문자열, 배열, 열거타입

문자열

문자열의 선언과 생성

```
String 변수;      // String 타입의 변수 선언
변수 = "문자열";  // String 타입의 변수에 문자열 대입
```

```
String s1 = "안녕, 자바!";     // String 타입의 변수 선언과 초기화
String s2 = "안녕, 자바!";     // String 타입의 변수 선언과 초기화
```

문자열 리터럴이다.

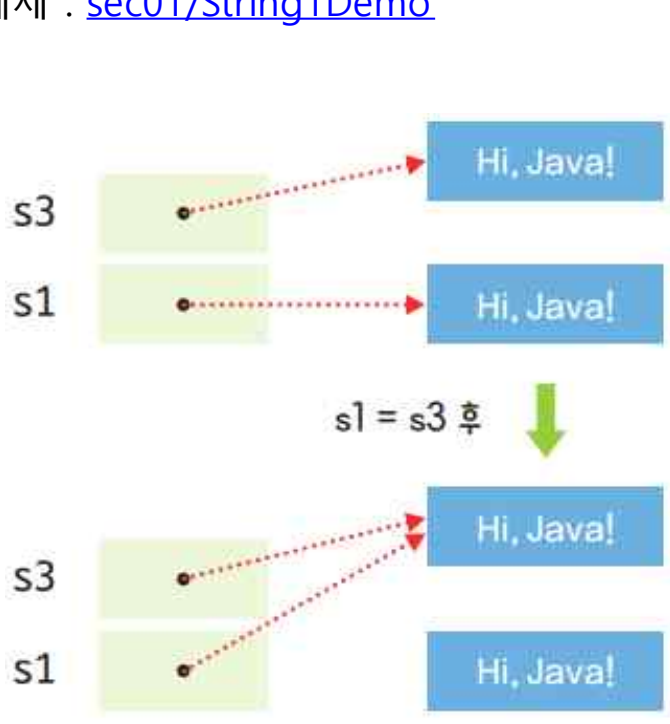
- 문자열 리터럴은 내부적으로 new String()을 호출해 생성한 객체이다.
- 따라서 s1은 new String("안녕, 자바!")를 호출해서 생성한 객체를 가리킨다.
- 그러나 내용이 같은 문자열 리터럴이라면 더 이상 새로운 String 객체를 생성하지 않은 채 기존 리터럴을 공유. 따라서 s1과 s2는 동일한 String 객체를 가리킨다.

문자열

문자열의 비교

- ==와 != 연산자는 두 문자열의 내용을 비교하는 것이 아니라 동일한 객체인지 검사

- 예제 : sec01/String1Demo




```
s1 == s2 -> true
s1 == s3 -> false
s3 == s4 -> false
s1 == s3 -> true
```

참조 변수가 없으므로 사용할 수 없는 객체가 된다.
따라서 후에 가비지 컬렉터로 자동 수거된다.

문자열

문자열의 비교

- String 클래스에서 제공하는 문자열 비교 메서드

메서드	설명
int compareTo(String s)	문자열을 사전 순으로 비교해 정수 값을 반환한다.
int compareToIgnoreCase(String s)	대 · 소문자를 무시하고, 문자열을 사전 순으로 비교한다.
boolean equals(String s)	주어진 문자열 s와 현재 문자열을 비교한 후 true/false를 반환한다.
boolean equalsIgnoreCase(String s)	주어진 문자열 s와 현재 문자열을 대 · 소문자 구분 없이 비교한 후 true/false를 반환한다.

- 예제 : sec01/String2Demo

```
true
false
false
true
7
0
-39
0
```

문자열

문자열 조작

- String 클래스에서 제공하는 메서드

메서드	설명
char charAt(int index)	index가 지정한 문자를 반환
String concat(String s)	주어진 문자열 s를 현재 문자열 뒤에 연결
boolean contains(String s)	문자열 s를 포함하는지 조사
boolean endsWith(String s)	끝나는 문자열이 s인지 조사
int indexOf(String s)	문자열 s가 나타난 위치를 반환
boolean isBlank( )	길이가 0 혹은 공백 있으면 true 반환(자바 11부터)
boolean isEmpty( )	길이가 0이면 true 반환
int length( )	길이를 반환
String repeat(int c)	c번 반복한 문자열을 반환(자바 11부터)
boolean startsWith(String s)	시작하는 문자열이 s인지 조사
String substring(int index)	index부터 시작하는 문자열의 일부를 반환
String toLowerCase( )	모두 소문자로 변환
String toUpperCase( )	모두 대문자로 변환
String trim( )	앞뒤에 있는 공백을 제거한 후 반환

문자열

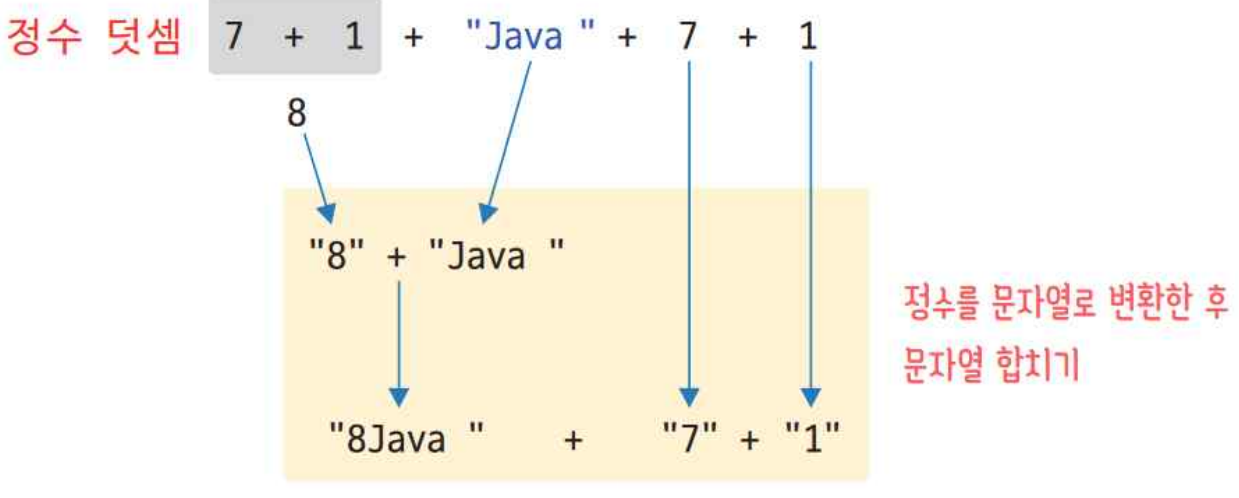
문자열의 조작

- 예제 : sec01/String3Demo

```
문자열 길이(s1) : 3
i
Hi, Java Java!
hi, java!
Java!
false
true
true
true
*-**-**-**-**-**-**-**-**-*
2
```

- 예제 : sec01/String4Demo

```
Java 7
Java 7
8Java 71
```



문자열

String 클래스에서 제공하는 유용한 정적 메서드

정적 메서드	설명
String format( )	주어진 포맷에 맞춘 문자열을 반환
String join( )	주어진 구분자와 연결한 문자열을 반환(자바 8부터)
String valueOf( )	각종 기초 타입이나 객체를 문자열로 반환

- 예제 : sec01/String5Demo

```
JDK 14
apple, banana, cherry, durian
3.14
```

문자열

텍스트 블록

- 자바 15부터 추가된 기능
- 멀티 라인의 문자열을 이스케이프 시퀀스(escape sequence) 없이 허용
- 소스 코드 작성을 편리하게 하고 코드의 가독성을 제고
- """ ~ """ 코드 사이에 있는 문자열을 이스케이프 문자나 스트링 조합 연산 없이 String 객체로 인식
- 블록을 시작하는 """ 뒤에는 문자열이 바로 나오면 컴파일 에러가 발생. 기존 문자열 리터럴과 구분하기 위하여 """ 후에 한 라인을 띄운 후 문자열을 작성해야 텍스트 블록으로 인식

텍스트 블록

```
String html = """
      <html>
          <body>
              <p>Hello, world</p>
          </body>
      </html>
      """;
```

문자열 조합

```
String html1 =
      "<html>₩n" +
      " <body>₩n" +
      " <p>Hello, world</p>₩n" +
      " </body>₩n" +
      "</html>₩n";
```

문자열

텍스트 블록

- 이중 인용부호(")가 필요할 경우 기존 방식의 ₩ " 과 달리 ₩없이 텍스트 블록에서는 바로 인식
- 개행 문자인 ₩n 없이 텍스트 블록에서는 바로 엔터 값을 인식
- 텍스트 블록에서 유일하게 ₩는 이스케이프 시퀀스로 인식되기 때문에 해당 값을 반영하길 원하면 기존처럼 ₩₩로 작성
- 텍스트 블록도 String 객체이기 때문에 String 클래스가 제공하는 모든 연산 사용 가능
- 텍스트 블록에서 들여쓰기 규칙은 블록을 종료하는 """의 위치에 의해 결정

```
String hi = """
      안녕
           하세요
      """;
```

➡

```
안녕
     하세요
```

```
String hi = """
      안녕
           하세요
""";
```

➡

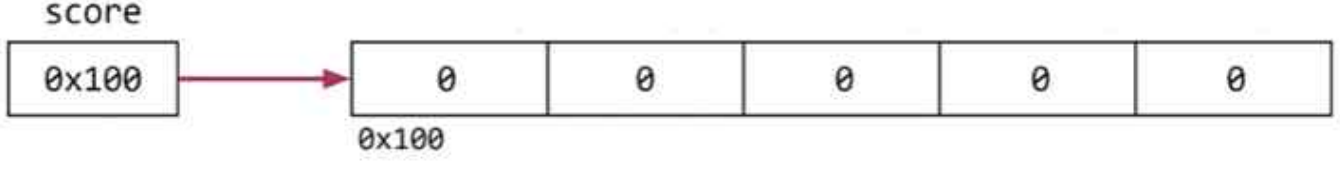
```
      안녕
           하세요
```

- 예제 : sec01/TextBlockDemo

배열 기초

배열이란


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



배열 기초

배열의 필요성

```java
int score1 = 100;
int score2 = 90;
int score3 = 50;
int score4 = 95;
int score5 = 85;

int sum = score1;
sum += score2;
sum += score3;
sum += score4;
sum += score5;
double average = sum / 5;
```

(a) 배열을 사용하지 않을 때

```java
int[] scores = { 100, 90, 50, 95, 85 };
int sum = 0;

for (int i = 0; i < 5; i++)
  sum += scores[i];
double average = sum / 5;
```

(b) 배열을 사용할 때

배열 기초

배열의 선언과 생성

- **배열의 선언** : 실제는 배열 변수의 선언, 배열을 다루기 위한 참조변수의 선언

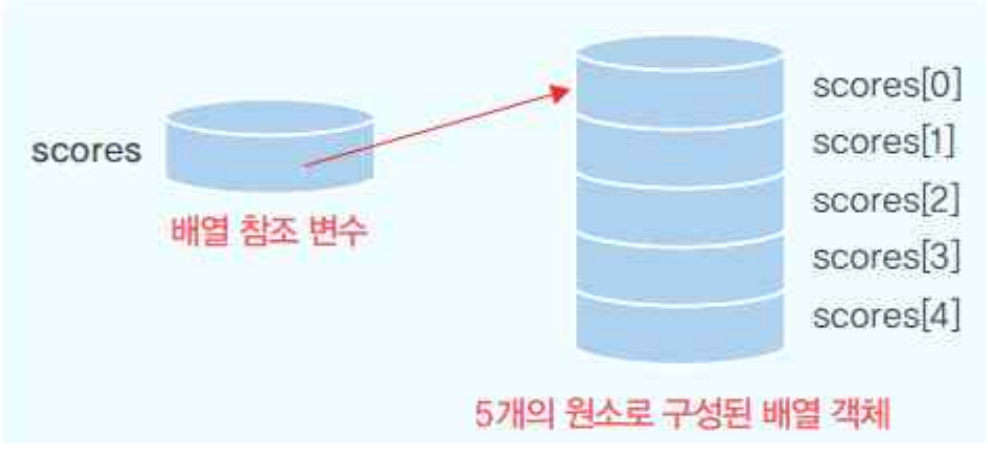
```
int[] scores;        혹은        int scores[];
```

```
int scores[5];
```

- **배열의 선언과 생성** : 실제는 배열 변수의 선언과 초기화, 배열의 생성은 저장공간이 만들어짐을 의미

배열의 크기

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



1) int 값을 저장할 수 있는 5개의 저장공간을 만들어준다.
2) 5개의 저장공간의 주소(예>0x100)를 참조변수 score 에 대입한다.

배열 기초

- 배열의 초기화 : 배열의 각 요소에 처음으로 값을 저장하는 것
- 배열은 기본적으로 기본값으로 자동 초기화가 된다.(int의 경우 0으로 자동 저장)
- 초기화하는 값에 규칙이 있다면 반복문으로 모든 요소에 특정값을 대입시켜 초기화할 수 있지만 { } 를 사용해 초기화한다.

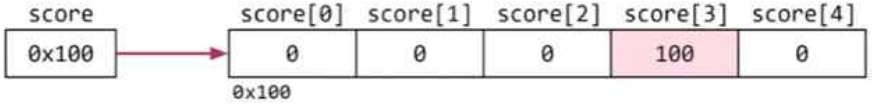
```
// 방법 ❶
int[] scores = { 100, 90, 50, 95, 85 };

// 방법 ❷
int[] scores = new int[] { 100, 90, 50, 95, 85 };

// 방법 ❸
int[] scores;
scores = new int[] { 100, 90, 50, 95, 85 };

// 방법 ❹
int[] scores;
scores = { 100, 90, 50, 95, 85 };
```

배열 기초

배열 원소의 접근

- *배열이름*[인덱스]; 인덱스는 각 요소(저장공간)에 자동으로 부여되는 일련번호로 0부터 배열의 길이-1 까지가 인덱스의 범위이다.
  - scores[3] = 100;

```
score[3] = 100;          // 배열 score의 4번째 요소에 100을 저장한다.
```



배열의 크기

- 배열이 생성될 때 배열의 크기가 결정 (즉, 한번 생성되면 그 길이를 바꿀 수 없다.)
- 배열의 length 필드가 배열의 크기를 나타냄. 예를 들어, scores가 가리키는 배열의 크기는 scores.length(int형 상수, 변경불가)
  - 배열의 크기를 변경 못하는 이유는 ?
  - 배열의 크기가 부족해서 늘리려면 ?

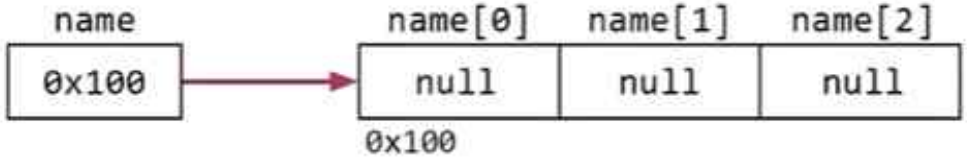
배열 기초

- 배열의 출력
- int[] arr = {1,2,3,4,5};
- System.out.println(arr); 의 출력 결과
  - [I@14318bb 와 같은 형식의 문자열이출력되는데 그 이유는?
- char[] cArr = {'a', 'b', 'c'};
- System.out.println(cArr); 의 출력 결과는?

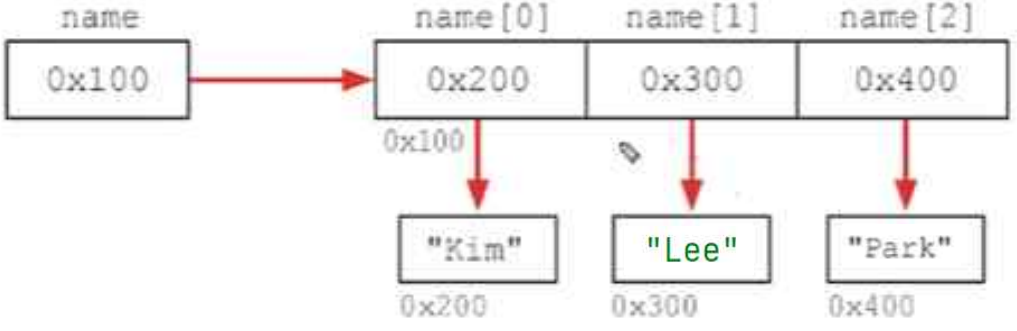
- 배열의 모든 요소를 출력하려면 for, foreach문을 사용하여 배열의 요소를 순서대로 하나씩 출력할 수 있다.
- System.out.println(Arrays.toString(arr)); 를 사용하면 배열의 요소를 출력할 수 있다.
  - [1,2,3,4,5]

문자열 배열

- **String[] name = new String[3];** // 3개의 문자열을 담을 수 있는 배열을 생성



자료형	기본값
boolean	false
char	'\u0000'
byte, short, int	0
long	0L
float	0.0f
double	0.0d 또는 0.0
참조형	null



참고> 자바의 정석

- String[] name = {"Kim", "Lee", "Park"}; 과 같은 방식으로 선언과 초기화 가능.

배열 기초

다차원 배열

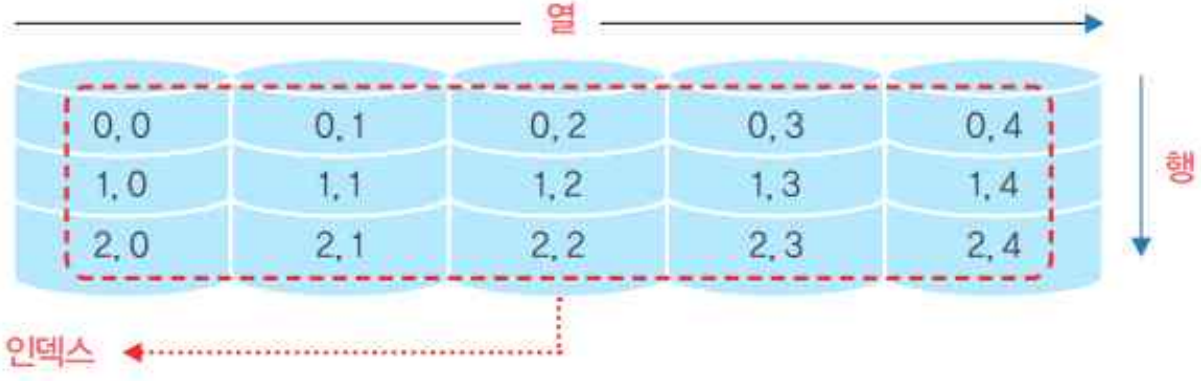
- 배열의 배열
- 예를 들어 학생 3명의 5과목 성적을 처리하는 정수 타입 2차원 배열(3행 × 5열)인 scores를 선언하고 생성해 보자.

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

2개의 대괄호는 2차원 배열을 표시

행의 개수

열의 개수



첫 번째 인덱스는 행 번호이며, 두 번째 인덱스는 열 번호이다.

배열 기초

다차원 배열

- 선언과 초기화

```
int[][] scores = {{100, 90, 50, 95, 85}, {70, 60, 82, 75, 40}, {90, 80, 70, 60, 50}};
```

첫 번째 행의 원소이다. / 두 번째 행의 원소이다. / 세 번째 행의 원소이다.

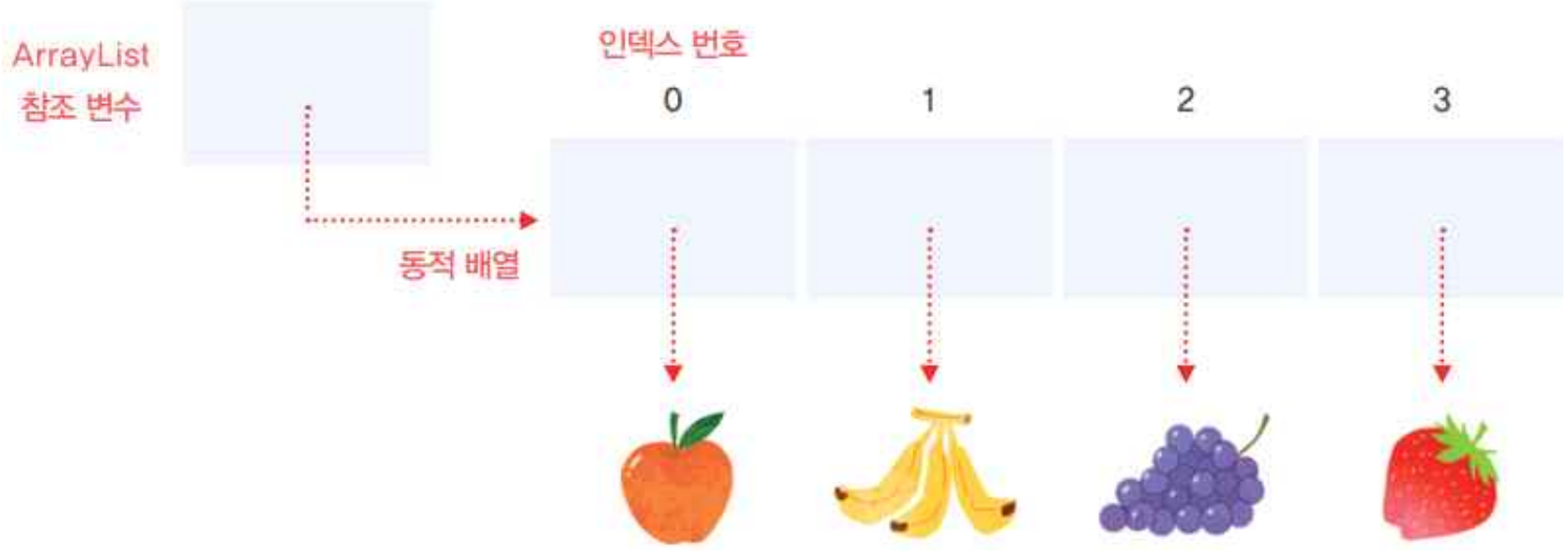
- 예제 : sec02/Array2Demo

```
1차년도 평균 이자율 = 3.13%
2차년도 평균 이자율 = 2.75%
3차년도 평균 이자율 = 2.63%
3년간 평균 이자율 = 11.33%
```

배열 기초

동적 배열

- 처리할 데이터의 개수가 고정된 경우가 아니라면 정적 배열은 자원을 낭비하거나 프로그램을 다시 컴파일
- 자바는 크기가 유동적인 배열을 지원하기 위하여 ArrayList 클래스를 제공



배열 기초

동적 배열

- ArrayList 객체 생성

```
ArrayList<참조타입> 참조변수 = new ArrayList<>();
```

기초 타입의 동적 배열이라면 Integer, Long, Short, Float, Double, Charater, Boolean 등을 사용한다.

- ArrayList 원소 접근

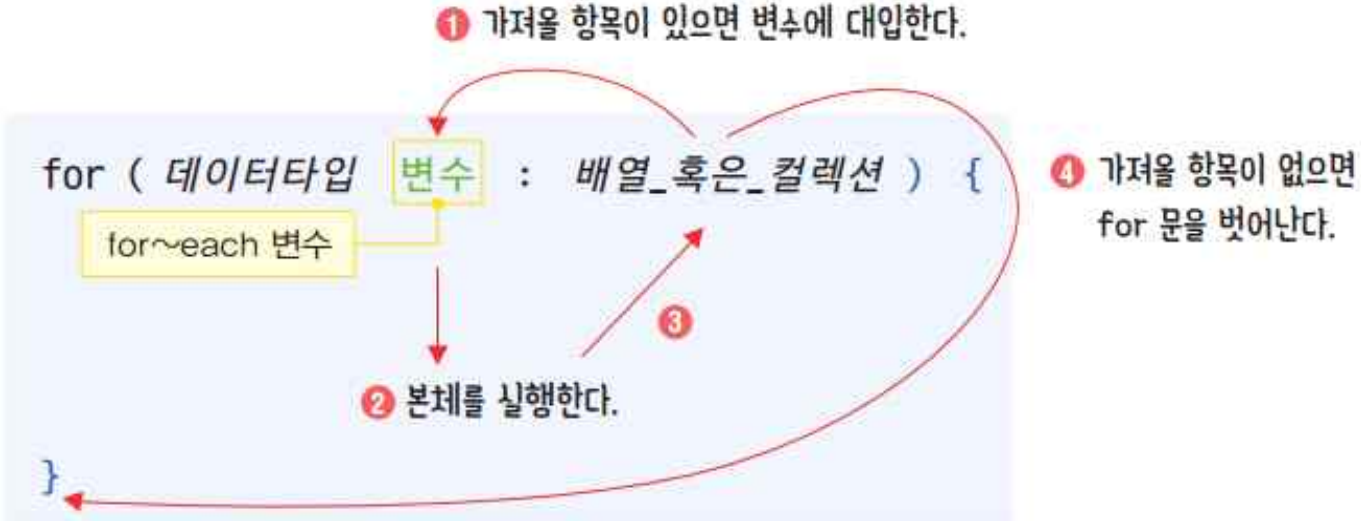
참조변수.add(데이터)	데이터를 동적 배열에 원소로 추가
참조변수.remove(인덱스번호)	동적 배열에서 인덱스 번호의 원소를 제거
참조변수.get(인덱스번호)	동적 배열에서 인덱스 번호의 원소를 가져오기
참조변수.size()	동적 배열에 포함된 원소 개수

- 예제 : sec02/ArrayListDemo

배열 응용

배열을 위한 반복문

- for~each 반복문 : JDK 5부터 도입된 것으로 for 문을 개선한 방식. 특정 원소를 나타내기 위한 인덱스를 사용하지 않는다.



예제 : [sec02/Array1Demo]

배열 응용

최대값과 최소값 찾기

- 배열의 요소 중에서 제일 큰 값과 제일 작은 값을 찾기
  - 배열의 첫번째 값으로 최대값과 최소값을 초기화한다.
  - 배열의 두번째 요소부터 읽기 위해 for 반복문의 변수 i 값을 1로 초기화한다.
  - 배열의 각 요소와 최소값을 비교하여 더 작은 값을 찾으면 그 요소로 최소값을 갱신한다.
  - 배열의 각 요소와 최대값을 비교하여 더 큰 값을 찾으면 그 요소로 최대값을 갱신한다.

shuffle 과 로또번호 생성

배열 응용

- 배열을 위한 반복문

```
20  public class ForEachDemo {
21      public static void main(String[] args) {
22          int[] one2five = {0, 1, 2, 3, 4};
23          int sum = 0;
24
25          for (int x = 0; x < one2five.length; x++)
26              one2five[x]++;
27
28          for (int x : one2five)
29              sum += x;
30
31          System.out.println("평균 = " + sum / 5.0);
32      }
33  }
```

final int x로 변경하면 x++에서 오류가 발생한다.

for 문은 특정 원소에 접근할 수 있지만 for-each 문은 할 수 없다.

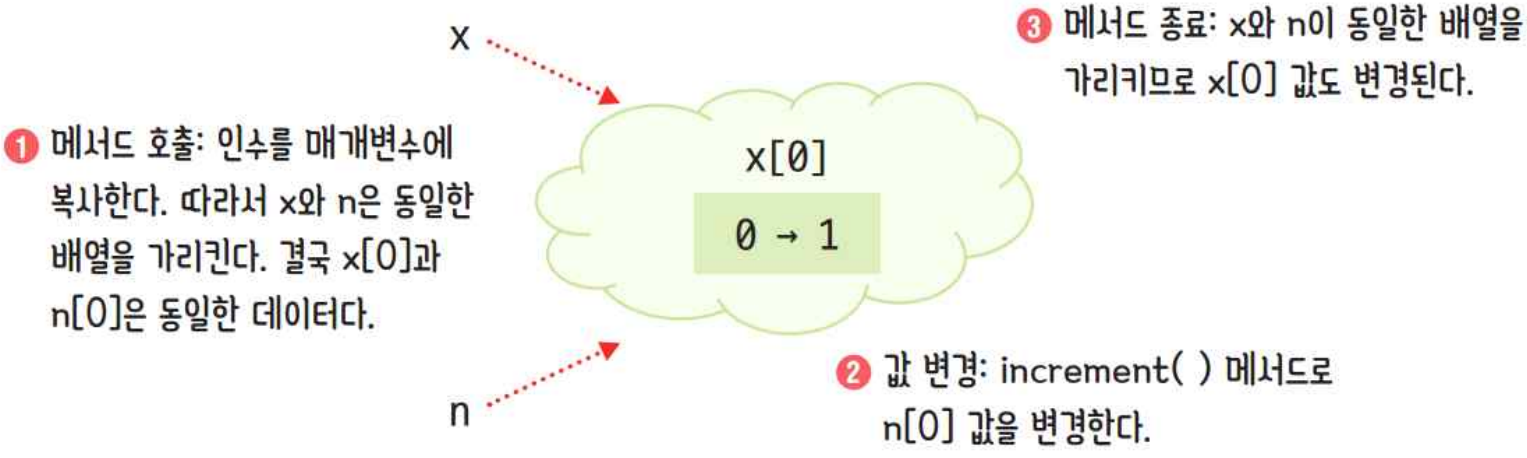
final이 없어도 변수 x는 final int 타입이다.

배열 응용

메서드의 인수로 배열 전달

- 예제 : sec03/IncrementDemo

```
호출 전의 x[0] = 0
increment() 메서드 안에서 n[0] = 0 ---> n[0] = 1
호출 후의 x[0] = 1
```



배열 응용

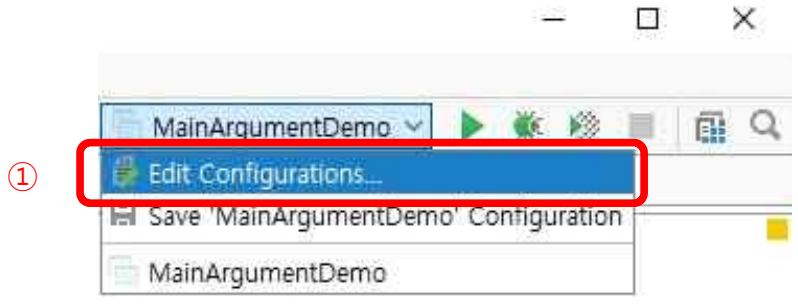
- **메인 메서드의 매개변수 전달**
  - 명령창에서의 실행 명령
  - 예제 : sec03/MainArgumentDemo

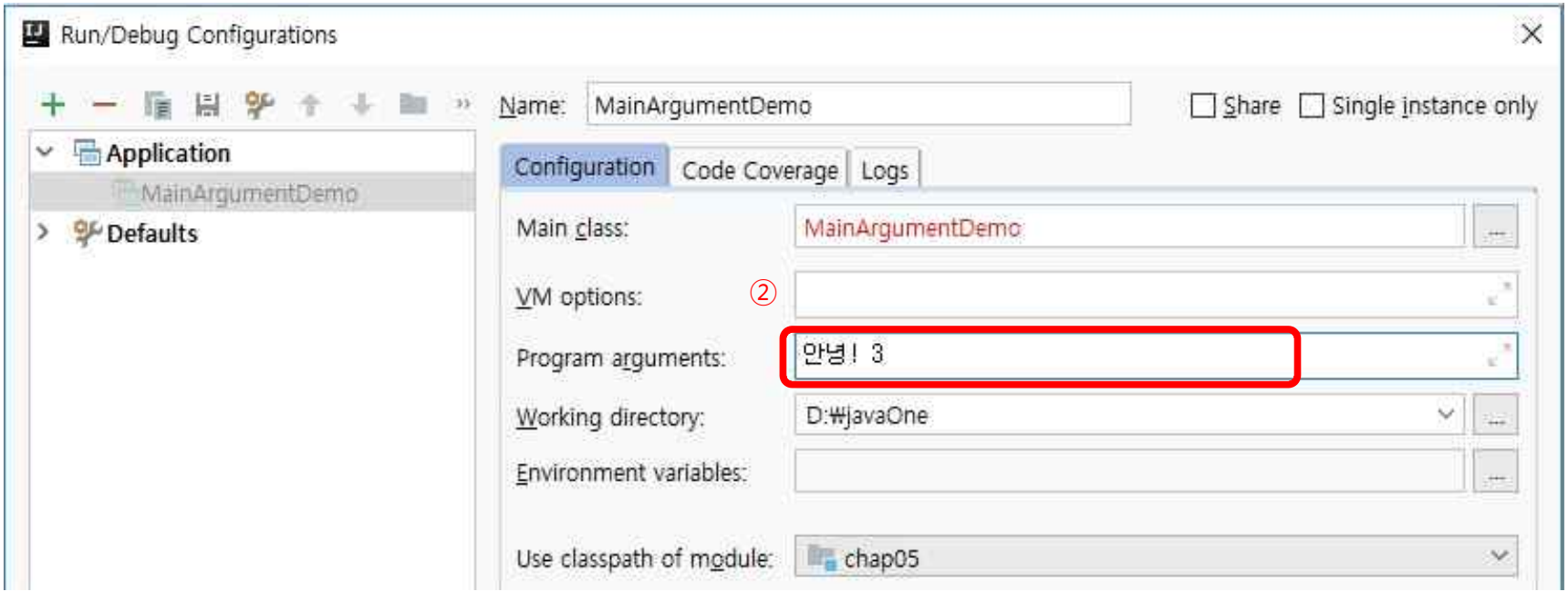
- **> java MainArgumentDemo “안녕!” 3**
- **또는 edit configuration 에서 매개변수 입력해서 실행**

배열 응용

■ 메인 메서드의 매개변수 전달

- 인텔리 J 아이디어에서 매개변수 제공

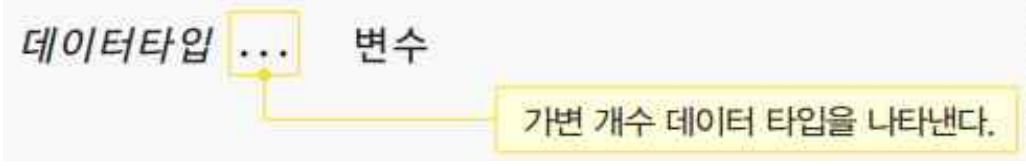




배열 응용

가변 개수 인수

- JDK 5부터는 메서드에도 데이터 타입이 같은 가변 개수(variable length)의 인수를 전달 가능



- 한 개의 가변 개수 매개변수만 사용 가능하며 가변 개수 매개변수는 마지막에 위치
- 가변 개수 인수를 가진 메서드를 호출하면 내부적으로 배열을 생성하여 처리

- 예제 : sec03/VarArgsDemo

배열 응용

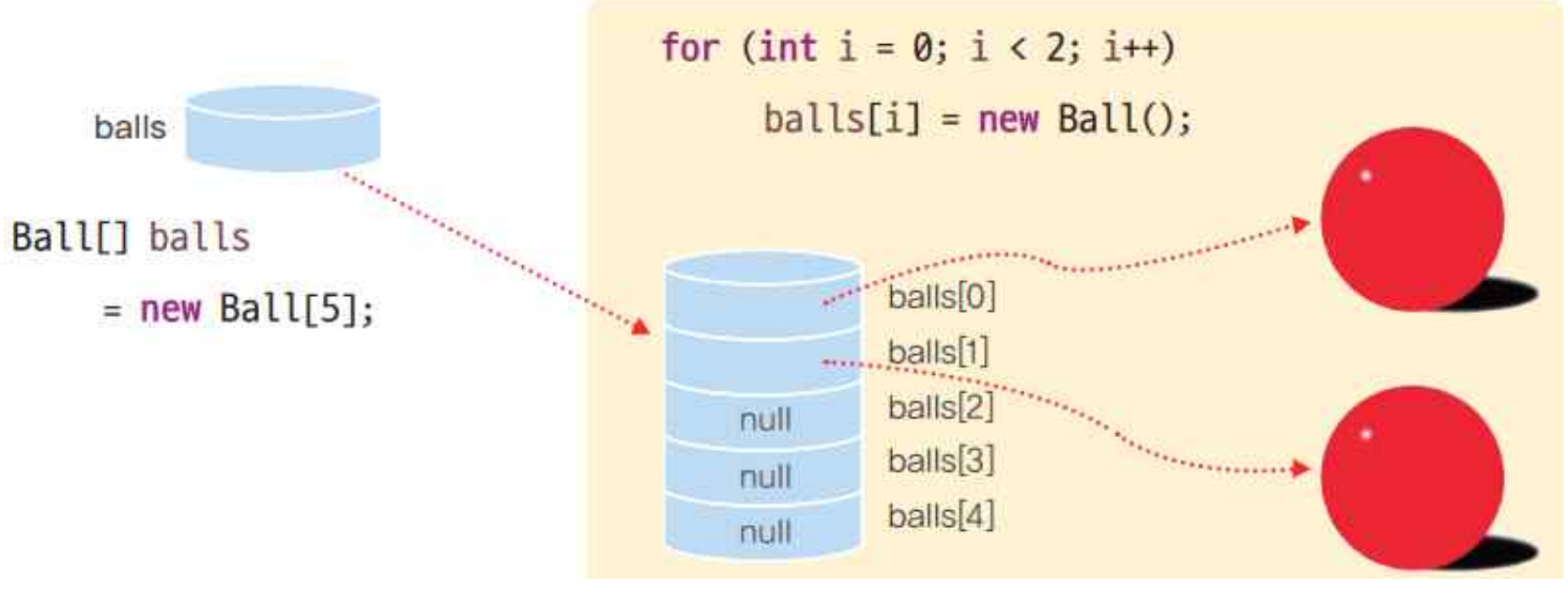
객체의 배열

- 객체 배열은 객체를 참조하는 주소를 원소로 구성
- 예를 들어 Ball 클래스의 객체로 구성된 배열을 선언하고 초기화

```
Ball[] balls = new Ball[5];
```

→ 5개의 Ball 객체를 생성하는 것이 아니라
5개의 Ball 객체를 참조할 변수를 준비

- 생성자를 호출하여 Ball 객체를 생성해야 함



배열 응용

객체의 배열

- 예제 : sec03/CircleArrayDemo

```
원의 넓이(반지름 : 1.0) = 3.14
원의 넓이(반지름 : 2.0) = 12.56
원의 넓이(반지름 : 3.0) = 28.26
원의 넓이(반지름 : 4.0) = 50.24
원의 넓이(반지름 : 5.0) = 78.50
```

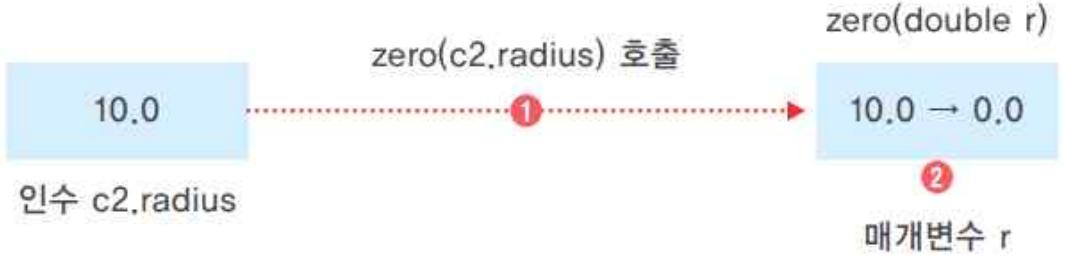
배열 응용

매개변수로 객체 전달

- 예제 : sec03/ObjectArgumentDemo

```
원(c1)의 반지름 : 0.0
원(c2)의 반지름 : 10.0
```

1. 인수 c1이 매개변수 c에 복사된다.
2. c1과 c는 동일한 Circle 객체를 가리킨다.
3. zero( ) 메서드에서 c.radius에 0.0을 대입한다.



열거 타입

- 열거형(enum)
- 관련된 상수들을 같이 묶어 놓은 것, Java는 타입에 안전한 열거형을 제공함

```
class People {
        static final int MALE = 0;
        static final int FEMALE = 1;
        static final int ONE = 1;
        static final int TWO = 2;
        final int kind;
        final int num;
}
```

```
if(PEOPLE.FEMALE == PEOPLE.ONE) // TRUE 임
```

```
if(PEOPLE.Gender.FEMALE == PEOPLE.Num.ONE) // 컴파일 오류
, 타입이 달라서 비교 불가
```

```
==> class People {
        enum Gender { MALE , FEMALE }   // 열거형 Gender를 정의
        enum Num { ONE, TWO }
        final Gender gender; // 타입이 int 가 아닌 Gender임
    }
```

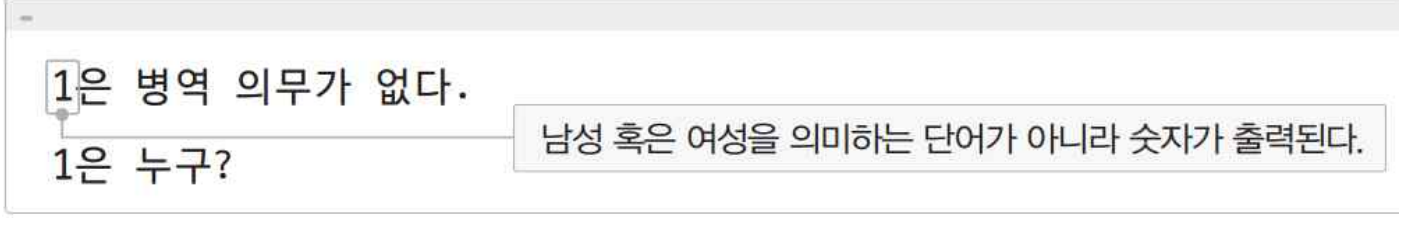
열거 타입

필요성

- 제한된 수의 일이나 사건 등에 대하여 숫자로 표현
  - 각 숫자에 대하여 부여된 의미를 개발자가 숙지 => 일이나 사건에 대한 경우의 수가 많다면 개발자 관점에서 불편
  - 부여되지 않은 의미 없는 숫자 => 컴파일러는 알 수 없다.
  - 출력 값이 의미 없는 숫자로 표현

- 제한된 사건에 대하여 숫자 대신에 상수를 정의해서 부여
  - 숫자에 부여된 의미를 개발자가 알 수 있지만, 여전히 나머지 문제가 미결

- 예제 : sec04/ConstantDemo



- 자바 5부터 열거 타입을 제공

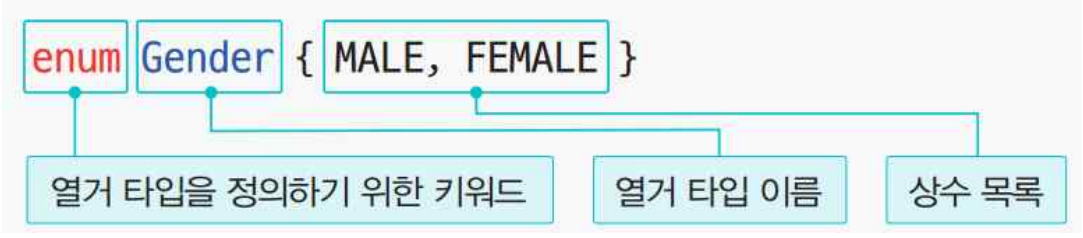
열거 타입

열거 타입과 응용

- 열거 타입 : 서로 연관된 사건들을 모아 상수로 정의한 java.lang.Enum클래스의 자식 클래스
- 선언

```
enum 열거타입이름 { 상수목록 }
```

- 예




```
class People {
    int age;
    Gender gender;  // Gender 에서 정의해놓은 상수만 사용할 수 있음
}
```

- 열거형 상수의 비교에 == 와 compareTo() 사용가능(<, > 비교 연산자 사용불가)

- 예제 : sec04/one/EnumDemo

- 열거형의 조상 - java.lang.Enum
- 모든 열거형은 Enum의 자손이며, 아래의 메소드를 상속받는다.

메서드	설명
Class<E> getDeclaringClass( )	열거형의 Class객체를 반환
String name( )	열거형 상수의 이름을 문자열로 반환
int ordinal( )	열거형 상수가 정의된 순서를 반환(0부터 시작)
T valueOf(Class<T> enumType, String name)	지정된 열거형에서 name과 일치하는 열거형 상수를 반환

열거 타입

열거 타입과 응용

- 일종의 클래스 타입인 열거 타입도 생성자, 필드 및 메서드를 가질 수 있다.
- 열거 타입 상수는 생성자에 의한 인스턴스이다.
- 이때 생성자, 필드 및 메서드와 열거 타입 상수를 구분하기 위하여 다음과 같이 열거 타입 상수 뒤에 반드시 세미콜론을 추가해야 한다.

```
enum 열거타입이름 {
    열거타입상수1, 열거타입상수2, . . . ;
    // 필드
    // 생성자
    // 메서드
}
```

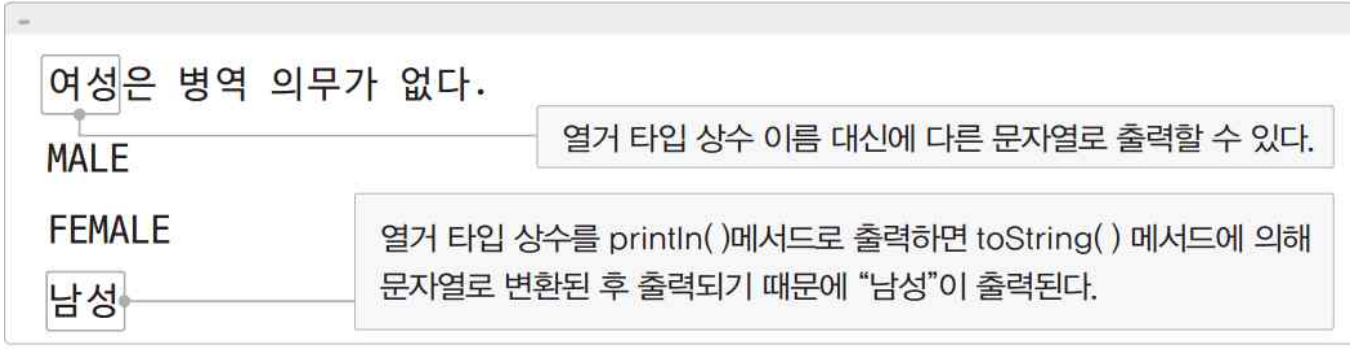
필드, 생성자, 메서드가 있다면 서로 구분하는 데 필요하다.

필요하면 추가할 수 있는 선택사항이다.

열거 타입

열거 타입과 응용

- 예제 : sec04/two/EnumDemo



- 예제 : sec04/SwitchDemo