BeJUG

Java Web Development

Spring framework

A practical introduction

Alef Arendsen
Software Engineer
JTeam / Support4J.net

JTeam
www.jteam.nl

Summary

Spring makes J2EE easier,

while enforcing best practices

and design patterns

and increasing efficiency

and overall code quality

Agenda

- Overview of goals and features
- Dependency Injection / IoC
- Spring MVC
- Exploring O/R mapping integration
- And the rest…
- Wrapping it up
- Questions and possibly also some answers

Agenda

- Overview of goals and features
- Dependency Injection / IoC
- Spring MVC
- Exploring O/R mapping integration
- And the rest…
- Wrapping it up
- Questions and possibly also some answers

Goal of the Spring framework

- Make J2EE easier to use
- Address end-to-end requirements rather than one tier (e.g. Struts)
- Eliminate need for middle-tier “glue”

- Provide the best IoC solution available
- Provide the best pure-Java AOP solution, focused on common problems (e.g. trans. mgt.)

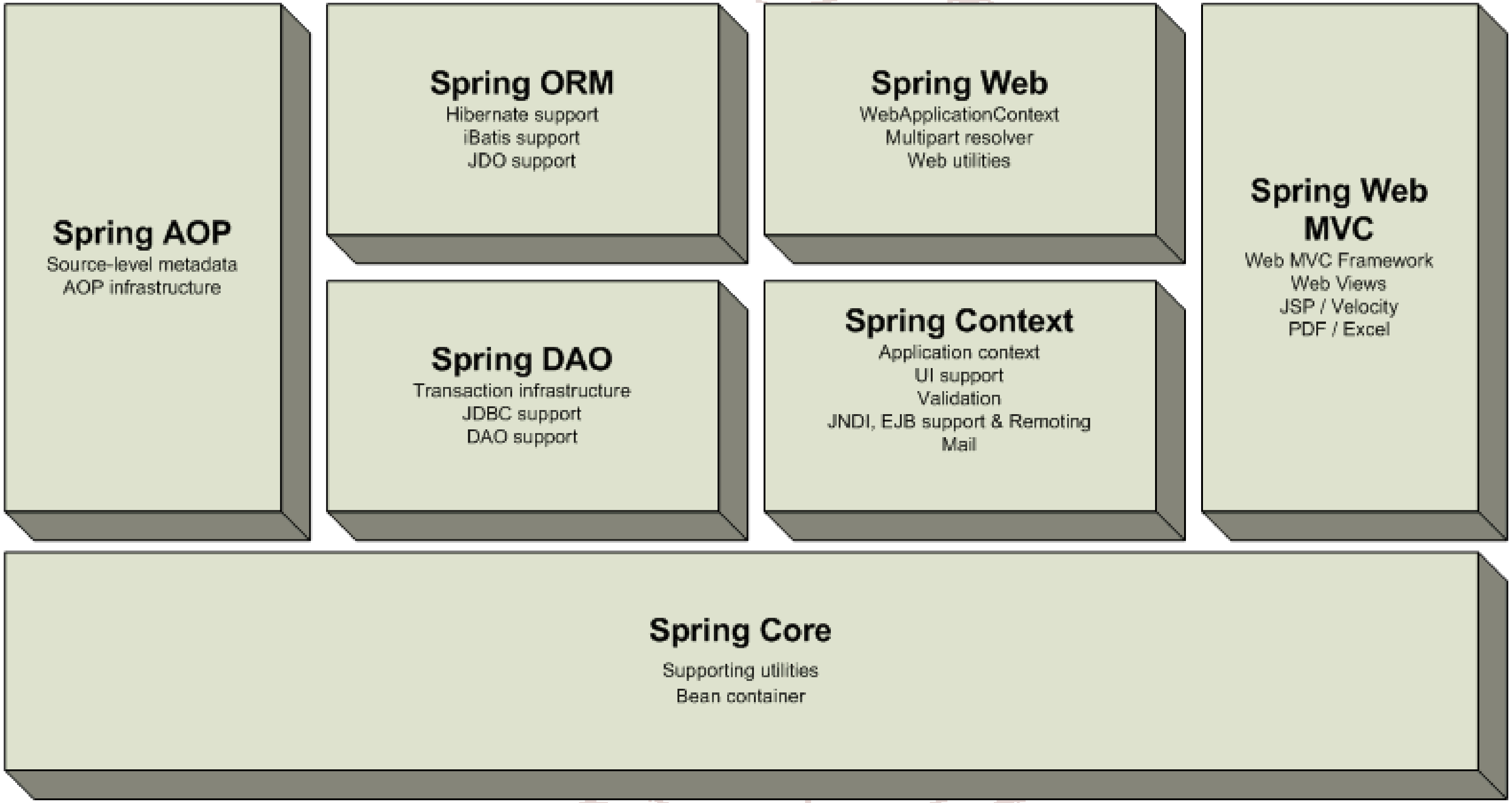
- Be as “non-invasive” as possible – little or no framework dependencies

- Enhance productivity compared to traditional approaches (TDD, OO best practices)

Key features

- Advanced IoC / Dependency Injection solution
- Extremely flexible MVC framework
- Declarative transaction management for POJOs
- Seamless integration of other technologies at all levels
- Easy to use pure-Java AOP implementation, usable across all layers

Architectural overview



Spring AOP
Source-level metadata
AOP infrastructure

Spring ORM
Hibernate support
iBatis support
JDO support

Spring Web
WebApplicationContext
Multipart resolver
Web utilities

Spring Web MVC
Web MVC Framework
Web Views
JSP / Velocity
PDF / Excel

Spring DAO
Transaction infrastructure
JDBC support
DAO support

Spring Context
Application context
UI support
Validation
JNDI, EJB support & Remoting
Mail

Spring Core
Supporting utilities
Bean container

Agenda

- Overview of goals and features
- **Dependency Injection / IoC**
- Spring MVC
- Exploring O/R mapping integration
- And the rest…
- Wrapping it up
- Questions and possibly also some answers

Dependency Injection / IoC

- Complete solution for managing objects across all layers
  - Write everything you need as POJOs
  - Wire them up using Springs BeanFactory simple and consistent XML format (although other formats are supported)
  - Supports constructor and setter-based dependency injection as well as manual injection or autowiring
- Business object do not depend on Spring
- Facilitates unit testing of your code
- No more environment-dependant lookups or server specific code
- No more resource management through the use of configurable object modes such as singleton, prototype, pooled, thread local

Spring IoC: Some concepts

- Constructor or setter-based Dependency Injection
- Singleton / Prototype / Pooled / ThreadLocal
- Lifecycle methods e.g. initialization and destruction
- Autowiring
- Dependency checking
- ApplicationContext & BeanFactory

A typical architecture

Presentation tier

JSPs, PDF, Excel,
Web controllers

Business tier

Domain model,
business objects

Integration tier

Persistence logic

A simple business object

```
public interface AccountManager {

    public void insertAccount(Account account);

    public List getAccounts();

    public void deleteAccount(Account account);
}
```

Wiring up the business object

```
<bean id=“accountManager”
    class=“example.AccountManagerImpl”>
  <property name=“dataSource”>
      <ref local=“dataSource”/>
  </property>
</bean>
```

void setDataSource(DataSource ds)

```
<bean id=“dataSource”
    class=“org.apache.commons.dbcp.BasicDataSource”>
    destroy-method=“close”>
  <property name=“url”>
      <value>${jdbc.url}</value>
  </property>
  . . . . .
</bean>
```

void setUrl(String url)

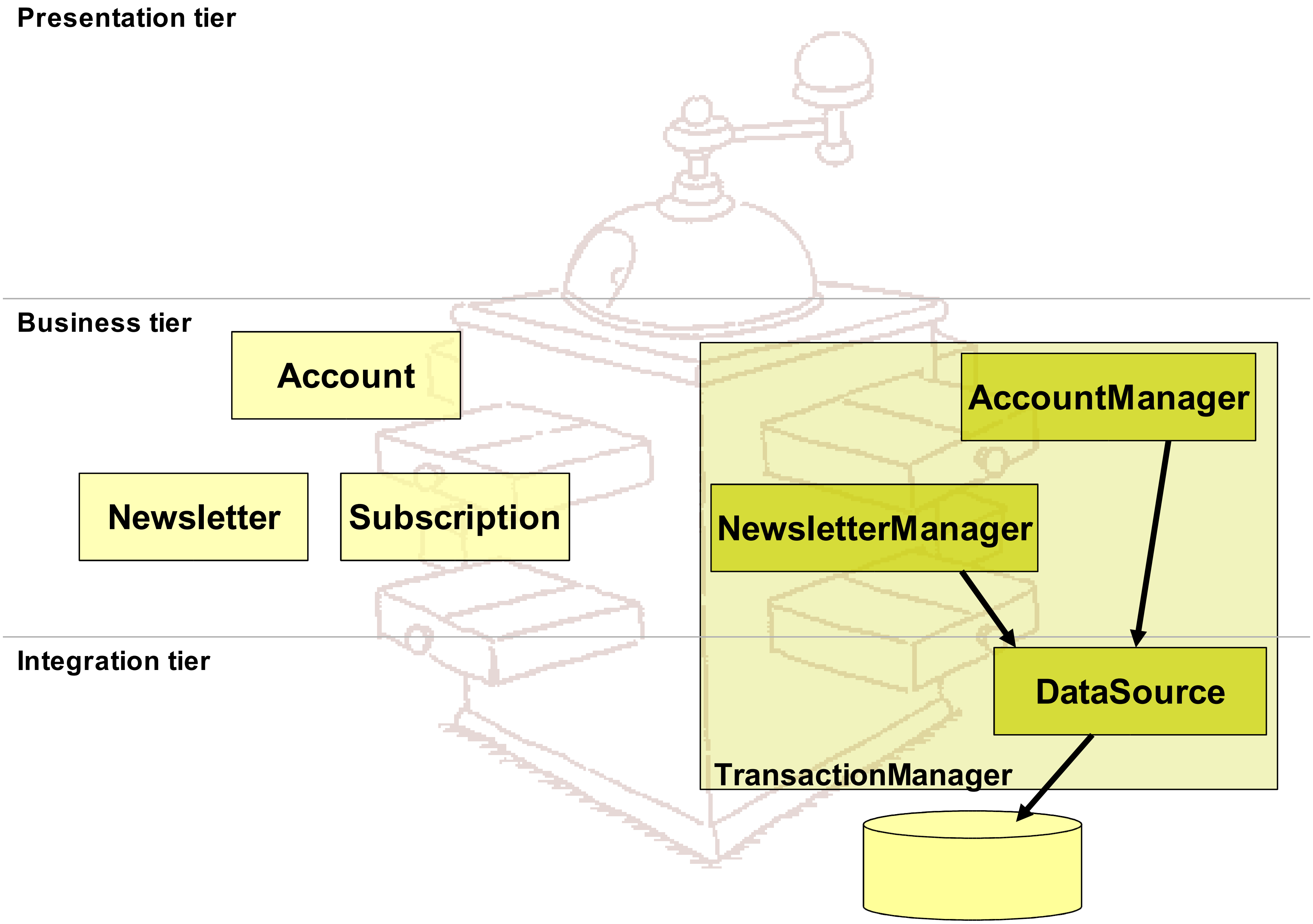
Adding transactional support

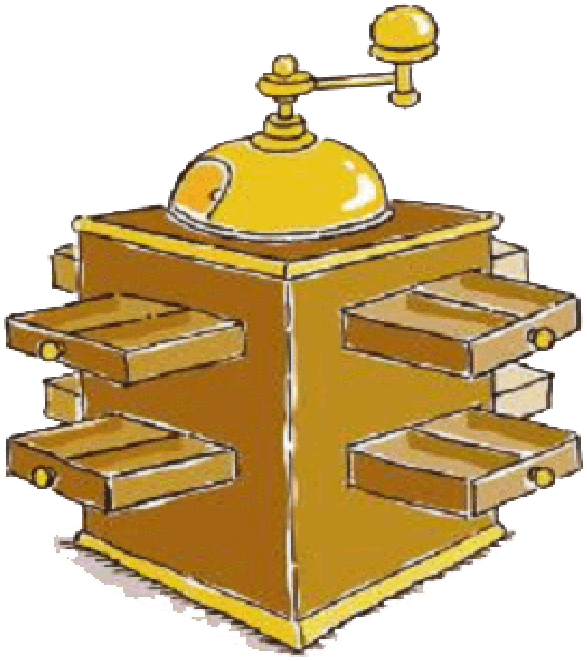
```
<bean id=“accountManagerTarget”
    class=“example.AccountManagerImpl”/>

 <bean id=“transactionManager”
     class=“...DataSourceTransactionManager”>
     <property name=“dataSource”
         <ref local=“dataSource”/>
     </property>
 </bean>

 <bean id=“accountManager”
     class=“...interceptor.TransactionProxyFactoryBean”>
     <property name=“target”>
         <ref local=“accountManagerTarget"/>
     </property>
     <property name=“transactionAttributes”>
        <props>
            <prop key=“insert*”>PROPAGATION_REQUIRED</prop>
            <prop key=“get*”>PROPAGATION_SUPPORTS</prop>
        </props>
    </property>
    . . . . .
 </bean>
```

The result so far





DEMO

Agenda

- Overview of goals and features
- Dependency Injection / IoC
- **Spring MVC**
- Exploring O/R mapping integration
- And the rest…
- Wrapping it up
- Questions and possibly also some answers

Flexible lean & mean MVC

- Integrated with BeanFactory and AOP
- No enforcing of superclasses (e.g. ActionForm)
- Transparent management of domain model
- Open, flexible, thin

Spring MVC: Some concepts

- DispatcherServlet (dispatches requests)
- WebApplicationContext (special AppContext)

- Controllers (classes doing the actual work)
- Model (Java Map)
- View (JSP / Velocity / Excel / XSLT / Tapestry)

- HandlerMapping (maps URLs to controllers)
- ViewResolver (maps viewnames to e.g. JSPs)

- Message ResourceBundles

Simple controller

```
public class SubscriptionViewController
extends AbstractController {


    private NewsletterManager manager;

    public void setNewsletterManager(
            NewsletterManager manager) {
        this.manager = manager;
    }


    public ModelAndView handleRequestInternal(...) {
        List subs = manager.getSubscriptions();
        ModelAndView mav = new
            ModelAndView(“subscriptionList”, “subs”, subs);
        return mav;
    }

}
```

Some wiring up…

```
<bean id="subscriptionViewController"
    name="/subscriptions.view"
```

```
<property name="urlMapping"
        class="...SimpleUrlHandlerMapping">
    <property name="interceptors">
        <list>
            <ref local="authenticationInterceptor"/>
        </list>
    </property>
    <property name="mappings">
        <props>
            <prop key="**/view*.html">
                subscriptionViewController
            </prop>
            <prop key="**/*logout*.do">
                logoutController
            </prop>
        </props>
    </property>
</property>
```

Form controller

```
public class SubscriptionController extends FormController {

    public void initBinder(ServletRequestDataBinder binder) {
        binder.registerCustomEditor(Account.class,
            new AccountEditor());
        binder.registerCustomEditor(Newsletter.class,
            new NewsletterEditor());
    }

    public Map referenceData(...) {
       Map m = new HashMap();
       m.put(“news”, newsletterManager.getNewsletters());
       return m;
    }


    public ModelAndView onSubmit(Object command) {
        newsletterManager.insertSubscription(
            (Subscription)command);
        return new ModelAndView(getSuccessView());
    }
}
```

Some wiring up…

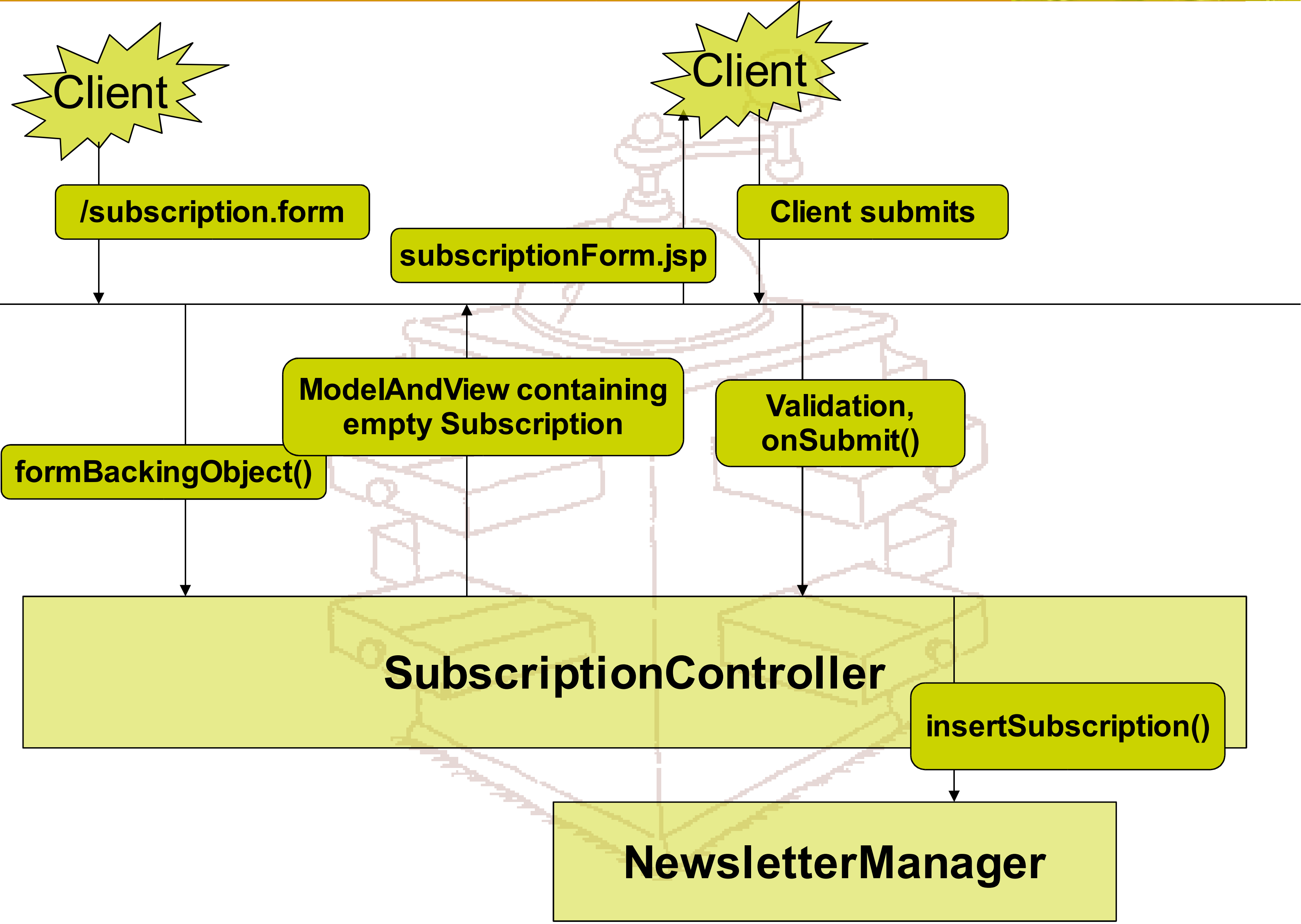
```
<bean id=“subscriptionController”
    name=“/subscriptions.form”
    class=“example.web.SubscriptionController>
    <property name=“commandClass”>
        <ref local=“example.Subscription”/>
    </property>
    <property name=“formView”>
        <value>subscriptionForm</value>
    </property>
    <property name=“successView”>
        <value>subscriptionCreated</value>
    </property>
    <property name=“validator”>
        <ref local=“subscriptionValidator”/>
    </property>
    <property name=“newsletterManager”>
        <ref bean=“newsletterManager”/>
    </property>
    <property name=“accountManager”>
        <ref bean=“accountManager”/>
    </property>
</bean>
```

Using propertyeditors in JSPs

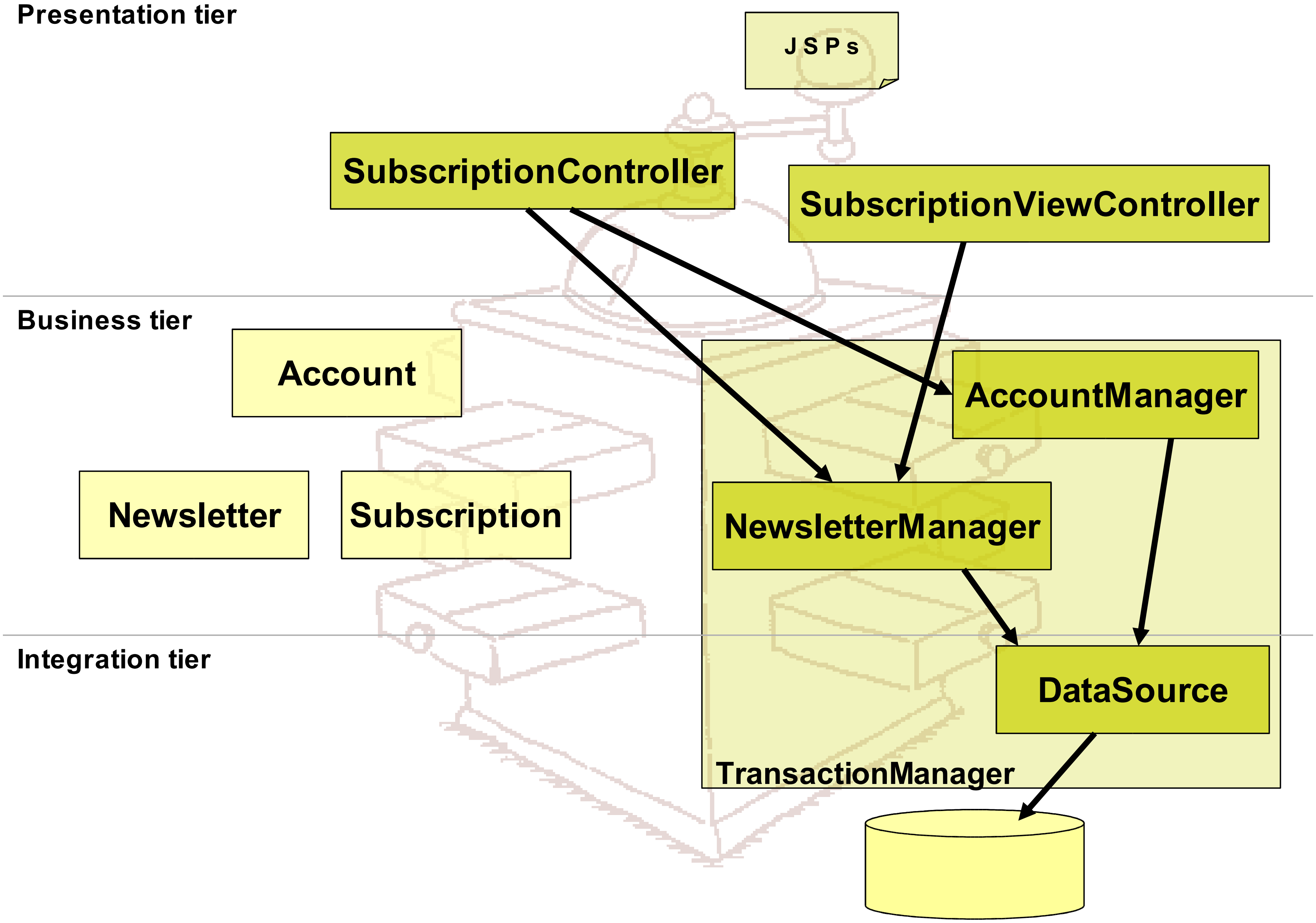
Using Springs data binding features (the PropertyEditors we’ve seen in the form controller before):

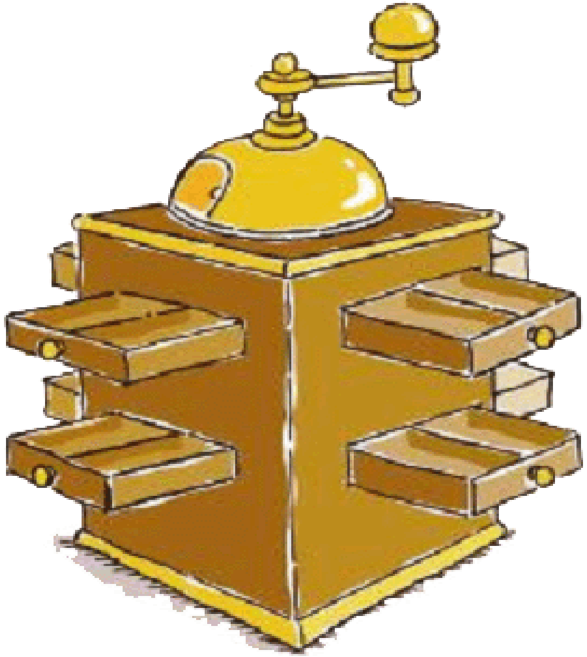
```
<spring:bind path=“command.newsletter”>
  <select name=“<c:out value=“${status.expression}”/>”>
    <c:forEach items=“${news}” var=“letter”>
      <option value=“<spring:transform value=“${letter}”/>”>
        <c:out value=“${letter.name}”/>
      </option>
  </select>
</spring:bind>
```

Let’s see the flow



The result so far





DEMO

Agenda

- Overview of goals and features
- Dependency Injection / IoC
- Spring MVC
- **Exploring O/R mapping integration**
- And the rest…
- Wrapping it up
- Questions and possibly also some answers

Adding Hibernate mappings

```
<bean id="sessionFactory"
        class=“...orm.hibernate.LocalSessionFactoryBean">
    <property name="dataSource">
        <ref local="dataSource"/>
    </property>
    <property name="mappingResources">
        <value>example/data/sample.hbm.xml</value>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                ${hibernate.dialect}
            </prop>
       </props>
    </property>
</bean>
```

The implementation

```
public class HibernateAccountManager
extends HibernateDaoSupport
implements AccountManager {

    public void insertAccount(Account account) {
        getHibernateTemplate().save(account);
    }

    public List getAccounts() {
        return getHibernateTemplate().find(“from Account‟ +
        ‟ account order by account.lastName,‟ +
        ‟ account.firstName");
    }

}
```

Other ORM and database tech.

- Similar abstraction layers all supporting transactionmanagement for
  - iBatis SQLMaps 1.3.x and 2.0
  - JDO
  - JDBC
- Unified exception hierarchy
- Error code translation (so no more ORA-9056)

Agenda

- Overview of goals and features
- Dependency Injection / IoC
- Spring MVC
- Exploring O/R mapping integration
- **And the rest…**
- Wrapping it up
- Questions and possibly also some answers

Other technologies

- Thin abstraction layers for sending email
- JNDI abstraction, removing the need to do lookups yourself (`<ref bean="ejb"/>`)
- EJB abstraction, Spring-aware
- Support for attribute-driven transaction management (commons-attributes and in the future JSR-175)
- Timer abstraction with out-of-the-box Quartz implementation
- Out-of-the-box remoting facilities for your bean, based on Hessian, Burlap, JAX-RPC or RMI

Agenda

- Overview of goals and features
- Dependency Injection / IoC
- Spring MVC
- Exploring O/R mapping integration
- And the rest…
- **Wrapping it up**
- Questions and possibly also some answers

Who’s using Spring

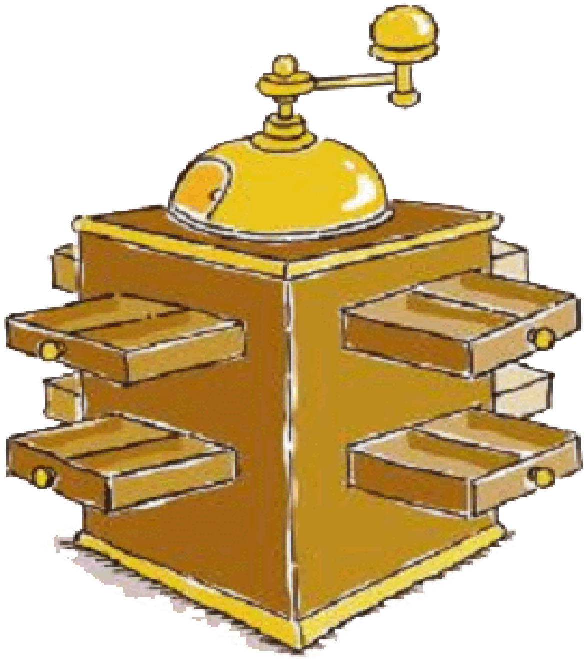
- Ilse Media / Sanoma, sales process management using Spring MVC, AOP, IoC and more
- Global investment bank, 2 projects live with Spring MVC, IoC and JDBC, 10.000 users
- FA Premier League
- German domestic bank
- Several Canadian, Austrian & UK-based consultancies

Quotes

- *I use the Spring Framework daily and I've simply been blown away by how much easier it makes development of new software components.*
- *The proof of concept went up to 150 requests per second! Man, you guys did a hell of job with the whole thing. Spring MVC overhead is *minimal* and it took only 15 hours to implement it, thanks for the dependency injection!*
- *I took some time last weekend and refactored AppFuse to use Spring to replace my Factories and Hibernate configuration. It only took me a couple of hours, which says a lot for Spring. I was amazed at how many things just worked. It actually lifted me out of my flu symptoms made me feel euphoric.*

Spring Roadmap

- 1.0 mid March 2004
- Alpha version of Eclipse plugin available
- Two books coming
  - J2EE without EJB (May 2004)
    Rod Johnson / Jürgen *Höller*
  - Spring Development (Q4 2004)
    Johnson / Höller / Risberg / Arendsen
- JMS & JMX support scheduled for 1.1
- Complete backward compatibility from 1.0RC1
- Extensive reference manual in the works



Q&A

Java Web Development