5th-week
문자열(String) 타입

> 자바에서 문자열은 String `객체`로 생성되어 String 변수에 객체의 번지가 대입된다.

문자열 비교

> String 객체를 문자열 리터럴로 생성하는지 new 연산자로 생성하는 지에 따라 비교 연산자의 결과가 달라질 수 있다.

리터럴 직접 대입으로 String 객체 생성

> 자바는 문자열 리터럴이 동일하다면 String 객체를 공유한다.

```
String name1 = "홍길동";
String name2 = "홍길동";
```

위 예제의 name1과 name2는 동일한 String 객체의 번지가 저장된다.

new 연산자로 String 객체 생성

> new 연산자로 직접 String 객체를 생성하고 대입하는 경우 각각 다른 번지를 가지게 된다.

```
String name1 = new String("홍길동");
String name2 = new String("홍길동");
```

위 예제의 name1과 name2는 서로 다른 String 객체의 번지를 가지게 된다.

```
String name1 = "홍길동";
String name2 = "홍길동";
String name3 = new String("홍길동");

name1 == name2    // true
name1 == name3    // false
```

위 예제처럼 name1과 name2는 동일한 객체를 참조하여 비교 연산자의 결과가 같지만, name3은 new 연산자로 String 객체를 별도로 생성했기 때문에 name1과 name3의 결과는 false이다.

String 내부 문자열만 비교시 equals()

> 동일한 String 객체인지 다른 객체인지 상관없이 내부 문자열만 비교할 경우 equals()를 사용한다.

`equals()` 메소드는 내부 문자열이 같은지 검사하고 대소문자를 구분해서 비교한다. 앞에 느낌표(`!`)를 붙여 내부 문자열이 다른지 검사하는 것도 가능하다.

```
boolean result = str1.equals(str2);  // 문자열이 같은지 검사
boolean result = !str1.equals(str2); // 문자열이 다른지 검사
```

String 변수에 빈 문자열(`""`)을 대입할 수 도 있다. 빈 문자열도 String 객체로 생성되기 때문에 변수가 빈 문자열을 참조하는지 조사하려면 equals() 메소드를 사용해야한다.

```
String hobby = "";
if(hobby.equals("")){
        System.out.println("hobby는 빈 문자열");
}
```

문자 추출 - charAt()

> 특정 위치의 문자를 얻고 싶다면 `charAt()` 메소드를 사용하면 된다.

자	바		프	로	그	래	밍
0	1	2	3	4	5	6	7

`자바 프로그래밍`이라는 문자열에서 charAt(3)은 3번 인덱스 위치에 있는 문자인 `프`가 해당된다.

문자열 길이 - length()

> 문자열에서 문자의 개수를 얻고 싶다면 `length()` 메소드를 사용한다.

```
String subject = "자바 프로그래밍";
int length = subject.length();  // 8
```

문자열 대체 - replace()

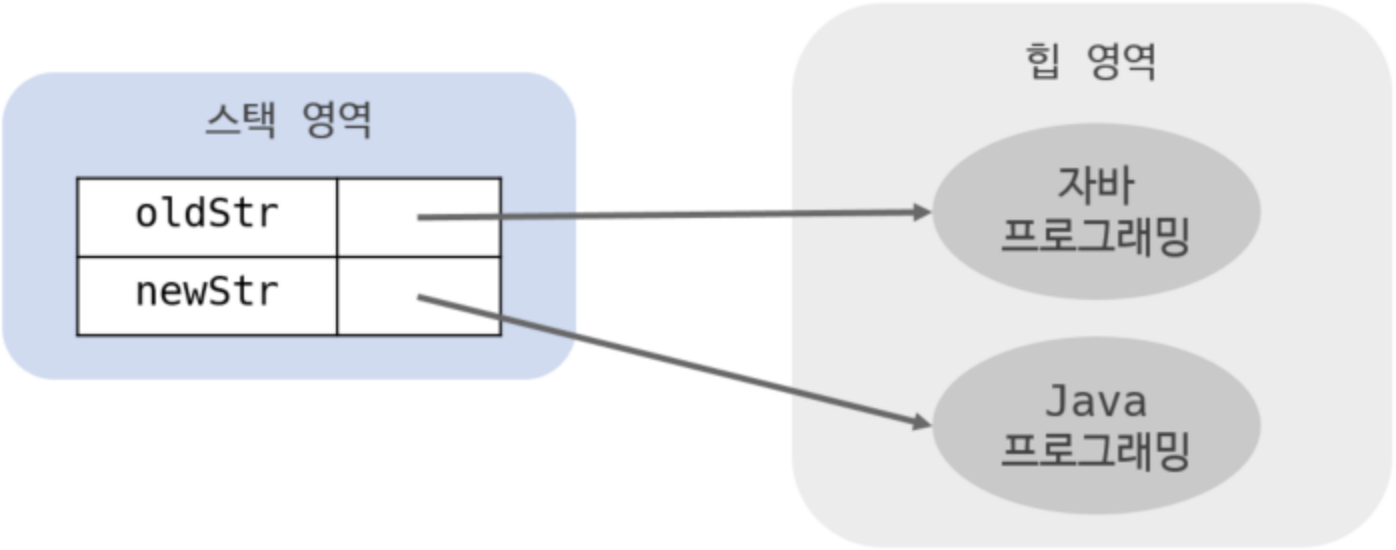
> 특정 문자열을 다른 문자열로 대체하고 싶다면 `replace()` 메소드를 사용한다.

`replace()` 메소드는 기존 문자열은 그대로 두고, 대체한 새로운 문자열을 리턴한다.

```
String oldStr = "자바 프로그래밍";
String newStr = oldStr.replacd("자바", "JAVA");
```

> String 객체의 문자열은 변경이 불가하다.

때문에 `replace()` 메소드가 리턴하는 문자열은 원래 문자열의 수정본이 아니라 완전히 새로운 문자열이다.



문자열 잘라내기 - substring()

> 문자열에서 특정 위치의 문자열을 잘라내어 가져올 경우 `substring()` 메소드를 사용한다.

메소드	설명
`substring(int beginindex)`	beginIndex부터 끝까지 잘라내기
`substring(int beginIndex, int endIndex)`	beginIndex부터 endIndex 앞까지 잘라내기

실제 값	1	2	3	4	5	6	-	7	8	9	1
인덱스	0	1	2	3	4	5	6	7	8	9	10

```
String ssn = "123456-7891";
String fistNum = ssn.substring(0,6);  // 123456
String secondNum = ssn.substring(7);  // 7891
```

문자열 찾기 - indexOf()(ps. contains())

> 특정 문자열의 위치를 찾고자 할 때에는 `indexOf()` 메소드를 사용한다.

`indexOf()` 메소드는 주어진 문자열이 시작되는 인덱스를 리턴한다.

자	바		프	로	그	래	밍
0	1	2	3	4	5	6	7

```
String subject = "자바 프로그래밍";
int index = subject.indexOf("프로그래밍");   // 3
```

주어진 문자열이 포함되어 있지 않다면 -1을 리턴한다.

```
String subject = "자바 프로그래밍";
int index = subject.indexOf("JAVA");    // -1
```

문자열 포함 여부 확인시 contains()

> 단순히 주어진 문자열이 포함되어 있는지 확일할 경우 contains()메소드를 사용한다.

원하는 문자열이 포함되어 있으면 `contains()` 메소드는 true를 리턴하고, 그렇지 않으면 false를 리턴한다.

```
String subject = "자바 프로그래밍";
boolean result = subject.contains("자바");    // true
boolean result = subject.contains("JAVA");    // false
```

문자열 분리 - split()

> 구분자를 사용해 문자열을 분리할 경우 split()메소드를 사용한다.

`split()` 메소드를 호출할 때 구분자를 제공하면 분리된 문자열로 구성 배열(array)를 리턴한다.

```
String board = "번호, 제목, 내용, 글쓴이";
String[] arr = board.split(", ");
```

arr[0]	arr[1]	arr[2]	arr[3]
번호	제목	내용	글쓴이

배열

> 연속된 공간에 값을 나열시키고, 각 값에 인덱스를 부여해 놓은 자료구조

배열의 특징

- 배열은 같은 타입의 값만 관리한다.
- 배열의 길이는 늘리거나 줄일 수 없다.

배열 변수 선언

배열을 사용하기 위해서는 배열 변수를 선언해야 한다.

- 선언방법

타입[] 변수;	타입 변수[];
int[] intArray;	int intArray[];
double[] doubleArray;	double doubleArray[];

타입[] 변수;	타입 변수[];
String[] strArray;	String strArray[];

배열 변수는 참조 변수이다. 참조할 배열이 없다면 `null`로 초기화할 수 있다.

```
타입[] 변수 = null;
```

배열 변수가 null 값을 가진 상태에서 변수[인덱스]로 값을 읽거나 저장하게 되면 `NullPointException`에러가 발생한다.

값 목록으로 배열 생성

> 타입[] 변수 = {값0, 값1, 값2, 값3, ... };

```
String[] season = {"Spring", "Summer", "Fall", "Winter"};
season[1] = "여름";
```

중괄호(`{}`)로 감싼 값 목록을 변수에 대입할 경우 미리 선언한 변수에 값 목록을 대입할 수는 없다.

String[] season;
~~season = {"Spring", "Summer", "Fall", "Winter"};~~ // Error

배열 변수를 선언한 시점과 값 목록이 대입되는 시점이 다르다면 `new 타입[]`을 중괄호 앞에 붙여주면 된다.

```
타입[] 변수;
변수 = new 타입[] {값0, 값1, 값2, 값3, ... };

String[] names = null;
names = new String[] {"김자바", "박자바", "국자바"};
```

메소드의 매개변수가 배열 타입일 경우에도 마찬가지이다.

```
// 메소드 선언
void printItem(int[] scores){ ... }

// 잘못된 메소드 호출
printItem( {98, 85, 90});   // Error

// 올바른 메소드 호출
printItem( new int[] {98, 85, 90});
```

new 연산자로 배열 생성

> 타입[] 변수 = new 타입[길이];

new 연산자는 해당 길이의 배열을 생성하고 배열의 번지를 리턴하기 때문에 배열 변수에 대입할 수 있다.

```
타입[] 변수 = null
변수 = new 타입[길이];

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

`intArray` 변수는 길이가 5인 int[] 배열을 생성하고 배열 번지를 intArray 변수에 대입한다.

intArray[0]	intArray[1]	intArray[2]	intArray[3]	intArray[4]
0	0	0	0	0

값 목록 설정 없이 new 연산자로 배열을 처음 생성하면 배열 항목은 기본값으로 초기화된다.(값이 비어있지 않음.) 타입별로 배열의 초기값은 다르다

데이터 타입	정수	실수	논리	참조
초기값	0	0.0	false	null

String[] names = new String[5];

- names의 초기값

names[0]	names[1]	names[2]	names[3]	names[4]
null	null	null	null	null

배열 생성 후 특정 인덱스 항목의 값을 변경하는 방법

> 변수[인덱스] = 값;

```
double[] arr1 = new double[3];

for(int i=0;i<3;i++){
        System.out.print("arr1[" + i + "] : " + arr2[i] + ", ");
}
System.out.println();
arr1[0] = 0.1;
arr1[2] = 0.2;
arr1[3] = 0.3;
for(int i=0;i<3;i++){
        System.out.print("arr1[" + i + "] : " + arr2[i] + ", ");
}
// arr1[0] : 0, arr1[1] : 0, arr1[2] : 0,
// arr1[0] : 0.1, arr1[1] : 0.2, arr1[2] : 0.3,
```

배열 길이 - length

> 배열 길이랑 배열을 저장할 수 있는 항목의 수를 말하며 length로 확인할 수 있다.

배열의 특징 중 하나인 배열의 길이는 늘리거나 줄일 수 없으므로 `length` 필드는 읽기만 가능하고 값은 변경할 수 없다.

~~intArray.length = 10~~ // Error

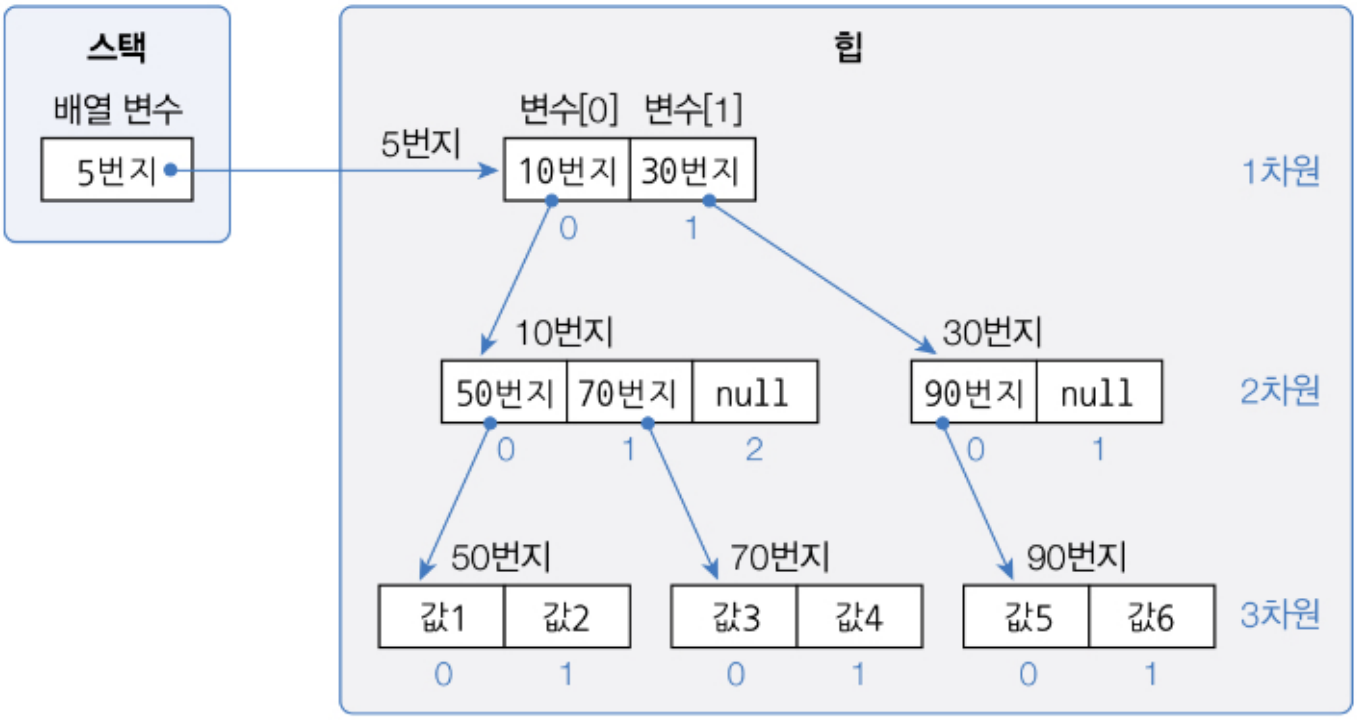
`length`는 for문에서 많이 사용된다.

```
int[] scores = {84, 90, 96};
int sum = 0;

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

System.out.println
("ArrayIndexOutOfBoundsException 에러발생 : " + scores[scores.length] );    // Error
```

위 예제의 마지막 줄처럼 배열의 인덱스를 초과해서 사용하면 `ArrayIndexOutOfBoundsException`가 발생한다.

다차원 배열

> 배열 항목에 또 다른 배열이 대입된 형태이다.



위 그림처럼 값1, 값3, 값4을 읽는 방법은 다음과 같다.

```
변수[0][0][0]    // 값1
변수[0][1][0]    // 값3
변수[1][0][1]    // 값6
```

값 목록으로 다차원 배열 생성

> 다차원 배열 변수 선언 시 타입 뒤에 대괄호 []를 차원의 수만큼 붙이고 값 목록도 차원의 수만큼 중괄호를 중첩시킨다.

```
타입[][] 변수 = {
        {값1, 값2, ...},   // 1차원 배열의 0 인덱스
        {값3, 값4, ...},   // 1차원 배열의 1 인덱스
}
```

예를 들어 2개의 반의 학생 점수를 저장하는 이차원 배열을 만들면 1차원 배열에는 각 반을 저장하고, 2차원 배열에는 각 반의 학생 점수 값을 저장하면된다.

```
int[][] scores = {
        {80, 90, 96},
        {76, 88}
}

int score = scores[0][2];  //96
int score = scores[1][1];  //88

scoers.length    // 반의 수 : 2
score[0].length  // 첫 번째 반의 학생 수 : 3
score[1].length  // 첫 번째 반의 학생 수 : 2
```

new 연산자로 다차원 배열 생성

> 배열 변수 선언 시 타입 뒤에 대괄호(`[]`)를 차원의 수만큼 붙이고, new 타입 뒤에도 차원의 수만큼 대괄호(`[]`)를 작성하면 된다.

```
타입[][] 변수 = new 타입[1차원 수][2차원 수];
```

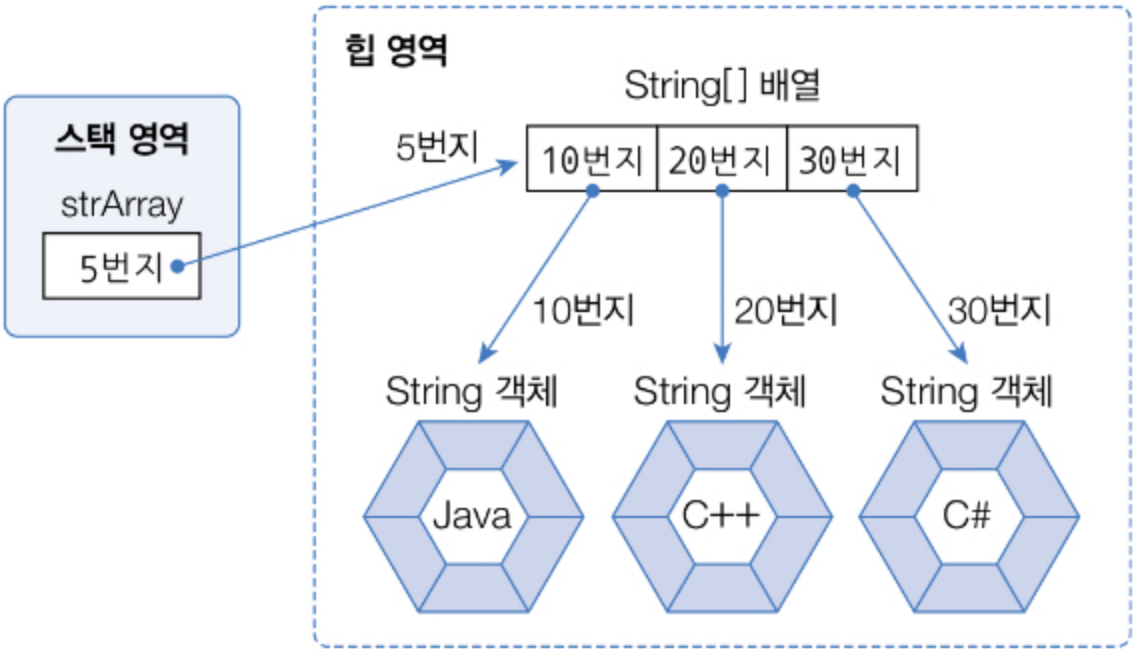
```
int[][] scores = new int[2][3];
```

객체를 참조하는 배열

> 기본 타입(byte, char, short, int, long, float, double, boolean) 배열은 각 항목에 값을 직접 저장하지만, 참조 타입(클래스, 인터페이스) 배열은 각 항목에 객체의 번지를 저장한다.

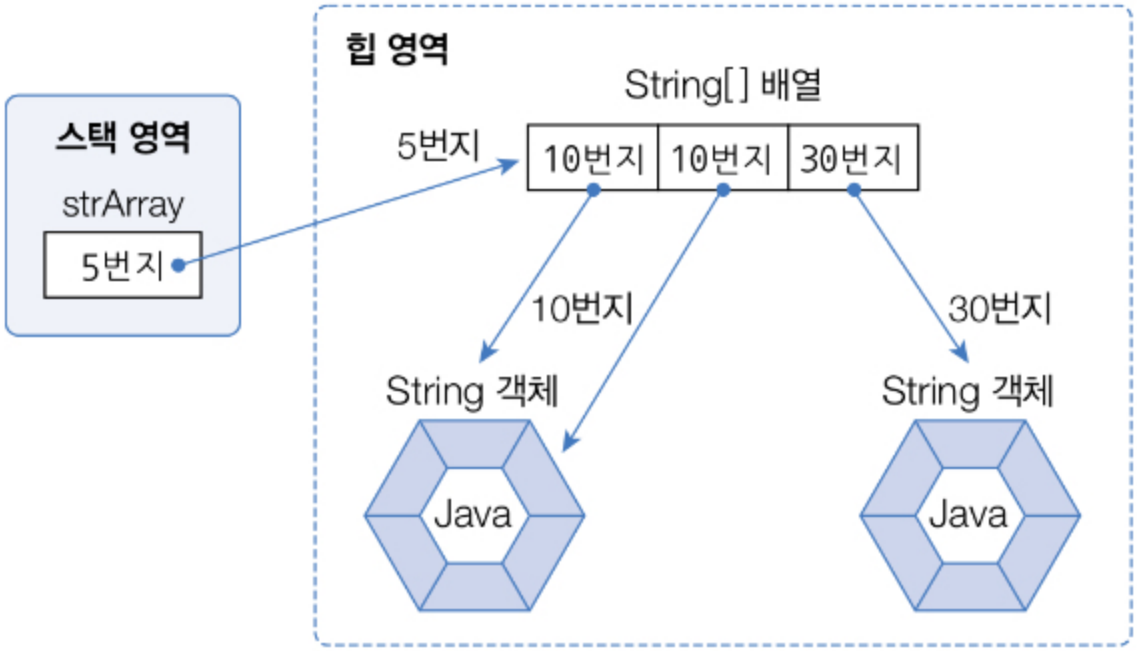
```
String[] strArray = new String[3];
strArray[0] = "Java";
```

```
strArray[1] = "C++";
strArray[2] = "C#";
```



비교 연산자(`==` , `!=`)를 사용하면 배열 항목이 참조하는 객체의 번지가 같은지(같은 객체인지)를 확인할 수 있고 `.equals()` 사용시 안에 문자열이 같은지 비교할 수 있다.

```
String[] languages = new String[3];
languages[0] = "Java";
languages[1] = "Java";
languages[2] = new String("Java");


languages[0] == languages[1]        // true - 같은 객체를 참조
languages[0] == languages[2]        // false - 다른 객체를 참조
languages[0].equals(languages[2])   // true - 문자열이 동일
```



배열 복사

> 배열은 한 번 생성하면 길이를 변경할 수 없다. 길이를 늘려야 하는 경우 더 큰 길이의 배열을 생성하고 복사해야 한다.

for문을 이용한 배열 복사

```
int[] oldArray = {1,2,3};
int[] newArray = new int[5];

for(int i=0;i<oldArray.length;i++){
        newArray[i] = oldArray[i];
}
// newArray = [1, 2, 3, 0, 0]
```

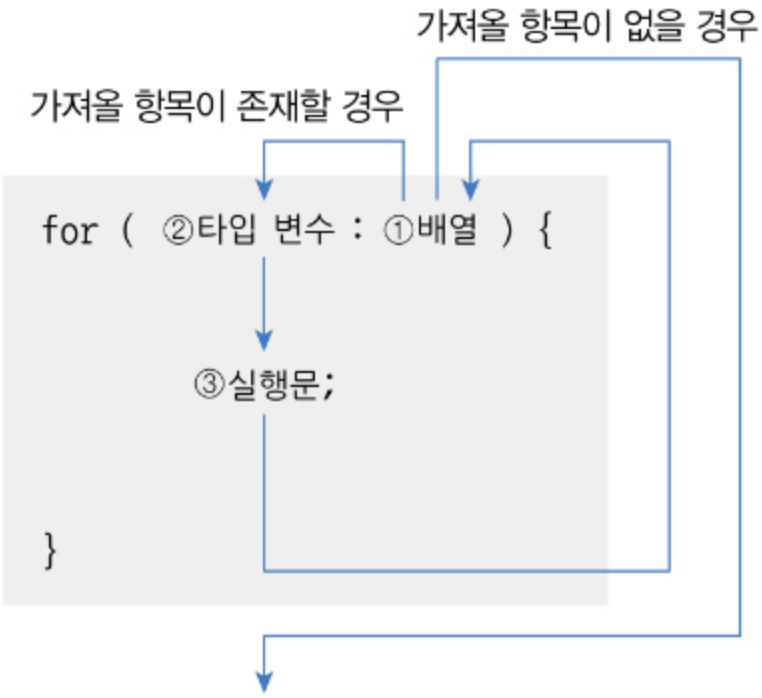
`System.arraycopy()`를 이용한 배열 복사

```
System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length);
```

- `Object src` : 원본 배열
- `int srcPos` : 원본 배열 복사 시작 인덱스
- `Object dest` : 새 배열
- `int destPos` : 새 배열 붙여넣기 시작 인덱스
- `int length` : 복사 항목 수

```
String[] oldArray = {"java", "array", "copy"};
String[] newArray = new String[5];

System.arraycopy(oldArray, 0, newArray, 0, oldArray.length);
// newArray = ["java", "array", "copy", null, null]
```

향상된 for문




```
int[] scores = {95, 71, 84, 93, 87};
int sum = 0;

for(int score: scores){
        sum +=score;
}
// sum = 430
```

---

main() 메소드의 String[] 매개변수 용도

Java 프로그램은 `java` 명령어를 통해 실행됩니다. 이 명령어는 JRE(Java Runtime Environment)를 실행시키며, JRE는 사용자가 지정한 특정 클래스를 로드합니다. 이렇게 로드된 클래스에서 `main()` 메소드를 찾아 실행합니다.

`main()` 메소드는 Java 프로그램의 시작점, 즉 진입점(entry point)입니다. 따라서 `main()` 메소드에는 규칙이 있습니다.

- `main()`은 시작점이므로 어디서든 접근 가능하게 `public` 접근 제어자를 사용해야 합니다.
- Java 프로그램 시작 시 어떠한 객체도 생성되지 않았기 때문에 `static`으로 선언되어야 합니다.
- 반환 값이 없으므로 `void`로 지정합니다. `main()` 메소드는 프로그램의 실행을 시작하고, 해당 메소드가 종료되면 프로그램이 종료되므로 별도의 반환 값은 필요하지 않습니다.
- String 타입의 배열을 파라미터로 가져야 합니다. 이 파라미터는 커맨드 라인 인자를 받습니다. 커맨드 라인 인자는 프로그램 실행 시 전달되는 값으로, 프로그램의 실행 환경이나 조건에 따라 변할 수 있는 정보를 담습니다. 예를 들어, DB 연결 정보나 처리할 파일의 경로 등을 받습니다. 커맨드 라인 인자는 문자열 형태로 전달되기 때문에 String 타입 이외의 타입으로 배열을 받을 시 컴파일 에러가 발생합니다.

```
java Sum 10 20
```

```
public class Sum{
        public static void main(String[] args){
                if(args.length != 2){
                        System.out.println("프로그램 입력값 부족");
                        System.exit(0);     // 프로그램 강제 종료
                }

                String strNum1 = args[0];
                String strNum2 = args[1];

                int num1 = Integer.parseInt(strNum1);
                int num2 = Integer.parseInt(strNum2);

                int sum = num1 + num2;
                System.out.println("두 수의 합: " + sum);    //  30
        }
}
```

열거(Enum) 타입

> 한정된 값을 갖는 타입을 열거 타입(Enum, enumeration type)이라 한다.

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

여기서 `Week`은 열거 타입 이름이며 `MONDAY` ~ `SUNDAY`는 열거 상수 목록이고 한정된 값을 표현한다.
관례적으로 열거 상수는 알파벳으로 정의하며, 모두 대문자로 작성한다. 열거 상수가 여러 단어로 구성될 경우에는 언더바(`_`)로 연결하는 것이 관례이다.

```
public enum Grade{
        USER_BASIC,
        USER_VIP
}
```

열거 타입 변수에 열거 상수를 대입할 수 있으며 열거 상수는 `열거 타입.열거 상수` 로 사용한다.

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