Week 4. Recursion.

– Techniques for Managing Structured Data and Object Behavior 2

황현석 / 25. 04. 12

Review.. Very Important

Types of Classes in Java

Type	Description
Concrete Class	기본적인 클래스
Abstract Class	일부만 구현된 클래스. 일반적인 클래스에서 상속을 통해 실제로 구현해야함.
Interface	기능의 명세만 가진 형태. 다중 구현이 가능하다.
Final Class	완성된 최종 클래스, 더 이상 상속이 불가능함.
Inner Class	클래스 안에 선언된 클래스.
Static Nested Class	inner 클래스가 static으로 선언된 클래스
Anonymous Class	이름없이 선언과 동시에 구현하는 클래스. 1회성
Record Class	데이터 만을 담고 운송하기 위해, 만들어짐. 간결하게 선언하는 불변 클래스

Class about Constructor

```java
class Point {  6 usages
    int x, y, z;  3 usages

    Point () {}  1 usage

    Point (int x) {  no usages
        this.x=x;
    }

    Point (int x, int y) {  2 usages
        this.x=x;
        this.y=y;
    }

    Point (int x, int y, int z) {  no usages
        this.x=x;
        this.y=y;
        this.z=z;
    }
}
```

```java
class Point {  6 usages
    int x, y, z;  1 usage

    Point () {}  1 usage

    Point (int x) {  1 usage
        this.x=x;
    }

    Point (int x, int y) {  3 usages
        this(x);
        this.y=y;
    }

    Point (int x, int y, int z) {  no usages
        this(x, y);
        this.z=z;
    }
}
```

Interface

What is that..? why use that..?

```java
interface Weapon {  4 usages  2 implementations
    void attack();  1 usage  2 implementations
}

class Sword implements Weapon {  no usages
    public void attack() {  1 usage
        System.out.println("검으로 찌른다!");
    }
}

class Bow implements Weapon {  no usages
    public void attack() {  1 usage
        System.out.println("활을 쏜다!");
    }
}

class Warrior {  no usages
    private Weapon weapon;  2 usages

    // 🗡️ 외부에서 무기를 주입받음 (Injection)
    public Warrior(Weapon weapon) {  no usages
        this.weapon = weapon;
    }

    public void fight() {  no usages
        weapon.attack(); // 어떤 무기든 쓸 수 있음!
    }
}
```

```java
public static void main(String[] args) throws IOException {
    Warrior warrior = new Warrior(new Bow());
    Warrior warrior2 = new Warrior(new Sword());

    Warrior warrior3 = new Warrior(new Weapon() {
        @Override  1 usage
        public void attack() {
            System.out.println("테스트용 무기 주입.");
        }
    });
```

Interface

That’s why We should use Anonymous Class

```
45 public class Main {
46     public static void main(String[] args) {
47         MyIntegerList list = new MyIntegerList();
48
49         //랜덤으로 100개의 수 넣기.
50         for (int i=0;i<10;i++) list.add(new Random().nextInt(100));
51
52         //익명 클래스를 파라미터를 넘겨주는 공간에서 만들어 버림.
53         list.sort(new MyComparator() {
54             @Override
55             public int compare(int a, int b) {
56                 return Integer.compare(Math.abs(a), Math.abs(b));
57             }
58         });
59
60         System.out.println(list);
61     }
62 }
63
```

ArrayList & Sort

https://www.acmicpc.net/problem/11650

5 11650번 | 제출 | 맞힌 사람 | 숏코딩 | 재채점 결과 | 채점 현황 | 내 제출 | 강의 | 질문 게시판

좌표 정렬하기 실패

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	168652	82439	64242	48.798%

문제

2차원 평면 위의 점 N개가 주어진다. 좌표를 x좌표가 증가하는 순으로, x좌표가 같으면 y좌표가 증가하는 순서로 정렬한 다음 출력하는 프로그램을 작성하시오.

다음을, Class, BufferedReader, BufferedWrtier
ArrayList, ArrayList.sort(),
anonymous Class implements Comparator interface,
를 사용하여 풀자.

How to Solve...?

```
class Point {  6개 사용 위치
    int x, y;  6개 사용 위치
    Point(int x, int y) {  1개 사용 위치
        this.x = x;
        this.y = y;
    }
}
```

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));

int N = Integer.parseInt(br.readLine());

ArrayList<Point> points = new ArrayList<>();

while(N-- > 0) {
    StringTokenizer st = new StringTokenizer(br.readLine());

    int x = Integer.parseInt(st.nextToken());
    int y = Integer.parseInt(st.nextToken());

    points.add(new Point(x, y));
}
```

How to Solve...?

```java
points.sort(new Comparator<Point>() {
    @Override
    public int compare(Point o1, Point o2) {
        if (o1.x == o2.x) return Integer.compare(o1.y, o2.y);
        return Integer.compare(o1.x, o2.x);
    }
});


for (Point p : points) {
    bw.write( str: p.x + " " + p.y + "\n");
}
bw.flush();
```

ArrayList의 Sort메서드를 이용한다.

Comparator〈Point〉 인터페이스로 작성하면,
compare 인자를 Point 타입으로 받아 올 수 있다.

Table of Contents

What is Recursive..?

자기 자신을 호출 하는 함수를 재귀 함수라고 합니다.

재귀에는 더이상 재귀하지 않고 종료하는 조건인

기저 조건 (Base Case) 라는 용어가 있습니다.

```
public static void recursive (int parameter) {
    if (parameter == 0) return;
    recursive( parameter: parameter- 1);
}
```

What is tail... head..?

```java
public static int factorial (int n) {
    if (n == 0) return 1;
    return n * factorial( n: n-1);
}
```

최종 값이 첫번 째 함수 호출에서 만들어 진다. 머리 재귀

```java
public static int factorial (int value, int n) {  1개 사
    if (n == 0) return 1;
    return factorial( value: value * n,  n: n-1);
}
```

최종 값이, 재귀의 재귀의… 가장 깊은 재귀 함수에서 만들어 진다.
이를 꼬리 재귀라고 한다.

What is tail... head..?

```java
public static int factorial (int n) {
    if (n == 0) return 1;
    return n * factorial( n: n-1);
}
```

최종 값이 첫번 째 함수 호출에서 만들어 진다. 머리 재귀

```java
public static int factorial (int value, int n) {  1개 사
    if (n == 0) return 1;
    return factorial( value: value * n,  n: n-1);
}
```

최종 값이, 재귀의 재귀의… 가장 깊은 재귀 함수에서 만들어 진다.
이를 꼬리 재귀라고 한다.

Why use tail recursive..?

자바에서는 꼬리 재귀 함수 최적화를 공식적으로 지원해 주지 않는다.

일반적인 재귀 호출은 각 호출마다 새로운 스택 프레임이 쌓인다.
호출이 많아지면, 메모리를 많이 사용하고, StackOverFlow가 발생 할 수 있다.

꼬리 재귀의 특징은 다음과 같다.

함수의 “마지막 동작”이 자기 자신에 대한 호출일 때,
현재 함수 scope의 변수들은 더이상 필요가 없다.

```java
public static int factorial (int value, int n) {  1개 사
    if (n == 0) return 1;
    return factorial( value: value * n,  n: n-1);
}
```

Why use tail recursive..?

꼬리 재귀 최적화(Tail Call Optimization)란?

컴파일러 또는 런타임이 이전 스택 프레임을 제거하고 새 호출로 덮어씌우는 방식으로 동작합니다.

즉, 새로운 스택을 쌓지 않고, 기존 것을 “재사용”합니다.

새로운 스택 프레임을 줄여 메모리 효율을 높이고 무한 재귀도 가능하게 함

Why use tail recursive..?

TCO를 지원해주는 언어들..

이 언어들은 **컴파일러 수준 또는 인터프리터 수준에서 TCO를 자동 적용**합니다.

언어	지원 여부	비고
Scheme	✅ 완전 지원	TCO는 스펙에 명시됨 (필수 조건)
Racket	✅ 지원	Scheme 계열
OCaml	✅ 제한적 지원	명확한 Tail Call에만 최적화 수행
F#	✅ 지원	.NET 기반이지만 TCO 수행
Haskell	✅ 지원	지연 평가 + 꼬리 재귀 최적화 (GHC 기준)
Scala	⚠️ 제한적 지원	`@tailrec` 애너테이션 사용 시만 최적화 가능
Erlang	✅ 지원	함수형 언어, 재귀 중심 구조에 최적화되어 있음
Elixir	✅ 지원	Erlang 기반 (BEAM VM)이라 TCO 지원
Standard ML (SML)	✅ 지원	함수형 언어로서 TCO 수행
Clojure	⚠️ `recur` 키워드로만 지원	일반 재귀는 스택 쌓임
Lua	✅ 지원	5.0 이후 버전에서 TCO 적용
Prolog	✅ 지원	내부적으로 꼬리 호출 최적화 사용함

Why use that...?

```java
public static ArrayList<Integer> []edges = new ArrayList[10];  1개

public static void depthFirstSearch (int node) {  1개 사용 위치
    for (int nextNode : edges[node]) {
        depthFirstSearch(nextNode);
    }
}
```

사진 처럼, 재귀 함수 하나로, 그래프를 탐색하는 함수를 구현 가능하다.

BackTracking 같은, 실패 후, 실행했던 과정들을 Undo하고 모든 상태를 저장해야 하는 알고리즘 같은 경우, 재귀함수로 구현하지 않을 시 굉장히 까다롭다.

함수의 StackFrame 안에 변수 값들을 저장해두는 기법은 굉장히 편하다.

Why use that...?

분할 정복 문제 해결 기법 :

문제를 분할 하여, 부분 문제를 해결한 뒤, 해결한 결과물을 합쳐,
최종적인 문제를 해결하는 최적화 전략

이때, 사용 되는 재귀의 개념 :

부분 문제로 분할 하였는데, 범위만 작은 똑같은 문제를 푸는 거라면?
파라미터만 다른, 재귀함수를 돌리는 것으로 생각해도 무방하다.

Key Algorithms in Recursion

유클리드 호재법

유클리드 호제법

두 양의 정수 $a, b\ (a > b)$에 대하여 $a = bq + r\ (0 \leq r < b)$[1]이라 하면, a, b의 최대공약수는 b, r의 최대공약수와 같다. 즉,

$$\gcd(a,\ b) = \gcd(b,\ r)$$

$r = 0$이라면, a, b의 최대공약수는 b가 된다.

최소 공약수를 빠르게 구하는 알고리즘 이다.
수학적 정의에서 조차, gcd(a, b) 를 또다른 함수 gcd(b, r) 라고 정의했다.

$\gcd(a,\ b) = G, \gcd(b,\ r) = G'$이라 하자. 적당한 서로소인 정수 A, B에 대해 $a = GA, b = GB$가 성립한다. 이를 $a = bq + r$에 대입하면, $GA = GBq + r$이고, $r = G(A - Bq)$이다. 여기서 G는 b와 r의 공약수임을 알 수 있다. 즉, G는 G'의 약수이다.

$G' = mG$로 두자. 그러면 적당한 서로소인 두 정수 k, l에 대해 $GB = G'k = Gmk, G(A - Bq) = G'l = Gml$이 성립한다.
각 변을 G로 나누면 $B = mk, A - Bq = ml$이다.
$A = ml + Bq = ml + mkq = m(l + kq)$에서 m은 A와 B의 공약수인데, $\gcd(A, B) = 1$이므로 항상 $m = 1$이고 $G' = G$이다.

Key Algorithms in Recursion

유클리드 호재법

```java
public static int gcd (int a, int b) {  1개 사용 위치
    if (b == 0) return a;
    return gcd(b,  b: a % b);
}
```

a와 b의 최대 공약수를 구하는 것은, b와 a%b의 최대 공약수를 구하는 것 과 같다.

수의 범위를 줄여서, 같은 문제를 푸는 것은 재귀의 성격과 부합한다.

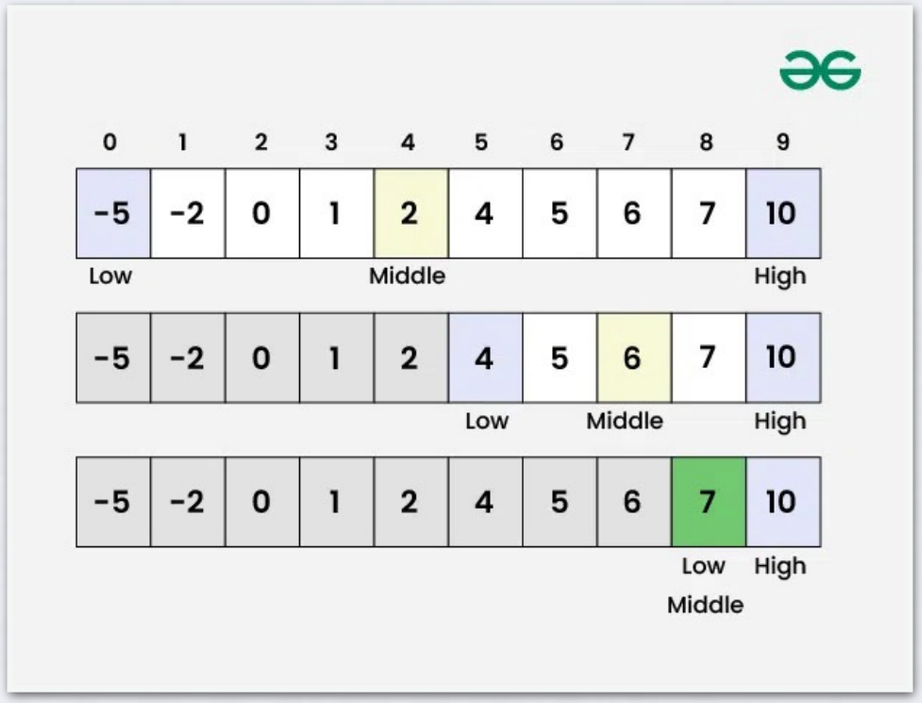
추가적인 로직처리를 위해, 파라미터 위치만 옮기는 재귀 함수다.

Key Algorithms in Recursion

이진 탐색

정렬된 배열에서 원하는 값 K를 찾을 때, 배열의 가운데 값을 먼저 비교하여 탐색 방향을 결정하고, 나머지 절반은 확인할 필요가 없습니다.

탐색 범위만 달라질 뿐 동일한 방식의 비교와 판단이 반복되므로, 이 과정을 재귀 함수로 바꿔 표현할 수 있습니다.



Key Algorithms in Recursion

이진 탐색

```java
public static int arr[];  2개 사용 위치

//구간 [L, R]에 value가 있는지 찾는 함수
public static boolean findFunction (int L, int R, int value) {  2개 사용 위치
    if (L == R) {
        return arr[L] == value;
    }

    int mid = (L + R) / 2;

    if (value <= arr[mid]) {
        return findFunction(L, mid, value);
    } else {
        return findFunction( L: mid+1, R, value);
    }
}
```

Key Algorithms in Recursion

MergeSort 병합정렬 알고리즘

정렬.. 그냥 하는 것은 힘들다.
정렬된 두 배열을 합치는 것은 두 배열의 원소의 합만큼 시간이 걸린다.
정렬된 두 배열의 가장 작은 원소끼리만 비교해서 차곡차곡 넣으면 된다.

그럼, 처음에, 재귀적으로 원소들을 분할 하다가, 한개가 남으면
그것은, 정렬된 하나의 배열이라고 봐도 무방하다.

Key Algorithms in Recursion

MergeSort 병합정렬 알고리즘

분할 재귀 함수를 통해, 각각의 재귀 함수에서 리턴한 정렬된 배열을 병합해주고 리턴하는 형식의 알고리즘이다.

```java
public static ArrayList<Integer> merge (ArrayList<Integer> o1, ArrayList<Integer> o2) {  1개
    //o1, o2가 정렬되어 있으면, 이 둘을 합치는 것은 배열크기 만큼 걸린다!
    ArrayList<Integer> result = new ArrayList<>();

    int iter1 = 0, iter2 = 0;

    while(true) {

        if (iter1 < o1.size() && iter2 < o2.size()) {
            if (o1.get(iter1) < o2.get(iter2)) {
                result.add(o1.get(iter1++));
            } else {
                result.add(o2.get(iter2++));
            }
        } else if (iter1 < o1.size()) {
            result.add(o1.get(iter1++));
        } else if (iter2 < o2.size()) {
            result.add(o2.get(iter2++));
        } else {
            return result;
        }
    }
}
```

Key Algorithms in Recursion

MergeSort 병합정렬 알고리즘

```java
//split 역할을 하지만 리턴할 때는 정렬된 배열을 리턴한다고 정의한다.
public static ArrayList<Integer> split (ArrayList<Integer> list) {  2개 사용 위치
    if (list.size() == 1) return list;

    int mid = list.size() / 2;

    ArrayList<Integer> o1 = new ArrayList<>(), o2 = new ArrayList<>();

    for (int i=0;i<mid;i++) {
        o1.add(list.get(i));
    }

    for (int i=mid;i<list.size();i++) {
        o2.add(list.get(i));
    }

    //한쪽은 mid개, 한쪽은 list.size() - mid개
    //즉, 반반씩 나눠서 재귀적으로 수행한다.

    o1 = split(o1);
    o2 = split(o2);

    //o1과 o2는 정렬된 배열이므로, merge를 통해 빠르게 합쳐 준다.
    return merge(o1, o2);
}
```

Practice..

https://www.acmicpc.net/problem/17478

Practice..

https://www.acmicpc.net/problem/17478

Practice..

```java
public static int N;  2개 사용 위치
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    N = sc.nextInt();

    System.out.println("어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.");
    rec( depth: 0);
}

public static void rec (int depth) {  2개 사용 위치
    String prefix = "_".repeat( count: depth * 4);

    System.out.println(prefix + "\"재귀함수가 뭔가요?\"");

    //baseCase
    if (depth == N) {
        System.out.println(prefix + "\"재귀함수는 자기 자신을 호출하는 함수라네\"");
        System.out.println(prefix + "라고 답변하였지.");
        return;
    }

    System.out.println(prefix + "\"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.");
    System.out.println(prefix + "마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.");
    System.out.println(prefix + "그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어.\"");

    rec( depth: depth + 1);

    System.out.println(prefix + "라고 답변하였지.");
}
```

We've completed Week 4