6. 배열

배열이란 동일한 타입의 데이터 집합을 하나의 묶음으로 처리할 수 있도록 지원하는 데이터 구조이다. Java 에서는 배열의 효율적인 활용을 위해서 length 변수를 사용할 수 있으며 2 차원 배열의 경우에는 가변 배열 개념을 지원하고 있다.

6.1 배열의 생성과 사용

배열은 같은 타입의 여러 변수들을 연속적으로 데이터 공간을 할당하여 하나의 묶음으로 다루는 것이다. Java 에서의 배열은 객체로 생성되어 사용되므로 참조 형 변수로 분류되는 배열 변수를 선언하여 사용한다.

■ 배열 변수 선언과 배열 생성

배열을 생성하여 사용하기 위해서는

- 어떠한 타입의 데이터들을 저장하여 처리할 것인가?
- 처리하고자 하는 데이터들의 최대 개수를 어떻게 되는가?

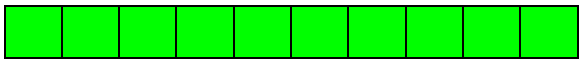
를 결정한 다음에 다음과 같은 구문으로 생성한다.

```
new 배열타입[배열크기]
```

new 라는 연산자 뒤에 배열의 타입을 지정하고 대괄호([]) 안에 배열의 크기를 지정한다. 배열의 크기라는 것은 배열을 구성하는 데이터를 저장할 방의 개수 즉, 저장 가능한 데이터의 개수가 된다. 배열을 구성하는 데이터를 원소(element)라고 한다. 다음은 배열의 생성 예이다.

```
new   int[10]          // int 형 원소 10개로 구성되는 배열을 생성한다.
new   double[50]       // double 형 원소 50 개로 구성되는 배열을 생성한다.
new   char[100]        // char 형 원소 100개로 구성되는 배열을 생성한다.
new   String[10]       // String 형 원소 10 개로 구성되는 배열을 생성한다.
```

new int[10] 은 다음과 같이 메모리상에 int 타입(4바이트)의 원소영역을 10개 생성하는 결과가 된다. 배열의 각 원소들의 메모리 영역은 연속적이다.



이렇게 생성되는 배열은 배열 변수를 통해서 사용하게 되며 배열 변수는 사용하려는 배열의 타입이 무엇인지에 따라서 배열 변수 선언 시에도 동일한 타입을 지정하며 배열을 참조하는 변수라는 것을 나타내기 위기 위해 다음과 같이 빈 대괄호([])를 붙여서 선언한다.

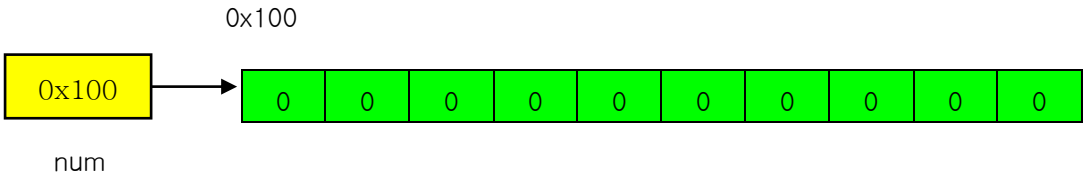
```
배열타입   배열변수명[ ];
배열타입[ ] 배열변수명;
```

int 형 배열을 사용하려면 int 형 배열 변수를 String 형 배열을 사용하려면 String 형 배열 변수를 다음과 같이 선언한다.

```
int[ ] numArray;          // int 형 배열 변수를 선언한다.
String  strArray[ ]       // double 형 배열 변수를 선언한다.
```

배열은 위와 같이 선언된 배열 변수에 초기화하여 배열 변수를 통해서 사용한다. 배열 변수를 초기화 한다는 것은 다음과 같이 배열의 참조 값 즉, 주소 값을 대입 받아 참조하게 된다는 것을 뜻한다.

```
int[] num = new int[10];
```



■ 배열의 초기화

배열은 힙이라는 메모리 영역에 생성되며 어떠한 타입의 배열이냐에 따라서 각 타입 별 기본값으로 자동 초기화가 된다. int 배열은 모든 원소들의 값이 0 으로 초기화 되고 double 형 배열은 0.0 그리고 참조형 배열은 null 이라는 값으로 초기화 된다. 다음 표는 각 타입 별 기본값을 소

개하는 표이다.

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

각 원소의 값들을 원하는 값들로 초기화 되는 배열(초기화 배열)을 생성하기 위해서는 다음과 같이 블록({ })을 사용한다.

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}      // 1 부터 10 까지의 값으로 초기화되는 배열을 생성.
new char[]{'A', 'B', 'C'}            // 'A', 'B', 'C' 값으로 초기화 되는 배열 생성.
```

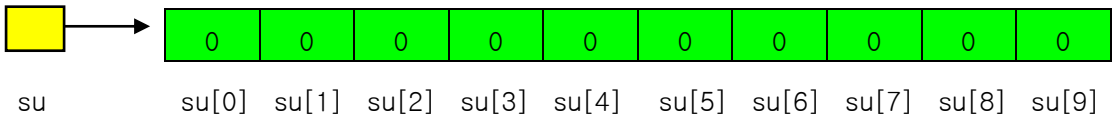
배열의 타입과 블록을 모두 지정하는 방법은 배열 변수를 선언하면서 초기화하는 경우와 이미 선언되어 있는 배열 변수에 초기화하는 경우 모두에서 사용될 수 있지만 블록만 사용하는 경우에는 배열 변수를 선언하면서 초기화하는 경우에만 사용 가능하다.

```
int array1[ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int array2[ ];
array2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};      // 허용되지 않음
char array3[ ] = new char[ ] {'A', 'B', 'C'};
char array4[ ];
array4 = new char[ ]{'A', 'B', 'C'};
String array5[ ] = { "I", " love",   "Java"};
String array6[ ];
array6 = new String[ ]{ "I", " love",   "Java"};
```

■ 배열의 사용

배열 변수는 배열을 다루는데 사용되는 변수로서 배열 변수를 통해서 원하는 위치의 원소를 사용하려는 경우 배열 변수 뒤에 [index] 를 붙여서 몇 번째 원소를 사용하려는 것인지를 지정한다. index 는 0 부터 지정하므로 마지막 원소의 index 는 배열의 크기에 1 을 뺀 값이 된다.

```
int[ ] su = new int[10];
```



su su[0] su[1] su[2] su[3] su[4] su[5] su[6] su[7] su[8] su[9]

```
int su[ ] = new   int[10];              // 각 원소의 값들은 0 으로 초기화 된다.
for (int i=0; i < 10; i++)              // 반복문으로 su 가 참조하는 배열의
    System.out.print(su[i] + "₩t");     // 모든 원소값을 출력한다.
su[0] = 100;                            // 첫 번째 원소의 값을 0 으로 대입한다.
su[1] = 200;                            // 두 번째 원소의 값을 0 으로 대입한다.
su[2] = 300;                            // 세 번째 원소의 값을 0 으로 대입한다.
System.out.print(su[0]+su[1]);          // 첫 번째와 두 번째 원소의 값을 덧셈하
여
                                        // 콘솔에 출력한다.
```

▶ length 변수의 활용

Java 에서의 배열은 객체로 생성되고 사용된다. 메모리에 배열이 만들어질 때 배열 객체 안에는 length 변수가 자동으로 선언되고 초기화 되는데 lengtn 변수에 초기화되는 값은 배열의 크기 즉, 배열 원소의 개수가 된다.

배열의 length 변수를 사용하면 배열의 원소개수만큼 반복하여 수행하려는 경우 유용하게 사용된다.

```
printArray(new int[ ] {10, 20, 30, 40, 50});                    // 5개 원소의 int 형 배열
전달
printArray(new int[10]);                                        // 10개 원소의 int 형 배열 전달
printArray({1, 2, 3, 4, 5});          // 이미 선언되어 있는 배열 변수에 대입하는 것이
므로
                                // 허용되지 않는다. 이렇게 블록만을 사용하는 초기화 배열
                                    // 사용하는 것은 배열 변수를 선언하면서 초기화할
경우이다.
printArray(new int[ ] {100, 200, 300});                 // 3개 원소의 배열을 생성하여 배열
                                                                의 크기에는 제한이 없다.

public static void printArray(int  array[ ]) {
    System.out.println("전달된 배열의 크기 : "+array.length);
    for(int i=0; i < array.length; i++)
```

▶ foreach 문의 사용

Java SE 5.0 부터 추가된 구문으로서 향상된 for문(Enhanced for)이라고도 한다. 배열이나 컬렉션 객체와 같은 데이터들의 집합에서 구성 요소들을 하나하나 읽어서 반복 처리하고자 하는 경우 유용하게 사용되는 구문이다.

```
for(변수 선언 : 배열 또는 컬렉션 객체)
{
    :
```

배열이나 컬렉션 객체에서 추출되는 데이터를 담을 변수

```
int num[] = {1, 2, 3, 4, 5};
int sum = 0;
for(int element : num) {
        sum += element;
}
```

배열을 구성하는 원소들을 첫 번째 원소부터 추출하여 선언되어 있는 변수에 담아서 반복 문장을 수행하도록 하며 더 이상 추출할 원소가 없으면 반복문 수행을 종료한다. 변수는 추출하는 원소의 타입과 동일한 타입을 설정하며 이 변수를 통해서 주어진 배열이나 콜렉션 객체의 원소들을 읽기만 가능하고 수정할 수는 없다.

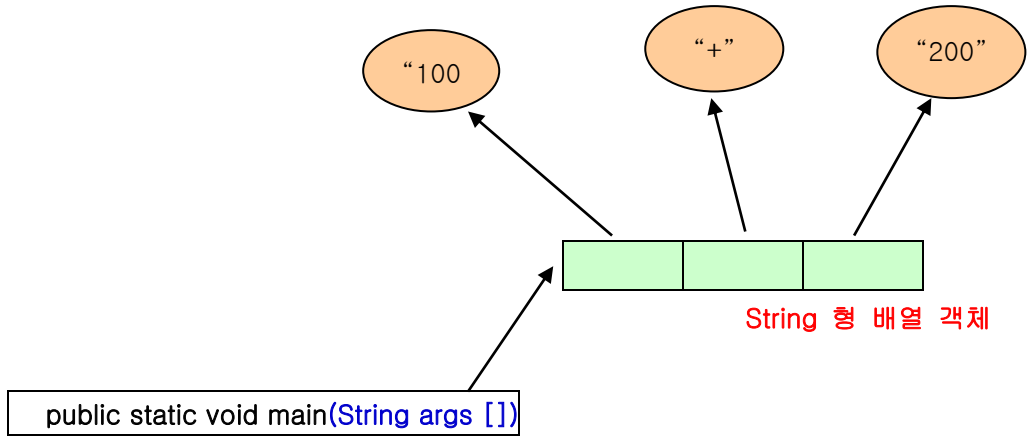
■ 명령 행 매개변수

명령 행 매개변수란 Java 프로그램을 실행시키는 명령 행에 작성된 데이터(명령 행 데이터)를 전달받는 받는 변수로서 main() 메서드의 매개변수로 선언된다. 다양한 타입의 여러 개의 데이터들을 전달할 수 있도록 하기 위해 Java 인터프리터에 의해서 String 형 배열로 구성하여 전달되므로 이 변수는 String 형 배열 변수로 선언되어야 한다.

```
java   TestApp1   100
java   TestApp2   a   b   c
java   TestApp3   100 + 200
java   TestApp4   Java “OOP Language”
```

수행시키려는 클래스 명 뒤에 전달하고자 하는 명령 행 데이터들을 나열하게 되며 이 데이터들은 다음 그림에서와 같이 공백을 기준으로 각각을 문자열(String)로 만들고 배열로 구성하여 main() 메서드 호출 시 전달된다.

java TestApp3 100 + 200



다음은 명령 행 데이터를 전달 받아서 화면에 행 단위로 출력하는 예제이다.

```
// 전달된 명령 행 데이터들을 화면에 행 단위로 출력하는 소스
public static void main(String args[]) {
    System.out.println("전달된 명령 행 데이터의 개수 : "+ args.length);
    for (String s : args)
        System.out.println( s);
    for (int i=0; i < args.length; i++)
        System.out.println( args[i]);
}
// 전달된 두 개의 명령 행 데이터들을 덧셈하여 출력하는 소스
// 명령 행 데이터는 전달되는 데이터의 종류에 관계없이 무조건 String 형으로 // 전달되므로 전달되는 데이터가 정수형 숫자인 경우 API(Integer.parseInt())
// 를 사용하여 int 형으로 변환한다.
public static void main(String args[]) {
    int num1 = Integer.parseInt(args[0]);
    int num2 = Integer.parseInt(args[1]);
    System.out.println("전달된 명령 행 데이터의 합 : "+(num1+num2));
}
```

6.2 다차원 배열

앞에서 학습한 배열은 1차원 배열이라고 할 수 있으며 배열을 생성할 때 그리고 배열 변수를 선언할 때 대활호([ ])의 개수를 몇 개 지정하는가에 따라서 배열의 차원이 정해진다.

Java 에서도 다차원 배열을 지원하며 일반적으로 2차원 배열을 많이 사용한다. 2차원 배열이란 논리적으로 행과 열이라는 테이블 구조로 데이터들을 구성하려 다루는 것을 의미한다. 테이블 구조로 데이터를 다루면 행 과 열이라는 개념을 적용하여 원하는 목적의 데이터들을 좀 더 효율적으로 처리할 수 있다.

예를 들어서 최근 10년 동안의 월별 데이터를 저장하여 처리하려는 경우 120 개의 크기를 갖는 1차원 배열을 생성하여 처리해도 되지만 10행 12열의 테이블 구조를 갖는 2차원 배열을 생성하여 저장하고 처리한다면 행은 년도, 열은 월이라는 구성으로 데이터들을 년도별로 또는 월별로 데이터들을 처할 때 훨씬 효율적으로 구현할 수 있다.

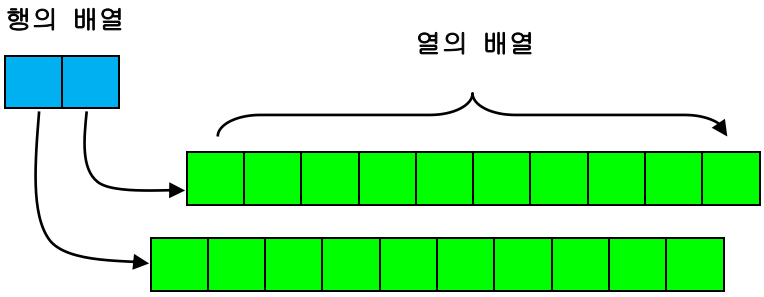
■ 이차원 배열의 생성

2차원 배열을 생성하는 방법은 1차원 배의 경우와 거의 같으며 대괄호([ ])를 두 개 사용한다.

```
new 배열타입[행의 크기][열의 크기]      // 행과 열의 크기 모두 지정
new 배열타입[행의 크기][ ]              // 행의 크기만 지정
```

Java 의 2차원 배열은 열의 크기만큼의 1차원 배열들을 행의 크기만큼 생성하고 생성된 각 1차원 배열들을 원소로 참조하는 1차원 배열로 구성된다. 즉, 각 원소가 1차원 배열인 배열이라고 할 수 있다.

new int[2][10] 은 메모리에 다음과 같은 구조의 2차원 배열 객체가 생성된다.



new int[2][ ] 은 메모리에 다음과 같은 구조의 2차원 배열 객체가 생성된다. 이렇게 행의 크기는 지정했으나 열의 크기를 지정하지 못한 경우에는 사용하기 전에 각각의 행마다 열에 해당되는 1차원 배열을 대입하여 사용한다. 이렇게 2차원 배열을 생성할 때 행의 크기만 지정한 후에 행마다 열에 해당하는 1차원 배열을 대입하는 방식의 배열을 **가변배열**이라고 한다.

■ 이차원 배열의 사용

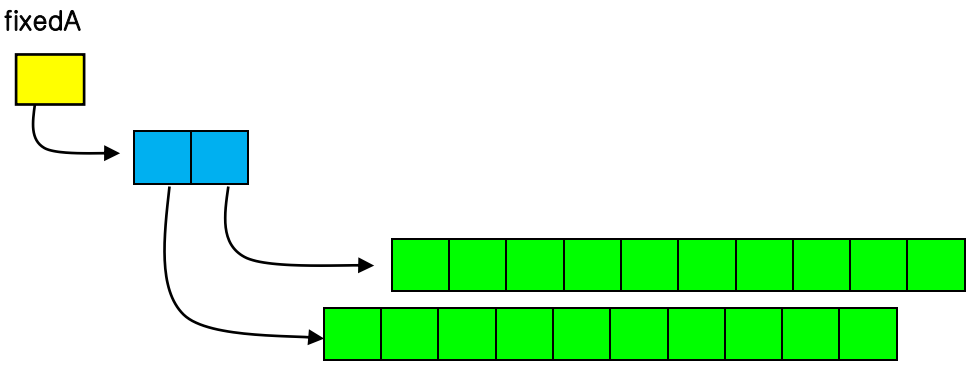
2차원 배열은 2차원 배열 변수에 담아서 사용한다. 2차원 배열 변수를 선언할 때는 2차원 배열을 생성할 때와 같이 대괄호([ ])를 2 개 사용한다.

```
int   twoArray[ ][ ];
int [ ] twoArray [ ];
int [ ][ ] twoArray;
twoArray = new int[2][10];
char[ ][ ] matrix = new char[10][10];
double[ ] tax[ ] = new double[ 12][10 ];
// 고정 2차원 배열
int fixedA[ ][ ] = new int[2][ 10];        // 모든 행의 열의 크기 동일
// 가변 2차원 배열
int changableA[ ][ ] = new int[2][ ];
changableA [0] = new int[5];
```

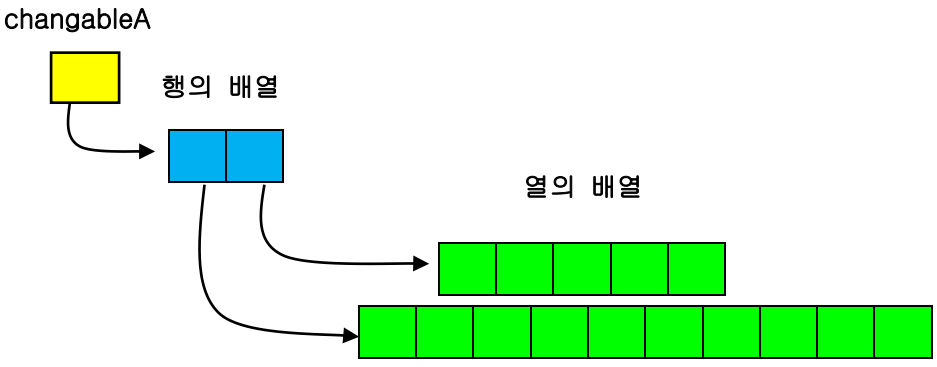
대괄호의 위치는 변수의 앞과 뒤 어디든 가능하다.

행마다 열의 크기를 가변적으로 설정할 수 있다

finxedA 변수가 참조하는 배열은 다음과 같은 구조로 2차원 배열이 만들어진다.



changableA 변수가 참조하는 배열은 다음과 같은 구조로 2차원 배열이 만들어진다.



2차원 배열의 경우에도 객체로 생성되며 length 라는 변수가 자동 초기화 된다.

chageableA.length = changableA 가 참조하는 배열의 크기(행의 크기) ---→ 2
chageableA[0].length = 첫 번째 행이 참조하는 배열의 크기(열의 크기) ---→ 5
chageableA[1].length = 두 번째 행이 참조하는 배열의 크기(열의 크기) ---→ 10

2차원 배열 변수를 사용하여 2차원 배열을 사용할 때에도 대괄호에 0 부터 시작하는 인덱스를 사용한다.

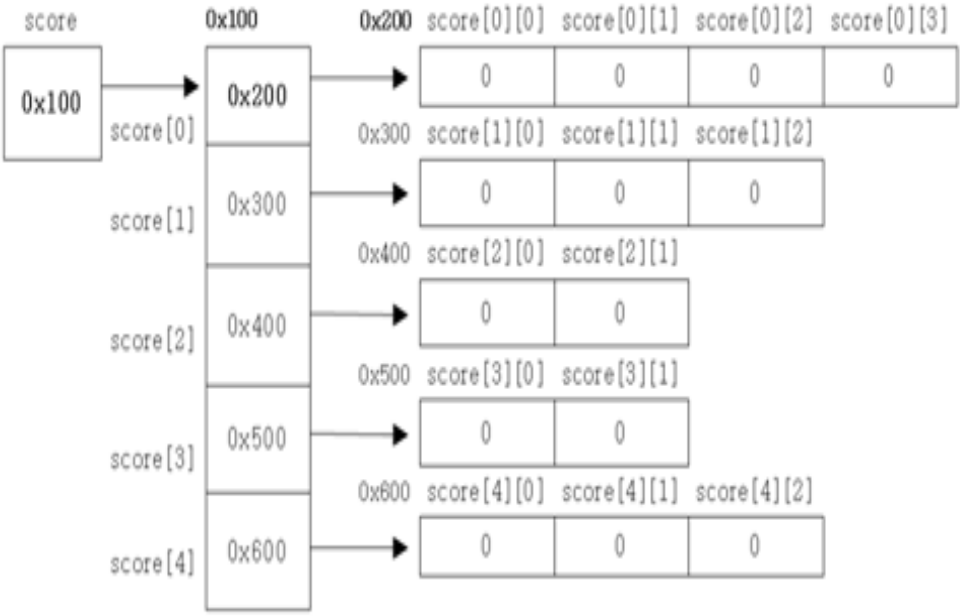
[0][0] 은 첫 번째 행의 첫 번째 열에 해당되는 원소
[0][1] 은 첫 번째 행의 두 번째 열에 해당되는 원소
[1][0] 은 두 번째 행의 첫 번째 열에 해당되는 원소
:

```
int   twoArray[ ][ ] = new int[3][5];        // 각 원소의 값이 0으로 초기화되는
                                             // 3행 5열 2차원 배열을 생성한다.
for (int i=0; i < twoArray.length; i++) {    // i 는 행의 크기만큼 변환된다.
    for (int j=0; j < twoArray[i].length; j++) // j 는 열의 크기만큼 변환된다.
        System.out.print(twoArray[i][j] + " ");
    System.out.println();
}
```

[가변 배열의 생성과 구조]

다음은 5행이면서 가 행의 열의 크기는 가변적인 2차원 배열의 예와 배열이 생성되어 사용되는 메모리 구조에 대한 그림이다.

```
int score [ ][ ] = new int[5][ ];          // 2차원 배열의 행의 크기만 지정하여
생성한다.
score[0] = new int[4];          // 첫 번째 행은 열의 크기가 4 이다.
score[1] = new int[3];          // 두 번째 행은 열의 크기가 3 이다.
score[2] = new int[2];          // 세 번째 행은 열의 크기가 2 이다.
score[3] = new int[2];          // 네 번째 행은 열의 크기가 2 이다.
score[4] = new int[3];          // 다섯 번째 행은 열의 크기가 3 이다.
```



2차원 배열의 경우에도 각 원소의 값은 기본값으로 자동 초기화 되며 원하는 값들로 초기화 되는 2차원 배열을 생성하려면 중첩된 블록 구조를 사용한다.

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