Chapter 6-3 Aspect Oriented Programming

```java
@PostMapping()
@ResponseStatus(HttpStatus.CREATED)
public void createPost(@RequestBody PostDto dto){
    logger.info("start processing");
    this.postService.createPost(dto);
    logger.info("finish processing");
}
```

시간 간격을 측정하기 위해 logger 사용

```java
@PostMapping()
@ResponseStatus(HttpStatus.CREATED)
public void createPost(@RequestBody Po

@GetMapping("{id}")
public PostDto readPost(
        @PathVariable("id") int id
){...}

@PutMapping("{id}")
@ResponseStatus(HttpStatus.ACCEPTED)
public void updatePost(
        @PathVariable("id") int id,
        @RequestBody PostDto dto
){...}
```

어느 한 함수 처리에 걸리는 시간을 측정하고 싶다.

- 실제 서비스의 흐름과는 별개
  - 로그를 남기는 기능
  - 서비스를 제공하기 위한 기능
- 서로 직접적인 연관은 없기 때문에 기능과는 별도로 작성하는 것이 이상적

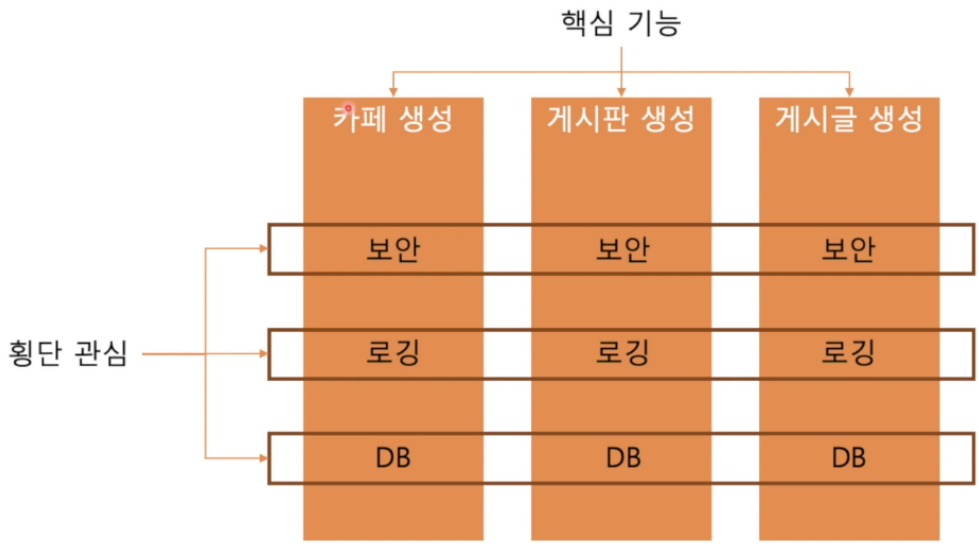
Aspect

서로 다른 역할의 객체들이 가지는 공통의 관심사

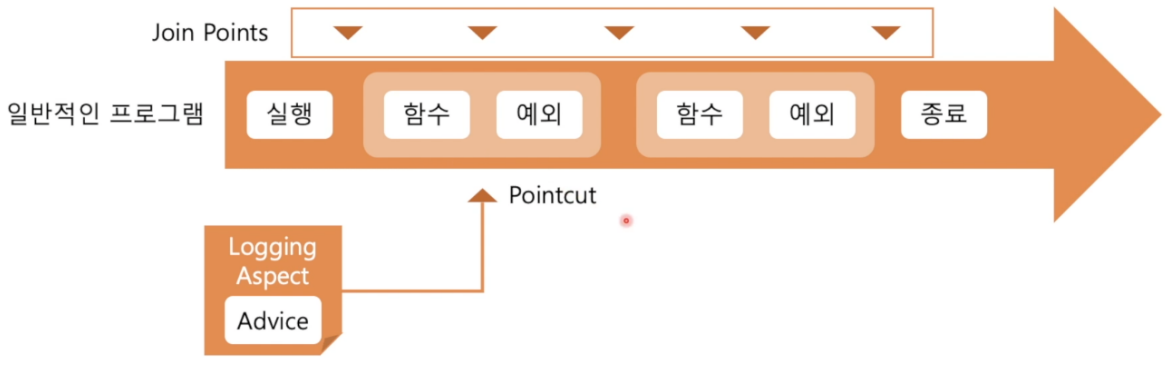
Aspect Oriented Programming

서로 다른 비즈니스 로직이 공통적으로 가지는 관심(횡단 관심)에 대하여 고민하는 개발 지향

AspectJ !!



횡단 관심을 해소하기 위해 등장



특정 관점에 대해서 어떤 행동을 하고 싶다 → Aspect

Aspect가 들어갈 수 있는 지점 : Join Points

Pointcut : 어떤 Join Points에 Aspect를 적용할 것인지 지정할 수 있는 것

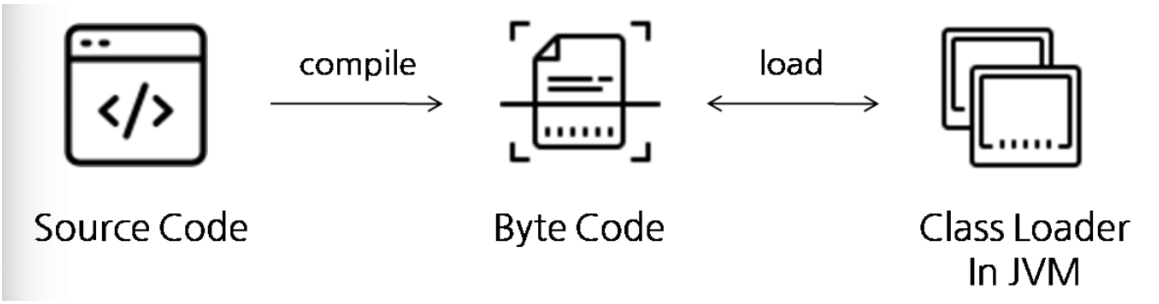
`@Target(ElementType.METHOD)`

`@Retention(RetentionPolicy.RUNTIME)`

`@Retention` 어노테이션의 속성으로 `RetentionPolicy` 라는 것이 있습니다.

여기에 올 수 있는 값은 source, class, runtime 이렇게 3가지가 있습니다. (공부하고나니 이름이 굉장히 직관적입니다.)

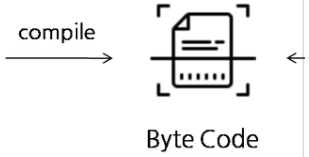
- RetentionPolicy.SOURCE : 소스 코드(.java)까지 남아있는다.
- RetentionPolicy.CLASS : 클래스 파일(.class)까지 남아있는다.(=바이트 코드)
- RetentionPolicy.RUNTIME : 런타임까지 남아있는다.(=사실상 안 사라진다.)



아무 관심 없던 @Retention 어노테이션 정리(RetentionPolicy SOURCE vs CLASS vs RUNTIME)

자바에서 지향하는 방법은 아니지만 필요에 의해서 커스텀 애노테이션(Annotation)을 만들어야 할 때가 있습니다. 보통 예제 샘플 코드를 보면 메타 애노테이션으로 항상 붙어있고 RetentionPolicy=RUNTIME으로 되어습니다. 그렇기 때문에 그럴 때나 보게되는 애노테이션이라 @Retention 은 무시하고 @Target 정도만 확인하고 써왔습니다.

https://jeong-pro.tistory.com/234



`ProceedingJoinPoint`