**AOP**

1. OOP를 보완하는 수단, 흩어진 Aspect를 모듈화 할 수 있는 프로그래밍 기법.
2. Concerns -> 여러 클래스에 나타나는 비슷한 코드 혹은 필드, 메서드
   1. e.g) 트랜잭션, 성능관련 로깅
3. AOP 주요 개념
   1. Aspect : 하나의 모듈 (advice와 pointcut이 구성됨)
   2. Advice : 해야 할 일들.
   3. Pointcut : 어디에 적용해야 하는지. (Join Point의 subset)
   4. Target : 적용이 되는 대상 (Class)
   5. Join Point : 메소드 실행 시점에 Advice를 넣어라. (끼워 넣을 수 있는 시점)
      * 1. e.g) 생성자 호출, 필드 접근 전, 필드에서 값을 가져갔을 때…
4. AOP 적용방법
   1. 컴파일(자바파일을 class파일로 만들 때) – 성능 저하 없음, 별도의 컴파일 과정을 거쳐야함
   2. 로드 타임(A 라는 Class는 순수하게 컴파일하고 A라는 Class파일을 로딩하는 시점(로드타임 위빙))
      1. 클래스 로딩할 때 약간의 부하가 생길 수 있음
      2. 로드타임 위버를 설정해줘야 한다.
      3. 약간의 성능 저하
      4. 다양한 문법을 사용할 수 있다(AspectJ)
   3. 런타임(A라는 class에 A 메소드를 적용해야 하는 것을 Spring은 알고 있고, A라는 Bean에 A type의 proxy 빈을 만들어서 프록시 Bean이 A 메서드를 호출하기 직전에 호출) – 프록시 기반 AOP
      1. Spring AOP를 사용하는 방법
      2. 최초 빈을 만들 때 약간의 성능 저하
      3. 별도의 컴파일이나 로드 타임 위버가 필요 없다.
      4. 문법이 쉽다.
      5. 별도의 AOP 공부를 많이 할 필요 없다.
      6. 가장 현실적이고 자주 쓰게 될 방법일 것임.

프록시 기반 AOP

* + - 1. 프록시 객체가 원래의 객체를 감싸서 실제 클라이언트의 요청을 처리한다.
         1. 패턴의 목적 : 접근 제어 및 부가 기능 추가를 위해
      2. Interface가 있는 경우 @Autowired는 Interface Type으로 데이터를 선언하는 것을 권장
      3. 클래스와 클라이언트 코드를 건드리지 않고 기능을 추가하거나 수정.
      4. Proxy의 타입은 같은 인터페이스를 사용해야한다.
      5. @Primary로 어노테이션을 설정해줘야 한다.
      6. Proxy 클래스를 만들 때도 중복코드가 발생할 수 있으며 코드 작성의 비용과 시간이 들며 다른 서비스에도 적용해야 된다면?
         1. 동적으로 Proxy 객체를 만든다. (런타임) -> 스프링 AOP

스프링 AOP

1. ProceedingJoinPoint : Advice가 적용되는 대상.
   1. 메서드를 실행하는 것 – Proceed
2. *@Around*("pointcut 이름 혹은 pointcut직접 정의")
   1. 이 advice를 어떻게 적용할 것인가?
3. *@Component*
4. *@Aspect*
5. public class PefAspect {
7. *@Around*("execution(\* com.example.demo.EventService.\*(..))")
8. public Object logPerf(ProceedingJoinPoint pjp) throws Throwable {
9. // createdEvent(), publishedEvent
10. long begin = System.*currentTimeMillis*();
12. Object retVal = pjp.proceed();
14. System.***out***.println(System.*currentTimeMillis*() - begin);
16. return retVal;
17. }
19. }

3.

package com.example.demo;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

*@Retention*(*RetentionPolicy*.***CLASS***)

public @interface *PerfLogging* {

}

* RetentionPolicy는 어노테이션 정보를 얼마까지 유지할 것 인가?