Java Persistence API

2006/12/02
SCA Conference

김원석

TmaxSoft

Agenda

- JPA Overview
- What JPA is
  - Entity
  - O/R Mapping
  - Relationships
  - persistence.xml
  - EntityManager API
  - Queries
- How to use
  - Java SE mode
  - Java EE mode
  - Spring's JPA Support
- Advanced Topics
- Summary & References

기존 Persistence 기술

- JDBC
- iBatis SQL Maps Framework
- EJB Entity Bean (BMP, CMP)
- JDO (Java Data Objects)
- O/R Mapping Layer – Hibernate, TopLink
- DAO (Data Access Object) – Spring Framework
  - Supports JDBC, iBatis, JDO, Hibernate, etc.

JPA Overview

- 기존 ORM 솔루션과 유사 - Hibernate, TopLink
- EJB 3.0 spec에서 제정 - EJB Entity Bean을 대체
- POJO 기반의 persistence 모델
  - 원격 컴포넌트가 아닌 로컬 Object
  - 더이상 EJB container에 의존하지 않음
  - Container 외부에서 Unit 테스트 가능
  - DTO(Data Transfer Object)로 사용 가능
- Java EE 와 Java SE 환경 모두 지원
- JDBC 기반의 Persistence Layer 이며 RDB만 지원
  - SQL 이 DB 종류에 맞게 자동 생성됨
- 원하는 persistence providers 를 플러그인 해서 사용 가능

JPA Overview

- O-R Mapping - Annotations, XML 을 통한 표준 mapping
  - Object 상속관계 등 풍부한 매핑
  - Object-level query
  - DB 종류에 독립적
  - Multiple tables, Composite keys, Native Query(SQL)
  - Optimistic locking, pessimistic locking(optional)
- Runtime performance - SQL 최소화
  - SQL 최적화
  - Lazy fetching, Changeset, Batch update
  - Caching (대부분의 read operations에 유용)
- application code가 직접 persistence operations 을 수행해야함 – EJB CMP 처럼 내부적으로 수행되는 방식이 아님
- 그다지 복잡하지는 않지만 배워야 할 필요는 있음

Persistence Providers

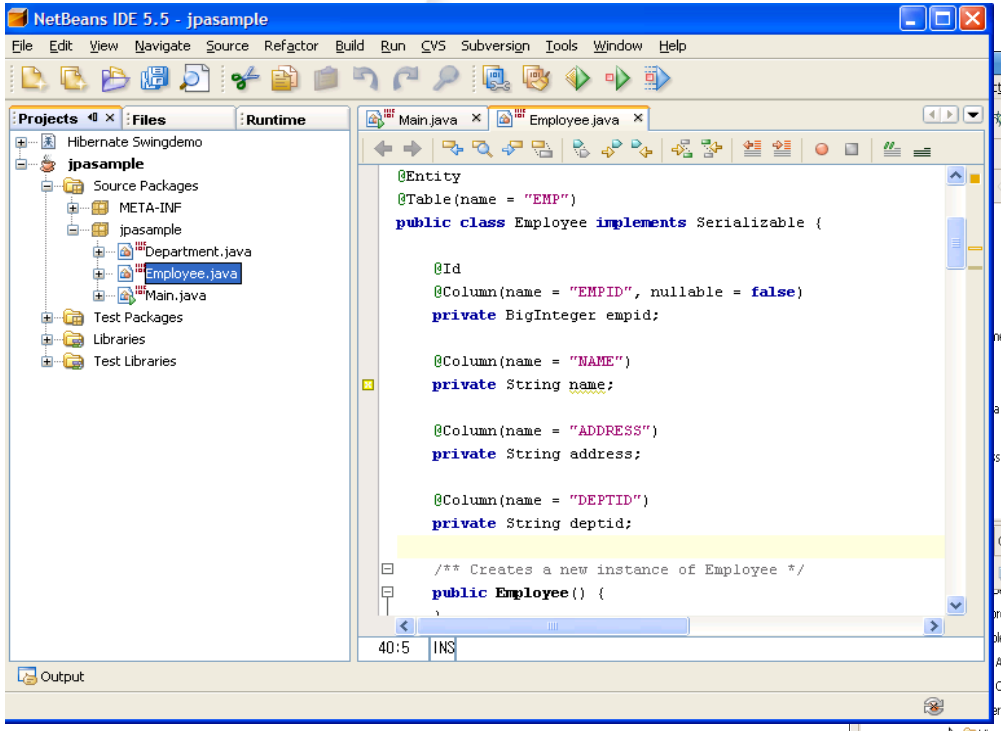
- 구현 엔진
- Java EE 5 AppServer에서는 default provider가 제공됨
- persistence unit 별로 provider 를 지정할 수 있음
- 현재 사용 가능한 구현체
  - GlassFish JPA TopLink Essentials - http://glassfish.dev.java.net
  - Hibernate EntityManager - http://www.hibernate.org
  - Apache OpenJPA - http://incubator.apache.org/openjpa/
  - Oracle TopLink(Commercial)
  - BEA Kodo(Commercial)

GlassFish JPA – TopLink Essentials

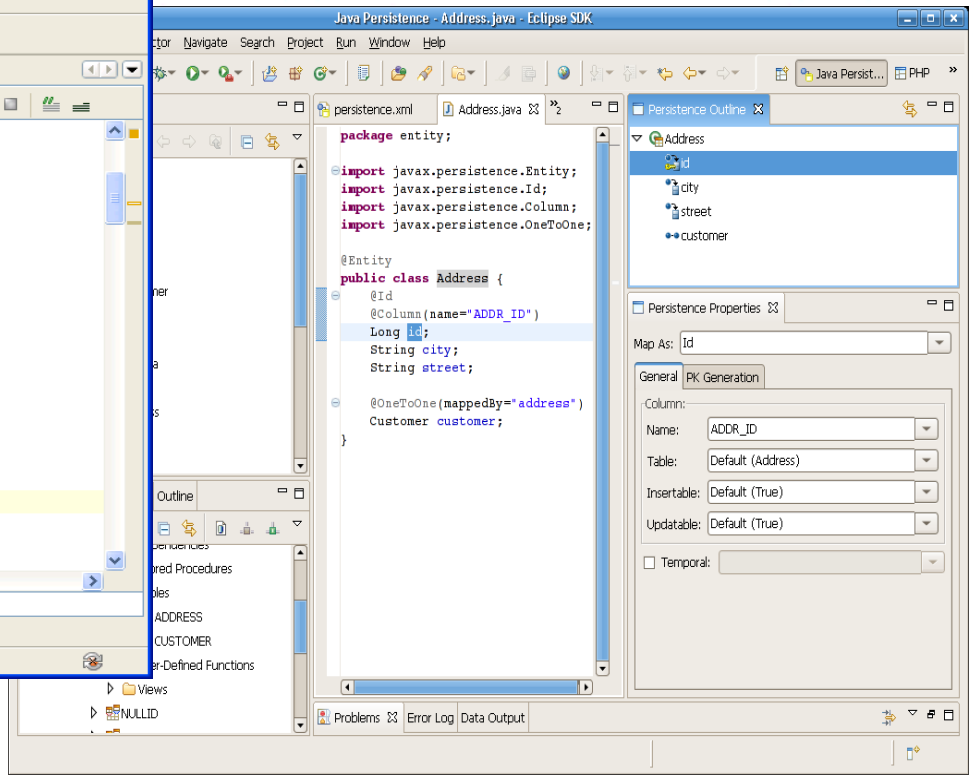
- JPA 표준 구현체 - RI(Reference Implementation)
- Oracle 이 초기 코드를 contribute 해서 시작
- Oracle, Sun, TmaxSoft가 공동으로 작업
- 다양한 제품에서 사용중
  - GlassFish
  - Java EE 5 SDK and Sun Application Server 9.0
  - Oracle AS 10.1.3.1
  - TmaxAS JEUS 6.0
  - JOnAS EJB 3.0 Container – EasyBeans
  - Spring 2.0
  - NetBeans 5.5
- https://glassfish.dev.java.net/javaee5/persistence/

Tools

- NetBeans 5.5 Enterprise Pack
- Eclipse 3.2 + Dali Plugin 0.5 Preview
- IntelliJ 6.0



NetBeans 5.5



Eclipse 3.2 + Dali

TmaxSoft

What JPA is

Entity

- DB 데이터 모델에 해당하는 도메인 모델 객체
- POJO 객체 (with no-arg constructor)
  - new operator에 의해 생성
  - 별도의 interface를 구현할 필요없음
  - field 혹은 property가 DB에 매핑됨
  - Persistent identity (Primary Key)를 가짐
  - 상속관계를 가질 수 있음
  - 다른 entity와 관계를 가질 수 있음 (1:1, 1:N, N:M)

- annotation 이나 XML로 표기
  - @Entity(name=”Customer”)
  - <entity name=”Customer” class=”a.b.Customer”>

Entity

```
// Ordinary POJO class
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;}
}
```

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

- Annotations 또는 XML (or both)로 지정
- 설정을 최소화 하기 위한 Default rule이 존재
  - table, column 이름
  - mapped attributes and types
  - 하지만 이 규칙을 알고 사용해야 함!
- 단순한 매핑의 경우 단순하게, 하지만 복잡한 경우도 지원하게 디자인 됨
- 필드 혹은 메소드(Property)가 mapping
- PK - Simple primary key, composite key
- 다양한 타입 지원
  - Basic, LOB, enum, Serializable, Embedded, Date/Time
- 다중 table mapping 지원
- Key 생성 지원 - Sequence, Identity, Table

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() {}
    ...
}
```

O/R mapping

```
@Embeddable
public class Address implements Serializable {
    protected String street;
    protected String city;
    protected String state;
    @Column(name="ZIP_CODE")
    protected Integer zip;

    public Address () {}
    ...
}
```

Relationships

- 데이타 모델의 관계를 표현 (1:1, 1:N, M:N, N:1)
- 다른 entity에 대한 참조나 참조 Collection으로 표현됨
- Collection, Set, List, Map 타입 지원
- 양방향(Bi-directional), 단방향(Uni-directional)
- Owning side가 foreign key를 가지게 됨
- Lazy fetching 지원
- cascading option – persist, remove, merge, refresh

Relationships

```
@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;
    ...
}
```

persistence.xml

- Persistence Units을 정의하는 디스크립터
- Persistence Unit
  - Entities는 persistence unit으로 묶여서 다루어짐
  - META-INF/persistence.xml 에 정의됨
  - 유일한 unit name으로 식별
  - 하나의 data-source에 대응 (one PU => one DB)
  - Persistence provider setting
  - Provider-specific configurations/properties

persistence.xml

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

EntityManager API

- Hibernate Session, JDO PersistenceManager와 유사
- Entity를 다루기 위한 단순하고도 중요한 인터페이스
- Application code는 이를 이용하여 DB로 저장하거나 로드함
- 거의 항상 Transaction과 같이 사용되어야 함

- EntityManager 종류
  - Application-managed – EntityManager instance 가 applications에 의해 직접 관리 (Java SE, Java EE)
  - Container-managed – EntityManager instance 가 container 에 의해 관리됨 (Java EE)

EntityManager API

- Entity lifecycle
  - persist() - Insert an entity into DB
  - remove() - Delete an entity from DB
  - refresh() - Reload the entity state from DB
  - merge() - Synchronize the state of detached entity
- Finder
  - find() - Get an entity with primary key
- Queries
  - createQuery() - Create a query object
- Others
  - contains() - Determine if entity is managed
  - flush() - Force synchronization to DB

EntityManager API

```
// container-managed
@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;
}
```

EntityManager API

```
//update
order.setDeliveredDate(date);
bean.updateOrder(order);
...
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);
}
```

Queries

- 다양한 조건에 맞는 데이터를 찾고자 할때 유용함
- Query 종류
  - Java Persistence QL – Object-level query
  - Native query - DB-specific SQL
  -
  - 동적(Dynamic) query – on-the-fly in runtime
  - 정적(Static) query – Named query
    - annotations 이나 XML로 정의 - @NamedQuery, @NamedNativeQuery
- Java Persistence QL (기존의 EJBQL)
  - SELECT, UPDATE, DELETE 지원
  - Subquery, Join, aggregate function 등 지원
  - 상속모델을 위한 Poloymorphic query 지원

Queries

```
//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();
```

How to use

Java SE mode

- Container 없이 라이브러리 처럼 사용
- Java SE 5가 지원되는 다음 환경에서 사용가능
  - Stand-alone application
  - Web container (e.g. Tomcat)
  - Legacy Java EE AppServer (e.g. J2EE 1.4 compliants)
- Application-managed EntityManager만 사용가능
- Transaction 관리
  - Resource-local transaction (JDBC transaction)
  - EntityTransaction API를 사용하여 tx 관리
  - JTA 지원은 벤더별 옵션

Java SE mode: persistence.xml

```
<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>
      <!-- JDBC settings should be given -->
      <property name="toplink.jdbc.driver"
     value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="toplink.jdbc.url"
     value="jdbc:derby://localhost:1527/sample"/>
      <property name="toplink.jdbc.user" value="app"/>
      <property name="toplink.jdbc.password" value="app"/>
    </properties>
```

Java SE mode

```
//Get EntityManagerFactory using bootstraping API
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("HR");
EntityManager em = emf.createEntityManager();
//Manage resource local transactions by application
EntityTransaction tx = em.getTransaction();
tx.begin();
...
em.persist(customer);
customer = em.find(Customer.class, id);
...
tx.commit();
em.close();
emf.close(); //Manage EMF and EM by application
```

Java EE mode

- Java EE 5 호환 container에서 지원
  - EJB container 혹은 Web container (Servlet, JSP, JSF)에서 사용가능
- 두 종류의 EntityManager 모두 사용가능
  - Application-managed EM – EntityManager instance 가 application에 의해 직접 관리
  - Container-managed EM – EntityManager instance 가 container에 의해 관리됨
- Persistence Unit 범위(scope)
  - EAR level – a JAR file inside EAR
  - EJB level – inside EJB-jar
  - WAR level - WEB-INF/classes, WEB-INF/lib/xxx.jar
  - AppClient level – inside AppClient-jar

Java EE mode: 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>
```

example: app-managed EM(1)

```
public class HelloServlet extends HttpServlet {
  // EntityManagerFactory is injected
  @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)

```
// EntityManagerFactory is looked up
@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;
    }
}
```

Spring's JPA support

- Spring 2.0부터 JPA abstraction layer를 제공
- EntityManagerFactory setup
- EntityManagerFactory/EntityManager injection
- JpaTemplate and JpaDaoSupport API
  - JdbcTemplate와 같은 편리한 API 제공
  - EntityManager instances를 관리
- Exception 변환
  - JPA PersistenceException -> Spring DataAccessException
- Transaction 관리
  - Spring의 declarative tx management 기능을 사용
  - JpaTransactionManager 혹은 JtaTransactionManager 설정
- Java SE, Java EE 환경에 관계없이 JPA를 사용할 수 있게 해주는 장점이 있음

Spring's JPA support

```
//applicationContext1.xml
    <!-- EntityManagerFactory for Java SE environment -->
    <bean id="entityManagerFactory1"
     class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
        <property name="persistenceUnitName" value="sample1"/>
    </bean>


    <!-- EntityManagerFactory for Java EE environment -->
    <!--<jee:jndi-lookup id="entityManagerFactory" jndi-name="jpa/myPersistenceUnit"/>-->


    <bean id="HRServiceBean1" class="springexamples.jpa.HRServiceBean1">
        <property name="entityManagerFactory" ref="entityManagerFactory1"/>
    </bean>
```

Spring's JPA support

```
//applicationContext2.xml
  <!-- container-based EntityManagerFactory -->
   <bean id="entityManagerFactory2"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
      <property name="persistenceXmlLocation" value="classpath:springex/jpa/persistence2.xml"/>
      <property name="persistenceUnitName" value="sample2"/>
      <property name="dataSource" ref="sampleDataSource"/>
      <property name="loadTimeWeaver">
      <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"/>
      </property>
   </bean>
  <!-- DataSource for Java SE environment -->
  <bean id="sampleDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="org.apache.derby.jdbc.ClientDriver"/>
    <property name="url" value="jdbc:derby://localhost/sample"/>
    ...
  </bean>
  <!-- DataSource for Java EE environment -->
  <!--<bean id="sampleDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:comp/env/jdbc/sample"/>
  </bean>-->
```

Spring's JPA support

```
...
  <!-- resource-local tx -->
  <bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory2"/>
  </bean>
  <!-- JTA tx -->
<!--<bean id="txManager" class="org.springframework.transaction.jta.JtaTransactionManager"/>-->


  <aop:config>
    <aop:pointcut id="HRServiceBeanMethods" expression="execution(*
    springexamples.jpa.HRServiceBean5.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="HRServiceBeanMethods"/>
  </aop:config>


  <tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
      <tx:method name="*" propagation="REQUIRED"/>
      <tx:method name="close" propagation="NOT_SUPPORTED"/>
    </tx:attributes>
  </tx:advice>
```

Spring's JPA support

```
//HRServiceBean.java
    JpaTemplate jpaTemplate;


    public void setEntityManagerFactory(EntityManagerFactory emf) {
        this.emf = emf;
        jpaTemplate = new JpaTemplate(emf);
    }
    public Department createDepartment(final String id, final String name) {
        return (Department) jpaTemplate.execute(new JpaCallback() {
            public Object doInJpa(EntityManager em) throws PersistenceException {
                dept = new Department(id, name);
                em.persist(dept);
                return dept;
            }
        });
    }
    public Department getDepartment(String id) {
        return jpaTemplate.find(Department.class, id);
    }
```

Advanced Topics

O/R mapping

- Logical and Physical mapping
  - Logical mapping – logical structure
  - Physical mapping – Database structure. prefered to be specified in XML to be portable across multiple DB
- Two Access types
  - Field : if mappings are specified on field
  - Property : if mappings are specified on getter method

O/R mapping

- Logical annotations
  - @Id – Simple primary key
  - @EmbeddedId, @IdClass – Composite primary keys
  - @Basic – basic type
  - @Lob – CLOB/BLOB
  - @Temporal – Date/Time
  - @Enumerated – enum type
  - @Transient – no mapping
  - @Version – optimistic locking version field
  - @Embedded, @Embeddable – embedded object
  - @GeneratedValue – primary key generation
  - @OneToOne, @OneToMany, @ManyToOne,@ManyToMany – entity associations

O/R mapping

- Physical annotations
  - @Table – table name
  - @SecondaryTable – multiple table
  - @Column – column name
  - @JoinColumn – foreign key column
  - @JoinTable – relation table (for 1:n or n:m)
  - @SequenceGenerator, @TableGenerator
  - @NamedNativeQuery – pre-defined SQL

Eager vs. Lazy fetching

- Eager fetching - associated entities are loaded together
  - @OneToMany(fetch=FetchType.EAGER)
- Lazy fetching - associated entities are loaded on demand
  - @OneToOne(fetch=FetchType.LAZY)
  - Lazy fetching is default for @OneToMany, @ManyToMany relationship - Collections type
- Simple types can have @Basic(fetch=FetchType.LAZY) but not recommended unless type is large object
- Lazy fetching is a hint to persistence engine, so it may not be supported

Inheritance

- OO concept is mapped to the relational Database
- Can extend the following classes
  - Another (abstract/concrete) entity class
  - Ordinary non-entity class
    - superclass fields are not persistent
  - Mapped superclass @MappedSuperclass
    - Similar to ordinary class but their fields are mapped also

Inheritance

```
@Entity
@Inheritance(strategy=SINGLE_TABLE)
@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;
    ...
```

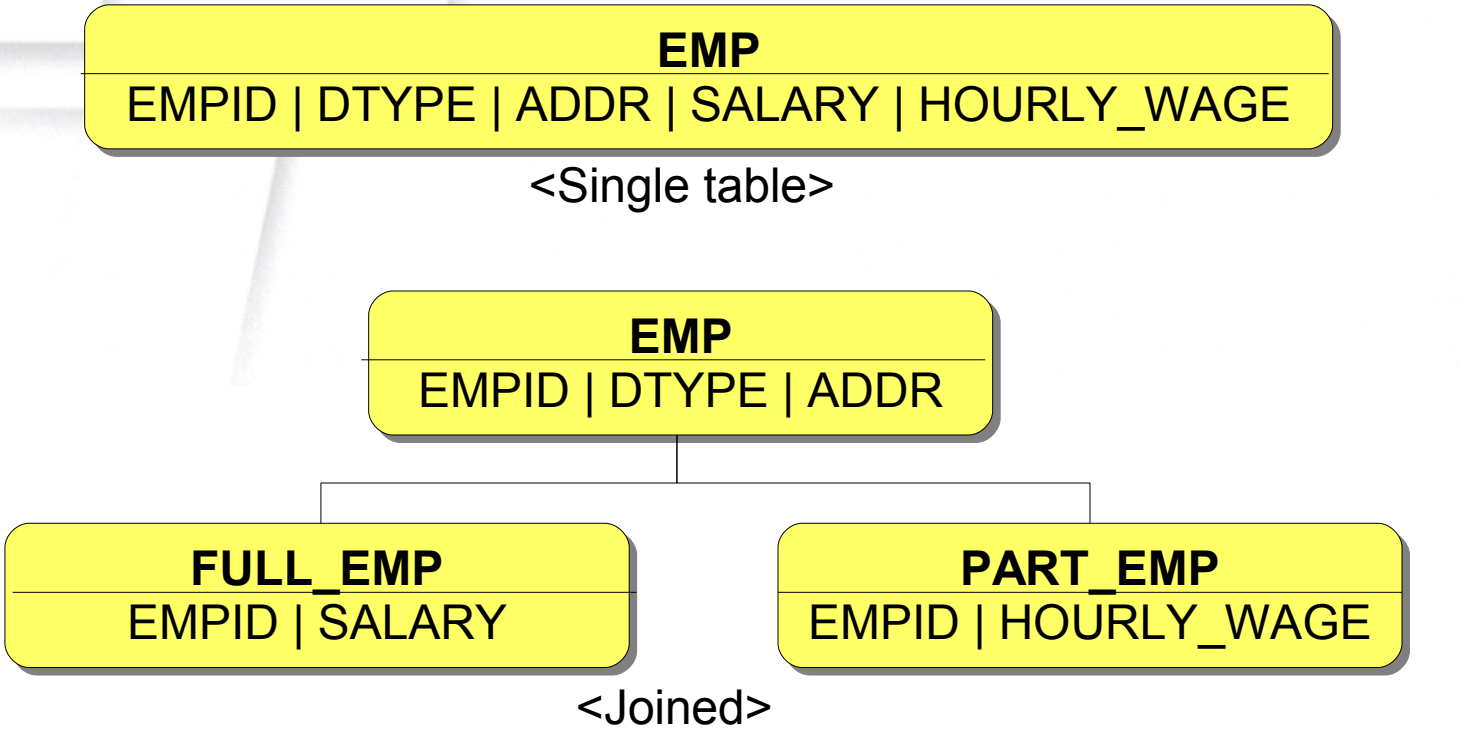
Inheritance: strategies

- @Inheritance(strategy=SINGLE_TABLE)
  - All entities in inheritance hierachy stored in one single table
  - Discriminator column is mandatory
- @Inheritance(strategy=JOINED)
  - Each entity class in inheritance hierachy stored in a separate table
  - Discriminator column is needed normally
- @Inheritance(strategy=TABLE_PER_CLASS)
  - Each concrete class stored in a separate table
  - Optional – not supported in TopLink Essentials

Inheritance: 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;
...
```

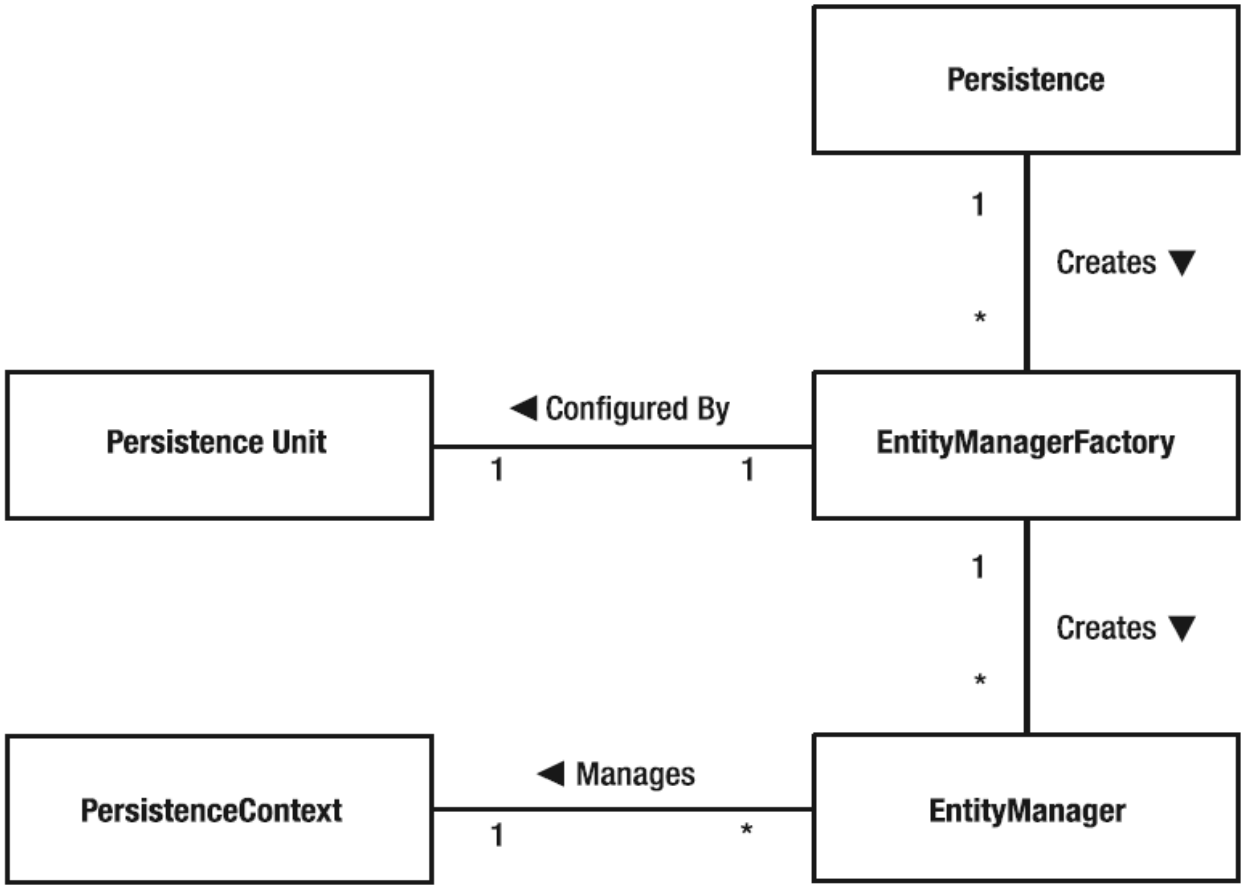
Inheritance: Data models



JPA API

- Persistence
  - Boostrapping API to get EntityManagerFactory in SE mode
- EntityManagerFactory
  - Thread-safe factory to create EntityManagers
  - One EMF is normally used for one Persistence Unit
  - Don’t need to handle this in container-managed mode
- EntityManager
  - Manages the Persistence Context
  - Not thread-safe, so should not be shared across threads
- Query
  - Created by EntityManager
  - Internally it’s connected to EntityManager

JPA API relationship



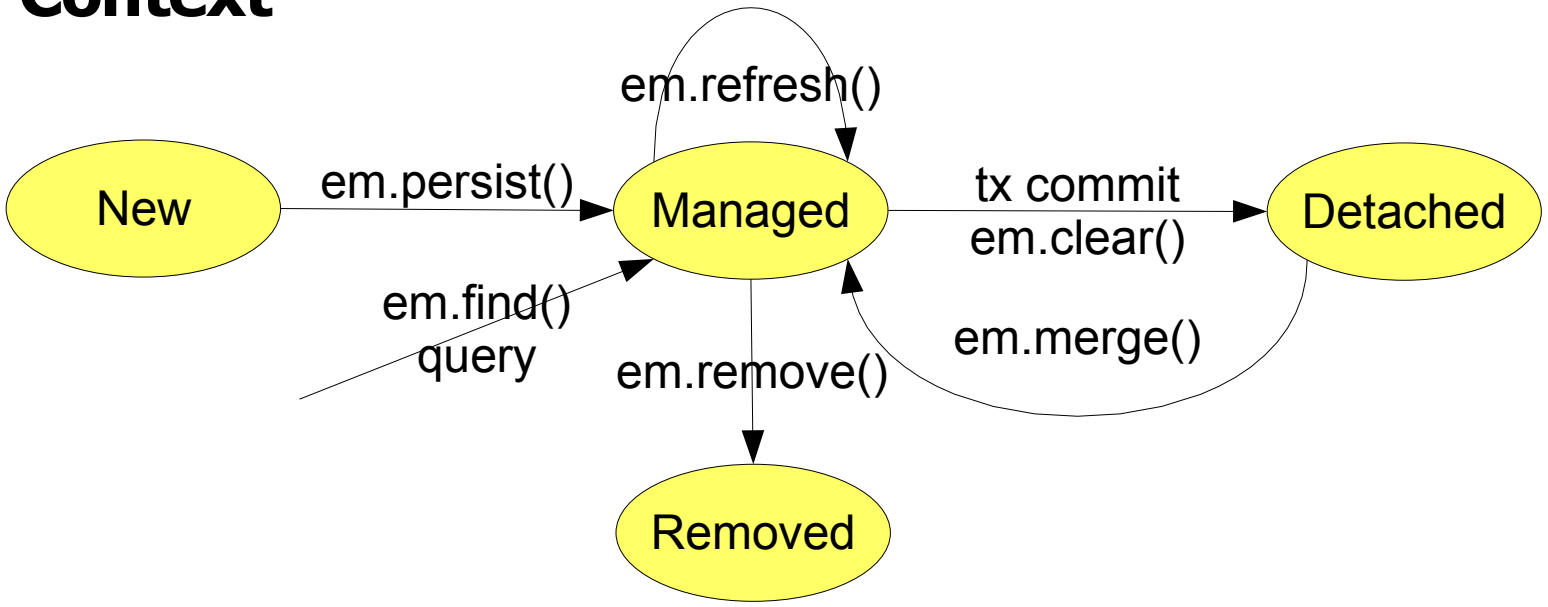
Source: Pro EJB 3 JPA, Mike Keith, Apress

App vs. Container-managed EM

- **Application-managed EntityManager**
  - Inject or lookup EntityManagerFactory
  - The lifecycle of EntityManager – create, close
  - Transaction – JTA or Resource-local
  - Persistence Context – like EntityManager.clear()
- **Container-managed EntityManager**
  - Inject or lookup 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
  - Support persistence context propagation

Entity Lifecycle

- An Entity instance becomes **managed** if entity manager persisted or loaded it
- Managed entities will be synchronized to database
- The set of managed entity instances is **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

Entity Lifecycle callbacks

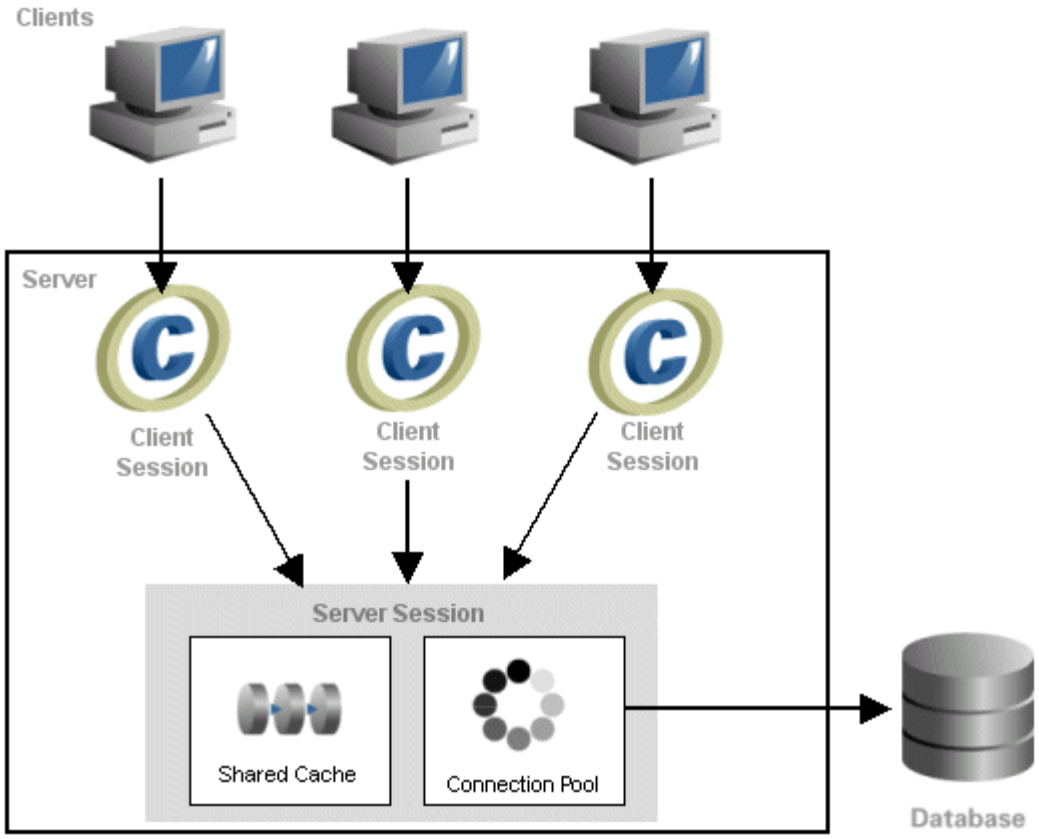
- Callbacks
  - @PrePersist: upon invocation of persist operation
  - @PostPersist: after database insert
  - @PreRemove: upon invocation of remove operation
  - @PostRemove: after database delete
  - @PreUpdate: before database update
  - @PostUpdate: after database update
  - @PostLoad: after loading the instance
- Define callback listener in
  - Inside entity class itself
  - Separate listener class (reusable)

Persistence Context

- 1st-level caching and working set
  - In the same PC, one entity instance is managed for one PK
  - All managed instances in PC are synchronized to DB
- Two types of PC
  - Extended scope
    - PC survives across tx
    - Should call clear() explicitly and manage memory carefully
    - Default for application-managed EntityManager
  - Transaction scope (Java EE only)
    - PC is cleared after tx commit/rollback
    - Default for container-managed EntityManager
- Container-managed EntityManager support PC propagation

2nd-level caching

- Persistence providers support 2nd-level caching to improve performance
- Shared cache for several persistence contexts



Lazy fetching and detachement

- Entities can be detached
  - if serialized to clients
  - if tx finishes
- Lazy fetched attributes should not be used after detachement

Lazy fetching and detachement

```
@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!
}
```

Query API

- Query object is created by EntityManager
  - createQuery(), createNativeQuery()
  - createNamedQuery()
- Named parameters and positional parameters
  - setParameter()
- Paging
  - setMaxResults(), setFirstResult()
- Flush mode
  - AUTO – PC is flushed automatically before executing query
  - COMMIT – PC is not flushed when executing query
- Provider-specific hints
  - e.g.) toplink.pessimistic-locking, toplink.refresh
- Execution
  - getResultList(), getSingleResult(), executeUpdate()

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
- Polymorphic queries support for Inheritance model

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
```

Summary

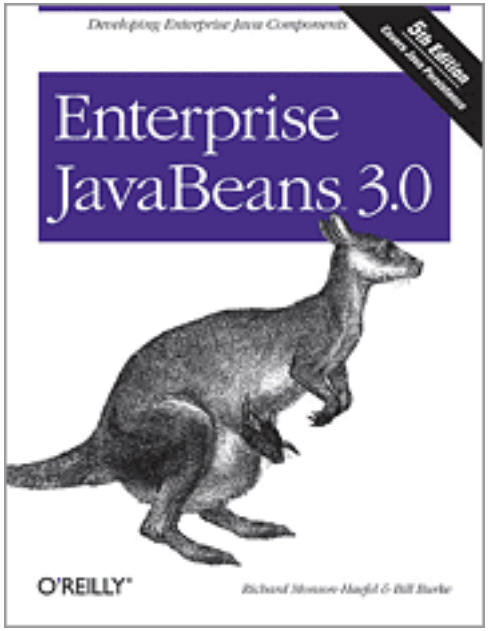
- JPA는 표준 ORM 솔루션
- 객체 지향적인 관점에서 DB를 다루며 상속관계와 같은 풍부한 객체 모델 지원
- DB의 다양한 기능 지원 및 풍부한 매핑 기능
- 강력한 Query 기능을 제공
- 다양한 환경(DB, Platform) 에서 사용가능한 솔루션
- 단순한 기능은 단순하게 구현가능할 뿐만 아니라 복잡한 상황도 잘 지원
- 성능 향상을 위한 레이어 제공 – SQL 최적화, caching
- API는 단순하나 개념을 제대로 이해하고 사용해야 함
- 각 persistence provider 는 최적화된 성능을 제공하기 위해 경쟁할 것임, 선택은 사용자의 몫

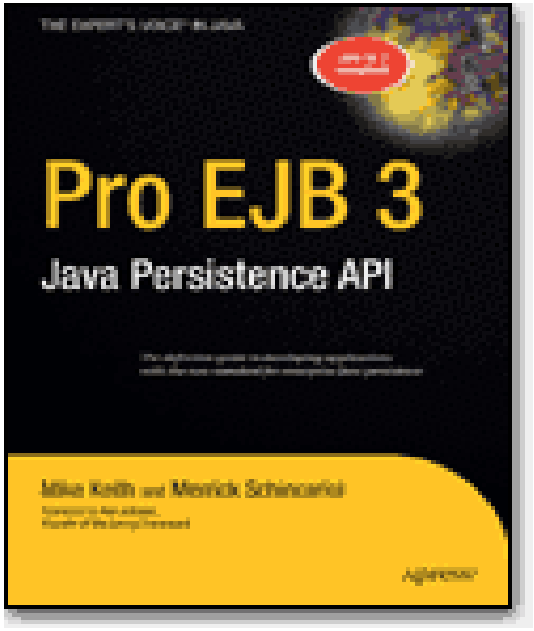
References

- JSR 220 - http://www.jcp.org/en/jsr/detail?id=220
  - Java Persistence API 1.0
- 2006 JavaOne - http://developers.sun.com/learning/javaoneonline
  - TS-3395 Java Persistence API
  - TS-9056 Java Persistence API In 60 Minutes
  - TS-1887 The Java Persistence API in the Web Tier
- 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/
- 마이크로소프트웨어 2006년 6월 특집
- http://static.springframework.org/spring/docs/2.0.x/reference/index.html
- My Blog - http://weblogs.java.net/blog/guruwons/

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 @ tmax.co.kr