**AOP(Aspect Oriented Programming) Mantığı**

Bu konunun mantığı şudur.Herhangi bir metodu çalıştırmadan önce çalıştırdıktan sonra yada çalışma esnasında yapılacakları belirtiyoruz.Örnek verecek olursak diyelim ki hibernate jpa ile veri tabanına kaydetme silme yada güncelleme işlemi yapacağız.Hibernate sorgularında başlangıçta transaction başlat-sonrasında transaction kapat gibi kodlar vardı.İşte biz örnekle saveQuery diye bir metod yazıyoruz.Yada delete yada merge farketmez.AOP ile bu metoddan önce transaction başlatma metodu yine AOP ile bu metoddan sonra transaction kapatma gibi metodlar yazıyoruz.Ve artık saveQuery yada deleteQuery metodunun içinde sadece kaydetme silme güncelleme kodu yazıyoruz.

Bunu projeye dahil etmek için mvnrepository’den aspectjrt ve aspectjweaver ın dependency sini alıp projeye dahil etmemiz gerekiyor.Daha sonra com.project.aop adında bir paket açıp AopLog adında class açıyoruz ve aşağıdaki gibi kodları yazıyoruz.

* **@Before-@After**

Bu alttaki 2 anotasyon kesin olmalı.!

@Aspect //Aspect olduğunu belli ettiğimiz anotasyon

@Component // Component yazıp bunu otomatik bean olarak tanımlıyoruz.

public class AopLog {

//@Before metoddan önce çalıştırılacağını belirtiyor.

//Bu alttaki mavi alanda execution kısmı standart,parantez içine ise hangi metodda uygulayacağımızı belirtiyoruz.

@Before("execution (public \* findProductNames(..))")

public void beforeAdvice() {

System.out.println("İşlem Başlatılıyor");

}

//@After metoddan sonra çalıştırılacağını belirtiyor.

//Bu alttaki mavi alanda execution kısmı standart,parantez içine ise hangi metodda uygulayacağımızı belirtiyoruz.

@After("execution (public \* findProductNames(..))")

public void afterAdvice() {

System.out.println("İşlem Kapatılıyor.");

}

}

**AppConfig Classı ile**

@Configuration

@ComponentScan("com.project")

@EnableAspectJAutoProxy

public class AppConfig {

}

* **@PointCut**

Bunun amacı kod fazlalığını engellemek için before yada after’ın içine yazığımız yukarıdaki kodu yani yolu tek bir yerde tutuyor.Sonra heryerde kullanıyor.

Bu birnevi before afterı kullancağımız metodu belirtip ona bir takma ad veriyoruz.Sonra before aftera gelip onlarda bu takma adı kullanıyoruz gibi düşün.

@Aspect

@Component

public class AopLog {

@Before("findProductNames()")

public void beforeAdvice() {

System.out.println("İşlem Başlatılıyor");

}

@After("findProductNames()")

public void afterAdvice() {

System.out.println("İşlem Kapatılıyor.");

}

//Hangi classta kullanacağımızı burada belirttik.

//Bu metodun ismi önemli çünkü yukarıda kullanıcaz.

\*\*\*\*\*\* @Pointcut("execution (public \* findProductNames())")

public void findProductNames() {

}

//-----------------------------------------------

//-----------------------------------------------

@Before("pointCutProductRepository()")

public void beforeAdvice1() {

System.out.println("İşlem Başlatılıyor");

}

@After("pointCutProductRepository()")

public void afterAdvice1() {

System.out.println("İşlem Kapatılıyor.");

}

//Bütün classlarda kullan demek.

\*\*\*\*\*\* @Pointcut("within (com.project.repositoryImp.UserRepository)")

public void pointCutProductRepository() {

}

}

* **@JoinPoint**

Bu konunun mantığı.Yukarıdaki kodlar console’da çalışınca şöyle bir çıktı oluşturuyordu.

İşlem Başlatılıyor.

(\*\*\*METODUN İÇİNDEKİ İŞLEMLER\*\*\*)

İşlem Kapatılıyor.

Fakat bu işlemlerin hangi classta olduğu vs bilgisi gelmiyordu.Sonra şu kodları ekledik.

@Before("pointCutProductRepository()")

public void beforeAdvice(JoinPoint joinPoint) {

System.out.println("İşlem Başlatılıyor " + joinPoint);

}

@After("pointCutProductRepository()")

public void afterAdvice(JoinPoint joinPoint) {

System.out.println("İşlem Kapatılıyor. " + joinPoint);

}

@Pointcut("within (com.project.repositoryImp.\*)")

public void pointCutProductRepository() {

}

Artık çıktı şu şekilde,bize o metodun konumunu da veriyor.

İşlem Başlatılıyor execution(List com.project.repository.IProductRepository. findProducts())

(\*\*\*METODUN İÇİNDEKİ İŞLEMLER\*\*\*)

İşlem Kapatılıyor. execution(List com.project.repository.IProductRepository. findProducts())

* **@AfterReturning**

Bu konunun mantığı şudur,Genellikle return değeri olan metodlar için kullanılır.Ve return değerinden hemen sonra ama yine metod içinde kullanılır.Mesela return değerini döndürdükten sonra istediğimiz logu tutubilir,ayrıca metodun return değerini de yazırabiliriz.Normal @After’da metoddan çıkıp yazdırdığı için return değerini yazdıramazdık.

Bu özelliği kullanabilmek için pointCut kullanmamız şart.Aşağıdaki gibi anotasyonun parantezine kodu yazıyoruz,Aşağıdaki metodda göründüğü gibi String türünde metin parametresi var,o string tipi loglamak istediğimiz metodun return değerinin tipiyle aynı olmak zorunda.Şayet integer tipinde bir metodda bu özelliği kullanmış olsaydık int xx yazacaktık ve bu xx aynı zamanda anotasyonun returning özelliğinde de yazmalıydı.

@AfterReturning(pointcut = "execution (public \* findProductNames())",returning = "metin")

**public** **void** **afterReturn**(JoinPoint joinPoint,String metin) {

System.***out***.println(metin);

System.***out***.println("Return edildikten hemen sonra" + joinPoint);

}

* **@AfterThrowing**

Bunun mantığı şudur.Hata patlattıktan sonra çağıralacak metoddur.Mesela diyelim ki findProductById diye ıd ile product bulan sql metodumuz var.Bu metodun içini şöyle doldurduk.

**public** Product **findProductById**(**int** productId) {

**if**(productId>0) {

TypedQuery<Product> **typedQuery** = **this**.entityManager.createQuery (ProductQueries.***findProductById***,Product.**class**);

typedQuery.setParameter("productId", productId);

Product **product** = typedQuery.getSingleResult();

**return** product;

}

**else** {

**throw** **new** RuntimeException(); //Bak burada eğer productId yi 0 dan küçük girerse runtimeexception fırlat dedik.

}

}

Ve bizim afterthrowing metodunun içine geldik ve aşağıdaki kodları yazarak dedik ki runtime exception diye bir hata patlarsa findProductById metodunda şu metodu çağır.

@AfterThrowing(pointcut = "execution (public \* com.project.repositoryImp. ProductRepository.findProductById(\*))",throwing = "exception")

public void afterThrowingAdvice(JoinPoint joinPoint,RuntimeException exception) {

System.out.println("Geçersiz index değeri HATA\*HATA\*HATA\*HATA\*HATA");

System.out.println("METOD= " + joinPoint);

System.out.println("HATA= " + exception);

}

//After throwing anotasyonunun içindeki throwing özelliğindeki isimle(exception) bu adviceyi kullanacağımız metodun parametresindeki hata ismiyle aynı olmalı

//poincut özelliği kesinlikle olmalı yoksa throwing özellğini kullanamazsın.

//Joinpointte kullanarak hatanın olduğu metodu belirtirsek daha güzel anlaşılır olur.

* **@Around**

Belirlenen bir metodun hem başlamasından önce hem bitmesinden sonra olacak işlemleri yapmamıza yardımcı oluyor.

@Around("findProductByIdAdvice()")

**public** Object **aroundAdvice**(ProceedingJoinPoint proceedingJoinPoint) **throws** Throwable {

System.***out***.println("Around başladı" + proceedingJoinPoint);

Object object=proceedingJoinPoint.proceed();

System.***out***.println("Around bitti" + proceedingJoinPoint);

**return** object; //Object türünden sonuç döndürmesi şart.

}

@Pointcut("within (com.project.repositoryImp.\*)")

**public** **void** **findProductByIdAdvice**() {

}

//ProceedingJoinPoint yazmamız şart

//İçerideki kodları throws Throwable ile yada try catch ile yapmamız şart

**Spring İmport**

Bu konunun mantığı şudur.Büyük projelerde tek bir applicationContext.xml dosyası kullanılmaz.Yine resources altında bir folder içinde yada dışında beans.xml, database.xml, abc.xml gibi farklı farklı xlm configuration dosyaları olur.Ve bunları mainde çağırırken isimleriyle falan uğraşmamak için bir tane applicationContext.xml olur herzamanki gibi ve oradan alırız.

**applicationContext ile yapma**

<import resource="beans/beans.xml"/> //bean folderının altındaki bean.xml i kullan

<import resource="beans.xml"/> //bean.xml i kullan

<import resource="database/database.xml"/> //bean folderının altındaki bean.xml i kullan

* Bu yukarıdaki kodları yazınca artık hangi beanı kullanmak istersek kullanalım main classpathxlm’de applicationContext.xml i yazacağız.

**AppConfig ile yapma**

@Configuration

**public** **class** BeanConfig {

BEANLAR BEANLAR BEANLAR

}

@Configuration

**public** **class** DatabaseConfig {

BEANLAR BEANLAR BEANLAR

}

@Configuration

@Import({

DatabaseConfig.**class**,BeanConfig.**class**

})

**public** **class** AppConfig {

}

* Bu yukarıdaki kodları yazınca artık hangi beanı kullanmak istersek kullanalım main annotation’de AppConfig classını kullanacağız.