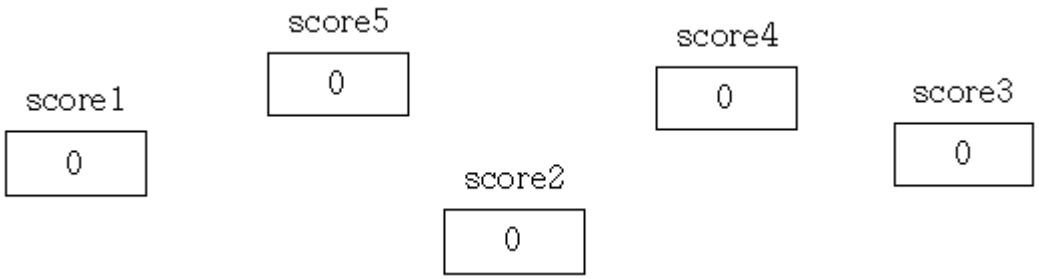
JAVA
- 배열

배열이라는 존재가 필요한 이유

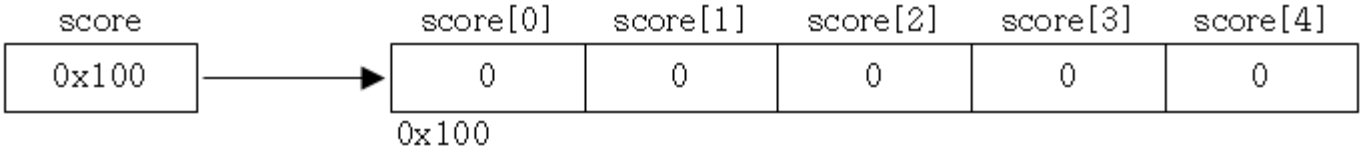
배열(array)이란?

- 같은 타입의 여러 변수를 하나의 묶음으로 다루는 것
- 많은 양의 값(데이터)을 다룰 때 유용하다.
- 배열의 각 요소는 서로 연속적이다.

```
int score1=0, score2=0, score3=0, score4=0, score5=0 ;
```




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



배열의 선언과 생성(1)

- 타입 또는 변수이름 뒤에 대괄호[]를 붙여서 배열을 선언한다.

선언방법	선언 예
타입[] 변수이름;	`int[] score;` `String[] name;`
타입 변수이름[];	`int score[];` `String name[];`

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 1000개라고 가정해 보면, 위의 차이는 더 극명하게 드러난다.

배열의 특성

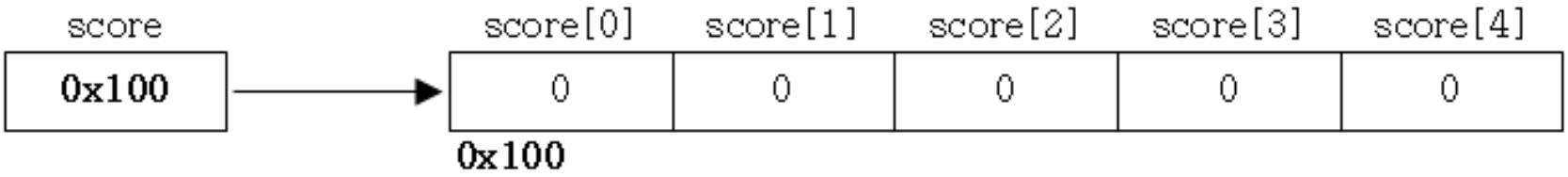
배열을 선언한다고 해서 값을 저장할 공간이 생성되는 것이 아니라 배열을 다루는데 필요한 변수가 생성된다.

int[] score;
배열을 선언한다.(생성된 배열을 다루는데 사용될 참조변수 선언)

score = new int[5];
배열을 생성한다. (5개의 int갑을 저장할 수 있는 저장공간 생성)

int[] score = new int[5];

배열의 특성



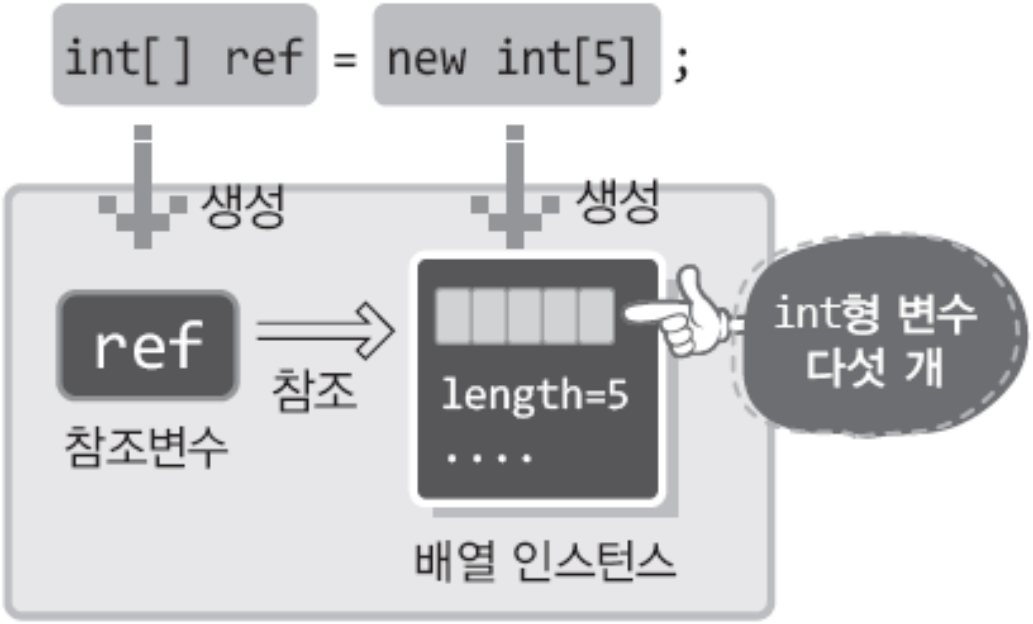
각 요소에 값은 기본값으로 초기화

자료형	기본값
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d 또는 0.0
참조형 변수	null

1차원 배열의 이해와 활용

배열 인스턴스의 생성 방법

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



배열 인스턴스와 참조의 생성 모델

배열 인스턴스의 생성 방법

배열 인스턴스의 생성의 예

```
Int[] arr1 = new int[7];
double[] arr1 = new double[7];
boolean[] arr1 = new boolean[7];
FruitSeller[] arr1 = new FruitSeller[7];
FruitBuyer[] arr1 = new FruitBuyer[7];
```

배열 인스턴스의 생성 방법은

일반적인 인스턴스의 생성 방법과 차이가 있다.

배열의 접근 방법

배열의 접근에는 0부터 시작하는 인덱스 값이 사용된다.

가장 첫 번째 배열 요소의 인덱스가 0이고 N번째 요소의 인덱스가 N-1이다.

배열 인스턴스의 멤버변수 length에는 배열의 길이 정보가 저장되어 있다.

```
class AccessArray {
        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);
        }
}
```

배열의 접근 방법

```
class AccessArray
{
	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);
	}
}
```

배열의 접근 방법

```
class InstanceArray
{
	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]);

	}
}
```

배열의 선언과 동시에 초기화

일반적인 인스턴스 배열의 선언

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

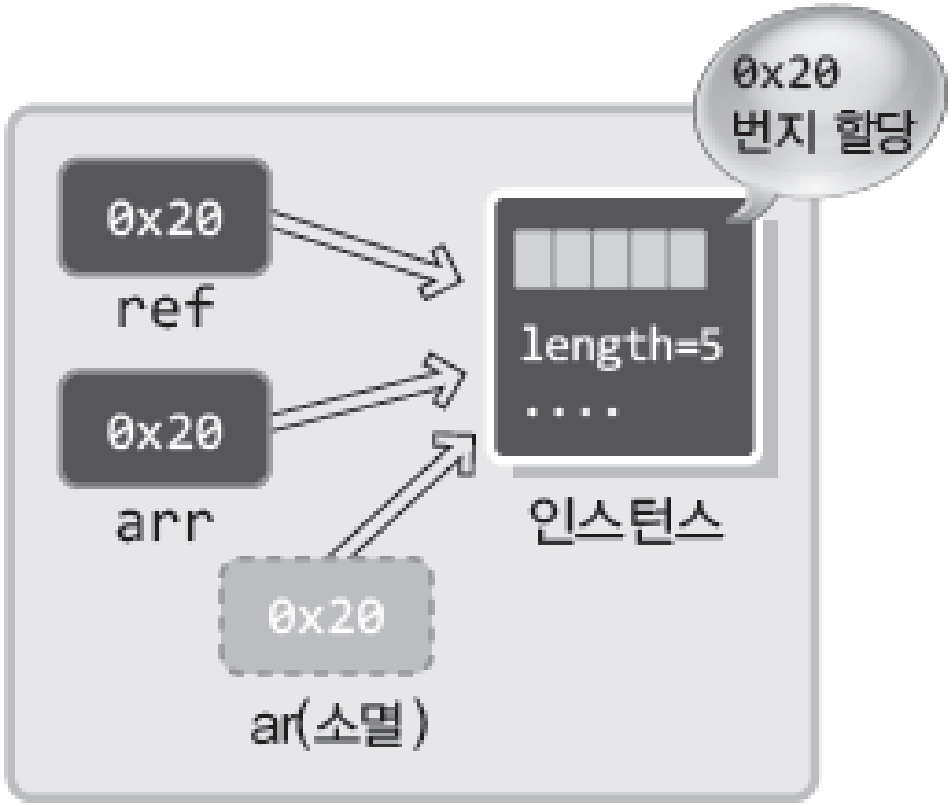
이렇게 줄여서 표현하는 것도 가능하다

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

string[] arrStr = {“월”, “화”, “수”, “목”, “금”, “토”, “일”};
```

배열과 메소드

배열도 인스턴스이기 때문에 메소드 호출시의
인자 전달 및 반환의 과정이
일반적인 인스턴스들과 같다.



배열과 메소드

```
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]+" ");
        }
}
```

배열의 활용

- 배열에 값을 저장하고 읽어오기

```
score[3] = 100;
        //배열 score의 4번째 요소에 100을 저장한다.
int value = score[3];
        //배열 score의 4번째 요소에 저장된 값을 읽어서 value에 저장한다.
```

- '**배열이름.length**'는 배열의 크기를 알려준다.

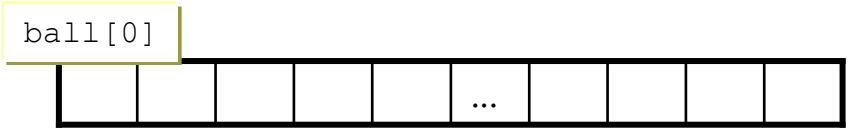
```
int[] score = { 100, 90, 80, 70, 60, 50 };

for(int i=0; i < 6; i++) {
    System.out.println(socre[i]);
}
```



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

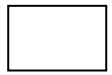
배열의 활용




```
class ArrayEx4 {
        public static void main(String[] args) {

                // 45개의 정수값을 저장하기 위한 배열 생성.
                int[] ball = new int[45];
                // 배열의 각 요소에 1~45의 값을 저장한다.
                for(int i=0; i < ball.length; i++)
                        ball[i] = i+1;    // ball[0]에 1이 저장된다.
                int temp = 0;  // 두 값을 바꾸는데 사용할 임시변수
                int j = 0;       // 임의의 값을 얻어서 저장할 변수
                // 배열에 저장된 값이 잘 섞이도록 충분히 큰 반복횟수를 지정한다.
                // 배열의 첫 번째 요소와 임의의 요소에 저장된 값을 서로 바꿔서 값을 섞는다.
                for(int i=0; i < 100; i++) {
                        j = (int)(Math.random() * 45); // 배열 범위(0~44)의 임의의 값을 얻는다.

                        temp = ball[0];
                        ball[0] = ball[j];
                        ball[j] = temp;
                }
                // 배열 ball의 앞에서 부터 6개의 요소를 출력한다.
                for(int i=0; i < 6; i++)
                        System.out.print(ball[i]+" ");
        }
}
```

temp 

예제

문제 1.

int형 1차원 배열을 매개변수로 전달 받아서 배열에 저장된 최대값,
그리고 최소값을 구해서 반환하는 메소드를 다음의 형태로 각각 정의.

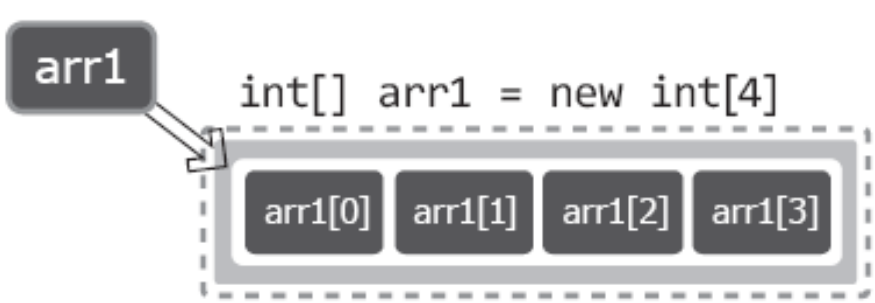
```
public static int miniValue(int[] arr) { . . . . } // 최소값 반환
public static int maxValue(int[] arr) { . . . . } // 최대값 반환
```

위의 두 매소드는 인자로 전달되는 배열의 길이에 상관없이 동작하도록 정의.
두 매소드 실행을 확인인하기 위한 main 메소드 정의.
int형 배열에 채워질 정수는 프로그램 사용자로부터 입력 받고, 배열의 길이는
임의로 결정

다차원 배열의 이해와 활용

1차원배열 vs. 2차원배열

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



1차원배열 vs. 2차원배열

2차원 배열의 선언 방법

가로길이가 2이고, 세로길이가 7인 int형 배열

```
int[][] arr1 = new int[7][2];
```

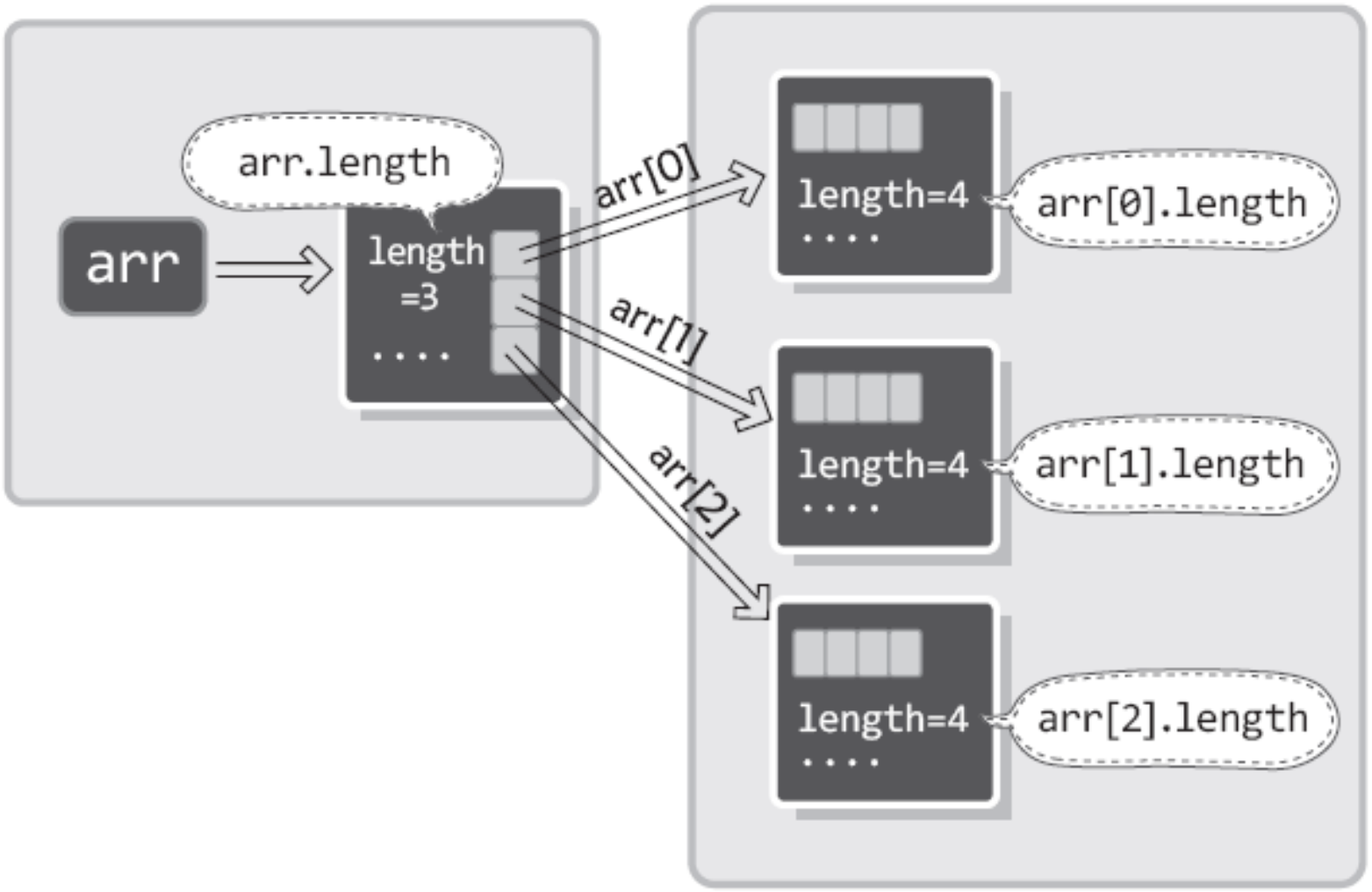
가로길이가 5이고, 세로길이가 3인 double 형 배열

```
double[][] arr2 = new double[3][5];
```

가로길이가 7이고, 세로길이가 3인 String 형 배열

```
double[][] arr3 = new double[3][7];
```

2차원 배열의 메모리 구조



2차원 배열의 메모리 구조

선언방법	선언예
타입[][] 변수이름;	int[][] score;
타입 변수이름[][];	int score[][];
타입[] 변수이름[];	int[] score[];

[표5-3] 2차원 배열의 선언

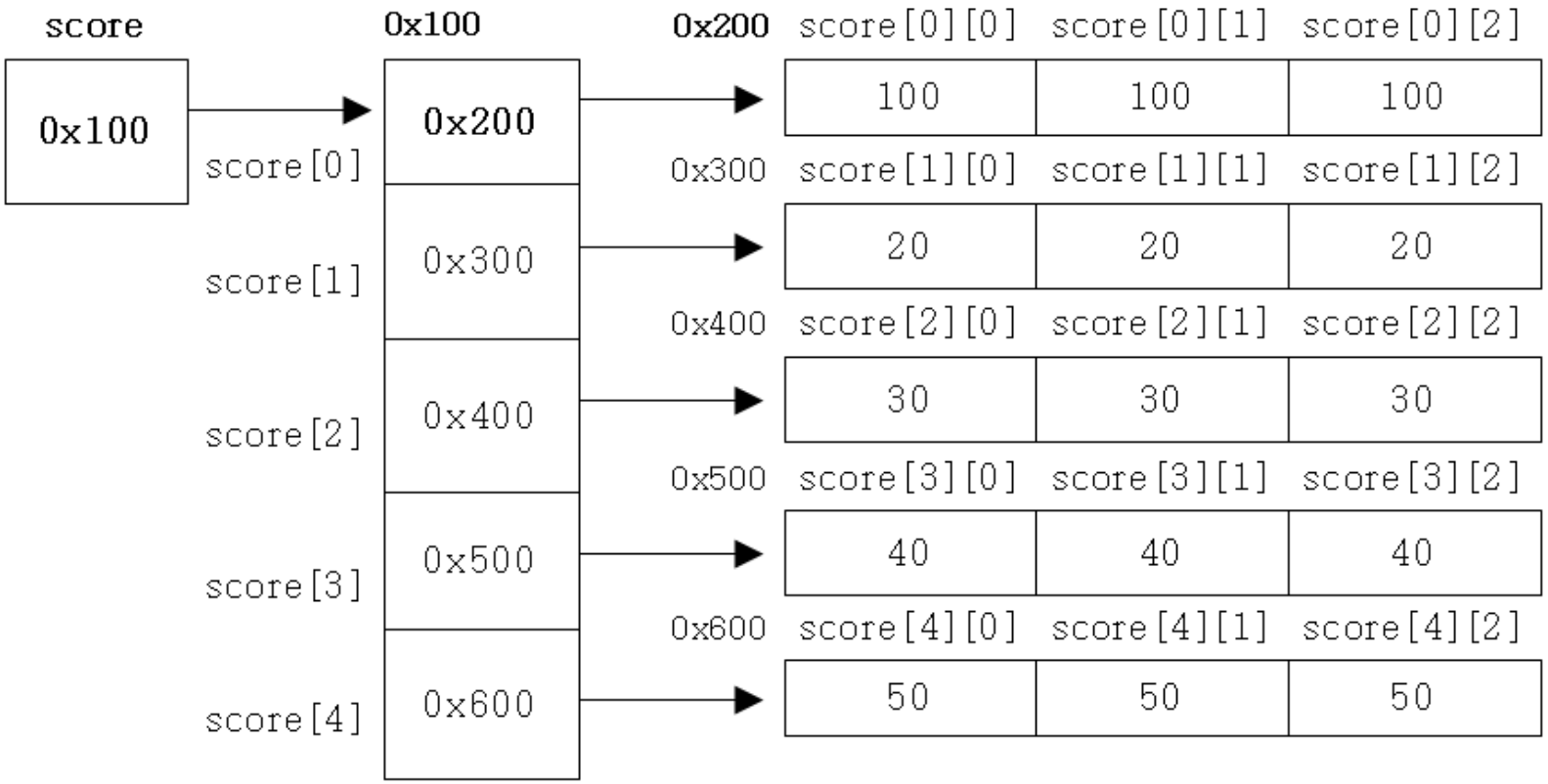
```
int[][] score = {
                    {100,100,100},
                    { 20, 20, 20},
                    { 30, 30, 30},
                    { 40, 40, 40},
                    { 50, 50, 50},
                };
```

```
int[][] score = new int[5][3];      // 5행 3열의 2차원 배열을 생성한다.
```

2차원 배열의 메모리 구조

	국어	영어	수학
1	100	100	100
2	20	20	20
3	30	30	30
4	40	40	40
5	50	50	50

```
for (int i=0; i < score.length; i++) {
    for (int j=0; j < score[i].length; j++) {
        score[i][j] = 10;
    }
}
```



예제를 통한 2차원 배열 구조의 이해

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

		for(int i=0; i<3; i++)
			for(int j=0; j<4; j++)
				arr[i][j]=i+j;

		for(int i=0; i<3; i++)
		{
			for(int j=0; j<4; j++)
				System.out.print(arr[i][j]+" ");

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

2차원 배열- 가변 배열

다차원 배열에서 마지막 차수의 크기를 지정하지 않고 각각 다르게 지정.

```
int[][] score = new int[5][3];      // 5행 3열의 2차원 배열을 생성한다.
```

```
int[][] score = new int[5][];
score[0] = new int[3];
score[1] = new int[3];
score[2] = new int[3];
score[3] = new int[3];
score[4] = new int[3];
```

```
int[][] score =
    {
        {100,100,100},
        { 20, 20, 20},
        { 30, 30, 30},
        { 40, 40, 40},
        { 50, 50, 50},
    };
```

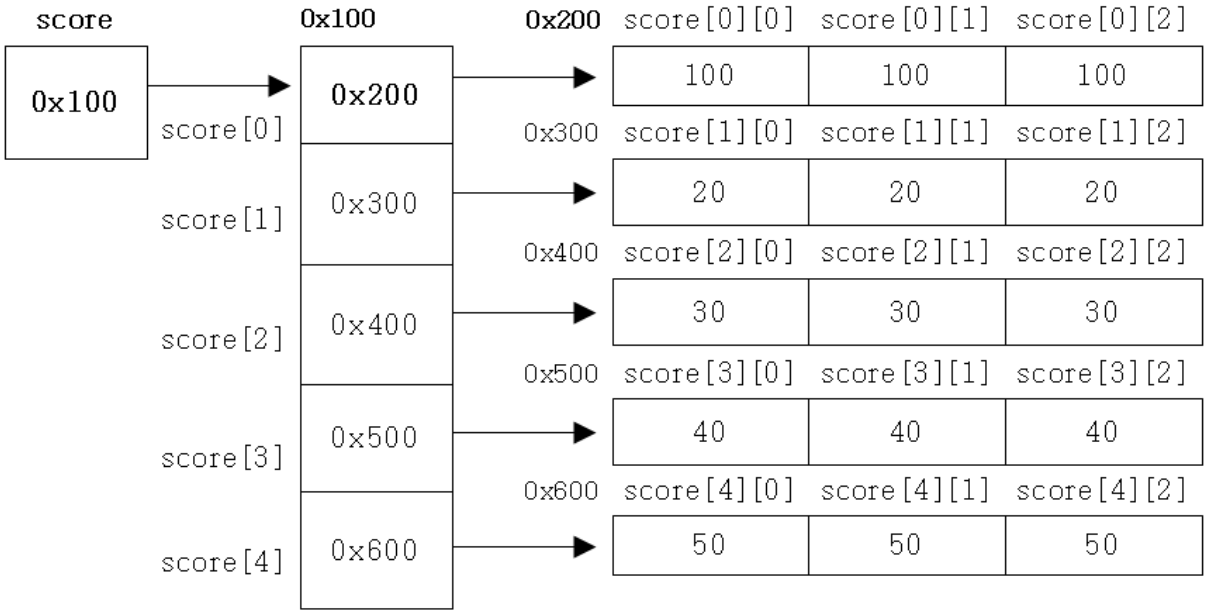
2차원 배열- 가변 배열

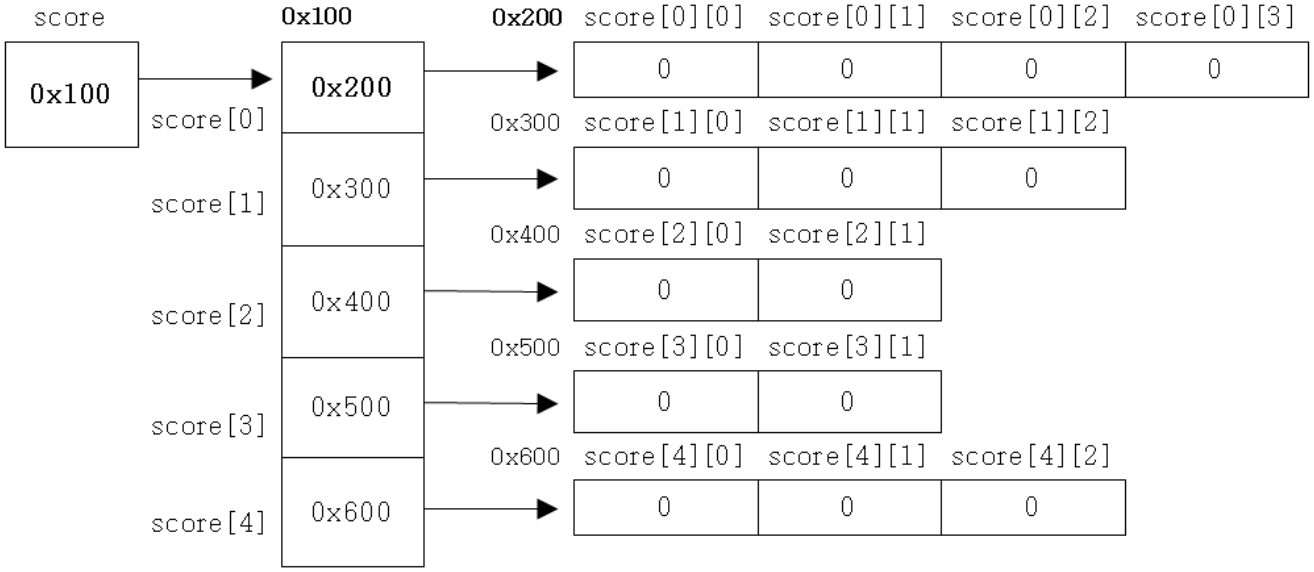
다차원 배열에서 마지막 차수의 크기를 지정하지 않고 각각 다르게 지정.

```
int[][] score = new int[5][];
score[0] = new int[4];
score[1] = new int[3];
score[2] = new int[2];
score[3] = new int[2];
score[4] = new int[3];
```

```
int[][] score =
    {
        {100,100,100,100},
        { 20, 20, 20},
        { 30, 30},
        { 40, 40},
        { 50, 50, 50},
    };
```

2차원 배열- 가변 배열


score
0x100
0x100
score[0]
score[1]
score[2]
score[3]
score[4]
0x200
0x300
0x400
0x500
0x600
0x200 score[0][0] score[0][1] score[0][2]
100 100 100
0x300 score[1][0] score[1][1] score[1][2]
20 20 20
0x400 score[2][0] score[2][1] score[2][2]
30 30 30
0x500 score[3][0] score[3][1] score[3][2]
40 40 40
0x600 score[4][0] score[4][1] score[4][2]
50 50 50


score
0x100
0x100
score[0]
score[1]
score[2]
score[3]
score[4]
0x200
0x300
0x400
0x500
0x600
0x200 score[0][0] score[0][1] score[0][2] score[0][3]
0 0 0 0
0x300 score[1][0] score[1][1] score[1][2]
0 0 0
0x400 score[2][0] score[2][1]
0 0
0x500 score[3][0] score[3][1]
0 0
0x600 score[4][0] score[4][1] score[4][2]
0 0 0

2차원 배열의 선언 및 초기화 1

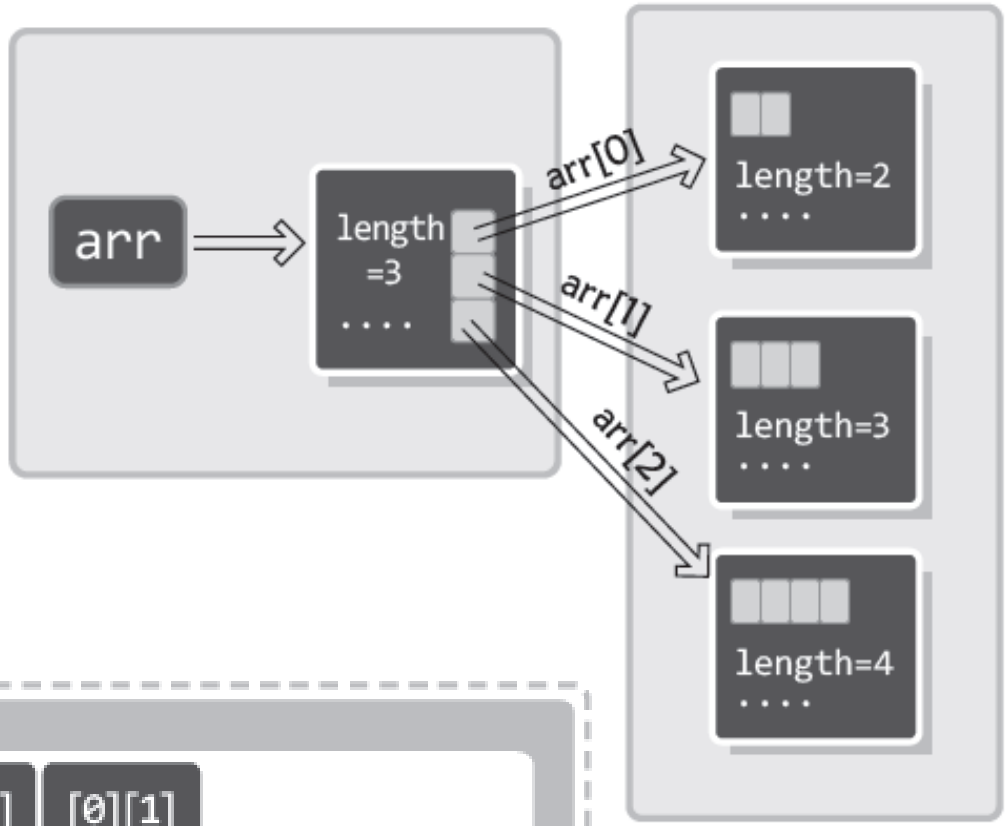
```
int[][]  arr={
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
};
```

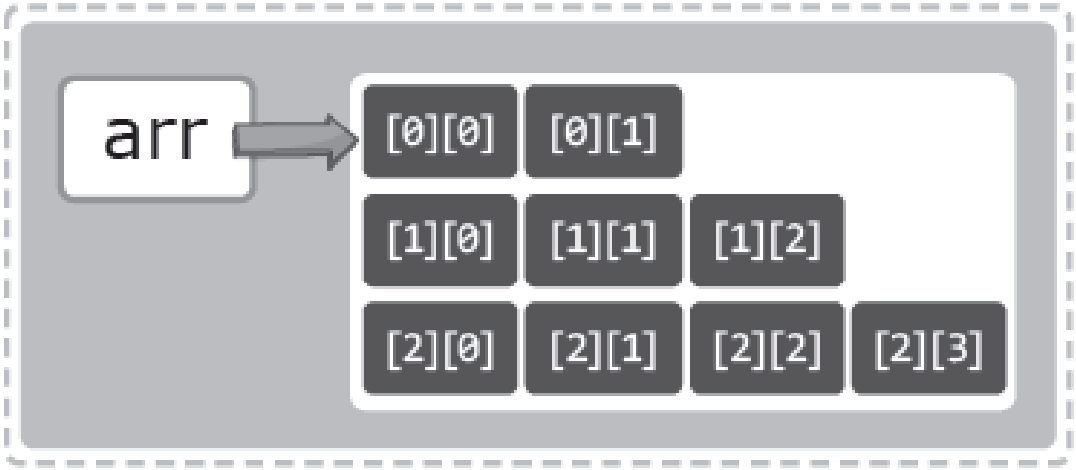
2차원 배열의 선언 및 초기화 2

```
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}
};
```





2차원 배열의 선언 및 초기화 3

```
class ArrayTest
{
	public static void main(String[] args)
	{
		int[][] arr={
			{1, 2},
			{3, 4, 5},
			{6, 7, 8, 9}
		};

		System.out.println("배열의 세로길이: "+arr.length);

		for(int i=0; i<arr.length; i++)
			System.out.printf("%d행의 길이: %d ₩n", i+1, arr[i].length);
	}
}
```

예제

문제 2.
다음 메서드는 int형 1차원 배열에 저장된 값을 두 번째 매개변수로 전달될 값의 크기만큼 전부 증가시킨다.

```
public static void addOneDArr(int[] arr, int add)
{
        for (int i=0, i<arr.length; i++)
                arr[i] +=add;
}
```

DI 메서드를 기반으로(이 메서들르 호출하는 형태로)int형 2차원 배열에 저장된 값 전부를 증가시키는 메서드를 다음의 형태로 정의하자.

```
public static void addOneDArr(int[][] arr, int add) { . . . . . }
```

단 위 메서드는 2차원 배열의 가로, 세로 길이에 상관없이 동작해야 하며, 위의 메서드가 제대로 동작하는지 확인하기 위한 main 메서드도 함께 정의해야 한다.

예제

문제 3.
다음의 형태로 표현된 2차원 배열이 존재한다고 가정해 보자.

1	2	3
4	5	6
7	8	9

이러한 형태를 갖는 int형 2차원 배열이 인자로 전달되면, 다음의 형태로 배열의 구조를 변경시키는 메서드를 정의해 보자.

7	8	9
1	2	3
4	5	6

즉 총 N행으로 이뤄진 2차원 배열이 존재한다면, 메서드 호출 시, 1행은 2행으로 이동이 이뤄져야한다. 이번에도 마찬가지로 배열의 가로, 세로길이에 상관 없이 동작을 하도록 메서드가 정의되어야 하며, 정의된 메서드의 확인을 위한 main메서드도 함께 정의하자.

for-each문

for-each문의 이해와 활용

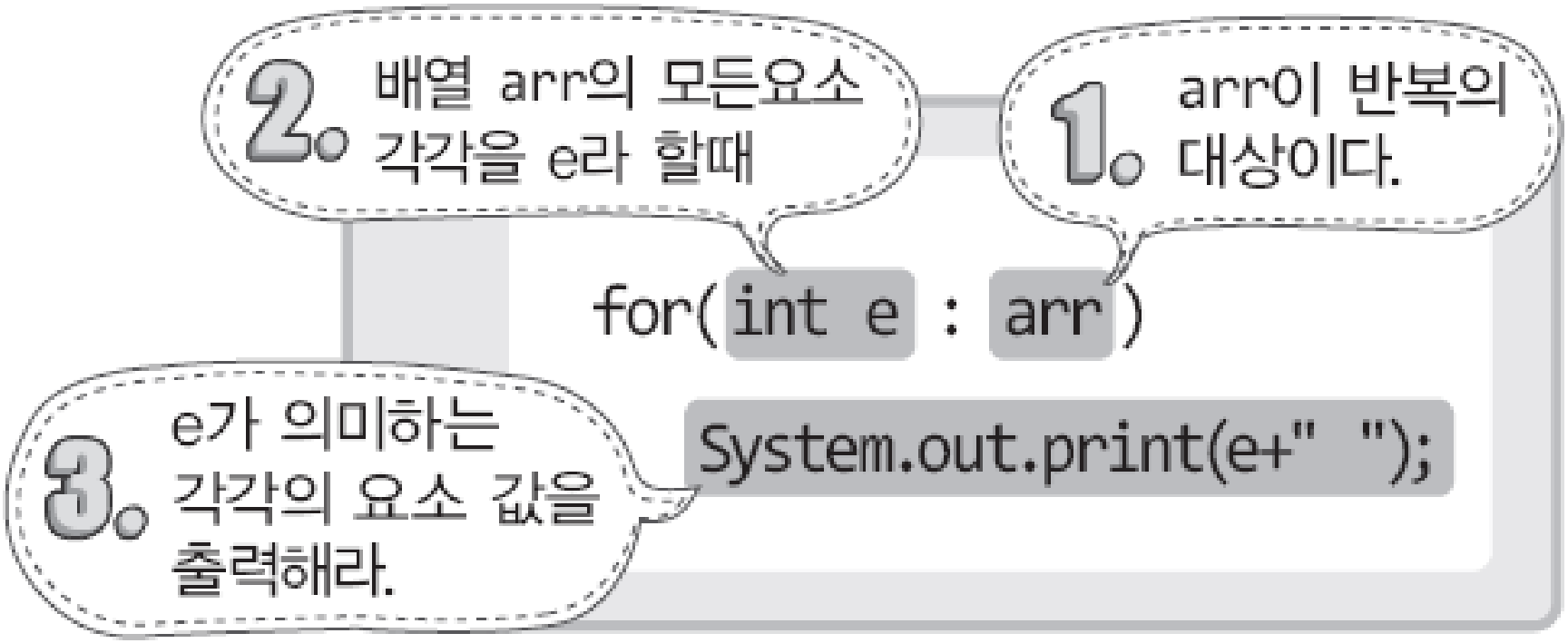
배열의 일부가 아닌, 배열의 전체를 참조할 필요가 있는 경우에 유용하게 사용할 수 있다.

```
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-each문의 이해와 활용



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

for-each문의 이해와 활용

```
class EnhancedFor {
        public static void main(String[] args)     {
                int[] arr={1, 2, 3, 4, 5};

                int sum=0;
                for(int e: arr)     sum+=e;

                System.out.println("배열 요소의 합: "+sum);

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

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

인스턴스 배열에 대한 for-each문

인스턴스 배열에 저장된 참조 값의 변경은 불가능 하지만,

참조 값을 통한 인스턴스의 접근은 가능하다!

(접근 과정에서의 데이터 변경)

인스턴스 배열에 대한 for-each문

```
class Number
{
        public int num;
        public Number(int num)
        {
                this.num=num;
        }
        public int getNum()
        {
                return num;
        }
}
```

```
class EnhancedForInst
{
        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()+" ");
        }
}
```

```
3 5 9
7 7 7
3 5 9
```

인스턴스 배열에 대한 for-each문

```
class Number{
        public int num;
        public Number(int num)
        {
                this.num=num;
        }
        public int getNum()
        {
                return num;
        }
}

class EnhancedForInst{
        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++;
```

인스턴스 배열에 대한 for-each문

```
                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()+" ");
        }
}
```

예제

문제 4.

문제 1을 for-each 문 형태로 변경.

예제

```
class ArrayEx1 {
	public static void main(String[] args)
	{
		int sum =0;				// 총점을 저장하기 위한 변수
		float average = 0f;			// 평균을 저장하기 위한 변수

		int[] score = {100, 88, 100, 100, 90};

		for (int i=0; i < score.length ; i++ ) {
			sum += score[i];
		}

		average = sum / (float)score.length ;
		// 계산결과를 float로 얻기 위함.

		System.out.println("총점 : " + sum);
		System.out.println("평균 : " + average);
	}
}
```

예제

```
class ArrayEx2
{
	public static void main(String[] args)
	{
		int[] score = { 79, 88, 91, 33, 100, 55, 95};

		int max = score[0]; // 배열의 첫 번째 값으로 최대값을 초기화 한다.
		int min = score[0]; // 배열의 첫 번째 값으로 최소값을 초기화 한다.

		for(int i=1; i < score.length;i++) {
			if(score[i] > max) {
				max = score[i];
			}
			if(score[i] < min) {
				min = score[i];
			}
		}	// end of for

		System.out.println("최대값 :" + max);
		System.out.println("최소값 :" + min);
	} // end of main
} // end of class
```

예제

```
class ArrayEx3 {
        public static void main(String[] args)
        {
                int[] number = new int[10];
                for (int i=0; i < number.length ; i++ ) {
                        number[i] = i;  // 배열을 0~9의 숫자로 초기화한다.
                        System.out.print(number[i]);
                }

                System.out.println();

                for (int i=0; i < 100; i++ ) {
                        int n = (int)(Math.random() * 10);
                        int temp = number[0];
                        number[0] = number[n];
                        number[n] = temp;
                }

                for (int i=0; i < number.length ; i++ ) {
                        System.out.print(number[i]);   // 배열의 내용을 출력한다.
                }
        }
}
```

예제

```
class ArrayEx5 {
	public static void main(String[] args) {
		int[] number = new int[10];
		for (int i=0; i < number.length ; i++ ) {
			System.out.print(number[i] = (int)(Math.random() * 10));
		}
		System.out.println();
		for (int i=0; i < number.length ; i++ ) {
			boolean changed = false; // 자리바꿈이 발생했는지를 체크한다.
			for (int j=0; j < number.length-1-i ; j++ ) {
				if(number[j] > number[j+1]) {
					// 옆의 값이 크면 서로 바꾼다.
					int temp = number[j];
					number[j] = number[j+1];
					number[j+1] = temp;
					changed = true; // 자리바꿈: changed true로.
				} // end if
			} // end for j
			if (!changed) break; // 자리바꿈이 없으면 반복문을 벗어난다.
			for(int k=0; k<number.length;k++)
				System.out.print(number[k]); // 정렬된 결과를 출력한다.
			System.out.println();
		} // end for i
	}
}
```

예제

```
class ArrayEx8
{
	public static void main(String[] args)
	{
		String src = "ABCDE";

		for(int i=0; i < src.length(); i++)
			System.out.println("src.charAt("+i+"):"+ src.charAt(i));
	}
}
```

예제

System.arraycopy(복사할 원본, 원본의 시작 index,
복사해서 넣을 배열,
복사해서 넣을 시작 index ,
원본의 요소 개수)

배열의 복사

```
class ArrayEx12 {
	public static void main(String[] args)
	{
		char[] abc = { 'A', 'B', 'C', 'D'};
		char[] number = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
		System.out.println(new String(abc));
		System.out.println(new String(number));
		// 배열 abc와 number를 붙여서 하나의 배열(result)로 만든다.
		char[] result = new char[abc.length+number.length];
		System.arraycopy(abc, 0, result, 0, abc.length);
		System.arraycopy(number, 0, result, abc.length, number.length);
		System.out.println(new String(result));
	// 배열 abc을 배열 number의 첫 번째 위치부터 배열 abc의 크기만큼 복사
		System.arraycopy(abc, 0, number, 0, abc.length);
		System.out.println(new String(number));
	    // number의 인덱스6 위치에 3개를 복사
		System.arraycopy(abc, 0, number, 6, 3);
		System.out.println(new String(number));
	}
}
```

정리합시다

- 배열과 배열을 다루는 목적
- 1차원 배열
- 2차원 배열
- 참조형 변수를 다루는 1차원 배열

Project

Project : ver 0.30

배열을 이용해서 프로그램 사용자가 입력하는 정보가 최대 100개까지
유지되도록 프로그램을 변경. 아래기능 삽입

저장 : 이름, 전화번호, 생년월일 정보를 대상으로 하는 저장
검색 : 이름을 기준으로 데이터를 찾아서 해당 데이터의 정보를 출력
삭제 : 이름을 기준으로 데이터를 찾아서 해당 데이터를 삭제

데이터 삭제 후 남아있는 데이터 처리는 데이터를 빈 공란이 없이 순차적으로
재정리 2번이 삭제되었다면 3번 이후 데이터들의 주소 값이 -1 처리되어 재저장.