Hibernate

Before we going to discuss about Hibernate, we must know about why those people introduce the Hibernate Framework.

Limitation of the JDBC technology

- JDBC is a DB specific.
- JDBC is not support caching and "Lazy Loading".
- JDBC is technology.
- In JDBC User is responsible for creation and closing the connections.
- JDBC does not support Association.
- Won't generate Primary Key automatically. Etc…

To overcome this limitation **Gavin King** & Team introduce the one framework that's name is **Hibernate**

Hibernate:

General meaning of hibernate is "sleep mode" or "switch off"

Hibernate was started in 2001 by Gavin King

Hibernate is a ORM tool and follows the ORM framework and is used to connect to the database and perform the CRUD operations.

- What is an ORM framework?

Object Relational Mapping (ORM) It is a programming technique which is used to convert data from relational database and object-oriented programming.

Ex :

- Hibernate: JAVA.
- Django ORM: Django.
- SQL Alchemy: Flask.
- Dapper ORM: C#
- JAVA Persistence API(JPA): JAVA

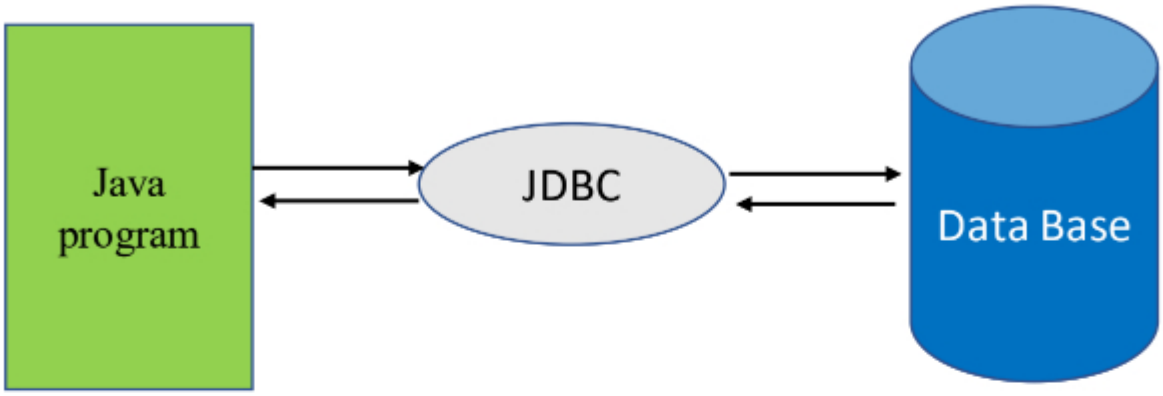
Advantages of Hibernate Framework:

- Open Source and Lightweight. Hibernate framework is open source .
- Fast Performance.
- Database Independent Query.
- Automatic Table Creation.
- Simplifies Complex Join.
- Provides Query Statistics and Database Status.

JPA:

- JPA is stands for Java Persistence API.
- The Java Persistence API (JPA) is **one possible approach to ORM**. Via JPA the developer can map, store, update and retrieve data from relational databases to Java objects and vice versa
- JPA can be used in Java-EE and Java-SE applications (already we are download the java-EE perspective and normally we are using java-SE perspective).

Before we are using Hibernate what is the Data flow:

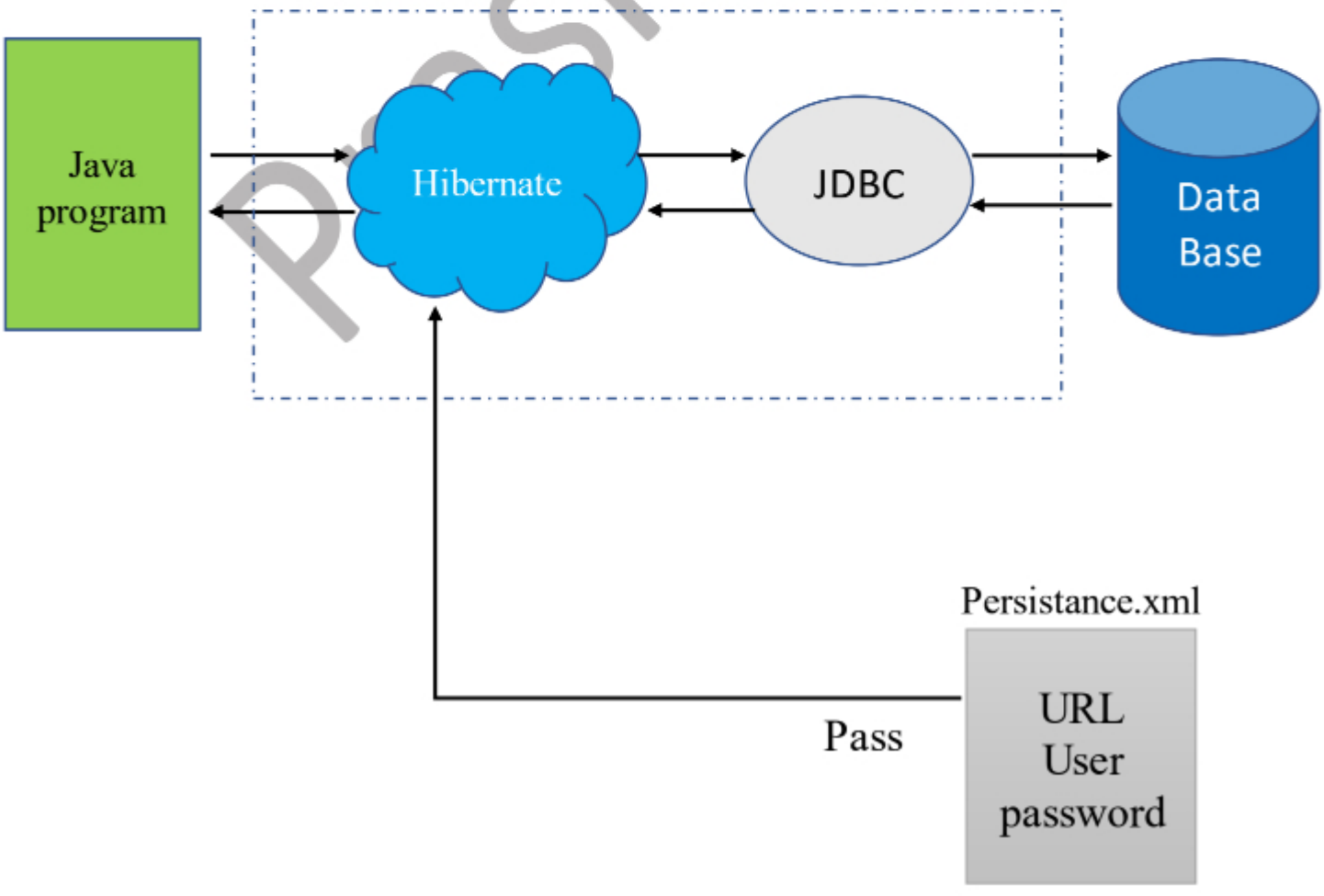


After Adding the Hibernate Frame work into Project:

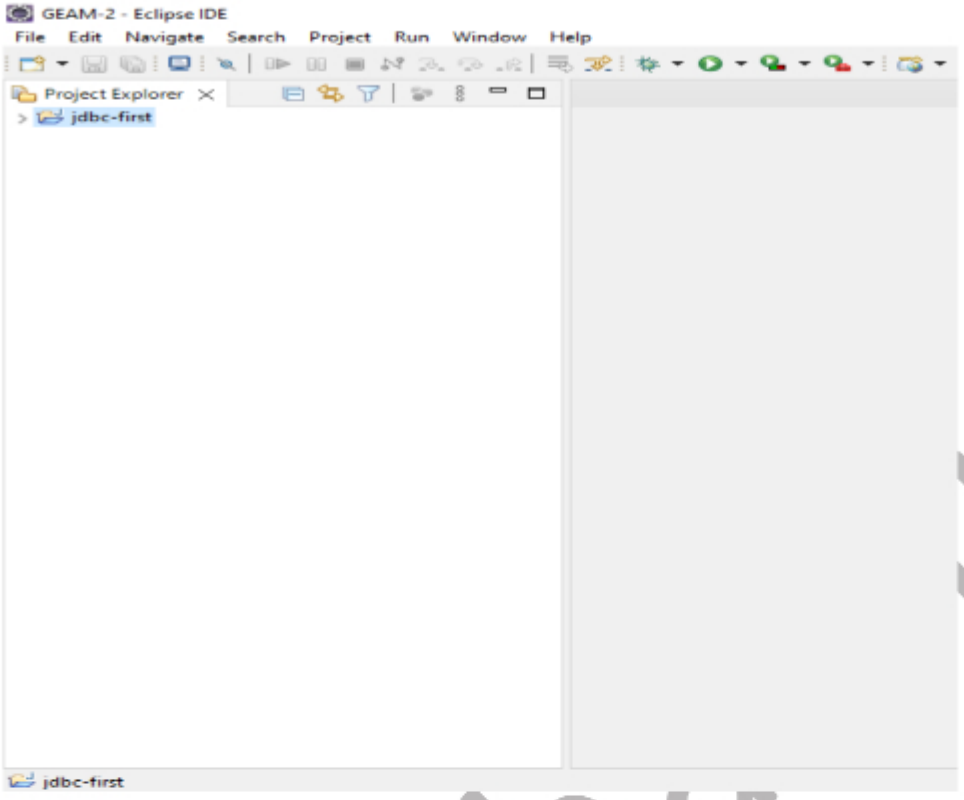
Note:

Before going to the data flow how can we add the Hibernate project

1. If we create a normal java project then we are using plugin process
2. If we are using maven project then we are using dependency tags

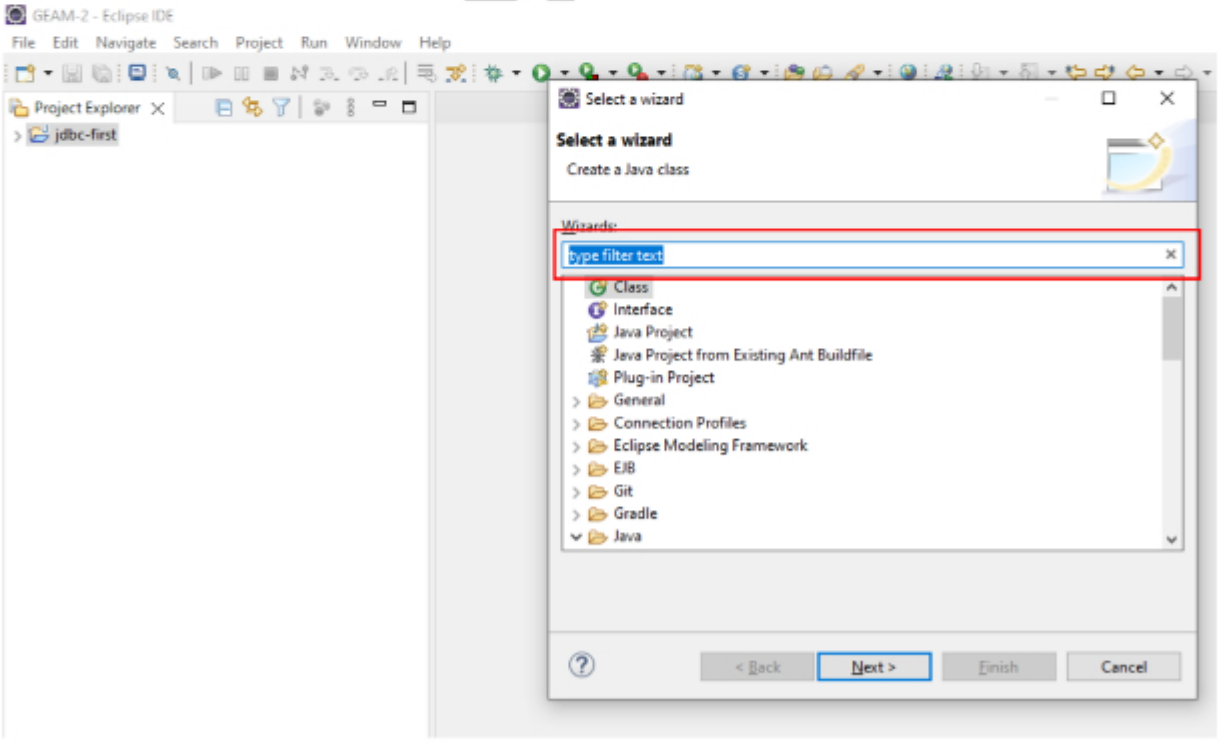


How Can we Create the Maven Project?

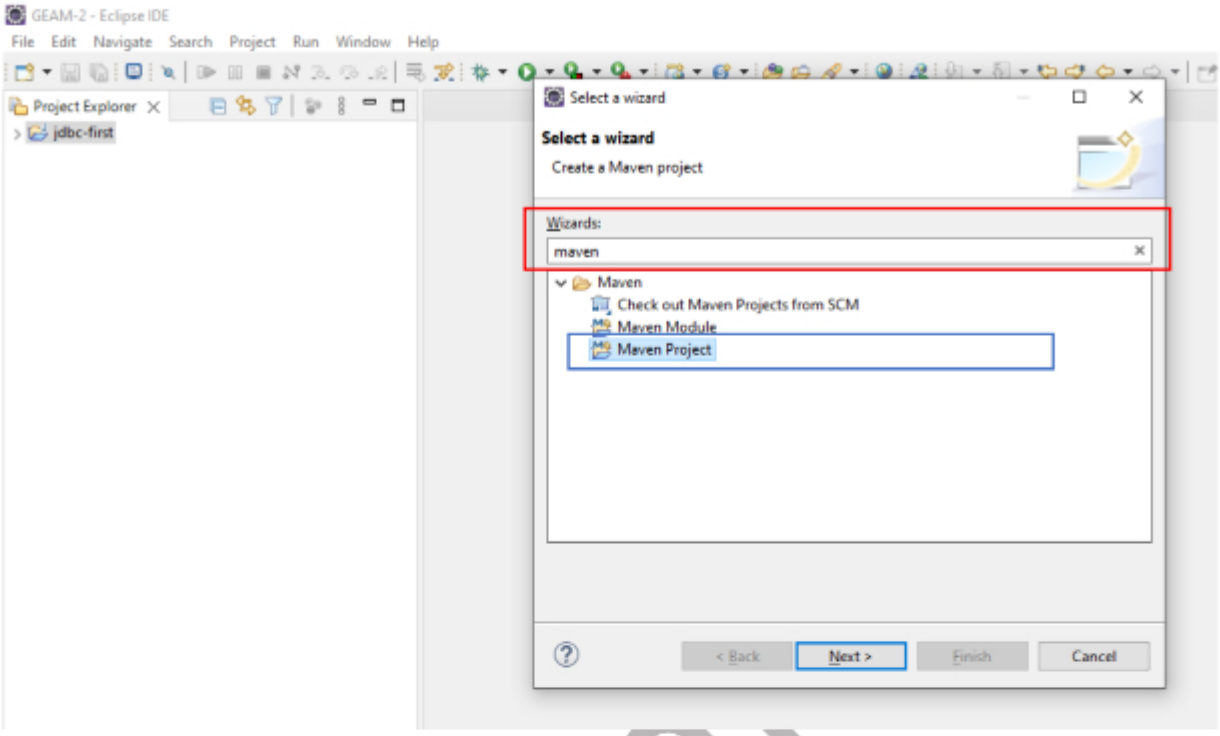


Click the Project Explorer

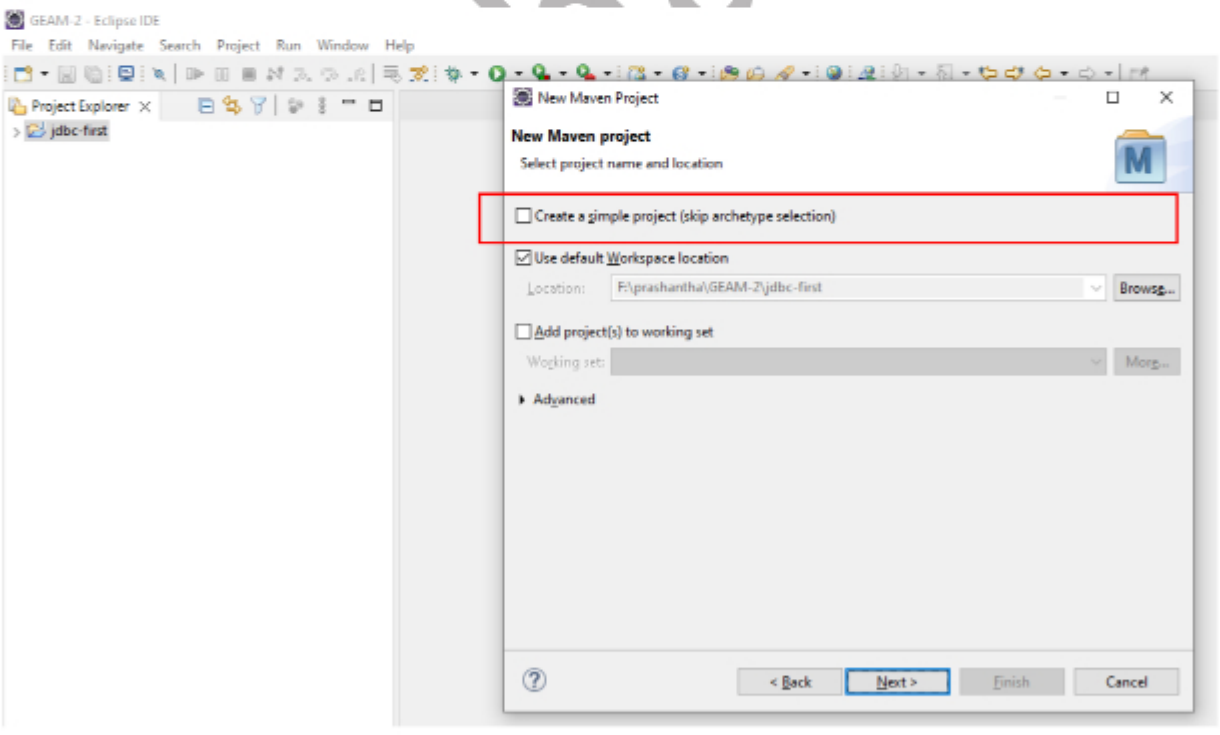
Ctrl+n



In that wizards type Maven

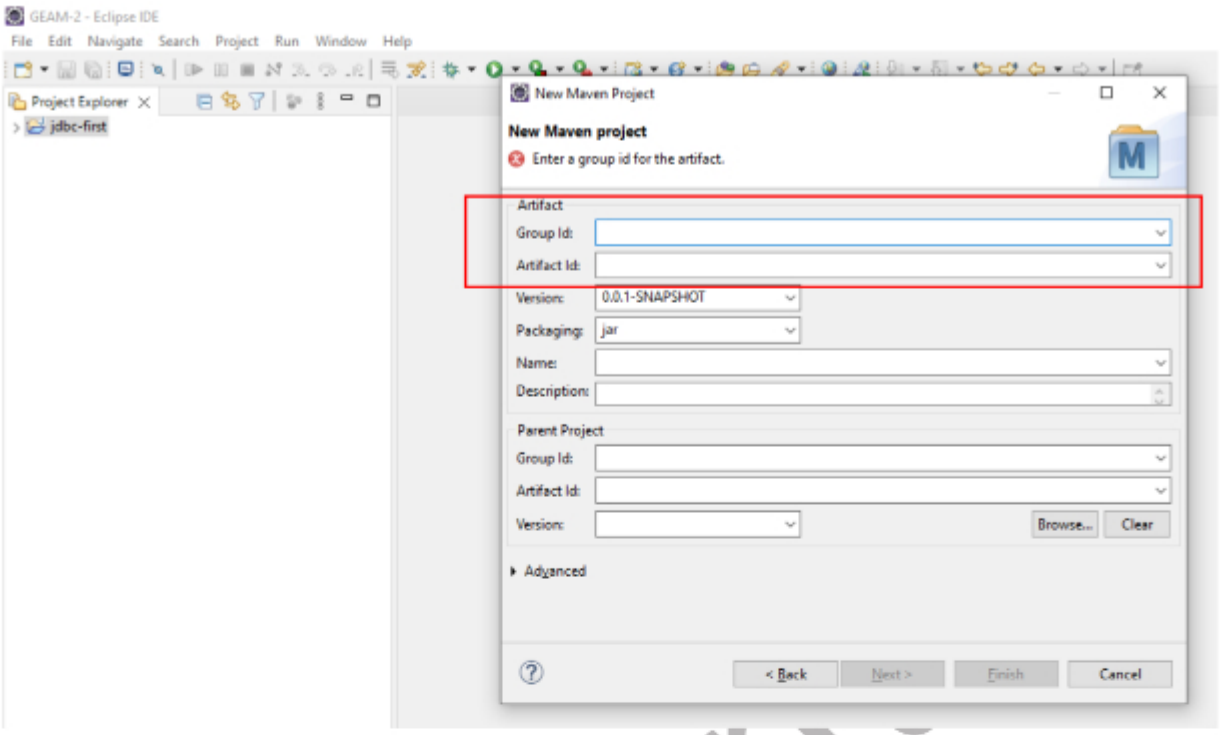


Select the Maven project and click the next



Check the checkbox for create a simple project

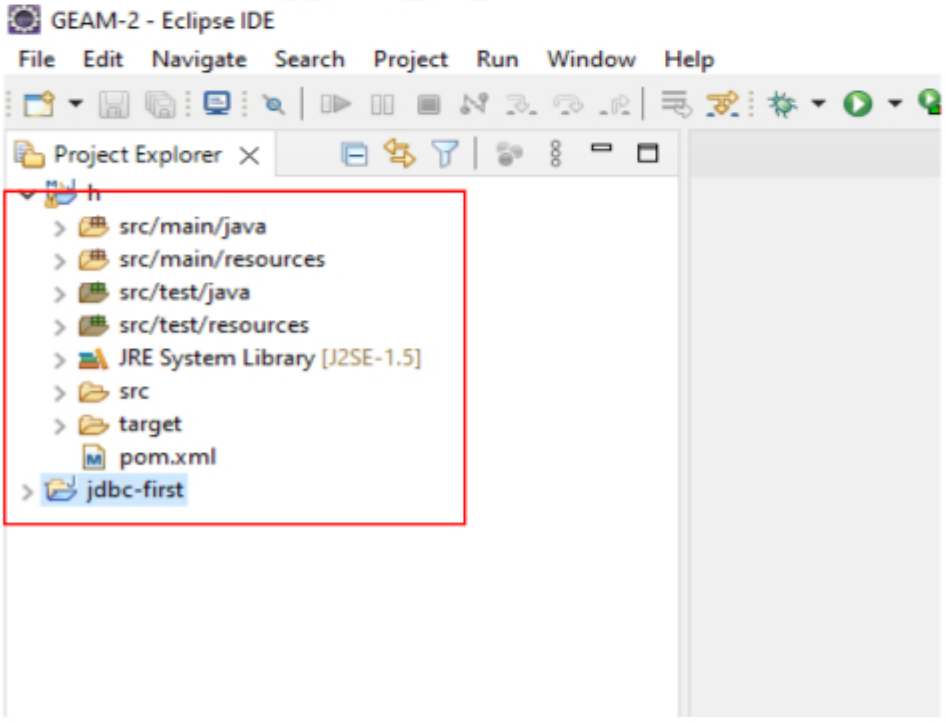
Click the next button



Group Id: give your basic package name

Artifact Id: give your Project name

Click the finish button then it will tack some time for creating the Maven project after finishing the building in your project explorer it will show the Maven project name



Inside the pom.xml we are adding dependency:

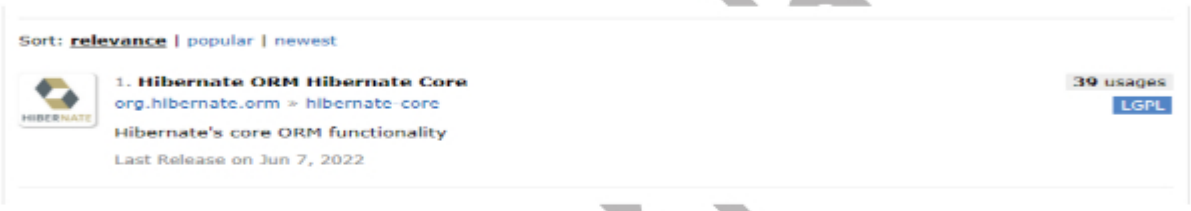
With help of the mvnrepository website only we are downloading the dependency

Link maven repository website - https://mvnrepository.com

Hibernate Dependency name:

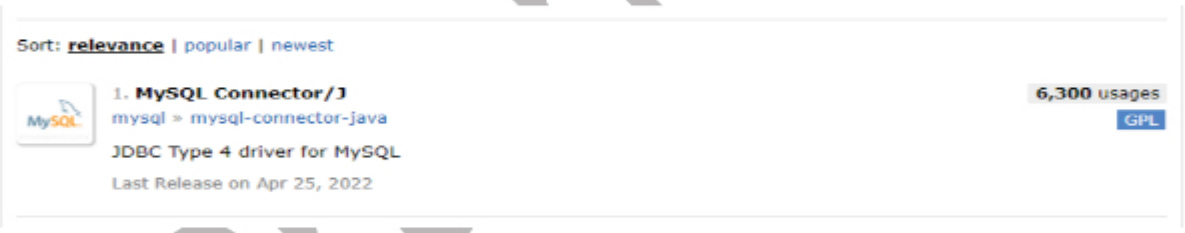
Hibernate Core Relocation and use 5.6.10 version

Link: https://mvnrepository.com/artifact/org.hibernate/hibernate-core/5.6.10.Final



MySql Dependency name:

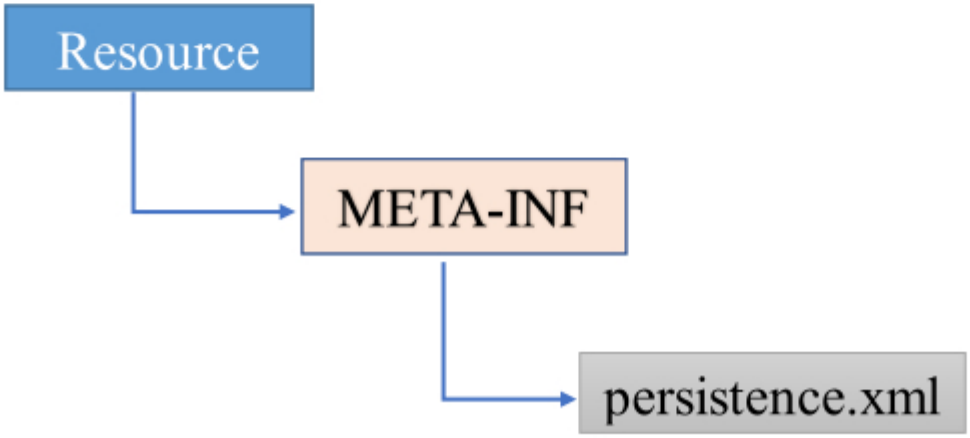
Mysql Connector/J and use 8.0.31 version



Persistence File:

It is a file which is the extension with .xml file and which is used to store a configurable data.

Where we create the Persistence.xml file



Persistence.xml file link:

www.github.com/Prashi974386 → supporting file → persistence → please remove for cache (25,26,27).

Entity Class:

A JPA entity class is a **POJO** (Plain Old Java Object) class, i.e. an ordinary Java class that is marked (annotated) as having the ability to represent objects in the database.

@Entity annotation marks this class as an entity bean.

Requirements for Entity Classes:

- The class must be annotated with the javax.persistence.Entity annotation.
- The class must have a public or protected, no-argument constructor.
- The class and members of the class must not be declared final.
- All the data member should be prefixed with private accesses modifier and all the data member having getter and setter method.

@Id: we use this annotation to make data member as a primary key.

@GeneratedValue(): we use this annotation to automatically set the value to primary key and it takes two parameter **Strategy** and **generator**

Program Ex:

```
package com.jsp.dto;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class Person {
    @Id
    @GeneratedValue (strategy =GenerationType.IDENTITY)
    private int id;
    private String name;

    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;
    }
}
```


If u want crud operation with help of the Hibernate on that time we are using three Elements

- EntityManagerFactory
- EntityManager
- EntityTransaction

EntityManagerFactory:

- EntityManagerFactory is an interface present in javax. persistence package, which is used to create connection between java application to Database and in this level only load or Register Driver is done.
- EntityManagerFactory is used to provide an EntityManager

How can we create EntityManagerFactory Object...?

EntityManagerFactory entityManagerFactory =

Persistence.createEntityManagerFactory("name of the persistence unit")

Persistence: is a helper class and it is present inside a javax. persistence package and with help of the createEntityManagerFactory () only we can get an EntityManagerFactory object.

We pass the persistence unit name inside createEntityManagerFactory method.

EntityManager

- EntityManager is an interface and it is present inside a javax.persistence package which is used to perform a crud operation.
- EntityManager is used to provide an EntityTransction.

How can we create EntityManager Object...?

```
EntityManager entityManger =
        entityManagerFactory.createEntityManager();
```

reference of the EntityManagerFactory

- entityManagerFactory.createEntityManager() is returned Object of EntityManager.

EntityTransaction

- EntityTransaction is an interface and it is present inside a javax.persistence package which is used to manage the transaction.
- With help of the EntityManager we get EntityTransaction Object.

```
EntityTransaction entityTransaction =
                    entityManger.getTransaction();
```

- When we use EntityTransaction object:

  When we are calling Non-select query method on that EntityTransaction object is mandatory.

```java
package com.jsp.demo;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Demo1 {
    public static void main(String[] args) {
        User user = new User();
        user.setId(4);
        user.setName("Lakku");
        user.setAge(21);

        EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("prashi");
        EntityManager entityManager =
entityManagerFactory.createEntityManager();
        EntityTransaction entityTransaction =
entityManager.getTransaction();

        entityTransaction.begin();
        entityManager.persist(user);
        entityTransaction.commit();

        System.out.println("Data Saved");
    }

}
```

Some Important Inbuilt method in Hibernate

Method	Description
persist (Object entity)	It is used to save the object into a database
remove (Object entity)	It is used to remove the record from database based on the primary key of that database
find(class<T> entity class, Object entity)	It is fetching the record from database based on the primary key of that database
merge (T entity)	Merge method is used into two ways when find the record with help the find () then object is existing that time merge method works an update the object else its acts like save new object to database.
createQuery (String s)	With help of this method we can write our own query and its return Query interface object

Query Parameter:

- Query parameters are a way to build and execute parameterized queries
- With help of this Query Parameter we are passing our own Dynamic Query into Hibernate

- Similar to JDBC prepared statement parameters, JPA specifies two different ways to write parameterized queries by using:
  - Positional parameters
  - Named parameters

Positional parameters

we declare these parameters within the query by typing a question mark, followed by a positive integer number

With help of this Query Parameter we are passing our own Dynamic Query into Hibernate with help of position we are passing values for query.

Ex:

String sql = "select u from User u where u.email=?1 and u.password=?2";

Query query = entityManager.cretaeQuery(sql);

query.setParameter (1, values);

Named parameters

we declare these parameters within the query by typing a colon, followed by a reference name.

With help of this Query Parameter we are passing our own Dynamic Query into Hibernate with help of key name we are passing values for query.

Ex:

String sql = "select u from User u where u.email=:email and u.password=:password";

Query query = entityManager.cretaeQuery(sql);

query.setParameter ("email", values);

Mapping in Hibernate:

- It is the one of the key features of Hibernate.
- A Hibernate mapping is the bridge between a database table structure and an Object-Oriented Domain Model.

Hibernate supports various mapping styles:

- Simple Mapping
- Collection Mapping
- Inheritance Mapping
- Association mapping etc...

Association mapping:

In association mapping mainly 4 mapping style are there

- One-to-One mapping
- One-to-Many mapping
- Many-to-One mapping
- Many-to-Many mapping

One-to-One mapping:

One to one mapping represents that a single entity is associated with a single instance of the other entity.

Ex:

One person has one passport, a passport is associated with a single person

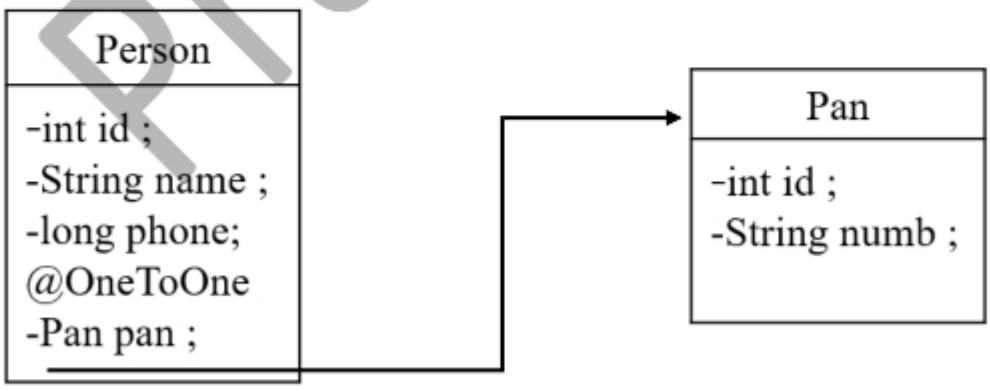
We have one employee ID, a employee ID is uniquely associated with a person

In database management systems one-to-one mapping is of two types

- One-to-one unidirectional
- One-to-one bidirectional

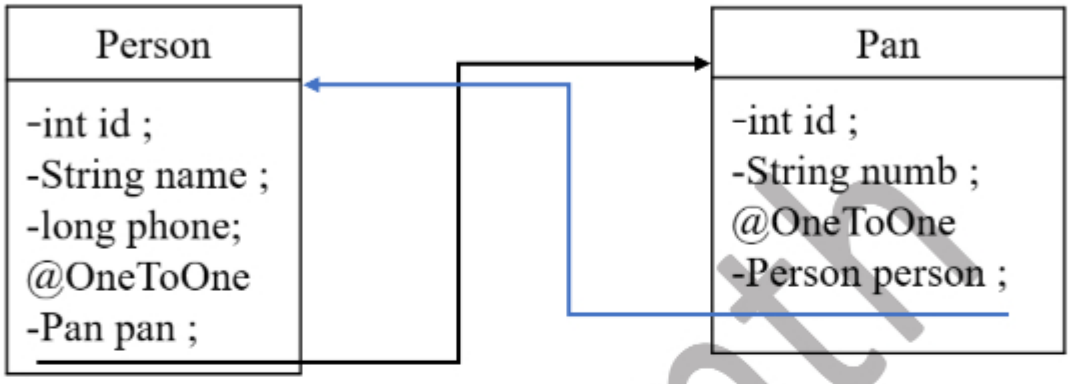
One-to-one unidirectional:

- In this type of mapping one entity has a property or a column that references to a property or a column in the target entity.



One-to-one bidirectional:

- In One-To-One Bidirectional Shared primary key mapping, **two tables share the same primary key**. The Bidirectional relationship means navigation is possible in both direction



- After One-to-one unidirectional mapping in a database two Table will be created, In that table which table having annotation **@OneToOne** in that class table only Foreign key row will be created.

@OneToOne : we use this annotation marks the relationship between two entities with one-to-one multiplicity.

@Column( ) : we use this annotation for set the column name.

Parameter of Column annotation

- **name** attribute permits the name of the column.
- **nullable** attribute permits the column to be marked NOT NULL when the schema is generated.
- **unique** attribute permits the column to be marked as containing only unique values.

- After One-to-one bidirectional mapping in a database two Table will be created, In that table which table having annotation **@JoinColumn** in that class table only Foreign key row will be created and other class Table must be having **mappedBy**

@JoinColumn : annotation to map the foreign key column of a managed association.

mappedBy attribute indicates that which entity owns the relationship and what reference is used for non-owning entity within owner entity.

Ex: Branch Have anAddrress

```
package com.jsp.dto;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

@Entity
public class Branch {
	@Id
	@GeneratedValue(strategy  = GenerationType.IDENTITY)
	private int id;
	private String name;
	@OneToOne(mappedBy = "branch")
	Address address;

	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;
	}
}
```

```
package com.jsp.dto;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;

@Entity
public class Address {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	private String city;

	@OneToOne
	@JoinColumn
	Branch branch;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
}
```

One-to-Many mapping or Many-to-One mapping:

- one to many mapping means that **one row in a table can be mapped to multiple rows in another table**
- many to one mapping means that **many rows in a table can be mapped to one row in another table**.

Ex :

One person has many ATM cards, a ATM card is associated with only a single person.

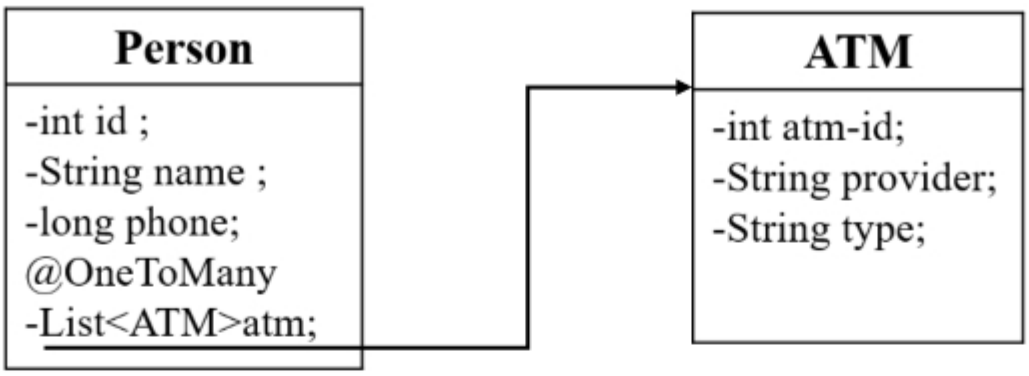
One Mobile have many sim cards, a sim card is associated with only a single Mobile.

In database management systems one-to-one mapping is of two types

- Unidirectional
  - One-to-Many unidirectional
  - &
  - Many-to-One unidirectional
- Bidirectional
  - One-to-Many unidirectional
  - or
  - Many-to-One unidirectional

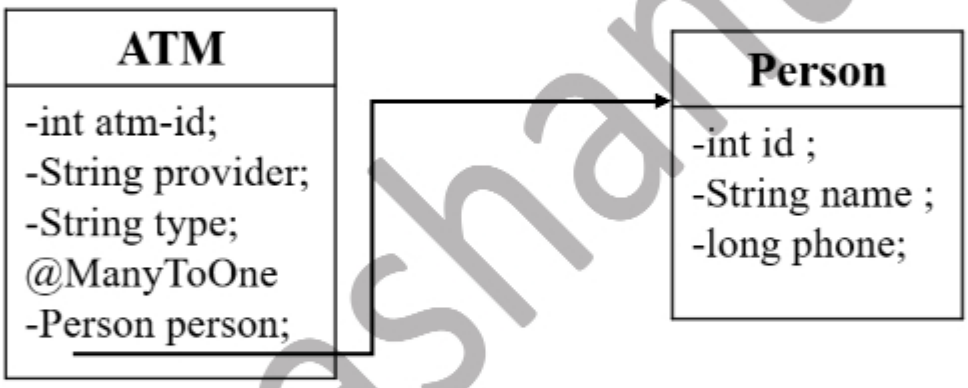
One-to-many unidirectional:

- In one-to-many unidirectional type of mapping one row in one table can have more than one matching row in another table.



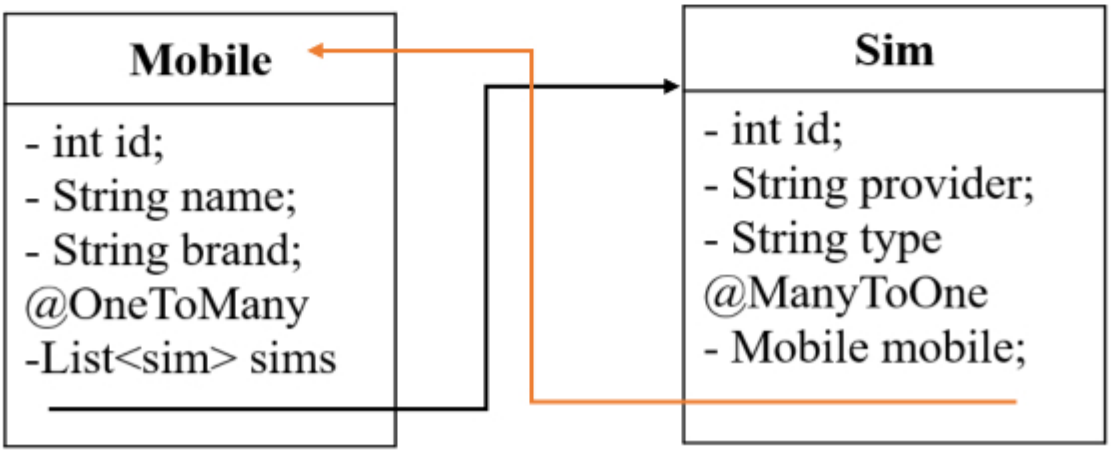
Many-to-one unidirectional:

- A many-to-one association is the most common kind of association where an Object can be associated with multiple objects



One-to-many bidirectional/Many-to-one bidirectional:

- Many-To-One Bidirectional mapping is exactly same as Many-To-One Unidirectional Mapping. **One table has a foreign key column that references the primary key of associated table.** In Bidirectional relationship, both side navigation is possible.



- After One-to-many unidirectional mapping in a database three table will be created, In that three table one table for one class and another table for another class and third table having two column one for first table primary key and other one for second table primary key , with help of the third table we can join the first and second table.
- After many-to-one unidirectional mapping in a database two Table will be created, In that table which table having annotation **@ManyToOne** in that class table only Foreign key row will be created.
- After One-to-many bidirectional or Many-to-one bidirectional mapping, If we declare the **@JoinColumn** below the **@OneToMany** in that same class, in a database three table will be created and in those three tables one table of one class and another table of another class and third table having two columns, one for the first table primary key and other one for the second table primary key , with help of the third table we can join the first and second table.
- Else we declare the **@JoinColumn** below **@ManyToOne** in that same class, In a database two table will be created , In that table which table having annotation **@ManyToOne** in that class table only Foreign key row will be created.

Many-to-Many mapping :

@ManyToMany is used to connect the entities(many entities associated with the many entities).

Ex :

customers can purchase various products, and products can be purchased by many customers.

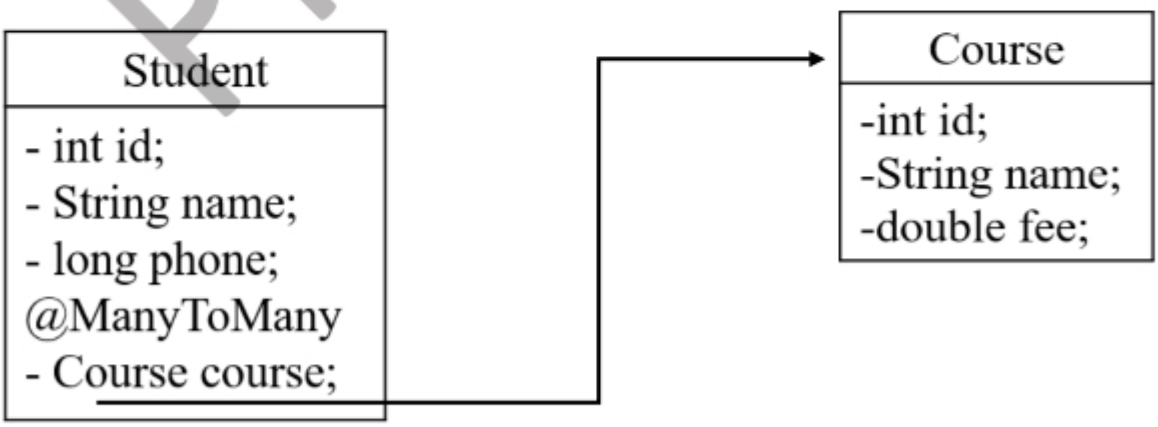
Student can select so many course and course having so many students.

In database management systems one-to-one mapping is of two types

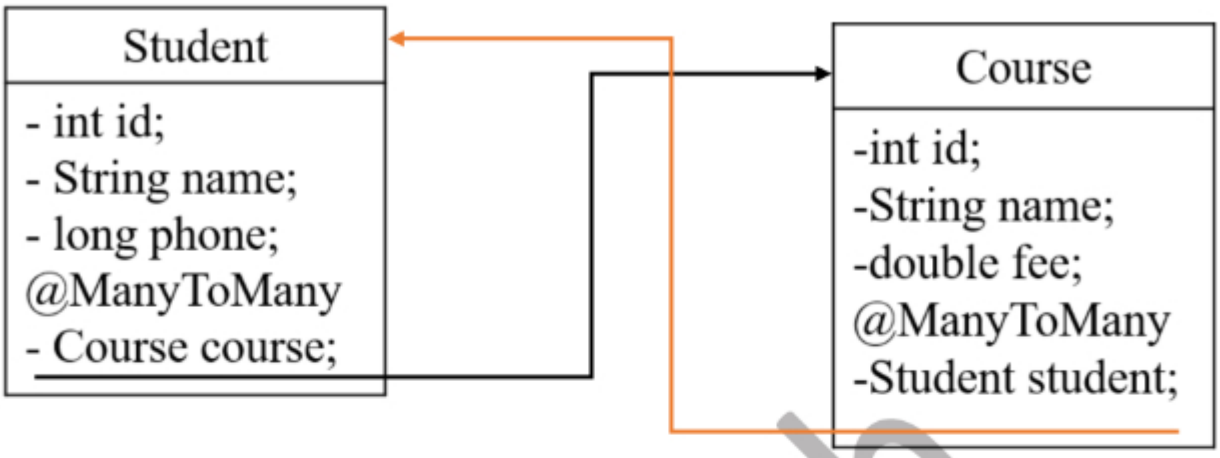
- Many-to-Many unidirectional
- Many-to-Many bidirectional

Many-to-Many unidirectional:

- In a relational database, many-to-many relationship, a row in table first can have more than one matching row in table second, a row in table second can have more than one matching row in table first.



Many-to-Many bidirectional:



- After many-to-many unidirectional mapping, In a database three tables will be created, In those three tables one table for one class and another table for another class and third table for having two column one for first table primary key and other one for second table primary key, with help of the third table we can join the first and second table.
- After many-to-many bidirectional mapping, In a database four tables will be created, In those four tables one table for one class and another table for another class and third table having two column one for first table primary key and other one for second table primary key and fourth table having two column one for second table primary key and other one for first table primary key with help of the third table we can join the first and second table. If we use in any one entity class @JoinTable , at that time In database third table and forth table both are merged.

@JoinTable : we use this annotation for merging the tables in databas