

# **The Java Persistence API (JPA)**

**Denver Java Users Group  
January 10, 2007**

**Bryan Noll  
bnoll@virtuas.com**

# ***JSR 220: Enterprise JavaBeans, Version 3.0***

- **Stated purpose...**

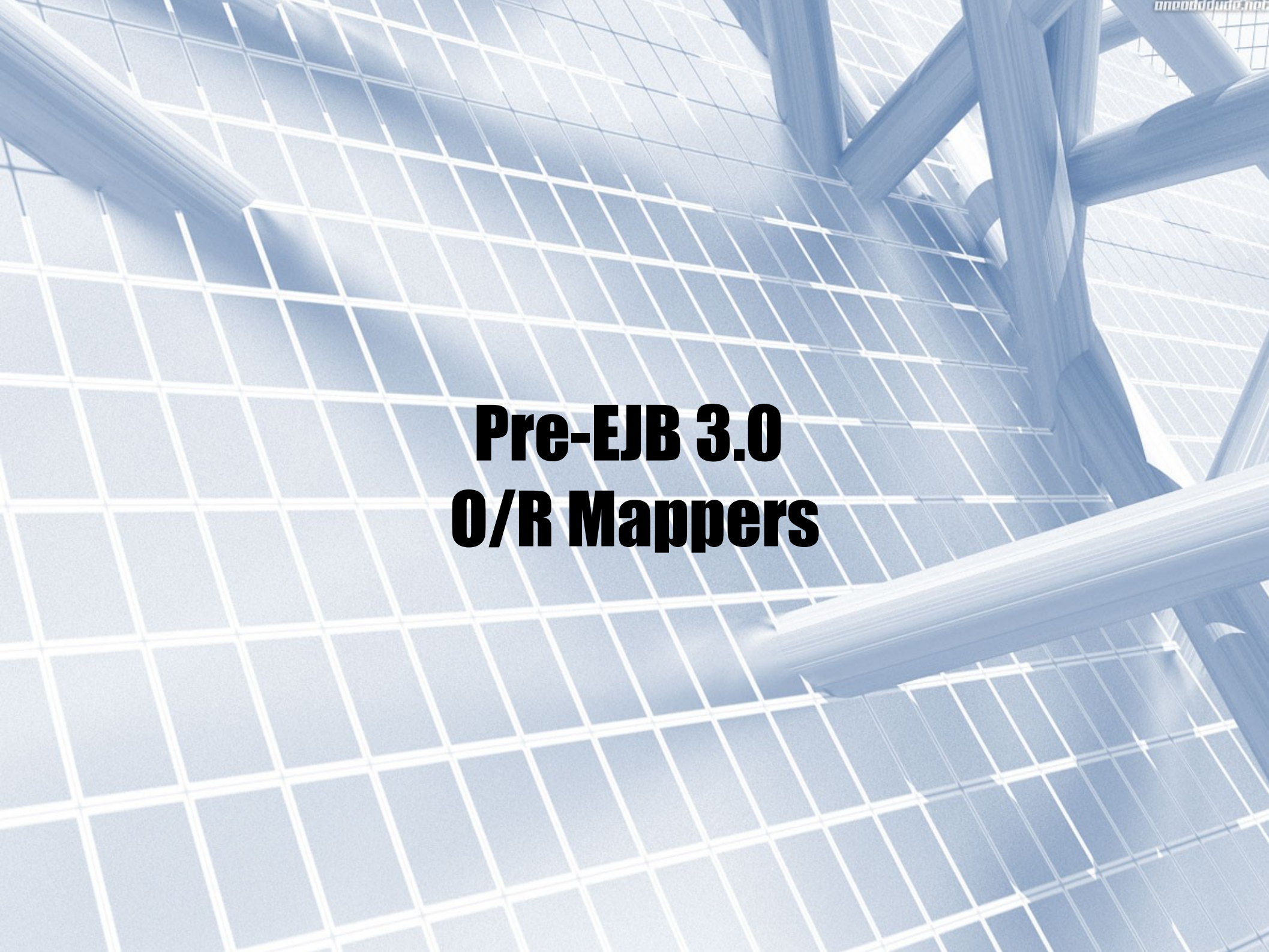
- **... to improve the EJB architecture by reducing its complexity from the EJB developer's point of view.**

- **How...**

- **It is expected that metadata attribute annotations will play a large role in this simplification. (... not limited to simplification through the use of metadata ...)**

# ***JSR 220: Enterprise JavaBeans, Version 3.0***

- **JSR 220**
  - **Java Persistence API**
  - **EJB Core Contracts and Requirements**
  
- **<http://jcp.org/en/jsr/detail?id=220>**



**Pre-EJB 3.0  
O/R Mappers**

# Kodo

- **Solarmetric founded in 2001**
- **JDO O/R Product from Solarmetric**
- **Solarmetric acquired by BEA in 2005**
- **Subset of Kodo open sourced under Apache as OpenJPA**
- **Check in of [org.apache.openjpa](http://org.apache.openjpa)\* began in Q2 of 2006**
- **Kodo 4.0 was the current version at that time**
- **Currently still part of the Apache Incubator**

# How does it stack up?

<b>Feature :</b>	<b>EJB2</b>	<b>JDO</b>	<b>JPA</b>
Java Objects	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
Advanced OO Concepts	<b>No</b>	<b>Yes</b>	<b>Yes</b>
Transactional Integrity	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
Concurrency	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
Large Data Sets	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
Existing Schema	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
Relational and Non-Relational Stores	<b>Yes</b>	<b>Yes</b>	<b>No</b>
Queries	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
Strict Standards	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
Simplicity	<b>No</b>	<b>Yes</b>	<b>Yes</b>
Meaningful Spring Support/Integration	<b>No</b>	<b>Yes</b>	<b>Yes</b>

# @Entity

```
@Entity  
public class Portfolio {  
  
    ...  
  
}
```

# @Table

```
@Entity
@Table(name = "djug_portfolio")
public class Portfolio {

    ...

}
```

# @Id

```
@Entity
@Table(name = "djug_portfolio")
public class Portfolio {

    @Id
    @GeneratedValue(strategy =
                    GenerationType.IDENTITY)
    public Integer getId() {
        return this.id;
    }

    //AUTO, TABLE, IDENTITY or SEQUENCE
}
```

# @Basic

```
@Entity
@Table(name = "djug_portfolio")
public class Portfolio {
    @Basic
    public String getName() {
        return this.name;
    }
}
```

# @Column

```
@Entity
@Table(name = "djug_portfolio")
public class Portfolio {
    @Basic
    @Column(name = "portfolio_name")
    public String getName() {
        return this.name;
    }
}
```

# @OneToMany

```
@Entity
@Table(name = "djugg_portfolio")
public class Portfolio {

    @OneToMany(mappedBy="portfolio",
               cascade={CascadeType.ALL})
    public Collection<Position> getPositions() {
        return positions;
    }
}
```

# @ManyToOne / @JoinColumn

```
@Entity
@Table(name = "djugg_position")
public class Position {

    @ManyToOne
    @JoinColumn(name = "portfolio_id")
    public Portfolio getPortfolio() {
        return this.portfolio;
    }
}
```

# @Transient

```
@Entity
@Table(name = "djugg_portfolio")
public class Portfolio {

    @Transient
    public BigDecimal getValue() {
        // biz logic to compute value of all
        // positions associated with the portfolio
    }
}
```

# @NamedQuery

```
@Entity
@Table(name="djug_security")
@NamedQuery(
    name =
        "Stock.findStockWithClosingPriceGreaterThan",
    query =
        "select theStock from Stock as theStock " +
        "where theStock.close > ?")
public class Stock extends Security {
```

# @NamedQuery - usage

```
public class BaseDaoJpa implements BaseDao {
...
    public List<Stock>
        findStockWithClosingPriceGreaterThan(BigDecimal price) {

        EntityManager em = this.entityManager;

        Query q = em.createNamedQuery(
            "Stock.findStockWithClosingPriceGreaterThan");

        // 1-based index
        q.setParameter(1, price);
        return q.getResultList();
    }
...
}
```

# **CascadeType**

**CascadeType . PERSIST**

**CascadeType . MERGE**

**CascadeType . REMOVE**

**CascadeType . REFRESH**

**CascadeType . ALL**

# Bytecode Enhancement

```
<project name="openjpa-example" default="default" basedir=".">
...
<taskdef name="openjpac-task"
  classname="org.apache.openjpa.ant.PCEnhancerTask">
  <classpath refid="project.classpath" />
</taskdef>

<target name="openjpac" depends="compile">
  <openjpac-task>
    <config propertiesFile =
      "${target.ant.classes.dir}/META-INF/persistence.xml"/>
    <classpath>
      <pathelement location="${target.ant.classes.dir}"/>
    </classpath>
    <fileset dir="${target.ant.classes.dir}"
      includes="${target.ant.model.dir}/*.class"/>
  </openjpac-task>
</target>
...
</project>
```

# /META-INF/persistence.xml

```
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  version="1.0">

  <persistence-unit name="" transaction-type="RESOURCE_LOCAL">

    <provider>
      org.apache.openjpa.persistence.PersistenceProviderImpl
    </provider>
    <!-- Enumerate your persistent classes here. -->
    <class>tutorial.persistence.model.Animal</class>
    <class>tutorial.persistence.model.Dog</class>
    <class>tutorial.persistence.model.Rabbit</class>
    <class>tutorial.persistence.model.Snake</class>

    <properties>
      <property name="openjpa.ConnectionURL"
        value="jdbc:mysql://localhost:3306/openjpa_petstore"/>
      <property name="openjpa.ConnectionDriverName"
        value="org.gjt.mm.mysql.Driver"/>
      <property name="openjpa.ConnectionUserName" value="root"/>
      <property name="openjpa.ConnectionPassword" value=""/>
    </properties>
  </persistence-unit>
</persistence>
```

# @PersistenceContext

```
public class BaseDaoJpa implements BaseDao {
    private static final Log log =
        LoggerFactory.getLog(BaseDaoJpa.class);

    private EntityManager entityManager;

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

    ...
}
```

# Spring 2.0 support

```
<bean id="entityManagerFactory"
      class=
      "org.springframework.orm.jpa.LocalEntityManagerFactoryBean"
  >
  <property name="jpaProperties">
    <props>
      <prop key="show_sql">true</prop>
    </props>
  </property>
</bean>
```

# Spring 2.0 support

```
<!--  
  PersistenceAnnotationBeanPostProcess  
  or  
-->  
<bean class =  
  "org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcesso  
  r" />
```

**Processes PersistenceUnit and PersistenceContext annotations for injection of JPA interfaces. Any such annotated fields or methods in any Spring-managed object will automatically be injected.**

# Spring 2.0 support

```
<tx:advice id="txAdvice" transaction-  
manager="transactionManager">  
  <tx:attributes>  
    <tx:method name="save*" propagation="REQUIRED" />  
    <tx:method name="update*" propagation="REQUIRED" />  
    <tx:method name="do*" propagation="REQUIRED" />  
    <tx:method name="delete*" propagation="REQUIRED" />  
    <tx:method name="*" propagation="SUPPORTS"  
      read-only="true" />  
  </tx:attributes>  
</tx:advice>
```

# Spring 2.0 support

```
<aop:config>  
    <aop:pointcut id="transactionalMethods"  
        expression=  
            "execution(* com.virtuas.hibfun.dao.*Dao.*(..))" />  
    <aop:advisor advice-ref="txAdvice"  
        pointcut-ref="transactionalMethods" />  
</aop:config>  
  
<bean id="baseDao"  
    class=  
        "com.virtuas.hibfun.dao.hibernate.BaseDaoJpa" />
```

# Limitations

- **Delete Orphan Cascading**
- **Proxying**
- **Criteria Queries**
- **IDE Support – bytecode enhancement**

# Greenfield Development

I'm starting a new project tomorrow. Should I use JPA?

<http://incubator.apache.org/openjpa/>

