

Bachelor of Science (BSc) in Informatik
Modul Advanced Software Engineering 1 (ASE1)

LE 03 – Spring Framework

Spring Core Framework Basics Aspect Oriented Programming

Institut für Angewandte Informationstechnologie (InIT)

Walter Eich (eicw) / Matthias Bachmann (bacn)

<https://www.zhaw.ch/de/engineering/institute-zentren/init/>

Lernziele LE 03 – Spring Framework

- Der Studierende
 - kennt die **Begriffe** der aspekt-orientierten Programmierung
 - Kennt den **Unterschied** zwischen der Spring-AOP und der AspectJ Implementierung
 - kann **aspekt-orientierte Programmierung** anwenden
 - kann Aspekte mit Hilfe einer **XML Konfigurationsdatei** deklarieren
 - kann Aspekte mit Hilfe von **Annotations** deklarieren
 - kann **Pointcuts** formulieren und kombinieren
 - Kann mittels **JoinPoint** Informationen ermitteln und mittels **ProceedingJoinPoint** zusätzlich noch den Programmfluss steuern
 - kann mittels **Introduction** das Konzept der Mehrfachvererbung zur Laufzeit anwenden

Agenda

3. Aspektorientierte Programmierung mit Spring

1. Was ist aspektorientierte Programmierung?
2. Grundlegende Begriffe der aspektorientierten Programmierung
3. Spring AOP und AspectJ
4. AOP mit Spring
5. Praktisches Beispiel von AOP
6. Pointcuts
7. JoinPoints
8. Introduction und die Mehrfachvererbung

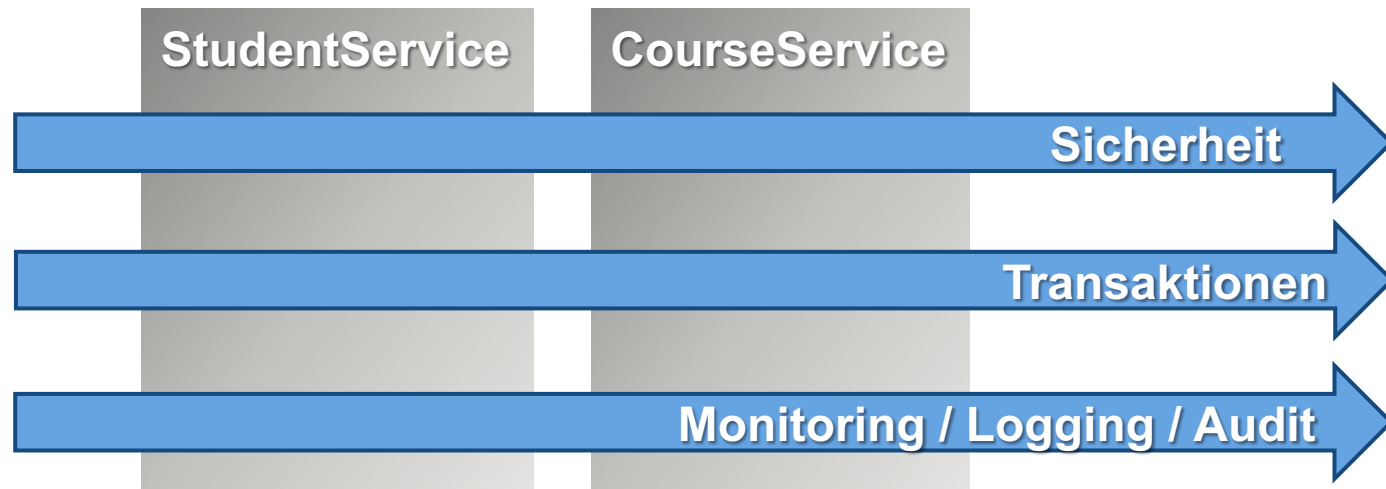
•

Objektorientierte Programmierung

- **Wiederbenutzbarkeit**
 - Klassen schachteln Funktionalität
 - Nur eine Vererbungshierarchie
- **Modularität - Code wird über viele Module verteilt**
 - Business Logik
 - Datenpersistenz
 - Benutzeroberfläche
- **Logging-Mechanismen**
 - Zu viel redundanter Code
- **Sicherheitsmechanismen**
 - Übergreifende Layer-Sicherheit
- **Transaktionen**

Problemstellung

- **CCC - , Cross-Cutting Concerns**
- Codeverteilung
 - Gleiche Problemstellung über mehrere Orte verteilt
- Codevermischung
 - Verschiedene Aspekte in einer Codebasis



Lösung

- **Aspektorientierte Programmierung**
 - Löst die Problemstellung mithilfe zentraler Aspekte
- **Beispiel Logging**
- **AOP erweitert OOP**
 - Ist kein eigenes Paradigma
 - Aspekte erweitern herkömmliche Logik um übergreifende Konzepte

Vorteile



- Modularer Code
 - Schachtelung externer Funktionalitäten in Module
- Anpassbarkeit
 - Entfernen und Hinzufügen von Aspekten zu jedem Zeitpunkt im Entwicklungszyklus
- Abhängigkeiten
 - Abhängigkeiten von diversen Bibliotheken befinden sich ausserhalb der Business-Logik

Anwendungsfälle

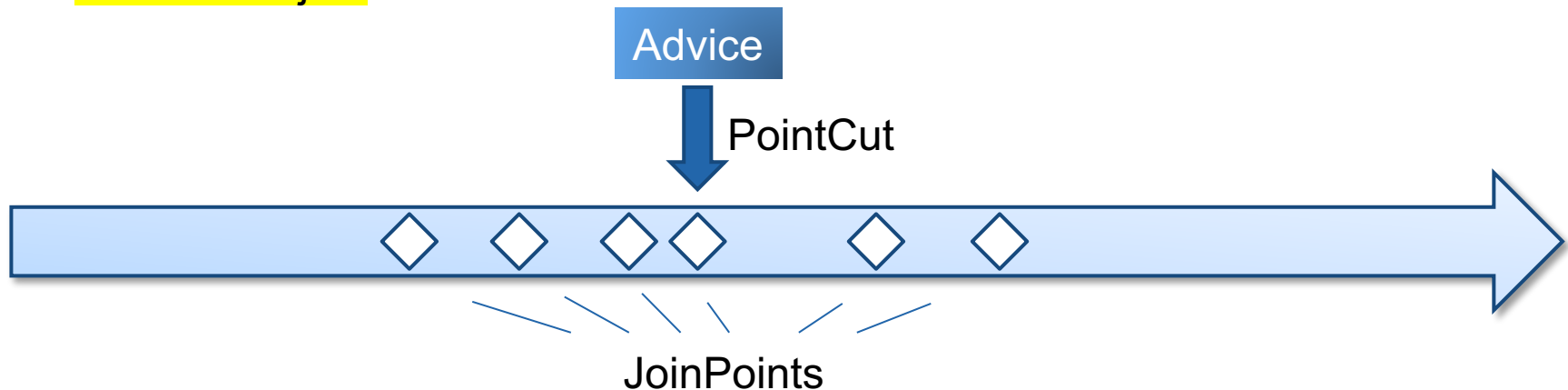
- **Transaktionsmanagement**
 - Übergreifende Kontrolle von Datenbankzugriff
 - Sicherstellung atomarer Zugriffe
- **Sicherheit**
 - Layer-übergreifende Sicherheitsmechanismen ausserhalb des «sensiblen» Codes
- **Überwachung**
 - Sicherstellung lückenloser Protokollierung von Vorgängen (Audits und Logging)
- **Events**
 - Abgreifen bestimmter Vorgänge
- **Debugging**
 - Protokollierung bestimmten Verhaltens

AOP Begriffe (1)

- **Aspekt**
 - Höchste Art der Kapselung
 - Sammelt verschiedene Anweisungen
 - Modularisierung eines einheitlichen Verhaltens über viele Objekte und Methoden hinweg
- **Advice** (Anweisung)
 - Methode
 - Wird vor anderen Methoden anderer Objekte ausgeführt
 - Code der an einem bestimmten Punkt ausgeführt wird (JoinPoint)
- **JoinPoint** (Einstiegspunkt)
 - Spezifiziert den Punkt, an dem etwas im regulären Code ausgeführt wird
 - Vor, nach oder stattdessen der Methode oder in Folge einer Exception

AOP Begriffe (2)

- **Pointcut**
 - Abfrageausdruck (definiert den/die Ort(e) des JoinPoints)
 - Welche Methode will ich mit meinem JoinPoint treffen
 - Methodenaufruf auf Klassen eines bestimmten Pakets
`edu.spring.*.*.main()`
- **Target** (Ziel)
 - Objekt, dessen Ausführung von AOP verändert wird
 - Advised Objekt



AOP Begriffe (3)

- **Weaving** (Prozess)
 - Der Prozess, der die Veränderungen am Java-Code durchführt
 - **Kompilierzeit (aspectJ)** -> benötigt den AspectJ Compiler)
 - **Laufzeit (AspectJ & Spring)** -> erst bei der Ausführung
- **Introduction** (Erweiterung)
 - Eine bestehende Klasse kann dynamisch um neue Methoden oder Felder erweitert werden
 - Mehrfachvererbung

Spring AOP

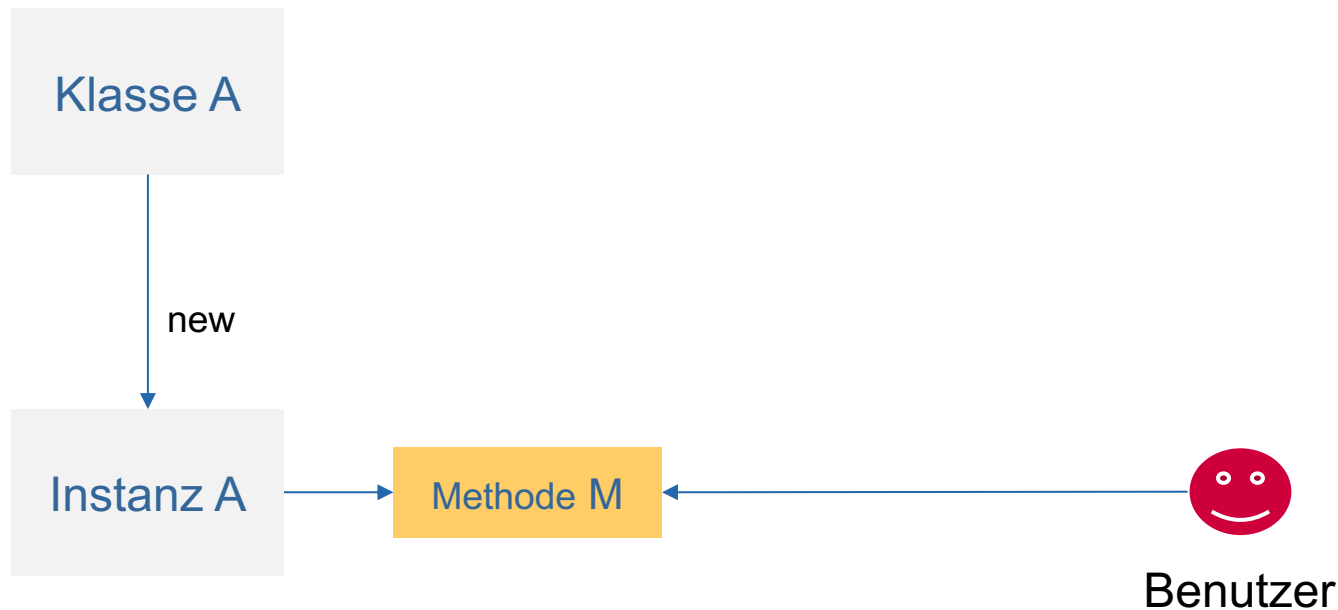
- Spring AOP
 - Basis des Spring Transaktionsmanagement
 - Spring Security
 - Komplett in der Bibliothek implementiert
- Spring AOP <-> AspectJ
 - Nur Laufzeitkompilierung
 - Spring AOP verfügt nur über ein Subset von AspectJ
 - Nur Methoden JoinPoints auf Beans
 - AspectJ ist viel flexibler -> benötigt allerdings den speziellen AspectJ Compiler

AspectJ

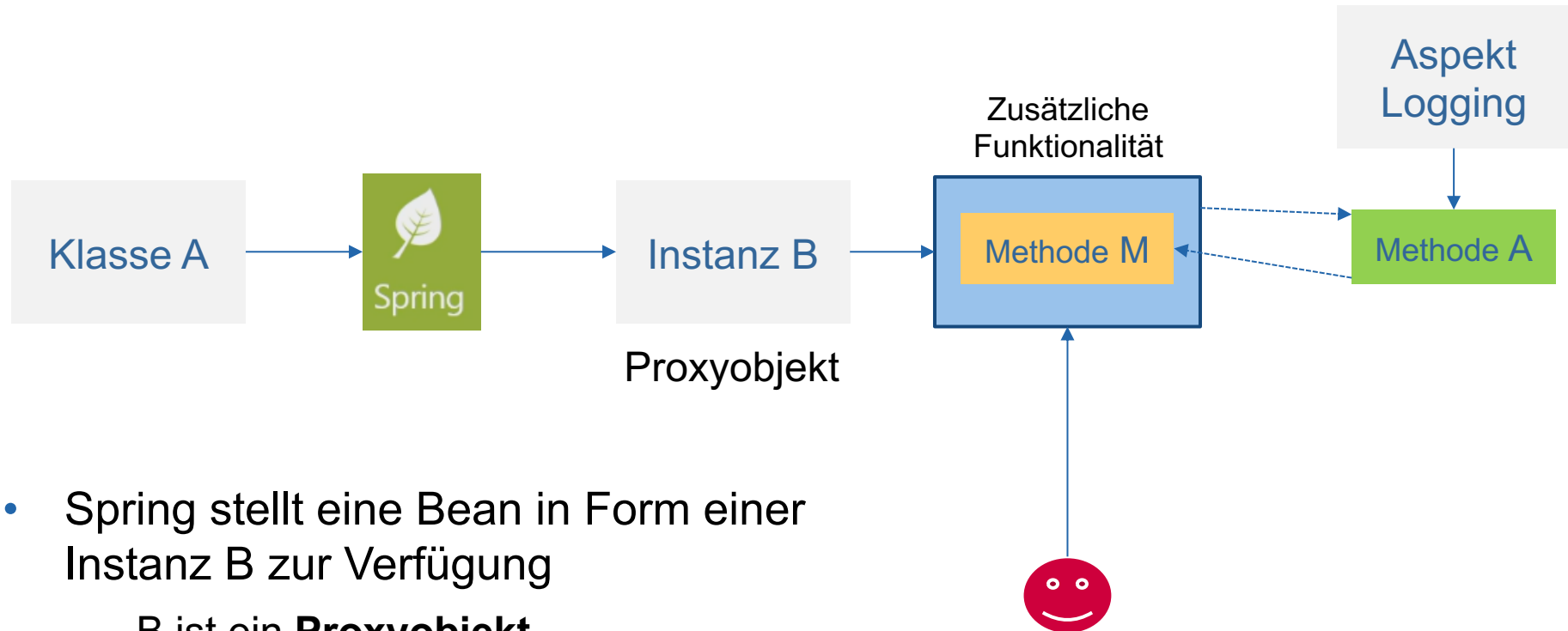
- Annotations
 - Nutzung eines Subset der AspectJ-Annotationen
 - Anwendbar für alle «Laufzeitaspekt»
- Keine Abhängigkeiten
 - **Kompiler und Weaver von AspectJ sind für Spring AOP nicht notwendig**

OOP ohne AOP

- Eine Instanz A wird mittels new erstellt und die Methode M aufgerufen



OOP mit AOP



- Spring stellt eine Bean in Form einer Instanz B zur Verfügung
 - B ist ein **Proxyobjekt**
 - Die Methode M verfügt über zusätzliche Funktionalität

Anpassung der XML-Konfiguration

```
<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-3.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

    <aop:aspectj-autoproxy />

    <aop:config>
        <aop:aspect id="aspectLogging" ref="logAspect"> ... </aop:aspect>
    </aop:config>

</beans>
```


Aspekt deklarieren

- Konvention Klassenname
 - **LoggingAspect**
- **@Aspect Annotation**
 - Entstammt der AspectJ API
 - `org.aspectj.lang.annotation.Aspect`
- Automatische Erkennung
 - AOP Autoproxy
 - Nutzung von **@Component** mit **@Aspect**

```
@Aspect
@Component
public class LoggingAspect {

}
```

Advice Arten (1)

- **Before**
 - Vor der Ausführung der Methode
- **After Throwing**
 - Sobald die Methode mit einer Exception beendet wurde
- **After Returning**
 - Sobald die Methode mit ihrer Rückgabe beendet wird
- **After Finally**
 - Egal unter welchen Bedingungen die Methode beendet wird

```
--Before--  
public void kontoBelasten(...) {  
    throw new Exception(); --After Throwing--  
    return true; --After Return--  
}
```

Advice Arten (2)

- **Around**
 - Steuert den Programmfluss
 - Umschliesst eine Methode komplett
 - Option, den **Methodenaufruf nicht vorzunehmen**
 - -> **Eigener Rückgabewert**
 - Nicht immer ideal zu nutzen
 - Aus **Performancegründen** nur verwenden falls notwendig


--Around--

```
public void kontoBelasten(...) {  
    throw new Exception();  
    return true;  
}
```

Advice und Pointcut

- Pointcut Ausdruck
 - Gibt den Ort des Pointcuts an
 - Beispiel: alle Methoden die mit **get** beginnen
- Wiederverwendung von PointCuts oder direkte Angabe
- Advice Ausdruck
- Arten
 - @Before
 - @ AfterReturning
 - @ AfterThrowing
 - @ Finally

```
public class LoggingAspect {  
  
    @Pointcut("execution(* get*(..))")  
    public void allGetters(){  
    }  
  
    @Before("allGetters()")  
    public void loggingAdvice(JoinPoint joinPoint)  
    ...  
}
```



```
public class LoggingAspect {  
  
    @Before("execution(* get*(..))")  
    public void loggingAdvice(JoinPoint joinPoint)  
    ...  
}
```

Around-Advice

- ProceedingJoinPoint
 - Kontrolle über die auszuführende Methode
 - proceed() um die Methode auszuführen
- Rückgabewerte
 - proceed gibt den Originalrückgabewert zurück
 - Beliebige Rückgabe möglich
 - Throwable, um mögliche Exceptions anzugeben

```
public class SurroundMeAdvice {  
    @Pointcut("...")  
    public void surroundMe() {  
    }  
  
    @Around("surroundMe()")  
    public Object  
    aroundSurroundMe(ProceedingJoinPoint pjp)  
    throws Throwable {  
        return pjp.proceed();  
    }  
}
```

Beispiel: 31-AOP-Auto: POM.XML

The screenshot displays an IDE interface with three main panels:

- Project Explorer:** Shows a project structure for '31 - AOP-Auto'. It includes 'Spring Elements', 'JRE System Library [JavaSE-1.8]', 'Maven Dependencies' (listing various Spring and JUnit jars), and source code directories 'src/main/java' (containing 'edu.spring.aop' and 'example') and 'src/main/resources'. The 'aop-config.xml' file is highlighted under 'src/main/resources'.
- Code Editor:** Displays the content of '31 - AOP-Auto/pom.xml'. The XML defines project properties (junit.version), dependencies (Spring AOP, Spring and Transactions, AspectJ, AspectJ weaver), and build plugins (Maven compiler plugin).
- Outline:** Provides a hierarchical view of the pom.xml content, showing sections like 'modelVersion', 'groupId', 'artifactId', 'version', 'properties', 'dependencies', and 'build'.

```
<!-- Test -->
<junit.version>4.11</junit.version>
</properties>

<dependencies>
  <!-- Spring AOP -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring-framework.version}</version>
  </dependency>

  <!-- Spring and Transactions -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring-framework.version}</version>
  </dependency>

  <!-- AspectJ -->
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${aspectj.version}</version>
  </dependency>

  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${aspectj.version}</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.5.1</version>
    </plugin>
  </plugins>
</build>
```

Beispiel: 31-AOP-Auto: Ausführen

Aug 31, 2016 3:17:23 PM

org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh
INFORMATION: Refreshing

org.springframework.context.support.ClassPathXmlApplicationContext@817b38:
startup date [Wed Aug 31 15:17:23 CEST 2016]; root of context hierarchy

Aug 31, 2016 3:17:23 PM

org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFORMATION: Loading XML bean definitions from class path resource [aop-config.xml]

Auto Tankauftrag: 20

Motor Startversuch

Vor dem Motorstart

Motor gestartet

Nach dem Motorstart

Fahrzeug fährt los..

Beispiel: 31-AOP-Auto: Implementierung

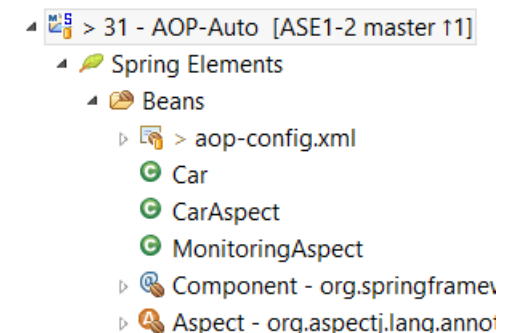
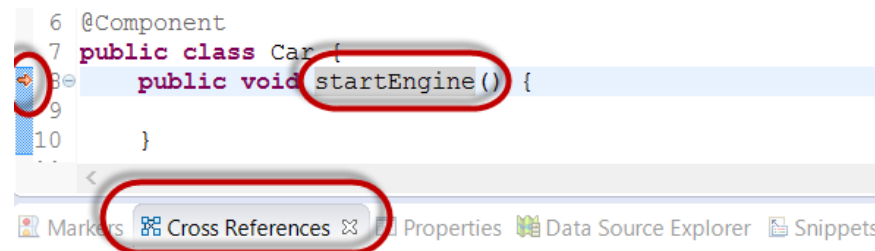
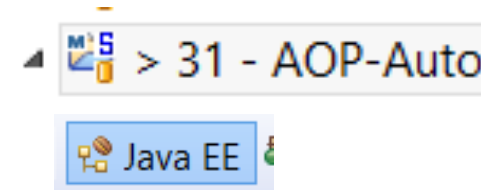
```
public class MainAOPAuto {  
  
    public static void main(String[] args) {  
  
        AbstractApplicationContext context = ...  
  
        Car car = context.getBean(Car.class);  
        car.addFuel(20);  
        car.startEngine();  
        car.drive();  
        context.close();  
    }  
}
```

```
@Component  
public class Car {  
    public void startEngine() {  
  
    }  
  
    public void addFuel(int litres) {  
  
    }  
  
    public void drive() {  
  
    }  
}
```

Auto Tankauftrag: 20
Motor Startversuch
Vor dem Motorstart
Motor gestartet
Nach dem Motorstart
Fahrzeug fährt los..

Beispiel: Advice Navigation mit Eclipse

- Voraussetzungen:
 - Spring Projekt (ein **S** ist im Projekt sichtbar)
 - JEE Perspektive
 - Im Projekt Explorer sind unter Spring Elements die Beans sichtbar:
falls nicht dann: **Project->Properties->Spring->Beans Support -> [SCAN]**
 - Öffnen der View AspectJ -> CrossReference
 - Dependency: org.springframework.spring-aspects hinzufügen
 - Kontext Menü auf Projekt: Spring Tools-> Enable Spring Aspects Tooling
Weaving wird durchgeführt -> kann durch AspectJ -> Remove entfernt werden



Beispiel: Advice Navigation mit IntelliJ

- In File->Project Structure unter Facets, den Kontext erstellen

The screenshot illustrates the setup for AOP advice navigation in IntelliJ IDEA. It shows three main components:

- Project Structure Dialog:** The 'Facets' tab is selected, showing a 'Spring' facet with several sub-facets (31-AOP-Auto, 32-AOP-Pointcut, 33-AOP-JoinPoints, 34-AOP-Introduction) and a 'Detection' sub-facet.
- Code Snippet:** A code snippet showing an `@Around("all()")` annotation and a `printParametersAndReturnVal` method. A red box highlights the `@` symbol, and a tooltip 'Navigate To Advised Methods' is visible.
- Spring Beans Configuration:** The 'Beans' tab is selected, showing a list of beans including `(edu.spring) [Component]`, `<aop:aspectj-autoproxy ... />`, `<aop:config ... />`, `car`, `carAspect`, `monitoringAspect`, and `xmlAspect`.

The bottom status bar shows the 'Spring' tab is active, along with other tabs like 'Run', 'Problems', 'Git', 'Terminal', 'Services', 'Build', and 'TODO'.

Beispiel: 31-AOP-Auto: CarAspect

- `beforeStartEngine()` / `startEnginePointCut`

```
@Pointcut("execution(* edu.spring.aop.example.Car.startEngine())")  
public void startEnginePointcut(){}  

```

```
@Before("startEnginePointcut()")  
public void beforeStartEngine() throws Throwable {  
    System.out.println("Vor dem Motorstart"); }  

```

- `beforeAddFuel(int litres)` / `addFuelPointcut`

```
@Pointcut(value = "execution(* edu.spring.aop.example.Car.addFuel(..)) && args(litres)")  
public void addFuelPointcut(int litres){}
```

```
@Before(value = "addFuelPointcut(litres)")  
public void beforeAddFuel(int litres) {  
    System.out.println("Auto Tankauftrag: " + litres); }  

```

- `@After`

```
@After("startEnginePointcut()")  
public void afterStartEngine() throws Throwable {  
    System.out.println("Nach dem Motorstart");  
}
```

Beispiel: 31-AOP-Auto: CarAspect

- aroundStartEngine

```
@Around("execution(* edu.spring.aop.example.Car.startEngine())")
public void aroundStartEngine(ProceedingJoinPoint joinPoint) throws Throwable
{
    System.out.println("Motor Startversuch");

    Random rn = new Random();
    if (rn.nextInt(100) < 50) {
        joinPoint.proceed();
        System.out.println("Motor gestartet");
    }
    else
        System.out.println("Motorstart fehlgeschlagen");
}
```

- Hat **Priorität** zu @Before
- 50% **Wahrscheinlichkeit** für joinPoint.proceed()
 - @Before Advice wird nur bei joinPoint.proceed() aufgerufen

Beispiel: 31-AOP-Auto: SampleXMLAspect

```
public class SampleXMLAspect {  
  
    public void logMessage() {  
        System.out.println("Fahrzeug fährt los..");  
    }  
  
}
```

```
<!-- XML Konfiguration -->  
<bean id="xmlAspect" class="edu.spring.aop.xml.SampleXMLAspect"/>  
  
<aop:config>  
    <aop:aspect ref="xmlAspect">  
        <aop:before method="logMessage"  
            pointcut="execution(* edu.spring..*.Car.drive(..))"/>  
    </aop:aspect>  
</aop:config>
```

Beispiel: 31-AOP-Auto: MonitoringAspect

```
@Component
@Aspect
public class MonitoringAspect {
    @Pointcut("execution(* edu.spring.aop.example..*(..))")
    private void all() {}

    @Around("all()")
    public Object printParametersAndReturnVal
        (ProceedingJoinPoint joinPoint) throws Throwable {

        System.out.println(". Klasse      : " +
                           joinPoint.getTarget().getClass());
        System.out.println(". Methode   : " +
                           joinPoint.getSignature().getName());
        System.out.print (" . Argumente : ");
        for(Object arg : joinPoint.getArgs())
            System.out.print(arg + " / ");

        Object ret = joinPoint.proceed();
        System.out.println("\n. Return      : " + ret + "\n");
        return ret;
    }
}
```

```
Auto Tankauftrag: 20
. Klasse      : class
  edu.spring.aop.example.Car
. Methode     : addFuel
. Argumente   : 20 /
. Return      : null
```

```
Motor Startversuch
Motorstart fehlgeschlagen
Nach dem Motorstart
Fahrzeug fährt los..
. Klasse      : class
  edu.spring.aop.example.Car
. Methode     : drive
. Argumente   :
. Return      : null
```

Beispiel: 31-AOP-Auto: aop-config.xml

```
<?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:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
...
       <context:component-scan base-package="edu.spring"/>

       <aop:aspectj-autoproxy/>

       <!-- XML Konfiguration -->
       <bean id="xmlAspect" class="edu.spring.aop.xml.SampleXMLAspect"/>

       <aop:config>
           <aop:aspect ref="xmlAspect">
               <aop:before method="LogMessage"
                           pointcut="execution(* edu.spring..*.Car.drive(..))"/>
           </aop:aspect>
       </aop:config>
</beans>
```

Pointcuts - Ausdruckarten

- Syntax Execution Pointcut:

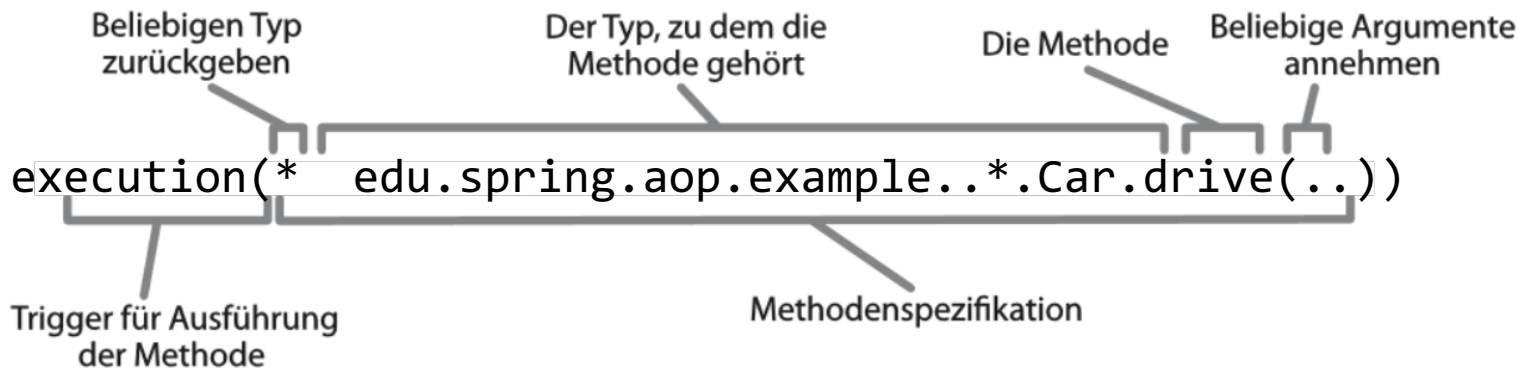
`execution`(Sichtbarkeit(opt) Returntype Ziel(Parameter))

Ausdruck	Beschreibung
<code>execution()</code>	Entspricht Joinpoints, die Methodenausführungen sind
<code>within()</code>	Restriktion auf JoinPoints innerhalb bestimmter Typen
<code>args()</code>	Argumente des Ziels müssen von bestimmtem Typ sein
<code>this()</code>	JoinPoint im aktuellen Type (Instanz der Proxyklasse)
<code>target()</code>	Zielobjekt muss Instanz des bestimmten Typs sein

<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html#aop-pointcuts>

Pointcuts Beispiele – execution()

- `execution(modifiers-pattern? ret-type-pattern declaring-type-pattern?name-pattern(param-pattern) throws-pattern?)`
? = optional



- Alle **privaten** Methoden: `execution(private * *(..))`
- Alle Methoden **unterhalb** des Package: `execution(* edu.spring.aop.example..*(..))`
- Alle Methoden die mit **get** beginnen: `execution(* get*(..))`

Pointcuts Beispiele – within()

- Syntax: `within(Typ)`
- Alle Methoden **direkt unterhalb** von `edu.spring.aop`:
 - `within(edu.spring.aop.*)`
- Alle Methoden **unterhalb** von `edu.spring.aop`:
 - `within(edu.spring.aop..*)`

Pointcuts Beispiele – this (), target(), args()

- Syntax: `this(Typ)`
 - Alle Methoden bei denen das Advised-Objekt `SecurityRelevant` implementiert wird:
`this(edu.spring.aop.example.SecurityRelevant)`
- Syntax: `target(Typ)`
 - Alle Methoden bei denen das Interface implementiert wird:
`target(edu.spring.aop.example.SecurityRelevant)`
- Syntax: `args(Typ)`
 - Alle Methoden die String als Argument haben:
`args(java.lang.String)`

Pointcuts Beispiele – Kombination

- Operatoren
 - && - logisches Und
 - || - logisches Oder
 - ! – logische Umkehrung

```
@Before("execution(* get*(..)) && target(eu.*.Security)")
```

```
@Before("execution(* get*(..)) || execution(* set*(..))")
```

```
@Before("methode1() || methode2()")
```

```
@Pointcut("...")
```

```
public void methode1()
```

```
@Pointcut("...")
```

```
public void methode2()
```

Beispiel – pom.xml

The screenshot displays an IDE interface with the following components:

- Project Explorer:** Shows the project structure for '32 - AOP-Pointcut [ASE1-2 master 12]'. It includes 'Spring Elements' (Beans), 'JRE System Library [JavaSE-1.8]', 'src/main/java' (containing 'edu.spring.aop.pointcut' with 'example' sub-package), 'Maven Dependencies', and 'src/main/resources' (containing 'aop-config.xml').
- Editor:** Displays the 'pom.xml' file with the following content:

```
25 <dependencies>
26 <!-- Spring AOP -->
27 <dependency>
28   <groupId>org.springframework</groupId>
29   <artifactId>spring-aop</artifactId>
30   <version>${spring-framework.version}</version>
31 </dependency>
32 <dependency>
33   <groupId>org.springframework</groupId>
34   <artifactId>spring-aspects</artifactId>
35   <version>4.3.2.RELEASE</version>
36 </dependency>
37
38 <!-- Spring and Transactions -->
39 <dependency>
40   <groupId>org.springframework</groupId>
41   <artifactId>spring-context</artifactId>
42   <version>${spring-framework.version}</version>
43 </dependency>
44
45
46 <!-- AspectJ -->
47 <dependency>
48   <groupId>org.aspectj</groupId>
49   <artifactId>aspectjrt</artifactId>
50   <version>${aspectj.version}</version>
51 </dependency>
52
53 <dependency>
54   <groupId>org.aspectj</groupId>
55   <artifactId>aspectjweaver</artifactId>
56   <version>${aspectj.version}</version>
57 </dependency>
```
- Outline:** Shows the project's properties and dependencies, including 'Generic properties', 'Spring', 'Test', 'dependencies', and 'plugins'.
- Task List:** Empty.
- Bottom Bar:** Includes tabs for 'Overview', 'Dependencies', 'Dependency Hierarchy', 'Effective POM', and 'pom.xml'. The 'pom.xml' tab is active.

Beispiel – aop-config.xml

```
<?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"
       xmlns:context="http://www.springframework.org/schema/context"
       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
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
    <aop:aspectj-autoproxy proxy-target-class="true"/>
    <context:component-scan base-package="edu.spring.aop.pointcut"/>
</beans>
```

Beispiel – Main Klasse

```
public class MainAopPointcut {  
  
    public static void main(String[] args)  
    {  
        AbstractApplicationContext context =  
            new ClassPathXmlApplicationContext("aop-config.xml");  
        HelloWorldService helloWorldService =  
            (HelloWorldService) context.getBean("helloWorldService");  
  
        System.out.println(helloWorldService.getMessage());  
        System.out.println(helloWorldService.getBaseMessage());  
        System.out.println(((ProxyInterface)helloWorldService).getProxyMessage());  
        context.close();  
    }  
}
```

```
Target Advice ausgeführt  
This Advice ausgeführt  
Within Advice ausgeführt  
Hello World!  
Target Advice ausgeführt  
This Advice ausgeführt  
Hello from Base!  
Hello from Proxy!
```

Beispiel – Pointcut Aspekt

```
@Aspect @Component
public class PointcutAspect {

    @Pointcut("execution(* get*())")
    public void allGetMethods() {}

    @Pointcut("target(edu.spring.aop.pointcut.example.HelloWorldService)")
    public void helloWorldService() {}

    @Pointcut("within(edu.spring.aop.pointcut.example.HelloWorldService)")
    public void helloWorldWithin() {}

    @Pointcut("this(edu.spring.aop.pointcut.introduction.ProxyInterface)")
    public void helloWorldThis() {}

    @Before("allGetMethods() && helloWorldService()")
    public void targetAdvice() { System.out.println("Target Advice ausgeführt"); }

    @Before("allGetMethods() && helloWorldWithin()")
    public void withinAdvice() { System.out.println("Within Advice ausgeführt"); }

    @Before("allGetMethods() && helloWorldThis()")
    public void thisAdvice() { System.out.println("This Advice ausgeführt"); }

}
```


Beispiel – HelloWorldService

```
public class HelloWorldBase {  
  
    public String getBaseMessage() {  
        return "Hello from Base!";  
    }  
}
```

```
public class ProxyInterfaceImpl  
implements ProxyInterface {  
    public String getProxyMessage() {  
        return "Hello from Proxy!";  
    }  
}
```

```
@Before("allGetMethods() && helloWorldService()")  
@Before("allGetMethods() && helloWorldThis()")  
@Before("allGetMethods() && helloWorldWithin()")
```

```
@Component("helloWorldService")  
public class HelloWorldService extends  
HelloWorldBase {  
    public String getMessage() {  
        return "Hello World!";  
    }  
}
```

1. Fall

Target Advice ausgeführt
This Advice ausgeführt
Within Advice ausgeführt
Hello World!

2. Fall

Target Advice ausgeführt
This Advice ausgeführt
Hello from Base!

3. Fall

Hello from Proxy!

```
1. Fall System.out.println(helloWorldService.getMessage());  
2. Fall System.out.println(helloWorldService.getBaseMessage());  
3. Fall System.out.println(((ProxyInterface)helloWorldService).getProxyMessage());
```

JoinPoint

- **Arten** von JoinPoints
 - **JoinPoint** (alle Advices)
 - **ProceedingJoinPoint** (nur Around Advice)
- **Zweck** JoinPoint
 - Hilft bei der Steuerung des Programmflusses (nur ProceedingJoinPoint)
 - Baut auf dem JoinPoint auf
 - Hilft dabei, Informationen über den aktuellen Kontext abzurufen
 - **Welches Objekt**
 - **Welche Methode wurde aufgerufen**
 - **Welche Argumente**

Beispiel: 33 – AOP-JoinPoint – pom.xml

The screenshot displays an IDE interface with the following components:

- Project Explorer (Left):** Shows the project structure for '33-AOP-JoinPoints'. It includes 'src/main/java' with 'example' containing 'BankAccount.java' and 'HelloWorldService.java', and 'src/main/resources' with 'aop-config.xml'. It also lists 'Maven Dependencies' such as 'spring-aop-4.3.2.RELEASE.jar', 'spring-beans-4.3.2.RELEASE.jar', 'spring-core-4.3.2.RELEASE.jar', 'commons-logging-1.2.jar', 'spring-aspects-4.3.2.RELEASE.jar', 'spring-context-4.3.2.RELEASE.jar', 'spring-expression-4.3.2.RELEASE.jar', 'aspectjrt-1.8.2.jar', 'aspectjweaver-1.8.2.jar', 'junit-4.11.jar', and 'hamcrest-core-1.3.jar'.
- Editor (Center):** Displays the 'pom.xml' file for '33-AOP-JoinPoints'. The code is as follows:

```
25 <dependencies>
26 <!-- Spring AOP -->
27 <dependency>
28   <groupId>org.springframework</groupId>
29   <artifactId>spring-aop</artifactId>
30   <version>${spring-framework.version}</version>
31 </dependency>
32 <dependency>
33   <groupId>org.springframework</groupId>
34   <artifactId>spring-aspects</artifactId>
35   <version>4.3.2.RELEASE</version>
36 </dependency>
37
38 <!-- Spring and Transactions -->
39 <dependency>
40   <groupId>org.springframework</groupId>
41   <artifactId>spring-context</artifactId>
42   <version>${spring-framework.version}</version>
43 </dependency>
44
45
46 <!-- AspectJ -->
47 <dependency>
48   <groupId>org.aspectj</groupId>
49   <artifactId>aspectjrt</artifactId>
50   <version>${aspectj.version}</version>
51 </dependency>
52
53 <dependency>
54   <groupId>org.aspectj</groupId>
55   <artifactId>aspectjweaver</artifactId>
56   <version>${aspectj.version}</version>
57 </dependency>
```
- Outline (Right):** Shows the project's configuration structure, including 'name', 'properties', 'dependencies', 'build', and 'plugins'.
- Bottom Bar:** Includes tabs for 'Overview', 'Dependencies', 'Dependency Hierarchy', 'Effective POM', and 'pom.xml'. Below these are icons for 'Markers', 'Cross References', 'Properties', 'Data Source Explorer', 'Snippets', 'Problems', 'Console', and 'Debug'.

Beispiel: 33 – AOP-JoinPoint – aop-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:context="http://www.springframework.org/schema/context"
       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/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">

    <aop:aspectj-autoproxy/>
    <context:component-scan base-package="edu.spring.aop"/>

</beans>
```

Beispiel: 33 – AOP-JoinPoint – Main

```
public static void main(String[] args) {  
    AbstractApplicationContext context =  
        new ClassPathXmlApplicationContext("aop-config.xml");  
    HelloWorldService helloWorldService =  
        (HelloWorldService) context.getBean("helloWorldService");  
  
    BankAccount meinKonto = (BankAccount) context.getBean("bankAccount");  
    meinKonto.setAccountID(4);  
    meinKonto.insertMoney(200);  
  
    helloWorldService.getMessage();  
    context.close();  
}
```

BankAccount und HelloWorldService

```
@Component("bankAccount")
@Scope("prototype")
public class BankAccount {

    private int accountID = 0;
    private double balance = 0;

    public void insertMoney(double amount) {
        setBalance(getBalance() + amount);
        System.out.println("Es wurden " + amount +
            "€ auf " + accountID + " eingezahlt");
    }
}
```

```
@Component("helloWorldService")
public class HelloWorldService {

    public String getMessage() {
        return "Hello World!";
    }
}
```

Klaue 200.0€

Es wurden 200.0€ auf 0 eingezahlt

[LOG] - [edu.spring.aop.joinpoint.example.HelloWorldService] - Method: getMessage

Beispiel: 33 - StealerAspect

```
@Aspect
@Component
public class StealerAspect {

    @Autowired
    ApplicationContext context;

    @Autowired
    BankAccount nummernKonto;

    @Around("execution(* edu.spring..*.BankAccount.insertMoney(..))")
    public Object stealMoney(ProceedingJoinPoint joinPoint) throws Throwable {
        if (joinPoint.getTarget().equals(nummernKonto)) {
            return joinPoint.proceed();
        }
        double amount = (Double)joinPoint.getArgs()[0];

        System.out.println("Klaue " + amount + "€");
        nummernKonto.insertMoney(amount);

        return true;
    }
}
```

Beispiel: 33 - LoggingAspect

```
@Aspect
@Component
public class LoggingAspect {

    @Before("target(edu.spring.aop.joinpoint.example.HelloWorldService)")
    public void logging(JoinPoint joinPoint) {
        Object[] args = joinPoint.getArgs();

        System.out.println(String.format(
            "[LOG] - [%s] - Method: %s",
            joinPoint.getSignature().getDeclaringTypeName(),
            joinPoint.getSignature().getName()
        ));

        if (args.length > 0) {
            StringBuilder builder = new StringBuilder("Arguments:\n");
            for (Object o : args){
                builder.append(o.toString());
                builder.append('\n');
            }
        }
    }
}
```


Introduction

- **Mehrfachvererbung**
 - Implementierung eines Interface, das vorher nicht deklariert was, auf Objekten zur Laufzeit
- **Zweck**
 - Bestehende Klasse um Funktionalitäten erweitern, ohne explizite Veränderungen
- **Zwei Arten von Introduction**
 - Neues Verhalten
 - Neue Variablen (neuer «State»)

Introduction deklarieren

- Klasse
 - Konvention: ...Introduction
- **@DeclareParents-Annotation**
 - Konfiguriert eine Implementierungsklasse für ein Interface auf eine Zielklasse
- **Value = Zielklasse(String)**
 - defaultImpl=StandardImplementierung (Ref)
 - Variable deklarierte Interface

```
@Aspect
@Component
public class HWIntroduction {

    @DeclareParents(
        value = "edu...HelloWorldService",
        defaultImpl = PrinterImpl.class)
    public Printer printer;

}
```

Beispiel: 34-AOP-Introduction

The screenshot displays an IDE interface for a Spring AOP introduction example. The Project Explorer on the left shows the project structure, including the source code and resources. The central editor shows the `aop-config.xml` file, which defines the dependencies for the application. The right sidebar shows the Outline view, listing the properties and dependencies of the project.

Project Explorer:

- 34-AOP-Introduction [ASE1-2 master t6]
 - Spring Elements
 - src/main/java
 - edu.spring.aop.introduction
 - fancy
 - FancyMessagePrinter.java
 - FancyMessagePrinterImpl.java
 - HelloWorldIntroduction.java
 - HelloWorldIntroductionService.java
 - MainAopIntroduction.java
 - MainAopIntroductionService.java
 - src/main/resources
 - aop-config.xml
 - src/test/java
 - src/test/resources
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - spring-aop-4.3.2.RELEASE.jar - F:\m2\repository\org\springframework\spring-aop\4.3.2.RELEASE\spring-aop-4.3.2.RELEASE.jar
 - spring-beans-4.3.2.RELEASE.jar - F:\m2\repository\org\springframework\spring-beans\4.3.2.RELEASE\spring-beans-4.3.2.RELEASE.jar
 - spring-core-4.3.2.RELEASE.jar - F:\m2\repository\org\springframework\spring-core\4.3.2.RELEASE\spring-core-4.3.2.RELEASE.jar
 - commons-logging-1.2.jar - F:\m2\repository\commons-logging\commons-logging\1.2\commons-logging-1.2.jar
 - spring-aspects-4.3.2.RELEASE.jar - F:\m2\repository\org\springframework\spring-aspects\4.3.2.RELEASE\spring-aspects-4.3.2.RELEASE.jar
 - spring-context-4.3.2.RELEASE.jar - F:\m2\repository\org\springframework\spring-context\4.3.2.RELEASE\spring-context-4.3.2.RELEASE.jar
 - spring-expression-4.3.2.RELEASE.jar - F:\m2\repository\org\springframework\spring-expression\4.3.2.RELEASE\spring-expression-4.3.2.RELEASE.jar
 - aspectjrt-1.8.2.jar - F:\m2\repository\org\aspectj\aspectjrt\1.8.2\aspectjrt-1.8.2.jar
 - aspectjweaver-1.8.2.jar - F:\m2\repository\org\aspectj\aspectjweaver\1.8.2\aspectjweaver-1.8.2.jar
 - junit-4.11.jar - F:\m2\repository\junit\junit\4.11\junit-4.11.jar
 - hamcrest-core-1.3.jar - F:\m2\repository\org\hamcrest\hamcrest-core\1.3\hamcrest-core-1.3.jar
- src
- target
- pom.xml

aop-config.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- Spring AOP -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>${spring-framework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aspects</artifactId>
        <version>4.3.2.RELEASE</version>
    </dependency>
    <!-- Spring and Transactions -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring-framework.version}</version>
    </dependency>
    <!-- AspectJ -->
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjrt</artifactId>
        <version>${aspectj.version}</version>
    </dependency>
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>${aspectj.version}</version>
    </dependency>
</beans>
```

Outline:

- properties
 - Generic properties
 - java.version 1.8
 - project.build.sourceEncoding UTF-8
 - project.reporting.outputEncoding UTF-8
 - Spring
 - spring-framework.version 4.3.2.RELEASE
 - AspectJ
 - aspectj.version 1.8.2
 - Test
 - junit.version 4.11
- dependencies
 - Spring AOP
 - dependency org.springframework:spring-aop:4.3.2.RELEASE
 - Spring and Transactions
 - dependency org.springframework:spring-context:4.3.2.RELEASE
 - AspectJ
 - dependency org.aspectj:aspectjrt:1.8.2
 - dependency org.aspectj:aspectjweaver:1.8.2
 - Test Artifacts
 - dependency junit:junit:4.11
- build
 - plugins
 - plugin org.apache.maven.plugins:maven-compiler-plugin:3.1

Beispiel: 34-AOP-Introduction

```
public static void main(String[] args) {  
    AbstractApplicationContext ctx =  
        new ClassPathXmlApplicationContext("aop-config.xml");  
    HelloWorldService helloWorldService =  
        (HelloWorldService)ctx.getBean("helloWorldService");  
  
    System.out.println(helloWorldService.getMessage());  
  
    FancyMessagePrinter fancyMessagePrinter =  
        (FancyMessagePrinter)helloWorldService;  
    System.out.println(fancyMessagePrinter.printMessage());  
    ctx.close();  
}
```

```
[aop-config.xml]  
Hello, World!  
doClose  
INFORMATION: Closing  
Hello from Introduction!
```

```
public class HelloWorldService {  
  
    public String getMessage() {  
        return "Hello, World!";  
    }  
}
```

Beispiel: HelloWorldIntroduction

```
@Aspect
@Component
public class HelloWorldIntroduction {

    @DeclareParents (
        value = "edu.spring.aop.introduction.HelloWorldService",
        defaultImpl = FancyMessagePrinterImpl.class
    )
    public FancyMessagePrinter fancyMessagePrinter;

}
```

```
public interface FancyMessagePrinter {

    public String printMessage();

}
```

```
public class FancyMessagePrinterImpl
    implements FancyMessagePrinter {

    @Override
    public String printMessage() {
        return "Hello from Introduction!";
    }

}
```

Beispiel: aop-config.xml

```
<?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"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    ... >
    <!-- Aktiviere AutoProxy mit proxy-target -->
    <aop:aspectj-autoproxy proxy-target-class="true"/>

    <bean id="helloWorldService"
      class="edu.spring.aop.introduction.HelloWorldService"/>

    <!-- Komponenten suchen - HelloWorldIntroduction -->
    <context:component-scan base-package="edu.spring.aop" />
</beans>
```

Zusammenfassung

3. Aspektorientierte Programmierung mit Spring

1. Was ist aspektorientierte Programmierung?
2. Grundlegende Begriffe der aspektorientierten Programmierung
3. Spring AOP und AspectJ
4. AOP mit Spring
5. Praktisches Beispiel von AOP
6. Pointcuts
7. JoinPoints
8. Introduction und die Mehrfachvererbung