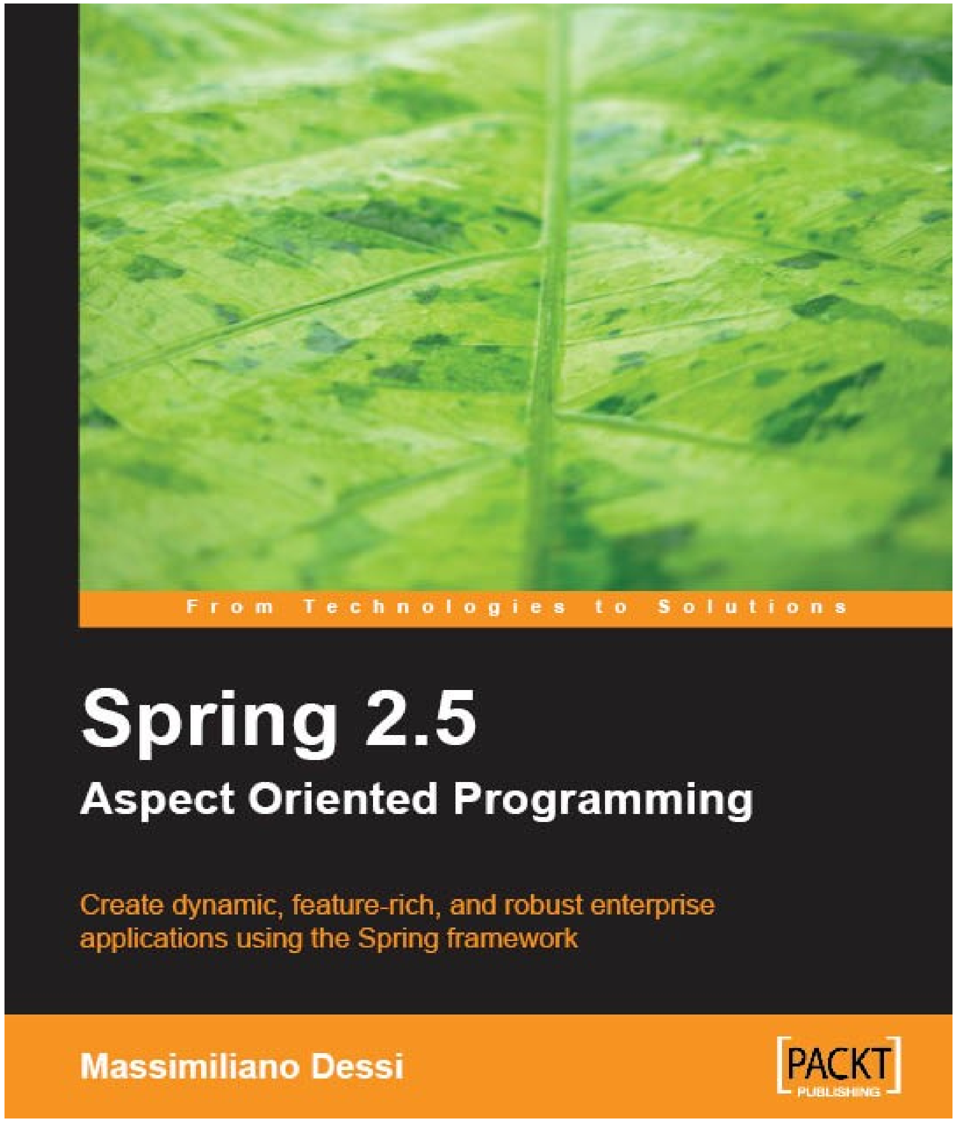

From Technologies to Solutions
Spring 2.5
Aspect Oriented Programming
Create dynamic, feature-rich, and robust enterprise applications using the Spring framework
Massimiliano Dessi
PACKT PUBLISHING

Aspect Oriented Programming and MVC with Spring Framework

Massimiliano Dessì - desmax74@yahoo.it
Lugano - 29/01/2009

Author

Java Architect 2008

Co-founder

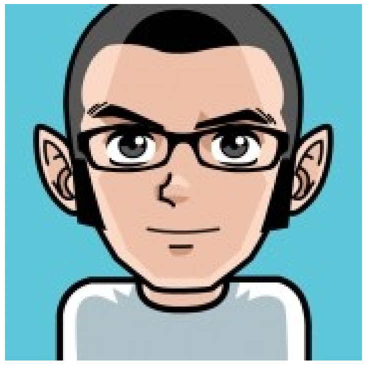
JugSardegna Onlus 2003

Founder:

SpringFramework Italian User Group 2006

Jetspeed Italian User Group 2003

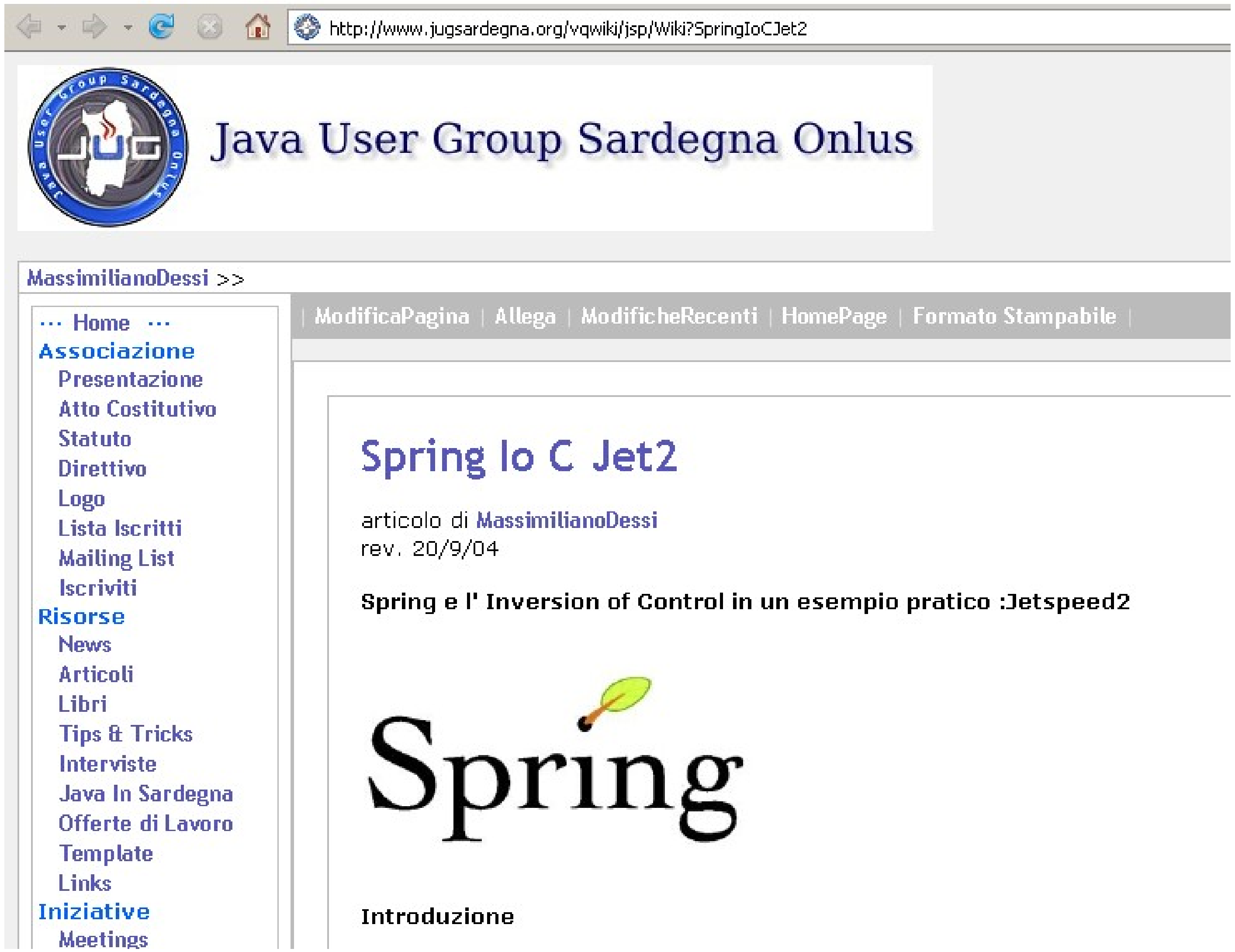
Groovy Italian User Group 2007



Spring Knowledge

Spring User
since July 2004
(Spring 1.1)

First article in
Italy,
september 2004
on JugSardegna.



http://www.jugsardegna.org/vqwiki/jsp/Wiki?SpringIoCJet2

Java User Group Sardegna Onlus

MassimilianoDessi >>

| ModificaPagina | Allega | ModificheRecenti | HomePage | Formato Stampabile |

··· Home ···
Associazione
Presentazione
Atto Costitutivo
Statuto
Direttivo
Logo
Lista Iscritti
Mailing List
Iscriviti
Risorse
News
Articoli
Libri
Tips & Tricks
Interviste
Java In Sardegna
Offerte di Lavoro
Template
Links
Iniziative
Meetings

Spring Io C Jet2

articolo di MassimilianoDessi
rev. 20/9/04

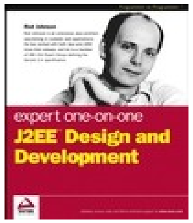
Spring e l' Inversion of Control in un esempio pratico :Jetspeed2

Spring

Introduzione

Spring origin

2003: Spring is a layered J2EE application framework based on code published in
"Expert One-on-One J2EE Design and Development" by Rod Johnson."



Spring core

The core of the Spring Framework is based on the principle of Inversion of Control (IoC).

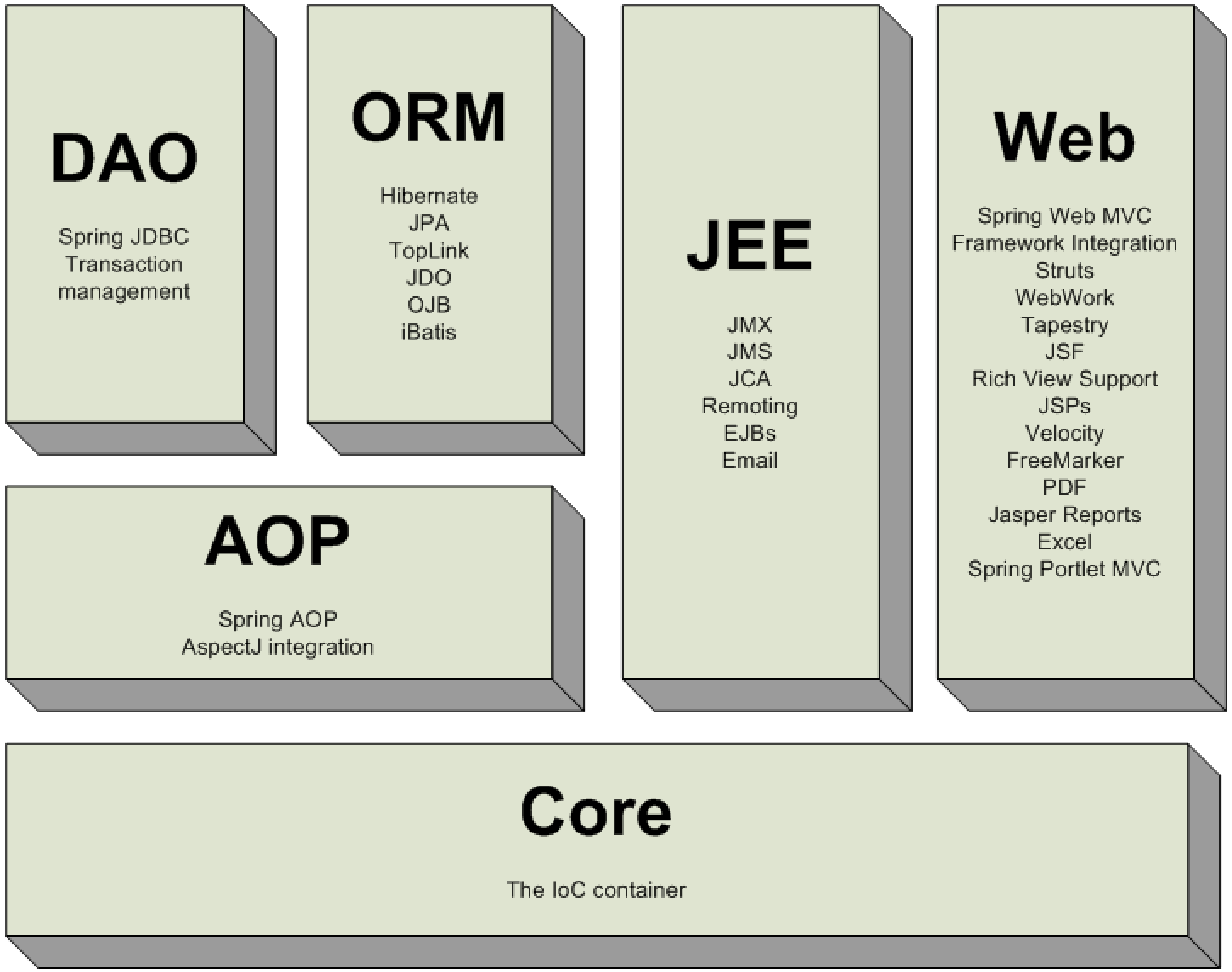
Applications that follow the IoC principle use configuration that describes the dependencies between its components.

It is then up to the IoC framework to satisfy the configured dependencies.

Ioc, also known as the Hollywood Principle - "Don't call us, we'll call you"

http://martinfowler.com/bliki/InversionOfControl.html

Spring overview



Spring XML configuration

```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:p="http://www.springframework.org/schema/p"
xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:component-scan base-package="it.freshfruits" />
<context:property-placeholder location="WEB-INF/conf/spring/config.properties" />

<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"
       p:viewClass="org.springframework.web.servlet.view.JstlView" p:prefix="WEB-INF/jsp/" p:suffix=".jsp"/>

<bean name="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean" p:jndiName="java:comp/env/jdbc/sffs"/>

<bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean" p:dataSource-ref="dataSource" p
       :configLocation="WEB-INF/conf/ibatis/sffs-sqlMapConfig.xml"/>

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager" p:dataSource-ref="dataSource"/>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
       <tx:attributes>
              <tx:method name="save*" propagation="REQUIRED" rollback-for="Exception"/>
              <tx:method name="insertOrder" propagation="REQUIRED" rollback-for="OrderItemsException"/>
              <tx:method name="insert*" propagation="REQUIRED" rollback-for="DataAccessException"/>
              <tx:method name="update*" propagation="REQUIRED" rollback-for="DataAccessException"/>
              <tx:method name="delete*" propagation="REQUIRED" rollback-for="DataAccessException"/>
              <tx:method name="disable*" propagation="REQUIRED" rollback-for="DataAccessException"/>
              <tx:method name="*" read-only="true"/>
       </tx:attributes>
</tx:advice>

<aop:config>
       <aop:pointcut id="repoOperations" expression="execution(* it.freshfruits.application.repository.*.*(..))" />
       <aop:advisor advice-ref="txAdvice" pointcut-ref="repoOperations"/>
</aop:config>

</beans>
```

AOP

The Aspect Oriented Programming supports the Object Oriented Programming in the implementation phase, wher it presents weak spots.

The AOP is complementary to OOP in the implementation phase.

The AOP must be used with

wisdom.

OOP limits

Code scattering, when a feature is implemented in different modules.

Two types:

- Blocks of duplicated code (for instance identical implementation of an interface in different classes)
- Complementary blocks of code of the same feature, located in different modules.

  (for instance in an ACL, a module executes the authentication and another one the authorization)

OOP limits

Code tangling: a module has too many tasks at the same time.

```
public ModelAndView list(HttpServletRequest req, HttpServletResponse res)
    throws Exception {

    log(req);                                           //logging

    if(req.isUserInRole("admin")){                      // authorization

        List users ;
        try {                                           //exception handling
    String username = req.getRemoteUser();
    users =cache.get(Integer.valueOf(conf.getValue("numberOfUsers")),
            username);                                  //cache with authorization
    } catch (Exception e) {
        users = usersManager.getUsers();
    }

        return new ModelAndView("usersTemplate", "users", users);

    }else{
      return new ModelAndView("notAllowed");
    }
}
```

Consequences

-Difficult evolution

-Poor quality

-Code not reusable

-Traceability

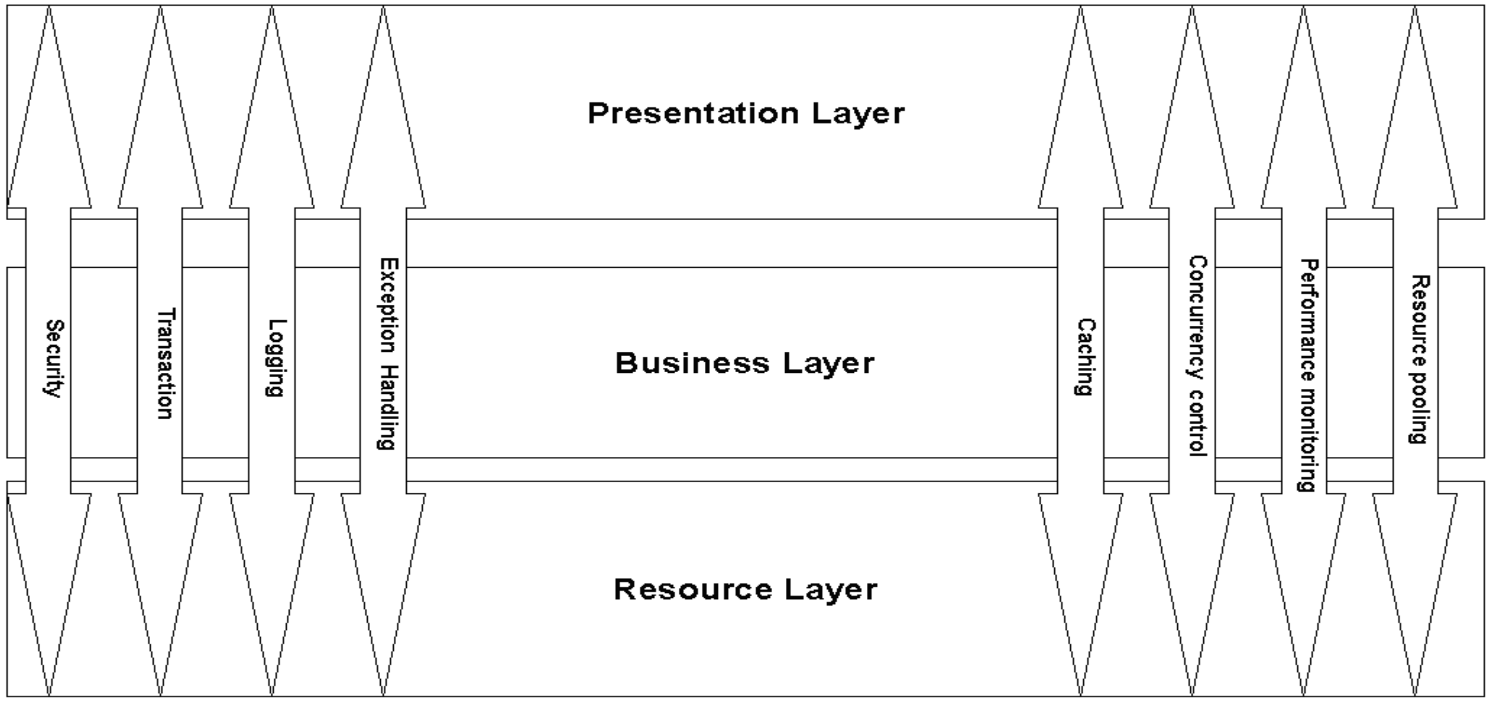
-Low productivity

Moreover, it's written in an Object Oriented language:

the implementation, not the language, is the problem !

The source of the problem

The classes have to implement transversal features.



The problem now is not a class that must have only one task/aim/responsibility/feature, the problem now is how that task is called to be used.

AOP Solution

The AOP provides the constructs and tools to centralize these features that cross the application transversal (crosscutting concerns).
Thus, the AOP permits to centralise the duplicated code and apply it according to predetermined rules in the requested places, during the execution flow.

AOP elements

Aspect: it corresponds to the class in the OOP; it contains the transversal feature (crosscutting concern).

Joinpoint: point of execution of the code
(for instance a constructor's invocation, a method's execution or the management of an Exception).

Advice: the action to be performed in the joinpoint.

Pointcut: it contains the expression that permits to locate the joinpoint to which we want to apply the advice.

AOP elements

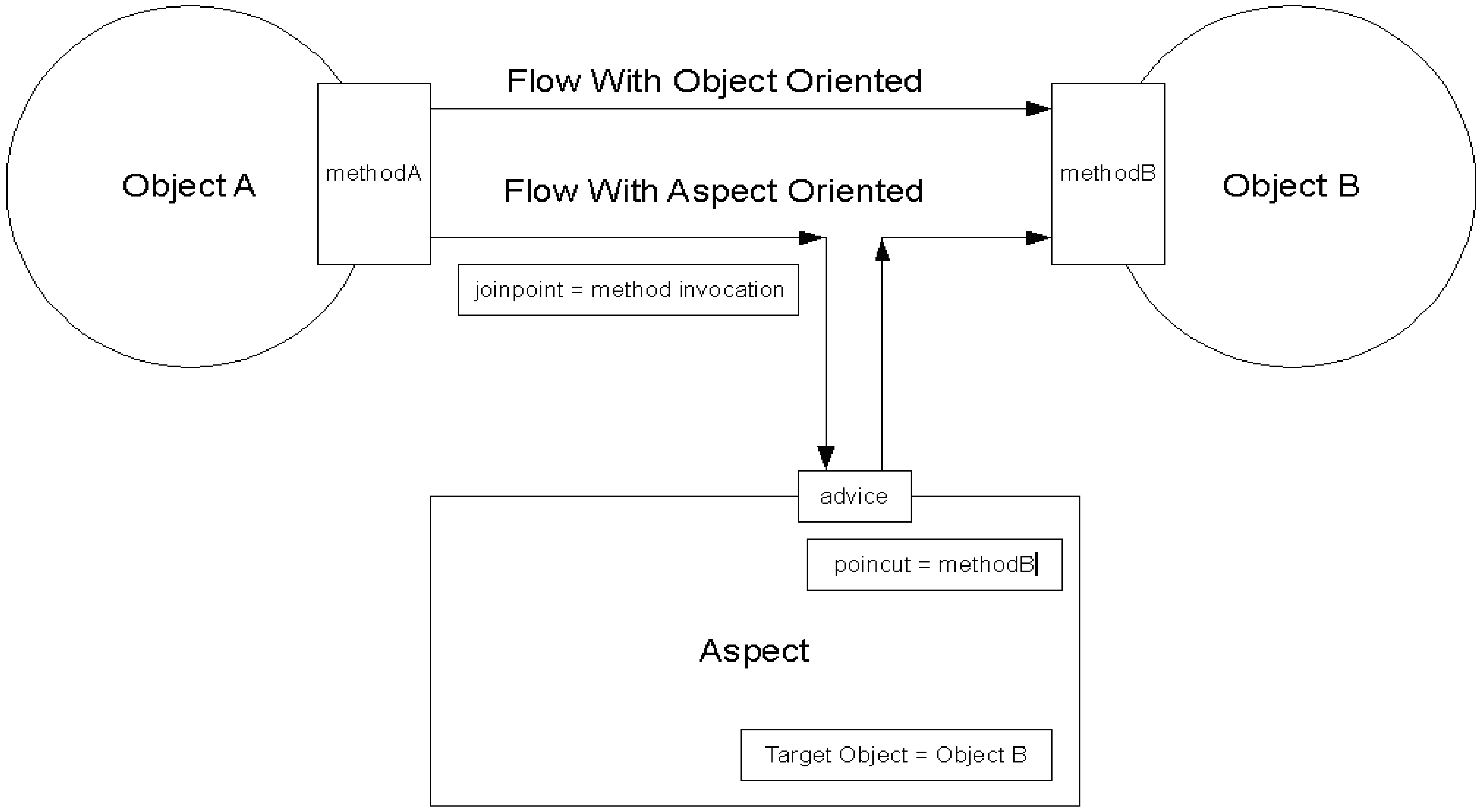
Introduction: it allows to add interfaces and the implementation in runtime to predetermined objects.

Target: class on which the aspect works with the advice

Weaving: This is the linking action between the aspect and the objects to which advices must be applied.

AOP diagram

How the AOP works



Spring AOP

To allow a simplified use of AOP, Spring uses the execution of the methods as joinpoint.

This means that we can act before, after, around a method, at the raise of an exception, whatever the result may be (finally).

We use normal Java classes with annotations or XML.

Who works in the shadow to allow this simplicity?

`java.lang.reflect.Proxy` or CGLIB

Spring AOP: take it easy

We only have to decide:

-what to centralise ?
-when does it have to come into action ?

What I can obtain:

-support to the Domain Driven Design
-less code
-code that is manageable and can be maintained
-less excuses to use really the OOP, and not to move data containers...

Spring AOP - Concurrence

```
@Aspect() @Order(0)
public class ConcurrentAspect {

  private final ReadWriteLock lock =
    new ReentrantReadWriteLock();
  private final Lock rLock = lock.readLock();
  private final Lock wLock = lock.writeLock();

  @Pointcut("execution (* isAvailable(..))")
  private void isAvailable() {}

  @Pointcut("execution (* retainItem(..))")
  private void retainItem() {}

  @Pointcut("execution (* release(..))")
  private void release() {}

  @Pointcut("release() || retainItem()")
  private void releaseOrRetain() {}
```

First part of the class:

- I create some locks for reading and writing

- I put @Aspect on the class

- I define with the annotation @Pointcut the expressions that indicate when the Aspect has to operate.

Spring AOP - Concurrence

```
    @Before("isAvailable()")
    public void setReadLock() {
        rLock.lock();
    }

    @After("isAvailable()")
    public void releaseReadLock() {
        rLock.unlock();
    }

    @Before("releaseOrRetain()")
    public void setWriteLock() {
        wLock.lock();
    }

    @After("releaseOrRetain()")
    public void releaseWriteLock() {
        wLock.unlock();
    }
}
```

Second part of the class:

I define reusing the names of the methods on which I've annotated the @Pointcut the logic to execute with locks

With the following annotations I declare when I want it to be executed

```
@Before
@After
@AfterReturning
@Around
@AfterThrowing
```

Spring AOP – Advice Type

```
@Before

@After

@AfterReturning

@Around

@AfterThrowing
```

Advices let us not only execute logic in the points defined by pointcuts, but even to obtain informations about execution (target class, method called, argument passed, return value)
With some kind of advices (around) even to have control on the execution flow.
The calling class of course doesn't know anything about what happens, it just sees a simple class ....

Spring AOP – Pointcut

- `execution`
- `within`
- `this`
- `target`
- `args`
- `@target`
- `@args`
- `@within`
- `@annotation`
- `bean`

In the previous pointcuts we've seen how the expression to define the application point was the execution of a method with a certain name.

We have at our disposal other pointcut designators as well to have the maximum control.

Spring AOP – JMX

```
@ManagedResource("freshfruitstore:type=TimeExecutionManagedAspect")
@Aspect() @Order(2)
public class TimeExecutionManagedAspect {

  @ManagedAttribute
  public long getAverageCallTime() {
    return (this.callCount > 0
     ? this.accumulatedCallTime / this.callCount : 0);
  }

  @ManagedOperation
  public void resetCounters() {
     this.callCount = 0;
     this.accumulatedCallTime = 0;
  }

  @ManagedAttribute
  public long getAverageCallTime() {
    return (this.callCount > 0 ?
     this.accumulatedCallTime / this.callCount : 0);
  }

    ...
```

besides being able to act in the execution flow, I can manage the Aspect with JMX as well,exposing as attributes or operations, the normal methods of Aspect which is always a Java class

Spring AOP – JMX

```
...

@Around("within(it.mypackage.service.*Impl)")
public Object invoke(ProceedingJoinPoint joinPoint)
     throws Throwable {

   if (this.isTimeExecutionEnabled) {
      StopWatch sw = new StopWatch(joinPoint.toString());
      sw.start("invoke");
      try {
          return joinPoint.proceed();
      } finally {
          sw.stop();
          synchronized (this) {
             this.accumulatedCallTime += sw.getTotalTimeMillis();
          }
             logger.info(sw.prettyPrint());
          }
      } else {
          return joinPoint.proceed();
      }
    }
...
```

Using JMX

I can change

the behaviour of

the Aspect at runtime

Spring AOP – Introductions

An introduction allows to decorate an object with interfaces and its implementation.
That allows us both to avoid the duplication of an implementation, and to simulate the multiple inheritance that Java doesn't have.

```
@Aspect
public class ParallelepipedIntroduction {

    @DeclareParents(value = "org.springaop.chapter.four.introduction.Box",
        defaultImpl = Titanium.class)
    public Matter matter;

    @DeclareParents(value = "org.springaop.chapter.four.introduction.Box",
        defaultImpl = Cube.class)
    public GeometricForm geometricForm;
}
```

SpringAOP - @Aspect

The annotations we've seen so far included pointcuts syntax are provided by AspectJ.

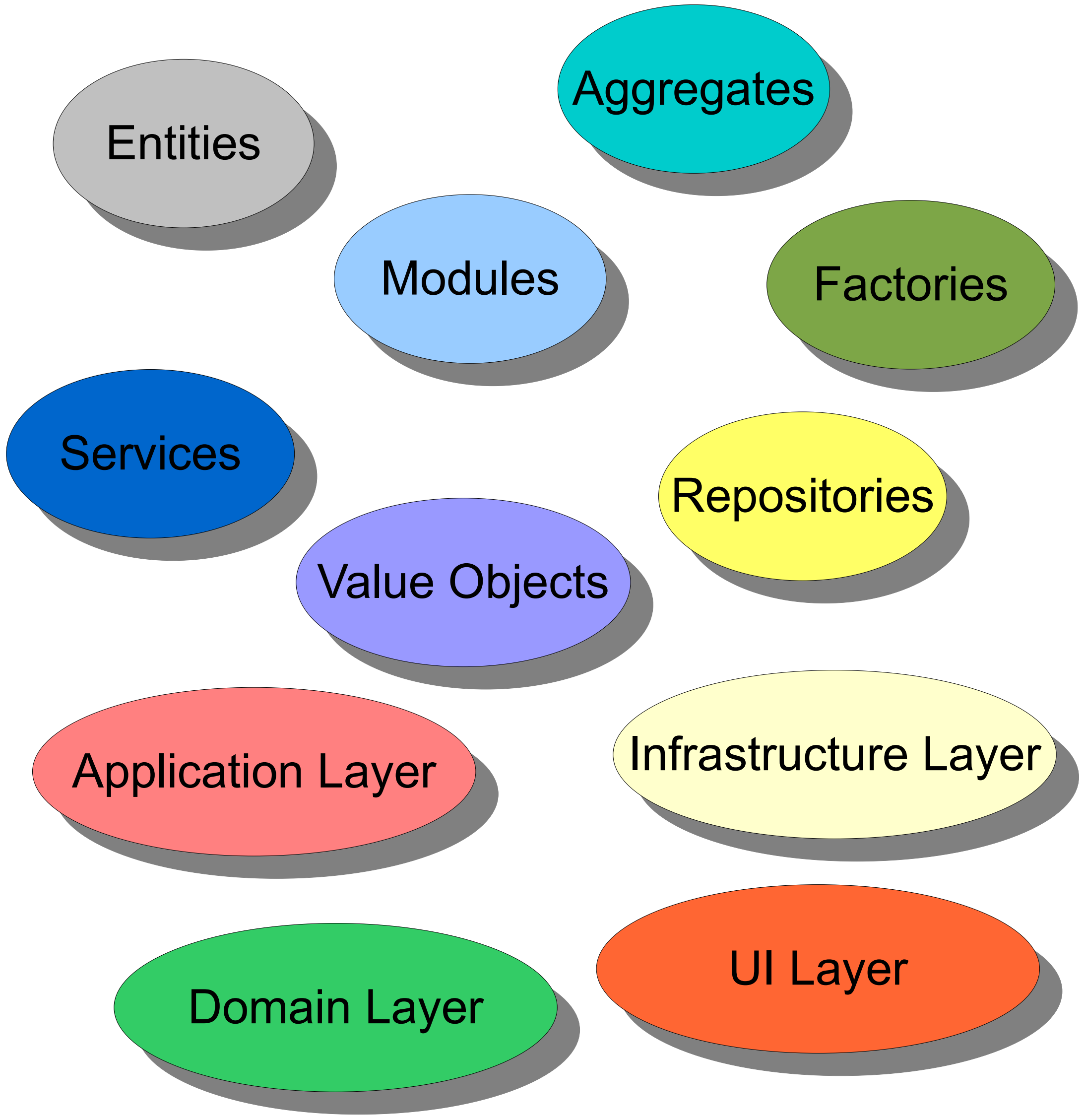
```
import org.aspectj.lang.annotation.*
```

but they're completely inside Spring's “context” and on Spring's beans.

Now let's look at what we can use of Sring (IoC) outside Spring's “context” through AspectJ.

We'll have to accept some compromise...

DDD



Domain-Driven Design is a way of thinking applications.
Suggesting to focus the attention on the problem domain, inviting to think by ojects and not with procedural style design.

DDD

```
public interface Customer extends NamedEntity {

    public Address getAddress();

    public ContactInformation getContact();

    public void modifyContactInformation(ContactInformation contact);

    public void modifyAddress(Address address);

    public Boolean saveCustomer();

    public Boolean createOrder();

    public Boolean saveOrder();

    public Order getOrder();

    public List<Order> getOrders();
}
```

In the entities it is concentrated the business logic, not in the services that execute “procedures”...

In this way our objects have data and behaviours

SpringAOP +DDD +AspectJ

```
@Configurable(dependencyCheck = true, autowire=Autowire.BY_TYPE)
public class CustomerImpl implements Customer, Serializable {

     @Autowired
     public void setCustomerRepository(@Qualifier("customerRepository") CustomerRepository customerRepo) {
          this.customerRepository = customerRepository;
     }

     @Autowired
     public void setOrderRepository(@Qualifier("orderRepository") OrderRepository orderRepo) {
          this.orderRepository = orderRepo;
     }


     public Boolean createOrder() {
          Boolean result = false;
          if (order == null) {
               order = new OrderImpl.Builder(Constants.ID_NEW, new Date(), id
                         .toString()).build();
               result = true;
          }
          return result;
     }

     public Boolean saveCustomer() {
          return customerRepository.saveCustomer(this);
     }

...
```

SpringAOP +AspectJ

With @Configurable we have declared that some dependencies will e injected to the class even if it isn't a Spring bean.

Spring applicationContext knows the class just as a bean prototype.

In order to have this feature we need to pass to the JVM the Spring's jar to use for the Load Time Weaving:

-javaagent:<path>spring-agent.jar

or to configure tomcat in this way to do it in our stead :

```
<Loader
loaderClass="org.springframework.instrument.classloading.tomcat.TomcatInstrumentableClassLoader"
     useSystemClassLoaderAsParent="false"/>
```

Spring AOP +AspectJ Weaver

Using AspectJ Weaver implies a different approach compared with SpringAOP's simplicity see so far.

Annotating classes as @Aspect and making create them by Spring, means staying anyway into the IoC Container, like defining pointcuts on methods executions by beans of Spring.

With LTW we're telling to Spring to work outside its “context, to inject dependencies on objects not created by the IoC container.

In order to use LTW we define in a file for AspectJ the classes on which it has to operate and which Aspects it has to create, Aspects that are not beans created by Spring...

LTW - aop.xml

100% AspectJ

```
<!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN"
"http://www.eclipse.org/aspectj/dtd/aspectj.dtd">
<aspectj>
<weaver options="-showWeaveInfo
-XmessageHandlerClass:org.springframework.aop.aspectj.AspectJWeaverMessageHandler">
     <!-- only weave classes in our application-specific packages -->
     <include within="it.freshfruits.domain.entity.*"/>
     <include within="it.freshfruits.domain.factory.*"/>
     <include within="it.freshfruits.domain.service.*"/>
     <include within="it.freshfruits.domain.vo.*"/>
     <include within="it.freshfruits.application.repository.*"/>
     <exclude within="it.freshfruits.aspect.*"/>
</weaver>
<aspects>
        <aspect name="it.freshfruits.aspect.ConcurrentAspect" />
        <aspect name="it.freshfruits.aspect.LogManagedAspect" />
        <aspect name="it.freshfruits.aspect.TimeExecutionManagedAspect" />
</aspects>
</aspectj>
```

Spring AOP +AspectJ Weaver

Once we have the dependencies in an entity of the domain, both with the injection of dependencies with LTW or pass dependencies as constructor arguments by a factory, we see the result on the User Interface

DDD UI

```
@Controller("customerController")
public class CustomerController {

    @RequestMapping("/customer.create.page")
    public ModelAndView create(HttpServletRequest req) {
        return new ModelAndView("customer/create", "result", UiUtils.getCustomer(req).createOrder());
    }

    @RequestMapping("/customer.order.page")
    public ModelAndView order(HttpServletRequest req) {
        return new ModelAndView("customer/order", "order", UiUtils.getOrder(req));
    }

    @RequestMapping("/customer.items.page")
    public ModelAndView items(HttpServletRequest req) {
        return new ModelAndView("customer/items", "items", UiUtils.getOrder(req).getOrderItems());
    }
...
}
```

The controllers will be completely stateless and without dependencies, and with a simple call on the entity.

DDD UI

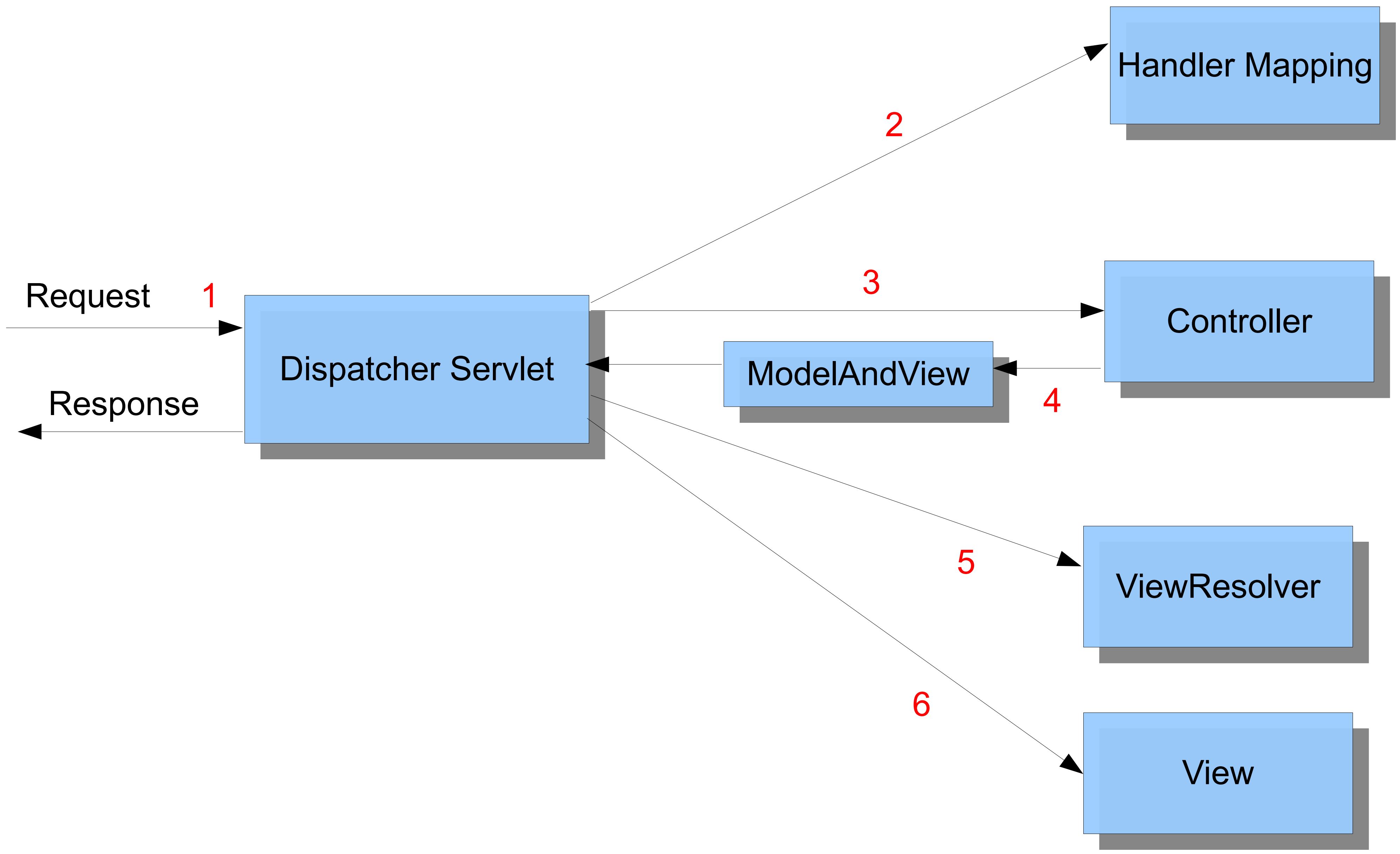
Handler Interceptor which puts int he HttpServletRequest the entity customer used by the Customer Controller

```
public class CustomerInterceptor extends HandlerInterceptorAdapter {

    @Override
    public boolean preHandle(HttpServletRequest req, HttpServletResponse res, Object handler) throws
Exception {
        req.setAttribute(Constants.CUSTOMER, customerFactory.getCurrentCustomer());
        return true;
    }

    @Autowired
    private CustomerFactory customerFactory;
}
```

Benefits

OOP + AOP + DDD
=
clean code
elegant code

Spring MVC


Request
1
Dispatcher Servlet
Response
2
Handler Mapping
3
Controller
ModelAndView
4
5
ViewResolver
6
View

Form Controller

```
@Controller("fruitController") @RequestMapping("/fruit.edit.admin")
public class FruitController {

    @RequestMapping(method = RequestMethod.POST)
    public String processSubmit(@ModelAttribute("fruit") FruitMap fruit, BindingResult result,
                SessionStatus status) {

        validator.validate(fruit, result);
        if (result.hasErrors()) {
            return "userForm";
        } else {
            fruit.save();
            status.setComplete();
            return "redirect:role.list.admin";
        }
    }

    @InitBinder()
    public void initBinder(WebDataBinder binder) throws Exception {
        binder.registerCustomEditor(String.class, new StringTrimmerEditor(false));
    }

    @RequestMapping(method = RequestMethod.GET)
    public String setupForm(@RequestParam(required = false, value = "id") Integer id, ModelMap model) {
        model.addAttribute(Constants.FRUIT, id == null ? new FruitMap() : fruitRepository.getFruitType(id));
        return "role/form";
    }

    @Autowired @Qualifier("fruitRepository")
    private FruitTypeRepository fruitRepository;
    @Autowired @Qualifier("fruitValidator")
    FruitValidator validator;
}
```

MultiAction Controller

```java
@Controller("customerController")
public class CustomerController {

    @RequestMapping("/customer.create.page")
    public ModelAndView create(HttpServletRequest req) {
        return new ModelAndView("customer/create", "result", UiUtils.getCustomer(req).createOrder());
    }

    @RequestMapping("/customer.save.page")
    public ModelAndView save(HttpServletRequest req) {
        return new ModelAndView("customer/save", "result", UiUtils.getCustomer(req).saveOrder());
    }

    @RequestMapping("/customer.show.page")
    public ModelAndView show(HttpServletRequest req) {
        return new ModelAndView("customer/show", "customer", UiUtils.getCustomer(req));
    }

    @RequestMapping("/customer.order.page")
    public ModelAndView order(HttpServletRequest req) {
        return new ModelAndView("customer/order", "order", UiUtils.getOrder(req));
    }

    @RequestMapping("/customer.items.page")
    public ModelAndView items(HttpServletRequest req) {
        return new ModelAndView("customer/items", "items", UiUtils.getOrder(req).getOrderItems());
    }

    @RequestMapping("/customer.remove.page")
    public ModelAndView remove(@RequestParam("id") String id, HttpServletRequest req) throws Exception {
        Order order = UiUtils.getOrder(req);
        return order.removeOrderItem(order.getId().toString(), id) ? new ModelAndView("customer/items",
           "items", order.getOrderItems()) : new ModelAndView("customer/remove", "result", false);
    }
}
```

Configuration

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:component-scan base-package="it.freshfruits.ui"/>
<context:annotation-config/>

<bean name="urlMapping" class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
      <property name="interceptors">
            <list>
                  <ref bean="customerInterceptor"/>
             </list>
       </property>
</bean>

<bean name="customerInterceptor" class="it.freshfruits.ui.interceptor.CustomerInterceptor"/>

</beans>
```

Spring Security

```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:p="http://www.springframework.org/schema/p" xmlns:sec="http://www.springframework.org/schema/security"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-2.0.4.xsd">

        <sec:http>
                <sec:intercept-url pattern="/log*.jsp" filters="none" />
                <sec:intercept-url pattern="/*.page" access="ROLE_USER" />
                <sec:intercept-url pattern="/*.admin" access="ROLE_ADMIN" />
                <sec:form-login login-page="/login.jsp" default-target-url="/" login-processing-url="/j_security_check"
                        authentication-failure-url="/loginError.jsp" />
                <sec:logout logout-url="/logout.jsp" invalidate-session="true" logout-success-url="/login.jsp" />
                <sec:remember-me />
                <sec:intercept-url pattern="*.htm" access="ROLE_USER,ROLE_ANONYMOUS" />
                <sec:intercept-url pattern="*.page" access="ROLE_USER,ROLE_ADMIN" />
                <sec:intercept-url pattern="*.edit" access="ROLE_USER,ROLE_ADMIN" />
                <sec:intercept-url pattern="*.admin" access="ROLE_ADMIN" />
        </sec:http>

        <sec:authentication-provider user-service-ref="sffsUserDetailservice"><sec:password-encoder hash="sha" /></sec:authentication-provider>

        <bean id="accessDecisionManager" class="org.springframework.security.vote.AffirmativeBased">
                <property name="decisionVoters">
                        <list>
                                <bean class="org.springframework.security.vote.RoleVoter" />
                                <bean class="org.springframework.security.vote.AuthenticatedVoter" />
                        </list>
                </property>
        </bean>

        <bean id="sffsUserDetailservice" class="it.freshfruits.security.AuthenticationJdbcDaoImpl"
                p:rolePrefix="ROLE_" p:dataSource-ref="dataSource"
                p:usersByUsernameQuery="SELECT username, password, enabled FROM authentication WHERE username = ?"
                p:authoritiesByUsernameQuery="SELECT username, authority FROM roles WHERE username = ?" />

        <sec:global-method-security access-decision-manager-ref="accessDecisionManager">
                 <sec:protect-pointcut expression="execution(* it.freshfruits.domain.entity.*.*(..))"
                        access="ROLE_USER,ROLE_ADMIN" />
        </sec:global-method-security>
...

</beans>
```

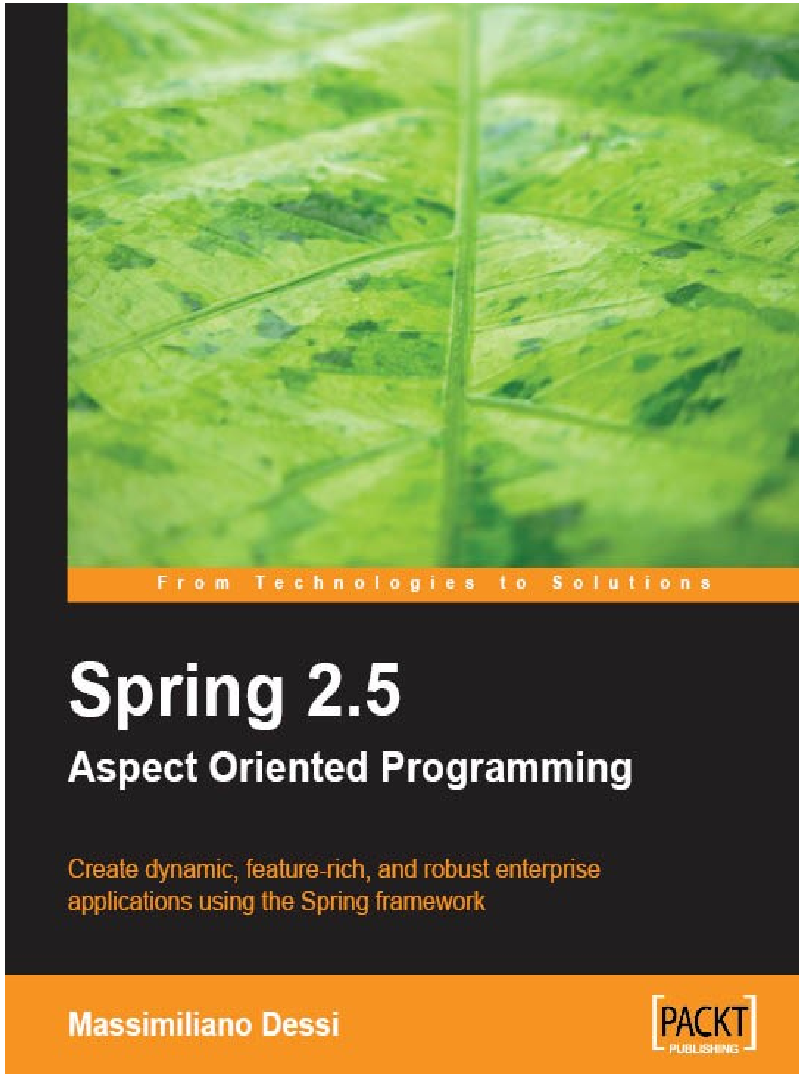
WEB.XML

```
...
<context-param>
     <param-name>webAppRootKey</param-name>
     <param-value>springFreshFruitsStore.root</param-value>
</context-param>
<context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>/WEB-INF/conf/spring/sffs-*.xml</param-value>
</context-param>
<listener>
     <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<filter>
     <filter-name>springSecurityFilterChain</filter-name>
     <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<servlet>
     <servlet-name>sffs</servlet-name>
     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
     <load-on-startup>0</load-on-startup>
</servlet>
<servlet-mapping><servlet-name>sffs</servlet-name><url-pattern>*.page</url-pattern></servlet-mapping>
<servlet-mapping><servlet-name>sffs</servlet-name><url-pattern>*.edit</url-pattern></servlet-mapping>
<servlet-mapping><servlet-name>sffs</servlet-name><url-pattern>*.admin</url-pattern></servlet-mapping>
<servlet-mapping><servlet-name>sffs</servlet-name><url-pattern>*.htm</url-pattern></servlet-mapping>
...
```

Jsp paginated list

```
<%@ include file="/WEB-INF/jsp/taglibs.jsp"%>
<tag:pager href="user.list.page"/>
<div>
    <div align="center"><span><spring:message code="ui.users"/></span></div>
    <c:if test="${msg ne ''}">${msg}</c:if>
    <table class="list">
        <thead><tr><th align="left"><spring:message code="ui.user.username"/></th></tr></thead>
        <tbody>
        <c:forEach var="user" items="${users}" varStatus="status">
            <c:choose>
            <c:when test="${status.count % 2 == 0}">
            <tr class="table-row-dispari"></c:when>
            <c:otherwise>
            <tr class="table-row-pari"></c:otherwise>
            </c:choose>
                <td align="left">${user.username}</td>
                <td align="center">
                  <a href="user.roles.page?sid=${user.id}"><img border="0" title="<spring:message code="ui.action.view"/>"
                  alt="<spring:message code="ui.action.view"/>" src="img/view.png"/></a>
                </td>
                <td align="center">
                  <a href="user.detail.page?sid=${user.id}"><img border="0" title="<spring:message code="ui.action.edit"/>"
                  alt="<spring:message code="ui.action.edit"/>" src="img/edit.png"/></a></td>
            </tr>
        </c:forEach>
        </tbody>
    </table>
</div>
<br/>
<tag:pager href="user.list.page"/>
<div align="center">
<input type="image" src="img/add.png" title="<spring:message code="ui.action.user.new"/>" alt="<spring:message
code="ui.action.user.new"/>" onclick="location.href = 'user.edit.page'" value="<spring:message code="ui.action.user.new"/>"/>
</div>
```

Spring AOP Book


From Technologies to Solutions
Spring 2.5
Aspect Oriented Programming
Create dynamic, feature-rich, and robust enterprise applications using the Spring framework
Massimiliano Dessi
PACKT PUBLISHING

More information
in February on :

http://www.packtpub.com

Thanks to Stefano Sanna for their support
in the
completion of the book.

Q & A

Thanks for your attention !

Massimiliano Dessì
desmax74 at yahoo.it

http://jroller.com/desmax
http://www.linkedin.com/in/desmax74
http://wiki.java.net/bin/view/People/MassimilianoDessi
http://www.jugsardegna.org/vqwiki/jsp/Wiki?MassimilianoDessi

Spring Framework Italian User Group
http://it.groups.yahoo.com/group/SpringFramework-it