4장

Goal :

1.Hello World Controller를 만들어 보기

2.Boot Run을 통해서 서버 실행해보기

먼저 해당 클래스가 스프링에서 관리하는 컨트롤러 bean이라는 표식을 해준다.

스프링에서는 annotation 기반으로 지정한다.

@Controller

@ResponseBody

위 2개 annotation은 아래 annotation으로 치환 가능하다.

@RestController

Ctrl+b 단축키로 annotation의 선언 파일 바로가기하여 볼 수 있다.

여기서 RestController 에 @Controller, @ResponseBody 어노테이션이 선언되어 있는 것을 볼 수 있다.

스프링 부트에는 톰캣 서버를 임베디드해서 사용할 수 있게 되어 있다.

5장.

Goal

1.SpringBoot 테스트 만들기

2.MockMvc 테스트 만들기



## [ 테스트 생성 단축키 ] Controller method에서 ctrl+shift+t를 누르면 test 디렉토리에 동일한 package에 동일한 이름으로 test class를 생성해준다.

@SpringBootTest : 해당 클래스가 spring boot test임을 표시하기 위함



@Test Annotation 추가할 때 org.junit 사용하면 Junit4를 사용할 때 사용하는 것이고,

Org.junit.jupiter.api 는 Junit5를 사용할 때 사용한다.



테스트 생성 시 Junit 5가 잘 설정되었는지, 의존성이 제대로 추가 되었는지, Test가 정상적으로 동작하는지 먼저 확인 필요하다.



정상 실행 시 test 문구 출력 확인 가능하다.

@Autowired의 의미 : 스프링 컨텍스트에서 bean을 주입하겠다는 의미이다.



Springboot start test에 포함된 assert를 이용해서 검증문 작성 가능

Build gradle에 아래 junit 추가 ( 이 강의와는 다른 assert 라이브러리 사용)





Import org.junit.Assert; 임포트 후 Assert.assertEquals 메서드로 테스트 실행 가능하였음.

## 또 하나의 테스트 방식이 있다. (스프링에서 제공하는 MockMvc 방식)

MockMvc란? 실제 객체와 비슷하지만 테스트에 필요한 기능만 가지는 가짜 객체를 만들어서 애플리케이션 서버에 배포하지 않고도 스프링 MVC 동작을 재현할 수 있는 클래스를 의미한다.

모의 HttpRequest와 HttpResponse를 만들어서 테스트를 진행하게 해주는 것이다.



위 그림에서 MockMvcBuilders.standaloneSetup().build()로 MockMvc 먼저 세팅

이후 MockMvc.perform() 메서드로 동작하게 한다.

# 6장

Goal :

1.JPA 의존성 추가하기

2.H2 DB 의존성 추가하기

->

3.Entitiy 생성 및 테스트하기

JPA : 기존 DB Query를 이용해 데이터를 처리하던 것을 java를 통해 처리할 수 있도록 한번 더 래핑해준 라이브러리라 생각할 수 있음.

DB 종류에 상관없이 JPA를 사용할 수 있다. JPA 문법에 맞게 로직을 작성하면 사용하는 종류에 맞는 적당한 query를 생성해서 ORM이 처리해준다.

사용법 : build.gradle에 아래 implementation 'org.springframework.boot:spring-boot-starter-data-jpa' 한 줄 추가



## H2는 초경량으로 사용할 수 있는 메모리 DB이다.

서비스용 db로 사용할만한 수준은 되지 않지만 테스트나 캐싱용으로 많이 사용되고 있음.

의존성 추가 필요 : implementation 'com.h2database:h2'

@Entity : 해당 클래스는 entity로 사용한다는 표식

@Id : 해당 값을 ID라고 표시

@GeneratedValue : 값을 자동으로 생성하기 위한 값



JPA에서는 JPARepository를 상속받은 Repository interface를 만들면 기본적으로 create, update, delete, select 메서드를 생성해준다.

JpaRepository의 generic 인자로 Entity type와 id type을 넣는다.

public interface PersonRepository extends JpaRepository<Person, Long> {  
}



.findAll() – 메소드로 레퍼지토리의 전체 entity 목록을 가져와서 테스트를 한다.

# 7장

Goal

1.Lombok - @Getter

2.Lombok - @Setter

3.Lombok - @ToString



Lombok 라이브러리 사용하기 위해서는 build.gradle에 위 의존성 추가해 주어야 한다.



그리고 위 사진에서 처럼 ‘enable annotation processing’을 반드시 체크해주어야 한다.

@Getter, @Setter는 맴버변수와 클래스 위에 둘 다 사용할 수 있다.



출처 블로그 : <https://hyerin6.github.io/2020-02-04/lombok/>

## Lombok이란?

자바에서 DTO, VO 등에서 반복적으로 사용되는 코드를 annotation을 통해 쉽게 작성할 수 있게 도와주는 라이브러리.

동작원리?

Lombok 어노테이션이 붙은 자바 소스를 컴파일 할 때 annotation processor가 어노테이션을 확인하고 그에 맞는 코드를 자동으로 생성해서 바이트코드로 변환 한다.



<https://free-strings.blogspot.com/2015/12/lombok.html>

## [JAVA] Annotation 이란?

* 자바 소스 코드에 추가하여 사용할 수 있는 메타데이터의 일종

<https://mysend12.medium.com/java-annotation-1-ee04963c7d73>

annotation을 사용하는 이유 :

장점 : 기계적으로 사용하는 여러 코드들을 자동적으로 만들 수 있다.

주석과 같은 용도로 사용할 수 도 있다.

단점 : 의미가 함축적이다. 즉 Annotation의 정보를 모른다면 코드를 읽을 수 없다.

리팩토링을 하기 위해서는 사용된 Annotation의 정보를 알아야 수정할 수 있다. 또한, Annotation 자체를 수정하기는 매우 부담스럽다.



@ToString

-모든 필드의 값을 리턴하는 toString()메서드를 자동으로 생성하여 준다.

@ToString(exclude = “phoneNumber”) – exclude를 쓰면 지정한 값은 제외하고 값을 생성한다.



하지만 멤버변수 부분에 @ToString.Exclude 처럼 쓰는 것이 개발자들의 실수를 줄일 수 있는 방법이므로 이 방법으로 사용하는 것이 좋다.

# 8장

Goal

1. Lombok - @Constructor
2. Lombok - @EqualsAndHashCode
3. Lombok - @Data

@NoArgsConstructor – 파라메터 없는 생성자 생성

@AllArgsConstructor – 모든 파라메터 가지는 생성자 생성

@RequiredArgsConstructor 선언 후 멤버변수에 @NonNull을 입력하면 해당 멤버변수만 가지는 생성자를 자동으로 만들어 준다.





# 10장

Goal

1. 추가 Entity생성을 통한 복습
2. @OneToOne Relation
3. Java Stream, Filter, Map

Service 레이어 표시를 위해 @Service 표시하여 bean 으로 등록 해주어야 함.



## JAVA Stream <https://futurecreator.github.io/2018/08/26/java-8-streams/>





위 그림의 .map(String::toUppserCase) 람다 구문이나

**.map(클래스명::해당 클래스 존재하는 메서드) 로 작성할 수 있는듯, 아래처럼 input값 생략 가능 ,**

아래 이미지 처럼의 .map(detail->detail.getItem() ) 구문처럼 사용할 수 있다.

**.map(detail->detail.getItem() ) / .map( 스트림에 존재하는 input값, output되는 return 값 )**







# 12장

Goal

-@OneToOne : CascadeType

-@OneToOne : FetchType

-@OneToOne : optional, orphanRemoval

## JPA 영속성 관리 내용 잘 정리된 블로그 - <https://doublesprogramming.tistory.com/259>

## 영속성 전이 관련 잘 정리된 블로그 - <http://wonwoo.ml/index.php/post/1002>

<https://yellowh.tistory.com/127>









위 처럼 CascadeType.PERSIST를 설정하면, Person Entity에서 Block에 대한 영속성을 함께 관리하겠다는 의미이다.

그래서 blockPerson.setBlock(new Block(name)); 처럼 block 객체를 먼저 DB에 저장하지 않고, save() 메서드를 호출하여도, block 객체도 함께 DB Insert 된 것을 확인할 수 있다.





CascadeType.MERGE 옵션을 주지 않으면 자식 Entity의 변경사항을 업데이트 하여도 위처럼 DB에 변경사항이 적용이 안되는 것으로 보임.



CascadeType.MERGE 옵션 주면 정상적으로 자식 Entity의 값 변경이 적용된 것을 확인 할 수 있음.



CascadeType.REMOVE – 타입 적용을 안하면 부모 엔티티를 삭제해도 자식 엔티티가 삭제되지 않는 것을 확인할 수 있다. (personRepository.delete(person); 했음에도 로그 상에 Block 엔티티 삭제되지 않은 것 확인)

CascadeType.REMOVE 적용하면 부모 엔티티 삭제 시 자식 엔티티도 함께 삭제 됨.

CascadeType.ALL == {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REMOVE}

CascadeType.ALL은 위 3가지 타입을 모두 적용한 것임.



위 케이스에서는 부모 엔티티에서 자식 엔티티(Block)에 null을 넣었을 때(부모-자식 엔티티 간 관계가 끊겼을 때), 여전히 자식 엔티티는 삭제되지 않고 남아있는다.

이럴 경우 **orphanRemoval = true** 옵션을 주면 참조가 끊긴 자식 엔티티는 저절로 삭제가 되도록 할 수 있다.