Curso de Spring Core

IT Formación

Profesor: Eduardo Escofet

2017

Objetivos

Entender la forma de trabajo con Spring

Poder desarrollar aplicaciones básicas Java con Spring

Entender la arquitectura Spring

Entender la inyección de dependencia en Spring

Modo de desarrollar aplicaciones Web y de servicios (clásico y moderno)

Entender la programación orientada a aspectos en Spring

Integrar conceptos en la práctica

Pre requisitos

Conocimiento del lenguaje Java

Forma de trabajo con aplicaciones Web y servicios

Rudimentos de trabajo con JSON y XML

Manejo básico con Maven

(Se tocarán brevemente si es necesario)

Prerrequisitos

Software:

JDK 8.x
Maven 3.x
IDE Eclipse
Contenedor Web (Tomcat, Jetty, Undertow)
Servidor de aplicaciones (Glassfish, Payara, Wildfly)
Bibliotecas JSON y XML (on demand)
Spring 4.x (on demand)

Bibliografía

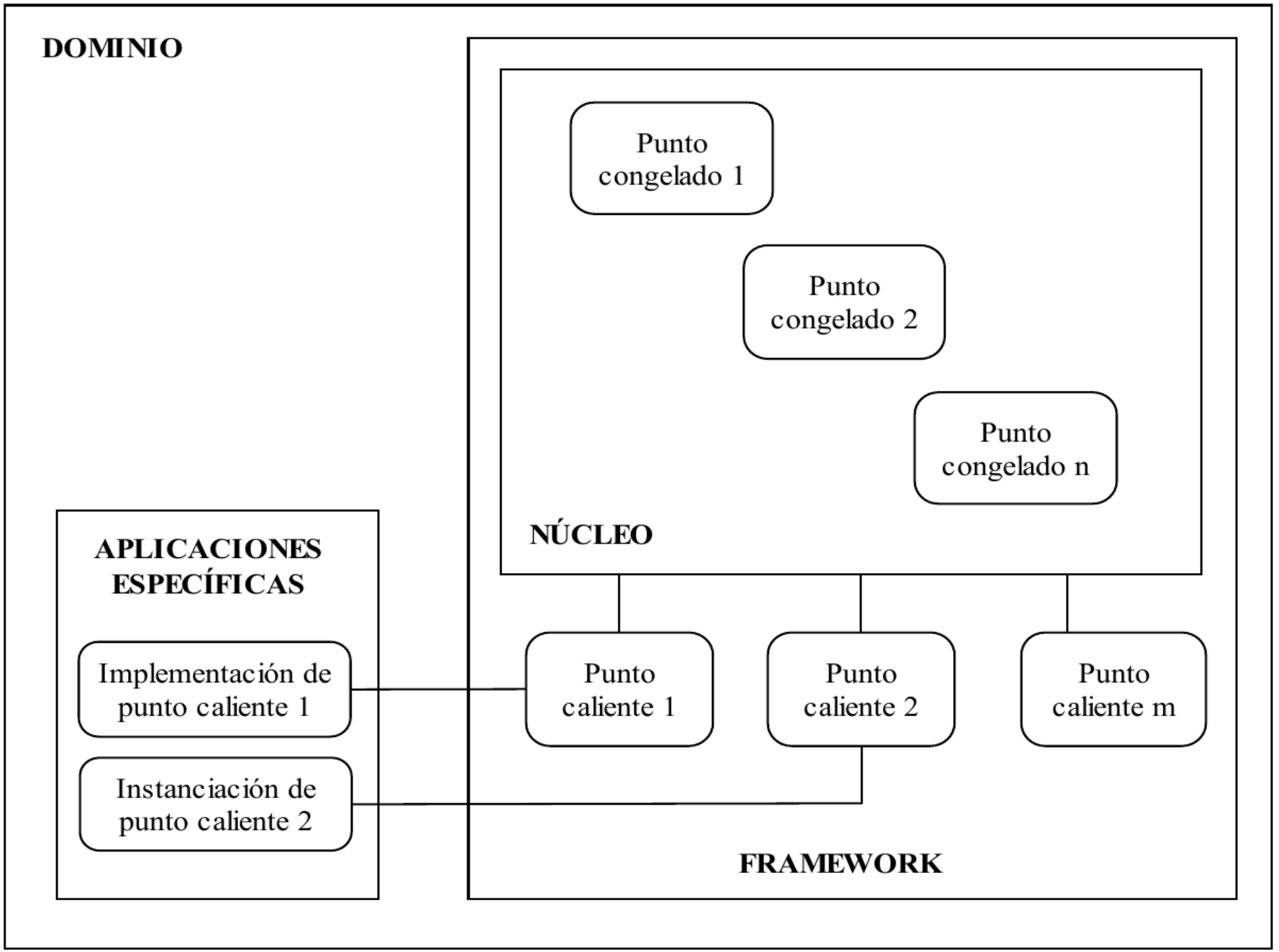
Spring in Action, 4th Edition, 2014.
Pro Spring Boot, 2016.
Spring MVC Cookbook, 2016.
Spring Boot in Action, 2015.
Learning Spring Application Development, 2015.
Beginning Spring, 2015.
Spring Recipes, 3rd Edition, 2014.
Introducing Spring Framework, 2014.

Framework Spring

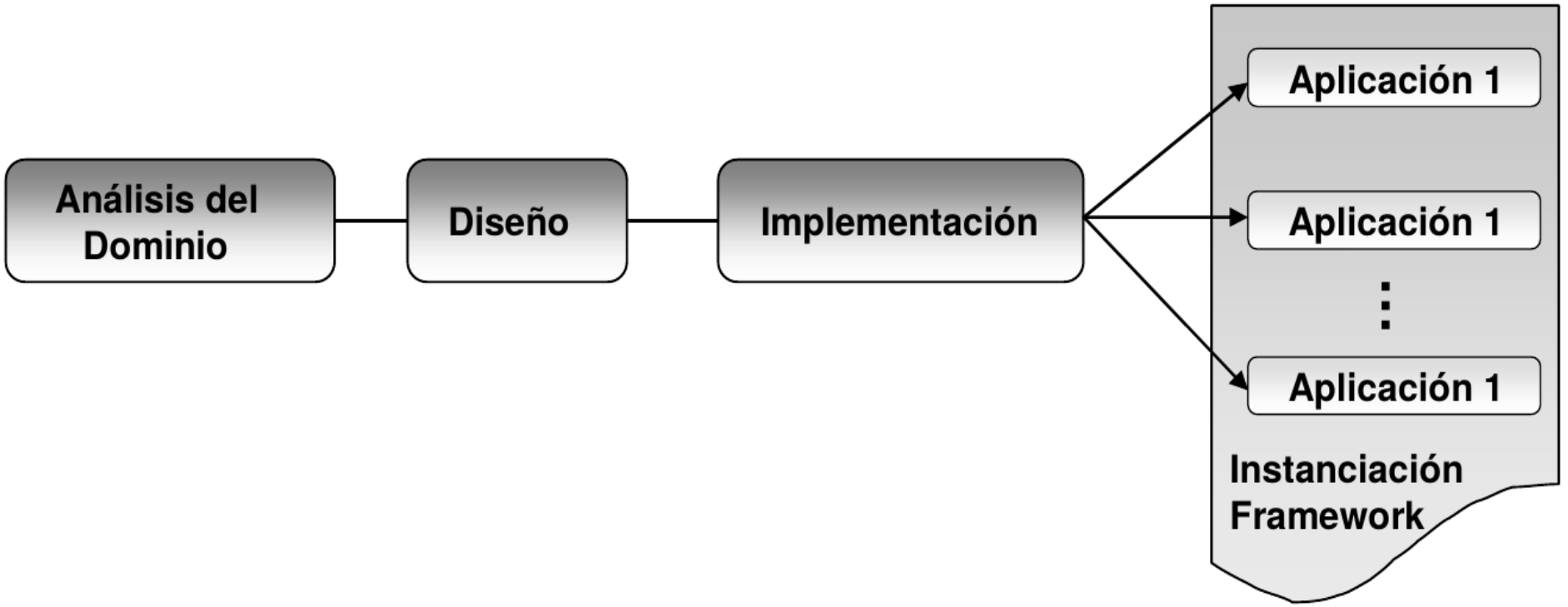
Es el conjunto de entidades software que modelan las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio arquitectónico.

Características:
Reusabilidad
Extensibilidad
Rendimiento
Productividad
Interoperabilidad
Buenas prácticas

Esquema abstracto de un framework



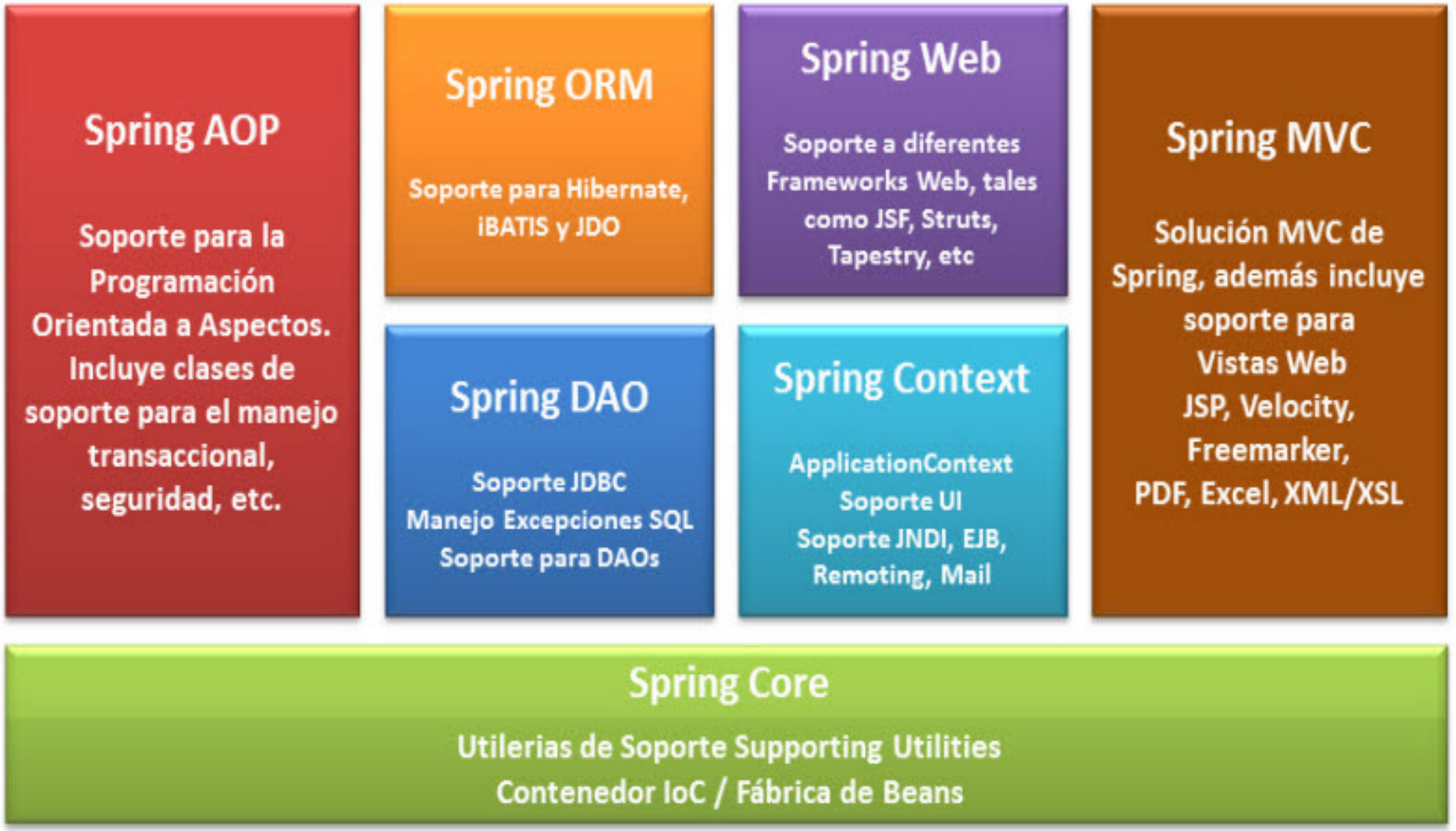
Desarrollo de un framework
(Rod Johnson, Markiewicz y Lucena)



Principios básicos sobre los que se basa el framework Spring

Versión más ligera que los servicios del JavaEE
Especificación abierta y dinámica
Respeto de buenas prácticas y patrones
Modularización
Separation of Concerns (Separación de intereses)
Contenedor de Inversión de Control
Programación Orientada a Aspectos

Módulos de Spring Framework



Algunos patrones que se ven en Spring

SoC
IoC
DI (caso específico del IoC)
Facade
Builder
Singleton
Factory
Abstract
Iterator
Template
etc.

Separación de intereses

La separación de intereses es un principio de diseño de modularización. Los intereses persisten a lo largo del código de las aplicaciones y pueden ser muy generales o particulares, eg. La optimización del código para un SO determinado, la validación de datos, logging, caching, etc.

La POO separa intereses mediante clases encapsuladas MVC aplica el principio separando la presentación del contenido SOA separa intereses aislando y desacoplando servicios.

Inversion of Control (IoC)

La inversión de control es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales.

Se le conoce como el “Principio Hollywood” donde se preparan respuestas a mensajes recibidos desde las bibliotecas de clases.

El caso más claro es cuando los frameworks implementan lógica de alto nivel y el bajo nivel queda al usuario.

Dependency Injection (DI)

La inyección de dependencias es un caso específico de la IoC y consiste en suministrar objetos a una clase (inyectar) En vez de que la propia clase cree los objetos (construir, builder, factory, etc.)

Los frameworks modernos incluyen, en la mayoría de los casos, este patrón de diseño y se conoce como contenedor DI o IoC.

Dependency Injection (DI)

Algunos de los frameworks más conocidos que incluyen contenedores DI son: CDI, Spring IoC, Guice, PicoContainer, Ninject, etc.

IoC es más general que DI, un ejemplo de esto es el patrón Template o el Service Locator

La utilización del SL no es recomendada porque depende de una entidad central para descubrir los servicios que permiten ejecutar una tarea, a menudo se identifica como un anti-pattern.

Dependency Injection (Before)

```
public class Client {
    // Internal reference to the service used by this client
    private Service service;

    public Client() {
    /* Specify an implementation in the constructor instead of
using dependency injection*/
        service = new ServiceExample();
    }

    // Method within this client that uses the services
    public String greet() {
        return "Hello " + service.getName();
    }
}
```

Dependency Injection (After)

```
public class Client {
    private Service service;

    public Client(Sevice service) {
    // Using constructor dependency injection
        this.service = service;
    }

     public void setService(Sevice service) {
     // Using setter dependency injection
        this.service = service;
    }

    public String greet() {
        return "Hello " + service.getName();
    }
}
```

Dependency Injection (After)

```
// Service setter interface.
public interface ServiceSetter {
    public void setService(Service service);
}

// Client class
public class Client implements ServiceSetter {
    // Internal reference to the service used by this client.
    private Service service;

    // Set the service that this client is to use.
    @Override
    public void setService(Service service) {
        this.service = service;
    }
}
```

Aspect-Oriented Programming (AOP)

La programación orientada a aspectos es un paradigma de desarrollo que permite modularizar intereses.

Un aspecto es una funcionalidad transversal (cross-cutting) y se puede identificar en un programa cuando:

1- Es una funcionalidad común que se aplica a lo largo de todo el programa.

2- Es una funcionalidad detectada que no pertenece lógicamente a un módulo.

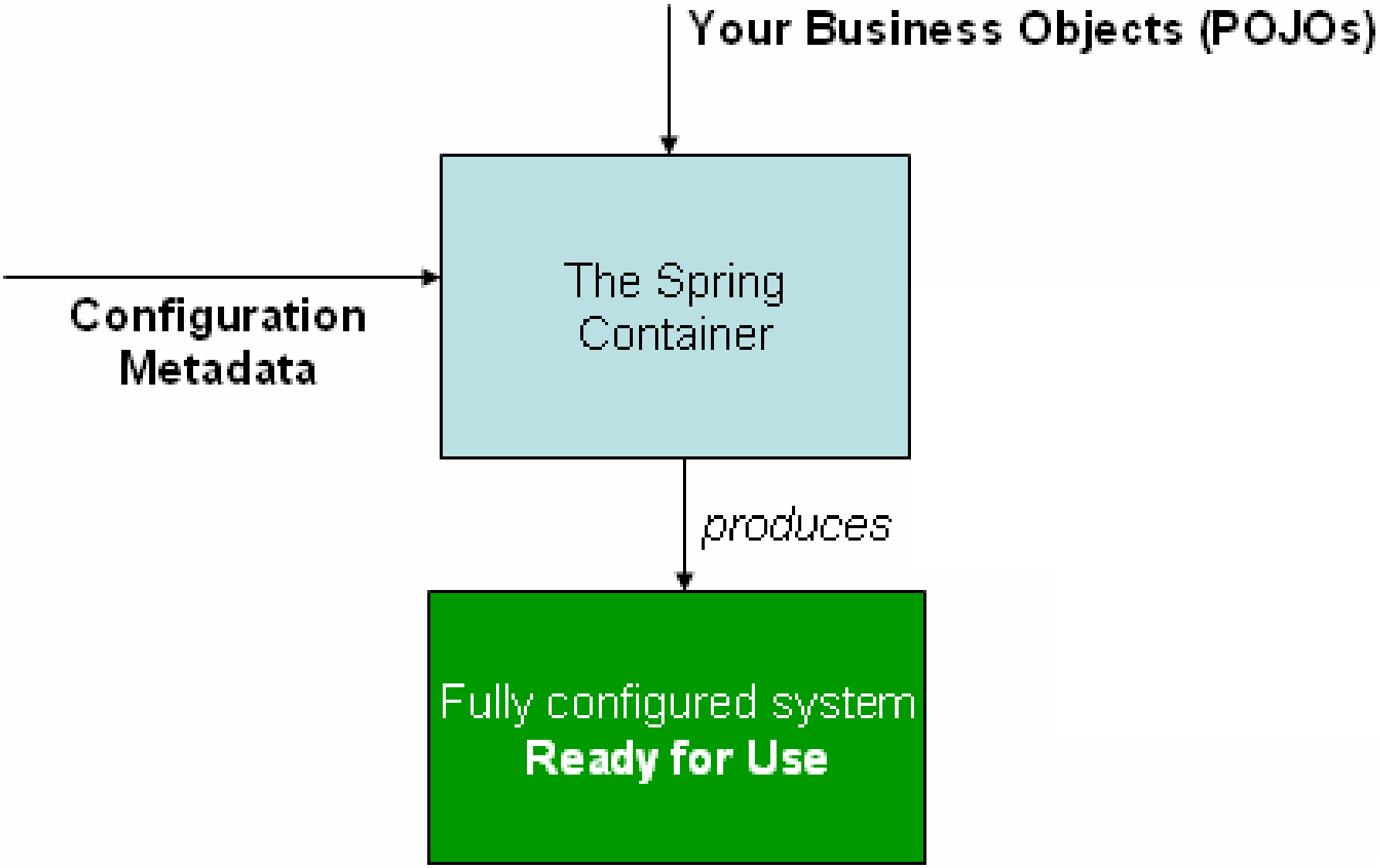
Aspect-Oriented Programming (AOP)

Algunos aspectos identificados en las aplicaciones modernas son el caching, logging y la seguridad de los módulos.

En Java existen diferentes extensiones al lenguaje para programar orientado a aspectos:

-AspectJ
-Google Guice
-Spring AOP
-JBoss AOP

Spring IoC


Your Business Objects (POJOs)
Configuration Metadata
The Spring Container
produces
Fully configured system
Ready for Use

Spring IoC

La configuración puede ser expresada en XML, pero Spring ha ido incorporando diversas formas de expresar esta metadata.

Desde Spring 2.5 se introdujo la configuración basada en anotaciones.

Desde Spring 3.0 muchas de las características del proyecto Spring JavaConfig pasan a formar parte del Core. Así, se puede usar beans externos a su aplicación escritos en Java y con anotaciones especiales como @Configuration, @Bean, @Import and @DependsOn annotations.

Spring IoC

Para la configuración en XML se crean los ficheros XML con las configuraciones de los Beans (clases Java contenedoras de datos con método de acceso y serializables)

Para utilizar el contenedor IoC hay que instanciarlo en nuestro programa

```
ApplicationContext context = new
ClassPathXmlApplicationContext(new String[] {"services.xml",
"daos.xml"});
```

Cuando hay muchas definiciones se suelen agrupar mediante importaciones. Generalmente casa definición XML representa una capa o módulo de la aplicación.

Spring IoC

Utilizarlo en un programa real evidencia su dinamismo:

```
// create and configure beans
ApplicationContext context = new
ClassPathXmlApplicationContext(new String[] {"services.xml",
"daos.xml"});

// retrieve configured instance
PetStoreService service = context.getBean("petStore",
PetStoreService.class);

// use configured instance
List<String> userList = service.getUsernameList();
```

SpringBeanAutowiring

Principales anotaciones:

@Configuration – Indica que una clase declara al menos un método @Bean y puede ser procesada por el contendor Spring para generar definiciones y llamadas a servicio en tiempo de ejecución.

@ComponentScan – Configura la búsqueda de componentes para usar conjuntamente con las clases @Configuration. Se busca en paquete base.

@Bean – Indica que un método produce un bean gestionado por el contenedor Spring.

@Component – Estereotipo genérico que indica que ese componente será gestionado por Spring

@Autowired – Permite sustituir a la etq property dejando al Spring el enlace que tratará de resolverlo por tipo

springintro

Ejercicio:

Dadas las salidas de la peticiones al sitio: http://fixer.io

elaborar un proyecto Maven – Spring que permita procesar esa salida y según configuraciones de sus propiedades XML obtener el cambio según la moneda especificada.

Para mayor facilidad tener un fichero json o un literal con la respuesta del servicio para procesarlo.

Herencia y templates

Existe la herencia de beans, que funciona parecido a la de Java.

Para establecer la herencia se utiliza el atributo parent especificando su valor como el nombre del bean padre

También se pueden escribir templates xml para que sirvan de plantilla de otros beans (sin necesidad de definir una clase extra)

Scope

El scope o alcance es el espacio de vida de un bean. Existen 5 tipos diferentes:

singleton - Limita al bean a una sola instancia por contenedor IoC (por defecto)

prototype – El bean puede tener cualquier número de instancias

request – Limita el bean a una petición HTTP (web-aware)

session - Limita el bean a una sesión HTTP (web-aware)

global-session – Limita el bean a una sesión HTTP global (web-aware)

Beans embebidos y colecciones

En el caso de las colecciones se pueden especificar cuatro tipos de propiedades diferentes:

list – Soporta inyectar una lista de valores permitiendo duplicados.

set – Soporta inyectar un conjunto de valores sin duplicados.

map – Soporta inyectar una colección nombre-valor siendo estos de cualquier tipo.

props – Soporta inyectar una colección nombre-valor donde estos son cadenas.

Beans embebidos y colecciones

Ejemplos sobre como pasar Beans embebidos (embedded/inner)

```
<bean id="textEditor" class="spelling.TextEditor">
    <property name="spellChecker">
        <bean id="spellChecker" class="spelling.SpellChecker"/>
    </property>
 </bean>
```

Para una colección

```
  <bean id="javaCollection" class="com.itformacion.JavaCollection">
    <!-- results in a setAddressList(java.util.List) call -->
    <property name="addressList">
      <list>
        <value>Spain</value>
        <value>Germany</value>
        <value>France</value>
      </list>
    </property> ...
```

Ejercicio integrador

Entender la problemática del ejercicio en el enlace GitHub indicado. Comprender la utilización de estructuras y modos específicos del Java 8 empleados en la parte ya resuelta del ejercicio. Comenzar a adicionarle funcionalidades que hasta ahora se han visto.

https://github.com/escofet/FlightSearch.git

Autowiring

La anotación @Autowired se utiliza para la inyección automática de dependencias

Existen 6 formas diferentes de autowiring:

Por nombre – En este caso se utiliza el setter para la inyección de la dependencia. También el nombre debe ser el mismo en la clase donde se inyecta la dependencia y en el fichero de configuración del bean.

Por tipo – En este caso, el tipo de la clase es utilizado. Por lo tanto, debe existir solo un bea cnfigurado para este tipo en el fichero de configuración.

Autowiring

Por constructor – Este es muy parecido al autowiring por tipo, la diferencia estriba en que se utiliza el constructor para inyectar la dependencia.

Por autodetección – A partir del Spring 3 está disponible esta opción. Es utilizada por las opciones de autowiring por tipo y por constructor. Se considera obsoleta en la actualidad.

Por anotación @Qualifier – Se utiliza para evitar conflictos en el mapeo de beans y debemos proporcionar el nombre del bean que se utilizará en el autowiring. Especialmente útil cunado existen varios beans para el mismo tipo.

Autowiring

Por anotación @Autowired – Podemos utilizar esta anotación para el autowiring de beans. Se puede aplicar en variables y métodos para resolver por tipo. Si se utiliza en un cosntructor se utiliza la resolución por constructor.

Para que esta anotación funcione se necesita habilitar la configuración basada en anotaciones en el fichero de configuración de beans del Spring. Puede ser hecho definiendo el elemento context:annotation-config o definiendo un bean de tipo

org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor

Definición de servicios Web

Un servicio web (web service) es un componente software que utiliza protocolos y estándares para intercambiar datos entre aplicaciones.

Principales conceptos

Web Services Protocol Stack: conjunto de servicios y protocolos de los servicios web.

XML (Extensible Markup Language): formato estándar para los datos que se vayan a intercambiar.

SOAP (Simple Object Access Protocol) o XML-RPC (XML Remote Procedure Call): protocolos sobre los que se establece el intercambio.

Principales conceptos

Otros protocolos: los datos en XML también pueden enviarse de una aplicación a otra mediante protocolos normales como Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), o Simple Mail Transfer Protocol (SMTP).

WSDL (Web Services Description Language): es el lenguaje de la interfaz pública para los servicios web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios web.

Principales conceptos

WS-Security (Web Service Security): protocolo de seguridad aceptado como estándar por OASIS (Organization for the Advancement of Structured Information Standards). Garantiza la autenticación de los actores y la confidencialidad de los mensajes enviados.

REST (Representational State Transfer): arquitectura que, haciendo uso del protocolo HTTP, proporciona una API que utiliza cada uno de sus métodos (GET, POST, PUT, DELETE, etcétera) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente.

Ventajas

Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.

Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

Desventajas

Debido al uso de XML para el paso de mensajes, SOAP es considerablemente más lento que otros middleware como CORBA, RMI.

Depende del WSDL.

Al contrario que Java, PHP o Python no se ofrece un apoyo adecuado para su uso ya sea a nivel de integración o de soporte IDE.

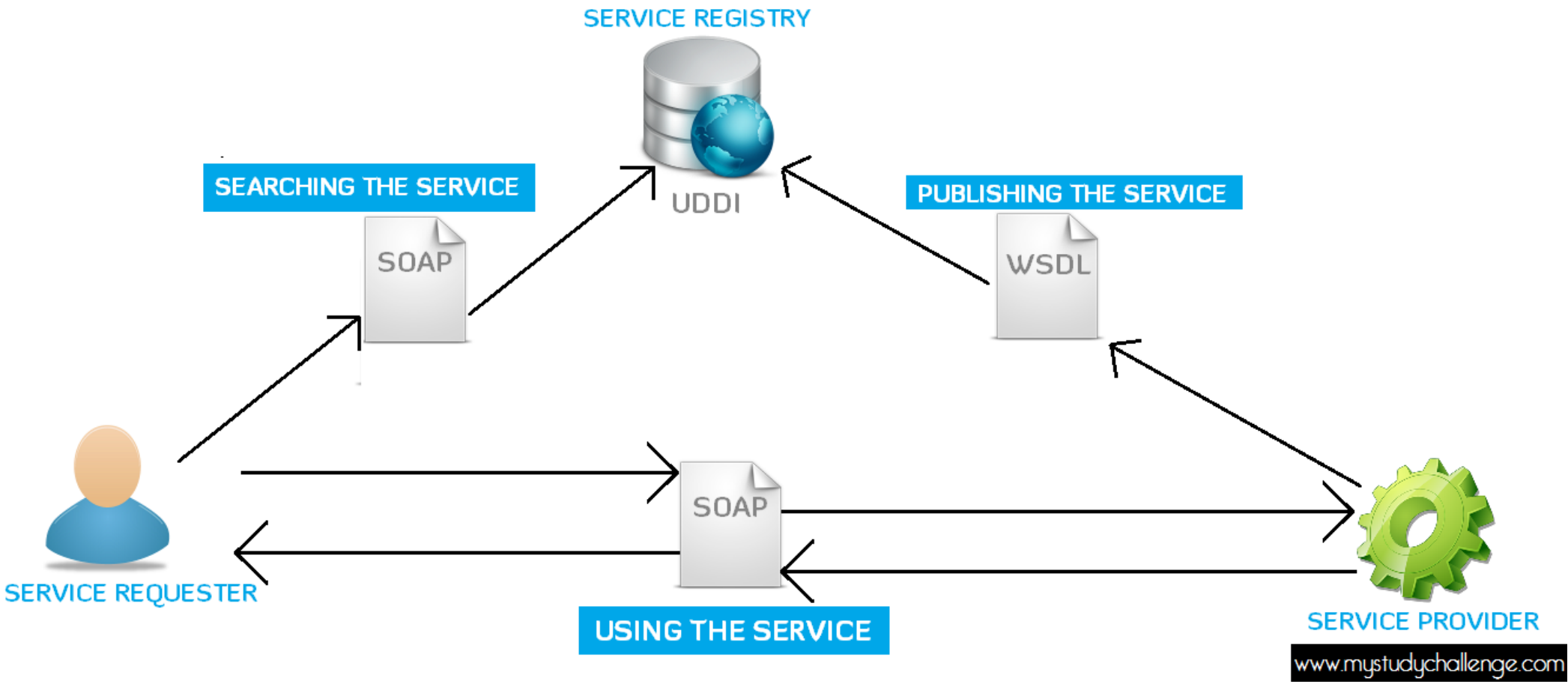
Se considera más complejo que otros tipos de servicios disponibles.

Frameworks y herramientas

Jersey (JAX-RS)	JAX-WS
Jackson (JSON)	GSON (JSON)
SoapUI	Maven
Sparkjava	Spring REST
JAXB	JAXP
Rested	Postman
NetBeans	Eclipse
Apache Tomcat	Glassfish

SOAP

Simple Object Access Protocol es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.



SOAP

UDDI - Catálogo de servicios de Internet denominado Universal Description, Discovery and Integration. Es un registro del catálogo de servicios en XML. Una analogía es un directorio telefónico.

WSDL - Web Services Description Language, es un formato XML que se utiliza para describir la interfaz púlica de los servicios web. Describe la forma de comunicarse, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en el catálogo.

Ejemplo SOAP de solicitud/respuesta

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productId>827635</productId>
     </getProductDetails>
   </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productName>Toptimate 3-Piece Set</productName>
         <productId>827635</productId>
         <description>3-Piece luggage set.  Black Polyester.</description>
         <price>96.50</price>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
</soap:Envelope>
```

REST

Representational State Transfer es una interfaz entre sistemas que utilizan directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos sin abstracciones extras.

Características:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes

REST - Características:

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Parecido a las operaciones CRUD en bases de datos

- Cada recurso es direccionable únicamente a través de su URI

- Es posible navegar de un recurso REST a otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

REST vs SOAP

REST es mejor cuando se exponen APIs públicas para gestionar operaciones CRUD sobre los datos.

REST se enfoca en acceder a recursos por nombre a través de una interfaz consistente.

SOAP define su propio protocolo.

SOAP se enfoca en exponer la lógica de la aplicación como servicios (no datos ni recursos). SOAP expone operaciones que implementan alguna lógica del negocio a través de diferentes interfaces.

REST vs SOAP

REST se recomienda en la mayoría de los casos por utilizar el protocolo HTTP (omnipresente y se lleva bien con proxies y firewalls y más sencillo de implementar)

REST tiene la ventaja de que en cada uso puede ser asignado a un servidor diferente (non-sticky).

REST admite varios formatos, SOAP se basa en XML

REST tiene mejor escalabilidad, sus lecturas pueden ser cacheadas al contrario de SOAP.

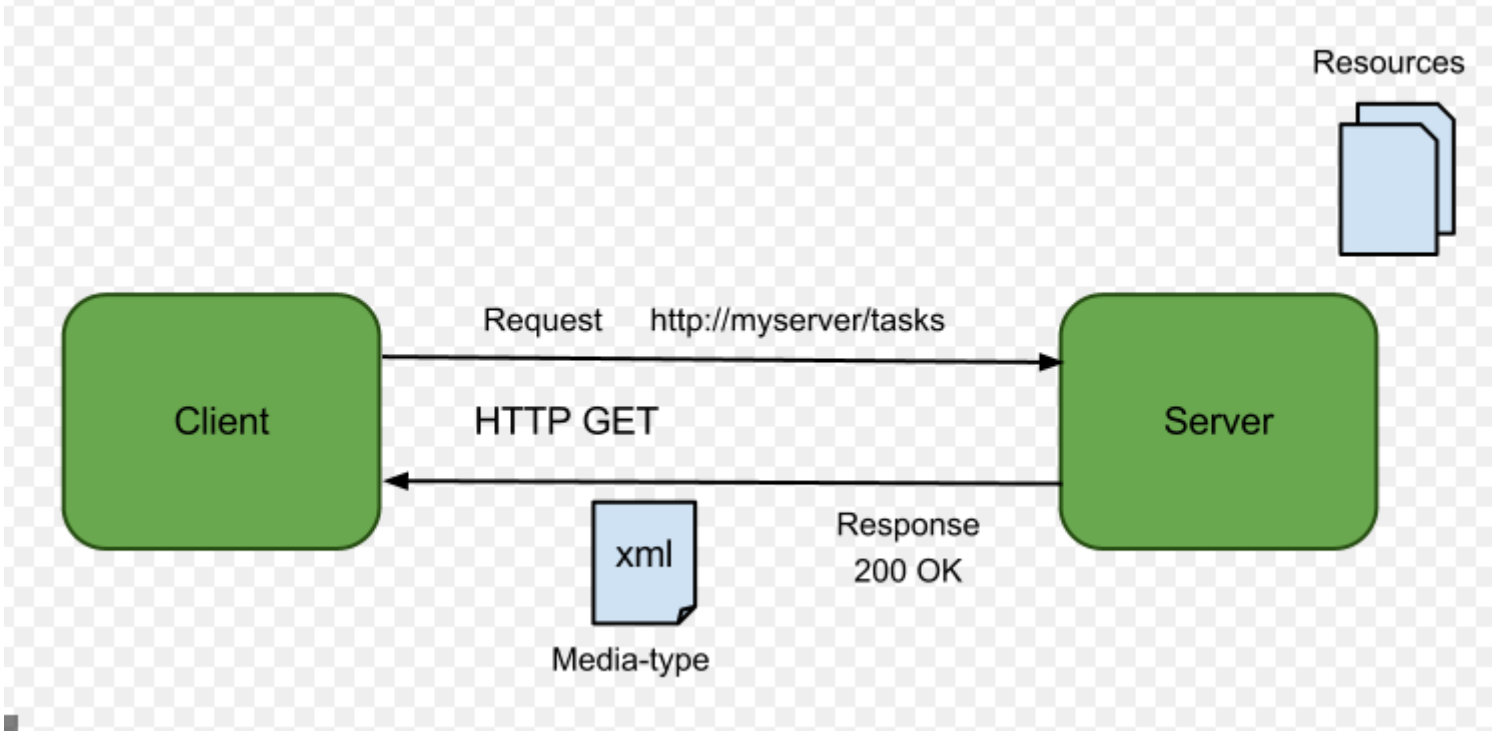
SOAP permite adicionar capas de seguridad para casos particulares en algunas empresas.

REST vs SOAP

SOAP es recomendado si se necesitan transacciones ACID (Atomicity, Consistency, Isolation and Durability) sobre el servicio. En Internet este caso es inusual.

SOAP tiene incluída una lógica para garantizar éxito y reprocesamiento y proporcionar confianza en una transacción.

REST



Toolbox

wsimport es una herramienta que a partir de un recurso WSDL (Web Services Description Language) permite generar los ficheos JAX-WS requeridos (fuentes y clases compiladas)

wsimport -keep -verbose http://www.webservicex.com/globalweather.asmx?wsdl

keep – mantener los fuentes
verbose – mostrar todo lo que está haciendo el compilador

http://www.thomas-bayer.com/axis2/services/BLZService?wsdl

Toolbox

xjc es una herramienta que toma un schema XML como entrada y genera un paquete de clases de Java que reflejan las reglas definidas en el schema.

xjc -p primer.po -d src po.xsd

Esto se puede lograr en código utilizando frameworks de trabajo parecidos como el JAXB o el XStream.

Toolbox

SoapUI permite realizar test funcionales a los servicios Web.

Se pueden definir proyectos de pruebas de apis REST o SOAP, test cases y ejecutarlos desde un ambiente gráfico

Una de las pocas herramientas que importa defiiciones Swagger (framework para especificación de servicios REST)

Tiene la posibilidad de exportar war por si desean utilizar esas en un servidor standalone.

Toolbox

HATEOAS (Hypermedia As The Engine Of Application State) es una condición de los servicios REST que permite navegar por los servicios sin necesidad de conocer más que los enlaces.

No se necesita tener montado linux o una máquina virtual de linux para utilizar la mayoría de sus comandos, puede:

Utilizar conexiones remota por ssh, scp o teamviewer

Instalar Cygwin o MSYS/MinGW para tener un subconjunto bastante completo de las órdenes de linux.

Toolbox

Algunos utilitarios linux muy recomendados son

curl (navegación y descarga de recursos online)

jq (procesamiento json)

cvskit (módulo python para consultas SQL sobre CSV)

awk (procesamiento de ficheros por línea)

xidel, xmllint, xquilla (consultas xquery)

Ejercicios

Crear un API de 4 servicios Web que defina las 4 acciones fundamentales sobre una tabla de una bbdd relacional (CRUD)

La tabla debe tener un campo id único para referencia e índice (no importa el motor utilizado: MySQL, SQLLite, PostgreSQL)

Sitio online sugerido: freesqldatabase

Ejercicios

Crear un API de servicios Web que nos permita hacer consultas sobre una o varias páginas web (scraping – ver ejemplo de scraping en GitHub público) según términos que deseemos buscar.

Los servicios deben retornar un segmento del código HTML en el que se encuentre insertado el elemento buscado.

Spring AOP

La programación orientada a aspectos (AOP - Aspect Oriented Programming) es un paradigma de programación que intenta formalizar y representar de forma concisa los elementos que son transversales a todo el sistema.

En los lenguajes orientados a objetos, la estructura del sistema se basa en la idea de clases y jerarquías de clases. La herencia permite modularizar el sistema, eliminando la necesidad de duplicar código. No obstante, siempre hay aspectos que son transversales a esta estructura.

Spring AOP

La programación orientada a aspectos intenta formular conceptos y diseñar construcciones del lenguaje que permitan modelar estos aspectos transversales sin duplicación de código.

En AOP, a los elementos que son transversales a la estructura del sistema y se pueden modularizar gracias a las construcciones que aporta el paradigma se les denomina aspectos (aspects).

Spring AOP

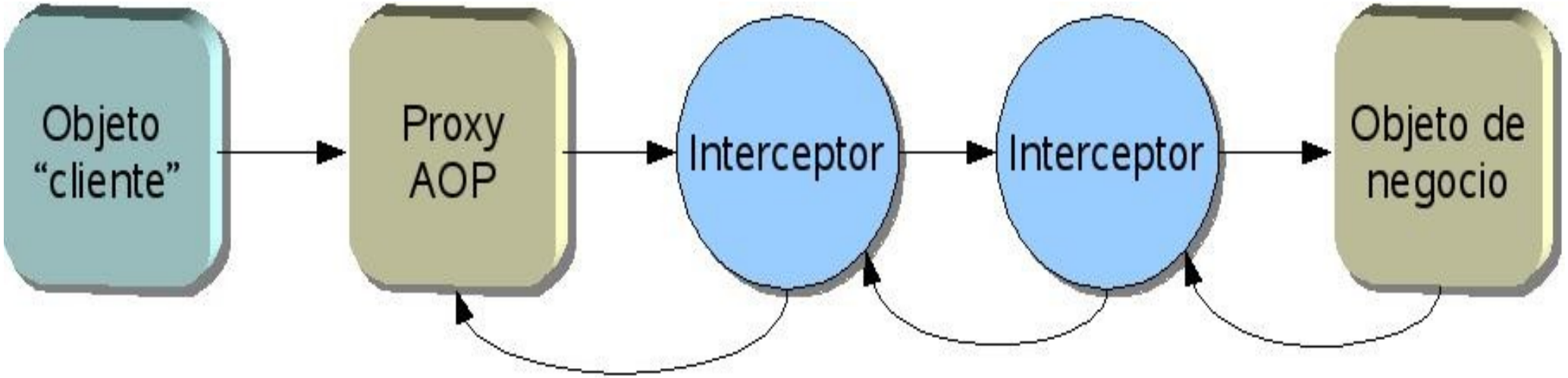
Un consejo (advice) es una acción que hay que ejecutar en determinado/s punto/s de un código, para conseguir implementar un aspecto.

El conjunto de puntos del código donde se debe ejecutar un advice se conoce como punto de corte o pointcut.

En Spring, el objeto que debe ejecutar esta acción se modela en la mayoría de casos como un interceptor: un objeto que recibe una llamada a un método propio antes de que se ejecute ese punto del código.

Spring AOP

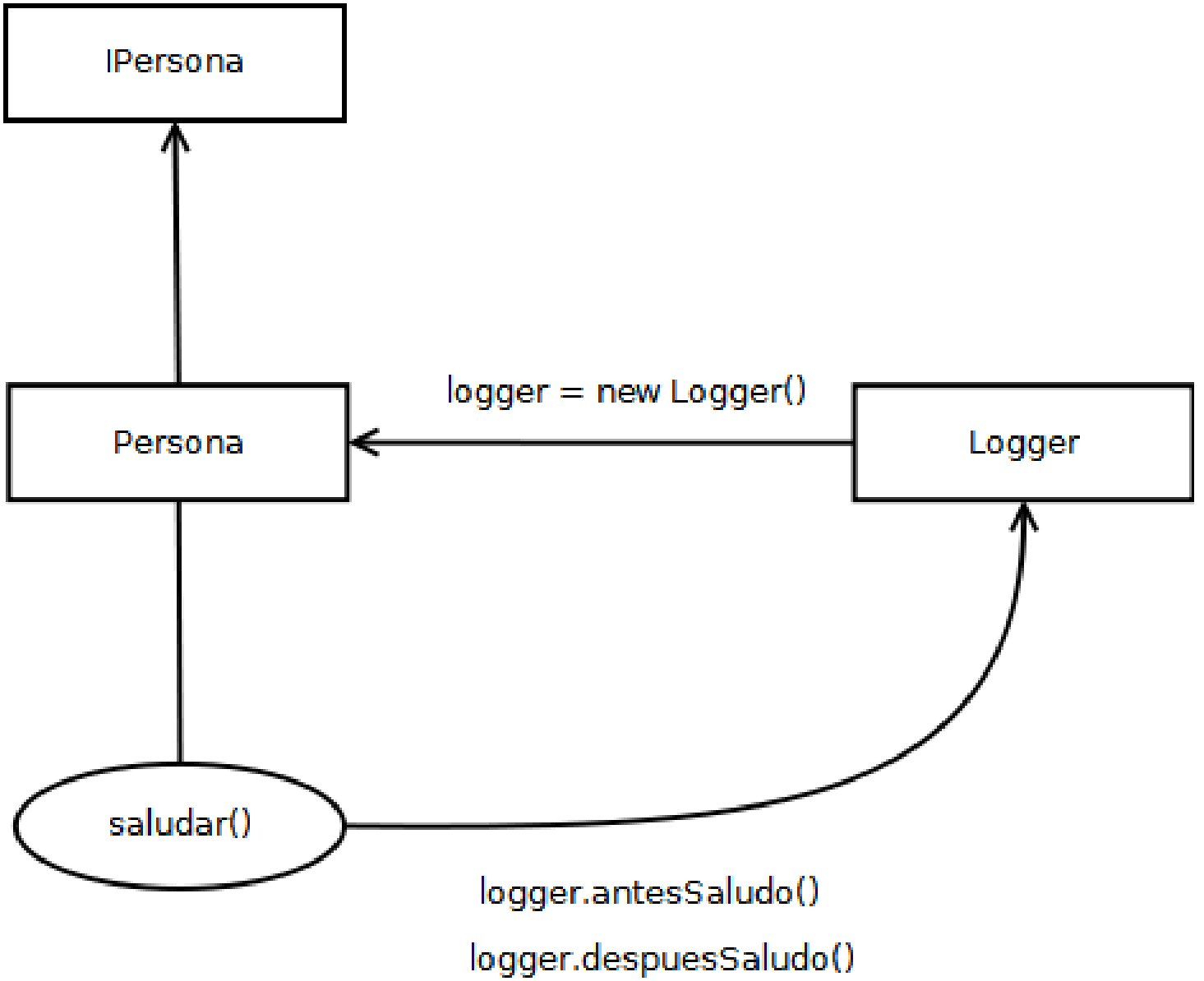
Interceptor



Cuando algún objeto llama a un método que forma parte del pointcut, el Spring AOP logra que en realidad se llame a un objeto proxy o intermediario, que tiene un método con el mismo nombre y signatura pero cuya ejecución lo que hace en realidad es redirigir la llamada por una cadena de interceptores hasta el método que se quería ejecutar.

Spring AOP

Logging sin AOP



Spring AOP

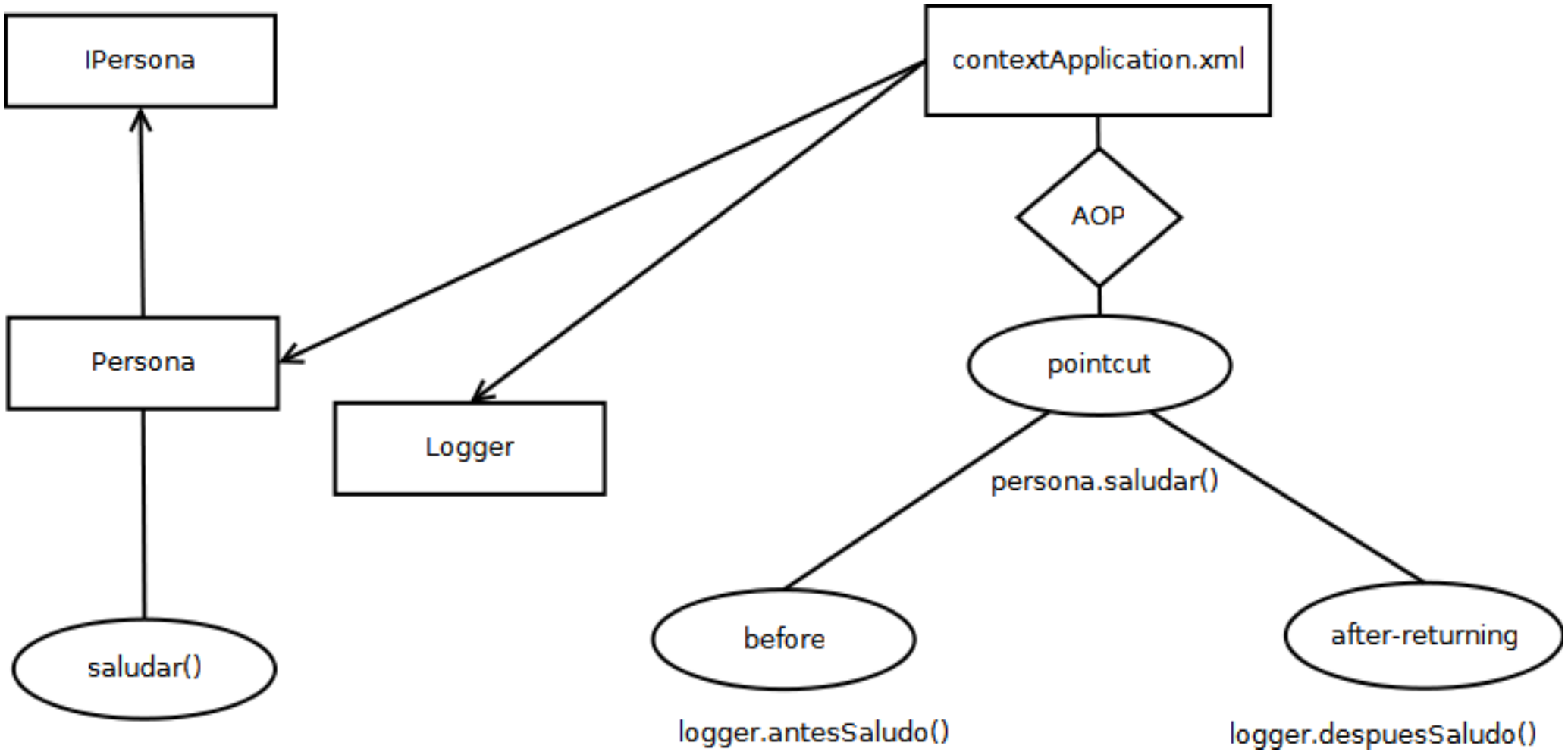
Logging sin AOP (Dependencia y repetición)

IPersona es la interfaz de la clase Persona.
Al crear una instancia de Persona creamos también un nuevo Logger.
Al ejecutar el método saludar() de Persona:

- Primero llamamos al método de logger antesSaludo() para apuntar en el log que se va a saludar.
- Luego mostramos un mensaje por pantalla : “Hola que tal”.
- Finalmente llamamos al método de logger despuesSaludo() para apuntar que ya se ha saludado correctamente.

Spring AOP

Logging con AOP



Spring AOP

Con Annotations

```
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class EjemploDeAspecto {

  //aquí vendrían los advices...
}
```

Spring AOP

Con Annotations

```
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class EjemploDeAspecto {

  //aquí vendrían los advices...
}
```

Spring AOP

@Before
Esta anotación ejecuta un advice antes de la ejecución del punto de corte especificado.

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class EjemploBefore {
    @Before("execution(public * get*())")
    public void controlaPermisos() {
    // ...
    }
}
```

Spring AOP

@AfterReturning
Esta anotación ejecuta un advice después de la ejecución del punto de corte especificado en normalidad

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.AfterReturning;

@Aspect
public class EjemploAfterReturning {
    @AfterReturning("execution(public * get*())")
    public void log() {
    // ...
    }
}
```

Spring AOP

@AfterReturning
En el caso del logging nos interesa saber el valor retornado por el punto de corte

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.AfterReturning;

@Aspect
public class EjemploAfterReturning {
	@AfterReturning(
	pointcut="execution(public * get*())", returning="valor")
	public void log(Object valor) {
	// ...
	}
}
```

Spring AOP

@AfterThrowing
Esta anotación ejecuta un advice después de la ejecución del poincut si genera una excepción

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.AfterThrowing;

@Aspect
public class EjemploAfterThrowing {
	@AfterThrowing(
	pointcut="execution(public * get*())", throwing="daoe")
	public void logException(DAOException daoe) {
	// ...
	}
}
```

Spring AOP

@After
Esta anotación ejecuta un advice después de la ejecución del punto de corte especificado, genere o no una excepción.

@Around
Esta anotación ejecuta parte del advice antes y parte después de la ejecución del punto de corte especificado. La filosofía consiste en que el usuario es el que debe especificar en el código del advice en qué momento se debe llamar al punto de corte.

Spring AOP

@Around

```
@Aspect
public class EjemploAround {

  @Around("execution(public * get*())")
  public Object ejemploAround(ProceedingJoinPoint  pjp)
        throws Throwable {
        System.out.println("ANTES");
        Object valorRetorno = pjp.proceed();
        System.out.println("DESPUES");
        return valorRetorno;
        }
}
```

Spring AOP

Para hacer accesibles al advice los argumentos del punto de corte se puede usar args. Por ejemplo:

```
//...
@AfterReturning("execution(public void set*(*)) &&
args(nuevoValor)")
public void log(int nuevoValor) {
    // ...
}
//...
```