Part - A

1. **What is Hibernate?**
   Hibernate is a powerful and widely used object-relational mapping (ORM) framework for Java applications.

2. **What is ORM?**
   The Hibernate Object-Relational (O-R) mapping process is a crucial aspect of the framework that allows developers to seamlessly integrate Java objects with relational databases.

3. **What is HQL?**
   HQL is an object-oriented query language provided by Hibernate. It allows developers to express database queries in terms of their Java objects, rather than SQL. HQL is translated into SQL by Hibernate.

4. **Define advantage of HQL**
   - Database independent
   - Supports polymorphic queries
   - Easy to learn for Java Programmer
   - User friendly syntax.

5. **What are Criteria queries?**
   HCQL stands for Hibernate Criteria Query Language. It is mainly used in search operations and works on filtration rules and logical conditions.

6. **Define Native SQL.**
   Native SQL queries are useful when you need to perform complex queries that cannot be expressed using Hibernate's Query Language (HQL) or Criteria API. Native SQL queries can be used to perform complex joins, aggregate functions, and subqueries.

7. **What is Java Persistence?**
   JPA (Java Persistence API) is a specification for ORM (Object-Relational Mapping) frameworks in Java. The JPA specification defines a set of interfaces for executing native SQL queries, including the EntityManager interface and the Query interface

Part - B

1. **Write short notes about the Hibernate concepts.**

   Hibernate is a powerful and widely used object-relational mapping (ORM) framework for Java applications. Here are some key concepts related to Hibernate in advance Java:

   1) **Object-Relational Mapping (ORM):**
      Hibernate facilitates the mapping of Java objects to database tables and vice versa. It allows developers to work with Java objects rather than SQL queries, making it easier to interact with databases.

   2) **Hibernate Configuration:**
      Hibernate requires configuration through a file named `hibernate.cfg.xml`. This file contains database connection details, such as database URL, driver class, username, and password.

   3) **Session Factory:**
      The Session Factory is a crucial concept in Hibernate. It's a factory class that produces sessions, which serve as a gateway for the application to interact with the database. The Session Factory is a heavyweight object, and typically, one is created for the entire application.

   4) **Session:**
      A Session in Hibernate is similar to a database connection in JDBC. It represents a single threaded unit of work and is used to perform database operations. Sessions are lightweight and created on demand.

   5) **Hibernate Mapping:**
      Hibernate uses XML or annotations to define the mapping between Java objects and database tables. This mapping specifies how the fields of a Java class correspond to the columns in a database table.

   6) **Hibernate Query Language (HQL):**
      HQL is an object-oriented query language provided by Hibernate. It allows developers to express database queries in terms of their Java objects, rather than SQL. HQL is translated into SQL by Hibernate.

   7) **Criteria API:**
      Hibernate provides a Criteria API that allows developers to create queries programmatically using a set of criteria. It's an alternative to HQL and is often used when the queries are more dynamic**.**

   8) **JPA (Java Persistence API):**
      JPA (Java Persistence API) is a specification for ORM (Object-Relational Mapping) frameworks in Java. The JPA specification defines a set of interfaces for executing native SQL queries, including the EntityManager interface and the Query interface

2. **Explain the Hibernate O-R mapping process.**

The Hibernate Object-Relational (O-R) mapping process is a crucial aspect of the framework that allows developers to seamlessly integrate Java objects with relational databases. Here's an overview of the Hibernate O-R mapping process:

1) **Entity Classes:**

   Start with defining your Java classes that represent entities in your application. These classes should encapsulate the data and business logic related to your application's domain.

2) **Mapping Metadata:**

   Create mapping metadata to establish the relationship between your Java classes and database tables. This metadata is specified using either XML mapping files or annotations within the entity classes.

3) **SessionFactory Creation:**

   Once the mapping metadata is in place, you need to create a SessionFactory. The SessionFactory is a heavyweight, thread-safe object that is typically created once during the application startup.

4) **Session Creation:**

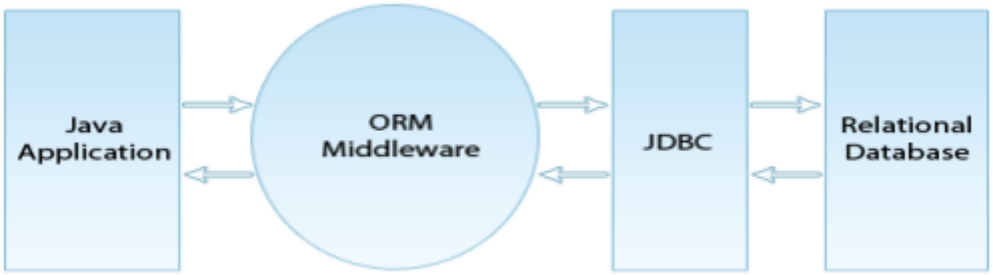
   Obtain a Session from the SessionFactory whenever you need to interact with the database. The Session is a short-lived, lightweight object that represents a single unit of work with the database.

5) **CRUD Operations:**

   Use the Session to perform CRUD (Create, Read, Update, Delete) operations on your Java objects. Hibernate takes care of translating these operations into corresponding SQL queries based on the O-R mapping defined earlier.

6) **Hibernate Query Language (HQL) and Criteria API:**

   Apart from basic CRUD operations, Hibernate provides HQL and the Criteria API for more complex querying. HQL is an object-oriented query language, and the Criteria API allows you to build queries programmatically.



The O-R mapping process in Hibernate simplifies database interactions by allowing developers to work with Java objects directly, abstracting the underlying database details and reducing the need for handwritten SQL queries.

3. What are the Hibernate Advantages?

Hibernate offers several advantages in advanced Java development, making it a popular choice for interacting with databases. Here are some key advantages of using Hibernate:

1) **Object-Relational Mapping (ORM):**
   Hibernate provides a powerful ORM framework, allowing developers to work with Java objects instead of dealing directly with SQL queries.

2) **Database Independence:**
   HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries, providing a level of database independence. Hibernate takes care of translating the Java objects to the appropriate database-specific queries.

3) **Open Source and Lightweight:**
   Hibernate framework is open source under the LGPL license and lightweight.

4) **Fast Performance:**
   The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

5) **Automatic Table Creation:**
   Hibernate framework provides the facility to create the tables of the database automatically. So, there is no need to create tables in the database manually.

6) **Simplifies Complex Join:**
   Fetching data from multiple tables is easy in hibernate framework.

7) **Provides Query Statistics and Database Status:**
   Hibernate supports Query cache and provide statistics about query and database status.

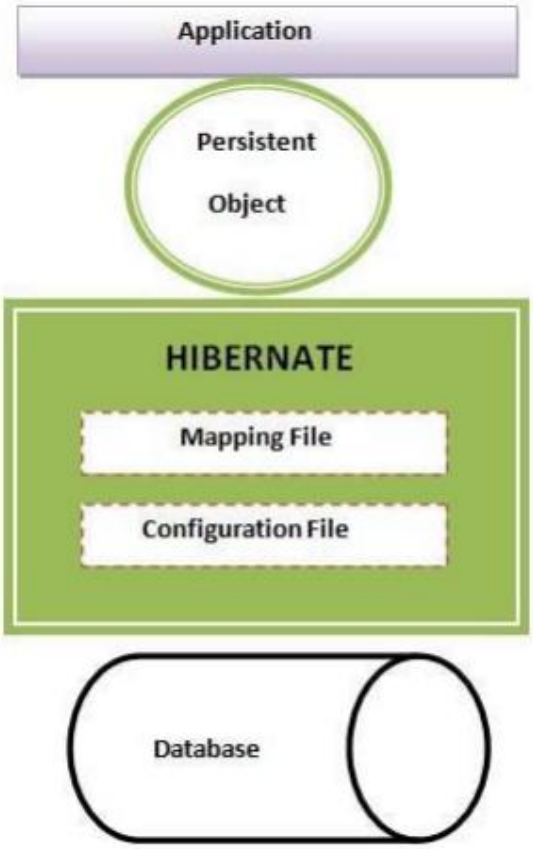
4. Explain the Hibernate Architecture.

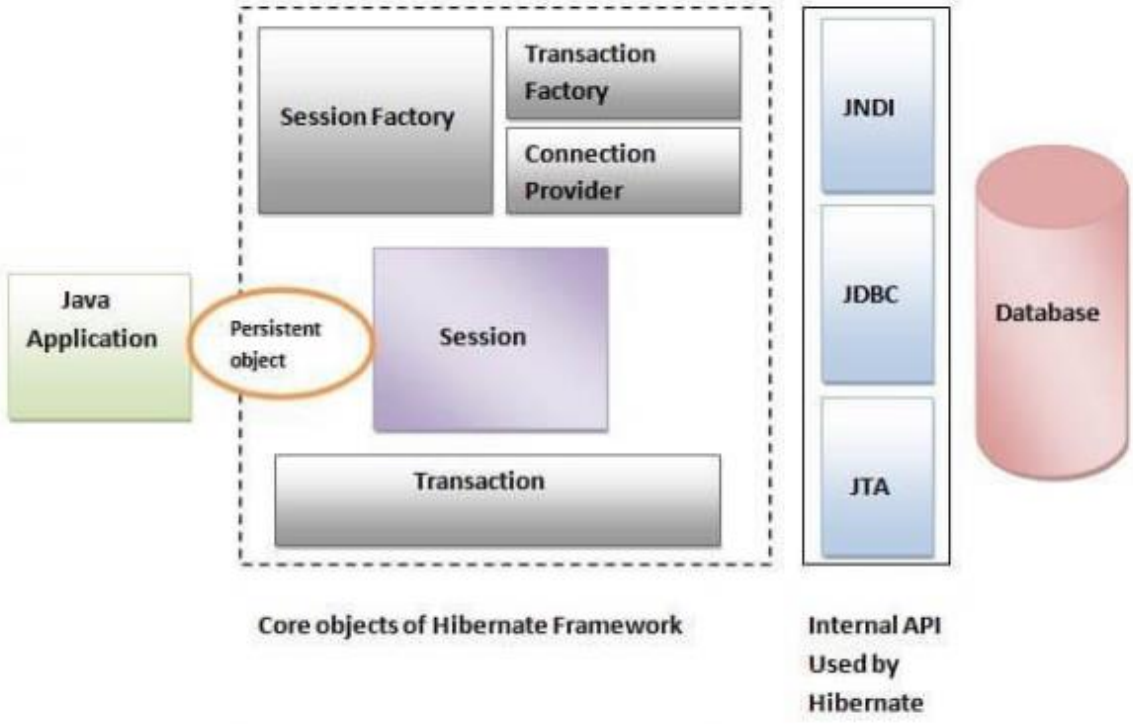
The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.

1) Java application layer
2) Hibernate framework layer
3) Backhand Api layer
4) Database layer

Diagram of hibernate architecture:





Elements of Hibernate Architecture:

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

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.

Session:
The session object provides an interface between the application and data stored in the database. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object.

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

ConnectionProvider:
It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory:
It is a factory of Transaction. It is optional.

5. Explain the advantage of using native SQL queries in Hibernate.

While Hibernate provides a powerful and easy-to-use ORM framework for accessing relational databases, there are some scenarios where native SQL queries can offer advantages over using the Hibernate Query Language (HQL) or Criteria API. Here are some advantages of using native SQL queries in Hibernate:

1) **Performance:**
   Native SQL queries can be optimized for performance and may outperform HQL queries or Criteria queries in some cases. This is because native SQL queries are executed directly by the database, bypassing the Hibernate framework and any overhead that it may introduce.

2) **Complex queries:**
   Native SQL queries can handle complex queries that are difficult or impossible to express using HQL or the Criteria API. For example, queries involving complex joins or subqueries can be expressed more easily and efficiently using native SQL.

3) **Integration with legacy systems:**
   In some cases, legacy systems may require the use of native SQL queries to access the database. By providing support for native SQL queries, Hibernate can integrate more easily with legacy systems and allow them to be modernized more gradually.

4) **Access to database-specific features:**
   Native SQL queries allow access to database-specific features that may not be available through Hibernate or JPA. This can include advanced features such as stored procedures, triggers, or custom data types.

5) **Familiarity with SQL:**
   Many developers are familiar with SQL and may prefer to use native SQL queries to express their queries in a more familiar syntax. This can make the code easier to read and maintain, especially for developers who are new to Hibernate or JPA.

6. **Explain the Transaction Interface in Hibernate**

In hibernate framework, a Transaction interface defines the unit of work. It maintains abstraction from the transaction implementation. A transaction is associated with Session and instantiated by calling **session.beginTransaction().**

The methods of Transaction interface are as follows:

1) **begin():**
   This method begins a new transaction. All subsequent database operation performed within the scope of this transaction will be treated as part of a single unit of work.
   Transaction transaction = session.beginTransaction();

2) **commit():**
   This method commits the transaction, making all changes made during the transaction permanent. If the transaction is not committed, the changes will be rolled back.
   transaction.commit();

3) **rollback():**
   This method rolls back the transaction, undoing any changes made during the transaction. It is typically called in case of an error or when the transaction needs to be aborted.
   transaction.rollback();

4) **isActive():**
   This method checks whether the transaction is currently active. It returns `true` if the transaction is still in progress and `false` if it has been committed or rolled back.

```
if (transaction.isActive()) {
    // Transaction is still active
}
```

Part - C

1. **Explain the Hibernate configuration approaches in detail.**
   Hibernate needs to know about the database configurations to connect to the database. There are three approaches with which we can do the configurations. These approaches are:

   1) **Programmatic Configurations:**
      Hibernate provides a Configuration class which provides certain methods to load the mapping files and database configurations programmatically.
      a) **Loading Mapping Files:**
         - **addResource():**
           We can call addResource() method and pass the path of mapping file available in a classpath. To load multiple mapping files, simply call addResources() method multiple times.

```
Configuration configuration = new Configuration();
configuration.addResource("com/hibernate/tutorial/user.hbm.xml");
configuration.addResource("com/hibernate/tutorial/account.hbm.xml ");
```

         - **addClass():**
           Alternatively we can call addClass() method and pass in the class name which needs to persist. To add multiple classes , we can call addClass() multiple times.

```
Configuration configuration = new Configuration();
configuration.addClass("com.hibernate.tutorial.user.class");
configuration.addClass("com.hibernate.tutorial.user.account.class ");
```

         - **addJar():**
           We can call addJar() to specify the path of jar file containing all the mapping files. This approach provides a generic way and need not to add mapping every time we add a new mapping.

```
Configuration configuration = new Configuration();
configuration.addJar(new File("mapping.jar"))
```

      b) **Loading Database Configurations:**
         - **setProperty():**
           we can call setProperty() method on configuration object and pass the individual property as a key value pair. We can call setProperty() multiple times.

```
<strong><strong><strong><strong>
Configuration configuration = new Configuration();
configuration.setProperty("hibernate.dialect",
"org.hibernate.dialect.MySQLDialect");
configuration.setProperty("hibernate.connection.driver_class",
"com.mysql.jdbc.Driver");
configuration.setProperty("hibernate.connection.url",
"jdbc:mysql://localhost:3306/tutorial");
```

```
configuration.setProperty("hibernate.connection.username", "root");
configuration.setProperty("hibernate.connection.password", "password");
</strong></strong></strong></strong>
```

- **setProperties()**:
  We can call setProperties() method on the configuration object to load the properties from system properties or can pass the property file instance explicitly.

```
Configuration configuration = new Configuration();
configuration.setProperties(System.getProperties());
```

2) **XML configurations:**
We can provide the database details in an XML file. By default, hibernate loads the file with name hibernate.cfg.xml but we can load the file with custom name as well.

The sample XML file looks like below-

```
<xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD//EN" "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">

<hibernate-configuration>
      <session-factory>
            <property name="hibernate.connection.url">
                  jdbc:mysql://localhost:3306/tutorial
            </property>
            <property name="hibernate.connection.username">
                  root
            </property>
            <property name="hibernate.connection.password">
                  password
            </property>
            <property name="dialect">
                  org.hibernate.dialect.MySQLDialect
            </property>
            <property name="show_sql">
                  False
            </property>
            <property name="hibernate.connection.driver_class">
                  com.mysql.jdbc.Driver
            </property>
            <mapping resource="user.hbm.xml"/>
      </session-factory>
</hibernate-configuration>
```

We need to instantiate Configuration object like below:

```
Configuration configuration = new Configuration().configure();
```

new Configuration() call, will load the hibernate.properties file and calling configure() method on configuration object loads hibernate.cfg.xml. In case any

property is defined in both hibernate.properties and hibernate.cfg.xml file then xml file will get precedence.

To load the custom files, we can call

```
Configuration configuration = new Configuration().configure("/configurations
/myConfiguration.cfg.xml")
```

The Above code snippet will load myConfiguration.cfg.xml from the configuration subdirectory of class path.

3) **Properties configurations:**

This approach uses a properties file for the configuration. By default hibernate loads the file with name hibernate.properties but we can load the file with custom name as well. Properties file provides the similar functionality as XML file provides with the exception that, we cannot add a mapping resource in properties file.

Below is the sample content of hibernate.properties file

```
<strong><strong><strong><strong>
hibernate.connection.url = jdbc:mysql://localhost:3306/tutrial
hibernate.connection.username = root
hibernate.connection.password = password
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
</strong></strong></strong></strong>
```

2. **How do create web application using hibernate?**

For creating the web application, we are using JSP for presentation logic, Bean class for representing data and DAO class for database codes.

Example to create web application using hibernate

In this example, we are going to insert the record of the user in the database. It is simply a registration form.

index.jsp

This page gets input from the user and sends it to the register.jsp file using post method.

```
<form action="register.jsp" method="post">
   Name:<input type="text" name="name"/><br><br/>
   Password:<input type="password" name="password"/><br><br/>
   Email ID:<input type="text" name="email"/><br><br/>
   <input type="submit" value="register"/>"
</form>
```

register.jsp

This file gets all request parameters and stores this information into an object of User class. Further, it calls the register method of UserDao class passing the User class object.

```
<%@page import="com.javatpoint.mypack.UserDao"%>
<jsp:useBean id="obj" class="com.javatpoint.mypack.User">
</jsp:useBean>
<jsp:setProperty property="*" name="obj"/>
<%
   int i=UserDao.register(obj);
   if(i>0)
   out.print("You are successfully registered");
%>
```

User.java

It is the simple bean class representing the Persistent class in hibernate.

```
package com.javatpoint.mypack;
public class User {
private int id;
private String name,password,email;
   //getters and setters
}
```

user.hbm.xml

It maps the User class with the table of the database.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
   “http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd”
>
<hibernate-mapping>
   <class name="com.javatpoint.mypack.User" table="u400">
      <id name="id">
         10
         <generator class="increment"></generator>
      </id>
      <property name="name"></property>
      <property name="password"></property>
      <property name="email"></property>
   </class>
</hibernate-mapping>
```

UserDao.java

A Dao class, containing method to store the instance of User class.

```
package com.javatpoint.mypack;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class UserDao {
   public static int register(User u){
   int i=0;

   StandardServiceRegistry ssr = new
   StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
   Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

   SessionFactory factory = meta.getSessionFactoryBuilder().build();
   Session session = factory.openSession();
   Transaction t = session.beginTransaction();

   i=(Integer)session.save(u);
   t.commit();
   session.close();
   return i;
   }
}
```

hibernate.cfg.xml

It is a configuration file, containing information about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    “http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd”
>
<hibernate-configuration>
    <session-factory>
    <property name="hbm2ddl.auto">create</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">jtp</property>
    <property name="connection.driver_class">
        oracle.jdbc.driver.OracleDriver
    </property>
    <mapping resource="user.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

3. Explain the Hibernate Mapping Types

The Java data types are mapped into RDBMS data types when you create a Hibernate mapping document. Java or SQL databases do not use the types stated and used in the mapping files. They are known as Hibernate mapping types and can convert Java data types to SQL data types and vice-versa.

Primary Types of Hibernate Mapping:

There are mainly three different kinds of mapping. As follows:

1) **One to one:**

   One property is mapped to another attribute in this form of relationship in a way that only one-to-one mapping is maintained. An illustration will help you comprehend this

better. Suppose, for instance, that only one employee works for one department. When a worker cannot work for another department at the same time, the mapping is known as one-to-one.

2) **One to many:**
   One property is mapped to many other characteristics in this form of relationship by being mapped to another attribute in a specific way. An illustration will help you comprehend this better. as in the case of a student who belongs to multiple groups, like a simultaneous cultural organization, sports team, and robotics team. The student and group relationship in this scenario is referred to be a many-to-one relationship.

3) **Many to many:**
   One attribute is mapped to another attribute in this type of connection so that any number of attributes can be linked to other attributes without a limit on the quantity. An example will help you better comprehend this. For instance, in the library, one individual may checkout a number of books, and one book may be issued to a number of other books. Many to many partnerships are what this form of relationship is known as. Before implementation, there must be a thorough knowledge of the business use case for this complex relationship.

Types of Mapping:

The primary fundamental forms of Hibernate mapping types are:

1) **Primitive Types:**
   Data types such as "integer," "character," "float," "string," "double," "Boolean," "short," "long," etc., are defined for this type of mapping. These can be found in the hibernate framework to map java data types to RDBMS data types.

2) **Binary and Large Object Types:**
   They include "clob," "blob," "binary," "text," etc. To maintain the data type mapping of big things like images and videos, blob and clob data types are present. Date and Time Types: "Date," "time," "calendar," "timestamp," etc. are some examples. We have data type mappings for dates and times that are similar to primitive.

3) **JDK-related Types:**
   This category includes some mappings for things outside the scope of the previous kind of mappings. "Class," "locale," "currency," and "timezone" are among them.