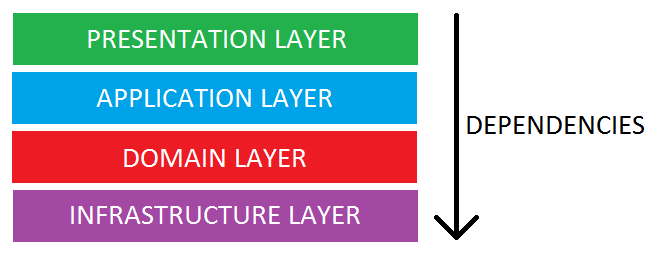
**What Is Layered Architecture?**

A Layered Architecture, as I understand it, is the organization of the project structure into four main categories: **presentation, application, domain, and infrastructure**. Each of the layers contains objects related to the particular concern it represents.

* **The presentation layer** contains all of the classes responsible for presenting the UI to the end-user or sending the response back to the client (in case we’re operating deep in the back-end).
* **The application layer** contains all the logic that is required by the application to meet its functional requirements and, at the same time, is not a part of the domain rules. In most systems that I've worked with, the application layer consisted of services orchestrating the domain objects to fulfill a use case scenario.
* **The domain layer**represents the underlying domain, mostly consisting of domain entities and, in some cases, services. Business rules, like invariants and algorithms, should all stay in this layer.
* **The infrastructure layer (also known as the persistence layer)**contains all the classes responsible for doing the technical stuff, like persisting the data in the database, like DAOs, repositories, or whatever else you’re using.

There are two important rules for a classical Layered Architecture to be correctly implemented:

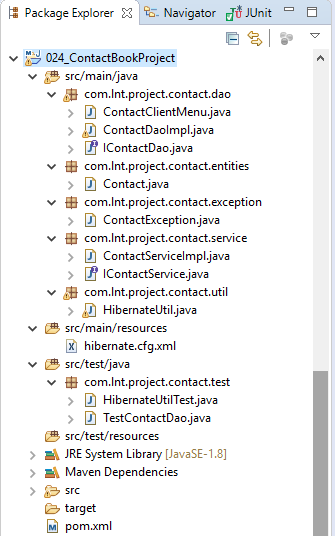
1. All the dependencies go in one direction, from presentation to infrastructure. (Well, handling persistence and domain are a bit tricky because the infrastructure layer often saves domain objects directly, so it actually knows about the classes in the domain)
2. No logic related to one layer’s concern should be placed in another layer. For instance, no domain logic or database queries should be done in the UI.



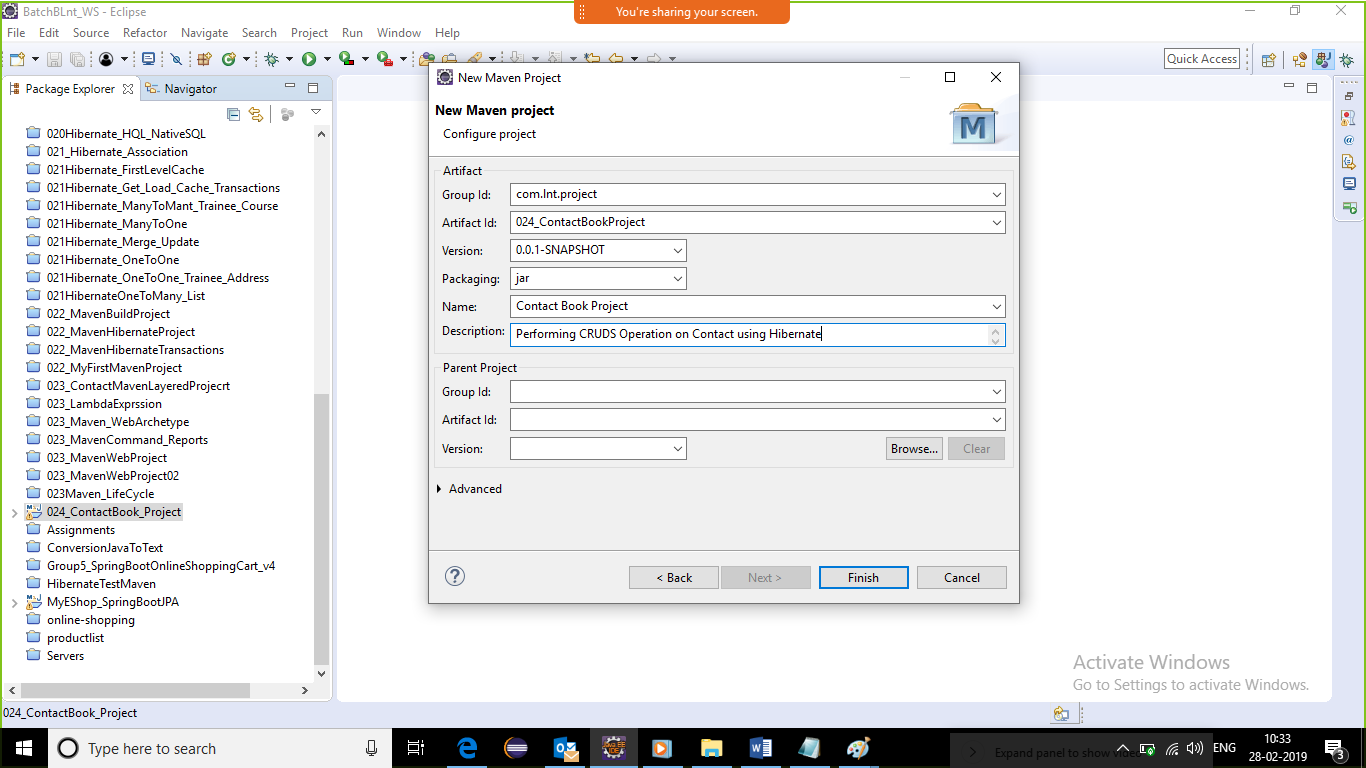
**The Essence of Layered Architecture**

Architecture is kind of an overloaded term, so we should probably dig deeper into what the term really means in the context of layers. The main idea behind Layered Architecture is a separation of concerns – as we said already, we want to avoid mixing domain or database code with the UI stuff, etc. The actual idea of separating a project into layers suggests that this separation of concerns should be achieved by source code organization. This means that apart from some guidance to what concerns we should separate, the Layered Architecture tells us nothing else about the design and implementation of the project. This implies that we should complement it with some other architectural processes, such as some upfront design, daily design sessions, or even full-blown Domain-Driven Design. Whichever option we choose doesn’t matter, at least for the sake of layering, but we need to remember: **Layered Architecture gives us nothing apart from a guideline on how to organize the source code**.

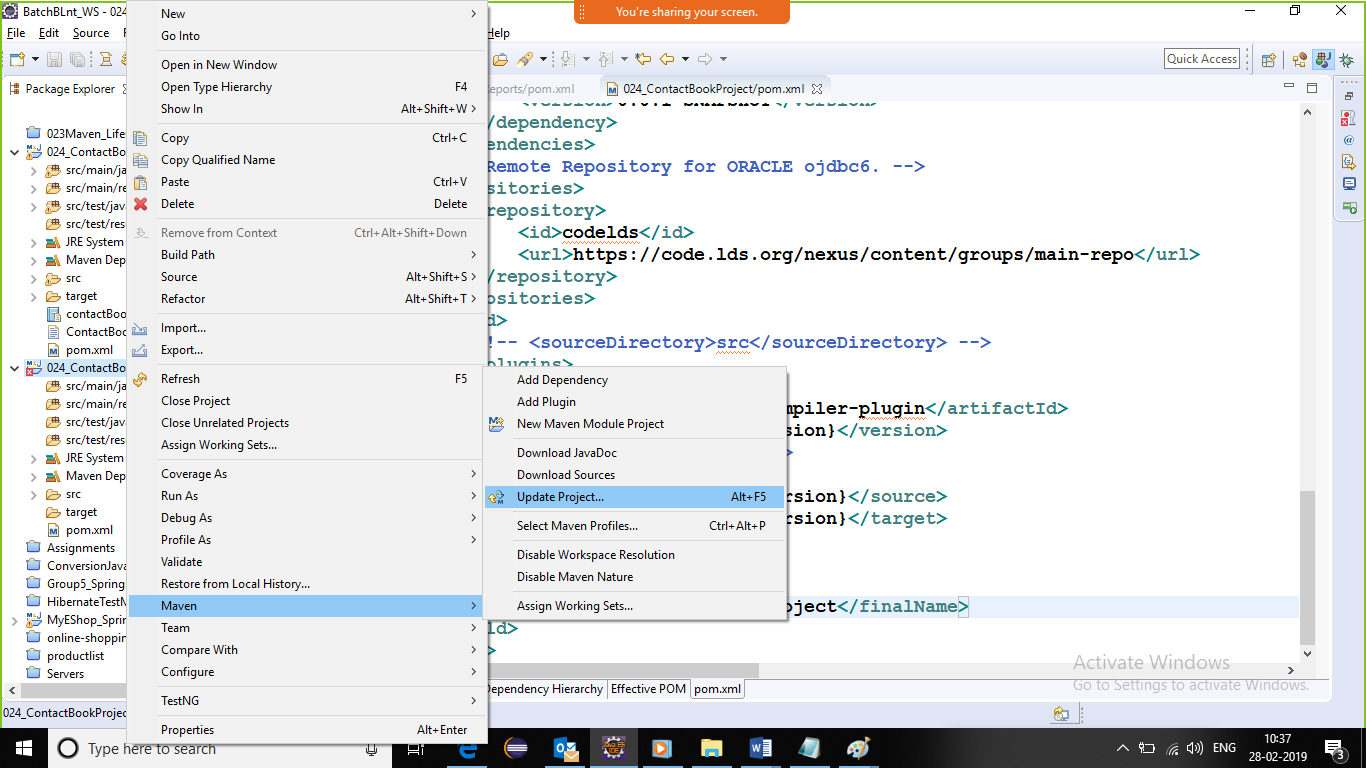
**Layering Java Contact Book Project**

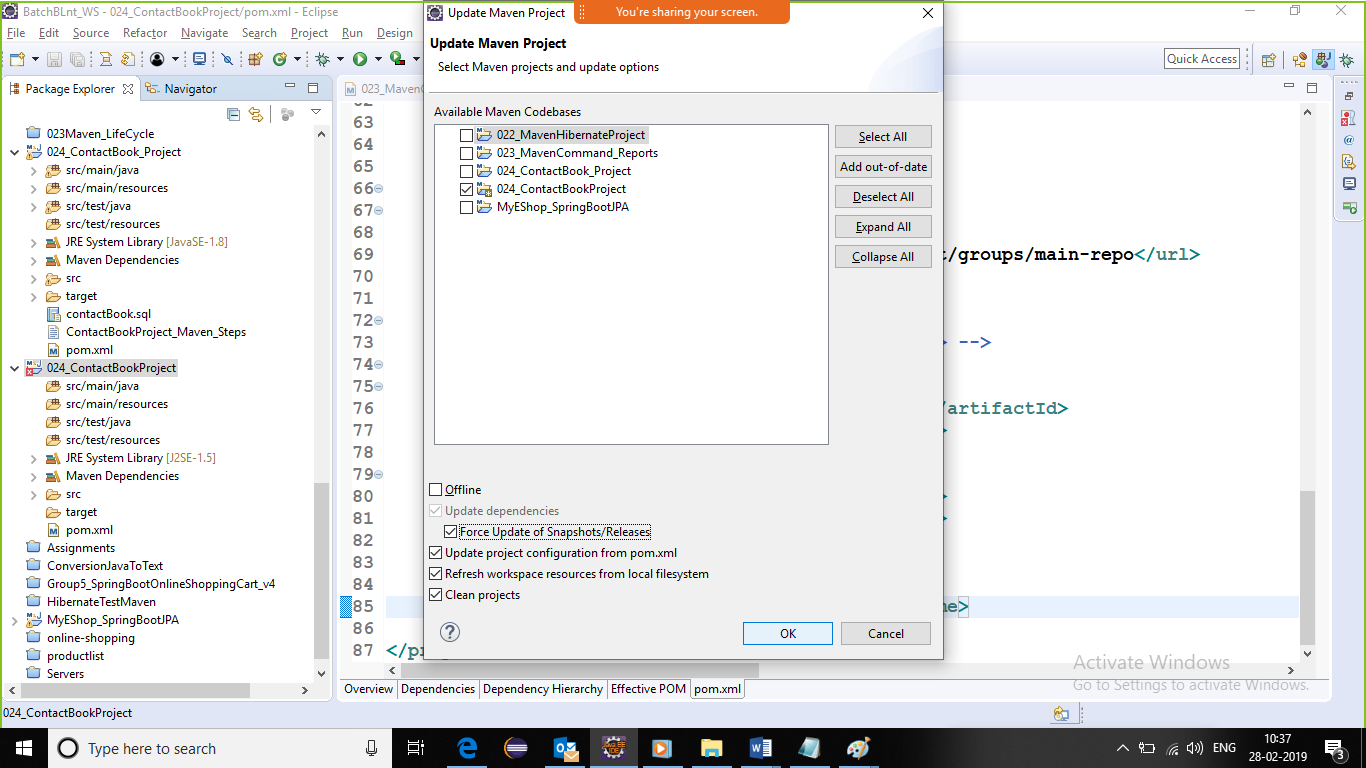
Contact Book Project -> 

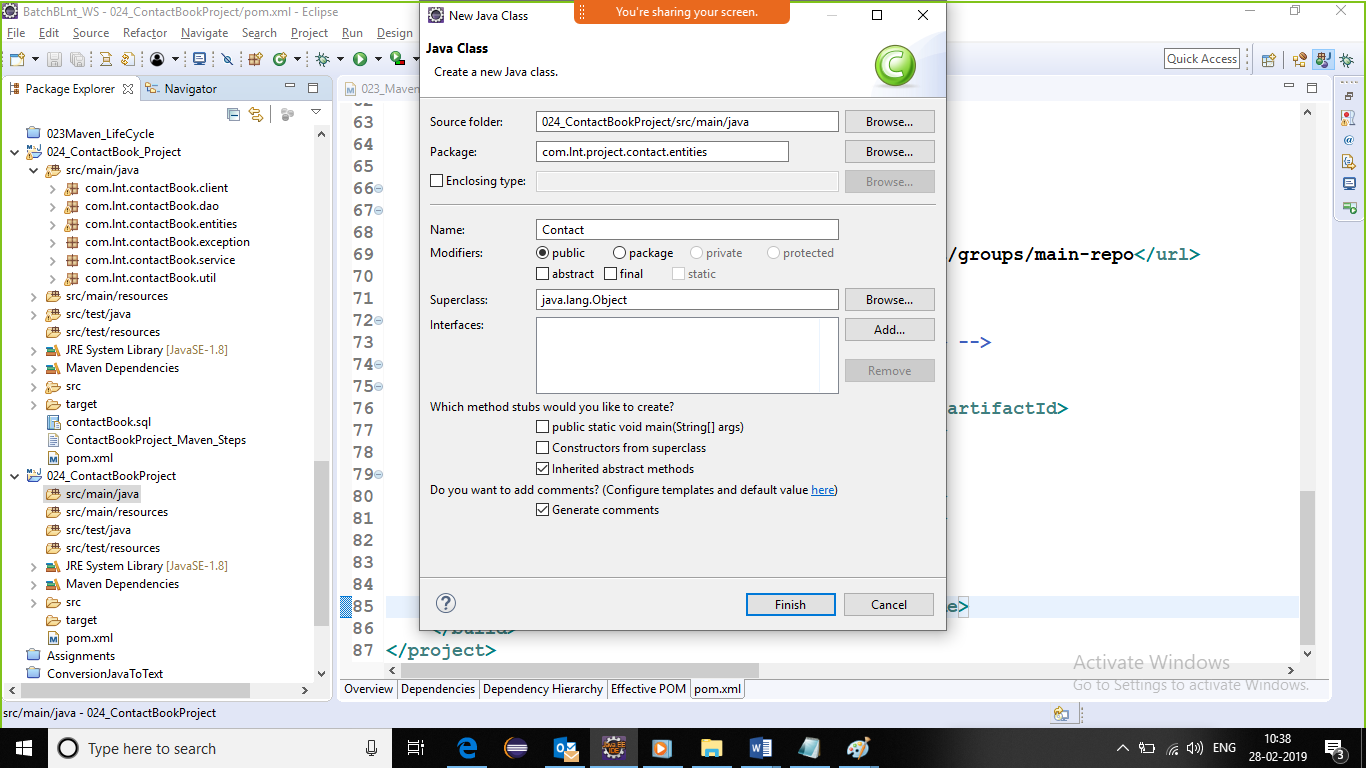
1. Create a New Maven Project



1. Add required dependencies, repositories, java plugins to the pom.xml
2. Maven-> force Update





1. Create Database table and Java Entity 
2. any entity must be created using following best practice

public class ,resides inside package, implements Serializable, generate Serial version id, private instance var ,No-Arg Constructor, Overloaded Constructor, getters and setters, toString() method

@Entity

@Table(name="MY\_CONTACT")

**public** **class** Contact **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 5763628598414506683L;

//primary column

@Id

@GeneratedValue(strategy=GenerationType.***AUTO***)

@Column(name="CONTACT\_ID")

**private** Integer contactId;

@Column(name="FIRST\_NAME")

**private** String firstName;

@Column(name="LAST\_NAME")

**private** String lastName;

@Column(name="PHONE\_NO")

**private** String phoneNo;

@Column(name="EMAIL")

**private** String email;

@Column(name="ADDRESS")

**private** String address;

@Column(name="DOB")

@Temporal(TemporalType.***DATE***)//in case of java.util.Date

**private** Date dob;

//No-Arg Constructor,Overloaded Constructor,getters and setters,toString() method

1. **Now lets configure hibernate connection and map entity. Create hibernate.cfg.xml file in src/main/resources folder.**

**<?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"*>oracle.jdbc.driver.OracleDriver</property>**

**<property name=*"connection.url"*>jdbc:oracle:thin:@localhost:1521:orahome</property>**

**<property name=*"connection.username"*>system</property>**

**<property name=*"connection.password"*>system</property>**

**<!-- JDBC connection pool (use the built-in) -->**

**<property name=*"connection.pool\_size"*>5</property>**

**<!--SQL dialect , db specific commands are prepared by hb , developer need not bother syntax-->**

**<property name=*"dialect"*>org.hibernate.dialect.Oracle10gDialect</property>**

**<!-- dialect will generate specific Sql for the specific db -->**

**<!-- 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>**

**<property name=*"hbm2ddl.auto"*>update</property>**

**<!-- create - every time existing table will be dropped and will create a new -->**

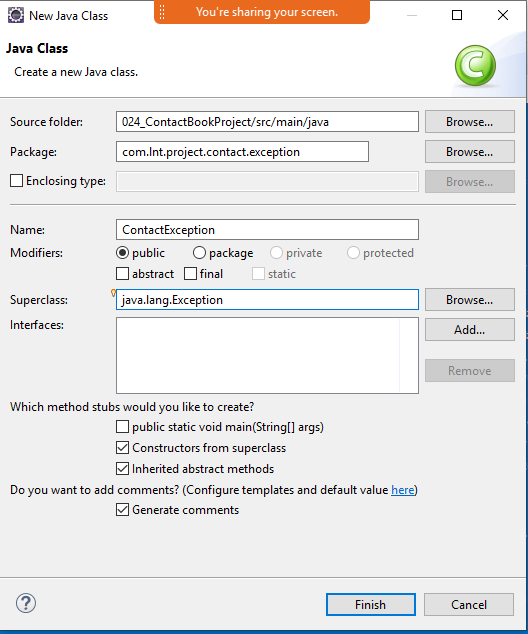
**<!-- update - existing table will be updated and if not exists then will create a new table-->**

**<!--Automatically create update validate -->**

**<mapping class=*"com.lnt.project.contact.entities.Contact"*/> <!-- annotation mapping -->**

**</session-factory>**

**</hibernate-configuration>**

1. **Create User defined Exception ContactException in **

**package com.lnt.project.contact.exception;**

**/\*\* \* @author Smita B Kumar \* \*/**

**public class ContactException extends Exception {**

**private static final long *serialVersionUID* = -7500232798281989303L;**

**public ContactException() {**

**super();**

**}**

**public ContactException(String message) {**

**super(message);**

**}**

**}**

1. Create the HibernateUtil for getting and closing sessionFactory

package com.lnt.project.contact.util;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

/\*\* \* @author Smita B Kumar \* \*/

public class HibernateUtil {

private static SessionFactory *sessionFactory*;

public static SessionFactory getSessionFactory() {

if(*sessionFactory*!=null) {

*sessionFactory*=new Configuration().configure().~~buildSessionFactory~~();

}

return *sessionFactory*;

}

public static void setSessionFactory(SessionFactory sessionFactory) {

HibernateUtil.*sessionFactory* = sessionFactory;

}

public static SessionFactory closeFactory(SessionFactory sessionFactory) {

if(sessionFactory!=null) {

sessionFactory.close();

sessionFactory=null;

}

return sessionFactory;

}

}

Create a test case for main method to test the HibernateUtil methods.

**package com.lnt.project.contact.test;**

**import static org.junit.Assert.*assertNotNull*;**

**import static org.junit.Assert.*assertNull*;**

**import org.hibernate.SessionFactory;**

**import org.junit.Test;**

**import com.lnt.project.contact.util.HibernateUtil;**

**/\*\* \* @author Smita B Kumar \* \*/**

**public class HibernateUtilTest {**

**private static SessionFactory *sessionFactory*;**

**/\*\***

**\* Test method for {@link com.lnt.project.contact.util.HibernateUtil#getSessionFactory()}.**

**\*/**

**@Test**

**public void testGetSessionFactory() {**

***sessionFactory* = HibernateUtil.*getSessionFactory*();**

***assertNotNull*("sessionFactory is null", *sessionFactory*);**

**}**

**/\*\***

**\* Test method for {@link com.lnt.project.contact.util.HibernateUtil#closeFactory(org.hibernate.SessionFactory)}.**

**\*/**

**@Test**

**public void testCloseFactory() {**

***sessionFactory* = HibernateUtil.*closeFactory*(*sessionFactory*);**

***assertNull*("sessionFactory not closed", *sessionFactory*);**

**}**

**}**

**//or with main method**

**public** **static** **void** main(String[] args) {

SessionFactory factory=*getSessionFactory*();

**try** {

**if**(factory!=**null**)

System.***out***.println("Factory configured!!");

**else**

System.***err***.println("Factory NOT configured!!");

}**catch** (Exception e) {

e.printStackTrace();

}

factory=*closeFactory*(factory);

**if**(factory==**null**)

System.***out***.println("Factory closed!!");

**else**

System.***err***.println("Factory NOT closed!!");

}

1. Create Dao Layer
   1. IContactDao interface
   2. ContactDaoImpl class

**package** com.lnt.project.contact.dao;

**import** java.util.List;

**import** com.lnt.project.contact.entities.Contact;

**import** com.lnt.project.contact.exception.ContactException;

/\*\* \* **@author** Smita B Kumar \* \*/

**public** **interface** IContactDao {

**public** Integer addNewContact(Contact contact)**throws** ContactException; //-Create //add contact

**public** List<Contact> getAllContacts()**throws** ContactException; //-Retreive //Get All contact

**public** Contact updateContact(Contact updatedContact)**throws** ContactException; //-Update //edit contact

**public** Contact removeContact(Contact contact)**throws** ContactException; //-Delete //remove contact

**public** Contact searchContactByName(String name)**throws** ContactException; //-Search

**public** Contact searchContactById(Integer contactId)**throws** ContactException;//find contact

**public** Object terminateApplication()**throws** ContactException;

}

* 1. ContactDaoImpl class

**package** com.lnt.project.contact.dao;

**import** java.util.List;

**import** org.hibernate.Query;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** com.lnt.project.contact.entities.Contact;

**import** com.lnt.project.contact.exception.ContactException;

**import** com.lnt.project.contact.util.HibernateUtil;

/\*\* \* **@author** Smita B Kumar \* \*/

**public** **class** ContactDaoImpl **implements** IContactDao {

//Prep Work-Session Factory

**private** **static** SessionFactory *sessionFactory*;

**private** Session session;

**private** Transaction tx;

**private** Query query;

**static** {

*sessionFactory*=HibernateUtil.*getSessionFactory*();

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.dao.IContactDao#addNewContact(com.lnt.project.contact.entities.Contact)

\*/

@Override

**public** Integer addNewContact(Contact contact) **throws** ContactException {

session= *sessionFactory*.openSession();

tx=session.beginTransaction();

Integer contactId=(Integer) session.save(contact);

tx.commit();

session.close();

**return** contactId;

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.dao.IContactDao#getAllContacts()

\*/

@Override

**public** List<Contact> getAllContacts() **throws** ContactException {

session= *sessionFactory*.openSession();

query = session.createQuery("from Contact");

List<Contact> contacts = query.list();

session.close();

**return** contacts;

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.dao.IContactDao#updateContact(com.lnt.project.contact.entities.Contact)

\*/

@Override

**public** Contact updateContact(Contact updatedContact) **throws** ContactException {

session= *sessionFactory*.openSession();

tx=session.beginTransaction();

session.update(updatedContact);

tx.commit();

session.close();

**return** updatedContact;

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.dao.IContactDao#removeContact(com.lnt.project.contact.entities.Contact)

\*/

@Override

**public** Contact removeContact(Contact contact) **throws** ContactException {

session= *sessionFactory*.openSession();

tx=session.beginTransaction();

session.delete(contact);

tx.commit();

session.close();

contact=**null**;

**return** contact;

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.dao.IContactDao#searchContactById()

\*/

@Override

**public** Contact searchContactById(Integer contactId) **throws** ContactException {

session=*sessionFactory*.openSession();

query=session.createQuery("from Contact where contactId=:contactId");

//Contact is the name of the class

query.setParameter("contactId", contactId);

query.setMaxResults(1);

Contact contact=(Contact) query.uniqueResult();

session.close();

**return** contact;

} /\* (non-Javadoc)

\* @see com.lnt.project.contact.dao.IContactDao#searchContactByName(java.lang.String)

\*/

@Override

**public** Contact searchContactByName(String name) **throws** ContactException {

session= *sessionFactory*.openSession();

query = session.createQuery("from Contact c where c.firstName=:firstName");

query.setParameter("firstName", name);

query.setMaxResults(1);

Contact contact = (Contact) query.uniqueResult();

session.close();

**return** contact;

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.dao.IContactDao#terminateApplication()

\*/

@Override

**public** Object terminateApplication() **throws** ContactException {

*sessionFactory* = HibernateUtil.*closeFactory*(*sessionFactory*);

**return** *sessionFactory*;

}

}

1. **Test Each Dao layer method with either testcases for main method.**

**package** com.lnt.project.contact.test;

**import** **static** org.junit.Assert.*assertNotNull*;

**import** **static** org.junit.Assert.*assertNull*;

**import** **static** org.junit.Assert.*assertTrue*;

**import** java.time.LocalDate;

**import** java.time.ZoneId;

**import** java.util.Date;

**import** java.util.List;

**import** org.junit.AfterClass;

**import** org.junit.BeforeClass;

**import** org.junit.Ignore;

**import** org.junit.Test;

**import** com.lnt.project.contact.dao.ContactDaoImpl;

**import** com.lnt.project.contact.dao.IContactDao;

**import** com.lnt.project.contact.entities.Contact;

**import** com.lnt.project.contact.exception.ContactException;

/\*\* \* **@author** Smita\* \*/

**public** **class** TestContactDao {

**private** **static** IContactDao *contactDao*;

LocalDate localDate=LocalDate.*of*(2011, 11, 11);

Date dob = Date.*from*(localDate.atStartOfDay().atZone(ZoneId.*systemDefault*()).toInstant());

Contact contact= **new** Contact("Smita", "Kumar", "7738206222",

"smitakumar@synergetics-india.com", "Mumbai", dob);

Integer contactId;

/\*\*

\* **@throws** java.lang.Exception

\*/

@BeforeClass

**public** **static** **void** setUpBeforeClass() **throws** Exception {

//inorder to call the method of ContactDaoImpl class , we should first create the object of ContactDaoImpl

*contactDao*= **new** ContactDaoImpl();

}

/\*\*

\* Test method for {@link com.lnt.contactBook.dao.ContactDaoImpl#addNewContact(com.lnt.contactBook.entities.Contact)}.

\* **@throws** ContactException

\*/

@Test

//@Ignore

**public** **void** testAddNewContact() **throws** ContactException {

contactId=*contactDao*.addNewContact(contact);

contact.setContactId(contactId);

*assertTrue*(contactId>0);

System.***out***.println("Contact Persisted with Id : "+contactId);

}

/\*\*

\* Test method for {@link com.lnt.contactBook.dao.ContactDaoImpl#updateContact(com.lnt.contactBook.entities.Contact)}.

\* **@throws** ContactException

\*/

@Test

@Ignore

**public** **void** testUpdateContact() **throws** ContactException {

Contact updatedContact= **new** Contact("Yara", "Khan", "9999977777",

"yarakhan@synergetics-india.com", "Hydrabad", dob);

updatedContact.setContactId(6);//existing contact

contact=*contactDao*.updateContact(updatedContact);

System.***out***.println("Uodated Contact : "+contact);

}

/\*\*

\* Test method for {@link com.lnt.contactBook.dao.ContactDaoImpl#getAllContacts()}.

\* **@throws** ContactException

\*/

@Test

**public** **void** testGetAllContacts() **throws** ContactException {

List<Contact> contactList=*contactDao*.getAllContacts();

*assertNotNull*("No Contact Records Found!!", contactList);

**for** (Contact contact : contactList) {

System.***out***.println(contact);

}

}

/\*\*

\* Test method for {@link com.lnt.contactBook.dao.ContactDaoImpl#removeContact(com.lnt.contactBook.entities.Contact)}.

\* **@throws** ContactException

\*/

@Test

//@Ignore

**public** **void** testRemoveContact() **throws** ContactException {

contact=*contactDao*.removeContact(contact);

*assertNull*("No Contact Records Found!!", contact);

System.***out***.println("Deleted Contact : "+contact);

}

/\*\*

\* Test method for {@link com.lnt.contactBook.dao.ContactDaoImpl#searchContactById(java.lang.Integer)}.

\* **@throws** ContactException

\*/

@Test

@Ignore

**public** **void** testSearchContactById() **throws** ContactException {

Integer id =15;

Contact expected = *contactDao*.searchContactById(id);

*assertNotNull*("No Contact Records Found!", expected);

System.***out***.println("Search Contact by Id "+id+": "+expected);

}

/\*\*

\* Test method for {@link com.lnt.contactBook.dao.ContactDaoImpl#searchContactByName(java.lang.String)}.

\* **@throws** ContactException

\*/

@Test

@Ignore

**public** **void** testSearchContactByName() **throws** ContactException {

String name ="Yara";

Contact expected = *contactDao*.searchContactByName(name);

*assertNotNull*("No Contact Records Found!", expected);

System.***out***.println("Search Contact by Name "+name+": "+expected);

}

/\*\*

\* Test method for {@link com.lnt.contactBook.dao.ContactDaoImpl#terminateApplication()}.

\* **@throws** ContactException

\*/

@AfterClass

**public** **static** **void** testTerminateApplication() **throws** ContactException {

Object obj=*contactDao*.terminateApplication();

*assertNull*("Not able to terminate the Session Factory!!", obj);

System.***out***.println("Session Factory Terminated : "+obj);

}

}

1. **Once all the methods of DaoImpl is running fine after testing. Lets Create the Service Layer**
   1. **IContactService Interface**
   2. **ContactServiceImpl class**

**package** com.lnt.project.contact.service;

**import** java.util.List;

**import** com.lnt.project.contact.entities.Contact;

**import** com.lnt.project.contact.exception.ContactException;

/\*\* \* **@author** Smita B Kumar \* \*/

**public** **interface** IContactService {

**public** Integer addNewContact(Contact contact)**throws** ContactException; //-Create //add contact

**public** List<Contact> getAllContacts()**throws** ContactException; //-Retrieve //Get All contact

**public** Contact updateContact(Contact updatedContact)**throws** ContactException; //-Update //edit contact

**public** Contact removeContact(Contact contact)**throws** ContactException; //-Delete //remove contact

**public** Contact searchContactByName(String name)**throws** ContactException; //-Search //find contact

**public** Object terminateApplication()**throws** ContactException;

**public** Contact searchContactById(Integer contactId)**throws** ContactException;

//if any vaidation or calculation to be performed

//then additional method to be provided at Service layer for the same

}

**package** com.lnt.project.contact.service;

**import** java.util.List;

**import** com.lnt.project.contact.dao.ContactDaoImpl;

**import** com.lnt.project.contact.dao.IContactDao;

**import** com.lnt.project.contact.entities.Contact;

**import** com.lnt.project.contact.exception.ContactException;

/\*\* \* **@author** Smita B Kumar \* \*/

**public** **class** ContactServiceImpl **implements** IContactService {

//prep work - Dao layer object

//then in each method call the dao layer method and return to client

**private** **static** IContactDao *contactDao*;

**static** {

*contactDao*=**new** ContactDaoImpl();

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.service.IContactService#addNewContact(com.lnt.project.contact.entities.Contact)

\*/

@Override

**public** Integer addNewContact(Contact contact) **throws** ContactException {

**return** *contactDao*.addNewContact(contact);

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.service.IContactService#getAllContacts()

\*/

@Override

**public** List<Contact> getAllContacts() **throws** ContactException {

**return** *contactDao*.getAllContacts();

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.service.IContactService#updateContact(com.lnt.project.contact.entities.Contact)

\*/

@Override

**public** Contact updateContact(Contact updatedContact) **throws** ContactException {

**return** *contactDao*.updateContact(updatedContact);

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.service.IContactService#removeContact(com.lnt.project.contact.entities.Contact)

\*/

@Override

**public** Contact removeContact(Contact contact) **throws** ContactException {

**return** *contactDao*.removeContact(contact);

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.service.IContactService#searchContactById(java.lang.Integer)

\*/

@Override

**public** Contact searchContactById(Integer contactId) **throws** ContactException {

**return** *contactDao*.searchContactById(contactId);

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.service.IContactService#searchContactByName(java.lang.String)

\*/

@Override

**public** Contact searchContactByName(String name) **throws** ContactException {

**return** *contactDao*.searchContactByName(name);

}

/\* (non-Javadoc)

\* @see com.lnt.project.contact.service.IContactService#terminateApplication()

\*/

@Override

**public** Object terminateApplication() **throws** ContactException {

**return** *contactDao*.terminateApplication();

}

}

1. **Lets create the client code with a main method which will call all the services of Service layer**

**package** com.lnt.project.contact.dao;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.time.LocalDate;

**import** java.time.ZoneId;

**import** java.util.Date;

**import** java.util.InputMismatchException;

**import** java.util.List;

**import** com.lnt.project.contact.entities.Contact;

**import** com.lnt.project.contact.exception.ContactException;

**import** com.lnt.project.contact.service.ContactServiceImpl;

**import** com.lnt.project.contact.service.IContactService;

/\*\* \* **@author** Smita B Kumar \* \*/

**public** **class** ContactClientApp {

// prep work Step 1: create object of service inorder to invoke service methods

// prep work Step 2:create object of Scanner and BufferedReader to accept input

// from user

**private** **static** IContactService *contactService*;

**static** **boolean** *status*;

**static** BufferedReader *br*;

**static** {

*contactService* = **new** ContactServiceImpl();

*status*=**true**;

*br* = **new** BufferedReader(**new** InputStreamReader(System.***in***));

}

**public** **static** **void** main(String[] args){

**try** {

*selectChoice*();

}**catch** (Exception e) {

System.***err***.println(**new** ContactException("Sorry Boss !, Something went Wrong while entering Input!!"+e));

}**finally** {

**if**(*br*!=**null**)

**try** {

*br*.close();

} **catch** (IOException e) {

System.***err***.println(**new** ContactException("Sorry Boss !, Something went Wrong while entering Input!!"+e));

}

}

}

**public** **static** **void** selectChoice(){

// display the menu

**int** choice=0;

**do**{

**try** {

*displayMenu*();

choice=Integer.*parseInt*(*br*.readLine());

**switch** (choice) {

**case** 1:

System.***out***.println("Your have selected ...to Add New Contact");

*addContactDetails*(*br*,**new** Contact());

**break**;

**case** 2:

System.***out***.println("Your have selected ...to List All Contacts");

*listAllContact*();

**break**;

**case** 3:

System.***out***.println("Your have selected ...to Update Existing Contact");

*updateContactDetails*(*br*);

**break**;

**case** 4:

System.***out***.println("Your have selected ...to Search Contact By Id");

*getContactById*(*br*);

**break**;

**case** 5:

System.***out***.println("Your have selected ...to Search Contact By Name");

*getContactByName*(*br*);

**break**;

**case** 6:

System.***out***.println("Your have selected ...to Remove Contact");

*removeContact*(*br*);

**break**;

**case** 7:

System.***out***.println("Your have selected ...to Terminate the contact Application"

+ "\n Thanks For Visting Our Aplication....Do Visit AGain !!");

*contactService*.terminateApplication();

System.*exit*(0);

**break**;

**default**:

System.***out***.println("Sorry Boss! you have entered a wrong choice....."

+ "\nKindly Select Your Choice from (1-7)Numeric Only\n");

**break**;

}

System.***out***.println("Hit 7 to Exit the Apllication ");

}**catch** (ContactException e) {

System.***err***.println("Sorry Boss !, Something went Wrong while accessing the Contact Book!!"+e);

//scan.nextLine(); // clears the buffer

*selectChoice*();

}**catch** (InputMismatchException e) {

System.***err***.println("Sorry Boss !, Input Missmatched while entering Input!!"+e);

*selectChoice*();

}**catch** (IOException e) {

System.***err***.println("Sorry Boss !, Something went Wrong while entering Input!!"+e);

*selectChoice*();

}

**catch** (Exception e) {

System.***err***.println("Sorry Boss !, Something went Wrong while entering Input!!"+e);

*selectChoice*();

}

}**while**(*status*);//end of while

}

**private** **static** **void** displayMenu() {

System.***out***.println(

"\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"

+ "\n Contact Book Application"

+ "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"

+ " Select Your Choice from (1-7)Numeric Only\n"

+ " 1. Add New Contact\n"

+ " 2. List AllContacts\n"

+ " 3. Update Contact\n"

+ " 4. Search Contact By Id\n"

+ " 5. Search Contact By Name\n"

+ " 6. Remove Contact\n"

+ " 7. Exit\n"

+ "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");

}

**private** **static** Contact acceptContactDetails(BufferedReader br,Contact contact) **throws** ContactException {

// accepting the contact details from user

**try** {

System.***out***.println("Enter Contact First Name : ");

String firstName = br.readLine();

System.***out***.println("Enter Contact Last Name : ");

String lastName = br.readLine();

System.***out***.println("Enter Contact email : ");

String email = br.readLine();

System.***out***.println("Enter Contact phoneNo : ");

String phoneNo = br.readLine();

System.***out***.println("Enter Contact Address : ");

String address = br.readLine();

LocalDate localDate = LocalDate.*of*(2011, 11, 11);

Date dob = Date.*from*(localDate.atStartOfDay().atZone(ZoneId.*systemDefault*()).toInstant());

contact.setFirstName(firstName);

contact.setLastName(lastName);

contact.setEmail(email);

contact.setPhoneNo(phoneNo);

contact.setAddress(address);

contact.setDob(dob);

} **catch** (Exception e) {

System.***out***.println("Something went worng while accepting Contact Details!!");

}

**return** contact;

}

**private** **static** Contact addContactDetails(BufferedReader br,Contact contact) **throws** ContactException {

// accepting the contact details from user

Contact addedContact= *acceptContactDetails*(br,contact);

**int** contactId = *contactService*.addNewContact(contact);

**if** (contactId > 0) {

System.***out***.println("Contact Added successfully with a unique contactId : " + contactId);

} **else** {

System.***err***.println("Sorry Boss !, Not able to persist the Contact Object in Contact Book");

}

**return** addedContact;

}

**private** **static** **void** listAllContact() {

List<Contact> contactList = **null**;

**try** {

contactList = *contactService*.getAllContacts();

**if** (contactList != **null**) {

System.***out***.println("List Of all the Contact Records Found");

contactList.forEach(System.***out***::println);// java 8

} **else** {

System.***err***.println("Sorry Boss !, No Contact Records found in the Contact Book");

}

} **catch** (ContactException e) {

System.***err***.println("Sorry "

+ "Boss !, No Contact Records found in the Contact Book" + e);

}

}

**private** **static** Contact updateContactDetails(BufferedReader br) **throws** ContactException {

// accepting the contact Id from user and checking weather user exists or not

Contact updatedContact = *getContactById*(br);

**if** (updatedContact != **null**) {

updatedContact= *acceptContactDetails*(br,updatedContact);

updatedContact = *contactService*.updateContact(updatedContact);

**if** (updatedContact != **null**) {

System.***out***.println("Contact Updated successfully : " + updatedContact);

} **else** {

System.***err***.println("Sorry Boss !, Not able to Update the Contact Object in Contact Book");

}

}

**return** updatedContact;

}

**private** **static** Contact getContactById(BufferedReader br) **throws** ContactException{

System.***out***.println("Enter the Contact Id to be Search....");

Integer id=0;

**try** {

id = Integer.*parseInt*(br.readLine());

} **catch** (IOException e) {

System.***out***.println("Something went worng while accepting Contact Name!!");

//selectChoice();

}

Contact searchedcontact=*contactService*.searchContactById(id);

**if**(searchedcontact!=**null**) {

System.***out***.println("Contact Found successfully : "+searchedcontact);

}**else** {

System.***err***.println("Sorry Boss !, Not able to Find the Contact Object in Contact Book");

}

**return** searchedcontact;

}

**private** **static** Contact getContactByName(BufferedReader br) **throws** ContactException{

System.***out***.println("Enter the Contact Name to be Search....");

String name=**null**;

**try** {

name = br.readLine();

} **catch** (IOException e) {

System.***out***.println("Something went worng while accepting Contact Name!!");

//selectChoice();

}

Contact searchedcontact=*contactService*.searchContactByName(name);

**if**(searchedcontact!=**null**) {

System.***out***.println("Contact Found successfully : "+searchedcontact);

}**else** {

System.***err***.println("Sorry Boss !, Not able to Find the Contact Object in Contact Book");

}

**return** searchedcontact;

}

**private** **static** **void** removeContact(BufferedReader br) **throws** ContactException, IOException {

System.***out***.println("Enter the contactId to be Deleted....");

Integer id= Integer.*parseInt*(br.readLine());

Contact contactFound=*contactService*.searchContactById(id);

**if**(contactFound!=**null**) {

Contact deletedContact= *contactService*.removeContact(contactFound);

**if**(deletedContact==**null**) {

System.***out***.println("Contact Removed successfully : "+contactFound);

contactFound=**null**;

}**else** {

System.***err***.println("Sorry Boss !, Not able to remove the Contact Object from the Contact Book");

}

}**else** {

System.***err***.println("Sorry Boss !, Not able to remove As Contact Onject Not Found");

}

}

}//end of class