CHAPTER 06

데이터를 일렬로 줄 세우는 배열

가장 기본적인 자료구조로, C 언어의 포인터, Java의 레퍼런스 개념 이해의 시작입니다.
아울러 언제 배열을 사용해야 하는지도 설명합니다.

배열(Array)은 두 가지의 중요한 의미를 포함합니다. 배열이라는 것은 모든 자료구조의 기본이 되기 때문에 여러 개의 자료를 처리하는 방법을 학습하기 위해서 알아야 할 필요가 있습니다. 배열이 중요한 두 번째 이유는 배열이라는 것이 여러분이 처음으로 익히는 일종의 포인터(Java에서는 Reference)라는 개념을 사용하기 때문입니다.

1 배열이란? 이름없는 변수들의 탑

프로그램을 작성하다 보면 여러 개의 데이터를 한 번에 계산하거나, 한 번에 루프를 이용해서 처리하는 경우가 종종 있는데 이럴 때는 여러 개의 변수를 선언하는 대신에 배열과 같은 자료구조를 활용합니다.

요리사가 때로는 여러 명이 먹을 수 있는 음식을 만들듯이 프로그래밍에서도 마찬가지입니다. 여러분이 어떤 많은 수의 데이터를 선언하려면 지금까지는 변수를 여러 번 선언해서 복잡하게 처리해야만 했습니다. 예를 들어 다섯 개의 데이터를 처리하려면 다섯 개의 변수가 필요했습니다.

코드

```
public class Ex1 {
      public static void main(String[] args) {
            int a = 10;
            int b = 20;
            int c = 30;
            int d = 40;
            int e = 50;
      }
}
```

만일 이 다섯 개의 데이터의 합을 구한다면 아마도 'int sum = a+b+c+d+e;'와 같이 처리할 겁니다.

1.1 이름 없이 데이터만 나열하는 배열

여러분이 위의 데이터 들의 합을 구한 후에 자리에서 일어나려고 할 때 누군가 여러분의 팔목을 잡을 수 있습니다. "아, 미안한데 사실은 데이터가 100개네요. 다시 좀 수정해주세요."라고 말한다면 우선은 화가 좀 나겠죠. "변수를 100개를 선언해야 한다?" 생각만 해도 아찔하군요. 그럼 그보다 더 많은 데이터를 처리하라고 한다면 어떻게 하시겠습니까? 매번 데이터 하나마다 변수를 선언하게 되면 다음과 같은 부분이 불편합니다.

- 변수마다 다른 이름을 붙여주어야 한다.
- 데이터가 추가될 때마다 변수의 선언 수도 변경되어야 하기 때문에 결과적으로 만든 프로그램을 매번 수정해야 할지도 모른다.

이런 불편을 해결해 주는 존재가 자료구조이고, 배열은 그 대표주자입니다.

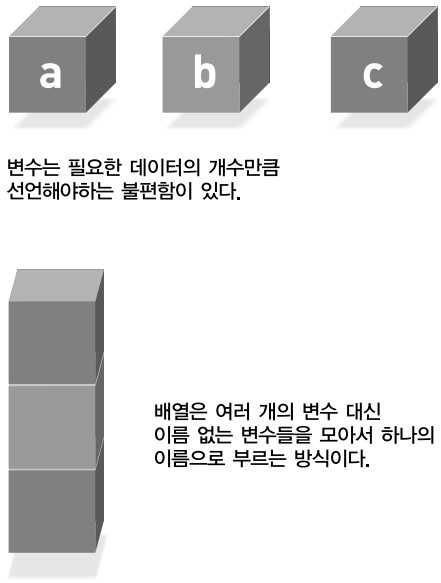


그림 1 배열은 이름 없는 변수들의 탑

1.2 데이터를 하나의 장소에 모으자.

변수를 하나의 상자로 생각해 보면 프로그램 안에서 너무 많은 상자를 만들면 각 상자에 이름(변수명)을 부여해야 하기 때문에 상당히 번거롭습니다. 대신에 아예 상자에 들어갈 만한 것들을 하나의 큰 상자에 데이터를 모아두기로 합니다.

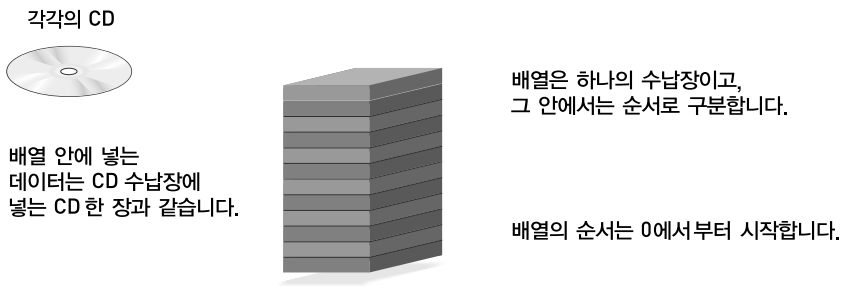


그림 2

배열과 그 안에 들어가는 데이터의 관계는 위의 그림처럼 CD와 CD를 저장해둔 수납장과 유사한 개념입니다. 앞에서 본 변수의 선언을 수정하여 다음과 같이 배열로 만들 수 있습니다.

예제

```
public class Ex1 {

      public static void main(String[] args) {

            int[] arr = {10,20,30,40,50};

      }
}
```

어떤가요? 모든 데이터를 변수로 선언하던 방식에 비하면 상당히 간결해지는 것을 알 수 있습니다. 지면 관계상 5개의 변수만 했지만, 100개의 데이터를 선언하는 경우를 생각해 보시면 좀 더 편하다는 느낌을 받으실 겁니다. "장난감은 장난감 상자에 모으고, CD는 CD 수납장에 모으는 것처럼 데이터를 하나의 공간에 같이 보관해 주자." 사실 배열의 시작은 이렇게 아주 단순했습니다.

■ **여러 개의 변수를 모을 때 [ ] 표시를 이용합니다.**

Java에서는 배열을 표시하기 위해 특별히 [ ] 기호를 선언해 줍니다. [ ] 기호의 의미는 "이것은 묶음 혹은 다발입니다."라는 신호와 같습니다. 즉, 위의 int[ ] arr 변수 선언은 데이터를 묶어준 수납장이나 공간을 의미한다는 겁니다.

2 배열의 문법적인 선언

배열의 개념은 모든 프로그래밍 언어에서 비슷하지만, 문법적인 표현이나 사용 방법은 언어마다 약간의 차이가 있습니다.

Java 언어에서의 배열은 보통 '자료의 타입[ ](배열 표시) arr(변수명) = 배열 선언'의 형태입니다. 아래 예에서 arr1이라는 변수는 int 타입의 배열을 의미합니다.

ex) int[ ] arr1 = {1, 2, 3};
int[ ] arr1;
arr1 = {1, 2, 3};

- 배열을 선언하는 동시에 배열 안의 내용물도 같이 선언하는 방식
- 배열을 선언해서 공간만 확보하고, 나중에 내용물을 채우는 방식

2.1 우선은 몇 개의 공간을 만들어야 하는지를 결정합니다.

배열을 만들려면 우선은 몇 개의 공간이 필요한지를 정확히 파악해주어야 합니다. 즉 이름 없는 변수가 몇 개나 필요한지 알아 두어야 한다는 겁니다. Java에서 배열을 선언할 때 주의할 점은 바로 배열을 처음 선언할 때 이런 크기를 지정해주어야 한다는 겁니다. 요즘 나오는 프로그래밍 언어 중에는 배열의 크기를 정하지 않아도 되는 언어들도 있지만, Java에서는 배열을 처음에 만들 때 반드시 크기를 정하고 만들어야만 합니다.

> Java로 배열을 선언할 때 가장 중요한 사실은 미리 배열의 크기를 결정해주어야만 한다는 사실입니다.

2.2 배열을 선언하는 법 ① 크기와 내용물을 동시에

우선 배열을 선언할 때 가장 간단한 방법은 처음 만드는 순간에 배열의 크기와 각 상자에 들어가는 내용물을 같이 넣어주는 겁니다.

```
int[ ] arr = {1, 3, 5};
```

만일 위와 같이 배열이 선언된다면 여러분은 int 변수 세 개가 1, 3, 5라는 값을 가지고 있고, 이 변수들이 쌓여 있다고 생각해주시면 됩니다.

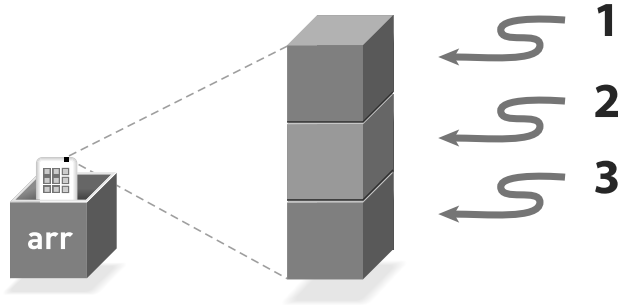


그림 3

그림에서 리모컨에 비유되는 개념이 나오기 시작하는 데 이것은 잠시 뒤에 설명하도록 하겠습니다.

2.3 배열을 선언하는 법 ② 크기만 먼저 정하고 내용물은 나중에

배열을 선언하는 두 번째 방법은 배열을 선언할 때에는 몇 개짜리 상자가 필요한지만 언급하고, 실제 데이터는 나중에 채우는 방식입니다.

예제

```
int[] arr = new int[5];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;
```

위의 코드를 보면 맨 위에서는 그냥 변수 다섯 개를 저장할 것이라는 선언만을 해두는 겁니다.

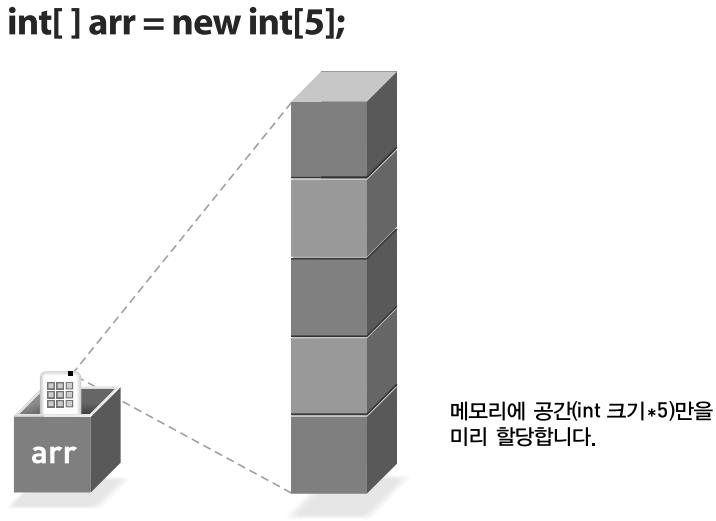


그림 4

그리고 아래 코드처럼 arr[0]과 같은 표기를 이용해서 변수에 값을 할당해줍니다.

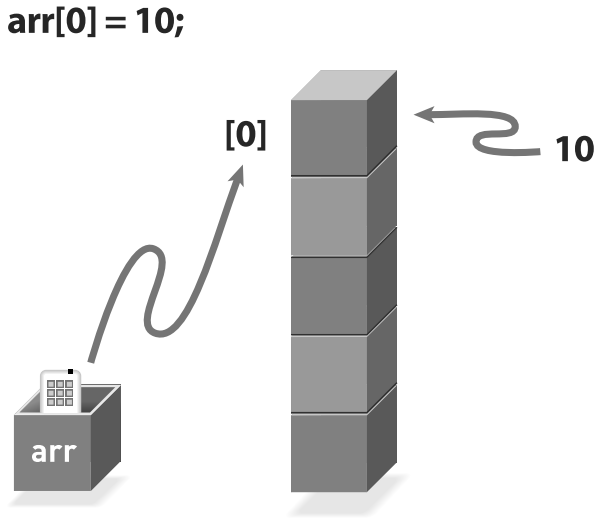


그림 5

2.4 배열을 선언한 변수와 인덱스 번호

배열이란 이름없는 변수들의 묶음이기 때문에 변수처럼 이름을 사용할 수 없고, 대신에 순번(Index)으로 변수를 찾고 사용합니다. 이때 사용하는 인덱스 번호는 0에서부터 시작합니다.

배열이란 결국은 이름이 없는 변수들이 쌓여 있는 형태이기 때문에 이 변수들을 이용하기 위해서는 기존의 변수를 이용하는 방식과는 다른 방법으로 접근해서 사용해야 합니다. 이런 방식의 힌트는 연극에서 역할 없는 배우들을 행인 1, 행인 2라 부르는 것에서 얻을 수 잇습니다. 즉 이름없는 배역에게 번호를 붙여 주는 것과 동일한 방식으로 사용합니다. 이렇게 이름이 없는 변수들을 사용하기 위해서 사용하는 숫자를 인덱스(Index) 번호라고 합니다. 일반적으로 거의 모든 프로그래밍에서 인덱스 번호는 0부터 시작합니다.

코드

```
int[] arr = new int[5];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;

System.out.println(arr[0]); ← 결과 10
System.out.println(arr[1]); ← 결과 20
```

위의 코드를 보면 System.out.println(arr[0]);을 이용하는 부분이 보입니다. 이것은 변수가 쌓여 있는 수납장에서 가장 위에 있는 변수를 가져다 쓰겠다고 말하는 것입니다. 따라서 위 코드의 실행 결과는 가장 위의 데이터인 10과 20이 출력됩니다. 변수를 사용하는 것과 배열의 인덱스 번호를 이용하는 것의 가장 중요한 포인트는 '아주 중요한 데이터'는 변수로 선언하는 것이고, 여러 개의 데이터를 한 번에 처리할 필요가 있을 때(예를 들어 루프를 돈다든가)에는 배열로 처리하는 것입니다.

■ **인덱스의 번호만 알면 어떤 데이터든지 알 수 있습니다.**

인덱스 번호를 이용해서 우리가 얻을 수 있는 진짜 장점은 바로 인덱스 번호만으로 데이터에 접근할 수 있다는 점입니다. 예를 들어 'int a = 10;', 'int b = 20;'과 같은 변수가 선언되어 있다면 프로그램을 만들 때 a와 b라는 변수명을 정확히 알아야만 데이터를 처리할 수 있습니다. 반면에 배열 내의 인덱스 번호를 이용하는 경우라면 [0], [1]과 같은 인덱스 번호로 데이터를 불러올 수 있습니다.

■ **인덱스 번호는 루프로 접근할 수 있습니다.**

배열에서 사용하는 인덱스 번호는 프로그램의 내부에서 인덱스 번호의 값을 변경하면서 접근이 가능하므로 주로 루프를 이용해서 모든 데이터에 접근할 수 있습니다.

예제 | 인덱스 번호는 연산할 수 있습니다.

```
public class Ex2 {
    public static void main(String[] args) {
```

```
        int[] arr = {10,20,30};
        System.out.println(arr[0]);
        System.out.println(arr[1]);
        System.out.println("------------------");
        int index = 0;

        System.out.println(arr[index++]); // 먼저 사용되고 증가하므로 0으로 사용된 다음 증가
        System.out.println(arr[index++]);
    }
}
```

```
10
20
------------------
10
20
```

코드를 보면 arr[0], arr[1]을 이용하는 방식을 이용할 때 직접 인덱스의 번호를 사용하고 있습니다. 조금 더 아래쪽에는 index라는 변수를 프로그램 내에서 자동으로 증가시키면서 배열 내의 데이터에 접근하고 있습니다. 어떤 변수를 이용해서 인덱스 번호를 마음대로 사용할 수 있다는 것은 결과적으로 루프와 같은 순환문을 이용할 수 있다는 것을 의미합니다.

예제 | 배열의 인덱스 번호는 루프에서 사용할 수 있습니다.

```
public class Ex2 {
    public static void main(String[] args) {
        int[] arr = {10,20,30};
        for(int i = 0; i < 3 ; i++){
            System.out.println(arr[i]);
        }
    }
}
```

```
10
20
30
```

위의 코드에서는 자세히 봐야 하는 부분이 몇 곳 있습니다. 우선은 루프를 돌릴 때 3이라는 숫자를 사용했다는 점입니다. 이렇게 되면 루프를 순환하는 동안 i의 값은 0, 1, 2까지 증가하게 될 겁니다. 배열이 세 칸짜리이기 때문에 인덱스 번호는 0에서 시작해서 2로 끝난다는 사실을 생각해

보면 인덱스 번호와 배열의 크기에는 1의 차이가 존재하는 것을 알 수 있습니다. 루프 안쪽에서는 arr[i] 코드가 이용되고 있습니다. 중요한 점은 배열 안에 있는 데이터에 접근할 때 인덱스의 번호를 특정한 변수로 이용해서 접근했다는 사실입니다. 만일 a, b와 같은 변수를 사용하는 경우라면 이런 코드는 불가능했을 겁니다. 인덱스의 번호를 변수를 이용해서 접근하는 방식은 모든 자료구조에서 사용되는 방식이기 때문에 반드시 잘 이해하고 있어야 합니다.

2.5 배열의 length는 배열의 길이

이제 배열에 대한 기본 문법으로 살펴볼 내용은 루프를 돌리는 코드를 수정 없이 사용하기 위해서 배열의 전체 길이를 알아내는 방법인 length를 이용하는 것을 설명할까 합니다.

예제 | 배열의 길이를 알고 싶을 때는 length

```
public class Ex3 {
      public static void main(String[] args) {
            int[] arr = {1,2,3,4,5,6,7,8,9,10}; // 10개의 변수가 쌓인 탑
            int len = arr.length; // 변수의 탑인 배열의 층수
            System.out.println(len);
      }
}
```

```
10
```

우선 코드의 내용은 전체 상자들의 개수를 알아보는 코드입니다. 배열에서 그 배열의 길이를 알고 싶다면 위의 코드에서 보는 것처럼 배열을 가리키는 '변수.length'라는 코드를 이용하면 됩니다. 배열의 공간 크기와 배열의 인덱스 번호와의 관계는 인덱스 번호가 0에서 시작하기 때문에 1의 차이가 발생합니다.

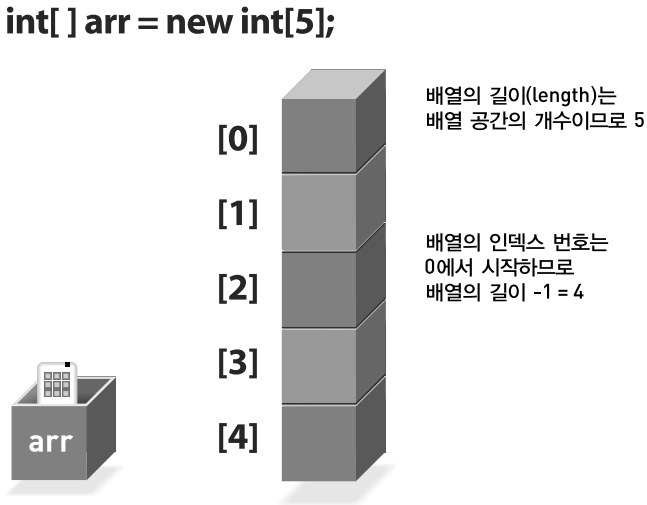


그림 6

위의 코드를 조금 더 수정해서 배열의 내용을 출력하는 코드를 만들어 보면 다음과 같습니다.

예제 | length는 주로 for 루프에서 자주 사용합니다.

```
public class Ex3 {
      public static void main(String[] args) {
            int[] arr = {1,2,3,4,5,6,7,8,9,10};
            int len = arr.length;

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

```
}
1
2
3
4
5
6
7
8
9
10
```

우선 배열의 공간의 크기를 알아내려고 'int len = arr.length'라는 코드를 활용했습니다. 지금 현재 배열의 길이가 10으로 출력되는 것은 위의 코드에서 설명했습니다. 코드의 아래쪽에서는 루프를 돌면서 배열 안에서 인덱스의 번호로 접근하는 방식을 보여주고 있습니다. 이 코드를 보면서 여러분이 생각해야 하는 것은 배열의 length 정보는 10이지만 실제로 배열의 루프에서 사용된 값은 0에서부터 9까지라는 사실입니다. 배열에서 가장 주의해야 할 것은 바로 이런 인덱스의 번호와 길이에 대한 문제입니다.

3 배열을 이용하는 프로그램을 만들어봅시다.

배열에서 반드시 알아야 하는 내용을 정리해 보자면 다음과 같습니다.

- 배열이라는 것은 이름을 가지지 못한 변수들이 쌓인 것이다.
- 배열 안의 특정 데이터에 접근하고 싶다면 인덱스 번호라는 것을 이용한다. 또한, 인덱스 번호를 이용하면 배열을 루프를 이용해서 순환할 때 사용하기 편리하다.
- 배열의 length를 이용하면 배열의 길이를 알 수 있다.

그럼 이제는 실제로 배열을 이용해서 데이터를 처리하는 예제를 만들어보도록 합시다.

3.1 1에서 10까지의 합을 구하는 또 다른 방법

앞에서 루프를 배울 때 1에서 10까지의 합을 구한 적이 있습니다.

예제 | 루프를 이용하여 합을 구하는 프로그램

```
public class SumEx1 {
      public static void main(String[] args) {
            int sum = 0;
            for(int i = 1; i <= 10; i++){
                  sum = sum + i;
                  //혹은
                  //sum += i;
            }
            System.out.println(sum);
      }
}
```

이제 위의 코드를 배열을 이용하게 수정해 볼까 합니다. 배열은 데이터를 보관하기 위해서 쓰이기 때문에 1에서 10이라는 데이터를 배열로 만들어주면 됩니다.

예제 | 배열을 이용하는 1에서 10의 합

```
public class SumEx2 {
      public static void main(String[] args) {
            int[] arr = {1,2,3,4,5,6,7,8,9,10};
            int sum = 0;
            for(int i = 0; i < arr.length; i++){
                  sum = sum + arr[i];
            }
            System.out.println(sum);
      }
}
```

코드를 보시면 우선은 int[ ] arr이라는 것을 선언했습니다. 그리고 그 배열 안에 1에서 10의 데이터를 변수처럼 선언해 두었습니다. 루프를 돌 때는 arr.length를 이용하고 있습니다. arr.length는 지금은 10입니다. 따라서 루프는 0에서부터 10 미만인 9까지 순환하게 됩니다. 이 번호가 인덱스 번호로 사용되면서 arr[0], arr[1], ..., arr[9]까지 사용되게 되는 겁니다. 1에서 10까지의 데이터만 가지고 배열을 쓰니 그냥 for 루프를 이용하는 것이 더 편리해 보일지도 모르겠습니다. 하지만, 배열인 경우에 데이터가 변경되면 어떻게 될까요?

3.2 배열을 시험 성적이라고 생각하시면 됩니다.

이제 arr 배열 안에 있는 데이터가 연속적인 값이 아니라 특정한 시험의 점수라고 생각해 보시면 어떨까요? 위의 코드에서 배열의 데이터를 아래와 같이 수정해보면 어떨까요?

코드 |

```
int[] arr = {100,82,73,64,55,46,37,28,19,10};
```

이렇게 수정해 주었지만, 나머지 코드는 수정하실 필요가 없습니다. 그냥 이전에 루프를 도는 것처럼 실행하시면 다음과 같은 결과를 얻을 수 있습니다.

루프라는 것은 정해진 방식대로 돌지만, 배열은 연속성이 전혀 없는 데이터를 한 번에 처리하기 편리할 수 있기 때문에 연속성이 없는 자료들을 처리할 때 유용하게 사용할 수 있습니다.

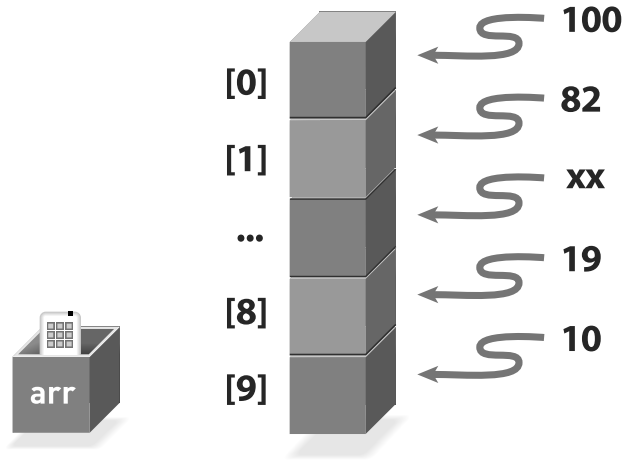


그림 7

■ **배열을 이용해서 시험 성적의 평균 점수도 구할 수 있을까요?**

위의 코드에서 배열을 이용해서 모든 데이터의 합계를 구했다면, 평균이라는 것은 합계/과목 수의 형태이므로 쉽게 처리할 수 있을 듯합니다.

예제 | 배열을 이용해서 합과 평균을 구하는 방법

```
public class SumEx2 {
	public static void main(String[] args) {
		int[] arr = {100,82,73,64,55,46,37,28,19,10};
		int sum = 0;

		for(int i = 0; i < arr.length; i++){
			sum = sum + arr[i];
		}
		System.out.println("총합:"+sum);
		//소수로 연산해야 한다.
		System.out.println("평균 : " + (sum/(float)arr.length));
```

```
	}
}
```

이제 여러분이 배열 안에 들어가는 내용만 바꿔주면 어떤 점수가 몇 개인지 상관없이 아래의 루프와 계산은 변경할 필요 없이 사용할 수 있게 되었습니다.

결과적으로 배열로 어떤 데이터들을 나열한다는 것은 루프를 이용할 가능성 99.9%라고 보면 됩니다. 비연속적이지만 모여진 데이터를 전체로 삼아 한 번에 처리해야 하는 경우가 바로 배열의 대상으로 가장 많이 선택되기 때문입니다.

3.3 배열에서 많이 실수하는 내용

배열을 공부하는 동안 가장 많이 실수하는 몇 가지의 사항에 대해서 알아보도록 합니다. 우선 가장 많이 실수하는 것은 배열이 3칸짜리인데 인덱스 번호 3으로 호출하는 경우와 같은 예입니다.

Java 언어에서는 배열이라는 것이 처음 선언 시에 메모리상에 크기를 어떤 형태로든 고정하기 때문에 인덱스 번호로 접근할 때 주의해야 합니다.

3.3.1 java.lang.ArrayIndexOutOfBoundsException: 배열의 크기를 넘어서는 경우

예제 배열의 범위를 벗어나면 예외(Exception)라는 것이 발생합니다.

```
public class Ex4 {
	public static void main(String[] args) {
		int[] arr = {1,2,3};
		System.out.println(arr[3]);
	}
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
	at Ex4.main(Ex4.java:8)
```

위의 코드를 실행하고 나면 'java.lang.ArrayIndexOutOfBoundsException'라는 에러가 발생합니다. 사실 정확히 말하면 에러라는 말보다는 예외(Exception)라고 표현해야만 합니다만, 여기

서는 그냥 에러라고 언급하겠습니다. 메시지는 'java.lang.ArrayIndexOutOfBoundsException' 인데 이 메시지를 유심히 살펴보면 배열(Array)의 인덱스(Index)가 범위를 벗어났다고(Out of Bounds) 말하고 있습니다. 즉, 위의 코드에서 사용된 배열은 3칸짜리 배열이기 때문에 이 배열의 안에 사용되는 인덱스의 번호도 0, 1, 2까지만 사용되어야 합니다. 그런데 실제 호출하는 부분에서는 arr[3]을 이용했습니다. 존재하지 않는 인덱스 번호의 데이터에 접근하려 한 상태라는 겁니다.

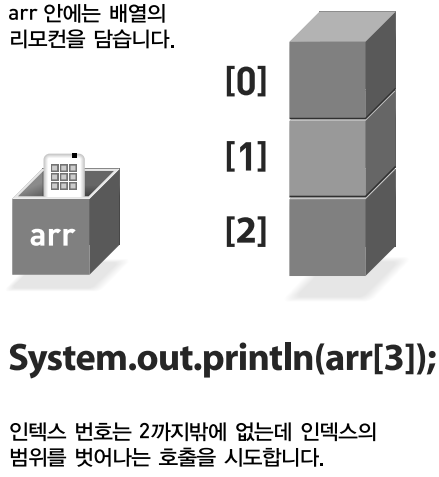


그림 8

3.3.2 배열은 초기화 시에 반드시 크기를 선언해주어야 합니다.

배열에 대해서 흔히들 하는 실수 중의 하나는 바로 배열을 초기화할 때 Java에서는(다른 언어는 조금 다릅니다만) 반드시 배열의 크기를 고정해 주어야 한다는 점입니다. 잘못된 배열 선언의 예를 보도록 합시다.

```
int[ ] arr = new int[ ]; ← 배열의 크기를 지정하지 않았다.
int[ ] arr = 10; ← 변수는 배열인데 실제로는 데이터를 지정하고 있다.
int[ ] arr = new int[1];
arr[1] = 20; ← 1칸짜리 배열에 2번째 칸을 요구하고 있다.
```

불행하게도 Java 언어에서의 배열은 고정형입니다. 즉 한번 크기를 정하고 나면 나중에 크기를 변경할 수 없습니다. 마지막의 두 구문을 보면 처음 배열을 선언할 때는 변수 1개의 공간의 필요

성 때문에 만들었지만, 두 번째 공간을 찾고 있습니다. 만일 배열이 가변적으로 마구 늘어나는 경우라면 이런 고민 없이 그냥 마음대로 처리하겠지만, Java 언어에서의 배열은 일단 한번 선언되고 나면 그 크기를 변경할 수 없기 때문에 항상 사용할 때 주의해야 합니다.

3.4 배열을 선언하는 기준 포인트

배열을 배우니 당연히 배열과 관련된 얘기를 하겠지만, 어떤 상황에서는 배열을 써야 하는지, 반대로 어떤 경우 변수를 써야 하는지에 대한 정확한 기준을 파악하는 것도 상당히 중요합니다. 가장 중요한 기준이 되는 내용은 아래와 같습니다.

- 변수로 선언하는 것은 중요해서 별도의 상자(변수)에 이름을 붙여서(변수명) 보관할 만한 것이다.
- 배열은 익명성(인덱스 번호)으로 대신해도 되는 데이터를 의미한다.

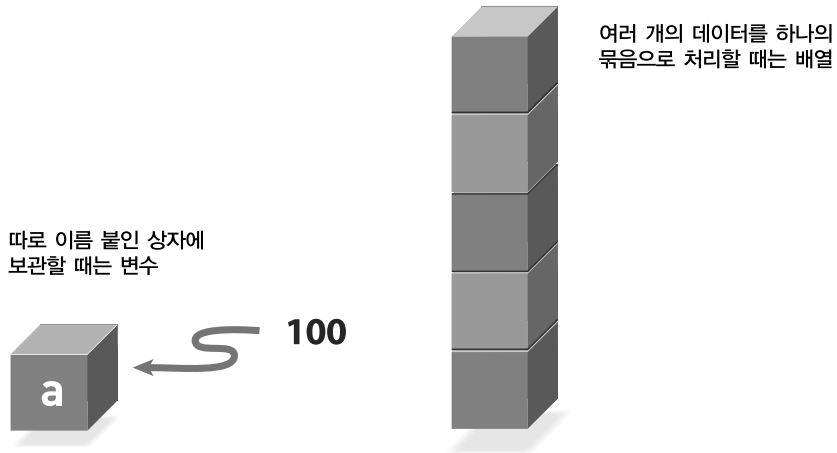


그림 9

■ **배열은 여러 개의 변수 선언을 대신해서 사용합니다.**

변수로 어떤 데이터를 선언한다는 것은 이 데이터가 프로그램의 내부에서는 적어도 조연급은 된다는 것을 의미합니다. 예를 들어 어떤 숫자들의 합을 구한다면 그 결과 데이터는 꽤 중요한 의미가 있기 때문에 변수로 선언해주는 겁니다. 반면에 배열이라는 상자 안에 들어가는 데이터는 그 정도의 중요성이 없는 겁니다. 그저 처리해야 하는 수많은 데이터 중의 하나이거나, 수많은 결과 데이터 중의 하나에 불과한 겁니다. 예를 들어 우리가 성적의 평균을 낸다고 하면 평균 성적 자체

는 변수로 선언할 만한 데이터이지만, 점수들의 경우는 '숫자도 많고, 많은 데이터 중 하나'의 의미만을 가집니다. 따라서 이런 데이터는 배열로 처리합니다.

■ **변수로 선언하기에는 모자라고 버리기엔 아깝고...**

성적을 처리하는 프로그램을 만들어야 한다면, 과목이 많을수록 변수로 선언해주기엔 너무나 일이 많습니다. 그렇다고 이 데이터를 버릴 수는 없는 상황입니다. 배열이라는 것은 이름이 없는 변수의 탑입니다. 이 탑에 이런 데이터를 보관해두는 겁니다.

4 배열에 대한 메모리 공간에 대한 이해

프로그램에서 배열이라는 것은 데이터를 저장하는 공간 자체를 의미합니다. 따라서 이제는 이 공간이라는 것이 프로그램이 실행되는 컴퓨터에서는 무얼 의미하는지 좀 더 자세히 알아보도록 합니다.

4.1 변수 하나로 데이터의 묶음(상자들)을 가리킨다.

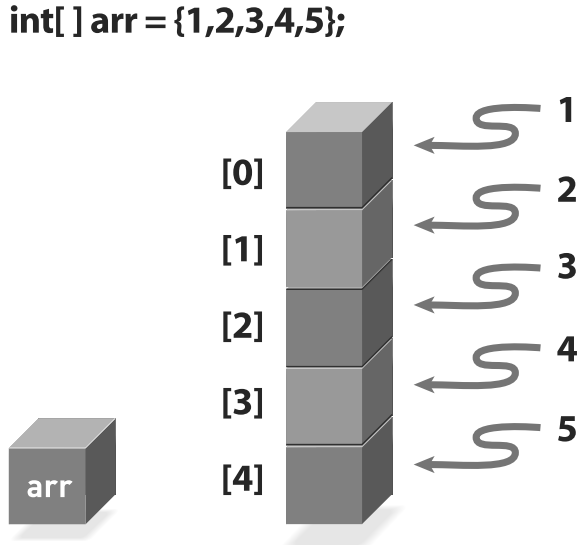


그림 10

배열은 이름(변수 선언)없는 데이터의 탑이기 때문에 위의 그림과 같은 형태로 만들어집니다. 그렇다면, 이런 데이터들을 담아둔 탑을 프로그램 내에서 어떻게 처리해야 할지 조금 고민해볼까 합니다.

■ **배열 자체를 변수처럼 복사해서 다른 변수에 옮긴다면?**

데이터를 여러 개 담는다는 것은 다른 의미로는 메모리에서 많은 공간을 차지한다는 것입니다.

> **int a = 10; (a라는 상자 안에 10이라는 데이터를 담는다.)**
> **int b = a; (b라는 상자 안에 a 상자의 내용물을 복사해서 담는다.)**

Java 언어에서의 '=' 연산은 무조건 복사입니다. 즉, 위의 코드에서처럼 a라는 상자의 내용물을 b라는 상자 안으로 넣는 형태라는 겁니다. 그럼 만일 a라는 상자가 엄청나게 큰 100MB의 데이터를 담은 변수라고 가정해 봅니다. 만일 변수 a의 상자 안의 내용물이 100MB이라면 'int b = a;' 구문을 처리하는 동안 100MB의 내용물을 꺼내서 다시 b라는 상자에 복사해서 넣어주게 됩니다. 실로 엄청나게 시간과 메모리를 많이 소모하는 작업이 될 것입니다. 더 큰 문제는 이런 커다란 상자를 복사하는 작업을 빈번하게 하면 메모리상에 엄청난 크기들의 데이터가 발생할 수 있기 때문에 프로그램의 실행 속도 역시 엄청 느려지게 될 겁니다.

코드

```
int[] arr = {100,82,73,64,55,46,37,28,19,10};
int[] arr2 = arr;
int[] arr3 = arr;
```

만일 위와 같은 코드를 만났다면 어떨까요? 변수의 할당이라는 것은 데이터의 복사라고 했던 것을 기억하실 겁니다. 따라서 위에서 arr이 40byte라면 벌써 위의 코드만으로 120byte의 메모리를 사용하게 될 겁니다.

만일 배열과 같이 큰 용량의 데이터를 그대로 복사하게
된다면 엄청나게 많은 메모리를 사용하게 됩니다.

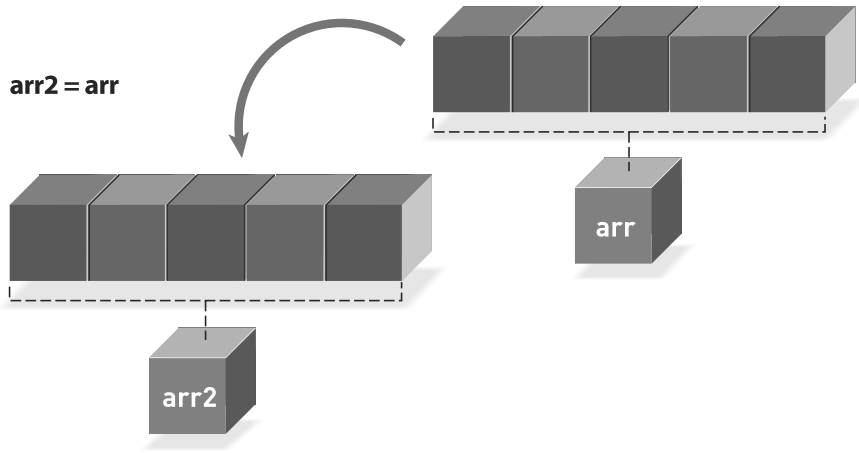


그림 11

■ **명심하세요! 배열을 의미하는 변수 상자는 좀 특별합니다.**

배열을 의미하는 변수가 이렇게 모든 데이터를 가지고 있다면 엄청나게 많은 메모리를 사용하는 프로그램을 작성하게 될 겁니다. 이 문제는 비단 Java만의 문제는 아닙니다. C 언어에서도 마찬가지입니다. 그렇다면, 프로그램의 속도를 높여주기 위해서 뭔가 다른 장치가 필요할 겁니다.

4.2 배열을 담는 변수는 좀 특별합니다.

```
int a = 10;
int b = a;
```

위의 코드를 보면 a라는 상자에 10이라는 값을 담고, b라는 상자에 a 상자의 내용물을 복사하는 방식입니다. 자 그럼 배열일 경우에는 어떨까요?

```
int[ ] arr = {10,20,30};
```

■ System.out.println으로 출력해봅시다.

우선 우리가 가장 빈번하게 사용했던 System.out.println( )을 이용해서 출력해보도록 합니다.

예제

```
public class Ex2 {
      public static void main(String[] args) {
            int[] arr = {10,20,30};
            System.out.println(arr);
      }
}
```

```
[I@de6ced ← 실행된 결과는 매번 달라질 수 있습니다.
```

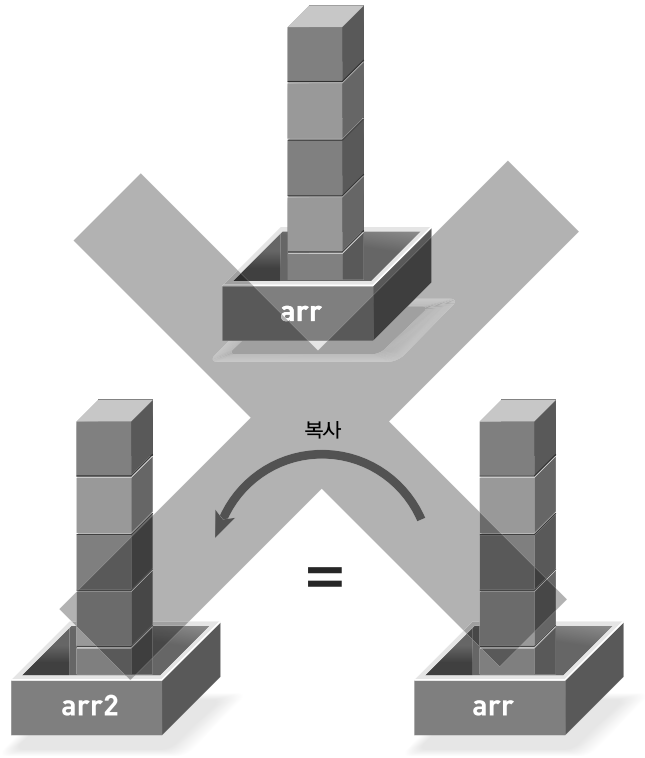
출력해본 결과를 보니 조금 이상합니다. 대체 이 의미는 무엇일까요?

4.3 배열은 너무 크기 때문에 변수(상자)에 진짜 값이 아니라 리모컨만 넣습니다.

기본 자료형의 변수는 상자 안에 데이터를 담고 있다면, 배열은 많은 데이터를 의미하기 때문에 배열을 의미하는 변수에 배열의 모든 데이터를 넣지 않습니다. 대신에 나중에 메모리상에 있는 배열을 사용할 수 있는 일종의 리모컨을 넣어주는 방식입니다. 만일 배열을 의미하는 변수가 기존의 기본 자료형처럼 처리된다면 어떤 일이 생길지 생각해 볼까요?

```
int[ ] arr = 100개의 숫자를 담는 배열;
int[ ] arr2 = arr;
```

만일 배열이 그 안에 데이터를 담는 방식이라면 =(할당 연산)이 일어날 때 100 * 4(int의 크기)의 데이터를 arr2 안으로 복사해서 넣어주게 됩니다.



배열과 같이 많은 데이터를 가진 변수를 복사하면 작업량이 너무 많아집니다.

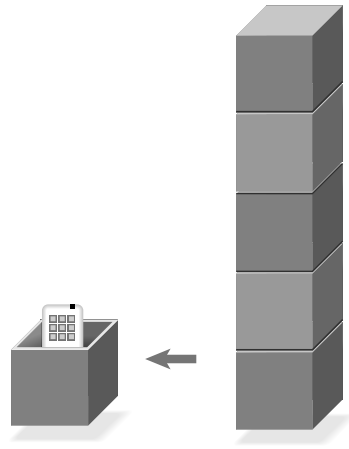
그림 12 배열과 리모컨

Java에서 변수의 할당이 복사라는 방식이므로 실제로 arr이라는 변수에 100개의 데이터가 있다면 복사를 하는 시간도 많이 걸릴 수밖에 없고, 복사를 하는 동안 메모리도 많이 사용하게 됩니다. 만일 arr에 담겨 있는 데이터가 수 만개의 데이터라고 생각해 보면 이런 방식의 문제를 간과할 수는 없습니다. 데이터를 복사하게 되면 발생하는 이런 문제를 해결하는 방법으로는 C 언어에서 사용하는 포인터(Pointer)라는 개념을 이용합니다. 이 방식은 실제 데이터를 메모리상에 다른 공간에 만들어 두고, 변수 안에는 메모리상에 있는 데이터를 찾아갈 수 있는 기호만을 넣어주는 방식입니다. 이 책에서는 이런 기호를 리모컨으로 비유하도록 하겠습니다. 실체는 조금 떨어진 곳에 있지만, 리모컨을 누르면 실체가 동작하는 것처럼 변수에 리모컨을 넣어주면 변수를 통해서 떨어져 있는 실체를 움직일 수 있기 때문입니다. 이렇게 리모컨이 들어가는 가장 큰 이유는 바로 다음과 같은 연산 작업을 할 때를 대비해서입니다.

코드 | 배열을 이용하는 변수의 할당은 리모컨의 복사입니다.

```
int[] arr = {100,82,73,64,55,46,37,28,19,10};
int[] arr2 = arr;
int[] arr3 = arr;
```

위의 코드를 어떤 방식으로 해석하는지 정확하게 아실 필요가 있습니다.



int[ ] arr2 = arr;　　실제 데이터

배열을 나타내는 변수에는 배열의 리모컨만 담습니다.

그림 13

그림을 보면 실제 데이터는 메모리상에 조금 떨어진 곳에 만들어지고, 변수 arr 안에는 실제 데이터를 움직이는 리모컨을 넣어줍니다.

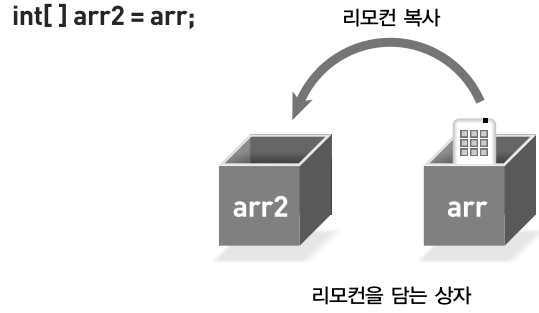
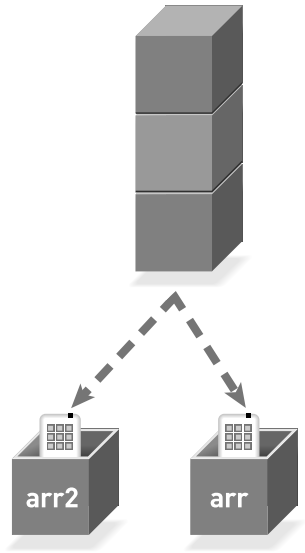


그림 14

int[ ] arr2 = arr; 이라는 코드를 실행하게 되면 arr이라는 변수에 담겨 있는 것은 실제 데이터가 아닌 리모컨이기 때문에, 복사가 되는 것 역시 리모컨 자체가 복사되는 방식으로 동작합니다.



복사된 리모컨은 같은 공간을 가리킨다.

그림 15

복사가 된 후의 결과를 보면 기존의 변수 arr 안에는 실제 데이터의 리모컨이 들어 있게 되고, arr2 변수에는 복사된 리모컨을 담게 됩니다. 따라서 arr 변수나 arr2 변수 모두 들어 있는 리모컨은 동일한 데이터를 움직이는 리모컨이 됩니다. 이렇게 메모리를 많이 차지하는 데이터를 리모컨 형태로 복사하게 되면 '=' 연산 작업을 할 때 리모컨만 복사되므로 처리되는 데이터의 양도 적어지고, 속도 역시 좋아집니다. 또한, 동일한 데이터를 다시 메모리상에 만들지 않기 때문에 결과적으로 메모리를 더욱 효과적으로 사용하게 됩니다. 다음의 예제를 보면서 다시 한번 정리해보도록 합니다.

예제 | 배열의 변수는 리모컨입니다.

```
int[] arr = {100,82,73,64,55,46,37,28,19,10};

int[] arr2 = arr;

int[] arr3 = arr;

System.out.println("arr: "+arr);
System.out.println("arr2: "+arr2);
System.out.println("arr3: "+arr3);
```

```
arr: [I@de6ced ← @ 이후의 값은 매번 달라질 수 있습니다.
arr2: [I@de6ced
arr3: [I@de6ced
```

위 코드의 결과를 보면 세 개의 변수가 모두 동일한 결괏값을 출력하는 것을 볼 수 있습니다. 원본 배열을 복사했지만 실제로 arr이라는 변수 안에 들어 있는 존재는 실제 데이터가 아닌 리모컨만 복사되었기 때문에 arr2, arr3 역시 동일한 리모컨이 담겨 있는 것을 확인할 수 있습니다. Java에서는 이런 리모컨의 존재를 레퍼런스(Reference)라 부르고 C 언어에서는 포인터라고 합니다. 잠시 뒤에 본격적으로 설명하도록 합니다.

4.4 변수 안에는 데이터인가? 레퍼런스인가?

좀 동떨어진 얘기 같지만, 자동차의 내비게이션이 보편화 되면서 길을 못 찾아서 헤매는 경우는 엄청나게 줄었습니다. 그저 내비게이션이 정해주는 길로 운행하면 목적지에 도착할 수가 있습니다. 친구들에게 주소만 알려주면 내비게이션이 알아서 길을 찾아주는 세상이 되었습니다. 또한, 내가 맘에 드는 상품에 대한 정보를 상대방에게 이미지로 보내지 않아도, 그냥 인터넷 주소만 전송해 주어도 상대방은 내가 원하는 상품을 알 수도 있습니다.

프로그래밍에서는 이렇게 어떤 식별할 수 있는 위치 정보를 이용해서 원하는 대상을 찾는 개념이 훨씬 전부터 적용되어 있다고 생각하시면 됩니다. 실제로 큰 덩어리의 데이터를 메모리에 만들어 두고 상자에는 그 데이터가 어디에 있는지에 대한 위치 정보만을 담아두는 겁니다.

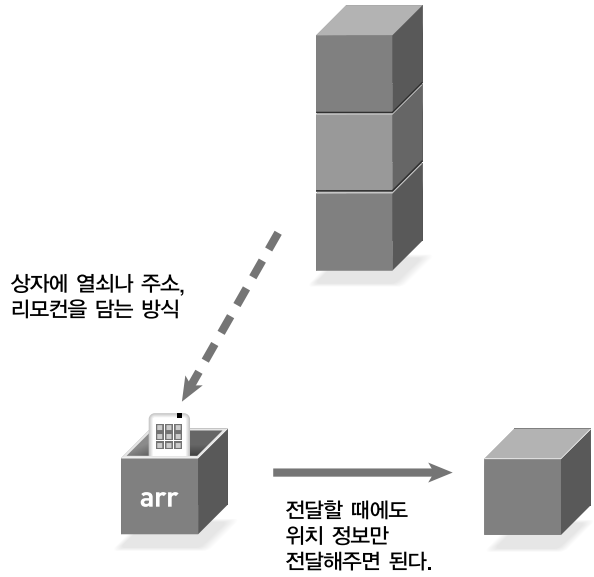


그림 16 배열과 위치 정보

4.5 Java에서는 레퍼런스(Reference)라고 합니다: 리모컨이라고 생각하세요.

'int[ ] arr = {1, 2, 3}'에서 arr이라는 변수에는 실제로 메모리에 만들어진 변수의 탑을 찾을 수 있는 리모컨을 넣는 방식을 사용합니다. 이때 들어가는 리모컨을 전문 용어로 '레퍼런스(Reference)'라고 합니다. 따라서 'int[ ] arr = {1, 2, 3};'이라는 코드를 해석할 때에는 "arr 변수는 {1,2,3} 배열의 레퍼런스를 담고 있다."라고 해석합니다.

■ @가 나오면 레퍼런스가 담겨 있다고 생각하세요.

앞에서 배열을 System.out.println( )으로 출력했을 때 나오는 결과를 다시 봅시다.

```
arr: [I@de6ced ← '@' 뒤의 값은 달라질 수 있습니다.
arr2: [I@de6ced
arr3: [I@de6ced
```

중간에 '@' 기호가 나오는 것이 보입니다. 이제부터는 어떤 데이터를 출력했을 때 '@'가 출력되면 이 변수는 값을 진짜로 저장한 것이 아니라 데이터가 어디에 있는지 알고 있고, 이 데이터를 사용할 수 있는 리모컨을 저장해두었다고 생각하시면 됩니다. 다만, 여러분에게 지금 설명하지 못하

는 내용이 하나 더 있는데, 그것은 '@' 이하에 나오는 숫자(16진수)가 실제 메모리의 주솟값은 아니라는 겁니다. Java에서는 이런 데이터를 출력할 때 hashCode( )라는 특정한 함수의 결괏값을 사용합니다. 이에 대해서는 객체를 좀 더 깊이 공부한 다음 설명하도록 하고 여기서는 어떤 변수를 출력했을 때 '@'가 나오면 "아. 이 변수는 리모컨이 담겨 있구나"라고 생각하시면 됩니다.

5 배열은 비연속적인 정보들을 하나로 모으고 싶을 때 씁니다.

이제 배열을 이용하는 문제를 하나 만들어 볼 차례가 되었습니다. 예를 들어 여러분이 입력하는 숫자들을 배열에 저장하고 이 데이터를 이용해서 평균 점수와 최고 점수, 최저 점수를 구하는 것을 생각해볼까 합니다. Java에서는 데이터를 보관하는 배열을 만들 때에는 반드시 처음에 몇 개의 변수 공간이 필요한지를 알려주어야 한다는 사실을 기억해 주시기 바랍니다. 이번 예제에서는 5개의 공간에 점수를 입력할 수 있게 할 예정입니다. 즉 5개의 공간에 점수를 넣는다는 뜻입니다.

예제 점수 계산을 위한 프로그램 : 배열만 선언한 모습

```
public class GradeEx {
      public static void main(String[] args) {
            int[] grades = new int[5];
      }
}
```

위의 코드를 보면 배열을 선언할 때 데이터가 정해지지 않았기 때문에 크기만 지정해 둔 배열을 선언해둔 것이 보입니다. 즉, 위의 grades라는 변수는 int 변수 5개를 쌓아둔 탑의 위치 정보(리모컨)를 가지고 있습니다.

■ Scanner와 인덱스 번호를 이용해서 데이터를 넣을 수 있게 합니다.

이제 변수의 탑에 실제로 데이터를 넣어주기 위해서 Scanner와 인덱스 번호를 이용하도록 합니다.

예제 | 사용자가 입력한 데이터를 배열의 특정 위치에 넣는 코드

```
import java.util.Scanner;

public class GradeEx {
      public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            int[] grades = new int[5];

            for(int i = 0; i < grades.length; i++){
                  System.out.println(i  + " 성적 점수를 넣어주세요.");
                  int userInput = scanner.nextInt();
                  grades[i] = userInput;
            }//end for
      }
}
```

```
0 성적 점수를 넣어주세요.
10
1 성적 점수를 넣어주세요.
20
2 성적 점수를 넣어주세요.
30
3 성적 점수를 넣어주세요.
40
4 성적 점수를 넣어주세요.
50
```

코드를 보면 중간에 루프를 도는 것이 보입니다. 배열은 인덱스 번호를 통해서 접근할 수 있기 때문에 위의 코드는 for 루프를 돌면서 배열 안의 각 각의 데이터를 채우게 됩니다.

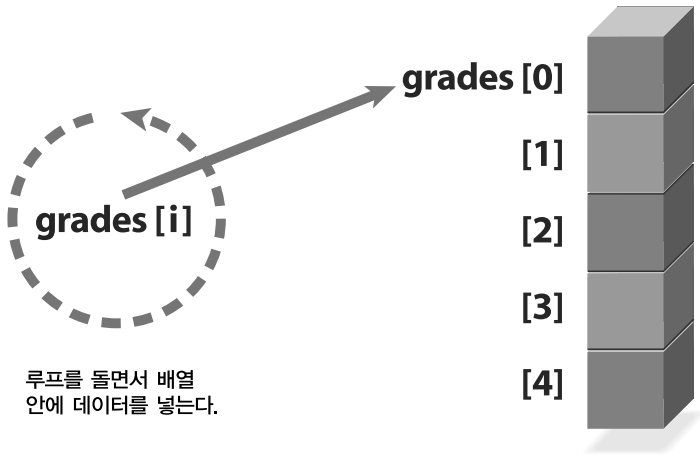


그림 17 배열과 연속적인 데이터 담기

■ Arrays.toString(배열): 배열 안에 있는 내용을 확인하고 싶을 때

위의 코드를 보면 데이터를 배열 안으로 넣어주긴 했지만, 실제로 데이터가 들어간 모습이 보이지 않아서 좀 갑갑합니다. 따라서 간단하게 배열 안쪽에 들어가 있는 상자들의 내용물을 엿볼 수 있는 방법을 사용해보도록 합니다.

예제 | 배열 안의 내용물을 보는 방법

```
import java.util.Arrays; // 반드시 추가해주어야 합니다.
import java.util.Scanner;

public class GradeEx {
      public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            int[] grades = new int[5];

            for(int i = 0; i < grades.length; i++){
                  System.out.println(i  + " 성적 점수를 넣어주세요.");
                  int userInput = scanner.nextInt();
                  grades[i] = userInput;
            }//end for

            System.out.println("처리할 점수들은 다음과 같습니다.");
            System.out.println(Arrays.toString(grades));
      }
}
```

Arrays.toString(배열을 가리키는 변수)을 이용하면 배열 안에 있는 내용물을 문자열로 변경해 줍니다. System.out.println( )과 같이 결합해서 사용해 주면 배열 안에 있는 데이터를 루프를 이용하지 않고도 볼 수 있습니다. 다만, 반드시 'import java.util.Arrays;'라는 코드를 추가해주어야 합니다.

```
0 성적 점수를 넣어주세요.
10
1 성적 점수를 넣어주세요.
20
2 성적 점수를 넣어주세요.
30
3 성적 점수를 넣어주세요.
40
4 성적 점수를 넣어주세요.
50
처리할 점수들은 다음과 같습니다.
[10, 20, 30, 40, 50]
```

만일 입력된 데이터의 평균을 구하려면 length를 활용하면 됩니다.

■ **배열의 길이: 평균은 '배열의 모든 내용의 합/데이터 수'로 구합니다.**

코드

```
//평균점수를 구한다.
int sum = 0;
for(int i = 0; i < grades.length; i++){
      sum = sum + grades[i];
}
System.out.println("평균 : " + (  sum / (float)grades.length)  );
```

■ **최고 점수는 기존 점수보다 큰 경우에만 바꿔치기합니다.**

이제 조금 더 배열을 이용하면서 어떤 점수가 기존의 최고 점수보다 큰 점수일 때 최고 점수 데이터를 현재 점수로 변경하는 예제를 만들어 보도록 합니다. 예를 들어 시작할 때 최고 점수는 0이라고 가정해 봅니다. 루프의 데이터가 {10, 20, 50, 30, 40}이라고 가정해 보면 10점은 최고 점수 0보다 크기 때문에 최고 점수를 10으로 변경해야 합니다. 20인 경우에는 현재 최고 점수 10보다 크기 때문에 변경합니다. 최고 점수가 50이 되면 뒤에 나오는 데이터는 최고 점수보다 크지 않기

때문에 변경되지 않도록 하는 방법을 사용하면 됩니다.

코드

```
//최고 점수를 구한다.
int max = 0;
for(int i = 0; i < grades.length; i++){
      if(grades[i] > max){
            max = grades[i];
      }
}
System.out.println("최고 점수 : " + max);
```

■ **최저 점수는 어떤 값보다 작을 때만 갱신하도록 합니다.**

최저 점수를 구하는 방법은 최고 점수와는 반대입니다. 즉 최초에 어떤 큰 값을 주는 겁니다. 그리고 그 값보다 작을 때에만 최저 점수를 바꿔주는 겁니다. 물론 위에서 구한 최고 점수를 이용하는 것이 가장 좋겠지만, 지금은 개념적인 설명이니까 임의의 숫자를 정해서 처리하도록 하겠습니다.

코드

```
//최저 점수를 구한다.
int min = 1000;
for(int i = 0; i < grades.length; i++){
      if(grades[i] < min){
            min = grades[i];
      }
}
System.out.println("최저 점수 : " + min);
```

이제 모든 프로그램이 완성된 형태는 다음과 같습니다.

예제 | 배열을 이용한 평균, 최고, 최저 점수 구하기

```
import java.util.Arrays;
import java.util.Scanner;

public class GradeEx {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		int[] grades = new int[5];

		for(int i = 0; i < grades.length; i++){
			System.out.println(i  + " 성적 점수를 넣어주세요.");
			int userInput = scanner.nextInt();
			grades[i] = userInput;
		}//end for

		System.out.println("처리할 점수들은 다음과 같습니다.");
		System.out.println(Arrays.toString(grades));

		//평균점수를 구한다.
		int sum = 0;
		for(int i = 0; i < grades.length; i++){
			sum = sum + grades[i];
		}
		System.out.println("평균 : " + (sum/(float)grades.length));


		//최고 점수를 구한다.
		int max = 0;
		for(int i = 0; i < grades.length; i++){
			if(grades[i] > max){
				max = grades[i];
			}
		}
		System.out.println("최고 점수 : " + max);

		//최저 점수를 구한다.
		int min = 1000;
		for(int i = 0; i < grades.length; i++){
			if(grades[i] < min){
				min = grades[i];
			}
		}
		System.out.println("최저 점수 : " + min);
```

```
        }
}
```

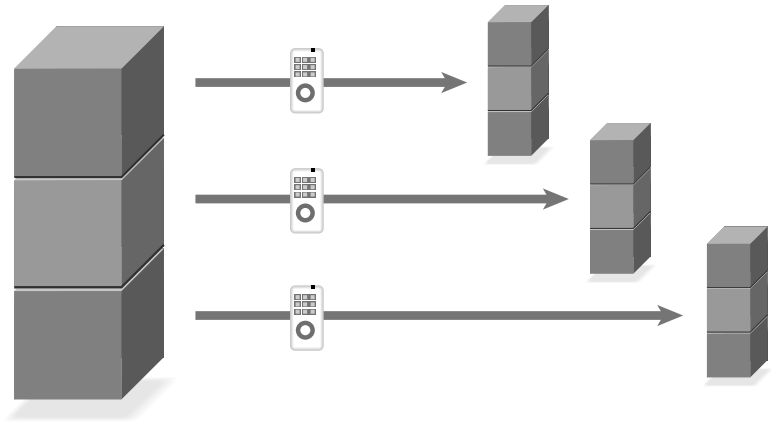
```
0 성적 점수를 넣어주세요.
10
1 성적 점수를 넣어주세요.
20
2 성적 점수를 넣어주세요.
30
3 성적 점수를 넣어주세요.
40
4 성적 점수를 넣어주세요.
50
처리할 점수들은 다음과 같습니다.
[10, 20, 30, 40, 50]
평균 : 30.0
최고 점수 : 50
최저 점수 : 10
```

6 배열의 남은 내용

앞에서 배열에 대한 기본적인 내용과 개념 위주로 설명을 해왔습니다. 이제는 배열의 남은 내용에 대해서 좀 더 설명하도록 하겠습니다.

6.1 다차원 배열: 배열 안에 또 배열

배열의 내부에 또 다른 배열을 넣어 두는 것을 다차원 배열이라고 합니다. 다차원 배열은 흔히들 수학에서의 행렬에 비유하는 경우를 자주 보는데 그보다는 바깥쪽 배열의 각 상자의 내용물이 다시 또 다른 배열을 가리키는 구조로 이해하는 것이 정확합니다.



맨 바깥쪽 배열 안의 요소는 다른 배열들의 리모컨이다.

그림 18

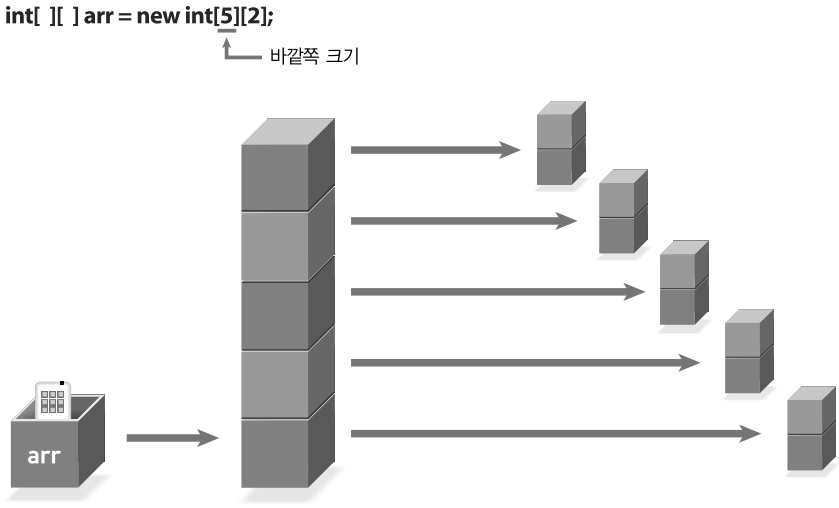
그림에서 보는 것처럼 바깥쪽의 배열 안의 상자 안쪽에 다시 또 다른 배열의 리모컨을 넣어주는 구조이기 때문에 중첩된 모습으로 표현될 수 있습니다. 다차원 배열은 n 차원까지 선언하는 것이 가능하지만 주로 2차원 배열까지 사용하는 경우가 많습니다. 다차원 배열의 선언은 일반 배열보다 뒤에 필요한 만큼의 배열 표시가 더 있는 형태입니다.

```
int[ ][ ] arr (2차원 배열)
int[ ][ ][ ] arr (3차원 배열)
```

다차원 배열에 대해서는 어떤 책들을 보면 마치 바둑판처럼 모양을 그려두고 설명하는 경우를 봅니다. 그런 설명에서는 주로 수학에서 나오는 행렬의 개념을 이용해서 설명하는 경우가 많은데, 사실 다차원 배열이란 그렇게 가로, 세로가 정확하게 만들어지지 않을 때에도 사용될 수 있습니다. 다차원 배열은 배열 내부에 있는 변수가 다시 배열을 가리키는 것을 의미합니다. 예를 들어 다음과 같은 선언을 봅니다.

```
int[ ][ ] arr = new int[5][2];
```

arr이라는 배열은 그림으로 그리면 다음과 같이 됩니다.



바깥쪽의 크기는 5이고 각 안쪽은 2개짜리 배열들의 리모컨이 들어가는 공간이다.

그림 19

배열의 각 상자에 다시 배열의 리모컨이 들어간다는 사실을 확인해보도록 합니다.

예제 간단히 2차원 배열의 안쪽을 조사해보기

```
public class MultiArr1 {

	public static void main(String[] args) {

		int[][] arr = new int[5][2];

		System.out.println(arr[0]);

	}
}
```

```
[I@de6ced ← @ 뒤에 나오는 값은 달라질 수 있습니다.
```

코드에서 출력하는 것은 arr이라는 배열의 상자에서 가장 위쪽에 있는 상자의 내용물입니다. 결과를 보면 가장 위쪽에 있는 상자에 내용물로 '@'와 '['가 포함된 결과가 보입니다. 이 두 기호가 의미하는 바는 현재 arr[0] 안에 들어 있는 내용물은 배열이라는 것입니다.

■ **배열에서는 바깥쪽 크기만 중요합니다.**

다차원 배열에서 가장 중요한 것은 바깥쪽 배열의 크기라는 점입니다. 즉 다차원 배열은 간단히 보자면 배열 안쪽에 다시 배열의 리모컨을 저장하는 구조이기 때문에 단순히 보자면 바깥쪽에서는 배열의 크기가 처음부터 고정되어야 한다는 점입니다. 따라서 다차원 배열은 다음과 같은 형태로 선언할 수 있습니다.

예제

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

이런 배열이라면 실제로 배열이 만들어지는 모양을 그리자면 다음과 같은 형태가 됩니다.

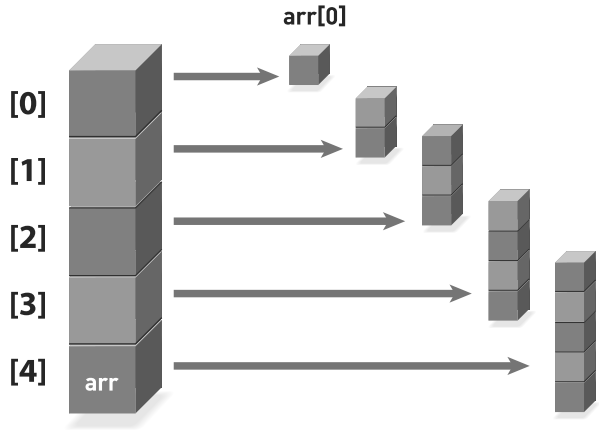


그림 20

다차원 배열에서 가장 중요한 것은 맨 바깥쪽의 크기를 고정하는 일이기 때문에 배열의 각 상자 안에는 지정된 타입의 배열의 리모컨이기만 하면 됩니다.

■ **다차원 배열에는 다중 루프를 이용하면 됩니다.**

2차원 배열 안에 있는 내부를 채우거나 2차원 배열 안의 내용을 보려면 어떻게 해야 할까요? 배열 안쪽에 다시 배열이 선언되어 있기 때문에 바깥쪽의 배열을 순환하면서 나오는 i번째의 요소는 배열의 리모컨입니다. 이런 방법으로 앞에서 선언된 배열에 차례대로 데이터를 넣도록 해봅니다.

예제 | 루프를 이용한 2차원 배열 사용하기

```
import java.util.Arrays;

public class MultiArr1 {
      public static void main(String[] args) {

            //int[][] arr = new int[5][2];
            //System.out.println(arr[0]);
            int[][] arr = new int[5][];
            arr[0] = new int[1];
            arr[1] = new int[2];
            arr[2] = new int[3];
            arr[3] = new int[4];
            arr[4] = new int[5];

            int num = 0; // 계속 증가하는 값

            for(int i = 0; i < arr.length ; i++){
                  // 바깥쪽 배열 안쪽은 다시 배열. 즉 temp는 arr[i] 번째 배열
                  int[] tempArr = arr[i];
                  for(int j = 0; j< tempArr.length ; j++){
                        tempArr[j] = ++num;
                  }
            }

            //foreach // 배열 안쪽이 내용물 확인

            for(int[] tempArr:arr){
                  System.out.println(Arrays.toString(tempArr));
            }
      }
}
```

```
[1]
[2, 3]
[4, 5, 6]
[7, 8, 9, 10]
[11, 12, 13, 14, 15]
```

앞의 예제를 보면 다차원 배열 안의 각 상자에 다시 각각 크기가 다른 배열을 넣어 주었습니다. 중간의 for 루프에서는 바깥쪽의 루프를 돌아서 각 상자에 접근하고 있습니다. 접근된 상자에 대해서 루프 내에서 다시 루프를 돌면서 번호를 채워주는 코드입니다. 출력하는 코드는 JDK1.5 이후에 지원되는 foreach 구문을 이용해서 출력하고 있습니다. foreach는 배열을 사용하는 for 구문을 좀 더 간략하게 작성할 수 있는 장점이 있습니다. 기존의 방식대로 작성한다면 다음과 같이 작성할 수 있습니다. 자세한 내용은 뒤에 설명하도록 합니다.

코드

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

위의 코드에서는 다차원 배열의 바깥쪽 상자의 내용물은 결국 다른 배열의 리모컨이라는 사실을 이용해서 안쪽 배열의 내용을 Arrays.toString( )을 이용해서 출력하고 있습니다.

6.2 System.arraycopy( )를 이용하는 손쉬운 배열 복사

가끔은 배열을 이용하다가 복사를 해야 하는 경우도 발생합니다. 배열을 다른 배열로 복사할 때에 손쉽게 루프를 이용해서 처리하는 방법도 있습니다만, 속도의 측면이나 복잡함 때문에 System.arraycopy( )라는 것을 이용해 보시는 것이 좋습니다. 사용하는 방법은 아주 단순합니다.

> **System.arraycopy([원본 배열], [원본의 복사 시작 인덱스 번호], [대상 배열], [대상 배열의 복사 시작 인덱스 번호], [개수]);**

주의해서 볼 점은 인덱스 번호는 0에서부터 시작하는 번호라는 것이고, 개수는 1, 2, 3, ...과 같이 진행된다는 점입니다.

예제 | System.arraycopy( )를 이용한 간편한 배열 복사

```
import java.util.Arrays;

public class ArrayCopy {
      public static void main(String[] args) {
```

```
            int[] arr = {1,2,3,4,5};
            int[] temp = {1,2,0,0,0};

            System.arraycopy(arr, 2, temp, 2,3);
            System.out.println(Arrays.toString(arr));
            System.out.println(Arrays.toString(temp));
      }
}
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

6.3 JDK1.5의 배열의 루프 처리

JDK1.5는 상당히 많은 부분에서 문법적인 변경이 있습니다. 따라서 foreach라고 얘기하는 문법적인 부분도 좀 알아 두시면 도움이 되실 듯합니다. 가장 확실한 것은 기존의 루프와 비교를 통해서 일 듯하니 바로 소스로 설명해볼까 합니다.

예제 | 전통적인 루프를 이용하는 배열 데이터의 출력

```
public class ForEx {
      public static void main(String[] args) {
            int[] arr = {10,20,30,40,50};
            for(int i = 0; i < arr.length; i++){
                  System.out.println(arr[i]);
            }//end for
      }//end main
}
```

전통적인 방식에서는 'int i = 0;'과 같이 내부적으로 변수의 인덱스를 사용하기 위한 변수를 설정해서 사용하는 방식으로 처리하고 있었습니다. 반면에 JDK1.5에서는 배열을 순환할 때 조금은 다른 방식을 사용합니다.

예제 | JDK1.5의 foreach 문

```
int[] arr = {10,20,30,40,50};

for(int value : arr){
      System.out.println(value);
}
```

루프가 순환되는 것을 보면 인덱스 번호를 따로 처리하지 않고 있음을 볼 수 있습니다. 대신에 루프의 앞에 쓰인 int value라는 변수가 매번 루프가 돌 때마다 'int value = arr[i]'와 같이 처리되는 방식을 사용하는 겁니다. foreach를 이용하면 매번 'int value = arr[i];'와 같이 인덱스 번호를 사용하지 않아도 된다는 점이 편리합니다. 하지만, 반대로 인덱스 번호가 홀수 번째와 짝수 번째가 다르게 동작한다든가, 인덱스 번호를 적극적으로 활용해야 하는 코드에서는 기존 방식의 for 루프를 이용하는 것이 더 편리합니다.

6.4 기본형 자료의 정렬도 가능합니다.

앞에서 Arrays.toString( )이라는 기능을 사용해본 적이 있습니다. Arrays.toString( )을 이용하면 배열 안의 데이터를 문자열로 변경해주기 때문에 좀 더 간편하게 배열을 들여다볼 수 있습니다. 사실 Arrays에는 toString( ) 외에 강력한 기능이 하나 더 있는데 그 기능은 바로 sort 기능, 즉 정렬입니다. 간단한 구문이기 때문에 직접 예제를 보면서 설명하도록 합니다.

예제 | Arrays.sort를 이용하면 기본 자료형의 정렬도 가능합니다.

```
import java.util.Arrays;

public class SortEx {
      public static void main(String[] args) {

            int[] arr  = {100,91,30,30,40,50 };
            System.out.println(Arrays.toString(arr));

            Arrays.sort(arr);
            System.out.println(Arrays.toString(arr));
      }
}
```

```
[100, 91, 30, 30, 40, 50]
[30, 30, 40, 50, 91, 100]
```

코드를 보면 처음에 배열을 Arrays.toString( )을 이용해서 문자열로 만들어서 출력해보았습니다. 이후에 'Arrays.sort(arr);'이라는 코드가 사용된 것이 보입니다. 이때 바로 데이터의 정렬이 일어납니다. 이제는 arr은 원래의 데이터 순서가 아닌 정렬된 순서를 가지게 됩니다. 마지막에는 정렬 작업을 끝낸 다음에 다시 데이터를 출력해보는 겁니다. 결과를 보면 원본 배열의 데이터의 순서가 변경되었음을 보실 수 있습니다.