Professional Coding Specialist

COS Pro JAVA 1 급

19 강-20 강. 모의고사 4 차

1. 모의고사 4 차(1-10 번)

과정 소개

COS Pro JAVA 1 급 모의고사 4 차를 풀어보며 문제 유형을 익히고, JAVA 를 이용하여 알고리즘을 구현하기 위해 필요한 관련 지식을 익혀보도록 한다.

학습 목차

학습 목표

1. YBM IT(www.ybmit.com) 에서 제공하는 COS Pro JAVA 1 급 모의고사(샘플 문제)를 풀어보며 JAVA 를 이용하여 주어진 문제를 해결하기 위한 알고리즘을 구성하는 능력을 배양한다.
2. 많이 등장하는 문제 유형을 익혀서 COS Pro 1 급 시험에 대비한다.

1. 문제 1

1) 문제 코드

```java
/*=======================================
   4차 1번   4차 1급 1_initial_code.java
 =======================================*/

import java.util.*;

public class Solution {
    String[] vowels = {"A", "E", "I", "O", "U"};
    ArrayList<String> words;
    public void create_words(int lev, String str) {
        words.add(str);
        for(int i=0;i<5;i++) {
            if(lev<5) {
                create_words(lev,str+vowels[i]);
            }
        }
    }

    public int solution(String word) {
        int answer = 0;
        words  = new ArrayList<String>();
        create_words(0,"");

        for(int i=0; i<words.size();i++) {
            if (word.equals(words.get(i))) {
                answer=i;
                break;
            }
        }

        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으
    public static void main(String[] args) {
        Solution sol = new Solution();
        String word1 = new String("AAAAE");
        int ret1 = sol.solution(word1);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");

        String word2 = new String("AAAE");
        int ret2 = sol.solution(word2);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제
- 재귀 호출 함수를 이용하여 주어진 알파벳으로 길이가 5 자 이하의 단어들을 생성한 후 제시된 단어가 몇 번째 단어인지 알아내는 프로그램에서 잘못된 곳을 찾아 한 줄 수정하는 문제
- 사전의 첫 번째 단어는 'A', 두 번째 단어는 'AA', 세 번째 단어는 'AAA' 순으로 단어를 생성하여 마지막 단어는 'UUUUU' 로 마무리

3) 재귀 함수

① 특징

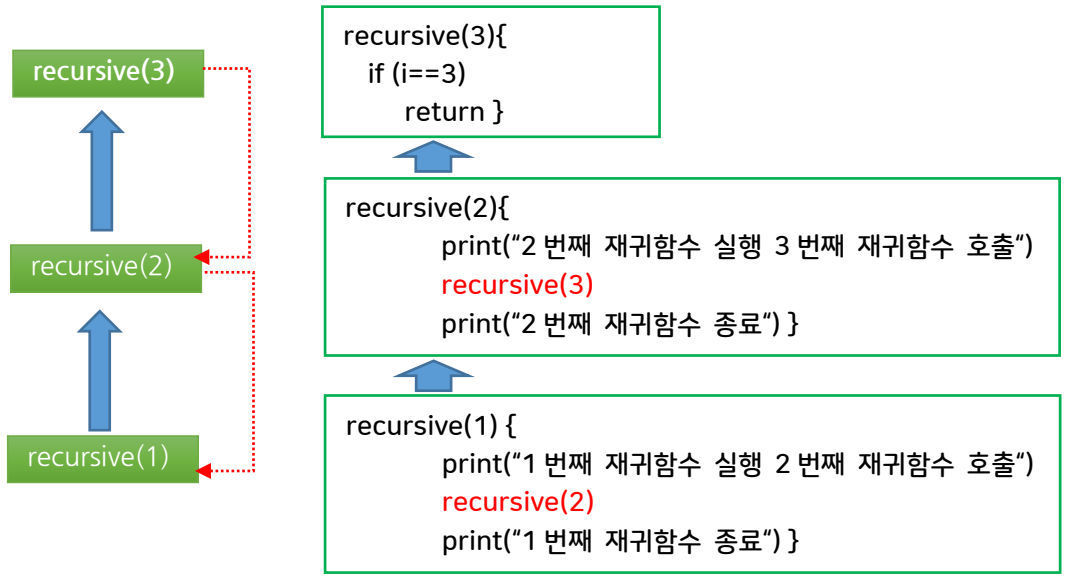
- 자기 자신을 호출하는 함수
- 호출된 함수 실행이 끝나면 마지막으로 호출한 함수부터 꺼내는 스택(Stack) 방식
- 재귀 함수를 호출할 때마다 함수 안의 지역 변수를 새로 생성하므로 지나친 재귀호출은 메모리를 과다 사용할 수 있고, 함수 호출/리턴하는 과정에서 프로세스 교환이 빈번히 일어나므로 성능 저하를 일으킬 수 있음

② 재귀 함수 예

```
public class ExamRecursive {
    public void recursive(int i) {
        if (i==3)
            return;

        System.out.println(i+"번째 재귀함수 실행 "+(i+1)+"번째 재귀함수 호출");
        recursive(i+1);
        System.out.println(i+"번째 재귀함수 종료");
    }

    public static void main(String[] args) {
        ExamRecursive recur= new ExamRecursive();
        recur.recursive(1);
    }
}
```



➔ recursive(2) 에서 recursive(3) 를 호출하기 때문에 recursive(3) 종료 후 recursive(2)에서 recursive(3) 를 호출한 다음 명령으로 돌아감

➔ recursive(1) 에서 recursive(2) 를 호출하기 때문에 recursive(2) 종료 후 recursive(1)에서 recursive(2) 를 호출한 다음 명령으로 돌아감

〈실행 결과〉

```
1 번째 재귀함수 실행 2 번째 재귀함수 호출
2 번째 재귀함수 실행 3 번째 재귀함수 호출
2 번째 재귀함수 종료
1 번째 재귀함수 종료
```

4) 정답

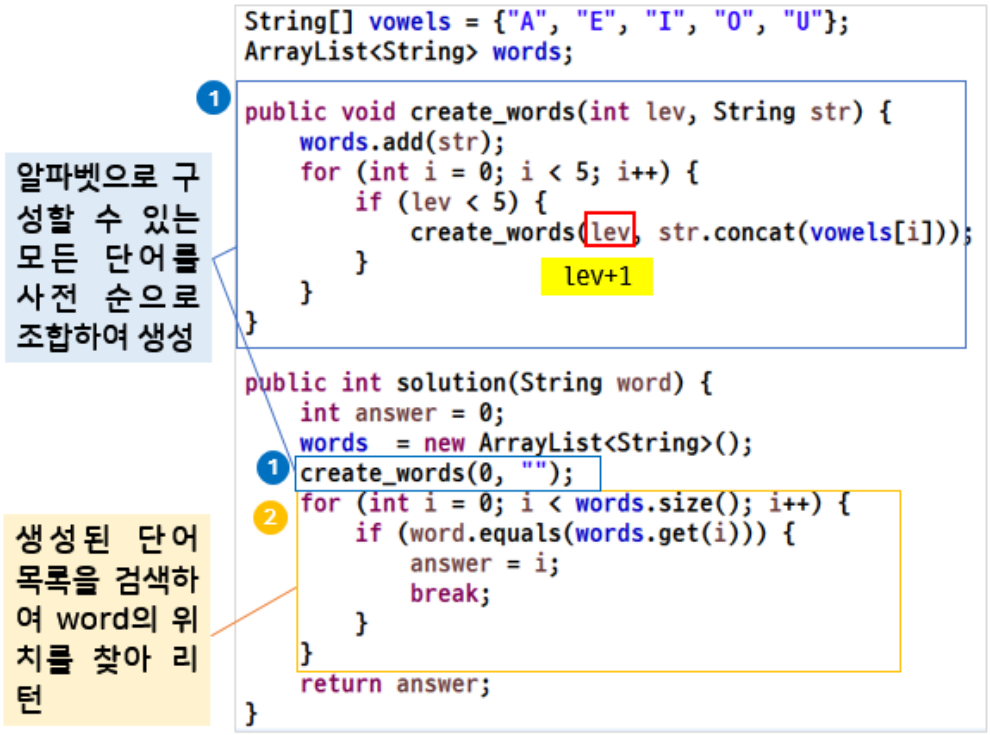
- 사전식 단어 생성 절차

글자 수 / 생성 순서	1	2	3	4	5
1	A				
2	A	A			
3	A	A	A		
4	A	A	A	A	
5	A	A	A	A	A
6	A	A	A	A	E
7	A	A	A	A	I
8	A	A	A	A	O
9	A	A	A	A	U
10	A	A	A	E	
11	A	A	A	E	A
…	…				

생성 단어 수

1 글자 수 단어 (5)
+ 2 글자 수 단어 (5*5)
+ 3 글자 수 단어 (5*5*5)
+ 4 글자 수 단어 (5*5*5*5)
+ 5 글자 수 단어 (5*5*5*5*5)
= 3905 개

- 정답 코드




```
String[] vowels = {"A", "E", "I", "O", "U"};
ArrayList<String> words;

public void create_words(int lev, String str) {
    words.add(str);
    for (int i = 0; i < 5; i++) {
        if (lev < 5) {
            create_words(lev, str.concat(vowels[i]));
        }
    }
}

public int solution(String word) {
    int answer = 0;
    words  = new ArrayList<String>();
    create_words(0, "");
    for (int i = 0; i < words.size(); i++) {
        if (word.equals(words.get(i))) {
            answer = i;
            break;
        }
    }
    return answer;
}
```

① 처음에 solution 메소드에서 create_words(0,”) 를 이용해 0 번 인덱스에 빈 값을 초기화 매개변수 str 에 있는 문자열을 words 배열에 추가

create_words() 함수

- for 문 반복을 이용하여 vowels 의 알파벳으로 구성된 5 자 이하의 단어를 생성 : 사전 순으로 조합하여 단어를 생성
- 매개변수 lev 의 값이 5 보다 작으면(현재 s 가 갖고 있는 문자열의 길이가 5 보다 작으면) 재귀 호출
- 재귀호출할 때 lev 의 값을 1 만큼 증가시킨 값과 매개변수 s 에 있는 문자열에 vowels[i] 번째 문자를 합한 것을 인수로 전달
- 문제코드에서 제시한 것처럼 재귀호출할 때, lev 값을 그대로 전달하면 무한하게 재귀호출하는 문제가 발생함

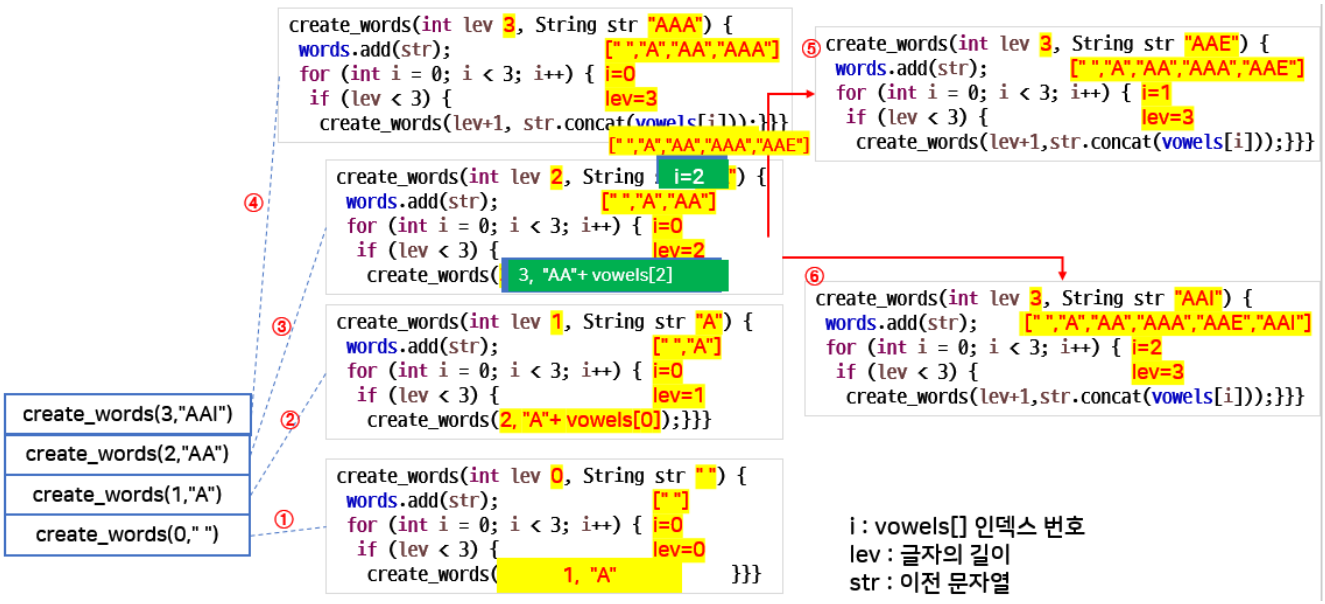
② 5 자 이하의 단어들을 순서대로 모아 놓은 배열 words 에서 단어를 가져와 매개변수 word 와 words 의 항목값 i 가 같으면 해당 항목을 가리키는 인덱스를 answer 로 저장한 후 break 명령을 이용하여 반복문 실행 종료

5) 참고 : 정답 코드의 create_words( ) 함수의 실행 process

: 알파벳 3 개 'A', E', 'I' 만을 사용하여 단어를 만드는 경우

```
String[] vowels = {"A", "E", "I"};
```

```
create_words(int lev , String str ) {
 words.add(str);
 for (int i = 0; i < 3; i++) {
   if (lev < 3) {
```



① create_words(0," ")를 실행하면 words 에 " "을 추가하고 create_words(1,"A") 호출

② create_words(1,"A")를 실행하면 words 에 "**A**"을 추가하고 create_words(2,"AA") 호출

③ create_words(2,"AA")를 실행하면 words 에 "**AA**"을 추가하고 create_words(3,"AAA") 호출

④ create_words(3,"AAA")를 실행하면 words 에 "**AAA**"을 추가하고 종료

➔ create_words(3,"AAA")가 종료되면 recursion stack 에서 create_words(3, "AAA") 관련 정보를 삭제하고, create_words(2, "AA")로 돌아가서 create_words(3,"AAE") 호출

⑤ create_words(3,"AAE")를 실행하면 words 에 "**AAE**"을 추가하고 종료

➔ create_words(3,"AAE")가 종료되면 recursion stack 에서 create_words(3, "AAE") 관련 정보를 삭제하고, create_words(2, "AA")로 돌아가서 create_words(3,"AAI") 호출

⑥ create_words(3,"AAI")를 실행하면 words 에 "**AAI**"을 추가하고 종료

➔ create_words(3,"AAI")가 종료되면 recursion stack 에서 create_words(3, "AAI") 관련 정보를 삭제한 후 create_words(2, "AA")로 돌아가고, create_words(2,"AA") 안에서 for 문에 의한 모든 반복이 실행되었으므로 create_words(2,"AA")를 종료

➔ create_words(2,"AA")가 종료되면 recursion stack 에서 create_words(2,"AA") 관련 정보를 삭제하고, create_words(1,"A")로 돌아가서 create_words(2,"AE") 호출

➔ 이러한 과정을 create_words(3,"III")를 실행할 때까지 진행

6) 브루트 포스 방식으로 구현한 사전식 단어 구성

```
String[] vowels = {"A", "E", "I", "O", "U"};
ArrayList<String> words;
```

```
int answer = 0;
words  = new ArrayList<String>();
words.add("");

for (int i1 = 0; i1 < 5; i1++) {
①   words.add(vowels[i1]);
    for (int i2 = 0; i2 < 5; i2++) {
②       words.add(vowels[i1]+vowels[i2]);
        for (int i3 = 0; i3 < 5; i3++) {
③           words.add(vowels[i1]+vowels[i2]+vowels[i3]);
            for (int i4 = 0; i4 < 5; i4++) {
④               words.add(vowels[i1]+vowels[i2]+vowels[i3]+vowels[i4]);
                for (int i5 = 0; i5 < 5; i5++) {
⑤                   words.add(vowels[i1]+vowels[i2]+vowels[i3]+vowels[i4]+vowels[i5]);
                 }
            }
        }
    }
}

for(int i=1; i<50;i++)
    System.out.println(words.get(i));
```

- 5 개의 중첩 for 문을 이용하여 1 글자 단어부터 5 글자 단어까지 사전 순으로 생성하여 words 에 저장
  - ①. 1 글자 단어 추가
  - ②. 2 글자 단어 추가
  - ③. 3 글자 단어 추가
  - ④. 4 글자 단어 추가
  - ⑤. 5 글자 단어 추가
- 재귀 함수로 구현할 때보다 실행 속도가 빠름

2. 문제 2

1) 문제 코드

```java
/*=====================================
   4차 2번   4차 1급 2_initial_code.java
 =====================================*/

import java.util.*;

class Solution {
    public String solution(String s) {
        s = s.toLowerCase();
        String answer = "";
        char previous = s.charAt(0);
        int counter = 1;

        for(int i=1; i<s.length(); i++){
            if(s.charAt(i) == previous)
                counter++;
            else {
                answer += previous;
                answer += counter;
                counter = 1;
                previous = s.charAt(0);
            }
        }
        answer += previous;
        answer += counter;
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없
    public static void main(String[] args) {
        Solution sol = new Solution();
        String s = new String("YYYYYbbbBbbBBBMmmM");
        String ret = sol.solution(s);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제
- 여러 개의 문자가 붙어 있는 단어를 문자와 빈도수로 압축하여 표현하는 문제에서 잘못된 곳을 찾아 수정하는 문제

3) 정답

```
public String solution(String s) {
    s = s.toLowerCase();
    String answer = "";
①   char previous = s.charAt(0);
    int counter = 1;

②   for(int i=1; i<s.length(); i++){
        if(s.charAt(i) == previous)
            counter++;
        else {
            answer += previous;
            answer += counter;
            counter = 1;
            previous = s.charAt(i);
        }
    }
③   answer += previous;
    answer += counter;
    return answer;
}
```

① 이전 문자를 저장하는 변수 previous 를 s 의 0 번 문자를 소문자로 변환하여 초기화

② for 문을 이용하여 s 의 1 번 인덱스부터 마지막 문자까지 차례대로 previous 와 같은 지 비교

- 비교하는 문자 s.charAt(i)와 previous 가 같으면, 반복되는 횟수를 저장하는 변수 counter 를 증가
- 비교하는 문자 s.charAt(i)와 previous 가 같지 않으면, previous 를 answer 에 추가하고 previous 의 빈도수를 저장하고 있는 counter 를 문자로 변환하여 answer 배열에 추가
- counter 는 1 로, previous 는 s.charAt((i)로 재설정
- 문제 코드에는 previous 를 s.charAt(0)으로 재설정하기 때문에 현재 문자와 이전 문자를 비교하는 것이 아니라 s.charAt(1)부터 마지막 문자까지 계속 s 의 첫 번째 문자와 비교하므로 원하는 결과를 얻을 수 없음

③ s 에 있는 마지막 문자와 그 빈도수를 answer 에 추가

3. 문제 3

1) 문제 코드

```
/*=====================================
   4차 3번   4차 1급 3_initial_code.java
 =====================================*/

class Solution {
    public long solution(int oneDayPrice, int multiDay, int multiDayPrice, long n){
        if(oneDayPrice * multiDay <= multiDayPrice)
            return n * oneDayPrice;
        else
            return (n % multiDay) * multiDayPrice + (n / multiDay) * oneDayPrice;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래에는 잘못된 부분이 없으니, 위의
    public static void main(String[] args) {
        Solution sol = new Solution();

        int oneDayPrice1 = 3;
        int multiDay1 = 5;
        int multiDayPrice1 = 14;
        long n1 = 6;
        long ret1 = sol.solution(oneDayPrice1, multiDay1, multiDayPrice1, n1);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");

        int oneDayPrice2 = 2;
        int multiDay2 = 3;
        int multiDayPrice2 = 5;
        long n2 = 11;
        long ret2 = sol.solution(oneDayPrice2, multiDay2, multiDayPrice2, n2);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제
- oneDay 티켓은 구매한 날 하루 동안 이용할 수 있는 티켓으로 가격은 oneDayPrice 에 저장
- multiDay 티켓은 구매한 날로부터 multiDay 동안 이용할 수 있는 티켓으로 가격은 multiDayPrice 에 저장
- n 일 동안 이용하는데 필요한 **최소 비용**을 구하기 위해 몫과 나머지를 이용하는 코드에서 한 줄 수정하는 문제

3) 정답

```
class Solution {
  public long solution(int one_day_price, int multi_day, int multi_day_price, long n){
    ① if(one_day_price * multi_day <= multi_day_price)
          return n * one_day_price;
      else
    ②     return (n / multi_day) * multi_day_price + (n % multi_day) * one_day_price;
  }
}
```

① (일일 이용권 금액 x multiDay)의 계산 결과가 multiDay 이용권 금액보다 작거나 같으면 (n x 일일 이용권 금액)을 계산하여 return

② 그렇지 않다면 multiDay 에 대해서는 multiDay 이용권을 사용하는 것이 저렴함

- (이용일 수가 저장된 n 을 multiDay 로 나눈 몫 x multiDay 이용권 금액) + (이용일 수가 저장된 n 을 multiDay 로 나눈 나머지 x 일일 이용권 금액) 을 계산하여 return
- 문제에서 제시된 코드는 계산식을 잘못 구성하였음

4. 문제 4

1) 문제 코드

```java
/*=======================================
   4차 4번   4차 1급 4_initial_code.java
 =======================================*/

import java.util.*;

class Solution {
    public static final int n = 4;

    public ArrayList<Integer> func_a(int[][] matrix) {
        ArrayList<Integer> ret = new ArrayList<Integer>();
        boolean [] exist = new boolean[n * n + 1];
        Arrays.fill(exist, false);
        for (int i = 0; i < n; i ++)
            for (int j = 0; j < n; j++)
                    exist[matrix[i][j]] = true;
        for (int i = 1; i <= n * n; i++)
            if (exist[i] == false)
                ret.add(i);
        return ret;
    }

    public ArrayList<Pair<Integer, Integer> > func_b(int[][] matrix) {
        ArrayList<Pair<Integer, Integer> > ret = new ArrayList<Pair<Integer, Integer> >();
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (matrix[i][j] == 0)
                    ret.add( new Pair<Integer, Integer>(i, j) );
        return ret;
    }
```

```
    public boolean func_c(int[][] matrix) {
        int sum = 0;
        for (int i = 1; i <= n * n; i++)
            sum += i;
        sum = sum / n;
        for (int i = 0; i < n; i++) {
            int rowSum = 0;
            int colSum = 0;
            for (int j = 0; j < n; j++) {
                rowSum += matrix[i][j];
                colSum += matrix[j][i];
            }
            if (rowSum != sum || colSum != sum)
                return false;
        }
        int mainDiagonalSum = 0;
        int skewDiagonalSum = 0;
        for (int i = 0; i < n; i++) {
            mainDiagonalSum += matrix[i][i];
            skewDiagonalSum += matrix[i][n-1-i];
        }
        if (mainDiagonalSum != sum || skewDiagonalSum != sum)
            return false;
        return true;
    }
```

```
    public int[] solution(int[][] matrix) {
        int[] answer = new int[6];
        int ansIdx = 0;
        ArrayList<Pair<Integer, Integer> > coords = func_@@@(@@@);
        ArrayList<Integer> nums = func_@@@(@@@);

        matrix[coords.get(0).getKey()][coords.get(0).getValue()] = nums.get(0);
        matrix[coords.get(1).getKey()][coords.get(1).getValue()] = nums.get(1);
        if (func_@@@(@@@)) {
            for (int i = 0; i <= 1; i++) {
                answer[ansIdx++] = coords.get(i).getKey() + 1;
                answer[ansIdx++] = coords.get(i).getValue() + 1;
                answer[ansIdx++] = nums.get(i);
            }
        }
        else {
            matrix[coords.get(0).getKey()][coords.get(0).getValue()] = nums.get(1);
            matrix[coords.get(1).getKey()][coords.get(1).getValue()] = nums.get(0);
            for (int i = 0; i <= 1; i++) {
                answer[ansIdx++] = coords.get(1-i).getKey() + 1;
                answer[ansIdx++] = coords.get(1-i).getValue() + 1;
                answer[ansIdx++] = nums.get(i);
            }
        }
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[][] matrix = {{16,2,3,13},{5,11,10,0},{9,7,6,12},{0,14,15,1}};
        int[] ret = sol.solution(matrix);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + Arrays.toString(ret) + " 입니다.");
    }
}
```

2) 문제 개요

- 문제 코드 안에 작성된 함수를 파악한 후 제시된 과제를 해결하기 위한 알고리즘대로 알맞은 함수를 호출하도록 코드를 완성하는 문제
- 4 x 4 행렬의 빈 곳을 채우고 채워진 4 x 4 행렬 마방진이 맞는지 확인하여 그 결과를 return 하는 프로그램에서 각 함수의 기능을 확인하여 코드가 바르게 실행하도록 func_a, func_b, func_c 와 매개변수를 채우는 문제

※ 4 차 4 번 문제는 javafx.util.Pair 가 import 되어 있으므로 javafx 를 설치해야 함

※ javafx 를 설치하여 Pair 를 사용하는 문제

※ javafx 의 GUI 를 구현하는 문제가 출제되는 것은 아님

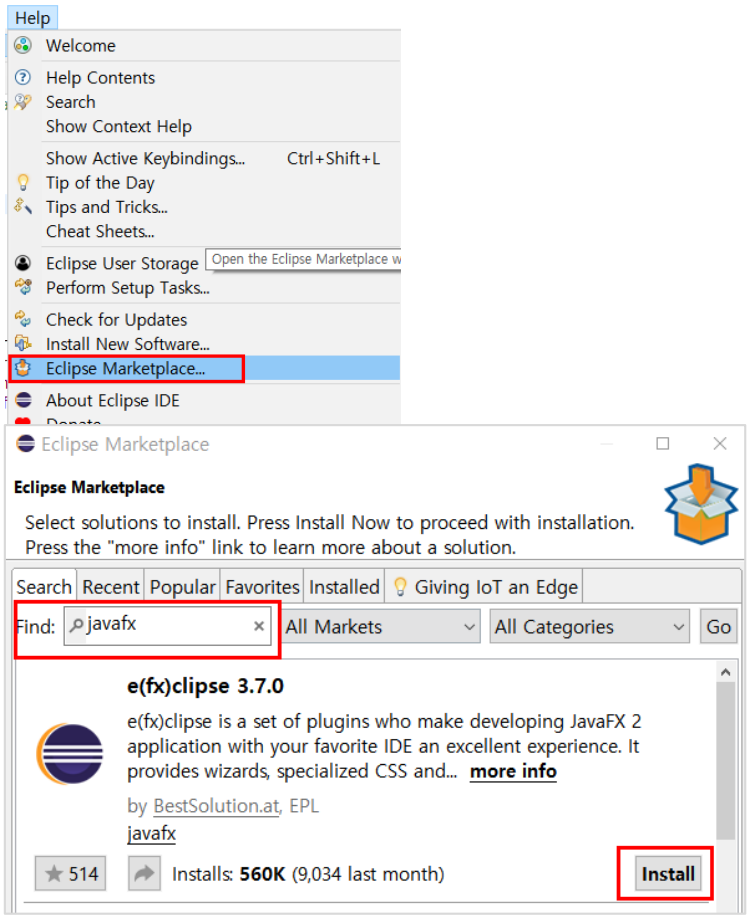
3) 자바 FX(JavaFX)

자바 FX(JavaFX)는 데스크톱 애플리케이션과 리치 인터넷 애플리케이션(RIA)을 개발하고 배포하는 소프트웨어 플랫폼으로, 다양한 장치에서 실행 가능함. 자바 FX 는 자바 SE 를 위한 표준 GUI 라이브러리로서 스윙을 대체하기 위해 고안되었으며, 자바 FX 는 마이크로소프트 윈도우, 리눅스, macOS 의 데스크톱 컴퓨터와 웹 브라우저를 지원함

※ 참조 : 위키백과, https://ko.wikipedia.org/wiki/자바 FX

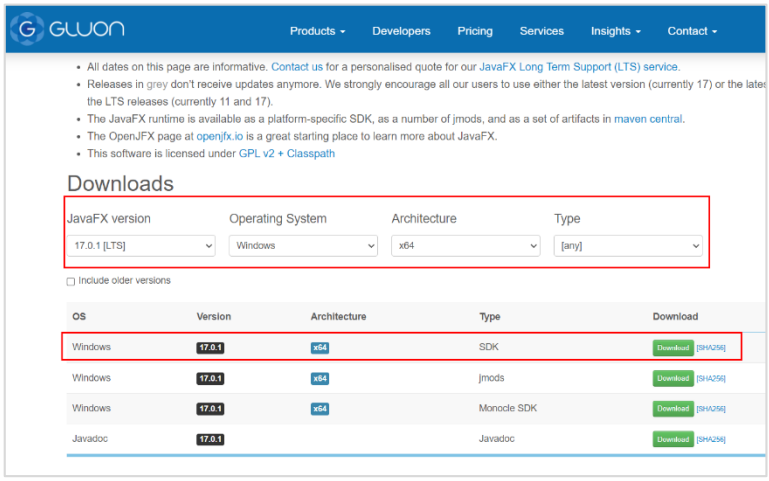
4) javaFX 설치

(1) 이클립스에 javafx 설치

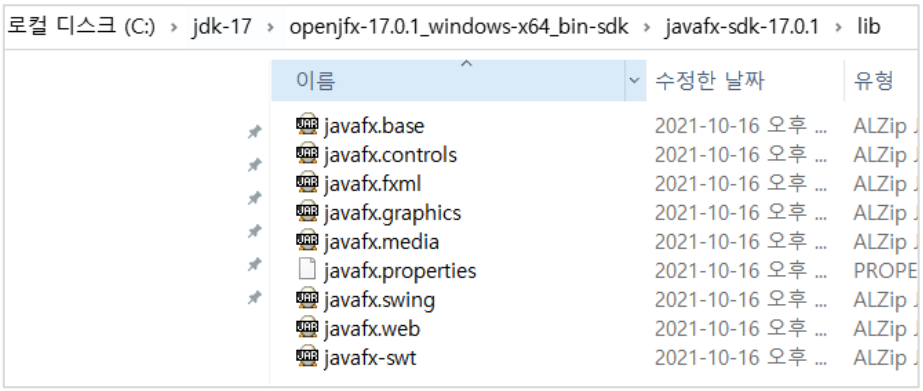


(2) JDK 다운로드

https://gluonhq.com/products/javafx/

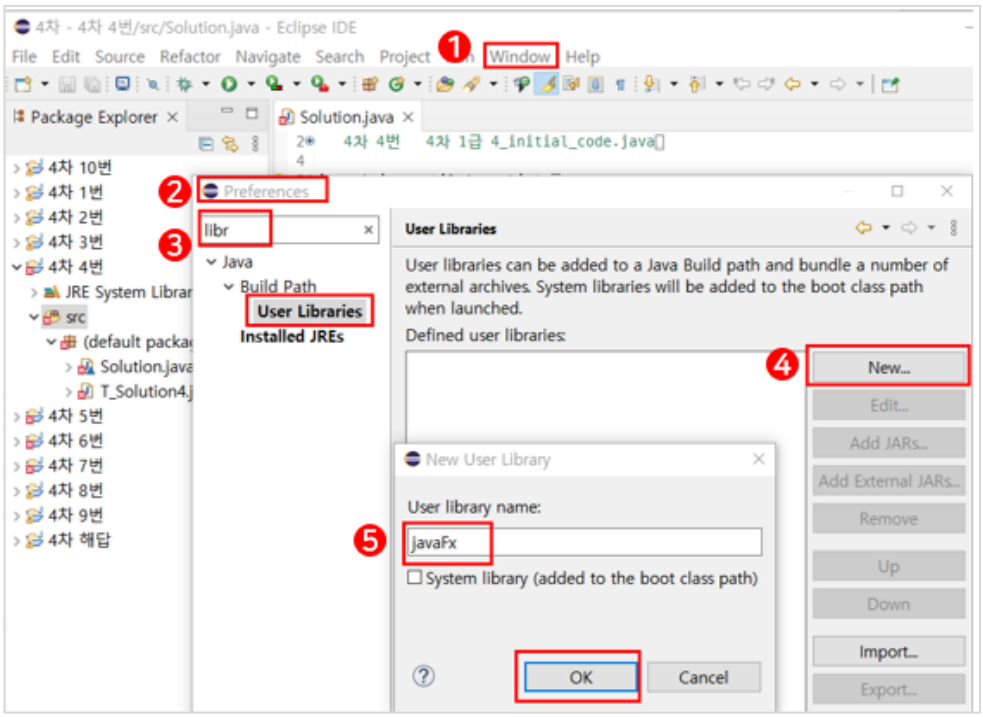


➔ 다운로드 후 원하는 폴더에 압축 풀기

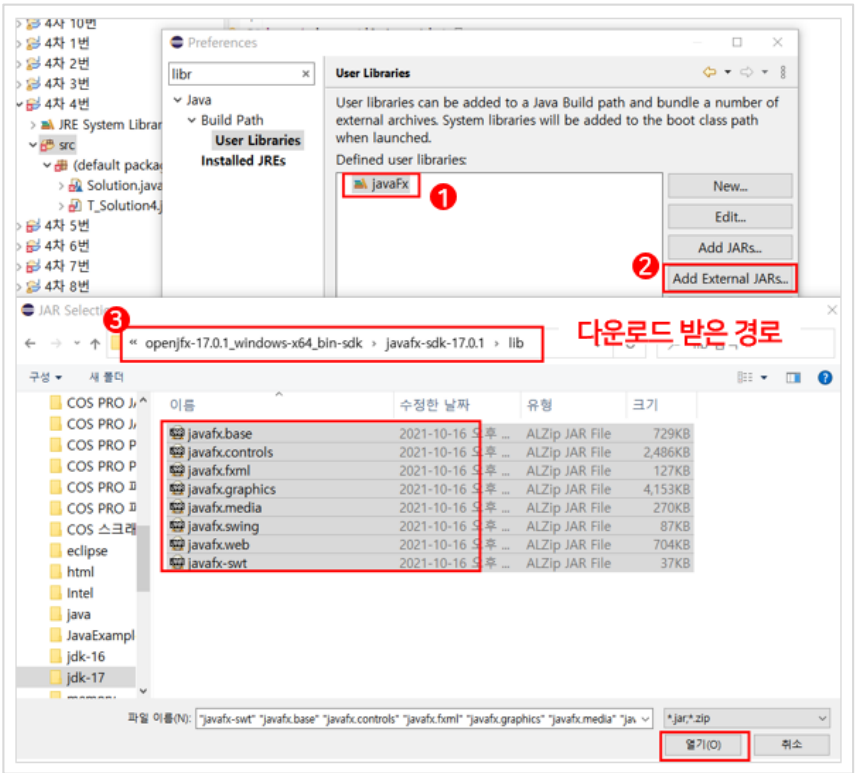


(3) 환경 설정

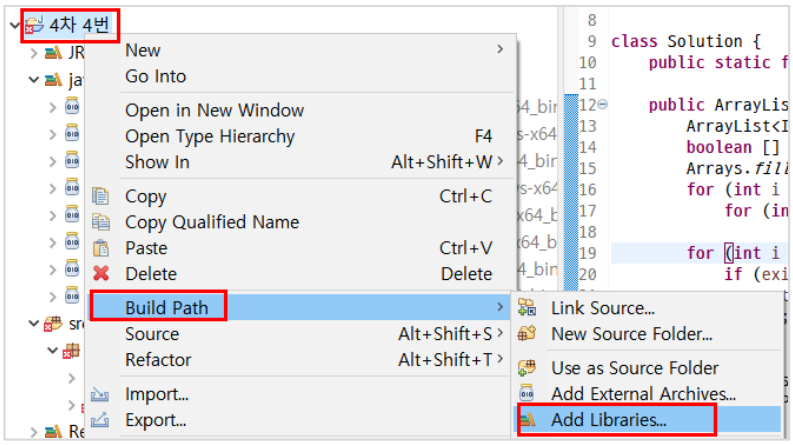
- 이클립스 [Window] - [Preferences] 에서 “libraries” 로 검색 후 [User Libraries]로 접근한 후 [New] 버튼 클릭
- “javaFx” 라고 적은 후 [OK] 버튼 클릭



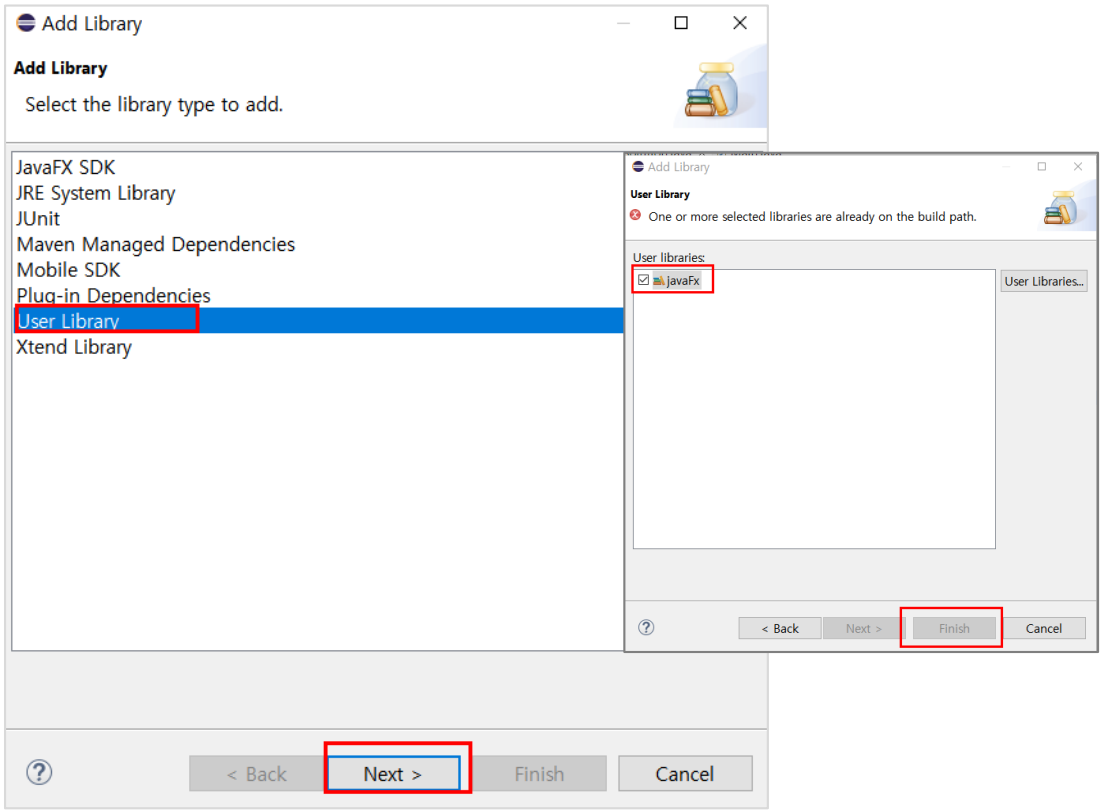
- 생성된 [javaFx]를 클릭한 후 [Add External JARs..] 버튼을 클릭하여 설치한 폴더의 lib 경로를 선택하여 모든 파일을 아래와 같이 선택



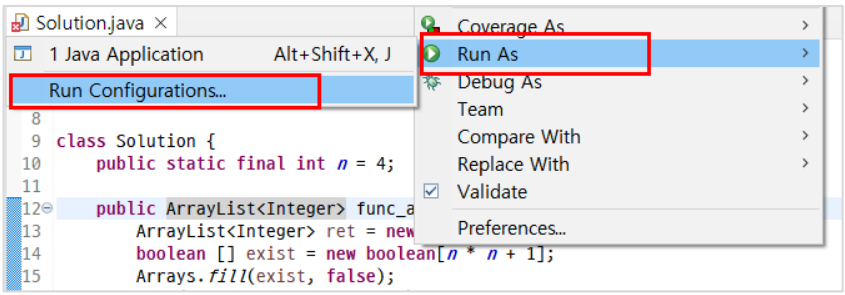
- 프로젝트명을 오른쪽 클릭하여 [Build Path] -> [Add Libraries]



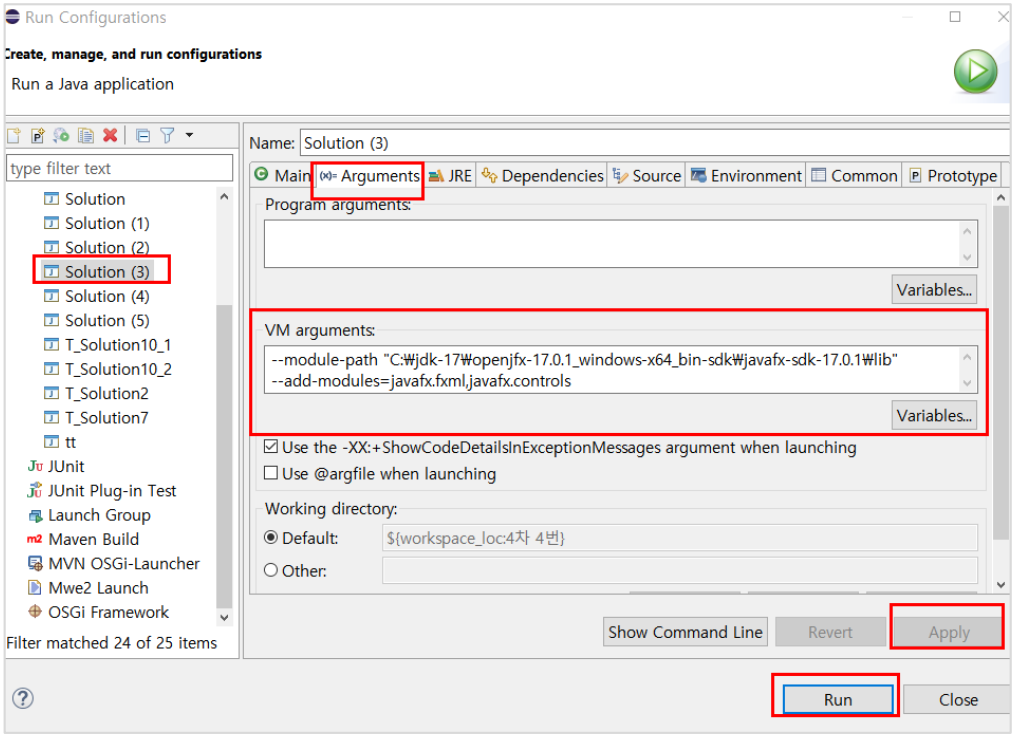
- [User Library] - [javaFx] 선택



- 코드에서 오른쪽 마우스 [Run Configurations..] - [Run As]



- 해당하는 코드에서 [Arguments] 탭의 [VM arguments]란에 띄어쓰기, 엔터 등 주의하여 아래와 같이 작성

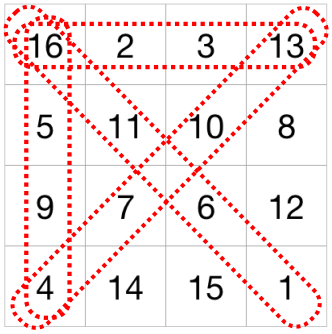



```
--module-path "본인의 설치 경로"
--add-modules=javafx.fxml,javafx.controls
```

5) 마방진(Magic Square)

① 마방진 소개

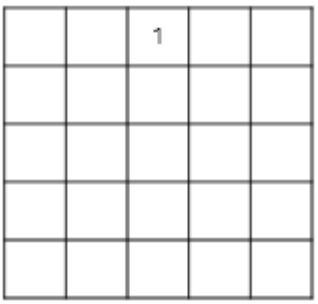
- 가로, 세로, 대각선 방향의 수를 더한 합이 모두 같은 정사각형 행렬
- 1 부터 정사각형 넓이까지 모든 수가 중복되지 않고 한 번씩 사용되어야 함
- 오른쪽 그림은 가로, 세로, 대각선 수의 합이 34 인 4x4 마방진



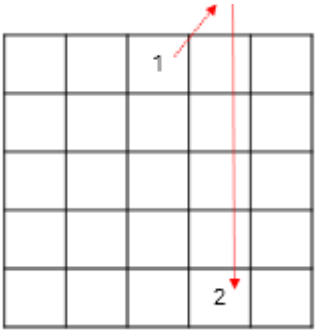
② 참고 : 홀수 N x N 마방진 만들기

❖ 알고리즘

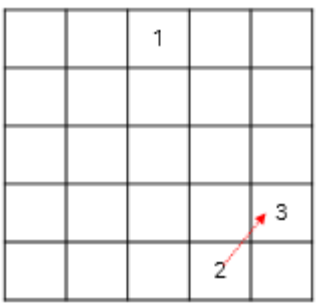
- 첫 번째 수 1 을 첫 번째 행 가운데에 할당 ex) N = 5 인 경우



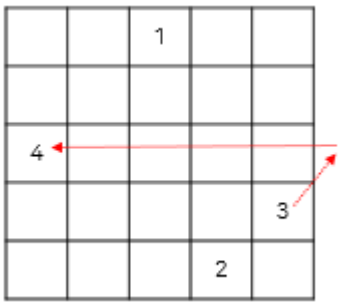
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 행이 0 보다 작으면 행의 위치를 마지막 행으로 변경



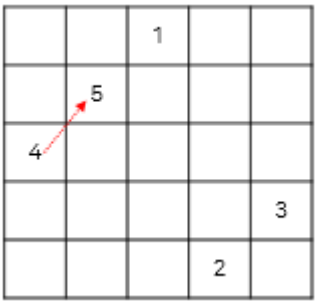
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당



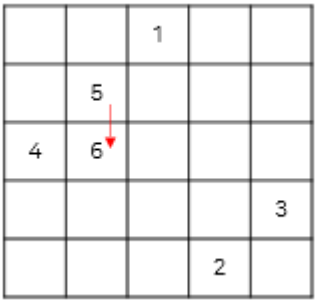
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 열이 마지막 열을 벗어나면 열의 위치를 첫 번째 열로 변경



- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당

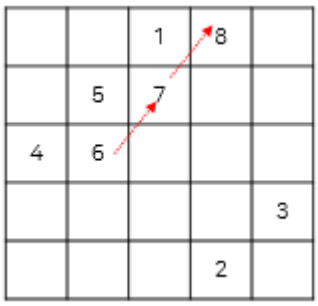


- 이전에 할당한 수가 N 의 배수이면 새로 할당할 수의 위치는 이전 행의 바로 아래 행으로 지정

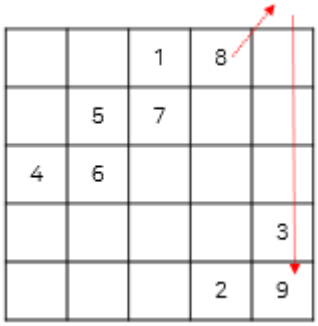


ex) N=5 일 때 이전에 할당한 수 5 가 5 의 배수이므로 5 의 바로 아래 행에 6 을 할당

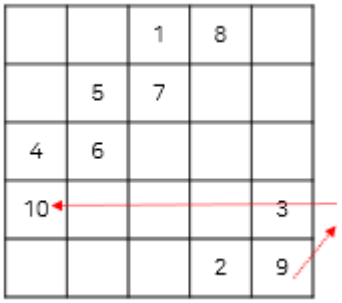
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당



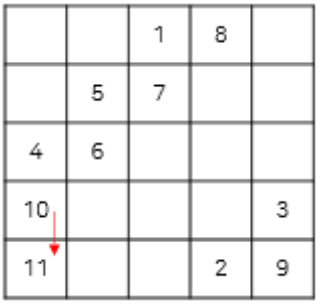
- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 행이 0 보다 작으면 행의 위치를 마지막 행으로 변경



- 이전에 할당한 수가 N 의 배수가 아니면 오른쪽 대각선 위 방향 칸에 새로운 수를 할당. 단, 새로운 위치의 열이 마지막 열을 벗어나면 열의 위치를 첫 번째 열로 변경



- 이전에 할당한 수가 N 의 배수이면 새로 할당할 수의 위치는 이전 행의 바로 아래 행으로 지정



ex) N=5 인 경우 이전 수 10 이 5 의 배수이므로 10 의 바로 아래 행에 11 을 할당

- 위의 과정을 빈 칸을 모두 채울 때까지 반복하여 N = 5 인 5 x 5 마방진을 오른쪽 그림과 같이 완성

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

6) 정답

- func_a( ) ~ func_c( ) 코드 정리

```
public static final int n = 4;

public ArrayList<Integer> func_a(int[][] matrix) {
    ArrayList<Integer> ret = new ArrayList<Integer>();
① boolean [] exist = new boolean[n * n + 1];
    Arrays.fill(exist, false);
② for (int i = 0; i < n; i ++)
        for (int j = 0; j < n; j++)
                exist[matrix[i][j]] = true;
③ for (int i = 1; i <= n * n; i++)
        if (exist[i] == false)
            ret.add(i);
    return ret;
}
```

❖ func_a( ) 함수는 4 x 4 형태의 2 차원 배열로 전달 받은 매개변수 matrix 에서 1 부터 16 까지의 수 중 존재하지 않는 수를 찾아 return

①. false 를 17(n*n+1)개 갖는 exist 배열 생성

② exist 의 인덱스가 matrix 배열의 항목 값과 같은 항목의 값을 true 로 변경
③ exist 의 항목 값이 false 로 남아 있는 인덱스를 모두 찾아 ret 에 항목으로 추가한 후 return

```
public ArrayList<Pair<Integer, Integer> > func_b(int[][] matrix) {
    ArrayList<Pair<Integer, Integer> > ret = new ArrayList<Pair<Integer, Integer> >();
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (matrix[i][j] == 0)
                ret.add( new Pair<Integer, Integer>(i, j) );
    return ret;
}
```

❖ func_b( ) 함수는 4 x 4 형태의 2 차원 배열로 전달 받은 매개 변수 matrix 에서 항목 값이 0 인 위치의 인덱스를 찾아 return
중첩 for 문을 이용하여 matrix 의 항목 값이 0 인 것을 찾아 해당 행과 열의 인덱스를 Pair 로 구성된 vector 배열 ret 에 추가

```
public boolean func_c(int[][] matrix) {
①   int sum = 0;
    for (int i = 1; i <= n * n; i++)
        sum += i;
    sum = sum / n;
    for (int i = 0; i < n; i++) {
        int rowSum = 0;
②       int colSum = 0;
        for (int j = 0; j < n; j++) {
            rowSum += matrix[i][j];
            colSum += matrix[j][i];
        }
        if (rowSum != sum || colSum != sum)
            return false;
    }
    int mainDiagonalSum = 0;
③   int skewDiagonalSum = 0;
    for (int i = 0; i < n; i++) {
        mainDiagonalSum += matrix[i][i];
        skewDiagonalSum += matrix[i][n-1-i];
    }
    if (mainDiagonalSum != sum || skewDiagonalSum != sum)
        return false;
    return true;
}
```

❖ func_c( ) 함수는 4 x 4 배열로 전달 받은 매개 변수 matrix 에서 같은 행에 있는 항목들의 합계, 같은 열에 있는 항목들의 합계, 대각선 방향에 있는 항목들의 합계가 동일한 지 확인하여 그 결과를 return

① sum 변수는 1 부터 16 까지의 합을 4 로 나눈 몫으로 초기화

② 중첩 for 문을 이용하여 matrix 의 각 행에 있는 항목 값의 합계와 각 열에 있는 항목 값의 합계를 구한 후 그 합계가 sum 변수의 값과 같지 않으면 false 를 return

③ for 문을 이용하여 matrix 의 2 개의 대각선 방향에 있는 항목 값의 합계를 구한 후 그 합계 값이 sum 변수의 값과 같지 않으면 false 를 return

- solution() 정리

```
public int[] solution(int[][] matrix) {
    int[] answer = new int[6];
    int ansIdx = 0;
① ArrayList<Pair<Integer, Integer> > coords = func_b(matrix);
② ArrayList<Integer> nums = func_a(matrix);

③ matrix[coords.get(0).getKey()][coords.get(0).getValue()] = nums.get(0);
  matrix[coords.get(1).getKey()][coords.get(1).getValue()] = nums.get(1);
④ if (func_c(matrix)) {
        for (int i = 0; i <= 1; i++) {
            answer[ansIdx++] = coords.get(i).getKey() + 1;
            answer[ansIdx++] = coords.get(i).getValue() + 1;
            answer[ansIdx++] = nums.get(i);
        }
    }
    else {
        matrix[coords.get(0).getKey()][coords.get(0).getValue()] = nums.get(1);
⑤     matrix[coords.get(1).getKey()][coords.get(1).getValue()] = nums.get(0);
        for (int i = 0; i <= 1; i++) {
            answer[ansIdx++] = coords.get(1-i).getKey() + 1;
            answer[ansIdx++] = coords.get(1-i).getValue() + 1;
            answer[ansIdx++] = nums.get(i);
        }
    }
    return answer;
}
```

①. func_b( )를 이용하여 matrix 배열에서 항목 값이 0 인 인덱스를 찾아 행과 열의 정보를 갖는 Pair 클래스로 구성된 정보를 coords 배열에 저장(coords 저장 예 : { {1, 2}, {3, 4} })

②. func_a( )를 이용하여 matrix 에서 1 부터 16 까지의 수 중에서 존재하지 않는 두 개의 수를 찾아 nums 에 저장

③. nums[0], nums[1] 값을 coords[0] 번째 항목이 가리키는 matrix 의 인덱스와 coords[1] 번째 항목이 가리키는 matrix 인덱스에 차례대로 할당

④. func_c( )를 이용하여 matrix 가 마방진이 되는 지 if 문을 통해 확인

- 현 상태에서 matrix 가 마방진이라면 for 문을 이용하여 coords[0]에 있는 (행 인덱스+1) 값과 (열 인덱스 +1)값을 answer 에 추가한 뒤 num[0]을 추가
- for 문에 의한 그 다음 반복에서는 coords[1]에 있는 (행 인덱스+1)값과 (열 인덱스+1) 값을 answer 에 추가한 뒤 num[1]을 추가

⑤. if 문에 의한 조건을 만족하지 못하면 nums[1], nums[0] 값을 coords[0] 번째 항목이 가리키는 matrix 인덱스와 coords[1] 번째 항목이 가리키는 matrix 인덱스에 차례대로 할당

- for 문을 이용하여 coords[1]에 있는 (행 번호+1)값과 (열 번호+1)값을 answer 에 추가한 뒤 num[0]을 answer 에 추가
- for 문에 의한 그 다음 반복에서는 coords[0]에 (행 번호+1)값과 (열 번호+1)값을 answer 에 추가한 뒤 num[1]을 answer 에 추가

5. 문제 5

1) 문제 코드

```java
/*=======================================
   4차 5번   4차 1급 5_initial_code.java
 =======================================*/

class Solution {
    public String reverse(String number) {
        String reverseNumber = "";
        for(int i = number.length()-1; i >= 0; i--)
            @@@;
        return reverseNumber;
    }

    public String solution(int n) {
        String answer = "";
        for(int i = 0; i < n; i++) {
            answer += Integer.toString(@@@);
            answer = reverse(answer);
        }
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int n = 5;
        String ret = sol.solution(n);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 수행하기 위해 코드의 빈 곳을 완성하는 문제
- 1 부터 9 까지의 숫자를 차례대로 문자열 뒤에 붙인 후 문자열 전체의 앞뒤를 뒤집어 나타내는 작업을 n 번째까지 하여 그 결과를 출력하는 문제

3) 예시 설명 : n = 5 인 경우

첫 번째 수	1	
두 번째 수	21	1 뒤에 2 를 붙인 후 순서를 뒤집음.
세 번째 수	312	1 뒤에 3 을 붙인 후 순서를 뒤집음.
네 번째 수	4213	2 뒤에 4 를 붙인 후 순서를 뒤집음.
다섯 번째 수	53124	3 뒤에 5 를 붙인 후 순서를 뒤집음.

4) 정답

```
public String reverse(String number) {
    String reverseNumber = "";
    for(int i = number.length()-1; i >= 0; i--)
    ② reverseNumber += number.charAt(i);
    return reverseNumber;
}

public String solution(int n) {
    String answer = "";
    for(int i = 0; i < n; i++) {
     ① answer += Integer.toString(i%9 + 1);
     ② answer = reverse(answer);
    }
    return answer;
}
```

①. n 번 반복하는 동안 1 부터 9 까지의 수를 반복해서 붙이기 위해 i 를 9 로 나눈 나머지에 1 을 더한 결과를 문자열로 변환하여 answer 가 갖고 있는 문자열의 끝에 붙임

- i 를 9 로 나눈 나머지는 0 부터 8 까지 나올 수 있는데, 이렇게 계산된 나머지에 1 을 더하면 1 부터 9 까지의 결과를 산출할 수 있음

②. 새로 생성한 answer 를 문자열의 마지막을 가리키는 인덱스 -1 부터 문자열의 처음까지 -1 만큼 줄이면서 answer 가 갖고 있는 문자열의 앞뒤 순서를 뒤집을 수 있음(reverse 함수에서는 number 매개변수로 받음)

6. 문제 6

1) 문제 코드

```java
/*=======================================
   4차 6번   4차 1급 6_initial_code.java
 =======================================*/

import java.util.Arrays;

class Solution {
    public int power(int base, int exponent) {
        int val = 1;
        for (int i = 0; i < exponent; i++)
            val *= base;
        return val;
    }
    public int[] solution(int k) {
        int range = power(10, k);
        int[] answer = new int[range];
        int count = 0;
        for (int i = range / 10; i < range; i++) {
            int current = i;
            int calculated = 0;
            while (current != 0) {
                @@@;
                @@@;
            }
            if (calculated == i)
                answer[count++] = i;
        }

        int[] ret = new int[count];
        for (int i = 0; i < count; i++)
            ret[i] = answer[i];
        return ret;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int k = 3;
        int[] ret = sol.solution(k);

        // 실행] 버튼을 누르면 출력값을 볼 수 있습니다.
        System.out.printf("solution 메소드의 반환 값은 ");
        System.out.printf(Arrays.toString(ret));
        System.out.printf(" 입니다.\n");
    }
}
```

2) 문제 개요

- 제시된 과제를 바르게 수행하기 위하여 비어 있는 코드를 완성하는 문제.
- k 자리 수가 주어졌을 때 그 수의 각 자릿수를 k 제곱한 합이 원래 수와 같으면 그 수를 자아도취 수라 정의 (예시 : $153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$)
- k 자리 자아도취 수를 모두 찾아 배열에 저장하는 프로그램에서 빈 곳을 채우는 문제.

3) 정답

❖ power() 함수 정리

```
public int power(int base, int exponent) {
    int val = 1;
    for (int i = 0; i < exponent; i++)
        val *= base;
    return val;
}
```

- for 문을 이용하여 매개변수 base 를 exponent 만큼 거듭제곱한 결과를 val 에 저장

※ for 문을 사용하지 않고 Math 의 pow 를 사용해도 동일

```
static double    pow(double a, double b)
```

❖ solution( ) 메소드 정리

```
public int[] solution(int k) {
① int range = power(10, k);
   int[] answer = new int[range];
   int count = 0;
② for (int i = range / 10; i < range; i++) {
       int current = i;
       int calculated = 0;
③      while (current != 0) {
           calculated += power(current % 10, k);
           current = current / 10;
       }
④      if (i == calculated) {
           answer[count] = i;
           count++;
       }
   }
⑤ int[] ret = new int[count];
   for (int i = 0; i < count; i++) {
       ret[i] = answer[i];
   }
   return ret;
}
```

① 위에서 정의한 power( ) 함수를 사용하여 10 을 k 제곱한 결과를 range 변수에 저장

② for 문을 이용하여 (range / 10) 부터 (range - 1)를 i 로 하나씩 받아 와서 current 변수를 i 로 초기화

- k=3 이라면 10^3 / 10 = 100 부터 10^3 - 1 = 999 까지의 수 즉 세 자리 수를 i 로 전달.

i 로 받아온 수의 각 자리 숫자에 대해 k 제곱한 값을 합산하기 위해 사용하는 변수 calculated 를 0 으로 초기화

③ current 를 10 으로 나눈 나머지를 이용하여 일의 자리부터 순서대로 current 의 각 자리 숫자를 가져와 k 제곱한 것을 calculated 에 합산

current 를 10 으로 나눈 몫을 current 에 다시 저장하여 앞에서 k 제곱에 사용한 자리 숫자를 current 에서 삭제

④ for 문을 이용해 받아온 k 자리 수 i 와 i 의 각 자리 숫자를 k 제곱한 것을 더한 값이 같으면 answer 배열에 추가

⑤ answer 배열의 내용을 ret 배열에 복사

7. 문제 7

1) 문제 코드

```
/*=======================================
   4차 7번   4차 1급 7_initial_code.java
 =======================================*/

import java.util.Arrays;

class Solution {
    class Unit {
        public int HP;
        public Unit() {
            this.HP = 1000;
        }
        public void underAttack(int damage) { }
    }

    class Monster @@@ {
        public int attackPoint;
        public Monster(int attackPoint) {
            this.attackPoint = attackPoint;
        }
        @@@ {
            this.HP -= damage;
        }
        @@@ {
            return attackPoint;
        }
    }

    class Warrior @@@ {
        public int attackPoint;
        public Warrior(int attackPoint) {
            this.attackPoint = attackPoint;
        }
        @@@ {
            this.HP -= damage;
        }
        @@@ {
            return attackPoint;
        }
    }
```

```
    class Healer @@@ {
        public int healingPoint;
        public Healer(int healingPoint) {
            this.healingPoint = healingPoint;
        }
        @@@ {
            this.HP -= damage;
        }
        @@@ {
            unit.HP += healingPoint;
        }
    }
    public int[] solution(int monsterAttackPoint, int warriorAttackPoint, int healingPoint) {
        Monster monster = new Monster(monsterAttackPoint);
        Warrior warrior = new Warrior(warriorAttackPoint);
        Healer healer = new Healer(healingPoint);

        //전사가 몬스터를 한 번 공격
        monster.underAttack(warrior.attack());
        //몬스터가 전사를 한 번 공격
        warrior.underAttack(monster.attack());
        //몬스터가 힐러를 한 번 공격
        healer.underAttack(monster.attack());
        //힐러가 전사의 체력을 한 번 회복
        healer.healing(warrior);
        //힐러가 몬스터의 체력을 한 번 회복
        healer.healing(monster);

        int[] answer = {monster.HP, warrior.HP, healer.HP};
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int monsterAttackPoint = 100;
        int warriorAttackPoint = 90;
        int healingPoint = 30;
        int[] ret = sol.solution(monsterAttackPoint, warriorAttackPoint, healingPoint);

        // 실행] 버튼을 누르면 출력값을 볼 수 있습니다.
        System.out.printf("solution 메소드의 반환 값은 ");
        System.out.printf(Arrays.toString(ret));
        System.out.printf(" 입니다.\n");
    }
}
```

2) 문제 개요

- Unit 클래스를 상속받는 Monster, Warrior, Healer 클래스를 정의하는 문제

3) 정답

```
/*==================================
   4차 1급 7_solution_code.java
   ybmit.com 사이트의 샘플 파일에는 정답이 제시되어 있지 않습니다.
   아래와 같은 정답을 제안 합니다. 더 좋은 알고리즘으로 구현해 보시기 바랍니다.
==================================*/
```

```java
class Unit {
     public int HP;
     public Unit() {
         this.HP = 1000;
     }
     public void underAttack(int damage) { }
 }
```

①
```java
class Warrior extends Unit{
     public int attackPoint;
     public Warrior(int attackPoint) {
         this.attackPoint = attackPoint;
     }
     public  void underAttack(int damage) {
         this.HP -= damage;
     }
     int attack() {
         return this.attackPoint;
     }
 }
```

②
```java
class Monster extends Unit {
    public int attackPoint;
    public Monster(int attackPoint) {
        this.attackPoint = attackPoint;
    }
    public void underAttack(int damage) {
        this.HP -= damage;
    }
    int attack() {
        return this.attackPoint;
    }
}
```

③
```java
class Healer extends Unit{
    public int healingPoint;
    public Healer(int healingPoint) {
        this.healingPoint = healingPoint;
    }
    public  void underAttack(int damage) {
        this.HP -= damage;
    }
    void healing(Unit unit) {
        unit.HP += healingPoint;
    }
}
```

① Unit 클래스를 상속받은 Monster 클래스를 정의
- 생성자에서 기초 클래스 Unit 의 생성자를 실행하여 HP 멤버 변수의 값을 초기화하고, 멤버 변수 attackPoint 를 매개 변수로 초기화
- Unit 클래스에 있는 underAttack( ) 메소드 오버라이딩 : damage 만큼 멤버 변수 HP 를 감소
- attack( ) 메소드를 정의 : 멤버 변수인 attackPoint 를 return, 문제에서는 attackPoint 를 리턴하고 있으나, this.attackPoint 라고 적으면 멤버 변수임을 더 명확히 할 수 있음

② Unit 클래스를 상속받은 Warrior 클래스를 정의
- 생성자에서 기초 클래스 Unit 의 생성자를 실행하여 HP 멤버 변수의 값을 초기화하고, 멤버 변수 attackPoint 를 매개 변수로 초기화
- Unit 클래스에 있는 underAttack( ) 메소드 오버라이딩 : damage 만큼 멤버 변수 HP 를 감소
- attack( ) 메소드를 정의 : 멤버 변수인 attackPoint 를 return, 문제에서는 attackPoint 를 리턴하고 있으나, this.attackPoint 라고 적으면 멤버 변수임을 더 명확히 할 수 있음

③ Unit 클래스를 상속받은 Healer 클래스를 정의
- Unit 클래스에 있는 underAttack( ) 메소드 오버라이딩 : damage 만큼 멤버 변수 HP 를 감소
- 매개 변수로 unit 를 갖는 healing( ) 메소드를 정의 : 매개변수로 받은 객체 unit 의 멤버 변수 HP 를 healingPoint 만큼 증가. 객체의 멤버 변수가 아닌 매개 변수로 받은 객체의 멤버 변수를 수정하므로 매개 변수 unit 을 참조 변수로 선언

8. 문제 8

1) 문제 코드

```
/*==========================================
   4차 8번   4차 1급 8_initial_code.java
 ==========================================*/

// 다음과 같이 import를 사용할 수 있습니다.
import java.util.*;

class Solution {
    public int solution(int[] card, int n) {
        // 여기에 코드를 작성해주세요.
        int answer = 0;
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int card1[] = {1, 2, 1, 3};
        int n1 = 1312;
        int ret1 = sol.solution(card1, n1);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");

        int card2[] = {1, 1, 1, 2};
        int n2 = 1122;
        int ret2 = sol.solution(card2, n2);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제
- 전달된 숫자 카드를 모두 사용하여 만들 수 있는 수를 배열로 저장한 후 solution( ) 메소드의 매개변수 n에 있는 수가 배열에 있는 수 중에서 몇 번째로 작은 수인지 return 하는 solution 메소드를 작성

3) 정답

```
ArrayList<Integer> num_list = new ArrayList<Integer>();

public int[] func_a(int[] card) {
  ① int card_count[] = new int [10];
  ② for (int i = 0; i < card.length; i++) {
        card_count[card[i]]++;
    }
    return card_count;
}
```

❖ func_a( ) 는 card 배열에 있는 숫자를 읽어 각 숫자의 개수를 card_count 에 집계하여 리턴 하는 함수

①. 10 개의 0 을 항목으로 갖는 card_count 배열을 생성하고 0 으로 초기화

②. for 문을 이용하여 card 에 있는 숫자를 인덱스로 하여 card_count 배열의 값을 1 만큼 증가
ex) card = [1, 2, 1, 3] 인 경우 card_count 에 집계되는 값

인덱스	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
항목	0	2	1	1	0	0	0	0	0	0

➔ card 에 1 이 2 개 존재하므로 card_count 의 1 번 인덱스 항목은 2
2 가 1 개 존재하므로 card_count 의 2 번 인덱스 항목은 1
3 이 1 개 존재하므로 card_count 의 3 번 인덱스 항목은 1

```
public void func_b(int level, int max_level, int num, int[] current_count, int[] max_count) {
    if (level == max_level) {                                                    ①
        num_list.add(num);
        return;
    }
    for (int i = 1; i <= 9; i++) {                                               ②
        if (current_count[i] < max_count[i]) {
            current_count[i] += 1;
            func_b(level + 1, max_level, num * 10 + i, current_count, max_count);
            current_count[i] -= 1;
        }
    }
}
```

❖ func_b( )는 max_count 에 집계되어 있는 숫자 카드별 총 개수들을 바탕으로 하여 숫자 카드 모두를 사용한 수를 생성한 후 전역 변수 num_list 에 생성한 수를 추가

매개변수 level	현재 생성한 수 num 을 만들기 위해 사용한 총 숫자 카드 개수
매개변수 max_level	숫자 카드 총 개수
매개변수 num	현재 생성한 수
매개변수 current_count	num 을 만들기 위해 각 숫자 카드별 사용한 개수
매개변수 max_count	각 숫자 카드별 총 개수

①. 현재 생성한 num 을 만들기 위해 사용한 총 카드 개수(level)가 숫자 카드 총 개수(max_level)와 동일하면 전역 변수로 선언된 num_list 에 num 을 추가

②. for 문을 이용하여 숫자 카드 1 부터 숫자 카드 9 까지의 숫자를 i 로 받음

- 현재 생성한 수(num)를 만들기 위해 숫자 카드 i 를 사용한 개수(current_count[i])가 숫자 카드 i 의 총 개수보다 작은 지 확인
- 숫자 카드 i 를 사용한 개수를 1 만큼 증가
- 수를 만들기 위해 재귀 함수 호출 : 숫자 카드 i 를 추가하여 새로운 수를 생성해서 호출함
  - 첫 번째 인수 : 수를 만들기 위해 사용한 전체 카드 개수 (level) + 1
  - 두 번째 인수 : 숫자 카드 총 개수 (max_level)
  - 세 번째 인수 : 새로 생성한 수 = 현재 생성한 수 (num) * 10 + i
  - 네 번째 인수 : 새로 생성한 수를 만들기 위해 각 숫자 카드별 사용한 개수 (current_count)
  - 다섯 번째 인수 : 각 숫자 카드별 총 개수 (max_count)

- 숫자 카드 i 를 사용하여 재귀 호출한 함수가 종료되면 숫자 카드 i 를 사용한 개수(current_count[i])를 다시 1 만큼 감소

```
public int func_c(ArrayList<Integer> list, int n) {
①   for (int i = 0; i < list.size(); i++) {
        if (n == list.get(i))
            return i + 1;
    }
②   return -1;
}
```

❖ func_c( )는 매개변수로 받은 list 배열에서 n 을 항목 값으로 갖는 인덱스를 찾아 리턴 하는 함수

①. n 이 list 배열의 항목 값으로 존재하는지 for 반복문에서 비교하여 찾고, 찾은 인덱스에 + 1 을 하여 return (순서는 첫 번째부터 시작하지만 인덱스는 0 부터 시작하기 때문)

②. n 이 list 배열에 존재하지 않으면 -1 을 리턴

```
public int solution(int[] card, int n) {
③   int[] card_count = func_a(card);
④   func_b(0, card.length, 0, new int[10], card_count);
⑤   int answer = func_c(num_list, n);
    return answer;
}
```

❖ solution( )은 card 와 n 을 매개변수로 받아 card 에 있는 숫자 카드를 이용해서 수를 만들고 n 이 생성한 수들 중에 몇 번째 수인지 리턴 하는 함수

①. func_a( )를 이용하여 card 에 있는 각 숫자 카드별 총 개수를 card_count 에 집계

②. func_b( )를 이용하여 card 에 있는 숫자 카드로 구성된 수를 만들어 num_list 에 저장

- 첫 번째 인수 : 0 - 처음 시작하기 때문에 사용한 숫자 카드가 없음
- 두 번째 인수 : len(card) - 숫자 카드 총 개수
- 세 번째 인수 : 0 - 처음 시작하기 때문에 생성한 수가 없음
- 네 번째 인수 : 10 개의 0 을 갖는 배열 - 생성한 수가 없기 때문에 숫자 카드 1 부터 9 까지 사용한 카드가 없음
- 다섯 번째 인수 : card_count - 각 숫자 카드별 총 개수

③. func_c( ) 를 이용하여 매개변수 n 의 값이 숫자 카드로 만든 수 중 몇 번째인지 확인

4) 정답 코드의 func_b( ) 함수의 실행 process

: card = [1, 2, 1, 3] 를 사용하여 수를 만드는 경우

```java
public void func_b(int level, int max_level, int num, int[] current_count, int[] max_count) {
    if (level == max_level) {
        num_list.add(num);
        return;
    }
    for (int i = 1; i <= 9; i++) {
        if (current_count[i] < max_count[i]) {
            current_count[i] += 1;
            func_b(level + 1, max_level, num * 10 + i, current_count, max_count);
            current_count[i] -= 1;
        }
    }
}
```

① solution( ) 에서 func_b( ) 호출 : ,

func_b(0, 4, 0, current_count, max_count)

- 매개 변수에 저장하는 값

level	= 0 (사용한 숫자 카드 개수 : 아직 생성한 수가 없기 때문에 0)
max_level	= 4 (card 의 항목 개수)
num	= 0
current_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 0 0 0 0 0 0 0 0 0 (생성한 수가 없기 때문에 사용한 숫자 카드가 없음)
max_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 1 0 0 0 0 0 0 (card 의 각 항목값별 개수)

- func_b( ) 실행
  - level < max_level 이므로 for 문을 실행.
  - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
  - current_count[1] < max_count[1] 이므로
    - current_count[1] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.
    - 재귀 호출이 종료되면 숫자 카드 1 을 이용한 작업이 종료되었으므로 current_count[1]을 1 만큼 감소.

② func_b( ) 첫 번째 재귀 호출 :

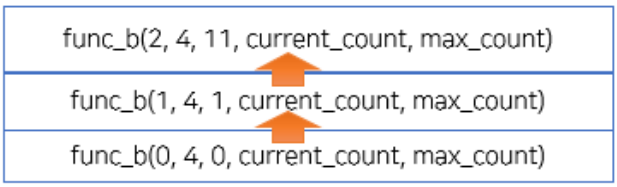
func_b(1, 4, 1, current_count, max_count)
func_b(0, 4, 0, current_count, max_count)

- 매개 변수에 저장하는 값

level	= 1 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 0 * 10 + 1 = 1 (숫자 카드 1 한 개를 사용하여 만든 수)
current_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 1 0 0 0 0 0 0 0 0 (num 을 만드는 데 사용한 숫자 카드 1 을 1 개 사용했기 때문)
max_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 1 0 0 0 0 0 0 (card 의 각 항목값별 개수)

- func_b( ) 실행
  - level < max_level 이므로 for 문을 실행
  - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음
  - current_count[1] < max_count[1] 이므로
    - current_count[1] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출
    - 재귀 호출이 종료되면 숫자 카드 1 을 이용한 작업이 종료되었으므로 current_count[1]을 1 만큼 감소

③ func_b( ) 두 번째 재귀 호출 :



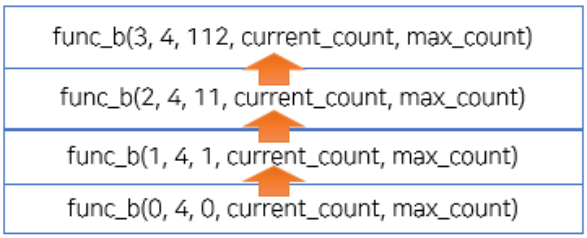
- 매개 변수에 저장되어 있는 값

level	= 2 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 1 * 10 + 1 = 11 (숫자 카드 1 두 개를 사용하여 만든 수)
current_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 0 0 0 0 0 0 0 0 (num 을 만드는 데 사용한 숫자 카드 1 을 2 개 사용했기 때문)
max_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 1 0 0 0 0 0 0 (card 의 각 항목값별 개수)

- func_b( ) 실행
  - level < max_level 이므로 for 문을 실행.
  - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
  - i 가 1 일 경우는 current_count[i] == max_count[i] 이지만, current_count[2] < max_count[2] 이므로
    - current_count[2] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출.

- 재귀 호출이 종료되면 숫자 카드 2 을 이용한 작업이 종료되었으므로 current_count[2] 을 1 만큼 감소.

④ func_b( ) 세 번째 재귀 호출 :



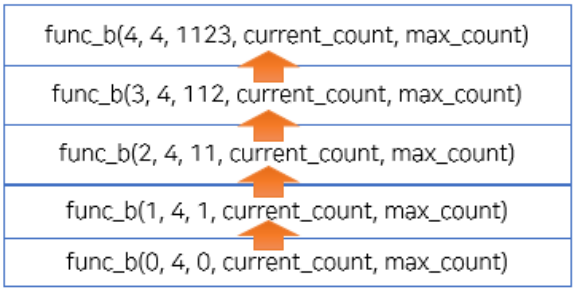
◆ 매개 변수에 저장되어 있는 값

level	= 3 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 11 * 10 + 2 = 112 (숫자 카드 1 두 개와 숫자 카드 2 한 개를 사용하여 만든 수)
current_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 0 0 0 0 0 0 0 (num 을 만드는데 사용한 숫자 카드 1 을 2 개, 숫자 카드 2 를 1 개 사용했기 때문)
max_count	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] 0 2 1 1 0 0 0 0 0 0 (card 의 각 항목값별 개수)

◆ func_b( ) 실행

- level < max_level 이므로 for 문을 실행.
- for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
- i 가 1, 2 일 경우는 current_count[i] == max_count[i] 이지만, current_count[3] < max_count[3] 이므로
  - current_count[3] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출
  - 재귀 호출이 종료되면 숫자 카드 3 을 이용한 작업이 종료되었으므로 current_count[3]을 1 만큼 감소

⑤ func_b( ) 네 번째 재귀 호출 :



- 매개 변수에 저장되어 있는 값

level	= 4 (num 을 만드는데 사용한 숫자 카드 개수)
max_level	= 4 (card 의 항목 개수)
num	= 112* 10 + 3 = 1123 (숫자 카드 1 두 개, 숫자 카드 2 한 개, 숫자 카드 3 한 개를 이용하여 만든 수)

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	1	0	0	0	0	0	0

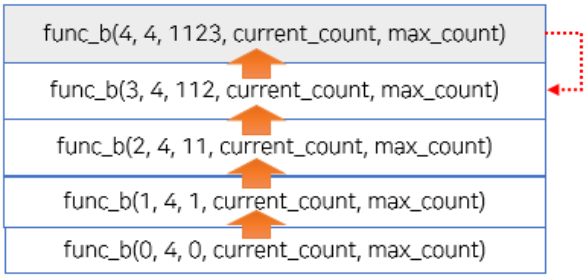
(num 을 만드는데 사용한 숫자 카드 1 을 2 개, 숫자 카드 2 를 1 개, 숫자 카드 3 을 1 개 사용했기 때문)

max_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	1	0	0	0	0	0	0

(card 의 각 항목값별 개수)

- func_b( ) 의 실행
  - level == max_level 이므로 num 을 num_list 에 추가하고 종료하여 func_b( )를 세 번째로 재귀 호출한 곳으로 돌아감



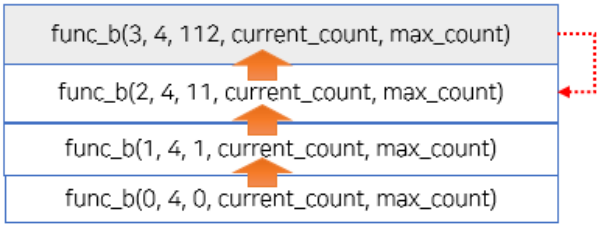
⑥ 네 번째 재귀 호출 → 세 번째 재귀 호출로 돌아간 후 func_b( ) 실행

- for 문에 의해 i=3 인 상태에서 실행되던 네 번째 재귀 호출을 종료한 후 current_count[3] 을 1 만큼 감소.

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	0	0	0	0	0	0	0

- i=4 부터 9 까지에 대해 current_count[i] == max_count[i] 이므로 나머지 반복에 대해서는 재귀 호출을 하지 않고 종료
- 세 번째 재귀 호출 종료 후 func_b( ) 를 두 번째로 재귀 호출한 곳으로 돌아감



⑦ 세 번째 재귀 호출 → 두 번째 재귀 호출로 돌아간 후 func_b( ) 실행

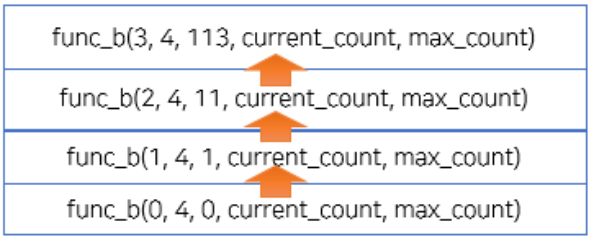
- for 문에 의해 i=2 인 상태에서 실행되던 세 번째 재귀 호출 종료 후 current_count[2] 을 1 만큼 감소

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	0	0	0	0	0	0	0	0

- i=3 부터 9 까지 반복문을 실행하는데 current_count[3] < max_count[3] 이므로
  - current_count[3] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출
  - 재귀 호출이 종료되면 숫자 카드 3 을 이용한 작업이 종료되었으므로 current_count[3]을 1 만큼 감소

⑧ func_b( ) 다섯 번째 재귀 호출 :



- 매개 변수에 저장되어 있는 값

level = 3 (num 을 만드는데 사용한 숫자 카드 개수)

max_level = 4 (card 의 항목 개수)

num = 11 * 10 + 3 = 113 (숫자 카드 1 두 개와 숫자 카드 3 한 개를 사용하여 만든 수)

current_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	0	1	0	0	0	0	0	0

(num 을 만드는데 사용한 숫자 카드 1 을 2 개, 숫자 카드 3 을 1 개 사용했기 때문)

max_count

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	2	1	1	0	0	0	0	0	0

(card 의 각 항목값별 개수)

- func_b( ) 실행
  - level < max_level 이므로 for 문을 실행.
  - for 문에 의해서 1 부터 9 까지의 숫자를 i 로 받음.
  - i 가 1 일 경우는 current_count[i] == max_count[i] 이지만, current_count[2] < max_count[2] 이므로
    - current_count[2] 을 1 만큼 증가시킨 후 func_b( )를 재귀 호출
    - 재귀 호출이 종료되면 숫자 카드 2 를 이용한 작업이 종료되었으므로 current_count[2]을 1 만큼 감소

➔ 위와 같은 과정을 [1, 1, 2, 3] 을 이용하여 만들 수 있는 4 자리수를 모두 생성할 때까지 반복

9. 문제 9

1) 문제 코드

```java
/*=======================================
   4차 9번   4차 1급 9_initial_code.java
 =======================================*/

// 다음과 같이 import를 사용할 수 있습니다.
import java.util.*;

class Solution {
    public String solution(int hour, int minute) {
        // 여기에 코드를 작성해주세요.
        String answer = "";
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int hour = 3;
        int minute = 0;
        String ret = sol.solution(hour, minute);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제
- 매개변수 hour 와 minute 를 이용하여 1 시간 동안 30 도, 1 분 동안 0.5 도 움직이는 시침과 1 시간 동안 360 도, 1 분 동안 6 도 움직이는 분침이 이루는 각도를 소수점 첫 번째 자리까지의 문자열로 return 하도록 코드를 작성하는 문제

3) 정답

시침	1 시간 동안 30 도 이동, 1 분 동안 30/60=0.5 도 이동
분침	1 분 동안 360/60 = 6 도 이동

➔ 시침이 이동하는 각도 = hour * 30 + minute * 0.5

➔ 분침이 이동하는 각도 = minute * 6

```
public String solution(int hour, int minute) {
      // 여기에 코드를 작성해주세요.
      String answer = "";
①     float h_hand = (float)360 / (60*12) * (hour*60 + minute);
      float m_hand = (float)360 / 60 * minute;

      float arc;

②     if (h_hand > m_hand)
          arc = h_hand - m_hand;
      else
          arc = m_hand - h_hand;

③     if (arc > 180)
          arc = 360 -arc;

      answer = Float.toString(arc);
      return answer;
  }
```

① 매개변수 hour 와 minute 를 이용하여 시침과 분침이 이동하는 각도를 각각 계산
② 두 바늘이 이루는 각도를 계산
③ 계산한 각도 값이 180 보다 크면 360 에서 각도 값을 빼서 각도 값으로 다시 할당하고 실수를 문자열로 변환하여 리턴

10. 문제 10

1) 문제 코드

```java
/*=======================================
   4차 10번   4차 1급 10_initial_code.java
 =======================================*/

// 다음과 같이 import를 사용할 수 있습니다.
import java.util.*;

class Solution {
    public int solution(int a, int b) {
        // 여기에 코드를 작성해주세요.
        int answer = 0;
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args){
        Solution sol = new Solution();
        int a = 6;
        int b = 30;
        int ret = sol.solution(a, b);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제
- 매개변수로 전달된 a, b 사이에 있는 수들 중에서 소수의 제곱수와 소수의 세제곱수를 찾아 그 개수를 return 하도록 코드를 작성하는 문제

3) 소수 찾기

① 소수의 정의

- 1 과 자기 자신만으로 나누어 떨어지는 수

② 소수를 판별하는 방법 1

- 어떤 수를 2 부터 시작하여 자기 자신보다 하나 작은 수까지의 수로 나누어 떨어지는 수가 없으면 소수로 판별.
- 만일 나누어 떨어지는 수가 존재하면 소수가 아님.
- 2 부터 k 까지의 수 중 소수를 찾는 JAVA 프로그램

```
int k=100000;
int arr[] = new int[k];
boolean prime;
int idx =0;


for(int n=2; n<=k; n++) {
    prime=true;
    for(int i=2;i<n;i++) {
        if(n%i ==0) {
            prime=false;
            break;
        }
    }
    if(prime==true)
        arr[idx++]=n;
}
```

- for 문을 이용하여 2 부터 k 까지의 수를 n 으로 받음
- 소수를 판별하기 위해 2 부터 n-1 까지의 수를 i 로 받음
- n 을 i 로 나눈 나머지가 0 이면 i 는 n 의 약수이고 n 은 소수가 아니므로, break 를 사용하여 소수 판별수 i 로 나누는 작업 종료
- break 를 실행하지 않고 안쪽 for 문이 종료된 경우 prime 이 true 이므로 n 을 arr 에 추가

➔ 숫자 k 가 커질수록 실행 시간이 커짐

③ 에라토스테네스의 체

- 입자의 크기가 다른 가루를 체로 거르는 것처럼 에라토스테네스의 체를 사용하여 특정 자연수 이하의 소수만 골라내는 방법.
- 원하는 범위만큼 배열을 생성하여 초기값을 T 로 지정.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	...

- 항목값이 T 인 최소 인덱스 2 에 대해서 인덱스가 2 보다 큰 2 의 배수인 항목들의 값을 모두 F 로 변경(소수 2 를 찾은 후 2 의 배수는 소수가 아닌 것을 나타냄)

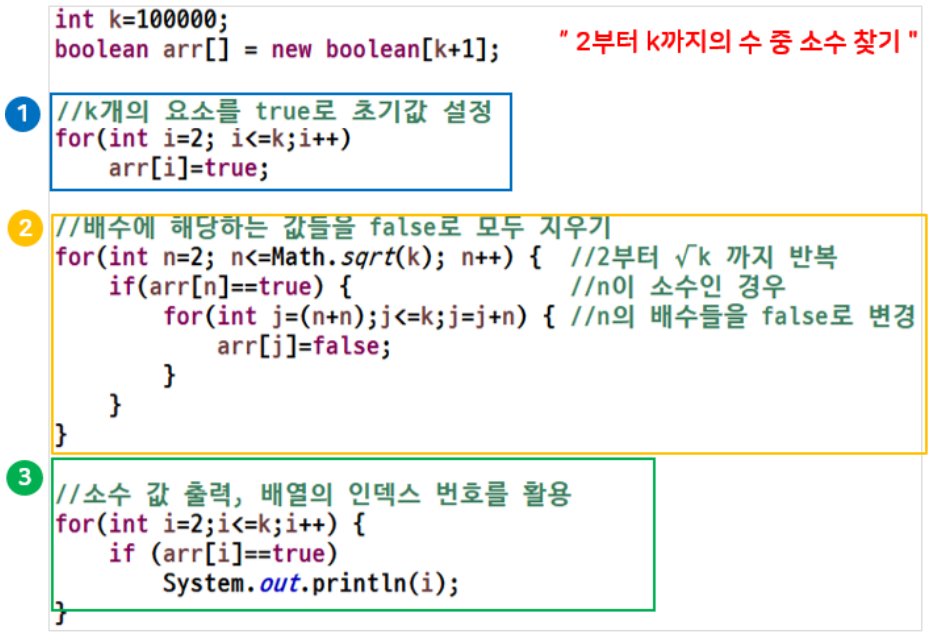
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	...

- 항목값이 T 인 최소 인덱스 3 에 대해서 인덱스가 3 보다 큰 3 의 배수인 항목들의 값을 모두 F 로 변경(소수 3 을 찾은 후 3 의 배수는 소수가 아닌 것을 나타냄)

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
	T	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	...

- 항목값이 T 로 남아있는 인덱스 중 가장 작은 수가 원하는 범위 n 의 제곱근을 넘을 때까지 항목값이 T 인 최소 인덱스를 찾아서 위와 같은 과정을 반복
  (∵ n 의 최대 약수가 $\sqrt{n}$)

- JAVA 코드로 구현




```java
int k=100000;
boolean arr[] = new boolean[k+1];

//k개의 요소를 true로 초기값 설정
for(int i=2; i<=k;i++)
    arr[i]=true;

//배수에 해당하는 값들을 false로 모두 지우기
for(int n=2; n<=Math.sqrt(k); n++) {  //2부터 √k 까지 반복
    if(arr[n]==true) {                //n이 소수인 경우
        for(int j=(n+n);j<=k;j=j+n) { //n의 배수들을 false로 변경
            arr[j]=false;
        }
    }
}

//소수 값 출력, 배열의 인덱스 번호를 활용
for(int i=2;i<=k;i++) {
    if (arr[i]==true)
        System.out.println(i);
}
```

①. k 개의 요소에 true 로 초기값 설정

②. for 문을 이용하여 2 부터 $\sqrt{n}$ 의 정수부까지의 수를 가져와 i 로 받음

- i 보다 큰 i 의 배수를 인덱스로 갖는 항목을 false 로 지정하여 소수가 아님을 표시

※ 중첩 반복문을 사용하기 때문에 실행 시간이 많이 소요될 것으로 예상되지만 실제로 시간 복잡도는 O(n log log n) 만큼 걸림

③. 소수 값 출력

4) 정답 - 제안

```
/*=======================================
  ybmit.com 사이트에는 정답이 없습니다.
  4차 10번 프로젝트의 src 폴더에 T_Solution10_1.java , T_Solution10_2.java 파일을
  참고 바랍니다.
 =======================================*/
```

① 브루트 포스 방식

```java
public int solution(int a, int b) {
    int answer = 0;
    boolean prime;
    for(int n=2;n<=Math.sqrt(b);n++) {
        prime=true;
        for(int i=2;i<n;i++) {
            if(n%i==0) {
                prime=false;
                break;
            }
        }
        if (prime==true) {
                int tmp=n*n;
                if (a<= tmp && tmp<=b)
                    answer+=1;
                if(n<=1000) {
                    int tmp2=n*n*n;
                    if (a<= tmp2 && tmp2<=b)
                        answer+=1;
                }
        }
    }
    return answer;
}
```

세제곱해서 10억이 넘지 않도록 미리 제한

①. 2 부터 $\sqrt{b}$ 의 정수값까지 가져오도록 Math.sqrt( ) 를 사용
prime=true 로 초기화

②. 소수 판별을 위해 i 를 2 부터 자연수(n) 보다 1 작은 동안 반복하면서

- 나누어 떨어지는 수가 있으면 소수가 아니므로 prime 을 false 변경

③. 소수이면서 범위를 만족하면 (a<=n^2<=b or a<= n^3<=b) answer 개수를 증가

- 세제곱해서 10 억이 넘지 않도록 값을 미리 제한

② 에라토스테네스의 체를 이용한 방식

```
public int solution(int a, int b) {
    int answer = 0;
    int k=(int)(Math.sqrt(b));        //마지막 소수는 b의 제곱근
    boolean arr[] = new boolean[k+1];

    //k개의 요소를 true로 초기값 설정
1   for(int i=2; i<=k; i++)
       arr[i]=true;

2   //소수의 배수에 해당하는 값들을 false로 지우기
    for(int n=2 ; n<=Math.sqrt(k) ; n++) {
        if(arr[n]==true) {                  //n이 소수이면
          for(int j=(n+n);j<=k;j=(j+n)) { //소수 n의 배수들을 false로 변경
              arr[j]=false;
          }
        }
     }
```

검색할 끝 범위를 $\sqrt{b}$ 의 정수값까지 가져오도록 Math.sqrt( ) 를 사용하여 k 에 저장

① 2 부터 $\sqrt{b}$ (k)까지 true 로 초기화

② 자연수(n) : 2 부터 $\sqrt{k}$ 까지 반복

- arr[n]이 ture 이면(n 이 소수이면)
- 이후 n 의 배수들을 false 로 변경

➔ 소수를 판별해야 하는 수의 범위가 커질수록 '에라토스테네스의 체' 방식을 사용하는 것이 더 효율적