*literal 과 메모리크기1

유(프기반)

	리터럴		수를 줄인다	32bit		2의 보수법에 따라 표기
	100		4byte 2진수로 변환	24bit	8bit	: 0x00000064
정수			↓	000 - - -	:01100100	→
			4byte 크기를 맞춘다			
	100L					
	100l		8byte 2진수로 변환	:0x 00 00 00 00 00 00 00 64		

*8byte 메모리에 저장된 100 이다.

*JAVA literal은 4byte/8byte를 표기

		IEEE 754 규약		
	12.375		수를 줄인다	
	12.375 D		8byte 2진수 변환	0x 40 28 c0 00 00 00 00 00
부동소수점	12.375 d			
	12.375 F			
	12.375 f	→	4byte 2진수 변환	0x41460000

---

*literal 과 메모리크기2

		unicode(UTF-16 BE)				
문자	: 'A'	→	2byte 2진수로 변환	→	0x0041	
	'가'	→	2byte 2진수로 변환	→	0xAC00	
논리값	TRUE	→	4byte 2진수로 변환	→	0x 00 00 00 01	*단 논리 배열인 경우 1byte로 변환
	FALSE	→		→	0X 00 00 00 00	TRUE 0x01
						FALSE 0x00
문자열	: "ABC"	→	String 클래스에 정의과 대로	→	0x 00 03 00 41 00 42 00 43	→ UTF-16
			그 형식에 맞춰 변환		0X 00 00 03 41 42 43	→ UTF-8

---

*메모리 크기와 값의 범위

	4bit				
	□□□□	: 0000			
		~	16개	양수만 저장하면 : 0 ~ 15	
		: 1111		양수, 음수 저장 : -8 ~7	

		4bit	-8 ~ +15			
정수	byte	8bit	-128 ~ + 127	4byte 정수 리터럴	약 21억 ~ +21억	ex)100
	chart	16bit	-32768 ~ +32767	→	8byte 정수 리터럴(L & l이 붙음)	약 -900경 ~ + 900경	ex)100L & 100l
	int	32bit	약 -21억 ~ + 21억			
	long	64bit	약 -900경 ~ + 900경	*자바의 리터럴은 크기를 줄인다. 4byte 약 ± 21억		

```
4바이트 또는 8바이트 정수의 최대값과 최소값
자바에서는 각 데이터 유형에 따라 최대/최소 값을 알아볼 수 있도록 특별한 변수를 제공한다.
System.out.println(Integer.MAX_VALUE); // 4바이트로 표현할 수 있는 정수 최대값    2147483647
System.out.println(Integer.MIN_VALUE); // 4바이트로 표현할 수 있는 정수 최소값   -2147483648
System.out.println(Long.MAX_VALUE);    // 8바이트로 표현할 수 있는 정수 최대값    9223372036854775807
System.out.println(Long.MIN_VALUE);    // 8바이트로 표현할 수 있는 정수 최소값   -9223372036854775808
```

부동소수점	32bit(float)	→	유효자리수(10진수의 거위) 7자리까지 숫자가 거의 정상적으로 저장	→	*소수점을 빼고 남은 수의 개수 = 유효자리수	→	3.14f
	single precision : 단정도		정수로 완벽하게 바뀌지 않음. (ex. 2.127)		*지수+부 가수부로 나뉘어서 하나로 합치면 : IEEE754		3.14F
			(2배 : double)				
	64bit(double)		유효자리수 15자리까지 숫자를 거의 정상적으로 저장	→	3.14	3.14D	3.14d
	double precision : 배정도						

정수는 값을 벗어날 경우 오류로 나옴

	부동소수점	4byte	7자리	범위예) e f (123.456f)
		8byte	16자리	[기본 : 여유있게 사용 가능]

```
//## 4byte(float) 부동소수점의 유효자릿수
//소수점을 뺀 후 7자리 숫자까지는 거의 정상적으로 저장된다.
System.out.println(999.9999f);                          999.9999
System.out.println(999999.9f);                          999999.9
System.out.println(9.999999f);                          9.999999
System.out.println("---------------------------");      --------
                                                        9.8765434E8
                                                        9.876543
//유효자릿수가 7자리를 넘어가는 경우 값이 불려져 저장될 수 있다.  9.876543E14
System.out.println(987654321.1234567f);                 --------
System.out.println(9.876543211234567f);                 9.876543211234568E8
System.out.println(987654321123456.7f);                 9.876543211234567
System.out.println("---------------------------");      9.876543211234568E14
                                                        --------
//## 8byte(double) 부동소수점의 유효자릿수                  0.7000000000000001
//소수점을 뺀 후 16자리 숫자까지는 거의 정상적으로 저장된다.
System.out.println(987654321.1234567);
System.out.println(9.876543211234567);
System.out.println(987654321123456.7);
System.out.println("---------------------------");

//유효자릿수가 16자리를 넘어가는 경우 값이 불려져 저장될 수 있다.
System.out.println(987654321.12345678);
System.out.println(9.876543211234567);
System.out.println(987654321123456.78);
System.out.println("---------------------------");
```

---

*문자의 코드

		unicode		utf-8		iso-8859-1		EUC-KR
	: 'A'	0X 00 41	→	0x41	→	0x41	→	x
	: '가'	0x AC 00	→	0xEABO80	→	X	→	-xBAA1
	: '?'	0x003F	→	0x3F	→	0x3F	→	
	'='	0X003D	→	0x3D	→	0x3D	→	
	'CR'							
	Carrage Return	0x000d	→	0x0D	→	0dx0D	→	

---

*줄바꿈 코드

Windows	→	CR LF	→	0D0A		
		1byte / 1byte				
Unix	→	LF	→	0A		
		1bite				

---

*문자 리터럴

: 'A', '가'
: ('\ u0041') = A *유니코드로 입력 -> 키보드로 입력이 안되는 문자를 입력할 때

```
// \ u 뒤에 문자에 부여된 코드 값을 16진수로 적는다.
// u는 소문자로 적어야 한다.
System.out.println('\u0041'); // A
System.out.println('\uac00'); // 가
System.out.println('\uAC00'); // 가 (16진수는 대소문자를 구분하지 않는다)

// \ 바로 뒤에 8진수로 적을 수 있다.
// 단, 0 ~ 377 범위의 8진수만 가능하다.
System.out.println('\101'); // A
System.out.println('\122'); // R
System.out.println('\377'); // ÿ
// 유니코드 값을 직접 적는 경우는
// 보통 키보드로 문자를 직접 입력할 수 없을 때이다.
System.out.println('\u4eba'); // 사람을 뜻하는 '인(人)'이라는 한자의 문자 코드
System.out.println('\u00a9'); // copyright를 뜻하는 '©' 문자 코드
System.out.println('\u03c0'); // 수학의 파이 기호 'π'
System.out.println('\u03a3'); // 수학의 시그마 기호 'Σ'
```

---

*문자 - 코드 - 폰트

				폰트	arial	굴림체	굿서체
	: 'A'	→	0x0041	→	A	A	A
	: '가'	→	0xAC00	→	X	가	>ხ
							↑

1. 운영체제에서 그 문자에 대한 코드값이 공백되어있는 데이블 (ex UTF-8)   폰트 파일에 해당 코드의 문자가 필요한 속력할수 있다.
2. UTF-8에 정의해 폰트 파일이 있어야함.

폰트

Raster 방식	픽셀 단위로 그리는 방식
	(Raster 폰트)
	그림 크기 확대 시 계단 현상 발생
	출력 속도가 빠름
Vector 방식	점과 점을 연결하고 특정 수학식에 따라 계산하여 그린다.
	(True Type 폰트)
	그림 크기 확대 -> 미려하게 출력 가능
	단, 계산이 필요하기때문에 출력 속도가 느리다.

Raster 방식   Vector 방식

비트맵과 Vector 방식

비트맵
Raster 방식
그림 크기 조정 → 계단현상발생

ex) jpg, gif 등

파일 크기가 크다

벡터이미지
Vector 방식
그림크기조정 → 그대로 유지

ex) 캐드, 파워포인트의 클립아트

파일 크기가 작다 (해상도가 떨어진다)
*단, 그림이 복잡하면, 택터 양들이 많아지기 때문에 파일 크기가 높아난다.

---

*Escape 문자

```
//# 이스케이프 문자 (escape character) - 문자 제어 코드
//- 화면에 출력하는 문자가 아니라 문자 출력을 제어하는 문자이다.
//- 제어 문자
// \n  - Line Feed(LF), 0x0a       문자나 문자열 속에 출력을
// \r  - Carriage Return(CR), 0x0d  코드를 삽입할 때 사용한다.
// \f  - Form Feed, 0x0c            주로 \n(LF) 코드만 사용한다.
// \t  - Tab, 0x09       띄어쓰기
// \b  - Backspace, 0x08
// \' - Single Quote, 0x27
// \" - Double Quote, 0x22
// \\ - Backslash, 0x5c
```

		출력
abc\r\nxyz	→	abc
		xyz
abc\n\nxyz	→	abc
		xyz
\f (Form Feed) →	1page	2page
abc\f\xyz →	abc	xyz
abc\b\xyz →	abxyz	

```
// \' - Single Quote, 0x27    : '\''
// \" - Double Quote, 0x22    : "\""
                              : '\c'
// \r  - Carriage Return(CR), 0x0d
커서를 다시 앞으로 옮겨 덮어쓰기
```

---

*변수 선언
메모리를 준비시키는 명령
값을 저장할 메모리

		데이터유형				메모리이름	:			
		↓					byte	b;	b	
		Date Type	byte	bit	정수+음수	변수명	:			1
	정수	byte	1	8	signed					
		short	2	16	signed					
		int	4	32	signed					
primitive type		long	8	64	signed					
자바명시 파일	부동소수점	float	4	32	signed					
		double	8	64	signed					
	논리	boolean	4							
	문자	char	2	16	unsigned					
		문자는 음수개념이 없음								

---

*변수에 값 저장하기(asignment)

변수명 . = 리터럴 :
         . = 변수명 :
         . = 명령문 : → Statement → Expression

Statement   값을 리턴하는 명령문
             명령문이 실행을 완료하면 값을 리턴
             expression

```
int      i;          i =      100;
메모리의 이름들 라고 하자     리터럴   변수값
100
4byte
-4byte를 확보했겠다.
```

변수명   . = 리터럴 ;
        할당 / 배정 / 대입 연산자
        assignment operator

```
int      i;           int      j;
i = 100               j = 100;

i =      100;        j =     i;
                          i의 값을 j의 값에 복사하려
int      s;

         100 + 100           s = 200
s =      i + j;              4byte
i = expression   결과값을 리턴하는 객체
```

---

*r-value 와 l-value

```
x        .=    i + j ;
l-value       r-value
↓             ↓
변수           변수, 리터럴, 표현식(expression)

왼쪽은 변수만 가능 / 계산식 x , 리터럴 x
```

---

*변수 선언

```
int i;
int i = 100;      선언과 동시에 값을 선언
int x,y,z ;       여러 변수를 한번에 선언 가능
int x =100, y 2=20;    여러 변수에 값을 한번에 넣을 수 있다.
```

---

*배열 - 같은 유형의 연속된 메모리들

```
배열의 유형   참조변수          new    만들 메모리의 유형 [개수];
int[]        arr        =     new    int[5];
                          0      1      2      3      4
                        4byte  4byte  4byte  4byte  4byte   <- 배열 인스턴스 (instance) = new 명령을 통해 준비한 메모리
                                        arr
arr -> 배열 레퍼런스
200(배열의 메모리 주소)

4byte              ex)  arr[0] = 100;      0 이라는 주소에 100이라는 값을 넣어라
                        arr[2] = 300;
                        arr[5] = 700;    <- ArrayIndexOutofBoundsException 발생!
                        [0] = index       arr의 값이 범위를 넘어가 오류가 발생!
```

---

*process 와 메모리

```
java.exe         실행         프로세스
↑                            (process)
컴파일러 프로그램
      └ Application s/w       실행중인 프로그램
                             App.sw
```

os   RAM

1. 실행하는 동안 데이터를 보관할 때 사용할 메모리를 달라고 요청

                2.의 락

porcess

3. 사용해서 데이터를 보관

---

*JVM과 메모리

os
                RAM
2.JVM이 사용할 메모리를 확보 → Method Area   크기 세등분으로 할당받은 메모리를 분할
                              JVM Stack   메소드를 실행하는 동안 로컬 변수를 저장하는 용도
        1. 실행 & 요청   Heap   인스턴스(객체)를 저장하는 용도
↓
JVM
                3. 메모리 사용 →
                              변수생성
                              값저장
                              객체 생성
                              바이트코드 로딩
                              etc