2023.01.13 스터디 발표

❄️ 내가 푼 문제

- LeetCode 733. Flood Fill

❄️ 방법

```
class Solution {
    public int[][] floodFill(int[][] image, int sr, int sc, int newColor) {
        int color = image[sr][sc];
        if (color != newColor) dfs(image, sr, sc, color, newColor);
        return image;
    }
    public void dfs(int[][] image, int r, int c, int color, int newColor) {
        if (image[r][c] == color) {
            image[r][c] = newColor;
            if (r >= 1) dfs(image, r - 1, c, color, newColor);
            if (c >= 1) dfs(image, r, c - 1, color, newColor);
            if (r + 1 < image.length) dfs(image, r + 1, c, color, newColor);
            if (c + 1 < image[0].length) dfs(image, r, c + 1, color, newColor);
        }
    }
}
```

❄️ 공부한 것

- `Projections.bean`
- `Projection.fields`
- `Projection.constructor`
- `@QueryProjection`

❓Projection이란?

- `Querydsl`를 이용하는 경우 엔티티와 다른 반환 타입인 경우 `Projections`를 사용
- 예를 들면, DTO 로 반환 타입을 가지는 경우

1️⃣ `Projections.bean`

- `setter` 메서드를 기반으로 동작
- 때문에 DTO 객체의 각 필드에 `setter` 메서드가 필요
- 일반적으로 응답과 요청 객체는 불변 객체를 지향하는 것이 바람직한 편

2️⃣ `Projection.fields`

- `getter`, `setter` 메서드 필요 없이 필드에 값을 직접 주입해주는 방식
- `Projections.bean` 방식과 마찬가지로 타입이 다를 경우 매칭이 되지 않음
- 이름이 다르면 별칭을 지정해주면 됨
- 컴파일 시점에서는 에러를 잡지 못하고 런타임 시점에서 에러 잡힘

3️⃣ `Projections.constructor`

- 생성자 기반 바인딩
- 객체의 불변성을 가지는 장점
- 바인딩 과정에서 문제 발생
  - DTO 객체의 생성자에게 직접 바인딩하는 것이 아니고 `Expression <?>... exprs` 값을 넘기는 방식
  - 값을 넘길 때 생성자의 순서와 일치시켜야 함
  - 값이 많아지는 경우 실수할 수 있는 문제가 발생할 수 있는 확률이 높아 권장하지 않음

4️⃣ `@QueryProjection`

- 불변 객체 선언, 생성자 그대로 사용을 할 수 있어 권장하는 패턴
- DTO 생성자를 쓰는 것이 아니라, DTO를 기반으로 생성된 QDTO를 사용하는 방식
- `Dto`라는 특성상 해당 객체는 많은 계층에서 사용하는 데, Querydsl의 의존성이 필요 없는 계층에서도 해당 의존성이 필요하게 되는 단점
- 런타임 시점이 아닌, 컴파일 시점에서 에러 잡힘