??

- 1. [Java] 암호 해독 - cipher.charAt(i - 1)
- 2. [Java] 대문자와 소문자
- 3. [Java] 영어가 싫어요 - numbers.replaceAll(a,b);
- 4. [Java] 인덱스 바꾸기 - return new String(arr);
- 5. [Java] 한 번만 등장한 문자 (문자열) - sb.toString();
- 16 . [Java] 문자열 여러번 뒤집기 - sb.charAt()
- 6. [Java] 약수구하기 - list.toArray(new String[0]);
- 7. [Java] 가장 큰 수 찾기 - return new int[]{max, idx};
- 8. [Java] 문자열 계산하기 - Character.isDigit(c)
- 9. [Java] 배열의 유사도 (배열, Set) - set.contains(str)
- 10. [Java] 숫자 찾기 - String.indexOf(), String.valueOf()
- 11. [Java] 삼각형의 완성조건 (2): case2는 왜 longer - shorter인가?
- 12. [Java] 옹알이 (1): break vs continue
- 13. [Java] 배열만들기 4: stack → int[] 변환
- 14. [Java] 주사위 게임 3
- 15. [Java] 9로 나눈 나머지

1. [Java] ?? ?? - cipher.charAt(i - 1)

> https://school.programmers.co.kr/learn/courses/30/lessons/120892

> 오답

```
class Solution {
    public String solution(String cipher, int code) {
        StringBuilder sb = new StringBuilder();
        for (int i = code; i <= cipher.length(); code++;) {
            sb.append(cipher[i-1]); // 인덱스는 0부터 시작
        }
        return sb.toString();
    }
}
```

? for?? ??? ??

```
for (int i = code; i <= cipher.length(); code++;)
```

- → code++→ 증가식 자리에 있어야 할 변수는 i (반복 제어 변수)
- → code++는 code 값을 계속 바꿔버려서 논리적으로도 틀림

? ??? ??? ?? ??

- Java에서 문자열은 []로 접근 불가 → charAt(index) 사용해야 함

> 정답 코드

```
class Solution {
    public String solution(String cipher, int code) {
        StringBuilder sb = new StringBuilder();
        for (int i = code; i <= cipher.length(); i += code) {
            sb.append(cipher.charAt(i - 1)); // □□□□ 0□□ □□
        }
        return sb.toString();
    }
}
```

2. [Java] ???? ???

https://school.programmers.co.kr/learn/courses/30/lessons/120893

오답

```
class Solution {
    public String solution(String my_string) {
        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < my_string.length(); i++) {
            if(Character.isUpperCase((my_string.charAt(i)))) {
                sb.append(Character.toLowerCase(my_string.charAt(i)));
            } else {
                sb.append(Character.toUpperCase(my_string.charAt(i)));
            }
        }
        return sb.toString();
    }
}
```

 my_string.charAt(i)는 위로 빼서 char c = my_string.charAt(i); 로 만들어 주면 코드가 간단해진다 .

개선한 코드

```
class Solution {
    public String solution(String my_string) {
        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < my_string.length(); i++) {
            char c = my_string.charAt(i);
            if(Character.isUpperCase(c)) {
                sb.append(Character.toLowerCase(c));
            } else {
                sb.append(Character.toUpperCase(c));
            }
```

```
        }
        return sb.toString();
    }
}
```

3. [Java] ??? ??? - numbers.replaceAll(a,b);

> https://school.programmers.co.kr/learn/courses/30/lessons/120894

> 오답

```
class Solution {
    public long solution(String numbers) {
        String[] words = {
            "zero", "one", "two", "three", "four", "five",
            "six", "seven", "eight", "nine"
        };
        for (int i = 0; i < words.length; i++) {
            numbers = numbers.replace(words[i], String.valueOf(i));
        }
        return Integer.parseInt(numbers);
    }
}
```

테스트는 통과했으나 답안 제출시 런타임 에러

image.png

image.png

? Integer.parseInt() ?? ? ??? ??

- 만약 숫자 문자열이 int의 범위를 초과하면 런타임 에러가 발생한다 .
- 그래서 일부 테스트 케이스에서 에러가 난 것
- 예를 들어서 , 만약 13자리 숫자가 들어오면 int(32비트 정수)의 최대값 2,147,483,647을 초과함
- → Integer.parseInt()는 처리 못 함 → 런타임 에러 (NumberFormatException) 발생
- 따라서 return 타입을 int가 아니라 long 타입으로 변경해야 한다 .

> 개선한 코드

```
class Solution {
    public long solution(String numbers) {
        String[] words = {
            "zero", "one", "two", "three", "four", "five",
            "six", "seven", "eight", "nine"
        };
        for (int i = 0; i < words.length; i++) {
            numbers = numbers.replaceAll(words[i], String.valueOf(i));
        }
        return Long.parseLong(numbers); // □ int → long 으로 변경
    }
}
```

??? VS ?????

? ??? (Primitive type)

- int → 기본형 정수 타입 (primitive int)
- long → 기본형 정수 타입 (primitive long)

> □ 메모리에 값 자체를 저장함
> □ 성능이 빠르고 메모리 사용이 효율적

? ??? (Reference type, Wrapper class)

- Integer → 참조형 (래퍼 클래스 , Wrapper class for int)
- Long → 참조형 (래퍼 클래스 , Wrapper class for long)

> □ 객체이기 때문에 null 값을 가질 수 있음
> □ ArrayList<Integer>처럼 제네릭에 사용 가능
> □ 기본형은 제네릭에서 사용할 수 없음 → 참조형 필요

4. [Java] ??? ??? - return new String(arr);

> https://school.programmers.co.kr/learn/courses/30/lessons/120895

> 오답

```
class Solution {
    public String solution(String my_string, int num1, int num2) {
        String[] arr = String.toCharArray(my_string);
        char tmp = arr[num1];
        arr[num1] = arr[num2];
        arr[num2] = tmp;
        return arr.toString();
    }
}
```

? String[] arr = String.toCharArray(my_string);

- → String.toCharArray()는 char[]를 리턴하므로 타입은 String[]이 아니라 char[]이어야 한다 .

? String.toCharArray() ??

- → my_string.toCharArray();
- toCharArray()는 String 클래스의 인스턴스 메서드이다 .
- 그래서 String.toCharArray() □ (클래스 메서드 아님)
- 반드시 인스턴스 (my_string)에 붙여야 한다 .

? return arr.toString();

- char[]에서 toString()을 쓰면 주소값이 출력된다 .
- → 대신 new String(arr)을 사용해야 문자열로 출력된다 .

> 개선한 코드

```
class Solution {
    public String solution(String my_string, int num1, int num2) {
        char[] arr = my_string.toCharArray();
        char tmp = arr[num1];
        arr[num1] = arr[num2];
        arr[num2] = tmp;
        return new String(arr);
    }
}
```

??? ??? VS ???? ???

자바에서 메서드는 크게 인스턴스 메서드와 **클래스 메서드 (static 메서드)**로 나뉘는데 , 이 둘은 메서드를 어떻게 호출하고 언제 사용하는지에 따라 중요한 차이를 가진다 .

클래스 메서드는 static 키워드로 정의되어 있어 객체를 만들지 않아도 클래스 이름으로 바로 호출할 수 있는 메서드이다 . 클래스 메서드는 객체의 고유 상태와는 무관하며 , 보통 모든 인스턴스에서 공통으로 사용하는 로직을 구현할 때 사용됩니다 . 대표적인 예로 Integer.parseInt("123")는 문자열을 정수로 변환하는 클래스 메서드로 , Integer라는 클래스 이름만으로 바로 호출할 수 있다 .

반면 인스턴스 메서드는 특정 객체가 생성되어야만 사용할 수 있는 메서드이다 . 즉 , new 키워드로 객체를 만든 후에야 호출이 가능하며 , 주로 그 객체의 상태 (멤버 변수)를 기반으로 동작한다 . 예를 들어 , str.length()는 문자열 객체 str의 길이를 반환하는 인스턴스 메서드인데 , 이는 str이라는 객체가 존재해야만 호출할 수 있다 .

인스턴스 메서드는 객체마다 상태가 다를 수 있기 때문에 , 같은 메서드를 호출하더라도 객체에 따라 결과가 달라진다 . 예를 들어 , Person이라는 클래스에 sayHello()라는 인스턴스 메서드가 있다면 , person1.sayHello()와 person2.sayHello()는 각각 다른 이름을 출력한다 . 이는 각 인스턴스가 가지고 있는 name 값이 다르기 때문이다 .

이처럼 인스턴스 메서드는 객체의 고유한 데이터를 다루고 조작할 때 유용하며 , 클래스 메서드는 그와 무관한 공통 기능을 제공할 때 사용된다 . 개발자는 기능의 목적에 따라 두 메서드 중 적절한 쪽을 선택해야 한다 . 객체마다 달라지는 동작이 필요하다면 인스턴스 메서드 , 객체 없이 호출해도 되는 범용적인 기능이라면 클래스 메서드를 사용하는 것이 좋다 .

? 1. ???? ??? (Instance Method)

정의

- 객체를 생성해야 호출할 수 있는 메서드
- 객체의 상태 (필드 값)에 따라 동작이 달라질 수 있음

특징

- new 키워드로 인스턴스를 만든 후 사용
- 보통 객체 내부의 데이터를 다룰 때 사용됨

예시

```
public class Person {
    String name;

    public void sayHello() {
        System.out.println("Hello, my name is " + name);
    }
}

// 사용
Person p = new Person();
p.name = "Dain";
p.sayHello();  // 출력: Hello, my name is Dain
```

? 2. ??? ??? (Class Method)

정의

- 객체 생성 없이 호출 가능한 메서드
- static 키워드로 정의됨

특징

- 클래스 이름으로 직접 호출 가능
- 객체의 상태와 무관하게 동작함
- 보통 공통 기능 , 유틸리티 기능 등에 사용됨

예시

```
public class MathUtil {
    public static int add(int a, int b) {
        return a + b;
    }
```

```
}

// □□
int result = MathUtil.add(3, 5);  // □□ □□ □□ □□
```

5. [Java] ? ?? ??? ?? (???) - sb.toString();

> https://school.programmers.co.kr/learn/courses/30/lessons/120896

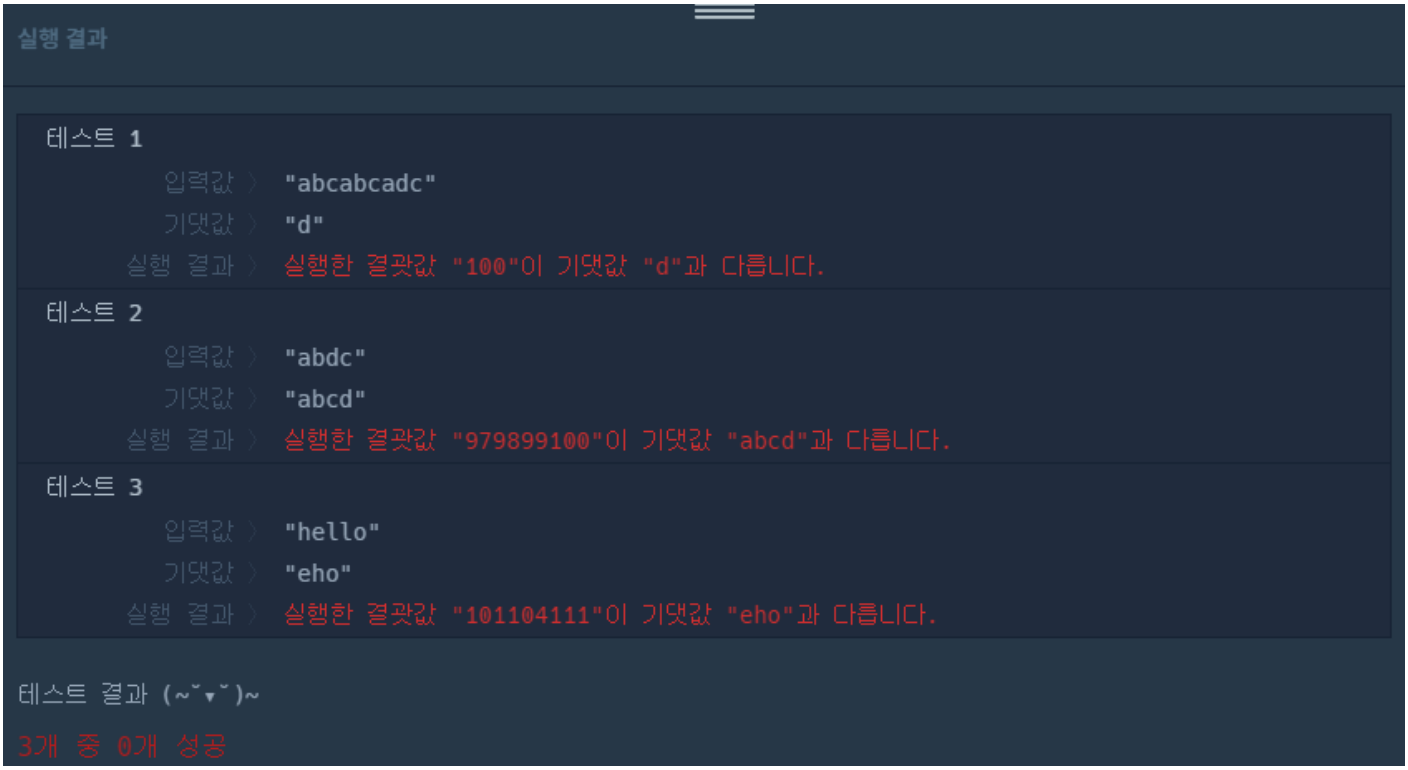
> □□

```
import java.util.*;

class Solution {
    public String solution(String s) {
        int[] cnt = new int[26];

        for(char c : s.toCharArray()) {
            cnt[c - 'a']++;
        }

        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 26; i++) {
            if (cnt[i] == 1) sb.append((char)i+'a');
        }
        return sb.toString();
    }
}
```



```
실행 결과

테스트 1
        입력값 〉 "abcabcadc"
        기댓값 〉 "d"
     실행 결과 〉 실행한 결괏값 "100"이 기댓값 "d"과 다릅니다.
테스트 2
        입력값 〉 "abdc"
        기댓값 〉 "abcd"
     실행 결과 〉 실행한 결괏값 "979899100"이 기댓값 "abcd"과 다릅니다.
테스트 3
        입력값 〉 "hello"
        기댓값 〉 "eho"
     실행 결과 〉 실행한 결괏값 "101104111"이 기댓값 "eho"과 다릅니다.

테스트 결과 (~˘▾˘)~
3개 중 0개 성공
```

? sb.append((char)i + 'a');

- 이 int를 char 타입으로 형변환하려는 의도지만, 실제로는 그냥 int로 연산된다.
- 예를 들어, i = 0일 때 char는 연산 시 int로 자동 변환되므로 → 0 + 97

```
(char)i + 'a' // => 정수 + 정수 → int로 연산됨
```

→ i + 'a' 전체를 char로 캐스팅해야 문자로 제대로 변환된다.

```
sb.append((char)(i + 'a'));
```

개선한 코드

```
import java.util.*;

class Solution {
    public String solution(String s) {
        int[] cnt = new int[26];

        for(char c : s.toCharArray()) {
            cnt[c - 'a']++;
```

```
        }

        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 26; i++) {
            if (cnt[i] == 1) sb.append((char)(i + 'a'));
        }
        return sb.toString();
    }
}
```

16 . [Java] ??? ??? ??? - sb.charAt()

> https://school.programmers.co.kr/learn/courses/30/lessons/181913

? ??

- 문자열 `my_string`과 2차원 정수 배열 `queries`가 주어짐
- `queries`의 각 원소 `[s, e]`는 `my_string`의 인덱스 **s**부터 **e**까지 부분 문자열을 뒤집으라 는 의미
- 모든 쿼리를 주어진 순서대로 적용한 뒤의 문자열을 반환

? ????

- `my_string`: 소문자만 , 길이 1~1,000
- `queries`: 길이 1~1,000
  - 각 원소는 `0 ≤ s ≤ e < my_string.length()`

> 코드

1. 문자열을 `StringBuilder`로 변환해서 수정 가능하도록 함
2. 각 `[s, e]`에 대해 투포인터로 문자열을 뒤집음
3. 모든 쿼리를 순서대로 적용 후 , 최종 문자열을 반환

```
import java.util.*;

class Solution {
    public String solution(String s) {
        int[] cnt = new int[26];

        for(char c : s.toCharArray()) {
            cnt[c - 'a']++;
        }

        StringBuilder sb = new StringBuilder();
```

```
        for (int i = 0; i < 26; i++) {
            if (cnt[i] == 1) sb.append((char)i+'a');
        }
        return sb.toString();
    }
}
```

위 코드에서 메서드를 따로 분리하면 아래와 같이 재사용성이 높은 코드가 된다 .

```
class Solution {
    public String solution(String my_string, int[][] queries) {
        StringBuilder sb = new StringBuilder(my_string);

        // 각 쿼리에 대해 부분 문자열을 뒤집기
        for (int[] query : queries) {
            int s = query[0];
            int e = query[1];
            reverse(sb, s, e);  // 구간 [s, e]를 뒤집는 메서드 호출
        }

        return sb.toString();
    }

    //특정 구간을 뒤집는 메서드 (swap 반복 사용)
    private void reverse(StringBuilder sb, int start, int end) {
        while (start < end) {
            swap(sb, start, end);  // 두 문자 위치 교환
            start++;
            end--;
        }
    }

    // 두 인덱스의 문자를 교환하는 메서드
    private void swap(StringBuilder sb, int i, int j) {
        char temp = sb.charAt(i);
        sb.setCharAt(i, sb.charAt(j));
        sb.setCharAt(j, temp);
    }
}
```

오답

처음에는 문자 두 개를 바꾸는 swap() 메서드를 별도로 구현했으나, 이것만으로는 로직 구현이 어렵다.

```
private void swap(StringBuilder sb, int i, int j) {
    char temp = sb.charAt(i);
    sb.setCharAt(i, sb.charAt(j));
    sb.setCharAt(j, temp);
}
```

특정 구간을 뒤집는 reverse() 메서드를 구현해서 시작 범위부터 끝 범위까지 반복되는 작업을 수행할 수 있도록 해야 한다.

```
private void reverse(StringBuilder sb, int start, int end) {
    while (start < end) {
        swap(sb, start, end);
        start++;
        end--;
    }
}
```

- `swap()` → 단일 문자 교환
- `reverse()` → **swap**을 여러 번 호출해 부분 문자열 뒤집기
- `solution()` → **queries**의 각 구간에 대해 **reverse** 호출

? for (int[] query : queries)

- for-each문 사용시 앞에 자료형을 int[]로 명시하면 정수형 배열을 순회할 수 있음

? sb.setCharAt(i, sb.charAt(j));

- 이 메서드는 Java에서 StringBuilder 객체 sb의 i번째 문자를 j번째 문자로 바꾸는 코드이다.

```
sb.setCharAt(i, sb.charAt(j));
```

메서드 이름	설명
`sb.charAt(j)`	`j`번째 문자를 가져옴 (읽기)
`sb.setCharAt(i, ...)`	`i`번째 위치에 문자를 덮어씀 (쓰기)

위 메서드의 사용법은 아래와 같다.

```
StringBuilder sb = new StringBuilder("hello");
sb.setCharAt(1, sb.charAt(4));  // 'e' → 'o'
System.out.println(sb); // 출력: "hollo"
```

`setCharAt(i, sb.charAt(j))`는 단순히 `i`번째 값을 `j`번째로 복사하는 것이기 때문에, 이 문제에서 요구한 문자 교환**(swap)**을 하려면 임시 저장 변수(temp)를 사용해야 한다.

---

6. [Java] ????? - list.toArray(new String[0]);

> https://school.programmers.co.kr/learn/courses/30/lessons/120897

> □□

```
import java.util.*;

class Solution {
    public int[] solution(int n) {
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = 0; i <= n; i++) {
            if(n%i==0) list.append(i);
        }
        int[] answer = list.toArray(new String[0]);
        return answer;
    }
}
```

실행 결과

```
/Solution.java:7: error: cannot find symbol
            if(n%i==0) list.append(i);
                           ^
  symbol:   method append(int)
  location: variable list of type ArrayList<Integer>
/Solution.java:9: error: incompatible types: inference variable T has incompatible boun
s
        int[] answer = list.toArray(new String[0]);
                                   ^
    lower bounds: int,Object
    lower bounds: String
  where T is a type-variable:
    T extends Object declared in method <T>toArray(T[])
2 errors

테스트 결과 (~˘▾˘)~
2개 중 0개 성공
```

? for (int i = 0; i <= n; i++)

- □ i = 0일 때 n % 0은 ArithmeticException (0으로 나눌 수 없음 !)
- □ □□□ i = 1부터 시작해야 한다 .

? list.add(i)

- ArrayList에서 값을 추가할 때는 append()가 아니라 add()를 사용해야 한다 .
- list.append(i) → list.add(i)

? int[] answer = list.toArray(new String[0]);

- List → Array 로 변환할때 참조형은 toArray() 메서드로 가능하지만 int 같은 기본형은 직접 변환할 수 없다 .
- 직접 반복문을 돌려서 변환해야 한다 .

> 개선한 코드

```
import java.util.*;

class Solution {
    public int[] solution(int n) {
```

```
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = 1; i <= n; i++) {
            if(n%i==0) list.add(i);
        }
        int[] answer = new int[list.size()];
        for(int i = 0; i < list.size(); i++) {
            answer[i] = list.get(i);
        }
        return answer;
    }
}
```

Stream ?? ??

```
import java.util.*;
import java.util.stream.*;

class Solution {
    public int[] solution(int n) {
        return IntStream.rangeClosed(1, n) // 1부터  n까지 숫자 생성
                .filter(i -> n % i == 0)   // 약수만 필터링
                .toArray();                // int[]로 반환
    }
}
```

List ? Array ???

List 타입	변환 방법
`List<String>`	`list.toArray(new String[0])`
`List<Integer>`	`list.toArray(new Integer[0])`
`List<Integer>` → `int[]`	반복문으로 수동 변환 필요

? 1. List ? Array (???, ?: Integer, String ?)

- toArray() 메서드로 변환이 가능하다 .

```
List<String> list = Arrays.asList("a", "b", "c");
String[] arr = list.toArray(new String[0]);
```

- list.toArray(new String[0]): 타입을 명시한 배열로 변환
- new String[0]은 사이즈가 0이어도 자동으로 맞춰준다 .

```
List<Integer> list = Arrays.asList(1, 2, 3);
Integer[] arr = list.toArray(new Integer[0]);
```

? 2. List ? Array (???, ?: int)

- Java의 List<Integer>는 int[]로 직접 변환 불가능
- 직접 반복문으로 넣어줘야 한다 .

```
List<Integer> list = Arrays.asList(1, 2, 3);
int[] arr = new int[list.size()];

for (int i = 0; i < list.size(); i++) {
    arr[i] = list.get(i);  // 언박싱 자동 수행
}
```

Array ? List ???

	변환 방법
`String[]`	`Arrays.asList(arr)`
`Integer[]`	`Arrays.asList(arr)`
`int[]`	Arrays.stream(arr).boxed().collect(Collectors.toList())

? 1. ??? ?? ? List (?: String[], Integer[] ?)

```
String[] arr = {"a", "b", "c"};
List<String> list = Arrays.asList(arr);
```

- Arrays.asList()는 배열을 고정 크기 리스트로 바꿔준다 .
- 추가 /삭제는 불가능 → 변경 가능한 리스트로 바꾸려면

```
List<String> modifiableList = new ArrayList<>(Arrays.asList(arr));
```

? 2. ??? ?? ? List (?: int[], double[] ?)

- 자바의 int[]는 바로 List<Integer>로 바꿀 수 없다 .
- Stream + boxed()를 활용한다 .

```
int[] arr = {1, 2, 3, 4};

List<Integer> list = Arrays.stream(arr)  // IntStream 생성
                          .boxed()      // int → Integer (박싱)
                          .collect(Collectors.toList());
```

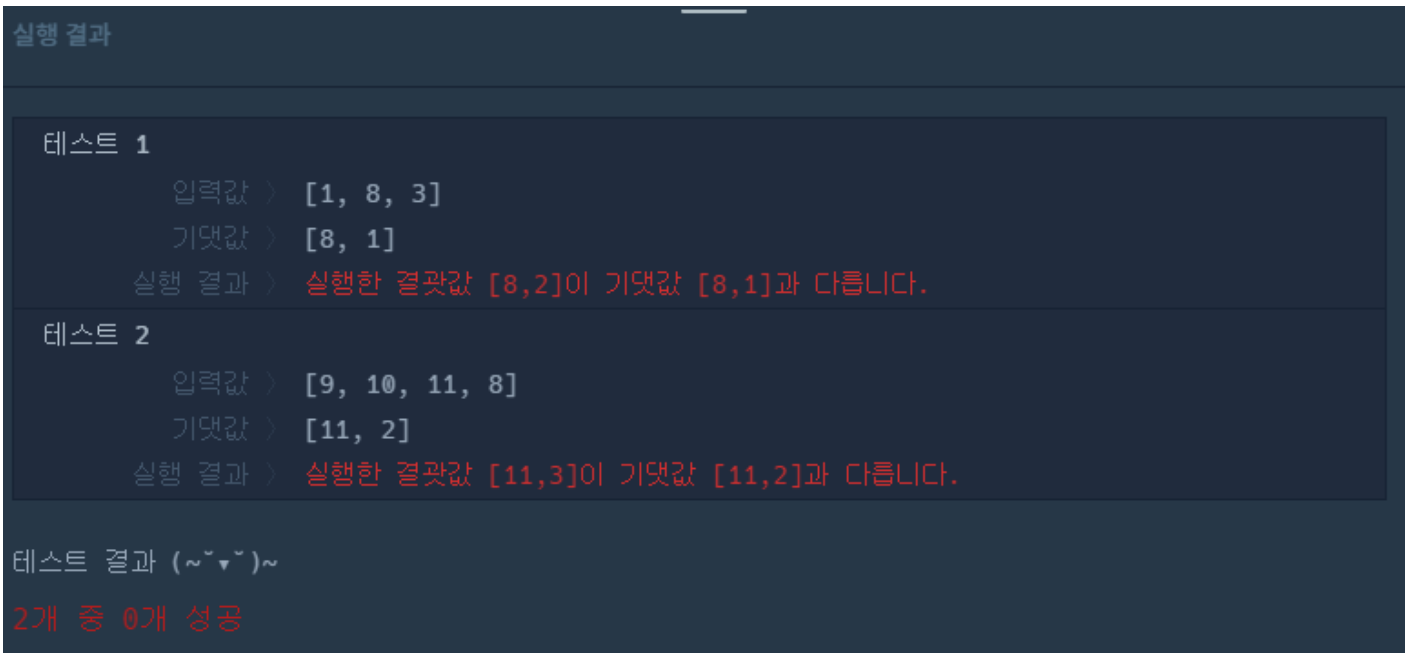
7. [Java] ?? ? ? ??- return new int[]{max, idx};

https://school.programmers.co.kr/learn/courses/30/lessons/120899

□□

```
import java.util.*;

class Solution {
    public int[] solution(int[] array) {
        Arrays.sort(array);
        int[] answer = new int[2];
        answer[0] = array[array.length-1];
        answer[1] = array.length-1;
        return answer;
    }
}
```



? index ?? ???

- array[array.length-1]는 정렬된 배열에서 최댓값이 맞지만, array.length-1은 최댓값의 원래 인덱스가 아님.
- 배열을 오름차순 정렬해버리면, 원래 배열에서 최댓값의 인덱스는 사라진다.
- 따라서 최댓값을 담을 변수 max, 인덱스를 담을 변수 idx를 선언하고 반복문을 돌려 최댓값과 인덱스를 갱신한 후, 두 변수를 배열로 변환해서 반환한다.

개선한 코드

```
import java.util.*;

class Solution {
    public int[] solution(int[] array) {
        int max = array[0];
        int idx = 0;

        for(int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
                idx = i;
            }
        }
        return new int[]{max, idx};
    }
}
```

배열 리터럴을 반환할 때 쓰는 코드

int형 두 개짜리 배열을 직접 생성하고, 그 배열을 메서드의 반환값으로 사용하는 코드이다.

```
return new int[]{max, idx};
```

아래와 같이 배열을 만들고 반환하는 코드인데 한 줄로 줄인 것이다.

```
int[] result = new int[]{max, idx};
return result;
```

Java에서 return은 값을 반환하기만 하면 되기 때문에, 굳이 int[] result = ...처럼 중간에 변수에 담을 필요가 없다.

배열 리터럴은 선언 없이 쓰면 컴파일 오류 난다.

```
int[] a;
a = {1, 2}; // □ □□□ □□
```

→ 대신 이렇게 써야 한다.

```
int[] a = new int[]{1, 2};
```

8. [Java] ??? ???? - Character.isDigit(c)

> https://school.programmers.co.kr/learn/courses/30/lessons/120902

> 틀린 코드 (1차 시도)

```
class Solution {
    public int solution(String my_string) {
        char[] arr = my_string.toCharArray();
        int sum = 0;
        for (char c : arr ) {
            if (c != '+' && c != '-' && c !=' ') {
                sum+= Integer.parseInt(c);
            }
        }
        return sum;
    }
}
```

실행 결과

```
/Solution.java:7: error: incompatible types: char cannot be converted to String
                sum+= Integer.parseInt(c);
                                       ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error

테스트 결과 (~˘▾˘)~
1개 중 0개 성공
```

? Integer.parseInt()

- String을 int형으로 변환하는 코드를 써서 에러가 났다 .
- char타입을 정수로 변환할 때는 Integer.parseInt() 대신 char 타입 변수에 - '0'을 한다 . (ASCII 코드)

> “ String → int VS char → int

1. String을 int로 변환할 때는 Integer.parseInt()

```
String str = "123";

// 문자열 → int
int num = Integer.parseInt(str);       // 123
```

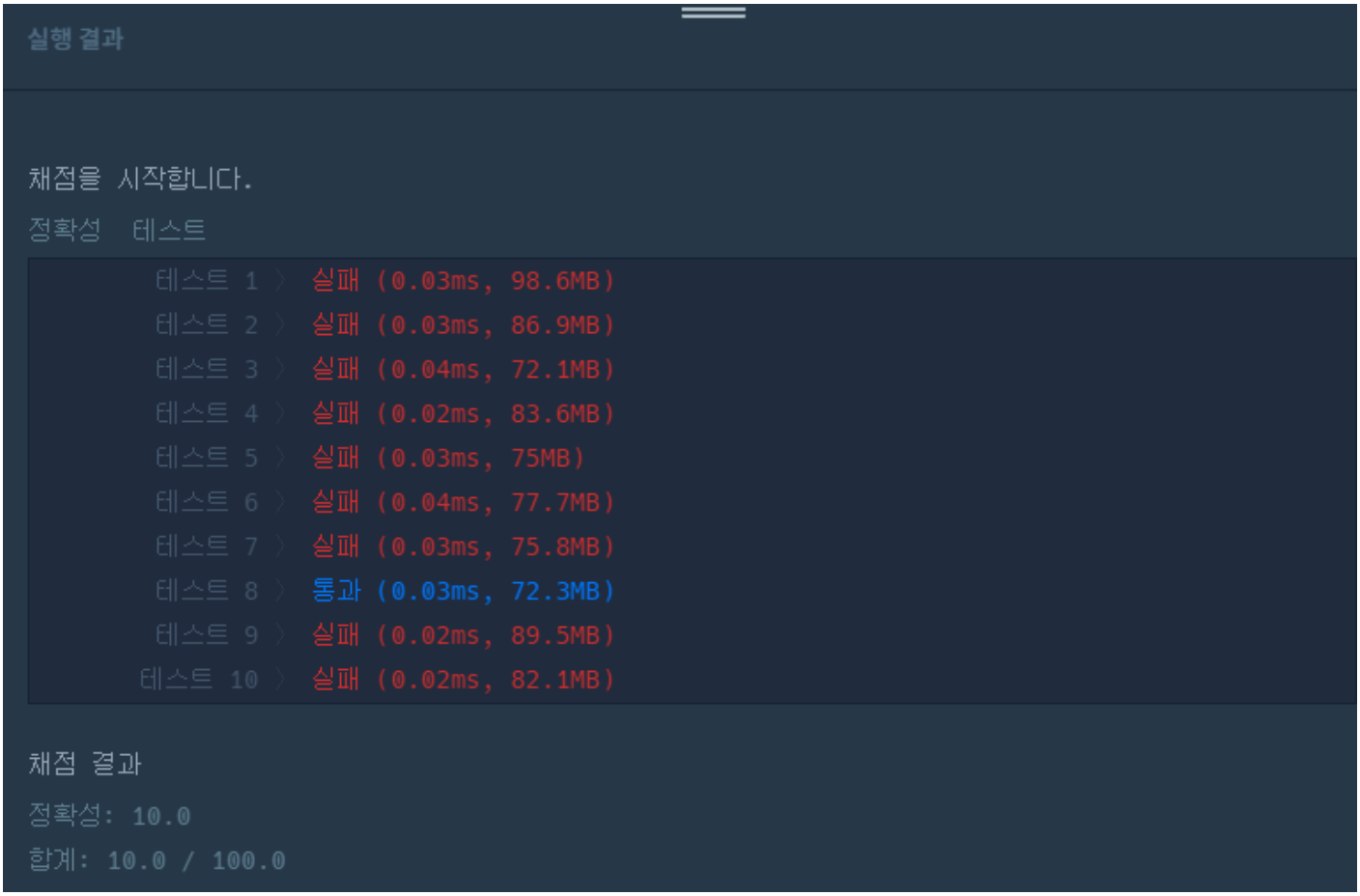
2. char을 숫자로 변환할 때는 char 타입 변수 - '0'

```
char c = '7';

// 문자 → 숫자 (정수값)
int digit = c - '0';   // '7' - '0' = 7
```

틀린 코드 (2차 시도)

```
class Solution {
    public int solution(String my_string) {
        char[] arr = my_string.toCharArray();
        int sum = 0;
        for (char c : arr ) {
            if (c != '+' && c != '-' && c !=' ') {
                sum+= c - '0';
            }
        }
        return sum;
    }
}
```



```
채점을 시작합니다.
정확성  테스트
      테스트 1 〉 실패 (0.03ms, 98.6MB)
      테스트 2 〉 실패 (0.03ms, 86.9MB)
      테스트 3 〉 실패 (0.04ms, 72.1MB)
      테스트 4 〉 실패 (0.02ms, 83.6MB)
      테스트 5 〉 실패 (0.03ms, 75MB)
      테스트 6 〉 실패 (0.04ms, 77.7MB)
      테스트 7 〉 실패 (0.03ms, 75.8MB)
      테스트 8 〉 통과 (0.03ms, 72.3MB)
      테스트 9 〉 실패 (0.02ms, 89.5MB)
     테스트 10 〉 실패 (0.02ms, 82.1MB)

채점 결과
정확성: 10.0
합계: 10.0 / 100.0
```

내가 쓴 코드는 모든 숫자를 "한 자리 숫자"로 처리한다. 즉, "12"가 있으면 1 + 2 = 3으로 계산한다.

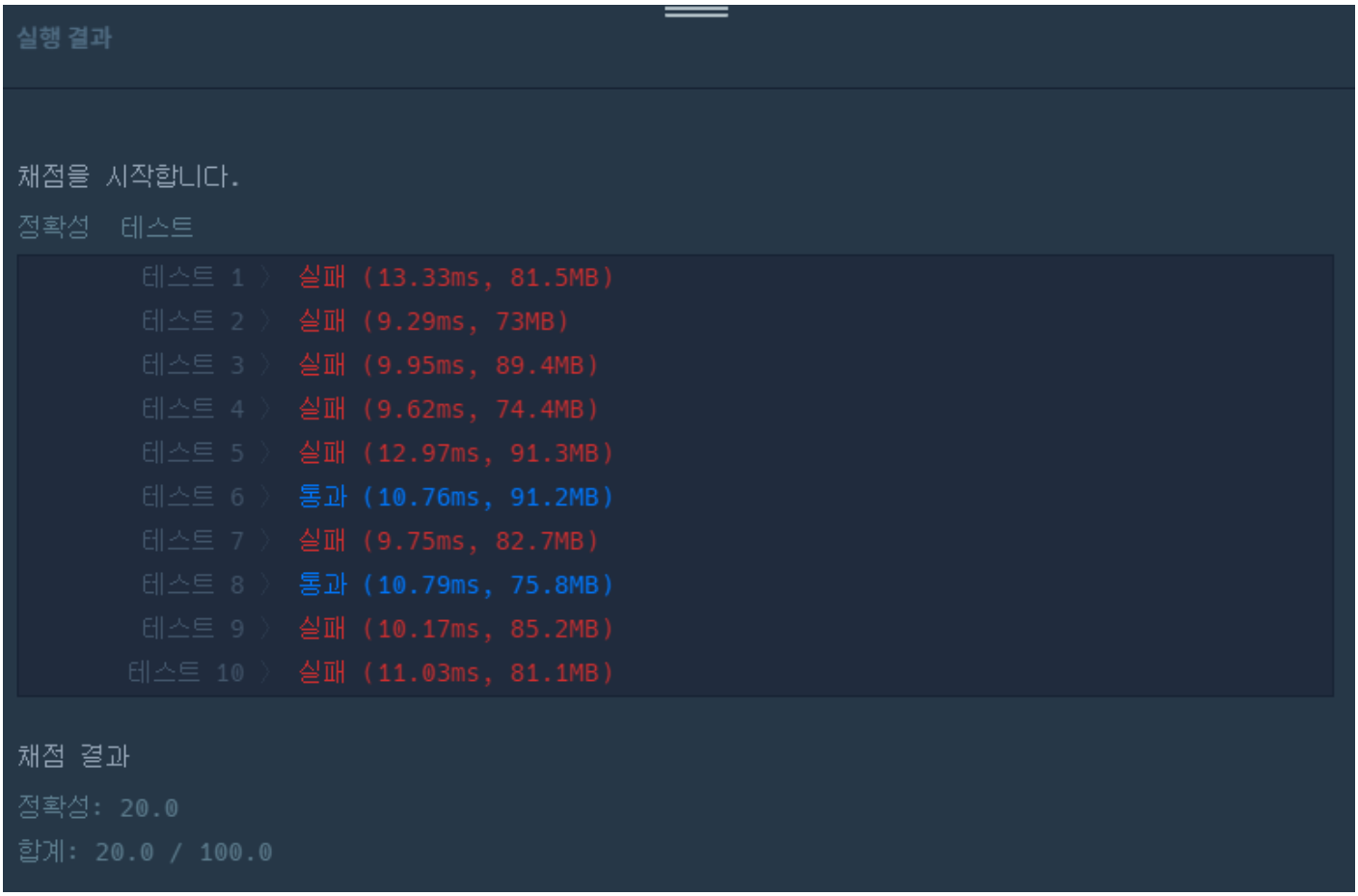
```
String my_string = "a12b3";
→ arr = ['a', '1', '2', 'b', '3']
→ 결과: 1 + 2 + 3 = 6 (□ □□ 12 + 3이 아님)
```

문제에서 원하는 것은 만약 "12 + 3" → 15로 계산하는 것이다.

String num 변수를 밖으로 빼고, Character.isDigit() 으로 c가 0~9사이 숫자형 문자열이면 num에 저장하는 형식으로 코드를 짜 주었다.

틀린 코드 (3차 시도)

```
class Solution {
    public int solution(String my_string) {
        char[] arr = my_string.toCharArray();
        int sum = 0;
        String num = "";
        for (char c : arr ) {
```



```
            if (Character.isDigit(c)) {
                num += c;
            } else {
                if (!num.isEmpty()) {
                    sum += Integer.parseInt(num);
                    num = "";
                }
            }
        }
        if (!num.isEmpty()) {
            sum += Integer.parseInt(num);
        }
        return sum;
    }
}
```

실행 결과

```
채점을 시작합니다.
정확성  테스트
        테스트 1 〉  실패 (13.33ms, 81.5MB)
        테스트 2 〉  실패 (9.29ms, 73MB)
        테스트 3 〉  실패 (9.95ms, 89.4MB)
        테스트 4 〉  실패 (9.62ms, 74.4MB)
        테스트 5 〉  실패 (12.97ms, 91.3MB)
        테스트 6 〉  통과 (10.76ms, 91.2MB)
        테스트 7 〉  실패 (9.75ms, 82.7MB)
        테스트 8 〉  통과 (10.79ms, 75.8MB)
        테스트 9 〉  실패 (10.17ms, 85.2MB)
       테스트 10 〉  실패 (11.03ms, 81.1MB)

채점 결과
정확성: 20.0
합계: 20.0 / 100.0
```

덧셈만 하는게 아니라 뺄셈도 있었다 . 연산자 (+, -)에 따라 연산을 모두 처리하는 코드를 짜야 한다 .

1. 문자열을 " "(공백) 기준으로 나눠서
2. 처음 값은 result에 저장
3. 이후에는 연산자 (+ 또는 -)와 숫자가 번갈아 등장

4. 연산자에 따라 +면 더하고, -면 빼야 함

틀린 코드 (4차 시도)

```
class Solution {
    public int solution(String my_string) {
        String[] splited = my_string.split(" ");
        int answer = Integer.parseInt(splited[0]);

        for(int i = 1; i < splited.length; i += 2) {
            String op = splited[i];
            int num = Integer.parseInt(splited[i + 1]);

            if (op == '+')) {
                answer += num;
            } else if (op == '-') {
                answer -= num;
            }
        }
        return answer;
    }
}
```

? if(op == '+')

- 이 코드는 컴파일 에러가 난다.
- op는 String 타입으로 선언되었는데 '+'는 char로 타입이 달라서 비교 불가능하기 때문이다.
- Java에서는 문자열 내용을 비교할 때 ==를 쓰지 않는다. ==는 두 객체가 같은 주소(참조)를 가리킬 때만 true이다.
- 내용 비교는 반드시 .equals()를 써야 한다.

개선한 코드

```
class Solution {
    public int solution(String my_string) {
        String[] splited = my_string.split(" ");
        int answer = Integer.parseInt(splited[0]);
```

```
        for(int i = 1; i < splited.length; i += 2) {
            String op = splited[i];
            int num = Integer.parseInt(splited[i + 1]);

            if (op.equals("+")) {
                answer += num;
            } else if (op.equals("-")) {
                answer -= num;
            }
        }
        return answer;
    }
}
```

9. [Java] ??? ???(??, Set) - set.contains(str)

> https://school.programmers.co.kr/learn/courses/30/lessons/120903

> Set 을 활용한 코드

```
import java.util.*;

class Solution {
    public int solution(String[] s1, String[] s2) {
        Set<String> set = new HashSet<>(Arrays.asList(s1));
        int cnt = 0;
        for (String str : s2) {
            if (set.contains(str)) cnt++;
        }
        return cnt;
    }
}
```

? Set<String> set = new HashSet<>(Arrays.asList(s1));

- Set은 중복을 허용하지 않고 , 검색 속도가 빠르기 때문에 비교 대상인 s1을 Set으로 변환하면 s2의 각 원소가 포함되어 있는지 빠르게 확인할 수 있다 .

> 이중 for문 (반복문)을 사용한 코드

```
class Solution {
    public int solution(String[] s1, String[] s2) {
        int cnt = 0;
        for (int i = 0; i < s1.length; i++) {
            for (int j = 0; j < s2.length; j++) {
```

```
                if (s1[i].equals(s2[j])) {
                    cnt++;
                    break;
                }
            }
        }
        return cnt;
    }
}
```

- 중복 카운팅을 피하려면 break;를 해야 한다.
- 문자열끼리 내용 비교는 .equals() 메서드를 사용해야 한다 (== 은 주소값 비교)

10. [Java] ?? ?? - String.indexOf(), String.valueOf()

> https://school.programmers.co.kr/learn/courses/30/lessons/120904

> 오답

```
class Solution {
    public int solution(int num, int k) {
        String numToString = Integer.toString(num);
        char[] arr = numToString.toCharArray();
        char kToChar = (char) k;
        int idx = -1;
        for(int i = 0; i < arr.length; i++ ) {
            if (arr[i] == kToChar) {
                return i+1;
            }
        }
        return idx;
    }
}
```

? char kToChar = (char) (k + '0');

- 틀린 부분 : char kToChar = (char) k;
- int 타입을 char 타입으로 변환하는 방법이 잘못됨
- → int 3을 (char)로 바꾸면 '3'이 아니라 아스키 문자 'ETX' 같은 이상한 문자가 됨
- → char kToChar = (char)(k + '0'); 또는 Character.forDigit(k, 10); 로 고쳐야 함

> 개선한 코드

```
class Solution {
    public int solution(int num, int k) {
        String numToString = Integer.toString(num);
        char[] arr = numToString.toCharArray();
```

```
        char kToChar = (char) (k + '0');
        int idx = -1;
        for(int i = 0; i < arr.length; i++ ) {
            if (arr[i] == kToChar) {
                return i+1;
            }
        }
        return idx;
    }
}
```

> 더 간단한 방법 (String API 활용)

```
public int solution(int num, int k) {
    String s = String.valueOf(num);
    int idx = s.indexOf(String.valueOf(k));
    return idx == -1 ? -1 : idx + 1;
}
```

? String.indexOf()

문자열 내에서 특정 문자나 문자열이 처음 등장하는 위치의 인덱스를 알려주는 매우 유용한 메서드이다 .

기본 사용법

```
String s = "29183";
int idx = s.indexOf("1");   // 결과: 2
```

- 문자열 "1"이 "29183"에서 0-based index 2에 처음 등장
- 못 찾으면 -1을 반환

자바 공식 문법

```
int indexOf(int ch)           // 문자 또는 문자 코드 (ex: 'a', 97)
int indexOf(String str)       // 문자열
```

? ?????

자바의 삼항 연산자 (ternary operator)는 if-else를 한 줄로 간단하게 쓸 수 있게 해주는 조건 연산자이다 .

기본 사용법

```
조건식 ? 참일 때 값 : 거짓일 때 값;
```

- 조건식이 true이면 ? 다음의 값을 반환
- false이면 : 다음의 값을 반환

예제

```
int age = 20;
String result = (age >= 18) ? "성인" : "미성년자";
System.out.println(result);  // 출력: 성인
```

11. [Java] ???? ????(2): case2?? longer - shorter???

> https://school.programmers.co.kr/learn/courses/30/lessons/120868

- 프로그래머스 삼각형의 완성조건 (2)
- 두 변이 주어졌을 때 , 나머지 한 변 x가 삼각형이 될 수 있는 정수 개수 를 구하는 문제
- 삼각형이 되기 위한 조건 : 가장 긴 변 < 나머지 두 변의 합

> 정수 범위 공식

- x ∈ (longer - shorter, longer + shorter)
- 즉 , 가능한 정수 x의 범위는
- longer - shorter + 1 ≤ x ≤ longer + shorter - 1

> 정답 코드

```
import java.util.*;

class Solution {
    public int solution(int[] sides) {
        Arrays.sort(sides); // 배열 오름차순 정렬
        int shorter = sides[0];
        int longer = sides[1];

        // case1: x가 가장 긴 변일 때 -> x < a + b
        int case1 = longer + shorter - 1;

        // case2: x가 짧거나 중간인 경우 -> x > max - min
        int case2 = longer - shorter;

        return case1 - case2;
    }
```

```
}
```

---

case2가 longer - shorter + 1이 아니라 longer - short인 이유

- `case2` 실제 시작값은 `longer - shorter + 1`
- 하지만 정수 개수를 계산할 때는 시작값 **- 1** 을 기준으로 계산해야 하므로 `case2 = longer - shorter`로 설정하는 것이 정확함
- 실제 시작값보다 1 작게 설정되어야 하므로 `+1`이나 `-1` 필요 없는 것

예시

- 입력 : sides = [3, 6]
- 가능한 x: 4, 5, 6, 7, 8 → 총 **5**개

- 계산 :
  case1 = 3 + 6 - 1 = 8
  case2 = 6 - 3 = 3
  answer = 8 - 3 = 5 ▯

12. [Java] ???(1): break vs continue

> https://school.programmers.co.kr/learn/courses/30/lessons/120956

- 프로그래머스 옹알이 (1)
- 머쓱이의 조카가 발음할 수 있는 단어의 개수 찾기

---

1. ?? ??

```
import java.util.*;

class Solution {
    public int solution(String[] babbling) {
        String[] canSay = {"aya", "ye", "woo", "ma"};
        int count = 0;

        for (String word : babbling) {
            String temp = word;
            boolean isValid = true;

            for (String say : canSay) {
                // 중복된 발음이 있는지 확인
                if (temp.contains(say + say)) {
                    isValid = false;
                    break;
                }
            }

            if (!isValid) continue;

            // 발음 가능 단어 제거
            for (String say : canSay) {
```

```
                temp = temp.replace(say, " ");
            }

            // 모두 제거되었는지 확인 (공백만 남은 경우 trim()으로 제거)
            if (temp.trim().isEmpty()) {
                count++;
            }
        }

        return count;
    }
}
```

2. ?? ?? ???? ???? ??

```
if (temp.contains(say + say)) {
    isValid = false;
    break;
}
```

- 이 코드는 같은 단어가 두 번 연속 사용된 문자열을 걸러내기 위해 사용된다 .
- 조카는 "ayaaya"처럼 같은 단어를 두 번 연속 발음하지 못하기 때문이다 .
- 그래서 "ayaaya", "mama", "woowoo" 같은 경우는 불가능한 발음이다 .
- 예르 들어 , say = "aya" 라고 할 때 say + say는 "ayaaya"이다 .
- temp.contains("ayaaya")는 temp 문자열 안에 "ayaaya"가 들어 있는지를 확인한다 .

---

3. continue

- isValid가 false일 경우 → 이 단어는 조카가 말할 수 없는 단어이므로 아래 로직은 더 이상 실행하지 않고 , 다음 단어로 넘어감
- 반복문 안에서 continue는 현재 반복을 중단하고 다음 반복으로 건너뜀을 의미한다 .

```
for (String word : babbling) {
    // 어떤 조건으로 유효한 발음인지 검사
    if (!isValid) continue;

    // 여기 있는 코드는 건너뜀!
}
```

- !isValid가 true이면, 즉 isValid == false이면
- → 현재 word는 조건에 맞지 않으니까
- → 아래 replace 등의 처리를 하지 않고
- → 바로 다음 word로 넘어감.

---

4. break

- break의 역할: "반복문 하나만 탈출"해서 즉시 종료
- 이중 루프 중에 안쪽에 있으면 그 안쪽 루프만 빠져나온다.

```
for (String say : canSay) {
    if (temp.contains(say + say)) {
        isValid = false;
        break; // □ □ for 루프를 빠져나옴!
    }
}
```

여기서 break;는 바로 canSay 배열을 순회하는 for문을 탈출한다.

따라서 그 다음 실행되는 건 → continue; 실행됨 → 이 단어는 skip하고 다음 단어로 넘어가는 로직이 실행된다.

```
if (!isValid) continue;
```

---

5. continue VS break

```
for each word in babbling:
    for each say in canSay:
        if word contains say+say:
            isValid = false
            break ← 여기서 안쪽 for문 탈출
    if (!isValid) continue ← 이 줄로 와서 현재 word 스킵
```

- break : 가장 가까운 반복문 전체를 탈출
- continue : 현재 반복을 건너뛰고, 다음 반복을 바로 실행

-

13. [Java] ?????4: stack ? int[] ??

> https://school.programmers.co.kr/learn/courses/30/lessons/181918?language=java

1. ?? ??

```
import java.util.*;

class Solution {
    public int[] solution(int[] arr) {
        Stack<Integer> stk = new Stack<>();
        int i = 0;

        while (i < arr.length) {
            if (stk.isEmpty()) {
                stk.push(arr[i]);
                i++;
            } else if (stk.peek() < arr[i]) {
                stk.push(arr[i]);
                i++;
            } else {
                stk.pop();
            }
        }

        // stack → int[]
        int[] answer = new int[stk.size()];
        for (int j = 0; j < stk.size(); j++) {
            answer[j] = stk.get(j);
        }
        return answer;
    }
```

```
}
```

2. for? ?? stack ? int[] ??

```
// stack → int[]
int[] answer = new int[stk.size()];
for (int j = 0; j < stk.size(); j++) {
    answer[j] = stk.get(j);
}
return answer;
```

- 장점
  - 가장 빠름 (불필요한 추상화 없음)
  - 성능 최적화에 유리
  - 자바의 기본 동작에 가장 가까움
- 단점
  - 약간 길어질 수 있음

3. stream() ?? stack ? int[] ??

```
// stream을 이용한 stack → int[] 변환
return stk.stream().mapToInt(Integer::intValue).toArray();
```

- stk.stream() : Stack<Integer> → Stream<Integer>
- .mapToInt(Integer::intValue) : Stream<Integer> → IntStream
- .toArray() : IntStream → int[]
- 장점
  - 코드 간결 , 가독성 좋음
  - 함수형 스타일 선호 시 유리
- 단점
  - 내부적으로 반복자 (iterator) 생성 및 `lambda` 적용 → 약간 느림
  - 작은 입력에서는 차이 없음 , 큰 배열 에서는 미세한 오버헤드

---

4. ?? ??

테스트 케이스 크기	**for-loop**	**stream**
100개	약 0.01ms	약 0.01ms

100,000개	약 1.5ms	약 2.2ms
1,000,000개	약 15ms	약 22ms

차이 있음: 데이터가 클수록 `stream()`이 약간 느림

하지만 일반적인 코테에서는 이 정도 차이는 오답 판정으로 이어질 정도는 아님

- 코테에서 성능이 아주 중요한 상황(ex. 제한시간 빠듯한 문제, 수십만~백만 이상 원소 다룸)에서는 for문을 쓰는 것이 좋다.
- 간결하게 표현하고 싶거나 입력 크기가 작을 때는(ex. 1000 이하) stream()을 사용해도 무방하다.

14. [Java] ??? ?? 3

> https://school.programmers.co.kr/learn/courses/30/lessons/181916

1. ??? ?? ?? ??

1. 네 주사위가 모두 같을 경우
   - 조건 : 하나의 숫자가 4번 등장
   - 점수 : 1111 × p
2. 세 개가 같고 하나가 다를 경우
   - 조건 : 한 숫자가 3번 , 다른 숫자가 1번 등장
   - 점수 : $(10 \times p + q)^2$
3. 두 개씩 같은 숫자가 나올 경우
   - 조건 : 두 숫자가 각각 2번씩 등장
   - 점수 : $(p + q) \times |p - q|$
4. 한 쌍과 나머지 두 숫자가 각각 다를 경우
   - 조건 : 한 숫자 2번 , 나머지 두 숫자는 각각 1번 (총 세 가지 숫자)
   - 점수 : 나머지 두 숫자의 곱 (q × r)
5. 모두 다른 숫자가 나올 경우
   - 조건 : 네 숫자가 모두 다름
   - 점수 : 가장 작은 숫자

---

2. ?? ??

```
import java.util.*;

class Solution {
    public int solution(int a, int b, int c, int d) {
```

```
int[] dice = {a, b, c, d};
Map<Integer, Integer> count = new HashMap<>();

// □□ □□
for (int num : dice) {
    count.put(num, count.getOrDefault(num, 0) + 1);
}

int size = count.size();
List<Integer> keys = new ArrayList<>(count.keySet());

if (size == 1) {
    // case 1: □ □ □□ □□ □
    int p = keys.get(0);
    return 1111 * p;

} else if (size == 2) {
    Collection<Integer> freqs = count.values();
    if (freqs.contains(3)) {
        // case 2: (p, p, p, q)
        int p = 0, q = 0;
        for (int key : keys) {
            if (count.get(key) == 3) p = key;
            else q = key;
        }
        return (int) Math.pow(10 * p + q, 2);
    } else {
        // case 3: (p, p, q, q)
        int p = keys.get(0);
        int q = keys.get(1);
        return (p + q) * Math.abs(p - q);
    }

} else if (size == 3) {
    // case 4: (p, p, q, r)
    int result = 1;
    for (int key : keys) {
        if (count.get(key) == 1) result *= key;
    }
    return result;
```

```
        } else {
            // case 5: 모두 다름
            return Arrays.stream(dice).min().getAsInt();
        }
    }
}
```

Map 쓰는 이유?

문제에서 "같은 숫자가 몇 개 나왔는지"를 기준으로 조건이 분기됨

문제 조건 다시 보면:

- 숫자가 **4**개 모두 같을 때
- 숫자가 **3**개 같고 **1**개 다를 때
- 숫자가 **2**개, **2**개 같을 때
- 숫자가 **2**개 같고 나머지 각각 다를 때
- 숫자 4개가 모두 다를 때

즉, **"각 숫자가 몇 번 나왔는지"를 기준으로 판단**해야 한다.
이 말은 곧 **"숫자 → 빈도 수"를 빠르게 셀 수 있는 구조**가 필요하다는 뜻이다.

그래서 Map<Integer, Integer> 구조가 적절하다. Map은 키(숫자)마다 몇 번 나왔는지(빈도)를 저장할 수 있어서 조건 분기문(예: if (count.size() == 2) 등)을 짜기 매우 직관적이다.

물론 이 문제에서는 숫자가 1~6 범위로 작고, 고정이므로 빈도 배열 int[7] freq도 가능하다.

예를 들어:

```
int[] freq = new int[7];
freq[a]++;
freq[b]++;
freq[c]++;
freq[d]++;
```

→ 숫자가 1~6로 제한되어 있다는 걸 사전에 알 때만 배열이 가능하고, 그 외 범위에서는 Map이 더 일반적이고 확장성 있다.

언제 Map을 써야 하는가?

“값 → 등장 횟수” 구조가 필요할 때 `Map`을 쓴다 . 즉 , Map은 "무엇이 얼마나 나왔는가 "라는 질문에 답해야 할 때 가장 먼저 떠올려야 할 자료구조이다 . 특히 문제에서 값의 개수 , 빈도 , 중복 여부가 중요한 역할을 한다면 , Map은 거의 정답에 가깝다고 보면 된다 .

- 값이 몇 번 나왔는지 셀 때
  예 : 숫자 3번 등장 여부 확인
- 중복된 값 개수로 조건을 나눌 때
  예 : 주사위 2개씩 같은 경우 등
- 문자나 숫자의 빈도 계산이 필요할 때
  예 : 문자열 문자 수 세기

> 프로그래머스에서 좋아요 제일 많이 받은 코드

??? , flao1 , plotojjy@gmail.com , sksmsgla2@gmail.com ? 146 ?

```
import java.util.Arrays;

class Solution {
    public int solution(int a, int b, int c, int d) {

        int[] dice = { a, b, c, d };
        Arrays.sort(dice);

        int ans = 0;

        if (dice[0] == dice[3]) {
            ans = 1111 * dice[3];
        } else if (dice[0] == dice[2] || dice[1] == dice[3]) {
            ans = (int) Math.pow(dice[1] * 10 + (dice[0] + dice[3] - dice[1]), 2);
        } else if (dice[0] == dice[1] && dice[2] == dice[3]) {
            ans = (dice[0] + dice[3]) * (dice[3] - dice[0]);
        } else if (dice[0] == dice[1]) {
            ans = dice[2] * dice[3];
        } else if (dice[1] == dice[2]) {
            ans = dice[0] * dice[3];
        } else if (dice[2] == dice[3]) {
            ans = dice[0] * dice[1];
        } else {
            ans = dice[0];
        }
```

```
        return ans;
    }
}
```

15. [Java] 9? ?? ???

> https://school.programmers.co.kr/learn/courses/30/lessons/181914

> 오답

```
class Solution {
    public int solution(String number) {
        int answer = 0;
        int sum = 0;
        for (int i = 0; i < number.length(); i++) {
            sum += number.charAt(i);
        }
        answer = sum % 9;
        return answer;
    }
}
```

위 코드에서 틀린 부분은 number을 순회하면서 sum변수에 누적하는 과정이다 .

```
sum += number.charAt(i);
```

- charAt(i)가 문자 (char)라서 아스키코드 값으로 더해지고 있기 때문이다.
- 3 → 아스키코드 51
- 우리가 원하는 건 숫자 3인데, 실제로는 51 이 더해져서 전체 합이 틀어진다.

```
sum += number.charAt(i) - '0';
```

- 문자 - '0' 으로 정수 변환해서 해결했다.
- '3' - '0' → 51 - 48 →3

> 개선한 코드

```
class Solution {
    public int solution(String number) {
        int answer = 0;
        int sum = 0;
        for (int i = 0; i < number.length(); i++) {
            sum += number.charAt(i) - '0';
        }
        answer = sum % 9;
        return answer;
    }
}
```

실행 결과

```
테스트 14 〉 통과 (0.38ms, 85.1MB)
테스트 15 〉 통과 (0.39ms, 74MB)
테스트 16 〉 통과 (2.97ms, 92.1MB)
테스트 17 〉 통과 (2.38ms, 83.5MB)
테스트 18 〉 통과 (1.90ms, 73.4MB)
테스트 19 〉 통과 (4.25ms, 79.8MB)
테스트 20 〉 통과 (3.80ms, 76.4MB)
테스트 21 〉 통과 (6.23ms, 83.4MB)
테스트 22 〉 통과 (5.12ms, 91.8MB)
테스트 23 〉 통과 (4.44ms, 77.1MB)
테스트 24 〉 통과 (5.78ms, 94MB)
테스트 25 〉 통과 (0.02ms, 73.8MB)
```

채점 결과

정확성: 100.0

합계: 100.0 / 100.0