println("생성자 호출되었습니다.");
 }

 public static void main(String[] args) {
 ConstructorTest ct = new ConstructorTest();
 }
}
```

생성자 메소드 호출

생성자는 반환타입이 없습니다.

생성자는 public 클래스명() 으로 선언하며, 반환타입이 없는 메소드입니다.  
때문에, return 키워드도 사용하지 못합니다.

# 9. 생성자

---

- 생성자를 직접 정의해 사용하는 두 가지 이유
  - 1. 멤버변수의 초기화(초기 값 할당)
  - 2. 인스턴스 생성과 동시에 다른 메소드를 호출

# 9. 생성자

---

- 생성자를 이용한 멤버변수 초기화 예제 \_ 1 / 2

```
public class CoffeeShop {

 int iceAmericano;
 int hotAmericano;

 public CoffeeShop() {
 iceAmericano = 2500; // 초기값 할당
 hotAmericano = 2000; // 초기값 할당
 }

 public int orderCoffee(int menu, int quantity) {
 if (menu == 1) {
 return iceAmericano * quantity;
 }
 return hotAmericano * quantity;
 }
}
```

# 9. 생성자

---

- 생성자를 이용한 멤버변수 초기화 예제 \_ 2 / 2

```
public static void main(String[] args) {
 // 생성자 호출
 CoffeeShop cs = new CoffeeShop();
 // 아이스아메리카노 5잔 주문
 int price = cs.orderCoffee(1, 5);
 System.out.println(price);
}
}
```

# 9. 생성자

- 파라미터가 있는 생성자 만들고 호출해보기
  - 커피 가격을 커피숍 인스턴스마다 다르게 할당하고 싶다면?

```
public class CoffeeShop {
 ... 생략 ...

 // 생성자에 파라미터를 추가합니다.
 public CoffeeShop(int iceAmericano, int hotAmericano) {
 iceAmericano = iceAmericano; // 초기값 할당
 hotAmericano = hotAmericano; // 초기값 할당
 }

 ... 생략 ...

 public static void main(String[] args) {
 // 생성자 호출
 CoffeeShop cs = new CoffeeShop(); // 에러!
 ... 생략 ...
 }
}
```

# 9. 생성자

---

- 기본 생성자가 없으면 자동으로 생성되는 것 아니였나요?
  - 생성자가 하나라도 존재한다면 기본생성자를 만들어주지 않습니다.
    - CoffeeShop에는 생성자가 이미 있기 때문에 기본 생성자는 호출할 수 없습니다.
    - 즉, 생성자를 호출하는 코드에 파라미터를 추가해 주어야 합니다.

```
public static void main(String[] args) {
 // 생성자 호출 (파라미터 추가)
 CoffeeShop cs = new CoffeeShop(4000, 3000);
 // 아이스아메리카노 5잔 주문
 int price = cs.orderCoffee(1, 5);
 System.out.println(price);
}
```

# 9. 생성자

- 생성자에 파라미터를 전달한 후 실행을 해보면
- 예상 결과는 20000이지만 실제 결과는 0이 나옵니다.
- 그 이유는 생성자에 있습니다.

```
public class CoffeeShop {
```

```
 int iceAmericano;
 int hotAmericano;
```

할당필요

```
 public CoffeeShop(int iceAmericano, int hotAmericano) {
```

```
 iceAmericano = iceAmericano; // 초기값 할당
 hotAmericano = hotAmericano; // 초기값 할당
```

```
}
```

파라미터 iceAmericano, hotAmericano 의 범위

파라미터가  
파라미터에게  
할당하고 있습니다.

... 생략 ...

# 9. 생성자

- 파라미터로 전달된 값들은 멤버변수에게 할당되어야 합니다.

```
public class CoffeeShop {

 int iceAmericano;
 int hotAmericano;

 public CoffeeShop(int iceAmericano, int hotAmericano) {
 this.iceAmericano = iceAmericano; // 멤버변수에 초기값 할당
 this.hotAmericano = hotAmericano; // 멤버변수에 초기값 할당
 }
 ... 생략 ...
```

- 이제 20000이 출력됩니다.

**this**

this란, 현재 호출되고 있는 인스턴스를 가리킵니다.

특히 생성자 내부에서의 this는 생성자가 만든 인스턴스를 가리킵니다.

# 9. 생성자

- this

|     | this의 의미             |
|-----|----------------------|
| 생성자 | 생성자가 만든 인스턴스 객체      |
| 메소드 | 점연산자(.)를 사용한 인스턴스 객체 |

```
public CoffeeShop(int iceAmericano, int hotAmericano) {
 this.iceAmericano = iceAmericano; // 초기값 할당
 this.hotAmericano = hotAmericano; // 초기값 할당
}
CoffeeShop cs = new CoffeeShop(4000, 3000);
```

```
public int orderCoffee(int menu, int quantity) {
 if (menu == 1) {
 return this.iceAmericano * quantity;
 }
 return this.hotAmericano * quantity;
}
int price = cs.orderCoffee(1, 5);
```

# 9. 데이터 클래스

# 9. 데이터 클래스

---

- 기능(인스턴스 메소드)이 없는 클래스도 있습니다.
  - 책, 커피, 음식 등
  - 속성은 있지만 기능이 없는 것 또한 클래스의 한 종류입니다.
  - 이런 클래스들은 데이터클래스(Data Class) 라고 부릅니다.
    - 일반적으로 VO(Value Object) 클래스로 부릅니다.
    - Java14 부터 Data Class를 쉽게 정의할 수 있게 되었습니다.

# 9. 데이터 클래스

---

- Java 14 버전 미만의 데이터 클래스

```
public class Coffee {
 String name;
 int price;

 public Coffee(String name, int price) {
 this.name = name;
 this.price = price;
 }
}
```

# 9. 데이터 클래스

- Java 14 버전 이후의 데이터 클래스

```
public record Coffee(String name, int price) {
}
```

- record Coffee 의 구조 (멤버변수, 생성자, 메소드 자동생성)

|                                      |
|--------------------------------------|
| final Coffee                         |
| final name: String                   |
| final price: int                     |
| Coffee(name: String, price: int)     |
| name(): String                       |
| price(): int                         |
| final toString(): String             |
| final hashCode(): int                |
| final equals(arg0: boolean): boolean |

# 9. 데이터 클래스

- DataClass와 record의 차이

|                       | Data Class | Record |
|-----------------------|------------|--------|
| 멤버변수 추가가 자유로운가?       | 예          | 아니오    |
| 멤버변수 재할당이 가능한가?       | 예          | 아니오    |
| 생성자 사용이 가능한가?         | 예          | 예      |
| 커스텀 생성자를 직접 만들어야 하는가? | 예          | 아니오    |
| 생성자 정의가 가능한가?         | 예          | 아니오    |
| 메소드를 추가할 수 있는가?       | 예          | 예      |

# 9. 데이터 클래스

---

- 데이터 클래스의 활용 \_ 1 / 3
  - CoffeeShop 클래스의 멤버변수를 Coffee로 변경해봅니다.

```
public class Coffee {

 String name;
 int price;

 public Coffee(String name, int price) {
 this.name = name;
 this.price = price;
 }
}
```

# 9. 데이터 클래스

- 데이터 클래스의 활용 \_ 2 / 3
  - CoffeeShop 클래스의 멤버변수를 Coffee로 변경해봅니다.

```
public class CoffeeShop {

 Coffee iceAmericano;
 Coffee hotAmericano;

 public CoffeeShop(Coffee iceAmericano, Coffee hotAmericano) {
 this.iceAmericano = iceAmericano; // 초기값 할당
 this.hotAmericano = hotAmericano; // 초기값 할당
 }

 public int orderCoffee(int menu, int quantity) {
 if (menu == 1) {
 System.out.println(this.iceAmericano.name);
 return this.iceAmericano.price * quantity;
 }
 System.out.println(this.hotAmericano.name);
 return this.hotAmericano.price * quantity;
 }
}
```

# 9. 데이터 클래스

---

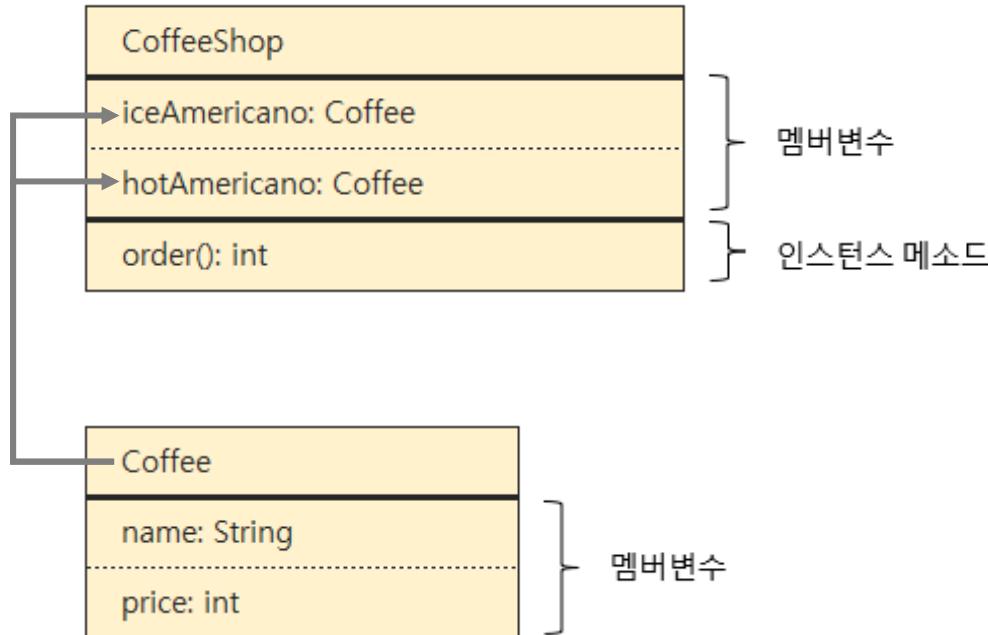
- 데이터 클래스의 활용 \_ 3 / 3
  - CoffeeShop 클래스의 멤버변수를 Coffee로 변경해봅니다.

```
public static void main(String[] args) {
 Coffee ice = new Coffee("아이스아메리카노", 2500);
 Coffee hot = new Coffee("따뜻한아메리카노", 2000);

 CoffeeShop cs = new CoffeeShop(ice, hot);
 // 아이스아메리카노 5잔 주문
 int price = cs.orderCoffee(1, 5);
 System.out.println(price);
}
}
```

# 9. 데이터 클래스

- 클래스 내에 클래스가 포함된 관계를 Has A 관계라고 합니다.
- CoffeeShop** has a **Coffee**



# 9. 데이터 클래스

---

- 클래스 has a 클래스 실습 (음료수 자판기)
  - 상품
    - 박카스 (상품명: 박카스, 가격: 900원, 재고 15개)
    - 몬스터 (상품명: 몬스터, 가격: 1500원, 재고 20개)
    - 핫식스 (상품명: 핫식스, 가격 1300원, 재고 10개)
    - 밀키스 (상품명: 밀키스, 가격 1400원, 재고 5개)
  - 기능
    - 주문하기(제품명, 주문수량): 상품
      - 몬스터를 5개 주문한다면  
상품(상품명: 몬스터, 재고: 5개, 가격: 1500원)을  
반환한다.
      - 주문을 하면 주문 수량만큼 재고가 감소된다.
      - 재고가 없다면 “상품이 품절되었습니다” 를 출력하고 null을 반환한다.
    - 입고하기(제품명, 입고수량): void
      - 입고를 하면 입고 수량만큼 재고가 증가된다.
    - 재고 출력() : void

# 9. 유틸리티 클래스

# 9. 유ти리티 클래스

---

- 속성(멤버변수)이 없는 클래스 (클래스메소드 파트에서 실습합니다.)
  - 기능만 존재하며 속성이 없는 클래스
  - 물리적으로 존재할 수 없는 형태이지만,  
OOP에서는 흔하게 사용하는 클래스 입니다.
  - 일반적으로 “Utility” 클래스를 만들 때 사용합니다.
  - 예>
    - 문자를 숫자로 변경하기
    - 문자가 숫자인지 검증하기
    - 문자에서 원하는 문자가 있는지 검사하기
    - 문자가 Null인지 검사하기 등등

# Java Programming

## 심화

---

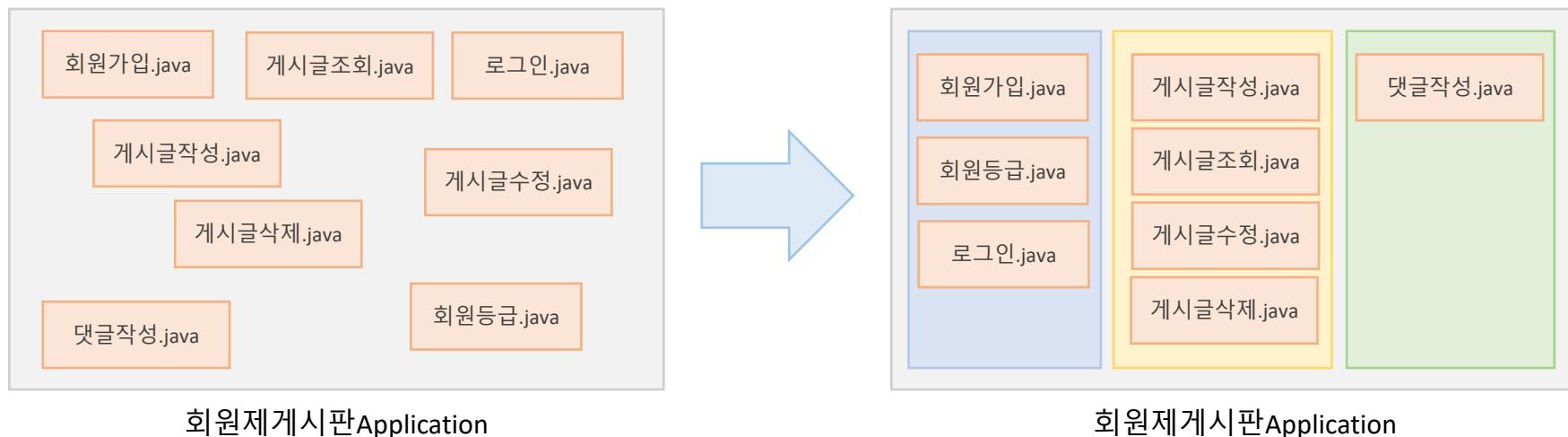
10. 패키지 / 접근제어자 / 정보은닉 / 캡슐화
11. 클래스 변수 / 클래스 메소드
12. 메소드 오버로딩
13. String
14. 배열
15. 상속(Inheritance)과 다형성(Polymorphism)
16. 메소드 오버라이딩
17. 상속의 목적

## **10. 패키지 / 접근제어지시자 / 정보은닉과 캡슐화**

# 10. 패키지 / Import

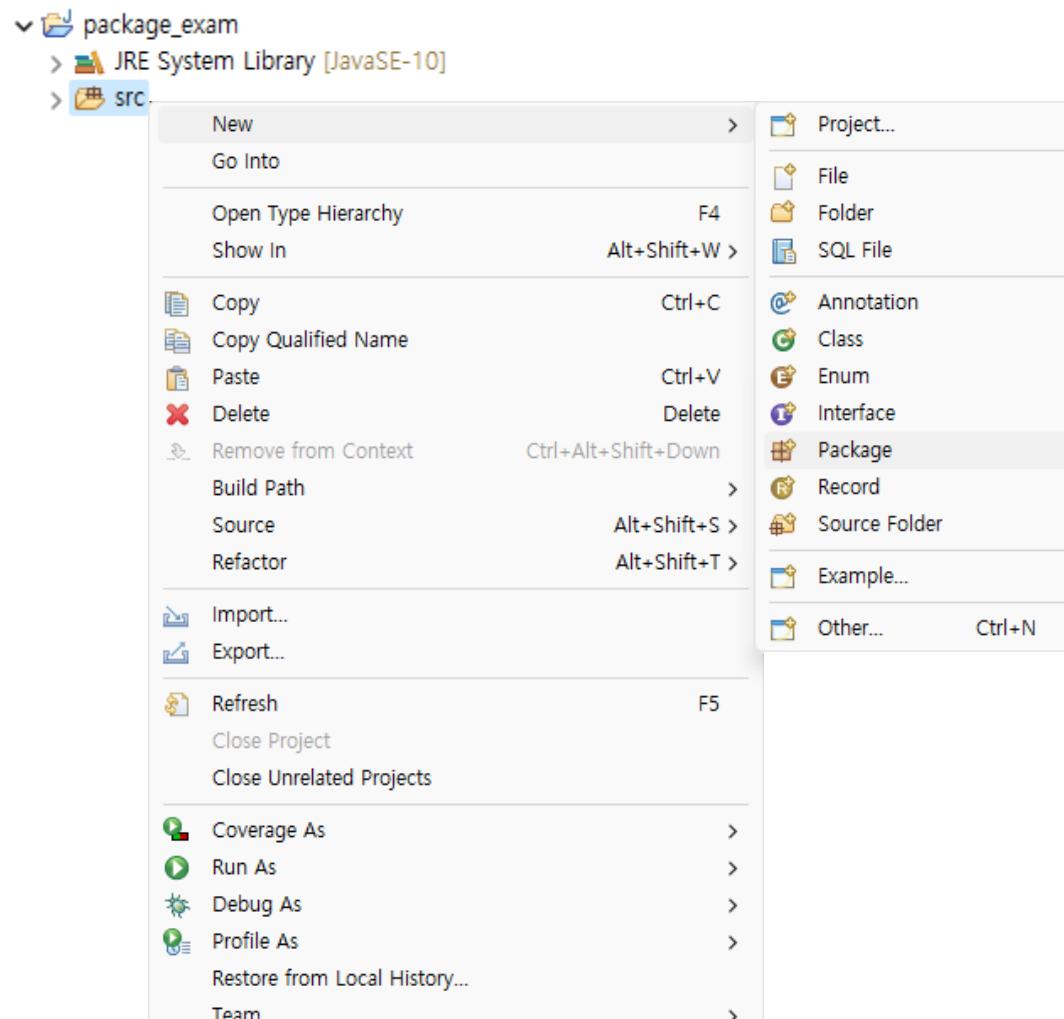
# 10. 패키지 / Import

- 패키지 (Package)
  - 여러 Java 파일 중 관련된 파일들만 정리하여 모아둔 폴더.
  - 한 패키지 내에서 같은 이름의 java 파일은 만들 수 없습니다.
  - 같은 이름의 java파일은 서로 다른 패키지에 만들 수 있습니다.



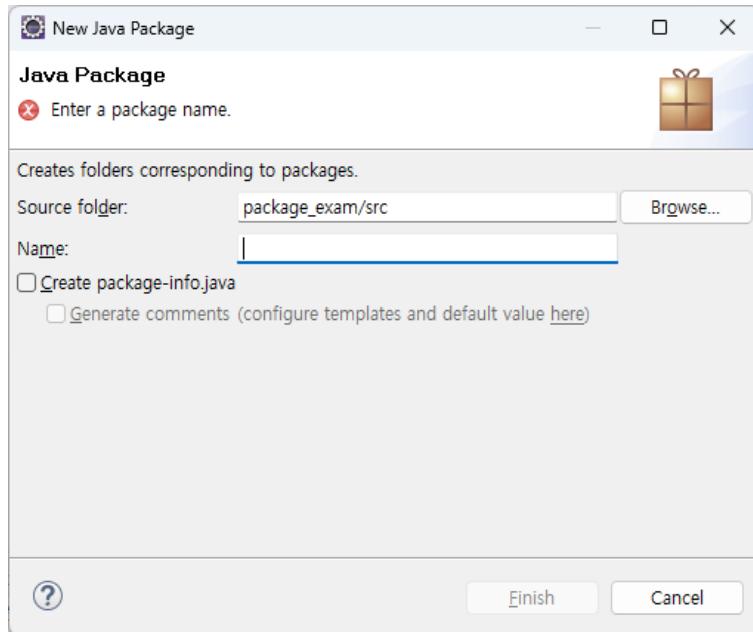
# 10. 패키지 / Import

- 패키지 만들기
  - src > 오른쪽 클릭 > New > Package



# 10. 패키지 / Import

- 패키지 만들기
  - Name 값은 모두 삭제
  - Create package-info.java 체크 해제



# 10. 패키지 / Import

---

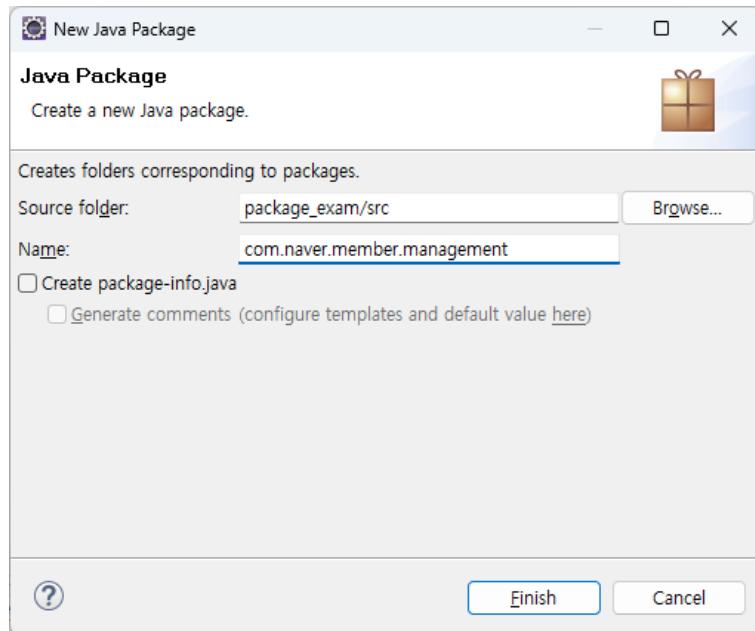
- 패키지 이름 정하기
  - 패키지 명은 애플리케이션을 개발하는 회사의 도메인을 역순으로 작성합니다.
  - 만약 naver.com 에서 package\_exam 애플리케이션을 개발한다면 com.naver 가 기본 패키지명이 됩니다.
  - 기본 패키지명이 작성된 후에는 업무명을 . 을 붙여 작성합니다.
  - 만약, 업무명이 “회원관리” 라고 한다면 업무명은 “member.management” 혹은 “management.member” 라고 작성할 수 있습니다.
  - 이렇게 완성된 패키지 명은 아래와 같습니다.
    - com.naver.member.management

## 패키지명 짓는 규칙

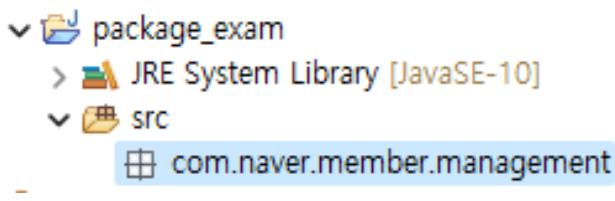
패키지명을 지을 때, 소문자와 점(“.”)을 제외한 다른 문자는 쓰지 않습니다. 모두 소문자로만 작성해야 하며 단어와 단어는 되도록 붙여 씁니다.

# 10. 패키지 / Import

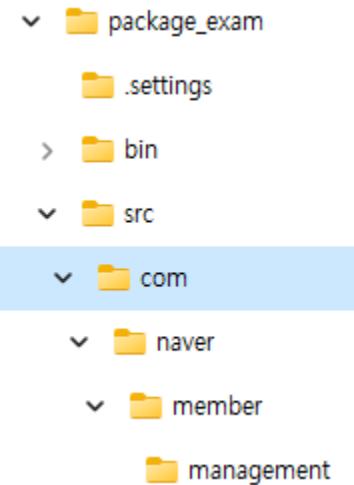
- 패키지명 입력하기
  - 완성된 패키지명을 입력하고 “Finish” 버튼을 클릭합니다.



# 10. 패키지 / Import

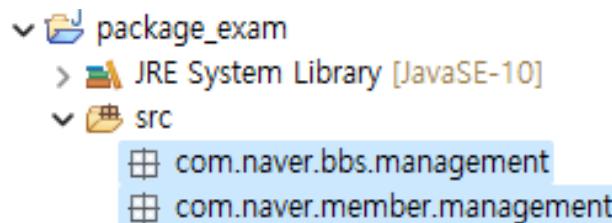
- 패키지 확인하기
  - Finish를 클릭해 패키지를 완성하면 아래 그림처럼 패키지가 만들어 집니다.

```
package_exam
 JRE System Library [JavaSE-10]
 src
 com.naver.member.management
```
  - 탐색기를 열어 package\_exam 프로젝트 폴더로 이동해보면 점(".")을 기준으로 실제 폴더가 만들어진 것을 확인 할 수 있습니다.

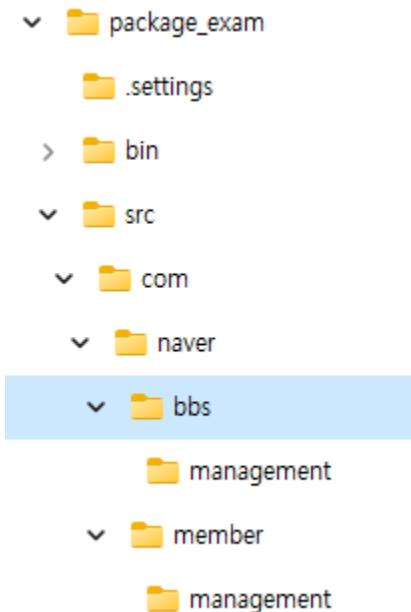


# 10. 패키지 / Import

- 다른 패키지 만들어보기
  - com.naver.bbs.management 패키지도 만들어봅니다.

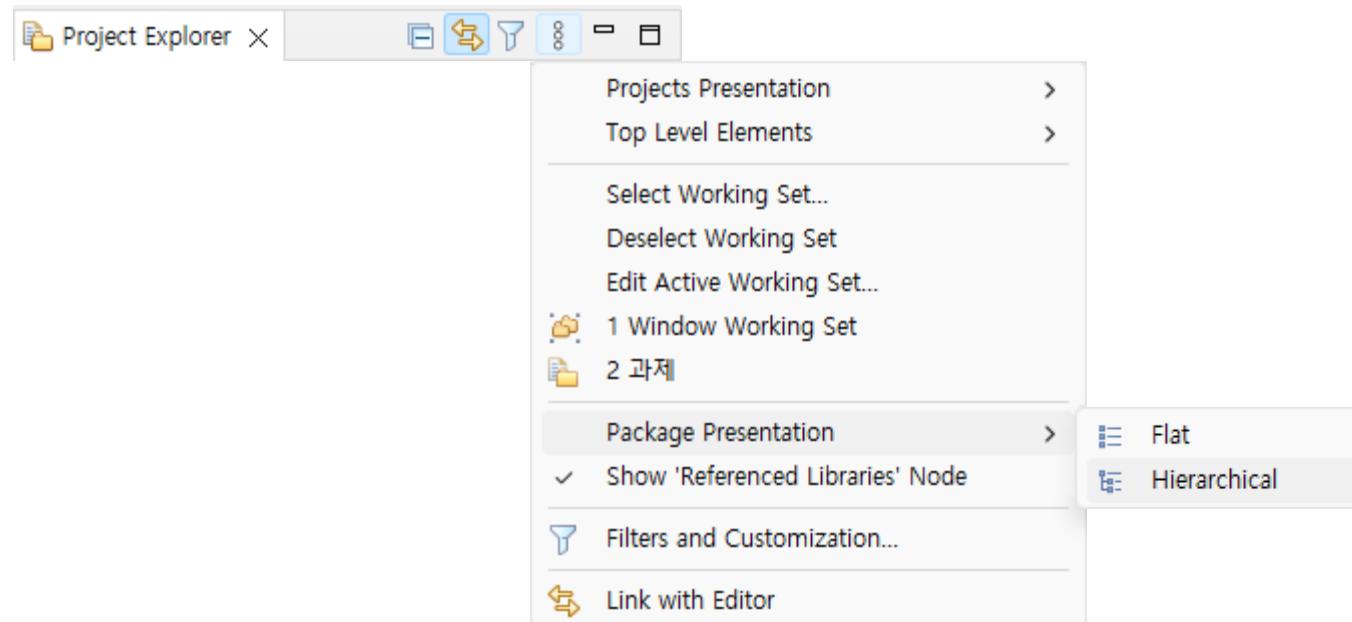


- 파일 탐색기에서도 패키지 폴더를 확인해 봅니다.



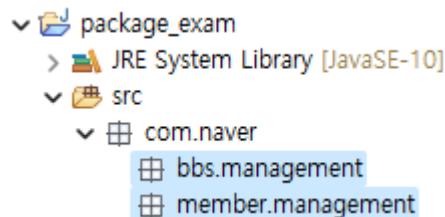
# 10. 패키지 / Import

- 패키지 구조를 파일 탐색기처럼 “트리” 형태로 보여주기
  - 기업형 애플리케이션을 만들다보면 패키지가 보통 적게는 200 ~ 300개 정도가 만들어집니다.
  - 많은 패키지를 목록형으로 보게되면 찾아보기도 힘들며, 관리하기도 쉽지 않게 되므로, 관리하기 편한 “트리” 형태로 바꾸어 봅니다.
  - Project Explorer에서  버튼을 클릭합니다.



# 10. 패키지 / Import

- 이제 패키지 구조가 “트리” 형태로 표시됩니다.



# 10. 패키지 / Import

---

- 패키지에 java 파일 작성해보기

```
package com.naver.member.management;

public class Member {

 String id;
 String name;

 public Member(String id, String name) {
 this.id = id;
 this.name = name;
 }

}
```

# 10. 패키지 / Import

---

- 패키지에 java 파일 작성해보기

```
package com.naver.bbs.management;

public class Article {

 public static void main(String[] args) {
 // Member cannot be resolved to a type
 Member member = new Member("ID", "관리자");
 System.out.println(member);
 System.out.println(member.id);
 System.out.println(member.name);

 }

}
```

# 10. 패키지 / Import

---

- Member 클래스에 에러가 발생하는 이유.
  - 다른 패키지에 있는 클래스를 사용하려면 어떤 패키지에 있는 클래스인지를 반드시 명시해야만 합니다.

# 10. 패키지 / Import

- Member 에 키보드 커서를 두고
  - "Ctrl + 1"을 누르면 "Quick Fix" 창이 나타납니다.
  - Import 'Member' (com.naver.member.management) 를 클릭하거나 선택 후 엔터를 입력합니다.

```
package com.naver.bbs.management;
```

```
public class Article {
```

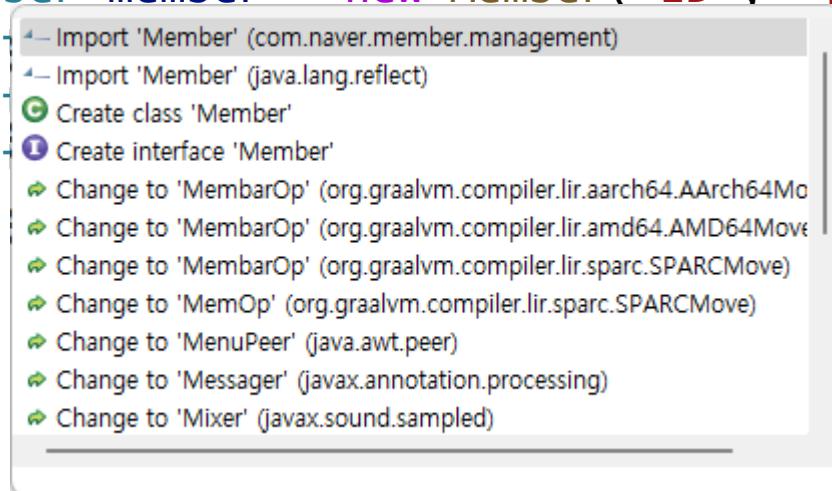
```
 public static void main(String[] args) {
 // Member cannot be resolved to a type
 Member member = new Member("ID", "관리자");
```

```
 System.out.println(member.getName());
```

```
 }
```

```
}
```

```
}
```



# 10. 패키지 / Import

---

- Member 에 키보드 커서를 두고
  - import 문장이 추가되면 에러가 사라집니다.

```
package com.naver.bbs.management;

import com.naver.member.management.Member;

public class Article {
 public static void main(String[] args) {

 Member member = new Member("ID", "관리자");
 System.out.println(member);
 // The field Member.id is not visible
 System.out.println(member.id);
 // The field Member.name is not visible
 System.out.println(member.name);
 }
}
```

# 10. 접근제어지시자

# 10. 접근제어지시자

---

- 패키지와는 별개의 문제로 클래스, 멤버변수, 메소드 등은 다른 클래스나 인스턴스가 접근을 마음대로 할 수 없도록 제한하는 기능이 있는데, 이것을 접근제어지시자 라고 합니다.
- Article.java 파일에서 아래 에러가 발생한 이유는 잘못된 접근제어지시자를 사용하고 있기 때문입니다.
  - The field Member.id is not visible
  - The field Member.name is not visible

# 10. 접근제어지시자

- 접근제어지시자(Access modifier)
  - 클래스, 멤버변수, 메소드의 외부로부터의 접근을 제한하는 키워드입니다.
  - 지시자는 제공되는 범위에 따라 총 4개로 구분됩니다.
    - private
    - default (접근제어 지시자가 누락된 경우)
    - protected
    - public

| 지시자       | 클래스 내부 | 동일 패키지 | 상속받은 클래스 | 이외의 영역 |
|-----------|--------|--------|----------|--------|
| private   | ●      | X      | X        | X      |
| default   | ●      | ●      | X        | X      |
| protected | ●      | ●      | ●        | X      |
| public    | ●      | ●      | ●        | ●      |

# 10. 접근제어지시자

- Article 클래스에서 사용하는 Member 클래스의 내용을 보면 멤버변수에 접근제어지시자가 없어 default 접근제어지시자가 사용되는 것을 알 수 있습니다.

```
package com.naver.member.management;

// public 접근제어지시자 적용
public class Member {

 String id; // 접근제어지시자가 없으므로 default 적용.
 String name; // 접근제어지시자가 없으므로 default 적용.

 // public 접근제어지시자 적용
 public Member(String id, String name) {
 this.id = id;
 this.name = name;
 }
}
```

| 지시자       | 클래스 내부 | 동일 패키지 | 상속받은 클래스 | 이외의 영역 |
|-----------|--------|--------|----------|--------|
| private   | ●      | X      | X        | X      |
| default   | ●      | ●      | X        | X      |
| protected | ●      | ●      | ●        | X      |
| public    | ●      | ●      | ●        | ●      |

# 10. 접근제어지시자

- 멤버변수에 public 접근제어지시자를 적용합니다.

```
package com.naver.member.management;

// public 접근제어지시자 적용
public class Member {

 public String id;
 public String name;

 // public 접근제어지시자 적용
 public Member(String id, String name) {
 this.id = id;
 this.name = name;
 }
}
```

| 지시자       | 클래스 내부 | 동일 패키지 | 상속받은 클래스 | 이외의 영역 |
|-----------|--------|--------|----------|--------|
| private   | ●      | X      | X        | X      |
| default   | ●      | ●      | X        | X      |
| protected | ●      | ●      | ●        | X      |
| public    | ●      | ●      | ●        | ●      |

# 10. 접근제어지시자

---

- Member 클래스를 사용하는 Article 클래스의 에러가 사라집니다.

```
package com.naver.bbs.management;

import com.naver.member.management.Member;

public class Article {
 public static void main(String[] args) {
 Member member = new Member("ID", "관리자");
 System.out.println(member);
 System.out.println(member.id);
 System.out.println(member.name);
 }
}
```

# **10. 정보은닉과 캡슐화**

# 10. 정보은닉과 캡슐화

---

- 인스턴스의 정보를 표현하는 멤버변수는 데이터의 보호를 위해 외부로부터 접근이 제한되어야 합니다.
- 따라서 멤버변수의 접근제한자는 public이 아닌 private으로 사용해야 하며 생성자를 통해 멤버변수의 값을 설정해야 합니다.

```
public class Member {

 private String id;
 private String name;

 public Member(String id, String name) {
 this.id = id;
 this.name = name;
 }
}
```

# 10. 정보은닉과 캡슐화

---

- 멤버변수가 보호된 Member 클래스의 인스턴스 사용.

```
public class Article {

 public static void main(String[] args) {

 Member member = new Member("ID", "관리자");
 // The field Member.id is not visible
 member.id; // member인스턴스의 id에 접근할 수 없습니다.

 }

}
```

# 10. 정보은닉과 캡슐화

---

- Member 클래스의 인스턴스인 member는 멤버변수에 직접 접근할 수 없도록 되어있기 때문에 내부의 값이 무엇인지 확인이 불가능합니다.
- Article 클래스에서 member 인스턴스의 id 변수의 값을 알 수 없습니다
- 이런 경우, member 인스턴스의 멤버변수 값을 반환해주는 별도의 인스턴스 메소드가 필요한데, 이것을 Getter 라고 합니다.

# 10. 정보은닉과 캡슐화

- Member 클래스에 Getter 정의.

```
public class Member {
 private String id;
 private String name;

 public Member(String id, String name) {
 this.id = id;
 this.name = name;
 }

 public String getId() {
 return this.id;
 }

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

# 10. 정보은닉과 캡슐화

---

- Getter 를 이용한 멤버변수 값 반환.

```
public class Article {

 public static void main(String[] args) {

 Member member = new Member("ID", "관리자");
 String id = member.getId();
 String name = member.getName();
 System.out.println(id); //ID
 System.out.println(name); //관리자

 }

}
```

# 10. 정보은닉과 캡슐화

---

- 정보은닉
  - member 인스턴스의 멤버변수들은 private으로 제한.
  - 값 할당은 생성자에서만 가능.
  - 멤버변수의 값을 Getter로 반환하여 사용.
  - 이런 구조를 “정보은닉”이라고 합니다.
- 멤버변수는 모두 보호 대상.
  - 모든 클래스들의 멤버변수는 예외없이 private으로 보호해야 하며
  - 값의 할당은 “생성자”로만 해야 합니다.
  - 값을 가져와 활용하려 할 경우 반드시 Getter를 통해야만 합니다.

# 10. 정보은닉과 캡슐화

---

- 멤버변수의 값은 재할당이 불가능 할까요?
  - 속성은 직접 변경이 불가능해야 합니다.
    - 항상 메소드를 통해 변경이 되어야 합니다.
- 멤버변수 변경의 예
  - 과일 하나에 500원에 판매하는 과일 판매자가 있습니다.
  - 과일 판매자를 클래스로 구현한다면 아래처럼 표현이 가능합니다.
    - 멤버변수
      - 소지금
      - 과일개수
      - 과일가격(상수)
    - 메소드
      - 판매

# 10. 정보은닉과 캡슐화

---

- 10개의 과일을 가지고 온 판매자가 5개의 과일을 판매할 경우
  - 과일의 재고는 5개가 줄어야 하며
  - 소지금은  $500 \times 5$ 원이 증가해야 합니다.
- 다시 말해 “판매(5)” 메소드를 실행할 경우,
  - 과일 판매자는 과일 재고와 소지금이 변화하게 됩니다.
- 캡슐화
  - 여러가지 변화를 하나의 메소드 내에 숨겨 처리하는 것

# 10. 정보은닉과 캡슐화

---

- 캡슐화가 적용된 과일 판매자 \_ 1 / 2

```
public class FruitSeller {

 private final int FRUIT_PRICE = 500;
 private int money;
 private int fruitStock;

 public FruitSeller(int money, int fruitStock) {
 this.money = money;
 this.fruitStock = fruitStock;
 }

 public int getMoney() {
 return this.money;
 }
}
```

# 10. 정보은닉과 캡슐화

- 캡슐화가 적용된 과일 판매자 \_ 2 / 2

```
public int getFruitStock() {
 return this.fruitStock;
}

public void sell(int quantity) {
 if (this.fruitStock >= quantity) {
 this.fruitStock -= quantity;
 this.money += quantity * FRUIT_PRICE;
 }
}
}
```

# 10. 정보은닉과 캡슐화

---

- 캡슐화된 클래스의 활용

```
public class Mart {

 public static void main(String[] args) {
 FruitSeller fs = new FruitSeller(5000, 10);
 System.out.println(fs.getMoney()); // 5000
 System.out.println(fs.getFruitStock()); // 10

 fs.sell(5);

 System.out.println(fs.getMoney()); // 7500
 System.out.println(fs.getFruitStock()); // 5
 }
}
```

# 10. 정보은닉과 캡슐화

---

- 만약 캡슐화를 사용하지 않는다면?
  - Mart의 main 메소드에서 FruitSeller의 멤버변수를 직접 제어해야만 합니다.

# 10. 정보은닉과 캡슐화

- 캡슐화를 사용하지 않는 FruitSeller \_ 1 / 2

```
public class FruitSeller {

 private final int FRUIT_PRICE = 500;
 private int money;
 private int fruitStock;

 public FruitSeller(int money, int fruitStock) {
 this.money = money;
 this.fruitStock = fruitStock;
 }

 public int getPrice() {
 return this.FRUIT_PRICE;
 }

 public int getMoney() {
 return this.money;
 }
}
```

# 10. 정보은닉과 캡슐화

- 캡슐화를 사용하지 않는 FruitSeller \_ 2 / 2

```
public int getFruitStock() {
 return this.fruitStock;
}
```

```
public void setMoney(int money) {
 this.money = money;
}
```

```
public void setFruitStock(int stock) {
 this.fruitStock = stock;
}
```

```
}
```

# 10. 정보은닉과 캡슐화

- 캡슐화를 사용하지 않는 FruitSeller를 사용하는 Mart

```
public class Mart {

 public static void main(String[] args) {
 FruitSeller fs = new FruitSeller(5000, 10);
 int money = fs.getMoney();
 int stock = fs.getFruitStock();

 System.out.println(money); // 5000
 System.out.println(stock); // 10

 fs.setMoney(money + (5 * fs.getPrice()));
 fs.setFruitStock(stock - 5);

 System.out.println(fs.getMoney()); // 7500
 System.out.println(fs.getFruitStock()); // 5
 }
}
```

# 10. 정보은닉과 캡슐화

---

- 코드가 길고 복잡해지며 의도를 파악하기 쉽지 않게 됩니다.
  - 즉, 가독성이 낮아지며
  - 가독성이 낮아지면 기능 추가 및 수정이 쉽지 않게 됩니다.

# 10. 정보은닉과 캡슐화

---

- 정보은닉 + 캡슐화 연습문제
  - 만화카페 구현해보기.
    - 만화카페는 다양한 만화책을 구비해놓고 있습니다.
    - 만화책을 관리하기 위해 아래 정보를 관리하고 있습니다.
      - 만화책 이름
        - 예. 슬램덩크 1화, 슬램덩크 2화
      - 만화책 대여 상태
        - 예. true: 대여 중, false: 대여안함
      - 만화책 대여비
      - 소지금(만화카페)
    - 만화책을 대여하면, 만화책의 대여 상태가 “대여 중”으로 변환되며 만화책 대여비만큼 소지금이 증가합니다.

# 10. 정보은닉과 캡슐화

---

- 정보은닉 + 캡슐화 연습문제
  - 만화카페 구현해보기.
    - 만화카페 기능.
      - 1. 모든 만화책목록 출력
      - 2. 만화책 대여
        - 이미 대여중인 만화책은 대여할 수 없습니다.
        - 대여할 때, 만화책 대여비를 지급해야 합니다.
      - 3. 만화책 반납.

# 11. 클래스 변수 / 클래스 메소드

# 11. 클래스 변수

# 11. 클래스 변수

---

- 클래스 변수
- 인스턴스가 아닌 클래스로 접근할 수 있는 변수를 말합니다.

```
public class Car {

 // 클래스 변수
 public static int instanceCount = 0;
 private String name;

 public Car(String name) {
 Car.instanceCount += 1;
 this.name = name;
 }

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

# 11. 클래스 변수

---

- 클래스 변수의 참조

```
public class CarMain {

 public static void main(String[] args) {
 Car car1 = new Car("경차");
 System.out.println(Car.instanceCount);

 Car car2 = new Car("소형차");
 System.out.println(Car.instanceCount);

 Car car3 = new Car("중형차");
 System.out.println(Car.instanceCount);

 Car car4 = new Car("대형차");
 System.out.println(Car.instanceCount);
 }

}
```

# 11. 클래스 변수

---

- 클래스 변수를 생성할 때는 변수의 타입 앞에 static 키워드를 사용합니다.

```
// 클래스 변수
public static int instanceCount = 0;
```

- static으로 정의된 변수는 인스턴스가 아닌 클래스로 접근이 가능하게 됩니다.

```
// 인스턴스 생성
Car car1 = new Car("경차");
// 인스턴스 메소드 호출
System.out.println(car1.getName());
// 클래스 변수 참조
System.out.println(Car.instanceCount);
```

# 11. 클래스 변수

---

- 클래스 변수는 인스턴스로도 참조가 가능한데  
이것은 Java 코딩 표준에 맞지 않아 Warning이 발생하므로  
되도록 클래스로 참조하도록 합니다.

```
// 인스턴스로 클래스 변수 참조
// The static field Car.instanceCount should be accessed in a static way
System.out.println(car1.instanceCount);
```

# 11. 클래스 변수

---

- 클래스 변수는 “클래스로 변수에 접근이 가능하다” 라는 점에서 애플리케이션 내부의 모든 Java파일에서 공유되는 “**공용 변수**” 입니다.
- 모든 Java 파일이 공유하는 변수이므로, 클래스 변수의 값이 변경되었을 경우, 원하지 않는 결과값이 생성될 가능성이 매우 높아집니다.
- 따라서 클래스 변수는 값이 변경되지 않도록 하기 위해 static 뒤에 final 키워드를 붙여 “**공용 상수(클래스 상수)**”로 사용합니다.

# 11. 클래스 변수

---

- FruitSeller 클래스의 FRUIT\_PRICE 를 클래스 상수로 변경.

```
public class FruitSeller {

 public static final int FRUIT_PRICE = 500;
 private int money;
 private int fruitStock;

 ... 생략 ...

 public void sell(int quantity) {
 if (this.fruitStock >= quantity) {
 this.fruitStock -= quantity;
 this.money += quantity * FruitSeller.FRUIT_PRICE;
 }
 }
}
```

# 11. 클래스 변수

---

- Mart에서 FRUIT\_PRICE 참조

```
public class Mart {

 public static void main(String[] args) {
 FruitSeller fs = new FruitSeller(5000, 10);
 System.out.println("과일 가격: " + FruitSeller.FRUIT_PRICE);
 System.out.println(fs.getMoney()); // 5000
 System.out.println(fs.getFruitStock()); // 10

 fs.sell(5);

 System.out.println(fs.getMoney()); // 7500
 System.out.println(fs.getFruitStock()); // 5
 }
}
```

# 11. 클래스 변수

---

- Java에서 지원하는 기본 클래스 상수

```
System.out.println(Math.PI);
System.out.println(Byte.MAX_VALUE);
System.out.println(Byte.MIN_VALUE);
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Float.MAX_VALUE);
System.out.println(Float.MIN_VALUE);
System.out.println(Double.MAX_VALUE);
System.out.println(Double.MIN_VALUE);
...
...
```

```
public static final double PI = 3.141592653589793;
public static final byte MAX_VALUE = 127;
public static final byte MIN_VALUE = -128;
...
...
```

# 11. 클래스 변수

---

- 클래스 변수 실습문제.
  - 만화카페의 “대여비” 를 클래스 상수로 변경해보세요.

# **11. 클래스 메소드**

# 11. 클래스 메소드

- 클래스 변수와 마찬가지로 클래스 메소드는 인스턴스가 아닌 클래스로 접근할 수 있는 메소드를 말합니다.
- 클래스로 접근하기 때문에 멤버변수의 영향을 받을 수 없으며  
**오직 파라미터의 영향만 받습니다.**
  - 예> “안녕” 10번 반복한 값을 반환하는 클래스 메소드

```
public class StringUtils {
 public static String repeat(String str, int times) {
 return str.repeat(times);
 }
}

public class ClassMethodTest {
 public static void main(String[] args) {
 String result = StringUtils.repeat("안녕", 10);
 System.out.println(result);
 }
}
```

# 11. 클래스 메소드

- 클래스 메소드의 접근 범위 (교재 190번 슬라이드)
  - 클래스 메소드는 인스턴스로 접근할 수 없습니다.

JVM(Java Virtual Machine)의 메모리 구조

## — Method (Static) —

상수(final) 명과 타입,  
멤버변수 명과 타입, 생성자, 인스턴스 메소드,  
클래스 변수(static) 명과 타입, **클래스 메소드(static)**,  
클래스 또는 인터페이스의 여부, 클래스 이름(타입), 슈퍼클래스의 이름(타입)

접근 가능

접근 불가능

## Stack

파라미터, 지역변수, 반환 값,  
연산에 사용된 임시 변수 등

접근 가능

접근 불가능

## Heap

new 키워드로 생성된  
객체 또는 배열 등의 Reference Type 객체  
(Method 영역에 적재된 클래스만 생성이 가능)

# 11. 클래스 메소드

---

- 클래스 메소드는 인스턴스 범위로 접근할 수 없습니다.

```
public class StringUtils {
 private double version = 1.0;

 public double getVersion() {
 return this.version;
 }

 public static String repeat(String str, int times) {
 // Cannot make a static reference to the non-static field version
 System.out.println(version);
 // Cannot make a static reference to the non-static method
 // getVersion() from the type StringUtils
 System.out.println(getVersion());
 return str.repeat(times);
 }
}
```

# 11. 클래스 메소드

- public static void main(String[] args)에서 static의 의미?
  - JVM이 Java 파일을 실행시킬 수 있도록 클래스 메소드로 정의
- JVM이 Java 파일을 실행하는 과정은 매우 복잡하지만,  
아래와 같이 단순하게 표현해 볼 수 있습니다.



Mart.main()

```
public class Mart {

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

# 11. 클래스 메소드

---

- 클래스 메소드는 “클래스로 메소드 접근이 가능하다” 라는 점에서 애플리케이션 내부의 모든 Java파일에서 공유되는 “**공용 메소드**” 입니다.
- 인스턴스 멤버변수, 인스턴스 메소드와 관계없이 1회성 처리만 하는 일종의 유틸리티 메소드가 필요할 때  
클래스 메소드로 정의해 “**공용 메소드**”로 사용합니다.
  - 예> 문자열이 비어있는지 확인하는 클래스 메소드
  - 문자열의 길이가 5글자 이상인지 확인하는 클래스 메소드
  - 문자열을 숫자로 변경해주는 클래스 메소드
  - 숫자를 문자열로 변경해주는 클래스 메소드
  - 문자열이 이메일 형식인지 확인하는 클래스 메소드
  - 문자열이 전화번호 형식인지 확인하는 클래스 메소드
  - 숫자가 지정 범위 내에 있는지 확인하는 클래스 메소드 등등

# 11. 클래스 메소드

---

- 클래스 메소드 실습 문제
- isValidAge(int age, int min, int max) 클래스 메소드를 만들고 age의 값이 min보다 크고 max 보다 작으면 true를, 아니면 false를 반환하도록 작성하고 호출해보세요.
- getCourseCredit(double average) 클래스 메소드를 만들고 average의 값을 4.5 만점 기준의 학점을 반환하도록 작성하고 호출해보세요.
- getABCDE(double courseCredit) 클래스 메소드를 만들고 학점 기준에 맞춰 ABCDE 등급을 반환하도록 작성하고 호출해보세요.
- 교재 199번 슬라이드 학생 평균 및 학점 구하기 실습문제 중 getCourseCredit과 getABCDE 인스턴스 메소드를 클래스 메소드로 변경해보세요.

# 12. 메소드 오버로딩

# 12. 메소드 오버로딩

- Java에서 작성하는 변수나 메소드(클래스/인스턴스, 생성자)는 유일한 이름으로 구분해야 합니다.
- 변수 중복선언, 메소드 중복선언은 할 수 없습니다.
- 만약, 강제로 선언한다면 아래와 같은 에러를 보게 됩니다.  
Duplicate field 클래스명.변수명  
Duplicate method 메소드명() in type 클래스명

```
public class FruitSeller {
 private int money;
 // Duplicate field FruitSeller.money
 private int money;
 // Duplicate method getMoney() in type FruitSeller
 public int getMoney() {
 return 0;
 }
 public int getMoney() {
 return this.money;
 }
}
```

# 12. 메소드 오버로딩

---

- 단, 메소드(클래스/인스턴스, 생성자)는 **메소드 오버로딩** 기법으로 동일한 이름의 메소드를 정의하고 사용할 수 있습니다.
- 메소드 오버로딩의 기준은 아래와 같습니다.
  - 1. 동일한 메소드 명
  - 2. 동일한 반환타입
  - 3. 파라미터의 개수가 다르거나 파라미터의 타입이 다름.

# 12. 메소드 오버로딩

---

- 아래 메소드는 FruitSeller 클래스의 sell 메소드입니다.
- quantity의 개수만큼 재고를 감소시키고 금액을 증가시킵니다.

```
public void sell(int quantity) {
 if (this.fruitStock >= quantity) {
 this.fruitStock -= quantity;
 this.money += quantity * FruitSeller.FRUIT_PRICE;
 }
}
```

- 파라미터가 없는 sell() 메소드를 만들어 이 메소드를 호출 할 경우 재고는 1개가 감소되며 금액은 1개의 과일 값 만큼 증가시키는 기능을 만들 수 있습니다.

# 12. 메소드 오버로딩

- 오버로딩 된 sell 메소드

```
public class FruitSeller {
 ... 생략...

 public void sell(int quantity) {
 if (this.fruitStock >= quantity) {
 this.fruitStock -= quantity;
 this.money += quantity * FruitSeller.FRUIT_PRICE;
 }
 }

 public void sell() {
 if (this.fruitStock >= 1) {
 this.fruitStock--;
 this.money += FruitSeller.FRUIT_PRICE;
 }
 }
}
```

# 12. 메소드 오버로딩

- 오버로딩 된 sell 메소드를 사용하는 Mart

```
public class Mart {

 public static void main(String[] args) {
 FruitSeller fs = new FruitSeller(5000, 10);
 System.out.println("오늘 과일 가격: " + FruitSeller.FRUIT_PRICE);
 System.out.println(fs.getMoney()); // 5000
 System.out.println(fs.getFruitStock()); // 10

 fs.sell(5); // 파라미터가 있는 sell 메소드가 실행됨.
 System.out.println(fs.getMoney()); // 7500
 System.out.println(fs.getFruitStock()); // 5

 fs.sell(); // 파라미터가 없는 sell 메소드가 실행됨.
 System.out.println(fs.getMoney()); // 8000
 System.out.println(fs.getFruitStock()); // 4
 }
}
```

## 12. 생성자 오버로딩

# 12. 생성자 오버로딩

- 생성자도 메소드이기 때문에 오버로딩이 가능합니다.
- 클래스 내에 생성자를 정의할 경우, 기본 생성자는 자동으로 만들어지지 않기 때문에 FruitSeller는 파라미터가 없는 `new FruitSeller();` 생성자는 호출 할 수 없습니다.

```
public class FruitSeller {
 ... 생략 ...

 public FruitSeller(int money, int fruitStock) {
 this.money = money;
 this.fruitStock = fruitStock;
 }

 ... 생략 ...
}
```

# 12. 생성자 오버로딩

- FruitSeller의 기본생성자 호출이 필요할 경우, 생성자 오버로딩을 이용해 기본 생성자를 만들어 사용할 수 있습니다.

```
public class FruitSeller {
 ... 생략 ...

 public FruitSeller() {
 this.money = 10000;
 this.fruitStock = 4;
 }

 public FruitSeller(int money, int fruitStock) {
 this.money = money;
 this.fruitStock = fruitStock;
 }
 ... 생략 ...
}
```

# 12. 생성자 오버로딩

- 기본 생성자를 사용하는 Mart

```
public class Mart {

 public static void main(String[] args) {
 FruitSeller fs = new FruitSeller();
 System.out.println("오늘 과일 가격: " + FruitSeller.FRUIT_PRICE);
 System.out.println(fs.getMoney()); // 10000
 System.out.println(fs.getFruitStock()); // 4

 fs.sell(3); // 파라미터가 있는 sell 메소드가 실행됨.
 System.out.println(fs.getMoney()); // 11500
 System.out.println(fs.getFruitStock()); // 1

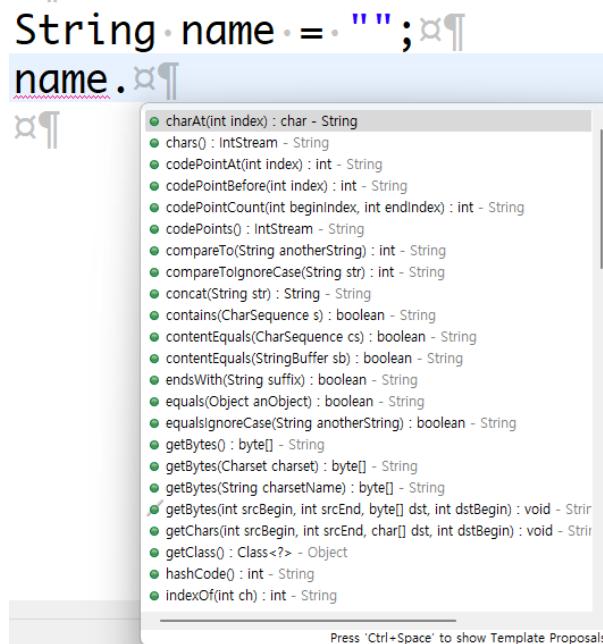
 fs.sell(); // 파라미터가 없는 sell 메소드가 실행됨.
 System.out.println(fs.getMoney()); // 12000
 System.out.println(fs.getFruitStock()); // 0
 }
}
```

<https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/String.html>

# 13. String

# 13. String

- 문자열을 할당할 수 있는 Java의 내장 클래스.
- 문자열을 간편히 제어할 수 있도록 많은 인스턴스 메소드를 제공하고 있습니다.



`String`은 생성자 없이 데이터를 할당 할 수 있습니다.

String 인스턴스에 값을 할당할 때, 생성자를 사용하지 않습니다.

Java에서 아주 많이 사용되므로, 간편히 사용할 수 있도록 JVM이 지원하고 있습니다.

# 13. String

- 자주 사용하는 String methods

| 메소드명             | 메소드 유형   | 반환타입     | 파라미터1                        | 파라미터2                 | 설명                           |
|------------------|----------|----------|------------------------------|-----------------------|------------------------------|
| contains         | Instance | boolean  | String - 찾을 문자열              | -                     | 내용이 포함되어있는지 확인               |
| endsWith         | Instance | boolean  | String - 비교할 문자열             | -                     | 파라미터로 끝나는지 확인                |
| equals           | Instance | boolean  | String - 비교할 문자열             | -                     | 내용이 정확히 동일한지 확인 (대소문자 구분 함)  |
| equalsIgnoreCase | Instance | boolean  | String - 비교할 문자열             | -                     | 내용이 정확히 동일한지 확인 (대소문자 구분 안함) |
| format           | Class    | String   | Object 바인딩될 데이터              | -                     | 포맷데이터에 문자열 바인딩               |
| formatted        | Instance | String   | Object 바인딩될 데이터              | -                     | 포맷데이터에 문자열 바인딩               |
| indexOf          | Instance | int      | String - 찾을 문자열              | -                     | 찾을 문자열의 첫 번째 위치 찾기           |
| isBlank          | Instance | boolean  | -                            | -                     | 문자열이 공백인지 확인 (스페이스 포함)       |
| isEmpty          | Instance | boolean  | -                            | -                     | 문자열이 공백인지 확인 (스페이스 미포함)      |
| join             | Class    | String   | String - 연결자                 | String - 연결할 문자열...   | 문자열을 연결자로 연결함                |
| lastIndexOf      | Instance | int      | String - 찾을 문자열              | -                     | 찾을 문자열의 마지막 위치 찾기            |
| length           | Instance | int      | -                            | -                     | 문자열의 길이 구하기                  |
| matches          | Instance | boolean  | String - 비교할 정규표현식           | -                     | 문자열의 형태 비교하기                 |
| replace          | Instance | String   | String - 찾을 문자열              | String - 새로운 문자열      | 문자열 찾아 바꾸기                   |
| replaceAll       | Instance | String   | String - 찾을 정규표현식            | String - 새로운 문자열      | 문자열 찾아 바꾸기 (정규표현식 지원)        |
| split            | Instance | String[] | String - 자르는 기준 문자열 혹은 정규표현식 |                       | 문자열을 파라미터로 잘라 배열로 반환         |
| startsWith       | Instance | boolean  | String - 비교할 문자열             | -                     | 파라미터로 시작하는지 확인               |
| subString        | Instance | String   | int - 잘라올 시작 인덱스             | int - 잘라올 마지막 인덱스 + 1 | 문자열을 잘라냄.                    |

# 13. String

---

- 자주 사용하는 String methods
- address에 "서울"이 포함되어 있는지 확인

```
String address = "서울특별시 서초구 효령로 176";
boolean isSeoul = address.contains("서울");
System.out.println(isSeoul);
```

- address가 176으로 끝나는지 확인

```
String address = "서울특별시 서초구 효령로 176";
boolean isSeoul = address.endsWith("176");
System.out.println(isSeoul);
```

- name이 ktdsUniversity와 정확히 같은지 비교

```
String name = "ktdsuniversity";
boolean isEqual = name.equals("ktdsUniversity");
System.out.println(isEqual);
```

# 13. String

---

- 자주 사용하는 String methods
- name이 ktdsUniversity와 같은지 비교(대소문자 비교안함)

```
String name = "ktdsuniversity";
boolean isEqual = name.equalsIgnoreCase("ktdsUniversity");
System.out.println(isEqual);
```

- format 바인딩 기능

```
String format = "%s에서 교육하는 %s과정";
String str = String.format(format, "ktdsuniversity", "Java");
System.out.println(str);
```

- format 바인딩 기능 (Java 15부터 가능)

```
String format = "%s에서 교육하는 %s과정";
// Since Java15
String str = format.formatted("ktdsuniversity", "Java");
System.out.println(str);
```

# 13. String

---

- 자주 사용하는 String methods

- 문자 c의 인덱스(위치) 찾기

```
String alphabets = "abcdefg";
int letterCIndex = alphabets.indexOf('c');
System.out.println(letterCIndex);
```

- 문자 C의 인덱스(위치) 찾기

```
String alphabets = "abcdefg";
int letterCIndex = alphabets.indexOf('C');
System.out.println(letterCIndex);
```

- 문자 def의 인덱스(위치) 찾기

```
String alphabets = "abcdefg";
int letterDEFIndex = alphabets.indexOf("def");
System.out.println(letterDEFIndex);
```

# 13. String

---

- 자주 사용하는 String methods
- str이 비어있거나 공백으로만 이루어져있는지 확인 (Java 11부터)

```
String str = " ";
// Since Java11
boolean isBlank = str.isBlank();
System.out.println(isBlank);
```

- str이 공백으로 비워져있는지 확인

```
String str = " ";
boolean isEmpty = str.isEmpty();
System.out.println(isEmpty);
```

- message와 name을 ", " 으로 연결하기

```
String message = "안녕하세요";
String name = "홍길동님";
String helloMessage = String.join(", ", message, name);
System.out.println(helloMessage);
```

# 13. String

---

- 자주 사용하는 String methods
- message에서 "a"의 마지막 인덱스(위치) 찾기

```
String message = "abcdefghijklm";
int letterALastIndex = message.lastIndexOf("a");
System.out.println(letterALastIndex);
```

- message에서 "jj"의 마지막 인덱스(위치) 찾기

```
String message = "abcdefghijklm";
int letterJJLastIndex = message.lastIndexOf("jj");
System.out.println(letterJJLastIndex);
```

- message의 문자열 길이 구하기

```
String message = "abcdefghijklm";
int length = message.length();
System.out.println(length);
```

# 13. String

---

- 자주 사용하는 String methods

- phone이 숫자인지 확인하기

```
String phone = "01012341234";
boolean isNumber = phone.matches("^[0-9]+$");
System.out.println(isNumber);
```

- message에서 홍길동을 ktds로 변경하기

```
String message = "안녕하세요, 홍길동님, 안녕히 가세요, 홍길동님.";
message = message.replace("홍길동", "ktds");
System.out.println(message);
```

- phone에서 숫자가 아닌 문자를 공백문자("")로 변경하기

```
String phone = "010-1234-1234";
phone = phone.replaceAll("[^0-9]", "");
System.out.println(phone);
```

# 13. String

---

- 자주 사용하는 String methods
- phone을 “-”로 잘라 배열에 할당하기

```
String phone = "010-1234-1234";
String[] phoneArea = phone.split("-");
System.out.println(phoneArea[0]);
System.out.println(phoneArea[1]);
System.out.println(phoneArea[2]);
```

- phone이 +82로 시작하는지 확인하기

```
String phone = "+82-010-1234-1234";
boolean isKoreaNum = phone.startsWith("+82");
System.out.println(isKoreaNum);
```

# 13. String

---

- 자주 사용하는 String methods
- datetime에서 2023(연) 글자만 잘라내어 할당하기

```
String datetime = "2023-05-02 14:56:13";
String year = datetime.substring(0, 4);
System.out.println(year);
```

- datetime에서 14(시) 글자만 잘라내어 할당하기

```
String datetime = "2023-05-02 14:56:13";
String hour = datetime.substring(11, 13);
System.out.println(hour);
```

- datetime에서 14:56:13(시:분:초) 글자만 잘라내어 할당하기

```
String datetime = "2023-05-02 14:56:13";
String time = datetime.substring(11);
System.out.println(time);
```

# 13. String

---

- 자주 사용하는 String methods
- datetime에서 앞뒤 공백모두 제거하기

```
String datetime = " 2023-05-02 14:56:13 ";
System.out.println(datetime.length());
System.out.println(datetime);
datetime = datetime.trim();
System.out.println(datetime.length());
System.out.println(datetime);
```

- int 타입 1을 문자열로 변경하기 (Overloading)

```
String iStr = String.valueOf(1);
System.out.println(iStr);
```

# 13. String

---

- String 실습 문제
- Scanner와 while(무한반복)을 이용해 끝말잇기 게임 만들기
- 게임은 “자전거” 단어로 시작합니다.
- 자전거가 출력된 이후부터 Scanner를 이용해 마지막으로 입력된 단어의 마지막 글자로 시작하는 3글자 이상의 단어를 입력해야 합니다.  
(좌우 공백은 모두 제거해야 합니다)
- 만약, 마지막 글자로 시작하지 않거나 3글자 미만의 글자를 입력했을 경우  
게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수를 출력합니다.

# 13. String

---

- String 실습 문제
- Scanner와 while(무한반복)을 이용해 같은 첫 글자로 시작하는 단어 게임 만들기
- 게임은 “나”로 시작합니다.
- “나”가 출력된 이후부터 “나”로 시작하는 두 글자 이상의 단어를 입력해야 합니다. (좌우 공백은 모두 제거해야 합니다)
- 만약 “나”로 시작하지 않거나 한 글자 미만의 글자를 입력했을 경우, 게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수를 출력합니다.

# 13. String

---

- String 실습 문제
- Scanner와 while(무한반복)을 이용해 같은 마지막 글자로 끝나는 단어 게임 만들기
- 게임은 “기”로 시작합니다.
- “기”가 출력된 이후부터 “기”로 끝나는 두 글자 이상의 단어를 입력해야 합니다.  
(좌우 공백은 모두 제거해야 합니다)
- 만약 “기”로 끝나지 않거나 한 글자 미만의 글자를 입력했을 경우,  
게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수를 출력합니다.

# 13. String

---

- String 실습 문제
- Scanner와 while(무한반복)을 이용해 "소"가 포함된 단어 게임 만들기
- 게임은 "소"로 시작합니다.
- "소"가 출력된 이후부터 "소"가 포함된 두 글자 이상의 단어를 입력해야 합니다.  
(좌우 공백은 모두 제거해야 합니다)
- 만약 "소"가 포함되지 않거나 한 글자 미만의 글자를 입력했을 경우,  
게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수를 출력합니다.

# 13. String

---

- String 실습 문제
- Scanner와 while(무한반복)을 이용해 입력한 글자의 두번째 글자로 시작하는 단어 게임 만들기
- 게임은 “알파센터우리”로 시작합니다.
- “알파센터우리”가 출력된 이후부터 마지막으로 입력된 단어의 두 번째 글자로 시작하는 세 글자 이상의 단어를 입력해야 합니다.  
(좌우 공백과 글자 사이의 공백은 모두 제거해야 합니다.)
- 만약 마지막으로 입력한 단어의 두 번째 글자로 시작하지 않거나 세 글자 미만의 글자를 입력했을 경우, 게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수를 출력합니다.

# 13. String의 패키지와 import

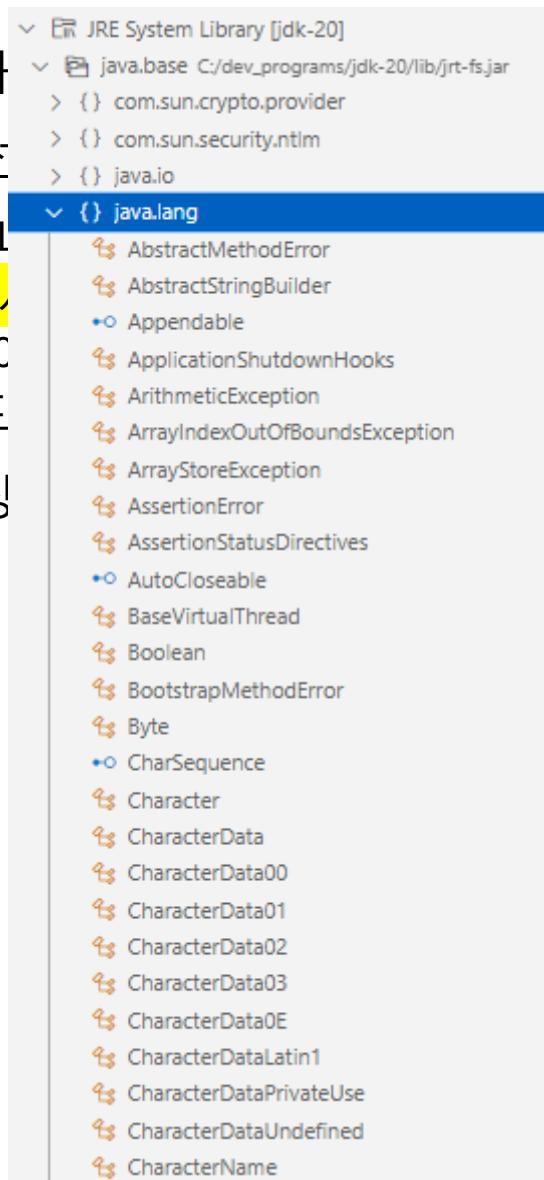
# 13. String의 패키지와 Import

---

- String 은 Java의 내장 클래스 입니다.
- java.lang 패키지 내부에 선언되어 있습니다.
- 10장. 패키지에서 배웠던 내용 중  
“패키지가 다를 경우 반드시 import 문을 작성해야 한다” 라고 했는데, 전혀 다른 패키지에 있는 String은 Import를 작성하지 않아도 사용이 가능했습니다.
- Java는 자주 사용하는 내장 클래스들을 java.lang 패키지에 모두 넣어 배포하는데, 이 패키지에 있는 클래스들은 import를 하지 않아도 사용이 가능합니다.

# 13. String의 패키지와 Import

- String 은 Java의 내장 클래스
- java.lang 패키지 내부에 선언
- 10장. 패키지에서 배웠던 “**패키지가 다를 경우 반드시**” 했는데, 전혀 다른 패키지에서 Import를 작성하지 않아도
- Java는 자주 사용하는 내장 클래스, 이 패키지에 있는 클래스



한다”라고

기지에 모두 넣어 배포하는  
방법도 사용이 가능합니다.

# 13. String Concatenation

# 13. String Concatenation

---

- 문자열과 문자열을 연결하려면 “+” 연산자를 사용합니다.

```
public static void main(String[] args) {
 String name = "홍길동";
 String greeting = "안녕하세요. ";
 String message = name + "님, " + greeting;
 System.out.println(message);
}
```

- 문자열 연결의 특징.
  - 문자열 + 모든 타입의 결과는 문자열이 됩니다.

```
public static void main(String[] args) {
 String message = 1 + "";
 System.out.println(message); // 1
 System.out.println(message.getClass()); // class java.lang.String
}
```

# 13. String Concatenation

---

- 대량 문자열 연결하기 (Java 15 버전 미만)

```
// Java 15 미만의 문자열 연결
String str = "Lorem Ipsum is simply dummy text"
 + " of the printing and typesetting industry."
 + " Lorem Ipsum has been the industry's "
 + "standard dummy text ever since the 1500s,"
 + " when an unknown printer took a galley of type"
 + " and scrambled it to make a type specimen book."
 + " It has survived not only five centuries,"
 + " but also the leap into electronic typesetting,"
 + " remaining essentially unchanged. "
 + "It was popularised in the 1960s with the release"
 + " of Letraset sheets containing Lorem Ipsum passages,"
 + " and more recently with desktop publishing software"
 + " like Aldus PageMaker including versions of Lorem Ipsum";
```

# 13. String Concatenation

---

- 대량 문자열 연결하기 (Java 15 버전 이상)

```
// Java 15 이상의 멀티라인 문자열
String str = """
Lorem Ipsum is simply dummy text
of the printing and typesetting industry.
Lorem Ipsum has been the industry's
standard dummy text ever since the 1500s,
when an unknown printer took a galley of type
and scrambled it to make a type specimen book.
It has survived not only five centuries,
but also the leap into electronic typesetting,
remaining essentially unchanged.
It was popularised in the 1960s with the release
of Letraset sheets containing Lorem Ipsum passages,
and more recently with desktop publishing software
like Aldus PageMaker including versions of Lorem Ipsum""";
```

# 13. String Concatenation

---

- 문자열은 Reference Type 이므로 항상 메모리를 참조합니다.
- 아래 코드에서 ?1 과 ?2 의 결과는 무엇일지 예상해보고 코드로 테스트를 해봅니다.

```
public class StringExam {

 public static void changeStr(String str) {
 str += "바뀌었을까요?";
 System.out.println(str); // ?1
 }

 public static void main(String[] args) {
 String message = "원본입니다.";
 changeStr(message);
 System.out.println(message); // ?2
 }
}
```

# 13. String Concatenation

---

- Reference Type은 두 가지 타입으로 구분됩니다.
  - **Immutable**: 메모리내의 값이 절대 변경될 수 없습니다.
  - **Mutable**: 메모리 내의 값이 자유롭게 변경될 수 있습니다.
- String 타입은 대표적인 Immutable 타입입니다.
  - String 의 값을 변경하려 시도할 경우, 새로운 메모리 공간을 할당하고 변경된 값을 할당합니다.
  - 즉, String이 참조하고있던 메모리 주소는 끊어지며 새로운 메모리 공간을 참조하게 됩니다.

```
public static void changeStr(String str) {
 str += "바뀌었을까요?";
 System.out.println(str);
}
```

# 13. String Concatenation

---

- 또한, 문자열 연결은 문자열 내의 값이 많을 경우 새로운 메모리 공간을 확보하고 값을 할당하는데 큰 비용(CPU, Memory)이 들기 때문에 추천하지 않는 방법입니다.
  - 작은 량의 문자열은 +로 연결하는 것이 효율적입니다.
- 즉, 많은 내용의 문자열을 연결할 필요가 있을 때 StringBuffer를 사용하는 것이 비용절약에 효율적인 방법입니다.

# **13. StringBuffer**

# 13. StringBuffer

- 문자열 이어 붙이기 위한 StringBuffer
  - StringBuffer 인스턴스의 append() 메소드를 이용해 문자열을 이어 붙입니다.
  - 마지막으로 StringBuffer의 toString() 메소드를 이용해 하나의 문자열 인스턴스로 변환합니다.

```
public static void main(String[] args) {
 StringBuffer sb = new StringBuffer();
 sb.append("I AM\n");
 sb.append("UNFORGIVEN\n");
 sb.append("Kitsch\n");
 sb.append("꽃\n");
 sb.append("손오공\n");
 sb.append("나의 바람(Wind And Wish)\n");
 sb.append("물론\n");
 sb.append("파이팅 해야지 (Feat. 이영지)");
 String str = sb.toString();
 System.out.println(str);
}
```

\n은 줄바꿈을 하는 Escape Character 입니다.  
이 외에도 \t, \", \', \\ 등이 있습니다.

# 13. StringBuffer

---

- StringBuffer를 사용하면 조건이나 반복에 의한 문자열 생성도 간단히 할 수 있습니다.

```
public static void main(String[] args) {
 boolean isSoloSinger = false;
 StringBuffer sb = new StringBuffer();
 if (!isSoloSinger) {
 sb.append("I AM\n");
 sb.append("UNFORGIVEN\n");
 sb.append("Kitsch\n");
 sb.append("손오공\n");
 sb.append("나의 바람(Wind And Wish)\n");
 sb.append("파이팅 해야지 (Feat. 이영자)")
 }
 else {
 sb.append("꽃\n");
 sb.append("물론\n");
 }

 String str = sb.toString();
 System.out.println(str);
}
```

# 13. StringBuffer

- 메모리 참조를 이용한 문자열 연결도 가능합니다.

```
public static void more(StringBuffer sb) {
 sb.append("Ditto\n");
 sb.append("CHRISTIAN\n");
}

public static void main(String[] args) {
 StringBuffer sb = new StringBuffer();
 sb.append("I AM\n");
 sb.append("UNFORGIVEN\n");
 sb.append("Kitsch\n");
 sb.append("손오공\n");
 sb.append("나의 바람(Wind And Wish)\n");
 sb.append("파이팅 해야지 (Feat. 이영자)\n");
 sb.append("꽃\n");
 sb.append("물론\n");
 more(sb);
 String str = sb.toString();
 System.out.println(str);
}
```

# 13. StringBuffer

---

- StringBuffer 실습 문제
- Scanner와 while(무한반복)을 이용해 끝말잇기 게임 만들기
- 게임은 “자전거” 단어로 시작합니다.
- 자전거가 출력된 이후부터 Scanner를 이용해 마지막으로 입력된 단어의 마지막 글자로 시작하는 3글자 이상의 단어를 입력해야 합니다. (좌우 공백은 모두 제거 해야 합니다)
- 만약, 마지막 글자로 시작하지 않거나 3글자 미만의 글자를 입력했을 경우 게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수와 단어들을 출력합니다.

# 13. StringBuffer

---

- StringBuffer 실습 문제
- Scanner와 while(무한반복)을 이용해 같은 첫 글자로 시작하는 단어 게임 만들기
- 게임은 “나”로 시작합니다.
- “나”가 출력된 이후부터 “나”로 시작하는 두 글자 이상의 단어를 입력해야 합니다. (좌우 공백은 모두 제거해야 합니다)
- 만약 “나”로 시작하지 않거나 한 글자 미만의 글자를 입력했을 경우, 게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수와 단어들을 출력합니다.

# 13. StringBuffer

---

- StringBuffer 실습 문제
- Scanner와 while(무한반복)을 이용해 같은 마지막 글자로 끝나는 단어 게임 만들기
- 게임은 "기"로 시작합니다.
- "기"가 출력된 이후부터 "기"로 끝나는 두 글자 이상의 단어를 입력해야 합니다.  
(좌우 공백은 모두 제거해야 합니다)
- 만약 "기"로 끝나지 않거나 한 글자 미만의 글자를 입력했을 경우, 게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수와 단어들을 출력합니다.

# 13. StringBuffer

---

- StringBuffer 실습 문제
- Scanner와 while(무한반복)을 이용해 "소"가 포함된 단어 게임 만들기
- 게임은 "소"로 시작합니다.
- "소"가 출력된 이후부터 "소"가 포함된 두 글자 이상의 단어를 입력해야 합니다.  
(좌우 공백은 모두 제거해야 합니다)
- 만약 "소"가 포함되지 않거나 한 글자 미만의 글자를 입력했을 경우, 게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수와 단어들을 출력합니다.

# 13. StringBuffer

---

- StringBuffer 실습 문제
- Scanner와 while(무한반복)을 이용해 입력한 글자의 두번째 글자로 시작하는 단어 게임 만들기
- 게임은 “알파센터우리”로 시작합니다.
- “알파센터우리”가 출력된 이후부터 마지막으로 입력된 단어의 두 번째 글자로 시작하는 세 글자 이상의 단어를 입력해야 합니다. (좌우 공백과 글자 사이의 공백은 모두 제거해야 합니다.)
- 만약 마지막으로 입력한 단어의 두 번째 글자로 시작하지 않거나 세 글자 미만의 글자를 입력했을 경우, 게임은 종료됩니다.
- 게임이 종료될 때, 이어나간 단어의 수와 단어들을 출력합니다.

# **14. 배열**

# 14. 배열

---

- 배열이란, 동일한 타입의 값들을 메모리에 차례대로 나열시킨 구조를 말합니다.
- 만약, 학생 5명의 점수를 각 변수에 할당시키고 합계를 구하는 코드를 배열을 사용하지 않고 작성한다면 아래와 같은 코드가 완성됩니다.

```
public static void main(String[] args) {
 int aScore = 50;
 int bScore = 70;
 int cScore = 65;
 int dScore = 95;
 int eScore = 55;

 // 일일이 더해야 합니다.
 int sum = aScore + bScore + cScore + dScore + eScore;
 System.out.println(sum);
}
```

# 14. 배열

---

- 위 코드에서 학생들의 점수를 5점씩 더해야 한다면 값을 일일이 수정해야 합니다.

```
public static void main(String[] args) {
 int aScore = 55;
 int bScore = 75;
 int cScore = 70;
 int dScore = 100;
 int eScore = 60;

 // 일일이 더해야 합니다.
 int sum = aScore + bScore + cScore + dScore + eScore;
 System.out.println(sum);
}
```

# 14. 배열

---

- 이제 평균을 구해봅니다.

```
public static void main(String[] args) {
 int aScore = 55;
 int bScore = 75;
 int cScore = 70;
 int dScore = 100;
 int eScore = 60;

 int sum = aScore + bScore + cScore + dScore + eScore;
 System.out.println(sum);

 double average = sum / 5.0;
 System.out.println(average);
}
```

# 14. 배열

- 계산을 해보니 학생 두 명을 누락했다는 것을 알게 되었습니다.
- 두 명의 학생을 추가하고 코드를 변경합니다.

```
public static void main(String[] args) {
 int aScore = 55;
 int bScore = 75;
 int cScore = 70;
 int dScore = 100;
 int eScore = 60;
 int fScore = 75;
 int hScore = 45;

 int sum = aScore + bScore + cScore + dScore + eScore
 + fScore + hScore;
 System.out.println(sum);

 double average = sum / 7.0;
 System.out.println(average);
}
```

# 14. 배열

- 다시 채점해보니 5점을 더하면 안됐다는 것을 알게 되었습니다.
- 다시 모든 학생들의 점수에서 5점을 뺍니다.

```
public static void main(String[] args) {
 int aScore = 50;
 int bScore = 70;
 int cScore = 65;
 int dScore = 95;
 int eScore = 55;
 int fScore = 70;
 int hScore = 40;

 int sum = aScore + bScore + cScore + dScore + eScore
 + fScore + hScore;
 System.out.println(sum);

 double average = sum / 7.0;
 System.out.println(average);
}
```

# 14. 배열

---

- 성적을 관리해야 하는 한 반의 학생의 수가 100명이면 어떤 일을 어떻게 해야 할지 생각해보세요.
  - 점수는 일일이 추가할 수 밖에 없겠지만 점수가 변경되거나, 학생이 누락되었을 경우 코드를 일일이 찾아 수정해야 합니다.
  - 또한, 100 명의 성적 합계를 구하기 위해 일일이 더하는 코드를 작성하면서 누락되는 변수도 존재할 수 있습니다.
  - 더 나아가 전교생의 성적을 관리해야 한다면 작업량은 더 늘어나게 됩니다.
- 
- 위 예처럼, 동일한 성격의 동일한 타입의 값들을 관리해야 할 때 배열을 사용하면 비교적 편하게 코드를 작성할 수 있고, 수정에 대한 버그 발생율이 현저히 감소하게 됩니다.

# 14. 배열

- 배열을 이용한 성적 관리 프로그램 작성

```
public static void main(String[] args) {
 int[] scoreArray = new int[7];
 scoreArray[0] = 50;
 scoreArray[1] = 70;
 scoreArray[2] = 65;
 scoreArray[3] = 95;
 scoreArray[4] = 55;
 scoreArray[5] = 70;
 scoreArray[6] = 40;

 // 배열을 순회하며 합계 구하기
 int sum = 0;
 for (int i = 0; i < scoreArray.length; i++) {
 sum += scoreArray[i];
 }
 System.out.println(sum);

 // 합계와 배열의 개수로 평균 점수 구하기
 double average = sum / (double) scoreArray.length;
 System.out.println(average);
}
```

# 14. 배열

- 배열을 이용한 성적 관리 프로그램 설명
- 배열의 선언과 생성 및 할당

```
// 배열타입[] 배열인스턴스명 = new 배열타입[배열의 길이];
int[] scoreArray = new int[7];
```

| 코드         | 역할        | 설명                                  |
|------------|-----------|-------------------------------------|
| int[]      | 배열 타입 선언  | int형 배열을 선언                         |
| new int[7] | 배열 생성자 호출 | int형 배열의 인스턴스를 생성하고 인덱스는 0부터 6까지 생성 |

- scoreArray 배열의 구조

| 구분    | scoreArray[0] | scoreArray[1] | scoreArray[2] | scoreArray[3] | scoreArray[4] | scoreArray[5] | scoreArray[6] |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| 값     | 0             | 0             | 0             | 0             | 0             | 0             | 0             |
| index | 0             | 1             | 2             | 3             | 4             | 5             | 6             |

- 배열의 각 칸을 인덱스(index)라고 하며, 인덱스는 항상 0부터 시작합니다.

# 14. 배열

- 배열을 이용한 성적 관리 프로그램 설명
- 배열 인덱스에 값 할당

```
// 배열인스턴스명[인덱스번호] = 값;
scoreArray[0] = 50;
scoreArray[1] = 70;
scoreArray[2] = 65;
scoreArray[3] = 95;
scoreArray[4] = 55;
scoreArray[5] = 70;
scoreArray[6] = 40;
```

| 구분    | scoreArray[0] | scoreArray[1] | scoreArray[2] | scoreArray[3] | scoreArray[4] | scoreArray[5] | scoreArray[6] |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| 값     | 0             | 0             | 0             | 0             | 0             | 0             | 0             |
| index | 50            | 70            | 65            | 95            | 55            | 70            | 40            |

- 배열인스턴스에 값을 할당하려면  
**배열인스턴스명[인덱스번호] = 값;** 형태로 작성합니다.

# 14. 배열

- 배열을 이용한 성적 관리 프로그램 설명
- 배열 순회하며 합계 구하기

```
int sum = 0;
for (int i = 0; i < scoreArray.length; i++) {
 sum += scoreArray[i];
}
```

| i의 값 | scoreArray.length의 값 | 반복 조건   | 반복 조건 결과 | 다음 반복 여부 | 참조 인덱스        | sum 변수의 값 |
|------|----------------------|---------|----------|----------|---------------|-----------|
| 0    | 7                    | $0 < 7$ | true     | 반복       | scoreArray[0] | 50        |
| 1    | 7                    | $1 < 7$ | true     | 반복       | scoreArray[1] | 120       |
| 2    | 7                    | $2 < 7$ | true     | 반복       | scoreArray[2] | 185       |
| 3    | 7                    | $3 < 7$ | true     | 반복       | scoreArray[3] | 280       |
| 4    | 7                    | $4 < 7$ | true     | 반복       | scoreArray[4] | 335       |
| 5    | 7                    | $5 < 7$ | true     | 반복       | scoreArray[5] | 405       |
| 6    | 7                    | $6 < 7$ | true     | 반복       | scoreArray[6] | 445       |
| 7    | 7                    | $7 < 7$ | false    | 종지       | -             |           |

- 배열인스턴스 인덱스의 값을 참조하려면  
변수 = 배열인스턴스명[인덱스번호]; 형태로 작성합니다.

# 14. 배열

---

- 배열을 이용한 성적 관리 프로그램 설명
- 배열인스턴스 인덱스의 개수로 평균 구하기

// 합계와 배열의 개수로 평균 점수 구하기

```
double average = sum / (double) scoreArray.length;
```

# 14. 배열

- 모든 학생들의 점수를 5점씩 더하기

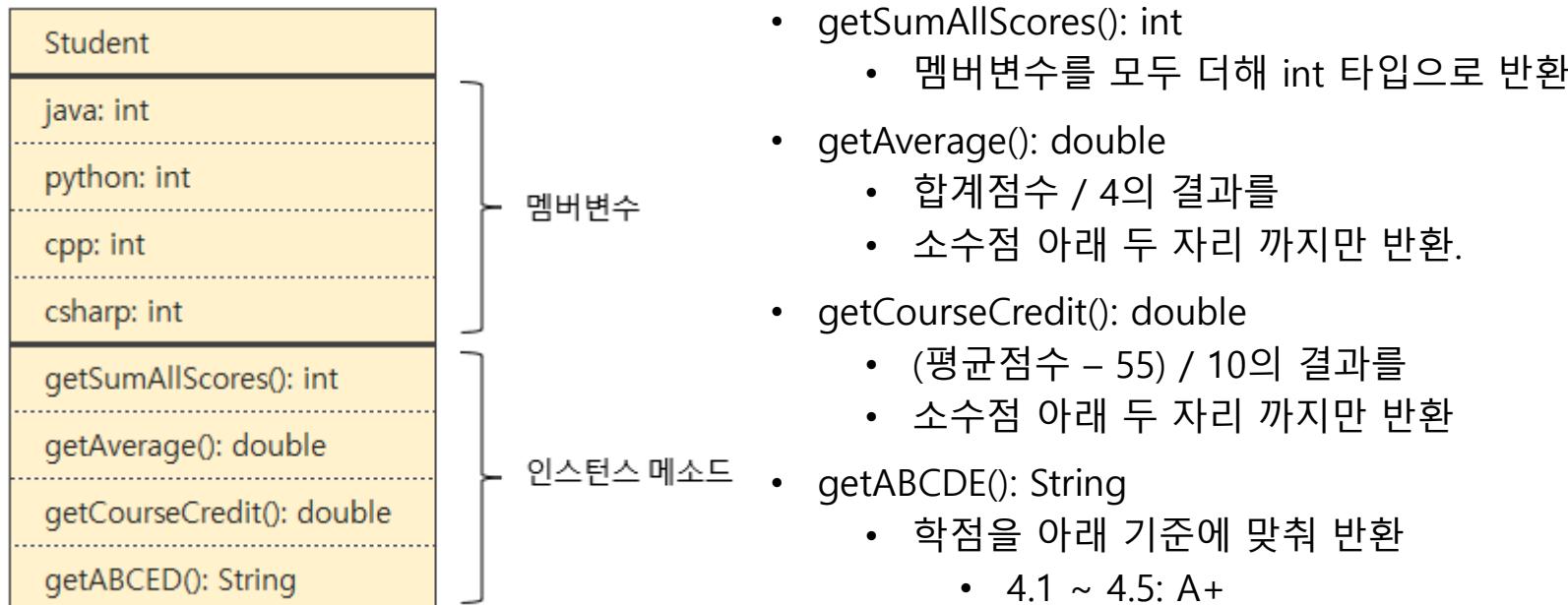
```
public static void main(String[] args) {
 // 배열타입[] 배열인스턴스명 = new 배열타입[배열의 길이];
 int[] scoreArray = new int[7];
 // 배열인스턴스명[인덱스번호] = 값;
 ... 생략 ...

 // 배열을 순회하며 5점씩 더하기
 for (int i = 0; i < scoreArray.length; i++) {
 scoreArray[i] += 5;
 }

 // 배열을 순회하며 합계 구하기
 ... 생략 ...
}
```

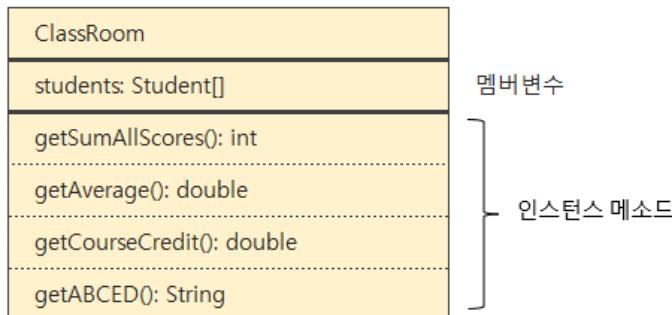
# 14. 배열

- Student 배열을 이용해 한 반의 성적 총합, 평균, 학점, 등급을 계산하고 출력하는 클래스를 만들고 실행해보세요.



# 14. 배열

- Student 배열을 이용해 한 반의 성적 총합, 평균, 학점, 등급을 계산하고 출력하는 클래스를 만들고 실행해보세요.



- getSumAllScores(): 한 반의 성적 총합
- getAverage(): 한 반의 평균 성적
- getCourseCredit(): 한 반의 학점
- getABCDE(): 한 반의 등급

# 14. 배열

---

- 배열 실습문제
- 상품클래스와 배열을 이용해 쇼핑몰 판매자 기능 구현해보기
  - 상품클래스(데이터 클래스) 속성
    - 상품명: String
    - 상품가격: int
    - 재고 수: int
  - 판매자 클래스 속성
    - 상품클래스 배열: 상품[]
  - 판매자 클래스 기능
    - 판매(상품명, 주문수량): void
      - 상품클래스 배열에서 상품명을 찾아 주문수량만큼 재고 감소시키기
      - 주문수량이 재고수보다 크거나
      - 재고수가 0이라면 “판매할 수 없습니다” 출력하기

# 15. 상속(Inheritance)과 다형성(Polymorphism)

# 15. 상속(Inheritance)

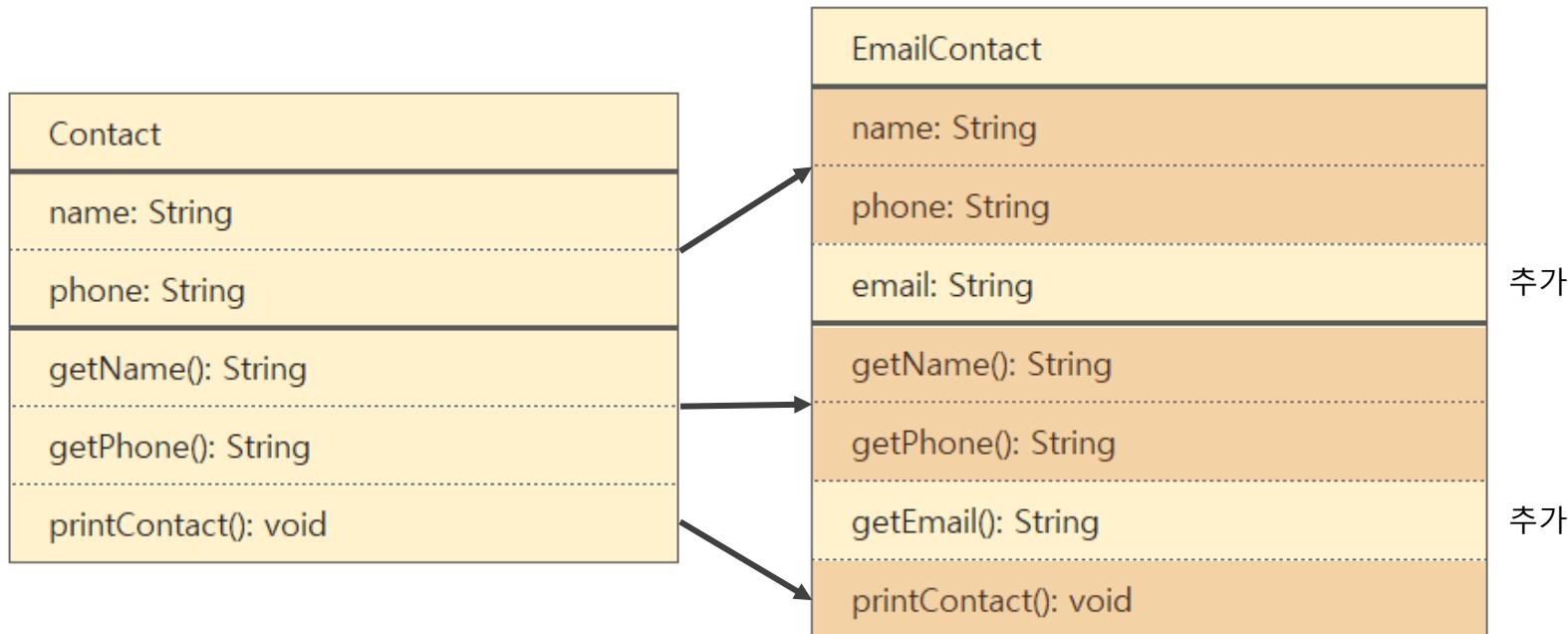
# 15. 상속(Inheritance)

---

- 상속이란, 부모클래스의 속성과 기능을 자식클래스에게 그대로 물려주는 것을 말합니다.
- 클래스의 멤버변수나 메소드의 추가가 필요할 경우
  - 1. 클래스를 직접 수정하는 방법
  - 2. 기존의 클래스를 상속받아 필요한 변수나 기능만 추가하는 방법
- 1번 방법
  - 연결 되어있던 코드를 모두 수정 해야 할 가능성이 매우 높아집니다.
  - 결국, 하나의 기능 추가 및 변경을 위해 모든 코드를 수정 해야 할 수도 있습니다.
- 2번 방법
  - 연결 되어있던 기존의 코드에 영향을 주지 않고 안전하게 수정사항을 반영할 수 있습니다.

# 15. 상속(Inheritance)

- 상속 예제 (연락처 관리 프로그램)
  - 이름과 연락처만 관리하는 Contact 클래스가 있습니다.
  - 시간이 지나 연락처에 Email도 관리해달라는 요구가 발생했습니다.
    - Contact 클래스에 Email을 직접 추가해도 문제없지만
    - 유연한 확장을 위해 EmailContact 클래스를 만들어 추가하려 합니다.
    - Contact는 이름과 연락처만을 관리하는 클래스.
    - EmailContact는 이메일까지 관리하는 클래스로 사용됩니다.



# 15. 상속(Inheritance)

---

- 상속 예제 (연락처 관리 프로그램) \_ 1 / 2

```
public class Contact {
 private String name;
 private String phone;

 public Contact(String name, String phone) {
 this.name = name;
 this.phone = phone;
 }

 public String getName() {
 return this.name;
 }

 public String getPhone() {
 return this.phone;
 }

 public void printContact() {
 System.out.println("이름: " + this.name + ", 연락처: " + this.phone);
 }
}
```

# 15. 상속(Inheritance)

- 상속 예제 (연락처 관리 프로그램) \_ 2 / 2

```
public class EmailContact extends Contact {

 private String email;

 public EmailContact(String name, String phone, String email) {
 super(name, phone);
 this.email = email;
 }

 public String getEmail() {
 return email;
 }

 @Override
 public void printContact() {
 super.printContact();
 System.out.println("이메일: " + this.email);
 }
}
```

# 15. 상속(Inheritance)

---

- 상속 예제 EmailContact 코드 설명

```
public class EmailContact extends Contact {
```

- EmailContact클래스는 Contact클래스를 상속(확장)합니다.
- Contact에 있는 모든 변수와 메소드를 그대로 사용할 수 있습니다.
  - 멤버변수: name, phone
  - 메소드: getName(), getPhone(), printContact()
  - 생성자: Contact()

# 15. 상속(Inheritance)

---

- 상속 예제 EmailContact 코드 설명

```
public EmailContact(String name, String phone, String email) {
 super(name, phone);
 this.email = email;
}
```

- super(name, phone)는 부모클래스(Contact)의 생성자를 호출합니다.
- 즉, Contact(name, phone)가 실행됩니다.
  - 이 코드로 인해서 EmailContact는 멤버변수 name, phone에 데이터를 할당할 수 있게 됩니다.
- 부모클래스에 파라미터가 있는 생성자만 있을 경우,  
자식 클래스에서 부모 클래스의 생성자를 반드시 호출해야 합니다.

# 15. 상속(Inheritance)

- 상속 예제 EmailContact 코드 설명

```
@Override
public void printContact() {
 super.printContact();
 System.out.println("이메일: " + this.email);
}
```

- @Override는 16장에서 자세히 설명합니다.
- super.printContact();는 부모클래스(Contact)의 printContact()를 호출합니다.
  - 반드시 호출해야 할 의무는 없습니다.  
다만, 부모클래스의 기능을 동시에 사용하려고 한다면 호출해야 합니다.
  - Contact의 printContact() 메소드는 이름과 연락처를 출력하므로,  
EmailContact의 printContact()는 이름, 연락처, 이메일을 모두 출력하게 됩니다.
  - 만약, super.printContact(); 코드가 없다면 이메일만 출력하게 됩니다.

# 15. 상속(Inheritance)

- 상속 클래스 사용 예제

```
public class Main {

 public static void main(String[] args) {

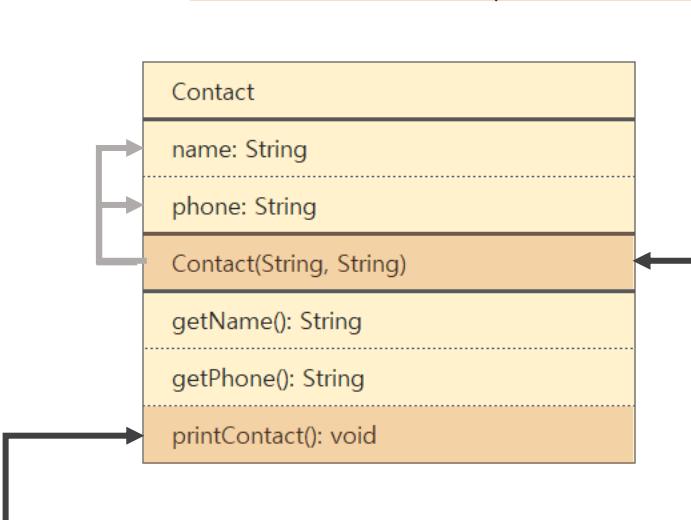
 Contact contact1 = new Contact("홍길동", "010-1111-2222");
 Contact contact2 = new Contact("언년", "010-3333-4444");
 contact1.printContact();
 contact2.printContact();

 EmailContact emailContact1 = new EmailContact("친구1",
 "010-2222-3333", "friends1@gmail.com");
 EmailContact emailContact2 = new EmailContact("친구2",
 "010-4444-5555", "friends2@gmail.com");
 emailContact1.printContact();
 emailContact2.printContact();
 }
}
```

# 15. 상속(Inheritance)

- 상속 클래스 사용예제의 호출 관계

```
Contact contact1 = new Contact("홍길동", "010-1111-2222");
```

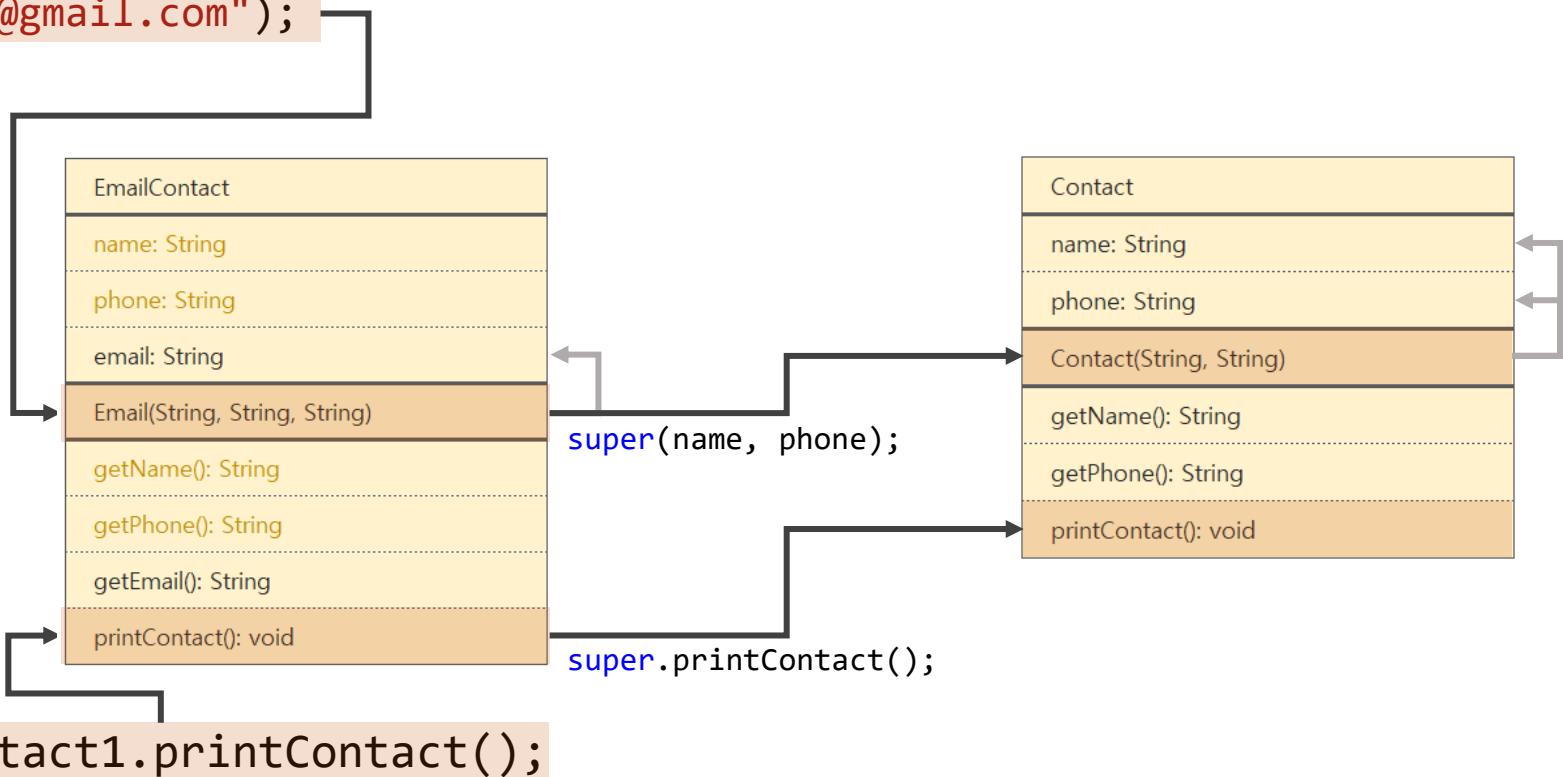


```
contact1.printContact();
```

# 15. 상속(Inheritance)

- 상속 클래스 사용예제의 호출 관계

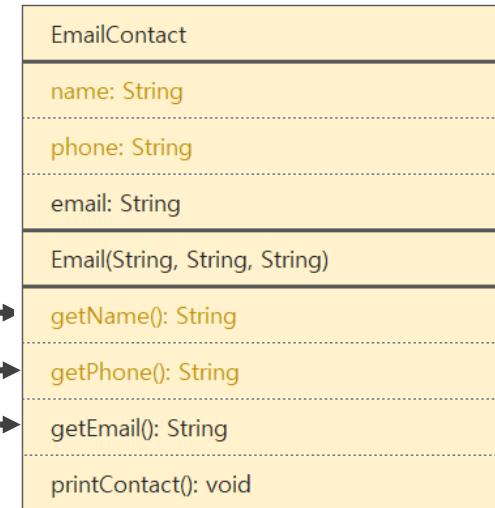
```
EmailContact emailContact1 = new EmailContact("친구1", "010-2222-3333",
"friends1@gmail.com");
```



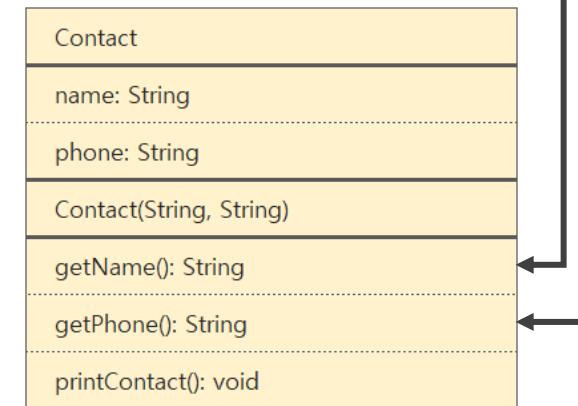
# 15. 상속(Inheritance)

- 상속 클래스 사용예제의 호출 관계

```
System.out.println(emailContact1.getName());
System.out.println(emailContact1.getPhone());
System.out.println(emailContact1.getEmail());
```

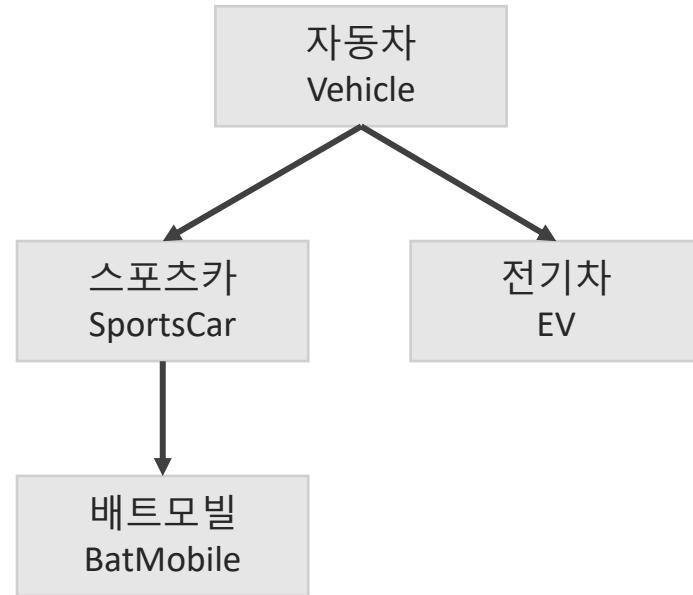


- 상속받은 클래스의 메소드를 실행할 경우  
부모클래스의 메소드를 실행시키게 됩니다.



# 15. 상속(Inheritance)

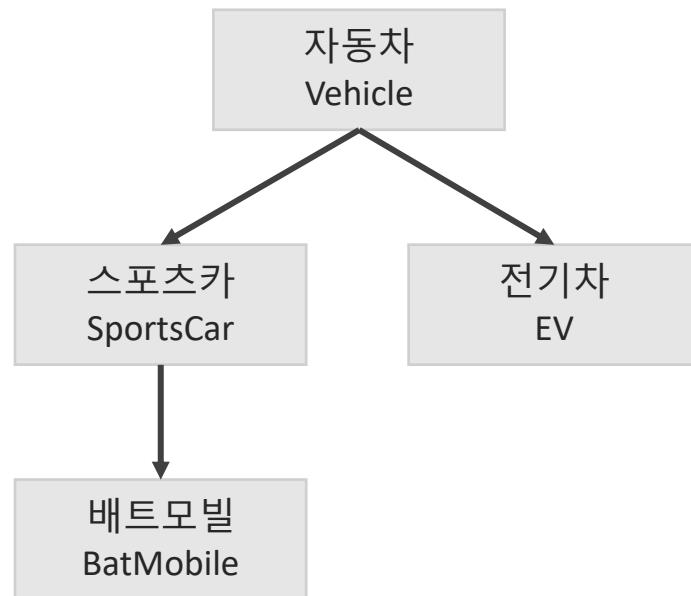
- 상속 연습문제
  - 자동차를 종류별로 구현해보고 인스턴스를 만들어보세요.
    - 자동차(Vehicle)
      - 속성: 자동차모델명
      - 기능: 시동걸기
    - 전기차(EV)
      - 속성: 자동차모델명
      - 속성: 배터리 양
      - 기능: 시동걸기
      - 기능: 배터리체크
    - 스포츠카(SportsCar)
      - 속성: 자동차모델명
      - 기능: 시동걸기
      - 기능: 터보모드
    - 배트모빌(BatMobile)
      - 속성: 자동차모델명
      - 기능: 시동걸기
      - 기능: 터보모드
      - 기능: 배트포트 분리



# **15. 상속의 유연성**

# 15. 상속의 유연성

- 상속 연습문제에서 만든 Vehicle에 기능 추가하기.
  - “시동끄기” 메소드를 만들어봅니다.
  - 스포츠카, 전기차, 배트모빌에서 “시동끄기” 메소드의 사용이 가능한지 확인해 봅니다.



# 15. 상속의 유연성

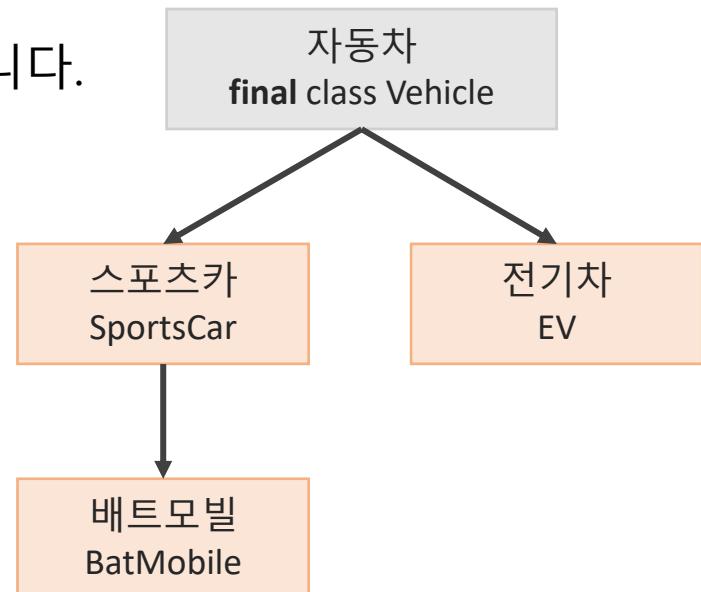
---

- 상속은 기존의 클래스에서 확장한 것으로 부모 클래스를 수정할 경우 자식 클래스에서 수정된 내용을 즉시 사용할 수 있습니다.
- 만약, 상속을 하지 않고 일일이 기능을 추가할 경우, 필요한 클래스를 모두 찾아 수정을 해야하는 불편함이 생기게 됩니다.
- 이처럼 상속은 클래스 속성이나 기능의 추가/수정/삭제를 유연하게 처리할 수 있게 해줍니다.

## 15. 상속이 불가능한 final class

# 15. 상속이 불가능한 final class

- Java의 모든 클래스는 상속이 가능합니다.
- 단, 클래스 선언부에 final 키워드가 존재할 경우는 상속이 불가능합니다.
  - Vehicle 클래스에 final을 추가해봅니다.
    - Vehicle를 상속받는 스포츠카, 전기차, 스포츠카를 상속받는 배트모빌 모두 에러가 발생함을 알 수 있습니다.
- 상속이 되지 않길 바라는 클래스에는 final 키워드를 추가해 상속을 제한 할 수 있습니다.



# 15. 상속을 제한하는 sealed class

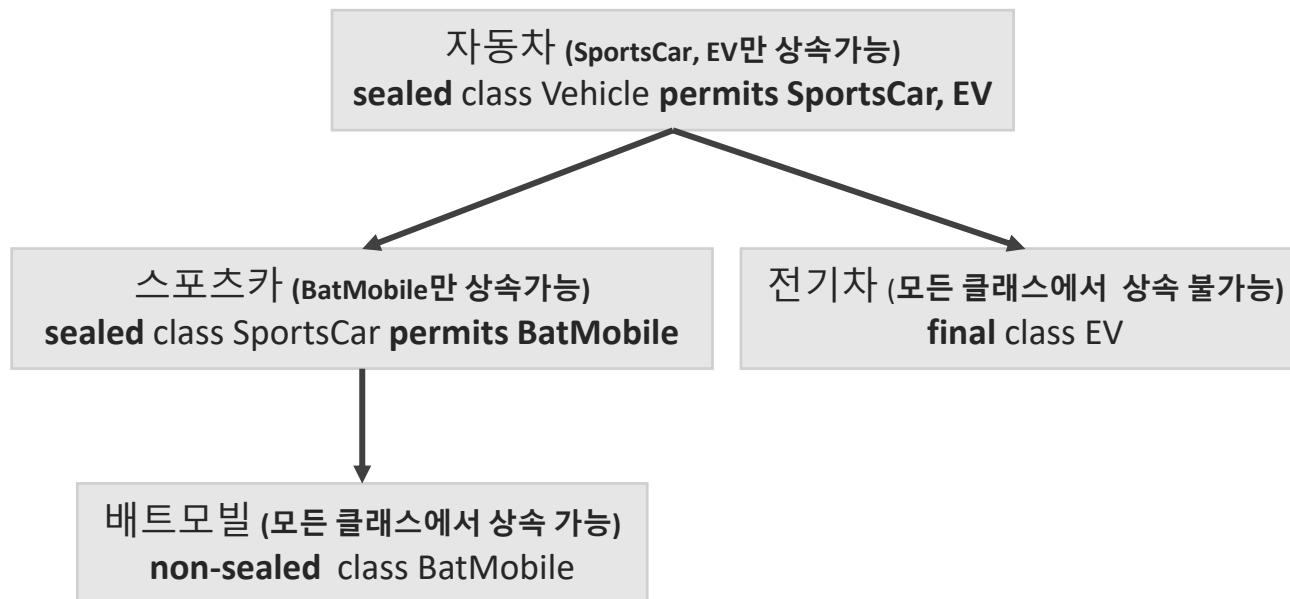
# 15. 상속을 제한하는 sealed class

---

- Java 17부터 추가된 sealed 키워드는 final과 마찬가지로 상속을 할 수 없도록 제한시키는 키워드입니다.
- final 키워드와의 차이점
  - final: 상속이 불가능한 클래스로 만듦.
  - sealed: 지정 클래스를 제외하고 상속이 불가능하도록 만듦.
    - sealed를 상속받는 클래스는 아래 3가지 중 하나를 선택해야 합니다.
      - sealed: 지정 클래스를 제외하고 상속이 불가능
      - non-sealed: 모든 클래스에서 상속 가능
      - final: 상속이 불가능

# 15. 상속을 제한하는 sealed class

- 아래 처럼 변경해 보세요.



# 15. 다형성(Polymorphism)

# 15. 다형성(Polymorphism)

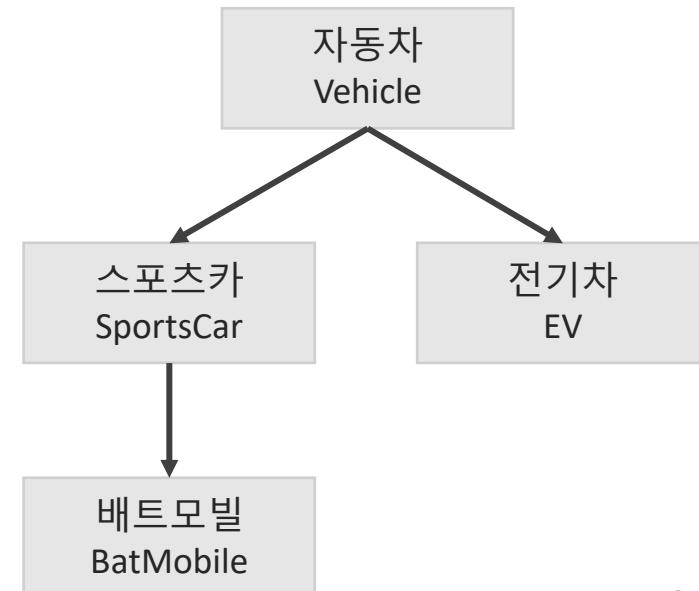
---

- 객체지향프로그래밍의 4가지 특징
  - 1. 캡슐화
    - 여러 기능을 메소드(캡슐)로 묶어 처리한다.
  - 2. 추상화
    - 기능의 정의만 존재하고 구현은 하지 않는다
  - 3. 다형성
    - 상속 및 구현 대상(부모클래스) 타입에 포함되는 것을 허가한다.
    - 즉, 하나의 타입으로 여러가지 타입을 표현할 수 있다.
  - 4. 상속
    - 부모클래스의 모든 속성, 기능을 확장한다.

## **15. is a 관계를 이용한 다형성**

# 15. is a 관계를 이용한 다형성

- 상속 관계에 있는 클래스들은 아래 규칙으로 표현이 가능합니다.
  - Sub class **is a** Super class.
    - Sub class: 상속을 받아 확장한 클래스
      - SportsCar는 Vehicle의 Sub class
      - BatMobile은 SportsCar, Vehicle의 Sub class
    - Super class: 상속의 대상이 되는 클래스
      - Vehicle은 EV, SportsCar, BatMobile의 Super class
      - SportsCar는 BatMobile의 Super class
  - BatMobile **is a** SportsCar
  - BatMobile **is a** Vehicle
  - SportsCar **is a** Vehicle
  - EV **is a** Vehicle



# 15. is a 관계를 이용한 다형성

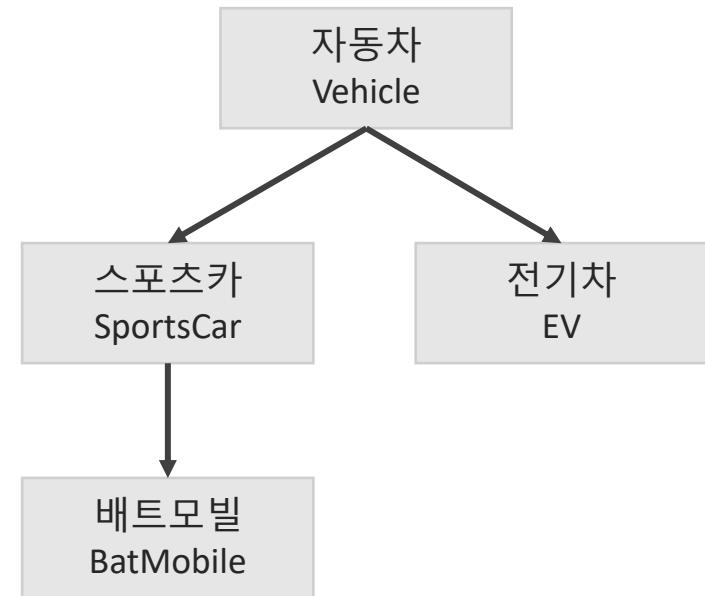
- is a 관계가 성립할 경우, 다형성을 이용해 유연한 변수 및 파라미터 타입 정의가 가능해집니다

```
// SportsCar is a Vehicle
Vehicle car1 = new SportsCar();
```

```
// BatMobile is a SportsCar
SportsCar car2 = new BatMobile();
```

```
// BatMobile is a Vehicle
Vehicle car3 = new BatMobile();
```

```
// EV is a Vehicle
Vehicle car4 = new EV();
```



# 15. is a 관계를 이용한 다형성

---

- Vehicle, SportsCar, EV, BatMobile을 하나의 메소드로 처리.
  - 다형성을 이용하면 Super class로 Sub class를 제어할 수 있게됩니다.
- startEngine(Vehicle vehicle) 메소드로 모든 차량의 시동걸기 메소드 (startEngine()) 호출해보기 실습

```
public void startEngine(Vehicle vehicle) {
 vehicle.startEngine();
}

public static void main(String[] args) {
 Vehicle car1 = new Vehicle();
 Vehicle car2 = new SportsCar();
 Vehicle car3 = new EV();
 Vehicle car4 = new BatMobile();

 Main main = new Main();
 main.startEngine(car1);
 main.startEngine(car2);
 main.startEngine(car3);
 main.startEngine(car4);
}
```

# 16. 메소드 오버라이딩

# 16. 메소드 오버라이딩

- 부모클래스의 메소드를 자식 클래스에서 재 정의하는 것.
  - 객체 지향 설계 5원칙 중 리스코프 치환의 원칙(LSP)은 is a 관계는 유지하되, 부모클래스의 메소드를 오버라이딩 하지 말 것을 권고.
  - 즉, 자식 클래스에서 부모클래스의 메소드를 아무렇게나 오버라이딩 하지 말 것.

```
public class EmailContact extends Contact {

 private String email;

 ... 생략 ...

 @Override ← Contact 클래스의 printContact 메소드를 EmailContact에서 재정의
 public void printContact() {
 super.printContact();
 System.out.println("이메일: " + this.email);
 }
}
```

# 16. 메소드 오버라이딩

---

- @Override 를 작성하지 않아도 메소드 오버라이딩은 정상 동작합니다.
  - 단, @Override 를 작성하지 않을 경우, 미미한 성능 하락을 일으킵니다.
  - 동시접속 사용자가 많은(트래픽이 많은|트랜잭션이 많은) 애플리케이션일 경우, 미미한 성능의 하락이 매우 큰 성능 하락을 초래하기 때문에, @Override는 반드시 작성해 주어야 합니다.

# 16. 오버라이딩이 불가능한 final method

# 16. 오버라이딩이 불가능한 final method

- 자식 클래스가 부모 클래스의 메소드를 재정의 하지 못하도록 막으려면 부모 클래스의 메소드에 final 을 붙여줍니다.

```
public sealed class Vehicle permits SportsCar, EV{
 private String name;

 public Vehicle(String name) {
 this.name = name;
 }

 public final void startEngine() {
 System.out.println(name + " 시동을 겁니다.");
 }
}
```

# 16. 오버라이딩이 불가능한 final method

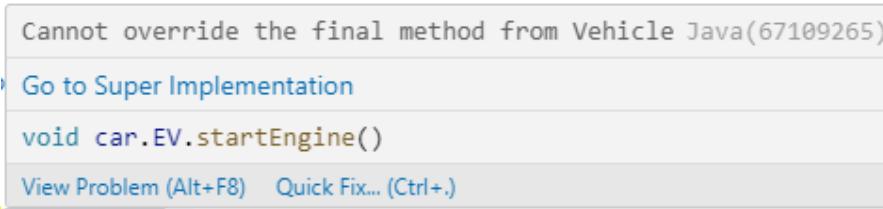
- Vehicle의 자식 클래스에서 startEngine()을 Override 하려하면 아래와 같이 에러가 발생합니다.

```
package car;
public final class EV extends Vehicle {

 public int battery;

 ... 생략 ...

 @Override
 public void startEngine() {
 System.out.println("소리 없습니다.");
 }
}
```



The screenshot shows a tooltip from an IDE. The main message is "Cannot override the final method from Vehicle Java(67109265)". Below it is a link "Go to Super Implementation". At the bottom of the tooltip, there are "View Problem (Alt+F8)" and "Quick Fix... (Ctrl+.)" buttons.

# Java Programming

## 고급

---

- 17. abstract class / interface / 추상화  
/ 익명클래스
- 18. 예외처리
- 19. 제네릭과 컬렉션
- 20. File I/O
- 21. 열거형 (Enum)
- 22. Calender / LocalDateTime

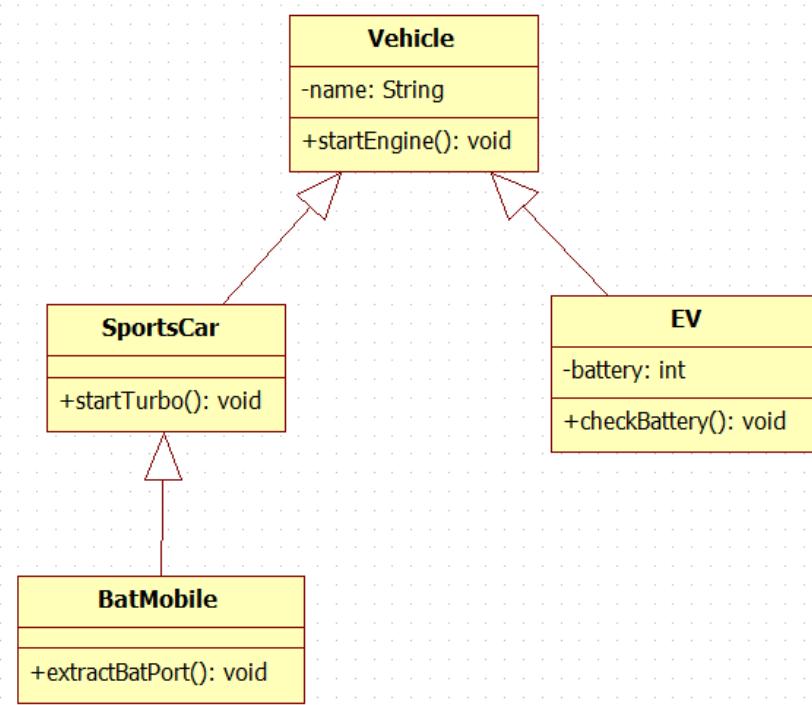
# 17. abstract class / interface / 추상화 / 익명클래스

메소드 실행 전 후에 공통 처리를 하고싶다면  
추상클래스가 답입니다!

## 17. abstract class

# 17. abstract class

- 상속과 is a 관계의 한계

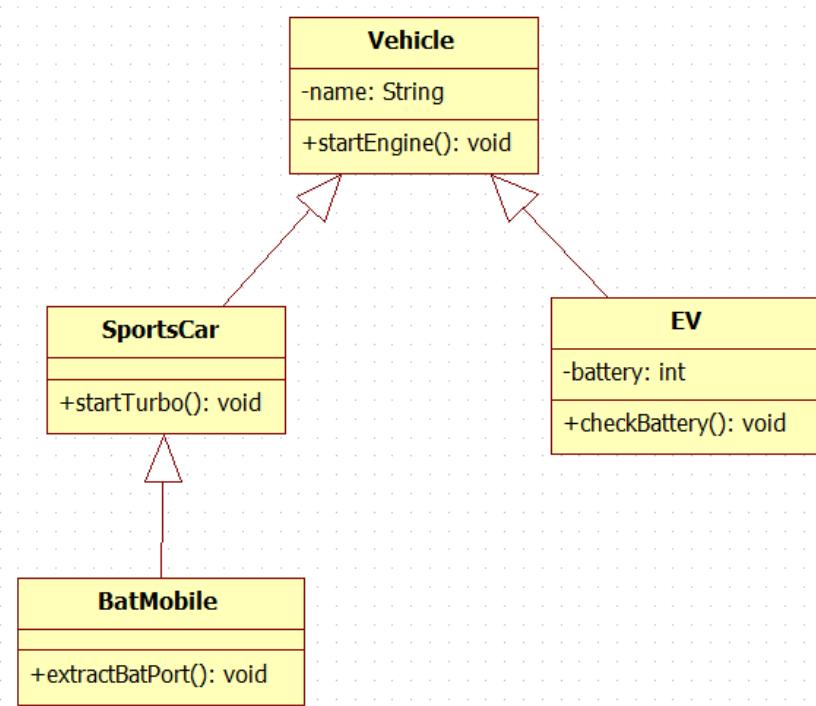


```
Vehicle car2 = new SportsCar();
Vehicle car3 = new EV();
Vehicle car4 = new BatMobile();
```

- car2 인스턴스는 startTurbo()
- car3 인스턴스는 checkBattery()
- car4 인스턴스는 startTurbo(), extractBatPort()를 사용할 수 없습니다.
- car2, car3, car4 인스턴스의 타입이 Vehicle로 선언되어있기 때문에 사용할 수 있는 메소드는 startEngine() 밖에 없습니다.

# 17. abstract class

- 인스턴스의 모든 메소드를 사용하고 싶다면?



```
SportsCar car2 = new SportsCar();
EV car3 = new EV();
BatMobile car4 = new BatMobile();
```

- 인스턴스 생성자와 인스턴스 타입을 동일하게 맞춰주면, `car2`, `car3`, `car4` 인스턴스들의 모든 메소드를 사용할 수 있습니다.
- 이렇게 사용할 경우 상속을 사용한 의미를 잃게 됩니다.

# 17. abstract class

- 추상클래스를 사용할 경우, 상속의 의미를 잃지 않으면서 자연스러운 is a 관계를 만들 수 있습니다.
- Vehicle을 추상 클래스로 만들어봅니다.

```
public sealed abstract class Vehicle permits SportsCar, EV {
 private String name;

 public Vehicle(String name) {
 this.name = name;
 }

 public final void startEngine() {
 System.out.println(name + " 시동을 겁니다.");
 }
}
```

## 추상클래스를 만들 때 주의!

추상 클래스는 객체화 시킬 수 없는 클래스입니다.  
따라서 Vehicle 클래스는 인스턴스화 할 수 없게 됩니다.

# 17. abstract class

- 그리고 SportsCar의 startTurbo, BatMobile의 extractBatPort, EV의 checkBattery 메소드를 Vehicle에 만들어줍니다.
- 이 때, 메소드들의 정의만 해줍니다.

```
package car;
public sealed abstract class Vehicle permits SportsCar, EV{
 private String name;
 ... 생략 ...
 public abstract void startTurbo();
 public abstract void extractBatPort();
 public abstract void checkBattery();
}
```

**추상 클래스는 추상 메소드를 만들 수 있습니다.**

추상의 의미는 “구체화되어있지 않은”입니다.

추상 메소드란, 어떤 기능을 할지 모르는 즉, 기능 구현이 되어있지 않은 대략적인 메소드를 말합니다. 기능 구현이 되어있지 않은 추상 메소드를 포함할 때, 추상 클래스로 만듭니다.

# 17. abstract class

---

- Vehicle이 추상 클래스로 변경이 되었고, 추상 메소드가 포함됨에 따라 SportsCar, BatMobile, EV 클래스에서 에러가 발생합니다.
- 추상 메소드가 포함된 추상 클래스를 상속받는 클래스는 추상 클래스의 모든 추상 메소드를 반드시 구현해야만 합니다.
- 따라서 SportsCar, Mobile, EV 클래스는 startTurbo(), extractBatMobile(), checkBattery() 메소드를 모두 구현해야만 합니다.

# 17. abstract class

```
public final class EV extends Vehicle {
 public int battery;

 public EV(String name, int battery) {
 super(name);
 this.battery = battery;
 }
 @Override
 public void checkBattery() {
 System.out.println("배터리가 " + this.battery + " 만큼 남아있습니다.");
 }

 @Override
 public void extractBatPort() {
 System.out.println("지원하지 않는 기능입니다.");
 }

 @Override
 public void startTurbo() {
 System.out.println("지원하지 않는 기능입니다.");
 }
}
```

# 17. abstract class

```
public sealed class SportsCar extends Vehicle permits BatMobile {

 public SportsCar(String name) {
 super(name);
 }

 @Override
 public void startTurbo() {
 System.out.println("터보 모드를 시작합니다!");
 }

 @Override
 public void extractBatPort() {
 System.out.println("지원하지 않는 기능입니다.");
 }

 @Override
 public void checkBattery() {
 System.out.println("지원하지 않는 기능입니다.");
 }
}
```

# 17. abstract class

---

```
public non-sealed class BatMobile extends SportsCar {

 public BatMobile(String name) {
 super(name);
 }

 @Override
 public void extractBatPort() {
 System.out.println("배트포트를 분리합니다.");
 }
}
```

# 17. abstract class

---

- 다시 is a 관계로 인스턴스를 만든 뒤 각 메소드를 호출해 보면, 이상없이 동작하는 모습을 볼 수 있습니다.

```
public class App {
 public static void main(String[] args) {
 Vehicle car2 = new SportsCar("");
 Vehicle car3 = new EV("", 50);
 Vehicle car4 = new BatMobile("");

 car4.startEngine();
 car3.checkBattery();
 car2.startTurbo();
 car2.checkBattery();
 }
}
```

개발자의 약속

이 형태로 만들고 이 형태로 호출해주세요!

## 17. interface

# 17. interface

- 네이버 국어사전에서 정의한 인터페이스의 의미

국어사전

다른 어학정보 9 ▾

## 인터페이스 (interface) ▶

명사

- 서로 다른 두 시스템, 장치, 소프트웨어 따위를 서로 이어 주는 부분. 또는 그런 접속 장치.
- 사용자인 인간과 컴퓨터를 연결하여 주는 장치. 키보드나 디스플레이 따위를 이른다.

### 인터페이스 / I/F

타 업무/시스템에 데이터를 보내거나 가져오는것  
오픈사전PRO

# 17. interface

- UI (User Interface)

손에 잡히는 IT 시사용어

## 사용자 인터페이스

[ User Interface  ]

약어 UI

UI란 사람과 컴퓨터 시스템·프로그램 간 상호작용을 의미한다. 그러므로 UI 디자인은 사용자와 컴퓨터·프로그램 간 의사소통의 효과성과 효율성을 극대화하기 위해 인간, 환경, 기술 요소를 통합하는 활동이라 할 수 있다.

UI는 디스플레이 화면, 키보드, 마우스, 라이트펜, 데스크톱 형태, 채색된 글씨들, 도움말 등 사람들과 상호작용을 하도록 설계된 모든 정보 관련 고안품을 포함하며, 응용 프로그램이나 웹사이트 등이 상호작용을 초래하거나 그 것에 반응하는 방법 등을 의미한다. 두 객체를 통합한 하나의 단일화된 시스템을 구축하기 위해 중간 매개체로서 인터페이스가 필요하게 되었고, 이를 바탕으로 사용자 인터페이스라는 용어가 나오게 된 것이다. 사용자 인터페이스란 상당히 주관적인 것 같으면서도 많은 부분을 모든 사람들이 공통적으로 느낀다.

최근 차세대 인터페이스를 도입하는 운용체계가 출시되었는데, 애플의 OS X 레오파드와 윈도 비스타가 그것이다. 비스타에서는 폰트를 바꾸면 실시간 프리뷰를 할 수 있거나 2D와 3D, 동영상 등을 한 화면에서 동시에 섞어 사용할 수 있는 소프트웨어를 이용할 수 있다. 또한 애플에서 출시한 아이폰(iPhone)에는 한꺼번에 다중 컨트롤이 가능한 멀티터치 디스플레이가 채용되어, 기존에는 사용자가 스타일러스 펜으로 아이콘을 누르던 것을 손가락으로 쉽게 선택하고, 여러 기능들을 두 손가락으로 조작할 수 있게 되었다.

# 17. interface

- API (Application Programming Interface)

두산백과

**API**

[ application programming interface ]

요약

운영체제와 응용프로그램 사이의 통신에 사용되는 언어나 메시지 형식을 말한다.

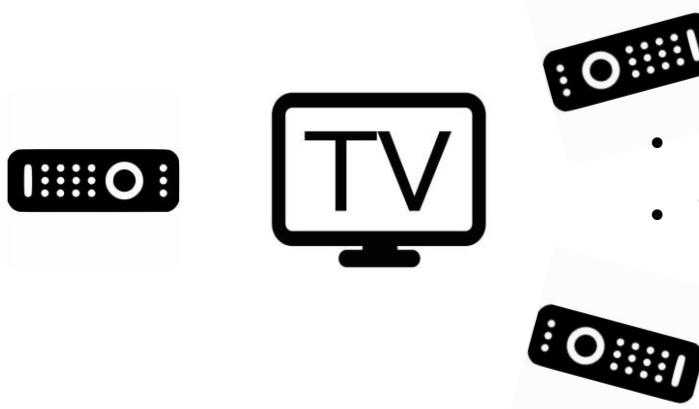
운영체제나 C, C++, Pascal 등과 같은 언어로 응용 프로그램을 만들 때, 윈도우를 만들고 파일을 여는 것과 같은 처리를 할 수 있도록 1,000여 개 이상의 함수로 구성되어 있다. API는 프로그래머를 위한 운영체제나 프로그램의 인터페이스로서 사용자와 직접 대하게 되는 그래픽 사용자 인터페이스나 명령형 인터페이스와 뚜렷한 차이가 있다.

API는 응용 프로그램이 운영체제나 데이터베이스 관리 시스템과 같은 시스템 프로그램과 통신할 때 사용되는 언어나 메시지 형식을 가지며, API는 프로그램 내에서 실행을 위해 특정 서브루틴에 연결을 제공하는 함수를 호출하는 것으로 구현된다. 그러므로 하나의 API는 함수의 호출에 의해 요청되는 작업을 수행하기 위해 이미 존재하거나 또는 연결되어야 하는 몇 개의 프로그램 모듈이나 루틴을 가진다.

좋은 API는 모든 building block을 제공함으로써 프로그램 개발을 쉽게 해준다. 프로그래머는 그 block을 함께 합치기만 하면 된다. API가 프로그래머를 위해서 만들어지기는 했지만, 사용자 입장에서도 같은 API를 사용한 프로그램은 비슷한 인터페이스를 가지기 때문에 새로운 프로그램의 사용법을 배우기가 쉬워진다.

# 17. interface

- Interface의 정의에서 공통으로 나오는 주제.
  - 연결, 상호작용, 메시지
- Java Interface의 의미
  - 1. 클래스와 클래스가 상호작용할 수 있도록 표준을 제공
  - 2. 개발자간 커뮤니케이션을 위한 표준을 제공
- 인터페이스는 “다형성(객체지향프로그래밍의 특징 중 하나)” 제공.
  - 하나의 인터페이스를 이용해 여러 개의 클래스를 생성 할 수 있다.



- 다형성
- TV와 리모콘이 통신하는 표준 규격을 통해 여러 형태의 리모콘이 TV와 연결될 수 있다.

# 17. interface

- 인터페이스 만들어보기

```
public interface SomeInterface {

 public abstract void doSomething1();
 public abstract void doSomething2();
 public abstract void doSomething3();
 public abstract int getSomething();
 public abstract String getSomething2();
}
```

생략가능

- 인터페이스는 오로지 추상메소드와 상수의 정의를 위해 사용합니다.
- 추상 클래스처럼 일반 메소드는 정의할 수 없습니다.
  - Java 1.8 부터는 Default Abstract Method를 정의할 수 있습니다.

# 17. interface

---

- 인터페이스를 만든 후 구현 클래스를 만들어 줍니다.
- 인터페이스는 직접 인스턴스화할 수 없으므로 구현 클래스가 반드시 필요합니다.
- 인터페이스를 구현할 때엔, implements 키워드를 사용합니다.

```
public class SomeClass implements SomeInterface {
}
```

- SomeInterface 인터페이스를 구현해 SomeClass 를 만듭니다.
- 인터페이스의 메소드는 SomeClass에서 반드시 구현을 해주어야 합니다.

# 17. interface

---

- SomeClass에 SomeInterface를 구현합니다.

```
public class SomeClass implements SomeInterface {
 @Override
 public void doSomething1() { ... }

 @Override
 public void doSomething2() { ... }

 @Override
 public void doSomething3() { ... }

 @Override
 public int getSomething() {
 return 0;
 }

 @Override
 public String getSomething2() {
 return null;
 }
}
```

# 17. interface

- 이제 SomeInterface와 SomeClass가 완성되었습니다.
- 추상클래스와 똑같이 인터페이스와 구현클래스에도 is a 관계가 성립합니다.
  - SomeClass is a SomeInterface
- 따라서 아래와 같이 인스턴스 정의가 가능합니다.

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

 SomeInterface some = new SomeClass();
 some.doSomething1();
 some.doSomething2();
 some.doSomething3();

 int something = some.getSomething();
 System.out.println(something);

 String something2 = some.getSomething2();
 System.out.println(something2);
}
```

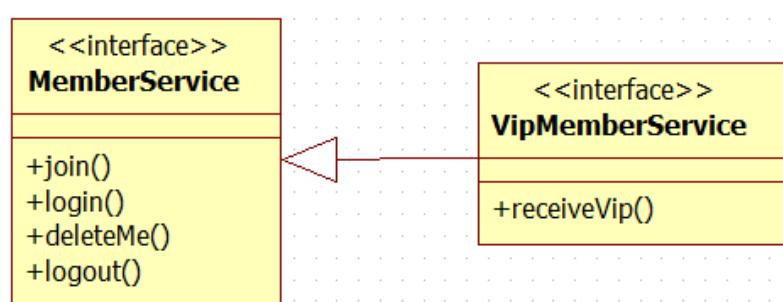
# 17. interface

---

- 추상 클래스나 인터페이스는 다형성을 제공합니다.
- 즉, 하나의 추상 클래스 또는 인터페이스를 통해 여러 개의 구현 클래스가 만들어질 수 있습니다.
- SomeInterface를 이용해 새로운 구현 클래스를 만들어보세요.

# 17. interface

- 인터페이스는 클래스처럼 상속이 가능합니다.
- 인터페이스는 클래스가 구현 해야 할 기능을 정의하는 것입니다.
- 만약, 아래와 같은 기능을 가진 회원 서비스를 개발하는 중이라고 가정합니다.
  - 1. 회원 가입
  - 2. 로그인
  - 3. 탈퇴
  - 4. 로그아웃
  - 5. VIP 혜택 받기
- 일반회원은 VIP 혜택을 받을 수 없고, 오로지 VIP 회원만 혜택을 받을 수 있다면 이렇게 구분 시킬 수 있습니다.



# 17. interface

---

- MemberService와 VipMemberService Interface를 만들어봅니다.
- MemberService.java

```
public interface MemberService {
 public void join();
 public void login();
 public void deleteMe();
 public void logout();
}
```

- VipMemberService.java

```
public interface VipMemberService extends MemberService {
 public void receiveVip();
}
```

- VipMemberService is a MemberService

# 17. interface

---

- 이제 MemberService 인터페이스를 구현한 클래스와 VipMemberService 인터페이스를 구현한 클래스를 만들 수 있습니다.
- NormalMemberService.java

```
public class NormalMemberService implements MemberService {
 @Override
 public void join() { ... }

 @Override
 public void login() { ... }

 @Override
 public void deleteMe() { ... }

 @Override
 public void logout() { ... }
}
```

# 17. interface

---

- 이제 MemberService 인터페이스를 구현한 클래스와 VipMemberService 인터페이스를 구현한 클래스를 만들 수 있습니다.
- VipClassMemberService.java

```
public class VipClassMemberService implements VipMemberService {
 @Override
 public void join() { ... }

 @Override
 public void login() { ... }

 @Override
 public void deleteMe() { ... }

 @Override
 public void logout() { ... }

 @Override
 public void receiveVip() { ... }
}
```

# 17. interface

---

- VipClassMemberService is a VipMemberService
- NormalMemberService is a MemberService
- VipMemberService is a MemberService

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

 MemberService normalClass = new NormalMemberService();
 MemberService vipClass1 = new VipClassMemberService();
 VipMemberService vipClass2 = new VipClassMemberService();

 normalClass.deleteMe();
 vipClass1.join();
 vipClass2.receiveVip();

}
```

# 17. interface

---

- VipClassMemberService 코드 줄여보기
- VipMemberService 는 MemberService 를 상속받은 인터페이스
- NormalMemberService는 MemberService 인터페이스를 구현한 클래스
- MemberService 의 미 구현 메소드는 NormalMemberService를 상속받아 구현

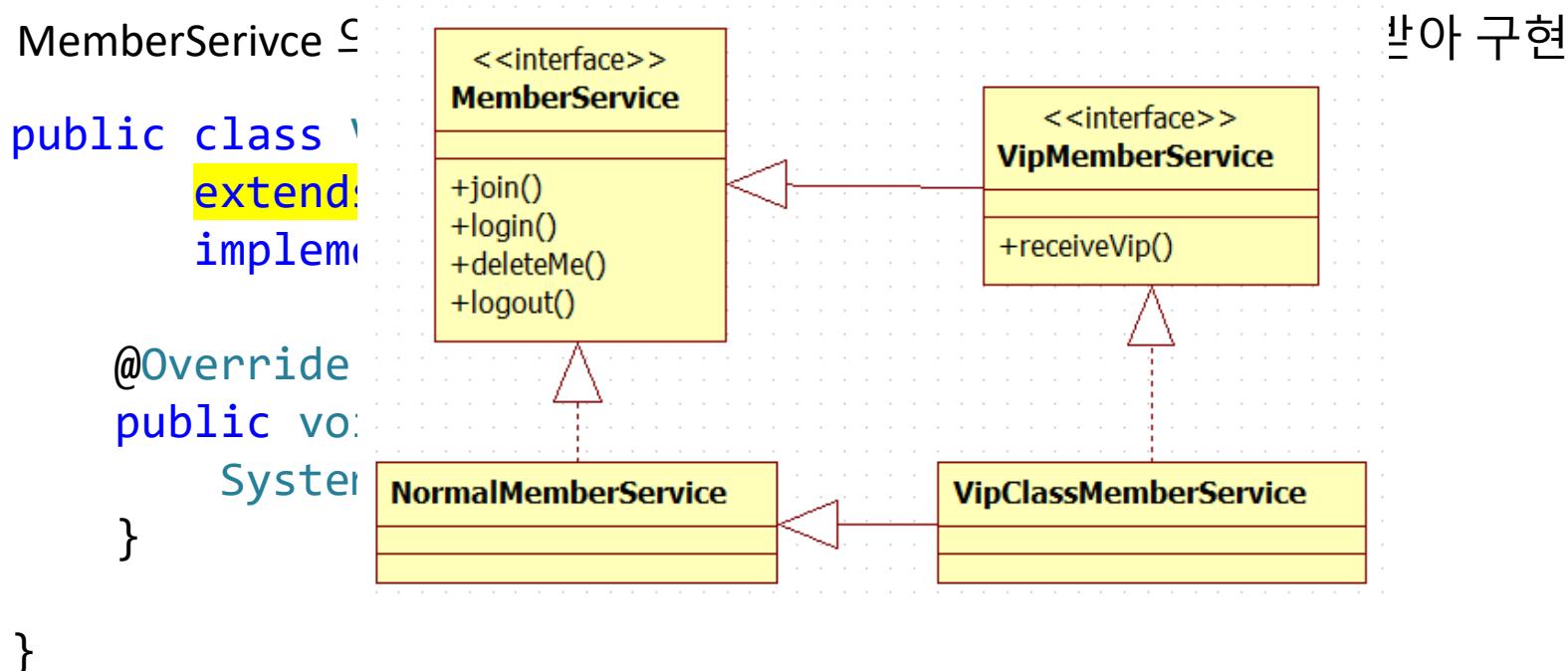
```
public class VipClassMemberService
 extends NormalMemberService
 implements VipMemberService {

 @Override
 public void receiveVip() {
 System.out.println("혜택 받아요");
 }

}
```

# 17. interface

- VipClassMemberService 코드 줄여보기
- VipMemberService 는 MemberService 를 상속받은 인터페이스
- NormalMemberService는 MemberService 인터페이스를 구현한 클래스
- MemberService ↳ 만아 구현



# 17. interface

- 다시 Vehicle 코드를 살펴보면, 조금 거슬리는 부분이 있습니다.

```
public class App {
 public static void main(String[] args) {
 Vehicle car2 = new SportsCar("");
 Vehicle car3 = new EV("", 50);
 Vehicle car4 = new BatMobile("");

 car4.startEngine();
 car3.checkBattery();
 car2.startTurbo();
 car2.checkBattery(); ← SportsCar에 지원되지 않는 기능이
 호출되는 것 자체가 거슬립니다.
 }
}
```

# 17. interface

---

- 그 뿐만 아니라 새로운 자동차 클래스를 만들 때마다 Vehicle을 수정해주어야 하며, Vehicle을 상속받고 있는 모든 클래스들도 수정을 해 주어야 합니다. (OCP 위반)

```
public sealed abstract class Vehicle permits SportsCar, EV{
 private String name;

 ... 생략 ...

 public abstract void startTurbo();
 public abstract void extractBatPort();
 public abstract void checkBattery();
}
```

- Interface는 이런 문제를 해결할 때 사용되며, Java Application을 만들 때 아주 많이 사용되는 Java 타입 중 하나입니다.

# 17. interface

- 인터페이스를 이용해 startTurbo, extractBatMobile, checkBattery 메소드들을 모두 분리해봅니다. (ISP)
- 먼저 startEngine을 담당할 인터페이스 부터 만듭니다.

```
public interface Rideable {
 public void startEngine();
}
```

- 다음은 startTurbo를 담당할 인터페이스를 만듭니다.

```
public interface UsingTurboEngine extends Rideable {
 public void startTurbo();
}
```

## 인터페이스도 추상메소드를 만들 수 있습니다.

추상클래스는 일반 메소드와 추상 메소드를 만들 수 있는 반면  
인터페이스는 오로지 추상 메소드와 상수만 만들 수 있습니다. 이 점이 다릅니다.  
\* Java 1.8 부터 default method로 일반 메소드도 만들 수 있습니다. (단, 사용에 제한이 있습니다.)

# 17. interface

---

- 다음은 extractBatMobile을 담당할 인터페이스를 만듭니다.

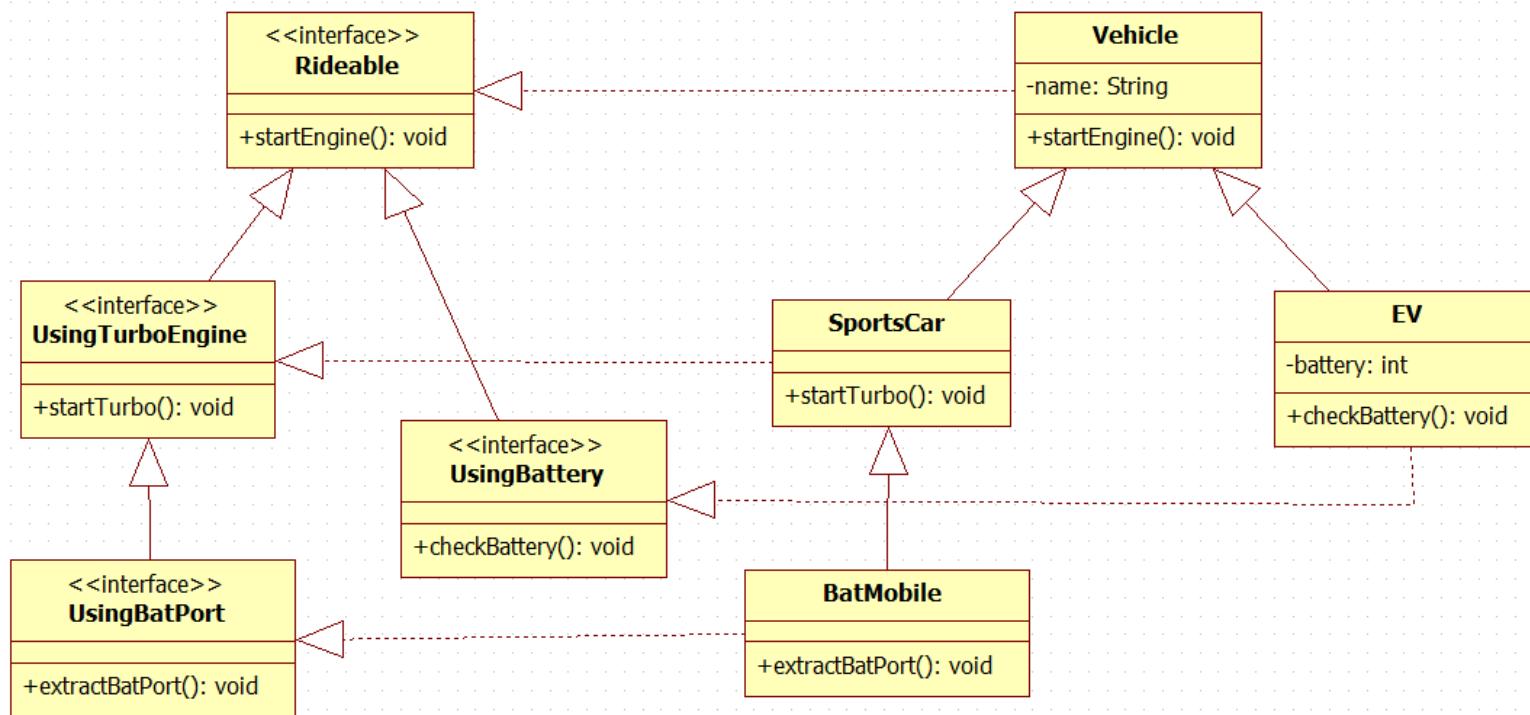
```
public interface UsingBatPort extends UsingTurboEngine {
 public void extractBatPort();
}
```

- 마지막으로 checkBattery를 담당할 인터페이스를 만듭니다.

```
public interface UsingBattery extends Rideable {
 public void checkBattery();
}
```

# 17. interface

- 인터페이스의 상속구조와 클래스들의 구현 예정 구조입니다.



# 17. interface

- Vehicle 클래스는 Rideable 인터페이스를 구현합니다.

```
public sealed class Vehicle implements Rideable permits SportsCar, EV {
 private String name;

 public Vehicle(String name) {
 this.name = name;
 }

 @Override
 public final void startEngine() {
 System.out.println(name + " 시동을 겁니다.");
 }
}
```

## 인터페이스 구현

인터페이스 내부의 추상 메소드를 오버라이딩하는 것을 “인터페이스를 구현한다”라고 표현합니다.  
자주 사용되는 표현입니다.  
인터페이스 구현은 implements 키워드를 사용합니다.

# 17. interface

---

- SportsCar 클래스는 UsingTurboEngine 인터페이스를 구현합니다.

```
public sealed class SportsCar extends Vehicle implements UsingTurboEngine
permits BatMobile {

 public SportsCar(String name) {
 super(name);
 }

 @Override
 public void startTurbo() {
 System.out.println("터보 모드를 시작합니다!");
 }
}
```

# 17. interface

- BatMobile 클래스는 UsingBatPort 인터페이스를 구현합니다.

```
public non-sealed class BatMobile extends SportsCar implements UsingBatPort {

 public BatMobile(String name) {
 super(name);
 }

 @Override
 public void extractBatPort() {
 System.out.println("배트포트를 분리합니다.");
 }
}
```

# 17. interface

---

- EV 클래스는 UsingBattery 인터페이스를 구현합니다.

```
public final class EV extends Vehicle implements UsingBattery {

 public int battery;

 public EV(String name, int battery) {
 super(name);
 this.battery = battery;
 }

 @Override
 public void checkBattery() {
 System.out.println("배터리가 " + this.battery + " 만큼 남아있습니다.");
 }
}
```

# 17. interface

---

- 이제 인터페이스 기반의 is a 관계가 만들어졌으니 테스트를 해봅니다.

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

 Rideable car1 = new Vehicle("");
 UsingTurboEngine car2 = new SportsCar("");
 UsingBattery car3 = new EV("", 50);
 UsingBatPort car4 = new BatMobile("");

 car1.startEngine();
 car2.startEngine();
 car2.startTurbo();
 car3.startEngine();
 car3.checkBattery();
 car4.startEngine();
 car4.startTurbo();
 car4.extractBatPort();
 }
}
```

# 17. interface

---

- Is a 관계 확인을 해봅니다.

```
Rideable car1 = new Vehicle("");
UsingTurboEngine car2 = new SportsCar("");
UsingBattery car3 = new EV("", 50);
UsingBatPort car4 = new BatMobile("");
```

```
System.out.println(car4 instanceof BatMobile);
System.out.println(car4 instanceof SportsCar);
System.out.println(car4 instanceof Vehicle);
System.out.println(car4 instanceof Rideable);
System.out.println(car4 instanceof UsingTurboEngine);
System.out.println(car4 instanceof UsingBattery);
System.out.println(car4 instanceof UsingBatPort);
```

# 17. interface

---

- 게시판 인터페이스를 만드려면 어떤 기능이 필요할까요?
  - 게시판은 아래 기능들이 필요합니다.
    - 1. 게시글 목록 조회
    - 2. 게시글 1개 내용 조회
    - 3. 게시글 작성
    - 4. 게시글 수정
    - 5. 게시글 삭제

```
public interface Articles {

 public void readAllArticles();
 public void readOneArticle();
 public void createArticle();
 public void modifyArticle();
 public void deleteArticle();

}
```

# 17. interface

---

- 주소록 인터페이스를 만드려면 어떤 기능이 필요할까요?
  - 필요한 기능을 정의해 인터페이스로 만들어보고 구현 클래스를 만들어보세요.

대충 이런 기능을 제공할거에요.  
필요하면 가져다 쓰세요.

## 17. 추상화

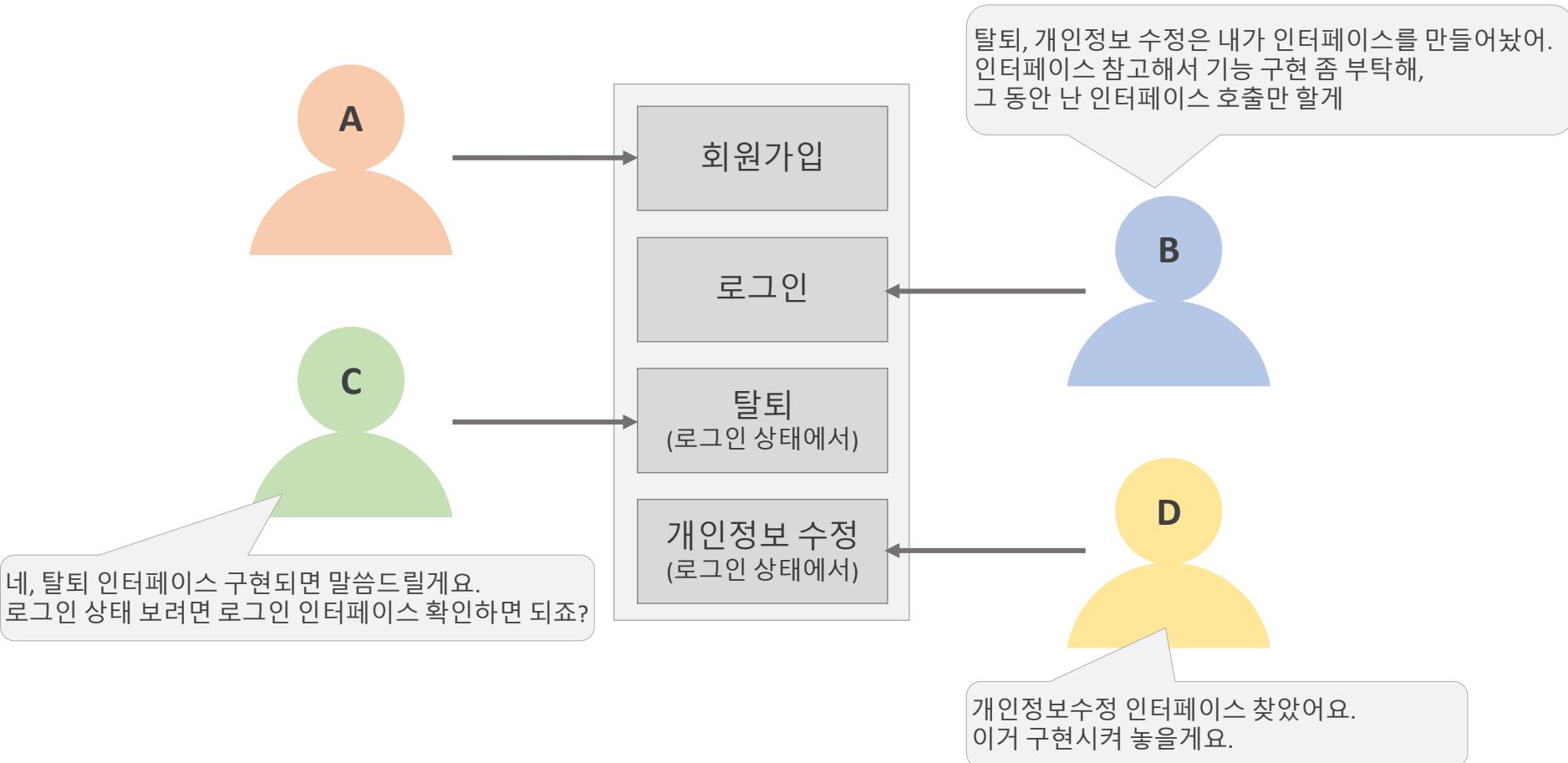
# 17. 추상화

---

- **추상화**(abstraction)는 복잡한 자료, 모듈, 시스템 등으로부터 핵심적인 개념 또는 기능을 간추려 내는 것을 말한다.
  - 출처: [https://ko.wikipedia.org/wiki/추상화\\_\(컴퓨터\\_과학\)](https://ko.wikipedia.org/wiki/추상화_(컴퓨터_과학))
- 추상화란.
  - 1. 제공해야하는 기능이 구현되지 않은 정의된 상태.
  - 2. 기능이 구현되어있지 않기 때문에 인스턴스화 할 수 없음
  - 3. 추상 클래스나 인터페이스를 구현하는 경우 추상 메소드를 반드시 구현해야 함
  - 4. 추상 메소드를 구현할 때 구조를 변경할 수 없음
  - 5. 따라서 추상화란, 기능에 대한 표준이며 규약

# 17. 추상화

- 예를 들어 아래와 같은 기능을 가진 애플리케이션을 만든다면



# 17. 추상화

---

- 추상화의 이점
  - 1. 인터페이스나 추상클래스만으로 코드 작성이 가능합니다.
  - 2. 하나의 인터페이스나 추상클래스로 여러 개의 구현 클래스를 만들 수 있고(다형성)
  - 3. 여러 개의 구현 클래스를 이용해 유연한 상황대처가 가능합니다.

# 17. 추상화

---

- 추상 클래스와 인터페이스의 차이
  - 추상클래스와 인터페이스는 추상 메소드를 만든다는 점에서 동일.
  - 일반적으로 추상 메소드의 비율로 추상화 정도를 나타냄
- 추상클래스 – class
  - 추상 메소드와 일반 메소드가 혼재할 가능성이 매우 높음
  - 비 완전 추상화
- 인터페이스 – interface
  - 추상 메소드만 존재함.
  - 완전 추상화
- 인터페이스가 추상클래스보다 추상화의 정도가 높음.

# 17. 추상화

---

- 추상클래스의 활용
  - 기존 코드를 확장하려 할 때
    - 기존의 코드는 수정하지 않고 추상 클래스를 활용해 코드 전 후에 해야 할 일을 작성.
- 인터페이스의 활용
  - 제공되는 기능에 대해서는 알고 있지만 환경에 따라 기능의 구현이 변경되어야 할 때
    - 환경:
      - OS, Database, 사용자 입력 값 등 여러 요인이 존재

# 17. 악명 클래스

# 17. 익명 클래스

---

- 익명 클래스란
  - 추상 클래스나 인터페이스를 강제로 인스턴스화 시키는 방법.
- 추상클래스나 인터페이스를 인스턴스화 할 수 없는 이유
  - 구현되지 않은 추상 메소드가 있기 때문입니다.
  - 인스턴스화의 조건은 모든 메소드가 구현되어 있어야 합니다.
- 추상클래스나 인터페이스를 강제로 인스턴스화 하려면
  - 인스턴스를 만들 때 추상클래스를 구현 시켜주면 됩니다.

# 17. 익명 클래스

- Rideable 인터페이스를 강제 인스턴스화 하기

```
public class App {
 public static void main(String[] args) throws Exception {

 Rideable car1 = new Rideable() {
 // 즉시 구현
 @Override
 public void startEngine() {
 System.out.println("시동을 걸었습니다!");
 }
 };

 car1.startEngine();
 }
}
```

# 17. 익명 클래스

- UsingTurboEngine 인터페이스를 강제 인스턴스화 하기

```
public class App {
 public static void main(String[] args) throws Exception {
 UsingTurboEngine car2 = new UsingTurboEngine() {
 @Override
 public void startEngine() {
 System.out.println("시동을 걸었습니다!");
 }

 @Override
 public void startTurbo() {
 System.out.println("터보모드를 시작합니다!");
 }
 };

 car2.startEngine();
 car2.startTurbo();
 }
}
```

# 17. 익명 클래스

---

- 익명 클래스는 언제 사용하면 좋을까요?
  - 단 한 번 사용할 인스턴스를 만들 때 사용합니다.
- 추상 클래스 또는 인터페이스의 구현체를 생성하는데,  
딱 한번 사용할 인스턴스를 만들기 위해 클래스를 정의하는 것은  
매우 비효율적입니다.
  - 이런 경우 클래스를 정의하지 않고 즉시 구현하여 인스턴스를 만드는 익명클래스를  
만드는 것이 유리합니다.

# 17. 익명 클래스

- 인터페이스를 파라미터로 받는 메소드가 있다고 가정합니다.

```
public class App {

 public void startEngine(Rideable rideable) {
 rideable.startEngine();
 }

 public static void main(String[] args) {
 App app = new App();

 }
}
```

- App의 startEngine을 호출하기 위해 파라미터로 클래스를 정의해 보낼 수도 있지만, 익명클래스를 보낼 수도 있습니다.

# 17. 익명 클래스

- startEngine 메소드를 호출하면서 익명클래스를 전달해 봅니다.

```
public class App {

 public void startEngine(Rideable rideable) {
 rideable.startEngine();
 }

 public static void main(String[] args) {
 App app = new App();
 app.startEngine(new Rideable() {
 @Override
 public void startEngine() {
 System.out.println("일회용 차의 시동을 겁니다!");
 }
 });
 }
}
```

# 17. 익명 클래스

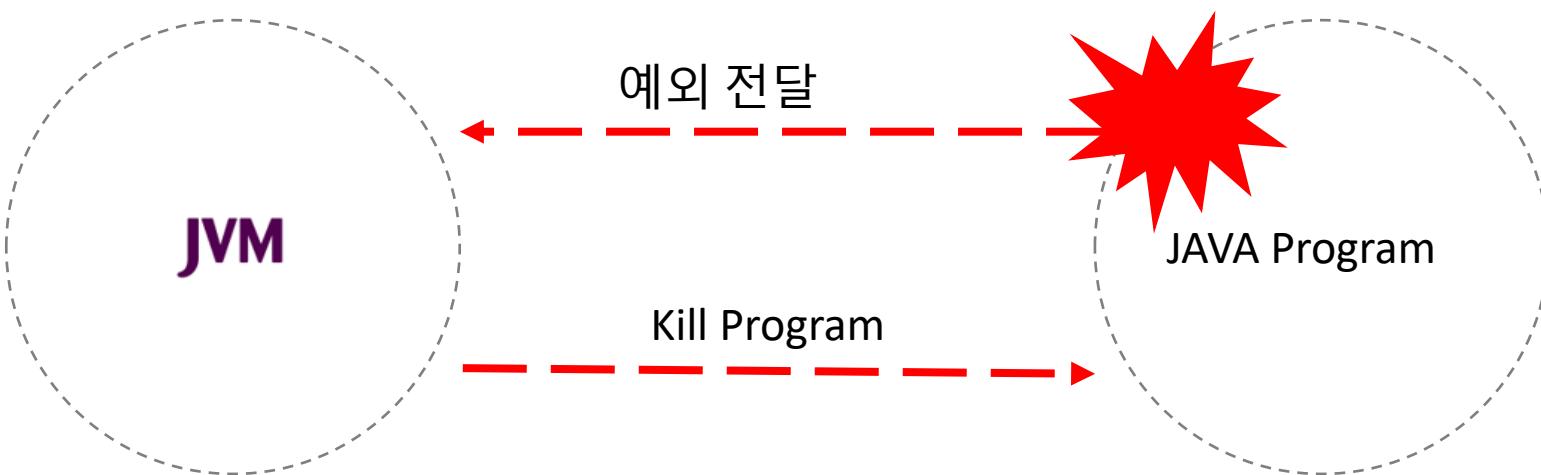
---

- 주소록 인터페이스로 익명 클래스를 만들어보세요.

# 18. 예외처리

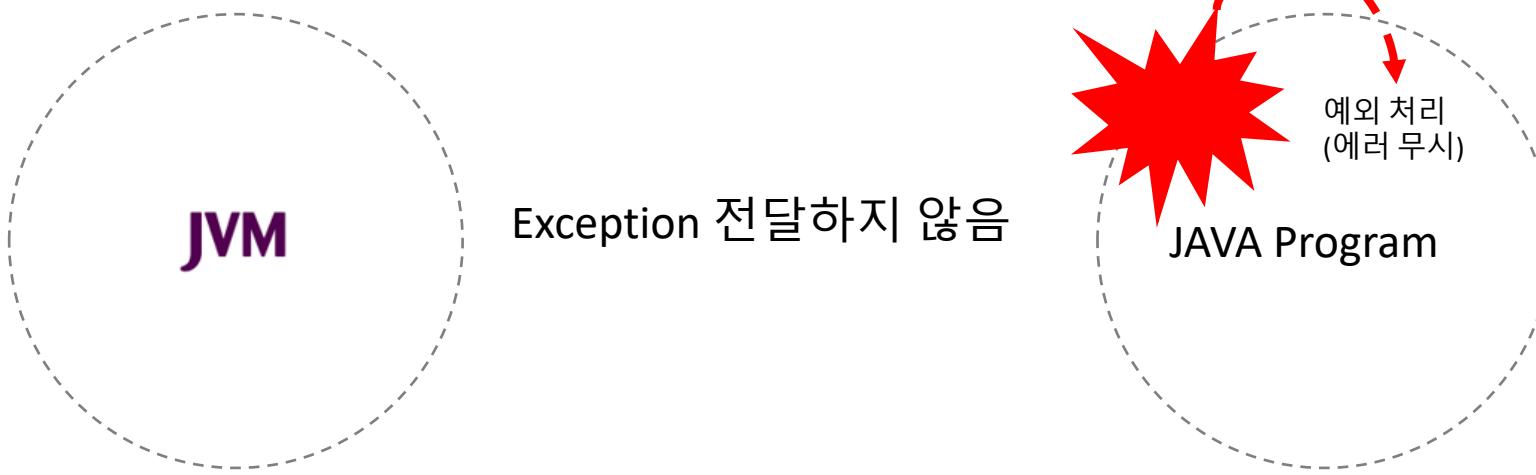
# 18. 예외처리

- 프로그램이 동작하는 과정 중 잘못된 입력 값이 전달되거나
  - 읽어야 하는 파일이 존재하지 않거나
  - 배열의 인덱스를 잘못 전달했거나
  - Null 인스턴스에 접근을 하려했거나 등등의 이유로 **오류**가 발생하는데
  - 이것을 “**예외**” 라고 합니다.
- 
- 예외가 발생한 경우 Java 프로그램은 JVM에 의해서 강제 종료됩니다.



# 18. 예외처리

- 예외 처리는 예외가 발생하더라도 프로그램이 종료되지 않도록 하는 방법이면서
- 예외가 발생하지 않도록 하는 방어코딩의 방법입니다.



- 예외 처리는 모든 언어에서 굉장히 중요하게 다루는 개념입니다.
- 반드시 알아두어야 합니다.

# 18. 예외처리

---

- Java에서 가장 많이 발생하는 예외 목록
  - 1. **NullPointerException**
    - Null Instance에 접근하려 함
  - 2. **ArrayIndexOutOfBoundsException**
    - List의 인덱스 범위를 벗어남
  - 3. **IndexOutOfBoundsException**
    - 배열의 인덱스 범위를 벗어남
  - 4. **NumberFormatException**
    - 숫자가 아닌 문자를 숫자로 변환하려 함

# 18. if를 활용한 예외처리

# 18. if를 활용한 예외처리

---

- 예외를 처리하는 가장 좋은 방법은 예외를 발생시키지 않는 것입니다.
- 아래 예외 목록은 if를 활용해 예외를 방지할 수 있습니다.
  - 1. NullPointerException
    - Null Instance에 접근하려 함
    - Instance가 Null인지 먼저 확인 함
  - 2. ArrayIndexOutOfBoundsException
    - List의 인덱스 범위를 벗어남
    - 찾으려는 인덱스가 존재하는지 먼저 확인 함
  - 3. IndexOutOfBoundsException
    - 배열의 인덱스 범위를 벗어남
    - 찾으려는 인덱스가 존재하는지 먼저 확인 함
  - 4. NumberFormatException
    - 숫자가 아닌 문자를 숫자로 변환하려 함
    - 문자가 숫자형태인지 확인 함
- 이렇게 먼저 확인하는 코드를 “Validation Logic” 이라고 하며 “입력값 검증” 이라고 표현하기도 합니다.

# 18. if를 활용한 예외처리

---

- NullPointerException이 발생하는 원인

```
public static void main(String[] args) {
 String name = null;

 // NullPointerException - null.equals("이름")
 System.out.println(name.equals("이름"));
}
```

- Null은 Reference Type의 기본 값으로 아무것도 할당되지 않은 상태를 말합니다.
- 할당된 것이 없으니 메모리에 아무것도 없는 상태이며, 존재하지 않는 메모리를 참조하려 한다는 에러가 발생합니다.

# 18. if를 활용한 예외처리

---

- NullPointerException 방지 방법

```
public static void main(String[] args) {
 String name = null;

 if (name != null) {
 System.out.println(name.equals("이름"));
 }
}
```

- 인스턴스를 참조하기 전, null 인지 확인합니다.
- Name 변수의 값은 null 이므로  
`System.out.println(name.equals("이름"));` 이 코드는 실행되지 않습니다.

# 18. if를 활용한 예외처리

---

- (Array)IndexOutOfBoundsException이 발생하는 원인

```
public static void main(String[] args) {
 // 배열의 간단한 정의
 int[] scores = {100, 200, 300};

 System.out.println(scores[0]); //100
 System.out.println(scores[1]); //200
 System.out.println(scores[2]); //300
 System.out.println(scores[3]); //ArrayIndexOutOfBoundsException
}
```

- scores 배열은 0, 1, 2 인덱스만 가지고 있는데 3 인덱스를 참조하려 했습니다.

# 18. if를 활용한 예외처리

- (Array)IndexOutOfBoundsException 방지방법 - 참조 전 배열의 길이 계산

```
public static void main(String[] args) {
 // 배열의 간단한 정의
 int[] scores = {100, 200, 300};
 if (scores.length > 0) {
 System.out.println(scores[0]); // 100
 }
 if (scores.length > 1) {
 System.out.println(scores[1]); // 200
 }
 if (scores.length > 2) {
 System.out.println(scores[2]); // 300
 }
 if (scores.length > 3) {
 System.out.println(scores[3]);
 }
}
```

# 18. if를 활용한 예외처리

---

- NumberFormatException이 발생하는 원인

```
public static void main(String[] args) {
 String numString = "123";
 int num = Integer.parseInt(numString);
 System.out.println(num); // 123

 numString = "ABC";
 num = Integer.parseInt(numString); // NumberFormatException
 System.out.println(num);
}
```

- 숫자 형태가 아닌 값을 숫자로 변경하려 했습니다.

# 18. if를 활용한 예외처리

- NumberFormatException 방지방법

```
public static void main(String[] args) {
 String numString = "123";
 if (numString.matches("^\\d+$")) {
 int num = Integer.parseInt(numString);
 System.out.println(num); // 123
 }

 numString = "ABC";
 if (numString.matches("^\\d+$")) {
 int num = Integer.parseInt(numString);
 System.out.println(num);
 }
}
```

- 변환 전 숫자의 형태를 체크합니다.

# 18. try – catch – finally 예외처리

# 18. try – catch – finally 예외처리

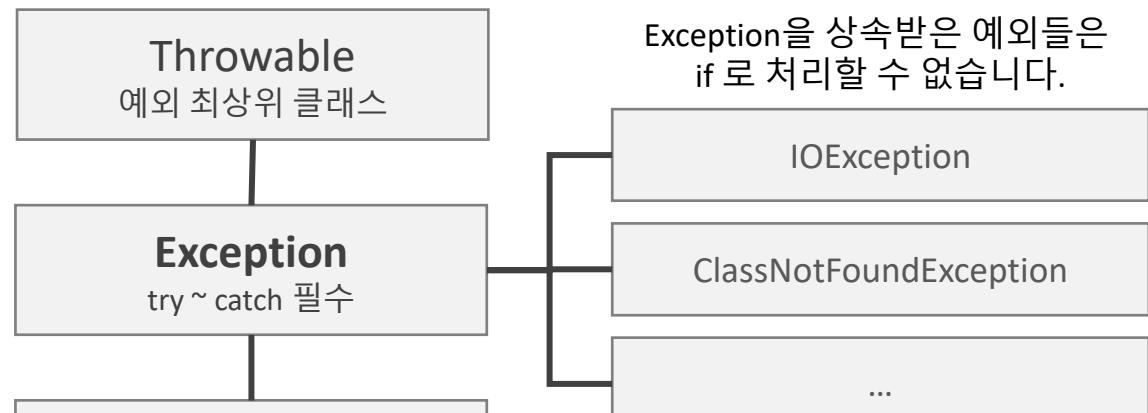
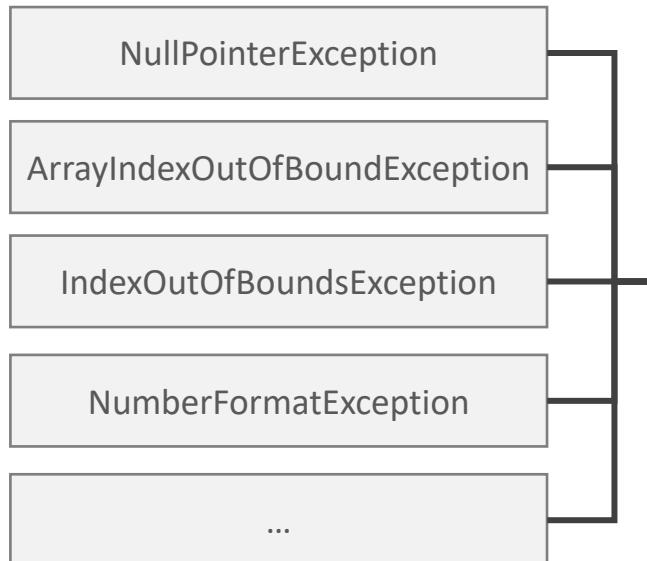
- if로 예외 처리할 수 없거나 애매할 경우 try ~ catch ~ finally를 사용합니다.

```
try {
 // ▼ NumberFormatException 발생
 int number = Integer.parseInt("ABC");
 System.out.println(number); // 이 코드는 실행되지 않습니다.
}
// ▼ NumberFormatException이 발생했을 때 catch가 실행됩니다.
catch (NumberFormatException e) {
 // try에서 예외가 발생하면 catch가 실행됩니다.
 System.out.println("에러가 발생했습니다. " + e.getMessage());
 e.printStackTrace(); // 에러내용 모두 출력
}
finally {
 // 예외 발생 여부와 관계없이 무조건 실행됩니다.
 System.out.println("처리가 완료됐습니다.");
}
```

# 18. try – catch – finally 예외처리

- if로 예외 처리할 수 없는 경우?
- Java의 예외 종류는 크게 두 종류가 있습니다.

RuntimeException을 상속받은 예외들은  
if로 처리할 수 있습니다.



# 18. try – catch – finally 예외처리

---

- If로 처리할 수 없는 예외들은 try ~ catch ~ finally를 사용합니다.
- 아래는 클래스를 동적 로딩하는 코드입니다.
- SomeAClass라는 클래스를 불러오게 되는데 SomeAClass라는 클래스는 없으니 에러(ClassNotFoundException)가 발생하게 됩니다.

```
Class.forName("SomeAClass");
```

- forName을 Ctrl을 누른채로 클릭해보면 다음과 같은 코드를 볼 수 있습니다.

```
public static Class<?> forName(String className)
 throws ClassNotFoundException {
 Class<?> caller = Reflection.getCallerClass();
 return forName(className, caller);
}
```

# 18. try – catch – finally 예외처리

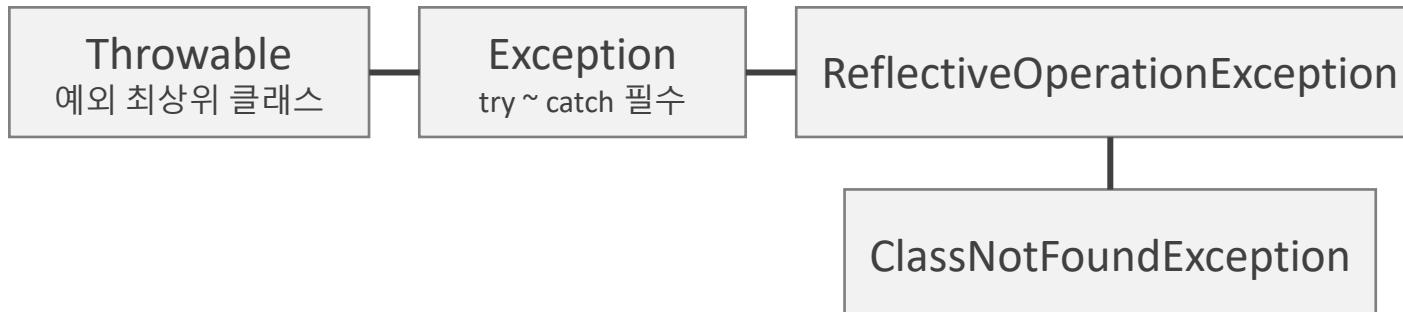
- ClassNotFoundException을 throws하고 있습니다. ClassNotFoundException을 Ctrl + 클릭해보면 ReflectiveOperationException을 상속받은 것을 알 수 있습니다.

```
public class ClassNotFoundException
 extends ReflectiveOperationException {
```

- 다시 ReflectiveOperationException을 Ctrl + 클릭해봅니다.

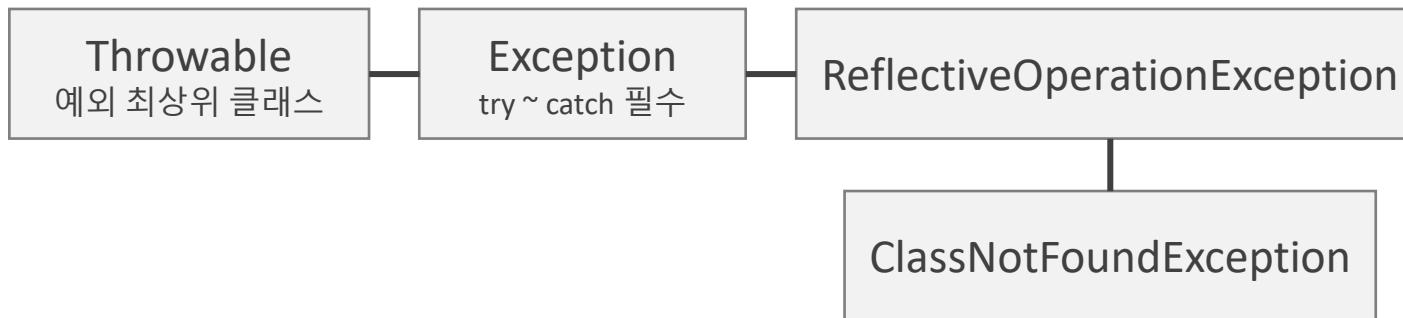
```
public class ReflectiveOperationException extends Exception {
```

- 결국 ReflectiveOperationException은 Exception을 상속받은 예외클래스라는 것을 알 수 있습니다.



# 18. try – catch – finally 예외처리

- 이 상속관계를 is a 관계로 정리해봅니다.



- **ClassNotFoundException** is a **ReflectiveOperationException**
- **ReflectiveOperationException** is a **Exception**
- **ClassNotFoundException** is a **Exception**
- **ClassNotFoundException**은 **Exception**을 상속받은 예외 클래스이기 때문에 try ~ catch가 반드시 필요합니다.

# 18. try – catch – finally 예외처리

- 이 코드를 실행해보면 다음과 같은 에러가 발생합니다.

```
public static void main(String[] args) {
 Class.forName("SomeAClass");
}
```

Unhandled exception type ClassNotFoundException

- ClassNotFoundException 타입의 예외를 처리하지 않았다라는 에러입니다.
- 이 에러를 처리하기 위해 try ~ catch를 사용해봅니다.

```
public static void main(String[] args) {
 try {
 Class.forName("SomeAClass");
 }
 catch(ClassNotFoundException cnfe) {
 cnfe.printStackTrace();
 }
}
```

# 18. try – catch – finally 예외처리

---

- cnfe.printStackTrace(); 코드에 의해 아래처럼 메시지가 나타나게 됩니다.

```
java.lang.ClassNotFoundException: SomeAClass
 at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
 at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
 at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:521)
 at java.base/java.lang.Class.forName0(Native Method)
 at java.base/java.lang.Class.forName(Class.java:391)
 at java.base/java.lang.Class.forName(Class.java:382)
 at App.main(App.java:4)
```

- Exception이 발생한 최초의 위치와 호출스택들이 한번에 노출되게 되며  
가상 상위에 Exception이 발생한 원인에 대해 설명해줍니다.

# 18. try – catch – finally 예외처리

- 예외처리는 try 내에서 발생한 Exception 을 처리합니다.
- 하나의 try에서 여러 Exception을 발생시킨다면 catch를 여러 번 사용합니다.

```
public static void main(String[] args) {
 String type = null;
 String number = "abc";

 try {
 if (type.equals("number")) {
 int num = Integer.parseInt(number);
 System.out.println(num);
 }
 }
 catch (NullPointerException npe) {
 npe.printStackTrace();
 }
 catch (NumberFormatException nfe) {
 nfe.printStackTrace();
 }
}
```

# 18. try – catch – finally 예외처리

- 여러 Exception에 대한 처리를 똑같이 해주고 있다면 Exception을 하나로 묶어 처리하는 것이 유용합니다.

```
public static void main(String[] args) {
 String type = null;
 String number = "abc";

 try {
 if (type.equals("number")) {
 int num = Integer.parseInt(number);
 System.out.println(num);
 }
 }
 catch (NullPointerException | NumberFormatException e) {
 e.printStackTrace();
 }
}

finally
```

현재 실습 코드는 RuntimeException을 catch하고 있습니다. 일반적으로 RuntimeException은 if로만 처리합니다. Exception과 관련된 코드들은 File I/O에서 다루고 있으므로 지금은 문법만 익히도록 합니다. finally 또한 File I/O에서 다룹니다.

# **18. 사용자 예외 발생시키기**

# 18. 사용자 예외 발생시키기

---

- Java에서 제공하는 내장 Exception은 Java Application이 동작할 때, 코드의 조건을 만족하지 못했을 때 지정된 Exception이 발생하도록 되어 있습니다.
- 몇 가지 코드의 조건을 살펴보면.
  - Null 값을 가진 Instance에 점 연산자를 사용함 → NullPointerException
  - 숫자 형태가 아닌 문자를 숫자로 변환하려 함 → NumberFormatException
  - 존재하지 않는 파일을 읽으려 함 → FileNotFoundException
  - 0 으로 나누려 함 → ArithmaticException
  - 배열의 인덱스 범위를 초과하여 참조하려 함 → ArrayIndexOutOfBoundsException
- 개발자가 직접 조건을 정의하고 지정된 Exception을 발생시킬 수도 있는데 이것은 사용자 예외 혹은 커스텀 예외라고 합니다.
  - 예>
  - 사용할 수 없는 ID 입니다. → InvalidUserIdException
  - 이미 사용 중인 ID 입니다. → AlreadyUseUserIdException
  - ID는 최소 8글자 이상 입력하세요. → UserIdLengthException
  - 비밀번호는 영어(대/소문자 포함), 숫자, 특수기호를 사용해 10자리 이상 입력하세요.
    - → PasswordFormatException
  - 등

# 18. 사용자 예외 발생시키기

---

- 사용자 예외는 항상 RuntimeException을 상속받아 생성합니다.

```
public class InvalidUserIdException extends RuntimeException {
}
```

- 생성자가 존재하는 클래스를 상속 받으면 Sub Class에서 Super Class의 생성자를 호출해주어야 합니다.
- RuntimeException의 생성자 목록을 확인해 봅니다.

# 18. 사용자 예외 발생시키기

- RuntimeException의 오버로딩된 생성자들

```
public class RuntimeException extends Exception {
 public RuntimeException() {
 super();
 }
 public RuntimeException(String message) {
 super(message);
 }
 public RuntimeException(String message, Throwable cause) {
 super(message, cause);
 }
 public RuntimeException(Throwable cause) {
 super(cause);
 }
 protected RuntimeException(String message, Throwable cause,
 boolean enableSuppression,
 boolean writableStackTrace) {
 super(message, cause, enableSuppression, writableStackTrace);
 }
}
```

# 18. 사용자 예외 발생시키기

---

- RuntimeException의 기본 생성자

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

- 예외를 발생시킬 때, 아무런 값도 전달하지 않습니다.
- 이 생성자를 사용할 경우, 예외를 받았을 때 아무런 정보를 얻을 수 없습니다.
- 잘 사용하지 않습니다.

# 18. 사용자 예외 발생시키기

---

- RuntimeException의 생성자들

```
public RuntimeException(String message) {
 super(message);
}
```

- 예외를 발생시킬 때, 원인이 된 메시지를 파라미터로 전달합니다.
- 가장 많이 사용합니다.

# 18. 사용자 예외 발생시키기

---

- RuntimeException의 생성자들

```
public RuntimeException(String message, Throwable cause) {
 super(message, cause);
}
```

- Throwable은 모든 예외들의 Root Class 입니다.
- RuntimeException is a Throwable  
Exception is a Throwable  
즉, 모든 예외타입은 is a Throwable 타입입니다.
- 보통 try ~ catch에서 처리한 Exception을 RuntimeException으로 변경하여  
다시 발생시킬 때 주로 사용합니다.
- Message는 예외를 발생시킨 원인이 된 메시지를 전달하며
- Cause는 catch에서 처리한 예외를 전달합니다.  
즉, RuntimeException이 발생된 원인이 됩니다.
- `public RuntimeException(String message) {}` 와 함께 많이 사용됩니다.

# 18. 사용자 예외 발생시키기

---

- RuntimeException의 생성자들

```
public RuntimeException(Throwable cause) {
 super(cause);
}
```

- 예외를 발생시킬 때, 원인이 된 예외를 파라미터로 전달합니다.
- 보통 try ~ catch 문 중 catch에서 발생된 예외를 RuntimeException으로 변경하여 발생시킬 때 사용합니다.
- 잘 사용하지 않습니다.

# 18. 사용자 예외 발생시키기

---

- 다시 InvalidUserIdException을 완성해봅니다.

```
public class InvalidUserIdException extends RuntimeException {

 public InvalidUserIdException(String message) {
 super(message);
 }

 public InvalidUserIdException(String message, Throwable cause) {
 super(message, cause);
 }

}
```

- 사용자 예외는 거창할 필요없이 필요한 생성자만 호출 해주면 됩니다.

# 18. 사용자 예외 발생시키기

- InvalidUserIdException을 발생시켜 봅니다.
- id가 root, admin, system 인 경우 위 예외를 발생시키는 코드입니다.

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

 Scanner input = new Scanner(System.in);

 System.out.println("ID를 입력하세요.");
 String userId = input.nextLine();

 if (userId.equals("root") || userId.equals("admin")
 || userId.equals("system")) {
 InvalidUserIdException iuie =
 new InvalidUserIdException(userId + "는 사용할 수 없습니다.");
 throw iuie;
 }
}
```

- throw Exception 객체 코드를 통해서 예외를 발생시키게 됩니다.

# 18. 사용자 예외 발생시키기

---

- 보통, 예외를 발생시킬 경우는 아래 코드처럼 한 줄로 사용합니다.

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

 Scanner input = new Scanner(System.in);

 System.out.println("ID를 입력하세요.");
 String userId = input.nextLine();

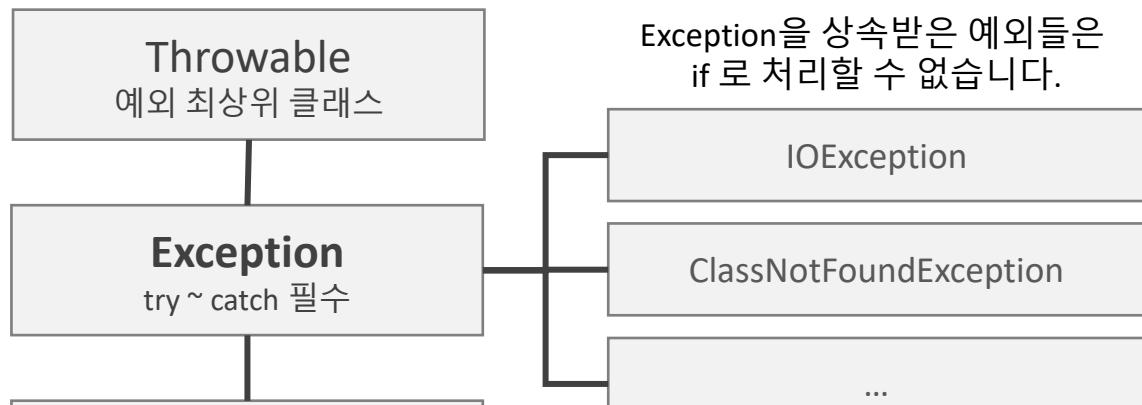
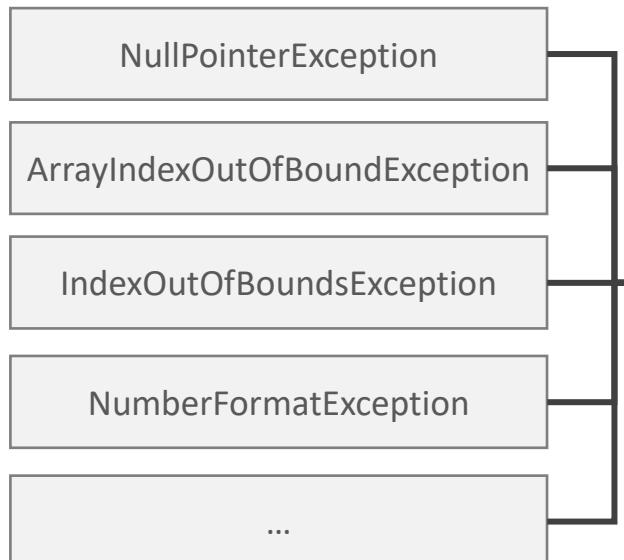
 if (userId.equals("root") || userId.equals("admin")
 || userId.equals("system")) {
 throw new InvalidUserIdException(userId + "는 사용할 수 없습니다.");
 }
}
```

# 18. 예외 위임하기

# 18. 예외 위임하기

- `RuntimeException`이 아닌 `Exception`은 `try ~ catch` 처리가 반드시 필요합니다.
- `ClassNotFoundException` 같은 `Exception` 타입의 예외가 발생하면 즉시 `try ~ catch`를 해주어야만 하고 그렇지 않을 경우 컴파일 에러가 발생합니다.

`RuntimeException`을 상속받은 예외들은  
if로 처리할 수 있습니다.



# 18. 예외 위임하기

- Java는 Exception 타입의 예외에 대한 처리를 호출자에게 위임할 수 있습니다.
- 간단한 예를 먼저 봅니다.
- 아래 코드는 Class.forName의 ClassNotFoundException을 처리하는 코드입니다.

```
public static void main(String[] args) {
 try {
 Class.forName("SomeAClass");
 }
 catch(ClassNotFoundException cnfe) {
 cnfe.printStackTrace();
 }
}
```

- 만약, 위 코드에서 ClassNotFoundException에 대한 처리를 호출자에게 위임하려다면 메소드 정의 부분에 throws ClassNotFoundException을 작성합니다.

```
public static void main(String[] args)
 throws ClassNotFoundException {
 Class.forName("SomeAClass"); // 에러가 사라집니다.
}
```

# 18. 예외 위임하기

- Throws 코드 제대로 만들어보기

```
public void loadClass(String className) throws ClassNotFoundException {
 Class.forName(className);
}
```

```
public static void main(String[] args) {
 App app = new App();
 app.loadClass("SomeAClass"); // try ~ catch 필요!
}
```

- loadClass 메소드가 ClassNotFoundException에 대한 처리를 위임했으므로
- 호출하는 부분에서 ClassNotFoundException에 대한 처리를 해야 합니다.

```
public static void main(String[] args) {
 App app = new App();
 try {
 app.loadClass("SomeAClass");
 }
 catch(ClassNotFoundException cnfe) {
 cnfe.printStackTrace();
 }
}
```

# 19. 제네릭과 컬렉션

# 19. 제네릭과 컬렉션

---

- Java 1.5 부터 추가된 제네릭(Generic)과 컬렉션 프레임워크(Collection Framework)는 유연한 데이터 처리를 가능케 하는 Java의 강력한 기능 중 하나입니다.
- 제네릭이란?  
자료구조의 형태만 제시하고 실제로 어떤 타입의 데이터가 관리될 것인지는 나중에 결정하는 기술을 말합니다.
- 컬렉션이란?  
제네릭을 이용해 Array를 조금 더 다양한 형태로 간편하게 사용할 수 있도록 제공하는 프레임워크(툴)을 말합니다.
- Java Application을 개발할 때, 아주 많이 사용되는 기술이므로 반드시 습득해 놓는 것이 좋습니다.

# 19. 제네릭(Generic)

# 19. 제네릭(Generic)

---

- 제네릭을 사용하면, 매우 유연한 자료구조를 만들어낼 수 있습니다.

```
public static void main(String[] args) {
 int[] scoreArray = new int[1];
 scoreArray[0] = 100;
}
```

- 위 코드는 정수형 배열을 만들어 값을 할당하는 코드입니다.
- 만약, 값을 하나 더 추가하려면 아래와 같은 코드가 필요합니다.

# 19. 제네릭(Generic)

- 만약, 값을 하나 더 추가하려면 아래와 같은 코드가 필요합니다.

```
public static void main(String[] args) {
 int[] scoreArray = new int[1];
 scoreArray[0] = 100;

 int[] scoreArray2 = new int[scoreArray.length + 1];
 // 배열 복사
 System.arraycopy(scoreArray, 0, scoreArray2, 0, scoreArray.length);
 scoreArray2[1] = 95;

 for(int n: scoreArray) {
 System.out.println("scoreArray: " + n);
 }

 for(int n: scoreArray2) {
 System.out.println("scoreArray2: " + n);
 }
}
```

# 19. 제네릭(Generic)

---

- 이런 배열의 불편함을 해소하기 위해 새로운 클래스 하나를 생성해봅니다.
- ScoreList.java ( 1 / 2 )

```
public class ScoreList {

 private int[] scoreArray;
 private int size;

 public ScoreList() {
 scoreArray = new int[2];
 }

 /**
 * scoreArray 배열에 값을 추가합니다.
 * @param score
 */
 public void add(int score) {

 }
```

# 19. 제네릭(Generic)

---

- 이런 배열의 불편함을 해소하기 위해 새로운 클래스 하나를 생성해봅니다.
- ScoreList.java ( 2 / 2 )

```
/**
 * scoreArray 배열에서 index의 값을 반환합니다.
 * @param index
 * @return
 */
public int get(int index) {
 return scoreArray[index];
}

public int size() {
 return size;
}

@Override
public String toString() {
 return super.toString();
}
}
```

# 19. 제네릭(Generic)

---

- ScoreList.java > add

```
/*
 * scoreArray 배열에 값을 추가합니다.
 * @param score
 */
public void add(int score) {
 if (size >= scoreArray.length) {
 int[] tempScoreArray = new int[scoreArray.length + 2];
 System.arraycopy(scoreArray, 0, tempScoreArray, 0, scoreArray.length);
 scoreArray = tempScoreArray;
 }

 scoreArray[size] = score;
 size += 1;
}
```

# 19. 제네릭(Generic)

---

- ScoreList.java > get

```
/**
 * scoreArray 배열에서 index의 값을 반환합니다.
 * @param index
 * @return
 */
public int get(int index) {
 if (index >= size) {
 throw new IndexOutOfBoundsException(index);
 }
 return scoreArray[index];
}
```

# 19. 제네릭(Generic)

---

- ScoreList.java > `toString`

```
@Override
public String toString() {
 StringBuffer sb = new StringBuffer();
 sb.append("ScoreList[");
 for (int i = 0; i < size; i++) {
 sb.append(scoreArray[i] + ", ");
 }
 sb.append("]");

 return sb.toString();
}
```

# 19. 제네릭(Generic)

---

- ScoreList를 사용해 봅니다.

```
public static void main(String[] args) {
 ScoreList score = new ScoreList();
 score.add(100);
 score.add(95);
 score.add(90);

 System.out.println(score.get(0));
 System.out.println(score.get(1));
 System.out.println(score.get(2));

 System.out.println(score.size());
 System.out.println(score);
}
```

# 19. 제네릭(Generic)

---

- 완성한 ScoreList는 int 형 타입만 사용할 수 있는 클래스입니다.
- ScoreList를 double 형 타입으로도 사용하려면 아마도 DoubleScoreList 라는 클래스를 만들어야 할 듯 합니다.

```
public class DoubleScoreList {
 private double[] scoreArray;
 private int size;

 public DoubleScoreList() {
 scoreArray = new double[2];
 }

 public void add(double score) {
 ...
 }

 public double get(int index) {
 ...
 }
}
```

# 19. 제네릭(Generic)

---

- 이런 방식 말고 사용할 타입을 나중에 정하도록 하는 제네릭을 사용하면 훨씬 더 편리해집니다.
- 제네릭을 이용한 코드로 변경해 봅니다.
- 제네릭은 클래스명 옆에 부등호(<>)를 이용해 정의합니다.

# 19. 제네릭(Generic)

- 제네릭을 적용한 ScoreList ( 1 / 2 )

```
public class ScoreList<T> {

 private Object[] scoreArray;
 private int size;

 public ScoreList() {
 scoreArray = new Object[2];
 }

 /**
 * scoreArray 배열에 값을 추가합니다.
 * @param score
 */
 public void add(T score) {
 if (size >= scoreArray.length) {
 Object[] tempScoreArray = new Object[scoreArray.length + 2];
 ... 생략 ...
 }
 ... 생략 ...
 }
}
```

# 19. 제네릭(Generic)

- 제네릭을 적용한 ScoreList ( 2 / 2 )

```
/**
 * scoreArray 배열에서 index의 값을 반환합니다.
 * @param index
 * @return
 */
public T get(int index) {
 if (index >= size) {
 throw new IndexOutOfBoundsException(index);
 }
 return (T) scoreArray[index];
}

... 생략 ...
}
```

# 19. 제네릭(Generic)

---

- <T> : 제네릭은 보통 한 글자 단위로 작성합니다.

```
public class ScoreList<T>
```

- 의미는 부여하기 나름으로 T는 Type의 앞 글자만 사용했습니다.
- 다른 타입을 사용해도 문제는 없습니다.

```
public void add(T score)
```

- 파라미터의 T는 클래스에 정의한 T와 같습니다.
- 만약, 클래스 제네릭에 String이 전달되었다면 add 메소드의 T 또한 String이 됩니다.

```
public T get(int index)
```

- 리턴타입의 T는 클래스에 정의한 T와 같습니다.
- 만약, 클래스 제네릭에 String이 전달되었다면 get 메소드의 T 또한 String이 됩니다.

# 19. 제네릭(Generic)

- 이제 제네릭 클래스를 사용해 봅니다.

```
public static void main(String[] args) {
 ScoreList<Integer> score = new ScoreList<>();
 score.add(100);
 score.add(95);
 score.add(90);

 System.out.println(score.get(0));
 System.out.println(score.get(1));
 System.out.println(score.get(2));

 System.out.println(score.size());
 System.out.println(score);
}
<Integer>
```

Class Generic에 Integer를 전달합니다. 그러면 ScoreList의 모든 T는 Integer로 변환됩니다.

## Integer?

정수형을 사용하는 int가 아닌 Integer 가 사용되었습니다.

제네릭에 전달될 수 있는 타입은 Reference Type만 사용할 수 있습니다. (이런 타입을 Wrapper 클래스라고 합니다.)

즉 Primitive Type인 int 는 사용될 수 없습니다.

# 19. 제네릭(Generic)

- 이제 String으로도 사용해봅니다.

```
public static void main(String[] args) {
 ScoreList<String> score = new ScoreList<>();
 score.add("백점");
 score.add("구십오점");
 score.add("구십점");

 System.out.println(score.get(0));
 System.out.println(score.get(1));
 System.out.println(score.get(2));

 System.out.println(score.size());
 System.out.println(score);
}
```

# 19. 제네릭(Generic)

- Java에서 제공하는 Primitive Type을 Reference Type으로 감싼 Wrapper Class 종류

| 기본형     | 클래스명              |
|---------|-------------------|
| byte    | java.lang.Byte    |
| short   | java.lang.Short   |
| int     | java.lang.Integer |
| long    | java.lang.Long    |
| float   | java.lang.Float   |
| double  | java.lang.Double  |
| char    | java.lang.Char    |
| boolean | java.lang.Boolean |

# 19. 컬렉션

# 19. 컬렉션

---

- 정확히는 컬렉션 프레임워크(Collection Framework)입니다.
- 컬렉션 프레임워크는 Java에서 흔하게 사용되는 여러 자료구조들을 제네릭을 이용해 미리 작성해 놓은 툴입니다.
- 크게 3가지 종류의 인터페이스가 있고, 이를 구현한 많은 클래스가 존재합니다.
- 인터페이스와 구현 클래스의 종류
  - List (Interface)
    - ArrayList (Implemented class)
      - 배열을 편하게 사용하기 위한 자료구조
      - 앞서 만든 ScoreList와 유사
    - Map (Interface)
      - HashMap (Implemented class)
        - 하나의 데이터가 Key:Value 쌍으로 이루어진 자료구조
    - Set (Interface)
      - HashSet (Implemented class)
        - ArrayList와 유사하지만 중복 값을 가지지 못하는 자료구조
        - 직접 사용할 일이 없습니다.

# 19. 컬렉션

---

- List & ArrayList
  - List 인터페이스를 구현한 ArrayList
  - ArrayList is a List
  - 배열 대신 사용

- List 인스턴스 만드는 방법

```
List<Integer> scoreList = new ArrayList<>();
```

- List Interface

```
public interface List<E> extends Collection<E>
```

- ArrayList Class

```
public class ArrayList<E> extends AbstractList<E>
 implements List<E>, RandomAccess, Cloneable, java.io.Serializable
```

# 19. 컬렉션

- List Interface의 주요 메소드들

| Type     | Name     | Parameters      |           | 반환타입    | 역할                       |
|----------|----------|-----------------|-----------|---------|--------------------------|
| Instance | size     | -               |           | int     | List 요소의 수               |
| Instance | isEmpty  | -               |           | boolean | List가 비어있는지 확인           |
| Instance | contains | Object o        | 비교 대상     | boolean | List내에 Object가 포함되었는지 확인 |
| Instance | add      | E e             | 값         | boolean | List에 E 추가               |
| Instance | remove   | int index       | 삭제할 인덱스   | E       | List에서 int index를 삭제     |
| Instance | addAll   | Collection c    | 복사할 리스트   | boolean | List에 Collection 전부를 추가  |
| Instance | clear    | -               |           | void    | List를 비어있는 상태로 변경        |
| Instance | get      | int index       | 인덱스       | E       | List에서 int index의 값을 반환  |
| Class    | of       | Variable Params | 값 (가변 인자) | List    | List 생성과 데이터 할당          |

# 19. 컬렉션

---

- List 사용 실습 1 (List에 데이터 추가)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 scoreList.add(100);
 scoreList.add(90);
 scoreList.add(80);
 scoreList.add(95);
 scoreList.add(85);

 System.out.println(scoreList);
 System.out.println(scoreList.size());
}
```

# 19. 컬렉션

---

- List 사용 실습 2 (List 데이터 조회)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 ... 생략 ...
 int score = scoreList.get(0);
 System.out.println(score); // 100

 score = scoreList.get(1);
 System.out.println(score); // 90

 score = scoreList.get(2);
 System.out.println(score); // 80

 score = scoreList.get(3);
 System.out.println(score); // 95

 score = scoreList.get(4);
 System.out.println(score); // 85

 score = scoreList.get(5); // IndexOutOfBoundsException
 System.out.println(score);
}
```

# 19. 컬렉션

---

- List 사용 실습 3 (List 반복 데이터 조회)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 scoreList.add(100);
 scoreList.add(90);
 scoreList.add(80);
 scoreList.add(95);
 scoreList.add(85);

 System.out.println(scoreList);
 System.out.println(scoreList.size());

 for (int i = 0; i < scoreList.size(); i++) {
 int score = scoreList.get(i);
 System.out.println(score);
 }
}
```

# 19. 컬렉션

---

- List 사용 실습 4 (List 데이터 삭제)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 scoreList.add(100);
 scoreList.add(90);
 scoreList.add(80);
 scoreList.add(95);
 scoreList.add(85);

 System.out.println(scoreList);
 System.out.println(scoreList.size());

 scoreList.remove(3);

 System.out.println(scoreList);
 System.out.println(scoreList.size());
}
```

# 19. 컬렉션

---

- List 사용 실습 5 (List 데이터 모두 삭제)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 scoreList.add(100);
 scoreList.add(90);
 scoreList.add(80);
 scoreList.add(95);
 scoreList.add(85);

 System.out.println(scoreList);
 System.out.println(scoreList.size());

 scoreList.clear();

 System.out.println(scoreList);
 System.out.println(scoreList.size());
}
```

# 19. 컬렉션

---

- List 사용 실습 6 (List가 비어있는지 확인)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 scoreList.add(100);
 scoreList.add(90);
 scoreList.add(80);
 scoreList.add(95);
 scoreList.add(85);

 System.out.println(scoreList.isEmpty());
 System.out.println(scoreList);
 System.out.println(scoreList.size());

 scoreList.clear();

 System.out.println(scoreList.isEmpty());
 System.out.println(scoreList);
 System.out.println(scoreList.size());
}
```

# 19. 컬렉션

---

- List 사용 실습 7 (값 존재 여부 확인)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 scoreList.add(100);
 scoreList.add(90);
 scoreList.add(80);
 scoreList.add(95);
 scoreList.add(85);

 System.out.println(scoreList);
 System.out.println(scoreList.size());

 if (! scoreList.contains(90)) {
 scoreList.add(90);
 }

 System.out.println(scoreList);
 System.out.println(scoreList.size());
}
```

# 19. 컬렉션

---

- List 사용 실습 7 (리스트 복사)

```
public static void main(String[] args) {
 List<Integer> scoreList = new ArrayList<>();
 scoreList.add(100);
 scoreList.add(90);
 scoreList.add(80);
 scoreList.add(95);

 List<Integer> scoreList2 = new ArrayList<>();
 scoreList2.addAll(scoreList);

 // 객체 고유 값 (메모리 주소)
 System.out.println(System.identityHashCode(scoreList));
 System.out.println(scoreList);
 System.out.println(scoreList.size());

 // 객체 고유 값 (메모리 주소)
 System.out.println(System.identityHashCode(scoreList2));
 System.out.println(scoreList2);
 System.out.println(scoreList2.size());
}
```

# 19. 컬렉션

---

- Map & HashMap
  - Map 인터페이스를 구현한 HashMap
  - HashMap is a Map
  - 데이터를 키:값 쌍(Entry)으로 관리할 때 사용
- Map 인스턴스 만드는 방법 (Key = String, Value = Integer)

```
Map<String, Integer> priceMap = new HashMap<>();
```

- Map Interface

```
public interface Map<K, V>
```

- HashMap Class

```
public class HashMap<K,V> extends AbstractMap<K,V>
 implements Map<K,V>, Cloneable, Serializable
```

# 19. 컬렉션

- Map Interface의 주요 메소드들

| Type     | Name          | Parameters     |                 | 반환타입      | 역할                         |
|----------|---------------|----------------|-----------------|-----------|----------------------------|
| Instance | put           | K key, V value | 맵에 추가할 키와 값     | V         | Map에 K, V 추가               |
| Instance | get           | Object key     | 조회할 키           | V         | Map에서 Key에 해당하는 Value를 조회  |
| Instance | size          | -              |                 | int       | Map에 추가된 Key Value Pair 개수 |
| Instance | clear         | -              |                 | void      | Map을 비어있는 상태로 변경           |
| Instance | putAll        | Map m          | Map에 추가할 다른 Map | void      | Map에 Map 전부를 추가            |
| Instance | remove        | Object key     | 삭제할 키           | V         | Map에서 Key에 해당하는 Value를 삭제  |
| Instance | containsKey   | Object key     | 비교 대상 키         | boolean   | Map에 key가 존재하는지 확인         |
| Instance | containsValue | Object value   | 비교 대상 값         | boolean   | Map에 value가 존재하는지 확인       |
| Class    | of            | K k1, V v1 ... | 맵에 추가할 키와 값     | Map<K, V> | Map 생성과 데이터 할당             |

# 19. 컬렉션

---

- Map 사용 실습 1 (Map에 데이터 추가)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 System.out.println(priceMap);
 System.out.println(priceMap.size());

 priceMap.put("LG Gram", 1_800_000);
 System.out.println(priceMap);
 System.out.println(priceMap.size());
}
```

# 19. 컬렉션

---

- Map 사용 실습 2 (Map 데이터 조회)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 int applePrice = priceMap.get("Apple Macbook Pro");
 System.out.println(applePrice);

 // java.lang.NullPointerException
 applePrice = priceMap.get("apple macbook pro");
 System.out.println(applePrice);
}
```

# 19. 컬렉션

---

- Map 사용 실습 3 (Map 데이터 삭제)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 System.out.println(priceMap);
 System.out.println(priceMap.size());

 priceMap.remove("Apple Macbook Pro");

 System.out.println(priceMap);
 System.out.println(priceMap.size());
}
```

# 19. 컬렉션

---

- Map 사용 실습 4 (Map 데이터 모두 삭제)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 System.out.println(priceMap);
 System.out.println(priceMap.size());

 priceMap.clear();

 System.out.println(priceMap);
 System.out.println(priceMap.size());
}
```

# 19. 컬렉션

- Map 사용 실습 5 (Map이 비어있는지 확인)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 boolean isEmpty = priceMap.isEmpty();
 System.out.println(isEmpty);
 System.out.println(priceMap);
 System.out.println(priceMap.size());

 priceMap.clear();
 isEmpty = priceMap.isEmpty();
 System.out.println(isEmpty);
 System.out.println(priceMap);
 System.out.println(priceMap.size());
}
```

# 19. 컬렉션

---

- Map 사용 실습 6 (Map에 동일한 Key가 있는지 확인)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 System.out.println(priceMap);
 System.out.println(priceMap.size());

 if (!priceMap.containsKey("LG Gram")) {
 priceMap.put("LG Gram", 1_600_000);
 }

 System.out.println(priceMap);
 System.out.println(priceMap.size());
}
```

# 19. 컬렉션

---

- Map 사용 실습 7 (Map에 동일한 Value가 있는지 확인)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 System.out.println(priceMap);
 System.out.println(priceMap.size());

 if (priceMap.containsValue(1_700_000)) {
 priceMap.put("LG Gram", 1_600_000);
 }

 System.out.println(priceMap);
 System.out.println(priceMap.size());
}
```

# 19. 컬렉션

- Map 사용 실습 8 (Map 복사)

```
public static void main(String[] args) {
 Map<String, Integer> priceMap = new HashMap<>();

 priceMap.put("Apple Macbook Pro", 3_500_000);
 priceMap.put("Samsung Galaxy Book", 1_500_000);
 priceMap.put("LG Gram", 1_700_000);

 Map<String, Integer> priceMap2 = new HashMap<>();
 priceMap2.putAll(priceMap);

 System.out.println(System.identityHashCode(priceMap));
 System.out.println(priceMap);
 System.out.println(priceMap.size());

 System.out.println(System.identityHashCode(priceMap2));
 System.out.println(priceMap2);
 System.out.println(priceMap2.size());
}
```

# 19. 컬렉션

---

- Set & HashSet
  - Set 인터페이스를 구현한 HashSet
  - HashSet is a Set

- Set 인스턴스 만드는 방법

```
Set<Integer> numbers = new HashSet<>();
```

- Set Interface

```
public interface Set<E> extends Collection<E>
```

- HashSet Class

```
public class HashSet<E>
 extends AbstractSet<E>
 implements Set<E>, Cloneable, java.io.Serializable
```

# 19. 컬렉션

- Map Interface의 주요 메소드들

| Type     | Name     | Parameters   |                 | 반환타입    | 역할                     |
|----------|----------|--------------|-----------------|---------|------------------------|
| Instance | size     | -            |                 | int     | Set에 추가된 값의 개수         |
| Instance | isEmpty  | -            |                 | boolean | Set이 비어있는지 확인          |
| Instance | contains | Object o     | 비교 대상           | boolean | Set에 값이 추가되어있는지 확인     |
| Instance | add      | E e          | 추가할 값           | boolean | Set에 값 추가 - 중복값은 제거    |
| Instance | remove   | Object o     | 제거할 값           | boolean | Set의 Object 값을 제거      |
| Instance | addAll   | Collection c | Set에 추가할 다른 Set | boolean | Set에 Collection을 모두 추가 |
| Instance | clear    | -            |                 | void    | Set을 비어있는 상태로 변경       |
| Class    | of       | E e1 ...     | 추가할 값           | Set<E>  | Set 생성과 데이터 할당         |

# 19. 컬렉션

---

- Set 사용 실습 1 (Set에 데이터 추가)

```
public static void main(String[] args) {
 Set<Integer> numbers = new HashSet<>();
 numbers.add(10);
 numbers.add(20);
 numbers.add(10);
 numbers.add(30);

 System.out.println(numbers);
 System.out.println(numbers.size());
}
```

# 19. 컬렉션

---

- Set 사용 실습 2 (Set 데이터 조회) – Set은 Index로 접근하지 못합니다.

```
public static void main(String[] args) {
 Set<Integer> numbers = new HashSet<>();
 numbers.add(10);
 numbers.add(20);
 numbers.add(10);
 numbers.add(30);

 for (int n: numbers) {
 System.out.println(n);
 }
}
```

# 19. 컬렉션

---

- Set 사용 실습 3 (Set 데이터 삭제)

```
public static void main(String[] args) {
 Set<Integer> numbers = new HashSet<>();
 numbers.add(10);
 numbers.add(20);
 numbers.add(10);
 numbers.add(30);

 System.out.println(numbers);
 System.out.println(numbers.size());

 numbers.remove(30);

 System.out.println(numbers);
 System.out.println(numbers.size());
}
```

# 19. 컬렉션

---

- Set 사용 실습 4 (Set 데이터 모두 삭제)

```
public static void main(String[] args) {
 Set<Integer> numbers = new HashSet<>();
 numbers.add(10);
 numbers.add(20);
 numbers.add(10);
 numbers.add(30);

 System.out.println(numbers);
 System.out.println(numbers.size());

 numbers.clear();

 System.out.println(numbers);
 System.out.println(numbers.size());
}
```

# 19. 컬렉션

---

- Set 사용 실습 5 (Set이 비어있는지 확인)

```
public static void main(String[] args) {
 Set<Integer> numbers = new HashSet<>();
 numbers.add(10);
 numbers.add(20);
 numbers.add(10);
 numbers.add(30);

 boolean isEmpty = numbers.isEmpty();
 System.out.println(isEmpty);
 System.out.println(numbers);
 System.out.println(numbers.size());

 numbers.clear();

 isEmpty = numbers.isEmpty();
 System.out.println(isEmpty);
 System.out.println(numbers);
 System.out.println(numbers.size());
}
```

# 19. 컬렉션

---

- Set 사용 실습 6 (Set에 동일한 값이 있는지 확인)

```
public static void main(String[] args) {
 Set<Integer> numbers = new HashSet<>();
 numbers.add(10);
 numbers.add(20);
 numbers.add(10);
 numbers.add(30);

 System.out.println(numbers);
 System.out.println(numbers.size());

 if (! numbers.contains(20)) {
 numbers.add(20);
 }

 System.out.println(numbers);
 System.out.println(numbers.size());
}
```

# 19. 컬렉션

---

- Set 사용 실습 7 (Set 복사)

```
public static void main(String[] args) {
 Set<Integer> numbers = new HashSet<>();
 numbers.add(10);
 numbers.add(20);
 numbers.add(10);
 numbers.add(30);

 Set<Integer> numbers2 = new HashSet<>();
 numbers2.addAll(numbers);

 System.out.println(System.identityHashCode(numbers));
 System.out.println(numbers);
 System.out.println(numbers.size());

 System.out.println(System.identityHashCode(numbers2));
 System.out.println(numbers2);
 System.out.println(numbers2.size());
}
```

# 20. File I/O

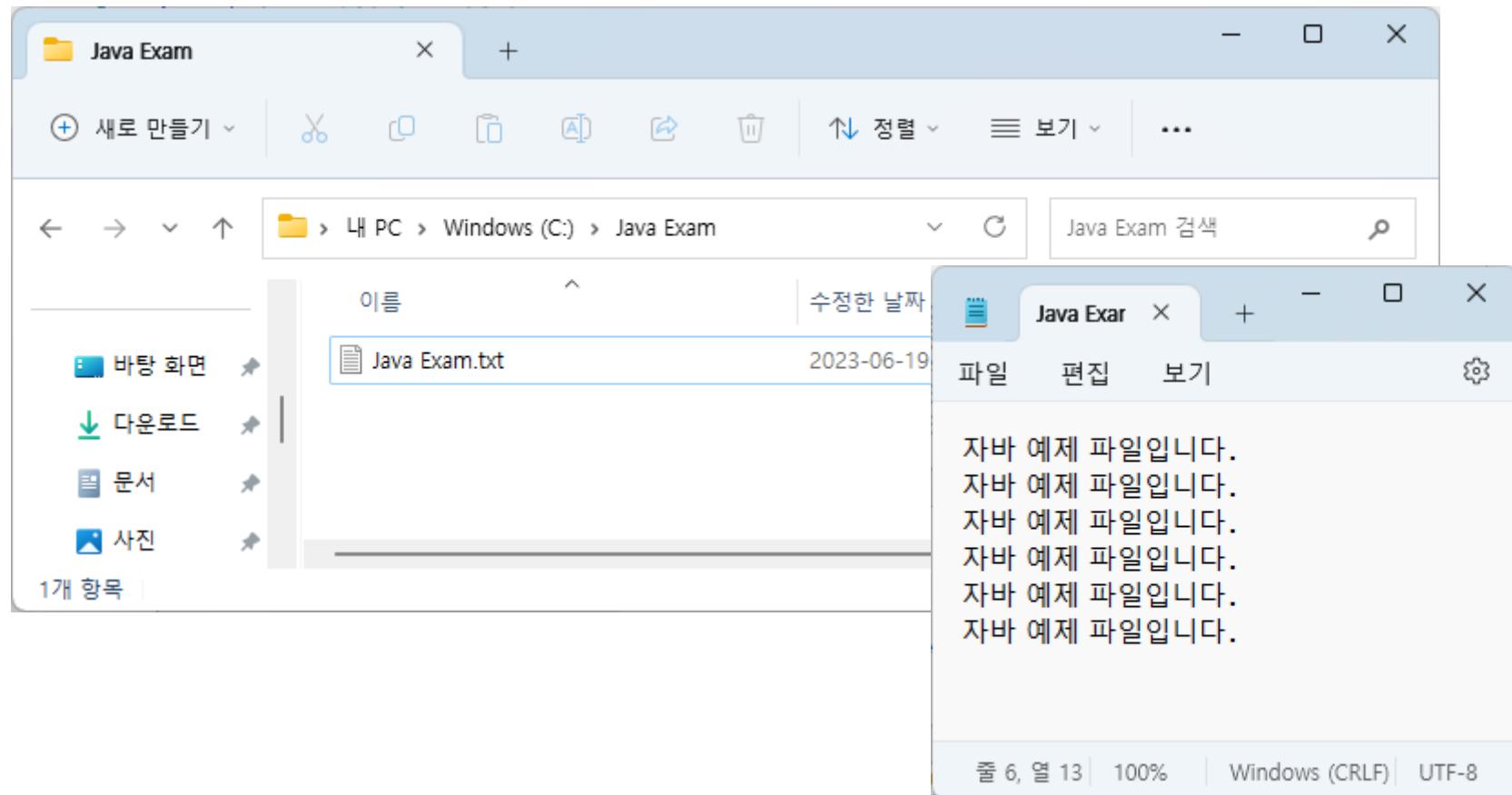
# 20. File I/O

---

- File Input/Output 으로 파일을 읽고 쓰는 방법입니다.
- File Input: 파일 읽기
  - File System에서 Java로 Input
- File Output: 파일 쓰기
  - Java에서 File System으로 Output
- Java 1.8 이전까지는 I/O가 매우 어렵고 복잡했으나
- Java 1.8 부터 비교적 간편해졌으며, 코드를 분석하기도 쉬워졌습니다.

# 20. File I/O

- File Input (파일 읽기)
- 파일을 읽기 위한 샘플 파일 하나를 생성합니다.



# 20. File I/O

- 파일을 읽어 내용을 출력하는 코드를 작성합니다. ( 1 / 3 )

```
public static void main(String[] args) {
 File file = new File("C:\\Java Exam", "Java Exam.txt");
 if (file.exists() && file.isFile()) {
 // C:\\Java Exam\\Java Exam.txt이 존재하며
 // C:\\Java Exam\\Java Exam.txt가 파일이라면

 FileReader reader = null;
 BufferedReader bufferedReader = null;
 try {
 // 파일을 바이트 단위로 읽어오는 FileReader를 선언
 reader = new FileReader(file, Charset.forName("UTF-8"));
 // 파일을 라인 단위로 읽어오는 BufferedReader를 선언
 bufferedReader = new BufferedReader(reader);
 // 파일을 라인단위로 읽어와 할당하기 위한 String 변수 선언
 String line = null;
 // 파일이 끝날 때 까지(EOF) 반복하며 line 변수에 한 줄 씩 읽어오기
 while ((line = bufferedReader.readLine()) != null) {
 // 읽어온 라인을 출력
 System.out.println(line);
 }
 }
```

# 20. File I/O

- 파일을 읽어 내용을 출력하는 코드를 작성합니다. ( 2 / 3 )

```
catch(IOException ioe) {
 // 파일을 읽다가 예외 발생하면 예외 메시지만 출력
 System.out.println(ioe.getMessage());
}
finally {
 // 파일을 끝까지 읽었거나 예외가 발생했을 경우
 // FileReader를 닫아준다.
 if (reader != null) {
 try {
 reader.close();
 }
 catch(IOException ioe) {}
 }
 // 파일을 끝까지 읽었거나 예외가 발생했을 경우
 // BufferedReader를 닫아준다.
 if (bufferedReader != null) {
 try {
 bufferedReader.close();
 }
 catch(IOException ioe) {}
 }
}
```

# 20. File I/O

---

- 파일을 읽어 내용을 출력하는 코드를 작성합니다. ( 3 / 3 )

```
 }
 }
}
```

# 20. File I/O

- Java 1.8부터 파일을 읽어오는 방법이 비교적 단순해 졌습니다.

```
public static void main(String[] args) {
 File file = new File("C:\\Java Exam", "Java Exam.txt");
 if (file.exists() && file.isFile()) {
 List<String> fileLine = new ArrayList<>();
 try {
 // 파일을 처음부터 끝까지 모두 읽어와 List에 할당합니다.
 Path filePath = Paths.get("C:\\Java Exam", "Java Exam.txt");
 Charset utf8 = Charset.forName("UTF-8");
 fileLine.addAll(Files.readAllLines(filePath, utf8));
 }
 catch(IOException ioe) {
 // 파일을 읽다가 예외가 발생했을 때, 예외 내용만 출력합니다.
 System.out.println(ioe.getMessage());
 }

 // 읽어온 파일을 모두 출력합니다.
 for (String line : fileLine) {
 System.out.println(line);
 }
 }
}
```

# 20. File I/O

- 파일의 기본적인 정보 얻어오기
- Java의 File 객체를 통해 파일의 기본적인 정보를 얻어올 수 있습니다.
  - 파일의 절대 경로
  - 파일의 크기
  - 등등..

```
File file = new File("C:\\Java Exam", "Java Exam.txt");
System.out.println(file.exists()); // 파일이 존재하는지 여부 확인
System.out.println(file.isFile()); // 파일 객체인지 확인
System.out.println(file.isDirectory()); // 폴더 객체인지 확인
System.out.println(file.getAbsolutePath()); // 파일의 절대경로 확인
System.out.println(file.getName()); // 파일의 이름 확인
System.out.println(file.length()); // 파일의 크기(byte) 확인
System.out.println(file.lastModified()); // 파일이 마지막으로 수정된 시간
Date date = new Date(file.lastModified());
System.out.println(date);
System.out.println(file.getParent()); // 파일이 존재하는 경로
// 객체가 폴더일 경우, 폴더내에 존재하는 모든 항목의 목록
System.out.println(file.listFiles());
```

# 20. File I/O

---

- 파일 객체의 정보를 이용해 파일 시스템 순회 탐색도 가능합니다.
- “C:\Program Files (x86)” 내부의 모든 폴더와 파일을 출력하는 코드를 작성해봅니다.

```
public void printAllItems(File dir) {
 if (dir.exists() && dir.isDirectory()) {
 File[] itemInDir = dir.listFiles();
 for (File item : itemInDir) {
 if (item.isDirectory()) {
 printAllItems(item);
 }
 else {
 System.out.println(item.getAbsolutePath());
 }
 }
 }
 else if (dir.isFile()) {
 System.out.println(dir.getAbsolutePath());
 }
}
```

# 20. File I/O

---

- `printAllItems`를 호출하면 해당 폴더부터 모든 내용들을 순회하며 출력하게 됩니다.

```
public static void main(String[] args) {
 App app = new App();
 File dir = new File("C:\\Program Files (x86)");
 app.printAllItems(dir);
}
```

# 20. File I/O

---

- 파일을 쓸 때엔 확인 해야 할 항목들이 많이 있습니다.
- C:\java\outputs 라는 폴더에 java\_output.txt 라는 파일을 쓰려 합니다.
  - 1. C:\java\outputs 라는 파일이 존재하는지 확인
  - 2. 없다면 폴더를 생성해야 합니다.
  - 3. java\_output.txt 파일이 존재하는지 확인
  - 4. 이미 있다면 새로 생성할 파일의 이름을 java\_output (2).txt 로 생성합니다.
  - 5. 이 마저도 있다면 java\_output (3).txt로 생성합니다.
  - 6. 파일이 없을 때 까지 순번을 증가시키며 찾습니다.
  - 7. 존재하지 않는 파일 순번을 찾았다면 파일 내용을 씁니다.

# 20. File I/O

---

- Java를 이용해 폴더를 만들고 파일을 생성/쓰기 하는 코드를 작성해봅니다. ( 1 / 3 )

```
public static void main(String[] args) {
 File file = new File("C:\\java\\outputs", "java_output.txt");
 if (! file.getParentFile().exists()) {
 file.getParentFile().mkdirs();
 }

 int index = 2;
 while (file.exists()) {
 file = new File("C:\\java\\outputs",
 "java_output (" + (index++) + ").txt");
 }

 FileWriter fw = null;
 BufferedWriter bw = null;
```

# 20. File I/O

---

- Java를 이용해 폴더를 만들고 파일을 생성/쓰기 하는 코드를 작성해봅니다. ( 2 / 3 )

```
try {
 fw = new FileWriter(file, Charset.forName("UTF-8"));
 bw = new BufferedWriter(fw);
 bw.write("파일을 씁니다1.\n");
 bw.write("파일을 씁니다2.\n");
 bw.write("파일을 씁니다3.\n");
 bw.flush();
}
catch(IOException ioe) {
 System.out.println(ioe.getMessage());
}
```

# 20. File I/O

---

- Java를 이용해 폴더를 만들고 파일을 생성/쓰기 하는 코드를 작성해봅니다. ( 3 / 3 )

```
finally {
 if (fw != null) {
 try {
 fw.close();
 }
 catch(IOException ioe) {}
 }
 if (bw != null) {
 try {
 bw.close();
 }
 catch(IOException ioe) {}
 }
}
System.out.println(file.getAbsolutePath());
}
```

# 20. File I/O

---

- Java 1.8부터 파일을 쓰는 방법이 비교적 단순해 졌습니다. ( 1 / 2 )

```
public static void main(String[] args) {
 File file = new File("C:\\java\\outputs", "java_output.txt");
 if (! file.getParentFile().exists()) {
 file.getParentFile().mkdirs();
 }

 int index = 2;
 while (file.exists()) {
 file = new File("C:\\java\\outputs",
 "java_output (" + (index++) + ").txt");
 }

 List<String> fileDesc = new ArrayList<>();
 fileDesc.add("파일을 씁니다1.");
 fileDesc.add("파일을 씁니다2.");
 fileDesc.add("파일을 씁니다3.");
```

# 20. File I/O

---

- Java 1.8부터 파일을 쓰는 방법이 비교적 단순해 졌습니다. ( 2 / 2 )

```
try {
 Path filePath = Paths.get(file.getParent(), file.getName());
 Charset utf8 = Charset.forName("UTF-8");
 Files.write(filePath, fileDesc, utf8);
}
catch(IOException ioe) {
 System.out.println(ioe.getMessage());
}

System.out.println(file.getAbsolutePath());
}
```

# 20. File I/O

---

- 파일에 내용을 덧붙이려면 옵션을 부여하면 됩니다.

```
try {
 Path filePath = Paths.get(file.getParent(), file.getName());
 Charset utf8 = Charset.forName("UTF-8");
 Files.write(filePath, fileDesc, utf8, StandardOpenOption.APPEND);
}
catch(IOException ioe) {
 System.out.println(ioe.getMessage());
}
```

# 20. File I/O

---

- 파일을 삭제하려면, File 객체의 delete 인스턴스 메소드를 호출합니다.

```
public static void main(String[] args) {
 File file = new File("C:\\java\\outputs", "java_output.txt");

 boolean isDeleted = file.delete();
 System.out.println(isDeleted);
}
```

- 폴더를 삭제하려면, File 객체의 delete 인스턴스 메소드를 호출합니다.

```
public static void main(String[] args) {
 File file = new File("C:\\java\\outputs");

 boolean isDeleted = file.delete();
 System.out.println(isDeleted);
}
```

- 그러나, 폴더가 비어있지 않다면 삭제되지 않습니다.

# 20. File I/O

---

- 비어있지 않은 폴더를 삭제하려면 파일을 순회 탐색하며 일일이 모든 파일을 삭제를 해주고 마지막에 폴더를 삭제해야 합니다.

```
public void deleteAllItems(File dir) {
 if (dir.exists() && dir.isDirectory()) {
 File[] items = dir.listFiles();
 for (File file : items) {
 if (file.isDirectory()) {
 deleteAllItems(file);
 }
 file.delete();
 }
 dir.delete();
 }
 else if (dir.isFile()) {
 dir.delete();
 }
}
```

# 20. File I/O

---

- 폴더를 삭제하는 `deleteAllItems`를 호출해 봅니다.

```
public static void main(String[] args) {
 File file = new File("C:\\java\\outputs");

 App app = new App();
 app.deleteAllItems(file);
}
```

# 21. 열거형 (Enum)

# 21. Enum

---

- 상수를 파라미터로 받는 메소드가 이따금씩 필요합니다.
- 아래 코드는 type의 값에 따라 사칙연산을 처리합니다. ( 1 / 2 )

```
private static final int ADD = 1;
private static final int SUB = 2;
private static final int MUL = 3;
private static final int DIV = 4;

public void calc(int type, int num1, int num2) {
 if (type == ADD) {
 System.out.println(num1 + num2);
 }
 else if (type == SUB) {
 System.out.println(num1 - num2);
 }
 else if (type == MUL) {
 System.out.println(num1 * num2);
 }
 else if (type == DIV) {
 System.out.println(num1 / num2);
 }
}
```

# 21. Enum

---

- 상수를 파라미터로 받는 메소드가 이따금씩 필요합니다.
- 아래 코드는 type의 값에 따라 사칙연산을 처리합니다. ( 2 / 2 )

```
public static void main(String[] args) {
 App app = new App();
 app.calc(ADD, 10, 20);
 app.calc(SUB, 10, 20);
 app.calc(MUL, 10, 20);
 app.calc(DIV, 10, 20);
}
```

# 21. Enum

---

- 이 코드는 상수를 전달받기를 의도했지만 type 파라미터에 상수를 전달하지 않고 일반 숫자를 전달할 수도 있다는 문제점을 가지고 있습니다.

```
public static void main(String[] args) {
 App app = new App();
 app.calc(1, 10, 20);
 app.calc(SUB, 10, 20);
 app.calc(MUL, 10, 20);
 app.calc(4, 10, 20);
}
```

# 21. Enum

- 파라미터로 상수를 전달받고자 할 때, 위 코드와 같은 문제점을 피하기 위해 Enum을 사용할 수 있습니다. (1 / 2)

```
public static enum Type {
 ADD, SUB, MUL, DIV
}
```

```
public void calc(Type type, int num1, int num2) {
 if (type == Type.ADD) {
 System.out.println(num1 + num2);
 }
 else if (type == Type.SUB) {
 System.out.println(num1 - num2);
 }
 else if (type == Type.MUL) {
 System.out.println(num1 * num2);
 }
 else if (type == Type.DIV) {
 System.out.println(num1 / num2);
 }
}

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

# 21. Enum

---

- 파라미터로 상수를 전달받고자 할 때, 위 코드와 같은 문제점을 피하기 위해 Enum을 사용할 수 있습니다. (2 / 2)

```
public static void main(String[] args) {
 App app = new App();
 app.calc(Type.ADD, 10, 20);
 app.calc(Type.SUB, 10, 20);
 app.calc(Type.MUL, 10, 20);
 app.calc(Type.DIV, 10, 20);
}
```

# 21. Enum

---

- Application 내부에서 유일한 객체로 사용할 수 있도록 하는 개발 패턴을 Singleton Pattern 이라고 합니다.
- Singleton Pattern으로 만들어진 클래스는 Application에서 단 한번만 인스턴스 생성을 할 수 있기 때문에, 처음 만들어진 인스턴스를 변경하지 않고 계속 사용할 수 있도록 합니다.
- Enum은 Java에서 가장 완벽한 Singleton Instance입니다.  
인스턴스 생성을 할 필요없이 한번 정의하는 것으로 유일한 값으로 사용할 수 있기 때문입니다.
- 따라서, 상수를 파라미터로 받고자 할 경우 Enum을 사용하는 것이 가장 좋은 방법입니다.

# 21. Enum

- Enum은 값을 가질 수도 있습니다.

```
public static enum Type {
 ADD("더하기"), SUB("빼기"), MUL("곱하기"), DIV("나누기");

 String name;
 Type(String name) {
 this.name = name;
 }

 public String getName() {
 return name;
 }
}

public void calc(Type type, int num1, int num2) {
 System.out.println(type.getName() + "연산을 시작합니다.");

 ... 생략 ...
```

# **22. Calendar / LocalDateTime**

# 22. Calendar / LocalDateTime

---

- Java에서는 날짜와 시간을 처리할 수 있는 3가지 클래스를 제공합니다.
  - 1. Date
  - 2. Calendar
  - 3. LocalDateTime (Java 1.8에서 추가)
- Java 1.8 이전까지 Calendar가 많이 사용되었지만, Java 버전이 꾸준히 릴리즈 됨에 따라 많은 메소드들이 Deprecated(권장하지 않음)되어 이제는 사용을 권장하지 않습니다.
- Java 1.8 이전의 버전을 사용하는 Project라면 Calendar를 사용하고 Java 1.8 이후의 버전을 사용하는 Project라면 LocalDateTime을 사용합니다.

# **22. Calendar**

# 22. Calendar

---

- Calendar는 날짜를 제어하는 클래스입니다.
- Calendar 사용 예제를 확인해 봅니다. ( 1 / 2 )

```
public static void main(String[] args) {
 // Calendar 인스턴스 가져오기
 Calendar nowCal = Calendar.getInstance();

 // 현재 연월일 시분초 조회하기
 System.out.println(nowCal.get(Calendar.YEAR));
 System.out.println(nowCal.get(Calendar.MONTH) + 1);
 System.out.println(nowCal.get(Calendar.DAY_OF_MONTH));
 System.out.println(nowCal.get(Calendar.HOUR));
 System.out.println(nowCal.get(Calendar.MINUTE));
 System.out.println(nowCal.get(Calendar.SECOND));
 // 1(일요일) ~ 7(토요일)
 System.out.println(nowCal.get(Calendar.DAY_OF_WEEK));
}
```

# 22. Calendar

---

- Calendar는 날짜를 제어하는 클래스입니다.
- Calendar 사용 예제를 확인해 봅니다. ( 2 / 2 )

```
public static void main(String[] args) {
 // 현재 연월일만 문자열로 가져오기
 // 현재 날짜/시간
 Date now = Calendar.getInstance().getTime();
 System.out.println(now);

 // 날짜 포맷 지정
 SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
 // 날짜를 포맷에 맞춰 변경
 String formatDate = format.format(now);
 // 출력
 System.out.println(formatDate);
}
```

# 22. Calendar

- Calendar를 이용해 날짜를 변경해 봅니다.

```
// Calendar 인스턴스 가져오기
Calendar nowCal = Calendar.getInstance();
// 날짜를 지정. (2022년 2월 1일)
nowCal.set(2022, 01, 01);
// 날짜에 10일 더하기
nowCal.add(Calendar.DAY_OF_MONTH, 10);
// 날짜에 20일 빼기
nowCal.add(Calendar.DAY_OF_MONTH, -20);
```

| Field                 | 의미          |
|-----------------------|-------------|
| Calendar.YEAR         | 연           |
| Calendar.MONTH        | 월 (00 ~ 11) |
| Calendar.DAY_OF_MONTH | 일           |
| Calendar.HOUR         | 시           |
| Calendar.MINUTE       | 분           |
| Calendar.SECOND       | 초           |

## **22. LocalDateTime**

# 22. LocalDateTime

---

- Java 1.8 이후에 Calendar를 대체하기 위해 날짜, 시간, 날짜시간을 처리할 수 있는 클래스 3개가 추가되었습니다.
- 각 코드에 대한 표현 방법을 확인합니다. ( 1 / 3 )
- 날짜 조회

```
public static void main(String[] args) {
 LocalDate nowDate = LocalDate.now();
 DateTimeFormatter dateFormatter =
 DateTimeFormatter.ofPattern("yyyy년 MM월 dd일");
 String strNowDate = dateFormatter.format(nowDate);
 System.out.println(nowDate);
 System.out.println(strNowDate);
}
```

# 22. LocalDateTime

---

- 각 코드에 대한 표현 방법을 확인합니다. ( 2 / 3 )
- 시간 조회

```
public static void main(String[] args) {
 LocalTime nowTime = LocalTime.now();
 DateTimeFormatter timeFormatter =
 DateTimeFormatter.ofPattern("HH시 mm분 ss초");
 String strNowTime = timeFormatter.format(nowTime);
 System.out.println(nowTime);
 System.out.println(strNowTime);
}
```

# 22. LocalDateTime

---

- 각 코드에 대한 표현 방법을 확인합니다. ( 3 / 3 )
- 날짜와 시간 조회

```
public static void main(String[] args) {
 LocalDateTime nowDateTime = LocalDateTime.now();
 DateTimeFormatter dateTimeFormatter =
 DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
 String strNowDateTime = dateTimeFormatter.format(nowDateTime);
 System.out.println(nowDateTime);
 System.out.println(strNowDateTime);
}
```

# 22. LocalDateTime

---

- LocalDateTime를 이용해 날짜를 변경해 해봅니다. ( 1 / 3 )

```
LocalDate nowDate = LocalDate.of(2022, 1, 1);
nowDate = nowDate.plusDays(10);
nowDate = nowDate.plusMonths(2);
nowDate = nowDate.plusYears(3);
DateTimeFormatter dateFormatter =
 DateTimeFormatter.ofPattern("yyyy년 MM월 dd일");
String strNowDate = dateFormatter.format(nowDate);
System.out.println(nowDate);
System.out.println(strNowDate);
```

# 22. LocalDateTime

---

- LocalDateTime를 이용해 날짜를 변경해 해봅니다. ( 2 / 3 )

```
LocalTime nowTime = LocalTime.of(00, 00, 01);
nowTime = nowTime.plusHours(10);
nowTime = nowTime.plusMinutes(5);
nowTime = nowTime.plusSeconds(55);
DateTimeFormatter timeFormatter =
 DateTimeFormatter.ofPattern("HH시 mm분 ss초");
String strNowTime = timeFormatter.format(nowTime);
System.out.println(nowTime);
System.out.println(strNowTime);
```

# 22. LocalDateTime

---

- LocalDateTime를 이용해 날짜를 변경해 해봅니다. ( 3 / 3 )

```
LocalDateTime nowDateTime = LocalDateTime.of(2022, 2, 1, 11, 39, 11);
nowDateTime = nowDateTime.plusDays(10);
nowDateTime = nowDateTime.plusMonths(2);
nowDateTime = nowDateTime.plusYears(3);
nowDateTime = nowDateTime.plusHours(10);
nowDateTime = nowDateTime.plusMinutes(5);
nowDateTime = nowDateTime.plusSeconds(55);
DateTimeFormatter dateTimeFormatter =
 DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
String strNowDateTime = dateTimeFormatter.format(nowDateTime);
System.out.println(nowDateTime);
System.out.println(strNowDateTime);
```

# 22. LocalDateTime

---

- Java 1.8 부터는 날짜와 날짜사이의 차이도 구할 수 있습니다.

```
LocalDate startDate = LocalDate.of(2022, 1, 1);
LocalDate endDate = LocalDate.of(2023, 5, 20);
Period between = Period.between(startDate, endDate);
System.out.println(between.getYears() + ", "
 + between.getMonths() + ", "
 + between.getDays());
```

# 감사합니다.

---

Java Programming  
Basic and Advanced

장민창

mcjang@hucloud.co.kr