Hibernate supports EJB 3 persistence annotations. The EJB 3 annotations are contained in thejavax.persistence package. If you avoid using Hibernate specific features in your application, then you will have the freedom to deploy your application in any environment that supports EJB 3. Anything imported from javax.persistence package tree is EJB 3 supported and anything imported from org.hibernate package tree is Hibernate specific.

The @**Entity** annotation is used to mark this class as an Entity bean. So the class should atleast have a package scope no-argument constructor.

The @**Table** annotation is used to specify the table to persist the data. The name attribute refers to the table name. If @Table annotation is not specified then Hibernate will by default use the class name as the table name.

The @**Id** annotation is used to specify the identifier property of the entity bean. The placement of the @Id annotation determines the default access strategy that Hibernate will use for the mapping. If the @Id annotation is placed over the field, then filed access will be used. Instead if it placed over the getter method of that field, then property access will be used. Here we use property access.

The @**GeneratedValue** annotation is used to specify the primary key generation strategy to use. If the strategy is not specified by default AUTO will be used.

The @**Column** annotation is used to specify the details of the column to which a field or property will be mapped. Here the courseId property is mapped to COURSE\_ID column in the COURSES table. If the @Column annotation is not specified by default the property name will be used as the column name.

The **cascade** option is used to cascade the required operations to the associated entity. If the cascade option is set to all then all the operations will be cascaded.

@OneToMany(cascade = CascadeType.ALL)

@JoinTable(name = "STUDENT\_PHONE", joinColumns = { @JoinColumn(name = "STUDENT\_ID") }, inverseJoinColumns = { @JoinColumn(name ="PHONE\_ID") })

<set name="studentPhoneNumbers" table="STUDENT\_PHONE"cascade="all">

<key column="STUDENT\_ID" />

<many-to-many  column="PHONE\_ID" unique="true"class="com.vaannila.student.Phone" />

</set>

@ManyToOne(cascade = CascadeType.ALL)

<many-to-one  name="studentAddress"class="com.vaannila.student.Address" column="STUDENT\_ADDRESS"cascade="all" not-null="true" />

@OneToOne(cascade = CascadeType.ALL)

<many-to-one  name="studentAddress"class="com.vaannila.student.Address" column="STUDENT\_ADDRESS"not-null="true" cascade="all" **unique="true"** />

@ManyToMany(cascade = CascadeType.ALL)

@JoinTable(name = "STUDENT\_COURSE", joinColumns = {@JoinColumn(name = "STUDENT\_ID") }, inverseJoinColumns = {@JoinColumn(name = "COURSE\_ID") })

<set name="courses" table="STUDENT\_COURSE" cascade="all">

<key column="STUDENT\_ID" />

<many-to-many  column="COURSE\_ID" class="com.vaannila.student.Course" />

</set>

The component element is used to map all the Address entity fields to the STUDENT table. In Hibernate terms the Address entity is called the **component** and it cannot have its own primary key, it uses the primary key of the enclosing Student entity.

@**Embedded**

public Address getStudentAddress() {

return this.studentAddress;

}

The @**Embedded** annotation is used to specify the Address entity should be stored in theSTUDENT table as a component.

@Embeddable annotation is used to specify the Address class will be used as a component. TheAddress class cannot have a primary key of its own, it uses the enclosing class primary key.

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name="connection.driver\_class">org.hsqldb.jdbcDriver</property>

<property name="connection.url">jdbc:hsqldb:hsql://localhost</property>

<property name="connection.username">sa</property>

<property name="connection.password"></property>

<!-- JDBC connection pool (use the built-in) -->

<property name="connection.pool\_size">1</property>

<!-- SQL dialect -->

<property name="dialect">org.hibernate.dialect.HSQLDialect</property>

<!-- Enable Hibernate's automatic session context management -->

<property name="current\_session\_context\_class">thread</property>

<!-- Disable the second-level cache -->

<property name="cache.provider\_class">org.hibernate.cache.NoCacheProvider</property>

<!-- Echo all executed SQL to stdout -->

<property name="show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hbm2ddl.auto">create</property>

<mapping resource="events/Event.hbm.xml"/>

</session-factory>

</hibernate-configuration>

private static final SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();

Session session = sessionFactory.getSessionFactory().openSession();

transaction = session.beginTransaction();

**Save:**

courseId = (Long) session.save(course);

**Select**:

List courses = session.createQuery("from Course").list();

**Update** :

Course course = (Course) session.get(Course.class, courseId);

course.setCourseName(courseName);

**Delete**:

Course course = (Course) session.get(Course.class, courseId);

session.delete(course);

transaction.commit();

|  |
| --- |
| **Q6) What is Association?**  Ans) Association **is a** relationship between two classes. In this relationship the object of one instance perform an action on behalf of the other class. The typical behaviour can be invoking the method of other class and using the member of the other class.  public class MyMainClass{  public void init(){  new OtherClass.init();  }  } |
| **Q7) What is Aggregation?**  Ans) Aggregation **has a** relationship between two classes. In this relationship the object of one class is a member of the other class. Aggregation always insists for a direction.  public class MyMainClass{  OtherClass otherClassObj = new OtherClass();  } |
| **Q8) What is Composition?**  Ans) Composition is a special type of aggregation relationship with a difference that its the compulsion for the **OtherClass**object (in previous example) to exist for the existence of **MyMainClass**. |