Introduction to Entity Beans

Entity Beans

- An *entity bean* represents a business object in a persistent storage mechanism. Some examples of business objects are customers, orders, and products.

  The bean represents a business entity, not a procedure. For example, CreditCardEJB would be an entity bean, but CreditCardVerifierEJB would be a session bean.

  The bean's state must be persistent. If the bean instance terminates or if the J2EE server is shut down or crashes, the bean's state still exists in persistent storage (a database).

Entity Beans: PERSISTENCE

- ***Persistence*** means that the entity bean's state exists beyond the lifetime of the application or the J2EE server process.

There are two types of persistence for entity beans: bean-managed and container-managed.

With *bean-managed persistence* (BMP), the entity bean code that you write contains the calls that access the database.

If your bean has *container-managed persistence* (CMP), the EJB container automatically generates the necessary database access calls. The code that you write for the entity bean does not include these calls.

Entity Beans: SHARED ACCESS

Entity beans may be shared by multiple clients.

- Because the clients might want to change the same data, it's important that entity beans work within transactions.

Typically, the EJB container provides transaction management. In this case, you specify the transaction attributes in the bean's deployment descriptor.

You do not have to code the transaction boundaries in the bean--the container marks the boundaries for you.

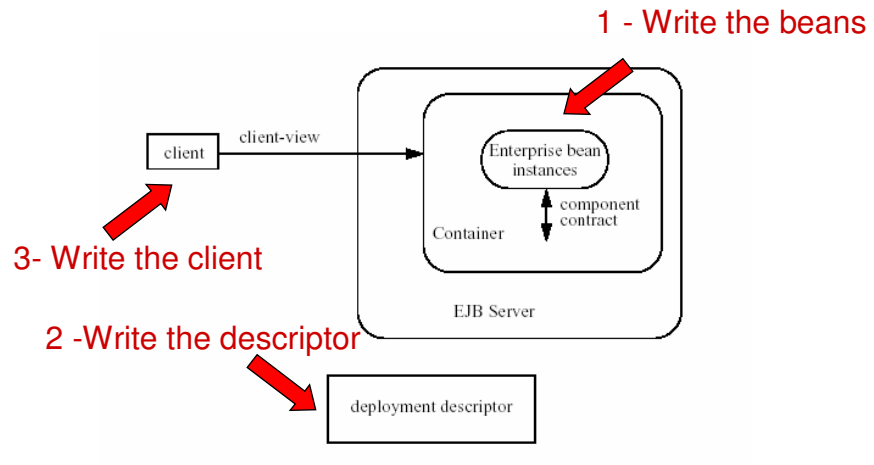
Entity Beans: DB-like features

Like in a relational database:
- Each entity bean has a unique object identifier;
- An entity bean may be related to other entity beans.

The unique identifier, or *primary key*, enables the client to locate a particular entity bean.

You implement relationships differently for entity beans with BMP and those with CMP:
- BMP: the code that you write implements the relationships.
- CMP: the EJB container takes care of the relationships for you. (*container-managed relationships*).

Architectural view



EJB ingredients

- **Interfaces:** The remote and home interfaces are required for remote access. For local access, the local and local home interfaces are required.

Enterprise bean class: Implements the methods defined in the interfaces.

Helper classes: Other classes needed by the enterprise bean class, such as exception and utility classes.

EJB ingredients

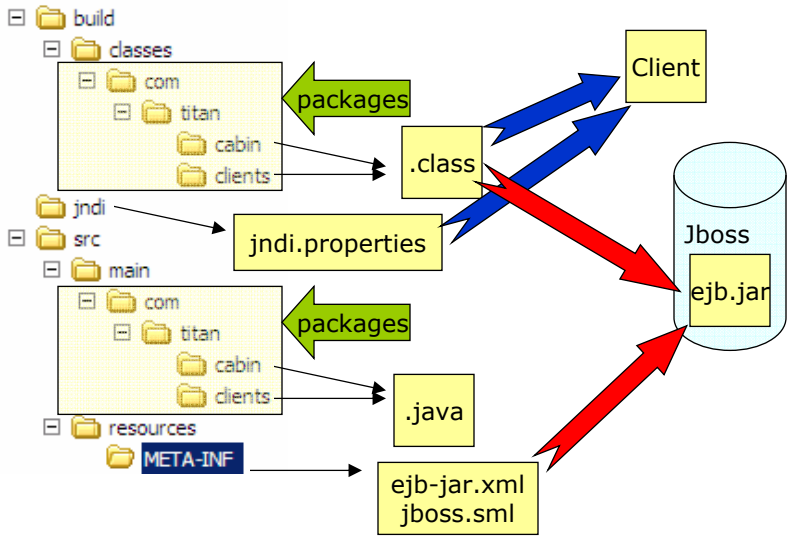
- **Deployment descriptor:** An XML file that specifies information about the bean such as its persistence type and transaction attributes.

- You package the files in the preceding list into an EJB JAR file, the module that stores the enterprise bean.
- To assemble a J2EE application, you package one or more modules--such as EJB JAR files--into an EAR file, the archive file that holds the application.

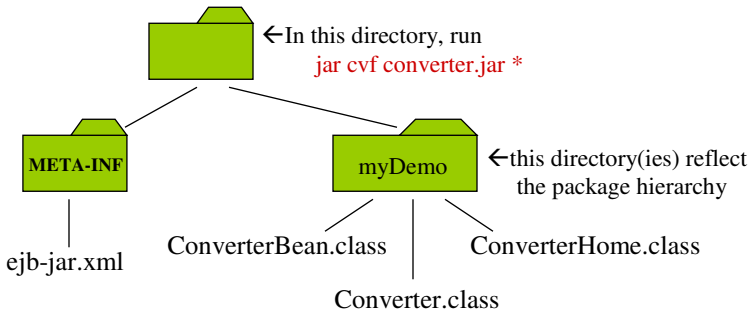
Introduction to Entity Beans

An example: part 1-the server

La struttura dei files



structure of the jar file



Watch out! Packages often make life complex.
(it's easy to make mistakes)
Try first without packages.

Esempio 1

1. CabinRemote

La classe deve estendere EJBObject

```
package com.titan.cabin;

import java.rmi.RemoteException;

public interface CabinRemote extends javax.ejb.EJBObject
{
  public String getName() throws RemoteException;
  public void setName(String str) throws RemoteException;
  public int getDeckLevel() throws RemoteException;
  public void setDeckLevel(int level) throws RemoteException;
  public int getShipId() throws RemoteException;
  public void setShipId(int sp) throws RemoteException;
  public int getBedCount() throws RemoteException;
  public void setBedCount(int bc) throws RemoteException;
}
```

DEFINIRE l'interfaccia per i metodi di accesso ai dati

Esempio 1

1. CabinBean

La classe deve implementare EntityBean

```
package com.titan.cabin;
import javax.ejb.EntityContext;
import javax.ejb.CreateException;
public abstract class CabinBean implements javax.ejb.EntityBean
{

  public Integer ejbCreate(Integer id) throws CreateException
  {
    this.setId(id);
    return null;
  }

  public void ejbPostCreate(Integer id) {}
```

DEFINIRE ejbCreate E ejbPostCreate (Anche se vuoto)

Esempio 1

1. CabinBean

```
  public abstract void setId(Integer id);
  public abstract Integer getId();

  public abstract void setShipId(int ship);
  public abstract int getShipId();

  public abstract void setName(String name);
  public abstract String getName();

  public abstract void setBedCount(int count);
  public abstract int getBedCount();

  public abstract void setDeckLevel(int level);
  public abstract int getDeckLevel();
```

DEFINIRE
I METODI ASTRATTI DI ACCESSO AI DATI
(quelli definiti dall'interfaccia)

Esempio 1

1. CabinBean

```
  public void setEntityContext(EntityContext ctx) {
       // Not implemented }

  public void unsetEntityContext()  { // Not implemented }

  public void ejbActivate()    { // Not implemented }

  public void ejbPassivate()  { // Not implemented }

  public void ejbLoad()         { // Not implemented }

  public void ejbStore()        { // Not implemented }

  public void ejbRemove()     { // Not implemented }
}
```

OBBLIGATORIO IMPLEMENTARE LE CALLBACK ANCHE SE VUOTE

Esempio 1

1. CabinHomeRemote

L'interfaccia deve estendere EJBHome

```
package com.titan.cabin;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface CabinHomeRemote extends javax.ejb.EJBHome
{
  public CabinRemote create(Integer id)
    throws CreateException, RemoteException;

  public CabinRemote findByPrimaryKey(Integer pk)
    throws FinderException, RemoteException;
}
```

DEFINIRE il metodo create e i finder

ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-
jar_2_0.dtd">

<ejb-jar>
 <enterprise-beans>
  ...
 </enterprise-beans>

 <assembly-descriptor>
  ...
 </assembly-descriptor>
</ejb-jar>
```

mysql-ds.xml

```
<datasources>
        <local-tx-datasource>
                <jndi-name>Cabin</jndi-name>
                <connection-url>
                    jdbc:mysql://localhost:3306/cabin</connection-
url>
                <driver-class>org.gjt.mm.mysql.Driver</driver-class>
                <user-name>root</user-name>
                <password/>
        </local-tx-datasource>
</datasources>
```

ejb-jar.xml

```
<enterprise-beans>
  <entity>
    <ejb-name>CabinEJB</ejb-name>
    <home>com.titan.cabin.CabinHomeRemote</home>
    <remote>com.titan.cabin.CabinRemote</remote>
    <ejb-class>com.titan.cabin.CabinBean</ejb-class>
    <persistence-type>Container</persistence-type>
    <prim-key-class>java.lang.Integer</prim-key-class>
    <reentrant>False</reentrant>
    <cmp-version>2.x</cmp-version>
    <abstract-schema-name>Cabin</abstract-schema-name>
    <cmp-field><field-name>id</field-name></cmp-field>
    <cmp-field><field-name>name</field-name></cmp-field>
    <cmp-field><field-name>deckLevel</field-name></cmp-field>
    <cmp-field><field-name>shipId</field-name></cmp-field>
    <cmp-field><field-name>bedCount</field-name></cmp-field>
    <primkey-field>id</primkey-field>
    <security-identity><use-caller-identity/></security-identity>
  </entity>
</enterprise-beans>
```

ejb-jar.xml

```
<assembly-descriptor>
  <security-role>
    <description>  This role represents everyone who is allowed full
        access to the Cabin EJB. </description>
    <role-name>everyone</role-name>
  </security-role>
  <method-permission>
    <role-name>everyone</role-name>
    <method>
      <ejb-name>CabinEJB</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
  <container-transaction>
    <method>
      <ejb-name>CabinEJB</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

Introduction to Entity Beans

An example: part 2-the client

jndi.properties

Il file jndi.properties deve essere nella application root

java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost:1099

Esempio 1

Client

```
import com.titan.cabin.CabinRemote;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.rmi.PortableRemoteObject;

public static Context getInitialContext()
    throws javax.naming.NamingException
  {
    return new InitialContext();
 /** context initialized by jndi.properties file
        Properties p = new Properties();
        p.put(Context.INITIAL_CONTEXT_FACTORY,
        "org.jnp.interfaces.NamingContextFactory");
        p.put(Context.URL_PKG_PREFIXES,
        "jboss.naming:org.jnp.interfaces");
        p.put(Context.PROVIDER_URL, "localhost:1099");
        return new javax.naming.InitialContext(p);
    */
  }
}
```

Contesto Inizializzato da Jndi.properties

Una possibile alternativa sarebbe:

Esempio 1

Client

```
public class Client_1
{
  public static void main(String [] args)
  {
    String command[]={
    "create",
    "read",
    "update",
    "delete"
    };
    if (args.length==0) {
      System.out.println("Usage:");
      System.out.println(command[0]);
      System.out.println(command[1]);
      System.out.println(command[2] +" name");
      System.out.println(command[3]);
      System.exit(0);
    }
    // continua...
```

Esempio 1

Client

Acquisici il contesto e stampalo

```
try
    {
     Context jndiContext = getInitialContext();
      java.util.Hashtable ht=jndiContext.getEnvironment();

      System.out.println("INITIAL_CONTEXT_FACTORY
        "+ht.get(Context.INITIAL_CONTEXT_FACTORY));

       System.out.println("URL_PKG_PREFIXES
        "+ht.get(Context.URL_PKG_PREFIXES));

       System.out.println("PROVIDER_URL
        "+ht.get(Context.PROVIDER_URL));

        Iterator i=ht.keySet().iterator();
       while (i.hasNext()) {
         Object o=i.next();
         System.out.println(o+" "+ht.get(o));
       }
```

Esempio 1

Client

```
Object ref = jndiContext.lookup("CabinHomeRemote");
//Object ref = jndiContext.lookup("CabinEJB");
CabinHomeRemote home = (CabinHomeRemote)
   PortableRemoteObject.narrow(ref,CabinHomeRemote.class);

if (args[0].equals(command[0])) { //create
  CabinRemote cabin_1 = home.create(new Integer(1));
  cabin_1.setName("Master Suite")
  cabin_1.setDeckLevel(1);
  cabin_1.setShipId(1);
  cabin_1.setBedCount(3);
 }

 else  if (args[0].equals(command[1])) { //read
  Integer pk = new Integer(1);
  CabinRemote cabin_2 = home.findByPrimaryKey(pk);
  System.out.println(cabin_2.getName());
  System.out.println(cabin_2.getDeckLevel());
  System.out.println(cabin_2.getShipId());
  System.out.println(cabin_2.getBedCount());
 }
```

home.create

home.find

Esempio 1

Client

```
else  if (args[0].equals(command[2])) { //update
  Integer pk = new Integer(1);
  CabinRemote cabin_3 = home.findByPrimaryKey(pk);
  cabin_3.setName(args[1]);
  System.out.println("Scritto: "+args[1]);
 }

else  if (args[0].equals(command[3])) { //delete
  Integer pk = new Integer(1);
  CabinRemote cabin_4 = home.findByPrimaryKey(pk);
  cabin_4.remove();
  System.out.println("Cancellato");
 }

else {
  System.out.print("Unrecognized command: "+args[0]);
 }
}
```

home.find

home.find

Esempio 1

Client

```
catch (java.rmi.RemoteException re){re.printStackTrace();}
catch (javax.ejb.RemoveException re){re.printStackTrace();}
catch (javax.naming.NamingException ne){ne.printStackTrace();
catch (javax.ejb.CreateException ce){ce.printStackTrace();}
catch (javax.ejb.FinderException fe){fe.printStackTrace();}
 }
}
```

Tratta le eccezioni e termina

- **Esecuzione:**
  - **ant run.client –Darg1=create** OK
  - **ant run.client –Darg1=read** OK
  - **ant run.client –Darg1=create** ERROR!
  - **ant run.client –Darg1=update pippo** OK
  - **ant run.client –Darg1=read** OK
  - **ant run.client –Darg1=delete** OK
  - **ant run.client –Darg1=read** ERROR!
  - **ant run.client –Darg1=create** OK

Esempio 1

Client

Perche'
Object ref = jndiContext.lookup("CabinHomeRemote");
Invece di
Object ref = jndiContext.lookup("CabinEJB");
?

jboss.xml

```
<?xml version="1.0" ?>
<jboss>
  <enterprise-beans>
    <entity>
      <ejb-name>CabinEJB</ejb-name>
      <jndi-name>CabinHomeRemote</jndi-name>
    </entity>
  </enterprise-beans>
</jboss>
```

Introduction to Entity Beans

An example: part 3-using ant

The build.xml

```
<?xml version="1.0"?>
<!-- JBoss build file                                          -->
<project name="JBoss" default="ejbjar" basedir=".">

  <property environment="env"/>
  <property name="src.dir" value="${basedir}/src/main"/>
  <property name="src.resources"
        value="${basedir}/src/resources"/>
  <property name="jboss.home" value="${env.JBOSS_HOME}"/>
  <property name="build.dir" value="${basedir}/build"/>
  <property name="build.classes.dir" value="${build.dir}/classes"/>

  <!-- Build classpath -->
  <path id="classpath">
      <fileset dir="${jboss.home}/client">
        <include name="**/*.jar"/>
      </fileset>
        <pathelement location="${build.classes.dir}"/>
        <!-- So that we can get jndi.properties for InitialContext -->
        <pathelement location="${basedir}/jndi"/>
  </path>
```

The build.xml

```
<property name="build.classpath" refid="classpath"/>
<!-- Prepares the build directory                              -->
<target name="prepare" >
   <mkdir dir="${build.dir}"/>
   <mkdir dir="${build.classes.dir}"/>
  </target>

<!-- Compiles the source code                                  -->
<target name="compile" depends="prepare">
   <javac srcdir="${src.dir}"
        destdir="${build.classes.dir}"
        debug="on"
        deprecation="on"
        optimize="off"
        includes="**">
         <classpath refid="classpath"/>
   </javac>
  </target>
```

The build.xml

```
<target name="ejbjar" depends="compile">
   <jar jarfile="build/titan.jar">
    <fileset dir="${build.classes.dir}">
          <include name="com/titan/cabin/*.class"/>
    </fileset>
    <fileset dir="${src.resources}/">
       <include name="**/*.xml"/>
    </fileset>
   </jar>
   <copy file="build/titan.jar"
        todir="${jboss.home}/server/default/deploy"/>
 </target>

 <!-- Cleans up generated stuff                                    -->
  <target name="clean.db">
   <delete dir="${jboss.home}/server/default/db/hypersonic"/>
 </target>
 <target name="clean">
   <delete dir="${build.dir}"/>
   <delete file="${jboss.home}/server/default/deploy/titan.jar"/>
 </target>
```

The build.xml

```
  <target name="run.client" depends="ejbjar">
   <java classname="com.titan.clients.Client_1" fork="yes" dir=".">
    <classpath refid="classpath"/>
    <arg value="${arg1}"/>
    <arg value="${arg2}"/>
   </java>
 </target>

</project>
```

NOTA: lanciare con $ant run.client –Darg1=update –Darg2=pippo

Introduction to Entity Beans

More details

Actors:

Component Interface:

- **Same role as in RMI**

Component Implementation:

Same role as in RMI

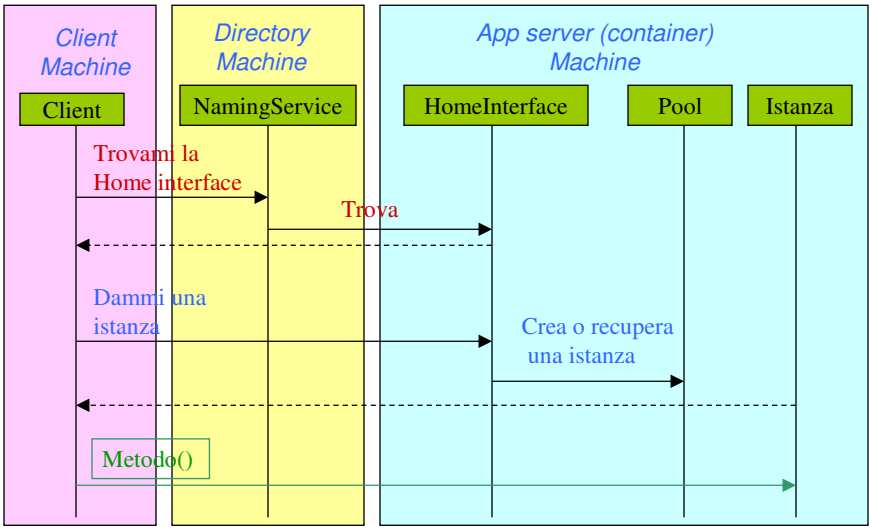
Home Interface:

Implemented by the server, plays the role of a factory class for the component

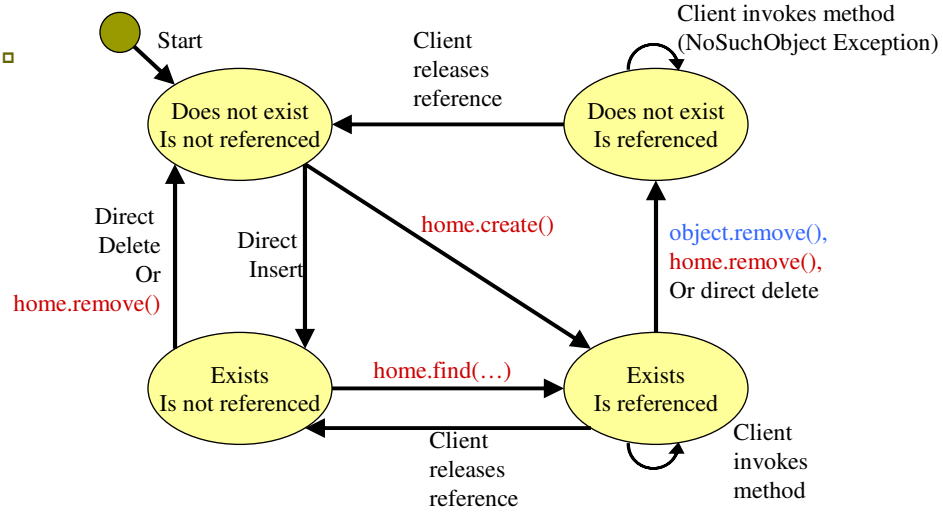
JNDI service:

Plays the same role as the register in RMI

The logical architecture

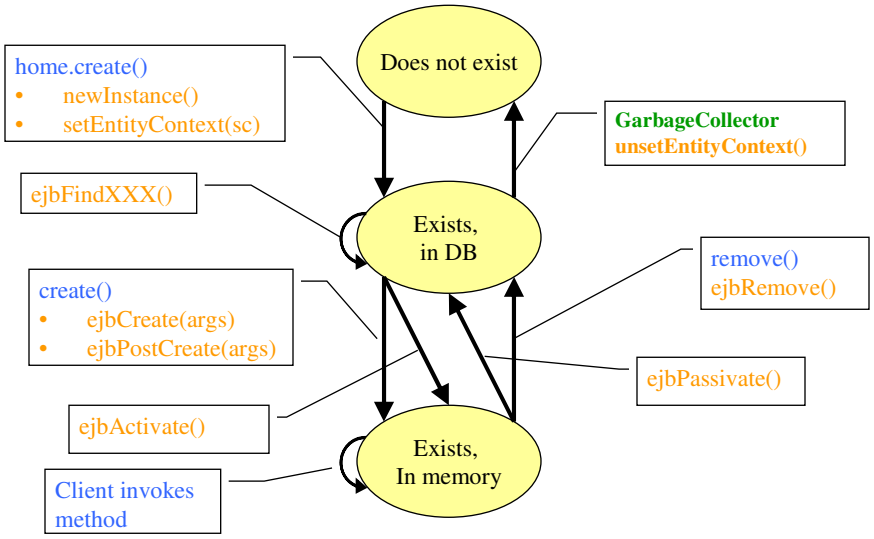


Entity Beans Lifecycle: client's view



19

Entity Beans Lifecycle (stati, transizioni e callbacks)



Introduction to Entity Beans

Local interfaces

Esempio 1

1. CabinLocal

La classe deve estendere EJBLocalObject

```
package com.titan.cabin;

import java.rmi.EJBException;

public interface CabinLocal extends javax.ejb.EJBLocalObject
{
  public String getName() throws EJBException;
  public void setName(String str) throws EJBException;
  public int getDeckLevel() throws EJBException;
  public void setDeckLevel(int level) throws EJBException;
  public int getShipId() throws EJBException;
  public void setShipId(int sp) throws EJBException;
  public int getBedCount() throws EJBException;
  public void setBedCount(int bc) throws EJBException;
}
```

Lancia EJBException anziche' RemoteException

Esempio 1

1. CabinHomeLocal

L'interfaccia deve estendere EJBHome

```
package com.titan.cabin;

import java.rmi.EJBException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface CabinHomeLocal
                    extends javax.ejb.EJBLocalHome
{
  public CabinLocal create(Integer id)
    throws CreateException, RemoteException;

  public CabinLocal findByPrimaryKey(Integer pk)
    throws FinderException, RemoteException;
}
```

DEFINIRE il metodo create e i finder

ejb-jar.xml

```
<enterprise-beans>
  <entity>
    <ejb-name>CabinEJB</ejb-name>
    <home>com.titan.cabin.CabinHomeRemote</home>
    <remote>com.titan.cabin.CabinRemote</remote>
    <local-home>com.titan.cabin.CabinHomeLocal</local-home>
    <local>com.titan.cabin.CabinLocal</local>
    <ejb-class>com.titan.cabin.CabinBean</ejb-class>
    ...
  </entity>
</enterprise-beans>
```

Esempio 1

1. Client

```
Object ref = jndiContext.lookup("CabinHomeRemote");
CabinHomeRemote home = (CabinHomeRemote)
        PortableRemoteObject.narrow(ref,CabinHomeRemote.class);
...
CabinRemote cabin_1 = home.create(new Integer(1));
...
CabinRemote cabin_2 = home.findByPrimaryKey(pk);
```

```
CabinHomeLocal home = (CabinHomeLocal)
        jndiContext.lookup("java:comp/env/ejb/CabinHomeLocal");
...
CabinLocal cabin_1 = home.create(new Integer(1));
...
CabinLocal cabin_2 = home.findByPrimaryKey(pk);
```