Test-Driven Development with Spring and Hibernate

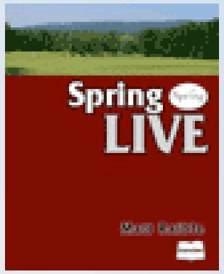
Matt Raible
mraible@virtuas.com

VIRTUAS

Introductions

- Your experience with webapps?
- Your experience with J2EE?
- What do you want to get from this tutorial?
- Open Source experience with Ant, XDoclet, Hibernate, Spring, Eclipse/IDEA?

Who is Matt Raible?



VIRTUAS

Agenda ~ Part I

- Overview of J2EE, Spring and Hibernate
- AppFuse: History, Overview and Demo
- Development Environment: JDK, Ant, AppFuse, Eclipse, MySQL and Tomcat

Agenda ~ Part II

- [Lab] Creating database tables using Ant and Hibernate
- JUnit and Test-Driven Development
- Review of interface vs. implementation classes
- [Lab] Writing a WeblogDAOTest JUnit test
- Overview of Spring's DAO Support classes for Hibernate
- [Lab] Write WeblogDAO interface and WeblogDAOHibernate implementation

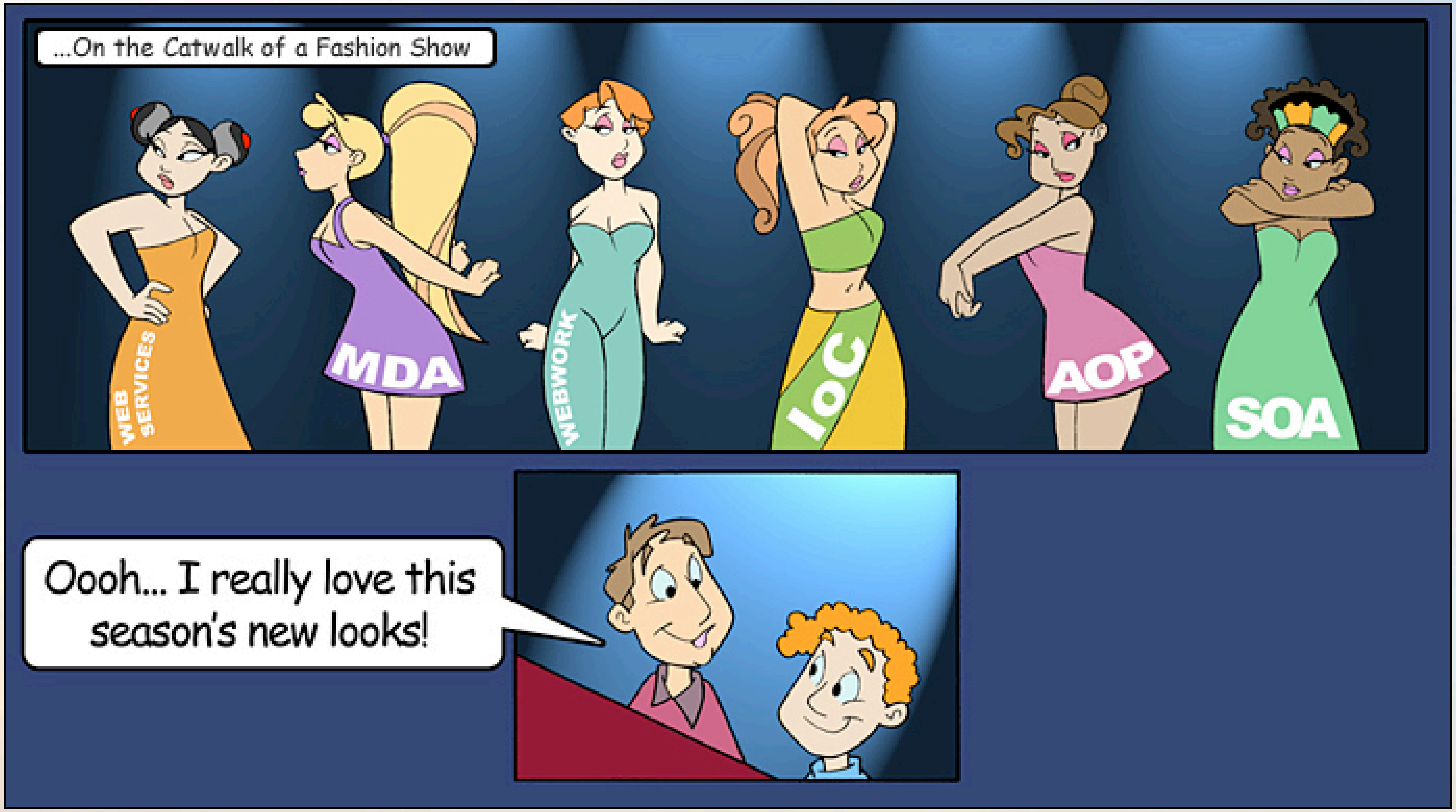
Agenda ~ Part III

- Business Facade: What is it and why should I use one?
- [Lab] Writing a WeblogManagerTest with jMock
- [Lab] Writing a WeblogManager interface and implementation
- [Lab] Defining the “weblogManager” bean and per-method transaction attributes
- Alternative I: Generic DAO and Manager
- Alternative II: Code Generation

Agenda ~ Optional

- Pick your favorite Java web framework:
  - JSF, Struts, Spring MVC, Tapestry or WebWork
- Testing Controllers (or page backing objects) for selected framework
- Testing the UI with Canoo WebTest and Ant

What is Spring?


...On the Catwalk of a Fashion Show
WEB SERVICES
MDA
WEBWORK
IoC
AOP
SOA
Oooh... I really love this season's new looks!

Spring Mission Statement

J2EE should be easier to use.

It's best to program to interfaces, rather than classes. Spring reduces the complexity of using interfaces to zero.

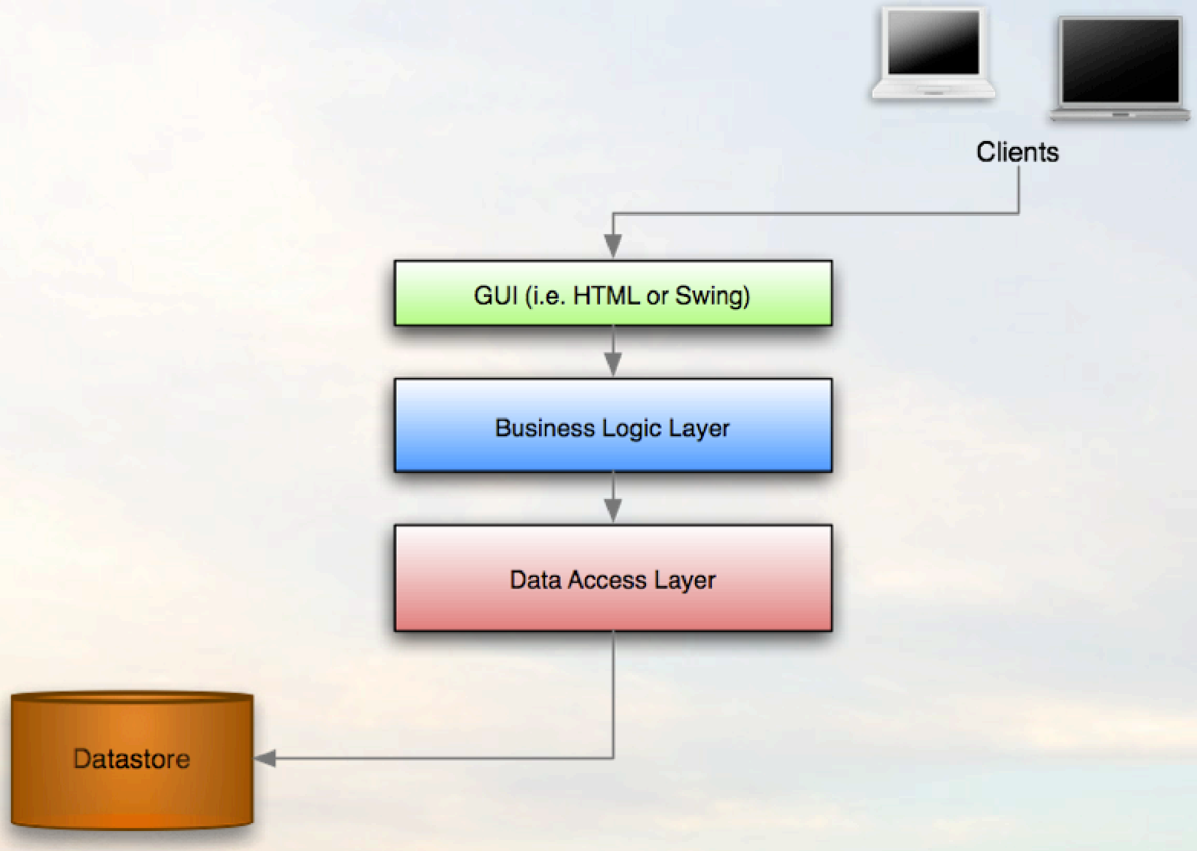
JavaBeans offer a great way of configuring applications.

Checked exceptions are overused in Java. A framework shouldn't force you to catch exceptions you're unlikely to recover from.

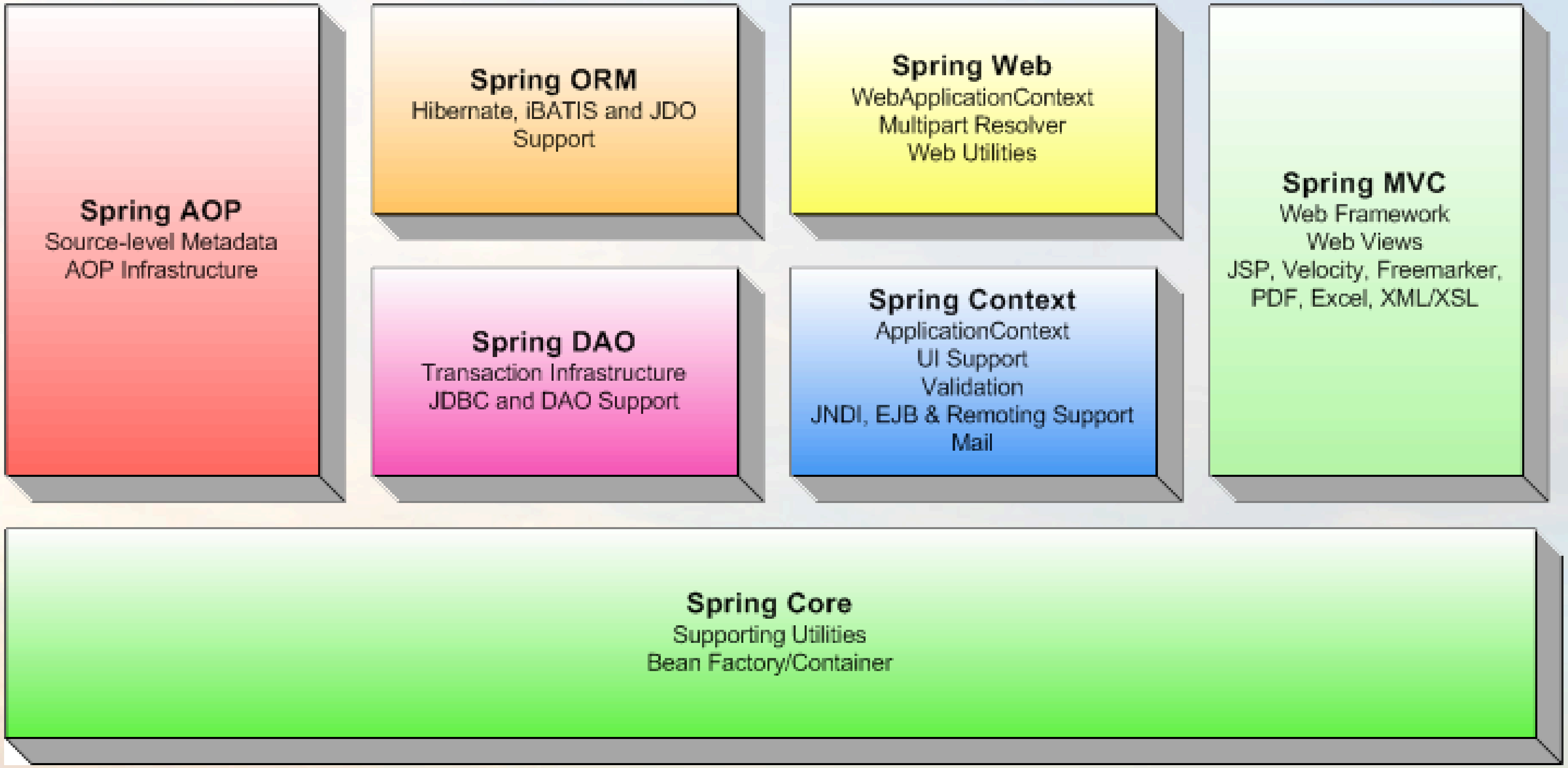
OO Design is more important than any implementation technology, such as J2EE.

Testability is essential, and a framework such as Spring should help make your code easier to test.

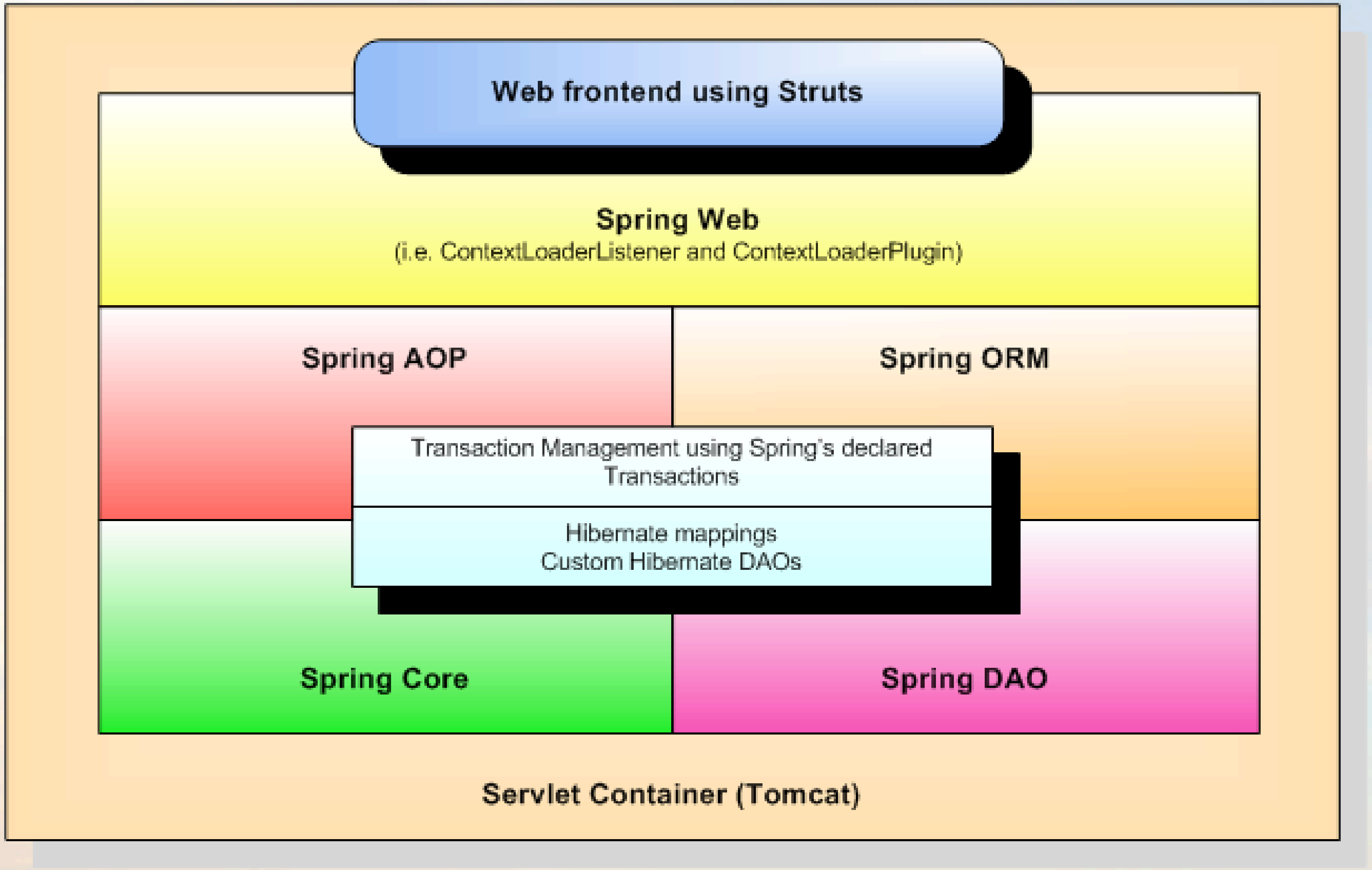
Typical Java Application



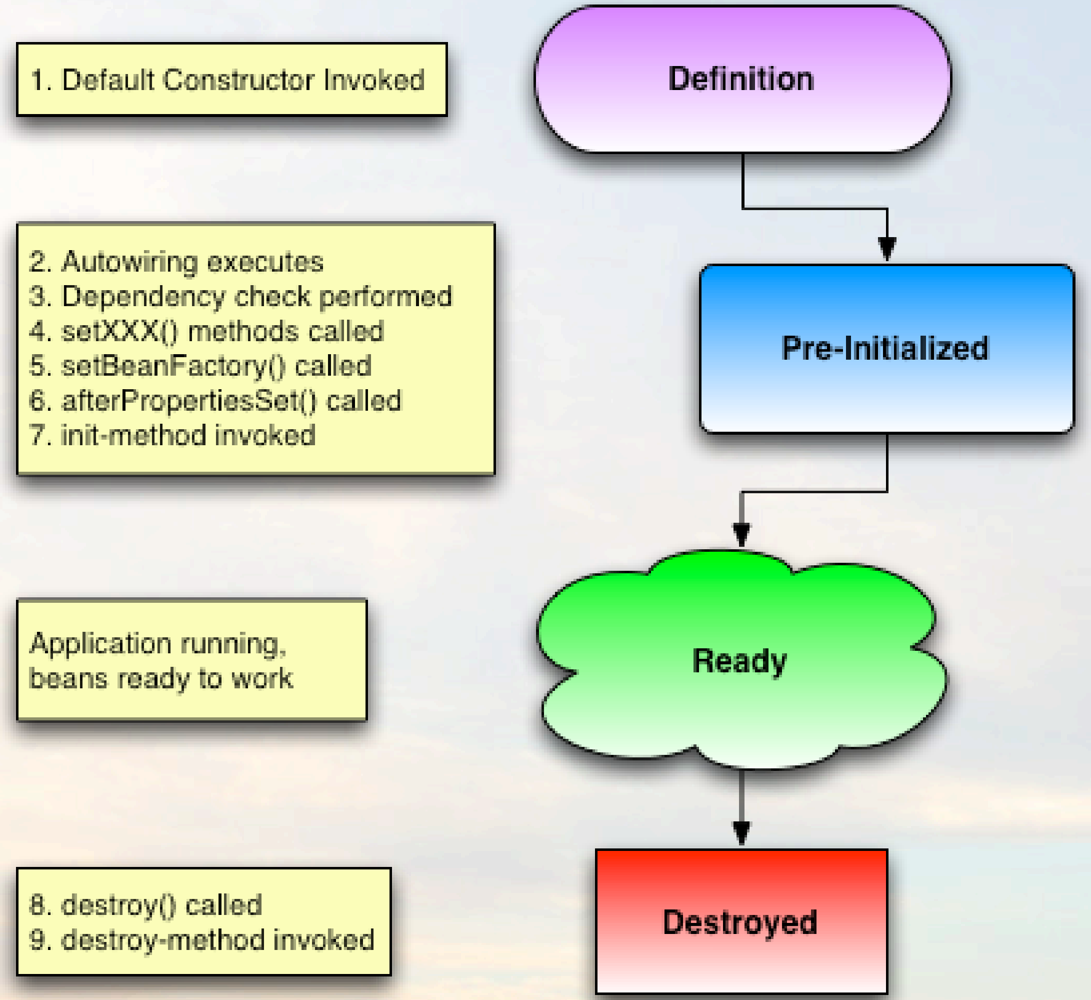
Spring Modules



Sample Architecture


Web frontend using Struts
Spring Web
(i.e. ContextLoaderListener and ContextLoaderPlugin)
Spring AOP
Spring ORM
Transaction Management using Spring's declared Transactions
Hibernate mappings
Custom Hibernate DAOs
Spring Core
Spring DAO
Servlet Container (Tomcat)

The BeanFactory


1. Default Constructor Invoked
Definition
2. Autowiring executes
3. Dependency check performed
4. setXXX() methods called
5. setBeanFactory() called
6. afterPropertiesSet() called
7. init-method invoked
Pre-Initialized
Application running, beans ready to work
Ready
8. destroy() called
9. destroy-method invoked
Destroyed

Get that Context!

- **ClassPathXmlApplicationContext** - load files from classpath
- **FileSystemXmlApplicationContext** - load from filesystem (relative or absolute paths)
- **StaticXmlApplicationContext** - allows programmatic registration of beans
- **XmlWebApplicationContext** - used by ContextLoaderListener to read paths from web.xml

ClassPathXmlApplicationContext

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("/WEB-INF/applicationContext.xml");
```

```
String paths[] = new String[] {"/WEB-INF/applicationContext-hibernate.xml",
                               "/WEB-INF/applicationContext.xml"};
ApplicationContext ctx = new ClassPathXmlApplicationContext(paths);
```

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("/WEB-INF/applicationContext*.xml");
```

Dependency Injection

Dependency Injection in Action
SO HOW DO YOU LIKE IT: SETTER, CONSTRUCTOR, OR INTERFACE?
THOUGHT LEADER
POJO
ORM
© 2004 HOMAM ALSAYED

Example: Controller and DAO

- Controller has a setter or a constructor argument for the DAO
- DAO is configured as a dependency of the Controller

```
<bean id="userController" class="org.appfuse.web.UserController">
    <property name="userDAO" ref="userDAO"/>
</bean>

<bean id="userDAO" class="org.appfuse.dao.UserDAOHibernate">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

- Makes it easy to swap out implementations
- Tests can verify operations succeed on the interface

Code before IoC

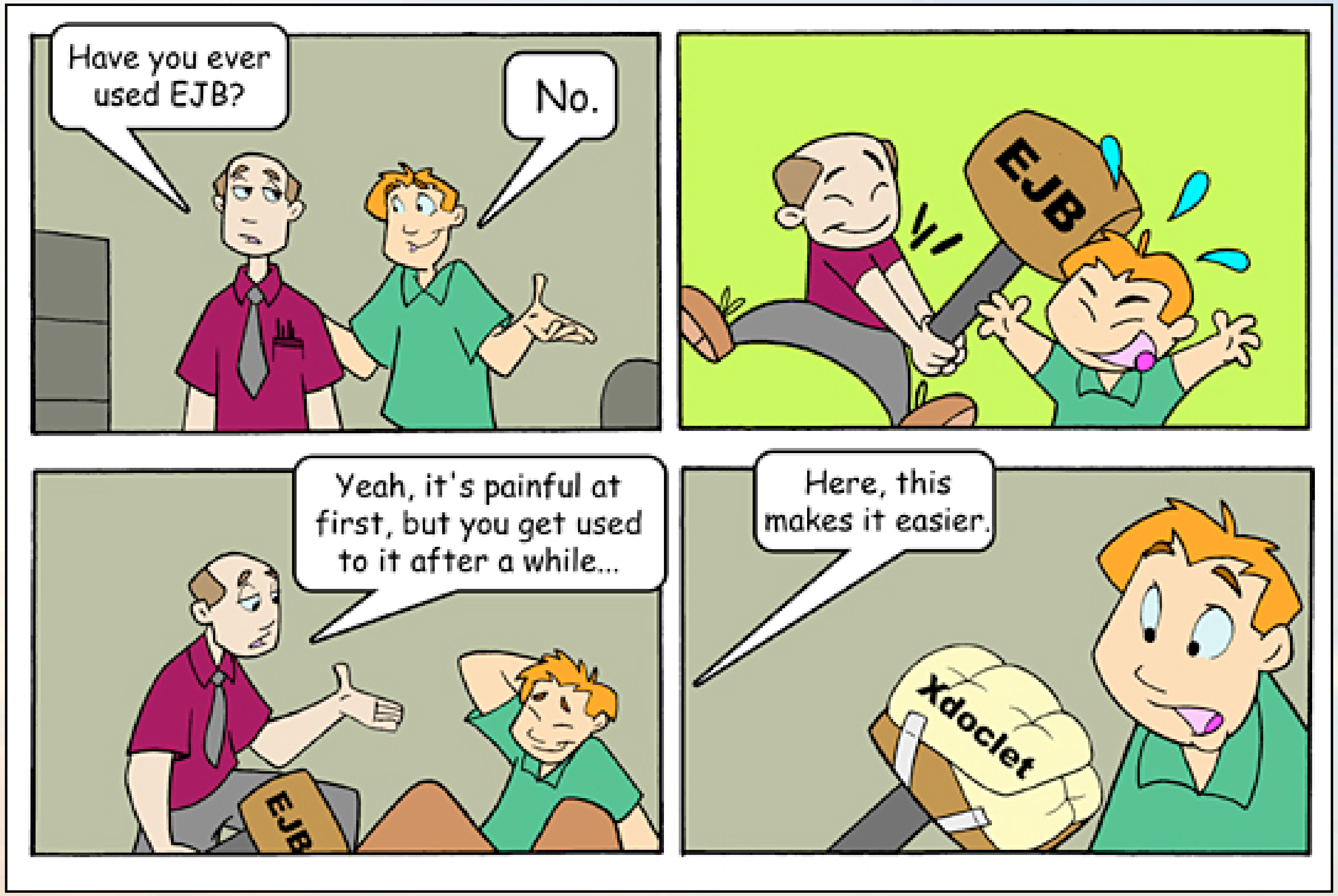
```
public class UserController implements Controller {

    public ModelAndView handleRequest(HttpServletRequest request,
                                      HttpServletResponse response)
    throws Exception {
        Connection conn = DatabaseUtils.getConnection();
        UserDAO dao = DAOFactory.createUserDAO("hibernate", conn);

        List users = dao.getUsers();

        DatabaseUtils.closeConnection(conn);

        return new ModelAndView("userList", "users", users);
    }
}
```

Code after IoC

```
public class UserController implements Controller {
    private UserDAO dao = null;

    public void setUserDAO(UserDAO userDAO) {
        this.dao = userDAO;
    }

    public ModelAndView handleRequest(HttpServletRequest request,
                                      HttpServletResponse response)
    throws Exception {

        List users = dao.getUsers();

        return new ModelAndView("userList", "users", users);
    }
}
```

Constructor vs. Setter Injection

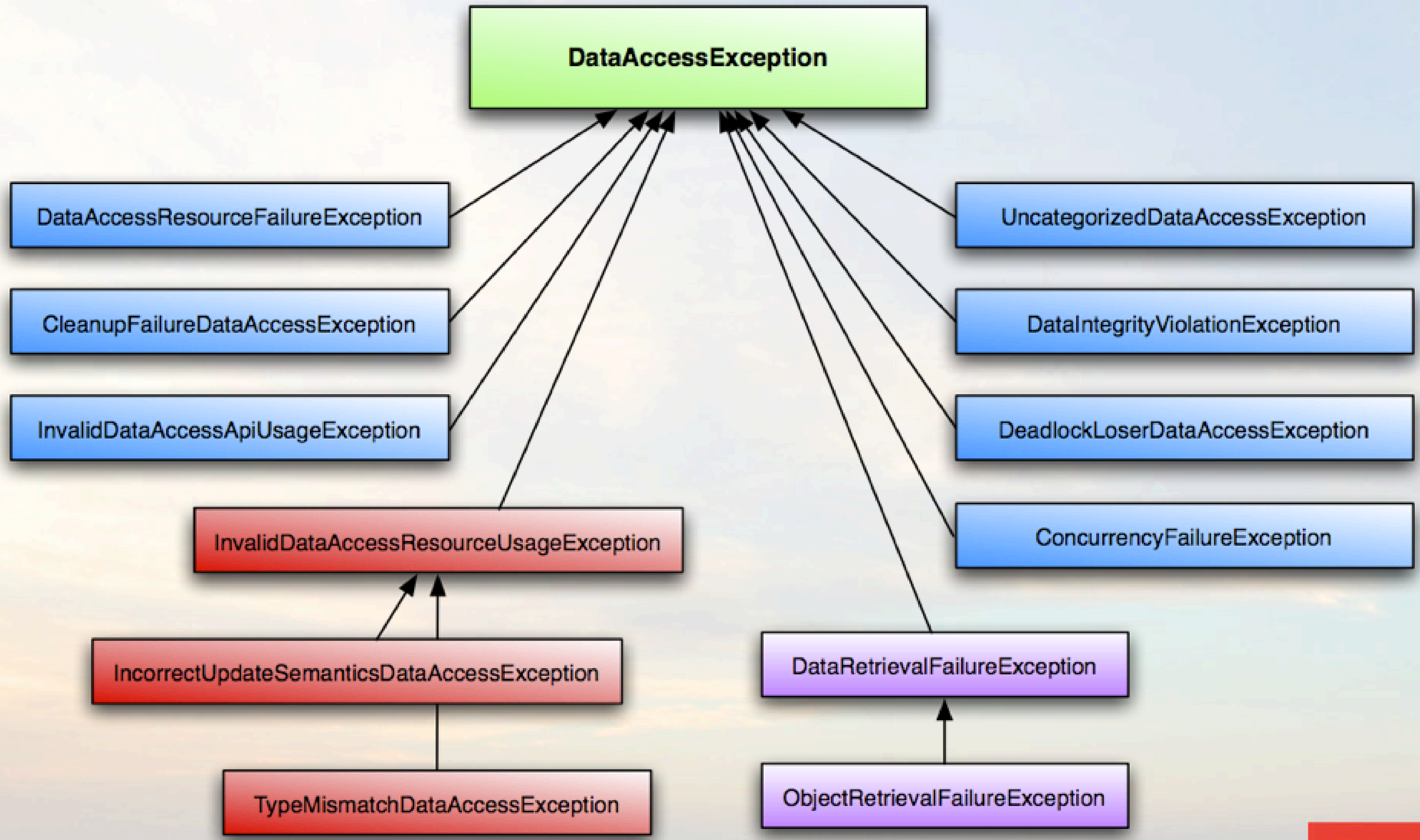
Spring vs. J2EE


Have you ever used EJB?
No.
EJB
Yeah, it's painful at first, but you get used to it after a while...
EJB
Here, this makes it easier.
Xdoclet

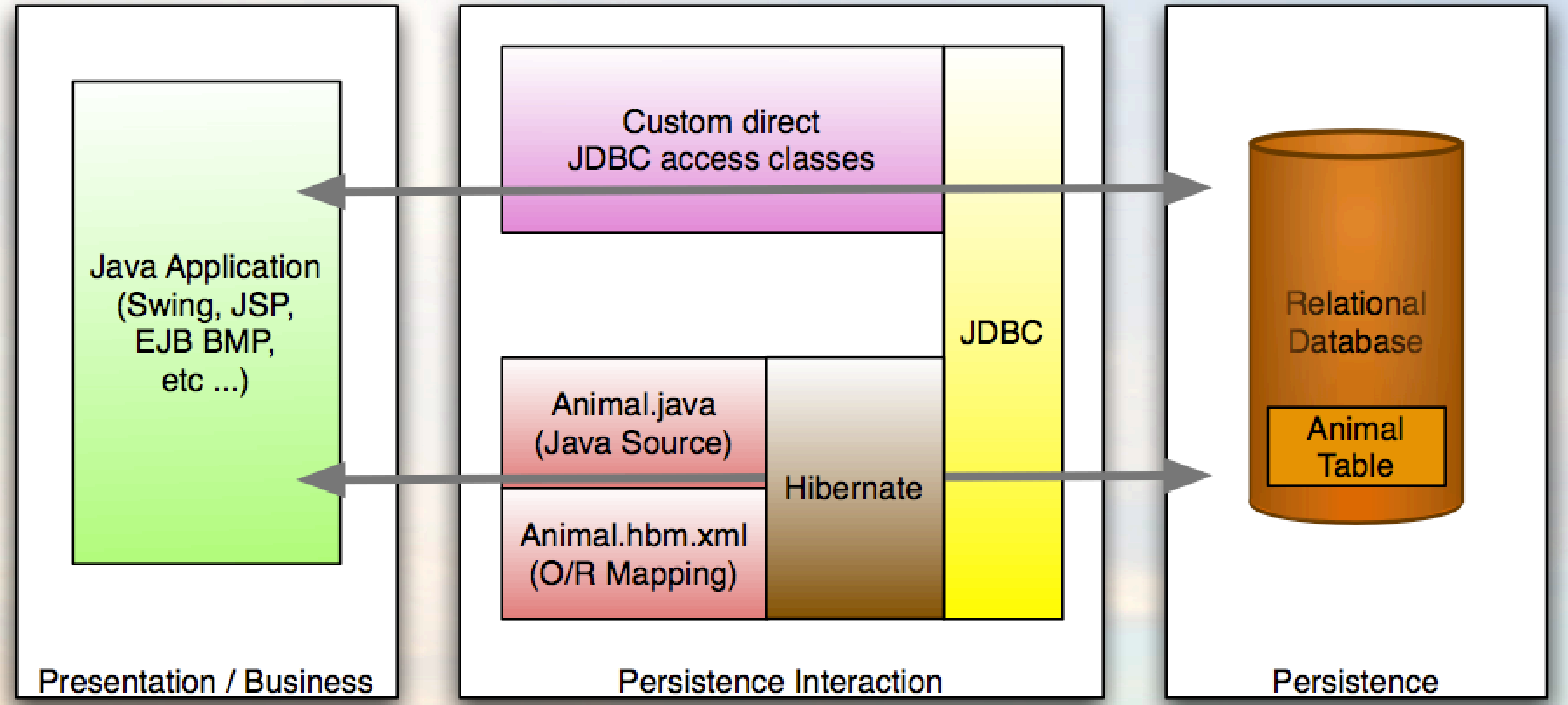
Data Access Support

- Meaningful Exceptions
- No more try/catch/finally
- DaoSupport and Template classes for many frameworks
  - Hibernate, iBATIS, JDO, OJB and TopLink

Rich Exception Hierarchy


DataAccessException
DataAccessResourceFailureException
CleanupFailureDataAccessException
InvalidDataAccessApiUsageException
InvalidDataAccessResourceUsageException
IncorrectUpdateSemanticsDataAccessException
TypeMismatchDataAccessException
UncategorizedDataAccessException
DataIntegrityViolationException
DeadlockLoserDataAccessException
ConcurrencyFailureException
DataRetrievalFailureException
ObjectRetrievalFailureException

Hibernate


Java Application (Swing, JSP, EJB BMP, etc ...)
Presentation / Business
Custom direct JDBC access classes
JDBC
Animal.java (Java Source)
Hibernate
Animal.hbm.xml (O/R Mapping)
Persistence Interaction
Relational Database
Animal Table
Persistence

Mapping a POJO

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="org.appfuse.model.User" table="app_user">

        <id name="id" column="id" unsaved-value="null">
            <generator class="native"/>
        </id>
        <property name="firstName" column="first_name" not-null="true"/>
        <property name="lastName" column="last_name" not-null="true"/>

    </class>
</hibernate-mapping>
```

Configuring Hibernate

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingResources">
        <list>
            <value>org/appfuse/model/User.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
        </props>
    </property>
</bean>

<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

VIRTUAS

Using Hibernate

```java
public class UserDAOHibernate extends HibernateDaoSupport implements UserDAO {

    public List getUsers() {
        return getHibernateTemplate().find("from User");
    }

    public User getUser(Long id) {
        return (User) getHibernateTemplate().get(User.class, id);
    }

    public void saveUser(User user) {
        getHibernateTemplate().saveOrUpdate(user);
    }

    public void removeUser(Long id) {
        getHibernateTemplate().delete(getUser(id));
    }
}
```

Callbacks

```java
public List getUsers(final User user) {
    if (user == null) {
        return getHibernateTemplate().find("from User");
    } else {
        // filter on properties set in the user object
        HibernateCallback callback = new HibernateCallback() {
            public Object doInHibernate(Session session) throws HibernateException {
                Example ex = Example.create(user).ignoreCase()
                                    .enableLike(MatchMode.ANYWHERE);
                return session.createCriteria(User.class).add(ex).list();
            }
        };
        return (List) getHibernateTemplate().execute(callback);
    }
}
```

Lazy-Loading Support

- OpenSessionInViewFilter and Interceptor
- Domain objects with lazy-loaded collections need to keep Session/PersistenceManager opened in unit tests

Configuring Filters

- /WEB-INF/web.xml

```
<filter>
    <filter-name>hibernateFilter</filter-name>
    <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>
</filter>

<!-- The 'hibernateFilter', a.k.a. Spring's OpenSessionInViewFilter
     guarantees one session per request.  Performance seems to be the
     same if it's enabled or disabled. -->
<filter-mapping>
    <filter-name>hibernateFilter</filter-name>
    <url-pattern>*.html</url-pattern>
</filter-mapping>
```

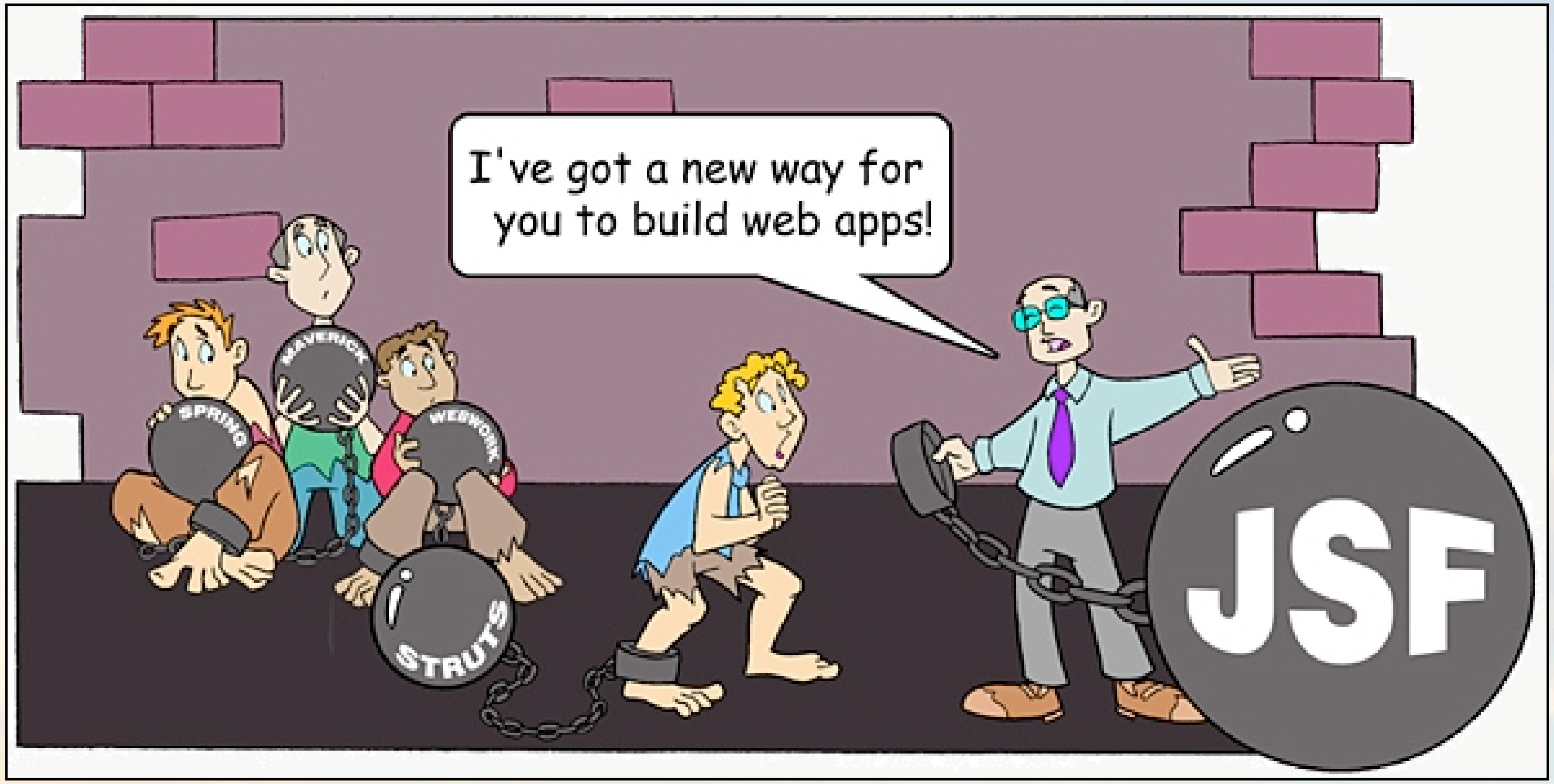
Testing Hibernate

```
public void testGetUsers() {
    // add a record to the database so we have something to work with
    User user = new User("Easter", "Bunny");
    dao.saveUser(user);

    List users = dao.getUsers();
    assertTrue(users.size() >= 1);
    assertTrue(users.contains(user));
}
```

Testing with Lazy-Loading

```
protected void setUp() throws Exception {
    // the following is necessary for lazy loading
    sf = (SessionFactory) ctx.getBean("sessionFactory");
    // open and bind the session for this test thread.
    Session s = sf.openSession();
        TransactionSynchronizationManager
          .bindResource(sf, new SessionHolder(s));

    // setup code here
}

protected void tearDown() throws Exception {
    // unbind and close the session.
    SessionHolder holder = (SessionHolder)
        TransactionSynchronizationManager.getResource(sf);
    Session s = holder.getSession();
    s.flush();
    TransactionSynchronizationManager.unbindResource(sf);
    SessionFactoryUtils.closeSessionIfNecessary(s, sf);

    // teardown code here
}
```

Spring MVC


I've got a new way for you to build web apps!
SPRING
MAVERICK
WEBWORK
STRUTS
JSF

Spring Supports many Java MVC Frameworks

- ContextLoaderListener loads beans into ServletContext - can retrieve with:

```
WebApplicationContext ctx =
            WebApplicationContextUtils.getWebApplicationContext(servletContext);
```

- Struts has ContextLoaderPlugin and ActionSupport classes
- WebWork has SpringObjectFactory
- Tapestry - override BaseEngine and stuff the context into the Global
- JSF - DelegatingVariableResolver

Transactions

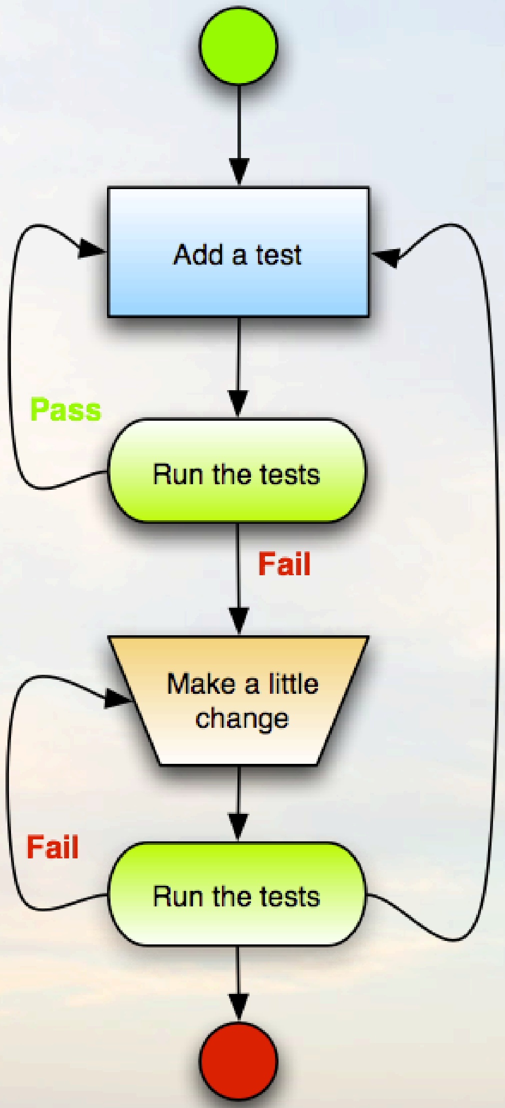
- Much easier to use than UserTransaction in J2EE
- CMT-like capabilities with XML and transparent AOP
- Supports Commons Attributes and JDK 5 Annotations
- Pluggable Transaction Managers, including JTA

AOP

- AOP - Reduces duplication among classes
- Interceptors make it easy to add before, around and after method advice
- Useful for caching, logging and security
    - EHCache, Performance/Tracing interceptors, Acegi for Security

Testing Spring Applications

Test-First Development



Unit vs. Integration Testing

Tools for Testing

- JUnit, TestNG, JTiger
  - Comparison: http://www.theserverside.com/news/thread.tss?thread_id=35248
- EasyMock and jMock
- Spring Mocks
- DbUnit
- Cactus
- jWebUnit, Canoo WebTest and Selenium
- Cargo
- Anthill, CruiseControl, Continuum

JUnit

```
package org.appfuse;

import junit.framework.TestCase;

import org.appfuse.model.User;

public class SimpleTest extends TestCase {

    public void testGetFullName() {
        User user = new User();
        user.setFirstName("Jack");
        user.setLastName("Raible");
        assertEquals("Jack Raible", user.getLastName());
    }
}
```

EasyMock and jMock

- **EasyMock**: provides mocks for interfaces in JUnit tests by generating them on-the-fly using Java's proxy mechanism.
- **jMock** is similar, but requires you to extend **MockObjectTestCase**.

EasyMock Example

```java
private UserManager mgr = new UserManagerImpl();
private MockControl control;
private UserDAO mockDAO;

protected void setUp() throws Exception {
    // Create a MockControl
    control = MockControl.createControl(UserDAO.class);
    // Get the mock
    mockDAO = (UserDAO) control.getMock();
    mgr.setUserDAO(mockDAO);
}

protected void testGetUser() {
    // Set expected behavior
    mockDAO.removeUser(new Long(1));
    control.setVoidCallable();

    // Active the mock
    control.replay();

    // Execute method to test
    mgr.removeUser("1");

    // Verify methods called
    control.verify();
}
```

jMock Example

```
private UserManager mgr = new UserManagerImpl();
private Mock mockDAO;

protected void setUp() throws Exception {
    // Create a Mock
    mockDAO = new Mock(UserDAO.class);

    // Set dependencies
    mgr.setUserDAO((UserDAO) mockDAO.proxy());
}

protected void testGetUser() {
    // Set expected behavior
    mockDAO.expects(once()).method("getUser")
           .with( eq(new Long(1)));

    // Execute method to test
    mgr.removeUser("1");

    // Verify expectations
    mockDAO.verify();
}
```

Spring Mocks

- Stubs for Servlet and JNDI API
- Spring's **org.springframework.test** package has a number of base classes to simplify testing
    - **AbstractDependencyInjectionSpringContextTests** - can do both setter or field-based dependency injection, cached context files
    - **AbstractTransactionalDataSourceSpringContextTests** - allows you to easily clear data from tables and rolls back any data entered into the database

IoC in Tests

```
public class UserManagerIntegrationTest extends AbstractTransactionalDataSourceSpringContextTests {
    private UserManager userManager;

    public void setUserManager(UserManager userManager) {
        this.userManager = userManager;
    }

    protected String[] getConfigLocations() {
        return new String[] {"classpath*:/WEB-INF/applicationContext*.xml"};
    }

    protected void onSetUpInTransaction() throws Exception {
        deleteFromTables(new String[] {"users"});
    }

    public void testGetUsers() {
        User kid1 = new User("Abbie", "Raible");
        User kid2 = new User("Jack", "Raible");
        userManager.saveUser(kid1);
        userManager.saveUser(kid2);

        assertEquals(2, userManager.getUsers().size());
    }
}
```

JNDI in Tests

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:comp/env/jdbc/appfuse"/>
</bean>
```

```
import org.springframework.mock.jndi.SimpleNamingContextBuilder;

try {
    SimpleNamingContextBuilder builder =
        SimpleNamingContextBuilder.emptyActivatedContextBuilder();

    DriverManagerDataSource ds = new DriverManagerDataSource();
        ds.setDriverClassName("com.mysql.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost/appfuse");
        ds.setUsername("root");
        ds.setPassword("");
    builder.bind("java:comp/env/jdbc/appfuse", ds);
} catch (NamingException ne) {
    // do nothing, test will fail on its own
}
```

DBUnit

```
private IDatabaseConnection conn = null;
private IDataSet dataSet = null;

protected void setUp() throws Exception {
    DataSource ds = (DataSource) ctx.getBean("dataSource");
    conn = new DatabaseConnection(ds.getConnection());
    dataSet = new XmlDataSet(new FileInputStream(
                                      "test/data/sample-data.xml"));
    // clear table and insert only sample data
    DatabaseOperation.CLEAN_INSERT.execute(conn, dataSet);
}

protected void tearDown() throws Exception {
    // clear out database
    DatabaseOperation.DELETE.execute(conn, dataSet);
    conn.close();
    conn = null;
}
```

Canoo WebTest

```
<target name="UserTests"
    description="Adds a new user profile">
    <canoo name="userTests">
        &config;
        <steps>
            <!-- View add screen -->
            <invoke url="/editUser.html"/>
            <verifytitle text="${userForm.title}"/>
            <!-- Enter data and save -->
            <setinputfield name="firstName" value="Test"/>
            <setinputfield name="lastName" value="Name"/>
            <clickbutton label="Save"/>
            <!-- View user list -->
            <verifytitle text="${userList.title}"/>
            <!-- Verify PDF contains new user -->
            <clicklink label="PDF"/>
            <verifyPdfText text="Test Name"/>
            <!-- Delete first user in table -->
            <invoke url="/users.html"/>
            <clicklink href="editUser.html"/>
            <verifytitle text="${userForm.title}"/>
            <clickbutton label="Delete"/>
            <verifytitle text="${userList.title}"/>
        </steps>
    </canoo>
</target>
```

jWebUnit

```
public class UserWebTest extends WebTestCase {

    public UserWebTest(String name) {
        super(name);
        getTestContext().setBaseUrl("http://localhost:8080/appfuse");
    }

    public void testWelcomePage() {
        beginAt("/");
        assertTitleEquals("AppFuse ~ Welcome");
    }

    public void testAddUser() {
        beginAt("/editUser.html");
        assertTitleEquals("AppFuse ~ User Details");
        setFormElement("firstName", "Jack");
        setFormElement("lastName", "Raible");
        submit("save");
        assertTextPresent("saved successfully");
    }
}
```

Cargo

```
Container container =
    new Resin3xContainer(new Resin3xStandaloneConfiguration("target/resin3x"));
container.setHomeDir("c:/apps/resin-3.0.8");

container.start();
```

```
<cargo-resin3x homeDir="c:/apps/resin-3.0.8" action="start"/>
```

AppFuse ~ what is it?

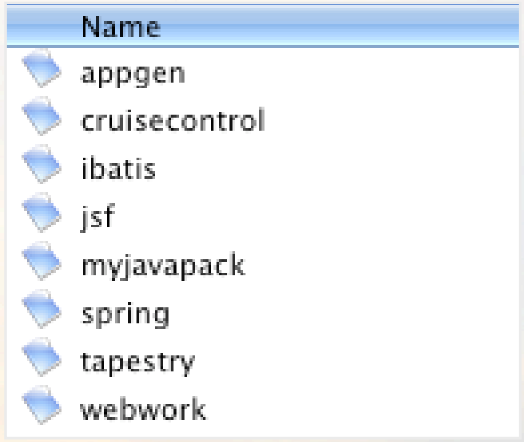
- A directory structure, build file and project classes to get your project started quickly
- The hard part is getting started and configuring dependencies
- Uses popular open-source tools: Ant, XDoclet, Spring, Hibernate, Struts (or JSF, Spring MVC, WebWork or Tapestry)
- Top 5 java.net project in hits, accesses and mail traffic

History

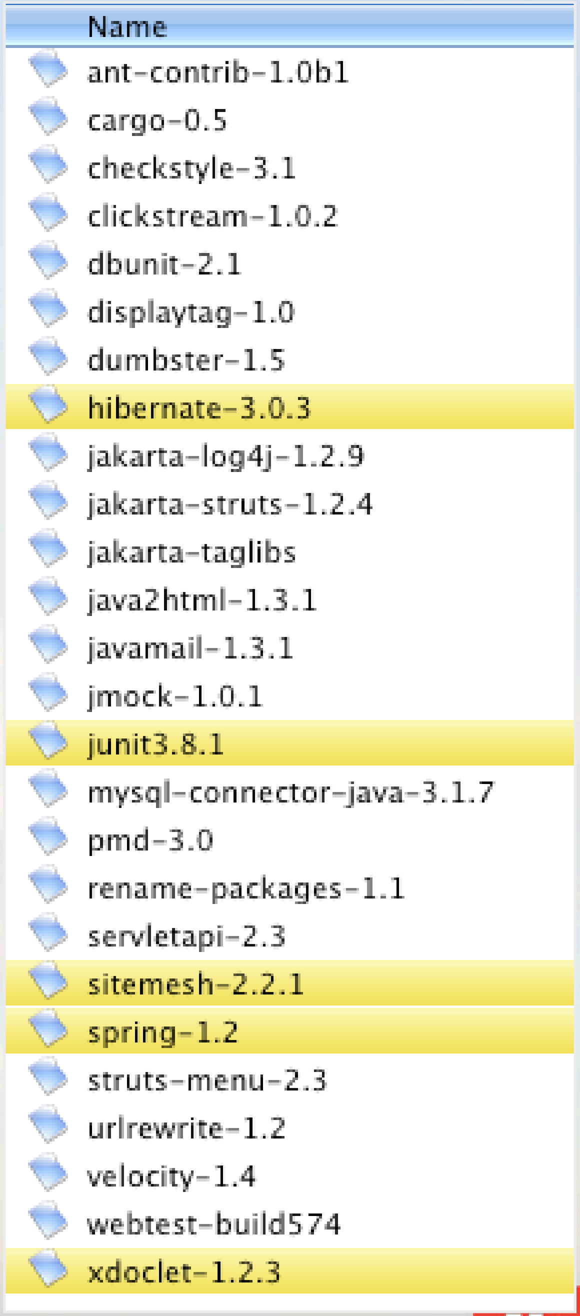
- Started as a sample app for Pro JSP
- Became a toolkit for starting new projects
- Lots of community feedback makes it a "best practices" webapp for J2EE
- Documentation and Tutorials (November 2003)
- AppGen - CRUD made easy (November 2004)
- New Committers: Nathan, Ben and Sanjiv (2005)

Dependencies

Optional Installs

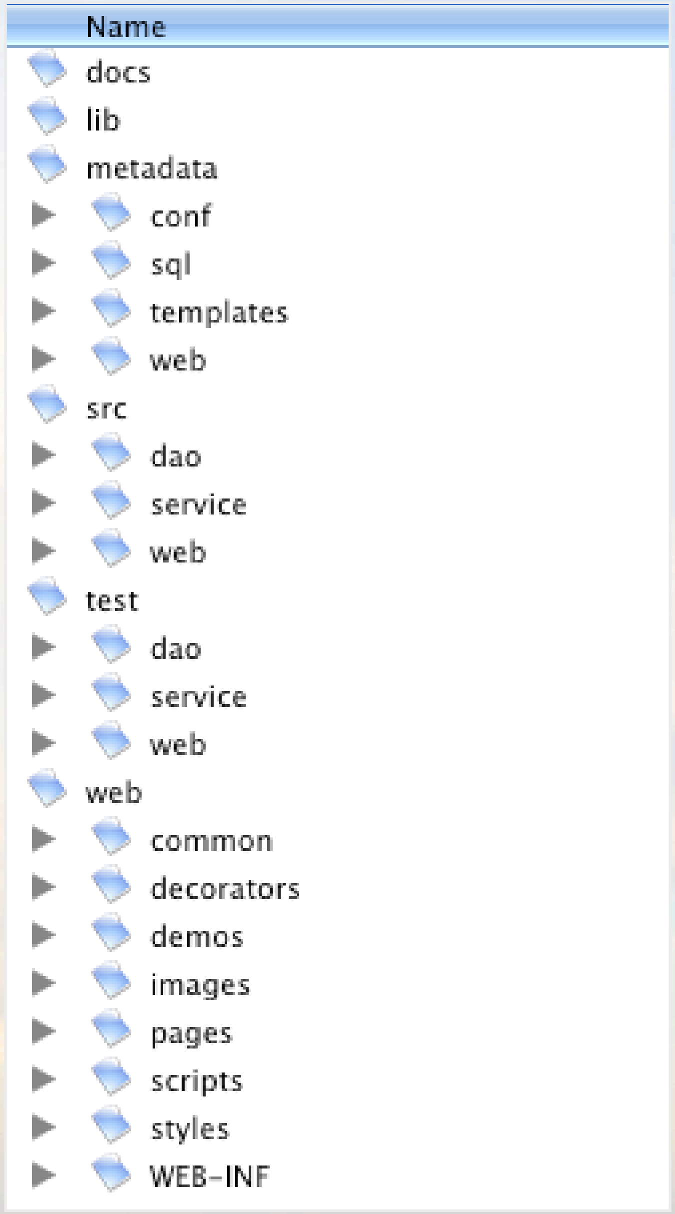


Name
- appgen
- cruisecontrol
- ibatis
- jsf
- myjavapack
- spring
- tapestry
- webwork



Name
- ant-contrib-1.0b1
- cargo-0.5
- checkstyle-3.1
- clickstream-1.0.2
- dbunit-2.1
- displaytag-1.0
- dumbster-1.5
- hibernate-3.0.3
- jakarta-log4j-1.2.9
- jakarta-struts-1.2.4
- jakarta-taglibs
- java2html-1.3.1
- javamail-1.3.1
- jmock-1.0.1
- junit3.8.1
- mysql-connector-java-3.1.7
- pmd-3.0
- rename-packages-1.1
- servletapi-2.3
- sitemesh-2.2.1
- spring-1.2
- struts-menu-2.3
- urlrewrite-1.2
- velocity-1.4
- webtest-build574
- xdoclet-1.2.3

Directory Structure



```
Name
docs
lib
metadata
  ▶ conf
  ▶ sql
  ▶ templates
  ▶ web
src
  ▶ dao
  ▶ service
  ▶ web
test
  ▶ dao
  ▶ service
  ▶ web
web
  ▶ common
  ▶ decorators
  ▶ demos
  ▶ images
  ▶ pages
  ▶ scripts
  ▶ styles
  ▶ WEB-INF
```

Demo of Features

- Acegi Security - makes security more portable between containers
- Remember Me and Self Registration
- GZip Compression Built-in
- Testing environment ready to go, many tutorials, CruiseControl files included
- http://demo.appfuse.org/appfuse

Development Environment

- Download and install:
    - Ant 1.6.2+
    - MySQL 4.1.x (or 5.0.x)
    - Tomcat 5.0.29
    - Eclipse 3.1 (or IDEA 4.5.x)
    - AppFuse 1.8.1
- http://raibledesigns.com/wiki/Wiki.jsp?page=DevelopmentEnvironment

Testing DAOs

Create database & table

- Create new project and database with Ant
- Create Weblog.java POJO and generate Hibernate mapping file with XDoclet
- Configure Spring to be aware of Weblog object
- Create “weblog” table from POJO using Ant

JUnit and Interfaces

- TDD makes you think before you code
- JUnit is easiest to test with because of plethora of tools and examples
- Writing interfaces allows de-coupling of layers and implementation
- Write code to test interface (integration test) or implementation (unit test)

Let’s write some code!

- What is a DAO?
- Create WeblogDAOTest - Test First!
- Create WeblogDAO Interface
- Create WeblogDAOHibernate implementation
- Create “weblogDAO” bean definition in Spring context file
- Run JUnit Test

Spring simplifies testing

- Spring's org.springframework.test package has a number of base classes to simplify testing
  - AbstractDependencyInjectionSpringContextTests - can do both setter or field-based dependency injection, cached context files
  - AbstractTransactionalDataSourceSpringContextTests - exposes a JdbcTemplate for easy querying and rolls back any data entered into the database

VIRTUAS

Testing with Mocks

Business Facades

- Encapsulates business logic for multiple clients
- Similar to Business Delegate pattern for EJBs
- Provide client-friendly, transactional access to data layer
- Facilitates calling multiple DAOs within the same transaction
- Can be exposed as web services

Yee haw ~ more code!

- Create WeblogManagerTest - a true unit test that uses jMock for mocking dependencies
- Create WeblogManager Interface
- Create WeblogManagerImpl implementation
- Run JUnit Test
- Create "weblogManager" bean definition

So this is better, huh?

- Created 7 classes, modified 2 files, generated 1
- Faster than traditional EJB/JDBC, but still painful
- Faster options:
  - BaseDAO/Manager classes in AppFuse for Generic CRUD on any POJO
  - Generate/modify files with AppGen or AppFuse Generator

AppGen

- To create CRUD for a table, it required you create 11 files and modify 5
- AppGen requires you create 1 and modify 1
- Uses BaseHibernateDAO and BaseManager for generic CRUD methods
- Still requires you to “pretty up” the UI
- Demonstration ➪

Web Development

Creating the UI

- WeblogControllerTest and WeblogFormControllerTest
- WeblogController and WeblogFormController
- weblogList.jsp and weblogForm.jsp
- Validation Rules with XDoclet and Commons Validator
- Testing the UI with Canoo WebTest and Ant

End of Coding

- Any code tweaks you'd like to see?
- Deploying to production
    - Setup MySQL with create-tables.sql
    - Setup Tomcat with JDBC Drivers
    - Deploy using Tomcat's Manager application
    - Hosting: kgbinternet.com, contegix.com

AppFuse Roadmap

- Continue to try and make IDE integration easier
- Support/Documentation for more app servers
- iBATIS Tutorials, refactor build/test process
- Possibly an AppFuse Plugin for Eclipse
- **AppFuse 2.0**: Annotations, JSP 2.0, JDK 5
- Other things you'd like to see?

Equinox

- AppFuse Light - designed for quick apps with few requirements (i.e. prototypes)
- No build-time dependencies (i.e. XDoclet), no out-of-the-box security
- Includes 5 MVC implementations: JSF, Spring MVC, Struts, Tapestry and WebWork
- Includes 5 Persistence frameworks: Hibernate, iBATIS, JDO, OJB, Spring JDBC
- Located at http://equinox.dev.java.net

Questions?

- AppFuse Official Homepage:
  - http://appfuse.dev.java.net
- Demos and Videos:
  - http://demo.appfuse.org/demos.html
- Tutorials:
  - http://appfuse.org/tutorials
- This Presentation:
  - http://appfuse.dev.java.net/TDDWithAppFuse.pdf