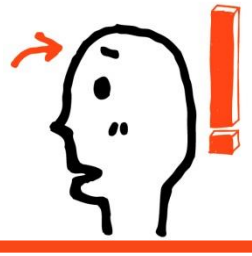
Android Java

12장 배열과 열거타입

배열



01 배열이라는 존재가 필요한 이유

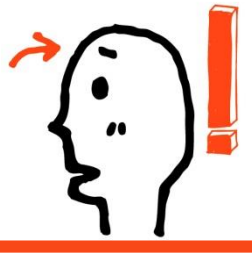
02 1차원 배열의 이해와 활용

03 다차원 배열의 이해와 활용

04 for-each문

05 main 메소드로의 데이터 전달

변수 선언의 편의성



배열을 이용하면 아무리 많은 수의 변수라 할지라도 하나의 문장으로 선언하는 것이 가능하다.

일반적인 방식의 변수 선언

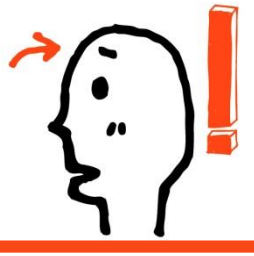
```
int num1, num2, num3;
int num4, num5, num6;
int num7, num8, num9;
```

배열 기반의 변수 선언

```
int[ ] numArr=new int[9];
```

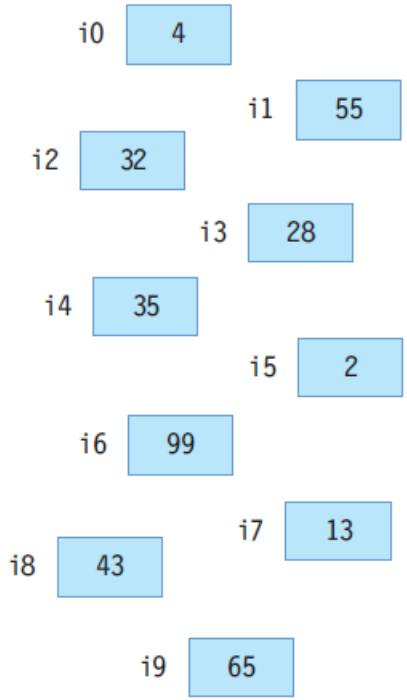
총 9개의 변수가 선언되었다는 사실에는 차이가 없다.

변수 선언의 편의성



(1) 10개의 정수형 변수를 선언하는 경우

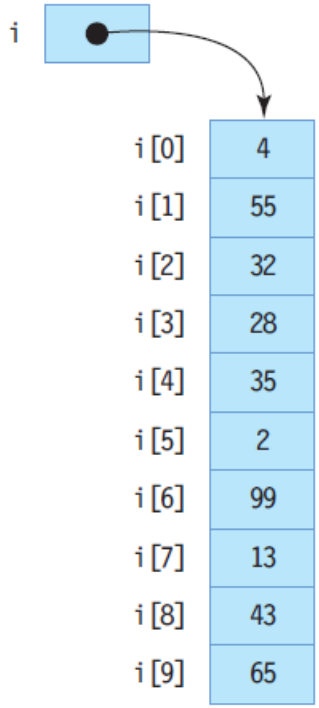
```
int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9;
```




```
sum = i0+i1+i2+i3+i4+i5+i6+i7+i8+i9;
```

(2) 10개의 정수로 구성된 배열을 선언하는 경우

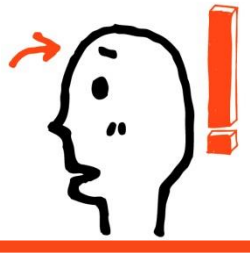
```
int i[] = new int[10];
```




```
for(sum=0, n=0; n<10; n++)
    sum += i[n];
```

배열이라는 존재가 필요한 이유

순차적 접근의 허용



배열을 이용하면 반복문을 이용한, 배열을 구성하는 변수에 순차적으로 접근할 수 있다.

일반적인 방식의 변수 선언

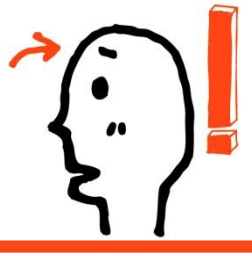
```
num1=num2=num3=10;
num4=num5=num6=10;
num7=num8=num9=10;
```

배열 기반의 변수 선언

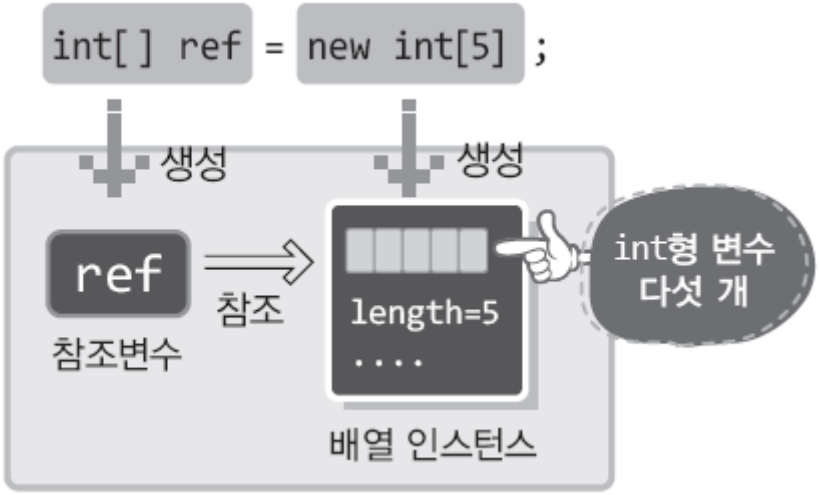
```
for(int i=0; i<9; i++)
{
    numArr[i]=10;
}
```

값을 참조 및 변경할 변수의 수가 100개라고 가정해보면, 위의 차이는 더 극명하게 드러난다.

배열 인스턴스의 생성방법



배열도 인스턴스이다. 둘 이상의 데이터를 저장할 수 있는 형태의 인스턴스이다.



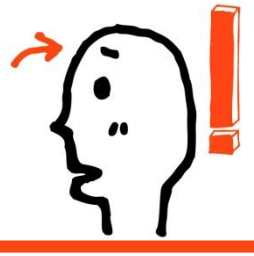
배열 인스턴스와 참조의 생성 모델

배열 인스턴스의 생성의 예

```
int[] arr1 = new int[7];
double[] arr2 = new double[10];
boolean[] arr3 = new boolean[6];
FruitSeller[] arr4 = new FruitSeller[5];
FruitBuyer[] arr5 = new FruitBuyer[8];
```

배열 인스턴스의 생성방법은 일반적인 인스턴스의 생성방법과 차이가 있다.

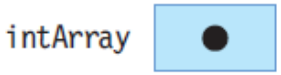
배열 인스턴스의 생성방법



(1) 배열에 대한 레퍼런스 변수 intArray 선언

```
int intArray [];
```

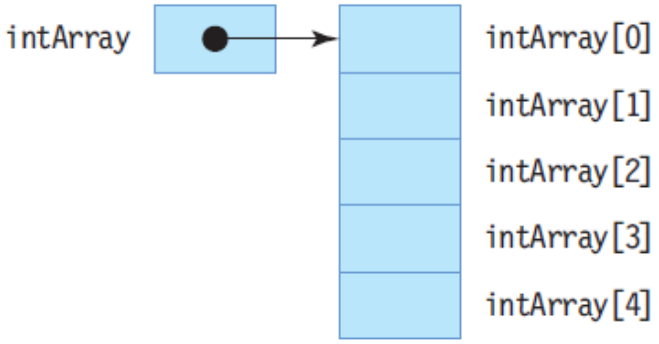
- `int` : 배열 타입
- `intArray` : 배열에 대한 레퍼런스 변수
- `[]` : 배열 선언



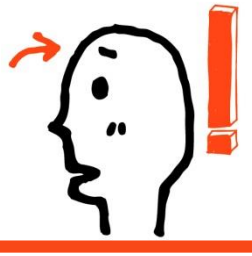
(2) 배열 생성

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

- `intArray` : 배열에 대한 레퍼런스 변수
- `new` : 배열 생성
- `int` : 타입
- `[5]` : 원소 개수



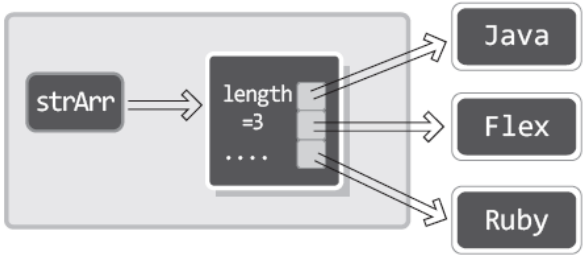
배열의 접근방법



• 배열의 접근에는 0부터 시작하는 인덱스 값이 사용된다. 가장 첫 번째 배열 요소의 인덱스가 0이고 N번째 요소의 인덱스가 N-1이다.
• 배열 인스턴스의 멤버변수 length에는 배열의 길이정보가 저정되어 있다.

기본 자료형 배열

```
public static void main(String[] args)
{
    int[] arr = new int[3];
    arr[0]=1;
    arr[1]=2;
    arr[2]=3;

    int sum=arr[0]+arr[1]+arr[2];
    System.out.println(sum);
}
```



인스턴스의 배열

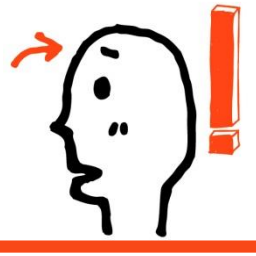
```
public static void main(String[] args)
{
    String[] strArr=new String[3];
    strArr[0]=new String("Java");
    strArr[1]=new String("Flex");
    strArr[2]=new String("Ruby");

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

위 예제는 배열요소의 순차 접근을 보이고 있다!

인스턴스 배열에는 인스턴스가 저장되는 것이 아니라,

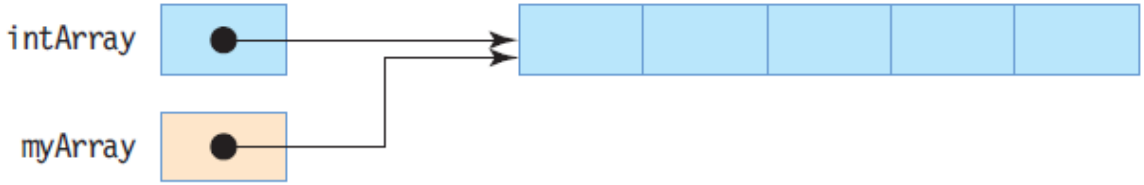
인스턴스의 참조 값이 저장된다.

배열의 참조

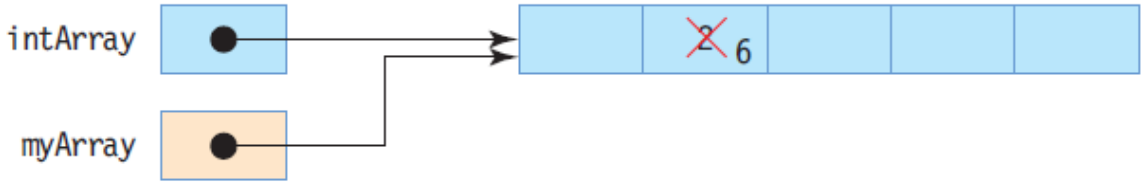


- 생성된 하나의 배열을 다수의 레퍼런스가 참조 가능

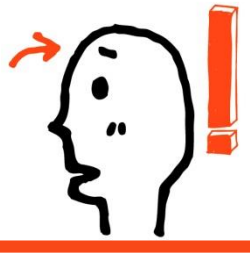
```
int intArray[] = new int[5];
int myArray[] = intArray;
```




```
intArray[1] = 2;
myArray[1] = 6;
```



배열의 선언과 동시에 초기화



일반적인 인스턴스 배열의 선언

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

초기화할 데이터들을 중괄호 안에 나열한다.

```
int[] arr = new int[3] {1, 2, 3};
```

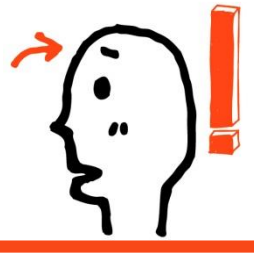
단, 초기화 데이터의 수를 통해서 길이의 계산이 가능하므로 길이정보 생략하기로 약속!

```
int[] arr = new int[ ] {1, 2, 3};
```

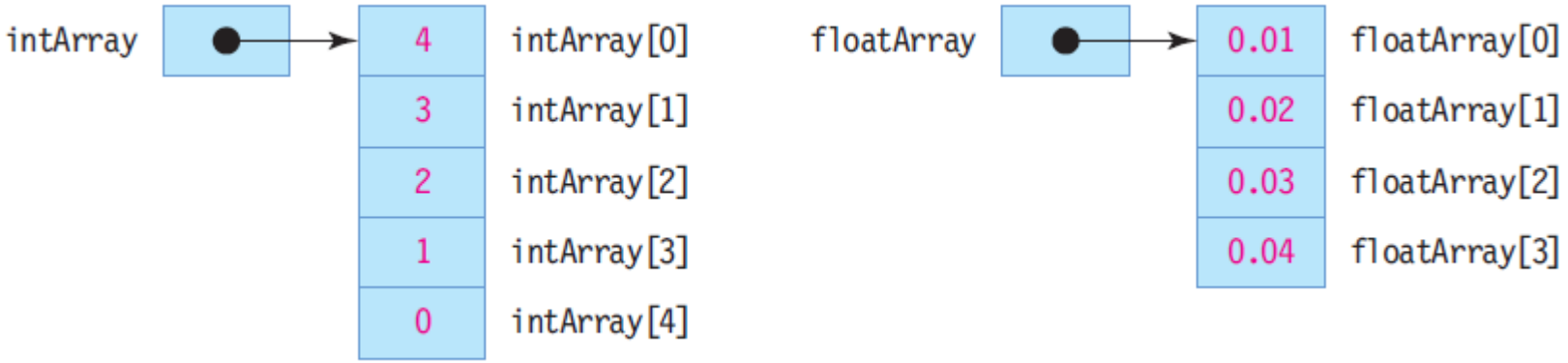
이렇게 줄여서 표현하는 것도 가능하다.

```
int[] arr = {1, 2, 3};
```

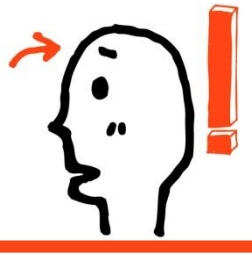
배열의 선언과 동시에 초기화



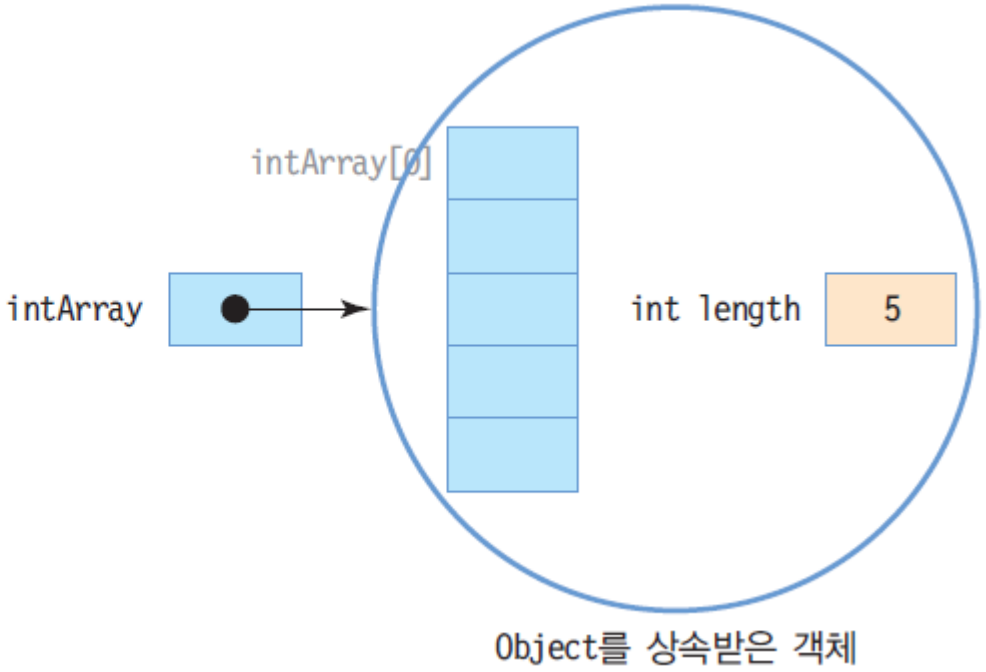
```
int intArray[] = {4, 3, 2, 1, 0};
float floatArray[] = {0.01, 0.02, 0.03, 0.04};
```



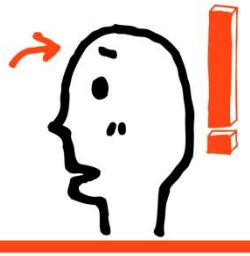
배열은 객체로 관리



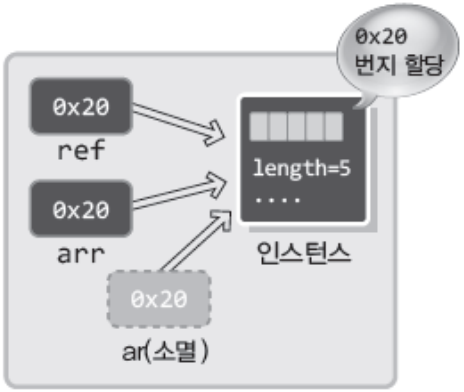
```
int intArray[];
intArray = new int[5];

int size = intArray.length;
// size는 5
```



배열과 메소드



배열도 인스턴스이기 때문에 메소드 호출시의 인자전달 및 반환의 과정이 일반적인 인스턴스들과 다르지 않다.

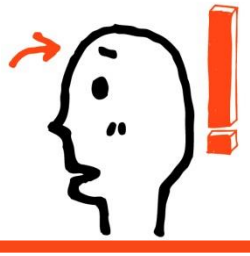



```
class ArrayAndMethods
{
    public static int[] addAllArray(int[] ar, int addVal)
    {
        for(int i=0; i<ar.length; i++)
            ar[i]+=addVal;
        return ar;
    }

    public static void main(String[] args)
    {
        int[] arr={1, 2, 3, 4, 5};
        int[] ref;

        ref=addAllArray(arr, 7);

        if(arr==ref)
            System.out.println("동일 배열 참조");
        else
            System.out.println("다른 배열 참조");

        for(int i=0; i<ref.length; i++)
            System.out.print(arr[i]+" ");
    }
}
```

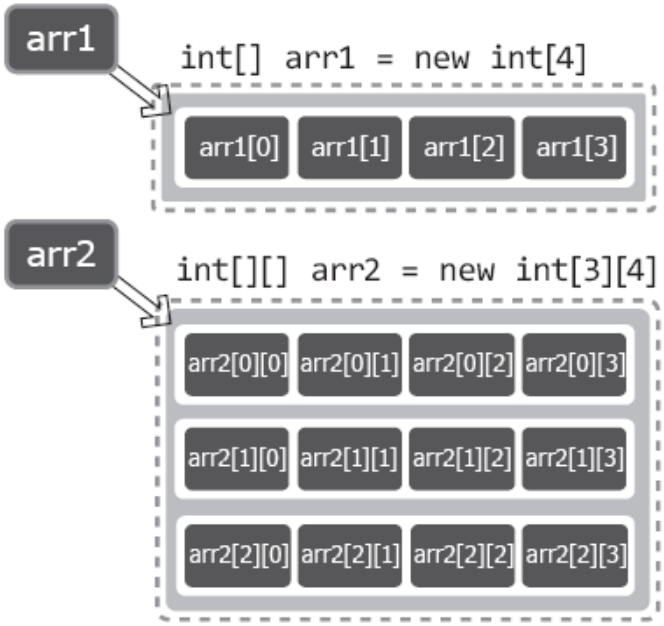
참조 값 ar을 그대로 반환

배열 arr의 참조 값 전달

1차원 배열 vs. 2차원 배열



2차원 배열은 2차원의 구조를 갖는 배열이다. 따라서 가로와 세로의 길이를 명시해서 인스턴스를 생성하게 되며, 배열에 접근할 때에도 가로와 세로의 위치정보를 명시해서 접근하게 된다.

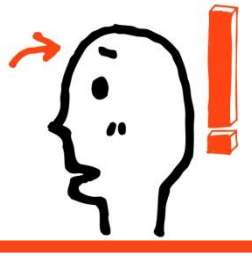


2차원 배열의 선언방법

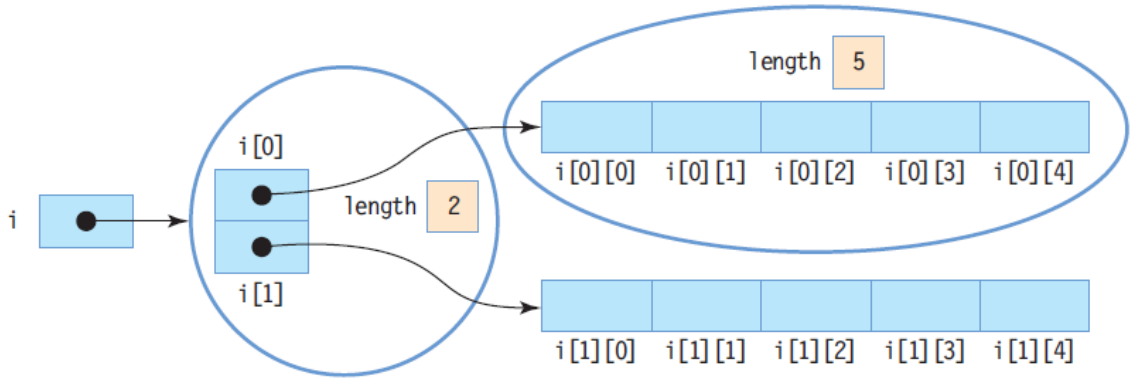
- 가로길이가 2이고, 세로길이가 7인 int형 배열
  → `int[][] arr1 = new int[7][2];`
- 가로길이가 5이고, 세로길이가 3인 double형 배열
  → `double[][] arr2 = new double[3][5];`
- 가로길이가 7이고, 세로길이가 3인 String 배열
  → `String[][] arr3 = new String[3][7];`

2차원 배열에 접근할 때에는 arr[세로][가로]의 형태로 위치를 지정한다.

2차원 배열의 length 필드

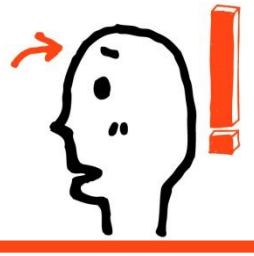


```
int i[][] = new int[2][5];
int size1 = i.length; // 2
int size2 = i[0].length; // 5
int size3 = i[1].length; // 5
```



- 2차원 배열의 length
  - i.length -> 2차원 배열의 행의 개수로서 2
  - i[n].length는 n번째 행의 열의 개수
    - i[0].length -> 0번째 행의 열의 개수로서 5
    - i[1].length -> 1번째 행의 열의 개수로서 역시 5

2차원 배열의 선언 및 초기화



2차원 배열의 선언 및 초기화 1

```
int[ ][ ] arr={
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

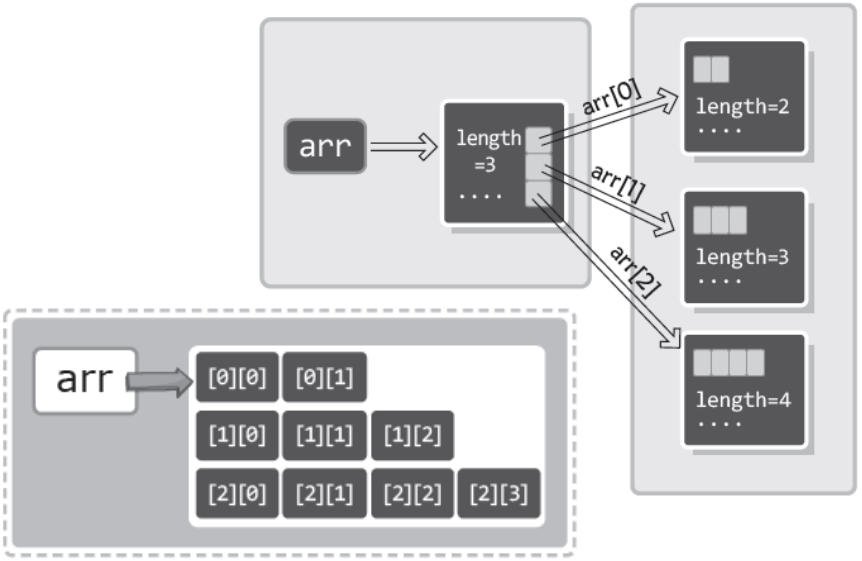
동일

2차원 배열의 선언 및 초기화 2

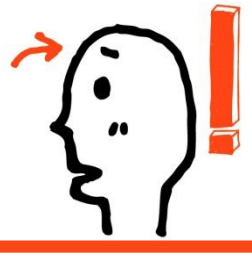
```
int[ ][ ] arr=new int[ ][ ] {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

2차원 배열의 선언 및 초기화 3

```
int[ ][ ] arr={
    {1, 2},
    {3, 4, 5},
    {6, 7, 8, 9}
};
```

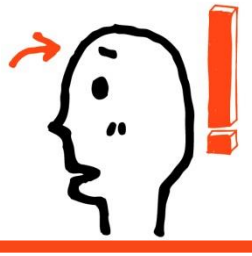


배열 복사

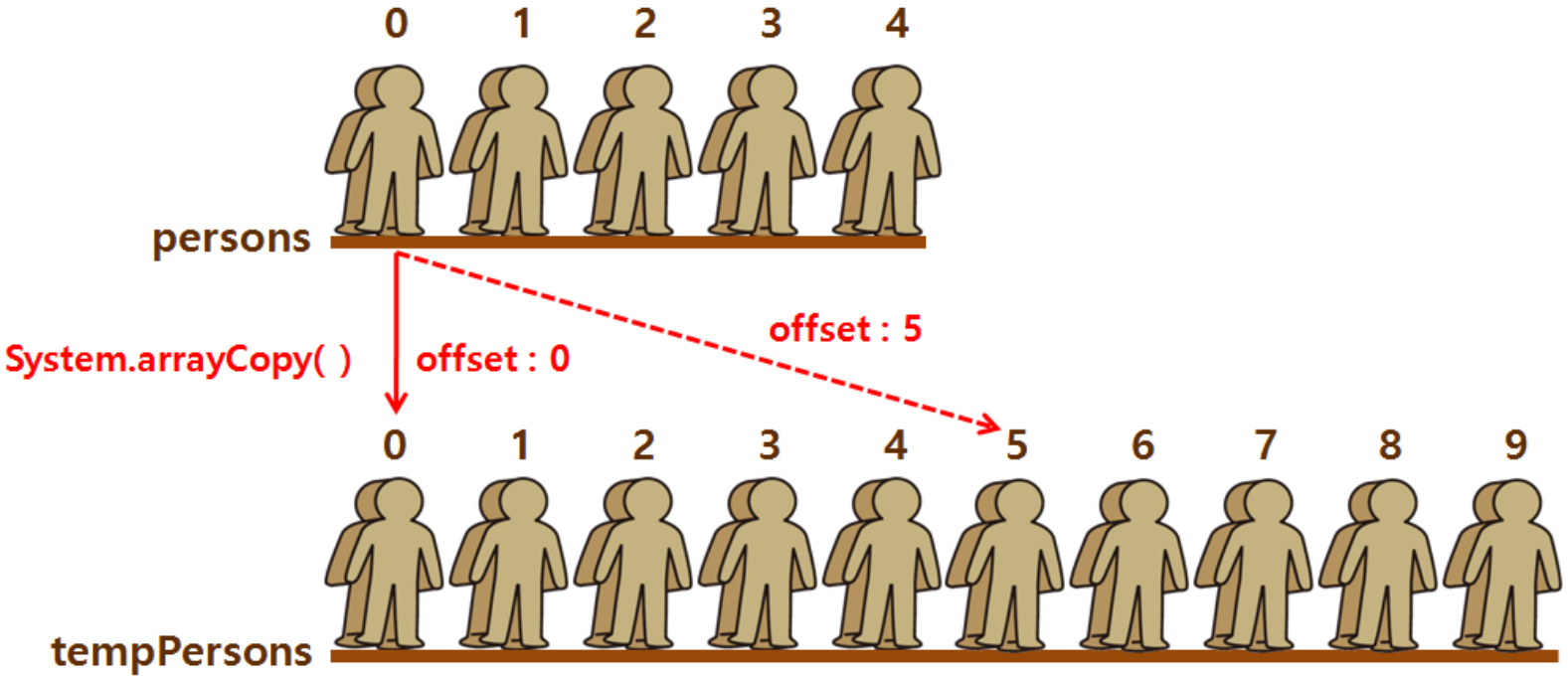


- 배열 복사
  - 배열은 한 번 생성하면 크기 변경 불가
  - 더 많은 저장 공간이 필요하다면 보다 큰 배열을 새로 만들고 이전 배열로부터 항목 값들을 복사
- 배열 복사 방법
  - for문 이용
  - System.arrayCopy() 메소드 이용

배열 복사

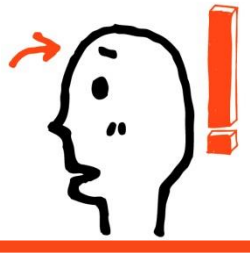


- **System.arrayCopy 메소드를 사용해 배열 복사**



System.arrayCopy(원래 배열 객체, 시작 인덱스, 새로운 배열 객체, 시작 인덱스, 길이)

3년간 매출 총액과 평균 구하기

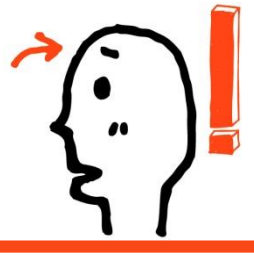


- 한 회사의 지난 3년간 분기별 매출의 총액과 연평균 매출을 구하는 프로그램을 작성하시오.

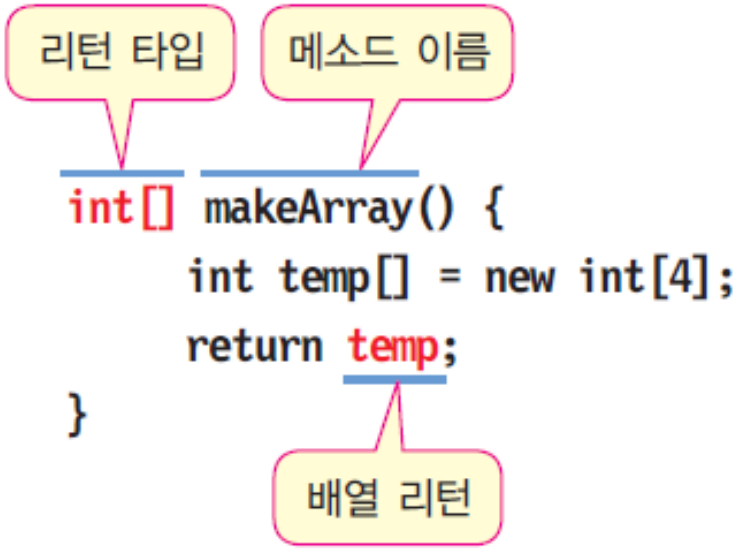
```
public class SalesRevenue {
   public static void main (String[] args) {
      int intArray[][] = {{90, 90, 110, 110},
                          {120, 110, 100, 110},
                          {120, 140, 130, 150}} ;
      double sum = 0;

      for (int i = 0; i < intArray.length; i++) // intArray.length=3
         for (int j = 0; j < intArray[i].length; j++) // intArray[i].length=4
            sum += intArray[i][j];

      System.out.println("지난 3년간 매출 총액은 " + sum + "이며 연평균  매출은 "
                         +  sum/intArray.length + "입니다.");
   }
}
```

```
지난 3년간 매출 총액은 1380.0이며 연평균 매출은 460.0입니다.
```

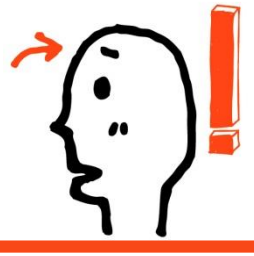
메소드에서 배열 리턴



- 메소드의 배열 리턴
  - 배열의 레퍼런스만 리턴
- 메소드의 리턴 타입
  - 메소드가 리턴하는 배열의 타입은 리턴 받는 배열 타입과 일치
  - 리턴 타입에 배열의 크기를 지정하지 않음



배열 리턴 예제

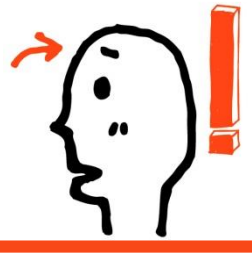


배열을 생성하고 각 원소 값을 출력하는 프로그램을 작성하시오. 배열 생성은 배열을 생성하여 각 원소의 인덱스 값으로 초기화하여 반환하는 메소드를 이용한다.

```
public class ReturnArray {
   static int[] makeArray() {
      int temp[] = new int[4];
      for (int i=0;i<temp.length;i++)
         temp[i] = i;
      return temp;
   }
   public static void main (String[] args) {
      int intArray [];
      intArray = makeArray();
      for (int i = 0; i < intArray.length; i++)
         System.out.println(intArray[i]);
   }
}
```

```
0
1
2
3
```

for-each문의 이해와 활용

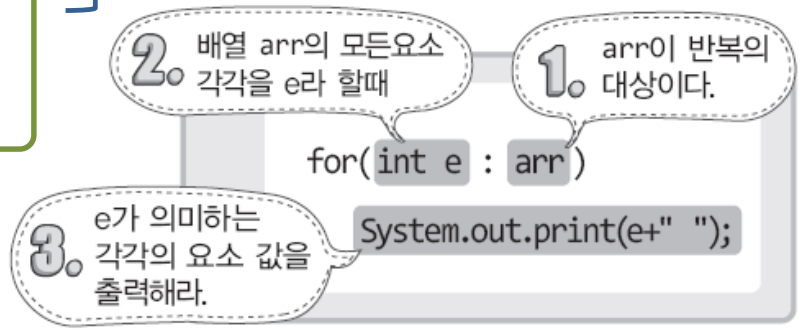


배열의 일부가 아닌, 배열의 전체를 참조할 필요가 있는 경우에 유용하게 사용할 수 있다.

```
for(int i=0; i<arr.length; i++)
    System.out.print(arr[i]+" ");
```

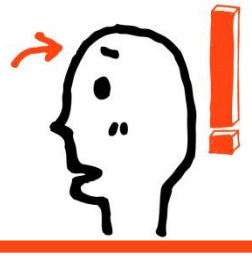
```
for(int e : arr)
    System.out.print(e+" ");
```

코드 분량이 짧아졌고, 필요로 하는 이름의 수가 arr, i, length에서 e와 arr로 그 수가 하나 줄었다.



for-each 문을 통한 값의 변경은 실제 배열에 반영되지 않으니, 값의 참조를 목적으로만 사용해야 한다.

for-each문의 이해와 활용

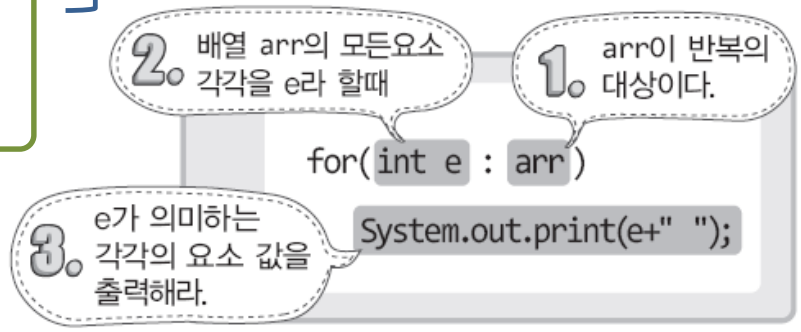


배열의 일부가 아닌, 배열의 전체를 참조할 필요가 있는 경우에 유용하게 사용할 수 있다.

```
for(int i=0; i<arr.length; i++)
    System.out.print(arr[i]+" ");
```

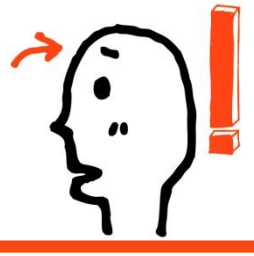
```
for(int e : arr)
    System.out.print(e+" ");
```

코드 분량이 짧아졌고, 필요로 하는 이름의 수가 arr, i, length에서 e와 arr로 그 수가 하나 줄었다.



for-each 문을 통한 값의 변경은 실제 배열에 반영되지 않으니, 값의 참조를 목적으로만 사용해야 한다.

for-each문의 간단한 예



- for-each 문
  - 배열의 각 원소를 순차적으로 접근하는데 유용한 for 문

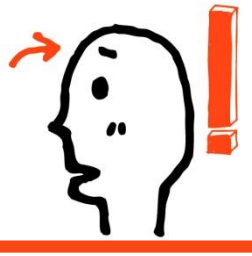
```
int[] num = { 1,2,3,4,5 };
int sum = 0;
for (int k : num) // 반복될 때마다 k는 num[0], num[1], ..., num[4] 값으로 설정
   sum += k;
System.out.println("합은 " + sum);
```

```
합은 15
```

```
String names[] = { "사과", "배", "바나나", "체리", "딸기", "포도" } ;
for (String s : names) // 반복할 때마다 s는 names[0], names[1], ..., names[5] 로 설정
   System.out.print(s + " ");
```

```
사과 배 바나나 체리 딸기 포도
```

인스턴스 배열에 대한 for-each문



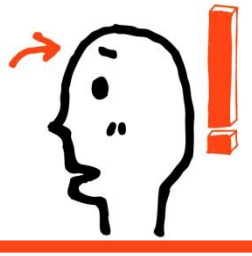
인스턴스 배열에 저장된 참조 값의 변경은 불가능하지만, 참조 값을 통한 인스턴스의 접근은 (접근 과정에서의 데이터 변경은) 가능하다!

```
public static void main(String[] args)
{
    Number[] arr=new Number[] {
        new Number(2),
        new Number(4),
        new Number(8)
    };
    for(Number e : arr)
        e.num++;
    for(Number e : arr)
        System.out.print(e.getNum()+" ");
    System.out.println("");
    for(Number e : arr)
    {
        e=new Number(5);
        e.num+=2;
        System.out.print(e.getNum()+" ");
    }
    System.out.println("");
    for(Number e : arr)
        System.out.print(e.getNum()+" ");
}
```

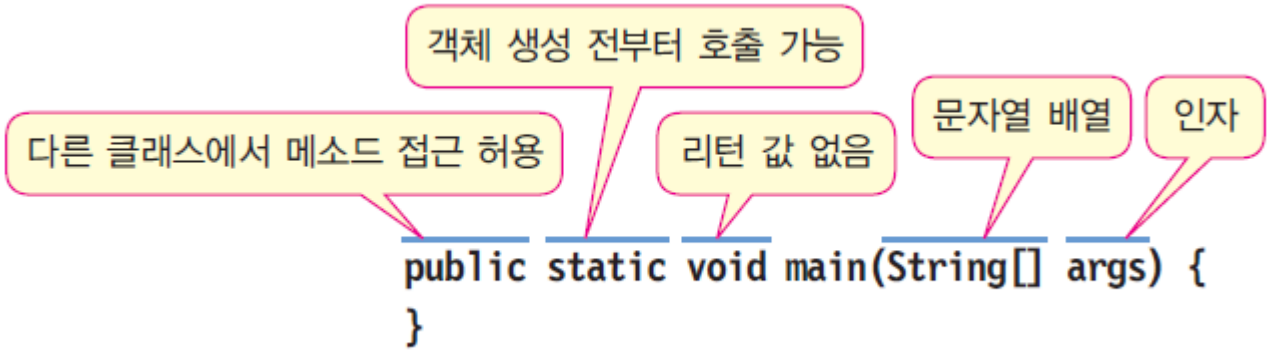
인스턴스의 접근이므로 반영됨

배열의 참조 값 변경이므로 반영 안됨

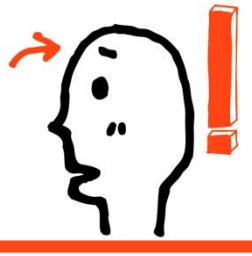
main의 매개변수 선언



- main()은 자바 응용프로그램의 실행 시작 메소드
- main()의 원형
  - 반드시 static
  - 반드시 public
  - 반드시 void
  - 반드시 매개 변수 타입은 문자열 배열



main의 매개변수 선언

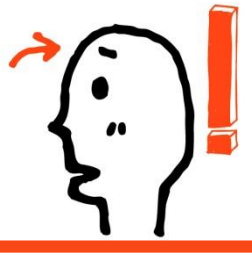


```
public static void main(String[] args) { . . . . }
```

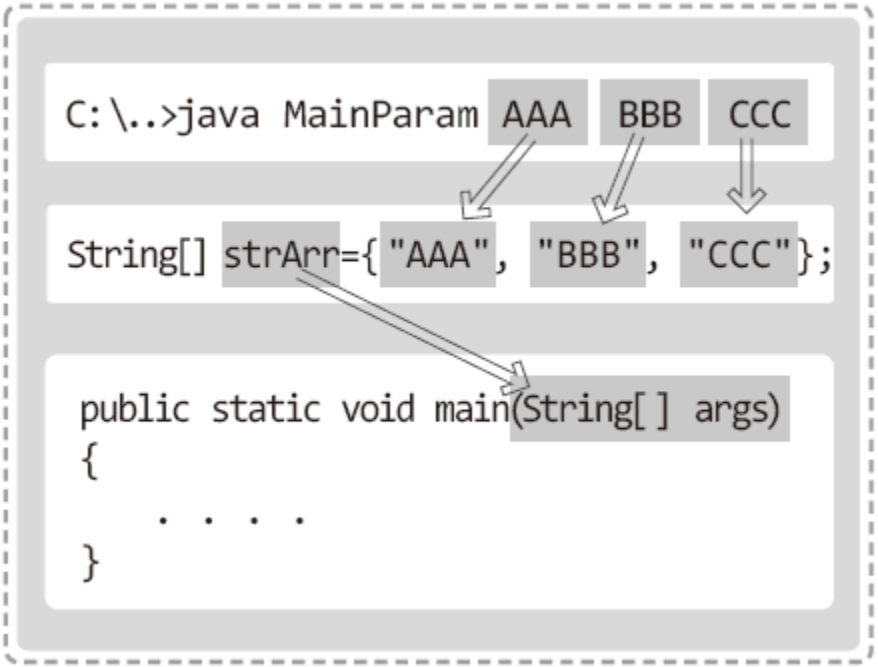
아래에서 보이듯이 main 메소드의 매개변수는 String 인스턴스 배열의 참조 값이 인자로 전달되어야 한다.

```
String[] strArr1={"AAA", "BBB", "CCC"};
String[] strArr2={"public", "static", "void", "main"};
```

main으로의 데이터 전달방법

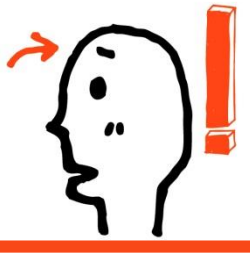


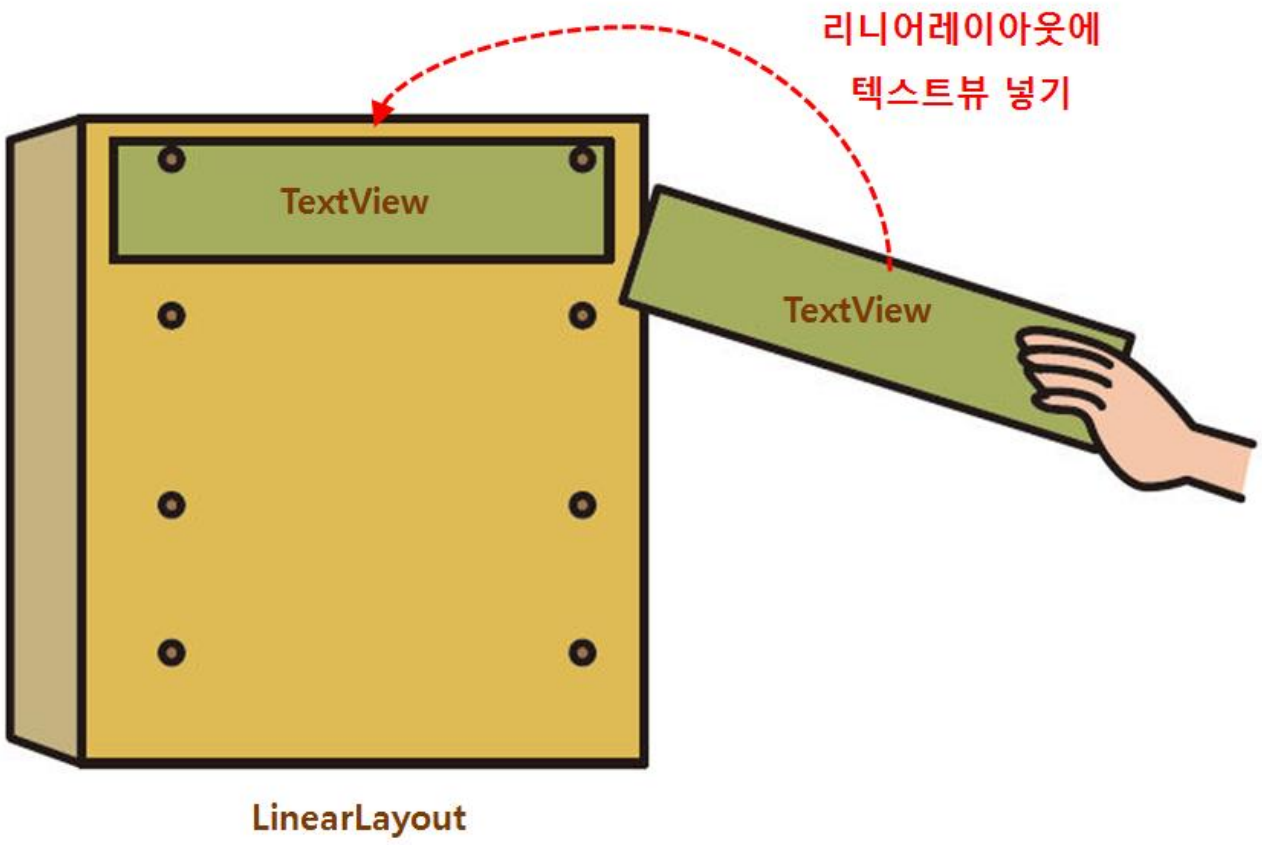
```
C:\JavaStudy>java MainParam AAA BBB CCC
AAA
BBB
CCC
```



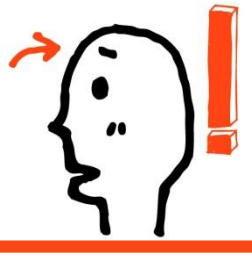
그림에서 보이듯이 명령 프롬프트상에서 전달되는, 공백으로 구분되는 문자열로 String 배열이 구성되어 이 배열의 참조 값이 전달된다.

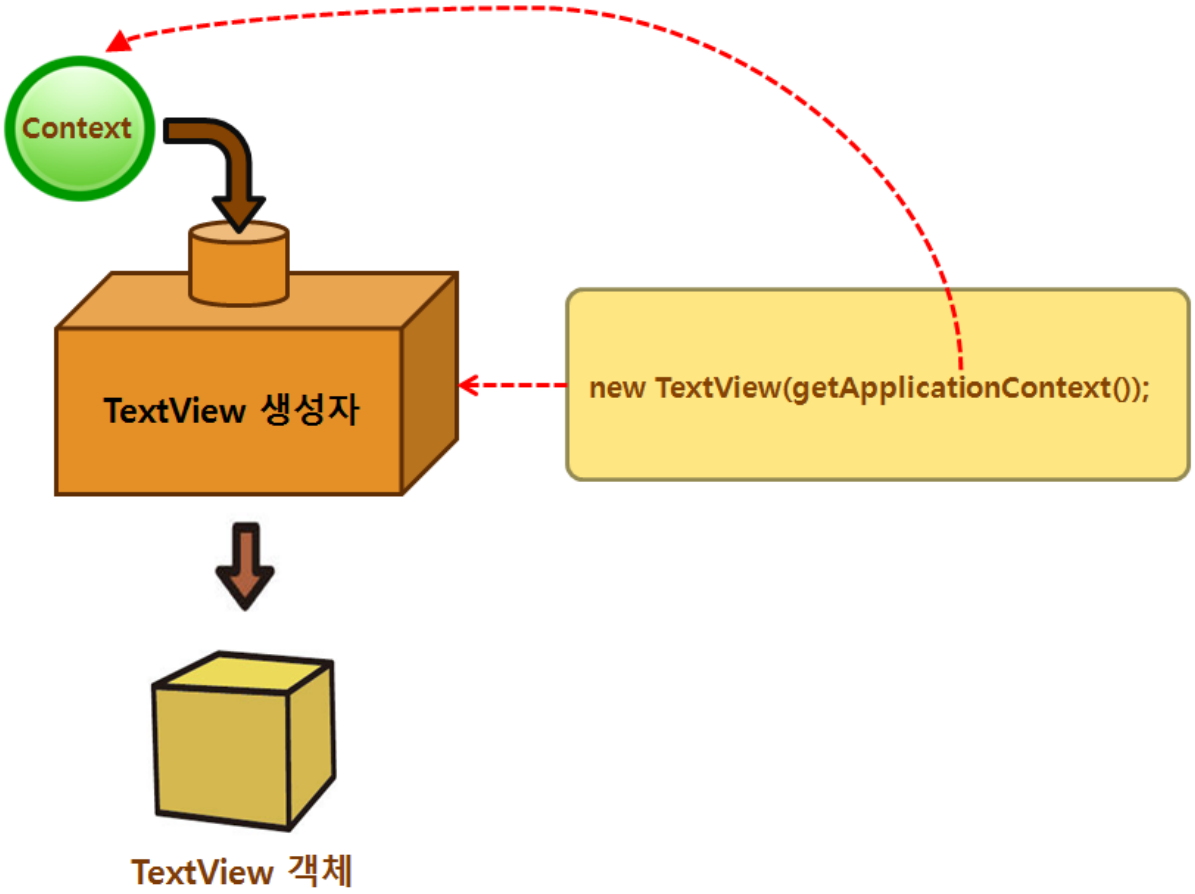
리니어 레이아웃에 텍스트뷰 추가하기



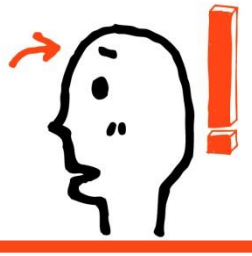


리니어 레이아웃에 텍스트뷰 추가하기





스크롤 뷰의 역할



스크롤뷰

리니어레이아웃

(1) 스크롤뷰 영역보다 작을 때

스크롤뷰

리니어레이아웃

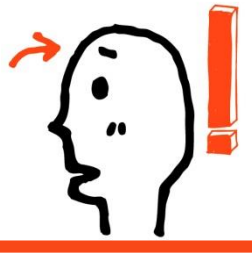
보이는 부분

보이지 않는 부분

스크롤

(2) 스크롤뷰 영역보다 클 때

스크롤 뷰의 역할



스크롤뷰

리니어레이아웃

(1) 스크롤뷰 영역보다 작을 때

스크롤뷰

리니어레이아웃

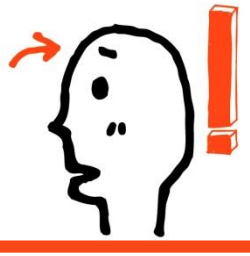
보이는 부분

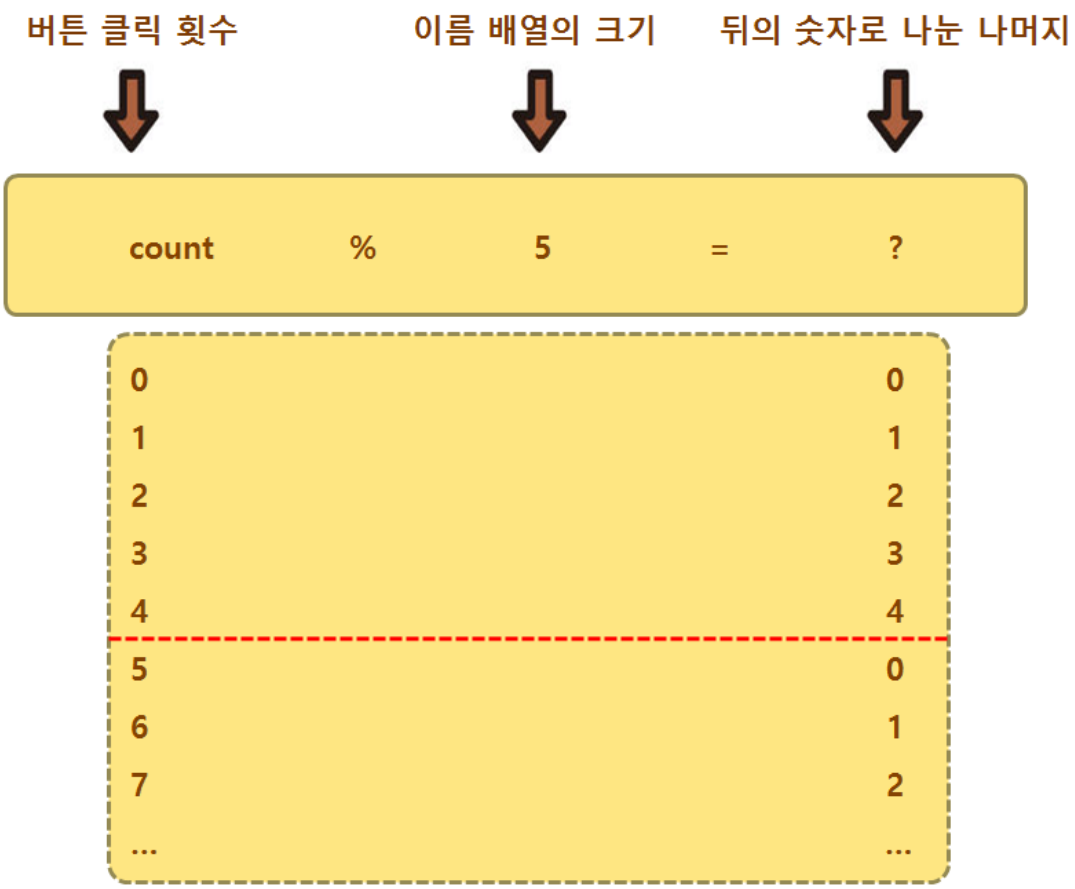
보이지 않는 부분

스크롤

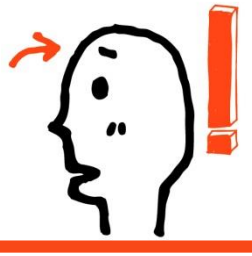
(2) 스크롤뷰 영역보다 클 때

% 연산자로 이름 값을 가지는 배열의 인덱스 계산하기

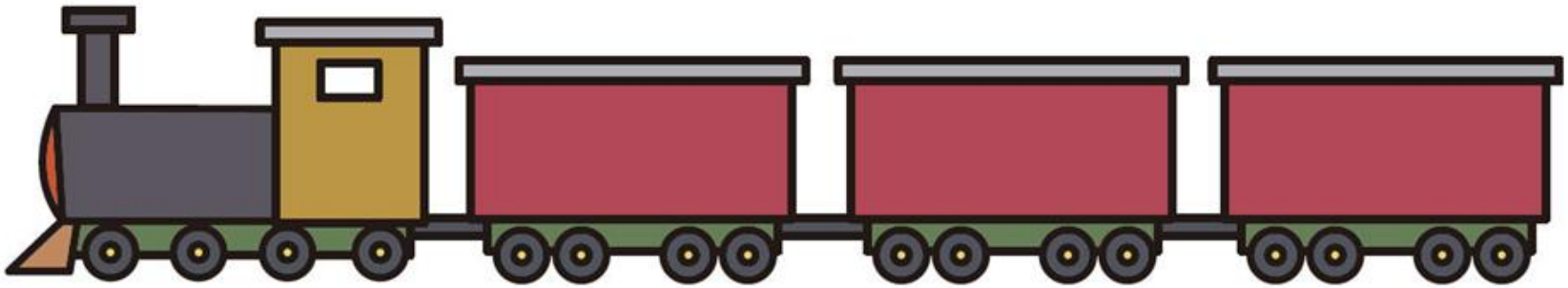




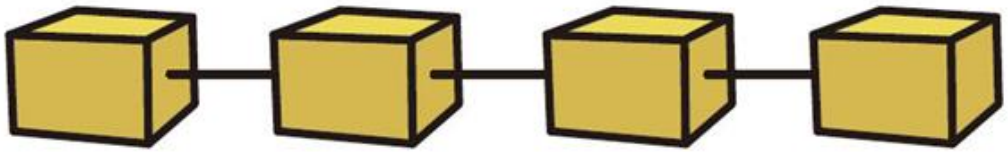
리스트란?



- 여러 개의 값이 서로 연결되어 있는 모양

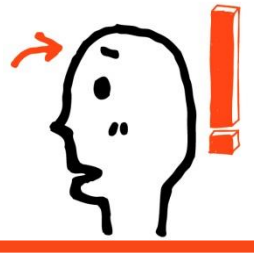


열차 차량이 차례대로 연결되어 있음



변수 상자가 차례대로 연결되어 있음

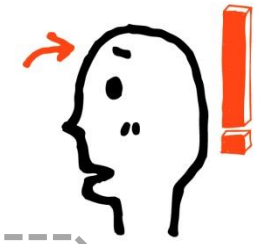
코드에서 리스트 객체 만들기



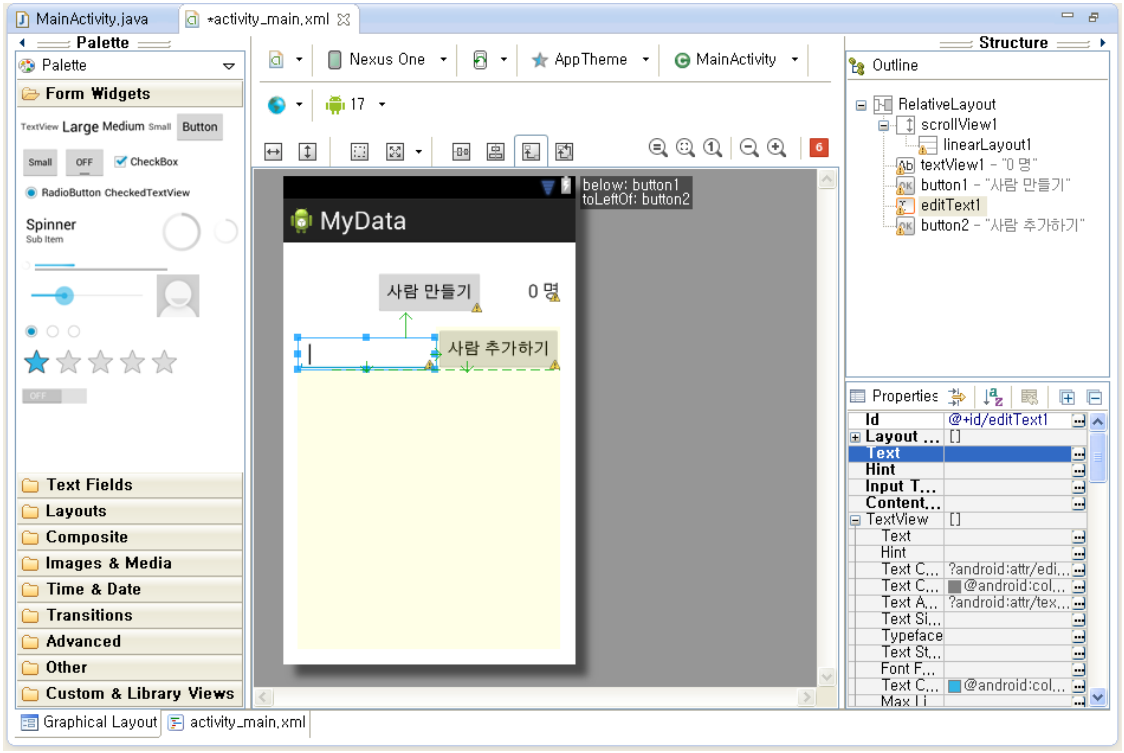
```
ArrayList<Person> persons = new ArrayList<Person>();
```

```
public void onClick(View arg0) {
  int nameIndex = count % 5;
  Person curPerson = new Person(names[nameIndex]);
  persons.add(curPerson);
  Toast.makeText(getApplicationContext(),
                 "사람 " + names[nameIndex] + "이 만들어졌습니다.",
                 Toast.LENGTH_LONG).show();
```

사람 객체를 추가하기 위한 화면 만들기



구분	내용
add( )	추가합니다.
get( )	가져옵니다.
size( )	크기 값을 알 수 있습니다.



리니어 레이아웃에 텍스트뷰 추가하기

Self-Check-1

1. Student 클래스를 만들고 그 안에 String 자료형으로 된 name, int 자료형으로 된 age 속성을 추가한 후 생성자와 함께 이 두 속성의 값을 가져오거나 설정하기 위한 Getter와 Setter 메서드를 만듭니다.

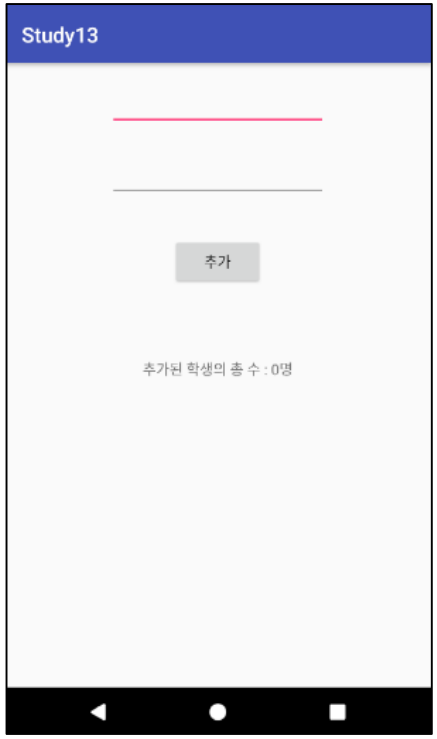
2. School 클래스를 만들고 그 안에 name, students라는 속성을 포함하도록 변수를 선언합니다. students 변수는 선언하면서 동시에 new 연산자를 이용해 객체로 만듭니다

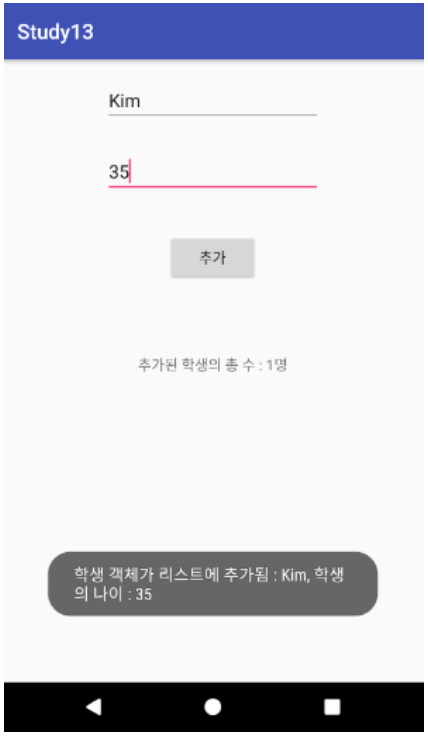
3. name은 String 자료형, students는 ArrayList로 정의하고 이 두 속성의 값을 가져오거나 설정하기 위한 Getter와 Setter 메서드를 만듭니다. 리스트 변수에 Student 객체를 추가할 수 있는 addItem 메서드와 리스트 변수에 들어 있는 Student 객체의 개수를 알 수 있는 size 메서드를 추가합니다

Self-Check-1

4. MainActivity.java 파일을 열고 첫 번째 입력 상자에'Kim'라는 글자를 입력하고, 두번째 입력 상자에'35'라는 글자를 입력합니다. 그런 다음 [추가] 버튼을 누르면 Student 객체를 하나 만들면 그 객체를 students 변수에 추가하도록 코드를 입력합니다.

5. 객체를 리스트 변수에 추가한 후에 토스트로 '학생 객체가 리스트에 추가됨 : Kim, 학생의 나이 : 35 ' 와 같은 메시지를 보여주고, 아래쪽의 텍스트뷰에는 추가된 학생의 총 수가 보이도록 코드를 입력합니다. 추가된 학생의 총 수는 리스트 변수의 size 메서드를 호출하여 알아내도록 합니다





Self-Check-2

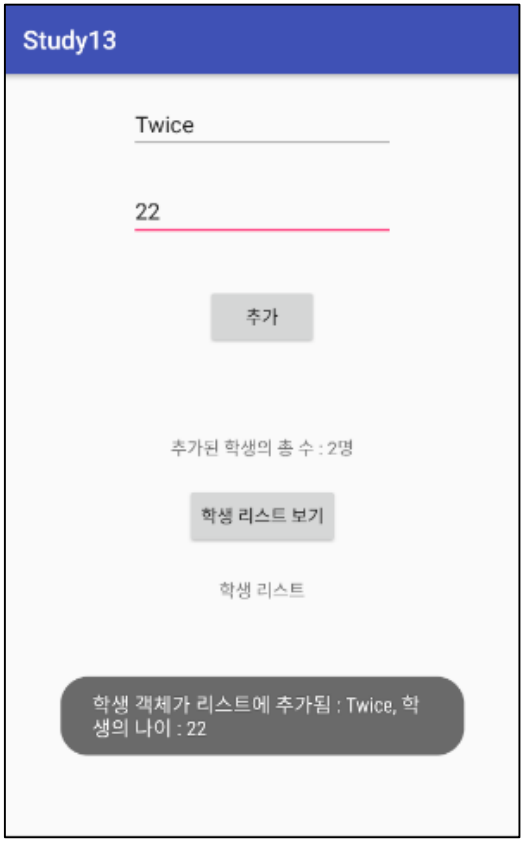
1. School 클래스에 toString이라는 메서드를 추가하고 이 메서드 안에는 Students라는 변수에 들어 있는 Student 객체들의 정보를 하나의 문자열로 변환하는 코드를 넣어줍니다. 다음 줄로 넘어가도록 만들기 위한 문자는 '₩n'이므로 이 문자를 이용해 각각의 Student 정보가 한 줄에 하나씩 보이도록 만들어줍니다.

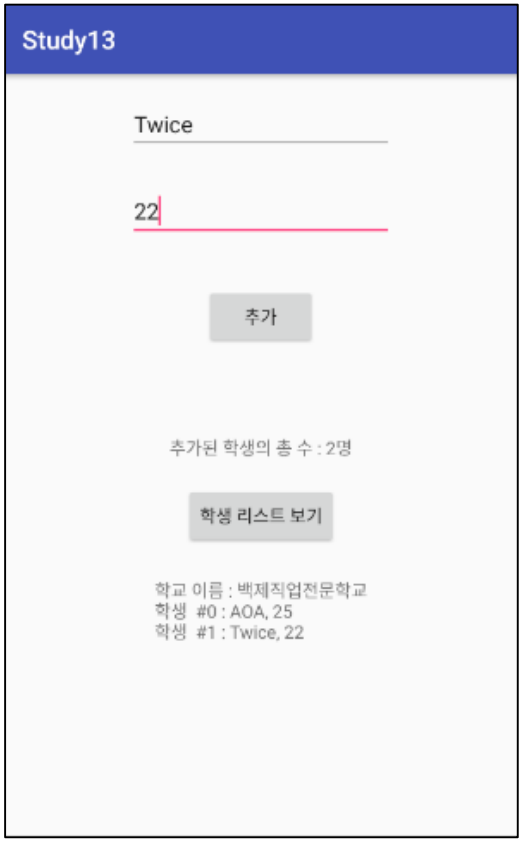
2. activity_main.xml 파일을 열고 화면에 [학생 리스트 보기]라는 버튼을 하나 더 추가합니다.

3. 버튼 아래쪽에는 텍스트뷰 1개를 추가합니다.

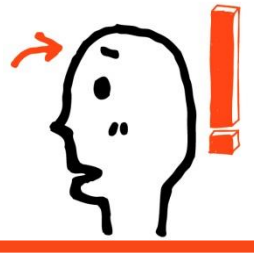
Self-Check-2

4. MainActivity.java 파일을 열고 버튼을 누르면 School 객체의 toString 메서드를 호출하여 학생 정보를 문자열로 바꾼 후 텍스트뷰에 표시하도록 코드를 입력합니다.





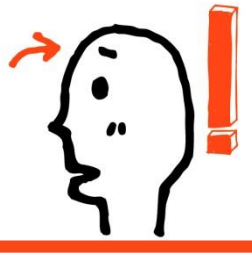
열거 타입



❖ 열거 타입(Enumeration Type)

- 한정된 값만을 갖는 데이터 타입
- 한정된 값은 열거 상수(Enumeration Constant)로 정의

열거 타입



❖ 열거 타입 선언

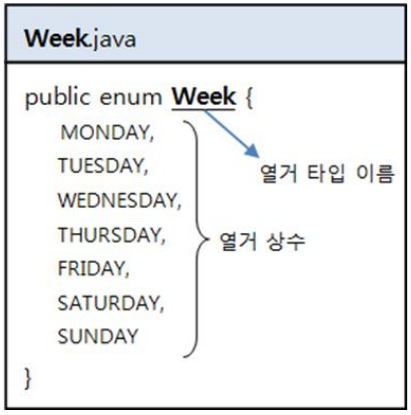
- 파일 이름과 동일한 이름으로 다음과 같이 선언 (첫 글자 대문자)

```
public enum 열거타입이름 { ... }
```

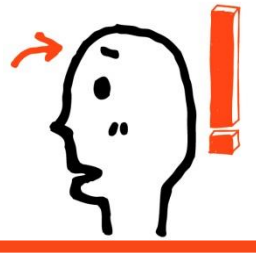
- 한정된 값인 열거 상수 정의
  - 열거 상수 이름은 관례적으로 모두 대문자로 작성
  - 다른 단어가 결합된 이름일 경우 관례적으로 밑줄(_)로 연결

```
public enum Week { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, ... }
```

```
public enum LoginResult { LOGIN_SUCCESS, LOGIN_FAILED }
```



열거 타입



❖ 열거 타입 변수

▪ 열거 타입 변수 선언

```
열거타입 변수;
```

```
Week today;
Week reservationDay;
```

▪ 열거 상수 값 저장 - 열거 타입 변수값은 열거 상수 중 하나

```
열거타입 변수 = 열거타입.열거상수;
```

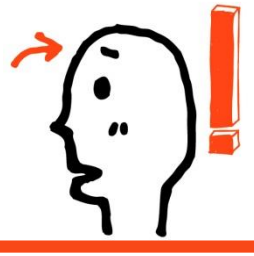
```
Week today = Week.SUNDAY;
```

▪ 열거 타입 변수는 참조 타입

- 열거 타입 변수는 참조 타입이므로 null 값 저장 가능

```
Week birthday = null;
```

자료형의 부여를 돕는 열거형

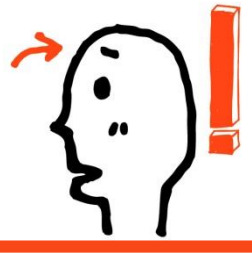


```
enum Scale {      // 열거 자료형 Scale의 정의

   DO, RE, MI, FA, SO, RA, TI

}     열거형 값 (Enumerated Values)
```

case문에서는 표현의 간결함을 위해 Do와 같이 '열거형 값'의 이름만 명시하기로 약속되어 있다.

```
public static void main(String[] args) {
   Scale sc = Scale.DO;
   System.out.println(sc);

   switch(sc) {
   case DO:
      System.out.println("도~ ");
      break;
   case RE:
      System.out.println("레~ ");
      break;
   case MI:
      System.out.println("미~ ");
      break;
   case FA:
      System.out.println("파~ ");
      break;
   default:
      System.out.println("솔~ 라~ 시~ ");
   }
}
```

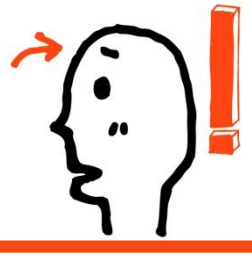
열거형 기반으로 수정한 결과와 개선된 부분



```
enum Animal {
   DOG, CAT
}

enum Person {
   MAN, WOMAN
}
```

```
class SafeEnum {
   public static void main(String[] args) {
      who(Person.MAN);   // 정상적인 메소드 호출
      who(Animal.DOG);   // 비정상적 메소드 호출
   }

   public static void who(Person man) {
      switch(man) {
      case MAN:
         System.out.println("남성 손님입니다.");
         break;
      case WOMAN:
         System.out.println("여성 손님입니다.");
         break;
      }
   }
}
```

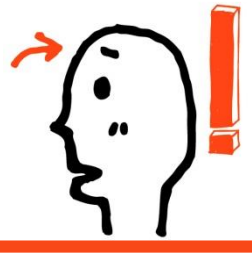
클래스 내에 열거형 정의 가능



```
class Customer {
   enum Gender {       // 클래스 내에 정의된 열거형 Gender
      MALE, FEMALE
   }

   private String name;
   private Gender gen;

   Customer(String n, String g) {
      name = n;

      if(g.equals("man"))
         gen = Gender.MALE;
      else
         gen = Gender.FEMALE;
   }
   . . .
}
```

클래스 내에 열거형이 정의되면 해당 클래스 내에서만 사용 가능한 열거형이 된다.

열거형 값의 정체: 이런 문장 삽입 가능합니다.



```
class Person {
   public static final Person MAN = new Person();
   public static final Person WOMAN = new Person();

   @Override
   public String toString() {
      return "I am a dog person";   // "나는 개를 사랑하는 사람입니다.“
   }
}

class InClassInst {
   public static void main(String[] args) {
      System.out.println(Person.MAN);
      System.out.println(Person.WOMAN);
   }
}
```

열거형 값의 정체: 열거형 값이 인스턴스라는 증거1

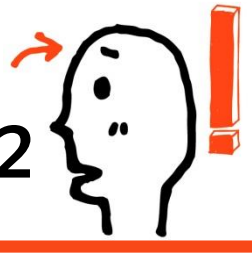
```
enum Person {
    MAN, WOMAN;

    @Override
    public String toString() { return "I am a dog person"; }
}

class EnumConst {
    public static void main(String[] args) {
        System.out.println(Person.MAN);    // toString 메소드의 반환 값 출력
        System.out.println(Person.WOMAN);    // toString 메소드의 반환 값 출력
    }
}
```

```
명령 프롬프트
C:₩JavaStudy>java EnumConst
I am a dog person
I am a dog person

C:₩JavaStudy>
```

모든 열거형은 java.lang.Enum<E> 클래스를 상속한다.

그리고 Enum<E>는 Object 클래스를 상속한다. 이런 측면에서 볼 때 열거형은 클래스이다.

열거형 값의 정체: 열거형 값이 인스턴스라는 증거2

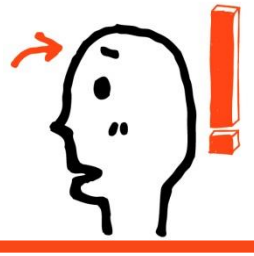


```
enum Person {
    MAN, WOMAN;

    private Person() {
        System.out.println("Person constructor called");
    }

    @Override
    public String toString() { return "I am a dog person"; }
}

class EnumConstructor {
    public static void main(String[] args) {
        System.out.println(Person.MAN);
        System.out.println(Person.WOMAN);
    }
}
```

열거형의 정의에도 생성자가 없으면 디폴트 생성자가 삽입된다.
다만 이 생성자는 private으로 선언이 되어 직접 인스턴스를 생성하는 것이 불가능하다.

```
명령 프롬프트
C:₩JavaStudy>java EnumConstructor
Person constructor called
Person constructor called
I am a dog person
I am a dog person

C:₩JavaStudy>_
```

열거형 값의 정체: 결론

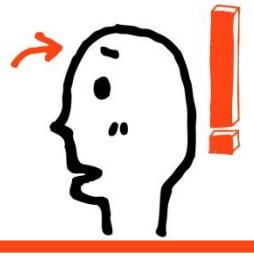


```
enum Person {
    MAN, WOMAN;
    . . . .
}
```

```
public static final Person MAN = new Person();

public static final Person WOMAN = new Person();
```

열거형 값의 실체를 설명하는 문장 실제로 이렇게 컴파일이 되지는 않음

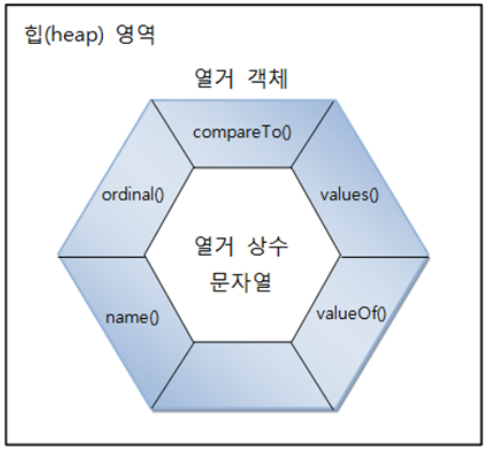
열거 타입

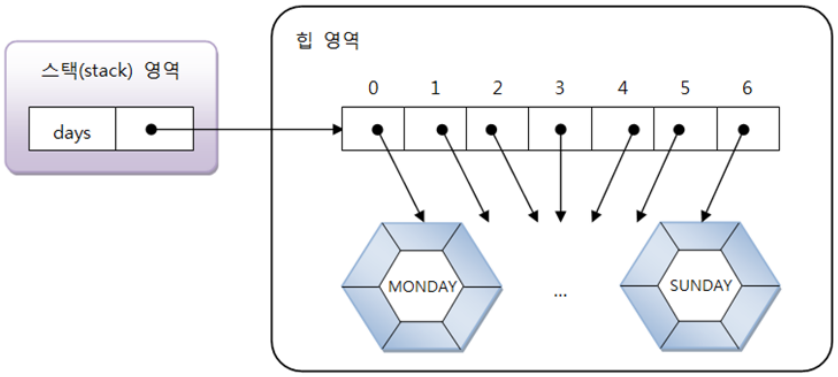


❖ 열거 객체의 메소드

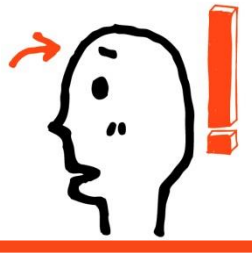
- 열거 객체는 열거 상수의 문자열을 내부 데이터로 가지고 있음
- 열거 타입은 컴파일 시 java.lang.Enum 클래스를 자동 상속
  - 열거 객체는 java.lang.Enum 클래스의 메소드 사용 가능

리턴타입	메소드(매개변수)	설명
String	name()	열거 객체의 문자열을 리턴
int	ordinal()	열거 객체의 순번(0 부터 시작)를 리턴
int	compareTo()	열거 객체를 비교해서 순번 차이를 리턴
열거타입	valueOf(String name)	주어진 문자열의 열거 객체를 리턴
열거배열	values()	모든 열거 객체들을 배열로 리턴





구구단의 결과를 다차원 배열에 저장



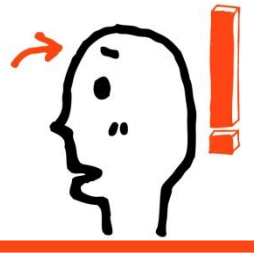
- 구구단의 결과를 2차원 배열에 저장한 다음 출력하는 프로그램을 작성해보자.

Problems @ Javadoc Declaration Console

<terminated> exam_arrygugu [Java Application] C:₩Program Files₩Java₩jre1.8.0_65₩bin₩javaw.exe (2016. 6. 16. 오전 8:53:58)

```
1X1= 1  2X1= 2  3X1= 3  4X1= 4  5X1= 5  6X1= 6  7X1= 7  8X1= 8  9X1= 9
1X2= 2  2X2= 4  3X2= 6  4X2= 8  5X2=10  6X2=12  7X2=14  8X2=16  9X2=18
1X3= 3  2X3= 6  3X3= 9  4X3=12  5X3=15  6X3=18  7X3=21  8X3=24  9X3=27
1X4= 4  2X4= 8  3X4=12  4X4=16  5X4=20  6X4=24  7X4=28  8X4=32  9X4=36
1X5= 5  2X5=10  3X5=15  4X5=20  5X5=25  6X5=30  7X5=35  8X5=40  9X5=45
1X6= 6  2X6=12  3X6=18  4X6=24  5X6=30  6X6=36  7X6=42  8X6=48  9X6=54
1X7= 7  2X7=14  3X7=21  4X7=28  5X7=35  6X7=42  7X7=49  8X7=56  9X7=63
1X8= 8  2X8=16  3X8=24  4X8=32  5X8=40  6X8=48  7X8=56  8X8=64  9X8=72
1X9= 9  2X9=18  3X9=27  4X9=36  5X9=45  6X9=54  7X9=63  8X9=72  9X9=81
```

배열을 이용한 성적 프로그램



- 다음은 키보드로부터 학생 수와 각 학생들의 점수를 입력받아서, 최고 점수 및 평균점수를 구하는 프로그램이다. 실행결과를 보고 알맞게 작성해보자.

```
Problems  Javadoc  Declaration  Console
<terminated> Array_Student [Java Application] C:₩Program Files₩Java₩jre1.8.0_65₩bin₩javaw.exe (2016. 6. 16. 오전 9:03:27)
------------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
------------------------------------------------------------
선택> 1
학생수> 3
------------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
------------------------------------------------------------
선택> 2
scores[0]> 85
scores[1]> 95
scores[2]> 93
------------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
------------------------------------------------------------
선택> 3
scores[0]: 85
scores[1]: 95
scores[2]: 93
------------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
------------------------------------------------------------
선택> 4
최고 점수: 95
평균 점수: 91.0
------------------------------------------------------------
1.학생수 | 2.점수입력 | 3.점수리스트 | 4.분석 | 5.종료
------------------------------------------------------------
선택> 5
프로그램 종료
```

THANK YOU

실무에서 알아야 할 기술은 따로 있다! 자바를 다루는 기술