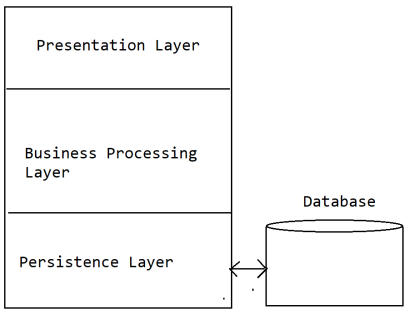
Class Notes Link: https://tinyurl.com/hibernate6pmnotes

Hibernate:
—---------
1. Introduction
2. Steps to prepare First Hibernate Application
3. Primary Key Generation Algorithms
4. Auto Generation Tools[Schema Export and Schema Update]
5. Bulk Updations
   [HQL, Native SQL, Criterion API]
6. Hibernate Filters
7. Hibernate Transaction Management
8. Hibernate Connection pooling Mechanisms
9. Hibernate Mappings
   1. Basic OR Mapping
   2. Component Mapping
   3. Inheritance Mapping
   4. Associations Mapping

—-------
—-------

Introduction:
—-------------
Enterprise Application:
Enterprise application is a Software application, it will be designed for an enterprise in order to simplify their internal business processing.

To prepare Enterprise Applications we have to use the following layers.
1. Presentation Layer
2. Business Processing Layer
3. Data Storage And Access Layer / Persistence Layer



Presentation Layer
—----------

The main purpose of the presentation layer is

1. It is an entry point for the users in order to access the application.
2. To improve look and feel to the enterprise application.
3. It will get the data from the users in order to submit data to the Server side application.
4. To provide different types of the requests like GET, POST, HEAD,... from client to the server side application we have to use a presentation layer.
5. To define the client side data validations by using Java script functions we need the presentation part.
6. To prepare the presentation layer in enterprise applications we have a separate logic that is Presentation Logic.
7. To prepare presentation logic in enterprise applications we have to use web technologies like Html, CSS, Java Script, Bootstrap,.....

Business Layer:
—----------

The main purpose of this layer is to define and execute all the business rules and regulations which are required by the clients.

To prepare the Business layer in the enterprise applications we have to use a separate logic called Business Logic.

To prepare Business Logics we will use the components like Servlets, Daos, Java Beans, EJB Session Beans,.....

Persistence Layer:
—----------

The main purpose of the Persistence layer is to interact with the database in order to perform the database operations from the enterprise applications.

To prepare the Persistence layer in the enterprise applications we will use a separate logic called “Persistence Logic”.

To prepare Persistence logic in the enterprise applications we have to use a set of technologies and tools like JDBC, EJBs Entity Beans, JPA, Open JPA,...

Data Persistence:
—----------------
Representing data permanently in the Databases is called Data Persistence.

To achieve data persistence we will perform a set of Operations called Persistence Operations.

In Data Persistence technologies we will perform the data persistence operations in the form of CRUD operations.

C —-----> Create / insert
R —-----> Read / retrieve
U —-----> Update
D —-----> Delete

In the enterprise applications , to perform the Persistence operations we will use a set of technologies called Persistence technologies.

To perform Data persistence w.r.t the java applications we will use the following persistence mechanisms.

1. Serialization and Deserialization
2. JDBC
3. ORM
    a. EJBs Entity Beans
    b. Hibernate
    c. JPA
        —----
        —----

Data Persistence through Serialization and Deserialization:
—---
Serialization: The process of separating data from an object is called Serialization.

Deserialization: The process of reconstructing an object on the basis of the data is called Deserialization.

In general, in Distributed applications it is a frequent requirement to send an object from One machine to another machine, where to send an object from one to machine another machine we have to Separate the data from an object at the sender machine , We have to send the separated data to the another

machine through the network, at the receiver machine we have to reconstruct an object on the basis of the data.

In Distributed applications, we will serialize an object and we will send the serialized data to the Remote machine.

In Standalone applications, if we want to perform Serialization over an object then we have to perform Serialization and we have to send the serialized data to a flat file.

To Perform Serialization and Deserialization JAVA has provided the following byte oriented streams.

ObjectOutputStream —----> Serialization
ObjectInputStream —-----> Deserialization

Steps to perform Serialization:
—-------------------------------

1. Create a Serializable class and its Object:
   In Java, by default all objects are not eligible for Serialization and deserialization, only the objects whose classes are implementing Serializabl marker interface are eligible for Serialization.

   ```
   class Employee implements Serializable{
         —------
   }
   Employee emp1 = new Employee();
   ```

2. Create FileOutputSTream with the target file.
   ```
   FileOutputStream fos = new FileOutputStream(“emp.txt”);
   ```

3. Create ObjectOutputStream with the FIleOutputStream:
   ```
   ObjectOutputStream oos = new ObjectOutputStream(fos);
   ```

4. Write a Serializable Object to the ObjectOutputStream.
   ```
   oos.writeObject(emp1);
   ```

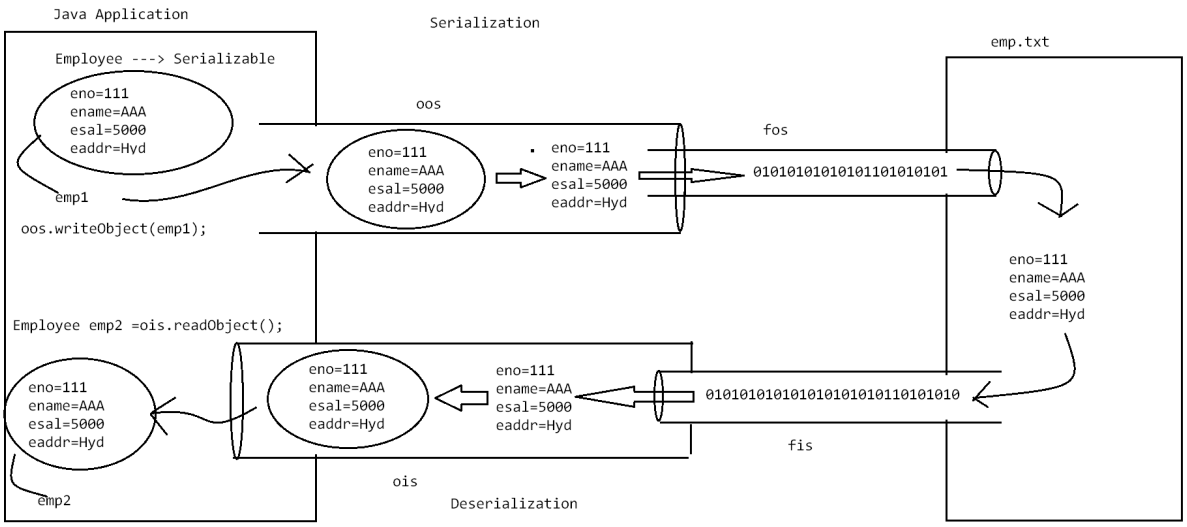
With the above Steps, Serialization will be performed and the Serialized data will be sent to the emp.txt file.

Steps to Perform Deserialization:
—---------------------------------

1. Create FileInputStream:
   FileInputStream fis = new FileInputStream(“emp.txt”);

2. Create ObjectInputStream with the FileInputStream:
   ObjectInputStream ois = new ObjectInputStream(fis);

3. Read Deserialized Object from the ObjectINputStream:
   Employee emp2 = (Employee)ois.readObject();



In the above Data Persistence mechanism we are able to achieve Data Persistence by storing the data[Serialized Data] in a flat file, where the flat file is a permanent storage, it stores the data permanently.

Serialization Deserialization Persistence mechanism is able to provide the following drawbacks:

1. Serialization And Deserialization Data persistence mechanism is highly recommended for the Standalone Applications, not for the enterprise Applications.
2. Serialization and Deserialization Data Persistence mechanism is supporting only byte oriented data , not other forms of the data.
3. Serialization and Deserialization Data persistence mechanism is providing only Create and Read persistence operations, not supporting Update and Delete operations.

4. Serialization and Deserialization Data persistence mechanism takes a lot of java code to perform the simple database operations like Create and Read.

To overcome all the above problems we have to use JDBC Persistence Mechanism.

Data Persistence through JDBC:
—----------------------------
JDBC: Java Database Connectivity

It is a step by step process, an API, an abstraction, a technology to connect with the database in order to perform database operations from a java application.

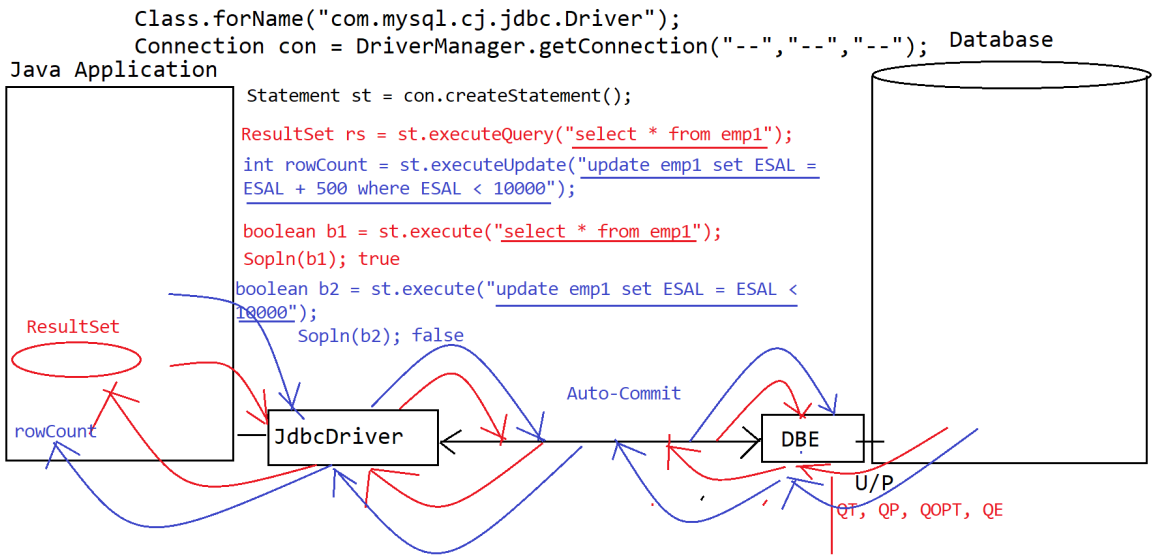
To prepare JDBC applications we have to use the following steps.

1. Load and Register Driver.
   ```
   Class cls = Class.forName("com.mysql.cj.jdbc.Driver");
   ```

2. Establish the connection between java application and database.
   ```
   Connection con = DriverManager.getConnection(
         "jdbc:mysql://localhost:3306/durgadb",
         "root",
         "root"
         );
   ```
3. Create either Statement or PreparedStatement or CallableStatement objects as per the requirement.
   ```
   Statement st = con.createStatement();
   ```

4. Write and Execute SQL queries:
   ```
   ResultSet rs = st.executeQuery("select * from emp1");

   int rowCount = st.executeUpdate("update emp1 set ESAL = ESAL + 500
   where ESAL < 10000");

   boolean b1 = st.execute("select * from emp1");
   boolean b2 = st.execute("delete from emp1 where ESAL < 10000");
   ```

5. Close the resources:
   ```
   con.close();
   ```



In Jdbc applications, when we establish the connection between a Java application and the database, automatically COnnection will have a default nature “Auto-Commit”.

As per the Connection’s Auto-Commit nature , when we submit an sql query to the Connection, Connection will carry the provided sql query to the Database Engine and the COnnection will make the database engine to execute the provided sql query and to store the results of the provided sql query in the Database, in this context, data will be represented in the database permanently, it is called Data persistence.

In JDBC data Persistence mechanism we are able to get the following problems.

1. In JDBC Data persistence mechanism, we have to provide lot of boilerplate code , that is the repeatable code like

   Load and Register Driver
   Establish COnnection between Java application and database.
   Create Statement object
   Close the resources.

2. In the JDBC Data Persistence mechanism, we must write sql queries explicitly , so developers must have knowledge on the SQL queries.

3. In JDBC Data Persistence mechanism we will hardcode the sql queries.

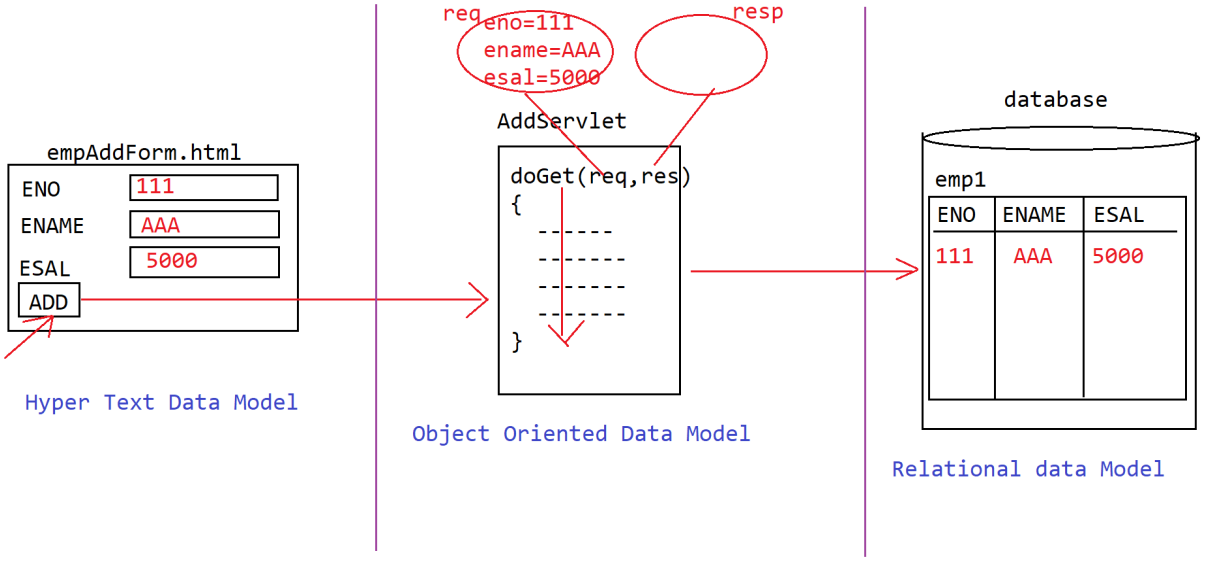
4. In JDBC Data persistence mechanism, we must have the focus on the Driver management.
5. In JDBC Data persistence mechanism, we must have the focus on Connection management.
6. JDBC Data Persistence mechanism is a database dependent mechanism.
7. JDBC data persistence mechanism is able to provide limited support for the transactions.

To overcome these problems we have to use an alternative that is ORM.

ORM:
—---
ORM: Object Relational Mapping

In general, in enterprise applications, we will represent the data in the following data models.

1. HyperText Data Model
2. Object Oriented Data Model
3. Relational data Model



IN the enterprise applications , developers are not required to concentrate on the conversion from the hypertext data model to the Object Oriented data model, because it will be performed by the Servers and containers internally.

In the enterprise applications, developers must concentrate on the conversions from the Object oriented data model to the Relational data model.

In general, the data representation is different in both the data models and which are having the different conventions to represent data , these different conventions are able to create mismatches between both the data models, here the mismatches between both the data models will reduce data persistence in the enterprise applications.

In the enterprise applications, to improve data persistence we have to avoid the mismatches between the Data Models, here to avoid the mismatches between the data models we have to use ORM tools.

ORM: ORM is a design approach, it is able to provide the mapping between the object oriented data model elements and the Relational Data model elements either through XML file or through annotations.

Object Oriented Data Model		Relational Data Model Elements
1. Class	——————————>	Table
2. Object	——————————>	Record
3. ID Property	——————————>	Primary Key Column
4. Property	——————————>	Column

In general, in the enterprise applications, we will get the following mismatches between the Object oriented data model and the Relational data Model.

1. Granularity Mismatch.
2. Subtypes mismatch.
3. Associations Mismatch.
4. Identity Mismatch.

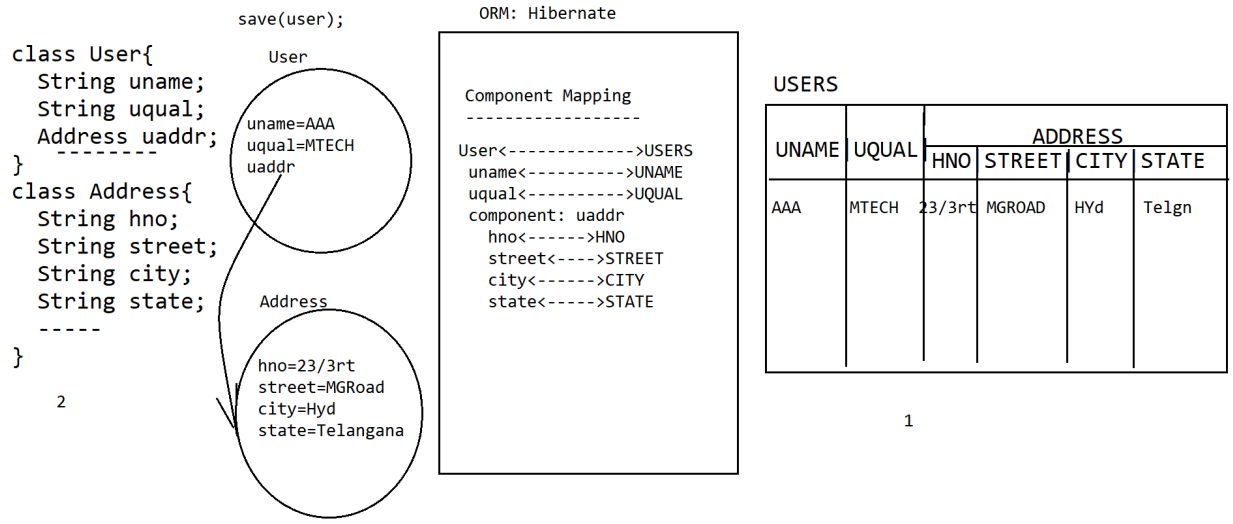
Granularity Mismatch:

The number of classes which we have used in the Object oriented data model is called Granularity.

The number of tables which we have used in the Relational data model is called Granularity.

IN an enterprise application, as per the requirement we may use the number of classes to represent data in Object Oriented data model and we may use the number tables in the relational data model, In the object oriented data model the granularity and the Relational Data Model granularity may not be matched, in this situation we may get less data persistence in the enterprise applications.

In the above context, to improve data persistence we have to resolve granularity mismatch between both the data models, for this Hibernate as an ORM implementation has provided a solution in the form of “Component Mapping”.



Subtypes Mismatch:

—-------------------

In enterprise applications, we will define inheritance relations between entity classes in the Object Oriented Data model in order to improve code reusability.

In the relational data model, we will define the relations between table in multiple ways like PK-FK, or a single table for all columns,....

In the Object oriented data model when we save subclass objects , automatically subclass objects data must be stored in the table in the relational data model.

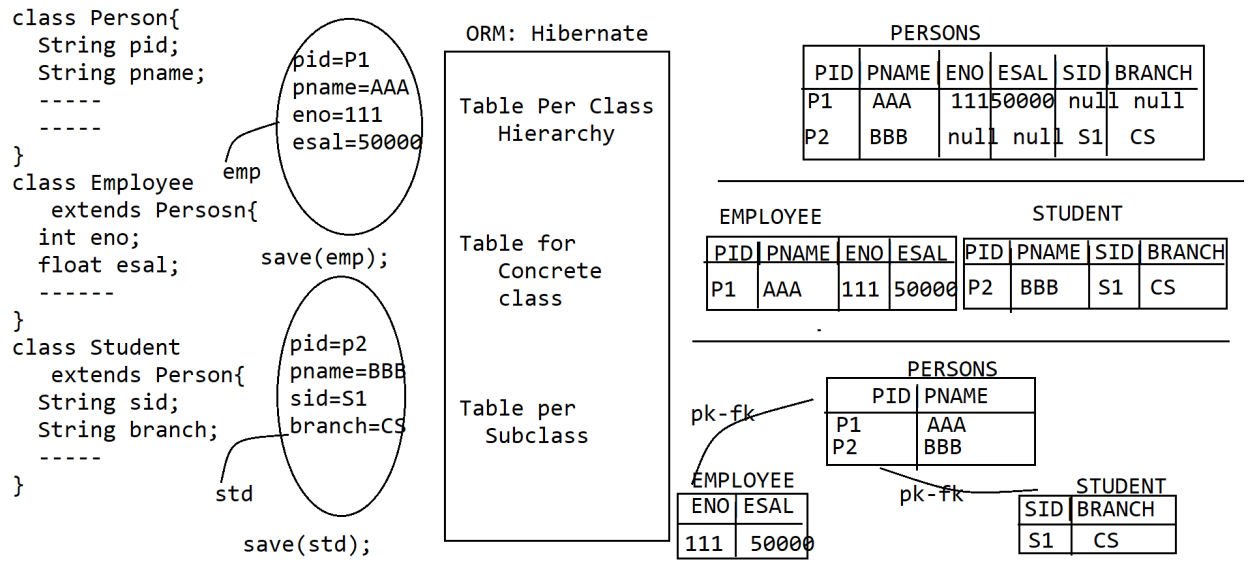
In the above context, both the data models are having their own approaches of representing data , it is the subtypes mismatches between Object oriented

data model and the relation data models, it will reduce data persistence in the enterprise applications.

In the above context, to improve data persistence we have to use ORM implemented tools like Hibernate.

To resolve subtypes mismatch between Object oriented data model and relational data model Hibernate has provided the following solutions.

1. Table Per class hierarchy
2. Table per concrete class
3. Table per subclass



Associations Mismatch:

—----------

In the Object orientation data model, we are able to use associations to provide communication between entities and to improve data navigation between entities.

In the Object Oriented Data model, we are able to provide associations by defining one class reference variable in another class.

```
class Employee{
      Account account;
}
class Account{
}
```

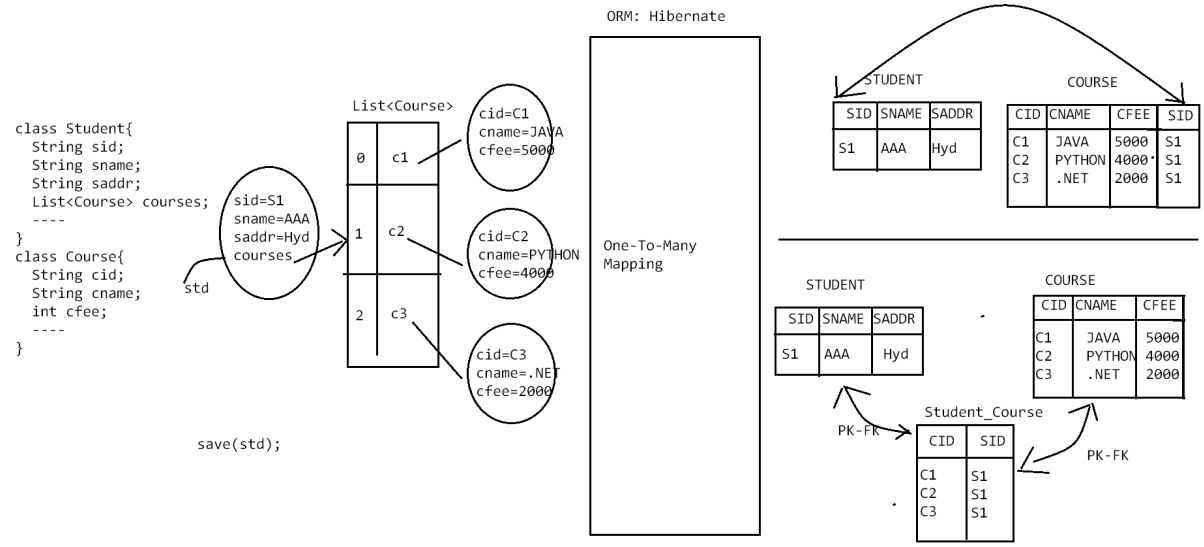
In the Relational Data Model , to achieve the associations we may use the number of alternatives like by using PK-FK relationships or by providing a Join column or by using a Join table.

In enterprise applications, we are able to use both Object oriented data model and Relational data model , but both are having their own approaches to achieve associations, it will create associations mismatch, it will reduce data persistence in the enterprise applications.

In the above context, to improve data persistence we have to use ORM implementations like Hibernate.

Hibernate has provided the following solutions for the association mismatches.

One-To-One Mapping
One-To-Many Mapping
Many-To-One Mapping
Many-To-Many Mapping



Identity Mismatch:
—--------------------
IN the Object oriented Data Model, to check whether two objects are equals or not we will use either == operator or equals() method.

In the relational data model, there is no specific approach to check whether two records are equal or not.

In the above context, there is a mismatch about the equality check, it will reduce data persistence.

IN the above context, ORM is able to provide mapping between both data models in order to check the equality between two objects.

ORM is an approach or a set of rules and regulations which are implemented by the number of tools.

1. EJB’s Entity Beans
2. Hibernate
3. JPA
4. Open JPA
   —--
   —---

Q)What are the differences between Entity Beans and Hibernate?
—---
Ans:
—----
1. EJB’s Entity beans is more API dependent , where the entity bean class must extend or implement predefined libraries which are provided by EJBs.

   Hibernate is less API dependent , because hibernate will use POJO classes as bean components, where POJO class is a Java bean class it will not extend or implement any predefined library.

2. As EJBs Entity Beans are more API dependent it is very difficult to perform debugging and Testing.

   As Hibernate is Less API dependent, it is very simple to perform debugging and testing.

3. IN case of EJBs entity beans, it is not possible to extend one bean component to another bean component, because bean components are already extended from some predefined library.

   In case of Hibernate , it is possible to extend one bean component to another bean component , because in Hibernate all bean components are POJO classes.

4. EJBs Entity beans require an application server to execute.

   Hibernate applications are executed with or without the application server.

5. EJBs entity beans are heavy weight.

   Hibernate is lightweight.

6. EJbs entity beans are less portable.

   Hibernate is more portable.

7. EJBs entity beans are slower data persistence mechanism

   Hibernate is relatively faster persistence mechanism

Q)What is the difference between JPA and Hibernate?
—----------

JPA [Java Persistence API] is an abstraction provided by SUN Microsystems and implemented by almost all the application servers like Weblogic, Wildfly, Glassfish,... and JPA defines a set of rules and regulations or a set of conventions to implement ORM rules and regulations.

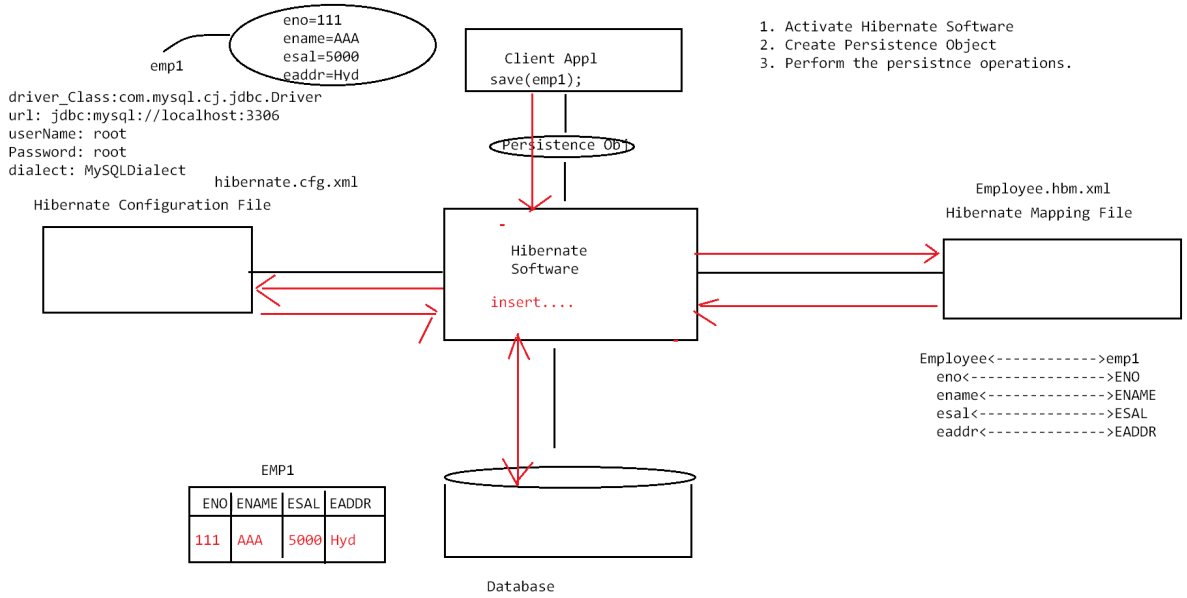
Hibernate is a tool, which implements ORM rules and regulations as per the JPA guidelines in order to provide data persistence in the enterprise applications.

Hibernate History:
—-------------------
Author: Gavin King
Objective: To simplify data persistence in enterprise applications.
Type: ORM Product
OPEN / Licenced: Open Source Software
Initial Version: Hibernate 1.x
Latest Version: Hibernate 6.x
Designed on : Java
Website : www.hibernate.org

Hibernate Features:
—-------------------

1. Hibernate is Database independent, it will be used for any type of database.
2. Hibernate is suitable for all the types of applications like web applications, distributed applications, standalone applications…..
3. Hibernate is able to provide very good support for associations and joins.
4. Hibernate has annotation support to reduce XML dependency.
5. Hibernate has its own implementations for primary key generation algorithms.
6. Hibernate has a very good Collection support while performing database operations.
7. Hibernate has its own query language in the form of HQL.
8. Hibernate has its own implementations for the Cache mechanisms to hold the results in order to reuse the same results in the applications.
9. Hibernate has a very good Connection pooling mechanism.
10. Hibernate is supported by almost all the servers and IDEs
11. In Hibernate applications, it is not required to write sql queries.
12. Hibernate is providing very good Transactions support.

Hibernate Architecture:
—----------------------



Where the Hibernate Configuration file will have the hibernate configuration details like Driver class name, Driver URL, database user name, database password, database dialect information,.... Which are required to establish connection with the database.

Where the Hibernate mapping file will provide the mapping details between the POJO class , POJO class properties with the database table name and the database column names,....

Where the main purpose of the CLient application is

1. Activate the Hibernate software.
2. Create Persistence objects
3. Perform the persistence operations

When we activate Hibernate software , Hibernate Software will read all the hibernate configuration details from the configuration file and Hibernate Software will create connections with the database and Hibernate software will make ready the mapping file.

After creating the Persistence Object and if we access any persistence methods then the Hibernate Software will perform the following actions.

1. Hibernate Software will recognize the persistence method and the provided persistence object.
2. Hibernate Software will goto the mapping file and get all the database related data on the basis of the Persistence object properties.
3. Hibernate Software will create the database dependent sql queries.
4. Hibernate software will execute the generated sql queries.
5. Hibernate software will send the return values to the CLient application.

Steps to Prepare Hibernate Applications:
—---

1. Download and Install Hibernate software.
2. Create a Java Project in Eclipse IDE and attach all the hibernate dependent jar files in the build path.
3. Create a POJO class as per the requirement.
4. Create a Hibernate Mapping file.
5. Create Hibernate configuration File
6. Create a Test Application.

Download and Install Hibernate software.
—---
Download the Hibernate software from the following link.
https://sourceforge.net/projects/hibernate/files/hibernate3/3.6.10.Final/

If we download Hibernate software then we will get “hibernate-distribution-3.6.10.Final-dist.zip”, extract zip file then we will get the “hibernate-distribution-3.6.10.Final” folder , it contains all the required Hibernate jar files.

Create a Java Project in Eclipse IDE and attach all the hibernate dependent jar files in the build path:
—---
After creating a java project in the eclipse IDE, provide the following jar files inside the project build path.

1. hibernate3.jar
2. antlr-2.7.6.jar
3. commons-collections-3.1.jar
4. dom4j-1.6.1.jar
5. javassist-3.12.0.GA.jar
6. jta-1.1.jar
7. slf4j-api-1.6.1.jar

Apart from all the above jar files we must add a driver jar file in the project build path.

ojdbc11.jar
mysql-connector-j-8.0.33.jar

Create a POJO class as per the requirement:
—--
POJO class is a Java bean class, it must not extend or implement a predefined library.

The main purpose of the POJO classes is to prepare Persistence objects.

To prepare POJO classes we have to use the following rules and regulations.

1. Prepare POJO classes as per the Database tables and their columns.In this case , POJO class name and its properties need not to be matched with the table name and its column names, but we must maintain the compatible data types.

   number —-----> int, long, ….
   varchar2 —---> String
   float—-------> float

2. In POJO classes, we must provide a set of setXXX() and getXXX() methods for each and every property.
3. In POJO class, declare all properties as private and declare all methods as public.
4. Declare all the POJO classes as public, non abstract and non final.

   Where the main purpose of the public in the POJO class declaration is to bring POJO class scope to the Hibernate software in order to create objects to the POJO class.

   Where the main purpose of declaring POJO class as a non abstract class is to allow the creation of objects for the POJO class.

   Where the main purpose of declaring POJO class as a non final class is to allow inheritance between POJO classes in order to improve code reusability.

5. If we want to compare two POJO class objects with our own comparison mechanisms then it is suggestible to override equals() method.

6. If we want to provide our own mechanisms to generate Hashcode values to the objects in order to arrange the elements in the collections like HashSet then it is suggestible to override hashcode() method in the POJO classes.

EX:

```
public class Employee implements java.io.Serializable{
      private int eno;
      private String ename;
      private float esal;
      private String eaddr;
      setXXX() and getXXX();
}
```

Create a Hibernate Mapping file:

—--------------------------------

The main purpose of the Hibernate Mapping file is to provide the mapping configurations between the following elements from the Object oriented data model and the Relational Data model.

```
Class  —----------------->    Table
Id Property —------------>    PK column
property —---------------->    Column
—---
—---
```

In Hibernate applications, the Mapping file is responsible for providing the following configuration details.

1. Nasic OR Mapping
2. Component Mapping
3. INheritance Mapping
4. Associations Mapping
   —----
   —----

IN Hibernate mapping file , to provide mapping between a POJO class and a database table we have to use the following xml tags.

```
<!DOCTYPE ….>
```

```
<hibernate-mapping>
      <class name=”--” table=”--”>
            <id name=”--” column=”--”/>
            <property name=”--” column=”--”/>
            —----
            —----
      </class>
</hibernate-mapping>
```

Where <class> tag is able to provide POJO class configuration.
Where the “name” attribute in <class> tag will take the “Fully Qualified name of the POJO class”.
Where the “table” attribute in <class> will take the database table name which we want to map with the POJO class.

Where <id> tag is able to provide the mapping between the id property of the POJO class and the primary key column in the table.

Where the “name” attribute will take the id property name.
Where the “column” attribute will take the Primary key column in the database table.
Note: if the POJO class provided id property name and the database table provided primary key column name are same then it is optional to provide column attribute.

Where <property> tag will provide the mapping between normal property from the POJO class and the normal column from the database table.

Where “name” attribute will take POJO class property name.
Where “column” attribute will take the database table column name.
Note: if the POJO class provided property name and the database table provided column name are same then it is optional to provide column attribute.

In general, the name of the mapping file is
POJOClassName.hbm.xml
EX: Employee.hbm.xml

EX:

```
<!DOCTYPE …. >
<hibernate-mapping>
      <class name=”com.durgasoft.beans.Employee” table=”emp1”>
            <id name=”eno” column=”eno”/>
            <property name=”ename” column=”ename”/>
            <property name=”esal” column=”esal”/>
            <property name=”eaddr” column=”eaddr”/>
      </class>
</hibernate-mapping>
```

If we use the database table emp1 like below then it is optional to provide column attribute

emp1

ENO|ENAME|ESAL|EADDR

EX:

```
<!DOCTYPE …. >
<hibernate-mapping>
      <class name=”com.durgasoft.beans.Employee” table=”emp1”>
            <id name=”eno” />
            <property name=”ename” />
            <property name=”esal” />
            <property name=”eaddr” />
      </class>
</hibernate-mapping>
```

Hibernate Configuration File:

—----------

The main purpose of the Hibernate configuration file is to provide all the configuration details of the hibernate software that we need to establish communication with the databases.

IN Hibernate applications, the Configuration file is responsible to provide the following configuration details.

1. Connection COnfigurations
2. Transactions configurations
3. Cache mechanisms configurations
4. Connection pooling Configurations
   —----
   —----

In Hibernate applications, to prepare configuration file we have to use the following XML tags

```
<!DOCTYPE...>
<hibernate-configuration>
      <sessionfactory>
            <property name=”---”>Value</property>
            —----
            <mapping resource=”--”/>
      </sessionfactory>
</hibernate-configuration>
```

Where <hibernate-configuration> tag is a root tag, it will represent hibernate configuration details.

Where <sessionfactory> tag is able to provide all the hibernate application required database configurations, connection pooling configurations,....

Where <property> tag will provide a single hibernate property.

Where the “name” attribute in the <property> tag will provide property name. In the hibernate configuration file we have to provide value to the property in the body of the <property> tag.

Where <mapping> tag is able to provide hibernate mapping file configuration. Where “resource” attribute in the mapping file will take the name and location of the hibernate mapping file.

EX:

```
<hibernate-configuration>
  <sessionfactory>
      <property name=”hibernate.connection.driver_class”>
            com.mysql.cj.jdbc.Driver
      </property>
      <property name=”hibernate.connection.url”>
            jdbc:mysql://localhost:3306/durgadb
      </property>
      <property name=”hibernate.connection.username”>root</property>
      <property name=”hibernate.connection.password”>root</property>
      <property name=”hibernate.dialect”>
            org.hibernate.dialect.MySQLDialect
      </property>
      <mapping resource=”/Employee.hbm.xml”/>
  </sessionfactory>
</hibernate-configuration>
```

Hibernate Test Application / Hibernate Client Application:
—--
The main purpose of the Hibernate Test Applications is

1. Activate Hibernate Software.
2. Create Persistence Objects.
3. Perform the persistence operations.

To prepare the Hibernate Test application we have to use the following steps.

1. Create Configuration class object:
   The main purpose of the Configuration class object is to encapsulate all the hibernate configuration details which we provided in the hibernate configuration file.

   In Hibernate applications, to represent Configuration objects Hibernate has provided a predefined class in the form of "org.hibernate.cfg.Configuration".

   EX:
   Configuration cfg = new Configuration();

   If we execute the above instructions, Hibernate SOftware will create a Configuration object in the heap memory without having configuration details.

   If we want to keep all the hibernate configuration details which we provided in the hibernate configuration file then we have to use the following methods.

   public Configuration configure()
   Public Configuration configure(String fileName)
   Public Configuration configure(URL url)
   Public Configuration configure(InputStream is)
   —---
   —---
   The configure() method will search for the configuration file with default name hibernate.cfg.xml and the Hibernate software will get all the details of the configuration file and it will keep all the configuration details inside the Configuration object.

Where configure(String fileName) method will take the user defined or custom configuration file name and location as an input and it will get the configuration details from the custom configuration file and encapsulate the configuration details in the Configuration object.

The configure(URL url) method is able to take the location URL of the configuration in the internet and it is able to get all the configuration details through the provided url into the COnfiguration object.

The configure(InputStream is) method is able to get all the configuration details from the provided InputStream object into the Configuration object.

In Hibernate applications, we will create a separate configuration object for each and every configuration file, in Hibernate applications we will create a separate configuration file for each and every database, so Configuration object is a thread safe object up to a particular database.

Configuration object is a heavy weight object in the Hibernate applications.

2. Create SessionFactory object:
   The main purpose of the Sessionfactory in the hibernate applications is to perform the following actions.
   a. Load the driver class bytecode to the memory.
   b. Creating Connection objects.
   c. Maintaining Connection pooling internally.
   d. Creating Statement or PreparedStatement or callableStatement object internally.
      –-----
      –------

   To represent SessionFactory object in the Hibernate applications, Hibernate API has provided a predefined interface in the form of org.hibernate.SessionFactory.

   To create SessionFactory object in the Hibernate applications we have to use the following method.

   public SessionFactory buildSessionFactory()
   EX: SessionFactory sf = cfg.buildSessionFactory();

In Hibernate Applications, we will create a Separate SessionFactory object for each and every Database.

In Hibernate Applications, SessionFactory object is a Thread Safe up to a single database.
In Hibernate applications, SessionFactory objects are heavy weight Objects.

3. Create Session object:

In Hibernate applications, After getting the SessionFactory object we have to perform the persistence operations, to perform the persistence operations we have to use a predefined library that must be provided by the Hibernate API.

In Hibernate Applications, Hibernate has provided the required predefined library in the form of org.hibernate.Session interface to perform the persistence operations.

To get Session objects in Hibernate applications we have to use the following method from SessionFactory.

```
public Session  openSession()
```

EX:

```
Session session = sessionFactory.openession();
```

In Hibernate , Session object is lightweight and it is not thread safe.

4. Perform the Persistence Operations:

In Hibernate , to achieve data persistence we will perform a set of operations called Persistence operations.

In general, in Database applications CRUD operations are treated as the Persistence operations.

To perform the persistence operations Session has provided the Following methods.

a. To insert a Persistence Object in the Database:
    1. save()
    2. persist()

b. To retrieve a Persistence object from the Database:
   1. get()
   2. load()

c. To perform Updations in the database:
   1. update()
   2. saveOrUpdate()

d. To delete a Persistence object from the Database:
   1. delete()

6. Create Transaction object as per the requirement:

In the Hibernate applications, if we perform non select operations with the persistence methods like save(), persist(), update(), saveOrUpdate(), delete(),.. We must use a Transaction object.

To get Transaction object we have to use the following method from the Session.

public Transaction beginTransaction()
It will create a Transaction object and it will begin the Transaction.
EX: Transaction tx = session.beginTransaction();

public Transaction getTransaction()
It will create a Transaction object, but we must begin the transaction explicitly by using the begin() method.
EX: Transaction tx = session.getTransaction();
tx.begin();

After performing the non select operations in the Hibernate application we must perform either commit operation or rollback operation.

public void commit()
public void rollback()

7. Close the resources:

To avoid security problems it is suggestible to close the resources after performing the persistence operations.

```
session.close();
sessionFactory.close();
```

EX-1:

antlr-2.7.6.jar
commons-collections-3.1.jar
dom4j-1.6.1.jar
hibernate3.jar
hibernate-jpa-2.0-api-1.0.1.Final.jar
javassist-3.12.0.GA.jar
required\jta-1.1.jar
ojdbc11.jar
Slf4j-api-1.6.1.jar

Employee.java

```
package com.durgasoft.entities;
public class Employee {

	private int eno;
	private String ename;
	private float esal;
	private String eaddr;

	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public float getEsal() {
		return esal;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
```

```
    }


}
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
     <class name="com.durgasoft.entities.Employee" table="emp1">
          <id name="eno" column="ENO"/>
          <property name="ename" column="ENAME"/>
          <property name="esal" column="ESAL"/>
          <property name="eaddr" column="EADDR"/>
     </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
     "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
     "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
     <session-factory>
          <property
name="hibernate.connection.driver_Class">oracle.jdbc.OracleDriver</pro
perty>
          <property
name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
          <property
name="hibernate.connection.username">system</property>
          <property
name="hibernate.connection.password">durga</property>
```

```
            <property
name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property
>
            <mapping resource="Employee.hbm.xml"/>
      </session-factory>
</hibernate-configuration>
```

Test.java

```
package com.durgasoft.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.durgasoft.entities.Employee;
public class Test {
      public static void main(String[] args) throws Exception {

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

            SessionFactory sessionFactory =
configuration.buildSessionFactory();

            Session session = sessionFactory.openSession();

            Employee employee = new Employee();
            employee.setEno(111);
            employee.setEname("Durga");
            employee.setEsal(5000);
            employee.setEaddr("Hyd");

            Transaction transaction = session.beginTransaction();
            session.save(employee);
            transaction.commit();
            System.out.println("Employee "+employee.getEno()+" Inserted
Successfully");

            session.close();
            sessionFactory.close();
```

```
	}
}
```

```
Employee 111 Inserted Successfully
```

Get the DOCTYPE definitions from the following path.

1. Open Referenced Libraries.
2. Open hibernate3.jar file.
3. Open org.hibernate package.
4. Open hibernate-configuration-3.0.dtd and hibernate-mapping-3.0.dtd files
5. Copy the DOCTYPE definition and keep them into the mapping files and configuration files.

Hibernate Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

Hibernate Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

EX:

Employee.java

```
package com.durgasoft.entities;
public class Employee {

	private int eno;
	private String ename;
	private float esal;
	private String eaddr;

	public int getEno() {
```

```
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public float getEsal() {
		return esal;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}



}
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
	<class name="com.durgasoft.entities.Employee" table="emp1">
		<id name="eno"/>
		<property name="ename"/>
		<property name="esal"/>
		<property name="eaddr"/>
```

```
	</class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
	<session-factory>
		<property
name="connection.driver_Class">oracle.jdbc.OracleDriver</property>
		<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
		<property name="connection.username">system</property>
		<property name="connection.password">durga</property>
		<property
name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property
>
		<mapping resource="Employee.hbm.xml"/>
	</session-factory>
</hibernate-configuration>
```

Test.java

```
package com.durgasoft.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.durgasoft.entities.Employee;
public class Test {
	public static void main(String[] args) {

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

		SessionFactory sessionFactory =
configuration.buildSessionFactory();
```

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

            Employee employee = new Employee();
            employee.setEno(222);
            employee.setEname("Anil");
            employee.setEsal(6000);
            employee.setEaddr("Chenni");

            Transaction transaction = session.beginTransaction();
            session.persist(employee);
            transaction.commit();
            System.out.println("Employee "+employee.getEno()+" Inserted
Successfully");

            session.close();
            sessionFactory.close();
      }
}
```

Q)What is the difference between save() method and persist() method?

Ans:

---

save() method can be used to insert an object as a record in the Database table, but it returns the inserted record's primary key value.

```
public Serializable save(Object obj)
EX:
Transaction transaction = session.beginTransaction();
int pk_Val = (Integer) session.save(employee);
transaction.commit();
if(pk_Val == employee.getEno()) {
      System.out.println("Employee "+employee.getEno()+" Inserted
Successfully");
}else {
      System.out.println("Employee Insertion Failure");
}
```

persist() method will insert an object as a record into the database table, but it does not return primary key value.

EX:

```
Transaction transaction = session.beginTransaction();
session.persist(employee);
transaction.commit();
System.out.println("Employee Inserted Successfully");
```

EX:

Employee.java

```
package com.durgasoft.entities;
public class Employee {

	private int eno;
	private String ename;
	private float esal;
	private String eaddr;

	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public float getEsal() {
		return esal;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}
```

```
}
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
      <class name="com.durgasoft.entities.Employee" table="emp1">
            <id name="eno"/>
            <property name="ename"/>
            <property name="esal"/>
            <property name="eaddr"/>
      </class>
</hibernate-mapping>
```

myconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
            <property
name="connection.driver_class">oracle.jdbc.OracleDriver</property>
            <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
            <property name="connection.username">system</property>
            <property name="connection.password">durga</property>
            <property
name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property
>
            <mapping resource="Employee.hbm.xml"/>
      </session-factory>
</hibernate-configuration>
```

Test.java

```
package com.durgasoft.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import com.durgasoft.entities.Employee;
public class Test {
      public static void main(String[] args) {
            Configuration configuration = new Configuration();
            configuration.configure("myconfig.xml");
            SessionFactory sessionFactory =
configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();
            Employee employee = (Employee)
session.get("com.durgasoft.entities.Employee", 111);
            if(employee == null) {
                  System.out.println("Employee Does not exist");
            }else {
                  System.out.println("Employee Details");
                  System.out.println("----------------------");
                  System.out.println("Employee Number     :
"+employee.getEno());
                  System.out.println("Employee Name       :
"+employee.getEname());
                  System.out.println("Employee Salary     :
"+employee.getEsal());
                  System.out.println("Employee Address    :
"+employee.getEaddr());
            }
            session.close();
            sessionFactory.close();
      }
}
```

```
Employee Details
----------------------
Employee Number     : 111
Employee Name       : Durga
Employee Salary     : 5000.0
Employee Address    : Hyd
```

If we want to work with the MySQL database we have to use the following steps.

1. Add MySQL connector jar file.
2. Provide MySQL Configurations.
   diverClassName: com.mysql.cj.jdbc.Driver
   url : jdbc:mysql://localhost:3306/durgadb
   Username: root
   password : root
   dialect: org.hibernate.dialect.MySQLDialect

Q)What is the difference between get() and load() method in Hibernate?
—----------

Ans:
—---

1. get() method can be used to get an object from the Database table, if the record is not available then get() method will return null value instead of raising an exception.

load() method can be used to get an object from the Database table, if the record is not available then the load() method will raise an exception like org.hibernate.ObjectNotFoundException

EX:
Employee.java

```
package com.durgasoft.entities;
public class Employee {
	private int eno;
	private String ename;
	private float esal;
	private String eaddr;
	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
```

```
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public float getEsal() {
		return esal;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}
	@Override
	public String toString() {
		return "Employee [eno=" + eno + ", ename=" + ename + ",
esal=" + esal + ", eaddr=" + eaddr + "]";
	}
}
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
	<class name="com.durgasoft.entities.Employee" table="emp1">
		<id name="eno"/>
		<property name="ename"/>
		<property name="esal"/>
		<property name="eaddr"/>
	</class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
            <property
name="hibernate.connection.driver_Class">com.mysql.cj.jdbc.Driver</pro
perty>
            <property
name="hibernate.connection.url">jdbc:mysql://localhost:3300/durgadb</p
roperty>
            <property
name="hibernate.connection.username">root</property>
            <property
name="hibernate.connection.password">root</property>
            <property name="hibernate.show_sql">true</property>
            <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
            <mapping resource="Employee.hbm.xml"/>
      </session-factory>
</hibernate-configuration>
```

Test.java

```
package com.durgasoft.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import com.durgasoft.entities.Employee;
public class Test {
      public static void main(String[] args) {
            Configuration configuration = new Configuration();
            configuration.configure();
            SessionFactory sessionFactory =
configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();
```

```
            Employee employee = (Employee) session.load(Employee.class,
222);
            System.out.println(employee);

            /*
             * if(employee == null) { System.out.println("Employee Does
Not Exist"); }else {
             * System.out.println("Employee Details");
             * System.out.println("----------------------");
             * System.out.println("Employee Number     :
"+employee.getEno());
             * System.out.println("Employee Name       :
"+employee.getEname());
             * System.out.println("Employee Salary     :
"+employee.getEsal());
             * System.out.println("Employee Address    :
"+employee.getEaddr()); }
             */
      }
}
```

Q)What is the difference between update() method and saveOrUpdate() method?

—----------

Ans:

—---

update() method will update the existing record, if no record is available then update() method will raise an exception.

saveOrUpdate() method will save the record if the record is not available , if the record is available then it will perform update operations.

EX:

Employee.java

```
package com.durgasoft.beans;
public class Employee {

      private int eno;
```

```
	private String ename;
	private float esal;
	private String eaddr;

	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
		this.eno = eno;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public float getEsal() {
		return esal;
	}
	public void setEsal(float esal) {
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}



}
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping>
	<class name="com.durgasoft.beans.Employee" table="emp1">
		<id name="eno"/>
		<property name="ename"/>
		<property name="esal"/>
		<property name="eaddr"/>
	</class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
	<session-factory>
		<property
name="hibernate.connection.driver_Class">com.mysql.cj.jdbc.Driver</pro
perty>
		<property
name="hibernate.connection.url">jdbc:mysql://localhost:3300/durgadb</p
roperty>
		<property
name="hibernate.connection.username">root</property>
		<property
name="hibernate.connection.password">root</property>
		<property name="hibernate.show_sql">true</property>
		<property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
		<mapping resource="Employee.hbm.xml"/>
	</session-factory>
</hibernate-configuration>
```

Test.java

```
package com.durgasoft.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
```

```
import org.hibernate.cfg.Configuration;
import com.durgasoft.beans.Employee;
public class Test {
	public static void main(String[] args) {
		Configuration configuration = new Configuration();
		configuration.configure();
		SessionFactory sessionFactory =
configuration.buildSessionFactory();
		Session session = sessionFactory.openSession();
		Employee employee = new Employee();
		employee.setEno(111);
		employee.setEname("XXX");
		employee.setEsal(9000);
		employee.setEaddr("Chennai");
		Transaction transaction = session.beginTransaction();
		session.update(employee);
		transaction.commit();
		System.out.println("Employee Updated Successfully");
		session.close();
		sessionFactory.close();
	}
}
```

```
Hibernate: update emp1 set ename=?, esal=?, eaddr=? where eno=?
Employee Updated Successfully
```

EX:
Employee.java

```
package com.durgasoft.beans;
public class Employee {

	private int eno;
	private String ename;
	private float esal;
	private String eaddr;

	public int getEno() {
		return eno;
	}
	public void setEno(int eno) {
```

```
            this.eno = eno;
      }
      public String getEname() {
            return ename;
      }
      public void setEname(String ename) {
            this.ename = ename;
      }
      public float getEsal() {
            return esal;
      }
      public void setEsal(float esal) {
            this.esal = esal;
      }
      public String getEaddr() {
            return eaddr;
      }
      public void setEaddr(String eaddr) {
            this.eaddr = eaddr;
      }



}
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
      <class name="com.durgasoft.beans.Employee" table="emp1">
            <id name="eno"/>
            <property name="ename"/>
            <property name="esal"/>
            <property name="eaddr"/>
      </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
            <property
name="hibernate.connection.driver_Class">com.mysql.cj.jdbc.Driver</pro
perty>
            <property
name="hibernate.connection.url">jdbc:mysql://localhost:3300/durgadb</p
roperty>
            <property
name="hibernate.connection.username">root</property>
            <property
name="hibernate.connection.password">root</property>
            <property name="hibernate.show_sql">true</property>
            <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
            <mapping resource="Employee.hbm.xml"/>
      </session-factory>
</hibernate-configuration>
```

Test.java

```
package com.durgasoft.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.durgasoft.beans.Employee;
public class Test {
      public static void main(String[] args) {
            Configuration configuration = new Configuration();
            configuration.configure();
            SessionFactory sessionFactory =
configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();
```

```
            Employee employee = new Employee();
            employee.setEno(222);
            employee.setEname("YYY");
            employee.setEsal(8000);
            employee.setEaddr("Pune");
            Transaction transaction = session.beginTransaction();
            session.saveOrUpdate(employee);
            transaction.commit();
            System.out.println("Employee Updated Successfully");
            session.close();
            sessionFactory.close();
      }
}
```

EX:

Employee.java

```
package com.durgasoft.beans;
public class Employee {

      private int eno;
      private String ename;
      private float esal;
      private String eaddr;

      public int getEno() {
            return eno;
      }
      public void setEno(int eno) {
            this.eno = eno;
      }
      public String getEname() {
            return ename;
      }
      public void setEname(String ename) {
            this.ename = ename;
      }
      public float getEsal() {
            return esal;
      }
      public void setEsal(float esal) {
```

```
		this.esal = esal;
	}
	public String getEaddr() {
		return eaddr;
	}
	public void setEaddr(String eaddr) {
		this.eaddr = eaddr;
	}



}
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
	<class name="com.durgasoft.beans.Employee" table="emp1">
		<id name="eno"/>
		<property name="ename"/>
		<property name="esal"/>
		<property name="eaddr"/>
	</class>
</hibernate-mapping>
```

hibermate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
	<session-factory>
		<property
name="hibernate.connection.driver_Class">com.mysql.cj.jdbc.Driver</pro
perty>
```

```
            <property
name="hibernate.connection.url">jdbc:mysql://localhost:3300/durgadb</p
roperty>
            <property
name="hibernate.connection.username">root</property>
            <property
name="hibernate.connection.password">root</property>
            <property name="hibernate.show_sql">true</property>
            <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
            <mapping resource="Employee.hbm.xml"/>
      </session-factory>
</hibernate-configuration>
```

Test.java

```
package com.durgasoft.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.durgasoft.beans.Employee;
public class Test {
      public static void main(String[] args) {
            Configuration configuration = new Configuration();
            configuration.configure();
            SessionFactory sessionFactory =
configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();
            Employee employee = new Employee();
            employee.setEno(222);

            Transaction transaction = session.beginTransaction();
            session.delete(employee);
            transaction.commit();
            System.out.println("Employee Deleted Successfully");
            session.close();
            sessionFactory.close();
      }
}
```

Hibernate Application with MAVEN:

Employee.java

```java
package com.durgasoft.entities;

public class Employee {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public int getEno() {
       return eno;
   }

   public void setEno(int eno) {
       this.eno = eno;
   }

   public String getEname() {
       return ename;
   }

   public void setEname(String ename) {
       this.ename = ename;
   }

   public float getEsal() {
       return esal;
   }

   public void setEsal(float esal) {
       this.esal = esal;
   }

   public String getEaddr() {
       return eaddr;
   }

   public void setEaddr(String eaddr) {
       this.eaddr = eaddr;
   }
}
```

Employee.hbm.xml

```
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="com.durgasoft.entities.Employee" table="emp1">
        <id name="eno"/>
        <property name="ename"/>
        <property name="esal"/>
        <property name="eaddr"/>
   </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</pr
operty>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3300/durgadb</
property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property
>
        <property name="hibernate.show_sql">true</property>
        <mapping resource="Employee.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

Main.java

```
package com.durgasoft;
import com.durgasoft.entities.Employee;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
```

```
public class Main {
   public static void main(String[] args) {
       Configuration configuration = new Configuration();
       configuration.configure();

       SessionFactory sessionFactory =
configuration.buildSessionFactory();
       Session session = sessionFactory.openSession();
       Employee employee = (Employee)
session.get("com.durgasoft.entities.Employee", 111);
       if(employee == null){
           System.out.println("Employee Does not exist");
       }else{
           System.out.println("Employee Details");
           System.out.println("----------------------");
           System.out.println("Employee NUmber    :
"+employee.getEno());
           System.out.println("Employee Name      :
"+employee.getEname());
           System.out.println("Employee Salary    :
"+employee.getEsal());
           System.out.println("Employee Address   :
"+employee.getEaddr());
       }
   }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.durgasoft</groupId>
   <artifactId>app08</artifactId>
   <version>1.0-SNAPSHOT</version>

   <properties>
       <maven.compiler.source>17</maven.compiler.source>
       <maven.compiler.target>17</maven.compiler.target>
```

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
       <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
       <dependency>
           <groupId>org.hibernate</groupId>
           <artifactId>hibernate-core</artifactId>
           <version>3.6.10.Final</version>
       </dependency>
       <!--
https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
       <dependency>
           <groupId>mysql</groupId>
           <artifactId>mysql-connector-java</artifactId>
           <version>8.0.33</version>
       </dependency>
       <!-- https://mvnrepository.com/artifact/javassist/javassist
-->
       <dependency>
           <groupId>javassist</groupId>
           <artifactId>javassist</artifactId>
           <version>3.12.1.GA</version>
       </dependency>

   </dependencies>
</project>
```

EX:

Employee.java

```
package com.durgasoft.beans;

import java.io.Serializable;

public class Employee implements Serializable {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;
```

```
    public int getEno() {
        return eno;
    }

    public void setEno(int eno) {
        this.eno = eno;
    }

    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }

    public float getEsal() {
        return esal;
    }

    public void setEsal(float esal) {
        this.esal = esal;
    }

    public String getEaddr() {
        return eaddr;
    }

    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }
}
```

EmployeeController.java

```
package com.durgasoft.controller;

import com.durgasoft.beans.Employee;
import com.durgasoft.factories.EmployeeServiceFactory;
```

```
import com.durgasoft.service.EmployeeService;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class EmployeeController {
   BufferedReader bufferedReader = new
BufferedReader(new InputStreamReader(System.in));
   EmployeeService employeeService =
EmployeeServiceFactory.getEmployeeService();
   public void addEmployee(){
       try {

           System.out.print("Employee Number    : ");
           int eno =
Integer.parseInt(bufferedReader.readLine());
           Employee emp =
employeeService.searchEmployee(eno);
           if(emp == null) {

               System.out.print("Employee Name      :
");
               String ename =
bufferedReader.readLine();
               System.out.print("Employee Salary    :
");
               float esal =
Float.parseFloat(bufferedReader.readLine());
               System.out.print("Employee Address   :
");
               String eaddr =
bufferedReader.readLine();

               Employee employee = new Employee();
```

```
                employee.setEno(eno);
                employee.setEname(ename);
                employee.setEsal(esal);
                employee.setEaddr(eaddr);

                String status =
employeeService.addEmployee(employee);
                System.out.println("Status    : " +
status);
            }else{
                System.out.println("Status    :
Employee Existed Already");
            }


        }catch (Exception exception){
            exception.printStackTrace();
        }
    }
    public void searchEmployee(){
        try {
            System.out.print("Employee Number   : ");
            int eno =
Integer.parseInt(bufferedReader.readLine());
            Employee employee =
employeeService.searchEmployee(eno);
            if(employee == null){
                System.out.println("Status    :
Employee Does Not Exist");
            }else{
                System.out.println("Employee Details");

System.out.println("---------------------");
```

```
                System.out.println("Employee Number
: "+employee.getEno());
                System.out.println("Employee Name
: "+employee.getEname());
                System.out.println("Employee Salary
: "+employee.getEsal());
                System.out.println("Employee Address
: "+employee.getEaddr());
            }
        }catch (Exception exception){
            exception.printStackTrace();
        }
    }
    public void updateEmployee(){
        try{
            System.out.print("Employee Number     : ");
            int eno =
Integer.parseInt(bufferedReader.readLine());
            Employee employee =
employeeService.searchEmployee(eno);
            if(employee == null){
                System.out.println("Status    :
Employee Does Not Exist");
            }else{
                System.out.print("Employee Name   : Old
: "+employee.getEname()+"  New : ");
                String ename_New =
bufferedReader.readLine();
                if(ename_New == null ||
ename_New.equals("")){
                    ename_New = employee.getEname();
                }
                System.out.print("Employee Salary  :
Old : "+employee.getEsal()+" New : ");
```

```
                String sal = bufferedReader.readLine();
                float esal_New = 0.0f;
                if(sal == null || sal.equals("")){
                    esal_New = employee.getEsal();
                }else {
                    esal_New = Float.parseFloat(sal);
                }
                System.out.print("Employee Address  :
Old  : "+employee.getEaddr()+" New  : ");
                String eaddr_New =
bufferedReader.readLine();
                if(eaddr_New == null ||
eaddr_New.equals("")){
                    eaddr_New = employee.getEaddr();
                }

                Employee employee_New = new Employee();
                employee_New.setEno(eno);
                employee_New.setEname(ename_New);
                employee_New.setEsal(esal_New);
                employee_New.setEaddr(eaddr_New);

                String status =
employeeService.updateEmployee(employee_New);
                System.out.println("Status    :
"+status);
            }
        }catch (Exception exception){
            exception.printStackTrace();
        }
    }
    public  void deleteEmployee(){
        try{
            System.out.print("Employee Number    : ");
```

```
            int eno =
Integer.parseInt(bufferedReader.readLine());
            Employee employee =
employeeService.searchEmployee(eno);
            if(employee == null){
                System.out.println("Status    : Employee
Does Not Exist");
            }else {
                String status =
employeeService.deleteEmployee(eno);
                System.out.println("Status    : " +
status);
            }
        }catch (Exception exception){
            exception.printStackTrace();
        }
    }
}
```

EmployeeService.java

```
package com.durgasoft.service;

import com.durgasoft.beans.Employee;

public interface EmployeeService {
    public String addEmployee(Employee employee);
    public Employee searchEmployee(int eno);
    public String updateEmployee(Employee employee);
    public String deleteEmployee(int eno);
}
```

EmployeeServiceImpl.java

```
package com.durgasoft.service;

import com.durgasoft.beans.Employee;
```

```java
import com.durgasoft.dao.EmployeeDao;
import com.durgasoft.factories.EmployeeDaoFactory;

public class EmployeeServiceImpl implements
EmployeeService{
    EmployeeDao employeeDao =
EmployeeDaoFactory.getEmployeeDao();
    @Override
    public String addEmployee(Employee employee) {
        String status = employeeDao.add(employee);
        return status;
    }

    @Override
    public Employee searchEmployee(int eno) {
        Employee employee = employeeDao.search(eno);
        return employee;
    }

    @Override
    public String updateEmployee(Employee employee) {
        String status = employeeDao.update(employee);
        return status;
    }

    @Override
    public String deleteEmployee(int eno) {
        String status = employeeDao.delete(eno);
        return status;
    }
}
```

EmployeeDao.java

```java
package com.durgasoft.dao;
```

```
import com.durgasoft.beans.Employee;

public interface EmployeeDao {
   public String add(Employee employee);
   public Employee search(int eno);
   public String update(Employee employee);
   public String delete(int eno);
}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import com.durgasoft.beans.Employee;
import com.durgasoft.factories.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

public class EmployeDaoImpl implements EmployeeDao{
   SessionFactory sessionFactory =
HibernateUtil.getSessionFactory();
   @Override
   public String add(Employee employee) {
       String status = "";
       try{
           Session session =
sessionFactory.openSession();
           Transaction transaction =
session.beginTransaction();
           int eno = (Integer) session.save(employee);
           transaction.commit();
           if(eno == employee.getEno()){
               status = "SUCCESS";
           }else{
```

```
                status = "FAILURE";
            }
        }catch (Exception exception){
            status = "FAILURE";
            exception.printStackTrace();
        }
        return status;
    }

    @Override
    public Employee search(int eno) {
        Employee employee = null;
        try{
            Session session =
sessionFactory.openSession();
            employee = (Employee)
session.get("com.durgasoft.beans.Employee", eno);
        }catch (Exception exception){
            exception.printStackTrace();
        }
        return employee;
    }

    @Override
    public String update(Employee employee) {
        String status = "";
        try{
            Session session =
sessionFactory.openSession();
            Transaction transaction =
session.beginTransaction();
            session.update(employee);
            transaction.commit();
            status = "SUCCESS";
```

```
        }catch (Exception exception){
            status = "FAILURE";
            exception.printStackTrace();
        }
        return status;
    }

    @Override
    public String delete(int eno) {
        String status = "";
        try{
            Session session =
sessionFactory.openSession();
            Transaction transaction =
session.beginTransaction();
            Employee employee = new Employee();
            employee.setEno(eno);
            session.delete(employee);
            transaction.commit();
            status = "SUCCESS";
        }catch (Exception exception){
            status = "FAILURE";
            exception.printStackTrace();
        }
        return status;
    }
}
```

Main.java

```
package com.durgasoft;

import com.durgasoft.controller.EmployeeController;
import
com.durgasoft.factories.EmployeeControllerFactory;
```

```
import com.durgasoft.factories.HibernateUtil;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Main {
    static {
        HibernateUtil.getSessionFactory();
    }
    public static void main(String[] args)throws
Exception {
        BufferedReader bufferedReader = new
BufferedReader(new InputStreamReader(System.in));
        EmployeeController employeeController =
EmployeeControllerFactory.getEmployeeController();
        System.out.println("Employee Management
Application");

System.out.println("==================================
========");
        while (true){
            System.out.println();
            System.out.println("1. ADD Employee");
            System.out.println("2. SEARCH Employee");
            System.out.println("3. UPDATE Employee");
            System.out.println("4. DELETE Employee");
            System.out.println("5. EXIT");
            System.out.print("Your Option   : ");
            int userOption =
Integer.parseInt(bufferedReader.readLine());
            switch (userOption){
                case 1:
                    System.out.println("ADD Employee
Module");
```

```
System.out.println("----------------------------");
                    employeeController.addEmployee();
                    break;
                case 2:
                    System.out.println("SEARCH Employee
Module");

System.out.println("--------------------------------")
;

employeeController.searchEmployee();
                    break;
                case 3:
                    System.out.println("UPDATE Employee
Module");

System.out.println("----------------------------------
");

employeeController.updateEmployee();
                    break;
                case 4:
                    System.out.println("DELETE Employee
Module");

System.out.println("------------------------------------
--");

employeeController.deleteEmployee();
                    break;
                case 5:
```

```
                    System.out.println("**********Thank
You for using Employee Management Application
******");
                    System.exit(0);
                    break;
                default:
                    System.out.println("INvalid Entry,
please provide the numbers from 1, 2,3,4 and 5");
                    break;
            }
        }

    }
}
```

Employee.hbm.xml

```
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dt
d">
<hibernate-mapping>
   <class name="com.durgasoft.beans.Employee"
table="emp1">
        <id name="eno"/>
        <property name="ename"/>
        <property name="esal"/>
        <property name="eaddr"/>
   </class>
</hibernate-mapping>
```

hibernate.cfg.xml

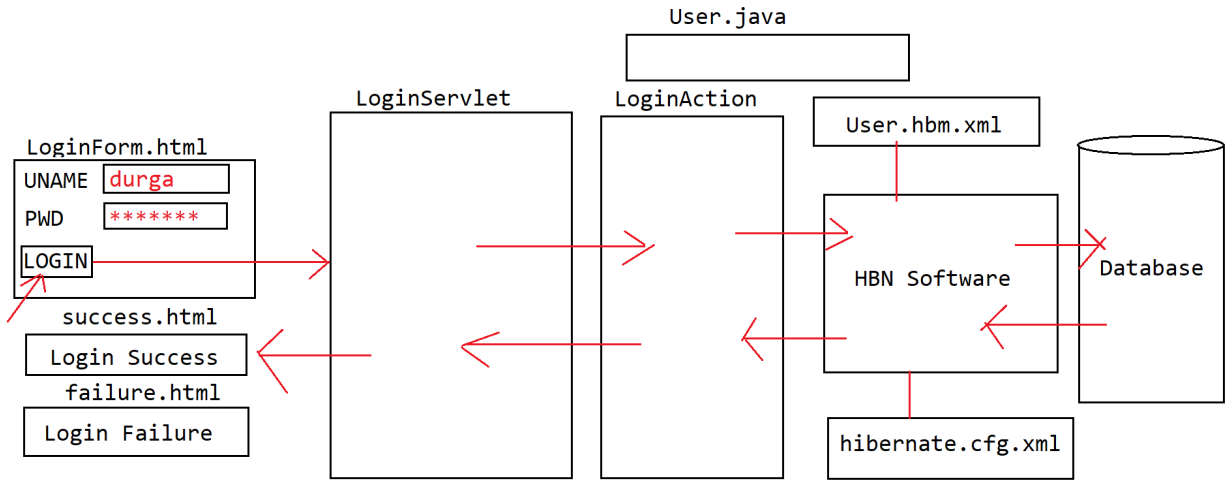
```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD
3.0//EN"
```

```
"http://www.hibernate.org/dtd/hibernate-configuration-
3.0.dtd">
<hibernate-configuration>
   <session-factory>
       <property
name="connection.driver_class">oracle.jdbc.OracleDrive
r</property>
       <property
name="connection.url">jdbc:oracle:thin:@localhost:1521
:xe</property>
       <property
name="connection.username">system</property>
       <property
name="connection.password">durga</property>
       <property
name="hibernate.dialect">org.hibernate.dialect.OracleD
ialect</property>
       <mapping resource="Employee.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.durgasoft</groupId>
   <artifactId>app11</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>

    <properties>

<maven.compiler.source>17</maven.compiler.source>

<maven.compiler.target>17</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sou
rceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>3.6.10.Final</version>
        </dependency>
        <dependency>
            <groupId>javassist</groupId>
            <artifactId>javassist</artifactId>
            <version>3.12.1.GA</version>
        </dependency>
        <dependency>
            <groupId>com.oracle.database.jdbc</groupId>
            <artifactId>ojdbc11</artifactId>
            <version>23.2.0.0</version>
        </dependency>
    </dependencies>

</project>
```



EX:

loginform.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>user Login Page</title>
</head>
<body>
<h2 style="color: red; text-align: center">Durga Software
Solutions</h2>
<h3 style="color: blue; text-align: center">User Login
Form</h3>
<form method="post" action="./login">
   <table style="margin-left: auto; margin-right: auto">
       <tr>
           <td>User Name</td>
           <td><input type="text" name="uname"></td>
       </tr>
       <tr>
           <td>Password</td>
           <td><input type="password" name="upwd"></td>
       </tr>
       <tr>
```

```
            <td><input type="submit" value="Login"></td>
        </tr>
    </table>
</form>
</body>
</html>
```

success.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User Login Status</title>
</head>
<body>
<h2 style="color: red; text-align: center">Durga Software
Solutions</h2>
<h3 style="color: blue; text-align: center">User Login
Status</h3>
<h1 style="color: green; text-align: center">Login
Success</h1>
<h4 style="text-align: center">
 <a href="./loginform.html">|Login Page|</a>
</h4>
</body>
</html>
```

failure.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>User Login Status</title>
</head>
<body>
<h2 style="color: red; text-align: center">Durga
Software Solutions</h2>
```

```
<h3 style="color: blue; text-align: center">User Login
Status</h3>
<h1 style="color: green; text-align: center">Login
Failure</h1>
<h4 style="text-align: center">
 <a href="./loginform.html">|Login Page|</a>
</h4>
</body>
</html>
```

LoginServlet.java

```
package com.durgasoft.app10.servlets;

import com.durgasoft.app10.action.LoginAction;
import com.durgasoft.app10.beans.User;
import
com.durgasoft.app10.factory.LoginActionFactory;
import com.durgasoft.app10.util.HibernateUtil;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

@WebServlet(urlPatterns = {"/login"}, loadOnStartup
= 1)
public class LoginServlet extends HttpServlet {
    @Override
    public void init() throws ServletException {
        HibernateUtil.getSessionFactory();
    }
```

```
   @Override
   protected void doPost(HttpServletRequest
request, HttpServletResponse response) throws
ServletException, IOException {
       String uname =
request.getParameter("uname");
       String upwd = request.getParameter("upwd");
       User user = new User();
       user.setUname(uname);
       user.setUpwd(upwd);
       LoginAction loginAction =
LoginActionFactory.getLoginAction();
       String status =
loginAction.checkLogin(user);
       RequestDispatcher requestDispatcher = null;
       if(status.equals("success")){
           requestDispatcher =
request.getRequestDispatcher("success.html");
           requestDispatcher.forward(request,
response);
       }else{
           requestDispatcher =
request.getRequestDispatcher("failure.html");
           requestDispatcher.forward(request,
response);
       }
   }
}
```

LoginAction.java

```
package com.durgasoft.app10.action;
```

```
import com.durgasoft.app10.beans.User;
import com.durgasoft.app10.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

public class LoginAction {

   public String checkLogin(User user){
       String status = "";
       try{
           SessionFactory sessionFactory =
HibernateUtil.getSessionFactory();
           Session session =
sessionFactory.openSession();
           User user1 = (User) session.get(User.class,
user.getUname());
           if(user1 == null){
               status = "failure";
           }else{
               if
(user.getUpwd().equals(user1.getUpwd())){
                   status = "success";
               }else{
                   status = "failure";
               }
           }
       }catch (Exception exception){
           exception.printStackTrace();
       }
       return status;
   }
}
```

LoginActionFactory.java

```java
package com.durgasoft.app10.factory;

import com.durgasoft.app10.action.LoginAction;

public class LoginActionFactory {
   private static LoginAction loginAction;
   static{
       loginAction = new LoginAction();
   }

   public static LoginAction getLoginAction() {
       return loginAction;
   }
}
```

User.java

```java
package com.durgasoft.app10.beans;

public class User {
   private String uname;
   private String upwd;

   public String getUname() {
       return uname;
   }

   public void setUname(String uname) {
       this.uname = uname;
   }

   public String getUpwd() {
       return upwd;
   }
```

```
    public void setUpwd(String upwd) {
        this.upwd = upwd;
    }
}
```

HibernateUtil.java

```
package com.durgasoft.app10.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static SessionFactory sessionFactory;
    static {
        Configuration configuration = new
Configuration();
        configuration.configure();
        sessionFactory =
configuration.buildSessionFactory();
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

User.hbm.xml

```
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dt
d">
<hibernate-mapping>
```

```
   <class name="com.durgasoft.app10.beans.User"
table="REG_USERS">
       <id name="uname"/>
       <property name="upwd"/>
   </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
       "-//Hibernate/Hibernate Configuration DTD
3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-
3.0.dtd">
<hibernate-configuration>
   <session-factory>
       <property
name="connection.driver_class">com.mysql.cj.jdbc.Drive
r</property>
       <property
name="connection.url">jdbc:mysql://localhost:3306/durg
adb</property>
       <property
name="connection.username">root</property>
       <property
name="connection.password">root</property>
       <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDi
alect</property>
       <mapping resource="User.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartae
e https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
         version="5.0">
   <welcome-file-list>
       <welcome-file>loginform.html</welcome-file>
   </welcome-file-list>
</web-app>
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.durgasoft</groupId>
 <artifactId>app10</artifactId>
 <version>1.0-SNAPSHOT</version>
 <name>app10</name>
 <packaging>war</packaging>

 <properties>

<project.build.sourceEncoding>UTF-8</project.build.sou
rceEncoding>
   <maven.compiler.target>11</maven.compiler.target>
   <maven.compiler.source>11</maven.compiler.source>
```

```
   <junit.version>5.9.2</junit.version>
 </properties>

 <dependencies>
<dependency>
     <groupId>jakarta.servlet</groupId>
     <artifactId>jakarta.servlet-api</artifactId>
     <version>5.0.0</version>
     <scope>provided</scope>
   </dependency>
<dependency>
     <groupId>org.junit.jupiter</groupId>
     <artifactId>junit-jupiter-api</artifactId>
     <version>${junit.version}</version>
     <scope>test</scope>
   </dependency>
     <dependency>
       <groupId>org.junit.jupiter</groupId>
       <artifactId>junit-jupiter-engine</artifactId>
       <version>${junit.version}</version>
       <scope>test</scope>
     </dependency>
     <!--
https://mvnrepository.com/artifact/org.hibernate/hiber
nate-core -->
     <dependency>
         <groupId>org.hibernate</groupId>
         <artifactId>hibernate-core</artifactId>
         <version>3.6.10.Final</version>
     </dependency>
     <dependency>
         <groupId>javassist</groupId>
         <artifactId>javassist</artifactId>
         <version>3.12.1.GA</version>
```

```
    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.0.33</version>
    </dependency>

 </dependencies>

 <build>
   <plugins>
<plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-war-plugin</artifactId>
       <version>3.3.2</version>
    </plugin>    </plugins>
 </build>
</project>
```

EX:

---

header.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
</head>
<body>
<h1 style="text-align: center; color: white">
 DURGA SOFTWARE SOLUTIONS
</h1>
</body>
</html>
```

menu.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
</head>
<body>

<h3 style="text-align: center">
   <a href="./addhome.jsp">ADD Student</a><br><br>
   <a href="./searchhome.jsp">SEARCH Student</a><br><br>
   <a href="./updatehome.jsp">UPDATE Student</a><br><br>
   <a href="./deletehome.jsp">DELETE Student</a>
</h3>
</body>
</html>
```

welcome.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
</head>
<body>
<h2 style="color: red; text-align: center">
   <marquee>
       Welcome To Durga Software Solutions
   </marquee>
</h2>
</body>
</html>
```

footer.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
</head>
<body>
<h3 style="text-align: center; color: white">
```

```
   Durgasoft India Private Ltd. , 202, HMDA, Mitrivanam, Ameerpet,
Hyd-38.
</h3>
</body>
</html>
```

addform.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
</head>
<body>
<form method="post" action="add.do">
 <table style="margin-left: auto; margin-right: auto">
   <tr>
     <td>Student Id</td>
     <td><input type="text" name="sid"></td>
   </tr>
   <tr>
     <td>Student Name</td>
     <td><input type="text" name="sname"></td>
   </tr>
   <tr>
     <td>Student Address</td>
     <td><input type="text" name="saddr"></td>
   </tr>
   <tr>
     <td><input type="submit" value="ADD"></td>
   </tr>
 </table>
</form>
</body>
</html>
```

Searchform.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
```

```
</head>
<body>
<form method="post" action="search.do">
   <table style="margin-left: auto; margin-right: auto">
       <tr>
           <td>Student Id</td>
           <td><input type="text" name="sid"></td>
       </tr>
       <tr>
           <td><input type="submit" value="SEARCH"></td>
       </tr>
   </table>
</form>
</body>
</html>
```

updateform.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
</head>
<body>
<form method="post" action="editform.do">
   <table style="margin-left: auto; margin-right: auto">
       <tr>
           <td>Student Id</td>
           <td><input type="text" name="sid"></td>
       </tr>
       <tr>
           <td><input type="submit" value="UPDATE"></td>
       </tr>
   </table>
</form>
</body>
</html>
```

deleteform.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
```

```
 <title>Title</title>
</head>
<body>
<form method="post" action="delete.do">
 <table style="margin-left: auto; margin-right: auto">
   <tr>
     <td>Student Id</td>
     <td><input type="text" name="sid"></td>
   </tr>
   <tr>
     <td><input type="submit" value="DELETE"></td>
   </tr>
 </table>
</form>
</body>
</html>
```

welcomehome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
   <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
   <tr style="height: 20%">
       <td style="background-color: maroon;" colspan="2">
           <jsp:include page="header.html"/>
       </td>
   </tr>
   <tr style="height: 65%">
       <td style="width: 20%; background-color: bisque;">
           <jsp:include page="menu.html"/>
       </td>
       <td style="background-color: aqua;">
           <jsp:include page="welcome.html"/>
       </td>
   </tr>
       </td>
   </tr>
   <tr style="height: 15%;">
       <td style="background-color: blue;" colspan="2">
```

```
            <jsp:include page="footer.html"/>
        </td>
    </tr>
</table>
</body>
</html>
```

addhome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
    <tr style="height: 20%">
        <td style="background-color: maroon;" colspan="2">
            <jsp:include page="header.html"/>
        </td>
    </tr>
    <tr style="height: 65%">
        <td style="width: 20%; background-color: bisque;">
            <jsp:include page="menu.html"/>
        </td>
        <td style="background-color: aqua;">
            <jsp:include page="addform.html"/>
        </td>
    </tr>
    </td>
    </tr>
    <tr style="height: 15%;">
        <td style="background-color: blue;" colspan="2">
            <jsp:include page="footer.html"/>
        </td>
    </tr>
</table>
</body>
</html>
```

searchhome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
   <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
   <tr style="height: 20%">
       <td style="background-color: maroon;" colspan="2">
           <jsp:include page="header.html"/>
       </td>
   </tr>
   <tr style="height: 65%">
       <td style="width: 20%; background-color: bisque;">
           <jsp:include page="menu.html"/>
       </td>
       <td style="background-color: aqua;">
           <jsp:include page="searchform.html"/>
       </td>
   </tr>
   </td>
   </tr>
   <tr style="height: 15%;">
       <td style="background-color: blue;" colspan="2">
           <jsp:include page="footer.html"/>
       </td>
   </tr>
</table>
</body>
</html>
```

updatehome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
   <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
   <tr style="height: 20%">
       <td style="background-color: maroon;" colspan="2">
           <jsp:include page="header.html"/>
```

```
        </td>
    </tr>
    <tr style="height: 65%">
        <td style="width: 20%; background-color: bisque;">
            <jsp:include page="menu.html"/>
        </td>
        <td style="background-color: aqua;">
            <jsp:include page="updateform.html"/>
        </td>
    </tr>
    </td>
    </tr>
    <tr style="height: 15%;">
        <td style="background-color: blue;" colspan="2">
            <jsp:include page="footer.html"/>
        </td>
    </tr>
</table>
</body>
</html>
```

deletehome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
    <tr style="height: 20%">
        <td style="background-color: maroon;" colspan="2">
            <jsp:include page="header.html"/>
        </td>
    </tr>
    <tr style="height: 65%">
        <td style="width: 20%; background-color: bisque;">
            <jsp:include page="menu.html"/>
        </td>
        <td style="background-color: aqua;">
            <jsp:include page="deleteform.html"/>
        </td>
    </tr>
```

```
    </td>
    </tr>
    <tr style="height: 15%;">
        <td style="background-color: blue;" colspan="2">
            <jsp:include page="footer.html"/>
        </td>
    </tr>
</table>
</body>
</html>
```

statushome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
    <tr style="height: 20%">
        <td style="background-color: maroon;" colspan="2">
            <jsp:include page="header.html"/>
        </td>
    </tr>
    <tr style="height: 65%">
        <td style="width: 20%; background-color: bisque;">
            <jsp:include page="menu.html"/>
        </td>
        <td style="background-color: aqua;">
            <jsp:include page="status.jsp"/>
        </td>
    </tr>
    </td>
    </tr>
    <tr style="height: 15%;">
        <td style="background-color: blue;" colspan="2">
            <jsp:include page="footer.html"/>
        </td>
    </tr>
</table>
</body>
</html>
```

studentdetailshome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
   <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
   <tr style="height: 20%">
       <td style="background-color: maroon;" colspan="2">
           <jsp:include page="header.html"/>
       </td>
   </tr>
   <tr style="height: 65%">
       <td style="width: 20%; background-color: bisque;">
           <jsp:include page="menu.html"/>
       </td>
       <td style="background-color: aqua;">
           <jsp:include page="studentdetails.jsp"/>
       </td>
   </tr>
   </td>
   </tr>
   <tr style="height: 15%;">
       <td style="background-color: blue;" colspan="2">
           <jsp:include page="footer.html"/>
       </td>
   </tr>
</table>
</body>
</html>
```

editformhome.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
   <title>Student Management System Home</title>
</head>
<body>
<table style="width: 100%; height: 100%">
```

```
    <tr style="height: 20%">
        <td style="background-color: maroon;" colspan="2">
            <jsp:include page="header.html"/>
        </td>
    </tr>
    <tr style="height: 65%">
        <td style="width: 20%; background-color: bisque;">
            <jsp:include page="menu.html"/>
        </td>
        <td style="background-color: aqua;">
            <jsp:include page="editform.jsp"/>
        </td>
    </tr>
    </td>
    </tr>
    <tr style="height: 15%;">
        <td style="background-color: blue;" colspan="2">
            <jsp:include page="footer.html"/>
        </td>
    </tr>
</table>
</body>
</html>
```

Student.java

```
package com.durgasoft.app12.beans;

public class Student {
    private String sid;
    private String sname;
    private String saddr;

    public String getSid() {
        return sid;
    }

    public void setSid(String sid) {
        this.sid = sid;
    }

    public String getSname() {
        return sname;
    }
```

```
    public void setSname(String sname) {
        this.sname = sname;
    }

    public String getSaddr() {
        return saddr;
    }

    public void setSaddr(String saddr) {
        this.saddr = saddr;
    }
}
```

ControllerServlet.java

```
package com.durgasoft.app12.controller;

import com.durgasoft.app12.beans.Student;
import com.durgasoft.app12.factory.StudentDaoFactory;
import com.durgasoft.app12.factory.StudentServiceFactory;
import com.durgasoft.app12.service.StudentService;
import com.durgasoft.app12.util.HibernateUtil;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

@WebServlet(urlPatterns = {"*.do"}, loadOnStartup = 1)
public class ControllerServlet extends HttpServlet {
    @Override
    public void init() throws ServletException {
        HibernateUtil.getSessionFactory();
        StudentDaoFactory.getStudentDao();
        StudentServiceFactory.getStudentService();

    }

    @Override
```

```
   protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
       String status = "";
       String sid = "";
       String sname = "";
       String saddr = "";
       Student student = null;
       RequestDispatcher requestDispatcher = null;
       StudentService studentService =
StudentServiceFactory.getStudentService();
       String requestURI = request.getRequestURI();
       if(requestURI.endsWith("add.do")){
           sid = request.getParameter("sid");
           sname = request.getParameter("sname");
           saddr = request.getParameter("saddr");
           student = new Student();
           student.setSid(sid);
           student.setSname(sname);
           student.setSaddr(saddr);
           Student student1 = studentService.searchStudent(sid);
           if(student1 == null) {
               status = studentService.addStudent(student);
           }else{
               status = "Student Existed Already";
           }
           request.setAttribute("status", status);
           requestDispatcher =
request.getRequestDispatcher("statushome.jsp");
           requestDispatcher.forward(request, response);
       }
       if(requestURI.endsWith("search.do")){
           sid = request.getParameter("sid");
           student = studentService.searchStudent(sid);

           if(student == null){
               status = "Student Does Not Exist";
               request.setAttribute("status", status);
               requestDispatcher =
request.getRequestDispatcher("statushome.jsp");
               requestDispatcher.forward(request, response);
           }else {
               request.setAttribute("student", student);
```

```
                requestDispatcher =
request.getRequestDispatcher("studentdetailshome.jsp");
                requestDispatcher.forward(request, response);
            }
        }
        if(requestURI.endsWith("editform.do")){
            sid = request.getParameter("sid");
            student = studentService.searchStudent(sid);
            if(student == null){
                status = "Student Does Not Exist";
                request.setAttribute("status", status);
                requestDispatcher =
request.getRequestDispatcher("statushome.jsp");
                requestDispatcher.forward(request, response);
            }else{
                request.setAttribute("student", student);
                requestDispatcher =
request.getRequestDispatcher("editformhome.jsp");
                requestDispatcher.forward(request, response);
            }
        }
        if(requestURI.endsWith("update.do")){
            sid = request.getParameter("sid");
            sname = request.getParameter("sname");
            saddr = request.getParameter("saddr");
            student = new Student();
            student.setSid(sid);
            student.setSname(sname);
            student.setSaddr(saddr);
            status = studentService.updateStudent(student);
            request.setAttribute("status", status);
            requestDispatcher =
request.getRequestDispatcher("statushome.jsp");
            requestDispatcher.forward(request, response);
        }
        if(requestURI.endsWith("delete.do")){
            sid = request.getParameter("sid");
            student = studentService.searchStudent(sid);
            if(student == null){
                status = "Student Does Not Exist";
                request.setAttribute("status", status);
                requestDispatcher =
request.getRequestDispatcher("statushome.jsp");
```

```
                requestDispatcher.forward(request, response);
            }else{
                status = studentService.deleteStudent(sid);
                request.setAttribute("status", status);
                requestDispatcher =
request.getRequestDispatcher("statushome.jsp");
                requestDispatcher.forward(request, response);
            }
        }

    }
}
```

StudentService.java

```
package com.durgasoft.app12.service;

import com.durgasoft.app12.beans.Student;

public interface StudentService {
    public String addStudent(Student student);
    public Student searchStudent(String sid);
    public String updateStudent(Student student);
    public String deleteStudent(String sid);
}
```

StudentServiceImpl.java

```
package com.durgasoft.app12.service;

import com.durgasoft.app12.beans.Student;
import com.durgasoft.app12.dao.StudentDao;
import com.durgasoft.app12.factory.StudentDaoFactory;

public class StudentServiceImpl implements StudentService{
    private StudentDao studentDao = StudentDaoFactory.getStudentDao();
    @Override
    public String addStudent(Student student) {
        String status = studentDao.add(student);
        return status;
    }

    @Override
    public Student searchStudent(String sid) {
        Student student = studentDao.search(sid);
```

```
        return student;
    }

    @Override
    public String updateStudent(Student student) {
        String status = studentDao.update(student);
        return status;
    }

    @Override
    public String deleteStudent(String sid) {
        String status = studentDao.delete(sid);
        return status;
    }
}
```

StudentDao.java

```
package com.durgasoft.app12.dao;

import com.durgasoft.app12.beans.Student;

public interface StudentDao {
    public String add(Student student);
    public Student search(String sid);
    public String update(Student student);
    public String delete(String sid);
}
```

StudentDaoImpl.java

```
package com.durgasoft.app12.dao;

import com.durgasoft.app12.beans.Student;
import com.durgasoft.app12.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

public class StudentDaoImpl implements StudentDao{
    SessionFactory sessionFactory =
HibernateUtil.getSessionFactory();
    @Override
    public String add(Student student) {
```

```
        String status = "";
        try{
            Session session = sessionFactory.openSession();
            Transaction transaction = session.beginTransaction();
            String pkValue = (String) session.save(student);
            transaction.commit();
            if(pkValue.equals(student.getSid())){
                status = "SUCCESS";
            }else{
                status = "FAILURE";
            }
        }catch (Exception exception){
            exception.printStackTrace();
        }
        return status;
    }

    @Override
    public Student search(String sid) {
        Student student = null;
        try{
            Session session = sessionFactory.openSession();
            student = (Student) session.get(Student.class, sid);
        }catch (Exception exception){
            exception.printStackTrace();
        }
        return student;
    }

    @Override
    public String update(Student student) {
        String status = "";
        try{
            Session session = sessionFactory.openSession();
            Transaction transaction = session.beginTransaction();
            session.update(student);
            transaction.commit();
            status = "SUCCESS";
        }catch (Exception exception){
            status = "FAILURE";
            exception.printStackTrace();
        }
        return status;
```

```
    }

    @Override
    public String delete(String sid) {
        String status = "";
        try{
            Session session = sessionFactory.openSession();
            Transaction transaction = session.beginTransaction();
            Student student = new Student();
            student.setSid(sid);
            session.delete(student);
            transaction.commit();
            status = "SUCCESS";
        }catch (Exception exception){
            status = "FAILURE";
            exception.printStackTrace();
        }
        return status;
    }
}
```

StudentServiceFactory.java

```
package com.durgasoft.app12.factory;

import com.durgasoft.app12.service.StudentService;
import com.durgasoft.app12.service.StudentServiceImpl;

public class StudentServiceFactory {
    private static StudentService studentService;
    static {
        studentService = new StudentServiceImpl();
    }

    public static StudentService getStudentService() {
        return studentService;
    }
}
```

StudentDaoFactory.java

```
package com.durgasoft.app12.factory;

import com.durgasoft.app12.dao.StudentDao;
import com.durgasoft.app12.dao.StudentDaoImpl;
```

```java
public class StudentDaoFactory {
    private static StudentDao studentDao;
    static {
        studentDao = new StudentDaoImpl();
    }

    public static StudentDao getStudentDao() {
        return studentDao;
    }
}
```

HibernateUtil.java

```java
package com.durgasoft.app12.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static SessionFactory sessionFactory;
    static {
        Configuration configuration = new Configuration();
        configuration.configure();
        sessionFactory = configuration.buildSessionFactory();
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Student.hbm.xml

```xml
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.durgasoft.app12.beans.Student" table="student">
        <id name="sid"/>
        <property name="sname"/>
        <property name="saddr"/>
    </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
       "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
       <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</pr
operty>
       <property
name="hibernate.connection.url">jdbc:mysql://localhost:3300/durgadb</
property>
       <property name="hibernate.connection.username">root</property>
       <property name="hibernate.connection.password">root</property>
       <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property
>
       <mapping resource="Student.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
        version="5.0">
   <welcome-file-list>
       <welcome-file>welcomehome.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.durgasoft</groupId>
 <artifactId>app12</artifactId>
```

```
 <version>1.0-SNAPSHOT</version>
 <name>app12</name>
 <packaging>war</packaging>

 <properties>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <maven.compiler.target>11</maven.compiler.target>
   <maven.compiler.source>11</maven.compiler.source>
   <junit.version>5.9.2</junit.version>
 </properties>

 <dependencies>
<dependency>
     <groupId>jakarta.servlet</groupId>
     <artifactId>jakarta.servlet-api</artifactId>
     <version>5.0.0</version>
     <scope>provided</scope>
   </dependency>
<dependency>
     <groupId>org.junit.jupiter</groupId>
     <artifactId>junit-jupiter-api</artifactId>
     <version>${junit.version}</version>
     <scope>test</scope>
   </dependency>
     <dependency>
       <groupId>org.junit.jupiter</groupId>
       <artifactId>junit-jupiter-engine</artifactId>
       <version>${junit.version}</version>
       <scope>test</scope>
     </dependency>
     <dependency>
         <groupId>org.hibernate</groupId>
         <artifactId>hibernate-core</artifactId>
         <version>3.6.10.Final</version>
     </dependency>
     <dependency>
         <groupId>com.mysql</groupId>
         <artifactId>mysql-connector-j</artifactId>
         <version>8.0.33</version>
     </dependency>
     <dependency>
         <groupId>javassist</groupId>
         <artifactId>javassist</artifactId>
```

```
        <version>3.12.1.GA</version>
    </dependency>
 </dependencies>

 <build>
   <plugins>
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.3.2</version>
    </plugin>    </plugins>
 </build>
</project>
```