HIBERNATE INTERCEPTORS

As you have learnt that in Hibernate, an object will be created and persisted. Once the object has been changed, it must be saved back to the database. This process continues until the next time the object is needed, and it will be loaded from the persistent store.

Thus an object passes through different stages in its life cycle and **Interceptor Interface** provides methods which can be called at different stages to perform some required tasks. These methods are callbacks from the session to the application, allowing the application to inspect and/or manipulate properties of a persistent object before it is saved, updated, deleted or loaded. Following is the list of all the methods available within the Interceptor interface:

S.N.	Method and Description
1	**findDirty()** This method is be called when the **flush()** method is called on a Session object.
2	**instantiate()** This method is called when a persisted class is instantiated.
3	**isUnsaved()** This method is called when an object is passed to the **saveOrUpdate()** method/
4	**onDelete()** This method is called before an object is deleted.
5	**onFlushDirty()** This method is called when Hibernate detects that an object is dirty (ie. have been changed) during a flush i.e. update operation.
6	**onLoad()** This method is called before an object is initialized.
7	**onSave()** This method is called before an object is saved.
8	**postFlush()** This method is called after a flush has occurred and an object has been updated in memory.
9	**preFlush()** This method is called before a flush.

Hibernate Interceptor gives us total control over how an object will look to both the application and the database.

How to use Interceptors?

To build an interceptor you can either implement **Interceptor** class directly or extend **EmptyInterceptor** class. Following will be the simple steps to use Hibernate Interceptor functionality.

Create Interceptors:

We will extend EmptyInterceptor in our example where Interceptor's method will be called automatically when **Employee** object is created and updated. You can implement more methods as per your requirements.

```
import java.io.Serializable;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.EmptyInterceptor;
```

```
import org.hibernate.Transaction;
import org.hibernate.type.Type;

public class MyInterceptor extends EmptyInterceptor {
   private int updates;
   private int creates;
   private int loads;

   public void onDelete(Object entity,
                    Serializable id,
                    Object[] state,
                    String[] propertyNames,
                    Type[] types) {
       // do nothing
   }

   // This method is called when Employee object gets updated.
   public boolean onFlushDirty(Object entity,
                    Serializable id,
                    Object[] currentState,
                    Object[] previousState,
                    String[] propertyNames,
                    Type[] types) {
       if ( entity instanceof Employee ) {
          System.out.println("Update Operation");
          return true;
       }
       return false;
   }
   public boolean onLoad(Object entity,
                   Serializable id,
                   Object[] state,
                   String[] propertyNames,
                   Type[] types) {
       // do nothing
       return true;
   }
   // This method is called when Employee object gets created.
   public boolean onSave(Object entity,
                   Serializable id,
                   Object[] state,
                   String[] propertyNames,
                   Type[] types) {
       if ( entity instanceof Employee ) {
          System.out.println("Create Operation");
          return true;
       }
       return false;
   }
   //called before commit into database
   public void preFlush(Iterator iterator) {
      System.out.println("preFlush");
   }
   //called after committed into database
   public void postFlush(Iterator iterator) {
      System.out.println("postFlush");
   }
}
```

Create POJO Classes:

Now let us modify a little bit our first example where we used EMPLOYEE table and Employee class to play with:

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
```

```
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

Create Database Tables:

Second step would be creating tables in your database. There would be one table corresponding to each object you are willing to provide persistence. Consider above objects need to be stored and retrieved into the following RDBMS table:

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

Create Mapping Configuration File:

This step is to create a mapping file that instructs Hibernate how to map the defined class or classes to the database tables.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator />
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

Create Application Class:

Finally, we will create our application class with the main() method to run the application. Here it should be noted that while creating session object we used our Interceptor class as an argument.

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try{
         factory = new Configuration().configure().buildSessionFactory();
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object." + ex);
         throw new ExceptionInInitializerError(ex);
      }

      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 10000);

      /* List down all the employees */
      ME.listEmployees();

      /* Update employee's records */
      ME.updateEmployee(empID1, 5000);

      /* Delete an employee from the database */
      ME.deleteEmployee(empID2);

      /* List down new list of the employees */
      ME.listEmployees();
   }
   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salary){
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return employeeID;
   }
   /* Method to  READ all the employees */
   public void listEmployees( ){
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         List employees = session.createQuery("FROM Employee").list();
         for (Iterator iterator =
                           employees.iterator(); iterator.hasNext();){
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName());
```

```
            System.out.print("  Last Name: " + employee.getLastName());
            System.out.println("  Salary: " + employee.getSalary());
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to UPDATE salary for an employee */
   public void updateEmployee(Integer EmployeeID, int salary ){
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                    (Employee)session.get(Employee.class, EmployeeID);
         employee.setSalary( salary );
   session.update(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to DELETE an employee from the records */
   public void deleteEmployee(Integer EmployeeID){
      Session session = factory.openSession( new MyInterceptor() );
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                   (Employee)session.get(Employee.class, EmployeeID);
         session.delete(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
}
```

Compilation and Execution:

Here are the steps to compile and run the above mentioned application. Make sure you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

- Create hibernate.cfg.xml configuration file as explained in configuration chapter.
- Create Employee.hbm.xml mapping file as shown above.
- Create Employee.java source file as shown above and compile it.
- Create MyInterceptor.java source file as shown above and compile it.
- Create ManageEmployee.java source file as shown above and compile it.
- Execute ManageEmployee binary to run the program.

You would get following result, and records would be created in EMPLOYEE table.

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........
```

```
Create Operation
preFlush
postFlush
Create Operation
preFlush
postFlush
Create Operation
preFlush
postFlush
First Name: Zara  Last Name: Ali  Salary: 1000
First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
preFlush
postFlush
preFlush
Update Operation
postFlush
preFlush
postFlush
First Name: Zara  Last Name: Ali  Salary: 5000
First Name: John  Last Name: Paul  Salary: 10000
preFlush
postFlush
```

If you check your EMPLOYEE table, it should have following records:

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 29 | Zara       | Ali       |   5000 |
| 31 | John       | Paul      |  10000 |
+----+------------+-----------+--------+
2 rows in set (0.00 sec

mysql>
```