

# Spring boot



## 1. सफल Data JPA



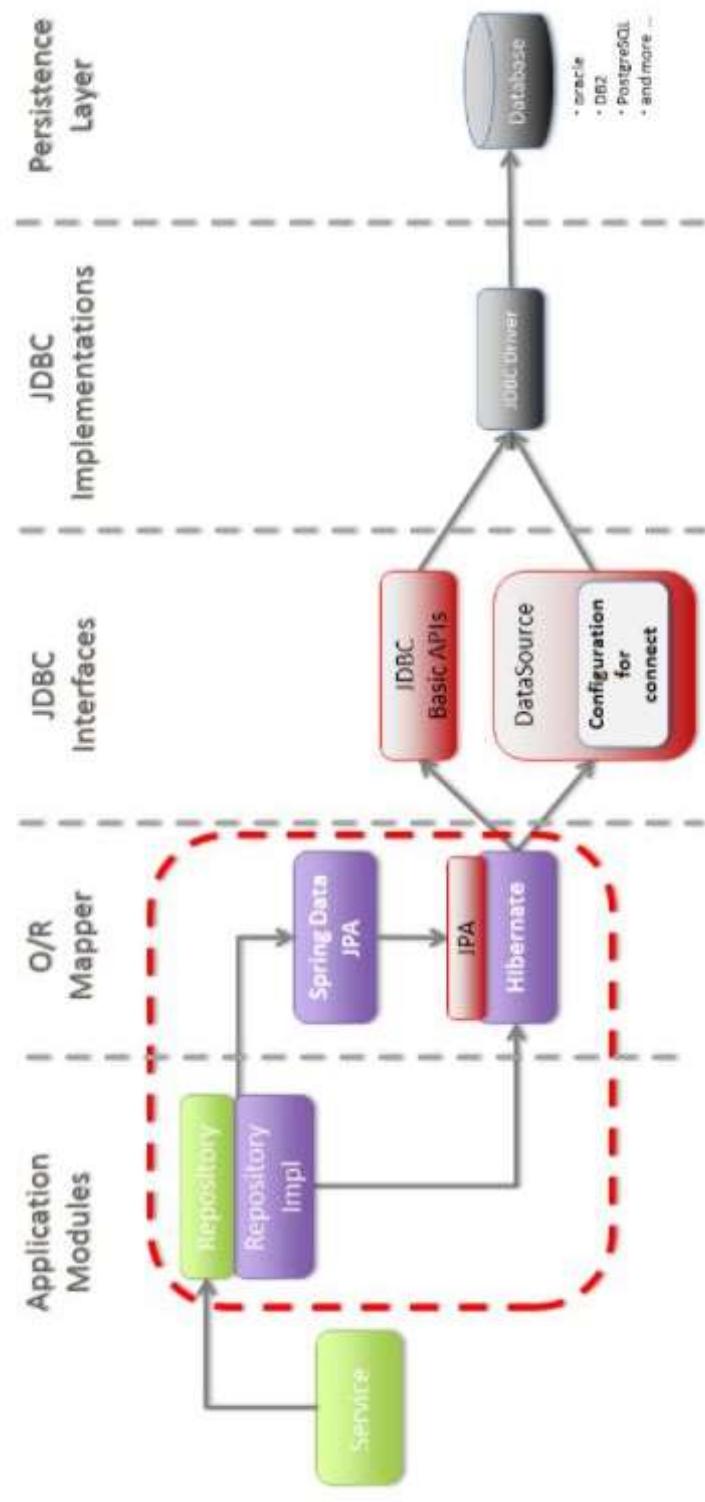
## 차례

- 1. Spring Data JPA
- 2. 프로젝트 생성

# 1. Spring Data JPA

## ■ JPA(Java Persistence API)

- 자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스
- 자바 ORM 기술에 대한 표준 명세로, JAVA에서 제공하는 API, 스프링에서 제공(X)
- 자바 클래스와 DB의 이를 매핑(sql을 매핑하지 않음)



# 1. Spring Data JPA

## ▪ SQL Mapper와 ORM

- ORM은 DB Entity를 자바 객체로 매팅함으로써 객체간의 관계를 바탕으로 SQL을 자동으로 생성하지만 Mapper 명시해 주어야 한다.
- ORM은 RDB의 관계를 Object에 반영하는 것이 목적이라면, Mapper는 단순히 필드를 매팅시키는 것이 목적이라. 지향점의 차이가 있음.

### SQL Mapper

- SQL ← mapping → Object 필드
- SQL 문으로 직접 데이터베이스를 조작
- Mybatis, jdbcTemplate

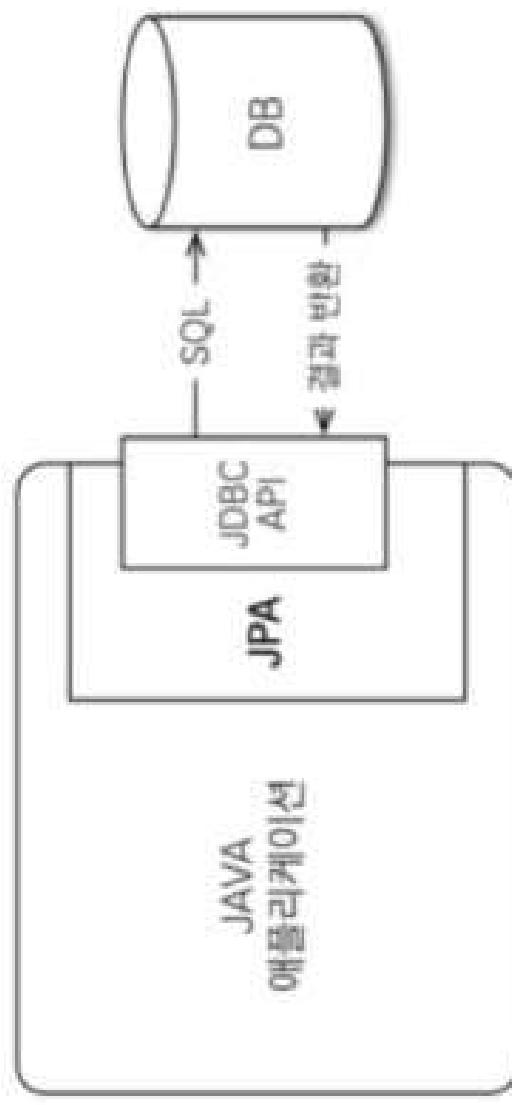
### ORM(Object-Relation Mapping/액체-관계 매팅)

- DB 데이터 ← mapping → Object 필드
  - 액체를 통해 간접적으로 디비 데이터를 다룬다.
- 객체와 디비의 데이터를 자동으로 매팅해줌.
- SQL 쿼리가 아니라 에서드로 데이터를 조작.
  - 액체간 관계를 바탕으로 sql을 자동으로 생성
- Persistant API라고 할 수 있다.
- JPA, Hibernate

## 1. Spring Data JPA

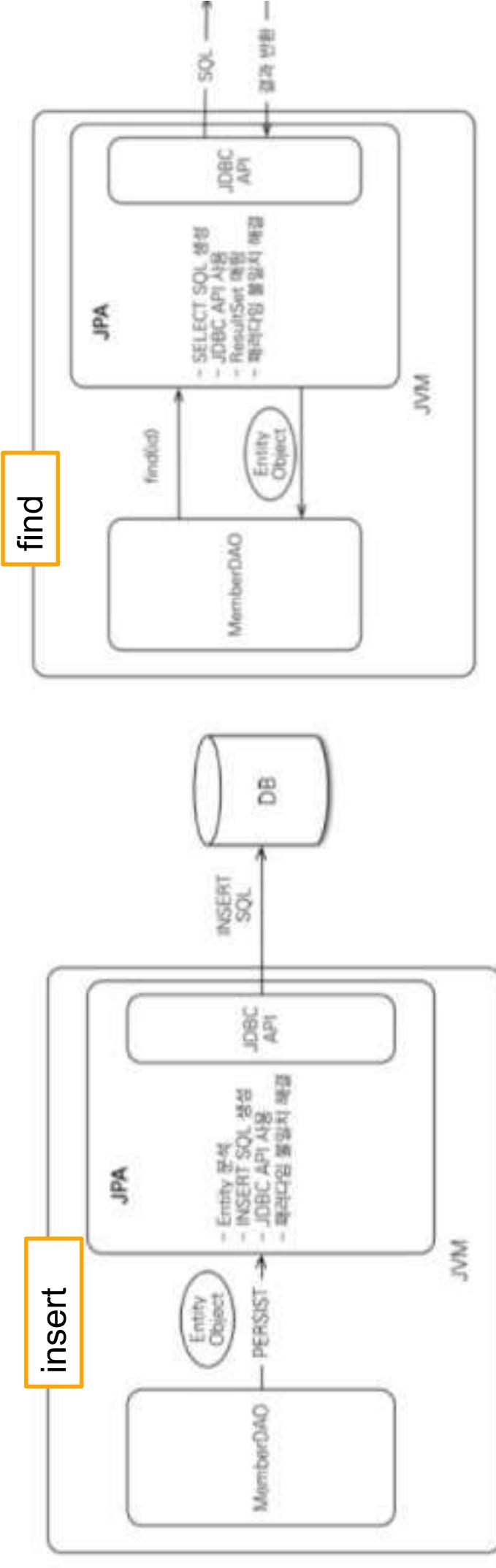
### ■ spring-data-jpa

- JPA는 ORM을 위한 자바 EE 표준이며 Spring-Data-JPA는 JPA를 쉽게 사용하기 위해 소프팅에서 제공하는 프레임워크
- 추상화 정도 : Spring-Data-JPA -> Hibernate -> JPA
- Spring Data JPA 장점
  - 구현체 교체의 용이성, 저장소 교체의 용이성



## 1. Spring Data JPA

### ■ JPA 동작 과정



## 2. JPA 프로젝트 생성



## 2. JPA 프로젝트 생성 개발 도구 지원

- Spring Initializr

개발 도구 지원

- eclipse
- IntelliJ
- VSCode

web상에서 프로젝트 생성 후 다운로드

- Maven/Gradle

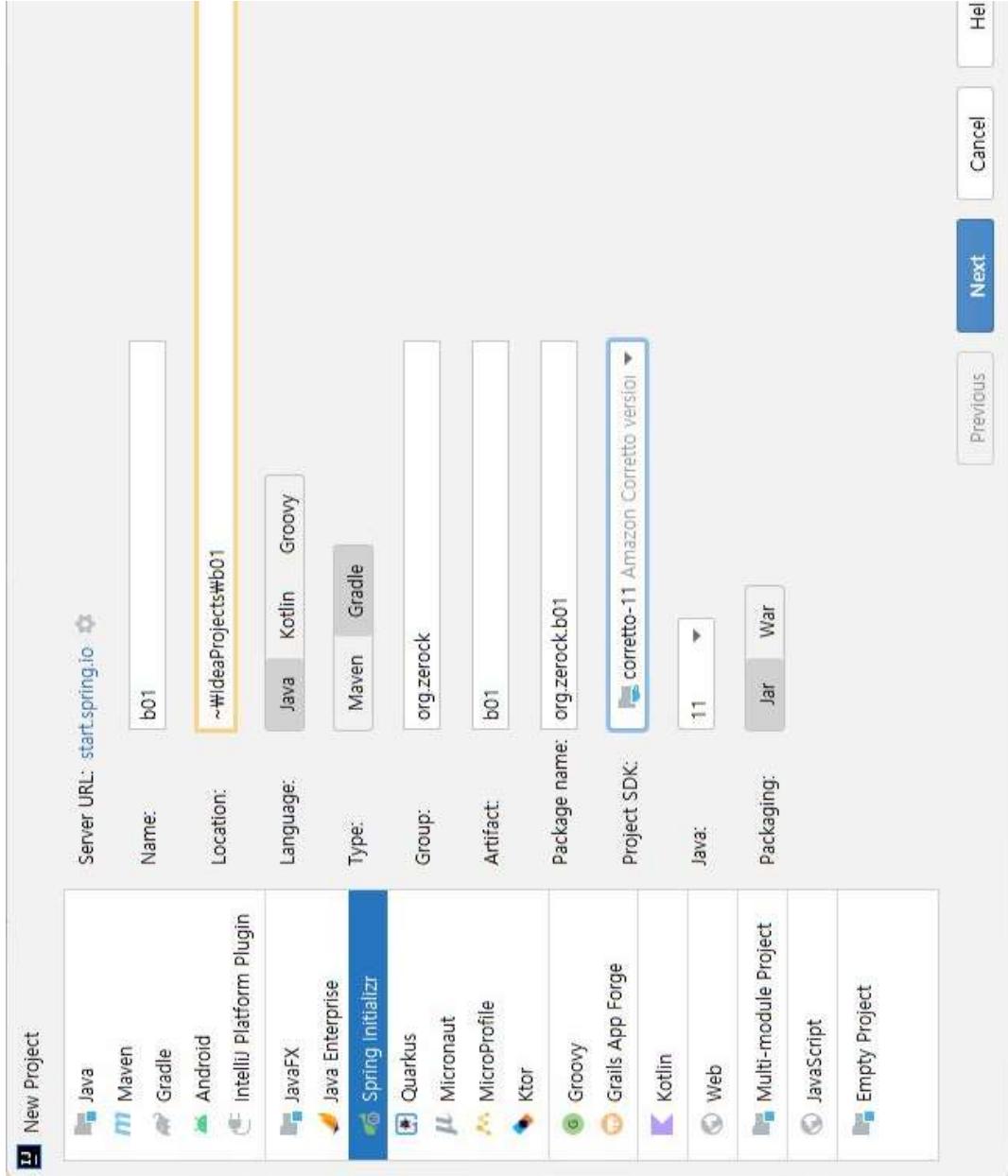
The screenshot shows the Spring Initializr web interface. At the top, there are buttons for 'ADD ...' and 'DEPENDENCIES'. Below that, there are sections for 'Project' (Maven Project selected), 'Language' (Java selected), and 'Dependencies' (No dependency selected). Under 'Project Metadata', there are sections for 'Server URL', 'Name' (b01), 'Location' ('~/ideaProjects/b01'), 'Language' (Kotlin selected), 'Type' (Spring Initializr selected), 'Group' (org.zerock), 'Artifact' (b01), 'Package name' (org.zerock:b01), 'Project SDK' (corretto-11 Amazon Corretto version), 'Java' (11 selected), 'Packaging' (War selected), and 'JavaScript' (Empty Project selected). At the bottom right, there are buttons for 'GENERATE', 'EXPLORE', and 'SHARE...'. The entire interface is framed by a red border.

The screenshot shows the 'New Project' dialog in IntelliJ IDEA. It lists various project types: Java, Maven, Gradle, Android, IntelliJ Platform Plugin, JavaFX, Java Enterprise, Quarkus, Micronaut, MicroProfile, Ktor, Groovy, Grails App Forge, Kotlin, Web, Multi-module Project, JavaScript, and Empty Project. The 'Java' type is selected. On the right, there are fields for 'Name' (b01), 'Location' ('~/ideaProjects/b01'), 'Language' (Kotlin selected), 'Type' (Spring Initializr selected), 'Group' (org.zerock), 'Artifact' (b01), 'Package name' (org.zerock:b01), 'Project SDK' (corretto-11 Amazon Corretto version), 'Java' (11 selected), 'Packaging' (War selected), and 'JavaScript' (Empty Project selected). At the bottom right, there are buttons for 'Previous', 'Next', 'Cancel', and 'Help'. The entire dialog is framed by a red border.

## 2. JPA 프로젝트 생성

### ▪ 프로젝트 생성 시 추가가 하는 의존성 라이브러리

- Spring Boot DevTools
- Lombok
- Spring Web
- Thymeleaf
- Spring Data JPA
- MysqlDB Driver



## 2. JPA 프로젝트 생성 : DataSource 설정

- 자동 설정기능으로 인해 의존성 라이브러리를 추가한 것 만으로 자동으로 관련 설정을 시동
- HikariCP를 기본으로 사용
- Spring Data JPA에서 사용할 데이터베이스 관련 설정 필요
- 설정 파일

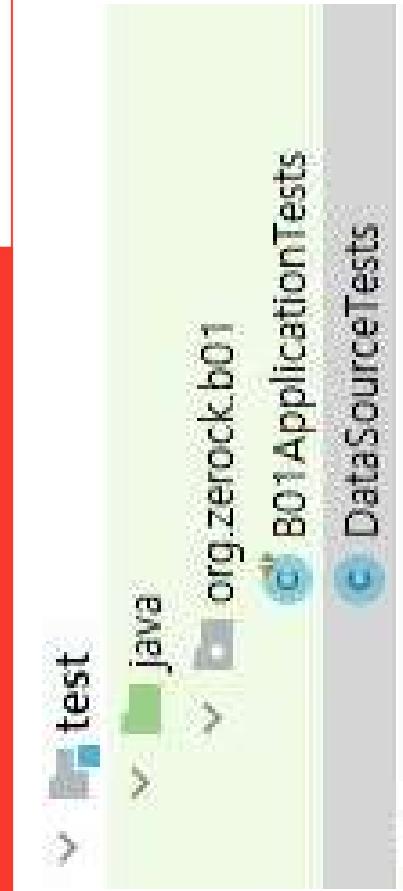
```
application.properties 푸드 application.xml 혼자서  
*****  
APPLICATION FAILED TO START  
*****
```

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured

Reason: Failed to determine a suitable driver class

## 2. JPA 프로젝트 생성 : 테스트 환경과 의존성 주입



```
@SpringBootTest  
@Log4j2  
public class DataSourceTests {  
  
    @Autowired  
    private DataSource dataSource;  
  
    @Test  
    public void testConnection() throws SQLException {  
        @Cleanup  
        Connection con = dataSource.getConnection();  
        log.info(con);  
  
        Assertions.assertNotNull(con);  
    }  
}
```

```
JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries in org.zerock.b01.DataSourceTests : Started DataSourceTests in 2.249 seconds (JVM running for 3.271)  
org.zerock.b01.DataSourceTests : HikariProxyConnection@20086483651 wrapping org.mariadb.jdbc.MariaDbConnection  
j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'  
com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...  
com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

## 2. JPA 프로젝트 생성 : Spring Data JPA를 위한 설정

- **spring.jpa.hibernate.ddl-auto=update ddl 처리방법**
- **spring.jpa.properties.hibernate.format\_sql=true //Hibernate의 DB에 보내는 모든 쿼리(DDL, DML) 보여줌**
- **logging.level.org.hibernate.type.descriptor.sql=trace //hibernate의 보여주는 로그에 있는 ?에 들보여줌**
- **spring.jpa.show-sql=true //**

속성값	의미
<b>none</b>	DDL을 하지 않음
<b>create-drop</b>	실행할때 DDL을 실행하고 종료시에 만들어진 테이블등을 모두 삭제
<b>create</b>	실행할때마다 새롭게 테이블들을 생성
<b>update</b>	기존과 다르게 변경된 부분이 있을때는 새로 생성
<b>validate</b>	변경된 부분만 알려주고 종료

## 2. JPA 프로젝트 생성 : 전체 설정

### ■ application.properties

server.port=8081

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/shop?useSSL=false&serverTimezone=Asia/Seoul&&characterEncoding=UTF-8  
spring.datasource.username=pgm  
spring.datasource.password=1234  
  
#실행되는 쿼리 콘솔 출력  
spring.jpa.properties.hibernate.show_sql=true  
  
#콘솔창에 출력되는 쿼리를 가독성이 좋게 포맷팅  
spring.jpa.properties.hibernate.format_sql=true
```

- **ddl-auto 옵션 종류**
  - ✓ create: 기존 테이블 삭제 후 다시 생성 (DROP + CREATE)
  - ✓ create-drop: create와 같으나 종료 시점에 테이블 DRO
  - ✓ update: 변경 분만 반영(운영 DB에서는 사용하면 안됨)
  - ✓ validate : 엔티티와 테이블이 정상 매핑되었는지 확인
  - ✓ none : 사용하지 않음(사실상 없는 값이지만 관례상 포함)

### • 주의할 점

- 운영 장비에서는 절대 create, create-drop, update 사용 X
  - ✓ 개발 초기 단계는 create 또는 update
  - ✓ 테스트 서버는 update 또는 validate
  - ✓ 스테이징과 운영 서버는 validate 또는 none
- ```
#쿼리에 물음표로 출력되는 바인드 파라미터 출력  
logging.level.org.hibernate.type.descriptor.sql=trace  
  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

## 2. JPA 프로젝트 생성 : 전체 설정

### ▪ application.yml

```
server:  
  port: 8083  
  
spring:  
  datasource:  
    driver-class-name: com.mysql.cj.jdbc.Driver  
    url: jdbc:mysql://localhost:3306/springdb?useSSL=false&serverTimezone=UTC  
    username: pgm  
    password: 1234  
  
  devtools:  
    livereload:  
      enabled: true # 코드수정 실시간 반영  
  
jpa:  
  hibernate:  
    ddl-auto: update # ddl-auto 옵션  
    show-sql: true # 실행되는 쿼리 콘솔 출력  
  
mvc:  
  view:  
    prefix: /WEB-INF/views/  
    suffix: .jsp
```

### 3. 스프링부트에서의 웹 개발



### 3. 스프링부트에서의 웹 개발

- `web.xml`이나 `servelt-context.xml`이 없는 환경에서 개발
- 설정을 위한 `@Configuration`이나 상속등을 사용
- 스프링부트는 기본적으로 JSP를 지원하지 않음

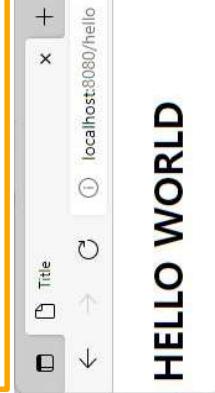
```
java
  org.zerock.b01
    controller
      SampleController

package org.zerock.b01.controller;
...
@Controller
@Log4j2
public class SampleController {

  @GetMapping("/hello")
  public void hello(Model model) {
    log.info("hello.....");
    model.addAttribute("msg", "HELLO WORLD");
  }
}
```

```
resources
  static
    templates
      hello.html

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <h1 th:text="${msg}"></h1>
</body>
</html>
```



The screenshot shows a browser window with the title 'Title'. The address bar shows 'localhost:8080/hello'. The page content is a single large blue header 'HELLO WORLD'.

### 3. 스프링부트에서의 웹 개발 : JSON 태이터 만들기

```
package org.zerock.b01.controller;
import lombok.extern.log4j.Log4j2;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@Log4j2
public class SampleJSONController {
    @GetMapping("/helloArr")
    public String[] helloArr() {
        log.info("helloArr.....");
        return new String[]{"AAA", "BBB", "CCC"};
    }
}
```



별도의 라이브러리 없이 스프링부트는  
json 처리를 지원

## 4. Thymeleaf



## 4. Thymeleaf

### ■ Thymeleaf의 특징

- JSP의 경우 서블릿으로 변환된 후에 실행되는 방식
- Thymeleaf는 서버 사이드 템플릿 엔진
- HTML의 구조에 추가적인 태그 없이 선언적으로 데이터 바인딩 처리

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1 th:text="${msg}"></h1>
</body>
</html>
```



```
@GetMapping("/ex/ex1")
public void ex1(Model model){
```

```
    List<String> list = Arrays.asList("AAA", "BBB", "CCC", "DDD");
    model.addAttribute("list", list);
}
```

The screenshot shows a browser window with the URL <http://localhost:8080/ex/ex1>. The page title is "Title". The content of the page is a list of four items: [AAA, BBB, CCC, DDD]. Above the browser window, there is a code editor or template view showing the Thymeleaf template code.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<h4>[$list]</h4>
<hr/>
<h4 th:text="${list}"></h4>
</body>
</html>
```

org.zerock.b01  
controller  
SampleController

templates

ex  
ex1.html

## 4. Thymeleaf : 주석 처리

- 주석 처리를 해야 할 때에는 '<!--/\* ... \*/-->' 를 이용

```
<body>
<h1 th:text="${msg}"></h1>

<!--/* <h3 th:each="${sos}">SOS</h3>
 */-->

<!--/* ${aaaa + bbb } */-->
<!--/* -->
<div>
<h1>AAAAA</h1>
</div>
*/-->

</body>
```



## 4. Thymeleaf - th:with을 이용한 변수 선언

반복문과 함께 사용자

```
<div th:with="num1 = ${10}, num2 = ${20}">
<h4 th:text="${num1 + num2}"></h4>
</div>
```

th:each를 이용해서 배열/리스트/컬렉션 처리

반복문의 status 변수

- 반복문에서 자주 사용하는 인덱스 번호나 개수등을 사용
- index/count/size/first/odd/even

th:if / th:unless / th:switch를 이용한 제어 구조

```
<ul>
<li th:each="str, status : ${list}">
<span th:if="${status.odd}"> ODD -- [[${str}]] </span>
<span th:unless="${status.odd}"> EVEN -- [[${str}]] </span>
</li>
</ul>
```

## 4. Thymeleaf : Thymeleaf를 이용한 링크 처리

절대 경로/컨텍스트 경로를 자동으로 처리

'@'를 이용해서 처리

커리 스트링 처리

```
<a th:href="@{/hello(name='AAA', age= 16) }">Go to /hello</a>
<a href="@{hello?name=AAA&age=16}">Go to /hello</a>

<a th:href="@{/hello(types=${ 'AA', 'BB', 'CC' } ), age= 16) }">Go to /hello<
<a href="@{/hello?type=AA&age=16}">Go to /he
```

## 4. Thymeleaf : thymeleaf의 특별한 기능

### ■ 인라인 처리 : JavaScript의 경우 변수를 자동으로 JavaScript Object 형태로 출력

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div th:text="${list}"></div>
    <div th:text="${map}"></div>
    <div th:text="${dto}"></div>
    <script>
        const list = ["Data1", "Data2", "Data3", "Data4", "Data5", "Data6", "Data7", Data8,
        {A=AAA, B=BBB}
        org.zerock.b01.controller.SampleController$SampleDTO()
    </script>
<script th:inline="javascrip">
    const map = {"A": "AAAA", "B": "BBBB"}
    const dto = {"p1": "Value -- p1", "p2": "Value -- p2", "p3": "Value -- p3"}
    console.log(list)
    console.log(map)
    console.log(dto)
    console.log(map)
    console.log(dto)
    </script>
</body>
</html>
```

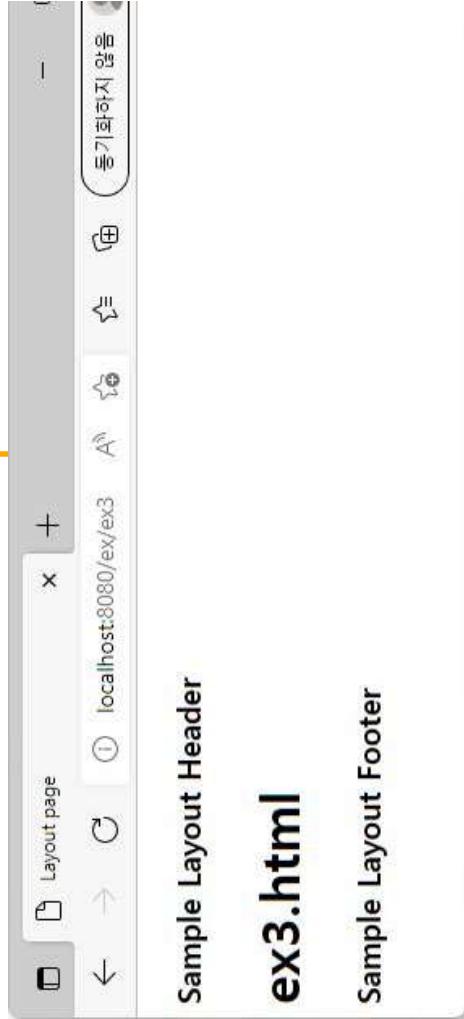
## 4. Thymeleaf : 레이아웃 기능

- <th:block>을 이용해 서적 틀을 블루마트 자서전 바시  
implementation 'nz.net.ultraq.thymeleaf:thymeleaf-layout-dialect:3.1.0'

layout:fragment를 이용해서 변경이 가능한 부분을  
지정하고 나중에 다른 내용으로 변경 가능

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout/layout1.html}">
```

```
<div layout:fragment="content">
<h1>ex3.html</h1>
</div>
```



```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Layout page</title>
</head>
<body>
    <div>
        <h3>Sample Layout Header</h3>
    </div>

    <div layout:fragment="content">
        <p>Page content goes here</p>
    </div>

    <div>
        <h3>Sample Layout Footer</h3>
    </div>

    <th:block layout:fragment="script" >
    </th:block>
</body>
</html>
```



## **5. Entity**



## 5. Entity

### ▪ 엔티티 설계

```
@Data  
@Entity  
public class JpaMember {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    private String name;  
    private String password;  
    private String email;  
    private String memo;  
    @Column(name="address")  
    private String addr;
```

```
@Entity  
@Table(name="item")  
@Getter @Setter @ToString  
public class Item {  
    @Id  
    @Column(name="item_id")  
    @GeneratedValue(strategy=GenerationType.AUTO)  
    private Long id; //상품 코드  
  
    @Column(nullable=false, length=50)  
    private String itemNm; //상품명  
  
    @Column(name="price", nullable=false)  
    private int price; //가격  
  
    @Column(nullable=false)  
    private int stockNumber; //재고수량  
  
    @Lob  
    @Column(nullable=false)  
    private String itemDetail; //상품 상세 설명  
  
    @Enumerated(EnumType.STRING)  
    private ItemSellStatus itemSellStatus; //상품 판매 상태  
  
    private LocalDateTime regTime;  
  
    private LocalDateTime updateTime;
```

## 5. Entity

### ■ 엔티티 매핑 관련 어노테이션

어노테이션	설명
@Entity	데이터베이스 테이블과 1:1로 매칭되는 객체, Entity 객체의 인스턴스 하나가 테이블에서 하나의 값을 의미
@Table	엔티티와 매핑할 테이블 이름 명시적으로 설정, 설정하지 않으면 table명 자동설정
@Id	테이블의 기본 키에 사용할 속성 지정
@GeneratedValue	키 값 생성 전략 명시
@Column	필드와 컬럼 매핑 및 필드 속성 지정(필드명, not null, length 등)
@Lob	BLOB, CLOB 타입 매핑
@CreationTimestamp	데이터 insert 시 시간 자동 저장
@UpdateTimestamp	데이터 update 시 시간 자동 저장
@Enumerated	Enum 타입 매핑
@Transient	해당 필드 데이터베이스 매팅 무시
@Temporal	날짜 타입 매핑
@CreateDate	엔티티가 생성되어 저장될 때 시간 자동 저장
@LastModifiedDate	조회한 엔티티의 값을 변경할 때 시간 자동 저장
@DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm:ss")	날짜 시간 포맷 설정

## 5. Entity

### ■ @Column 어노테이션의 속성

속성	설명	기본값
name	필드와 매핑할 컬럼 이름 설정	각 커스텀 이름 필드
unique(DDL)	유니크 제약조건 설정	
insertable	insert 가능 여부	true
updateable	update 가능 여부	true
length	String 타입의 문자 길이 제약조건 설정	255
nullable(DDL)	null 값의 허용 여부 설정, false 설정 시 DDL 생성 시에 not null 제약조건 설정	
columnDefinition	데이터베이스 컬럼 정보 기술 예) @Column(columnDefinition="varchar(5) default '10' not null")	
precision, scale(DDL)	BigDecimal 타입에서 사용(BigInteger 가능) precision은 소수점을 포함한 전체 자리수이고, scale은 소수점 자리수, Double과 float 타입에 적용되지 않음	

## 5. Entity

### ▪ @GeneratedValue 어노테이션 4가지 전략

생성전략	설명
GenerationType.AUTO(default)	JPA 구현체가 자동으로 생성 전략 결정
GenerationType.IDENTITY	기본키 생성을 데이터베이스 위임 MySQL 데이터베이스의 경우 AUTO_INCREMENT 를 사용하여 기본 키 생성 ex) @GeneratedValue(strategy=GenerationType.IDENTITY) <pre>private Long id;</pre>
GenerationType.SEQUENCE	데이터베이스 시퀀스 오브젝트를 사용하여 기본 키 생성 @SequenceGenerator를 사용하여 시퀀스를 등록이 필요 ex) @SequenceGenerator(name='seq', sequenceName="jpa_sequence") @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq") <pre>private Long id;</pre>
GenerationType.TABLE	키 생성용 테이블을 사용 @TableGenerator 필요

## 6. Repository



## 6. Repository

- **Repository 설계**

```
public interface MemberRepository extends JpaRepository<Member, Long>{}
```

- **쿼리 메소드 :** 메소드 이름으로 쿼리 생성

- 스프링 데이터 JPA가 제공하는 쿼리 메소드 기능

- 메소드 이름으로 쿼리 생성 : 메소드 이름을 분석해서 JPQL 쿼리 실행하는 것

- 메소드 이름으로 JPA NamedQuery 호출

- **@Query 어노테이션을 이용한 쿼리 직접 정의**

- Spring Data JPA QueryDSL

## 6. Repository

---

### ▪ 큐리메소드 세부 기능

- 조회: find...By ,read...By ,query...By get...By : <T> 타입 반환
- COUNT: count...By : long 타입 반환
- EXISTS: exists...By : Boolean 타입 반환
- 삭제: delete...By, remove...By : long 타입 반환
- DISTINCT: findDistinct, findMemberDistinctBy
- LIMIT: findFirst3, findFirst, findTop, findTop3

## 6. Repository

- 쿼리 메소드
- Repository에서 지원하는 기본 메소드

메소드	기능
save(S entity)	엔티티 저장 및 수정
delete(T entity)	엔티티 삭제
count()	엔티티 총 개수 반환
findAll()	모든 엔티티 조회
findOne()	primary key로 한건의 레코드 찾기

- findBy, countBy Entity필드명 메소드

메소드	기능
findBy_____	쿼리를 요청하는 메서드 임을 알림
countBy_____	쿼리 결과 레코드 수를 요청하는 메서드임을 알림

## 6. Repository

- Query DSL에 포함할 수 있는 키워드

메서드 이름 키워드	샘플	설명
And	findByEmailAndUserId(String email, String userId)	여러 필드를 and로 검색
Or	findByEmailOrUserId(String email, String userId)	여러 필드를 or로 검색
Between	findByCreatedBetween(Date fromDate, Date toDate)	필드의 두 값 사이에 있는 항목 검색
LessThan	findByAgeGreaterThanOrEqual(int age)	작은 항목 검색
GreaterThanOrEqual	findByAgeGreaterThanOrEqual(int age)	크거나 같은 항목 검색
Like	findByNameLike(String name)	like 검색
IsNull	findByJobIsNull()	null인 항목 검색
In	findByJob(String ... jobs)	여러 값중에 하나인 항목 검색
OrderBy	findByEmailOrderByNameAsc(String email)	검색 결과를 정렬하여 전달

## 6. Repository

### ▪ Pageable

- Query 메소드의 입력변수로 **Pageable** 변수를 추가하면 **PageEntity**를 반환형으로 사용
- **Pageable** 객체를 통해 페이징과 정렬을 위한 파라미터를 전달

```
@RestController  
@RequestMapping("/member")  
public class MemberController {  
  
    @Autowired  
    MemberService memberService;  
  
    @RequestMapping("")  
    Page<Member> getMembers(Pageable pageable){  
        return memberService.getList(pageable)  
    }  
}
```

query parameter 명	설명
page	몇 번째 페이지 인지를 전달
size	한 페이지에 몇 개의 항목을 보여줄 것인지 전달
sort	정렬정보를 전달. 정렬 정보는 필드이름, 정렬방식으로 전달한다. 여러 필드로 순차적으로 가능하다. 예 : sort=createdAt,desc&sort=userId,asc

## 6. Repository

### ▪ Spring DATA JPA @Query 어노테이션 사용

- SQL과 유사한 JPQL(Java Persistence Query Language)라는 각 체지향 쿼리 언어 사용

```
@Query("select i from Item i where i.itemDetail like %:itemDetail% order by i.price desc")  
List<Item> findByItemDetail(@Param("itemDetail") String itemDetail);
```

```
@Query(value="select * from item i where i.item_detail like %:itemDetail% order by i.price desc", nativeQuery=true)  
List<Item> findByItemDetailByNative(@Param("itemDetail") String itemDetail);
```

## 6. Repository

---

### ■ Spring Data JPA Querydsl

- 동적 쿼리 생성
- 쿼리 재사용할 수 있어 제약조건 조립 및 가독성 향상
- 문자열이 아닌 자바 소스코드로 작성하기 때문에 컴파일 시점 오류 발견
- IDE 도움을 받아서 자동 완성 기능 사용 가능

## 6. Repository

### ▪ Dependency 설정 및 plugin 설정(pom.xml)

```
<dependency>
  <groupId>com.querydsl</groupId>
  <artifactId>querydsl-jpa</artifactId>
</dependency>

<dependency>
  <groupId>com.querydsl</groupId>
  <artifactId>querydsl-apt</artifactId>
</dependency>
```

```
<plugin>
  <groupId>com.mysema.maven</groupId>
  <artifactId>apt-maven-plugin</artifactId>
  <version>1.1.3</version>
  <executions>
    <execution>
      <goals>
        <goal>process</goal>
      </goals>
      <configuration>
        <outputDirectory>target/generated-sources/java</outputDirectory>
        <processor>com.querydsl.apt.jpa.JPAAnnotationProcessor</processor>
      </configuration>
    </execution>
  </executions>
</plugin>
```

## 6. Repository

- **ItemRepository 작성**

```
public interface ItemRepository extends JpaRepository<Item, Long> {  
    List<Item> findByNameOrItemDetail(String itemNm, String itemDetail);  
  
    List<Item> findByPriceLessThan(Integer price);  
  
    List<Item> findByPriceLessThanOrderByPriceDesc(Integer price);  
}
```

## 6. Repository

### ▪ SpringBoot Test

```
@SpringBootTest
@PropertySource(locations="classpath:application-test.properties")
public class ItemRepositoryTest {

    @Autowired
    ItemRepository itemRepository;

    @Test
    @DisplayName("상품저장 테스트")
    public void createItemTest() {
        Item item = new Item();
        item.setItemNm("테스트 상품");
        item.setPrice(10000);
        item.setItemDetail("테스트 상품 상세 설명");
        item.setItemSellStatus(ItemSellStatus.SELL);
        item.setStockNumber(100);
        item.setRegTime(LocalDateTime.now());
        item.setUpDate(LocalDateTime.now());
        Item savedItem = itemRepository.save(item);
        System.out.println(savedItem.toString());
    }
}
```

## 6. Repository

```
@Test  
@DisplayName("상품명 조회 테스트")  
public void findByItemNmTest(){  
    this.createItemList();  
    List<Item> itemList = itemRepository.findByItemNm("테스트 상품 1");  
    for(Item item : itemList){  
        System.out.println(item.toString()+"1111");  
    }  
}
```

```
@Test  
@DisplayName("상품명, 상품상세설명 or 테스트")  
public void findByItemNmOrItemDetailTest(){  
    this.createItemList();  
    List<Item> itemList = itemRepository.findByItemNmOrItemDetail("테스트 상품 1", "테스트 상품 설명 5");  
    System.out.println(itemList);  
    for(Item item : itemList){  
        System.out.println(item.toString());  
    }  
}
```

## 6. Repository

### ▪ Query 대소드 사용

```
@Test  
@DisplayName("가격 LessThan 테스트")  
public void findByPriceLessThanTest(){  
    this.createItemList();  
    List<Item> itemList = itemRepository.findByPriceLessThan(100005);  
    for(Item item : itemList){  
        System.out.println(item.toString());  
    }  
}
```

```
@Test  
@DisplayName("가격 내림차순 조회 테스트")  
public void findByPriceLessThanOrderByPriceDesc(){  
    this.createItemList();  
    List<Item> itemList = itemRepository.findByPriceLessThanOrderByPriceDesc(100005);  
    for(Item item : itemList){  
        System.out.println(item.toString());  
    }  
}
```

## 6. Repository

### ▪ Query 대소드 사용

```
public interface ItemRepository extends JpaRepository<Item, Long> {  
    ...  
  
    //Query를 이용한 상품 조회  
    @Query("select i from Item i where i.itemDetail like %:itemDetail% order by i.price desc")  
    List<Item> findByItemDetail(@Param("itemDetail") String itemDetail);  
  
    //nativeQuery속성을 이용한 상품 조회  
    @Query(value="select * from item i where i.item_detail like %:itemDetail% order by i.price desc", nativeQuery=true)  
    List<Item> findByItemDetailByNative(@Param("itemDetail") String itemDetail);  
}
```

## 6. Repository

- **Query를 이용한 상품 조회**

```
@Test  
@DisplayName("@Query를 이용한 상품 조회 테스트")  
public void findByItemDetailTest(){  
    this.createItemList();  
    List<Item> itemList = itemRepository.findByItemDetail("테스트 상품 상세 설명");  
    for(Item item : itemList){  
        System.out.println(item.toString());  
    }  
}
```

- **nativeQuery속성을 이용한 상품 조회**

```
@Test  
@DisplayName("nativeQuery속성을 이용한 상품 조회 테스트")  
public void findByItemDetailByNativeTest(){  
    this.createItemList();  
    List<Item> itemList = itemRepository.findByItemDetailByNative("테스트 상품 상세 설명");  
    for(Item item : itemList){  
        System.out.println(item.toString());  
    }  
}
```

## 6. Repository

- **Querydsl Test**를 위한 **Repository** 수정

- **QuerydslPredicateExecutor<Object>** 추가

```
public interface ItemRepository extends JpaRepository<Item, Long>, QuerydslPredicateExecutor<Item> {  
  
    List<Item> findByNameOrItemDetail(String itemNm, String itemDetail);  
  
    List<Item> findByPriceLessThan(Integer price);  
  
    List<Item> findByPriceLessThanOrderByPriceDesc(Integer price);  
  
    @Query("select i from Item i where i.itemDetail like %:itemDetail% order by i.price desc")  
    List<Item> findByItemDetail(@Param("itemDetail") String itemDetail);  
    @Query(value="select * from item i where i.item_detail like %:itemDetail% order by i.price desc", nativeQuery=true)  
    List<Item> findByItemDetailNative(@Param("itemDetail") String itemDetail);  
}
```

## 6. Repository

### ▪ Querydsl Test

- QuerydslPredicateExecutor 메소드

메소드	설명
long count(Predicate)	조건에 맞는 데이터의 총 개수를 반환
Boolean exists(Predicate)	조건에 맞는 데이터 존재 여부 반환
Iterable findAll(Predicate)	조건에 맞는 모든 데이터 반환
Page<T> findAll(Predicate, Pageable)	조건에 맞는 페이지 데이터 반환
Iterable findAll(Predicate, sort)	조건에 맞는 정렬된 데이터 반환
T findOne(Predicate)	조건에 맞는 1개의 데이터 반환

## 6. Repository

### ▪ Querydsl Test

```
import javax.persistence.EntityManager;
public class ItemRepositoryTest {
    @Autowired
    ItemRepository itemRepository;

    @PersistenceContext //EntityManager 빈 주입
    EntityManager em;

    @Test
    @DisplayName("Querydsl 조회 테스트 1")
    public void queryDslTest(){
        this.createItemList();
        JPAQueryFactory queryFactory = new JPAQueryFactory(em); //JPAQueryFactory를 이용하여 쿼리를 동적으로 생성
        QItem qItem = QItem.item; //Querydsl을 통해 쿼리를 생성하기 위해 플러그인을 통해 자동생성된 QItem 객체를 O
        JPAQuery<Item> query = queryFactory.selectFrom(qItem) //자바소스 쿼리 작성
            .where(qItem.itemSellStatus.eq(ItemSellStatus.SELL))
            .where(qItem.itemDetail.like("%" + "테스트 상품" + "%"))
            .orderBy(qItem.price.desc()); //쿼리 결과 반출

        List<Item> itemList = query.fetch(); //쿼리 결과 반출

        for(Item item : itemList){
            System.out.println(item.toString());
        }
    }
}
```

## 6. Repository

### ■ JPQL Query 데이터 반환메소드

메소드	설명
List<T> fetch()	조회 결과 리스트를 반환
T fetchOne()	조회 결과가 1건인 경우 T타입을 반환
T fetchFirst()	조회 대상 중 1건만 반환
Long fetchCount()	조회 대상 개수를 반환
QueryResult<T> fetchResults()	조회한 리스트의 전체 개수를 포함한 QueryResult를 반환

## 6. Repository

```
@DisplayName("Querydsl 조회 테스트를 위한 상품입력")
public void createItemList2(){
    for(int i=1;i<=5;i++){
        Item item = new Item();
        item.setItemNm("테스트 상품" + i);
        item.setPrice(10000 + i);
        item.setItemDetail("테스트 상품 상세 설명" + i);
        item.setItemSellStatus(ItemSellStatus.SELL);
        item.setStockNumber(100);
        item.setRegTime(LocalDateTime.now());
        item.setUpDate(LocalDateTime.now());
        itemRepository.save(item);
    }

    for(int i=6;i<=10;i++){
        Item item = new Item();
        item.setItemNm("테스트 상품" + i);
        item.setPrice(10000 + i);
        item.setItemDetail("테스트 상품 상세 설명" + i);
        item.setItemSellStatus(ItemSellStatus.SOLD_OUT);
        item.setStockNumber(0);
        item.setRegTime(LocalDateTime.now());
        item.setUpDate(LocalDateTime.now());
        itemRepository.save(item);
    }
}
```

## 6. Repository

```
@Test  
@DisplayName("상품 Querydsl 조회 테스트 2")  
public void queryDsITest2(){  
  
    this.createItemList2();  
  
    BooleanBuilder booleanBuilder = new BooleanBuilder();  
  
    QItem item = QItem.item;  
    String itemDetail = "테스트 상품 상세 설명";  
    int price = 10003;  
    String itemSellStat = "SELL";  
  
    booleanBuilder.and(item.itemDetail.like("%" + itemDetail + "%"));  
    booleanBuilder.and(item.price.gt(price));  
    System.out.println(itemSellStatus.SELL);  
    if(itemSellStat.equals(itemSellStatus.SELL)){  
        booleanBuilder.and(item.itemSellStatus.eq(itemSellStatus.SELL));  
    }  
  
    Pageable pageable = PageRequest.of(0, 5);  
    Page<Item> itemPagingResult = itemRepository.findAll(booleanBuilder, pageable);  
    System.out.println("total elements : " + itemPagingResult.getTotalElements());  
  
    List<Item> resultItemList = itemPagingResult.getContent();  
    for(Item resultItem: resultItemList){  
        System.out.println(resultItem.toString());  
    }  
}
```