Experts in delivering business-driven technology solutions.

Spring – ADD Developer

Annotated Driven Development

About Speaker

Perficient

- Speaker @ JavaOne, NFJS, Devcon, Borcon
- Sun Certified Java 2 Architect.
- Instructor for VisiBroker for Java, OOAD, Rational Rose, and Java Development.
- JBoss Certified Developer

Agenda

- Annotations and the MV
- Spring 2.5/3 Annotations
  - Spring Business Tier
- Spring Data Tier
- Spring MVC Tier
- Summary

Industry Forces

- Annotations
  - EJB 3.X
  - JSR-250 Common Annotations
  - JSR-299 Web Beans
- Guice / SEAM
- XML... way too much XML

Industry Move to Annotations

Commons Annotation

Perficient

@Resource

@PreDestroy

@PostConstruct

EJB Annotation

Perficient

Commons

@Resource

@PreDestroy

@PostConstruct

EJB Annotation

Commons

@Resource

@PreDestroy

@PostConstruct

@Home

@SecurityRoles

@Init

@EJB

@MessageDriven

@MethodPermissions

@Stateful

@Interceptor

@Inject

@RunAs

@Stateless

@TransactionManagement

@TransactionAttribute

JPA Annotation

Perficient

Commons

@Resource

@PreDestroy

@PostConstruct

EJB

@Home

@EJB

@MessageDriven

@Stateful

@Inject

@Stateless

@SecurityRoles

@RunAs

@MethodPermissions

@Init

@Interceptor

@TransactionManagement

@TransactionAttribute

JPA Annotation

Perficient

@Column

@JoinColumn

@GeneratedValue

@Entity

Commons

@Resource

@PreDestroy

@PostConstruct

@Table

@Version

@Serialized

EJB

@Home

@EJB

@MessageDriven

@Stateful

@Inject

@Stateless

@SecurityRoles

@RunAs

@MethodPermissions

@Init

@Interceptor

@TransactionManagement

@TransactionAttribute

@DisciminatorColumn

@EmbeddedId

@OneToMany

@OneToOne

@Id

@ManyToOne

@AssociationTable

@Transien

@ManyToMany

JPA Annotation

Perficient

Commons

@Resource
@PreDestroy
@PostConstruct

EJB

@Home
@EJB
@MessageDriven
@Stateful
@Inject
@Stateless
@SecurityRoles

JPA

@Entity
@GeneratedValue
@JoinColumn
@Version
@Serialized
@RunAs

@Table
@Column
@OneToMany
@OneToOne
@Id
@ManyToOne

@ManyToMany
@DisciminatorCol
@AssociationTabl
@EmbeddedId
@Transient

@MethodPermissions
@Init
@Interceptor
@TransactionManagement
@TransactionAttribute

Web Services

@Resource

@PreDestroy

@PostConstruct

@Home

@MethodPermissions

@Init

@EJB

@MessageDriven

@TransactionManagement

@Stateful

@TransactionAttribute

@Inject

@Stateless

@SecurityRoles

@WebServiceRef

@WebServiceRefs

@ManyToMany

@Table

@Column

@OneToMany

@EmbeddedId

@OneToOne

@Transient

@Entity

@ManyToOne

@Version

@Serialized

@RunAs

Web Beans

@Resource

@PreDestroy

@PostConstruct

@WebServiceRefs

@Home

@MethodPermissions

@Init

@EJB

@MessageDriven

@TransactionManagement

@Stateful

@TransactionAttribute

@Inject

@Stateless

@SecurityRoles

@New

@Out

@In

@Model

@SessionScoped

@Current

@LoggedIn

@Produces

@Interceptor

@Secure

@Decorator

@Synchronous

@Asynchronous

@ManyToMany

@Table

@Column

@OneToMany

@EmbeddedId

@OneToOne

@Transient

@Entity

@ManyToOne

@Id

@Version

@Serialized

@RunAs

Servlet 3

@New
@Resource
@In
@PreDestroy
@PostConstruct
@WebServlet
@ServletFilter
@InitParam
@WebServletContextListenter
@Out
@ManyToMany
@Table
@Home
@MethodPermissions
@OneToMany
@OneToOne
@Transient
@Entity
@Inject
@Stateless
@Decorator
@SecurityRoles
@Asynchronous
@Secure
@Version
@Serialized
@RunAs

JSR 303: Bean Validation

@New
@Resource
@In
@PreDestroy
@PostConstruct
@WebServiceRef
@SessionScoped
@Home
@MethodPermissions
@EJB
@LoggedIn
@MessageDriven
@TransactionManagement
@Stateful
@Interceptor
@TransactionAttribute
@Inject
@WebServlet
@Stateless
@Decorator
@SecurityRoles
@InitParam
@Asynchronous

@NotNull
@ConstraintValidator
@Length
@Min
@Pattern
@Size
@Valid
@NotEmpty

@Out
@ManyToMany
@Table
@Column
@OneToMany
@Id
@OneToOne
@Transient
@Entity
@ManyToOne
@GeneratedV
@ServletFilter
@Secure
@Version
@Serialized
@RunAs

JSR 303: Bean Validation

@New
@Resource
@In
@PreDestroy
@ConstraintValidator
@PostConstruct
@WebServiceRef
@SessionScoped
@Home
@RolesAllowed
@EJB
@LoggedIn
@MessageDriven
@Size
@TransactionManagement
@Stateful
@Interceptor
@TransactionAttribute
@Inject
@NotEmpty
@WebServlet
@Stateless
@Decorator
@SecurityRoles
@InitParam
@Asynchronous

@NotNull
@Out
@ManyToMany
@Table
@Column
@AssociationTable
@OneToMany
@Pattern
@EmbeddedId
@OneToOne
@Transient
@Entity
@ManyToOne
@GeneratedValue
@ServletFilter
@Secure
@Version
@Serialized
@RunAs

JSR-299 Context and DI for Java

Perficient

- @NonBinding
- @Named
- @Stereotype
- @Interceptor
- @InterceptorBindingType
- @Decorator
- @Decorates
- @ScopeType
- @ApplicationScoped
- @RequestScoped
- @SessionScoped
- @ConverationScoped
- @Dependent
- @BindingType
- @DeploymentType
- @Produces
- @Disposes
- @Specializes
- @Realizes
- @Initializer
- @New
- @Current
- @Production
- @Standard
- @Obtains
- @Initialized
- @Deployed
- @Observes
- @IfExists
- @Asynchronously
- @AfterTransactionCompletion
- @AfterTransactionFailure
- @AfterTransactionSuccess
- @BeforeTransactionCompletion
- @Fires
- @Model

Annotation Frustrations

- Not Mentioned
  - JMX 2.0
  - JAX-RS
  - JUnit 4 / TestNG
  - AOP frameworks
  - Spring

Perficient

Spring Annotations

Spring 3 - Annotation Support

Perficient

- JSR 250 - @PostConstruct, @Resource…
- JAX-WS 2.0's - @WebServiceRef
- EJB 3.0 - @EJB
- MVC annotations - @RequestParam, @RequestMapping…
- Test Enhancements - Junit 4.4 and TestNG
- Stereotypes - @Component, @Controller…
- Spring enhancements - @Autowired,
- AOP - @Configurable

Spring Annotations

- Spring 2.x Data Access Annotations
- Spring 2.x Aspects
- Spring 2.5 Context Annotations
- Spring 2.5 Stereotypes
- Spring 2.5 Factory Annotations
- Spring 2.5 MVC Annotations
- Spring 3.0 REST

Spring 2.x Data Access Annotations

Perficient

- @Transactional
  - Provides annotation driven demarcation for transactions
- @Repository
  - Indicates that a class functions as a repository or a data access object (DAO)
  - Exceptions are transparently translated
    - Springs DataAccessException Hierarchy

Spring 2.5 Stereotypes

Perficient

- @Component **
  - Indicates that a class is a component
  - Class is a candidate for auto-detection
  - Custom component extensions
- @Controller
  - Specialized Component
  - Typically used with RequestMapping annotation
  - Discussed in section on web mvc
- @Repository
  - 2.0 stereotype… previously mentioned
  - Now an extension of @Component
- @Service
  - Intended to be a business service facade

Spring 2.5 Factory Annotations

Perficient

- @Autowired
  - Marks a constructor, field, setter or config method for injection.
  - Fields are injected
    - After construction
    - Before config methods
  - @Autowired(required=false)
  - Config:
    - AutowiredAnnotationBeanPostProcessor
- @Configurable
  - Marks class as being eligible for Spring-driven configuration
  - Used with AspectJ
- @Qualifier
  - Qualifies a bean for autowiring
  - May be customized
- @Required
  - Marks a method as being injection required

Types of Injections

Perficient

- Constructor

```
public class AccountService {

    private AccountDAO dao;

     @Autowired
    public AccountService(AccountDAO dao) {
        this.dao = dao;
    }
```

- Setter

```
public class AccountService {

    private AccountDAO dao;

     @Autowired
    public void setDao(AccountDAO dao) {
        this.dao = dao;
    }
```

- Field

```
public class AccountService {

    @Autowired
    private AccountDAO dao;
```

New Injection Type

Perficient

- configuration method

```
public class AccountService {

    private AccountDAO dao;

     @Autowired
    public void init(AccountDAO dao) {
        this.dao = dao;
    }
```

- with any number of arguments

```
public class AccountService {

    private AccountDAO dao;

     @Autowired
    public void init( AccountDAO dao, LogLevel level) {
        this.dao = dao;
        level.getLabel();
    }
```

Let me Qualify that

Perficient

```
@Autowired
@Qualifier("xyzDataSourceName")
Private DataSource dataSource
```

- Or

```
@Autowired
public void init(@Qualifier("xyzNAme") DataSource
                 dataSource,  Object2 obj)
{…}
```

Perficient

DEMO

Perficient

Spring / JPA

JPA - Specification

Perficient

- Packaging
- Entities
- Entity Operations
- Queries
- Metadata
- Life-cycle Model
- Callbacks

Persistence.xml

- In the classpath under the META-INF directory.

```
<persistence-unit name="unit1" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
                <property name="hibernate.hbm2ddl.auto" value="create"/>
                <property name="hibernate.ejb.autodetection" value="class"/>
                <property name="hibernate.connection.url"
                        value="jdbc:hsqldb:hsql://localhost:1234/employee"/>
                <property name="hibernate.connection.driver_class"
                        value="org.hsqldb.jdbcDriver"/>
                <property name="hibernate.connection.username" value="sa"/>
                <property name="hibernate.connection.password" value=""/>
        </properties>
  </persistence-unit>
</persistence>
```

Customer Entity (from spec)

Perficient

```
@Entity(access=FIELD)
public class Customer {
  @Id(generate=AUTO) Long id;
  @Version protected int version;
  @ManyToOne Address address;
  @Basic String description;
  @OneToMany(targetEntity=com.acme.Order.class,
  mappedBy="customer")
  Collection orders = new Vector();

  @ManyToMany(mappedBy="customers")
  Set<DeliveryService> serviceOptions = new HashSet();
  public Customer() {}

  public Collection getOrders() { return orders; }
  public Set<DeliveryService> getServiceOptions() {
```

JPA Persistence Interfaces

Perficient

- EntityManager
  - Interface to interact with persistence context.
  - @PersistenceContext
- EntityManagerFactory
  - Creates an EntityManager
  - @PersistenceUnit

Acquiring a Manager

Perficient

- Injection in Stateless Bean

```
@PersistenceContext
public EntityManager em;
```

- OR

```
@PersistenceContext(unitName="order")
EntityManager em;
```

- From Java Application

```
EntityManagerFactory emf =
        Persistence.createEntityManagerFactory("unit1");
EntityManager em = emf.createEntityManager();
```

Spring 2 JPA Support

Perficient

- org.springframework.orm.jpa package
- Contains subset of the JPA container
- JpaDaoSupport
  - similar to other DAO support classes like HibernateDaoSupport
- LocalEntityManagerFactoryBean
  - Provides resource bootstrapping for non-jndi lookups

Spring / JPA Approaches

Perficient

- JpaDaoSupport Approach
  - Not preferred approach
  - Similar to HibernateDaoSupport
  - Requires Spring Configuration of the EntityManager
- Pure JPA Approach
  - Preferred approach
  - No spring references necessary in the code
    - with the exception of @Transactional

Approach 2: Spring / Pure JPA Configuration

- Leverage the persistence.xml in classpath:/META-INF

```
<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean" >
   <property name="persistenceUnitName" value="unit1"/>
</bean>
```

- DAO with no Spring references, however it contains @PersistenceContext annotated EntityManager

```
<bean id="conferenceDao"
class="com.codementor.jpa.domain.ConferenceDAOImpl"/>
```

- Spring configuration which injects JPA annotationed EntityManager

```
<bean
class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor‟ />
```

Pure JPA Code Example: ConferenceDaoImpl

```
package com.nfjs.jpa;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.transaction.annotation.Transactional;

public class ConferenceDAOImpl implements ConferenceDAO {

    @PersistenceContext
    private EntityManager entityManager;


    public void setEntityManager(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    ...
```

Pure JPA Spring Configuration

```
<bean id="entityManagerFactory"
  class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean" >
         <property name="persistenceUnitName" value="nfjs"/>
      </bean>

<bean id="conferenceDao" class="com.nfjs.jpa.ConferenceDAOImpl"/>

  <bean
  class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"
  />


  </beans>
```

No PU No Problem

Perficient

- The LocalContainerEntityManagerFactoryBean can be configured with all Persistent Unit information.

```
<bean id="entityManagerFactory"
 class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="jpaVendorAdapter">
   <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="showSql" value="true"/>
    <property name="generateDdl" value="true"/>
    <property name="databasePlatform"
        value="org.hibernate.dialect.HSQLDialect"/>
   </bean>
  </property>
</bean>
```

Transactions

Perficient

- XML Configuration

```
<tx:annotation-driven />
```

- Annotation

```
@Transactional(readOnly = false,
  propagation =  Propagation.REQUIRES_NEW)
Public void doSomething() {
```

** transaction manger bean id must be transactionManger or configured with the xml configuration above.

Test JPA with Spring

Perficient

```
public class SpeakerDAOTest extends AbstractJpaTests {

private SpeakerDAO speakerDao;

  public void setSpeakerDao(SpeakerDAO speakerDao) {
    this.speakerDao = speakerDao;
  }

  protected String[] getConfigLocations() {
    return new String[] {"classpath:/jpaContext.xml"};
  }

  protected void onSetUpInTransaction() throws Exception {
    jdbcTemplate.execute(
  "insert into speaker (id, name, company) values (1, 'Ken', 'CodeMentor')");
```

AbstractJpaTests Benefits

Perficient

- getConfigLocations ()
  - Separates test from production configuration
  - Allows for multiple configurations
- Injected Dependencies By Type
  - field references
- Every Test
  - Starts a Transactions
  - Rolls back Transaction
- Leverage jdbcTemplate for SQL checks

Perficient

DEMO

Spring MVC

Spring 2.5 MVC Annotations

Perficient

- @Controller
  - Stereotype used to "Controller" of MVC
  - Scanned for RequestMappings
- @RequestMapping
  - Annotates a handler method for a request
  - Very flexible
- @RequestParam
  - Annotates that a method parameter should be bound to a web request parameter
- @SessionAttributes
  - Marks session attributes that a handler uses

New Controller Issues

- Doesn’t implement an Interface
- Multiple request mappings
- High degree of flexibility

Advantages of Controller Interfaces

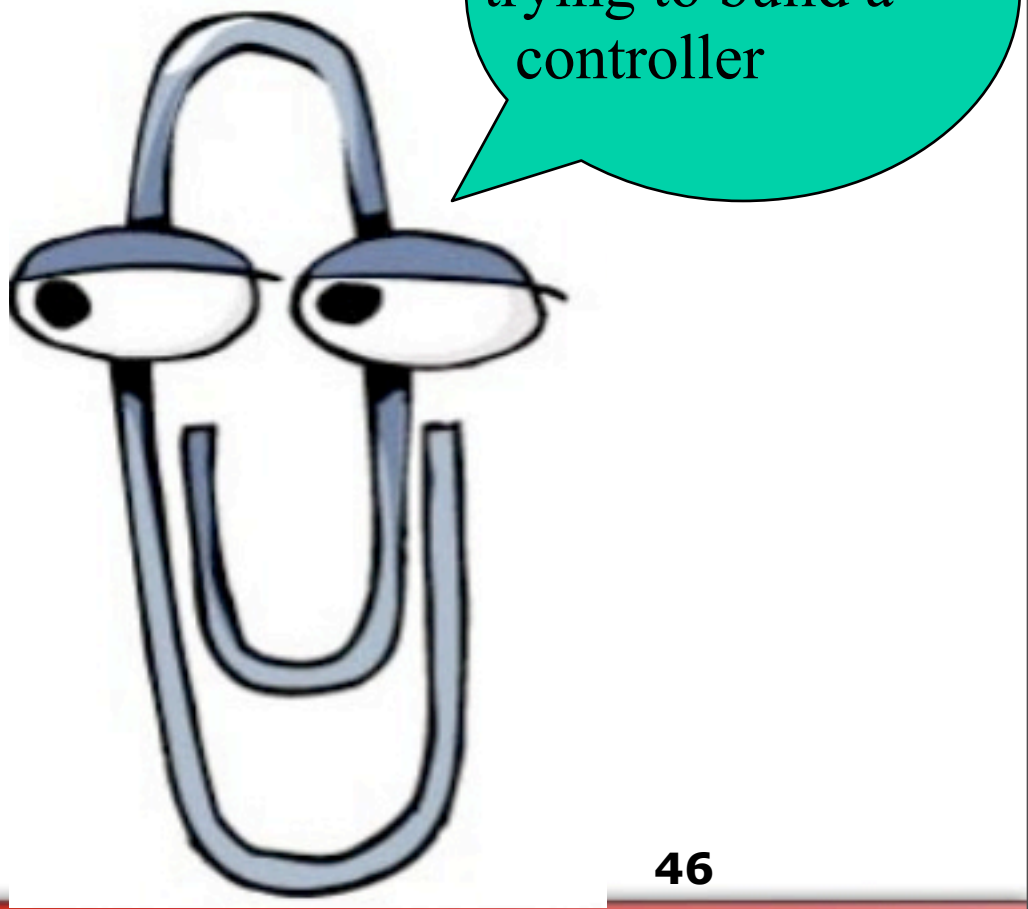
Perficient

```
public class HelloWorldController implements Controller {

}
```

Advantages of Controller Interfaces

Perficient

```
public class HelloWorldController implements Controller {

}
```



Advantages of Controller Interfaces

Perficient

```
public class HelloWorldController implements Controller {


    public ModelAndView handleRequest(HttpServletRequest httpServletRequest,
                                      HttpServletResponse httpServletResponse)
        return null;  //todo: implement me
    }
}
```

A World Without Rules

Perficient

- Return Type?
- Parameters?

@RequestMapping - Extreme Flexibility

Perficient

- Parameters can be
  - Request / response / session
  - WebRequest
  - InputStream
  - OutputStream
  - @RequestParam
  - +++
- Return types
  - ModelAndView Object
  - Model Object
  - Map for exposing model
  - View Object
  - String which is a view name
  - Void… if method wrote the response content directly

Spring 2.5 Controller Example

Perficient

```
@Controller
public class ConfController {

    @Autowired
    private confDB confDB;

    @RequestMapping("/sessionList")
    public String showSessionList(ModelMap model)    {
        model.addAttribute("sessions", this.confDB.getSessions());
        return "sessionList";
    }

    @RequestMapping("/speakerImage")
    public void streamSpeakerImage(@RequestParam("name") String name,
                                   OutputStream outputStream) throws IOException   {
        this.confDB.getSpeakerImage(name,outputStream);
    }

    @RequestMapping("/clearDatabase")
    public String clearDB() {
        this.confDB.clear();
        return "redirect:sessionList";
    }
}
```

Spring MVC By Convention

Perficient

Conventions:
hotel = HotelController
list = method

GET /hotel/list

View selected from request path

```
@Controller
public class HotelController {

   @Autowired
   private HotelService hotelService;

   @RequestMapping()
   public List<Hotel> list () {
         return hotelService.findAll();
   }
}
```

Added to Model

Multi-Action Convention

Perficient

```
@Controller
public class HotelController {

  @RequestMapping
  public void index() {}

  @RequestMapping
  public void show() {}

  @RequestMapping
  public List<Hotel> list () {
      return hotelService.findAll();
  }
```

/hotel/index

/hotel/show

/hotel/list

Working With Parameters

Perficient

```
@Controller
public class HotelController {

  @RequestMapping
  public void index() {}

  @RequestMapping
  public void show(@RequestParam Long id, Model m)  {
      m.addAttribute(hotelService.get(id));

  }
```

/hotel/show?id=42

@PathVariable - RESTFUL

Perficient

GET /owner/show/2

```
@RequestMapping(value = "/show/{id}", method = RequestMethod.GET)
public void show(@PathVariable long id ) {

}
```

Submitting Forms

Perficient

```
@RequestMapping(method = RequestMethod.POST)
public String form(Owner owner, BindingResult result, Model model) {
    Collection<Owner> results = clinic.findOwners(owner.getLastName());
    if(results.size() <1) {
        return null; //default view
    }
    if(results.size() > 1) {
        return "/owner/list";
    }
    else {
        return "redirect:/owner/show?id=" + getOwnerID(owner);
    }
}
```

Demo

Summary - ADD

I’m sorry...

Were we talking about something...

Oh Yeah...

ADD

Summary

- @nnotations
  - They @re every where!
  - They c@n incre@se productivity

Questions

Perficient

- **Please Fill Out Surveys**

kensipe@gmail.com

twitter: @kensipe

kensipe.blogspot.com