6. 다음으로

#1.인강/jpa활용편/활용편2/강의

- /스프링 데이터 JPA 소개
- /QueryDSL 소개

스프링 데이터 JPA 소개

- https://spring.io/projects/spring-data-jpa

스프링 데이터 JPA는 JPA를 사용할 때 지루하게 반복하는 코드를 자동화 해준다. 이미 라이브러리는 포함되어 있다. 기존의 `MemberRepository`를 스프링 데이터 JPA로 변경해보자.

MemberRepository

```
package jpabook.jpashop.repository;

import jpabook.jpashop.domain.Member;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Repository;

import javax.persistence.EntityManager;
import java.util.List;

@Repository
@RequiredArgsConstructor
public class MemberRepository {

    private final EntityManager em;

    public void save(Member member) {
        em.persist(member);
    }

    public Member findOne(Long id) {
        return em.find(Member.class, id);
    }

    public List<Member> findAll() {
        return em.createQuery("select m from Member m", Member.class)
```

```
                .getResultList();
    }

    public List<Member> findByName(String name) {
        return em.createQuery("select m from Member m where m.name = :name",
Member.class)
                .setParameter("name", name)
                .getResultList();
    }
}
```

스프링 데이터 JPA 적용

```
package jpabook.jpashop.repository;

import jpabook.jpashop.domain.Member;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface MemberRepository extends JpaRepository<Member, Long> {
    List<Member> findByName(String name);
}
```

findOne() ➜ findById()로 변경해야 한다.

- 스프링 데이터 JPA는 `JpaRepository`라는 인터페이스를 제공하는데, 여기에 기본적인 CRUD 기능이 모두 제공된다. (일반적으로 상상할 수 있는 모든 기능이 다 포함되어 있다.)
- `findByName`처럼 일반화 하기 어려운 기능도 메서드 이름으로 정확한 JPQL 쿼리를 실행한다.
  - `select m from Member m where m.name = :name`
- 개발자는 인터페이스만 만들면 된다. 구현체는 스프링 데이터 JPA가 애플리케이션 실행시점에 주입해준다.

스프링 데이터 JPA는 스프링과 JPA를 활용해서 애플리케이션을 만들때 정말 편리한 기능을 많이 제공한다. 단순히 편리함을 넘어서 때로는 마법을 부리는 것 같을 정도로 놀라운 개발 생산성의 세계로 우리를 이끌어 준다.
하지만 **스프링 데이터 JPA는 JPA를 사용해서 이런 기능을 제공할 뿐이다. 결국 JPA 자체를 잘 이해하는 것이 가장 중요**하다.

> 참고: 스프링 데이터 JPA에 대한 내용은 분량이 상당하다. **실전! 스프링 데이터 JPA** 강의를 참고하자.

QueryDSL 소개

http://www.querydsl.com

실무에서는 조건에 따라서 실행되는 쿼리가 달라지는 동적 쿼리를 많이 사용한다.
주문 내역 검색으로 돌아가보고, 이 예제를 Querydsl로 바꾸어 보자.

Querydsl로 처리

```
public List<Order> findAll(OrderSearch orderSearch) {

    QOrder order = QOrder.order;
    QMember member = QMember.member;

    return query
            .select(order)
            .from(order)
            .join(order.member, member)
            .where(statusEq(orderSearch.getOrderStatus()),
                    nameLike(orderSearch.getMemberName()))
            .limit(1000)
            .fetch();
}

private BooleanExpression statusEq(OrderStatus statusCond) {
    if (statusCond == null) {
        return null;
    }
    return order.status.eq(statusCond);
}

private BooleanExpression nameLike(String nameCond) {
    if (!StringUtils.hasText(nameCond)) {
        return null;
    }
    return member.name.like(nameCond);
}
```

build.gradle에 querydsl 추가

```
//querydsl 추가
buildscript {
    dependencies {
        classpath("gradle.plugin.com.ewerk.gradle.plugins:querydsl-
plugin:1.0.10")
    }
}

plugins {
    id 'org.springframework.boot' version '2.1.9.RELEASE'
    id 'java'
}

apply plugin: 'io.spring.dependency-management'
apply plugin: "com.ewerk.gradle.plugins.querydsl"

group = 'jpabook'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-devtools'
    implementation 'com.fasterxml.jackson.datatype:jackson-datatype-hibernate5'

    implementation 'com.github.gavlyukovskiy:p6spy-spring-boot-starter:1.5.6'

    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'com.h2database:h2'

    annotationProcessor 'org.projectlombok:lombok'
```

```
    testImplementation 'org.springframework.boot:spring-boot-starter-test'

    //querydsl 추가
    implementation 'com.querydsl:querydsl-jpa'
    //querydsl 추가
    implementation 'com.querydsl:querydsl-apt'
}


//querydsl 추가
//def querydslDir = 'src/main/generated'
def querydslDir = "$buildDir/generated/querydsl"

querydsl {
    library = "com.querydsl:querydsl-apt"
    jpa = true
    querydslSourcesDir = querydslDir
}

sourceSets {
    main {
        java {
            srcDirs = ['src/main/java', querydslDir]
        }
    }
}

compileQuerydsl{
    options.annotationProcessorPath = configurations.querydsl
}

configurations {
    querydsl.extendsFrom compileClasspath
}
```

> **참고: 스프링 최신 버전에서는 Querydsl 설정이 달라진다. Querydsl 설정이 잘 안된다면 다음을 참고하자. 스프링 부트 2.x, 3.x 모두 참고할 수 있다.**
> https://bit.ly/springboot3

Querydsl은 SQL(JPQL)과 모양이 유사하면서 자바 코드로 동적 쿼리를 편리하게 생성할 수 있다.

실무에서는 복잡한 동적 쿼리를 많이 사용하게 되는데, 이때 Querydsl을 사용하면 높은 개발 생산성을 얻으면서 동시에 쿼리 오류를 컴파일 시점에 빠르게 잡을 수 있다.
꼭 동적 쿼리가 아니라 정적 쿼리인 경우에도 다음과 같은 이유로 Querydsl을 사용하는 것이 좋다.

- 직관적인 문법
- 컴파일 시점에 빠른 문법 오류 발견
- 코드 자동완성
- 코드 재사용(이것은 자바다)
- JPQL new 명령어와는 비교가 안될 정도로 깔끔한 DTO 조회를 지원한다.

Querydsl은 JPQL을 코드로 만드는 빌더 역할을 할 뿐이다. 따라서 JPQL을 잘 이해하면 금방 배울 수 있다.
Querydsl은 JPA로 애플리케이션을 개발 할 때 선택이 아닌 필수라 생각한다.

> 참고: Querydsl에 대한 내용은 분량이 상당하다. **실전! Querydsl** 강의를 참고하자.