04. 자바 배열

강사 강태광

1] 배열 개요 1

1) 배열

- 같은 형의 데이터들을 다수 개 저장할 수 있는 기억장소를 의미
- 하나의 이름으로 다수 개의 데이터를 사용할 수 있음
- 학생 10명 성적의 평균을 구하는 프로그램을 작성시 많은 변수를 사용하여 프 로그램을 복잡하게 하여 오류를 발생시킬 수 있다 .

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

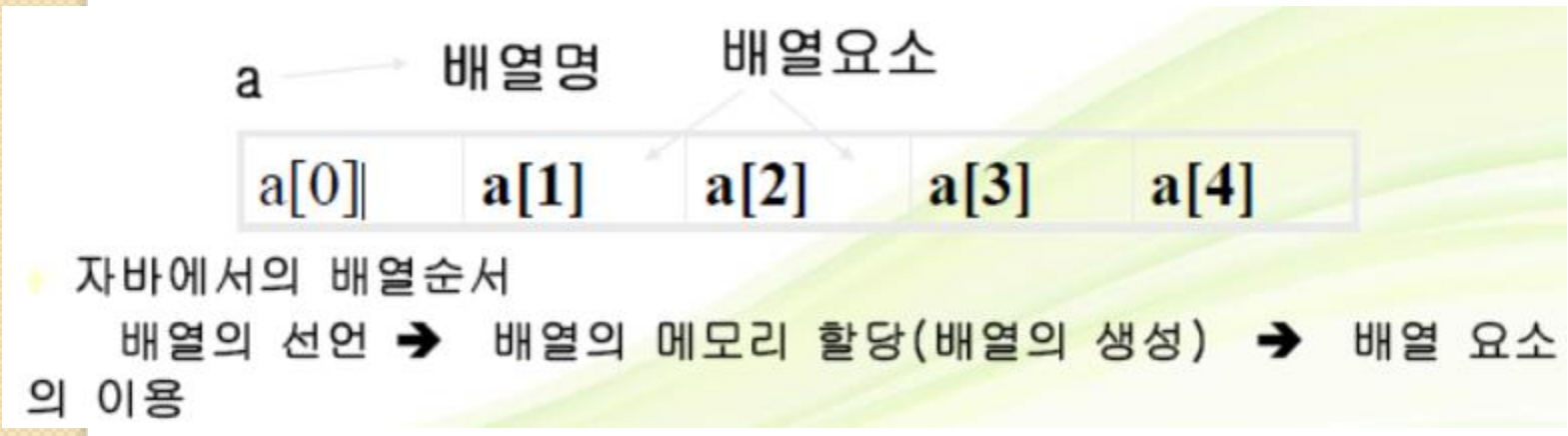
```
int[] score;          // 배열을 선언한다.(생성된 배열을 다루는데 사용될 참조변수 선언)
score = new int[5]; // 배열을 생성한다.(5개의 int값을 저장할 수 있는 공간생성)
```

[참고] 위의 두 문장은int[] score = new int[5];와 같이 한 문장으로 줄여 쓸 수 있다.

1] 배열 개요

2) 같은 종류의 데이터를 저장하기 위한 자료구조

- 배열을 객체로 취급
- 배열을 선언할때 배열의 크기를 명시하지 않는다.
- [ ]는 앞에 오나 뒤에 오나 상관이 없다.
- 어떤 형으로 사용할 것인가를 정하고 배열변수를 정해준다.



자바에서의 배열순서

배열의 선언 ➔ 배열의 메모리 할당(배열의 생성) ➔ 배열 요소의 이용

I] 배열 개요

3) 학생 5명의 성적을 저장해야 한다면

예	int s1, s2, s3, s4, s5;

만약 전교생(10000명)의 저장해야 한다면 ???

① 배열
- 자료형이 동일한 여러 개의 값을 연이어 저장할 수 있도록 하는 기억 공간의 집합체(모임)

② 원소(Element)
- 배열에 저장된 각각의 값

③ 색인, 인덱스 : index
- 배열의 원소에 접근하기 위한 첨자
- 만일 배열의 이름이 a라면 배열 원소는 a[0], a[1], a[2], ... 로 표시

2] 배열 원소와 색인/인덱스

[1] 5개의 정수형을 저장하는 배열 선언

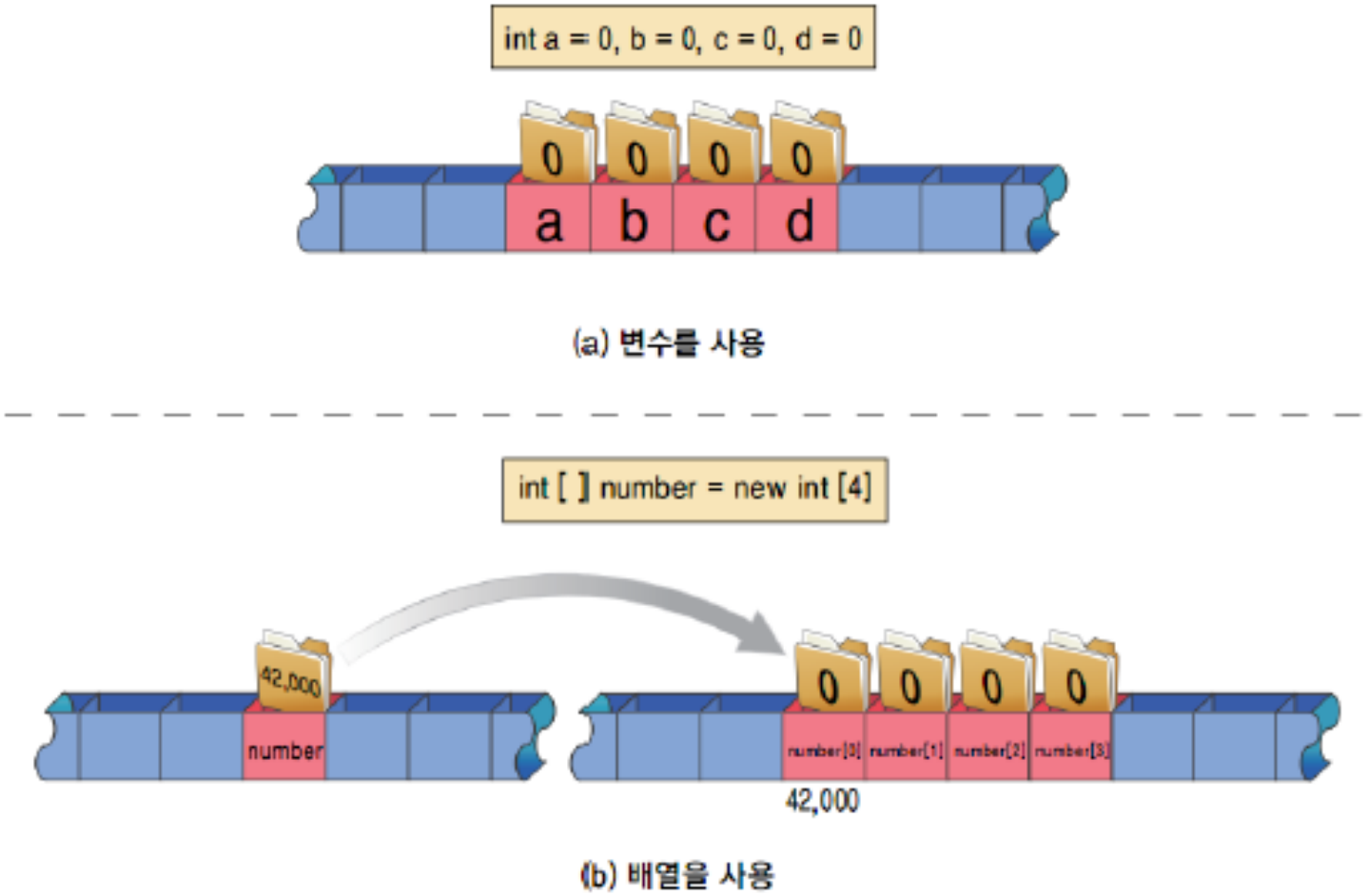
예

```
int [ ]  a = new int [ 5 ];    int a[ ] = new int[5];
①   ②              ③
```

① 각 원소에 저장할 값에 대한 자료형
② 배열명
③ 배열의 원소의 개수

2] 배열 원소와 색인/인덱스

- 변수를 사용하는 경우와 배열을 사용하는 경우

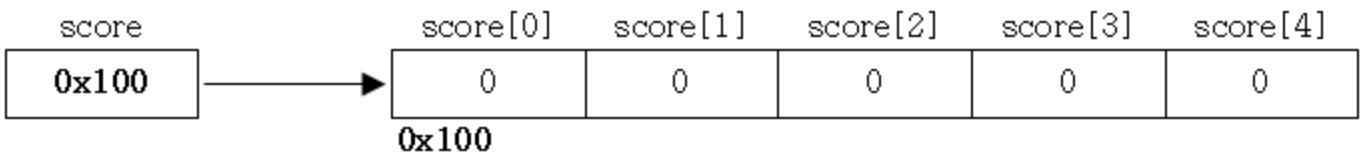


3] 배열의 선언, 생성, 이용

- 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 int값을 저장할 수 있는 공간생성)
```

[참고] 위의 두 문장은int[] score = new int[5];와 같이 한 문장으로 줄여 쓸 수 있다.



3] 배열의 선언, 생성, 이용

▶ 배열요소의 이용

배열요소는 배열변수명과 인덱스 두 가지를 이용한다.

※인덱스란? 배열의 맨 처음부터의 위치이다.

주의할 점은 자바에서의 인덱스는 **0부터 시작**한다는 것이다.

배열 변수명[인덱스]

Name[0]=3;

4] 1차원 배열

- 초기 값을 콤마로 구분하여 여러 번 기술하고 이들을 중괄호 { }로 감싸 준다.

예

```
int [ ]a ={10, 20, 30, 40, 50};
```

- 배열의 원소 값들을 출력하기

예

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

a.length
배열의 개수를 알아 낼 수 있다.

4] 1차원 배열

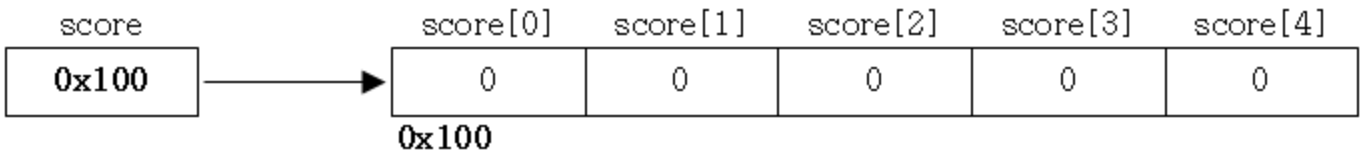
```
// 배열 선언 후 값 대입과 원소 출력하기

public class Ex01 {
    public static void main(String[] args) {
        int []a=new int [5];
        int i;
        a[0]=50;
        a[1]=100;
        a[2]=150;
        a[3]=200;
        a[4]=250;
        for(i=0; i<5; i++)
            System.out.println( (i+1)+" th a[" + i + "]=" +a[i]);
    }
}
```

4] 1차원 배열

- 생성된 배열에 처음으로 값을 저장하는 것(배열 초기화)

```
int[] score = new int[5]; // 크기가 5인 int형 배열을 생성한다.
score[0] = 100;           // 각 요소에 직접 값을 저장한다.
score[1] = 90;
score[2] = 80;
score[3] = 70;
score[4] = 60;
```




```
int[] score = { 100, 90, 80, 70, 60};           // 1번
int[] score = new int[]{ 100, 90, 80, 70, 60}; // 2번
```

```
int[] score;
score = { 100, 90, 80, 70, 60}; // 에러 발생!!!

int[] score;
score = new int[]{ 100, 90, 80, 70, 60}; // OK
```

4] 1차원 배열

배열에 값을 저장하고 읽어오기(배열의 활용)

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

'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]);
}
```

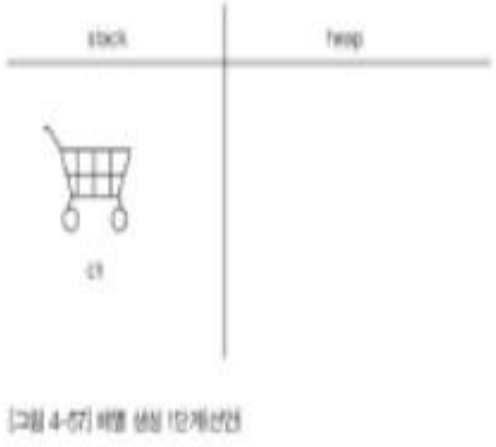
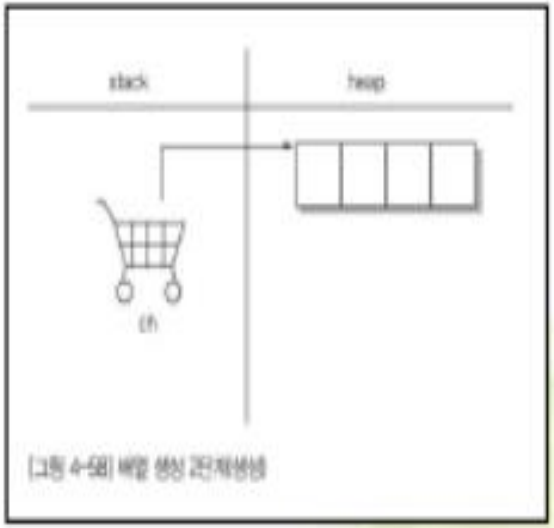
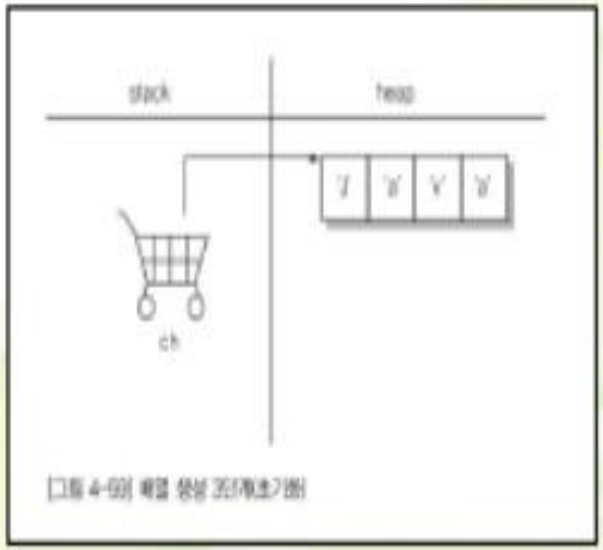
4] 1차원 배열

- 배열 예제(1차원배열)

```
class ArrayTest {
	public static void main(String[] arg){
		int array[];
		array = new int[10];
		for(int i=0; i<10; i++)   array[i]=i+1;
		for(int i=0; i<10; i++)
			System.out.println("["+i+"]"+array[i]);
	}
}
```

4] 1차원 배열

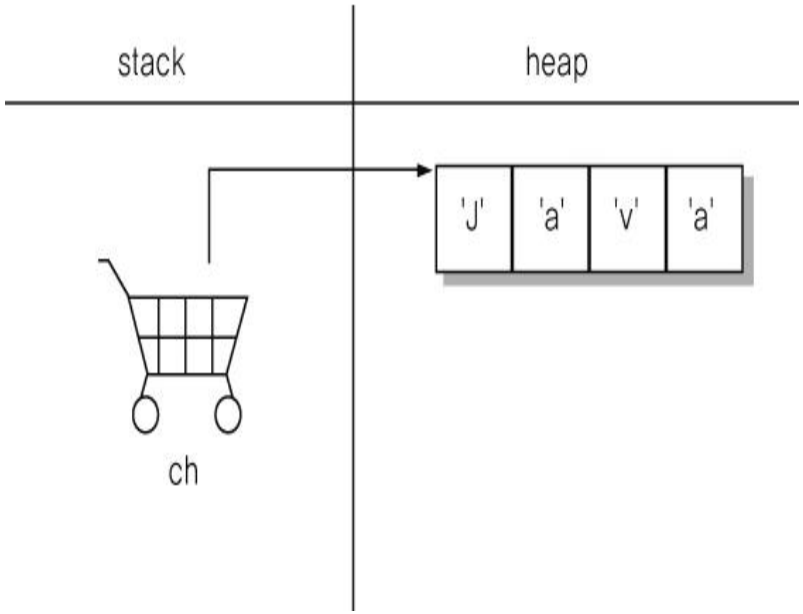
- 배열의 단계적 메모리 할당

1) 배열 선언	2) 배열 생성	3) 배열 초기화
char[] ch; 또는 char ch[];	ch = new char[4];	ch[0]='J'; ch[1]='a'; ch[2]='v'; ch[3]='a';
		
[그림 4-57] 배열 생성 1단계(선언)	[그림 4-58] 배열 생성 2단계(생성)	[그림 4-59] 배열 생성 3단계(초기화)

4] 1차원 배열

- 배열의 메모리

```
class ArrayEx1{
	public static void main(String[] args){
		char[] ch;              // 배열 선언
		ch = new char[4]; // 배열 생성
		//배열 초기화
		ch[0] = 'J';
		ch[1] = 'a';
		ch[2] = 'v';
		ch[3] = 'a';
		//배열 내용 출력
		for(int i = 0 ; i < ch.length ; i++)
			System.out.println("ch["+i+"]:"+ch[i]);
	}
}
```



4] 1차원 배열(배열의 메모리)

- 객체형 배열

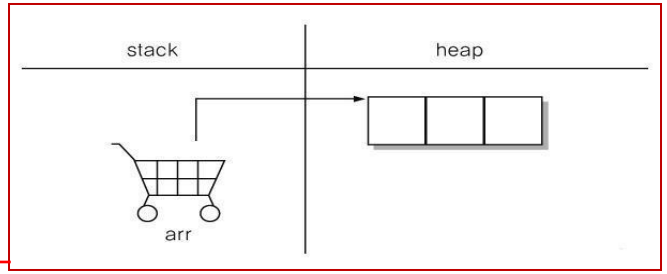
. 객체형 배열은 객체를 가리킬 수 있는 참조 값(주소)들의 묶음 실제 값이 저장된 기본 자료 형과는 다르게 객체의 reference들의 집합

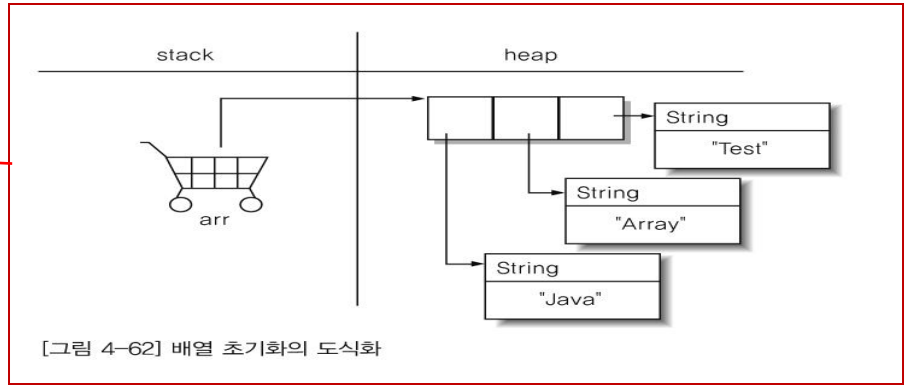
. 객체형 배열은 집집마다 우편함을 한곳에 모아둔 것과 같다.
각 우편함들은 나름 대로 특정한 가정이라는 객체(Object)의 주소를 대신하는 것을 의미하며 이들의 묶음(집합)이 곧 reference 배열 또는 객체형 배열이라 한다

```
class ObjArrayEx1 {
    public static void main(String[] args){

        String[] arr;
        arr = new String[3];

        arr[0] = "Java ";
        arr[1] = "Array ";
        arr[2] = "Test";
    }
}
```





[그림 4-62] 배열 초기화의 도식화

4] 1차원 배열 예제

```
class ArrayTest2 //직접할당 {
    public static void main(String[] arg){
        int array[]={1,2,3,4,5,6,7,8,9,10};
        for(int i=0; i<array.length; i++)
        System.out.println("array["+i+"] :"+array[i]);
    }
}
```

문제1) 76,45,34,89,100,50,90,92 8개의 값을 1차원 배열로 초기화 하고 값에 합계와 평균 그리고 최대값과 최소값을 구하는 프로그램을 작성 하시요. (ArrayTest3.java)

결과값 : 합계 = 576 평균 = 72 최대값은 = 100 최소값은 = 34

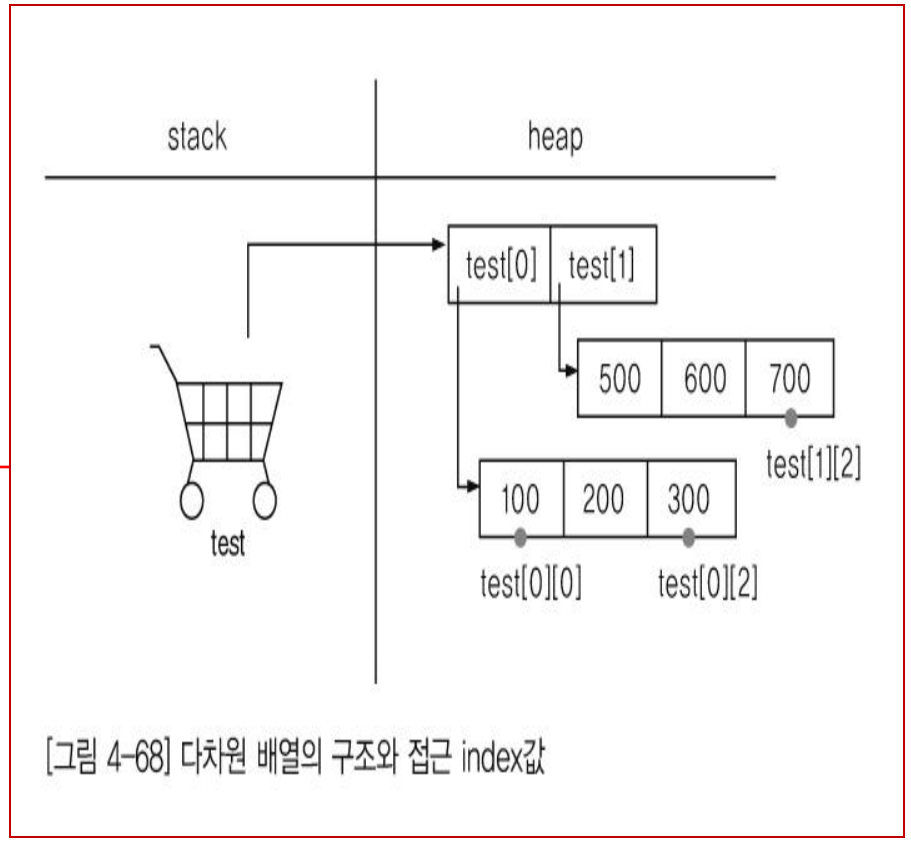
문제2) 76,45,34,89,100,50,90,92 8개의 값을 1차원 배열로 초기화 하고 이들 값들을 크기 순으로 나타내는 프로그램을 작성 하시요. (ArrayTest4.java)

결과값 : 34 45 50 76 89 90 92 100

5] 다차원 배열

1차원 배열이 여러 개 모인 것을 다차원 배열이라 한다.

```
class ObjArrayEx4 {
  public static void main(String[] args){
        int[][] test; // 다차원 배열 선언
        test = new int[2][3];
        test[0][0] = 100;
        test[0][1] = 200;
        test[0][2] = 300;
 //--------------- 1행 초기화 끝
        test[1][0] = 500;
        test[1][1] = 600;
        test[1][2] = 700;
//--------------- 2행 초기화 끝
        }
}
```

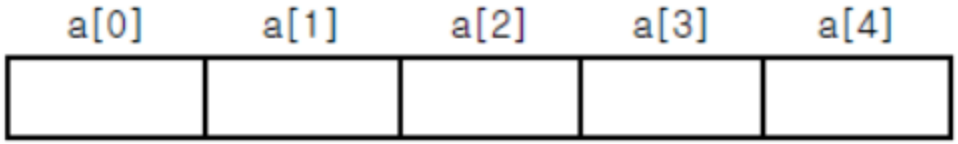


[그림 4-68] 다차원 배열의 구조와 접근 index값

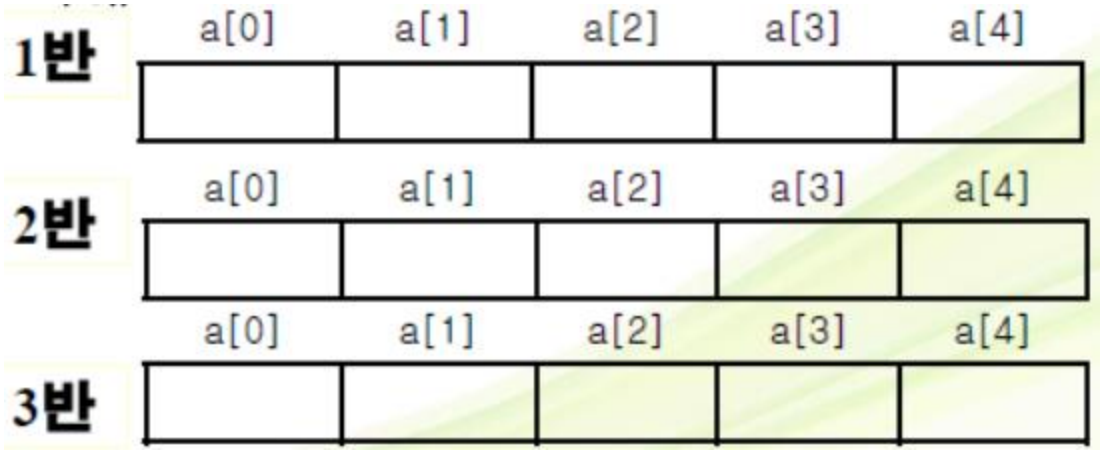
5] 다차원 배열 (2차원 배열)

- 1차원 배열

예	int [ ] a = new int [5];

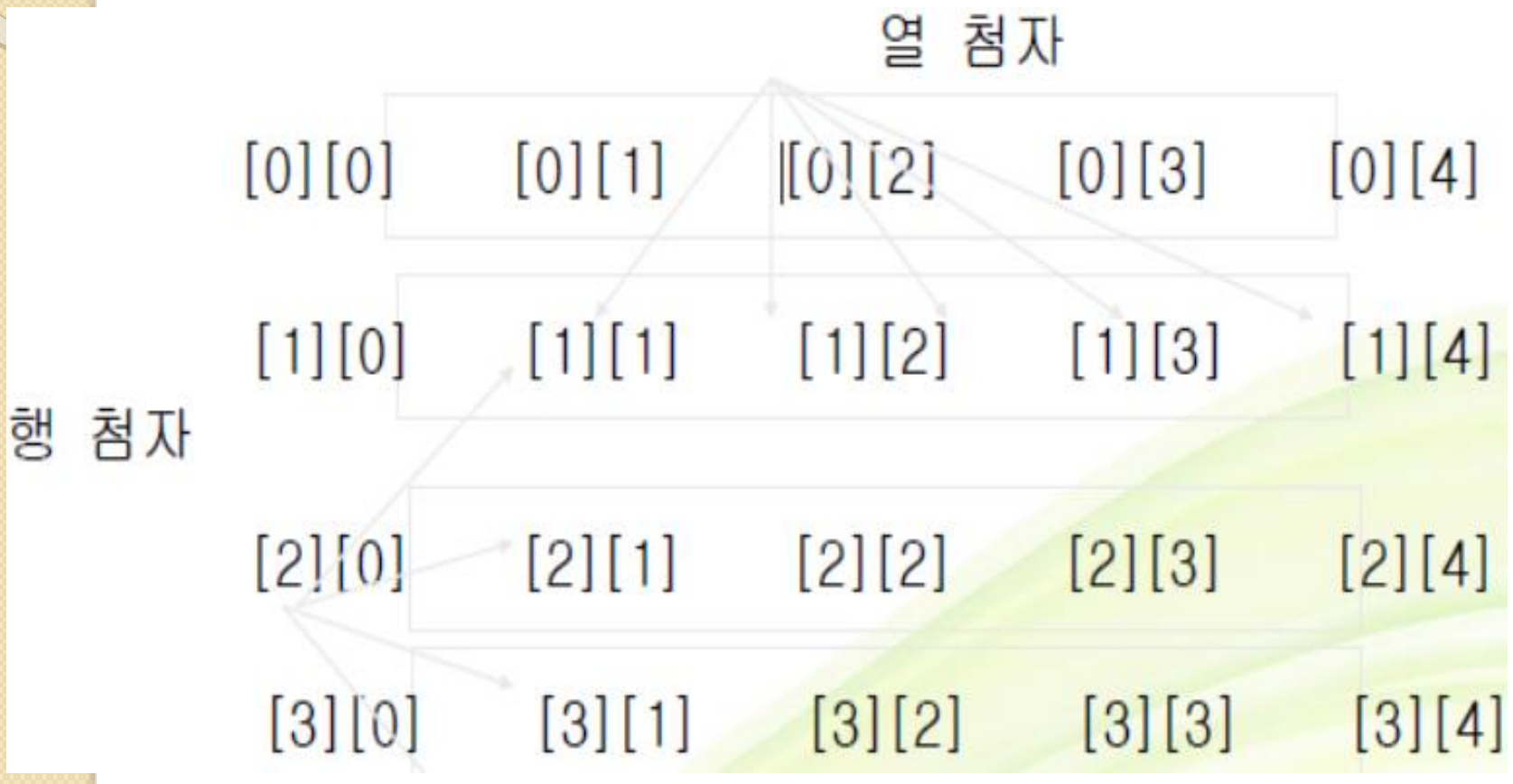


- 2차원 배열은 1차원 배열을 여러 개 묶어서 사용한 형태로 이해



5] 다차원 배열 (2차원 배열) 선언

- 데이터형 배열변수명[][]; 혹은 데이터형[][] 배열변수명;
  int name[ ][ ]; 또는 int[ ][ ] name;(권장)



5] 다차원 배열 (배열의 생성)

- array-name = new type[size];

예 : int name[ ][ ] = new int[4][5];(정적)

int name[ ][ ] = new int[3][ ];(동적)

※ int name[5], int name = new int[ ][3] 은 가능하지 않다.

직접: int[ ] name = {1,2,3,4}, int[ ][ ] name={{1,2},{2,3}}

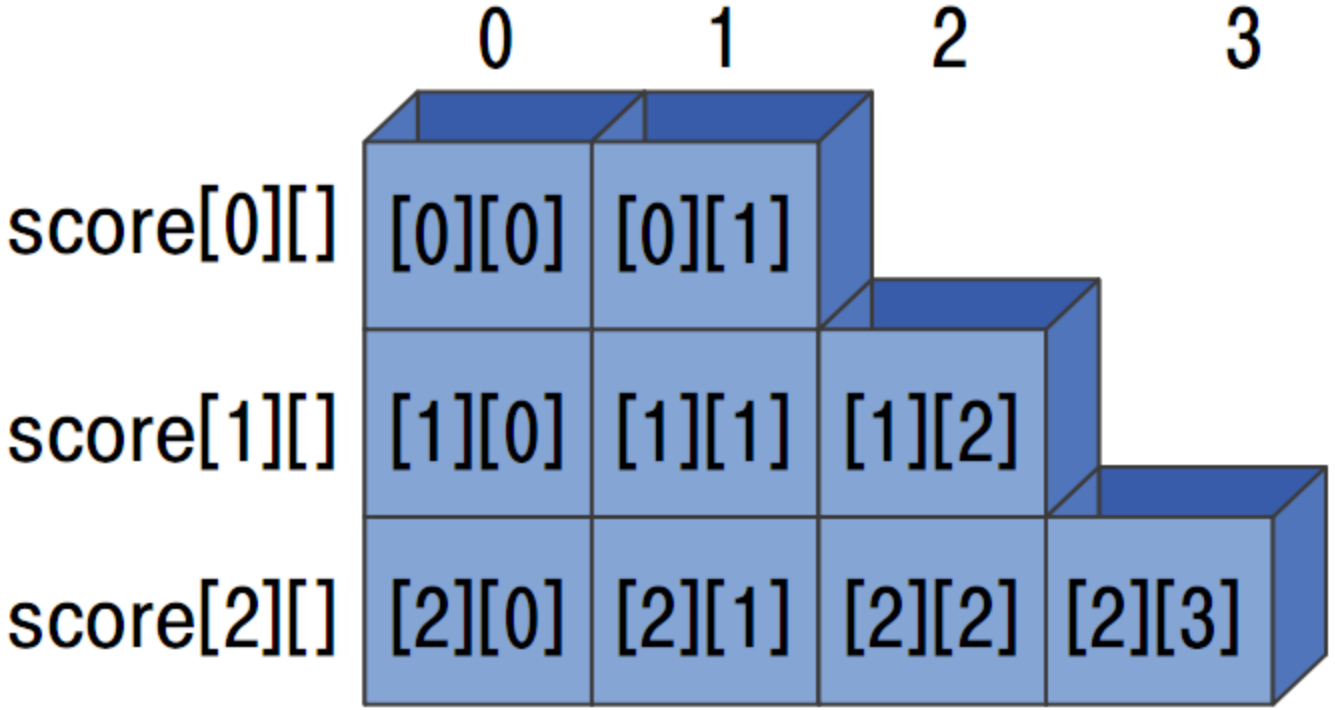
Tip : 자바의 배열은 한번 생성된 후에는 크기가 변할 수 없다

※ name은 정수형 데이터 5개를 갖고 있는 배열의 레퍼런스 변수이다.

-레퍼런스 변수란? 기본 데이터를 값으로 갖는 것이 아니고, 메모리상이 다른 객체나 배열을 가리키고 있는 변수이다

5] 다차원 배열 (배열의 선언과 생성)

- 각 행의 요소의 개수가 다른 배열의 형태

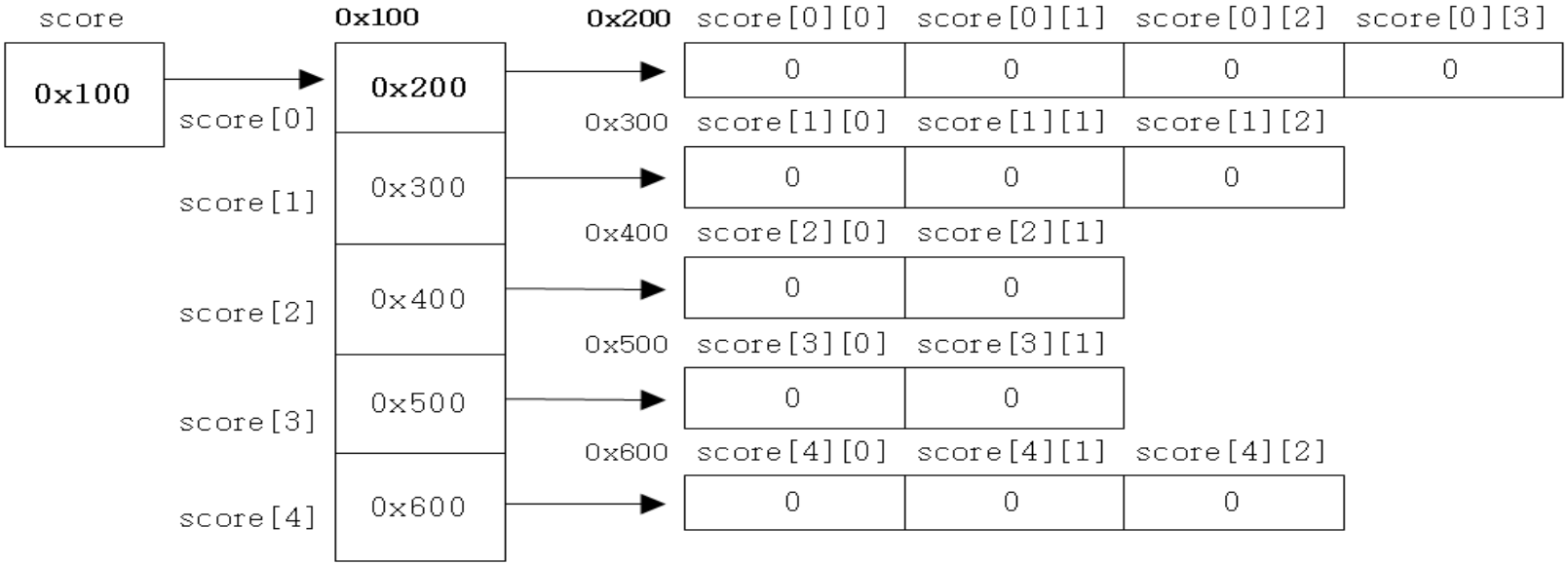


5] 다차원 배열 (가변배열)

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},
        };
```



[그림5-3] 가변배열

6] 배열의 복사

- for문을 이용한 배열의 복사

```
int[] number = {1,2,3,4,5};
int[] newNumber = new int[10];

for(int i=0; i<number.length;i++) {
    newNumber[i] = number[i];// 배열 number의 값을 newNumber에 저장한다.
}
```

number

1	2	3	4	5

newNumber

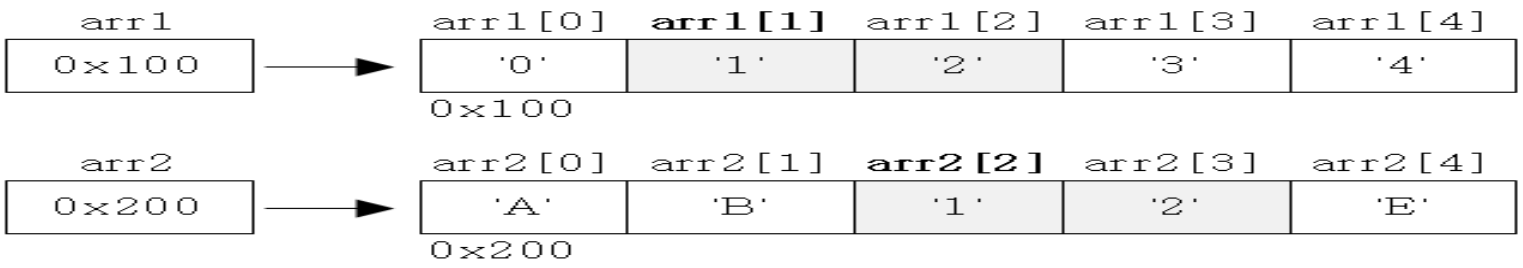
0	0	0	0	0	0	0	0	0	0

- System.arraycopy()를 이용한 배열의 복사

```
System.arraycopy(arr1, 0, arr2, 0, arr1.length);
```

arr1[0]에서 arr2[0]으로 arr1.length개의 데이터를 복사

```
System.arraycopy(arr1, 1, arr2, 2, 2);
```



6] 예시 문제 풀이

ArrayEx12 .java

- 배열 abc와 number를 붙여서 하나의 배열(result)로 만든다
- 배열 abc을 배열 number의 첫 번째 위치부터 배열 abc의 크기만큼 복사
- number의 인덱스6 위치에 3개를 복사

성적표

이름	국어	영어	수학	총점	평균
원더걸스	90	80	70		
비스트	76	86	90		
투피엠	90	78	90		
소녀시대	80	80	80		
합계					

ArrayTest9.java

매출현황

제품명	1월	2월	3월	4월	합계	평균
냉장고	250	170	300	780		
테레비	170	120	150	220		
청소기	450	230	400	250		
총계						

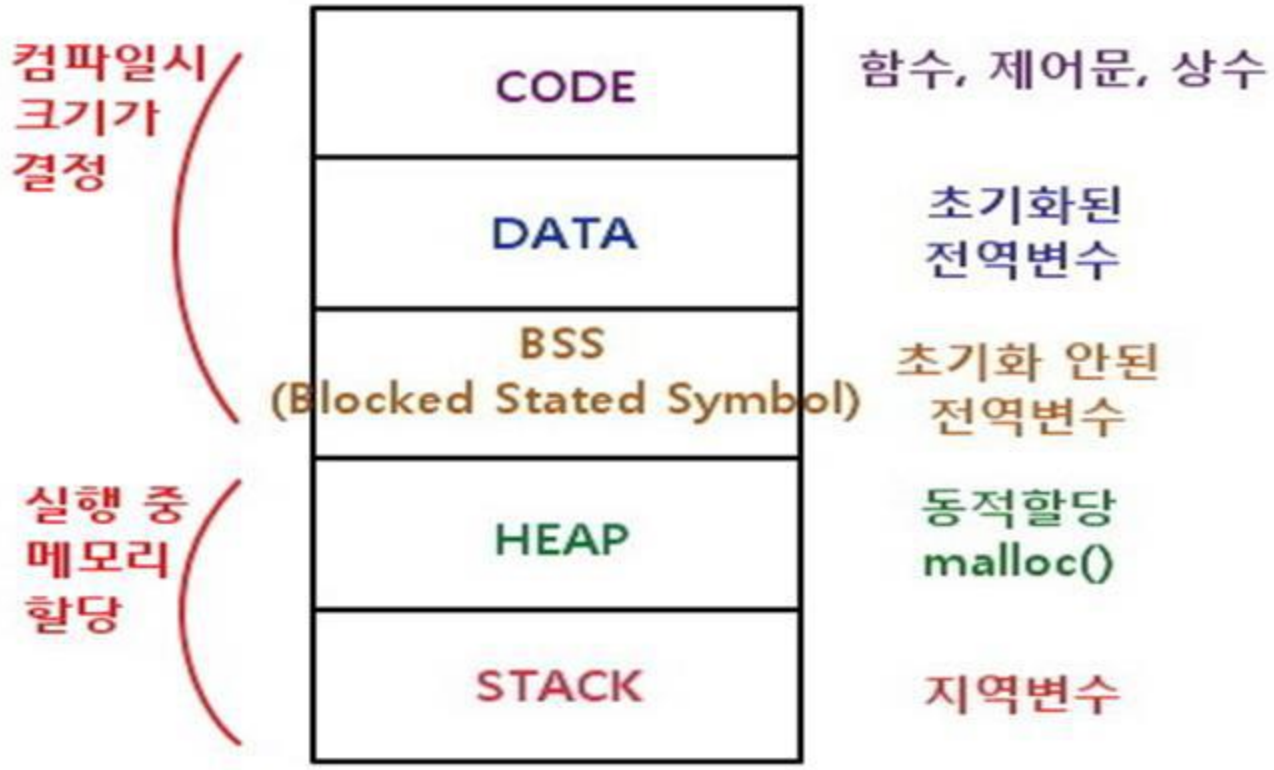
7]프로세스의 메모리 영역1

설정

32비트 시스템에서 프로세스(실행 중인 프로그램) 생성시 4GB의 메모리를 할당받을 수 있는데, 이는 램만으로는 충당하기엔 턱없이 부족하다.

그래서 운영체제는 램과 하드디스크를 하나로 묶어 가상 메모리로 관리한다.

대부분의 시스템에서는 주로 페이징(Paging)이라는 기법으로 가상 메모리를 관리한다. 가상 메모리는 프로세스당 아래와 같은 구조를 가지는데, 모두 5대영역으로 나뉜다. 아래 그림과 같이 데이터를 분류하여 각 영역에 저장하는 것이다



7] 프로세스의 메모리 영역2

운영체계에서 메모리 영역은 각종 데이터의 저장공간이다. 그런데 데이터 별로 용량이 다르고 타입이 다르고 용도가 다르다

1. 코드영역 : 원시소스와 명령어등이 저장되는 영역이다. 우리가 프로그램을 짜면 소스가 모드 코드영역에 text파일로 저장된다

2. 데이터 영역 : 전역변수, 정적변수, 상수 등이 저장된다

3. Heap영역 : 프로그래머가 메모리를 동적할당할때 저장이 되는 영역이다.
프로그래밍을 할 때 일반 변수를 선언하는 것은 정적인 것으로 사용자가 메모리 크기를 유동적으로 조절할 수가 없지만 동적할당을 하면 사용자가 필요한 만큼 메모리를 잡을 수 있다

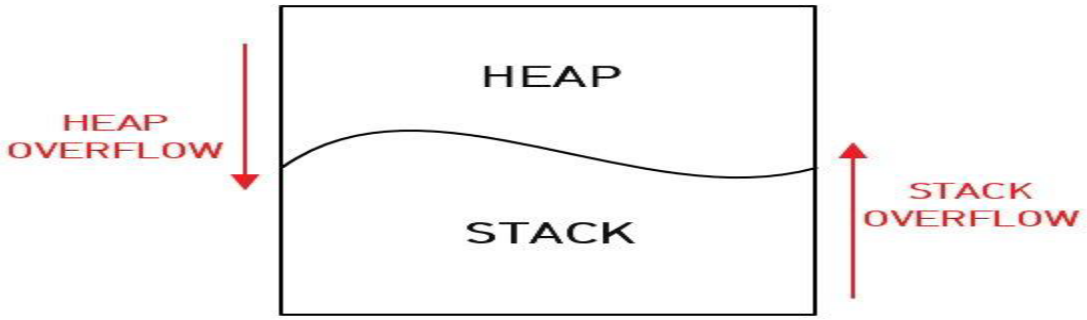
4. 스텍 영역 : 지역변수, 전달인자, 지역함수 등 임시적인 성격의 데이터가 저장되는 곳이다. 지역변수가 왜 임시적이냐 해당 지역에서만 가능하니까.
스텍 영역은 약 1MB정도이기 때문에 용량을 많이 차지하는 변수를 선언할 때는 동적 할당으로 Heap영역에 메모리를 잡으면 된다

7] 프로세스의 메모리 영역3

메모리 영역은 크게 두가지로 나눌 수 있는데 컴파일시 크기가 고정되는 code, data, bss 영역과 실행시 메모리가 할당되었다
반납되는 heap, stack영역으로 나눌 수 있다.

① 메모리 할당이 고정되는 영역 * 코드 영역 - 코드영역은 실행 파일을 구성하는 명령어들이 올라가는 메모리 영역으로 함수, 제어문, 상수 등이 여기에 지정된다.
•데이터 영역 & BSS- 데이터 영역은 BSS와 함께 묶어서 데이터 영역으로 칭하기도 하는데
•이는 전역변수와 Static변수가 지정되는 영역이다.
•초기화 되지 않은 전역변수들은 BSS에 지정된다.

② 실행 중에 메모리를 할당하는 영역(할당과 반납이 이루어짐)
• HEAP 영역 - HEAP 영역은 malloc(), calloc() 등으로 프로그래머가 자율적으로 메모리 크기를 할당할 수 있는 영역이다. 위의 함수들은 free()함수로 할당된 영역을 반납해줘야하므로 동적할당 영역에 속한다.
• STACK 영역 - STACK 영역은 지역변수가 할당되는 영역으로 함수가 호출되면 할당되었다 함수의 종료시 반납되는 영역이다.
위의 HEAP과 STACK영역은 사실 같은 공간을 공유한다. HEAP이 메모리 위쪽 주소부터 할당되면 STACK은 아래쪽 부터 할당되는 식이다.
그래서 각 영역이 상대 공간을 침범하는 일이 발생할 수 있는데
이를 각각 HEAP OVERFLOW, STACK OVERFLOW라고 칭한다.



7] 데이터의 종류와 메모리 영역4

변수의 종류
전역변수 : main()함수 밖에서 선언
지역변수 : 메소드 안에서 선언
일반변수 : 일반적으로 사용하는 변수
정적변수 : 선언할 때 한번만 초기화 되고 이후는 대입만 함

	생성시기	소멸시기
전역변수	프로그램이 시작할 때	프로그램이 끝날 때
지역변수	지역에서 선언할 때	해당지역이 끝날 때
일반변수	지역에서 선언할 때	해당지역이 끝날 때
정적변수	프로그램이 시작할 때	프로그램이 끝날 때