[스파르타코딩클럽] Spring 심화반 - 5주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 5주차 오늘 배울 것

1. AOP 개념 이해 및 사용방법 파악
    1. '나만의 셀렉샵'의 Top5 회원 찾기
    2. 중복 폴더명 저장 시 에러 처리
2. DB 트랙잭션 이해
3. 스프링 예외 처리방법 이해

02. Top5 회원 찾기

▼ 1) 요구사항

1. '나만의 셀렉샵' 의 '고객 감사 이벤트'
    - Top5 회원들에게 선물 증정
2. Top5 회원의 기준??
    - '나만의 셀렉샵' 페이지에 머문 시간??
        - 웹 브라우져에 '나만의 셀렉샵' 페이지를 띄워두고 아무런 동작을 하지 않는다면?

- '나만의 셀렉샵' **서버 사용시간**으로 하기로 함
  - **서버 사용시간**: '나만의 셀렉샵' 모든 API 수행시간의 총합
    1. 상품 조회 API ("GET /api/search") 수행시간
    2. 관심상품 등록 API ("POST /api/products") 수행시간
    3. 폴더 저장 API ("POST /api/folders") 수행시간
    4. ...
  - 예) 회원 A 의 "서버 사용시간"
    - 상품 조회 API: **6시간**
    - 관심상품 등록 API: **3시간**
    - 폴더 저장 API: **1시간**
    - ⇒ 총합: **10시간**

3. 관리자만 "회원 별 API 수행시간" 조회 가능

▼ 2) API 수행시간 측정 방법

- 함수 기준 수행시간 측정 방법

  예) **totalSum()** 함수의 수행시간
  * totalSum() 함수: 1 에서 "입력된 숫자"까지의 합계를 구하는 함수

  ▼ **[코드스니펫] 함수의 수행시간 측정**

```
class Scratch {
    public static void main(String[] args) {
        // 측정 시작 시간
        long startTime = System.currentTimeMillis();

        // 함수 수행
        long output = totalSum(1_000_000_000);

        // 측정 종료 시간
        long endTime = System.currentTimeMillis();

        System.out.println("시작시간: " + startTime);
        System.out.println("종료시간: " + endTime);

        long runTime = endTime - startTime;
        System.out.println("소요시간: " + runTime);
    }

    private static long totalSum(long input) {
        long output = 0;

        for (int i = 1; i < input; ++i) {
            output = output * i;
        }

        return output;
    }
}
```

- API 수행시간 측정
  - 예) 상품 조회 API (SearchRequestController 클래스)

    AS-IS)

```
@GetMapping("/api/search")
public List<ItemDto> getItems(@RequestParam String query) {
    String resultString = naverShopSearch.search(query);
    return naverShopSearch.fromJSONtoItems(resultString);
}
```

TO-BE)

```
@GetMapping("/api/search")
public List<ItemDto> getItems(@RequestParam String query, @AuthenticationPrincipal UserDetailsImpl userDetails) {
  // 측정 시작 시간
  long startTime = System.currentTimeMillis();

  try {
      String resultString = naverShopSearch.search(query);
      return naverShopSearch.fromJSONtoItems(resultString);
  } finally {
      // 측정 종료 시간
      long endTime = System.currentTimeMillis();
      // 수행시간 = 종료 시간 - 시작 시간
      long runTime = endTime - startTime;
}
```

▼ 3) 회원별 총 API 수행시간 ⇒ DB 에 저장

▼ [코드스니펫] UserTime 클래스

```
package com.sparta.springcore.model;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Setter
@Getter // get 함수를 일괄적으로 만들어줍니다.
@NoArgsConstructor // 기본 생성자를 만들어줍니다.
@Entity // DB 테이블 역할을 합니다.
public class UserTime {
    // ID가 자동으로 생성 및 증가합니다.
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Id
    private Long id;

    @OneToOne
    @JoinColumn(nullable = false)
    private User user;

    @Column(nullable = false)
    private long totalTime;

    public UserTime(User user, long totalTime) {
        this.user = user;
        this.totalTime = totalTime;
    }

    public void updateTotalTime(long totalTime) {
        this.totalTime = totalTime;
    }
}
```

▼ [코드스니펫] UserTimeRepository 클래스

```
package com.sparta.springcore.repository;

import com.sparta.springcore.model.User;
import com.sparta.springcore.model.UserTime;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserTimeRepository extends JpaRepository<UserTime, Long> {
    UserTime findByUser(User user);
}
```

▼ [코드스니펫] SearchRequestContoller 클래스

```
package com.sparta.springcore.controller;
```

```
import com.sparta.springcore.dto.ItemDto;
import com.sparta.springcore.model.Product;
import com.sparta.springcore.model.User;
import com.sparta.springcore.model.UserTime;
import com.sparta.springcore.repository.UserTimeRepository;
import com.sparta.springcore.security.UserDetailsImpl;
import com.sparta.springcore.util.NaverShopSearch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController // JSON으로 응답함을 선언합니다.
public class SearchRequestController {

    private final NaverShopSearch naverShopSearch;

    private final UserTimeRepository userTimeRepository;

    @Autowired
    public SearchRequestController(NaverShopSearch naverShopSearch, UserTimeRepository userTimeRepository) {
        this.naverShopSearch = naverShopSearch;
        this.userTimeRepository = userTimeRepository;
    }

    @GetMapping("/api/search")
    public List<ItemDto> getItems(@RequestParam String query, @AuthenticationPrincipal UserDetailsImpl userDetails) {
        // 측정 시작 시간
        long startTime = System.currentTimeMillis();

        try {
            String resultString = naverShopSearch.search(query);
            return naverShopSearch.fromJSONtoItems(resultString);
        } finally {
            // 측정 종료 시간
            long endTime = System.currentTimeMillis();
            // 수행시간 = 종료 시간 - 시작 시간
            long runTime = endTime - startTime;
            // 로그인 회원 정보
            User loginUser = userDetails.getUser();

            // 수행시간 및 DB 에 기록
            UserTime userTime = userTimeRepository.findByUser(loginUser);
            if (userTime != null) {
                // 로그인 회원의 기록이 있으면
                long totalTime = userTime.getTotalTime();
                totalTime = totalTime + runTime;
                userTime.updateTotalTime(totalTime);
            } else {
                // 로그인 회원의 기록이 없으면
                userTime = new UserTime(loginUser, runTime);
            }

            System.out.println("[User Time] User: " + userTime.getUser().getUsername() + ", Total Time: " + userTime.getTotal
            userTimeRepository.save(userTime);
        }
    }
}
```

▼ 4) (관리자용) 회원별 총 API 수행시간 조회

- API 명: "GET /user/time"
- 관리자만 조회 가능
  - API 위에 `@Secured("ROLE_ADMIN")` 추가

▼ [코드스니펫] UserTimeController 클래스

```
package com.sparta.springcore.controller;

import com.sparta.springcore.dto.UserTimeDto;
import com.sparta.springcore.model.UserTime;
import com.sparta.springcore.repository.UserTimeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.annotation.Secured;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
```

```
import java.util.List;

@RestController
public class UserTimeController {
    private final UserTimeRepository userTimeRepository;

    @Autowired
    public UserTimeController(UserTimeRepository userTimeRepository) {
        this.userTimeRepository = userTimeRepository;
    }

    // (관리자용) 회원별 API 수행 조회
    @Secured("ROLE_ADMIN")
    @GetMapping("/user/time")
    public List<UserTimeDto> getUserTime() {
        List<UserTime> allUserTime = userTimeRepository.findAll();

        // UserTime -> UserTimeDto 로 변환
        List<UserTimeDto> allUserTimeDto = new ArrayList<>();
        for (UserTime userTime : allUserTime) {
            String username = userTime.getUser().getUsername();
            long totalTime = userTime.getTotalTime();
            UserTimeDto dto = new UserTimeDto(username, totalTime);
            allUserTimeDto.add(dto);
        }

        return allUserTimeDto;
    }
}
```

▼ **[코드스니펫] UserTimeDto 클래스**

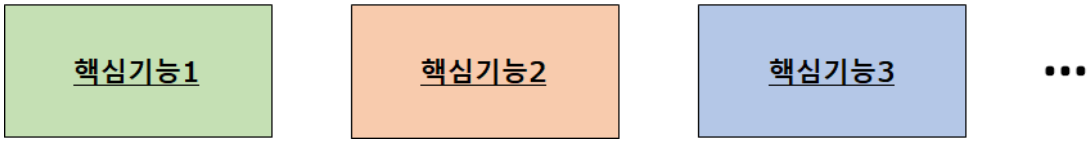
```
package com.sparta.springcore.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;

@AllArgsConstructor
@Getter
public class UserTimeDto {
    String username;
    long totalTime;
}
```

03. AOP 란?

▼ 5) 부가기능 모듈화의 필요성

- 'Top5 회원 찾기' 기능 추가 후
  - '나만의 셀렉샵' 회원들이 느끼는 변화??
- **'핵심기능'**: 각 API 별 수행해야 할 비즈니스 로직

  ex) 상품 키워드 검색, 관심상품 등록, 회원 가입, 상품에 폴더 추가, ...



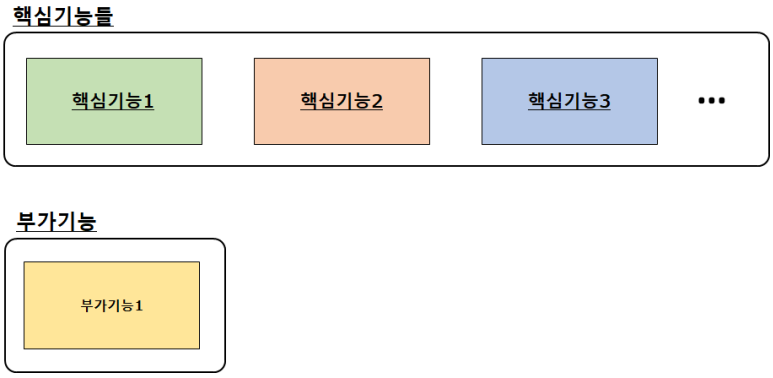
- '부가기능': 핵심기능의 수행시간을 기록


```
// 측정 시작 시간
long startTime = System.currentTimeMillis();

try {
  // 핵심기능 수행
  String resultString = naverShopSearch.search(query);
  return naverShopSearch.fromJSONtoItems(resultString);
} finally {
  // 측정 종료 시간
  long endTime = System.currentTimeMillis();
  // 수행시간 = 종료 시간 - 시작 시간
  long runTime = endTime - startTime;
  // 수행시간을 DB 에 기록
  ...
}
```

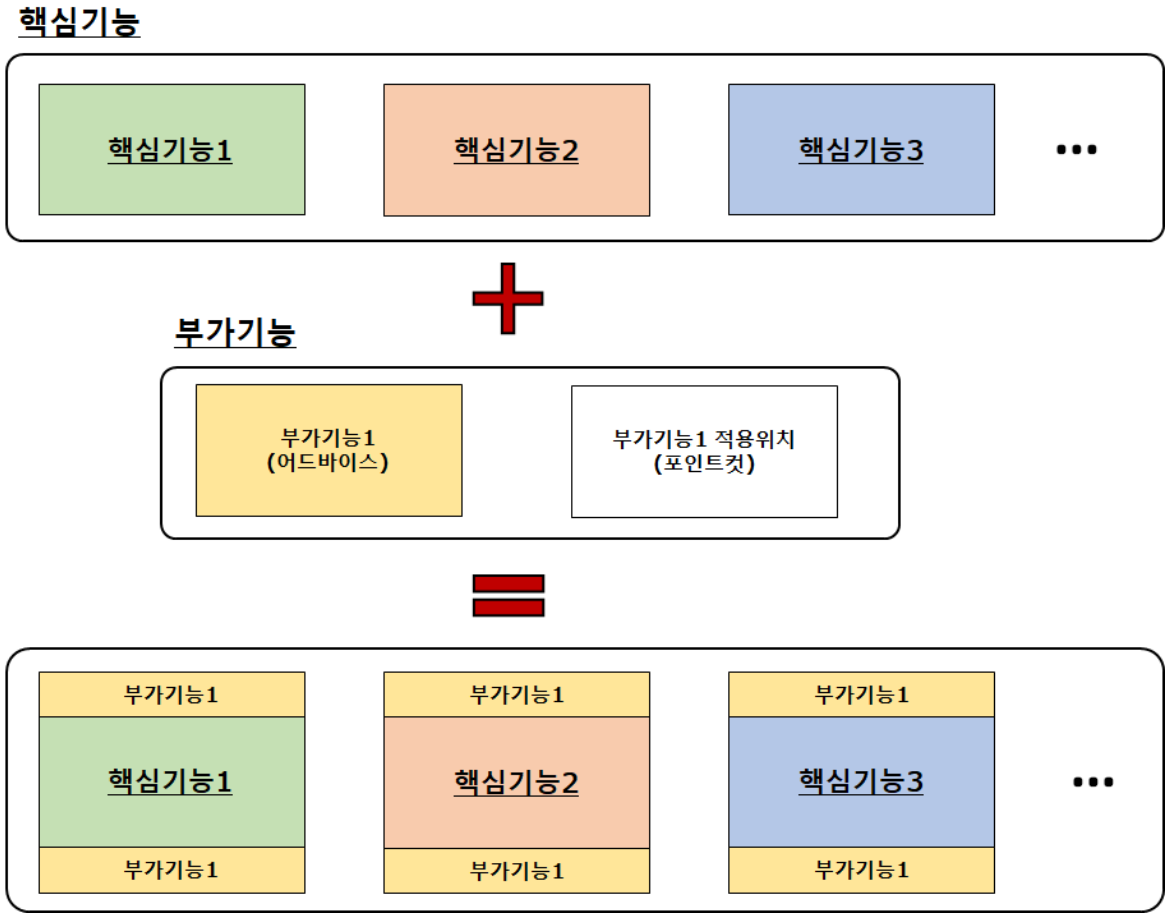
- 문제점
  - 모든 '**핵심기능**'의 Controller 에 '부가기능' 코드를 추가했을 때..
  - '**핵심기능**' 이 100개라면??
    - 100개의 '**핵심기능**' 모두에 동일한 내용의 코드 추가 필요
  - '**핵심기능**' 이 추가된다면?
    - 항상 '부가기능' 추가를 신경써야 함
    - '부가기능' 추가를 깜박한다면?
      - 일부 API 수행시간이 추가되지 않음 → Top5 회원의 신뢰성 이슈
  - '**핵심기능**' 수정 시
    - 같은 함수 내에 '**핵심기능**'과 '부가기능'이 섞여 있음
    - '**핵심기능**' 이해를 위해 '부가기능'까지 이해 필요
  - '부가기능'의 변경이 필요하다면??
    - '**핵심기능**'의 개수만큼 '부가기능'도 수정해 줘야 함

▼ 6) 부가기능을 모듈화

- AOP (Aspect Oriented Programming) 를 통해 부가기능을 모듈화
  - '**부가기능**'은 '핵심기능'과는 관점(Aspect), 관심이 다름
  - 따라서 '핵심기능'과 또옥~!! 분리해서 '**부가기능**' 중심으로 설계, 구현 가능



▼ 7) 스프링이 제공하는 AOP



- 어드바이스: 부가기능
- 포인트컷: 부가기능을 적용할 위치

04. 스프링 AOP 적용

▼ 8) 스프링 AOP 사용

- SearchRequestController 의 부가기능 제거

▼ **[코드스니펫] SearchRequestController 클래스**

```
package com.sparta.springcore.controller;

import com.sparta.springcore.dto.ItemDto;
import com.sparta.springcore.util.NaverShopSearch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController // JSON으로 응답함을 선언합니다.
public class SearchRequestController {

    private final NaverShopSearch naverShopSearch;

    @Autowired
    public SearchRequestController(NaverShopSearch naverShopSearch) {
        this.naverShopSearch = naverShopSearch;
    }

    @GetMapping("/api/search")
    public List<ItemDto> getItems(@RequestParam String query) {
        String resultString = naverShopSearch.search(query);
        return naverShopSearch.fromJSONtoItems(resultString);
    }
}
```

```
package com.sparta.springcore.controller;

import com.sparta.springcore.dto.ItemDto;
import com.sparta.springcore.util.NaverShopSearch;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController // JSON으로 응답함을 선언합니다.
public class SearchRequestController {

    private final NaverShopSearch naverShopSearch;

    @Autowired
    public SearchRequestController(NaverShopSearch naverShopSearch) {
        this.naverShopSearch = naverShopSearch;
    }

    @GetMapping("/api/search")
    public List<ItemDto> getItems(@RequestParam String query) {
        String resultString = naverShopSearch.search(query);
        return naverShopSearch.fromJSONtoItems(resultString);
    }
}
```

- AOP 사용해 모든 Controller 에 부가기능 추가

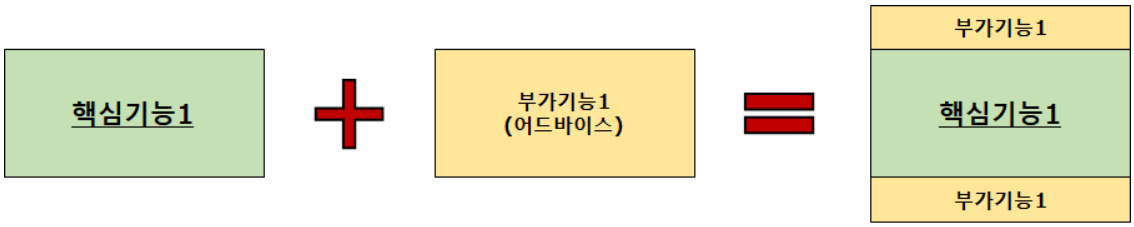
  ▼ [코드스니펫] aop > UserTimeAop 클래스

```
package com.sparta.springcore.aop;

import com.sparta.springcore.model.User;
import com.sparta.springcore.model.UserTime;
import com.sparta.springcore.repository.UserTimeRepository;
import com.sparta.springcore.security.UserDetailsImpl;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

@Component // 스프링 IoC 에 빈으로 등록
@Aspect
public class UserTimeAop {
    private final UserTimeRepository userTimeRepository;

    public UserTimeAop(UserTimeRepository userTimeRepository) {
        this.userTimeRepository = userTimeRepository;
```

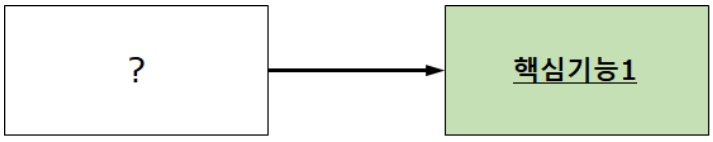
```
    }

    @Around("execution(public * com.sparta.springcore.controller..*(..))")
    public Object execute(ProceedingJoinPoint joinPoint) throws Throwable {
        // 측정 시작 시간
        long startTime = System.currentTimeMillis();

        try {
            // 핵심기능 수행
            Object output = joinPoint.proceed();
            return output;
        } finally {
            // 측정 종료 시간
            long endTime = System.currentTimeMillis();
            // 수행시간 = 종료 시간 - 시작 시간
            long runTime = endTime - startTime;
            // 로그인 회원이 없는 경우, 수행시간 기록하지 않음
            Authentication auth = SecurityContextHolder.getContext().getAuthentication();
            if (auth != null && auth.getPrincipal().getClass() == UserDetailsImpl.class) {
                // 로그인 회원 -> loginUser 변수
                UserDetailsImpl userDetails = (UserDetailsImpl) auth.getPrincipal();
                User loginUser = userDetails.getUser();

                // 수행시간 및 DB 에 기록
                UserTime userTime = userTimeRepository.findByUser(loginUser);
                if (userTime != null) {
                    // 로그인 회원의 기록이 있으면
                    long totalTime = userTime.getTotalTime();
                    totalTime = totalTime + runTime;
                    userTime.updateTotalTime(totalTime);
                } else {
                    // 로그인 회원의 기록이 없으면
                    userTime = new UserTime(loginUser, runTime);
                }

                System.out.println("[User Time] User: " + userTime.getUser().getUsername() + ", Total Time: " + userTime.
                userTimeRepository.save(userTime);
            }
        }
    }
}
```

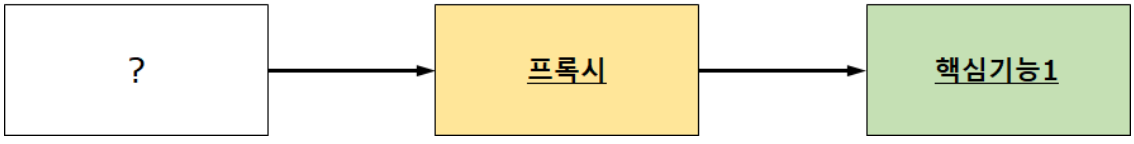
▼ 9) 스프링 AOP 이해

- 개념적 이해



- 스프링 실제 동작





- 스프링 서버가 기동될 때
  - 핵심 기능 DI 시

- 프록시 객체를 중간에 삽입

05. 중복 폴더 예외처리

▼ 10) 요구사항

폴더들을 저장 시, "중복 폴더명 저장 시" 에러를 발생시킴

예)

- 회원 username: **"르탄이"**
- **"르탄이"**의 저장 폴더명:
  1. 생필품
  2. 과자
  3. IT 기기
- 추가 요청된 폴더명
  1. 신발
  2. 옷
  3. **과자**
     - 중복된 폴더명이 존재!!

> 👉 요구사항에서 사용되는 용어 중 AS-IS, TO-BE 가 있습니다.
> AS-IS: 기존 동작
> TO-BE: 변경 동작

AS-IS)

- 중복을 제외한 모든 폴더명이 저장됨
- **"르탄이"**의 저장 폴더명:
  1. 생필품
  2. 과자
  3. IT 기기
  4. 신발
  5. 옷

TO-BE)

- Alert 팝업을 띄워서 사용자가 폴더명을 수정할 수 있도록 유도!!
  - 중복된 폴더명 표시
- 저장 시도한 폴더 모두 저장되지 않음
- **"르탄이"**의 저장 폴더명:
  1. 생필품
  2. 과자
  3. IT 기기
  4. ~~신발~~ *(저장되지 않음!)*
  5. ~~옷~~ *(저장되지 않음!)*

▼ 11) 현재 동작 확인 (with 디버깅)

FolderService.java 에서 createFolders() 함수

▼ 12) 해결방법 (1)

- 예외 발생 시, 그동안 DB 에 저장된 폴더들을 삭제
- FolderService.java 에서 createFolders() 함수

▼ [코드스니펫] 예외 발생 시 저장된 폴더들 삭제

```
public List<Folder> createFolders(List<String> folderNameList, User user) {
    List<Folder> folderList = new ArrayList<>();

    for (String folderName : folderNameList) {
        // 1) DB 에 폴더명이 folderName 인 폴더가 존재하는지?
        Folder folderInDB = folderRepository.findByName(folderName);
        if (folderInDB != null) {
            // 그동안 저장된 폴더들을 모두 삭제!
            for (Folder folder : folderList) {
                folderRepository.delete(folder);
            }

            // DB 에 중복 폴더명 존재한다면 Exception 발생시킴
            throw new IllegalArgumentException("중복된 폴더명 (" + folderName +") 을 삭제하고 재시도해 주세요!");
        }

        // 2) 폴더를 DB 에 저장
        Folder folder = new Folder(folderName, user);
        folder = folderRepository.save(folder);

        // 3) folderList 에 folder Entity 객체를 추가
        folderList.add(folder);
    }

    return folderList;
}
```

▼ 13) 해결방법 (2)

- 트랜잭션 (@Transactional) 을 이용

▼ [코드스니펫] 트랜잭션 이용

```
@Transactional
public List<Folder> createFolders(List<String> folderNameList, User user) {
    List<Folder> folderList = new ArrayList<>();

    for (String folderName : folderNameList) {
        // 1) DB 에 폴더명이 folderName 인 폴더가 존재하는지?
        Folder folderInDB = folderRepository.findByName(folderName);
        if (folderInDB != null) {
            // DB 에 중복 폴더명 존재한다면 Exception 발생시킴
            throw new IllegalArgumentException("중복된 폴더명 (" + folderName +") 을 삭제하고 재시도해 주세요!");
        }

        // 2) 폴더를 DB 에 저장
        Folder folder = new Folder(folderName, user);
        folder = folderRepository.save(folder);

        // 3) folderList 에 folder Entity 객체를 추가
        folderList.add(folder);
    }

    return folderList;
}
```

06. 트랜잭션의 이해

▼ 14) 트랜잭션이란?

> 트랜잭션: 데이터베이스에서 데이터에 대한 하나의 논리적 실행단계
>
> ACID (원자성, 일관성, 고립성, 지속성)는 데이터베이스 트랜잭션이 안전하게 수행된다는 것을 보장하기 위한 성질을 가리키는 약어
>
> 출처: 위키백과

- 트랜잭션의 특징
  - 더 이상 쪼갤 수 없는 최소단위의 작업
  - 모두 저장되거나, 아무 것도 저장되지 않거나를 보장!!
  - **모!** 아니면 **도!!**

▼ 15) 데이터 1개 저장 시

1. 회원 등록 요청

```
User user = new User(username, password, email, role);
user = userRepository.save(user);
```

2. 만약, DB 저장 시 에러로 다음과 같이 저장된다면??
   - 저장 요청 DATA
     - USERNAME: **"삼식이"**
     - PASSWORD: **"$2a^A..."** *(암호화된 패스워드)*
     - EMAIL: **"sameat@sparta.com"**
     - ROLE: **"USER"**
   - 실제 DB 에 저장된 DATA

ID	USERNAME	PASSWORD	EMAIL	ROLE
1	삼식이		sameat@sparta.com	ADMIN

3. 회원 저장 요청이 1개의 트랜잭션
   - DB 는 1개의 회원 정보가 안전하게 저장됨을 보장

▼ 16) 데이터 2개 이상 저장 시

A 계좌→ B계좌로 200,000 원 이체 시

- A 계좌 잔고: 1,000,000 원
- B 계좌 잔고: 1,000,000 원

정상 케이스

1. A 계좌 잔고 **200,000 원** 이상 확인
   - A 계좌 잔고: 1,000,000 원
2. A 계좌 잔고 **200,000 원** 금액 감소
   - A 계좌 잔고: 800,000 원 (1,000,000 원 - 200,000 원)
3. B 계좌 잔고 **200,000 원** 금액 증가
   - B 계좌 잔고: 1,200,000 원 (1,000,000 원 + 200,000 원)

만약, 3번 과정에서 에러가 발생 시

- A 계좌 잔고: **800,000 원 ??**
- B 계좌 잔고: 1,000,000 원 ??

▼ 17) 데이터 2개 이상 저장 시 (with 트랜잭션)

트랜잭션 시작

1. A 계좌 잔고 **200,000 원** 이상 확인
    - A 계좌 잔고: 1,000,000 원
2. A 계좌 잔고 **200,000 원** 금액 감소
    - A 계좌 잔고: 800,000 원 (1,000,000 원 - 200,000 원)
3. B 계좌 잔고 **200,000 원** 금액 증가
    - B 계좌 잔고: 1,200,000 원 (1,000,000 원 + 200,000 원)

모두 성공 시 ⇒트랜잭션 Commit

중간에 하나라도 실패 시 ⇒ 트랜잭션 Rollback

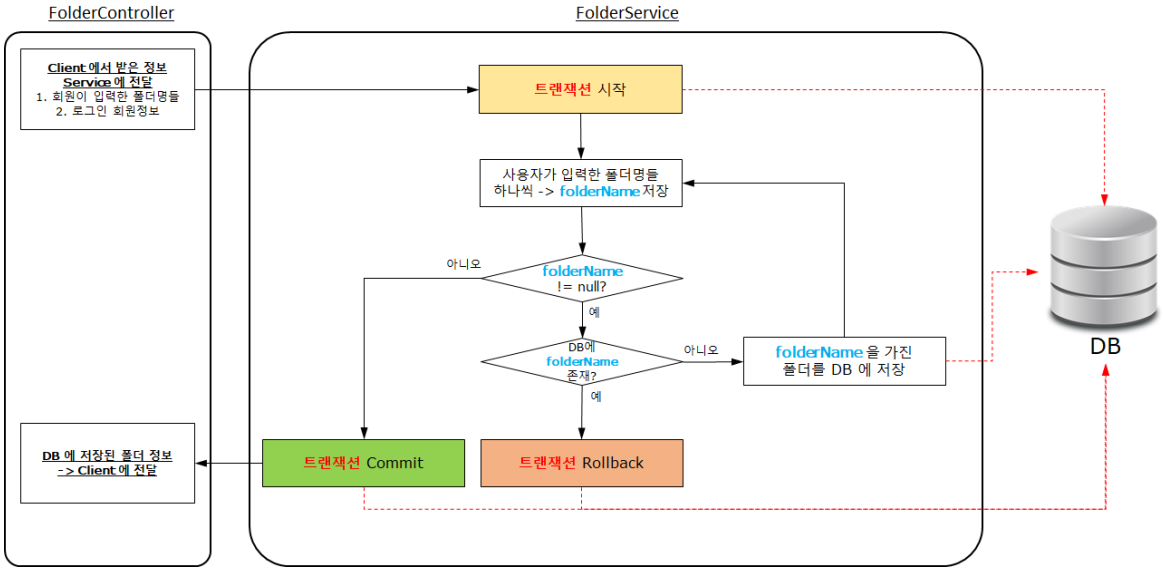
07. @Transactional 의 정체

▼ 18) 트랜잭션을 사용한 폴더 생성 코드

```
public List<Folder> createFolders(List<String> folderNameList, User user) {
    // 트랜잭션의 시작
    TransactionStatus status = this.transactionManager.getTransaction(new DefaultTransactionDefinition());

    try {
        List<Folder> folderList = new ArrayList<>();

        for (String folderName : folderNameList) {
            // 1) DB 에 폴더명이 folderName 인 폴더가 존재하는지 확인
            Folder folderInDB = folderRepository.findByName(folderName);
            if (folderInDB != null) {
                // DB 에 중복 폴더명 존재한다면 Exception 발생시킴
                throw new IllegalArgumentException("중복된 폴더명 (" + folderName + ") 을 삭제하고 재시도해 주세요!");
            }

            // 2) 폴더를 DB 에 저장
            Folder folder = new Folder(folderName, user);
            folder = folderRepository.save(folder);

            // 3) folderList 에 folder Entity 객체를 추가
            folderList.add(folder);
        }

        // 트랜잭션 commit
        transactionManager.commit(status);

        return folderList;

    } catch (Exception ex) {
        // 트랜잭션 rollback
        transactionManager.rollback(status);
        throw ex;
    }
}
```

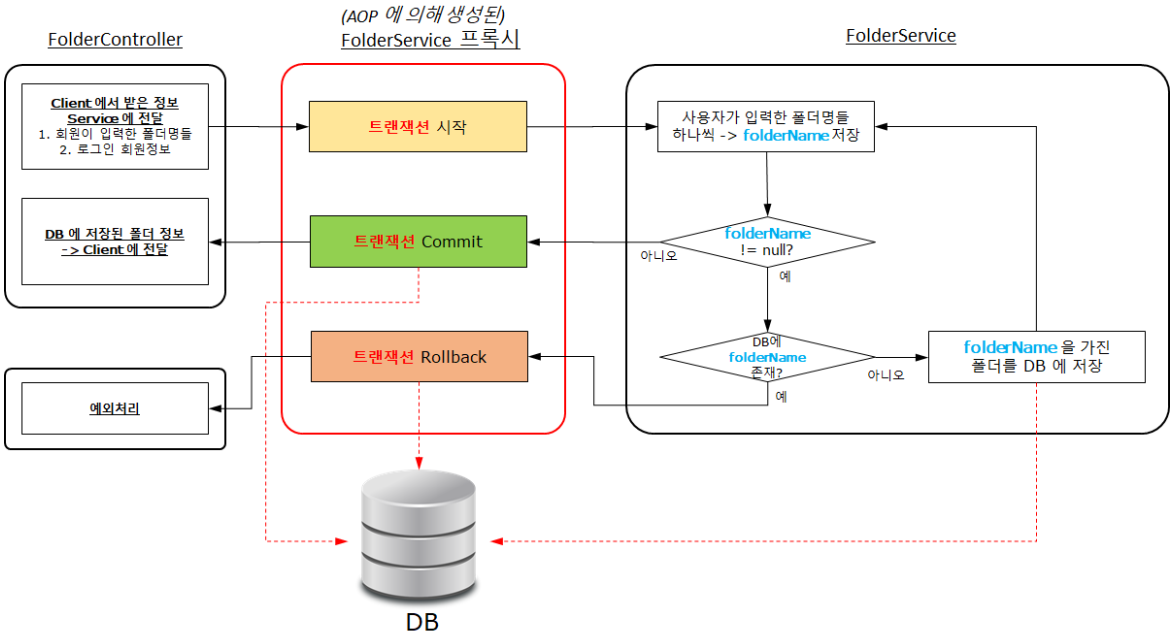
▼ 19) 트랜잭션을 사용한 폴더 생성 Flowchart

- 비즈니스 로직에 트랜잭션 코드가 포함됨



▼ 20) @Transactional 사용 시 폴더 생성 Flowchart

```
@Transactional
public List<Folder> createFolders(List<String> folderNameList, User user) {
  // ...
}
```



▼ 21) Quiz

다음과 같이 코드를 리팩토링 했는데 문제점이 생겼습니다. 무엇이 문제점일까요?

```
public List<Folder> createFolders(List<String> folderNameList, User user) {
    List<Folder> folderList = new ArrayList<>();

    for (String folderName : folderNameList) {
        // 1) folderName 이 DB 에 존재하지 않는 경우만 폴더 생성 (존재하면 Exception 발생)
        Folder folder = createFolderIfNotExisted(user, folderName);
```

```
            // 2) folderList 에 folder Entity 객체를 추가
            folderList.add(folder);
        }

        return folderList;
    }

    @Transactional
    public Folder createFolderIfNotExisted(User user, String folderName) {
        // 1) DB 에 폴더명이 folderName 인 폴더가 존재하는지?
        Folder folderInDB = folderRepository.findByName(folderName);
        if (folderInDB != null) {
            // DB 에 중복 폴더명 존재한다면 Exception 발생시킴
            throw new IllegalArgumentException("중복된 폴더명 (" + folderName + ") 을 삭제하고 재시도해 주세요!");
        }

        // 2) 폴더를 DB 에 저장
        Folder folder = new Folder(folderName, user);
        folder = folderRepository.save(folder);
        return folder;
    }
```

▼ 22) Quiz 정답

DB 에 이미 저장된 폴더명들이 Rollback 되지 않음

- 이유: 하나의 folder 생성마다 하나의 트랜잭션으로 처리되기 때문

08. 현업에서 DB 운영 방식 (Primary, Replica)

▼ 23) 현업에서 DB 운영방식

- 웹 서비스에서 DB 에 담겨있는 Data 는 소중한 재산
  - 회원 정보
  - 서비스 이용 정보
- DB 훼손 가능성
  - DB 도 결국 물리적인 HDD (하드 디스크) 에 존재
  - DB 가 저장된 하드디스크 고장
  - DB 가 저장된 컴퓨터 고장
- 현업에서 DB 를 1대 이상 운영

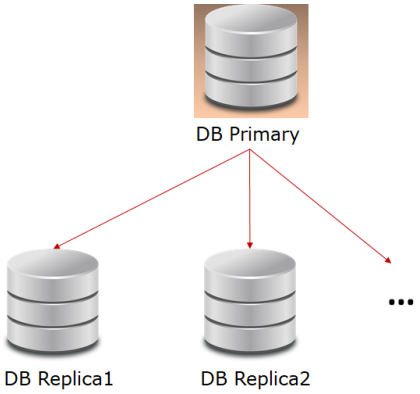
  - 문제점: DB1 과 DB2 를 어떻게 데이터 Sync 를 하지??
  - 예제를 통한 이해
    - 회원 A 계좌 잔고: 100만원
      - DB1: 100만원
      - DB2: 100만원

    1. 70만원 인출 시도
       1. DB1 에서 70만원 인출하여 잔고 30만원
       2. DB2 에서도 동일하게 잔고 30만원으로 데이터 Sync 필요!
    2. 50만원 인출 시도
       1. DB1 를 통해 잔고 확인 시

        - 회원 A 의 "잔고 30만원"이기 때문에, 인출불가 에러 발생
    2. DB2 를 통해 잔고 확인 시
        - DB2 에 "첫번째 인출 시도한 데이터(70만원)"가 Sync 적용되기 전이라고 가정
            - 혹은 DB2 에 데이터 Sync 중 에러가 발생했다고 가정
        - DB2 에서는 회원 A의 "잔고 100만원" 이 남아 있다고 판단
        - 50만원이 정상 인출됨
3. DB1 과 DB2 의 데이터 불일치 ⇒ 어느 정보를 믿어야하지?
    1. DB1 에서 회원 A 의 잔고: 30만원
    2. DB2 에서 회원 B 의 잔고: 50만원

▼ 24) Primary / Replica 운영방식

- 쓰기 전용 DB (Primary) 와 읽기 전용 DB (Replica) 를 구분



- Primary: 쓰기 전용
    - @Transactional 의 readOnly 속성

```
@Transactional(readOnly = false)
```

    - readOnly 를 코드에 적지 않으면, 기본값은 false

```
import org.springframework.transaction.annotation.Transactional;

@Transactional
public List<Folder> createFolders(List<String> folderNameList, User user) {
```

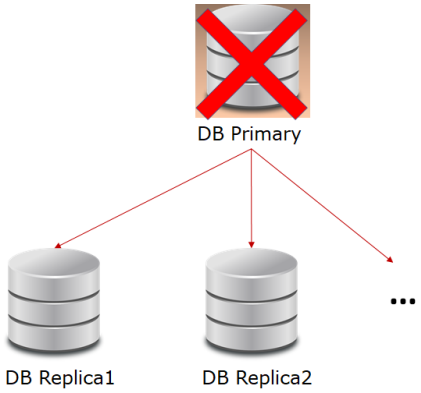
    - Write 된 Data (Create, Update, Delete) 가 Replica 로 Sync 됨 (Replication)
- Replica (Secondary): 읽기 전용

```
@Transactional(readOnly = true)
```

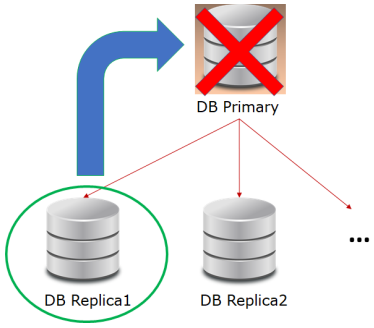
> 수십년 동안 통용되던 용어가 '노예제'와 관련된다는 이유로 대체됨
>
> 마스터 (Master) → Primary
> 슬레이브 (Slave) → Replica, Secondary

> 출처: 위키백과

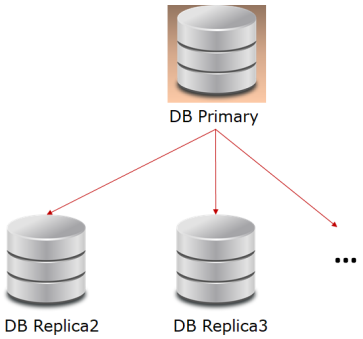
▼ 25) Primary 에 문제가 생겼을 때



- Replica 중 1개가 Primary 가 됨



- 다시 Primary - Replica 로 정상 운영



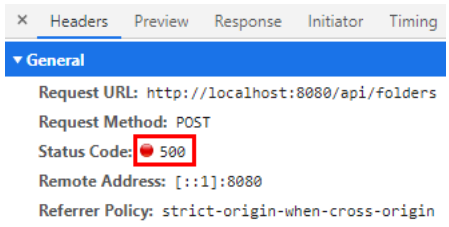
09. 스프링 예외 처리 방법

▼ 26) 스프링 기본 에러 처리

1. DB 에 중복된 폴더명 존재 시 Exception 발생

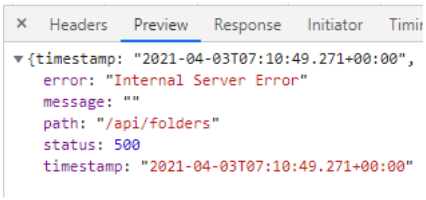
```
// DB 에 중복 폴더명 존재한다면 Exception 발생시킴
throw new IllegalArgumentException("중복된 폴더명 (" + folderName + ") 을 삭제하고 재시도해 주세요!");\
```

2. 클라이언트에서는 에러 메시지를 받지 못하고 있음
    - Headers



> 👉 **HTTP 500** Internal Server Error
> Link: MDN Web Docs

    - Body



▼ 27) 스프링 예외처리 방법

▼ **[코드스니펫] basic.js**

```
function addFolder() {
    const folderNames = $('.folderToAdd').toArray().map(input => input.value);
    folderNames.forEach(name => {
        if (name == '') {
            alert('올바른 폴더명을 입력해주세요');
            return;
        }
    })
    $.ajax({
        type: "POST",
        url: `/api/folders`,
        contentType: "application/json",
        data: JSON.stringify({
            folderNames
        }),
        success: function (response) {
            $('#container2').removeClass('active');
            alert('성공적으로 등록되었습니다.');
            window.location.reload();
        },
        error: function (response) {
            // 서버에서 받은 에러 메시지를 노출
            if (response.responseJSON && response.responseJSON.message) {
                alert(response.responseJSON.message);
            } else {
                alert("알 수 없는 에러가 발생했습니다.");
            }
        }
    })
}
```

▼ **[코드스니펫] ApiException.java**

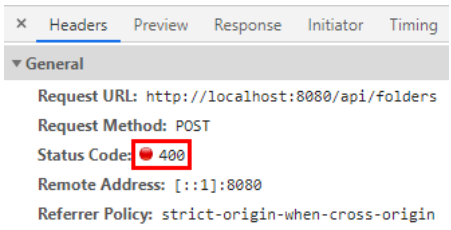
```
package com.sparta.springcore.exception;
```

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import org.springframework.http.HttpStatus;

@AllArgsConstructor
@Getter
public class ApiException {
    private final String message;
    private final HttpStatus httpStatus;
}
```

▼ **[코드스니펫] FolderController.java**

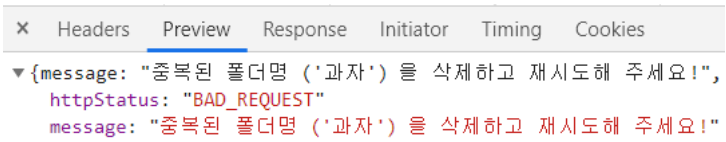
```
    @ExceptionHandler({ IllegalArgumentException.class })
    public ResponseEntity<Object> handle(IllegalArgumentException ex) {
        ApiException apiException = new ApiException(
                ex.getMessage(),
                // HTTP 400 -> Client Error
                HttpStatus.BAD_REQUEST
        );

        return new ResponseEntity<>(
                apiException,
                HttpStatus.BAD_REQUEST
        );
    }
```

- 변경된 결과
  - Headers



> 👉 **HTTP 400** Bad Request
> Link: MDN Web Docs

  - Body



▼ 28) Global 예외처리 방법

▼ **[코드스니펫] ApiRequestException.java**

```
package com.sparta.springcore.exception;

public class ApiRequestException extends IllegalArgumentException {
    public ApiRequestException(String message) {
        super(message);
    }

    public ApiRequestException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

▼ **[코드스니펫] FolderService.java**

```
    @Transactional(readOnly = false)
    public List<Folder> createFolders(List<String> folderNameList, User user) {

        List<Folder> folderList = new ArrayList<>();

        for (String folderName : folderNameList) {
            // 1) DB 에 폴더명이 folderName 인 폴더가 존재하는지?
            Folder folderInDB = folderRepository.findByName(folderName);
            if (folderInDB != null) {
                // DB 에 중복 폴더명 존재한다면 Exception 발생시킴
                throw new ApiRequestException("중복된 폴더명 ('" + folderName + "') 을 삭제하고 재시도해 주세요!");
            }

            // 2) 폴더를 DB 에 저장
            Folder folder = new Folder(folderName, user);
            folder = folderRepository.save(folder);

            // 3) folderList 에 folder Entity 객체를 추가
            folderList.add(folder);
        }

        return folderList;
    }
```

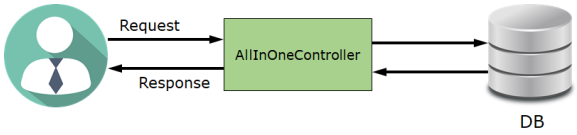
▼ **[코드스니펫] ApiExceptionHandler.java**

```
package com.sparta.springcore.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class ApiExceptionHandler {

    @ExceptionHandler(value = { ApiRequestException.class })
    public ResponseEntity<Object> handleApiRequestException(ApiRequestException ex) {
        ApiException apiException = new ApiException(
                ex.getMessage(),
                // HTTP 400 -> Client Error
                HttpStatus.BAD_REQUEST
        );

        return new ResponseEntity<>(
                apiException,
                // HTTP 400 -> Client Error
                HttpStatus.BAD_REQUEST
        );
    }
}
```

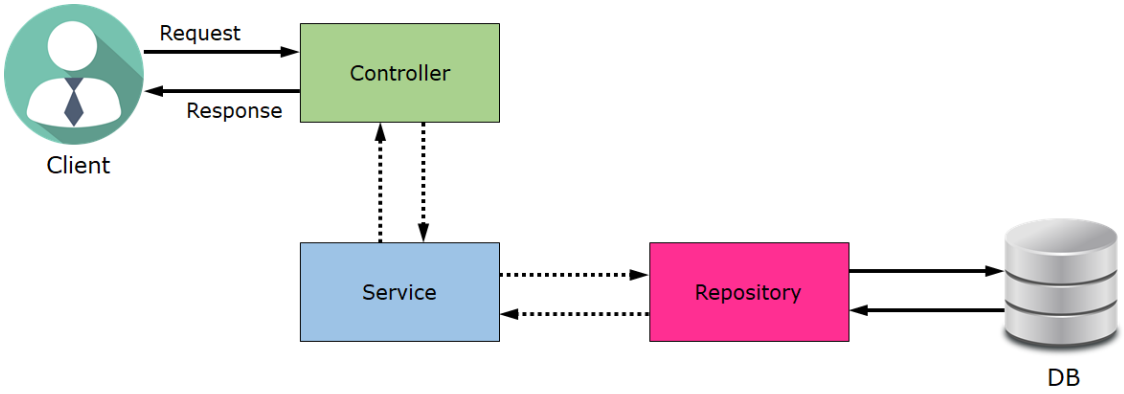
10. 5주차 끝 & 숙제 설명

▼ 29) OOP 복습

- 객체지향 프로그래밍 (OOP: Object-Oriented Programming)
- 하나의 클래스로 모든 기능 수행 가능



- 관심사별로 분리 ⇒ 코드 이해와 유지보수 수월



- 필요 시 관심사 별로 더 분리 가능
  - ProductService 의 createProduct() 함수

```
@Transactional // 메소드 동작이 SQL 쿼리문임을 선언합니다.
public Product createProduct(ProductRequestDto requestDto, Long userId) {
    // 요청받은 DTO 로 DB에 저장할 객체 만들기
    Product product = new Product(requestDto, userId);
    productRepository.save(product);
    return product;
}
```

  - Product 생성자

```
public Product(ProductRequestDto requestDto, Long userId) {
    // 입력값 Validation
    if (userId == null || userId < 0) {
        throw new IllegalArgumentException("회원 Id 가 유효하지 않습니다.");
    }

    if (requestDto.getTitle() == null || requestDto.getTitle().isEmpty()) {
        throw new IllegalArgumentException("저장할 수 있는 상품명이 없습니다.");
    }

    if (!URLValidator.urlValidator(requestDto.getImage())) {
        throw new IllegalArgumentException("상품 이미지 URL 포맷이 맞지 않습니다.");
    }

    if (!URLValidator.urlValidator(requestDto.getLink())) {
        throw new IllegalArgumentException("상품 최저가 페이지 URL 포맷이 맞지 않습니다.");
    }

    if (requestDto.getLprice() <= 0) {
        throw new IllegalArgumentException("상품 최저가가 0 이하입니다.");
    }

    // 관심상품을 등록한 회원 Id 저장
    this.userId = userId;
    this.title = requestDto.getTitle();
    this.image = requestDto.getImage();
    this.link = requestDto.getLink();
    this.lprice = requestDto.getLprice();
    this.myprice = 0;
}
```

  - ProductValidator 생성하여 분리 가능
    - ProductValidator: 관심상품을 Valiation 하는 것에만 관심이 있는 클래스

```
@Bean
class ProductValidator {
  // 관심상품 생성 validation
  public void validateCreate(ProductRequestDto requestDto, Long userId) {
```

```
    if (userId == null || userId < 0) {
        throw new IllegalArgumentException("회원 Id 가 유효하지 않습니다.");
    }

    if (requestDto.getTitle() == null || requestDto.getTitle().isEmpty()) {
        throw new IllegalArgumentException("저장할 수 있는 상품명이 없습니다.");
    }

    if (!URLValidator.urlValidator(requestDto.getImage())) {
        throw new IllegalArgumentException("상품 이미지 URL 포맷이 맞지 않습니다.");
    }

    if (!URLValidator.urlValidator(requestDto.getLink())) {
        throw new IllegalArgumentException("상품 최저가 페이지 URL 포맷이 맞지 않습니다.");
    }

    if (requestDto.getLprice() <= 0) {
        throw new IllegalArgumentException("상품 최저가가 0 이하입니다.");
    }
  }

  // 관심상품 업데이트 validation
  public void validateUpdate(ProductMypriceRequestDto requestDto) {
    // 변경될 관심 가격이 유효한지 확인합니다.
    int myPrice = requestDto.getMyprice();
    if (myPrice < MIN_PRICE) {
        throw new IllegalArgumentException("유효하지 않은 관심 가격입니다. 최소 " + MIN_PRICE + " 원 이상으로 설정해 주세요.");
    }
}
```

- ProductValidator 적용

```
public Product(ProductRequestDto requestDto, Long userId) {
    // 입력값 Validation
    ProductValidator.validateCreate(requestDto, userId);

    // 관심상품을 등록한 회원 Id 저장
    this.userId = userId;
    this.title = requestDto.getTitle();
    this.image = requestDto.getImage();
    this.link = requestDto.getLink();
    this.lprice = requestDto.getLprice();
    this.myprice = 0;
}
```

▼ 30) OOP VS AOP

- OOP 는 **핵심기능**을 모듈화
- AOP 는 **부가기능**을 모듈화
  - 부가기능의 예
    - 로깅, 트랜잭션, API 시간 측정
- AOP 는...
  - OOP 를 "**대체**": X
  - OOP 를 "**보완**": O

▼ 31) 숙제 설명

- 요구사항

  AS-IS)
  - 회원별 총 API 수행시간 ⇒ DB 에 저장 (UserTime 테이블)

  TO-BE)
  - 회원별 총 API 수행시간 ⇒ DB 에 저장 (UserTime 테이블)
  - **회원별 총 API 호출 횟수** ⇒ DB 에 저장 (UserTime 테이블)

11. 5주차 숙제 답안 코드

▼ **[코드스니펫] - 5주차 숙제 답안 코드**

전체 코드

▼ **src > main > java > com.sparta.springcore > model > UserTime.java**

```
package com.sparta.springcore.model;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Setter
@Getter // get 함수를 일괄적으로 만들어줍니다.
@NoArgsConstructor // 기본 생성자를 만들어줍니다.
@Entity // DB 테이블 역할을 합니다.
public class UserTime {
    // ID가 자동으로 생성 및 증가합니다.
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Id
    private Long id;

    @OneToOne
    @JoinColumn(nullable = false)
    private User user;

    @Column(nullable = false)
    private long totalTime;

    @Column(nullable = false, columnDefinition = "bigint default 0")
    private long totalCount;

    public UserTime(User user, long totalTime) {
        this.user = user;
        this.totalTime = totalTime;
        this.totalTime = 1;
    }

    public void updateTotalTime(long totalTime, long totalCount) {
        this.totalTime = totalTime;
        this.totalCount = totalCount;
    }
}
```

▼ **src > main > java > com.sparta.springcore > aop > UserTimeAop.java**

```
package com.sparta.springcore.aop;

import com.sparta.springcore.model.User;
import com.sparta.springcore.model.UserTime;
import com.sparta.springcore.repository.UserTimeRepository;
import com.sparta.springcore.security.UserDetailsImpl;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

@Component // 스프링 IoC 에 빈으로 등록
@Aspect
public class UserTimeAop {
    private final UserTimeRepository userTimeRepository;

    public UserTimeAop(UserTimeRepository userTimeRepository) {
        this.userTimeRepository = userTimeRepository;
    }

    @Around("execution(public * com.sparta.springcore.controller..*(..))")
    public Object execute(ProceedingJoinPoint joinPoint) throws Throwable {
        // 측정 시작 시간
        long startTime = System.currentTimeMillis();
```

```
        try {
            // 핵심기능 수행
            Object output = joinPoint.proceed();
            return output;
        } finally {
            // 측정 종료 시간
            long endTime = System.currentTimeMillis();
            // 수행시간 = 종료 시간 - 시작 시간
            long runTime = endTime - startTime;
            // 로그인 회원이 없는 경우, 수행시간 기록하지 않음
            Authentication auth = SecurityContextHolder.getContext().getAuthentication();
            if (auth != null && auth.getPrincipal().getClass() == UserDetailsImpl.class) {
                // 로그인 회원 -> loginUser 변수
                UserDetailsImpl userDetails = (UserDetailsImpl) auth.getPrincipal();
                User loginUser = userDetails.getUser();

                // 수행시간 및 DB 에 기록
                UserTime userTime = userTimeRepository.findByUser(loginUser);
                if (userTime != null) {
                    // 로그인 회원의 기록이 있으면..

                    // API 전체 수행 시간
                    long totalTime = userTime.getTotalTime();
                    totalTime = totalTime + runTime;

                    // API 전체 수행 횟수
                    long totalCount = userTime.getTotalCount();
                    totalCount++;

                    userTime.updateTotalTime(totalTime, totalCount);
                } else {
                    // 로그인 회원의 기록이 없으면
                    userTime = new UserTime(loginUser, runTime);
                }

                System.out.println("[User Time] User: " + userTime.getUser().getUsername() + ", Total Time: " + userTime.getT
                userTimeRepository.save(userTime);
            }
        }
    }
}
```

12. Outro.

▼ 주니어 개발자 & 시니어 개발자