

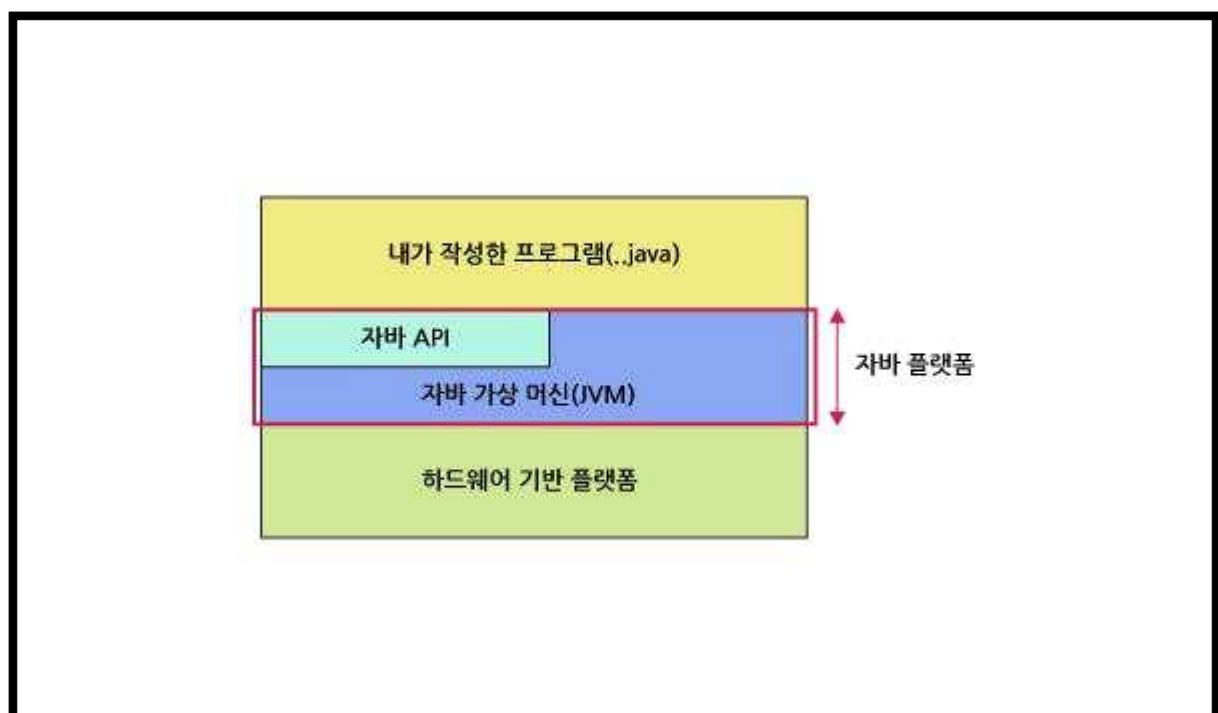
1. 자바의 탄생

- ① 썬 마이크로시스템즈의 제임스 고슬링(James Gosling)과 다른 연구원들에 의해 개발된 객체 지향적 프로그래밍 언어
- ② 처음에는 가전제품 내에 탑재되어 동작하는 프로그램을 위해 개발되었지만 현재 웹 어플리케이션 개발에 가장 많이 사용되는 언어

2. 자바의 특징

- ① 객체지향 언어
- ② 플랫폼 독립성 : 이식성이 높음
- ③ 가비지 컬렉션 : 자동으로 메모리를 관리
- ④ 멀티스레드 : 쉽고 간편하게 멀티스레드를 적용가능
- ⑤ 네트워크 / 분산처리 지원

3. 자바 플랫폼



① 자바 플랫폼의 구성

■ 자바 API(Application Programming Interface)

GUI(Graphical User Interface)와 같은 작은 장치들과 유용한 능력을 제공하는 많은 클래스와 인터페이스들의 묶음이며 패키지로 제공

■ JVM(자바가상머신, Java Virtual Machine)

② 자바 플랫폼의 종류

■ Java Platform, SE(Standard Edition)

표준 플랫폼으로서 응용 프로그램을 개발하기 위해 제공되는 환경

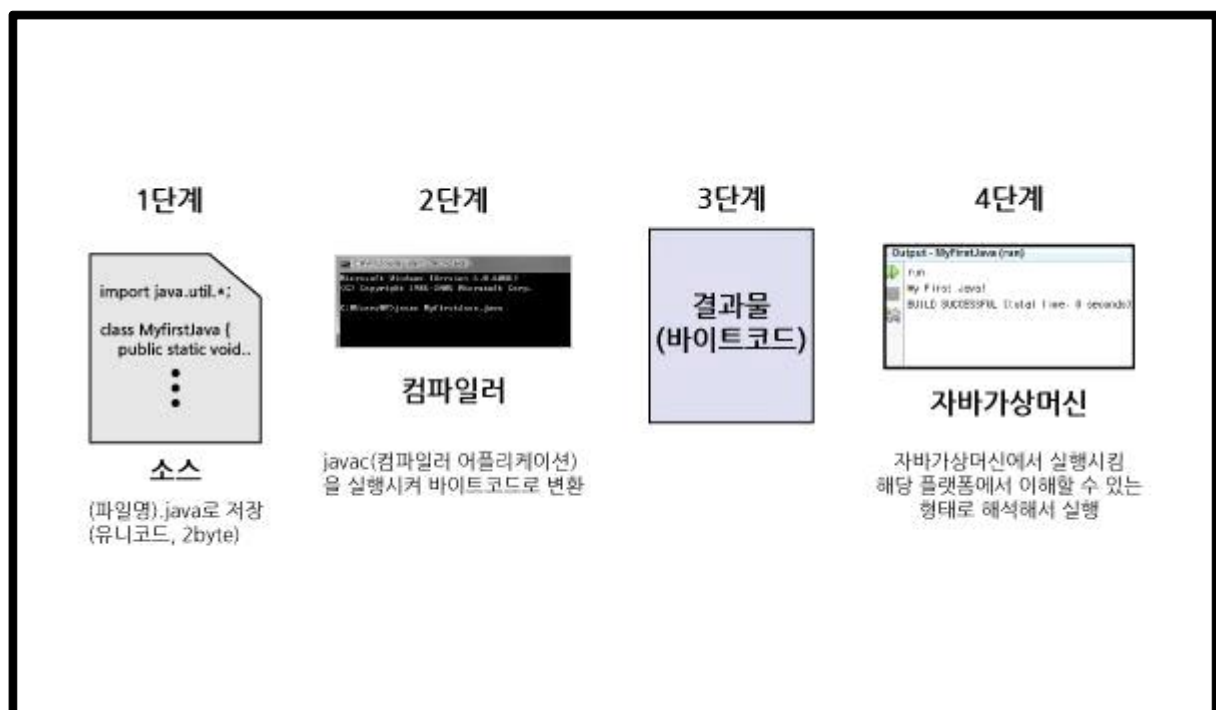
■ Java Platform, EE(Enterprise Edition)

Java SE를 바탕으로 추가적인 tool이나 개발에 필요한 API를 제공

■ Java Platform, ME(Micro Edition)

모바일기기나 가전기기 등의 장치에서 필요로 하는 응용프로그램들의 실행을 위해 제공되는 유연하고 견고한 환경

4. 자바의 실행단계



5. JVM(자바가상머신, Java Virtual Machine)

① 자바 바이트코드를 수행할 수 있는 환경

② 대부분의 운영체제나 웹 브라우저 등 여러가지 플랫폼에

설치되어 사용될 수 있고, 휴대전화나 가전기기에도 설치가능

③ JVM의 구성

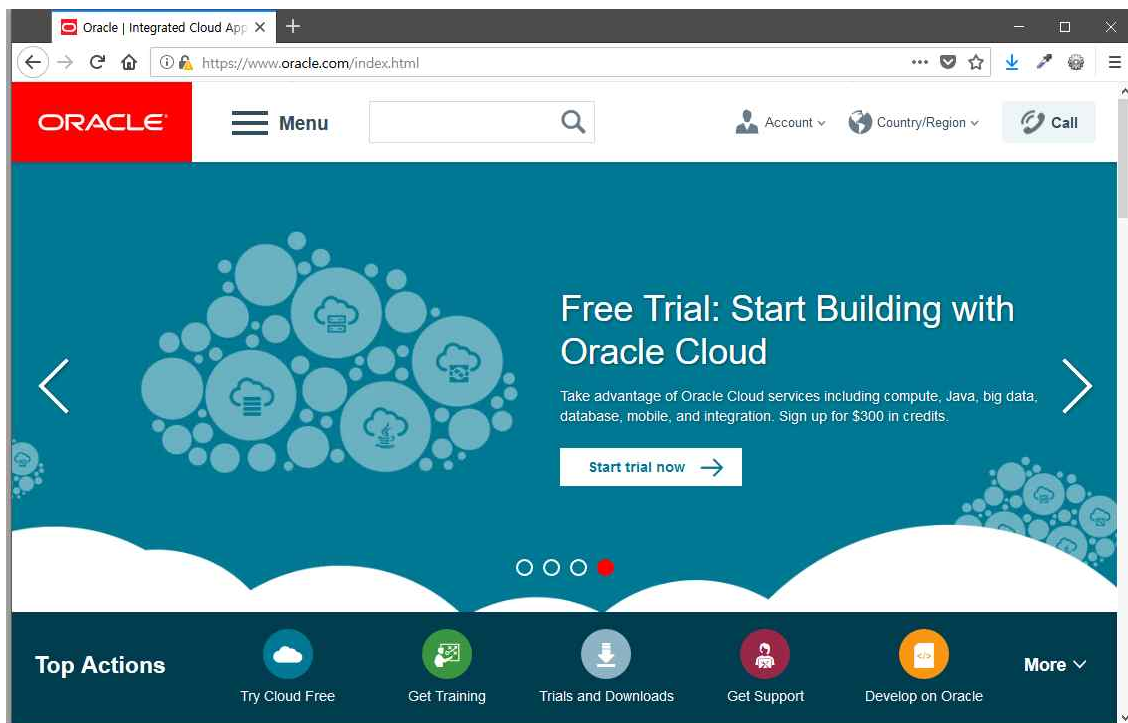
■ 클래스 영역 : 클래스코드를 저장하는 영역

■ 자바스택(Java Stack) : 메서드를 호출할 때 관련정보를 저장하는 영역

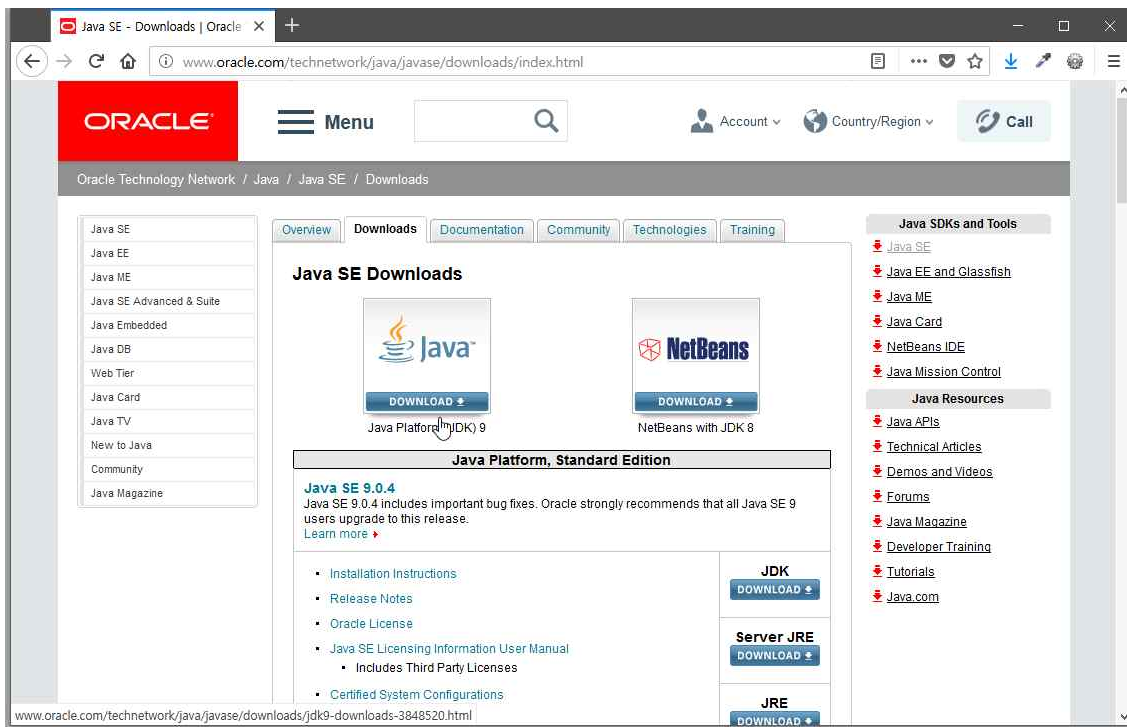
■ 힙(Heap) : new라는 키워드를 통해 객체가 생성될 때 할당받는 영역

■ 네이티브 메서드 스택(Native Method Stack)

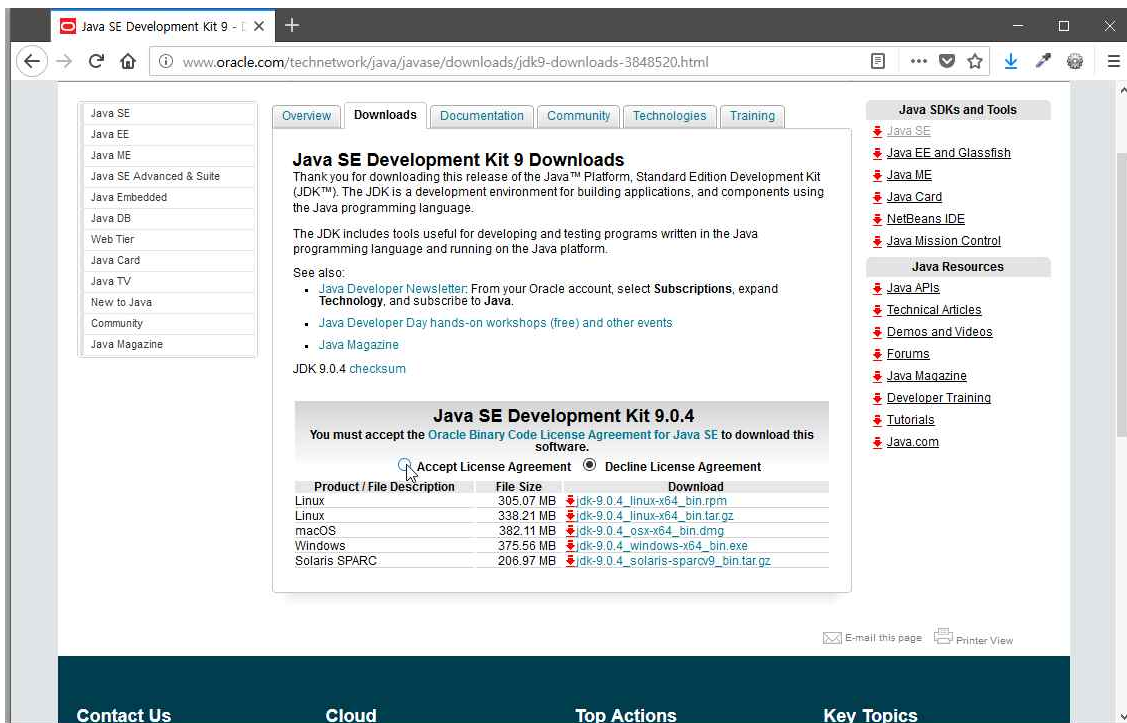
6. 자바의 설치



- <http://oracle.com>



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



- accept후 window 다운로드



- JDK 설치



- 설치중



- JRE 설치

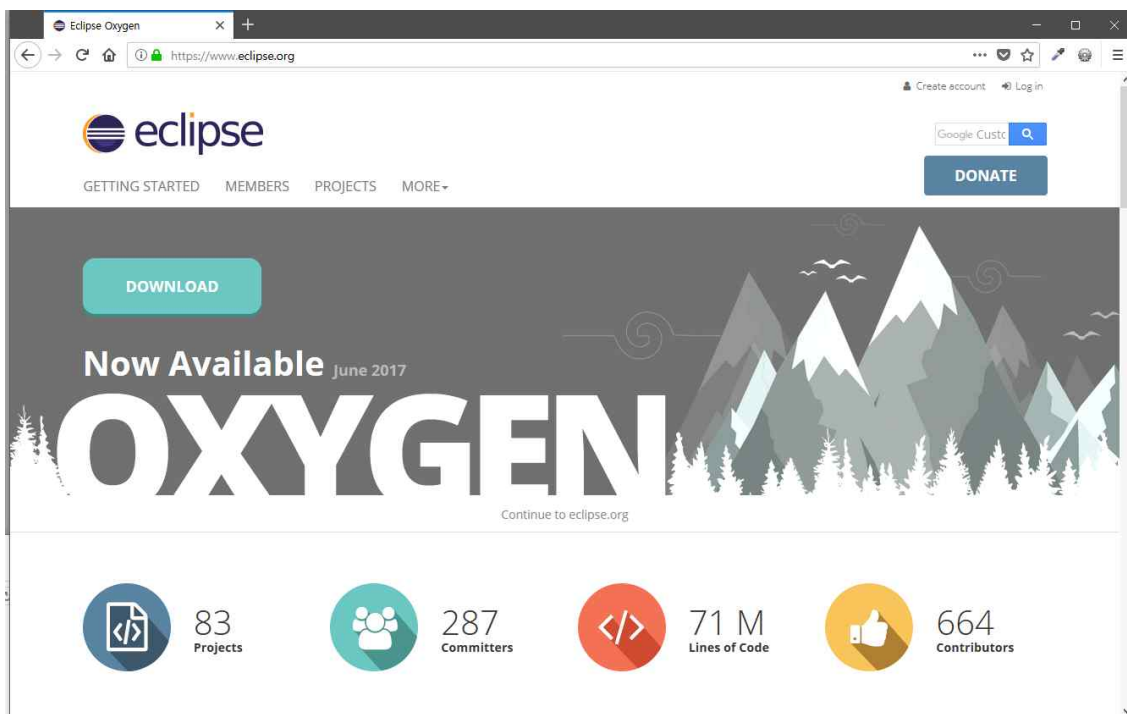


- 설치중



- 설치 완료

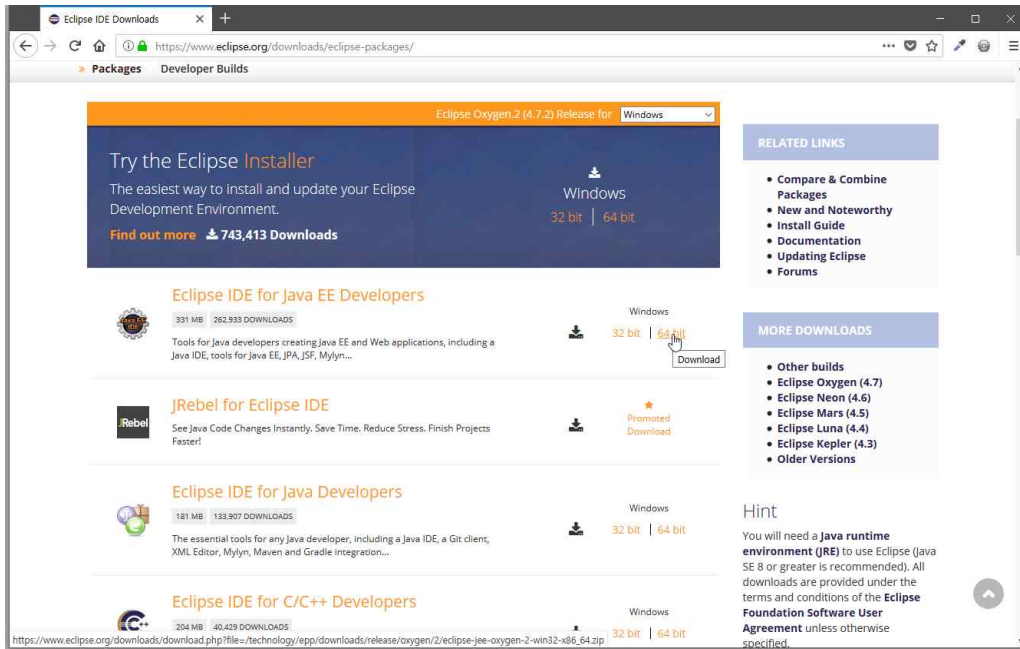
7. Eclipse



- <https://www.eclipse.org/>

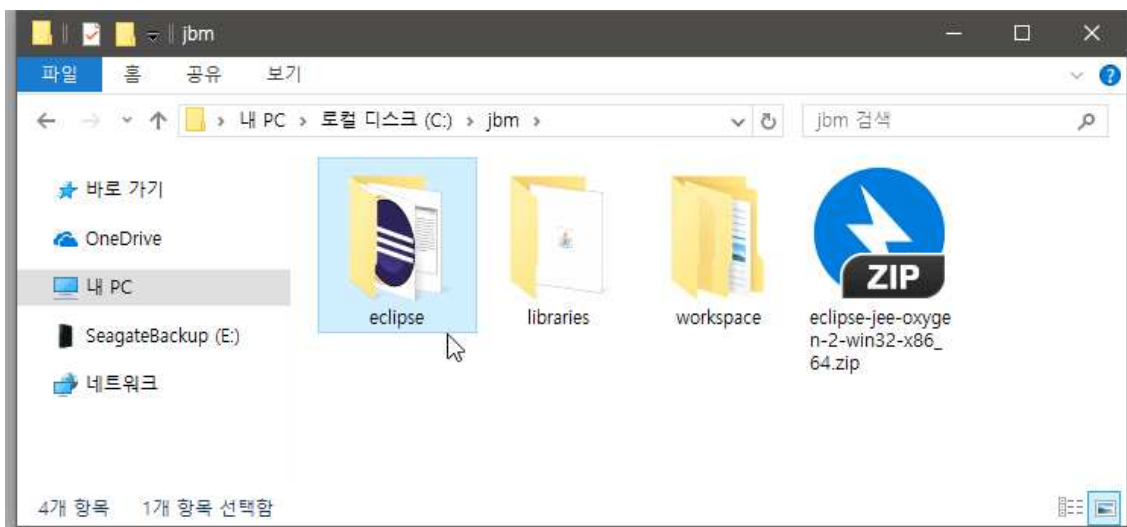
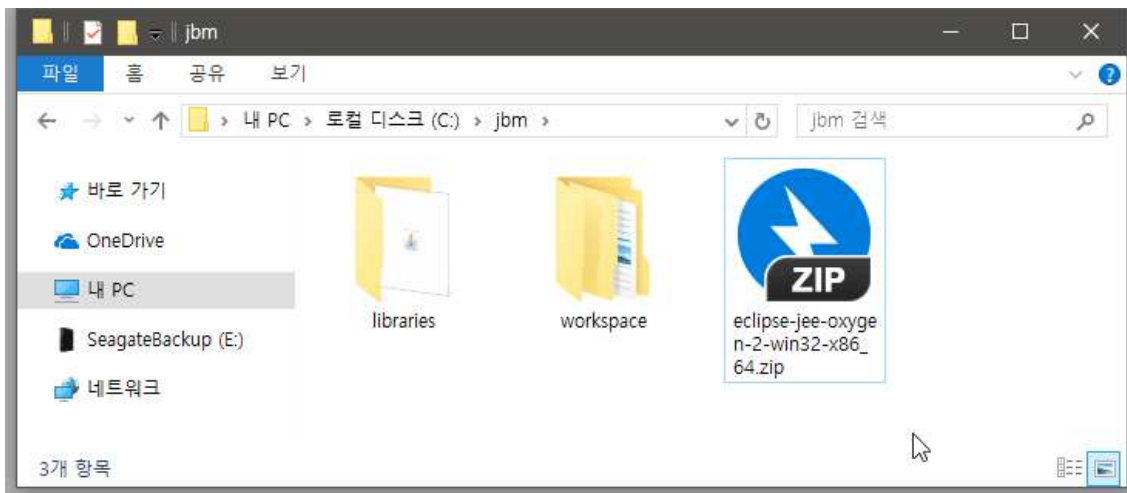
- 한국에서 가장 많이 쓰는 IDE

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

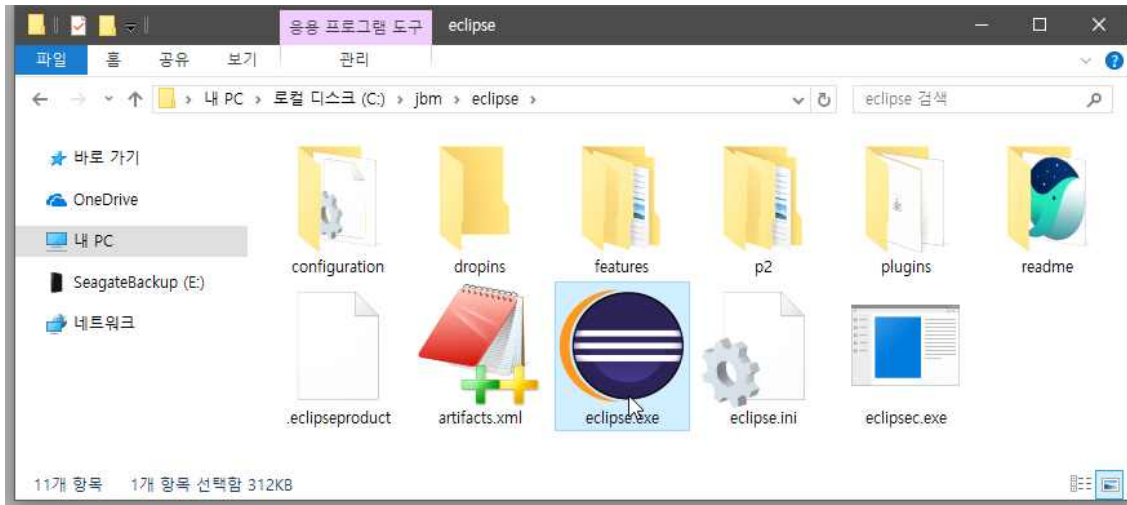


- exe 파일도 가능하나 zip파일로 다운받음

- jbm 폴더에 압축파일을 가져다놓고 알아서 압축풀기



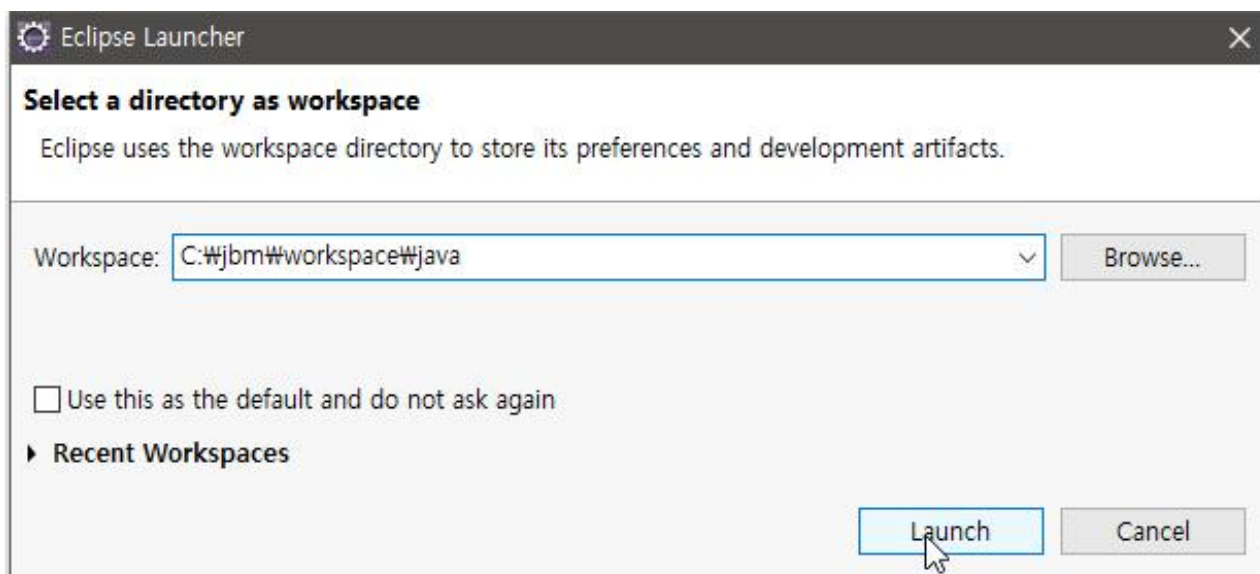
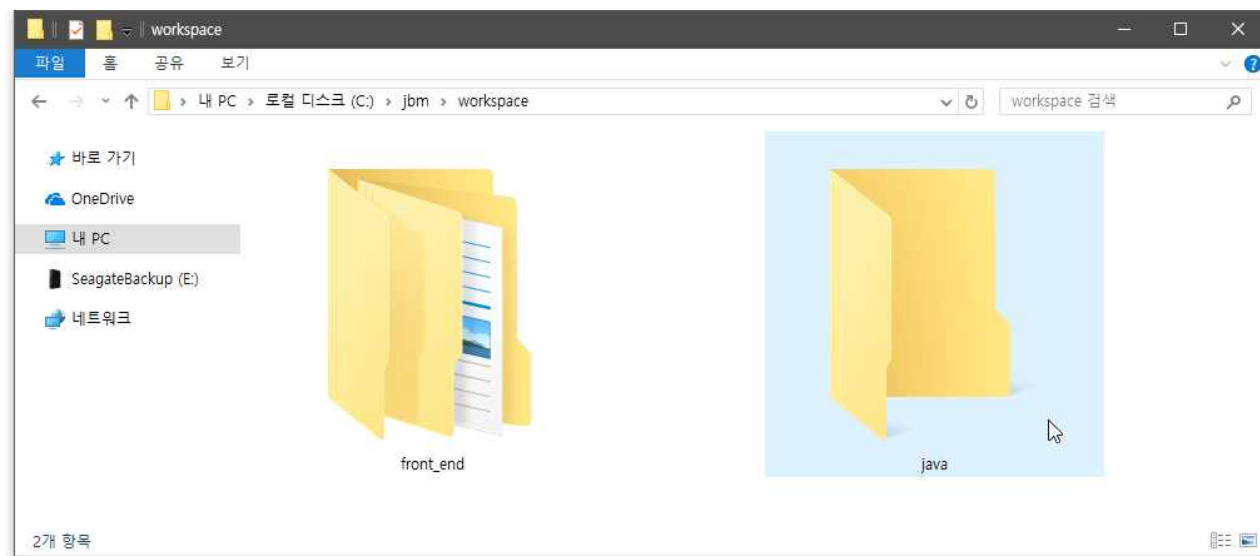
- eclipse라는 폴더가 생성됨



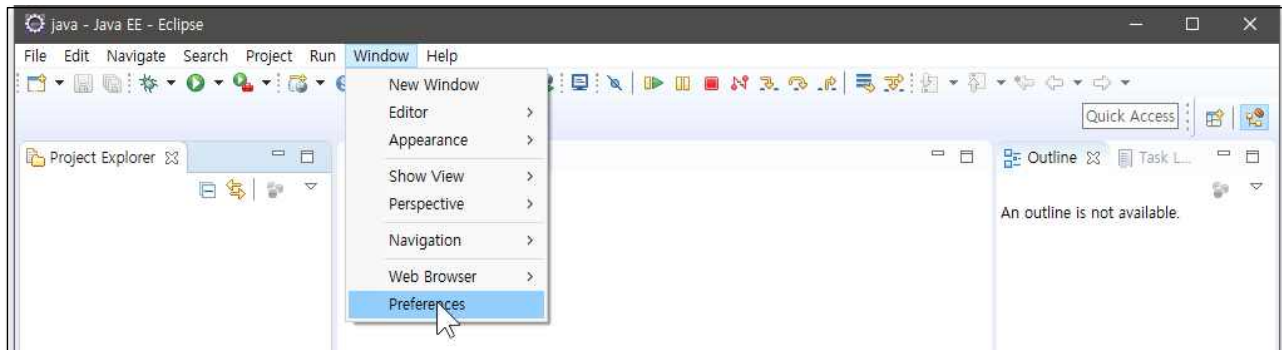
- eclipse.exe 파일을 실행

7. Eclipse 설치후 workspace 설정

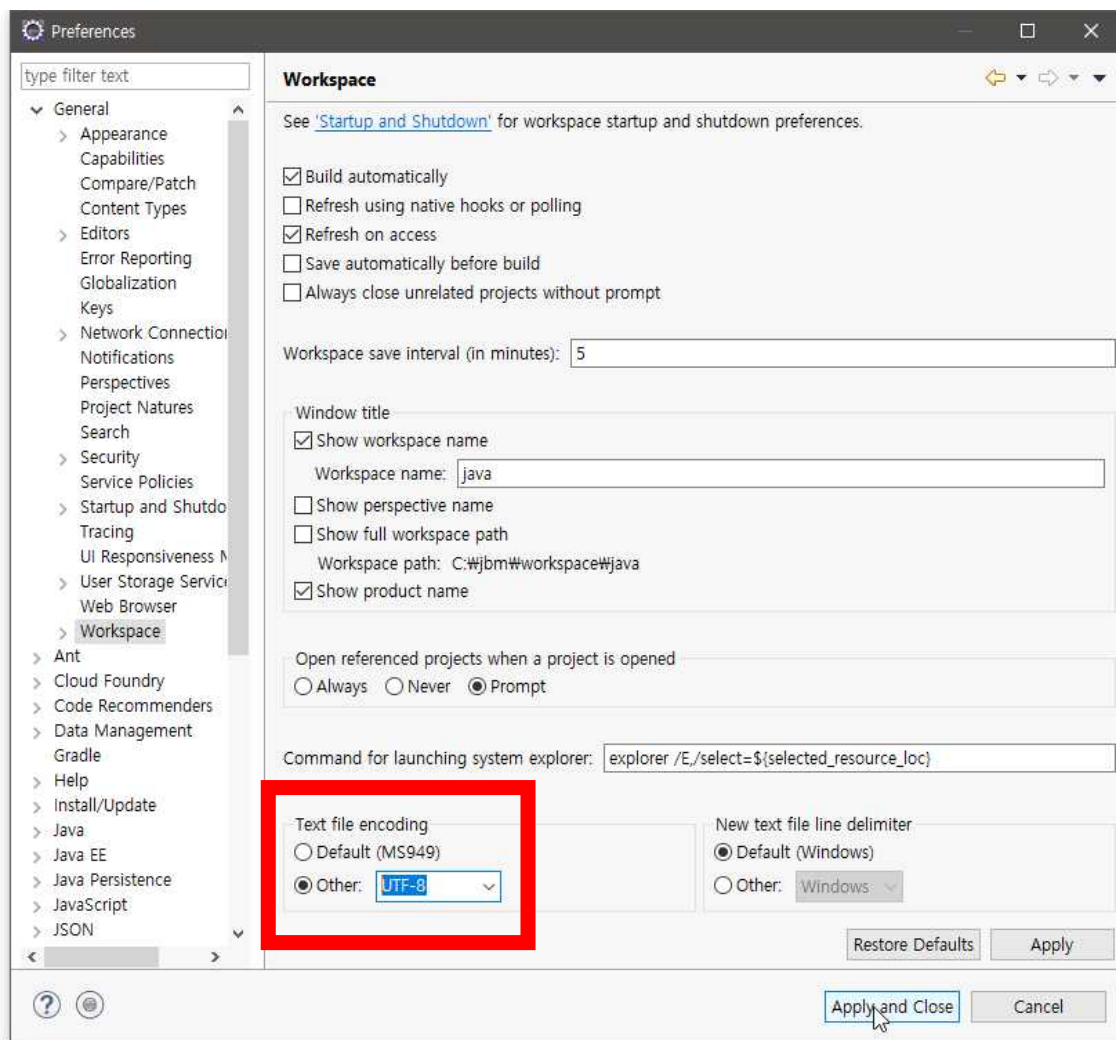
- c드라이브 -> jbm -> workspace -> java 폴더 생성



■ 한글 인코딩 설정

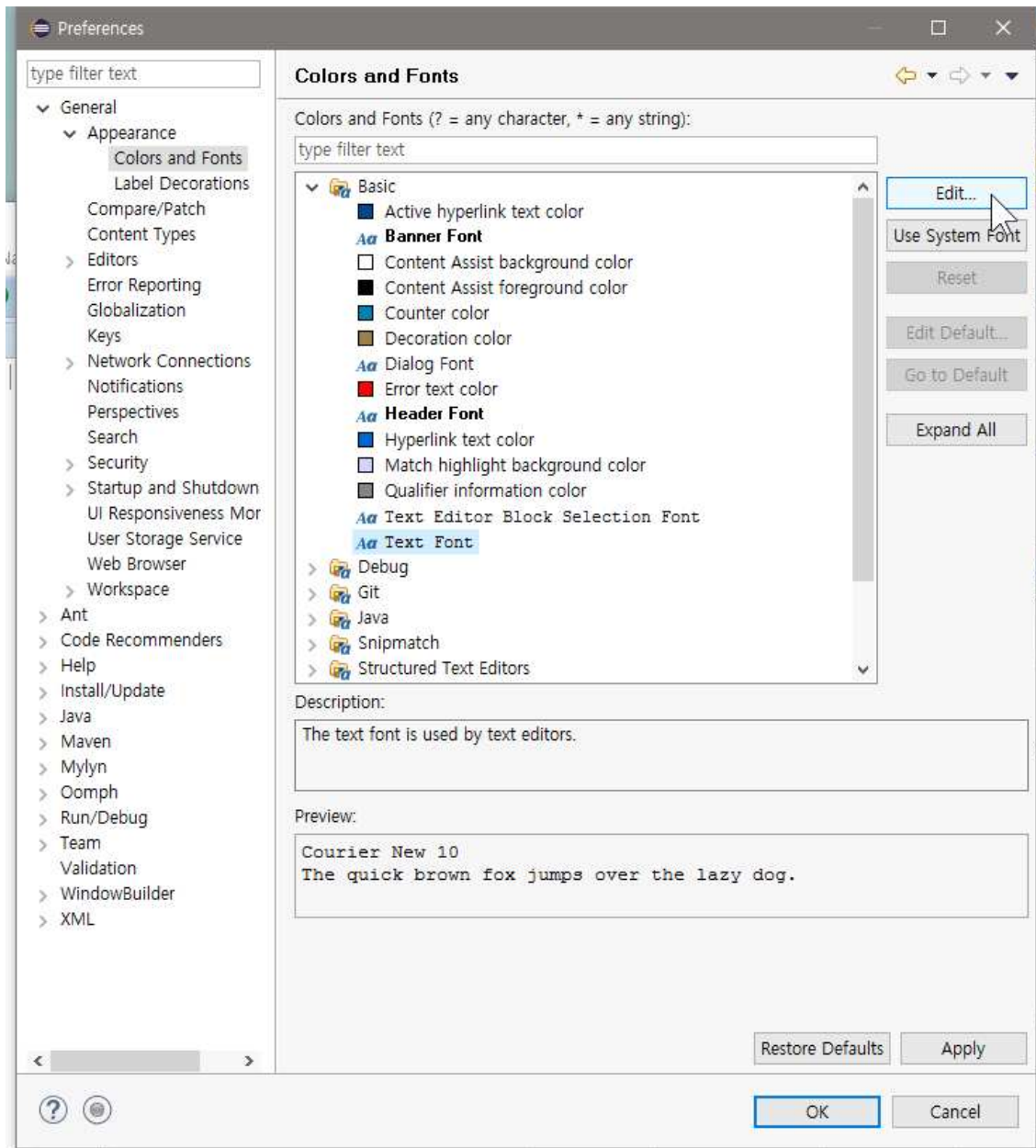


window -> preferences -> General -> Workspace



■ 폰트 설정

General -> Appearance -> Colors and Fonts -> Basic -> Text Font



8. 식별자

■ 구별할 수 있는 이름

- 쉽게 말해 ‘철수’, ‘영희’ 같은 **이름**

■ 자바에서는 **클래스, 메서드, 변수, 생성자, 상수** 가 존재함

■ 식별자를 정의하는 규칙

- 중간에 공백문자, 일반적인 특수문자 사용 불가
- 단 ‘\$’ 와 ‘_’ 는 사용가능 (첫 글자로는 사용하지 않는 것이 좋음)
- 시작은 숫자로 사용하지 않음
- 예약어 사용 불가
- 글자수 제한은 없지만 가독성이 좋게 작성

■ 자바의 예약어

- 자바프로그래밍에서 벌써 사용하고 있는 키워드

abstract	assert	boolean	break	byte	case	catch
char	class	const	continue	default	do	double
else	enum	extends	false	final	finally	float
for	goto	if	implements	import	instanceof	int
interface	long	native	new	null	package	private
protected	public	return	short	static	strictfp	super
switch	synchronized	this	try	void	while	

■ 클래스의 정의 규칙

- 첫글자를 대문자 / 나머지는 소문자

예) Dog, Test...

- 두 단어가 합쳐질 경우 단어의 첫글자는 대문자 :

예) SuperMan, WebDesign

※ 단어의 첫글자를 대문자로 지정하는 것을 낙타(camel)표기법이라 함
(모양이 낙타의 혹같이 생겼다고 하여 붙여진 이름)

■ 변수의 정의 규칙

- 첫글자 소문자 / 단어가 합쳐질 경우 첫글자는 대문자

예) dog, test, superMan, webDesign...

■ 메서드의 정의 규칙

- 첫글자 소문자 / 단어가 합쳐질 경우 첫글자는 대문자
- 제일 뒤에 ‘()’ 가 붙음

예) dog(), test(), superMan(), webDesign()...

■ 상수의 정의 규칙

- 모두 대문자 / 단어가 합쳐질 경우 ‘_’로 표기

예) DOG, TEST, SUPER_MAN, WEB_DESIGN...

■ 생성자의 정의 규칙

- 클래스명에 ‘()’가 붙음

예) Dog() , Cat(), SuperMan(), WebDesign()

■ 예제 1

HelloJava.java

```
/**
 *
 * 자바 API 주석
 *
 */
public class HelloJava {

    //프로그램의 시작점(JVM이 작동되면
    // 여기를 실행시킵니다.)

    public static void main(String[] args) {

        //콘솔창에 출력하는 코드

        System.out.print(33);

    } //main() 메서드 끝

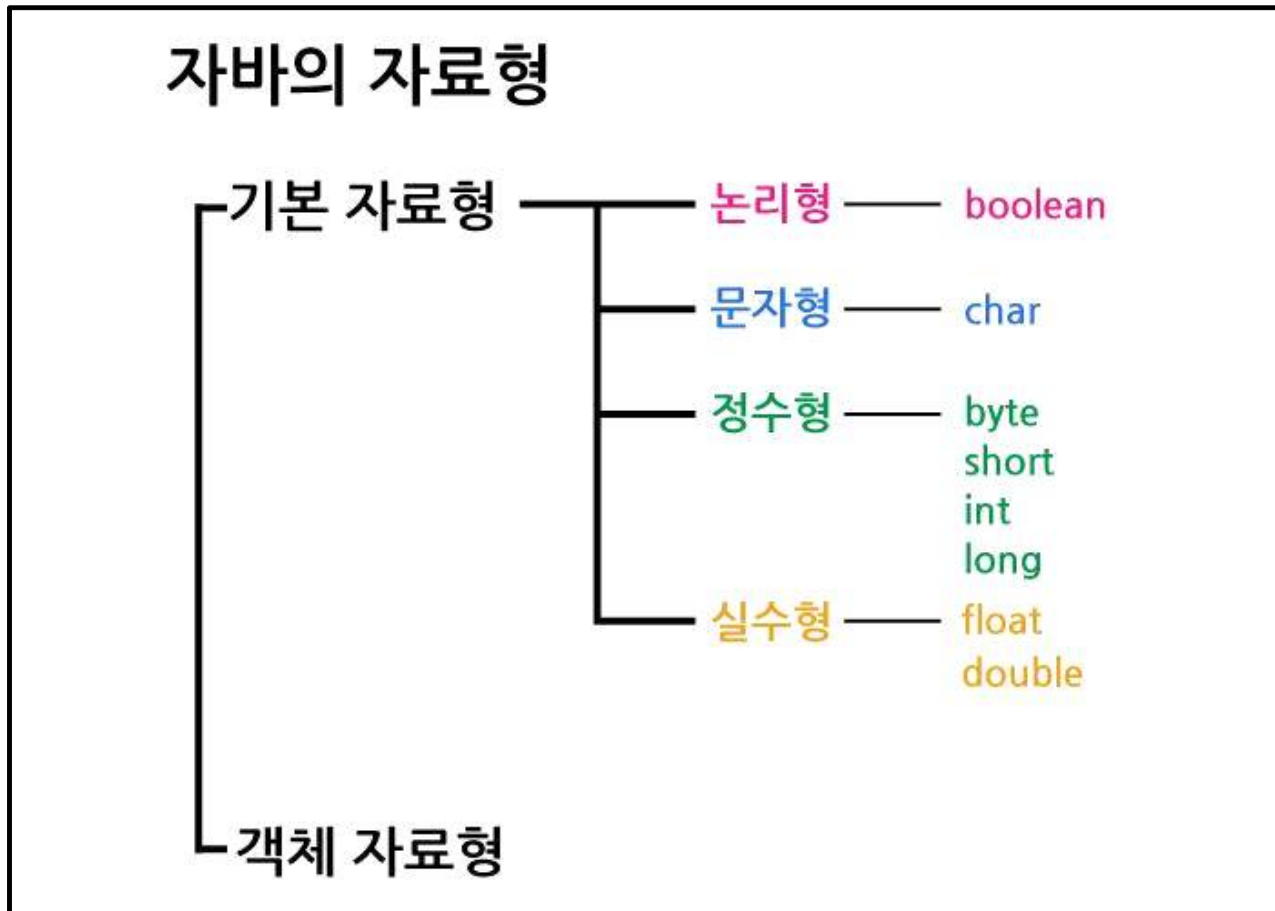
    //보라색은 '키워드'입니다.
    // 자바언어안에 필요한 이름들입니다.

    //이런이름의 변수를 만들 수 없습니다.

    // 자바의 한줄 주석(자바스크립트 마찬가지)

    /*
    * 자바의 여러줄 주석
    *
    */
}
```

9. 자료형



■ 논리형 (boolean)

- 저장 공간의 크기 : 1 bit(jvm에 따라 다름)
- 표현범위 : true, false (기본값 : false)

(다른 언어에서 사용되는 0 과 1은 사용 될 수 없음)

예) `boolean b = true;`

■ 문자형 (char)

- 저장 공간 : 2byte
- 2byte의 유니코드 값

※ 유니코드(Unicode)는 전세계의 모든 문자를 컴퓨터에서 일관되게 표현하고
다룰 수 있도록 설계된 산업 표준

- 내부적으로 정수 처리

```
예) char c1 = 'A';  
     char c2 = 65;  
     char c3 = '\u0042';
```

■ 정수형

- 모두 기본값은 0
- byte : 1byte 크기의 -128 ~ 127의 표현 범위
- short : 2byte 크기의 -32,768 ~ 32,767 표현 범위
- int : 4byte 크기의 -2,147,483,648 ~ 2,147,483,647 표현 범위
- long : 8byte 크기의 -9,223,372,036,854,775,808 ~ ...807 표현 범위

■ 실수형

- float : 4byte 크기의 -3.4E38(3.4*10의 38승) ~ + 3.4E38
- double : 8byte 크기의 -1.7E308(1.7*10의 308승) ~ +1.7E308

■ 예제2

TypeEx1.java

```
public class TypeEx1 {  
  
    //main메서드  
    public static void main(String[] args) {  
  
        //자바의 자료형은 크게 2가지  
  
        //1. 기본자료형
```

```

// 1) boolean : 논리형

//키워드 : boolean
//크기 : 상관없음
//범위 : true, false값
//기본값 : false

// 변수는 선언, 대입

//변수는 선언하고 반드시 한번이상
//대입해야 합니다.

//1번째 대입하는 행위를 초기화라고 합니다.

//선언할때 앞에 자료형을 써줍니다.

//변수의 선언

//자료형 변수명
boolean b;

//자료의 대입
b = true;
//변수 b에 true라는 값으로 초기화

//변수 b안에 뭐가 들었는지 확인
//콘솔에 b값을 출력
System.out.print(b);

b = false;
b = true;

//2. 객체자료형

} //main() 메서드 끝

} //TypeEx1 클래스 끝

```

■ 예제3

TypeEx2.java

```
public class TypeEx2 {

    //실행되는 곳
    public static void main(String[] args) {

        //2. 문자형(한 글자)

        // 키워드 : char (')
        // 크기 : 2byte (0~65535) : 유니코드
        // 범위 : 0 ~ 65535(음수가 없습니다.)
        //      (내부적으로는 정수처리)

        //초기값 : '\u0000' <----

        //char형 변수 c를 선언
        char c;

        //char형변수 c을 1000으로 초기화
        c = 1000;

        //char형 변수 c에 '박'을 대입
        c = '박';

        // 역슬레시u 한다음에 뒤에 16진수 4자리
        c = '\uAD65';

        System.out.print(c);

    } //main() end

} //TypeEx2 클래스 end
```

■ 예제4

TypeEx3.java

```
public class TypeEx3 {

    //JVM 실행시 호출하는 메서드
    public static void main(String[] args) {
```

```

// 정수형

// 1. byte : 1byte
// 2. short : 2byte
// 3. int (자바의 기본정수) 4byte
// 4. long : 8byte


// 키워드 : byte
// 공간 : 1byte (
// 범위 : -128~ -1, 0 , 1 ~ 127
// 초기값 : 0


//아~ byte형 변수 b를 선언
byte b;

b = -128;

System.out.println(b);


// 키워드 : short
// 크기 : 2byte
// 범위 : -32768 ~ 32767
// 초기값 : 0


//short형 변수 s를 선언하고
//200으로 초기화합니다.
short s = 200;

System.out.println(s);


//short s; 한번 선언된 변수는 다시 선언불가

s = 32767;


// 키워드 : int (자바의 기본정수)
// 크기 : 4byte
// 범위 : -2147483648 ~ 2147483647
// 초기값 : 0


//int형 변수 i를 선언하고 45555값으로 초기화
int i = 45555;

System.out.println(i);


//키워드 : long

```

```

//크기 : 8byte
//범위 : -9,223,372,036,854,775,808 ~ ...807
//초기값 : 0L

//long형임을 나타내기 위해서 숫자끝에 L을

//long형 변수 l을 선언하고 long형숫자 1L을 대입
//(초기화)
long l = 1L;

System.out.println(l);

//i = 1L; 1L은 long형숫자 int형변수에
// 들어가지 못합니다.

i = 1;

} //main() 메서드 end

} //TypeEx3 클래스 end

```

■ 예제5

TypeEx4.java

```

public class TypeEx4 {

    public static void main(String[] args) {

        // 실수

        //1 float
        //2 double (기본)

        //키워드 : float
        //크기 : 4byte
        //범위 : 얼마어마합니다.
        //초기값 : 0.0f

        //키워드 : double
        //크기 : 8byte
        //범위 : 엄청나요
        //초기값 : 0.0
    }
}

```

```
//double형 변수 d를 선언하고  
//실수 3.14로 초기화  
double d = 3.14;
```

```
System.out.println(d);
```

```
float f = 3.14F;
```

```
char c = '\t';
```

```
System.out.print(c+0);
```

```
System.out.println(5);
```

```
}// main() 메서드 end
```

```
}//TypeEx4 클래스 end
```

예약어

abstract	assert	boolean	break	byte	case	catch
char	class	const	continue	default	do	double
else	enum	extends	false	final	finally	float
for	goto	if	implements	import	instanceof	int
interface	long	native	new	null	package	private
protected	public	return	short	static	strictfp	super
switch	synchronized	this	try	void	while	

기본자료형의 종류

자료형	키워드	크 기	기본값	표현 범위
논리형	boolaen	1bit(jvm마다 다름)	false	true, false
문자형	char	2byte	\u0000	0 ~ 65,535(양수만)
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	9223372036854775807 ~ -9223372036854775808
실수형	float	4byte	0.0f	-3.4E38 ~ +3.4E38
	double	8byte	0.0	-1.7E308 ~ +1.7E308

기본자료형의 크기

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

형변환

종 류	설 명	코딩 예
프로모션	더 큰 자료형으로 변환(자동) 정보의 손실 없음	short a, b; a = b = 10; int c = a + b;
디모션	더 작은 자료형으로 변환(명시) 정보의 손실 가능성이 있음	int c = 0; short s = 10; c = (int)(10+3.5f);

1. 연산자의 종류와 우선순위

종 류	연산자	우선순위
최우선 연산자	. , [], ()	1
단항 연산자	!, ~, +/-, ++/--, (cast)	2
산술 연산자	+, -, *, /, %	3
시프트 연산자	<<, >>, >>>	4
관계 연산자	>, <, >=, <=, ==, !=	5
비트 연산자	&, ^,	6
논리 연산자	&&,	7
조건(삼항) 연산자	조건 ? 항1 : 항2	8
배정 대입 연산자	=, +=, -=, *=, /=, %=, <<=, >>=, ^=, &=, !=	9
후위형 증감 연산자	++/--	10
순차 연산자	,	11

2. 최우선 연산자

■ .(period) 연산자

1) 특정 범위(객체나 클래스) 내에 속해 있는 멤버를 지칭할 때 사용함

ex) `System.out.println("test");`

■ [](대괄호) 연산자

1) 배열 참조 연산자

2) 자료형이나 클래스와 함께 사용되어 해당 변수나 객체가 배열로 선언됨 을 알리는 역할

ex) `String[] arr = {"AA", "BB", "CC"};`

■ ()(괄호) 연산자

1) 특정 연산자들을 묶어서 먼저 처리할 수 있도록 만들어주는 연산자

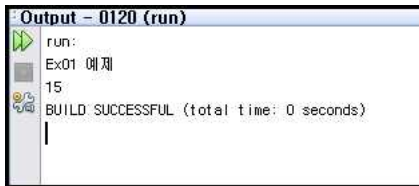
ex) `int 5 * (3+2);`

■ 예제 1

예제 1

```
public class Ex01 {  
    public static void main(String[] args) {  
        System.out.println("Ex01 예제");  
        //. 은 특정 범위의 하위 멤버에 접근할 때 사용함  
  
        int[] iArry = {1, 2, 3};  
        //{ }는 해당 변수나 객체가 배열로 선언됨  
  
        int i = 3 * 5 + (4 - 2) / 3;  
        //( )는 특정 연산자들을 묶어 먼저 처리할 수 있도록 만들어주는 연산자  
  
        System.out.println(i);  
    }  
}
```

결과화면



예제1 : 최우선 연산자에 대한 예제

3. 단항 연산자

■ !(논리부정) 연산자

1) 논리 자료형의 데이터 값을 부정하는 연산자

ex) `boolean bool = false;`

`boolean bool2 = !bool;`

■ ~(비트 부정) 연산자

1) 비트값으로 존재하는 모든 자료들에 대해 부정의 값을 취할 수 있는 연산자

2) 단, `boolean`, `float`, `double`형은 `~` 연산자를 사용할 수 없음

3) `byte`, `short`, `char`, `int`형은 ‘`~`’연산 결과후 `int`, `long`형에만 답을 수 있음

ex) `byte b = 120; int i = ~b;`

4) `long`형은 ‘`~`’ 연산 후 `long`형에만 답을 수 있음

ex) `long l = 120L; long l = ~l;`

■ +/- (양수, 음수 판별) 연산자

1) 양수, 음수 판별해주는 연산자(+ 생략 가능)

ex) `int i = -120;`

■ ++/-- (선위증감) 연산자

1) 특정 변수의 값을 하나 증가시키거나 하나 감소시키는 연산자

ex) `int a = 4; int b = ++a;`

2) 후위 연산자와 우선순위의 차이가 존재함

■ 예제

예제 2

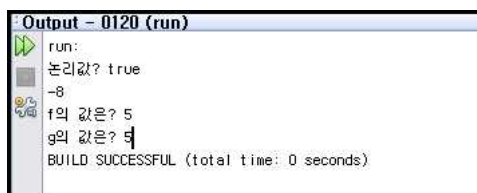
```
public class Ex02 {
    public static void main(String[] args) {
        boolean bool = true;
        bool = !bool;
        //! 는 논리 자료형의 데이터 값을 부정함
        System.out.println("논리값? " + !(4<3));

        byte b = 7;
        int i = ~b;
        // ~의 경우 byte, short, char, int형은 '~'연산 결과후 int, long형에만 담을 수 있음
        System.out.println(i);
        int j = -120;
        // +/- 부호 연산자

        int f = 4;
        int g = ++f;

        System.out.println("f의 값은? " + f);
        System.out.println("g의 값은? " + g);
    }
}
```

결과화면



```
run:
논리값? true
-8
f의 값은? 5
g의 값은? 5
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. 산술 연산자

■ +, -, *, / , % 연산자

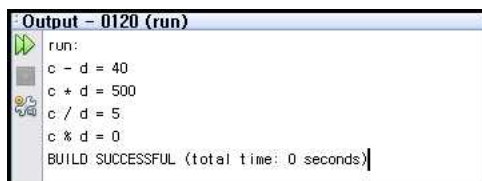
- 1) byte, short, char, int 자료형 사이의 연산에서는 결과가 int임
- 2) long, float, double 자료형이 연산되면 큰 자료형으로 결과가 결정 됨
- 3) '/' 몫, '%' 나머지 값(자바스크립트와는 다름)

■ 예제

예제 3

```
public class Ex03 {  
    public static void main(String[] args) {  
        byte a = 20;  
        byte b = 30;  
        //byte c = a + b; byte, short 연산은 연산후 int값에 대입시켜야 함  
        int c = a + b;  
        int d = 10;  
  
        int e = c - d;  
        System.out.println("c - d = " + e);  
  
        e = c * d;  
        System.out.println("c * d = " + e);  
  
        e = c / d;  
        System.out.println("c / d = " + e);  
  
        e = c % d;  
        System.out.println("c % d = " + e);  
    }  
}
```

결과화면



5. 시프트 연산자

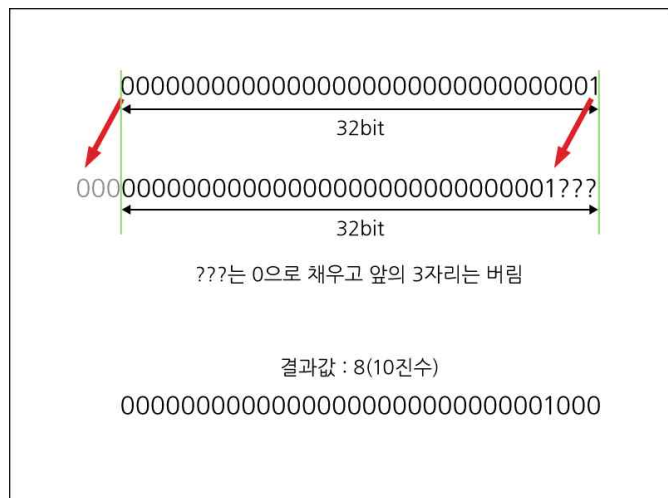
■ 대상 변수의 값을 2진 비트로 바꾼 후 특정 비트 수만큼 이동시켜 원하는 부분의 비트 데이터를 얻어 내는 연산자

■ <<(left shift) 연산자

- 1) 대상 변수값을 2진 비트로 바꾼 후 왼쪽으로 특정 비트 수만큼 이동

2) 빈자리는 0값으로 채움

ex) $1 \ll 3$

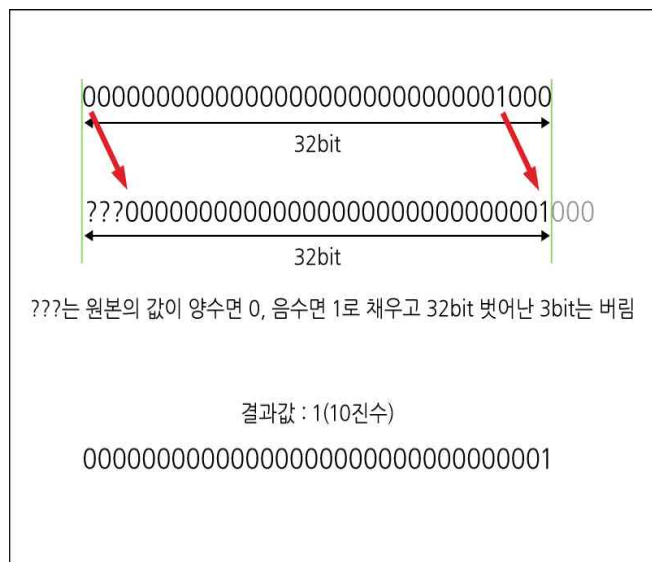


■ >>(right shift) 연산자

1) 대상 변수값을 2진 비트로 바꾼 후 왼쪽으로 특정 비트 수만큼 이동

2) 빈자리는 0값으로 채움

ex) $8 \gg 3$



■ >>>(unsigned right shift) 연산자

1) ' >>' 와 기본적으로 같음

2) 그러나 원본데이터가 음수일 경우에도 빈 비트를 0으로 채움

ex) $-3 \gg 3 = -1$

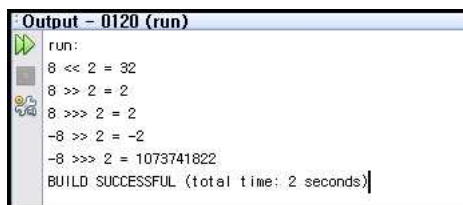
$-3 \ggg 3 = 536870911$

■ 예제

예제 4

```
public class Ex04 {  
    public static void main(String[] args) {  
  
        System.out.println("8 << 2 = " + (8 << 2)); //8 * 2의 2승과 같다  
        System.out.println("8 >> 2 = " + (8 >> 2)); //8 / 2의 2승과 같다  
        System.out.println("8 >>> 2 = " + (8 >>> 2));  
  
        //>> 와 >>> 의 차이점  
        System.out.println("-8 >> 2 = " + (-8 >> 2));  
        System.out.println("-8 >>> 2 = " + (-8 >>> 2));  
  
    }  
}
```

결과화면



```
Output - 0120 (run)  
run:  
8 << 2 = 32  
8 >> 2 = 2  
8 >>> 2 = 2  
-8 >> 2 = -2  
-8 >>> 2 = 1073741822  
BUILD SUCCESSFUL (total time: 2 seconds)
```

예제4 : 시프트 연산자에 대한 예제

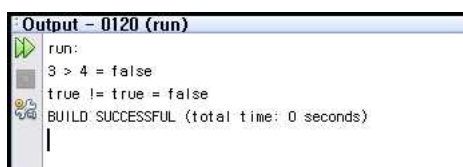
6. 관계 연산자

- <, >, <=, >= (비교 관계) 연산자
- ==, != (항등 관계) 연산자
- 예제

예제 5

```
public class Ex05 {  
    public static void main(String[] args) {  
        System.out.println("3 > 4 = " + (3 > 4));  
        System.out.println("true != true = " + (true != true));  
    }  
}
```

결과화면



```
Output - 0120 (run)  
run:  
3 > 4 = false  
true != true = false  
BUILD SUCCESSFUL (total time: 0 seconds)
```

예제5 : 관계 연산자에 대한 예제

7. 비트 연산자

■ &(AND) 연산자

값1	값2	결 과
0	0	0
1	0	0
0	1	0
1	1	1

■ |(OR) 연산자

값1	값2	결 과
0	0	0
1	0	1
0	1	1
1	1	1

■ ^(Exclusive OR) 연산자

값1	값2	결 과
0	0	0
1	0	1
0	1	1
1	1	0

예제 6

```
public class Ex06 {  
  
    public static void main(String[] args) {  
        int a = 10;  
        int b = 7;  
  
        int c = a & b; // AND  
        int d = a | b; // OR  
        int e = a ^ b;  
  
        System.out.println ("a AND b : " + c);  
        System.out.println ("a OR b : " + d);  
        System.out.println ("a xor(배타적or) b : " + e);  
    }  
}
```

결과화면


```
Output - 0120 (run)
run:
a AND b : 2
a OR b : 15
a xor(배타적or) b : 8
BUILD SUCCESSFUL (total time: 1 second)
```

8. 논리 연산자

■ &와 && 연산자의 차이

예제 7

```
public class Ex07 {

    public static void main(String[] args) {
        int a = 0, b =0;
        if(++a > 0 || b++ > 0){

            System.out.println("이것은 실행이 됩니다.");

        }

        System.out.println("a의 값:"+a); // 실행이 되어서 연산이 된 결과 값
        System.out.println("b의 값:"+b); // 실행이 되지 않아서 연산이 안된 결과 값.

    }
}
```

결과화면

```
run:
이것은 실행이 됩니다.
a의 값:1
b의 값:0
BUILD SUCCESSFUL (total time: 1 second)
```

9. 삼항 연산자

■ 조건항 ? 항1(true일때) : 항2(false일때)

예제 8

```
public class Ex08 {

    public static void main(String[] args) {

        int a =20, b=30, max;
```

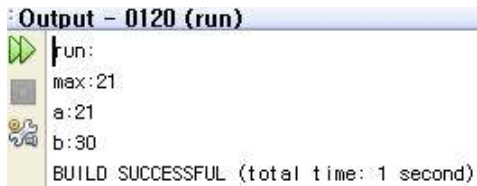
```

        max = a < b ? ++a : ++b;

        System.out.println("max:"+max);
        System.out.println("a:"+a);
        System.out.println("b:"+b);
    }
}

```

결과화면



```

run:
max:21
a:21
b:30
BUILD SUCCESSFUL (total time: 1 second)

```

10. 배정 대입 연산자

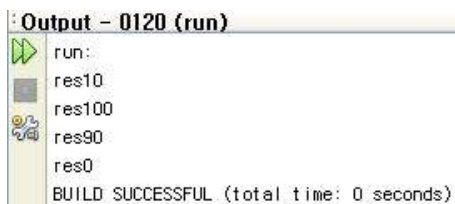
예제 9

```

public class Ex09 {
    public static void main(String[] args) {
        int a =10;
        int res=0;
        res += a;
        System.out.println("res"+res);
        res *= a;
        System.out.println("res"+res);
        res -= a;
        System.out.println("res"+res);
        res %=a;
        System.out.println("res"+res);
    }
}

```

결과화면

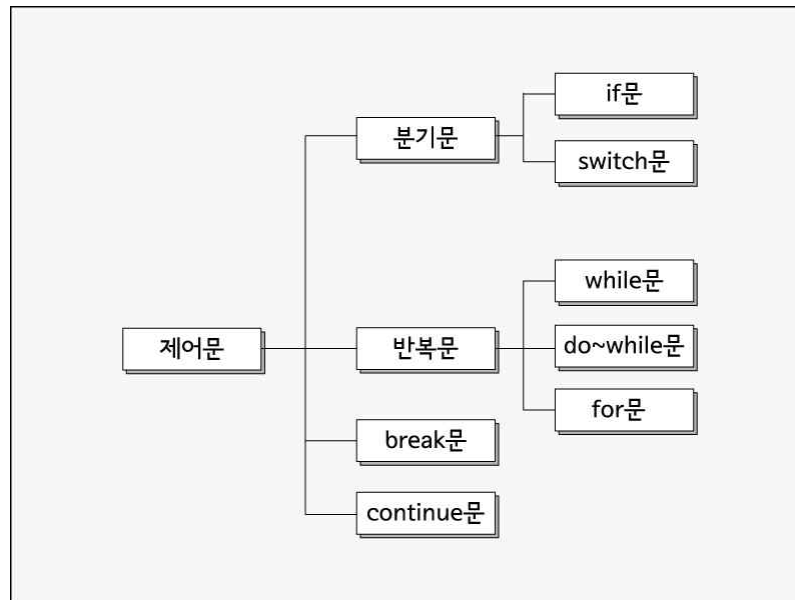


```

run:
res10
res100
res90
res0
BUILD SUCCESSFUL (total time: 0 seconds)

```

2. 제어문



■ 분기문(비교문)

주어진 조건의 결과에 따라 실행문장을 달리하여 전혀 다른 결과를 얻기 위해 사용하는 제어문

■ 단순 if 문

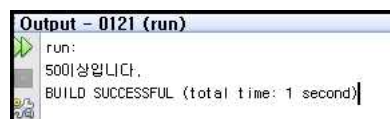
```
if(조건식) {  
    조건에 만족할 때 수행할 문장  
}
```

1) 수행할 문장이 단행일 경우 {} 생략 가능

예제 1

```
public class Ex01 {  
    public static void main(String[] args) {  
        int a = 51;  
        String str = "";  
        if(a >= 50) {  
            str = "50이상";  
        }  
        System.out.println(str + "입니다.");  
    }  
}
```

결과화면



■ 단일 if ~ else 문

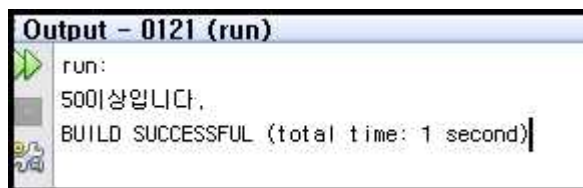
- 1) 조건을 만족할 때와 만족하지 않을 때, 각각 서로 다른 수행문장이 실행
- 2) 수행할 문장이 단행일 경우 {} 생략 가능

```
if(조건식) {  
    조건값이 true일때 수행할 문장  
}  
else {  
    조건값이 false일때 수행할 문장  
}
```

예제 2

```
public class Ex02 {  
    public static void main(String[] args) {  
        int a = 51;  
        String str = "";  
  
        if(a > 50) {  
            str = "50이상";  
        }  
        else {  
            str = "50미만";  
        }  
  
        System.out.println(str + "입니다.");  
    }  
}
```

결과화면



■ 다중 if문

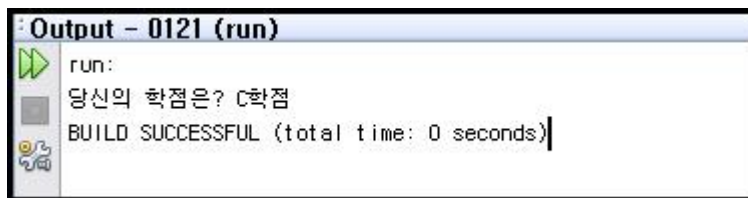
- 1) 조건 비교가 한 가지로 충분하지 않을 경우에 사용
- 2) 예 : 91점이상 A학점, 90점이하 81점이상 B학점...

```
if(조건식) {  
    조건값이 true일때 수행할 문장  
}  
else if(조건식2) {  
    조건식1이 false이고, 조건식2가 true일 때 수행하는 문장  
}  
else {  
    조건식1, 조건식2가 모두 false일 때 수행할 문장  
}
```

예제 3

```
public class Ex03 {  
    public static void main(String[] args) {  
        int a = 80;  
        String grade = "";  
  
        if(a > 90) {  
            grade = "A학점";  
        }else if(a > 80) {  
            grade = "B학점";  
        }else if(a > 70) {  
            grade = "C학점";  
        }else if(a > 60) {  
            grade = "D학점";  
        }else {  
            grade = "F학점";  
        }  
  
        System.out.println("당신의 학점은? " + grade);  
    }  
}
```

결과화면



■ switch문

- 1) 다양한 처리문을 두고 조건값에 의해 하나의 처리문이나 여러 개의 처리문을 한 번에 수행하는 데 유용한 분기(비교)문
- 2) if문은 조건값이 boolean형, switch문은 정수형(byte, short, int)과 문자형(char)
- 3) long, 문자열, boolean, float, double형 사용 불가

```
switch(인자값) {  
    case 비교값 1 :  
        수행할 문장; break;  
    case 비교값 2 :  
        수행할 문장; break;  
    case 비교값 3 :  
        수행할 문장; break;  
    default : 수행할 문장;  
}
```

4) break문은 하나의 조건값 마다 하나의 수행문만 필요할 경우 사용

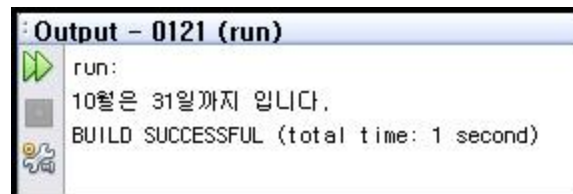
break문이 없을 경우 다음 break문을 만날 때 까지 모든 수행문 실행

5) 인자값과 비교값 1,2,30이 모두 일치하지 않는다면 default 실행

예제 4

```
public class Ex04 {  
    public static void main(String[] args) {  
        int month = 10;  
        String res;  
        switch(month) {  
            case 1:  
            case 3:  
            case 5:  
            case 7:  
            case 8:  
            case 10:  
            case 12: res = "31"; break;  
            case 4:  
            case 6:  
            case 9:  
            case 11: res = "30"; break;  
            case 2: res = "29"; break;  
            default : res ="월이 아닙니다.";  
        }  
        System.out.println(month + "월은 " + res + "일까지 입니다.");  
    }  
}
```

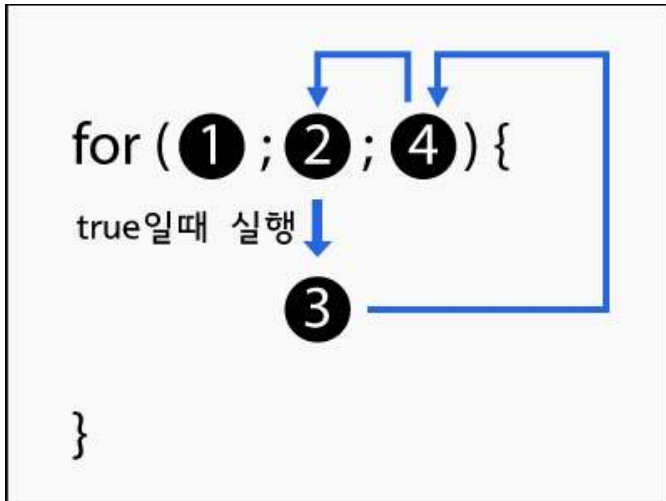
결과화면



■ 반복문

반복문은 하나의 수행문 또는 중괄호({}) 범위 내에 있는 모든 수행문을 반복할 수 있게 하는 문장

■ for문



1) 특정한 수행문을 원하는 만큼 반복적으로 처리할 때 사용하는 제어문

```
for(초기식 ; 조건식 ; 증감식) {  
    수행할 문장;  
    .....  
}
```

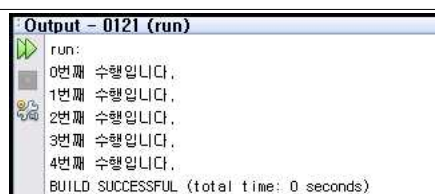
2) 초기식 : 가장 먼저 한번만 수행됨

3) 조건식 : 초기식 다음으로 수행되고 루프(loop)가 돌 때마다 한번씩 비교하여 반복문을 수행할 지 반복문을 벗어날 지 결정함

4) 증감식 : 루프를 수행할 때마다 조건식에 비교하기 전에 항상 수행하며, 변수값을 증가 또는 감소시켜 루프를 원활하게 수행시킴

예제 5

```
public class Ex05 {  
    public static void main(String[] args) {  
        for ( int i = 0 ; i < 5 ; i++) {  
            System.out.println(i + "번째 수행입니다.");  
        }  
    }  
}
```



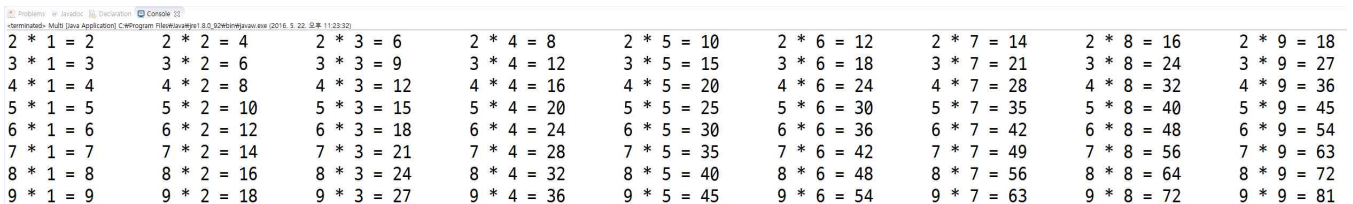
■ 다중 for문

1) for문 안에 for문이 있는 경우

```
for(초기식 ; 조건식 ; 증감식) {  
    for(초기식 ; 조건식 ; 증감식) {  
        수행할 문장;....  
    }  
    수행할 문장;....  
}
```

예제 6

```
public class Ex06 {  
    public static void main(String[] args) {  
        for(int i = 2; i < 10 ; i++) {  
            for(int j = 1 ; j < 10 ; j++) {  
                System.out.print(i + " * " + j + " = " + (i*j) + "\t");  
            }  
            System.out.println("");  
        }  
    }  
}
```



2 * 1 = 2	2 * 2 = 4	2 * 3 = 6	2 * 4 = 8	2 * 5 = 10	2 * 6 = 12	2 * 7 = 14	2 * 8 = 16	2 * 9 = 18
3 * 1 = 3	3 * 2 = 6	3 * 3 = 9	3 * 4 = 12	3 * 5 = 15	3 * 6 = 18	3 * 7 = 21	3 * 8 = 24	3 * 9 = 27
4 * 1 = 4	4 * 2 = 8	4 * 3 = 12	4 * 4 = 16	4 * 5 = 20	4 * 6 = 24	4 * 7 = 28	4 * 8 = 32	4 * 9 = 36
5 * 1 = 5	5 * 2 = 10	5 * 3 = 15	5 * 4 = 20	5 * 5 = 25	5 * 6 = 30	5 * 7 = 35	5 * 8 = 40	5 * 9 = 45
6 * 1 = 6	6 * 2 = 12	6 * 3 = 18	6 * 4 = 24	6 * 5 = 30	6 * 6 = 36	6 * 7 = 42	6 * 8 = 48	6 * 9 = 54
7 * 1 = 7	7 * 2 = 14	7 * 3 = 21	7 * 4 = 28	7 * 5 = 35	7 * 6 = 42	7 * 7 = 49	7 * 8 = 56	7 * 9 = 63
8 * 1 = 8	8 * 2 = 16	8 * 3 = 24	8 * 4 = 32	8 * 5 = 40	8 * 6 = 48	8 * 7 = 56	8 * 8 = 64	8 * 9 = 72
9 * 1 = 9	9 * 2 = 18	9 * 3 = 27	9 * 4 = 36	9 * 5 = 45	9 * 6 = 54	9 * 7 = 63	9 * 8 = 72	9 * 9 = 81

■ while문

1) for문과 유사함. 특정 명령들을 반복적으로 처리

2) for문은 반복횟수를 정확히 알고 있는 경우에 많이 사용하고 while문은 반복횟수를 정확히 알지 못할 때 사용함

3) 무한루프에 빠질 수 있으므로 주의

```
while(조건식) {  
    수행할 문장;....  
}
```

예제 7

```
public class Ex07 {
```

```

public static void main(String[] args) {

    int i = 0;
    while(i<10){
        System.out.println("i 의 값 "+i);
        i++;
    }
}

=====

public class Ex09_1 {
public static void main(String[] args) {

    int i = 0;
    while(i<31){

        if(i % 7 == 0){
            System.out.println("");
        }else {
            System.out.print(+i+" \t");

        }
        i++;
    }
    System.out.println("");
}
}

```

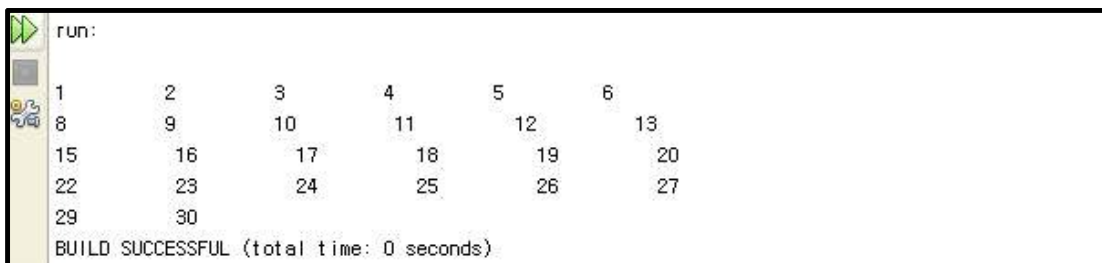
결과화면



```

run:
i 의 값 0
i 의 값 1
i 의 값 2
i 의 값 3
i 의 값 4
i 의 값 5
i 의 값 6
i 의 값 7
i 의 값 8
i 의 값 9
BUILD SUCCESSFUL (total time: 0 seconds)

```



```

run:
1      2      3      4      5      6
8      9      10     11     12     13
15     16     17     18     19     20
22     23     24     25     26     27
29     30
BUILD SUCCESSFUL (total time: 0 seconds)

```

■ do~while문

- 1) while문과는 달리 do{}를 무조건 한 번 수행한 후 값을 비교
- 2) while(조건식) 뒤에 ;(세미콜론)을 생략하면 오류 발생

```
do {  
    수행할 문장;....  
}while(조건식);
```

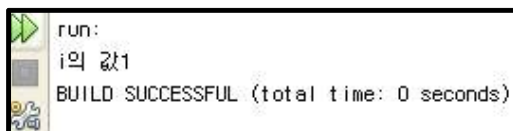
예제 10

```
public class Ex10 {  
    public static void main(String[] args) {  
        int i =0;  
        do{  
            System.out.println("i의 값"++i);  
        }while(i==0);  
    }  
}
```

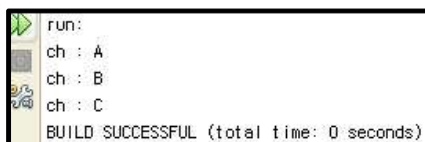
```
=====
```

```
public class Ex10_1 {  
    public static void main(String[] args) {  
        char ch ='A';  
        do{  
            System.out.println("ch : "+ch);  
            ch++;  
        }while(ch <= 'C');  
    }  
}
```

결과화면



```
run:  
i의 값1  
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
run:  
ch : A  
ch : B  
ch : C  
BUILD SUCCESSFUL (total time: 0 seconds)
```

■ break문

- 1) 반복문(for, while), switch문에서 쓰임

- 2) 강제적으로 해당 반복문을 빠져나갈 때 쓰이는 제어문
- 3) break문을 만날 때 가장 가까운 반복문 한 개를 탈출함

```
for(초기식 ; 조건식 ; 증감식) {
    수행할 문장...
    break;
}
```

예제 11

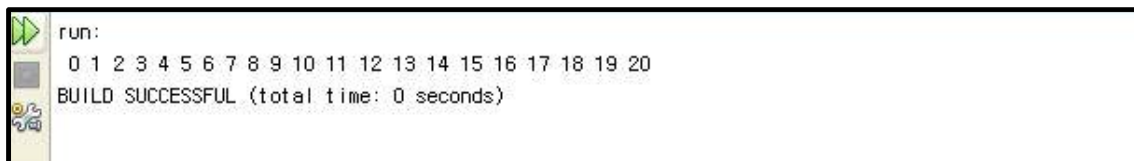
```
public class Ex11 {
    public static void main(String[] args) {
        int i =0;
        while(i<100){

            System.out.print(" "+i);

            if(i == 20) break;

            i++;
        }
        System.out.println("");
    }
}
```

결과화면



■ break label문

- 1) break label문은 break문과 달리 여러개의 반복문을 빠져나갈 때 사용함

레이블명:

```
for(초기식 ; 조건식 ; 증감식) {
    for(초기식 ; 조건식 ; 증감식) {
        수행문;
        break 레이블명; //레이블명 있는 곳까지 빠져 나감
    }
}
```

예제 12

```

public class Ex12 {

    public static void main(String[] args) {
        One: for(int i =0; i< 5; i++){
            Two: for(int j=0; j<3; j++){
                if(j==2) break One;
                System.out.print(j+" x "+i+" ");
            }
            System.out.println("");
        }
    }
}

```



```

run:
0 x 0 1 x 0 BUILD SUCCESSFUL (total time: 0 seconds)

```

■ continue문

1) 반복문에서 사용되며 어느 특정 문장이나 여러 문장을 건너뛰고자 할 때 사용됨

```

for(초기식 ; 조건식 ; 증감식) {
    continue;
    수행문;
}

```

예제 13

```


public class Ex13 {
    public static void main(String[] args) {

        for(int i=0; i<=20; i++){

            if(i % 2 == 0) continue;
            System.out.print(i+" "); // 홀수만 출력 한다.
        }
        System.out.println("");
    }
}

```

결과화면



```

run:
1 3 5 7 9 11 13 15 17 19
BUILD SUCCESSFUL (total time: 0 seconds)

```

■ continue label문

1) continue label문은 레이블까지 수행시점이 이동함

레이블명:

```
for(초기식 ; 조건식 ; 증감식) {  
    for(초기식 ; 조건식 ; 증감식) {  
        수행문;  
        continue 레이블명;  
    }  
}
```

예제 14

```
public class Ex14 {  
    public static void main(String[] args) {  
        F1 : for(int i=0; i< 5; i++){  
            F2: for(int j=0; j<3; j++){  
                if(j == 1){  
                    continue F1;  
                }  
                System.out.print(j+" x "+i+" ");  
            }  
            System.out.println("");  
        }  
    }  
}
```

결과화면



■ 향상된 for문

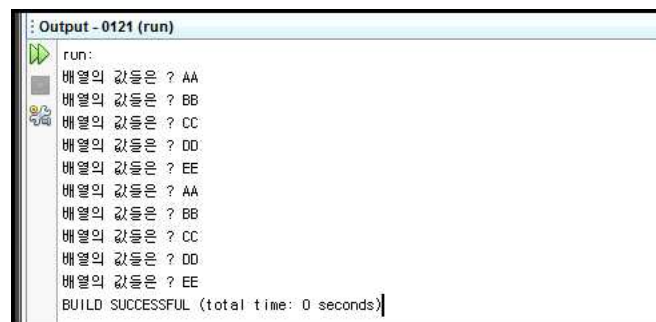
- 1) 자바 5.0에 추가된 for문
- 2) for문의 반복 카운트 변수가 배열의 인덱스일 경우 사용
- 3) 카운트 변수를 제어해야 할 경우에는 사용 할 수 없음

```
for(배열내부의 변수자료형 변수 : 배열이름) {  
    수행할 문장;  
}
```

예제 15

```
public class Ex15 {  
    public static void main(String[] args) {  
        String[] arr = {"AA", "BB", "CC", "DD", "EE"};  
  
        for(String s : arr) {  
            System.out.println("배열의 값들은 ? " + s);  
        }  
  
        for(int i = 0 ; i < arr.length ; i++) {  
            System.out.println("배열의 값들은 ? " + arr[i]);  
        }  
    }  
}
```

결과화면



1. 객체지향언어의 역사

- 초창기에는 주로 과학실험이나 미사일 발사실험과 같은 모의실험(Simulation)을 목적으로 사용됨
- 과학자들은 모의실험을 위해 실제 세계와 유사한 가상 세계를 컴퓨터 속에 구현하고자 노력하였으며, 이러한 노력은 객체지향이론을 탄생시킴
- 객체지향이론의 기본 개념은 '실제 세계는 사물(객체)로 이루어져 있으며, 발생하는 모든 사건들은 사물간의 상호작용이다.'라는 것
- 실제 사물의 속성과 기능을 분석한 다음, 데이터(변수)와 함수로 정의함으로써 실제 세계를 컴퓨터 속에 옮겨 놓은 듯한 가상세계를 구현하고 이 가상세계에서 모의실험을 함으로써 많은 시간과 비용을 절약
- 객체지향이론은 상속, 캡슐화, 추상화 개념을 중심으로 점차 구체적으로 발전되었으며, 1960년대 중반에 객체지향이론을 프로그래밍언어에 적용한 Simula라는 최초의 객체지향언어가 탄생
- FORTRAN이나 COBOL과 같은 절차적 언어들이 주류를 이루었으며, 객체지향언어는 널리 사용되지 못하고 있었음
- 1980년대 중반에 C++을 비롯하여 많은 수의 객체지향언어가 발표되면서, 객체지향언어가 본격적으로 개발자들의 관심을 끌기 시작하였지만 사용자 층이 넓지 못했음
- 프로그램의 규모가 점점 커지고 사용자들의 요구가 빠르게 변화해가는 상황을 절차적언어로는 극복하기 어렵다는 한계를 느끼고 객체지향언어를 이용한 개발방법론이 대안으로 떠오르게 되면서 조금씩 입지를 넓혀가고 있었음
- 자바가 1995년에 발표되고 1990년대 말에 인터넷의 발전과 함께 크게 유행하면서 객체지향언어는 이제 프로그래밍의 주류가 되었음



오디오 컴포넌트

객체지향 프로그래밍은 **하나**의 웹서비스(혹은 어플리케이션)를 하나의 파일로 만드는 것이 아니라 **기능**에 따라 **나뉘**서 개발하는 것을 말합니다.

학사관리 프로그램을 만든다고 하면,

현실에도 학생이 있고, 교사가 있고, 수업이 있고, 클래스가 있듯이

컴퓨터 프로그램에서도 학생들 객체가 있고, 교사객체가 있고, 수업객체가 있고, 클래스 객체가 있는 겁니다.

The M4 Carbine is a compact version of the M16A2 rifle, with a collapsible stock, a flat-top upper receiver accessory rail and a detachable handguard/rail aperture sight assembly. The M4 enables a soldier operating in close quarters to engage targets at extended range with accurate, lethal fire.

Caliber: 5.56 mm **Weight:** 7.5 lbs (loaded weight with sling & one magazine) **Max effective range:** 600 m (area target) 500 m (point target)



46

2. 객체지향언어

- 코드의 재사용성이 높다.
 - 새로운 코드를 작성할 때 기존의 코드를 이용하여 쉽게 작성할 수 있다.
- 코드의 관리가 용이하다.
 - 코드간의 관계를 이용해서 적은 노력으로 쉽게 코드를 변경할 수 있다.
- 신뢰성이 높은 프로그래밍을 가능하게 한다.
 - 제어자와 메서드를 이용해서 데이터를 보호하고 올바른 값을 유지하도록 하며, 코드의 중복을 제거하여 코드의 불일치로 인한 오동작을 방지할 수 있다.
- 객체지향의 가장 큰 장점 : ‘코드의 재사용성이 높고 유지보수가 용이하다.’
- 3대 특징 : 캡슐화(은닉화), 상속, 다형성

캡슐화란?

캡슐화를 사용하면 블랙박스기능을 구현할 수 있음
해당코드를 다른 사람이 그것의 작동 방법을 알 필요가 없게 한다는 뜻
사용자는 기능을 어떻게 호출하고 그 결과가 나타날 것인가만 알면 되는 것임

상속이란?

상속은 기존의 객체의 특징을 확장하는 새로운 객체를 만들 수 있는 것을 의미함

다형성이란?

상속에서의 다형성과 메서드 오버로딩의 다형성으로 나눌 수 있음

3. 클래스와 객체의 정의와 용도

- 클래스란 객체를 정의해 놓은 것(클래스는 객체의 설계도)
- 클래스는 객체를 생성하는데 사용됨
- 객체의 정의 : 실제로 존재하는 것 혹은 사물 또는 개념
- 객체의 용도 : 객체가 가지고 있는 기능과 속성에 따라 다름

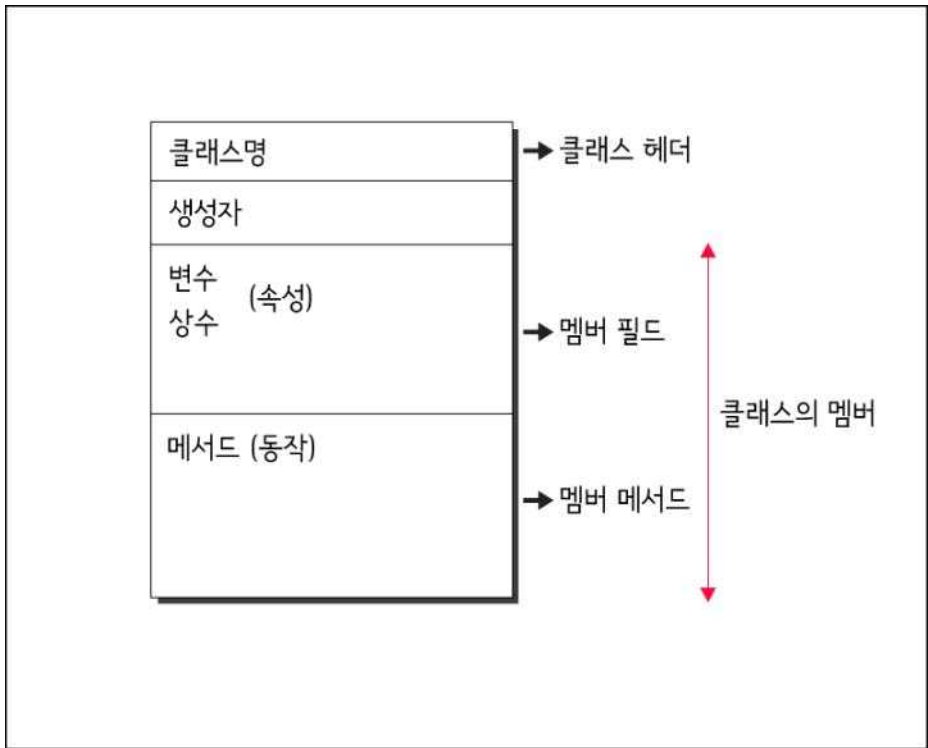
유형의 객체	책상, 의자, 자동차, TV...
무형의 객체	수학공식, 컴퓨터 에러...

■ 클래스와 객체와의 관계 : 객체의 설계도가 클래스

클래스	객체
제품 설계도	제품
와플틀	와플
건축 설계도	건축물

■ 우리가 TV를 보기 위해서는, TV(객체)가 필요한 것이지 TV설계도(클래스)가 필요한 것은 아니며, TV설계도(클래스)는 단지 TV라는 제품(객체)을 만드는 데만 사용될 뿐입니다. 그리고 TV설계도를 통해 TV가 만들어진 후에야 사용할 수 있는 것입니다.

4. 클래스의 구조



■ 클래스 헤더

[접근제한자(public, default)] [클래스 종류(final, abstract)] class 클래스명

1) 접근제한자 : 현재 클래스를 생성하고 사용하는 데 있어 제한을 둔다는 의미

public, 아무것도 쓰지 않는 방법(default)

2) 클래스 종류 : final, abstract(추상) 등 어떤 클래스인지 알리는 수식어

생략시 일반 클래스

■ 멤버 필드

- 1) 변수, 상수
- 2) 객체가 만들어질 때 특징적인 속성을 담아두는 것
- 3) static 변수, 상수와 instance 변수, 상수로 나뉨

■ 멤버 메서드

- 1) 특정한 일을 수행하는 행위, 다시 말해 동작을 나타냄
- 2) static 메서드, instance 메서드로 나뉨

5. 클래스의 정의

예제 1

```
public class Book {  
    String name;  
    String writer;  
    int price;  
    int nowPage = 1;  
    int page;  
    String isbn;  
  
    public void nextPage() {  
        nowPage++;  
    }  
  
    public void previousPage() {  
        nowPage--;  
    }  
}
```

6. 객체의 생성

■ 객체 선언

```
Book book;
```

■ 객체 생성

```
book = new Book();
```

1) new라는 키워드를 통해 무조건 메모리의 공간을 할당받고, 생성자를 호출하여 생성

7. 메모리 구조



■ stack : 메서드 호출, 또는 관련 정보를 주고받는 일 처리를 기록하는 곳

원통형 구조 : FIFO(first in, last out) 구조

■ heap : 클래스 영역에 존재하는 클래스 중 new라는 키워드로 객체를 생성시 로딩되는 곳

■ static : static으로 선언된 변수, 메서드 등이 로딩되는 곳으로 프로그램 실행시

가장 먼저 인식되는 부분

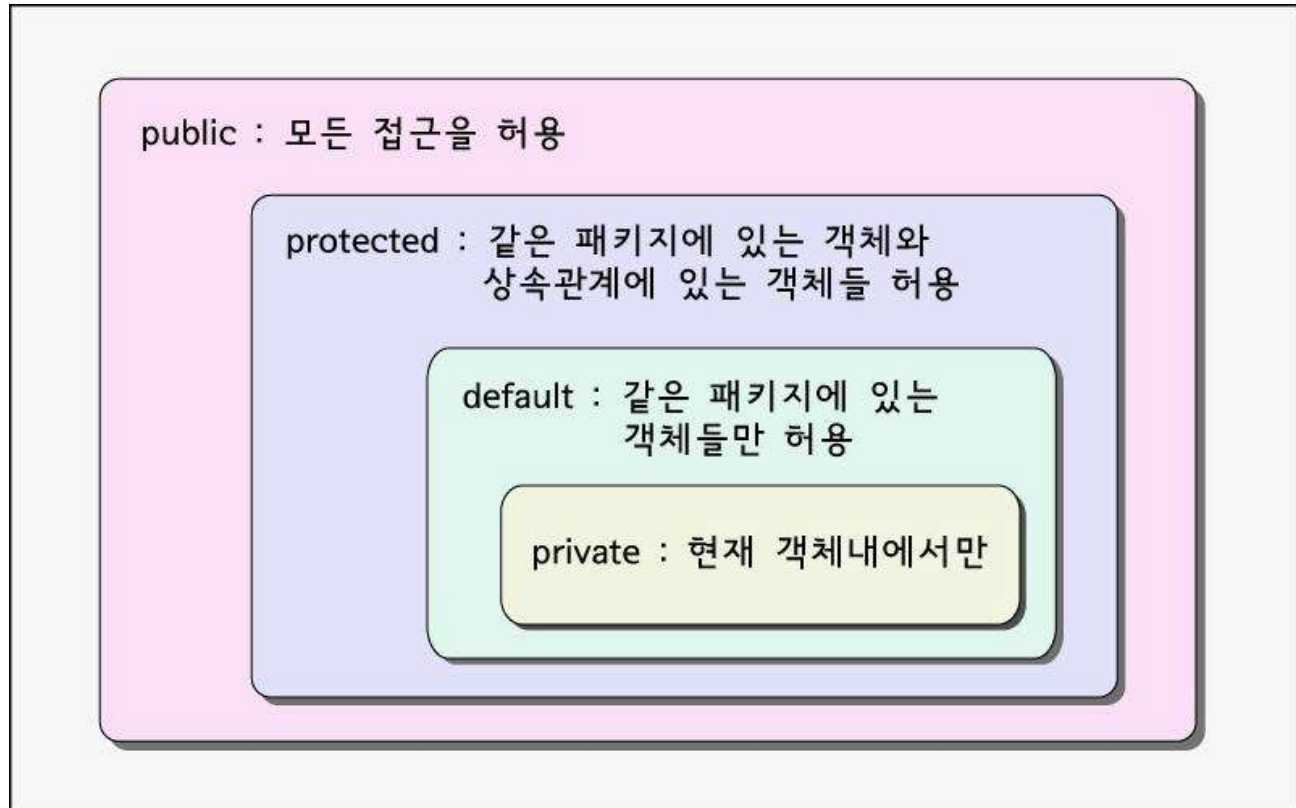
■ 클래스 영역 : import문을 통해 필요로 하는 클래스들이 로드됨

메서드들은 메서드 영역에 저장, 상수들은 상수풀에 저장

■ 네이티브 메서드 스택

C언어와 같은 다른 언어의 기능을 잠시 빌려서 사용할 때 JNI라는 기술을 이용하는데, 여기서 사용되는 네이티브 메서드들이 바이트 코드로 변환되면서 사용되고 기록됨

8. 접근제한자



■ 멤버 접근법

1) 객체를 참조 하는 참조변수(reference)를 통해 ‘.’을 사용하여 해당 객체의 메서드나 변수, 즉 멤버들을 접근 한다.


2) 형식 : 참조변수.멤버필드 / 참조변수.멤버메소드

ex) tv.name; / tv.turnOn();

예제 2

```
public class Ex02 {  
  
    int a;  
  
    public void ex02Met(){  
        System.out.println("메서드가 실행이 됩니다.");  
    }  
}  
=====
```

```
public class Ex02_Test {  
  
    public static void main(String[] args) {  
  
        Ex02 ref = new Ex02();  
        ref.a = 10;  
        int c = 10;  
        System.out.println("힙안의 a의 값"+ref.a);  
        System.out.println("c의값 "+c);  
  
        ref.ex02Met();  
    }  
}
```

 run:
힙안의 a의 값10
c의값 10
메서드가 실행이 됩니다.
BUILD SUCCESSFUL (total time: 0 seconds)

5. 캡슐화

■ 캡슐화

여러개의 처리 과정을 하나의 부품처럼 사용하므로 객체간의 이식성이 높고
자료 또는 내부 수정 작업을 했을 때에도 외부객체에서는 이것을 인식 하지 못하는
독립적인 장점이 있다.

[기술적인 면] -> 사용법을 제공을 받아서 사용 하는 면

즉 캡슐화로 된 자료를 사용자에게 내부적인 접근을 허용하지 않더라도 사용자에게
사용방법을 알려주고 사용 하게 해주는 것이다.

■ 정보은닉

정보은닉은 캡슐화의 장점에 속하는 것이다.

외부에서 “참조형변수.멤버필드 “ 의 형식을 차단하고, 접근이 용이한 메서드를 통해 결과를 받게 하는 것이다.

예제 1

```
public class Ex03 {

    private int pay;

    public int getPay() {
        return pay;
    }

    public void setPay(int pay,String pass) {
        if(pass.equals("1234"))
            this.pay = pay;
    }

}

=====

public class Ex03_Test {

    public static void main(String[] args) {

        Ex03 ref = new Ex03();

        // ref.pay = 10000;

        // System.out.println("내 계좌를 마음대로 "+ref.pay);

        ref.setPay(1000,"1234");
        int aa = ref.getPay();
        System.out.println("계좌에 입금한 금액"+aa);
    }

}
```

```
run:
  힙안의 a의 값10
  c의값 10
  메서드가 실행이 됩니다.
  BUILD SUCCESSFUL (total time: 0 seconds)
```

■ 변수



5. 멤버변수

instance 변수 와 static 변수(클래스 변수)

■ instance 멤버필드

객체가 new라는 키워드로 메모리 영역 중 heap 영역에 생성될 때 객체 안에 같이 생성되어 객체의 속성을 저장 하는 곳

```
new TestC(); -----> public class TestC{    int a;  String color;  }
```

■ static 멤버필드

하나의 클래스로 여러 개의 객체가 생성 될 때 단 하나만 생성된다. - 공유

예제 1

```
public class Ex04 {
    int pay;
    String id;
    static int box;
}

=====
public class Ex04_Test {

    public static void main(String[] args) {

        Ex04 ref = new Ex04();
        Ex04 ref2 = new Ex04();

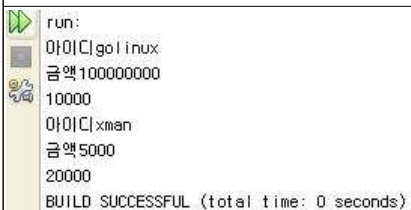
        ref.id="golinux";
        ref.pay=100000000;

        System.out.println("아이디"+ref.id);
        System.out.println("금액"+ref.pay);
        Ex04.box = 10000;
        System.out.println(""+Ex04.box);

        ref2.id="xman";
        ref2.pay=5000;

        System.out.println("아이디"+ref2.id);
        System.out.println("금액"+ref2.pay);
        Ex04.box = 20000;
        System.out.println(""+Ex04.box);

    }
}
```



```
run:
아이디golinux
금액100000000
10000
아이디xman
금액5000
20000
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. 메서드

■ 메서드 - 객체가 할 수 있는 동작

instance 메서드 와 static 메서드(클래스 메서드)로 분류

■ 메서드의 구성과 정의

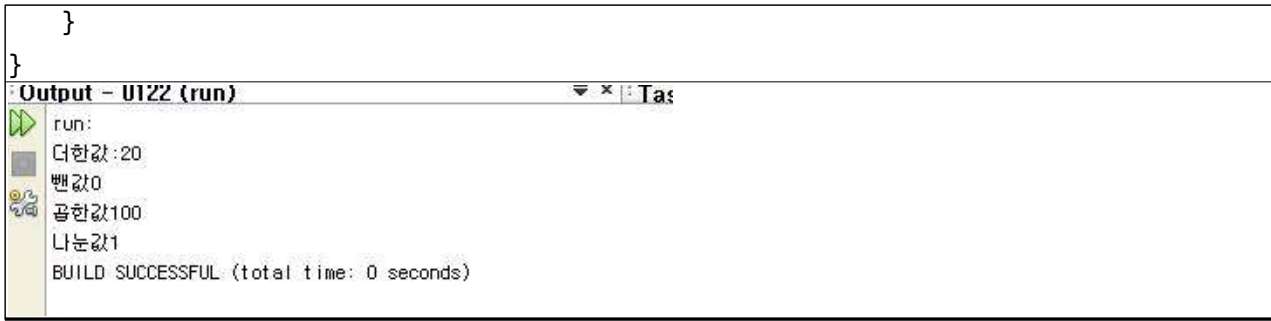
```
[접근제한] 반환형 메서드명(자료형인자1,.....){  
    수행문1; 수행문2; .....  
}
```

- 1) 접근제한 : public , protected, default, private
- 2) 반환형 : 메서드가 종료되고 나서 호출한 곳으로 반환하는 값의 자료형이다.
반환형이 없을 경우 void
- 3) 메서드명 : 사용자 정의 이름
- 4) 인자 : 메서드를 호출 할 때에 반드시 인자의 자료형과 수가 일치
- 5) 수행문 : 실행문

예제 1

```
public class Ex05 {  
  
    public int sum(int i,int j){  
        return i + j;  
    }  
    public int sub(int i,int j){  
        return i - j;  
    }  
    public int multi(int i, int j){  
        return j * i;  
    }  
    public int divi(int i, int j){  
        return j / i;  
    }  
}  
=====
```

```
public class Ex05_Ex {  
    public static void main(String[] args) {  
        Ex05 ref = new Ex05();  
        int i =10;  
        int j =10;  
        System.out.println("더한값:"+ref.sum(i, j));  
        System.out.println("뺀값"+ref.sub(i, j));  
        System.out.println("곱한값"+ref.multi(i, j));  
        System.out.println("나눈값"+ref.divi(i, j));  
    }  
}
```



1. 인자전달 방식

- 메서드를 호출할 때 여러 개의 값을 전달하고 또는 때에 따라 return문으로 값을 다시 받기도 함
- 메서드를 사용함에 있어 인자 전달 방식은 매우 중요함
- 전달방식에는 값 호출(Call By Value)와 참조 호출(Call By reference)가 있음

2. 값 호출(Call By Value)

- 기본 자료형을 가지고 메서드를 호출할 때 사용되는 인자 전달 방식
- 인자가 직접 전달되는 것이 아니라 값의 복사본이 전달됨

예제 1

```

package ex1;

public class Ex01 {

    public static void main(String[] args) {
        int i =10;

        Ex01.call(i);

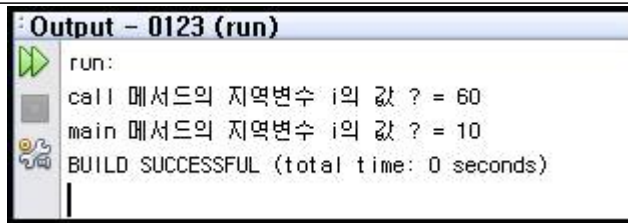
        System.out.println("main 메서드의 지역변수 i의 값 ? = " + i);
    }

    public static void call(int i) {
        i += 50;
        System.out.println("call 메서드의 지역변수 i의 값 ? = " + i);
    }

}

```

결과화면



3. 참조 호출(Call By Reference)

- 객체 또는 배열의 인자 전달 방식
- 참조 호출의 경우 참조하고 있는 주소가 전달되므로 같은 객체를 가리키고 있음

예제 2

```
package ex2;

public class Ex02 {
    int a = 10;
}
```

예제 2

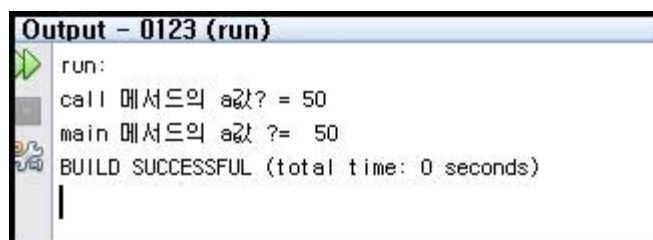
```
package ex2;

public class Ex02Test {
    public static void main(String[] args) {
        Ex02 reference1 = new Ex02();
        Ex02Test.call(reference1);

        System.out.println("main 메서드의 a값 ? = " + reference1.a);
    }

    public static void call(Ex02 reference2) {
        reference2.a = 50;
        System.out.println("call 메서드의 a값 ? = " + reference2.a);
    }
}
```

결과화면



4. Varargs(Variable Arguments)

- 인자의 갯수를 동적으로 바꿔 메서드를 호출 가능하게 함
- 같은 자료형의 경우에만 가능

예제 3

```
package ex3;

public class Ex03 {

    public static void test(String ... n) {
        for(String s : n) {
            System.out.println("인자로 받은 값들 ? = " + s);
        }
    }

    public static void main(String[] args) {
        Ex03.test("하나", "둘");

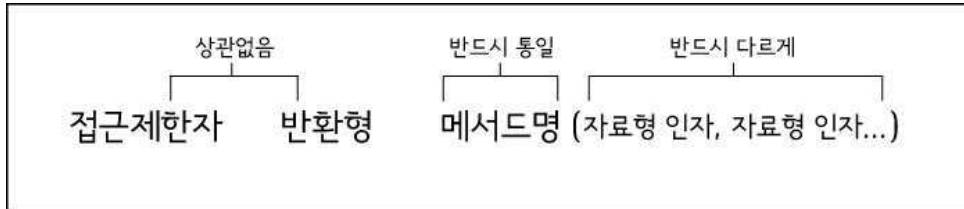
        Ex03.test("A", "B", "C", "D");
    }
}
```

결과화면



5. 메서드 오버로딩(Method Overloading)

- 하나의 클래스에서 같은 이름을 가진 메서드가 여러 개 정의되는 것을 말함
- 같은 이름의 메서드에 인자가 다름
- 인자가 다르다는 것은 개수가 다르거나, 자료형이 다르거나, 인수의 순서가 다른 것



- 같은 목적으로 비슷한 동작을 수행하는 메서드들을 모아 이름을 같게 만들어 일관성 유지

void	<code>println()</code>	Terminates the current line by writing the line separator string.
void	<code>println(boolean x)</code>	Prints a boolean and then terminate the line.
void	<code>println(char x)</code>	Prints a character and then terminate the line.
void	<code>println(char[] x)</code>	Prints an array of characters and then terminate the line.
void	<code>println(double x)</code>	Prints a double and then terminate the line.
void	<code>println(float x)</code>	Prints a float and then terminate the line.
void	<code>println(int x)</code>	Prints an integer and then terminate the line.
void	<code>println(long x)</code>	Prints a long and then terminate the line.
void	<code>println(Object x)</code>	Prints an Object and then terminate the line.
void	<code>println(String x)</code>	Prints a String and then terminate the line.

예제 4

```

package ex4;

public class Ex04 {

    int add(int a, int b) {
        System.out.print("int add(int a, int b) - ");
        return a+b;
    }

    long add(int a, long b) {
        System.out.print("long add(int a, long b) - ");
        return a+b;
    }

    long add(long a, int b) {
        System.out.print("long add(long a, int b) - ");
    }
  
```



```

        return a+b;
    }

    long add(long a, long b) {
        System.out.print("long add(long a, long b) - ");
        return a+b;
    }

    int add(int[] a) {          // 배열의 모든 요소의 합을 결과로 돌려준다.
        System.out.print("int add(int[] a) - ");
        int result = 0;
        for(int i=0; i < a.length;i++) {
            result += a[i];
        }
        return result;
    }
}

```

예제 4

```

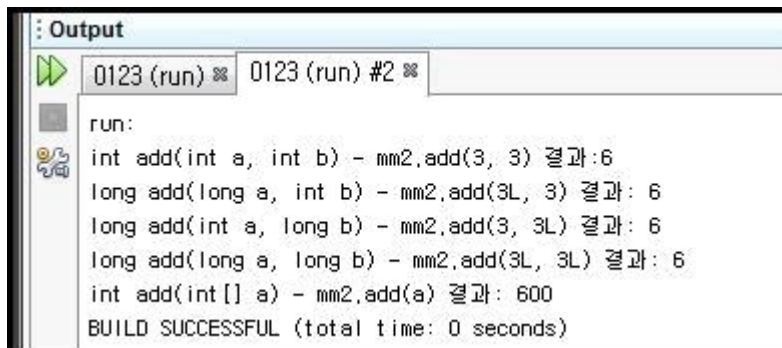
package ex4;

public class Ex04Test {
    public static void main(String[] args) {
        Ex04 ex = new Ex04();
        System.out.println("mm2.add(3, 3) 결과:" + ex.add(3,3));
        System.out.println("mm2.add(3L, 3) 결과: " + ex.add(3L,3));
        System.out.println("mm2.add(3, 3L) 결과: " + ex.add(3,3L));
        System.out.println("mm2.add(3L, 3L) 결과: " + ex.add(3L,3L));

        int[] a = {100, 200, 300};
        System.out.println("mm2.add(a) 결과: " + ex.add(a));
    }
}

```

결과화면



6. 생성자

- 메모리 내에 객체가 생성될 때 호출되어 객체의 구조를 인식하게 하고 생성되는 멤버변수 들을

초기화하는 데 목적을 둬

- 생성자명은 클래스명과 같아야 하고, return type를 정의하지 말아야 함

```
[접근제한] 생성자명([자료형 인자1, 자료형 인자2....]) {  
    수행문1;  
    수행문2;  
    ....  
}
```

- 프로그래머가 어떠한 생성자도 정의하지 않았을 경우 컴파일러가 default 생성자를 자동으로 정의해 줌
- default 생성자 : 인자가 없는 생성자
- 생성자의 필요성

생성자 - 초기화 작업.

예제 6

```
package ex6;  
  
public class MyClass1 {  
  
    private String name;  
  
    public MyClass1(String n){  
        name = n;  
    }  
  
    public void setName(String n){  
  
        name = n;  
    }  
  
    public String getName(){  
  
        return name;  
    }  
}  
=====
```

```
package ex6;  
  
public class MyClassTest {  
    public static void main(String[] args) {
```

```

// MyClass1 mc1 = new MyClass1();
MyClass1 mc1 = new MyClass1("www.hanbiteni.co.kr");
// mc1.setName("www.hanbiteni.co.kr");
System.out.println("Value:"+ mc1.getName());

}
}

```

■ 생성자 접근제한의 의미

- 1) 생성자의 접근제한을 둘 경우 해당 객체를 생성할 수 있는 접근권한을 가짐
- 2) 클래스의 접근제한이 public으로 정의 되어도 생성자를 private로 정의 하면
클래스 내부에서만 접근 가능 하다.
- 3) 만약 protected로 정의되는 클래스는 상속관계의 객체들만 생성할 수 있음

■ 생성자 오버로딩

- 1) 객체를 생성할 수 있는 방법을 여러 개 제공하는 것과 같음

예제 6

```

package ex6;

public class Ex1 {
    int age;
    String name,subject;
    //오버로딩
    public Ex1(String name, String subject){
        this.name=name;
        this.subject=subject;
        age=20;
    }

    public Ex1(String subject,int age,String name){
        this.name=name;
        this.age=age;
        this.subject=subject;
    }

    // get
    public int getAge(){

        return this.age;
    }
}

```

```

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

    public String getSubject(){

        return this.subject;
    }
}

package ex6;
public class Ex1Test {
    public static void main(String[] args) {

        Ex1 ref = new Ex1("김길동","방송연예과");
        Ex1 ref1 = new Ex1("항공운항과", 30,"이슬이");

        System.out.print("신입생 정보 : 이름:"+ref.getName());
        System.out.println(" 나이:"+ref.getAge()+" 학과: "+ref.getSubject());

        System.out.println("");
        System.out.print("신입생 정보 : 이름:"+ref1.getName());
        System.out.println(" 나이:"+ref1.getAge()+" 학과: "+ref1.getSubject());

    }
}

```

7. this와 this()

- this란 특정 객체 내에서 자신이 생성되었을 때의 주소 값 변수
- 객체의 주소는 생성 전까지는 모르기 때문에 객체 생성 후 자신의 주소로 대체됨

예제 1

```

package ex7;

public class Ex1 {

    String name,jumin,tel;

    public Ex1(){
        this.name="Guest";
    }
}

```

```

        this.jumin="000000-00000000";
        tel="000-0000-0000";

    }
    public Ex1(String name){
        this();
        this.name=name;
    }
    public Ex1(String name,String jumin){
        this(name);
        this.jumin=jumin;
    }

    public Ex1(String name,String jumin,String tel){
        this(name,jumin);
        this.tel=tel;
    }

    public String getJumin() {
        return jumin;
    }

    public String getName() {
        return name;
    }

    public String getTel() {
        return tel;
    }
}

```

```

package ex7;

public class Ex1Test {

    public static void main(String[] args) {

        Ex1 ref = new Ex1();
        // 이름만 입력 했을 경우
        Ex1 ref1 =new Ex1("name");
        System.out.println("Name"+ref1.getName());
        System.out.println("TEL"+ref1.getTel());
        System.out.println("Jumin"+ref1.getJumin());

    }
}

```

```
Output - 0123 (run)
run:
Namename
TEL000-0000-0000
Jumin0000000-00000000
BUILD SUCCESSFUL (total time: 0 seconds)
```

- `this()`는 현재 객체의 생성자를 의미함
- 생성자 안에서 오버로딩 된 다른 생성자를 호출할 경우에 `this()`라는 키워드로 호출함

8. static 예약어

- 메서드나 멤버변수에 정의할 수 있으며 지역변수나 클래스에는 정의 불가
- `static` 키워드를 사용하면 `static` 변수(클래스 변수), `static` 메서드라 지칭함
- 멤버변수나 멤버 메서드는 해당 객체가 생성될 때 heap 영역에 존재함
- `static`으로 선언된 필드, 메서드는 `static` 영역에 유일하게 만들어지면서 모든 객체들이 사용할 수 있는 공유개념을 가지기 때문
- 객체를 생성하지 않더라도 사용할 수 있음
- 클래스명.변수명 or 클래스명.메서드명 으로 접근 가능

static 메서드 선언법

```
[접근제한자] static 반환형 메서드명([자료형 인자들...]) {}
```

static 필드 선언법

```
[접근제한자] static 자료형 변수명;
```

■ 생성자 오버로딩

```
public class Book {  
  
    public Book() {  
        this(true);  
        System.out.println("★");  
    }  
  
    public Book(String name) {  
        System.out.println("■");  
    }  
  
    public Book(boolean b) {  
        this(3);  
        System.out.println("●");  
    }  
  
    public Book(Book b) {  
        this(false);  
        System.out.println("◆");  
    }  
  
    public Book(int price) {  
        this("Book");  
        System.out.println("▲");  
    }  
  
    public static void main(String[] args) {  
        Book book = new Book();  
        Book book2 = new Book(book);  
    }  
}
```

■ 메서드 오버로딩

```
public class TestMethod {  
    public void test() {  
    }  
  
    public String test(int i) {  
        return "s";  
    }  
  
    private int test() {  
        return 3;  
    }  
  
    boolean test(String s) {  
        return true;  
    }  
  
    void test(boolean b) {  
  
    }  
  
    public void test(int s) {  
  
    }  
  
    public void test(int i, int j) {  
  
    }  
}
```