* 배열 (Array)

변수를 선언한다는 것은 어떤 종류 어떤 크기의 메모리를 만드는지 정의하는 것이다.

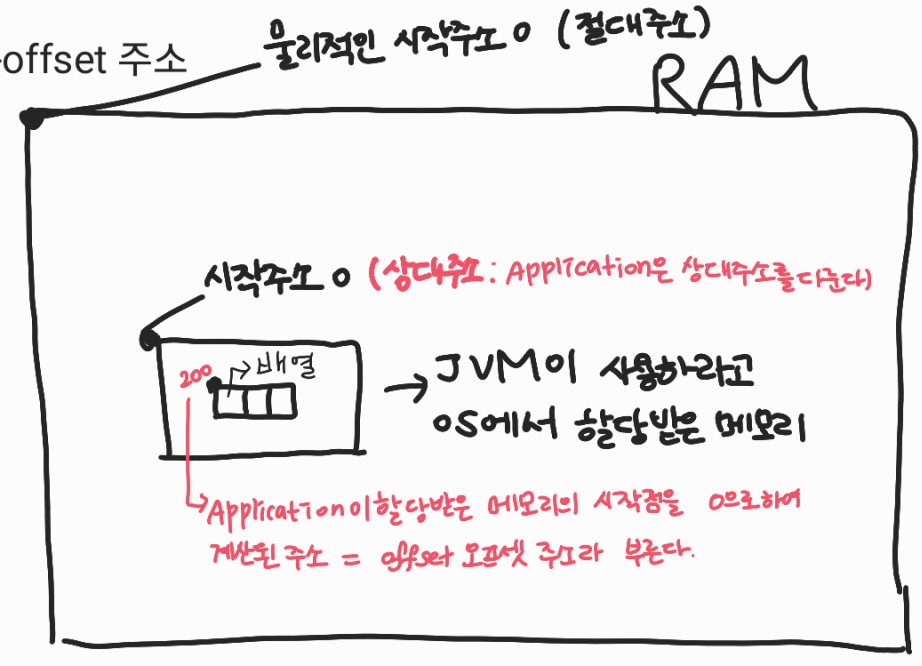
같은 데이터 타입의 메모리를 여러개 선언하는 문법

int[] arr= new int[3];
int[] : 배열 타입을 가리킨다.
arr : 배열의 주소를 저장하는 변수
new : 메모리 준비 명령
int : 데이터 타입
3: 개수

int[3]; = 연속된 메모리 준비 : 배열의 '인스턴스'
arr = 레퍼런스

*배열과 메모리 = 인스턴스와 레퍼런스
arr는 배열 메모리의 offset 주소

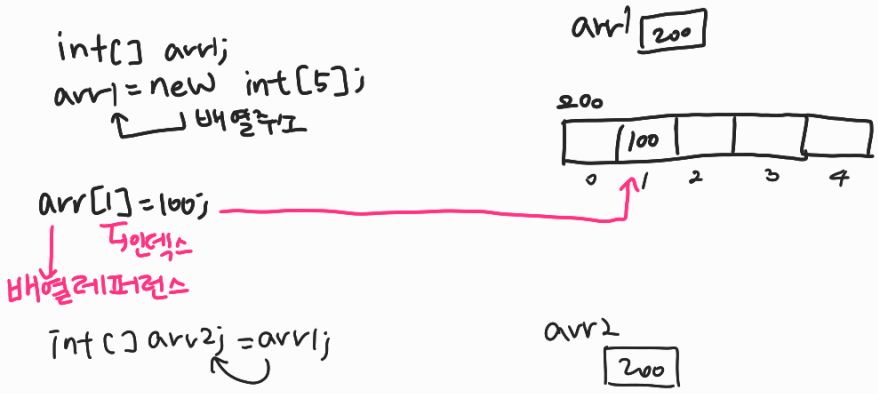
-offset 주소



offset : 메모리의 시작점에서 떨어져있는 위치

위치를 기준으로 몇 번째 인지 따질 때 얼만큼 떨어져있는지를 계산하는 위치가 offset
사용으로 허락받은 메모리로부터 시작점을 0으로 간주하고 몇 번째 떨어졌는지 간주하는 것이 offset 주소
jvm이 할당받은 메모리의 시작점에서 떨어져있는 위치 : offset 주소
어플리케이션은 상대주소를 다룬다.

*배열 레퍼런스와 인스턴스



*배열 레퍼런스와 인스턴스
int[] arr1;
arr1 =new int[5];
arr1=null;
특정 인스턴스의 주소를 담고있지 않다고 선언한다.
이 때 컴파일하면 NullPointerException 에러가 뜬다.

*배열 메모리의 기본 값
로컬 변수는 값을 저장하지 않고 사용할 수 없다.
new 명령으로 메모리를 만들고 기본 값을 주지 않았을 때 정해진 값으로 자동 초기화된다.
- 정수 배열(byte[], short[], int[], long[]) : 0
- 부동소수점 배열(float[], double[]) : 0.0
- 논리 배열(boolean[]) : false
- 문자 배열(char[]) : '\u0000'
- 주소 변수(Object[]) : null

* 배열 선언 + 초기화
선언과 초기화 같이 할 때 new int 생략 가능한다
int[] arr3 = {100, 90, 80, 60, 70};

*자바의 변수 종류
byte b;
short s;
int i;
long l;
----------정수 값 저장
floar f;
double d;
----------부동 소수점 값 저장
boolean b;
----------논리 값 저장

char c;
----------유니코드 저장
----------primitive data type 변수

+ 그 외 = 주소 변수 (배열, 레퍼런스 포함)

*로컬 변수, static 변수, class 변수, 인스터스 변수
class A {
int v1; ----------인스턴스 변수 : new 명령을 실행할 때 생성된다.
static int v2;--------클래스 변수:클래스를 로딩할 때 생성된다.
static void m(){
  int v3;---------로컬 변수 : 매서드를 실행할 때 생성된다.
}
}

static 이 붙은 블록(스태틱 메서드=클래스 메서드)에서는
그 블록 바깥 쪽에 있는 스태틱 변수(클래스 변수)를 사용할 수 있다.
그러나 static 붙지 않은 인스턴스 변수는 사용할 수 없다.

*static type binding vs dynamic type binding
자바
int i;
i=100;
i= 3.14f; - 안됨
i = true; - 안됨
i = "Hello"; -안됨
변수의 타입이 한 번 정해지면 변경되지 않는다.

자바 스크립트
var i;
i=100;
i = 3.14;
i = true;
i = "Hello";
다 가능.
변수에 값을 저장하는 순간 데이터 타입이 결정/변경된다.

* 명시적 형변환
큰 정수 변수의 값을 작은 정수 변수에 저장
명시적 형변환 시 4byte 값을 1byte 에 넣으면 앞의 3byte 는 짤린다.

1) 정수 메모리끼리 형변환이 가능하다.
2) 부동소수점을 정수로 형변환이 가능한다.
3) 형변환 없이 정수는 부동소수점 메모리에 저장할 수 있다.
4) 숫자를 문자 코드로 형변환 가능한다.
그 외에는 형변환 불가!

*파라미터
 메서드의 아규먼트를 받는 로컬 변수이다.
```
public static void main(String[] args/*로컬변수=파라미터*/) {
    int c; // 로컬 변수
  }
```
예) 위의 코드에서 main()의 args 로컬 변수

*연산자 (Operator)
1. 산술 연산자 : +, -, *, / , %
  -우선 순위 () > ++, -- >   / , * , % > +, - > =
 - 정수의 산술 연산의 최소 단위는 int

만약 byte, short 형에 리터럴 4바이트 정수 값을 저장할 수 있다면, 허용해준다.
ㄴ 리터럴 값을 작은 크기의 변수에 저장할 수 있다면 컴파일 허락한다.
```
byte b = 5 + 6; // 가능
```

하지만... 자바의 정수 연산은 최소 단위가 4바이트다.
바이트형과 바이트 형의 연산은 불가능하다. 연산 전에 int 형으로 바꿔준다..
```
byte x =5;
byte y =6;
z = x;
z = y;

byte z = x + y; // 요거 불가능
```
x,y값이 바이트니까!

임시의 int 메모리를 준비하고 연산하면 연산 결과도 int가 된다.
%% byte 연산의 결과는 byte 가 아니라는 것 !!!!! %%

이와 같이 short 와 short 값을 더하면 int 변수 값이 나온다.

결과 값을 담을 int 변수 (임시 4바이트 메모리) 를 만든다.
byte 변수는 보통 파일의 데이터를 읽을 때 사용한다.

이미지 파일이나 텍스트파일.

-데이터 타입과 연산자
데이터 타입에 따라 사용할 수 있는 연산자가 정해져 있다.

문자열의 경우 + 만 가능.
- , * 는 컴파일 오류 발생.

Boolean타입은 산술 연산자를 사용할 수 없다.

System.out.println(true && true); ㅡ 가능
System.out.println(10 && 10); ㅡ 불가능
&& 연산자는 정수에 사용할 수 없다.

-연산의 결과 타입은 피연산자와 같다.
int i = 5;
int j = 2;
float r = i / j;

r의 값은 2.0이 된다.
int와 int의 연산 결과는 항상 int이다. 그 int 값을 r에 넣기 때문에 2.0이다.

2.5 값이 나오게하기 위해선 명시적 형변환을 해야한다.
int / int = int
(float) int / (float) int = float

int와 int의 연산 결과는 int이다.
다른 타입이 될 수 없다.
=> 0111 1111 1111 1111 1111 1111 1111 1111 = Integer.MAX_VALUE

int x = Integer.MAX_VALUE; // 0x7fffffff = 약 +21억
int y = Integer.MAX_VALUE; // 0x7fffffff = 약 +21억
int r1 = x + y;
0111 1111 1111 1111 1111 1111 1111 1111(x)
+ 0111 1111 1111 1111 1111 1111 1111 1111(y)

1111 1111 1111 1111 1111 1111 1111 1110(r1)
System.out.println(r1); // int(4byte) + int(4byte) = int(4byte)
ㄴ 결과 -2

%%int와 int의 연산 결과가 int의 범위를 넘어가면 의도한 결과가 나오지 않을 수 있다.%%
ㄴ 오버플로우 값이 짤리고 음수 값이 나타날 것이다.

그래서 int와 int의 연산 결과를 더 큰 메모리에 담는다면 해결될까?
long r2 = x + y;
System.out.println(r2);
ㄴ 결과 -2

int 와 int의 연산 결과는 피연산자와 같은 4바이트 int가 된다.
그래서 8바이트 long 변수에 저장하기 전에 이미 그 결과는 int 값으로 -2가 되기 때문에
long 변수의 값이 -2가 된다.

int와 int 연산 결과가 int 크기를 넘어갈 것 같으면 형변환해야한다.
r2 = (long)x + (long)y;
System.out.println(r2);

-암시적 형변환
데이터 타입이 서로 같을 때만 연산 가능하다.
데이터 타입이 다를 경우 임시적 형변환이 일어나 계산해 더 큰 데이터 타입의 값을 내놓는다.

2.관계 연산자(relational operators: <, <=, >, >=)와 등위 연산자 (equality operators: ==, !=)

비교의 결과는 true 또는 false이다
즉, boolean값이다.
int 로 담을 수 없다!!!!

- 부동소수점 비교하기
float f1 = 0.1f;
float f2 = 0.1f;
System.out.println(f1 * f2 == 0.01f);
ㄴ FALSE 값이 나온다.

극한의 작은 값은 EPSILON이라고 부른다.

double EPSILON = 0.0001;
Math.abs((f1 + f2) - (x + y)) <EPSILON);

부동 소수점을 비교할 땐 비교값에서 비교값을 빼고 그 남은 값이 사용자가 정한 극한의 값보다 작으면

같다고 치자 ~~ 이렇게 처리된다.
부동소수점 비교 시 절대 그냥 == 로 처리해선 안된다.

앱실론을 정하지 않고 결과 결과를 절대값으로 만든 후에
Float.POSITIVE_INFINITY 를 사용한다 (극한의 값을 미리 정해놓음)

예시)
float r = f1 * f2 - 0.01f;
 System.out.println(Math.abs(r) <= Float.POSITIVE_INFINITY);

결과는 TRUE이다.

*논리 연산자 (&&, ||, !(not), ^(XOR; exclusive-OR)
&&, ||, !(not), ^(XOR; exclusive-OR)
-AND 연산자 &&
두 개의 논리 값이 모두 true일 때 결과가 true가 된다.
-OR 연산자
두 개의 논리 값 중 한 개라도 true이면 결과는 true가 된다.

-exclusive-OR(XOR)연산자
배타적 비교 연산자라 부른다.
두 개의 값이 다를 때 true이다.
XOR 연산자를 정수 값에 대해 수행하면 비트 단위로 연산을 수행한다.

* && vs & 논리 연산자
a = false;
b = false;
r = a && (b = true);
첫번째 피연산자로 결과를 예측할 수 있다면 두 번째 피연산자는 실행하지 않는다

a =false;
b=false;
r = a & (b=true);
앞의 피연산자값에 상관없이 끝까지 실행한다.
평상시엔 되도록이면 && 나 || 같이 두개 짜리를 사용해야한다.

*비트 연산자 (&, |, ^, ~)

정수 값에 대해서는 &&와 ||, !을 사용할 수 없다.
비트 연산은 이미지 표현, 색 처리에 주로 사용된다.

*조건연산자
 = 조건? 표현식 : 표현식;

첫 번째 표현식은 참일 때 실행
두 번째 표현식은 거짓일 때 실행

문장 (statement) 표현식 (expression)
문장 : 작업을 수행시키는 명령어이다.
표현식 : 문장을 실행한 후에 결과를 리턴한다.
statement 중에서 결과를 리턴하는 statement를 expression이라 부른다.

조건 연산자는 할당 연산자 (=)의 왼편에 변수를 선언해야 한다.
선언하지 않으면 문법오류

조건 연산자의 결과 값이 왼편의 변수 타입과 일치해야한다.
결과 값이 없거나 타입이 일치하지 않으면 문법 오류

* 증감연산자 : 전위(prefix) / 후위(postfix)
int i =100;
컴파일 할 때 다음 문장으로 변환된다

전위 연산자 ++i;
i = i + 1

후위 연산자 i++;
temp = i;
i = i+ 1;

리터럴에 대해 증감 연산자 적용할 수 없다.
변수에 동시에 적용할 수 없다
++y++; < - 이런 식이 안된다는 뜻

*할당 연산자
int i =2;

i = i + 20; == i +=20;

i =2;
i *= 5;