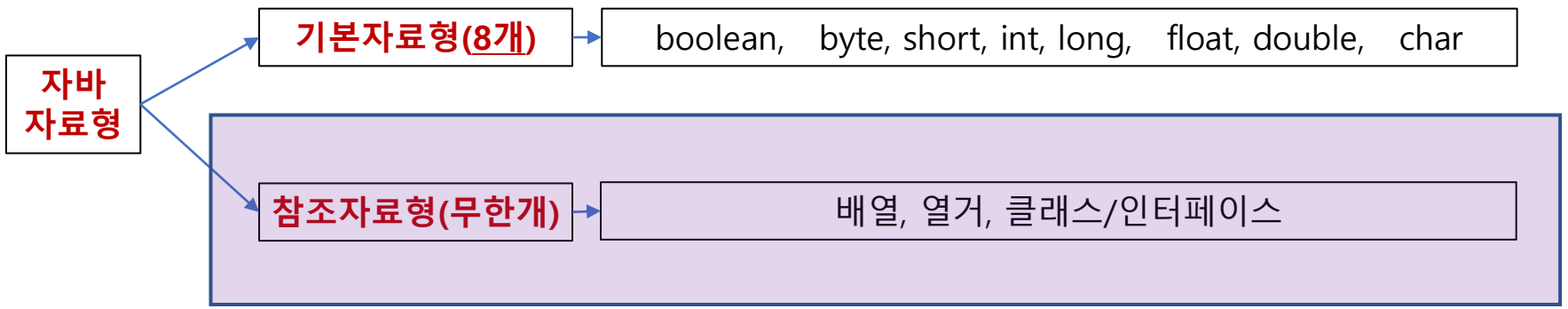
참조자료형

참조 자료형



배열-1차원 배열
(배열의 특징/선언/객체생성/메모리구조)

배열-1차원 배열

☞ 배열의 두 가지 특징

1

특징 1. 동일한 자료형만 묶어서 저장 가능
특징 2. 생성시 크기를 지정 (이후 크기 변경 불가)

☞ 배열의 선언

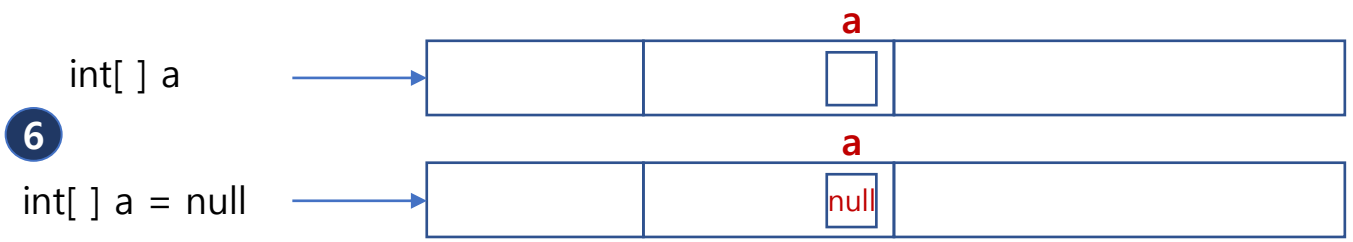
자료형[ ] 변수명 또는 **자료형** 변수명**[ ]**

2

ex
int[ ] a;
double[ ] b;
String[ ] c;

ex
int a[ ] ;
double b[ ];
String c[ ];

6



3

TIP

선언시 배열의 첫번째 특징 (동일한 자료형 저장)이 나타나야 함 (어떤 자료형을 저장하는 배열인지여부)

ex
int[ ] a; → int 자료형만 저장 가능
double[ ] a; → double 자료형만 저장 가능
String[ ] a; → String 자료형만 저장 가능

4

TIP

변수형 앞에 자료형을 두는 것이 일관성이 있음

ex
(int 자료형 a)
int a;

ex
(int 배열 자료형 a)
int[ ] a;

5

TIP

메모리의 구조

배열-1차원 배열

☞ 배열의 **객체 생성**

- 여러 개의 값을 저장할 수 있는 공간
- **힙(Heap) 메모리**에 생성

1

new 자료형[배열의 길이]

☞ 배열의 선언과 객체 생성

- 방법1. 선언과 동시에 객체 생성
- 방법2. 선언 이후 객체 생성

2

ex int[ ] a = new int[3];

ex
```
int[ ] a;
a = new int[3];
```

cf.
```
int a = 3;

int a;
a=3;
```

3

TIP

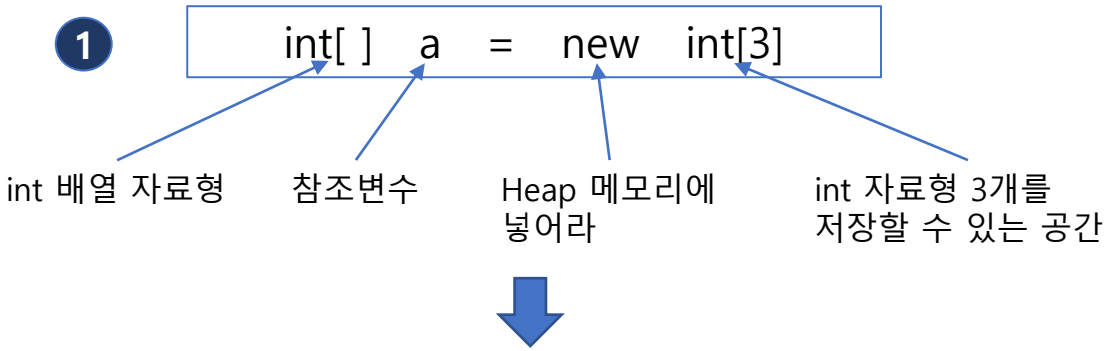
배열 **객체 생성**시 배열의 두번째 특징 (생성시 개수가 지정되어야 함)이 나타나야 함

ex
```
int[ ] a = new int[3]; (O)
double[ ] a = new double[5]; (O)
String[ ] a = new String[10]; (O)

int[] a = new int[ ]; (X) → 오류가 발생하는 이유는?
```

배열-1차원 배열

☞ 배열 객체의 선언 및 생성시 **메모리 구조**

1



2

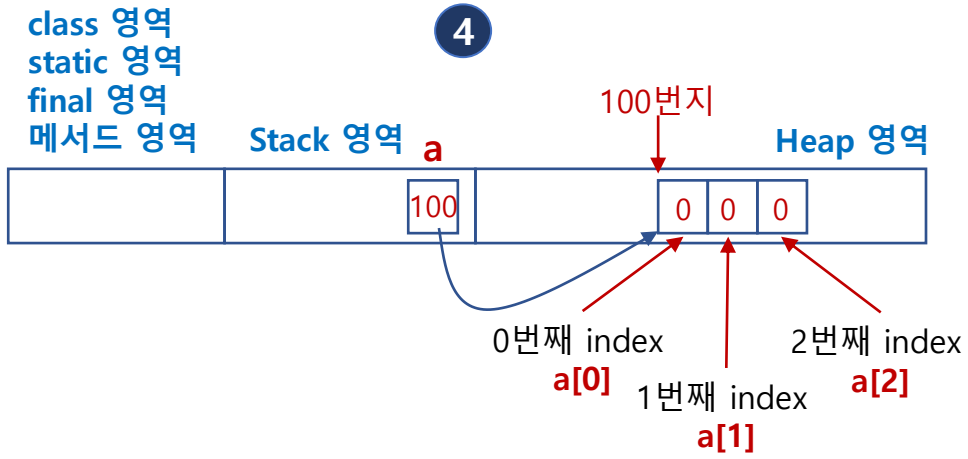
int 자료형 3개를 저장할 수 있는 공간을 Heap 메모리에 넣어두고 어디에 넣었는지를 참조변수 a에 저장하여라!

3

TIP

new 키워드
- "Heap 메모리에 넣어라 " 의 의미
- JVM이 비워있는 공간에 객체를 생성
- 어느 공간에 넣었는지 알 수가 없기 때문에 JVM은 객체의 위치정보를 리턴해주어야 함
- 리턴된 값을 참조변수에 저장

4



배열-1차원 배열

☞ 배열 객체의 **값 대입 및 값 읽기**

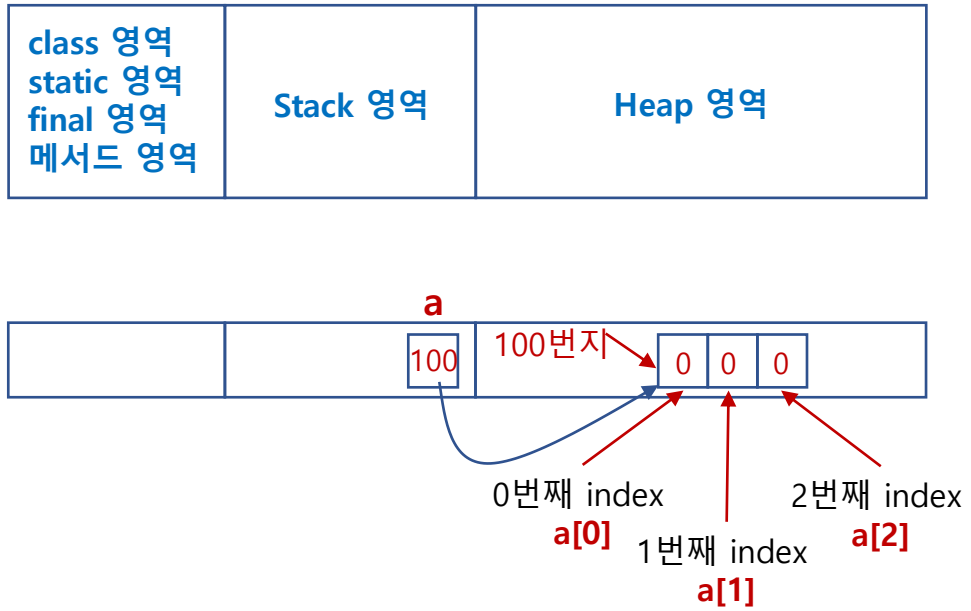
- 배열의 값 대입 ①

참조변수명[인덱스번호]=값

ex

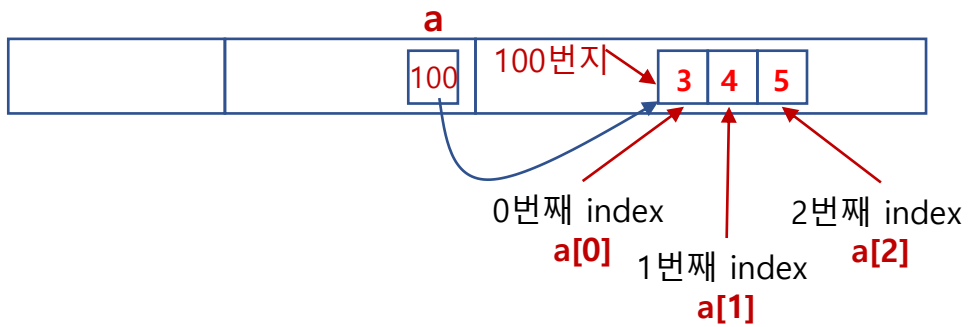
int[] a = new int[3]

②




```
a[0]=3;
a[1]=4;
a[2]=5;
```

③



- 배열의 값 읽기 ④

참조변수명[인덱스번호]

ex

⑤ System.out.println(a[0]); → 3 | System.out.println(a[1]); → 4 | System.out.println(a[2]); → 5

The End

배열-1차원 배열
(세 가지 초기값 부여방식 및 출력하기)

배열-1차원 배열

☞ **1차원** 배열의 생성 및 값 대입

1

▶ 방법 1 (배열 객체의 생성 + 값 대입)

```
자료형[] 참조변수명 = new 자료형 [배열의 길이];

참조변수명[0]=값;
참조변수명[1]=값;
...
참조변수명[배열의 길이-1]=값;
```

ex

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

2

▶ 방법 2 (배열 객체의 생성 및 값 대입)

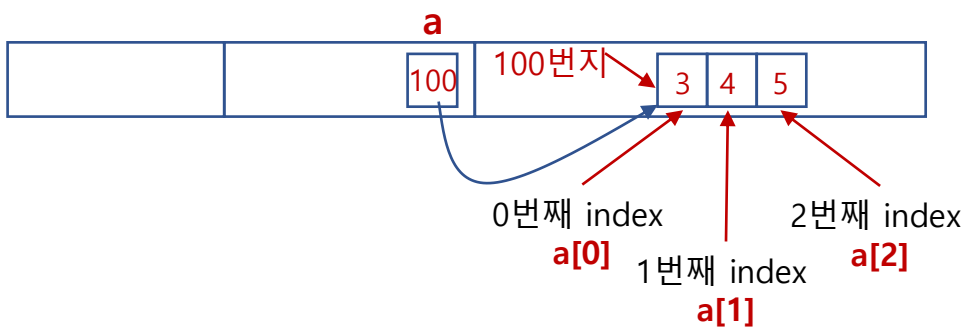
```
자료형[] 참조변수명 = new 자료형[ ]{값, 값, ..., 값};
```

이 경우 배열의 길이는 쓰지 않음

배열의 길이는 값의 개수로 결정

ex

```
int[] a = new int[]{3,4,5};
```



배열-1차원 배열

☞ **1차원** 배열의 생성 및 값 대입

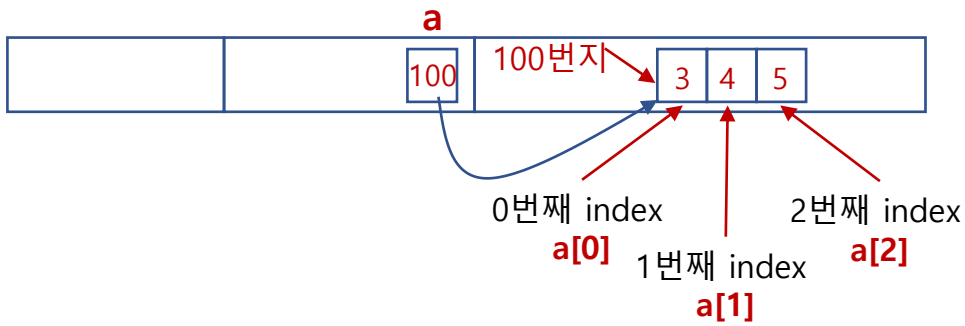
1

▶ 방법 3 (값 대입) : 대입할 값만 입력

자료형[] 참조변수명 = {값, 값, ..., 값};

ex

```
int[] a = {3, 4, 5}
```



2

▶ 방법 2 vs. 방법 3

- 방법 2는 선언과 값의 대입 분리 가능
- 방법 3은 선언과 값의 대입 분리 불가능

ex

```
방법2.
    int[] a = new int[]{3,4,5}; (O)

    int[] a;
    a = new int[]{3,4,5}; (O)

방법 3.
    int[] a = {3,4,5}; (O)

    int[] a;
    a = {3,4,5}; (X)
```

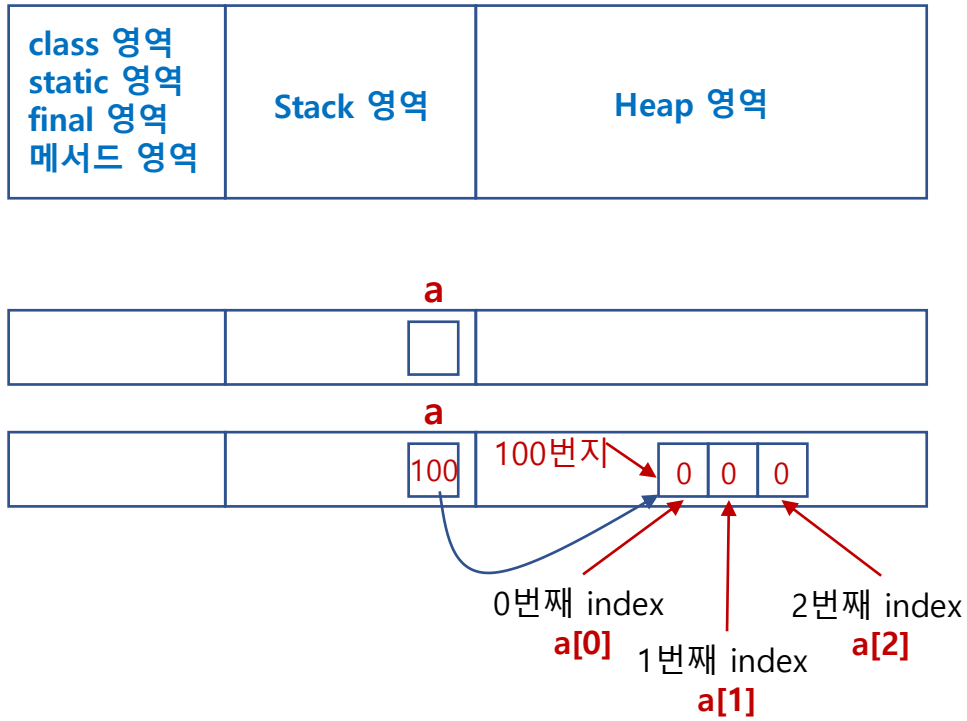
배열-1차원 배열

☞ 배열 객체의 **강제 초기값**

1

```
int[ ]  a ;
```

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



2

TIP

Stack 메모리의 초기값

- 초기값을 부여하지 않는 경우 **빈칸**으로 존재 (읽기 불가능)

ex

int a; → stack 메모리에 공간 생성 (빈칸으로 존재)
System.out.println(a); → 오류발생

Heap 메모리의 초기값

- 빈칸으로 존재할 수 없으며 디폴트 초기값이 강제 설정
- 기본자료형
  숫자(int, double 등) 디폴트값 : 0
  boolean 디폴트: false
- 참조자료형 디폴트값: null

3

ex

```
int a;
int[] b;
System.out.println(a); (X)
System.out.println(b); (X)

int a=0;
int[] b=null; //null: 객체를 가리키지 않음
System.out.println(a); (O)
System.out.println(b); (O)
```

배열-1차원 배열

☞ 참조자료형으로서의 배열의 특징

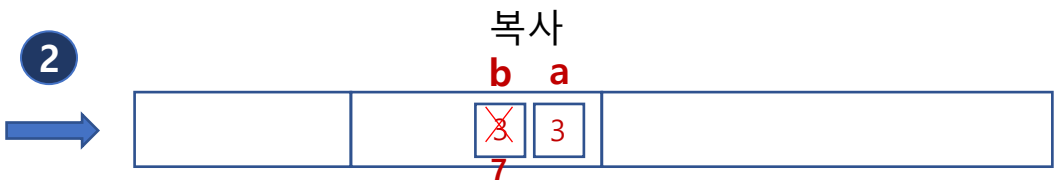
TIP

- '='는 stack 메모리의 값을 복사
- 기본자료형의 변수 복사: 값의 복사
- 참조자료형의 변수 복사: 위치(번지) 복사

▶ 기본자료형의 변수 복사

ex

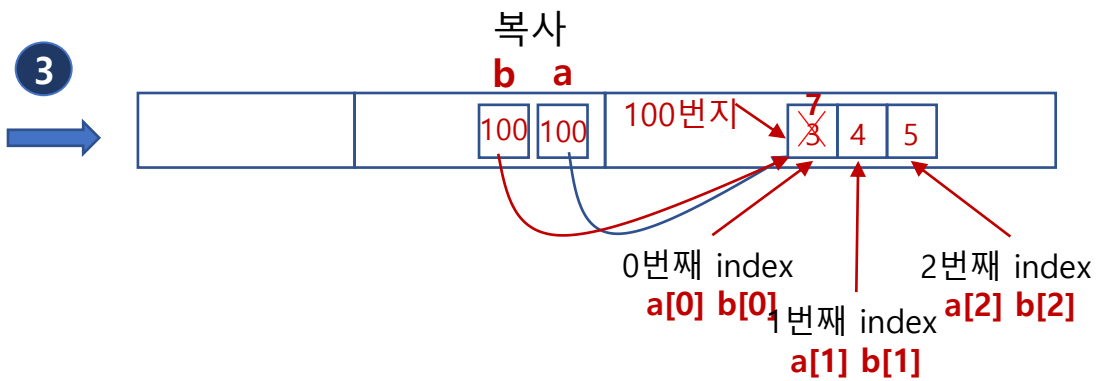
```
int a = 3;
int b = a;
b=7;
System.out.println(a); → 3
System.out.println(b); → 7
```



▶ 참조자료형의 변수 복사

ex

```
int[] a = {3, 4, 5};
int[] b = a;
b[0]=7;
System.out.println(a[0]); → 7
System.out.println(b[0]); → 7
```



배열-1차원 배열

☞ 반복문을 이용한 데이터 읽기

▶ 배열의 길이 가져오기

배열의 길이 = 배열참조변수명.length

1

ex

```
int[] a = new int[]{3,4,5,6,7};
System.out.println(a.length); → 5
```

▶ 배열의 길이가 큰 데이터 읽기

2

ex

```
int[] a = new int[100];
a[0]=1, a[1]=2, …, a[99]=100;

System.out.println(a[0]); → 1
System.out.println(a[1]); → 2
…
System.out.println(a[99]); → 100
```

100 라인

배열-1차원 배열

☞ 반복문을 이용한 데이터 읽기

TIP

for-each 구문 :
for(원소자료형 변수 : 묶음 참조자료형) {
집합객체 원소의 자료형 / 배열, 컬렉션 등 집합객체
}

2 배열 또는 컬렉션과 같이 다수의 원소를 가지고 있는 객체에서 원소를 하나씩 차례로 꺼내는 기능을 반복적으로 수행

▶ for 반복문을 이용한 배열 데이터 읽기

ex

1
```
int[] a = new int[100];
a[0]=1, a[1]=2, …, a[99]=100;

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

▶ for-each 반복문을 이용한 배열 데이터 읽기

ex

3
```
int[] a = new int[100];
a[0]=1, a[1]=2, …, a[99]=100;

for(int k: a){
    System.out.println(k);
}
```

TIP

4 **Arrays.toString(1차원배열) → 배열값 출력 정적 메서드** ex) System.out.println(Arrays.toString(new int[]{1,2,3}); //[1,2,3]

The End

배열-2차원 배열

배열-2차원 배열

☞ 2차원(정방행렬) 배열

- 가로 및 세로 방향으로 정렬된 배열

1 **int[][] a** →

	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

2

TIP

선언시 배열의 첫번째 특징 (동일한 자료형 저장)이 나타나야 함 (어떤 자료형을 저장하는 배열인지여부)

ex

int[ ][ ] a; → int 자료형만 저장 가능
double[ ][ ] a; → double 자료형만 저장 가능
String[ ][ ] a; → double 자료형만 저장 가능

☞ 2차원(다차원) 배열의 선언

자료형[ ][ ] 변수명 또는 자료형 변수명[ ][ ] 또는 자료형[ ] 변수명[ ]

3

ex
```
int[ ][ ] a;
double[ ][ ] b;
String[ ][ ] c;
```

ex
```
int a[ ][ ] ;
double b[ ][ ];
String c[ ][ ];
```

ex
```
int[] a[ ] ;
double[] b[ ];
String[] c[ ];
```

4

TIP

변수형 앞에 자료형을 두는 것이 일관성이 있음
(2차원도 동일)

ex
(int 자료형 a)
int a;

ex
(2차원 int 배열 자료형 a)
int[ ][ ] a;

배열-2차원 배열

☞ 2차원(정방행렬) 배열의 **객체 생성**

▶ 방법 1 (배열 객체의 생성 + 값 대입)

1

```
자료형[ ][ ] 참조변수명
            = new 자료형[행의 길이] [열의 길이];

참조변수명[0][0]=값;
참조변수명[0][1]=값;
...
참조변수명 [행의 길이-1] [열의 길이-1]=값;
```

ex

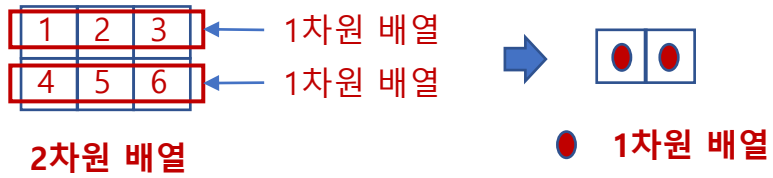
```
int[][] a = new int[2][3];
a[0][0]=1;
a[0][1]=2;
a[0][2]=3;
a[1][0]=4;
a[1][1]=5;
a[1][2]=6;
```

1	2	3
4	5	6

2

TIP

- 메모리는 데이터를 1차원으로만 저장 가능
- 2차원 데이터를 저장하기 위해서는 1차원(행의 단위)으로 나누어 저장 **(1차원 배열을 원소로 가지는 1차원 배열)**

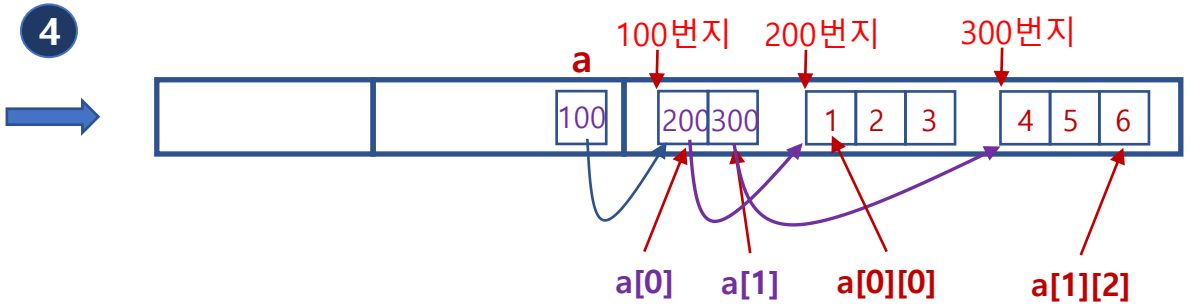


3

int[] a; → int를 저장할 수 있는 1차원 배열

int[] [] a; → int[]를 저장할 수 있는 1차원 배열

4



5

ex

```
System.out.println(a.length);    // → 2
System.out.println(a[0].length); // → 3
System.out.println(a[1].length); // → 3
```

배열-2차원 배열

5

Q. 배열의 두 번째 특징(선언시 크기가 지정되어 있어야 함)을 위배?

위배되지 않음. 참조변수 a가 가리키는 배열은 몇 행인지만 정해지면 배열 선언이 가능하기 때문에 행의 정보만 있으면 배열 생성 가능

☞ 2차원(정방행렬) 배열의 객체 생성

▶ **방법 2**

1

- 2차원 배열 객체의 행 성분(1차원 배열)만 생성
- 각 행에 열 성분(1차원 배열) 생성

2

```
자료형[ ][ ] 참조변수명
                = new 자료형[행의 길이] [ ];

참조변수명[0] = 1차원 배열의 생성;
참조변수명[1] = 1차원 배열의 생성;
...
참조변수명 [행의 길이-1] = 1차원 배열의 생성;
```

열의 길이는 표시하지 않음

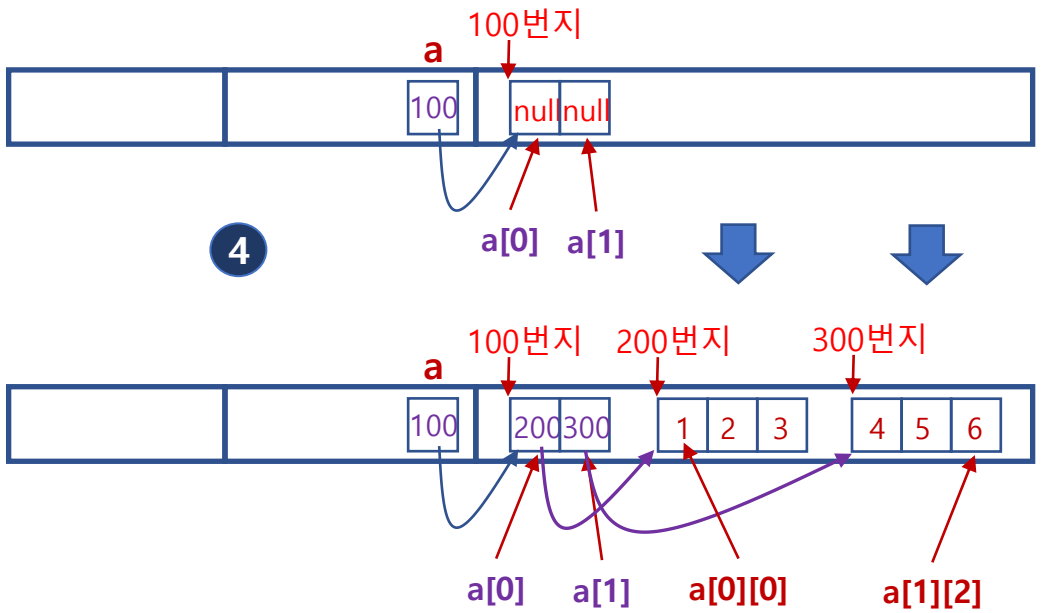
3

ex 각 행의 1차원 배열을 방법 1로 생성

```
int[][] a = new int[2][];
a[0] = new int[3];
a[0][0]=1; a[0][1]=2; a[0][2]=3;

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

ex 각 행의 1차원 배열을 방법 2로 생성

```
int[][] a = new int[2][];
a[0] = new int[]{1, 2, 3};
a[1] = new int[]{4, 5, 6};
```



배열-2차원 배열

☞ 2차원(정방행렬) 배열의 객체 생성

▶ **방법 3 (값 대입) : 자료형과 대입할 값만 입력**

1

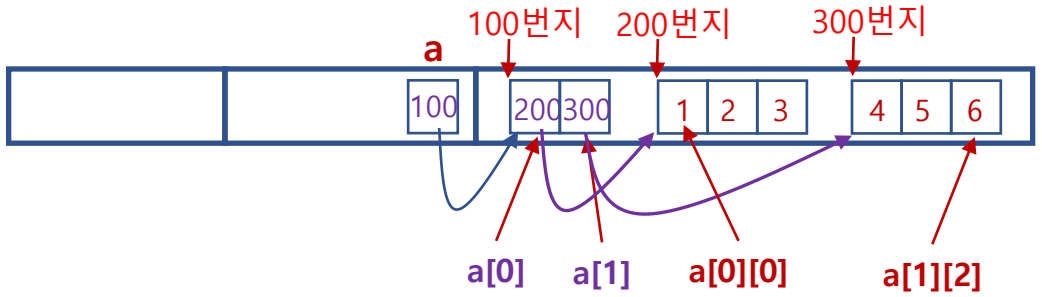
```
자료형[ ][ ] 참조변수명
  = new 자료형[ ][ ] {{값, 값, ..., 값}, ..., {값, 값, ..., 값}};
```

이 경우 배열의 길이는 쓰지 않음 — 0번째 행 데이터 — 마지막 행 데이터

ex

```
int[][] a = new int[][]{{1,2,3}, {4,5,6}};
```

2



3

ex

```
System.out.println(a.length);    // → 2
System.out.println(a[0].length); // → 3
System.out.println(a[1].length); // → 3
```

4

TIP

1차원 배열 때와 동일하게 분리하여 표형 가능

ex

```
int[][] a = new int[][]{{1,2,3}, {4,5,6}}; (O)

int[][] a;
a = new int[][]{{1,2,3}, {4,5,6}}; (O)
```

배열-2차원 배열

☞ 2차원(정방행렬) 배열의 객체 생성

▶ **방법 4 (값 대입) : 대입할 값만 입력**

1

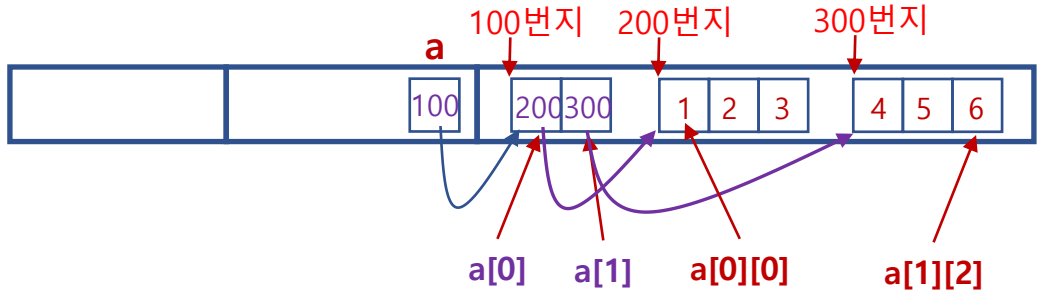
```
자료형[ ][ ] 참조변수명
        = {{값, 값, …, 값}, …, {값, 값, …, 값}};
```

0번째 행 데이터 / 마지막 행 데이터

ex

```
int[][] a = {{1,2,3}, {4,5,6}}; (O)
```

2



3

TIP

값만 대입하는 방법 4의 경우 1차원 배열과 마찬가지로 선언과 동시에 값을 대입하는 경우만 가능

ex

```
int[][] a;
a = {{1,2,3}, {4,5,6}}; (X)
```

배열-2차원 배열

1	2	
3	4	5

☞ 2차원(비정방행렬) 배열

1 - 행마다 열의 개수가 서로 다른 2차원 행렬

▶ **방법 1**
- 2차원 배열 객체의 행 성분(1차원 배열)만 생성
- 각 행에 열 성분(1차원 배열) 생성

2
```
자료형[ ][ ] 참조변수명
            = new 자료형[행의 길이] [];

참조변수명[0] = 1차원 배열의 생성;
참조변수명[1] = 1차원 배열의 생성;
...
참조변수명 [행의 길이-1] = 1차원 배열의 생성;
```
열의 길이는 표시하지 않음

int[][] a →

	0	1	2	3
0	a[0][0]	a[0][1]		
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	

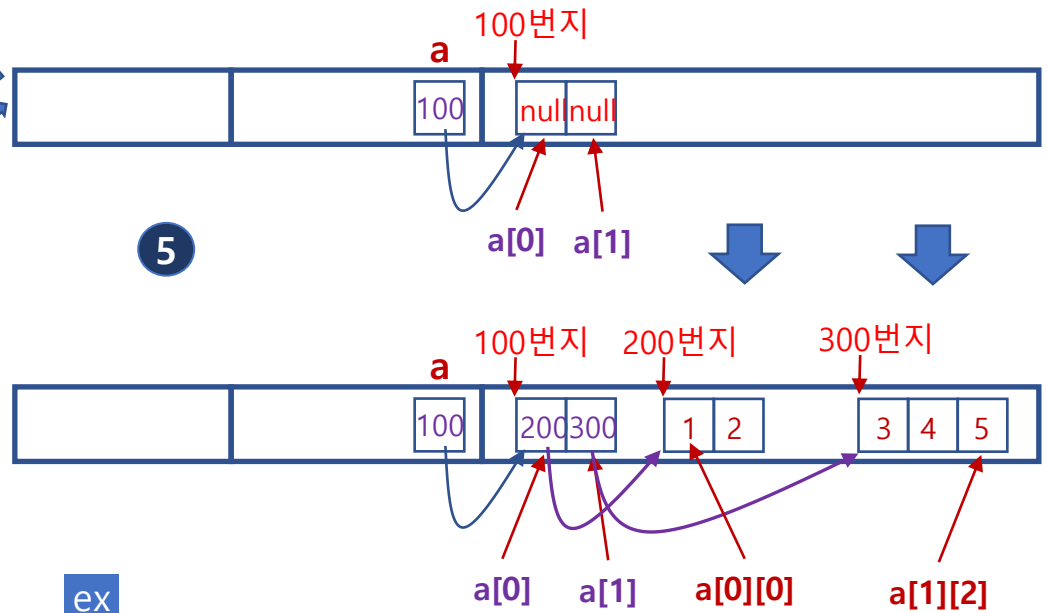
3 ex 각 행의 1차원 배열을 방법 1로 생성
```
int[][] a = new int[2][];
a[0] = new int[2];
a[0][0]=1; a[0][1]=2;

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

1	2	
3	4	5

4 ex 각 행의 1차원 배열을 방법 2로 생성
```
int[][] a = new int[2][];
a[0] = new int[]{1, 2};
a[1] = new int[]{3, 4, 5};
```

1	2	
3	4	5

5



6 ex
```
System.out.println(a.length);     // → 2
System.out.println(a[0].length); // → 2
System.out.println(a[1].length); // → 3
```

배열-2차원 배열

☞ 2차원(비정방행렬) 배열

- 행마다 열의 개수가 서로 다른 2차원 행렬

3

ex

```
System.out.println(a.length);    // → 2
System.out.println(a[0].length); // → 2
System.out.println(a[1].length); // → 3
```

▶ **방법 2 (값 대입) : 자료형과 대입할 값만 입력**

1

```
자료형[ ][ ] 참조변수명
  = new 자료형[ ][ ] {{값, 값, …, 값}, …, {값, 값, …, 값}};
```

이 경우 배열의 길이는 쓰지 않음

0번째 행 데이터

마지막 행 데이터

TIP

1차원 배열 때와 동일하게 분리하여 표현 가능

4

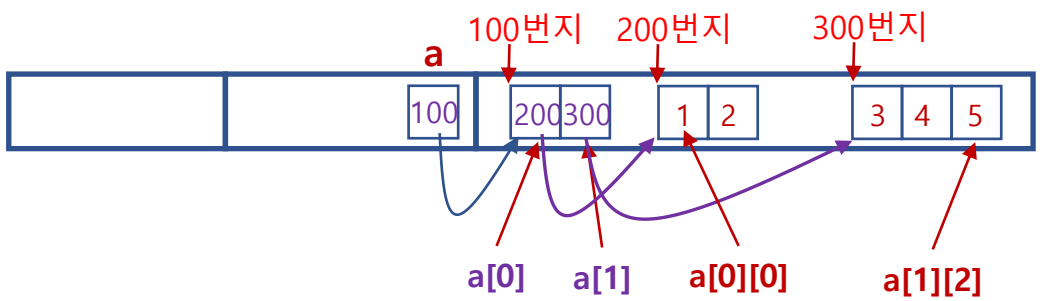
ex

```
int[][] a = new int[][]{{1,2}, {3,4,5}}; (O)

int[][] a;
a = new int[][]{{1,2}, {3,4,5}}; (O)
```

ex

```
int[][] a = new int[][]{{1,2}, {3,4,5}};
```

2



배열-2차원 배열

☞ 2차원(비정방행렬) 배열

- 행마다 열의 개수가 서로 다른 2차원 행렬

▶ **방법 3 (값 대입) : 대입할 값만 입력**

1

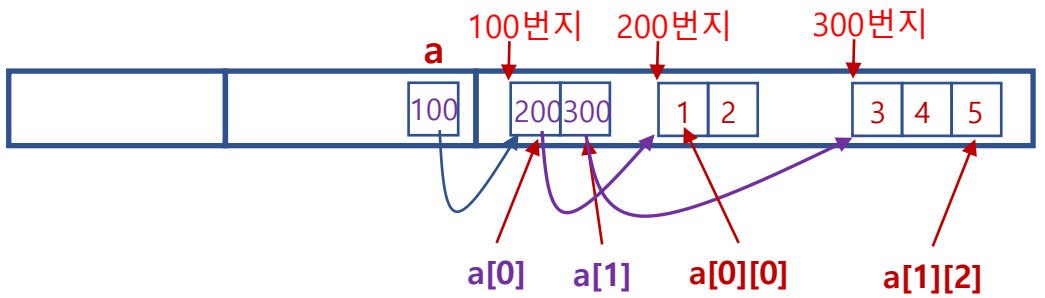
```
자료형[ ][ ] 참조변수명
        = {{값, 값, …, 값}, …, {값, 값, …, 값};
```

0번째 행 데이터 / 마지막 행 데이터

ex

```
int[][] a = {{1,2}, {3,4,5}}; (O)
```

2



3

ex

```
System.out.println(a.length);    // → 2
System.out.println(a[0].length); // → 2
System.out.println(a[1].length); // → 3
```

4

TIP

값만 대입하는 방법 3의 경우 1차원 배열과 마찬가지로 선언과 동시에 값을 대입하는 경우만 가능 (분리 불가능)

ex

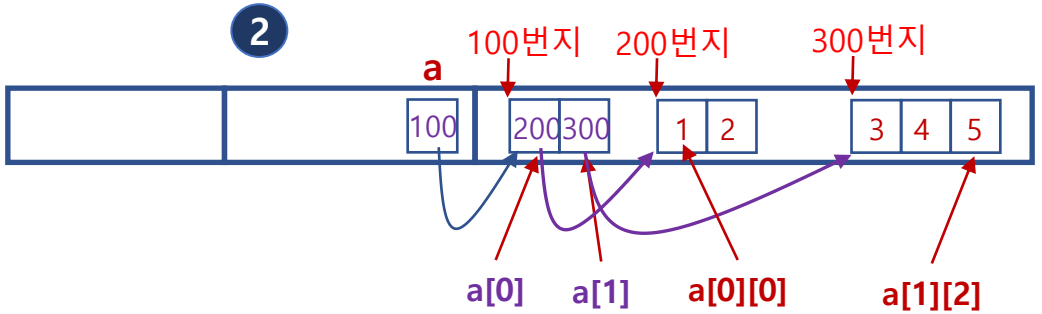
```
int[][] a = {{1,2}, {3,4,5}}; (O)

int[][] a;
a = {{1,2}, {3,4,5}}; (X)
```

배열-2차원 배열

☞ 2차원 배열의 출력

- 행마다 열의 개수가 서로 다른 2차원 행렬

2



▶ for 반복문을 이용한 2차원 배열 데이터 읽기

1

ex

```
int[][] a = {{1,2},{3,4,5}};

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

1	2	
3	4	5

3

ex

```
System.out.println(a.length);    // → 2
System.out.println(a[0].length); // → 2
System.out.println(a[1].length); // → 3
```

▶ for-each 반복문을 이용한 배열 데이터 읽기

4

ex

```
int[][] a = {{1,2},{3,4,5}};

for(int[] m: a){
    for(int n: m){
        System.out.println(n);
    }
}
```

1	2	
3	4	5

5

TIP

2차원 배열의 원소는 1차원 배열이기 때문에 2차원 배열의 for-each를 통해 추출된 원소는 **1차원 배열**값임

배열-main() 메서드 매개변수

배열-main() 메서드 매개변수

☞ main 메서드의 매개변수

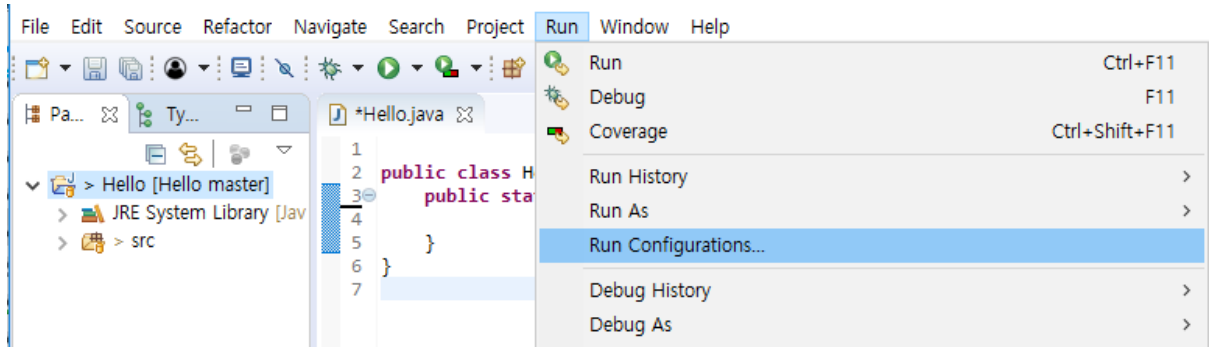
- main 메서드가 처음 실행될 때 **main 메서드로 매개변수 전달** 가능

Run → Run Configurations → (x)=Arguments

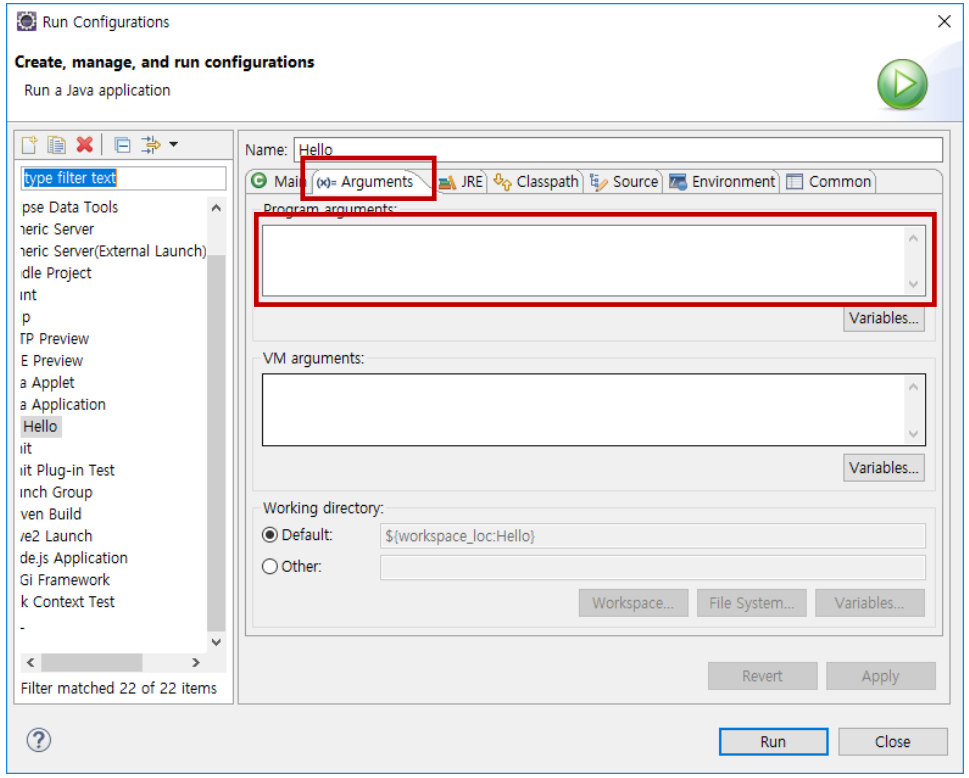
1

```
public static void main(String[] ar){
   …
}
```

1



3



배열-main() 메서드 매개변수

☞ main 메서드의 매개변수

- main함수가 처음 실행될 때 main 함수에 매개변수 전달 가능

ex

```
public static void main(String[] ar){
  String a = ar[0];
  String b = ar[1];
  String c = ar[2];

  System.out.println(a);
  System.out.println(b);
  System.out.println(c);

  System.out.println(b+1);
  System.out.println(c+1);

  int d = Integer.parseInt(b);
  double e = Double.parseDouble(c);

  System.out.println(d+1);
  System.out.println(e+1);
}
```

2

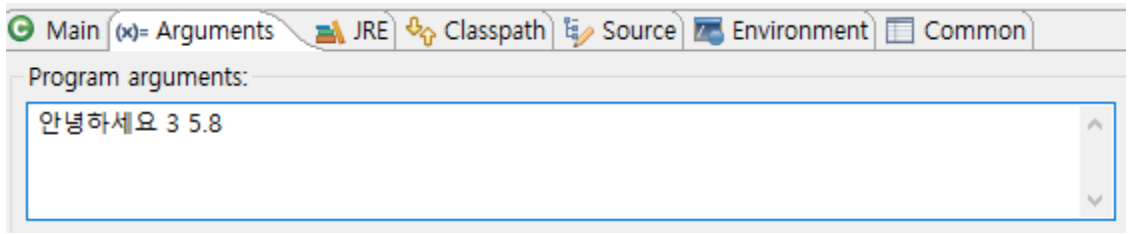
TIP

3

문자열 → 정수 : Integer.parseInt(문자열)
문자열 → 실수 : Double.parseDouble(문자열)

1

Run → Run Configurations → (x)=Arguments



4

```
안녕하세요
3
5.8
31
5.81
4
6.8
```

The End

String 참조자료형

String 참조자료형

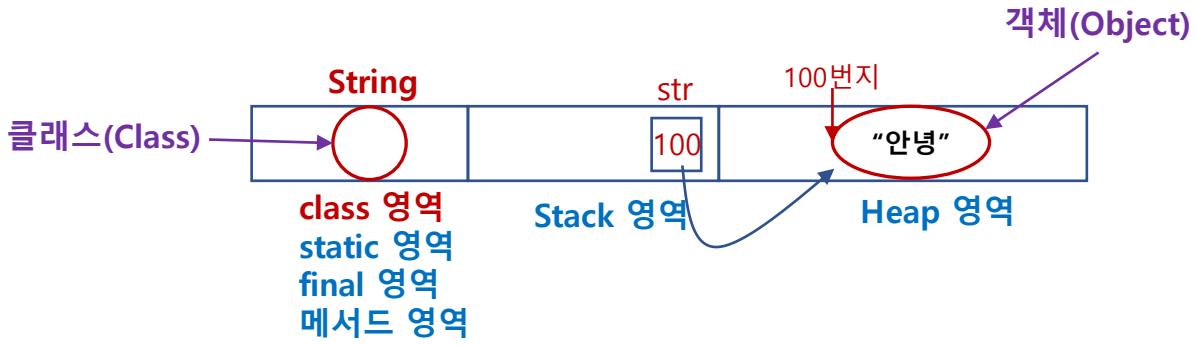
5

TIP

1 클래스(Class)는 타입의 구조(붕어빵기계)를 나타내며
객체(Object)는 해당 타입으로 생성된 실체(붕어빵)를 나타냄

☞ String **클래스**

2
- 문자열 저장 클래스 타입
- 문자열은 쌍따옴표 (" ")안에 표기



☞ String **객체**를 생성하는 2가지 방법

▶ 방법 1 : new 키워드를 사용한 객체 생성

3

String 참조변수명 = new String("문자열");

ex

String str = new String("안녕");

▶ 방법 2 : 참조변수에 바로 문자열 리터럴 입력

4

String 참조변수명 = "문자열";

ex

String str = "안녕";

String 참조자료형

☞ 일반적인 참조자료형과는 다르게 String 클래스만이 가지고 있는 **2가지 특징**

1

특징 1. 객체내의 값 변경 불가능 → 값 변경시 새로운 객체를 생성하여 작성
특징 2. 리터럴을 바로 입력한 데이터는 문자열이 같은 경우 하나의 객체를 공유

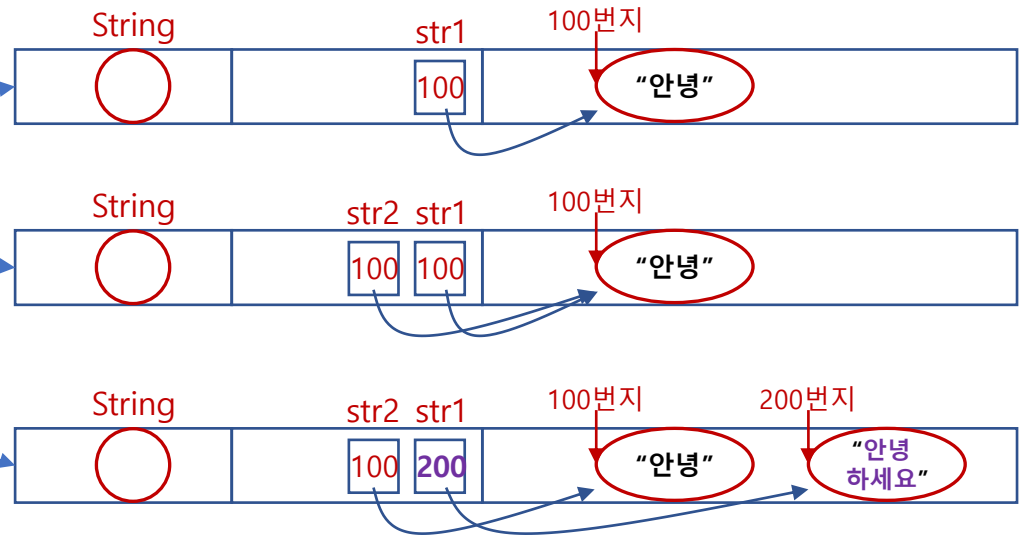
▶ **특징 1 :** 객체내의 값 변경 불가능 → **값 변경시 새로운 객체를 생성하여 작성**

3

2

ex

```
String str1 = new String("안녕");
String str2 = str1;
str1 = "안녕하세요";

System.out.println(str1); //안녕하세요
System.out.println(str2); //안녕
```



String 참조자료형

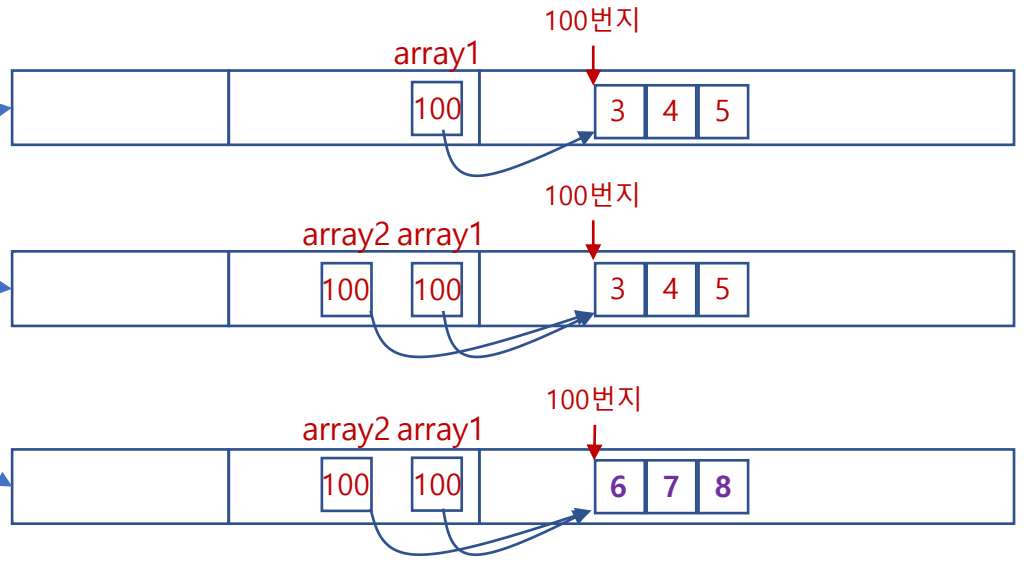
☞ 일반적인 참조자료형과는 다르게 String 클래스만이 가지고 있는 2가지 특징

특징 1. 객체내의 값 변경 불가능 → 값 변경시 새로운 객체를 생성하여 작성
특징 2. 리터럴을 바로 입력한 데이터는 문자열이 같은 경우 하나의 객체를 공유

▶ **특징 1 :** 객체내의 값 변경 불가능 → **값 변경시 새로운 객체를 생성하여 작성** 2

cf **참조자료형 배열의 경우**

1
```
int[] array1 = new int[] {3, 4, 5};
int[] array2 = array1;
array1[0]=6; array1[1]=7; array1[2]=8;

System.out.println(Arrays.toString(array1));
                                    //[6, 7, 8]
System.out.println(Arrays.toString(array2));
                                    //[6, 7, 8]
```



TIP
3 Arrays.toString(1차원 배열) → 배열값 출력 정적 메서드

String 참조자료형

☞ 일반적인 참조자료형과는 다르게 String 클래스만이 가지고 있는 2가지 특징

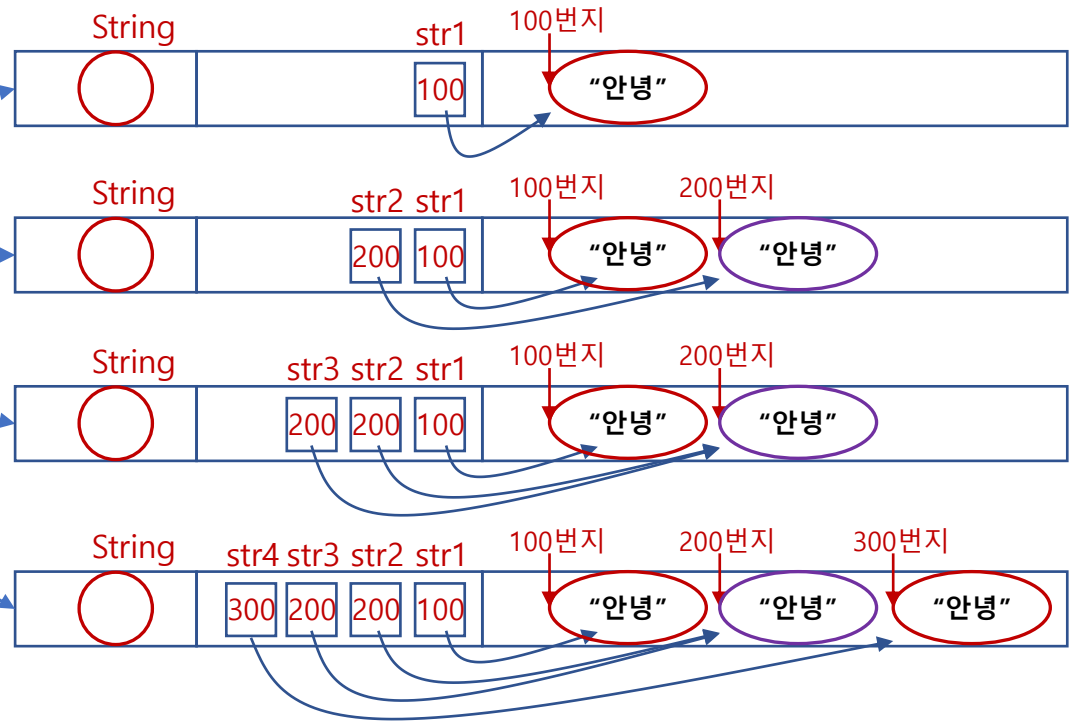
특징 1. 객체내의 값 변경 불가능 → 값 변경시 새로운 객체를 생성하여 작성
특징 2. 리터럴을 바로 입력한 데이터는 문자열이 같은 경우 하나의 객체를 공유

1 ▶ **특징 2 : 리터럴을 바로 입력한 데이터**는 문자열이 같은 경우 하나의 객체를 **공유** 3

ex

2

```
String str1 = new String("안녕");
String str2 = "안녕";
String str3 = "안녕";
String str4 = new String("안녕");

//@stack메모리값 비교 (객체의 번지(위치))
System.out.println(str1==str2); //false
System.out.println(str2==str3); //true
System.out.println(str3==str4); //false
System.out.println(str4==str1); //false
```



String 참조자료형

☞ String 객체의 '+' 연산 (Plus Operation)

1
- **CASE#1**. 문자열 + 문자열 → 문자열을 연결
- **CASE#2**. 문자열 + 기본자료형 또는 기본자료형 + 문자열 → 기본자료형을 문자열로 변환 + 문자열 연결

2 **CASE#1** 문자열 + 문자열 → 문자열을 연결

ex

3
```
String str1 = "안녕"+"하세요"+"!";
System.out.println(str1); //안녕하세요!

String str2 = "반갑";
str2+="습니다";
str2+="!";
System.out.println(str2); //반갑습니다!
System.out.println();
```

4

String str1 = "안녕" + "하세요" + "!";

"안녕하세요"

"안녕하세요!"

5

String 참조자료형

☞ String 객체의 '+' 연산 (Plus Operation)

- **CASE#1**. 문자열 + 문자열 → 문자열을 연결
- **CASE#2**. 문자열 + 기본자료형 또는 기본자료형 + 문자열 → 기본자료형을 문자열로 변환 + 문자열 연결

1 **CASE#2** 문자열 + 기본자료형 또는 기본자료형 + 문자열 → 기본자료형을 문자열로 변환 + 문자열 연결

2 ex

```
String str3 = "안녕" + 1;
String str4 = "안녕" + String.valueOf(1);
String str5 = "안녕" + "1";

System.out.println(str3);     //안녕1
System.out.println(str4);     //안녕1
System.out.println(str5);     //안녕1
System.out.println();
```

3 TIP

String.valueOf(기본자료형) → 기본자료형을 String으로 변환

4 ex

```
System.out.println(1+"안녕");       //1안녕
System.out.println(1+"안녕"+2);     //1안녕2
System.out.println("안녕"+1+2);     //안녕12
System.out.println(1+2+"안녕");     //3안녕
```

String 참조자료형

☞ String의 주요 메서드

1	문자열 길이	int	length()	문자열의 길이
2	문자열 검색	char	charAt(int index)	인덱스 위치에서의 문자
		int	indexOf(int ch) indexOf(int ch, int fromIndex) indexOf(String str) indexOf(String str, int fromIndex)	문자열에 포함된 문자 또는 문자열의 위치를 앞에서 부터 검색하였을 때 일치하는 인덱스 값(fromIndex는 검색의 시작위치)
		int	lastIndexOf(int ch) lastIndexOf(int ch, int fromIndex) lastIndexOf(String str) lastIndexOf(String str, int fromIndex)	문자열에 포함된 문자 또는 문자열의 위치를 뒤에서 부터 검색하였을 때 일치하는 인덱스 값(fromIndex는 검색의 시작위치)
3	문자열 변환 및 연결	**static** String	String.valueOf(boolean b) String.valueOf(char c) String.valueOf(int i) String.valueOf(long l) String.valueOf(float f) String.valueOf(double d)	boolean, char, int, long, float, double 값을 문자열로 변환하기 위한 정적 메서드
		String	concat(String str)	문자열의 연결 (String의 + 연산시 자동 동작) ex. "안녕"+"하세요 " = "안녕".concat("하세요")

String 참조자료형

☞ String의 주요 메서드

1	문자열 배열 변환	byte[]	getBytes() getBytes(Charset charset)	문자열을 byte[]로의 변환 (2바이트/3바이트 문자를 바이트로 변환시 문자셋(charset) 지정가능)
		char[]	toCharArray()	문자열을 char[]로 변환
2	문자열 수정	String	toLowerCase()	영문 문자를 모두 소문자로 변환
		String	toUpperCase()	영문 문자를 모두 대문자로 변환
		String	replace(char oldChar, char newChar)	oldChar 문자열을 newChar문자열로 대체한 문자열 생성
		String	substring(int beginIndex) substring(int beginIndex, int endIndex)	beginIndex부터 끝가지의 문자열 생성 beginIndex부터 endIndex-1 위치까지의 문자열 생성
		String[]	split(String regex) split(String regex, int limit)	regex를 기준으로 문자열을 분할하며 분할한 문자열 배열을 생성 (regex 구분기호는 '|' 기호로 여러 개 사용 가능) limit는 분할의 최대 개수
		String	trim()	문자열의 앞뒤 공백 제거
3	문자열 내용비교	boolean	equals()	문자열의 실제 **내용비교** (cf. ==는 메모리번지(stack) 비교)
		boolean	equalsIgnoreCase(String anotherString)	대소문자 구분없이 문자열의 실제 내용 비교

String 참조자료형

☞ String의 주요 메서드

- **문자열 길이 (length())**

1

ex **length()**

```
String str1 = "Hello Java!";
String str2 = "안녕하세요! 반갑습니다.";

System.out.println(str1.length());     //11
System.out.println(str2.length());     //13
System.out.println();
```

- **문자열 검색 (charAt(.), indexOf(.), lastIndexOf(.))**

2

ex **charAt(.)**

```
System.out.println(str1.charAt(1));     //e
System.out.println(str2.charAt(1));     //녕
System.out.println();
```

3

ex **indexOf(.), lastIndexOf(.)**

```
System.out.println(str1.indexOf('a',8));          //9
System.out.println(str1.lastIndexOf('a',8));      //7

System.out.println(str1.indexOf("Java"));         //6
System.out.println(str1.lastIndexOf("Java"));     //6

System.out.println(str2.indexOf("하세요"));       //2
System.out.println(str2.lastIndexOf("하세요"));   //2

System.out.println(str1.indexOf("Bye"));          //-1
System.out.println(str2.indexOf("고맙습니다."));//-1
System.out.println();
```

String 참조자료형

☞ String의 주요 메서드

- **문자열 변환 및 연결 (String.valueOf(.), concat(.))**

ex **String.valueOf()**

1
```
String str3 = String.valueOf(2.3);
String str4 = String.valueOf(false);
System.out.println(str3);          //2.3
System.out.println(str4);          //false
```

ex **concat()**

2
```
String str5 = str3.concat(str4);
System.out.println(str5);          //2.3false
```

ex **String.valueOf() + concat() = + Operation**

3
```
String str6 = "안녕"+3;
String str7 = "안녕".concat(String.valueOf(3));
System.out.println(str6);          //안녕3
System.out.println(str7);          //안녕3
```

- **문자열→byte[](getBytes()) / 문자열→ char[](toCharArray())**

ex **getBytes(), toCharArray()**

4
```
String str8 = "Hello Java!";
String str9 = "안녕하세요!";

byte[] array1 = str8.getBytes();
System.out.println(Arrays.toString(array1));
   //[72, 101, 108, 108, 111, 32, 74, 97, 118, 97, 33]
byte[] array2 = str9.getBytes();
System.out.println(Arrays.toString(array2));
 //[-66, -56, -77, -25, -57, -49, -68, -68, -65, -28, 33]

char[] array3 = str8.toCharArray();
System.out.println(Arrays.toString(array3));
                                   //[H, e, l, l, o,  , J, a, v, a, !]
char[] array4 = str9.toCharArray();
System.out.println(Arrays.toString(array4));
                                   //[안, 녕, 하, 세, 요, !]
```

String 참조자료형

☞ String의 주요 메서드

- **문자열 수정 (toLowerCase(), toUpperCase(), replace(), substring(), split(), trim())**

ex

1

```
//@toLowerCase(), toUpperCase()
String str1 = "Java Study";
System.out.println(str1.toLowerCase()); //java study
System.out.println(str1.toUpperCase()); //JAVA STUDY

//@replace(.,.)
System.out.println(str1.replace("Study", "공부")); //Java 공부

//@substring(.)
System.out.println(str1.substring(0,5)); //Java

//@split(.)
String[] strArray = "abc/def-ghi jkl".split("/|-| ");
System.out.println(Arrays.toString(strArray)); //[abc, def, ghi, jkl]

//@trim()
System.out.println("  abc  ".trim()); //abc
System.out.println();
```

- **문자열 내용비교 (equals(), equalsIgnoreCase(.))**

ex

2

```
String str2 = new String("Java");
String str3 = new String("Java");
String str4 = new String("java");

//@ == 메모리번지 비교 (stack 메모리 값 비교)
System.out.println(str2==str3);   //false
System.out.println(str3==str4);   //false
System.out.println(str2==str4);   //false
System.out.println();

//@ equals(.), equalsIgnoreCase(.) 내용 비교
System.out.println(str2.equals(str3));     //true
System.out.println(str3.equals(str4));     //false
System.out.println(str3.equalsIgnoreCase(str4));  //true
```

The End