

웹 개발 스터디

특강: Java, Gradle, Spring Framework로 구현하는 간단한 게시판 API 예제

<https://github.com/Tianea2160/spring-practice>



프로젝트 디렉토리: remind



gradle 빌드 디렉토리①

IDE 임시파일

gradle 빌드 디렉토리②

★JAVA 소스코드, 리소스 등...

빌드옵션: JDK버전, 의존성 등

DB 자동설정을 위한 docker-compose

Spring: 스프링의 기본 철학

- 모든 레벨에서 선택권을 제공한다
 - 커스텀이 대부분 가능함
- 다양한 관점을 수용한다
 - 관점지향 프로그래밍
- 이전 버전과 강력한 호환성을 유지한다
- API 설계에 관심을 기울인다
 - 다양한 디자인 패턴이 복잡하게 적용됨
- 코드 퀄리티에 대한 높은 기준을 설정한다.

오늘 할 일

➤ 코드예제를 위한 환경설정

- 데이터베이스
- 자바환경 (JDK, JRE)
- IDE 환경설정 등...

➤ 프로젝트 디렉토리 구조 이해

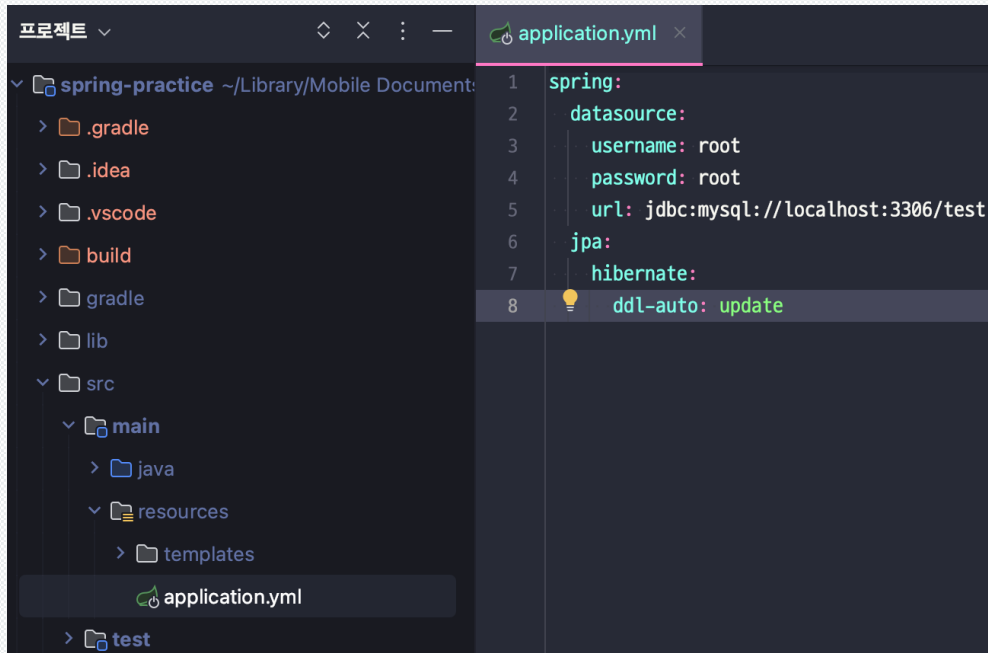
- 프로젝트 디렉토리 어느 위치에 어떤 파일이 있는지
- 각 파일이 어떤 역할을 하는지, 어떤 정보를 기술하는지

Spring Framework: entity

- Database상에 같은 유형의 데이터는 같은 Table에 저장
 - Table은 여러 속성(Column)으로 구성되어 있다.
 - 여러 속성을 갖는 데이터의 나열 정도로 이해 하자
- 이를 위해 SQL을 사용해야 하지만, 이번 실습에서는 ORM 사용
 - SQL 쿼리없이, 자바객체 형태로 테이블을 정의하고 질의(Query)

Spring Framework: entity

- DB Connection에 대한 정보는 application.yml에

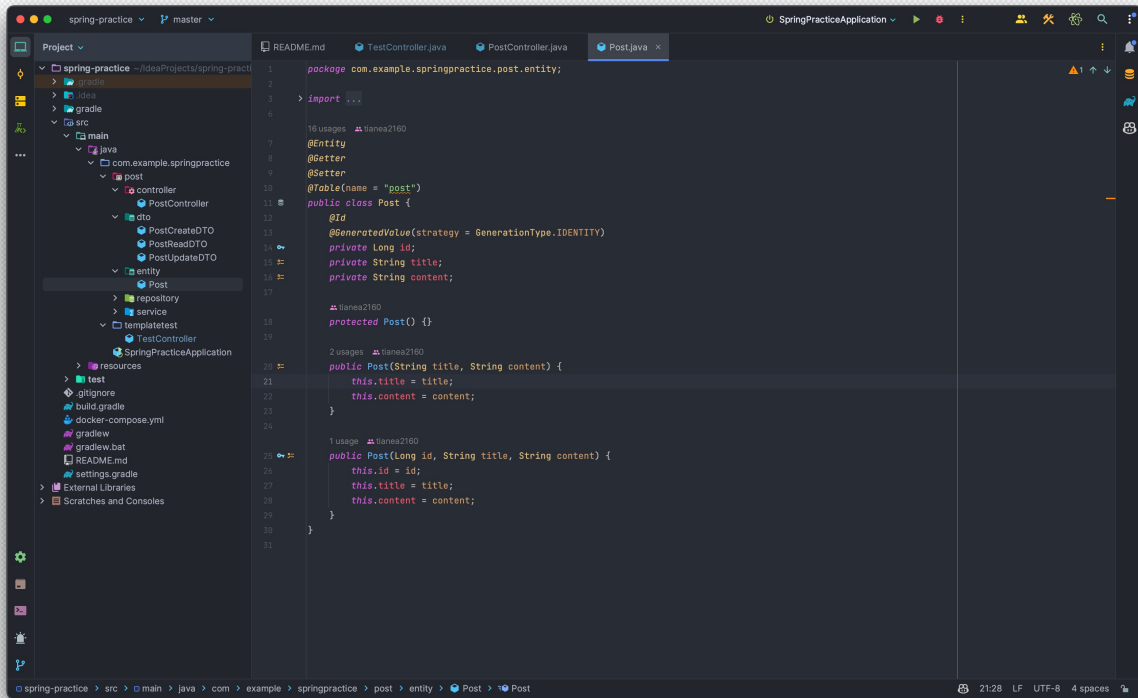


The screenshot shows a code editor with the file `application.yml` open. The left sidebar displays a project tree for `spring-practice` located at `~/Library/Mobile Documents`. The tree includes folders like `.gradle`, `.idea`, `.vscode`, `build`, `gradle`, `lib`, `src`, `main` (containing `java` and `resources`), and `test`. The `application.yml` file is selected in the tree and also in the editor tab. The editor shows the following YAML configuration:

```
1 spring:
2   datasource:
3     username: root
4     password: root
5     url: jdbc:mysql://localhost:3306/test
6   jpa:
7     hibernate:
8       ddl-auto: update
```

Spring Framework: entity

- DB Connection에 대한 정보는 application.yml에



Spring Framework: entity

- @Entity 어노테이션은 JPA엔티티를 선언하고 Table과 매핑
 - 매핑될 테이블은 @Table(name = "some_table")에 명시
 - 각 Column에 대한 명세는 멤버변수(필드)의 형태로 정의
 - @Getter @Setter 어노테이션을 통해 각 필드에 대한 접근 메서드 제공

```
16 usages  📄 linaea2160
7  @Entity
8  @Getter
9  @Setter
10 @Table(name = "post")
11 public class Post {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Long id;
15     private String title;
16     private String content;
17
18     📄 linaea2160
19     protected Post() {}
20
21     2 usages  📄 linaea2160
22     public Post(String title, String content) {
23         this.title = title;
24         this.content = content;
25     }
26
27     1 usage  📄 linaea2160
28     public Post(Long id, String title, String content) {
29         this.id = id;
30         this.title = title;
31         this.content = content;
32     }
33 }
```


Spring Framework: entity

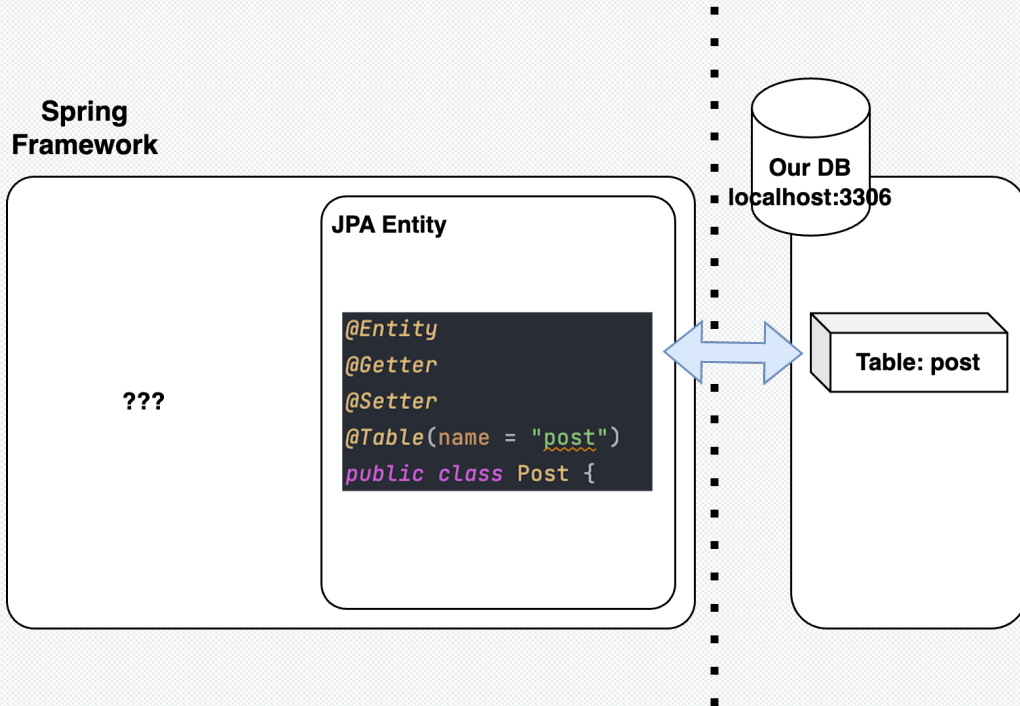
```
16 usages  tienea2160
7      @Entity
8      @Getter
9      @Setter
10     @Table(name = "post")
11     public class Post {
12         @Id
13         @GeneratedValue(strategy = GenerationType.IDENTITY)
14         private Long id;
15         private String title;
16         private String content;
17
18         tienea2160
19         protected Post() {}
20
21         2 usages  tienea2160
22         public Post(String title, String content) {
23             this.title = title;
24             this.content = content;
25         }
26
27         1 usage  tienea2160
28         public Post(Long id, String title, String content) {
29             this.id = id;
30             this.title = title;
31             this.content = content;
32         }
33     }
```

Spring Framework: entity

```
:: Spring Boot :: (v3.0.6)

2023-05-13T17:09:49.668+09:00 INFO 31521 --- [main] c.e.s.SpringPracticeApplication : Starting SpringPracticeApplication using Ja
2023-05-13T17:09:49.670+09:00 INFO 31521 --- [main] c.e.s.SpringPracticeApplication : No active profile set, falling back to 1 de
2023-05-13T17:09:49.962+09:00 INFO 31521 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories
2023-05-13T17:09:49.991+09:00 INFO 31521 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in
2023-05-13T17:09:50.222+09:00 INFO 31521 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http
2023-05-13T17:09:50.228+09:00 INFO 31521 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-05-13T17:09:50.228+09:00 INFO 31521 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.
2023-05-13T17:09:50.279+09:00 INFO 31521 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplication
2023-05-13T17:09:50.279+09:00 INFO 31521 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
2023-05-13T17:09:50.350+09:00 INFO 31521 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [
2023-05-13T17:09:50.381+09:00 INFO 31521 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.1.7
2023-05-13T17:09:50.516+09:00 INFO 31521 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-05-13T17:09:50.675+09:00 INFO 31521 --- [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection com.mysql.c
2023-05-13T17:09:50.676+09:00 INFO 31521 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-05-13T17:09:50.712+09:00 INFO 31521 --- [main] SQL dialect : HHH000400: Using dialect: org.hibernate.dia
Hibernate: create table post (id bigint not null auto_increment, content varchar(255), title varchar(255), primary key (id)) engine=InnoDB
2023-05-13T17:09:51.108+09:00 INFO 31521 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation
2023-05-13T17:09:51.113+09:00 INFO 31521 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for pe
2023-05-13T17:09:51.249+09:00 WARN 31521 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by defau
2023-05-13T17:09:51.409+09:00 INFO 31521 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
2023-05-13T17:09:51.413+09:00 INFO 31521 --- [main] c.e.s.SpringPracticeApplication : Started SpringPracticeApplication in 1.95 s
```

Spring Framework: entity



Spring Framework: Repository

> 데이터 저장소에 대한 추상화를 제공

- Entity는 테이블과 객체 사이의 매핑을 제공
- CRUD와 같은 데이터베이스와의 상호작용 인터페이스를 제공
- 필연적으로 두 개념은 매우 밀접

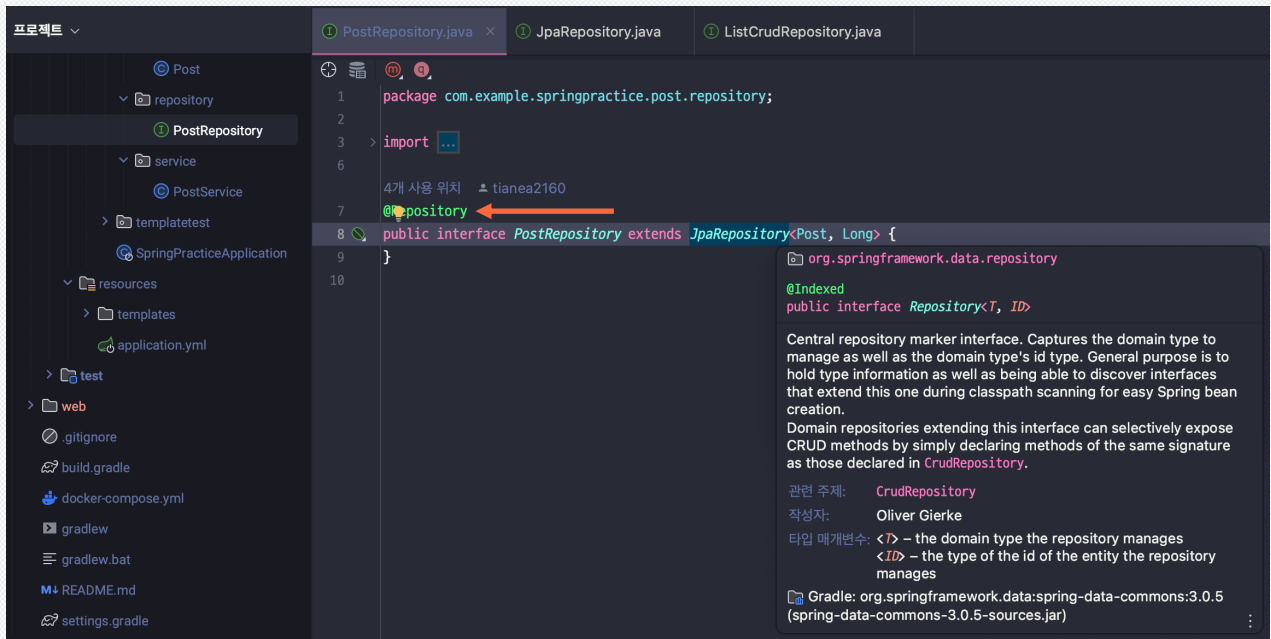
> JpaRepository를 상속하여 구현

- 기본적인 작업은 JpaRepository가 제공하지만, 어떤 Type인지 모른다
- 따라서, Generic을 통해 어떤 Entity, PK를 가진 Entity를 다룰지 명세

Spring Framework: Repository

> @Repository

- 난 데이터 처리하는 녀석이니까, 데이터 접근에러를 처리하고 Bean등록해줘



```
1 package com.example.springpractice.post.repository;
2
3 > import ...
4
5 4개 사용 위치 tianea2160
6
7 @Repository
8 public interface PostRepository extends JpaRepository<Post, Long> {
9 }
10
```

org.springframework.data.repository

@Indexed

public interface Repository<T, ID>

Central repository marker interface. Captures the domain type to manage as well as the domain type's id type. General purpose is to hold type information as well as being able to discover interfaces that extend this one during classpath scanning for easy Spring bean creation.

Domain repositories extending this interface can selectively expose CRUD methods by simply declaring methods of the same signature as those declared in **CrudRepository**.

관련 주제: **CrudRepository**

작성자: **Oliver Gierke**

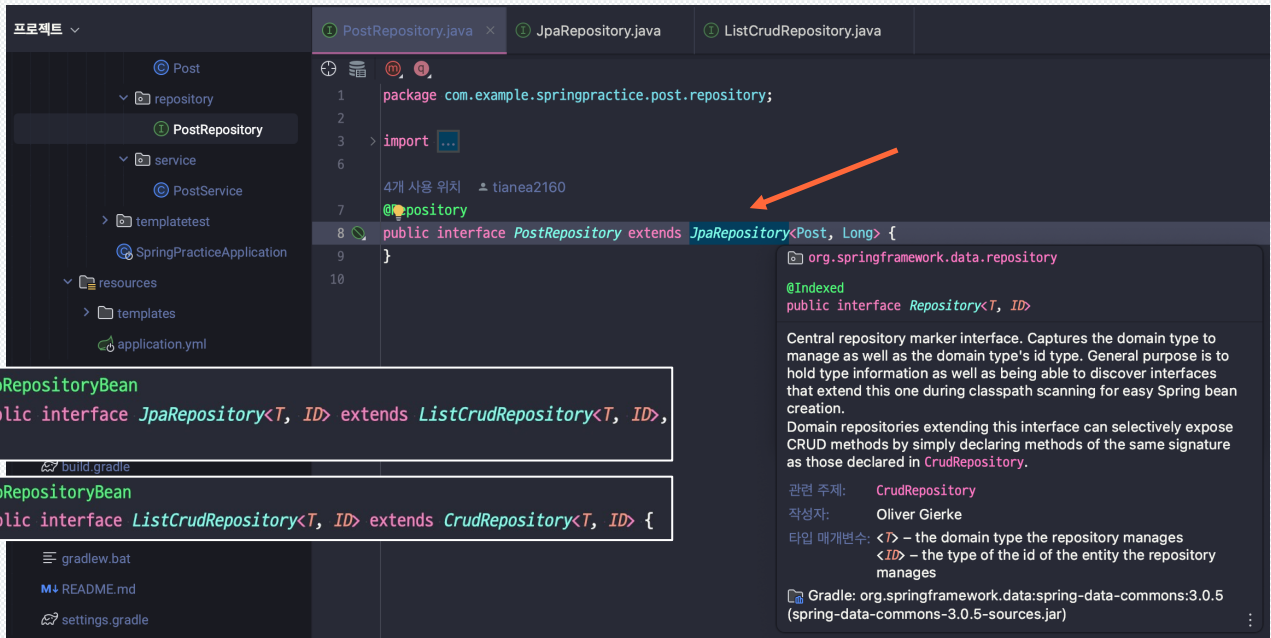
타입 매개변수: **<T>** – the domain type the repository manages
<ID> – the type of the id of the entity the repository manages

Gradle: org.springframework.data:spring-data-commons:3.0.5 (spring-data-commons-3.0.5-sources.jar)

Spring Framework: Repository

➤ JpaRepository

- 대부분의 CRUD기능은 내가 제공 할테니, 데이터 타입과 PK만 알려줘



The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Shows a project structure with folders like 'repository', 'service', 'templateTest', and 'resources'. The 'PostRepository' file is selected.
- Code Editor (Center):** Displays the code for 'PostRepository.java'. It shows the package 'com.example.springpractice.post.repository', an import statement for 'JpaRepository', and the definition of the 'PostRepository' interface which extends 'JpaRepository<Post, Long>'. An orange arrow points to the 'JpaRepository' in the extends clause.
- Documentation Panel (Right):** Provides details about the 'JpaRepository' interface, including its role as a 'Central repository marker interface' and its relationship to 'CrudRepository'.
- Code Snippets (Bottom):** Two snippets are shown, both marked with '@NoRepositoryBean':
 - Snippet 1: `public interface JpaRepository<T, ID> extends ListCrudRepository<T, ID>,`
 - Snippet 2: `public interface ListCrudRepository<T, ID> extends CrudRepository<T, ID> {`

Documentation Panel Content:

org.springframework.data.repository

`@Indexed`
`public interface Repository<T, ID>`

Central repository marker interface. Captures the domain type to manage as well as the domain type's id type. General purpose is to hold type information as well as being able to discover interfaces that extend this one during classpath scanning for easy Spring bean creation.

Domain repositories extending this interface can selectively expose CRUD methods by simply declaring methods of the same signature as those declared in `CrudRepository`.

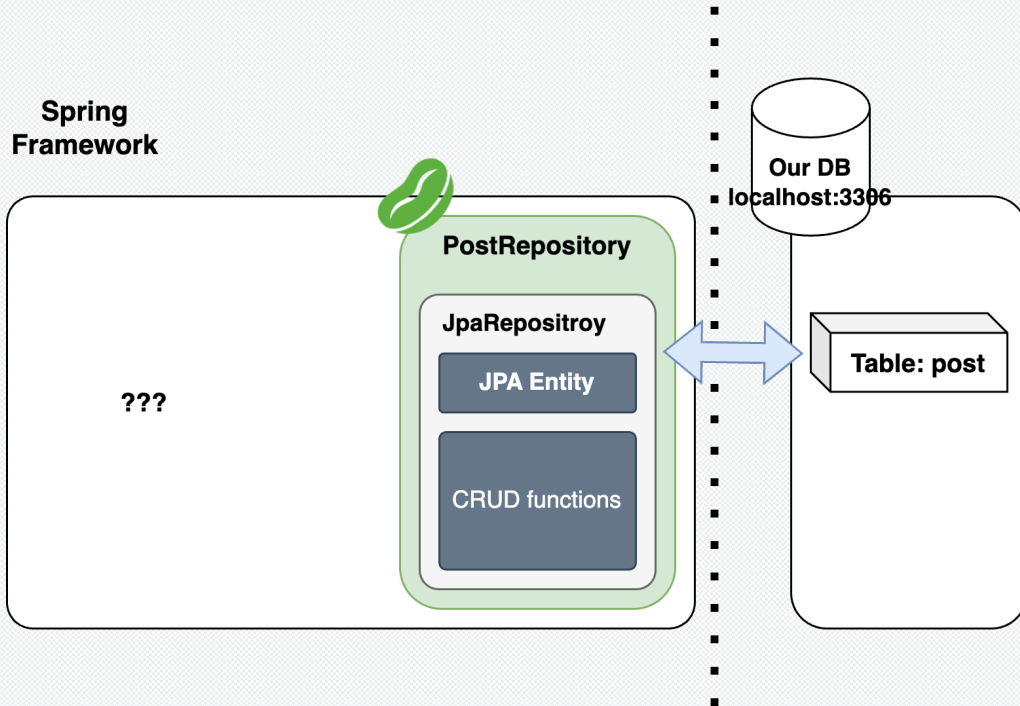
관련 주제: `CrudRepository`

작성자: Oliver Gierke

타입 매개변수: `<T>` - the domain type the repository manages
`<ID>` - the type of the id of the entity the repository manages

Gradle: org.springframework.data:spring-data-commons:3.0.5 (spring-data-commons-3.0.5-sources.jar)

Spring Framework: Repository



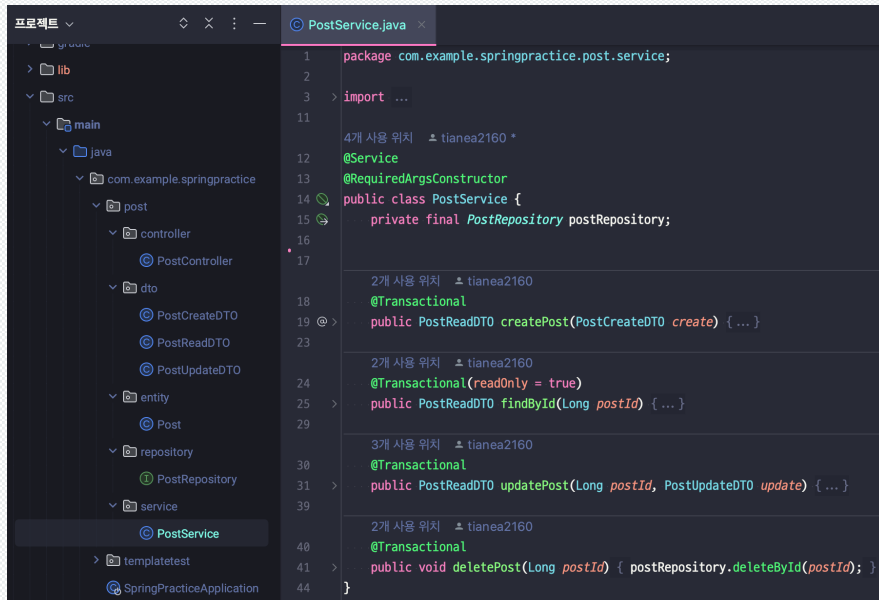
Spring Framework: Service

- 비즈니스 로직을 수행하는 요소(Class, Bean)
 - Eg. 거래 요청정보를 전달받아서 DB등을 활용, 실제 작업을 수행하는 것
 - 서비스 또한 Bean으로 관리
 - Client입장에서 어떤 서비스(기능)를 제공한다
- 다소 큰 범주의 서비스 안에 세부적인 메서드를 구현
 - 멤버 필드에 Bean(Repository, 연관 Service)을 주입 받는다
 - (의존성) 주입: 외부 서비스 요소의 생성/삭제를 프레임워크가 대신 수행

Spring Framework: Service

➤ @Service

- 난 비즈니스 로직을 명시할거니까 인스턴스 만들어주고, 날 쓰려는 다른 요소에게 알아서 전달해줘



```
1 package com.example.springpractice.post.service;
2
3 import ...
4
11
12 @Service
13 @RequiredArgsConstructor
14 public class PostService {
15     private final PostRepository postRepository;
16
17
18     @Transactional
19     public PostReadDTO createPost(PostCreatedDTO create) { ... }
20
21
22     @Transactional(readOnly = true)
23     public PostReadDTO findById(Long postId) { ... }
24
25
26     @Transactional
27     public PostReadDTO updatePost(Long postId, PostUpdateDTO update) { ... }
28
29
30     @Transactional
31     public void deletePost(Long postId) { postRepository.deleteById(postId); }
32
33 }
```

Spring Framework: Service

```
12 @Service
13 @RequiredArgsConstructor
14 public class PostService {
15     private final PostRepository postRepository;
16
17
18     @Transactional
19     public PostReadDTO createPost(PostCreateDTO create) { ... }
23
24     @Transactional(readOnly = true)
25     public PostReadDTO findById(Long postId) { ... }
29
30     @Transactional
31     public PostReadDTO updatePost(Long postId, PostUpdatedDTO update) { ... }
39
40     @Transactional
41     public void deletePost(Long postId) { postRepository.deleteById(postId); }
44 }
```

2개 사용 위치 tianea2160

2개 사용 위치 tianea2160

3개 사용 위치 tianea2160

2개 사용 위치 tianea2160

Spring Framework: Service

2개 사용 위치 tianea2160 *

```
@Transactional(readonly = true)
public PostReadDTO findById(Long postId) {
    Post post = postRepository.findById(postId).orElseThrow(
        () -> new IllegalArgumentException("해당 게시글이 없습니다. id=" + postId)
    );
    return PostReadDTO.from(post);
}
```

Spring Framework: Controller

➤ 웹 요청을 처리하는 요소

- 단순히 POJO(Java)코드로 응답할 수 있다
- Service를 통해 추상화된 로직을 수행한 결과를 응답할 수 있다
- `RequestBody`, `GetParam` 등 요청에 대한 분석

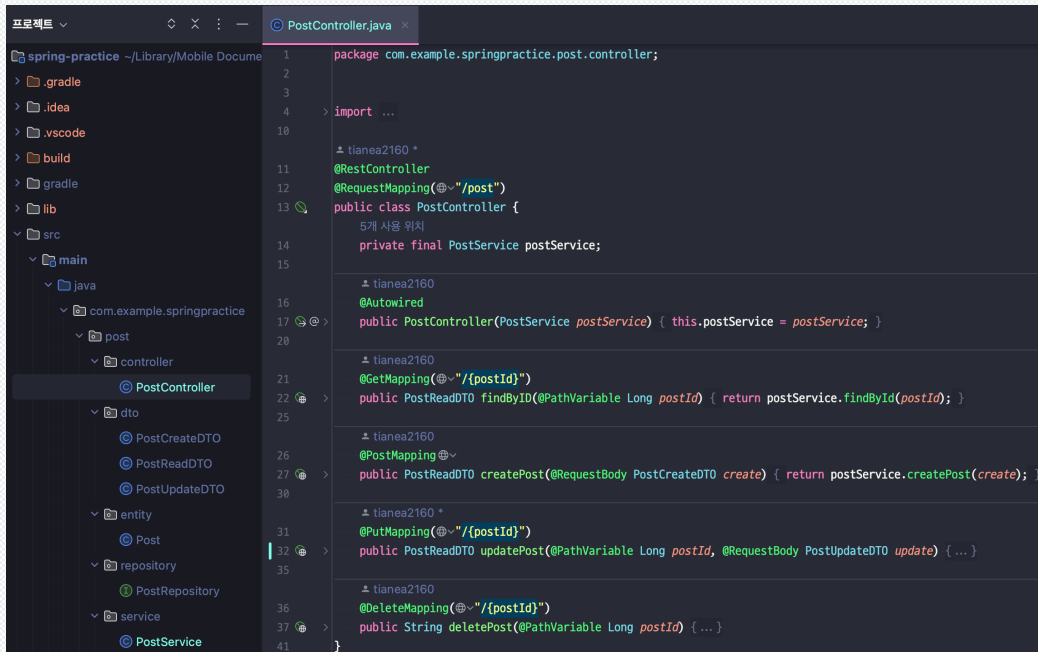
➤ 계층적으로, 특정 URL의 하위 URL을 모아서 정의

- */user/profile*
- */user/posts*
- */user* URL하위에 정의된 여러 URL은 큰 결에서 유사한 기능을 제공할 것
- `UserController`라는 `Controller`에 정의하면 명쾌하다!

Spring Framework: Controller

➤ @RestController

- 난 Rest라는 규격으로 통신하는 웹요청에 대한 Controller야

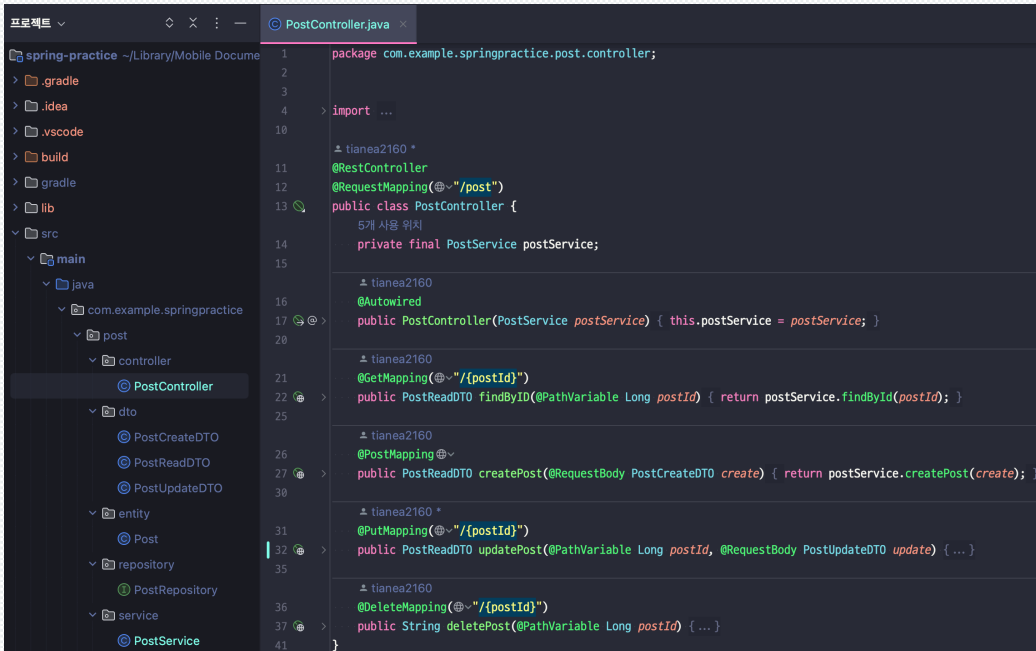


```
1 package com.example.springpractice.post.controller;
2
3
4 import ...
5
6
7
8
9
10
11
12
13 public class PostController {
14     5개 사용 위치
15     private final PostService postService;
16
17     1 tianea2160
18     @Autowired
19     public PostController(PostService postService) { this.postService = postService; }
20
21     2 tianea2160
22     @GetMapping("/{postId}")
23     public PostReadDTO findByID(@PathVariable Long postId) { return postService.findById(postId); }
24
25     3 tianea2160
26     @PostMapping
27     public PostReadDTO createPost(@RequestBody PostCreateDTO create) { return postService.createPost(create); }
28
29     4 tianea2160
30     @PutMapping("/{postId}")
31     public PostReadDTO updatePost(@PathVariable Long postId, @RequestBody PostUpdateDTO update) { ... }
32
33     5 tianea2160
34     @DeleteMapping("/{postId}")
35     public String deletePost(@PathVariable Long postId) { ... }
36
37
38
39
40
41 }
```

Spring Framework: Controller

➤ @RequestMapping("some-url")

- 프레임워크에 HTTP요청이 들어온 경우, "/post"로 시작하는 경로라면 날 쥐



```
1 package com.example.springpractice.post.controller;
2
3
4 import ...
5
6
7
8
9
10
11 @RestController
12 @RequestMapping("/post")
13 public class PostController {
14     private final PostService postService;
15
16     @Autowired
17     public PostController(PostService postService) { this.postService = postService; }
18
19
20
21     @GetMapping("/{postId}")
22     public PostReadDTO findId(@PathVariable Long postId) { return postService.findId(postId); }
23
24
25
26     @PostMapping
27     public PostReadDTO createPost(@RequestBody PostCreateDTO create) { return postService.createPost(create); }
28
29
30
31
32     @PutMapping("/{postId}")
33     public PostReadDTO updatePost(@PathVariable Long postId, @RequestBody PostUpdateDTO update) { ... }
34
35
36
37     @DeleteMapping("/{postId}")
38     public String deletePost(@PathVariable Long postId) { ... }
39
40
41 }
```

Spring Framework: Controller

> @PostMapping

- Controller의 Base URL("/post")로 전달된 POST요청은 내가 처리한다
- RequestBody를 PostCreateDTO의 형태로 받아서 postService에게 전달하고, 그 반환값을 Client한테 전해줄래

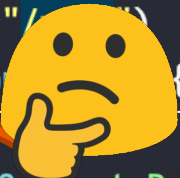
```
@RestController
@RequestMapping(🌐✓"/post")
public class PostController {
```

```
    @PostMapping(🌐✓)
    public PostReadDTO createPost(@RequestBody PostCreateDTO create) {
        return postService.createPost(create);
    }
```

Spring Framework: DTO

```
@RestController
@RequestMapping("/api")
public class PostController {

    @PostMapping
    public PostReadDTO createPost(@RequestBody PostCreatedDTO create) {
        return postService.createPost(create);
    }
}
```

A yellow thinking emoji with a hand on its chin, positioned over the code. An orange oval is drawn around the `PostReadDTO` parameter in the `createPost` method signature.

The image shows a code snippet from a Spring Framework application. The code defines a `PostController` with a `createPost` method. The method signature is `public PostReadDTO createPost(@RequestBody PostCreatedDTO create)`. The return type `PostReadDTO` is circled in orange, and a thinking emoji is placed over the code, suggesting a point of confusion or a key concept to understand.

Spring Framework: DTO


➤ DTO(Data Transfer Object)

- 표현하려는 Class의 속성을 Member로 표현함
- Getter / Setter를 가짐

➤ 분리된 계층 간의 데이터 전달

- 데이터의 접근권한또한 명세하여 다른 계층에서의 사용 범위를 제한
- 다른 계층이 정확히 필요로 하는 데이터 형식을 DTO로 명세하고 전달

Spring Framework: DI, Bean



```
@RestController
public class GreetingController {
    @Autowired
    private GreetingService greetingService;

    @GetMapping("/greet")
    public String greet(@RequestParam String name) {
        return greetingService.greet(name);
    }
}
```

Spring Framework: DI, Bean

> 의존성주입(DI, Dependency Injection)

- 멤버 객체(Service, Repository)를 직접 생성하는 건 너무 의존적이야
- 그럼 생성과 생명주기 관리를 대신 해주는 누군가가 있으면 되잖아?
- 프레임워크(Spring IOC Container)가 주입될 후보(Bean)을 미리 1개 생성해두고 필요에 따라 주입(Inject)한다

> 빈(Been)

- Spring Framework가 관리하는 객체(Service, Repository)인스턴스

> @Autowired

- Injection이 필요한 부분에 @Autowired 어노테이션을 명시한다

Spring Framework: DI, Bean

> @Autowired

- 멤버객체 또는 생성자에 @Autowired를 명시하여 Bean을 주입받는다

```
11 @RestController
12 @RequestMapping("/post")
13 public class PostController {
14     5개 사용 위치
15     private final PostService postService;
16
17     @Autowired
18     public PostController(PostService postService) {
19         this.postService = postService;
20     }
```

Spring Framework: DI, Bean

➤ @RequiredArgsConstructor

- 자동으로 생성자를 생성, Spring Framework는 생성자가 단 하나라면 @Autowired없이도 알아서 의존성 주입을 한다

```
12  @Service
13  @RequiredArgsConstructor
14  public class PostService {
15      private final PostRepository postRepository;
16
17
18      @Transactional
19      public PostReadDTO createPost(PostCreatedDTO create) {
20          Post post = postRepository.save(create.toEntity());
21          return PostReadDTO.from(post);
22      }
```