JEE ORM Frameworks

UPA

Unstructured Persistence API

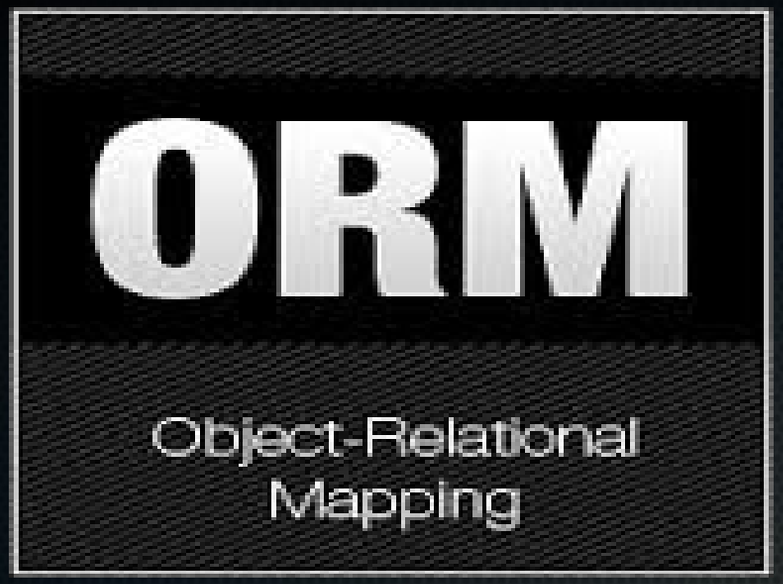
AS A CHALLENGER

JCertif

Agenda

- Part One
  - JEE and ORMs
  - Hibernate study case
  - MyBatis study case
  - JPA study case
  - Likes and Dislikes

- Part Two
  - UPA, the challenger
  - UPA in Action
  - Best Practices

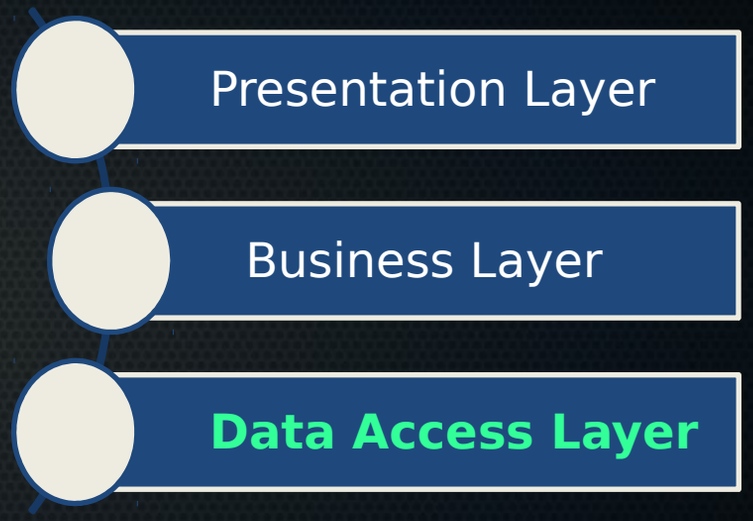


- Object Model
- Relational Storage
- Serialization/Deserialization
- CRUD operations
- Advanced features

- Benifits
  - Application Design
    - Design Patterns
  - Productivity
    - ~100% DAL
  - Code Reuse
    - <> Applications
  - Application Maintainability
    - Code strengh
    - Refactoring made easy

Application Design

- Enforces 3 Tiers Model
  - Implements Data Access Layer (DAL)
  - Provides DAO (Data Access Object) instances
  - Start coding from **Business Layer**
- Clean Implementation of Design Patterns
  - Singleton : Persistence Unit
  - Façade : Persistence Manager
  - Caching Pattern : Cached Queries/Objects
  - Others : prototype, flyweight, proxy, observer, interpreter ...



```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>SELECT Operation</title>
</head>
<body>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/TEST"
     user="root"  password="pass123"/>

<sql:query dataSource="${snapshot}" var="result">
SELECT * from Employees;
</sql:query>

<table border="1" width="100%">
<tr>
   <th>Emp ID</th>
   <th>First Name</th>
   <th>Last Name</th>
   <th>Age</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
   <td><c:out value="${row.id}"/></td>
   <td><c:out value="${row.first}"/></td>
   <td><c:out value="${row.last}"/></td>
   <td><c:out value="${row.age}"/></td>
</tr>
</c:forEach>
</table>
```

Productivity

- No more
  - SQL, but pure OO language (Java …)
    - Complex querying (joins, ….)
    - Parameter escaping (think of dates, ...)
  - Serialization (convert objects to SQL insert/update)
  - Deserialization (convert cursor/resultset to List of objects)
  - Raw JDBC, too verbose
    - try/catch/connection/preparedStatement/resultSet/while next

```java
public class SelectJDBC {

    public List<Client> findClientByName(String name) {
        List<Client> clients = new ArrayList<Client>();
        Connection c = null;PreparedStatement ps = null;ResultSet rs = null;
        try {
            try {
                c = DriverManager.getConnection("jdbc:derby://localhost:1527/example");
                ps = c.prepareStatement("Select * from Client where name=?");
                ps.setString(1, "%" + name + "%");
                rs = ps.executeQuery();
                while (rs.next()) {
                    Client client = new Client();
                    client.setId(rs.getInt(1));
                    client.setName(rs.getString(2));
                    client.setBirthDate(rs.getDate(3));
                    clients.add(client);
                }
            } finally {
                if (rs != null) {rs.close();}
                if (ps != null) {ps.close();}
                if (c != null) {c.close();}
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return clients;
    }
}
```

Code Reuse

- Multiple Applications , same schema
  - Users, Clients, Products, ...
- Avoid
  - Serialization/deserialization implemented for each use
- Single „jar“ library to include in all applications : DAL

APP 1

APP 2

APP 3

```
public class SelectJDBC {

    public List<Client> findClientByName(String name) {
        List<Client> clients = new ArrayList<Client>();
        Connection c = null;PreparedStatement ps = null;ResultSet rs = null;
        try {
            try {
                c = DriverManager.getConnection("jdbc:derby://localhost:1527/example");
                ps = c.prepareStatement("Select * from Client where name=?");
                ps.setString(1, "%" + name + "%");
                rs = ps.executeQuery();
                while (rs.next()) {
                    Client client = new Client();
                    client.setId(rs.getInt(1));
                    client.setName(rs.getString(2));
                    client.setBirthDate(rs.getDate(3));
                    clients.add(client);
                }
            } finally {
                if (rs != null) {rs.close();}
                if (ps != null) {ps.close();}
                if (c != null) {c.close();}
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return clients;
    }
}
```

Taha Ben Salah/ Jcertif 04-2014

Maintainability

- ORM implementations are
  - Robust
  - Confirmed
  - Tested
  - Proofed against
    - exotic data/storage
    - Performance issues
    - Thread Safe (*)

Maintainability

- Concentrate on Business
- Let ORM handle DAL
- Better Handling of
  - Schema changes
  - Tunings and Optimizations

What about JEE

- JEE enforces 3 tiers
- JEE application is EAR (Entreprise Archive)
  - War applications handles „Presentation Layer“
    - JSR 314: JavaServer Faces 2.0
    - JSR 315: JavaTM Servlet 3.0 Specification
  - JAR application handles business logic (jar)
    - JSR 318: Enterprise JavaBeansTM 3.1
    - Session, Message Driver beans,...
  - Persistence Framework
    - JSR 317: JavaTM Persistence 2.0

JPA or not JPA

- MyBatis
- Hibernate
- JPA
  - API (java interfaces), no implementation
  - Needs Implementation
    - Hibernate
    - EclipseLink
    - OpenJPA, …
- UPA


iBATIS


MyBatis

IBatis to MyBatis

- 2001 : iBatis (Apache Fondation)
  - https://ibatis.apache.org/
- 2010 : v3 of iBatis moved to Google Code : MyBatis
  - http://www.mybatis.org/
- As of 3.2 moved to github
  - https://github.com/mybatis
- IBatis is retired/deprecated
- Current version : MyBatis 3.2.7

„The king is dead, long live the king!“

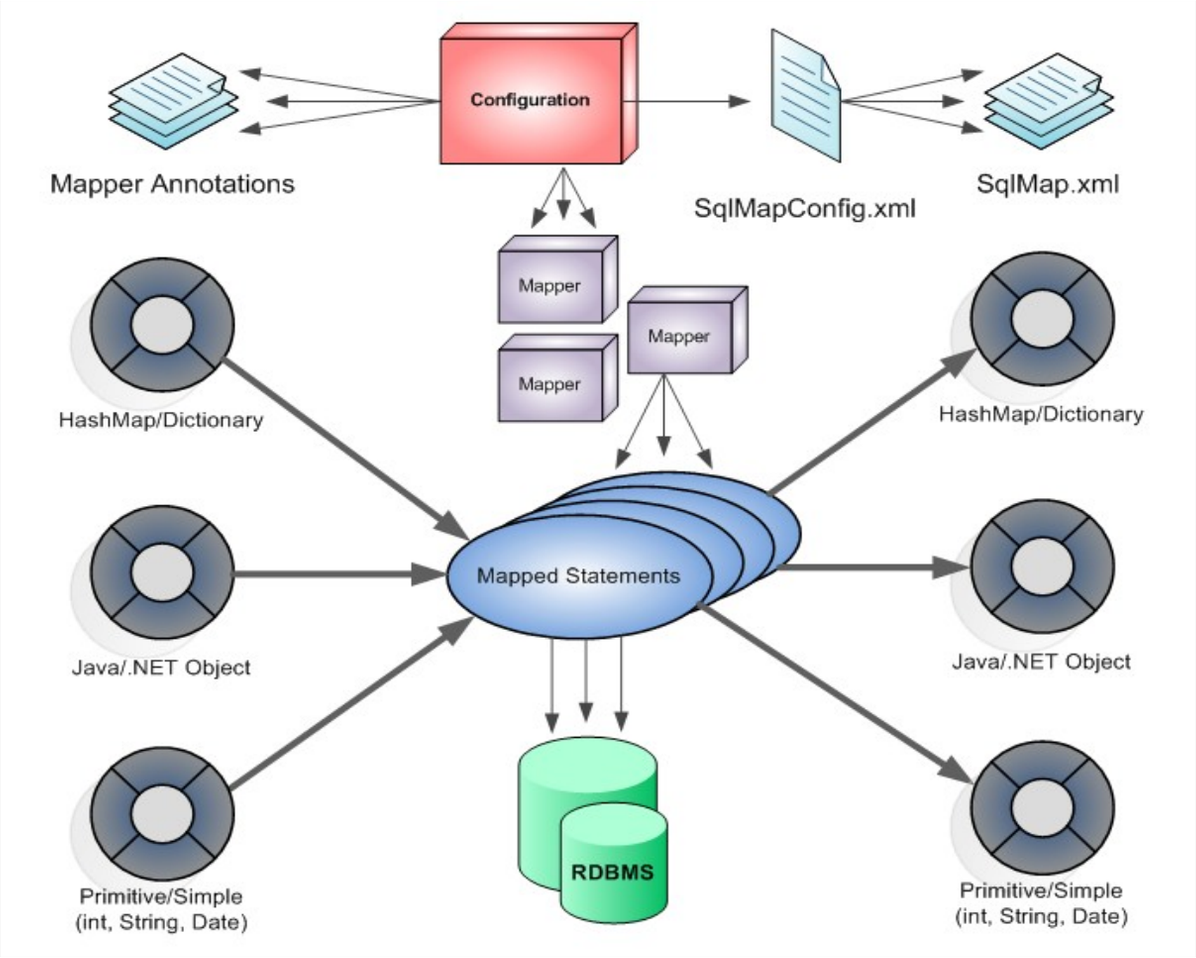
Main Features

- Open Source
- Mainly Java
- „Ported“ to .Net and Ruby
  - https://code.google.com/p/mybatisnet/
-

Main Features

- Mapping Framework, no more, no less
  - SQL, stored procedures for processing
  - Java for invoking and manipulating objects
  - XML for mapping
    - Classes/Attributes to tables/Columns
    - Methods to stored procedures/select statements
- Bottom Up Framework
  - Needs Database to be created manually
- Non portable / RDBMS specific
  - Specific SQL dialect

Architecture



Getting Started

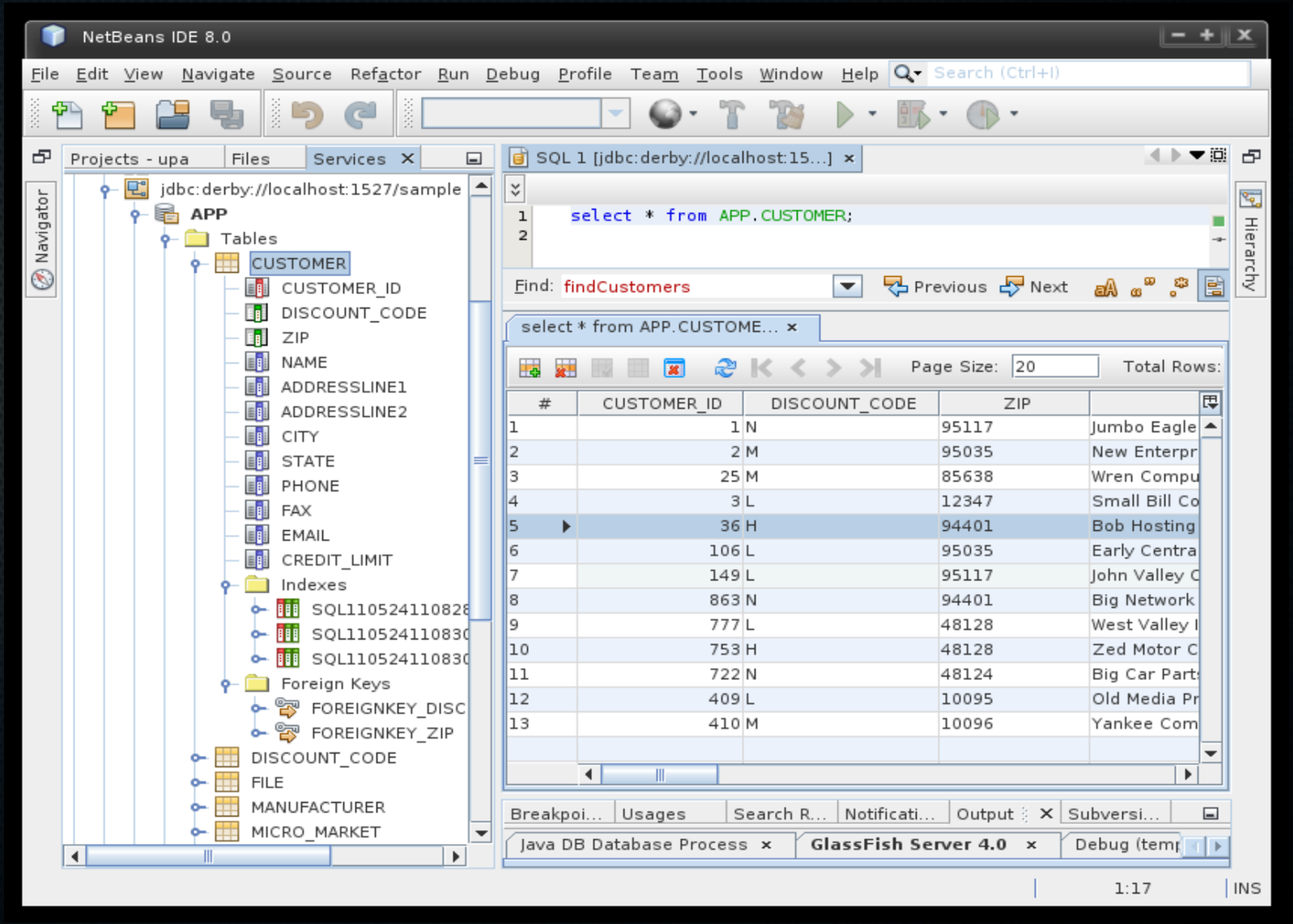
- Using Maven
- Add dependency
  to mybatis and driver
- As an example here using
  - Netbeans IDE
  - Web Application

```xml
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>6.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.2.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.9.1.0</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

MyBatis

Getting Started

- Create/Define Database
  - Create tables, stored procedures, ….
-



Getting Started

- Create Model Class
- Make valid bean
  - Empty constructor
  - Public get/set

```java
public class Customer {

    private Integer id;
    private String name;
    private String zip;

    public Integer getId() {return id;}

    public void setId(Integer id) {this.id = id;}

    public String getName() {return name;}

    public void setName(String name) {this.name = name;}

    public String getZip() {return zip;}

    public void setZip(String zip) {this.zip = zip;}

}
```

Getting Started

- Create Mapper Interface
- Java interface
- Contains DAO methods
  - Select (find)
  - Insert (create)
  - Update (update)
  - Delete (delete)
- Any name for methods

```
public interface CustomerMapper {

    Customer findCustomer(int id);

    List<Customer> findCustomers();
}
```

Getting Started

- Create Mapper XML
- CustomerMapper.xml
- Under „Other Sources“
- Contains SQL for each method in your Mapper Interface
- Contains Mapping for Results

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="temp.mybatis.example.CustomerMapper">
  <select id="findCustomer" resultType="temp.mybatis.example.Customer">
    select CUSTOMER_ID, NAME, ZIP from CUSTOMER where CUSTOMER_ID = #{id}
  </select>
  <select id="findCustomers"
          resultType="temp.mybatis.example.Customer"
          resultMap="customerMap">
    select CUSTOMER_ID, NAME, ZIP from CUSTOMER
  </select>
  <resultMap id="customerMap" type="temp.mybatis.example.Customer">
      <result property="id" column="CUSTOMER_ID"/>
      <result property="name" column="NAME"/>
      <result property="zip" column="ZIP"/>
  </resultMap>
</mapper>
```

Getting Started

- Create config XML
- mybatis-config.xml
- Under „Other Sources“
- Contains
  - database conenction info
  - List of mappers
  - Environment ...

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver"
                  value="org.apache.derby.jdbc.ClientDriver"/>
        <property name="url"
                  value="jdbc:derby://localhost:1527/sample"/>
        <property name="username" value="app"/>
        <property name="password" value="app"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="CustomerMapper.xml"/>
  </mappers>
</configuration>
```

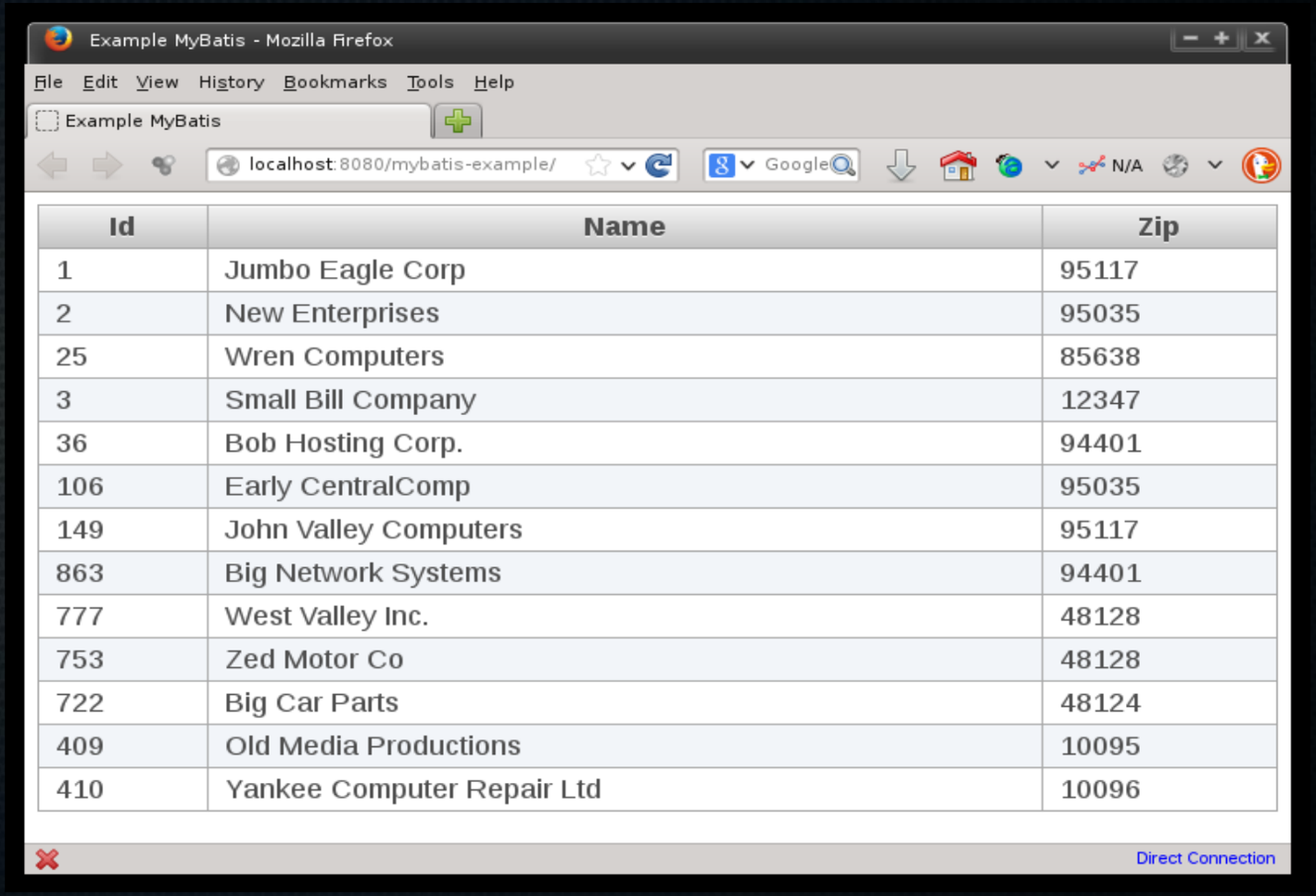
Getting Started

- Create DAL Class
- Create SqlSessionFactory
- Create Session
- Get Mapper
- Invoke
- Close session

```java
private void onCreate() {
    InputStream inputStream;
    try {
        inputStream = Resources.getResourceAsStream("mybatis-config.xml");
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    } catch (IOException ex) {
        Logger.getLogger(CustomerController.class.getName()).log(Level.SEVERE, null, ex);
    }
    SqlSession session = null;
    try {
        session = sqlSessionFactory.openSession();
        CustomerMapper mapper = session.getMapper(CustomerMapper.class);
        setCustomers(mapper.findCustomers());
    } finally {
        if (session != null) {
            session.close();
        }
    }
}
```

Getting Started

- Final Result



Id	Name	Zip
1	Jumbo Eagle Corp	95117
2	New Enterprises	95035
25	Wren Computers	85638
3	Small Bill Company	12347
36	Bob Hosting Corp.	94401
106	Early CentralComp	95035
149	John Valley Computers	95117
863	Big Network Systems	94401
777	West Valley Inc.	48128
753	Zed Motor Co	48128
722	Big Car Parts	48124
409	Old Media Productions	10095
410	Yankee Computer Repair Ltd	10096

Spring integration

- Sous projet MyBatis-Spring
- Utiliser l'inject de dépondence @AutoWired

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.2.1</version>
</dependency>
```

```
<bean id="sqlSessionFactory">
    <property name="dataSource" ref="dataSource" />
    <property name="typeAliasesPackage" value="temp.mybatis.example" />
    <property name="mapperLocations" value="classpath*:sqlMap/**/*.xml" />
</bean>

<bean id="sqlMapClient" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSessionFactory" />
</bean>
```

Advanced Features

- Advanced cache Patterns
  - Customizable
- Dynamic SQL
- Annotation Driven SQL

```
<select id="findActiveBlogLike"
      parameterType="Blog" resultType="Blog">
    SELECT * FROM BLOG
    <where>
        <if test="state != null">
            state = #{state}
        </if>
        <if test="title != null">
            AND title like #{title}
        </if>
        <if test="author != null and author.name != null">
            AND author_name like #{author.name}
        </if>
    </where>
</select>
```

```
@Select("Select * from CUSTOMER")
List<Customer> findCustomers();
```

Advanced Features

- Plugin creation
  - Intercept and MODIFY lowlevel processing of MyBatis
  - Not Recommanded

```java
@Intercepts({@Signature(
type= Executor.class,
method = "update",
args = {MappedStatement.class,Object.class})})

public class ExamplePlugin implements Interceptor {

    public Object intercept(Invocation invocation) throws Throwable
        System.out.println("Here");
        return invocation.proceed();
    }

    public Object plugin(Object target) {
        return Plugin.wrap(target, this);
    }

    public void setProperties(Properties properties) {

    }
}
```


HIBERNATE

Main Features

- First class ORM Framework
- Slogan „Hibernate. Everything data.“
- Current version 4.3.5.Final

Main Features

- Hibernate is more than ORM
  - Hibernate ORM (support JSR 317)
  - Hibernate Search
  - Hibernate Validator (support JSR303)
  - Hibernate OGM (NoSQL support)
  - Hibernate Tools
-

Main Features

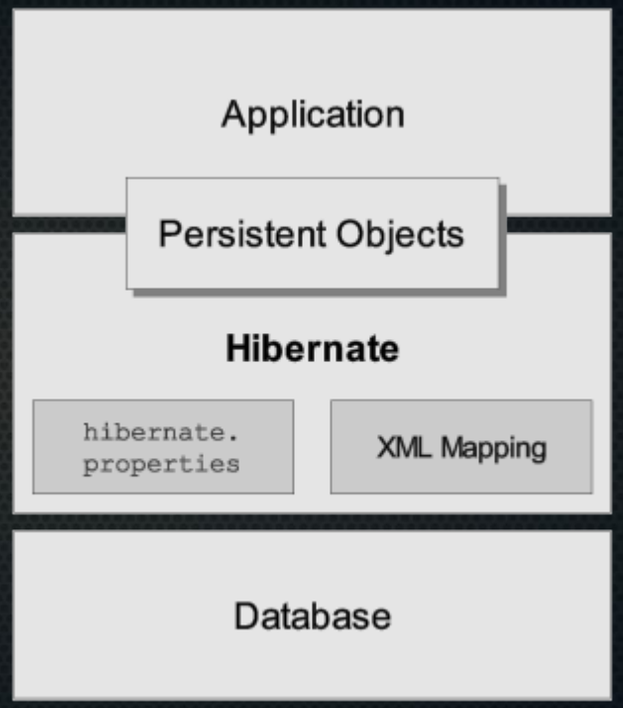
- RDBMS abstraction Layer
  - New Language : HQL
- HQL plateform independent
- Supports all major RDBS
  - MySQL, PostgreSQL
  - DB2/NT, Oracle, Microsoft SQL Server Database
  - Sybase SQL Server, Informix Dynamic Server
  - HSQL Database Engine , FrontBase

Main Features

- JPA Provider
- Idiomatic persistence
  - inheritance, polymorphism, association
  - Composition
  - and the Java collections framework
- High Performance
  - lazy initialization
  - numerous fetching strategies and optimistic locking

Main Features

- Centralized configuration
- XML/Annotation based



Main Dependencies

- hibernate-core
  - main artifact, supports annotations and hbm.xml
- hibernate-entitymanager
  - JPA implementation

Getting Started

- Using Maven
- Add dependency
  to hibernate-core and driver
- As an example here using
  - Netbeans IDE
  - Web Application
- Here primefaces and javaee-web-api are needed for web

```
<dependencies>
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>4.0</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.3.5.Final</version>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>6.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.9.1.0</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Getting Started

- Create mapping file for each entity
- File xyz.hbm.xml
- In resources
- Contains mapping for
  - Class / Table
  - Fields/Columns
  - Java Type/SQL Type
  - PK, FK, constraints

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="temp.hibernate.example.Customer" table="CUSTOMER">
        <id name="id" type="int" column="CUSTOMER_ID" >
            <generator class="native"/>
        </id>
        <property name="name"/>
        <property name="zip"/>
    </class>
</hibernate-mapping>
```

Getting Started

- Create config file
- Contains
  - Connection properties
  - Mapping hbm files references

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">
            org.apache.derby.jdbc.ClientDriver</property>
        <property name="connection.url">
            jdbc:derby://localhost:1527/sample</property>
        <property name="connection.username">app</property>
        <property name="connection.password">app</property>
        <property name="connection.pool_size">10</property>
        <property name="show_sql">true</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="hbm2ddl.auto">update</property>
        <!-- Mapping files -->
        <mapping resource="Customer.hbm.xml"/>
        <mapping resource="Discount.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Getting Started

- Create Model Class
- Make valid bean
  - Empty constructor
  - Public get/set

```java
public class Customer {

    private Integer id;
    private String name;
    private String zip;

    public Integer getId() {return id;}

    public void setId(Integer id) {this.id = id;}

    public String getName() {return name;}

    public void setName(String name) {this.name = name;}

    public String getZip() {return zip;}

    public void setZip(String zip) {this.zip = zip;}

}
```

Getting Started

- Create DAL Class
- Create SessionFactory
- Create Session
- Create query
- Invoke list()
- Close session

```
private void onCreate() {
    Session session = null;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        List<Customer> list = session
                .createQuery("Select a from Customer a")
                .list();
        setCustomers(list);
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        if (session != null) {
            session.close();
        }
    }
}
```

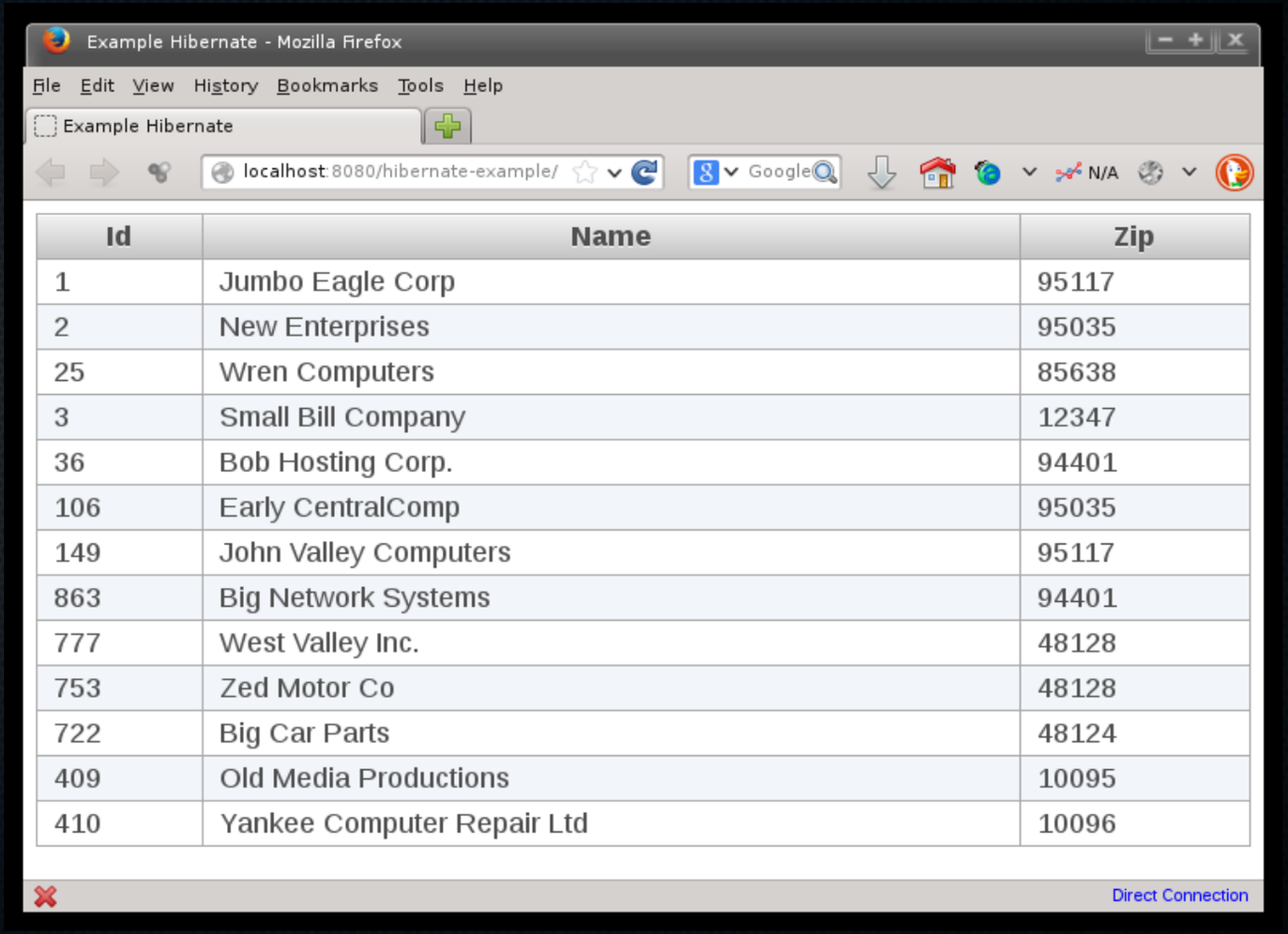
Getting Started

- You need define a helper class if not using Spring or JPA
- Helper Class enable creating Hibernate Sessions

```java
public class HibernateUtil {

    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            Configuration config = new Configuration();
            config.configure();
            StandardServiceRegistryBuilder regBuilder = new StandardServiceRegistryBuilder();
            regBuilder.applySettings(config.getProperties());
            return config.buildSessionFactory(regBuilder.build());
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

}
```

Getting Started

- Et voilà
- Same result as MyBatis



Id	Name	Zip
1	Jumbo Eagle Corp	95117
2	New Enterprises	95035
25	Wren Computers	85638
3	Small Bill Company	12347
36	Bob Hosting Corp.	94401
106	Early CentralComp	95035
149	John Valley Computers	95117
863	Big Network Systems	94401
777	West Valley Inc.	48128
753	Zed Motor Co	48128
722	Big Car Parts	48124
409	Old Media Productions	10095
410	Yankee Computer Repair Ltd	10096

Getting Started

- CRUD operations
- Persist = INSERT
- Merge = UPDATE
- Delete = DELETE

```java
private void createUpdateDelete() {
    Session session = null;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        Customer c=new Customer();
        c.setName("Hammadi");
        c.setZip("4054");
        //insert
        session.persist(c);
        //update
        c.setName("Alia");
        session.merge(c);
        //delete
        session.delete(c);
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        if (session != null) {
            session.close();
        }
    }
}
```

Getting Started

- Using Transactions
- Same Session
- Explicit Operations
- BeginTransaction
- Commit
- Rollback

```java
private void createUpdateDelete() {
    Session session = null;
    Transaction transaction=null;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        Customer c = new Customer();
        c.setName("Hammadi");
        c.setZip("4054");
        //insert
        session.persist(c);
        //update
        c.setName("Alia");
        session.merge(c);
        //delete
        session.delete(c);
        transaction.commit();
    } catch (Exception ex) {
        ex.printStackTrace();
        if (transaction != null) {
            transaction.rollback();
        }
    } finally {
        if (session != null) {
            session.close();
        }
    }
}
```

Associations

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="temp.hibernate.example.Customer" table="CUSTOMER">
        <id name="id" type="int" column="CUSTOMER_ID" >
            <generator class="native"/>
        </id>
        <property name="name"/>
        <property name="zip" length="10"/>
        <many-to-one name="discount" column="DISCOUNT_CODE"
                class="temp.hibernate.example.Discount"/>
    </class>
</hibernate-mapping>
```

Customer.hbm.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="temp.hibernate.example.Discount" table="DISCOUNT_CODE">
        <id name="code" column="DISCOUNT_CODE" />
        <property name="rate"/>
    </class>
</hibernate-mapping>
```

Discount.hbm.xml

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">
            org.apache.derby.jdbc.ClientDriver</property>
        <property name="connection.url">
            jdbc:derby://localhost:1527/sample</property>
        <property name="connection.username">app</property>
        <property name="connection.password">app</property>
        <property name="connection.pool_size">10</property>
        <property name="show_sql">true</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="hbm2ddl.auto">update</property>
        <!-- Mapping files -->
        <mapping resource="Customer.hbm.xml"/>
        <mapping resource="Discount.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

hibernate.cfg.xml

```java
public class Customer {

    private Integer id;
    private String name;
    private String zip;

    private Discount discount;

    public Discount getDiscount() { return discount; }

    public void setDiscount(Discount discount) {this.discount = discount;}
```

```java
public class Discount {

    private String code;
    private double rate;
```

Advanced Features

- Schema creation : hbm2ddl.auto
  - validate | update | create | create-drop
- hibernate-envers
  - historical auditing of changes to your entities.
- Inheritence
- Programatic configuration

```
Configuration config = new Configuration();
    config.addResource("Item.hbm.xml");
    config.addResource("Bid.hbm.xml");
```

Advanced Features

- StatelessSession
- Locking : Optimistic, Pessimistic
- Caching
- MultiTenant : limited
  - SCHAMA/DATABASE/DISCRIMINATOR (planned for v5.0)
- OSGI support
- Annotation Driven

Integration

- hibernate-c3p0
  - integration with C3P0 connection pool library
- hibernate-proxool
  - Integration with Proxool connection pool library
- hibernate-ehcache
  - Integration with EhCache, as a second-level cache.
- hibernate-infinispan
  - Integration with Infinispan, as a second-level cache.

JPA

JPA

- Open Standard
- JSR 317 (JPA 2)
- Java Interfaces (no implementation)
- Needs underlying Provider
- Annotation Driven

Getting Started

- Using Maven
- Add dependency to eclipselink and driver

```
<dependencies>
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>4.0</version>
    </dependency>
    <dependency>
        <groupId>org.eclipse.persistence</groupId>
        <artifactId>eclipselink</artifactId>
        <version>2.5.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.eclipse.persistence</groupId>
        <artifactId>org.eclipse.persistence.jpa.modelgen.processor</artifactId>
        <version>2.5.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>6.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.9.1.0</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Getting Started

- Create META-INF/peristence.xml
- In Maven, under „Other Sources“ aka main/resources

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="myPU" transaction-type="JTA">
        <properties>
            <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"
            <property name="javax.persistence.jdbc.url"
                      value="jdbc:derby://localhost:1527/sample" />
            <property name="javax.persistence.jdbc.user" value="app" />
            <property name="javax.persistence.jdbc.password" value="app" />

             <!--EclipseLink should create the database schema automatically-->
            <property name="eclipselink.ddl-generation" value="create-tables" />
            <property name="eclipselink.ddl-generation.output-mode"
                      value="both" />
        </properties>
    </persistence-unit>
</persistence>
```

Getting Started

- Using datasource ? Even easier

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persi
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persis
                 http://xmlns.jcp.org/xml/ns/persistence/persist
    <persistence-unit name="myPU" transaction-type="JTA">
        <jta-data-source>jdbc/sample</jta-data-source>
    </persistence-unit>
</persistence>
```

Getting Started

- Create Model Class
- Make valid bean
  - Empty constructor
  - Dont forget Public get/set

```
@Entity
@Table(name = "DISCOUNT_CODE")
public class Discount {

    @Id
    @Column(name = "DISCOUNT_CODE")
    private String code;
    private double rate;
```

```
@Entity
@Table(name = "CUSTOMER")
public class Customer {

    @Id
    @GeneratedValue
    @Column(name = "CUSTOMER_ID")
    private Integer id;
    private String name;
    private String zip;
    @ManyToOne
    @JoinColumn(name = "DISCOUNT_CODE")
    private Discount discount;
```

Getting Started

- Create DAL Class
- inject EntityManager
- Create query
- Invoke getResultList()

```
@PersistenceContext
EntityManager em;

@PostConstruct
private void onCreate() {
    List<Customer> list = em
            .createQuery("Select a from Customer a")
            .getResultList();
    setCustomers(list);
}
```

Getting Started

- CRUD is not any harder
- Transaction are Managed by container (JTA)
- CRUD operations
- Persist = INSERT
- Merge = UPDATE
- **Remove** = DELETE

```
private void createUpdateDelete() {
    Customer c = new Customer();
    c.setName("Hammadi");
    c.setZip("4054");
    //insert
    em.persist(c);
    //update
    c.setName("Alia");
    em.merge(c);
    //delete
    em.remove(c);
}
```

Getting Started

- Using Transactions
- Annotation Driven
- Method scoped
- Multiple Types

```
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
private void createUpdateDelete() {
    Customer c = new Customer();
    c.setName("Hammadi");
    c.setZip("4054");
    //insert
    em.persist(c);
    //update
    c.setName("Alia");
    em.merge(c);
    //delete
    em.remove(c);
}
```

- Et voilà
- Same result as MyBatis and Hibernate
- Much cleaner!

Example JPA - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Example JPA | Error - An ... | Error - An ... | Example JPA | Error - An ... | Example JPA | Examp...

localhost:8080/jpa-example/

Id	Name	Zip
1	Jumbo Eagle Corp	95117
2	New Enterprises	95035
25	Wren Computers	85638
3	Small Bill Company	12347
36	Bob Hosting Corp.	94401
106	Early CentralComp	95035
149	John Valley Computers	95117
863	Big Network Systems	94401
777	West Valley Inc.	48128
753	Zed Motor Co	48128
722	Big Car Parts	48124
409	Old Media Productions	10095
410	Yankee Computer Repair Ltd	10096

Direct Connection

Transaction Types

- MANDATORY
  - Proceed, Error
- REQUIRED
  - Proceed, Create
- REQUIRES_NEW
  - Suspend+Create+Resume, Create
-
- SUPPORTS
  - Proceed, Proceed
- NOT_SUPPORTED
  - Suspend+Resume, Proceed
- NEVER
  - Error, Proceed


success Ln
Failure Dr

What's the point ?

- All 3 technos are
  - Mature
  - Widely used
  - Feature Rich
- But Three distinct
  - Concepts
  - Targets

JPA Likes

- JPA is the only „standard“
  - Sustainability
  - Elite Community (JCR, : Oracle, IBM, HP, Microsoft, …)
  - Open Standard
  - Active Community, getting better
- Intuitive
- Small mastering curve

JPA Likes

- Well suited for „almost“ all projects
- Enforces Object Model Domain
- Scales Well
- All common features available
- Multiple providers/implementations
  - Helps tuning performance according to the implementation
  - Main Providers (Real World)
    - Hibernate
    - EclipseLink

JPA Dislikes

- More suited to Top/Down projects
- Fails to portray complex relational models
- Timid
  - SPI (Service Provider Interface)
  - Meta Model representation
- Useless callback system
- No MultiTenant support
- HelloWorld (Anti) Pattern

JPA Dislikes

- Unable to deploy twice the same app on the same schema
- Anti Design Patterns
  - Model Classes ARE also the DAO descriptors
- Lack of Security Model
- No support for
  - Stored procedures
  - Views
  - ...

Hibernate Likes

- Best performances
- Rich features
- Implements JPA
- Both XML and Annotation Driven
- Deploy same app twice needs to configure at deploy time of hbm.xml files.
- Well suited for Top/Down and Bottom/Up

Hibernate Likes

- Rich features
  - Envers (history)
  - Search (fulltext)
  - Programmatic configuration

Hibernate Dislikes

- Deploy same app twice unavailable with Annotations
- Timid
  - SPI
  - Meta Model representation
- Multi Tenant support is irrelevant
- Complex Callback System
  - Interceptor/Event (equivalent !!)

MyBatis

- Likes
  - Inevitable for complex relational model schemas (Bottom/Up)
  - Inevitable for mapping Views/Stored Procedures
  - When mastering SQL, best performances
- Dislikes
  - Too verbose
  - Non Portable
  - Error Prone
  - No support for Multi-Tenant

UPA

http://tahabensalah.net/libre-is-not-free/upa.html

UPA

- Motivation
- Common Features
- Unique Features
- Study case

UPA : Motivation

- Provide a clean way to get customizable schema for end users
  - Extra fields in Contact/Invoice
  - Extra entities at runtime
  - Add Associations for Plugin based applications
- Should handle CREATE/ALTER/DROP... as JPA handles INSERT/UPDATE/DELETE
- Better SQL Concepts handling (portably)
- Focus on
  - Productivity
  - Extensibility
- Started as Proof of Concept

UPA : Motivation

- Security Issues ?
  - May be handled in RDBMS profiling
- Gain in performance
  - Vertical Tables vs Horizontal Tables

Common Features

- Intuitive
  - Almost all JPA features/concepts are maintained
- Vendor Neutral Persistence Layer
  - helps build a persistence layer that is vendor neutral and any persistence provider can be used. Although, UPA provides a reference implementation that is particularly ready to use.
- Pluggable Providers
  - supports pluggable, third party persistence providers as it is defined as an API with a reference implementation

Common Features

- Inside/Outside containers
  - UPA application can run outside the container also. So, developers can use UPA capabilities in desktop applications also.
  - The very same code runs Inside/Outside containers. NO modification is needed.
  - Helps Sketching and Testing JEE Apps

Common Features

- Annotations based meta-data
  - No deployment descriptors required
  - Very similar of JPA's annotations.
  - Annotations defaults can be used in model class, which saves a lot of development time
  - Support for XML Mapping
  - Support for override mapping

Common Features

- Standardized ORM
  - Provides clean, easy, and standardized object-relational mapping
- Query language
  - UPQL is very powerfully query language provided by UPA providing abstraction layer over the persistence model. UPQL makes it possible to avoid specific RDBMS dialects.
  - Similar to JPQL and HQL (almost compatible)

Common Features

- Model generation
  - UPA application can be configured to generate database schema based on persistence model
- Portability
  - It is meant to be easy to switch to most performing persistence provider. You can move to any commercial persistence providers when needed

Unique Features

- Reflexion API
  - Large applications need better reflexion mechanisms to handle general purpose use cases. This feature enables developers to be aware of used data model at runtime : entities, fields, datatypes, ...
- Dynamic data definition and alteration
  - enable at runtime data structure alteration by creating new entities or altering existing entities by introducing new fields, removing some fields etc.
  - helpful for dynamic model based applications that usually uses vertical tables (columns as rows) which are of very little performance.

Unique Features

- Comprehensive model structuring
  - Persistence Group
  - Persistence Unit
  - Package
  - Module
  - Entity
  - Section
  - Field

Unique Features

- Generated & Formula fields
  - Supports natively customizable formula fields.
    - values are generated according to custom expressions and conditions.
    - Exemple : Total field
  - Support for Sequences
    - identifiers or any other field
    - of any type : integers, String, date

Unique Features

- Ready to use Entity Patterns
  - Tree Entities
    - Recursive association
    - Depth search supported
  - Singleton entities
    - Single Row
  - Union Entities
    - More powerful than „Inheritence“
  - Views (Portable)

Unique Features

- Uniform persistence context access
  - The API can be used in the very same way from
    - Java desktop application
    - Web containers
    - EJB containers
    - Spring container

Unique Features

- Shared Model
  - Classes can be mutualized for multiple Entities
  - Same Table can be described by multiple Entities
  - Example
    - generic class holder "NamedEntity" (with solely id and name) to use for all drop down components.
    - Single Table with descriminator (semantically not an inheritence)

Unique Features

- Dynamic Lazy/Partial loading
  - Runtime selection of needed fields/relations for retrieval
  - In most cases, listing entities does not require retrieval of all information
- Dynamic Partial updates
  - Better interaction with outside world

Unique Features

- Very flexible persistence processing
  - Select from View
  - Insert to 2 Tables
  - Update with stored procedure call
  - Delete with custom Java call

Unique Features

- Rich Callback System
  - Interceptors on Data Updates and Structure Alterations
  - Soft/Hard(*) Triggers
  - Updatable data in Callback context

Unique Features

- Portable support for custom and complex datatypes
  - Support for all common data types
  - Provides a portable manner to extend supported types with new custom/complex ones
  - Similar to Embedded/Embeddable features in JPA although it provides a more extensible manner

Unique Features

- Language Portable
  - Designed to work equally on Java/JEE and DotNet plateforms
  - Same code base (using j2cs tool)
- Import/Export API
  - Support for large files
  - CSV, XML, XLS, XLSX
  - Support for associations

Getting Started

- Using Maven
- Add dependency to upa-impl
- Add dependency to upa-web (if web)

```
<dependencies>
    <dependency>
        <groupId>net.vpc.upa</groupId>
        <artifactId>upa-impl</artifactId>
        <version>1.2.0.19.0</version>
    </dependency>
    <dependency>
        <groupId>net.vpc.upa</groupId>
        <artifactId>upa-web</artifactId>
        <version>1.0.0.0.4</version>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.9.1.0</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>6.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>4.0</version>
    </dependency>
</dependencies>
```

Getting Started

- Using Maven
- Add dependency to upa-impl
- Add dependency to upa-web (if web)

```
<repositories>
    <repository>
        <id>vpc</id>
        <name>vpc Maven Repository</name>
        <layout>default</layout>
        <url>http://46.105.16.224:8081/nexus/content/repositories/releases</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

```
<dependencies>
    <dependency>
        <groupId>net.vpc.upa</groupId>
        <artifactId>upa-impl</artifactId>
        <version>1.2.0.19.0</version>
    </dependency>
    <dependency>
        <groupId>net.vpc.upa</groupId>
        <artifactId>upa-web</artifactId>
        <version>1.0.0.0.4</version>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.9.1.0</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>6.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>4.0</version>
    </dependency>
</dependencies>
```

Getting Started

- Create META-INF/upa.xml
- In Maven, under „Other Sources“ aka main/resources
- upa url : simpler than of JDBC
- Still possible to use JDBC native URL

```xml
<?xml version="1.0" encoding="UTF-8"?>
<upa>
    <connection>
        <connectionString>derby:default://localhost/sample2;structure=create</connectionString>
        <userName>app</userName>
        <password>app</password>
    </connection>
</upa>
```

Getting Started

- Create Model Class
- Make valid bean
  - Empty constructor
  - Dont forget Public get/set

```java
@Entity
public class Discount {

    @Id
    private String code;
    private double rate;

    public String getCode() {
```

```java
@Entity
public class Customer {

    @Id
    @Sequence
    private Integer id;
    private String name;
    private String zip;

    @ManyToOne
    private Discount discount;

    public Discount getDiscount() {
```

Getting Started

- Create DAL Class
- No CDI : masked by UPA
- Create query
- Invoke getEntityList()

```
private void onCreate() {
    PersistenceUnit pu = UPA.getPersistenceUnit();
    List<Customer> list = pu
            .createQuery("Select a from Customer a")
            .getEntityList();
    setCustomers(list);
}
```

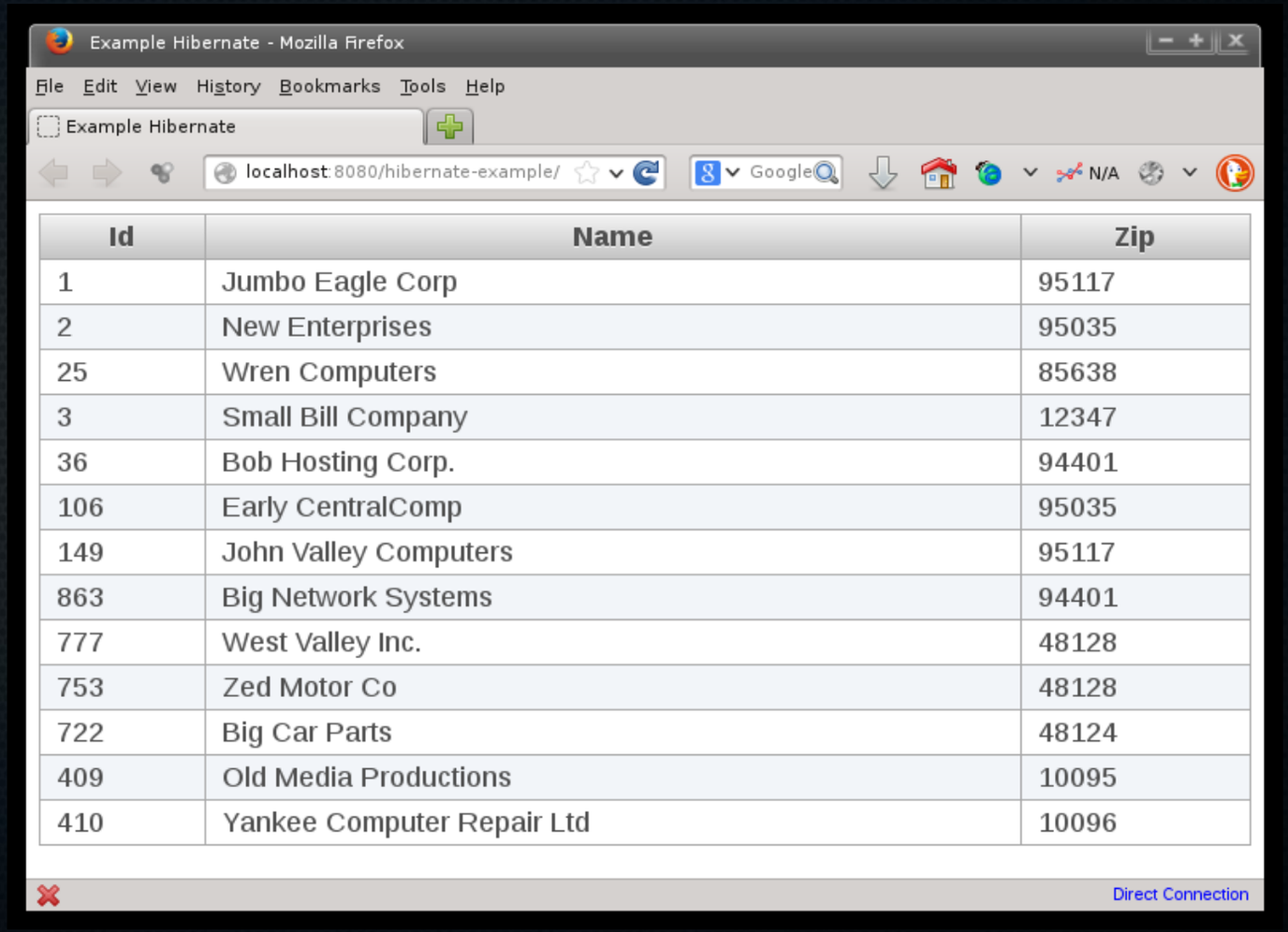
Getting Started

- CRUD is not any harder
- Transaction are Managed by container (JTA) if datasource or UPA

```
private void createUpdateDelete() {
    PersistenceUnit pu = UPA.getPersistenceUnit();
    Customer c = new Customer();
    c.setName("Hammadi");
    c.setZip("4054");
    //insert
    pu.insert(c);
    //update
    c.setName("Alia");
    pu.update(c);
    //delete
    pu.delete(c.getId());
}
```

Getting Started

- Et voilà
- Same result as MyBatis and Hibernate
- Much cleaner!



Id	Name	Zip
1	Jumbo Eagle Corp	95117
2	New Enterprises	95035
25	Wren Computers	85638
3	Small Bill Company	12347
36	Bob Hosting Corp.	94401
106	Early CentralComp	95035
149	John Valley Computers	95117
863	Big Network Systems	94401
777	West Valley Inc.	48128
753	Zed Motor Co	48128
722	Big Car Parts	48124
409	Old Media Productions	10095
410	Yankee Computer Repair Ltd	10096

Advanced Features : Partial Config

- Plain Model
- MappingObject

```
public class Invoice {

    private String invoiceId;
    private Date date;
    private Integer customerId;
    private Customer customer;
    private double totalTaxFree;
    private double totalIncludingTax;
```

```
@Entity(entityType = Invoice.class,path = "Tutorial/Sell")
public class InvoiceMO {
    @Id
    @Sequence(format = "{datepart(year,currentDate())}/{#}",initi
    private FieldDesc invoiceId;
```

Advanced Features : Formulas

- Formulas

```
@Formula("Coalesce((Select Sum(x.priceTaxFree) From InvoiceDetail x Where x.invoiceId=this.invoiceId),0)")
private FieldDesc totalTaxFree;
```

- Sequences

```
@Id
@Sequence(format = "{datepart(year,currentDate())}/{#}",initialValue=1,allocationSize=1)
private FieldDesc invoiceId;
```

Advanced Features : password

- Hash password
- D'ont change queries

```java
@Entity(entityType = User.class)
public class UserMO extends AbstractMO {

    @Field(nullable = BoolEnum.FALSE)
    private FieldDesc login;

    @Field(modifiers = FieldModifier.MAIN, nullable = BoolEnum.FALSE)
    private FieldDesc name;

    @ManyToOne
    private FieldDesc customer;

    @PasswordStrategy
    private FieldDesc password;

}
```

```java
public User authentificate(String login, String password) {
    PersistenceUnit pu = UPA.getPersistenceUnit();
    User e = pu
            .createQuery("Select a from User a where a.login=:login and a.password=:password and a.active=true")
            .setParameter("login", login)
            .setParameter("password", password)
            .getEntity();
```

Advanced Features : Alteration

- Add entities at runtime
- Add fields at runtime

```
UPA.getPersistenceUnit().addEntity(Invoice.class);
```

```
Entity entity = UPA.getPersistenceUnit().getEntity(Invoice.class);
Section tracking = entity.addSection("Tracking");
tracking.addField("creationDate", null, null, TimestampType.DEFAULT)
        .setInsertFormula("currentTimestamp()");
```

Advanced Features : Callback

- Tracking made VERY easy
- All Entities will provide five extra fields columns even if not defined in class

```
@Callback
public class TrackingFeature extends DefinitionListenerAdapter implements EntityDefinitionListener {
    @Override
    public void entityAdded(EntityDefinitionEvent event) {
        if (event.isAfter()) {
            Entity entity = event.getEntity();
            Section tracking = entity.addSection("Tracking");
            tracking.addField("creationDate", null, null, TimestampType.DEFAULT)
                    .setInsertFormula("currentTimestamp()");
            tracking.addField("creationUser", null, null, StringType.DEFAULT)
                    .setInsertFormula("currentUser()");
            tracking.addField("modificationDate", null, null, TimestampType.DEFAULT)
                    .setFormula("currentTimestamp()");
            tracking.addField("modificationUser", null, null, StringType.DEFAULT)
                    .setFormula("currentUser()");
            tracking.addField("revision", null, 0L, LongType.DEFAULT)
                    .setUpdateFormula("revision+1");
        }
    }
}
```

Advanced Features : Filters

- Make Entity with filtered Criteria
- Anonymous dynamic View
- Entity will be replaced with subsequent criteria whenever needed

```
invoice.addFilter("today", "modificationDate>=currentTimestamp()");
List<InvoiceDetail> list=UPA.getPersistenceUnit().createQuery("Select a InvoiceDetail a "
        + "where a.invoice.customer.name like '%Ben%'").getEntityList();
```

Advanced Features : MultiTenant

- Can be implemented simply using filters and callbacks
- Full Example in tutorial

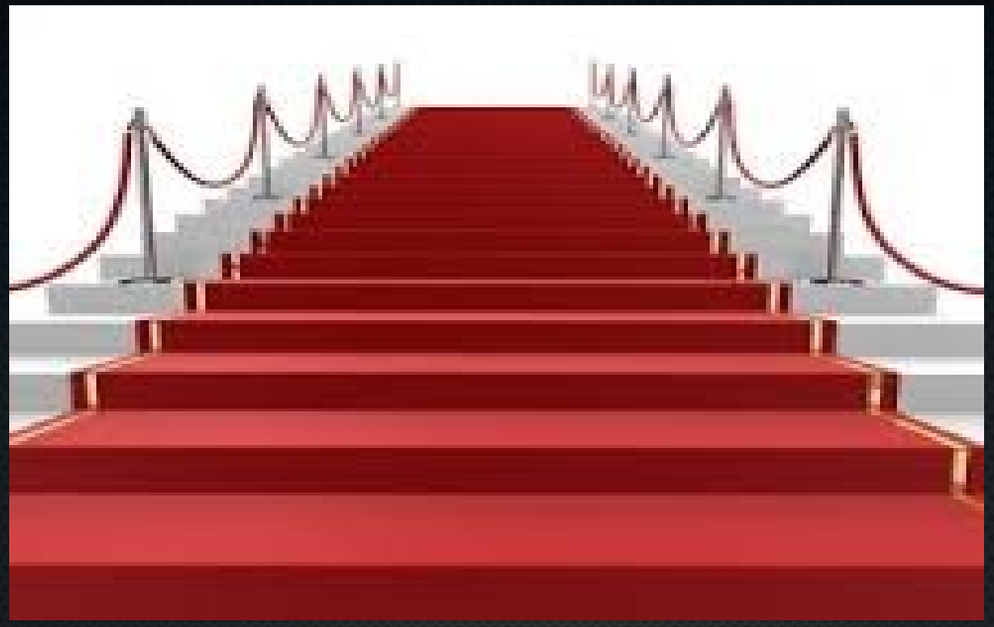
```
@Override
public void entityAdded(EntityDefinitionEvent event) {
    //tenantId field is added to all Entities except System ones (ones created by the framework)
    if (event.isAfter()) {
        Entity entity = event.getEntity();
        //define new Field which value is a formula calling the custom "CurrentTenant" function
        entity.addField("tenantId", "MultiTenant", null, null, IntType.DEFAULT).setInsertFormula("CurrentTenant()");
        // filter entities by tenantId
        entity.addFilter("MultiTenant", "tenantId=CurrentTenant()");
    }
}
```

Advanced Features : Persistent Names

- Specify locally table/column names
- Bulk (prefix/suffix) naming
  - Annotation based
  - XML Based

```
@PersistenceNames(
        value = {
            @PersistenceName(value = "CUSTOMER", type = PersistenceNameType.TABLE),
            @PersistenceName(value = "CUSTOMER_PK", type = PersistenceNameType.PK_CONSTRAINT)
        }
)
public class Customer {

    @Id
    @Sequence
    private int customerId;
```

```
@PersistenceUnit()
@PersistenceNameStrategy(
        globalPersistenceName = "APP_*",
        localPersistenceName = "F_*",
        names = {
            @PersistenceName(type = PersistenceNameType.TABLE, value = "T_{OBJECT_NAME}"),
            @PersistenceName(type = PersistenceNameType.COLUMN, value = "F_{OBJECT_NAME}"),
            @PersistenceName(type = PersistenceNameType.PK_CONSTRAINT, value = "TPK_{OBJECT_NAME}"),
            @PersistenceName(type = PersistenceNameType.FK_CONSTRAINT, value = "TFK_{OBJECT_NAME}"),
            @PersistenceName(type = PersistenceNameType.INDEX, value = "IDX_{OBJECT_NAME}"),}
)
public class ApplicationMapping {
```



Time for coding now...

http://tahabensalah.net/libre-is-not-free/upa.html

taha.bensalah@gmail.com