1.개요

주차	10주차	날짜	11.24
주제	Recursive SpringBoot Test SpringBoot Data JPA		
목표			
실습	1. Controller Mock Test만들기 2. Service Mock Test만들기		
Links	java-algorithm Kyeongrok/java-algorithm (github.com) 코딩테스트 고득점 Kit	프로그래머스 스쿨 (programmers.co.kr)	

1.1. 재귀로 피보나치 수열 만들기

이번에는 for문과 같은 반복문이 아닌 재귀를 통해 피보나치 수열을 만들어보겠습니다. 앞에서는 머릿속에서 생각나는 대로 피보나치 수열을 구현 해보았습니다. 1, 1을 만들어서 먼저 넣고 2, 3, 5, 8은 앞에 두개를 더해서 구했습니다.

소스코드가 아주 길지는 않지만 재귀로 풀었을때에 비해 꽤 깁니다. 재귀로 짜면 3줄이면 됩니다. 재귀로 짜면 구현 할때 생각을 반대로 해야 합니다. 앞에서는 1, 1, 2, 3, 5 이렇게 순서대로 만들었지만 재귀는 역산을 하는 방식을 이용합니다.

이를테면 8이 나올려면 5, 3이 있어야 하고 5가 나올려면 3과 2가 있어야 하고 3이 나올려면 2와 1이 있어야 합니다. 8이 나올려면 3과 5가 있어야 한다고 표현을 하지 않고 5와 3이 있어야 한다고 썼습니다. 생각을 완전히 반대로 해야지 구현할 때 덜 헷갈리기 때문에 우리가 익숙한 오름차순이 아니고 내림차순으로 생각을 바꿔서 해보시는 것을 권해드립니다.

그러면 fib(6)은 8이라는 것을 우리가 알고 있습니다. 재귀는 자기 자신을 호출해서 어떤 결과를 만들어 내야 합니다.

```
fib(1) = 1
fib(2) = 1      fib(1)
fib(3) = 2      fib(2) + fib(1)
fib(4) = 3      fib(3) + fib(2)
fib(5) = 5      fib(4) + fib(3)
fib(6) = 8      fib(5) + fib(4)
```

앞에서 재귀를 사용하지 않고 만들었던 fib()는 위 처럼 1, 1, 2 이렇게 순서대로 구했지만 재귀는

재귀로 피보나치 수열의 n번째 값을 구하고 싶어도 예를들어 fib(6)을 구하고 싶다면 fib(5)와 fib(4)를 알아야 합니다.

fib(6)은 fib(5) + fib(4)입니다. 즉 fib(6)은 5 + 3의 결과인 8 입니다.

이 식을 소스코드로 만들어 보겠습니다.

```
def fib(n):
  return 5 + 3

print(fib(6))
```

결과

```
8
```

결과 해석
피보나치 수열 값이 들어있는 리스트 [1, 1, 2, 3, 5, 8]에서 6번째에 있는 값은 8입니다.

1.1.1. return에서 재귀 호출

앞에서는 return에서 다시 fib()을 호출 하지 않았으니 재귀 함수가 아닙니다. 재귀로 피보나치수열을 구현 해보기로 했기 때문에 재귀 로직을 구성 해보겠습니다.

return 5 + 3 에서 5는 fib(5)이고 3은 fib(4)입니다. 그래서 return fib(5) + fib(4)로 바꾸고 실행 해보겠습니다.

```
public class Fibonacci3 {
   public static int fib(int n) {
       return fib(n - 1) + fib(n - 2);
   }
   public static void main(String[] args) {
       int r = fib(6);
       System.out.println(r);
   }
}
```

결과

```
Traceback (most recent call last):
--- 중략 ---
    return fib(5) + fib(4)
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```

결과 해석
역시나 재귀가 끝나는 조건을 만들지 않았기 때문에 maximum recursion depth exceeded 에러가 났습니다.

재귀는 자신을 계속 호출하게끔 되어 있으므로 재귀의 깊이가 깊어지는 문제 뿐만 아니라 무한하게 반복될 수 있고 컴퓨터가 다운이 될 수도 있습니다. 그래서 재귀가 끝나는 조건을 꼭 넣고 나서 재귀 호출을 해야 합니다.

파이썬에서는 재귀가 무한히 반복되는 것을 막기 위해 리밋이 설정 되어 있습니다. 위 로직 같은 경우는 996번 이상 호출 되었다고 에러가 났습니다. 리밋은 늘릴 수 있지만 재귀가 너무 깊어질 것 같으면 다른 방법을 생각해 보는 것이 좋습니다. 피보나치 수열 같은 경우는 앞에서 재귀를 쓰지 않고 구했던 방법도 있고 뒤에서 배울 다이내믹 프로그래밍 방법도 재귀가 너무 많이 쌓이지 않으면서도 속도도 잘 나오는 방법이 있습니다.

하지만 이번장은 재귀에 대한 장이기 때문에 재귀에 대한 내용만 다루겠습니다.

1.1.2. 피보나치 재귀의 핵심 로직 추가

아직 재귀로 핵심 로직을 구현하기 전이지만 피보나치 수열이 1, 1, 2, 3, 5 이렇게 진행 될 때 fib(1)의 실행 결과는 1이고 fib(2)도 1, fib(3)은 2 이렇게 n번째 값을 리턴하는 함수로 디자인을 했습니다.

n	1	2	3	4	5	6
fib(n)	1	1	2	3	5	8

피보나치 수열은 앞에 두개의 값을 더한 것이 그 다음 값이 되는 수열입니다. 그래서 fib(1) + fib(2)는 fib(3)이 됩니다. fib(2) + fib(3)은 fib(4)가 되는 구조 입니다.

fib(2) + fib(3) = fib(4)를 다시 써보면 fib(4) = fib(3) + fib(2)가 되는 것입니다.

하지만 함수에 전달되는 값은 fib(4), fib(5)처럼 n번째에 해당하는 숫자가 전달되기 때문에 fib(n)의 값은 fib(n-1)과 fib(n-2)을 알기 전에는 구할 수 없습니다.

여기에서 핵심 로직이 나옵니다. fib(4) = fib(3) + fib(2) 이 식을 n을 이용해 표현을 해보겠습니다. 여기에서 n은 4 입니다.

```
n = 4
fib(n) = fib(n - 1) + fib(n - 2)
```

n이 4라면 fib(n)은 fib(4)이고 fib(n - 1)은 fib(3), fib(n - 2)는 fib(2)입니다.

이런식으로 n을 1씩, 2씩 줄이면서 fib(2) + fib(1)까지 내려가야 1 + 1이 시작됩니다.

피보나치수열을 재귀로 구현 할 때는 거꾸로 쌓아올리면서 구현한다고 반대로 생각을 해볼 필요가 있습니다.

피보나치 수열은 1, 1, 2, 3, 5 이렇게 반복이 됩니다만 재귀는 5, 3, 2, 1, 1 이렇게 올라오는 방식으로 답을 구하게 됩니다.

재귀는 반복되는 연산이 앞에서 계산한 값에 영향을 미쳐야 하는 경우 주로 사용하게 됩니다. 그래서 피보나치수열이 재귀에 대해 알아보기 좋은 예제가 됩니다.

fib(4)는 피보나치 수열의 네번째 값을 리턴한다고 했을 때 1, 1, 2, 3에서 3을 리턴 해야 합니다.

```
result = fib(4)
print(result)
```

위 코드를 실행 했을때 3을 리턴 해야 하는 것입니다.

3은 2 + 1한 결과 입니다. 2는 fib(3)이고 1은 fib(2)입니다.

2는 1 + 1한 결과 입니다. 피보나치 수열이 1, 1로 시작 하기 때문에 fib(1), fib(2)각각도 1입니다.

식으로 표현을 해보면 아래와 같이 fib(4)는 fib(3)과 fib(2)의 합입니다. 코드로는 아래와 같이 쓸 수 있습니다.

```
result = fib(3) + fib(2)
```

fib(3)	fib(2) + fib(1)
fib(2)	fib(1) + fib(0)

각각 fib(3) fib(2)도 풀어서 써보겠습니다. 풀어쓰면 아래와 같이 됩니다.

```
result = fib(2) + fib(1) + fib(1) + fib(0)
```

fib(1)은 1, 1, 2, 3에서 1이라는 것을 알지만 fib(0)이 나왔습니다. 이 부분도 식에 포함이 되어 있기 때문에 처리 해주어야 합니다.

n	1	2	3	4	5	6
fib(n)	1	1	2	3	5	8

앞에서 위와 같이 1, 1, 2, 3으로 이어지는 피보나치 수열을 가지고 예제를 진행해보고 있습니다. 피보나치 수열이 '앞에 두 수를 더한 값이 그 다음 값이 되는 수열'이라고 하면 fib(0)을 0으로 한다면 이 조건에도 맞습니다.

n	0	1	2	3	4	5	6
fib(n)	0	1	1	2	3	5	8

우리에게 익숙한 것은 1, 1, 2, 3, 5, 8이지만 피보나치 수열을 0, 1, 1, 2, 3, 5, 8 이렇게 맨 앞에 0으로 시작을 해도 문제가 없습니다.

그러면 앞에 나왔던 식을 한번 더 풀어서 써보겠습니다.

```
result = fib(2) + fib(1) + fib(1) + fib(0)
```

fib(2) = fib(1) + fib(0)으로 풀어서 쓸수 있습니다.

```
result = fib(1) + fib(0) + fib(1) + fib(1) + fib(0)
```

이제는 더 풀어서 쓸 것이 없습니다. 위표를 참고하여 식을 완성 해보겠습니다.

```
result = 1 + 0 + 1 + 1 + 0
```

fib(4)는 fib(1) + fib(0) + fib(1) + fib(1) + fib(0) 로 풀어지고 fib(1)은 1이고 fib(0)은 0이기 때문에 fib(4)는 3이 됩니다. 재귀로 푸는 피보나치 수열은 fib()로 전달하는 n의 크기가 점점 줄어들어서 1부터 리턴하기 시작하여 답을 완성하게 됩니다.

핵심 로직인 fib(n - 1) + fib(n - 2)를 적용해서 재귀를 구성 해보겠습니다.

```
def fib(n):
  return fib(n - 1) + fib(n - 2)

print(fib(4))
```

생각보다 너무 간단한 로직이 나와서 더 어렵습니다.

위 로직을 실행 하면 역시나 에러가 날 것입니다.

결과

```
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```

결과 해석
앞에서 났던 maximum recursion depth exceeded 에러가 났습니다. 역시나 끝나는 조건이 없기 때문입니다. 여기에 재귀가 끝나는 탈출 조건을 넣어서 로직을 완성 해보겠습니다.

1.1.3. 탈출 조건 만들기

앞에서 탈출조건을 만들어주지 않아서 재귀가 계속 반복되다가 리밋에 가까워지자 에러가 났습니다. 재귀는 반드시 탈출조건을 만들어야 합니다.

피보나치 수열에서 탈출조건은 n <= 1일때 리턴을 해주는 것입니다. 왜냐하면 fib(6) = fib(5) + fib(4)인데 이 호출이 계속 반복되어 fib(1), fib(0)까지 가기 때문입니다. 재귀는 언젠가는 끝나야 합니다.

리턴을 해줄 때 return 1을 해주는 것이 아니고 return n을 해줍니다.

```
public class Fibonacci3 {
    public static int fib(int n) {
        if(n == 1 | n == 2 ) return 1; // 1일때 1 2일때 1
        return fib(n - 1) + fib(n - 2);
    }
    public static void main(String[] args) {
        int r = fib(6);
        System.out.println(r);
    }
}
```

결과

```
8
```

결과 해석
result = fib(6)에서 fib(6)은 fib(5) + fib(4)를 리턴 하기 때문에 result = fib(5) + fib(4)가 됩니다.

fib(6)은 fib(5) + fib(4)를 리턴 합니다.

```
fib(6) = fib(5) + fib(4)
```

fib(5) + fib(4)에서 **fib(5)**는 fib(4) + fib(3)입니다. 각 fib(4), fib(3)은 아래와 같이 fib(1), fib(0)이 나올때까지 n을 하나씩 줄이면서 계속 fib()함수를 호출 합니다.

```
fib(5) = fib(4) + fib(3)
        fib(4) = fib(3) + fib(2)
                fib(3) = fib(2) + fib(1)
                        fib(2) = fib(1) + fib(0)
                                fib(1) = 1
                                fib(0) = 0
                        fib(1) = 1
                fib(2) = fib(1) + fib(0)
                        fib(1) = 1
                        fib(0) = 0

        fib(3) = fib(2) + fib(1)
                fib(2) = fib(1) + fib(0)
                        fib(1) = 1
                        fib(0) = 0
                fib(1) = 1
```

fib(5) + fib(4)에서 **fib(4)**는 fib(3) + fib(2)입니다. 여기에서 fib(3), fib(2)등을 구하는 연산이 반복되어 나오는 것을 볼 수 있습니다. 이것이 재귀의 문제입니다. 이미 결과가 나온 연산을 또 하게 되는 문제를 재귀는 가지고 있습니다. 이것은 뒤에서 다룰 다이내믹 프로그래밍에서 해결할 수 있습니다.

```
fib(4) =  fib(3) + fib(2)
        fib(3) = fib(2) + fib(1)
                fib(2) = fib(1) + 0
                        fib(1) = 1
                fib(1) = 1
        fib(2) = fib(1) + fib(0)
                fib(1) = 1
                fib(0) = 0
```

위 로직에서 재귀를 타지 않는 경우는 n이 1인 경우, 0인 경우 두가지 뿐입니다.

```
result = fib(2)
```

```
if 2 <= 1:
    return n
return fib(2 - 1) + fib(2 - 2)
```

n이 2일 때 fib(1) + fib(0)을 리턴하게 됩니다. 하지만 fib(1), fib(0)은 위와 같이 if조건을 타기 때문에 1인 경우는 1을 리턴하고 더이상 재귀 호출을 하지 않습니다. 0인 경우도 마찬가지 0을 리턴하고 재귀를 더이상 호출하지 않습니다.

n의 개수를 계속 줄여가면서 1과 0이 나올때까지 내려가다가 재귀를 호출하지 않게되는 1, 0까지 내려간 후 1 + 1부터 시작해서 1 + 2, 2 + 3, 3 + 5를 차례로 계산하여 결과를 리턴하는 구조입니다.

실습
코드업 fib(n)
https://codeup.kr/problem.php?id=1855

fibonacci 수열 재귀 9:21까지

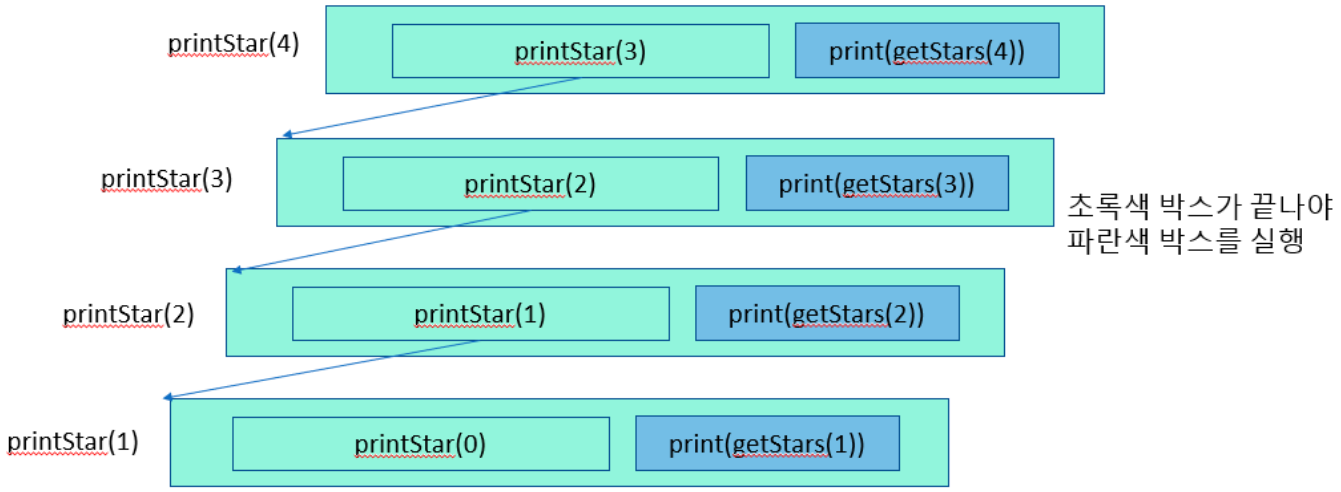
별 찍기
https://codeup.kr/problem.php?id=1859

```
public static String getStars(int n) {
    if(n == 0) return "";
    return "*" + getStars(n - 1);
}
```

```
public class Codeup1859 {
    public static String getStars(int n) {
        // 별을 만든다
        if(n == 0) return "";
        return "*" + getStars(n - 1);
    }
    public static void printStar(int n) { // 별을 출력한다
        if(n == 0) return;
        // n은 작아지지만
        // 출력은 1부터 n까지
        printStar(n - 1);
        System.out.println();
    }
    public static void main(String[] args) {
        System.out.println(getStars(4));
    }
}
```

```java
if(n == 0) return;
// n은 작아지지만
// 출력은 1부터 n까지
printStar(n - 1);
System.out.println(getStars(n));
```

printStar(4)
printStar(3) -> print(별만들기(4))
printStar(2) -> print(별만들기(3))
printStar(1) -> print(별만들기(2))
printStar(0) -> print(별만들기(1))



10:25까지

숫자 삼각형
https://codeup.kr/problem.php?id=1860

```java
public static void printNums(int n) {
    if (n == 0) return;
    printNums(n - 1); // 먼저 작은 숫자를 호출해서 스택에 넣어줍니다.
    System.out.println("n까지 숫자를 출력");
}
```

호출한 역순으로 출력이 되도록 제어하는 것

통과한 코드

```java
public static String getNums(int n) {
```

```
    // n까지의 숫자를 만든다
    if(n == 1) return "1";
    return getNums(n - 1) + " " + n;  // 더 작은 숫자가 먼저 나오게 처리 함
}
```

통과 못한 경우

```
public static String getNums(int n) {
        if(n == 0) return "";
        return getNums(n - 1) + " " + n;
    }
```

여기에서는 n ==0 “”공백 리턴하는데 “ “ space가 들어가므로
1
1 2
1 2 3

그래서 앞에 “ “가 들어가지 않도록 1에서 끝냄

10:50까지

두 수 사이의 홀수
https://codeup.kr/problem.php?id=1904

```
public static void printNum(int a, int b) {
    if(a > b) return;
    printNum(a, b - 1);
    if(b % 2 != 0) System.out.printf("%d ", b);
}
```

[도전]
codeup에서 '재귀'로 검색해서 나오는 문제 위에서 n개 풀고 명렬표에 개수 적기

OneToMany, ManyToOne 양방향 매핑

```java
public class Hospital {
    @Id
    private Integer id;
    private String roadNameAddress;
    private String hospitalName;

    @OneToMany(mappedBy = "hospital", fetch = FetchType.LAZY)
    private List<Review> reviews;
```

```java
public class Review {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
```

```
    private String content;
    private String userName;

    @ManyToOne
    @JoinColumn(name = "hospital_id")
    private Hospital hospital;
```

Hibernate: alter table hospital add column hospital_name varchar(255)
Hibernate: alter table review add constraint FK49momwbwjdvu64cevtop3n4va foreign key (hospital_id) references hospital (id)

실습

리뷰 등록 기능

POST /api/v1/hospitals/{id}/reviews

> ⌘B 또는 Ctrl+B
>
> 코드 안에서 다른 코드로 이동하고 싶을 때가 자주 있습니다. ⌘B(MacOS) 또는 Ctrl+B(Windows/Linux)를 사용하면 심볼의 선언으로 이동할 수 있습니다. 예를 들어, 필드 위에서 이 단축키를 누르면 커서가 해당 필드의 선언으로 이동합니다. 클래스 이름 위에서 누르면 해당 클래스 파일로 이동합니다. ⌥⌘B(MacOS) 또는 Ctrl+Alt+B(Windows/Linux)를 누르면 구현으로 이동할 수 있습니다.

ReviewCreateRequest.java

```
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
public class ReviewCreateRequest {
    private Integer hospitalId;
    private String title;
    private String content;
    private String userName;
}
```

ReviewCreateResponse.java

```
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
public class ReviewCreateResponse {
    private Long id;
    private String title;
    private String content;
    private String userName;
    private String message;
}
```

ReviewService

.add(ReviewCreateRequest)기능
1.hospitalId를 이용해 hospital을 조회 해서 ManyToOne으로 매핑 되어 있는 Review에 hospital을 넘겨 Review오브젝트를 만드는 기능
2.만든 Review를 저장하는 기능
3.저장이 완료 되면 message에 "리뷰 등록이 성공했습니다." 메세지 작성 기능

```
@Service
public class ReviewService {

    private final ReviewRepository reviewRepository;
    private final HospitalRepository hospitalRepository;

    public ReviewService(ReviewRepository reviewRepository,
HospitalRepository hospitalRepository) {
        this.reviewRepository = reviewRepository;
        this.hospitalRepository = hospitalRepository;
    }

    public ReviewCreateResponse add(ReviewCreateRequest reviewCreateRequest)
{
        Optional<Hospital> hospital =
hospitalRepository.findById(reviewCreateRequest.getHospitalId());
        Review review = Review.builder()
```

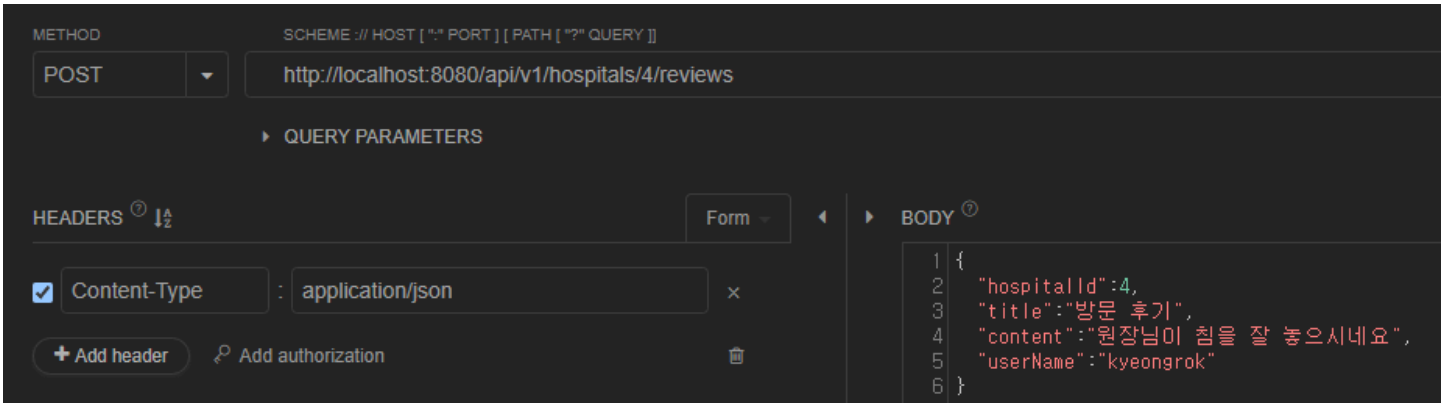
```
                .title(reviewCreateRequest.getTitle())
                .content(reviewCreateRequest.getContent())
                .userName(reviewCreateRequest.getUserName())
                .hospital(hospital.get())
                .build();
        Review savedReview = reviewRepository.save(review);
        return new ReviewCreateResponse(savedReview.getId(),
savedReview.getTitle(), savedReview.getContent(), savedReview.getContent(),
                "리뷰 등록이 성공 했습니다.");
    }
}
```

HospitalController.java

```
@PostMapping("/{id}/reviews")
public ResponseEntity<ReviewCreateResponse> get(@PathVariable Integer id,
@RequestBody ReviewCreateRequest reviewCreateRequest) {
    return ResponseEntity.ok().body(reviewService.add(reviewCreateRequest));
}
```

HospitalRepository.java

```
public interface HospitalRepository extends
JpaRepository<Hospital, Integer> {
}
```



```
{
  "id": 1,
  "title": "방문 후기",
  "content": "원장님이 침을 잘 놓으시네요",
  "userName": "원장님이 침을 잘 놓으시네요",
  "message": "리뷰 등록이 성공 했습니다."
}
```
lines nums copy

	id	content	title	user_name	hospital_id
1	1	원장님이 침을 잘 놓으시네요	방문 후기	kyeongrok	4

OneToMany, ManyToOne 양방향 할 때 불편한점

Hospital을 조회 할 때 review도 같이 조회 하기 위해 양방향 mapping을 했으나
Review쪽에 fk가 있기 때문에 Review를 등록 할 때 Hospital을 select해서 등록해야하는 불편함이 발생함

```
@OneToMany(mappedBy = "hospital", fetch = FetchType.LAZY)
private List<Review> reviews;
```

```
@ManyToOne
@JoinColumn(name = "hospital_id")
private Hospital hospital;
```

1:35까지 실습

병원 리뷰 사이트에서 리뷰 등록 기능 만들기

SpringBoot + MySql + JPA

다룰 내용들

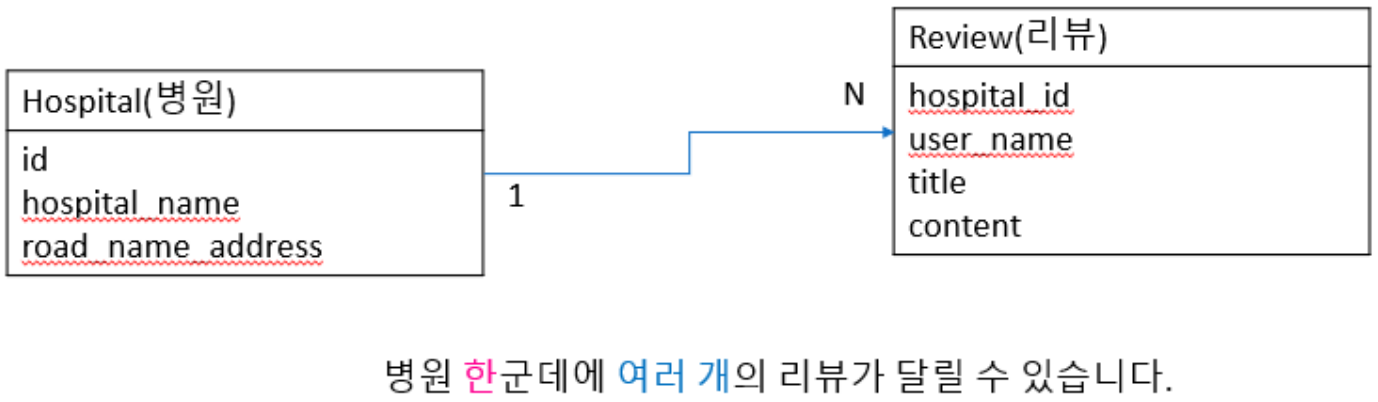
1. 요구사항
2. ERD
3. DB 새로 만들기
4. 프로젝트 새로 빌드
5. 새로 만든 DB에 연동
6. Hospital, Review OneToMany, ManyToOne연관관계 매핑
7. HospitalController에 add() POST /api/v1/hospitals/{id}/reviews
8. HospitalService에 add(HospitalCreateRequest)
9. List<Review> → List<ReviewResponse>로 전이 .stream().map(review -> ReviewResponse)사용
10. ReviewCreateResponse – message추가

요구사항

병원에 리뷰를 달 수 있는 기능을 만들어 주세요

이 한줄로 위에 10줄이 나오는 마법

ERD



병원 한군데에 여러 개의 리뷰가 달릴 수 있습니다.

3:46까지 실습

유투브 썸네일

https://docs.google.com/presentation/d/1U8l8CG8HOqsqzO7Ns-X0cMTxb5c4XfwG7amBYjnfeqA/edit#slide=id.g19803a4dd21_0_0

오늘 복습 동영상 1~5

멋쟁이 사자처럼 백엔드 - YouTube

리뷰 조회 기능

1개 조회 기능

GET /api/v1/reviews/{id}

해당 병원의 리뷰만 조회 하는 기능

GET /api/v1/hospitals/{id}/reviews

병원 정보와 함께 조회 기능

GET /api/v1/hospitals/{id}
hospital조회 했을 때 review도 보이게
id로 조회 했을때 review가 보이면 됩니다.

[도전]
/hospitals에 각 병원 리뷰 개수 추가
리뷰가 있는 병원만 조회 되는 기능

10주차	1. 댓글, 리뷰 기능 – Many To One
11주차	2. 누가 어떤 병원에 방문 했는지 Many To Many 3. 누가 어떤 병원에 방문해서 어떤 진료를 몇번 받았는지 Many To Many 4. 누가 어떤 병원에 방문해서 어떤 진료를 몇번 받았는데 진료비는 얼마인지 Aggregate
14주차	종합프로젝트
