

# 자바2

알고리즘1  
(List/LinkedList/Stack/Queue)

# Chapter 03

## 알고리즘1

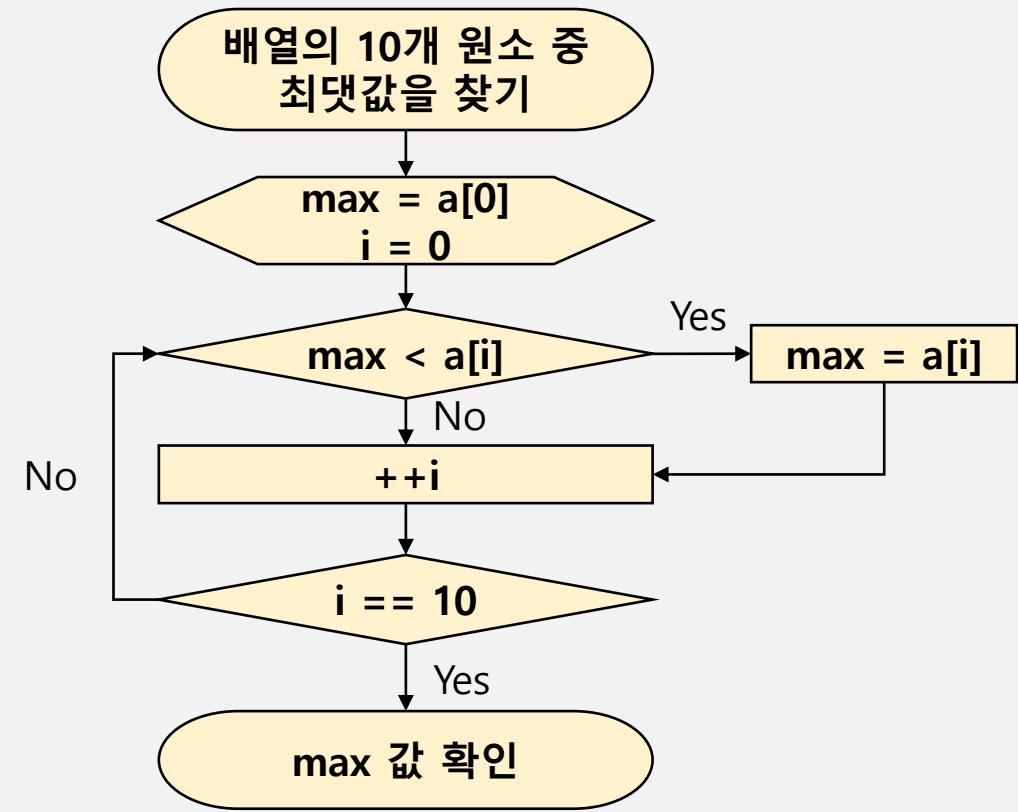
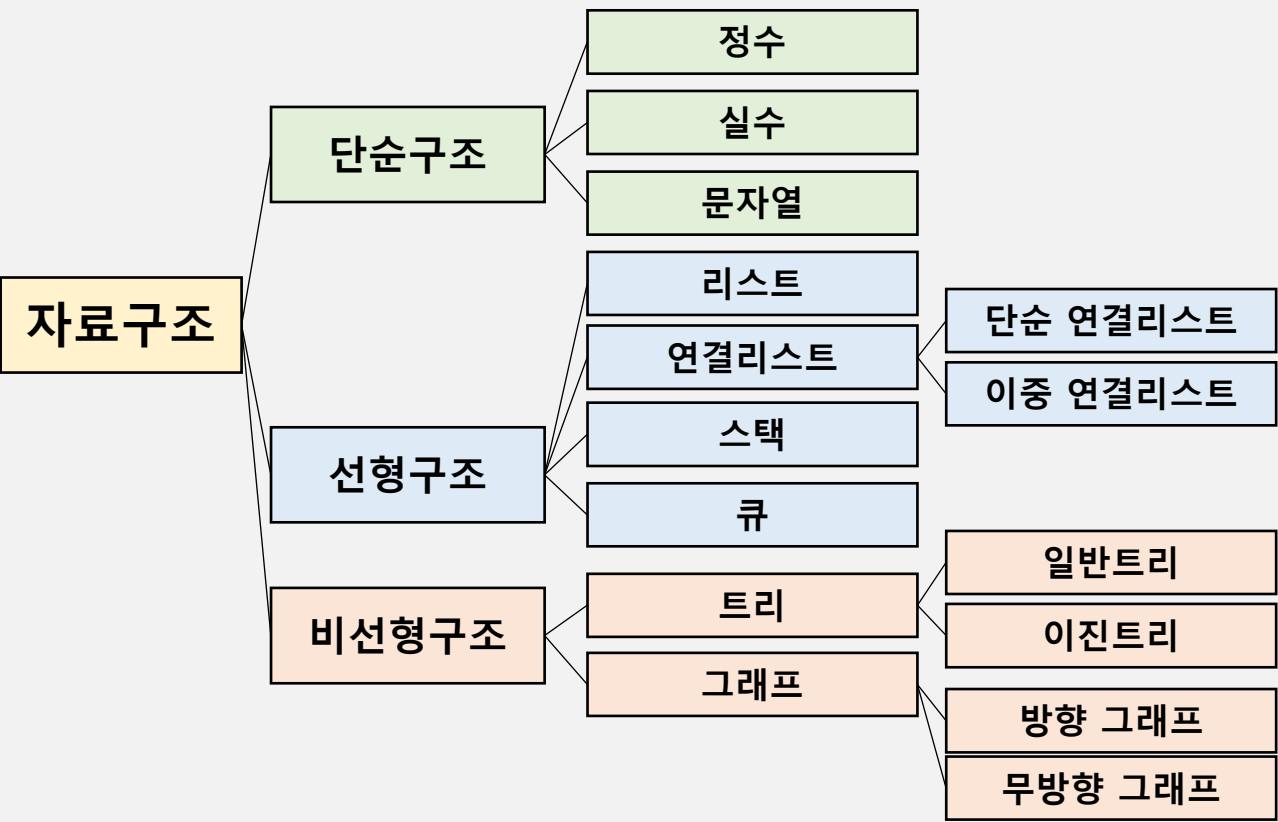
### Chapter03의 학습목표

- 알고리즘과 자료구조에 대해 학습한다
- 다양한 알고리즘 설계 기법과 복잡도에 대해 알아본다
- List, LinkedList의 차이와 Stack, Queue자료구조를 이해한다.

알고리즘

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)
축소 정복(decrease-and-conquer)	주어진 문제를 하나의 좀 더 작은 문제로 축소하여 해결하는 방법	-
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

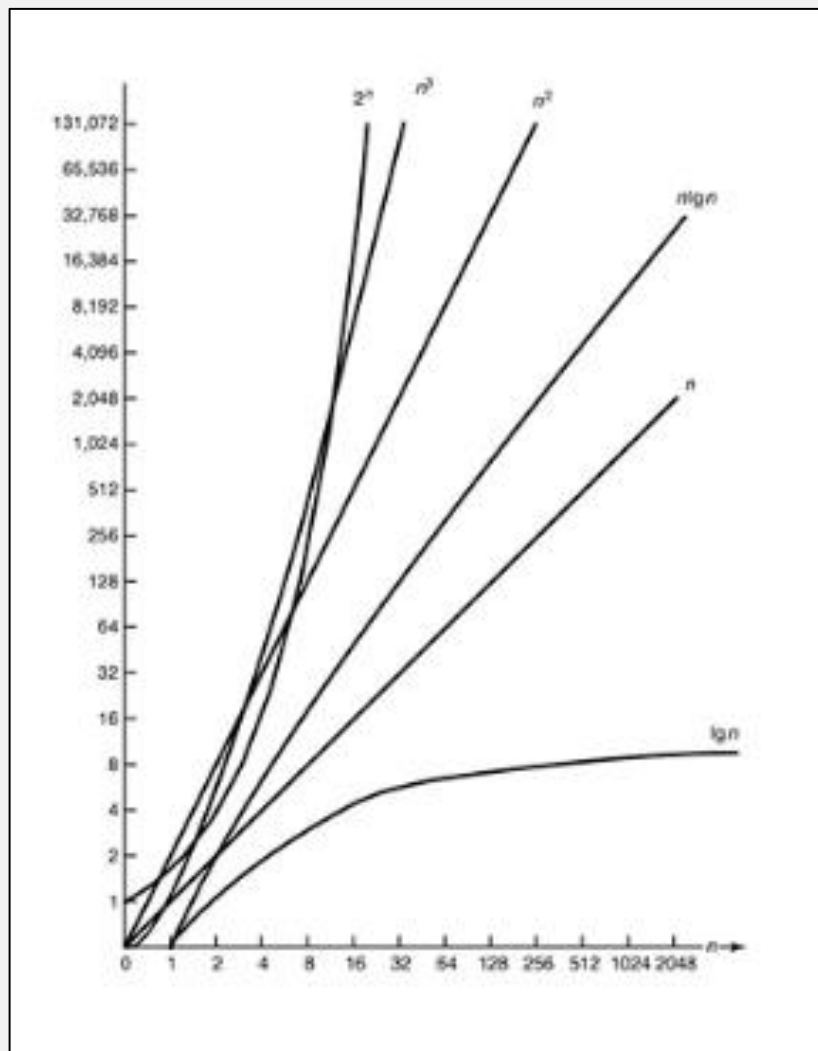
1년은 31,536,000초  $< 10^8$ , 우주의 연령 = 137억년  $\rightarrow 10^{11}$ 년



### 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	i=1,2,4	3
n=7	i=1,2,4	3
n=8	i=1,2,4,8	4
...	...	...
n=16	i=1,2,4,8,16	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(logn) - 로그형	0	2	3	4	5	이진 탐색
O(nlogn) - 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

리스트 자료구조

리스트와(List) 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) –  $O(n)$



리스트 자료구조

리스트와(List) 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

리스트 자료구조

리스트와(List) 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!

리스트 자료구조

## 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;  
Linkded<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

## 리스트 자료구조

### 자바의 연결 리스트(Linked list) – 2

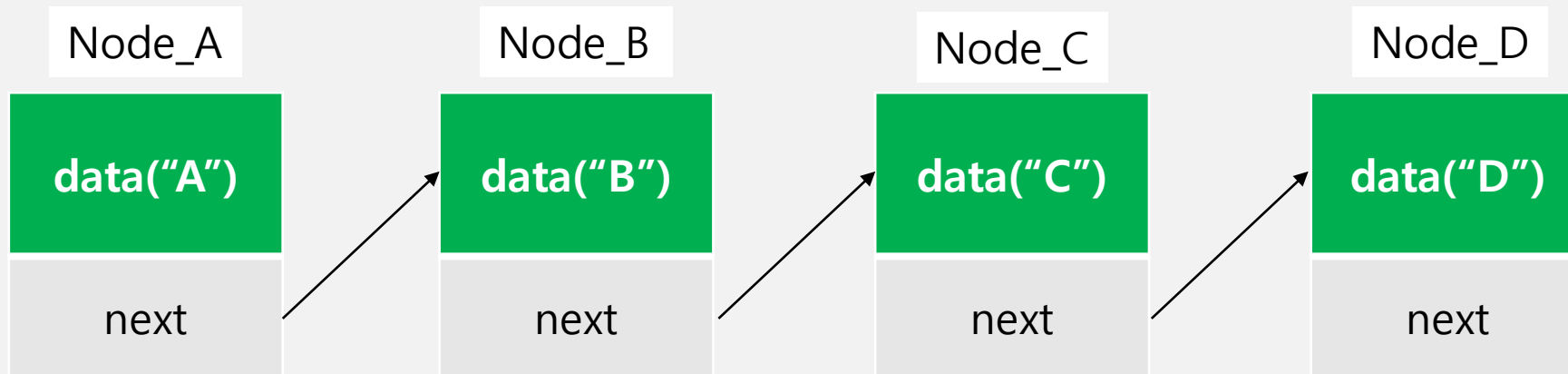
- 연결 리스트는 노드 Node를 사용해 class로 구현가능

#### Node의 구현

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

#### Node의 생성과 연결

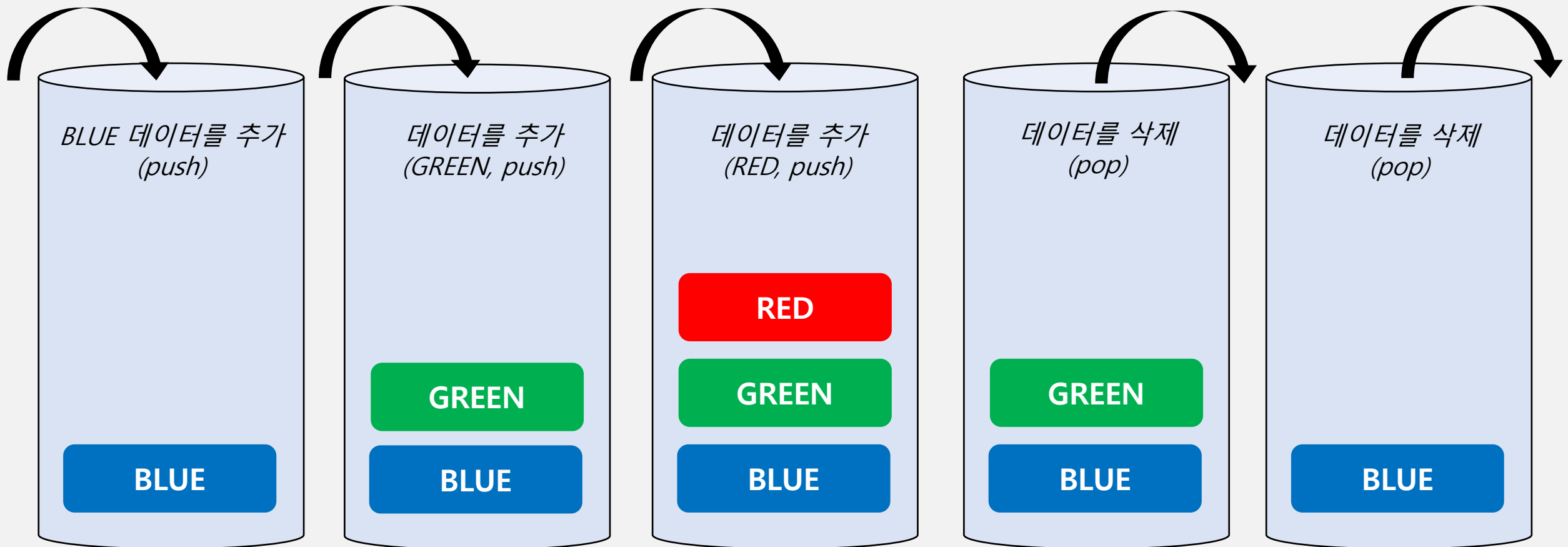
```
Node nodeA = new Node("A", null);  
Node nodeB = new Node("B", nodeA);  
Node nodeC = new Node("C", nodeB);  
Node nodeD = new Node("D", nodeC);
```



### 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

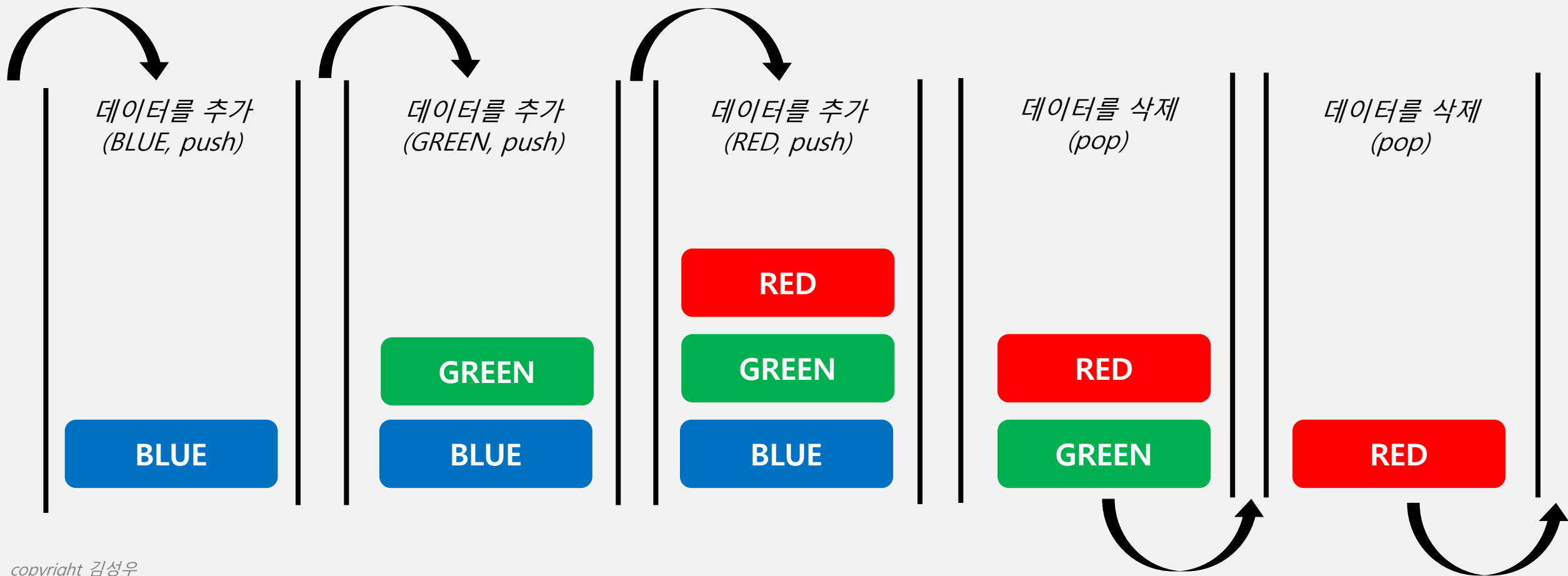




스택과 큐

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 이신전심이다", "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;
}
```

스택과 큐

<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 return 하고, 올바르지 않은 괄호이면 *false*를 출력하세요.

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

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

### 연습 문제

**아래 지시를 읽고 입력하는 임의의 문자열 s에 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 출력**합니다.

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     |

### 스택과 큐

#### 추천 문제

##### 스택/큐

1. <https://programmers.co.kr/learn/courses/30/lessons/42587> (프린터)
2. <https://programmers.co.kr/learn/courses/30/lessons/42583> (다리를 지나는 트럭)

수고하셨습니다.