

AOP

(Aspect-Oriented-Programming)

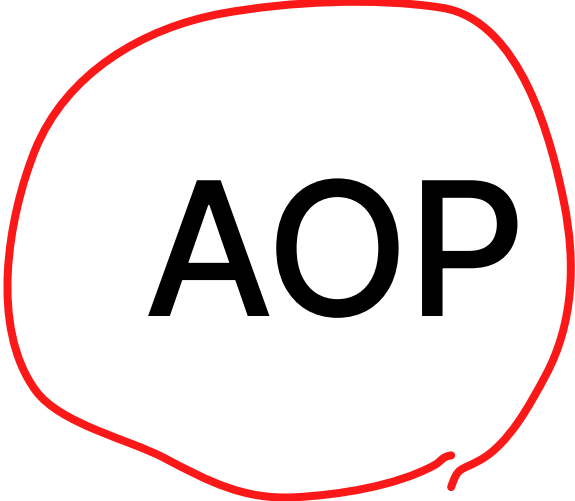
관점 지향 프로그래밍

Spring 3대 요소

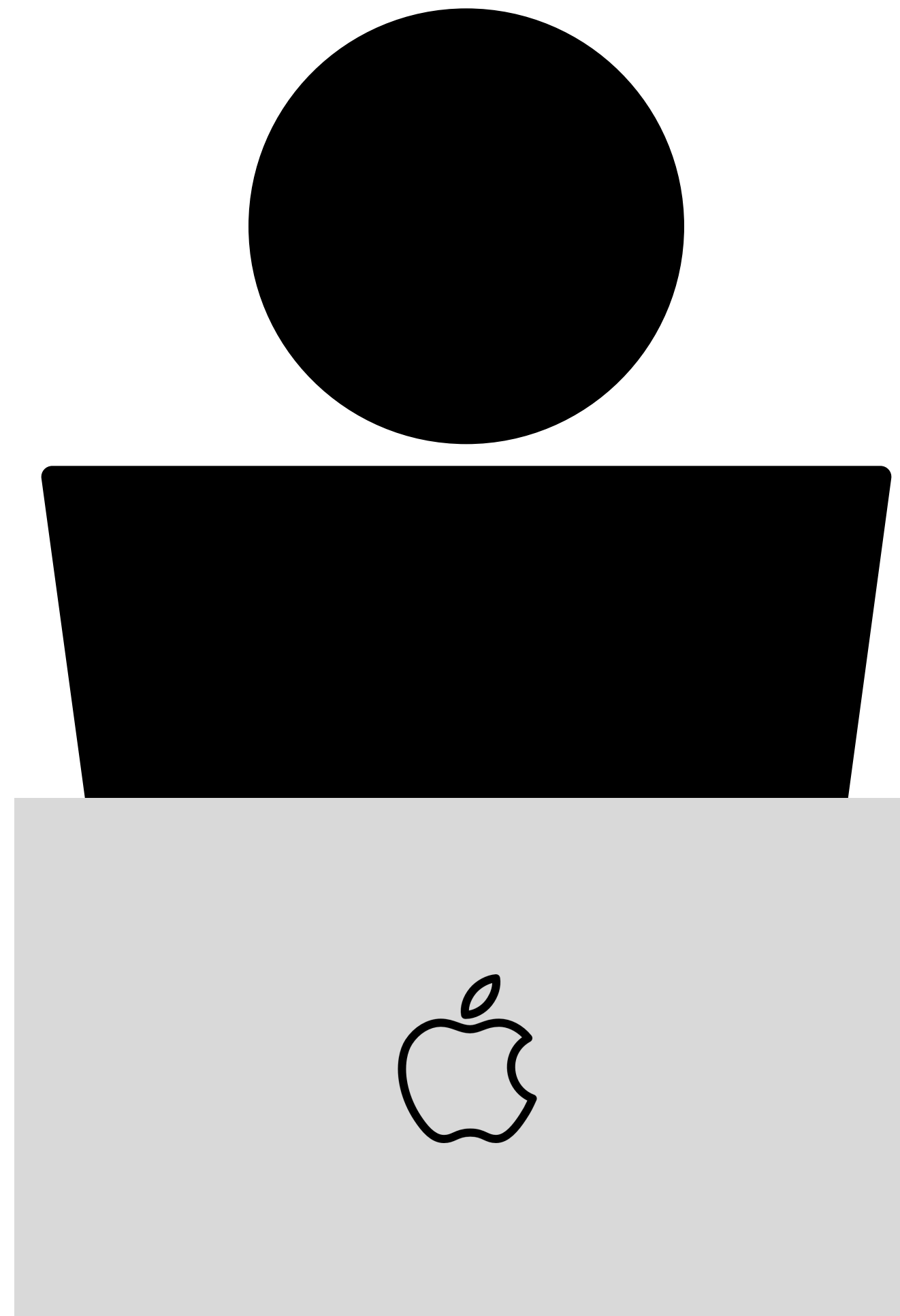
IoC/DI

PSA

AOP



AOP

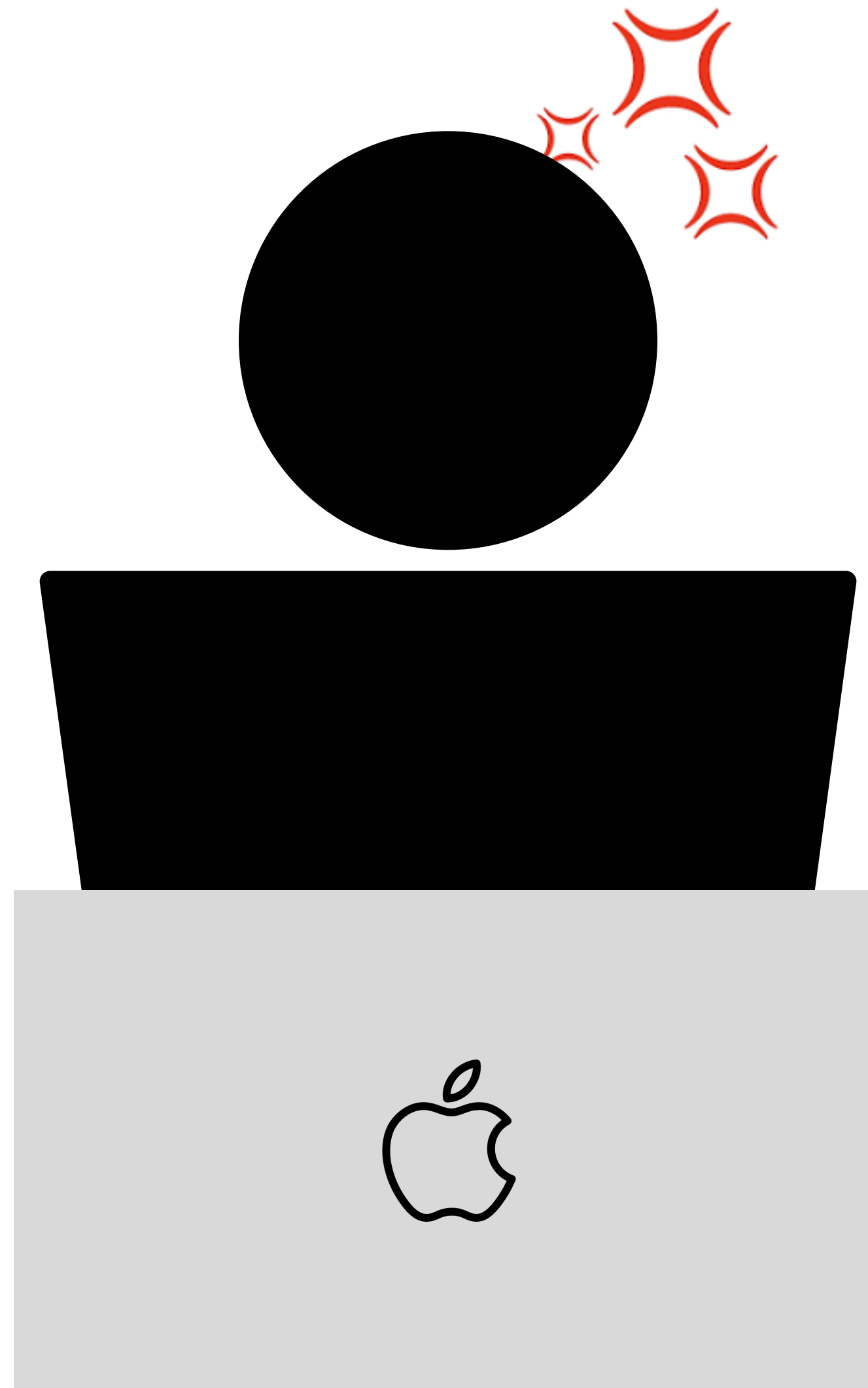


AOP

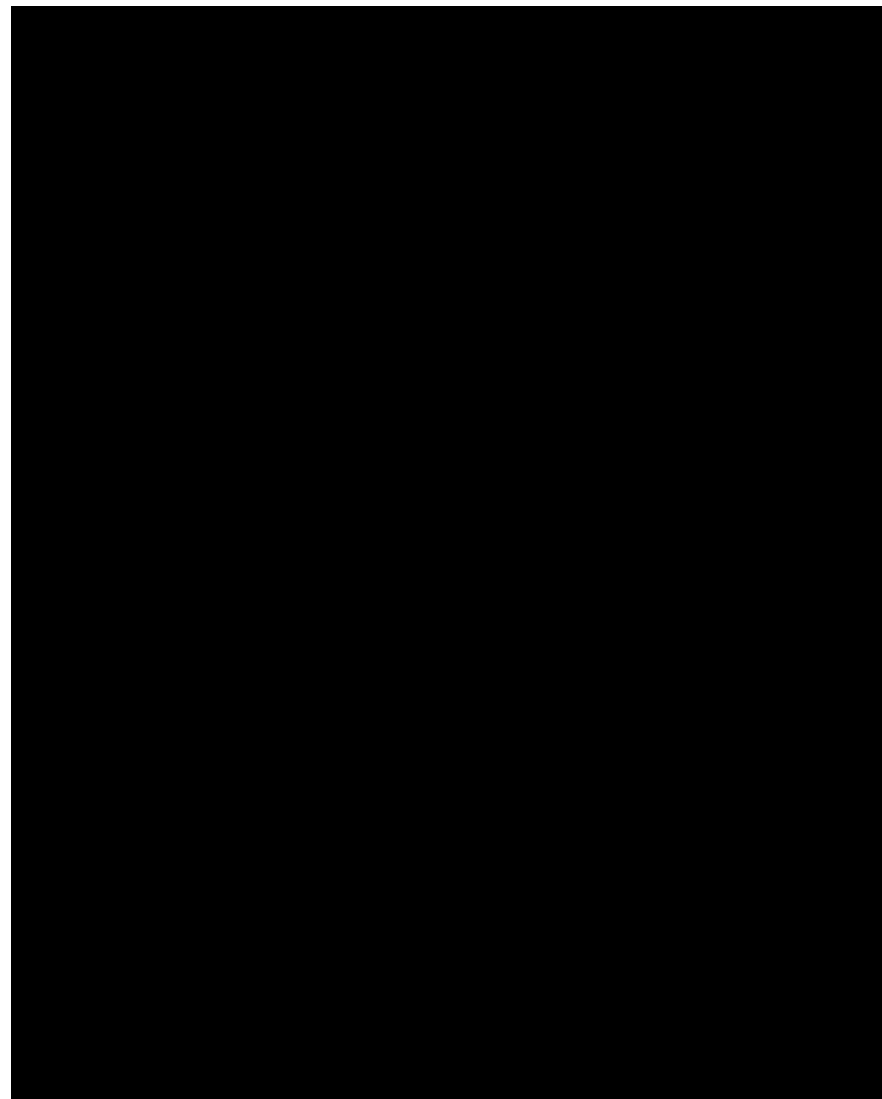
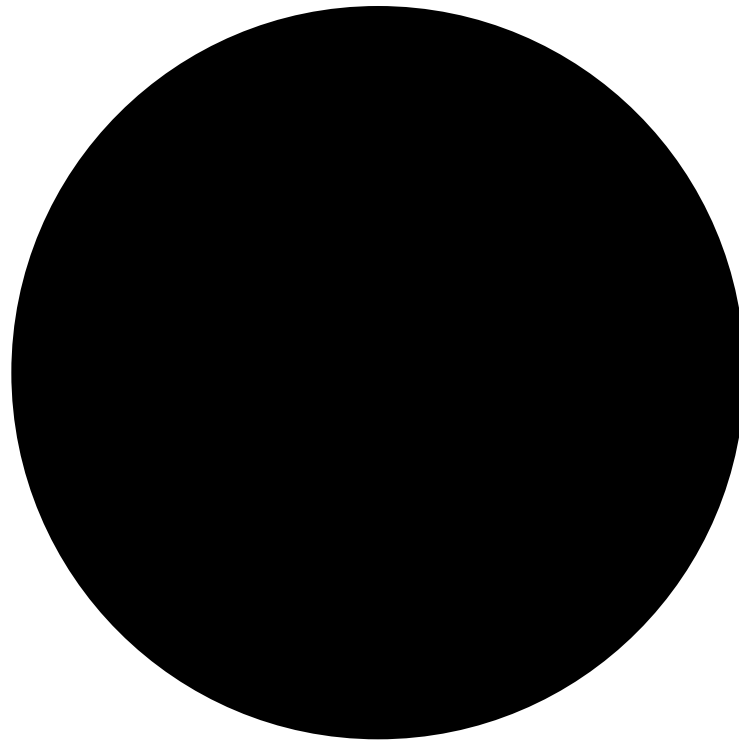
```
Java ▾ 복사 캡션 ..  
  
StopWatch stopWatch = new StopWatch();  
stopWatch.start();  
log.info("토큰을 요청하는 중...");  
// POST 방식으로 key-value 데이터 요청  
OAuthToken oauthTokenRes = wc.post()  
    .uri(TOKEN_URI)  
    .body(BodyInserters.fromFormData(params))  
    .header("Content-type", "application/x-www-form-urlencoded; charset=utf-8")  
    .retrieve()  
    .bodyToMono(OAuthToken.class).block();  
  
stopWatch.stop();  
log.info("토큰 발급 완료! {} ms", stopWatch.getLastTaskTimeMillis());
```

```
-02-27 01:01:25.112 INFO 3619 --- [nio-8080-exec-1] AuthService : "토큰 발급 완료! 437 ms"
```

AOP

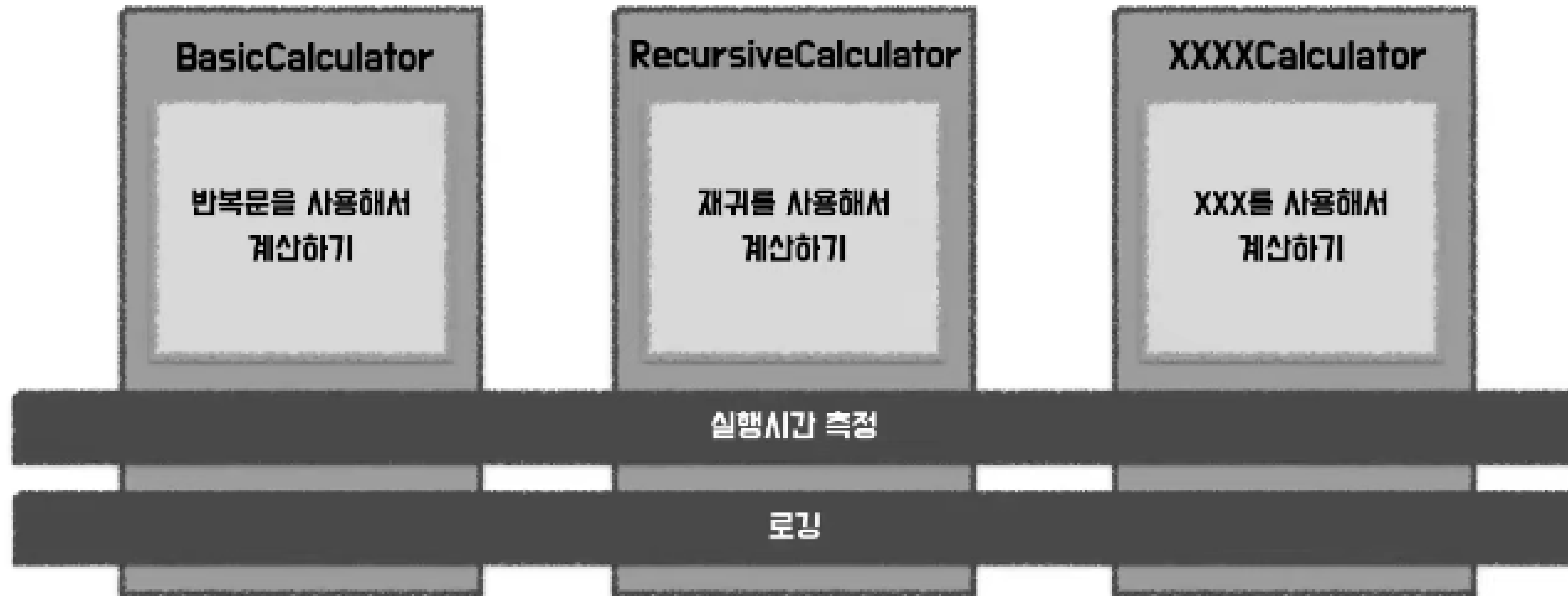


AOP



너무 좋은데, 밀리초 말고 그냥
초 까지만 찍히게 해주세요.

AOP



공통의 관심사를 분리하여 모듈화 하는 것

AOP

AOP == 모듈화 인가요?



AOP

핵심로직

부가로직

AOP

부가로직
(= 인프라 로직)

- 전 영역에서 나타나는 경우가 많다. 모듈화 필요
- 그냥 쓰게 되면 중복 코드가 굉장히 많아진다.
- 비즈니스 로직과 함께 있으면 알기 어려워진다.

AOP

target

- 어떤 대상에 부가 기능을 부여할 것인가?

join point

- 어디에 적용할 것인가?
- ex) 메서드, 필드, 객체, 생성자

advice


- 어떤 부가기능인가? (실행시점)

point cut

- 실제 advice가 적용될 지점

AOP

컴파일 클래스 로드



프록시

컴파일러나 클래스 로더 설정을 하지 않아도 가능

오버라이딩 개념으로 동작, 메서드 실행시점에만 AOP를 적용

스프링 컨테이너가 관리할 수 있는 빈에만 AOP 적용 가능

타겟 class를 proxy로 감싸서 실행하는 방식

AOP

```
@Aspect
@Slf4j
@Component
public class OrderTimeAdvisor {
    @Around("execution(* com.example.cafekiosk.spring.api.service.order.OrderService.*(..))")
    public Object stopWatch(ProceedingJoinPoint joinPoint) throws Throwable {
        Stopwatch stopWatch = new Stopwatch();
        try {
            stopWatch.start();
            return joinPoint.proceed();
        } finally {
            stopWatch.stop();
            log.info("주문 생성 완료! {} ms", stopWatch.getLastTaskTimeMillis());
        }
    }
}
```

AOP

```
@Aspect
@Slf4j
@Component
public class OrderTimeAdvisor {
    @Around("execution(* com.example.cafekiosk.spring.api.service.order.OrderService *(..))")
    public Object stopWatch(ProceedingJoinPoint joinPoint) throws Throwable {
        Stopwatch stopWatch = new Stopwatch();
        try {
            stopWatch.start();
            return joinPoint.proceed();
        } finally {
            stopWatch.stop();
            log.info("주문 생성 완료! {} ms", stopWatch.getLastTaskTimeMillis());
        }
    }
}
```

Point cut

AOP

```
com.example.cafekiosk.spring.api.service.order.OrderServiceImpl
```

AOP 적용 (x)

```
com.example.cafekiosk.spring.api.service.order.OrderServiceImpl$$SpringCGLIB$$0
```

AOP 적용 (o)

```
2024-02-29T15:46:12.534+09:00 INFO 1316 --- [nio-8080-exec-1] c.e.c.spring.api.aop.OrderTimeAdvisor : 주문 생성 완료! 107 ms
```

AOP

@IwantcallbyAOP로 AOP를 실행할 때

```
public void join (JoinRequest joinRequestDto) {  
    help();  
}  
  
@IwantCallbyAOP  
private void help() {  
    System.out.println("취업 시켜줘, 돈 벌게해줘어~~~~~");  
}
```


AOP

```
© OrderServiceImpl.java ×
@Override
public OrderResponse createOrder(OrderCreateRequest request, LocalDateTime registeredDateTime) {
    List<Product> duplicateProducts = findProductsBy(request.getProductNumbers());

    Order order = Order.create(duplicateProducts, registeredDateTime);
    Order savedOrder = orderRepository.save(order);

    // Order
    return OrderResponse.of(savedOrder);
}

/*
 * 아메리카노 1개, 아메리카노 1개와 같은 중복 주문을 위해 총 가격을 List 화.
 * */
1 usage new *
private List<Product> findProductsBy(List<String> productNumbers) {
    System.out.println(this.getClass().getName());
    List<Product> products = productRepository.findAllByProductInfo_ProductNumberIn(productNumbers);
    Map<String, Product> productMap = products.stream()
        .collect(Collectors.toMap(p -> p.getProductInfo().getProductNumber(), p -> p));

    return productNumbers.stream() Stream<String>
        .map(productMap::get) Stream<Product>
        .collect(Collectors.toList());
}
```

```
com.example.cafekiosk.spring.api.service.order.OrderServiceImpl
```

대표적인 AOP

@Transactional

@Intercept

@Filter

Spring AOP

AspectJ

목표

간단한 AOP 기능 제공

완벽한 AOP 기능 제공

join point

메서드 레벨만 지원

생성자, 필드, 메서드 등
다양하게 지원

weaving

런타임 시에만 가능

런타임은 제공하지 않음.
compile-time, post-compile,
load-time 제공

대상

Spring Container가 관리하는
Bean에만 가능

모든 Java Object에 가능