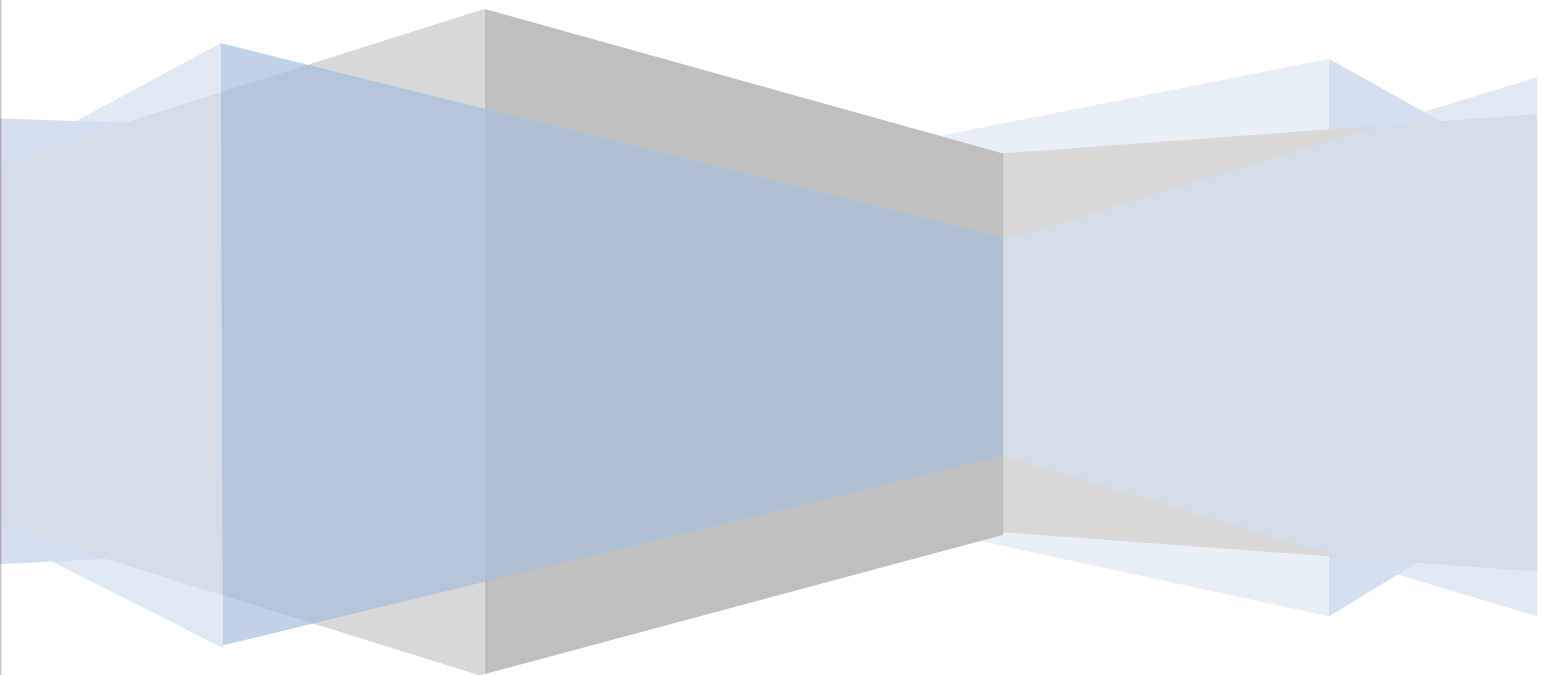


Java Programming



자바 소개

- 1995년 Sun Microsystems 에서 발표
- 오라클(Oracle)이 2010 년에 Sun Microsystems 를 인수

자바 응용

● 웹 애플리케이션

- 자바는 웹 개발을 위한 Servlet, JSP 등을 제공한다.

● 엔터프라이즈 애플리케이션

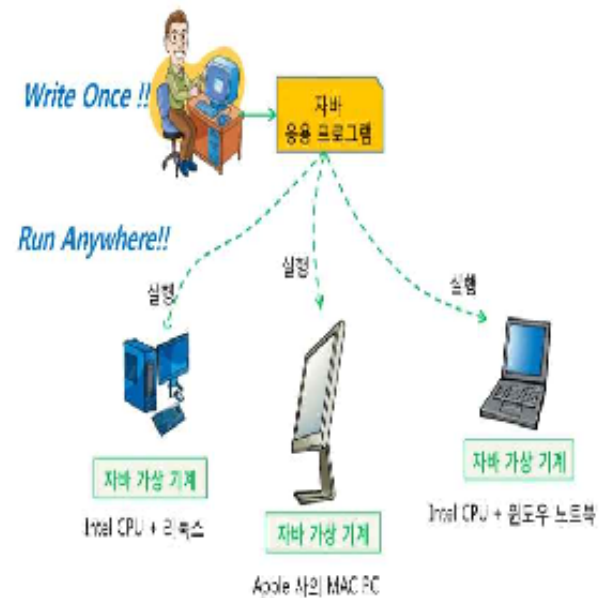
- 자바는 엔터프라이즈 애플리케이션을 개발하기 위한 Java EE(Java Enterprise Edition) 플랫폼을 제공한다.
- ERP(Enterprise Resource Planning) 시스템, 고객자원관리(CRM) 시스템 등

● 모바일 애플리케이션

- 자바는 모바일 애플리케이션을 빌드하기 위한 크로스 플랫폼 프레임워크인 J2ME 기능을 제공한다.
- 모바일 운영체제인 Android가 Java 기반 Android SDK를 사용하여 개발되었다.

자바의 특징

- 객체 지향 프로그래밍 언어(OOP) 이다.
- 플랫폼 독립적이다.
- WORA(Write Once Run Anywhere)
- 견고하다.
 - 자동 가비지 컬렉션 및 예외 처리
 - 컴파일과 런타임 시에 오류 검사

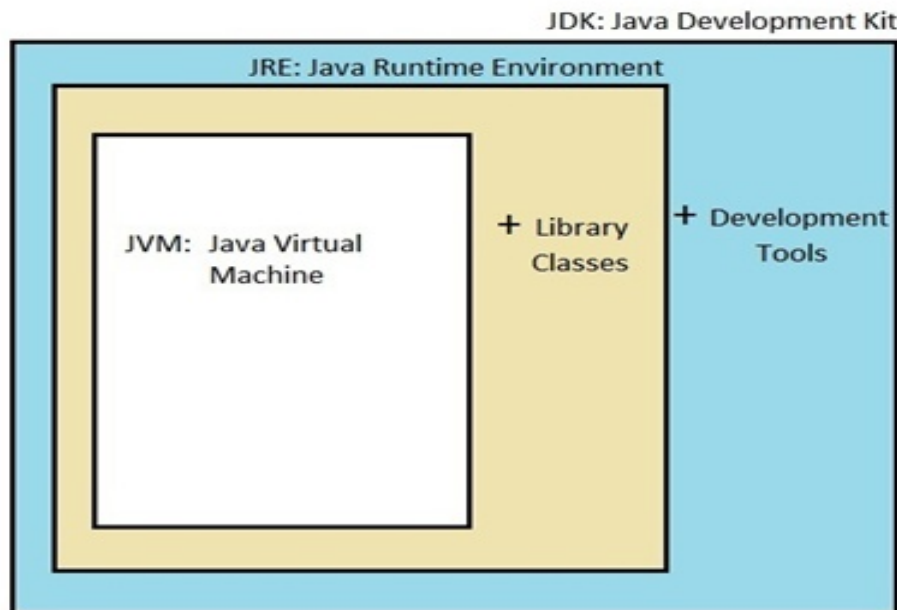


※ 플랫폼이란?

- 운영체제와 CPU 아키텍처를 말한다.

자바 개발 도구(JDK) 설치

- 자바 애플리케이션을 개발하는데 필요한 도구(Tool)와 환경을 제공한다.



JDK = JRE + Development Tools

JRE = JVM + Library Classes

출처 : https://velog.io/@sung_hyuki/JVM-JRE-JDK

JDK 의 종류

- Open JDK : <http://openjdk.java.net/>
 - 개발, 학습, 상업용 모두 무료(오픈소스기반)로 사용

- Oracle JDK : <https://www.oracle.com/java/>
 - 개발, 학습용은 무료
 - 상업용으로 사용할 경우 연간 사용료 지불
 - 장기 기술 지원(LTS : Long Term Support)으로 안정적이다.

Oracle JDK 설치 (jdk-11.0.14)

1. <https://www.oracle.com/java/> 페이지로 이동한다.
2. 상단에서 'Download Java' 버튼을 선택한다.
3. Java 11 을 선택하고 Windows 항목을 선택한다.
4. jdk-11.0.14_windows-x64_bin.exe 파일을 다운로드한다.

Java SE subscribers have more choices

Also available for development, personal use, and to run other licensed Oracle products.

Java 8 **Java 11**

Java SE Development Kit 11.0.14


Java SE subscribers will receive JDK 11 updates until at least **September of 2026**.

These downloads can be used for development, personal use, or to run Oracle licensed products. Use for other purposes, including production or commercial use, requires a Java SE or Oracle license.

JDK 11 software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE.

JDK 11.0.14 checksum

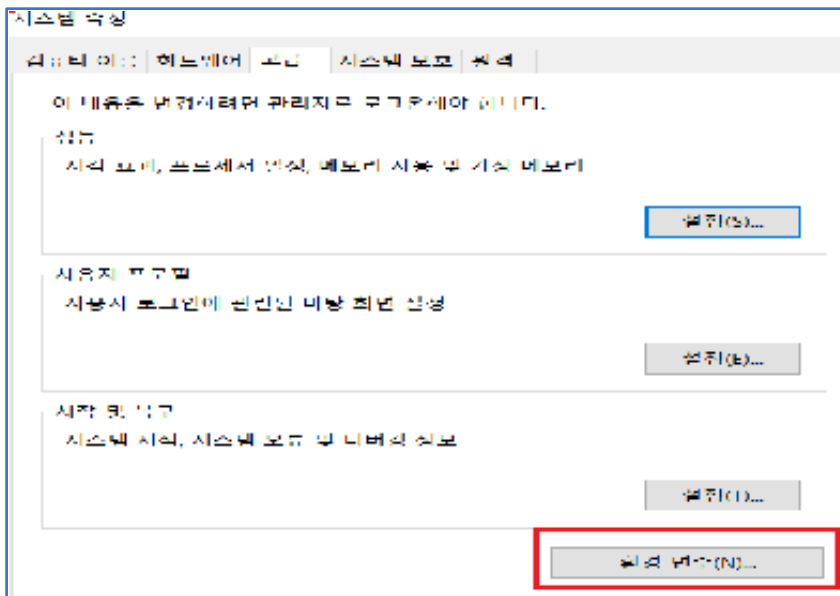
Linux macOS Solaris **Windows**

Product/file description	File size	Download
x64 installer	140.24 MB	 jdk-11.0.14_windows-x64_bin.exe

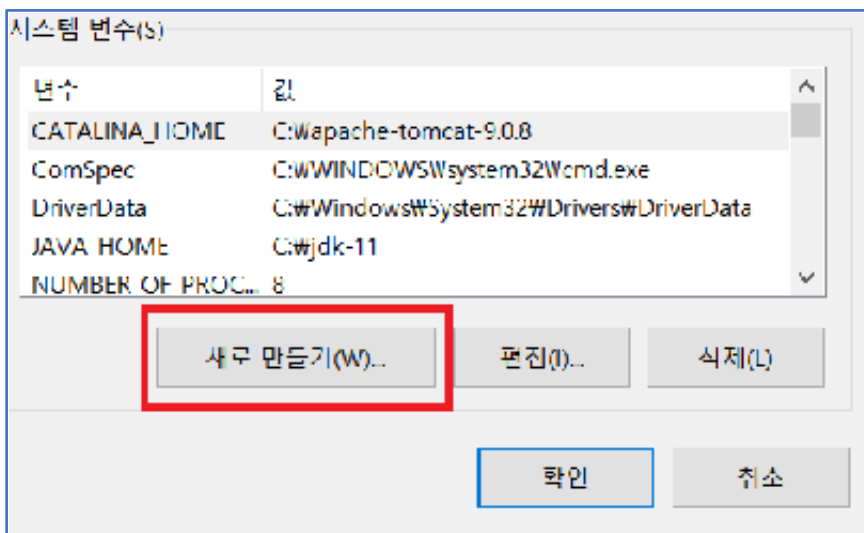
※ 설치 경로 : C:\jdk-11.0.14

환경변수 설정

1. 검색창에 시스템 환경 변수 편집을 입력한다.
2. 환경 변수(N)... 버튼을 선택한다.

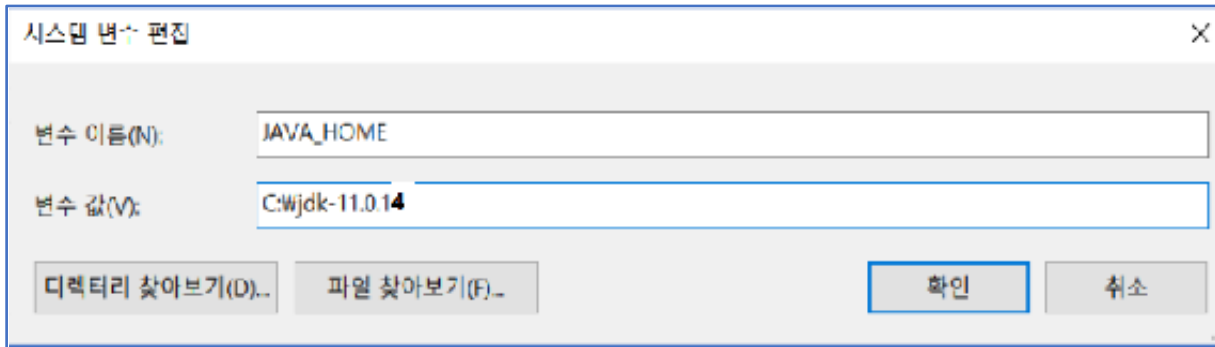


3. 시스템 변수 에서 새로 만들기(W)... 버튼을 선택한다.

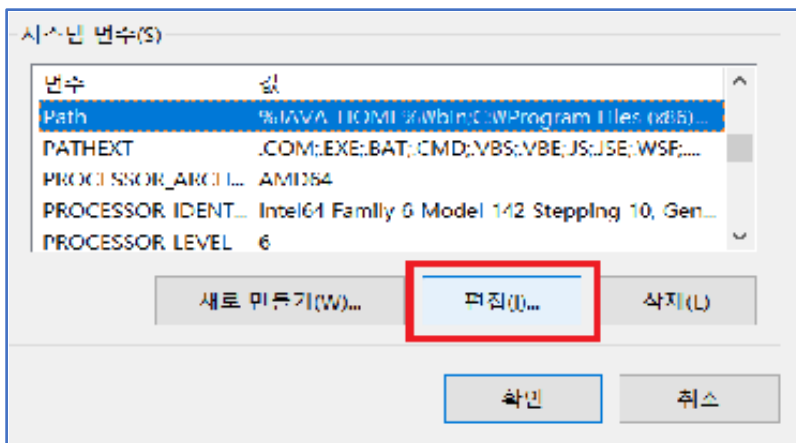


4. 변수 이름 : JAVA_HOME

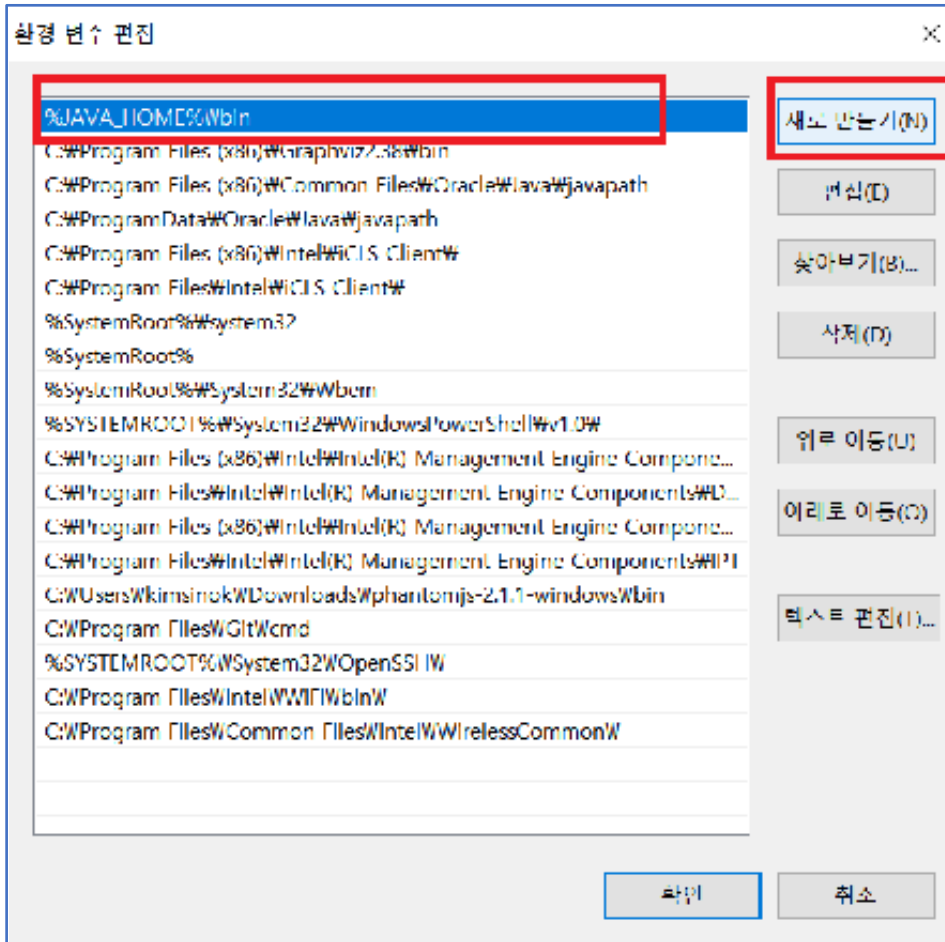
변수 값 : JDK 설치 경로



5. 시스템 변수에서 Path를 선택하고 편집(E)... 버튼을 선택한다.



6. 새로 만들기 버튼을 선택한 후 %JAVA_HOME%\bin 을 입력한 후 맨 위로 이동시킨다.



7. 확인 버튼을 선택한다.

JDK 버전 확인 - CLI (Command-line interface)

1. 시작 메뉴에서 명령 프롬프트를 선택한다. (또는 검색창에 cmd를 입력한다.)
2. 명령 프롬프트 창에서 SET을 입력한다.

- SET : 명령 프롬프트 환경에서 환경변수 설정 및 확인하는 명령어
- 환경 변수 JAVA_HOME, PATH를 확인한다.

C:\W> SET PATH

C:\W >SET JAVA_HOME

3. JDK 버전 확인

C:\W> java -version

간단한 자바 프로그램 작성하기

1. 소스 파일을 작성한다. (**Hello.java**)

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

2. Java Compiler (**javac**)를 사용하여 자바 소스 파일을 컴파일한다.

컴파일을 하면 자바 바이트 코드 (**Hello.class**)가 생성된다.

```
C:\user> javac Hello.java
```

3. 자바 명령어(**java**)는 자바 바이트 코드를 해석(interpret)하고 실행한다.

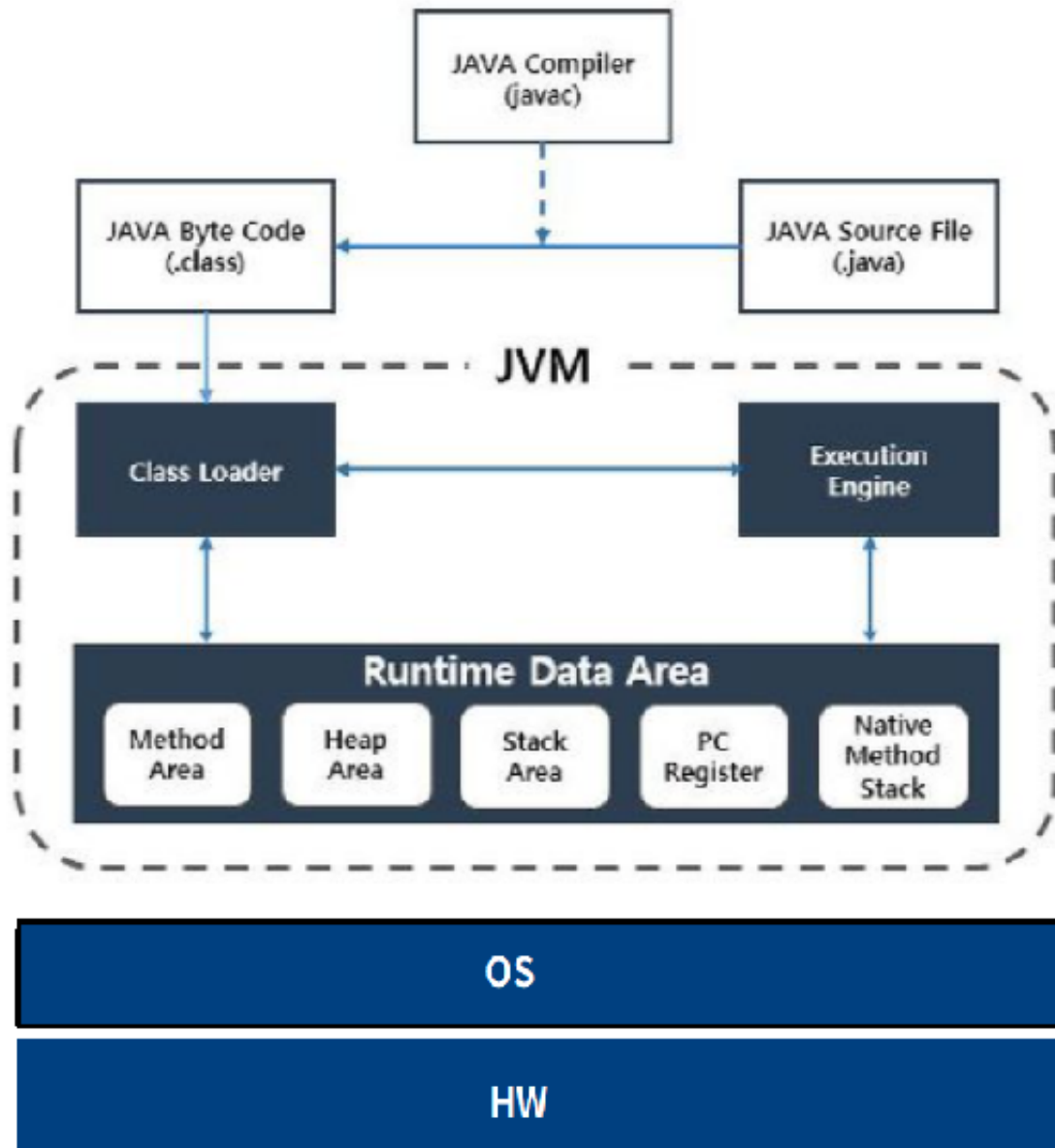
이 과정에서 바이트 코드가 바이너리 코드로 변경된다.

```
C:\user> java Hello
```

※ 자바 바이트 코드란?

- 자바 가상 머신이 실행하는 명령어의 형태이다.

자바 프로그램 실행과정 및 JVM 의 구조



● Java Compiler (javac)

- Java 소스 파일은 JVM 이 해석할 수 있는 자바 바이트 코드(.class 파일)로 변경된다.

● Class Loader

- JVM 내로 .class 파일을 로드한다. 로딩된 클래스들은 Runtime Data Area 에 배치된다.

● Execution Engine

- 로딩된 클래스의 바이트 코드를 해석(interpret)하고 실행한다. 이 과정에서 바이트 코드가 바이너리 코드로 변경된다.

● Runtime Data Area

- JVM 이 OS(운영체제)로 부터 할당받은 메모리 영역이다.

Runtime Data Area

- JVM 이 프로그램을 수행하기 위해서 OS(운영체제)로 부터 할당받은 메모리 영역이다.

● Method Area

- Class 와 Interface 의 자바 바이트 코드 및 메타 데이터가 저장된다.
- Field Information, Method Information, Class Variable(클래스 변수), 상수
- Java 8 이후로는 Metaspace 라는 OS 가 관리하는 영역으로 변경되었다.

● Heap Area

- new 명령어로 생성된 인스턴스가 저장되는 영역
- Garbage Collection의 대상이 되는 영역

● Stack Area

- 메소드 내에 사용되는 값들(매개변수, 지역변수, 리턴값 등)이 저장되는 영역.

● PC Register

- CPU 의 register 와 역할이 비슷하다. 현재 수행중인 JVM 명령의 주소값이 저장된다.

● Native Method Stack

- 다른 언어(C/C++ 등)로 작성된 네이티브 코드들을 위한 영역
- JNI(Java Native Interface)를 통해 호출되는 C/C++ 코드

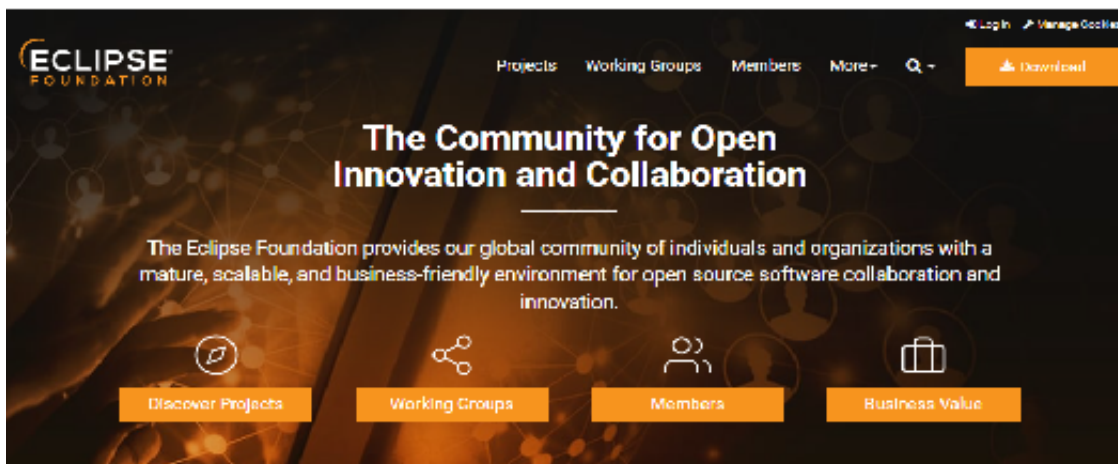
Eclipse 개발 환경 구축

● Eclipse 란?

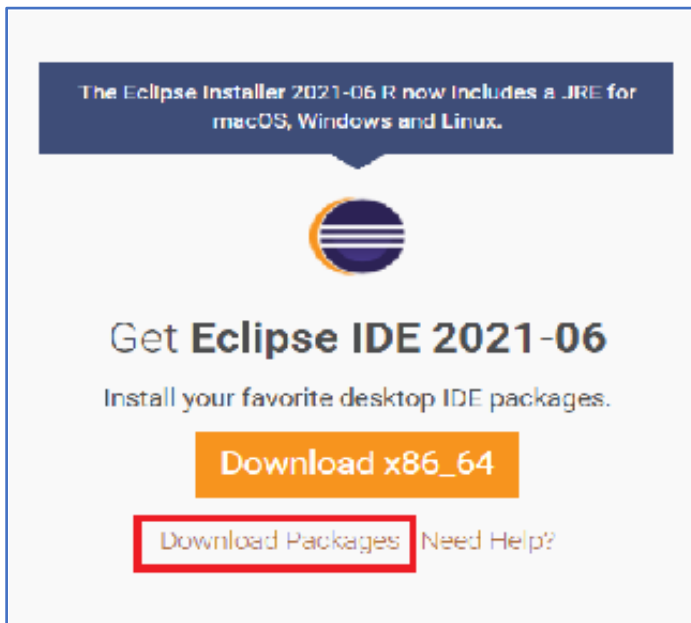
- 무료 오픈 소스 통합 개발 환경(IDE : Integrated Development Environment) 도구이다.
- IDE : 프로젝트 생성, 자동 코드 완성, 디버깅 등과 같이 개발에 필요한 여러가지 기능을 통합적으로 제공해주는 도구이다.

● Eclipse 설치 : Eclipse IDE 2022-03 R Packages

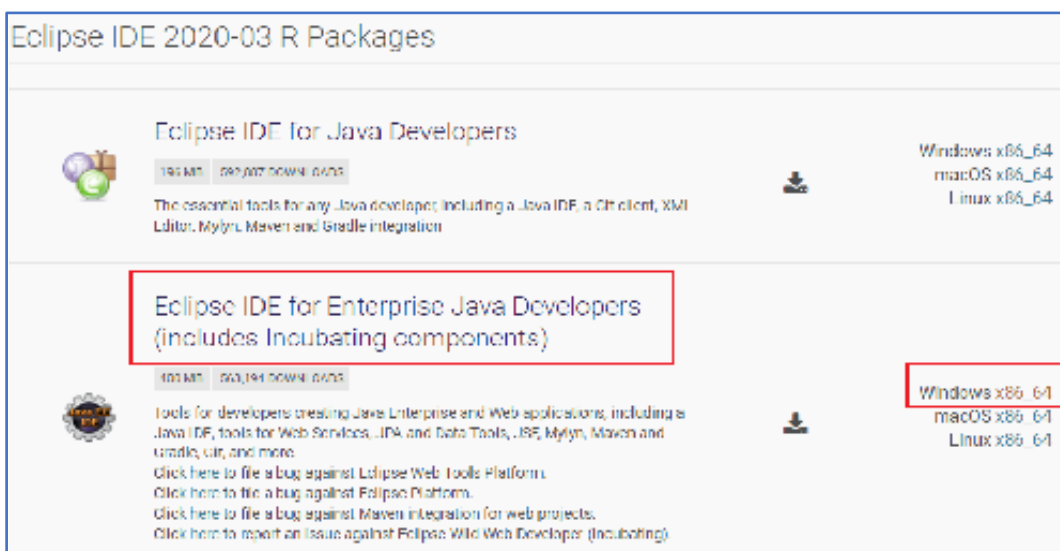
1. <https://www.eclipse.org> 사이트로 이동한다.
2. Download 버튼을 선택한다.



3. Download Packages 선택한다.



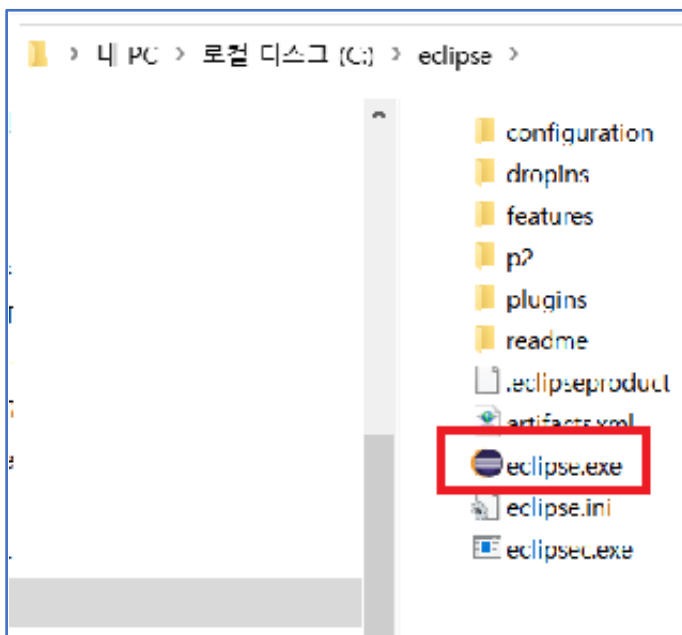
4. Eclipse IDE for Enterprise Java Developers 에서 운영체제에 맞는 항목을 선택한다.



5. eclipse-jee-2020-03-R-incubation-win32-x86_64.zip 파일을 다운로드한다.



6. C:\eclipse 폴더에 설치한다.



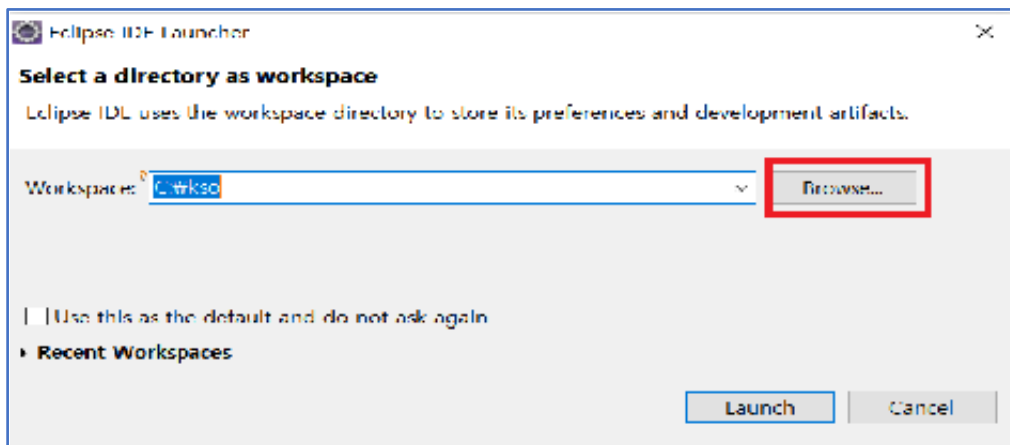
7. eclipse.exe 파일을 실행한다.

8. Browse... 버튼을 클릭하여 워크스페이스를 선택한 후 Launch를 선택한다.

※ 워크스페이스란?

- 프로젝트를 저장하는 작업 폴더이다.

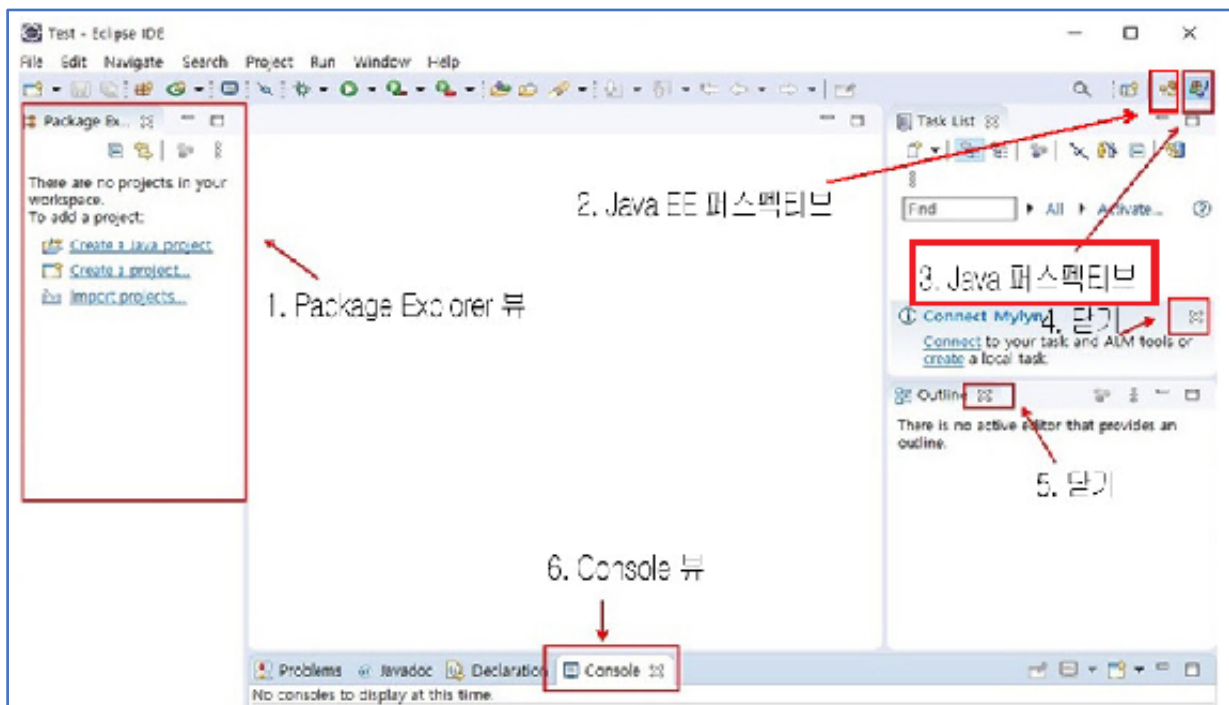
- C:\w 본인이름폴더(ex : kso)



Eclipse 화면 구성

● 퍼스펙티브와 뷰

- 퍼스펙티브는 이클립스에서 유용하게 사용할 수 있는 뷰들의 묶음을 말하며, 뷰는 이클립스 내부에 있는 작은창을 말한다.

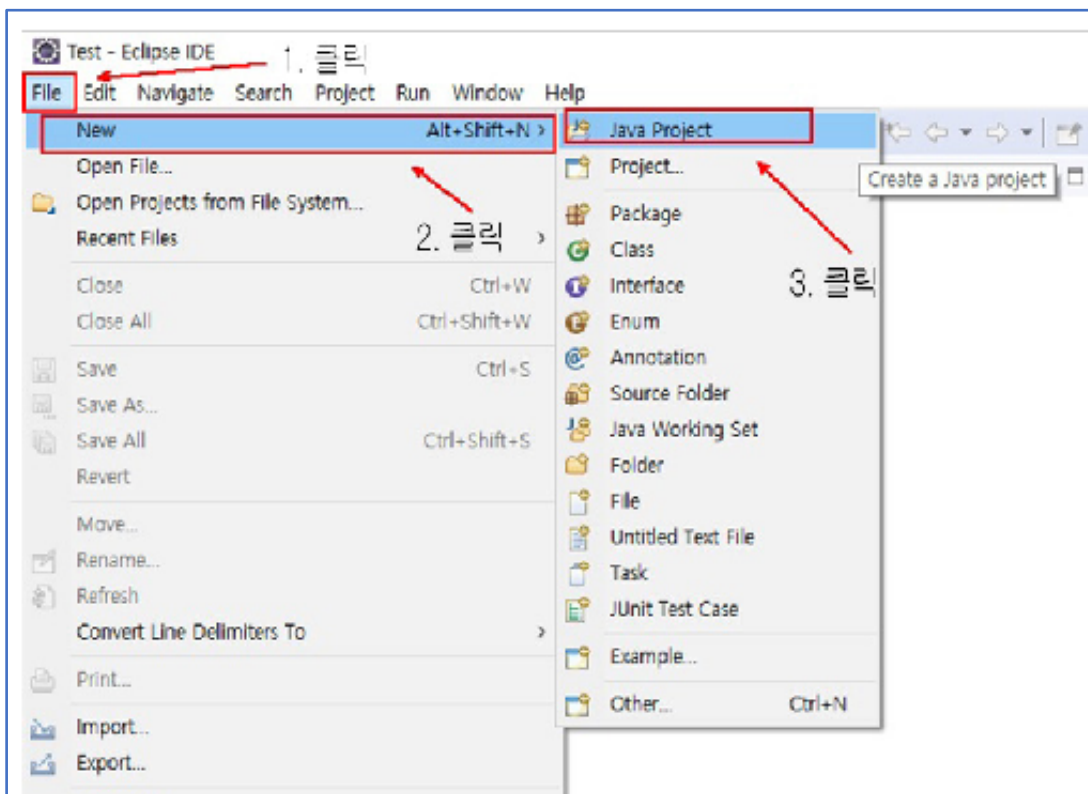


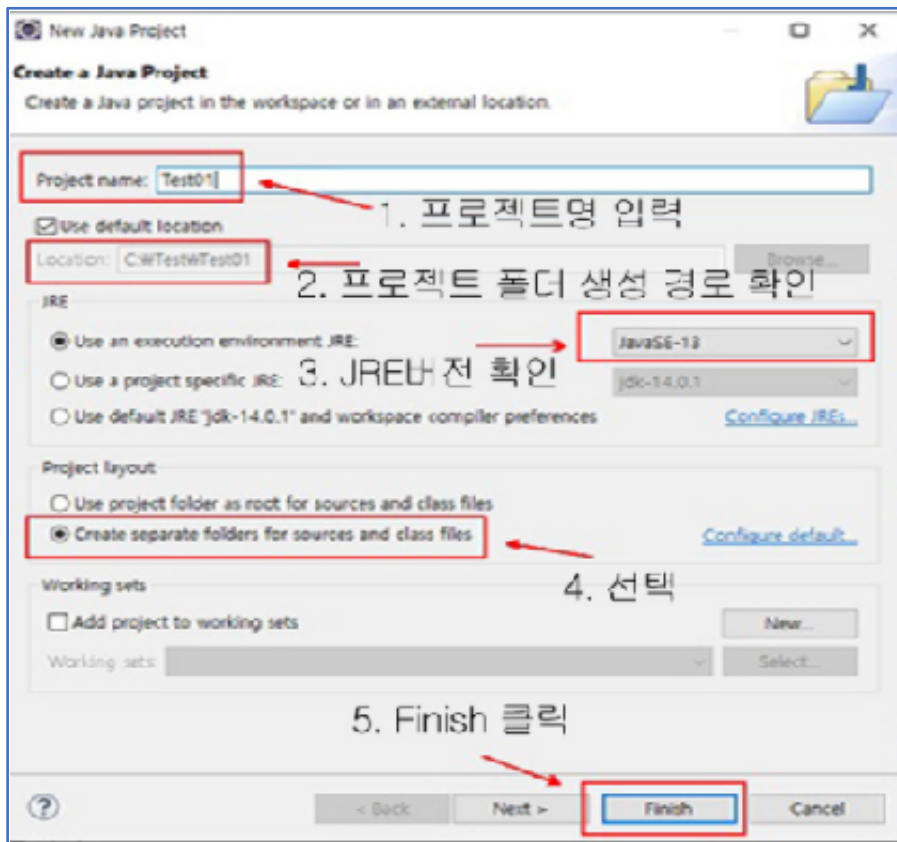
- Java 퍼스펙티브
- Package Explorer 뷰
- Console 뷰

Eclipse 에서 Java Project 생성 및 실행

1. 자바 프로젝트 생성

· File -> New -> Java Project 를 선택한다.

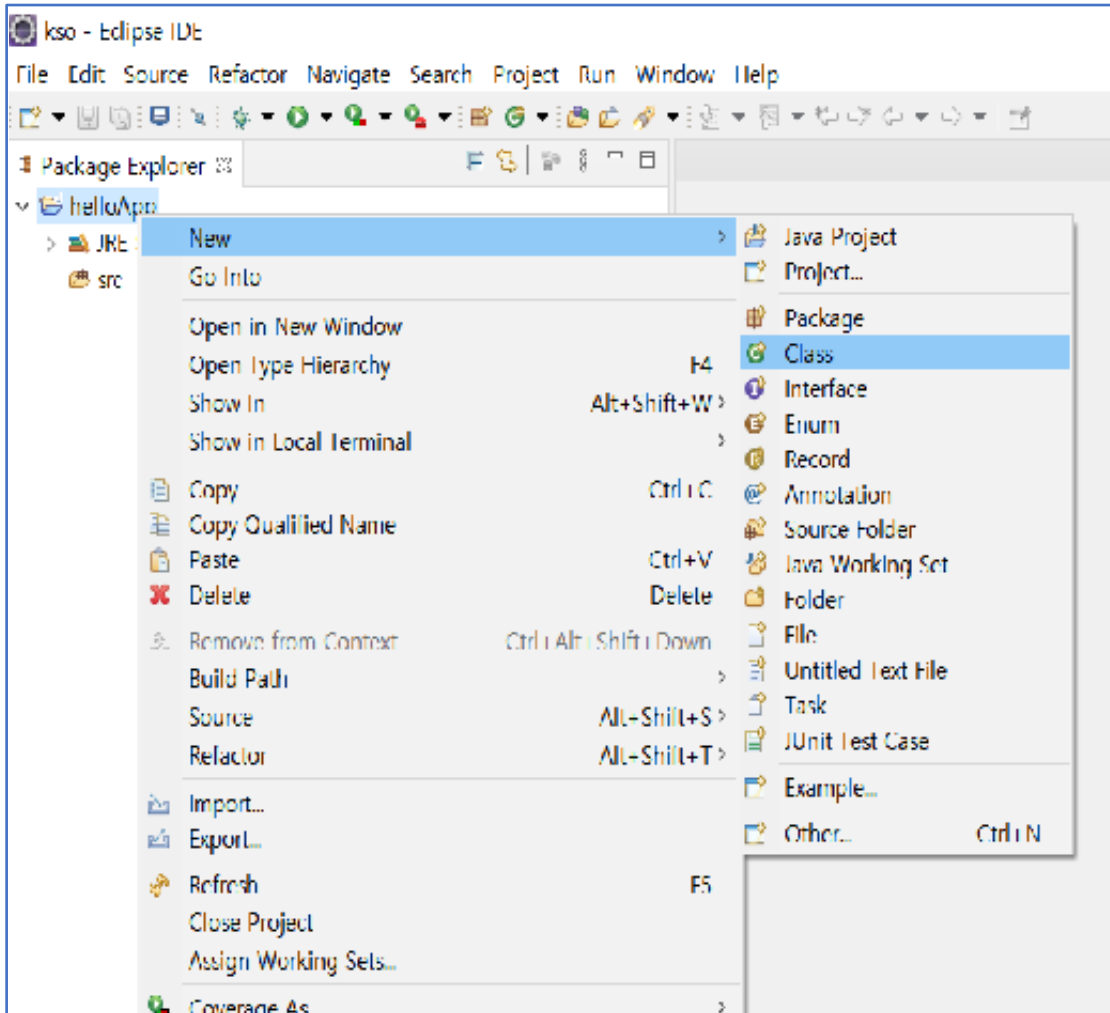




- Project name : HelloApp

2. 소스파일 생성 (Hello.java)

- HelloApp 프로젝트를 선택한 후 New -> Class 를 선택한다.



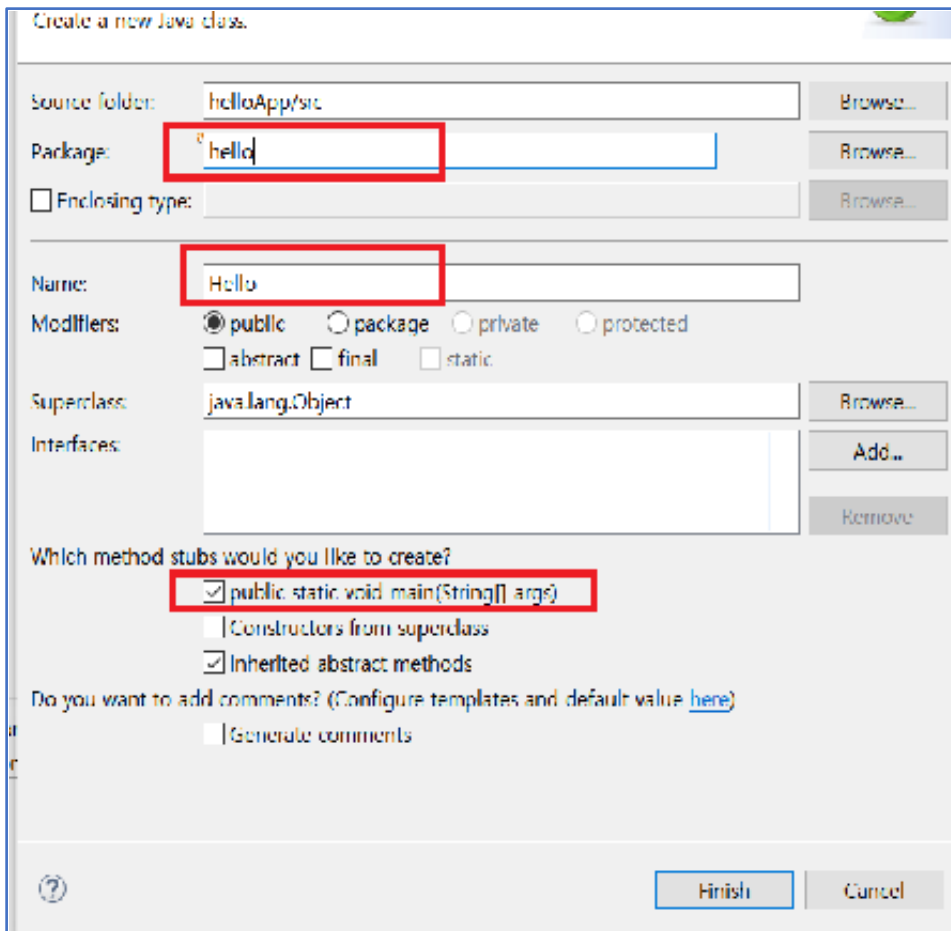
● 자바 클래스 생성

- Package : 클래스를 기능별로 분류하기 위한 폴더를 말한다.

패키지명 (명명규칙 : 첫글자는 소문자로 시작한다.) hello

- Name : 클래스명 (명명규칙 : 첫글자는 대문자로 시작한다.) Hello

- public static void main(String[] args) 항목을 체크한다.



Create a new Java class.

Source folder: Browse...

Package: Browse...

☐ Enclosing type: Browse...

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: Browse...

Interfaces: Add... Remove...

Which method stubs would you like to create?

☒ public static void main(String[] args)

☐ Constructors from superclass

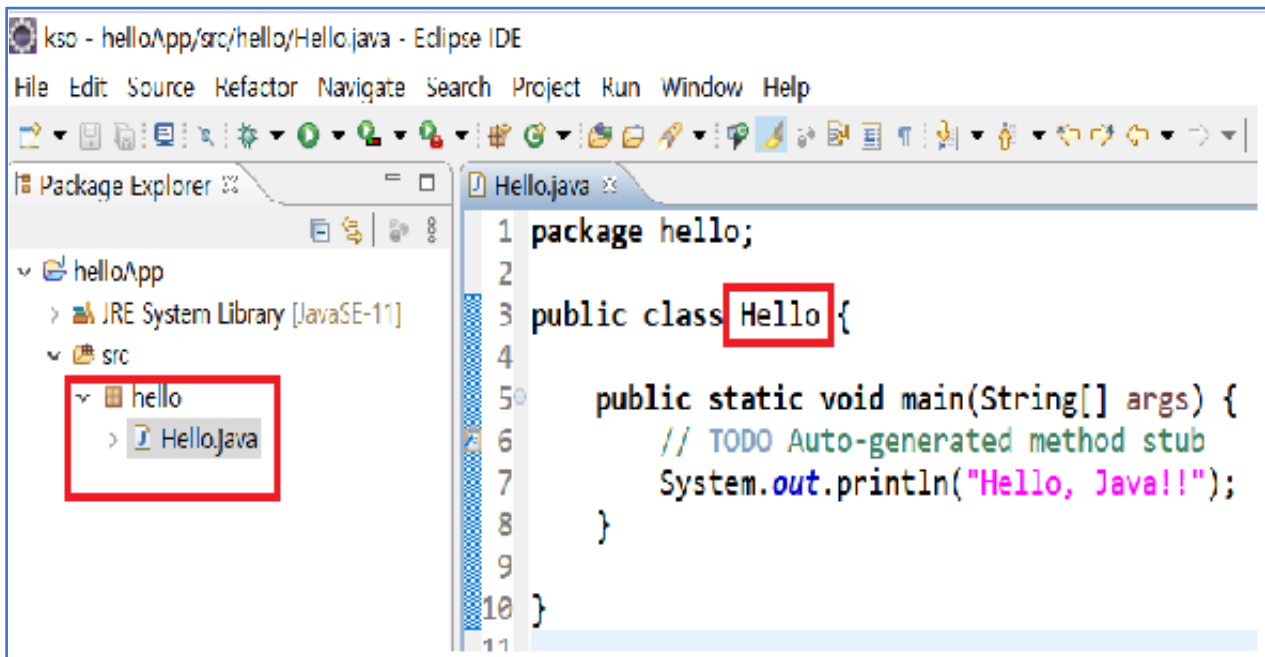
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

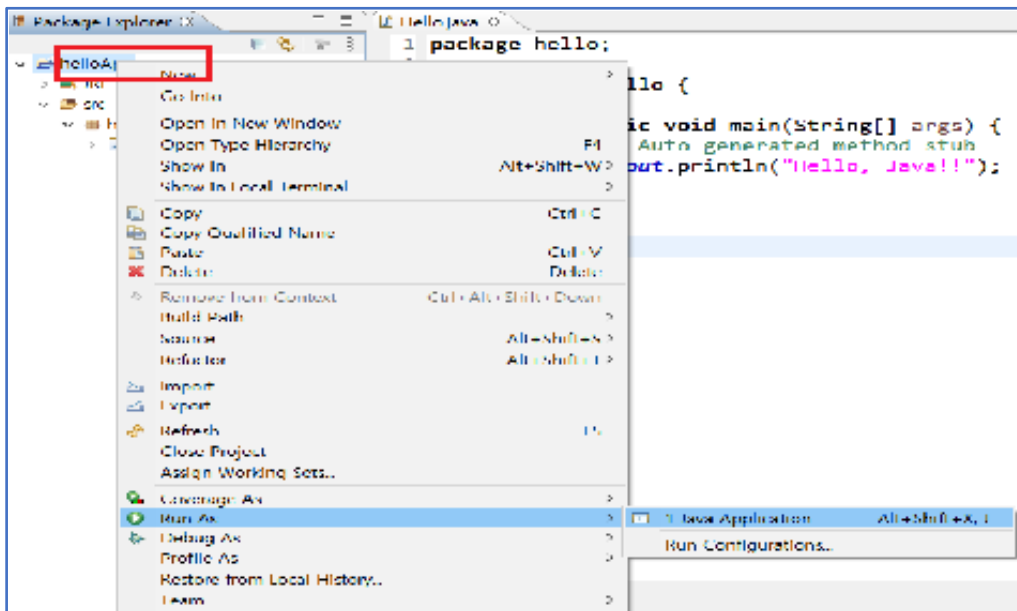
☐ Generate comments

Finish Cancel

3. 소스 코드 작성



4. 프로그램 실행 : CTRL + F11 (단축키)



주석 (Comment)

- 프로그램 소스 코드에 부연 설명을 달 때 사용한다.
- 주석은 프로그램 수행에 영향을 미치지 않는다. 왜냐하면 컴파일 시 제외된다.
- 자바에는 두 가지 형태의 주석이 있다.

- 라인 주석 : 이클립스 주석달기 단축키 (Ctrl + /)

```
ex] int age;    // 나이
```

- 블록 주석 : 이클립스 주석달기 단축키 (Ctrl + Shift + /)

```
/*
```

```
    작성자 : 홍길동
```

```
    작성일자 : 2021-01-01
```

```
    프로그램 설명 :
```

```
*/
```

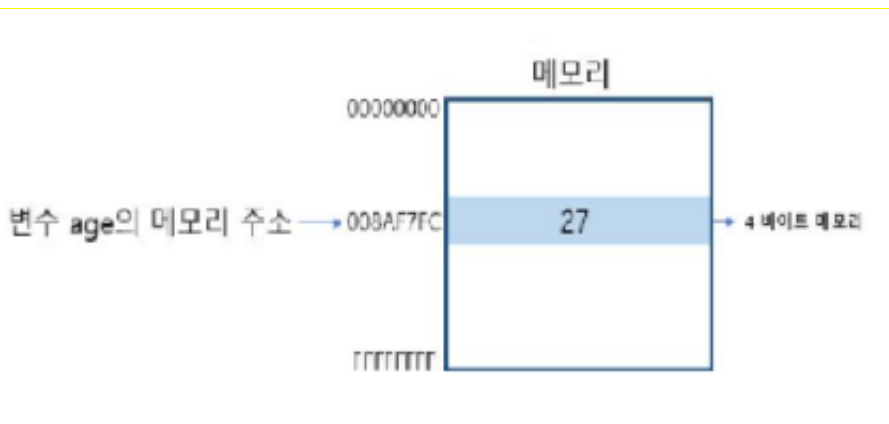
변수 (Variable)

- 변수는 데이터를 저장할 수 있는 메모리 공간을 의미하며, 저장된 값은 변경될 수 있다.

● 변수 선언 및 값 할당

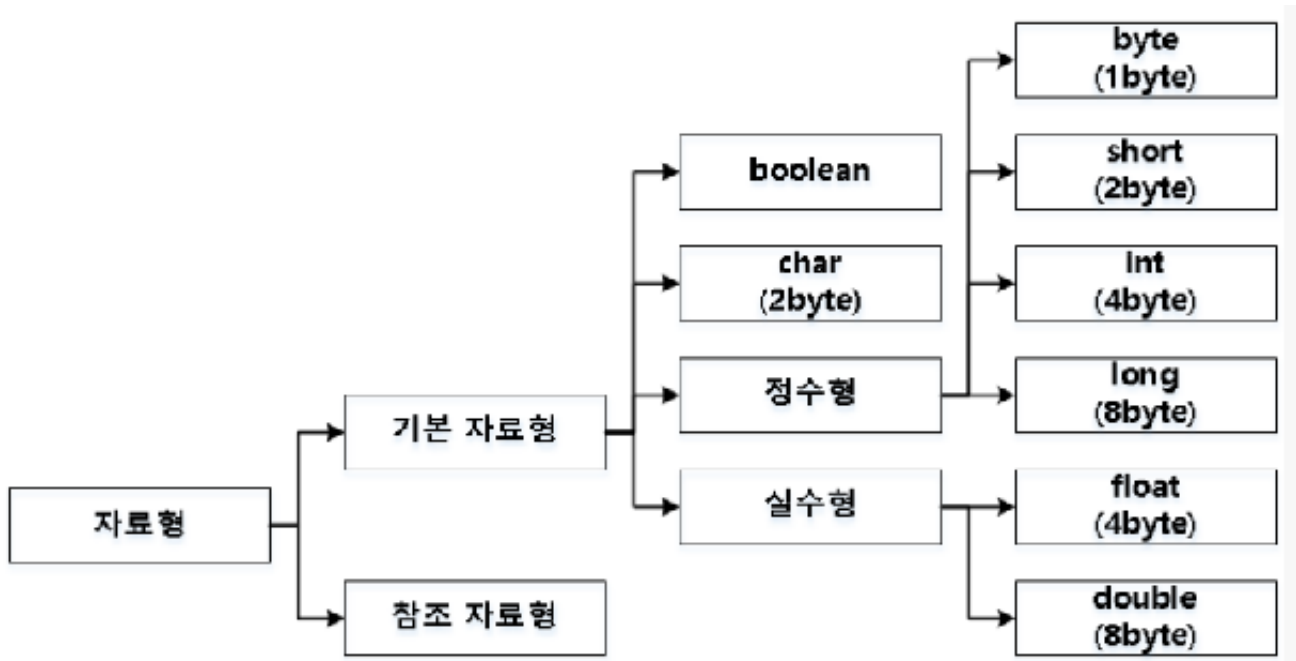
```
int age = 27 ;
```

● 변수와 메모리



- 하나의 메모리 공간에는 1 바이트의 데이터가 저장된다.
- 자료형이 int 이므로 4 바이트 공간이 할당된다.
- 변수의 이름은 첫 번째 메모리 주소인 008AF7FC 만을 가리키게 된다.

자료형 (Data Type)



기본 자료형 (Primitive Data Type)

- 자바에서 여러 형태의 데이터 타입을 미리 정의(Built-in)하여 제공한다.
- 실제 값(리터럴)을 저장하는 자료형이다.

● char 형

- 문자 리터럴은 유니코드로 변환되어 저장된다.
- char 는 정수 타입이므로 10 진수, 16 진수 형태의 유니코드 저장 가능하다.

※ 문자 인코딩

- 문자 셋(문자 집합, Character set, Charset)은 정보를 표현하기 위한 글자나 기호들의 집합을 정의한 것이다.
- 문자 인코딩은 문자 셋을 컴퓨터에서 표현하기 위해 약속된 규칙에 따라 변환하는 과정을 말한다. ASCII(아스키), 유니코드

● ASCII

· 미국 표준 정보 교환 코드

· 7 비트 표현 방식으로 27 총 128 개의 문자를 표현할 수 있다. (0- 127 코드값)

10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자
0	0X00	NULL	16	0X10	DLE	32	0x20	SP	48	0x30	0
1	0X01	SOH	17	0X11	DC1	33	0x21	!	49	0x31	1
2	0X02	STX	18	0X12	SC2	34	0x22	"	50	0x32	2
3	0X03	ETX	19	0X13	SC3	35	0x23	#	51	0x33	3
4	0X04	EOT	20	0X14	SC4	36	0x24	\$	52	0x34	4
5	0X05	ENQ	21	0X15	NAK	37	0x25	%	53	0x35	5
6	0X06	ACK	22	0X16	SYN	38	0x26	&	54	0x36	6
7	0X07	BEL	23	0X17	ETB	39	0x27	'	55	0x37	7
8	0X08	BS	24	0X18	CAN	40	0x28	(56	0x38	8
9	0X09	HT	25	0x19	EM	41	0x29)	57	0x39	9
10	0X0A	LF	26	0x1A	SUB	42	0x2A	*	58	0x3A	:
11	0X0B	VT	27	0x1B	ESC	43	0x2B	+	59	0x3B	;
12	0X0C	FF	28	0x1C	FS	44	0x2C	,	60	0x3C	<
13	0X0D	CR	29	0x1D	GS	45	0x2D	-	61	0x3D	=
14	0X0E	SO	30	0x1E	RS	46	0x2E	.	62	0x3E	>
15	0X0F	SI	31	0x1F	US	47	0x2F	/	63	0x3F	?

10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자
64	0x40	@	80	0x50	P	96	0x60	`	112	0x70	p
65	0x41	A	81	0x51	Q	97	0x61	a	113	0x71	q
66	0x42	B	82	0x52	R	98	0x62	b	114	0x72	r
67	0x43	C	83	0x53	S	99	0x63	c	115	0x73	s
68	0x44	D	84	0x54	T	100	0x64	d	116	0x74	t
69	0x45	E	85	0x55	U	101	0x65	e	117	0x75	u
70	0x46	F	86	0x56	V	102	0x66	f	118	0x76	v
71	0x47	G	87	0x57	W	103	0x67	g	119	0x77	w
72	0x48	H	88	0x58	X	104	0x68	h	120	0x78	x
73	0x49	I	89	0x59	Y	105	0x69	i	121	0x79	y
74	0x4A	J	90	0x5A	Z	106	0x6A	j	122	0x7A	z
75	0x4B	K	91	0x5B	[107	0x6B	k	123	0x7B	{
76	0x4C	L	92	0x5C	\	108	0x6C	l	124	0x7C	
77	0x4D	M	93	0x5D]	109	0x6D	m	125	0x7D	}
78	0x4E	N	94	0x5E	^	110	0x6E	n	126	0x7E	~
79	0x4F	O	95	0x5F	_	111	0x6F	o	127	0x7F	DEL

영문 대문자 'X' ----->>> 10 진수 아스키 코드값 (88)

10 진수 (88)----->>> 2 진수 변환 (1011000)

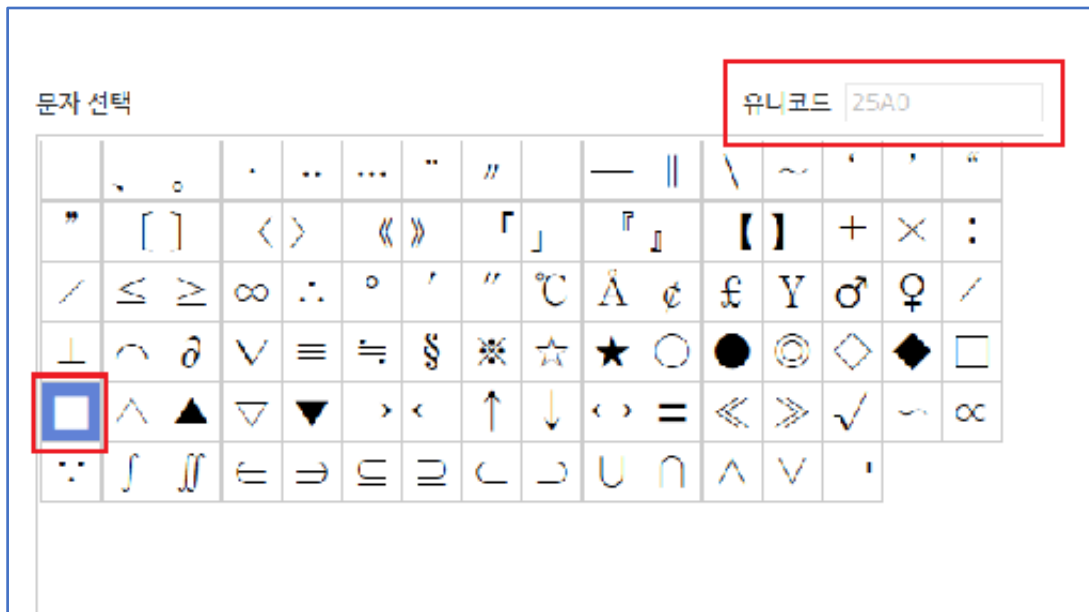
2	88	
2	44	... 0
2	22	... 0
2	11	... 0
2	5	... 1
2	2	... 1
2	1	... 0

● UNICODE

- 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현할 수 있다.
- 16 비트 표현 방식으로 2^{16} 총 65,536 개의 문자를 표현할 수 있다.

(0- 65535 코드값)

ex] char ch = 'Wu2605' ;



※ Escape 문자

- 문자열 내에서 사용하는것으로 특수 기능을 제어할 때 사용한다.

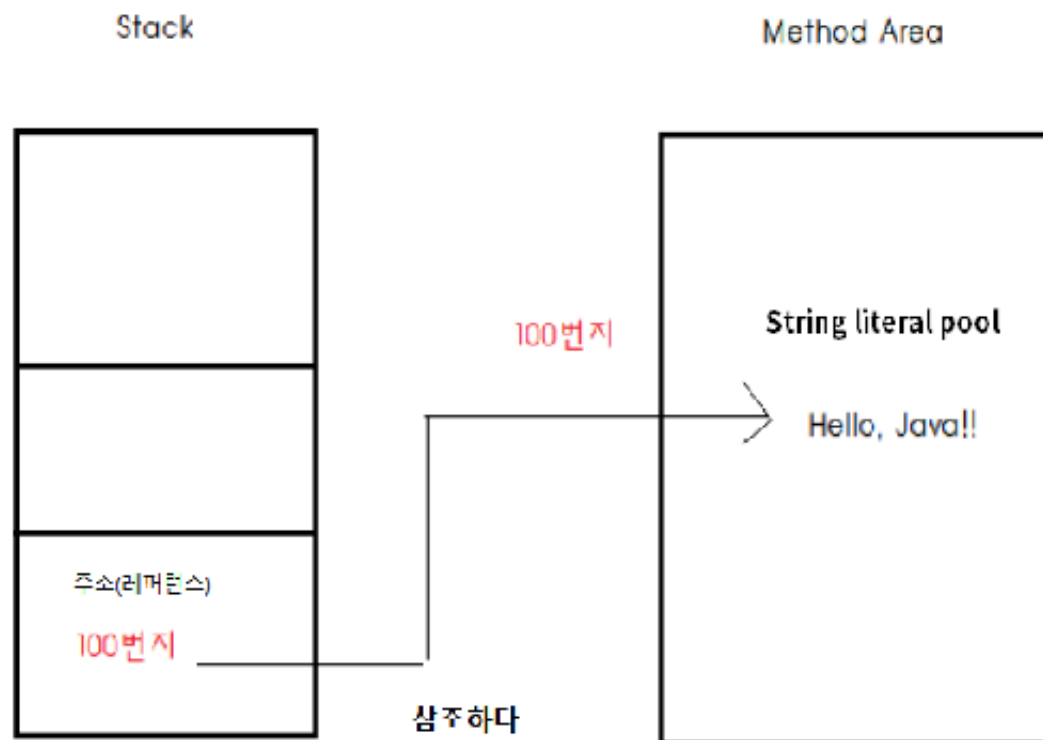
\n	개행 (줄바꿈)
\t	탭 (8 공백)
\'	작은 따옴표 (싱글쿼테이션) 표시
\"	큰 따옴표 (더블쿼테이션) 표시
\\	역슬래시 표시

참조형 (Reference Data Type)

- 객체의 저장 공간인 주소를 저장하고, 주소를 통해 객체를 참조하는 타입이다.
- 클래스, 인터페이스, 배열, 열거형(Enum)

```
String str = "Hello, Java!!"
```

클래스 레퍼런스 변수



리터럴 (literal)

- 소스 코드의 고정된 값이다.
- 정수, 실수, 문자, 논리, 문자열 리터럴 등이 있다.
- 정수형 리터럴

10진수

8진수(숫자 0으로 시작) 012

16진수(숫자 0x) 0x123

2진수(0b) 이 있다. 0b1

- 정수형 리터럴은 int 형으로 컴파일되고, long형 리터럴은 숫자뒤에 L / l을 붙인다.
- 실수형 리터럴은 double 형으로 컴파일되고, float형 리터럴은 숫자뒤에 F/ f을 붙인다.
- 문자형 리터럴은 단일 인용부호를 붙인다.
- 문자열형 리터럴은 이중 인용부호를 붙인다.
- 논리형 리터럴은 true, false 가 있다.

타입 변환 (Data Type Conversion)

- 하나의 자료형을 다른 자료형으로 바꾸는 것을 말한다.
- 자바에서는 boolean 형을 제외한 나머지 기본 자료형 간의 타입 변환을 수행할 수 있다.

● 자동 타입 변환 (Promotion)

- 작은 메모리 크기의 데이터 타입을 큰 메모리 크기의 데이터 타입으로 변환하는 것이다.
- 연산을 수행 시 컴파일러가 자동으로 타입 변환을 수행해준다.

byte (1) < short (2) < int (4) < long (8) < float(4) < double(8)

● 강제 타입 변환(Casting)

- 사용자가 캐스트 연산자를 사용하여 강제적으로 타입 변환을 수행한다.
- 작은 메모리 크기의 데이터 타입 = (작은 메모리 크기의 데이터 타입) 큰 메모리 크기의 데이터 타입
- 데이터의 손실이 발생할 수 있다.

● 자바 정수 연산에서의 자동 타입 변환

- 정수형 변수가 산술 연산식에서 피연산자로 사용되면 int 타입보다 작은 byte, short 형 변수는 int 형으로 자동 타입 변환된다.
- 두 피연산자 중 int 형 보다 큰 경우 큰 자료형으로 변환되어 연산 된다.

● + 연산에서의 문자열 자동 타입 변환

- 피연산자 중 하나가 문자열일 경우 나머지 피연산자도 문자열로 자동 변환되고 문자열 결합 연산을 수행한다.

● 문자열을 기본 자료형으로 강제 변환

<pre>String str = "12345"; int num = Integer.parseInt(str);</pre>	String -> int
<pre>String str = "123.45"; double num = double.parseDouble(str);</pre>	String -> double

콘솔 입출력

● Scanner 클래스

- 읽은 바이트를 문자, 정수, 실수, 불린, 문자열 등 다양한 타입으로 변환하여 리턴한다.
- 입력되는 데이터를 공백(whitespace) 또는 개행(줄바꿈) 으로 구분되는 토큰 단위로 읽는다.

```
Scanner scan = new Scanner ( System.in );
```

메소드 명	설명
next()	다음 토큰을 문자열로 반환 (공백을 기준으로 한 단어를 읽음)
nextLine()	문자열 반환 (개행을 기준으로 한 줄을 읽음)
nextInt()	다음 토큰을 int형 으로 반환
nextDouble()	다음 토큰을 double형 으로 반환
hasNext()	현재 입력된 토큰이 있으면 true, 아니면 무한 대기 상태 그러다 입력이 들어오면 true, 입력이 완료되면 false를 반환

※ Console 이란?

- 시스템을 사용하기 위해 키보드로 입력을 받고 화면으로 출력하는 소프트웨어로 리눅스 or 유닉스 : 터미널 / 윈도우 : 명령 프롬프트 / 이클립스 : Console 뷰

※ Enter의 아스키코드 값

- Enter 는 Windows 에서는 $\backslash r \backslash n$ 이며 $\backslash r$ 은 아스키코드 값 13 CR 을 나타낸다. CR 은 Carriage Return 의 약자로 커서 표시 위치를 같은 줄(행) 맨 앞의 위치로 복귀시키는 것을 의미한다.
 $\backslash n$ 은 아스키코드값 10 LF 를 나타내고, LF 는 Line Feed 의 약자로 모니터의 커서 위치나 프린터의 인쇄 위치를 한 줄 아래로 내리는 일을 의미한다.

● System.out.printf (“출력 서식”, 출력할 내용)

`%[argument_index$][flags][width][.precision]conversion`

- 출력 서식의 지시자(conversion)를 제외한 나머지는 생략 가능하다.
- width : 출력할 전체 자리수 지정(오른쪽 정렬).
예) %3d, 전체자리수가 3 인 정수
- (flags) 0 : 전체 자리수가 지정된 경우 왼쪽의 남은 자리에 0 을 출력. 예) %03d
- (flags) - : 전체 자리수가 지정된 경우 왼쪽 정렬하고 빈칸에 공백 출력.
- .precision : 소수점 아래 자리수 지정. 잘리는 소수점 자리수는 반올림하여 출력한다.

※ 지시자(Conversion)

지시자	설명
%b	불리언(boolean) 형식으로 출력
%d	10진(decimal) 정수의 형식으로 출력
%o	8진(octal) 정수의 형식으로 출력
%x, %X	16진(hexa_decimal) 정수의 형식으로 출력
%f	부동 소수점의 형식으로 출력
%e, %E	지수 표현식의 형식으로 출력
%c	문자로 출력
%s	문자열로 출력

연산자 (Operator)

● 단항 연산자

- 항이 하나인 연산자이다.

연산자	의미	사용법	설명
<code>+, -</code>	부호 연산자	<code>+var, -var</code>	변수 <code>var</code> 의 부호를 바꾼다.
<code>!</code>	부정 연산자	<code>!var</code>	참은 거짓 반대로 거짓은 참으로 바꾼다.
<code>++</code>	증가 연산자	<code>++var; , var++;</code>	변수 <code>var</code> 의 값을 1 증가한다.
<code>--</code>	감소 연산자	<code>--var; , var--;</code>	변수 <code>var</code> 의 값을 1 감소한다.

● 이항 연산자

- 항이 두개인 연산자이다.

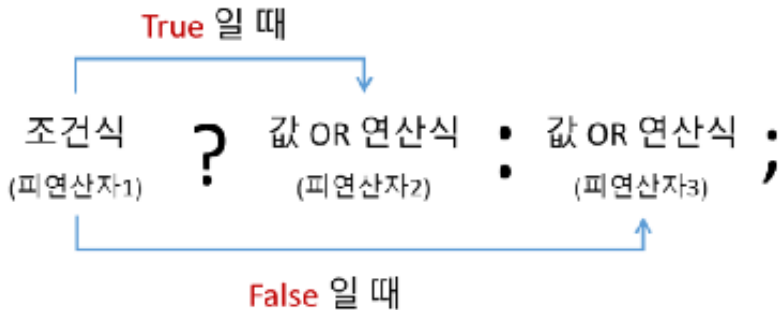
산술 연산자

연산식			설명
피연산자	+	피연산자	덧셈 연산
피연산자	-	피연산자	뺄셈 연산
피연산자	*	피연산자	곱셈 연산
피연산자	/	피연산자	좌측 피연산자를 우측 피연산자로 나눴셈 연산
피연산자	%	피연산자	좌측 피연산자를 우측 피연산자로 나눈 나머지를 구하는 연산

비교 연산자

구분	연산식			설명
동등 비교	피연산자	==	피연산자	두 피 연산자의 값이 같은지를 검사
	피연산자	!=	피연산자	두 피 연산자의 값이 다른지를 검사
크기 비교	피연산자	>	피연산자	피 연산자 1 이 큰지를 검사
	피연산자	>=	피연산자	피 연산자 1 이 크거나 같은지를 검사
	피연산자	<	피연산자	피 연산자 1 이 작은지를 검사
	피연산자	<=	피연산자	피 연산자 1 이 작거나 같은지를 검사

● 삼항 연산자



● 대입 연산자

구분	연산식			설명
단순 대입 연산자	변수	=	피연산자	우측의 피연산자의 값을 변수에 저장
복합 대입 연산자	변수	+=	피연산자	우측의 피연산자의 값을 변수의 값과 더한 후에 다시 변수에 저장 (변수=변수+피연산자 와 동일)
	변수	-=	피연산자	우측의 피연산자의 값을 변수의 값에서 뺀 후에 다시 변수에 저장 (변수=변수-피연산자 와 동일)
	변수	*=	피연산자	우측의 피연산자의 값을 변수의 값과 곱한 후에 다시 변수에 저장 (변수=변수*피연산자 와 동일)
	변수	/=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 다시 변수에 저장 (변수=변수/피연산자 와 동일)
	변수	%=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 나머지를 변수에 저장 (변수=변수%피연산자 와 동일)
	변수	&=	피연산자	우측의 피연산자의 값과 변수의 값을 & 연산 후 결과를 변수에 저장 (변수=변수&피연산자 와 동일)
	변수	=	피연산자	우측의 피연산자의 값과 변수의 값을 연산 후 결과를 변수에 저장 (변수=변수 피연산자 와 동일)

● 논리 연산자

구분	연산식			결과	설명
AND (논리곱)	true	&& 또는 &	true	true	피연산자 모두가 true일 경우만 연산 결과는 true
	true		false	false	
	false		true	false	
	false		false	false	
OR (논리합)	true	 또는 	true	true	피연산자중 하나만 true이면 연산 결과는 true
	true		false	true	
	false		true	true	
	false		false	false	
XOR (배타적 논리합)	true	^	true	false	피 연산자가 하나는 true 이고 다른 하나가 false 일 경우에만 연산 결과는 true
	true		false	true	
	false		true	true	
	false		false	false	
NOT (논리부정)		!	true	false	피 연산자의 논리값을 바꿈
			false	true	

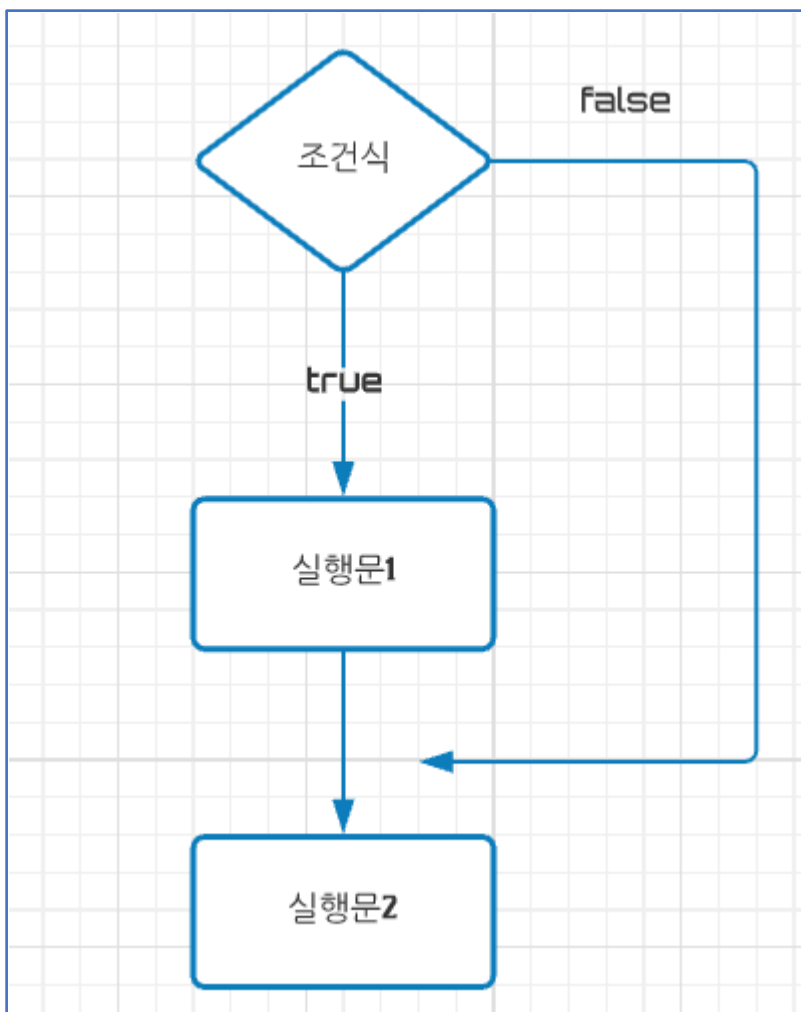
제어문

- 프로그램의 동작 흐름에 조건 / 반복을 통해 제어할 수 있는 실행문이다.
- 제어문에는 조건문과 반복문이 있다.

조건문

- 조건식에 따라 다른 명령문을 실행하기 위해서 사용한다.

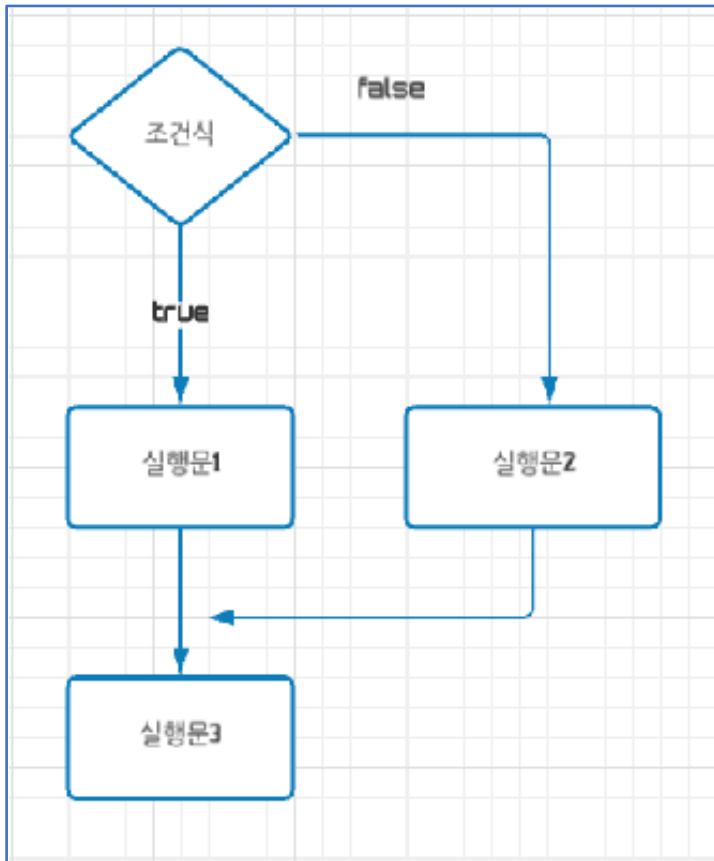
1. if문



```

if(조건식) {
    실행문1;
}
실행문2;
  
```

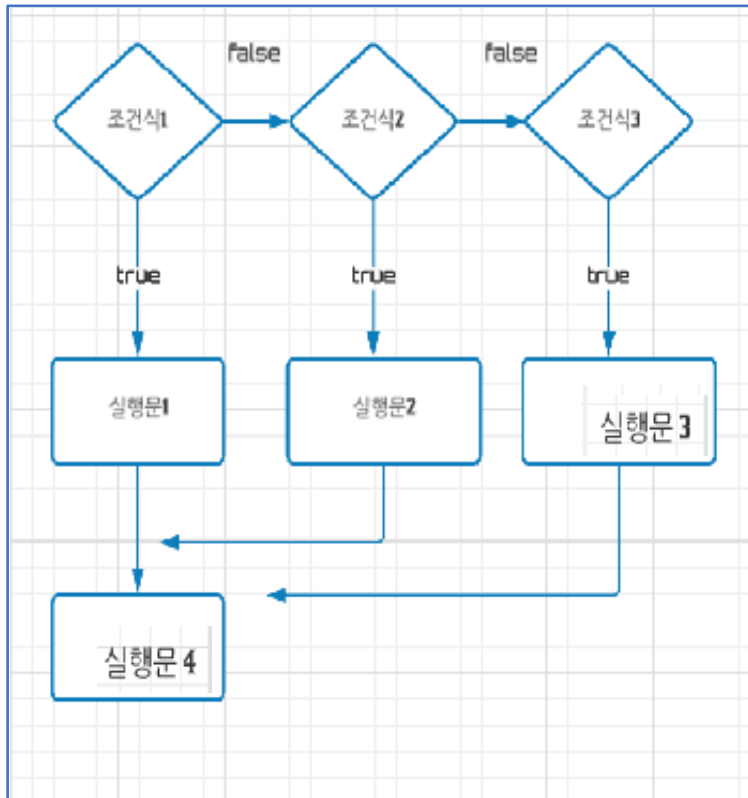
2. if-else문 (양자 택일)



```

if(조건식) {
    실행문1;
} else {
    실행문2;
}
실행문3;
  
```

3. 다중 if-else문 (다중선택)

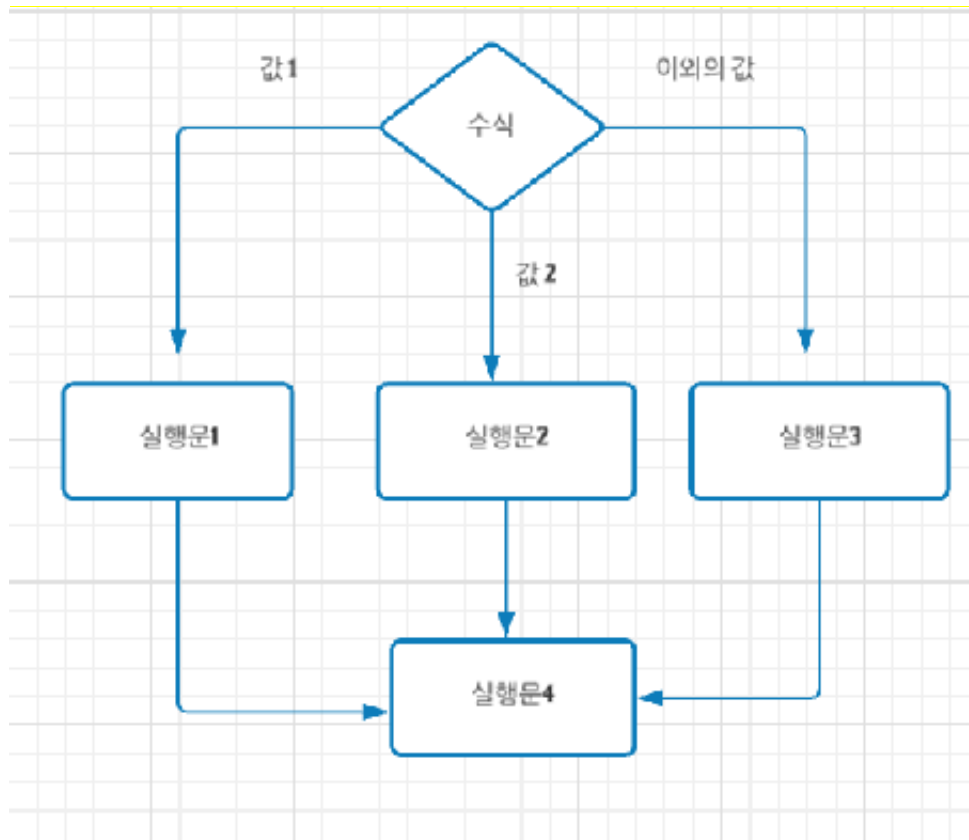


```

if(조건식1) {
    실행문1;
} else if (조건식2) {
    실행문2;
} else if(조건식3) {
    실행문3;
}
실행문4;
  
```

4. swich-case문

※ 수식에는 정수형, 문자열형, 열거형만 올 수 있다.



```

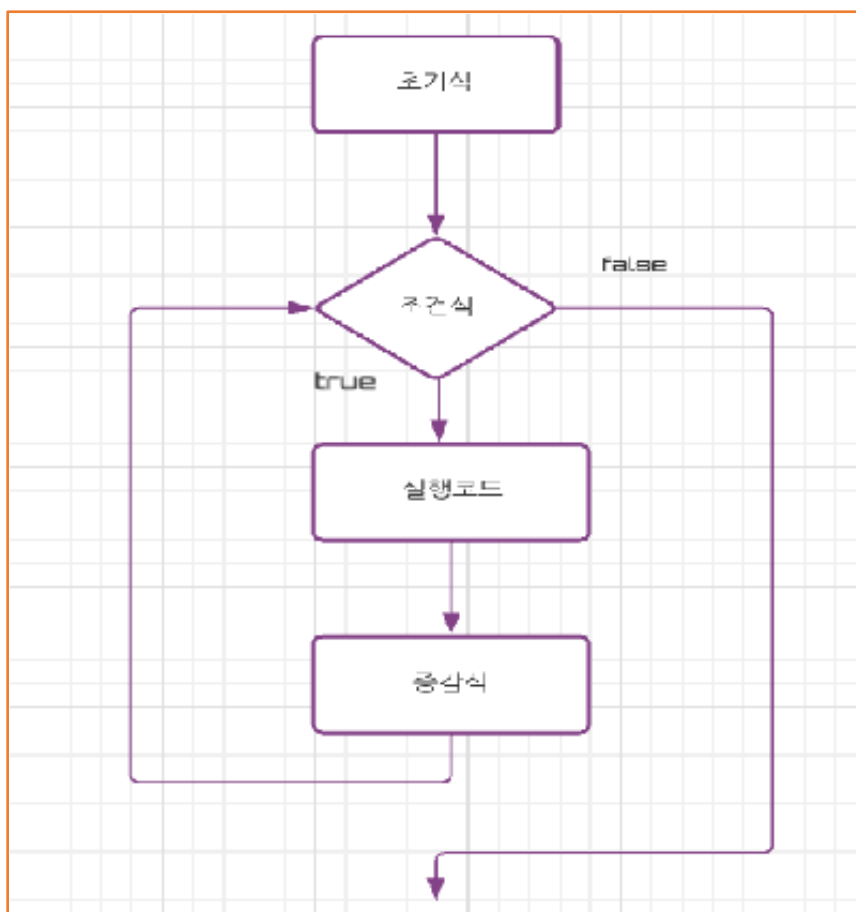
switch( 수식 ) {
    case 값1: 실행문1; break;
    case 값2: 실행문2; break;
    default: 실행문3;
}
실행문4;
  
```

반복문 - for 문

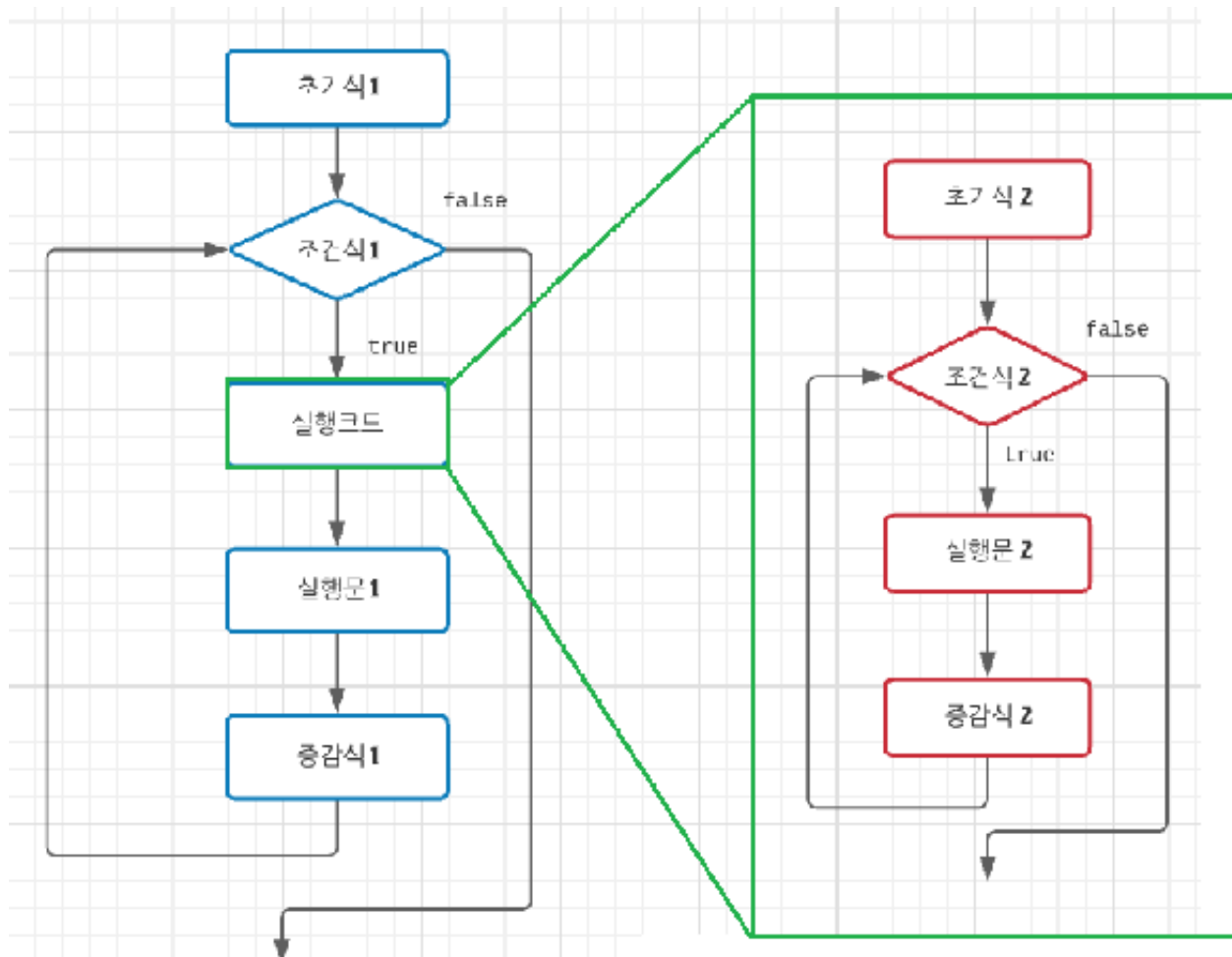
```
for ( 초기식; 조건식; 증감식 ) {
```

조건식의 결과가 참인 동안 반복적으로 실행하고자 하는 명령문;

```
}
```



다중 for문



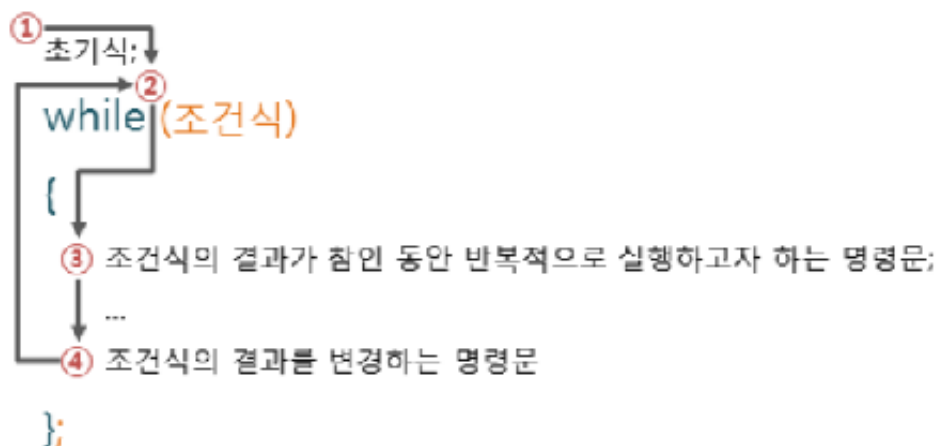
while 문

- 특정 조건을 만족할 때까지 계속해서 주어진 명령문을 반복 실행한다.
- while 문은 루프에 진입하기 전에 먼저 조건식부터 검사한다. (선조건 후수행)

```
while ( 조건식 ) {
```

조건식의 결과가 참인 동안 반복적으로 실행하고자 하는 명령문;

```
}
```



do~while 문

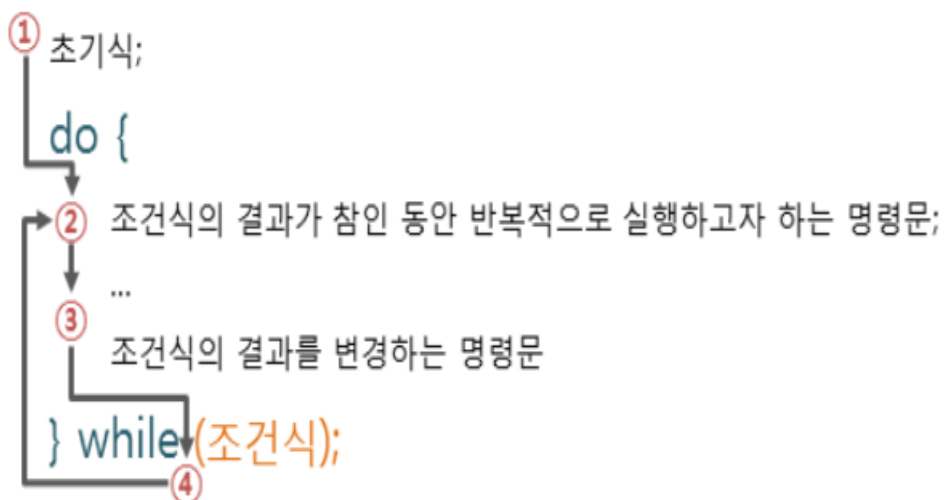
· do / while 문은 먼저 루프를 한 번 실행한 후에 조건식을 검사한다.

(선수행 후조건)

do {

조건식의 결과가 참인 동안 반복적으로 실행하고자 하는 명령문;

} while (조건식);



break 문

- 문 또는 label문을 종료하고, 그 다음 문으로 프로그램 제어를 넘긴다.

- break [label] ;

continue 문

- 반복문 (while, do-while, for)을 다시 시작하기 위해 사용된다.

- continue 문을 사용하는 경우 가장 안쪽의 while, do-while, for 문을 둘러싼 현재 반복을 종료하고, 다음 반복으로 루프의 실행을 계속한다. break문과 달리, continue 문은 전체 루프의 실행을 종료하지 않는다. while 루프에서 그것은 다시 조건으로 이동하고, for 루프에서 그것은 증가 표현으로 이동한다.

배열 (Array)

- 인덱스와 인덱스에 대응하는 데이터들로 이루어진 자료 구조를 나타낸다.
- 일반적으로 배열에는 같은 종류의 데이터들이 순차적으로 저장된다.
- 자바에서는 배열의 길이를 구하기 위해서 length 속성을 사용한다.

● 배열 선언 및 생성

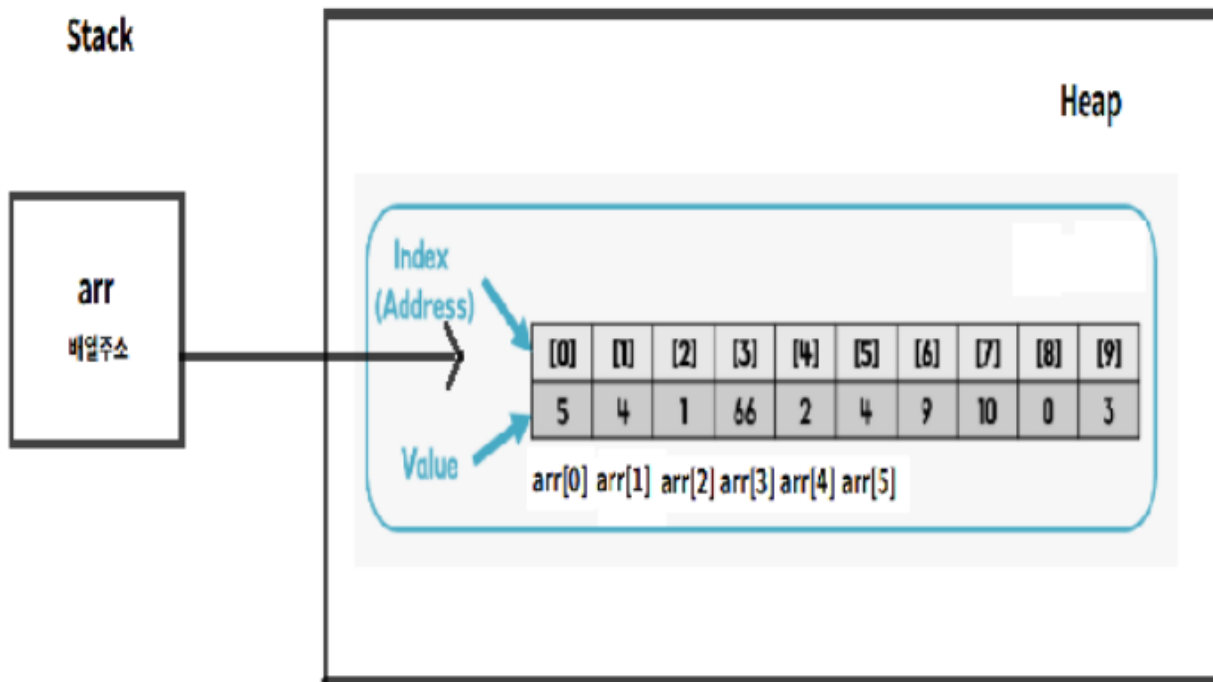
```
자료형[ ] 배열이름 = new 자료형 [배열길이];
```

● 배열 초기화

```
자료형[ ] 배열이름 = { 배열요소 1, 배열요소 2, ... };
```

● 예시 (1 차원 배열)

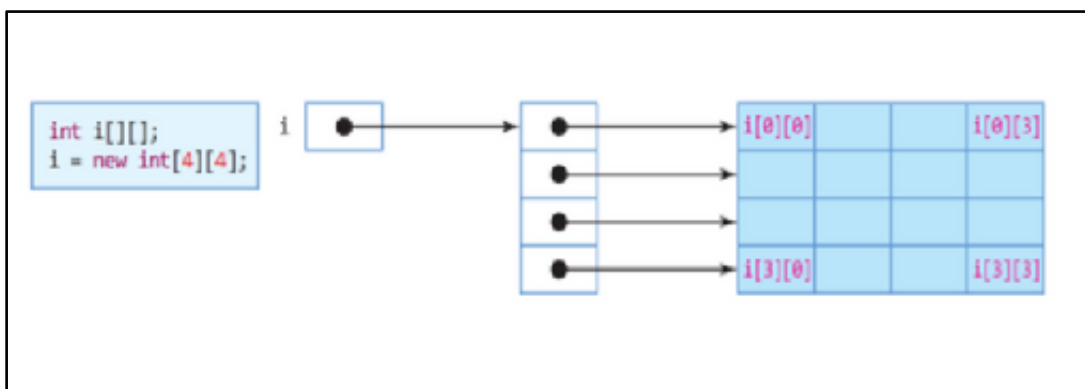
```
int[] arr = { 5, 4, 1, 66, 2, 4, 9, 10, 0, 3 } ;
```



● 다차원 배열

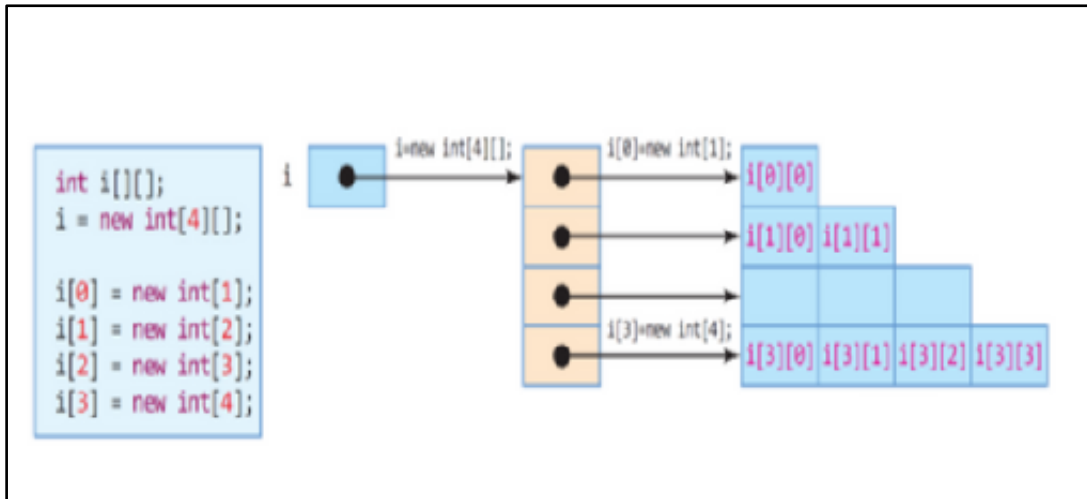
```
int[ ][ ] i = new int[4][4]; // 이차원 배열 선언 및 생성
```

● 정방형 배열



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

● 비정방형 배열



```
int[ ][ ] i = new int[4][ ] ;
```

```
i[0] = new int[1] ;
```

```
i[1] = new int[2] ;
```

```
i[2] = new int[3] ;
```

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


for each 구문 (Enhanced for 문)

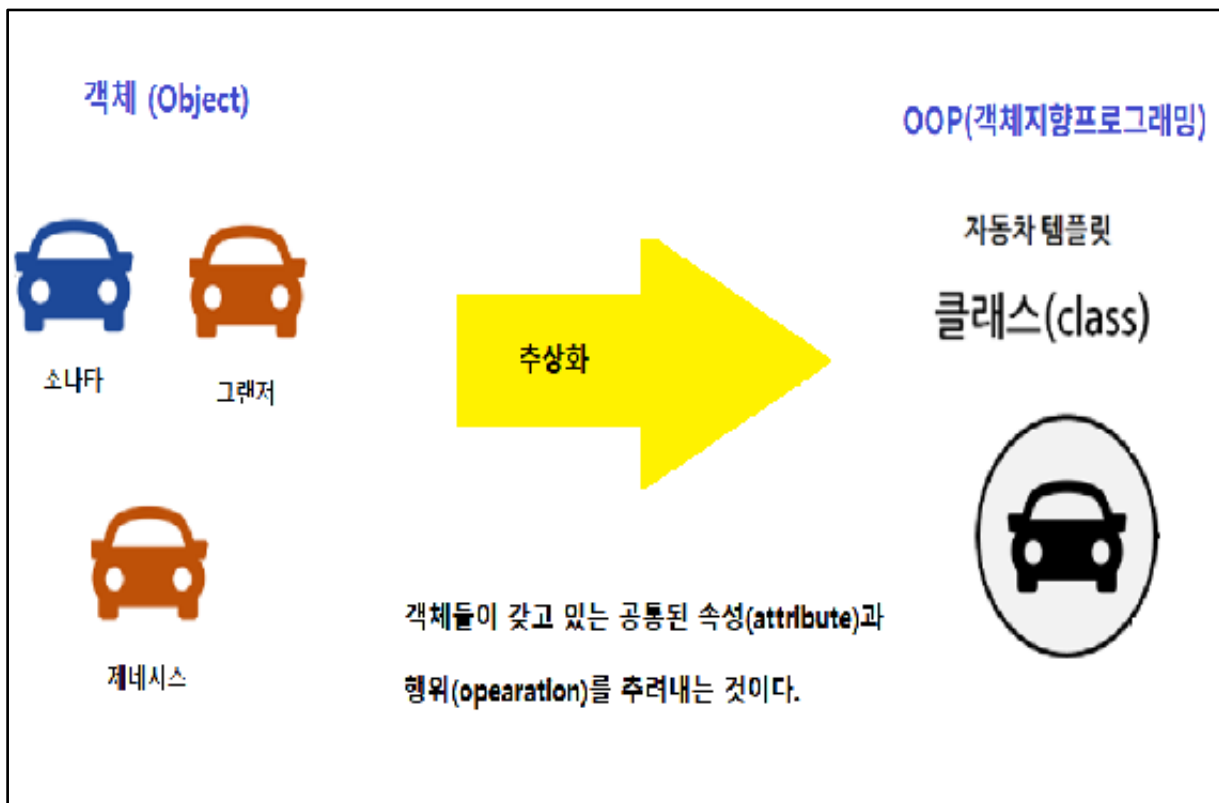
```
for ( 자료형 변수 : 배열이름 ) {  
    실행문;  
}
```

- 배열의 길이 만큼 실행문을 반복하는데 반복이 이루어질 때마나 배열 원소를 순서대로 가져와 변수에 자동으로 대입해준다.

객체 지향 프로그래밍 (OOP)

● 객체(Object)

- 모델링(설계) 관점에서 실세계에 존재하는 모든 유형 및 무형의 사물을 말한다.



● 클래스(class)

- 클래스는 실세계를 구성하고 있는 객체(object)를 구현하기 위해 정의하는 템플릿(template) 이다.
- 실세계의 객체를 클래스로 정의하고 클래스로 인스턴스를 생성하는 것이다.

● 예시

- 클래스명 : 자동차
- attribute : 이름, 색상, 방향, 속도
- operation : 우회전하다, 좌회전하다, 가속하다, 감속하다

G 자동차
<ul style="list-style-type: none"> 이름: 문자열 색상: 문자열 방향: 정수 속도: 정수
<ul style="list-style-type: none"> 우회전하다(): void 좌회전하다(): void 가속하다(): void 감속하다(): void

G Car
javaApp
<ul style="list-style-type: none"> name: String color: String direction: int speed: int
<ul style="list-style-type: none"> Car() Car(name: String, color: String, direction: int, speed: int) turnRight(): void turnLeft(): void accelerate(): void decelerate(): void

● 클래스 선언

접근제어자 **class** 클래스이름 {

// 구성 멤버

1. 필드

2. 생성자 메소드

3. 메소드

};

● 인스턴스 생성

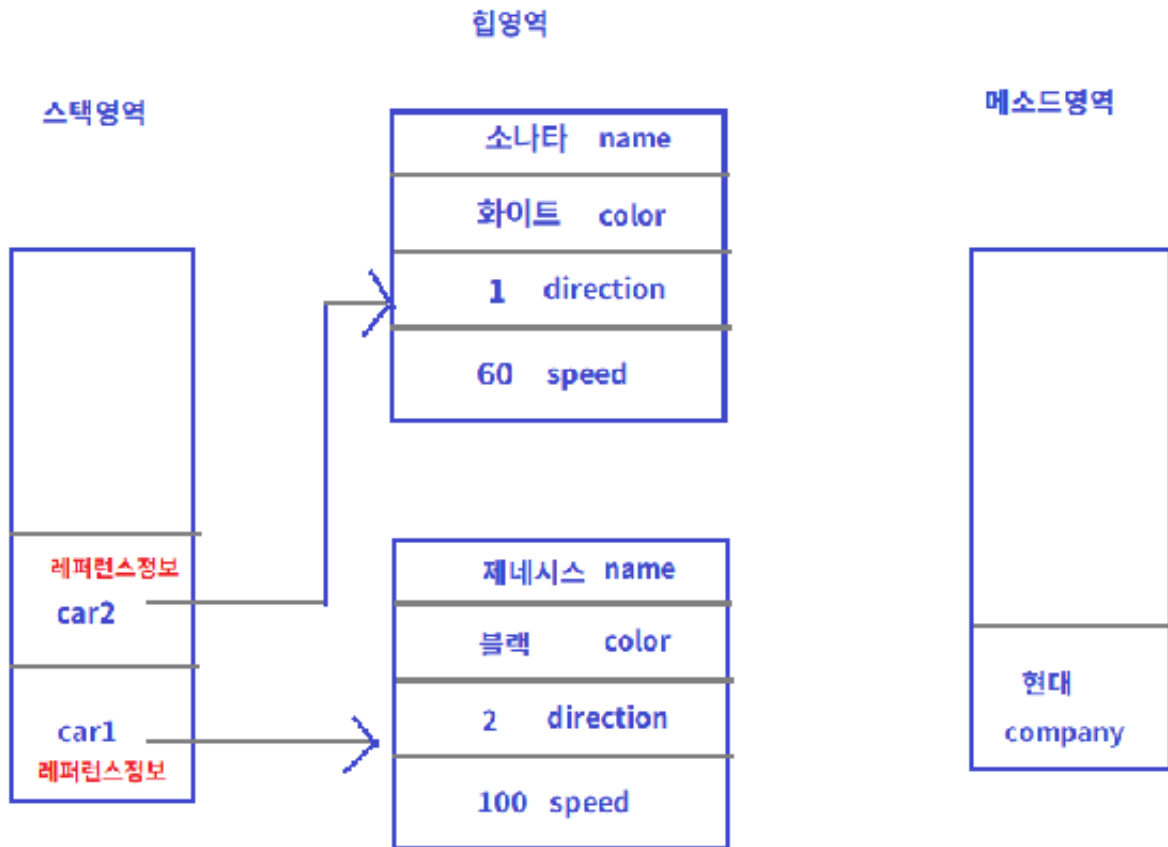
클래스명 변수 = new 클래스명();

※ 레퍼런스 변수라고 한다.

● 예시

```
public static void main(String[] args) {  
    Car car1 = new Car("소나타", "화이트", 1, 60);  
    Car car2 = new Car("제네시스", "블랙", 2, 100);  
}
```

● 메모리 구조



※ 인스턴스 변수를 참조하는 경우

car1.name

car1.color

※ 클래스(static) 변수를 참조하는 경우

Car.company

● 필드(field)

- 클래스에 선언된 변수(variable)를 필드라고 한다.
- 클래스 내에서 필드는 선언된 위치에 따라 다음과 같이 구분된다.
 1. **클래스 변수**(static variable) ; static 키워드를 갖는다.
 2. **인스턴스 변수**(instance variable)
 3. **지역 변수**(local variable)

변수	생성 시기	소멸 시기	저장 메모리	사용 방법
클래스 변수	클래스가 메모리에 올라갈 때	프로그램이 종료될 때	메소드 영역	클래스이름.변수이름
인스턴스 변수	인스턴스가 생성될 때	인스턴스가 소멸할 때	힙 영역	인스턴스이름.변수이름
지역 변수	블록 내에서 변수의 선언문이 실행될 때	블록을 벗어날 때	스택 영역	변수이름

- 클래스 변수는 인스턴스를 생성하지 않고도 바로 사용할 수 있다.
- 클래스 변수는 해당 클래스의 모든 인스턴스가 공유해야 하는 값을 유지하기 위해 사용된다.
- 인스턴스 변수는 인스턴스마다 고유한 값을 유지하기 위해 사용된다.

생성자 메소드 (Constructor)

- new 연산자를 통해 객체(인스턴스)가 생성될 때 인스턴스 변수를 초기화하기 위해 단 한번만 호출되는 메소드이다.
- 생성자 메소드 이름은 클래스 이름과 반드시 동일해야 한다.
- 생성자 메소드는 여러 개 작성 가능하다. (메소드 오버로딩이라고 한다.)
- 생성자는 리턴 타입을 지정할 수 없다.
- 개발자가 생성자를 작성하지 않았으면 컴파일러가 자동으로 기본 생성자를 삽입한다.

메소드 오버로딩(method overloading)

- 한 클래스 내에 같은 이름의 메소드를 중복하여 정의하는 것을 말한다.
- 매개변수의 개수나 타입 (메소드 시그니처)이 반드시 달라야 한다.

● 클래스 초기화 블록

```
static {  
    company = “현대”  
}
```

- 클래스 초기화 블록은 클래스가 처음으로 메모리에 로딩될 때 클래스 변수를 초기화하기 위해서 단 한 번만 실행된다.

메소드(Method)

- 중복되는 코드의 반복적인 프로그래밍을 피할 수 있고, 모듈화로 인해 코드의 가독성도 좋아진다.
- 메소드를 작성할 때는 되도록 하나의 메소드가 하나의 기능만을 수행하도록 작성하는 것이 좋다. (응집도가 높다.)

● 메소드 선언

```
접근제어자 [static] 리턴타입 메소드명 ( 자료형 변수명, ... ) {  
    실행코드 ;  
    return 값; // return  
}
```

1. 접근 제어자 : 해당 메소드에 접근할 수 있는 범위를 명시한다.
2. 리턴타입 : 메소드가 모든 작업을 마치고 반환(return)하는 자료형을 명시한다.

단, 리턴값이 없는 경우 void 를 지정한다.
3. 메소드 이름 : 메소드를 호출하기 위한 이름을 명시한다.

(첫글자는 소문자를 원칙으로한다.)
4. 매개변수 목록 (parameterlist) : 메소드 호출 시에 전달되는 인수의 값을 저장할 변수들을 명시한다.

● 메소드 구분

1. 클래스 메소드(static method)

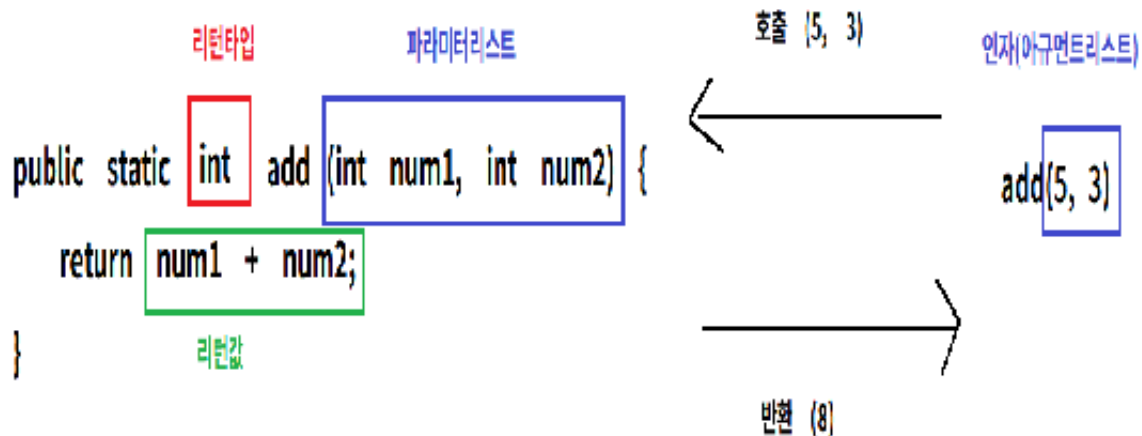
- static 키워드를 갖는다.
- 클래스 메소드는 클래스 변수와 마찬가지로 인스턴스를 생성하지 않고도 바로 사용할 수 있다. 따라서 클래스 메소드는 메소드 내부에서 인스턴스 변수를 사용할 수 없다.

```
Car.getCompany();    // 클래스명.메소드명 ( ... );
```

2. 인스턴스 메소드(instance method)

```
car1.turnLeft( );    // 레퍼런스변수.메소드명 ( ... );
```

● 메소드의 호출과 반환



● 메소드 호출 방식

- 프로그래밍 언어에서 변수를 다른 함수의 인자로 넘겨 줄 수 있다. 이 때 이 변수의 '값'을 넘겨 주는 호출 방식을 Call by Value, 이 변수의 '참조값' (혹은 주소, 포인터)를 넘겨 주는 호출 방식을 Call by Reference 라고 한다.

1. Call By Value

2. Call By Reference

● 예시 : Call by Value

```
public static void swap ( int num1, int num2 ) {  
    int temp = num1 ;  
    num1 = num2 ;  
    num2 = temp ;  
}
```

```
public static void main(String[] args) {  
    int num1 = 5, num2 = 10 ;  
    swap (num1, num2);  
    System.out.printf("num1 = %d, num2 = %d\n", num1, num2);  
}
```

● 메소드 호출과 메모리 (Call by Value)

```
public static void swap(int num1, int num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

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

    swap(num1, num2);

    System.out.printf("num1 = %d, num2 = %d\n", num1, num2);
}
```

스택영역

swap() 스택프레임

temp	5
num1	10
num2	5

main() 스택프레임

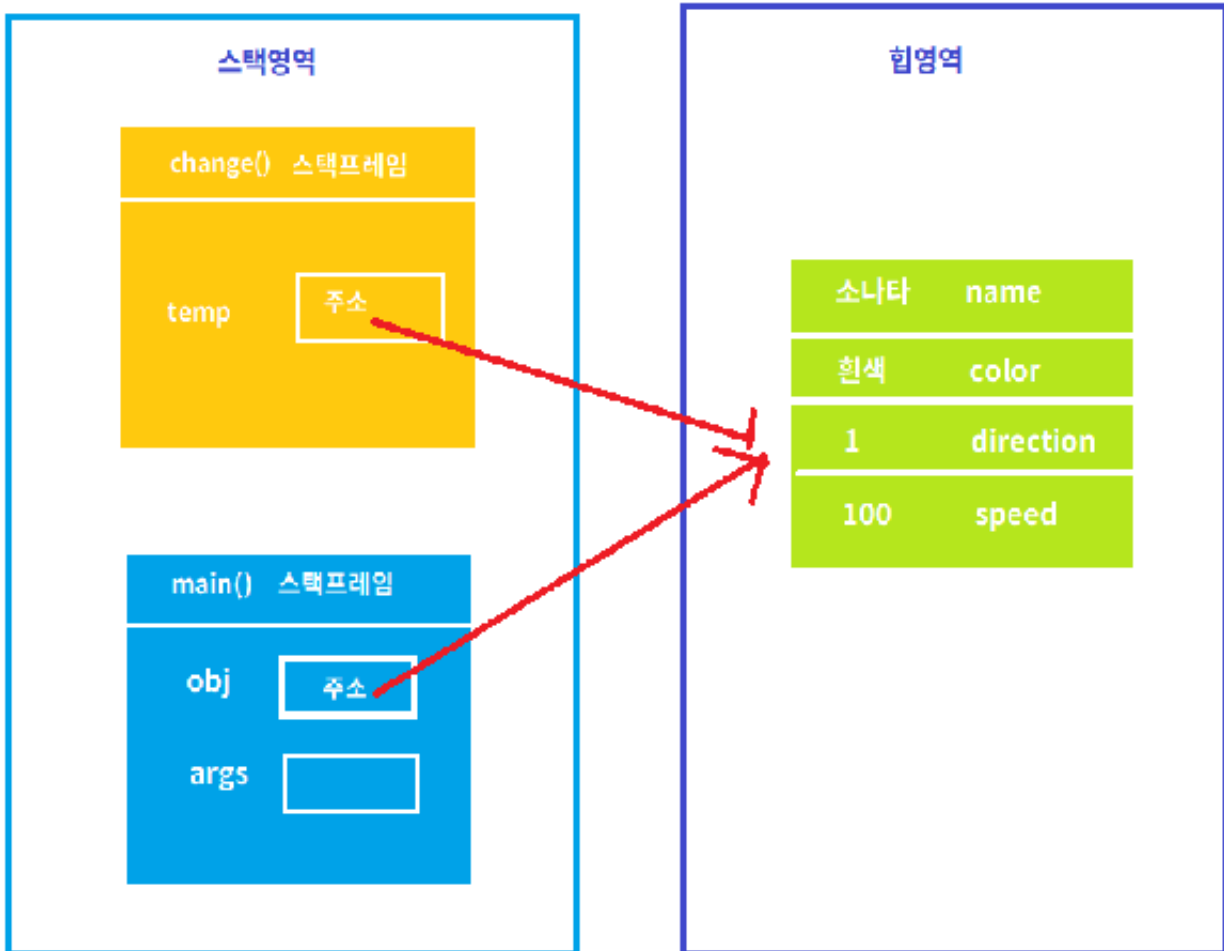
num2	10
num1	5
args	

※ 스택 : LIFO(후입선출)

● 예시 : Call by Reference

```
public class MethodExam {  
  
    public static void change(Car temp) {  
        // 자동차 색상을 변경하다.  
        temp.setColor("검정색");  
    }  
  
    public static void main(String[] args) {  
  
        Car obj = new Car("소나타", "흰색", 1, 100);  
  
        System.out.println(obj.getColor());  
  
        // 메소드 호출  
        change(obj);  
  
        System.out.println(obj.getColor());  
  
    }  
}
```


● 메소드 호출과 메모리 (Call by Reference)



this

- 인스턴스가 자기 자신을 참조하는 데 사용하는 변수이다.
- 모든 인스턴스 메소드에는 this 참조변수를 사용하여 인스턴스 변수에 접근할 수 있다.
- 생성자 메소드에서 매개변수 이름과 인스턴스 변수의 이름이 같을 경우에는 인스턴스 변수 앞에 this 키워드를 붙여 구분해야 한다.

접근 제어자 (Access Modifier)

- 캡슐화 (Encapsulation) : 데이터 보호
- 클래스나 멤버(필드, 메소드)를 선언할 때 접근 제어자를 사용하여 접근 범위를 지정한다.
- 접근제어자는 생략가능하며 생략했을 때는 자동으로 default 접근제어자가 설정된다.
- public > protected > default > private

접근 제어자	같은 클래스의 멤버	같은 패키지의 멤버	자식 클래스의 멤버	그 외의 영역
public	○	○	○	○
protected	○	○	○	×
default	○	○	×	×
private	○	×	×	×

상속 (inheritance)

1. 객체 지향 프로그래밍의 중요한 특징 중 하나이다.
2. 코드의 재사용이 가능하다.
3. 클래스 간의 계층적 관계를 구성할 수 있다. (다형성)

[syntax]

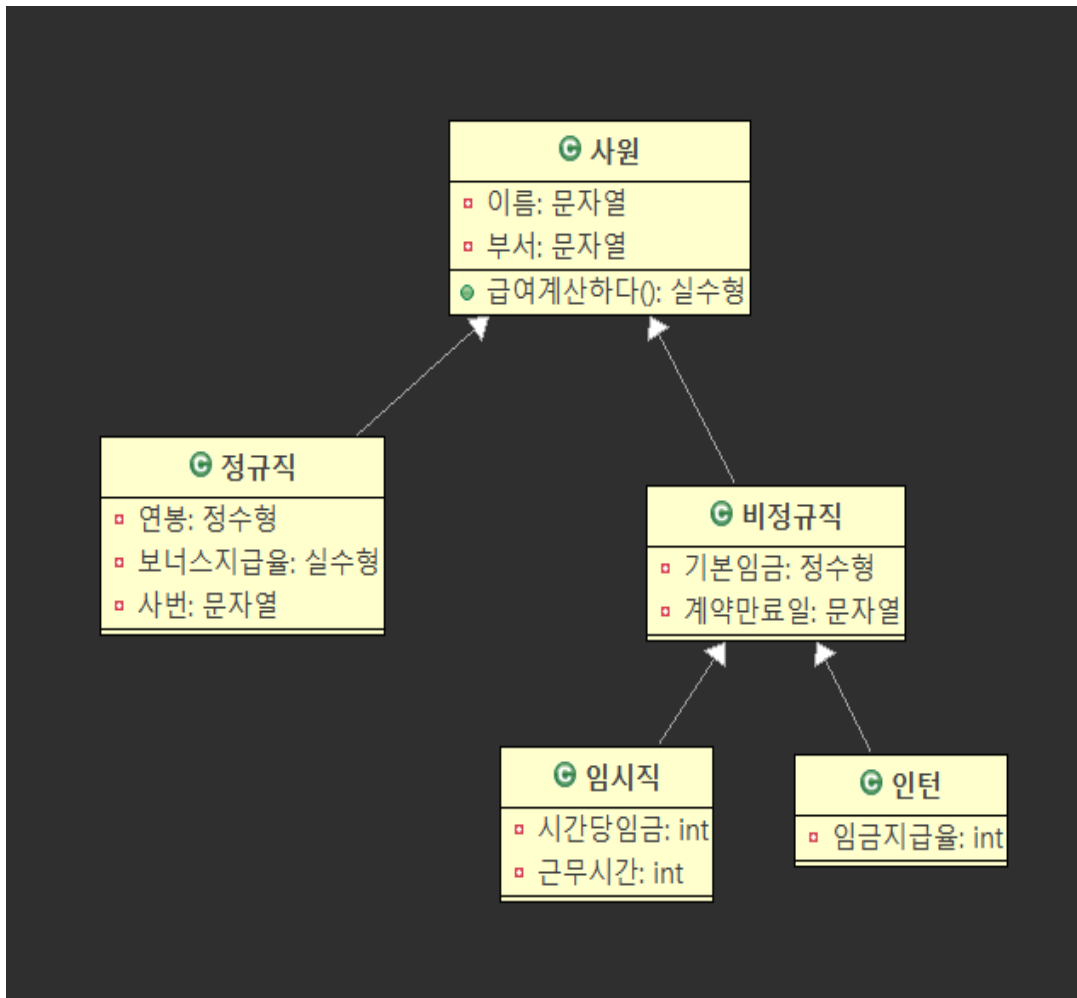
```
class 하위클래스명 extends 상위클래스명 {  
  
}  

```

● 예시

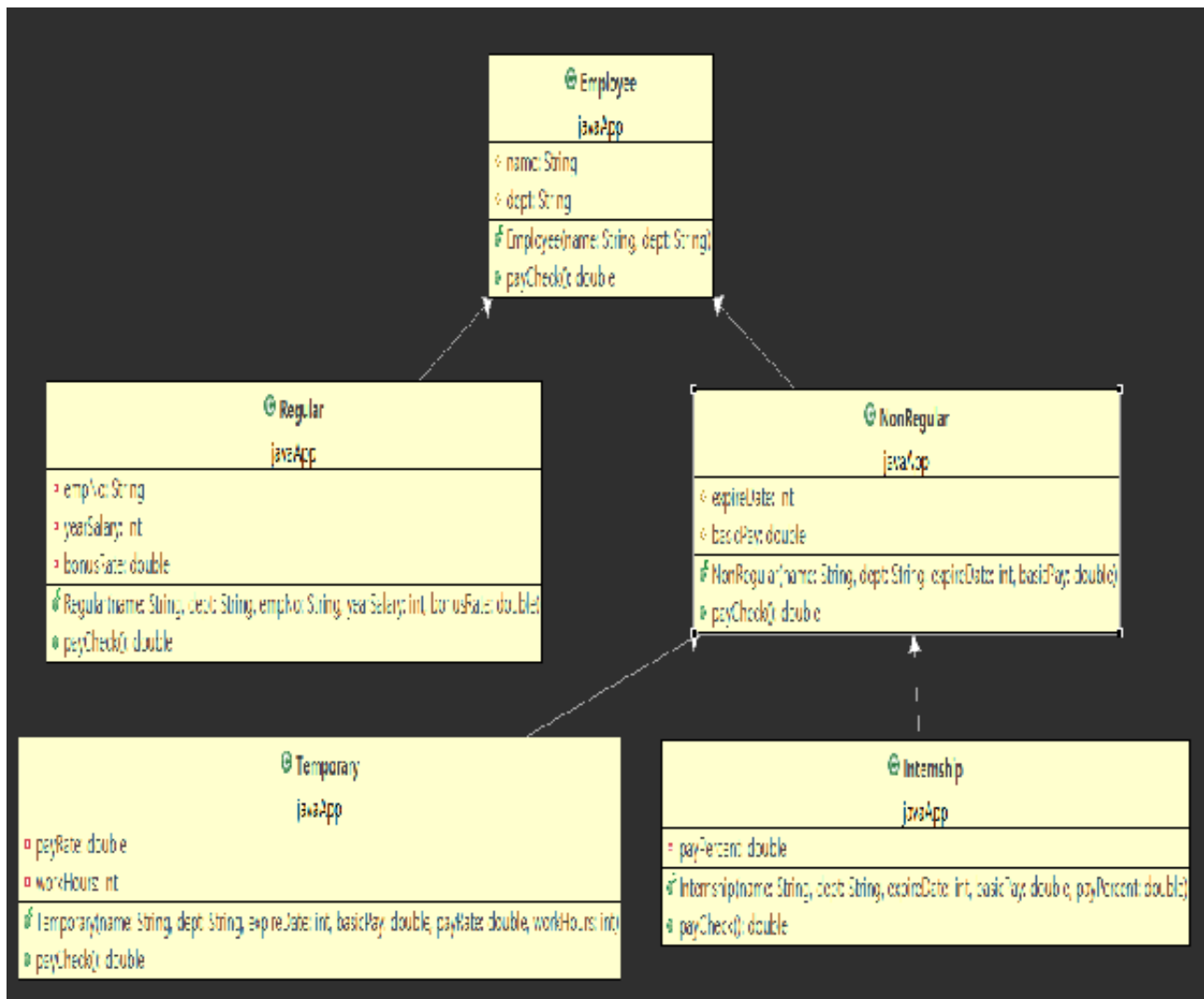
- 인사급여 관리시스템을 개발한다고 가정하자
- 직원에는 정규직, 비정규직이 있고, 비정규직 사원에는 임시 직원과 인턴 사원이 있다.
- 이름, 소속 부서는 직원 모두 공통사항을 입력받는다.
- 정규직은 사원번호, 연봉, 보너스 지급율을 추가로 입력받는다.
- 비정규직은 계약 만료일, 기본임금을 추가로 입력받는다.
- 임시직원은 시간당임금, 근무시간을 추가로 입력받는다.
- 인턴사원은 임금지급율을 입력받는다.
- 정규직원의 급여는 연봉(yearSalary) + 보너스 지급율(bonusRate)
- 비정규직원의 급여는 기본 임금(basicPay)
- 임시직원의 급여는 기본 임금(basicPay) + 시간당 임금(payRate) * 근무시간(workHours)
- 인턴사원의 급여는 기본임금 + 임금지급율(payPercent)

● UML 클래스 다이어그램



※ 일반화 관계 (Generalization)

- 한 클래스가 다른 클래스를 포함하는 상위 개념일 때 두 클래스 사이에는 일반화 관계가 존재한다.
- IS-A 관계 이다.



메소드 오버라이딩(method overriding)

· 상속받은 상위 클래스의 메소드를 하위클래스에서 재정의하여 사용하는 것을 의미한다.

● 오버라이딩의 조건

1. 메소드의 선언부는 기존 메소드와 완전히 같아야 한다.

하지만 메소드의 반환 타입은 부모 클래스의 반환 타입으로 타입 변환할 수 있는 타입이라면 변경할 수 있다.

2. 부모 클래스의 메소드보다 더 큰 범위의 예외를 선언할 수 없다.

super

- 상위 클래스로부터 상속받은 필드나 메소드를 하위 클래스에서 참조하는 데 사용하는 참조 변수이다.

- super.dept

- super.payCheck()

● super() 메소드

- 상위 클래스의 생성자 메소드를 호출할 때 사용한다.

- 하위 클래스의 인스턴스를 생성하면, 해당 인스턴스에는 하위 클래스의 고유 인스턴스 변수뿐만 아니라 상위 클래스의 모든 인스턴스 변수까지도 포함된다. 따라서 상위 클래스의 인스턴스 변수를 초기화하기 위해서는 하위 클래스의 생성자에서 상위 클래스의 생성자까지 호출해야 한다.

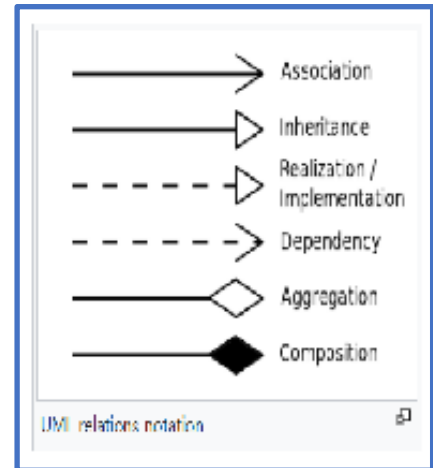
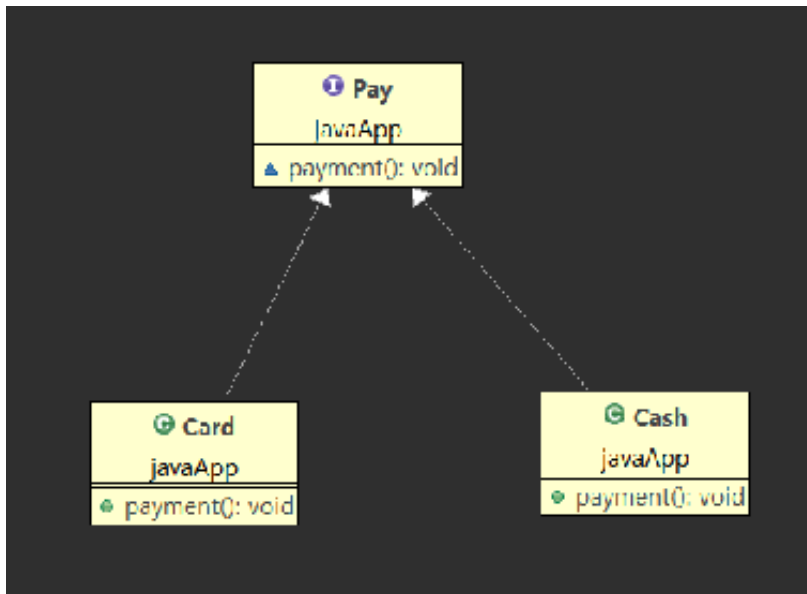
인터페이스 (interface)

- 기능 명세서이다.
- 모든 기능을 추상화로 정의만 하고 구현은 하지 않은것이다.

● 인터페이스 (interface)를 사용하는 이유

1. 협업을 할때 미리 인터페이스를 작성함으로써 메소드를 정할 수 있다.
2. 클래스간 결합도(코드 종속성)를 낮춘 유연한 방식의 프로그래밍이 가능해진다.
3. 자바에서는 다중 상속을 구현하기 위해서 사용한다.

● UML 클래스 다이어그램



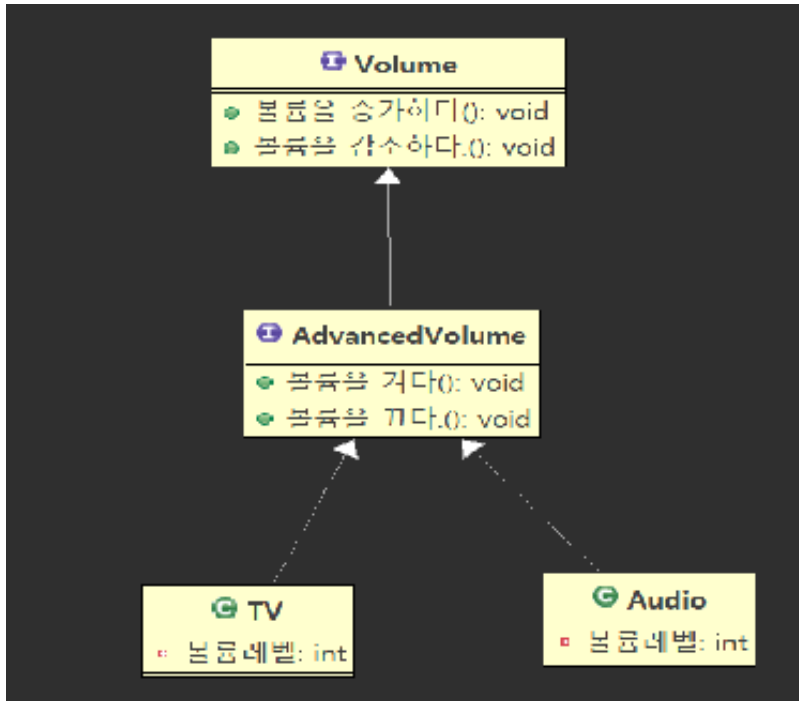
※ Realization 관계

- 인터페이스의 Spec(명세, 정의) 만 있는 메소드를 오버라이딩하여 실제 기능으로 구현하는 것을 말한다.

● 문법

```
접근제어자 interface 인터페이스이름 {  
    //상수  
  
    // abstract 메소드  
  
    // default 메소드 (Java8 추가)  
  
    // static 메소드 (Java8 추가)  
  
}
```

● 인터페이스 간의 상속



접근제어자 **interface** 하위인터페이스이름 **extends** 상위인터페이스이름 {

 //상수

 // abstract 메소드, default 메소드, static 메소드

}

다형성(polymorphism)

- 하나의 객체가 여러 가지 타입을 가질 수 있는 것을 의미한다.
- 자바에서는 다형성을 상위 클래스 타입의 참조 변수로 하위 클래스 타입의 인스턴스를 참조할 수 있도록 하여 구현하고 있다.
- 다형성은 상속, 추상화와 더불어 객체 지향 프로그래밍을 구성하는 중요한 특징 중 하나이다.

● 참조 변수의 다형성

```
Employee emp = null;  
emp = new Regular();           // 업캐스팅  
emp = new Temporary();         // 업캐스팅  
emp = new Internship();        // 업캐스팅
```

- Employee 클래스에 정의된 메소드만 호출할 수 있다.

```
Employee emp = new Regular();  
Regular emp1 = (Regular)emp; // 다운캐스팅
```

- emp1 참조 변수를 사용하여 Regular 클래스에 정의된 메소드를 호출할 수 있다.

※ instanceof 연산자

- instanceof 연산자를 사용하면 다형성으로 인해 런타임에 참조 변수가 실제로 참조하고 있는 인스턴스의 타입을 확인할 수 있다.

object instanceof type

- object가 type이거나 type을 상속받는 클래스라면 true를 리턴하고. 그렇지 않으면 false를 리턴한다.

자바 API

java.lang.Object 클래스

- 모든 자바 클래스의 최상위 클래스이다.

1. toString() 메소드

- 해당 인스턴스에 대한 정보를 문자열로 반환한다.
- 클래스 이름@ 16진수 해시 코드(hash code)
- 해시코드 : 인스턴스의 주소를 가리키는 값이다.

2. equals() 메소드

- 해당 객체와 전달받은 객체가 같은지 여부를 반환한다.

3. hashCode() 메소드

- 해당 객체의 해시 코드값을 반환한다.

java.lang.String 클래스

- 자바에서는 문자열을 위한 String이라는 클래스를 별도로 제공한다.
- 변하지 않는 문자열을 자주 사용할 경우

1. charAt() 메소드

- charAt() 메소드는 해당 문자열의 특정 인덱스에 해당하는 문자를 반환한다.

2. String[] split(String regex)

- 해당 문자열을 전달된 정규 표현식(regular expression)에 따라 분리하여 나눠서 반환한다.

3. String substring(int begin, int end)

- 해당 문자열의 전달된 시작 인덱스부터 마지막 인덱스 - 1까지를 새로운 문자열로 반환한다.

4. length()

- 해당 문자열의 길이를 반환한다.

StringBuffer vs StringBuilder 클래스

- 문자열의 추가, 수정, 삭제 등이 빈번히 발생하는 경우 사용한다.
- String 클래스의 인스턴스는 한 번 생성되면 그 값을 읽기만 할 수 있고, 변경할 수는 없는 불변 클래스이다.
- StringBuffer/StringBuilder 클래스의 인스턴스는 그 값을 변경할 수도 있고, 추가할 수도 있다. 이를 위해 StringBuffer 클래스는 내부적으로 버퍼(buffer)를 사용한다.
- 덧셈(+) 연산자를 이용해 String 인스턴스의 문자열을 결합하면 내용이 합쳐진 새로운 String 인스턴스를 생성한다. 따라서 문자열을 많이 결합하면 결합할수록 공간의 낭비뿐만 아니라 속도 또한 매우 느려진다.
- StringBuffer/StringBuilder 인스턴스를 사용하면 문자열을 바로 추가할 수 있으므로 공간의 낭비도 없으며 속도도 매우 빠르다.
- StringBuilder는 동기화를 지원하지 않는 반면, StringBuffer 는 동기화를 지원하여 멀티 스레드 환경에서도 안전하게 동작한다.

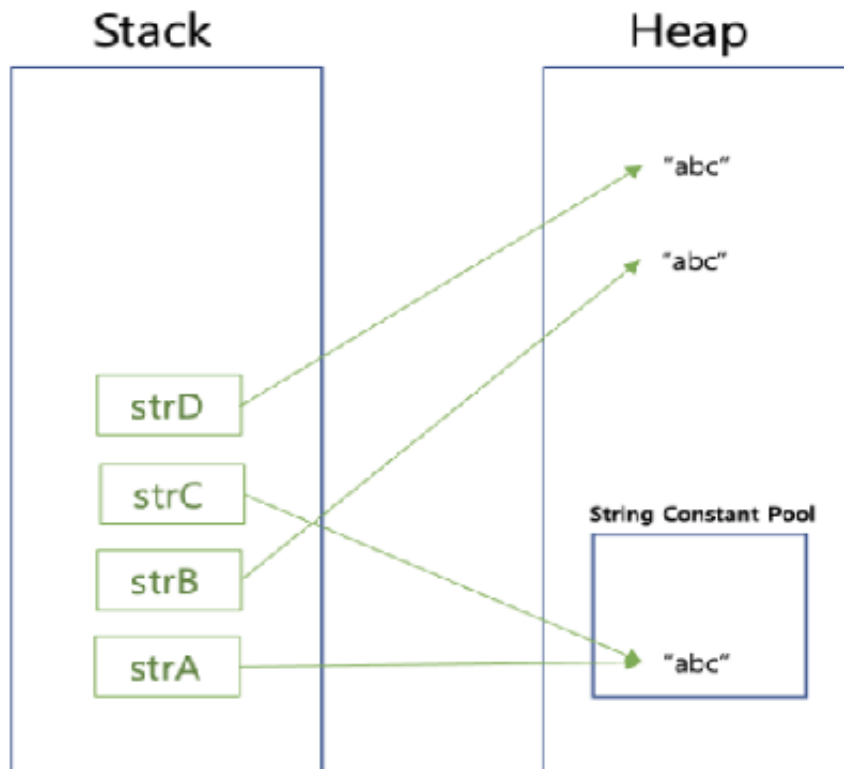
1. append() 메소드

- 인수로 전달된 값을 문자열로 변환한 후 해당 문자열의 마지막에 추가한다.

2. delete() 메소드

- 전달된 인덱스에 해당하는 부분 문자열을 해당 문자열에서 제거한다.

● String vs StringBuilder 메모리 구조



```
String strA = "abc" ;
String strB = new String("abc") ;
String strC = "abc" ;
String strD = new String("abc") ;
```

java.lang.Math 클래스

1. random() 메소드

- 0.0 이상 1.0 미만의 범위에서 임의의 double 형 값을 하나 생성하여 반환한다.

java.time.LocalDate 클래스

- JDK 8 버전 부터 제공
- 날짜를 표현하는 클래스이다.

`LocalDate.now();`

- 시스템에 default로 지정된 시간과 타임존을 이용하여 현재 날짜를 가져온다

※ 날짜 포맷

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

```
now.format(formatter);
```

```
int year = now.getYear();
```

```
int monthValue = now.getMonthValue();
```

```
String dayOfWeek = now.getDayOfWeek().toString();
```

```
int dayOfWeekValue = now.getDayOfWeek().getValue();
```

```
// 텍스트 요일 구하기 (한글)
```

```
DayOfWeek dayOfWeek = now.getDayOfWeek();
```

```
System.out.println(dayOfWeek.getDisplayName(TextStyle.FULL, Locale.KOREAN));
```

```
System.out.println(dayOfWeek.getDisplayName(TextStyle.NARROW, Locale.KOREAN));
```

java.time.LocalDateTime 클래스

- 시간을 표현하는 클래스이다.

```
LocalTime.now();
```

- LocalDateTime 클래스를 이용하여 현재 시간을 구할 수 있습니다.

※ 시간 포맷

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH 시 mm 분 ss 초");
```

```
String formattedNow = now.format(formatter);
```

```
int hour = now.getHour();
```

```
int minute = now.getMinute();
```

```
int second = now.getSecond();
```

예외(Exception)

● 예외(Exception) 란?

- 프로그램을 실행하는 과정에서 발생하는 예상치 못한 에러를 말한다.
- 네트워크 연결에 실패한 경우
- 파일이 존재하지 않는 경우
- 데이터베이스 연결에 실패한 경우

● 예외 처리 (Exception Handling)

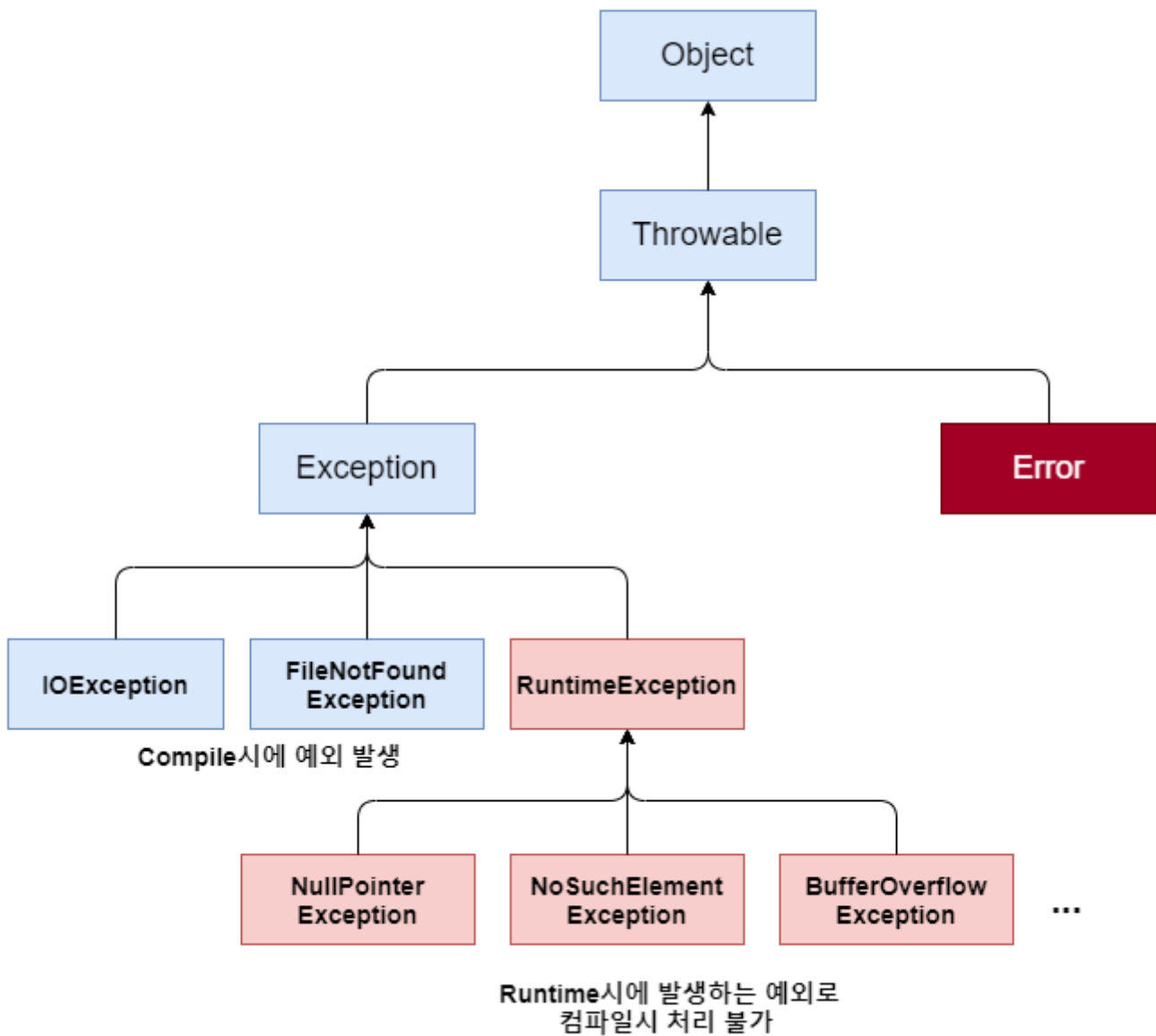
1. try ~ catch 구문

```
try {  
    예외가 발생할만한 코드  
} catch (Exception클래스명 e1) {  
    e1 예외가 발생할 경우에 실행될 코드;  
} catch (Exception클래스명 e2) {  
    e2 예외가 발생할 경우에 실행될 코드;  
}  
...  
finally {  
    예외 발생 여부와 상관없이 무조건 실행될 코드;  
}
```

2. throws 구문

```
접근지정자 [static] 리턴타입 메소드명 ( ) throws Exception 클래스이름 {  
    throw new Exception 클래스이름();  
}
```

● Exception 클래스 계층도



● Throwable 클래스

- 모든 예외의 최상위 클래스이다.

메소드	설 명
String getMessage()	해당 Throwable객체에 대한 자세한 내용을 문자열로 반환
void printStackTrace()	해당 Throwable 객체와 표준 오류 스트림에서 해당 객체의 스택 트레이스를 출력함.
String toString()	해당 Throwable객체에 대한 간략한 내용을 문자열로 반환

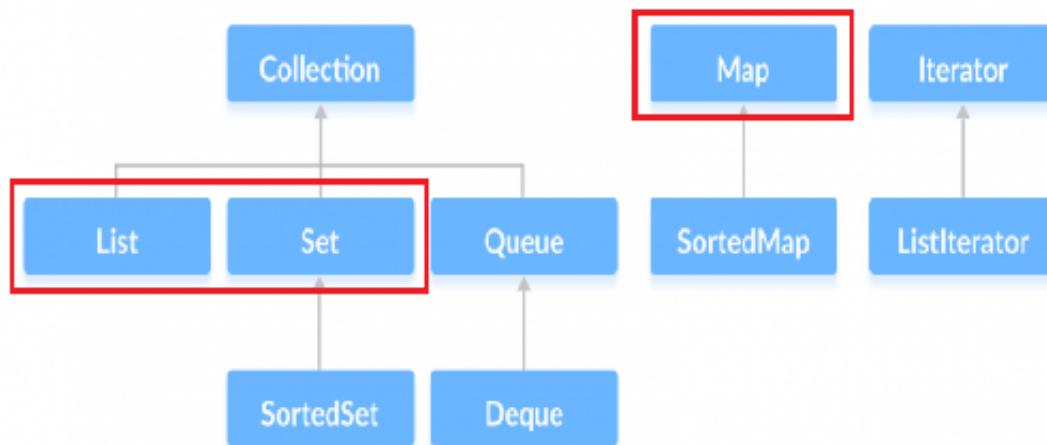
※ 스택 트레이스란?

- 프로그램이 시작된 시점부터 현재 위치 까지의 메서드 호출 목록 입니다. 이는 예외가 어디서 발생했는지 알려주기 위해 JVM이 자동으로 생성한다.

자바 컬렉션 프레임워크

- 다수의 데이터를 쉽고 효과적으로 처리할 수 있는 표준화된 방법을 제공한다.
- 데이터를 저장하는 자료 구조와 데이터를 처리하는 알고리즘을 구조화하여 클래스로 구현해 놓은 것이다.

Java Collections Framework



<https://www.programiz.com/java-programming/collections>

● Collection 인터페이스에서 제공하는 메소드

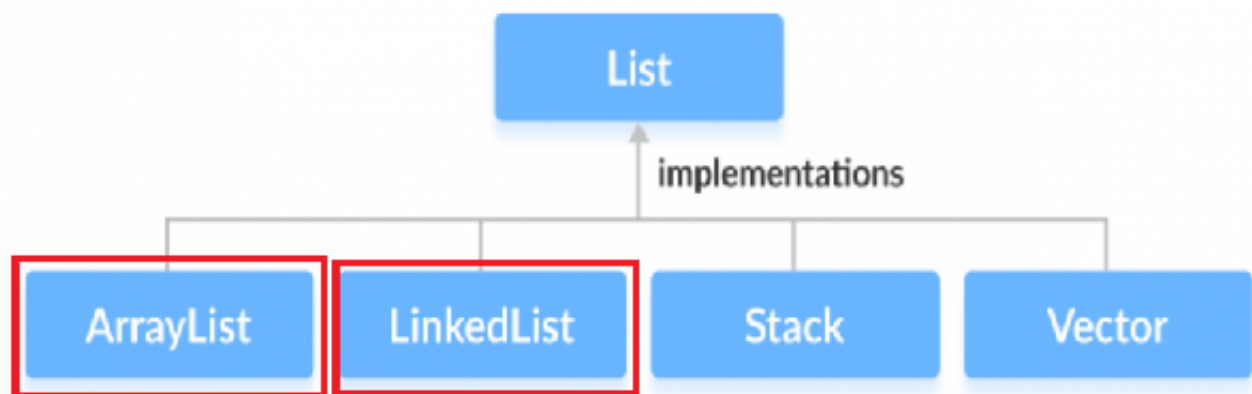
메소드	설명
<code>boolean add(E e)</code>	허당 컬렉션(collection)에 전달된 요소를 추가함. (선택적 기능)
<code>void clear()</code>	허당 컬렉션의 모든 요소를 제거함. (선택적 기능)
<code>boolean contains(Object o)</code>	허당 컬렉션이 전달된 객체를 포함하고 있는지를 확인함.
<code>boolean equals(Object o)</code>	허당 컬렉션과 전달된 객체가 같은지를 확인함.
<code>boolean isEmpty()</code>	허당 컬렉션이 비어있는지를 확인함.
<code>Iterator<E> iterator()</code>	허당 컬렉션의 반복자(iterator)를 반환함.
<code>boolean remove(Object o)</code>	허당 컬렉션에서 전달된 객체를 제거함. (선택적 기능)
<code>int size()</code>	허당 컬렉션의 요소의 총 개수를 반환함.
<code>Object[] toArray()</code>	허당 컬렉션의 모든 요소를 Object 타입의 배열로 반환함.

● 주요 인터페이스

인터페이스	설명	구현 클래스
List<E>	순서가 있는 데이터의 집합으로, 데이터의 중복을 허용함.	Vector, ArrayList, LinkedList, Stack, Queue
Set<E>	순서가 없는 데이터의 집합으로, 데이터의 중복을 허용하지 않음.	HashSet, TreeSet
Map<K, V>	키와 값의 한 쌍으로 이루어지는 데이터의 집합으로, 순서가 없음. 이때 키는 중복을 허용하지 않지만, 값은 중복될 수 있음.	HashMap, TreeMap, Hashtable, Properties

List<E> 인터페이스

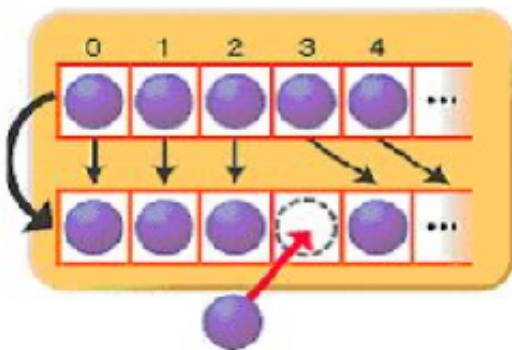
- 순서가 있는 데이터의 집합으로 데이터의 중복을 허용한다.



List인터페이스의 구현 클래스들

● ArrayList<E> 클래스

- ArrayList 클래스는 배열을 이용하기 때문에 인덱스를 이용해 배열 요소에 빠르게 접근할 수 있다. (임의의 접근이 빠르다)
- 기존 배열과 차이점은 배열의 크기가 가변적이다.
- 요소의 추가 및 삭제 시 성능이 느리다는 단점을 갖고 있다.

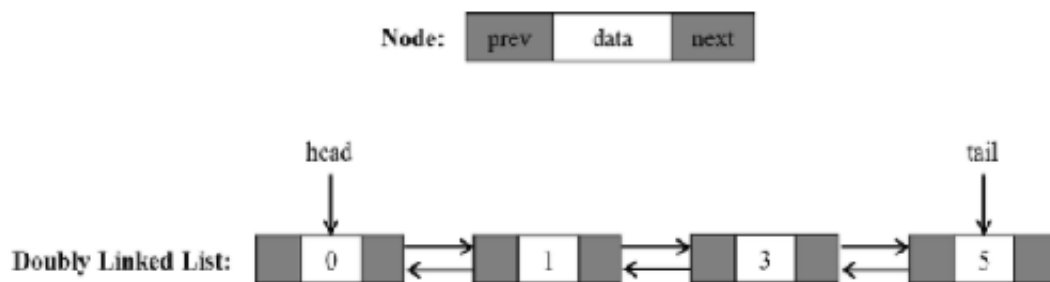


[문법]

```
ArrayList<클래스타입> list = new ArrayList<클래스타입> ();
```

● LinkedList<E> 클래스

- 중간에 데이터를 추가나 삭제하더라도 전체의 인덱스가 한 칸씩 뒤로 밀리거나 당겨지는 일이 없기에 ArrayList에 비해서 데이터의 추가나 삭제가 용이하다.
- 인덱스가 없기에 특정 요소에 접근하기 위해서는 순차 탐색이 필요로 하여 탐색 속도가 떨어진다는 단점이 있다.

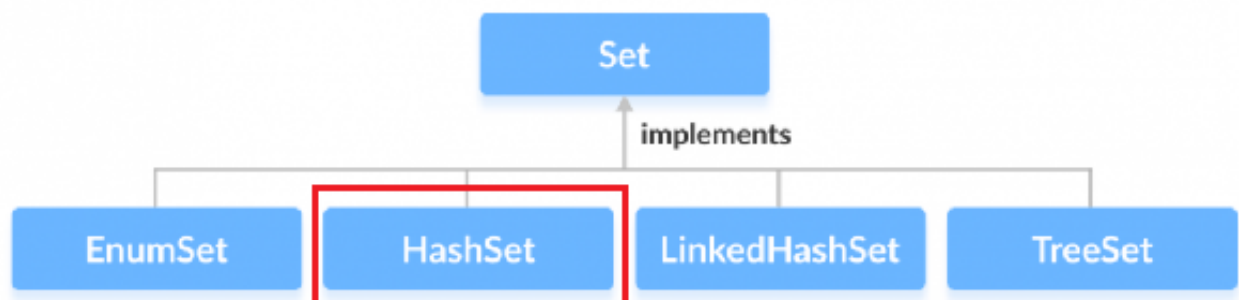


[문법]

```
LinkedList<클래스타입> lnkList = new LinkedList<클래스타입> ( ) ;
```

Set<E> 인터페이스

- 순서가 없는 데이터의 집합으로 데이터의 중복을 허용하지 않는다.



● HashSet<E> 클래스

- Set은 객체를 중복해서 저장할 수 없고, 저장 순서가 유지되지 않는다.
- Set은 비선형 구조이므로 순서가 없으며 인덱스도 존재하지 않는다.

※ 중복을 걸러내는 과정

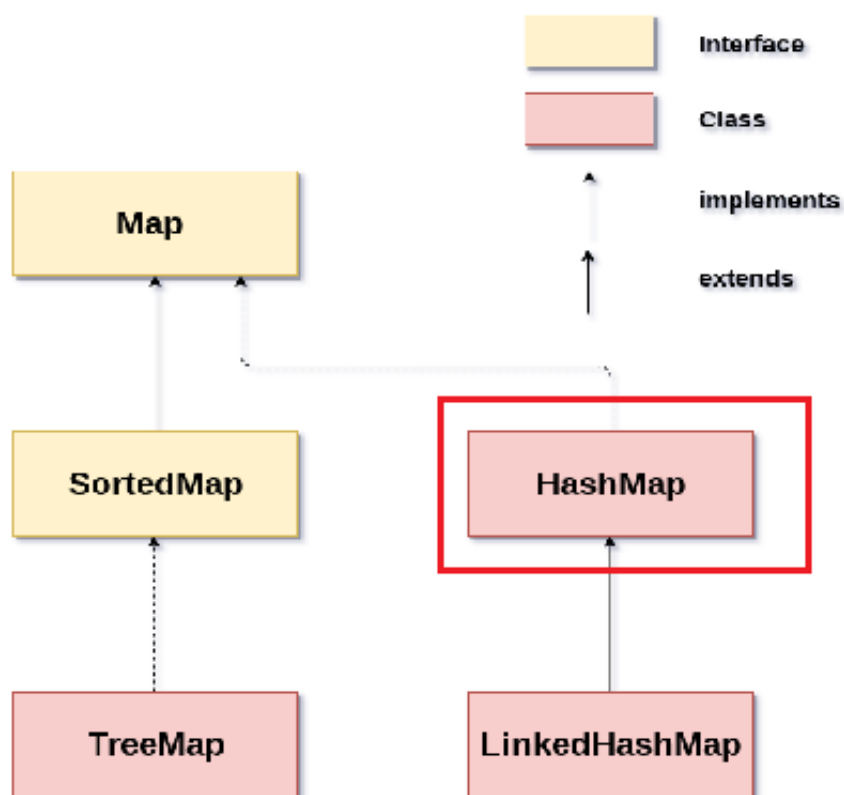
- HashSet은 객체를 저장하기 전에 먼저 객체의 `hashCode()` 메소드를 호출해서 해시 코드를 구한 후 HashSet에 저장되어 있는 객체들의 해시 코드와 비교한 뒤 같은 해시 코드가 있다면 다시 `equals()` 메소드로 두 객체를 비교해서 true가 나오면 동일한 객체로 판단하고 중복 저장을 하지 않는다.

[문법]

```
HashSet<클래스타입> set1 = newHashSet<클래스타입>();
```

Map<K, V> 인터페이스

- 키와 값의 한쌍으로 이루어지는 한 쌍의 집합으로 순서가 없음.
- 키는 중복을 허용하지 않으나 값은 중복될 수 있음.



● HashMap<K, V> 클래스

· Map은 키와 값으로 구성된 Entry객체를 저장하는 구조를 가지고 있는 자료구조이다.

· 값은 중복 저장될 수 있지만 키는 중복 저장될 수 없다. 만약 기존에 저장된 키와 동일한

키로 값을 저장하면 기존의 값은 없어지고 새로운 값으로 대체된다.

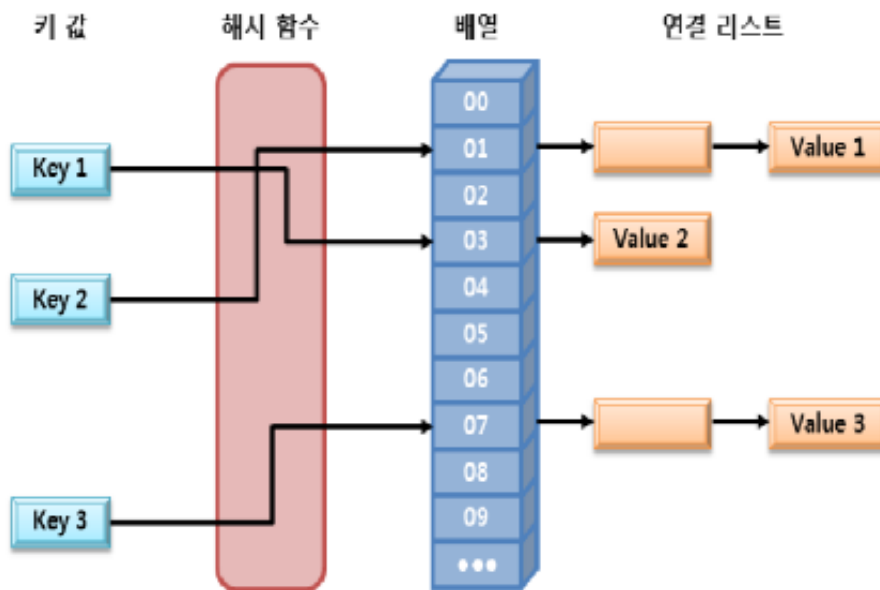
· HashMap 클래스는 해시 알고리즘(hash algorithm)을 사용하여 검색 속도가 매우 빠르다.

[문법]

```
HashMap<클래스이름, 클래스이름> map =  
    new HashMap< 클래스이름 , 클래스이름> ( );
```

※ 해시 알고리즘 (hash algorithm)

- 해시 알고리즘(hash algorithm)이란 해시 함수(hash function)를 사용하여 데이터를 해시 테이블(hash table)에 저장하고, 다시 그것을 검색하는 알고리즘이다.



- 자바에서 해시 알고리즘을 이용한 자료 구조는 위의 그림과 같이 배열과 연결 리스트로 구현된다. 저장할 데이터의 키값을 해시 함수에 넣어 반환되는 값으로 배열의 인덱스를 구한다. 그리고서 해당 인덱스에 저장된 연결 리스트에 데이터를 저장하게 된다.

Generic (제네릭)

● Generic(제네릭)이란?

- 데이터 타입(data type)을 일반화하는(generalize) 것을 의미한다.
- 제네릭은 클래스나 메소드를 선언할 때 데이터 타입을 정하는 것이 아니라, 인스턴스를 생성할 때나 메소드를 호출할 때 데이터 타입을 정한다.

● Generic(제네릭)의 장점

- 제네릭을 사용하면 잘못된 데이터 타입이 들어올 수 있는 것을 컴파일 단계에서 방지할 수 있다.
- 클래스 외부에서 타입을 지정해주기 때문에 따로 데이터 타입을 체크하고 변환해줄 필요가 없다.

● 제네릭 클래스

```
public class Box<T> { //Type Parameter

    private T item;

    public Box(T item) {
        super();
        this.item = item;
    }

    public T getItem() {
        return item;
    }

    public void setItem(T item) {
        this.item = item;
    }

}
```

● 제네릭 타입 제한

- 제네릭을 사용할 때 extends, super 키워드를 사용하여 데이터 타입을 제한할 수 있다.

```
class AnimalList <T extends LandAnimal> {  
  
    ...  
  
}
```

- 클래스와 인터페이스를 동시에 구현해야 한다면

```
class AnimalList <T extends LandAnimal & WarmBlood> {  
  
    ...  
  
}
```

< ? > // 타입 변수에 모든 타입을 사용할 수 있다. <? extends Object>

< ? extends T > // T 타입과 T 타입을 상속받는 하위 클래스 타입만을 사용할 수 있다.

< ? super T > // T 타입과 T 타입이 상속받은 상위 클래스 타입만을 사용할 수 있다.

● 제네릭 메소드

```
public static <T extends Comparable<? super T>>  
    void sort (List<T> list)
```

- T는 Comparable 인터페이스를 구현한 타입이어야 한다.