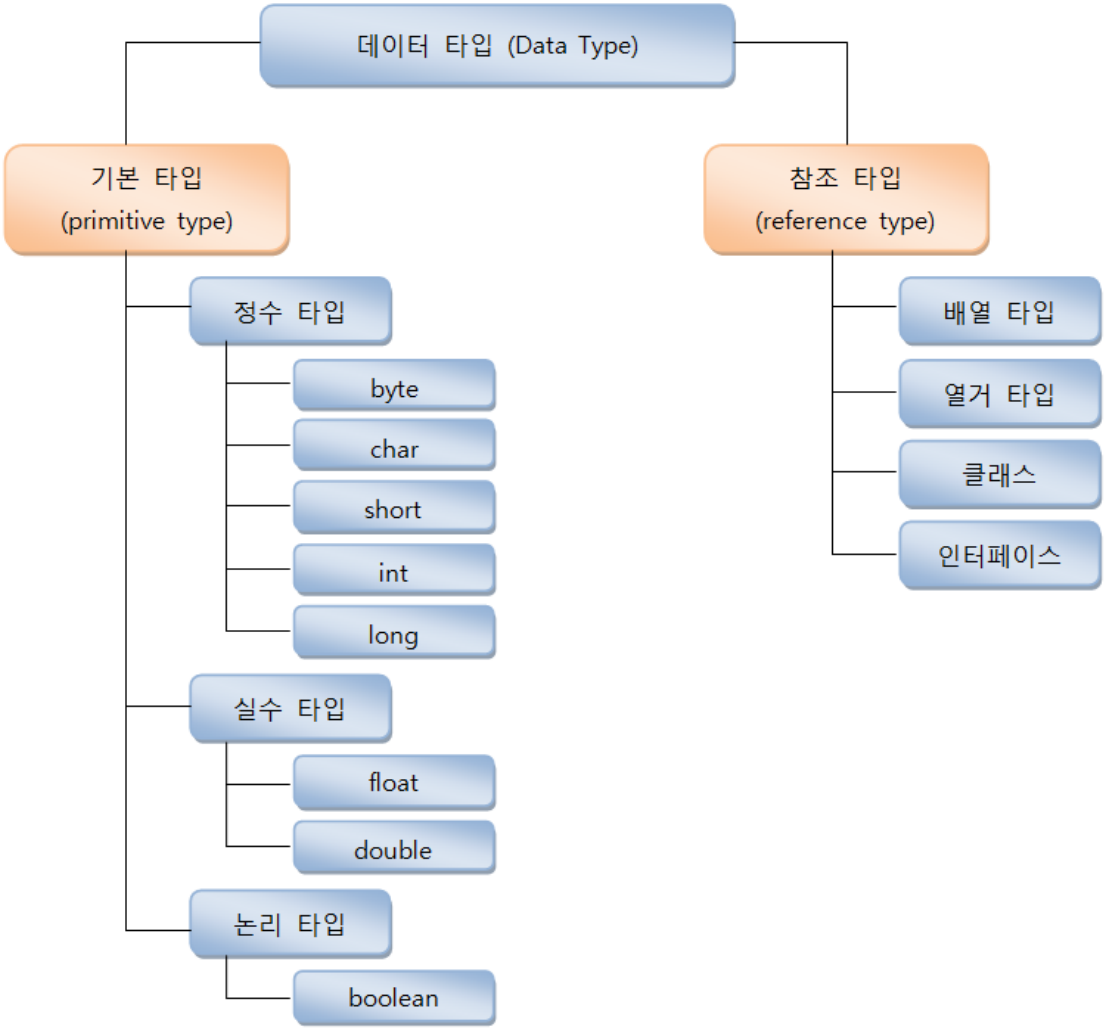
프로그래밍 언어 활용 강의안
(타입 변환과 배열)

1.데이터 타입 분류

데이터 타입 분류



1. 데이터 타입 변환

타입 변환

데이터 타입을 다른 타입으로 변환하는 것

byte ↔ int, int ↔ double

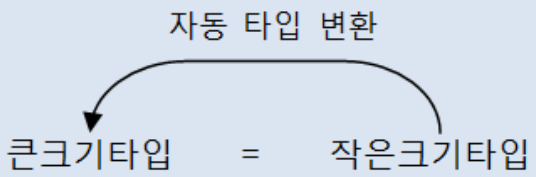
종류

자동(묵시적) 타입 변환: Promotion

강제(명시적) 타입 변환: Casting

자동 타입 변환

프로그램 실행 도중 작은 타입은 큰 타입으로 자동 타입 변환 가능

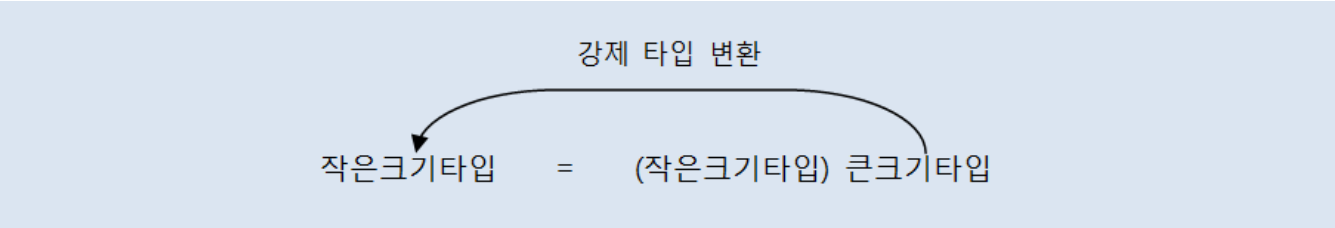


byte(1) < short(2) < int(4) < long(8) < float(4) < double(8)

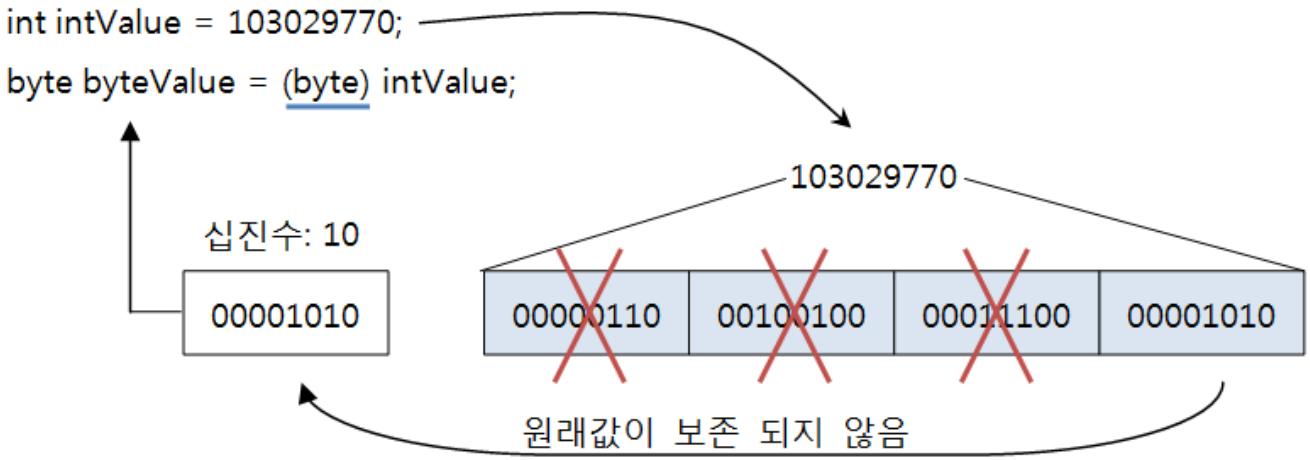
2. 데이터 타입 변환

강제 타입 변환

큰 타입을 작은 타입 단위로 쪼개기
끝의 한 부분만 작은 타입으로 강제적 변환



Ex) int 를 byte에 담기



3. 메모리 사용 영역

변수의 메모리 사용

기본 타입 변수 – 실제 값을 변수 안에 저장

참조 타입 변수 – 주소를 통해 객체 참조

[기본 타입 변수]

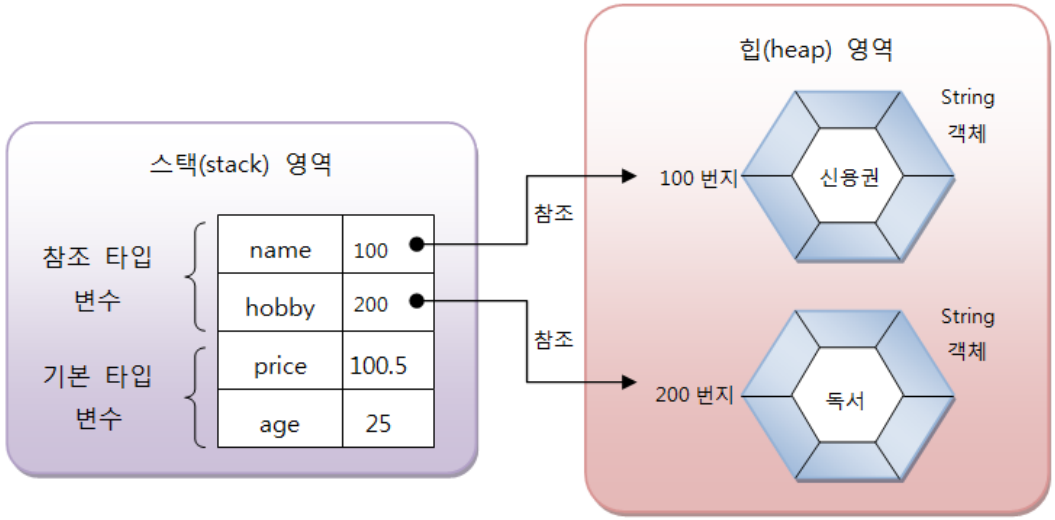
int age = 25;

double price = 100.5;

[참조 타입 변수]

String name = "신용권"

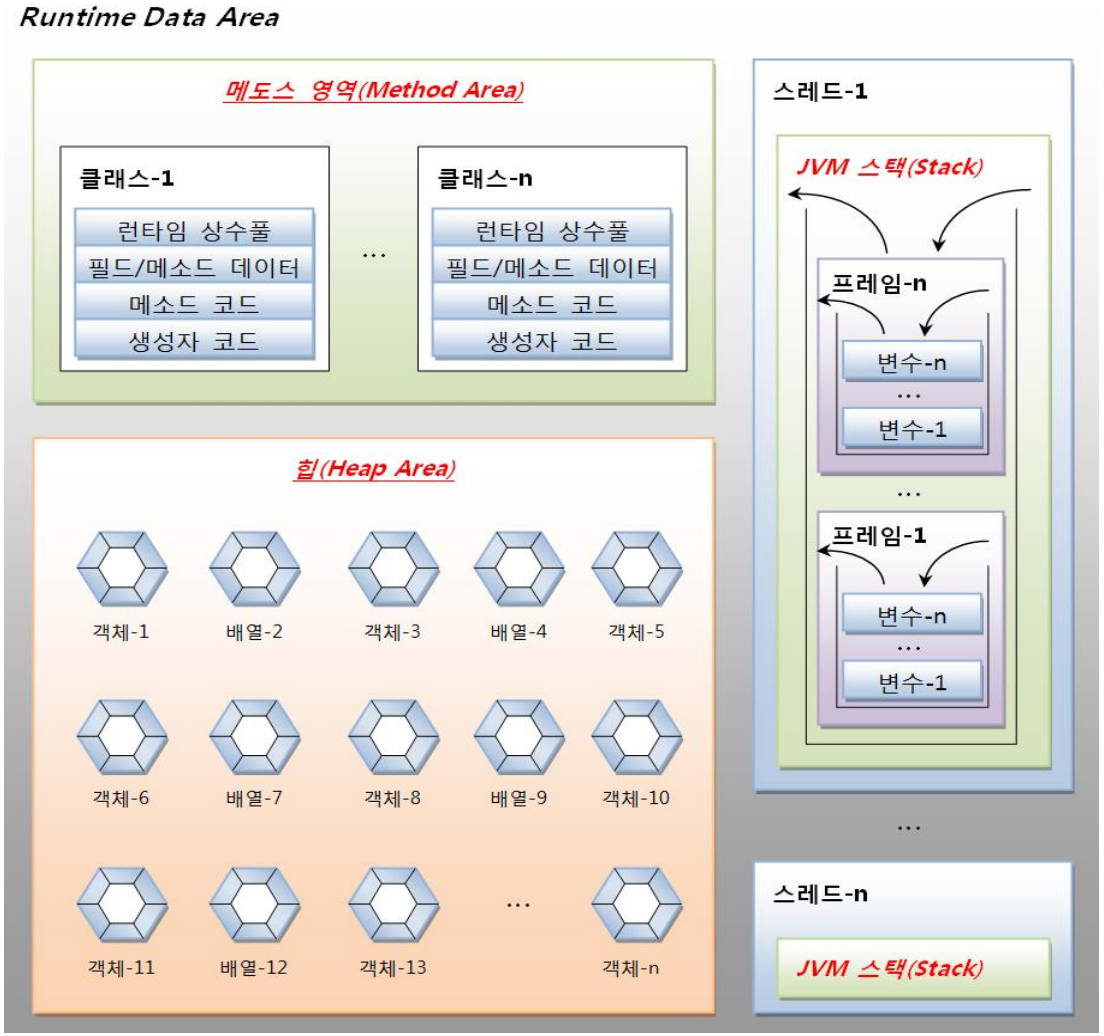
String hobby = "독서";



4. 메모리 사용 영역

JVM이 사용하는 메모리 영역

OS에서 할당 받은 메모리 영역(Runtime Data Area)을 세 영역으로 구분



5. 참조 변수의 == != 연산

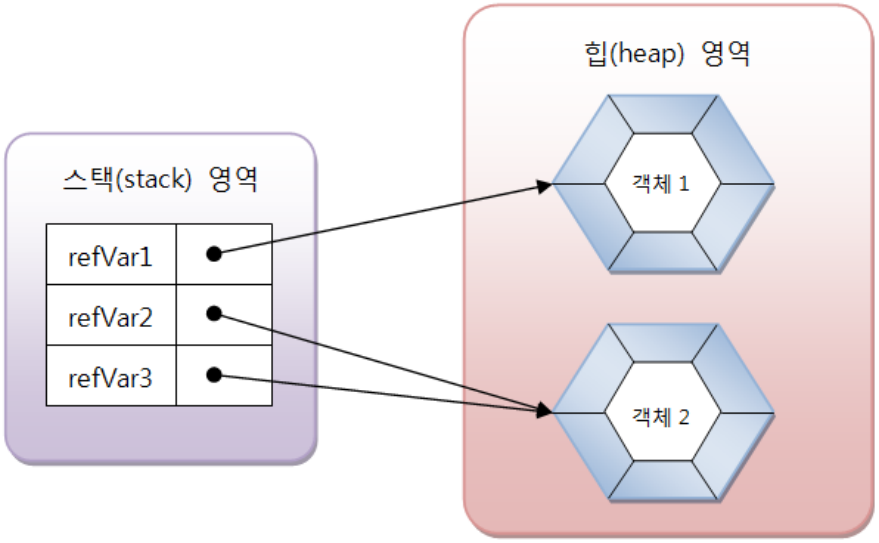
변수의 값이 같은지 다른지 비교

기본 타입: byte, char, short, int, long, float, double, boolean

의미 : 변수의 값이 같은지 다른지 조사

참조 타입: 배열, 열거, 클래스, 인터페이스

의미 : 동일한 객체를 참조하는지 다른 객체를 참조하는지 조사



refVar1 == refVar2 결과: false

refVar1 != refVar2 결과: true

refVar2 == refVar3 결과: true

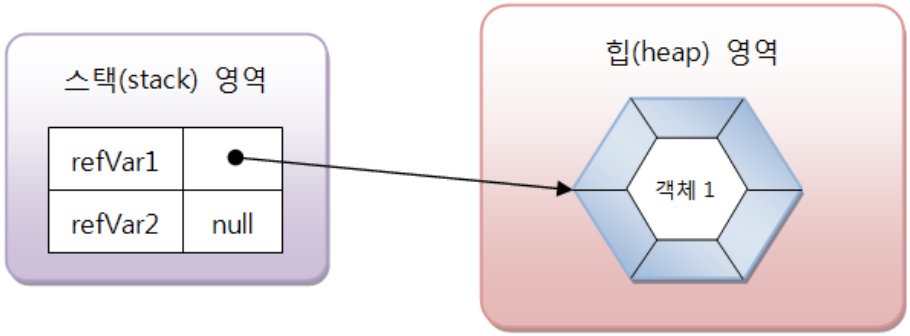
refVar2 != refVar3 결과: false

```
if( refVar2 == refVar3 ) { ... }
```

5. null 과 nullPointerException

null(널)

변수가 참조하는 객체가 없을 경우 초기값으로 사용 가능
참조 타입의 변수에만 저장가능
null로 초기화된 참조 변수는 스택 영역 생성



==, != 연산 가능

그림에서 refVar1 은 힙 영역의 객체를 참조하므로 연산의 결과는 다음과 같다.

```
refVar1 == null        결과:  false
refVar1 != null        결과:  true
```

refVar2 는 null 값을 가지므로 연산의 결과는 다음과 같다.

```
refVar2 == null        결과:  true
refVar2 != null        결과:  false
```

5. null 과 nullPointerException

NullPointerException의 의미

예외(Exception)

사용자의 잘못된 조작 이나 잘못된 코딩으로 인해 발생하는 프로그램 오류

NullPointerException

참조 변수가 null 값을 가지고 있을 때

객체의 필드나 메소드를 사용하려고 했을 때 발생

```
int[] intArray = null;
intArray[0] = 10;          //NullPointerException
```

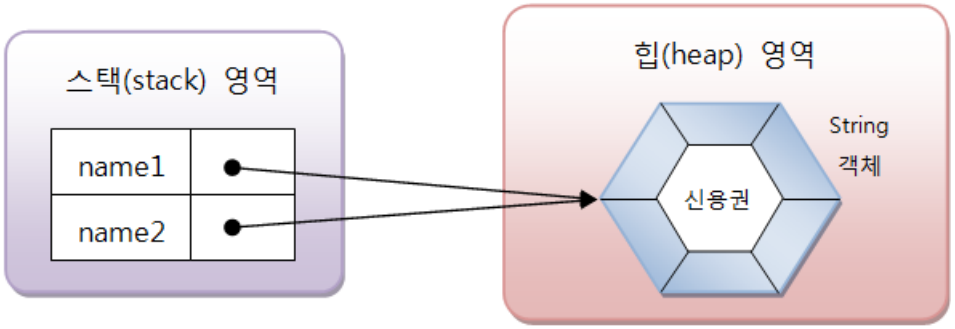
```
String str = null;
System.out.println("총 문자수: " + str.length());    //NullPointerException
```

6. String 타입

String 타입

문자열 리터럴 동일하다면 String 객체 공유

```
String name1 = "신용권";
String name2 = "신용권";
```

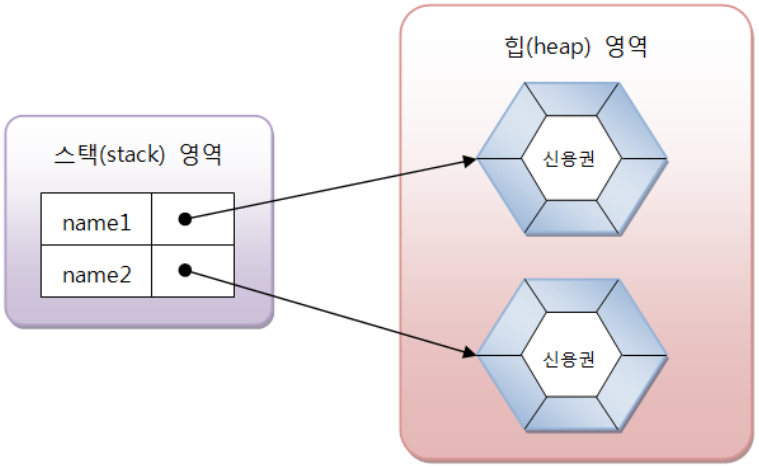


new 연산자를 이용한 String 객체 생성

힙 영역에 새로운 String 객체 생성

String 객체를 생성한 후 번지 리턴

```
String name1 = new String("신용권");
String name2 = new String("신용권");
```



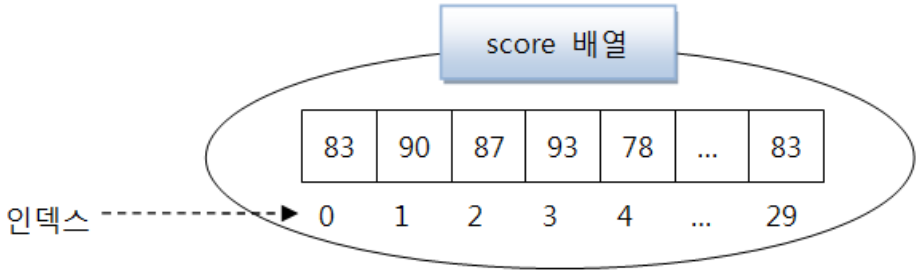
7. 배열 타입

배열이란?

같은 타입의 데이터를 연속된 공간에 저장하는 자료구조

각 데이터 저장 위치는 인덱스 부여해 접근

```
int score1= 83;
int score2 = 90;
int score3 = 87;
 :
int score30= 75;
```



항목 접근: 배열이름[인덱스] ex) score[0], score[3]

7. 배열 타입

배열의 장점

중복된 변수 선언 줄이기 위해 사용

반복문 이용해 요소들을 쉽게 처리

```
int sum = score1;
sum += score2;
sum += score3;
  :
sum += score30;
int avg = sum / 30;
```

```
int sum = 0;
for(int i=0; i<30; i++) {
   sum += score[i];
}
int avg = sum / 30;
```

7. 배열 타입

배열 선언

배열을 사용하기 위해 우선 배열 변수 선언

```
타입[ ] 변수;
```

```
타입 변수[ ];
```

```
int[] intArray;
double[] doubleArray;
String[] strArray;
```

```
int intArray[];
double doubleArray[];
String strArray[];
```

배열 변수는 참조 변수 - 배열 생성되기 전 null로 초기화 가능

```
타입[] 변수 = null;
```

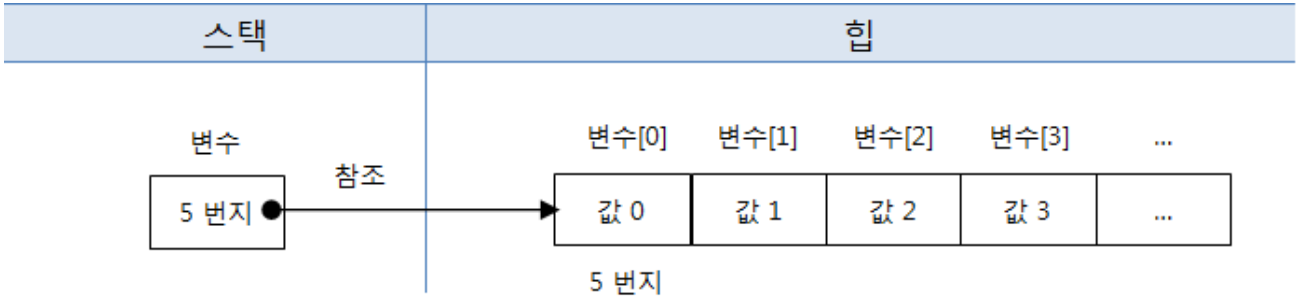
배열 변수가 null 값을 가진 상태에서 항목에 접근 불가
변수[인덱스]" 못함
NullPointerException 발생

7. 배열 타입

값 목록으로 배열 생성하는 방법

변수 선언과 동시에 값 목록 대입

```
데이터타입[] 변수 = { 값 0, 값 1, 값 2, 값 3, ... };
```



변수 선언 후 값 목록 대입

```
데이터타입[] 변수;
변수 = new 타입[] { 값 0, 값 1, 값 2, 값 3, ... };
```

7. 배열 타입

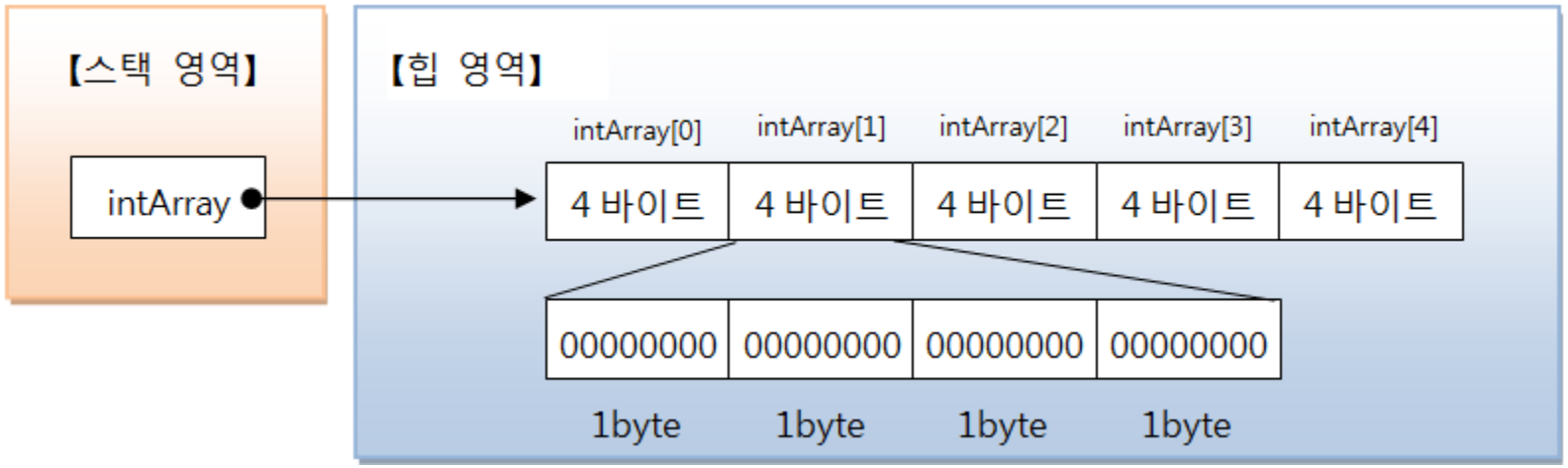
new 연산자로 배열 생성

- 배열 생성시 값 목록을 가지고 있지 않음
- 향후 값들을 저장할 배열을 미리 생성하고 싶을 경우

```
타입[] 변수 = new 타입[길이];
```

```
타입[] 변수 = null;
변수 = new 타입[길이];
```

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



7. 배열 타입

타입 별 항목의 기본값

분류	데이터 타입	초기값
기본 타입 (정수)	byte[]	0
	char[]	'₩u0000'
	short[]	0
	int[]	0
	long[]	0L
기본 타입 (실수)	float[]	0.0F
	double[]	0.0
기본 타입 (논리)	boolean[]	false
참조 타입	클래스[]	null
	인터페이스[]	null

7. 배열 타입

배열의 길이

배열에 저장할 수 있는 전체 항목 수
코드에서 배열의 길이 얻는 방법

```
배열변수.length;
```

```
int[] intArray = { 10, 20, 30 };
int num = intArray.length;
```

배열의 길이는 읽기 전용

```
intArray.length = 10;   //잘못된 코드
```

배열의 길이는 for문의 조건식에서 주로 사용

```
int[] scores = { 83, 90, 87 };

int sum = 0;                                3
for(int i=0; i<scores.length; i++) {
   sum += scores[i];
}
System.out.println("총합 : " + sum);
```

배열 복사

- 배열은 한 번 생성하면 크기 변경 불가
- 더 많은 저장 공간이 필요하다면 보다 큰 배열을 새로 만들고 이전 배열로부터 항목 값들을 복사

배열 복사 방법

- for문 이용
- System.arrayCopy() 메소드 이용
- Arrays 클래스 이용

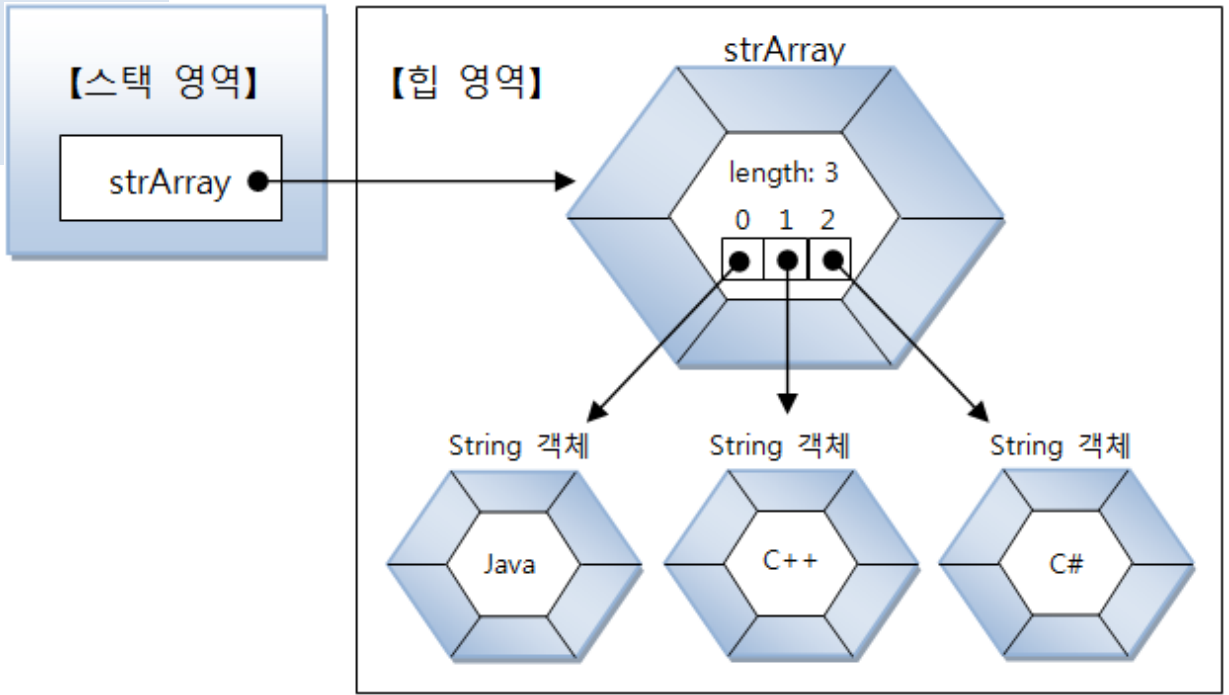
7. 배열 타입

객체를 참조하는 배열

기본 타입(byte, char, short, int, long, float, double, boolean) 배열
각 항목에 직접 값을 가지고 있음

참조 타입(클래스, 인터페이스) 배열 - 각 항목에 객체의 번지 가짐

```
String[] strArray = new String[3];
strArray[0] = "Java";
strArray[1] = "C++";
strArray[2] = "C#";
```



7. 배열 타입

다차원 배열

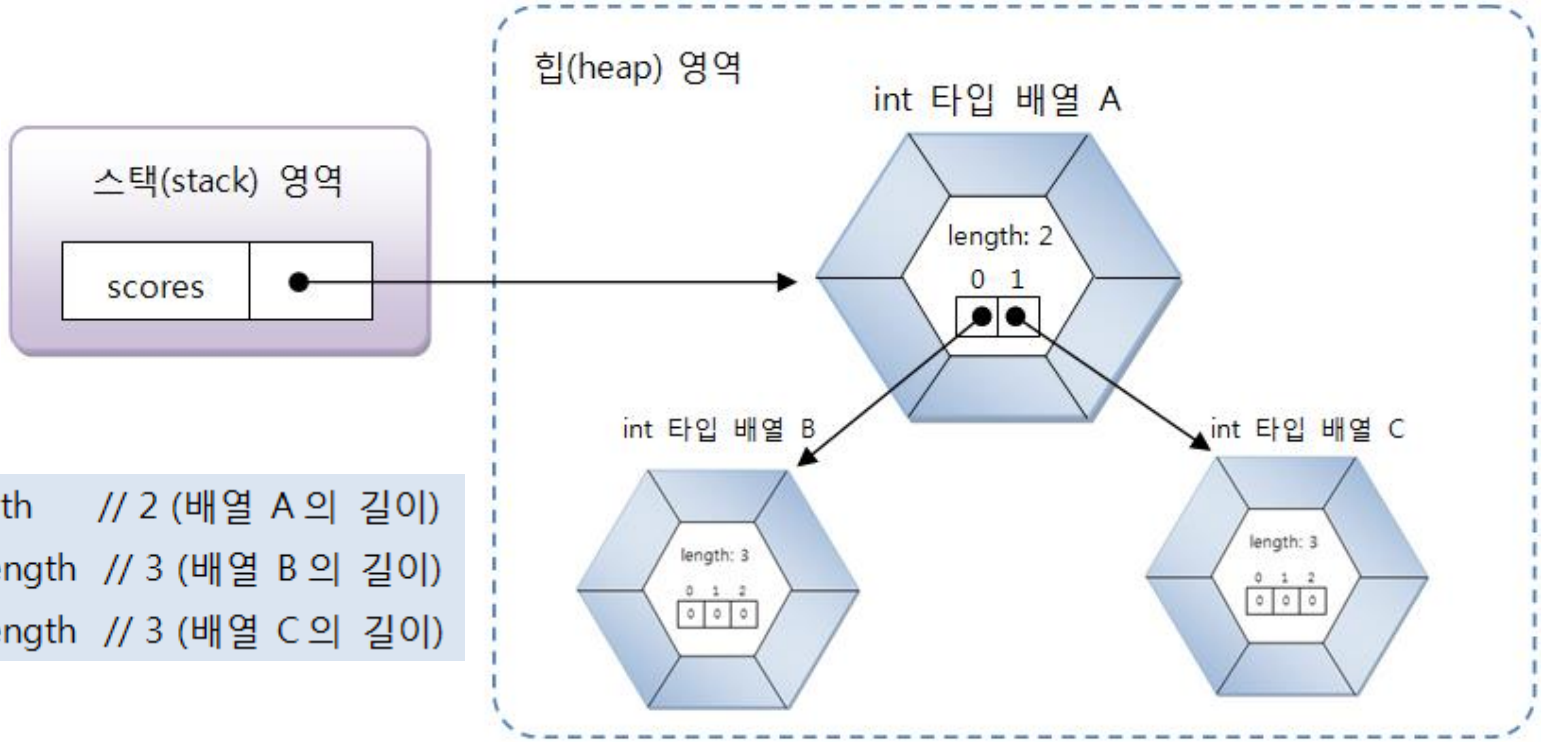
2차원 배열 이상의 배열

수학의 행렬과 같은 자료 구조

【2 x 3 행렬의 구조】

행 \ 열	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)

자바는 1차원 배열을 이용해 2차원 배열 구현



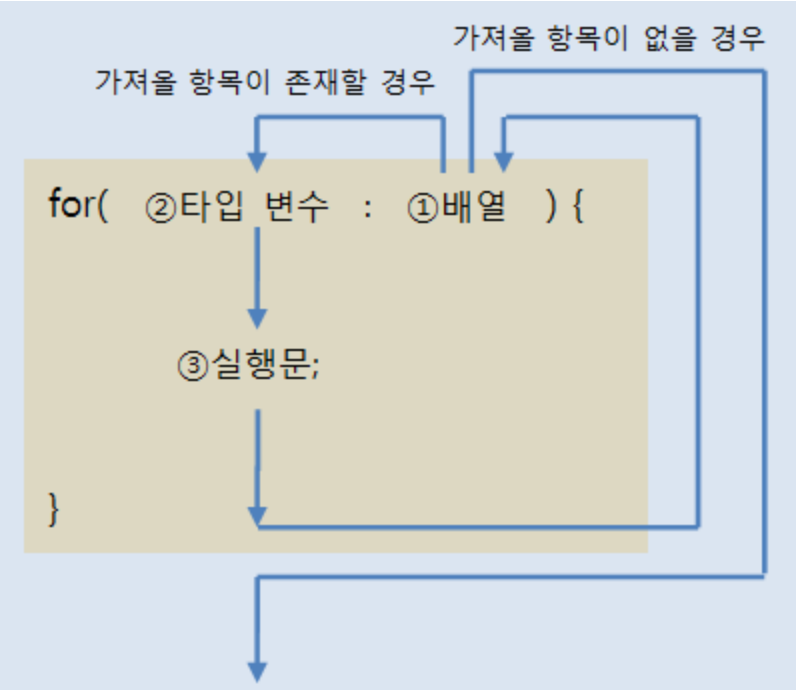

```
scores.length     // 2 (배열 A의 길이)
scores[0].length  // 3 (배열 B의 길이)
scores[1].length  // 3 (배열 C의 길이)
```

8. 향상된 for문

향상된 for 문(개선된 for문, for each 문)

배열 및 컬렉션의 항목 요소를 순차적으로 처리

인덱스 이용하지 않고 바로 항목 요소 반복




```
int[] scores = { 95, 71, 84, 93, 87 };

int sum = 0;
for (int score : scores) {
   sum = sum + score;
}
```