Professional Coding Specialist

COS Pro JAVA 1 급

25 강-26 강. 모의고사 6 차

1. 모의고사 6 차(1-10 번)

과정 소개

COS Pro JAVA 1 급 모의고사 6 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
/*=======================================
   6차 1번   6차 1급 1_initial_code.java
 =======================================*/

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

class Solution {
    public int solution(int n, int[][] garden) {
        // 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 n1 = 3;
        int[][] garden1 = {{0, 0, 0}, {0, 1, 0}, {0, 0, 0}};
        int ret1 = sol.solution(n1, garden1);

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

        int n2 = 2;
        int[][] garden2 = {{1, 1}, {1, 1}};
        int ret2 = sol.solution(n2, garden2);

        // [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
- n x n 격자 모양의 정원에 개화하지 않은 꽃과 개화한 꽃을 심은 상황
- 하루가 지나면 꽃이 피어 있던 영역의 상, 하, 좌, 우에 있는 피지 않았던 꽃도 개화함
- 정원 크기 n 과 현재 정원에 꽃이 피어 있는 상태를 표현하는 2 차원 배열 garden 이 주어졌을 때 꽃이 모두 피는 데 걸리는 일자 수를 계산하는 문제

3) 정답

- 주요 아이디어 정리 : garden = [[0, 0, 0], [0, 1, 0], [0, 0, 0]] 인 경우

첫 날 개화 현황

0	0	0
0	1	0
0	0	0

- [1, 1] 영역의 꽃이 피어 있음

1 일 경과 후 개화 현황

0	1	0
1	1	1
0	1	0

- [1, 1]을 기준으로 상/하/좌/우 의 위치인 [0, 1], [2, 1], [1, 0], [1, 2] 영역의 꽃이 추가로 개화

2 일이 경과 후 개화 현황

1	1	1
1	1	1
1	1	1

- [0, 1], [2, 1], [1, 0], [1, 2] 4 개 영역을 기준으로 상/하/좌/우 의 영역에 추가로 꽃이 개화함으로써 전체 정원의 꽃이 모두 개화

➔ 첫 날 개화한 영역을 이용하여 1 일 경과 후에 개화하는 영역을 구하고, 1 일 경과 후 새로 개화한 영역을 이용하여 2 일 경과 후에 개화하는 영역을 구함

➔ 새로 개화한 영역만을 이용하여 꽃이 피지 않은 영역 중 다음 날 개화하는 좌표와 소요 일자를 구하기 위해서는 큐(Queue)를 사용하는 것이 효과적

- 정답 코드

```
1 class Flower {
      int x, y, day;
      Flower(int x, int y, int day) {
          this.x = x;
          this.y = y;
          this.day = day;
      }
  }
```

①. Flower 클래스 : 큐에 저장될 자료의 형태를 클래스로 정의(행, 열, 일수)

- x, y : 꽃이 피는 위치 (x:행, y:열)
- day : 꽃이 피는 날

```
  int answer = 0;

2 int[] dx = { -1, 1, 0, 0 };
  int[] dy = { 0, 0, -1, 1 };

3 Queue <Flower> q = new LinkedList <Flower>();

  for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) {
          if(garden[i][j] == 1)
              q.offer(new Flower(i, j, 0));
      }
  }
```

② 선택한 영역의 상,하,좌,우 인덱스 지정을 위한 x 좌표 변화값을 dx 배열로 저장하고, dx 의 각 항목에 대응하는 y 인덱스 변화값을 dy 배열에 순서대로 저장

③ 매개변수 garden 배열에서 항목값이 개화한 영역을 나타내는 값인 1 인 항목을 찾아서 큐에 추가

- 큐에 추가하는 항목 = (현 위치의 x 좌표를 나타내는 인덱스, 현 위치의 y 좌표를 나타내는 인덱스, 현재 위치에 개화하는데 소요된 일자)로 구성된 Flower 클래스
- q 에 대한 offer( ) 메소드를 이용하여 (i , j , 0) 로 된 항목을 큐의 초기 항목으로 추가
  - i : 현 위치의 x 좌표를 나타내는 인덱스 값
  - j : 현 위치의 y 좌표를 나타내는 인덱스 값
  - 0 : 시작하는 상태이므로 소요 일자가 없기 때문에 0 으로 지정

```
while(!q.isEmpty()) {
4-1   Flower flower = q.peek();
      q.poll();

4-2   for(int i = 0; i < 4; i++) {
          int nextX = flower.x + dx[i];
          int nextY = flower.y + dy[i];
          int nextDay = flower.day + 1;

4-3       if( (0 <= nextX && nextX < n && 0 <= nextY && nextY < n)
             && garden[nextX][nextY] == 0) {
              garden[nextX][nextY] = 1;
              answer = nextDay;
              q.offer(new Flower(nextX, nextY, nextDay));
          }
      }
}
return answer;
```

④ while 문에서 새로 개화한 영역을 저장하는 큐 q 에 대한 empty( ) 메소드를 실행하여 q 가 비어 있지 않는 동안 반복하도록 지정

4-1) q.peek( )을 사용하여 맨 위의 데이터를 조회해서 x, y, day 정보를 담을 수 있는 Flower 객체 타입으로 flower 변수에 넣고, q.poll() 로 맨 위의 데이터를 꺼낸다. (peek()을 사용하지 않아도 가능)

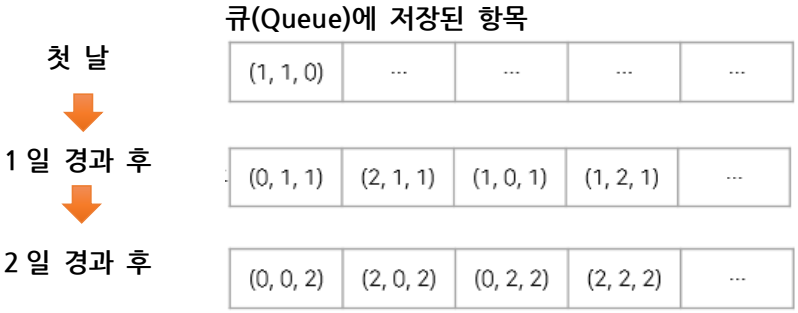
4-2) for 문을 이용하여 현재 개화한 영역의 상/하/좌/우 4 개의 영역을 차례대로 가져오기 위해 4 번 반복하도록 지정

- 큐에서 가져온 영역의 x 좌표에 x 좌표 변화값을 더하여 nextX 에 저장하고, y 좌표에 y 좌표 변화값을 더하여 nextY 에 저장 → nextX, nextY 는 새로 개화하는 영역의 좌표
- 새로 개화하는데 소요된 일자는 day(큐에서 가져온 소요 일자) + 1 로 계산하여 nextDay 에 저장

4-3) nextX 와 nextY 의 값이 garden 배열의 인덱스 범위를 벗어나지 않고, 해당하는 위치의 항목값이 = 0 인지 확인

- garden 배열에서 새로 구한 영역의 항목값을 1 로 할당하여 개화한 상태로 변경하고 리턴 값을 저장하는 변수 answer 에 새로 개화하는데 소요된 일자인 nextDay 를 할당
- 새로 개화한 영역에 대한 정보를 (x 좌표, y 좌표, 소요 일자) 형태의 Flower 객체로 생성하여 큐에 새로운 항목으로 추가 (q.offer())

※ garden = [[0, 0, 0], [0, 1, 0], [0, 0, 0]] 인 경우 큐(Queue)의 변화



2. 문제 2

1) 문제 코드

```
/*=======================================
   6차 2번   6차 1급 2_initial_code.java
 =======================================*/


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

class Solution {
    public int solution(int K, String[] words) {
        // 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 K = 10;
        String[] words = {new String("nice"), new String("happy"), new String("hello"), new String("world"), new String("hi")};
        int ret = sol.solution(K, words);

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

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제
- 한 줄에 K 자를 적을 수 있는 메모장에 주어진 단어들을 한 칸씩 띄워 적어야 하고, 한 줄에 적지 못하는 단어는 그 다음 줄부터 새로 적어야 함

- 위의 조건에서 주어진 단어를 모두 적었을 때 생성되는 줄 수를 리턴하는 프로그램을 작성

3) 정답

- 주요 아이디어 정리 : 한 줄에 10 글자를 적을 수 있고, 주어진 단어들이 [“nice” , “happy”, “hello”, “world”, “hi”] 인 경우

Line No.1	“nice_happy”	10 칸이 모두 채워졌으므로 세 번째 단어는 다음 줄에 작성
Line No.2	“hello_____”	남은 칸이 5 칸인데 네 번째 단어를 적기 위해 필요한 공간은 공란 1 칸 + world 의 글자 수 5 = 6 칸이므로 네 번째 단어는 다음 줄에 작성
Line No.3	“world_hi__”	네 번째 단어와 다섯 번째 단어를 세 번째 줄에 모두 작성

- 정답 코드

```
/*=======================================
   6차 1급 2_solution_code.java
   ybmit.com 사이트의 샘플 파일에는 정답이 제시되어 있지 않습니다.
   아래와 같은 정답을 제안합니다.
=========================================*/
```

```
public int solution(int K, String[] words) {
    // 여기에 코드를 작성해주세요.
 ① int answer = 1;
    int sum = 0;

 ② for (int i=0; i<words.length ; i++) {
     ③ int len = words[i].length();
     ④ if (sum !=0)
            sum++;
     ⑤ sum+=len;
     ⑥ if (sum > K) {
            answer++;
            sum=len;
        }
    }
    return answer;
}
```

① 작성된 줄 수를 저장하는 변수 answer 를 1 로 초기화 하고, 작성된 칸 수를 저장하는 변수 sum 을 0 으로 초기화

② for 문을 이용하여 매개 변수 words 에 있는 단어를 하나씩 읽어 들임

③ 적어야 할 단어의 길이를 변수 len 에 저장

④ 새로 시작되는 줄이 아니라면 단어 사이에 한 칸을 띄어야 하므로 sum 을 1 만큼 증가

⑤ sum 에 적어야 할 단어의 길이 len 을 합산. sum 은 현재 줄에서 단어가 적힌 총 칸수가 합산됨

⑥. 적은 칸수가 한 줄에 적을 수 있는 칸수 K 보다 크면, 새로운 줄에 단어를 적어야 하므로 작성된 줄 수를 갖는 answer 를 1 만큼 증가시키고 칸수가 모자라서 적지 못한 단어를 적기 위해 sum 에 현재 단어의 길이 저장

3. 문제 3

1) 문제 코드

```
/*=======================================
   6차 3번   6차 1급 3_initial_code.java
 =======================================*/


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

class Solution {
    public int solution(int[] arr, int K) {
        // 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[] arr = {9, 11, 9, 6, 4, 19};
        int K = 4;
        int ret = sol.solution(arr, K);

        // [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 arr 배열에서 K 개의 항목을 선택하여 부분 배열을 만드는데, 그 부분 배열에서의 최댓값과 최솟값의 차이가 최소가 되도록 선택해서 그 최솟값을 리턴해야 함

3) 정답

- 주요 아이디어 정리
  - 부분 배열에서 최대/최소 항목값의 차이가 가장 작은 것을 찾기 위해 원본 배열의 항목 크기가 비슷한 값들이 모이도록 sort( ) 메소드를 사용해서 정렬
  - 정렬된 원본 배열에 대해서 K 개만큼 슬라이싱한 것을 부분 배열로 만들고 부분 배열의 최대/최소 값의 차이를 비교

- 정답 코드

```
/*=======================================
   6차 1급 3_solution_code.java
```

```
    ybmit.com 사이트의 샘플 파일에는 정답이 제시되어 있지 않습니다.
    아래와 같은 정답을 제안합니다.
==========================================*/
```

```java
import java.util.*;

class Solution {
    public int solution(int[] arr, int K) {
        // 여기에 코드를 작성해주세요.
        int answer = 0;
    ①   Arrays.sort(arr);
    ②   answer =1001;

    ③   for(int i =0; i<= arr.length-K ; i++)
    ④       answer = Math.min(answer, arr[i+K-1]-arr[i]);
        return answer;
    }
}
```

① 매개변수로 받은 배열 arr 에 대해 Arrays.sort( ) 를 사용하여 오름차순으로 정렬
Arrays 는 java.util.Arrays 에 있기 때문에 문제에서 이미 java.util.* 이 import 되어 있으므로 import 는 신경 쓰지 않아도 됨

② (부분 배열에서의 최댓값 - 부분 배열에서의 최솟값)의 최솟값의 초기값을 설정

③ for 문을 이용하여 arr 의 0 번 인덱스 항목부터 시작해서 arr 배열의 총 길이에서 K 개를 뺀 길이까지의 항목을 탐색

④ (부분 배열에서의 최댓값 - 부분 배열에서의 최솟값)과 지금까지 계산된 최솟값 answer 를 비교하여 더 작은 값을 answer 에 저장
- 인덱스 (i-K+1)부터 K 개 항목의 배열
- 부분 배열에서의 최솟값은 인덱스가 (i-K+1)인 항목의 값
- 부분 배열에서의 최댓값은 인덱스가 i 인 항목의 값

4. 문제 4

1) 문제 코드

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

class Solution {
    public int solution(int n, int mix, int k) {
        int answer = 0;

        int[] card = new int[n];
        for(int i = 0; i < n; i++)
            card[i] = i+1;

        while((mix--) != 0) {
            int[] cardA = new int[n/2];
            int[] cardB = new int[n/2];

            for(int i = 0; i < n; i++) {
                if(i < n/2)
                    cardA[i] = card[i];
                else
                    cardB[i-n/2] = card[i];
            }

            for(int i = 0; i < n; i++) {
                if(i % 2 == 0)
                    card[i] = cardA[i/2];
                else
                    card[i] = cardB[i/2];
            }
        }

        answer = card[k-1];

        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. 아래에는 잘못된 부
    public static void main(String[] args) {
        Solution sol = new Solution();
        int n = 6;
        int mix = 3;
        int k = 3;
        int ret = sol.solution(n, mix, k);

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

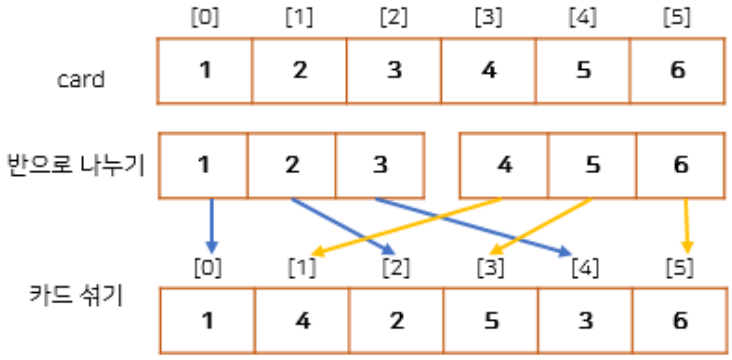
2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제
- 1 부터 n 까지의 숫자를 가진 카드 뭉치를 이등분한 후 두 개의 뭉치에서 한 개씩 번갈아 가져와 섞었을 때 아래에서부터 k 번째에 있는 카드를 알아내기 위해 작성한 프로그램에서 잘못된 부분을 찾아 한 줄만 수정하는 문제

- 1 부터 n 까지의 숫자를 배열에 차례대로 저장한 후 문제에서 제시한 방법대로 숫자를 배열에 재배열하는 패턴이 제시된 문제 코드로 제대로 구현되었는지 확인해야 함

3) 정답

- 주요 아이디어 정리



➔ 두 개로 나눈 카드 뭉치에서 앞쪽 카드 뭉치의 카드들은 짝수 인덱스에 할당하고, 뒤쪽 카드 뭉치의 카드들은 홀수 인덱스에 할당하여 카드를 섞도록 구현

- 정답 코드

```
public int solution(int n, int mix, int k) {
    int answer = 0;

①   int[] card = new int[n];
    for(int i = 0; i < n; i++)
        card[i] = i+1;

    while((mix--) != 0) {
②       int[] cardA = new int[n/2];
        int[] cardB = new int[n/2];

③       for(int i = 0; i < n; i++) {
            if(i < n/2)
                cardA[i] = card[i];
            else
                cardB[i] = card[i];
        }                    i - n/2

④       for(int i = 0; i < n; i++) {
            if(i % 2 == 0)
                card[i] = cardA[i/2];
            else
                card[i] = cardB[i/2];
        }
    }
⑤   answer = card[k-1];
    return answer;
}
```

①. 1 부터 n 까지의 수를 항목으로 갖는 card 배열을 생성

②. 카드 섞는 횟수만큼 반복하기

- card 를 절반으로 나누어 저장할 배열 cardA, cardB 를 만들어서 항목값을 0 으로 초기화

③. i 가 0 부터 n 미만까지 반복되면서

- i 값이 card 의 항목 개수를 반으로 나눈 몫보다 작으면, card[i]를 cardA [i]로 저장
- i 값이 card 의 항목 개수를 반으로 나눈 몫보다 크면, (i - card 의 항목 개수를 반으로 나눈 몫)을 cardB 배열의 인덱스로 지정하면서 card [i] 를 차례대로 저장. 즉 card 배열의 절반 이후의 항목들이 cardB 의 0 번째 항목부터 차례대로 복사됨. 만일 문제 코드처럼 cardB[i] = card[i] 를 실행하면 cardB 배열의 인덱스 범위를 벗어나게 되므로 오류 발생

④. i 가 0 부터 n 미만까지 반복되면서

- card 에 재배치할 항목의 인덱스가 짝수일 때는 cardA 의 해당하는 항목을 가져와서 저장
- card 에 재배치할 항목의 인덱스가 홀수일 때는 cardB 의 해당하는 항목을 가져와서 저장

⑤. k 번째 수를 찾아 리턴하기 위해 card[k-1]를 answer 에 저장

5. 문제 5

1) 문제 코드

```java
/*=======================================
   6차 5번   6차 1급 5_initial_code.java
 =======================================*/

class Solution {
    public int solution(int[][] board) {
        int answer = 0;

        int[][] coins = new int[4][4];
        for(int i = 0; i < 4; i++) {
            for(int j = 0; j < 4; j++) {
                if(i == 0 && j == 0)
                    coins[i][j] = board[i][j];
                else if(i == 0 && j != 0)
                    coins[i][j] = board[i][j] + coins[i][j-1];
                else if(i != 0 && j == 0)
                    coins[i][j] = board[i][j] + coins[i-1][j];
                else
                    coins[i][j] = board[i][j] + Math.max(coins[i][j], coins[i-1][j-1]);
            }
        }

        answer = coins[3][3];
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. 아래에는 잘못된 부분이 없으니 위
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[][] board = {{6, 7, 1, 2}, {3, 5, 3, 9}, {6, 4, 5, 2}, {7, 3, 2, 6}};
        int ret = sol.solution(board);

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

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제
- 오른쪽과 아래로만 이동할 수 있는 말을 사용해서 4 x 4 격자 보드의 가장 왼쪽 위에서 가장 오른쪽 아래로 말을 이동시키며 지나가는 구역의 코인을 누적 합산
- 말이 보드의 종료 지점에 도착했을 때 최대로 얻을 수 있는 누적 코인을 계산하는 프로그램에서 잘못된 부분을 찾아 한 줄만 수정하는 문제

3) 정답

- 주요 아이디어 정리

〈 보드 구역별 코인 〉

6	7	1	2
3	5	3	9
6	4	5	2
7	3	2	6

〈 보드 구역별 최대 누적 코인 〉

① 첫 번째 행의 각 구역별 최대 누적 코인은 현재 구역의 바로 왼쪽 구역의 누적 코인 값 + 현재 구역 코인으로 계산

6	13	14	16

② 첫 번째 열의 각 구역별 최대 누적 코인은 바로 윗줄 구역의 누적 코인 값 + 현재 구역 코인으로 계산

6	13	14	16
9			
15			
22			

③ 보드의 나머지 구역별 최대 누적 코인은 현재 구역의 바로 왼쪽 구역의 누적 코인과 현재 구역의 바로 윗줄 구역의 누적 코인 중 큰 값 + 현재 구역 코인으로 계산(윗줄의 누적 코인이 더 크기 때문에 : 13 + 5 = 18)

6	13	14	16
9	18		
15			
22			

- 정답 코드

```
public int solution(int[][] board) {
    int answer = 0;

 ① int[][] coins = new int[4][4];
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
        ②   if(i == 0 && j == 0)
                coins[i][j] = board[i][j];
        ③   else if(i == 0 && j != 0)
                coins[i][j] = board[i][j] + coins[i][j-1];
        ④   else if(i != 0 && j == 0)
                coins[i][j] = board[i][j] + coins[i-1][j];
        ⑤   else
                coins[i][j] = board[i][j] + Math.max(coins[i-1][j], coins[i][j-1]);
        }
    }

 ⑥ answer = coins[3][3];
    return answer;
}
```

① 각 구역의 누적 코인을 저장하는 4 x 4 크기의 2 차원 배열 coins 를 생성하여 0 으로 초기화
② 현재 구역의 행과 열이 모두 0 이면 보드의 시작점이므로 현재 구역의 코인만 누적 코인 값으로 할당
③ 현재 구역의 행 = 0 and 열 != 0 이면 시작점을 제외한 보드의 첫 번째 행에 위치한 구역이기 때문에 현재 구역의 이전 구역은 왼쪽 구역만 존재 → (현재 구역의 코인 값 + 바로 왼쪽 구역의 누적 코인)을 현재 구역의 누적 코인 값으로 할당
④ 현재 구역의 행 != 0 and 열 = 0 이면 시작점을 제외한 보드의 첫 번째 열에 위치한 구역이기 때문에 현재 구역의 이전 구역은 위쪽 구역만 존재 → (현재 구역의 코인 값 + 바로 윗줄 구역의 누적 코인)을 현재 구역의 누적 코인 값으로 할당
⑤ 나머지 구역의 누적 코인 값은 { 현재 구역의 코인 값 + max(바로 윗줄 구역의 누적 코인, 바로 왼쪽 구역의 누적 코인) } 으로 할당. 문제 코드는 계산하지도 않은 현 구역의 누적 코인 값과 현 구역의 왼쪽 대각선 위쪽 방향 구역 중 큰 값을 더하도록 되어 있어서 원하는 값을 얻을 수 없음
⑥ 도착점에 저장되어 있는 누적 코인 값은 answer 에 저장

6. 문제 6

1) 문제 코드

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

class Solution {
    public int solution(int[][] grid) {
        int answer = 0;
        for(int i = 0; i < 4; i++)
            for(int j = 0; j < 4; j++)
                for(int k = j + 1; k < 4; k += 2)
                    answer = Math.max(answer, Math.max(grid[i][j] + grid[j][k], grid[j][k] + grid[k][i]));
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. 아래에는 잘못된 부분이 없으니 위의 코드만 수정하세요.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[][] grid = {{1, 4, 16, 1}, {20, 5, 15, 8}, {6, 13, 36, 14}, {20, 7, 19, 15}};
        int ret = sol.solution(grid);

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

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제
- 4 x 4 격자로 구성된 정사각형 종이를 가로축 혹은 세로축에 평행한 격자 선을 따라 접었을 때 만나는 수의 합이 최대가 되는 것을 리턴하는 프로그램에서 잘못된 곳 한 줄만 찾아 수정.

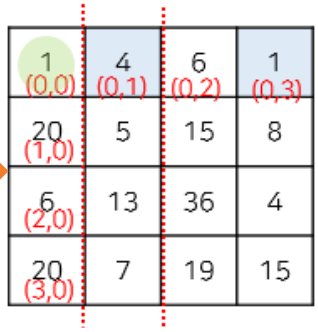
3) 정답

- 주요 아이디어 정리

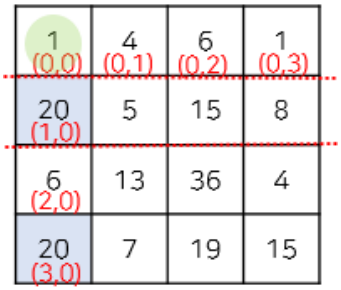
〈 4x4 종이에 적힌 수 〉

1 (0,0)	4 (0,1)	6 (0,2)	1 (0,3)
20 (1,0)	5	15	8
6 (2,0)	13	36	4
20 (3,0)	7	19	15

〈 종이를 접었을 때, (0,0) 좌표에 있는 1 이 만나는 수 〉



➔ 종이를 세로축에 평행하게 접으면 (0, 1)에 있는 4 와 (0, 3)에 있는 1 을 만날 수 있음



➔ 종이를 가로축에 평행하게 접으면 (1, 0)에 있는 20 와 (3, 0)에 있는 20 을 만날 수 있음

∴ (row, column) 위치에 있는 수가 4 x 4 종이를 접었을 때 만나는 수

- 종이를 **세로**로 접을 경우 : (row, column+1)과 (row, column+3) 에 있는 수
- 종이를 **가로**로 접을 경우 : (row+1, column)과 (row+3, column) 에 있는 수

(단, 종이의 범위는 벗어나지 않아야 함)

- 정답 코드

```
public int solution(int[][] grid) {
    int answer = 0;
  ①for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
  ②         for(int k = j + 1; k < 4; k += 2)
  ③             answer = Math.max(answer, Math.max(grid[i][j] + grid[j][k], grid[j][i] + grid[k][i]));
    return answer;
}
```

①. 중첩 for 문을 이용하여 0 부터 3 까지의 수를 i 와 j 에 각각 받아서 좌표를 구성하는 수로 이용

②. for 문을 이용하여 j+1 부터 2 만큼 증가하여 4 보다 작은 수를 k 로 받음

③. 기준 좌표 (i, j) 에 대해서 세로로 접어서 만나는 항목인 (i, k)을 더한 값, 기준 좌표 (j, i) 에 대해서 가로로 접어서 만나는 항목인 (k, i)을 더한 값 중 큰 값을 구한 후 기존에 구해 놓은 최댓값과 비교하여 최댓값을 다시 지정

7. 문제 7

1) 문제 코드

```java
/*=====================================
   6차 7번   6차 1급 7_initial_code.java
 =====================================*/

class Solution {
    public int solution(int K, int[] numbers, String[] UpDown) {
        int left = 1;
        int right = K;
        for(int i = 0; i < numbers.length; i++){
            int num = numbers[i];
            if(UpDown[i].equals("UP"))
                left = @@@;
            else if(UpDown[i].equals("DOWN"))
                right = @@@;
            else if(UpDown[i].equals("RIGHT"))
                return 1;
        }
        return @@@;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int K1 = 10;
        int[] numbers1 = {4, 9, 6};
        String[] UpDown1 = {new String("UP"), new String("DOWN"), new String("UP")};
        int ret1 = sol.solution(K1, numbers1, UpDown1);

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

        int K2 = 10;
        int[] numbers2 = {2, 1, 6};
        String[] UpDown2 = {new String("UP"), new String("UP"), new String("DOWN")};
        int ret2 = sol.solution(K2, numbers2, UpDown2);

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

        int K3 = 100;
        int[] numbers3 = {97, 98};
        String[] UpDown3 = {new String("UP"), new String("RIGHT")};
        int ret3 = sol.solution(K3, numbers3, UpDown3);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret3 + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 빈 곳을 채우는 문제
- 출제자가 생각한 수를 참가자가 맞추는 게임을 구현하는 프로그램으로 참가자가 제시한 수에 대해서 출제자가 “UP”, “DOWN”, “RIGHT” 을 대답하고 그 대답을 기초로 정답이 될 수 있는 수의 개수를 리턴하는 프로그램에서 빈 곳에 알맞은 코드를 채워 넣는 문제
- 출제자가 생각하는 수의 범위를 1 ~ K 로 시작하여 참가자가 제시하는 수에 대한 출제자의 답을 보고 수의 범위를 좁혀가며 정답의 개수를 구함

3) 정답

- 주요 아이디어 정리

✓ 참가자가 제시한 수에 대한 출제자의 답이 'UP' 인 경우 추측하는 수 범위의 최소 한계값을 참가자가 제시한 수와 현재 최소 한계값 중 큰 것으로 변경

✓ 참가자가 제시한 수에 대한 출제자의 답이 'DOWN' 인 경우 추측하는 수 범위의 최대 한계값을 참가자가 제시한 수와 현재 최대 한계값 중 작은 것으로 변경

- 정답 코드

```
public int solution(int K, int[] numbers, String[] UpDown) {
    int left = 1;
    int right = K;
 ① for(int i = 0; i < numbers.length; i++){
        int num = numbers[i];
    ② if(UpDown[i].equals("UP"))
            left = Math.max(left, num + 1);
    ③ else if(UpDown[i].equals("DOWN"))
            right = Math.min(right, num - 1);
    ④ else if(UpDown[i].equals("RIGHT"))
            return 1;
    }
 ⑤ return right - left + 1;
}
```

①. for 문에서 numbers 배열의 길이만큼 반복하며 numbers 배열과 upDown 배열의 내용을 읽어 들임

②. upDown 배열의 항목 값이 "UP" 이면 최소 한계 값을 num 과 현재의 최소 한계 값 중 큰 값으로 변경

③. upDown 배열의 항목 값이 "DOWN" 이면 최대 한계 값을 num 과 현재의 최대 한계 값 중 작은 값으로 변경

④. upDown 배열의 항목 값이 "RIGHT" 이면 num 이 정답이므로 1 을 리턴

⑤. 정답이 될 수 있는 수의 개수로 (최대 한계 값 - 최소 한계 값 - 1)을 계산하여 리턴
(ex : 정답이 될 수 있는 범위가 1 보다 크고 4 보다 작은 경우 정답이 될 수 있는 수의 개수는 4 - 1 - 1 = 2 개)

8. 문제 8

1) 문제 코드

```java
/*=======================================
  6차 8번   6차 1급 8_initial_code.java
 =======================================*/

class Solution {
    final int INC = 0;
    final int DEC = 1;
    int[] func_a(int[] arr){
        int length = arr.length;
        int[] ret = new int[length];
        ret[0] = 1;
        for(int i = 1; i < length; i++){
            if(arr[i] != arr[i-1])
                ret[i] = ret[i-1] + 1;
            else
                ret[i] = 2;
        }
        return ret;
    }

    int[] func_b(int[] arr){
        int length = arr.length;
        int[] ret = new int[length];
        ret[0] = -1;
        for(int i = 1; i < length; i++){
            if(arr[i] > arr[i-1])
                ret[i] = INC;
            else if(arr[i] < arr[i-1])
                ret[i] = DEC;
        }
        return ret;
    }
    int func_c(int[] arr){
        int length = arr.length;
        int ret = 0;
        for(int i = 0; i < length; i++)
            if(ret < arr[i])
                ret = arr[i];
        if(ret == 2)
            return 0;
        return ret;
    }
```

```
    public int solution(int[] S) {
        int[] check = func_@@@(@@@);
        int[] dp = func_@@@(@@@);
        int answer = func_@@@(@@@);
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[] S1 = {2, 5, 7, 3, 4, 6, 1, 8, 9};
        int ret1 = sol.solution(S1);

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

        int[] S2 = {4, 3, 2, 1, 10, 6, 9, 7, 8};
        int ret2 = sol.solution(S2);

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

        int[] S3 = {1, 2, 3, 4, 5};
        int ret3 = sol.solution(S3);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret3 + " 입니다.");
    }
}
```

2) 문제 개요

- 문제 코드 안에 작성된 함수를 파악한 후 제시된 과제를 해결하기 위한 알고리즘대로 알맞은 함수를 호출하도록 코드를 완성하는 문제
- 주어진 수열에서 지그재그 수열로 구성된 구간의 최대 길이를 구하여 return 하기 위해 알맞은 함수와 인수를 적는 문제
- 지그재그 수열이란 첫 번째 원소부터 인접한 원소의 차이가 증가 → 감소 → 증가 → 감소 … 혹은 감소 → 증가 → 감소 → 증가 … 순으로 나타나는 수열을 뜻함

3) 정답

- 주요 아이디어 정리

✓ 주어진 배열의 항목을 이전 항목과 비교하여 증가/감소했는지 나타내는 배열을 생성

✓ 생성한 배열에서 각 항목별로 증가/감소 상태를 번갈아 나타내는 길이를 계산하여 또 다른 배열에 저장해서 활용

- 정답 코드

```
final int INC = 0;
final int DEC = 1;
```

```
int[] func_a(int[] arr){
    int length = arr.length;
  ① int[] ret = new int[length];
    ret[0] = 1;
    for(int i = 1; i < length; i++){
      ② if(arr[i] != arr[i-1])
            ret[i] = ret[i-1] + 1;
      ③ else
            ret[i] = 2;
    }
    return ret;
}
```

❖ func_a( ) : 배열의 각 항목을 마지막으로 하는 지그재그 수열 중 가장 긴 구간의 길이를 각 항목별로 구하여 배열로 저장해서 리턴 하는 함수

①. 매개변수 arr 과 길이가 동일한 배열 ret 를 생성하고 첫 번째 항목 값을 1 로 초기화

②. arr 의 i 번 인덱스 항목을 그 이전 항목과 비교했을 때 같지 않으면

- arr 배열의 항목들은 INC(증가) 혹은 DEC(감소)를 값으로 가짐
- 현재 항목과 이전 항목의 값이 다르다는 것은 증가와 감소가 번갈아 나타나는 것을 의미
- 이런 경우는 지그재그 수열이 되는 구간이므로 ret[i - 1] + 1 을 ret[i]의 값으로 저장(배열의 이전 항목을 이용하는 다이나믹 프로그래밍 방법)

③. arr 의 i 번 인덱스 항목을 그 이전 항목과 비교했을 때 같으면 ret[i]의 값을 2 로 저장

- 만일 arr 에서 이전 항목과 현재 항목이 모두 INC(증가)를 갖는 경우 다음 항목이 DEC(감소)가 오게 되면 다음 항목에 대한 지그재그 수열 구간의 길이가 3 으로 구해져야 하기 때문

  ex) S = [1, 2, 5, 3] 인 경우

  증감을 나타내는 배열 arr = [-1, INC, INC, DEC]으로 집계됨

  arr[1] == arr[2] == INC 이고, arr[3] == DEC 이므로

  ret[2] = 2, ret[3] = 3 이 되어 지그재그 수열 구간인 [2, 5, 3]에 대한 길이 3 을 구하게 됨

```
int[] func_b(int[] arr){
    int length = arr.length;
  ④ int[] ret = new int[length];
    ret[0] = -1;
    for(int i = 1; i < length; i++){
      ⑤ if(arr[i] > arr[i-1])
            ret[i] = INC;
      ⑥ else if(arr[i] < arr[i-1])
            ret[i] = DEC;
    }
    return ret;
}
```

❖ func_b( ) : 배열에 저장되어 있는 각 항목의 숫자가 이전 항목보다 증가/감소했는지 나타내는 배열을 생성하여 리턴 하는 함수

④. arr 길이와 동일한 길이의 배열 ret 를 생성하고 첫 번째 항목 값을 -1 로 초기화
⑤. arr[i]이 이전 항목보다 크면 ret[i]에 INC 에 저장되어 있는 값을 할당
⑥. arr[i]이 이전 항목보다 작으면 ret[i]에 DEC 에 저장되어 있는 값을 할당

```
int func_c(int[] arr){
    int length = arr.length;
    int ret = 0;
  ⑦for(int i = 0; i < length; i++)
        if(ret < arr[i])
            ret = arr[i];
  ⑧if(ret == 2)
        return 0;
    return ret;
}
```

❖ func_c( ) : 배열에 저장되어 지그재그 구간의 길이 중 가장 큰 값을 리턴 하는 함수
⑦. arr 에 저장되어 있는 항목 값 중 최댓값을 구하여 ret 에 저장
⑧. ret 에 저장된 값이 2 이면 0 을 리턴

```
public int solution(int[] S) {
  ⑨ int[] check = func_b(S);
  ⑩ int[] dp = func_a(check);
  ⑪ int answer = func_c(dp);
    return answer;
}
```

❖ solution( )
⑨. func_b( )를 이용하여 S 에 저장되어 있는 항목들을 이전 항목과 비교한 결과를 check 에 저장
⑩. func_a( )를 이용하여 check 에 저장되어 있는 증감 현황에 대해서 각 항목별 지그재그 수열의 최대 길이를 구하여 dp 에 저장
⑪. func_c( )를 이용하여 dp 에 저장되어 있는 최댓값을 구함

9. 문제 9

1) 문제 코드

```
/*=====================================
   6차 9번   6차 1급 9_initial_code.java
 =====================================*/
import java.util.*;
import java.util.ArrayList;

class Solution {
    Integer func_a(ArrayList<Integer> stack) {
        Integer item = stack.remove(stack.size() - 1);
        return item;
    }

    void func_b(ArrayList<Integer> stack1, ArrayList<Integer> stack2) {
        while(!func_@@@(@@@)) {
            Integer item = func_@@@(@@@);
            stack2.add(item);
        }
    }

    boolean func_c(ArrayList<Integer> stack) {
        return (stack.size() == 0);
    }
```

```
    public int solution(ArrayList<Integer> stack1, ArrayList<Integer> stack2) {
        if(func_@@@(@@@)) {
            func_@@@(@@@);
        }
        Integer answer = (int)func_@@@(@@@);
        return answer;
    }

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

        ArrayList<Integer> stack1_1 = new ArrayList<Integer>();
        ArrayList<Integer> stack2_1 = new ArrayList<Integer>();
        stack1_1.add(1);
        stack1_1.add(2);
        stack2_1.add(3);
        stack2_1.add(4);
        int ret1 = sol.solution(stack1_1, stack2_1);
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");


        ArrayList<Integer> stack1_2 = new ArrayList<Integer>();
        ArrayList<Integer> stack2_2 = new ArrayList<Integer>();
        stack1_2.add(1);
        stack1_2.add(2);
        stack1_2.add(3);
        int ret2 = sol.solution(stack1_2, stack2_2);
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
}
```

2) 문제 개요

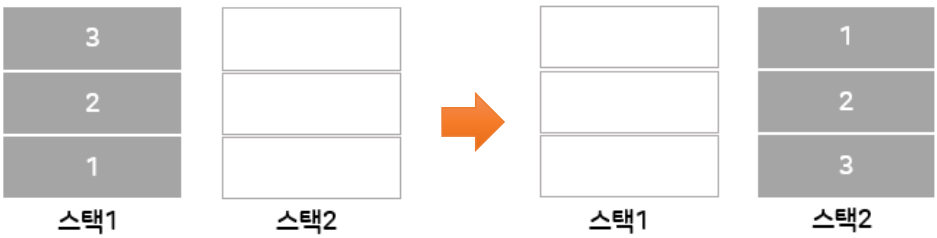
- 문제 코드 안에 작성된 함수를 파악한 후 제시된 과제를 해결하기 위한 알고리즘 대로 알맞은 함수를 호출하도록 코드를 완성하는 문제
- LIFO 방식으로 자료를 관리하는 스택(Stack) 두 개를 이용하여 FIFO 방식으로 자료를 관리하는 큐(Queue)의 pop 함수를 구현하는 프로그램
- 배열 두 개를 두 개의 스택으로 사용하여 Queue 의 pop( ) 함수와 같이 작동하도록 알맞은 함수와 인수를 적는 문제

3) 정답

- 주요 아이디어 정리

✓ 스택 1 과 스택 2 가 있는데, 스택 2 가 비어 있으면 스택 1 의 모든 항목을 pop( ) 하여 스택 2 로 push( )를 수행한 후에 스택 2 에서 pop( ) 을 실행

✓ 스택 2 가 이미 값을 가지고 있다면, 이미 스택 1 에 있는 값을 스택 2 로 옮긴 것이므로 스택 2 에서 pop( ) 을 수행



➔ 스택 1 의 모든 항목을 pop( )해서 스택 2 로 push( ) 를 하면 스택 2 에 역순으로 저장

- 정답 코드

```
4 Integer func_a(ArrayList<Integer> stack) {
      Integer item = stack.remove(stack.size() - 1);
      return item;
  }

3 void func_b(ArrayList<Integer> stack1, ArrayList<Integer> stack2) {
      while(!func_@@@(@@@)) {   c(stack1)
          Integer item = func_@@@(@@@);   a(stack1)
          stack2.add(item);
      }
  }

2 boolean func_c(ArrayList<Integer> stack) {
      return (stack.size() == 0);
  }

  public int solution(ArrayList<Integer> stack1, ArrayList<Integer> stack2) {
      if(func_@@@(@@@)) {   c(stack2)
1         func_@@@(@@@);   b(stack1, stack2)
      }
      Integer answer = (int)func_@@@(@@@);   a(stack2)
      return answer;
  }
```

❖ func_a( ) : pop(). stack 에서 마지막 항목 값을 리턴

④. 매개 변수 stack 에 대해서 stack 에 저장된 마지막 항목 값을 가져오고, 마지막 항목 값을 stack 에서 지우고 삭제한 마지막 항목을 리턴

❖ func_b( ) : stack1 에 있는 항목을 모두 pop( )해서 stack2 배열에 push( )

③. - func_c( )를 이용하여 stack1 의 길이가 0 인지 확인하여 0 이 아닌 동안 실행 stack1 이 빈 상태가 아닌 동안 실행
- func_a( )를 이용하여 stack1 에서 pop( )한 값을 item 에 할당
- item 에 있는 값을 stack2 에 push( ). stack2 의 마지막 항목으로 추가

❖ func_c( ) : stack 의 인덱스가 -1 인지 확인

②. stack 의 길이가 0 과 같은지 논리연산을 수행하여 그 결과를 리턴. stack 이 빈 상태(empty)임을 의미

❖ solution( )

①. - func_c( )를 이용하여 stack2 가 비어 있는 상태인지 확인
- func_b( )를 이용하여 stack1 에 있는 모든 값을 pop 해서 stack2 에 순차적으로 push. stack2 에 역순으로 추가됨
- func_a( )를 이용하여 stack2 에서 pop( ). 먼저 push 한 값이 먼저 pop 되는 효과를 얻어 queue 효과와 같음

10. 문제 10

1) 문제 코드

5 차 10 번 문제와 동일

```
/*=======================================
   5차 10번   5차 1급 10_initial_code.java
 =======================================*/
class Solution {
    class Job {
        public int salary;

        public Job() {
            this.salary = 0;
        }
        public int getSalary() {
            return salary;
        }
    }

    class PartTimeJob @@@ {
        public int workHour, payPerHour;

        public PartTimeJob(int workHour, int payPerHour) {
            this.workHour = workHour;
            this.payPerHour = payPerHour;
        }
        @@@ {
            salary = workHour * payPerHour;
            if(workHour >= 40)
                salary += (payPerHour * 8);

            return salary;
        }
    }

    class SalesJob @@@ {
        public int salesResult, payPerSale;

        public SalesJob(int salesResult, int payPerSale) {
            this.salesResult = salesResult;
            this.payPerSale = payPerSale;
        }
        @@@ {
            if(salesResult > 20)
                salary = salesResult * payPerSale * 3;
            else if(salesResult > 10)
                salary = salesResult * payPerSale * 2;
            else
                salary = salesResult * payPerSale;

            return salary;
        }
    }
```

```
    public int solution(int[][] partTimeJobs, int[][] salesJobs) {
        int answer = 0;

        for(int i = 0; i < partTimeJobs.length; i++) {
            PartTimeJob partTimeJob = new PartTimeJob(partTimeJobs[i][0], partTimeJobs[i][1]);
            answer += partTimeJob.getSalary();
        }

        for(int i = 0; i < salesJobs.length; i++) {
            SalesJob salesJob = new SalesJob(salesJobs[i][0], salesJobs[i][1]);
            answer += salesJob.getSalary();
        }

        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[][] partTimeJobs = {{10, 5000}, {43, 6800}, {5, 12800}};
        int[][] salesJobs = {{3, 18000}, {12, 8500}};
        int ret = sol.solution(partTimeJobs, salesJobs);

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

2) 문제 개요

- Job 클래스를 상속받도록 PartTimeJob 클래스와 SalesJob 클래스를 정의하는 문제
- PartTimeJob 클래스와 SalesJob 클래스의 부모 클래스를 지정하고, 각각의 자식 클래스에서 부모 클래스인 Job 의 getSalary( ) 메소드를 재정의하여 문제에서 제시된 대로 급여를 계산하도록 메소드 정의 부분의 빈 칸을 채워야 함

3) 정답

```
class Job {
    public int salary;

    public Job() {
        this.salary = 0;
    }

    public int getSalary() {
        return salary;
    }
}
```

❖ Job 클래스 정의

① 생성자 메소드를 이용하여 멤버 변수 salary 를 생성하고 0 으로 초기화

② getSalary( ) 메소드는 멤버 변수 salary 에 저장되어 있는 값을 리턴

```
class PartTimeJob extends Job {
    public int workHour, payPerHour;

    public PartTimeJob(int workHour, int payPerHour) {
        this.workHour = workHour;
        this.payPerHour = payPerHour;
    }

    public int getSalary() {
        salary = workHour * payPerHour;
        if(workHour >= 40)
            salary += (payPerHour * 8);

        return salary;
    }
}
```

❖ PartTimeJob 클래스 정의

③. PartTimeJob 클래스가 Job 클래스를 상속받도록 부모 클래스를 Job 으로 지정

④. 부모 클래스의 생성자 메소드를 실행하여 멤버 변수 salary 를 생성하고 0 으로 초기화

⑤. 매개 변수로 받은 값을 이용하여 멤버 변수 workHour 와 멤버 변수 payPerHour 를 생성하고 초기화

⑥. 부모 클래스 Job 에 있는 getSalary( ) 메소드를 오버라이드

⑦. 멤버 변수 workHour * payPerHour 를 계산한 값을 멤버 변수 salary 로 저장

⑧. 멤버 변수 workHour >= 40 이면 멤버 변수 payPerHour * 8 을 계산한 것을 멤버 변수 salary 에 추가로 더함

```
class SalesJob extends Job {
    public int salesResult, payPerSale;

    public SalesJob(int salesResult, int payPerSale) {
        this.salesResult = salesResult;
        this.payPerSale = payPerSale;
    }

    public int getSalary() {
        if(salesResult > 20)
            salary = salesResult * payPerSale * 3;
        else if(salesResult > 10)
            salary = salesResult * payPerSale * 2;
        else
            salary = salesResult * payPerSale;

        return salary;
    }
}
```

❖ SalesJob 클래스 정의

⑨. SalesJob 클래스가 Job 클래스를 상속받도록 부모 클래스를 Job 으로 지정

⑩. 부모 클래스의 생성자 메소드를 실행하여 멤버 변수 salary 를 생성하고 0 으로 초기화

⑪. 매개 변수로 받은 값을 이용하여 멤버 변수 salesResult 와 멤버 변수 payPerSale 를 생성하고 초기화

⑫. 부모 클래스 Job 에 있는 getSalary( ) 메소드를 오버라이드