분반 : 01
학번 : 201802092
이름 : 박민

## 문제 해결 방법

1) 프랙탈 삼각형 변 길이 구하기

기본적인 아이디어는 프랙탈 삼각형 변의 길이가 항에 따른 점화식이 존재하기 때문에 이를 재귀로 일반화 하면 된다는 것이다. 점화식은 다음과 같다 (3^n / 2^(n-1))

재귀를 설계할 때 Base-Case는 항이 2 이하 즉 1, 0일 때 분자에 3을 곱한 값을 리턴, 아닐 경우 점화식을 따르는 것으로 했다. 이 때 분자와 분모 따로 리턴을 해주어야 하는데, python과 다르게 java는 리턴을 한번에 2개를 할 수 없기 때문에 분수를 나타내는 Fraction 클래스를 만들었다. (아래 참조)

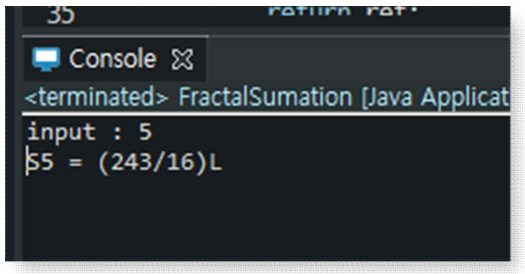
```java
/* @class fraction 분수를 표현할 객체 */
class Fraction {
    int mother; // 분모
    int son; // 분자

    public Fraction() { // 1/1로 시작
        this.mother = 1;
        this.son = 1;
    }

    public void setSon(int son) { // setter
        this.son = son;
    }

    public void setMother(int mother) { // settr
        if(mother == 0) // divide by zero exception
            System.out.println("cannot divide by zero");
        else
            this.mother = mother;
    }
    @Override
    public String toString() { // 결과 출력 시
        return "(" + this.son + "/" + this.mother + ")L";
    }
}
```

처음에는 분모가 0이 올 수 없기 때문에 setter를 만들어 Exception Handling 했지만 곰곰이 생각해보니 그럴 필요가 없었다. 그래도 그냥 삭제하지는 않았다.

```java
public class FractalSumation {
    /* 문제 풀이 재귀 함수 */
    /* 점화식 = (3^n / 2^(n-1)) */
    static Fraction run(int n, Fraction ret) {
        if(n < 2) { // base case
            ret.setSon(ret.son * 3);
            return ret;
        } else {
            ret.setSon(ret.son * 3);
            ret.setMother(ret.mother * 2);
            return run(n - 1, ret); // recursion calling
        }
    }
```

풀이에 사용된 메서드는 정적 메서드를 사용했다. 앞서 말했듯이 base-case를 설정하고 그 이외의 상황에는 점화식을 따르는 방향으로 재귀 함수를 설계했다.

결과는 다음과 같다.



2) 파스칼 삼각형

파스칼 삼각형 또한 점화식으로 일반화 하여 푸는 것이 쉬울 것이라고 생각해 1번과 같은 방식으로 하되 출력에 공을 들였다.

```
/* 문제 풀이 재귀 함수 */
static int solve(int i, int j) { // i = row, j = coloum
    if ((j == 1) || (j == i)) return 1; // base cases
    else return solve(i - 1, j - 1) + solve(i - 1, j); // 파스칼 삼각형 점화식
}
```

재귀로 짜여진 solve 함수는 row와 coloum을 받아와서 그 위치에 해당하는 값을 리턴한다. 파스칼 삼각형은 좌상단과 우상단의 값을 더한 값이 나오도록 만들면 되고 base-case에서는 1을 출력하도록 했다.

```
/* 예쁘게 출력하는 함수 */
static void print(int n, int space, int i, int j) { // i = row, j = coloum
    if (n < 1)  return;// base case
    else if (space != 0) { // 공백 출력
        System.out.print(String.format("%4s", " ")); // 공백 채워주고
        print(n, space - 1, i, j); // recursion call
    } else if (i < j) { // 개행
        System.out.println(); // 한줄 띄고
        space = (n - 1) - 1; // 공백 개수 다시 정해주고
        print(n - 1, space, i + 1, 1); // recursion call
    } else { // 파스칼 삼각형에 맞는 수를 가져옴
        System.out.print(String.format("%8d", solve(i, j)));
        print(n, space, i, j + 1); // recursion call
    }
}
```

출력하는 함수는 파스칼 삼각형이 정삼각형 형태로 출력되도록 만든 것이다. 우선 basecase는 n이 0인 경우, 즉 파스칼 삼각형을 모두 출력했을 때이다. 또한 분기문의 경우 여러 가지 경우로 나뉘었는데, space 변수는 앞으로 더 출력해야할 공백의 횟수를 카운트 해주며 이것이 0이 아니라면(공백이 필요한 경우라면) 공백을 출력한다. 또한 row보다 coloum이 큰 경우에는 개행을 해주며 그것도 아닌 경우라면 해당 위치에 맞는 파스칼 수가 들어간다.

결과는 다음과 같다.