



Sun Java User Group Seminar

2006/07/08

EJB 3.0, Java Persistence API and
Open JavaEE GlassFish

Wonseok Kim

TmaxSoft



Agenda

- Java EE 5 Overview
- Open JavaEE GlassFish
- Tmax Application Server JEUS 6
- Java EE 5 IDE
- EJB 3.0
- Java Persistence API
- References



Java EE 5 Overview

- Goal: Make it easier to develop Java EE applications
- **Metadata @annotation** from Java SE 5
 - Simple and developer-friendly declarative model
 - Descriptors are now optional
 - application.xml, ejb-jar.xml, webserivces.xml, ...
- Simplified API, **Extensive use of defaults**
- **POJO-based programming model**, Dependency Injection, Interceptor, ...
- Components
 - EJB 3.0, JPA(Java Persistence API) 1.0
 - JAX-WS 2.0, JAXB 2.0, WSM 2.0, ...
 - Servlet 2.5, JSP 2.1, JSF 1.2, JSTL 1.2, ...



J2EE 1.4 vs Java EE 5

Application Name	Item Measured	J2EE 1.4	Java EE 5	Improvement
Adventure Builder	Number of classes	67	43	36% fewer classes
	Lines of code	3,284	2,777	15% fewer lines of code
RosterApp	Number of classes	17	7	59% fewer classes
	Lines of code	987	716	27% fewer lines of code
	Number of XML files	9	2	78% fewer XML files
	Lines of XML code	792	26	97% fewer lines of XML code



Open JavaEE GlassFish



GlassFish

<https://glassfish.dev.java.net/>

- Full Java EE 5 compatible Reference Implementation
- Open Source (CDDL) Project and Community
- Base of Java EE 5 SDK, Sun Java System App Server 9.0
- Participation from Sun, TmaxSoft, Oracle, BEA
- FCS(v1) is available now
- Maven repository for components
- Various sub-projects



GlassFish sub-projects

- Ajax (jMaki)
- Blueprints solution catalog: guidelines and best practices for Java EE applications
- Fast Infoset: Binary encoding for the XML Information Set
- Generic RA for JMS: Generic Resource Adapter for JMS
- Glassfish-samples: Relevant sample applications to demonstrate Java EE Technologies
- Glassfish-plugins: Plugins for both NetBeans and Eclipse
- JAXB: Java API for XML Binding
- JAXP: Java API for XML Parsing
- JAXR: Java API for XML Registry
- JAX-RPC: Java API for XML RPC
- JAX-WS: Java API for XML Web Services
- Phobos: A lightweight, scripting-friendly, web application environment



GlassFish sub-projects

- JSF: Java Server Faces
- SAAJ: The Standard Implementation for SAAJ
- Shoal: Java based clustering framework
- SJSXP : Sun Java Streaming XML Parser
- WSIT: Web Service Interoperability Technology
- XWSS WebServices Security
- Glassfish-Corba: CORBA ORB that is used in the GlassFish



GlassFish Status

- V1 UR1(Update Release) Bug fixes ~ Aug 2006
- V2 Enterprise features ~ Early 2007
 - New WS Stack Tango
 - Performance, Startup time
 - Load Balancing, Clustering
 - Some Scripting Support (Project Phobos)
 - Unified Test Framework (Based on TestNG)
 - Improved Usability and Error Messages
- V3
 - Architecture changes
 - More Web 2.0 features



GlassFish Community

- The Aquarium – Blog about GlassFish
<http://blogs.sun.com/roller/page/theaquarium>
- Documentation
 - Javadoc, Manual, Wiki
 - Tips and Blogs
<https://glassfish.dev.java.net/public/TipsandBlogs.html>
- Forums, Mailing-list
- Bug reports
- Code contribution
- CATfish Program - Community Adoption Team for GlassFish



Tmax Application Server JEUS 6



TmaxSoft

- The first Java EE 5 Certified Application Server!
 - Full Java EE 5 feature
 - Clustering support (Load balancing, Fail over)
 - Management feature – WebAdmin, JMX, SNMP, etc.
 - Connector to various systems
 - Localized in Korean
 - Professional Customer Service
- JEUS has highest market share in Korea
- Preview is available Now!
- GA release will be available in 1Q 2007

<http://www.tmax.co.kr>



Java EE 5 IDE

- NetBeans 5.5
 - <http://www.netbeans.org>
- Eclipse Dali Project (JPA)
 - <http://www.eclipse.org/dali/>
- Oracle JDeveloper 10.1.3
 - <http://www.oracle.com/technology/products/jdev/>
- More will be available in the future!



EJB 3.0



EJB Overview

- Distributed component model based on RMI/IIOP
- Business logic and data-access layer
- Session, Message-driven, Entity bean
- EJB Container provides managed services
 - Concurrency
 - Transaction
 - Security
 - Resource pooling
 - Environment access
 - Persistence



EJB 2.1

- Powerful, but too complex
- Complex programming model
 - Home interface - EJBHome/EJBLocalHome
 - Component interface - EJBObject/EJBLocalObject
 - EnterpriseBean interfaces
 - mandatory ejbXXX methods in bean class
 - JNDI for environment access
 - mandatory ejb-jar.xml deployment descriptor
 - vendor-specific descriptor for binding JNDI name
- Lots of boilerplate codes for bean and its client



example: EJB 2.1

// Component interface

```
public interface Hello extends EJBObject {  
    public String sayHello() throws RemoteException;  
}
```

// Home interface

```
public interface HelloHome extends EJBHome {  
    Hello create() throws RemoteException, CreateException;  
}
```



// Bean class

```
public class HelloBean implements SessionBean {  
    public String sayHello(){  
        return "Hello EJB!";  
    }  
  
    public void ejbCreate() {}  
    public void ejbRemove() {}  
    public void ejbActivate() {}  
    public void ejbPassivate() {}  
    public void setSessionContext(SessionContext sc) {}  
}
```



// **ejb-jar.xml**

```
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <enterprise-beans>
    <session>
      <ejb-name>HelloBean</ejb-name>
      <home>hello.HelloHome</home>
      <remote>hello.Hello</remote>
      <ejb-class>hello.HelloBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
```



```
<assembly-descriptor>
  <method-permission>
    <unchecked/>
    <method>
      <ejb-name>HelloBean</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
  <container-transaction>
    <method>
      <ejb-name>HelloBean</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
```



example: EJB 2.1 Client View

```
Context ic = new InitialContext();
Object objref = ic.lookup("java:comp/env/ejb/HelloEJB");
HelloHome home = (HelloHome)
    PortableRemoteObject.narrow(objref, HelloHome.class);
Hello hello = home.create();

String message = hello.sayHello();
```



EJB 3.0 Approach

- Easier and more productive way to do it
- Simplified API
- **POJO(Plain Old Java Object) style**
- Metadata Annotation
 - Required declarations on the code
 - ejb-jar.xml descriptor is now optional
 - All annotations have equivalent XML definition
- **Configuration by exception** using defaults
 - But you need to know what is default
- IoC(Inversion of Control): **Dependency Injection**
- **Interceptor** – separate cross-cutting concerns
- Container do more things for developer!



example: EJB 3.0

// Business interface

@Remote

```
public interface Hello {  
    public String sayHello();  
}
```

// No Home interface

// Bean class

@Stateless

@PermitAll

```
public class HelloBean implements Hello {  
    public String sayHello(){  
        return "Hello EJB!";  
    }  
}
```

// No ejb-jar.xml



Business interfaces

@Remote

```
public interface Hello {  
    public String sayHello();  
}
```

- Plain Java interface
 - No more extends EJBObject/EJBLocalObject
- **Analogous remote and local interfaces**
 - @Remote, @Local or default (Local)
 - Remote methods not required to throw RemoteException
 - Remote interface still need to have RMI/IIOP types
- Multiple business interfaces can be defined for a bean
- Home interface requirements is removed



Bean class

`@Stateless`

`@PermitAll`

`//@TransactionManagement(TransactionManagementType.CONTAINER)`

`//@TransactionAttribute(REQUIRED)`

`public class HelloBean implements Hello {`

`public String sayHello(){`

`return "Hello EJB!";`

`}`

`}`

- POJO class with default constructor
- Bean class can **implement business interfaces directly**
- Bean type specified by annotation or XML
 - `@Stateless`, `@Stateful`, `@MessageDriven`
- Don't need to implement EnterpriseBean interfaces
 - No more implements unused callbacks



Lifecycle Callbacks

@PostConstruct

```
private void init(){  
    ds = (DataSource)ctx.lookup("jdbc/HR");  
}
```

@PreDestroy

```
private void destroy(){  
    connection.close();  
}
```

- Specify only required events callbacks
 - @PostConstruct, @PreDestroy
 - @PrePassivate, @PostActivate
 - = ejbCreate(), ejbRemove(), ejbPassivate(), ejbActivate()
in EJB 2.1
- **Any method which has x void X() signature**
- Callbacks can be reused - Separate Callback Listener class (interceptor class)



example: EJB 3.0 Client View

```
public class HelloClient {  
    @EJB  
    private static Hello hello;  
  
    public static void main(String[] args) throws Exception {  
        String message = hello.sayHello();  
        System.out.println(message);  
    }  
}
```



Simplification of Client View

- Use of dependency injection
- Removal of Home interface
- Simple business interface view
- Removal of RemoteExceptions
 - Don't need to handle checked exceptions



Removal of Home interface

- Stateless Session Beans
 - Home interface not needed anyway
 - EJB 2.1 specification Home.create() didn't really create
 - Container creates or reuses bean instance when business method is invoked
- Stateful Session Beans
 - Container creates bean instance when business method is invoked
 - Initialization is part of application semantics
 - Don't need a separate interface for it!
- **In EJB 3.0, lookup/injection contains Home.create() logic**
- Support for legacy home interfaces (EJB 2.1 Client View)
 - By @RemoteHome, @LocalHome



Environment references

- Internal resources
 - EJBContext, TimerService, UserTransaction, ORB
 - EntityManager, EntityManagerFactory, TransactionSynchronizationRegistry...
- External resources
 - Simple environment entry, EJB, DataSource, ConnectionFactory, Queue, Topic, WebServiceRef, URL, ...
- Component has own logical namespace for environment
 - **ENC(Environment Naming Context) – `java:comp/env/`**
- Logical entries are mapped to physical resources



Environment references

- In J2EE 1.4
 - descriptor is used to define references
 - `<env-entry>`, `<ejb-ref>`, `<resource-ref>`, ...
 - Lots of JNDI lookup code to get references

`<ejb-ref>`

`<ejb-ref-name>ejb/Calc</ejb-ref-name>`

`<ejb-ref-type>Session</ejb-ref-type>`

`<home>sample.CalcHome</home>`

`<remote>sample.Calc</home>`

`</ejb-ref>`

`Context ctx = new InitialContext();`

`Object objref = ctx.lookup("java:comp/env/ejb/Calc");`

...



Environment references

- In Java EE 5
 - Define references by annotations or XML
 - **Dependency injection (Resource injection)**
 - Simple lookup - `EJBContext.lookup()`
- Annotations
 - `@Resource`
 - Simple environment entries, Connection factories, topics/queues, `EJBContext`, `UserTransaction`, etc.
 - `@EJB` - EJB references (Business or Home interfaces)
 - `@PersistenceContext` – Container-managed `EntityManager`
 - `@PersistenceUnit` – `EntityManagerFactory`
 - `@WebServiceRef` – Web Service



example: Dependency Injection

@Stateless

```
public class PayrollBean implements Payroll {
```

```
    @Resource SessionContext sc;
```

```
    @Resource UserTransaction ut;
```

```
    @Resource DataSource myDB;
```

```
    @EJB Calculator calc;
```

```
    @Resource
```

```
    private void setAnotherDB(DataSource db){...}
```

```
    @PostConstruct
```

```
    private void init(){
```

```
        connection = myDB.getConnection();
```

```
    }
```

```
}
```



Dependency Injection

- Injection type
 - Field Injection
 - Setter method Injection
- Target
 - any access modifier – even private
 - non-static (except application client)
- Occurs when instance of bean class is created
- No assumptions as to order of injection
- Optional @PostConstruct method is called when injection is complete



Simple lookup

- New EJBContext.lookup() method
- References are declared by annotations on the class or XML

```
@Stateless
```

```
@Resource(name="jdbc/myDB", type=DataSource.class)
```

```
@EJB(name="ejb/calc", beanInterface=Calculator.class)
```

```
public class PayrollBean implements Payroll {
```

```
    @Resource SessionContext sc;
```

```
    public void doSomething(){
```

```
        DataSource myDB = (DataSource)sc.lookup("jdbc/myDB");
```

```
        Calculator calc = (Calculator)sc.lookup("ejb/calc");
```

```
        ...
```

```
    }
```

```
}
```



Environment references Mapping

- Mapping is still required for external resources
 - EJB, DataSource, ConnectionFactory, Queue/Topic, URL...
- Mapping is not standard part, but vendor-specific part
- Injection entry has also default ENC name
 - e.g.) java:comp/env/sample.PayollBean/calc
- Mapping in Vendor-specific DD

```
<ejb-ref>  
  <ref-name>ejb/Calc</ref-name>  
  <export-name>CalcEJB</export-name>  
</ejb-ref>
```

```
<res-ref>  
  <ref-name>jdbc/myDB</ref-name>  
  <export-name>oracleDatasource</export-name>  
</ref-ref>
```



Environment references Mapping

- Spec provides Optional mapped-name element
 - vendor-specific element
- Vendor provides default mapping
 - useful inside same module/application
 - need to know what is default and do not rely on it!

```
@EJB(mappedName="CalcualtorBean")  
Calculator calc;
```

```
@Resource(mappedName="oracleDatasource")  
DataSource myDB;
```

```
<ejb-ref>  
  <ejb-ref-name>ejb/Calc</ejb-ref-name>  
  <mapped-name>CalculatorBean</mapped-name>  
</ejb-ref>
```



Interceptors

- Derived from AOP concept
- Handles **Cross-cutting Concerns**
 - Profiling, exception handling, parameter checking, custom security, transaction management
- Scope: Default, Class, Method-level interceptors
- Interception type - @AroundInvoke
 - before and after bean method invocation
- Interceptor instances have same lifecycle as bean instance
- Multiple interceptors may decrease runtime performance



@Stateless

@Interceptors({ProfilingInterceptor.class, AnotherInterceptor.class})

public class PayrollBean implements Payroll {...}

public class ProfilingInterceptor {

@EJB Profiler profiler;

@AroundInvoke

public Object profile(InvocationContext invocation) throws Exception {

long start = System.currentTimeMillis();

try {

return invocation.proceed();

} finally {

long time = System.currentTimeMillis() - start;

profiler.log(invocation.getMethod(), time);

}



Transaction - CMT

- Container-Managed Transaction (default)
 - Declaratively specified for each business method
 - More easily specifiable by annotations
 - `@TransactionAttribute(REQUIRED)` is now default

@Stateless

```
//@TransactionManagement(TransactionManagementType.CONTAINER)
```

```
//@TransactionAttribute(REQUIRED)
```

```
public class PayrollBean implements Payroll {
```

```
    public int getTaxDeductions(int empId){...}
```

```
    @TransactionAttribute(MANDATORY)
```

```
    public void setTaxDeductions(int empId, int deductions){...}
```

```
}
```



Transaction - BMT

- Bean-Managed Transaction
 - By using UserTransaction API
 - Programmatically control of transaction
 - Useful for tx across method-calls in stateful session bean

@Stateless

`@TransactionManagement(TransactionManagementType.BEAN)`

```
public class PayrollBean implements Payroll {
```

```
    @Resource UserTransaction ut;
```

```
    public void setTaxDeductions(int empId, int deductions){
```

```
        ut.begin();
```

```
        ...
```

```
        ut.commit();
```

```
    }
```

```
}
```



Security

- Method permissions
 - Security roles that are allowed to execute methods
 - By @RolesAllowed, @PermitAll, @DenyAll
- Caller principal
 - Security principal under which a method is executed
 - Runtime security role determination
 - isCallerInRole, getCallerPrincipal
 - @DeclareRoles to define logical roles
- @RunAs – switch role
- Annotations can be specified on class/method level
- No defaults
- Need to be overridden by descriptor for deployed environment



Security

@Stateless

@PermitAll

```
public class PayrollBean implements Payroll {
```

```
    public double getAverage(){...}
```

```
    @RolesAllowed({"HR_Manager", "HR_Admin"})
```

```
    public double getSalary(int empId){...}
```

```
    @RolesAllowed({"HR_Manager"})
```

```
    public void setSalary(int empId, double salary){...}
```

@DenyAll

```
    public void xxx(){...}
```



System Exceptions

- In EJB 2.1
 - Remote system exceptions were checked exceptions: subtypes of `java.rmi.RemoteException`
 - Local system exceptions were unchecked exceptions: subtypes of `EJBException`
 - Runtime exception is always wrapped by system exception
- In EJB 3.0, same unchecked exceptions
 - Independent of whether remote or local
 - Subtypes of `EJBException`
 - `NoSuchEJBException`, `EJBTransactionRequiredException`, `EJBTransactionRolledbackException`, `EJBAccessException`, `ConcurrentAccessException`



Application Exceptions

- Business logic exceptions
- Can be checked or unchecked
- @ApplicationException or XML definition
 - Can be applied to unchecked (RuntimeException)
 - transaction for rollback attribute (default is false)

@ApplicationException

```
public class InvalidIdException throws RuntimeException {...}
```

@Stateless

```
public class PayrollBean implements Payroll {  
    public int getTaxDeductions(int empId) throws InvalidIdException {  
        //...  
        throw new InvalidIdException("Id " + empId + " is invalid");  
    }  
}
```



Stateful Session Bean

- Conversational state – preserved across method-calls
- Session is created by lookup or injection
- Session is removed by remove method or timeout
 - @Remove for remove method



@Stateful

```
public class CartBean implements Cart {
```

```
...
```

@Remove

```
public void checkout(){...}  
}
```

```
@EJB Cart cart;
```

```
//or
```

```
Cart cart = (Cart)ctx.lookup("ejb/Cart");
```

```
cart.addItem(item1);
```

```
cart.checkout();
```

```
// cart session is removed.
```



Message-driven Bean

- Asynchronous component which handles JMS or other messages
- Also POJO
 - No need to implement MessageDrivenBean
 - Implement Message listener interface directly
 - Or designates with @MessageListener
- MDB configuration by annotation
 - @MessageDriven, @ActivationConfigProperty



```
@MessageDriven(
```

```
    activationConfig = {
```

```
        @ActivationConfigProperty(propertyName="destinationType",  
            propertyValue="javax.jms.Queue"),
```

```
        @ActivationCongigProperty(propertyName="acknowledgeMode",  
            propertyValue="Auto-acknowledge")
```

```
    })
```

```
public class MessageServiceBean
```

```
    implements javax.jms.MessageListener {
```

```
    @Resource MessageDrivenContext ctx;
```

```
    public void onMessage(Message msg) {...}
```

```
}
```



Deployment Descriptors

- Alternative to annotations
 - Some developers prefer them
- Some configurations are only in descriptor
 - Default interceptors
- Override annotations
- Useful for deferred configuration
 - Security permissions
- Can be sparse, full (metadata-complete)
- Vendor-specific descriptors is still needed for server-specific configurations (optional)
 - jeus-ejb-dd.xml, sun-ejb-jar.xml



Java Persistence API



JPA Overview

- POJO-based persistence model
 - Unit-testable outside container
 - Usable as DTO(Data Transfer Object)
- Replacement of Entity bean
- Similar to existing solution - Hibernate, Oracle TopLink
- Standard O/R mapping
 - Annotations and/or XML
- Object inheritance, Enhanced Query, Native Query support
- Based on JDBC and only for relational database
 - SQL is generated for databases



JPA Overview

- Support for Java EE and Java SE
- No more tiers – database is accessed in its tier
- Support for optimistic locking
- Pluggability of third-party persistence providers
- Persistence operations should be done by application code – not automatic
- Not too complex, but need to learn



Persistence providers

- Implementation engine
- In Java EE 5, default provider is used if none specified
- Provider can be specified per persistence unit
- Availables
 - GlassFish Toplink Essentials
 - Oracle Toplink
 - <http://www.oracle.com/technology/products/ias/toplink/>
 - Hibernate EntityManager
 - <http://www.hibernate.org>
 - BEA Kodo
 - <http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/kodo/>



Entity

- Domain model object to be mapped to data model
- POJO with no-arg constructor
 - Created by new operator
 - No required interfaces
 - Fields or properties are mapped to DB
 - Have **Persistent identity** (Primary Key)
 - Can have inheritance relationship
 - extends other entities, mapped superclasses, ordinary classes
- Specified by annotation or XML
 - `@Entity(name="xxx")`
 - `<entity name="xxx" class="class_name">`



example: Ordinary POJO

```
public class Customer implements Serializable {  
    protected Long id;  
    protected String name;  
  
    public Customer() {}  
  
    public Long getId() {return id;}  
    protected void setId(Long id) {this.id = id;}  
  
    public String getName() {return name;}  
    public void setName(String name) {this.name = name;}  
}
```



example: Entity

@Entity

```
public class Customer implements Serializable {
```

@Id

```
protected Long id;
```

```
protected String name;
```

```
public Customer() {}
```

```
public Long getId() {return id;}
```

```
protected void setId(Long id) {this.id = id;}
```

```
public String getName() {return name;}
```

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

```
}
```



O/R mapping

- Specified by annotations, XML or using both
- **Defaults for minimal configuration**
 - table, column names
 - mapped attributes(fields or properties)
 - Still need to know what is default value
- Mappings are specified on
 - field -> field access type
 - getter method -> property access type



O/R mapping

- Logical annotations
 - @Id, @EmbeddedId, @IdClass
 - @Basic, @Lob, @Temporal, @Enumerated, @Version, @Transient
 - @Embedded, @Embeddable
 - @GeneratedValue
- Physical annotations
 - @Table, @SecondaryTable, @Column
 - @SequenceGenerator, @TableGenerator
 - @NamedNativeQuery
- Physical mappings are preferred to be specified in XML to be portable across DB



example: O/R mapping

@Entity

@Table(name="CUST")

```
public class Customer implements Serializable {
```

```
    @Id @GeneratedValue
```

```
    @Column(name="CUST_ID")
```

```
    protected Long id;
```

```
    protected String name;
```

```
    @Embedded
```

```
    protected Address address;
```

```
    @Transient
```

```
    protected int orderCount;
```

```
    public Customer() {}
```

```
    ...
```

```
}
```



@Embeddable

```
public class Address implements Serializable {  
    protected String street;  
    protected String city;  
    protected String state;  
    protected Integer zip;  
  
    public Address () {}  
  
    ...  
}
```



Relationships

- Represents data model association(1:1, 1:N, M:N, N:1)
- In domain model, entity-to-entity relationship is expressed by a reference or Collection of references
 - JPA support for Collection, Set, List, Map types
- ORM annotations
 - @OneToOne, @OneToMany, @ManyToMany, @ManyToOne
 - @JoinTable, @JoinColumn
 - @MapKey, @OrderBy...
- Support for cascading – persist, remove, merge, refresh
- EAGER vs. LAZY fetching
 - @OneToMany(fetch=FetchType.EAGER)



example

```
@Entity public class Customer {  
    @Id protected Long id;  
    protected String name;  
    @OneToMany(cascade={PERSIST, MERGE, REFRESH},  
        fetch=FetchType.EAGER, mappedBy="customer")  
    protected Set<Order> orders;  
    ...  
}
```

```
@Entity public class Order {  
    @Id protected Long id;  
    @ManyToOne @JoinColumn(name="CUST_ID")  
    protected Customer customer;  
    ...  
}
```



Inheritance

- Can extend
 - Other (abstract/concrete) entity class
 - Ordinary class
 - superclass fields are not persistent
 - Mapped superclass @MappedSuperclass
- Polymorphic query is supported
- Entity-entity inheritance
 - Inheritance strategies
 - @Inheritance(strategy=SINGLE_TABLE)
 - @Inheritance(strategy=JOINED)
 - @Inheritance(strategy=TABLE_PER_CLASS)
 - Annotations
 - @DiscriminatorColumn, @DiscriminatorValue
 - @PrimaryKeyJoinColumn



example – single table

@Entity

@DiscriminatorColumn(name="DTYPE", discriminatorType=STRING)

```
public abstract class Employee {  
    @Id protected Integer empId;  
    @ManyToOne protected Address address;  
    ...  
}
```

@Entity

```
public class FullTimeEmployee extends Employee {  
    protected Integer salary;  
    ...  
}
```

@Entity

```
public class PartTimeEmployee extends Employee {  
    protected Float hourlyWage;  
    ...  
}
```



example – Joined strategy

@Entity

@Inheritance(strategy=JOINED)

```
public abstract class Employee {  
    @Id protected Integer empId;  
    @ManyToOne protected Address address;  
    ...  
}
```

@Entity

```
public class FullTimeEmployee extends Employee {  
    protected Integer salary;  
    ...  
}
```

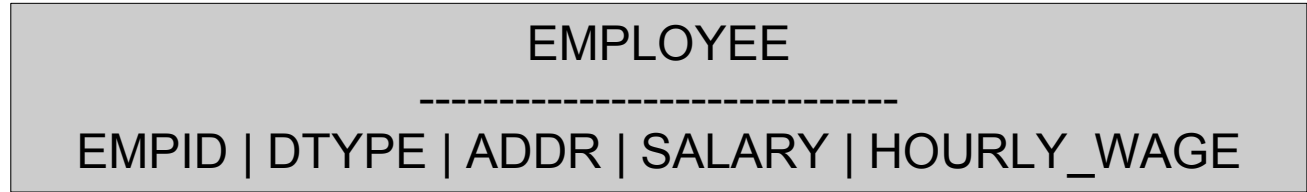
@Entity

```
public class PartTimeEmployee extends Employee {  
    protected Float hourlyWage;  
    ...  
}
```

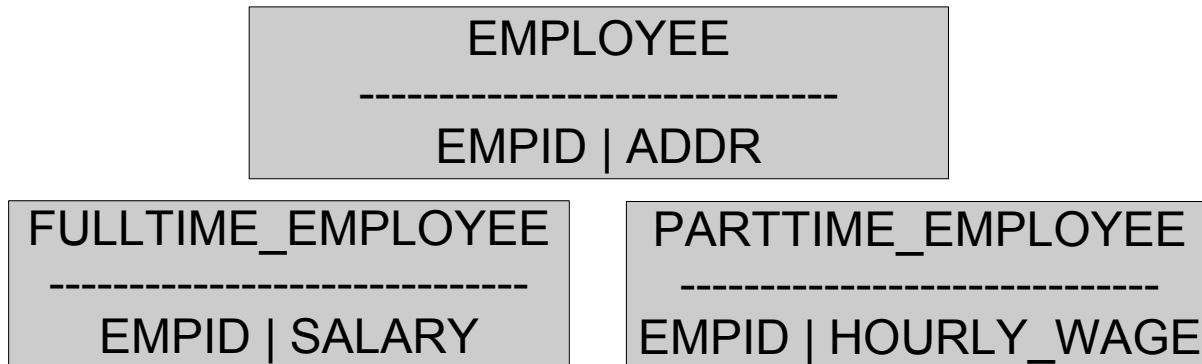


Data Models

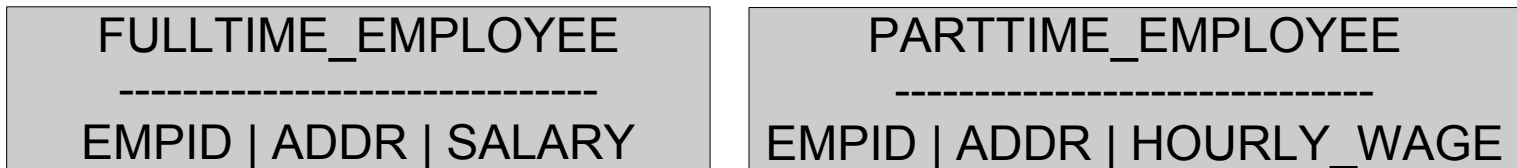
- Single table



- Joined



- Table per concrete class (optional)





EntityManager API

- Simple and central interface to manipulate entities
- Application code use this to persist to/load from DB
- Entity lifecycle operations
 - `persist()`, `remove()`, `refresh()`, `merge()`
- Finder operations
 - `find()`, `getReference()`
- Methods to create queries
 - `createNamedQuery()`, `createQuery()`, `createNativeQuery()`
- Other operations
 - `contains()`, `flush()`, `clear()`, `lock()`, ...
- EntityManager instance is not thread-safe



example

```
@PersistenceContext EntityManager em;
```

```
//persist
```

```
public Order addNewOrder(Customer customer, Product product) {  
    Order order = new Order(product);  
    customer.addOrder(order);  
    em.persist(order);  
    return order;  
}
```

```
//find
```

```
public Customer findCustomer(Long customerId) {  
    Customer customer = em.find(Customer.class, customerId);  
    return customer;  
}
```



//update

```
order.setDeliveredDate(date);
```

```
...
```

```
public Order updateOrder(Order order) {
```

```
    return em.merge(order);
```

```
}
```

//remove

```
public void deleteOrder(Long orderId) {
```

```
    Order order = em.find(Order.class, orderId);
```

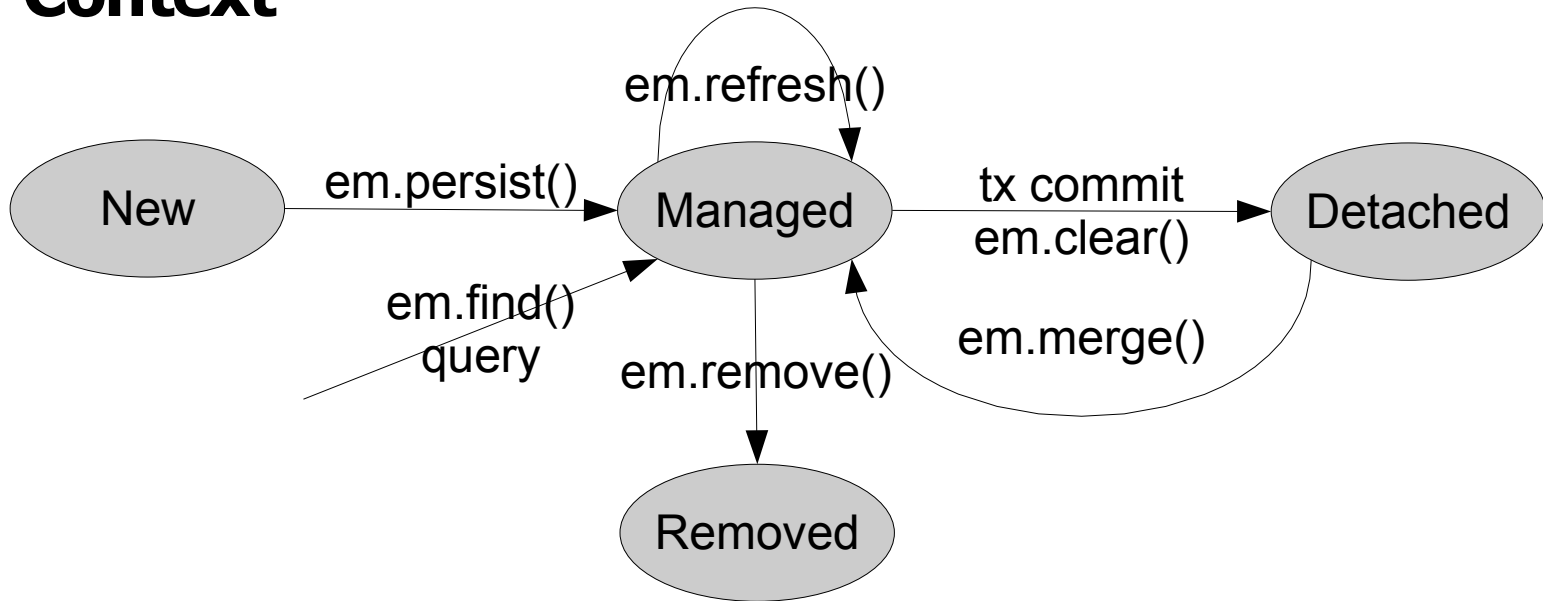
```
    em.remove(order);
```

```
}
```



Entity Lifecycle

- An Entity instance becomes **managed** if entity manager persisted or loaded it
- Managed entities are linked to database
- The set of managed entity instances called **Persistence Context**





Entity Lifecycle

- New
 - New entity instance is created
 - Entity is not yet managed or persistent
- persist/Managed
 - Entity becomes managed
 - Entity becomes persistent in database on transaction commit
- remove/Removed
 - Entity is deleted from database on transaction commit
- refresh
 - Entity's state is reloaded from database
- merge
 - State of detached entity is merged back into managed entity



Lifecycle callbacks

- @PrePersist: upon invocation of persist operation
- @PostPersist: after database insert
- @PreRemove: upon invocation of remove operation
- @PostRemove: after database delete
- @PreUpdate: before database update
- @PostUpdate: after database update
- @PostLoad: after loading the instance
- Can be defined on entity class or separate listener class



Lazy fetching and detachment

- Entities can be detached to be transferred as DTO
- Lazy fetched attributes should not be used after detachment
- Lazy fetch is default for @OneToMany, @ManyToMany relationship - Collections type
- Simple types can be @Basic(fetch=FetchType.LAZY) but not recommended unless type is large object
- Lazy fetch is a hint to persistence engine



example

```
@Entity public class Customer {  
    ...  
    @OneToMany // Lazy fetch is default  
    protected Set<Order> orders;  
    ...  
    public Set<Order> getOrders(){  
        return orders;  
    }  
}
```

```
Customer customer = em.find(Customer.class, customerId);  
//after tx finishes  
for(Order order : customer.getOrders()){  
    // causes problem!  
}
```



Queries

- Query type
 - Java Persistence QL – Object query (from EJBQL)
 - Also support for **Native query** - DB-specific SQL
- Dynamic query – on-the-fly in runtime
- Static query – Named query
 - Defined by annotations or XML
 - @NamedQuery, @NamedNativeQuery
- Query objects are created from EntityManager
- Query API
 - setParameter(), setMaxResults(), setFirstResult()
 - getResultList(), getSingleResult(), executeUpdate()



example

//Dynamic Query

```
Query query = em.createQuery("select c from Customer c where  
    c.name=:name");  
query.setParameter("name", name);  
List list = query.getResultList();
```

//Static Query

```
@Entity
```

```
@NamedQuery(name="findCustomerByName", query="select c from  
    Customer c where c.name=:name")
```

```
public class Customer {..}
```

```
Query query = em.createNamedQuery("findCustomerByName");  
query.setParameter("name", name);  
List list = query.getResultList();
```



Java Persistence Query Language

- SQL-like object model query language
- An extension of EJBQL
 - Projection list
 - Explicit JOINS
 - Subqueries
 - GROUP BY, HAVING
 - EXISTS, ALL, SOME/ANY
 - Bulk UPDATE, DELETE



example

//Query: Projection

```
SELECT e.name, d.name FROM Employee e JOIN e.department d
WHERE e.status = 'FULLTIME'
```

```
SELECT new com.example.EmployeeInfo(e.id, e.name, e.salary,
    e.status, d.name)
FROM Employee e JOIN e.department d
WHERE e.address.state = 'CA'
```

//Query: Subqueries

```
SELECT DISTINCT emp FROM Employee emp
WHERE EXISTS (
    SELECT mgr FROM Manager mgr
    WHERE emp.manager = mgr AND emp.salary > mgr.salary
)
```



example

//Query: Joins

```
SELECT DISTINCT o FROM Order o JOIN o.lineItems l JOIN l.product p  
WHERE p.productType = 'shoes'
```

```
SELECT DISTINCT c FROM Customer c JOIN FETCH c.orders  
WHERE c.address.city = 'San Francisco'
```

```
SELECT DISTINCT c FROM Customer c LEFT JOIN FETCH c.orders  
WHERE c.address.city = 'San Francisco'
```

//Query: Group by, Having

```
SELECT p.category, avg(p.price) FROM Products p  
GROUP BY p.category HAVING p.category IN ('clothing', 'shoes',  
'jewelry')
```



example

//Query: Update, Delete

```
UPDATE Employee e SET e.salary = e.salary * 1.1  
WHERE e.department.name = 'Engineering'
```

```
DELETE FROM Customer c  
WHERE c.status = 'inactive' AND c.orders IS EMPTY AND c.balance = 0
```



How to use in Java EE?

- Can be used in EJB tier or Web tier (Servlet, JSP, JSF)
- **Application-managed EntityManager**
 - The lifecycle of EntityManager - EntityManagerFactory
 - Transaction – JTA or Resource-local
 - Persistence Context - EntityManager.clear()
- **Container-managed EntityManager**
 - Container manages lifecycle of EntityManager, Transaction
 - JTA Transaction only
 - Transaction-scoped persistence context (default)
 - Extended persistence context (only in stateful session bean)
 - Keep entities managed across multiple transaction



example: app-managed EM(1)

```
// EntityManagerFactory injection
@PersistenceUnit(unitName="HR")
private EntityManagerFactory emf;

//...
// resource-local transaction is used
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();
tx.begin();

em.persist(customer);

tx.commit();
em.close();
```



example: app-managed EM(2)

```
// ENC entry is defined for lookup
@PersistenceUnit(name="emf/HR", unitName="HR")
public class HelloServlet extends HttpServlet {
    EntityManagerFactory emf;
    @Resource UserTransaction ut;

    @PostConstruct
    private void _init(){
        //lookup ENC entry
        InitialContext context = new InitialContext();
        emf = (EntityManagerFactory)
            context.lookup("java:comp/env/emf/HR");
        //...
    }
}
```



example: app-managed EM(2)

```
// JTA transaction is used  
EntityManager em = emf.createEntityManager();  
ut.begin();  
em.joinTransaction();  
  
em.persist(customer);  
  
ut.commit();  
em.close();
```



example: container-managed EM

```
public class CustomerServiceEJB {  
    // EntityManager is injected  
    @PersistenceContext(unitName="HR")  
    private EntityManager em;  
  
    public Customer createCustomer(long id, ...){  
        ...  
        em.persist(customer);  
        return customer;  
    }  
}
```



Packaging

- Persistence Unit
 - **collection of entity classes**
 - described by META-INF/persistence.xml metadata
 - identified by unit name
 - one data-source (one PU => one DB)
 - persistence provider
 - provider-specific properties
 - scope - EAR, EJB, WAR or AppClient-level
- Entities can be packaged into
 - EJB JAR file
 - WAR WEB-INF/classes, WAR WEB-INF/lib/xxx.jar
 - AppClient JAR file
 - JAR file inside EAR



example: persistence.xml

```
<persistence version="1.0"
  xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="HR" transaction-type="JTA">
    <jta-data-source>jdbc/sample</jta-data-source>
    <properties> <!-- vendor-specific -->
      <property name="toplink.ddl-generation" value="create-tables"/>
    </properties>
  </persistence-unit>
</persistence>
```



How to use in Java SE?

- Stand-alone application can use JPA provider
- **Application-managed EntityManager only**
 - Resource-local transaction (JDBC transaction)
 - EntityTransaction API for controlling transaction
- Using Bootstrap API - `javax.persistence.spi.Persistence`
- `persistence.xml` should be META-INF/ on the classpath
 - Should enumerate all entity classes
 - provider-specific database configurations
- Default persistence provider is selected from classpath if not specified
- About legacy Java EE Server (e.g J2EE 1.4)?
 - Java SE mode can be used in any environment



example: Usage in Java SE

```
EntityManagerFactory emf =  
    Persistence.createEntityManagerFactory("HR");  
EntityManager em = emf.createEntityManager();  
EntityTransaction tx = em.getTransaction();  
tx.begin();  
...  
em.persist(customer);  
...  
tx.commit();  
em.close();  
emf.close();
```

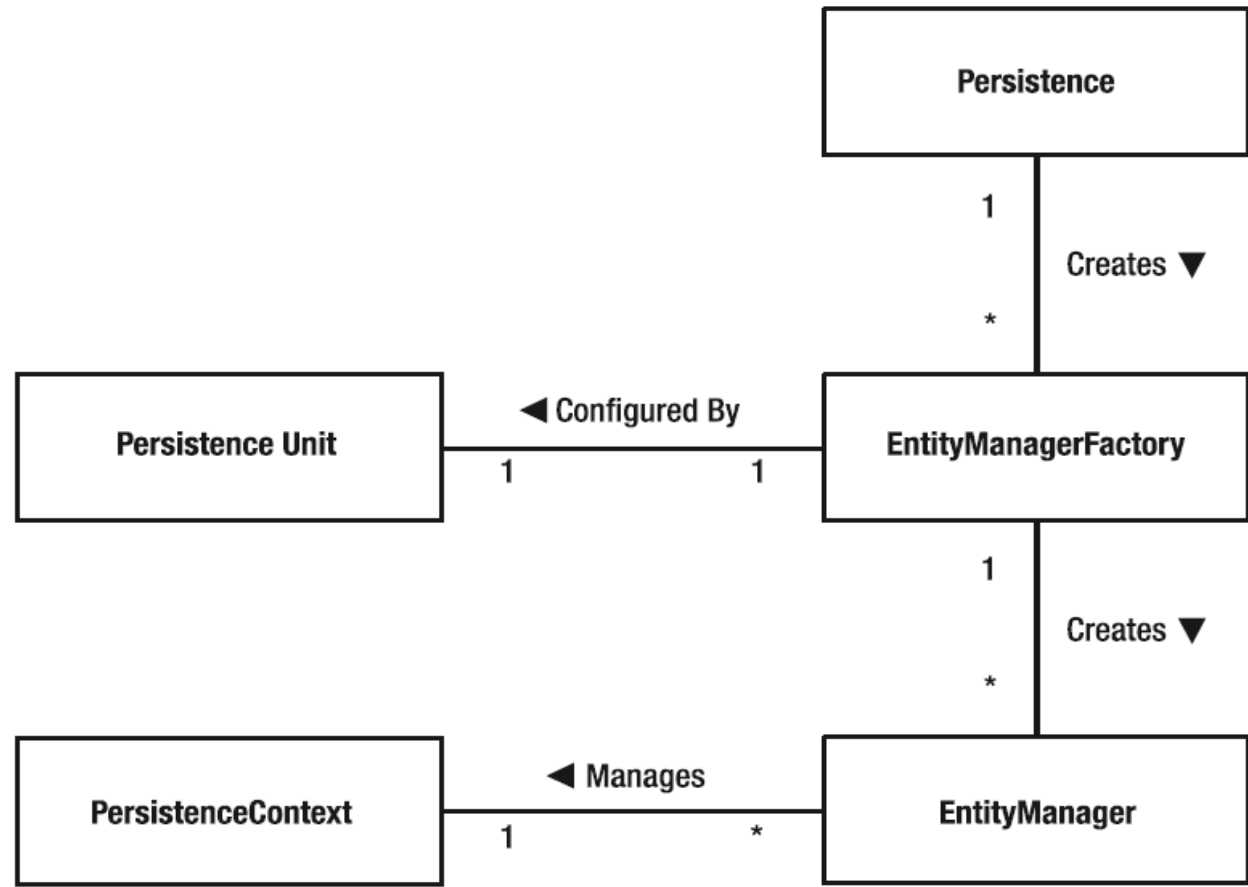


example: persistence.xml (2)

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">  
  <persistence-unit name="HR">  
    <!-- All entity classes should be listed -->  
    <class>sample.entities.Employee</class>  
    <class>sample.entities.Department</class>  
    <class>sample.entities.Team</class>  
    <properties>  
      <property name="toplink.ddl-generation" value="create-tables"/>  
      <property name="toplink.jdbc.driver"  
value="org.apache.derby.jdbc.ClientDriver"/>  
      <property name="toplink.jdbc.url"  
value="jdbc:derby://localhost:1527/sample;create=true"/>  
      <property name="toplink.jdbc.user" value="app"/>  
      <property name="toplink.jdbc.password" value="app"/>  
      <property name="toplink.logging.level" value="INFO"/>  
    </properties>  
  </persistence-unit>  
</persistence>
```



JPA API relationship



Source: Pro EJB 3 JPA, Mike Keith, Apress



References: EJB 3.0 Spec

- JSR 220 - <http://www.jcp.org/en/jsr/detail?id=220>
- EJB 3.0 is composed of three spec documents
 - EJB 3.0 Simplified API (59p) – simple and easy
 - Core Contracts and Requirements (562p) – reference
 - Java Persistence API 1.0 (256p)



References: JavaOne Sessions

- TS-3396 EJB 3.0 Specification
- TS-1365 Extending EJB 3.0 Specification With Interceptors
- TS-3395 Java Persistence API
- TS-9056 Java Persistence API In 60 Minutes
- TS-1887 The Java Persistence API in the Web Tier
- TS-1969 Blueprints for Using the Simplified Java™ EE 5 Programming Model
- TS-3616 Building EJB 3.0 Based Applications
- TS-1624 Writing Performant EJB Beans in the Java EE 5 Using Annotations
- TS-3274 Project GlassFish, Building a Java EE 5 Application Server



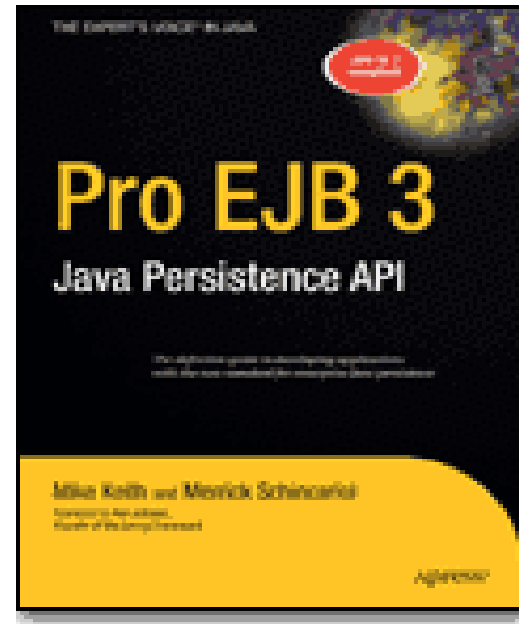
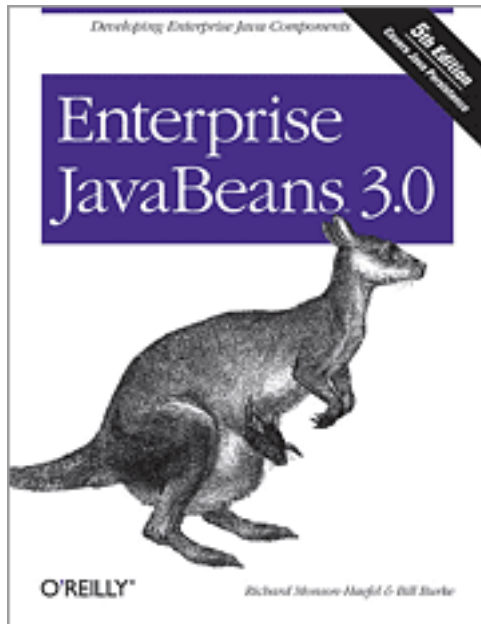
References

- 2006 JavaOne Online Technical Sessions – audio, PDFs
 - <http://developers.sun.com/learning/javaoneonline>
- Wonseok Kim's Blog - <http://weblogs.java.net/blog/guruwons/>
- Java EE Official Site - <http://java.sun.com/javaee/>
- An Introduction to the Java EE 5 Platform
 - http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/
- The Java Persistence API - A Simpler Programming Model for Entity Persistence
 - <http://java.sun.com/developer/technicalArticles/J2EE/jpa/>
- Java EE 5 Tutorial - <http://java.sun.com/javaee/5/docs/tutorial/doc/>
- Magazine: Microsoft 2006 July Cover Story – Java EE 5



EJB 3.0 Books

- Enterprise Java Beans 3.0 5th ed.
Bill Burke and Richard Monson-Haefel, O'Reilly
- Pro EJB 3 Java Persistence API
Mike Keith and Merrick Schincariol, Apress





Q&A

`guruwons at tmax.co.kr`