



## 스터디 3주차(2021.08.29)

섹션 6. 스프링 DB 접근 기술

섹션 7. AOP

ORM : 객체와 RDB의 데이터를 매핑해주는 것

? Java는 클래스를 사용하고, RDB는 테이블을 사용하기에 모델 간 불일치 해결 필요

### JDBC와 JDBC 템플릿

JDBC : 자바로 DB를 접근할 수 있게 해주는 표준 인터페이스

JDBC 템플릿 : 스프링에서 제공하는 클래스로 JDBC API의 불편함을 해소

JDBC 템플릿을 이용해 할 수 있는 것

#### 1. 구조적인 반복의 해결

```
public void create() throws SQLException {  
    Connection connection = null;  
    PreparedStatement stmt = null;  
    try {  
        connection = dataSource.getConnection();  
        stmt = connection.prepareStatement("create table mytable (id int, name varchar(255))");  
        stmt.execute();  
    } catch (SQLException e) {  
        // 예외처리  
    } finally {  
        // 자원반납  
        if (stmt != null) stmt.close();  
        if (connection != null) connection.close();  
    }  
}
```

위 코드의 경우 실제 쿼리와 관련된 부분은 표시된 영역 한 줄 뿐임  
—> try-catch, 자원 반납 등 실제 작업과 무관한 코드가 많아 직관성 ↓

```
public void create() {
    jdbcTemplate.execute("create table mytable (id int, name varchar(255))");
}
```

JDBC 템플릿의 경우 작업과 무관한 코드는 템플릿 내부에서 처리함

## 2. 트랜잭션을 간단하게

```
connection.setAutoCommit(false);
connection.commit();
```

commit을 위한 코드(JDBC)

```
@Transactional
public void get() {
    // TODO
}
```

@Transactional annotation으로 해결 (JDBC 템플릿)

JDBC API만을 사용하게 되면 트랜잭션 시 commit 시 별도의 코드 작성 필요

—> 결국 작업과 무관한 코드가 작성되게 된다.

## 3. 데이터를 자바 객체로 매핑하기 쉬워진다.

```
stmt = connection.prepareStatement("select * from mytable where id=1");
rs = stmt.executeQuery();
if (rs.next()) {
    Person person = new Person(
        rs.getInt("id"),
        rs.getString("name")
    );
}
```

쿼리 결과를 resultSet에 받아와 next()를 통해 일일이 매핑을 시켜줌 (JDBC)

```
List<Person> persons = jdbcTemplate.query("select * from mytable where id=1",
    (resultSet, i) -> new Person(
        resultSet.getInt("id"),
        resultSet.getString("name")
    ));
```

맵퍼를 애용해 받아온 모든 데이터를 매핑 시켜줌, 객체를 어떻게 생성할 지에 대한 고려만 집중

하지만 JDBC는 SQL 쿼리를 써줘야 한다.

## JPA

자바의 ORM 기술 표준 API로 DB 관리 시 개발자가 직접 SQL을 작성하지 않도록 함

JPA는 내부적으로 JDBC API를 사용하며, JDBC와는 달리 객체를 자동으로 Mapping해준다.

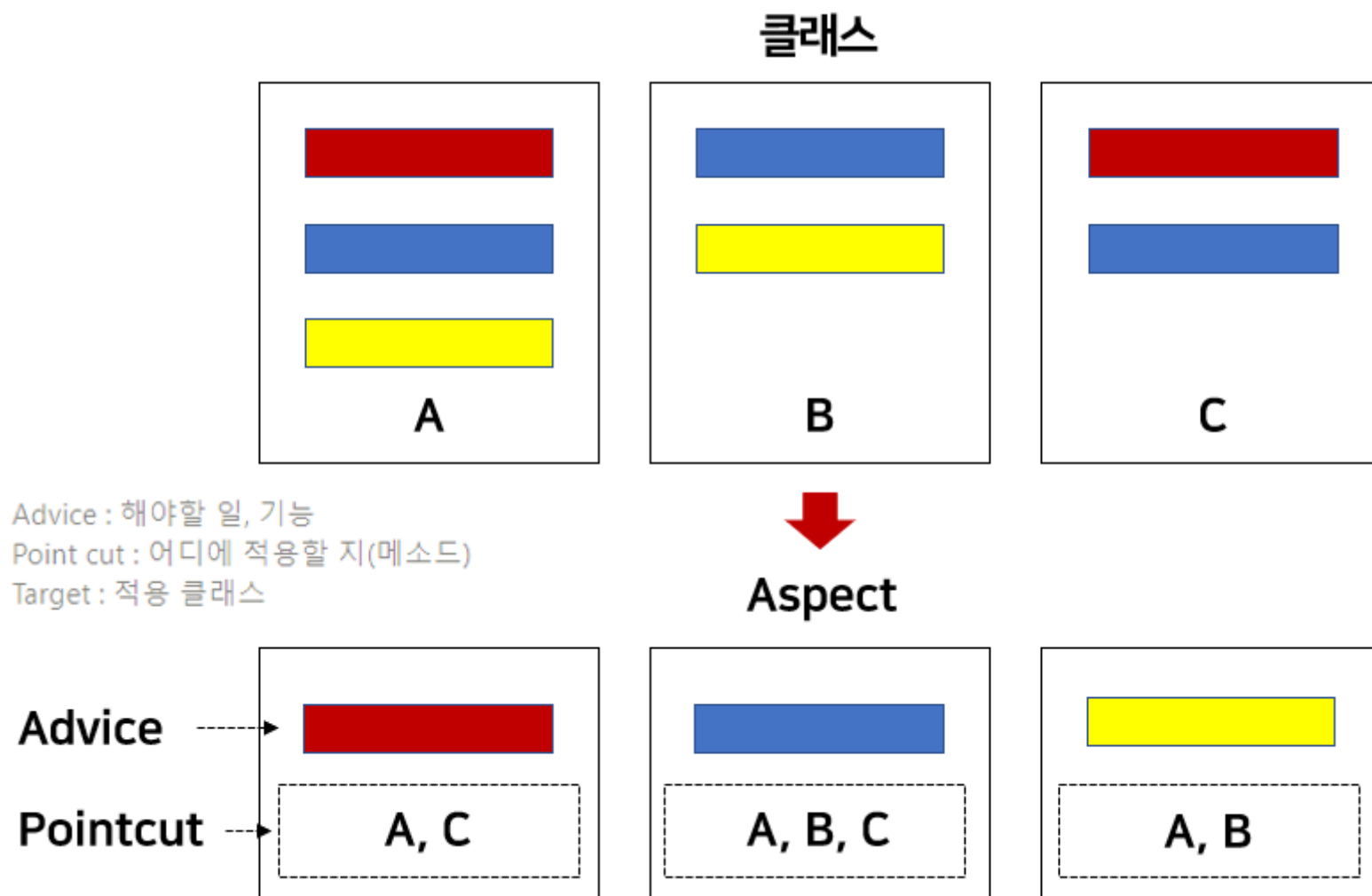
Hibernate : 가장 많이 사용되는 ORM 프레임워크

—> Hibernate 기반으로 만들어진 ORM 기술 표준이 바로 JPA

- 스프링 데이터 JPA는 스프링 프레임워크에서 JPA 사용을 보다 쉽게 할 수 있도록 함
- 조회의 경우 findByName을 통해 변수로 넘겨주면 된다.
- JPA 사용 시 entity manager가 만들어져 DB와 내부적으로 통신, JPA는 인터페이스에 대한 기본 구현체를 만들고 스프링 빈에 등록함

## AOP(Aspect-Oriented Programming)

: 흩어진 Aspect들을 모아서 모듈화 하는 기법



—> 서로 다른 클래스라도 비슷한 기능을 하는 부분(Cross-cutting concern)은 있는데, 이때 각각의 Concern을 Aspect로 만들어 주고 어느 클래스에 사용할 지 입력하는 방식

핵심 관심 사항(Core concern)이 아닌, 공통 관심 사항을 분리  
AOP는 Proxy를 통해 수행

## 스프링 핵심 원리

섹션 1 객체 지향 설계와 스프링

스프링 : 자바 엔터프라이즈 애플리케이션 개발에 사용되는 오픈소스 경량급 프레임워크

스프링부트 : 스프링 프레임워크를 편리하게 사용할 수 있도록 지원

✓ 객체 지향의 특징을 잘 살려내서 활용할 수 있다.

Tomcat과 같은 웹 서버 내장을 통해 별도 웹 서버 설치 필요 ✗

서드 파티 라이브러리 버전과 스프링 버전을 최적화 시켜줌

Starter를 이용해 dependency를 자동으로 추가해준다 —> 빌드 구성 용이

객체지향 특징



추상화(Abstraction), 다형성(Polymorphism), 상속(Inheritance), 캡슐화(Encapsulation)

—> 객체의 변경이 유연하고 용이하다

—> 역할과 구현의 자유성이 높다

역할 : 인터페이스 🚗

구현 : 인터페이스를 구현한 클래스, 구현 객체 🚚 🚛 🚓

역할과 구현을 명확히 분리하기에 유저는 구현에 대해 알 필요가 없다  
개발자는 구현 객체를 바꾸고 기능을 추가하여 인터페이스와 연결만 하면 된다  
✓ 즉, 클라이언트를 변경하지 않고도 서버 구현 기능을 유연하게 변경 가능하다  
! 때문에 초기 인터페이스를 안정적으로 설계하는 것은 정말 정말 중요하다

자바 언어의 다형성은 인터페이스에서 잘 쓰이며, 구현된 객체는 오버라이딩을 통해 유연하게 작동  
—> 인터페이스를 통해 구현한 객체 인스턴스는 실행 시점에 유연하게 변경할 수 있음  
! 스프링에서 제어의 역전(LoC), 의존 관계 주입(DI)는 이러한 다형성을 활용

좋은 객체 지향 설계의 SOLID 원칙

#### 1. SRP(단일 책임 원칙, Single Responsibility Principle)

💡 한 클래스는 하나의 책임만 가져야 한다

각각의 클래스에서 책임을 잘 조절해야 하며, 변경 시 파급 효과가 적다면 단일 책임 원칙(SRP)을 잘 따른 것이다

#### 2. OCP(개방-폐쇄 원칙, Open-Closed Principle) ★★★★★

💡 SW 요소는 확장에는 열려있으나 변경에는 닫혀 있어야 한다

다형성을 이용한 것으로, 변경은 없지만 기능은 추가될 수 있어야 함 (ex : 인터페이스 이용)  
—> 객체를 생성하고, 연관관계를 맺어주는 별도의 조립, 설정자가 필요함 (코드 변경 방지)

#### 3. LSP(리스코프 치환 원칙, Liskov Substitution Principle)

💡 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다

하위 클래스는 인터페이스의 규약을 다 지켜줘야하며, 컴파일 성공여부를 떠나서 기능적으로 보장

#### 4. ISP(인터페이스 분리 원칙, Interface Segregation Principle)

💡 특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다

인터페이스의 규모가 많이 커진다면 분리시키는 게 효율적일 수 있음  
—> 수정이 일어나거나 기능을 추가해도 영향을 주지 않기 때문에 명확성이 높아진다.

#### 5. DIP(의존관계 역전 원칙, Dependency Inversion Principle)

💡 추상화에 의존해야지, 구체화에 의존하면 안된다

구현 클래스가 아닌, 인터페이스에 의존하라는 뜻  
즉, 역할(인터페이스)에 의존하여 근본을 보는 것이 추후 변경에 더 용이할 수 있다

가장 중요한 것은 역할(인터페이스)과 기능(구현체)을 완벽하게 분리하는 것이다  
클라이언트가 인터페이스에 의존해야 구현체를 유연하게 변경할 수 있음  
구현체에 의존한다면 클라이언트가 내부 원리를 알아야 하는 경우가 발생할 수 있다

스프링은 이러한 관점에서 Dependency Injection(의존성 주입)을 통해 OCP와 DIP를 가능하게 함  
인터페이스를 잘 구현한다면, 하부 구현 기술에 대한 선택은 미룰 수 있음

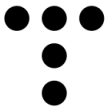
—> 하지만 인터페이스 구현은 추상화에 대한 리스크가 있기에 기능 확장 계획이 없다면 인터페이스 없이 구체 클래스로 구현한 뒤 추후 리팩토링

## Reference.

JdbcTemplate 와 JDBC API의 비교

Spring에서 제공하는 클래스로 JDBC API의 불편함을 해소시켜주는 역할을 한다. (템플릿 메서드 패턴/ 전략 패턴이 사용됨) JDBC API 는 자바로 데이터베이스에 접근할 수 있게 해주는 표준 인터페이스다.JAVA - JdbcTemplate - JDBC API - DB 의 단계로 생각할 수 있다. 구조적인 반복의 해결 JDBC API 를 사용하면 try-catch 문을 만들고 Connection -


📄 <https://soongjamm.tistory.com/134>



[Spring] AOP란?

Aspect-Oriented Programming의 약자이다. 흩어진 Aspect들을 모아서 모듈화 하는 기법이다. 서로 다른 클래스라고 하더라도 비슷한 기능을 하는 부분(ex 비슷한 메서드, 비슷한 코드)이있다. 이 부분을 Concern이라고 한다.(아래 색칠 되어 있는 부분) 이 때 만약 노란색 기능을 수정하여야하면, 각각 클래스의 노란색 기능을 수정해주어야 하기 때문에, 유지

📄 <https://velog.io/@max9106/Spring-AOP%EB%9E%80-93k5zjsm95>



## Source.

본 문서는 인프런의 스프링 핵심 원리 - 기본편(김영한) 강의를 수강하면서 정리한 내용입니다.

<https://www.infllearn.com/course/%EC%8A%A4%ED%94%84%EB%A7%81-%ED%95%B5%EC%8B%AC-%EC%9B%90%EB%A6%AC-%EA%B8%B0%EB%B3%B8%ED%8E%B8/dashboard>