Entity EJB with EJB 2 on a database view

This tutorial explain how you create an Entity EJB which shows a database view. The advantage is that you can achieve a very high performance as you can optimize your view query in SQL.

General

Author:
Sascha Wolski
Sebastian Hennebrueder
http://www.laliluna.de/tutorials.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
April, 6 2005

Software:
Eclipse 3.x
MyEclipse 3.8.x or xDoclet

Downloads
PDF: http://www.laliluna.de/download/ejb-on-database-view-en.pdf
Sources: http://www.laliluna.de/download/ejb-on-database-views-source.zip

What are database views

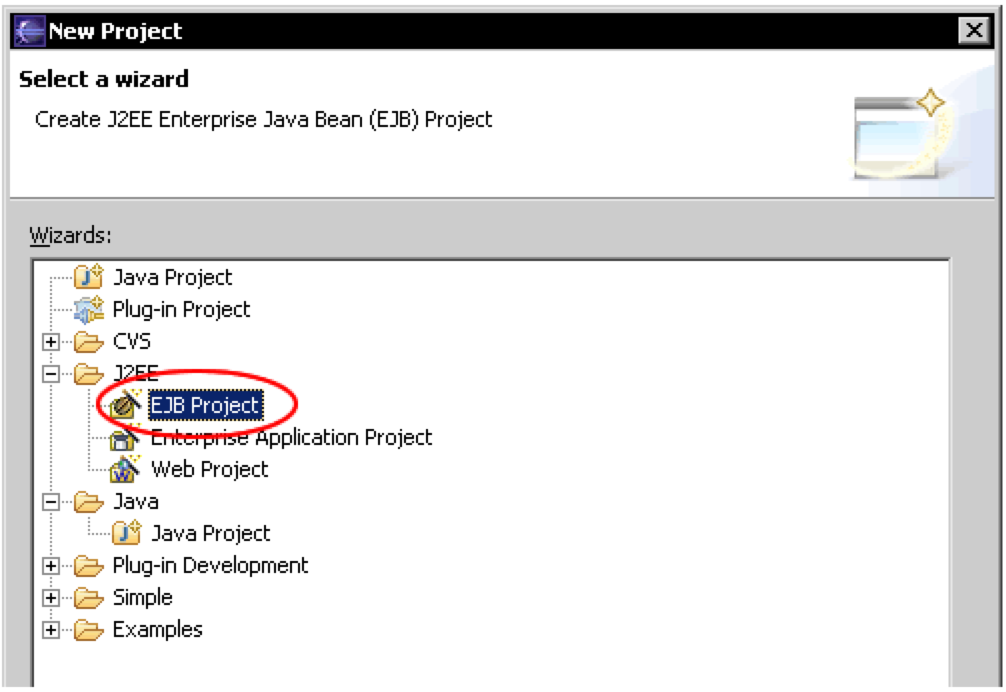
Database views are saved queries (views) in the database of object tables. They are write protected and can access like a normal database table. You can use a database view to provide selections of data, that can not be modified.

The advantage is that you can achieve a very high performance as you can optimize your view query in SQL.

(There are some advanced databases where you can even update a view or where they are not only queries but real database entries.)

Create the EJB project

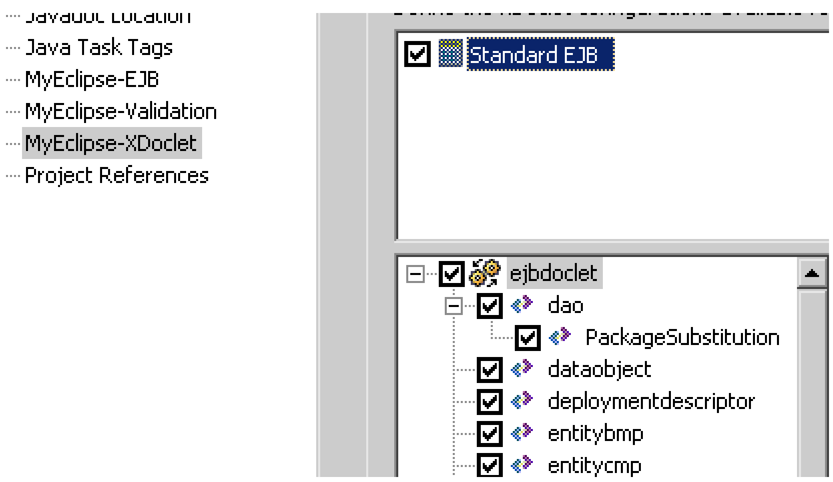
Let's start. Create a new EJB project and name it *DatabaseViewEjb*.



Configure xDoclet

Right click on the project and choose *Properties* (Alt + Enter).

Choose *MyEclipse-XDoclet* and click on *Standard EJB*.



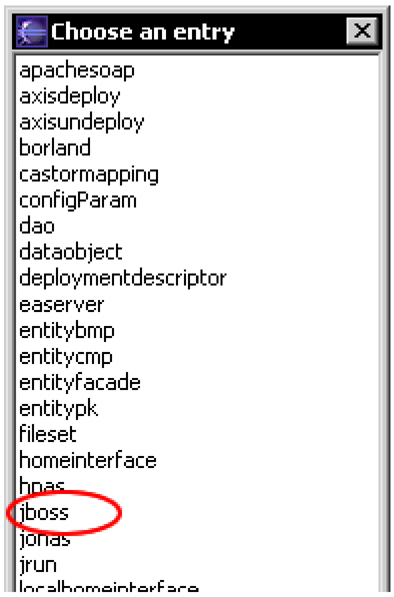
Java Task Tags
MyEclipse-EJB
MyEclipse-Validation
MyEclipse-XDoclet
Project References

☑ Standard EJB

☑ ejbdoclet
  ☑ dao
    ☑ PackageSubstitution
  ☑ dataobject
  ☑ deploymentdescriptor
  ☑ entitybmp
  ☑ entitycmp

In the window below right click and choose *Add*.

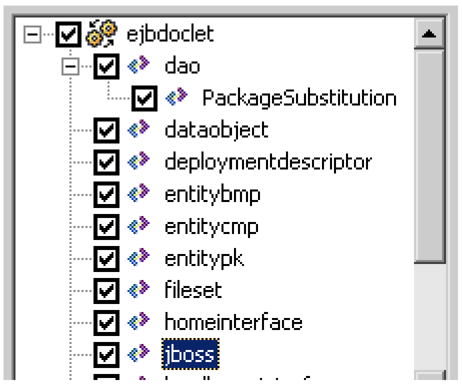
Choose *jboss* from the list.



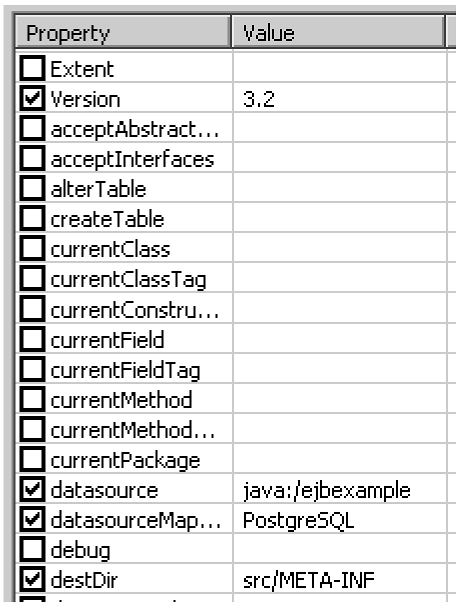
Choose an entry

apachesoap
axisdeploy
axisundeploy
borland
castormapping
configParam
dao
dataobject
deploymentdescriptor
easerver
entitybmp
entitycmp
entityfacade
entitypk
fileset
homeinterface
hpas
jboss
jonas
jrun

Select *jboss* on the list and add the *xDoclet* settings.



☑ ejbdoclet
  ☑ dao
    ☑ PackageSubstitution
  ☑ dataobject
  ☑ deploymentdescriptor
  ☑ entitybmp
  ☑ entitycmp
  ☑ entitypk
  ☑ fileset
  ☑ homeinterface
  ☑ jboss



Property	Value
☐ Extent	
☑ Version	3.2
☐ acceptAbstract…	
☐ acceptInterfaces	
☐ alterTable	
☐ createTable	
☐ currentClass	
☐ currentClassTag	
☐ currentConstru…	
☐ currentField	
☐ currentFieldTag	
☐ currentMethod	
☐ currentMethod…	
☐ currentPackage	
☑ datasource	java:/ejbexample
☑ datasourceMap…	PostgreSQL
☐ debug	
☑ destDir	src/META-INF

Close the property window of the project.

Notice: dataSourceMapping and datasource:

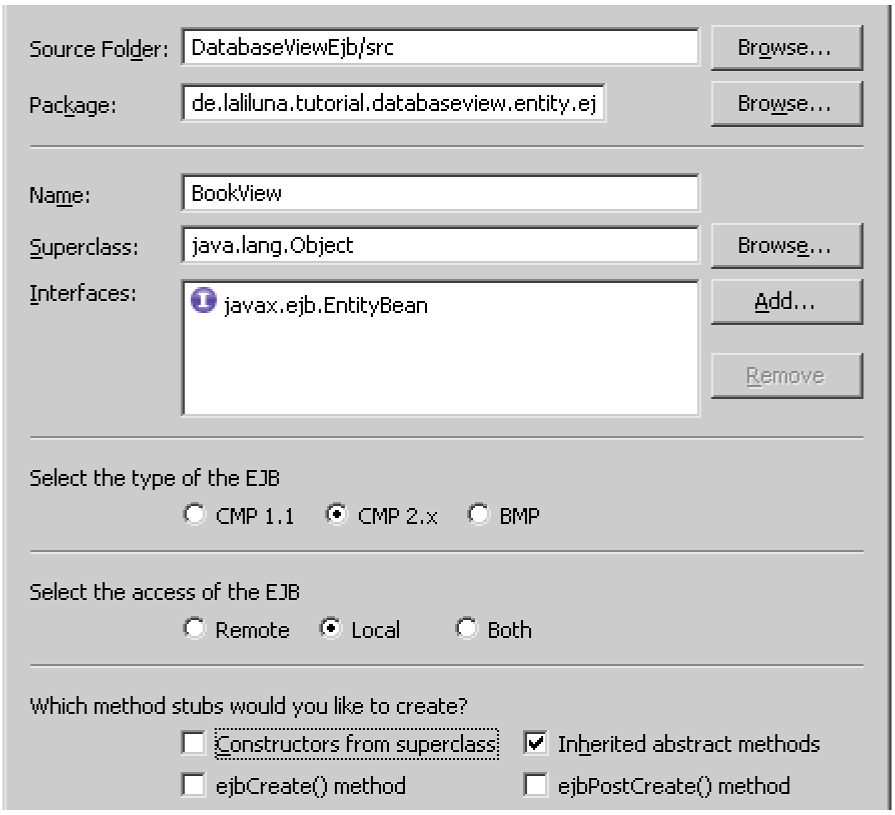
Have a look in the basic EJB tutorials http://www.laliluna.de/simple-xdoclet-ejb-tutorial.html to find more information on how to change the configuration for other databases.

Create the entity bean

First create a new package *de.laliluna.tutorial.databaseview.entity.ejb*.

Create a new entity bean *BookView.* Right click on the project and choose *New > Entity Bean*.

You do not need to create the *ejbCreate()* and *ejbPostCrearte()* method on an entity bean which refers to a view, because the view is write protected and the methods are only needed to create new entries.



Source Folder: DatabaseViewEjb/src

Package: de.laliluna.tutorial.databaseview.entity.ej

Name: BookView

Superclass: java.lang.Object

Interfaces: javax.ejb.EntityBean

Select the type of the EJB
CMP 1.1 [✓ CMP 2.x] BMP

Select the access of the EJB
Remote [✓ Local] Both

Which method stubs would you like to create?
☐ Constructors from superclass ☑ Inherited abstract methods
☐ ejbCreate() method ☐ ejbPostCreate() method

The entity bean will refers to a view *vbook* in our *ejbexample* database, we will create later. The view contains two columns, *id* and *title*.

Now lets look at the *xDoclet* comments. We have to add some settings for the entity bean class.

The following source code shows the xDoclet class comments.

Define a value object *BookView*.

Set the *jboss.persistence* properties *create-table* and *remove-table* to *false*, because jboss can`t create or remove a view. The view is write protected, so set the *jboss.persistence* property *read-only* to *true*.

Define a finder *findAll()* which returns all entries of the view.

```
/**
 * @author laliluna.de
 *
 * @ejb.bean name="BookView"
 *           display-name="Name for BookView"
 *           description="Description for BookView"
 *           jndi-name="ejb/BookView"
 *           type="CMP"
 *           cmp-version="2.x"
 *           view-type="local"
 *           primkey-field = "id"
 *
 * @ejb.util generate="physical"
 * @ejb.persistence table-name = "vbook"
 * @ejb.value-object match = "*" name="BookView"
 *
 * @jboss.persistence  create-table = "false"
 *                     remove-table = "false"
 *                     read-only = "true"
 *
 * @ejb.finder description = "Find all"
 *              signature = "java.util.Collection findAll()"
 *              query = "select object(c) from BookView as c"
 *
 *
 */
```

Now create the getter and setter methods for the two columns, *fid* and *ftitle*, of the view.

```
    /**
     * @ejb.interface-method view-type = "local"
     * @ejb.persistence column-name = "fid"
     *
     * @ejb.pk-field
     *
     *
     * @return
     */
    public abstract Integer getId();

    /**
     * @ejb.interface-method view-type = "local"
     * @param id
     */
    public abstract void setId(Integer id);


    /**
```

```
     * @ejb.interface-method view-type = "local"
     * @ejb.persistence column-name = "ftitle"
     *
     * @return
     */
    public abstract String getTitle();

    /**
     * @ejb.interface-method view-type = "local"
     * @param title
     */
    public abstract void setTitle(String title);
```

Note:
Its recommend to run *xDoclet* first time to generate the interface classes. Right click on the project and choose *MyEclipse > Run xDoclet*.

Lets provide a getter and setter method for the generated value object.

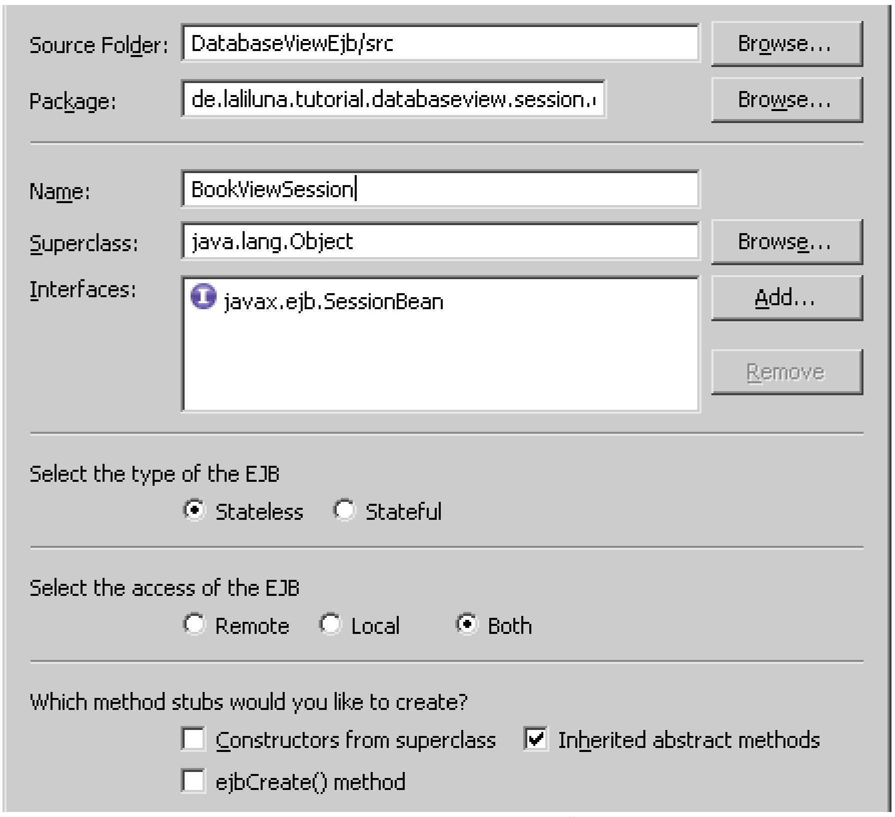
```
    /**
     * @ejb.interface-method view-type = "local"
     * @return
     */
    public abstract BookViewValue getBookViewValue();

    /**
     * @ejb.interface-method view-type = "local"
     * @param bookViewValue
     */
    public abstract void setBookViewValue(BookViewValue bookViewValue);
```

Thats all, the entity bean for a view is finished.

Create the session bean

Create a new package *de.laliluna.tutorial.databaseview.session.ejb* and create a new Session Bean *BookViewSession*.


Source Folder: DatabaseViewEjb/src
Package: de.laliluna.tutorial.databaseview.session.
Name: BookViewSession
Superclass: java.lang.Object
Interfaces: javax.ejb.SessionBean

Select the type of the EJB
[✓ Stateless] Stateful

Select the access of the EJB
Remote Local [✓ Both]

Which method stubs would you like to create?
Constructors from superclass [✓ Inherited abstract methods]
ejbCreate() method

Open the session bean class and provide a method *getAll()*, which returns a collection of *BookViewValue* objects.

The following source code shows the session bean method *getAll()*:

```
    /**
     * Return a collection of BookViewValue objects
     *
     * @ejb.interface-method view-type = "both"
     */
    public Collection getAll() throws EJBException {

        Collection collection = null;

        try {
            Context context = new InitialContext();

            // get the local home
            BookViewLocalHome localHome = (BookViewLocalHome) context
                    .lookup(BookViewLocalHome.JNDI_NAME);

            // get all entries of the local home
            Collection localCollection = localHome.findAll();

            // fill the collection that will be returned
            collection = new ArrayList();
            for (Iterator iter = localCollection.iterator(); iter.hasNext();) {
                BookViewLocal element = (BookViewLocal) iter.next();
                collection.add(element.getBookViewValue());
            }

        } catch (FinderException e) {
            e.printStackTrace();
        } catch (NamingException e) {
            e.printStackTrace();
        }

        return collection;
    }
```

That's all for the session bean class.

Note:
Run *xDoclet* to generate the session bean interface classed. Right click on the project and choose *MyEclipse > Run xDoclet*.

Provide the database view

Create a new database *ejbexample* with your favorite Postgre manager.

Provide a table *tbook* with two columns *fid* of type serial and *ftitle* of type text.

The postgre-sql query for creating the table looks like the following:

```
CREATE TABLE tbook
(
  fid serial NOT NULL,
  ftitle text
)
WITH OIDS;
```

Insert some dummy data for testing.

Create a view *vbook* for this table.

The postgre-sql query for the view looks like the following:

```
CREATE OR REPLACE VIEW vbook AS
SELECT tbook.fid, tbook.ftitle
FROM tbook;
```

Datasource mapping file

Create a new datasource mapping file named *ejbexmaple-ds.xml* and place it in the folder *../jboss-root/server/default/deploy/* to have access to the database.

The content of the file looks like the following:

```
<datasources>
<local-tx-datasource>
<jndi-name>ejbexample</jndi-name>
<connection-url>jdbc:postgresql://localhost:5432/ejbexample</connection-url>
<driver-class>org.postgresql.Driver</driver-class>
<user-name>postgres</user-name>
<password>pgsql</password>
</local-tx-datasource>
</datasources>
```

Note:
Deploy the EJB project to the Jboss server.

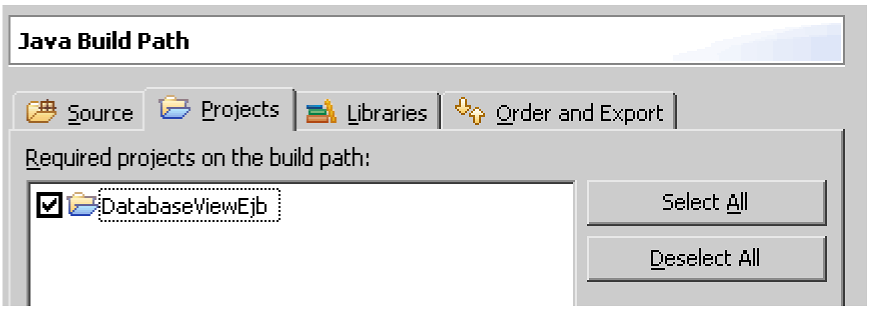
Create the test client

Create a new Java project *DatabaseViewClient* to test the EJB project.

Add a source folder *src*, right click on the project and choose *New > Source Folder*.

Provide a package named *de.laliluna.tutorial.databaseview*.

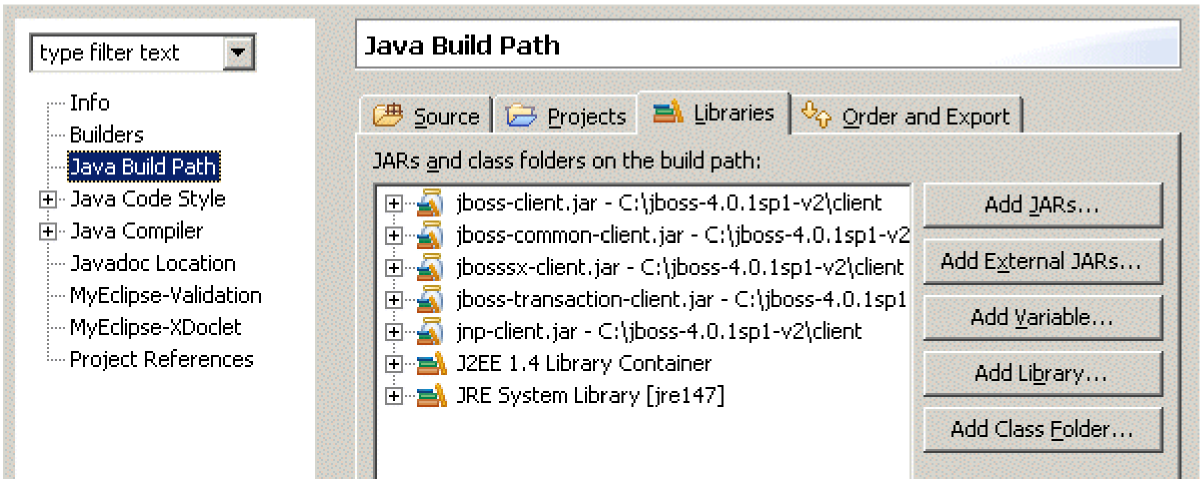
Add the EJB project on *Projects* to access to the EJB classes.
Right click on the project and choose *Properties > Java Build Path*.


Java Build Path

Required projects on the build path:

☑ DatabaseViewEjb

You have to add the J2EE Library and the following JBoss libraries to use a normal Java project for testing an EJB project. If you like you can use the jboss-all-client.jar instead of the single libraries.



Java Build Path

JARs and class folders on the build path:

- jboss-client.jar - C:\jboss-4.0.1sp1-v2\client
- jboss-common-client.jar - C:\jboss-4.0.1sp1-v2
- jbosssx-client.jar - C:\jboss-4.0.1sp1-v2\client
- jboss-transaction-client.jar - C:\jboss-4.0.1sp1
- jnp-client.jar - C:\jboss-4.0.1sp1-v2\client
- J2EE 1.4 Library Container
- JRE System Library [jre147]

The test class

Create a new Java class *TestView* in the package *de.laliluna.tutorial.databaseview*.

We have to set some properties to lookup the EJBs in the JNDI context of jboss. You can do this within the constructor.

Create a method *testEJB()* where you put the code for testing the EJB.

In the *main(..)* method you call the *testEJB()* method.

The following source code shows the class *TestView*:

```
public class TestView {

    Properties properties;

    public TestView() {
        properties = new Properties();
        properties.put("java.naming.factory.initial",
                       "org.jnp.interfaces.NamingContextFactory");
        properties.put("java.naming.factory.url.pkgs",
                       "org.jboss.naming:org.jnp.interfaces");
        properties.put("java.naming.provider.url", "jnp://localhost:1099");
        properties.put("jnp.disableDiscovery", "true");
    }

    public static void main(String[] args) {
        TestView testView = new TestView();
        // call the testEJB method
        testView.testEJB();
    }
    public void testEJB(){

        try {
            InitialContext context = new InitialContext(properties);

            // get the session home interface
            BookViewSessionHome sessionHome = (BookViewSessionHome) context
                    .lookup(BookViewSessionHome.JNDI_NAME);

            // create a session object
            BookViewSession session = sessionHome.create();

            // output data
            Collection collection = session.getAll();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                BookViewValue element = (BookViewValue) iter.next();
                System.out.print(element.getId() + ", ");
                System.out.print(element.getTitle() + ", ");
            }

        } catch (CreateException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }
}
```

That's all for the testing class. Now you can now run the class as java Application. Right click on the project and choose *Run* > *Java Application.*