
# Chapter 13. 배열

# 13-1. 배열이라는 존재가 필요한 이유

# ■ 변수 선언의 편의성



배열을 이용하면 아무리 많은 수의 변수라 할지라도 하나의 문장으로 선언하는 것이 가능하다.

**일반적인 방식의 변수 선언**

```
int num1, num2, num3;
int num4, num5, num6;
int num7, num8, num9;
```

**배열 기반의 변수 선언**

```
int[ ] numArr=new int[9];
```

총 9개의 변수가 선언되었다는 사실에는 차이가 없다.

# ■ 순차적 접근의 허용



배열을 이용하면 반복문을 이용한, 배열을 구성하는 변수에 순차적으로 접근할 수 있다.

**일반적인 방식의 변수 선언**

```
num1=num2=num3=10;
num4=num5=num6=10;
num7=num8=num9=10;
```

**배열 기반의 변수 선언**

```
for(int i=0; i<9; i++)
{
    numArr[i]=10;
}
```

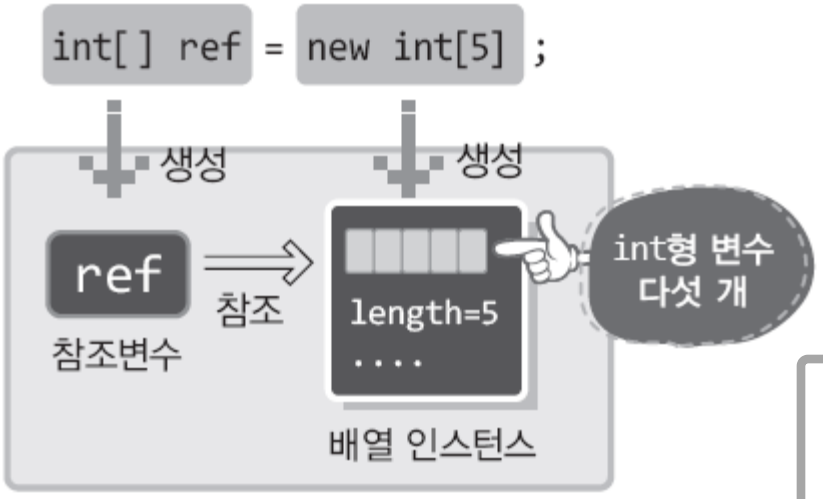
값을 참조 및 변경할 변수의 수가 100개라고 가정해보면, 위의 차이는 더 극명하게 드러난다.

# 13-2. 1차원 배열의 이해와 활용

# 배열 인스턴스의 생성방법



배열도 인스턴스이다. 둘 이상의 데이터를 저장할 수 있는 형태의 인스턴스이다.




배열 인스턴스와 참조의 생성 모델

배열 인스턴스의 생성의 예

```
int[] arr1 = new int[7];
double[] arr2 = new double[10];
boolean[] arr3 = new boolean[6];
FruitSeller[] arr4 = new FruitSeller[5];
FruitBuyer[] arr5 = new FruitBuyer[8];
```

배열 인스턴스의 생성방법은 일반적인 인스턴스의 생성방법과 차이가 있다.

# ■ 배열의 접근방법

나 정말 JAVA를 공부한 적 없는데

- 배열의 접근에는 0부터 시작하는 인덱스 값이 사용된다. 가장 첫 번째 배열 요소의 인덱스가 0이고 N번째 요소의 인덱스가 N-1이다.
- 배열 인스턴스의 멤버변수 length에는 배열의 길이정보가 저정되어 있다.

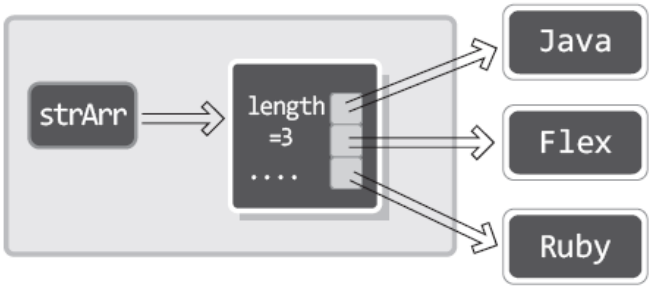
### 기본 자료형 배열

```
public static void main(String[] args)
{
    int[] arr = new int[3];
    arr[0]=1;
    arr[1]=2;
    arr[2]=3;

    int sum=arr[0]+arr[1]+arr[2];
    System.out.println(sum);
}
```

### 인스턴스의 배열

```
public static void main(String[] args)
{
    String[] strArr=new String[3];
    strArr[0]=new String("Java");
    strArr[1]=new String("Flex");
    strArr[2]=new String("Ruby");

    for(int i=0; i<strArr.length; i++)
        System.out.println(strArr[i]);
}
```

위 예제는 배열요소의 순차 접근을 보이고 있다!




인스턴스 배열에는 인스턴스가 저장되는 것이 아니라,
인스턴스의 참조 값이 저장된다.

# ■ 배열의 선언과 동시에 초기화



일반적인 인스턴스 배열의 선언

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

초기화할 데이터들을 중괄호 안에 나열한다.

```
int[] arr = new int[3] {1, 2, 3};
```

단, 초기화 데이터의 수를 통해서 길이의 계산이 가능하므로 길이정보 생략하기로 약속!

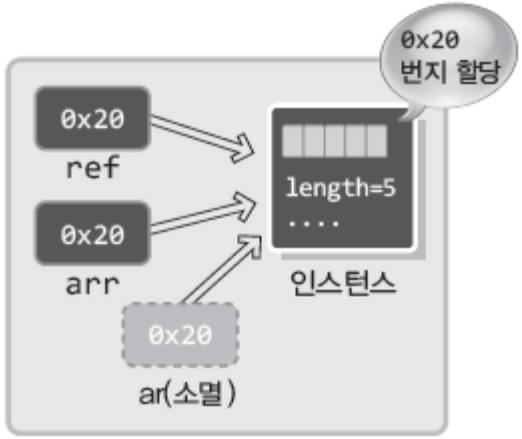
```
int[] arr = new int[ ] {1, 2, 3};
```

이렇게 줄여서 표현하는 것도 가능하다.

```
int[] arr = {1, 2, 3};
```

# 배열과 메소드

배열도 인스턴스이기 때문에 메소드 호출시의 인자전달 및 반환의 과정이 일반적인 인스턴스들과 다르지 않다.




```
class ArrayAndMethods
{
    public static int[] addAllArray(int[] ar, int addVal)
    {
        for(int i=0; i<ar.length; i++)
            ar[i]+=addVal;
        return ar;
    }

    public static void main(String[] args)
    {
        int[] arr={1, 2, 3, 4, 5};
        int[] ref;
        ref=addAllArray(arr, 7);
        if(arr==ref)
            System.out.println("동일 배열 참조");
        else
            System.out.println("다른 배열 참조");
        for(int i=0; i<ref.length; i++)
            System.out.print(arr[i]+" ");
    }
}
```

참조 값 ar을 그대로 반환

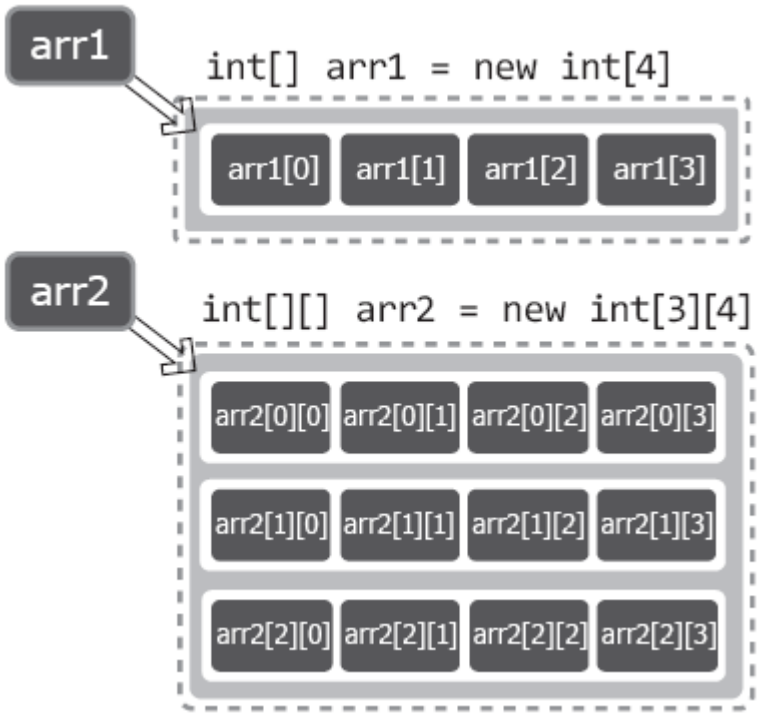
배열 arr의 참조 값 전달

# 13-3. 다차원 배열의 이해와 활용

# ■ 1차원 배열 vs. 2차원 배열



2차원 배열은 2차원의 구조를 갖는 배열이다. 따라서 가로와 세로의 길이를 명시해서 인스턴스를 생성하게 되며, 배열에 접근할 때에도 가로와 세로의 위치정보를 명시해서 접근하게 된다.




**2차원 배열의 선언방법**

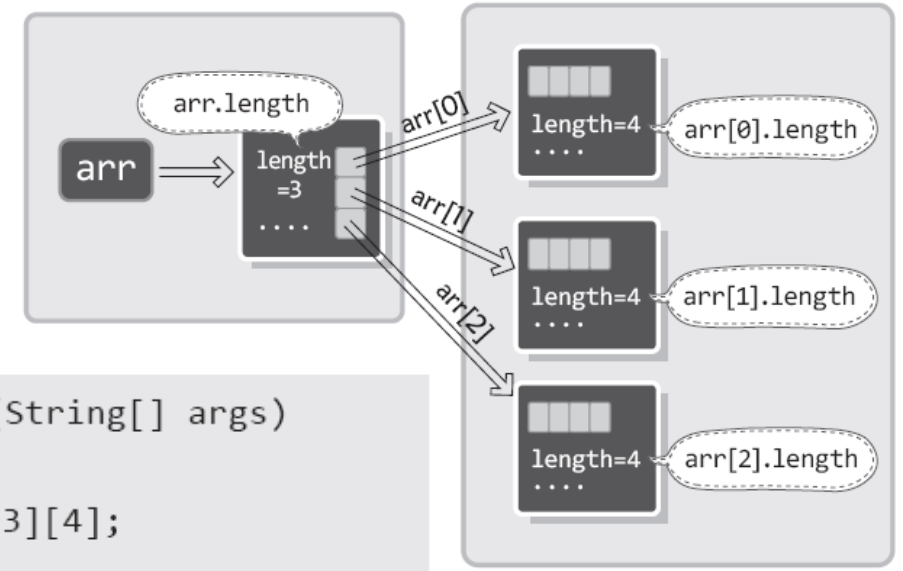
- 가로길이가 2이고, 세로길이가 7인 int형 배열
  → `int[][] arr1 = new int[7][2];`
- 가로길이가 5이고, 세로길이가 3인 double형 배열
  → `double[][] arr2 = new double[3][5];`
- 가로길이가 7이고, 세로길이가 3인 String 배열
  → `String[][] arr3 = new String[3][7];`

2차원 배열에 접근할 때에는 arr[세로][가로]의 형태로 위치를 지정한다.

# ■ 예제를 통한 2차원 배열구조의 이해



2차원 배열의 메모리 구조 ▶




```
public static void main(String[] args)
{
    int[][] arr=new int[3][4];

    for(int i=0; i<arr.length; i++)
        for(int j=0; j<arr[i].length; j++)
            arr[i][j]=i+j;

    for(int i=0; i<arr.length; i++)
    {
        for(int j=0; j<arr[i].length; j++)
            System.out.print(arr[i][j]+" ");

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

이를 통해서 2차원 배열은 둘 이상의 1차원 배열을 묶은 형태라는 사실을 알 수 있다!

# ■ 2차원 배열의 선언 및 초기화

나는 정말 JAVA를 공부한 적이 없다

### 2차원 배열의 선언 및 초기화 1

```
int[ ][ ] arr={
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

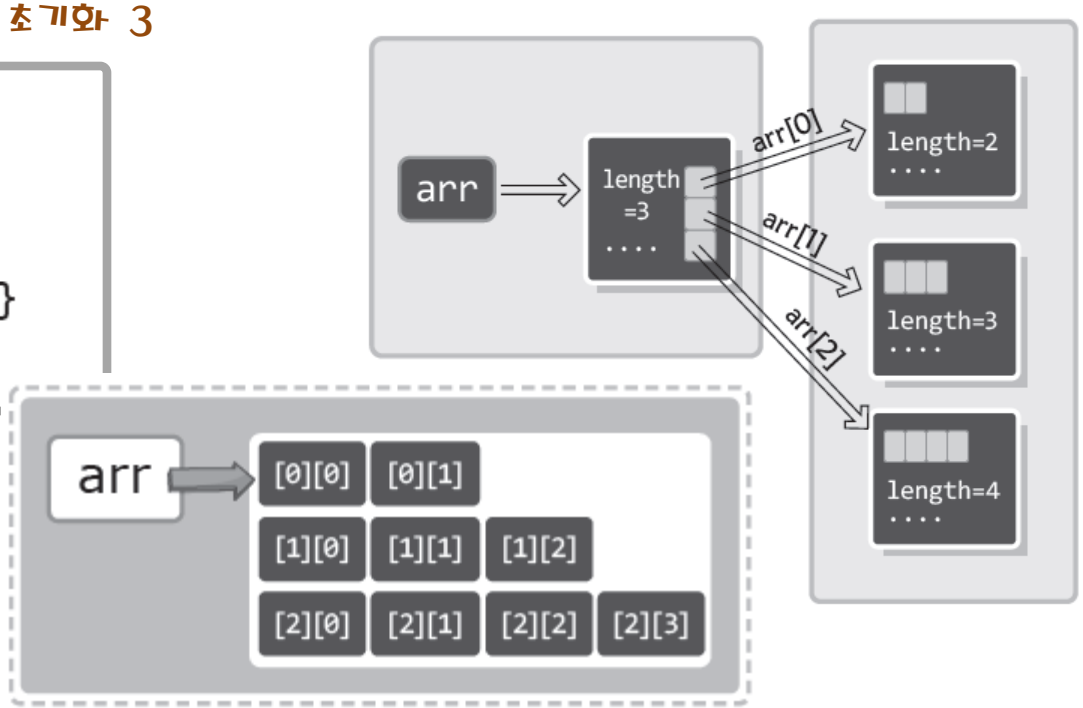
동일

### 2차원 배열의 선언 및 초기화 2

```
int[ ][ ] arr=new int[ ][ ] {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

### 2차원 배열의 선언 및 초기화 3

```
int[ ][ ] arr={
    {1, 2},
    {3, 4, 5},
    {6, 7, 8, 9}
};
```



# 13-4. for-each문

# for-each문의 이해와 활용

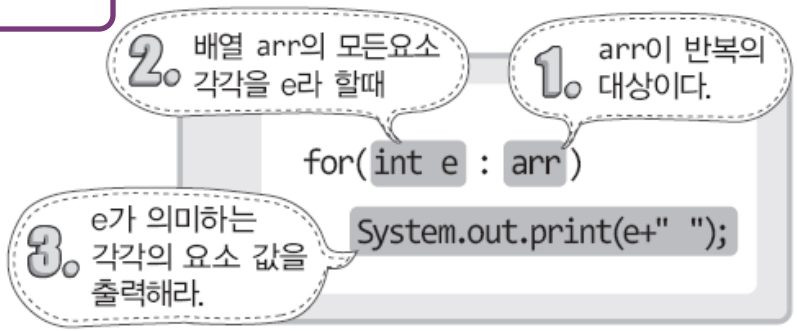


배열의 일부가 아닌, 배열의 전체를 참조할 필요가 있는 경우에 유용하게 사용할 수 있다.

```
for(int i=0; i<arr.length; i++)
    System.out.print(arr[i]+" ");
```

```
for(int e : arr)
    System.out.print(e+" ");
```

코드 분량이 짧아졌고, 필요로 하는 이름의 수가 arr, i, length에서 e와 arr로 그 수가 하나 줄었다.




for-each 문을 통한 값의 변경은 실제 배열에 반영되지 않으니, 값의 참조를 목적으로만 사용해야 한다.

# 인스턴스 배열에 대한 for-each문



인스턴스 배열에 저장된 참조 값의 변경은 불가능하지만, 참조 값을 통한 인스턴스의 접근은(접근 과정에서의 데이터 변경은) 가능하다!

```
public static void main(String[] args)
{
    Number[] arr=new Number[] {
        new Number(2),
        new Number(4),
        new Number(8)
    };
    for(Number e : arr)
        e.num++;
    for(Number e : arr)
        System.out.print(e.getNum()+" ");
    System.out.println("");
    for(Number e : arr)
    {
        e=new Number(5);
        e.num+=2;
        System.out.print(e.getNum()+" ");
    }
    System.out.println("");
    for(Number e : arr)
        System.out.print(e.getNum()+" ");
}
```

인스턴스의 접근이므로 반영됨

배열의 참조 값 변경이므로 반영 안됨

# 13-5. main 메소드로의 데이터 전달

# main의 매개변수 선언

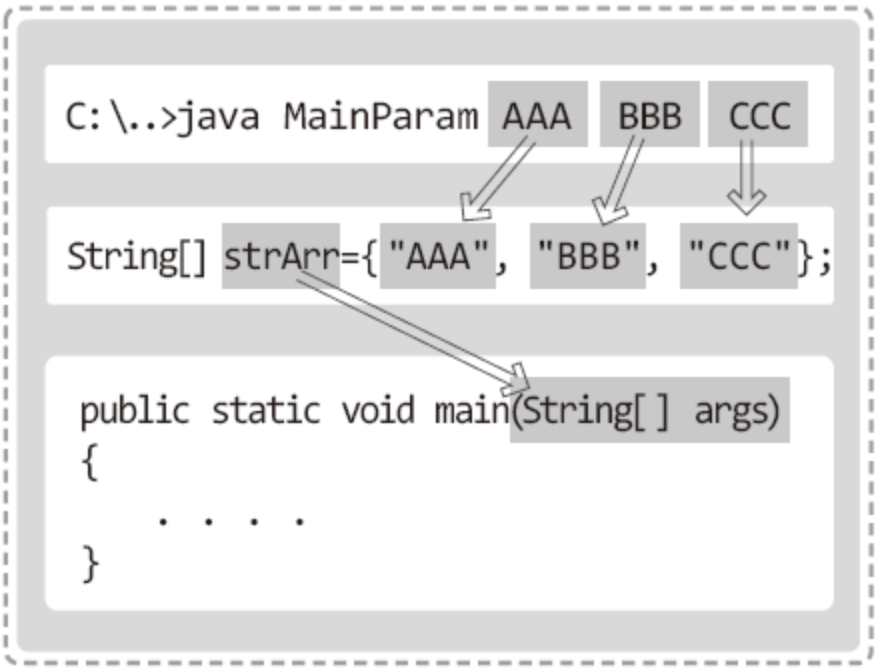
```
public static void main(String[] args) { . . . . }
```

아래에서 보이듯이 main 메소드의 매개변수는 String 인스턴스 배열의 참조 값이 인자로 전달되어야 한다.

```
String[] strArr1={"AAA", "BBB", "CCC"};
String[] strArr2={"public", "static", "void", "main"};
```

# main으로의 데이터 전달방법

```
C:\JavaStudy>java MainParam AAA BBB CCC
AAA
BBB
CCC
```




그림에서 보이듯이 명령 프롬프트상에서 전달되는, 공백으로 구분되는 문자열로 String 배열이 구성되어 이 배열의 참조 값이 전달된다.