Creating Finder for EJB 2 with xDoclet

Generals

Author:
Sascha Wolski
http://www.laliluna.de/tutorials.html – Tutorials für Struts, JSF, EJB, xdoclet und eclipse.
Date:
December, 06th 2004
Software:
EJB 2
Eclipse 3.0
MyEclipse Plugin 3.8
Jboss 3.2.5
Postgre SQL Server 8.0 beta 4
Downloads:
PDF: http://www.laliluna.de/download/ejb-finder-tutorial-en.pdf
Sources: http://www.laliluna.de/download/ejb-finder-sources.zip

Introduction

This tutorial explains step by step how to create finders with xDoclet. Within a sample application different finders are provided and used in a business logic. It is pointed out, how to use a finder in relationships between two entity beans and which operators are supported by the query language.

Requirements

We are not repeating all the basics here. Have a look at the basic EJB tutorials before you start this tutorial.

What are finders ?

A finder is similar to a normal SQL SELECT instruction. It is used to read individuell data from the database. A finder is always a method, which is placed in the home interfaces of an entity bean. Only the method header is defined within the home interfaces. The method itself is created by the application server by using a query mapping.

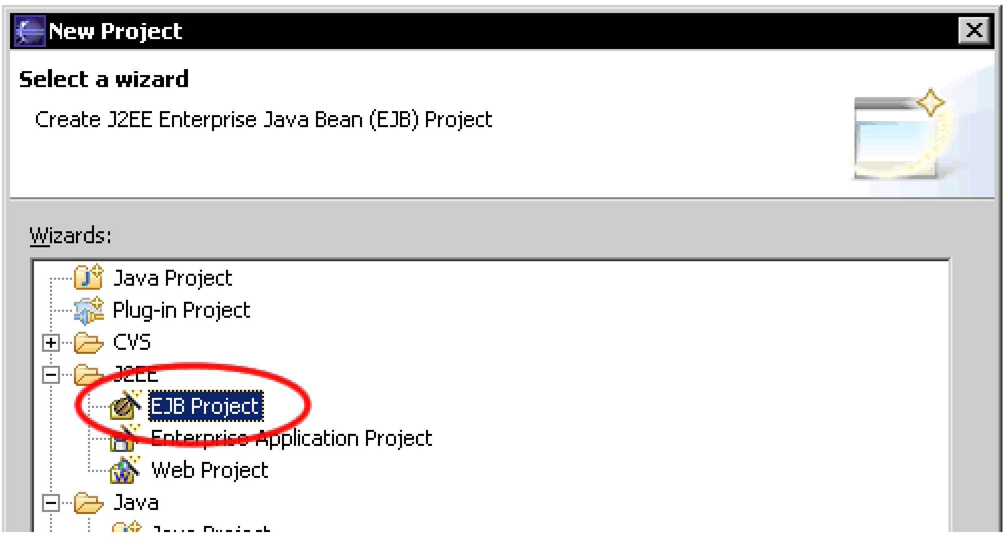
```
public java.util.Collection findAll()
      throws javax.ejb.FinderException,java.rmi.RemoteException;
```

For each finder a query mapping entry is defined in the ejb-jar.xml configuration file. The following example shows a mapping entry.

```
<query>
      <description><![CDATA[Find all customer]]></description>
      <query-method>
            <method-name>findAll</method-name>
                  <method-params>
            </method-params>
      </query-method>
      <ejb-ql><![CDATA[select object(c) from Customer as c]]></ejb-ql>
</query>
```

Create an EJB Project

Create a new enterprise java bean project (File > New > J2EE > EJB Project).



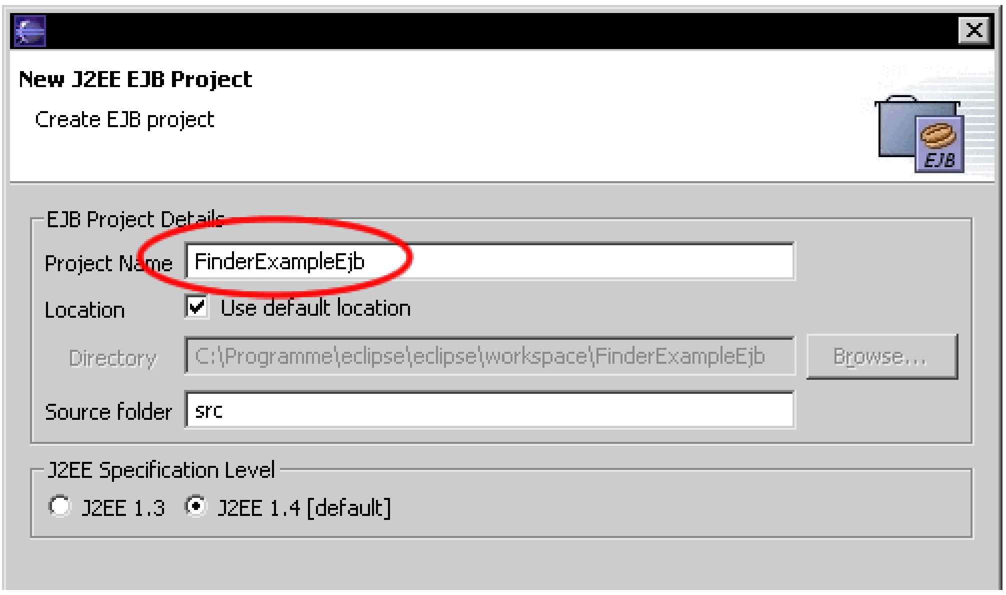
New Project

Select a wizard

Create J2EE Enterprise Java Bean (EJB) Project

Wizards:

Set a nice name for the project. It`s recommend to add Ejb to the end of the project name, so it is evident at the project name, that an EJB project is present.



New J2EE EJB Project

Create EJB project

EJB Project Details

Project Name: FinderExampleEjb

Location: [✓ Use default location]

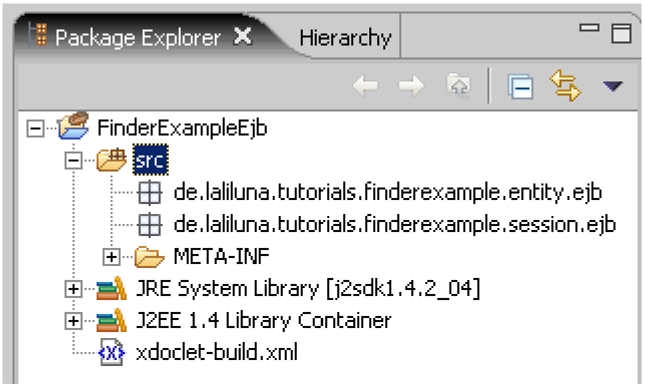
Directory: C:\Programme\eclipse\eclipse\workspace\FinderExampleEjb

Source folder: src

J2EE Specification Level

J2EE 1.3 [✓ J2EE 1.4 [default]]

Add a package for the entity and session beans.
*At the end of the package you have to add **.ejb**, so that xDoclet runs correctly.*



```
FinderExampleEjb
  src
    de.laliluna.tutorials.finderexample.entity.ejb
    de.laliluna.tutorials.finderexample.session.ejb
    META-INF
  JRE System Library [j2sdk1.4.2_04]
  J2EE 1.4 Library Container
  xdoclet-build.xml
```

Create the datasource and datasource file

Create a new postgre database `ejbfinderexample`. Add a datasource file named `ejbfinderexample-ds.xml` in the folder `Jboss/server/deploy`.

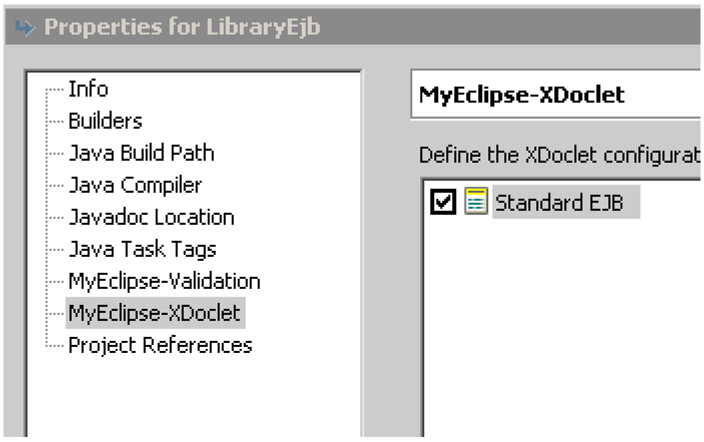
Beispiele für alle möglichen unterstützten Datenbanken kann man im folgenden Verzeichnis finden: ...\jboss-3.2.5\docs\examples\jca

Add the following configuration to the file.

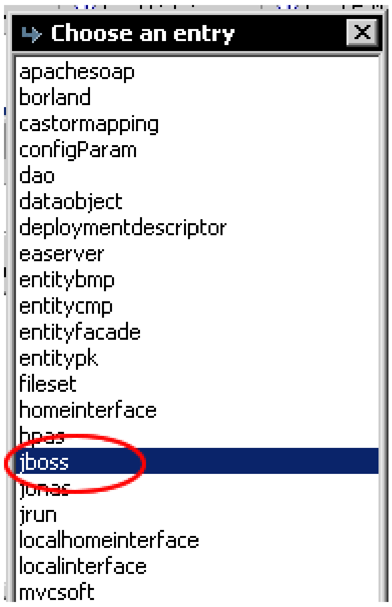
```
<datasources>
<local-tx-datasource>
<jndi-name>ejbfinderexample</jndi-name>
<connection-url>jdbc:postgresql://localhost:5432/ejbfinderexample</connection-
url>
<driver-class>org.postgresql.Driver</driver-class>
<user-name>postgres</user-name>
<password>pgsql</password>
</local-tx-datasource>
</datasources>
```
Be sure that the port, the user and the password are correct for your database.

Configuration of xDoclet

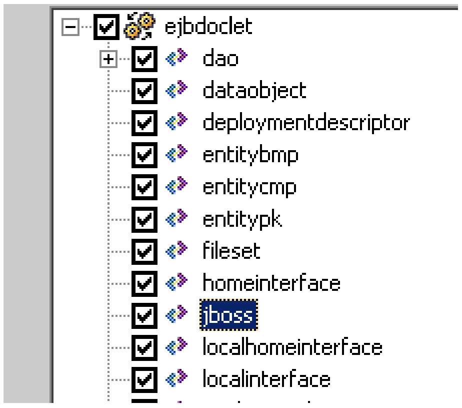
Open the project properties (Strg + Enter). On „MyEclipse- Xdoclet" right click in the upper right window and choose „Add Standard", than „Standard EJB". Now you will find „Standard EJB" in the upper right window.



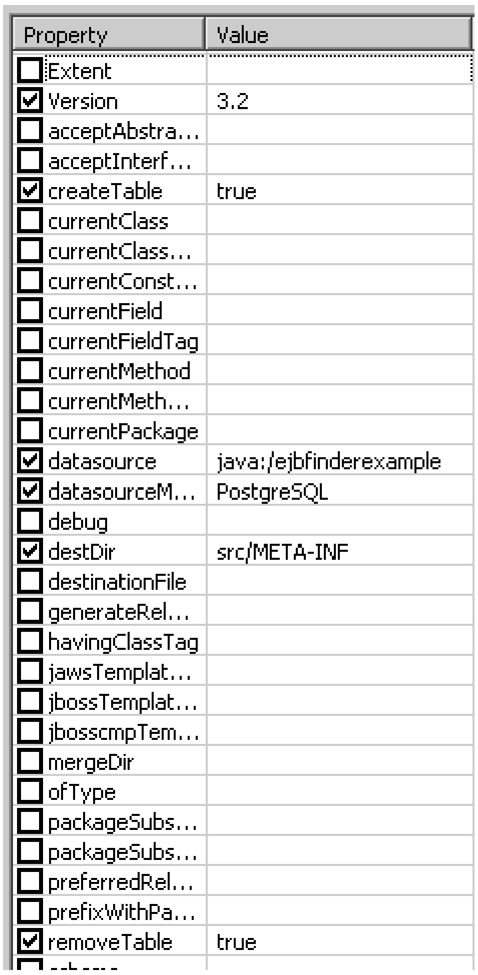
Choose „Standard EJB" and call the menu „Add" (right click) on the window below. Choose „jboss" from the list.



Now you find a menu entry jboss.



Set up xDoclet as shown in the picture below.



	Property	Value
☐	Extent	
☑	Version	3.2
☐	acceptAbstra...	
☐	acceptInterf...	
☑	createTable	true
☐	currentClass	
☐	currentClass...	
☐	currentConst...	
☐	currentField	
☐	currentFieldTag	
☐	currentMethod	
☐	currentMeth...	
☐	currentPackage	
☑	datasource	java:/ejbfinderexample
☑	datasourceM...	PostgreSQL
☐	debug	
☑	destDir	src/META-INF
☐	destinationFile	
☐	generateRel...	
☐	havingClassTag	
☐	jawsTemplat...	
☐	jbossTemplat...	
☐	jbosscmpTem...	
☐	mergeDir	
☐	ofType	
☐	packageSubs...	
☐	packageSubs...	
☐	preferredRel...	
☐	prefixWithPa...	
☑	removeTable	true

That's all we need to do to configure xDoclet.

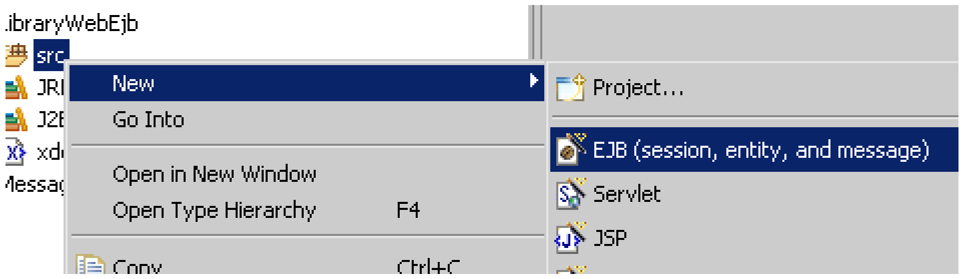
Entity Beans

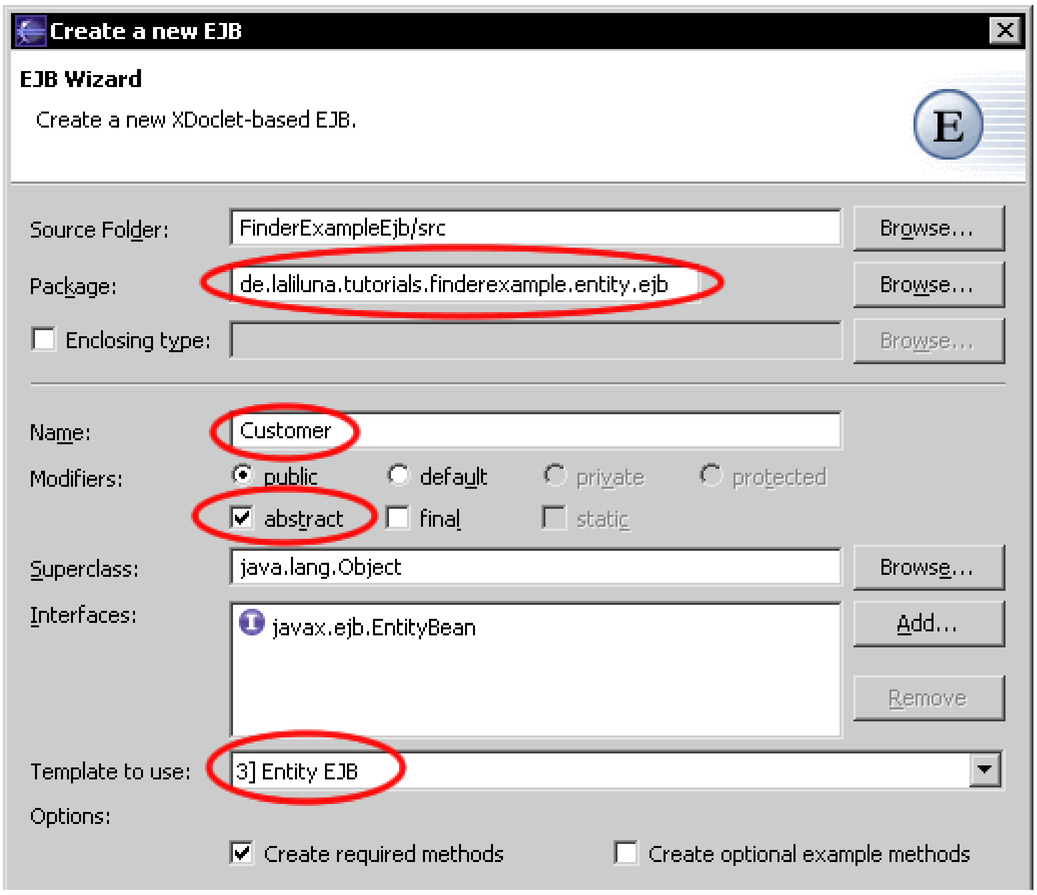
So that special sample applications of finders can be explained, we need two entity beans with a 1:n relation. In the next step we create the two entity beans

- Customer
- Book

Customer entity bean

Richt mouse button on the package `de.laliluna.tutorial.finderexample.entity.ejb` > New > EJB(session...).





Source Folder: FinderExampleEjb/src
Package: de.laliluna.tutorials.finderexample.entity.ejb
[ ] Enclosing type:
Name: Customer
Modifiers: [✓ public] default private protected
[✓ abstract] [ ] final [ ] static
Superclass: java.lang.Object
Interfaces: javax.ejb.EntityBean
Template to use: 3] Entity EJB
Options:
[✓ Create required methods] [ ] Create optional example methods

Open the created class `Customer` and change the xDoclet tags above the class to the following.

```
/**
 * @author laliluna.de
 *
 * @ejb.bean name = "Customer"
 *           type = "CMP"
 *           cmp-version = "2.x"
 *           display-name = "Customer"
 *           description = "Customer EJB"
 *           view-type = "both"
 *           jndi-name = "ejb/CustomerHome"
 *           local-jndi-name = "ejb/CustomerLocalHome"
 *                 primkey-field = "id"
 *
 * @ejb.util generate="physical"
 * @ejb.persistence table-name = "tcustomer"
 * @ejb.value-object match = "*" name="Customer"
 *
 * @jboss.persistence create-table = "true"
 *                    remove-table = "true"
 * @jboss.entity-command name="postgresql-fetch-seq"
 */
public abstract class Customer implements EntityBean {
```

Now add getter- and setter methods for the following fields.

- id (autoincrement key)
- name (name of the customer)
- age (age of the customer)

The methods look like the following.

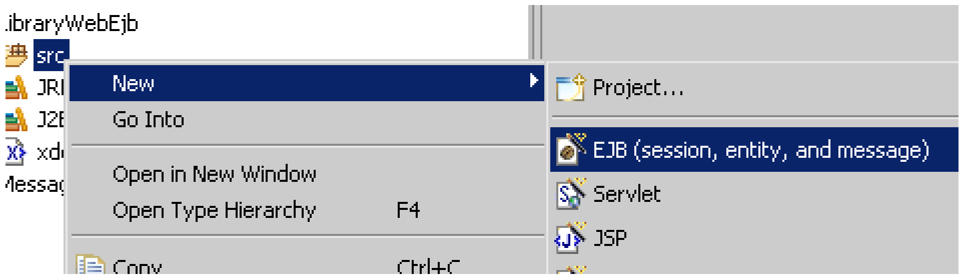
```
/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fid"
 *                  jdbc-type = "INTEGER"
 *                  sql-type = "SERIAL"
 * @ejb.pk-field
 * @jboss.persistence auto-increment = "true"
 * @return
 */
public abstract Integer getId();

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

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fname"
 * @return
 */
public abstract String getName();

/**
 * @ejb.interface-method view-type = "both"
 * @param name
 */
public abstract void setName(String name);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fage"
 * @return
 */
public abstract Integer getAge();

/**
 * @ejb.interface-method view-type = "both"
 * @param age
 */
public abstract void setAge(Integer age);
```

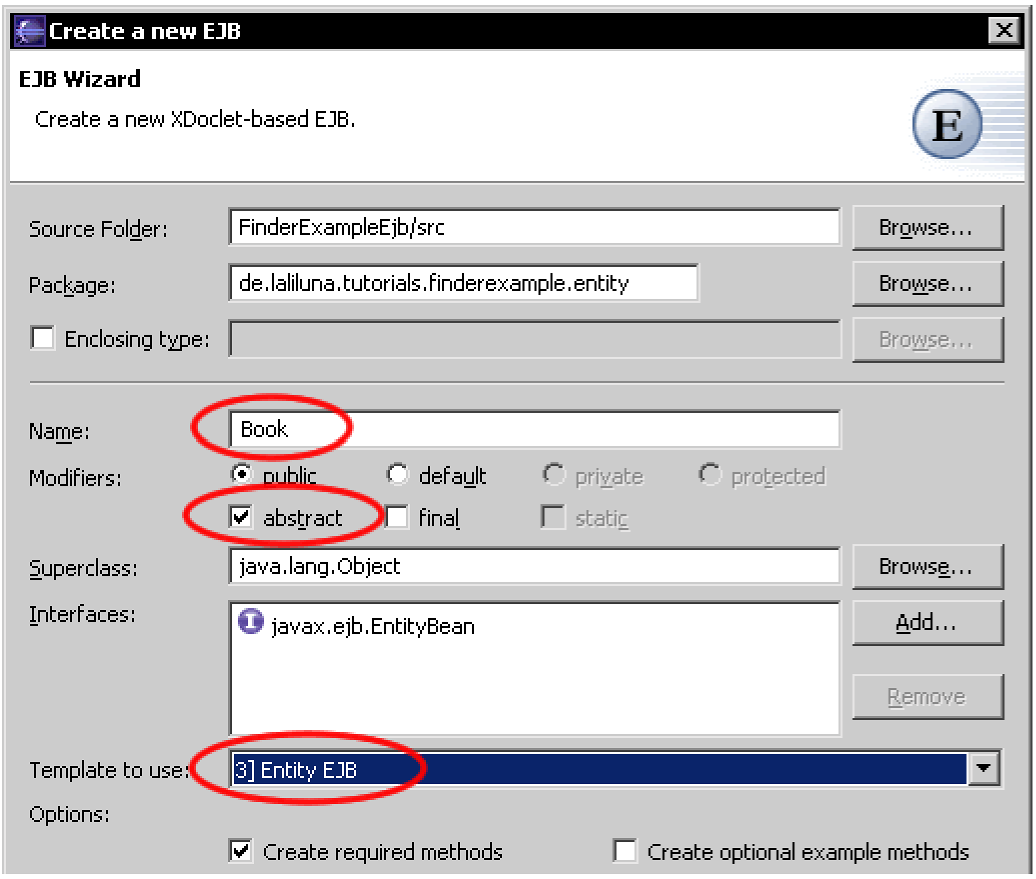
Within the class modify the **ejbCreate()** method. This must always exhibit the return typ of the primary key. (in our case the property id)

```
public Integer ejbCreate() throws CreateException {
      return null;
}
```

Book entity bean

Create entity bean named book. Right mouse button on the package `de.laliluna.tutorial.finderexample.entity.ejb` > New > EJB(session...).




Create a new EJB

EJB Wizard

Create a new XDoclet-based EJB.

Source Folder: FinderExampleEjb/src

Package: de.laliluna.tutorials.finderexample.entity

Enclosing type:

Name: Book

Modifiers: [✓ public] default private protected

[✓ abstract] final static

Superclass: java.lang.Object

Interfaces: javax.ejb.EntityBean

Template to use: 3] Entity EJB

Options: [✓ Create required methods] Create optional example methods

First modify the xDoclet tags, found above the class.

```
/**
 * @author laliluna.de
 *
 * @ejb.bean name = "Book"
 *          type = "CMP"
 *          cmp-version = "2.x"
 *          display-name = "Book"
 *          description = "Book EJB"
 *          view-type = "both"
 *          jndi-name = "ejb/BookHome"
 *          local-jndi-name = "ejb/BookLocalHome"
 *                primkey-field = "id"
 *
 * @ejb.util generate="physical"
 * @ejb.persistence table-name = "tbook"
 * @ejb.value-object match = "*" name="Book"
 *
 * @jboss.persistence create-table = "true"
 *                      remove-table = "true"
 * @jboss.entity-command name="postgresql-fetch-seq"
 */
public abstract class Book implements EntityBean {
```

In the next step we add getter- and setter-methods for the following fields.

- id (autoincrement key)
- title
- price
- customer_id (relation field)

The getter and setter look like the following.

```
/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fid"
 *                  jdbc-type = "INTEGER"
 *                  sql-type = "SERIAL"
 * @ejb.pk-field
 * @jboss.persistence auto-increment = "true"
 * @return
 */
public abstract Integer getId();

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

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

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

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fprice"
 * @return
 */
public abstract Float getPrice();

/**
 * @ejb.interface-method view-type = "both"
 * @param price
 */
public abstract void setPrice(Float price);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fcustomer_id"
 * @return
 */
```

```
public abstract Integer getCustomerId();

/**
 * @ejb.interface-method view-type = "both"
 * @param customer_id
 */
public abstract void setCustomerId(Integer customer_id);
```

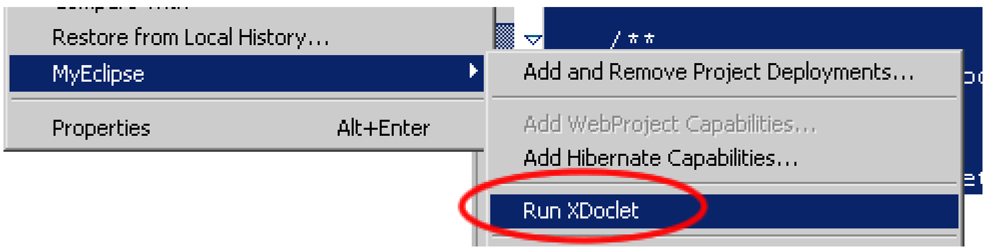
Within the class modify the **ejbCreate()** method. This must always exhibit the return typ of the primary key. (in our case the property id)

```
public Integer ejbCreate() throws CreateException {
      return null;
}
```

Run xDoclet

After all settings of xDoclet and all getter- and setter-methods are created, we run xDoclet to generate all needed interfaces.

Right mouse button on the project name in the package explorer > MyEclipse > Run xDoclet.


Restore from Local History...
MyEclipse
Properties
Alt+Enter
Add and Remove Project Deployments...
Add WebProject Capabilities...
Add Hibernate Capabilities...
Run XDoclet

if xDoclet runs succesfully, there will be a new package `de.laliluna.tutorial.finderexample.entity.interface` in the package explorer, which contains the interface classes of the two entity beans.

Relation between customer and book

Next we provide the relation between the entity bean customer and the entity bean book.

You will find informations how to create EJB relationships in this tutorial.
http://www.laliluna.de/simple-xdoclet-ejb-cmp-relation-tutorial.html

Open the entity bean customer and add the following getter and setter method.

```
/**
 * @ejb.interface-method view-type = "local"
 * @ejb.relation name = "customer_book"
 *                    role-name = "customer_book"
 *
 * @return Collection
 */
public abstract Collection getBooks();

/**
 * @ejb.interface-method view-type = "local"
 * @param books
 */
public abstract void setBooks(Collection books);
```

Open the entity bean book and also add a getter and setter method.

```
/**
 * @ejb.interface-method view-type = "local"
 * @ejb.relation name = "customer_book"
 *               role-name = "book_customer"
 * @jboss.relation related-pk-field = "id"
 *                 fk-column = "fcustomer_id"
 *                 fk-constraint = "true"
 * @return
 */
public abstract CustomerLocal getCustomer();

/**
 * @ejb.interface-method view-type = "local"
 * @param customer
 */
public abstract void setCustomer(CustomerLocal customer);
```

Value objects

At the end we need a getter and setter method for the value object for each entity bean.

Add the following getter and setter to the entity bean customer.

```
/**
 * @ejb.interface-method view-type = "both"
 * @return
 */
public abstract CustomerValue getCustomerValue();

/**
 * @ejb.interface-method view-type = "both"
 * @param customerValue
 */
public abstract void setCustomerValue(CustomerValue customerValue);
```

Within the entity bean book add the following getter and setter

```
/**
 * @ejb.interface-method view-type = "both"
 * @return
 */
public abstract BookValue getBookValue();

/**
 * @ejb.interface-method view-type = "both"
 * @param bookValue
 */
public abstract void setBookValue(BookValue bookValue);
```

The entity beans are created. Now we can start to create the finders for each entity bean.

Create xDoclet finder

Xdoclet supports a tag **@ejb.finder** to create a finder. This tag can only be used in the java docs above the class.

In the following example the most important parameters of the tag will be explained.

```
@ejb.finder description = "Find all customer"
            signature = "java.util.Collection findAll()"
            query = "select object(c) from Customer as c"
```

```
description = "Find all customer"
```
Description of the finder (optional)

```
signature = "java.util.Collection findAll()"
```
Defines the signature of the method, which will be added to the home interfaces of the entity bean by xDoclet. The return value and the parameters of the method must be set with the full package path (for example: java.util.Collection). (requiered)

```
query = "select object(c) from Customer as c"
```
Defines the EJB-QL query (optional)

EJB-QL query language

EJB-QL is a language to use for database querys with enterprise java beans. The syntax has some differences to SQL.

Basic Syntax

```
SELECT OBJECT( alias )
FROM EntityBean [AS] alias
[ WHERE criterion to restrict the query result ]
```

The following query gets all customers from the database.

```
SELECT OBJECT(c) FROM Customer as c
```

The selection can be restricted in the WHERE clause. Within the WHERE clause you always have to use the alias in front of a fields, which you want to access.

```
SELECT OBJECT(c) FROM Customer as c WHERE c.name = „Karl“
```

The IN operator allows to access elements in a collection relation field. The following query returns the books of all costumers.

```
SELECT OBJECT(b) FROM Customer as c, IN(c.books) as b
```

IS NULL is used to test for null valued fields. NOT is needed for negation. The following example returns all costumers, where the name is NULL.

```
SELECT OBJECT(c) FROM Customer as c WHERE c.name NOT IS NULL
```

IS EMPTY is used to test for empty sets, in particular relation fields that holds a collection of objects. For example, the following query returns all customers without known books in the database.

```
SELECT OBJECT(c) FROM Customer as c WHERE c.books IS EMPTY
```

JBOSS query

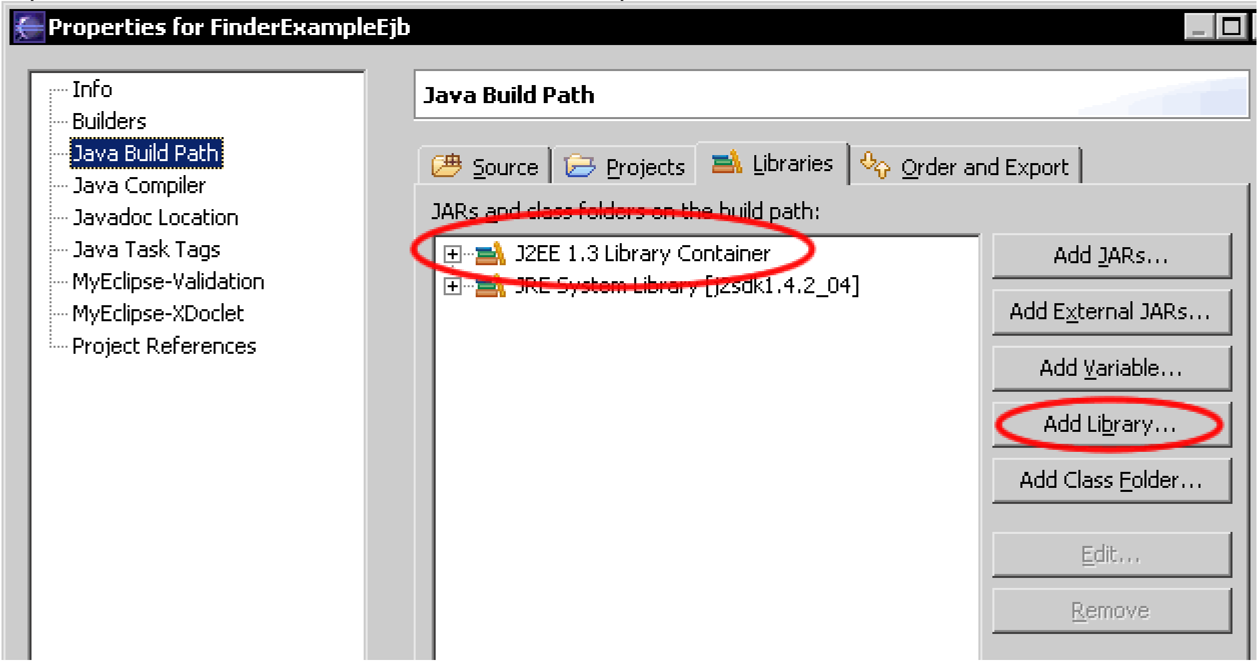
The tag **@jboss.query** can be used to take advantage from the Jboss-QL language. Jboss-QL extends the ejb query language. You must always specify both tags, ejb.finder and jboss.qery, but the first one is actuallay overwritten by the last one. Do not ask me why.

The following example extends a ejb finder to use an ORDER BY.
ORDER BY is supported by ejb-ql since J2EE 1.4. For previous versions you have to extend the ejb-ql with jboss-ql to use order by.

```
@ejb.finder description = "Find all customer"
            signature = "java.util.Collection findAll()"
            query = "SELECT OBJECT(c) FROM Customer AS c GROUP BY c.name"
@jboss.query description = "Find all customer"
```

```
             signature = "java.util.Collection findAll()"
             query = "SELECT OBJECT(c) FROM Customer AS c ORDER BY c.name"
```

If you want to test this, remove the the J2EE 1.4 library and add the J2EE 1.3 library on the project properties > Java-Build-Path > Libraries like the picture shown below.


Properties for FinderExampleEjb

Info
Builders
Java Build Path
Java Compiler
Javadoc Location
Java Task Tags
MyEclipse-Validation
MyEclipse-XDoclet
Project References

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

J2EE 1.3 Library Container
JRE System Library [j2sdk1.4.2_04]

Customer bean finders

findAll()

Open the entity bean Customer and add the xDoclet finder. The defined finder returns a collection of all local customer objects.

```
@ejb.finder description = "Find all customer"
            signature = "java.util.Collection findAll()"
            query = "select object(c) from Customer as c"
```

findByName(java.lang.String name)

This finder returns a customer, where the value of the parameter name is equal to the value of the field name in the entity bean. The return typ of the finder is the local inferface customer and must be set with the full package path.
(`de.laliluna.tutorials.finderexample.entity.interfaces.`**`CustomerLocal`**)

Also the parameter of the method must be set with the full package path.
(`java.lang.String`)

With ?<position of the parameter> (in our case ?1) you can access a parameter in the ejb query language.

```
* @ejb.finder description = "Find customers by name"
 *                     signature =
"de.laliluna.tutorials.finderexample.entity.interfaces.CustomerLocal
findByName(java.lang.String name)"
 *                     query = "select object(c) from Customer as c where c.name =
?1"
```

findByBookPrice(java.lang Float)

This finder returns all customers, which holds a book with a specific price. The IN operator get the collection books (relation field) and restrict the selection with the where clause.

```
@ejb.finder description = "Find customers by book price"
            signature = "java.util.Collection findByBookPrice(java.lang.Float
price)"
            query = "select object(c) from Customer as c, IN(c.books) b where
b.price = ?1"
```

Book bean finders

findAll()

The same finder like above, but here it gets all books.

```
@ejb.finder description = "Find all books"
            signature = "java.util.Collection findAll()"
            query = "select object(c) from Book as c"
```

findByPriceRange(java.lang.Float from, java.lang.Float to)

The finder returns all books, where the price of the books is between two values.

```
@ejb.finder description = "Find by price range"
            signature = "java.util.Collection findByPriceRange(java.lang.Float
from, java.lang.Float to)"
            query = "select object(c) from Book as c where c.price between ?1
and ?2"
```

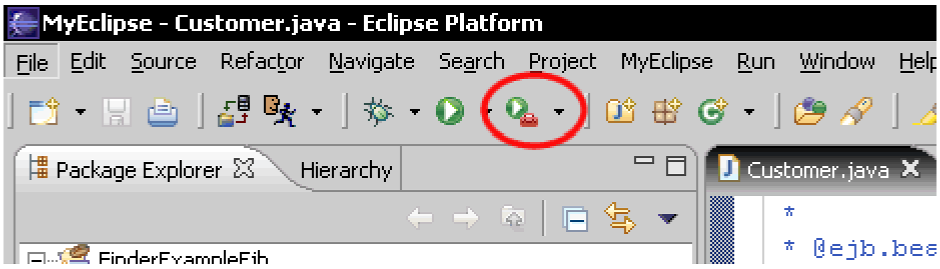
findByCustomerName(java.lang.String name)

Returns a list (collection) of books, where the value of the customer name is equal to the value of the method parameter. You do not use the IN Operator, because its a single valued relation field you have to accces to get the property name of the customer.

```
@ejb.finder description = "Find books by customer name"
         signature = "java.util.Collection findByCustomerName(java.lang.String
name)"
         query = "select object(c) from Book as c where c.customer.name = ?1"
```

Run xDoclet

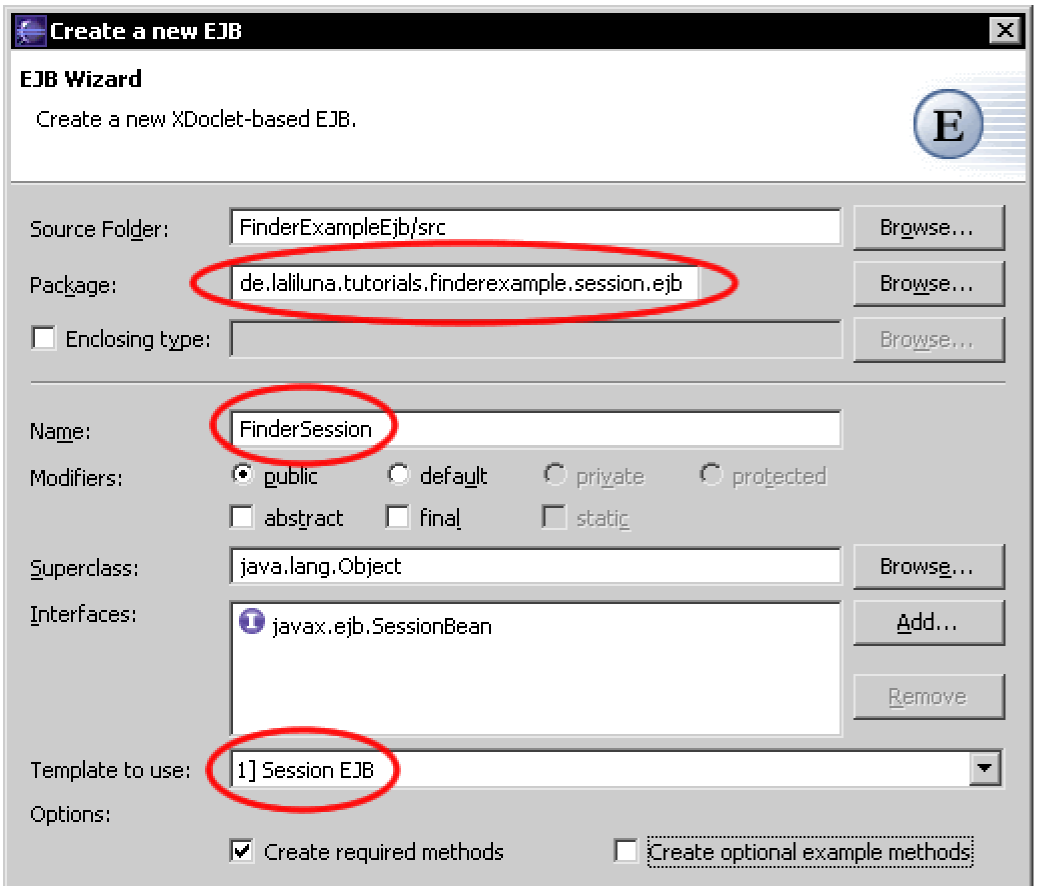
Now we have finished adding all finders and run xDoclet again. For the next time it is sufficiently to click on the run external tools symbol, shown in the picture below.


MyEclipse - Customer.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
Package Explorer
Hierarchy
Customer.java
FinderExampleEjb
*
* @ejb.bea

If xDoclet ran succesfully the finder methods are placed in the home interfaces of the entity beans.

Create a session bean

Create a new session bean `FinderSession` in the package
`de.laliluna.tutorial.finderexample.session.ejb`


Create a new EJB

EJB Wizard

Create a new XDoclet-based EJB.

Source Folder: FinderExampleEjb/src

Package: de.laliluna.tutorials.finderexample.session.ejb

Enclosing type: [ ]

Name: FinderSession

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces: javax.ejb.SessionBean

Template to use: 1] Session EJB

Options:

[✓] Create required methods [ ] Create optional example methods

In the next step we add the session methods, which calls the finder methods.

The session methods

getAllCustomers()

The session method getAllCustomers() calls the finder method findAll() in the local home interface CustomerLocalHome and returns a collection.

```
/**
 * Auslesen aller Kunden aus dem Local Home Interface
 *
 * @ejb.interface-method view-type = "both"
 */
public Collection getAllCustomers() throws EJBException {

      try {
            InitialContext context = new InitialContext();
            //hole das Local Home Interface über JNDI vom Applikationserver
            CustomerLocalHome customerLocalHome =
(CustomerLocalHome)context.lookup(CustomerLocalHome.JNDI_NAME);

            //Aufrufen der Finder Methode
            Collection collection = customerLocalHome.findAll();

            //hole ValueObjekt vom LocalObjekt und füge es der collection hinzu
            Collection customerList = new ArrayList();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  CustomerLocal element = (CustomerLocal) iter.next();
                  customerList.add(element.getCustomerValue());
            }

            //zurück geben der Collection von Kunden
            return customerList;

      } catch (FinderException e) {
            throw new EJBException(e.getMessage());
      } catch (NamingException e) {
            throw new EJBException(e.getMessage());
      }
}
```

getCustomerByName(String name)

This method calls the finder findByName(...) and returns the value object of customer.

```
/**
 * Auslesen eines Kunden anhand des Namen
 *
 * @ejb.interface-method view-type = "both"
 */
public CustomerValue getCustomerByName(String name) throws EJBException {

      try {
            InitialContext context = new InitialContext();
            //hole das Local Home Interface über JNDI vom Applikationserver
            CustomerLocalHome customerLocalHome =
(CustomerLocalHome)context.lookup(CustomerLocalHome.JNDI_NAME);

            //Aufrufen der Finder Methode
            CustomerLocal customerLocal = customerLocalHome.findByName(name);

            //zurück geben des value objektes des Kunden
            return customerLocal.getCustomerValue();

      } catch (FinderException e) {
            throw new EJBException(e.getMessage());
      } catch (NamingException e) {
            throw new EJBException(e.getMessage());
      }
}
```

getCustomersByBookPrice(Float price)

Within this method the finder findByBookPrice(..) is called.

```
/**
 * Auslesen aller Kunden anhand des Bücherdatums
 *
 * @ejb.interface-method view-type = "both"
 */
public Collection getCustomerByBookPrice(Float price) throws EJBException {

      try {
            InitialContext context = new InitialContext();
            //hole das Local Home Interface über JNDI vom Applikationserver
            CustomerLocalHome customerLocalHome =
(CustomerLocalHome)context.lookup(CustomerLocalHome.JNDI_NAME);

            //Aufrufen der Finder Methode
            Collection collection = customerLocalHome.findByBookPrice(price);
            //hole ValueObjekt vom LocalObjekt und füge es der collection hinzu
            Collection customerList = new ArrayList();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  CustomerLocal element = (CustomerLocal) iter.next();
                  customerList.add(element.getCustomerValue());
            }

            //zurück geben der Collection von Kunden
            return customerList;

      } catch (FinderException e) {
            throw new EJBException(e.getMessage());
      } catch (NamingException e) {
            throw new EJBException(e.getMessage());
      }
}
```

getAllBooks()

The method getAllBooks() calls the finder method findAll(..) of the local home interface BookLocalHome and returns a collection of books.

```
/**
 * Auslesen aller Bücher
 *
 * @ejb.interface-method view-type = "both"
 */
public Collection getAllBooks() throws EJBException {

      try {
            InitialContext context = new InitialContext();
            //hole das Local Home Interface über JNDI vom Applikationserver
            BookLocalHome bookLocalHome =
(BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);

            //Aufrufen der Finder Methode
            Collection collection = bookLocalHome.findAll();

            //hole ValueObjekt vom LocalObjekt und füge es der collection hinzu
            Collection bookList = new ArrayList();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  BookLocal element = (BookLocal) iter.next();
                  bookList.add(element.getBookValue());
            }

            //zurück geben der Collection von Büchern
            return bookList;

      } catch (FinderException e) {
            throw new EJBException(e.getMessage());
      } catch (NamingException e) {
            throw new EJBException(e.getMessage());
      }
}
```

getBooksByPriceRange(Float from, Float to)

Return a collection of books found by the finder method findByPriceRange(..)

```
/**
 * Auslesen von Büchern anhand einer Preisspanne
 *
 * @ejb.interface-method view-type = "both"
 */
public Collection getBooksByPriceRange(Float from, Float to) throws EJBException
{

      try {
            InitialContext context = new InitialContext();
            //hole das Local Home Interface über JNDI vom Applikationserver
            BookLocalHome bookLocalHome =
(BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);

            //Aufrufen der Finder Methode
            Collection collection = bookLocalHome.findByPriceRange(from, to);

            //hole ValueObjekt vom LocalObjekt und füge es der collection hinzu
            Collection bookList = new ArrayList();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  BookLocal element = (BookLocal) iter.next();
                  bookList.add(element.getBookValue());
            }

            //zurück geben der Collection von Büchern
            return bookList;

      } catch (FinderException e) {
            throw new EJBException(e.getMessage());
      } catch (NamingException e) {
            throw new EJBException(e.getMessage());
      }
}
```

getBooksByCustomerName(String name)

This method calls the finder findByCustomerName(..) and returns a collection of books, which were found.

```
/**
 * Auslesen von Büchern anhand des Namen des Kunden
 *
 * @ejb.interface-method view-type = "both"
 */
public Collection getBooksByCustomerName(String name) throws EJBException {

      try {
            InitialContext context = new InitialContext();
            //hole das Local Home Interface über JNDI vom Applikationserver
            BookLocalHome bookLocalHome =
(BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);

            //Aufrufen der Finder Methode
            Collection collection = bookLocalHome.findByCustomerName(name);

            //hole ValueObjekt vom LocalObjekt und füge es der collection hinzu
            Collection bookList = new ArrayList();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  BookLocal element = (BookLocal) iter.next();
                  bookList.add(element.getBookValue());
            }

            //zurück geben der Collection von Büchern
            return bookList;

      } catch (FinderException e) {
            throw new EJBException(e.getMessage());
```

```
        } catch (NamingException e) {
             throw new EJBException(e.getMessage());
        }
}
```

Session method to provide some dummy data.

We need some dummy data to test the finders.

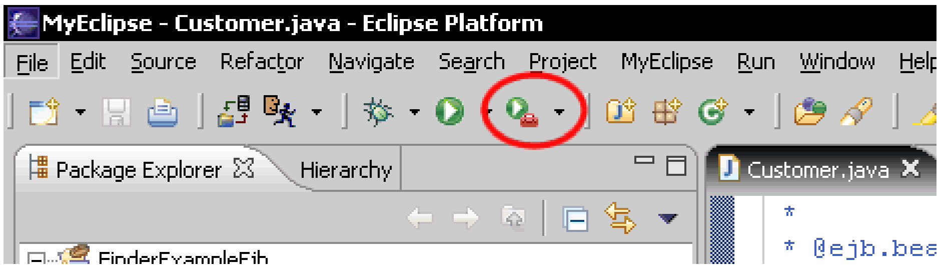
Copy the source code to the session bean class.

initSomeTestData()

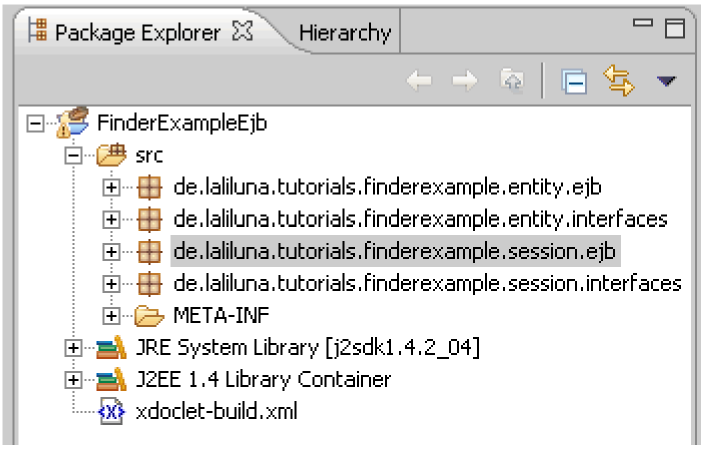
```
/**
 * Initalisieren von einigen Test-Daten
 *
 * @ejb.interface-method view-type = "both"
 */
public void initSomeTestData() throws EJBException {

      try {
            InitialContext context = new InitialContext();
            //hole das Local Home Interface über JNDI vom Applikationserver
            BookLocalHome bookLocalHome =
(BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);
            CustomerLocalHome customerLocalHome =
(CustomerLocalHome)context.lookup(CustomerLocalHome.JNDI_NAME);


            //delete all existing data
            //lösche alle vorhandenen Daten
            try {
                  for (Iterator iter = bookLocalHome.findAll().iterator();
iter.hasNext();) {
                        BookLocal element = (BookLocal) iter.next();
                        element.remove();
                  }
                  for (Iterator iter = customerLocalHome.findAll().iterator();
iter.hasNext();) {
                        CustomerLocal element = (CustomerLocal) iter.next();
                        element.remove();
                  }
            } catch (EJBException e1) {
                  e1.printStackTrace();
            } catch (FinderException e1) {
                  e1.printStackTrace();
            } catch (RemoveException e1) {
                  e1.printStackTrace();
            }


            //erstelle einige kunden
            CustomerLocal customerLocal1 = customerLocalHome.create();
            customerLocal1.setId(new Integer(1001));
            customerLocal1.setName("Karl");
            customerLocal1.setAge(new Integer(34));

            CustomerLocal customerLocal2 = customerLocalHome.create();
            customerLocal2.setId(new Integer(356));
            customerLocal2.setName("Betti");
            customerLocal2.setAge(new Integer(25));

            CustomerLocal customerLocal3 = customerLocalHome.create();
            customerLocal3.setId(new Integer(8887));
            customerLocal3.setName("Peter");
            customerLocal3.setAge(new Integer(65));

            //erstelle einige bücher
            BookLocal bookLocal1 = bookLocalHome.create();
            bookLocal1.setId(new Integer(89));
            bookLocal1.setTitle("Lost in Space");
            bookLocal1.setPrice(new Float(10.99));

            BookLocal bookLocal2 = bookLocalHome.create();
            bookLocal2.setId(new Integer(655));
            bookLocal2.setTitle("1984");
            bookLocal2.setPrice(new Float(20.99));

            BookLocal bookLocal3 = bookLocalHome.create();
            bookLocal3.setId(new Integer(3));
            bookLocal3.setTitle("Hero");
            bookLocal3.setPrice(new Float(8.99));

            //set the relations between customer and book
            bookLocal1.setCustomer(customerLocal2);
            bookLocal2.setCustomer(customerLocal3);
            bookLocal3.setCustomer(customerLocal1);


      } catch (CreateException e1) {
            throw new EJBException(e1.getMessage());
      } catch (NamingException e) {
            throw new EJBException(e.getMessage());
      }
}
```

Run xDoclet

Having finished the session bean , run xDoclet again.

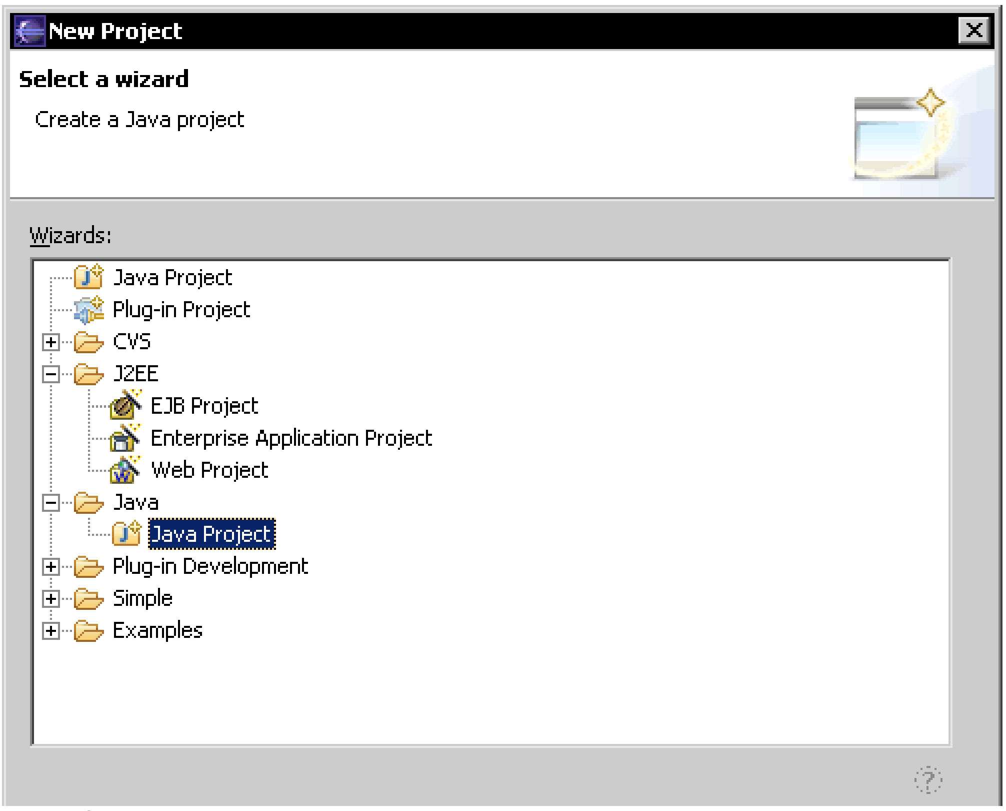

MyEclipse - Customer.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
Package Explorer
Hierarchy
Customer.java
FinderExampleEjb
* @ejb.bea

You will find a new Package, which was created by xDoclet in the package explorer.
`de.laliluna.tutorial.finderexample.session.interfaces`

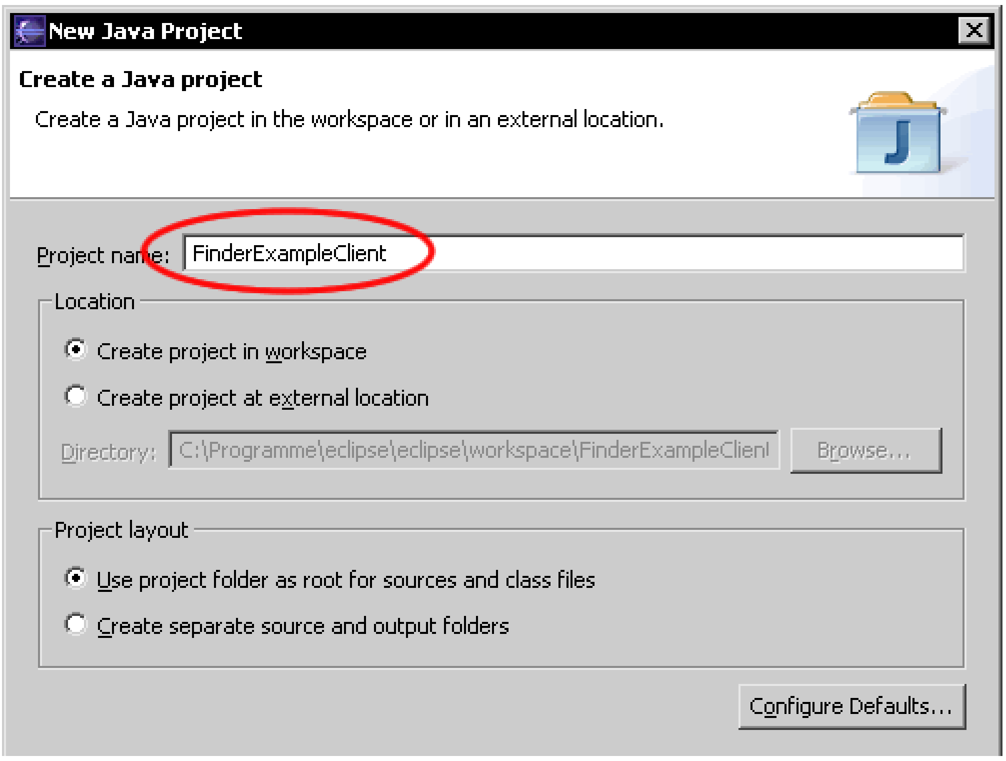

Package Explorer
Hierarchy
FinderExampleEjb
src
de.laliluna.tutorials.finderexample.entity.ejb
de.laliluna.tutorials.finderexample.entity.interfaces
de.laliluna.tutorials.finderexample.session.ejb
de.laliluna.tutorials.finderexample.session.interfaces
META-INF
JRE System Library [j2sdk1.4.2_04]
J2EE 1.4 Library Container
xdoclet-build.xml

Client application

We use a simple java project to test the enterprise java beans.
Create a new project (File > New > Project).



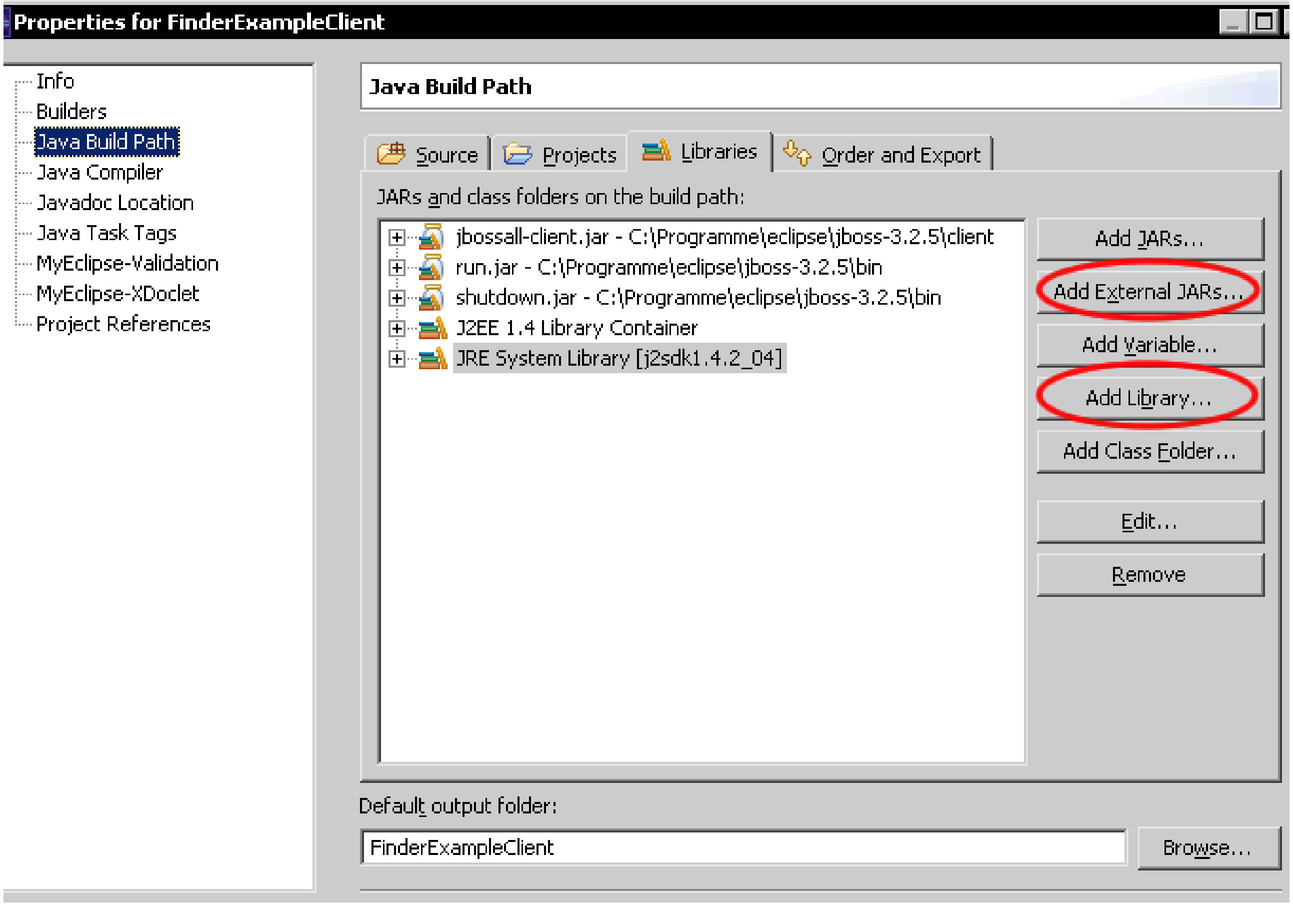
Set a nice name for the project.


New Java Project
Create a Java project
Create a Java project in the workspace or in an external location.
Location
Create project in workspace
Create project at external location
Browse...
Project layout
Use project folder as root for sources and class files
Create separate source and output folders
Configure Defaults...

Project name: FinderExampleClient

Directory: C:\Programme\eclipse\eclipse\workspace\FinderExampleClient

Some Libaries and JAR Files will be needed for the project. Add the libraries and jar files, which are shown in the picture below.


Properties for FinderExampleClient

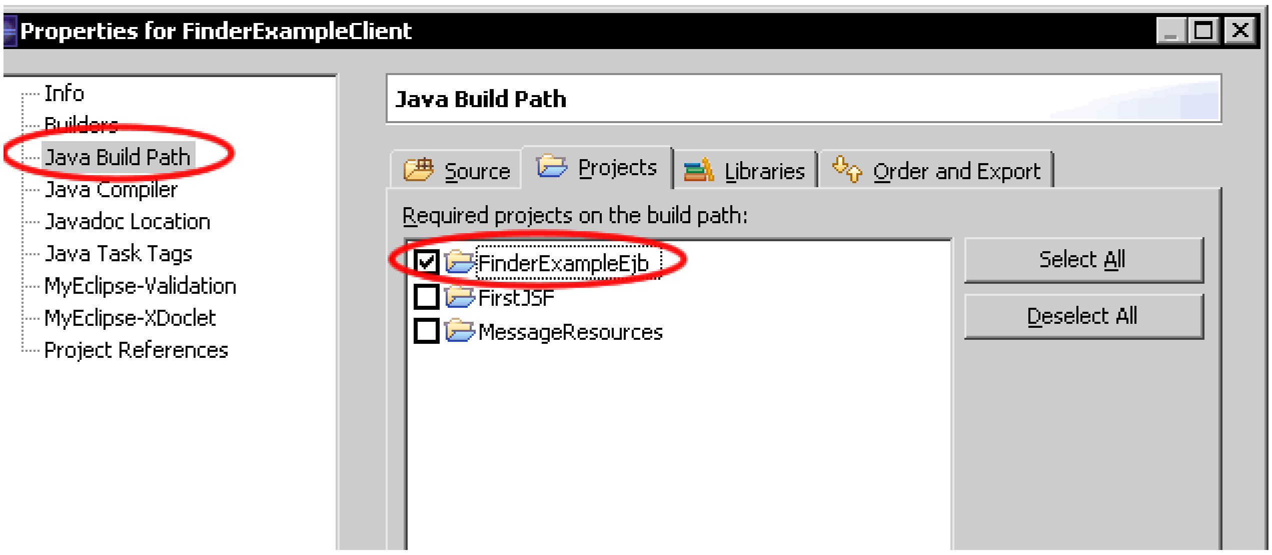
Java Build Path

JARs and class folders on the build path:

```
jbossall-client.jar - C:\Programme\eclipse\jboss-3.2.5\client
run.jar - C:\Programme\eclipse\jboss-3.2.5\bin
shutdown.jar - C:\Programme\eclipse\jboss-3.2.5\bin
J2EE 1.4 Library Container
JRE System Library [j2sdk1.4.2_04]
```

Default output folder: FinderExampleClient

On Projects assign the EJB project, so that the classes and methods of the EJB project can be accessed.


Properties for FinderExampleClient

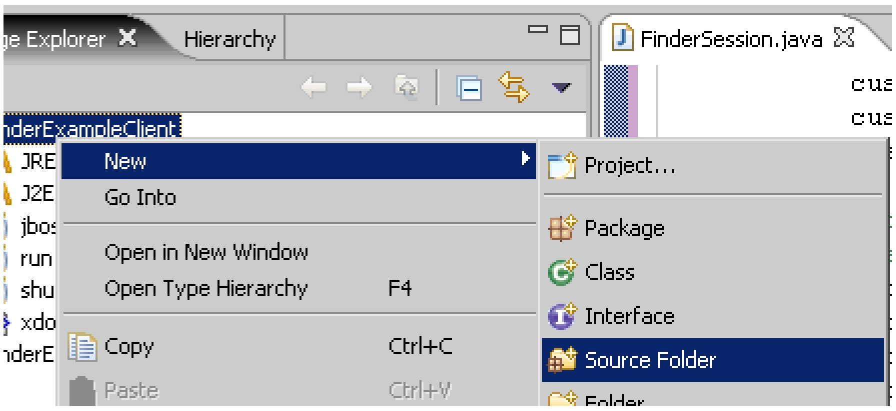
Java Build Path

Required projects on the build path:

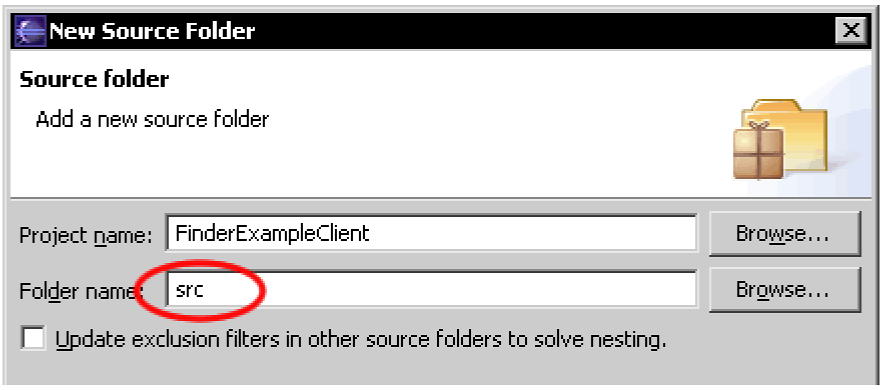
[✓] FinderExampleEjb
[ ] FirstJSF
[ ] MessageResources

Create a source folder and package

Create a new source folder. Right click on the project > New > Source Folder.



Set a name for the folder.



Source folder

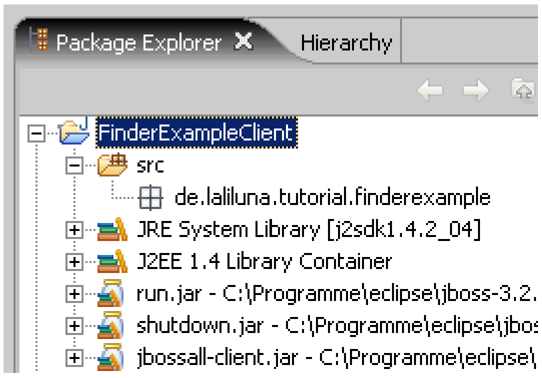
Add a new source folder

Project name: FinderExampleClient

Folder name: src

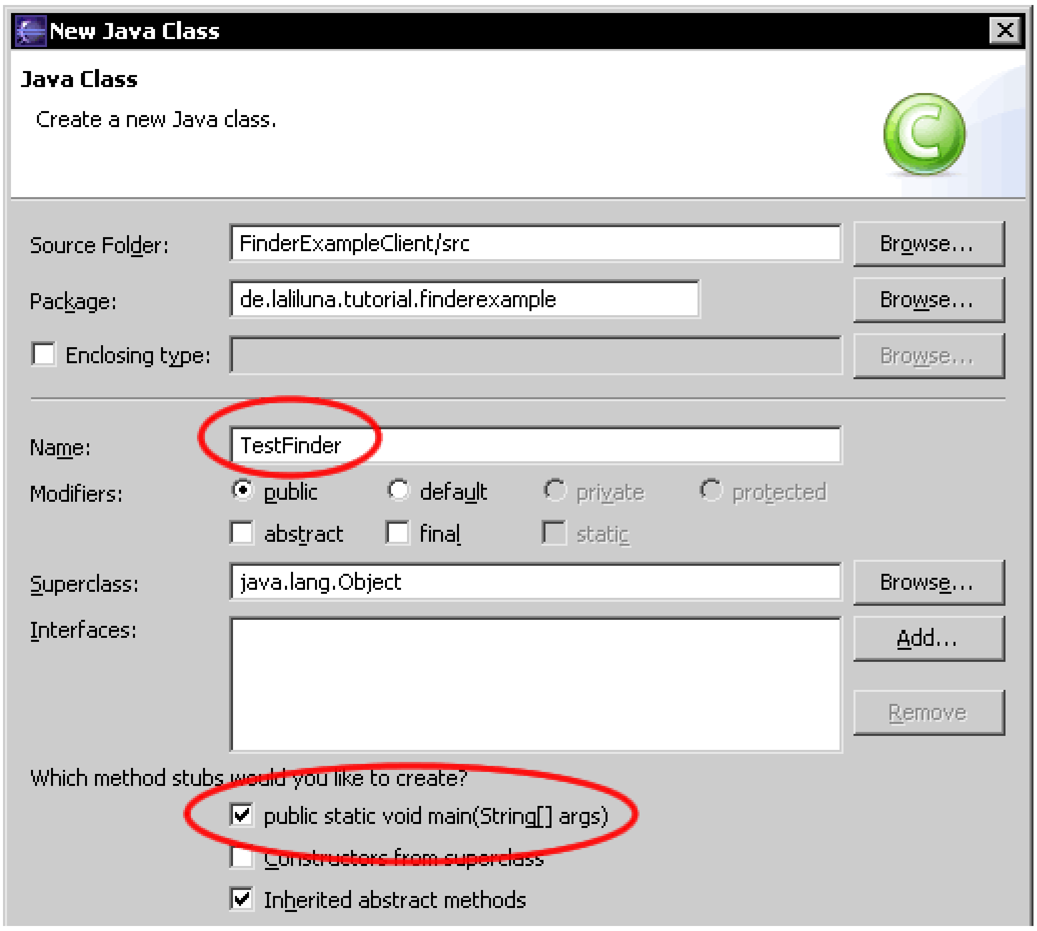
Update exclusion filters in other source folders to solve nesting.

Create a new package `de.laliluna.tutorial.finderexample` in the source folder.


Package Explorer
Hierarchy
FinderExampleClient
src
de.laliluna.tutorial.finderexample
JRE System Library [j2sdk1.4.2_04]
J2EE 1.4 Library Container
run.jar - C:\Programme\eclipse\jboss-3.2.
shutdown.jar - C:\Programme\eclipse\jbos
jbossall-client.jar - C:\Programme\eclipse\

Create the test class

Create a class and a `main(..)` method in the package
`de.laliluna.tutorial.finderexample`



New Java Class

Java Class

Create a new Java class.

Source Folder: FinderExampleClient/src

Package: de.laliluna.tutorial.finderexample

☐ Enclosing type:

Name: TestFinder

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

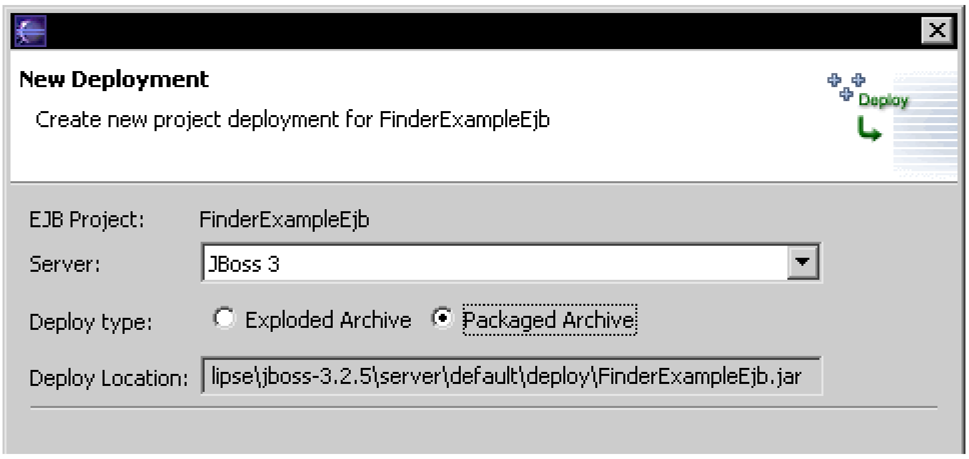
Add the following source code to the main(..) method.

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

      try {
            InitialContext context = new InitialContext(System.getProperties());
            //get the home interface with JNDI from the application server
            //hole das Home Interface über JNDI vom Applikationserver
            FinderSessionHome finderSessionHome = 
(FinderSessionHome)context.lookup(FinderSessionHome.JNDI_NAME);

            //create the session Objekt
            //erstelle das Session Objekt
            FinderSession finderSession = finderSessionHome.create();

            //call the init method to add some test data
            //aufrufen der Init Methode zum hinzufügen einiger Test-Daten
            finderSession.initSomeTestData();

            //Get all customers
            //Alle Kunden ausgeben
            System.out.println("All customers");

System.out.println("----------------------------------------------------");
            System.out.println("id, name, age");
            System.out.println("------------------");
            Collection collection = finderSession.getAllCustomers();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  CustomerValue element = (CustomerValue) iter.next();
                  System.out.print(element.getId() + ", ");
                  System.out.print(element.getName() + ", ");
                  System.out.print(element.getAge() + "\n");
            }



            //Get customer by Name
            //Kunde anhand seines Namens ausgeben
            System.out.println("\nCustomer 'Peter'");

System.out.println("-----------------------------------------------------");

            CustomerValue customerValue =
finderSession.getCustomerByName("Peter");
            System.out.print("id = " + customerValue.getId());
            System.out.print(",name = " + customerValue.getName());
            System.out.print(",age = " + customerValue.getAge() + "\n");



            //Get customers by book date
            //Kunden anhand des Bücherdatums ausgeben
            System.out.println("\nCustomer of books with the price 10.99");

System.out.println("-----------------------------------------------------");
            System.out.println("id, name, age");
            System.out.println("------------------");

            collection = finderSession.getCustomerByBookPrice(new Float(10.99));
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  CustomerValue element = (CustomerValue) iter.next();
                  System.out.print(element.getId() + ", ");
                  System.out.print(element.getName() + ", ");
                  System.out.print(element.getAge() + "\n");
            }

            //Get all books
            //Alle Bücher ausgeben
            System.out.println("\nAll books");

System.out.println("-----------------------------------------------------");
            System.out.println("id, name, age");
            System.out.println("------------------");

            collection = finderSession.getAllBooks();
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  BookValue element = (BookValue) iter.next();
                  System.out.print(element.getId() + ", ");
                  System.out.print(element.getTitle() + ", ");
                  System.out.print(element.getPrice() + "\n");
            }

            //Get books by price range
            //Bücher zwischen einer Preisspanne ausgeben
            System.out.println("\nBooks where the price is between 10 and 25");

System.out.println("-----------------------------------------------------");
            System.out.println("id, name, age");
            System.out.println("------------------");
            collection = finderSession.getBooksByPriceRange(new Float(10), new
Float(25));
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  BookValue element = (BookValue) iter.next();
                  System.out.print(element.getId() + ", ");
                  System.out.print(element.getTitle() + ", ");
                  System.out.print(element.getPrice() + "\n");
            }

            //Get books by customer name
            //Bücher anhand des Kundennamens ausgeben
            System.out.println("\nGet the books of customer 'Betti'");

System.out.println("---------------------------------------------------");
            System.out.println("id, name, age");
            System.out.println("------------------");

            collection = finderSession.getBooksByCustomerName("Betti");
            for (Iterator iter = collection.iterator(); iter.hasNext();) {
                  BookValue element = (BookValue) iter.next();
                  System.out.print(element.getId() + ", ");
                  System.out.print(element.getTitle() + ", ");
                  System.out.print(element.getPrice() + "\n");
            }




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

}
```

Test the application

Deployen des EJB Projektes

Start the Jboss Server and deploy the EJB project as Packaged Archive



New Deployment

Create new project deployment for FinderExampleEjb

EJB Project: FinderExampleEjb

Server: JBoss 3

Deploy type: Exploded Archive [✓ Packaged Archive]

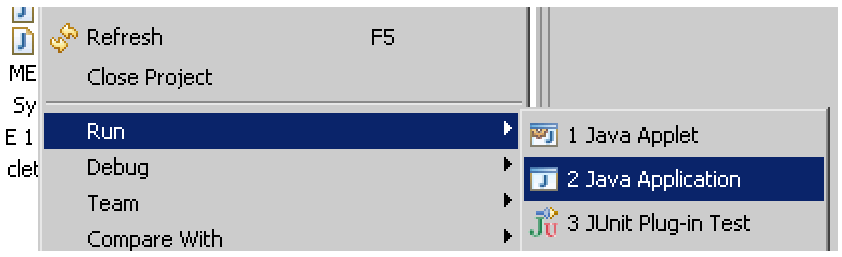
Deploy Location: lipse\jboss-3.2.5\server\default\deploy\FinderExampleEjb.jar

After a succesfull deployment you find the following lines in the console.

```
15:51:21,102 INFO  [EjbModule] Deploying Customer
15:51:21,102 INFO  [EjbModule] Deploying Customer
15:51:21,222 INFO  [EjbModule] Deploying Book
15:51:21,312 INFO  [EjbModule] Deploying FinderSession
15:51:22,384 INFO  [Customer] Created table 'tcustomer' successfully.
15:51:22,584 INFO  [Book] Created table 'tbook' successfully.
15:51:22,644 INFO  [Book] Added foreign key constraint to table 'tbook'
15:51:22,714 INFO  [EJBDeployer] Deployed: file:/C:/Programme/eclipse/jboss-
3.2.5/server/default/deploy/FinderExampleEjb.jar
```

Run the client application

Right mouse button on the project > Run > Java Application.



You get the following outputs in your console.

```
All customers
---------------------------------------------------
id, name, age
------------------
1, Karl, 34
2, Betti, 25
3, Peter, 65

Customer 'Peter'
---------------------------------------------------
id = 3,name = Peter,age = 65

Customer of books with the price 10.99
---------------------------------------------------
id, name, age
------------------
2, Betti, 25

All books
---------------------------------------------------
id, name, age
------------------
1, Lost in Space, 10.99
2, 1984, 20.99
3, Hero, 8.99

Books where the price is between 10 and 25
---------------------------------------------------
id, name, age
------------------
1, Lost in Space, 10.99
2, 1984, 20.99

Get the books of customer 'Betti'
---------------------------------------------------
id, name, age
------------------
1, Lost in Space, 10.99
3, Hero, 8.99
```