



CS544 EA  
Spring

AOP: Terminology

# Advice

- The **implementation of the cross cutting concern** is called advice.
  - Advice is implemented as a method in a class

```
@Aspect
@Component
public class LogAspect {
    private static final Logger logger = LogManager.getLogger(LogAspect.class.getName());

    @Before("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logBefore(JoinPoint joinpoint) {
        logger.warn("About to exec: " + joinpoint.getSignature().getName());
    }

    @After("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logAfter(JoinPoint joinpoint) {
        logger.warn("Just execed: " + joinpoint.getSignature().getName());
    }
}
```

Advice Method

Another Advice Method

# JoinPoint

- JoinPoint is a **specific point** (method) in code
  - **Where the advice will be applied**

```
@Service
public class CustomerService {

    public void doSomething() {
        System.out.println("something");
    }

    public void otherThing() {
        System.out.println("other");
    }

}
```

doSomething() is a JoinPoint

otherThing() is a JoinPoint

# Target

- While executing an advice, the **object on which the joinpoint is** located is called the target
  - Here target is an object of the CustomerService class

```
@Aspect
@Component
public class LogAspect {
    private static final Logger logger = LogManager.getLogger(LogAspect.class.getName());

    @Before("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logTargetBefore(JoinPoint joinpoint) {
        logger.warn("About to exec a method on: " + joinpoint.getTarget());
    }

    @After("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logTargetAfter(JoinPoint joinpoint) {
        logger.warn("Just execed a method on: " + joinpoint.getTarget());
    }
}
```

```
09:35:04.004 About to exec a method on: cs544.spring40.aop.terms.CustomerService@7bd7d6d6
09:35:04.033 Just execed a method on: cs544.spring40.aop.terms.CustomerService@7bd7d6d6
```

# Pointcut

- A Pointcut is a **collection of points**
  - Described in the Pointcut Expression Language

```
@Aspect
@Component
public class LogAspect {
    private static final Logger logger = LogManager.getLogger(LogAspect.class.getName());

    @Before("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logBefore(JoinPoint joinpoint) {
        logger.warn("Method: " + joinpoint.getSignature().getName());
    }
}
```

This PointCut expression says that all methods of CustomerService are JoinPoints

# Aspect

- Aspect is the combination of advice and pointcut
  - What (**advice**) should execute where (**pointcut**)

This class is an Aspect

```
@Aspect
@Component
public class LogAspect {
    private static final Logger logger = LogManager.getLogger(LogAspect.class.getName());

    @Before("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logBefore(JoinPoint joinpoint) {
        logger.warn("Method: " + joinpoint.getSignature().getName());
    }

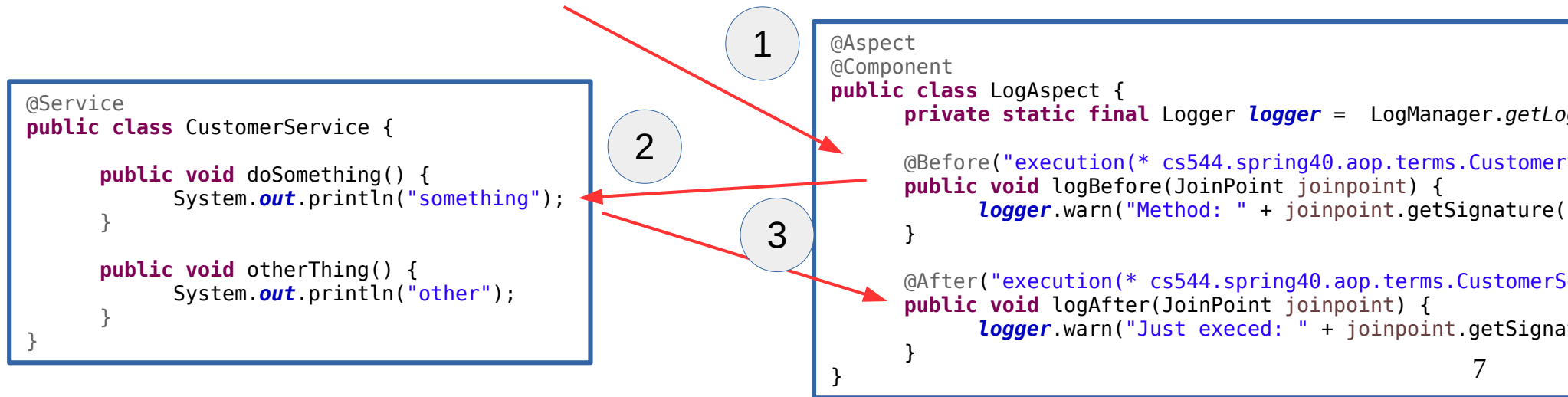
    @After("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logAfter(JoinPoint joinpoint) {
        logger.warn("Just execed: " + joinpoint.getSignature().getName());
    }
}
```

Pointcut

Advice

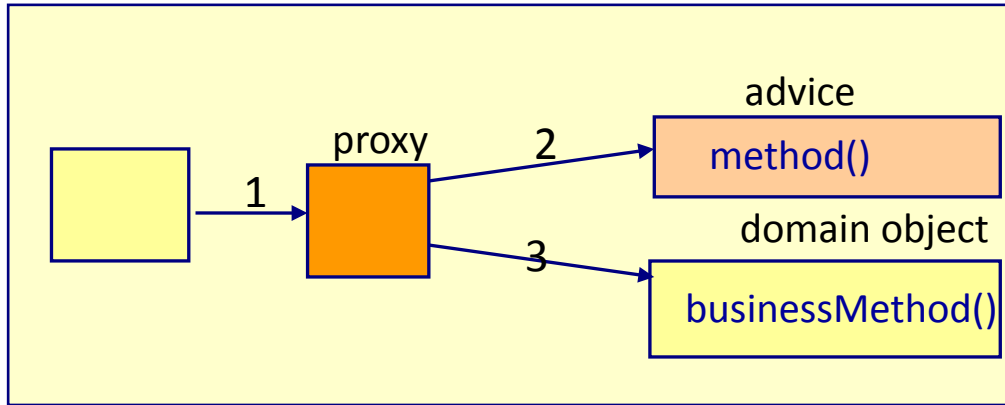
# Weaving

- Weaving is seen at execution time
  - **Execution weaves** back and forth between advice and the actual method
  - For example, when calling **doSomething()** on **CustomerService**

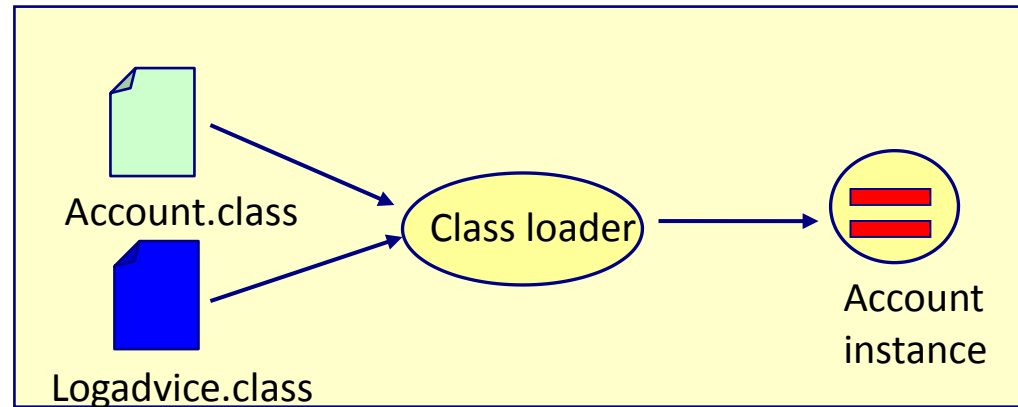


# Weaving

## Proxy-based weaving

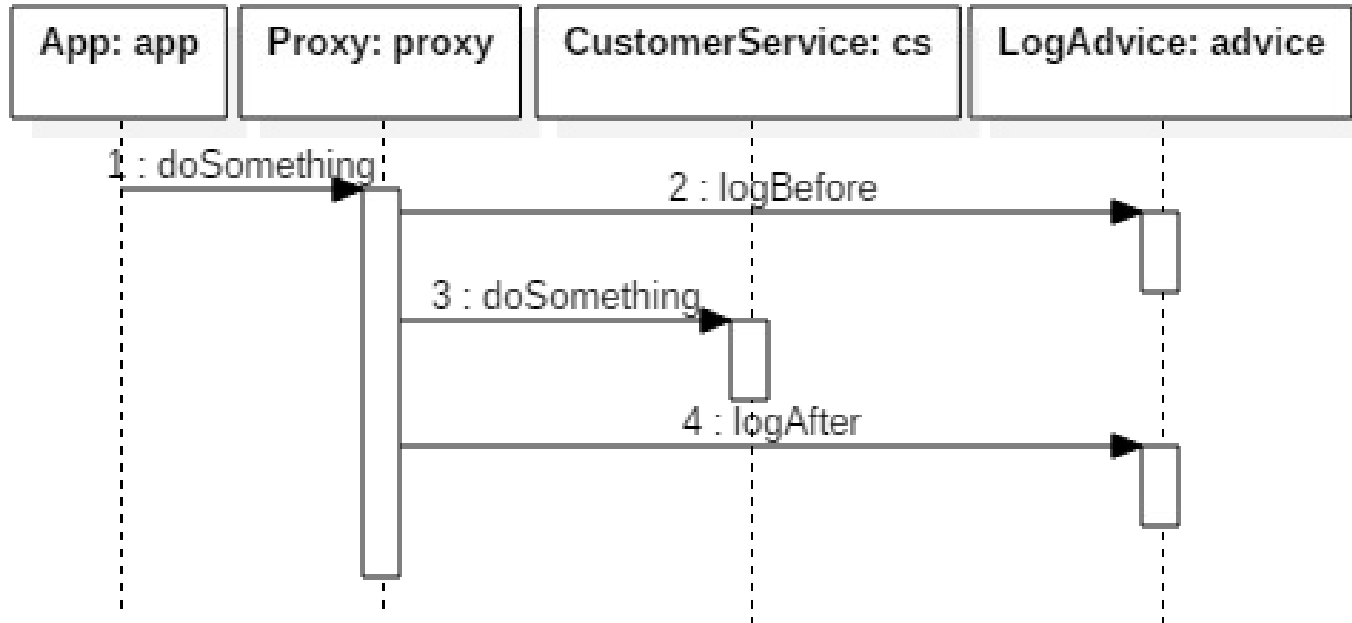


## Bytecode weaving





# Proxy Weaving Sequence Diagram



# Full Example Code

```
package cs544.spring40.aop.terms;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;
```

```
@Aspect
@Component
```

```
public class LogAspect {
    private static Logger logger = LogManager.getLogger(LogAspect.class.getName());

    @Before("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logBefore(JoinPoint joinpoint) {
        logger.warn("About to exec: " + joinpoint.getSignature().getName());
    }
    @After("execution(* cs544.spring40.aop.terms.CustomerService.*(..))")
    public void logAfter(JoinPoint joinpoint) {
        logger.warn("Just execed: " + joinpoint.getSignature().getName());
    }
}
```

Needs @Component to be a Bean

```
package cs544.spring40.aop.terms;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration
@ComponentScan("cs544.spring40.aop.terms")
@EnableAspectJAutoProxy
public class Config {
}
```

Tells Spring to look for AspectJ annotations on its beans and create proxies for them

# Full Example Code

```
package cs544.spring40.aop.terms;
```

```
import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class App {  
    public static void main(String[] args) {  
        ConfigurableApplicationContext context;  
        //context = new ClassPathXmlApplicationContext("cs544/spring40/aop/terms/springconfig.xml");  
        context = new AnnotationConfigApplicationContext(Config.class);  
        ICustomerService cs = context.getBean("customerService", ICustomerService.class);  
        cs.doSomething();  
  
        context.close();  
    }  
}
```

```
package cs544.spring40.aop.terms;
```

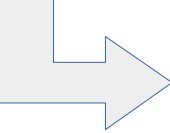
```
public interface ICustomerService {  
    void doSomething();  
    void otherThing();  
}
```

```
package cs544.spring40.aop.terms;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
```

```
public class CustomerService  
    implements ICustomerService{  
  
    public void doSomething() {  
        System.out.println("something");  
    }  
    public void otherThing() {  
        System.out.println("other");  
    }  
}
```



```
15:51:44.416 About to exec: doSomething  
something  
15:51:44.451 Just excec: doSomething
```

# XML Configuration

- Alternately XML can setup AspectJ annotations

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd">

  <aop:aspectj-autoproxy />
  <bean id="customerService" class="cs544.spring40.aop.terms.CustomerService" />
  <bean id="LogAspect" class="cs544.spring40.aop.terms.LogAspect" />
</beans>
```

Important:  
the AOP namespace

Tell spring to look for  
AspectJ annotations  
on its beans

LogAspect is a bean  
just like everything else  
(can also be injected into)

# Force CGLIB Proxies

- Proxy target class (instead of from interfaces)

```
package cs544.spring40.aop.terms;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration
@ComponentScan("cs544.spring40.aop.terms")
@EnableAspectJAutoProxy(proxyTargetClass=true)
public class Config {
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">

    <aop:aspectj-autoproxy proxy-target-class="true"/>
    <bean id="customerService" class="cs544.spring40.aop.terms.CustomerService" />
    <bean id="LogAspect" class="cs544.spring40.aop.terms.LogAspect" />
</beans>
```