

Your First Aspect

Eberhard Wolff
<http://ewolff.com>
eberhard.wolff@gmail.com



pluralsight 
hardcore developer training

Your First Aspect

- **Create an aspect**
- **Already get rid of some boiler plate code!**

What is an Aspect?

- Aspect implements cross cutting concern
- i.e. otherwise shattered throughout the code
- Get rid of the boiler plate code!

Aspect = *Pointcut* + *Advice*

t

*Where the
Aspect is
applied*

*What code
is executed*

Tracing

```
public void doSomething() {  
    final String METHODNAME = "doSomething";  
    logger.trace("entering " + CLASSNAME + "." + METHODNAME);  
    TransactionStatus tx = transactionManager.getTransaction(  
        new DefaultTransactionDefinition());  
    try {  
  
    } catch (RuntimeException ex) {  
        logger.error("exception in " + CLASSNAME + "." + METHODNAME, ex);  
        tx.setRollbackOnly();  
        throw ex;  
    } finally {  
        transactionManager.commit(tx);  
        logger.trace("exiting " + CLASSNAME + "." + METHODNAME);  
    }  
}
```

Simple Tracing Aspect

```
@Component
@Aspect
public class TracingAspect {

    Logger logger = Logger.getLogger(...);

    Advice: What is executed
    Before Advice: Before the original code
    @Before(
        "execution(void doSomething()) "
    )                                     Poincut expression
    public void entering() {           Method execution
        logger.trace("entering method");
    }

}
```

Simple Tracing Aspect for all methods

```
@Component
@Aspect
public class TracingAspect {

    Logger logger = Logger.getLogger(...);

    @Before(
        "execution(void doSomething()) "
    )
    public void entering() {
        logger.trace("entering method");
    }

}
```

Simple Tracing Aspect for all methods

```
@Component
@Aspect
public class TracingAspect {

    Logger logger = Logger.getLogger(...);

    @Before(
        "execution(* * (..)) "
        )
        type          Return      Metho
        wildcard      d name     parameters
    public void entering() {
        logger.trace("entering method");
    }
}
```

JoinPoint

- Point in the control flow of a program
- Advices can be presented with information about the join point
- E.g. method name, class name etc

Tracing Aspect With JoinPoint Information

```
@Component
@Aspect
public class TracingAspect {

    Logger logger =
        Logger.getLogger(TracingAspect.class.getName());

    @Before(
        "execution(void doSomething())"
    )
    public void entering() {
        logger.trace("entering method");
    }

}
```

Tracing Aspect With JoinPoint Information

```
@Component
@Aspect
public class TracingAspect {

    Logger logger =
        Logger.getLogger(TracingAspect.class.getName());

    @Before(
        "execution(void doSomething())"
    )
    public void entering(JoinPoint joinPoint) {
        logger.trace("entering "
            + joinPoint.getStaticPart().getSignature().toString());
    }
}
```

Enable Aspects in Spring XML Configuration

Enable @Aspect Annotation

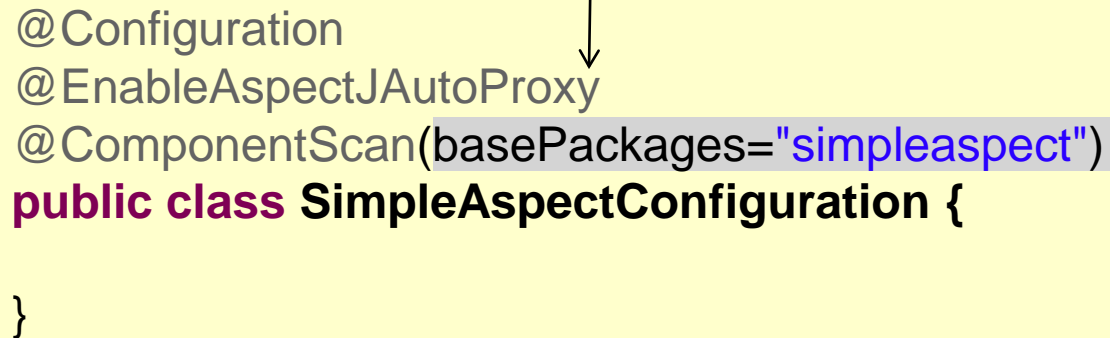
```
<beans>  
  
  <aop:aspectj-autoproxy />  
  
  <context:component-scan base-package="simpleaspect" />  
  
</beans>
```

Scan for Spring Beans

Enable Aspects in Spring XML Configuration

Enable @Aspect Annotation

Scan for Spring Beans



```
@Configuration
@EnableAspectJAutoProxy
@ComponentScan(basePackages="simpleaspect")
public class SimpleAspectConfiguration {

}
```

The diagram shows two annotations from the text above pointing to the code below. An arrow from "Enable @Aspect Annotation" points to `@EnableAspectJAutoProxy`. Another arrow from "Scan for Spring Beans" points to `@ComponentScan`.

Sum Up

- **Aspect =**
 - Advice – what code is executed +
 - Pointcut – where the code is executed
- **Aspects are Spring Beans**
 - Add @Aspect annotation
- **Advices are methods**
- **@Before annotation**
 - Contains pointcut expression
- **Activate AOP in XML or JavaConfig**