Declaring Advice(어드바이스 선언)

"어드바이스는 특정 메서드에 대해 추가 작업을 수행하는 코드입니다. 포인트컷이라는 규칙을 사용해 어드바이스가 어떤 메서드에 적용될지를 결정합니다. 이렇게 매칭된 메서드가 실행되기 전에, 실행된 후에, 또는 실행 중에 어드바이스가 실행됩니다.

포인트컷 표현식의 방법

1. **인라인 표현식(Inline expression)** : 인라인 표현식은 프로그래밍에서 특정 규칙이나 로직을 코드 내에서 직접 정의하는 방식을 의미합니다. spectJ에서 포인트컷을 별도의 메서드나 변수로 정의하지 않고, 어드바이스(Advice) 안에서 직접적으로 작성하는 방식입니다.
   특징 :
   - 직접성: 포인트컷 인라인 표현식은 어드바이스(Advice) 내에서 직접 작성됩니다. 별도의 메서드나 변수로 분리하지 않고, 해당 어드바이스에서 바로 사용할 포인트컷 조건을 정의합니다. 예를 들어, @Before("execution(* com.example.service..(..))")와 같은 형태입니다.
   - 간결성: 인라인 표현식은 짧고 간결한 코드를 작성할 수 있게 합니다. 어드바이스와 포인트컷을 한 곳에서 정의하므로, 특정 조언이 적용될 메서드가 무엇인지 바로 알 수 있어 코드를 이해하기 쉽습니다.
   - 재사용성 부족: 인라인 표현식은 정의된 곳에서만 사용되기 때문에, 동일한 포인트컷 조건을 여러 어드바이스에서 사용하려면 매번 반복 작성해야 합니다. 이로 인해 코드 중복이 발생할 수 있으며, 재사용성이 떨어집니다.
   - 유지보수 어려움: 여러 곳에서 동일한 인라인 표현식을 사용하고 있을 경우, 조건을 수정해야 할 때 모든 인라인 표현식을 찾아 수정해야 합니다. 이로 인해 코드가 분산되어 있을 경우 유지보수가 어려워질 수 있습니다.
   - 명확성: 인라인 표현식은 어드바이스와 포인트컷이 같은 곳에 정의되기 때문에, 해당 어드바이스가 어떤 포인트컷에 적용되는지를 한눈에 파악할 수 있습니다. 이는 코드의 명확성을 높여주고, 개발자가 의도한 바를 쉽게 이해하게 합니다.
   - 사용 시기: 인라인 표현식은 간단한 포인트컷 조건을 정의하거나, 특정 조건이 반복 사용되지 않는 경우에 적합합니다. 그러나 복잡한 조건이 필요하거나 여러 어드바이스에서 동일한 포인트컷을 사용할 경우, Named Pointcut으로 정의하는 것이 더 바람직합니다.
   예제코드 :

```
@Aspect
public class MyAspect
 {

    @Before("execution(* com.example.service.*.*(..))")
    public void beforeMethod(JoinPoint joinPoint)
    {
        System.out.println("Before method: " +
joinPoint.getSignature().getName());
    }
}
```

2. **named pointcut** : 재사용 가능하고 명확하게 정의된 포인트컷입니다. 포인트컷은 어떤 지점에서 어드바이스(Advice)를 적용할지를 결정하는 데 사용되며, Named Pointcut은 이러한 포인트컷을 별도의 이름을

부여하여 정의한 것입니다.
특징 :

- 재사용성: Named Pointcut은 한 번 정의해두면 여러 곳에서 재사용할 수 있습니다. 이는 코드의 중복을 줄이고, 유지보수성을 높이는 데 기여합니다.
- 가독성: 포인트컷에 이름을 부여함으로써 코드의 가독성이 향상됩니다. 다른 개발자가 코드의 의도를 쉽게 이해할 수 있습니다.
- 관리 용이성: 포인트컷 로직이 변경될 때, 해당 Named Pointcut을 참조하는 모든 어드바이스에서 자동으로 변경 사항이 반영되므로 관리가 용이합니다.
  예제코드 :

```
@Aspect
public class MyAspect
{

    // Named Pointcut 정의
    @Pointcut("execution(* com.example.service.*.*(..))")
    public void serviceLayer() {}

    // 정의된 Named Pointcut을 사용하는 어드바이스
    @Before("serviceLayer()")
    public void beforeServiceMethod(JoinPoint joinPoint) {
        System.out.println("Before method: " +
joinPoint.getSignature().getName());
    }
}
```

After Returning Advice

After returning 어드바이스는 매칭된 메서드 실행이 정상적으로 리턴될 때 실행됩니다. 그이유는 이 어드바이스가 정상적인 실행 흐름에 따라 후속 작업을 수행하기 위한 것이기 때문입니다. 예외 상황에서는 이 어드바이스가 실행되지 않고, 대신 예외 처리에 특화된 어드바이스가 실행됩니다. 이렇게 함으로써, 정상 흐름과 예외 흐름을 구분하여 더 세밀한 제어가 가능해집니다.

```
@Aspect
// 포인트컷 정의: com.example.service 패키지 내의 모든 메서드를 타겟팅
public class MyAfterReturning
{
    @AfterReturning
    (
        pointcut = "execution(* com.example.service.*.*(..))",
        returning = "retVal"
    )
    public void afterReturningMethod(Joinpoint jp, Object retVal)
    {
        // 실행된 메서드 이름 출력
        System.out.println("Method :" + jp.getSignature().getName());

        // 반환된 결과값 출력
```

```
        System.out.println("Returned value: " + retVal);
    }
}
```

> 동일한 어드바이스 내에 여러 개의 어드바이스 선언(및 다른 멤버들)을 가질 수 있습니다.

해당 코드에서 returning 은 어드바이스가 실행될 때 메서드의 반환 값을 받을 변수명을 지정합니다. 이 예제에서는 retVal 이름을 사용했습니다.

returning 속성에서 지정한 이름(예: returning="retVal")은 어드바이스 메서드의 매개변수 이름과 동일해야 합니다. 예를 들어, 위의 어드바이스 메서드 `afterReturningMethod` 첫 번째 매개변수 이름이 retVal이라고 가정할 때, returning 속성에서 사용된 "retVal"이라는 이름은 이 매개변수와 일치해야 합니다.

메서드가 정상적으로 실행을 마치고 값을 반환할 때, 그 반환 값은 어드바이스 메서드의 해당 매개변수로 전달됩니다. 또한, returning 절은 특정 타입(예를 들어, Object)의 값을 반환하는 메서드만 매칭하도록 제한합니다. 즉, 지정된 타입의 값을 반환하는 메서드에서만 이 어드바이스가 적용됩니다.

After Throwing Advice

Throwing 어드바이스는 매치된 메서드가 실행이 예외를 던지며 종료될 때 실행됩니다. @AfterThrowing 어노테이션을 사용하여 선언할 수 있습니다 .

종종 특정 타입의 예외가 발생할 때만 어드바이스가 실행되도록 하고 싶을 때가 있습니다. 이때 throwing 속성을 사용하면, 특정 예외 타입에 매칭을 제한할 수 있으며, 발생한 예외를 어드바이스 메서드의 파라미터로 바인딩할 수 있습니다. 만약 특정 예외 타입에만 한정하지 않고, 모든 예외를 처리 하고싶다면 Throwable을 사용합니다.

```
@Aspect
public class AFterThrowing
{
    @AfterThrowing
    (
        pointcut = "execution(* com.example.service.*.*(..))"
        throwing = "ex"
    )
    public void    afterThrwingMethod(DataAccessException ex)
    {
        //Do somthing.....
    }


    //만약 특정 예외 타입에만 한정하지 않고, 모든 예외를 처리 하고싶다면 Throwable을 사용합니다.
    @AfterThrowing
    (
        pointcut = "execution(* com.example.service.*.*(..))"
        throwing = "ex"
    )
        public void afterThrwingMethodWithTrowable(Throwable ex)
    {
```

```
        //Do somthing.....
    }
}
```

After (Finally) Advice

After (finally) 어드바이스는 매치된 메서드의 실행이 종료될 때 항상 실행됩니다. 이 어드바이스는 @After 어노테이션을 사용하여 선언됩니다. After 어드바이스는 메서드가 정상적으로 리턴하든, 예외를 던지든 상관없이 실행됩니다. 주로 리소스를 해제하거나, 마지막으로 꼭 실행되어야 하는 정리 작업을 수행하는 데 사용됩니다.

```
@Aspect
public class AfterFinallyExample
{


    @After("execution(* com.xyz.dao.*.*(..))")
    public void doReleaseLock() {
        // ...
    }
}
```

> AspectJ에서 @After 어드바이스는 try-catch 문의 finally 블록과 유사한 "after finally 어드바이스"로 정의됩니다. 이는 조인 포인트(사용자가 선언한 타겟 메서드)에서 정상적으로 리턴되든 예외가 발생하든 모든 결과에 대해 호출됩니다. 이는 성공적인 정상 리턴에만 적용되는 @AfterReturning과는 대조적입니다.

Around Advice

Around 어드바이스는 매치된 메서드의 실행 전후에 실행됩니다. 이 어드바이스를 사용하면 메서드가 실행되기 전과 후에 작업을 수행할 수 있습니다. 또한, 메서드가 실제로 실행될지, 언제 실행될지, 그리고 어떻게 실행될지를 제어할 수 있는 기회를 제공합니다.

Around 어드바이스는 주로 메서드 실행 전후에 상태를 스레드 안전(thread-safe) 방식으로 공유해야 할 때 사용됩니다. 예를 들어, 메서드 실행 전 타이머를 시작하고, 메서드 실행 후 타이머를 정지하는 경우에 유용합니다.

> 항상 요구 사항을 충족하는 가장 단순한 형태의 어드바이스를 사용해야 합니다. 예를 들어, before 어드바이스로 충분하다면 around 어드바이스를 사용하지 마십시오.

- **@Around 어노테이션**: 어라운드 어드바이스로 사용될 메서드에 `@Around` 어노테이션을 붙입니다. 이 어노테이션은 해당 메서드가 어드바이스 메서드로 동작하도록 지정합니다.
- **리턴 타입**: 어라운드 어드바이스 메서드는 항상 `Object` 타입을 리턴해야 합니다. 이는 타겟 메서드의 리턴 값을 처리하기 위해 필요합니다.
- **첫 번째 파라미터**: 어드바이스 메서드는 반드시 `ProceedingJoinPoint` 타입의 첫 번째 파라미터를 가져야 합니다. 이 파라미터는 실제로 호출될 타겟 메서드의 실행 및 정보를 제공합니다.
- **proceed() 메서드 호출**: 어라운드 어드바이스 메서드 내부에서 `proceed()` 메서드를 호출해야 타겟 메서드가 실행됩니다. 이 메서드를 호출하지 않으면 타겟 메서드가 실행되지 않습니다.

- **아규먼트 전달**: `proceed()`를 호출할 때 별도의 인자를 전달하지 않으면, 원래 메서드에 전달된 아규먼트들이 그대로 사용됩니다.

  -하지만, 만약 전달하는 아규먼트를 변경하고 싶다면, Object[] 배열을 아규먼트로 전달하는 proceed(Object[] args) 메서드를 사용할 수 있습니다. 이 배열 안에 있는 값들이 타겟 메서드의 새로운 아규먼트로 사용됩니다.

```
@Aspect
@Component
public class ArgumentModificationAspect
{

    @Around("execution(* com.example.service.*.*(..))")
    public Object modifyArguments(ProceedingJoinPoint joinPoint) throws
Throwable
    {
        // 원래 메서드에 전달된 아규먼트 가져오기
        Object[] args = joinPoint.getArgs();

        // 첫 번째 아규먼트가 String 타입이라고 가정하고, 값을 변경해봅니다.
        if (args != null && args.length > 0 && args[0] instanceof String) {
            args[0] = "Modified Argument";
        }

        // 변경된 아규먼트로 메서드 실행
        Object result = joinPoint.proceed(args);

        // 결과 반환
        return result;
    }
}
```

ProceedingJoinPoint 란?

"Spring AOP와 AspectJ의 proceed() 메서드 동작 차이와 호환성 고려사항"

Advice Parameters

어떤 advice 메서드든 첫 번째 파라미터로 org.aspectj.lang.JoinPoint 타입의 파라미터를 선언할 수 있습니다. 특히 around advice는 ProceedingJoinPoint 타입의 첫 번째 파라미터를 선언해야 하는데, 이 타입은 JoinPoint의 하위 클래스입니다.

JoinPoint 인터페이스는 여러 유용한 메서드를 제공합니다:

- getArgs(): 메서드 아규먼트를 리턴합니다.
- getThis(): 프록시 객체를 리턴합니다.
- getTarget(): 타겟 객체를 리턴합니다.
- getSignature(): 현재 advice가 적용되고 있는 메서드의 설명을 반환합니다.
- toString(): advice가 적용되는 메서드에 대한 유용한 설명을 출력합니다.

Passing Parameters to Advice

이전 섹션에서는 리턴 값이나 예외 값을 바인딩하는 방법(`after returning` 및 `after throwing` 어드바이스)을 살펴보았습니다. 이와 비슷하게, 메서드 아규먼트를 어드바이스 본문에 전달하기 위해 `args` 바인딩 표현식을 사용할 수 있습니다. `args` 표현식에서 타입 이름 대신 파라미터 이름을 사용하면, 해당 파라미터에 전달된 값이 어드바이스가 호출될 때 함께 전달됩니다.

예를 들어, DAO 작업을 수행하는 메서드 중 첫 번째 파라미터로 `Account` 객체를 받는 메서드의 실행을 어드바이스로 처리하고 싶다면, 다음과 같이 작성할 수 있습니다:

```
@Before("execution(* com.xyz.dao.*.*(..)) && args(account,..)")
public void validateAccount(Account account) {
    // ...
}
```

여기서 `args(account,..)` 부분은 두 가지 역할을 합니다.

1. 아규먼트가 최소 하나 이상 존재하고, 그 아규먼트가 `Account` 타입의 인스턴스인 경우에만 메서드 실행을 매칭합니다.
2. 실제 `Account` 객체를 `account` 파라미터를 통해 어드바이스에서 사용할 수 있도록 합니다.

또 다른 방법으로는, `Account` 객체 값을 제공하는 포인트컷을 선언하고, 그 포인트컷을 어드바이스에서 참조하는 방법이 있습니다. 예를 들어 다음과 같이 작성할 수 있습니다:

```
@Pointcut("execution(* com.xyz.dao.*.*(..)) && args(account,..)")
private void accountDataAccessOperation(Account account) {}

@Before("accountDataAccessOperation(account)")
public void validateAccount(Account account) {
    // ...
}
```

프록시 객체(`this`), 타겟 객체(`target`), 그리고 애너테이션(`@within`, `@target`, `@annotation`, `@args`)도 유사한 방식으로 바인딩할 수 있습니다. 다음은 `@Auditable` 애너테이션이 적용된 메서드 실행을 매칭하고, 감사 코드를 추출하는 예제입니다:

먼저 `@Auditable` 애너테이션을 정의합니다:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Auditable {
    AuditCode value();
}
```

다음은 `@Auditable` 메서드의 실행을 매칭하는 어드바이스입니다:

```
@Before("com.xyz.Pointcuts.publicMethod() && @annotation(auditable)")
public void audit(Auditable auditable) {
    AuditCode code = auditable.value();
    // ...
}
```