

Hibernate Example with MySQL Database

Development Steps

1. Create a Simple Maven Project
2. Project Directory Structure
3. Add jar Dependencies to pom.xml
4. Creating the JPA Entity Class(Persistent class)
5. Create a Hibernate configuration file - Java Configuration
6. Create StudentDao Class
7. Create the Main class and Run an Application

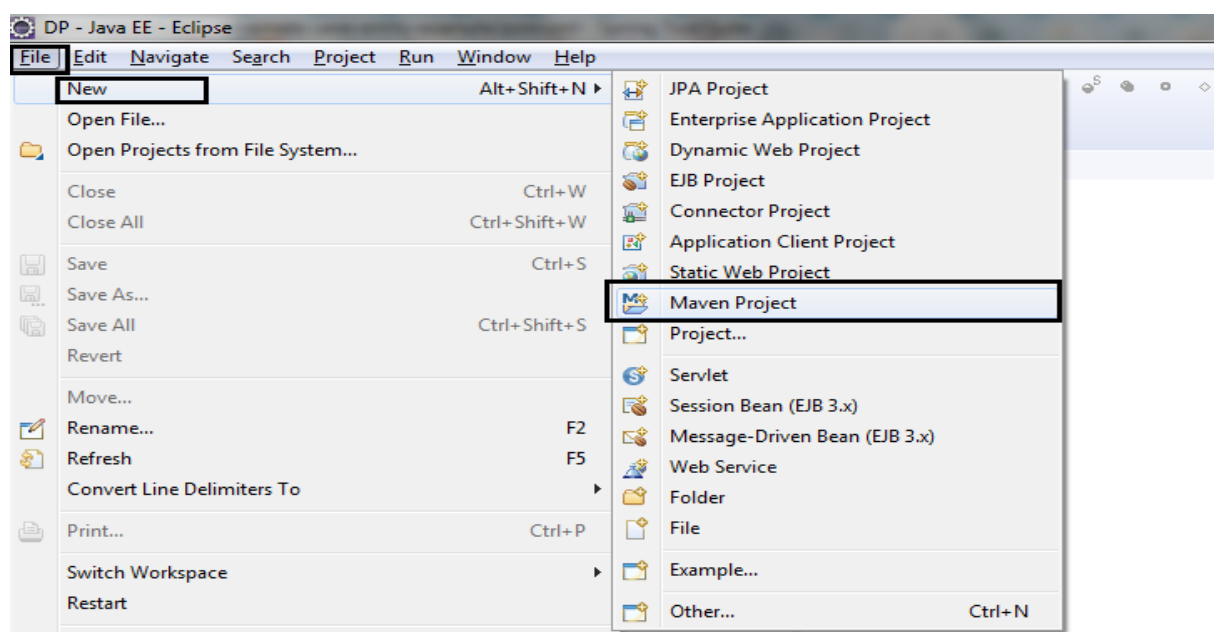
1. Create a Simple Maven Project

Use the **How to Create a Simple Maven Project in Eclipse** article to create a simple Maven project in Eclipse IDE.

Let's create step by step a simple maven project in Eclipse IDE.

Step-1

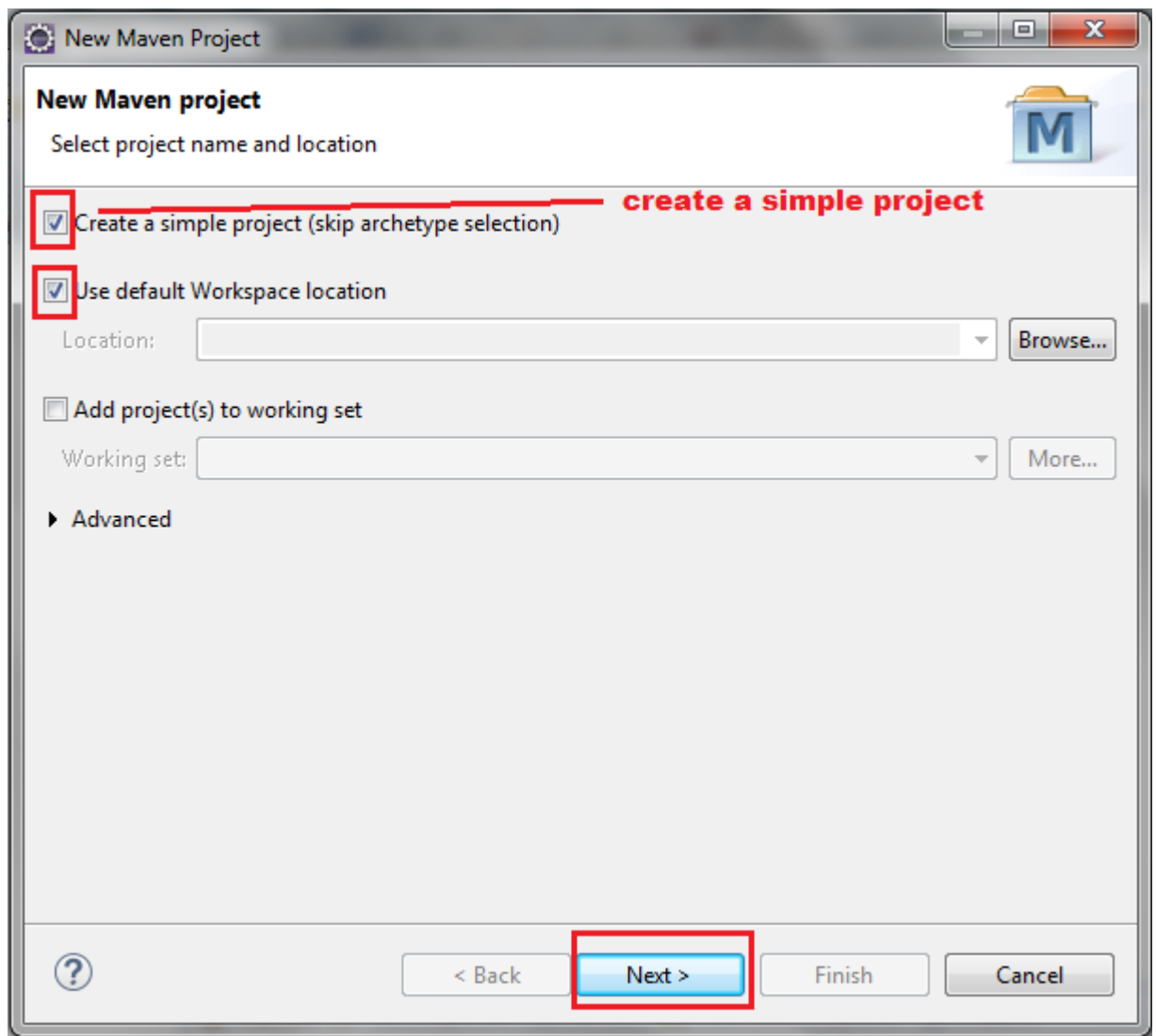
- Open Eclipse
- Click on *File* -> *New* -> *Maven Project*



Step-2

Click on Checkbox for both

- Create a simple project
- Use default Workspace location
- Click on next button



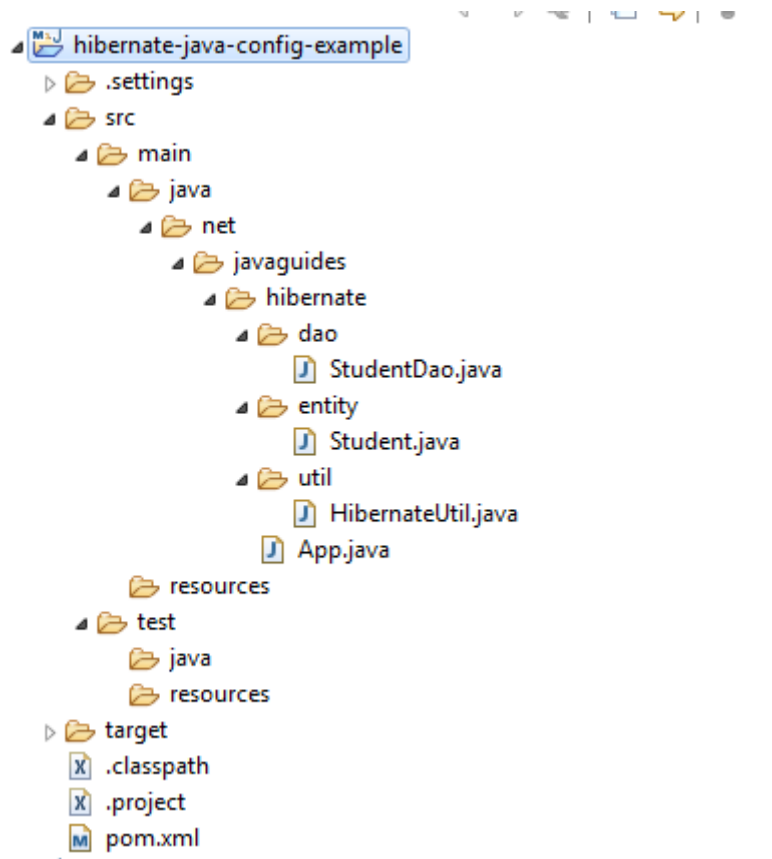
Step-3

Provide GroupId and ArtifactId in next screen.

- **GroupId:** com.cg.hibernatemavendemo
- **Artifact Id:** hibernate-maven-demo-project
- **Name:** hibernate-maven-demo-project
- **Description:** Sample Maven Demo Project Using MySQL

2. Project Directory Structure

The project directory structure for your reference -



3. Add jar Dependencies to pom.xml

Open the *pom.xml* file in your Hibernate project and add the below code to it:

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId> com.cg.hibernatejavademodemo </groupId>
    <artifactId> hibernate-maven-demo-project </artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>hibernate-java-config-example</artifactId>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```

</properties>
<dependencies>
  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.13</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.7.Final</version>
  </dependency>
</dependencies>
<build>
  <sourceDirectory>src/main/java</sourceDirectory>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>

```

Note that we are using *mysql-connector-java* driver dependency to connect Java Hibernate application to MySQL database:

```

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.13</version>
</dependency>

```

We are using Hibernate core dependency:

```

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.3.7.Final</version>
</dependency>

```

We are using the *maven-compiler-plugin* to compile Java Hibernate application with JRE 1.8 version:

```

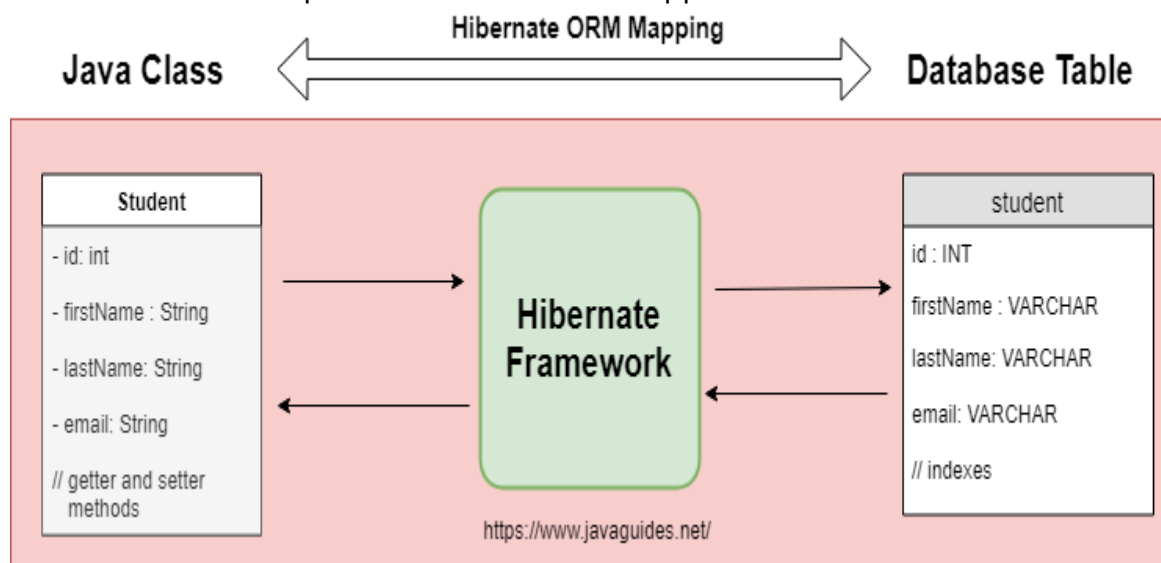
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>

```

```
<version>3.5.1</version>
<configuration>
  <source>1.8</source>
  <target>1.8</target>
</configuration>
</plugin>
```

4. Creating the JPA Entity Class(Persistent class)

Let's create a *Student* persistent class that is mapped to a student database table:



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 the *newInstance()* method.
- **Provide an identifier property:** It is better to assign an attribute as id. This attribute behaves as a primary key in a 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, which depends on the persistent class. The application programmer will not be able to use proxies for lazy association fetching.

Create a *Student* entity class under *net.javaguides.hibernate.entity* package as follows.

```
package com.cg.hibernate.mavendemo.entity;
```

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "student")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    public Student() {

    }

    public Student(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    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;
    }
}
```

```

    }

    public String getEmail() {
        return email;
    }

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

    @Override
    public String toString() {
        return "Student [id=" + id + ", firstName=" + firstName + ", lastName=" +
lastName + ", email=" + email + "]";
    }
}

```

JPA annotations are used in mapping java objects to the database tables, columns, etc.

JPA annotations that we are using in the *Student* entity:

@Entity - This annotation specifies that the class is an entity.

@Table - This annotation specifies the table in the database with which this entity is mapped.

@Column - The @Column annotation is used to specify the mapping between a basic entity attribute and the database table column.

@Id - This annotation specifies the primary key of the entity

@GeneratedValue - This annotation specifies the generation strategies for the values of primary keys.

5. Create a Hibernate configuration file - Java Configuration

The *HibernateUtil* Java configuration file contains information about the database and mapping file.

Let's create a *HibernateUtil* file and write the following code in it.

```

package com.cg.hibernatejavendemo.util;

import java.util.Properties;

import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

```

```

import org.hibernate.cfg.Environment;
import org.hibernate.service.ServiceRegistry;

import net.javaguides.hibernate.entity.Student;

public class HibernateUtil {
    private static SessionFactory sessionFactory;
    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            try {
                Configuration configuration = new Configuration();

                // Hibernate settings equivalent to hibernate.cfg.xml's properties
                Properties settings = new Properties();
                settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
                settings.put(Environment.URL,
"jdbc:mysql://localhost:3306/hibernate_db?useSSL=false");
                settings.put(Environment.USER, "root");
                settings.put(Environment.PASS, "root");
                settings.put(Environment.DIALECT,
"org.hibernate.dialect.MySQL5Dialect");

                settings.put(Environment.SHOW_SQL, "true");

                settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

                settings.put(Environment.HBM2DDL_AUTO, "create-drop");

                configuration.setProperties(settings);

                configuration.addAnnotatedClass(Student.class);

                ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder()
                    .applySettings(configuration.getProperties()).build();

                sessionFactory =
configuration.buildSessionFactory(serviceRegistry);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return sessionFactory;
    }
}

```

6. Create StudentDao Class

Let's create a separate *StudentDao* class to separate out hibernate and database-related stuff.

```

package com.cg.hibernate.mavendemo.dao;

import java.util.List;

```



```

import org.hibernate.Session;
import org.hibernate.Transaction;

import net.javaguides.hibernate.entity.Student;
import net.javaguides.hibernate.util.HibernateUtil;

public class StudentDao {
    public void saveStudent(Student student) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            // start a transaction
            transaction = session.beginTransaction();
            // save the student object
            session.save(student);
            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public List < Student > getStudents() {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("from Student", Student.class).list();
        }
    }
}

```

7. Create the main App class and Run an Application

Let's test Hibernate application to connect MySQL database.

```

package com.cg.hibernateavendemo;

import java.util.List;

import net.javaguides.hibernate.dao.StudentDao;
import net.javaguides.hibernate.entity.Student;

public class App {
    public static void main(String[] args) {
        StudentDao studentDao = new StudentDao();
        Student student = new Student("Ramesh", "Fadatare",
"rameshfadatare@javaguides.com");
        studentDao.saveStudent(student);

        List < Student > students = studentDao.getStudents();
        students.forEach(s -> System.out.println(s.getFirstName()));
    }
}

```

```
}  
}
```

Output

```
Nov 22, 2018 8:02:19 PM org.hibernate.Version logVersion  
INFO: HHH0000412: Hibernate Core (5.1.7.Final)  
Nov 22, 2018 8:02:19 PM org.hibernate.cfg.Environment <clinit>  
INFO: HHH0000206: hibernate.properties not found  
Nov 22, 2018 8:02:19 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>  
INFO: HHH000000001: Hibernate Commons Annotations (5.0.4.Final)  
Nov 22, 2018 8:02:20 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure  
WARN: HHH000010002: Using Hibernate built-in connection pool (not for production use!)  
Nov 22, 2018 8:02:20 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator  
INFO: HHH000010005: using driver [com.mysql.cj.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/hibernate_db?useSSL=false]  
Nov 22, 2018 8:02:20 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator  
INFO: HHH000010001: Connection properties: {user=root, password=****}  
Nov 22, 2018 8:02:20 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator  
INFO: HHH000010005: Autocommit mode: false  
Nov 22, 2018 8:02:20 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>  
INFO: HHH00001015: Hibernate connection pool size: 1 (min=1)  
Nov 22, 2018 8:02:20 PM org.hibernate.dialect.Dialect <init>  
INFO: HHH0000400: using dialect: org.hibernate.dialect.MySQL5Dialect  
Hibernate: drop table if exists student  
Nov 22, 2018 8:02:21 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection  
INFO: HHH00001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@a15b709] for  
Hibernate: create table student (id integer not null auto_increment, email varchar(255), first_name varchar(255), last_name varchar(255), primary key (id)) engine=MyISAM  
Nov 22, 2018 8:02:22 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection  
INFO: HHH00001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@a1a53ef] for  
Nov 22, 2018 8:02:22 PM org.hibernate.tool.schema.internal.SchemaCreatorImpl applyImportSources  
INFO: HHH0000476: executing import script 'org.hibernate.tool.schema.internal.exec.ScriptSourceInputNonExistentImpl@20e659aa'  
Hibernate: insert into student (email, first_name, last_name) values (?, ?, ?)  
Hibernate: insert into student (email, first_name, last_name) values (?, ?, ?)  
Nov 22, 2018 8:02:22 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService  
INFO: HHH0000397: Using ASQueryTranslatorFactory  
Hibernate: select student0_.id as id1_0_, student0_.email as email2_0_, student0_.first_name as first_na3_0_, student0_.last_name as last_name4_0_ from student student0_  
Print student email id : rameshfadatar@jvavguides.com  
Print student email id : john@jvavguides.com
```