

# 자바2

자료구조 / 알고리즘 1  
(Collection Framework)

# Chapter 01

## Collection Framework

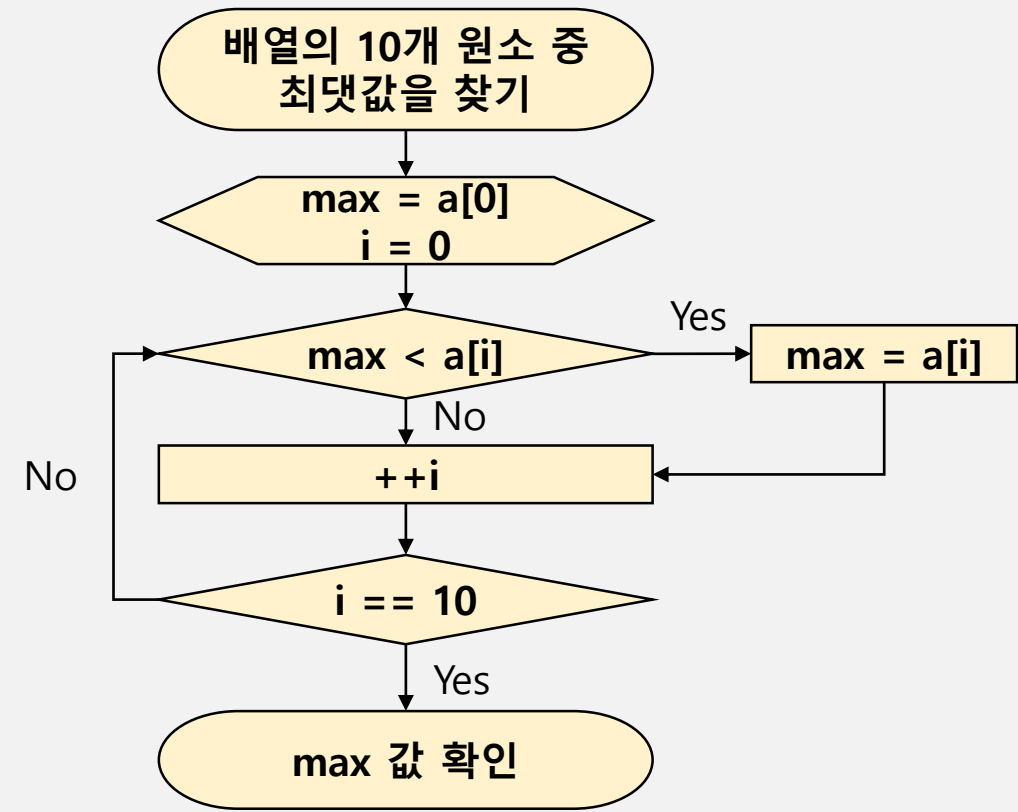
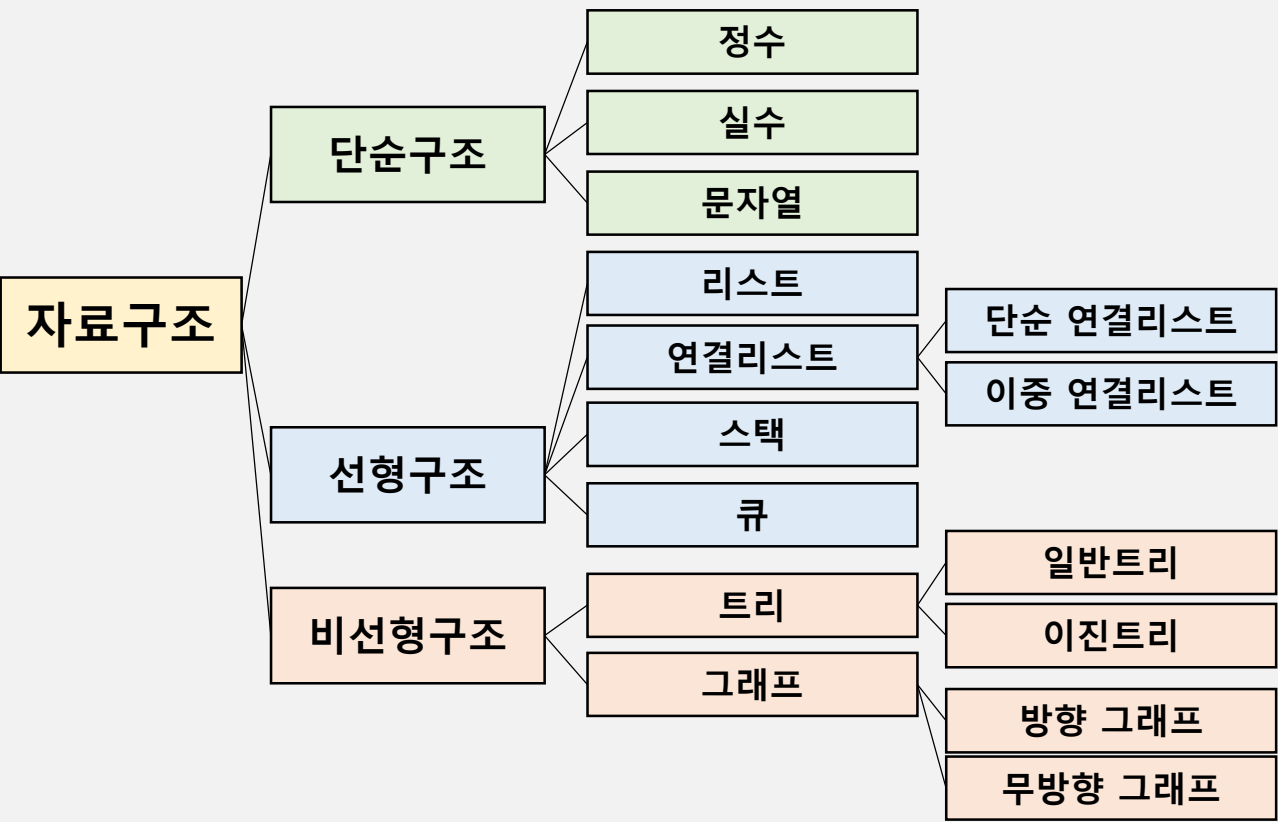
### Chapter1의 학습목표

- 자바에서 사용할 수 있는 자료구조들을 살펴보고 사용해본다
- 알고리즘과 자료구조에 대해 학습한다
- 다양한 알고리즘 설계 기법과 복잡도에 대해 알아본다
- List, LinkedList의 차이와 Stack, Queue자료구조를 이해한다.

자료구조

자료구조와 알고리즘

- 자료구조 : 자료의 사용 방법이나 성격에 따라 효율적으로 사용하기 위하여 조직하고 저장하는 방법
- 알고리즘 : 어떤 문제를 해결하기 위해 명령들로 구성된 순서화 되어있는 절차



알고리즘

알고리즘의 설계 기법

- 알고리즘 문제를 풀기 위해 연구된 기법

알고리즘 기법	방법	알고리즘 종류
Brute-force 알고리즘 완전탐색(Exhaustive search)	원하는 답을 구할 때까지 모든 가능한 경우를 테스트 모든 경우의 수를 계산/검사 하여 문제를 푸는 방법	DFS (깊이 우선 탐색)
분할 정복(divide and conquer)	주어진 문제를 여러 개의 더 작은 문제로 분할하여 해결 가능한 작은 문제로 만든 다음 해결하는 방법	합병 정렬 (Merge Sort)
동적 계획법 (dynamic programming)	원래의 문제를 더 작은 문제로 나누는 면에서 분할 정복과 유사하지만, 작은 문제를 해결하고 결과를 저장한 후 큰 문제를 해결할 때 사용하는 방법	피보나치 수열
탐욕 알고리즘(greedy algorithm)	문제를 해결하는 과정에서 모든 경우를 고려해보지 않고 그 순간의 최적의 값을 선택하여 큰 문제를 해결하는 방법	동전 문제, 배낭 문제
백 트래킹 (backtracking)과 분기 한정 알고리즘	문제의 답을 찾아가는 과정에서 현재의 답이 최종 답이 될 수 없다고 판단되면 더 이상 탐색하지 않고 되돌아가서 다른 후보 답을 탐색하는 방법	여왕 문제


## 알고리즘

### 알고리즘의 설계

#### 완전 탐색 알고리즘 예

1.  $n$ 개의 숫자가 나열된 수열을 하나 만든다.
2. 1에서 만든 수열이 왼쪽부터 작은 순서대로 나열되어 있으면 그것을 출력한다. 작은 순서대로 나열되어 있지 않다면 1로 돌아가 수열을 다시 만든다.
3. 다시 만들 때는 지금까지 만든 수열과 다르게 수열을 만든다.

운이 좋을 경우 (가장 빠른 경우)  한 번 수열을 생성 (단 1회로 정답 출력)

운이 나쁠 경우 (가장 느릴 경우)   $n!$ 번 만큼의 수열을 생성

$n = 50$ 일 경우,

$$50! = 50 * 49 * 48 * 47 * \dots 3 * 2 * 1$$

$$\rightarrow (50 * 49 * 48 * 47 * \dots 13 * 12 * 11) > (50 * 49 * 48 * 47 * \dots 3 * 2 * 1)$$

$$\rightarrow (50 * 49 * 48 * 47 * \dots 13 * 12 * 11) > 10^{40}$$

1초에 1조( $10^{12}$ )의 수열을 확인할 수 있다고 했을 때  $\rightarrow 10^{40} / 10^{12} = 10^{28}$ 초

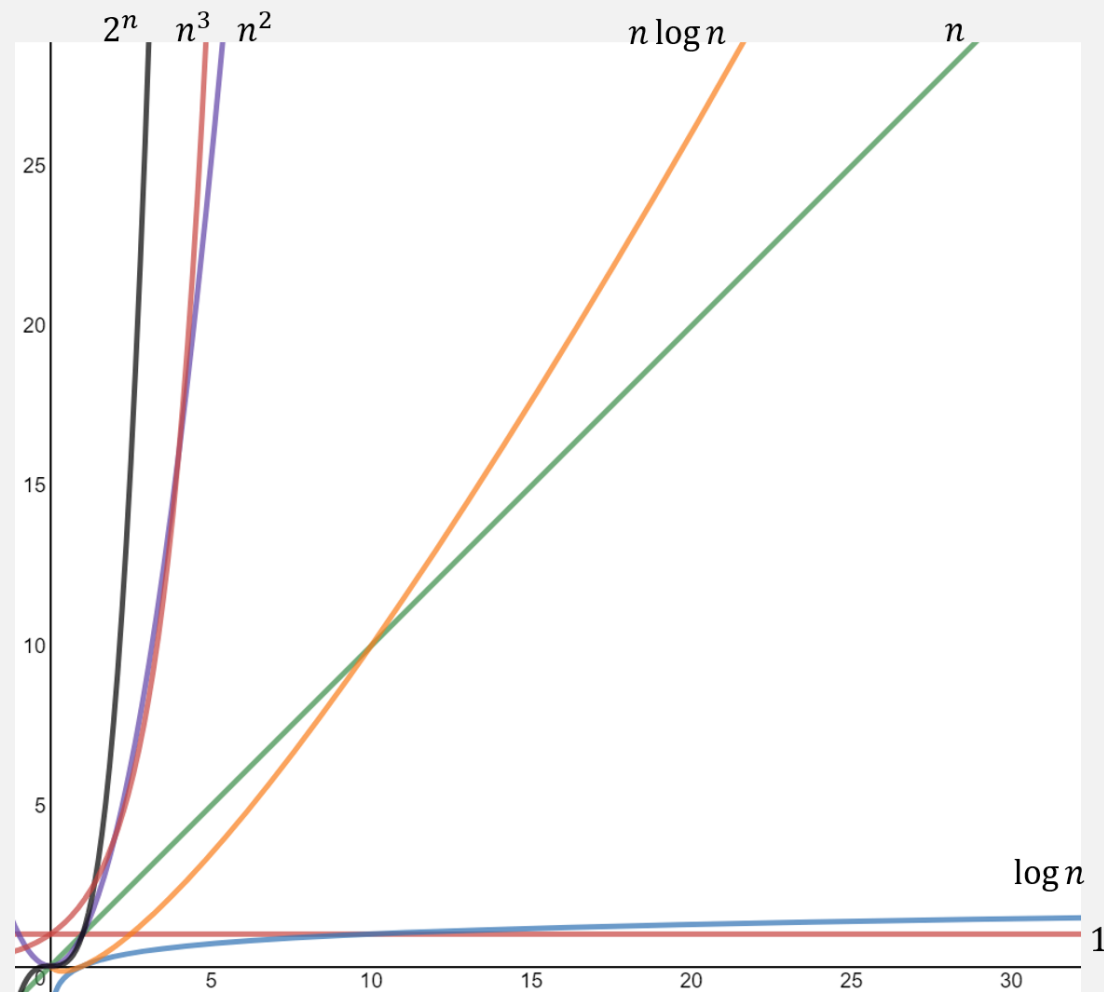
1년은 31,536,000초  $> 10^7$ 초,  $\frac{10^{28}}{10^7} = 10^{21}$ 년, 우주의 연령 = 137억년  $\rightarrow 10^{11}$ 년



## 복잡도

### 시간복잡도와 공간복잡도

- 시간 복잡도(Time Complexity) :  
알고리즘이 어떤 문제를 해결하는데 걸리는 시간 (연산의 계산량)
- 공간 복잡도(Space Complexity) :  
알고리즘이 어떤 문제를 해결하는데 필요한 공간 (자원의 양)
- Big-O표기법(O-notation) :  
점근 상향을 나타내기 위해 사용하는 표기법



### 복잡도

## 공간복잡도

### 1) 1차원 배열을 사용한 경우

```
private static int get_sum(int[] arr, int n) {  
    int sum = 0;  
    for (int i = 0; i < n; i++) {  
        sum += arr[i];  
    }  
  
    return sum;  
}
```

공간복잡도 :  $n + 3$

### 2) 2차원 배열을 사용한 경우

```
private static int get_sum(int[][] arr, int n, int m) {  
    int sum = 0;  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < m; j++) {  
            sum += arr[i][j];  
        }  
    }  
  
    return sum;  
}
```

공간복잡도 :  $n * m + 5$

### 복잡도

## 시간복잡도 - 1

### 1) 1차원 배열을 사용한 경우

```
private static int get_sum(int[] arr, int n) {  
    int sum = 0;  
    for (int i = 0; i < n; i++) {  
        sum += arr[i];  
    }  
  
    return sum;  
}
```

시간복잡도 :  $n$

### 2) 2차원 배열을 사용한 경우

```
private static int get_sum(int[][] arr, int n, int m) {  
    int sum = 0;  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < m; j++) {  
            sum += arr[i][j];  
        }  
    }  
  
    return sum;  
}
```

시간복잡도 :  $n^2$



복잡도

시간복잡도 - 2

1차원 list에서 증감 값이 있을 경우

```
private static int get_sum(int[] arr, int n, int m) {  
    int sum = 0;  
    for (int i = 1; i <= n; i *= 2) {  
        sum += arr[i];  
    }  
  
    return sum;  
}
```

시간복잡도 : log(n)

n의 값	n의 값에 따른 i의 값	연산 횟수
n=1	i=1	1
n=2	i=1,2	2
n=3	i=1,2	2
n=4	i=1,2,4	3
n=5	i=1,2,4	3
n=6	<b>i=1,2,4</b>	3
n=7	<b>i=1,2,4</b>	3
n=8	<b>i=1,2,4,8</b>	4
...	...	...
n=16	<b>i=1,2,4,8,16</b>	5

복잡도

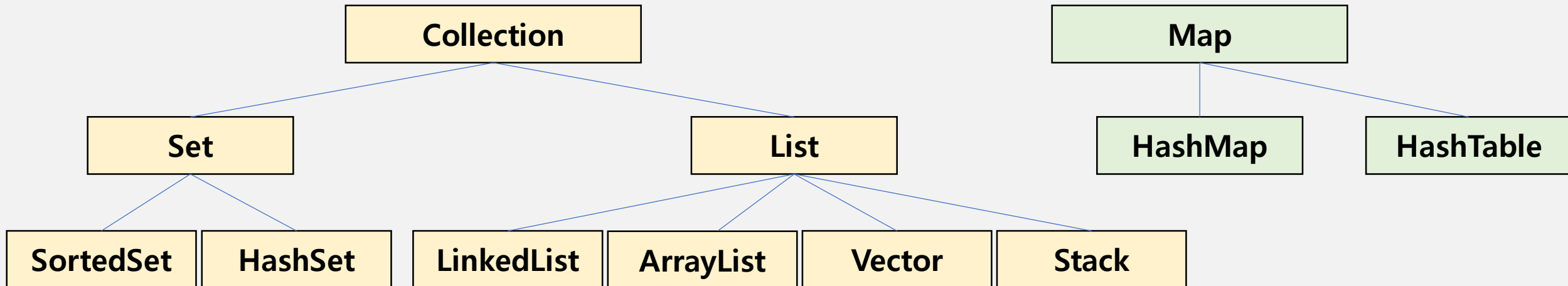
시간복잡도 - 3

분류 / n의 숫자(데이터)	1	4	8	16	32	예시
$O(1)$ - 상수형	1	1	1	1	1	반복이 없는 경우
$O(n)$ - 선형	1	4	8	16	32	선형 탐색, 단순 탐색
$O(\log n)$ - 로그형	0	2	3	4	5	이진 탐색
$O(n \log n)$ - 선형 로그형	0	8	24	64	160	퀵 정렬(Quick Sort)
$O(n^2)$ - 평방형	1	16	64	256	1024	이중 반복, 선택 정렬(Selection Sort)
$O(n^3)$ - 입방형	1	64	512	4096	32768	삼중 반복
$O(2^n)$ - 지수형	2	16	256	65536	4294967296	
$O(n!)$ - 계승형	1	24	40320	...	...	외판원 문제

### Collection

## Collection Framework

- 널리 알려진 여러 가지 자료 구조를 자바에서 미리 구현하여 제공하는 패키지
- 요소를 수집해서 저장하는 것을 의미하며 객체를 수집해서 저장하는 역할을 함
- 프레임워크(Framework): 사용 방법을 미리 정해 놓은 라이브러리
- 크게 Collection계열과 Map계열로 나뉨



Collection

Collection 계열 클래스의 기능 (메소드)

인터페이스 분류		설명	구현하는 클래스
Collection	List	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList
	Set	- 순서를 유지하지 않고 저장 - 중복 저장 안 됨	HashSet, TreeSet
Map		- 키와 값의 쌍으로 저장 - 키는 중복 저장 불가	HashMap, Hashtable, TreeMap, Properties

메소드	설명
boolean add(E e)	객체(데이터)를 collection에 추가
void clear()	
boolean contains(Object o)	현재 collection에 인자로 전달된 데이터가 있는지 판별
boolean isEmpty()	현재 collection에 데이터가 있는지 검사
Iterator<E> iterator()	현재 collection의 객체를 iterator로 반환
boolean remove(Object o)	현재 collection에 인자로 전달된 객체를 제거
int size()	현재 collection의 데이터 개수를 반환
Object[] toArray()	현재 collection이 가지고 있는 객체들을 배열로 만들어 반환

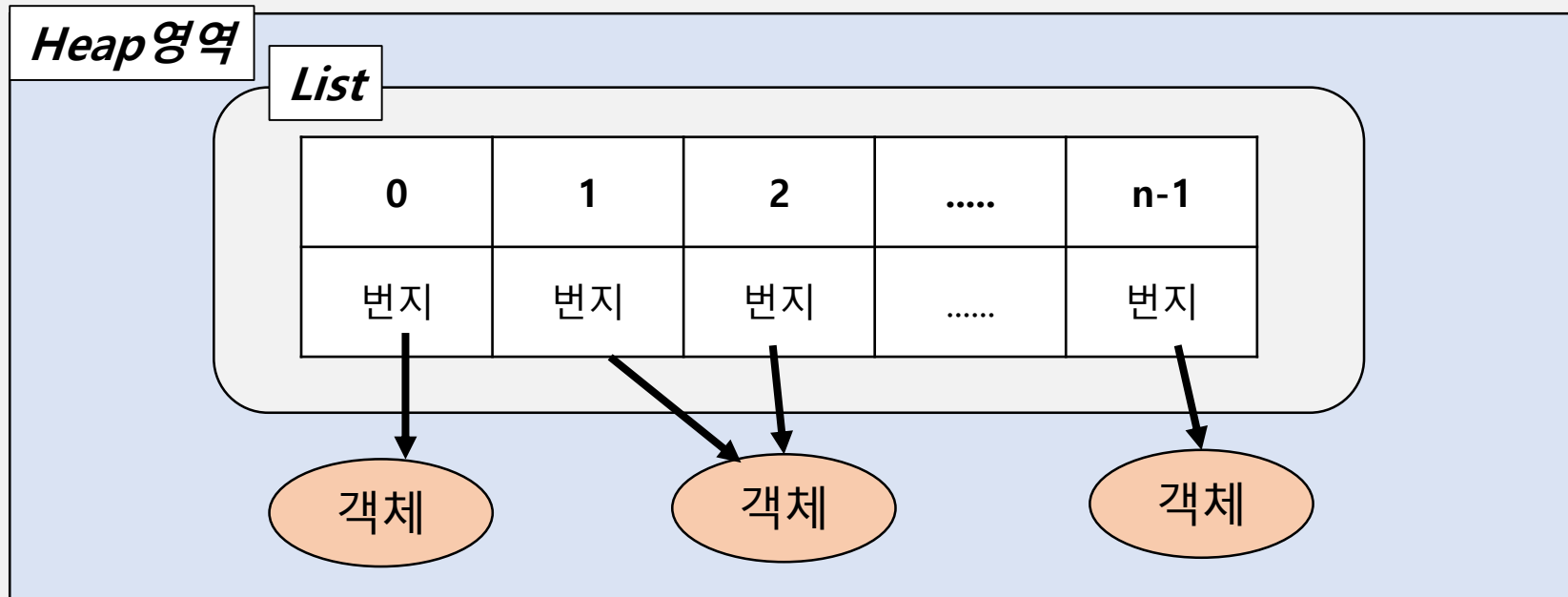
## List

### List Collection

- 객체를 일렬로 늘어놓은 구조를 가지며, 객체를 index로 관리하기 때문에 객체를 저장하면 자동으로 index가 부여됨
- 객체 자체를 저장하는 것이 아니라 객체의 번지를 참조
- ArrayList: List 인터페이스의 구현 클래스. 동적 배열이라는 점에서 일반 배열과 큰 차이점을 가짐
- Vector: ArrayList와 동일한 내부 구조를 가지지만 동기화된 메소드로 구성되어 있어 Thread Safe하다는 특징이 있음

```
ArrayList<E> array = new ArrayList<E>(크기)
```

```
Vector<E> v = new Vector<E>();
```



List

리스트와(ArrayList) 연결 리스트(Linked list)

- 리스트 : 원소 값들을 메모리에 차례대로 저장 (임의 접근)
- 연결 리스트(Linked list) : 원소를 메모리에 저장하고 원소에 다음 원소의 주소 값을 저장하여 원소들을 연결 (순차 접근)

리스트의 장점 : 어떤 원소이든지 한번에 찾아 사용이 가능 (원소의 탐색) –  $O(1)$   
리스트의 단점 : 원소의 추가/삭제가 어려움 –  $O(n)$

연결 리스트의 장점 : 원소의 추가/삭제가 쉬움 –  $O(1)$   
연결 리스트의 단점 : 마지막 원소를 찾기 위해서는 첫 원소부터 순서대로 이동해야 함 (원소의 탐색) –  $O(n)$



List

리스트와(ArrayList) 연결 리스트(Linked list)

List에 새로운 요소를 추가할 경우

브런치 먹기	차 마시기	음악 듣기	추가불가!



브런치 먹기	차 마시기	음악 듣기	책 읽기

메모리의 빈 공간으로 이동



요소를 추가, 삽입, 삭제 할 때마다 메모리에서 최적의 장소로 이동해야 함!  
하지만 요소를 탐색할 때는 해당 list의 주소만 알면 바로 해당 요소로 이동 가능

List

리스트와(ArrayList) 연결 리스트(Linked list)

Linked list에 새로운 요소를 추가할 경우

브런치 먹기			
		차 마시기	
음악 듣기			



다음 값의 메모리 주소를 저장

브런치 먹기			
		차 마시기	
음악 듣기			
			책 읽기

각 요소는 다음 요소의 메모리 주소를 저장하여 메모리의 빈 공간 어디든 값을 저장 가능하고 요소의 삭제도 편함  
하지만 요소를 탐색할 때는 처음부터 찾아가야 함!



List

자바의 연결 리스트(Linked list) – 1

- List를 구현하면서 Queue 인터페이스도 함께 구현되어 있음
- List의 첫번째, 혹은 마지막 요소에 값을 추가하고 삭제하는 것이 메소드를 통해 가능

```
import java.util.LinkedList;
LinkdedList<WrapperClass> linked_list = new LinkedList<WrapperClass>();
```

메소드명	설명
add(값)	List에 값을 추가
contains(값)	List에 해당 값이 있는지 검사하고 boolean값을 반환
getFirst() gerLast() get(index번호)	List의 첫 번째 요소를 반환 List의 마지막 요소를 반환 List의 index위치의 값을 반환
peek()	List에 가장 처음에 넣은 요소의 값을 반환
poll()	List에 가장 처음에 넣은 요소를 제거하고 값을 반환
remove(값 혹은 index)	List에서 값/index위치의 값을 제거
size()	List의 길이를 반환

## List

### 자바의 연결 리스트(Linked list) – 2

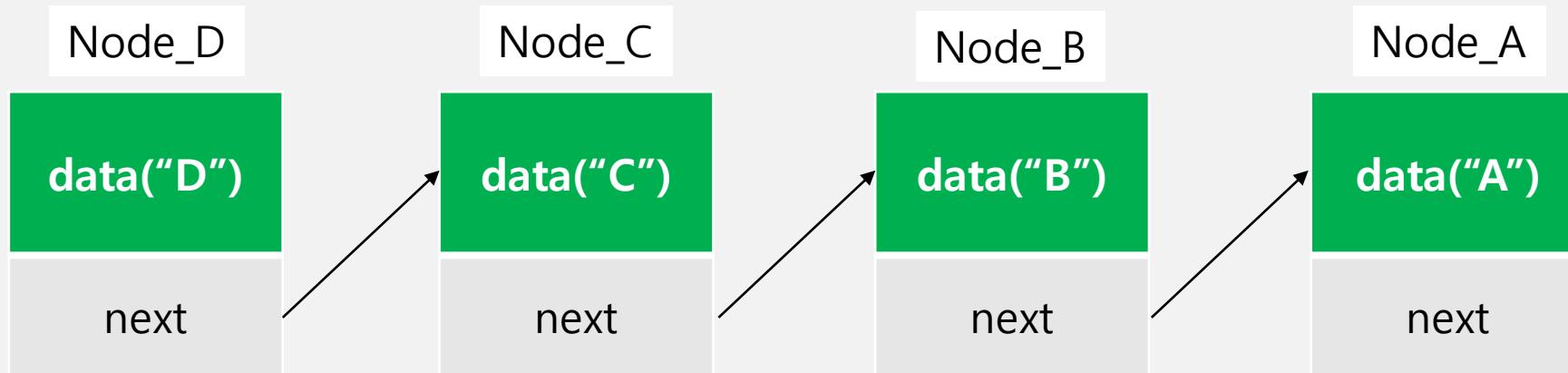
- 연결 리스트는 노드 Node를 사용해 class로 구현가능

#### Node의 구현

```
class Node{  
    Object data;  
    Node address;  
    Node(Object data, Node address){  
        this.data = data;  
        this.address = address;  
    }  
}
```

#### Node의 생성과 연결

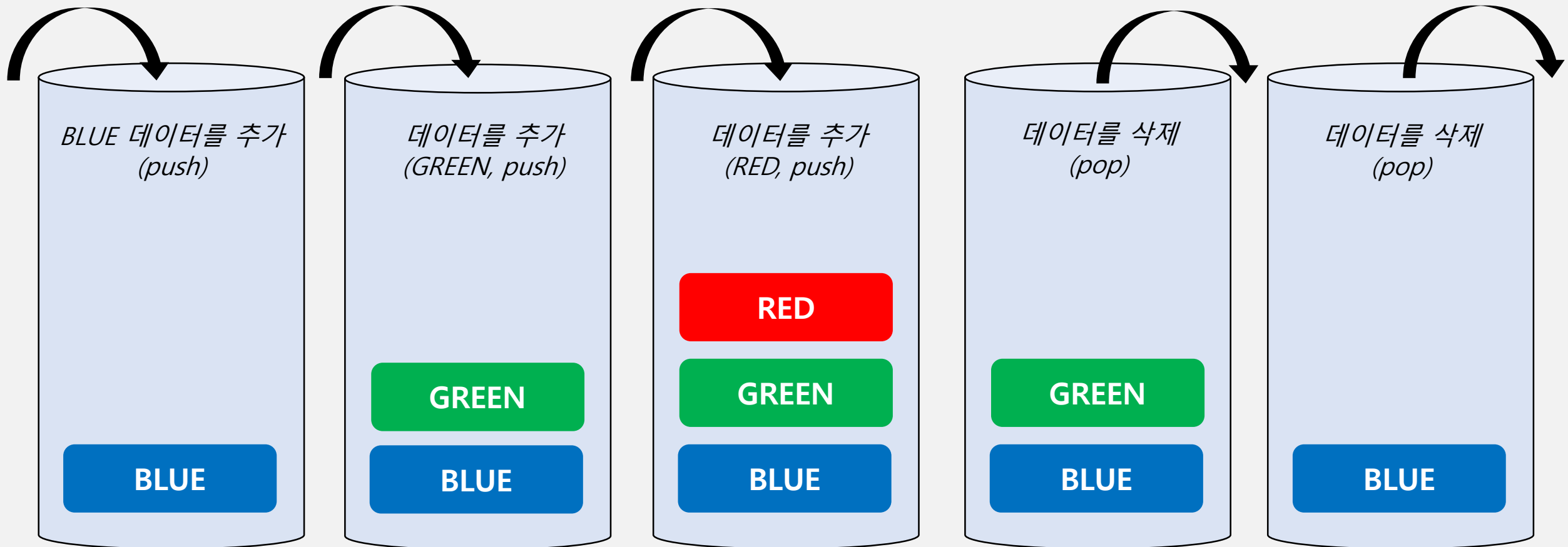
```
Node nodeA = new Node("A", null);  
Node nodeB = new Node("B", nodeA);  
Node nodeC = new Node("C", nodeB);  
Node nodeD = new Node("D", nodeC);
```



### 스택과 큐

#### 스택(Stack)

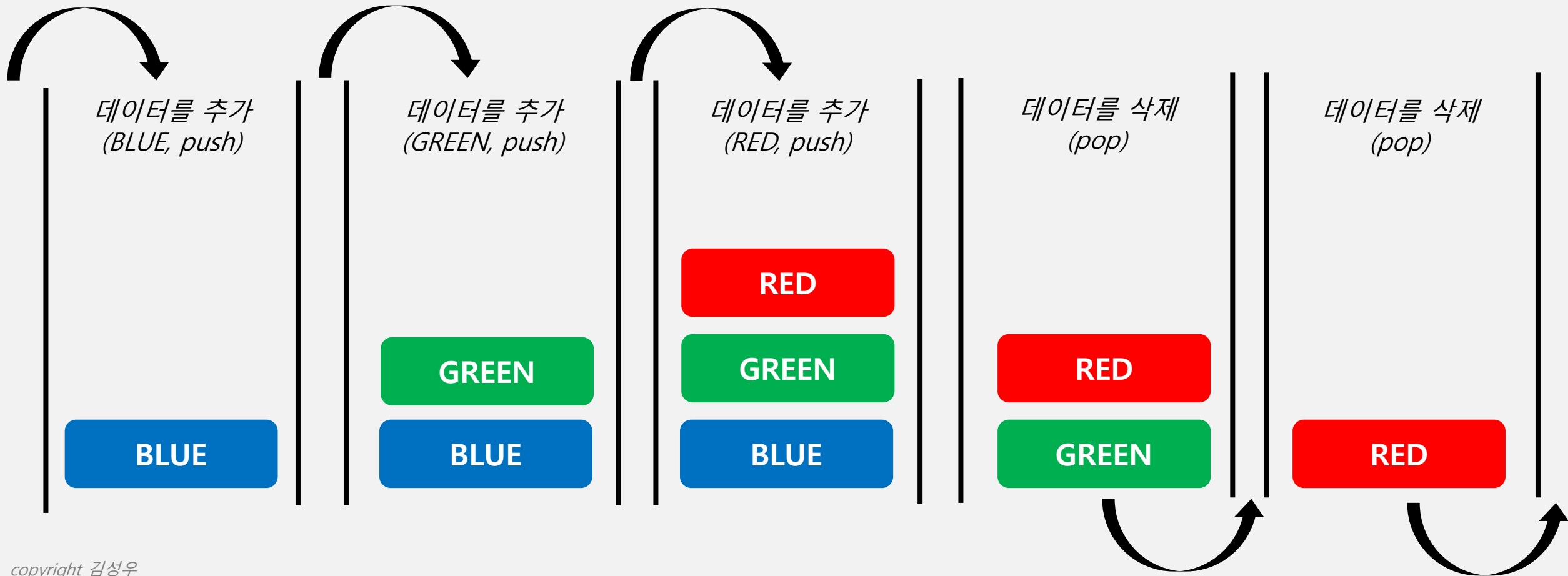
- 데이터를 삽입 할 때 기존의 데이터 위에 쌓고, 데이터를 꺼낼 때 가장 위에 있는 데이터부터 꺼내서 사용하는 자료구조 방식
- 후입선출(Last In First Out : LIFO) 구조를 가지며 데이터 삽입과 삭제가 단방향임



스택과 큐

큐(Queue)

- 데이터를 삽입 할 때 기존의 데이터 위에 쌓고, 데이터를 꺼낼 때 가장 아래에 있는 데이터부터 꺼내서 사용하는 자료구조 방식
- 선입선출(First In First Out : FIFO) 구조를 가지며 데이터 삽입과 삭제가 단방향임



스택과 큐

자바의 Stack/Queue

- 예외 발생 : add(), element(), remove()
  - 예외가 발생하지 않음 : offer(), peek(), poll()
- ```
Stack<Wrapper Class> stack = new Stack<Wrapper>();
Queue<Wrapper Class> queue = new LinkedList<Wrapper>();
```

| 자료구조  | 메소드명             | 반환 타입   | 설명                                           |
|-------|------------------|---------|----------------------------------------------|
| Stack | empty()          | boolean | Stack이 비어 있는지 알려줌                            |
|       | peek()           | Object  | stack의 가장 마지막에 추가된 요소를 반환                    |
|       | pop()            | Object  | Stack의 가장 마지막에 추가된 요소를 꺼냄                    |
|       | push(값)          | Object  | Stack에 값을 추가                                 |
|       | search(Object o) | int     | Stack에서 주어진 객체[o]의 위치를 반환, 객체가 안에 없다면 -1을 반환 |
| Queue | offer(값)         | boolean | Queue에 값을 추가                                 |
|       | peek()           | Object  | Queue의 가장 빨리 추가된 요소 반환                       |
|       | poll()           | Object  | Queue의 가장 빨리 추가된 요소를 반환 후 제거                 |

### 스택과 큐

## 팬린드롬 문자열(Palindrome)

- 팬린드롬(Palindrome) : 앞 뒤가 똑같은 단어나 문장으로, 뒤집어도 같은 말이 되는 단어 혹은 문장  
ex) "다시 합창합시다", "다 이심전심이다", "race car"

문자열을 사용자에게 입력 받아 해당 문자열이 팬린드롬인지 확인하라. (대소문자를 구분하지 않고 영문, 숫자 대상) 문자열이 팬린드롬이면 True, 아니면 False를 출력.

```
private static boolean isPanlindrome(String s) {
    char[] chars = s.toCharArray();
    LinkedList<Character> queue = new LinkedList<Character>();

    for (int i = 0; i < chars.length; i++) {
        if(chars[i] != ' ') {
            queue.offer(chars[i]);
        }
    }
    while(queue.size() > 1) {
        if(queue.pollFirst() != queue.pollLast()) {
            return false;
        }
    }
    return true;
}
```

실전 문제 1

<https://programmers.co.kr/learn/courses/30/lessons/12909>

괄호 변환

'(' 또는 ')' 로만 이루어진 문자열 *s*를 사용자에게 입력 받고,  
문자열 *s*가 올바른 괄호이면 **true**를 출력 하고, 올바르지 않은 괄호이면 **false**를 출력하세요.

- `"()"` 또는 `"(())"` 는 올바른 괄호입니다.
- `")()("` 또는 `"(]("` 는 올바르지 않은 괄호입니다.

| s      | answer |
|--------|--------|
| "()"   | true   |
| "(())" | true   |
| ")()(" | false  |
| "(]("  | false  |

Iterator

Iterator (반복자)

- Collection 클래스에 저장된 데이터를 일정한 방법으로 접근할 수 있도록 함
- 각 컬렉션 클래스에 저장할 때처럼 가져올 때도 일관성 있게 가져오자는 개념에서 나온 클래스
- Iterator는 모든 Collection 클래스에서 사용이 가능

Iterator iterator = <V>.iterator();

| 클래스명     | 메소드명      | 반환 타입   | 설명                                          |
|----------|-----------|---------|---------------------------------------------|
| Iterator | hasNext() | boolean | iteration이 element를 가지고 있는지 판별              |
|          | next()    | E       | iteration의 다음 element를 반환                   |
|          | remove()  | void    | iteration이 마지막으로 반환한 element다음의 element를 제거 |



Set

Set - HashSet

- Set은 집합이라는 뜻으로, 요소들을 집합적으로 모아놓은 자료구조를 의미
- 중복된 데이터를 가지지 않으며 저장 순서를 유지하지 않는다는 특징이 있음 (null도 하나만 저장 가능)

```
HashSet<E> 변수명 = new HashSet<>(); //순서가 없음
HashSet<E> 변수명 = new LinkedHashSet<>(); //순서를 유지
```

| 메소드명        | 반환 타입    | 설명                                                                |
|-------------|----------|-------------------------------------------------------------------|
| add(값)      | boolean  | HashSet에 값을 추가. 해당 값이 이미 존재한다면 추가하지 않고 <b>false</b> 를 반환          |
| remove(값)   | boolean  | 해당 값이 HashSet에 존재한다면 삭제 후 <b>true</b> 를 반환, 없다면 <b>false</b> 를 반환 |
| clear()     | -        | HashSet에 있는 모든 값을 제거                                              |
| size()      | int      | HashSet의 데이터 크기[개수]를 반환                                           |
| next()      | object   | HashSet의 다음 데이터를 가져옴                                              |
| hasNext()   | Boolean  | HashSet안에 데이터가 있다면 <b>true</b> 를, 없다면 <b>false</b> 를 반환           |
| iterator()  | iterator | Iterator 반복자를 사용하여 데이터를 순회                                        |
| contains(값) | Boolean  | HashSet안에 해당 값이 있다면 <b>true</b> , 없다면 <b>false</b> 를 반환           |

### Set

## HashSet

```
HashSet<String> hashSet = new HashSet<>();  
hashSet.add("demon");  
hashSet.add("banana");  
hashSet.add("tomato");  
hashSet.add("apple");  
hashSet.add("cargo");  
  
hashSet.add("apple");  
hashSet.add("banana");  
  
Iterator iterator = hashSet.iterator();  
while(iterator.hasNext()) {  
    System.out.println(iterator.next());  
}
```

banana  
apple  
demon  
tomato  
cargo

```
HashSet<String> hashSet = new LinkedHashSet<>();  
  
hashSet.add("demon");  
hashSet.add("banana");  
hashSet.add("tomato");  
hashSet.add("apple");  
hashSet.add("cargo");  
  
hashSet.add("apple");  
hashSet.add("banana");  
  
Iterator iterator = hashSet.iterator();  
while(iterator.hasNext()) {  
    System.out.println(iterator.next());  
}
```

demon  
banana  
tomato  
apple  
cargo

### Set

## HashSet

hashCode()와 equals()를 오버라이딩하여  
같은 객체로 인식하게 만들기

```
public class Test {  
    public static void main(String[] args) {  
        Set<Member> set = new HashSet<>();  
        set.add(new Member("홍길동", 30));  
        set.add(new Member("홍길동", 30));  
  
        System.out.println("총 객체 수: " + set.size());  
    }  
}
```

```
import java.util.HashSet;  
import java.util.Set;  
  
class Member{  
    public String name;  
    public int age;  
  
    public Member(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        if (obj instanceof Member) {  
            Member member = (Member) obj;  
            return member.name.equals(name) && (member.age == age);  
        }  
  
        return false;  
    }  
  
    @Override  
    public int hashCode() {  
        return name.hashCode() + age;  
    }  
}
```

Map

Map - HashMap

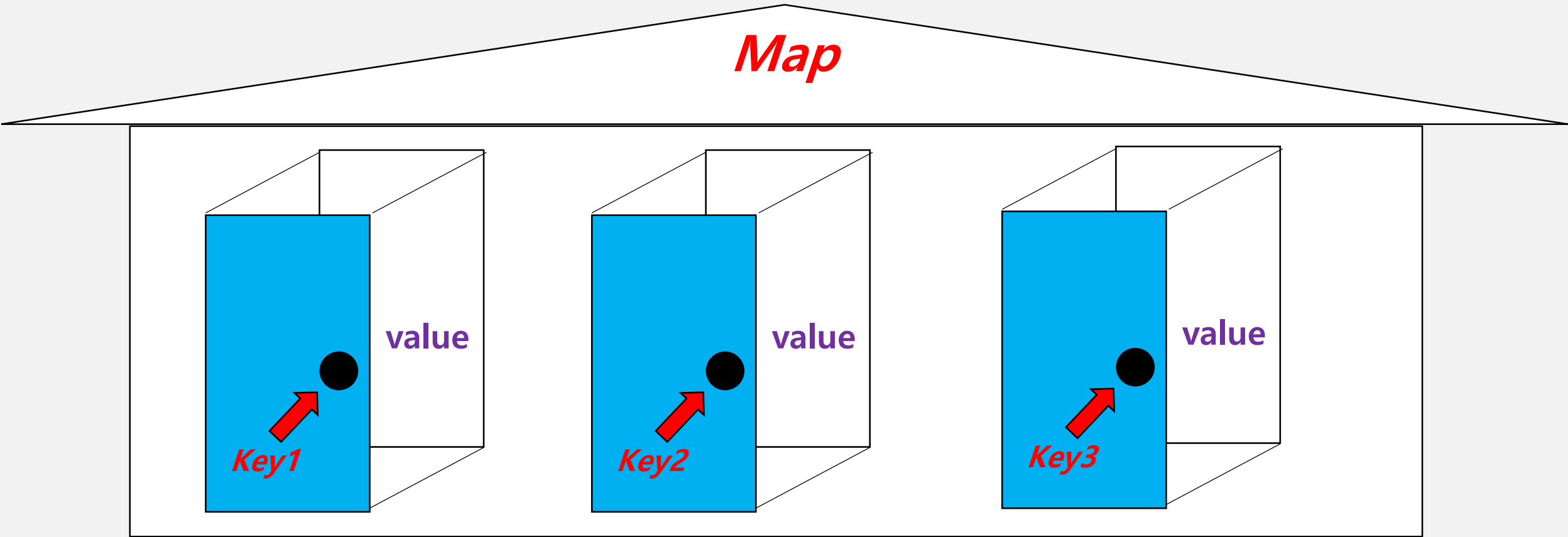
- Map 인터페이스는 키와 값을 쌍으로 저장하는 구조로, key값은 중복이 허용되지 않지만 value 값은 중복이 허용됨
- HashMap은 내부적으로 해싱(Hashing) 검색을 사용하기 때문에 대용량 데이터관리에 성능이 좋음

HashMap<Key, Value> hashMap = new HashMap<>();

| 메소드명              | 반환 타입         | 설명                                      |
|-------------------|---------------|-----------------------------------------|
| containsKey(값)    | boolean       | HashMap 안에 해당 key가 있는지 여부를 반환           |
| containsValue(값)  | boolean       | HashMap 안에 해당 value가 있는지 여부를 반환         |
| get(key)          | object        | key의 쌍인 value값을 반환                      |
| values()          | Collection<V> | HashMap에 존재하는 value를 Collection 타입으로 반환 |
| Set<key> KeySet() | Set           | 키들을 Set형태로 반환                           |
| put(key, value)   |               | key와 value 한 쌍을 HashMap에 추가             |
| remove(key)       |               | HashMap에서 해당 key를 가진 쌍을 제거              |
| size()            | int           | HashMap 전체 요소의 개수를 반환                   |

Map

Map



### Map

## HashMap

```
Scanner input = new Scanner(System.in);
HashMap<String, Integer> hashMap = new HashMap<>();

hashMap.put("종이", 100);
hashMap.put("연필", 1000);
hashMap.put("가위", 3000);

System.out.println("가격을 알고싶은 물건을 입력하세요 >>> ");
String object = input.next();
if(hashMap.containsKey(object)) {
    int price = hashMap.get(object);
    System.out.println(object + "는 " + price + "원 입니다.");
}
else {
    System.out.println("해당 물건은 존재하지 않습니다.");
}
```

### Map

## Map - Hashtable과 Properties

- **Hashtable**: HashMap과 동일한 내부 구조를 가지며, 역시 hashCode(), equals() 메소드로 동일한 Key값을 판단함
- 동기화된 메소드로 구성되어 있기 때문에 스레드 안전하다는 차이점이 있음
- **Properties**: Hashtable의 하위 클래스로 Hashtable의 모든 특징을 가지고 있으며, Key와 Value값이 String로 제한이 있음
- Application의 옵션 정보, DB연결 정보, 다국어 정보가 저장된 프로퍼티(.properties)파일을 읽을 때 주로 사용
- Properties로 만든 정보는 ISO 8859-1문자셋으로 저장되며 한글은 유니코드로 자동 변환됨

```
Map<K, V> map = new Hashtable<K, V>();
```

```
Properties property = new Properties();
```

"database.properties"키=값으로 구성된 프로퍼티

```
driver=oracle.jdbc.OracleDriver  
url=jdbc:oracle:thin:@localhost:1521:orcl  
username=scott  
password=tiger
```

실전 문제 2

<https://programmers.co.kr/learn/courses/30/lessons/42576>

완주하지 못한 선수

수많은 마라톤 선수들이 마라톤에 참여하였습니다. 단 한 명의 선수를 제외하고는 모든 선수가 마라톤을 완주하였습니다. 마라톤에 참여한 선수들의 이름이 담긴 배열 participant와 완주한 선수들의 이름이 담긴 배열 completion이 주어질 때, 완주하지 못한 선수의 이름을 출력하세요.

- 마라톤 경기에 참여한 선수의 수는 1명 이상 100,000명 이하입니다.
- completion의 길이는 participant의 길이보다 1 작습니다.
- 참가자의 이름은 1개 이상 20개 이하의 알파벳 소문자로 이루어져 있습니다.
- 참가자 중에는 동명이인이 있을 수 있습니다.

| participant                                       | completion                               | 출력       |
|---------------------------------------------------|------------------------------------------|----------|
| ["leo", "kiki", "eden"]                           | ["eden", "kiki"]                         | "leo"    |
| ["marina", "josipa", "nikola", "vinko", "filipa"] | ["josipa", "filipa", "marina", "nikola"] | "vinko"  |
| ["mislav", "stanko", "mislav", "ana"]             | ["stanko", "ana", "mislav"]              | "mislav" |



## Tree

### 트리 (Tree)

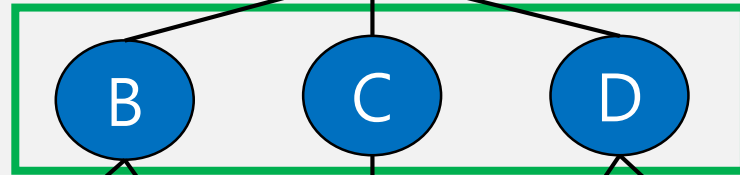
- 루트 노드를 시작으로 줄기를 뺀어 나가듯 원소를 추가해 나가는 비선형 구조
- 방향성이 있고 노드와 노드 간에 간선이 존재하여 간선으로 연결됨
- 차수 (Degree) : 노드의 하위 간선의 개수 (자식의 개수 = 간선의 개수)
- 깊이 (depth) : 루트에서 어떤 루트까지의 경로 길이

LEVEL1



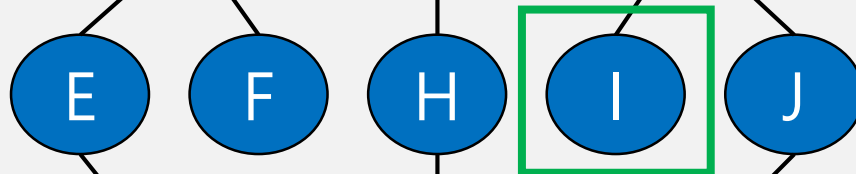
루트 노드 : 가장 꼭대기에 있는 노드

LEVEL2



자식 노드 : 노드 하위에 인접한 노드

LEVEL3



단말 노드 : 자식이 없는 노드

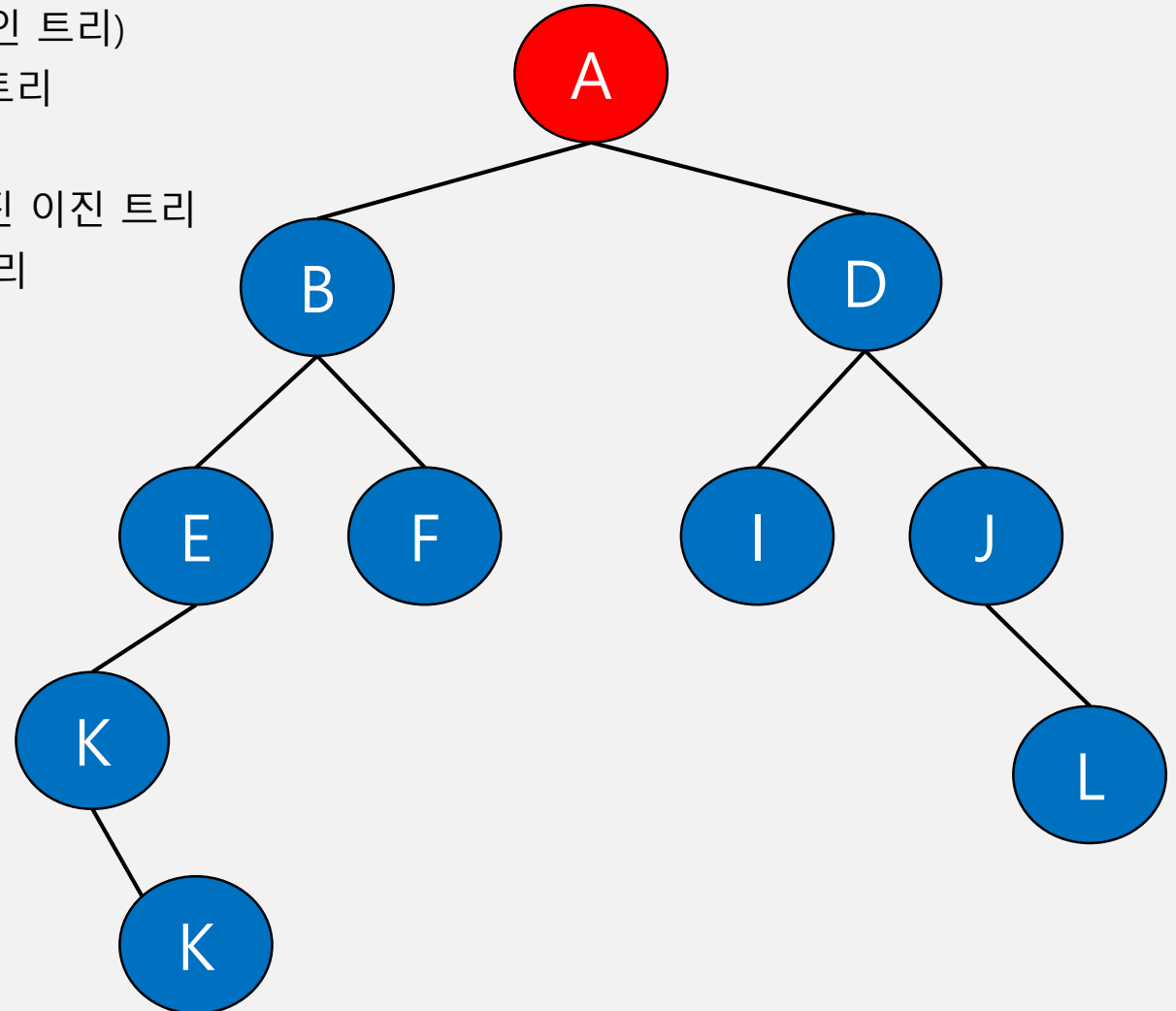
LEVEL4



### Tree

## 이진 탐색 트리 (BST : Binary Search Tree)

- 트리의 최대 차수가 2인 트리 (모든 노드의 자식이 최대 2개인 트리)
- 부모 노드보다 작으면 왼쪽, 크면 오른쪽 자식 노드가 되는 트리
- 중복된 데이터를 허용하지 않음
- 완전 이진 트리 : 마지막 노드를 제외하고 모든 노드가 채워진 이진 트리
- 포화 이진 트리 : 모든 단말 노드의 깊이가 같은 완전이진 트리
- 균형 이진 트리 : 좌우의 높이의 차이가 같거나 최대 1인 트리



Tree

Set – TreeSet

- 중복 불가, 순서가 없고 데이터 정렬 기능을 제공함
- 이진 탐색 트리의 구조로 이루어져 있기 때문에 데이터 검색에 유리하다는 특징이 있음

| 메소드명                           | 반환 타입        | 설명                                                                                 |
|--------------------------------|--------------|------------------------------------------------------------------------------------|
| ceiling(E e)                   | E            | 인자로 전달되는 <b>element</b> 보다 작은 <b>element</b> 를 반환 (바로 위)                           |
| descendingIterator()           | Iterator<E>  | <b>element</b> 들을 <b>set</b> 타입으로 정렬하는데 사용되는 <b>Comparator</b> 객체를 반환              |
| first()                        | E            | <b>set</b> 에 있는 가장 첫 번째 <b>element</b> 를 반환 (가장 낮은)                                |
| floor(E e)                     | E            | 인자로 전달되는 <b>element</b> 보다 큰 <b>element</b> 중 가장 큰 <b>element</b> 를 반환 (바로 아래)     |
| higher(E e)                    | E            | 인자로 전달되는 <b>element</b> 보다 큰 <b>element</b> 중 가장 작은 <b>element</b> 를 반환 (바로 위)     |
| last()                         | E            | <b>set</b> 에서 가장 마지막 <b>element</b> 를 반환 (가장 높은)                                   |
| tailSet(E fElement)            | SortedSet<E> | <b>set</b> 에서 <b>fElement</b> 보다 크거나 같은 <b>element</b> 들을 정렬한 후 <b>set</b> 타입으로 반환 |
| subSet(E fElement, E tElement) | SortedSet<E> | <b>fElement</b> 와 <b>tElement</b> 사이의 <b>element</b> 들을 정렬한 후 <b>set</b> 타입으로 반환   |

### Tree

## Set – TreeSet

```
public static void main(String[] args) {  
    TreeSet ts = new TreeSet();  
    ts.add("홍길동");  
    ts.add("차범근");  
    ts.add("유재석");  
    ts.add("박명수");  
    ts.add("김유신");  
    ts.add("홍길동");  
  
    Iterator ite = ts.iterator();
```

```
        System.out.println("오름차순 정렬");  
        while(ite.hasNext()) {  
            System.out.println(ite.next());  
        }  
  
        System.out.println("\n내림차순 정렬");  
        Iterator ite2 = ts.descendingIterator();  
        while(ite2.hasNext()) {  
            System.out.println(ite2.next());  
        }  
  
        System.out.println(ts);  
    }
```

Tree

Map – TreeMap

- 저장 시 내부적으로 key를 오름차순 정렬하여 저장. 저장되는 것은 Key, Value 쌍을 가지는 MapEntry객체
- 이진 탐색 트리의 구조로 이루어져 있기 때문에 데이터 검색에 유리하다는 특징이 있음

| 메소드명                 | 반환 타입           | 설명                                                                       |
|----------------------|-----------------|--------------------------------------------------------------------------|
| ceilingKey(K key)    | K               | 인자로 전달되는 <b>element</b> 보다 크거나 같은 <b>key</b> 중 가장 작은 <b>key</b> 를 반환     |
| firstKey()           | K               | <b>key</b> 에 있는 가장 첫 번째(가장 작은) <b>key</b> 를 반환                           |
| firtstEntry()        | Map.Entry<K, V> | <b>key</b> 에 있는 가장 첫 번째(가장 작은) <b>key</b> 를 반환                           |
| descendingKeySet()   | NavigableSet<K> | <b>map</b> 에 존재하는 <b>key</b> 의 역순으로 된 <b>NavigableSet</b> 을 반환           |
| headMap(K toKey)     | SortedMap<K,V>  | <b>map</b> 에서 <b>toKey</b> 보다 작은 <b>key</b> 들을 정렬한 후 반환                  |
| lastKey()            | K               | <b>key</b> 에서 가장 마지막 <b>element</b> 를 반환                                 |
| lowerKey(K key)      | K               | <b>map</b> 에서 인자로 전달된 <b>key</b> 보다 작은 <b>key</b> 중 가장 큰 <b>key</b> 를 반환 |
| subMap(K fK, K tKey) | SortedMap<K,V>  | <b>map</b> 에서 <b>fKey</b> 와 <b>tKey</b> 사이의 키 값을 가지는 <b>key</b> 들을 반환    |
| tailMap(K fKey)      | SortedMap<K,V>  | <b>map</b> 에서 <b>fKey</b> 보다 크거나 같은 <b>key</b> 들을 반환                     |

정렬

Comparable / Comparator

- 정렬 인터페이스이며 Integer, Double, String 등 많은 클래스들이 Comparable 인터페이스를 구현함
- TreeSet, TreeMap에 객체를 저장할 때는 java.lang.Comparable을 구현한 객체가 필요

| 메소드명           | 반환 타입 | 설명                                                 |
|----------------|-------|----------------------------------------------------|
| compareTo(T o) | int   | 주어진 객체와 같으면 0<br>주어진 객체보다 작으면 음수<br>주어진 객체보다 크면 양수 |

| 메소드명                | 반환 타입 | 설명                                                                          |
|---------------------|-------|-----------------------------------------------------------------------------|
| compare(T o1, T o2) | int   | o1과 o2가 같다면 0을 반환<br>o1이 o2보다 앞에 오게 하려면 음수를 반환<br>o1이 o2보다 뒤에 가게 하려면 양수를 반환 |

### 정렬

## Comparable

```
class Snack implements Comparable<Snack> {
    String name;
    int price;

    public Snack(String name, int price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public int compareTo(Snack o) {
        if(price < o.price) return 1;
        else if(price == o.price) return 0;
        else return -1;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        TreeSet<Snack> tree = new TreeSet<Snack>();

        tree.add(new Snack("감자칩", 2000));
        tree.add(new Snack("초코칩", 1000));
        tree.add(new Snack("맛동산", 4000));

        for (Snack snack : tree) {
            System.out.println(snack.name);
            System.out.println(snack.price);
        }
    }
}
```

### 정렬

## Comparator

```
class DescendingComparator implements Comparator<Fruit>{
    @Override
    public int compare(Fruit o1, Fruit o2) {
        if(o1.price < o2.price) return 1;
        else if(o1.price == o2.price) return 0;
        else return -1;
    }
}

class Fruit{
    String name;
    int price;

    public Fruit(String name, int price) {
        this.name = name;
        this.price = price;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        ArrayList<Fruit> arr = new ArrayList<Fruit>();
        arr.add(new Fruit("사과", 2000));
        arr.add(new Fruit("딸기", 1000));
        arr.add(new Fruit("포도", 4000));

        arr.sort(new DescendingComparator());
        for (Fruit fruit : arr) {
            System.out.println(fruit.name);
            System.out.println(fruit.price);
        }
    }
}
```



### 실전 문제 3

아래 지시를 읽고 입력하는 임의의 문자열  $s$ 에 따라 결과를 출력하세요.

괄호는 아래와 같이 네 종류가 있다고 가정합니다

- $()$ ,  $\{\}$ ,  $[]$ ,  $< >$

동일한 형태의 괄호가 여러 번 사용될 수 있습니다.

괄호를 정상적으로 사용했는지 검증한 결과를 출력합니다.

#### 지시사항

인덱스는 0부터 시작합니다.

여닫는 괄호의 짝이 맞지 않으면 닫는 괄호의 인덱스를 음수로 출력합니다.

괄호가 열려 있는 상태로 문자열이 끝나면 문자열의 마지막 인덱스를 음수로 출력합니다.

답이 중복으로 존재하는 경우 문자열 왼쪽 기준으로 먼저 등장하는 것을 출력합니다.

모든 괄호를 정상적으로 사용했다면 총 괄호 쌍의 개수를 출력합니다.

첫 번째 문자가 닫는 괄호이거나 괄호가 없는 경우에는 0을 출력합니다.

두 번째 문자 이후에서 닫는 괄호가 먼저 나오는 경우에는 닫는 괄호의 인덱스를 음수로 출력합니다.

예를 들어.

연습 문제

아래 지시를 읽고 입력하는 임의의 문자열 s에 따라 결과를 출력하세요.

- 예)
- ' line [{(<plus>)] ' 14번째 괄호가 짝이 맞지 않기 때문에 인덱스 14의 음수인 -14를 출력
  - ' ABC({ABC)ABC ' 의 경우에는 짝이 맞지 않는 괄호와 닫히지 않은 괄호가 동시에 존재하며 이때 왼쪽 기준으로 우선인 -8 출력
  - ' line [{(<plus>)] ' 문자열은 괄호 1개가 닫히지 않고 끝나기 때문에 마지막 인덱스 15의 음수인 -15를 출력
  - ' (A)[B] ' 라는 문자열은 2개의 괄호 쌍이 존재하기 때문에 2를 출력
  - ' ABC)ABC ' 의 경우에는 -3을 출력

| 입력하는 문자열            | 출력 결과 |
|---------------------|-------|
| Hello, world!       | 0     |
| line [{(<plus>)]    | -14   |
| line [{(<plus>)]    | -15   |
| >_<                 | 0     |
| x * (y + z) ^ 2 = ? | 1     |

자료구조/알고리즘

### 추천 문제

스택/큐

<https://programmers.co.kr/learn/courses/30/lessons/42587> (프린터)

<https://programmers.co.kr/learn/courses/30/lessons/42583> (다리를 지나는 트럭)

정렬

<https://programmers.co.kr/learn/courses/30/lessons/42748> (K번째 수)

완전탐색

<https://programmers.co.kr/learn/courses/30/lessons/42840> (모의고사)

Hash

<https://programmers.co.kr/learn/courses/30/lessons/42578> (위장)

수고하셨습니다.