**스프링 부트와 JPA 활용 – 1**

* **프로젝트 생성**
* Spring Initializer를 사용한 프로젝트 생성.

**1. Maven, Gradle 선택**

요즘에는 Maven 보다 Gradle을 많이 사용한다. 아직 레거시 프로젝트나 과거 프로젝트들은 아직 Maven로 남아있는게 많은데, Spring 라이브러리 관리 자체도 Gradle로 넘어왔다.

**2. 스프링 부트 버전 선택**

SNAPSHOT, M1 같은 것들은 정식 release된 버전들이 아니다.

**3. 프로젝트 메타데이터 입력**

Group, Artifact, Name, Description, Package name 등을 입력.

**4. Dependencies 설정**

스프링 부트 기반으로 프로젝트를 시작할 건데, 어떤 라이브러리를 가져와서 사용할지 정함. 검색하여 필요한 것을 선택함. 예) Spring Web Starter, Thymeleaf(Template Engines)

**5. Java 버전 & Packaging 방법 선택**

**6. GENERATE 버튼 눌러 프로젝트 설치 및 압축 해제**

**7. IntelliJ로 프로젝트 Open**

* 프로젝트 생성 시 추가시키는 라이브러리 종류 및 개념
* **Spring Web Starter**
* Spring MVC를 사용하여 RESTful한 웹 애플리케이션을 빌드할 수 있다.
* Apache Tomcat을 기본 임베디드 컨테이너로 사용.
* **Thymeleaf**
* 템플릿 엔진으로 스프링 프레임워크의 MVC 구조에서 V 즉, 뷰(View)를 담당하는 라이브러리.
* html파일을 가져와서 파싱과 분석 후 정해진 위치에 데이터를 치환해서 웹 페이지를 생성.
* 스프링 부트가 자동 설정을 지원하는 웹 템플릿 엔진.
* 작성된 페이지를 웹 브라우저로 열어보면 실제 보여질 내용과 동일하게 보여진다. 디자이너 또는 퍼블리셔가 페이지를 생성하거나 수정할 때 톰캣 같은 웹서버를 실행하지 않고 오프라인에서 수정을 할 수 가 있다는 장점이 있다.
* **Spring Data JPA**

1. JPA
   * Java Persistence API의 약자로, 자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스이다. 라이브러리가 아니다.
   * JPA는 기술 명세이다. 어떻게 사용해야 하는지 정의해 놓은 것일 뿐이다.
2. Hibernate
   * JPA라는 명세의 구현체이다. 즉, 인터페이스를 직접 구현한 라이브러리이다.
   * JPA를 구현하기 위해서 꼭 사용되는 것은 아니다. 언제든 다른 라이브러리를 사용하여 구현해도 된다.
3. Spring Data JPA
   * JPA를 쓰기 편하게 만들어 놓은 모듈이다.
   * JPA를 한 단계 추상화한 Repository 라는 인터페이스를 제공함으로써 이루어진다.
   * 개발자가 Repository 인터페이스에 정해진 규칙대로 메소드를 입력하면, Spring이 알아서 해당 메소드 이름에 적합한 쿼리를 날리는 구현체를 만들어서 Bean으로 등록해준다.

* **H2 Database**
* 자바 기반의 오픈소스 관계형 데이터 베이스 관리 시스템(RDBMS).
* 서버(Server) 모드와 임베디드(Embedded) 모드의 인메모리 DB 기능을 지원.
* 브라우저 기반의 콘솔모드를 이용할 수 있으며, 별도의 설치과정이 없고 용량도 2MB(압축버전) 이하로 매우 저용량.
* DB자체가 매우 가볍기 때문에 매우 가볍고 빠르며, JDBC API 또한 지원.
* 어플리케이션 개발 단계의 테스트 DB로서 많이 이용됨.
* **Lombok**
* Model Object를 만들 때, 멤버필드(프로퍼티)에 대한 Getter/Setter, ToString과 멤버필드에 주입하는 생성자를 만드는 코드 등 불필요하게 반복적으로 작성했던 코드들을 어노테이션을 통해 줄여 주는 라이브러리.
* **강의 내용과 상관없이 개념 정리가 필요한 것들 정리**
* **비즈니스 로직**

1. 데이터 처리를 수행하는 응용 프로그램의 일부.
2. 사용자에게 보여지는 것의 그 뒤에서 일어나는 일련의 데이터 처리 과정을 의미.
3. Ex)
4. 회원 가입 시 ID 작성 후 중복 확인.
5. 회원 정보가 있는 DB 접근.
6. DB에 해당 ID가 있는지 Select.
7. 있는지 없는지 여부를 데이터화 하여 저장.
8. DB 연결 해제.
9. View 영역으로 가공된 데이터 전달.
10. 화면 표시. (중복된 아이디입니다. 또는 사용가능한 아이디입니다.)

* **커넥션 풀(Connection pool)**

1. 스레드 풀과 마찬가지로 DB 커넥션 객체를 모아둔 pool.
2. 커넥션 객체 생성과 삭제에 드는 오버헤드를 방지하기 위한 객체 재활용 방식.
3. 커넥션 객체를 일정량 모아둔 Pool을 생성해두고 계속 재활용하는 방식.
4. HikariCP 라는 라이브러리가 있음.

* **자바빈즈(JavaBeans)**

1. 자바 클래스로써 값을 가지는 속성(멤버변수)과 값을 설정하는 메소드(Setter), 값을 추출하는 메소드(Getter)로 이루어져 있는 것.
2. 자바 관련 소프트웨어 개발에 있어 재사용이 가능한 표준 컴포넌트 모델.
3. JSP 페이지가 복잡한 자바 코드로 구성되는 것을 피하고 JSP 페이지에는 HTML과 같은 쉽고 간단한 코드만으로 구성되도록 할 수 있음.

* **DAO**

1. Data Access Object의 약자로써 실질적으로 DB에 접근하는 객체.
2. Database에 접근을 하기 위한 로직과 비즈니스 로직을 분리하기 위해서 사용.
3. 효율적인 커넥션 관리와 보안성을 위해 사용.
4. DAO는 Persistence layer이다. Persistence는 영속성이란 말 그대로 영구히 지속되는 성질인데, DAO는 DB(영구 저장소)에 직접 접근하여 data를 CRUD하기 때문이다.

* **DTO (= VO)**

1. Data Transfer Object의 약자로써 VO(Value Object)로 바꿔 말할 수 있는데 계층 간 데이터 교환을 위한 자바빈즈를 말함.
2. 여기서 말한 계층이라 함은 Controller, View, Business Layer, Persistent Layer를 말함.
3. DTO와 VO는 동일한 개념이지만 엄밀히 따지면 VO는 read only 특성을 가짐.
4. 로직을 갖고 있지 않은 순수한 데이터 객체이며, 속성과 그 속성에 접근하기 위한 getter, setter 메소드만 가진 클래스를 말함.

* **IntelliJ 단축키(Window)**
* **Ctrl + Alt + V**: 선택한 표현식을 기반으로 새 로컬 변수 또는 상수를 작성하고 표현식으로 초기화 한 다음 마지막으로 메소드에서 모든 표현식을 새로 도입된 변수에 대한 참조로 바꾸어 줌.
* **Ctrl + Shift + T**: 해당 class의 Test class 자동 생성.
* **Shift + F6**: 코드에서 어떤 변수의 이름을 변경하고자 할 때, 그 변수가 사용된 모든 부분에서 동시에 바꿀 수 있다.
* **Ctrl + Y**: 한 줄 지우고 공간 없애기.
* **Shift + Alt + 위, 아래 방향키**: 코드 위치 위아래 변경.
* **Alt + Insert**: 필요한 코드(getter, setter, constructor 등)를 선택하여 자동 생성해 준다.
* **Alt + Enter**: 현재 컨텍스트에 대한 액션을 표시하여 뭐든지 수정하는 데 사용된다.
* **F2**: 다음 오류, 경고 또는 제안으로 점프.
* **Ctrl + Alt + L**: 코드 정리.
* **Type sout** or **soutm** or **soutp** or **soutv** **+** **Tab**: print와 관련한 코드를 자동으로 생성.