8. 배열

#1.인강/0.자바/1.자바-입문

배열 시작

배열이 필요한 이유

학생의 점수를 출력하는 간단한 프로그램을 작성해보자.

Array1

```
package array;

public class Array1 {
    public static void main(String[] args) {
        int student1 = 90;
        int student2 = 80;
        int student3 = 70;
        int student4 = 60;
        int student5 = 50;

        System.out.println("학생1 점수: " + student1);
        System.out.println("학생2 점수: " + student2);
        System.out.println("학생3 점수: " + student3);
        System.out.println("학생4 점수: " + student4);
        System.out.println("학생5 점수: " + student5);
```

```
    }
}
```

실행 결과

```
학생1 점수: 90
학생2 점수: 80
학생3 점수: 70
학생4 점수: 50
학생5 점수: 90
```

- 학생을 몇 명 더 추가해야 한다면 변수를 선언하는 부분과 점수를 출력하는 부분의 코드도 추가해야한다. 학생을 몇 명 더 추가하는 것은 개발자가 코딩으로 해결할 수 있겠지만, 학생을 수백 명 이상 추가해야 한다면 코드가 상당히 많이 늘어날 것이다. 결국 학생 수가 증가함에 따라 코딩 양이 비례해서 증가하는 문제가 발생한다.
- 변수를 선언하는 부분을 보면 학생 수가 증가함에 따라 `int` 형 변수를 계속해서 추가해야 한다. 학생 수가 5명이면 `int` 형 변수를 5개 선언해야 하고, 학생 수가 100명이라면 `int` 형 변수를 100개 선언해야 한다. 결국 비슷한 변수를 반복해서 선언하는 문제가 발생한다.
- 반복문으로 해결할 수 있을 것 같지만, 점수를 출력하는 부분을 보면 변수의 이름이 다르기 때문에 반복문도 적용할 수 없다.

이렇게 같은 타입의 변수를 반복해서 선언하고 반복해서 사용하는 문제를 해결하는 것이 바로 배열이다.

배열의 선언과 생성

배열은 같은 타입의 변수를 사용하기 편하게 하나로 묶어둔 것이다. 이전 예제를 배열을 사용하도록 변경해보자.
참고로 단계적으로 구조를 변경해 나갈 것이다.

Array1Ref1

```
package array;

public class Array1Ref1 {
    public static void main(String[] args) {
        int[] students; //배열 변수 선언
        students = new int[5]; //배열 생성

        //변수 값 대입
        students[0] = 90;
```

```
        students[1] = 80;
        students[2] = 70;
        students[3] = 60;
        students[4] = 50;

        //변수 값 사용
        System.out.println("학생1 점수: " + students[0]);
        System.out.println("학생2 점수: " + students[1]);
        System.out.println("학생3 점수: " + students[2]);
        System.out.println("학생4 점수: " + students[3]);
        System.out.println("학생5 점수: " + students[4]);
    }
}
```

지금부터 아주 간단해보이는 다음 두 줄을 아주 자세히 설명하겠다. 집중해서 따라오자.

```
int[] students; //1. 배열 변수 선언
students = new int[5]; //2. 배열 생성
```

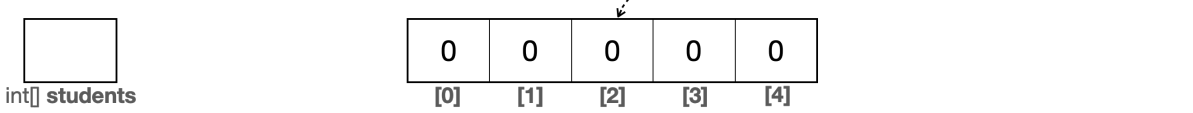
1. 배열 변수 선언

1. int[] students; //배열 변수 선언

- 배열을 사용하려면 `int[] students;`와 같이 배열 변수를 선언해야 한다.
- 일반적인 변수와 차이점은 `int[]`처럼 타입 다음에 대괄호(`[]`)가 들어간다는 점이다.
- 배열 변수를 선언한다고해서 아직 사용할 수 있는 배열이 만들어진 것은 아니다!
  - `int a`에는 정수를, `double b`에는 실수를 담을 수 있다.
  - `int[] students`와 같은 배열 변수에는 배열을 담을 수 있다. (배열 변수에는 10, 20 같은 값이 아니라 배열이라는 것을 담을 수 있다)

2. 배열 생성

2. students = new int[5]; //배열 생성



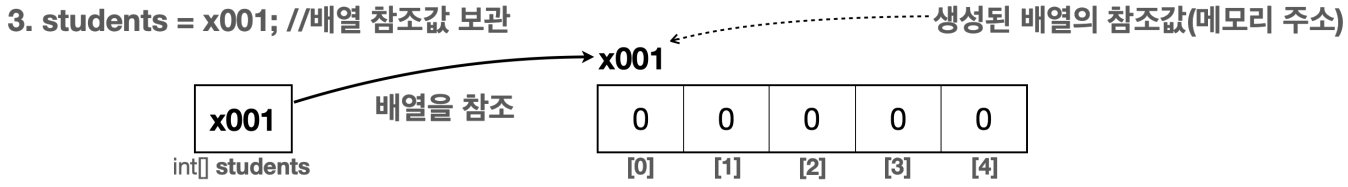
- 배열을 사용하려면 배열을 생성해야 한다.
- `new int[5]`라고 입력하면 오른쪽 그림과 같이 총 5개의 `int`형 변수가 만들어진다.
- `new`는 새로 생성한다는 뜻이고, `int[5]`는 `int`형 변수 5개라는 뜻이다. 따라서 `int`형 변수 5개를 다룰 수 있는 배열을 새로 만든다는 뜻이다.

- 앞서 `int student1`, `int student2` ... `int student5`까지 총 5개의 변수를 직접 선언했다. 배열을 사용하면 이런 과정을 한번에 깔끔하게 처리할 수 있다.

배열과 초기화

- `new int[5]`라고 하면 총 5개의 `int`형 변수가 만들어진다. 자바는 배열을 생성할 때 그 내부값을 자동으로 초기화한다.
- 숫자는 `0`, `boolean`은 `false`, `String`은 `null`(없다는 뜻이다.)로 초기화 된다.

3. 배열 참조값 보관



- `new int[5]`로 배열을 생성하면 배열의 크기만큼 메모리를 확보한다.
  - `int`형을 5개 사용하면 `4byte * 5` ➜ `20byte`를 확보한다.
- 배열을 생성하고 나면 자바는 메모리 어딘가에 있는 이 배열에 접근할 수 있는 참조값(주소)(`x001`)을 반환한다.
  - 여기서 `x001`이라고 표현한 것이 참조값이다. (실제로 `x001`처럼 표현되는 것은 아니고 이해를 돕기 위한 예시이다.)
- 앞서 선언한 배열 변수인 `int[] students`에 생성된 배열의 참조값(`x001`)을 보관한다.
- `int[] students` 변수는 `new int[5]`로 생성한 배열의 참조값을 가지고 있다.
  - 이 변수는 참조값을 가지고 있다. 이 참조값을 통해 배열을 참조할 수 있다. 쉽게 이야기해서 참조값을 통해 메모리에 있는 실제 배열에 접근하고 사용할 수 있다.
  - 참고로 배열을 생성하는 `new int[5]` 자체에는 아무런 이름이 없다! 그냥 `int`형 변수를 5개 연속으로 만드는 것이다. 따라서 생성한 배열에 접근하는 방법이 필요하다. 따라서 배열을 생성할 때 반환되는 참조값을 어딘가에 보관해두어야 한다. 앞서 `int[] students`변수에 참조값(`x001`)을 보관해두었다. 이 변수를 통해서 이 배열에 접근할 수 있다.

이 부분을 풀어서 설명하면 다음과 같다.

```
int[] students = new int[5]; //1. 배열 생성
int[] students = x001; //2. new int[5]의 결과로 x001 참조값 반환
students = x001 //3. 최종 결과
```

참조값을 확인하고 싶다면 다음과 같이 배열의 변수를 출력해보면 된다.

```
System.out.println(students); //[I@4617c264 @앞의 [I는 int형 배열을 뜻한다. @뒤에 16진수는 참조값을 뜻한다.
```

- 참조값에 대한 더 자세한 내용은 뒤에서 다룬다. 지금은 생성한 배열을 참조할 수 있는, 메모리의 주소를 나타내는

특별한 값이 있다는 정도만 이해하면 충분하다.

배열 사용

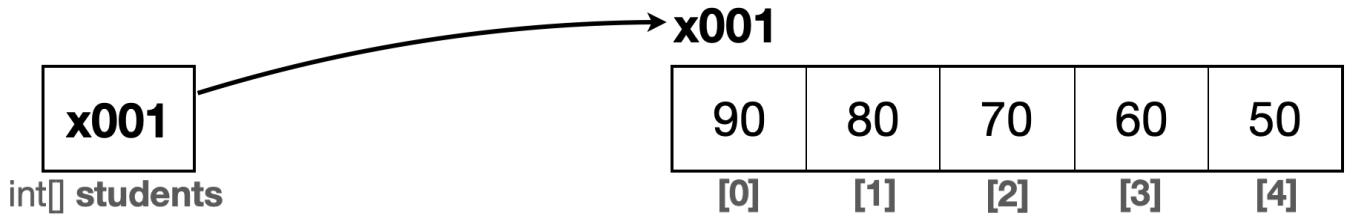
인덱스

배열은 변수와 사용법이 비슷한데, 차이점이 있다면 다음과 같이 `[]` 사이에 숫자 번호를 넣어주면 된다. 배열의 위치를 나타내는 숫자를 인덱스(index)라 한다.

```
//변수 값 대입
students[0] = 90;
students[1] = 80;

//변수 값 사용
System.out.println("학생1 점수: " + students[0]);
System.out.println("학생2 점수: " + students[1]);
```

배열 참조 그림



배열은 0부터 시작한다

`new int[5]`와 같이 5개의 요소를 가지는 `int`형 배열을 만들었다면 인덱스는 `0,1,2,3,4`가 존재한다.
여기서 주의해야 할 점이 있는데 인덱스는 0부터 시작한다는 것이다. 배열의 요소를 5개로 생성했지만, 인덱스는 0부터 시작한다. 따라서 사용 가능한 인덱스의 범위는 `0 ~ (n-1)`이 된다. 그래서 `students[4]`가 배열의 마지막 요소이다.

만약 `students[5]`와 같이 접근 가능한 배열의 인덱스 범위를 넘어가면 다음과 같은 오류가 발생한다.

인덱스 허용 범위를 넘어설 때 발생하는 오류

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5 at array.Array1Ref1.main(Array1Ref1.java:14)
```

배열에 값 대입

배열에 값을 대입하든 배열의 값을 사용하든 간에 일반적인 변수와 사용법은 같다. 추가로 `[]`를 통해 인덱스만 넣어주면 된다.

```
students[0] = 90; //1. 배열에 값을 대입
x001[0] = 90; //2. 변수에 있는 참조값을 통해 실제 배열에 접근. 인덱스를 사용해서 해당 위치의 요소에 접근, 값 대입
```

```
students[1] = 80; //1. 배열에 값을 대입
x001[1] = 80; //2. 변수에 있는 참조값을 통해 실제 배열에 접근. 인덱스를 사용해서 해당 위치의 요소에 접근, 값 대입
```

배열 값 읽기

```
//1. 변수 값 읽기
System.out.println("학생1 점수: " + students[0]);
//2. 변수에 있는 참조값을 통해 실제 배열에 접근. 인덱스를 사용해서 해당 위치의 요소에 접근
System.out.println("학생1 점수: " + x001[0]);
//3. 배열의 값을 읽어옴
System.out.println("학생1 점수: " + 90);
```

배열을 사용하면 이렇게 참조값을 통해서 실제 배열에 접근하고 인덱스를 통해서 원하는 요소를 찾는다.

기본형 vs 참조형

자바의 변수 데이터 타입을 가장 크게 보면 기본형과 참조형으로 분류할 수 있다. 사용하는 값을 직접 넣을 수 있는 기본형, 그리고 방금 본 배열 변수와 같이 메모리의 참조값을 넣을 수 있는 참조형으로 분류할 수 있다.

- 기본형(Primitive Type): 우리가 지금까지 봤던 `int`, `long`, `double`, `boolean`처럼 변수에 사용할 값을 직접 넣을 수 있는 데이터 타입을 기본형(Primitive Type)이라 한다.
- 참조형(Reference Type): `int[] students`와 같이 데이터에 접근하기 위한 참조(주소)를 저장하는 데이터 타입을 참조형(Reference Type)이라 한다. 뒤에서 학습하는 객체나 클래스를 담을 수 있는 변수들도 모두 참조형이다.

참고

배열은 왜 이렇게 복잡하게 참조형을 사용할까? 지금까지 배운 변수처럼 단순히 그 안에 값을 넣고 사용하면 되는 것 아닐까?

기본형은 모두 사이즈가 명확하게 정해져있다.

```
int i; //4byte
long l;//8byte
double d; //8byte
```

그런데 배열은 다음과 같이 동적으로 사이즈를 변경할 수 있다.

```
int size=10000; //사용자가 입력한 값을 넣었다고 가정해보자.
new int[size]; //이 코드가 실행되는 시점에 배열의 크기가 정해진다.
```

- 기본형은 선언과 동시에 크기가 정해진다. 따라서 크기를 동적으로 바꾸거나 할 수는 없다. 반면에 앞서본 배열과 같은 참조형은 크기를 동적으로 할당할 수 있다. 예를 들어서 `Scanner`를 사용해서 사용자 입력에 따라 `size` 변수의 값이 변하고, 생성되는 배열의 크기도 달라질 수 있다. 이런 것을 동적 메모리 할당이라 한다. 기본형은 선언과 동시에 사이즈가 정적으로 정해지지만, 참조형을 사용하면 이처럼 동적으로 크기가 변해서 유연성을 제공할 수 있다.
- 기본형은 사용할 값을 직접 저장한다. 반면에 참조형은 메모리에 저장된 배열이나 객체의 참조를 저장한다. 이로 인해 참조형은 더 복잡한 데이터 구조를 만들고 관리할 수 있다. 반면 기본형은 더 빠르고 메모리를 효율적으로 처리한다.

기본형과 참조형에 대한 내용은 객체를 설명할 때 더 자세히 다룬다. 지금은 기본형과 참조형이라는 2가지 구분이 있다는 점만 이해하면 충분하다.

배열 리펙토링

배열 리펙토링 - 변수 값 사용

이제 배열을 사용해서 코드를 단계별로 리펙토링 해보자.

먼저 변수 값을 사용한 부분을 변경해보자.

```
//변수 값 사용
System.out.println("학생1 점수: " + students[0]);
System.out.println("학생2 점수: " + students[1]);
System.out.println("학생3 점수: " + students[2]);
System.out.println("학생4 점수: " + students[3]);
System.out.println("학생5 점수: " + students[4]);
```

변수명이 `students`로 같기 때문에 숫자가 반복되는 부분만 해결하면 반복문을 도입할 수 있을 것 같다. `for`문을 사용해서 문제를 해결해보자.

> **참고: 리펙토링**
> 리펙토링(Refactoring)은 기존의 코드 기능은 유지하면서 내부 구조를 개선하여 가독성을 높이고, 유지보수를 용이하게 하는 과정을 뜻한다. 이는 중복을 제거하고, 복잡성을 줄이며, 이해하기 쉬운 코드로 만들기 위해 수행된

다. 리펙토링은 버그를 줄이고, 프로그램의 성능을 향상시킬 수도 있으며, 코드의 설계를 개선하는 데에도 도움이 된다.
쉽게 이야기해서 작동하는 기능은 똑같은데, 코드를 개선하는 것을 리펙토링이라 한다.

Array1Ref2

```
package array;

public class Array1Ref2 {
    public static void main(String[] args) {
        int[] students; //배열 변수 선언
        students = new int[5]; //배열 생성

        //변수 값 대입
        students[0] = 90;
        students[1] = 80;
        students[2] = 70;
        students[3] = 60;
        students[4] = 50;

        //변수 값 사용
        for (int i = 0; i < students.length; i++) {
            System.out.println("학생" + (i + 1) + " 점수: " + students[i]);
        }
    }
}
```

- 반복문을 사용해서 배열을 통해 값을 사용하는 부분을 효과적으로 변경했다.
- 배열의 인덱스는 0부터 시작하기 때문에 반복문에서 `i = 0`을 초기값으로 사용했다.
- `students.length`
  - 배열의 길이를 제공하는 특별한 기능이다.
  - 참고로 이 값은 조회만 할 수 있다. 대입은 할 수는 없다.
  - 현재 배열의 크기가 5이기 때문에 여기서는 5가 출력된다.
- for문의 조건이 `i < students.length`이기 때문에 `i`는 `0,1,2,3,4`까지만 반복한다.
  - `i`가 5가 되면 `5 < 5`가 되면서 조건이 거짓이 되고, 반복을 종료한다.

배열 리펙토링 - 초기화

배열은 `{}`를 사용해서 생성과 동시에 편리하게 초기화 하는 기능을 제공한다.

```
int[] students;
```

```
students = new int[]{90, 80, 70, 60, 50}; //배열 생성과 초기화
```

Array1Ref3

```
package array;

public class Array1Ref3 {
    public static void main(String[] args) {
        int[] students;
        students = new int[]{90, 80, 70, 60, 50}; //배열 생성과 초기화

        for (int i = 0; i < students.length; i++) {
            System.out.println("학생" + (i + 1) + " 점수: " + students[i]);
        }
    }
}
```

이해를 돕기 위해 배열 변수의 선언과 배열의 생성 및 초기화를 두 줄로 나누었지만 다음과 같이 한줄도 가능하다.

```
int[] students = new int[]{90, 80, 70, 60, 50}; //배열 변수 선언, 배열 생성과 초기화
```

배열 리펙토링 - 간단한 배열 생성

배열은 `{}`만 사용해서 생성과 동시에 편리하게 초기화 하는 기능을 제공한다.

배열의 편리한 초기화

```
int[] students = {90, 80, 70, 60, 50};
```

단 이때는 예제와 같이 배열 변수의 선언을 한 줄에 함께 사용할 때만 가능하다.
물론 이렇게 하더라도 자바가 내부에서 배열 요소의 크기를 보고 `new int[5]`을 사용해서 배열을 생성한다. 따라서 기존 코드를 조금 더 편리하게 사용할 수 있는 편의 기능이라 생각하면 된다.

오류

```
int[] students;
students = {90, 80, 70, 60, 50};
```

Array1Ref4

```
package array;

public class Array1Ref4 {
    public static void main(String[] args) {
```

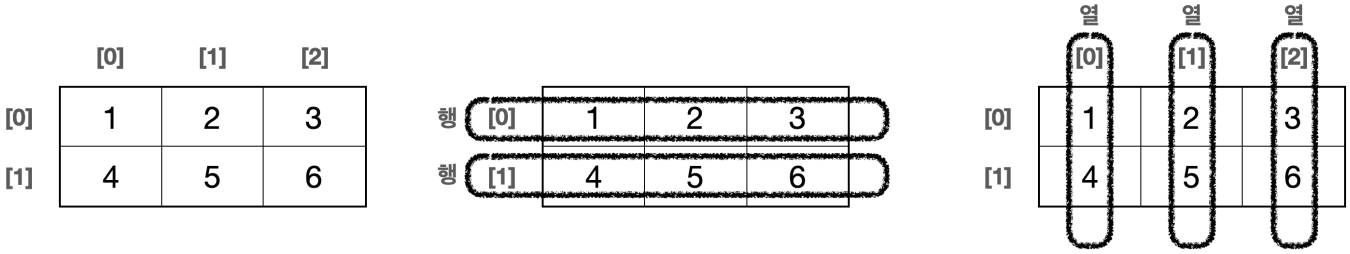
```
        //배열 생성 간략 버전, 배열 선언과 함께 사용시 new int[] 생략 가능
        int[] students = {90, 80, 70, 60, 50};

        for (int i = 0; i < students.length; i++) {
            System.out.println("학생" + (i + 1) + " 점수: " + students[i]);
        }
    }
}
```

이제 학생의 점수를 추가해도 `{90, 80, 70, 60, 50}`의 내용만 변경하면 된다. 그러면 나머지 코드는 변경하지 않아도 된다.
배열을 사용한 덕분에 프로그램을 전체적으로 잘 구조화 할 수 있었다.

2차원 배열 - 시작

지금까지 학습한 배열은 단순히 순서대로 나열되어 있었다. 이것을 1차원 배열이라 한다.
이번에 학습할 2차원 배열은 이름 그대로 하나의 차원이 추가된다. 2차원 배열은 행과 열로 구성된다.



2차원 배열은 `int[][] arr = new int[2][3]`와 같이 선언하고 생성한다. 그리고 `arr[1][2]`와 같이 사용하는데, 먼저 행 번호를 찾고, 그 다음에 열 번호를 찾으면 된다.
행은 영어로 row(로우), 열은 영어로 column(컬럼)이라 한다. 자주 사용하는 단어이니 알아두자.

2차원 배열의 사용법은 `[]`가 하나 추가되는 것을 제외하고는 앞서본 1차원 배열과 같다.

그림의 배열에 들어있는 데이터는 다음과 같다.
`arr[행][열]`, `arr[row][column]`

그림의 2차원 배열 데이터

- `arr[0][0]: 1`
- `arr[0][1]: 2`
- `arr[0][2]: 3`

- `arr[1][0]: 4`
- `arr[1][1]: 5`
- `arr[1][2]: 6`

코드를 통해서 2차원 배열의 사용법을 알아보자.

ArrayDi0

```
package array;

public class ArrayDi0 {
    public static void main(String[] args) {
        // 2x3 2차원 배열을 만든다.
        int[][] arr = new int[2][3]; //행(row), 열(column)

        arr[0][0] = 1; //0행, 0열
        arr[0][1] = 2; //0행, 1열
        arr[0][2] = 3; //0행, 2열
        arr[1][0] = 4; //1행, 0열
        arr[1][1] = 5; //1행, 1열
        arr[1][2] = 6; //1행, 2열

        //0행 출력
        System.out.print(arr[0][0] + " "); //0열 출력
        System.out.print(arr[0][1] + " "); //1열 출력
        System.out.print(arr[0][2] + " "); //2열 출력
        System.out.println(); //한 행이 끝나면 라인을 변경한다.

        //1행 출력
        System.out.print(arr[1][0] + " "); //0열 출력
        System.out.print(arr[1][1] + " "); //1열 출력
        System.out.print(arr[1][2] + " "); //2열 출력
        System.out.println(); //한 행이 끝나면 라인을 변경한다.
    }
}
```

- 이 코드는 2차원 배열을 만들고, 배열에 값을 1부터 6까지 순서대로 직접 입력한다.
- 다음과 같은 결과를 만들기 위해 0행에 있는 0,1,2열을 출력한다. 그리고 다음으로 1행에 있는 0,1,2열을 출력한다.

실행 결과

```
1 2 3 //[0][0], [0][1], [0][2]
4 5 6 //[1][0], [1][1], [1][2]
```

2차원 배열 - 리펙토링1

구조 개선 - 행 출력 반복

구조 변경

코드 구조를 보면 비슷한 부분이 반복된다.

```
//0행 출력
System.out.print(arr[0][0] + " "); //0열 출력
System.out.print(arr[0][1] + " "); //1열 출력
System.out.print(arr[0][2] + " "); //2열 출력
System.out.println(); //한 행이 끝나면 라인을 변경한다.

//1행 출력
System.out.print(arr[1][0] + " "); //0열 출력
System.out.print(arr[1][1] + " "); //1열 출력
System.out.print(arr[1][2] + " "); //2열 출력
System.out.println(); //한 행이 끝나면 라인을 변경한다.
```

코드를 보면 행을 출력하는 부분이 거의 같다. 차이가 있는 부분은 행에서 `arr[0]`으로 시작할지 `arr[1]`로 시작할지의 차이다.

다음과 같이 행(row)에 들어가는 숫자만 하나씩 증가하면서 반복하면 될 것 같다.

```
//row를 0, 1로 변경하면서 다음 코드를 반복
System.out.print(arr[row][0] + " "); //0열 출력
System.out.print(arr[row][1] + " "); //1열 출력
System.out.print(arr[row][2] + " "); //2열 출력
System.out.println(); //한 행이 끝나면 라인을 변경한다.
```

반복문을 사용하도록 코드를 변경해보자.

ArrayDi1

```
package array;

public class ArrayDi1 {
    public static void main(String[] args) {
        // 2x3 2차원 배열을 만듭니다.
```

```
        int[][] arr = new int[2][3]; //행(row), 열(column)

        arr[0][0] = 1; //0행, 0열
        arr[0][1] = 2; //0행, 1열
        arr[0][2] = 3; //0행, 2열
        arr[1][0] = 4; //1행, 0열
        arr[1][1] = 5; //1행, 1열
        arr[1][2] = 6; //1행, 2열

        for (int row = 0; row < 2; row++) {
            System.out.print(arr[row][0] + " "); //0열 출력
            System.out.print(arr[row][1] + " "); //1열 출력
            System.out.print(arr[row][2] + " "); //2열 출력
            System.out.println(); //한 행이 끝나면 라인을 변경한다.
        }
    }
}
```

- for문을 통해서 행(row)을 반복해서 접근한다. 각 행 안에서 열(column)이 3개이므로 `arr[row][0]`, `arr[row][1]`, `arr[row][2]` 3번 출력한다. 이렇게하면 for문을 한번 도는 동안 3개의 열을 출력할 수 있다.
  - `row=0`의 for문이 실행되면 `arr[0][0]`, `arr[0][1]`, `arr[0][2]`로 1 2 3 이 출력된다.
  - `row=1`의 for문이 실행되면 `arr[1][0]`, `arr[1][1]`, `arr[1][2]`로 4 5 6 이 출력된다.

실행 결과

```
1 2 3
4 5 6
```

구조 개선 - 열 출력 반복

다음 부분을 보면 같은 코드가 반복된다.

```
System.out.print(arr[row][0] + " "); //0열 출력
System.out.print(arr[row][1] + " "); //1열 출력
System.out.print(arr[row][2] + " "); //2열 출력
```

다음과 같이 열(column)에 들어가는 숫자만 하나씩 증가하면서 반복하면 될 것 같다.

```
//column를 0, 1, 2로 변경하면서 다음 코드를 반복
System.out.print(arr[row][column] + " "); //column열 출력
```

코드를 수정해보자.

ArrayDi2

```
package array;

public class ArrayDi2 {
    public static void main(String[] args) {
        // 2x3 2차원 배열을 만듭니다.
        int[][] arr = new int[2][3]; //행(row), 열(column)

        arr[0][0] = 1; //0행, 0열
        arr[0][1] = 2; //0행, 1열
        arr[0][2] = 3; //0행, 2열
        arr[1][0] = 4; //1행, 0열
        arr[1][1] = 5; //1행, 1열
        arr[1][2] = 6; //1행, 2열

        for (int row = 0; row < 2; row++) {
            for (int column = 0; column < 3; column++) {
                System.out.print(arr[row][column] + " ");
            }
            System.out.println(); //한 행이 끝나면 라인을 변경한다.
        }
    }
}
```

- for문을 2번 중첩해서 사용하는데, 첫번째 for문은 행을 탐색하고, 내부에 있는 두번째 for문은 열을 탐색한다.
- 내부에 있는 for문은 앞서 작성한 다음 코드와 같다. for문을 사용해서 열을 효과적으로 출력했다.

```
System.out.print(arr[row][0] + " "); //0열 출력
System.out.print(arr[row][1] + " "); //1열 출력
System.out.print(arr[row][2] + " "); //2열 출력
```

2차원 배열 - 리펙토링2

구조 개선 - 초기화, 배열의 길이

이 코드를 보니 2가지 개선할 부분이 있다.

1. 초기화: 기존 배열처럼 2차원 배열도 편리하게 초기화 할 수 있다.

2. `for` 문에서 배열의 길이 활용: 배열의 길이가 달라지면 `for` 문에서 `row < 2`, `column < 3` 같은 부분을 같이 변경해야 한다. 이 부분을 배열의 길이를 사용하도록 변경해보자. 배열이 커지거나 줄어들어도 for문의 코드를 변경하지 않고 그대로 유지할 수 있다.

코드를 개선해보자.

ArrayDi3

```
package array;

public class ArrayDi3 {
    public static void main(String[] args) {
        // 2x3 2차원 배열, 초기화
        int[][] arr = {
                {1, 2, 3},
                {4, 5, 6}
        };

        // 2차원 배열의 길이를 활용
        for (int row = 0; row < arr.length; row++) {
            for (int column = 0; column < arr[row].length; column++) {
                System.out.print(arr[row][column] + " ");
            }
            System.out.println();
        }
    }
}
```

초기화

1차원 배열은 다음과 같이 초기화 한다.

```
{1,2,3}
```

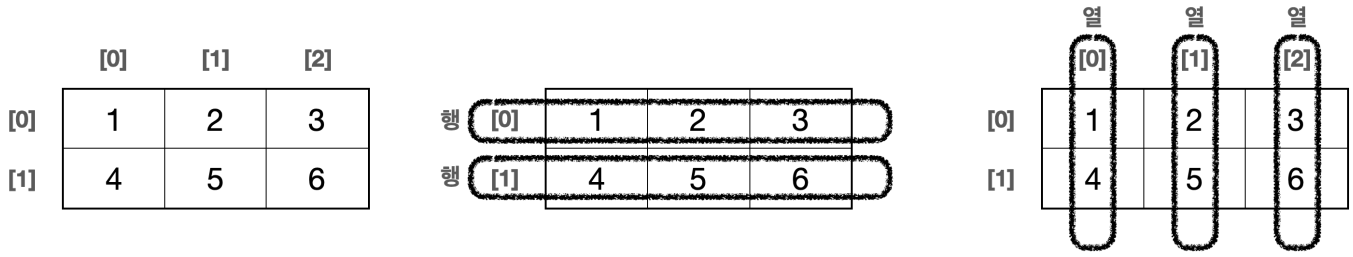
2차원 배열은 다음과 같이 초기화 한다. 밖이 행이 되고, 안이 열이 된다. 그런데 이렇게 하면 행열이 잘 안느껴진다.

```
{{1, 2, 3},{4, 5, 6}}
```

이렇게 라인을 적절하게 넘겨주면 행과 열이 명확해진다. 따라서 코드를 더 쉽게 이해할 수 있다.

```
{
  {1, 2, 3},
  {4, 5, 6}
}
```

배열의 길이



`for`문에서 2차원 배열의 길이를 활용했다.

- `arr.length`는 행의 길이를 뜻한다. 여기서는 2가 출력된다.
  - `{{},{}}`를 생각해보면 `arr`배열은 `{}`, `{}` 2개의 배열 요소를 가진다.
- `arr[row].length`는 열의 길이를 뜻한다. 여기서는 3이 출력된다.
  - `arr[0]`은 `{1,2,3}` 배열을 뜻한다. 이 배열에는 3개의 요소가 있다.
  - `arr[1]`은 `{4,5,6}` 배열을 뜻한다. 이 배열에는 3개의 요소가 있다.

잘 동작하는지 확인해보자.

이제 배열의 초기화 부분만 다음과 같이 변경하면 바로 적용된 결과를 확인할 수 있다. 나머지 코드는 변경하지 않아도 된다.

```
{1, 2, 3},
{4, 5, 6},
{7, 8, 9}
```

구조 개선 - 값 입력

이번에는 배열에 직접 1,2,3 숫자를 적어서 값을 할당하는 것이 아니라, 배열의 크기와 상관없이 배열에 순서대로 1씩 증가하는 값을 입력하도록 변경해보자.

```
package array;

public class ArrayDi4 {
    public static void main(String[] args) {
        // 2x3 2차원 배열, 초기화
        int[][] arr = new int[2][3];

        int i = 1;
        // 순서대로 1씩 증가하는 값을 입력한다.
        for (int row = 0; row < arr.length; row++) {
            for (int column = 0; column < arr[row].length; column++) {
```

```
                arr[row][column] = i++;
            }
        }

        // 2차원 배열의 길이를 활용
        for (int row = 0; row < arr.length; row++) {
            for (int column = 0; column < arr[row].length; column++) {
                System.out.print(arr[row][column] + " ");
            }
            System.out.println();
        }
    }
}
```

- 중첩된 `for` 문을 사용해서 값을 순서대로 입력한다.
- `arr[row][column] = i++` 후의 증감 연산자(`++`)를 사용해서 값을 먼저 대입한 다음에 증가한다.
  - 코드에서 `int i = 1` 으로 `i` 가 1부터 시작한다.

2차원 배열 선언 부분인 `new int[2][3]` 을 `new int[4][5]` 처럼 다른 숫자로 변경해도 잘 동작하는 것을 확인할 수 있다.

new int[4][5]로 변경시 출력

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
```

향상된 for문

앞서 반복문에서 설명하지 않은 내용이 하나 있는데, 바로 향상된 for문(Enhanced For Loop)이다. 향상된 for문을 이해하려면 배열을 먼저 알아야 한다. 각각의 요소를 탐색한다는 의미로 for-each문이라고도 많이 부른다.
향상된 `for` 문은 배열을 사용할 때 기존 `for` 문 보다 더 편리하게 사용할 수 있다.

향상된 for문 정의

```
for (변수 : 배열 또는 컬렉션) {
  // 배열 또는 컬렉션의 요소를 순회하면서 수행할 작업
}
```

코드로 확인해보자.

EnhancedFor1

```
package array;

public class EnhancedFor1 {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};

        //일반 for문
        for(int i = 0; i < numbers.length; ++i) {
            int number = numbers[i];
            System.out.println(number);
        }

        //향상된 for문, for-each문
        for (int number : numbers) {
            System.out.println(number);
        }

        //for-each문을 사용할 수 없는 경우, 증가하는 index 값 필요
        for(int i = 0; i < numbers.length; ++i) {
            System.out.println("number" + i + "번의 결과는: " + numbers[i]);
        }
    }
}
```

일반 for문

```
//일반 for문
for(int i = 0; i < numbers.length; ++i) {
    int number = numbers[i];
    System.out.println(number);
}
```

먼저 일반 for문을 살펴보자. 배열에 있는 값을 순서대로 읽어서 `number` 변수에 넣고, 출력한다.
배열은 처음부터 끝까지 순서대로 읽어서 사용하는 경우가 많다. 그런데 배열의 값을 읽으려면 `int i`와 같은 인덱스를 탐색할 수 있는 변수를 선언해야 한다. 그리고 `i < numbers.length`와 같이 배열의 끝 조건을 지정해야주어야 한다. 마지막으로 배열의 값을 하나 읽을 때 마다 인덱스를 하나씩 증가해야 한다.

개발자 입장에서는 그냥 배열을 순서대로 처음부터 끝까지 탐색하고 싶은데, 너무 번잡한 일을 해주어야 한다. 그래서 향상된 for문이 등장했다.

향상된 for문

```
//향상된 for문, for-each문
for (int number : numbers) {
    System.out.println(number);
}
```

- 앞서 일반 for문과 동일하게 작동한다.
- 향상된 for문은 배열의 인덱스를 사용하지 않고, 종료 조건을 주지 않아도 된다. 단순히 해당 배열을 처음부터 끝까지 탐색한다.
- `:`의 오른쪽에 `numbers`와 같이 탐색할 배열을 선택하고, `:`의 왼쪽에 `int number`와 같이 반복할 때 마다 찾은 값을 저장할 변수를 선언한다. 그러면 배열의 값을 하나씩 꺼내서 왼쪽에 있는 `number`에 담고 for문을 수행한다. for문의 끝에 가면 다음 값을 꺼내서 `number`에 담고 for문을 반복 수행한다. `numbers` 배열의 끝에 도달해서 더 값이 없으면 for문이 완전히 종료된다.
- 향상된 for문은 배열의 인덱스를 사용하지 않고도 배열의 요소를 순회할 수 있기 때문에 코드가 간결하고 가독성이 좋다.

향상된 for문을 사용하지 못하는 경우

그런데 향상된 for문을 사용하지 못하는 경우가 있다.

향상된 for문에는 증가하는 인덱스 값이 감추어져 있다. 따라서 `int i`와 같은 증가하는 인덱스 값을 직접 사용해야 하는 경우에는 향상된 for문을 사용할 수 없다.

```
//for-each문을 사용할 수 없는 경우, 증가하는 index 값 필요
for(int i = 0; i < numbers.length; ++i) {
    System.out.println("number" + i + "번의 결과는: " + numbers[i]);
}
```

이 예제에서는 증가하는 `i`값을 출력해야 하므로 향상된 for문 대신에 일반 for문을 사용해야 한다.

물론 다음과 같이 억지스럽게 향상된 for문을 사용하는 것이 가능하지만, 이런 경우 일반 for문을 사용하는 것이 더 좋다.

```
int i = 0;
for (int number : numbers) {
    System.out.println("number" + i + "번의 결과는: " + number);
    i++;
}
```

문제와 풀이1

> **코딩이 처음이라면 필독!**
> 프로그래밍이 처음이라면 아직 코딩 자체가 익숙하지 않기 때문에 문제와 풀이에 상당히 많은 시간을 쓰게 될 수 있다. 강의를 들을 때는 다 이해가 되는 것 같았는데, 막상 혼자 생각해서 코딩을 하려니 잘 안되는 것이다. 이것은 아직 코딩이 익숙하지 않기 때문인데, 처음 코딩을 하는 사람이라면 누구나 겪는 자연스러운 현상이다.
> 문제를 스스로 풀기 어려운 경우, 너무 고민하기 보다는 먼저 **강의 영상의 문제 풀이 과정을 코드로 따라하면서 이해하자. 반드시 코드로 따라해야 한다.** 그래야 코딩하는 것에 조금씩 익숙해질 수 있다. 그런 다음에 정답을 지우고 스스로 문제를 풀어보면 된다. 참고로 강의를 듣는 시간만큼 문제와 풀이에도 많은 시간을 들어야 제대로 성장할 수 있다!

문제 - 배열을 사용하도록 변경

다음 문제를 배열을 사용해서 개선하자.

```
package array.ex;

public class ArrayEx1 {
    public static void main(String[] args) {
        int student1 = 90;
        int student2 = 80;
        int student3 = 70;
        int student4 = 60;
        int student5 = 50;

        int total = student1 + student2 + student3 + student4 + student5;
        double average = (double) total / 5;

        System.out.println("점수 총합: " + total);
        System.out.println("점수 평균: " + average);
    }
}
```

실행 결과 예시

```
점수 총합: 350
점수 평균: 70.0
```

정답

```java
package array.ex;

public class ArrayEx1Ref {
    public static void main(String[] args) {
        int[] students = {90, 80, 70, 60, 50};

        int total = 0;
        for (int i = 0; i < students.length; i++) {
            total += students[i];
        }

        double average = (double) total / students.length;
        System.out.println("점수 총합: " + total);
        System.out.println("점수 평균: " + average);
    }
}
```

문제 - 배열의 입력과 출력

사용자에게 5개의 정수를 입력받아서 배열에 저장하고, 입력 순서대로 출력하자.

출력시 출력 포멧은 1, 2, 3, 4, 5와 같이 `, `쉼표를 사용해서 구분하고, 마지막에는 쉼표를 넣지 않아야 한다.

실행 결과 예시를 참고하자.

실행 결과 예시

```
5개의 정수를 입력하세요:
1
2
3
4
5
출력
1, 2, 3, 4, 5
```

정답

```java
package array.ex;

import java.util.Scanner;

public class ArrayEx2 {
```

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

        System.out.println("5개의 정수를 입력하세요:");
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = scanner.nextInt();
        }

        System.out.println("출력");
        for (int i = 0; i < numbers.length; i++) {
            System.out.print(numbers[i]);
            if (i < numbers.length - 1) {
                System.out.print(", ");
            }
        }
    }
}
```

문제 - 배열과 역순 출력

사용자에게 5개의 정수를 입력받아서 배열에 저장하고, 입력받은 순서의 반대인 역순으로 출력하자.

출력시 출력 포멧은 5, 4, 3, 2, 1과 같이 `, `쉼표를 사용해서 구분하고, 마지막에는 쉼표를 넣지 않아야 한다.

실행 결과 예시를 참고하자.

실행 결과 예시

```
5개의 정수를 입력하세요:
1
2
3
4
5
입력한 정수를 역순으로 출력:
5, 4, 3, 2, 1
```

정답

```
package array.ex;

import java.util.Scanner;
```

```
public class ArrayEx3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numbers = new int[5];

        System.out.println("5개의 정수를 입력하세요:");
        for (int i = 0; i < 5; i++) {
            numbers[i] = scanner.nextInt();
        }

        System.out.println("입력한 정수를 역순으로 출력:");
        for (int i = 4; i >= 0; i--) {
            System.out.print(numbers[i]);
            if (i > 0) {
                System.out.print(", ");
            }
        }
    }
}
```

문제 - 합계와 평균

사용자에게 5개의 정수를 입력받아서 이들 정수의 합계와 평균을 계산하는 프로그램을 작성하자.

실행 결과 예시

```
5개의 정수를 입력하세요:
1
2
3
4
5
입력한 정수의 합계: 15
입력한 정수의 평균: 3.0
```

정답

```
package array.ex;

import java.util.Scanner;
```

```java
public class ArrayEx4 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numbers = new int[5];
        int sum = 0;
        double average;

        System.out.println("5개의 정수를 입력하세요:");
        for (int i = 0; i < 5; i++) {
            numbers[i] = scanner.nextInt();
            sum += numbers[i];
        }

        average = (double) sum / 5;

        System.out.println("입력한 정수의 합계: " + sum);
        System.out.println("입력한 정수의 평균: " + average);
    }
}
```

문제 - 합계와 평균2

이전 문제에서 입력받을 숫자의 개수를 입력받도록 개선하자.

실행 결과 예시를 참고하자

실행 결과 예시1

```
입력받을 숫자의 개수를 입력하세요:3
3개의 정수를 입력하세요:
1
2
3
입력한 정수의 합계: 6
입력한 정수의 평균: 2.0
```

실행 결과 예시2

```
입력받을 숫자의 개수를 입력하세요:5
5개의 정수를 입력하세요:
1
2
```

```
3
4
5
입력한 정수의 합계: 15
입력한 정수의 평균: 3.0
```

정답

```java
package array.ex;

import java.util.Scanner;

public class ArrayEx5 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("입력 받을 숫자의 개수를 입력하세요:");
        int count = scanner.nextInt();

        int[] numbers = new int[count];
        int sum = 0;
        double average;

        System.out.println(count + "개의 정수를 입력하세요:");
        for (int i = 0; i < count; i++) {
            numbers[i] = scanner.nextInt();
            sum += numbers[i];
        }

        average = (double) sum / count;

        System.out.println("입력한 정수의 합계: " + sum);
        System.out.println("입력한 정수의 평균: " + average);
    }
}
```

문제와 풀이2

문제 - 가장 작은 수, 큰 수 찾기

사용자로부터 n개의 정수를 입력받아 배열에 저장한 후, 배열 내에서 가장 작은 수와 가장 큰 수를 찾아 출력하는 프로그램을 작성하자. 실행 결과 예시를 참고하자.

실행 결과 예시

```
입력받을 숫자의 개수를 입력하세요:3
3개의 정수를 입력하세요:
1
2
5
가장 작은 정수: 1
가장 큰 정수: 5
```

정답

```
package array.ex;

import java.util.Scanner;

public class ArrayEx6 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("입력받을 숫자의 개수를 입력하세요:");
        int n = scanner.nextInt();

        int[] numbers = new int[n];
        int minNumber, maxNumber;

        System.out.println(n + "개의 정수를 입력하세요:");
        for (int i = 0; i < n; i++) {
            numbers[i] = scanner.nextInt();
        }

        minNumber = maxNumber = numbers[0];
        for (int i = 1; i < n; i++) {
            if (numbers[i] < minNumber) {
                minNumber = numbers[i];
            }
            if (numbers[i] > maxNumber) {
                maxNumber = numbers[i];
```

```
            }
        }

        System.out.println("가장 작은 정수: " + minNumber);
        System.out.println("가장 큰 정수: " + maxNumber);
    }
}
```

문제 - 2차원 배열1

사용자로부터 4명 학생의 국어, 수학, 영어 점수를 입력받아 각 학생의 총점과 평균을 계산하는 프로그램을 작성하자.
2차원 배열을 사용하고, 실행 결과 예시를 참고하자.

실행 결과 예시

```
1번 학생의 성적을 입력하세요:
국어 점수:100
영어 점수:80
수학 점수:70
2번 학생의 성적을 입력하세요:
국어 점수:30
영어 점수:40
수학 점수:50
3번 학생의 성적을 입력하세요:
국어 점수:60
영어 점수:70
수학 점수:50
4번 학생의 성적을 입력하세요:
국어 점수:90
영어 점수:100
수학 점수:80
1번 학생의 총점: 250, 평균: 83.33333333333333
2번 학생의 총점: 120, 평균: 40.0
3번 학생의 총점: 180, 평균: 60.0
4번 학생의 총점: 270, 평균: 90.0
```

정답

```
package array.ex;

import java.util.Scanner;
```

```java
public class ArrayEx7 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] scores = new int[4][3];
        String[] subjects = {"국어", "영어", "수학"};

        for(int i=0; i<4; i++){
            System.out.println((i+1) + "번 학생의 성적을 입력하세요:");
            for(int j=0; j<3; j++){
                System.out.print(subjects[j] + " 점수:");
                scores[i][j] = scanner.nextInt();
            }
        }

        for(int i=0; i<4; i++){
            int total = 0;
            for(int j=0; j<3; j++){
                total += scores[i][j];
            }
            double average = total / 3.0;
            System.out.println((i+1) + "번 학생의 총점: " + total + ", 평균: " +
average);
        }
    }
}
```

문제 - 2차원 배열2

이전 문제에서 학생수를 입력받도록 개선하자.

실행 결과 예시를 참고하자.

실행 결과 예시

```
학생수를 입력하세요:3
1번 학생의 성적을 입력하세요:
국어 점수:10
영어 점수:20
수학 점수:30
2번 학생의 성적을 입력하세요:
국어 점수:10
영어 점수:10
```

```
수학 점수:10
3번 학생의 성적을 입력하세요:
국어 점수:20
영어 점수:20
수학 점수:20
1번 학생의 총점: 60, 평균: 20.0
2번 학생의 총점: 30, 평균: 10.0
3번 학생의 총점: 60, 평균: 20.0
```

정답

```
package array.ex;

import java.util.Scanner;

public class ArrayEx8 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("학생수를 입력하세요:");
        int studentCount = scanner.nextInt();

        int[][] scores = new int[studentCount][3];
        String[] subjects = {"국어", "영어", "수학"};

        for (int i = 0; i < studentCount; i++) {
            System.out.println((i + 1) + "번 학생의 성적을 입력하세요:");
            for (int j = 0; j < 3; j++) {
                System.out.print(subjects[j] + " 점수:");
                scores[i][j] = scanner.nextInt();
            }
        }

        for (int i = 0; i < studentCount; i++) {
            int total = 0;
            for (int j = 0; j < 3; j++) {
                total += scores[i][j];
            }
            double average = total / 3.0;
            System.out.println((i + 1) + "번 학생의 총점: " + total + ", 평균: " +
average);
        }
    }
```

```
}
```

문제와 풀이3

상품 관리 프로그램 만들기

자바를 이용하여 상품 관리 프로그램을 만들어 보자. 이 프로그램은 다음의 기능이 필요하다:

- 상품 등록: 상품 이름과 가격을 입력받아 저장한다.
- 상품 목록: 지금까지 등록한 모든 상품의 목록을 출력한다.

다음과 같이 동작해야 한다:

- 첫 화면에서 사용자에게 세 가지 선택을 제시한다: "1. 상품 등록", "2. 상품 목록", "3. 종료"
- "1. 상품 등록"을 선택하면, 사용자로부터 상품 이름과 가격을 입력받아 배열에 저장한다.
- "2. 상품 목록"을 선택하면, 배열에 저장된 모든 상품을 출력한다.
- "3. 종료"를 선택하면 프로그램을 종료한다.

제약 조건

상품은 최대 10개까지 등록할 수 있다.

다음은 사용해야 하는 변수 및 구조이다:

- `Scanner scanner`: 사용자 입력을 받기 위한 Scanner 객체
- `String[] productNames`: 상품 이름을 저장할 String 배열
- `int[] productPrices`: 상품 가격을 저장할 int 배열
- `int productCount`: 현재 등록된 상품의 개수를 저장할 int 변수

실행 결과 예시를 참고하자.

실행 결과 예시

```
1. 상품 등록 | 2. 상품 목록 | 3. 종료
메뉴를 선택하세요:1
상품 이름을 입력하세요:JAVA
상품 가격을 입력하세요:10000
1. 상품 등록 | 2. 상품 목록 | 3. 종료
메뉴를 선택하세요:1
상품 이름을 입력하세요:SPRING
```

```
상품 가격을 입력하세요:20000
1. 상품 등록 | 2. 상품 목록 | 3. 종료
메뉴를 선택하세요:2
JAVA: 10000원
SPRING: 20000원
1. 상품 등록 | 2. 상품 목록 | 3. 종료
메뉴를 선택하세요:3
프로그램을 종료합니다.
```

상품을 더 등록할 수 없는 경우

```
1. 상품 등록 | 2. 상품 목록 | 3. 종료
메뉴를 선택하세요:1
더 이상 상품을 등록할 수 없습니다.
```

등록된 상품이 없는 경우

```
1. 상품 등록 | 2. 상품 목록 | 3. 종료
메뉴를 선택하세요:2
등록된 상품이 없습니다.
```

정답

```java
package array.ex;

import java.util.Scanner;

public class ProductAdminEx {
    public static void main(String[] args) {
        int maxProducts = 10;
        String[] productNames = new String[maxProducts];
        int[] productPrices = new int[maxProducts];
        int productCount = 0;

        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.print("1. 상품 등록 | 2. 상품 목록 | 3. 종료\n메뉴를 선택하세요:");
            int menu = scanner.nextInt();
            scanner.nextLine();

            if (menu == 1) {
                if (productCount >= maxProducts) {
                    System.out.println("더 이상 상품을 등록할 수 없습니다.");
```

```
                    continue;
                }

                System.out.print("상품 이름을 입력하세요:");
                productNames[productCount] = scanner.nextLine();

                System.out.print("상품 가격을 입력하세요:");
                productPrices[productCount] = scanner.nextInt();

                productCount++;
            } else if (menu == 2) {
                if (productCount == 0) {
                    System.out.println("등록된 상품이 없습니다.");
                    continue;
                }
                for (int i = 0; i < productCount; i++) {
                    System.out.println(productNames[i] + ": " + productPrices[i]
+ "원");
                }
            } else if (menu == 3) {
                System.out.println("프로그램을 종료합니다.");
                break;
            } else {
                System.out.println("잘못된 메뉴를 선택하셨습니다.");
            }
        }
    }
}
```

정리