22.11.23 - Array (Reference type)

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b2481ebf-dc1c-4197-9b75-3948b9c7cfe0/JAVA_5.pdf

자바에는 **기본 자료형(Primitive type)**과 **참조 자료형(Reference type)** 두 가지 종류의 자료형이 있다.

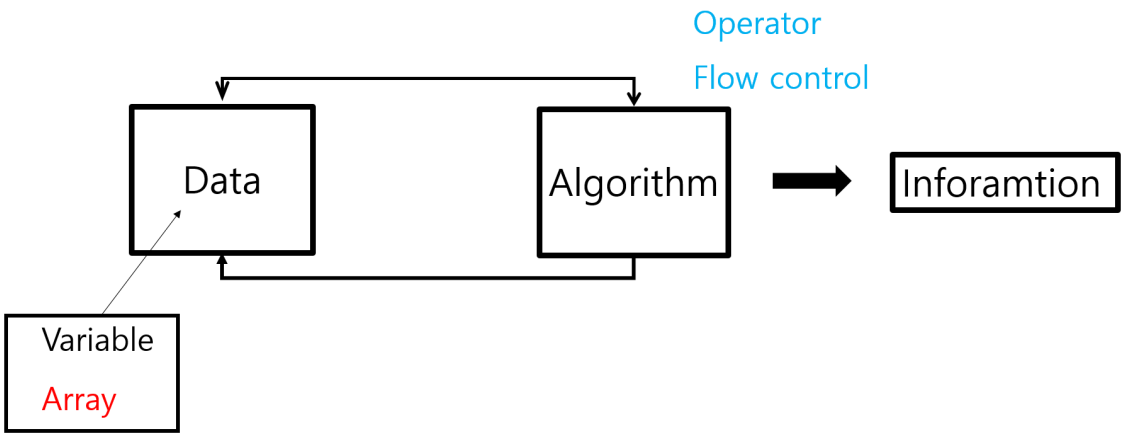
`기본 타입`으로 선언된 변수와 `참조 타입`으로 선언된 변수의 차이점은 `★`저장되는 값`★`!!

`기본 타입`	`참조 타입`
실제 값을 변수에 저장	메모리의 번지를 변수에 저장

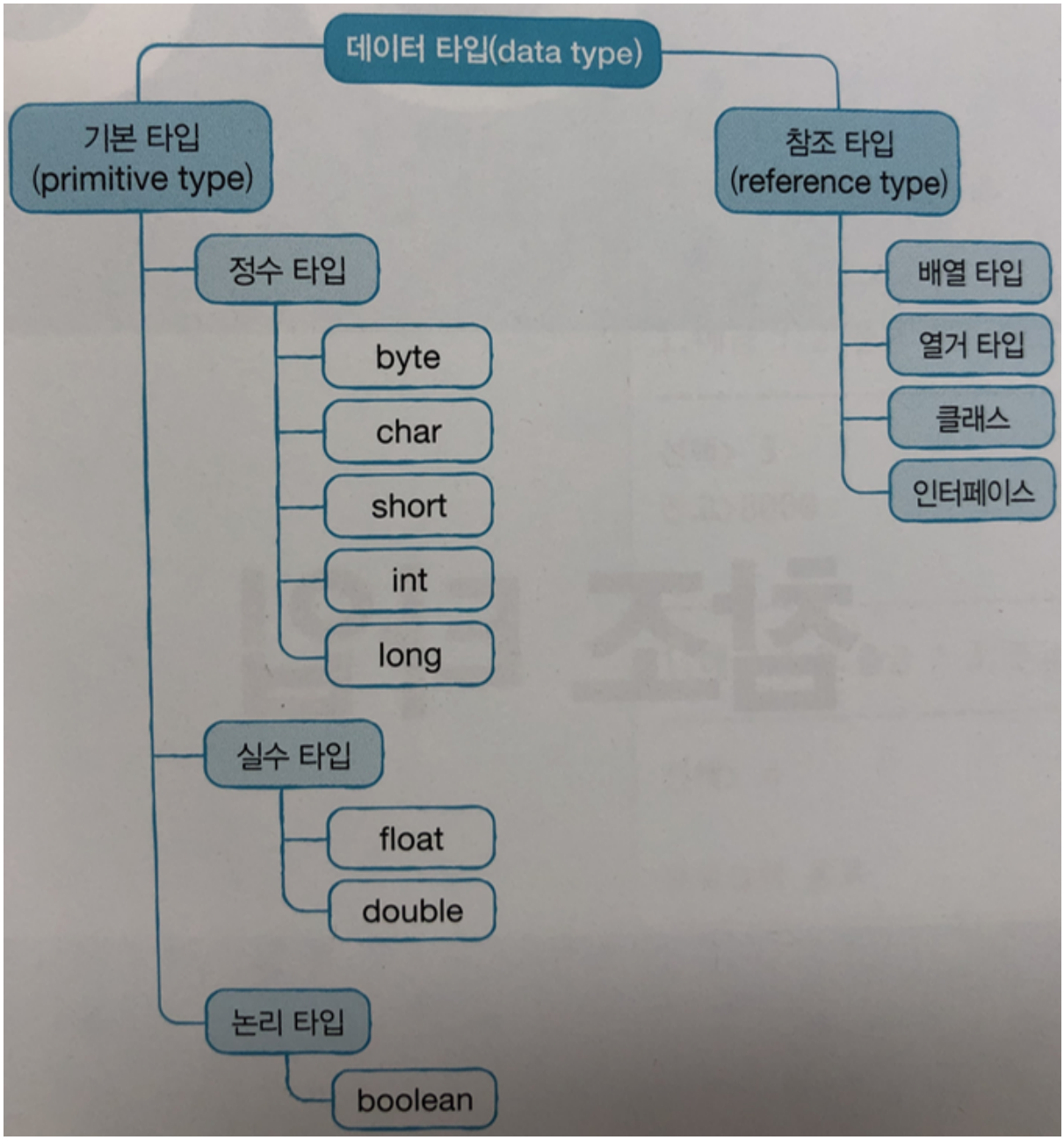
객체의 번지를 참조 (Reference type)
배열, 열거, **클래스**, 인터페이스

프로그램 형식

A PROGRAM



> **구조적 자료형**
> 여러 자료를 묶어서 하나의 단위로 처리하는 자료형을 `구조적 자료형(structured data type)`이라 하는데,
> `배열(동질형 자료의 모임)` 과 `레코드(이질형 자료의 모임)`로 구분할 수 있다.
> c언어에 서는 레코드를 `( 구조체)` 라고 한다

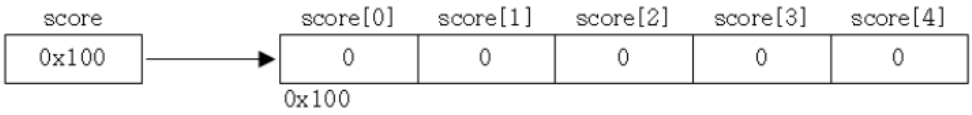

데이터 타입(data type)
기본 타입
(primitive type)
참조 타입
(reference type)
정수 타입
byte
char
short
int
long
실수 타입
float
double
논리 타입
boolean
배열 타입
열거 타입
클래스
인터페이스

`Array` 이란?

• ★**같은 자료형**을 가지는 ★**변수들이 나열**된 집합
(같은 타입의 여러 변수를 하나의 묶음으로 다루는 것)
• `Array`**의 각 요소는 서로 연속적이다**
• `Array` 는 data 에 속 한다 , data 를 효율적으로 저장하기 위해서 사용⇒ 변수의 확장판

나열 = "중간에 빈틈없이 연속적으로 줄을 세우는 것"

```
int[] score = new int[5]; // 5개의 int 값을 저장할 수 있는 배열을 생성한다.
```



`배열`의 사용 이유는?

• 큰 덩어리의 변수들을 효율적으로 관리하기 위한 방안으로 `배열` 을 사용
• 변수의 선언과 달리 다루어야 할 데이터의 수가 아무리 많아도 배열의 길이만 바꾸면 됨

`배열`의 선언과 생성

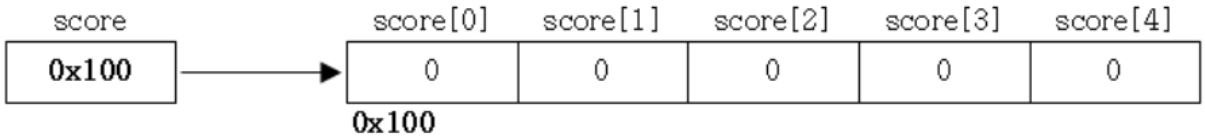
선언 방법	선언 예
타입 [ ] 변수 이름 ;	int [ ] foo ;
★ 타입 변수 이름 [ ] ; ★	int bar [ ] ;

생성 방법	생성 예
변수 이름 = `new` 타입 [길이]	score = `new` int [5]

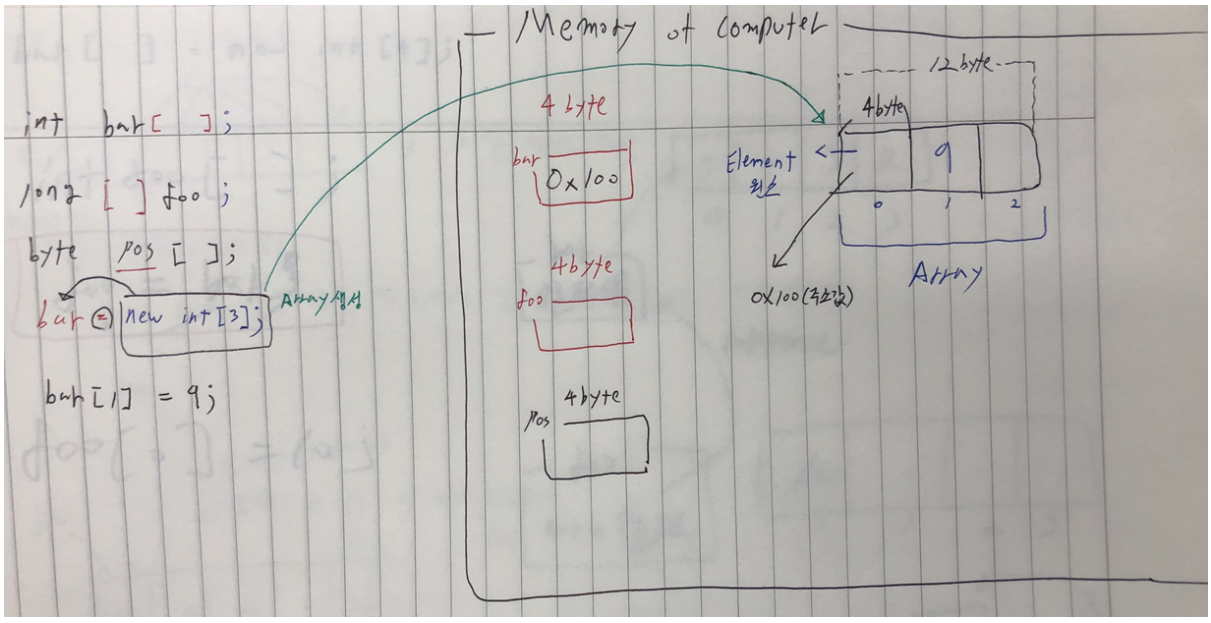
(new 연산자) 선언과 생성 동시에 하는 방법

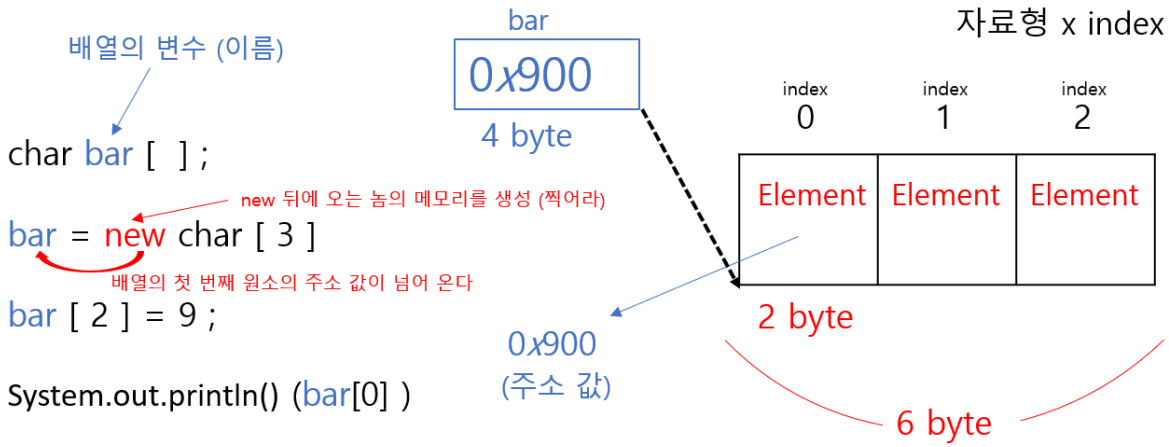
타입 [ ] 변수 = **new 타입** [길이] ;

```
int [ ] score ;       // 배열 선언 (생성된 배열을 다루는데 사용 될 참조 변수 선언)
score = new int [5]; // 배열 생성 (5개의 int 값을 저장할 수 있는 공간생성)

=> 위의 두 문장은 int score [ ] = new int [5] 로 한문장으로 줄여 쓸 수 있다.

| ★ 배열을 선언한다고 해서 값을 저장할 공간이 생성되는 것이 아니라
| ★ 배열을 다루는데 필요한 변수가 생성된다.


★ new = 객체를 메모리상에 찍는다는 의미 (동적)
         (프로그램 실행도중에 메모리를 원하는 만큼 메모리상에 할당 시켜준다)


「프로그램의 실행 기준에 따라」
정적 - 프로그래밍이 실행 되기 **前** 이미 컴퓨터에서 사용될 메모리 크기가 정해져 있음

동적 - 프로그래밍이 실행 되는 **途中**  (실행 後) 에 필요에 따라서
       컴퓨터가 메모리를 늘려 나가는 것
```



Memoru of Computer





`원소(Element)` - 배열 안의 각각의 항목 • `인덱스(index)` - 배열의 요소마다 붙여진 일련번호

메모리 주소

특정 데이터가 있으면 데이터를 메모리에 저장하기 위해
각각의 공간에 대한 `좌표 값` = `메모리 주소` 을 찍어야 한다 .
메모리는 각각의 데이터가 저장이 되면 그 저장된 공간에 대한 `메모리 (상의)_ 주소` 가 있다

⇒ 참조 타입은 객체의 번지 값을 가지고 있다.
⇒ 주소 값이 있어야 위치를 지정하고 위치에 대한 값을 읽을 수 있다.
⇒ 배열의 주소는 메모리 주소 값을 저장한다.

※ 메모리 주소 값은 운영체제의 영향을 받는다 ※
ex) **32 bit - 4 byte**
64 bit - 8 byte

자바는 JVM의 영향을 받는다 , JVM이 몇 Byte 사용 하는지에 따라 변수의 크기가 바뀐다
★★★ 배열의 변수는 항상 4byte ⇒ 주소 값을 저장하기 때문이★★★

배열의 변수가 `자료형`을 가지는 이유

배열의 변수를 선언할 때 `자료형`을 붙인다.

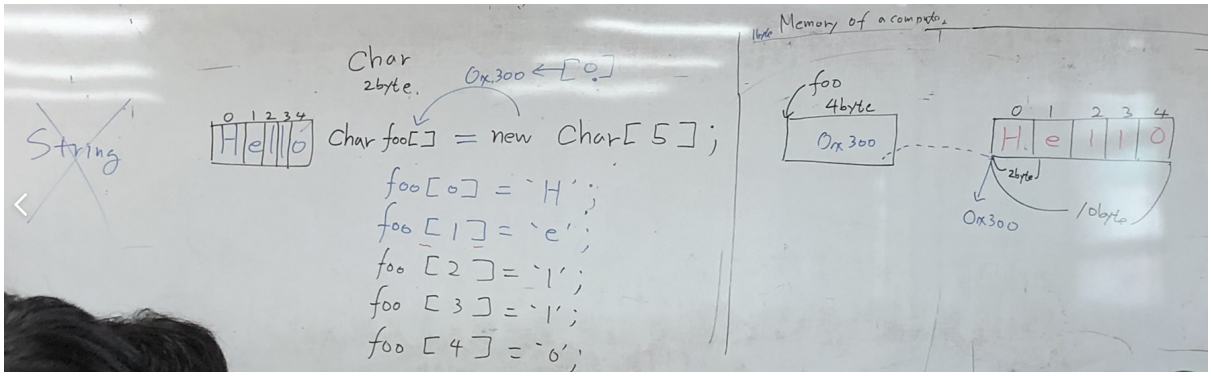
★ 배열의 첫 번째 원소의 주소 값이 넘어와서 변수에 저장 된다.

첫 번째 원소를 기준으로 배열의 자료형 크기에 따라
Byte를 더해 원하는 원소 값을 찾기 위해서 이다

⇒ 배열의 원하는 원소(배열의 주소 값)를 찾아가기 위해서 자료형을 선언한다.

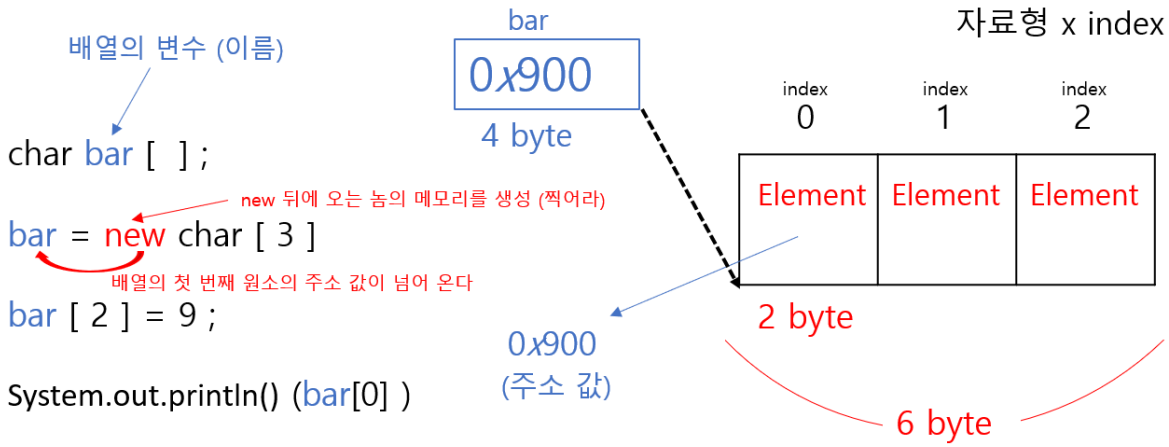
※ **자료형이 다르면 element를 찾아 갈 수 가 없다** ※

char 형 배열



하나의 배열을 구성하면 메모리 상에서 크게 2part로 나눠서 데이터가 구분된다.

1. 실제 배열 안의 원소 값을 저장하는 공간
2. 이 배열을 가르키기 위한 배열의 변수의 공간이 구성된다



- 메모리 상에서 배열의 변수는 만들어진 배열의 첫 번째 주소를 담기 위해서 4byte로 정해진다 (주소의 크기는 jvm에 의존적)
- 메모리상에서 데이터가 있으면 어디에 있는지 구분하기 위해 `메모리 주소`를 사용한다
- 각각의 데이터는 메모리 상의 주소 값을 가진다

변수의 동작 모드 (변수로 할 수 있는 것)

- `get` - 변수의 값을 읽어 오는 것
- `set` - 변수에 값을 저장하는 것

초기 값

변수의 초기 값 = 변수가 만들어지면서 처음으로 넣는 값

배열의 초기 값 ⇒ 배열의 변수 선언 후 뒤에 new 찍을 필요 없이
리터럴 상수가 블레이스에 싸여져 있으면 jvm 에서 자동으로 배열을 생성한다

배열을 생성하면서 초기 값까지 넣는 방법

```
int bar[ ] = {10 , 7 , 3};
```

```
    // 초기값 10

    int foo[] = new int[3];
```

```
  foo[0] = 10;
  foo[1] = 7;
  foo[2] = 3;

  // 배열을 생성하면서 초기 값까지 넣을 수 있는 방법
  int bar[ ] = {10 , 7 , 3};

  System.out.println(bar[2]);
```

⇒ 블레이스에 묶여 있으면 자동적으로 배열 생성 , 콤마개수 + 1 = 원소의 개수

⇒ 리터럴 상수를 이용해서 배열을 생성하면서 동시에 초기 값까지 넣을 수 있다

```
// 방법 1     (C언어 스타일 )

  // h e l l o
  char bar [ ] = {'h','e','l','l','o'};

  for (int i = 0; i < bar.length; i++) {
    System.out.println(bar[i]);

// 방법 2      (Java 스타일 )
  // h e l l o
  char bar[] = new char[]{ 'h', 'e', 'l', 'l', 'o' }; // 브라켓 안에 정수의 개수 x
                                        //정수의 개수는 뒤에오는 리터럴 상수 개수와 매핑
  for (int i = 0; i < bar.length; i++) {
    System.out.println(bar[i]);
  }
```

※ 상수는 자료형 앞에 `final` 를 쓰고 상수 이름을 대문자로 쓴다 ※

배열의 원소 개수는 1개의 상수로 관리하면 편하다

```
  // 1 ~ 50 사이 정수 중 난수로 10개를 선택하시오   => 개수가 10개인 INT 형 배열 만들기

  final int THE_NUM_OF_ELEMENT = 10;   // 상수

  int bar [ ] = new int[THE_NUM_OF_ELEMENT]; // 배열의 크기는 하나의 상수로 관리 한다 !

  for (int i = 0; i < 10; i++) {

    bar[i]= (int)(Math.random()*50)+1;

  }

  for (int i = 0; i < 10; i++) {
```

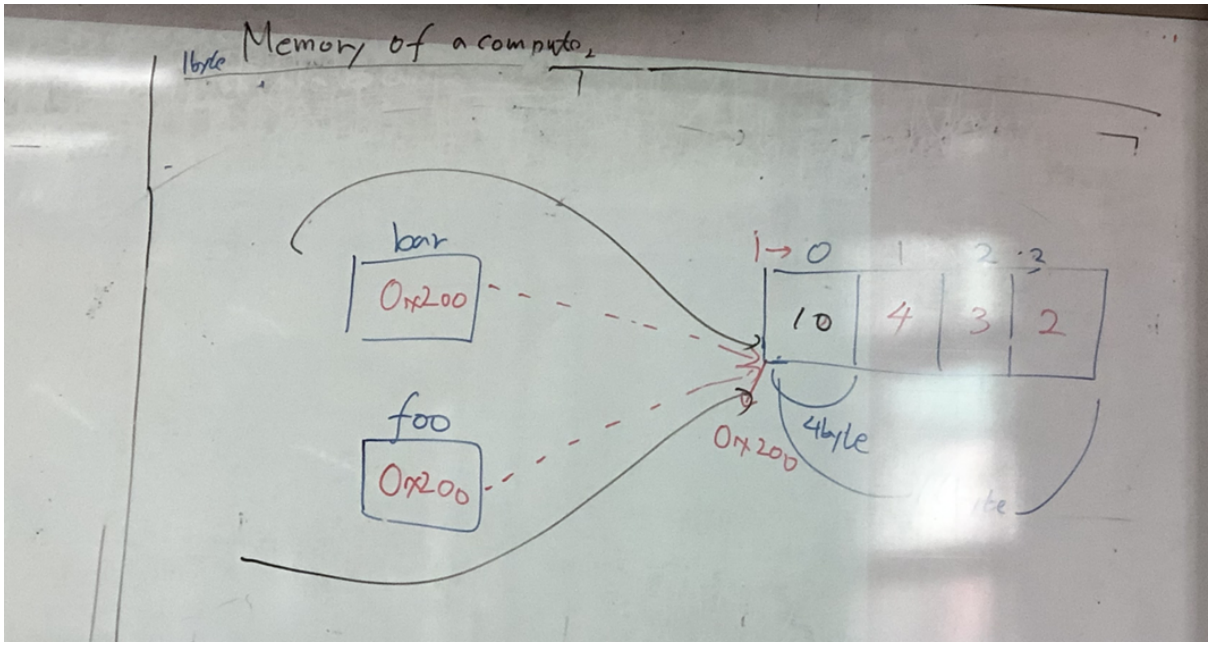
```
      System.out.println(" " + bar[i]);
    }
```

배열의 길이

- 배열의 원소의 개수를 반환 한다.
- 자바에서 배열은 객체라는 단위로 관리한다
- 사용 방법 ⇒ `배열 변수.length;`
- 배열의 인덱스 값을 조절하면 반복문을 이용해 배열 안에 있는 원소의 값을 읽어 올 수 있다

「 배열의 변수는 메모리 주소 값을 저장 」

```
// array 길이 연습

  int bar [ ] = new int[4]; // new 해서 생성 후 엘리먼트 값들은 0으로 초기 값이 다들어 간다


  // for문 안에 초기변수 여러개 선언 가능
  for (int i = 0, j = 5 ; i < bar.length ; i++ , j--) {    // i =>  0 1 2 3

    bar [i] = j;          //  bar 변수 생성
                           // 0 들이 지워 지고  | 5 | 4 | 3 | 2 |   으로 bar 배열에 저장
  }


  int foo [];  // foo 변수 생성

  foo = bar;  // bar 배열의 첫 번째 주소 값이 저장  ( 0x200 )  => 동일한 배열값을 가르킴

  foo[0] = 10;

  System.out.println(bar[0]);        결과 값 =>  10


          ★ 동인한 배열을 가르키는 여러개의 배열의 변수가 존재할 수 있다 ★
```



`Math.random( )`

`0.0 이상` ~ `1.0 미만` **사이**의 **double 형**의 값을 반환하는 함수 (난수를 만들 때 자주 사용)
⇒ **0.xxxx ~ 0.9xxx 까지의 값을 반환**한다.

랜덤 함수는 `기본형이 Double형` **이기에** `(int)` **로** `정수화` 시켜주어야 한다.

1 부터의 값을 뽑고 싶다면 **랜덤 함수는 0부터 나오기 때문에 +1을 꼭 시켜줘야 함**

例)

```
Math.random() );  // 0.23279967568276427
Math.random() * 10 );  // 2.3279967568276427 (0.xxx... ~ 9.xxx 까지의 값 반환)
(int) Math.random() * 10 );  // 2 ( 0부터~9까지의 값 반환 )
(int) Math.random() * 10 +1 );  // 3 ( 1부터~10까지의 값 반환 )
```

問題. 로또

```
// 로또 번호 생성 프로그램 작성
    // 1 ~ 45 중복되지 않는 정수 6개 랜덤으로 선택
    // 정수 6개는 배열에 저장.


    // 배열 생성
    int rotto[] = new int[6];

// 1) for_1 - 총 인덱스 6개에를 하나씩 => [0] [1] [2] [3] [4] [5] = 6개
  /* 1 */ for (int index_1 = 0; index_1 < rotto.length; index_1++) {

      // 랜덤 생성
      int ransu = (int) (Math.random() * 45) + 1; // 0 ~ 1 미만 0.999999

      // 1.1)  rotto[0]인 경우
      if (index_1 == 0) {
        rotto[0] = ransu;
      }


      // 1.2)  rotto[1]이상 인 경우
      else {

// 2) for_2 - for_1 보다 작은 범위까지 돌려서 안의 내용 확인
    /* 2 */ for (int index_2 = 0; index_2 < index_1; index_2++) {

          // 2.1) 값 중복
          if (rotto[index_2] == ransu) {
            index_1 -= 1; // 임의로 index 값 조정
            break;
          }

          // 2.2) 값 중복 안함
          rotto[index_1] = ransu;
        }
      }
    }

    // ※ 출력 ※
    System.out.println("나눔\nLotto 6/45 \n" + "대박나세요~!");
    System.out.println("----------------------------------------");
    for (int i = 0; i < rotto.length; i++) {
      System.out.print(rotto[i] + "\t");
    }
```