

자바

김용만

1. 자바의 특징 및 자바 플랫폼의 이해

1-1 자바의 탄생

- (1) 1991년경 제임스 고슬링과 아서 밴 호프와 같은 Sun Microsystems의 연구진들이 가전제품에 탑재될 소프트웨어를 만들 목적으로 '오크(Oak)'라는 언어 개발
- (2) 인터넷에 적합하도록 Oak의 개발 방향을 바꾸면서 이름을 자바(Java)로 변경
- (3) 1996년1월 자바의 정식 버전 발표

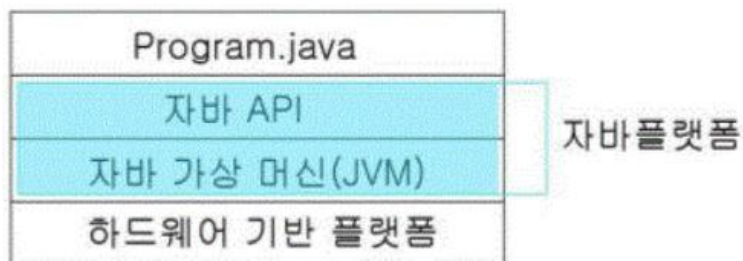
1-2. 자바의 특징

- (1) 개발하기 쉬운 객체지향 언어
- (2) 플랫폼 독립적
- (3) 멀티 스레드 지원
- (4) 네트워크와 분산처리 지원
- (5) 가비지 컬렉션

1-3. 자바 플랫폼

자바 프로그램이 실행되기 위해 제공되는 '하드웨어적 프로그램'. 서로 다른 하드웨어나 운영체제에서 동일한 프로그램이 거의 유사하게 동작할 수 있도록 해주는 환경을 의미한다. 다양한 운영체제에서 동작할 수 있도록 자바가상머신(JVM)이 제공된다.

(1) 플랫폼의 구성



(2) JVM(Java Virtual Machine)

- 자바 바이트코드를 실행하는 주체
- 운영체제와 자바프로그램을 연결시켜주는 역할
- 플랫폼에 독립적으로 동작
- JRE에 포함되어 배포됨
- 가비지 컬렉션을 수행
- 인터프리터나 JIT 컴파일 방식으로 바이트코드를 실행할 수 있도록 함(과거 순수 인터프리터 방식으로 실행시간이 늦은 편이었으나 JIT(Just-In-Time) 컴파일을 구현하여 성능을 개선)

컴파일러

고급 언어로 쓰여진 프로그램이 컴퓨터가 이해할 수 있는 저급 언어로 번역 번역과정이 번거롭고 번역 시간이 오래 걸림 한번 번역한 후에는 다시 번역하지 않으므로 실행 속도가 빠름.

인터프리터

프로그램을 한 단계씩 기계어로 해석하여 실행. 실행 시간이 길어 속도가 늦음. 프로그램이 직접 실행되므로 목적 프로그램이 생성되지 않음.

JIT 컴파일

프로그램을 실제 실행하는 시점에 기계어로 번역하는 컴파일 기법. 실행 시점에 기계어 코드를 생성하면서 해당 코드를 캐싱 함수가 여러 번 호출될 때마다 매번 기계어 코드가 생성되는 걸 방지 인터프리터의 실행 속도가 느린 단점을 보완

(3) 자바 플랫폼의 종류

- 1) Java SE (Java Standard Edition) : 데스크탑 애플리케이션 개발환경
- 2) Java EE (Java Enterprise Edition) : 엔터프라이즈 기반 애플리케이션 개발환경
- 3) Java ME (Java Micro Edition) : 모바일 및 임베디드 기반 애플리케이션 개발환경

1-4 JDK 다운로드 및 설치, 환경설정

(1) JDK(Java Development Kit) 설치

JRE(Java Runtime Environment)

자바 프로그램을 실행하기 위한 환경 라이브러리, JVM(자바 가상 기계), 기타 컴포넌트들을 제공. 자바 프로그램을 단순히 실행만 하고 개발을 하지 않는 일반 사용자용

JDK(Java Development Kit)

자바 프로그램을 개발하기 위한 컴파일러, 디버거와 같은 명령어 행 개발도구를 추가한 것 JRE를 포함

오라클 사이트 접속 <https://www.oracle.com/java/technologies/>



JDK DOWNLOAD

JDK 17

<https://www.oracle.com/java/technologies/downloads/#java17>

Overview

기술 상세 정보

최신 다운로드

Java SE 20

Java SE 17.0.6(LTS)

Java SE 11.0.18(LTS)

Java SE 8u361

Java Card 3.1

JDK 20

JDK 17

GraalVM for JDK 20

GraalVM for JDK 17

JDK Development Kit 17.0.8 downloads

JDK 17 binaries are free to use in production and free to redistribute, at no cost, under the [Oracle No-Fee Terms and Conditions](#).

JDK 17 will receive updates under these terms, until September 2024, a year after the release of the next LTS.

Linux

macOS

Windows

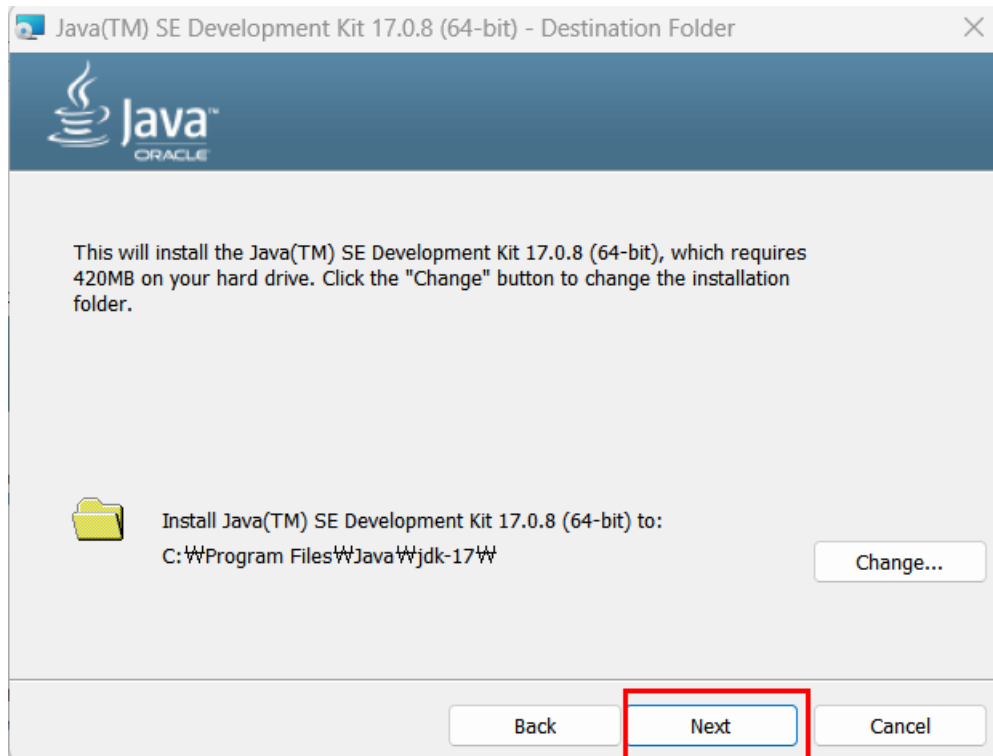
Product/file description	File size	Download
x64 Compressed Archive	172.38 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256)
x64 Installer	153.48 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256)
x64 MSI Installer	152.27 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256)

(2) JDK 설치하기

다운로드한 파일을 클릭한다 아래 화면에서 [Next]를 클릭



[Next]를 클릭

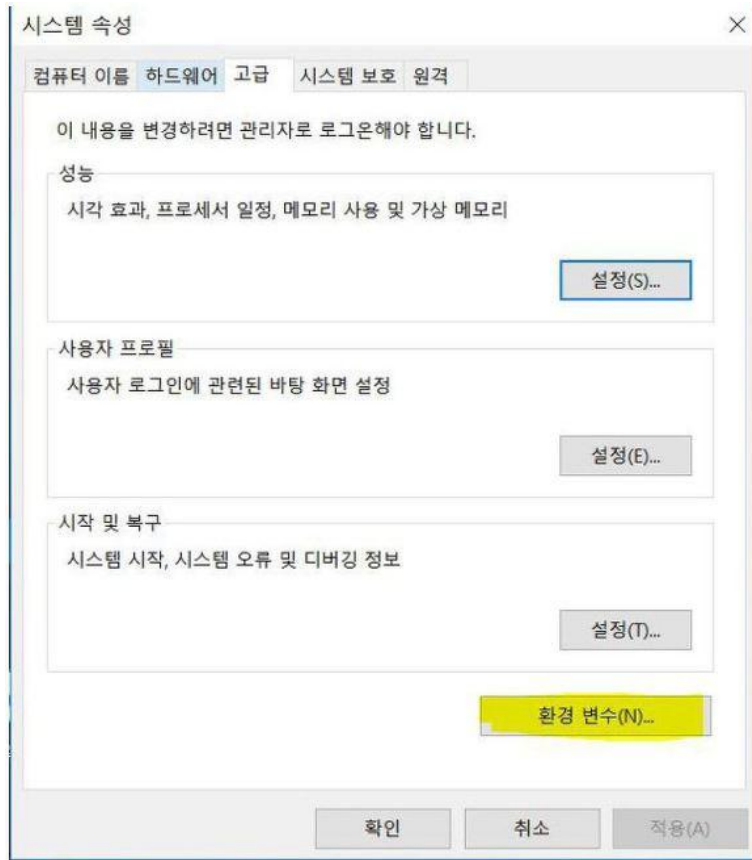


[close]을 클릭하면 설치가 완료된다.



(3) 자바 환경 변수 설정하기

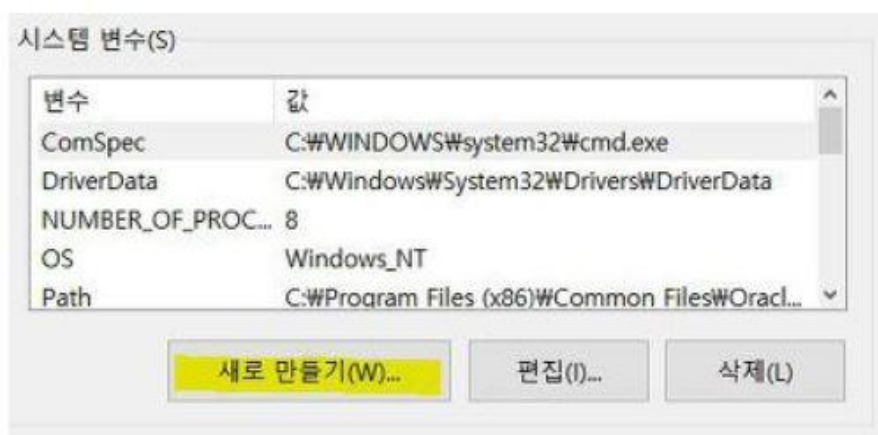
제어판>시스템 및 보안>시스템>고급 시스템 설정 으로 이동해서 [환경변수]를 클릭



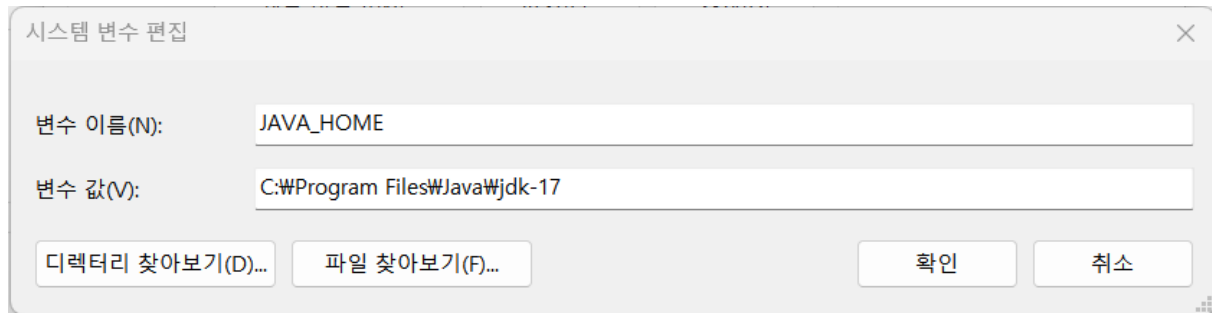
1) JAVA_HOME 시스템변수 등록(JDK설치 경로 지정)

예) C:\Program Files\Java\jdk-17

시스템 변수 항목의 [새로 만들기]를 클릭



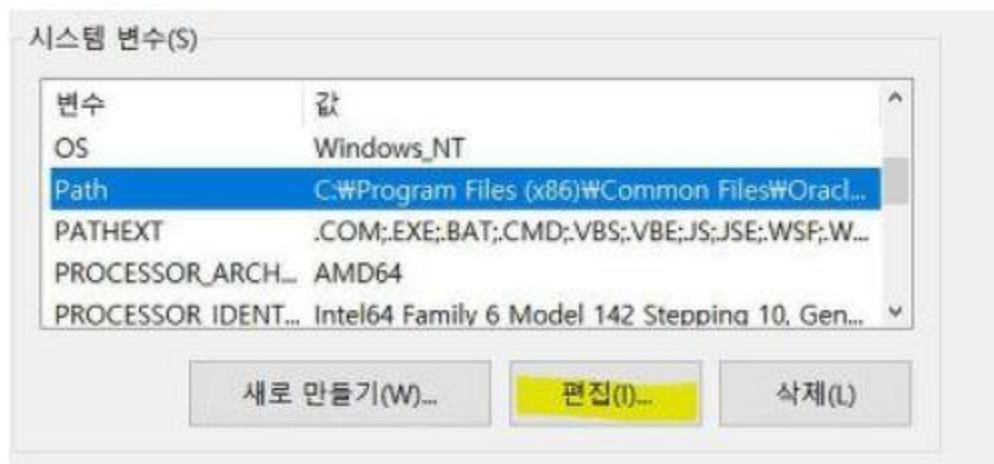
변수 이름과 변수 값을 입력하고 [확인]을 클릭



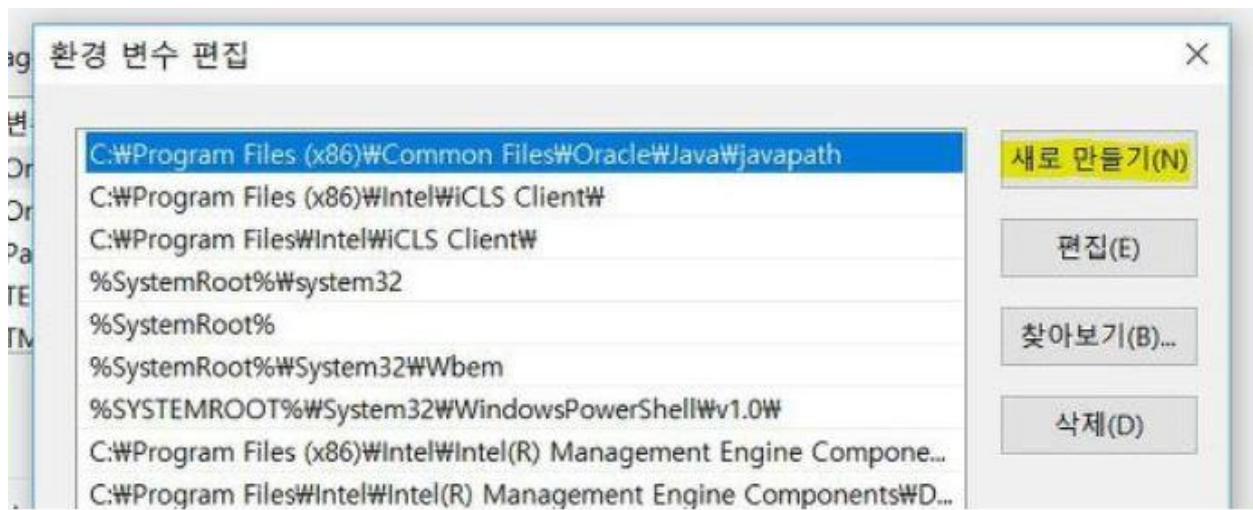
2) Path 지정(JDK의 실행파일 등록)

예) %JAVA_HOME%\bin

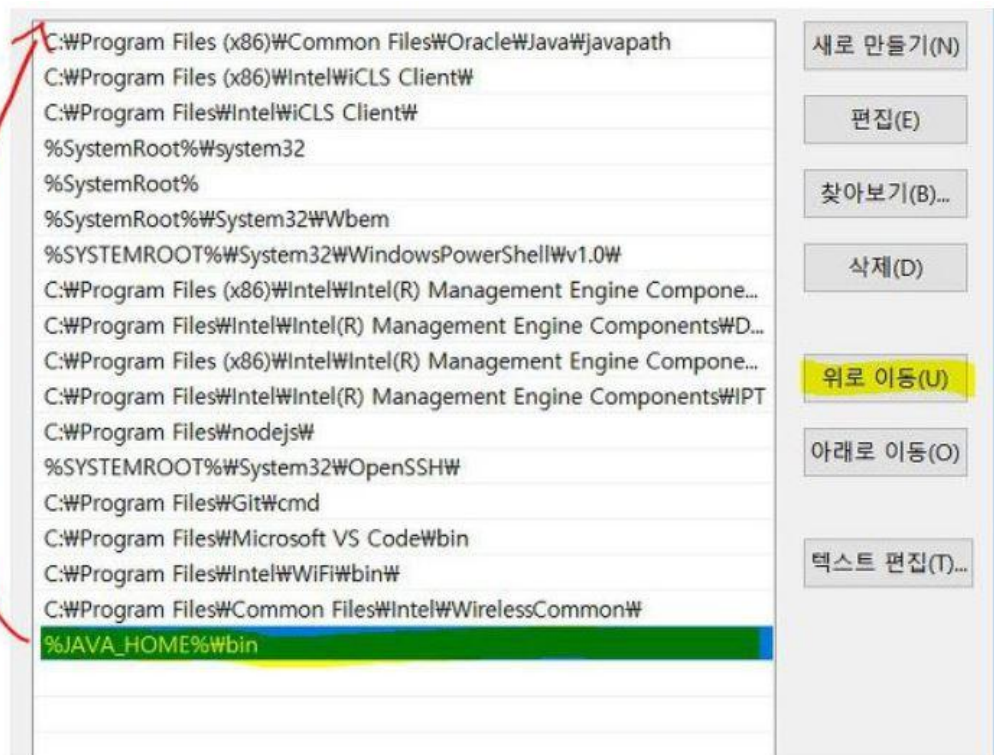
Path 항목을 선택하고 **[편집]**을 클릭



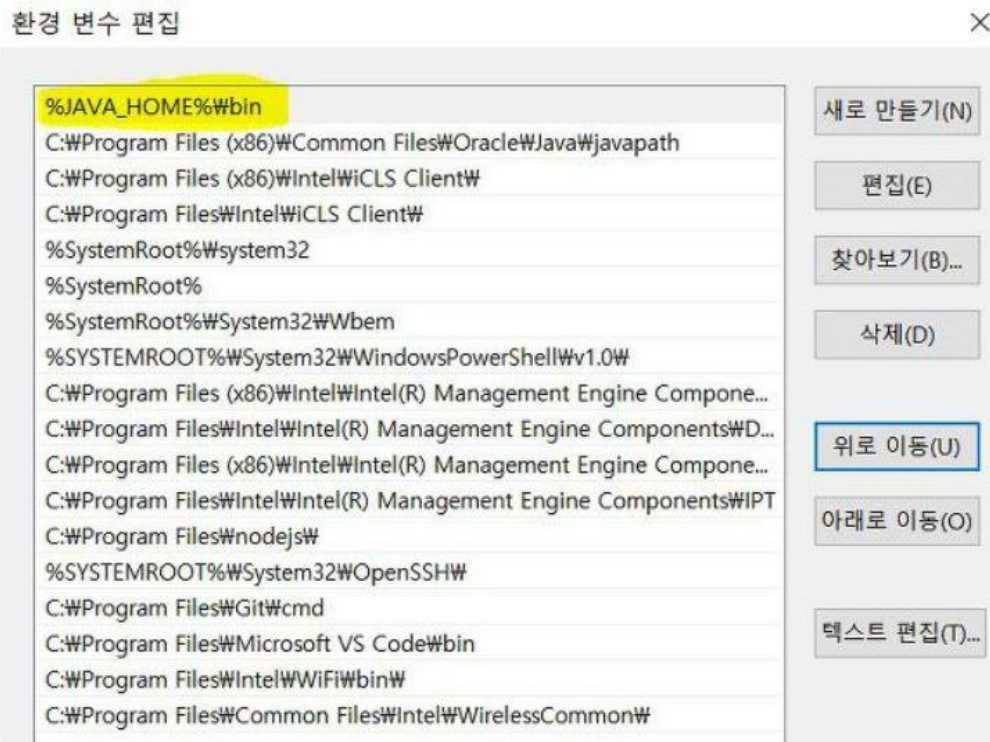
[새로 만들기]를 클릭



%JAVA_HOME%\bin 을 입력하고 맨 위로 이동시킨다



환경 변수 편집이 완료된 화면



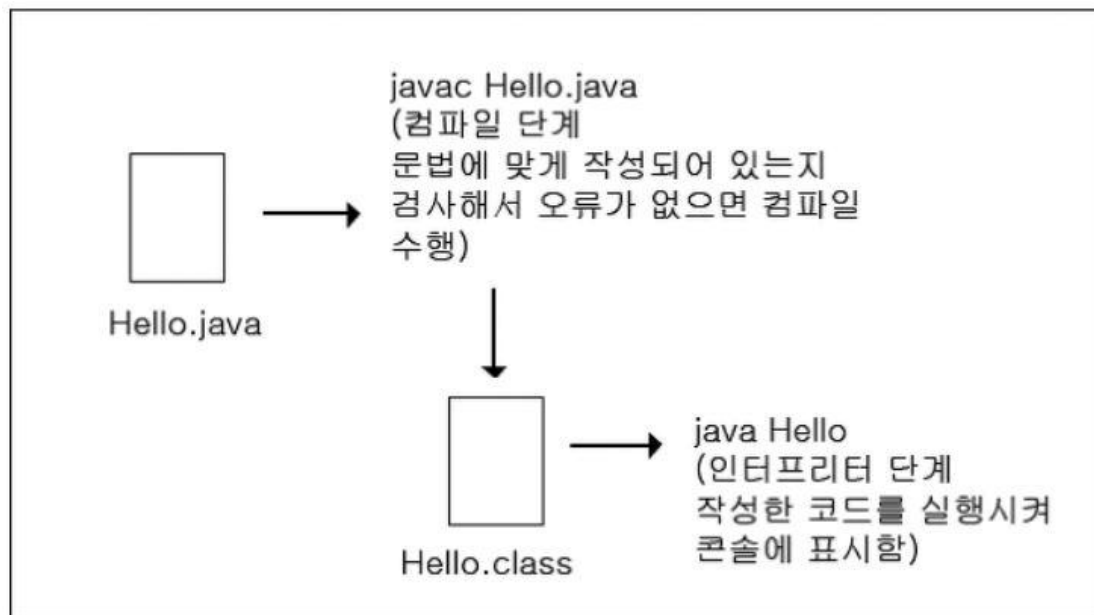
(3) 환경 변수 Path 적용 여부 확인

명령 프롬프트(cmd 모드)를 실행한 후 **java -version** 입력한다.

```
명령 프롬프트
Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

C:\Users\drago>java -version
java version "17.0.8" 2023-07-18 LTS
Java(TM) SE Runtime Environment (build 17.0.8+9-LTS-211)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.8+9-LTS-211, mixed mode, sharing)
```

1-5 자바의 실행 구조



1-6 코드 작성 및 컴파일 하기

(1) 코드 작성

```
01 public class Hello {
02     public static void main(String[] args){
03         System.out.println("Hello, world.");
04     }
05 }
```

(2) 명령 프롬프트(cmd 모드)에서 컴파일 하고 실행하기

컴파일 : javac Hello.java

```
C:\Windows\system32\cmd.exe
C:\W>cd javaWork
C:\W\javaWork>dir/w
C 드라이브의 볼륨: BOOTCAMP
볼륨 일련 번호: 1011-F916

C:\W\javaWork 디렉터리

[.]          [..]          Hello.java
              1개 파일      128 바이트
              2개 디렉터리 21,275,152,384 바이트 남음

C:\W\javaWork>javac Hello.java

C:\W\javaWork>dir/w
C 드라이브의 볼륨: BOOTCAMP
볼륨 일련 번호: 1011-F916

C:\W\javaWork 디렉터리

[.]          [..]          Hello.class  Hello.java
              2개 파일      545 바이트
              2개 디렉터리 21,274,849,280 바이트 남음

C:\W\javaWork>
```

컴파일 된 파일 실행 : java Hello

```
C:\W\javaWork>dir/w
C 드라이브의 볼륨: BOOTCAMP
볼륨 일련 번호: 1011-F916

C:\W\javaWork 디렉터리

[.]          [..]          Hello.class  Hello.java
              2개 파일      545 바이트
              2개 디렉터리 21,274,849,280 바이트 남음

C:\W\javaWork>java Hello
Hello, world.

C:\W\javaWork>
```

자바 17 API 문서

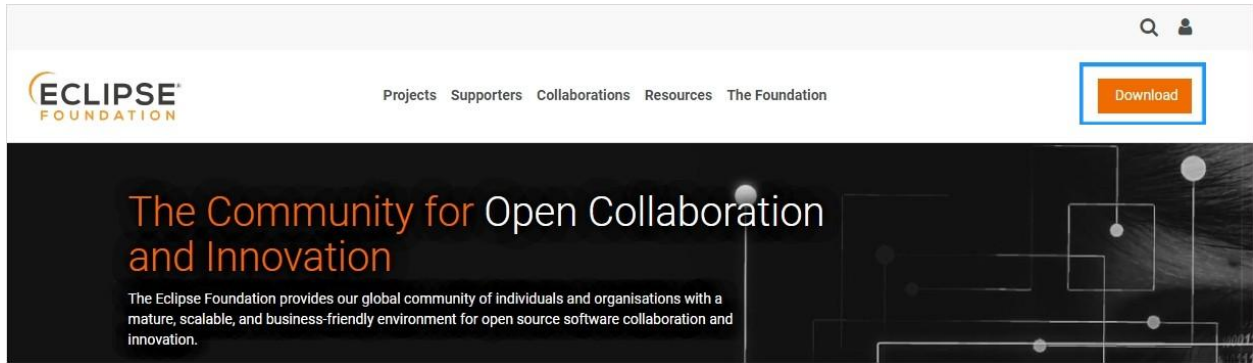
<https://docs.oracle.com/en/java/javase/17/docs/api/index.html>

1-7 에디터 설치

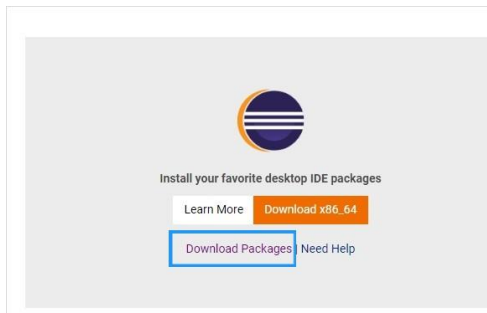
이클립스 설치 <http://www.eclipse.org>

(1) 이클립스 다운로드

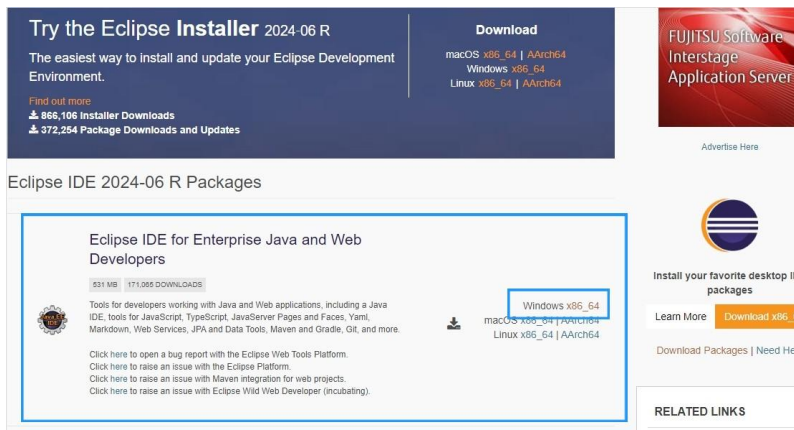
이클립스 사이트에 접속해서 첫 페이지의 [Download] 버튼을 클릭함



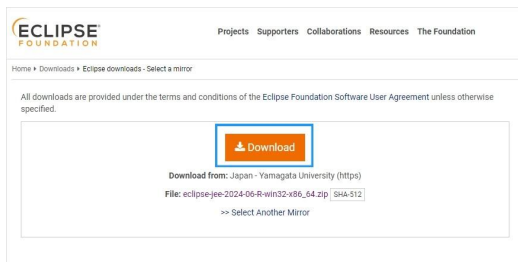
아래 화면 왼쪽의 [Download Packages]를 클릭함



아래 화면에서 Eclipse IDE for Enterprise Java and Web Developers 항목에서 Windows x86_64 클릭



아래 화면의 중앙의 [Download] 버튼을 클릭해서 파일을 다운로드함



(2) 이클립스 설치하기

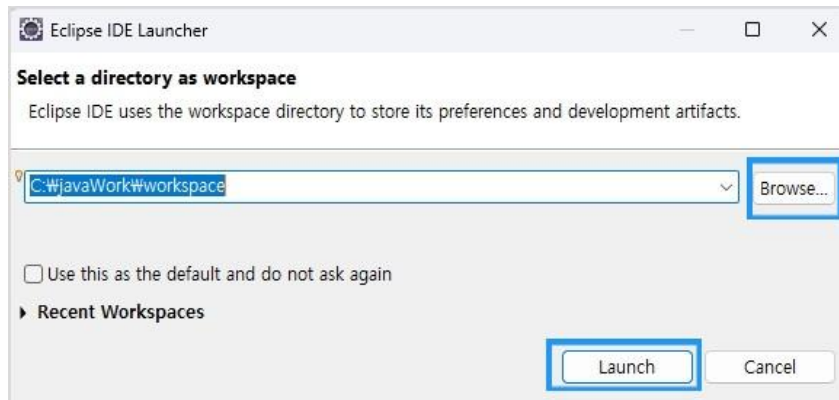
다운로드한 파일을 작업 경로에 옮겨놓고 압축을 해제



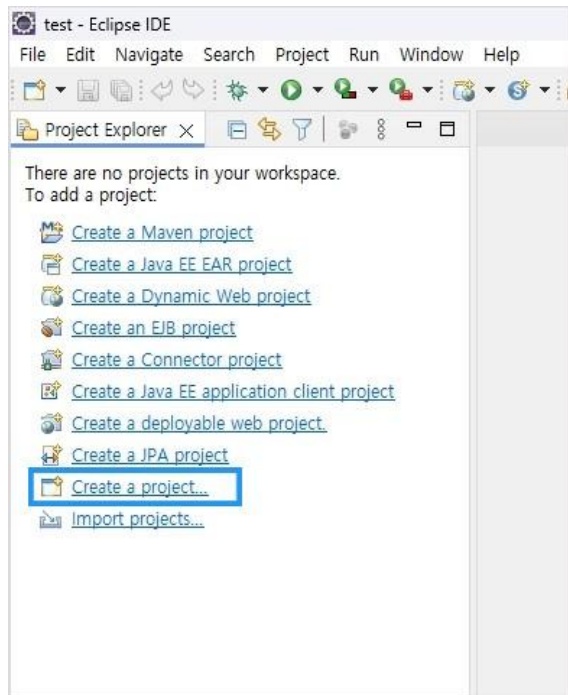
(3) 이클립스 실행하기

(3-1) Java Project 생성

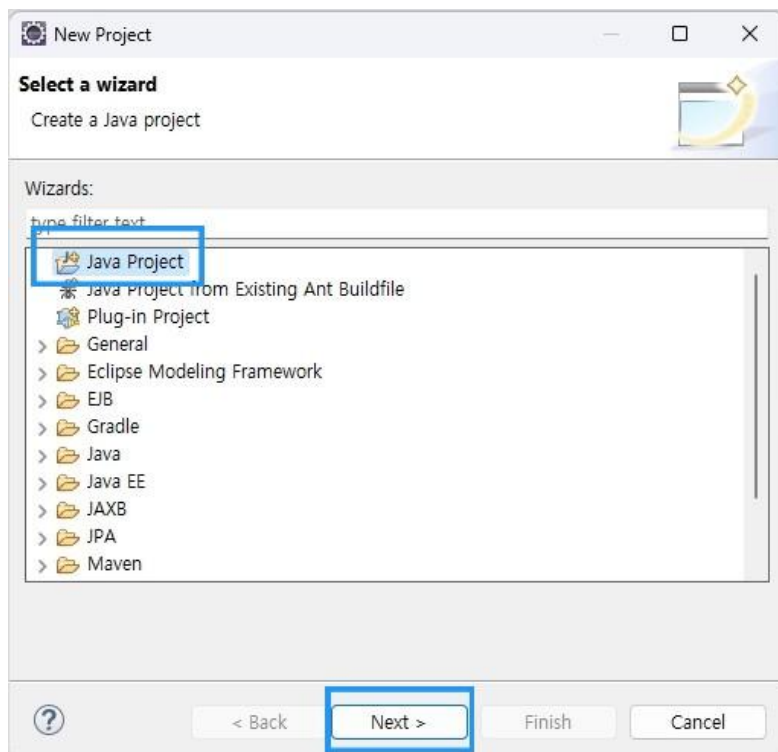
Eclipse IDE Launcher에서 [Browse..] 버튼을 클릭해서 자바 코드를 입력할 수 있는 작업 폴더를 선택하고[Launch] 버튼을 클릭한다



자바 파일을 만들고 코드를 입력하기 위해 Project Explorer 영역에서 Create a project를 클릭

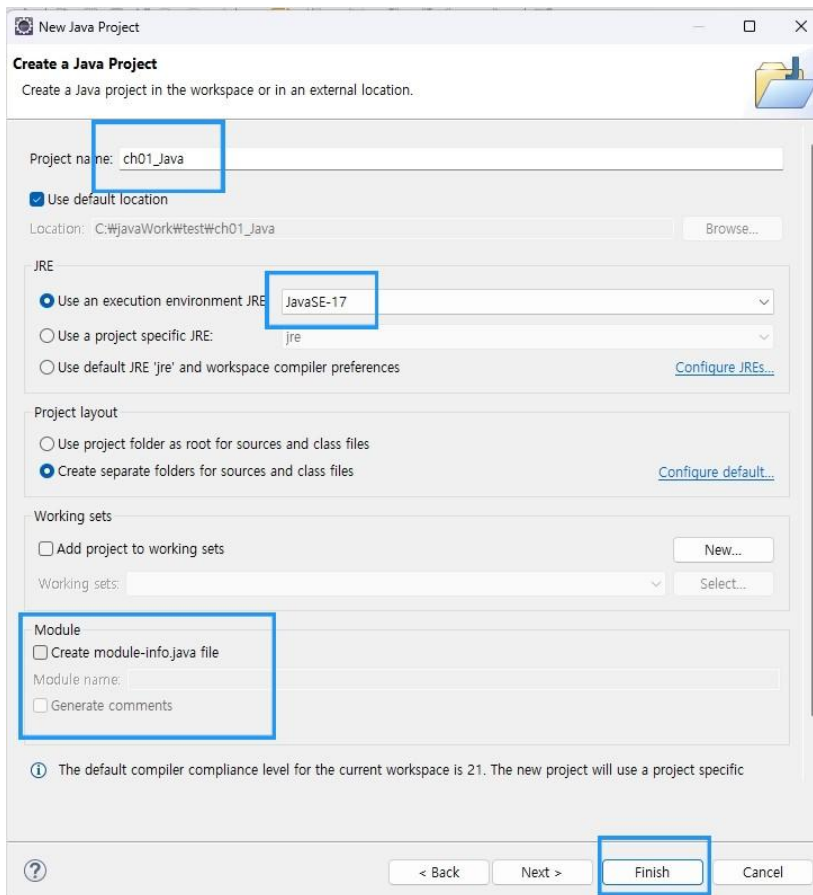


Java Project를 선택한 후 [Next] 버튼을 클릭

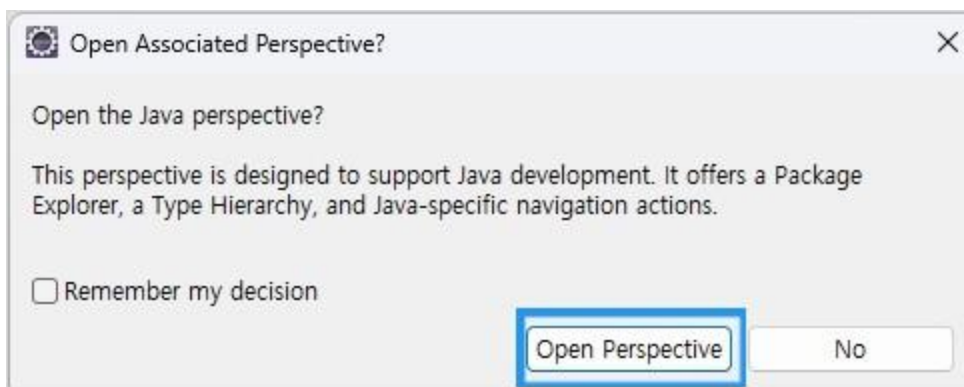


Project name을 지정하고 **JRE** 항목에서 **JavaSE-21**를 **JavaSE-17**로 변경.
Module 항목에서 **Create module-info.java file**에 체크를 해제함

[Finish] 를 클릭



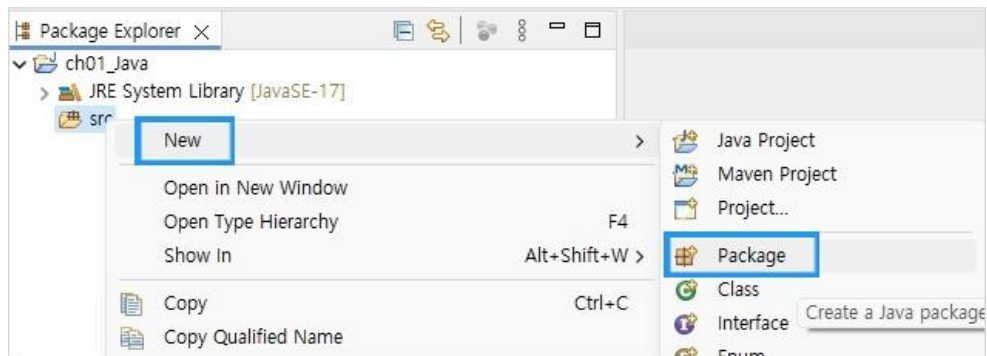
이클립스 기본 화면 구성이 **Java EE**로 되어 있는데 **Java Project**를 만들면 **Java** 코딩에 맞는 화면 구성으로 바꿀 것인지를 묻는 창이다. **[Open Perspective]**를 클릭해서 화면 구성을 **Java**로 변경한다



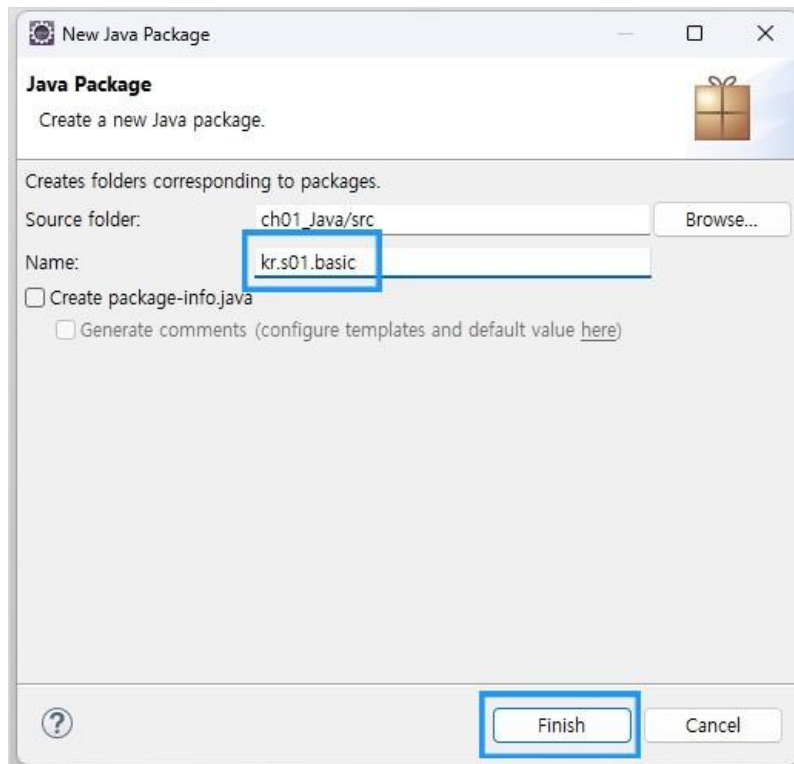
(3-2) Package 및 Java 파일 생성

- 1) 패키지 생성. **src** 폴더를 클릭하고 마우스 오른쪽 버튼을 클릭한 후 **New > Package** 를

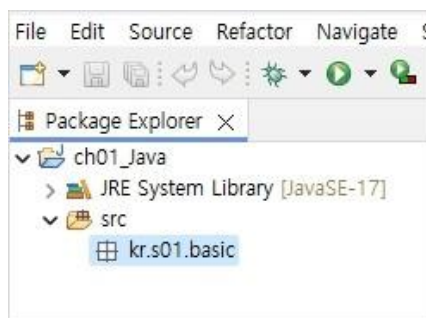
클릭한다



Package의 name을 지정하고 Finish를 클릭

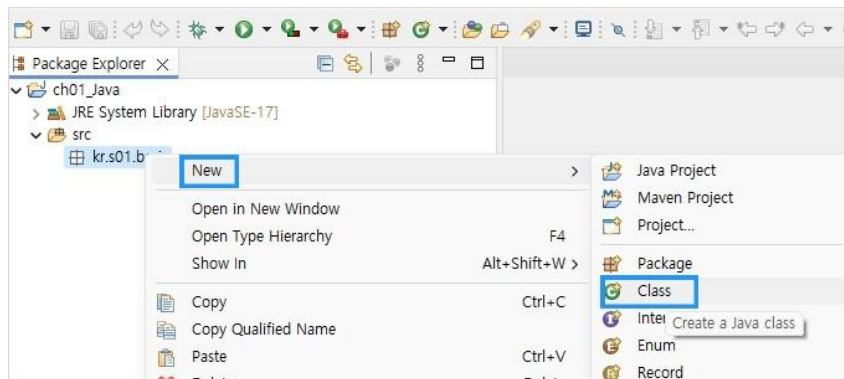


생성된 패키지

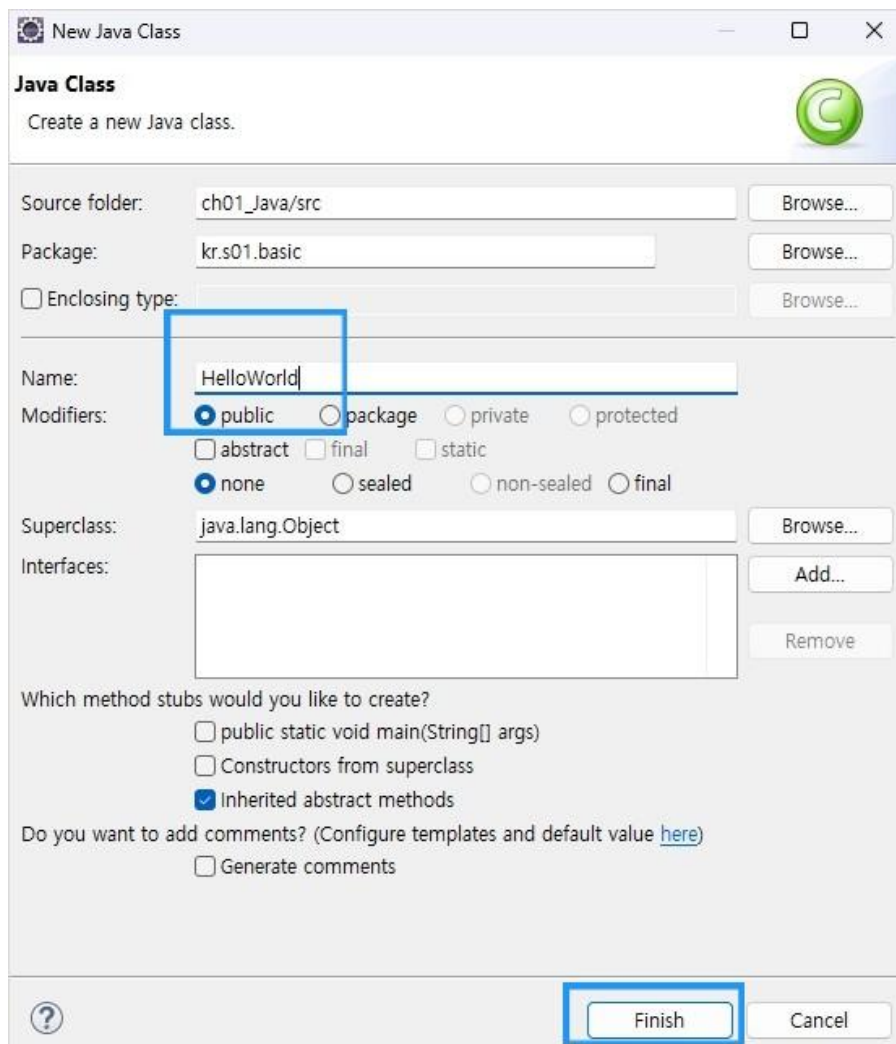


2) 클래스 생성. 생성된 패키지를 클릭하고 마우스 오른쪽 버튼을 클릭한 후 New >Class

를 클릭한다



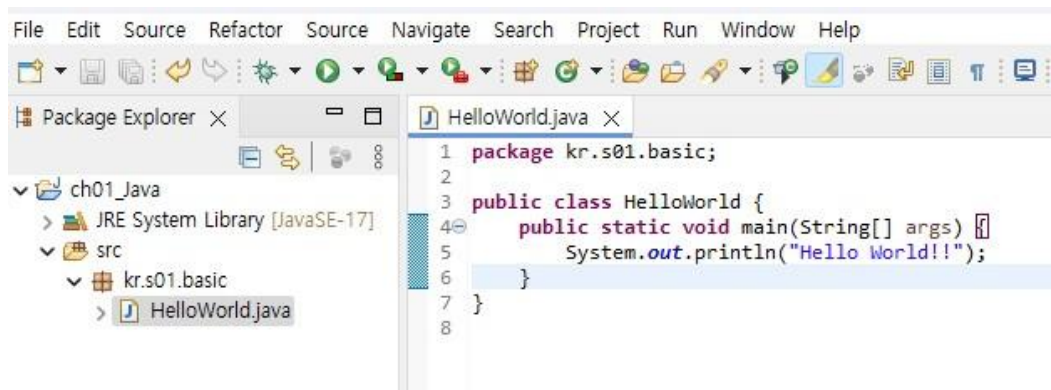
새로운 자바 클래스를 생성하기 위해서 **Name**을 지정하고 **[Finish]**를 클릭한다



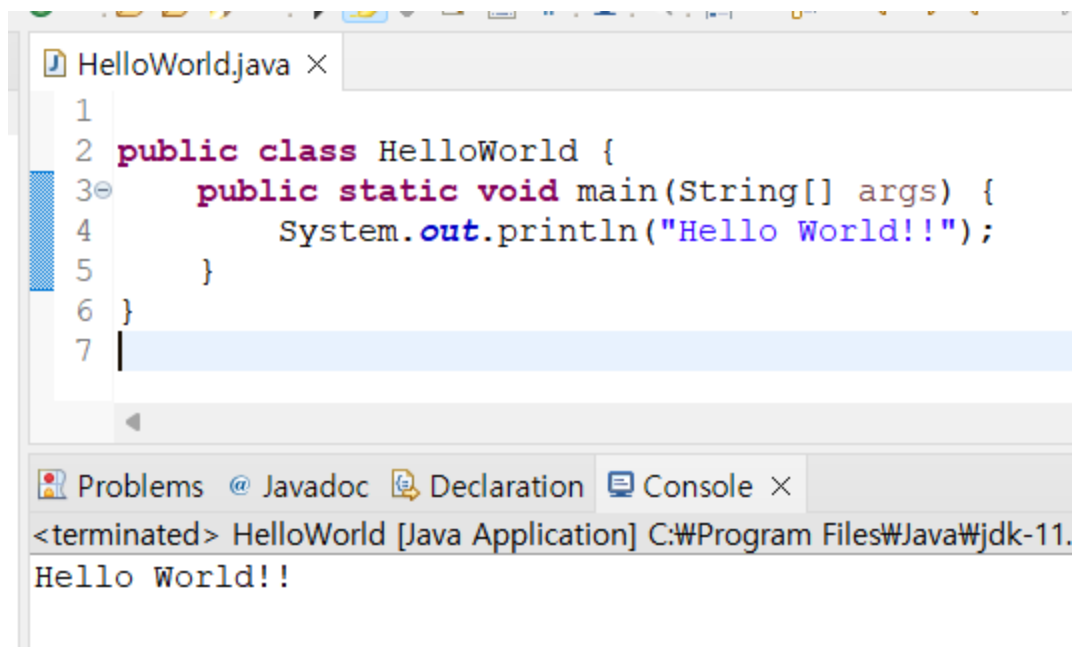
(3-3) 자바 코드 작성 및 실행

아래와 같이 코드를 작성한 후 메뉴의 **Run > Run As > Java Application**을 선택

콘솔에 코드가 실행되어 Hello World를 출력한 모습



콘솔에 코드가 실행되어 Hello World를 출력한 모습



2. 자바 프로그램 작성 기초

2-1 주석

자바 코드에 대한 설명을 작성하거나 일부 코드를 실행되지 않도록 처리하고자 할 때 주석을 사용한다.

```
01  /**
02  HTML문서화 주석 : API와 같은 도움말 페이지를 만들 때 설명 삽입 기능
03  */
04  public class Hello {
05      //한 줄 주석
```

```

06 //클래스를 실행하고자 할때 main 메서드를 반드시 작성해야 함
07 public static void main(String[] args){
08     System.out.println("Hello, world.");
09     //여러줄 주석
10     /*
11     System.out.println("Seoul");
12     System.out.println("Korea");
13     */
14 }
15 }

```

2-2 출력문

데이터의 출력

```

01 public class PrintEx {
02     public static void main(String[] args){
03         //문자열
04         //출력 후 줄바꿈
05         System.out.println("Hello World!!");
06
07         //출력 후 줄바꿈 없음
08         //System.out.print("봄");
09
10         //문자
11         System.out.println('A');
12
13         //숫자
14         System.out.println(23);
15
16         //논리값(boolean)
17         System.out.println(true);
18
19         // 포맷문자 전달될 데이터
20         System.out.printf("%c\n", 'A');
21     }
22 }

```

System.out.printf()를 이용해서 출력시 사용하는 포맷문자

서식	설명
%c	char 문자 1개를 출력하는 서식
%숫자c	char 문자 1개를 해당숫자의 자리 수만큼 확보한 상태로 출력하는 서식(숫자가 음수이면 좌측 정렬, 숫자가 양수이면 우측 정렬)

<code>%d(%o,%x)</code>	<code>byte,short,int,long</code> 형의 데이터를 10진(8진,16진)수로 출력하는 서식
<code>%숫자d(%숫자o,%숫자x)</code>	<code>byte,short,int,long</code> 형의 데이터를 10진(8진,16진)수로 해당 숫자의 자리 수만큼 확보한 상태로 출력하는 서식(숫자가 음수이면 좌측 정렬, 숫자가 양수이면 우측 정렬)
<code>%0숫자d(%0숫자o,%0숫자x)</code>	<code>byte,short,int,long</code> 형의 데이터를 10진(8진,16진)수로 해당 숫자의 자리수만큼 확보한 상태로 출력하는 서식인데 확보한 자리가 공백이면 0 값을 공백에 채우게 하는 서식(숫자가 음수이면 런타임 에러 발생)
<code>%f(%g,%e)</code>	<code>float,double</code> 형의 데이터를 실수(근사,지수)로 출력하는 서식
<code>%숫자(%숫자g,%숫자e)</code>	<code>float,double</code> 형의 데이터를 실수(근사,지수)로 해당 숫자의 자리 수만큼 확보한 상태로 출력하는 서식(숫자가 음수이면 좌측 정렬, 숫자가 양수이면 오른쪽 정렬). 또한 숫자는 소수 이하 1자리를 나타낼 수도 있다. 예를 들면 <code>'%10.2f'</code> 라는 서식은 전체 10자리를 확보하고 그 중에서 소수점 이하 2자리를 나타내라는 뜻이다. 만약 소수 이하 세 번째 자리에서 반올림이 가능하면 반올림을 하라는 뜻도 됨
<code>%0숫자(%0숫자g,%0숫자e)</code>	<code>float,double</code> 형의 데이터를 실수(근사,지수)로 해당 숫자의 자리 수만큼 확보한 상태로 출력하는 서식인데 확보한 자리가 공백이면 0 값을 공백에 채우게 하는 서식(숫자가 음수이면 런타임 에러 발생), 또한 숫자는 소수 이하 1자리를 나타낼 수도 있다. 예를 들면 <code>'%10.2f'</code> 라는 서식은 전체 10자리를 확보하고 그 중에서 소수점 이하 2자리를 나타내라는 뜻이다. 만약 소수 이하 세 번째 자리에서 반올림이 가능하면 반올림을 하라는 뜻도 된다.
<code>%s</code>	문자열(String) 데이터를 출력하는 서식
<code>%숫자s</code>	문자열(String) 데이터를 해당 숫자의 자리 수만큼 확보한 상태로 출력하는 서식(숫자가 음수이면 좌측 정렬, 숫자가 양수이면 우측 정렬)

2-3 식별자(Identifier)

(1) 식별자란?

자바코드내에서 개발자가 사용한 이름을 식별자라고 한다. 클래스이름, 변수이름, 메서드이름 등을 지정할 때 사용

(2) 식별자 명명규칙

- 1) 영문자(A~Z,a~z)와 숫자(0~9)와 ‘_’,’\$’의 조합
- 2) 첫글자는 반드시 영문자나 ‘_’로 시작. 숫자로 시작 불허
- 3) 식별자는 대소문자를 철저히 구분
- 4) 자바에서 사용되는 예약어는 식별자로 사용할 수 없다.
- 5) 상수 값을 표현하는 단어 **true**, **false**, **null**은 식별자로 사용할 수 없다.

(3) 세부 식별자 정의 규칙

구분	정의 규칙	사용 예
클래스	- 첫 문자는 항상 대문자로 표현 - 하나 이상의 단어가 합쳐질 때는 각 단어의 첫 문자들만 대문자로 표현	<pre>class JavaTest{ ...; }</pre>
변수와 메서드	- 첫 문자는 항상 소문자로 표현 - 하나 이상의 단어가 합쳐질 때는 두 번째부터 오는 단어의 첫 문자들만 대문자로 표현	<pre>String itLand; public void getTest(){ ...; }</pre>
상수	- 모든 문자를 대문자로 표현 - 하나 이상의 단어가 합쳐질 때 공백 필요 시 under score(_)를 사용하여 연결한다.	<pre>final int JAVATEST = 10; final int JAVA_TEST = 20;</pre>

(4) 예약어

- 자바 프로그래밍을 하는데 있어 특정한 의미가 부여되어 이미 만들어진 식별자를 말한다.
 - 예약어에 등록되어 있는 것을 프로그래밍에서 식별자로 사용할 수 없다.
- (**const**와 **goto**는 예약어로 등록만 되어 있을 뿐 사용되지 않는 예약어이다.)

abstract	continue	goto	package	this
assert	default	if	private	throw
boolean	do	implements	protected	throws
break	double	import	public	transient

byte	else	instanceof	return	try
case	extends	int	short	void
catch	final	interface	static	while
char	finally	long	super	
class	float	native	switch	
const	for	new	synchronized	

2-4 변수

- 값을 담아두는 기억 공간(메모리 공간)
- 데이터를 저장할 수 있는 그릇과 같음
- 하나의 데이터 값을 저장할 수 있음
- 정해진 값은 고정되어 있지 않고 계속 변할 수 있음
- 저장되는 데이터에 따라 변수의 자료형(Data Type)이 결정됨

```

01 public class Hello {
02     public static void main(String[] args){
03         int num;           //변수의 선언
04         num = 3;           //변수의 초기화
05         System.out.println(num);    //출력
06     }
07 }

```

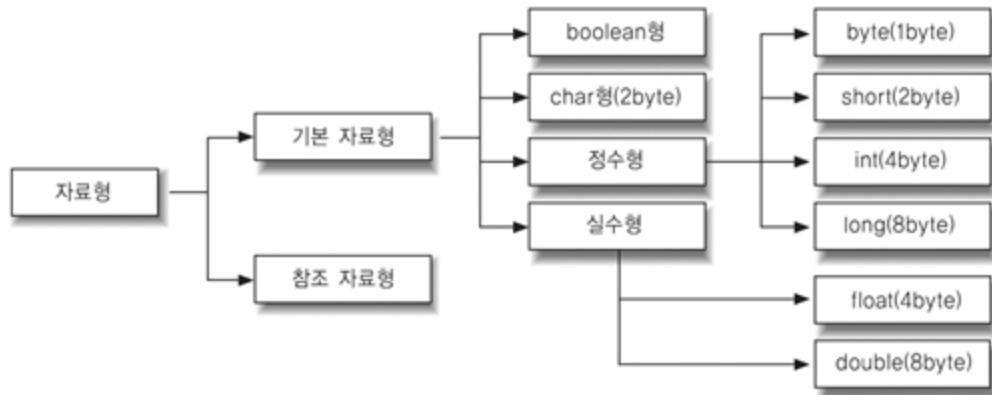
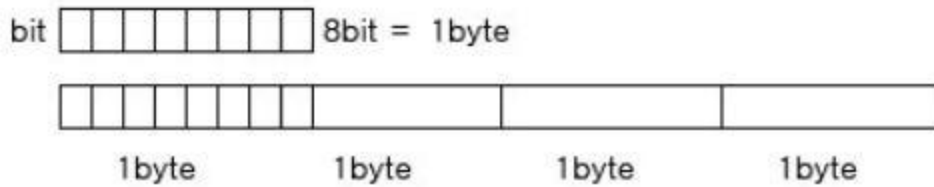
2-5 자료형

(1) 기본 자료형 (primitive data type)

자바 컴파일러에 의해서 해석되는 자료형

(2) 참조 자료형 (reference data type)

자바 **API**에서 제공되거나 프로그래머에 의해서 만들어진 클래스를 자료형으로 선언하는 경우
클래스 타입, 인터페이스 타입, 배열 타입, 열거 타입



(3) 기본자료형의 종류

자료형	키워드	크기	기본값	표현 범위
논리형	boolean	1byte	false	true 또는 false (0과1로 대체할 수 없음)
문자형	char	2byte	\u0000	0 ~ 65,535
정수형	byte	1byte	0	-128 ~ 127
	short	2byte	0	-32,768 ~ 32,767
	int	4byte	0	-2,147,483,648 ~ 2,147,483,647
	long	8byte	0	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
실수형	float	4byte	0.0	-3.4E38 ~ + 3.4E38
	double	8byte	=	-1.7E308 ~ + 1.7E308

(4) 아스키 코드

10진수	ASCII	10진수	ASCII	10진수	ASCII	10진수	ASCII
0	NULL	32	SP	64	@	96	,
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	SC2	50	2	82	R	114	r
19	SC3	51	3	83	S	115	s
20	SC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

(5) 확장 특수 출력 문자(escape sequence)

종류	의미
\a	경고음이 난다
\n	엔터 키의 기능을 갖는다. 줄을 바꾼다.(new line)
\t	수평 탭으로 일정한 간격을 띄운다.(tab)
\b	백스페이스로 뒤로 한 칸 후진한다.(backspace)

\r	동일한 줄의 맨 앞 칸으로 커서만 옮긴다(carriage return)
\f	출력 용지를 한 페이지 넘긴다.(form feed)
\\	\문자를 의미
\'	'문자를 의미(single quote)
\"	"문자를 의미(double quote)
\0	널문자를 의미(null)
\u16진수	16진수에 해당하는 유니코드

(7) 형변환

- 데이터나 변수의 자료형을 다른 자료형으로 변환 시키는 것
- 자바의 데이터는 서로 같은 자료형일 때 연산이 가능하다.
- 서로 다른 자료형들은 같은 타입으로 변경시킨 후 연산 가능하다
- 기본형과 참조형 모두 형변환이 가능하다.
(기본형은 기본형끼리만, 참조형은 참조형끼리만 형변환 가능)
- **boolean**은 **false**와 **true**만을 저장하기 위해 특별히 만들어진 데이터 타입이므로 형변환이 불가능하다.
- 명시적 형변환과 묵시적 형변환으로 나뉜다.

1) 묵시적 형변환(자동 형변환)

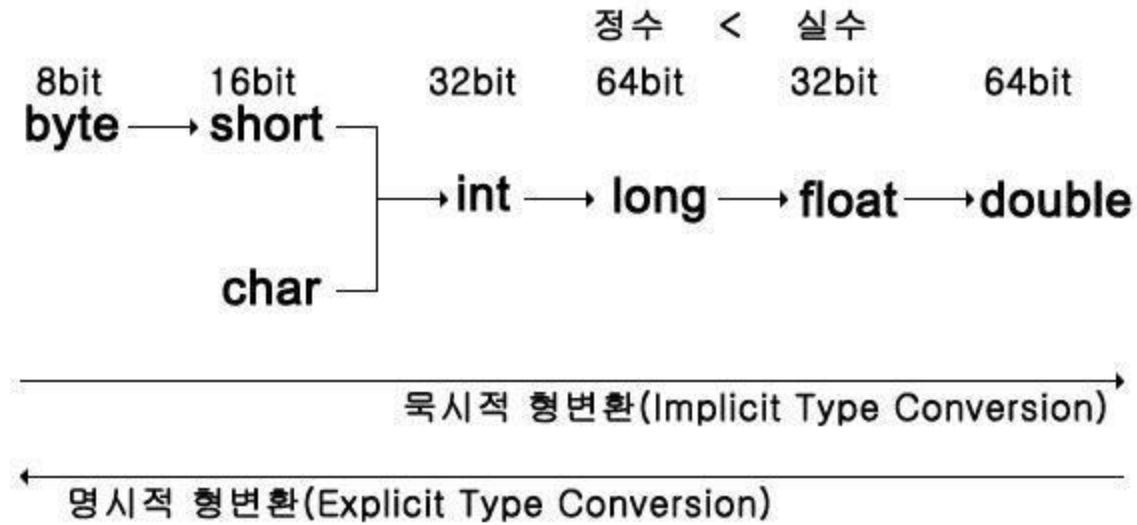
- 프로그램 실행 도중 자동으로 일어나는 형변환
- 작은 타입을 큰 타입으로 변수에 할당하면 자동으로 변환된다.
- 자동 형변환이 발생하면 변환 이전의 값과 변환 이후 값이 동일하다.

(주의)

char는 **2byte**이지만 표현범위가 **0~65535**이므로 음수를 저장할 수 없다. 음수가 저장될 수 있는 **byte**타입을 **char**타입으로 자동 형변환할 수 없다.

2) 명시적 형변환(강제 형변환)

- [데이터타입] [데이터 또는 변수]
- 넓은 범위를 표현가능 한 큰 타입의 데이터를 좁은 범위를 표현가능 한 작은 타입으로 형변환 할 때에는 명시적으로 형변환을 해야 한다.



(7) 리터럴 (literal)

소스 코드 내에 데이터 값 그대로 쓴 상수

```

01 public class LiteralEx{
02     public static void main(String[] args){
03         int num = 1; -----> 소수점이 없는 수치 리터럴은 기본적으로 int 타입
04         double sum = num + 0.5; -----> 소수점이 있는 수치 리터럴은 기본적으로
05         double 타입
06         System.out.println("sum = " + sum); -> 큰 따옴표로 묶은 문자열은 String 타입
07         System.out.println('끝'); -----> 작은 따옴표로 묶은 하나의 문자는 char 타입
08     }
09 }
  
```

3 연산자

연산자란 자료의 가공을 위해 정해진 방식에 따라 계산하고 결과를 얻기 위한 기호

종류	연산자	우선 순위
증감연산자	++, --	1순위
산술연산자	+, -, *, /, %	2순위
시프트연산자	>>, >>>, <<	3순위
비교연산자	>, <, >=, <=, ==, !=	4순위
비트연산자	&, , ^, ~	~만 1순위, 나머지는 5순위

논리연산자	&&, , !	!만 1순위, 나머지는 6순위
조건(삼항)연산자	?, :	7순위
대입연산자	=, +=, -=, *=, /=, %=	8순위

(1) 증감 연산자

1씩 증가 또는 1씩 감소시키는 연산자

연산자	의미
++	1씩 증가시킴
--	1씩 감소시킴

(2) 산술 연산자

연산자	의미
+	덧셈
-	뺄셈
*	곱하기
/	나누기
%	나머지 값 구하기

(3) 시프트 연산자

bit단위의 연산처리를 하며 자료의 가공을 위해 bit 값을 오른쪽 또는 왼쪽으로 이동하여 값에 대한 변화를 일으키는 연산자

연산자	의미
>>	bit 값을 오른쪽으로 이동(빈 자리는 부호값으로 대입)
>>>	bit 값을 오른쪽으로 이동(빈 자리는 0으로 대입)
<<	bit 값을 왼쪽으로 이동(빈 자리는 0으로 대입)

(4) 비교 연산자(관계 연산자)

변수나 상수의 값을 비교할 때 쓰이는 연산자로서 결과가 항상 true 또는 false인

논리값(boolean)

연산자	의미
>	크다
<	작다
>=	크거나 같다
<=	작거나 같다
==	같다
!=	같지 않다

(5) 비트 연산자

피연산자 즉 연산의 대상이 되는 값들을 내부적으로 **bit**단위로 변경한 후 연산을 수행

연산자	의미
&	비트 단위의 AND
	비트 단위의 OR
^	비트 단위의 XOR
~	비트 반전(0은 1로 1은 0으로 바뀜)

비트단위의 AND 연산

값1	값2	결과
0	0	0
0	1	0
1	0	0
1	1	1

비트단위의 OR 연산

값1	값2	결과
0	0	0
0	1	1
1	0	1
1	1	1

비트단위의 XOR 연산

값1	값2	결과
0	0	0

비트단위의 NOT 연산

값	결과
0	1

0	1	1
1	0	1
1	1	0

1	0
---	---

(6) 논리 연산자

true나 **false**인 논리 값을 가지고 조건 연산

연산자	의미	설명
&&	and(논리곱)	주어진 조건들이 모두 true 일 때만 true 를 나타냄
	or(논리합)	주어진 조건들 중 하나라도 true 이면 true 를 나타냄
!	부정	true 는 false 로 false 는 true 로 나타냄

- 논리 연산자의 수행 방식

연산자	설명
&&	선조건이 true 일 때만 후조건을 실행하며 선조건이 false 이면 후조건을 실행하지 않는다
	선조건이 true 이면 후조건을 실행하지 않으며 선조건이 false 일 때만 후조건을 실행한다

(7) 조건 연산자(삼항 연산자)

하나의 조건을 정의하여 만족 시에는 '참값' 을 반환하고 만족하지 못할 시에는 '거짓값' 을 반환

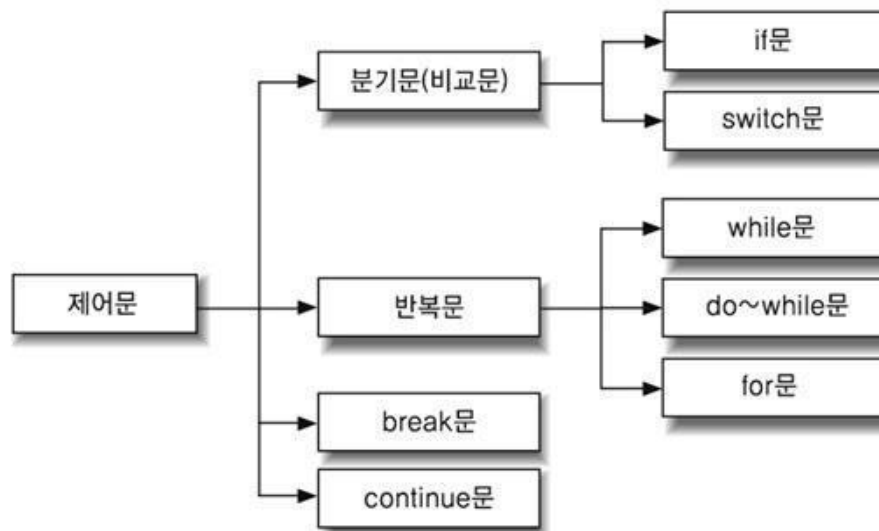
연산자	의미	설명
?:	조건을 만족하면 참값, 만족하지 못하면 거짓값 반환	조건?참값:거짓값

(8) 대입 연산자

연산자	의미
=	연산자를 중심으로 오른쪽 변수 값을 왼쪽 변수에 대입
+=	연산자 왼쪽 변수의 값과 연산자 오른쪽의 값(변수의 값)을 덧셈하여 왼쪽 변수에 대입

-=	연산자 왼쪽 변수의 값과 연산자 오른쪽의 값(변수의 값)을 뺄셈하여 왼쪽 변수에 대입
*=	연산자 왼쪽 변수의 값과 연산자 오른쪽의 값(변수의 값)을 곱하여 왼쪽 변수에 대입
/=	연산자 왼쪽 변수의 값과 연산자 오른쪽의 값(변수의 값)을 나누어 왼쪽 변수에 대입
%=	연산자 왼쪽 변수의 값과 연산자 오른쪽의 값(변수의 값)을 나누어 만들어진 나머지를 값을 왼쪽 변수에 대입

24. 제어문



(1) if

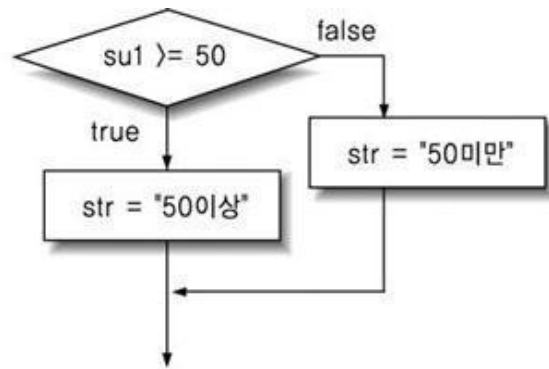
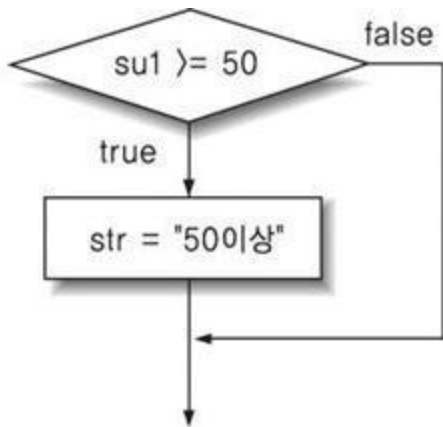
주어지는 조건의 결과가 **true**이면 if문 블록의 코드를 실행함

단일 if문

```
if(su1 >= 50)
    str = "50이상";
```

if ~ else

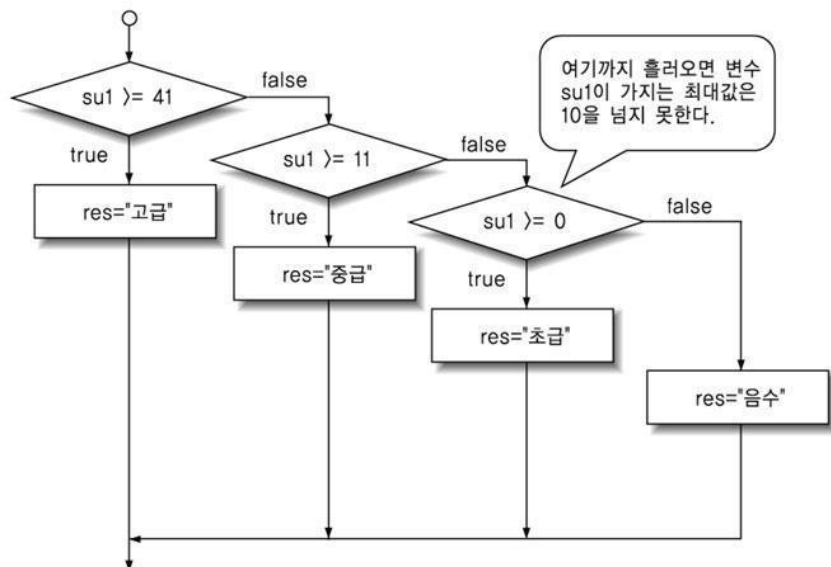
```
if(su1 >= 50)
    str = "50이상";
else
    str = "50미만";
```



다중 if문

```

if(su1 >= 41)
    res = "고급";
else if(su1 >= 11)
    res = "중급";
else if(su1 >= 0)
    res = "초급";
else
    res = "음수";
  
```



(2) switch

If문의 조건값은 boolean형인데 비해 switch문의 조건값은 long형을 제외한 정수형(byte, short, int) 또는 char형인 것이 다르다. **JDK7.0**이상부터는 문자열도 지원함

```

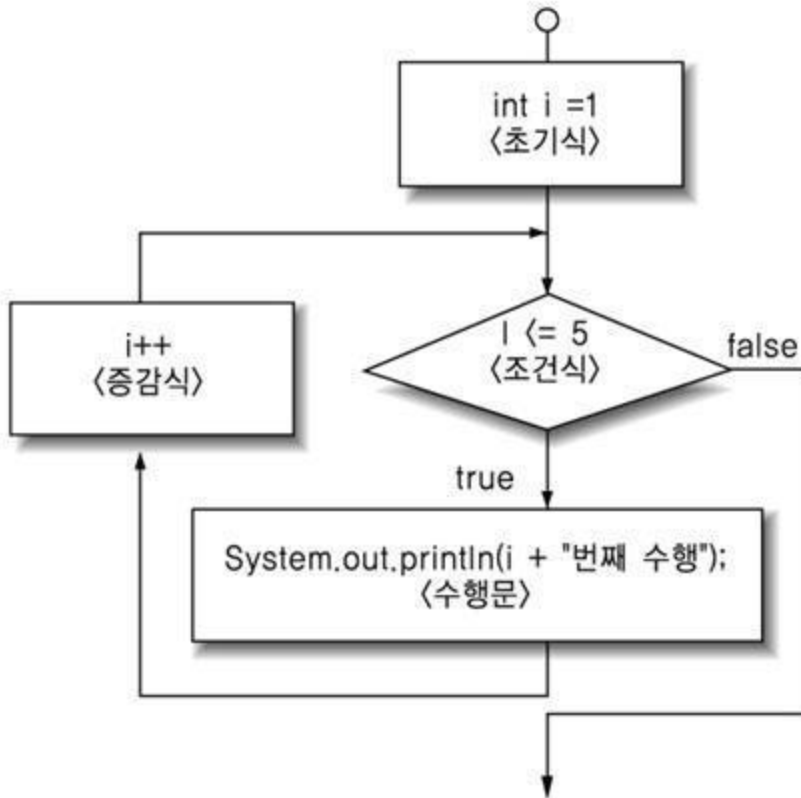
switch(인자값) {
case 조건값1 :
    수행문; break;
case 조건값2 :
    수행문; break;
case 조건값3 :
    수행문; break;
default :
    수행문;
  
```

```
}
```

(3) for

특정한 명령들을 정해진 규칙에 따라 반복처리 할 때 사용

```
for(초기식 ; 조건식 ; 증감식){  
    수행문1;  
    수행문2;  
}  
  
for(int i = 1 ; i <= 5 ; i++){  
    System.out.println(i+"번째 수행");  
}
```



다중 for문

단일 for문에서 끝나는 것이 아니라 그것을 다시 여러 번 반복하는 제어문이다. 다시 말해서 for문 안에 for문이 있는 경우를 다중 for문이라 한다.

```
for(초기식1 ; 조건식1 ; 증감식1) {
```

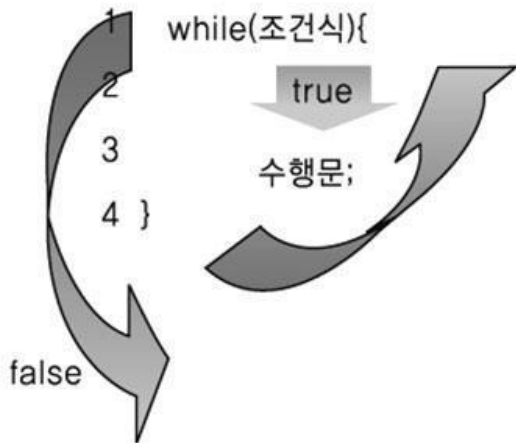
```
    for(초기식2 ; 조건식2 ; 증감식2){  
        명령어2;  
    }
```

```
    명령어1;
```

```
}
```

(4) while

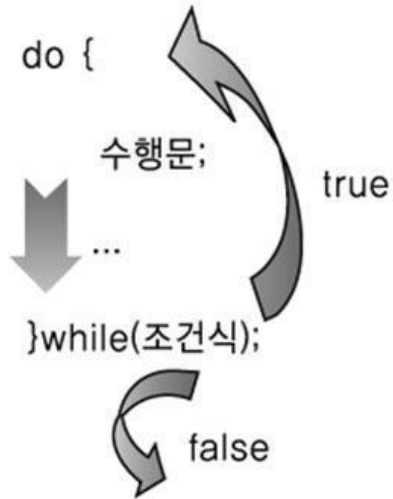
조건비교에 만족 할 때에만 반복 처리하는 제어문. 선 비교, 후 처리



```
01 public class WhileEx{  
02     public static void main(String args[]){  
03         int cnt = 0;  
04         while(cnt < 10){  
05             System.out.println(cnt);  
06             cnt++;  
07         }  
08         System.out.println("끝!");  
09     }  
10 }
```

(5) do ~ while

조건비교에 불 만족하다 할지라도 무조건 한번은 수행. 선 처리, 후 비교



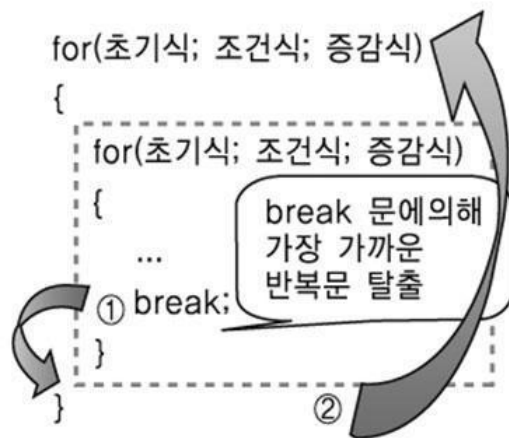
* while 문과의 차이점

- 조건식을 검사하기 전에 무조건 실행 부분을 한 번 실행
- 마지막에 세미콜론(;)을 반드시 써야 함

```
01 public class DoWhileEx{
02     public static void main(String args[]){
03         int cnt = 0;
04         //do 블록을 한 번 실행하고 나서 조건을 체크함
05         do{
06             System.out.println(cnt);
07             cnt++;
08         }while(cnt < 10);
09         System.out.println("끝!");
10     }
11 }
```

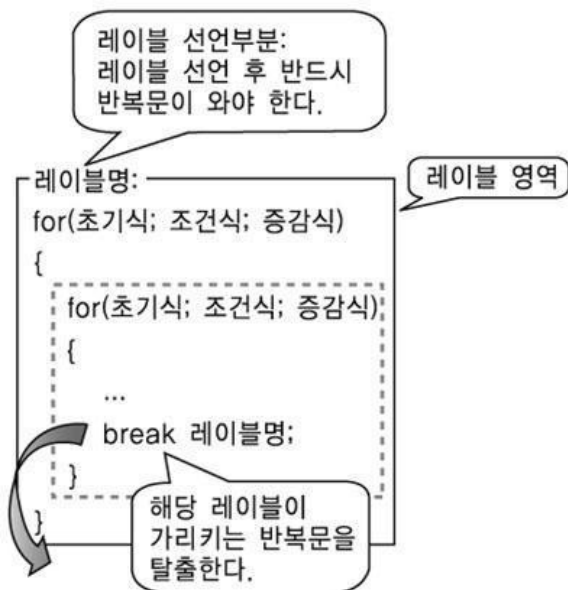
(6) break

가장 가까운 반복문을 탈출할 때 쓰이는 제어문



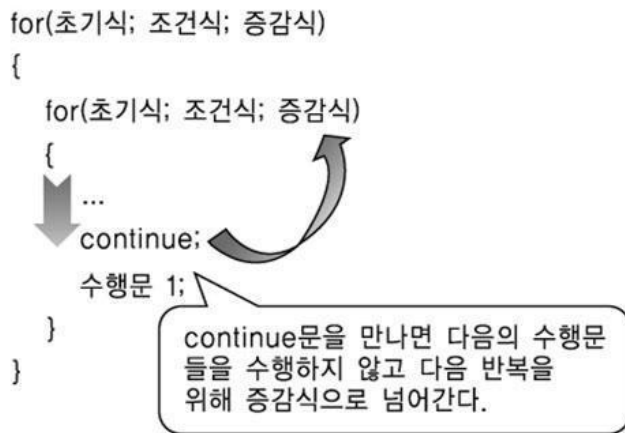
break label문

break label은 break문과 같지만 다중 반복문에서 한번에 바깥쪽 반복문을 탈출할 때 많이 쓰이는 제어문



(7) continue

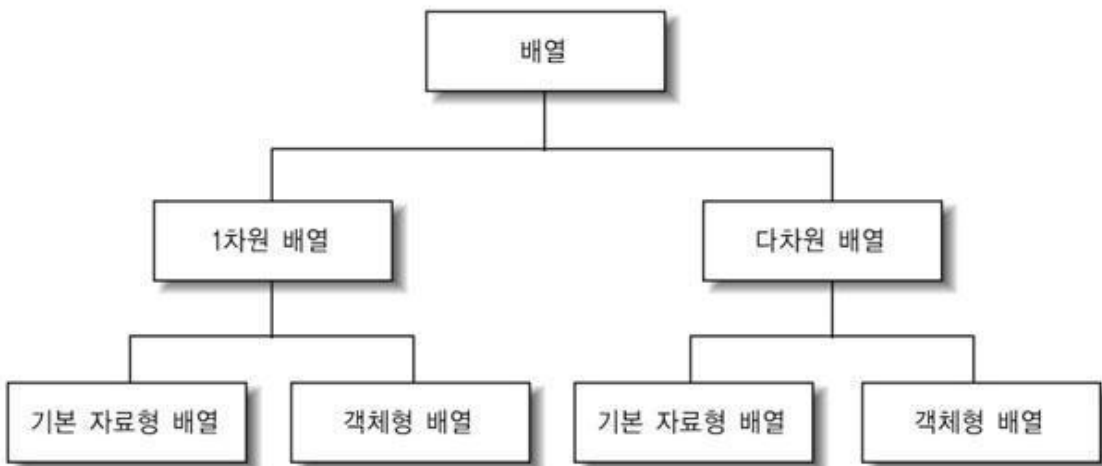
반복문을 탈출하기위해 사용되는 것이 아니라 continue문 이하의 수행문들을 포기하고 다음 회차의 반복을 수행하기위한 제어문



5. 배열

- 배열은 같은 자료형들끼리 모아두는 하나의 묶음
- 자바에서 하나의 배열은 하나의 객체로 인식

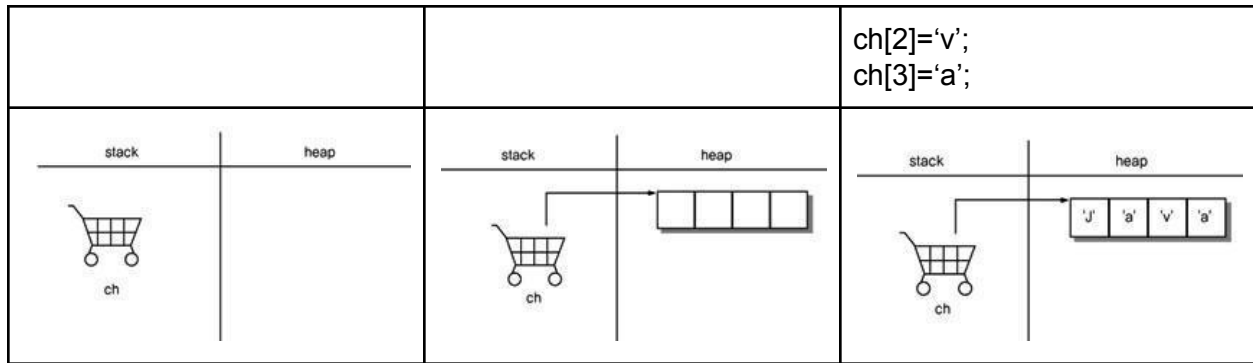
배열의 종류와 구분



5-1 1차원 배열

- 배열의 단계적 작업

배열의 선언 char[] ch; 또는 char ch[];	배열의 생성 ch = new char[4];	배열의 초기화 ch[0]='J'; ch[1]='a';
------------------------------------	-----------------------------	-------------------------------------



- 배열의 선언, 생성, 초기화 예

```

01 public class ArrayEx{
02     public static void main(String[] args){
03         //배열 선언
04         char[] ch;
05         //배열 생성
06         ch = new char[4];
07
08         //배열 초기화
09         ch[0] = 'J';
10         ch[1] = 'a';
11         ch[2] = 'v';
12         ch[3] = 'a';
13
14         //배열 내용 출력
15         for(int i = 0 ; i < ch.length ; i++)
16             System.out.println("ch["+i+"]:"+ch[i]);
17     }
18 }
19
20

```

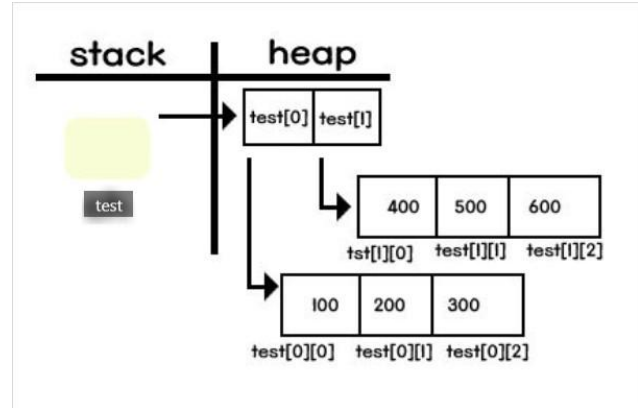
5-2 2차원 배열

```

public class ArrayEx {
    public static void main(String[] args){
        //2차원 배열 선언
        int[][] test;
        //2차원 배열 생성
        test = new int[2][3];
        //2차원 배열 초기화
        test[0][0] = 100;
        test[0][1] = 200;
        test[0][2] = 300;

        test[1][0] = 500;
        test[1][1] = 600;
        test[1][2] = 700;
    }
}

```



5-3 클래스 main 메서드

클래스를 실행시키면 클래스의 main 메서드가 실행되고 main 메서드 내부의 코드가 동작한다. main 메서드를 살펴보면 main(String[] args)로 표시되어 있어 main 메서드 실행시 문자열 배열을 인자로 전달받고 있다. 이 표시는 실제로 클래스 실행시 외부에서 데이터를 클래스의 main 메서드에 전달할 수 있다는 것을 의미함

```

>java Season 봄 여름 가을 겨울 ----- 클래스 실행시 입력한 데이터가 배열로
|                                           main 메서드에 인자로 전달된다.
V
public class Season{
    public static void main(String[] args){
        for(int i = 0; i<args.length; i++){
            System.out.println(args[i]);
        }
    }
}
[출력]
봄
여름
가을
겨울

```

6. 클래스와 객체

클래스(Class)는 한 마디로 건물을 지을 때 제일 먼저 필요로 하는 설계도면과 같다. 객체(Object)는 설계도면을 통해 완성된 건물이다. 건물이 지어지면 건물에 주소(Reference)가

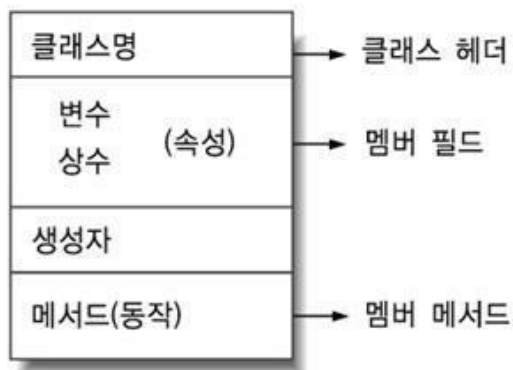
부여되고 주소를 통해 건물을 찾아갈 수 있다.

객체(**Object**)는 컴퓨터, 책상, 사람, 강아지 등 현실 세계에서 흔히 찾아볼 수 있는 대상을 모델링한 것이다. 모든 사물을 프로그램의 객체로 만들 필요는 없고 프로그램에서 필요한 대상들만 객체화한다.

객체(**Object**)란 물리적으로 존재하거나 추상적으로 생각할 수 있는 것 중에서 자신의 속성을 가지고 있고 다른 것과 식별 가능한 것을 말한다. 객체는 속성과 동작으로 구성되어 있는데 자바에서는 이 속성과 동작들을 각각 필드(**field**)와 메서드(**method**)라 부른다.

객체로 정의된 대상들의 관계를 표현하는 프로그래밍이 객체 지향 프로그래밍이다.

6-1 클래스의 구조



(1) 클래스 헤더

클래스 헤더는 클래스를 선언하는 부분을 의미

```
[접근제한] [클래스 종류] class 클래스명  
public      abstract      Hello  
                final
```

- 접근제한

접근제한은 말 그대로 현재 클래스를 접근하여 생성하고 사용하는데 있어 제한을 두겠다는 의미. 클래스 접근제한은 **public**과 **default** 두가지를 사용할 수 있음

- 클래스 종류

클래스의 종류는 일반 클래스는 명시하지 않으면 추상클래스는 **abstract**를 명시하고 상속을 금지할 때 사용하는 **final**이 있음

- 클래스명

클래스의 이름을 의미. 식별자 명명규칙에 의해 대문자로 시작함.

```
public class Hello{  
  
    .....  
}
```

(2) 멤버필드

변수 또는 상수로 구성되어 있다. 상수는 고정된 값을 의미하며 프로그램이 종료 될 때까지 절대로 변하지 않는 값을 의미한다.

변수는 상수와는 반대로 프로그램이 종료 되기 전에 변경될 수 있는 값을 의미한다.

```
public class Hello{  
    int a; //변수  
    final int NUMBER = 10; //상수  
}
```

(3) 생성자

객체 생성시 호출되어 멤버변수를 초기화하는 역할을 수행. 생성자 내부에서 특정 작업을 수행할 수 있고 데이터를 인자에 전달하여 전달받은 데이터를 활용할 수도 있다.

(4)멤버 메서드

메서드는 특정한 일을 수행하는 행위, 다시 말해 동작을 의미한다. 메서드가 갖고 있는 코드를 수행할 수도 있고 멤버 필드가 가지고 있는 데이터를 활용해 동작을 수행할 수도 있다.

```
public class Hello{  
    //멤버메서드  
    public void drive(){  
        System.out.println("운전하다");  
    }  
}
```

6-2 객체의 생성 및 멤버 접근법

1)객체 선언

객체가 생성되면 객체의 주소를 보관할 변수를 할당하는 과정. 변수를 할당할 때는 변수앞에 자료형을 명시해주는데 생성할 객체의 클래스명이 자료형으로 사용되고 해당 자료형을 참조자료형이라고 한다.

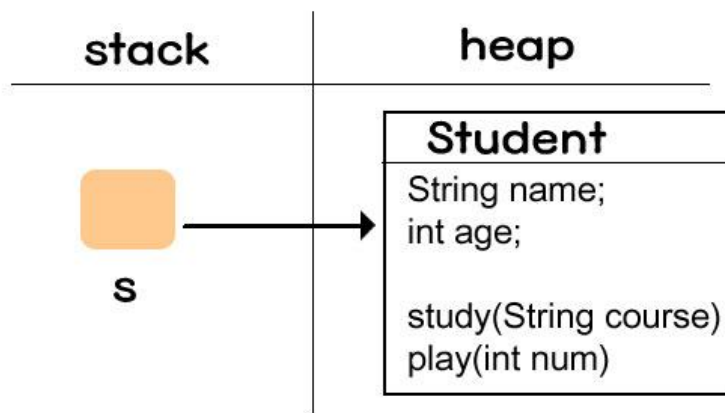
Student s;



2) 객체 생성

new 연산자를 사용하여 객체 생성을 희망하는 클래스의 생성자를 명시하고 메모리상에 객체를 생성한다. 객체가 생성되면 주소를 통해 객체에 접근할 수 있는데 주소를 참조변수에 보관하고 원할 때 참조변수를 통해 객체에 접근한다.

s = new Student();



(3) 멤버 접근법

참조변수에 저장된 객체의 주소를 통해 **heap** 영역에 생성된 객체의 멤버필드 또는 멤버 메서드를 호출한다.

```
s.study("국어");
```

6-3 멤버변수와 멤버 메서드

(1) 멤버 변수

1) 인스턴스 변수

객체가 생성될 때 각 객체들마다 따로 따로 생성 되어 고유의 값을 받아 각 객체의 속성으로 자리 잡는 변수가 바로 **instance**변수이다.

```
public class Hello{  
    String color;  
    int memory;  
}
```

2) static(클래스) 변수

여러 개의 객체가 생성될 때 단 하나만 생성 되며 모든 객체들이 공유하는 개념으로 사용되는 변수가 **static**변수이다.

```
public class Hello{  
    String color;  
    int memory;  
    static String maker;  
}
```

(2) 멤버 메서드

1) 인스턴스 메서드

객체 생성시 객체에 포함되어 객체 범위에서 실행되는 메서드. 객체를 생성해야만 호출이 가능함

메서드의 구성

[접근제한] [반환형] [메서드명](자료형 인자1, 자료형 인자2,...){

```
    수행문1;  
    수행문2;  
    ...;  
}
```

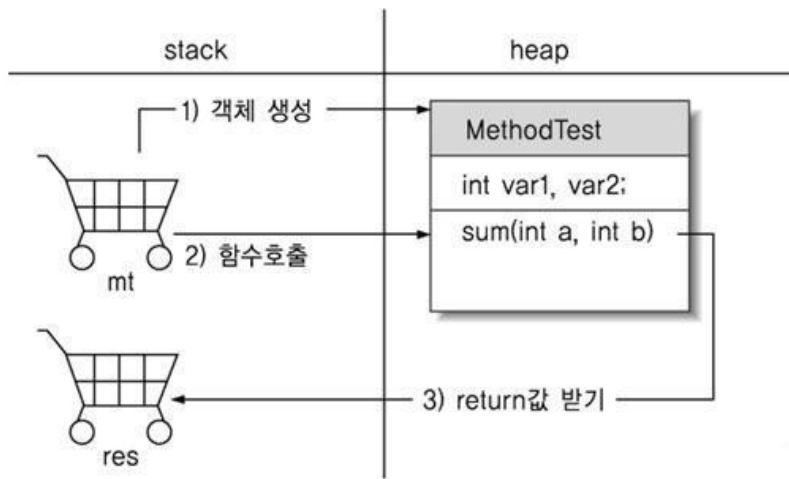
```
public int add(int a, int b){  
    return a + b;  
}
```

```
public class MethodEx {  
  
    int var1,var2; // 멤버 변수  
  
    public int sum(int a, int b){ // 멤버 메서드  
        return a+b;  
    }  
}
```

```

    }
    public static void main(String[] args){
        MethodEx me = new MethodEx();
        int res = me.sum(1000, -10);
        System.out.println("res="+res);
    }
}

```



2) static 메서드

객체 생성과 무관하며 객체를 생성하지 않아도 메서드를 직접 호출할 수 있음

```

public static int add(int a, int b){
    return a + b;
}

```

6-4 패키지과 import

(1) 패키지

서로 관련 있는 클래스와 인터페이스를 하나의 단위로 묶는 것을 의미. 서로 관련있는 파일을 보관할 때 디렉토리를 사용하는 것처럼 동일 작업을 패키지로 그룹핑하는 작업을 할 수 있다.

```
package 경로명;
```

(2) 패키지 사용 및 실행

클래스에 패키지 명시하기

```

package mypack.p1;
public class MyPackOne{

```

```

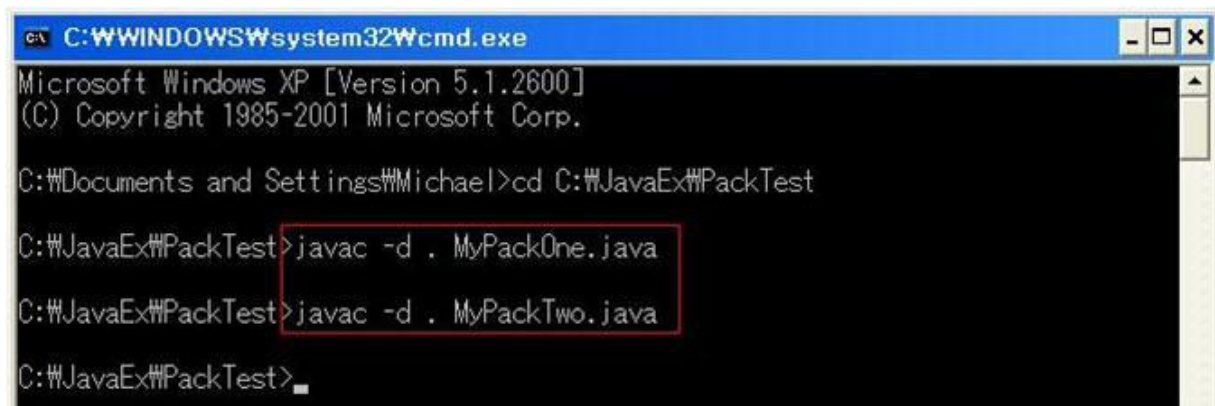
    public void one(){
        System.out.println("MyPackOne클래스의 one메서드");
    }
}

package mypack.p2;
public class MyPackTwo{

    public void two(){
        System.out.println("MyPackTwo클래스의 two메서드");
    }
}

```

cmd창에서 클래스 컴파일 및 실행



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Michael>cd C:\JavaEx\PackTest

C:\JavaEx\PackTest>javac -d . MyPackOne.java
C:\JavaEx\PackTest>javac -d . MyPackTwo.java
C:\JavaEx\PackTest>_

```

(3) import문

같은 패키지의 클래스들은 클래스명만 명시해서 호출 가능하지만 다른 패키지에 있는 특정한 클래스를 사용하려면 패키지까지 명시해서 클래스를 호출해야 한다. mypack.p1 패키지의 MyPackOne이 mypack.p2의 MyPackTwo를 호출하려면 mypack.p2.MyPackTwo라고 명시해야 호출이 가능하다. 매번 이렇게 패키지를 함께 명시하지 않고 클래스명만 표시하기를 원한다면 import문을 사용할 수 있다.

```

package mypack.p1

import mypack.p2.MyPackTwo;

public class MyPackOne{

    public void one(){
        MyPackTwo two = new MyPackTwo();
    }
}

```

```

    }
}

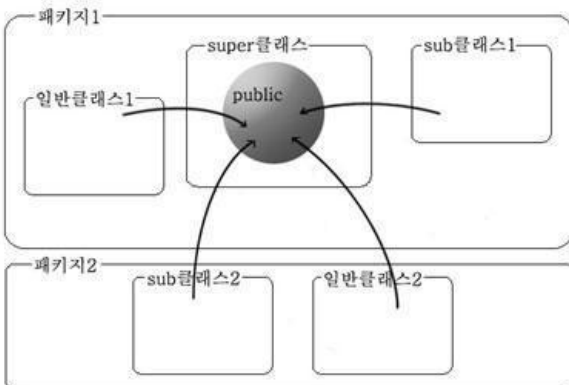
```

6-5 접근 제한자

클래스를 설계할 때에는 외부 클래스에서 접근할 수 있는 멤버와 접근할 수 없는 멤버로 구분해서 필드, 생성자, 메서드를 설계하는 것이 바람직하다. 객체 생성을 막기 위해 생성자를 호출하지 못 하게 하거나 객체의 특성 데이터를 보호하기 위해 해당 필드에 접근하지 못하도록 막아야 할 때가 있다. 자바는 접근 제한자(**Access Modifier**)를 이용해서 이러한 기능을 구현한다.

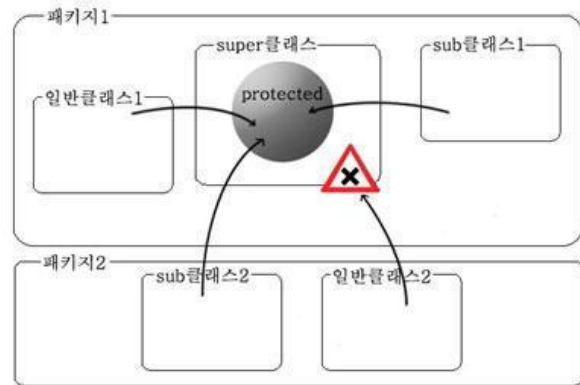


public

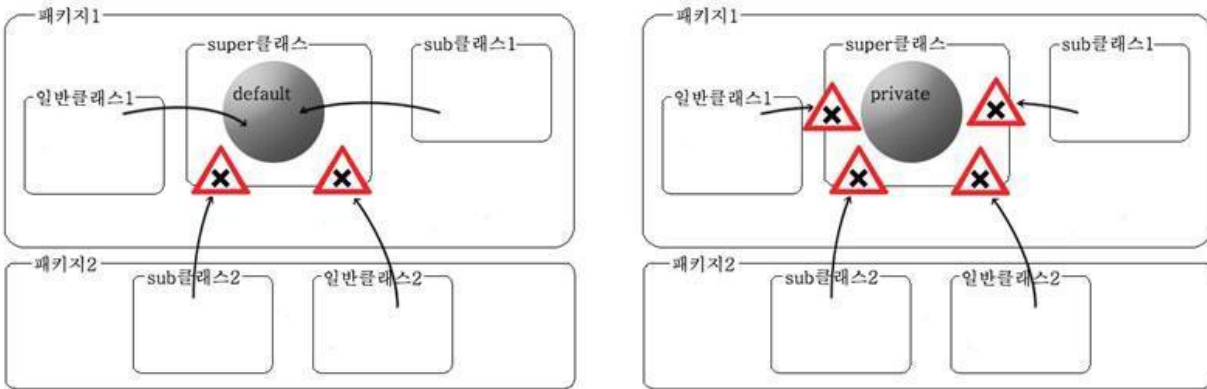


default

protected



private



접근지정자의 사용

클래스 : public, default
 생성자 : public, protected, default, private
 변수, 메서드 : public, protected, default, private

6-6 캡슐화

캡슐화란 객체의 필드, 메서드를 하나로 묶고, 실제 구현 내용을 감추는 것을 말한다. 외부 객체는 객체 내부의 구조를 알지 못하며 객체가 노출해서 제공하는 필드와 메서드만 이용할 수 있다. 필드와 메서드를 캡슐화하여 보호하는 이유는 외부의 잘못된 사용으로 인해 객체가 손상되지 않도록 하는데 있다.

```

01 public class Hello{
02     private int a;
03
04     public int getA(){
05         return a;
06     }
07
08     public void setA(int n){
09         a = n;
10     }
11 }
12 
```

6-7 인자 전달 방식

(1) 값 호출(Call by value)

기본 자료형의 값을 인자로 전달하는 방식. 값을 복사하여 전달

(2) 참조 호출(Call by reference)

메서드 호출시 전달하려는 인자를 참조(객체) 자료형을 사용할 경우를 의미. 주소(reference)를 복사하여 전달

(3) Varargs(Variable Arguments)

자료형이 일치할 때 전달하고자 하는 값의 갯수를 다르게 지정할 수 있다. 전달되는 데이터는 내부적으로 배열로 인식함

6-8 메서드 오버로딩

Overloading(중복정의)이라는 것은 하나의 클래스 내에서 같은 이름을 가지는 메서드가 여러 개 정의되는 것을 말한다. 이것은 컴파일 시 컴파일러에 의해 각 메서드들이 구별되며 기준은 인자가 된다. 인자의 타입 또는 갯수, 배치된 순서가 다를 경우 다른 메서드로 인식함.

[접근제한] [반환형]	[메서드명]	(자료형 인자1, 자료형 인자2, ...){}
--교체 가능--	반드시 동일	----- 반드시 다르게 -----

이런 **Overloading**기법이 필요한 이유는 같은 목적으로 비슷한 동작을 수행하는 메서드들을 모아 이름을 같게 하면 프로그래머로 하여금 다양한 메서드들을 같은 이름으로 일관된 작업을 할 수 있다는 편리함이 있다.

```
01 public class OverloadingMain {
02     public void print(int n) {
03         System.out.println("정수 n = " + n);
04     }
05
06     public void print(double n) {
07         System.out.println("실수 n = " + n);
08     }
09
10     public void print(double n, long a) {
11         System.out.println("실수 n = " + n + ", 정수 a = " + a);
12     }
13
14     public void print(long a, double n) {
15         System.out.println("정수 a = " + a + ", 실수 n = " + n);
16     }
17
18     public static void main(String[] args) {
19         OverloadingMain ot = new OverloadingMain();
20         ot.print(20);
21         ot.print(5.6);
22         ot.print(3.7, 1234L);
23         ot.print(1234L, 5.8);
24     }
25 }
```

6-9 생성자

생성자는 객체가 생성될 때 자동적으로 단 한번 호출되어 필드의 초기화하거나 객체 생성시 반드시 호출되어야 하는 메소드를 호출하는 역할을 한다. 생성자의 구조는 메소드와 비슷하지만 메소드명이 반드시 클래스와 동일해야 하고 리턴타입이 없다.

(1) 생성자의 특징

- 1) **return Type**이 전혀 정의되지 않는다.
- 2) 생성자의 이름이 클래스 명과 같아야 한다.
- 3) 생성자 내부에서 특정 작업을 수행할 수 있고 데이터를 인자에 전달하여 전달받은 데이터를 활용할 수도 있다.

(2) 생성자의 구성

```
[접근제한][생성자명](자료형 인자1, 자료형 인자2,...){
```

```
    수행문1;
    수행문2;
    ...;
}
```

```
public class Hello{
```

```
    //기본 생성자는 생략 가능. 생략할 경우 컴파일러가 자동으로 생성
    public Hello(){}
```

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

(3) 생성자 오버로딩

기본 생성자 하나만을 명시할 수 있는 것이 아니라 다양한 인자를 전달 받아 가공할 수 있는 생성자를 여러개 만들어 사용할 수 있다. 메서드 오버로딩처럼 인자의 타입, 갯수, 배치 순서의 다를 경우 다른 생성자로 인식.

```
01 public class Hello{
02
03     int age;
04
```

```

05  String name;
06
07  public Hello(){
08
09  public Hello(int a){
10      age = a;
11  }
12
13
14  public Hello(String n){
15      name = n
16  }
17
18  public static void main(String[] args){
19      Hello h = new Hello();
20      Hello h2 = new Hello(35);
21      Hello h3 = new Hello("홍길동");
22  }
23  }
24  }

```

6-10 this와 this()

(1) this

객체 내부에서 객체 자신을 칭하고 싶을 때나 아니면 지역변수와 멤버변수를 구별해야 할 때도 있을 것이다. 이럴 때 사용하는 객체 자신을 가리킬 수 있는 유일한 **reference!** 이것이 바로 **this**이다.

```

01  public class Hello{
02      private int a;
03
04      public int getA(){
05          return a;
06      }
07
08      public void setA(int a){
09          this.a = a;
10      }
11  }
12  }

```

(2) this()

이것은 현재 객체의 생성자를 의미하는 것이다. 주의 해야 할 점은 생성자의 첫 행에 정의해야 한다는 것이며 그렇지 않으면 **Compile**시 오류가 발생한다. 다시 말하면 이 **this()**를 이용하여 한 클래스내의 특정 생성자에서 **Overloading**되어 있는 다른 생성자를 호출할 수 있는 것이다. 그렇게 함으로 해서 생성자에서 코딩 내용이 중복되는 부분을 막을 수 있다.

```

01  public class Hello{
02      private int a;
03
04      //아래 생성자가 호출되면 this()를 통해 int 타입의 한 개의 데이터가 인자로
05      //전달되는 생성자를 생성자 내부에서 다시 호출한다.
06      public Hello(){
07          this(10);
08      }
09
10      public Hello(int a){
11          this.a = a;
12      }
13
14  }

```

6-11 static

static 예약어는 멤버메서드나 멤버변수에 정의 할 수 있으며 지역 변수나 클래스에게는 정의 할 수 없다. 멤버메서드나 멤버변수에 **static**이라는 예약어를 정의하면 **static**메서드(클래스메서드)와 **static**변수(클래스변수)라고 불리게 된다. 이유는 멤버변수나 멤버메서드들은 해당 객체가 생성될 때 객체가 생성된 메모리 공간에 같이 존재 하게 되지만 **static**으로 선언된 메서드(멤버함수)나 변수들은 **static**영역(메소드영역)이라는 곳에 유일하게 만들어 지면서 모든 객체(Object)들이 사용 할 수 있도록 공유개념을 가지기 때문이다.

static 초기화

클래스가 실행될 때 가장 먼저 호출되는 영역을 만들고자 한다면 호출 블록에 **static**를 명시한다. 클래스 실행시 단 한번만 실행되어 클래스 실행시 초기화할 내용을 배치하는데 유용하다.

```

01  public class Hello{
02      //main 메서드보다 먼저 실행됨
03      static{
04          System.out.println("static 초기화 블록 실행");
05      }
06      public static void main(String[] args){
07          System.out.println("main 메서드 실행");
08      }
09
10  }

```

7. 객체의 활용

7-1 상속

(1) 클래스의 상속

상속(**inheritance**)은 기존 클래스를 확장해서 새로운 클래스를 만드는 기술을 의미하며 마치 일상 생활에서 부모가 보유하고 있는 재산 중 일부를 자식이 물려받는 것과 같다고 할 수 있다.

(2) 상속 정의

```
class [sub클래스명] extends [super클래스명] {  
    ...;  
}
```

(3) 상속의 중요성

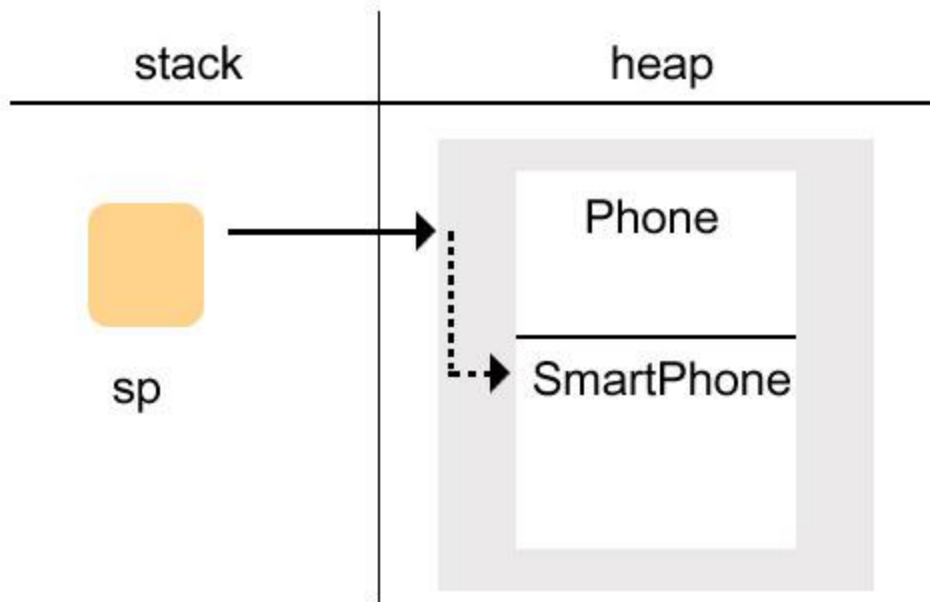
클래스 상속은 객체의 재사용이라는 장점뿐만 아니라 코드의 간결성을 제공해 주는 객체지향적 언어의 장점과 중요한 특징이 된다. 그러므로 잘 정의된 **super**클래스가 있다면 **sub**클래스의 작성이 간편해지고 무엇보다 개발 시간이 단축된다는데 상속의 중요성과 장점을 들 수 있다.

```
01  public class Phone{  
02      protected String number; // 전화번호  
03      protected String color; //색깔  
04      protected int speed; //통신속도  
05  
06      public String getNumber(){  
07          return number;  
08      }  
09  
10  
11      public String getColor(){  
12          return color;  
13      }  
14  
15      public int getSpeed(){  
16          return speed;  
17      }  
18  }  
19  
20  
21  public class SmartPhone extends Phone{  
22      private String os; // OS  
23  
24      public SmartPhone (String number,String color, String os){  
25          this.number = number;  
26          this.color = color;  
27          this.pixel = pixel;  
28      }  
29  }
```

```

30
31     public int getOs(){
32         return os;
33     }
34 }

```



7-2 메서드 오버라이딩

오버라이딩은 [메서드 재정의]라고도 불리며 이는 서로 상속관계로 이루어진 객체들간의 관계에서 비롯된다.

super클래스가 가지는 메서드를 **sub**클래스에서 똑 같은 것을 새롭게 만들게 되면 더 이상 **super**클래스의 이름이 같은 메서드를 호출할 수 없게 된다. 이를 **Overriding**이라 하고 또는 멤버 은폐라고도 한다.

[접근제한]	[반환형]	[메서드명]	(자료형 인자1, 자료형 인자2, ...){}
	반드시 동일	반드시 동일	반드시 동일

*접근제한의 경우 부모클래스의 메서드의 접근제한보다 제한범위가 넓어야 한다. 부모클래스 메서드의 접근제한자가 **protected**이면 **protected**로 명시하거나 **protected**보다 범위가 넓은 **public**으로 표시하지만 부모클래스 메서드의 접근제한이 **public**이면 반드시 **public**으로 명시해야 함

7-3 super와 super()

(1) super

super는 앞서 배운 **this**와 함께 객체를 참조할 수 있는 **reference**변수이다. **this**는 특정 객체

내에서 자기 자신의 객체를 참조할 수 있는 유일한 **reference**변수이다. 그리고 **super**라는 **reference**변수는 현재 객체의 바로 상위인 **super**클래스(부모클래스)를 참조할 수 있는 것이다.

(2) super()

super()는 바로 **super**클래스의 생성자를 의미하는 것이다. 인자가 있다면 인자의 형태와 일치하는 생성자를 의미한다.

7-4 참조자료형 형변환 및 다형성

상속 관계의 있는 객체의 자료형은 자식 클래스의 클래스명을 참조자료형으로 사용하는데 부모 클래스의 자료형으로 변경할 수 있다. 이것을 참조자료형 형변환이라고 한다. 객체가 생성된 후 부모클래스의 자료형을 사용하면 호출할 수 있는 범위가 부모 클래스의 멤버변수, 멤버메서드로 한정된다.

참조자료형 형변환을 통해 다형성(**Polymorphism**)을 구현할 수 있다. 다형성은 한 객체가 여러 모습을 가질 수 있다는 것을 의미한다. 다형성을 자바 언어 측면에서 살펴보면 한 객체가 여러 타입이 될 수 있다는 것을 의미한다.

7-5 final

(1) 상수화

변수에 **final**을 적용 시 상수를 의미한다.

(2) 메서드에 final 표시

메서드에 **final**을 적용 시 오버라이딩으로의 확장이 불가능하다.

(3) 클래스에 final 표시

클래스에 **final**을 적용 시 더 이상의 상속 확장이 불가능하다.

7-6 추상클래스

(1) 추상클래스란?

추상화라는 것은 구체적인 개념으로부터 공통된 부분들만 추려내어 일반화 할 수 있도록 하는 것을 의미한다. 다시 말해서 일반적으로 사용할 수 있는 단계가 아닌 아직 미완성(未完成)적 개념인 것이다.

추상클래스는 일반클래스와 달리 미완성되어 있는 클래스이기 때문에 객체 생성이 불가능하고 상속관계를 맺은 자식 클래스가 객체 생성되어야 사용이 가능하다.

```
public abstract class Parent{  
  
}
```

(2) 추상메서드

추상클래스는 일반적으로 한 개이상의 추상메서드를 갖는다. 추상메서드는 일반메서드와 다르게 메서드가 수행할 코드를 갖고 있지 않다. 따라서 추상메서드는 상속시 자식 클래스에 반드시 구현되어야 자식클래스가 객체 생성될 수 있다.

```
01 public abstract class Parent{
02     public abstract void drive();
03 }
04
05 public class Child extends Parent{
06     //추상메서드를 구현하지 않으면 Child 객체 생성 불가능
07     public void drive(){
08         System.out.println("달리다");
09     }
10 }
11 }
```

(3) 추상 클래스의 상속 관계

추상 클래스들간에도 상속이 가능하다. 일반 클래스들간의 상속과 유사하지만 추상 클래스들간의 상속에서는 상속 받은 추상 메서드들을 꼭 재정의할 필요는 없다. 그냥 상속만 받아두고 있다가 언젠가 일반 클래스와 상속관계가 이루어 질 때가 있을 것이다. 이때 재정의 하지 못했던 상속 받은 추상 메서드들을 모두 일반 클래스 내에서 재정의해도 되기 때문이다.

7-7 인터페이스

클래스가 멤버필드(변수,상수)와 실행 가능한 메서드를 구성요소로 한다면 인터페이스는 상수와 추상메서드를 구성요소로 한다. 상수는 **static**한 상수이기 때문에 객체 생성 없이 호출이 가능하지만 추상메서드는 일반클래스에 구현되어야만 사용이 가능하다.

```
[접근제한] interface [인터페이스명] {
    상수;
    추상메서드;
}
```

(1) 상수

```
01 public interface Number{
02     public static final int NUM = 10;
03 }
04
05 public class Hello{
06     public static void main(String[] args){
07         System.out.println(Number.NUM);
08     }
09 }
```

```
10 }
```

(2) 추상메서드

```
01 public interface Inter{
02     public abstract void product();
03 }
04
05 public class Item implements Inter{
06     //추상메서드를 반드시 구현해야 함
07     public void product(){
08         System.out.println("제 품을 생산하다");
09     }
10 }
11 }
```

(3) 인터페이스의 장점

인터페이스를 이용하면 정의해야 하는 메소드(프로그램기능)을 표준화하고 강제할 수 있다. 또한 메소드화 시켜야 하는 기능을 분류해야 하는 고민 없이 구현만 하면 되므로 개발시간을 단축시킬 수 있다. 일반 클래스 상속을 이용해서 자식 클래스들의 관계를 맺는 것보다 간편하게 관계를 맺을 수 있다

8 내부클래스

8-1 내부클래스

특정 클래스 내에 또 다른 클래스가 정의되는 것을 의미한다.
이런 내부 클래스가 필요한 이유는 지금까지 작업해 왔던 클래스들과는 다르게 독립적이지는 않지만 하나의 멤버처럼 사용할 수 있는 특징이 있다.

내부 클래스의 종류

종류	설명
Member	멤버 변수나 멤버 메서드들과 같이 클래스가 정의된 경우에 사용한다.
Local	특정한 메서드 내에 클래스가 정의된 경우에 사용
Static	static 변수(클래스 변수)와 같이 클래스가 static 으로 선언된 경우에 사용
Anonymous	참조할 수 있는 이름이 없는 경우에 사용

(1) 멤버내부클래스

객체를 생성해야만 사용할 수 있는 멤버들과 같은 위치에 정의되는 클래스를 말한다. 즉 내부 클래스를 생성하려면 외부 클래스의 객체를 생성한 후에 생성할 수 있다.

```
public class Outer {  
    ...  
    class Inner {  
  
    }  
    ...  
}
```

(2) 로컬내부클래스

Local 내부 클래스는 특정 메서드 안에서 정의되는 클래스를 말한다. 다시 말해서 특정 메서드 안에서 선언되는 지역변수와 같은 것이다. 메서드가 호출될 때 생성할 수 있으며 메서드의 수행력이 끝나면 지역변수와 같이 자동 소멸된다.

```
public class Outer {  
    ...  
    public void methodA() { // 멤버 메서드  
        class Inner {  
  
        }  
    }  
    ...  
}
```

(3) static 내부클래스

static 내부 클래스로 어쩔 수 없이 정의하는 경우가 있는데 그것은 바로 내부 클래스 안에 **static**변수를 가지고 있다면 어쩔 수 없이 해당 내부 클래스는 **static**으로 선언하여야 한다.

```
public class Outer {  
    ...  
    static class Inner {  
  
    }  
    ...  
}
```

(4) 익명내부클래스

익명이란? 이름이 없는 것을 의미한다. 이것을 자바의 프로그램적으로 해석하면 정의된 클래스의 이름이 없다는 것이 된다.

```

class Outer {
    ...
    Inner inner = new Inner(){
        ...;
    };
    public void methodA() { // 멤버 메서드
        new Inner() {
            ...;
        };
    }
    ...
}

```

9. enum

(1) enum

열거 형은 상수를 가지고 생성되는 객체들을 한 곳에 모아둔 하나의 묶음이다.
 자바에서 열거 타입이 있기 전에는 코드화된 정수나 문자를 이용하여 해당 값을 표현했음

```

[접근제한]enum [열거형 이름]{
    상수1, 상수2, ..., 상수n
}

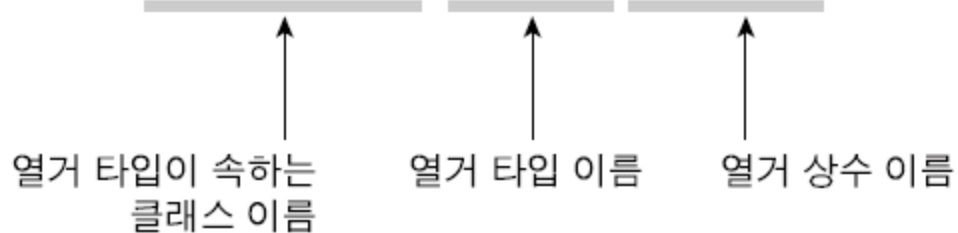
enum Season{
    SPRING, SUMMER, FALL, WINTER
}

```

(2) 클래스에 종속된 열거 타입

클래스/인터페이스 안에 선언된 열거 타입은 그 클래스/인터페이스에 종속됨

season = ClothingInfo.Season.SPRING;



(3) values 메소드와 valueOf 메소드

- 열거 타입은 컴파일하고 나면 내부적으로 클래스가 됨
 -> 그 클래스에는 values와 valueOf라는 정적 메소드가 자동으로 추가됨

- **values** 메소드 : 모든 열거 상수를 리턴하는 메소드
- **valueOf** 메소드 : 주어진 문자열에 해당하는 열거 상수를 리턴하는 메소드

(4) 열거 상수를 다른 값과 연관짓기

```

01  enum Season{
02      SPRING("봄"), SUMMER("여름"), FALL("가을"), WINTER("겨울");
03      final private String name;
04
05      //열거상수 호출시 지정된 값이 전달되어 name변수에 저장됨
06      Season(String name){
07          this.name = name;
08      }
09
10      //열거 상수에 연관된 값을 리턴하는 메서드
11      String value(){
12          return name;
13      }
14  }

```

10. 자바 기본 클래스의 이해

10-1 String

- 문자열 객체
- 문자열 수정 불가능(불변적 특징)
- + 연산자를 이용하여 새로운 문자열 객체 생성

```

암시적 객체 생성 String s1 = "Hello";
명시적 객체 생성 String s2 = new String("Hello")

```

10-2 StringBuffer

- 문자열 버퍼 객체
- 문자열 추가 변경 가능
- **append()** 메서드를 이용하여 문자(열) 추가

```

StringBuffer sb = new StringBuffer("Hello");
sb.append(" World!!");

```

10-3 StringTokenizer

- 문자열 분리 객체
- nextToken() 메서드를 이용하여 문자(열) 분리

```
StringTokenizer st = new StringTokenizer("2014/12/17", "/");
while(st.hasMoreTokens()){
    String token = st.nextToken();
    System.out.println(token);
}
```

10-4 Date

```
Date now = new Date();
System.out.println(now);
```

10-5 Calendar

```
Calendar calendar = Calendar.getInstance();
int year = calendar.get(Calendar.YEAR);
int month = calendar.get(Calendar.MONTH) + 1;
int date = calendar.get(Calendar.DAY_OF_MONTH);
```

10-6 Math

```
01 public class MathTest {
02     public static void main(String[] args) {
03         int a = Math.abs(-10);
04         System.out.println(a);
05
06         double b = Math.ceil(3.3);
07         System.out.println(b);
08
09         double c = Math.floor(3.7);
10         System.out.println(c);
11
12         int d = Math.round(3.7f);
13         System.out.println(d);
14
15         int e = Math.max(3, 5);
16         System.out.println(e);
17
18         int f = Math.min(4, 7);
19
20     }
```

```

21         System.out.println(f);
22     }
23 }

```

10-7 Random

난수 발생 기능만을 제공하는 클래스

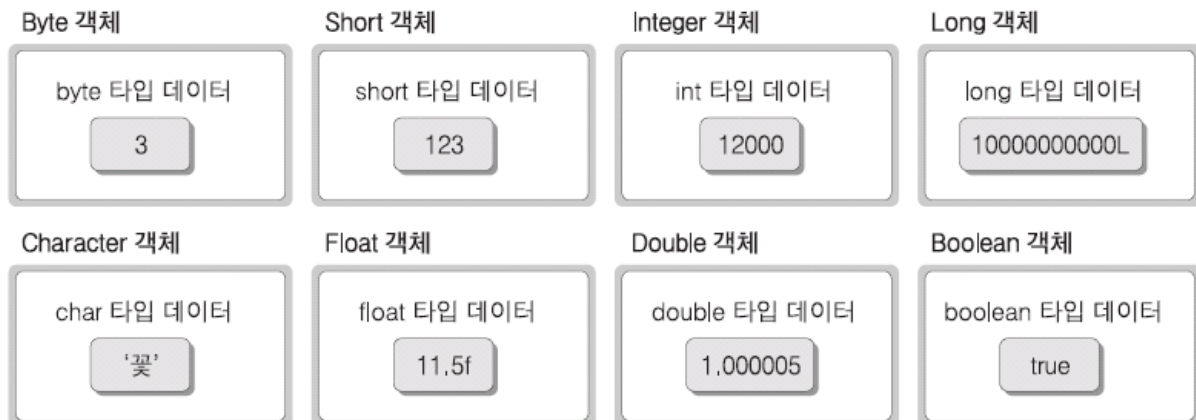
10-8 Wrapper 클래스

(1) Wrapper 클래스

프리미티브 타입을 객체로 표현하는 데 사용되는 다음 클래스들의 통칭

기본 자료형	클래스 이름
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double

프리미티브 타입을 데이터를 감싸는 역할을 하는 Wrapper 클래스



(2) 자동 Boxing과 자동 Unboxing

• 자동 Boxing

래퍼 객체를 써야 할 자리에 프리미티브 값을 썼을 때 일어나는 래퍼 객체로의 자동 변환

• 자동 Unboxing

프리미티브 값을 써야 할 자리에 래퍼 객체를 썼을 때 일어나는 프리미티브 값으로의 자동 변환

11. 예외처리

11-1 예외와 예외처리

(1) 예외와 예외처리

예상하지 못한 일들을 ‘예외’라 하고 이를 대비하고 준비하는 것이 바로 ‘예외처리’다.

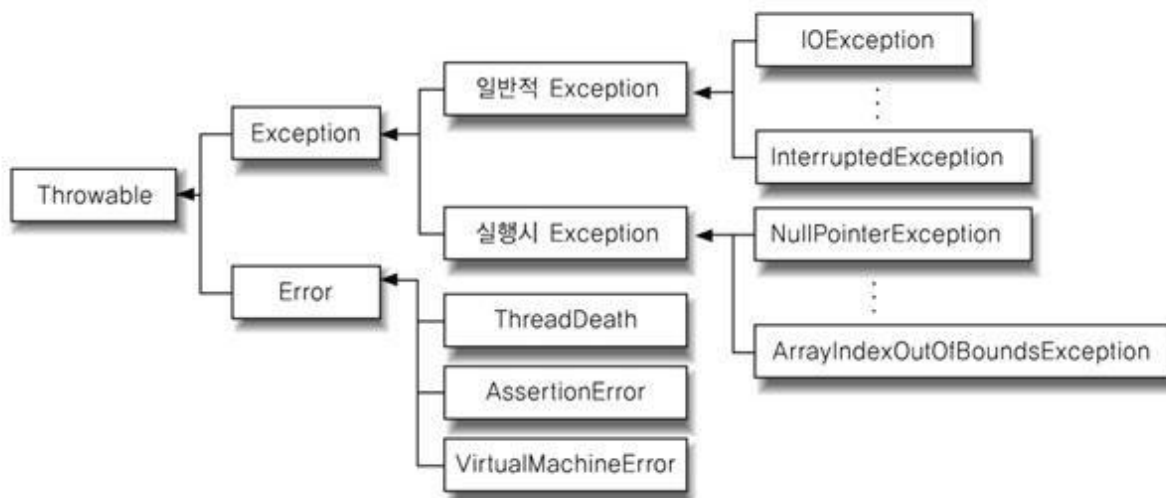
(2) 예외처리에 대한 필요성과 이해

자바에서 프로그램의 실행하는 도중에 예외가 발생하면 발생한 그 시점에서 프로그램이 바로 종료가 된다. 때에 따라서는 예외가 발생 했을 때 프로그램을 종료시키는 것이 바른 판단일 수도 있다. 하지만 가벼운 예외이거나 예상을 하고 있었던 예외라면 프로그램을 종료시키는 것이 조금은 가혹(?)하다고 느껴진다. 그래서 ‘예외처리’라는 수단(**mechanism**)이 제안되었고 예외 처리를 통해 우선 프로그램의 비 정상적인 종료를 막고 발생한 예외에 대한 처리로 정상적인 프로그램을 계속 진행할 수 있도록 하는 것이 예외처리의 필요성이라 할 수 있다.

(3) 오류의 구분

구분	설명
예외(Exception)	가벼운 오류이며 프로그램적으로 처리
오류(Error)	치명적인 오류이며 JVM에 의존하여 처리

(4) 예외의 종류



11-2 예외처리

(1) 예외처리

```
try{
    // 예외가 발생 가능한 문장들;
}catch(예상되는_예외객체 변수명){
    // 해당 예외가 발생했을 시 수행할 문장들;
}
```

try ~ catch문에서의 주의 사항

```
try{
    System.out.println((i+1)+"번째");
    System.out.println("var["+i+"] : "+var[i]); <-- 예외가 발생
    System.out.println("~~~~~"); //예외 발생시 수행하지 못 함
}catch(ArrayIndexOutOfBoundsException ae){
    System.out.println("배열을 넘었습니다.");
}
```

(2) 다중 catch문

다중 **catch**문은 하나의 **try**문 내에 여러 개의 예외가 발생 가능할 때 사용한다. 구성은 다음과 같다.

```
try{
    // 예외가 발생 가능한 문장들;
}catch(예상되는_예외객체1 변수명){
    // 해당 예외가 발생했을 시 수행할 문장들;
}catch(예상되는_예외객체2 변수명){
    // 해당 예외가 발생했을 시 수행할 문장들;
}catch(예상되는_예외객체3 변수명){
    // 해당 예외가 발생했을 시 수행할 문장들;
}
```

다중 **catch**문의 주의 사항

일반적 예외(**Exception**)에서 가장 상위(**parent**) 클래스가 **Exception**이다. 그러므로 가장 아래쪽에 정의 해야 한다. 이렇게 하는 이유는 예외는 상위(**parent**) 클래스가 모든 예외를 가지고 있으므로 가장 위에 정의를 하게 되면 모든 예외를 처리하게 되므로 두 번째 **catch**문부터는 절대로 비교 수행할 수 없게 된다.

11-3 throws예약어

예외를 처리하기 보다는 발생한 예외 객체를 양도하는 것이다. 즉, 현재 메서드에서 예외처리를 하기가 조금 어려운 상태일 때 현재 영역을 호출해준 곳으로 발생한 예외 객체를 대신 처리해 달라며 양도 하는 것이다. 사용법은 다음의 구성과 같이 **throws**라는 예약어를 통해 메서드를 선언하는 것이다.

[접근제한] [반환형] [메서드명](인자1, ...인자n) throws 예외클래스1,...예외클래스n{

```
public void setData(String n) throws NumberFormatException{
    if(n.length() >= 1){
        String str = n.substring(0,1);
        printData(str);
    }
}
```

11-4 throw 예외의 인위적인 발생

사용자(프로그래머)가 의도적으로 예외를 발생시킬 때 **throw**문을 사용해 인위적으로 예외를 발생시킨다.

throw new 예외클래스(전달인자);

```
public class ThrowEx1 {
    public void methodA(String[] n)throws Exception{
        if(n.length > 0){
            for(String s : n)
                System.out.println(s);
        }else
            throw new Exception("배열에 요소가 없습니다.");
    }
    public static void main(String[] args) {
        ThrowEx1 te = new ThrowEx1();
        try{
            te.methodA(args);
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

11-5 finally의 필요성

예외가 발생하든 발행하지 않든 무조건 수행하는 부분이 바로 **finally**영역이다. 이것은 뒤에서 **Database**처리나 **File**처리를 한다면 꼭 필요한 부분이다. 이유는 **Database**를 열었다거나 또는 **File**을 열었다면 꼭 닫아주고 프로그램이 종료되어야 하기 때문이다.

```
try{
    // 예외가 발생 가능한 문장들;
```

```
} catch(예상되는_예외객체1 변수명){  
    // 해당 예외가 발생했을 시 수행할 문장들;  
}  
finally{  
    // 예외발생 여부와 상관없이 수행할 문장들;  
}
```

11-6 사용자 정의 예외

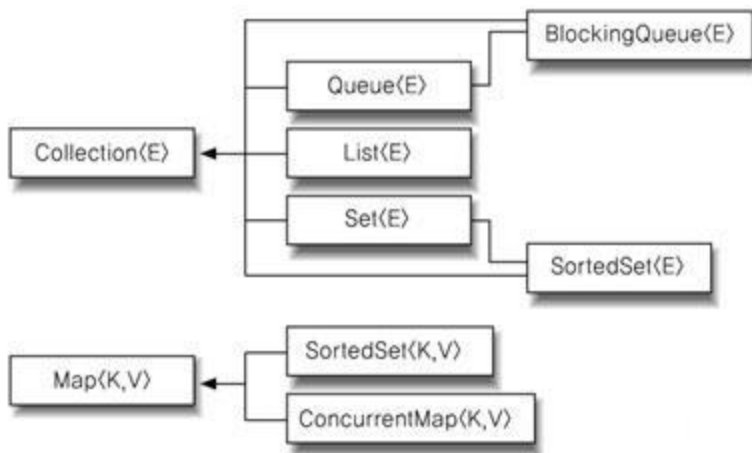
사용자 정의 **Exception**이 필요한 이유는 표준예외가 발생할 때 예외에 대한 정보를 변경하거나 정보를 수정하고자 한다면 사용자가 직접 작성하여 보안된 예외를 발생시켜 원하는 결과를 얻는데 있다.

```
public class UserException extends Exception{  
  
    private int port = 772;  
    public UserException(String msg){  
        super(msg);  
    }  
    public UserException(String msg, int port){  
        super(msg);  
        this.port = port;  
    }  
    public int getPort(){  
        return port;  
    }  
}
```

12. 컬렉션

12-1 컬렉션

자바에서 얘기하는 **Java Collections Framework**는 객체들을 한 곳에 모아 관리하고 또 그것을 편하게 사용하기 위해 제공되는 환경이다. 여기에는 다음과 같이 구조를 이루고 있다.

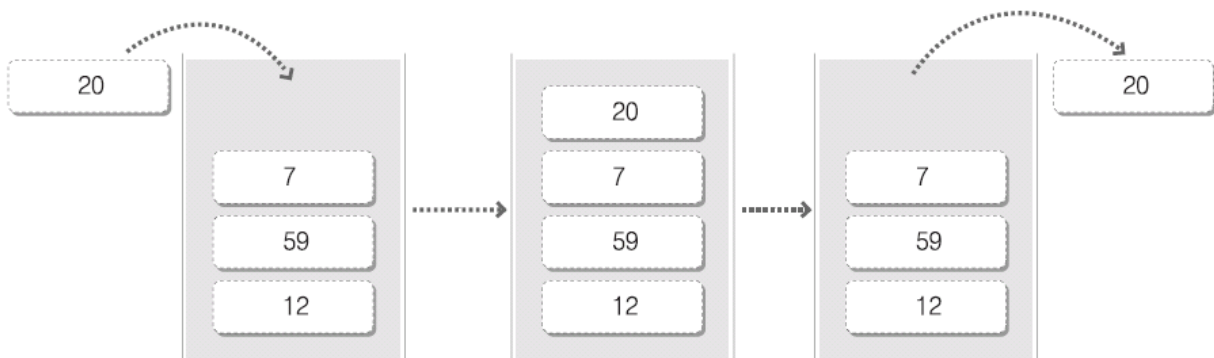


12-1 List

리스트(list)는 데이터를 1차원으로 늘어놓은 형태의 자료구조를 말한다. 1차원 형태로 데이터를 저장하는 방법으로는 배열도 있지만, 리스트는 배열과는 달리 데이터의 검색과 추가, 삭제가 가능하다.

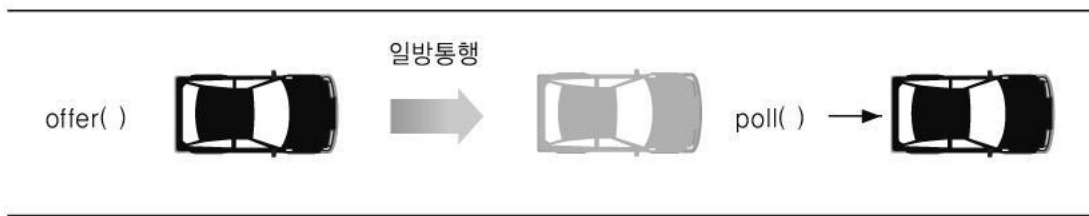
12-2 Stack

스택(stack)은 배낭에 물건을 넣을 때처럼 제일 마지막에 넣은 데이터부터 순서대로 꺼낼 수 있는 자료구조를 말한다. 이렇게 가장 나중에 넣었던 데이터를 가장 먼저 꺼내는 입출력 방식을 LIFO(Last-In First-Out)방식이라고 한다.



12-2 Queue

큐(queue)는 매표소에 줄을 서서 기다리는 사람들처럼 들어온 순서대로 데이터를 꺼낼 수 있는 자료구조. 가장 처음에 넣었던 데이터를 가장 먼저 꺼내는 입출력 방식을 FIFO(First-In First-Out)방식이라고 한다.



[그림 7-23] Queue의 도식화

12-3 Set

Set내에 저장되는 객체들은 특별한 기준에 맞춰서 정렬되지 않는다. 그리고 저장되는 객체들간의 중복된 요소가 발생하지 못하도록 내부적으로 관리되고 있다.

12-4 Map

Key와 Value를 매핑하는 객체이다. 여기에 사용되는 Key는 절대 중복될 수 없으며 각 Key는 1개의 Value만 매핑할 수 있다. 정렬의 기준이 없으며 이는 마치 각 Value에 열쇠 고리를 달아서 큰 주머니에 넣어두고 오로지 Key로 각 Value를 참조 할 수 있도록 해둔 구조라 할 수 있다.

12-5 제네릭

Generics는 컬렉션(자료구조) 즉, 쉽게 말해서 객체들을 저장(수집)하는 구조적인 성격을 보강하기 위해 제공되는 것이다. 특정 컬렉션(자료구조)에 원하는 객체 타입을 명시하여 실행하기 전에 컴파일단계에서 지정된 객체가 아니면 절대 저장이 불가능하게 할 수 있다.

1) 제네릭 클래스 선언

[접근제한자] **class** [클래스이름] <Type1, Type2, ...> { }

클래스를 선언할 때 클래스 이름 옆에 <>를 이용하여 타입 변수를 지정해주면 된다. 제네릭 타입 변수는 식별자 명명 규칙에 따라 어느 것이든 가능하지만 일반적으로 T를 많이 사용한다. 이후 클래스 내부에 변수나 메소드를 선언할 때 일반 데이터 타입과 동일하게 사용하면 된다. 이후 클래스를 생성할 때 타입 변수를 특정 타입으로 지정해주면 해당 타입으로 메모리에 할당하여 사용 가능하다.

```
public class GenericEx<T> { T object;

void setObject(T object) { this.object = object; } T getObject() { return object; }
}
```

```
GenericEx<String> ex = new GenericEx<String>();
```

만약 두 개 이상의 타입으로 제네릭 타입 변수를 사용하고 싶다면 콤마로 구분하여 다른 이름으로 타입 변수를 지정해주면 된다.

2) 제네릭 메소드

메소드에서 사용할 타입을 제네릭을 이용하여 선언할 수 있다.

```
[접근제한자]<Type1, Type2, ...> [리턴타입] [메소드이름](매개변수, ...) { ... }
```

제네릭 메소드를 호출할 때에는 타입 변수를 명시적으로 지정할 수도 있고 컴파일러가 매개값의 타입을 보고 추정하도록 할 수도 있다.

3) 제네릭 제한하기

일반적인 방법으로 제네릭을 이용했을 경우 타입에 대한 제한이 없다. 하지만 제네릭 타입 변수에 **extends** 키워드를 사용하면 타입의 종류를 제한할 수 있다.

```
class Point { int x; int y; }  
class Triangle<T extends Point> { T pos1, pos2, pos3; }
```

extends 뒤에 명시된 타입의 자손들만 타입 변수에 대입할 수 있게 된다.

4) 와일드카드

타입 변수를 매개변수나 리턴 타입으로 사용할 때 구체적인 타입 대신 와일드 카드를 사용할 수 있다. 타입 변수를 대치할 수 있는 구체적인 타입으로 제한을 두지 않거나 **extends** 키워드를 이용해 해당 타입과 자손 타입들만 가능하도록 제한하거나 **super** 키워드를 이용해 해당 타입과 부모 타입들만 가능하도록 제한하는 방법이 있다.

- 와일드카드 이용 3가지 방법

```
<?> : 제한 없음  
<? extends 상위타입> :상위 클래스 제한  
<? super 하위 타입> :하위 클래스 제한
```

5) 제네릭 상속

제네릭 클래스도 다른 클래스와 마찬가지로 부모 클래스가 될 수 있다. 부모 클래스의 제네릭 타입을 상속해서 자식 클래스에서도 제네릭 타입을 상속할 수 있다.

```
public class ChildProduct<T, K> extends Product<T, K> { ... }
```

자식 제네릭 클래스는 추가적인 타입 변수를 가질 수 있다. `public class ChildProduct<T, K, S> extends Product<T, K> { ... }`

13. GUI 프로그래밍

13-1 GUI 프로그래밍이란

GUI는 과거에 사용하였던 DOS(CUI방식)와 같은 방식의 텍스트 기반 운영체제가 아닌 그래픽을 이용하여 사용자와 프로그램 간의 상호작용을 할 수 있도록 해주는 인터페이스를 의미한다.

13-2 AWT

(1) AWT

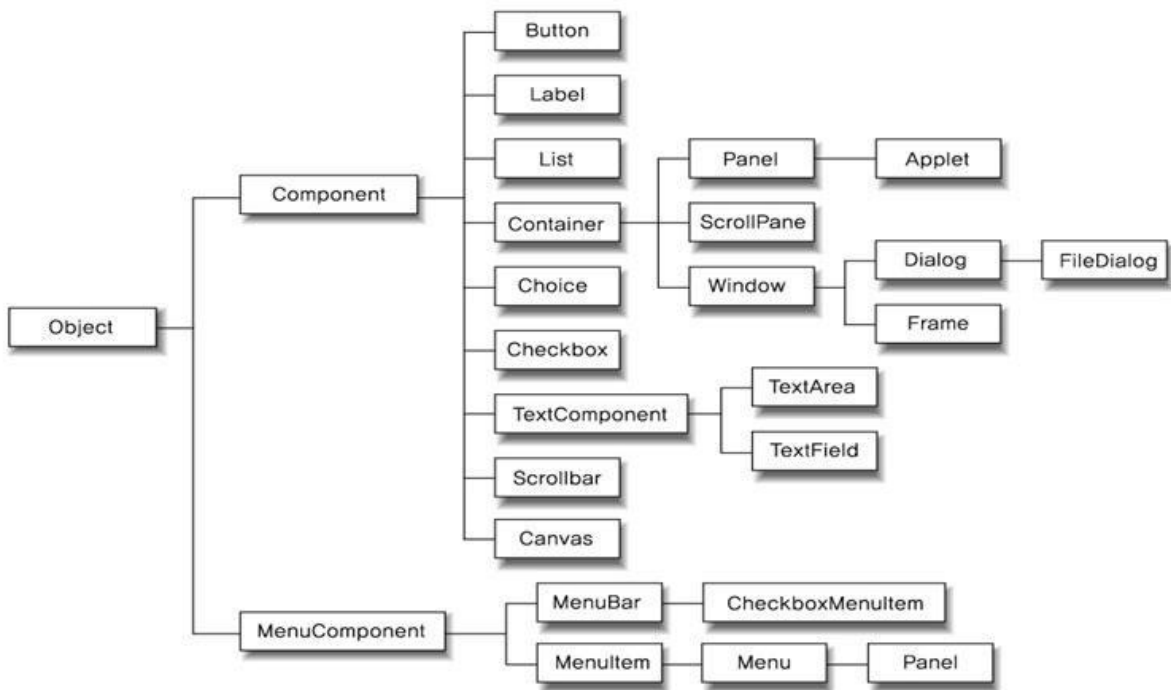
AWT(Abstract Window Toolkit)는 GUI 프로그래밍을 제작하기 위해 자바에서 제공하는 라이브러리를 모아놓은 것이다.

AWT는 모든 GUI 프로그램에 사용되는 컴포넌트 및 툴킷을 제공하고 있으며 향후에는 JFC와 같은 스윙(Swing) 및 Java2D의 모태가 되는 개념이다.

AWT는 운영체제에 구애받지 않고 쓸 수 있도록 운영체제의 것을 그대로 사용하지 않고 공통적이고 기본적인 컴포넌트들을 추상화시켜 제공한다.

실행되는 운영체제에 따라 다르게 보이거나 동작 방식에 차이가 있을 수 있다.

(2) java.awt 패키지



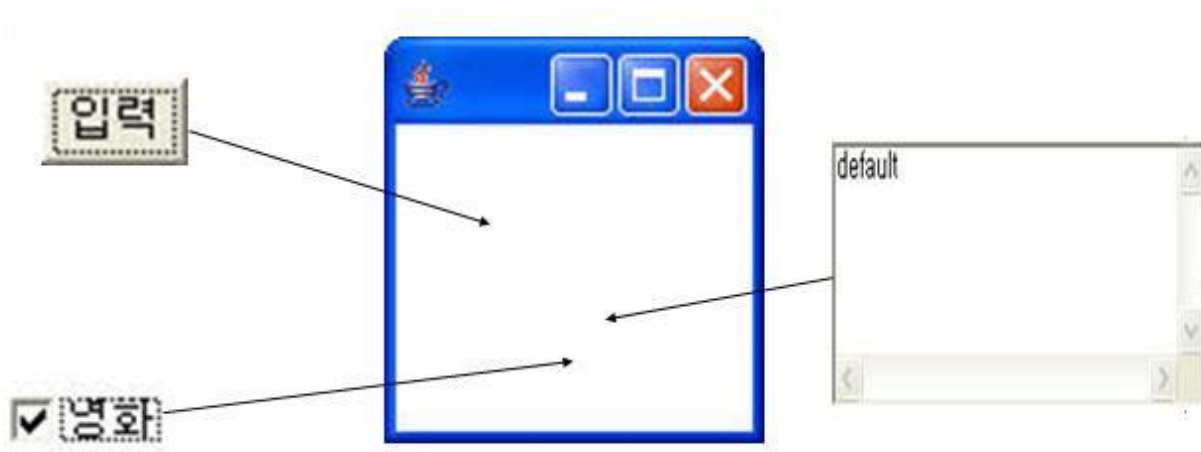
(3) Container

자신의 영역에 컴포넌트를 포함시키고 관리하는 역할을 하며 컨테이너가 다른 컨테이너를 포함할 수도 있다.

컴포넌트도 또한 컨테이너에 부착시키지 않으면 독자적으로 화면에 출력될 수가 없고 반드시 컨테이너에 부착을 시켜야만 화면에 출력이 될 수 있다.

컨테이너의 종류에는 **Frame, Window, Panel, Applet, Dialog, FileDialog, ScrollPane**이 있다.

컨테이너에 컴포넌트를 부착시키기 위해 **add()**메서드를 사용한다.



(4) Frame

Window 클래스의 하위 클래스로 일반적인 응용프로그램에서 윈도우를 생성하기 위해

사용되는 클래스이다.

Frame 클래스의 상위 클래스인 **Window** 클래스는 타이틀, 메뉴 등이 지원되지 않기 때문에 일반적으로 사용하지 않고 **Frame** 클래스를 사용한다.

Frame 클래스는 기본적으로 경계선(**Border**), 타이틀, 메뉴, 시스템상자(최소화, 최대화, 종료 버튼) 등의 기능을 제공한다.

Frame은 다른 윈도우에 속해 있지 않은 윈도우로 최상위 레벨 윈도우라 한다.

setSize(), **setBounds()**메서드 등을 이용해서 **Window**의 크기를 설정한 후 **setVisible()**, **show()**메서드를 통해서 화면에 출력시킬 수 있다.

(5) Component

모든 컴포넌트들의 **super** 클래스로서 **GUI** 프로그램을 구성하는 구성단위로 각 컴포넌트들에서 공통으로 사용되어지는 메서드들을 가지고 있다.

(6) LayoutManager

컨테이너는 자기 자신에 컴포넌트를 붙일 때 어디에, 어떤방식으로 배치하여 붙일것인가를 이미 결정하고 있다.

즉, 미리 정해진 레이아웃에 따라 컴포넌트들을 자동으로 배치하는 기능을 가지고 있는 객체를 컨테이너들은 가지고 있는데 이것을 배치관리자(**LayoutManager**)라 한다.

자바에서 사용하는 배치관리자는 **FlowLayout**, **BorderLayout**, **GridLayout**, **GridBagLayout**, **CardLayout**의 5가지가 있다.

배치관리자는 각자 다른 방식으로 배치기능을 가지고 있으며 컨테이너는 기본적으로 하나의 배치관리자를 가지고 있다.

사용자가 임의로 배치관리자는 다시 설정할 수 있으며 배치관리자를 제거하고 수동으로 좌표를 이용해서 배치할 수도 있다.

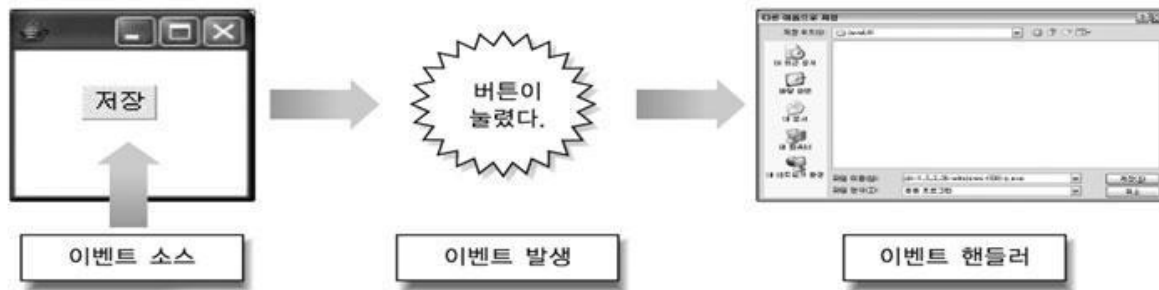
(7) 이벤트

1) 이벤트

이벤트(**Event**)라는 것은 윈도우 프로그래밍에서 어떤 특정한 행동이 발생한 그 자체를 의미한다.

예를 들어 메뉴를 선택했다는가, 아니면 마우스를 클릭하거나, 윈도우의 크기를 조절하거나 등의 행위를 뜻하는 것이다.

이런한 방식의 프로그래밍을 이벤트 중심의 프로그래밍이라고 하는데 윈도우 프로그래밍에서 중요한 개념중에 하나이다.



- 이벤트 소스(Event Source)

이벤트 소스는 이벤트가 발생하는 컴포넌트를 말한다. 즉, 버튼, 체크박스, 리스트, 프레임, 마우스 등과 같은 컴포넌트들이 이벤트 소스이다.

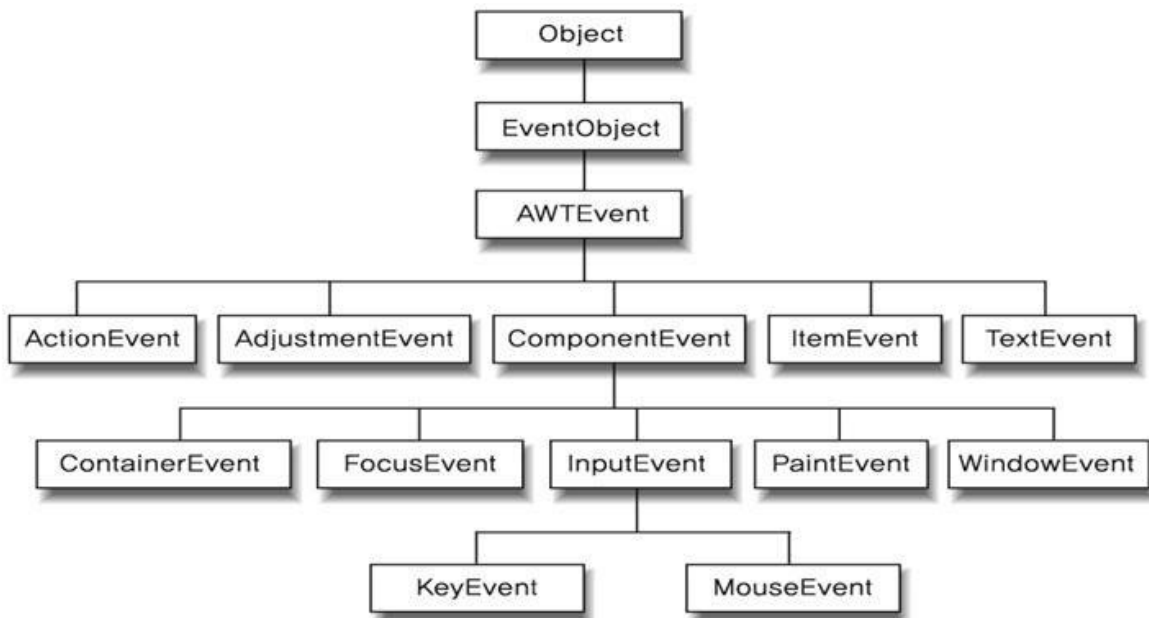
- 이벤트 리스너(Event Listener)

이벤트 소스에서 이벤트가 발생하는지를 검사하고 있다가 이벤트가 발생이 되면 실제적으로 이벤트를 처리할 수 있도록 만든 인터페이스이다.

- 이벤트 핸들러(Event Handler)

이벤트 리스너에 전달된 이벤트를 실제로 처리할 수 있도록 이벤트 리스너에 포함되어있는 메서드로 발생된 이벤트 객체를 받아와서 실제적으로 처리해주는 기능을 가지고 있다.

(2) 이벤트 클래스 구조



(3) 이벤트의 종류와 설명

이벤트	설명
ActionEvent	버튼, 리스트, 메뉴 등의 컴포넌트가 눌리거나 선택이 되었을 때 발생하는 이벤트
AdjustmentEvent	스크롤바와 같은 조정 가능한 컴포넌트에서 조정이 일어나면 발생하는 이벤트
ComponetEvent	컴포넌트의 모습이나 이동, 크기가 변화될 때 발생하는 이벤트
ItemEvent	리스트와 같은 선택항목이 있는 컴포넌트에서 선택항목이 선택될 때 발생하는 이벤트
TextEvent	텍스트 컴포넌트에서 값이 입력될 때 발생하는 이벤트
ContainerEvent	컨테이너에 컴포넌트가 추가되거나 제거될 때 발생하는 이벤트
FocusEvent	컴포넌트에 초점(Focus)이 들어 올 때 발생하는 이벤트
PaintEvent	컴포넌트가 그려져야 할 때 발생하는 이벤트
WindowEvent	윈도우가 활성화되거나 비활성화 될 때, 최소, 최대, 종료 될 때 발생하는 이벤트
KeyEvent	키보드로부터 입력이 될 때 발생하는 이벤트
MouseEvent	마우스가 눌러지거나 움직일 때, 마우스 커서가 컴포넌트 영역에 들어가거나 벗어날 때 발생하는 이벤트

(4) 리스너

ActionListener
ActionEvent를 처리하는 이벤트 리스너가 ActionListener이다.
ItemListener
ItemEvent를 처리하는 이벤트 리스너가 ItemListener이다.
TextListener
TextEvent를 처리하는 이벤트 리스너가 TextListener이다.
KeyListener
KeyEvent를 처리하는 이벤트 리스너가 KeyListener이다.
MouseListener
마우스와 관련 있는 이벤트 중 MouseEvent를 처리하는 이벤트 리스너가 MouseListener이다.
MouseMotionListener
마우스와 관련 있는 이벤트 중 MouseEvent를 처리하는 이벤트 리스너가 MouseMotionListener이다.
WindowListener
WindowEvent를 처리하는 이벤트 리스너가 WindowListener이다.

(5) Adapter 클래스의 종류

이벤트	이벤트 리스너	이벤트 어댑터
ComponentEvent	ComponentListener	ComponentAdapter
ContainerEvent	ContainerListener	ContainerAdapter
FocusEvent	FocusListener	FocusAdapter
KeyEvent	KeyListener	KeyAdapter
MouseEvent	MouseListener	MouseAdapter
MouseEvent	MouseMotionListener	MouseMotionAdapter
WindowEvent	WindowListener	WindowAdapter

13-3 그래픽

(1) Graphics 클래스

Graphics 클래스는 그래픽 작업을 할 수 있도록 기능들을 추상화시킨 클래스로 그림을 그릴 수 있는 각종 메서드를 지원하고 있다.

API메서드 중에 fill이 붙어 있는 것들은 채우기 기능을 가지고 있는 메서드들이다.

(2) Color 클래스

그래픽 컨텍스트에 색상을 설정하기 위해 사용하는 클래스이다.

Color 클래스 객체를 생성할 때 사용할 색상을 지정하여 그래픽 컨텍스트에 설정하면 그 이후의 모든 색상에 적용하여 사용할 수 있다.

(3) Font 클래스

그래픽 컨텍스트에 글꼴을 설정하기 위해 사용하는 클래스이다.

Font 클래스 객체를 생성할 때 사용할 글꼴의 속성을 지정하여 그래픽 컨텍스트에 설정하면 그 이후의 모든 글자에 적용하여 사용할 수 있다.

13-4 Swing

(1) Swing

자바의 JFC(Java Foundation Class)는 GUI 프로그래밍에 필요한 각종 툴킷을 모아놓은 것으로 현재는 GUI의 기능들을 구현할 수 있는 스윙, 2D, Drag&Drop 등을 지원한다.

스윙을 사용하는 방법은 AWT와 거의 유사하나 AWT보다는 많은 컴포넌트 및 기능을 지원하고 있다.

스윙은 AWT와 달리 자바 프로그래밍으로 자체적인 제작된 컴포넌트이므로 플랫폼에 관계없이 모양이 동일하게 사용할 수 있다.

- 룩앤필(Look & Feel)

스윙에서 가장 획기적으로 바뀐 것 중에 하나가 컴포넌트의 화려함이다.

이러한 외관(Look & Feel)을 프로그램을 실행하는 도중에 여러가지 형태로 바꾸어 사용할 수 있는 기능을 제공한다.

스윙은 순수한 자바로만 만들어졌기 때문에 어떤 플랫폼에서라도 동일한 룩앤필(Look & Feel)을 유지할 수가 있다.

- 경량의 컴포넌트

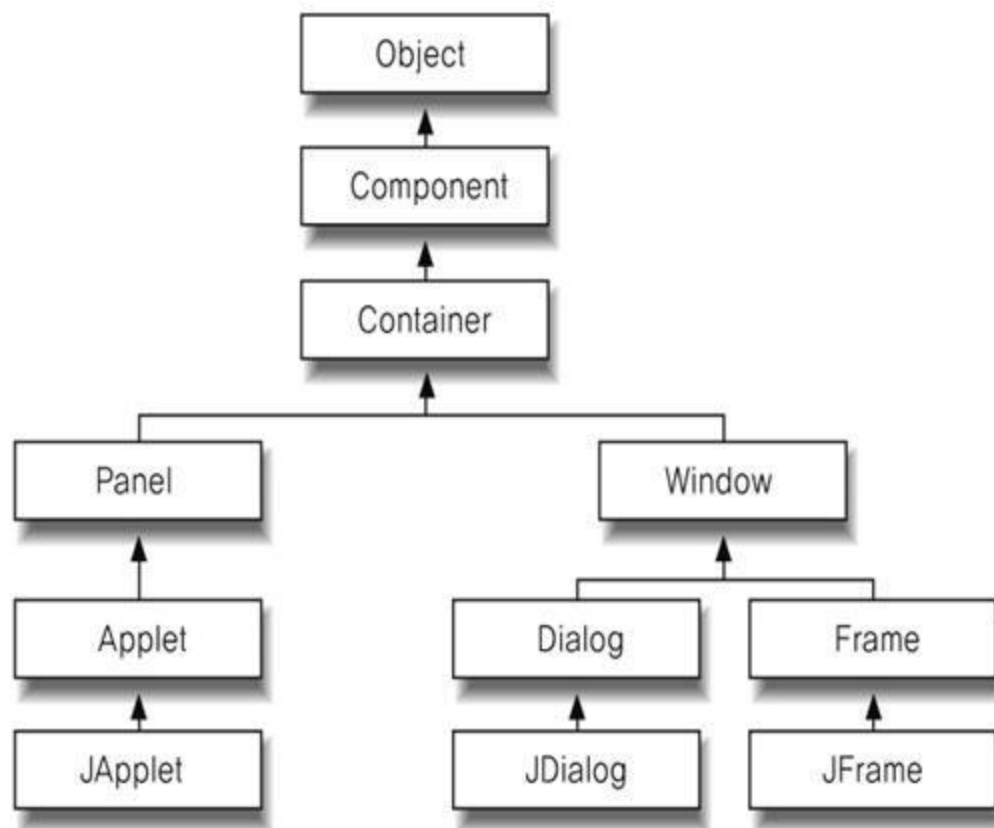
AWT에서 제공하는 컴포넌트들은 JVM이 기반으로 설치되어있는 네이티브 플랫폼에 의존하여 그 컴포넌트들을 그대로 가져다 사용하는 중량의 컴포넌트들이다.

스윙은 순수 자바로 구현되어 있는 컴포넌트들이기 때문에 어떤 플랫폼을 사용하더라도 거기에 의존하지 않고 독립적으로 사용할 수 있는 경량의 컴포넌트이다.

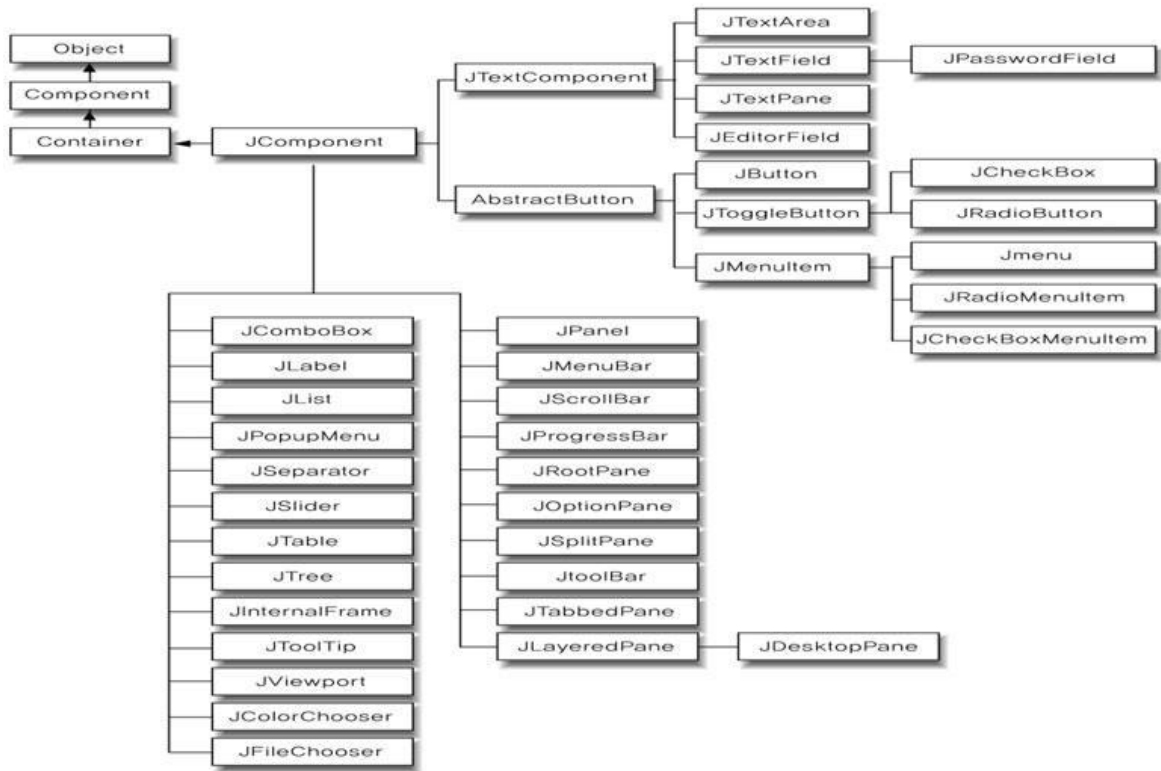
- DupleBuffering지원

더블버퍼링기능은 그래픽의 성능을 향상시키기 위해 도입된 방식으로 AWT에서는 사용자에게 의해 직접 구현해야 되지만 스윙에서는 자체적으로 더블버퍼링 기능을 제공한다.

(2) 컨테이너 구조



(3) 컴포넌트 구조

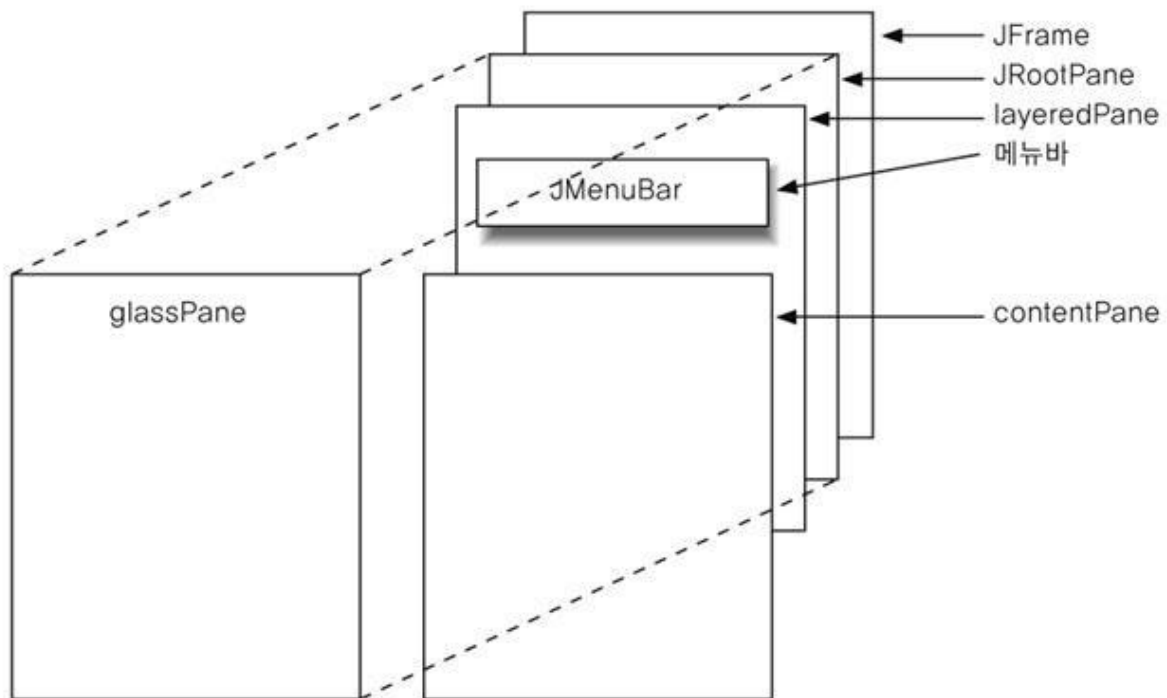


(4) JFrame 클래스

스윙의 **JFrame**은 AWT의 **Frame**과 달리 좀 복잡한 구조로 되어있다.

프레임자체로 구성되어 있는 것이 아니라 그 안에 4개의 페인(**pane**)이 층으로 구성되어 있다.

다음 그림은 **JFrame**의 내부 구조이다.



14. 스레드

14-1 스레드

자바 프로그램을 구성하는 명령문은 순서대로 하나씩 처리되는 것이 기본이다. 조건문, 반복문, 메서드 호출문으로 복잡하게 얹혀있는 프로그램도 명령문이 실행되는 순서를 따라가다보면 길게 연결된 한 가닥 실과 같은 흐름이 있다. 그래서 이런 실행 흐름을 스레드(**thread**)라고 부른다.

(1) 멀티 태스킹

프로세스란 운영체제에서 실행중인 하나의 프로그램을 말한다.

멀티 프로세스란 두 개 이상의 프로세스가 실행되는 것을 말한다.

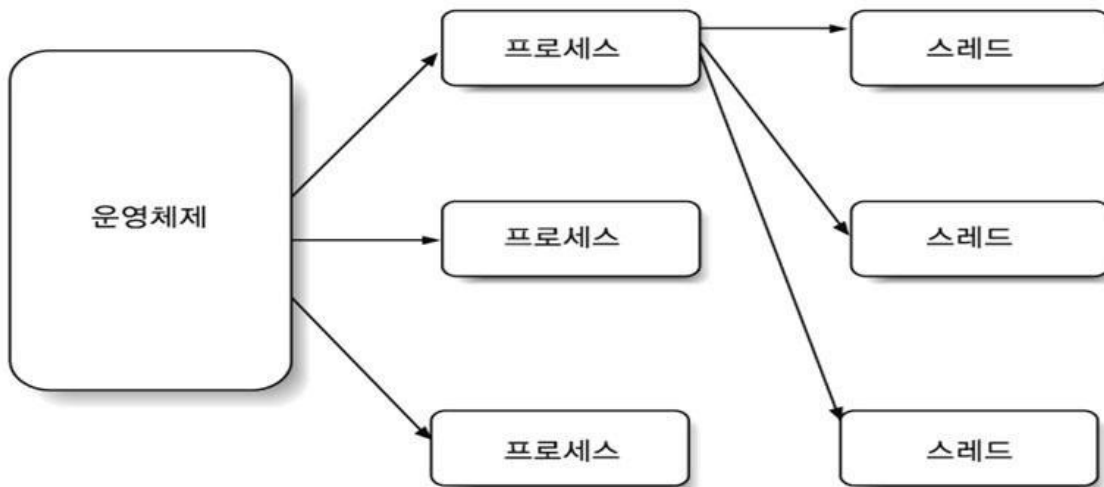
멀티 태스킹이란 두 개 이상의 프로세스를 실행하여 일을 처리하는 것을 말한다.

(2) 멀티 스레드

스레드란 프로세스 내에서 실행되는 세부 작업 단위이다.

멀티 스레드란 하나의 프로세스에서 여러 개의 스레드가 병행적으로 처리되는 것을 말한다.

(3) 프로세스와 스레드의 관계

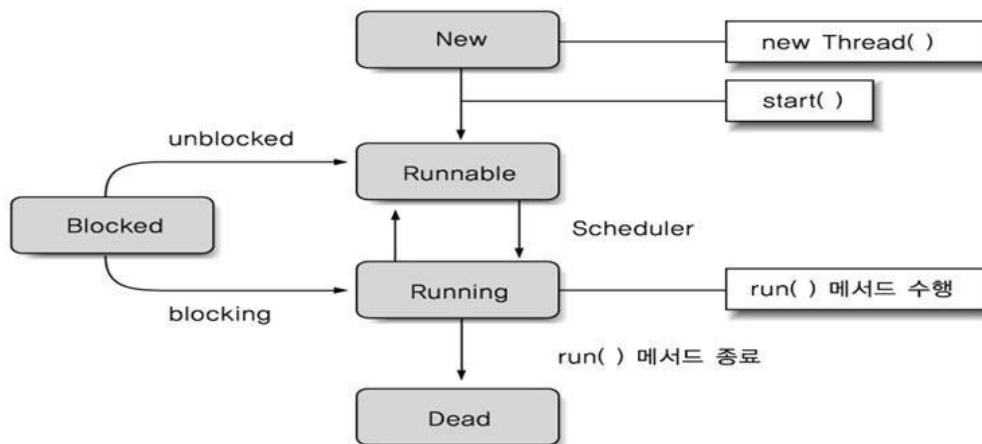


14-2 스레드의 생성 방법

Thread 클래스를 상속 받는 방법

Runnable 인터페이스를 구현하는 방법

(1) 스레드의 생명주기



(2) Thread 클래스를 상속 받는 방법

```

01 public class CreateThread extends Thread{
02     public void run(){
03
04     }
05     public static void main(String[] args){
  
```

```
06      CreateThread ct = new CreateThread();
07      ct.start();
08  }
09 }
```

(3) Runnable 인터페이스를 구현하는 방법

```
01 public class CreateRunnable implements Runnable{
02     public void run(){
03     }
04     public static void main(String[] args){
05         CreateRunnable ct = new CreateRunnable();
06         Thread t = new Thread(ct);
07         t.start();
08     }
09 }
```

14-3 join() 메서드 사용법

join() 메서드는 join() 메서드를 호출한 스레드가 종료할 때까지 종료할 때까지 현재의 스레드를 기다리게 된다.

14-4 스레드 스케줄러

멀티 스레드가 수행될 때 어떤 스레드가 먼저 수행될지는 스레드 스케줄러가 결정하게 된다.

14-5 스레드 우선순위

Thread 클래스에서는 스레드의 우선순위를 부여하는 setPriority(int newPriority) 메서드를 제공한다.

14-6 동기화

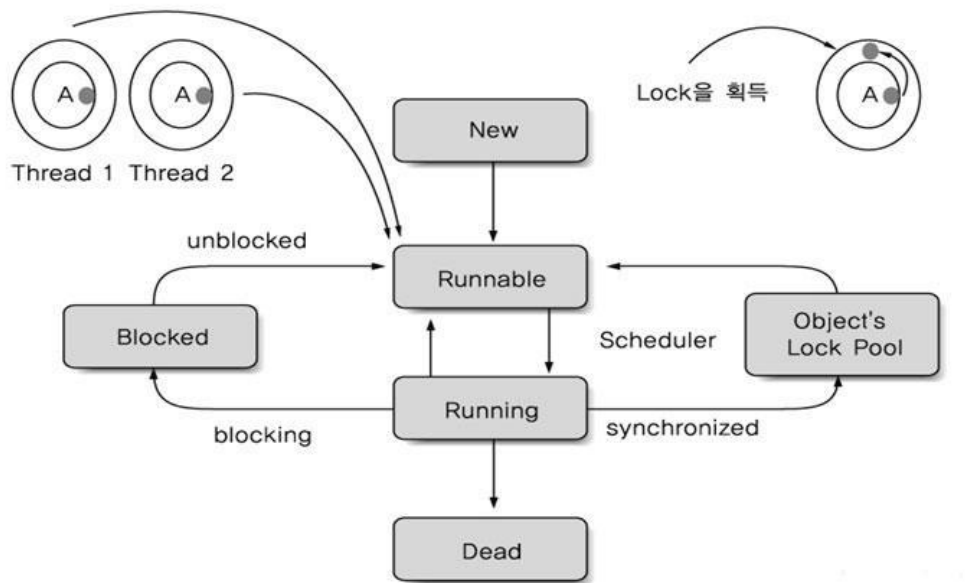
임계영역이란 멀티 스레드에 의해 공유자원이 참조될 수 있는 코드의 범위를 말한다.

멀티 스레드 프로그램에서 임계영역을 처리하는 경우 심각한 문제가 발생할 수 있다.

이러한 상황을 해결할 수 있는 방법이 동기화를 이용하는 것이다.

동기화를 처리하기 위해 모든 객체에 락(lock)을 포함 시켰다.

락이란 공유 객체에 여러 스레드가 동시에 접근하지 못하도록 하기 위한 것으로 모든 객체가 힙 영역에 생성될 때 자동으로 만들어 진다.



15. 입출력 스트림

15-1 스트림(Stream)이란?

데이터를 목적지로 입출력하기 위한 방법이다.

스트림에 데이터를 쓸 수 있고, 스트림에서 데이터를 읽을 수 있다.

스트림에 데이터를 쓸 경우, 이러한 스트림을 출력 스트림(output stream)이라고 한다.

스트림에서 데이터를 읽을 경우, 이러한 스트림을 입력 스트림(input stream)이라고 한다.

15-2 스트림의 특징

스트림은 FIFO 구조이다. – FIFO구조란 먼저 들어간 것이 먼저 나오는 형태로서 데이터의 순서가 바뀌지 않는다는 특징이 있다.

스트림은 단방향이다. – 자바에서 스트림은 읽기, 쓰기가 동시에 되지 않는다. 따라서 읽기, 쓰기가 필요하다면 읽는 스트림과 쓰는 스트림을 하나씩 열어 사용해야 한다.

스트림은 지연될 수 있다. – 스트림은 넣어진 데이터가 처리되기 전까지는 스트림에 사용되는 스레드는 지연상태에 빠진다. 따라서 네트워크 내에서는 데이터가 모두 전송되기 전까지 네트워크 스레드는 지연상태가 된다.

15-3 스트림의 분류

- 용도에 의한 분류

① 1차 스트림 : 디바이스에 직접 연결되는 스트림

② 2차 스트림 : 1차 스트림과 연결을 통해 디바이스에 연결되는 스트림

- 전송 방향에 의한 분류

- ① 입력 스트림 : 디바이스로부터 데이터를 읽어오는 스트림
- ② 출력 스트림 : 디바이스로 데이터를 출력하는 스트림

- 전송 단위에 의한 분류

- ① 바이트 스트림 : 1 Byte 단위로 입력, 출력하는 스트림
- ② 문자 스트림 : 한 문자(2 Byte) 단위로 입력, 출력하는 스트림

- 보조 스트림

스트림의 기능을 향상시키거나 새로운 기능을 추가시킴
직접적인 데이터 입출력을 할 수 없다

15-4 File 클래스

시스템에 있는 파일이나 디렉토리를 추상화한 클래스이다.

File 클래스를 이용하면 파일의 크기, 생성, 삭제, 변경 및 마지막 수정 날짜 등 다양한 정보를 알 수 있는 메서드를 제공하고 있다.

15-5 바이트 스트림

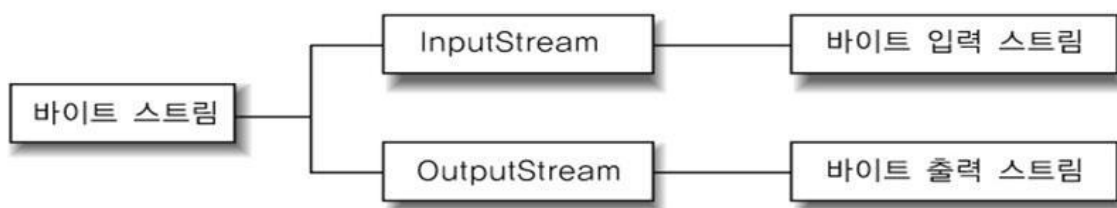
(1) 바이트 스트림

바이트 스트림은 1 byte를 입출력 할 수 있는 스트림이다.

일반적으로 바이트로 구성된 파일, 즉 동영상 파일, 이미지 파일, 음악 파일을 처리하기에 적합한 스트림이다.

(2) 바이트 스트림의 종류

InputStream과 **OutputStream**으로 구성되어 있다.



(3) 바이트 입력 스트림(InputStream)

InputStream은 바이트 입력을 수행하는 데 필요한 메서드를 정의하는 추상 클래스이다.

자바 프로그램은 객체를 생성하고 생성된 객체와 바이트 스트림과 연결함으로써 파일을 연다.

자바는 다른 장치들과도 바이트 스트림을 연결할 되어 있고 프로그램이 시작되면 장치들과 연결된 세 개의 객체(**System.in**, **System.out**, **System.err**)를 생성한다.

System.in 객체는 키보드로 바이트를 입력할 수 있는 **InputStream** 객체이다.

(4) 바이트 출력 스트림(OutputStream)

OutputStream은 바이트 출력을 수행하는 필요한 메서드를 정의한 추상 클래스이다.

프로그램이 시작 되면 장치와 연결된 두 개의 출력 스트림은 **System.out**, **System.err**를 생성한다.

System.out 객체는 화면에 데이터를 출력 한다.

System.err 객체는 화면에 오류 메시지를 출력하게 된다.

15-6 문자 스트림

(1) 문자 스트림의 특징

바이트 스트림에 추가하여 **Reader**와 **Writer** 클래스를 제공하는데, 이것은 2 바이트를 입출력 할 수 있는 문자 기반 스트림이다.

바이트 스트림은 1바이트를 입출력하기 때문에 일반적으로 영문자로 구성된 파일, 동영상 파일, 음악 파일의 입출력 등에 적합한 스트림이다.

문자 스트림은 2바이트를 입출력하기 때문에 세계 모든 언어로 구성된 파일을 입출력 하기에 적합하다.

(2) 문자 스트림의 구조

문자 스트림은 **Reader**와 **Writer**로 나눈다.

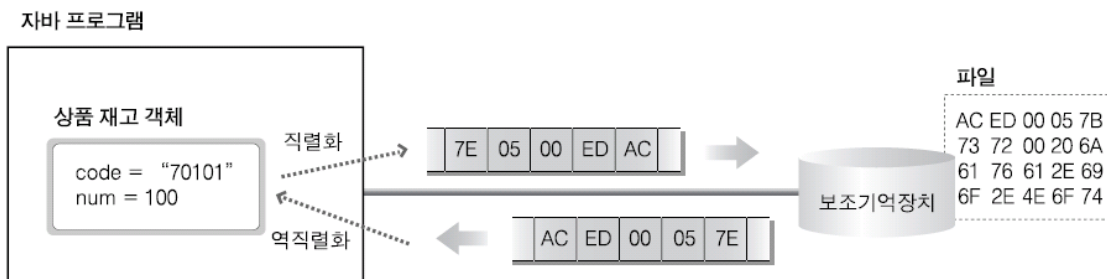
문자 입력 스트림 – **Reader**

문자 출력 스트림 - **Writer**

15-6 객체의 직렬화

(1) 객체의 직렬화

- 직렬화(**serialization**) : 객체를 스트림으로 만드는 작업
- 역직렬화(**deserialization**) : 스트림을 객체로 만드는 작업



(2) Serializable

Serializable 인터페이스를 구현한 클래스를 작성하면 해당 클래스의 모든 멤버변수가 직렬화 대상이 된다.

객체가 스트림을 통해 직렬화 될 때는 객체에 있는 멤버변수가 직렬화 되는 것이다.

객체의 멤버변수 중에 직렬화 대상에 제외하고 싶다면 **transient** 키워드를 사용하면 된다.

16. 네트워크

16-1 네트워크

(1) 네트워크

네트워크란 다른 장치로 데이터를 이동시킬 수 있는 컴퓨터들과 주변 장치들의 집합이다.

네트워크의 연결된 모든 장치들을 노드라고 한다.

다른 노드에게 하나 이상의 서비스를 해주는 노드를 호스트라 부른다.

하나의 컴퓨터에서 다른 컴퓨터로 데이터를 이동시킬 때 복잡한 계층을 통해 전송되는데, 이런 복잡한 레이어의 대표적인 모델이 OSI 계층 모델이다.

OSI 계층 모델은 모두 7계층으로 이루어졌다.

데이터 통신을 이해하는데 OSI 계층 모델은 상당한 역할을 하지만, 인터넷 기반의 표준 모델로 사용하는 TCP/IP 계층 모델을 주로 사용하고 있다.

자바에서 이야기하는 네트워크 프로그래밍은 TCP/IP 모델을 사용하고 있다.

(2) 인터넷 주소(IP 주소)

모든 호스트는 인터넷 주소(Host 또는 IP 주소)라 불리는 유일한 32비트 숫자로 구성된 주소체계를 이용하여 서로를 구분할 수 있다.

IP 주소는 32비트 숫자를 한번에 모두를 표현하는 것이 힘들기 때문에, 8 비트씩 끊어서 표현하고, 각 자리는 1바이트로 0~255 까지의 범위를 갖게 된다.

32비트의 주소 체계를 IP 버전 4(IPv4) 주소라고 한다.

192.168.0.1

오늘날 IPv4는 포화 상태이고, 이를 극복하고자 나온 것이 IP 버전 6(IPv6)이다.

IPv6는 128 비트의 주소 체계를 관리하고 있으며, 16비트씩 8부분으로 나누어 16진수 표시한다.

FECD:BA98:7654:3210:FECD:BA98:7652:3210

(3) DNS

각 호스트는 도메인 이름을 컴퓨터가 사용하는 주소(IP 주소)로 바꾸어 주어야 한다. 이렇게 IP 주소를 도메인 이름으로 바꾸어 주는 시스템을 DNS(Domain Name System)이라고 한다.

(4) 포트

포트는 크게 두 가지로 구분된다.

컴퓨터의 주변장치를 접속하기 위한 '물리적인 포트'와 프로그램에서 사용되는 접속 장소인 '논리적인 포트'가 있다.

이 장에서 말하는 포트는 '논리적인 포트'를 말한다.

포트번호는 인터넷번호 할당 허가 위원회(IANA)에 의해 예약된 포트번호를 가진다.

이런 포트번호를 '잘 알려진 포트들'라고 부른다.

예약된 포트번의 대표적인 예로는 80(HTTP), 21(FTP), 22(SSh), 23(TELNET)등이 있다.

포트번호는 0~65535까지이며, 0~1023까지는 시스템에 의해 예약된 포트번호이기 때문에 될 수 있는 한 사용하지 않는 것이 바람직하다.

(5) 프로토콜

프로토콜은 클라이언트와 서버간의 통신 규약이다.

프로토콜은 통신 시스템이 데이터를 교환하기 위해 사용하는 통신 규칙이다.

통신규약이란 상호 간의 접속이나 절단방식, 통신방식, 주고받을 데이터의 형식, 오류검출 방식, 코드변환방식, 전송속도 등에 대하여 정의하는 것을 말한다.

대표적인 인터넷 표준 프로토콜에는 TCP와 UDP가 있다.

(6) TCP와 UDP

TCP/IP 계층 모델은 4계층의 구조를 가지고 있다.

애플리케이션, 전송, 네트워크, 데이터 링크 계층이 있다.

이 중 전송계층에서 사용하는 프로토콜에는 TCP와 UDP가 있다.

- TCP

TCP(Transmission Control Protocol)는 신뢰할 수 있는 프로토콜로서, 데이터를 상대측까지 데대로 전달되었는지 확인 메시지를 주고 받음으로써 데이터의 송수신 상태를 점검한다.

연결형 서비스를 지원하는 프로토콜.

- UDP

UDP(User Datagram Protocol)은 신뢰할 수 없는 프로토콜로서, 데이터를 보내기만 하고 확인 메시지를 주고 받지 않기 때문에 제대로 전달했는지 확인하지 않는다.

비연결형 서비스를 지원하는 프로토콜.

TCP – 전화, UDP - 편지

16-2 InetAddress 클래스

InetAddress 클래스는 IP 주소를 표현한 클래스이다.

자바에서는 모든 IP 주소를 InetAddress 클래스를 사용한다.

16-3 URL 클래스

URL(Uniform Resource Locator)이란 인터넷에서 접근 가능한 자원의 주소를 표현할 수 있는 형식을 말한다.

<u>http://www.sist.co.kr:80/member/mem.jsp?name=sung#content</u>					
protocol	host	port	path	query	reference
<schema> ://	<authority>		<path> ?	<query>#	<fragment>

16-4 URLConnection 클래스

어플리케이션과 URL간의 통신연결을 위한 작업을 하는 클래스 URL 내용을 읽어오거나, URL 주소에 GET / POST 메소드 형식으로 데이터를 전달할 때 사용 추상 클래스로써 자신의 객체를 생성할 수 없고 URL 객체의 openConnection()을 통해 사용한다

URL -> openConnection() -> URLConnection -> getInputStream -> InputStream(내용읽기) 순으로 처리한다

URL 클래스의 openStream()은 입력 스트림만 개설하지만 URLConnection을 이용하면 입력 스트림과 출력 스트림 모두 개설할 수 있다

16-4 소켓

자바 프로그램은 소켓(Socket)이라는 개념을 통해서 네트워크 통신을 한다.

소켓은 네트워크 부분의 끝 부분을 나타내며, 실제 데이터가 어떻게 전송되는지 상관하지 않고 읽기/쓰기 인터페이스를 제공한다.

네트워크 계층과 전송 계층이 캡슐화 되어 있기 때문에 두 개의 계층을 신경 쓰지 않고 프로그램을 만들 수 있다.

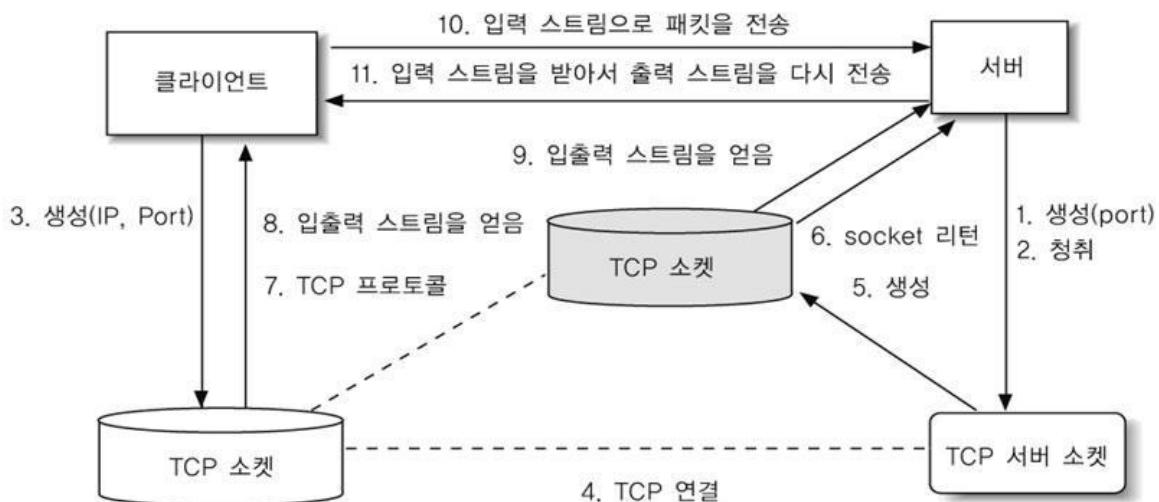
소켓은 캘리포니아 대학교에서 빌 조이(Bill Joy)에 의해 개발되었다.

자바는 이식성과 크로스 플랫폼 네트워크 프로그램을 위해서 소켓을 핵심 라이브러리 만들었다.

TCP/IP 계층의 TCP를 지원하기 위해서 Socket, ServerSocket 클래스를 제공하고 있다.

클라이언트는 Socket 객체를 생성하여 TCP 서버와 연결을 시도한다.

서버는 ServerSocket 객체를 생성하여 TCP 연결을 청취하여 클라이언트와 서버가 연결된다.

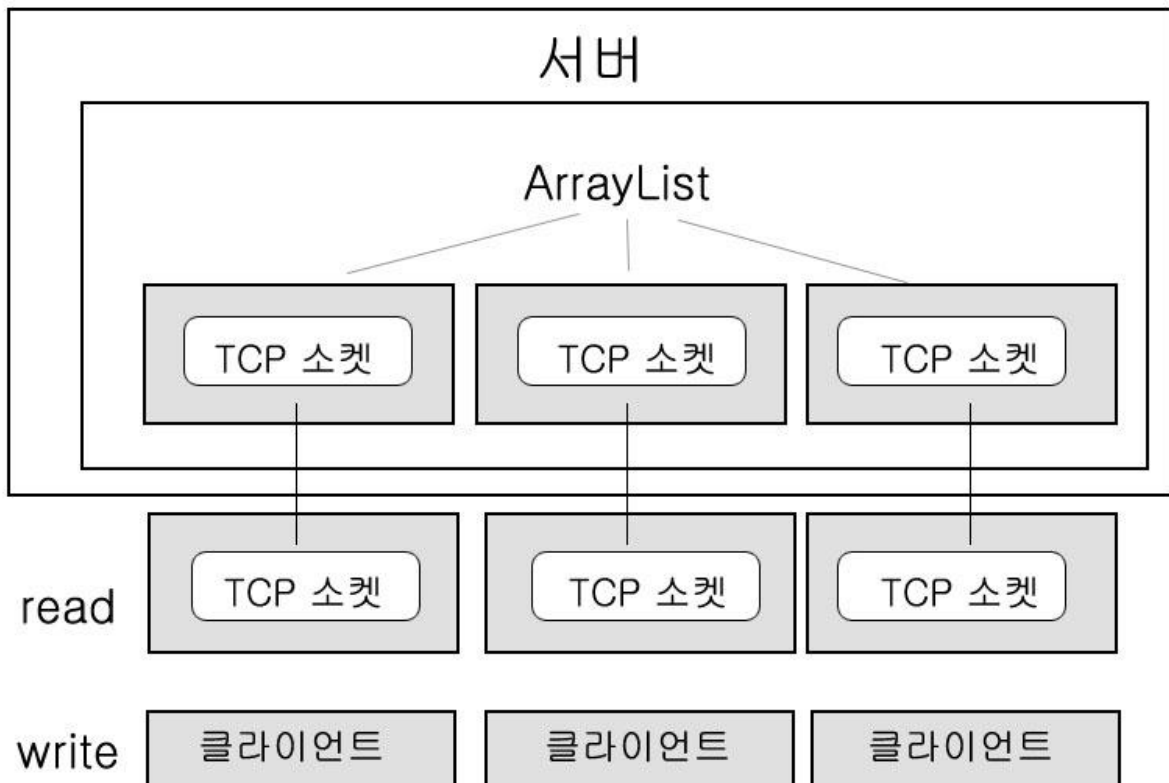


16-5 UDP(User Datagram Protocol)

UDP 통신은 User Datagram Protocol의 약자로 TCP 통신과 더불어 데이터를 주고 받는 통신

체계이다. 차이점은 **TCP** 통신은 매번 통신할 때마다 세션을 설정하여 연결을 유지하는데 반하여 **UDP** 통신은 세션 설정 없이 **IP** 주소만으로 데이터를 전송한다. 따라서 하나의 통신에서 고속으로 데이터를 전송할 수 있다는 장점이 있다. 그러나 **TCP**와 다르게 수신 상태를 확인할 수 없어서 데이터가 제대로 전달되었는지 여부는 알 수 없는 단점이 있다.

16-6 멀티캐스팅



모든 클라이언트가 서버에 접속할 때 저장공간(**ArrayList**)을 두어 클라이언트가 접속할 때마다 스레드에 **TCP** 소켓을 저장하고, 다시 스레드를 **ArrayList**에 저장하는 구조를 가지고 있다. 그래서 클라이언트가 접속할 때마다 해당 클라이언트의 스레드를 **ArrayList**에 저장하고 **ArrayList**에 저장된 각각의 클라이언트에 대한 스레드를 가지고 메시지를 전송하게 된다.

멀티캐스트 프로그램에 필요한 클래스 구조

클래스명	설명
MultiServer	모든 클라이언트의 TCP 요청을 받아 소켓 객체를 생성한다. 소켓을 유지하기 위한 스레드를 생성하고 이 스레드를 저장할 Collection(ArrayList) 을 생성하는 클래스다.
MultiServerThread	각각의 클라이언트의 소켓 객체를 유지하기 위한 클래스다. 이

	클래스는 멀티서버에 있는 컬렉션을 가지고 있기 때문에 다른 클라이언트에게 메시지를 보낼 수 있다.
MultiClient	스윙으로 구현된 클라이언트 클래스다. 이 클래스에서는 메시지를 보낼 때는 이벤트에서 처리했고, 다른 클라이언트가 보낸 메시지를 받기 위해 MultiClientThread 객체를 생성했다.
MultiClientThread	다른 클라이언트의 메시지를 받기 위한 클래스다

17. JDBC

17-1 데이터 베이스

(1) 데이터 베이스

지속적으로 저장되는 연관된 정보의 모음이다.

즉, 특정 관심의 데이터를 수집하여 그 데이터의 성격에 맞도록 잘 설계하여 저장하고 관리함으로써 필요한 데이터를 효율적으로 사용할 수 있는 자원이다.

(2) DBMS(Database Management System)

데이터를 효율적으로 관리할 수 있는 시스템을 말한다.

이런 데이터를 효율적으로 관리하기 위해서는 데이터 베이스에 추가, 삭제, 변경, 검색을 할 수 있는 기능이 있어야 한다.

RDBMS 관계형 데이터 베이스 특징

SQL문을 실행하여 액세스하고 수정
물리적인 포인터가 없는 테이블을 가고 있다.
연산자 집합을 사용한다.

(3) SQL(Standard Query Language)

SQL은 RDBMS의 표준 언어이다.

SQL문을 이용해서 단순한 쿼리뿐만 아니라 데이터 베이스 객체를 만들거나, 제거하고, 데이터를 삽입, 갱신, 삭제하거나 다양한 운영 작업을 할 수 있다.

SQL문이 첫선을 보인 것은 1970년대 IBM에 의해서이며, 이후 ANSI/ISO 표준으로 편입되어 여러 차례의 개량과 개발을 거쳤다.

SQL의 종류는 크게 데이터와 구조를 정의하는 DDL, 데이터의 검색과 수정을 위한 DML, 데이터 베이스의 권한을 정의하는 DCL로 구분할 수 있다.

(4) SQL문의 종류

DDL문

SQL문	설명
CREATE	데이터베이스 객체를 생성
DROP	데이터베이스 객체를 삭제
ALTER	기존에 존재하는 데이터베이스의 객체를 다시 정의하는 역할

DML문

SQL문	설명
SELECT	데이터베이스 객체의 데이터로부터 데이터를 검색
INSERT	데이터베이스 객체에 데이터를 입력
UPDATE	데이터베이스 객체에 데이터를 갱신
DELETE	데이터베이스 객체에 데이터를 삭제

DCL문

SQL문	설명
GRANT	데이터베이스 객체에 권한을 부여
REVOKE	이미 부여된 데이터베이스 객체의 권한을 취소

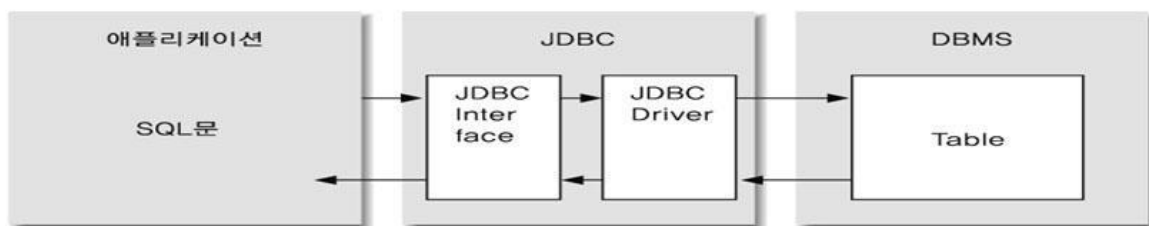
17-2 JDBC(Java Database Connectivity)

(1) JDBC

JDBC란 자바를 이용하여 데이터베이스에 접근하여 각종 SQL문을 수행할 수 있도록 제공하는 API를 말한다.

(2) JDBC의 구조와 역할

JDBC는 크게 JDBC 인터페이스와 JDBC 드라이버로 구성되어 있다.



(3) JDBC 드라이버의 종류

JDBC 드라이버는 DBMS의 벤더나 다른 연구 단체들에서 만들어진다.

(4) JDBC를 이용한 데이터베이스 연결 방법

1 단계 : 드라이버를 로드 한다.

2 단계 : **Connection** 객체를 생성한다.

3 단계 : **Statement** 객체를 생성한다.

4 단계 : SQL문에 결과물이 있다면 **ResultSet** 객체를 생성한다.

5 단계 : 모든 객체를 닫는다.