**Session object in Hibernate**

* Session provides a physical connectivity between your application and database.
* The Session will be established each time your application wants to do something with database.
* Session object will be provided by SessionFactory object.
* All the persistent objects will be saved and retrieved through Session object.
* The session object must be destroyed after using it.

**Session object Lifecycle**

* The lifecycle of a Session is bounded by the beginning and end of a logical transaction.
* The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes.
* Instances may exist in one of three states:

1. ***transient***: never persistent, not associated with any Session.
2. ***persistent***: associated with a unique Session.
3. ***detached***: previously persistent, not associated with any Session.

**Hibernate 5 configurations to create a session configuration xml (**hibernate.cfg.xml**) file:**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <!-- JDBC connection settings -->  <property name="connection.driver\_class">com.mysql.jdbc.Driver</property>  <property name="connection.url">jdbc:mysql://localhost/musicdb</property>  <property name="connection.username">root</property>  <property name="connection.password"/>  <!-- JDBC connection pool, use Hibernate internal connection pool -->  <property name="connection.pool\_size">5</property>  <!-- Defines the SQL dialect used in Hibernate's application -->  <property name="dialect">org.hibernate.dialect.MySQL5Dialect</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.internal.NoCachingRegionFactory</property>  <!-- Display and format all executed SQL to stdout -->  <property name="show\_sql">true</property>  <property name="format\_sql">true</property>  <!-- Drop and re-create the database schema on startup -->  <property name="hbm2ddl.auto">update</property>  <!-- Mapping to hibernate mapping files -->  <mapping resource="org/kodejava/example/hibernate/model/Label.hbm.xml"/>  </session-factory>  </hibernate-configuration> |

**Session Factory Helper Class to get session:**

|  |
| --- |
| package org.kodejava.example.hibernate.app;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  public class SessionFactoryHelper {  private static final SessionFactory sessionFactory;  static {  try {  // Build a SessionFactory object from session-factory config  // defined in the hibernate.cfg.xml file. In this file we  // register the JDBC connection information, connection pool,  // the hibernate dialect that we used and the mapping to our  // hbm.xml file for each pojo (plain old java object).  Configuration config = new Configuration();  sessionFactory = config.configure().buildSessionFactory();  } catch (Throwable e) {  System.err.println("Error in creating SessionFactory object."  + e.getMessage());  throw new ExceptionInInitializerError(e);  }  }  public static void main(String[] args) {  Session session = SessionFactoryHelper.getSessionFactory()  .getCurrentSession();  System.out.println("session = " + session);  }  /\*\*  \* A static method for other application to get SessionFactory object  \* initialized in this helper class.  \*/  public static SessionFactory getSessionFactory() {  return sessionFactory;  }  } |

* To create a SessionFactory we can define the configuration in hibernate.properties, hibernate.cfg.xml or create it programmatically.
* In this example we are using the hibernate.cfg.xml configuration file, which is mostly used when creating Hibernate application.

**Hibernate mapping with JPA (Java Persistence Annotations)**

* JPA entities are plain POJOs. Actually, they are Hibernate persistent entities. Their mappings are defined through JDK 5.0 annotations instead of hbm.xml files. JPA annotations are in the javax.persistence.\* package.
* @Entity - Marking a POJO as persistent entity: Every persistent POJO class is an entity and is declared using the @Entity annotation at the class level

|  |
| --- |
| @Entity  public class Employee implements Serializable {  ...  } |

* @Id - Marking primary key of an entity: @Id declares the identifier property of the given entity. The mapped column for the primary key of the entity is assumed to be the primary key of the primary table.
* @Table - Defining the table: @Table is set at the class level; it allows you to define the table, catalog, and schema names for your entity mapping. “@Table(name="EMPLOYEES")”

|  |
| --- |
| @Entity  @Table(name="EMPLOYEES")  public class Employee implements Serializable {  ...  } |

* If no @Table is defined the default values are used: the unqualified class name of the entity.The @Table element contains a schema and catalog attributes, if they need to be defined.
* You can also define unique constraints to the table using the @UniqueConstraint annotation in conjunction with @Table for a unique constraint bound to a single column, it is recommended to use the @Column.unique approach.

|  |
| --- |
| @Entity  @Table(name="EMPLOYEES",      uniqueConstraints = {@UniqueConstraint(columnNames={"email", "emp\_code"})})  public class Employee implements Serializable {  ...  } |

* @Column - Declaring column attributes: The column(s) used for a property mapping can be defined using the @Column annotation. You can use this annotation at the property level.

|  |
| --- |
| @Entity  @Table(name="EMPLOYEES")  public class Employee implements Serializable {        @Id      @Column(name="EMP\_ID")      private Long empId;        private String name;        private String department;        private Long salary;        @Column(name="JOINED\_ON")      private Date joinedOn;      ...  } |