배열이란?

- 번호(인덱스)와 번호에 대응하는 데이터들도 이루어진 자료 구조 (by 위키백과)
- 첨자 연산자를 이용해 접근할 수 있는 인접한 원소들의 시퀀스
- 프로토타입의 자료 구조

1. Java의 배열 - 객체

배열의 원소 타입 - Java의 네 종류 타입(프리미티브, 클래스, 인터페이스, 배열) 중 하나

프리미티브 타입 int	int [] a
객체 타입 String	String [] args
인터페이스 타입 List	List[] lists
배열 타입 double []	double [][] matrix
배열 타입 int [][]	int[][][] x3d

배열 객체 선언만 하고 있다.

→ 할당되기 전에는 참조 변수 값이 null

→ new 연산자로 할당

프리미티브 타입

boolean: false, char: ₩00, byte: 0, short: 0, int: 0, long: 0L, float: 0.0F, double: 0.0

ArrayIndexOutOfBoundsException: 배열이 주어진 범위에서 벗어나면 발생

int [] a = {11, 22, 33, 44};	선언문에서 할당과 동시에 초기화
Int [] a; a = {11, 22, 33, 44};	컴파일 오류 - illegal start of expression
print(new int[] {11, 22, 33, 44});	익명 배열의 생성 & 메소드로 전달
a = new int [] {55, 66, 77, 88}	기존 배열을 재할당

b = a;	한 배열을 다른 배열에 할당해도 실제로 복사되는 것은 아니다 - 다른 이름을 가진 동일 배열
final in[] a = {11, 22, 33, 44}; a[3] = 99; → O a = new int [8]; → X	배열이 final로 선언되면 그 참조는 재할당할 수 없다.
b = (int []) a.clone()	clone 메소드 - a와 b의 내용과 길이는 같지만 주소가 다르다.

2. Java에서 배열의 프린팅

int [] array = {11, 22, 33, 44, 55, 66, 77} ;

System.out.println(array); → 주소 형태(16진수)로 출력: [I7 ~

[: 배열, I7: 7개의 int 배열

3. 간단한 배열 알고리즘

(1) 자리교환 (swap)

```
void swap (int [] a, int i, int j) {
        int ai = a[i], aj = a[j];
        if(ai == aj) return;
        a[i] = aj;
        a[j] = ai;
}
```

- 두 원소가 동등할 때, 모든 단계 생략

- 6개의 배열 접근 대신에 4개의 접근 만을 사용하여 더 효율적이다.

- 일부 정렬 알고리즘에서 swap() 메소드를 대량으로 호출할 때, 약간의 개선이 전체 실행시간을 크게 단축할 수 있다.

4. 객체의 배열

```
public class ObjectArray {
        String s = "Test";
        Float f = new Float(3.14159);
        java.util.Date d = new java.util.Date();
        int [] a = new int [] {11, 22, 33};

        Object [] objects = {s, f, d, a};
}
```

원소 타입이 참조이면 실제 원소는,

해당 배열에 대해 선언된 원소 타입의 확장에 속하기만 한다면, 다른 타입이 될 수 있다.
→ 이질 배열

5. 순차 탐색 (= 선형 탐색, 직렬 탐색)

주어진 목표값을 찾아 리스트를 앞에서부터 순차적으로 탐색

```
public int search (int [] a, int target)
{
        for (int i = 0; i <a.length; i++)
                if(a[i] == target)
                        return i;

        return -a.length;
}
```

- 목표값을 찾으면, 목표값 반환
- 목표값을 찾지 못하면, (배열의 길이 * -1) 반환
- 복잡도: Θ(n) → 빠른 탐색 방법은 아니다.

6. 복잡도 분석

시간 복잡도:

- 입력을 나타내는 문자열 길이의 함수로서 작동하는 알고리즘을 취해 시간을 정량화하는 것
- 기본적인 연산을 수행하는데 어떤 고정된 시간이 걸릴 때, 알고리즘에 의해서 수행되는 기본 연산의 개수를 세어 예측
- T(n): 최악의 시간 복잡도의 알고리즘 시간 → 크기 n의 모든 입력에 대해 걸리는 최대의 시간

점근적: 매우 큰 n값에 대한 함수의 행위

O(g(n)) = {f(n) | f(n) / g(n)에 한계없이 주어짐} (big oh of g(n))
- g보다 점근적으로 더 느리거나 동등한 모든 함수들의 집합

Ω(g(n)) = {f(n) | g(n) / f(n)에 한계없이 주어짐} (big omega of g(n))
- g보다 점근적으로 더 빠르거나 동등한 모든 함수들의 집합

Θ(g(n)) = {f(n) | f(n) / g(n)과 g{n} / f(n)에 한계없이 주어짐} (big theta of g(n))
- g와 점근적으로 동등한 모든 함수들의 집합

o(g(n)) = {f(n) | n → ∞일 때 f(n) / g(n) → 0} (little oh of g(n))
- g보다 점근적으로 더 느린 모든 함수들의 집합

ω(g(n)) = {f(n) | n → ∞일 때 g(n) / f(n) → 0} (little omega of g(n))
- g보다 점근적으로 더 빠른 모든 함수들의 집합

7. 이진 탐색: 분할과 정복(divide and conquer)

주어진 배열을 반복적으로 반으로 나누어가며, 각 단계에서 그 범위에 목표를 포함하는 반쪽에 집중

```
BinarySearch(A[0..N-1], value, low, high) {
    if (high < low)
        return -1 // not found
    mid = (low + high) / 2
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid-1)
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high)
    else
        return mid // found
}
```

- 최대 반복 횟수: lg(n)의 올림
  ex) n: 100 → lg(100) = 6.64... → 7번
  n: 1000 → lg(1000) = 9.96... →10번

- 복잡도: Θ(lg(n)) → 빠른 방법

**** 시간 복잡도가 lg(n) 인 이유**

횟수	N = 처음 입력된 개수	
1	$\frac{N}{2}$	
2	$\frac{1}{2} \times \frac{N}{2}$	
3	$\frac{1}{2} \times \frac{1}{2} \times \frac{N}{2}$	
K	$\left(\frac{1}{2}\right)^K N$	→ k번의 시행 후 남는 개수
탐색이 끝나는 시점	$\left(\frac{1}{2}\right)^K N \approx 1$	최악의 경우, 즉 찾는 데이터가 없는 경우 → 자료가 1개가 남은 시점

양변에 2^K 곱하기 $2^K \approx N$

양변에 2를 밑으로 하는 로그 $K \approx \log_2 N$

K = 시행 횟수 이므로, $\log_2 N$ 가 자료의 개수 N에 따른 시행 횟수이다.

8. java.util.Arrays 클래스

배열 조작을 위한 여러 유틸리티 메소드를 제공한다.

sort – 오름차순으로 정렬
equals – 두 배열의 동등성을 테스트
binarySearch – 이진 탐색 제공
fill (double [] a, int lo, int hi, double x) – x를 a[l0] 부터 a[hi]까지 삽입

9. java.util.Vector 클래스

크기 조정 가능한 배열의 유익한 예들을 제공

크기 조정 가능 배열: 한 배열을 동일 원소들을 포함하는 더 큰 배열로 대체

10. 다차원 배열

배열의 원소로는 임의의 타입의 객체들을 사용할 수 있다.

그러므로 배열의 원소가 다시 배열이 될 수 있다. → 2차원 배열

- 각 컴포넌트의 배열의 크기가 다르므로 울퉁불퉁한 배열 (ragged array)

```
int [][] a = new int [3][];
a[0] = new int[]{22, 44, 66, 88};
a[2] = new int[]{33, 77};
```

22	**44**	**66**	**88**	a[0] - 4개의 int 배열:
			NULL	a[1] – 배열 할당 X → NULL 값
		33	**77**	a[2] - 2개의 int 배열:

- 균일한 배열

```
int [][] b = { {22, 44, 66, 88},
               {0, 0, 0, 0} ,
               {33, 55, 77, 0}} ;
```

22	**44**	**66**	**88**
0	**0**	**0**	**0**
33	**55**	**77**	**0**