

# JAVA 5강 배열

강사 박주병

## 1. 메모리

배열을 배우기 전에 컴퓨터의 메모리에 관해서 알아둬야 한다. 앞으로 배열 과 객체에 대해 설명할 때 메모리를 보여주며 설명이 필요하기 때문이다.

그림1은 컴퓨터에 반드시 필요한 3가지이다. 기본적으로 프로그램 및 각종 데이터들은 HDD, SSD인 저장장치에 기록이 된다. 컴퓨터를 켜거나 프로그램이 실행될 때 저장장치에 기록된 프로그램 실행에 필요한 데이터들이 메모리로 이동이 된다.

즉 메모리에 데이터들이 올라가 있다는 말은 해당 프로그램이 실행중이라는 것이다.

그리고 메모리에 있는 데이터(프로그램 실행에 필요한 소스코드, 정확히는 소스코드를 컴파일한 바이너리 정보)를 CPU가 하나씩 가져와 순서대로 계산하고 그 결과를 다시 메모리에 저장한다. 이 작업이 프로그램이 실행되는 과정이다.



그림 1

우리가 자바 소스코드를 작성하며 변수를 생성한다는 것은 바로 메모리에 특정 공간을 할당하여 값을 보관해두는 것이다.

## 2. 배열

지금까지는 변수를 만들 때 변수마다 이름을 지정해줬어야 했다. 하지만 학생 100명의 성적을 다루기 위한 변수 100개를 만들려면 어떻게 해야 하는가?

```
int student1=10;
int student2=30;
int student3=100;
int student4=50;
int student5=35;
int student6=70;
```

그림 2

그림2처럼 변수명 뒤에 숫자를 붙여가면서 100개의 변수를 만들어야 할 것이다. 이렇게 하면 변수를 만드는것도 번거로울뿐만 아니라 100명의 성적을 출력 하고 싶으면 반복문을 이용 할 수도 없다. 변수 이름이 다 다르기 때문이다.

그래서 이러한 상황일 때 여러개의 변수를 한번에 생성하며 일관된 이름으로 사용 할 수 있게 해주는 것이 배열이다.

### 가. 배열의 선언과 생성

```
int[] student = new int[100];
int student2[] = new int[100];
```

그림 3

```
int[] student = new int[100];
int student2[] = new int[100];
```

그림 4

그림3의 빨간색 부분을 보면 일반적인 변수를 만들 듯이 데이터 타입을 작성후[] 대괄호를 적어주었다. 이것이 int타입의 변수를 여러개 생성하겠다는 것이다. 그리고 student는 배열의 이름이다. 이렇게 student라는 배열을 선언하였다.

하지만 이렇게 만해서는 배열을 사용 할 수 없다. 그림4의 빨간 네모를 보면 new 연산자와 데이터 타입 그리고 [] 사이에 숫자 100을 적어주었다. 이 부분이 바로 배열을 생성하는 것이다.

정리하자면 int 타입의 변수 100개를 배열로 만들어 student라는 변수에 저장을 한 것이다.

배열은 이렇게 선언만 하고서 바로 사용하지 못한다. 반드시 new 키워드를 사용해 생성을 한 후에 사용해야 한다.

```
int[] student;  
  
student = new int[100];
```

그림 5

그림5는 배열은 담아둘 student변수를 만들고 다음 줄에서 배열을 생성 후 초기화 하고 있다. 일반변수와 마찬가지로 변수를 선언하고 초기화 하는 것이 분리될수 있다.

#### 나. 배열의 요소

배열을 사용 할 때는 대괄호를 이용해 몇 번째 변수에 접근할 것 인가를 지정한다. 이렇게 배열안에 하나하나의 변수들을 보고 요소라고 부른다. 즉 student배열은 int 타입의 요소 100개를 가지고 있는 것이다.

```
int[] student= new int[5]; // 배열 생성
```

```
student[0] =50; //값 초기화  
student[1] =30;  
student[2] =70;  
  
System.out.println(student[2]);
```

그림 6

## 다. 배열의 메모리 구조

배열은 일반변수 와는 저장 구조가 다르다. 1장에서 변수를 배울 때 기본형, 참조형을 배운적이 있다.

기본형은 지금까지 흔히 써왔던 int, char 같은 것들이다. 이러한 기본형 변수들은 변수에 직접 데이터를 가지고 있는 형태이다.

```
int score = 10;
```

score

메모리주소	값
0x000A	10
0x000B	
0x000C	
0x000D	
0x000E	
0x000F	

그림 7

그림7의 score 라는 변수는 숫자 10을 저장하고 있다. 메모리에서는 score변수의 영역에 직접 데이터10을 보관하고 있는중이다.

```
int[] score = new int[100];
```

score

메모리주소	값
0x000A	0x000B
0x000B	score[0]
0x000C	score[1]
0x000D	score[2]
0x000E	score[3]
0x000F	score[4]

그림 8

그림8을 보면 score 라는 변수를 만들었다. 해당 변수의 타입은 배열 타입이므로 배열을 저장해 둘 수 있는 변수이다. 배열타입 역시 참조타입이므로 score는 참조형 변수라고 할 수 있다. 참조형 변수의 특징은 직접 데이터를 가지고 있지 않다는 것이다.

그림8의 score의 값을 보면 0x000B 라고 되어 있다. 실제 값이 아닌 값의 위치를 나타내는 주소를 가지고 있다. 0x000B가 실질적인 배열의 요소를 가지고 있는 시작 주소이다.

new int[100]를 이용해 배열을 생성하는 순간 0x000B부터 100개의 데이터를 보관할 수 있는 영역을 할당하는 것이다. 즉 0x000B부터 시작해서 100개의 메모리 영역이 실질적인 배열 데이터인 것이다. 그리고 score 변수는 해당 배열의 시작 주소를 가지고 있는 변수이다.

## 라. NULL

배열을 다루면서 부터는 참조형을 다루는 것이기에 NULL이라는 것을 알아야 한다. NULL을 한마디로 정의 하자면 아무것도 없음을 의미한다.

```
int[] student;
```

그림9의 예시를 보면 student를 사용하려면 에러가 발생한다. 앞에서 배운 new 연산자를 이용해 배열을 생성하지 않았기 때문이다.

```
student[0] = 50;
```

student 변수는 new 연산자를 이용해 배열을 생성한후 그 시작 주소값으로 초기화 해주어야 하지만 배열을 생성하지 않았기에 참조형 변수의 기본값인 null이라는 값을 가진다.

null은 아무것도 없음을 의미하기에 이 상태에서 student를 이용해 배열을 사용하려 한다면 nullPoint 에러가 발생한다.

그림 9

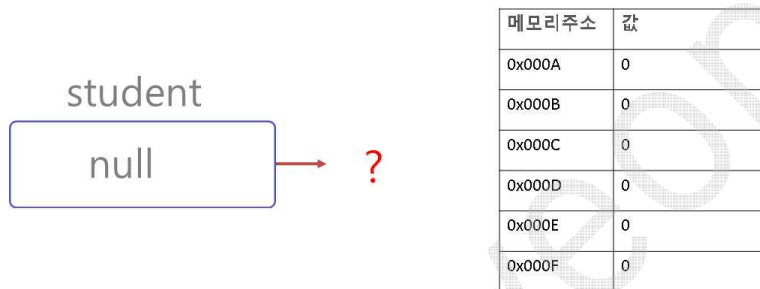


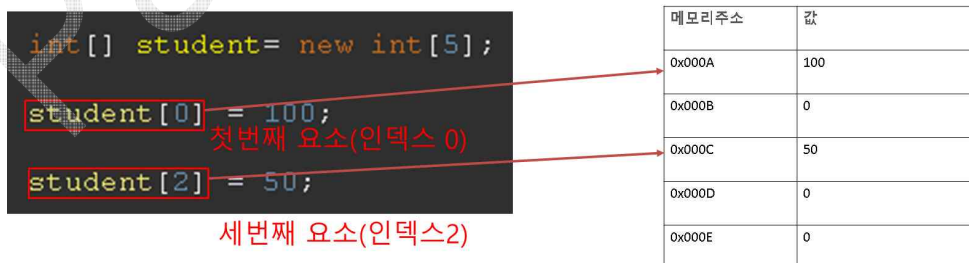
그림 10

## 마. 배열의 접근방법

배열을 저장할 변수를 선언하고 배열을 생성하여 해당 변수에 주소값을 넣어두면 이제부터 배열을 사용 할 수 있다.

사용 할 때는 변수명[인덱스번호]로 사용을 한다. 인덱스라하면 배열의 요소를 가리키는 번호를 의미한다.

주의해야 할 것은 배열은 0부터 시작을 한다는 것이다. 그래서 인덱스를 0으로 지정해야 첫 번째 요소를 접근하는 것이다.



배열은 인덱스를 지정하여 일반변수와 동일하게 사용 할 수 있다. 대괄호 안의 인덱스번호는 리터럴 뿐만 아니라 변수나 계산식을 적어 넣어도 된다. 최종적인 결과만 인덱스 범위 안의 정수이면 된다.

```
for(int i =1 ; i<100 ; i++)
    student[i-1] = i;
```

연산의 결과가 인덱스 범위에 속하면 된다.

그림 12

바. 배열의 길이

배열을 생성할 때 대괄호에 배열의 크기를 지정한다. `int[] arr = new int[5];` 이라고 한다면 길이5(요소의 개수가 5개)의 배열을 생성한 것이다.

그러나 인덱스는 0부터 시작하기 때문에 사용가능한 인덱스 번호는 0~4이다. 따라서 생성시 `new int[5]`로 배열을 생성하여도 사용할 때 `student[5] = 30;`처럼 사용을 하면 인덱스의 범위를 넘어서 에러가 발생한다.

```
int[] student = new int[5];

student[0] =0;
student[1] =10;
student[2] =20;
student[3] =30;
student[4] =40;
student[5] =50;
```

배열의 범위를 넘어서 ERROR!

그림 13

사. 길이0의 배열

```
int[] student= new int[0];
student[0] = 10;
```

그림 14

그러면 사이즈가 0이면 배열 생성은 가능하지만 사용할 수는 없다는 것이다. 그렇다면 이런 기능은 왜 있는것일까? `int[] student;` 이렇게 배열을 가리킬 변수만 생성해 놓은것과 길이0의 배열을 만든 것은 기능적으로는 차이가 없다. 둘다 쓸 수 없기 때문이다.

하지만 길이0배열은 어찌되었든 배열을 생성하여 메모리의 시작주소를 할당한 채 변수에 주소값을 넣어둔 상태이다. 따라서 해당배열의 요소는 사용 할 수 없지만 그 외에 배열에서 기본적으로 제공해주는 기능들은 사용이 가능하다.

그렇다면 무슨 기능들이 있는것일까? 여러 가지 기능들이 있지만 가장 기본적인 배열의 길이 정보를 가지고 있는 `length`변수에 대해 살펴보자.

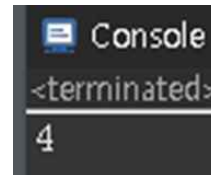
## 아. 배열의 length 변수

`int[] student = new int[5];`에서 `student` 변수는 참조형 변수라고 하였다. 이런 참조형 변수는 객체를 저장 할 수 있는 변수이다. 아직 객체에 대해 자세히 배우지 않았지만 일반적인 데이터와는 다르게 여러개의 변수와 다양한 기능들을 함께 가질 수 있는 데이터 이다.

배열 역시도 사실은 객체 이기 때문에 여러 가지 기능들을 가지고 있는 것이다.

```
int[] student = new int[4];  
  
System.out.println(student.length);
```

그림 15



Console  
<terminated>  
4

그림 16

변수명 뒤에 `.`(마침표)를 작성후 `length` 라고 사용하면 바로 `student` 변수가 가리키고 있는 배열의 `length` 변수를 가져 오는 방법이다.

배열뿐만 아니라 모든 객체들의 이용 방법이 바로 객체를 가리키고 있는 변수명 뒤에 마침표를 붙여서 객체 내부에 들어있는 변수들과 기능들을 사용 하는 것이다.

그림16을 보면 `length`의 값이 4임을 알 수 있다. 즉 `length`는 해당 배열의 요소의 개수를 나타내는 변수이다. 그리고 `length` 변수는 읽기 전용인 상수로 선언되어 있으므로 임의의 값으로 초기화 시킬 수 없다.

```
int[] student = new int[5];  
  
for(int i = 0 ; i < 5; i++)  
    System.out.println(student[i]);
```

그림 17

그림17처럼 배열을 반복을 이용해 접근할 때 배열의 길이를 리터럴로 직접 적어주는 것보다 `length` 변수를 활용하는 것이 코드의 유지보수 측면에서 효과적이다.

잠시 아까전의 주제로 되돌아 가보자면 길이0의 배열의 필요성에 대해 다시 한번 생각해 보자.

`int[] student;`로 변수만 생성하는 것과 `int[] student = new int[0];`으로 길이0의 배열을 생성한 것의 차이가 바로 배열 내부에 있는 변수나 기능들을 활용할 때 나타난다.

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

그림 18

`NullPointerException` 에러가 발생한다. 그러나 `student`를 길이0의 배열로 초기화 시켜 놓았다면 `length`가 0이 들어있으므로 에러 없이 그냥 반복문이 실행되지 않고 넘어 갈 것이다.

배열을 가리키는 변수를 만드는 때에 아직 배열의 길이가 몇으로 생성되어야 하는지 정해지지 않고 차후에 배열 길이가 정해져서 생성되는 상황일 때 일단은 안전하게 길이0의 배열을 생성해서 넣어둔다면 그 뒤의 코드들에서는 항상 `null`인지 아닌지를 체크할 필요가 없어진다.

그림18처럼 반복문 사용시 `length` 변수를 활용 할 때 만약 `student` 변수만 선언해놓고 사용한다면 어떻게 될까?

`student` 변수는 `null`을 가지고 있기 때문에 해당 변수 내부에는 아무것도 없는 상태 임으로 `length` 사용시



## 자. 배열의 요소 초기화

```
int a;  
  
System.out.println(a);
```

그림 19

자바에서는 변수(지역변수)를 초기화 하지 않으면 사용할 수 없다. 그림19와 같이 변수를 만든후 초기화 하지 않은채로 그냥 값을 사용하려하면 컴파일 에러가 발생할 것이다. 이는 지역변수에 쓰레기값이 들어가 있기 때문에 그냥 사용 할 수 없게 해놓은 것이다.

```
int[] student= new int[3];  
  
for(int i =0 ; i<student.length;i++)  
    System.out.println(student[i]);
```

그림 20

지금까지 배운 대로라면 이 역시도 값을 사용하려 할 시 컴파일 에러가 발생 해야만 한다. 하지만 직접 코드를 작성해서 테스트 해보면 알겠지만 컴파일이 되며 심지어 int배열 이라면 0이 출력될 것이다. 배열의 요소들은 일반적인 변수들과는 다르게 초기화를 하지 않아도 해당 타입의 기본 값들로 초기화가 되어 있다.

그러나 그림20의 예시를 보자. student는 int타입의 요소를 가지는 배열이다. 그리고 반복문을 통해 요소의 값을 출력해보고 있는데 가만히 보면 new int[3]를 이용해 배열을 만들기는 했지만 배열의 요소에 대해서는 초기화를 하지 않았다. student[0] = 30;처럼 말이다.

```
int[] student = new int[5];  
  
student[0] = 30;
```

그림 21

그림21과 같이 배열을 만든후 초기화를 해도 되지만 배열의 길이가 길다면 값을 초기화 하는것도 번거로울수 있다. 그래서 배열을 생성할 때 간편하게 초기화 해줄수 있는 여러 가지 방법들을 제공해 준다.

```
int[] student= new int[] {30,20,10};
```

그림 22

그림22처럼 배열을 생성할 때 바로 중괄호 사이에 초기화 할 값을 여러개 지정 해줄수 있다. 값은 요소의 순서대로 자동으로 초기화를 해준다.(0번째 요소에 30, 두 번째요소에 20, 세 번째요소에 10)

여기서 주의해 줘야 하는 것은 배열을 이렇게 초기화 할 시 new int[3] {30,20,10}; 과 같이 배열의 길이를 지정해 주면 컴파일 오류이다. 초기화 할 값들의 개수로 요소의 개수가 파악이 되기 때문에 굳이 적어줄 필요도 없고 적는다 하더라도 컴파일 에러라는 점을 주의하자.



```
int[] student = {30, 20, 10};
```

그림 23

그림23처럼 new 와 int를 생략하여도 된다.

하지만 student변수의 생성과 {30,20,10}의 값 초기화 코드가 다른 라인이 되면 new 와 int는 생략할수 없다.

```
int[] student;
student = {30, 20, 10};
```

선언과 초기화 분리시 new int[] 를 생략할 수 없다. ERROR!

#### 차. 배열의 길이 증가

배열 길이 5를 사용하던 중 늘릴 필요가 있을 때 어떻게 해야 할까?

```
int[] student = new int[5];

student[0]=10;
student[1]=50;
student[2]=30;

student = new int[10];
```

그림25와 같이 student배열을 사용도중 길이가 10으로 늘어날 필요가 있어 길이10짜리 배열을 새로 생성하여 해당 주소값을 student에 넣어서 쓰면될까?

안타깝지만 이렇게 할 경우 기존에 저장해두었던 데이터들은 모두 없어진다.

그림 25

```
int[] student = new int[5];

student[0]=10;
student[1]=50;
student[2]=30;

student = new int[10];

System.out.println(student[0]);
System.out.println(student[1]);
System.out.println(student[2]);
```

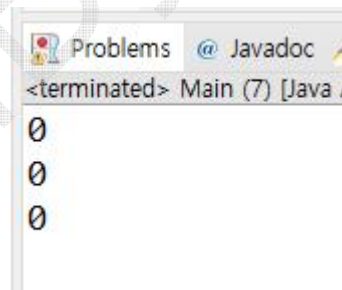


그림 27 값이 0으로 사라졌다.

그림 26

이렇게 되는 이유는 사실 앞서 배운 것을 잘 생각해보면 당연하다.

student는 그저 배열의 시작 주소를 가지고 있는 변수이다. new int[10]를 만들어 student 변수에 넣는순간 기존에 존재하던 길이5의 배열의 주소를 잃어버리는 것이다. 해당 배열은 이제 자신을 가리키는 변수가 없고 이럴 경우 일정시간이 지나면 가비지컬렉션이 메모리를 삭제한다. 그렇다면 배열의 길이는 사용도중 늘릴수 없는것인가?

방법은 존재하지만 정확히는 기존 배열의 사이즈를 증가 시키는 방법이 아니다. 바로 길이10의 새로운 배열을 만든후 기존의 길이5배열 요소의 값들을 모두 복사 해준뒤 기존 길이5배열을 가리키는 참조변수의 주소값을 길이10의 배열 주소값으로 교체 하는 것이다. 그림28은 해당 방법을 작성한 소스코드이다.

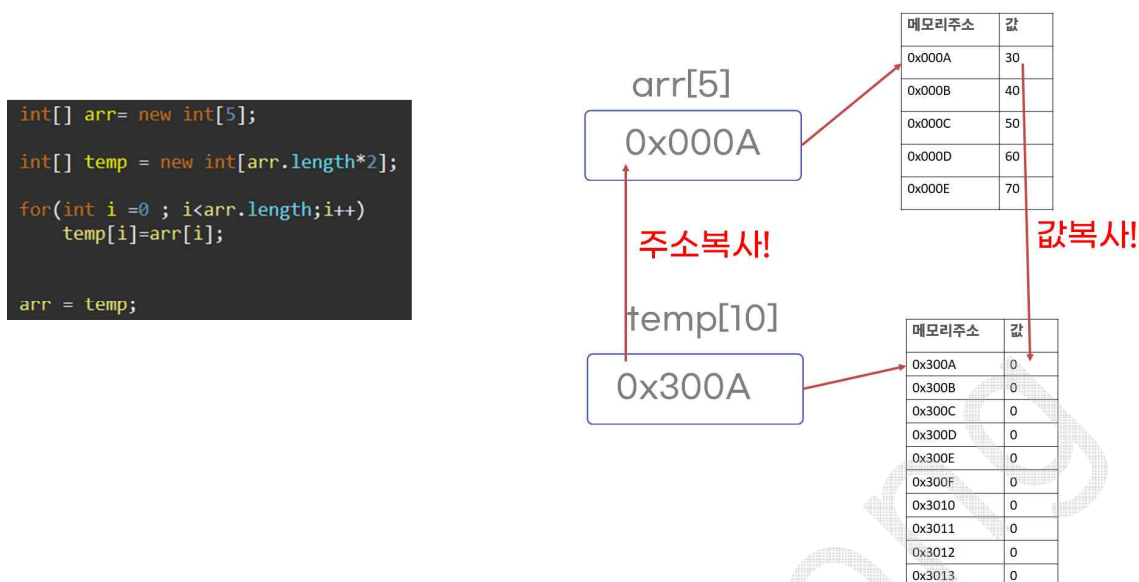


그림 28

### 카. 향상된for(foreach)

배열의 요소들을 모두 출력 하거나 혹은 초기화 할 때 반복문을 이용하여 첫 번째 요소부터 마지막 요소까지 순서대로 접근하는 것이 일반적이다. 하지만 배열을 사용 하다보면 이런 경우가 굉장히 많이 발생한다. 코드를 작성하다보면 이런 반복적인 코드를 계속 작성하기 번거롭다. 그래서 JAVA는 기존의 for문을 개선하여 배열을 쉽게 사용할수 있는 기능을 제공해준다.

```

int[] arr = new int[10];

for(int i = 0 ;i<arr.length; i++)
    arr[i] = (int)(Math.random()*10);
    
```

기존의 For문을 이용한 배열 순차 접근



```

for(int i : arr)
    System.out.println(i);
    
```

배열을 순차접근하는 목적이라면 향상된 for문을 사용하자!

그림 29

그림29를 보면 기존의 반복문과 다르게 아래쪽은 코드가 상당히 간결한 것을 볼수 있다. 이것이 바로 향상된 for이다. arr배열의 첫 번째 요소부터 순서대로 가져와 i변수에 할당하여 배열의 길이만큼 반복해준다. 그러니 왼쪽의 변수타입은 배열의 요소타입과 일치 해야 한다.

### 3. 다차원배열

지금까지의 배열은 배열의 요소들이 단일데이터 하나였다. 하지만 우리가 데이터를 다루다 보면 아래와 같이 표 형태의 데이터를 다뤄야 할 때도 많다.

아래의 표는 학생들의 성적리스트이다.

김길동	30	20	80
홍길동	50	20	89
고길동	80	80	100

이러한 데이터들은 변수로써 어떻게 표현 해야할까? 바로 이런 문제들을 다차원배열을 통해 해결한다.

지금까지 배열의 요소가 단일데이터인 것을 보고 1차원 배열 이라고 부른다. 그리고 2차원 이상의 배열을 다차원 배열 이라고 부른다.

```
int[][] arr = new int[5][3];
```

그림 30

우선 2차원 배열이 예시를 보자. 그림30을 보면 기존의 1차원 배열과는 다르게 대괄호를 두 개 연속으로 적어 주었다. 그리고 배열을 생성할 때 길이를 보면 대괄호 두군데 모두 길이를 작성해 주었다.

이것의 의미는 길이3의 배열을 요소로 가지는 길이5의 배열을 만든다는 것이다.

쉽게 말해 arr의 배열의 길이는 5이다. 그리고 그 5개의 요소들이 각각 길이3의 배열인 것이다.

**3열**

**5행**

arr[0]	int	int	int
arr[1]	int	int	int
arr[2]	int	int	int
arr[3]	int	int	int
arr[4]	int	int	int

arr[0]을 통해 0번째 요소를 가져와보면 해당 요소가 바로 길이3의 배열인 것이다.

그리고 길이3의 배열의 요소까지 접근하여 실질적인 int 데이터를 가져오는 방법은 arr[1][2]라고 쓰면 된다. 쉽게 생각하면 좌표와 같다고 보면된다. 왼쪽 가장위쪽의 데이터는 [0][0]이며 오른쪽 가장 아래쪽 데이터는 [4][2]로 접근하면 된다.

### 길이3의 배열을 저장하는 길이5배열

그림 31

int	int	int
int	int	int
int	int	int
int	int	int
int	int	int

arr[2][1]

그림 32

### 가. 2차원 배열의 초기화

2차원 배열 역시 배열 생성과 동시에 값을 초기화 할 수 있다.

```
int[][] arr = new int [][]{{0,1,2},{0,1,2}};
```

그림 33

1차원 배열과 마찬가지로 대괄호내부의 길이는 비워 두어야 하며 중괄호를 중첩으로 하여 2차원 배열을 표현했다. 2차원배열의 첫 번째 요소는 {0,1,2} 이며 길이3의 배열인 것이다.

```
int[][] arr = {{0,1,2},{0,1,2}};
```

2차원 배열 역시 new 키워드와 데이터타입 부분은 생략해도 된다.

실질적으로 위와 같이 한 줄로 적기보다는 가독성을 위해 그림35처럼 작성하는 것이 일반적이다.

```
int[][] arr = {  
    {0,1,2}  
    ,{0,1,2}  
};
```

그림 35

그림36은 2차원 배열을 생성하고 값을 초기화하는 예제이다.

### 2차원 배열 예시

```
int[][] arr = new int[5][3];
```

```
arr[0][0] = 10;  
arr[0][1] = 20;  
arr[0][2] = 30;
```

```
arr[1][0] = 40;  
arr[1][1] = 50;  
arr[1][2] = 60;
```

10	20	30
40	50	60
0	0	0
0	0	0
0	0	0

그림 36

## 나. 2차원 배열의 length

length는 배열에서 제공해주는 길이 값을 가지고 있는 변수이다. 배열변수이름.length로 쓰는 것을 배웠었다. 2차원 배열에서 length를 사용하는 것을 보도록 하자.

```
int[][] arr = new int [5][3];  
System.out.println(arr.length);
```



```
<terminated>  
5
```

arr.length의 값은 5가 나오는 것을 확인 할 수 있다. 2차원 배열의 의미를 생각해보면 당연하다고 볼수 있다. arr은 길이5의 배열이며 각각의 요소가 길이3의 배열이 들어가 있을뿐이다. 그러므로 arr.length는 5를 가지고 있는 것이다.

```
int[][] arr = new int [5][3];  
System.out.println(arr[3].length);
```



```
<terminate  
3
```

이번에는 arr[3].length를 확인해보면 3이 나오는 것을 볼수 있다. 바로 arr배열의 요소가 길이3의 배열이기 때문이다. 그렇다면 arr[0].length는 얼마일까?

그렇다 2차원의 인덱스가 0으로 바뀐다하여도 그대로 3일 것이다. 각각의 요소들이 모두 길이3의 배열이기 때문이다.

## 다. 반복문을 이용한 2차원 배열의 접근

### 1) 중첩반복문

```
int[][] arr = new int [5][3];  
  
for(int i =0; i<arr.length;i++)  
    for(int j =0; j<arr[0].length;j++)  
        arr[i][j] = 50;
```

그림 39  
중첩 반복문을 활용하여 2차원 배열을 사용 한다.

2차원 배열역시 대괄호의 index번호를 지정하여 요소들을 접근한다.

그림39를 보면 중첩 반복문을 이용해 2차원 배열의 인덱스를 지정하는 것을 볼수 있다. 물론 반복문을 하나만써서 2차원과 1차원의 인덱스를 적절히 계산해서 반복할수도 있다. 하지만 그렇게 반복문의 개수를 줄인다 하여도 성능상의 이점도 없고 코드의 가독성만 안좋아 지기에 일반적으로

### 2) 향상된for

```
int[][] arr = new int [5][3];  
  
for(int[] temp : arr)  
    for(int i : temp)  
        System.out.println(temp[i]);
```

다음은 2차원 배열을 향상된for문을 이용해 사용하는 방법이다. temp의 데이터타입을 주의깊게 봐야 한다. arr는 2차원 배열이고 각각의 요소가 또다시 1차원 배열이기 때문에 temp는 1차원배열 타입이어야 한다. 그리고 내부의 for에서 temp의 요소는 실질적인 데이터가 있는 int 타입이기에 i의 타입은 int가 된다.

그림 40

## 라. 가변배열

지금까지 2차원 배열을 보면 2차원의 요소가 1차원 배열이며 이 배열들의 길이는 모두 일정한 것을 봤다. 하지만 가변배열은 2차원배열 요소들의 1차원배열 길이가 제각각인 것을 말한다.

```
int[][] score = new int [5][];
```

우선 1차원의 길이는 요소마다 다르게 설정하기 위해 사이즈를 지정하지 않는다.

이때 2차원배열을 생성은 하였지만 각각의 요소들은 모두 null값을 가진다.

그림 41

```
System.out.println(score[0]);
```

null

첫 번째 요소의 값을 출력해보면 null인 것을 확인해 볼 수 있다.

```
int[][] score = new int [5][];  
  
score[0] = new int[4];  
score[1] = new int[3];  
score[2] = new int[2];  
score[3] = new int[1];  
score[4] = new int[6];
```

int	Int	Int	Int		
int	Int	Int			
int	Int				
int					
int	Int	Int	Int	Int	Int

그림 43

그림 44 가변배열을 표현

그 후 각각의 요소에 적절한 길이의 1차원 배열을 생성하여 채워 넣을 수 있다.

즉 다차원 배열일 경우 요소의 배열 길이는 모두 달라도 상관이 없다.