

## 5. 배열

---

충남대학교  
컴퓨터공학과



# 학습 내용

---

1. 배열의 개념과 사용 목적
2. 배열의 생성 및 사용
3. 배열과 메소드
4. 객체들의 배열
5. 다차원 배열
6. 정렬과 탐색



# 배열의 사용

- 배열이 필요한 경우
  - 대용량의 데이터를 처리하는 프로그램을 작성
  - 같은 타입의 값들을 여러 개 저장해야 하는 경우
  - 예를 들어 학과 학생들의 점수, 회사 근로자들의 급여액
- 배열은 같은 타입의 값들의 리스트
  - 각 값들은 배열 원소(**array element**)
  - 각 원소는 인덱스(**index**)가 부여된다.

전체 배열에  
단일 이름이 주어진다

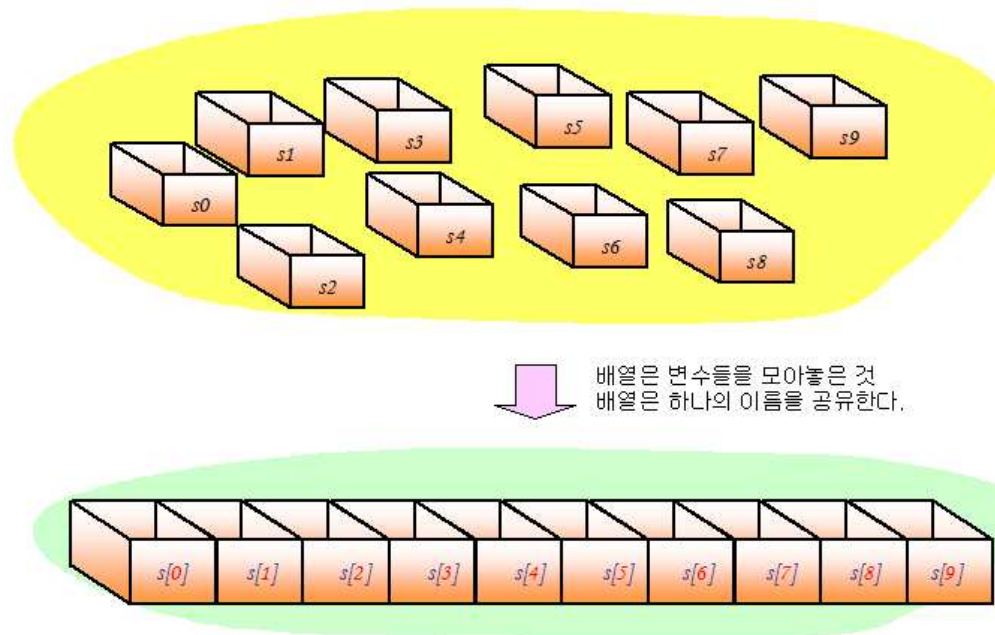
각 값에는 숫자 인덱스가 부여된다

	0	1	2	3	4	5	6	7	8	9
scores	71	80	64	92	97	88	83	71	84	81



# 배열의 개념

- 배열(array): 같은 타입의 변수들의 모임

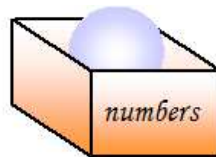




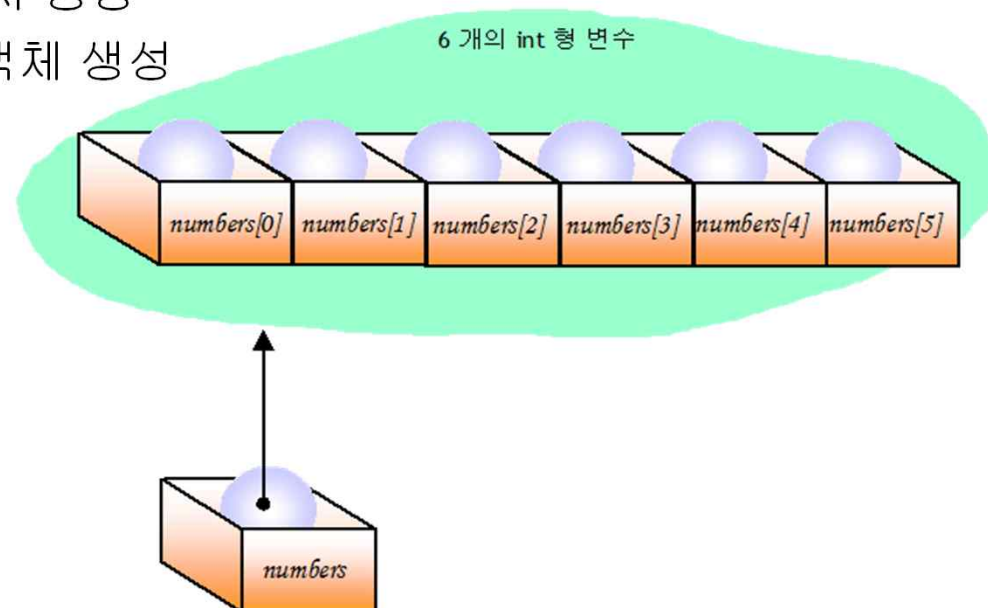
# 배열을 만드는 절차

1. 먼저 배열 참조 변수부터 선언

`int[] numbers;` // 배열 참조 변수 선언



2. 배열을 `new` 연산자를 사용하여 생성  
`numbers = new int[6];` // 배열 객체 생성





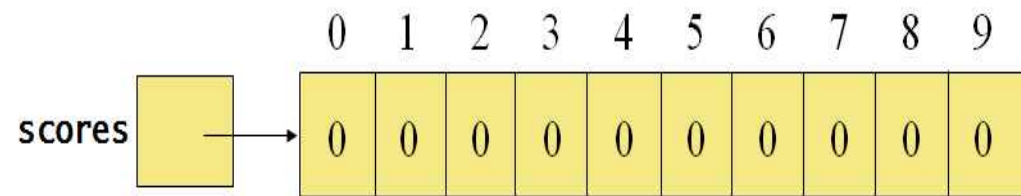
# 배열 선언

- 배열 선언

배열원소타입[] 배열이름;

int[] scores;

scores = new int[10];



---

## Key Point

배열도 다른 객체처럼 **new** 연산자를 이용해서 반드시 실체화되어야 함

---

- 배열 선언 및 초기화

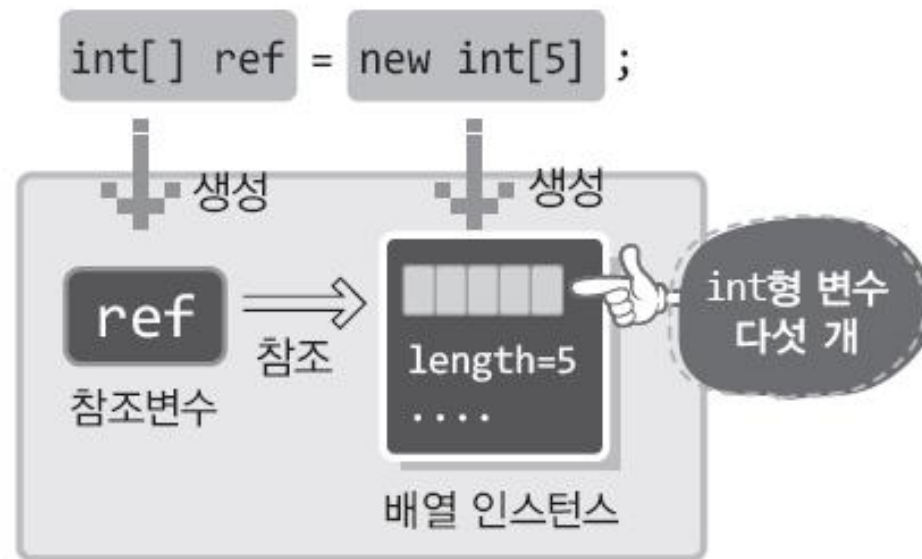
배열원소타입[] 배열이름 = **new** 배열원소타입[크기];

int[] scores = new int[10];



# 배열 객체의 생성 방법

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



**배열 객체와 참조의 생성 모델**



# 배열 선언 예

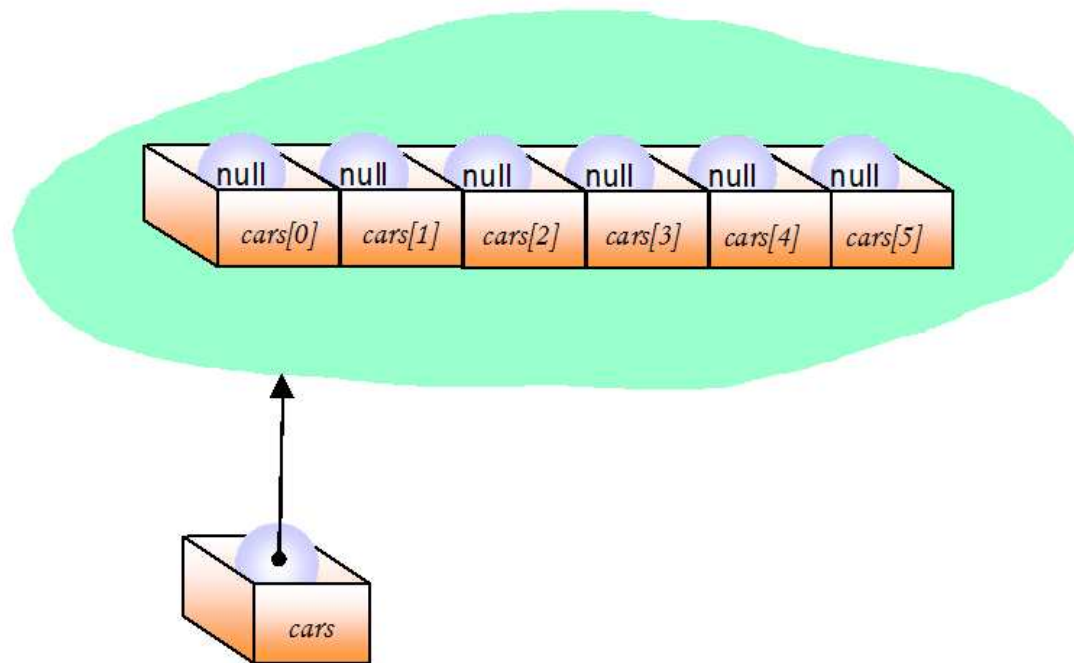
- `int[] prices = new int[100];` // int 배열 선언 및 생성
- `char[] grades;` // char 배열 선언
- `grades = new char[40];` // char 배열 생성
  
- `String[] members = new String[10];`  
// String 배열 선언 및 생성
- `Student[] students;` // Student 배열 선언
- `students = new Student[100];` // Student 배열 생성





# 객체들의 배열

- 객체들의 배열에서는 객체에 대한 참조값만을 저장  
`Car[] cars = new Car[5];`



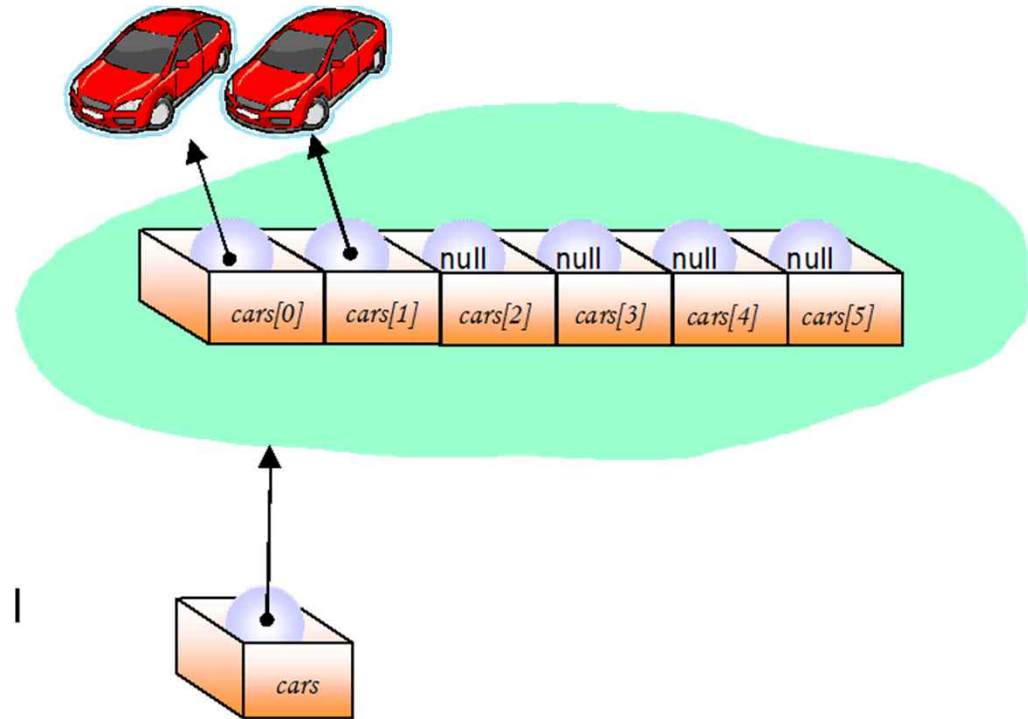
# 객체들의 배열



- 각 원소에 들어가는 객체는 따로 생성하여야 한다.

`cars[0] = new Cars();`

`cars[1] = new Cars();`



# 배열 접근 방법

- 배열의 접근에는 0부터 시작하는 인덱스 값이 사용된다. 가장 첫 번째 배열 요소의 인덱스가 0이고 N번째 요소의 인덱스가 N-1이다.
- 배열 객체의 멤버변수 **length**에는 배열의 길이정보가 저장되어 있다.

## 기본 자료형 배열

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

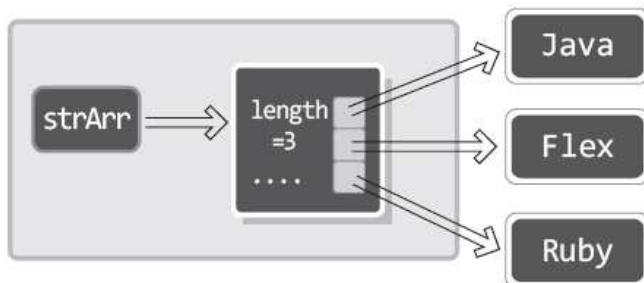
    int sum=arr[0]+arr[1]+arr[2];
    System.out.println(sum);
}
```

## 객체 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!



객체 배열에는 객체가 저장되는 것이 아니라,  
객체의 참조 값이 저장된다.



# 기본 자료형 배열 예제1

ArrayTest1.java

```
import java.util.Scanner;

public class ArrayTest1 {
    public static void main(String[] args) {
        int[] salary = new int[2];
        Scanner scan = new Scanner(System.in);
        System.out.print("직원1의 월급을 입력하시오: ");
        salary[0] = scan.nextInt();
        System.out.print("직원2의 월급을 입력하시오: ");
        salary[1] = scan.nextInt();
        System.out.println("직원1의 월급은 " + salary[0]);
        System.out.println("직원2의 월급은 " + salary[1]);
    }
}
```



## 기본 자료형 배열 예제2

```
import java.util.Scanner;

public class ArrayTest4 {
    public static void main(String[] args) {
        final int STUDENTS = 5;
        int total = 0;
        Scanner scan = new Scanner(System.in);
        int[] scores = new int[STUDENTS];
        for (int i = 0; i < STUDENTS; i++) {
            System.out.print("성적을 입력하시오:");
            scores[i] = scan.nextInt();
        }
        for (int i = 0; i < STUDENTS; i++)
            total += scores[i];
        System.out.println("평균 성적은" + total / STUDENTS + "입니다");
    }
}
```

### 실행결과

```
성적을 입력하시오:10
성적을 입력하시오:20
성적을 입력하시오:30
성적을 입력하시오:40
성적을 입력하시오:50
평균 성적은30입니다
```



# 객체 배열 예제

– CarTest.java

```
import java.util.Scanner;
class Car {
    public int speed // 속도
    public int mileage // 주행거리
    public String color // 색상

    public Car() {
        speed = mileage = 0;
        color = "red";
    }
    public void speedUp() { // 속도 증가 메소드
        speed += 10;
    }
    public String toString() { // 객체의 상태를 문자열로 반환하는 메소드
        return "속도: " + speed + " 주행거리: " + mileage + " 색상: " + color;
    }
}
```



# 객체 배열 예제

```
public class CarTest {  
    public static void main(String[] args) {  
        final int NUM_CARS = 5;  
        Car[] cars = new Car[NUM_CARS];  
        for (int i = 0; i < cars.length; i++)  
            cars[i] = new Car();  
        for (int i = 0; i < cars.length; i++)  
            cars[i].speedUp();  
        for (int i = 0; i < cars.length; i++)  
            System.out.println(cars[i].toString());  
    }  
}
```

## 실행결과

```
속도: 10 주행거리: 0 색상: red  
속도: 10 주행거리: 0 색상: red  
속도: 10 주행거리: 0 색상: red  
속도: 10 주행거리: 0 색상: red  
속도: 10 주행거리: 0 색상: red
```





## 또 다른 배열 선언 방법

- `int[] values;` // ① 자바 방식
- `int values[];` // ② C언어 유사 방식

배열의 크기가 들어가면 오류!

- `int[] values, grades;` // 2개의 배열 참조 변수 선언
- `int values, grades[];` // `grades`만 배열 참조 변수
- `int values[], grades[];` // `values, grades`는 배열 참조 변수





# 배열의 초기화

일반적인 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};
```

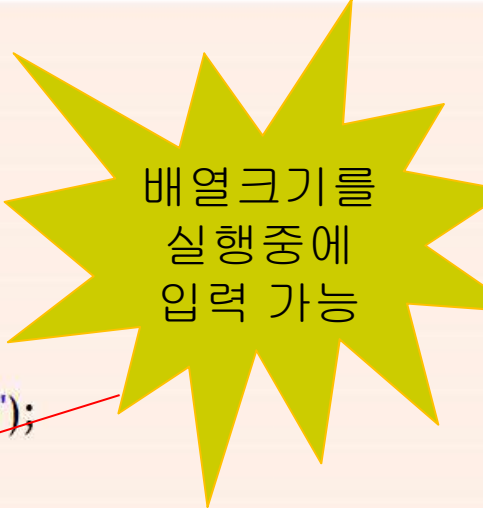
new를 사용하지 않아도 주어진 초기값 개수만큼 배열객체가 자동적으로 생성됨.



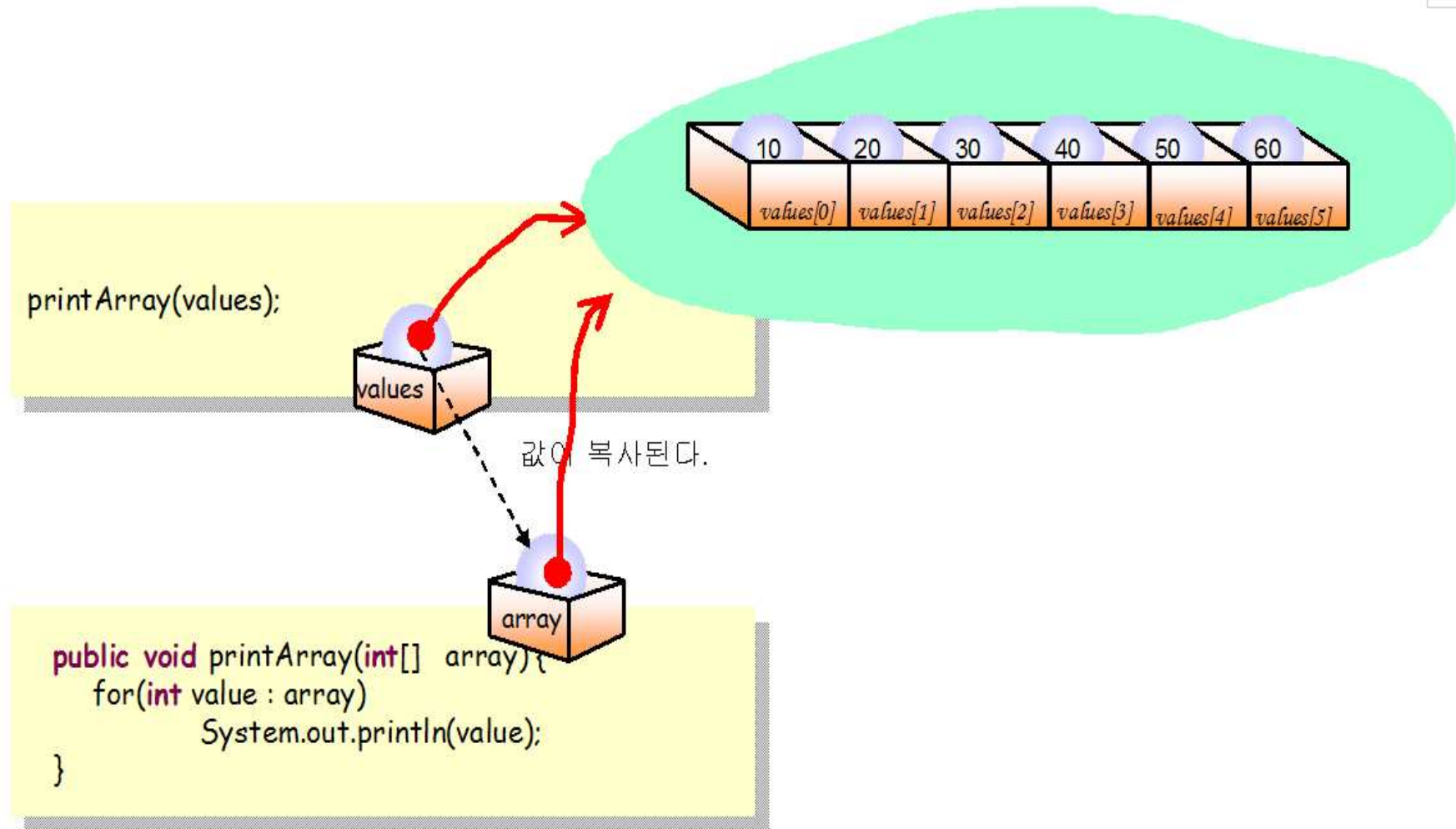
# 사용자가 배열의 크기를 지정

```
import java.util.Scanner;

public class ScoreTest {
    public static void main(String[] args) {
        int total = 0;
        int size;
        Scanner scan = new Scanner(System.in);
        System.out.print("배열의 크기를 입력하시오:");
        size = scan.nextInt();
        int[] scores = new int[size];
        for (int i = 0; i < scores.length; i++) {
            System.out.print("성적을 입력하시오:");
            scores[i] = scan.nextInt();
        }
        for (int i = 0; i < scores.length; i++)
            total += scores[i];
        System.out.println("평균 성적은" + total / scores.length + "입니다");
    }
}
```



# 배열을 메소드의 매개 변수로 전달





## 예제

```
import java.util.Scanner;

public class ScoreTest1 {
    final static int STUDENTS = 5;

    public static void main(String[] args) {
        int[] scores = new int[STUDENTS];
        getValues(scores);
        getAverage(scores);
    }

    private static void getValues(int[] array) {
        Scanner scan = new Scanner(System.in);
        for (int i = 0; i < array.length; i++) {
            System.out.print("성적을 입력하시오:");
            array[i] = scan.nextInt();
        }
    }
}
```



## 예제

```
private static void getAverage(int[] array) {  
    int total = 0;  
    for (int i = 0; i < array.length; i++)  
        total += array[i];  
    System.out.println("평균 성적은 " + total / array.length + "입니다");  
}  
}
```

### 실행결과

```
성적을 입력하시오:10  
성적을 입력하시오:20  
성적을 입력하시오:30  
성적을 입력하시오:40  
성적을 입력하시오:50  
평균 성적은 30입니다
```



# 메소드의 반환값으로 배열 반환

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        int[] array;
        array = getData();
        printData(array);
    }

    private static int[] getData() {
        int[] testData = { 10, 20, 30, 40, 50 };
        return testData;
    }

    private static void printData(int[] array) {
        for (int i = 0; i < array.length; i++)
            System.out.println(array[i]);
    }
}
```



# 명령줄 인수

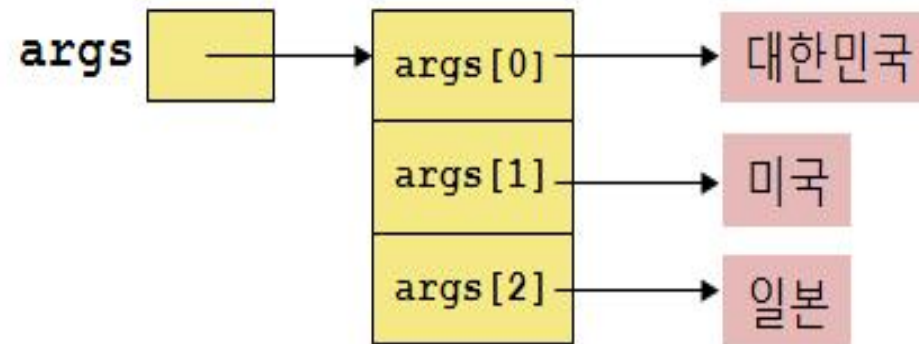
- 명령줄 인수(**command-line arguments**)
  - 프로그램을 실행시킬 때 명령줄에서 실행 프로그램 이름 뒤에 쓰는 정보들
  - 프로그램에서 처리할 정보(예를 들어 파일 이름들)를 전달
  - 이 이름들은 main 메소드에 **String** 배열 형태로 전달된다.
  - `public static void main(String[] args)`





```
1  /*****
2  * Nations.java
3  *
4  * 명령줄 인수를 받아 프린트하는 프로그램
5  *****/
6
7  public class Nations
8  {
9      public static void main(String[] args)
10     {
11         System.out.println("국가 리스트");
12         for (int n= 0; n < args.length; n++)
13             System.out.println(n+1 + "번째 국가: " + args[n]);
14     }
15 }
```

- > java Nations 대한민국 미국 일본







# 가변 길이 매개변수 리스트

- 호출할 때마다 처리할 데이터 개수가 달라지는 메소드를 생각해보자
- 예: 정수 매개변수들의 평균을 반환하는 `average`  
// 3개의 값의 평균 계산을 위한 호출  
`mean1 = average(82, 79, 67);`  
// 7개의 값의 평균 계산을 위한 호출  
`mean2 = average(65, 49, 93, 88, 77, 56, 95);`



# 가변 길이 매개변수 리스트

- (1) 중복정의(**overloading**)를 사용하여 여러 개의 메소드를 정의
  - 단점: 매개변수 개수에 대해 별도의 메소드 정의가 필요하다.
- (2) 정수 배열을 받아들이는 메소드를 정의한다.
  - 단점: 메소드 호출 전에 매번 배열을 생성하고 정수 값들을 배열에 저장한 후에 매개변수로 전달해야 한다.
- (3) 가변 길이 매개변수 리스트(**variable length parameter list**)
  - 동일 타입의 임의의 개수의 매개변수를 받는 메소드 정의

가변 길이 매개변수리스트를 나타냄

```
public double average (int ... list)
{
    // 본체 내용
}
```

↓

원소 타입      배열 이름

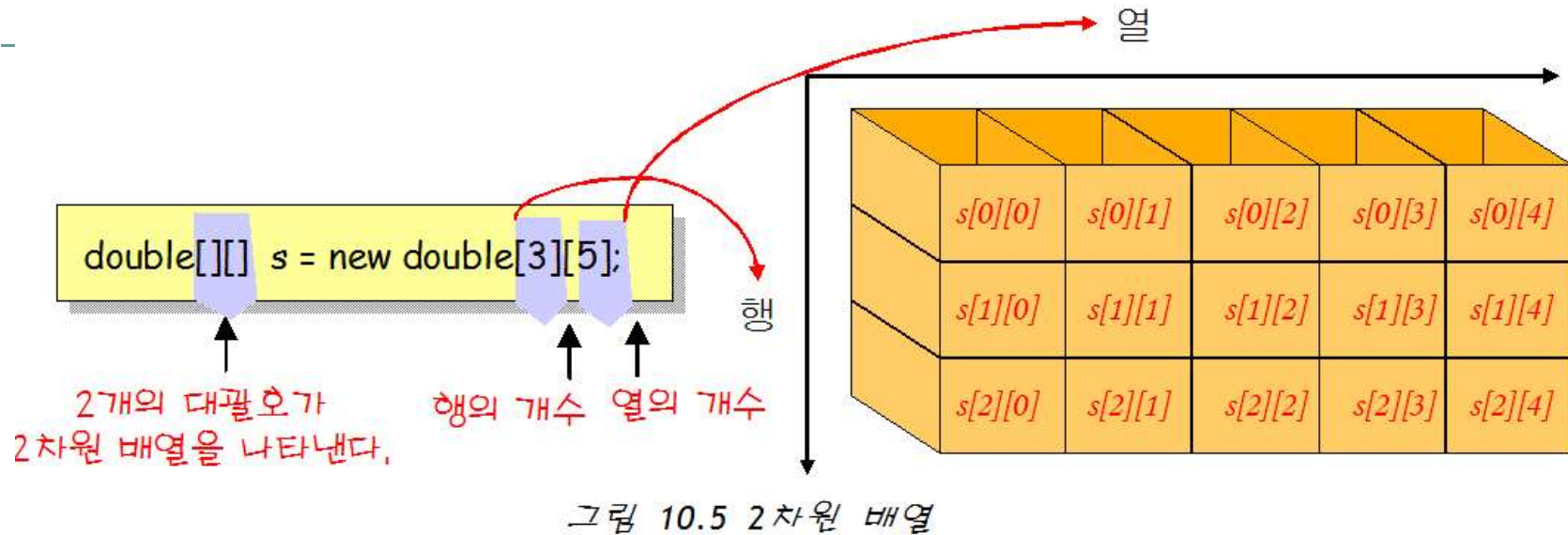


# 가변 길이 매개변수 리스트 예제

```
1 public class UtilityClass
2 {
3     /**
4      * Returns the largest of any number of int values.
5      */
6     public static int max(int... arg)
7     {
8         if (arg.length == 0)
9         {
10             System.out.println("Fatal Error: maximum of zero values.");
11             System.exit(0);
12         }
13
14         int largest = arg[0];
15         for (int i = 1; i < arg.length; i++)
16             if (arg[i] > largest)
17                 largest = arg[i];
18         return largest;
19     }
20 }
```

*This is the file UtilityClass.java.*

# 2차원 배열



```
for (int i=0 ; i < 3 ; i++)  
    for (int j=0 ; j < 5 ; j++)  
        System.out.println(s[i][j]);
```

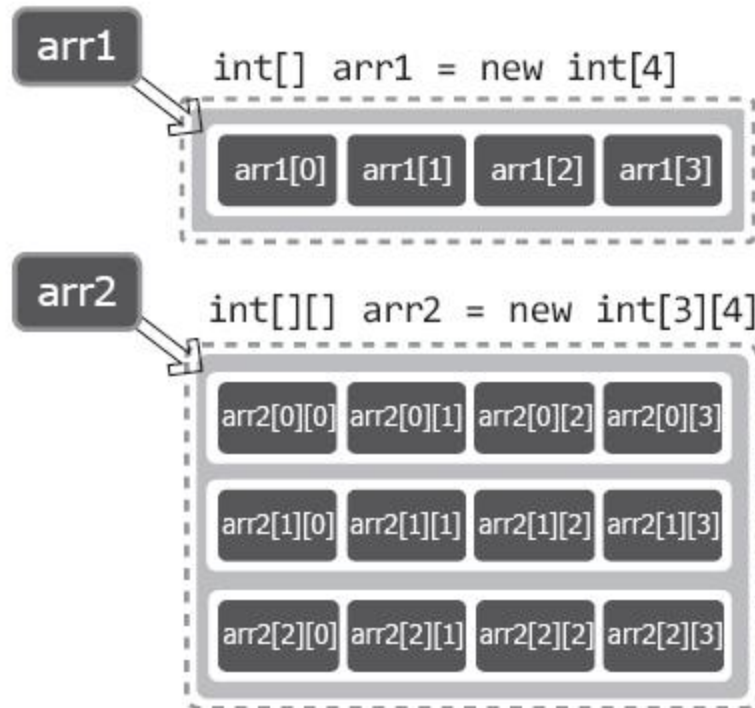
- 3차원 배열

```
double[][][] sales = new double[3][2][12];
```



# 1차원 배열 vs. 2차원 배열

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



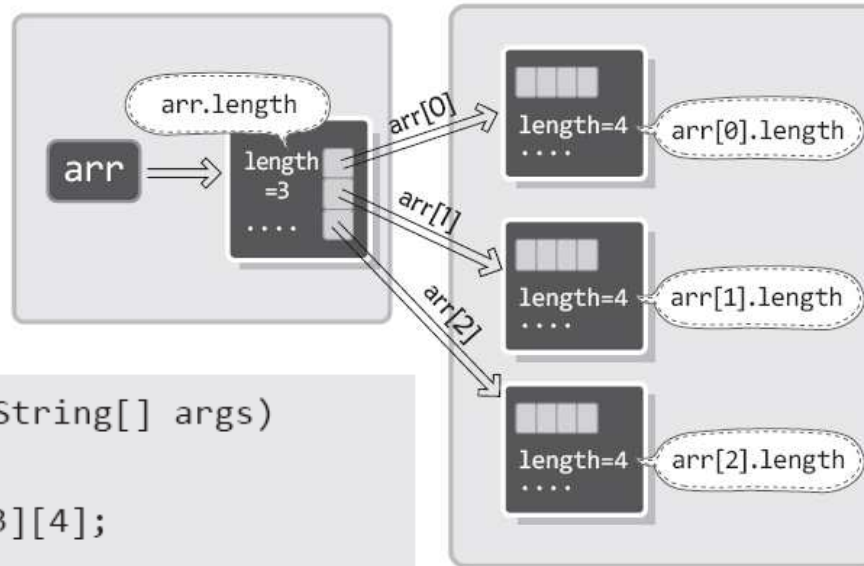
## 2차원 배열의 선언방법

- 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 배열 구조의 이해

2차원 배열의 메모리 구조 ▶



```
public static void main(String[] args)
{
    int[][] arr=new int[3][4];
    for(int i=0; i<arr.length; i++)
        for(int j=0; j<arr[i].length; j++)
            arr[i][j]=i+j;
    for(int i=0; i<arr.length; i++)
    {
        for(int j=0; j<arr[i].length; j++)
            System.out.print(arr[i][j]+" ");
        System.out.println("");
    }
}
```

이를 통해서 2차원 배열은 둘  
이상의 1차원 배열을 묶은  
형태라는 사실을 알 수 있다!

3년동안의 분기 강수량을 처리하여  
연도별 강수량을 출력하는 프로그램

```
import java.util.Scanner;

public class Rainfall {
    public static void main(String[] args) {
        final int YEARS = 3;
        final int QUARTERS = 4;
        double[][] rain = new double[YEARS][QUARTERS];
        Scanner scan = new Scanner(System.in);

        for (int y = 0; y < YEARS; y++){
            for (int q = 0; q < QUARTERS; q++) {
                System.out.print(y + "차년 도" + q + "분기 강수량: ");
                rain[y][q] = scan.nextDouble();
            }
        }

        for (int y = 0; y < YEARS; y++) {
            double total = 0.0;
            for (int q = 0; q < QUARTERS; q++) {
                total += rain[y][q];
            }
            System.out.println(y + "차년 도 강수량은 " + total);
        }
    }
}
```





# 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

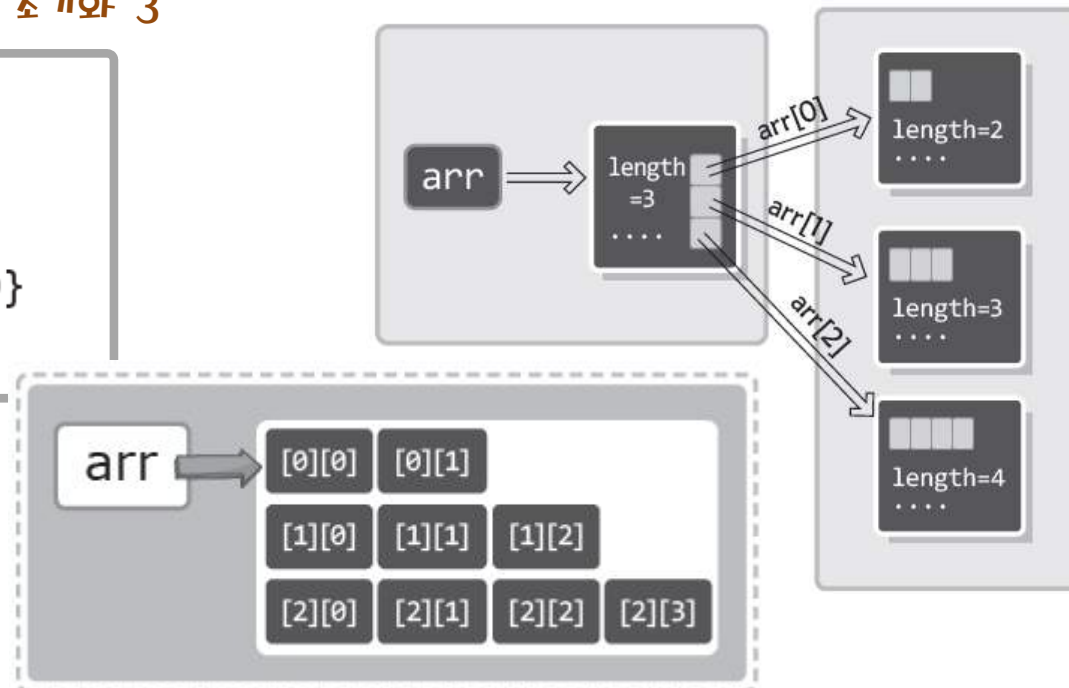
## 2차원 배열의 선언 및 초기화 2

```
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}  
};
```

톱니형(ragged) 배열





# 톱니형(ragged) 배열



- 각 행의 크기가 다른 2차원 배열

```
int[][] raggedArray = new int[3][]; // 2차원 배열이 부분적으로 생성
```

```
raggedArray[0] = new int[3]; // 첫번째 행을 생성
```

```
raggedArray[1] = new int[4]; // 두번째 행을 생성
```

```
raggedArray[2] = new int[5]; // 세번째 행을 생성
```

```
for (int i=0 ; i < raggedArray.length ; i++)
```

```
    System.out.println(i + “행의 길이는 “ + raggedArray[i].length);
```

톱니형 배열의 장점 : 메모리를 절약할 수 있다.

## 2차원 배열의 초기화 예제



ArrayTest.java

```
import java.util.Scanner;

public class ArrayTest {
    public static void main(String[] args) {
        int[][] array = { { 10, 20, 30, 40 }, { 50, 60, 70, 80 },
                          { 90, 100, 110, 120 } };

        for (int r = 0; r < array.length; r++) {
            for (int c = 0; c < array[r].length; c++) {
                System.out.println(r + "행" + c + "열:" + array[r][c]);
            }
        }
    }
}
```

배열의 행의 개수

배열 각 행의 열의 개수

실행결과

```
0행0열:10
0행1열:20
...
2행2열:110
2행3열:120
```



# for-each 루프

```
for (자료형 변수 : 배열이름)
{
    //반복 문장들
}
```

반복이 진행되면서 **자료형 변수**에  
배열 원소가 차례대로 대입됨

Numbers.java

```
public class Numbers {
    public static void main(String[] args) {
        int[] numbers = new int[5];
        for (int i = 0; i < numbers.length; i++)
            numbers[i] = (int) (Math.random()*1000);
        for (int value : numbers)
            System.out.println(value);
    }
}
```

실행결과

688  
773  
94  
691  
349



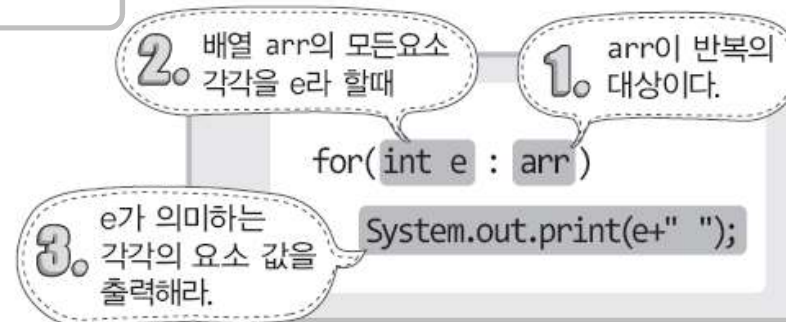
# for-each 루프의 이해

배열의 일부가 아닌, 배열 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-each 루프를 쓸 수 없는 경우

- 항상 for-each 루프를 사용할 수 있는 것은 아님
- 다음의 경우는 for-each 루프 사용 불가
  - 배열 원소의 값을 변경하는 경우
  - 역순으로 배열 원소를 처리하는 경우
  - 전체가 아닌 일부 원소만을 처리하는 경우
  - 하나의 반복루프에서 두개 이상의 배열을 처리하는 경우



# 예제

Strings2.java

```
public class Strings2 {  
    public static void main(String[] args) {  
        String[] strings = { "Java", "C", "C++" };  
        for (String s : strings)  
            System.out.println(s);  
    }  
}
```

실행결과

```
Java  
C  
C++
```



# ArrayList

---

- 배열의 단점
  - 크기가 한번 결정되면 고정되어 조정할 수 없다
- ArrayList
  - 이러한 단점을 보완하기 위한 동적인 자료구조
  - ArrayList에는 객체(정확히 말하면 객체에 대한 참조)를 저장할 수 있다.
  - 원소를 추가함에 따라 자동적으로 크기가 증가한다.

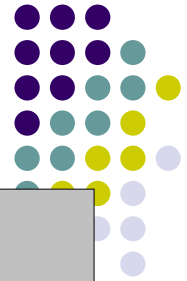
---

## Key Point

ArrayList는 동적으로 크기가 변하는 일종의 배열이다.

---

# ArrayList 메소드



메소드	설명
<code>public ArrayList()</code>	빈 리스트를 생성한다.
<code>public ArrayList(int initial)</code>	명시된 최초 용량을 갖는 리스트를 생성한다.
<code>void add(Object o)</code>	끝에 객체 <code>o</code> 를 원소로 추가한다.
<code>void add(int index, Object o)</code>	<code>index</code> 위치에 객체 <code>o</code> 를 원소로 삽입한다.
<code>void remove(int index)</code>	<code>index</code> 위치의 원소를 제거한다.
<code>void remove(Object o)</code>	객체 <code>o</code> 와 일치하는 첫 번째 원소를 삭제한다.
<code>Object get(int index)</code>	<code>index</code> 위치의 원소를 리턴한다.
<code>int size( )</code>	원소 개수를 리턴한다.
<code>int indexOf(Object o)</code>	객체 <code>o</code> 와 일치하는 첫 번째 원소의 인덱스 리턴
<code>void clear( )</code>	모든 원소를 제거한다.





# PhoneBook 클래스 작성

- ArrayList를 이용하여 PhoneBook 클래스 작성
  - ArrayList는 크기가 고정되어 있지 않음
  - `phoneBook = new ArrayList( );`
  - 크기는 ArrayList가 제공하는 `size` 메소드를 이용 가능
- PhoneBook 클래스의 `add` 메소드
  - Person 객체를 생성하고 ArrayList의 `add` 메소드를 이용하여 추가
  - `phoneBook.add(p);`
- PhoneBook 클래스의 `lookup` 메소드
  - ArrayList의 `get` 메소드를 이용하여 PhoneBook 내의 i-번째 Person 객체를 가져온다.
  - `p = (Person) PhoneBook.get(i);`

```

1  /*****
2  * PhoneBookList.java
3  *
4  * 전화번호부 클래스
5  *****/
6  import java.util.ArrayList;
7
8  /**
9  * ArrayList를 이용한 전화번호부 클래스 구현
10 */
11 class PhoneBookList
12 {
13     private ArrayList phoneBook;
14
15     public PhoneBookList( )
16     {
17         phoneBook = new ArrayList( );
18     }
19
20     public void add(String name, String phoneNumber)
21     {
22         Person p;
23
24         p = new Person(name, phoneNumber);
25         phoneBook.add(p);
26     }
27

```





```
28     public String lookup(String name)
29     {
30         Person p;
31
32         for (int i = 0; i < phoneBook.size(); i++)
33         {
34             p = (Person) phoneBook.get(i);
35             if (p.getName().equals(name))
36                 return p.getPhone();
37         }
38
39         return null;
40     }
41 }
```



# 배열의 응용: 정렬

- 선택 정렬: 최소값을 정렬되지 않은 첫번째 원소와 교환

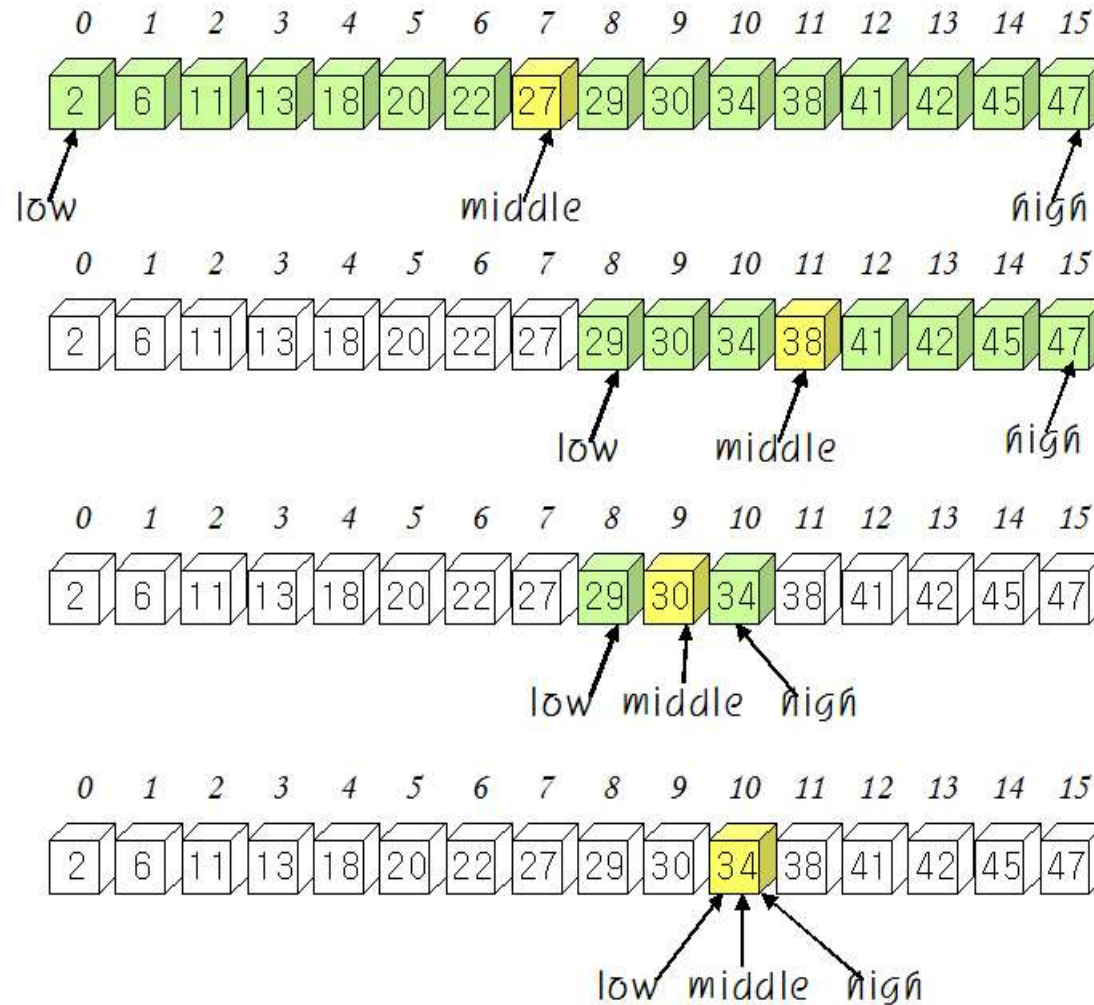


# 선택 정렬 코드



```
public static void selectionSort(int[] list) {  
    int temp, least;  
  
    for (int i = 0; i < list.length - 1; i++) {  
        least = i;  
        for (int j = i + 1; j < list.length; j++)  
            // 최소값 탐색  
            if (list[j] < list[least])  
                least = j;  
        // i번째 원소와 least 위치의 원소를 교환  
        temp = list[i];  
        list[i] = list[least];  
        list[least] = temp;  
    }  
}
```

# 배열의 응용: 이진 탐색





# 이진 탐색

*BinarySearch.java*

```
import java.util.Scanner;

public class BinarySearch {
    public static void main(String[] args) {
        int[] data = { 10, 20, 30, 40, 50, 60, 70, 80 };
        selectionSort(data);
        int retValue = binarySearch(data, 60);
        if (retValue != -1)
            System.out.println("위치 " + retValue + " 에서 발견");
        else
            System.out.println("탐색 실패");
    }

    public static void selectionSort(int list[]) {
        ...//앞의 코드 참조
    }
}
```



# 이진 탐색



```
public static int binarySearch(int[] list, int key) {  
    int low, high, middle;  
  
    low = 0;  
    high = list.length - 1;  
  
    while (low <= high) { // 아직 숫자들이 남아있으면  
        middle = (low + high) / 2; // 중간 요소 결정  
        if (key == list[middle]) // 일치하면 탐색 성공  
            return middle;  
        else if (key > list[middle]) // 중간 원소보다 크다면  
            low = middle + 1; // 새로운 값으로 low 설정  
        else  
            high = middle - 1; // 새로운 값으로 high 설정  
    }  
    return -1;  
}
```