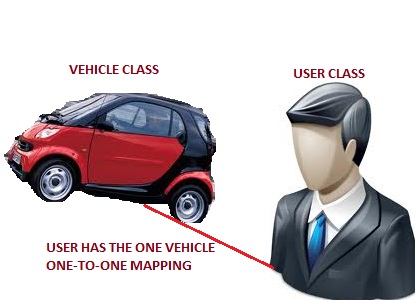
One to One Mapping in Hibernate Example

In this tutorial of One to One Mapping in Hibernate Example we will learning what happens when an **entity class** has the field of the **entity type object**. Means that **one entity is inside the one entity** known as One-2-One Mapping.

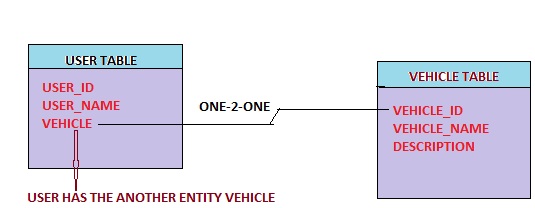


**Hibernate Mapping One-to-One**

In this example you will learn how to map one-to-one relationship using Hibernate. Consider the following relationship between ***UserDetails***and ***Vehicle***entity.



According to the relationship each user should have a unique vehicle.



For that we will use the following annotation.  
**@OneToOne:**  
**Target:**  
Fields (including property get methods)Defines a single-valued association to another entity that has one-to-one multiplicity. It is not normally necessary to specify the associated target entity explicitly since it can usually be inferred from the type of the object being referenced. If the relationship is bidirectional, the non-owning side must use the **mappedBy**element of the **@OneToOne**annotation to specify the relationship field or property of the owning side. The **@OneToOne**annotation may be used within an embeddable class to specify a relationship from the embeddable class to an entity class.

Now we look the following Example related to the One to One mapping.

**1. First Create Vehicle Class**  
 **Vehicle.java**

package com.sdnext.hibernate.tutorial.dto;

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=”VEHICLE”)**  
public class Vehicle  
{  
**@Id  
@GeneratedValue(strategy=GenerationType.AUTO)  
@Column(name=”VEHICLE\_ID”)**  
private int vehicleId;

**@Column(name=”VEHICLE\_NAME”)**  
private String vehicleName;

public int getVehicleId() {  
return vehicleId;  
}  
public void setVehicleId(int vehicleId) {  
this.vehicleId = vehicleId;  
}  
public String getVehicleName() {  
return vehicleName;  
}  
public void setVehicleName(String vehicleName) {  
this.vehicleName = vehicleName;  
}  
}

**2. Create the User Class**  
 **UserDetails.java**

package com.sdnext.hibernate.tutorial.dto;

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

**@Entity  
@Table (name=”USER\_DETAIL”)**  
public class UserDetails  
{  
**@Id  
@Column(name=”USER\_ID”)  
@GeneratedValue(strategy=GenerationType.AUTO)**  
private int    userId;

**@Column(name=”USER\_NAME”)**  
private String userName;

**@OneToOne  
    @JoinColumn(name=”VEHICLE\_ID”)**  
private Vehicle vehicle;

public Vehicle getVehicle() {  
return vehicle;  
}  
public void setVehicle(Vehicle vehicle) {  
this.vehicle = vehicle;  
}  
public int getUserId() {  
return userId;  
}  
public void setUserId(int userId) {  
this.userId = userId;  
}  
public String getUserName() {  
return userName;  
}  
public void setUserName(String userName) {  
this.userName = userName;  
}  
}

**3. Create the hibernate configuration file.**  
 **hibernate.cfg.xml**:

<?xml version=”1.0″ encoding=”UTF-8″?>

<!DOCTYPE hibernate-configuration PUBLIC  
“-//Hibernate/Hibernate Configuration DTD 3.0//EN”  
“http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd”>

<hibernate-configuration>  
<session-factory>  
<!– Database connection settings –>  
<property name=”connection.driver\_class”>com.mysql.jdbc.Driver</property>  
<property name=”connection.url”>jdbc:mysql://localhost:3306/hibernateDB</property>  
<property name=”connection.username”>root</property>  
<property name=”connection.password”>root</property>

<!– JDBC connection pool (use the built-in) –>  
<property name=”connection.pool\_size”>100</property>

<!– SQL dialect –>  
<property name=”hibernate.dialect”>org.hibernate.dialect.MySQL5Dialect</property>

<!– Enable Hibernate’s automatic session context management –>  
<property name=”current\_session\_context\_class”>thread</property>

<!– Disable the second-level cache –>  
<property name=”cache.provider\_class”>org.hibernate.cache.NoCacheProvider</property>

<!– Echo all executed SQL to stdout –>  
<property name=”show\_sql”>true</property>

<!– Drop and re-create the database schema on startup –>  
<property name=”hbm2ddl.auto”>create</property>

<mapping class=”com.sdnext.hibernate.tutorial.dto.UserDetails”/>  
<mapping class=”com.sdnext.hibernate.tutorial.dto.Vehicle”/>

</session-factory>  
</hibernate-configuration>

**4. Create Test Demo class for run this code.**  
 **HibernateTestDemo.java**

package com.sdnext.hibernate.tutorial;

import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.AnnotationConfiguration;

import com.sdnext.hibernate.tutorial.dto.UserDetails;  
import com.sdnext.hibernate.tutorial.dto.Vehicle;

public class HibernateTestDemo {  
  
public static void main(String[] args)  
{  
UserDetails user = new UserDetails(); //create the user entity  
Vehicle vehicle = new Vehicle(); //create the vehicle entity

vehicle.setVehicleName(“SWIFT Car”); //set vehicle name

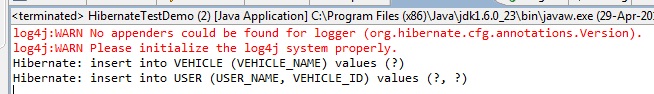
user.setUserName(“Jayanth V”); //set the user name  
user.setVehicle(vehicle); //set the vehicle entity to the field of the user entity i.e. **vehicle entity inside the user entity**

SessionFactory sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory(); //create session factory object  
Session session = sessionFactory.openSession(); //create the session object  
session.beginTransaction();//create the transaction from the session object

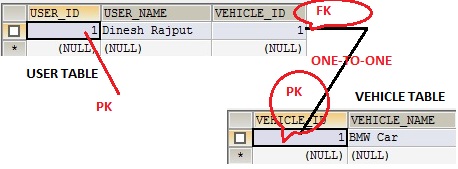
session.save(vehicle); // save the vehicle entity to the database  
session.save(user); // save the user entity to the database

session.getTransaction().commit(); //close the transaction  
session.close(); //close the session  
}  
}

**OUTPUT:**  
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.annotations.Version).  
log4j:WARN Please initialize the log4j system properly.  
Hibernate: insert into VEHICLE (VEHICLE\_NAME) values (?)  
Hibernate: insert into USER (USER\_NAME, VEHICLE\_ID) values (?, ?)



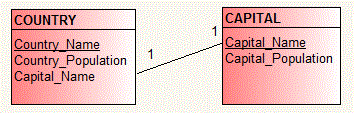
Now we look the created tables for that.



***Hands-on***

In this example, we will see how to implement one to one relationship using annotations.

Lets take example of Country and Capital.One Country has one capital.Following is relationship diagram among them.



Now to create above tables in database, you need to create two java files i.e. Country.java and Capital.java.

### 1.Country.java

Country class will be used to create COUNTRY table in database.  
Create C**ountry.java**in src->org.arpit.javapostsforlearning.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | package org.arpit.javapostsforlearning;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.Id;  import javax.persistence.JoinColumn;  import javax.persistence.OneToOne;  import javax.persistence.Table;    @Entity  @Table(name="COUNTRY")  public class Country {        @Id      @Column(name="Country\_Name")      String countryName ;        @OneToOne      @JoinColumn(name="Capital\_Name")      Capital capital;        @Column(name="Country\_Population")      long countryPopulation;        public Country()      {        }        public Country(String countryName, long countryPopulation) {    this.countryName = countryName;    this.countryPopulation = countryPopulation;  }        public long getCountryPopulation() {    return countryPopulation;  }        public void setCountryPopulation(long countryPopulation) {    this.countryPopulation = countryPopulation;  }        public String getCountryName() {          return countryName;      }        public void setCountryName(String countryName) {          this.countryName = countryName;      }        public Capital getCapital() {    return capital;  }        public void setCapital(Capital capital) {    this.capital = capital;  }  } |

Zorin OS 16 Review: It's a Visual Masterpiece

@OneToOne annotation is used to create one to one relationship between Country and Capital entities.  
@joinColumn  
is used to specify a mapped column for joining an entity association.

### 2.Capital.java

Capital class will be used to create CAPITAL table in database.  
Create **Capital.java**in src->org.arpit.javapostsforlearning.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | package org.arpit.javapostsforlearning;import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.Id;  import javax.persistence.Table;    @Entity  @Table(name="CAPITAL")  public class Capital {    @Id  @Column(name="Capital\_Name")  String capitalName;    @Column(name="Capital\_Population")  long capitalPopulation;    public Capital()  {    }  public Capital(String capitalName, long capitalPopulation) {    super();    this.capitalName = capitalName;    this.capitalPopulation = capitalPopulation;  }    public String getCapitalName() {    return capitalName;  }    public void setCapitalName(String capitalName) {    this.capitalName = capitalName;  }  public long getCapitalPopulation() {    return capitalPopulation;  }    public void setCapitalPopulation(long capitalPopulation) {    this.capitalPopulation = capitalPopulation;  }    } |

### 3.Hiberante.cfg.xml:

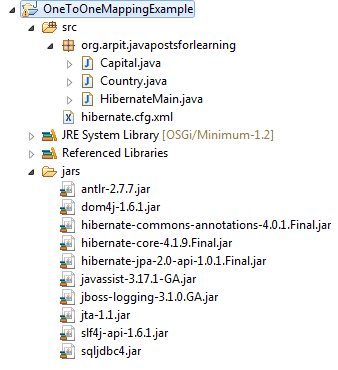
Create a file named “hibernate.cfg.xml” in **src** folder.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | <?xml version='1.0' encoding='utf-8'?>  <!DOCTYPE hibernate-configuration PUBLIC          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">    <hibernate-configuration>        <session-factory>            <!-- Database connection settings -->          <property name="connection.driver\_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>          <property name="connection.url">jdbc:sqlserver://localhost:1433;database=UserInfo</property>          <property name="connection.username">sa</property>          <property name="connection.password"></property>            <!-- JDBC connection pool (use the built-in) -->          <property name="connection.pool\_size">1</property>            <!-- SQL dialect -->          <property name="dialect">org.hibernate.dialect.SQLServer2005Dialect</property>            <!-- Enable Hibernate's automatic session context management -->          <property name="current\_session\_context\_class">thread</property>            <!-- Disable the second-level cache  -->          <property name="cache.provider\_class">org.hibernate.cache.NoCacheProvider</property>            <!-- Echo all executed SQL to stdout -->          <property name="show\_sql">true</property>            <!-- Drop and re-create the database schema on startup -->          <property name="hbm2ddl.auto">create</property>      <mapping class="org.arpit.javapostsforlearning.Country"></mapping>    <mapping class="org.arpit.javapostsforlearning.Capital"></mapping>        </session-factory>    </hibernate-configuration> |

### 4.Main Class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package org.arpit.javapostsforlearning;    import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  import org.hibernate.service.ServiceRegistry;  import org.hibernate.service.ServiceRegistryBuilder;    public class HibernateMain {    public static void main(String[] args) {    Configuration configuration=new Configuration();  configuration.configure();  ServiceRegistry sr= new ServiceRegistryBuilder().applySettings(configuration.getProperties()).buildServiceRegistry();  SessionFactory sf=configuration.buildSessionFactory(sr);  Session ss=sf.openSession();    Country countryIndia=new Country("India",50000000);  Capital capitalDelhi=new Capital("Delhi",4000000);  countryIndia.setCapital(capitalDelhi);  Country countryFrance=new Country("France",20000000);  Capital capitalParis=new Capital("Paris",1000000);  countryFrance.setCapital(capitalParis);  ss.beginTransaction();  ss.save(countryIndia);  ss.save(capitalDelhi);  ss.save(countryFrance);  ss.save(capitalParis);  ss.getTransaction().commit();  ss.close();    }    } |

### Project Struture:



### 5.SQL output:

COUNTRY table in database

https://www.java2blog.com/wp-content/uploads/2013/02/CountryOneToOneMappingOutput.gif

CAPITAL table in database

https://www.java2blog.com/wp-content/uploads/2013/02/CapitalOneToOneMappingOutput.gif