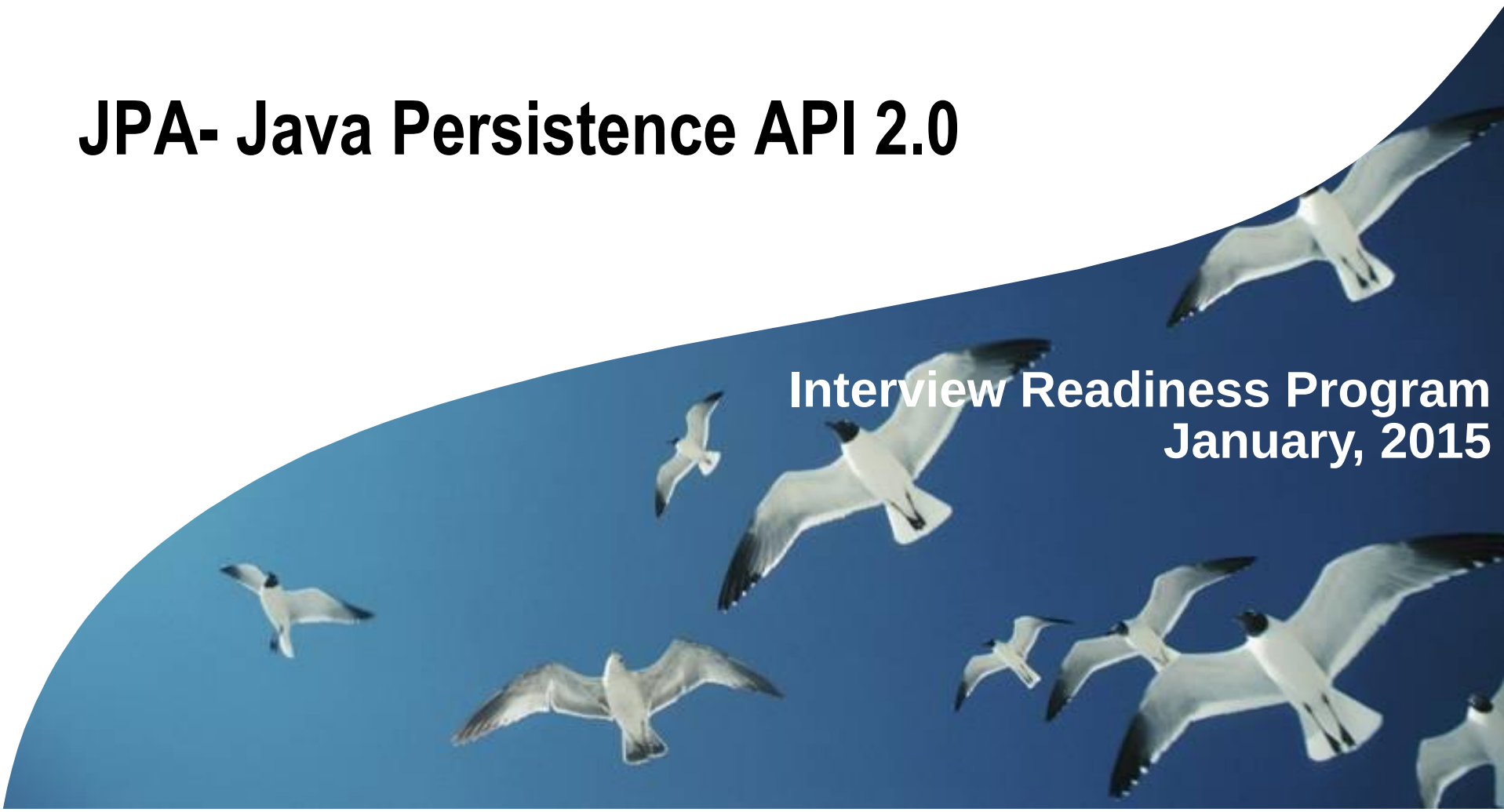
JPA- Java Persistence API 2.0



Interview Readiness Program
January, 2015

People matter, results count.

Objectives of JPA

- **Purpose**:
  - This presentation will introduce Java Persistence API in general and some of the JPA 2.0 features
- **Product**:
  - To understand JPA standard and JPA components
  - Comparison of JDBC and JPA
  - A brief understanding of various ORM technology
  - Understanding of JPA implementation and its benefit
- **Process**:
  - Theory Sessions
  - A recap at the end of the session in the form of Quiz

Table of contents

Introduction of JPA

- Standard persistence technology
- Object/Relational Mapping
- Developed as part of Java Specification Request (JSR) 220
- JPA 1.0 introduced in Java EE 5
- JPA 2.0 is part of Java EE 6 standards
- Designed for highly distributed applications
- Especially web-enabled applications
- Life cycle manageable by application server to increase quality of service
- Simplifies development
- Provides a framework and tools around providing automatic persistence
- Objects are mapped to tables using metadata
- Metadata is used to transform data from one representation to another

Introduction ….contn…

- JPA specifications made very significant simplifications:

- Standardizes O/R mapping metadata (not the case for EJB 2)
- Java SE 5.0 annotations can be used instead of XML deployment descriptor
- No deploy code implementing abstract classes— Entities are POJOs
- Application server is not required

- The Java Persistence API is available outside of containers

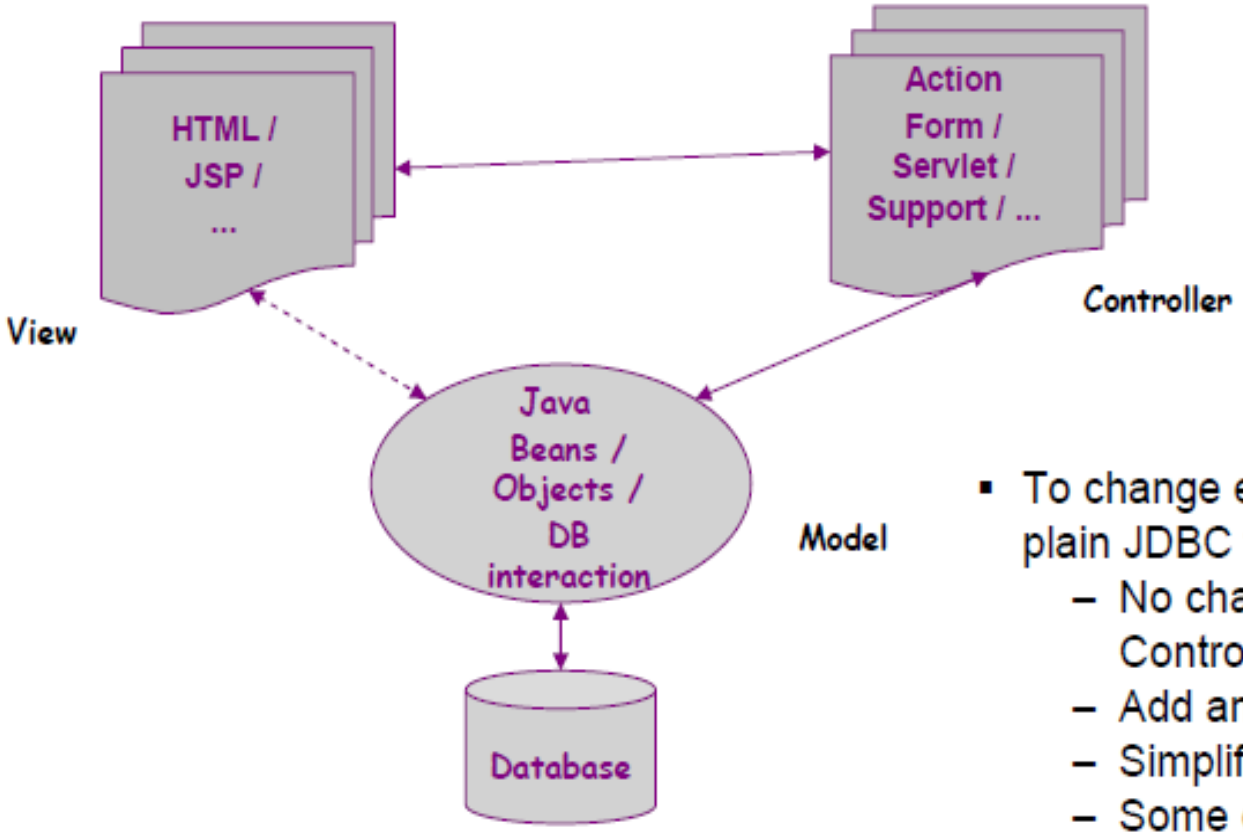
- Unit testing greatly simplified
- Removal of checked exceptions (for instance, remote exceptions)
- No bean (or business) interface required

Java Persistence Frameworks

- JDBC with DAO Pattern
- Hibernate/JPA
- OpenJPA from Apache
- TopLink from Oracle
- iBATIS SQL Mapper

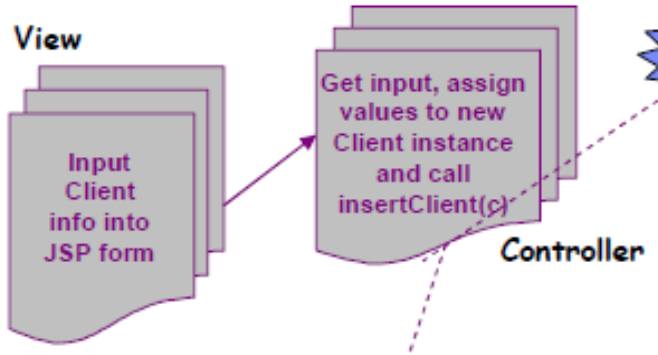
JDBC Vs. JPA

Web application example – MVC framework



- To change existing applications from plain JDBC to use JPA
  - No change required to View and Controller
  - Add annotations to Java Objects
  - Simplify DB interaction code
  - Some configuration changes (for example, persistence.xml)

JPA compared to Plain JDBC – Insert



JPA

```
@PersistenceContext(unitName="test")
EntityManager em;

public void insertClient(Client client){
  doInsert(Client)
}//end insertClient
public void doInsert(Object newObject){
   em.persist(newObject);
}//end doInsert
```

JDBC

```
public void insertClient(Client client){
    String insert ="INSERT INTO clientInfo ("name, industry,"+
      "headquarterLocation, cepClient, betaClient,"+
      "productsInUse, wasVersions) VALUES('" +
      client.getName()+"', '"+client.getIndustry()+"', '"+
      client.getHeadquarterLocation()+"', '"+
      client.getCepClient()+"','"+ client.getBetaClient()+"', '"+
      client.getProductsInUse()+"', '"+client.getWasVersions()+"');" ;
  doInsert(insert);
}//end insertClient

public void doInsert(String insert){
 Statement stmt;
 Connection conn;
 CONSTANTS c = new CONSTANTS();
 try {
  java.lang.Class.forName(c.DRIVER).newInstance();
 }catch (Exception E) {
     System.err.println("Unable to load driver.");
     E.printStackTrace();
 }
 try{
   conn = DriverManager.getConnection(c.DB_CONNECTION);
   stmt = conn.createStatement();
   stmt.executeUpdate(insert);
   stmt.close();
   conn.close();
  }catch (SQLException E) {
     System.out.println("SQLException: " + E.getMessage());
  }
}//end doInsert
```

Main JPA Components

- Entity
- Entity Relationships
- Persistence Unit
- Persistence Context
- EntityManager
- Fetch Plan
- Proxy

Hibernate vs. JPA Components

JPA	Hibernate
Entity Classes	Persistent Classes
EntityManagerFactory	SessionFactory
EntityManager	Session
Persistence	Configuration
EntityTransaction	Transaction
Query	Query
Persistence Unit	Hibernate Config

Persistence Unit

- Defines all entity classes that are managed by JPA
- Defined in META-INF/persistence.xml
- An application can have multiple persistence units

persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
 http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
 version="1.0“>

 <persistence-unit name="BankingApp">

 <provider> org.hibernate.ejb.HibernatePersistence </provider>
 <mapping-file>orm.xml</mapping-file>
 <class>hibernate.vo.Account</class>
 ...
```


persistence.xml ….contn…

```
....
<properties>
        <!-- VENDOR SPECIFIC TAGS -->
        <property name="hibernate.connection.driver_class"
            value="oracle.jdbc.driver.OracleDriver"/>
        <property name="hibernate.connection.url"
            value="jdbc:oracle:thin:@localhost:1521:XE"/>
        <property name="hibernate.connection.username"
            value="lecture10"/>
        <property name="hibernate.connection.password"
            value="lecture10"/>
        <property name="hibernate.dialect"
            value="org.hibernate.dialect.Oracle10gDialect"/>
```

persistence.xml ….contn…

```
        <property name="hibernate.show_sql"
                  value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```


Entity

- Lightweight persistent domain object – the thing you persist
- Managed objects mapped in one of two ways
  - Described in the orm.xml mapping file
  - Marked with annotations in individual classes
- Identified as managed with @**Entity**
- Primary key identified through the @**Id**

orm.xml Mapping File

```
<entity-mappings
…….
<persistence-unit-metadata>
!-- identifies the orm.xml as the only source for class definition, telling the
   engine to ignore annotations in classes -->
<xml-mapping-metadata-complete/>
<persistence-unit-defaults>
<cascade-persist/>
</persistence-unit-defaults>
</persistence-unit-metadata>
...
```

orm.xml Mapping File ….contn…

- ...

```
<package>hibernate.vo</package>
<entity class="Account" access="FIELD">
<table name="ACCOUNT" />
<attributes>
<id name="accountId">
<column name="ACCOUNT_ID" />
<generated-value strategy="AUTO" />
</id>
<basic name="balance" optional="false">
<column name="BALANCE" />
</basic>
```


orm.xml Mapping File ….contn…

```
<version name="version">
<column name="VERSION" /> Notice no type definitions!
</version>
<attributes>
</entity>
</entity-mappings>
```


Annotations: Account Entity

- @Entity

```
public class Account {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name="ACCOUNT_ID")
private long accountId;
...
public long getAccountId() {...}
public void setAccountId(long newId) {...}
...
}
```

EntityManagerFactory

- Used to create EntityManager in JavaSE environment
- Similar to Hibernate SessionFactory
- Created through a static method on Persistence

EntityManagerFactory emf =Persistence

createEntityManagerFactory("BankingApp");

Entity Manager

- Creates and removes persistent entity instances
- Finds entities by their primary key
- Allows for data querying
- Interacts with the persistence context
- Similar to Hibernate Session

- Core methods a JPA application will use
  - persist
  - remove
  - find
  - createQuery
  - close

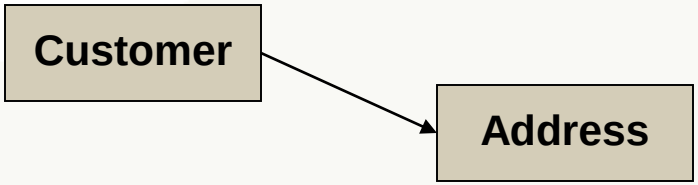
Persistence Context

- Set of unique entity instances that an application works with at any one time
- “local working copy” of persistent objects

```
public class AccountSessionBean {
        @PersistenceContext
        EntityManager em;
        public Account getAccount(int  accountId){
                Account account =em.find(Account.class, accountId);
                return account;
        }
}
```

Fetch Plan

- Determines when an associated entity should be loaded
  - EAGER: when the owning entity object is loaded
  - LAZY: when your code accesses the associated object or collection



Proxy

- A placeholder object generated by Hibernate at runtime
- Used by hibernate to implement LAZY loading of anyToOne/OneToOne related Entities
- Proxy is a subclass of the mapped class
- a placeholder for the actual object that contains the data

Entity Relationships

- Defined by relationships in the database schema
  - ManyToOne
  - OneToOne
  - OneToMany
  - ManyToMany

Bidirectional Association Example

```
@Entity
public class Account {
@OneToMany(mappedBy="account")
private Set ebills;
...
}

@Entity
public class EBill {
@ManyToOne
@JoinColumn(name="ACCOUNT_ID")
private Account account
...
}
```

@Inheritance

- Three strategies
  - Single Table Per Class Hierarchy
    - SINGLE_TABLE
  - Joined Subclass
    - JOINED
  - Table Per Concrete Class
    - TABLE_PER_CLASS
    - Support for this strategy is optional, and may not be supported by all Java Persistence API providers

SINGLE_TABLE with Annotations

```
@Entity
@Table(name = "ACCOUNT")
@Inheritance(strategy =
InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "ACCOUNT_TYPE",
discriminatorType = DiscriminatorType.STRING)
public class Account {
@Id
long accountId;
...
}
@Entity
@DiscriminatorValue("CHECKING")
public class CheckingAccount extends Account {
String checkStyle;
...
}
```

“join” strategy

```
@Entity
@Inheritance(strategy=JOINED)
@Table(name = "ACCOUNT")
@DiscriminatorColumn(name = "ACCOUNT_TYPE",
discriminatorType = DiscriminatorType.STRING, length = 10)
public class Account {
@Id
long accountId;
...
}
@Entity
@Table(name = "CHECKING_ACCOUNT")
@DiscriminatorValue("CHECKING")
public class CheckingAccount extends Account {
String checkStyle;
...
}
```

JPA Query Language

- HQL is an object-oriented, SQL Dialect agnostic query language
- JPQL is the query language standardized in the JPA specification
- JPQL is a subset of HQL

- Provides the @NamedQuery and @NamedNativeQuery annotations

JPA OrderColumn Example:

- Specifies a column that is used to maintain the persistent order of a list. The persistence provider is responsible for maintaining the order upon retrieval and in the database. The persistence provider is responsible for updating the ordering upon flushing to the database to reflect any insertion, deletion, or reordering affecting the list.

Named Query Annotations

```
import javax.persistence.*;

@NamedQueries( {
  @NamedQuery(
        name = "getAllAccounts"
        query = "from Account")

  @NamedQuery(
        name = "getAccountByBalance"
        query = "from Account where
        balance = :balance")
})
```

JPQL

Positional parameters notation:

```
SELECT e
FROM Employee e
WHERE e.salary > ?1
```

```
query.setParameter(1, salary).getResultList();
```

JPQL

Named Parameters Notation:

```
SELECT e
FROM Employee e
WHERE e.name = :name
```

```
query.setParameter("name",employeeName);
```

JPQL

JPA Case Expressions:

SELECT CASE WHEN t.salary = 20000 THEN '20k' WHEN t.salary = 40000 THEN '40k' ELSE 'No salary' END FROM Teacher t

```
Query caseQuery = em.createQuery("SELECT CASE WHEN t.salary = 20000 THEN '20k' WHEN t.salary = 40000 THEN '40k' ELSE 'No salary' END FROM Teacher t");

List<String> salaries = (List<String>) caseQuery.getResultList();

                for (String salary : salaries) {

                        System.out.println(salary);

                }
```

Criteria API

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query = cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
query.select(teacher).where(cb.equal(teacher.get("firstName"),"prasad"));
```

```
TypedQuery<Teacher> typedQuery = em.createQuery(query);
List<Teacher> teachers = (List<Teacher>) typedQuery.getResultList();
    for (Teacher t : teachers) {
        System.out.println(t.getFirstName() + " "
            + t.getLastName() + " " + t.getSalary());
    }
```

Criteria API

Criteria API query roots

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query =
cb.createQuery(Teacher.class);
Root<Teacher> teacher = query.from(Teacher.class);
```

Criteria API

JPA ParameterExpression:

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query =
cb.createQuery(Teacher.class);
ParameterExpression<String> firstName =
cb.parameter(String.class,"firstName");
Root<Teacher> teacher = query.from(Teacher.class);
query.select(teacher);
query.where(cb.equal(teacher.get("firstName"), firstName));
```

Criteria API

JPA ParameterExpression:

```
TypedQuery<Teacher> typedQuery =
em.createQuery(query);
List<Teacher> teachers =
typedQuery.setParameter("firstName",
"prasad").getResultList();

for(Teacher t: teachers){
        System.out.println("Firstname = " + t.getFirstName() +
" Last Name = " + t.getLastName()+ " salary = " +
t.getSalary());
}
```

Criteria API

- **Selecting multiple expressions using tuples**

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Tuple> query = cb.createTupleQuery();
Root<Teacher> teacher = query.from(Teacher.class);
query.select(cb.tuple(teacher.get("firstName"), teacher.get("lastName")));

List<Tuple> results = em.createQuery(query).getResultList();
for (Tuple tuple : results) {
        System.out.println("First name = " + tuple.get(0) + " "
                + "Last Name = " + tuple.get(1));
}
```

SELECT t.firstName, t.lastName FROM Teacher t

Criteria API

- **OrderBy Clause**

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Teacher> query =
cb.createQuery(Teacher.class);
Root<Teacher> t = query.from(Teacher.class);
query.select(t).orderBy(cb.asc(t.get("salary")));
```

Criteria API

- **GroupBy Clause**

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Object[]> query =
cb.createQuery(Object[].class);
Root<Department> d = query.from(Department.class);
Join<Department,Teacher> teachers = d.join("teachers");
query.multiselect(d.get("name"),cb.count(teachers)).groupBy
(d.get("name"));
```

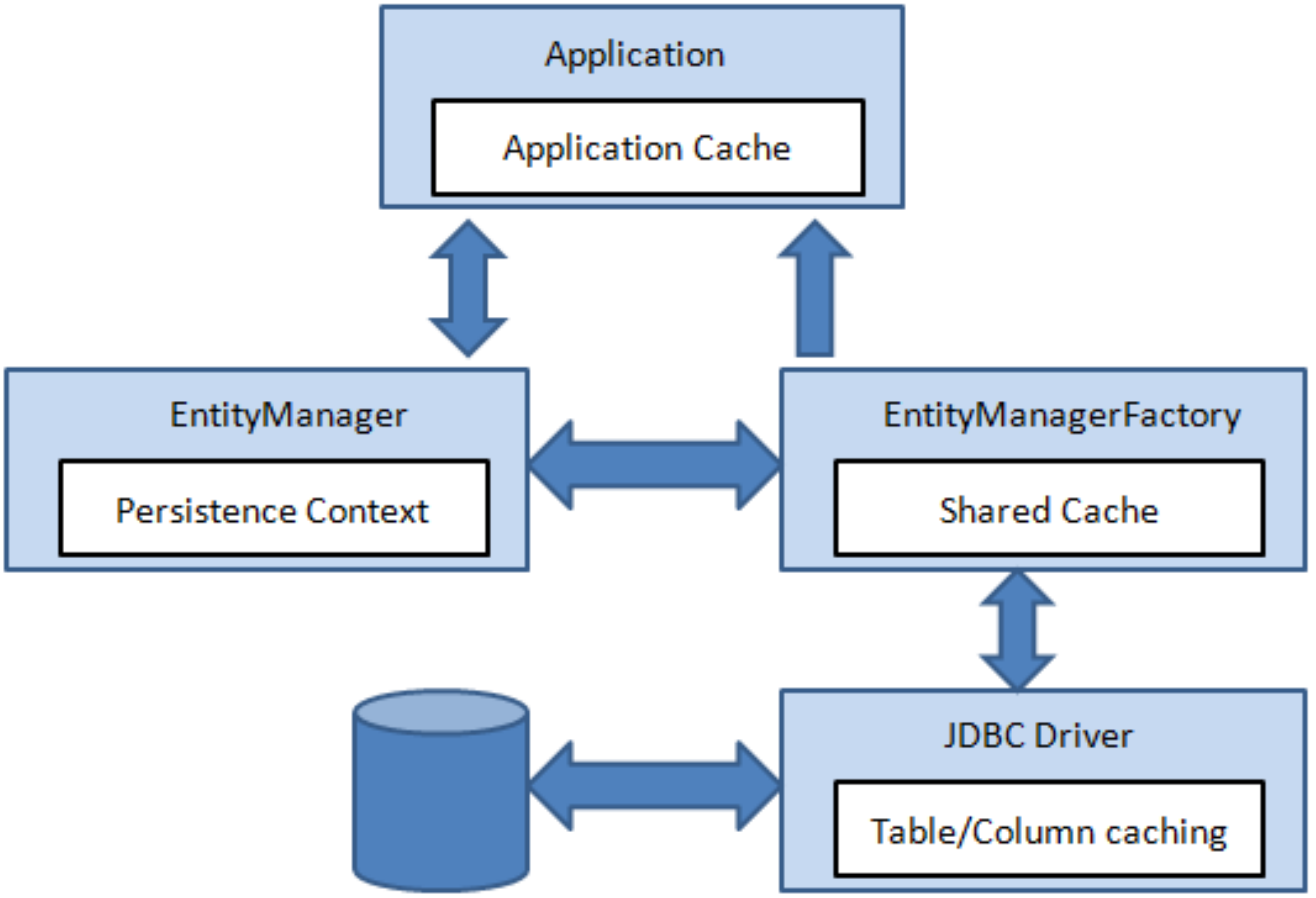
Criteria API

```
Query query = em.createQuery("SELECT d.name, COUNT(t) FROM
Department d JOIN d.teachers t GROUP BY d.name");

List<Object[]> results = query.getResultList();

        for (int i = 0; i < results.size(); i++) {
                Object[] arr = results.get(i);
                for (int j = 0; j < arr.length; j++) {
                        System.out.print(arr[j] + " ");
                }
                System.out.println();
                }
```

LifeCycle CallBack Methods

- @PrePersist :- Entity is notified when em.persist() is successfully invoked on entity.
- @PostPersist :- Notified when entity is persisted in database.
- @PreUpdate :- Notified when entity is modified.
- @PostUpdate :- Notified when updated state of entity is inserted in database
- @PreRemove :- Notified when remove operation is called for entity by entity manage.
- @PostRemove:- Notified when entity is deleted from database.

JPA Caching



JPA Caching

- Application Level:

Though application level cannot be considered as a real level of JPA caching, we can write some applications such that they hold references to jpa entities. These entities can become stale as they will be placed in application level.

- Persistence Context level:
  - This jpa caching level is the first real caching level. It is the first level from which a persistence provider can retrieve entities from memory and within transaction boundaries. Cached entities are not available when a transaction completes. In case of extended persistence contexts, the cache is unavailable after transaction is closed.

JPA Caching

- Shared Cache level:
  - JPA Caching in EntityManagerFactory is the real second level cache JPA provides. Cached entities from EntityManagerFactory are shared across all entitymanagers and hence the name shared cache.
- JDBC Cache level:
  - This isn't also a real JPA caching level. JDBC drivers can cache connections and statements. Some drivers can track tables and columns.

JPA Caching

Vehicle vehicle = em.find(Vehicle.class,1);

- Application from which **find** method is called looks in its local cache for vehicle with id = 1. It if doesn't find, it issues the call on entity manager.
- EntityManager checks for vehicle in its persistence context. This is the first real jpa caching level. If it is available in persistence context, it is returned otherwise, entity is searched in shared cache.
- If the entity is present in shared cache, an instance is created and put in persistence context level cache.
- If entity is not present in shared cache also, sql query is generated to select the entity row from database. The query result is used to create an object and it is passed back to shared cache.
- This newly created instance is put into shared cache and a new copy of the same instance is put into persistence context level cache where it becomes managed by entity manager.

Configure JPA Caching

- **shared-cache-mode** element in **persistence.xml**
- **@Cacheable** annotation

Cache mode settings are as below.

- **ALL** : All entities are stored in second level cache for a given persistence unit.
- **NONE** : No entities are stored in second level cache.
- **ENABLE_SELECTIVE**: Cache only those entities which are explicitly marked with **@Cacheable(true)** or **@Cacheable** as default value for **@Cacheable** is **true.**
- **DISABLE_SELECTIVE**: Cache all entities except for those who are marked with **@Cacheable(false).**
- **UNSPECIFIED:** Behaviour is undefined. Persistence provider defaults may apply.

Persistence.xml

```
<persistence-unit name="jpademo">
        <class>com.jpa.entities.Bike</class>
        <shared-cache-mode>ALL</shared-cache-mode>
        <properties>
            <!-- your connection properties here -->
        </properties>
</persistence-unit>
```

```
@Entity
@Cacheable(false)
public class Bike implements Serializable {
}
```


JPA Advantages

- Automatic scanning of deployed metadata
- Standardized configuration
  – Persistence Unit
- Standardized data access code, lifecycle, and querying capability that is fully portable
- Can override annotations with descriptor file

JPA Disadvantages

- **Though standard interfaces are nice, some-what lenient spec may present gaps when switching vendor implementations**
  - Not all inheritance strategies supported
  - 'Standardized' descriptor file is basically a wrapper around vendor specific implementations

- **Missing some beneficial aspects from Hibernate**
  - Query by Example, Query by Criteria (expected later)
  - EntityManager propagation across methods/objects
  - 2nd level Cache

Additional Resources

- **JSR 220 Specification**
  - http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html
  - http://java.sun.com/javaee/technologies/persistence.jsp
- **WebSphere Application Server V6.1 Feature Pack for EJB 3.0**
  - http://www1.ibm.com/support/docview.wss?rs=177&uid=swg21287579
- **Apache OpenJPA code**
  - http://openjpa.apache.org/

Thank You For Your Time

People matter, results count.