Let us understand the **CRUD** (**C**REATE, **R**EAD, **U**PDATE, **D**ELETE) operations in Hibernate.

CRUD stands for Create, Read, Update and Delete.

* CREATE is used to create a record in the database table.
* READ is used to read the data from the database table.
* UPDATE is used for updating a database table record.
* DELETE is used for deleting a database table record.

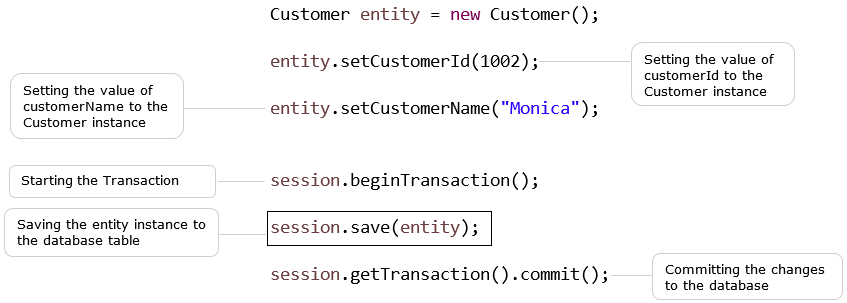
The Session API provides methods for performing the CRUD operation.

* To CREATE/UPDATE the data, the methods available are :
  + save()
  + persist()
  + update()
  + saveOrUpdate()
* To READ the data, the methods available are:
  + get()
  + load()
  + refresh()
* To DELETE the data, the method available is:
  + delete()

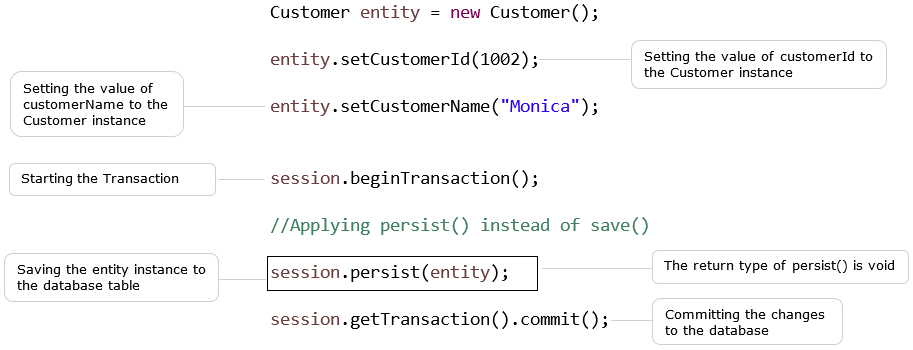
An entity object can be persisted to the corresponding database table by invoking the following Session methods.

* Serializable save (Object object)
  + Persists the entity object to the database table.
  + The primary key value of the persisted record is returned.
* void persist (Object object)
  + Persists the entity object to the database.

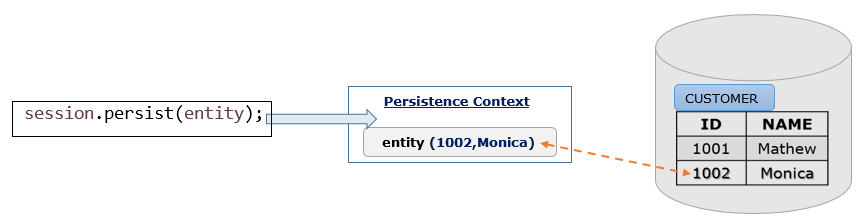
Refer to the Customer entity class created in Hibernate mapping section of this course. The below code snippet demonstrates the usage of save() method to perform the insertion of Customer object into the database. Assuming the session object is already obtained (refer to the earlier section).



The same example of inserting Customer record in the database can also be achieved using **persist()**method as below.



On commit, the Customer object will be stored in the customer table.



Hibernate persist() is similar to save() (with transaction) for insert operation. However, persist() does not return the identifier whereas save() does.

Let us see an example based on Infy Bank **'Managing Customer Records'** use case. This example demonstrates the **Create** operation.

Following are the files used in this example:

Customer.java: A Java class having Customer details which need to persist. It uses annotations for mapping.

ICustomer.java: Interface that defines methods for creating customer records. This will be modified later in this course with the remaining CRUD operations.

CustomerDAO.java: This Java class provides the implementation for the methods of ICustomer interface.

Hibernate.cfg.xml: This is the Hibernate configuration file.

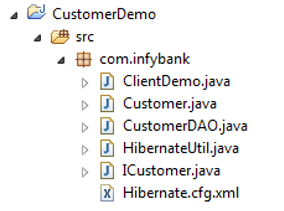
HibernateUtil.java: A utility class for obtaining the instance of SessionFactory API.

ClientDemo.java: This is a Java class used for user interaction. It uses CustomerDAO for inserting the records into the database table.

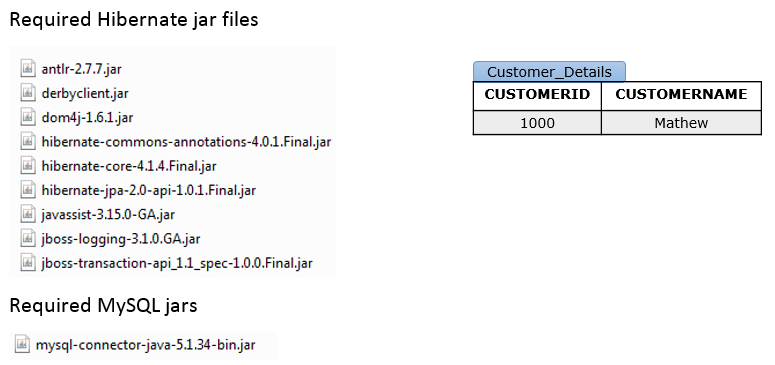
This example uses **MySQL** database and the table name is **Customer\_Details**.

Following are the steps used to develop the Java project in Eclipse Editor.

* Use the File menu and create a new Java project.
* The file structure of the Java project is shown below.



* Following are the jar files, database table used in this Java project.



**Customer.java**is defined as a Hibernate entity with the required annotations as below.



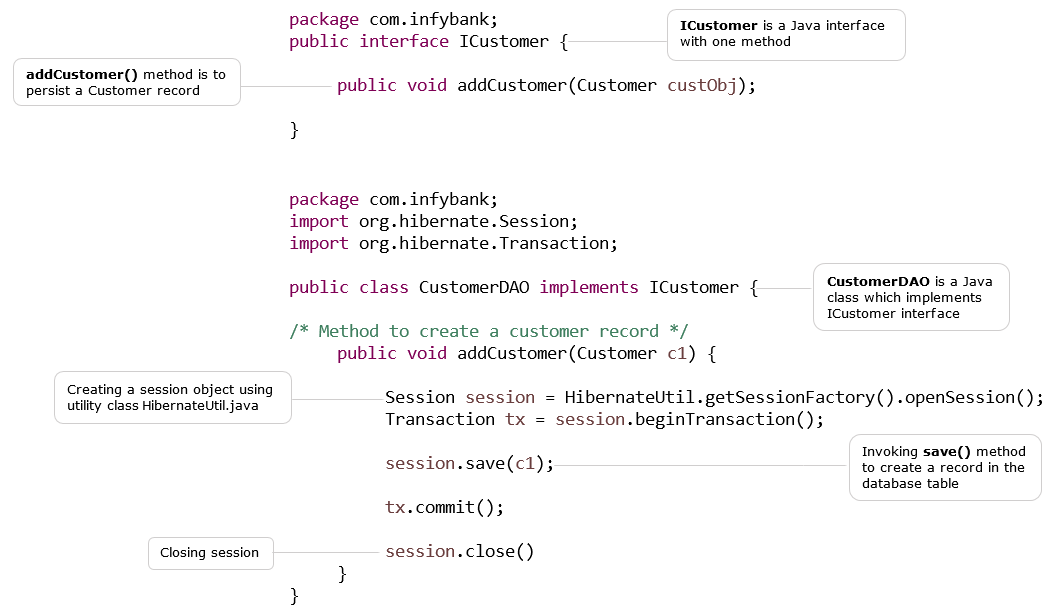
Following is the **Hibernate.cfg.xml** used in this example to define the database connection properties, hibernate dialect and mapping class details.

2. <?xml version="1.0" encoding="UTF-8"?>
3. <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5. <hibernate-configuration>
6. <session-factory>
7. <property name="hibernate.dialect">org.hibernate.dialect.MYSQL5Dialect</property>
8. <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>
9. <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedb</property>
10. <property name="hibernate.connection.username">root</property>
11. <property name="hibernate.connection.password">root</property>
12. <property name="hibernate.show\_sql">true</property>
13. <mapping class="com.infybank.Customer"/>
14. </session-factory>
15. </hibernate-configuration>

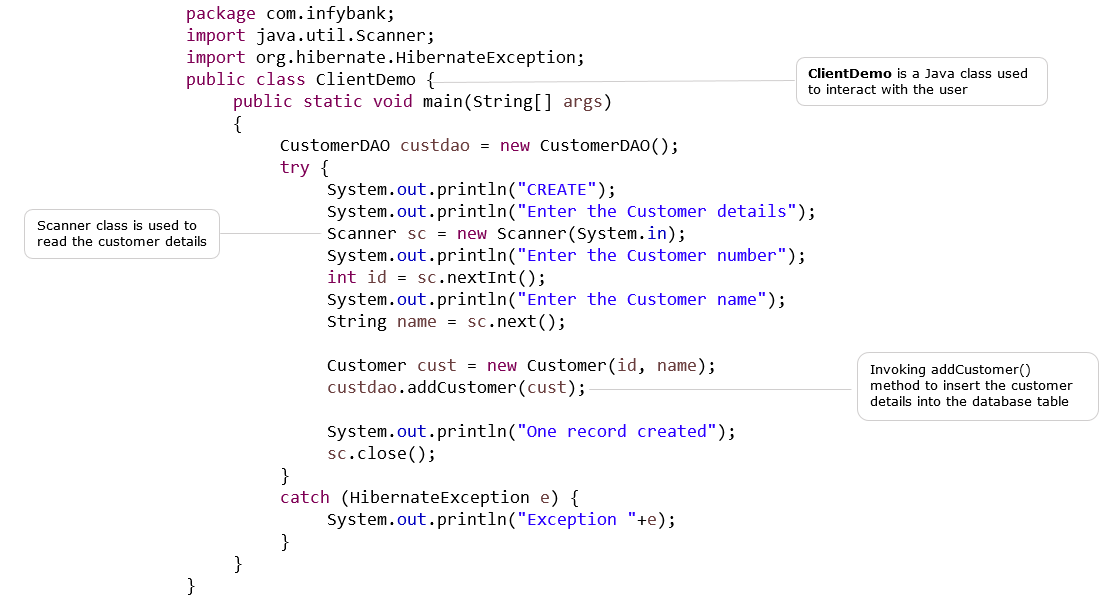
The below class**HibernateUtil.java** is used to create SessionFactory instance. This class is using Configuration, ServiceRegistry, ServiceRegistryBuilder classes for creating the SessionFactory instance.

2. package com.infybank;
3. import org.hibernate.SessionFactory;
4. import org.hibernate.cfg.Configuration;
5. import org.hibernate.service.ServiceRegisteryBuilder;
6. public class HibernateUtil{
7. private static final SessionFactory sessionFactory;
8. static{
9. try{
10. Configuration cfg=new configuration().configure("/com/infybank/Hibernate.cfg.xml");
11. ServiceRegistry serviceRegistery=new ServiceRegisteryBuilder()
12. .applySettings(cfg.getProperties()).buildServiceRegistry();
13. sessionFactory=cfg.buildSessionFactory(serviceRegistery);
14. }
15. catch(Throwable ex){
16. System.err.println("Initial SessionFactory creation failed." +ex);
17. throw new ExceptionInInitializerError(ex);
18. }
19. }
20. public static SessionFactory getSessionFactory(){
21. return sessionFactory;
22. }
23. }

**ICustomer.java**is an interface with a method **addCustomer()**method. This method is used to add a customer record into the database table.**CustomerDAO.java** provides the implementation of the ICustomer interface.



**ClientDemo.java** is a Java class for user interaction. It reads the customer details as the input and saves it into the database table.



Following is the output of ClientDemo.java :

CREATE

Enter the Customer details

Enter the Customer number

1002

Enter the Customer name

Monica

Hibernate: insert into Customer\_Details (CUSTOMERNAME, CUSTOMERID) values (?, ?)

One record created

As a result, the database table Customer\_Details is updated with one more record.

| Customer\_Details | |
| --- | --- |
| **CUSTOMERID** | **CUSTOMERNAME** |
| 1000 | Mathew |
| 1002 | Monica |