Hibernate Tutorial

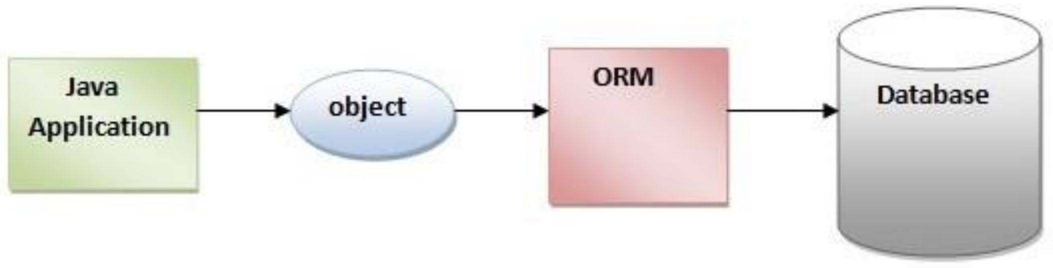
This hibernate tutorial provides in-depth concepts of Hibernate Framework with simplified examples. It was started in 2001 by Gavin King as an alternative to EJB2 style entity bean.

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.

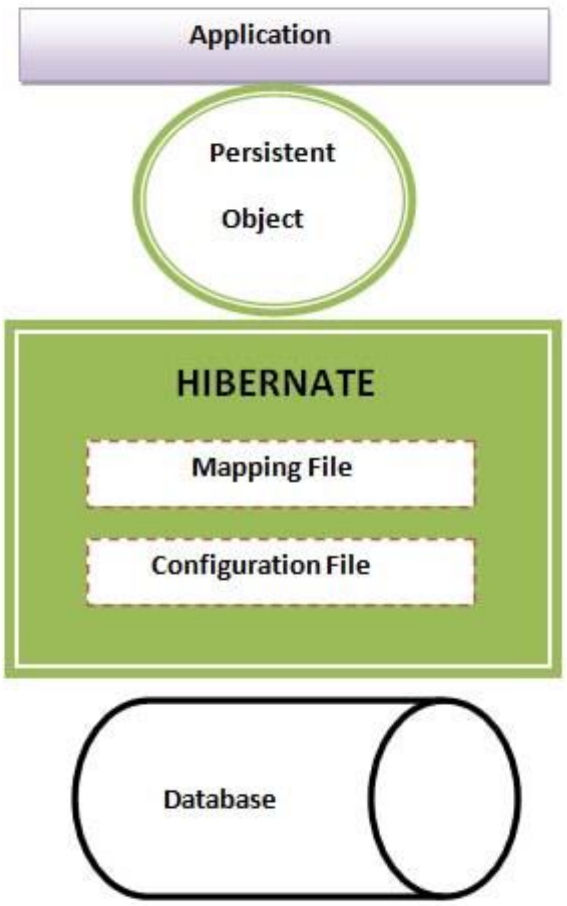
Hibernate Architecture

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.

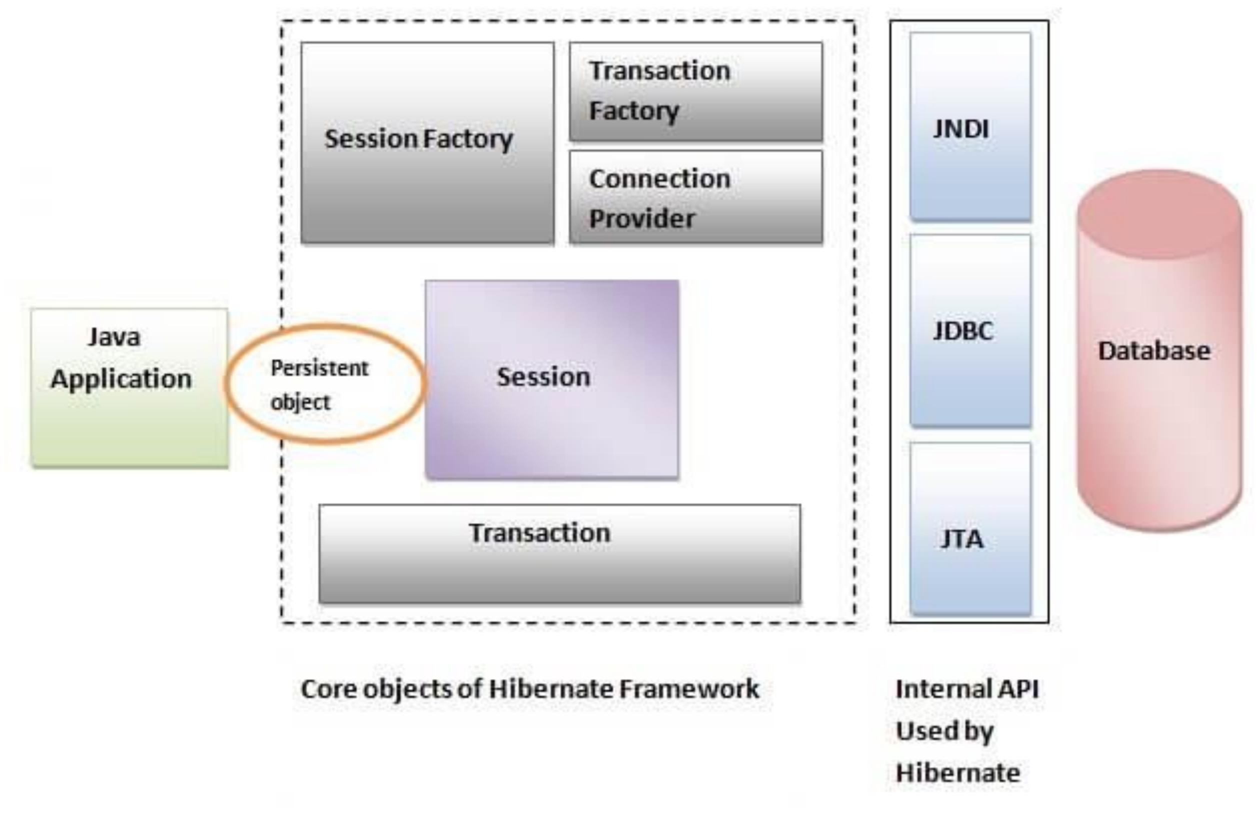
- Java application layer
- Hibernate framework layer
- Backhand api layer
- Database layer

Let's see the diagram of hibernate architecture:



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

OOPs Concepts in Java



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 with IDE

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.

3) 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.
- 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** .

4) 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.
- 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.

5) Create the Configuration file

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

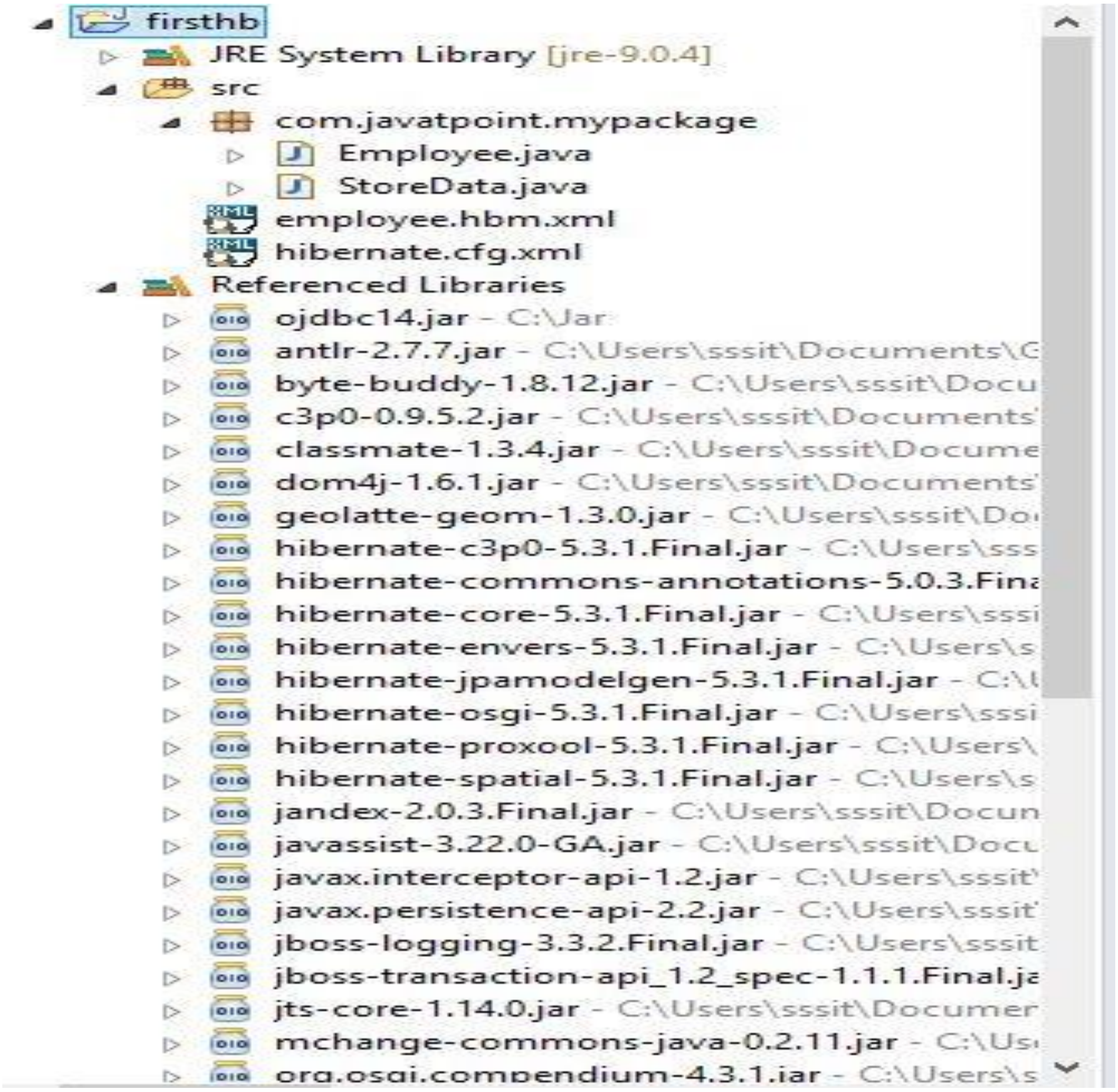
The configuration file contains all the informations for the database such as connection_url, driver_class, username, password etc. The hbm2ddl.auto property is used to create the table in the database automatically. We will have in-depth learning about Dialect class in next topics. To create the configuration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

6) Create the class that retrieves or stores the object

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

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.Fina
    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:\U
    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
```

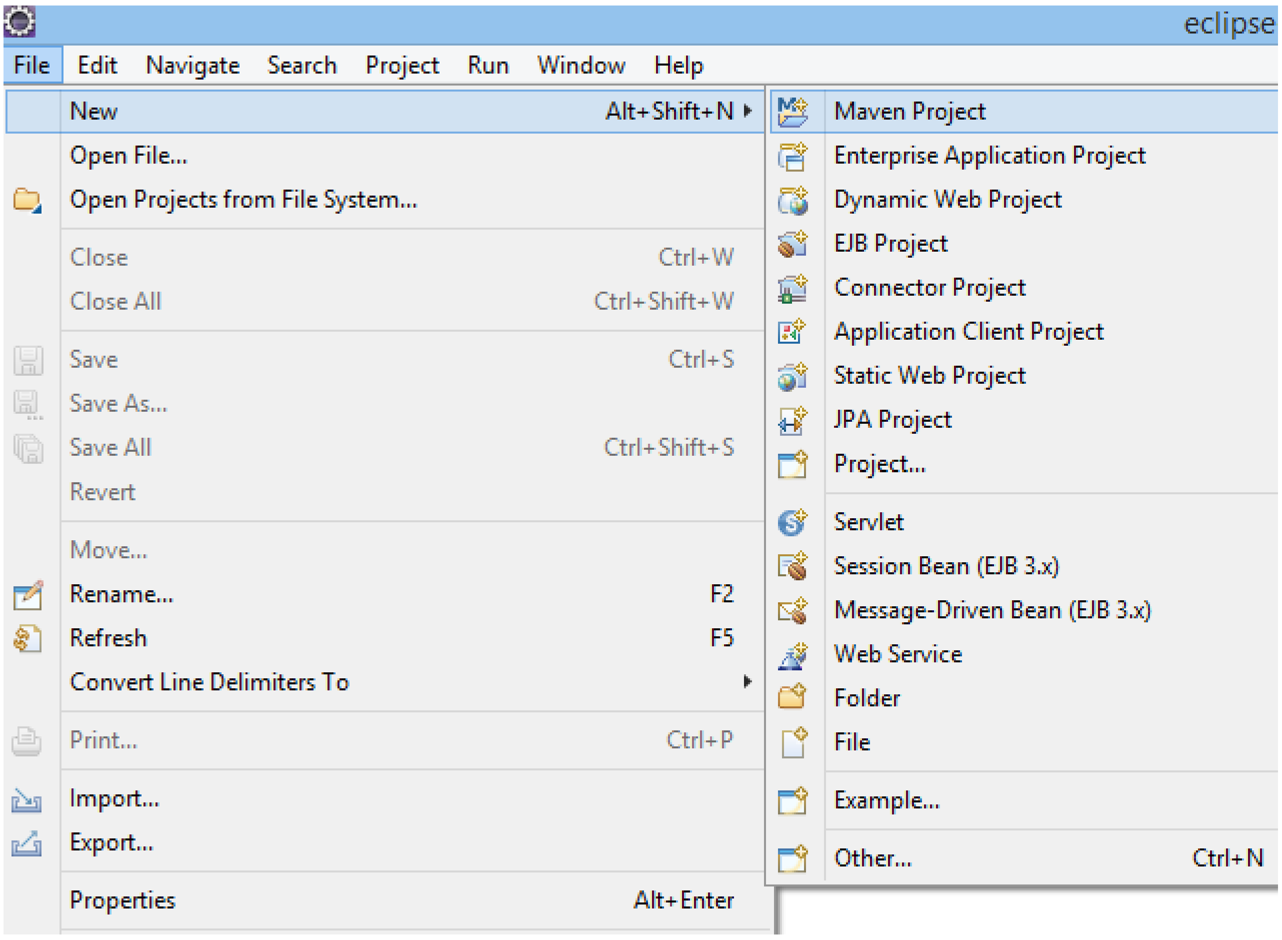
To run the hibernate application, right click on the StoreData class - Run As – Java Application.

Example to create the hibernate application with Annotation

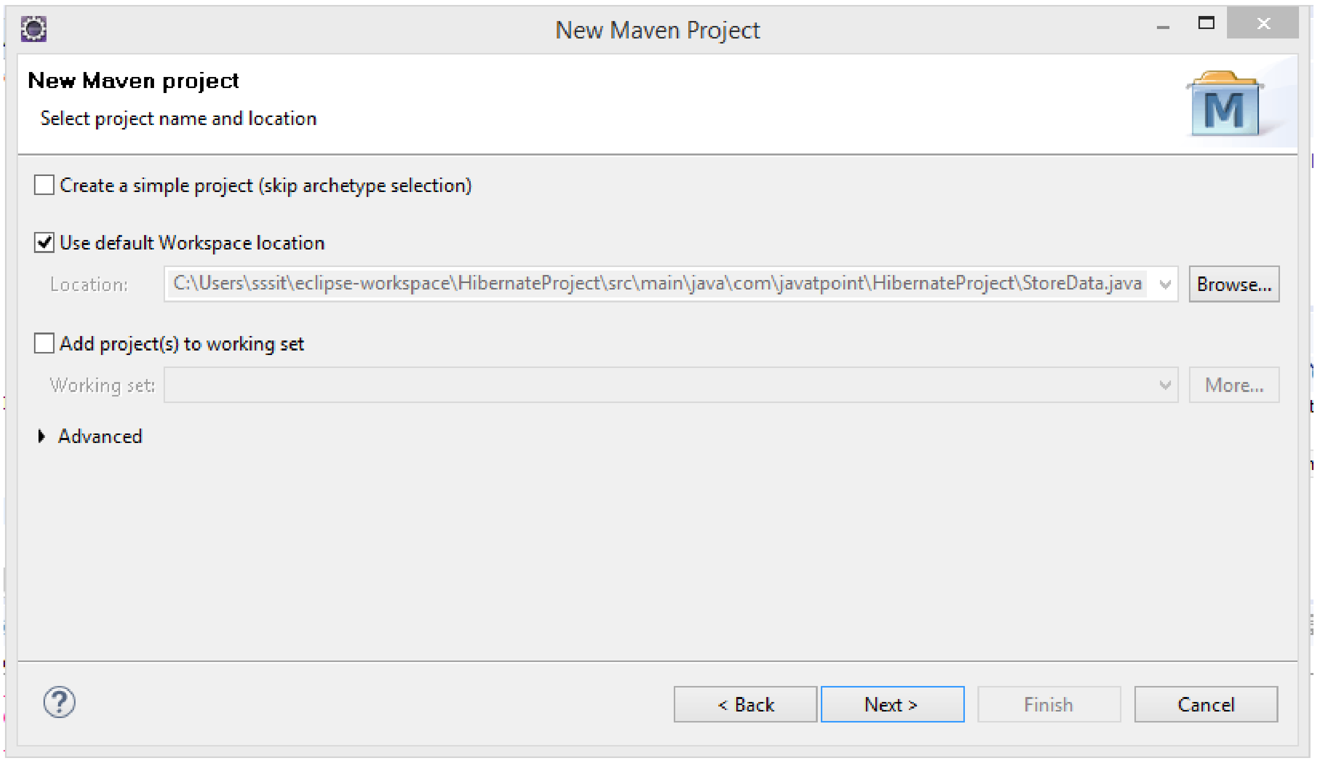
Here, we are going to create a maven based hibernate application using annotation in eclipse IDE. For creating the hibernate application in Eclipse IDE, we need to follow the below steps:

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

New Maven project

Select project name and location

[ ] Create a simple project (skip archetype selection)

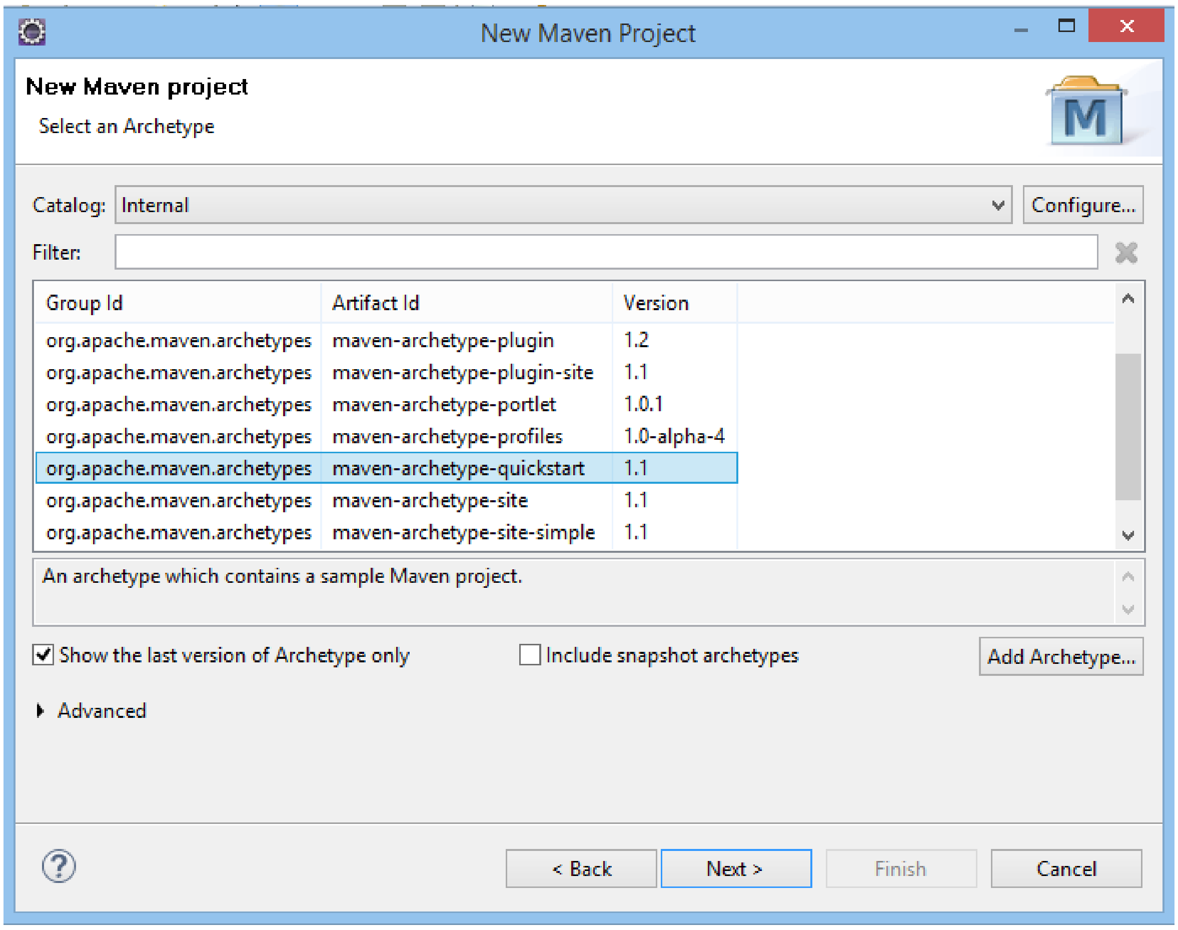
[✓] Use default Workspace location

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**.



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

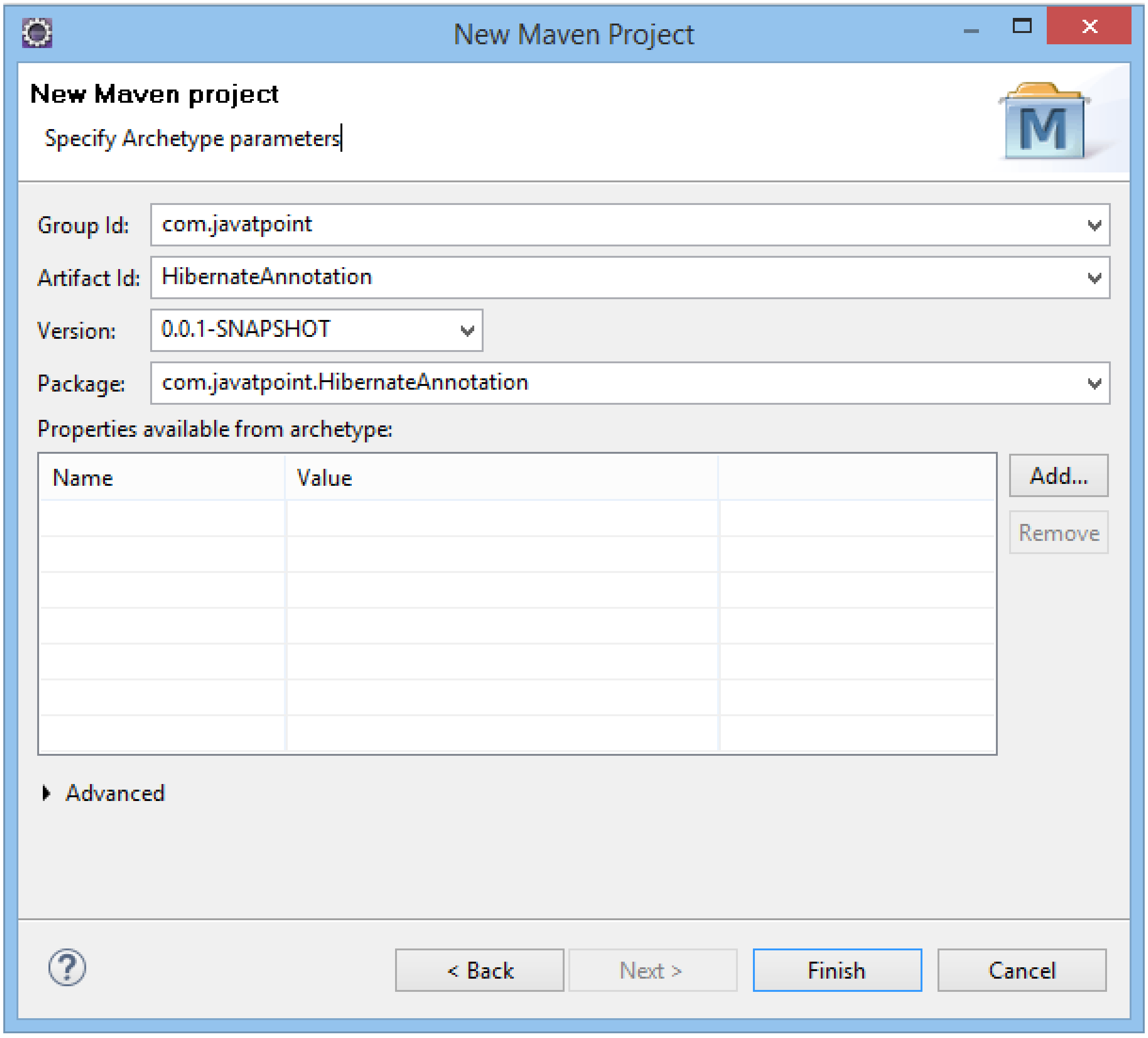
An archetype which contains a sample Maven project.

[✓] 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>
```

3) Create the Persistence class.

Here, we are creating the same persistent class which we have created in the previous topic.

@Entity – use to mark any class as Entity.

@Table – use to change the table details.

@Id- use to mark column as id(primary key).

@GeneratedValue- hibernate will automatically generate values for that using an internal sequence.

Therefore we don’t have to set it manually.

@Column-Can be used to specify column mappings. For example, to change the column name

in the associated table in database.

@Transient-This tells hibernate not to save this field.

@Temporal- @Temporal over a date field tells hibernate the format in which the date needs to be saved

@Lob-@Lob tells hibernate that this is a large object, not a simple object.

@OneToOne , @OneToMany , @ManyToOne, @JoinColumn etc.

To create the Persistence class, right click on **src/main/java - New - Class -** specify the class name with package - **finish**.

4) Create the Configuration file

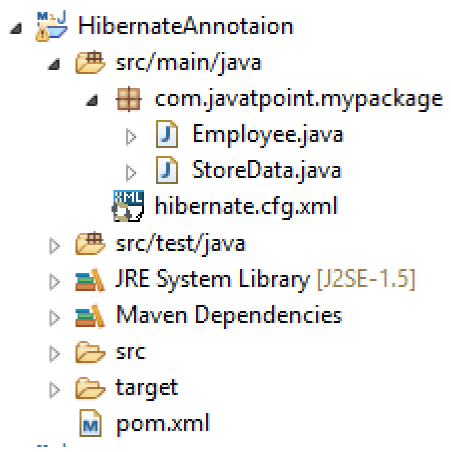
To create the configuration file, right click on **src/main/java - new - file** - specify the file name (e.g. hibernate.cfg.xml) - **Finish**.

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

StoreData.java

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**.

HQL (HIBERNATE QUERY LANGUAGE)

NEED :-

1. To get Data in Hibernate
2. To update or Delete
3. To Fetch Complex Data
4. To join two Tables
5. It Supports SQL
6. HQL is database Independent
7. Suitable for Programmers
8. It uses Entity not Database

SELECT QUERY

```
String query= “from Employee”

Query q1= session.createQuery(query)

List <Employee>list =q1.list();

For (Employee e1 :list) {

System.out.println(e1.getLastname());

}

Single Result = uniqueResult

 Multiple Result = list();
```

DELETE QUERY

```
String query = "delete from Employee as e where e.firstname=:y";
            Query q= session.createQuery(query);
            q.setParameter("y", "Arpita");
            int r = q.executeUpdate();

            System.out.println("Number of Records Deleted="+r);
```

UPDATE QUERY

```
String query = "update Employee e set lastname='Agrawal' ,firstname='Raman' where e.firstname=:y";
            Query q= session.createQuery(query);
            q.setParameter("y", "Rahul");
            int r = q.executeUpdate();
            System.out.println("Number of Records Updated="+r);
```

AGGREGATE FUNCTION WITH RETURN TYPES

Select sum(id) from Employee :- long

Select count(id) from Employee : - long

Select max(id) from Employee : - int

Select min(id) from Employee : - int

Select avg(id) from Employee : - double

Example:-

```
String query = "select count(id) from Employee";
                              Query q= session.createQuery(query);
                long sum= (Long)q.uniqueResult();
```