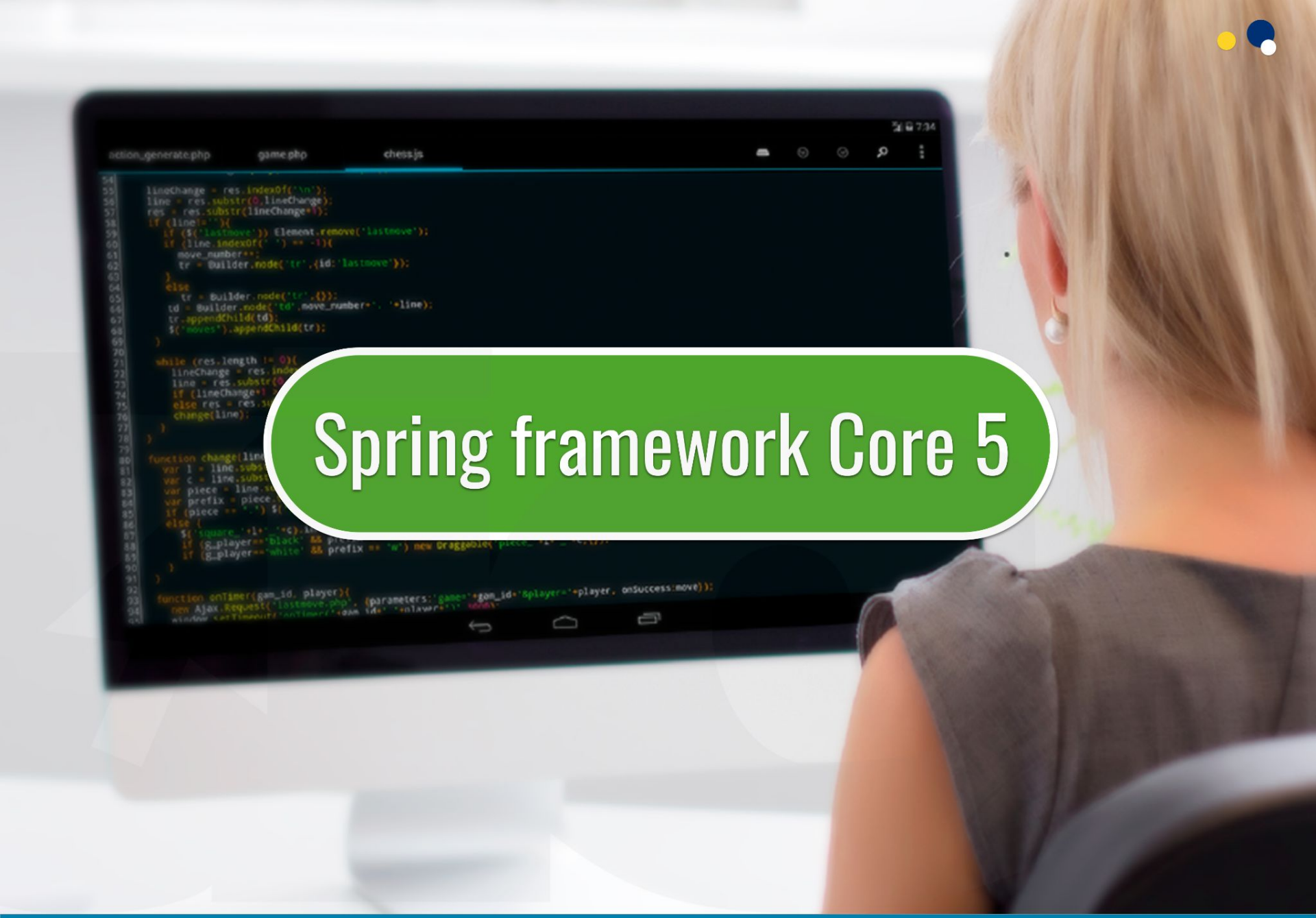

Spring framework Core 5

Agenda

1. Presentación
2. Objetivos
3. Contenido
4. Despedida

3. Contenido

i. Introducción a Spring Framework
ii. Spring Core
iii. Spring AOP
iv. Spring JDBC – Transaction
v. Spring ORM – Hibernate 5
vi. Spring Data JPA
vii. Fundamentos Spring MVC y Spring REST

iii. Spring AOP

iii. Spring AOP (a)

i. ¿Qué es AOP?

ii. POO vs AOP

iii. Spring AOP

a. Conceptos Básicos

b. Capacidades y objetivos

c. Proxies

iii. Spring AOP (b)

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d. Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML Práctica

g. Advisors configuración XML

iii. Spring AOP (c)

vi. Spring AOP con @Anotaciones

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 23. Spring AOP configuración @AspectJ

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP.

Práctica 24. Spring AOP Logging y Profiling

iii.i ¿Qué es AOP?

Objetivos de la lección

iii.i ¿Qué es AOP?

- Comprender el significado de Aspect-Oriented Programming.
- Comprender que cuestiones en el software pueden ser aspectos.
- Analizar la forma en la cual AOP puede beneficiar estructuralmente el diseño de aplicaciones de software.

iii.i ¿Qué es AOP? (a)

- ¿Qué es un Aspecto?

- Definición.

- Un aspecto es una unidad de software funcional del sistema que aparece y se entremezcla en otras unidades de software del sistema.

iii.i ¿Qué es AOP? (b)

- Aspect-Oriented Programming

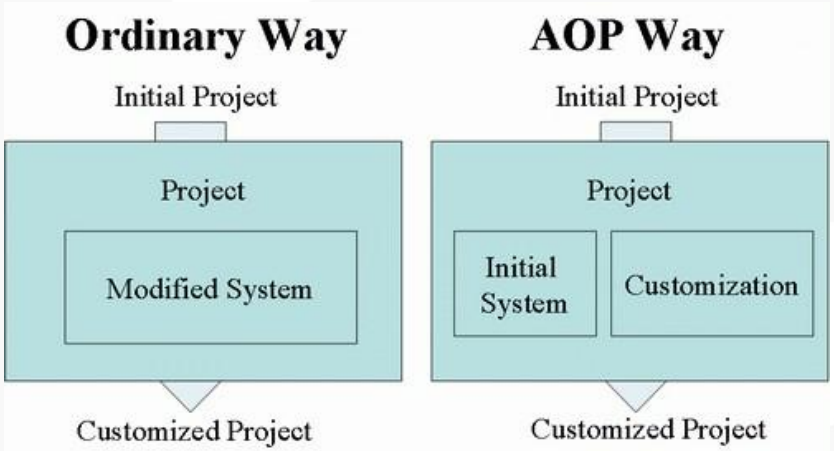
- Es un paradigma de programación que promueve separar **aspectos** o **intenciones** (concerns) de un producto software, en diferentes módulos según su responsabilidad (separation of concerns).

- Estas intenciones o incumbencias (concerns) se aplican en múltiples partes del sistema, por tanto se les considera como transversales (crosscutting concerns).

iii.i ¿Qué es AOP? (c)

- Aspect-Oriented Programming

- AOP complementa a la OOP debido a que provee una forma diferente de implementar la estructura de un software.



iii.i ¿Qué es AOP? (d)

- Un aspecto puede ser cualquier cuestión relativa a:
  - Seguridad
  - Profiling
  - Transaccionabilidad
  - Logging
  - Cache
  - entre otras cuestiones transversales del sistema.

Resumen de la lección

iii.i ¿Qué es AOP?

- Comprendimos que es Aspect-Oriented Programmig.
- Analizamos diferentes escenarios donde puede aplicarse AOP.
- Visualizamos a grandes rasgos que comportamientos de software pueden figurar como aspectos.

Esta página fue intencionalmente dejada en blanco.

iii. Spring AOP - iii.i ¿Qué es AOP?

iii.ii POO vs AOP

Objetivos de la lección

iii.ii POO vs AOP

- Comprender la implementación tradicional de aspectos transversales en el desarrollo de software.
- Comprender las diferencias entre POO y AOP.
- Comprender el objetivo de AOP.
- Analizar el beneficio de modularizar componentes por responsabilidad o preocupación.

iii.ii POO vs AOP (a)

- La POO promueven cuatro principios fundamentales.
  - Encapsulamiento
  - Abstracción
  - Herencia
  - Polimorfismo

- El componente principal de modularidad en POO es la clase.
- POO define clases y objetos del mundo real.
- POO promueve encapsulamiento.
- POO promueve los principios SOLID.
- **POO no define como los objetos se deban de relacionar entre sí.**

iii.ii POO vs AOP (b)

- AOP promueve una forma diferente de analizar los intereses, preocupaciones o lo concerniente a diversos aspectos transversales del sistema, mismos que se ejecutan, aplican o entremezclan en distintas partes del sistema.

- El componente principal de modularidad en AOP es el aspecto.
- AOP procura SOLID.
- AOP promueve modularización por intereses o preocupaciones (aspectos).
- **AOP promueve que las clases tengan una única responsabilidad.**

iii.ii POO vs AOP (c)

- Verificación de permisos sin AOP

```
public class BusinessService {
  public void doSomething() throws NoAccessAllowedException {
      verifyPermissions();
      ...
  }
  public void processData() throws NoAccessAllowedException {
      verifyPermissions();
      ...
  }
  protected void verifyPermissions() throws NoAccessAllowedException {
    ...
  }

}
```

iii.ii POO vs AOP (d)

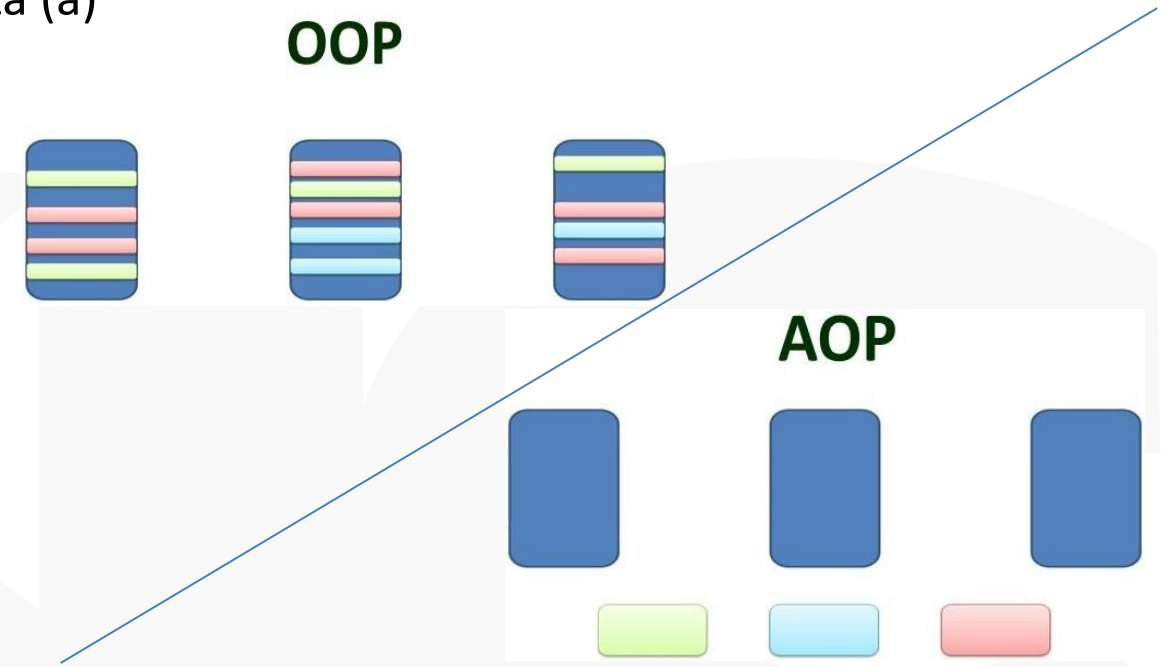
- Verificación de permisos con AOP

```
public class BusinessService {
  public void doSomething() {
      ...
  }
  public void processData() {
      ...
  }
}
```

```
public class SecurityService {
  public void verifyPermissions() throws
                        NoAccessAllowedException {
    ...
  }
}
```

iii.ii POO vs AOP (e)

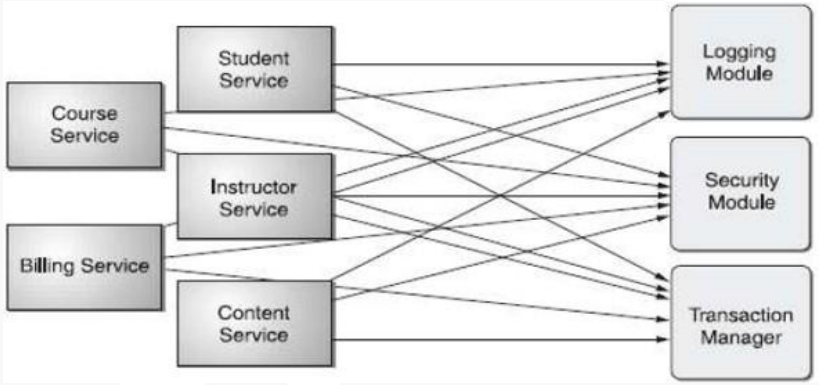
- Analiza (a)



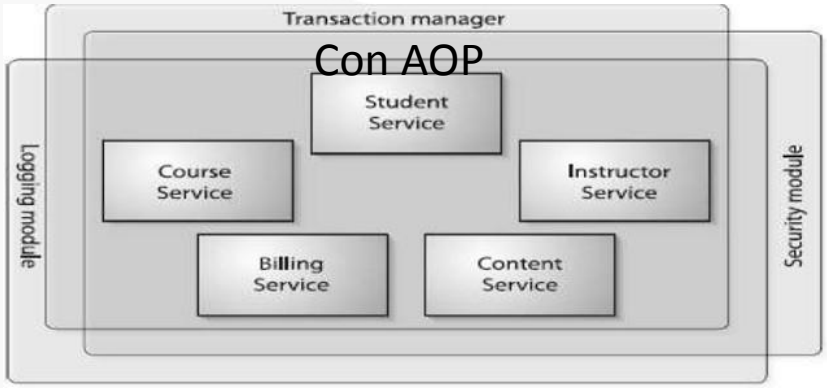
iii.ii POO vs AOP (f)

- Analiza (b)

Sin AOP

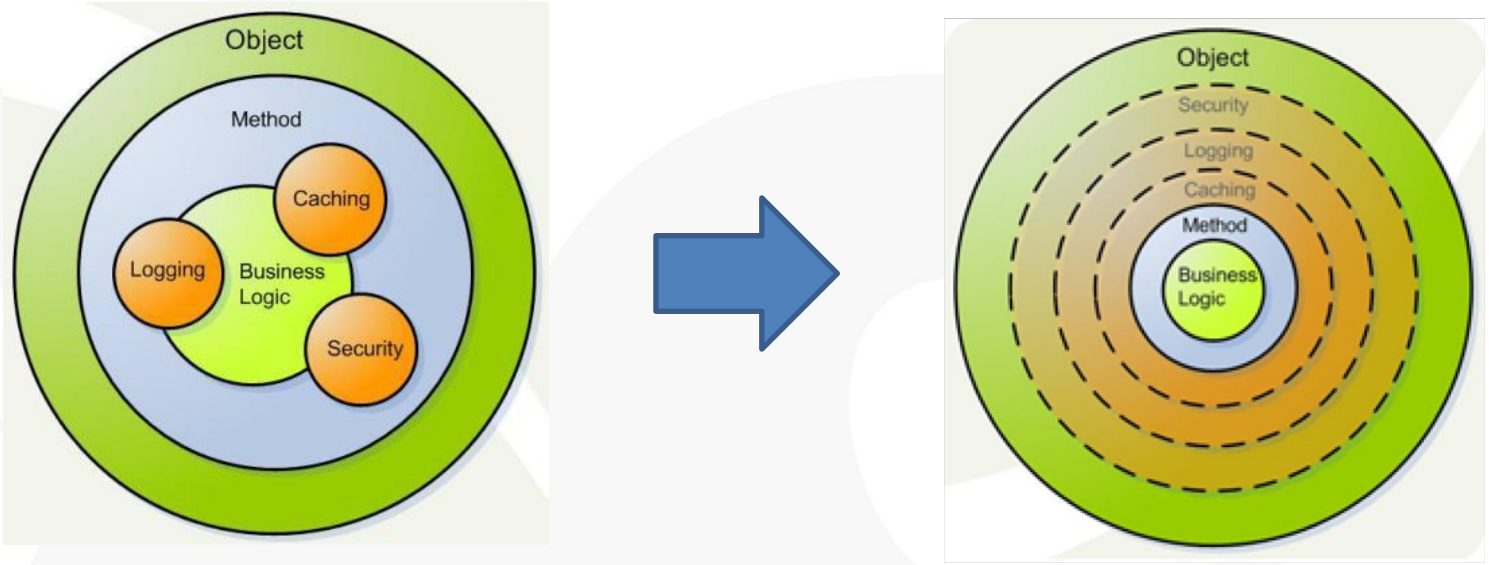


Con AOP



iii.ii POO vs AOP (g)

- Analiza (c)



Resumen de la lección

iii.ii POO vs AOP

- Comprendimos la implicación de implementar aspectos transversales mediante POO tradicional.
- Analizamos la diferencia entre implementar aspectos transversales mediante POO y AOP.
- Comprendimos el objetivo de AOP.
- Analizamos que la modularidad de AOP beneficia análisis, diseño, pruebas, construcción y desacopla módulos verticales vs aspectos transversales.

Esta página fue intencionalmente dejada en blanco.

iii. Spring AOP - iii.ii POO vs AOP

iii.iii Spring AOP

Objetivos de la lección

iii.iii Spring AOP

- Comprender la terminología utilizada en AOP.
- Comprender como Spring AOP implementa aspectos.
- Conocer los diferentes tipos de Advice.
- Conocer los capacidades y objetivos de Spring AOP.
- Comprender el funcionamiento de proxies basados por JDK dynamic proxies y CGLIB proxies.

iii. Spring AOP

a. Conceptos Básicos

b. Capacidades y objetivos

c. Proxies

iii.iii Spring AOP (a)

- El módulo Spring AOP es uno de los componentes clave de Spring Framework, mediante aspectos Spring implementa mucha funcionalidad que provee como servicios empresariales (especialmente como reemplazo de servicios declarativos EJB) por ejemplo **seguridad** y **transaccionabilidad**.

- El contenedor de IoC de Spring no depende de Spring AOP, Spring AOP si depende del contenedor de IoC de Spring.

- Spring AOP provee un API realmente poderosa para implementación de aspectos como soluciones middleware.

iii. Spring AOP - iii.iii Spring AOP

iii.iii Spring AOP (b)

- Spring AOP se configura mediante:
  - Basado por configuración XML
  - Basado por anotaciones @AspectJ

- @AspectJ es un framework Java para la implementación de aspectos.

- Spring no reinventa la rueda, reutiliza.

- Spring AOP utiliza proxies para tejer o entrelazar aspectos.

- Es posible utilizar tejido de aspectos en tiempo de carga (no cubierto en este curso).

iii. Spring AOP

a. Conceptos Básicos

b. Capacidades y objetivos

c. Proxies

iii.iii Spring AOP (a)

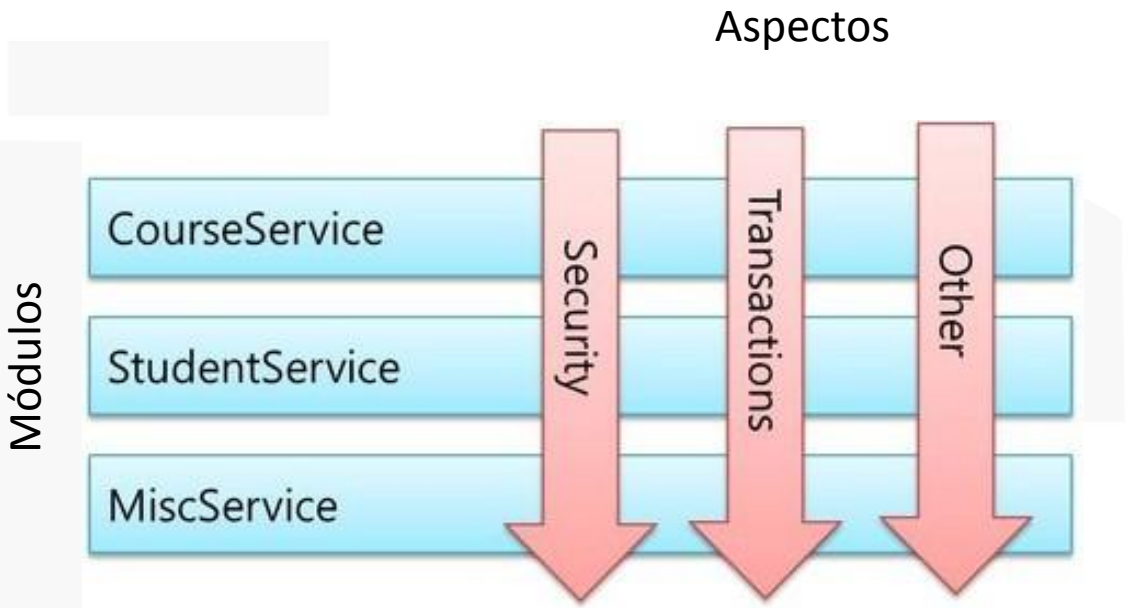
- Conceptos básicos: Terminología.

- Desafortunadamente la terminología utilizada para AOP no es particularmente intuitiva.

- Spring no define su propia terminología referente AOP, debido a que comprender AOP sería más confuso.

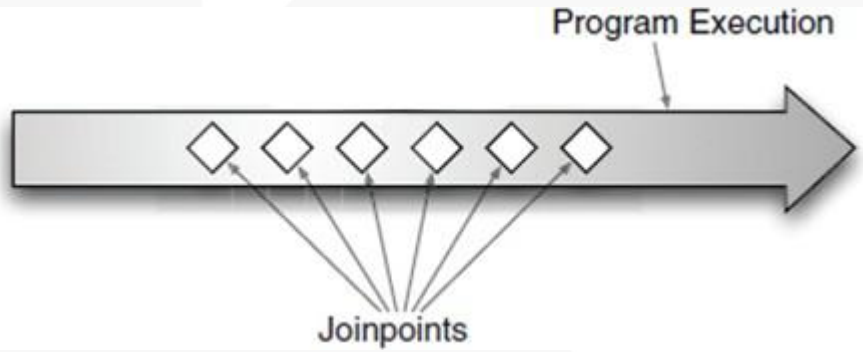
iii.iii Spring AOP (b)

- **Aspect (Aspecto)**: Es la modularización de un interés o una preocupación que se aplica en múltiples clases o módulos del sistema.



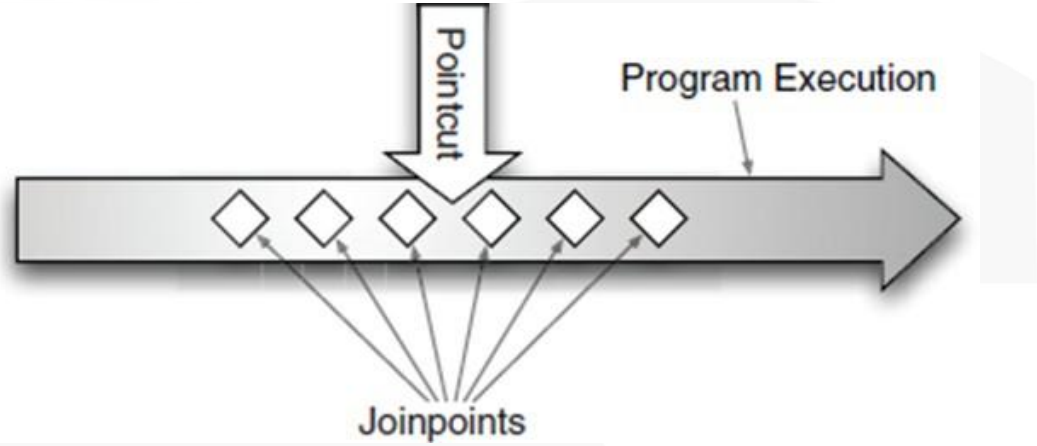
iii.iii Spring AOP (c)

- **Join Point (Punto de unión)**: Representa un punto específico de la aplicación en el tiempo durante su ejecución.
- Es el lugar donde un aspecto se puede ejecutar.
- En Spring AOP un Join Point siempre será representado por la ejecución de un método.



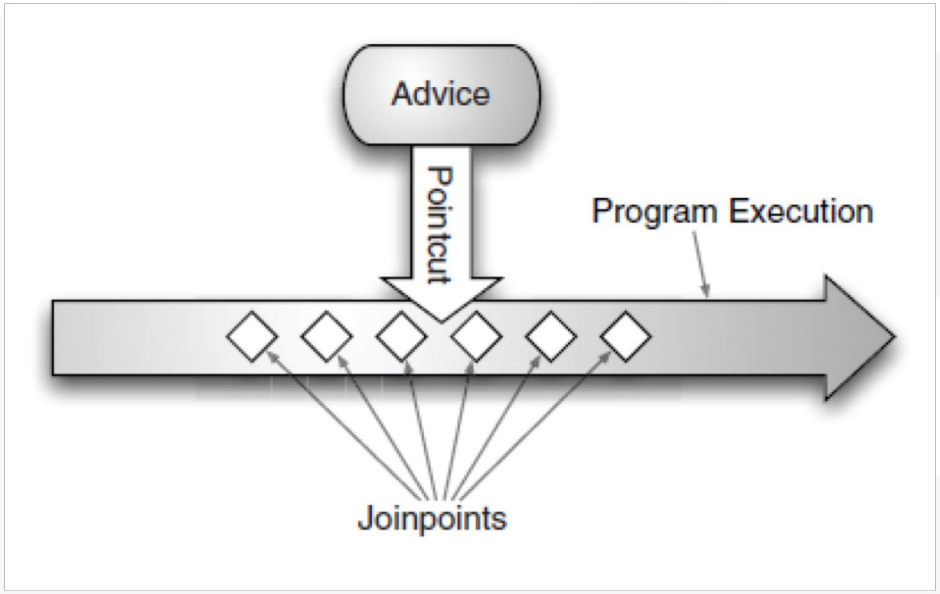
iii.iii Spring AOP (d)

- **Pointcut (Punto de corte)**: Es un predicado o expresión que "***matchea***" a un conjunto de Join Points.
- Se utiliza la sintaxis de AspectJ para definir pointcuts.



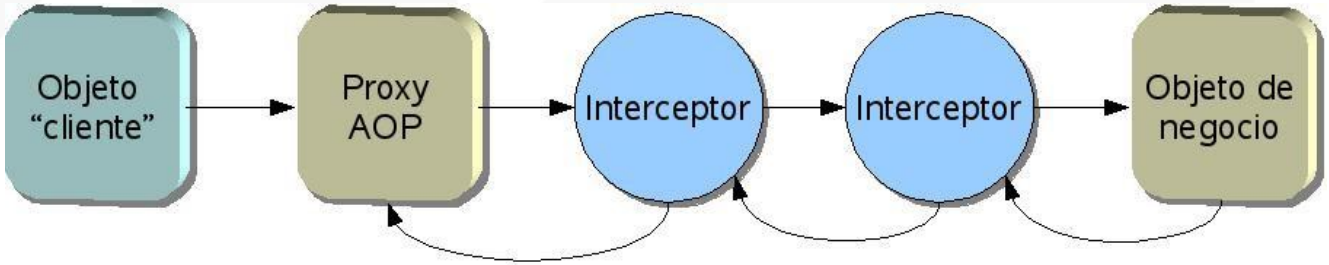
iii.iii Spring AOP (e)

- **Advice (Consejo/Aviso) (a)**: Es la acción que un aspecto debe ejecutar en un Join Point específico.
- Un advice se asocia a uno o más pointcut para que pueda ejecutarse en cualquier Join Point definido.



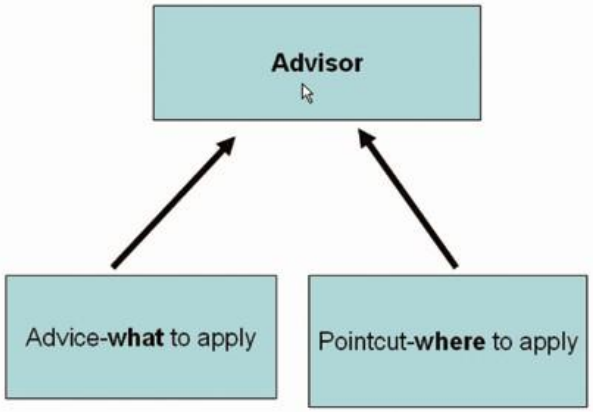
iii.iii Spring AOP (f)

- **Advice (Consejo/Aviso) (b)**: Un advice se modela particularmente como una cadena de interceptores los cuales se ejecutaran alrededor del Join Point.
- Existen diversos tipos de advices (around, before y after).



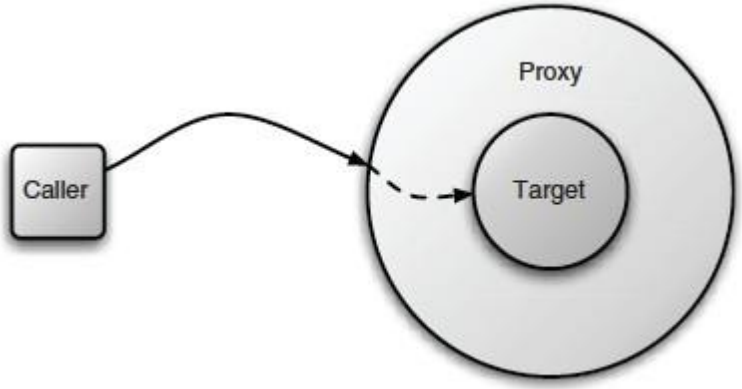
iii.iii Spring AOP (g)

- **Advisor**: Un Advisor es un aspecto ”auto-contenido” que implementa uno o más tipos de advice.
- Define la ejecución de uno o varios advice en un pointcut determinado bajo una misma definición.
- No tiene un equivalente en @AspectJ.



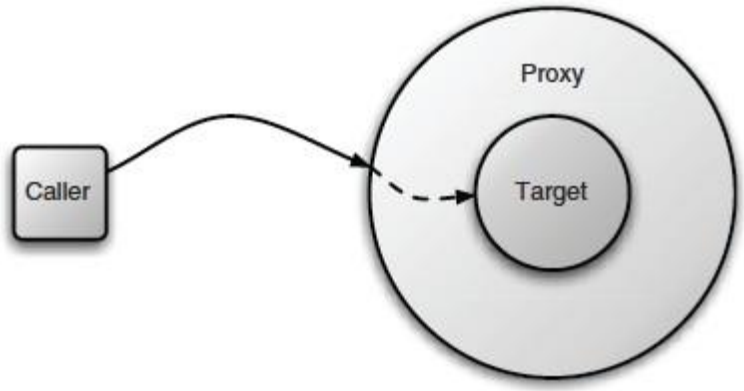
iii.iii Spring AOP (h)

- **AOP Proxy (Proxy AOP)**: Un proxy es un objeto intermedio entre un objeto *caller* y un objeto *worker.*
- En Spring AOP, un proxy AOP es un objeto creado con la finalidad de implementar aspectos.
- Un proxy AOP es un objeto que encapsula el target object o advised object.



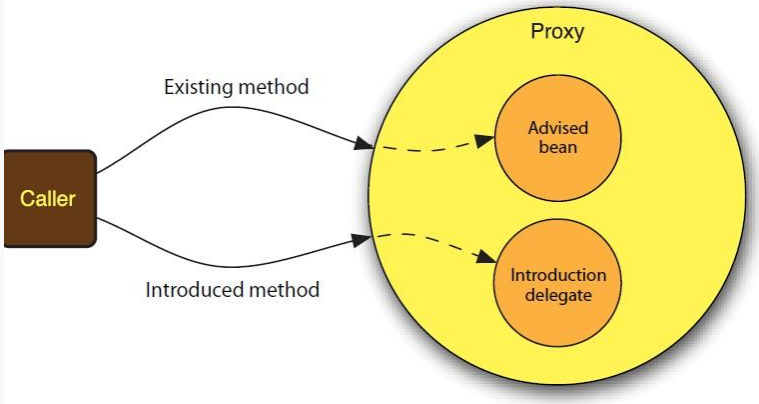
iii.iii Spring AOP (i)

- **Target object/Advised object (Objeto aconsejado)**: Es el objeto que será aconsejado (advised) por uno o más aspectos.
- Spring AOP implementa advised objects a partir de proxies creados en tiempo de ejecución, por tanto un target object o advised object será siempre un objeto envuelto por un *proxy* (proxied object).



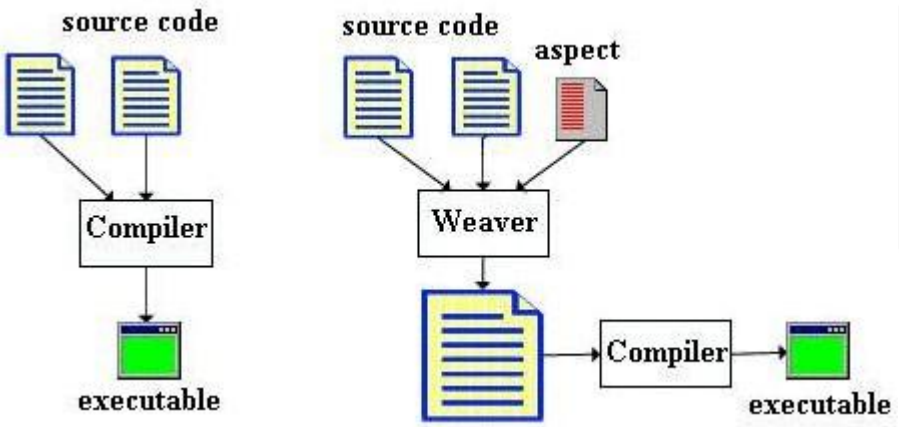
iii.iii Spring AOP (j)

- **Introduction (Introducción)**: Es el hecho de realizar declaraciones adicionales de métodos o atributos en un objeto determinado.
- Spring AOP permite introducir nuevas interfaces, con su correspondiente implementación a un target object o advised object (objeto aconsejado).



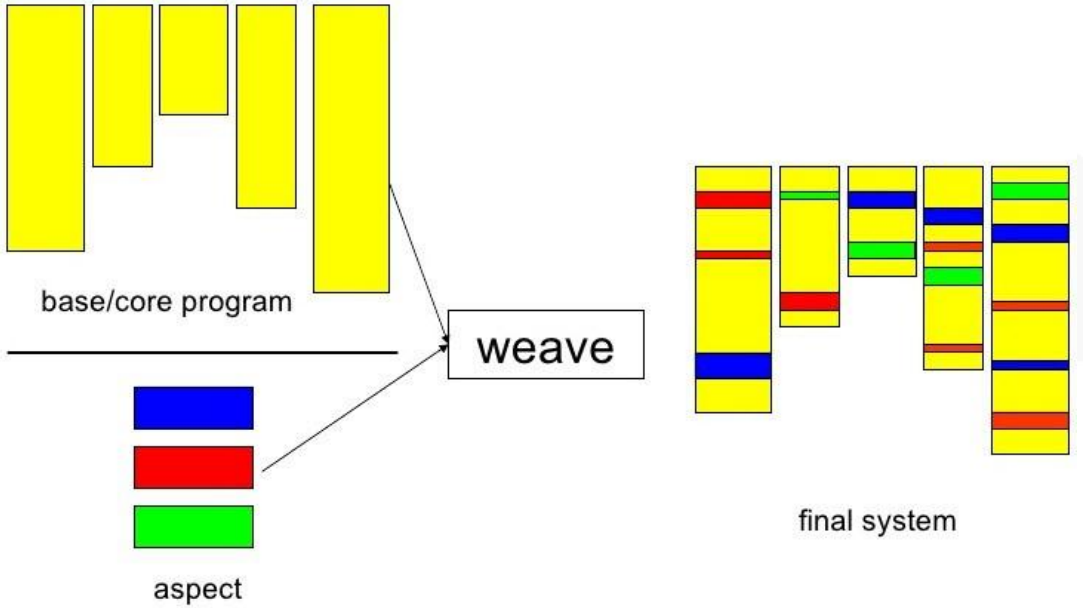
iii.iii Spring AOP (k)

- **Weaving (Tejido)**: Es el proceso de vincular aspectos con los puntos de corte de la aplicación (Pointcuts) para que éstos puedan ejecutar sus métodos advices donde haya sido especificado.
- El weaving crea los objetos aconsejados (adviced objects).



iii.iii Spring AOP (I)

- **Weaving (Tejido)**



iii.iii Spring AOP (m)

- Conceptos básicos: Tipos de Advice.

- Existen diferentes tipos de Advice
  - Before advice
  - After Returning advice
  - After Throwing advice
  - After advice (finally)
  - Around advice

iii.iii Spring AOP (n)

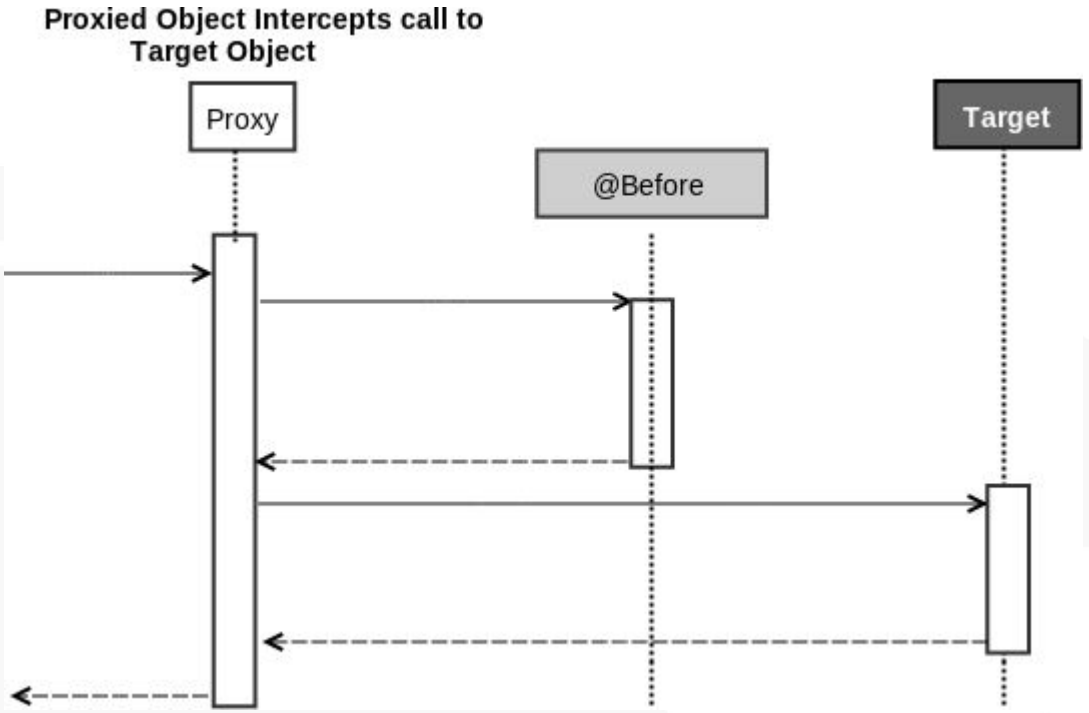
- **Before advice**: Este advice (consejo) se ejecuta antes de la ejecución de un Join Point, no tiene la capacidad de evitar que se ejecute el flujo de llamadas hacia el target object a menos que lance una excepción.

- Su interface principal es MethodBeforeAdvice.

```
public interface MethodBeforeAdvice extends BeforeAdvice {
    void before(Method m, Object[] args, Object target) throws Throwable;
}
```

iii.iii Spring AOP

(ñ)



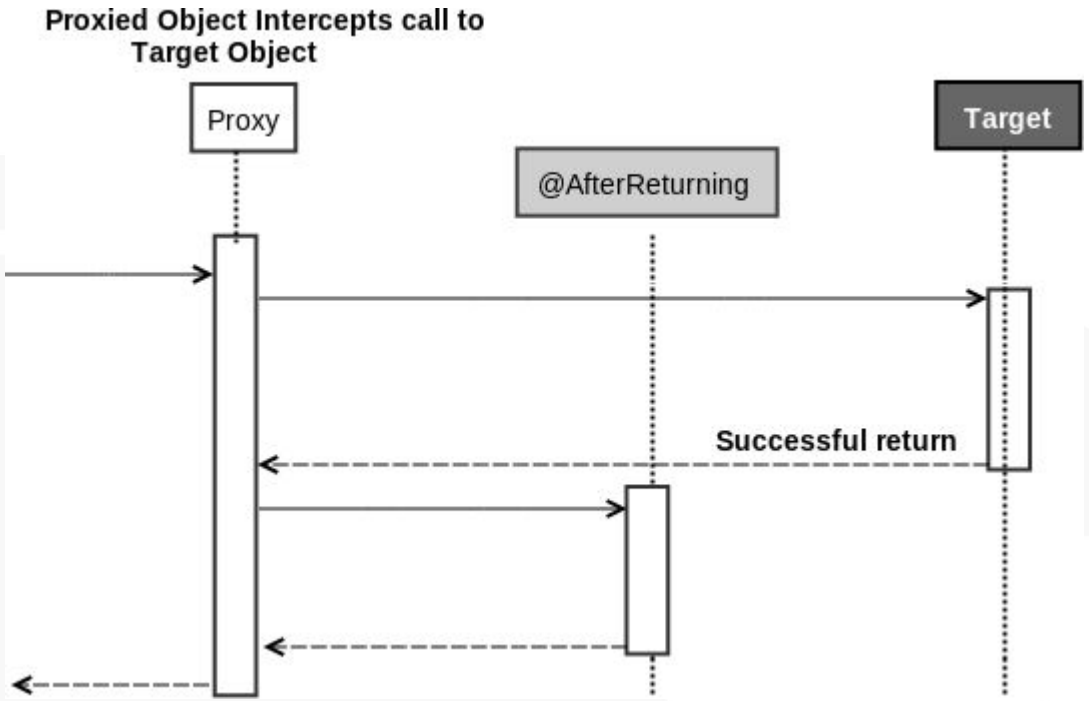
iii.iii Spring AOP (o)

- **After Returning advice**: Este advice (consejo) se ejecuta después de la ejecución de un Join Point, sólo si éste se completó normalmente (sin lanzar excepción).

- Su interface principal es AfterReturningAdvice.

```
public interface AfterReturningAdvice extends AfterAdvice {
    void afterReturning(Object returnValue, Method m, Object[] args,
                                                    Object target) throws Throwable;
}
```

iii.iii Spring AOP (p)



iii.iii Spring AOP (q)

- **After Throwing advice (a)**: Este advice (consejo) se ejecuta después de la ejecución de un Join Point, sólo si éste lanzó una excepción.

- Su interface principal es ThrowsAdvice.

```
public interface ThrowsAdvice extends AfterAdvice {

}
```

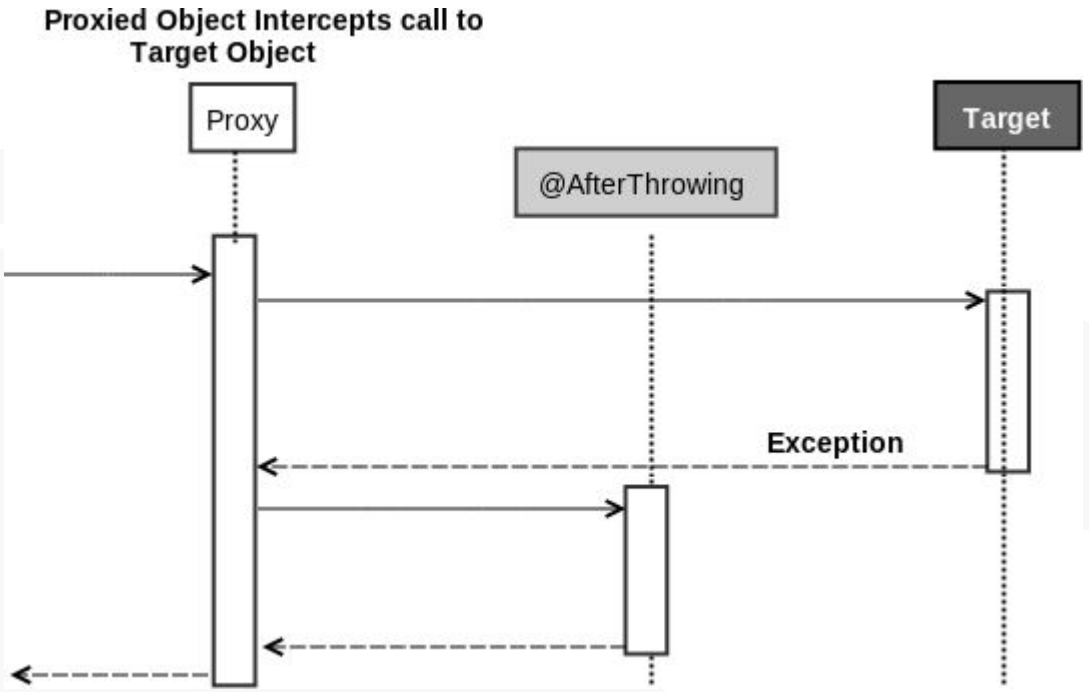
iii.iii Spring AOP (r)

- **After Throwing advice (b)**: Las clases que requieran implementar ThrowsAdvice deberán hacerlo de la siguiente forma:

  afterThrowing(*[Method, args, target]*, **subclassOfThrowable**)

```
public class RemoteThrowsAdvice implements ThrowsAdvice {
    public void afterThrowing(RemoteException ex) throws Throwable {
        // Do something with remote exception
    }
}

public class ServletThrowsAdviceWithArguments implements ThrowsAdvice {
    public void afterThrowing(Method m, Object[] args, Object target,
                              ServletException ex) {
        // Do something with all arguments

    }
}
```

iii.iii Spring AOP (s)



iii.iii Spring AOP (t)

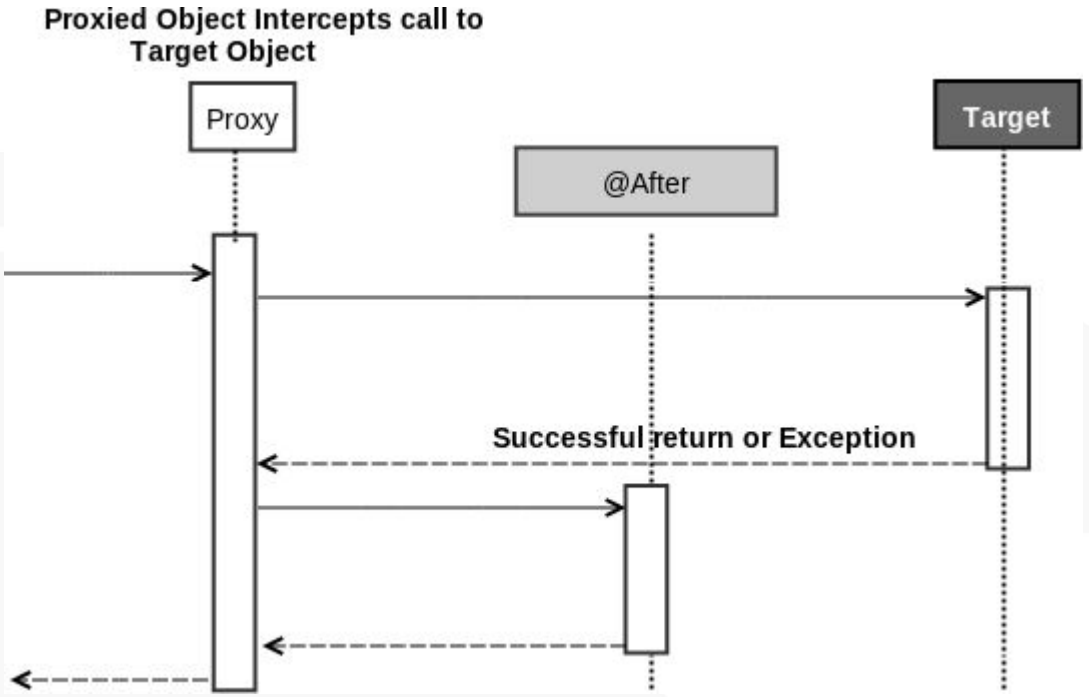
- **After advice (finally)**: Este advice (consejo) se ejecuta después de la ejecución de un Join Point, ya sea si concluyó normalmente o si lanzó una excepción.

- No define una interface principal. AfterAdvice únicamente debe ser implementada a través de ThrowsAdvice o AfterReturningAdvice.

- Sólo es una "**marker**" interface.

```
public interface AfterAdvice extends Advice {
}
```

iii.iii Spring AOP (u)



iii.iii Spring AOP (v)

- **Around advice**: Este advice (consejo) envuelve la ejecución de un Join Point como si fuera una invocación a un método cualquiera.
  Esto significa que éste advice, se encarga de invocar programáticamente la invocación al método del target object.

- El Around advice es responsable de proseguir con la invocación al Join Point o, en su defecto, anularlo mediante un corto circuito devolviendo un valor especifico (sin llamar al target object) o lanzando una excepción.

iii.iii Spring AOP (w)

- **Around advice (a)**: Su interface principal es MethodInterceptor:

```
public interface MethodInterceptor extends Interceptor {
    Object invoke(MethodInvocation invocation) throws Throwable;
}
```

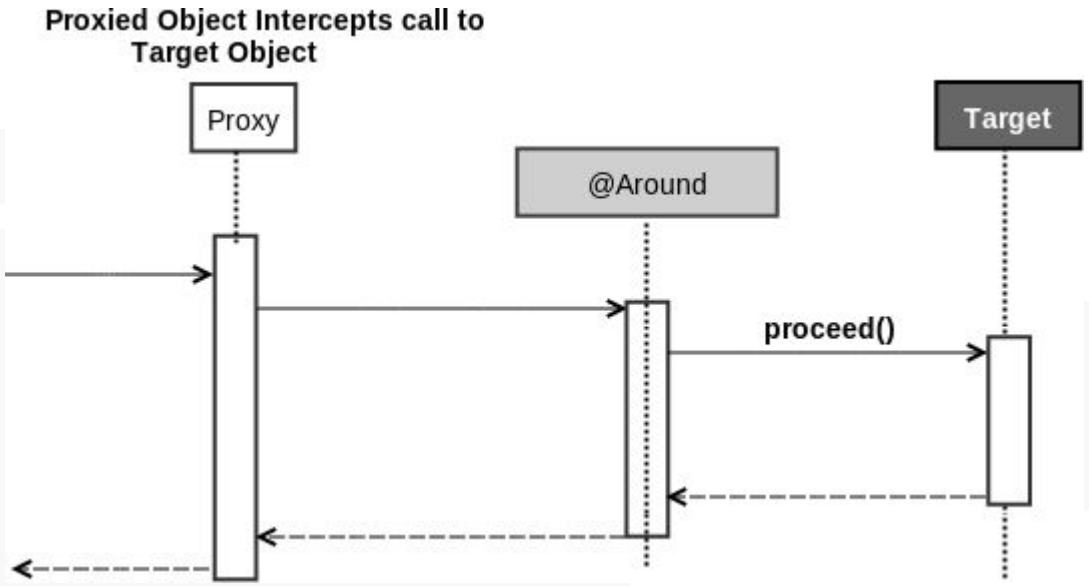
- El objeto MethodInvocation que recibe como argumento contiene el método proceed() para poder ejecutar el “target-object”.

iii.iii Spring AOP (x)

- **Around advice (b)**: Una implementación MethodInterceptor podría ser:

```
public class DebugInterceptor implements MethodInterceptor {
    public Object invoke(MethodInvocation invocation) throws Throwable {

        System.out.println("Before: invocation=[" + invocation + "]");
        Object rval = invocation.proceed();

        System.out.println("Invocation returned");

        return rval;

    }
}
```

iii.iii Spring AOP (y)



iii. Spring AOP

a. Conceptos Básicos

b. Capacidades y objetivos

c. Proxies

iii.iii Spring AOP (a)

- Capacidades y objetivos.

- Spring AOP esta totalmente implementado en Java, no requiere compilación especial para implementar aspectos.

- Spring AOP no requiere control sobre class loaders y puede funcionar con o sin servidores de aplicaciones, sólo necesita JVM.

- Spring AOP construye proxies para implementar aspectos, así como para agregar servicios empresariales a problemas reales en aplicaciones Java EE.

iii.iii Spring AOP (b)

- Spring AOP soporta solamente intercepción de ejecución de métodos. Para utilizar intercepción sobre propiedades se recomienda utilizar otro framework como AspectJ.

- El objetivo de Spring AOP no es proveer una implementación completa de AOP, sino facilitar la implementación de Aspectos.

- El objetivo de Spring AOP es proveer servicios de modularidad mediante aspectos al contenedor IoC de Spring que permita solucionar problemas comunes típicos en aplicaciones empresariales tal como transaccionabilidad, seguridad, logging, entre otros.

iii. Spring AOP - iii.iii Spring AOP

iii. Spring AOP

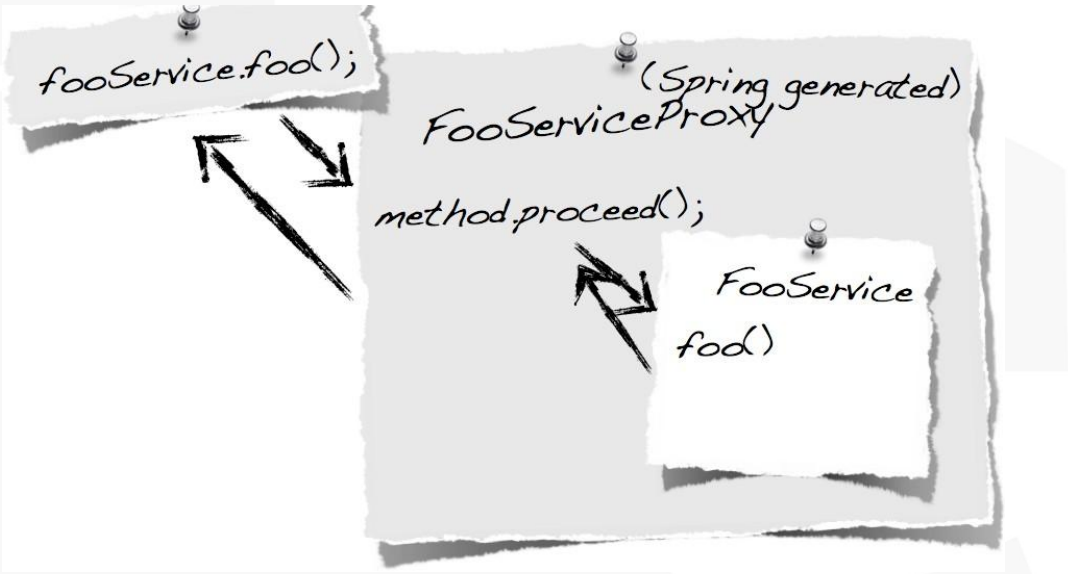
a. Conceptos Básicos

b. Capacidades y objetivos

c. Proxies

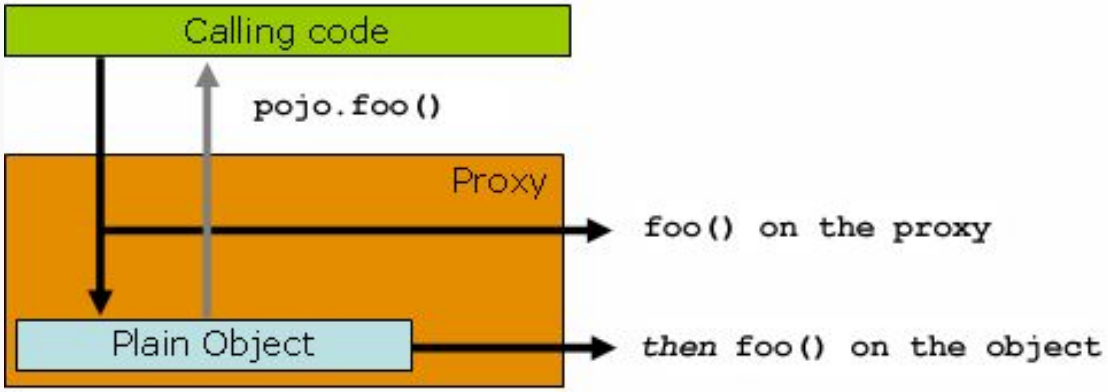
iii.iii Spring AOP (a)

- Spring AOP utiliza proxies para implementar aspectos.



iii.iii Spring AOP (b)

- Mediante la implementación de proxies, es posible ejecutar funcionalidad, antes, después, o durante la ejecución del método en el target object.



iii.iii Spring AOP (c)

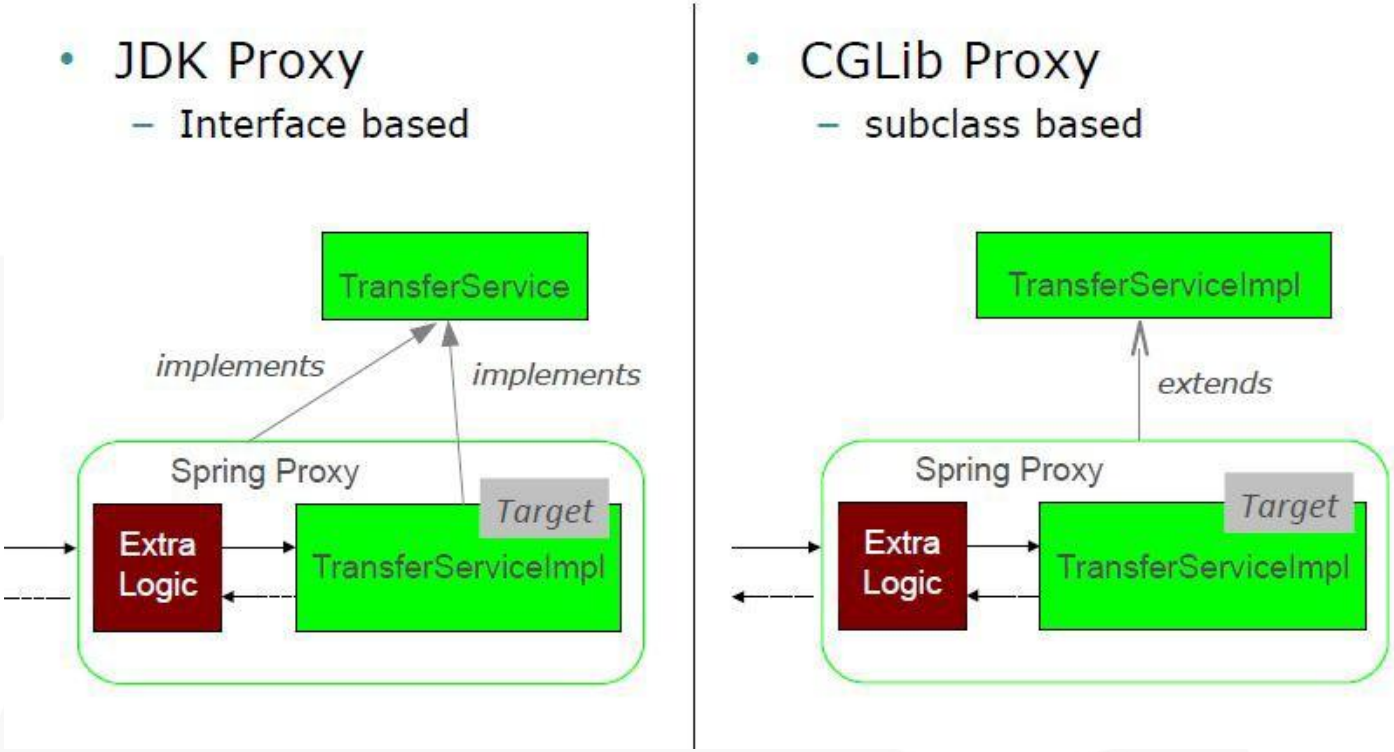
- Spring AOP por defecto utiliza el estándar JDK dynamic proxies, ésta aproximación permite habilitar a cualquier interface ser *proxeada.*

- Spring AOP también puede utilizar CGLIB proxies.

- Implementar CGLIB proxies es necesario para *proxear* clases que no implementan una interface. Por default se usa CGLIB para *proxear* objetos que no implementan una interface.

- La generación de proxies mediante CGLIB debe ser sólo sobre clases no finales.

iii.iii Spring AOP (d)



Resumen de la lección

iii. Spring AOP - iii.iii Spring AOP

- Aprendimos la terminología general de AOP.
- Comprendimos que es un aspecto.
- Conocimos los diferentes tipos de advice.
- Analizamos la forma que implementa Spring AOP para crear aspectos mediante proxies.
- Revisamos las limitaciones de Spring AOP.
- Comprendimos la diferencia entre aspectos JDK dynamic proxies y CGLIB proxies.

Esta página fue intencionalmente dejada en blanco.

iii.v Spring AOP con XML

Objetivos de la lección

iii.v Spring AOP con XML

- Conocer el API Spring AOP para configurar aspectos.
- Comprender como se implementan aspectos por medio de configuración XML.
- Comprender como se implementan los distintos tipos de advice por medio de configuración XML.
- Comprender como se pasan parámetros a los métodos advice mediante configuración XML.

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d. Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML (a)

- Dependencias

- Spring AOP no se incluye por defecto en Spring core.

- Para habilitar Spring AOP es necesario agregar la dependencia:

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

iii.vSpring AOP con XML (b)

- Dependencias

- Para habilitar el soporte de AspectJ para definir aspectos:

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

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

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d.Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML (a)

- Configuración de aspectos

- Habilitar el namespace *aop*.

```
<?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-4.2.xsd">
</beans>
```

iii.vSpring AOP con XML (b)

- Toda configuración de Spring AOP por medio de XML requiere encapsular los aspectos en <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: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-4.2.xsd">
        <aop:config>
                <!-- Aspects configuration -->
        </aop:config>

</beans>
```

iii.vSpring AOP con XML (c)

- Definición de un aspecto

- El tag <aop:aspect> se utiliza para definir un aspecto, el cual debe ser referenciado por un bean de Spring

```
<aop:config>
    <aop:aspect id="componentAspect" ref="componentAspectBean">
        ...
    </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>
```

iii.vSpring AOP con XML (d)

- Definición de un pointcut

- Es posible definir un pointcut a nivel global dentro de <aop:config> o definirlo de manera única para un aspecto especifico dentro de <aop:aspect>

- Global

```
<aop:config>
    <aop:pointcut id="businessService"
                  expression="execution(* com.xyz.myapp.service.*.*(..))"/>
</aop:config>
```

iii.vSpring AOP con XML (f)

- Expresiones de definición de pointcut

- **execution**: Define la firma de ejecución de métodos Join Point. Este es el *designador* primario que se utiliza para trabajar con Spring AOP.

- **within**: Limita hacer *match* con Join Points que estén definidos dentro del tipo (clase) especificado.

- **this**: Limita hacer *match* con Join Points donde la referencia del bean, el aop proxy, es una instancia del tipo (clase) especificado.

iii.vSpring AOP con XML (g)

- **target**: Limita hacer *match* con Join Points donde la referencia del target object, es una instancia del tipo (clase) especificado.

- **args**: Limita hacer *match* con Join Points donde los argumentos del método son instancias de los tipos (clases) especificados.

- **@target**: Limita hacer *match* con Join Points donde la clase del target object tiene una @Anotacion del tipo (interface) especificado.

iii.vSpring AOP con XML (h)

- **@args**: Limita hacer *match* con Join Points donde los argumentos del método en tiempo de ejecución tienen una @Anotacion del tipo (interface) especificado.

- **@within**: Limita hacer *match* con Join Points que estén definidos dentro de algún tipo (clase) con la @Anotación del tipo (interface) especificado.

- **@annotation**: Limita hacer *match* con Join Points donde el método tenga una @Anotación del tipo (interface) especificado.

iii.vSpring AOP con XML (i)

- **bean**: Limita hacer *match* con Join Points que sean referenciados por un nombre de bean particular.

- Combinando expresiones

- Es posible utilizar los operadores lógicos (&&, || y !) para combinar expresiones en un único pointcut.

- El operador * indica *cualquiera* y el operador .. indica *cualquiera cuantos quiera.*

iii.vSpring AOP con XML (j)

- Formato de expresiones pointcut

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

- Ejemplo:

execution(public void com.xyz.app.Service.operation(String, ..))

execution(public void com.xyz.app.Service.operation(String, *))

execution(void com.xyz.app.Service.method(com.xyz.model.Account, ..)) &&
args(account,..)

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d.Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML (a)

- Configuración de advices: Before advice, con referencia a pointcut.

```
<aop:config>
   <aop:pointcut id="businessServiceExecution"
                  expression="execution(* com.xyz.myapp.service.*.*(..))"/>

   <aop:aspect id="componentAspect" ref="componentAspectBean">
      <aop:before pointcut-ref=" businessServiceExecution "
                                                  method="doAccessCheck"/>

   </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>
```

iii.vSpring AOP con XML (b)

- Configuración de advices: Before advice, definiendo pointcut en línea.

```
<aop:config>
    <aop:aspect id="componentAspect" ref="componentAspectBean">
        <aop:before pointcut="execution(* com.xyz.myapp.service.*.*(..))"
                                                    method="doAccessCheck"/>
    </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>
```

iii.vSpring AOP con XML (c)

- Configuración de advices: After Returning advice, sin returning value.

```
<aop:config>
   <aop:pointcut id="businessServiceExecution"
                 expression="execution(* com.xyz.myapp.service.*.*(..))"/>

   <aop:aspect id="componentAspect" ref="componentAspectBean">
      <aop:after-returning pointcut-ref=" businessServiceExecution "
                           method="doAccessCheck"/>
   </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>
```

iii.vSpring AOP con XML (d)

- Configuración de advices: After Returning advice, con returning value.

```
<aop:config>
    <aop:aspect id="componentAspect" ref="componentAspectBean">
        <aop:after-returning pointcut="execution(* com.xyz.myapp.service.*.*(..))"
                             returning="returnValue" method="doAccessCheck"/>
    </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>

public class ComponentAspect {

    public void doAccessCheck(Object returnValue) { ... }
}
```

iii.vSpring AOP con XML (e)

- Configuración de advices: After Throwing advice, sin throwing exception.

```
<aop:config>
   <aop:pointcut id="businessServiceExecution"
                 expression="execution(* com.xyz.myapp.service.*.*(..))"/>

   <aop:aspect id="componentAspect" ref="componentAspectBean">
      <aop:after-throwing pointcut-ref=" businessServiceExecution "
                          method="doRecoveryActions"/>
   </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>
```

iii.vSpring AOP con XML (f)

- Configuración de advices: After Throwing advice, con throwing exception.

```
<aop:config>
    <aop:aspect id="componentAspect" ref="componentAspectBean">
        <aop:after-throwing pointcut="execution(* com.xyz.myapp.service.*.*(..))"
                            throwing="dataAccessException" method="doRecoveryActions"/>
    </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>

public class ComponentAspect {

    public void doRecoveryActions(DataAccessException dataAccessException) { ... }
}
```

iii.vSpring AOP con XML (g)

- Configuración de advices: After advice (finally), con referencia a pointcut.

```
<aop:config>
   <aop:pointcut id="businessServiceExecution"
                 expression="execution(* com.xyz.myapp.service.*.*(..))"/>

   <aop:aspect id="componentAspect" ref="componentAspectBean">
      <aop:after pointcut-ref=" businessServiceExecution "
                                                  method="doAccessCheck"/>
   </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>
```

iii.vSpring AOP con XML (h)

- Configuración de advices: After advice (finally), definiendo pointcut en línea.

```
<aop:config>
    <aop:aspect id="componentAspect" ref="componentAspectBean">
        <aop:after pointcut="execution(* com.xyz.myapp.service.*.*(..))"
                                                    method="doAccessCheck"/>
    </aop:aspect>
</aop:config>

<bean id="componentAspectBean" class="ComponentAspect"/>
```

iii.vSpring AOP con XML (i)

- Configuración de advices: Around advice
- El Around advice se ejecuta alrededor de la ejecución del método del target object.
- Permite trabajar con todos los diferentes advices de forma programática.
- Para definir un Around advice mediante configuración XML se usa el tag <aop:around>

iii.vSpring AOP con XML (j)

- Configuración de advices: Around advice

- El método around advice a ejecutar recibe como primer parámetro un objeto del tipo **ProceedingJoinPoint**.

- Estando en el interior del método around advice se invoca al método **proceed()** del objeto **ProceedingJoinPoint** para proceder con la ejecución del método en el target object.

- El método **proceed()** puede recibir un **Object[]** indicando los argumentos a enviar a la ejecución del método en el target object.

iii.vSpring AOP con XML (k)

- Configuración de advices: Around advice.

```
<aop:config>
    <aop:aspect id="componentAspect" ref="componentAspectBean">
        <aop:around pointcut="execution(* com.xyz.myapp.service.*.*(..))"
                                                    method="doProfiling"/>
    </aop:aspect>
</aop:config>
<bean id="componentAspectBean" class="ComponentAspect"/>


public class ComponentAspect {


    public Object doProfiling(ProceedingJoinPoint pjp) throws Throwable {
        return pjp.proceed();
    }
```

iii.vSpring AOP con XML (I)

- Configuración de advices: Acceso al Join Point actual

- Cualquier método advice (excepto Around) puede declarar como primer parámetro un objeto del tipo **org.aspectj.lang.JoinPoint.**

- **JoinPoint**: Provee de numerosos métodos útiles como:
  - getArgs(): Retorna los argumentos del método.
  - getThis(): Retorna el proxy object.
  - getTarget(): Retorna el target object.
  - getSignature(): retorna la firma del método aconsejado (advised méthod).

iii.vSpring AOP con XML (m)

- Configuración de advices: Pasando parámetros al advice

- Para pasar parámetros al método advice es necesario usar el *designador* **args**.

- El *designador* **args** requiere de una expresión donde se le pase el nombre de los parámetros de los argumentos requeridos. El nombre del parámetro, así como el orden asignado en **args** debe *matchear* con la firma del método advice.
  En caso de que el nombre del parámetro no haga *match* especificar el nombre de o los atributos en el atributo **arg-names** separados por

  comma.

iii.vSpring AOP con XML (n)

- Configuración de advices: Pasando parámetros al advice

- Dado la siguiente clase target object:

```
package

org.certificatic.spring.aop.practica24.bank.service.account.api.impl;

@Service

public class AccountService implements IAccountService {
    public void updateAccountBalance(Account account, Long amount) {
        …
    }
}
```

iii.vSpring AOP con XML (o)

- Configuración de advices: Pasando parámetros al advice (a)

```
<aop:aspect ref="accountLoggingAspect">
    <aop:before
        pointcut="within(org.certificatic.spring.aop.practica24.bank.service..*) and
                        args(account, ..)" method="beforeAccountMethodExecution_" />
</aop:aspect>

@Component("accountLoggingAspect")
public class AccountLoggingAspect {
    public void beforeAccountMethodExecution_(Account account) {
        …
    }
}
```

iii.vSpring AOP con XML (p)

- Configuración de advices: Pasando parámetros al advice (b)

```
<aop:pointcut id="serviceLayerAndArgsAccount“
    expression="within(org.certificatic.spring.aop.practica24.bank.service..*) and args(yy, ..)" />
<aop:aspect ref="accountLoggingAspect">
   <aop:before pointcut-ref="serviceLayerAndArgsAccount“
                                                    method="beforeAccountMethodExecution" />
</aop:aspect>

@Component("accountLoggingAspect")
public class AccountLoggingAspect {
   public void beforeAccountMethodExecution(Account yy) {
      …
   }
}
```

iii.vSpring AOP con XML (q)

- Configuración de advices: Pasando parámetros al advice (c)

```
<aop:aspect ref="accountLoggingAspect">
    <aop:before pointcut="within(org.certificatic.spring.aop.practica24.bank.service..*)
                          and args(cuenta, monto, ..)“
                                                  method="beforeAccountMethodExecution2"
                                                  arg-names="monto, cuenta" />
</aop:aspect>

@Component("accountLoggingAspect")
public class AccountLoggingAspect {
    public void beforeAccountMethodExecution2(JoinPoint jp, Long ammount, Account acc) {
        …
    }
}
```

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d. Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML. Práctica 22. (a)

- Práctica 22. Spring AOP configuración XML.

- Implementar aspectos con Spring AOP mediante configuración XML.
- Implementar los cinco tipos de advice mediante configuración XML.

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d.Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML (a)

- Introductions configuración XML

- Los “**introductions**” permiten a los aspectos poder implementar, sobre los “**target-objects**”, una nueva interfaz y proveer una implementación de dicha interfaz en nombre dicho “**target-object**”.

- Los “**introductions**” basados en configuración XML permiten que la implementación de dicha interface a agregar a un bean, sea otro bean.

- La implementación de “**introductions**” es similar a Method Injection o Method Replacement.

iii.vSpring AOP con XML (b)

- Introductions configuración XML

- La implementación de "**introductions**" por configuración por @Anotaciones no se recomienda, debido a que no es tan extensible como la configuración XML.

- Spring utiliza "**introductions**" extensivamente de forma interna, sin embargo, lo hace a través de su API mediante **ProxyFactory** y **ProxyFactoryBean** (no cubierto en este curso).

- No se recomienda la utilización de ProxyFactory y ProxyFactoryBean.

iii.vSpring AOP con XML (c)

- Configuración de “Introductions” por XML

- Se utiliza el elemento <aop:declare-parents> para declarar que los tipos específicos (que “**matchean**”) implementarán una nueva interface y que dicha nueva interface es implementada por un bean en el cual se delega la responsabilidad de implementación.

```
<aop:declare-parents
        types-matching=”<matching types and subtypes>"
        implement-interface=”<new interface to implement in matching types>"
        delegate-ref=”<bean reference that implements new interface>" />
```

iii.vSpring AOP con XML (d)

- Configuración de “Introductions” por XML

```
<aop:declare-parents
        types-matching="org.certificatic..IStudentAdditionalDetails+"
        implement-interface="org.certificatic..IStudent"
        delegate-ref="student" />

<bean id="student" class="org.certificatic.spring.aop..bean.Student” /

<bean id="studentAdditionalDetails"
    class="org.certificatic.spring.aop..bean.StudentAdditionalDetails” />
```

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d.Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML. Práctica f (a)

- Práctica f. Introductions configuración XML.

- Implementar “introductions” con Spring AOP mediante configuración XML.
- Extender dinámicamente un bean a través de la implementación de nuevas interfaces a través de aspectos y la implementación del concepto de “introductions”.

v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d.Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML (a)

- Advisors configuración XML
- Un advisor es un aspecto ”auto-contenido” que implementa uno o más tipos de advice.
- Define la ejecución de uno o varios advice en un pointcut determinado bajo una misma definición.
- No tiene un equivalente en @AspectJ.
- Los advisor son ampliamente implementados en Spring Tx.

iii.vSpring AOP con XML (b)

- Advisors configuración XML

```
public class BusinessService {

    @MonitorPerformance
    public void execute() {
        log.info("Init execute method ...");
        // long running task
        log.info("execute method ended.");
    }
}
```

```
@Retention(RetentionPolicy.RUNTIME)
public @interface MonitorPerformance {

}
```

iii.vSpring AOP con XML (c)

- Advisors configuración XML

```
public class PerformanceAdvice implements MethodBeforeAdvice, AfterReturningAdvice {
        private long startTime = 0;
        private long finishTime = 0;

        @Override
        public void before(Method method, Object[] args, Object target) throws Throwable {
                startTime = System.currentTimeMillis();
                System.out.println("Executing method " + method.getName() + " on object " +
                                                                target.getClass().getName());
        }

        …
```

iii.vSpring AOP con XML (d)

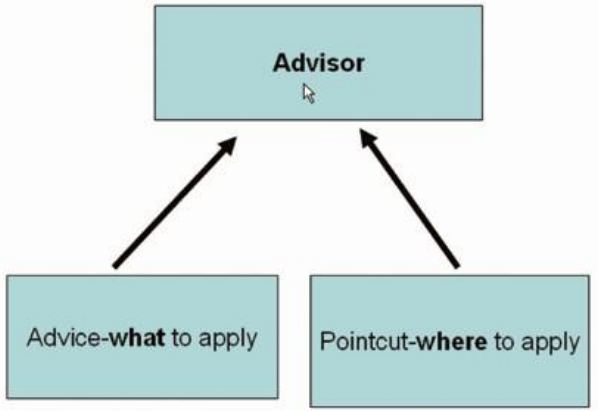
- Advisors configuración XML

```
…
@Override
public void afterReturning(Object returnValue, Method method,
                                        Object[] args, Object target) throws Throwable {
    finishTime = System.currentTimeMillis();
    double totalDuration = finishTime - startTime;

    System.out.println("Finished executing method " + method.getName()

                    + " on object " + target.getClass().getName() + " in " +
                                        totalDuration / 1000 + " seconds."));
    }
}
```

iii.vSpring AOP con XML (e)

- Advisors configuración XML

```
<aop:config>
        <aop:pointcut id="performanceMonitoredMethod"
                        expression="execution(* org.certificatic..BusinessService(..)) and
                                        @annotation(org.certificatic..annotation.MonitorPerformance)" />

        <aop:advisor advice-ref="performanceAdvice"
                        pointcut-ref="performanceMonitoredMethod" />

</aop:config>

<bean id="performanceAdvice"
        class="org.certificatic..advice.PerformanceAdvice" />
```



v. Spring AOP con XML

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 22. Spring AOP configuración XML

d.Introductions configuración XML Práctica

f. Introductions configuración XML

e. Advisors configuración XML

Práctica g. Advisors configuración XML

iii.vSpring AOP con XML. Práctica g (a)

- Práctica g. Advisors configuración XML.

- Implementar advisors con Spring AOP mediante configuración XML.
- Implementar las interfaces comunes de bajo nivel de Spring AOP para la integrar advisors.

Resumen de la lección

iii.v Spring AOP con XML

- Conocimos cuales son las dependencias requeridas para Spring AOP.
- Aprendimos como configurar aspectos con Spring AOP.
- Aprendimos a definir pointcuts.
- Aprendimos a configurar advices.
- Comprendimos como configurar Introductions y Advisors.
- Implementamos todos los posibles advices que provee Spring AOP mediante configuración por XML.

Esta página fue intencionalmente dejada en blanco.

iii. Spring AOP - iii.v Spring AOP con XML

iii.vi Spring AOP con @Anotaciones

Objetivos de la lección

iii.vi Spring AOP con @Anotaciones

- Comprender como se implementan aspectos por medio de configuración con anotaciones @AspectJ.
- Comprender como se implementan los distintos tipos de advice por medio de configuración con anotaciones @AspectJ.
- Comprender como se pasan parámetros a los métodos advice mediante configuración con anotaciones @AspectJ.

vi. Spring AOP con @Anotaciones

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 23. Spring AOP configuración @Anotaciones

iii.vi Spring AOP con @Anotaciones (a)

- Dependencias

- Spring AOP utiliza por default AspectJ para definir pointcuts mediante expresiones.

- El Framework AspectJ contiene sus propias anotaciones para la implementación de aspectos, por tanto Spring AOP no reinventa nuevas anotaciones y éstas se encuentran en **aspectjrt-<version>.jar**

- Spring AOP mediante configuración por anotaciones requiere la dependencia **spring-aop**.

vi. Spring AOP con @Anotaciones

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 23. Spring AOP configuración @Anotaciones

iii.vi Spring AOP con @Anotaciones (a)

- Configuración de aspectos

- Habilitar el namespace *aop.*

- Habilitar el soporte para configuración de aspectos mediante anotaciones @AspectJ
  - Usando configuración de beans por XML:
    <aop:aspectj-autoproxy/>
  - Usando configuración de beans por Anotaciones:
    @EnableAspectJAutoProxy

iii.vi Spring AOP con @Anotaciones (b)

- Definición de un aspecto

- La anotación @org.aspectj.lang.annotation.Aspect se utiliza para definir un aspecto, el cual debe ser también por un bean de Spring

```
@Aspect
@Component
public class MyAspect {
    …
}
```

- La anotación **@Aspect** no es escaneada por <context:component-scan/> para la detección del Bean, es requerido utilizar **@Component**.

iii.vi Spring AOP con @Anotaciones (c)

- Definición de un pointcut
- Es posible definir un pointcut mediante la anotación **@Pointcut**
- La definición de un pointcut sobre un método siempre debe tener void como tipo de retorno.
- La anotación **@Pointcut** define la expresión de Join Points y la firma del método anotado define el id del pointcut.
- Un aspecto no puede ser interceptado (advised) por otro aspecto.

iii.vi Spring AOP con @Anotaciones (d)

- Definición de un pointcut

```
@Aspect
public class PointcutDefinition {
    @Pointcut("within(org.bank.web..*)")
    public void webLayer() {}

}
```

```
<aop:pointcut id="webLayer"
        expression="within(org.bank.web..*)"/>
```

- Es posible definir un @Pointcut en una clase no manejada por Spring, sin embargo la clase debe anotarse con @Aspect.

iii.vi Spring AOP con @Anotaciones (e)

- Combinando expresiones

- Es posible utilizar los operadores lógicos (&&, || y !) para combinar expresiones en un único pointcut definido por @Pointcut o combinar expresiones pointcut en otro único @Pointcut.

- Al igual que en configuración Spring AOP por XML, el operador * indica *cualquiera* y el operador .. indica *cualquiera cuantos quiera.*

iii.vi Spring AOP con @Anotaciones (f)

- Combinando expresiones

- Ejemplo:

```
@Pointcut("execution( void * *(com.xyz.model.Account, ..) ) && args(account,..) ")
private void anyMethodWithAccount() {}
```

```
<aop:pointcut id="anyMethodWithAccount"
        expression=" execution( void * *(com.xyz.model.Account, ..) ) && args(account,..) "/>
```

iii.vi Spring AOP con @Anotaciones (g)

- Combinando expresiones pointcut

- Ejemplo:

```
@Pointcut("execution( void * *(com.xyz.model.Account, ..) ) && args(account,..) ")
private void anyMethodWithAccount() {}

@Pointcut("within(com.xyz.app.trading..*)")
private void inTradingPackage() {}

@Pointcut("anyMethodWithAccount() &&
inTradingPackage()")  private void
tradingOperationWithAccount() {}
```

iii.vi Spring AOP con @Anotaciones (h)

- Combinando expresiones pointcut

- Es posible hacer referencia a identificadores de @Pointcut (nombre del método) desde otro aspecto (clase) utilizando el nombre calificado del método.

```
@Aspect
public class TradingAspect {
    @Pointcut("com.xyz.aop.PointcutDefinition.anyMethodWithAccount()
    &&
                                com.xyz.aop.PointcutDefinition.inTradingPackage()")
    private void tradingOperationWithAccount() {}
}
```

iii.vi Spring AOP con @Anotaciones (i)

- Combinando expresiones pointcut

- Ejemplo:

```
<aop:pointcut id="anyMethodWithAccount"
        expression="execution( void * *(com.xyz.model.Account, ..) ) && args(account,..)"/>

<aop:pointcut id="inTradingPackage"
        expression="within(com.xyz.app.trading..*)"/>

<aop:pointcut id="tradingOperationWithAccount"
        expression="anyMethodWithAccount &&
        inTradingPackage"/>
```

vi. Spring AOP con @Anotaciones

a. Dependencias

b. Configuración de aspectos

c. Configuración de advices

Práctica 23. Spring AOP configuración @Anotaciones

iii.vi Spring AOP con @Anotaciones (a)

- Configuración de advices: Before advice, con referencia a pointcut.

```
@Aspect
@Component
public class BeforeExample {

    @Before("com.xyz.app.PointcutDefinition.dataAccessOperation()")
    public void doAccessCheck() {
        ...
    }
}
```

```
<aop:before pointcut-ref="dataAccessOperation"
                                    method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (b)

- Configuración de advices: Before advice, definiendo pointcut en línea.

```
@Aspect
@Component
public class BeforeExample {
   @Before("execution(*
   com.xyz.app.dao..*.*(..))")  public void
   doAccessCheck() {
}      ...
   }
```

```
<aop:before pointcut="execution(* com.xyz.app.dao..*.*(..))"
                                        method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (c)

- Configuración de advices: After Returning advice, sin returning value.

```
@Aspect
@Component
public class AfterReturningExample {
    @AfterReturning("com.xyz.app.PointcutDefinition.dataAccessOperation()")
    public void doAccessCheck() {
        ...
    }
}
```

```
<aop:after-returning pointcut-ref="dataAccessOperation"
                     method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (d)

- Configuración de advices: After Returning advice, con returning value.

```
@Aspect
@Component
public class AfterReturningExample {
    @AfterReturning(value="execution(* com.xyz.app.dao..*.*(..))",
                    returning="returnValue")
    public void doAccessCheck(Object returnValue) {
        ...
    }
}
```

```
<aop:after-returning pointcut="execution(* com.xyz.app.dao..*.*(..))"
                     returning="returnValue" method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (e)

- Configuración de advices: After Throwing advice, sin throwing exception.

```
@Aspect
@Component
public class AfterThrowingExample {
    @AfterThrowing("com.xyz.app.PointcutDefinition.dataAccessOperation()")
    public void doAccessCheck() {
        ...
    }
}
```

```
<aop:after-throwing pointcut-ref="dataAccessOperation"
                    method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (f)

- Configuración de advices: After Throwing advice, con throwing exception.

```
@Aspect
@Component
public class AfterThrowingExample {
    @AfterThrowing(value="execution(* com.xyz.app.dao..*.*(..))", throwing="ex")
    public void doAccessCheck(DataAccessException ex) {
        ...
    }
}
```

```
<aop:after-throwing pointcut="execution(* com.xyz.app.dao..*.*(..))"
                    throwing="ex" method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (g)

- Configuración de advices: After advice (finally), con referencia a pointcut.

```
@Aspect
@Component
public class AfterExample {
    @After("com.xyz.app.PointcutDefinition.dataAccessOperation()")
    public void doAccessCheck() {
        ...
    }
}
```

```
<aop:after pointcut-ref="dataAccessOperation"
                    method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (h)

- Configuración de advices: After advice (finally), definiendo pointcut en línea.

```
@Aspect
@Component
public class AfterExample {

   @After("execution(* com.xyz.app.dao..*.*(..))")
   public void doAccessCheck() {

   } ...
}
```

```
<aop:after pointcut="execution(* com.xyz.app.dao..*.*(..))"
                                        method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (i)

- Configuración de advices: Around advice
- El Around advice se ejecuta alrededor de la ejecución del método del target object.
- Permite trabajar con todos los diferentes advices de forma programática.
- Para definir un Around advice mediante configuración por anotaciones se usa la anotación **@Around**

iii.vi Spring AOP con @Anotaciones (k)

- Configuración de advices: Around advice.

```
@Aspect
@Component
public class AroundExample {

    @Around("execution(* com.xyz.app.dao..*.*(..))")
    public Object doAccessCheck(ProceedingJoinPoint pjp) throws Throwable {
        return pjp.proceed();
    }

}
```

```
<aop:around pointcut="execution(* com.xyz.app.dao..*.*(..))"
            method="doAccessCheck"/>
```

iii.vi Spring AOP con @Anotaciones (I)

- Configuración de advices: Acceso al Join Point actual

- Cualquier método advice (excepto Around) puede declarar como primer parámetro un objeto del tipo **org.aspectj.lang.JoinPoint.**

- **JoinPoint**: Provee de numerosos métodos útiles como:
  - getArgs(): Retorna los argumentos del método.
  - getThis(): Retorna el proxy object.
  - getTarget(): Retorna el target object.
  - getSignature(): retorna la firma del método aconsejado (advised méthod).

iii.vi Spring AOP con @Anotaciones (m)

- Configuración de advices: Pasando parámetros al advice

- Para pasar parámetros al método advice es necesario usar el *designador* **args**.

- El *designador* **args** requiere de una expresión donde se le pase el nombre de los parámetros de los argumentos requeridos. El nombre del parámetro, así como el orden asignado en **args** debe *matchear* con la firma del método advice.
  En caso de que el nombre del parámetro no haga *match* especificar el nombre de o los atributos en el atributo **argNames** separados por *comma*.

iii.vi Spring AOP con @Anotaciones (n)

- Configuración de advices: Pasando parámetros al advice

- Dado la siguiente clase target object:

```
package

org.certificatic.spring.aop.practica24.bank.service.account.api.impl;

@Service

public class AccountService implements IAccountService {
    public void updateAccountBalance(Account account, Long amount) {
        …
    }
}
```

iii.vi Spring AOP con @Anotaciones (o)

- Configuración de advices: Pasando parámetros al advice (a)

```
@Aspect
@Component
public class AccountLoggingAspect {

    @Before("within(org.certificatic.spring.aop.practica24.bank.service..*) && args(account, ..)")
    public void beforeAccountMethodExecution(Account account) {
        …
    }
}
```

iii.vi Spring AOP con @Anotaciones (p)

- Configuración de advices: Pasando parámetros al advice (b)

```
@Aspect
@Component
public class AccountLoggingAspect {

    @Pointcut("within(org.certificatic.spring.aop.practica24.bank.service..*) && args(yy, ..)")
    public void pointcutAccountMethodExecution(Account yy) { }

    @Before("pointcutAccountMethodExecution(bb)")
    public void beforeAccountMethodExecution(Account bb) {
        …
    }
}
```

iii.vi Spring AOP con @Anotaciones (q)

- Configuración de advices: Pasando parámetros al advice (c)

```
@Aspect
@Component
public class AccountLoggingAspect {

    @Before(value="within(org.certificatic.spring.aop.practica24.bank.service..*) && " +
            "args(cuenta, monto, ..)", argNames="monto, cuenta")
    public void beforeAccountMethodExecution(JoinPoint jp, Long amount, Account account) {
        …
    }
}
```

iii.vi Spring AOP con @Anotaciones (r)

- Recomendaciones sobre configuración de advices.

- Se recomienda la utilización de anotaciones @AspectJ tales como @Before, @After, @AfterThrowing, @AfterReturning y @Around en lugar de implementar las interfaces de bajo nivel del API de Spring AOP debido a que son interfaces de integración y de la especificación del API de AOP Alliance Org.

vi. Spring AOP con @Anotaciones

a. Dependencias
b. Configuración de aspectos
c. Configuración de advices

Práctica 23. Spring AOP configuración @Anotaciones

iii.vi Spring AOP con @Anotaciones. Práctica 23. (a)

- Práctica 23. Spring AOP configuración @Anotaciones.

- Implementar aspectos con Spring AOP mediante configuración con anotaciones @AspectJ.
- Implementar los cinco tipos de advice mediante configuración con anotaciones @AspectJ.

Resumen de la lección

iii.vi Spring AOP con @Anotaciones

- Aprendimos cómo configurar aspectos con Spring AOP con configuración por anotaciones @AspectJ
- Aprendimos a definir pointcuts con anotaciones @AspectJ.
- Aprendimos a configurar advices con anotaciones @AspectJ.
- Implementamos todos los posibles advices que provee Spring AOP mediante configuración por anotaciones.

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP.

Objetivos de la lección

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP

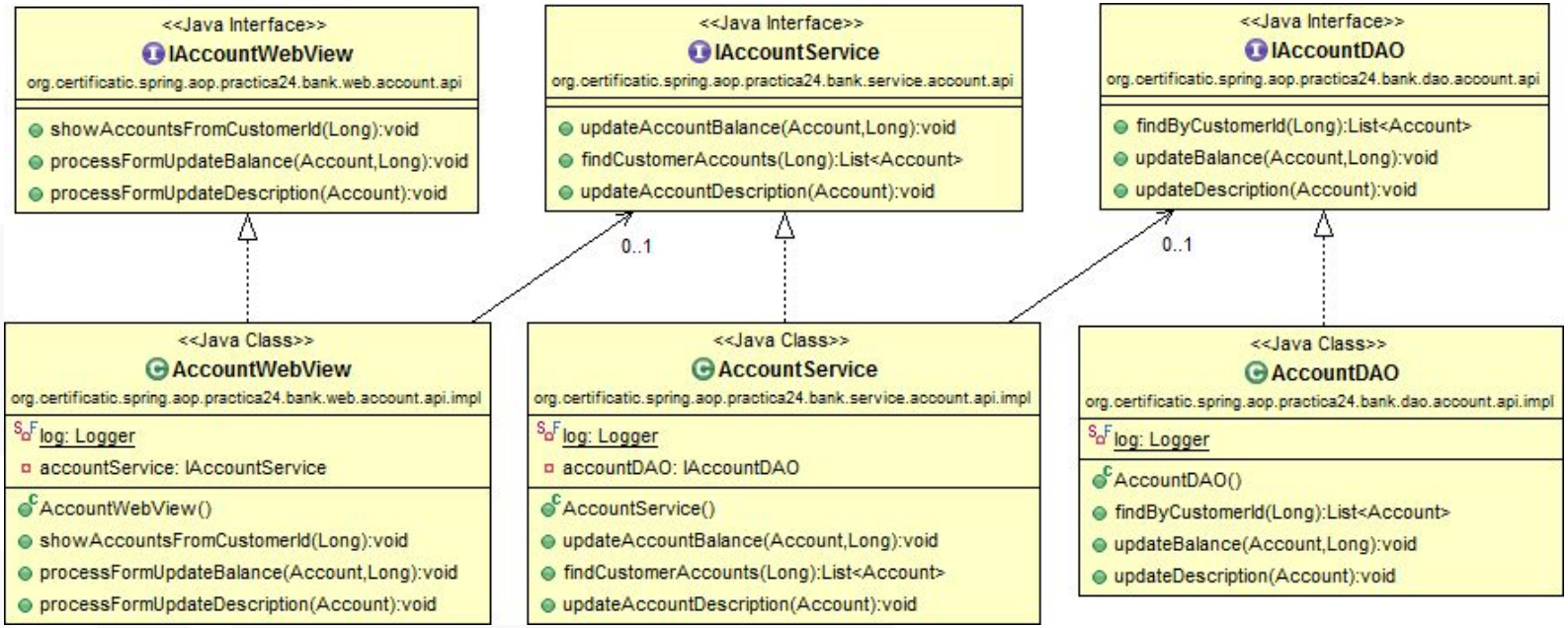
- Separar responsabilidades en componentes de servicios mediante la aplicación de aspectos.
- Comprender más en detalle la configuración de aspectos mediante Spring AOP.
- Implementar aspectos de logging y profiling con configuración por anotaciones @AspectJ y configuración por XML.

ii. Spring Core – T.I. 3. Implementación de Logging y Profiling mediante AOP

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP

Práctica 24. Spring AOP Logging y Profiling

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP (a)



ii. Spring Core – T.I. 3. Implementación de Logging y Profiling mediante AOP

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP (b)

- Dado el diseño anterior configurar los aspectos:
  - DAOAccountLoggingAspect
  - ServiceAccountLoggingAspect
  - WebAccountLoggingAspect, y
  - ProfilingAspect

- Nota: Configurar aspectos por configuración XML y anotaciones @AspectJ

ii. Spring Core – T.I. 3. Implementación de Logging y Profiling mediante AOP

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP

Práctica 24. Spring AOP Logging y Profiling

Trabajo de Integración 3. Práctica 24. (a)

- Práctica 24. Spring AOP Logging y Profiling

- Implementar aspectos por medio de configuración por anotaciones @AspectJ (guiado por instructor).

- Implementar aspectos por medio de configuración por XML.

ii. Spring Core – T.I. 3. Implementación de Logging y Profiling mediante AOP

Resumen de la lección

Trabajo de Integración 3. Implementación de Logging y Profiling mediante AOP (b)

- Separamos las responsabilidades de logging y profiling de las capas de presentación, servicios y acceso a datos.
- Comprendimos más a fondo el beneficio de implementar módulos transversales (cross-cutting concerns) mediante Spring AOP.
- Implementamos aspectos de logging y profiling con configuración por anotaciones @AspectJ y configuración por XML.

ii. Spring Core – T.I. 3. Implementación de Logging y Profiling mediante AOP

Esta página fue intencionalmente dejada en blanco.

ii. Spring Core – T.I. 3. Implementación de Logging y Profiling mediante AOP