EJB3 (Entity, Stateless Session) + Visual JSF project

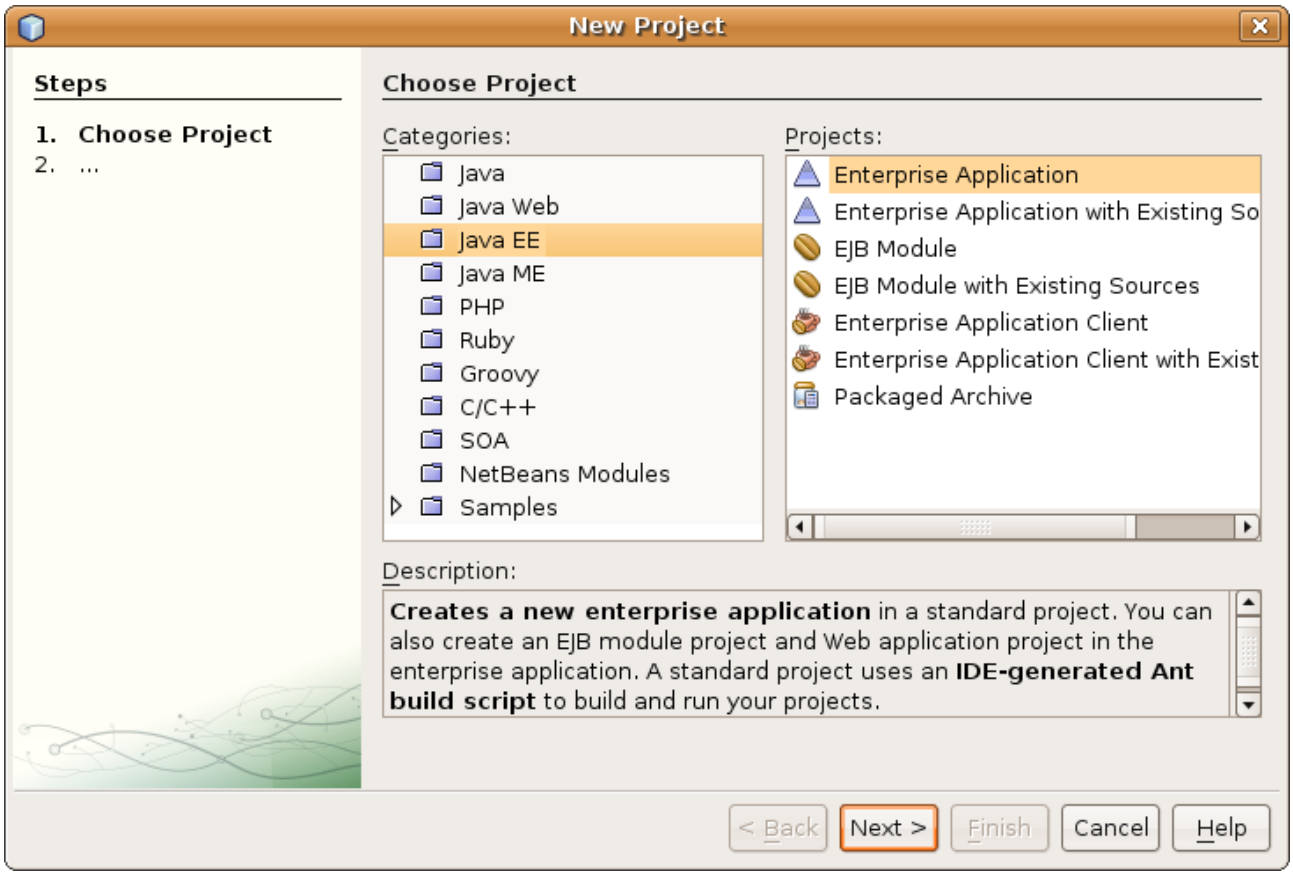
(Erich Lorman, 05/2009, Dezadata s.r.o.)

System pre-requirements

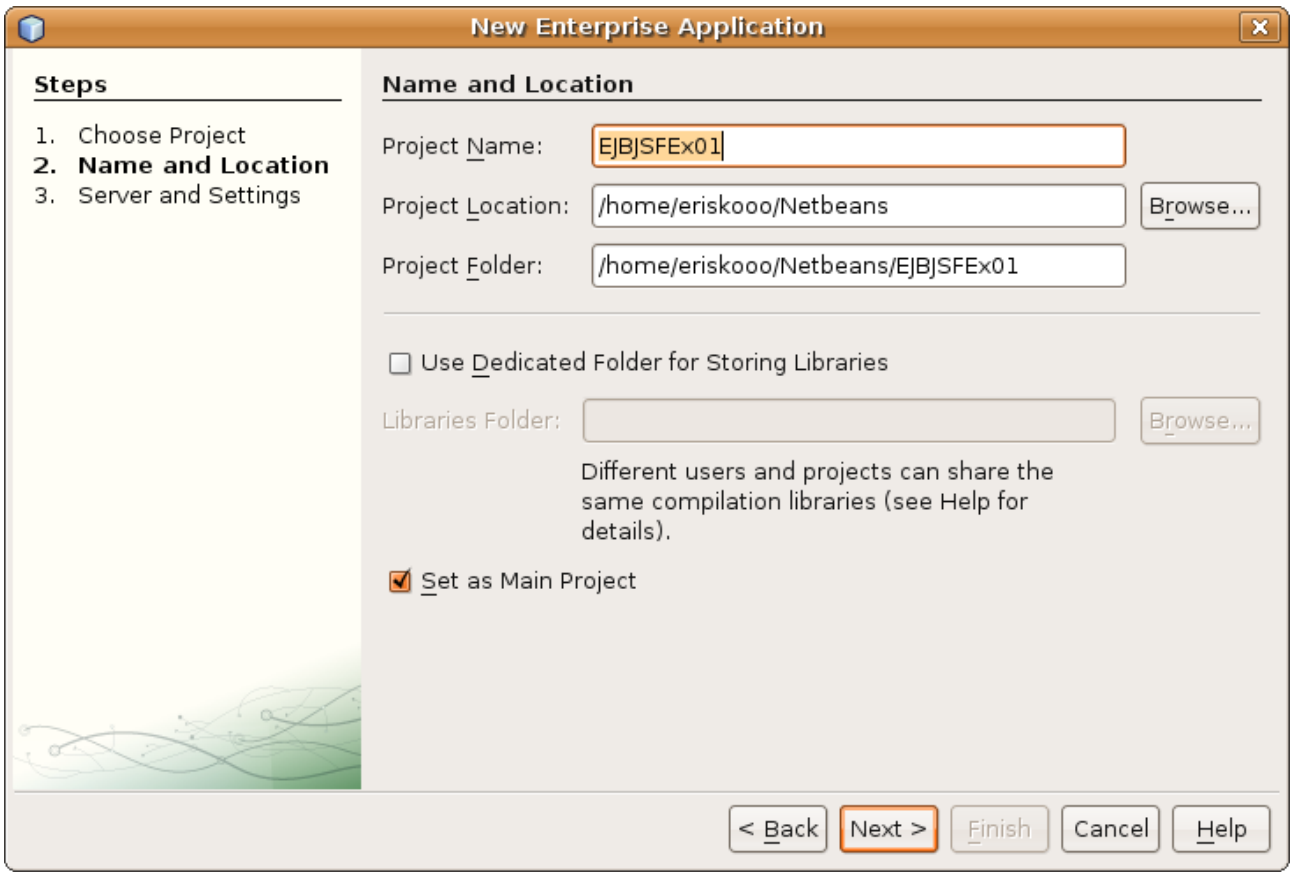
- Netbeans 6.5.1 or above – full bundle (J2EE, Glassfish2)

Creating Project Workspace

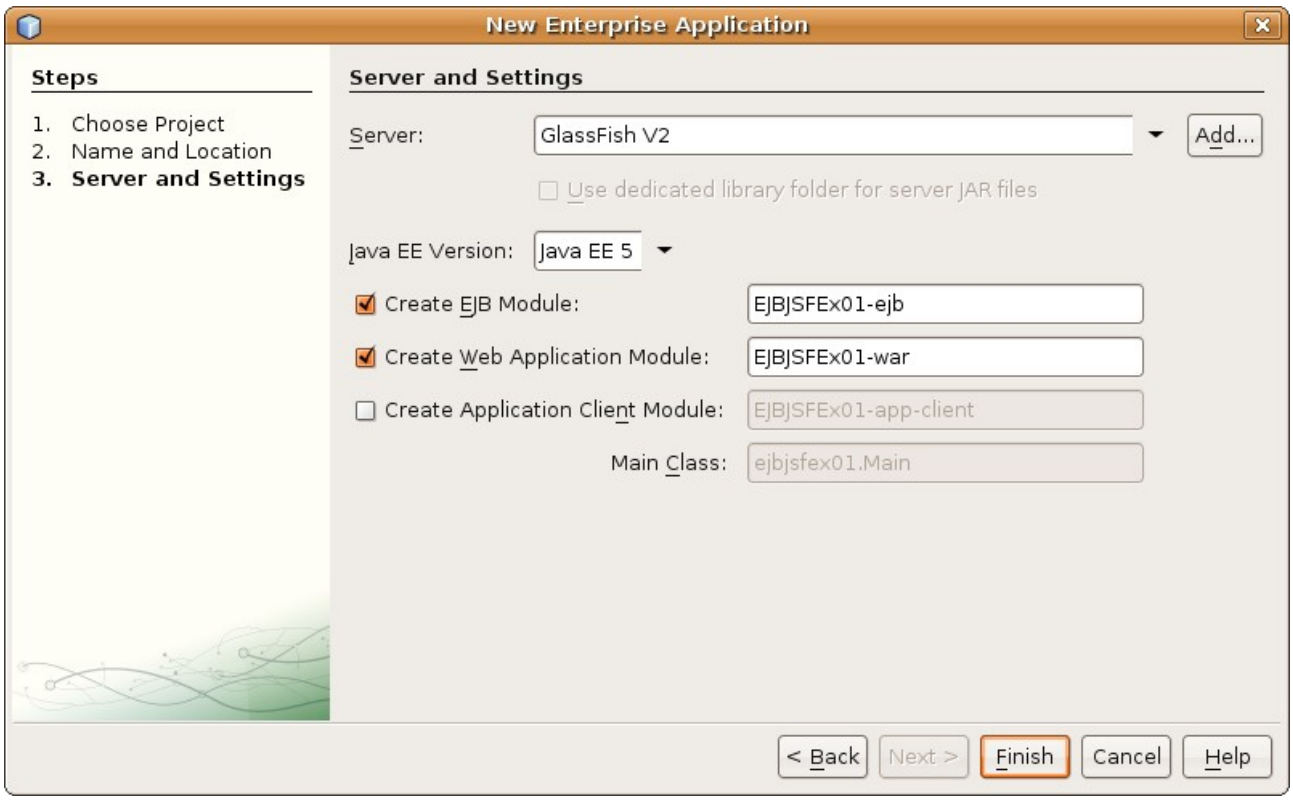
- Open Netbeans IDE
- File, New Project
- Choose Java EE / Enterprise Application



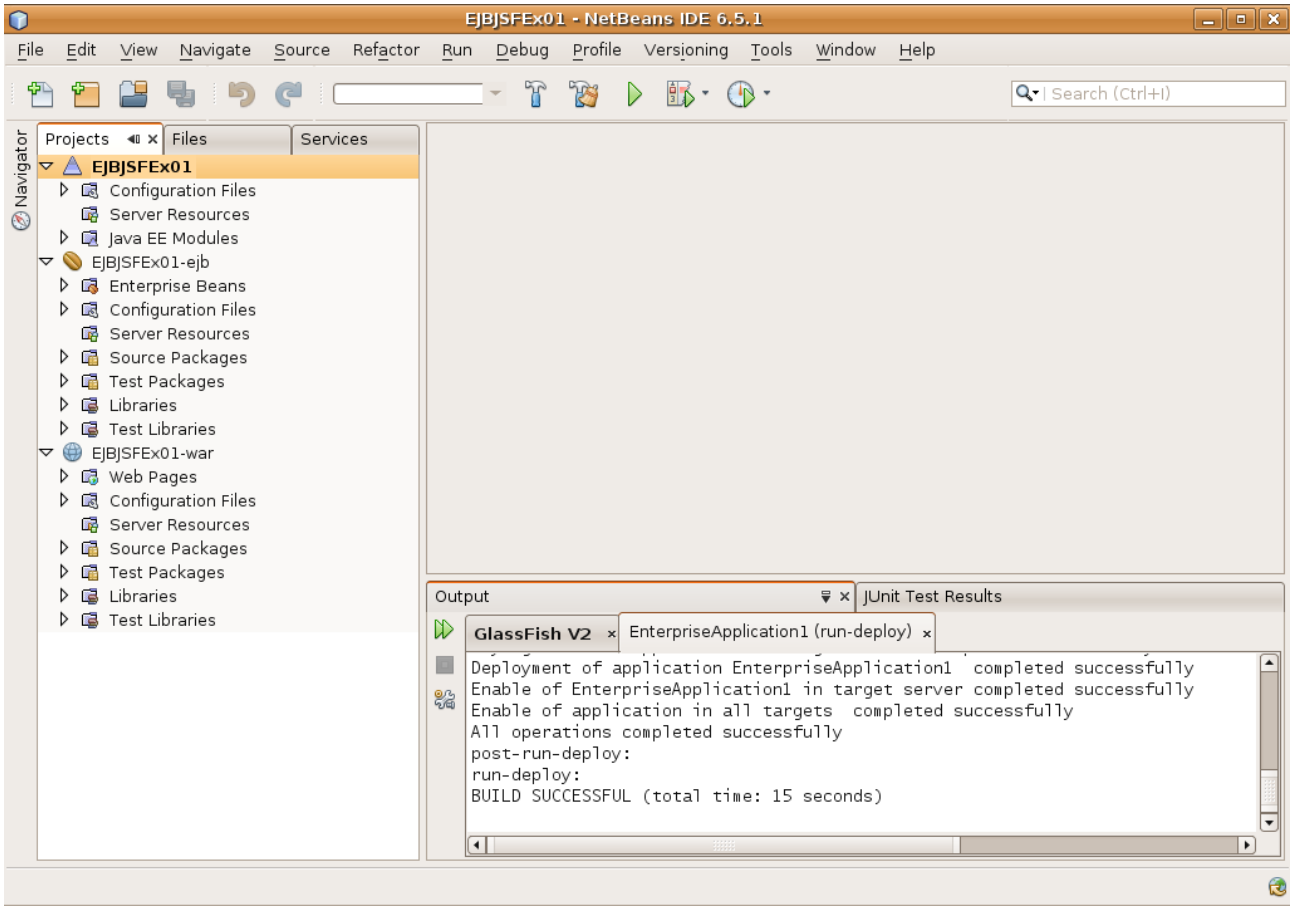
- click Next



- choose project name EJBJSFEx01
- click Next



- click Finish

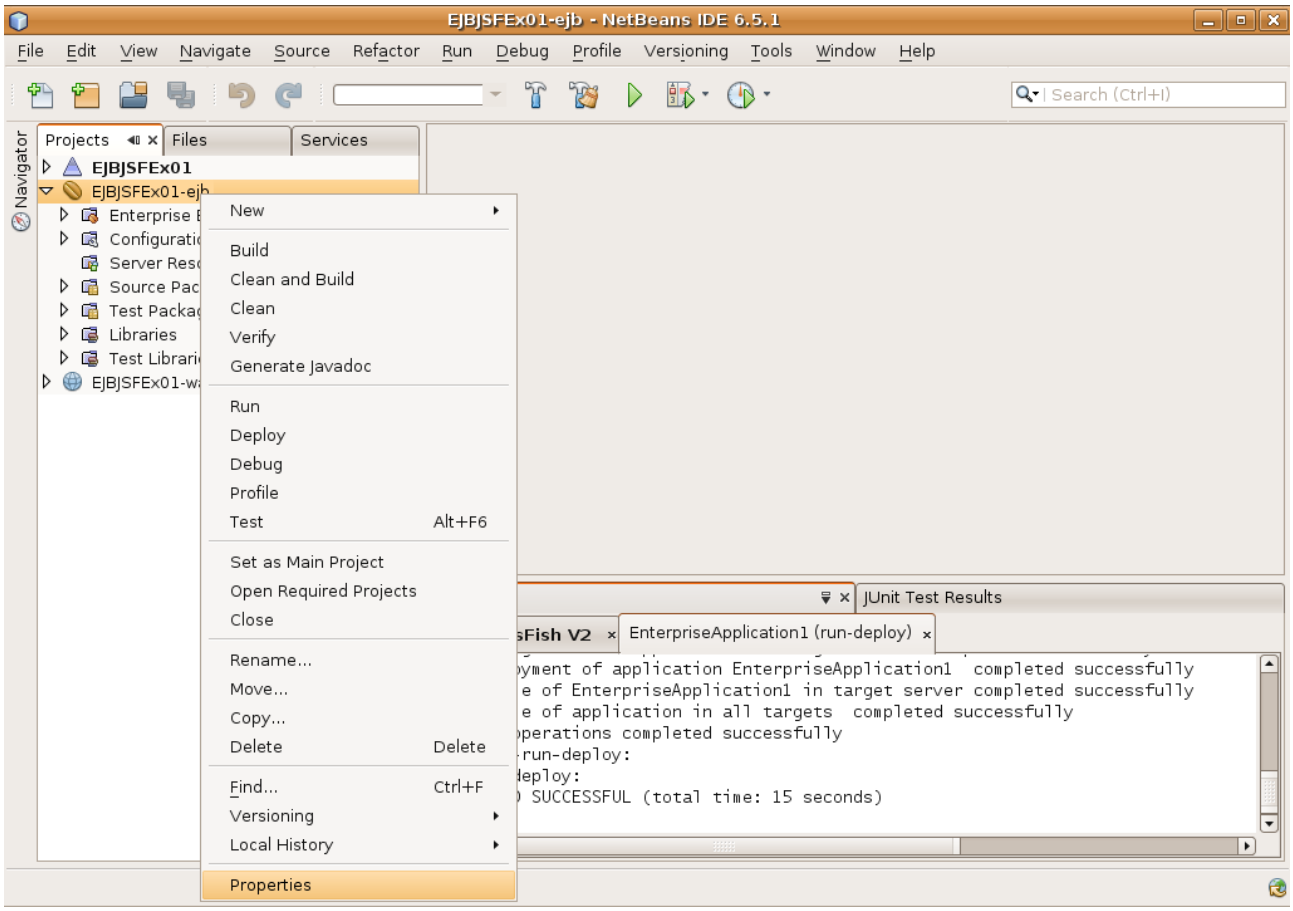


- your IDE should look like on picture above

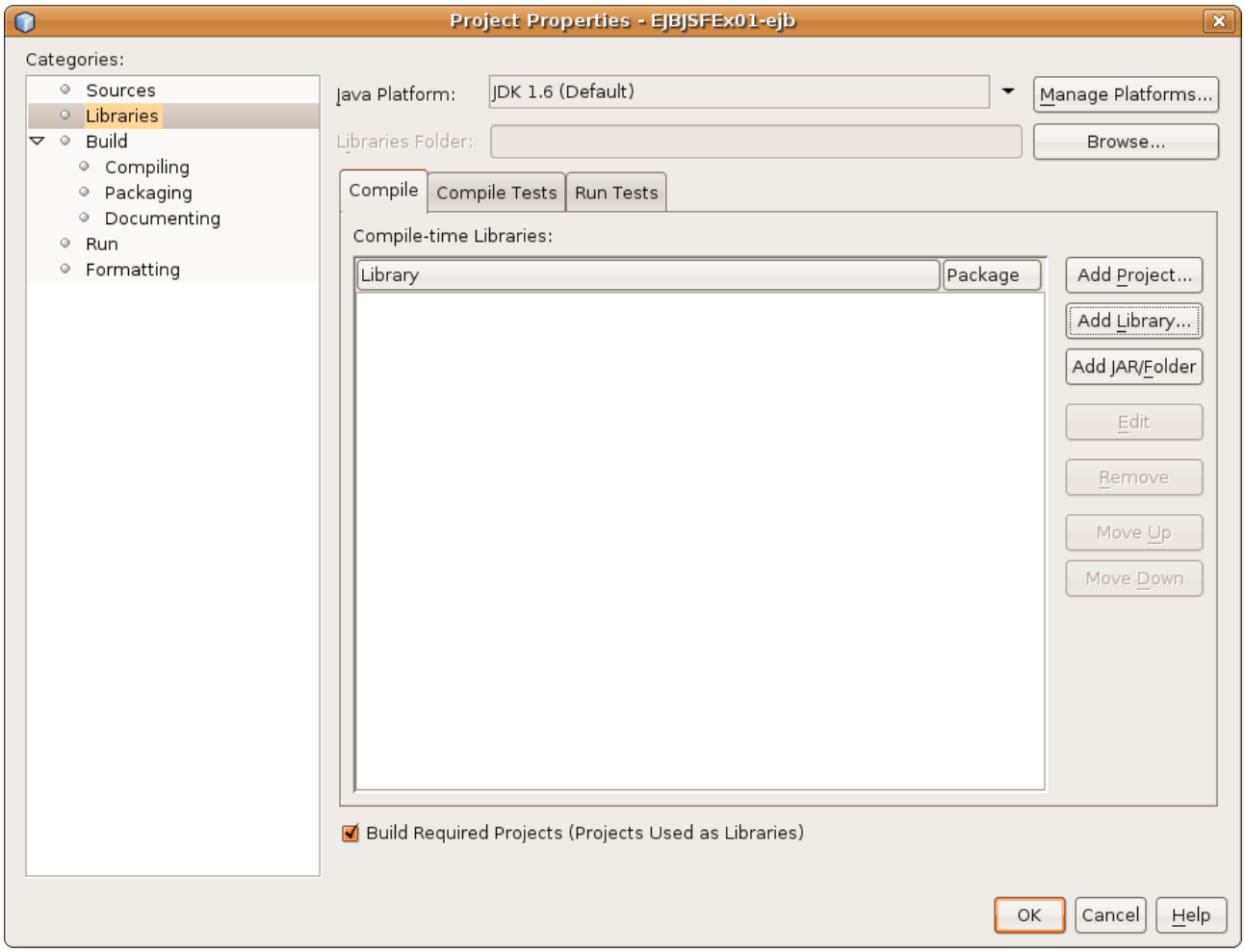
Creating EJB Module

Add neccessary extra libraries to module

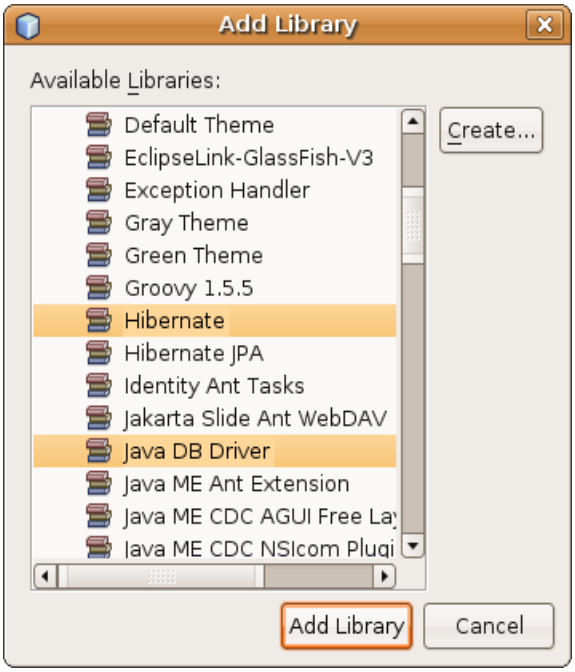
- right click on EJBJSFEx01-ejb, choose properties



- go to node 'Libraries'



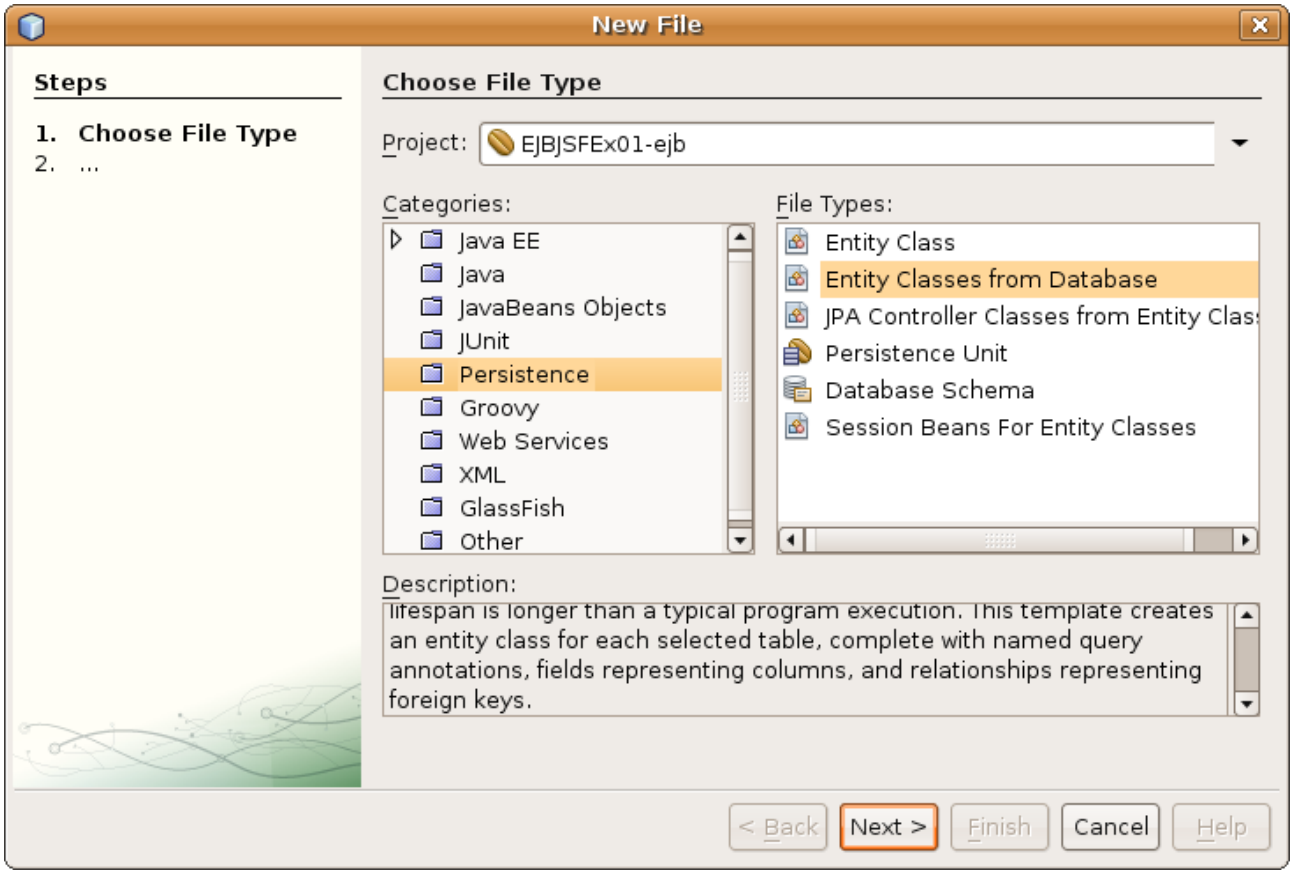
- click on add library button, select Hibernate and JavaDB option



- click on add library, in next screen press ok

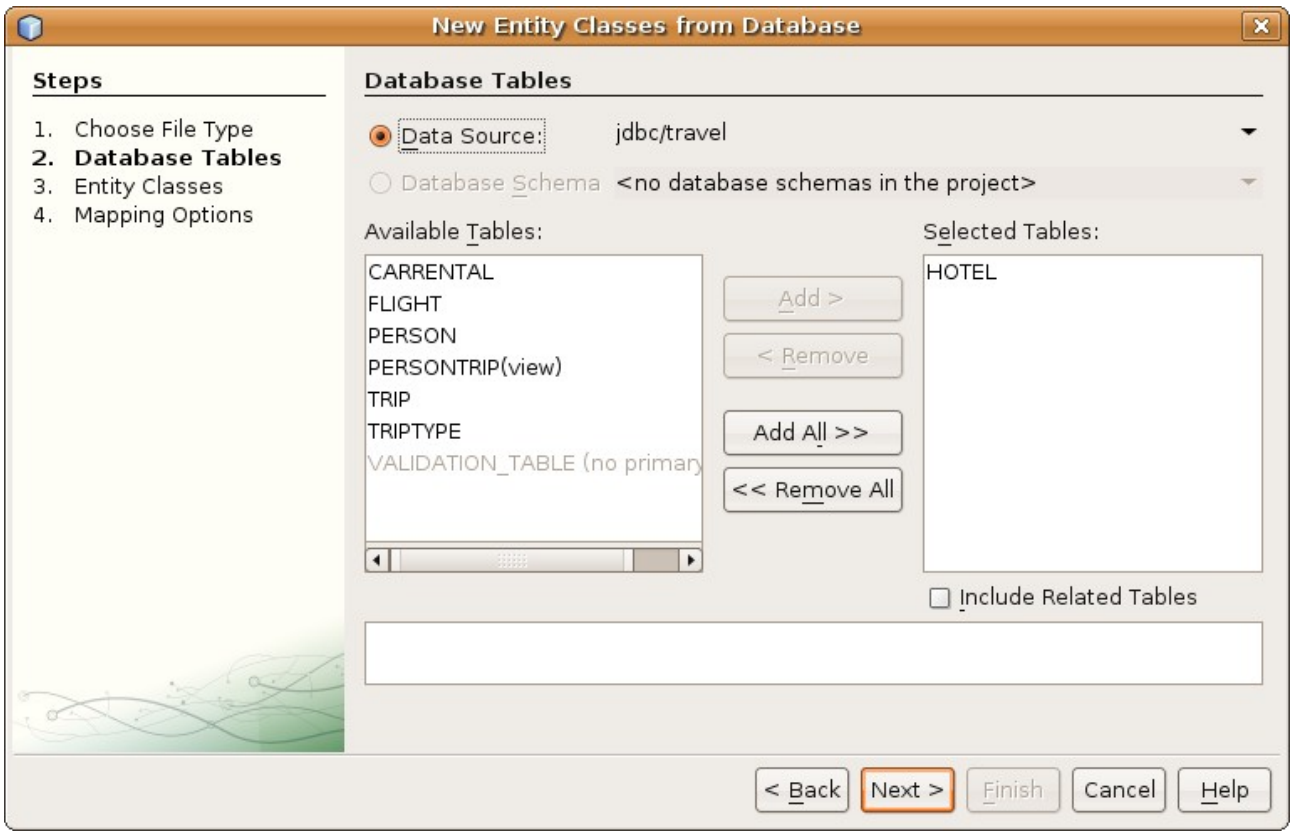
Create Entity Classes from database table

- right click on EJBJSFEx01-ejb, choose new, other
- select persistence, choose Entity classes from database

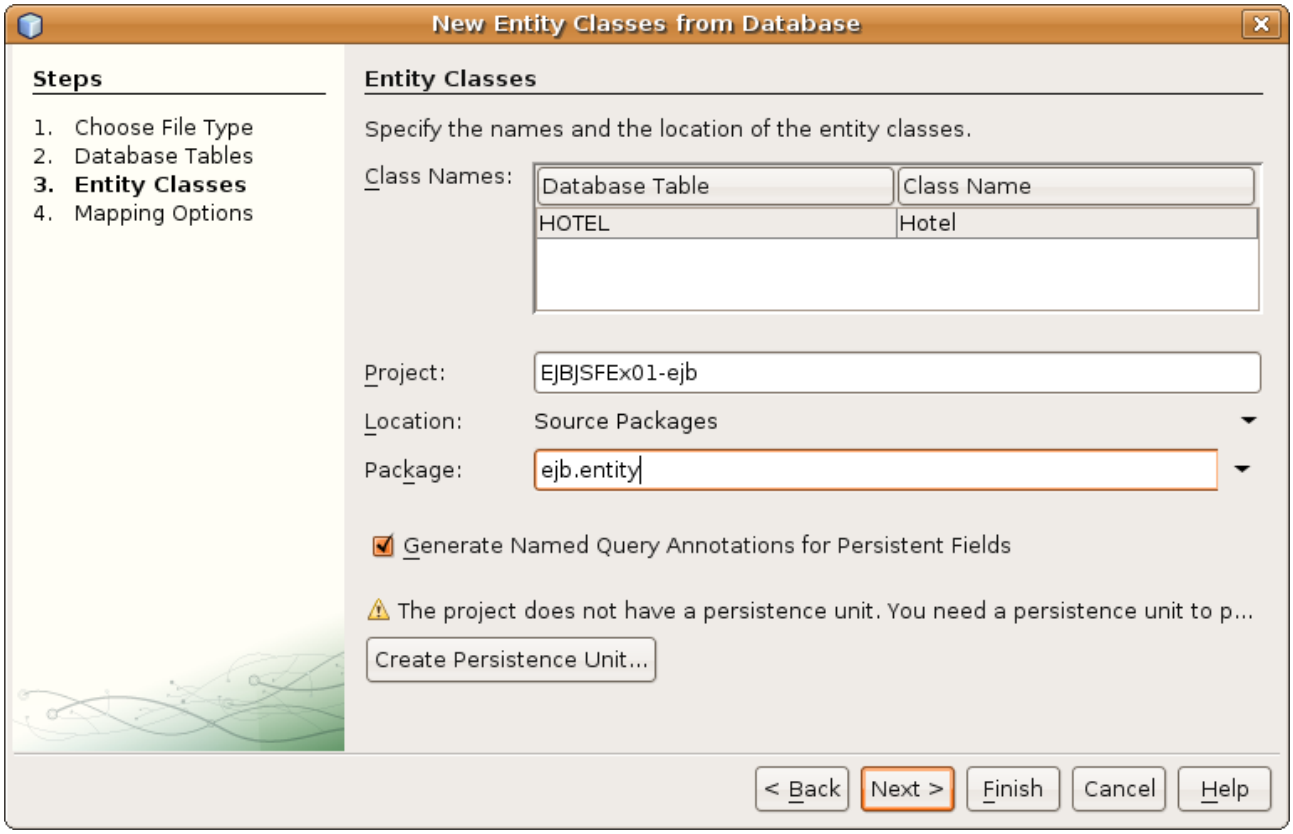


- click next

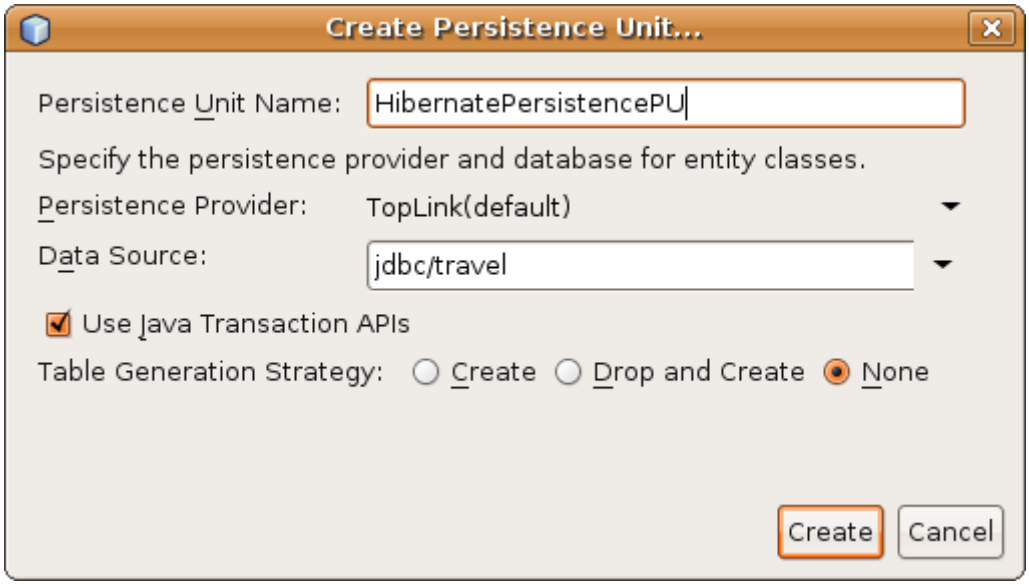
- For this tutorial we use built-in datasource jdbc/travel



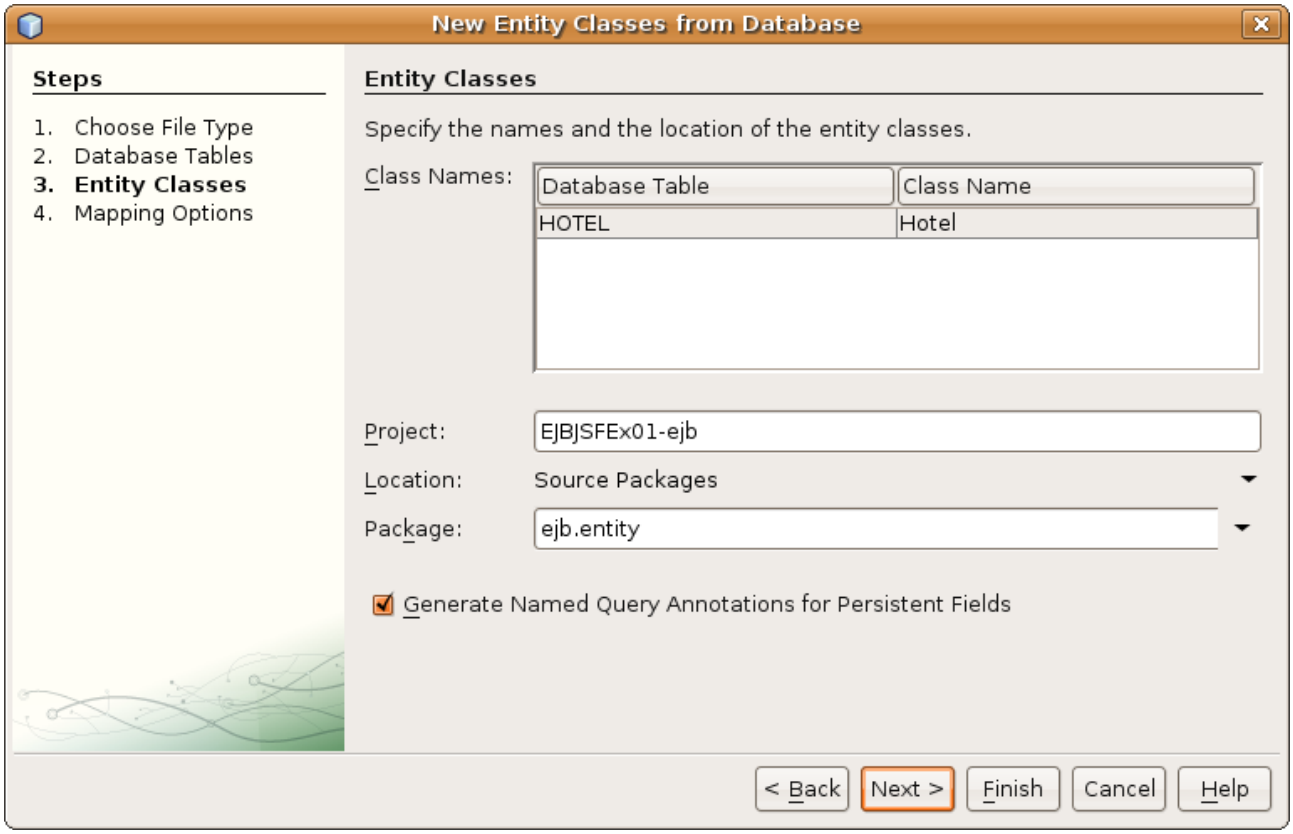
- unmark check box include related tables
- choose option 'Hotel', add to selected tables
- click Next



- modify package name to 'ejb.entity'
- click on 'create persistence unit



- select persistence unit name as 'HibernatePersistencePU'
- select data source to 'jdbc/travel'
- click on 'create'

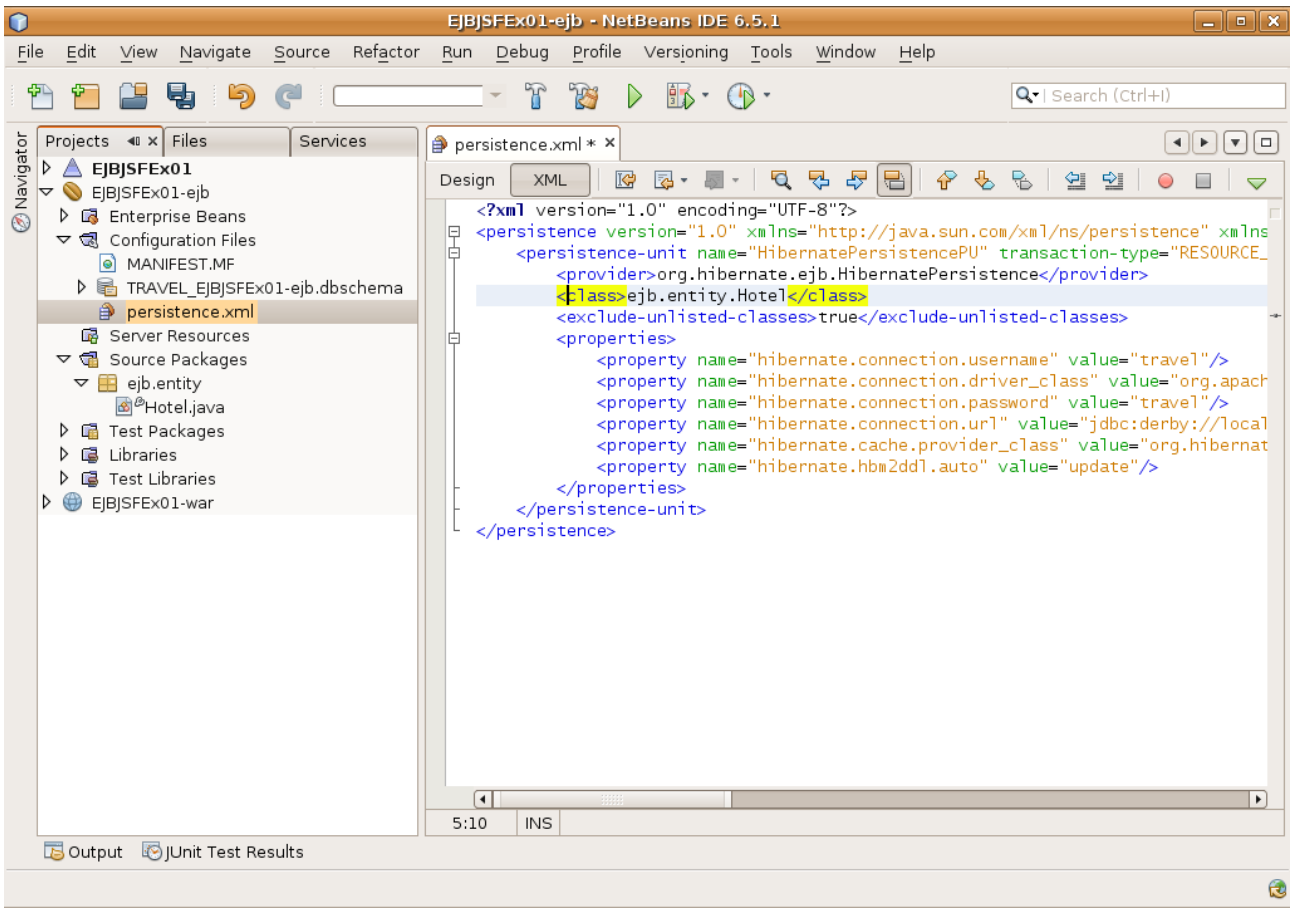


- click on 'next'

- click on 'finish'

Modify Persistence unit

- find file persistence.xml

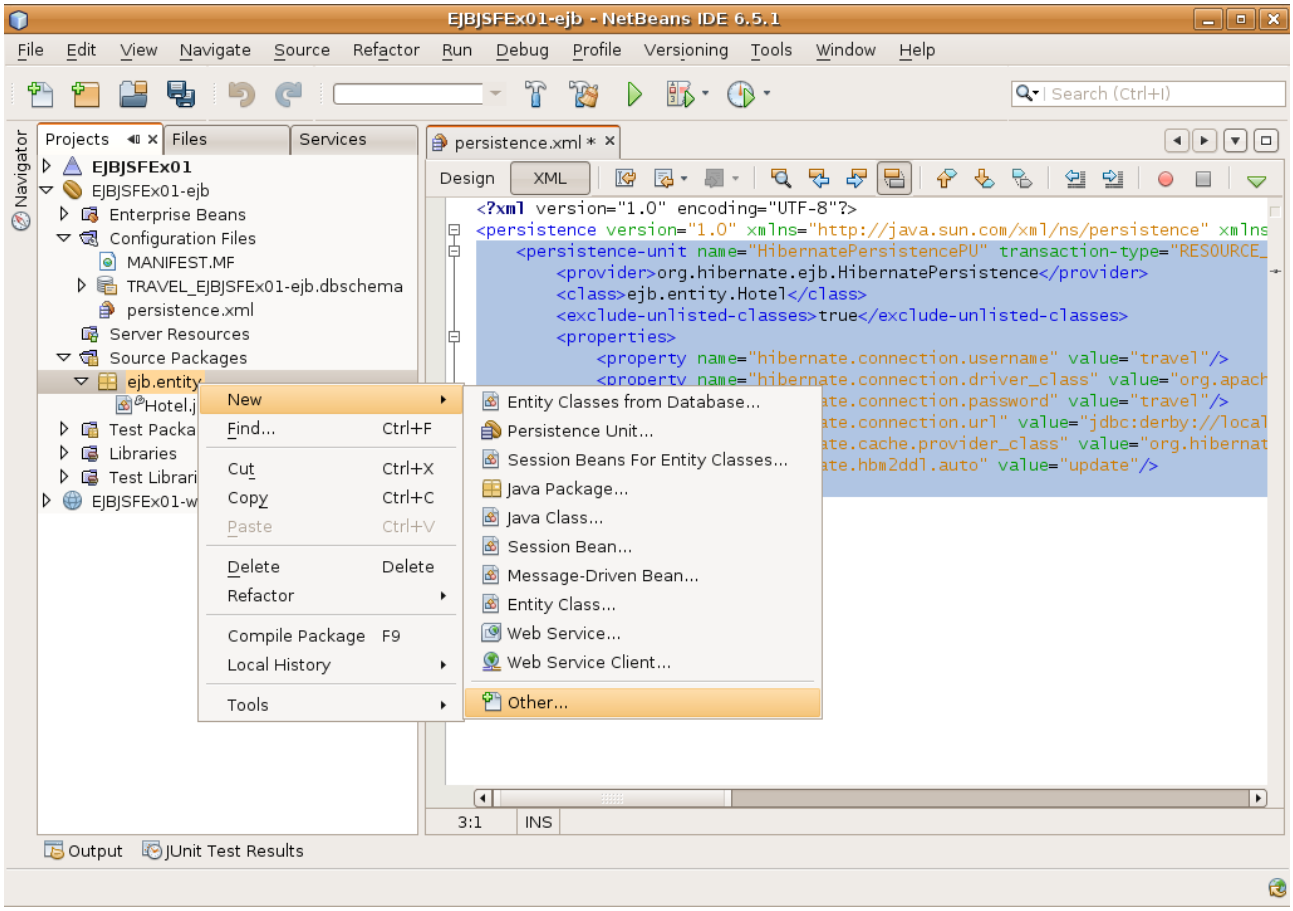


- modify node persistence-unit as shown / or equally to your data source

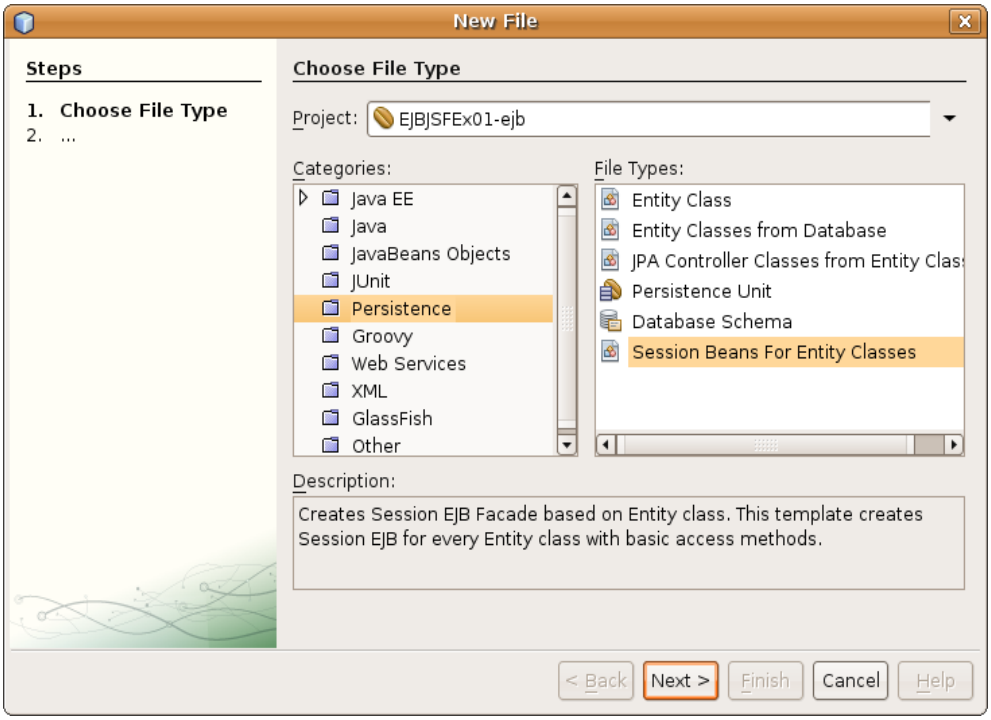
```
<persistence-unit name="HibernatePersistencePU" transaction-type="RESOURCE_LOCAL">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <class>ejb.entity.Hotel</class>
  <exclude-unlisted-classes>true</exclude-unlisted-classes>
  <properties>
    <property name="hibernate.connection.username" value="travel"/>
    <property name="hibernate.connection.driver_class" value="org.apache.derby.jdbc.ClientDriver"/>
    <property name="hibernate.connection.password" value="travel"/>
    <property name="hibernate.connection.url" value="jdbc:derby://localhost:1527/travel"/>
    <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
    <property name="hibernate.hbm2ddl.auto" value="update"/>
  </properties>
</persistence-unit>
```

Create Stateless Session Beans for Entity Classes

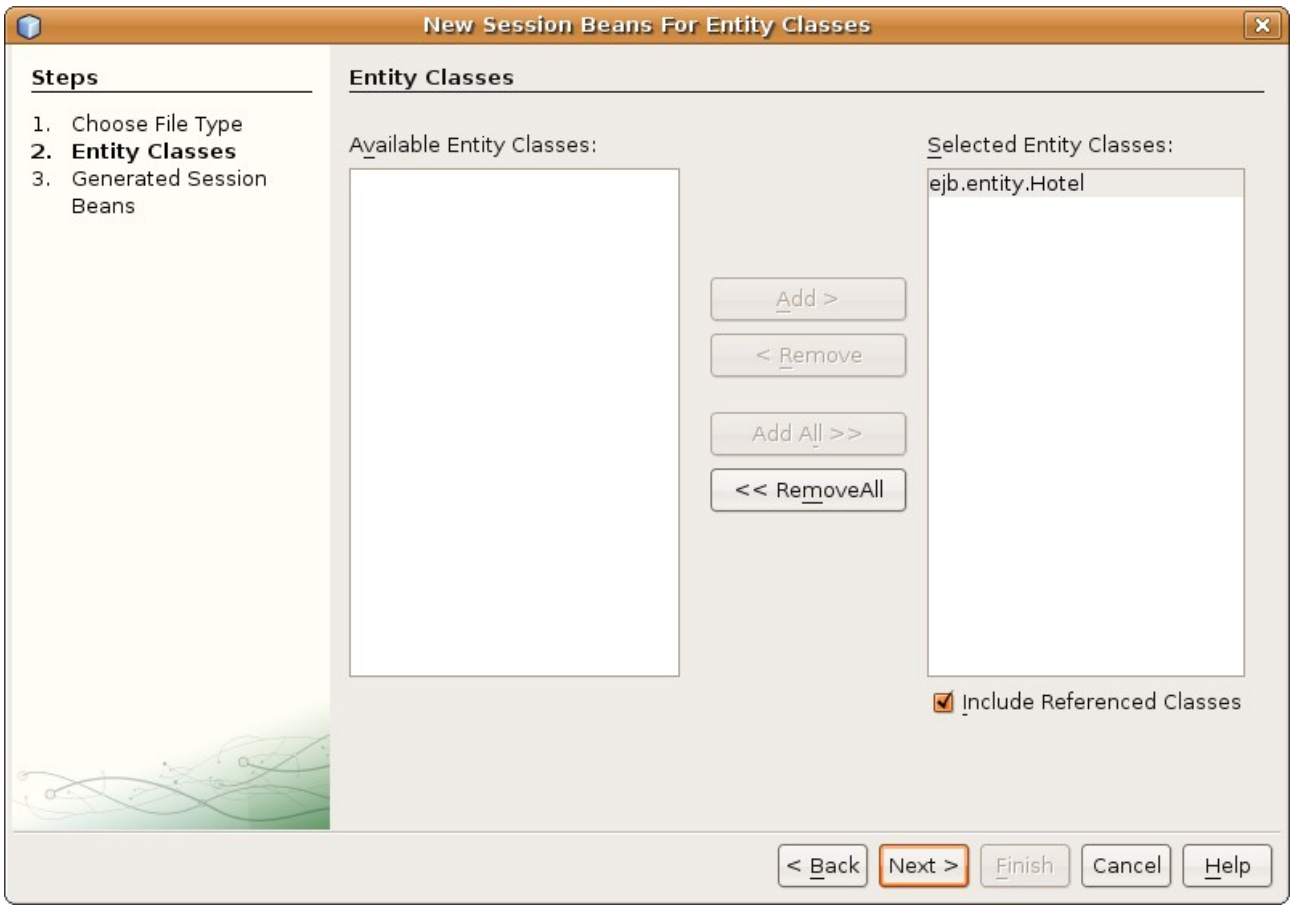
- right-click on ejb.entity package, select new, select other



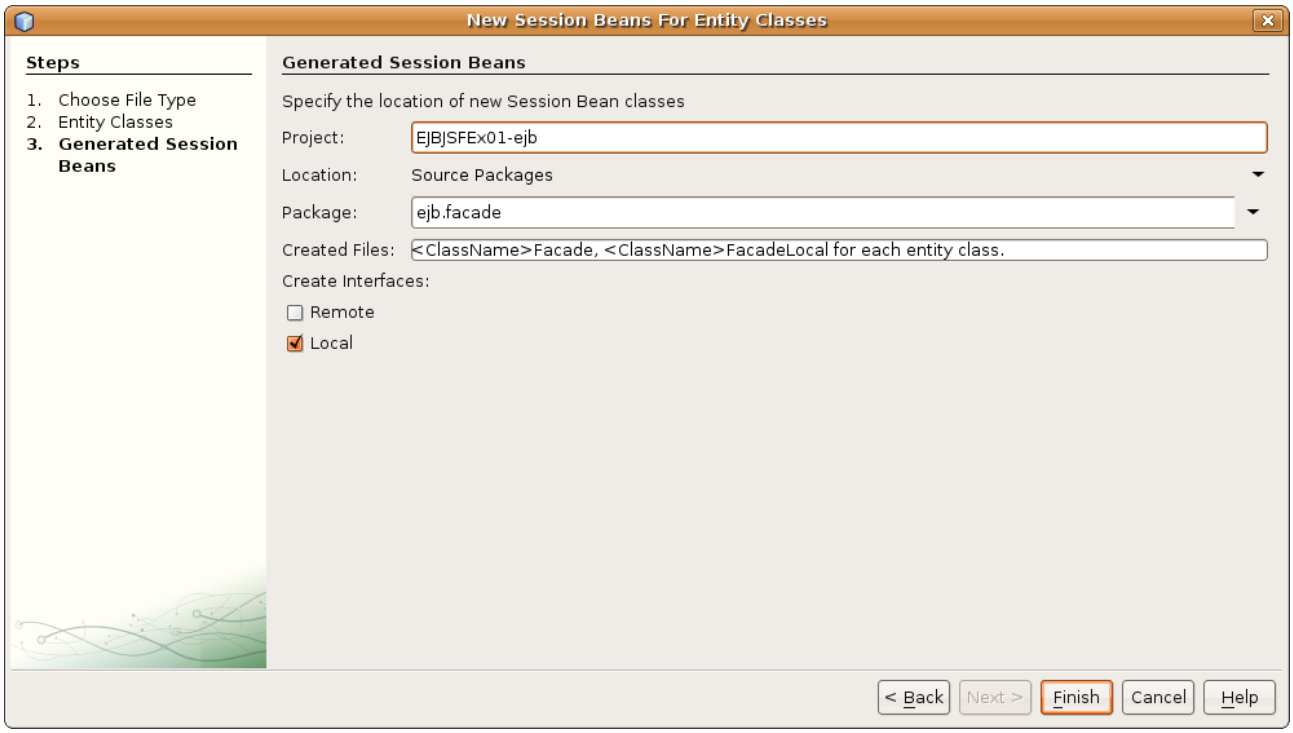
- choose 'persistence' and 'session beans for entity classes'



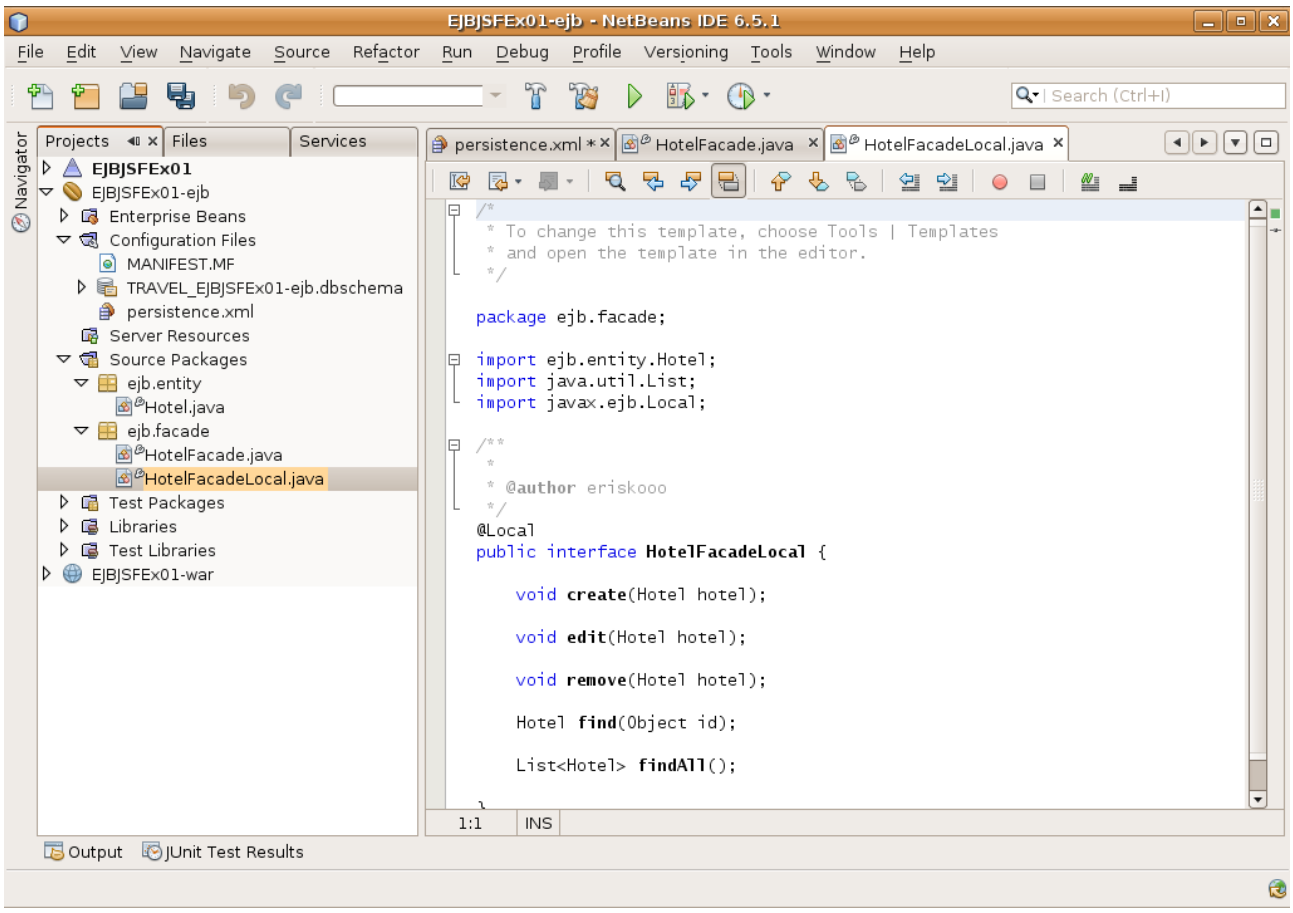
click next



- add ejb.entity.Hotel class to selected entity classes field
- click next



- modify package name to ejb.facade
- click finish

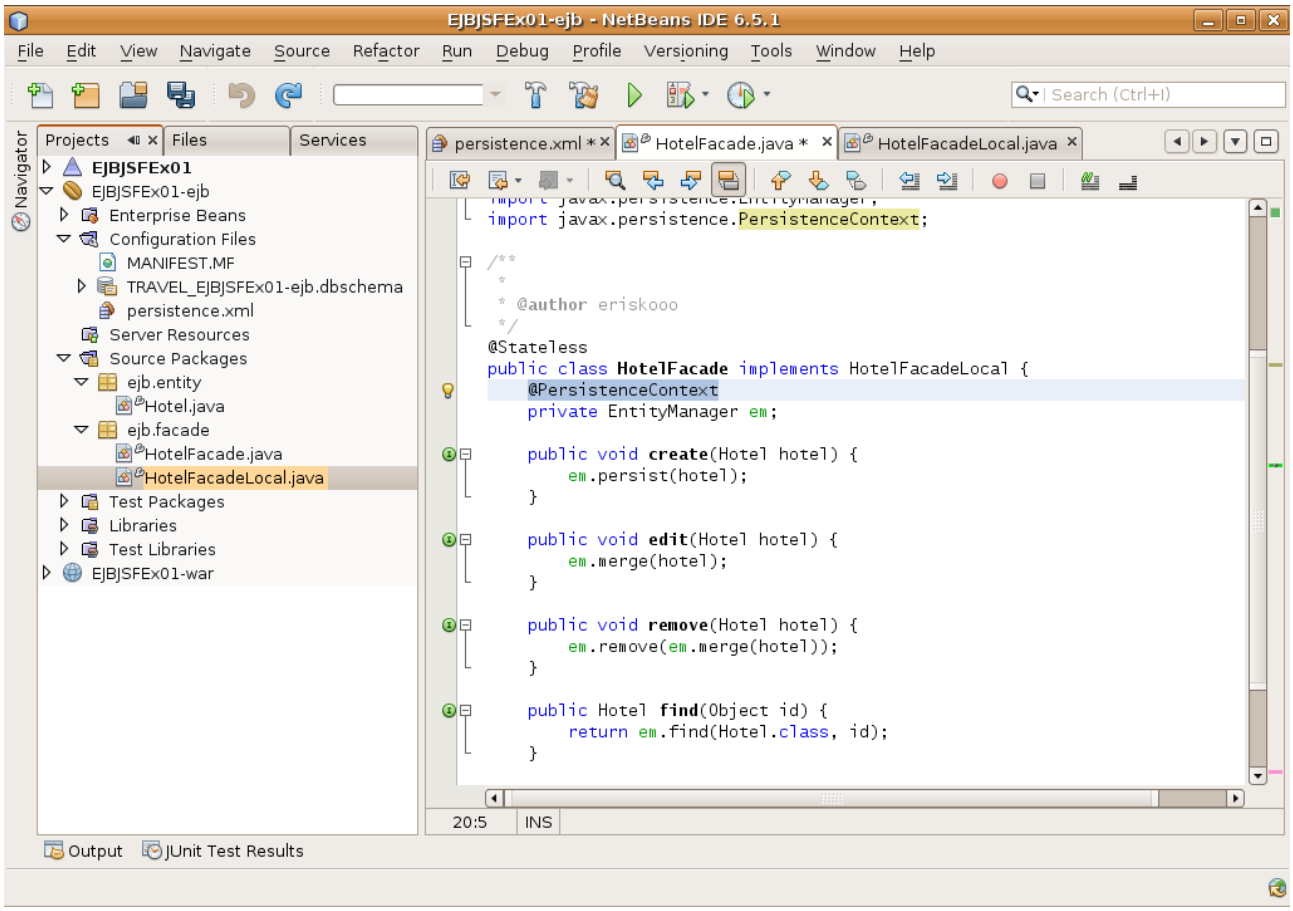


- your IDE should look like on picture above

Modify receiving of entity-manager in generated session-bead

Class HotelFacade represents your stateless session bean (EJB3 specification). Now we modify receiving instance of entity manager, class responsible for persistence management.

- Remove annotation @PersistenceContext before declaration of EntityManager



- Add method to class to get EntityManager, as singleton

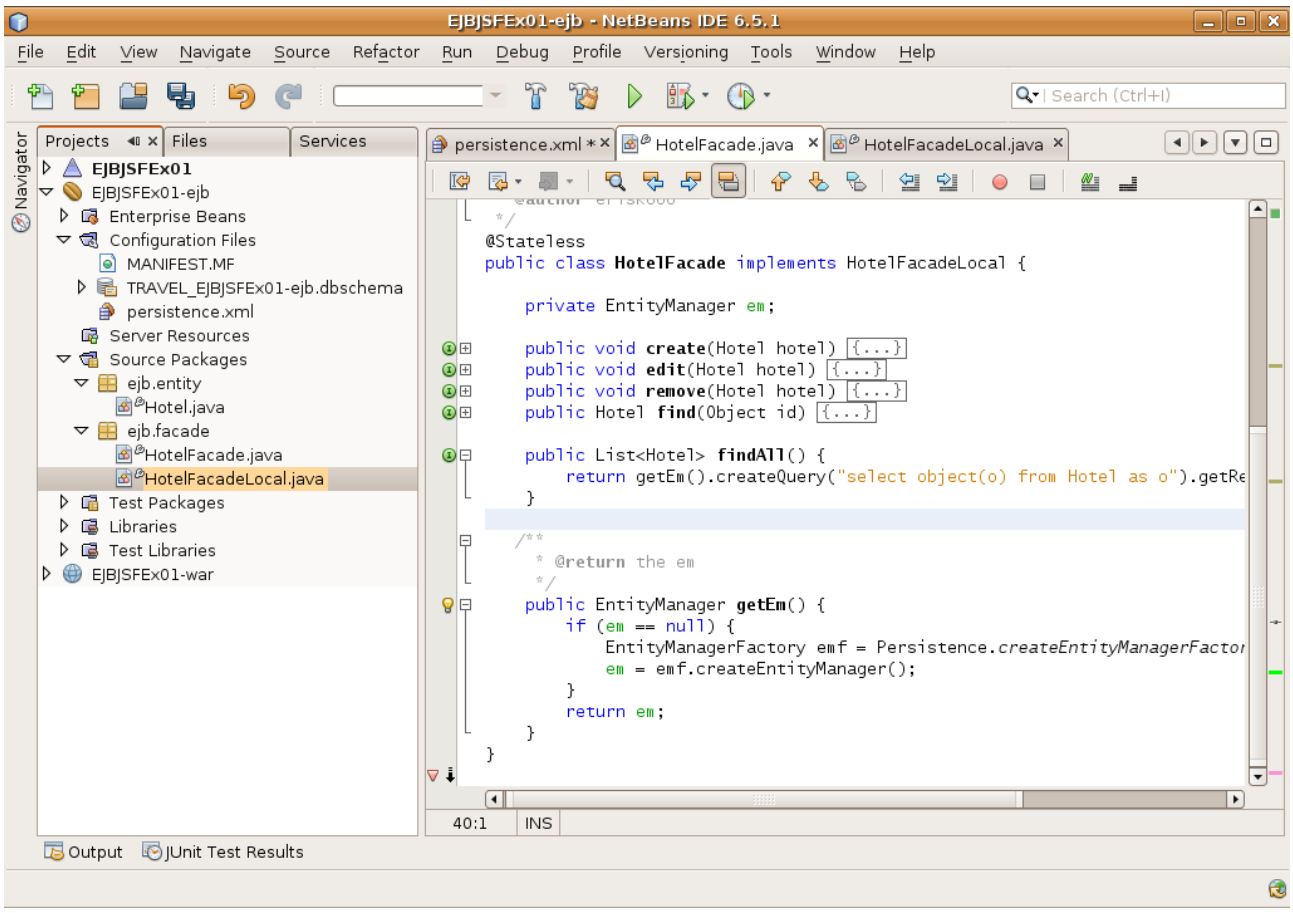
```
/**
 * @return the em
 */
public EntityManager getEm() {
    if (em == null) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("HibernatePersistencePU");
        em = emf.createEntityManager();
    }
    return em;
}
```

- fix neccesaryy import by pressung ctrl+shift+i

In our application we will use only one method in bean – find all. Therefore we modify this class like this :

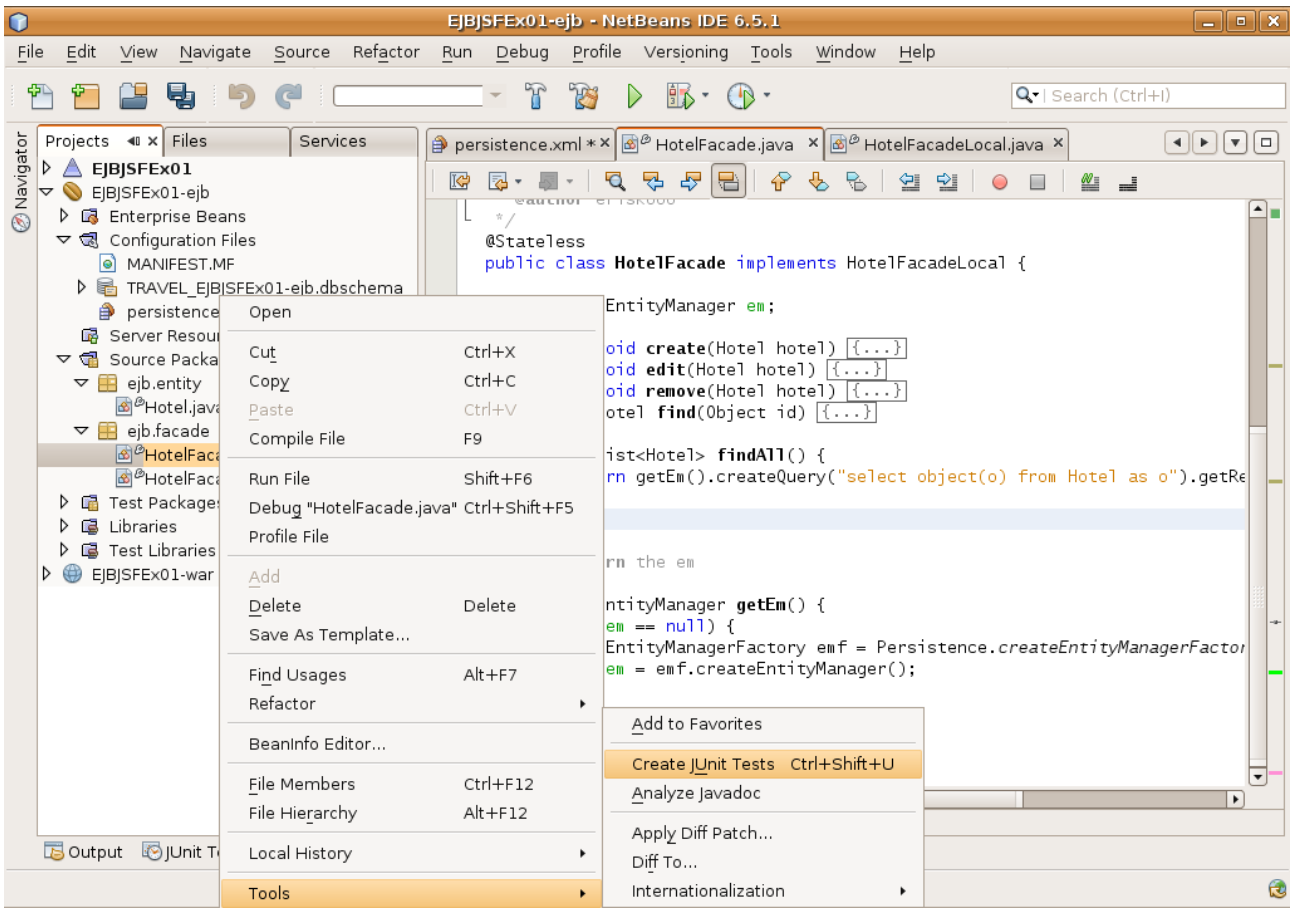
```
public List<Hotel> findAll() {
    return getEm().createQuery("select object(o) from Hotel as o").getResultList();
}
```

- our session bean should look like

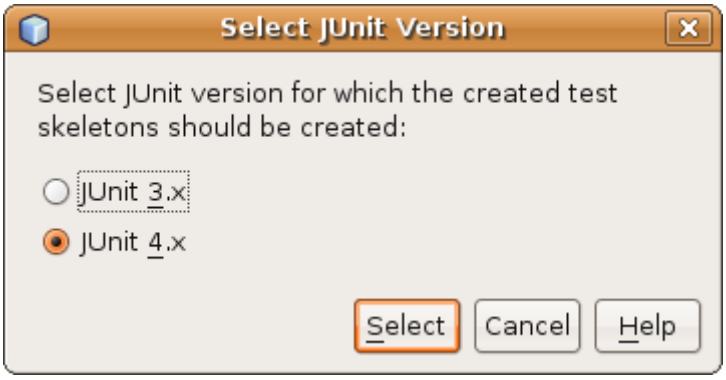


Test created session bean

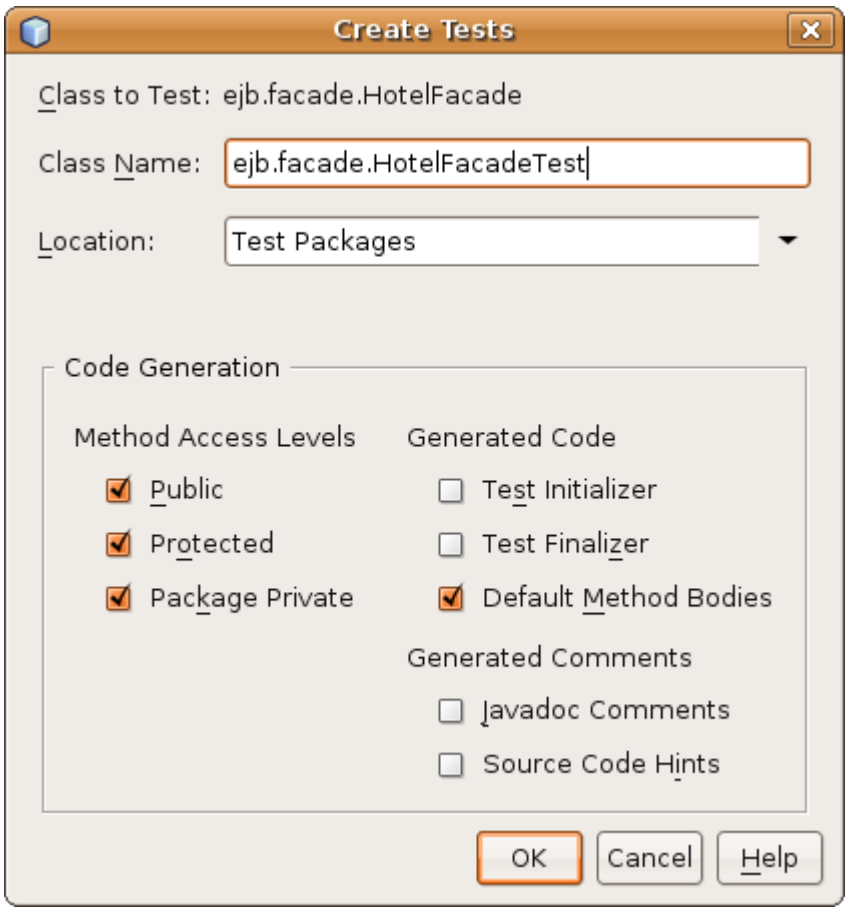
- to test create bean, right click on 'HotelFacade' bean, select tools, then create JUnit test



- select JUnit 4.x



- click 'select'

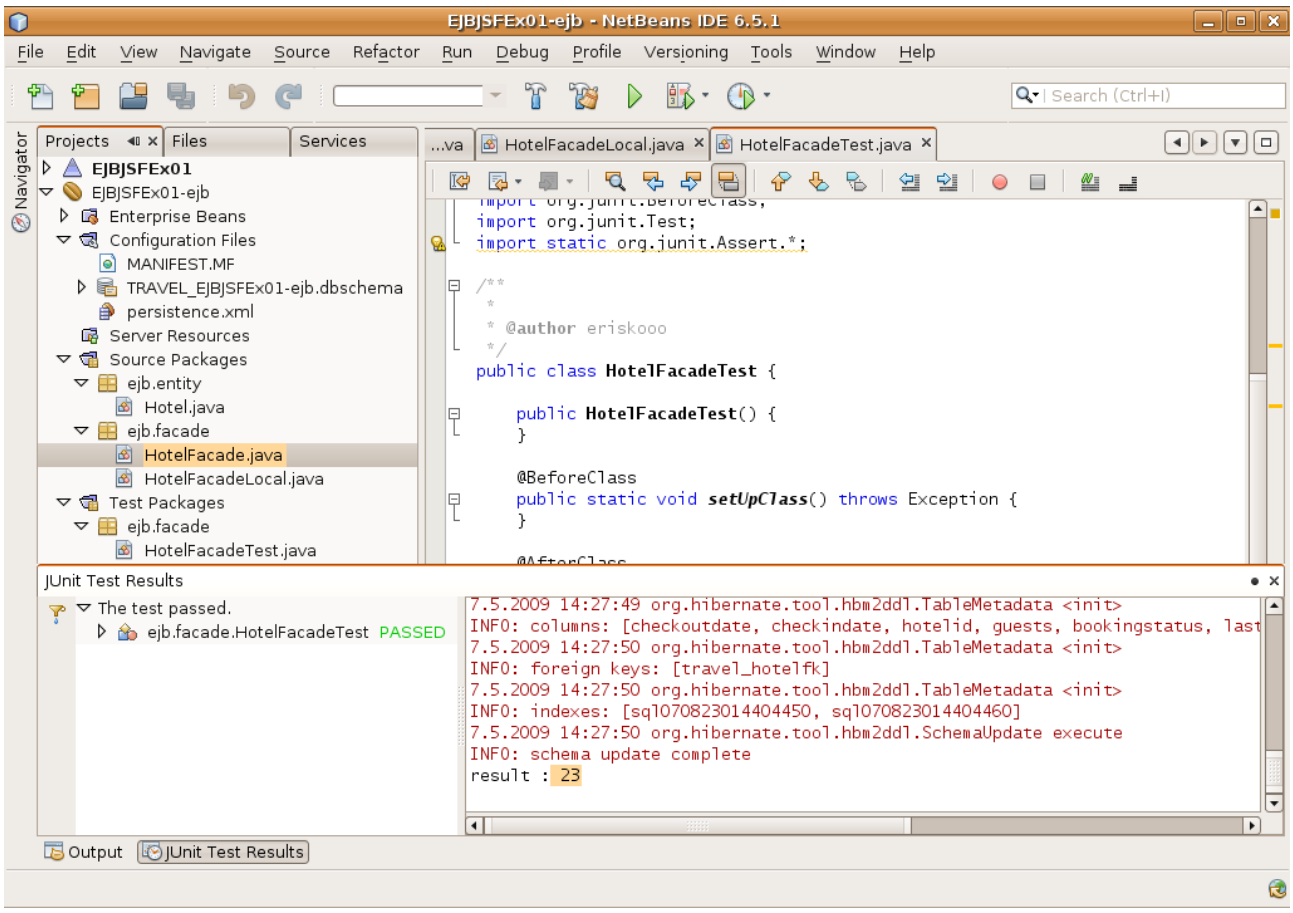


- modify fields as on picture above
- click 'ok'

Remove all methods anotated with @Test except for 'public void testFindAll'. In this test we would like to see as output, how many 'Hotel' items we have in our database. Therefore modify method testFindAll as this :

```
@Test
public void testFindAll() {
    System.out.println("findAll");
    HotelFacade instance = new HotelFacade();
    List<Hotel> result = instance.findAll();
    System.out.println("result : " + result.size());
}
```

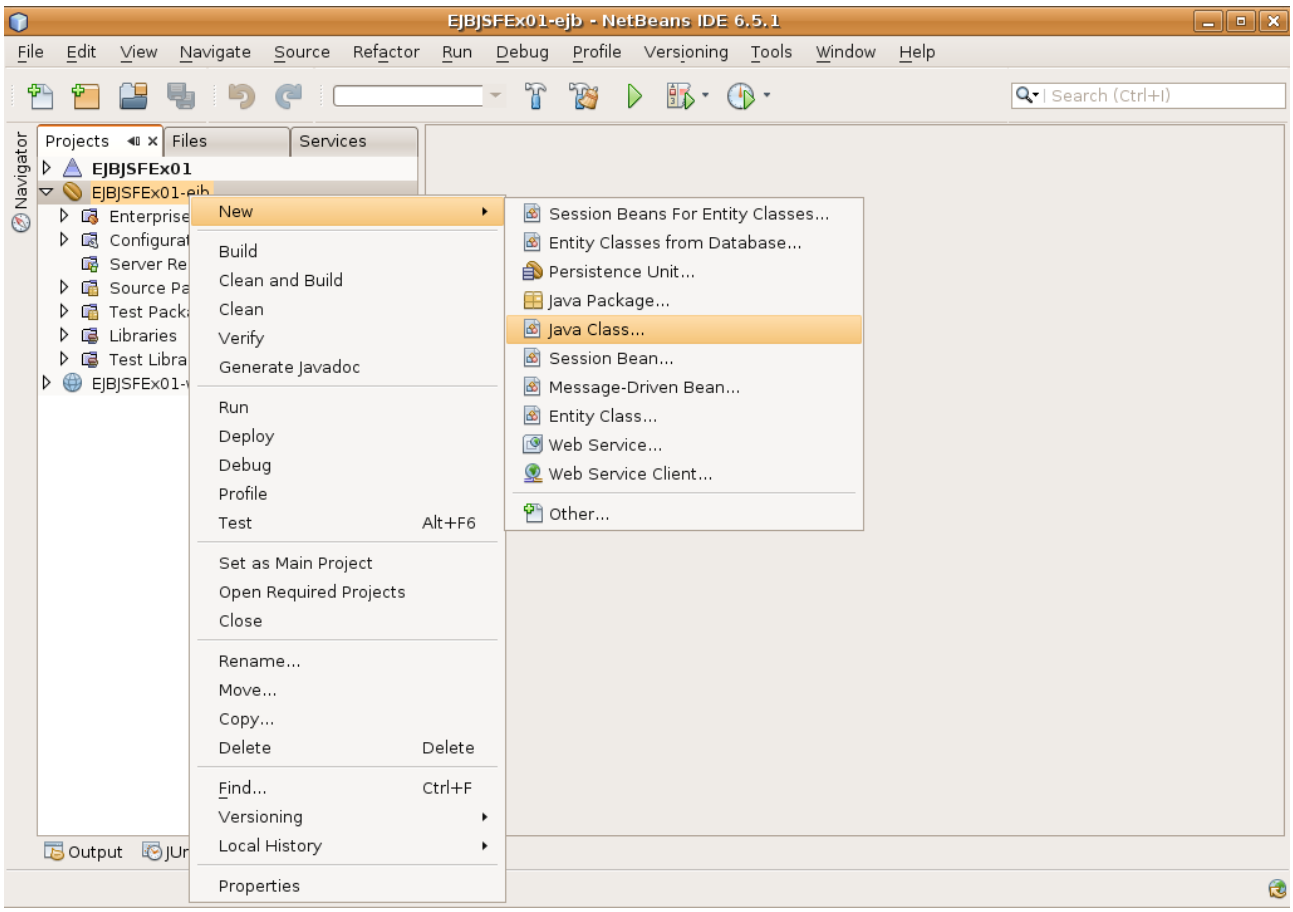
Press shift + F6 to run JunitTest. Your output should look like



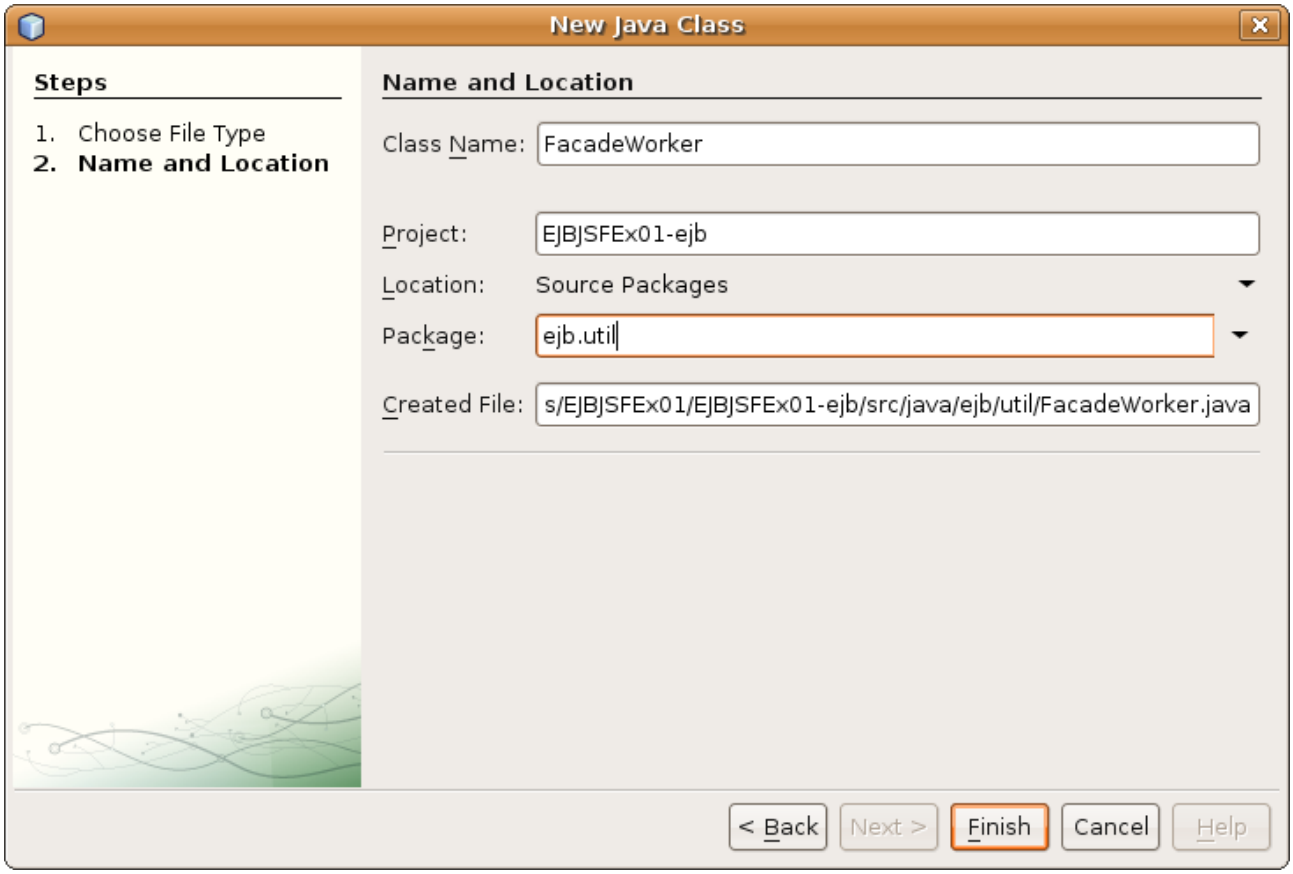
Now, our application is able to communicate with data source. In next step we create Web part of application, which will call methods of HotelFacade session bean.

Our Web part will use FacadeWorker class, which will create singleton instance of requested Facade

- click on 'EJBJSFEx01-ejb' node, select new, select Java Class, if is not visible, find it at 'other' node



- name new Class FacadeWorker, select package 'ejb.util'

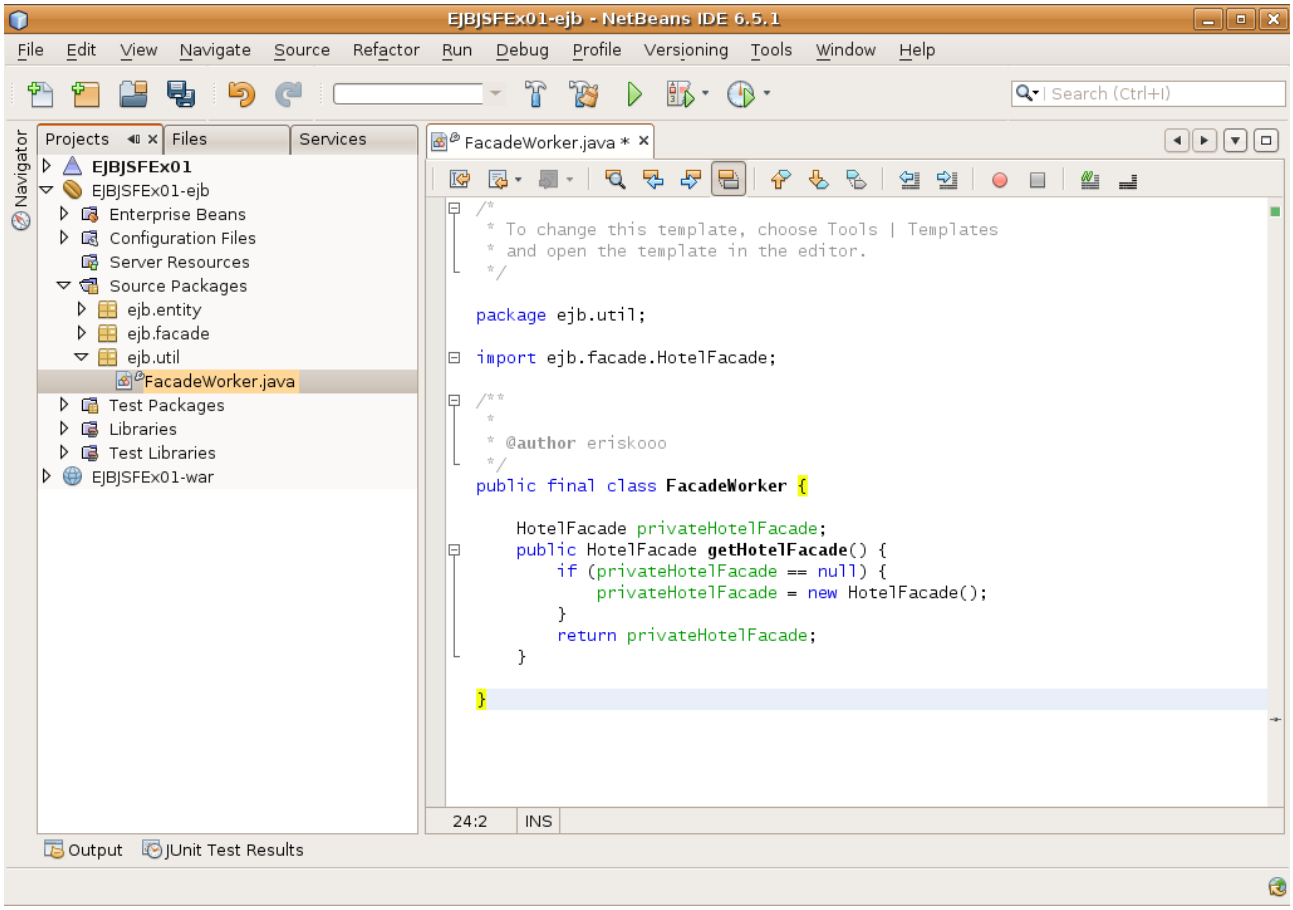


- click 'finish'

to newly created class FacadeWorker insert method, which will grant us singleton instance of HotelFacade

```
public final class FacadeWorker {
   HotelFacade privateHotelFacade;
   public HotelFacade getHotelFacade() {
      if (privateHotelFacade == null) {
         privateHotelFacade = new HotelFacade();
      }
      return privateHotelFacade;
   }
}
```

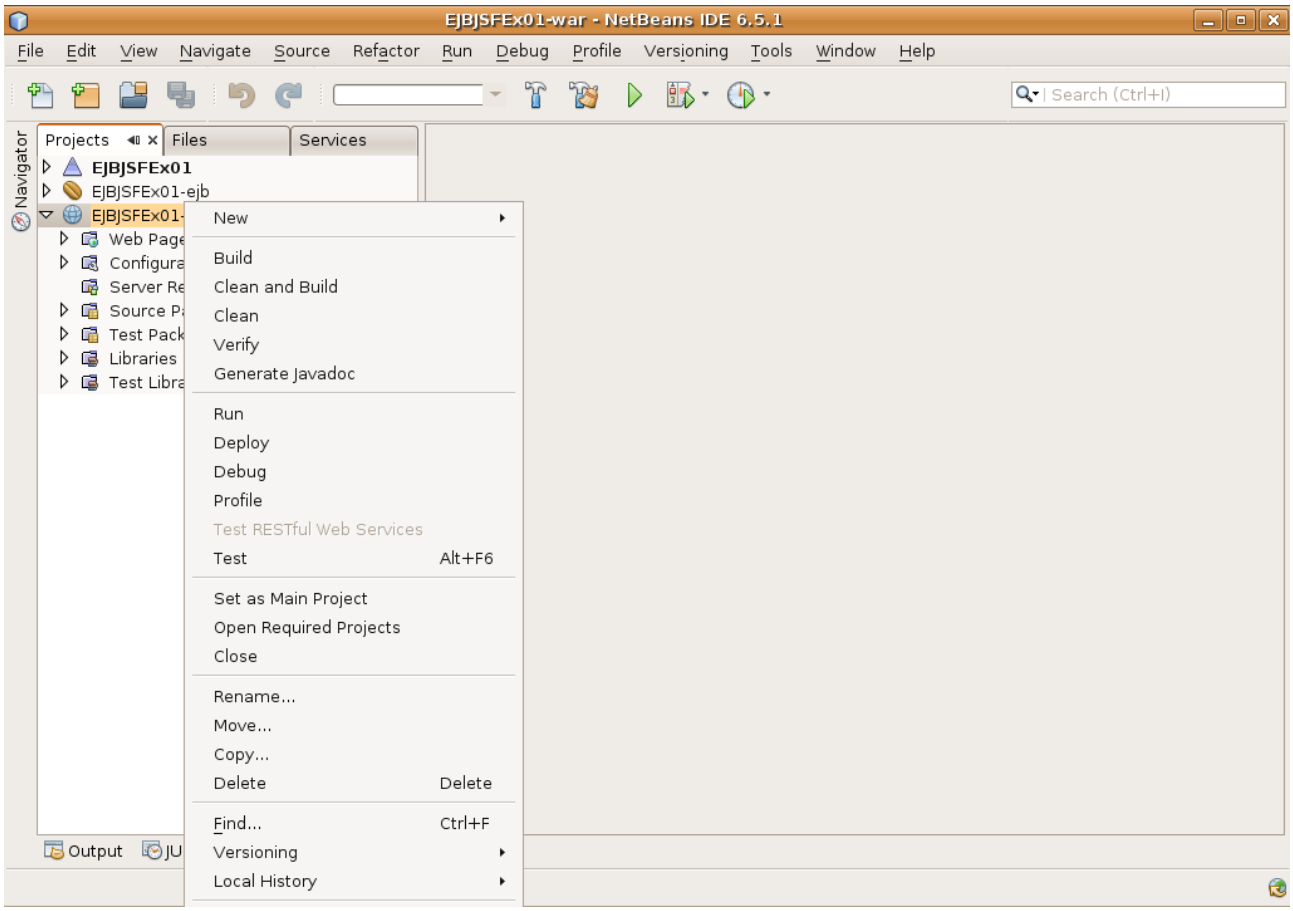
- fix imports by pressing ctrl+shift+i
- your screen should look like



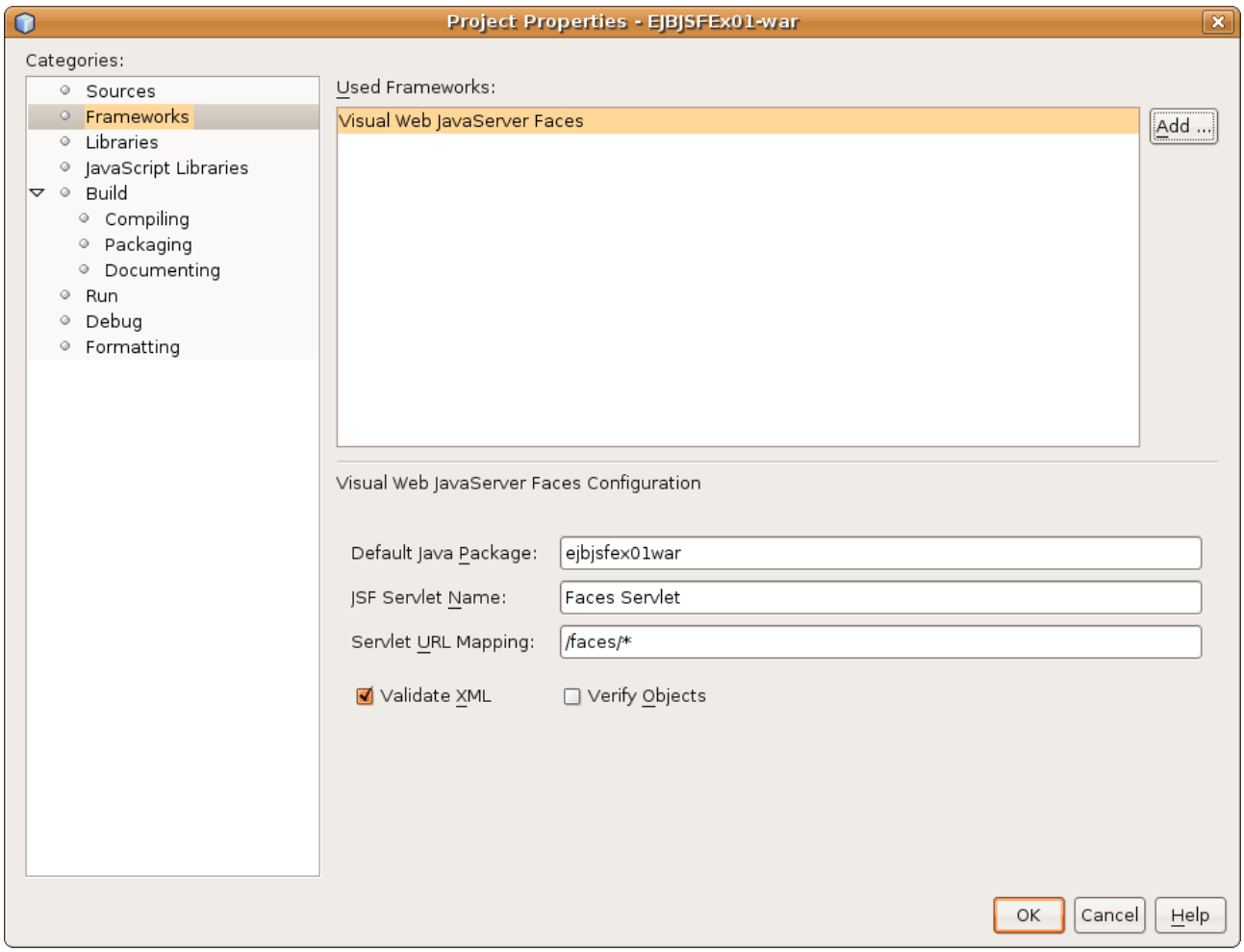
Creating Web Module

In our workspace we have pre-generated Web project called 'EJBJSFEx01-war'

Add requested Visual JSF Technology ability to our web – project



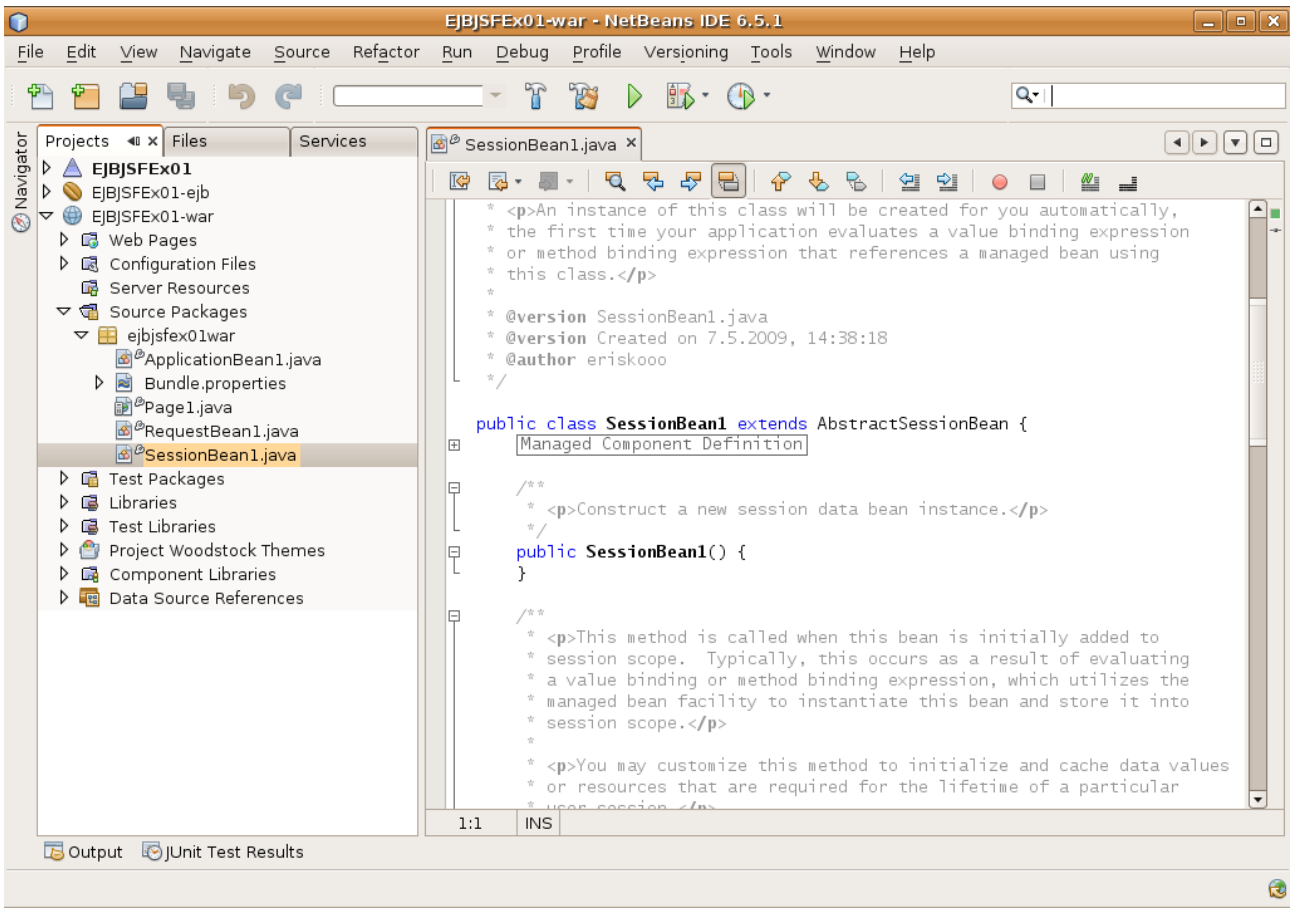
- right-click on EJBJSFEx01-war, select all way down properties (not visible on screenshot)



- select Framework node, add Visual Web JavaServer Faces
- click ok

Now we create part of application, which will grant 'binding' of data received through our 1st EJB project, through FacadeWorker class.

- Find in project file 'SessionBean1'



- add to file declaration of proprery representing FacadeWorker, and create singleton getter to it.

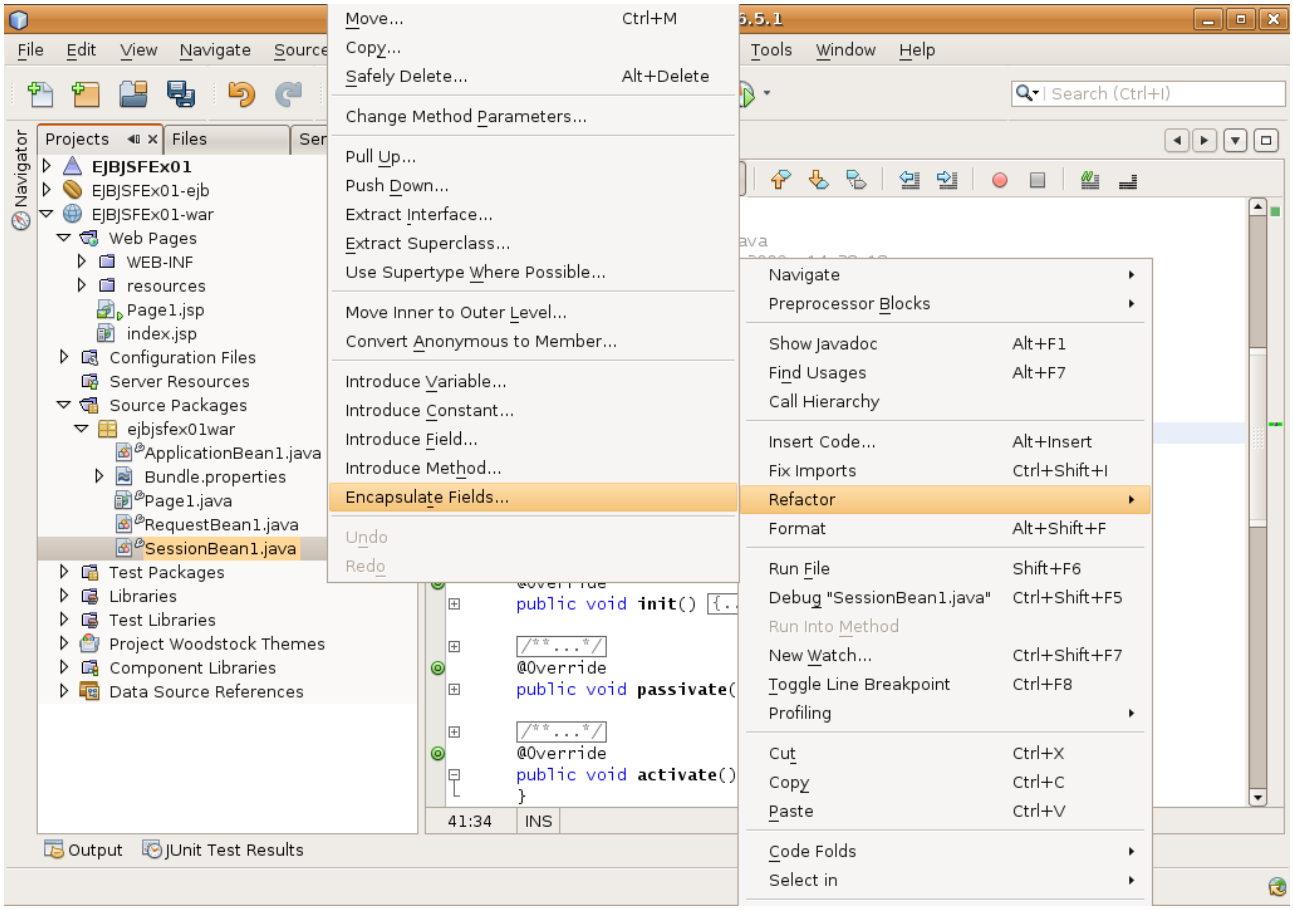
```
private FacadeWorker privateFacadeWorker;

    /**
     * @return the privateFacadeWorker
     */
    public FacadeWorker getPrivateFacadeWorker() {
        if (privateFacadeWorker == null) {
            privateFacadeWorker = new FacadeWorker();
        }
        return privateFacadeWorker;
    }

    /**
     * @param privateFacadeWorker the privateFacadeWorker to set
     */
    public void setPrivateFacadeWorker(FacadeWorker privateFacadeWorker) {
        this.privateFacadeWorker = privateFacadeWorker;
```

}

- fix neccesary import by pressing ctrl+shift+i
- now our FacadeWorker is ready to provide us HotelFacade

Aim of our application is to create table on Page1.jsp. Data binding for table can be provided by creating of field containing requested objects – in our case Hotel[] + get/set properties.

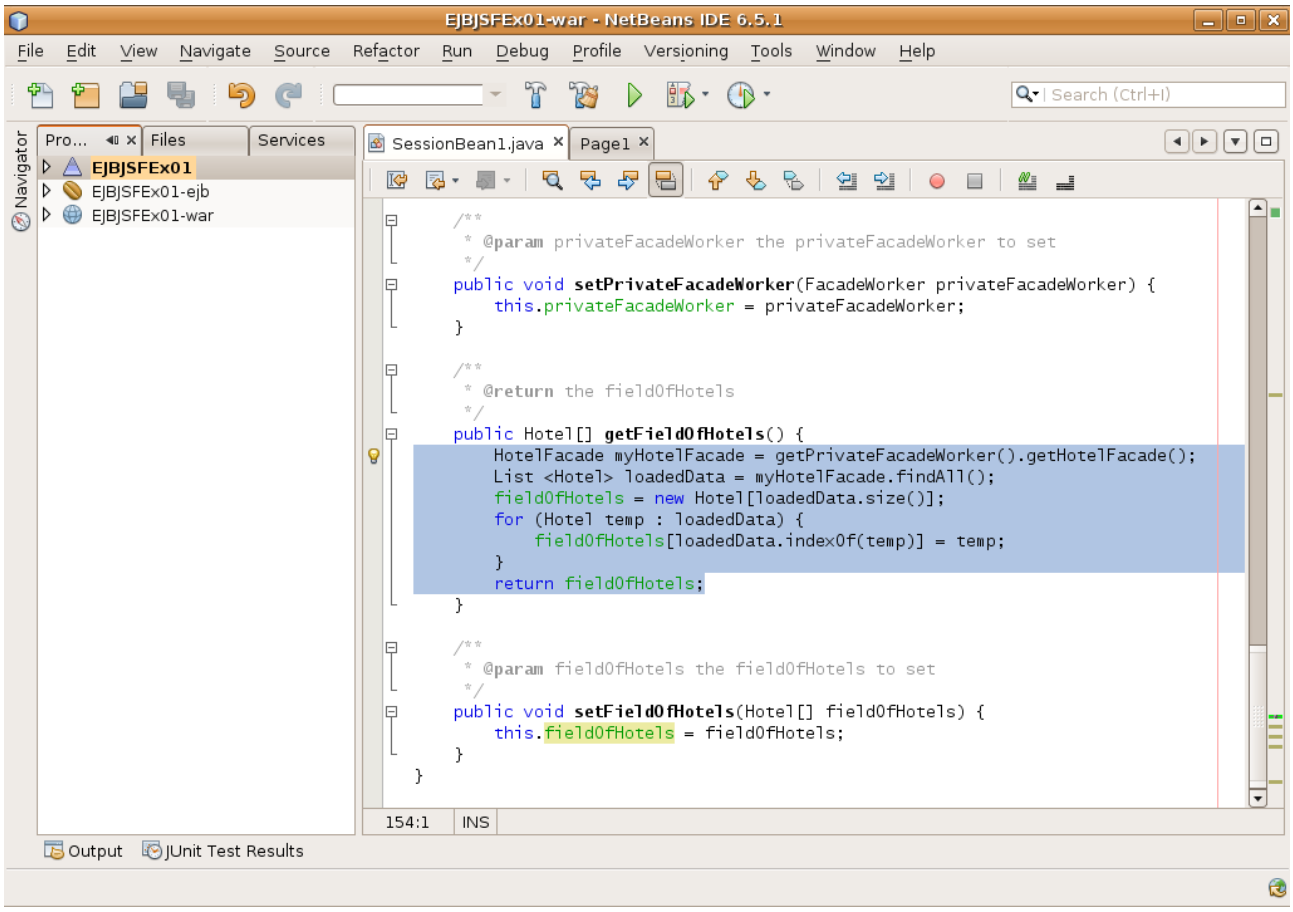
- Find in project file 'SessionBean1'
- add private property Hotel[] fieldOfHotels;
- fix neccesary import by pressing ctrl+shift+i
- select property fieldOfHotels in code, right click
- click refactor, then encapsulate fields



- click on 'refactor' button. This action will generate get/set for selected property
- modify get methodof fieldOfHotel property

```
HotelFacade myHotelFacade = getPrivateFacadeWorker().getHotelFacade();
List <Hotel> loadedData = myHotelFacade.findAll();
fieldOfHotels = new Hotel[loadedData.size()];
for (Hotel temp : loadedData) {
```

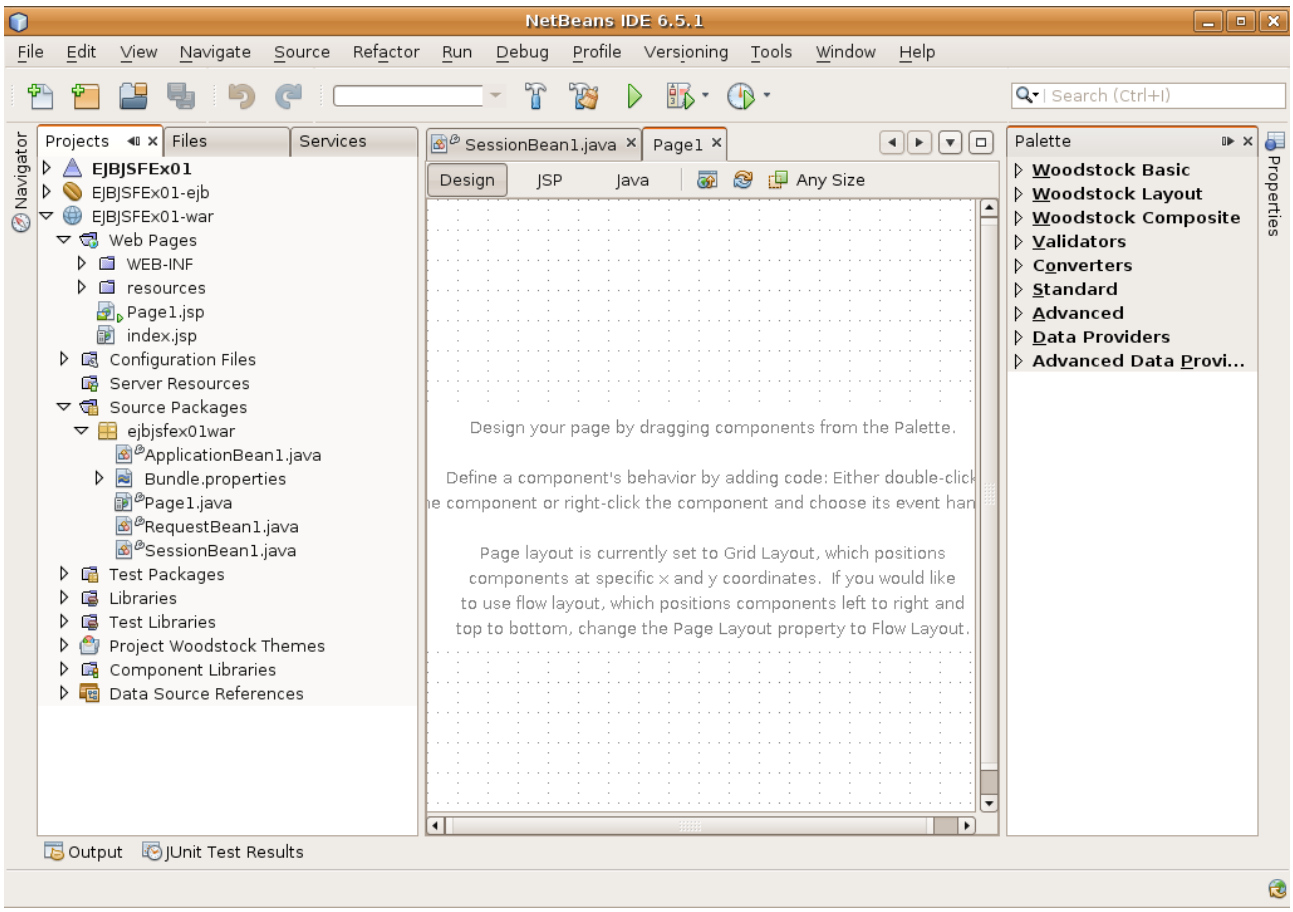
```
        fieldOfHotels[loadedData.indexOf(temp)] = temp;
    }
    return fieldOfHotels;
```



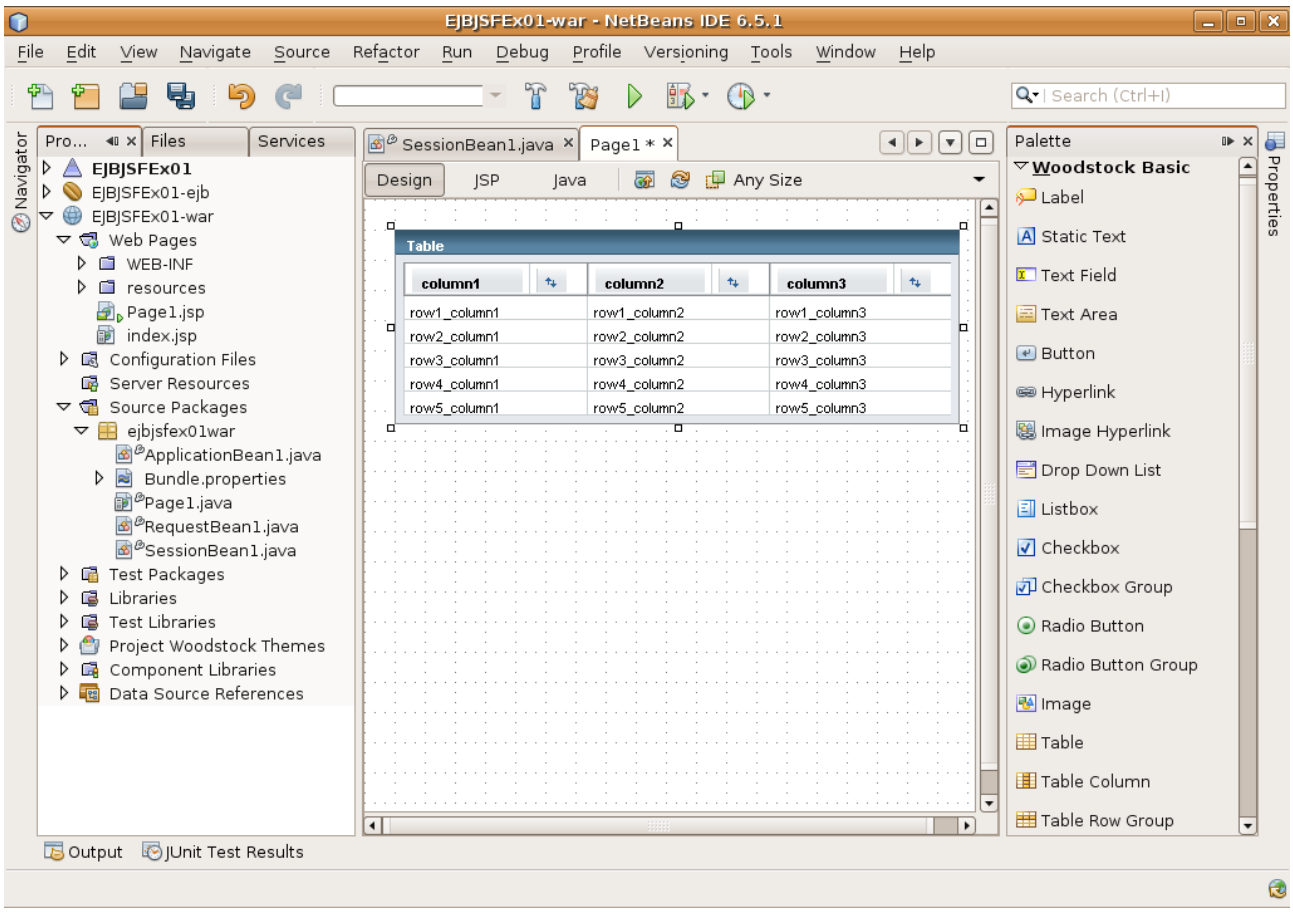
now, our application has ready all data-source related steps to use all – power of netbeans visual jsf editor.

Create table, bind data to created data source

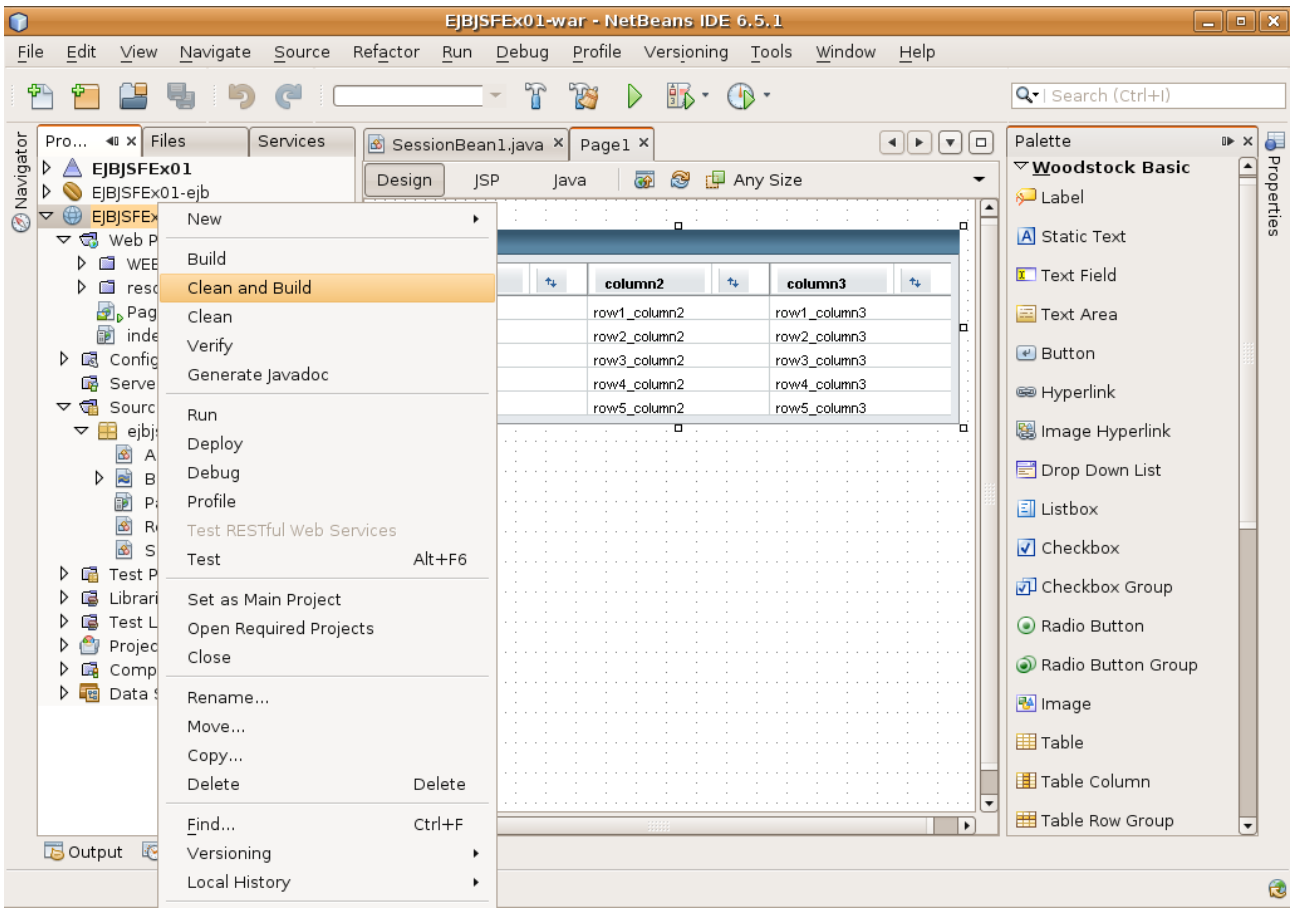
- click on Page1.jsp



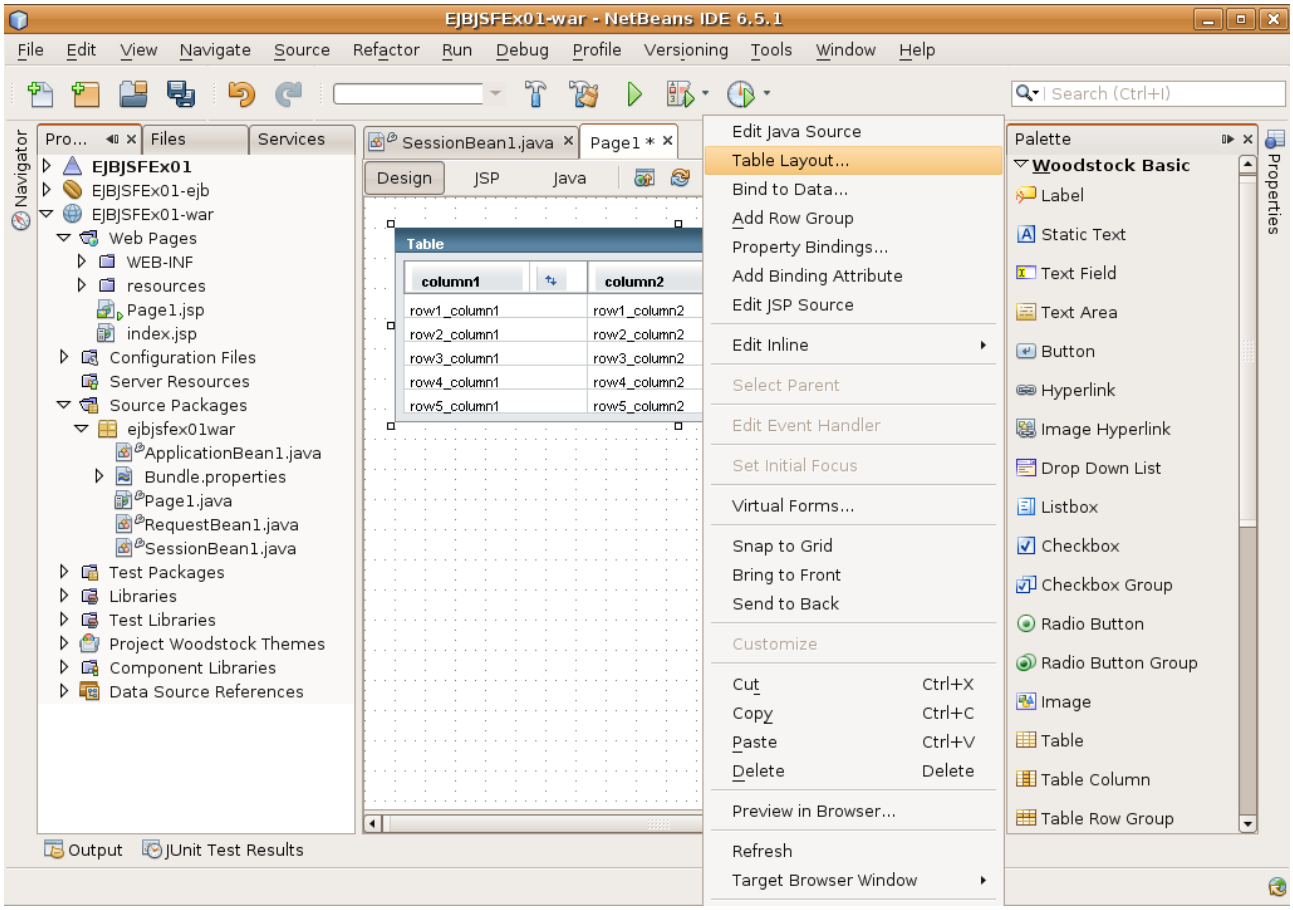
- in right side of screen, find in pallete node 'woodstock basic' and expand it
- drag component 'table' to the middle part od netbeans ide



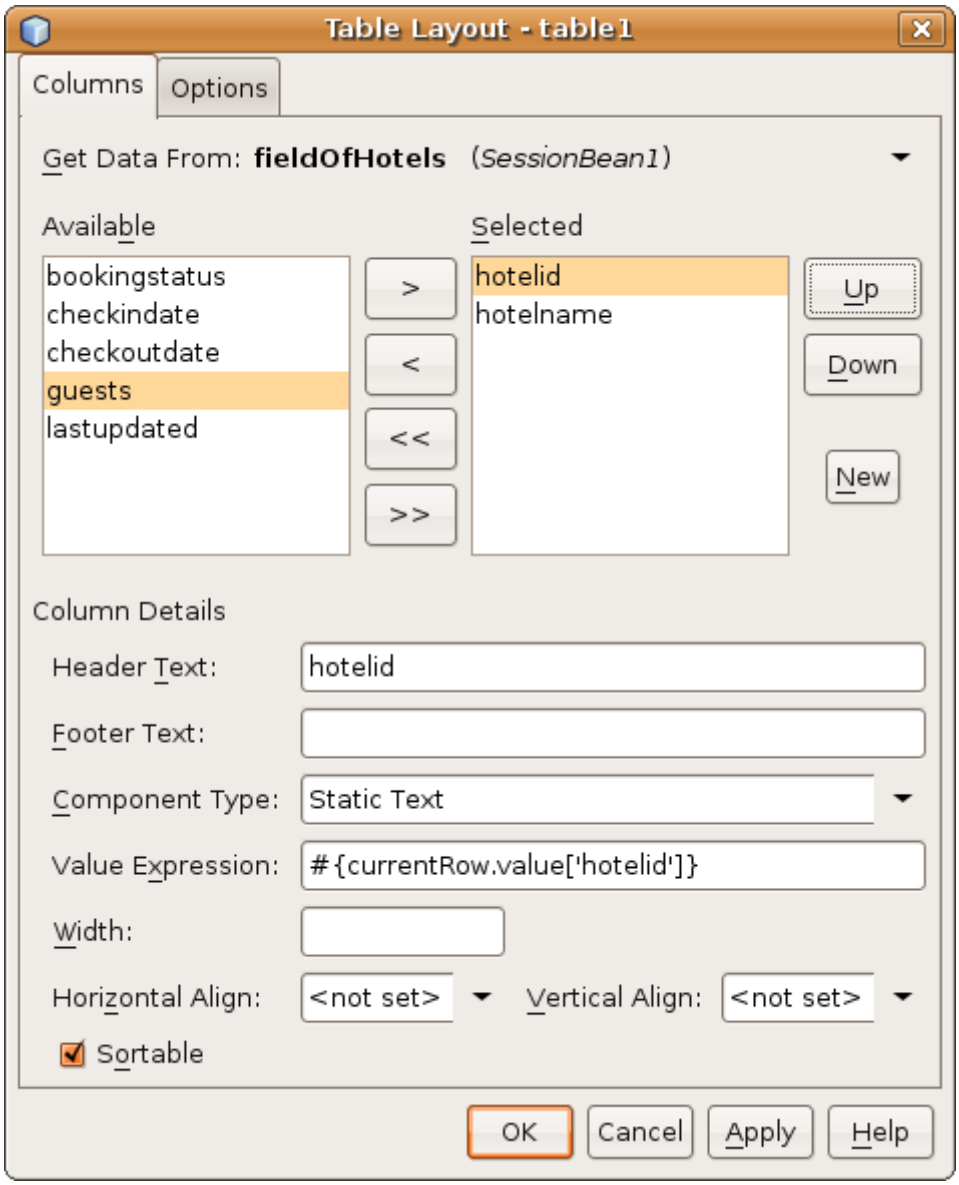
- your screen should look like screen above
- now we need to clean-build web project, that netbeans ide recognize our data-source in next steps



- right click on table, select 'table layout'



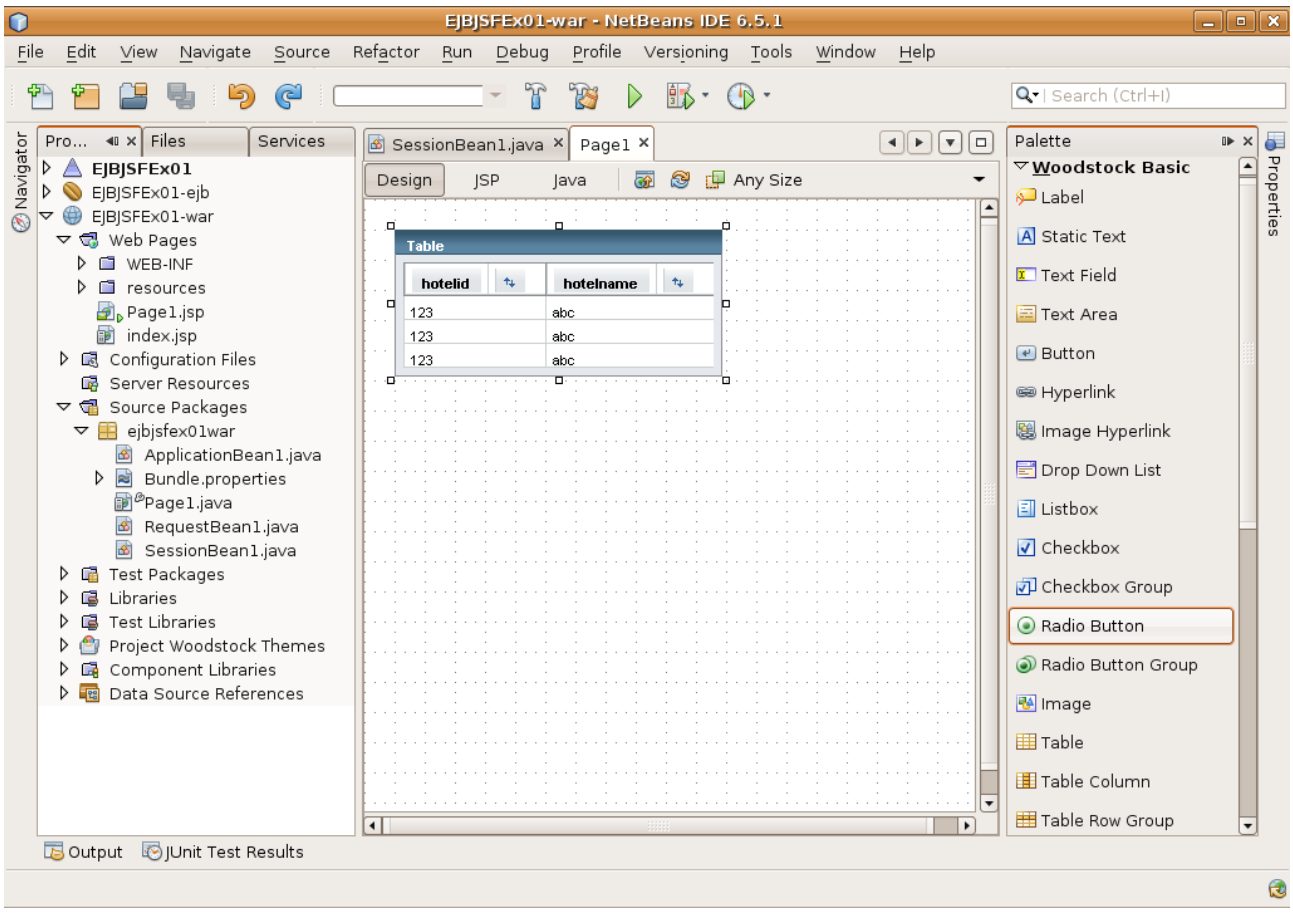
- set property get data from to our fieldOfHotels
- select collums called hotelid and hotelname



- modi fied

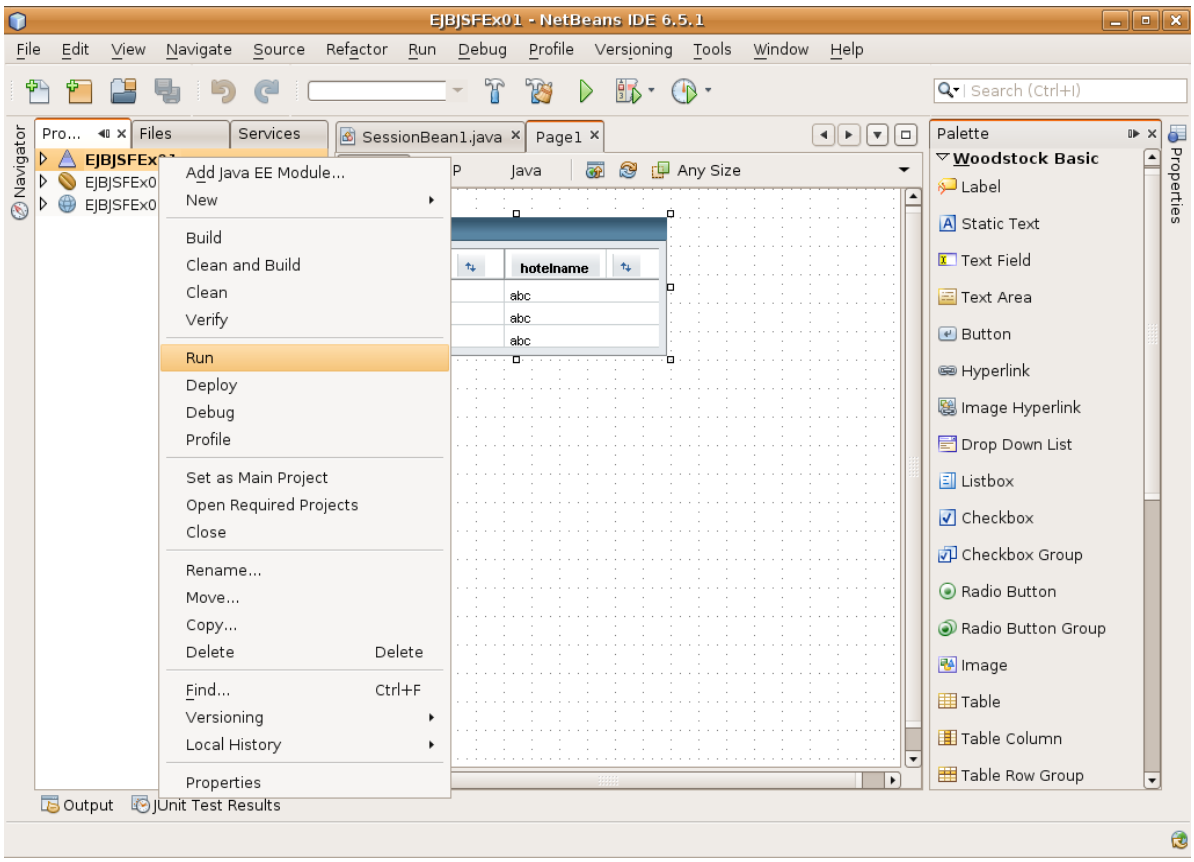
property screen should look like one above

- click ok
- your table should now look like on picture below

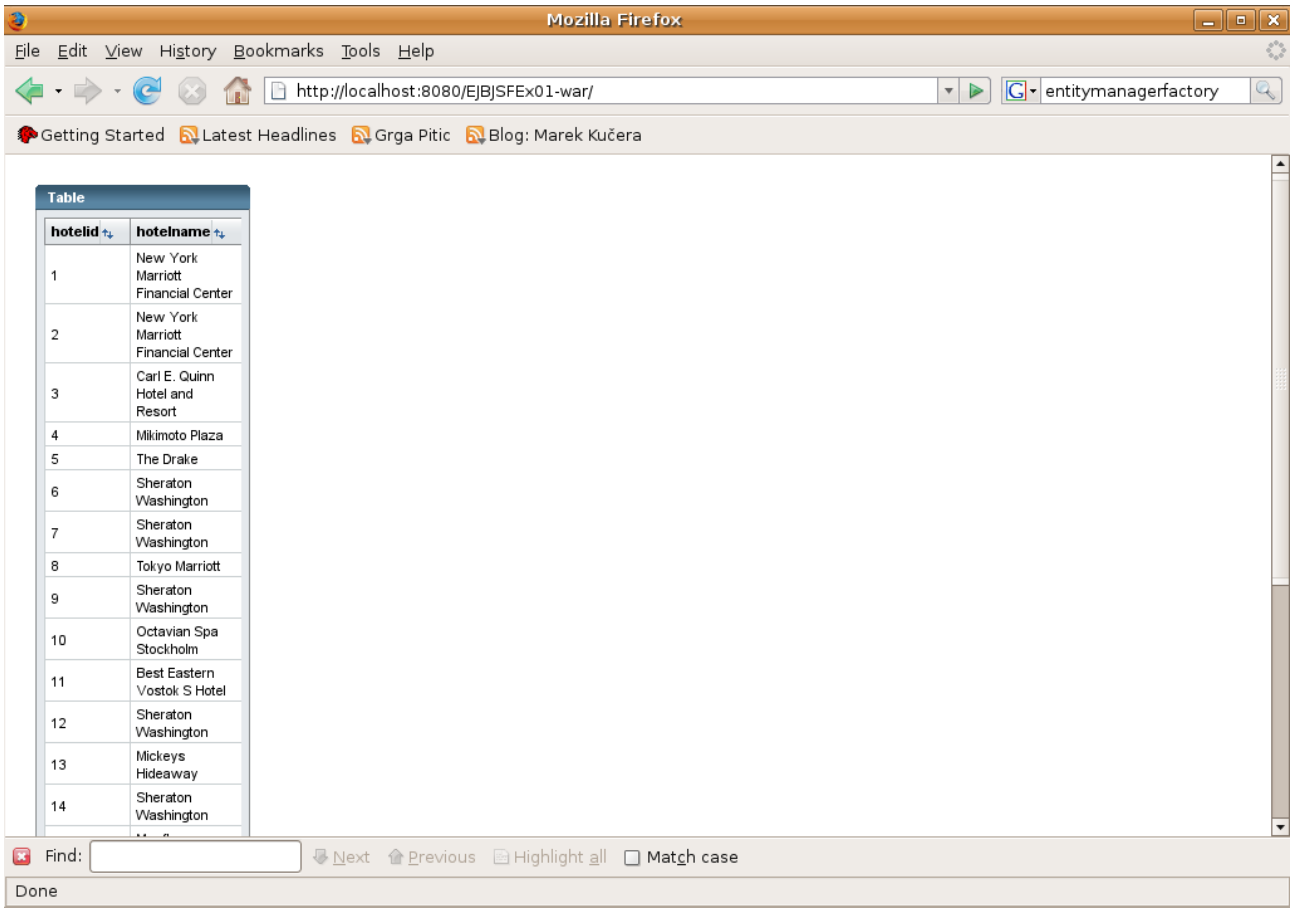


Test Application

- Save all your work by pressing ctrl+shif+s
- right-click on main enterprise application 'EJBJSFEx01', press run


EJBJSFEx01 - NetBeans IDE 6.5.1
File
Edit
View
Navigate
Source
Refactor
Run
Debug
Profile
Versioning
Tools
Window
Help
Search (Ctrl+I)
Navigator
Pro...
Files
Services
SessionBean1.java
Page1
Java
Any Size
Add Java EE Module...
New
Build
Clean and Build
Clean
Verify
Run
Deploy
Debug
Profile
Set as Main Project
Open Required Projects
Close
Rename...
Move...
Copy...
Delete
Find...
Ctrl+F
Versioning
Local History
Properties
hotelname
abc
abc
abc
Palette
Woodstock Basic
Label
Static Text
Text Field
Text Area
Button
Hyperlink
Image Hyperlink
Drop Down List
Listbox
Checkbox
Checkbox Group
Radio Button
Radio Button Group
Image
Table
Table Column
Table Row Group
Properties
Output
JUnit Test Results

after successfull deploy, your application will be accessible



Congratulations, you have just create real J2EE application !

For further questions, please contact autor at elorman@dezadata.cz