

Hibernate 3.5

Lesson 7: Java Persistence
API(JPA)

Lesson Objectives

➤ In this lesson, you will learn:

- Introduction to JPA
- Main JPA components
- Persistence Unit
- Entity classes
- Using Annotations with Hibernate
- Entity Manager API
- JPA – Life cycle
- Demonstration on Entity class



Introduction to JPA

- **Developed as part of Java Specification Request (JSR) 220**
 - Original goal to simplify EJB CMP entity beans
- **Simplifies the development of Java EE and Java SE applications using data persistence**
- **JPA standardized the ORM persistence technology for Java developers. JPA is not a product and can't be used as it is for persistence. It needs an ORM implementation to work and persist the Java Objects. ORM frameworks that can be used with JPA are Hibernate, Toplink, Open JPA etc.**
- **Usable both within Java SE environments as well as Java EE**
 - POJO based
 - Works with XML descriptors and annotations

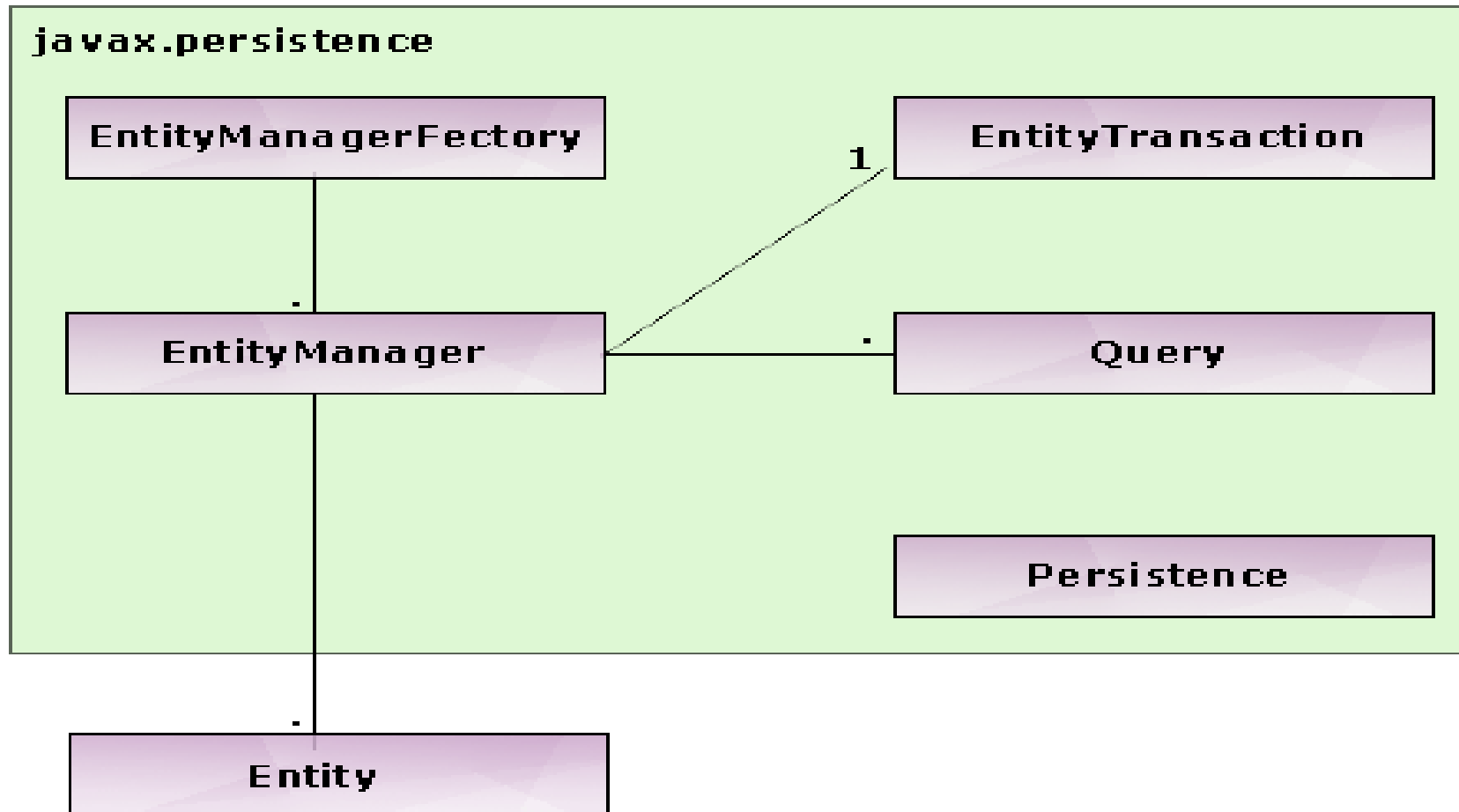
Benefits of JPA

- **Simplified Persistence technology**
- **ORM frameworks independence: Any ORM framework can be used**
- **Data can be saved in ORM way**
- **Supported by industry leaders**

Main JPA Components

- **Entity Classes**
- **Entity Manager**
 - Persistence Context
- **EntityManagerFactory**
- **EntityTransaction**
- **Persistence Unit**
 - persistence.xml
- **Java Persistence Query Language (JPQL)**
 - Query

Main JPA Components



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**
- **Identified in the persistence.xml configuration file**
- **Entity classes and configuration files are packaged together**
 - The JAR or directory that contains the persistence.xml is called the root of the persistence unit
 - Needs to be inside a META-INF directory
 - Whether or not inside a jar

Persistence context

➤ Persistence context

- A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle is managed by a particular entity manager. The scope of this context can either be the transaction, or an extended unit of work.

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>courses.hibernate.vo.Account</class>
    .
    .
    .
```

persistence.xml

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

Auto Entity Detection

➤ JPA provides for auto detection

- No need to list individual Entity classes in persistence.xml. Looks for annotated classes and mapping files
- Does NOT work with non-JPA Hibernate
- `<persistence-unit name="BankingApp">` Enabled by default
- ...
- `<property`
- `name="hibernate.archive.autodetection" value="class, hbm*" />` **Enabled by default**
- ...
- `</persistence-unit>`

Entity

➤ **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

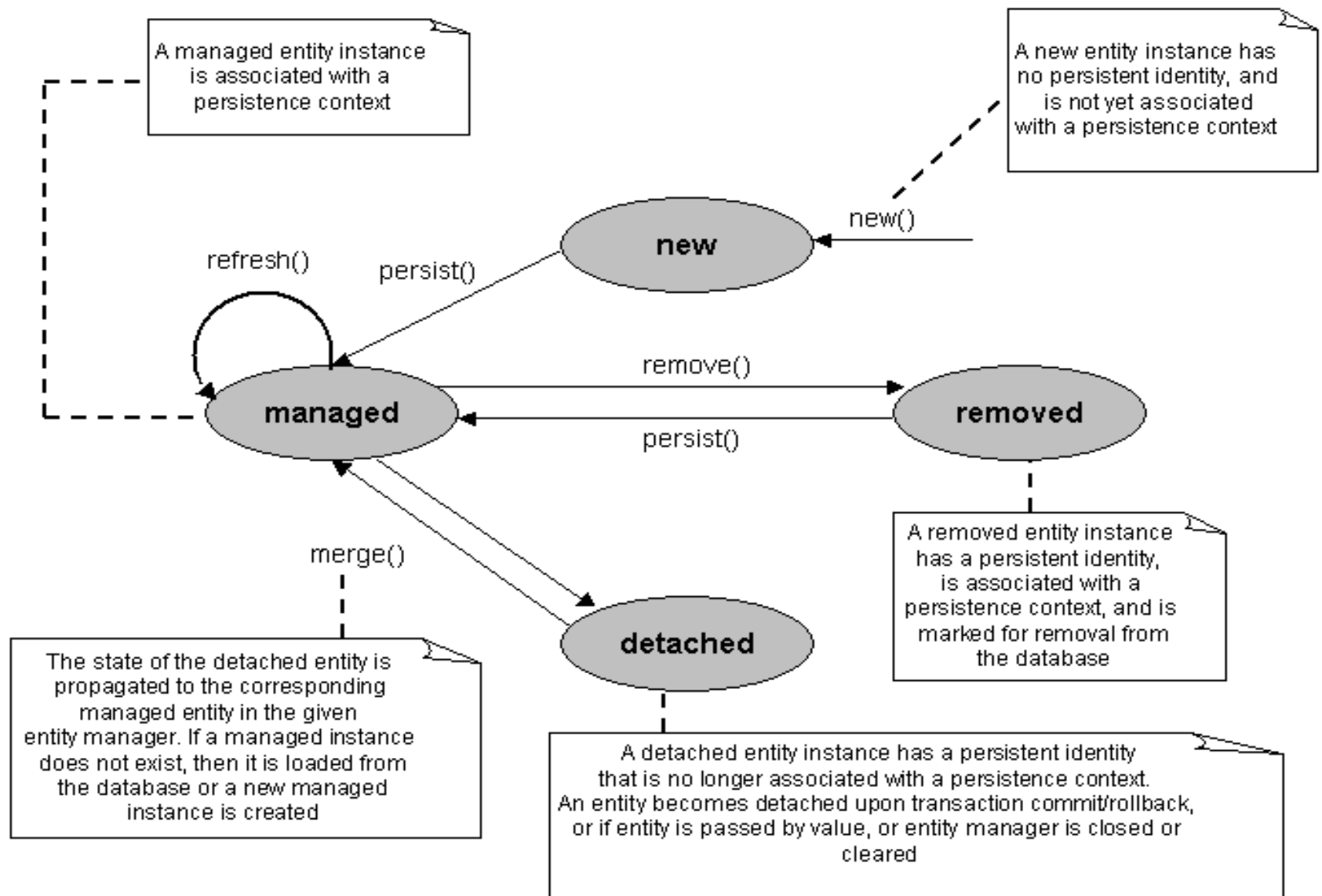
➤ **Contains persistent fields or properties**

- Attributes accessed through getters/setters are 'properties'
- Directly accessed attributes are referred to as 'fields'
- Can not combine fields and properties in a single entity
- Must define ALL attributes in your entity, even if they're not persisted
 - Mark as 'transient' if attribute is not managed

Entity states


- **New (transient):** an entity is new if it has just been instantiated using the new operator, and it is not associated with a persistence context. It has no persistent representation in the database and no identifier value has been assigned.
- **Managed (persistent):** a managed entity instance is an instance with a persistent identity that is currently associated with a persistence context.
- **Detached:** the entity instance is an instance with a persistent identity that is no longer associated with a persistence context, usually because the persistence context was closed or the instance was evicted from the context.
- **Removed:** a removed entity instance is an instance with a persistent identity, associated with a persistence context, but scheduled for removal from the database

JPA Lifecycle



orm.xml Mapping File

```
<entity-mappings
xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
version="1.0">
<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>
```



Set any defaults across the persistence unit entities

orm.xml Mapping File

```
<package>courses.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>
    <version name="version">
      <column name="VERSION" />
    </version>
  </attributes>
</entity></entity-mappings>
```

Notice no type definitions!

Notice no type definition

Annotations: Property Access

@Entity

```
public class Account {  
    private long accountId;
```

```
    ...
```

```
    @Id
```

```
    @GeneratedValue(strategy=GenerationType.AUTO)
```

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

```
    public long getAccountId() {...}
```

```
    public void setAccountId(long newId) {...}
```

```
    ...
```

```
}
```

Account Entity: Field Access

@Entity

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

Annotations

- The **@Entity** annotation marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope
- Other JPA 2 rules for an entity bean class are that the class must not be final, and that the entity bean class must be concrete
- Each entity bean has to have a primary key, which you annotate on the class with the **@Id** annotation. Typically, the primary key will be a single field, though it can also be a composite of multiple fields.
- By default, the **@Id** annotation will automatically determine the most appropriate primary key generation strategy to use—you can override this by also applying the **@GeneratedValue** annotation. This takes a pair of attributes: strategy and generator

Generator Type

➤ **There are four different types of primary key generators on `GeneratorType`, as follows:**

- **AUTO:** Hibernate decides which generator type to use, based on the database's support for primary key generation.
- **IDENTITY:** The database is responsible for determining and assigning the next primary key.
- **SEQUENCE:** Some databases support a `SEQUENCE` column type. See the “Generating Primary Key Values with `@SequenceGenerator`” section later in the chapter.
- **TABLE:** This type keeps a separate table with the primary key values. See the “Generating Primary Key Values with `@TableGenerator`”

Generating Primary Key Values with @Sequence Generator

- we can declare the primary key property as being generated by a database sequence. A sequence is a database object that can be used as a source of primary key values

```
@Id
@SequenceGenerator(name="seq1",sequenceName="HIB_SEQ")
@GeneratedValue(strategy=SEQUENCE,generator="seq1")
public int getId() {
    return id;
}
```

Entity Manager Factory

- The **EntityManagerFactory** is used to create an instance of **EntityManager** in **JavaSE** environment.
 - Similar to Hibernate **SessionFactory**
- **Created through a static method on Persistence**
- **EntityManagerFactory emf =**
- **Persistence.createEntityManagerFactory("BankingApp");**



Remember name of persistence unit

Entity Manager

- **The EntityManager interface is providing the API for interacting with the Entity.**
- **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**

Entity Manager

- `clear()` // clears the context
- `close()` // closes the manager
- `contains()` // checks for existing object
- `createNamedQuery()` // create named query
- `createNativeQuery()` // create SQL query
- `getTransaction()` // returns the current transaction
- `lock()` // locks an object
- `persist()` // makes an object persisten
- `refresh()` // refreshes an object from the database
- `remove()` // deletes an object from the database
- `find()` // retrieves an object from the database
- `setFlushMode()` // like Hibernate, but missing MANUAL

Application Managed Entity Manager

- Created and destroyed explicitly by the application
- Created through the EntityManagerFactory class
- EntityManagerFactory emf =
- Persistence.createEntityManagerFactory("BankingApp");
- EntityManager em = emf.createEntityManager();

EntityManager API

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="student")
public class Student {
    @Id
    @GeneratedValue
    private int id;
    public int getId() {return id;}
    public void setId(int id) {      this.id = id;
    }
    @Column(name="sname", length=100,nullable=false)
    private String sname;
    .
    .
    .
```

Persisting using EntityManager

```
EntityManagerFactory emf =  
Persistence.createEntityManagerFactory("jpa");//name of persistence unit  
EntityManager em = emf.createEntityManager();  
try{  
em.getTransaction().begin();  
Student st = new Student();  
st.setSname("Vinod Kumar");  
st.setSroll(25);  
st.setScourse("MBBS");  
em.persist(st);  
em.getTransaction().commit();  
}catch(Exception e){  
System.out.println(e.getMessage());  
}finally{em.close();}
```

Update using Entity Manager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
EntityManager em = emf.createEntityManager();
try{
    em.getTransaction().begin();
    //find
    Student stud1 = em.find(Student.class,1);
    stud1.setSname("Deepak");
    //update
    em.merge(stud1);
    em.getTransaction().commit();
}
catch(Exception e){
    System.out.println(e.getMessage());
}finally{em.close();
}
```

Reading Data

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
    EntityManager em = emf.createEntityManager();
    try{        em.getTransaction().begin();
                //Select all the record from student table
    Query query = em.createQuery("SELECT st FROM Student st");
    List lst = query.getResultList();
        Iterator it = lst.iterator();
        while (it.hasNext()){
            Student student = (Student) it.next();
            System.out.print("Id:"+student.getId());
            System.out.print(" Name:"+student.getSname());
            System.out.println(" Course:"+student.getScourse());
        }em.getTransaction().commit();}
    catch(Exception e){System.out.println(e.getMessage());}finally{em.close();
    }
```

Deleting data

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
EntityManager em = emf.createEntityManager();
try{
    em.getTransaction().begin();
    //find
    Student stud1 = em.find(Student.class,1);
    //delete
    em.remove(stud1);
    em.getTransaction().commit();
}
catch(Exception e){
    System.out.println(e.getMessage());
}
finally{em.close();}
}
```

Demo

JPACRUDApplication

Reference

- **you can refer hibernate refernce book:**
- **http://docs.jboss.org/hibernate/entitymanager/3.5/reference/en/html_single/#doe980**