Q.1) What is Hibernate?

**Ans:**

1. Hibernate is an **ORM** (Object Relational mapping) implementation
2. Hibernate is an **Open source**
3. Hibernate is a **framework**
4. Hibernate is an open-source and lightweight ORM tool that is used to

**store, manipulate**,and **retrieve data** from the database.

1. Hibernate can run **with or without server.**

Q.2) What is ORM?

**Ans:**

1. ORM is an acronym for Object/Relational mapping. It is a programming strategy **to map object** with the **data stored in the database**.
2. It simplifies data **creation, data manipulation, and data access.**

Q.3) What is a framework?

**Ans:**

1.A framework is **reusable semi finished** application that can be customized to

develop a specific application.

Q.4)What are the features of hibernate?

**Ans:**

1. Hibernate **persists java objects into database.**
2. **Database independent.**
3. Hibernate generates **efficient querie**s for java application to communicate with Database.
4. In hibernate we only have **Un-checked** exceptions, so no need to write **try, catch**, or no need to write throws.
5. it supports **synchronization** between in-memory java objects and relational records.
6. Hibernate supports **Inheritance, Associations, Collections**.
7. Hibernate supports a special **query language(HQL)** which is Database vendor independent.
8. Hibernate supports annotations, apart from **XML.**

### Q.5)  What are the advantages of Hibernate over JDBC?

### **Clean Readable Code**

### **HQL (Hibernate Query Language)**

### **Transaction Management**

### **Exception Handlin**

### OOPs features like inheritance, associations and also supports collections.

Q.6) What are the disadvantages of hibernate?

**Ans:**

1. Since hibernate generates lots of SQL statements at runtime so it is slower **than pure JDBC .**
2. Hibernate is **not much flexible in case of composite mapping**.
3. Hibernate does not support **some type of queries** which are supported by JDBC.
4. **Boilerplate code issue**, actually we need to write same code in several files in the same application, but spring eliminated this.

Q.7) Hibernate Architecture?

The Hibernate architecture is categorized in four layers.

1. **Java application layer**
2. **Hibernate framework layer**
3. **Backhand api layer**
4. **Database layer**

Let's see the diagram of hibernate architecture:



**Fig 1.1 This is the high level architecture of Hibernate with *mapping file and configuration file.***



***Fig 1.2 Componments Of Hibernate.***

## Elements of Hibernate Architecture

#### 1.SessionFactory

#### The SessionFactory is a factory of session and client of ConnectionProvider.

#### It holds second level cache (optional) of data.

#### The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

1. SessionFactory is a **thread-safe object**, many threads cannot

access it simultaneously.

#### 2.Session

1. The session object provides an interface between the application and **data stored in the database.**
2. It is a short-lived object and wraps the JDBC connection.
3. It is factory of Transaction, **Query and Criteria**.
4. It holds **a first-level cache** (mandatory) of data.
5. The org.hibernate.Session interface provides methods **to insert, update and delete the object.**
6. Session is **not a thread-safe object**, many threads can access it simultaneously.

#### 3.Transaction

1. The transaction object specifies the **atomic unit of work**. It is optional.
2. The org.hibernate.Transaction interface **provides methods for transaction management**.

Using Session Object we can create Transaction object in two ways.

1. Transaction transaction =Session.getTransaction();

2.Transaction transction = session.beginTransaction();

#### 4.ConnectionProvider

1. It is a factory of **JDBC connections**.
2. It abstracts the application from **DriverManager or DataSource.**

#### 5.TransactionFactory

1.It is a factory of **Transaction**. It is optional.

Q.8) Why SessionFactory is heavy weight?

**Ans:**

SessionFactory encapsulates Session objects, **Connections, Hibernate-properties, cashing** and **mappings.**

Q.9) What are the Simple Hibernate Application Requirements?  
**Ans:**

1. Entity class .(POJO class)
2. Mapping file(Required if you are not using annotations).
3. Configuration file.
4. DAO class (Where we write our logic to work with database).

Q.10) What is hibernate configuration file?

**Ans:**

1. 1.It is an XML file in which **database connection details** (username, password, url,

driver class name) and , **Hibernate Properties**(dialect, show-sql, second-level-cache ... etc) and **Mapping file name(s)** are t specified to the hibernate .

1. Hibernate uses this file to establish connection **to the particular database server**.
2. Standard for this file **is <hibernate.cgf.xml>.**
3. We must create **one configuration file** for each database we are going to use, suppose if we want to connect **with 2 databases.**
4. <hibernate-configuration>
6. <session-factory>
7. <property name="hbm2ddl.auto">update</property>
8. <property name="dialect">org.hibernate.dialect.MySQLDailect </property>
9. <property name="connection.url">jdbc:mysql:localhost:3306/Database Name </prop>
10. <property name="connection.driver\_class">com.mysql.jdbc.Driver</pr>
11. <property name="connection.username">root</property>
12. <property name="connection.password">root</property>
14. <mapping resource="employee.hbm.xml"/>
15. <mapping resource="address.hbm.xml"/>
17. </session-factory>
18. </hibernate-configuration>

Q.11) What is hibernate mapping file?

**Ans:**

1. In this file hibernate application developer specify the mapping from **entity class name** to **table name** and **entity properties names** to table column names.
2. Standard name for this file **is <hibernate.hbm.xml>**
3. <hibernate-mapping>
4. <**class** name="com.javatpoint.Address" table="address22">
5. <id name="addressId">
6. <generator **class**="increment"></generator>
7. </id>
8. <property name="addressLine1"></property>
9. <property name="city"></property>
10. <property name="state"></property>
11. <property name="country"></property>
12. <property name="pincode"></property>
13. </**class**>
14. </hibernate-mapping>

Q.12) What do you know about dialect in Hibernate?

**Ans:**

1. Dialect class is a simple java class, which contains mapping between java language **data type and database data type.**
2. Dialect class contains **queries format for predefined hibernate methods**.
3. All Dialect classes must **extend** 'Dialect' (abstract) class.
4. If we want we can write our **own dialect by extending Dialect class**
5. Dialect class is used **convert HQL queries into database** specific queries.
6. the Dialect class name in **hibernate.cfg.xml file**

Q.13) Explain about annotations which are used in our application to map entity to table?

**Ans:**

we used the following annotations.

1. [*@Entity*](mailto:@Entity)

annotation marks this class as an entity(POJO Class).

1. [*@table*](mailto:2.@table)

annotation specifies the table name where data of this entity is to be persisted.

1. [*@id*](mailto:3.@id)

It is used to Automatically generate Primary Key.

1. [*@column*](mailto:4.%20@column)

annotation specifies the details of the column for this property or field.

1. *@generateValue :*It is used to Auto Increament specified column*.*
2. *@ Temporal*

Data can have DATE, TIME, or TIMESTAMP precision.

All the above annotations we are taking from **java.persistence** package

Q.14) Explain Formula annotation in Hibernate.?

1. @Formula annotation is used to calculate a given entity attribute using an SQL query expression.
2. It defines a formula (derived value) which is a SQL fragment that acts as a @Column alternative in most cases. It represents read-only state.

**@Entity**

**@Table**(name="area")

**public** **class** **Area** **implements** Serializable {

**private** **static** **final** **long** serialVersionUID = **1L**;

**@Id**

**@Column**(name="id")

**private** **int** id;

**@Column**(name="length")

**private** **int** length;

**@Column**(name="width")

**private** **int** width;

**@Formula**(" length \* width ")

**public** **long** area;

...

}

Q.15) What is Configuration object?

**Ans:**

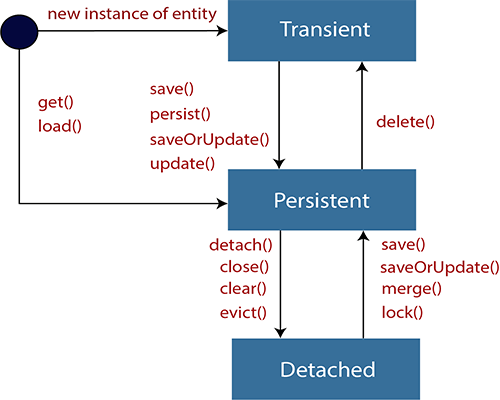
1. Object Oriented representation of **hibernate configuration file along with mapping file** is known as - Configuration object.
2. By default Hibernate **reads configuration file** with name **"hibernate.cfg.xml"** which is located in **"classes"** folder.
3. Configuration object Stores the **configuration file data in different variables**. Finally all these **variables are grouped and create one high level hibernate object** called SessionFactory object.
4. So Configuration **object only meant for creating SessionFactory object.**

# Q.16) Hibernate Lifecycle

In Hibernate, either we create an object of an entity and save it into the database, or we fetch the data of an entity from the database.

The Hibernate lifecycle contains the following states: -

* Transient state
* Persistent state
* Detached state



1.Transient state

* The transient state is the initial state of an object.
* Once we create an instance of POJO class, then the object entered in the transient state.
* Here, an object is not associated with the Session. So, the transient state is not related to any database.
* Hence, modifications in the data don't affect any changes in the database.
* The transient objects exist in the heap memory. They are independent of Hibernate.

Employee e=**new** Employee(); //Here, object enters in the transient state.

e.setId(101);

e.setFirstName("Gaurav");

e.setLastName("Chawla");

2.Persistent state

* As soon as the object associated with the Session, it entered in the persistent state.
* Hence, we can say that an object is in the persistence state when we save or persist it.
* Here, each object represents the row of the database table.
* So, modifications in the data make changes in the database.

We can use any of the following methods for the persistent state.

1. session.save(e);
2. session.persist(e);
3. session.update(e);
4. session.saveOrUpdate(e);
5. session.lock(e);
6. session.merge(e);

3.Detached State

* Once we either close the session or clear its cache, then the object entered into the detached state.
* As an object is no more associated with the Session, modifications in the data don't affect any changes in the database.
* However, the detached object still has a representation in the database.
* If we want to persist the changes made to a detached object, it is required to reattach the application to a valid Hibernate session.
* To associate the detached object with the new hibernate session, use any of these methods - load(), merge(), refresh(), update() or save() on a new session with the reference of the detached object.

We can use any of the following methods for the detached state.

1. session.close();
2. session.clear();
3. session.detach(e);
4. session.evict(e);

### Q.17) Hibernate saveOrUpdate?

### saveOrUpdate results into insert or update queries based on the provided data. If the data is present in the database, update query is executed.

### Q.18) What is the difference between update and merge method?

|  |  |
| --- | --- |
| Update | Merge |
| Update means to edit something. | Merge means to combine something. |
| update() should be used if the session doesn't contain an already persistent state with the same id. It means an update should be used inside the session only. | merge() should be used if you don't know the state of the session, means you want to make the modification at any time. |

### Q.19) What is the difference between session.save() and session.persist() method?

|  |  |
| --- | --- |
| Save() | Persist() |
| returns the identifier (Serializable) of the instance. | Return nothing because its return type is void. |
| Syn: public Serializable save(Object o) | Syn: public void persist(Object o) |
| It is Only supported Hibernate | It is Supported JPA |
| It is Object Store In Database | It is Also Object Store In Database |

### Q.20) Differentiate between save() and saveOrUpdate() methods in hibernate session ?

|  |  |
| --- | --- |
| save() | saveOrUpdate() |
| save() generates a new identifier and INSERT record into a database | Session.saveOrUpdate() can either INSERT or UPDATE based upon existence of a record. |
| The insertion fails if the primary key already exists in the table. | In case the primary key already exists, then the record is updated. |
| returns the identifier (Serializable) of the instance. | The return type of the saveOrUpdate() method is void. |

### Q.21) What is the difference between get and load method?

|  |  |
| --- | --- |
| Get() | Load() |
| Returns null if an object is not found. | Throws ObjectNotFoundException if an object is not found. |
| get() method always hit the database. | load() method doesn't hit the database. |
| It returns the real object, not the proxy. | It returns proxy object. |
| It should be used if you are not sure about the existence of instance. | It should be used if you are sure that instance exists. |

Q.22) How many types of association mapping are possible in hibernate?

There can be 4 types of association mapping in hibernate.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

### Q.23) it possible to perform collection mapping with One-to-One and Many-to-One?

No, collection mapping can only be performed with One-to-Many and Many-to-Many

Q.24) How do I create an immutable class in hibernate?

Configuring the entity class property mutable to false (mutable="false"), class becomes an immutable class. By default, it is mutable="true".

### Q.25) What is the difference between Lazy Loading and Eager Loading method?

|  |  |
| --- | --- |
| Lazy | Eager |
| In Lazy loading, associated data loads only when we explicitly call getter or size method. | In Eager loading, data loading happens at the time of their parent is fetched |
| ManyToMany and OneToMany associations used lazy loading strategy by default. | ManyToOne and OneToOne associations used lazy loading strategy by default. |
| It can be enabled by using the annotation parameter :  fetch = FetchType.LAZY | It can be enabled by using the annotation parameter :  fetch = FetchType.EAGER |
| Initial load time much smaller than Eager loading | Loading too much unnecessary data might impact performance |

### Q.26) What is hibernate caching?

### **Hibernate caching** is the strategy for improving the application performance by pooling objects in the cache so that the queries are executed faster.

### Hibernate caching is particularly useful when fetching the same data that is executed multiple times.

### Rather than hitting the database, we can just access the data from the cache.

### This results in reduced throughput time of the application.

### Q.27)What is the difference between first level cache and second level cache?

|  |  |
| --- | --- |
| first level cache | Second level cache |
| First Level Cache is associated with Session. | Second Level Cache is associated with SessionFactory. |
| It is enabled by default. | It is not enabled by default. |
| The first level cache is available only until the session is open, once the session is closed, the first level cache is destroyed. | The second-level cache is available through the application’s life cycle, it is only destroyed and recreated when an application is restarted. |

### Q.28) Can you tell the difference between getCurrentSession and openSession methods?

|  |  |
| --- | --- |
| getCurrentSession | openSession |
| This method returns the session bound to the context. | This method always opens a new session. |
| This session object gets closed once the session factory is closed. | It's the developer’s responsibility to close this object once all the database operations are done. |
| In a single-threaded environment, this method is faster than openSession(). | In single threaded environment, it is slower than getCurrentSession()single-threadeda |

Q.29) What is HQL (Hibernate Query Language)?

Hibernate Query Language is known as an object oriented query language. It is like structured query language (SQL).

1. You don't need to learn SQL
2. Database independent
3. Simple to write query

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database.

This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

Q.30) Query and Criteria Interface.

**Ans:**

This interface allows the user to perform queries and also control the flow of the query execution

Q.31) What is POJO?

**Ans:**

1. POJO stands for **plain old java objects**.
2. These are just basic JavaBeans that have defined **setter and getter** methods for all the properties that are there in that bean.
3. Besides they can also have **some business logic related to that property.**

Q.32) Hibernate: Can a Entity Class be declared final?

**Yes**, a Hibernate Entity class can be declared final, however it is

not a good practice.

Q.33) How to call stored procedure in Hibernate?

In Hibernate, there are 3 approaches to call a database store procedure.

* Use **createSQLQuery()**to call a store procedure directly.
* Use **@NamedNativeQueries** annotation.
* Declare your store procedure inside the **sql-query** tag.

Q.34) Explain transaction management in hibernate.

1. In hibernate framework, we have **Transaction** interface that defines the unit of work.
2. It maintains abstraction from the transaction implementation (JTA,JDBC).
3. A transaction is associated with Session and instantiated by calling **session.beginTransaction**().

The transaction interfaces defines the below methods.

1. void **begin**() starts a new transaction.
2. void **commit**() ends the unit of work unless we are in FlushMode.NEVER.
3. void **rollback**() forces this transaction to rollback.
4. void **setTimeout**(int seconds) it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. boolean **isAlive**() checks if the transaction is still alive.
6. void **registerSynchronization**(Synchronization s) registers a user synchronization callback for this transaction.
7. boolean **wasCommited**() checks if the transaction is commited successfully.
8. boolean **wasRolledBack**() checks if the transaction is rolledback successfully.

Q.35) That are the Collection types in Hibernate ?

* Bag
* Set
* List
* Array
* Map
* Properties

Q.36) List out the design patterns used in Hibernate framework.?

* **Domain Model Pattern** - An object model of the domain that incorporates both behavior and data,
* **Data Access Object** (DAO) Design Pattern,
* **Abstract Factory,**
* **Data Mapper,**
* **Proxy** for lazy loading,
* Object-Relational Mapping (**ORM**),
* **Query Object** for Criterion API,
* and **facade.**

Q.37) Difference between criteria and HQL In hibernate.?

|  |  |
| --- | --- |
| criteria | HQL |
| Criteria query performs only SELECT operations. | HQL query performs both SELECT and NON-SELECT operations.HQL can be used to perform SELECT, INSERT, UPDATE, DELETE. |
| Criteria supports pagination. | HQL doesn't support pagination. |
| Criteria query is safe from SQL injection because of dynamic query generation. | HQL queries are either fixed or parametrized, SQL injection may happen if the developer doesn't parameterize properly. |

Q.38) Does hibernate support polymorphism?

**Yes**, hibernate provides complete support to polymorphism.

Polymorphism queries and associations are supported at all the mapping strategies of hibernate.

Q.39) Explain HibernateTemplate.

1. HibernateTemplate is a helper utility class that provides different methods for querying/retrieving data from the database.
2. It also converts checked Hibernate Exceptions to unchecked DataAccessExceptions.
3. It manages the session and transactions by automatically opening and closing when you execute the code.

Q.40) How objects can be identified in Hibernate?

Object identification can be done in hibernate in following 3 ways.

* Using Object Identity: Using == operator.
* Using Object Equality: Using equals() method.
* Using database identity: Relational database objects can be identified if they represent same row.

Q.41) What is CRUD?

A CRUD operation deals with **creating, retriving , updating and deleting** from the table.

Q.42) What are Scalar queries in Hibernate?

SQL queries that gets a list of **scalars** (values) is referred as scalar queries.

*sess.createSQLQuery("SELECT \* FROM Emp").list();*

These will return a List of Object arrays (Object[]) with scalar values for each column in the Emp table.

Q.43) Explain the role of Configurationclass in Hibernate ?

1. The org.hibernate.cfg.Configuration is used to build an immutable org.hibernate. SessionFactoryobject .
2. Configuration class object activates hibernate software and configure () is the factory method of hibernate.cfg.Configuration class which reads configuration properties from hibernate.cfg.xml file.
3. builtSessionFactory() method uses hibernate.cfg.xml properties of Configuration object Creates jdbc connection pool.

Q.44) Explain Hibernate named query.

1. Hibernate named queries lets developer to put all HQL into the XML mapping file or via annotation so that it is easy to maintain and separated from Java code.
2. The named query is supported in both HQL or native SQL.

The queries can be retrieved using the query names as shown below.

1. Query query = session.getNamedQuery("HQL\_GET\_ALL\_EMPLOYEE");

Q.45) What is cascade in Hibernate?

Cascading consists in propagating the **Parent entity state transition to one or more Child entities**, and it can be used for both unidirectional and bidirectional associations.

Q.46)Explain JPA Entity Object Life Cycle.

**Ans:**

The life cycle of entity objects consists of four states:

**1.New,2. Managed, 3.Removed and 4.Detached**.

1. When an entity object is initially created its state is **New**.

In this state the object is not yet associated with an EntityManager and has no representation in the database.

1. An entity object becomes **Managed** when it is persisted to the database.
2. A managed entity object can also be retrieved from the database and move it to **removed**, by using the EntityManageris remove method within an active transaction.
3. **Detached** represents entity objects that have been disconnected from the EntityManager.

### 

### Q.47) How to make an immutable class in hibernate?

Ans: If you mark a class as mutable="false", the class will be treated as an immutable class.

By default, it is mutable="true".

Q.48) Difference between JPA and Hibernate.?

|  |  |
| --- | --- |
| JPA | Hibernate |
| Java Persistence API (JPA) defines the management of relational data in the Java applications. | Hibernate is an Object-Relational Mapping (ORM) tool which is used to save the state of Java object into the database. |
| t is just a specification. Various ORM tools implement it for data persistence. | It is one of the most frequently used JPA implementation. |
| It is defined in **javax.persistence** package. | It is defined in **org.hibernate** package. |
| he **EntityManagerFactory** interface is used to interact with the entity manager factory for the persistence unit. | It uses **SessionFactory** interface to create Session instances. |
| It uses **Java Persistence Query Language** (JPQL) as an object-oriented query language to perform database operations. | It uses **Hibernate Query Language** (HQL) as an object-oriented query language to perform database operations. |

49) Hibernate Abstract methods

**public interface ProductDao {**

**public List<Product> getAllProductData();**

**public Product getProductDataById(int productId);**

**public void saveAllProductData(List<Product> product);**

**public void saveProductData(Product product);**

**public void updateProductData(int productId, Product product);**

**public void deleteProductDataById(int productId);**

**public void deleteAllProductData();**

}

50) DaoImpl Methods

package com.csi.Dao;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.AnnotationConfiguration;

import com.csi.model.Product;

public class ProductDaoImpl implements ProductDao {

**private static SessionFactory factory = new AnnotationConfiguration().configure().buildSessionFactory();**

**@Override**

public List<Product> **getAllProductData(**) {

Session session = factory.openSession();

List<Product> custlist = session.createQuery("from Product").list();

return custlist;

}

**@Override**

public Product **getProductDataById**(int productId) {

Session session = factory.openSession();

Product custlist = (Product) session.load(Product.class, productId);

return custlist;

}

**@Override**

public void **saveAllProductData**(List<Product> product) {

Session session = factory.openSession();

for(Product pp: product)

{

Transaction transaction = session.beginTransaction();

session.save(pp);

transaction.commit();

}

}

**@Override**

public void **saveProductData**(Product product) {

Session session = factory.openSession();

Transaction transaction = session.beginTransaction();

session.saveOrUpdate(product);

transaction.commit();

}

**@Override**

public void **updateProductData**(int productId, Product product) {

Session session = factory.openSession();

Transaction transaction = session.beginTransaction();

List<Product> custlist = session.createQuery("from Product").list();

for (Product p : custlist) {

if (p.getProductId() == productId) {

p.setProductName(product.getProductName());

p.setProductCode(product.getProductCode());

p.setProductPrice(product.getProductPrice());

p.setProductLaunchDate(product.getProductLaunchDate());

session.update(p);

transaction.commit();

}

}

}

**@Override**

public void **deleteProductDataById**(int productId) {

Session session = factory.openSession();

Transaction transaction = session.beginTransaction();

List<Product> custlist = session.createQuery("from Product").list();

for (Product p : custlist) {

if (p.getProductId() == productId) {

session.delete(p);

transaction.commit();

}

}

}

**@Override**

public void **deleteAllProductData**() {

Session session = factory.openSession();

Transaction transaction = session.beginTransaction();

List<Product> custlist = session.createQuery("from Product").list();

for (Product p : custlist) {

session.delete(p);

}

transaction.commit();

}

}