PART 4 - 연구과제 풀이

각도기 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120829?language=java

📌 문제 해결 개요

이 문제는 주어진 `angle` 값을 이용하여 각도를 네 가지 유형으로 분류하는 문제입니다.

🔷 핵심 개념

- `angle < 90` → 예각(1)
- `angle == 90` → 직각(2)
- `90 < angle < 180` → 둔각(3)
- `angle == 180` → 평각(4)

🚀 해결 방법

1. `if-else` 문을 사용하여 `angle`의 범위를 확인하고 각 유형에 맞는 값을 반환한다.

```
class Solution {
    public int solution(int angle) {
        int answer = 0;

        if (angle < 90) {
            answer = 1; // 예각
        } else if (angle == 90) {
            answer = 2; // 직각
        } else if (angle < 180) {
            answer = 3; // 둔각
        } else {
            answer = 4; // 평각
        }

        return answer;
    }

    public static void main(String[] args) {
        Solution s = new Solution();

        // 테스트 예제 실행
        System.out.println(s.solution(70));  // 1
        System.out.println(s.solution(91));  // 3
        System.out.println(s.solution(180)); // 4
    }
}
```

숫자 비교하기 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120807?language=java

📌 문제 해결 개요

이 문제는 두 정수를 비교하여 같으면 `1`, 다르면 `-1`을 반환하는 단순한 비교 연산 문제입니다.

🔷 핵심 개념

- `num1 == num2` → 1 반환
- `num1 != num2` → -1 반환

🚀 해결 방법

1. `if-else` 문 또는 삼항 연산자를 사용하여 두 수를 비교한다.
2. 조건을 만족하는 값을 반환한다.

```
class Solution {
    public int solution(int num1, int num2) {
        int answer = 0;

        if (num1 == num2) {
            answer = 1;
        } else {
            answer = -1;
        }

        return answer;
    }

    public static void main(String[] args) {
        Solution s = new Solution();

        // 테스트 예제 실행
        System.out.println(s.solution(2, 3));  // -1
        System.out.println(s.solution(11, 11)); // 1
        System.out.println(s.solution(7, 99));  // -1
    }
}
```

옷가게 할인 받기 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120818?language=java

📌 문제 해결 개요

이 문제는 주어진 가격에 따라 다른 할인율을 적용하여 최종 결제 금액을 계산하는 문제입니다.

🔷 핵심 개념

- 가격이 `500,000` 이상이면 `20%` 할인
- 가격이 `300,000` 이상이면 `10%` 할인
- 가격이 `100,000` 이상이면 `5%` 할인
- 할인 조건에 맞지 않으면 가격 그대로 반환
- **소수점 이하를 버리기 위해 정수형 변환 필요**

🚀 해결 방법

1. 가격이 `500,000` 이상인지 확인하고 `0.8`을 곱하여 할인 적용
2. 가격이 `300,000` 이상이면 `0.9`를 곱하여 할인 적용
3. 가격이 `100,000` 이상이면 `0.95`를 곱하여 할인 적용
4. 해당되지 않는 경우 가격 그대로 반환

```
class Solution {
    public int solution(int price) {
        int answer = 0;
        if (price >= 500000) {
            answer = (int) (price * 0.8); // 20% 할인 적용
        } else if (price >= 300000) {
            answer = (int) (price * 0.9); // 10% 할인 적용
        } else if (price >= 100000) {
            answer = (int) (price * 0.95); // 5% 할인 적용
        } else {
            answer = price; // 할인 없음
        }

        return answer;
    }

    public static void main(String[] args) {
        Solution s = new Solution();

        // 테스트 예제 실행
        System.out.println(s.solution(150000)); // 142500
        System.out.println(s.solution(580000)); // 464000
    }
}
```

피자 나눠 먹기 (1) (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120814?language=java

📌 문제 해결 개요

이 문제는 `n`명이 최소한 한 조각씩 먹기 위해 필요한 **피자의 최소 판 수**를 구하는 문제입니다. 피자는 한 판에 `7`조각으로 제공됩니다.

🔷 핵심 개념

- 한 판(7조각)으로 `n`명이 나눠 먹을 수 있도록 최소한의 판 수를 구해야 함
- `(n / 7)` → `n`을 7로 나눈 몫이 기본적으로 필요한 판 수
- `n % 7 > 0` → 나머지가 있다면 한 판을 추가해야 함

🚀 해결 방법

1. `n / 7`을 계산하여 기본적으로 필요한 피자 판 수를 구한다.
2. `n % 7 > 0`이면 추가로 한 판을 더 주문해야 한다.
3. 최종적으로 계산된 피자 판 수를 반환한다.

```
class Solution {
    public int solution(int n) {
        int answer = n / 7; // 7로 나눈 몫을 기본 피자 판 수로 설정

        if (n % 7 > 0) { // 나머지가 있는 경우 (즉, 추가 피자가 필요한 경우)
            answer++; // 피자 한 판을 추가
        }

        return answer; // 계산된 피자 판 수 반환
    }

    public static void main(String[] args) {
        Solution s = new Solution();

        // 테스트 예제 실행
        System.out.println(s.solution(7));  // 1
        System.out.println(s.solution(1));  // 1
        System.out.println(s.solution(15)); // 3
    }
}
```

피자 나눠 먹기 (3) (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120816?language=java

📌 문제 해결 개요

이 문제는 `n`명이 최소한 한 조각씩 먹기 위해 필요한 **피자의 최소 판 수**를 구하는 문제입니다. 피자는 `slice` 조각으로 잘려 제공됩니다.

🔷 핵심 개념

- `n / slice` → `n`명을 위해 필요한 기본 피자 판 수
- `n % slice != 0` → 나머지가 있다면 한 판을 추가해야 함

🚀 해결 방법

1. `n / slice`를 계산하여 기본적으로 필요한 피자 판 수를 구한다.
2. `n % slice != 0`이면 추가로 한 판을 더 주문해야 한다.
3. 최종적으로 계산된 피자 판 수를 반환한다.

```
class Solution {
    public int solution(int slice, int n) {
        int answer = n / slice; // 기본적으로 필요한 피자 판 수 계산

        if (n % slice != 0) { // 나머지가 있으면 한 판 추가
            answer++;
        }

        return answer; // 최종적으로 필요한 피자 판 수 반환
    }

    public static void main(String[] args) {
        Solution s = new Solution();

        // 테스트 예제 실행
        System.out.println(s.solution(7, 10)); // 2
        System.out.println(s.solution(4, 12)); // 3
    }
}
```

치킨 쿠폰 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120884?language=java

📌 문제 해결 개요

이 문제는 `chicken` 마리를 주문했을 때 쿠폰을 이용해 받을 수 있는 **최대 서비스 치킨의 수**를 계산하는 문제입니다.

🔷 핵심 개념

- 치킨 `1`마리당 `1`장의 쿠폰이 발급됨
- `10`장의 쿠폰을 모으면 `1`마리의 서비스 치킨을 받을 수 있음
- 서비스 치킨도 쿠폰이 발급되므로 반복적으로 교환 가능
- `while` 반복문을 활용하여 쿠폰을 최대한 활용

🚀 해결 방법

1. 주문한 치킨의 수만큼 초기 쿠폰을 받는다.
2. `while`문을 사용하여 쿠폰이 `10`장 이상일 때마다 서비스 치킨을 받을 수 있도록 한다.
3. 서비스 치킨을 받은 만큼 쿠폰 개수를 업데이트한다.
4. 최종적으로 받을 수 있는 서비스 치킨의 개수를 반환한다.

```
class Solution {
    public int solution(int chicken) {
        int answer = 0; // 받을 수 있는 총 서비스 치킨 수

        while (chicken >= 10) { // 쿠폰이 10장 이상일 때만 교환 가능
            int service = chicken / 10; // 10개당 서비스 치킨 수 계산
            int remainder = chicken % 10; // 교환 후 남은 쿠폰 수 계산
            answer += service; // 총 서비스 치킨 수에 추가
            chicken = service + remainder; // 새로운 쿠폰 개수로 업데이트
        }

        return answer;
    }

    public static void main(String[] args) {
        Solution s = new Solution();

        // 테스트 예제 실행
        System.out.println(s.solution(100));  // 11
        System.out.println(s.solution(1081)); // 120
    }
}
```

구슬을 나누는 경우의 수 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120840?language=java

📌 문제 해결 개요

이 문제는 `balls`개의 구슬 중 `share`개의 구슬을 선택하는 **조합(combination)**의 개수를 구하는 문제입니다.
재귀 호출 없이 **반복문을 활용하여 효율적으로 조합을 계산**하는 방법을 사용합니다.

🔷 핵심 개념

✅ 조합 공식

$$C(n, r) = \frac{n!}{(n-r)! \times r!}$$

즉, `n`개의 항목 중 `r`개를 선택하는 경우의 수는 위의 공식으로 구할 수 있습니다.

✅ 조합의 대칭성

조합의 중요한 성질 중 하나는 다음과 같습니다.

$$C(n, r) = C(n, n-r)$$

예를 들어, $C(5, 3) = C(5, 2)$ 입니다.

- `5`개의 항목 중 `3`개를 선택하는 경우는, **5개 중 나머지 `2`개를 제외하는 경우**와 동일합니다.
- 즉, 구해야 하는 선택 개수를 **작은 값으로 변환**하면 불필요한 연산을 줄일 수 있습니다.
- 예를 들어, $C(30, 15) = C(30, 15)$, 하지만 $C(30, 15) = C(30, 15)$ 는 효율적인 계산을 위해 작은 값인 `15`를 기준으로 연산하는 것이 더 효율적입니다.

✅ 조합 최적화 (팩토리얼을 직접 계산하지 않음)

- 팩토리얼을 직접 계산하면 수가 매우 커져서 **오버플로우 위험**이 있음.
- 대신, **불필요한 곱셈을 줄이고 필요한 값만 계산**하여 효율적으로 조합을 구함.

🚀 해결 방법

1. $C(n, r) = C(n, n-r)$ 성질을 활용하여 **r을 더 작은 값으로 변환** ($r > n - r$ 이면 $r = n - r$).
2. 조합 공식에서 **곱셈과 나눗셈을 순차적으로 수행하여 큰 수 연산을 방지**.
3. 최종적으로 계산된 조합 값을 반환.

```
public class Solution {
    public int solution(int balls, int share) {
        return combination(balls, share);
    }

    public int combination(int n, int r) {
```

```
        if (r > n - r) {
            r = n - r; // 조합의 대칭성: C(n, r) == C(n, n-r)
        }

        long result = 1; // 결과값을 저장할 변수 (오버플로우 방지 위해 long 사용)

        // (n-r+1)부터 n까지 곱하고, 동시에 1부터 r까지 나눠서 계산
        for (int i = 0; i < r; i++) {
            result *= (n - i);
            result /= (i + 1);
        }

        return (int) result; // 결과 반환
    }

    public static void main(String[] args) {
        Solution s = new Solution();

        // 테스트 예제 실행
        System.out.println(s.solution(3, 2));  // 3
        System.out.println(s.solution(5, 3));  // 10
        System.out.println(s.solution(30, 15)); // 155117520
    }
}
```