HIBERNATE

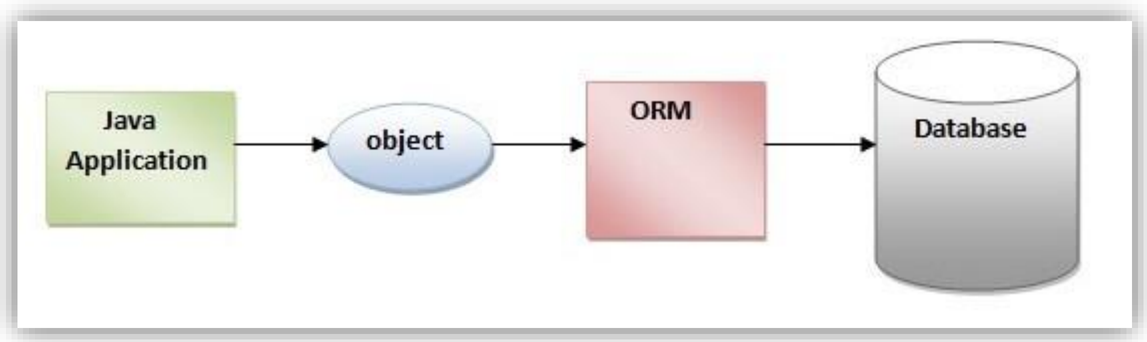
It was started in 2001 by Gavin King as an alternative to EJB2 style entity bean.

⇒ Hibernate Framework:

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

⇒ ORM Tool:

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

⇒ What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

⇒ Advantages of Hibernate Framework:

Following are the advantages of hibernate framework:

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance

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

3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

⇒ HIBERNATE ARCHITECTURE:-

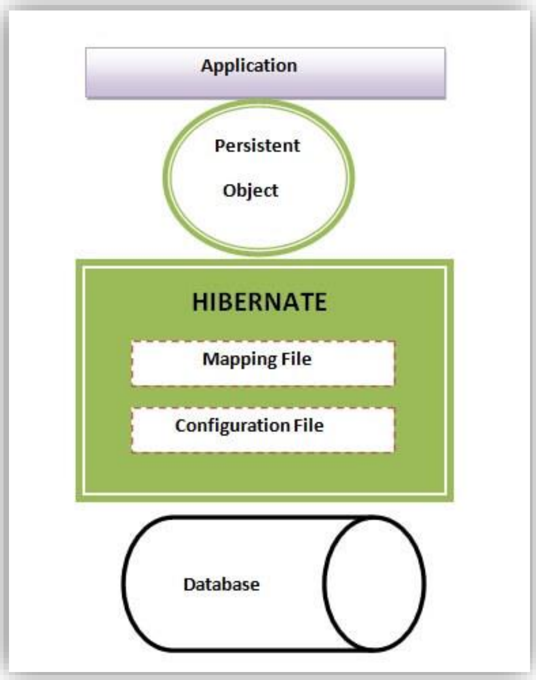
The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

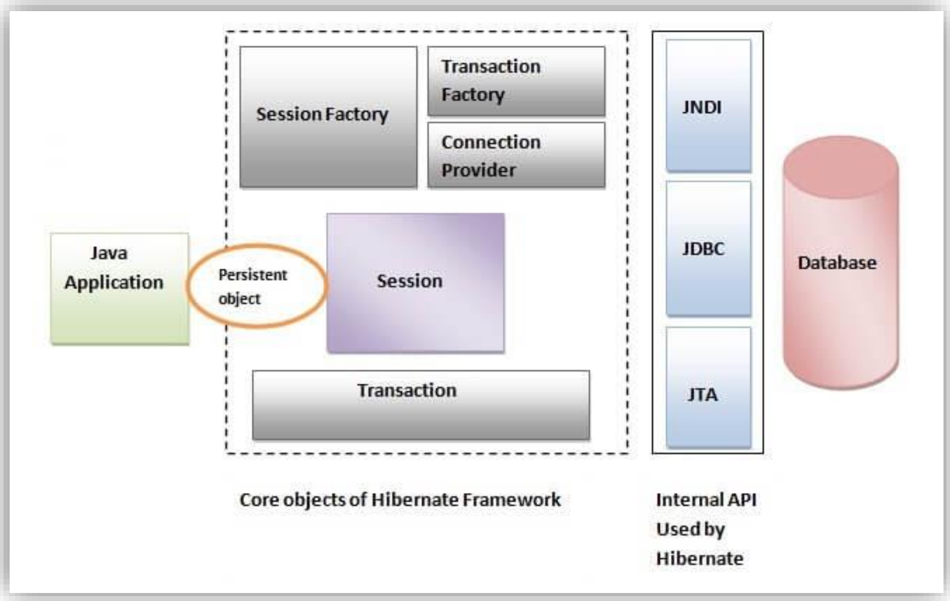
The Hibernate architecture is categorized in four layers:

- Java application layer
- Hibernate framework layer
- Backhand API layer
- Database layer

Let's see the diagram of hibernate architecture:

This is the high-level architecture of Hibernate with mapping file and configuration file.





Hibernate framework uses many objects such as session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

⇒ Elements of Hibernate Architecture:

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

- ***SessionFactory***

  The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

- ***Session***

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

- ***Transaction***

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

- ***ConnectionProvider***

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

- ***TransactionFactory***

  It is a factory of Transaction. It is optional.

➢First Hibernate Example without IDE:

Here, we are going to create the first hibernate application without IDE. For creating the first hibernate application, we need to follow the following steps:

1. Create the Persistent class
2. Create the mapping file for Persistent class
3. Create the Configuration file
4. Create the class that retrieves or stores the persistent object
5. Load the jar file
6. Run the first hibernate application by using command prompt

1) Create the Persistent class

A simple Persistent class should follow some rules:

- **A no-arg constructor:** It is recommended that you have a default constructor at least package visibility so that hibernate can create the instance of the Persistent class by newInstance() method.
- **Provide an identifier property:** It is better to assign an attribute as id. This attribute behaves as a primary key in database.

- **Declare getter and setter methods:** The Hibernate recognizes the method by getter and setter method names by default.
- **Prefer non-final class:** Hibernate uses the concept of proxies, that depends on the persistent class. The application programmer will not be able to use proxies for lazy association fetching.
-

Let's create the simple Persistent class:

Employee.java

```java
package com.javatpoint.mypackage;

public class Employee {
    private int id;
    private String firstName, lastName;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

}
```

2) Create the Mapping file for Persistent class

The mapping file name conventionally, should be class_name.hbm.xml. There are many elements of the mapping file.

- **hibernate-mapping :** It is the root element in the mapping file that contains all the mapping elements.

- **class :** It is the sub-element of the hibernate-mapping element. It specifies the Persistent class.
- **id :** It is the subelement of class. It specifies the primary key attribute in the class.
- **generator :** It is the sub-element of id. It is used to generate the primary key. There are many generator classes such as assigned, increment, hilo, sequence, native etc. We will learn all the generator classes later.
- **property :** It is the sub-element of class that specifies the property name of the Persistent class.

Let's see the mapping file for the Employee class:

employee.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
          "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
          "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

          <hibernate-mapping>
          <class name="com.javatpoint.mypackage.Employee" table="emp1000">
          <id name="id">
          <generator class="assigned"></generator>
          </id>

          <property name="firstName"></property>
          <property name="lastName"></property>

          </class>

          </hibernate-mapping>
```

3) Create the Configuration file

The configuration file contains information about the database and mapping file. Conventionally, its name should be hibernate.cfg.xml .

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>

    <session-factory>
```

```
        <property name="hbm2ddl.auto">update</property>
                                                                      <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
                                                                      <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
                                                                      <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

4) Create the class that retrieves or stores the object

In this class, we are simply storing the employee object to the database.

```
package com.javatpoint.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

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

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

        Employee e1 = new Employee();
        e1.setId(101);
        e1.setFirstName("Gaurav");
        e1.setLastName("Chawla");

        session.save(e1);
        t.commit();
        System.out.println("successfully saved");
        factory.close();
        session.close();
    }
}
```

5) Load the jar file

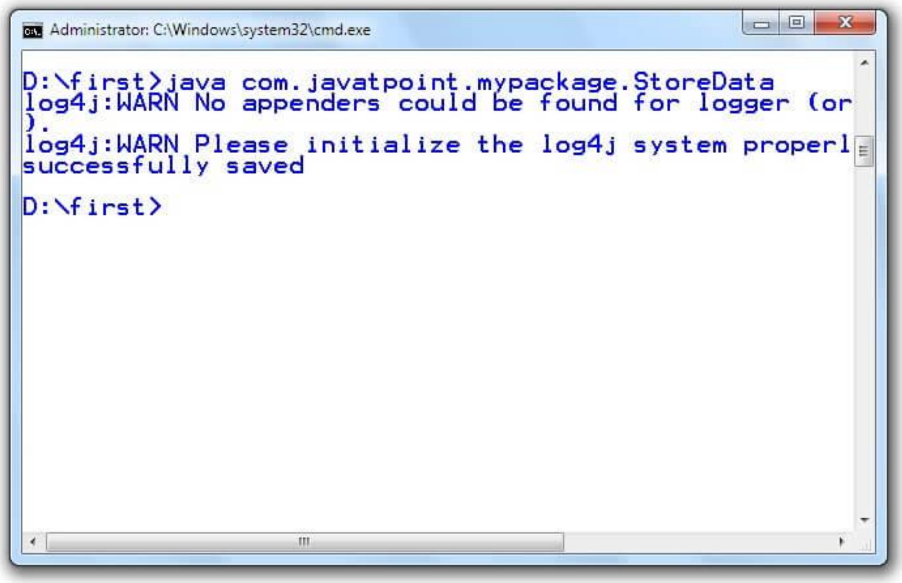
For successfully running the hibernate application, you should have the hibernate5.jar file. **Download the required jar files for hibernate**

6) How to run the first hibernate application without IDE

We may run this hibernate application by IDE (e.g. Eclipse, Myeclipse, Netbeans etc.) or without IDE. We will learn about creating hibernate application in Eclipse IDE in next chapter.

To run the hibernate application without IDE:

- Install the oracle10g for this example.
- Load the jar files for hibernate. (One of the way to load the jar file is copy all the jar files under the JRE/lib/ext folder). It is better to put these jar files inside the public and private JRE both.
- Now, run the StoreData class by **java com.javatpoint.mypackage.StoreData**



```
D:\first>java com.javatpoint.mypackage.StoreData
log4j:WARN No appenders could be found for logger (or
).
log4j:WARN Please initialize the log4j system properl
successfully saved

D:\first>
```

➢ Hibernate Example using XML in Eclipse:

Here, we are going to create a simple example of hibernate application using eclipse IDE. For creating the first hibernate application in Eclipse IDE, we need to follow the following steps:

1. Create the java project
2. Add jar files for hibernate
3. Create the Persistent class
4. Create the mapping file for Persistent class
5. Create the Configuration file
6. Create the class that retrieves or stores the persistent object
7. Run the application

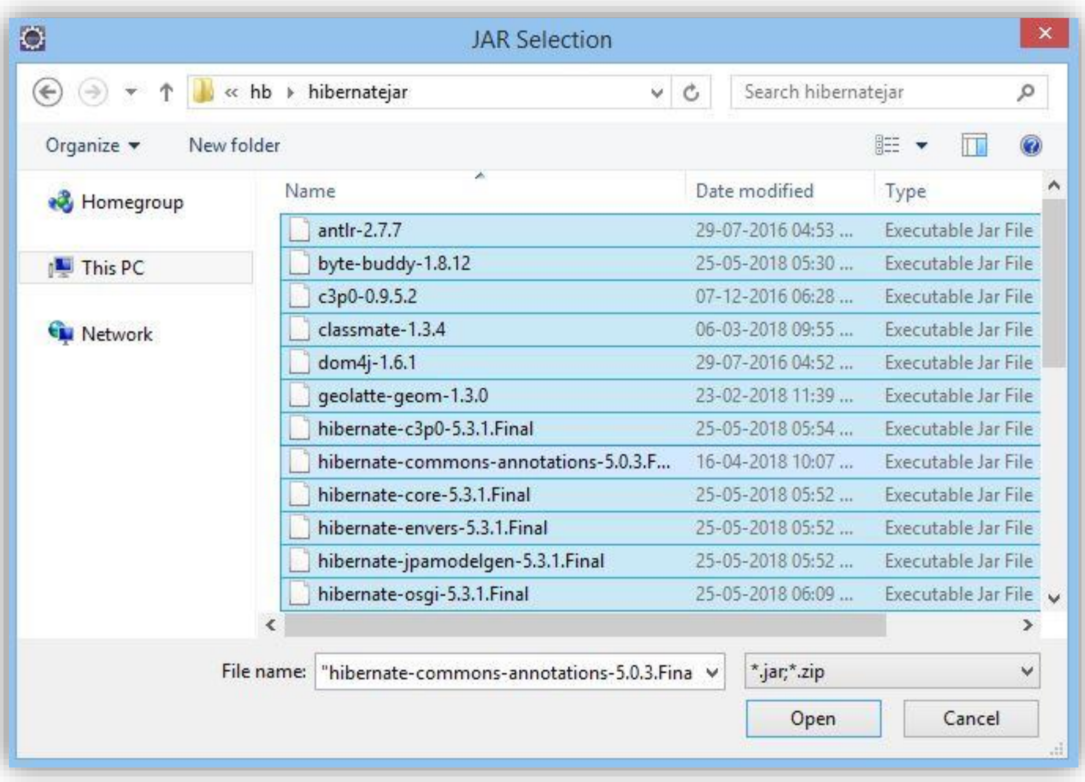
1) Create the java project

Create the java project by **File Menu** - **New** - **project** - **java project** . Now specify the project name e.g. firsthb then **next** - **finish** .

---

2) Add jar files for hibernate

To add the jar files **Right click on your project** - **Build path** - **Add external archives**. Now select all the jar files as shown in the image given below then click open. **Download the required jar file**



Name	Date modified	Type
antlr-2.7.7	29-07-2016 04:53 ...	Executable Jar File
byte-buddy-1.8.12	25-05-2018 05:30 ...	Executable Jar File
c3p0-0.9.5.2	07-12-2016 06:28 ...	Executable Jar File
classmate-1.3.4	06-03-2018 09:55 ...	Executable Jar File
dom4j-1.6.1	29-07-2016 04:52 ...	Executable Jar File
geolatte-geom-1.3.0	23-02-2018 11:39 ...	Executable Jar File
hibernate-c3p0-5.3.1.Final	25-05-2018 05:54 ...	Executable Jar File
hibernate-commons-annotations-5.0.3.F...	16-04-2018 10:07 ...	Executable Jar File
hibernate-core-5.3.1.Final	25-05-2018 05:52 ...	Executable Jar File
hibernate-envers-5.3.1.Final	25-05-2018 05:52 ...	Executable Jar File
hibernate-jpamodelgen-5.3.1.Final	25-05-2018 05:52 ...	Executable Jar File
hibernate-osgi-5.3.1.Final	25-05-2018 06:09 ...	Executable Jar File

File name: "hibernate-commons-annotations-5.0.3.Fina

.jar;.zip

In this example, we are connecting the application with oracle database. So you must add the ojdbc14.jar file. **download the ojdbc14.jar file**

3) Create the Persistent class

Here, we are creating the same persistent class which we have created in the previous topic. To create the persistent class, Right click on **src** - **New** - **Class** - specify the class with package name (e.g. com.javatpoint.mypackage) - **finish** .

Employee.java

```
package com.javatpoint.mypackage;

public class Employee {
    private int id;
    private String firstName, lastName;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

}
```

4) Create the mapping file for Persistent class

Here, we are creating the same mapping file as created in the previous topic. To create the mapping file, Right click on **src** - **new** - **file** - specify the file name (e.g. employee.hbm.xml) - **ok**. It must be outside the package.

employee.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE  hibernate-mapping  PUBLIC  "-//Hibernate/Hibernate  Mapping  DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.mypackage.Employee" table="emp1000">
        <id name="id">
            <generator class="assigned"></generator>
        </id>

        <property name="firstName"></property>
        <property name="lastName"></property>

    </class>

</hibernate-mapping>
```

5) **Create the Configuration file**

The configuration file contains all the information for the database such as connection_url, driver_class, username, password etc. The hbm2ddl.auto property is used to create the table in the database automatically. We will have in-depth learning about Dialect class in next topics. To create the configuration file, right click on src - new - file. Now specify the configuration file name e.g. hibernate.cfg.xml.

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
DTD     5.3//EN"     "http://hibernate.sourceforge.net/hibernate-configuration-
5.3.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
                                                                     <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
                                                                     <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
                                                                     <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
```

```
        <mapping resource="employee.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

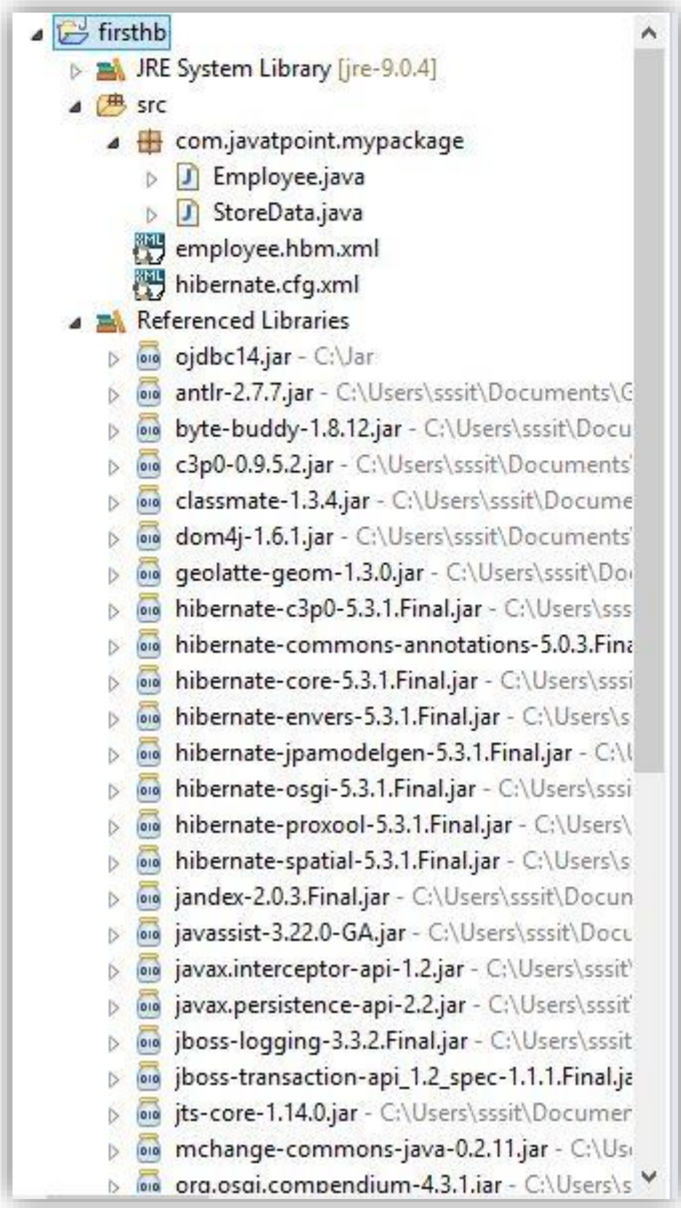
6) Create the class that retrieves or stores the persistent object

In this class, we are simply storing the employee object to the database.

```
package com.javatpoint.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

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

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

        Employee e1 = new Employee();
        e1.setId(101);
        e1.setFirstName("Gaurav");
        e1.setLastName("Chawla");

        session.save(e1);
        t.commit();
        System.out.println("successfully saved");
        factory.close();
        session.close();

    }
}
```

7) Run the Application

Before running the application, determine that directory structure is like this.



```
firsthb
  JRE System Library [jre-9.0.4]
  src
    com.javatpoint.mypackage
      Employee.java
      StoreData.java
    employee.hbm.xml
    hibernate.cfg.xml
  Referenced Libraries
    ojdbc14.jar - C:\Jar
    antlr-2.7.7.jar - C:\Users\sssit\Documents\G
    byte-buddy-1.8.12.jar - C:\Users\sssit\Docu
    c3p0-0.9.5.2.jar - C:\Users\sssit\Documents
    classmate-1.3.4.jar - C:\Users\sssit\Docume
    dom4j-1.6.1.jar - C:\Users\sssit\Documents
    geolatte-geom-1.3.0.jar - C:\Users\sssit\Do
    hibernate-c3p0-5.3.1.Final.jar - C:\Users\sss
    hibernate-commons-annotations-5.0.3.Fin
    hibernate-core-5.3.1.Final.jar - C:\Users\sssi
    hibernate-envers-5.3.1.Final.jar - C:\Users\s
    hibernate-jpamodelgen-5.3.1.Final.jar - C:\
    hibernate-osgi-5.3.1.Final.jar - C:\Users\sssi
    hibernate-proxool-5.3.1.Final.jar - C:\Users\
    hibernate-spatial-5.3.1.Final.jar - C:\Users\s
    jandex-2.0.3.Final.jar - C:\Users\sssit\Docun
    javassist-3.22.0-GA.jar - C:\Users\sssit\Docu
    javax.interceptor-api-1.2.jar - C:\Users\sssit
    javax.persistence-api-2.2.jar - C:\Users\sssit
    jboss-logging-3.3.2.Final.jar - C:\Users\sssit
    jboss-transaction-api_1.2_spec-1.1.1.Final.ja
    jts-core-1.14.0.jar - C:\Users\sssit\Documer
    mchange-commons-java-0.2.11.jar - C:\Us
    org.osgi.compendium-4.3.1.jar - C:\Users\s
```

To run the hibernate application, right click on the StoreData class - Run As - Java Application.

➢ Hibernate Example using Annotation in Eclipse:

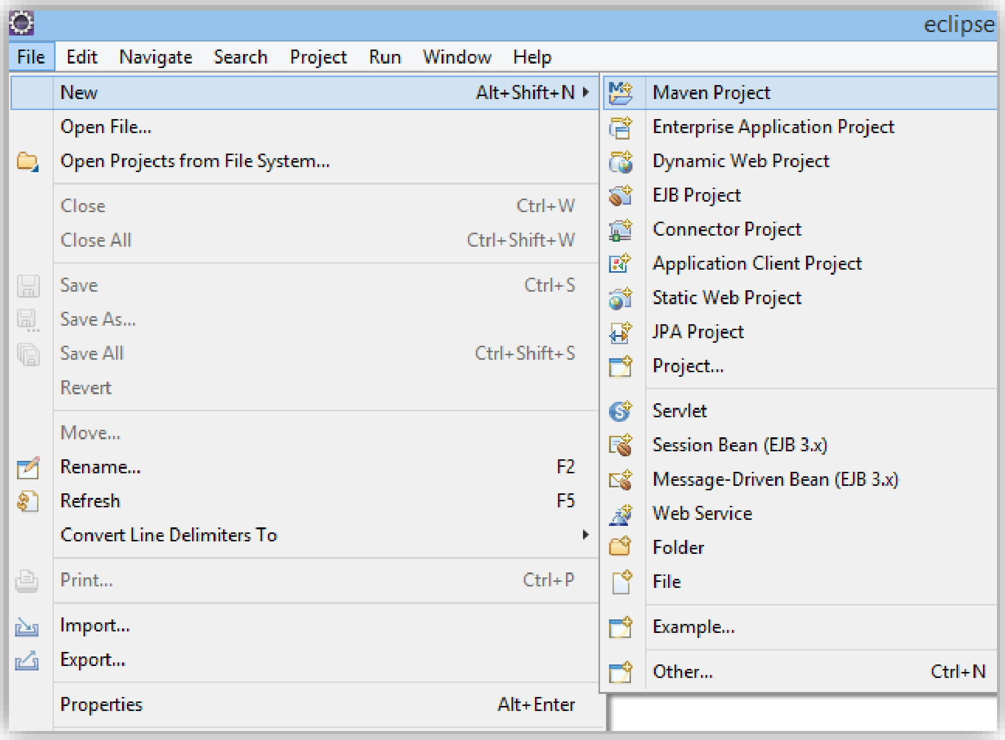
The hibernate application can be created with annotation. There are many annotations that can be used to create hibernate application such as @Entity, @Id, @Table etc.
Hibernate Annotations are based on the JPA 2 specification and supports all the features.
All the JPA annotations are defined in the **javax.persistence** package. Hibernate EntityManager implements the interfaces and life cycle defined by the JPA specification.
The core advantage of using hibernate annotation is that you don't need to create mapping (hbm) file. Here, hibernate annotations are used to provide the meta data.

❖ *Example to create the hibernate application with Annotation:*

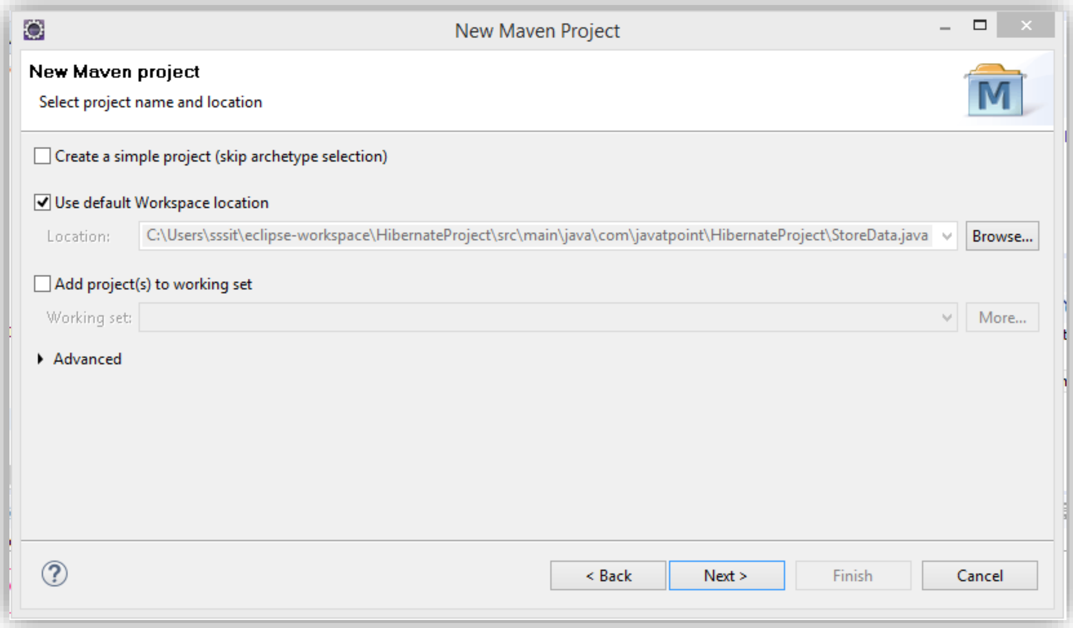
Here, we are going to create a maven based hibernate application using annotation in eclipse IDE. For creating the hibernate application in Eclipse IDE, we need to follow the below steps:

1) Create the Maven Project

- To create the maven project left click on **File Menu** - **New**- **Maven Project**.



- The new maven project opens in your eclipse. **Click Next**.



New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

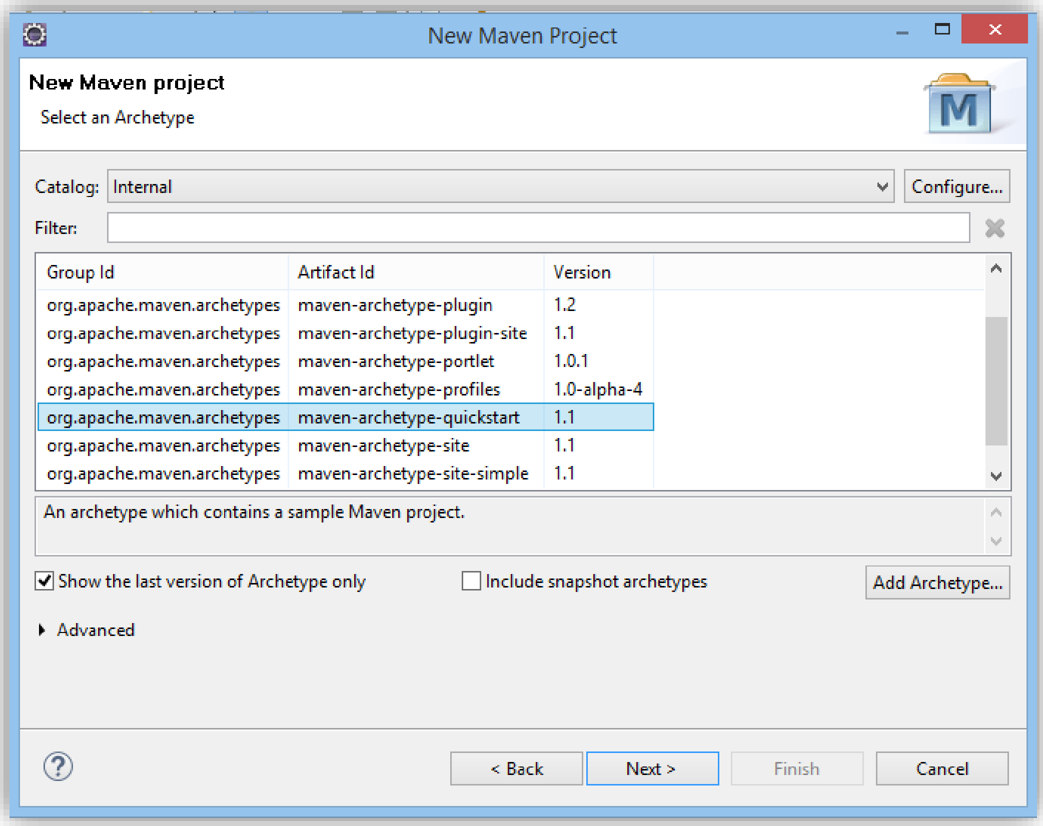
☑ Use default Workspace location

Location: C:\Users\sssit\eclipse-workspace\HibernateProject\src\main\java\com\javatpoint\HibernateProject\StoreData.java

☐ Add project(s) to working set

Working set:

- Now, select catalog type: internal and maven archetype - **quickstart** of 1.1 version. Then, **click next**.



New Maven project

Select an Archetype

Catalog: Internal

Filter:

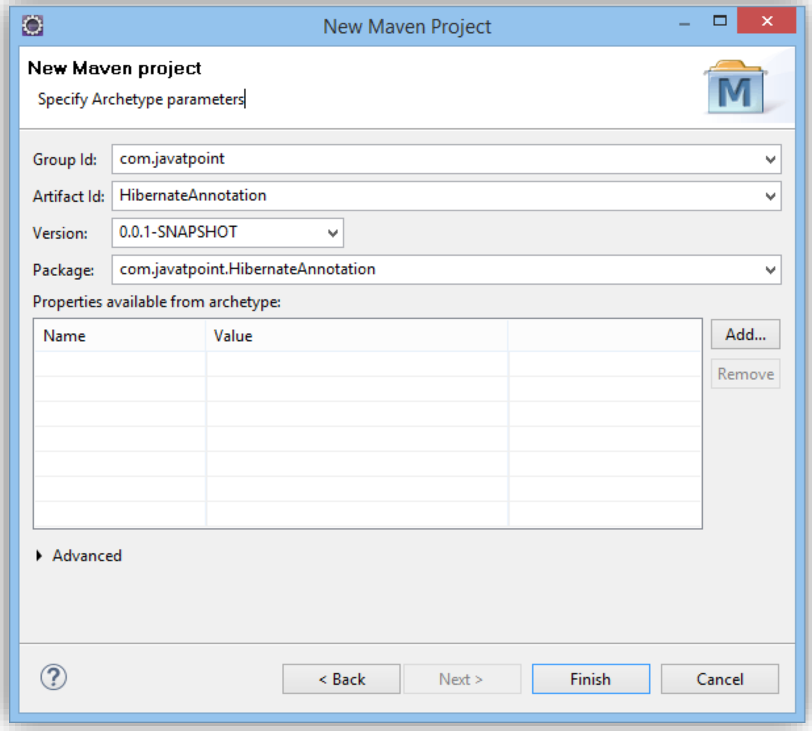
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1

An archetype which contains a sample Maven project.

☑ Show the last version of Archetype only

☐ Include snapshot archetypes

- Now, specify the name of Group Id and Artifact Id. The Group Id contains package name (e.g. com.javatpoint) and Artifact Id contains project name (e.g. HibernateAnnotation). Then **click Finish**.


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.javatpoint

Artifact Id: HibernateAnnotation

Version: 0.0.1-SNAPSHOT

Package: com.javatpoint.HibernateAnnotation

Properties available from archetype:

Name	Value

2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag. These dependencies are used to add the jar files in Maven project.

```
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.3.1.Final</version>
    </dependency>


    <dependency>
      <groupId>com.oracle</groupId>
      <artifactId>ojdbc14</artifactId>
      <version>10.2.0.4.0</version>
    </dependency>
```

Due to certain license issues, Oracle drivers are not present in public Maven repository. We can install it manually. To install Oracle driver into your local Maven repository, follow the following steps:

- **Install Maven**
- Run the command : install-file -Dfile=Path/to/your/ojdbc14.jar -DgroupId=com.oracle -DartifactId=ojdbc14 -Dversion=12.1.0 -Dpackaging=jar

3) Create the Persistence class.

Here, we are creating the same persistent class which we have created in the previous topic. But here, we are using annotation.
@Entity annotation marks this class as an entity.
@Table annotation specifies the table name where data of this entity is to be persisted. If you don't use @Table annotation, hibernate will use the class name as the table name by default.
@Id annotation marks the identifier for this entity.
@Column annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name by default.
To create the Persistence class, right click on **src/main/java - New - Class -** specify the class name with package - **finish**.

Employee.java

```java
package com.javatpoint.mypackage;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name= "emp500")
public class Employee {

@Id
private int id;
private String firstName,lastName;

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

}
```

4) Create the Configuration file

To create the configuration file, right click on **src/main/java - new - file -** specify the file name (e.g. hibernate.cfg.xml) - **Finish**.

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="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <mapping class="com.javatpoint.mypackage.Employee" />
    </session-factory>
</hibernate-configuration>
```

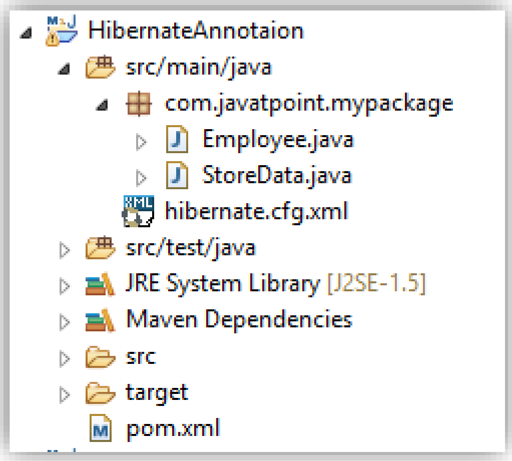
5) Create the class that retrieves or stores the persistent object.

StoreData.java

```
package com.javatpoint.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;


public class StoreData {
public static void main(String[] args) {

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

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

    Employee e1=new Employee();
    e1.setId(101);
    e1.setFirstName("Gaurav");
    e1.setLastName("Chawla");

    session.save(e1);
    t.commit();
    System.out.println("successfully saved");
    factory.close();
    session.close();

}
}
```

6) Run the application

Before running the application, determine that the directory structure is like this.



```
HibernateAnnotaion
  src/main/java
    com.javatpoint.mypackage
      Employee.java
      StoreData.java
    hibernate.cfg.xml
  src/test/java
  JRE System Library [J2SE-1.5]
  Maven Dependencies
  src
  target
  pom.xml
```

To run the hibernate application, right click on the **StoreData - Run As - Java Application**.

➢ Web Application with Hibernate (using XML):

Here, we are going to create a web application with hibernate. For creating the web application, we are using JSP for presentation logic, Bean class for representing data and DAO class for database codes.
As we create the simple application in hibernate, we don't need to perform any extra operations in hibernate for creating web application. In such case, we are getting the value from the user using the JSP file.

Example to create web application using hibernate

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

index.jsp

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

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

</form>
```

register.jsp

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

```
<%@page import="com.javatpoint.mypack.UserDao" %>
    <jsp:useBean id="obj" class="com.javatpoint.mypack.User"></jsp:useBean>
    <jsp:setProperty property="*" name="obj" />

    <% int i=UserDao.register(obj); if(i>0)
        out.print("You are successfully registered");
      %>
```

User.java

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

```
package com.javatpoint.mypack;

public class User {
    private int id;
    private String name, password, email;
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

}
```

user.hbm.xml

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

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE   hibernate-mapping   PUBLIC   "-//Hibernate/Hibernate   Mapping   DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.mypack.User" table="u400">
        <id name="id">
            <generator class="increment"></generator>
        </id>
        <property name="name"></property>
        <property name="password"></property>
        <property name="email"></property>
    </class>

</hibernate-mapping>
```

UserDao.java

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

```
package com.javatpoint.mypack;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class UserDao {

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

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

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

        i = (Integer) session.save(u);

        t.commit();
        session.close();

        return i;

    }
}
```

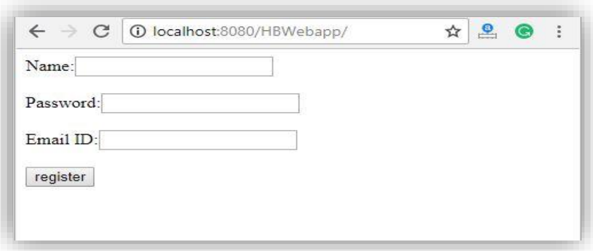
hibernate.cfg.xml

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

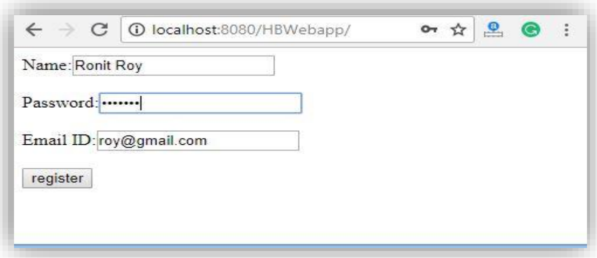
```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
DTD     5.3//EN"     "http://hibernate.sourceforge.net/hibernate-configuration-
5.3.dtd">

<!-- Generated by MyEclipse Hibernate Tools. -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
                                                                        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
                                                                        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">oracle</property>
                                                                        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="user.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

➢ Output:



```
localhost:8080/HBWebapp/
Name:
Password:
Email ID:
register
```



```
localhost:8080/HBWebapp/
Name: Ronit Roy
Password: •••••••
Email ID: roy@gmail.com
register
```

Download

➢Collection Mapping in Hibernate:

We can map collection elements of Persistent class in Hibernate. You need to declare the type of collection in Persistent class from one of the following types:

- java.util.List
- java.util.Set
- java.util.SortedSet
- java.util.Map
- java.util.SortedMap
- java.util.Collection

or write the implementation of org.hibernate.usertype.UserCollectionType

The persistent class should be defined like this for collection element.

```
package com.javatpoint;

import java.util.List;

public class Question {
    private int id;
    private String qname;
    private List<String> answers;

      //getters and setters

}
```

⇒Mapping collection in mapping file:

There are many subelements of **<class>** elements to map the collection. They are **<list>**, **<bag>**, **<set>** and **<map>**. Let's see how we implement the list for the above class:

```
  <class name="com.javatpoint.Question" table="q100">
    <id name="id">
      <generator class="increment"></generator>
    </id>
    <property name="qname"></property>

    <list name="answers" table="ans100">
      <key column="qid"></key>
      <index column="type"></index>
      <element column="answer" type="string"></element>
    </list>
  </class>
```

There are three subelements used in the list:

- **<key>** element is used to define the foreign key in this table based on the Question class identifier.
- **<index>** element is used to identify the type. List and Map are indexed collection.
- **<element>** is used to define the element of the collection.

This is the mapping of collection if collection stores string objects. But if collection stores entity reference (another class objects), we need to define **<one-to-many>** or **<many-to-many>** element. Now the Persistent class will look like:

```
package com.javatpoint;2. 3. import java.util.List;4. 5.

public class Question {

    private int id;7.
    private String qname;8.
    private List<Answer> answers; // Here, List stores the objects of Answer
class

      // getters and setters

}
```

```
package com.javatpoint;

import java.util.List;

public class Answer {
        private int id;
        private String answer;
        private String posterName;

      //getters and setters

}
```

Now the mapping file will be:

```
<class name="com.javatpoint.Question" table="q100">
    <id name="id">
        <generator class="increment"></generator>
    </id>
    <property name="qname"></property>
    <list name="answers">
        <key column="qid"></key>
        <index column="type"></index>
        <one-to-many class="com.javatpoint.Answer" />
    </list>
</class>
```

Here, List is mapped by one-to-many relation. In this scenario, there can be many answers for one question.

⇒ Understanding key element:

The key element is used to define the foreign key in the joined table based on the original identity. The foreign key element is nullable by default. So for non-nullable foreign key, we need to specify not-null attribute such as:

1. <key column="qid" not-**null**="true" ></key>
   The attributes of the key element are column, on-delete, property-ref, not-null, update and unique.

```
<key column="columnname"
on-delete="noaction|cascade"
not-null="true|false"
property-ref="propertyName"
update="true|false"
unique="true|false" />
```

⇒ Indexed collections:

The collection elements can be categorized in two forms:

- **indexed** ,and
- **non-indexed**

The List and Map collection are indexed whereas set and bag collections are non-indexed. Here, indexed collection means List and Map requires an additional element **<index>**.

⇒ Collection Elements:

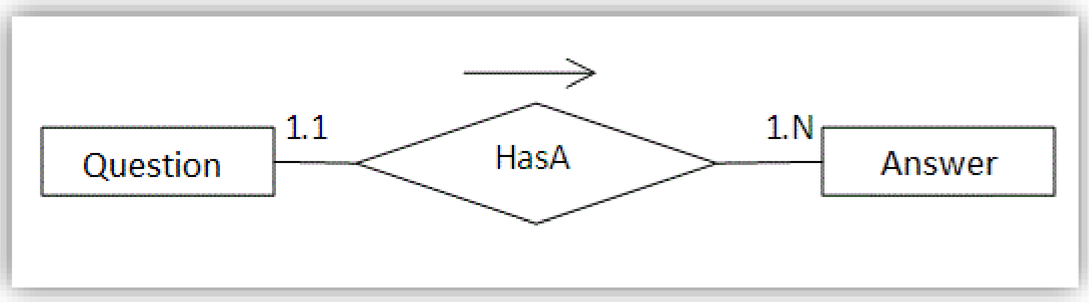
The collection elements can have value or entity reference (another class object). We can use one of the 4 elements

- **element**
- **component-element**
- **one-to-many**, or
- **many-to-many**

The element and component-element are used for normal value such as string, int etc. whereas one-to-many and many-to-many are used to map entity reference.

➢ Mapping List in Collection Mapping (using xml file):

If our persistent class has List object, we can map the List easily either by <list> element of class in mapping file or by annotation.
Here, we are using the scenario of Forum where one question has multiple answers.



Let's see how we can implement the list in the mapping file:

```
<class name="com.javatpoint.Question" table="q100">
    ...
    <list name="answers" table="ans100">
        <key column="qid"></key>
        <index column="type"></index>
        <element column="answer" type="string"></element>
    </list>
    ...
</class>
```

List and Map are index-based collection, so an extra column will be created in the table for indexing.

⇒Example of mapping list in collection mapping:

In this example, we are going to see full example of collection mapping by list. This is the example of List that stores string value not entity reference that is why we are going to use **element** instead of **one-to-many** within the list element.

1) Create the Persistent class

This persistent class defines properties of the class including List.

```
package com.javatpoint;

import java.util.List;

public class Question {
    private int id;
    private String qname;
```

```java
    private List<String> answers;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getQname() {
        return qname;
    }

    public void setQname(String qname) {
        this.qname = qname;
    }

    public List<String> getAnswers() {
        return answers;
    }

    public void setAnswers(List<String> answers) {
        this.answers = answers;
    }
}
```

2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
  <class name="com.javatpoint.Question" table="q100">
    <id name="id">
      <generator class="increment"></generator>
    </id>
    <property name="qname"></property>

    <list name="answers" table="ans100">
      <key column="qid"></key>
      <index column="type"></index>
      <element column="answer" type="string"></element>
    </list>
  </class>
</hibernate-mapping>
```

3) Create the configuration file

This file contains information about the database and mapping file.

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

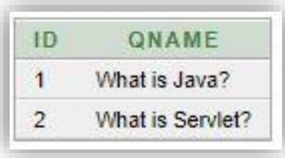
```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="question.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

4) Create the class to store the data

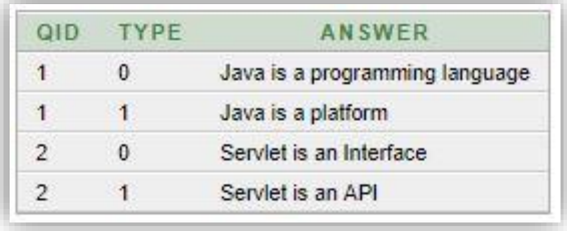
In this class we are storing the data of the question class.

```
package com.javatpoint;

import java.util.ArrayList;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
  public static void main(String[] args) {

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

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

    Transaction t = session.beginTransaction();

    ArrayList<String> list1 = new ArrayList<String>();
    list1.add("java is a programming language");
    list1.add("java is a platform");

    ArrayList<String> list2 = new ArrayList<String>();
    list2.add("Servlet is an Interface");
    list2.add("Servlet is an API");

    Question question1 = new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(list1);

    Question question2 = new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(list2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");
  }
}
```

⇒ **Output:**



ID	QNAME
1	What is Java?
2	What is Servlet?



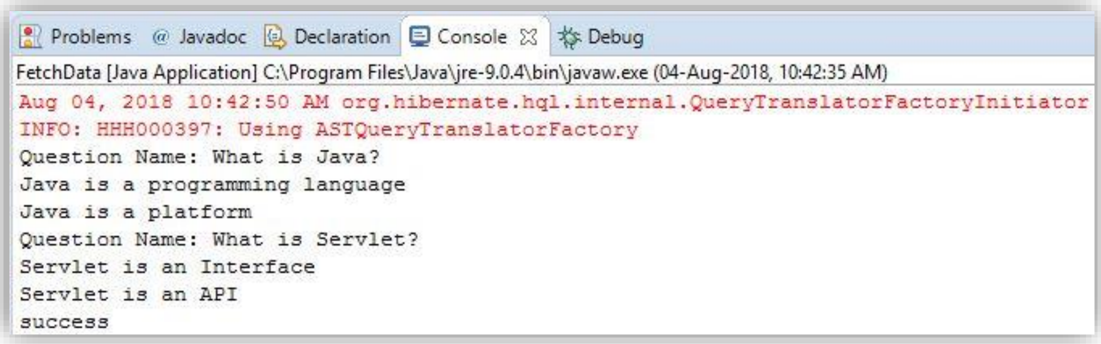
QID	TYPE	ANSWER
1	0	Java is a programming language
1	1	Java is a platform
2	0	Servlet is an Interface
2	1	Servlet is an API

⇒ How to fetch the data of List?

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent.

```
package com.javatpoint;

import javax.persistence.TypedQuery;
import java.util.*;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchData {
    public static void main(String[] args) {

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

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

        TypedQuery query = session.createQuery("from Question");
        List<Question> list = query.getResultList();

        Iterator<Question> itr = list.iterator();
        while (itr.hasNext()) {
            Question q = itr.next();
            System.out.println("Question Name: " + q.getQname());

            // printing answers
            List<String> list2 = q.getAnswers();
            Iterator<String> itr2 = list2.iterator();
            while (itr2.hasNext()) {
                System.out.println(itr2.next());
            }

        }
        session.close();
        System.out.println("success");

    }
}
```

⇒ **Output**:

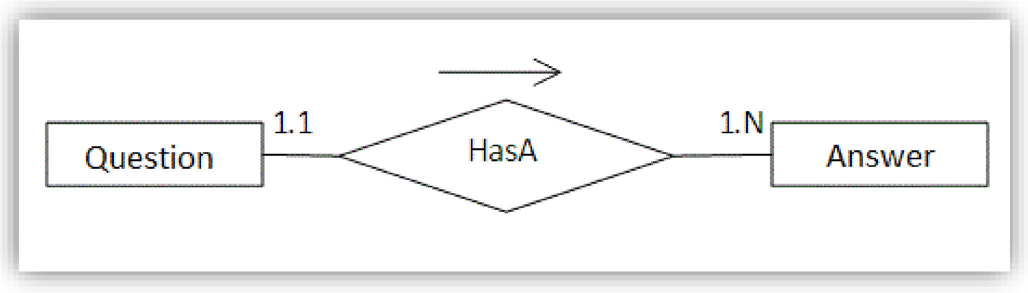


```
FetchData [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (04-Aug-2018, 10:42:35 AM)
Aug 04, 2018 10:42:50 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
Question Name: What is Java?
Java is a programming language
Java is a platform
Question Name: What is Servlet?
Servlet is an Interface
Servlet is an API
success
```

Download

➢ Mapping Bag in Collection Mapping (using xml file):

If our persistent class has List object, we can map the List by list or bag element in the mapping file. The bag is just like List but it doesn't require index element.
Here, we are using the scenario of Forum where one question has multiple answers.



Let's see how we can implement the bag in the mapping file:

```
<class name="com.javatpoint.Question" table="q100">
    ...
    <bag name="answers" table="ans100">
        <key column="qid"></key>
        <element column="answer" type="string"></element>
    </bag>
    ...
</class>
```

⇒ *Example of mapping bag in collection mapping:*

In this example, we are going to see full example of collection mapping by bag. This is the example of bag if it stores value not entity reference that is why are going to use **element** instead of **one-to-many**. If you have seen the example of mapping list, it is same in all cases instead mapping file where we are using bag instead of list.

1) Create the Persistent class

This persistent class defines properties of the class including List.

```
package com.javatpoint;

import java.util.List;

public class Question {
    private int id;
    private String qname;
    private List<String> answers;

      //getters and setters
}
```

2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
  <class name="com.javatpoint.Question" table="q101">
    <id name="id">
      <generator class="increment"></generator>
    </id>
    <property name="qname"></property>

    <bag name="answers" table="ans101">
      <key column="qid"></key>
      <element column="answer" type="string"></element>
    </bag>

  </class>

</hibernate-mapping>
```

3) Create the configuration file

This file contains information about the database and mapping file.

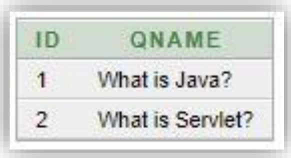
```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="question.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

4) Create the class to store the data

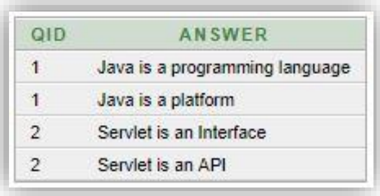
In this class we are storing the data of the question class.

```
package com.javatpoint;

import java.util.ArrayList;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
 public static void main(String[] args) {

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

     SessionFactory factory=meta.buildSessionFactory();
     Session session=factory.openSession();

    Transaction t=session.beginTransaction();

    ArrayList<String> list1=new ArrayList<String>();
    list1.add("Java is a programming language");
    list1.add("Java is a platform");

    ArrayList<String> list2=new ArrayList<String>();
    list2.add("Servlet is an Interface");
    list2.add("Servlet is an API");

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(list1);

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(list2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");
 }
}
```

⇒ **Output:**



ID	QNAME
1	What is Java?
2	What is Servlet?



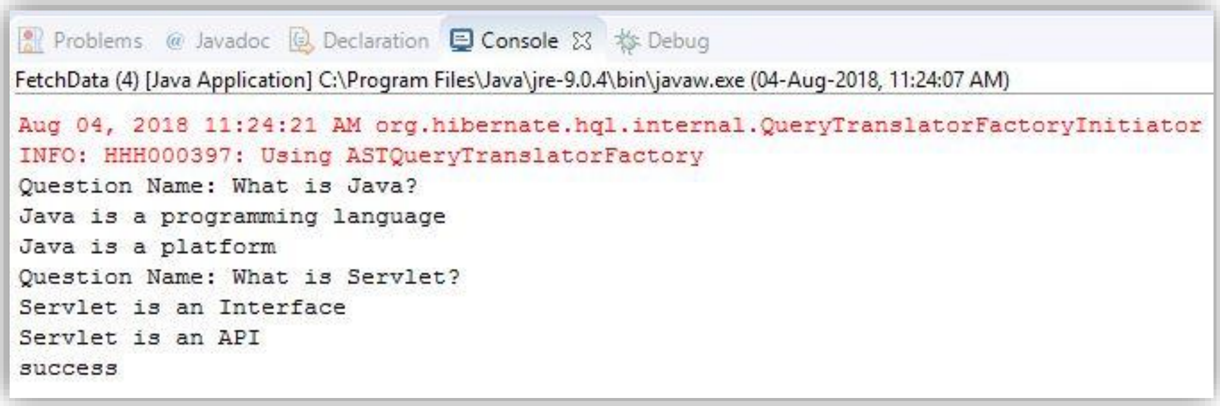
QID	ANSWER
1	Java is a programming language
1	Java is a platform
2	Servlet is an Interface
2	Servlet is an API

⇒How to fetch the data?

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent.

```
 package com.javatpoint;

import java.util.*;

import javax.persistence.TypedQuery;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;


public class FetchData {
public static void main(String[] args) {

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

     SessionFactory factory=meta.buildSessionFactory();
     Session session=factory.openSession();

    TypedQuery query=session.createQuery("from Question");
    List<Question> list=query.getResultList();

    Iterator<Question> itr=list.iterator();
    while(itr.hasNext()){
        Question q=itr.next();
        System.out.println("Question Name: "+q.getQname());

        //printing answers
        List<String> list2=q.getAnswers();
        Iterator<String> itr2=list2.iterator();
        while(itr2.hasNext()){
            System.out.println(itr2.next());
        }

    }
    session.close();
    System.out.println("success");

}
}
```

⇒ <u>Output</u>:



```
FetchData (4) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (04-Aug-2018, 11:24:07 AM)
Aug 04, 2018 11:24:21 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
Question Name: What is Java?
Java is a programming language
Java is a platform
Question Name: What is Servlet?
Servlet is an Interface
Servlet is an API
success
```

Download

➢Hibernate Mapping Set using XML:

If our persistent class has Set object, we can map the Set by set element in the mapping file. The set element doesn't require index element. The one difference between List and Set is that, it stores only unique values.
Let's see how we can implement the set in the mapping file:

```
<class name="com.javatpoint.Question" table="q1002">
    ...
    <set name="answers" table="ans1002">
        <key column="qid"></key>
        <element column="answer" type="string"></element>
    </set>
    ...
</class>
```

⇒*Example of mapping set in collection mapping*:

In this example, we are going to see full example of collection mapping by set. This is the example of set that stores value not entity reference that is why are going to use element instead of one-to-many.

1) Create the Persistent class

This persistent class defines properties of the class including Set.

```
package com.javatpoint;

import java.util.Set;

public class Question {
private int id;
private String qname;
private Set<String> answers;

//getters and setters

}
```

2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.Question" table="q1002">
        <id name="id">
            <generator class="increment"></generator>
        </id>
        <property name="qname"></property>

        <set name="answers" table="ans1002">
            <key column="qid"></key>
            <element column="answer" type="string"></element>
        </set>

    </class>

</hibernate-mapping>
```

3) Create the configuration file

This file contains information about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="question.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

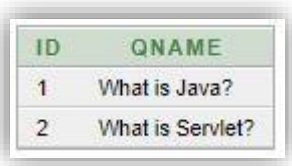
4) Create the class to store the data

In this class we are storing the data of the question class.

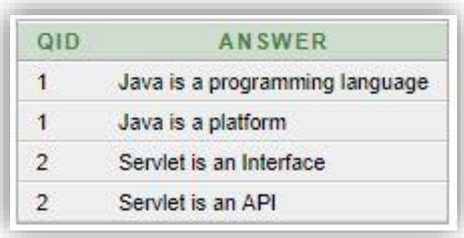
```
package com.javatpoint;

import java.util.HashSet;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
 public static void main(String[] args) {

     StandardServiceRegistry ssr=new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
       Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();

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

 Transaction t=session.beginTransaction();


    HashSet<String> set1=new HashSet<String>();
    set1.add("Java is a programming language");
    set1.add("Java is a platform");

    HashSet<String> set2=new HashSet<String>();
    set2.add("Servlet is an Interface");
    set2.add("Servlet is an API");

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(set1);

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(set2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");
 }
}
```

⇒ **Output:**



ID	QNAME
1	What is Java?
2	What is Servlet?



QID	ANSWER
1	Java is a programming language
1	Java is a platform
2	Servlet is an Interface
2	Servlet is an API

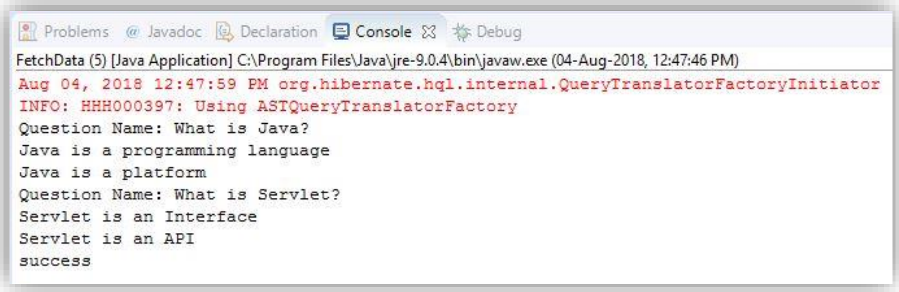
⇒How to fetch the data of Set?

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent.

```
package com.javatpoint;

import java.util.*;

import javax.persistence.TypedQuery;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class FetchData
{
       Public static void main(String[] args)
    {

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

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

        Transaction t = session.beginTransaction();

        TypedQuery query = session.createQuery("from Question");
        List<Question> list = query.getResultList();

        Iterator<Question> itr = list.iterator();
        while (itr.hasNext())
        {
            Question q = itr.next();
            System.out.println("Question Name: " + q.getQname());

            // printing answers
            Set<String> set = q.getAnswers();
            Iterator<String> itr2 = set.iterator();
            while (itr2.hasNext())
            {
                System.out.println(itr2.next());
            }
        }
        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



```
FetchData (5) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (04-Aug-2018, 12:47:46 PM)
Aug 04, 2018 12:47:59 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
Question Name: What is Java?
Java is a programming language
Java is a platform
Question Name: What is Servlet?
Servlet is an Interface
Servlet is an API
success
```

Download

➢Hibernate Mapping Map using xml file:

Hibernate allows you to map Map elements with the RDBMS. As we know, list and map are index-based collections. In case of map, index column works as the key and element column works as the value.

⇒*Example of Mapping Map in collection mapping using xml file*:

You need to create following pages for mapping map elements.

- Question.java
- question.hbm.xml
- hibernate.cfg.xml
- StoreTest.java
- FetchTest.java

Question.java

```
package com.javatpoint;

import java.util.Map;

public class Question {
private int id;
private String name,username;
private Map<String,String> answers;

public Question() {}
public Question(String name, String username, Map<String, String> answers) {
    super();
    this.name = name;
    this.username = username;
    this.answers = answers;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}
```

```java
public void setName(String name) {
    this.name = name;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public Map<String, String> getAnswers() {
    return answers;
}

public void setAnswers(Map<String, String> answers) {
    this.answers = answers;
}
}
```

question.hbm.xml

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>

    <class name="com.javatpoint.Question" table="question736">
        <id name="id">
            <generator class="native"></generator>
        </id>
        <property name="name"></property>
        <property name="username"></property>

        <map name="answers" table="answer736" cascade="all">
            <key column="questionid"></key>
            <index column="answer" type="string"></index>
            <element column="username" type="string"></element>
        </map>
    </class>

</hibernate-mapping>
```

hibernate.cfg.xml

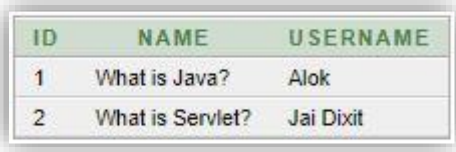
```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>
```

```
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <mapping resource="question.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

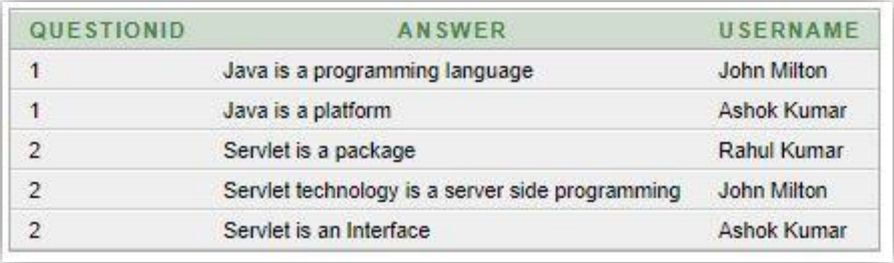
StoreTest.java

```
package com.javatpoint;

import java.util.HashMap;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;


public class StoreTest {
public static void main(String[] args) {

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

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

Transaction t=session.beginTransaction();

HashMap<String,String> map1=new HashMap<String,String>();
map1.put("Java is a programming language","John Milton");
map1.put("Java is a platform","Ashok Kumar");

HashMap<String,String> map2=new HashMap<String,String>();
map2.put("Servlet technology is a server side programming","John Milton");
map2.put("Servlet is an Interface","Ashok Kumar");
map2.put("Servlet is a package","Rahul Kumar");

Question question1=new Question("What is Java?","Alok",map1);
Question question2=new Question("What is Servlet?","Jai Dixit",map2);

session.persist(question1);
session.persist(question2);

t.commit();
session.close();
System.out.println("successfully stored");
}
}
```

⇒ **Output:**



ID	NAME	USERNAME
1	What is Java?	Alok
2	What is Servlet?	Jai Dixit



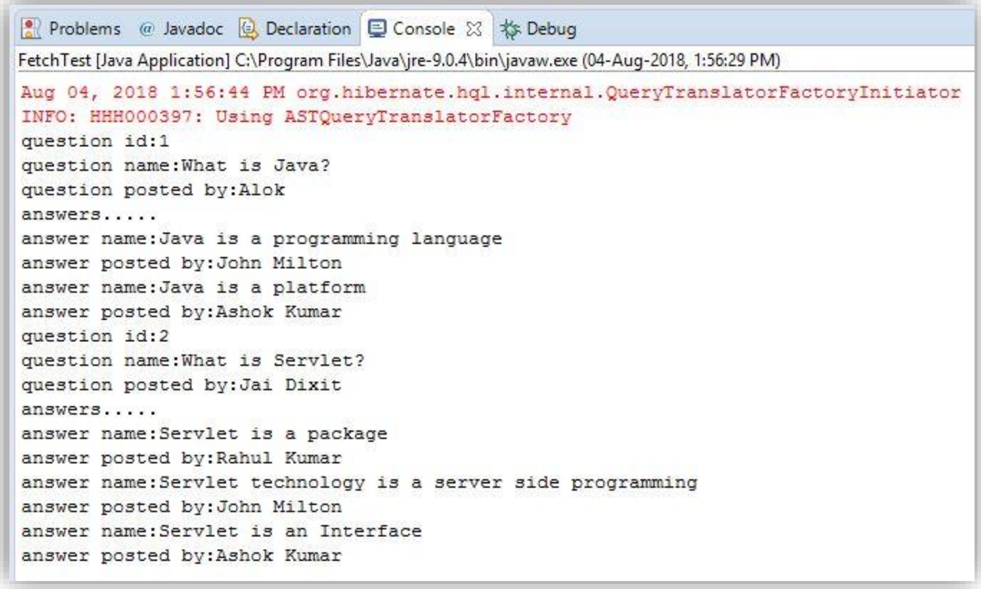
QUESTIONID	ANSWER	USERNAME
1	Java is a programming language	John Milton
1	Java is a platform	Ashok Kumar
2	Servlet is a package	Rahul Kumar
2	Servlet technology is a server side programming	John Milton
2	Servlet is an Interface	Ashok Kumar

FetchTest.java

```
package com.javatpoint;
import java.util.*;

import javax.persistence.TypedQuery;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchTest {
public static void main(String[] args) {
    StandardServiceRegistry ssr=new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
    Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();

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

 TypedQuery query=session.createQuery("from Question ");
 List<Question>
    list=query.getResultList();

 Iterator
    <Question>
        iterator=list.iterator();
 while(iterator.hasNext()){
  Question question=iterator.next();
  System.out.println("question id:"+question.getId());
  System.out.println("question name:"+question.getName());
  System.out.println("question posted by:"+question.getUsername());
  System.out.println("answers.....");
  Map
        <String ,String=>
```

```
            map=question.getAnswers();
  Set
            <Map.Entry
                <String ,String=>
                    > set=map.entrySet();

  Iterator
                    <Map.Entry
                        <String ,String=>
                            > iteratoranswer=set.iterator();
  while(iteratoranswer.hasNext()){
   Map.Entry
                            <String ,String=>
                                entry=(Map.Entry
                                <String ,String=>)iteratoranswer.next();
   System.out.println("answer name:"+entry.getKey());
   System.out.println("answer posted by:"+entry.getValue());
  }
 }
session.close();
}
}
```

⇒ **Output:**



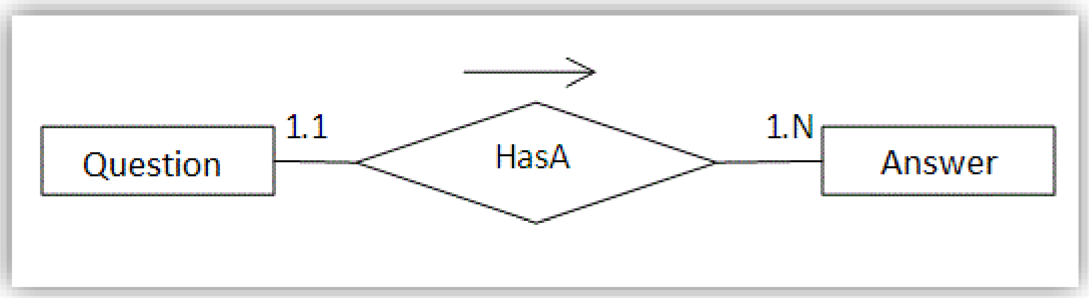
```
Aug 04, 2018 1:56:44 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
question id:1
question name:What is Java?
question posted by:Alok
answers.....
answer name:Java is a programming language
answer posted by:John Milton
answer name:Java is a platform
answer posted by:Ashok Kumar
question id:2
question name:What is Servlet?
question posted by:Jai Dixit
answers.....
answer name:Servlet is a package
answer posted by:Rahul Kumar
answer name:Servlet technology is a server side programming
answer posted by:John Milton
answer name:Servlet is an Interface
answer posted by:Ashok Kumar
```

Problems
Javadoc
Declaration
Console
Debug
FetchTest [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (04-Aug-2018, 1:56:29 PM)

Download

➢ Hibernate One to Many Example using XML:

If the persistent class has list object that contains the entity reference, we need to use one-to-many association to map the list element.
Here, we are using the scenario of Forum where one question has multiple answers.



In such case, there can be many answers for a question and each answer may have its own informations that is why we have used **list** in the persistent class (containing the reference of Answer class) to represent a collection of answers.

Let's see the persistent class that has list objects (containing Answer class objects).

```
package com.javatpoint;

import java.util.List;

public class Question {
        private int id;
        private String qname;
        private List<Answer> answers;

        //getters and setters

}
```

The Answer class has its own informations such as id, answername, postedBy etc.

```
package com.javatpoint;

public class Answer {
        private int id;
        private String answername;
        private String postedBy;

        //getters and setters

}
```

The Question class has list object that have entity reference (i.e. Answer class object). In such case, we need to use **one-to-many** of list to map this object. Let's see how we can map it.

```
        <list name="answers" cascade="all">
            <key column="qid"></key>
            <index column="type"></index>
            <one-to-many class="com.javatpoint.Answer" />
        </list>
```

⇒ *Full example of One to Many mapping in Hibernate by List:*

In this example, we are going to see full example of mapping list that contains entity reference.

1) Create the Persistent class

This persistent class defines properties of the class including List.

Question.java

```
package com.javatpoint;

import java.util.List;

public class Question {
    private int id;
    private String qname;
    private List<Answer> answers;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getQname() {
        return qname;
    }

    public void setQname(String qname) {
        this.qname = qname;
    }

    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }
}
```

Answer.java

```java
package com.javatpoint;

public class Answer {
    private int id;
    private String answername;
    private String postedBy;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getAnswername() {
        return answername;
    }

    public void setAnswername(String answername) {
        this.answername = answername;
    }

    public String getPostedBy() {
        return postedBy;
    }

    public void setPostedBy(String postedBy) {
        this.postedBy = postedBy;
    }
}
```

2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.Question" table="q501">
        <id name="id">
            <generator class="increment"></generator>
        </id>
        <property name="qname"></property>

        <list name="answers" cascade="all">
            <key column="qid"></key>
            <index column="type"></index>
            <one-to-many class="com.javatpoint.Answer" />
        </list>

    </class>

    <class name="com.javatpoint.Answer" table="ans501">
        <id name="id">
```

```
            <generator class="increment"></generator>
        </id>
        <property name="answername"></property>
        <property name="postedBy"></property>
    </class>

</hibernate-mapping>
```

3) Create the configuration file

This file contains information about the database and mapping file.

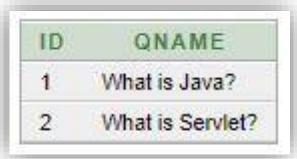
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">


<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="question.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

4) Create the class to store the data

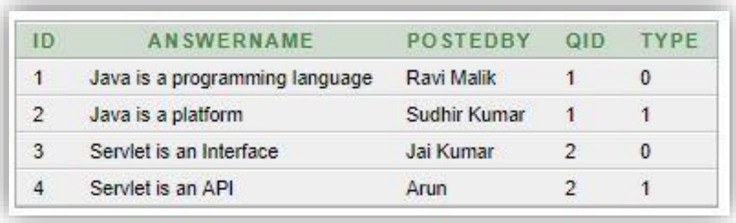
In this class we are storing the data of the question class.

```
package com.javatpoint;

import java.util.ArrayList;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

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

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

        Answer ans1 = new Answer();
        ans1.setAnswername("Java is a programming language");
        ans1.setPostedBy("Ravi Malik");

        Answer ans2 = new Answer();
        ans2.setAnswername("Java is a platform");
        ans2.setPostedBy("Sudhir Kumar");

        Answer ans3 = new Answer();
        ans3.setAnswername("Servlet is an Interface");
        ans3.setPostedBy("Jai Kumar");

        Answer ans4 = new Answer();
        ans4.setAnswername("Servlet is an API");
        ans4.setPostedBy("Arun");

        ArrayList<Answer> list1 = new ArrayList<Answer>();
        list1.add(ans1);
        list1.add(ans2);

        ArrayList<Answer> list2 = new ArrayList<Answer>();
        list2.add(ans3);
        list2.add(ans4);

        Question question1 = new Question();
        question1.setQname("What is Java?");
        question1.setAnswers(list1);

        Question question2 = new Question();
        question2.setQname("What is Servlet?");
        question2.setAnswers(list2);

        session.persist(question1);
        session.persist(question2);

        t.commit();
        session.close();
        System.out.println("success");
    }
}
```

⇒ OUTPUT:



ID	QNAME
1	What is Java?
2	What is Servlet?



ID	ANSWERNAME	POSTEDBY	QID	TYPE
1	Java is a programming language	Ravi Malik	1	0
2	Java is a platform	Sudhir Kumar	1	1
3	Servlet is an Interface	Jai Kumar	2	0
4	Servlet is an API	Arun	2	1

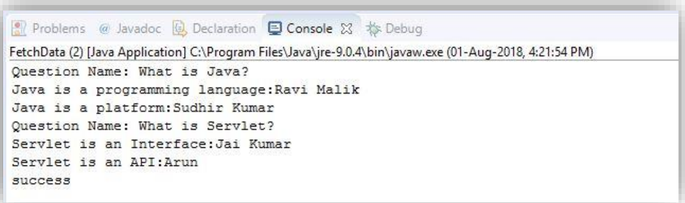
⇒ How to fetch the data of List?

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent. Here, we are direct printing the object of answer class, but we have overridden the **toString()** method in the Answer class returning answername and poster name. So it prints the answer name and postername rather than reference id.

FetchData.java

```java
package com.javatpoint;

import java.util.*;
import javax.persistence.TypedQuery;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchData {
    public static void main(String[] args) {

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

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

        TypedQuery query = session.createQuery("from Question");
        List<Question> list = query.getResultList();

        Iterator<Question> itr = list.iterator();
        while (itr.hasNext()) {
            Question q = itr.next();
            System.out.println("Question Name: " + q.getQname());

            // printing answers
            List<Answer> list2 = q.getAnswers();
            Iterator<Answer> itr2 = list2.iterator();
            while (itr2.hasNext()) {
                Answer a = itr2.next();
                System.out.println(a.getAnswername() + ":" + a.getPostedBy());
            }
        }
        session.close();
        System.out.println("success");
    }

}
```

⇒ **OUTPUT:**

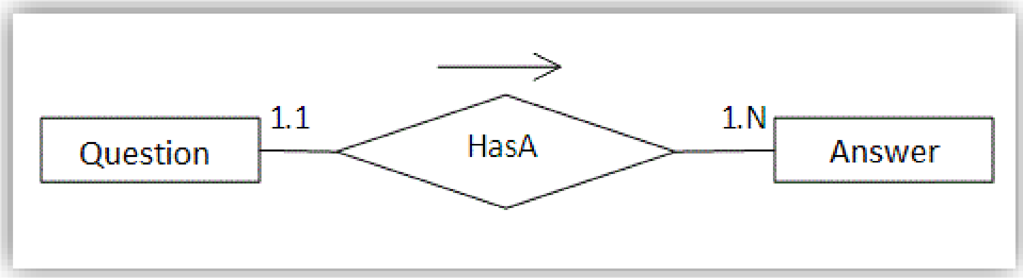


```
FetchData (2) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (01-Aug-2018, 4:21:54 PM)
Question Name: What is Java?
Java is a programming language:Ravi Malik
Java is a platform:Sudhir Kumar
Question Name: What is Servlet?
Servlet is an Interface:Jai Kumar
Servlet is an API:Arun
success
```

➢ Hibernate One to Many Example using Annotation:

In this section, we will perform one-to-many association to map the list object of persistent class using annotation.

Here, we are using the scenario of Forum where one question has multiple answers.



In such case, there can be many answers for a question and each answer may have its own information that is why we have used list in the persistent class (containing the reference of Answer class) to represent a collection of answers.

Example of One to Many mapping using annotation:

1) Create the Persistent class

This persistent class defines properties of the class including List.

Question.java

```
package com.javatpoint;

import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "q5991")
public class Question {

    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    private int id;
    private String qname;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "qid")
    @OrderColumn(name = "type")
    private List<Answer> answers;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
```

```java
    }

    public String getQname() {
        return qname;
    }

    public void setQname(String qname) {
        this.qname = qname;
    }

    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }
}
```

Answer.java

```java
package com.javatpoint;

import javax.persistence.*;

@Entity
@Table(name = "ans5991")
public class Answer {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)

    private int id;
    private String answername;
    private String postedBy;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getAnswername() {
        return answername;
    }

    public void setAnswername(String answername) {
        this.answername = answername;
    }

    public String getPostedBy() {
        return postedBy;
    }

    public void setPostedBy(String postedBy) {
        this.postedBy = postedBy;
    }
}
```

2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies> .... </dependencies> tag.

```xml
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>
```

3) Create the configuration file

This file contains information about the database and mapping file.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.javatpoint.Question" />
    </session-factory>

</hibernate-configuration>
```

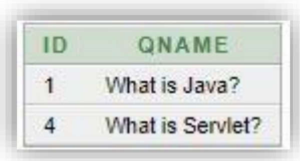
4) Create the class to store the data

In this class we are storing the data of the question class.

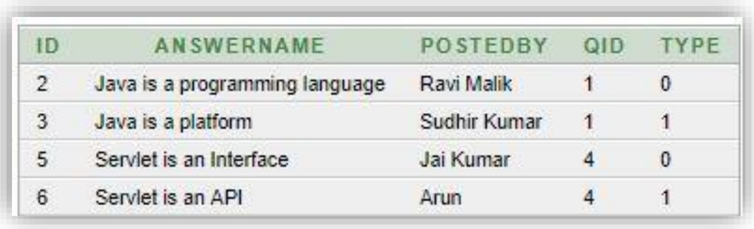
```java
package com.javatpoint;

import java.util.ArrayList;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

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

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

        Transaction t = session.beginTransaction();

        Answer ans1 = new Answer();
        ans1.setAnswername("Java is a programming language");
        ans1.setPostedBy("Ravi Malik");

        Answer ans2 = new Answer();
        ans2.setAnswername("Java is a platform");
        ans2.setPostedBy("Sudhir Kumar");

        Answer ans3 = new Answer();
        ans3.setAnswername("Servlet is an Interface");
        ans3.setPostedBy("Jai Kumar");

        Answer ans4 = new Answer();
        ans4.setAnswername("Servlet is an API");
        ans4.setPostedBy("Arun");

        ArrayList<Answer> list1 = new ArrayList<Answer>();
        list1.add(ans1);
        list1.add(ans2);

        ArrayList<Answer> list2 = new ArrayList<Answer>();
        list2.add(ans3);
        list2.add(ans4);

        Question question1 = new Question();
        question1.setQname("What is Java?");
        question1.setAnswers(list1);

        Question question2 = new Question();
        question2.setQname("What is Servlet?");
        question2.setAnswers(list2);

        session.persist(question1);
        session.persist(question2);

        t.commit();
        session.close();
        System.out.println("success");
    }
}
```

Note -***Using these annotations in a similar way, we can also perform one-to-many association for set, map and bag objects.***

⇒ ***Output:***



ID	QNAME
1	What is Java?
4	What is Servlet?



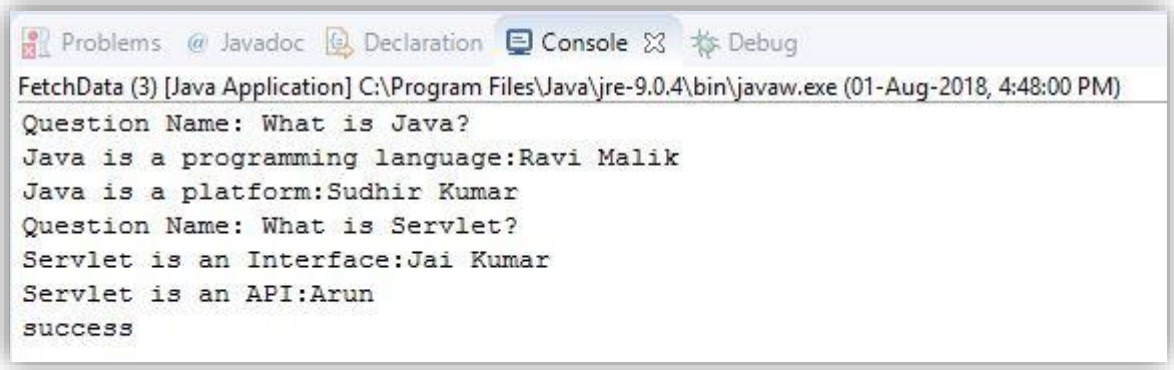
ID	ANSWERNAME	POSTEDBY	QID	TYPE
2	Java is a programming language	Ravi Malik	1	0
3	Java is a platform	Sudhir Kumar	1	1
5	Servlet is an Interface	Jai Kumar	4	0
6	Servlet is an API	Arun	4	1

⇒How to fetch the data of List?

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent.

```
package com.javatpoint;

import java.util.*;
import javax.persistence.TypedQuery;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchData {
    public static void main(String[] args) {

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

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

        TypedQuery query = session.createQuery("from Question");
        List<Question> list = query.getResultList();

        Iterator<Question> itr = list.iterator();
        while (itr.hasNext()) {
            Question q = itr.next();
            System.out.println("Question Name: " + q.getQname());

            // printing answers
            List<Answer> list2 = q.getAnswers();
            Iterator<Answer> itr2 = list2.iterator();
            while (itr2.hasNext()) {
                Answer a = itr2.next();
                System.out.println(a.getAnswername() + ":" + a.getPostedBy());
            }
        }
        session.close();
        System.out.println("success");
    }
}
```

⇒ ***Output:***



FetchData (3) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (01-Aug-2018, 4:48:00 PM)

```
Question Name: What is Java?
Java is a programming language:Ravi Malik
Java is a platform:Sudhir Kumar
Question Name: What is Servlet?
Servlet is an Interface:Jai Kumar
Servlet is an API:Arun
success
```

➢ Hibernate Many to Many Example using XML:

We can map many to many relation either using list, set, bag, map, etc. Here, we are going to use list for many-to-many mapping. In such case, three tables will be created.

⇒ Example of Many to Many Mapping:

In this example, we will generate a many to many relation between questions and answers using list.

1) Create the Persistent class

There are two persistent classes Question.java and Answer.java. Question class contains Answer class reference and vice versa.

Question.java

```java
package com.javatpoint;

import java.util.List;

public class Question {
    private int id;
    private String qname;
    private List<Answer> answers;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getQname() {
        return qname;
    }

    public void setQname(String qname) {
        this.qname = qname;
    }

    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }
}
```

Answer.java

```java
package com.javatpoint;

import java.util.*;

public class Answer {
    private int id;
    private String answername;
    private String postedBy;
    private List<Question> questions;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getAnswername() {
        return answername;
    }

    public void setAnswername(String answername) {
        this.answername = answername;
    }

    public String getPostedBy() {
        return postedBy;
    }

    public void setPostedBy(String postedBy) {
        this.postedBy = postedBy;
    }

    public List<Question> getQuestions() {
        return questions;
    }

    public void setQuestions(List<Question> questions) {
        this.questions = questions;
    }
}
```

2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml and answer.hbm.xml file for defining the list.

question.hbm.xml

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>

    <class name="com.javatpoint.Question" table="ques1911">
        <id name="id" type="int">
```

```
            <column name="q_id" />
            <generator class="increment" />
        </id>
        <property name="qname" />

        <list name="answers" table="ques_ans1911" fetch="select" cascade="all">
            <key column="q_id" />
            <index column="type"></index>
            <many-to-many class="com.javatpoint.Answer" column="ans_id" />
        </list>
    </class>
</hibernate-mapping>
```

answer.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.Answer" table="ans1911">
        <id name="id" type="int">
            <column name="ans_id" />
            <generator class="increment" />
        </id>
        <property name="answername" />
        <property name="postedBy" />
    </class>
</hibernate-mapping>
```

3) Create the configuration file

This file contains information about the database and mapping file.

hibernate.cfg.xml

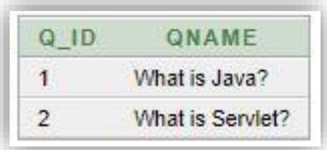
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">create</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="question.hbm.xml" />
        <mapping resource="answer.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

4) Create the class to store the data

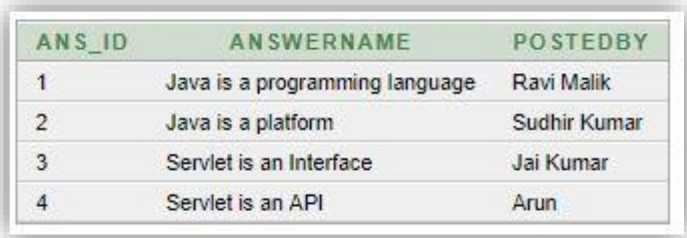
StoreData.java

```java
package com.javatpoint;

import java.util.ArrayList;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
        SessionFactory factory = meta.getSessionFactoryBuilder().build();
        Session session = factory.openSession();
        Transaction t = session.beginTransaction();

        Answer ans1 = new Answer();
        ans1.setAnswername("Java is a programming language");
        ans1.setPostedBy("Ravi Malik");

        Answer ans2 = new Answer();
        ans2.setAnswername("Java is a platform");
        ans2.setPostedBy("Sudhir Kumar");

        Question q1 = new Question();
        q1.setQname("What is Java?");
        ArrayList<Answer> l1 = new ArrayList<Answer>();
        l1.add(ans1);
        l1.add(ans2);
        q1.setAnswers(l1);

        Answer ans3 = new Answer();
        ans3.setAnswername("Servlet is an Interface");
        ans3.setPostedBy("Jai Kumar");

        Answer ans4 = new Answer();
        ans4.setAnswername("Servlet is an API");
        ans4.setPostedBy("Arun");

        Question q2 = new Question();
        q2.setQname("What is Servlet?");
        ArrayList<Answer> l2 = new ArrayList<Answer>();
        l2.add(ans3);
        l2.add(ans4);
        q2.setAnswers(l2);
        session.persist(q1);
        session.persist(q2);

        t.commit();
        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



Q_ID	QNAME
1	What is Java?
2	What is Servlet?



ANS_ID	ANSWERNAME	POSTEDBY
1	Java is a programming language	Ravi Malik
2	Java is a platform	Sudhir Kumar
3	Servlet is an Interface	Jai Kumar
4	Servlet is an API	Arun

Download

➢ Hibernate Many to Many Example using Annotation:

In the previous section, we have performed many to many mapping using XML file. Here, we are going to perform this task using annotation.

We can map many to many relation either using list, set, bag, map etc. Here, we are going to use list for many-to-many mapping. In such case, three tables will be created.

⇒ Example of Many to Many Mapping:

In this example, we will generate a many to many relation between questions and answers by list.

1) Create the Persistent class

Question.java

```
package com.javatpoint;

import java.util.List;
import javax.persistence.*;

@Entity
@Table(name = "ques1123")
public class Question {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String qname;

    @ManyToMany(targetEntity = Answer.class, cascade = { CascadeType.ALL })
    @JoinTable(name = "q_ans1123", joinColumns = { @JoinColumn(name = "q_id") },
inverseJoinColumns = {
            @JoinColumn(name = "ans_id") })
    private List<Answer> answers;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getQname() {
        return qname;
    }

    public void setQname(String qname) {
        this.qname = qname;
    }
    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }
}
```

Answer.java

```
package com.javatpoint;

import javax.persistence.*;

@Entity
@Table(name = "ans1123")
public class Answer {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String answername;
    private String postedBy;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getAnswername() {
        return answername;
    }

    public void setAnswername(String answername) {
        this.answername = answername;
    }

    public String getPostedBy() {
        return postedBy;
    }

    public void setPostedBy(String postedBy) {
        this.postedBy = postedBy;
    }

}
```

2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>
```

3) Create the configuration file

This file contains information about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">create</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.javatpoint.Question" />
        <mapping class="com.javatpoint.Answer" />
    </session-factory>

</hibernate-configuration>
```

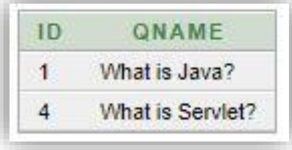
4) Create the class to store the data

StoreData.java

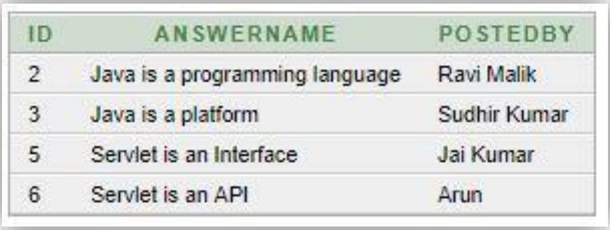
```
package com.javatpoint;

import java.util.ArrayList;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
        SessionFactory factory = meta.getSessionFactoryBuilder().build();
        Session session = factory.openSession();
        Transaction t = session.beginTransaction();

        Answer an1 = new Answer();
        an1.setAnswername("Java is a programming language");
        an1.setPostedBy("Ravi Malik");

        Answer an2 = new Answer();
        an2.setAnswername("Java is a platform");
        an2.setPostedBy("Sudhir Kumar");

        Question q1 = new Question();
        q1.setQname("What is Java?");
        ArrayList<Answer> l1 = new ArrayList<Answer>();
        l1.add(an1);
        l1.add(an2);
        q1.setAnswers(l1);

        Answer ans3 = new Answer();
        ans3.setAnswername("Servlet is an Interface");
        ans3.setPostedBy("Jai Kumar");

        Answer ans4 = new Answer();
        ans4.setAnswername("Servlet is an API");
        ans4.setPostedBy("Arun");

        Question q2 = new Question();
        q2.setQname("What is Servlet?");
        ArrayList<Answer> l2 = new ArrayList<Answer>();
        l2.add(ans3);
        l2.add(ans4);
        q2.setAnswers(l2);

        session.persist(q1);
        session.persist(q2);

        t.commit();
        session.close();
        System.out.println("success");

    }
}
```

⇒ **Output:**



ID	QNAME
1	What is Java?
4	What is Servlet?



ID	ANSWERNAME	POSTEDBY
2	Java is a programming language	Ravi Malik
3	Java is a platform	Sudhir Kumar
5	Servlet is an Interface	Jai Kumar
6	Servlet is an API	Arun

Download

➢ Hibernate One to One Example using XML:

There are two ways to perform one to one mapping in hibernate:

- By many-to-one element (using unique="true" attribute)
- By one-to-one element
-

Here, we are going to perform one to one mapping by one-to-one element. In such case, no foreign key is created in the primary table.

In this example, one employee can have one address and one address belongs to one employee only. Here, we are using bidirectional association. Let's look at the persistent classes.

1) Persistent classes for One to One mapping

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

Employee.java

```
package com.javatpoint;

public class Employee {
    private int employeeId;
    private String name, email;
    private Address address;

    // setters and getters
}
```

Address.java

```
package com.javatpoint;

public class Address {
    private int addressId;
    private String addressLine1, city, state, country;
    private int pincode;
    private Employee employee;

    // setters and getters
}
```

2) Mapping files for the persistent classes

The two mapping files are employee.hbm.xml and address.hbm.xml.

employee.hbm.xml

In this mapping file we are using **one-to-one** element in both the mapping files to make the one to one mapping.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.Employee" table="emp212">
        <id name="employeeId">
            <generator class="increment"></generator>
        </id>
        <property name="name"></property>
        <property name="email"></property>

        <one-to-one name="address" cascade="all"></one-to-one>
    </class>

</hibernate-mapping>
```

address.hbm.xml

This is the simple mapping file for the Address class. But the important thing is generator class. Here, we are using **foreign** generator class that depends on the Employee class primary key.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.Address" table="address212">
        <id name="addressId">
            <generator class="foreign">
                <param name="property">employee</param>
            </generator>
        </id>
        <property name="addressLine1"></property>
        <property name="city"></property>
        <property name="state"></property>
        <property name="country"></property>
        <property name="pincode"></property>

        <one-to-one name="employee"></one-to-one>
    </class>

</hibernate-mapping>
```

3) Configuration file

This file contains information about the database and mapping file.

hibernate.cfg.xml

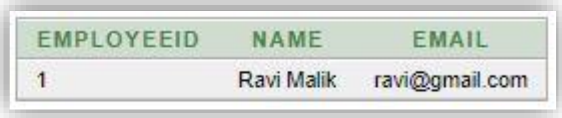
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="employee.hbm.xml" />
        <mapping resource="address.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

4) User classes to store and fetch the data

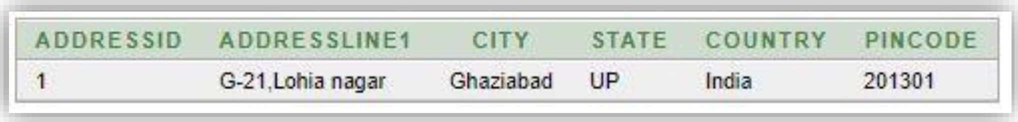
Store.java

```
package com.javatpoint;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Store {
    public static void main(String[] args) {

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

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

        Transaction t = session.beginTransaction();

        Employee e1 = new Employee();
        e1.setName("Ravi Malik");
        e1.setEmail("ravi@gmail.com");

        Address address1 = new Address();
        address1.setAddressLine1("G-21,Lohia nagar");
        address1.setCity("Ghaziabad");
        address1.setState("UP");
        address1.setCountry("India");
        address1.setPincode(201301);

        e1.setAddress(address1);
        address1.setEmployee(e1);

        session.persist(e1);
        t.commit();

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



EMPLOYEEID	NAME	EMAIL
1	Ravi Malik	ravi@gmail.com

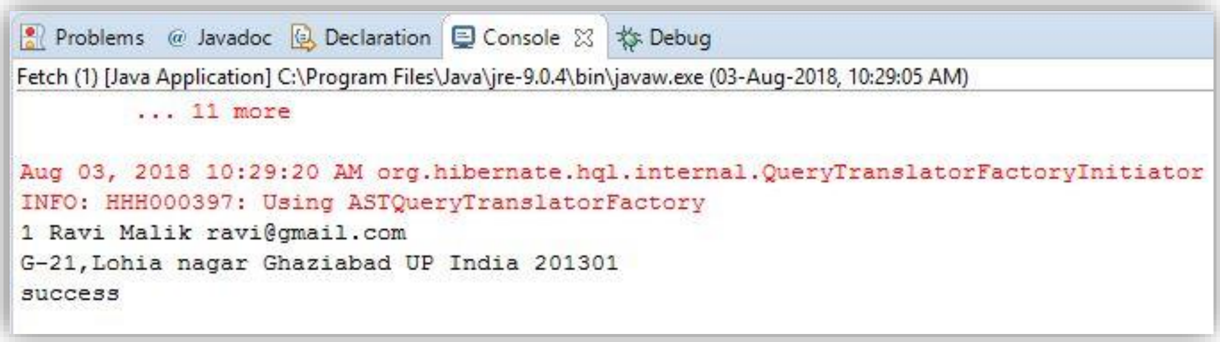


ADDRESSID	ADDRESSLINE1	CITY	STATE	COUNTRY	PINCODE
1	G-21,Lohia nagar	Ghaziabad	UP	India	201301

Fetch.java

```
package com.javatpoint;

import java.util.*;
import javax.persistence.TypedQuery;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Fetch {
    public static void main(String[] args) {
        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

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

        TypedQuery query = session.createQuery("from Employee e");
        List<Employee> list = query.getResultList();

        Iterator<Employee> itr = list.iterator();
        while (itr.hasNext()) {
            Employee emp = itr.next();
            System.out.println(emp.getEmployeeId() + " " + emp.getName() + " " +
emp.getEmail());
            Address address = emp.getAddress();
            System.out.println(address.getAddressLine1() + " " + address.getCity()
+ " " +
                        address.getState() + " " + address.getCountry() + " " +
address.getPincode());
        }

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



```
Fetch (1) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (03-Aug-2018, 10:29:05 AM)
        ... 11 more

Aug 03, 2018 10:29:20 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
1 Ravi Malik ravi@gmail.com
G-21,Lohia nagar Ghaziabad UP India 201301
success
```

Download

➢ Hibernate One to One Example using Annotation:

Here, we are going to perform one to one mapping by one-to-one element using annotation. In such case, no foreign key is created in the primary table.

In this example, one employee can have one address and one address belongs to one employee only. Here, we are using bidirectional association. Let's look at the persistent classes.

1) Persistent classes for One to One mapping

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

Employee.java

```java
package com.javatpoint;

import javax.persistence.*;

@Entity
@Table(name = "emp220")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @PrimaryKeyJoinColumn
    private int employeeId;
    private String name, email;
    @OneToOne(targetEntity = Address.class, cascade = CascadeType.ALL)
    private Address address;

    public int getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

}
```

Address.java

```
package com.javatpoint;

import javax.persistence.*;

@Entity
@Table(name = "address220")
public class Address {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int addressId;
    private String addressLine1, city, state, country;
    private int pincode;

    @OneToOne(targetEntity = Employee.class)
    private Employee employee;

    public int getAddressId() {
        return addressId;
    }

    public void setAddressId(int addressId) {
        this.addressId = addressId;
    }

    public String getAddressLine1() {
        return addressLine1;
    }

    public void setAddressLine1(String addressLine1) {
        this.addressLine1 = addressLine1;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }
    public void setState(String state) {
        this.state = state;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public int getPincode() {
        return pincode;
    }

    public void setPincode(int pincode) {
        this.pincode = pincode;
    }

    public Employee getEmployee() {
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }
}
```

2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag. These dependencies are used to add the jar files in Maven project.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>
```

3) Configuration file

This file contains information about the database and mapping file.

hibernate.cfg.xml

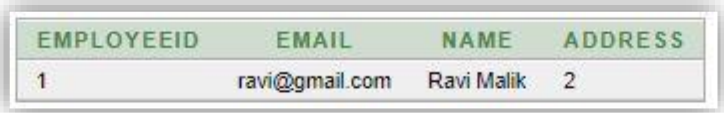
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.javatpoint.Address" />
        <mapping class="com.javatpoint.Employee" />
    </session-factory>

</hibernate-configuration>
```

4) User classes to store and fetch the data

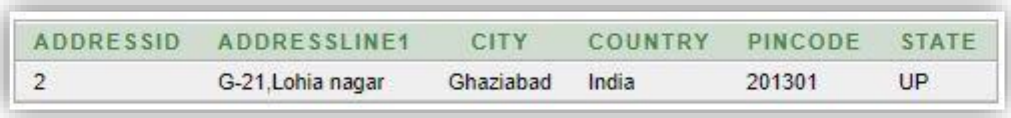
Store.java

```
package com.javatpoint;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Store {
    public static void main(String[] args) {

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

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

        Transaction t = session.beginTransaction();

        Employee e1 = new Employee();
        e1.setName("Ravi Malik");
        e1.setEmail("ravi@gmail.com");

        Address address1 = new Address();
        address1.setAddressLine1("G-21,Lohia nagar");
        address1.setCity("Ghaziabad");
        address1.setState("UP");
        address1.setCountry("India");
        address1.setPincode(201301);

        e1.setAddress(address1);
        address1.setEmployee(e1);

        session.persist(e1);
        t.commit();

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



EMPLOYEEID	EMAIL	NAME	ADDRESS
1	ravi@gmail.com	Ravi Malik	2



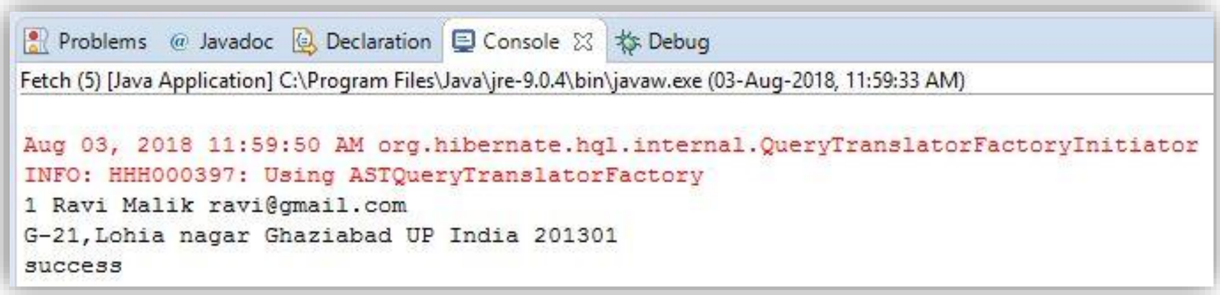
ADDRESSID	ADDRESSLINE1	CITY	COUNTRY	PINCODE	STATE
2	G-21,Lohia nagar	Ghaziabad	India	201301	UP

Fetch.java

```
package com.javatpoint;

import java.util.Iterator;
import java.util.List;

import javax.persistence.TypedQuery;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Fetch {
    public static void main(String[] args) {
        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

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

        TypedQuery query = session.createQuery("from Employee");
        List<Employee> list = query.getResultList();

        Iterator<Employee> itr = list.iterator();
        while (itr.hasNext()) {
            Employee emp = itr.next();
            System.out.println(emp.getEmployeeId() + " " + emp.getName() + " " +
emp.getEmail());
            Address address = emp.getAddress();
            System.out.println(address.getAddressLine1() + " " + address.getCity()
+ " " +
                    address.getState() + " " + address.getCountry() + " " +
address.getPincode());
```

```
        }

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



```
Fetch (5) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (03-Aug-2018, 11:59:33 AM)

Aug 03, 2018 11:59:50 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
1 Ravi Malik ravi@gmail.com
G-21,Lohia nagar Ghaziabad UP India 201301
success
```

Download

➢Hibernate Many to One Mapping using XML:

In many to one mapping, various attributes can be referred to one attribute only.

In this example, every employee has one company address only and one address belongs to many employees. Here, we are going to perform many to one mapping using XML.

1) Persistent classes for One to One mapping

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

Employee.java

```
package com.javatpoint;

public class Employee {
    private int employeeId;
    private String name, email;
    private Address address;

    // setters and getters
}
```

Address.java

```
package com.javatpoint;

public class Address {
    private int addressId;
    private String addressLine1, city, state, country;
    private int pincode;
    private Employee employee;

    // setters and getters
}
```

2) Mapping files for the persistent classes

The two mapping files are employee.hbm.xml and address.hbm.xml.

employee.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
```

```
    <class name="com.javatpoint.Employee" table="emp22">
        <id name="employeeId">
            <generator class="increment"></generator>
        </id>
        <property name="name"></property>
        <property name="email"></property>

        <many-to-one name="address" cascade="all"></many-to-one>
    </class>

</hibernate-mapping>
```

address.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.Address" table="address22">
        <id name="addressId">
            <generator class="increment"></generator>
        </id>
        <property name="addressLine1"></property>
        <property name="city"></property>
        <property name="state"></property>
        <property name="country"></property>
        <property name="pincode"></property>
    </class>
</hibernate-mapping>
```

3) Configuration file

This file contains information about the database and mapping file.

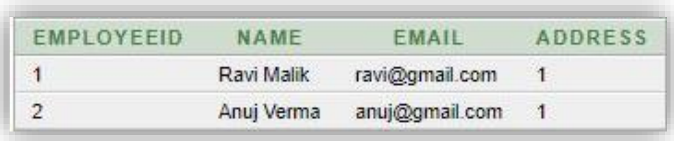
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="employee.hbm.xml" />
        <mapping resource="address.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

4) User classes to store and fetch the data

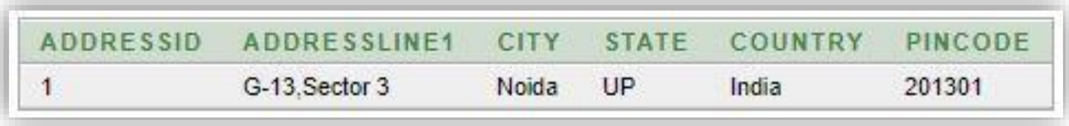
Store.java

```
package com.javatpoint;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Store {
    public static void main(String[] args) {

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

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

        Transaction t = session.beginTransaction();

        Employee e1 = new Employee();
        e1.setName("Ravi Malik");
        e1.setEmail("ravi@gmail.com");

        Employee e2 = new Employee();
        e2.setName("Anuj Verma");
        e2.setEmail("anuj@gmail.com");

        Address address1 = new Address();
        address1.setAddressLine1("G-13,Sector 3");
        address1.setCity("Noida");
        address1.setState("UP");
        address1.setCountry("India");
        address1.setPincode(201301);

        e1.setAddress(address1);
        e2.setAddress(address1);

        session.persist(e1);
        session.persist(e2);
        t.commit();

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



EMPLOYEEID	NAME	EMAIL	ADDRESS
1	Ravi Malik	ravi@gmail.com	1
2	Anuj Verma	anuj@gmail.com	1

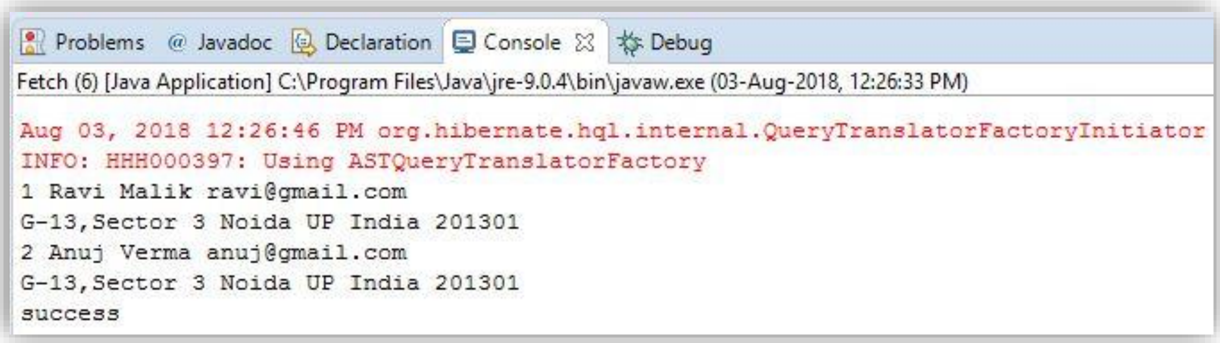


ADDRESSID	ADDRESSLINE1	CITY	STATE	COUNTRY	PINCODE
1	G-13,Sector 3	Noida	UP	India	201301

Fetch.java

```
 package com.javatpoint;

import java.util.Iterator;
import java.util.List;

import javax.persistence.TypedQuery;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Fetch {
    public static void main(String[] args) {
        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

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

        TypedQuery query = session.createQuery("from Employee e");
        List<Employee> list = query.getResultList();

        Iterator<Employee> itr = list.iterator();
        while (itr.hasNext()) {
            Employee emp = itr.next();
            System.out.println(emp.getEmployeeId() + " " + emp.getName() + " " +
emp.getEmail());
            Address address = emp.getAddress();
            System.out.println(address.getAddressLine1() + " " + address.getCity()
+ " " +
                    address.getState() + " " + address.getCountry() + " " +
address.getPincode());
        }

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**


Problems @ Javadoc Declaration Console Debug

Fetch (6) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (03-Aug-2018, 12:26:33 PM)

```
Aug 03, 2018 12:26:46 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
1 Ravi Malik ravi@gmail.com
G-13,Sector 3 Noida UP India 201301
2 Anuj Verma anuj@gmail.com
G-13,Sector 3 Noida UP India 201301
success
```

Download

➢Hibernate Many to One Mapping using Annotation:

In many to one mapping, various attributes can be referred to one attribute only.

In this example, every employee has one company address only and one address belongs to many employees. Here, we are going to perform many to one mapping using annotation.

Let's look at the persistent classes

1) Persistent classes for One to One mapping

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

Employee.java

```java
package com.javatpoint;

import javax.persistence.*;

@Entity
@Table(name = "emp107")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int employeeId;
    private String name, email;
    @ManyToOne(cascade = CascadeType.ALL)
    private Address address;

    public int getEmployeeId() {
        return employeeId;
    }

    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }
}
```

Address.java

```
package com.javatpoint;

import javax.persistence.*;

@Entity
@Table(name = "address107")
public class Address {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int addressId;
    private String addressLine1, city, state, country;
    private int pincode;
    @OneToOne(cascade = CascadeType.ALL)
    private Employee employee;

    public int getAddressId() {
        return addressId;
    }

    public void setAddressId(int addressId) {
        this.addressId = addressId;
    }

    public String getAddressLine1() {
        return addressLine1;
    }

    public void setAddressLine1(String addressLine1) {
        this.addressLine1 = addressLine1;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }
```

```java
    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public int getPincode() {
        return pincode;
    }

    public void setPincode(int pincode) {
        this.pincode = pincode;
    }

    public Employee getEmployee() {
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }
}
```

2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag. These dependencies are used to add the jar files in Maven project.

```xml
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>
```

3) Configuration file

This file contains information about the database and mapping file.

hibernate.cfg.xml

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

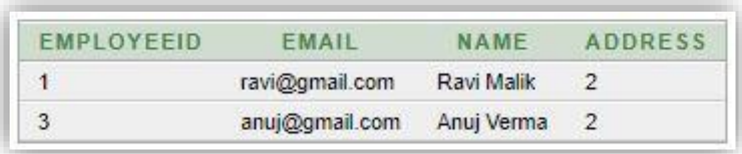
```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">create</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.javatpoint.Address" />
        <mapping class="com.javatpoint.Employee" />
    </session-factory>

</hibernate-configuration>
```

4) User classes to store and fetch the data

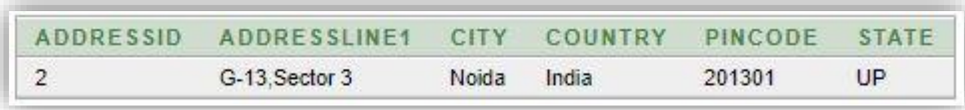
Store.java

```
package com.javatpoint;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Store {
    public static void main(String[] args) {

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

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

        Transaction t = session.beginTransaction();

        Employee e1 = new Employee();
        e1.setName("Ravi Malik");
        e1.setEmail("ravi@gmail.com");

        Employee e2 = new Employee();
        e2.setName("Anuj Verma");
        e2.setEmail("anuj@gmail.com");

        Address address1 = new Address();
        address1.setAddressLine1("G-13,Sector 3");
        address1.setCity("Noida");
        address1.setState("UP");
        address1.setCountry("India");
        address1.setPincode(201301);
        e1.setAddress(address1);
        e2.setAddress(address1);

        session.persist(e1);
        session.persist(e2);
        t.commit();

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



EMPLOYEEID	EMAIL	NAME	ADDRESS
1	ravi@gmail.com	Ravi Malik	2
3	anuj@gmail.com	Anuj Verma	2

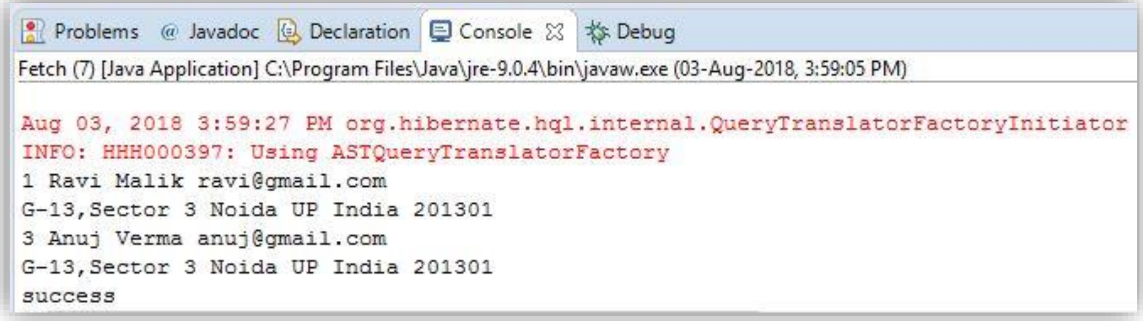


ADDRESSID	ADDRESSLINE1	CITY	COUNTRY	PINCODE	STATE
2	G-13,Sector 3	Noida	India	201301	UP

Fetch.java

```
package com.javatpoint;

import java.util.Iterator;
import java.util.List;

import javax.persistence.TypedQuery;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Fetch {
    public static void main(String[] args) {
        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

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

        TypedQuery query = session.createQuery("from Employee e");
        List<Employee> list = query.getResultList();

        Iterator<Employee> itr = list.iterator();
        while (itr.hasNext()) {
            Employee emp = itr.next();
            System.out.println(emp.getEmployeeId() + " " + emp.getName() + " " +
emp.getEmail());
            Address address = emp.getAddress();
            System.out.println(address.getAddressLine1() + " " + address.getCity()
+ " " +
                    address.getState() + " " + address.getCountry() + " " +
address.getPincode());
        }

        session.close();
        System.out.println("success");
    }
}
```

⇒ **Output:**



```
Fetch (7) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (03-Aug-2018, 3:59:05 PM)

Aug 03, 2018 3:59:27 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
1 Ravi Malik ravi@gmail.com
G-13,Sector 3 Noida UP India 201301
3 Anuj Verma anuj@gmail.com
G-13,Sector 3 Noida UP India 201301
success
```

Download

➢ Bidirectional Association:

Bidirectional association allows us to fetch details of dependent object from both side. In such case, we have the reference of two classes in each other.

Let's take an example of Employee and Address, if Employee class has-a reference of Address and Address has a reference of Employee.
Additionally, you have applied one-to-one or one-to-many relationship for the classes in mapping file as well, it is known as bidirectional association.

➢ Hibernate Lazy Collection:

Lazy collection loads the child objects on demand, it is used to improve performance. Since Hibernate 3.0, lazy collection is enabled by default.

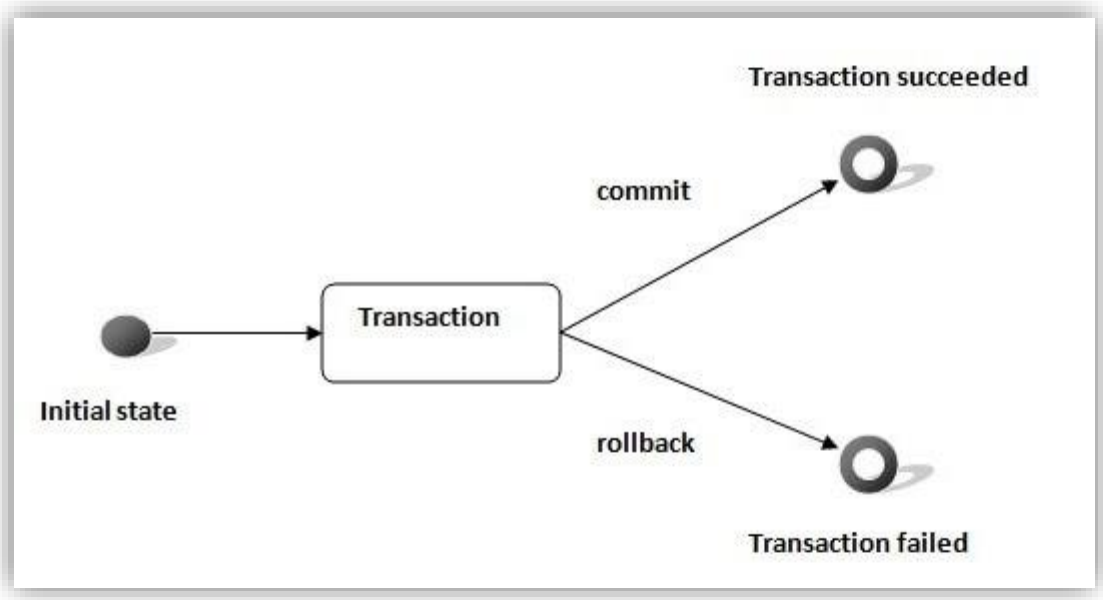
To use lazy collection, you may optionally use lazy="true" attribute in your collection. It is by default true, so you don't need to do this. If you set it to false, all the child objects will be loaded initially which will decrease performance in case of big data.

Let's see the hibernate mapping file where we have used lazy="true" attribute.

```
<list name="answers" lazy="true">
        <key column="qid"></key>
         <index column="type"></index>
        <one-to-many class="com.javatpoint.Answer"/>
</list>
```

➢ Hibernate Transaction Management Example:

A **transaction** simply represents a unit of work. In such case, if one step fails, the whole transaction fails (which is termed as atomicity). A transaction can be described by ACID properties (Atomicity, Consistency, Isolation and Durability).



❖ Transaction Interface in Hibernate:

In hibernate framework, we have **Transaction** interface that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC).

A transaction is associated with Session and instantiated by calling **session.beginTransaction()**.

The methods of Transaction interface are as follows:

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

⇒Example of Transaction Management in Hibernate:

In hibernate, it is better to rollback the transaction if any exception occurs, so that resources can be free. Let's see the example of transaction management in hibernate.

```
Session session = null;
Transaction tx = null;

try {
session = sessionFactory.openSession();
tx = session.beginTransaction();
//some action

tx.commit();

}catch (Exception ex) {
ex.printStackTrace();
tx.rollback();
}finally {session.close();}
```

➢ Hibernate Query Language (HQL):

Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depend on the table of the database. Instead of table name, we use class name in HQL. So it is database independent query language.

⇒ Advantage of HQL:

There are many advantages of HQL. They are as follows:

- database independent
- supports polymorphic queries
- easy to learn for Java Programmer

❖ Query Interface:

It is an object-oriented representation of Hibernate Query. The object of Query can be obtained by calling the createQuery() method Session interface.

The query interface provides many methods. There is given commonly used methods:

1. **public int executeUpdate()** is used to execute the update or delete query.
2. **public List list()** returns the result of the ralation as a list.
3. **public Query setFirstResult(int rowno)** specifies the row number from where record will be retrieved.
4. **public Query setMaxResult(int rowno)** specifies the no. of records to be retrieved from the relation (table).
5. **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
6. **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.

Example of HQL to get all the records:

```
//here persistent class name is Emp
Query query=session.createQuery("from Emp");
List list=query.list();
```

Example of HQL to get records with pagination:

```
Query query=session.createQuery("from Emp");
query.setFirstResult(5);
query.setMaxResult(10);
List list=query.list();      //will return the records from 5 to 10th number
```

Example of HQL update query:

```
Transaction tx=session.beginTransaction();
Query q=session.createQuery("update User set name=:n where id=:i");
q.setParameter("n","Udit Kumar");
q.setParameter("i",111);

int status=q.executeUpdate();
System.out.println(status);
tx.commit();
```

Example of HQL delete query:

```
Query query=session.createQuery("delete from Emp where id=100");
//specifying class name (Emp) not tablename
query.executeUpdate();
```

❖ HQL with Aggregate functions:

You may call avg(), min(), max() etc. aggregate functions by HQL. Let's see some common examples:

⇒Example to get total salary of all the employees:

```
Query q=session.createQuery("select sum(salary) from Emp");
List<Integer> list=q.list();
System.out.println(list.get(0));
```

Example to get maximum salary of employee:

```
Query q=session.createQuery("select max(salary) from Emp");
```

Example to get minimum salary of employee:

```
Query q=session.createQuery("select min(salary) from Emp");
```

Example to count total number of employee ID:

```
Query q=session.createQuery("select count(id) from Emp");
```

Example to get average salary of each employees:

```
Query q=session.createQuery("select avg(salary) from Emp");
```

➢ HCQL (Hibernate Criteria Query Language):

The Hibernate Criteria Query Language (HCQL) is used to fetch the records based on the specific criteria. The Criteria interface provides methods to apply criteria such as retreiving all the records of table whose salary is greater than 50000 etc.

Advantages of HCQL:

The HCQL provides methods to add criteria, so it is **easy** for the java programmer to add criteria. The java programmer is able to add many criteria on a query.

Criteria Interface:

The Criteria interface provides many methods to specify criteria. The object of Criteria can be obtained by calling the **createCriteria()** method of Session interface.

Syntax of createCriteria() method of Session interface:

```
public Criteria createCriteria(Class c)
```

The commonly used methods of Criteria interface are as follows:

1. **public Criteria add(Criterion c)** is used to add restrictions.
2. **public Criteria addOrder(Order o)** specifies ordering.
3. **public Criteria setFirstResult(int firstResult)** specifies the first number of record to be retreived.
4. **public Criteria setMaxResult(int totalResult)** specifies the total number of records to be retreived.
5. **public List list()** returns list containing object.
6. **public Criteria setProjection(Projection projection)** specifies the projection.

❖ Restrictions class:

Restrictions class provides methods that can be used as Criterion. The commonly used methods of Restrictions class are as follows:

- **public static SimpleExpression lt(String propertyName,Object value)** sets the **less than** constraint to the given property.
- **public static SimpleExpression le(String propertyName,Object value)** sets the **less than or equal** constraint to the given property.
- **public static SimpleExpression gt(String propertyName,Object value)** sets the **greater than** constraint to the given property.
- **public static SimpleExpression ge(String propertyName,Object value)** sets the **greater than or equal** than constraint to the given property.
- **public static SimpleExpression ne(String propertyName,Object value)** sets the **not equal** constraint to the given property.

- **public static SimpleExpression eq(String propertyName,Object value)** sets the **equal** constraint to the given property.
- **public static Criterion between(String propertyName, Object low, Object high)** sets the **between** constraint.
- **public static SimpleExpression like(String propertyName, Object value)** sets the **like** constraint to the given property.

❖ Order class:

The Order class represents an order. The commonly used methods of Restrictions class are as follows:

1. **public static Order asc(String propertyName)** applies the ascending order on the basis of given property.
2. **public static Order desc(String propertyName)** applies the descending order on the basis of given property.

Examples of Hibernate Criteria Query Language:

There are given a lot of examples of HCQL.

⇒Example of HCQL to get all the records:

```
Crietria c=session.createCriteria(Emp.class);//passing Class class argument
List list=c.list();
```

⇒Example of HCQL to get the 10th to 20th record:

```
Crietria c=session.createCriteria(Emp.class);
c.setFirstResult(10);
c.setMaxResult(20);
List list=c.list();
```

⇒Example of HCQL to get the records whose salary is greater than 10000:

```
Crietria c=session.createCriteria(Emp.class);
c.add(Restrictions.gt("salary",10000));//salary is the propertyname
List list=c.list();
```

⇒Example of HCQL to get the records in ascending order on the basis of salary:

```
Crietria c=session.createCriteria(Emp.class);
```

```
c.addOrder(Order.asc("salary"));
List list=c.list();
```

❖ HCQL with Projection:

We can fetch data of a particular column by projection such as name etc. Let's see the simple example of projection that prints data of NAME column of the table only.

```
Criteria c=session.createCriteria(Emp.class);
c.setProjection(Projections.property("name"));
List list=c.list();
```

➢Hibernate Named Query:

The hibernate named query is way to use any query by some meaningful name. It is like using alias names. The Hibernate framework provides the concept of named queries so that application programmer need not to scatter queries to all the java code.

There are two ways to define the named query in hibernate:

- by annotation
- by mapping file.

⇒Hibernate Named Query by annotation:

If you want to use named query in hibernate, you need to have knowledge of @NamedQueries and @NamedQuery annotations.

@NameQueries annotation is used to define the multiple named queries.
@NameQuery annotation is used to define the single named query.

Let's see the example of using the named queries:

```
@NamedQueries(
    {
        @NamedQuery(
        name = "findEmployeeByName",
        query = "from Employee e where e.name = :name"
        )
    }
)
```

⇒***Example of Hibernate Named Query by annotation:***

In this example, we are using annotations to defined the named query in the persistent class. There are three files only:

- Employee.java
- hibernate.cfg.xml
- FetchDemo

In this example, we are assuming that there is em table in the database containing 4 columns id, name, job and salary and there are some records in this table.

Employee.java

It is a persistent class that uses annotations to define named query and marks this class as entity.

```java
package com.javatpoint;

import javax.persistence.*;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@NamedQueries(
    {
        @NamedQuery(
        name = "findEmployeeByName",
        query = "from Employee e where e.name = :name"
        )
    }
)

@Entity
@Table(name="em")
public class Employee {

    public String toString(){return id+" "+name+" "+salary+" "+job;}

    int id;
    String name;
    int salary;
    String job;
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)

    //getters and setters
}
```

hibernate.cfg.xml

It is a configuration file that stores the informations about database such as driver class, url, username, password and mapping class etc.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
5.3//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.javatpoint.Employee" />
    </session-factory>

</hibernate-configuration>
```

FetchData.java

It is a java class that uses the named query and prints the informations based on the query. The **getNamedQuery** method uses the named query and returns the instance of Query.

```
package com.javatpoint;

import java.util.*;
import javax.persistence.*;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Fetch {
    public static void main(String[] args) {

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

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

        // Hibernate Named Query
        TypedQuery query = session.getNamedQuery("findEmployeeByName");
        query.setParameter("name", "amit");

        List<Employee> employees = query.getResultList();

        Iterator<Employee> itr = employees.iterator();
        while (itr.hasNext()) {
            Employee e = itr.next();
            System.out.println(e);
        }
        session.close();
    }
}
```

❖ Hibernate Named Query by mapping file:

If want to define named query by mapping file, you need to use **query** element of hibernate-mapping to define the named query.

In such case, you need to create hbm file that defines the named query. Other resources are same as given in the above example except Persistent class

Employee.java where you don't need to use any annotation and hibernate.cfg.xml file where you need to specify mapping resource of the hbm file.

The hbm file should be like this:

emp.hbm.xml

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.javatpoint.Employee" table="em">
        <id name="id">
            <generator class="native"></generator>
        </id>
        <property name="name"></property>
        <property name="job"></property>
        <property name="salary"></property>
    </class>

    <query name="findEmployeeByName">
        <![CDATA[from Employee e where e.name = :name]]>
    </query>

</hibernate-mapping>
```

The persistent class should be like this:

Employee.java

```java
package com.javatpoint;

public class Employee {
    int id;
    String name;
    int salary;
    String job;

    // getters and setters
}
```

Now include the mapping resource in the hbm file as:

hibernate.cfg.xml

```xml
<mapping resource="emp.hbm.xml"/>
```

➢Caching in Hibernate:

Hibernate caching improves the performance of the application by pooling the object in the cache. It is useful when we have to fetch the same data multiple times.

There are mainly two types of caching:

- First Level Cache, and
- Second Level Cache

❖ *First Level Cache:*

Session object holds the first level cache data. It is enabled by default. The first level cache data will not be available to entire application. An application can use many session object.

❖ *Second Level Cache:*

SessionFactory object holds the second level cache data. The data stored in the second level cache will be available to entire application. But we need to enable it explicitely.

- EH (Easy Hibernate) Cache
- Swarm Cache
- OS Cache
- JBoss Cache

❖ Hibernate Second Level Cache:

Hibernate second level cache uses *a common cache for all the session object of a session factory*. It is useful if you have multiple session objects from a session factory. **SessionFactory** holds the second level cache data. It is global for all the session objects and not enabled by default.

Different vendors have provided the implementation of Second Level Cache.

1. EH Cache
2. OS Cache
3. Swarm Cache
4. JBoss Cache

Each implementation provides different cache usage functionality. There are four ways to use second level cache.

1. **read-only:** caching will work for read only operation.
2. **nonstrict-read-write:** caching will work for read and write but one at a time.
3. **read-write:** caching will work for read and write, can be used simultaneously.
4. **transactional:** caching will work for transaction.

The cache-usage property can be applied to class or collection level in hbm.xml file.
The example to define cache usage is given below:

```
<cache usage="read-only" />
```

Let's see the second level cache implementation and cache usage.

Implementation	read-only	nonstrict-read-write	read-write	transactional
EH Cache	Yes	Yes	Yes	No
OS Cache	Yes	Yes	Yes	No
Swarm Cache	Yes	Yes	No	No
JBoss Cache	No	No	No	Yes

❖ Hibernate Second Level Cache Example:

To understand the second level cache through example, we need to follow the following steps:

1. Create the persistent class using Maven
2. Add project information and configuration in pom.xml file
3. Create the Configuration file
4. Create the class that retrieves the persistent object.

Here, we are assuming, there is emp1012 table in the oracle database containing some records.

1) Create the persistent class using Maven.

File: Employee.java

```
package com.javatpoint;

import javax.persistence.*;
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

@Entity
@Table(name = "emp1012")
@Cacheable
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
public class Employee {
    @Id
    private int id;
    private String name;
    private float salary;

    public Employee() {
    }

    public Employee(String name, float salary) {
        super();
        this.name = name;
        this.salary = salary;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public float getSalary() {
        return salary;
    }

    public void setSalary(float salary) {
        this.salary = salary;
    }
}
```

2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag.

```xml
 <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.16.Final</version>
</dependency>

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>

<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.10.3</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
    <version>5.2.16.Final</version>
</dependency>
```

3) Create the Configuration file

File: hibernate.cfg.xml

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 5.2.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-5.2.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <property name="cache.use_second_level_cache">true</property>
        <property
name="cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory
</property>
        <mapping class="com.javatpoint.Employee" />
    </session-factory>
</hibernate-configuration>
```

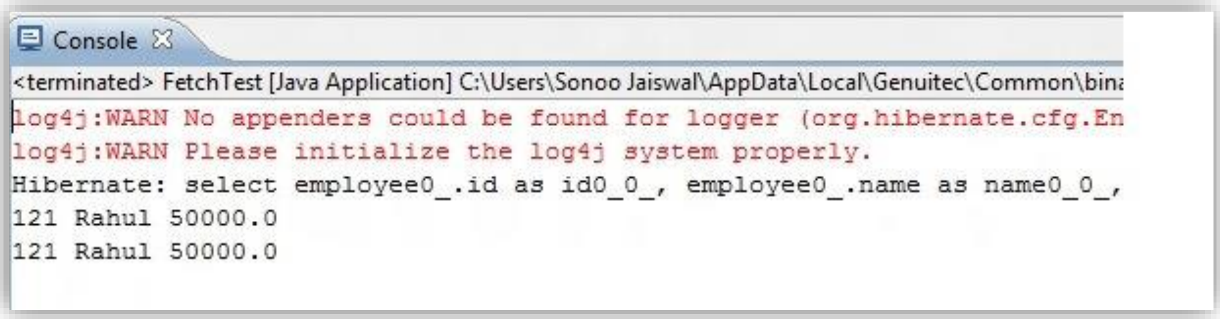
To implement second level cache, we need to define **cache.provider_class** property in the configuration file.

4) Create the class that retrieves the persistent object.

File: FetchTest.java

```
package com.javatpoint;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchTest {
    public static void main(String[] args) {
        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

        SessionFactory factory = meta.getSessionFactoryBuilder().build();

        Session session1 = factory.openSession();
        Employee emp1 = (Employee) session1.load(Employee.class, 121);
        System.out.println(emp1.getId() + " " + emp1.getName() + " " +
emp1.getSalary());
        session1.close();

        Session session2 = factory.openSession();
        Employee emp2 = (Employee) session2.load(Employee.class, 121);
        System.out.println(emp2.getId() + " " + emp2.getName() + " " +
emp2.getSalary());
        session2.close();

    }
}
```

⇒ ***Output:***



```
<terminated> FetchTest [Java Application] C:\Users\Sonoo Jaiswal\AppData\Local\Genuitec\Common\bina
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.En
log4j:WARN Please initialize the log4j system properly.
Hibernate: select employee0_.id as id0_0_, employee0_.name as name0_0_,
121 Rahul 50000.0
121 Rahul 50000.0
```

As we can see here, hibernate does not fire query twice. If you don't use second level cache, hibernate will fire query twice because both query uses different session objects.