Using the Java Persistence API

Mike Keith
Oracle Corp.

Patrick Linskey
BEA Systems

Background

- Part of JSR-220 (EJB 3.0)
- Began as simplification of entity beans
  - Evolved into POJO persistence technology
- Scope expanded at request of community to support general use in Java™ EE and Java SE environments
- Implementations
  - Oracle TopLink Essentials (RI)
  - BEA Kodo / Apache OpenJPA
  - RedHat Hibernate

Primary Features

- POJO-based persistence model
  - Simple Java classes—not components
- Support for enriched domain modeling
  - Inheritance, polymorphism, etc.
- Expanded query language
- Standardized object/relational mapping
  - Using annotations and/or XML
- Usable in Java EE and Java SE environments
- Support for pluggable persistence providers

Entities

- Plain old Java objects
  - Created by means of **`new`**
  - No required interfaces
  - Have persistent identity
  - May have both persistent and non-persistent state
    - Simple types (e.g., primitives, wrappers, enums, serializable)
    - Composite dependent object types (e.g., Address)
    - Non-persistent state (transient or @Transient)
  - Can extend other entity and non-entity classes
  - No need for data transfer objects

Entity Example

```
@Entity
public class Customer implements Serializable {
    @Id @GeneratedValue protected Long id;
    protected String name;
    @Embedded protected Address address;
    protected PreferredStatus status;
    @Transient protected int orderCount;

    public Customer() {}

    public Long getId() {return id;}

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

    ...
}
```

Session Bean using an Entity

```
import javax.persistence.*;
import javax.ejb.*;

@Stateless
@Remote(OrderManager.class)
public OrderManagerImpl implements OrderManager {

    @PersistenceContext private EntityManager em;

    public Order newOrderForProduct(long custId,
        long prodId) {
        Customer c = em.find(Customer.class, custId);
        Product p = em.find(Product.class, prodId);

        Order o = new Order(customer);
        em.persist(o);
        o.addLineItem(new Item(p));

        return o;
    }
}
```

Entity Identity

- Every entity has a persistence identity
  - Maps to primary key in database
- Can correspond to simple type
  - Annotations
    - @Id—single field/property in entity class
    - @GeneratedValue—value can be generated automatically
- Can correspond to user-defined class
  - Annotations
    - @EmbeddedId—single field/property in entity class
    - @IdClass—corresponds to multiple Id fields in entity class
- Must be defined on root of entity hierarchy or mapped superclass

Persistence Context

- Set of managed entity instances at runtime
- Unique entity identity for any persistent identity
- Entity instances all belong to same persistence unit; all mapped to same database
  - Persistence unit is unit of packaging and deployment
- EntityManager API
  - manages persistence context
  - controls lifecycle of entities
  - finds entities by id
  - factory for queries

Persistence Context

- Entity becomes managed
- Entity becomes persistent in db at commit time

```
@Stateless public class OrderManagementBean
    implements OrderManagement {
    …
    @PersistenceContext EntityManager em;
    …
    public Order addNewOrder(Customer customer,
                             Product product) {

        Order order = new Order(product);
        customer.addOrder(order);
        em.persist(order);
        return order;
    }
}
```

Relationships

```
@Entity public class Customer {
    @Id protected Long id;
    …
    @OneToMany protected Set<Order> orders = new HashSet();
    @ManyToOne protected SalesRep rep;
    …
    public Set<Order> getOrders() {return orders;}
    public SalesRep getSalesRep() {return rep;}
    public void setSalesRep(SalesRep rep) {this.rep = rep;}
}

@Entity public class SalesRep {
    @Id protected Long id;
    …
    @OneToMany(mappedBy=“rep”)
    protected Set<Customer> customers = new HashSet();
    …
    public Set<Customer> getCustomers() {return customers;}
    public void addCustomer(Customer customer) {
        getCustomers().add(customer);
        customer.setSalesRep(this);}
}
```

Cascading persist

```
@Entity
public class Customer {
    @Id protected Long id;
    ...
    @OneToMany(cascade=PERSIST)
    protected Set<Order> orders = new HashSet();
}

...
public Order addNewOrder(Customer customer, Product
product) {

    Order order = new Order(product);
    customer.addOrder(order);
    return order;
}
```

Queries

- Static queries
  - Defined in Java annotations or XML
- Dynamic queries
- Use JPQL or SQL
- Named or positional parameters
- EntityManager is factory for Query objects
  - createNamedQuery, createQuery, createNativeQuery
- Query methods for controlling max results, pagination, flush mode

Dynamic Query

```
@PersistenceContext EntityManager em;

...
public List findByZipcode(int zip) {
    return em.createQuery ( “SELECT p FROM Person p “
        + “WHERE p.address.zip = :zipcode”)
    .setParameter(“zipcode”, zip)
    .setMaxResults(20)
    .getResultList();
}
```

Named Query

```
@NamedQuery(name="Person.findByZipcode", query =
"SELECT p FROM Person p WHERE p.address.zipcode = :zip")
@Entity public class Person { ... }


public List findPersonByZipcode(int zipcode) {
    return em.createNamedQuery ("Person.findByZipcode")
    .setParameter("zip", zipcode)
    .setMaxResults(20)
    .getResultList();
}
```

JP QL

- An extension of EJB QL
  - Like EJB QL, a SQL-like language
- Added functionality
  - Projection list (SELECT clause)
  - Explicit JOINS
  - Subqueries
  - GROUP BY, HAVING
  - EXISTS, ALL, SOME/ANY
  - UPDATE, DELETE operations
  - Additional functions

JP QL Examples

```
SELECT e FROM Employee e WHERE e.status = :stat

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’

SELECT DISTINCT o
   FROM Invoice i JOIN o.lineItems l JOIN l.product p
   WHERE p.productType = ‘shoes’

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

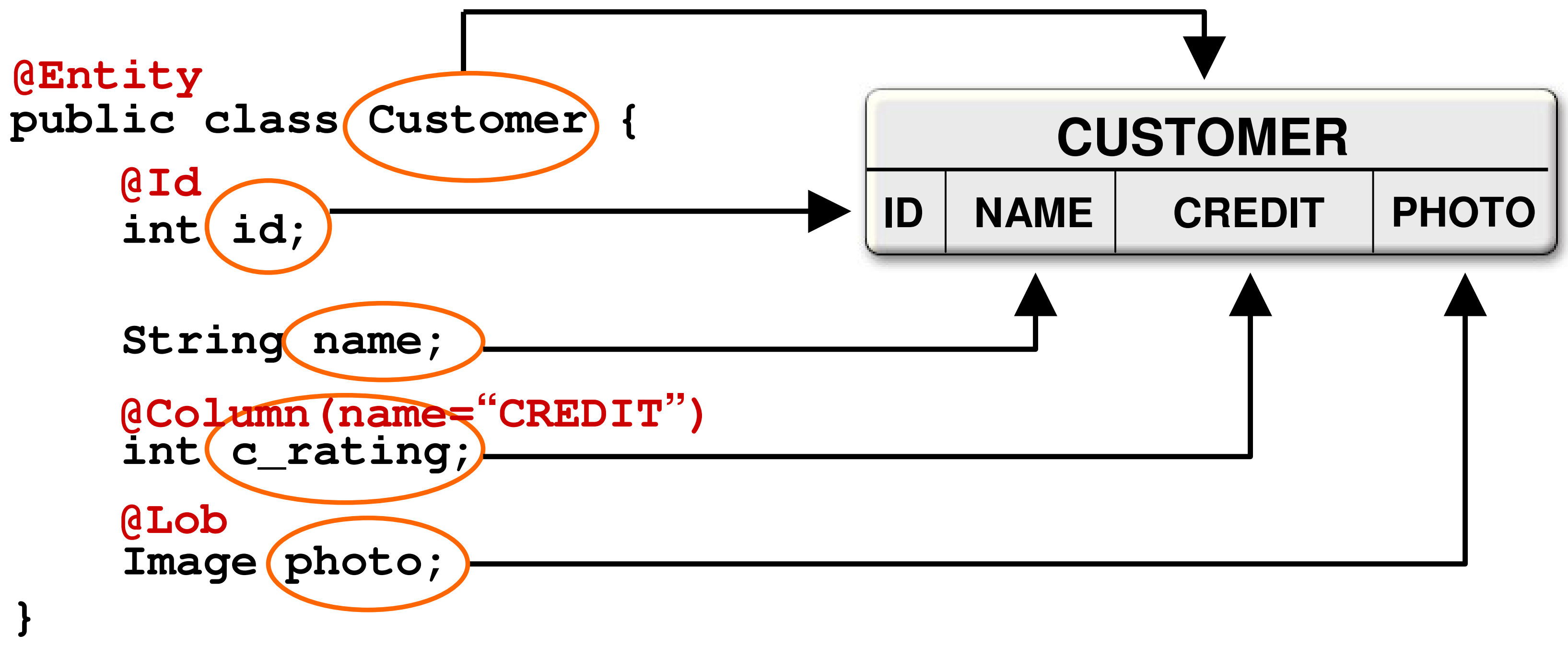
O/R Mapping

- Map persistent object state to relational database
- Map relationships to other entities
- Mapping metadata may be annotations or XML (or both)
- Annotations
  - Logical—object model (e.g., @OneToMany, @Id, @Transient)
  - Physical—DB tables and columns (e.g., @Table, @Column)
- XML
  - Elements for mapping entities and their fields or properties
  - Can specify metadata for different scopes
- Rules for defaulting of database table and column names

O/R Mapping

- State or relationships may be loaded or "fetched" as EAGER or LAZY
  - LAZY is a hint to the Container to defer loading until the field or property is accessed
  - EAGER requires that the field or relationship be loaded when the referencing entity is loaded
- Cascading of entity operations to related entities
  - Setting may be defined per relationship
  - Configurable globally in mapping file for persistence-by-reachability

Simple Mappings



```
@Entity
public class Customer {
    @Id
    int id;

    String name;
    @Column(name="CREDIT")
    int c_rating;
    @Lob
    Image photo;
}
```

Simple Mappings

```
<entity class=“com.acme.Customer”>
  <attributes>
    <id name=“id”/>
    <basic name=“c_rating”>
      <column name=“CREDIT”/>
    </basic>
    <basic name=“photo”><lob/></basic>
  </attributes>
</entity>
```

Persistence Unit

- Set of entities and related classes that share the same configuration
- Convenient packaging and deployment unit
- Runtime configuration defined in persistence.xml
- Can reference additional classes on classpath or additional jar
- One or more O/R mapping files
- Scoping boundary for queries and id generators

Example

```
<persistence>
  <persistence-unit name="OrderMgmt">
    <provider>com.acme.PersistenceProvider</provider>
    <jta-data-source>jdbc/MyOrderDB</jta-data-source>
    <mapping-file>order-mappings.xml</mapping-file>
    <jar-file>myparts.jar</jar-file>
    <properties>
      <property
        name="com.acme.persistence.logSQL"
        value="ALL"/>
    </properties>
  </persistence-unit>
</persistence>
```

Configuration

- One XML file: META-INF/persistence.xml

```
<persistence>
  <persistence-unit name="OrderManagement">
    <jta-data-source>jdbc/MyOrderDB</jta-data-source>
    <mapping-file>ormap.xml</mapping-file>
  </persistence-unit>
</persistence>
```

- May define multiple units in same XML file
- Persistence provider is automatically located if not specified in XML
  - Container’s choice in container
  - Classpath order outside a container
- Entity types are auto-detected by the container

Transactions

- EntityManagers are configured to be of a particular transaction type
  - Global JTA transactions – the most common
  - Private or ‘resource-local’ JDBC-style transactions
- JTA transactions
  - Used by either container-managed or application-managed EntityManagers
  - Demarcated externally to the EM (either by Container or application)
- Resource-local transactions
  - Only in application-managed EntityManagers
  - Demarcated by invoking on the EM

ORACLE

bea Think liquid.

Entity Transactions

- Resource-level transaction akin to a JDBC transaction
- Isolated from transactions in other EntityManagers
- Transaction demarcation under explicit application control using EntityTransaction API
  - begin(), commit(), setRollbackOnly(), rollback(), isActive()
- Underlying (JDBC) resources allocated by EntityManager as required

Java SE Example

```
EntityManagerFactory emf =
   Persistence.createEntityManagerFactory("orders");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();

try {
    Collection<Customer> customers = loadCustomersFromFile
         (new File("nightly-upload.csv"));
    for (Customer customer : customers)
        em.persist(customer);
    em.getTransaction().commit();
} finally {
    if (em.getTransaction().isActive())
        em.getTransaction().rollback();
}
em.close();
emf.close();
```

Detached Entities

- Instances become detached when
  - the persistence context ends
  - upon serialization
- Detached entities can be accessed and modified either in the current VM or in another VM
- Changes to detached instances can be merged into the original persistence context or a different one

```
void updatePerson(Person personDTO) {
    Person p = em.merge(personDTO);
    p.setLastUpdated(new Date());
}
```

Detached Entities

- Detached instances are useful for transfer to a different physical tier
  - Must implement java.io.Serializable
- Represent a conversion from the persistent domain to the data transfer domain
- May only access loaded state

Optimistic Locking

```
@Entity
public class Employee {
  @Id @GeneratedValue private long pk;
  @Version private int oplock;
  private String name;
}
```

- JPA currently does not standardize pessimistic locking
- Version field is maintained by the persistence provider
- “Offline Optimistic Lock” pattern is automatically handled by JPA detachment
- Bulk updates require manual lock field increment (or vendor-specific feature)

Releases

- Final Release part of EJB 3.0, which is part of Java EE 5.0
- JPA specification available at http://jcp.org/en/jsr/detail?id=220
- Popular Implementations
  - Oracle TopLink Essentials (RI)
  - BEA Kodo / Apache OpenJPA
  - RedHat Hibernate

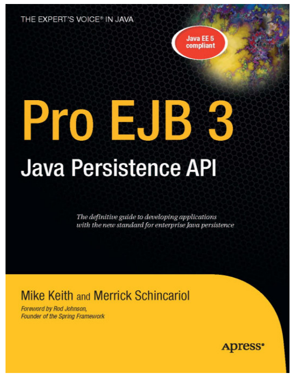
Summary

- Entities are simple Java classes
  - Easy to develop and intuitive to use
  - Can be moved to other server and client tiers
- EntityManager
  - Simple API for operating on entities
  - Supports use inside and outside Java EE containers
- Standardization
  - O/R mapping using annotations or XML
  - Named and dynamic query definition
  - SPI for pluggable persistence providers

For More Information

- Resources
  - http://dev2dev.bea.com/persistence
  - http://incubator.apache.org/projects/openjpa
  - http://otn.oracle.com/jpa
  - michael.keith@oracle.com
  - patrick.linskey@bea.com
- Books
  - Pro EJB 3: Java Persistence API

    Mike Keith & Merrick Schincariol
    (Foreword by Rod Johnson)


THE EXPERT'S VOICE® IN JAVA

Java EE 5 compliant

Pro EJB 3

Java Persistence API

The definitive guide to developing applications with the new standard for enterprise Java persistence

Mike Keith and Merrick Schincariol

Foreword by Rod Johnson, Founder of the Spring Framework

Apress®