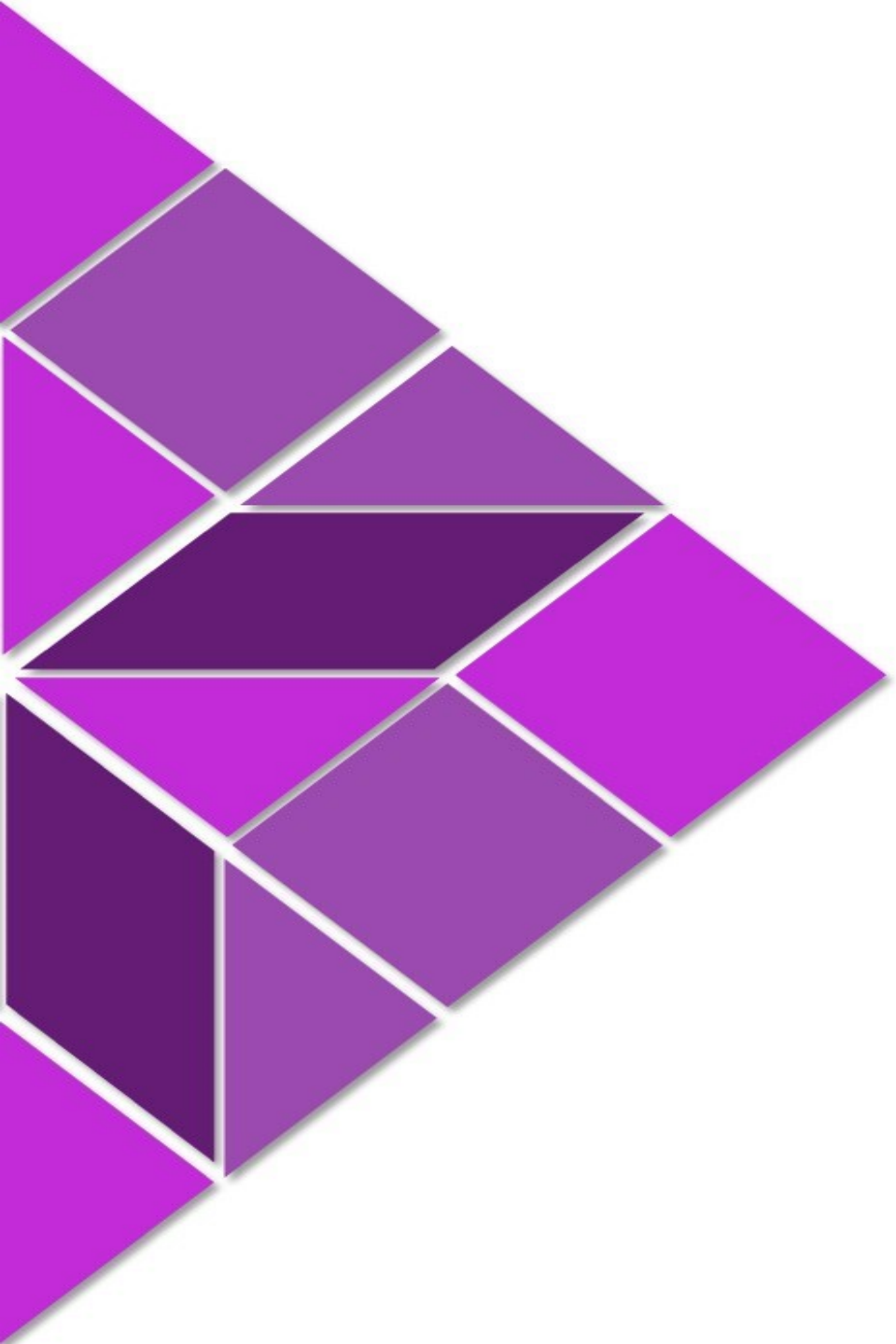


Spring AOP

Presented by Hosein Zare
Twitter: @zare88r

Aspect Oriented Programming

- در AOP از مفهوم aspect یه جای کلاس استفاده می شود
- این مفهوم باعث تفکیک شدن منطق و کدهای برنامه به بخش های مجزایی می شود
  - به طور مثال Logging – Security , ...
- توابعی که بخش های مختلف یک برنامه را به هم وصل می کنند را Cross-Cutting Concernsمی گویند

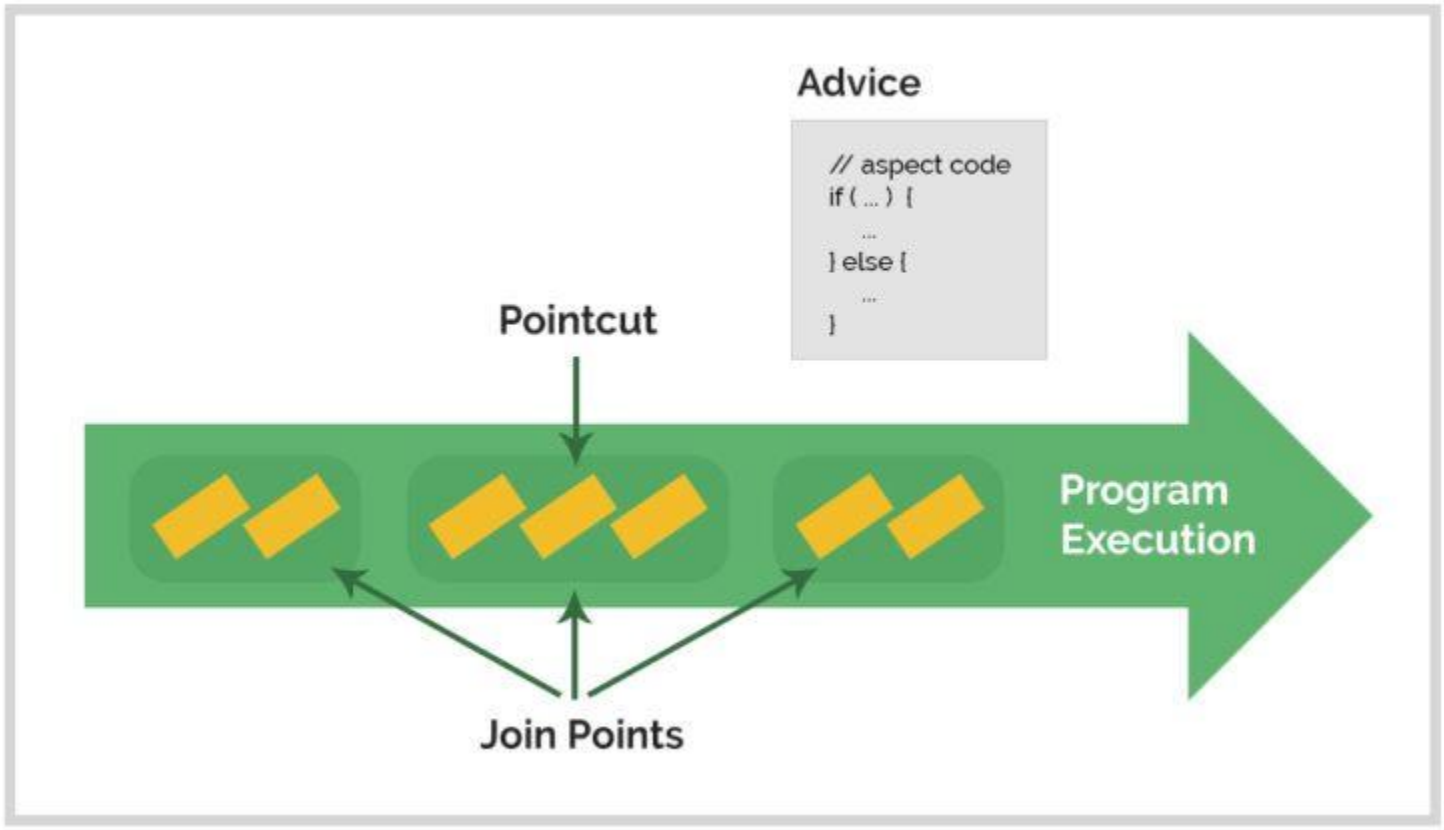
Aspect Oriented Programming

- Spring AOP، رهگیری های کدی در اختیار شما قرار می دهد که می توانید فرآیند اجرای application را قطع کنید برای مثال ، وقتی که یک متد اجرا می شود شما می توانید کاربردهایی را برای مراحل قبل و بعد از اجرای متد ، به آن اضافه کنید .

مفاهیم

- Aspect ماژولی است که دارای تعدادی APIبوده و زمینه های لازم جهت Cross-Cuttingرا فراهم می کند
- Join Point نقطه ای در برنامه شماست که از طریق آن می توانید AOP Aspectرا به برنامه متصل کنید
- Pointcut مجموعه ای از یک یا چندین Join Pointاست که adviseبایستی در آن اجرا شود
- Target Object شی است که توسط یک یا چند aspectمورد توجه قرارگرفته و همیشه یک proxy خواهد بود . همچنین به آن advised objectهم می گویند

Aspect Oriented Programming



مدل های مختلف Advice

- before این advice قبل از اجرای متد مورد نظر اجرا می شود .
- after این advice بعد از اجرای متد مورد نظر و بدون توجه به خروجی آن اجرا می شود .
- after-returning این adviceپس از اجرای متد ، به شرطی که کاملا به درستی اجرا شده باشد رخ می دهد .
- after-throwing این advice نیز پس از اجرای متد ، به شرطی که در آن متود exception پرتاب شده باشد انجام می شود .
- around این adviceنیز قبل و بعد از فراخوانی متد مورد نظر ، اجرا می شود .

AOP with AspectJ

- زمانی که نیاز به advice object هایی داشته باشیم که توسط spring container مدیریت نمیشوند AspectJ بهترین گزینه است
- همچنین اگر زمانی نیاز به اضافه کردن point cut ها روی field ها ، constructor ها و ... داشتیم (نه فقط روی متودها) از این قابلیت میتوانیم استفاده کنیم

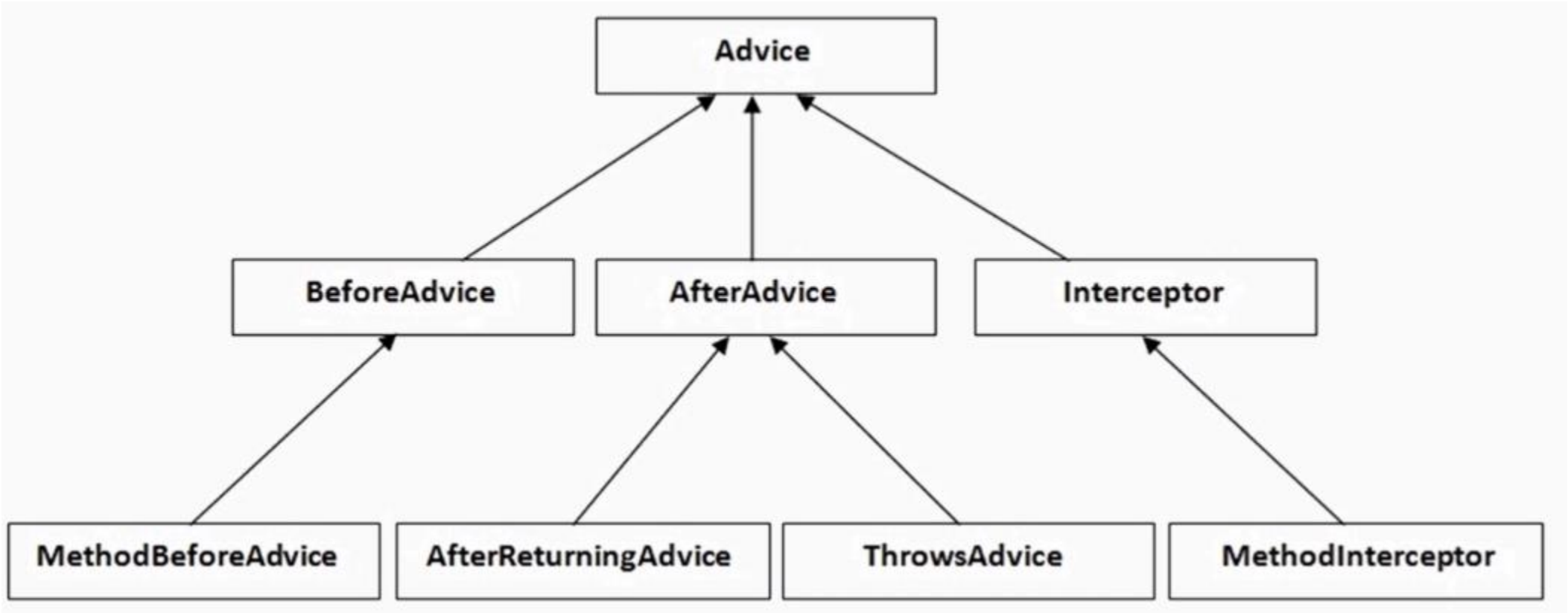
AOP with AspectJ

```
@Aspect
@Component
public class LogAspect {
    @Pointcut("execution(public * com.demisco.fod.*.*(..))")
    public void allMethods() {
    }
    @Pointcut("@annotation(com.demisco.fod.Log)")
    public void withLogAnnotation() {

    }
    @Around("allMethods() && withLogAnnotation()")
    public Object logAround(ProceedingJoinPoint joinPoint) throws Throwable {

        Object returnObj = joinPoint.proceed(joinPoint.getArgs());

        return returnObj;
     }
}
```

Spring AOP

- Spring AOP با جاوای خام نوشته میشود و نیازی به یادگیری فریم ورک جدید ندارد
- نیازی به فرآیند جداگانه کامپایل نیست
- در زمان Runtime هندل میشود لذا در فضاهایی مثل servlet container به مشکل نمیخورد

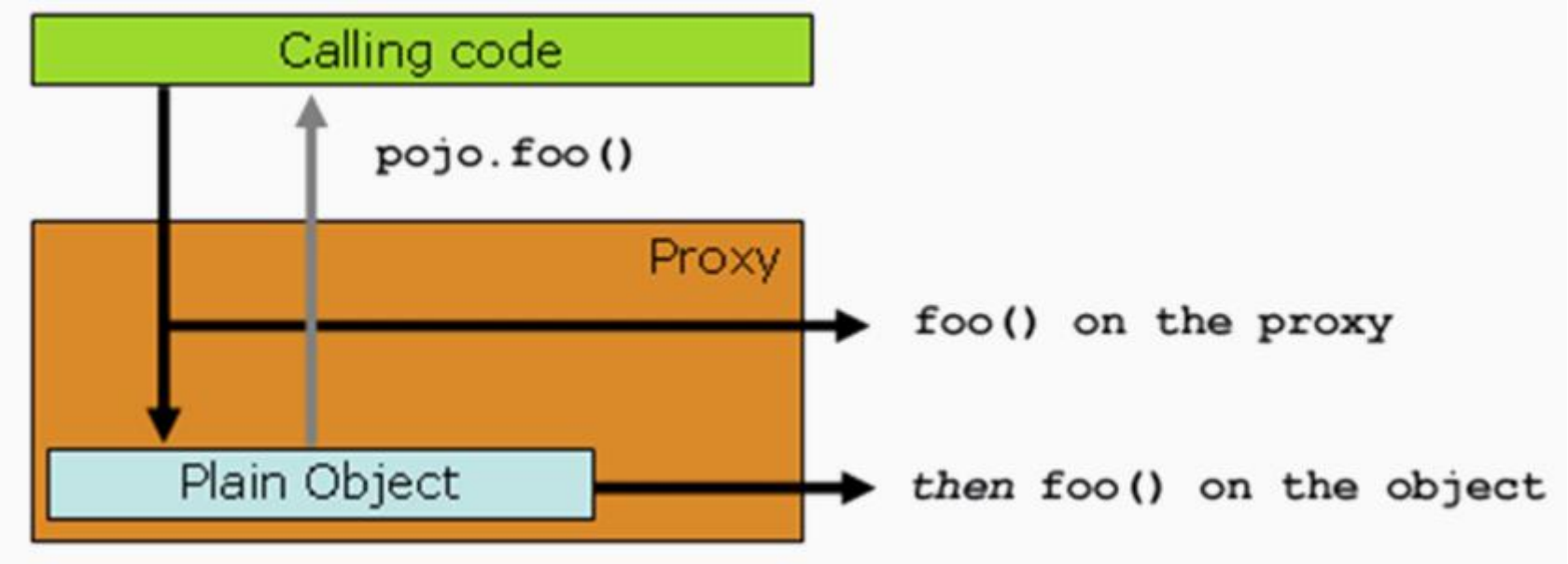
Spring AOP


Advice
BeforeAdvice
AfterAdvice
Interceptor
MethodBeforeAdvice
AfterReturningAdvice
ThrowsAdvice
MethodInterceptor

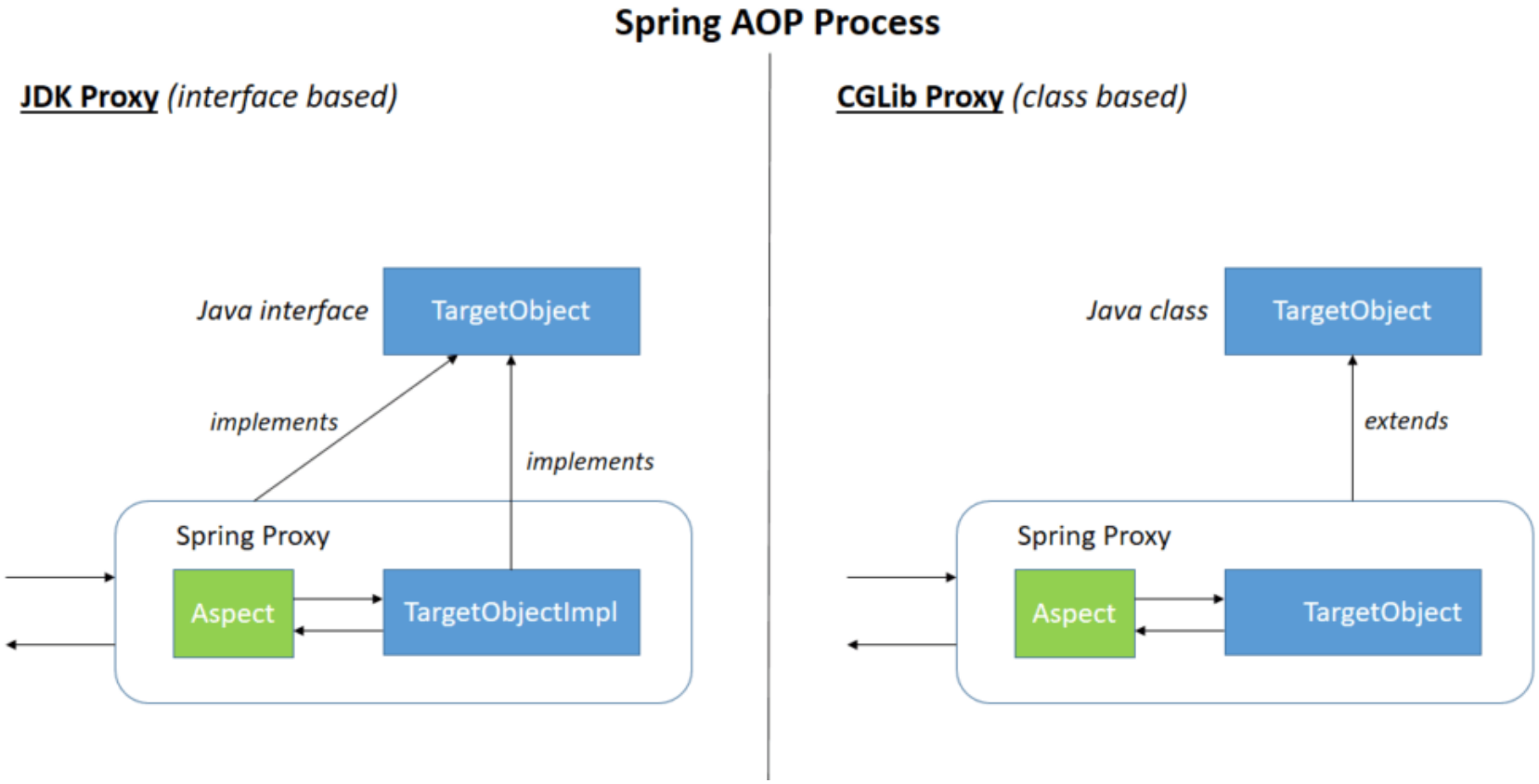
Spring AOP

```
public class AroundMethodInterceptor implements MethodInterceptor {
    private static Log LOGGER = LogFactory.getLog("Logger-AOP");

    public Object invoke(MethodInvocation methodInvocation) throws Throwable {
        Object[] args = methodInvocation.getArguments();
        String signature = methodInvocation.getMethod().toString();

        Object result = methodInvocation.proceed();

        LOGGER.info(String.format(
                "Method %s has called from class %s with return value type %s",
                signature,
                methodInvocation.getThis().getClass().toString(),
                result.getClass().getName()));

        return result;
    }

}
```

Spring AOP


Calling code
pojo.foo()
Proxy
foo() on the proxy
Plain Object
then foo() on the object

Spring AOP


Spring AOP Process
JDK Proxy (interface based)
Java interface
TargetObject
implements
implements
Spring Proxy
Aspect
TargetObjectImpl
CGLib Proxy (class based)
Java class
TargetObject
extends
Spring Proxy
Aspect
TargetObject

Spring AOP

Spring AOP	AspectJ
Implemented in pure Java	Implemented using extensions of Java programming language
No need for separate compilation process	Needs AspectJ compiler (ajc) unless LTW is set up
Only runtime weaving is available	Runtime weaving is not available. Supports compile-time, post-compile, and load-time Weaving
Less Powerful – only supports method level weaving	More Powerful – can weave fields, methods, constructors, static initializers, final class/methods, etc…
Can only be implemented on beans managed by Spring container	Can be implemented on all domain objects
Supports only method execution pointcuts	Support all pointcuts
Proxies are created of targeted objects, and aspects are applied on these proxies	Aspects are weaved directly into code before application is executed (before runtime)
Much slower than AspectJ	Better Performance
Easy to learn and apply	Comparatively more complicated than Spring AOP