Documentation

Class: 5AHITT
Last modification: 11/28/2014
Document Version: 1.0

Dezentrale Systeme

2014/2015

Christian Bobek, Osman Özsoy

Java Naming and Directory Interface

List of contents

Statement of task

Follow the introduction and instructions of the **Java Naming and Directory Interface (JNDI)** described in this tutorial https://docs.oracle.com/javase/tutorial/jndi/TOC.html.

Setup your own or use an pre-existing naming service and implement following operations:

- Lookup an Object (1)
- List the Context (1)
- Add, Replace or Remove a Binding (2)
- Rename (1)
- Create and Destroy Subcontexts (1)
- Attribute Names (1)
- Read Attributes (1)
- Modify Attributes (1)
- Add, Replace Bindings with Attributes (2)
- Search
  * Basic Search (1)
  * Filters (1)
  * Scope (1)
  * Result Count (1)
  * Time Limit (1)

Create a protocol in which you describe 1) your **installation steps**, 2)**source code snippets** used to perform the naming operations and 3)**result of each operation**. Pack the protocol and sources into a **JAR file** and upload it here.

Size of Group: **2 persons**
(if you work without a team member then I will recommend to use the name service of another group)

Apportionment of work with effort estimation

Competent person(s)	Task	Description	Estimated time in h
Bobek, Özsoy	Setting up a naming service or preparing a pre-existing naming service	The description of each task is specified in the given tutorial: https://docs.oracle.com/javase/tutorial/jndi/TOC.html	2
Özsoy	Lookup an Object		1
Bobek	List the Context		1
Özsoy	Add, Replace or Remove a Binding		1
Bobek	Rename		1
Özsoy	Create and Destroy Subcontexts		1
Bobek	Attribute Names		1
Özsoy	Read Attributes		1
Bobek	Modify Attributes		1
Özsoy	Add, Replace Bindings with Attributes		1
Bobek	Basic Search		1
Özsoy	Filters		1
Bobek	Scope		1
Özsoy	Result Count		1
Bobek	Time Limit		1
Bobek, Özsoy	Creating a protocol	Create a protocol in which you describe 1) your **installation steps**, 2) **source code snippets** used to perform the naming operations and 3) **result of each operation**.	2

Estimated total time exposure

Person	Time exposure in h
Bobek	11
Özsoy	11
Sum:	**22**

Final time apportionment

Competent person(s)	Task	Estimated time in h	Actual time in h	Comment
Bobek, Özsoy	Setting up a naming service or preparing a pre-existing naming service	2	6	The setup of a new naming service was very difficult, so it takes more time to get it successfully work
Özsoy	Lookup an Object	1	1	/
Bobek	List the Context	1	2	/
Özsoy	Add, Replace or Remove a Binding	1	3	/
Bobek	Rename	1	1	/
Özsoy	Create and Destroy Subcontexts	1	1	/
Bobek	Attribute Names	1	0	No implementation was necessary at this point. There were just information to read in the given tutorial.
Özsoy	Read Attributes	1	2	/
Bobek	Modify Attributes	1	1.5	Modify Attributes was also not available in the tutorial, so weh ad to implement it by ourselves
Özsoy	Add, Replace Bindings with Attributes	1	1.5	The tutorial contains only bindings “without” attributes, so we implemented it again by ourselves
Bobek	Basic Search	1	0.5	/
Özsoy	Filters	1	0.5	/
Bobek	Scope	1	0.5	/
Özsoy	Result Count	1	0.5	/
Bobek	Time Limit	1	0.5	/
Bobek, Özsoy	Creating a protocol	2	5	We did more changes in the protocol while doing the exercise as expected

Actual total time exposure

Person	Time exposure in h
Bobek	26.5
Özsoy	26.5
Sum:	**53**

Task execution

Setting up a new naming service – OpenLDAP-Installation

The first step in our task execution was **setting up a new naming service**. We decided to do the setup in a virtual machine, which we imported in VMware Workstation. The operating system of the virtual machine was **debian**.

We had to install and configure **OpenLDAP**, which is our naming service. The following installation and configuration steps were necessary to get OpenLDAP work:

Updating the VM:

```
# apt-get update
```

Installing the OpenLDAP server:

```
# apt-get install slapd ldap-utils libldap-2.4-2 libdb4.6
```

During the installation process, we are requested to determine a password for the **admin entry**. So we set the admin password to: **admin**

Once the installation-process is finished, then the OpenLDAP server should run successfully.

Standardly OpenLDAP uses the database named "`olcDatabase={1}mdb.ldif`", which is located in "`/etc/ldap/slapd.d/cn=config`".

To manage the entries of the used database on the OpenLDAP server, we have to install **JXplorer** under debian:

```
# apt-get install jexplorer
```

To start JXplorer, use the following command:

```
# jxplorer
```

The properties below are standardly determined (by the installation-process):

```
dc=nodomain
cn=admin
```

We changed the domain name (dc) to "**jndi_dezsys**". To do this, we had to change all of the "**dc=nodomain**" properties in the file "/etc/ldap/slapd.d/cn=config/olcDatabase={1}mdb.ldif" to "**dc=jndi_dezsys**".

Consider: Before modifying this file, stop OpenLDAP server with following command:

```
# service slapd stop
```

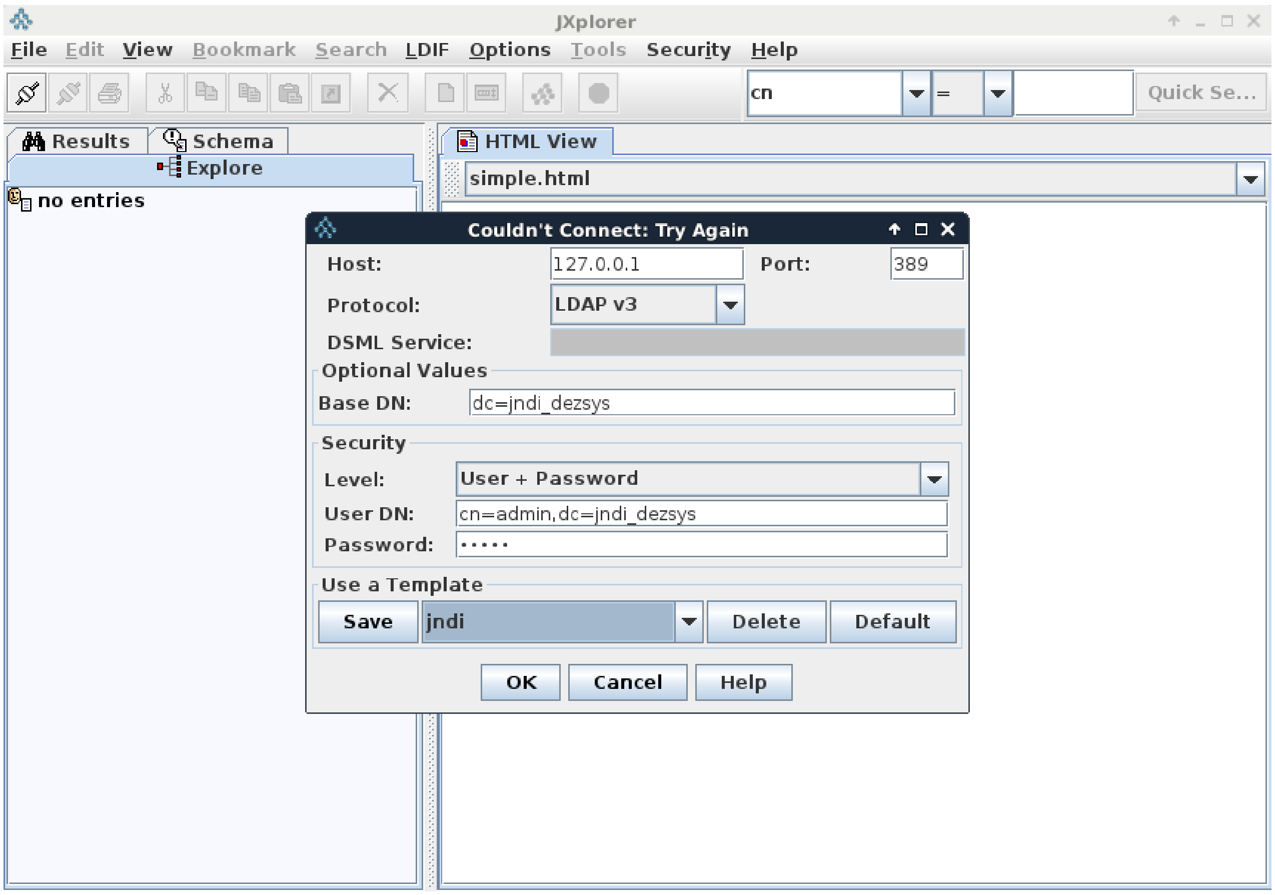
After finishing the modification start the server again with:

```
# service slapd start
```

After we have configured all necessary properties, we started **JXplorer to manage our entries** in our OpenLDAP database. To connect to our database over JXplorer, we have to push the **connect-button** and type in mandatory connection information.

Once you have filled in the information, you can save it as a **template** and use this saved template when reconnecting to the same database.

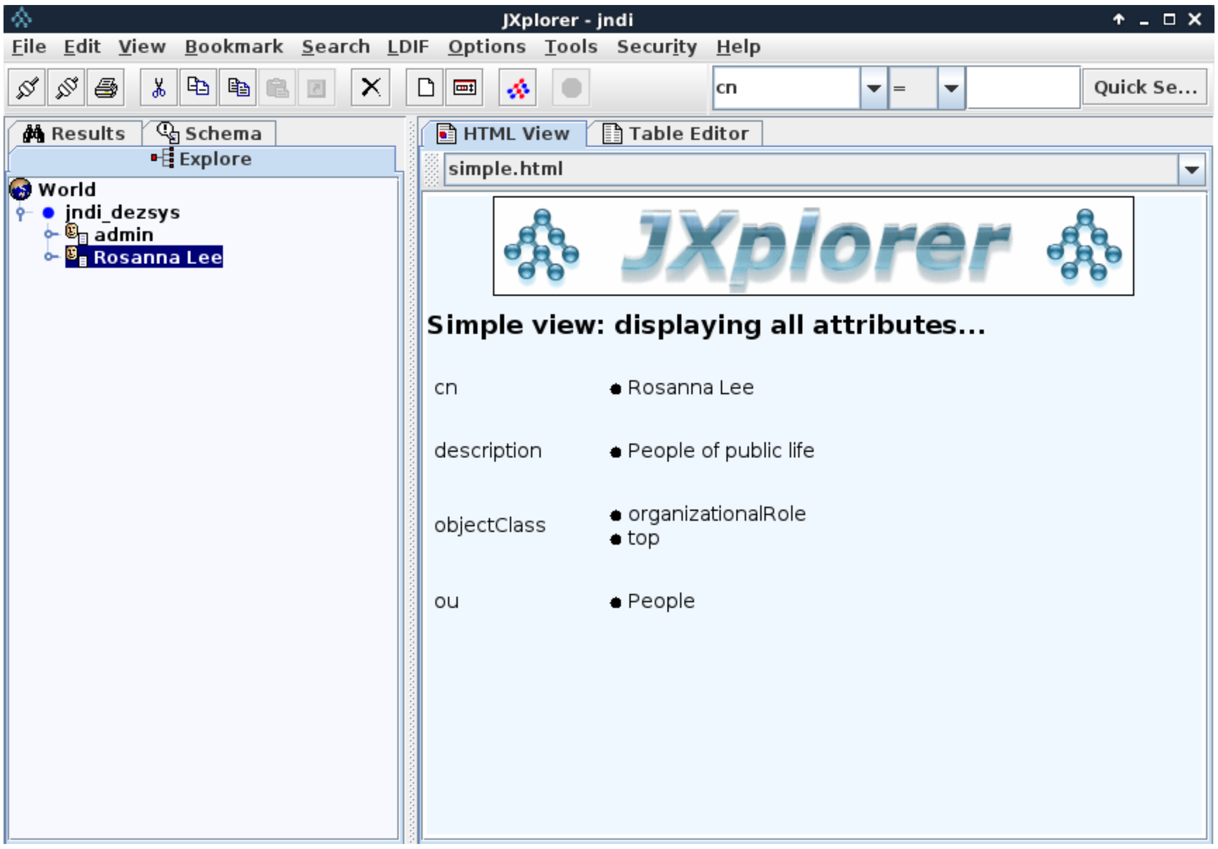
Our **connection-template** in JXplorer looks as below obvious (we named the template "**jndi**"):



Host: 127.0.0.1 Port: 389
Protocol: LDAP v3
DSML Service:
Base DN: dc=jndi_dezsys
Level: User + Password
User DN: cn=admin,dc=jndi_dezsys
Password: •••••
Template: jndi

After successful connection to the database, we should be able to **add**, **delete** or **modify entries** of the **database**. The "**admin**"-entry should be **standardly** in the database.

In the next step, we created a new entry named “**Rosanna Lee**” by right-clicking “**jndi_dezsys**” and selecting the **“New entry”-option**:



Simple view: displaying all attributes...

cn	● Rosanna Lee
description	● People of public life
objectClass	● organizationalRole ● top
ou	● People

OpenLDAP-Installation towards Prof. Borko's description

Like above, use or create a virtual machine (with a linux distribution) to type in the following commands for setting up/installing the LDAP-Server on it.

```
apt-get update
apt-get install slapd
dpkg-reconfigure -plow slapd
==>
tgm.ac.at
ldapadmin
MDM
remove, and don't move old database
<==

vim /etc/default/slapd
==>
SLAPD_SERVICES="ldap:/// ldapi:/// ldaps:///"
<==

vim loglevel.ldif
==>
dn: cn=config
changetype: modify
add: olcLoglevel
olcLoglevel: -1
<==

vim memberof.ldif
==>
dn: cn=module,cn=config
cn: module
objectclass: olcModuleList
objectclass: top
olcmoduleload: memberof.la
olcmodulepath: /usr/lib/ldap

dn: olcOverlay={0}memberof,olcDatabase={1}mdb,cn=config
objectClass: olcConfig
objectClass: olcMemberOf
objectClass: olcOverlayConfig
objectClass: top
olcOverlay: memberof
dn: cn=module,cn=config
cn: module
objectclass: olcModuleList
objectclass: top
olcmoduleload: refint.la
olcmodulepath: /usr/lib/ldap
<==
vim refint.ldif
==>
dn: olcOverlay={1}refint,olcDatabase={1}mdb,cn=config
objectClass: olcConfig
objectClass: olcOverlayConfig
objectClass: olcRefintConfig
objectClass: top
olcOverlay: {1}refint
olcRefintAttribute: memberof member manager owner
<==

ldapmodify -Y EXTERNAL -H ldapi:/// -f loglevel.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f memberof.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f refint.ldif

ldapsearch -D cn=admin,dc=tgm,dc=ac,dc=at -W -b dc=tgm,dc=ac,dc=at
```

Now we should be able to connect to the LDAP-Server (= this VM) via **ApacheDirectoryStudio**. **Adaptations** should be done via **ldif-Files**.

Necessary LDAP-Setup/-Configuration for the tutorial after LDAP-Installation

Directory Schema

The shemes "**java.schema**" and "**corba.schema**" must be installed for this tutorial. To make sure, if the necessary shemes are already installed on the server, go to "`/etc/ldap/schema/`" and execute the command "`ls`". If there are the following files, then you don't have to do anything, otherwise you have to install this shemes towards the LDAP-Setup description[1] in the tutorial:

- java.schema
- corba.schema
- java.ldif
- corba.ldif

Providing directory content for this tutorial

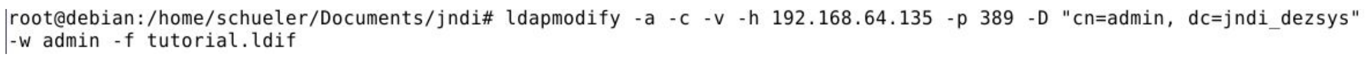
To get the scheduled directory content, you have to download the LDIF-File, called "tutorial.ldif" from the tutorial website and save it into a arbitrary place/folder on the server.

We had to do the following changes after downloading this file, to get the directory content into the database of our LDAP-Server:

We replaced "`o=JNDITutorial`" to "`o=jndi_dezsys`" in the whole file.

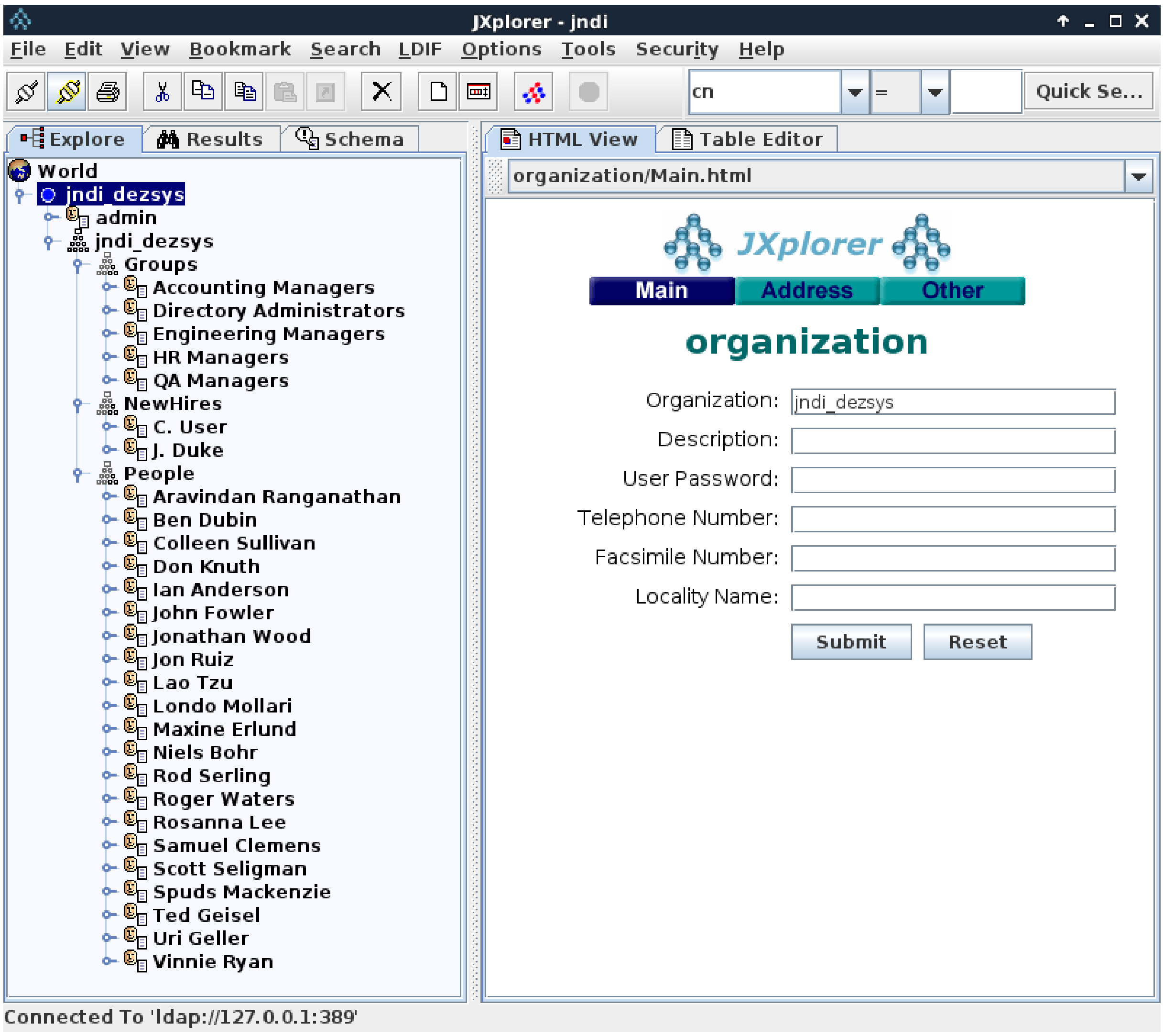
We added "`, dc=jndi_dezsys`" to the rowend of each line, which begins with "`dn:`".

After doing this changes, we executed the following command in the directory, where the ldif-file is placed:



```
root@debian:/home/schueler/Documents/jndi# ldapmodify -a -c -v -h 192.168.64.135 -p 389 -D "cn=admin, dc=jndi_dezsys"
-w admin -f tutorial.ldif
```

Afterwards we started the JXplorer and connected us to our LDAP-Database by using our saved template called "**jndi**" and made sure that the provided content is inserted into LDAP:



```
World
  jndi_dezsys
    admin
    jndi_dezsys
      Groups
        Accounting Managers
        Directory Administrators
        Engineering Managers
        HR Managers
        QA Managers
      NewHires
        C. User
        J. Duke
      People
        Aravindan Ranganathan
        Ben Dubin
        Colleen Sullivan
        Don Knuth
        Ian Anderson
        John Fowler
        Jonathan Wood
        Jon Ruiz
        Lao Tzu
        Londo Mollari
        Maxine Erlund
        Niels Bohr
        Rod Serling
        Roger Waters
        Rosanna Lee
        Samuel Clemens
        Scott Seligman
        Spuds Mackenzie
        Ted Geisel
        Uri Geller
        Vinnie Ryan
```

organization

Organization:	jndi_dezsys
Description:	
User Password:	
Telephone Number:	
Facsimile Number:	
Locality Name:	

Connected To 'ldap://127.0.0.1:389'

But you can also check the insertations by using the following command:

```
root@debian:/home/schueler/Documents/jndi# ldapsearch -D cn=admin,dc=jndi_dezsys W -b dc=jndi_dezsys -w admin
```

Implementing the Naming and Directory Operations

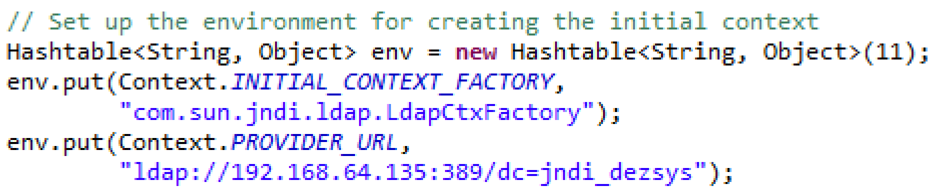
To implement the Naming and Directory Operations, we created a **Java-Project** in **Eclipse** and downloaded all the Sample-Files for the provided operations.

The downloaded examples didn’t work, so we had to adapt many implementations in the source-codes to get the examples run.

To set up the environment for creating the initial context in the Java-Classes:

- GetAllAttrs
- List
- ListBindings
- Lookup
- Search
- SearchCountLimit
- SearchObject
- SearchRetAll
- SearchSubtree
- SearchTimeLimit
- SearchWithFilter
- SearchWithFilterRetAll

we implemented the following lines:



```
// Set up the environment for creating the initial context
Hashtable<String, Object> env = new Hashtable<String, Object>(11);
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL,
        "ldap://192.168.64.135:389/dc=jndi_dezsys");
```

To set up the environment for creating the initial context in the Java-Classes:

- Bind
- BindWithAttrs
- Create
- Destroy
- ModifyAttrs
- Rebind
- RebindWithAttrs
- Rename
- Unbind

we implemented the following lines:

```
// Set up the environment for creating the initial context
Hashtable<String, Object> env = new Hashtable<String, Object>(11);
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL,
        "ldap://192.168.64.135:389/dc=jndi_dezsys");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=admin,dc=jndi_dezsys");
env.put(Context.SECURITY_CREDENTIALS, "admin");
```

Lookup an Object

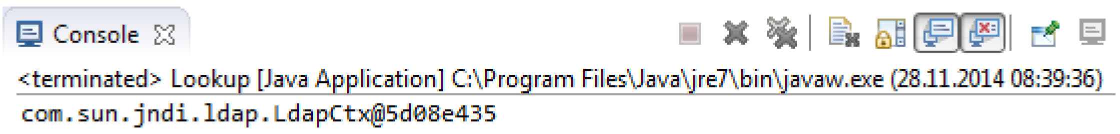
To look up an object we had to create the initial context at first and then we were able to perform a lookup. The following code looks up the object "`cn=Rosanna Lee,ou=People,o=jndi_dezsys`":

```
// Create the initial context
Context ctx = new InitialContext(env);

// Perform lookup and cast to target type
LdapContext b = (LdapContext) ctx
        .lookup("cn=Rosanna Lee,ou=People,o=jndi_dezsys");

System.out.println(b);

// Close the context when we're done
ctx.close();
```

cn ……. common name
ou …… People
o ……... jndi_dezsys

Result:

As a result we get a message from the LDAP-Server that the object is looked up:



```
<terminated> Lookup [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 08:39:36)
com.sun.jndi.ldap.LdapCtx@5d08e435
```

Otherwise we could get an error message in the console.

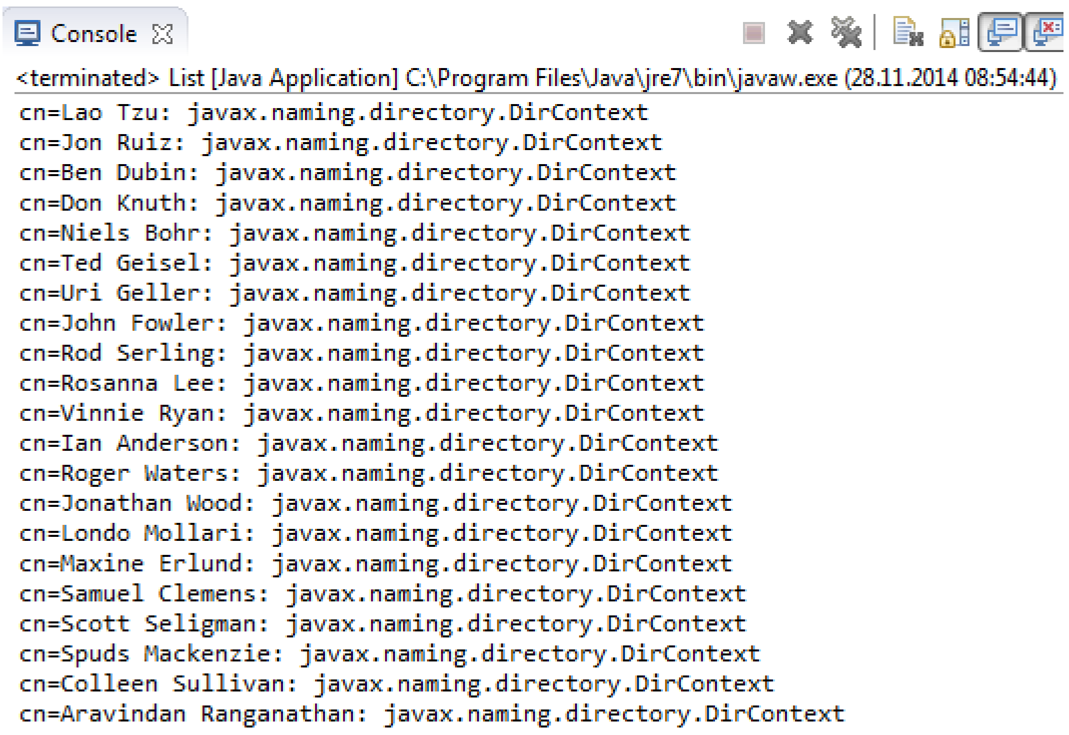
List the Context

We implemented two classes “`List`” and “`ListBindings`” to list the context. The class “`List`” lists the name and class of objects in a context and “`ListBindings`” lists the bindings in a context.

`ctx.list()` returns an enumeration of `NameClassPair`. Each `NameClassPair` consists of the object’s name and its class name. The following code fragment lists the contents of the “`ou=People,o=jndi_dezsys`” directory (i.e., the files and directories found in “`ou=People,o=jndi_dezsys`” directory):

```
// Create the initial context
Context ctx = new InitialContext(env);

// Get listing of context
NamingEnumeration list = ctx.list("ou=People,o=jndi_dezsys");

// Go through each item in list
while (list.hasMore()) {
    NameClassPair nc = (NameClassPair) list.next();
    System.out.println(nc);
}

// Close the context when we're done
ctx.close();
```

Result:

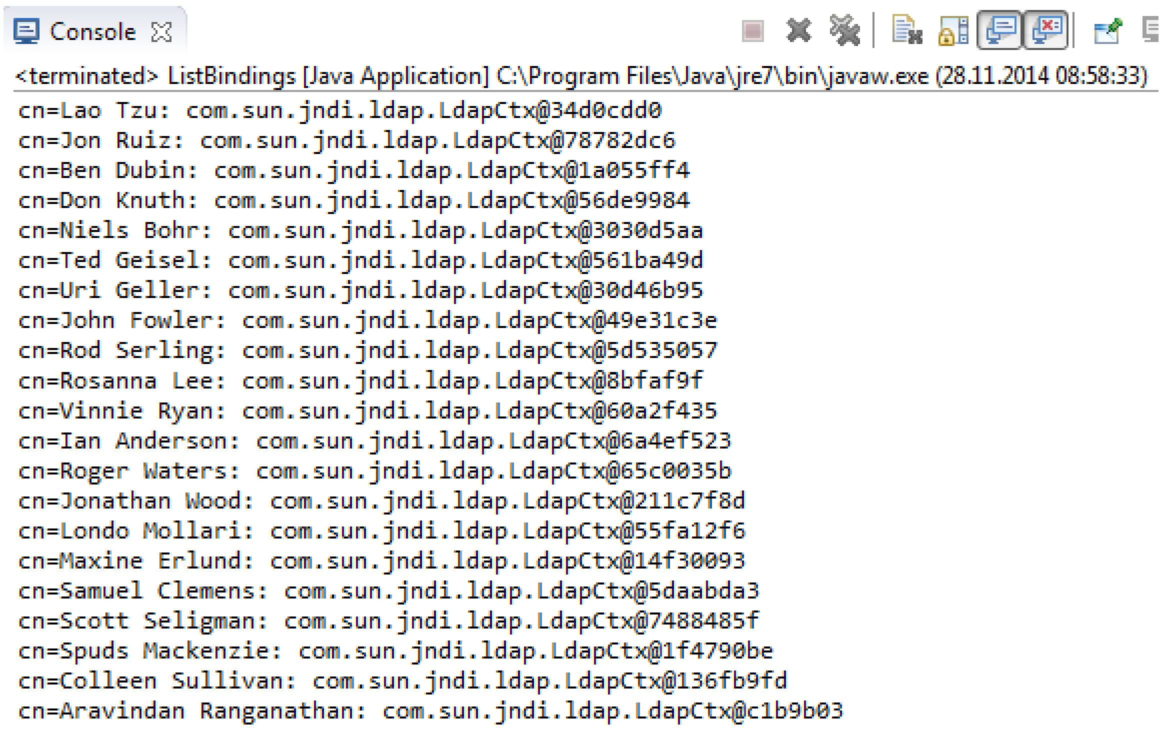


```
<terminated> List [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 08:54:44)
cn=Lao Tzu: javax.naming.directory.DirContext
cn=Jon Ruiz: javax.naming.directory.DirContext
cn=Ben Dubin: javax.naming.directory.DirContext
cn=Don Knuth: javax.naming.directory.DirContext
cn=Niels Bohr: javax.naming.directory.DirContext
cn=Ted Geisel: javax.naming.directory.DirContext
cn=Uri Geller: javax.naming.directory.DirContext
cn=John Fowler: javax.naming.directory.DirContext
cn=Rod Serling: javax.naming.directory.DirContext
cn=Rosanna Lee: javax.naming.directory.DirContext
cn=Vinnie Ryan: javax.naming.directory.DirContext
cn=Ian Anderson: javax.naming.directory.DirContext
cn=Roger Waters: javax.naming.directory.DirContext
cn=Jonathan Wood: javax.naming.directory.DirContext
cn=Londo Mollari: javax.naming.directory.DirContext
cn=Maxine Erlund: javax.naming.directory.DirContext
cn=Samuel Clemens: javax.naming.directory.DirContext
cn=Scott Seligman: javax.naming.directory.DirContext
cn=Spuds Mackenzie: javax.naming.directory.DirContext
cn=Colleen Sullivan: javax.naming.directory.DirContext
cn=Aravindan Ranganathan: javax.naming.directory.DirContext
```

`ctx.listBindings()` returns an enumeration of `Binding`. Binding is a subclass `NameClassPair`. A binding contains not only the object's name and class name, but also the object. The following code enumerates the "`ou=People,o=jndi_dezsys`" context, printing out each binding's name and object:

```
// Create the initial context
Context ctx = new InitialContext(env);

// Get listing of context
NamingEnumeration bindings = ctx
        .listBindings("ou=People,o=jndi_dezsys");

// Go through each item in list
while (bindings.hasMore()) {
    Binding bd = (Binding) bindings.next();
    System.out.println(bd.getName() + ": " + bd.getObject());
}

// Close the context when we're done
ctx.close();
```

Result:

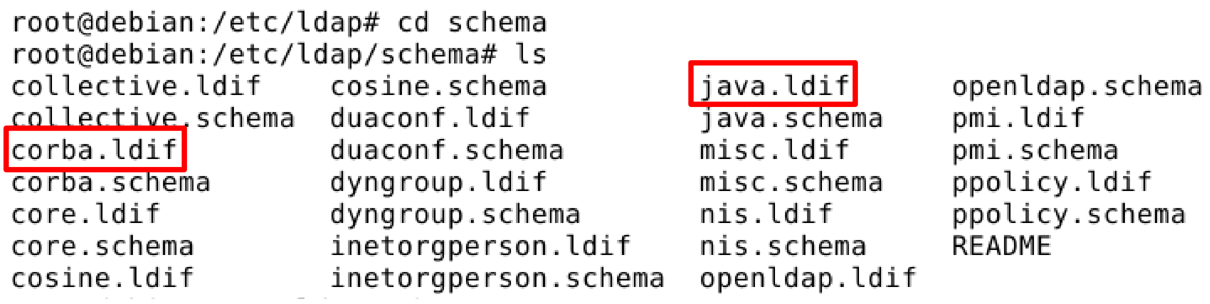


```
<terminated> ListBindings [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 08:58:33)
cn=Lao Tzu: com.sun.jndi.ldap.LdapCtx@34d0cdd0
cn=Jon Ruiz: com.sun.jndi.ldap.LdapCtx@78782dc6
cn=Ben Dubin: com.sun.jndi.ldap.LdapCtx@1a055ff4
cn=Don Knuth: com.sun.jndi.ldap.LdapCtx@56de9984
cn=Niels Bohr: com.sun.jndi.ldap.LdapCtx@3030d5aa
cn=Ted Geisel: com.sun.jndi.ldap.LdapCtx@561ba49d
cn=Uri Geller: com.sun.jndi.ldap.LdapCtx@30d46b95
cn=John Fowler: com.sun.jndi.ldap.LdapCtx@49e31c3e
cn=Rod Serling: com.sun.jndi.ldap.LdapCtx@5d535057
cn=Rosanna Lee: com.sun.jndi.ldap.LdapCtx@8bfaf9f
cn=Vinnie Ryan: com.sun.jndi.ldap.LdapCtx@60a2f435
cn=Ian Anderson: com.sun.jndi.ldap.LdapCtx@6a4ef523
cn=Roger Waters: com.sun.jndi.ldap.LdapCtx@65c0035b
cn=Jonathan Wood: com.sun.jndi.ldap.LdapCtx@211c7f8d
cn=Londo Mollari: com.sun.jndi.ldap.LdapCtx@55fa12f6
cn=Maxine Erlund: com.sun.jndi.ldap.LdapCtx@14f30093
cn=Samuel Clemens: com.sun.jndi.ldap.LdapCtx@5daabda3
cn=Scott Seligman: com.sun.jndi.ldap.LdapCtx@7488485f
cn=Spuds Mackenzie: com.sun.jndi.ldap.LdapCtx@1f4790be
cn=Colleen Sullivan: com.sun.jndi.ldap.LdapCtx@136fb9fd
cn=Aravindan Ranganathan: com.sun.jndi.ldap.LdapCtx@c1b9b03
```

Add, Replace or Remove a Binding

Before we were able to add, replace or remove a binding, we had to add the specified shemes by executing the following commands in the directory “`/etc/ldap/schema/`”:

```
# cd /etc/ldap/schema
# ldapadd –Y EXTERNAL –H ldapi:///-f java.ldif
# ldapadd –Y EXTERNAL –H ldapi:///-f corba.ldif
# ls
```

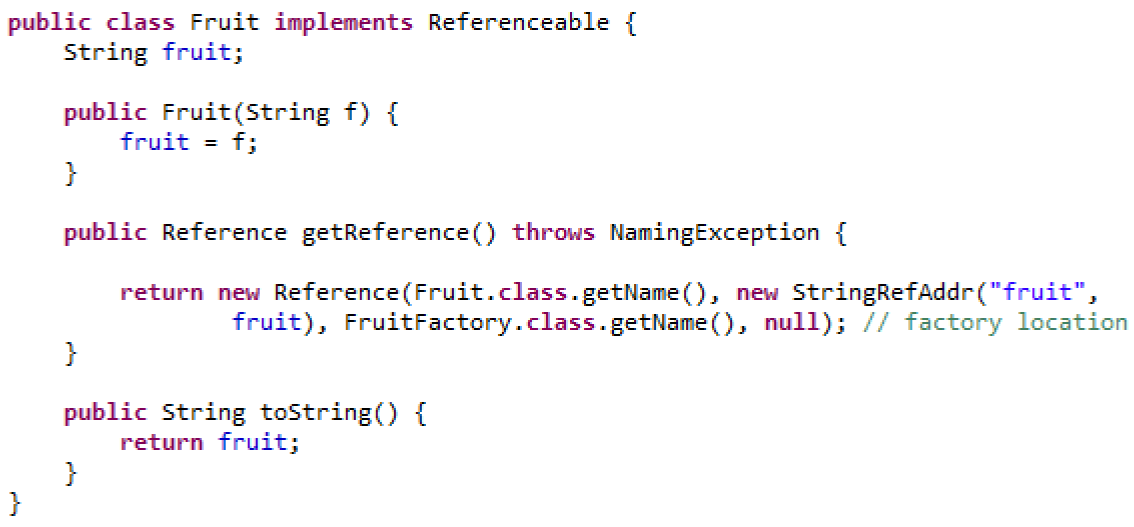


```
root@debian:/etc/ldap# cd schema
root@debian:/etc/ldap/schema# ls
collective.ldif     cosine.schema          java.ldif       openldap.schema
collective.schema   duaconf.ldif           java.schema     pmi.ldif
corba.ldif          duaconf.schema         misc.ldif       pmi.schema
corba.schema        dyngroup.ldif          misc.schema     ppolicy.ldif
core.ldif           dyngroup.schema        nis.ldif        ppolicy.schema
core.schema         inetorgperson.ldif     nis.schema      README
cosine.ldif         inetorgperson.schema   openldap.ldif
```

The following classes were necessary to create the objects to be bound:

Class “`Fruit.java`”:

```
This class is used by the Bind example. It is a referenceable class that can
be stored by service providers like the LDAP and file system providers.
```



```
public class Fruit implements Referenceable {
    String fruit;

    public Fruit(String f) {
        fruit = f;
    }

    public Reference getReference() throws NamingException {

        return new Reference(Fruit.class.getName(), new StringRefAddr("fruit",
                fruit), FruitFactory.class.getName(), null); // factory location
    }

    public String toString() {
        return fruit;
    }
}
```

Class “`FruitFactory.java`”:

```
This is an object factory that when given a reference for a Fruit object, will
create an instance of the corresponding Fruit.
```

```
public class FruitFactory implements ObjectFactory {

    public FruitFactory() {
    }

    public Object getObjectInstance(Object obj, Name name, Context ctx,
            Hashtable<?, ?> env) throws Exception {

        if (obj instanceof Reference) {
            Reference ref = (Reference) obj;

            if (ref.getClassName().equals(Fruit.class.getName())) {
                RefAddr addr = ref.get("fruit");
                if (addr != null) {
                    return new Fruit((String) addr.getContent());
                }
            }
        }
        return null;
    }
}
```

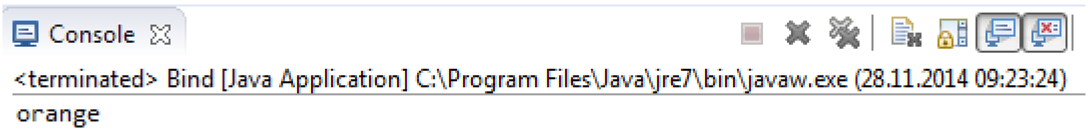
Now we are able to create the objects by using this classes.

The following code fragment creates an object of class `Fruit` and binds it to the name `“cn=Favorite Fruit,ou=People,o=jndi_dezsys”` in the context `ctx`. If you subsequently looked up the name `“cn=Favorite Fruit,ou=People,o=jndi_dezsys”` in `ctx`, then you would get the `fruit` object. Note that to compile the `Fruit` class, `FruitFactory` class is needed. By running this code twice, then the second attempt would fail with a `NameAlreadyBoundException`. This happens because the name `“cn=Favorite Fruit,ou=People,o=jndi_dezsys”` is already bound. For the second attempt to succeed, you would have to use `rebind()`.

```
// Create the initial context
Context ctx = new InitialContext(env);

// Create the object to be bound
Fruit fruit = new Fruit("orange");

// Perform bind
ctx.bind("cn=Favorite Fruit,ou=People,o=jndi_dezsys", fruit);

// Check that it is bound
Object obj = ctx
        .lookup("cn=Favorite Fruit,ou=People,o=jndi_dezsys");
System.out.println(obj);

// Close the context when we're done
ctx.close();
```

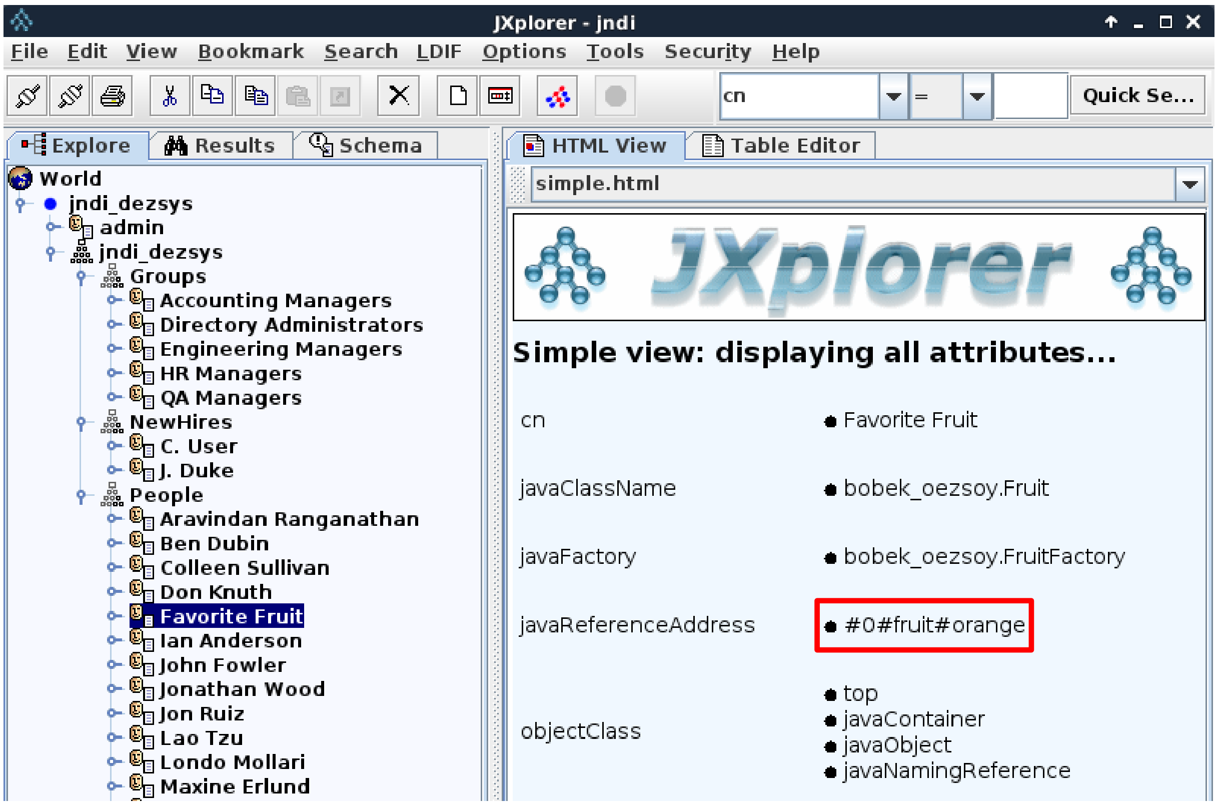
Result:

In the terminal we got the message:



```
<terminated> Bind [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 09:23:24)
orange
```

And if we take a look to the LDAP-Server, we can make sure that the object is bound successfully:



Simple view: displaying all attributes...

cn	Favorite Fruit
javaClassName	bobek_oezsoy.Fruit
javaFactory	bobek_oezsoy.FruitFactory
javaReferenceAddress	#0#fruit#orange
objectClass	top javaContainer javaObject javaNamingReference

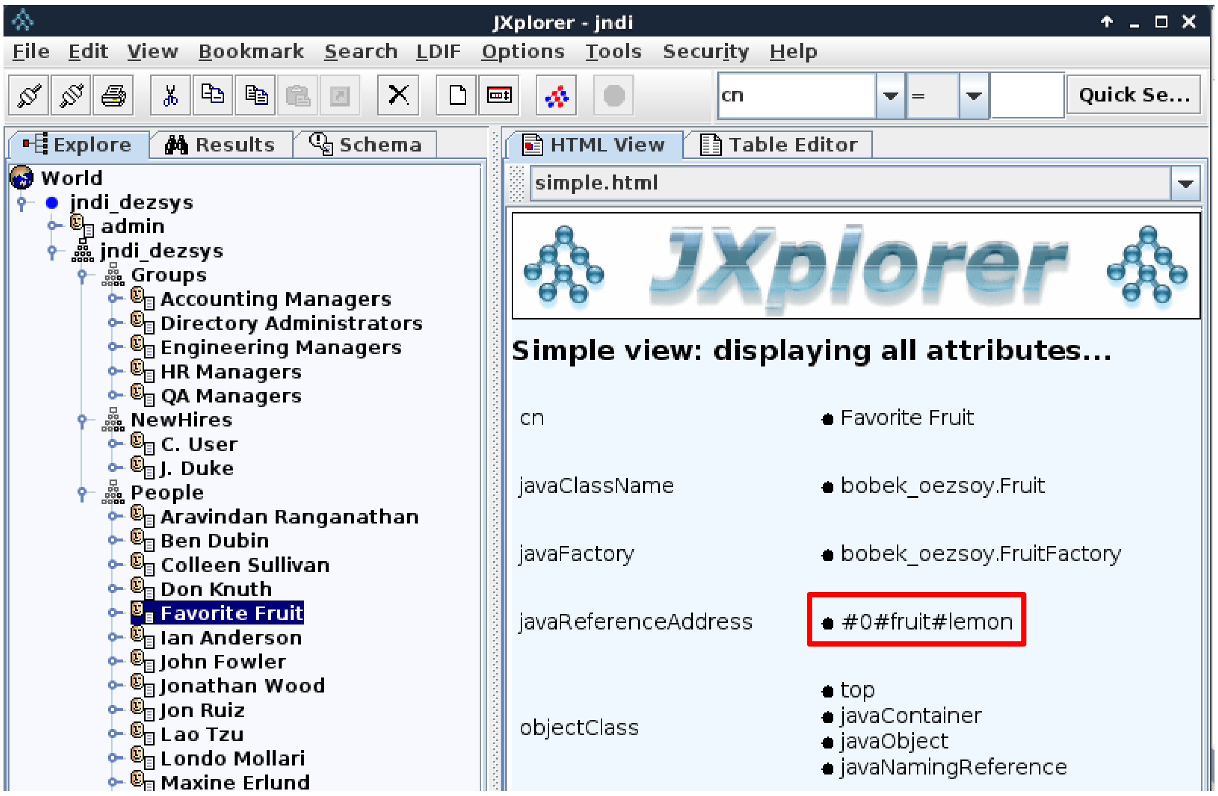
Now we are going to check the `rebind()`. It is used to add or replace a binding and accepts the same arguments as `bind()`, but the semantics are such that if the name is already bound, then it will be unbound and the newly given object will be bound.

The following code shows how to overwrite an existing binding:

```
// Create the initial context
Context ctx = new InitialContext(env);

// Create the object to be bound
Fruit fruit = new Fruit("lemon");

// Perform the bind
ctx.rebind("cn=Favorite Fruit,ou=People,o=jndi_dezsys", fruit);

// Check that it is bound
Object obj = ctx
        .lookup("cn=Favorite Fruit,ou=People,o=jndi_dezsys");
System.out.println(obj);

// Close the context when we're done
ctx.close();
```

Result:

By executing this code, the fruit "`orange`" in the name "`cn=Favorite Fruit,ou=People,o=jndi_dezsys`" is going to be replaced with the fruit "`lemon`":



Simple view: displaying all attributes...

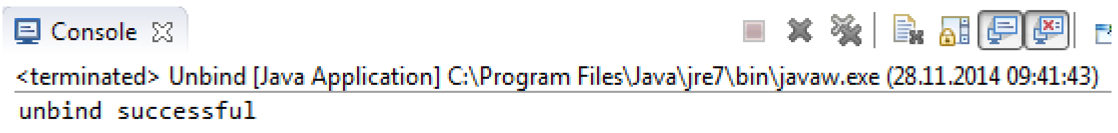
cn	● Favorite Fruit
javaClassName	● bobek_oezsoy.Fruit
javaFactory	● bobek_oezsoy.FruitFactory
javaReferenceAddress	● #0#fruit#lemon
objectClass	● top ● javaContainer ● javaObject ● javaNamingReference

To remove a binding, we have to use unbind() as follows:

```
// Create the initial context
Context ctx = new InitialContext(env);

// Remove the binding
ctx.unbind("cn=Favorite Fruit,ou=People,o=jndi_dezsys");

// Check that it is gone
Object obj = null;
try {
    obj = ctx.lookup("cn=Favorite Fruit,ou=People,o=jndi_dezsys");
} catch (NameNotFoundException ne) {
    System.out.println("unbind successful");
    return;
}

System.out.println("unbind failed; object still there: " + obj);

// Close the context when we're done
ctx.close();
```

Result:

We got the message “`unbind successful`” in the console and the entry “`cn=Favorite Fruit,ou=People,o=jndi_dezsys`” doesn’t exists anymore:



```
<terminated> Unbind [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 09:41:43)
unbind successful
```

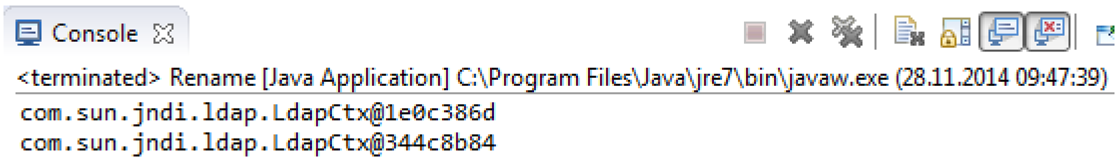
Rename

To rename an object in a context you have to use ctx.rename(). The following code renames the object that was bound to “cn=Scott Seligman,ou=People,o=jndi_dezsys” to “cn=Scott S,ou=People,o=jndi_dezsys”. After verifying that the object got renamed, the program renames it to its original name, as follows:

```
// Create the initial context
Context ctx = new InitialContext(env);

// Rename to Scott S
ctx.rename("cn=Scott Seligman,ou=People,o=jndi_dezsys",
        "cn=Scott S,ou=People,o=jndi_dezsys");

// Check that it is there using new name
Object obj = ctx.lookup("cn=Scott S,ou=People,o=jndi_dezsys");
System.out.println(obj);

// Rename back to Scott Seligman
ctx.rename("cn=Scott S,ou=People,o=jndi_dezsys",
        "cn=Scott Seligman,ou=People,o=jndi_dezsys");

// Check that it is there with original name
obj = ctx.lookup("cn=Scott Seligman,ou=People,o=jndi_dezsys");
System.out.println(obj);

// Close the context when we're done
ctx.close();
```

Result:

We got the following messages in the console:



```
<terminated> Rename [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 09:47:39)
com.sun.jndi.ldap.LdapCtx@1e0c386d
com.sun.jndi.ldap.LdapCtx@344c8b84
```

This messages mean that the rename-process is successfully finished, otherwise we could get an error-message.

Create and Destroy Subcontexts

Before going on with this point, we had to make sure that the shemes “java.ldif” and “corba.ldif” as LDIF-Files are added and installed (look at the description in “Add, Replace or Remove a Binding”).

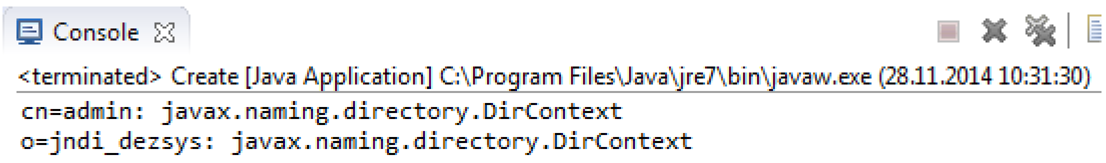
To create a naming context, we have to supply to `createSubcontext()` the name of the context that we want to create. To create a context that has attributes, we have to supply to DirContext.createSubcontext() the name of the context that we want to create and its attributes.

The following code creates a new context called “`ou=NewOu,o=jndi_dezsys`” that has an attribute “`objectclass`” with two values, “top” and “`organizationalUnit`”, in the context `ctx`:

```
// Create the initial context
DirContext ctx = new InitialDirContext(env);

// Create attributes to be associated with the new context
Attributes attrs = new BasicAttributes(true); // case-ignore
Attribute objclass = new BasicAttribute("objectclass");
objclass.add("top");
objclass.add("organizationalUnit");
attrs.put(objclass);

// Create the context
Context result = ctx.createSubcontext("ou=NewOu,o=jndi_dezsys",
        attrs);

// Check that it was created by listing its parent
NamingEnumeration list = ctx.list("");

// Go through each item in list
while (list.hasMore()) {
    NameClassPair nc = (NameClassPair) list.next();
    System.out.println(nc);
}

// Close the contexts when we're done
result.close();
ctx.close();
```

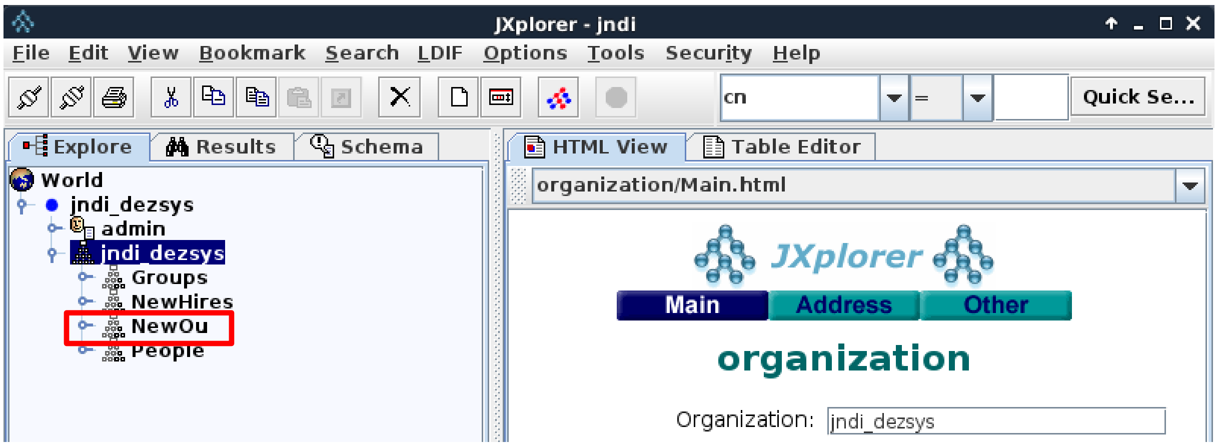
Result:

We got the following messages in the console:



```
<terminated> Create [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 10:31:30)
cn=admin: javax.naming.directory.DirContext
o=jndi_dezsys: javax.naming.directory.DirContext
```

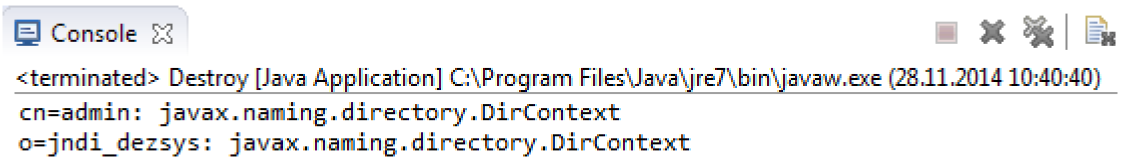
In the naming service, we can make sure that that specified context is created:



To destroy a context, we have to supply to `destroySubcontext()` the name of the context to destroy. The following code destroys the context "`ou=NewOu,o=jndi_dezsys`" in the context `ctx`:

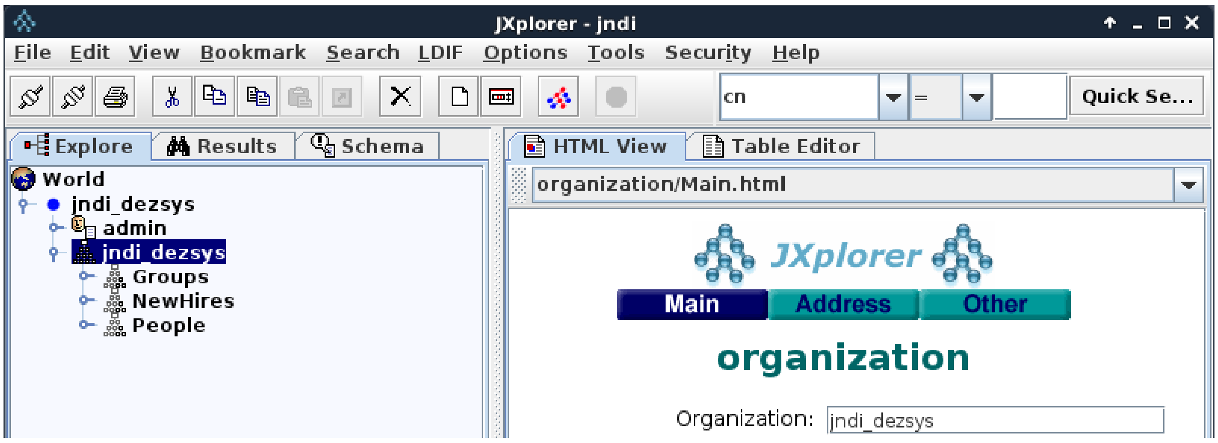
```
// Create the initial context
Context ctx = new InitialContext(env);

// Destroy the context
ctx.destroySubcontext("ou=NewOu,o=jndi_dezsys");

// Check that it has been destroyed by listing its parent
NamingEnumeration list = ctx.list("");

// Go through each item in list
while (list.hasMore()) {
    NameClassPair nc = (NameClassPair) list.next();
    System.out.println(nc);
}

// Close the context when we're done
ctx.close();
```

Result:



```
<terminated> Destroy [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 10:40:40)
cn=admin: javax.naming.directory.DirContext
o=jndi_dezsys: javax.naming.directory.DirContext
```

The created context doesn't exists in the naming service any more:



```
World
  jndi_dezsys
    admin
    jndi_dezsys
      Groups
      NewHires
      People
```

Organization: jndi_dezsys

Attribute Names

No implementation was necessary at this point. There were just information to read in the given tutorial.

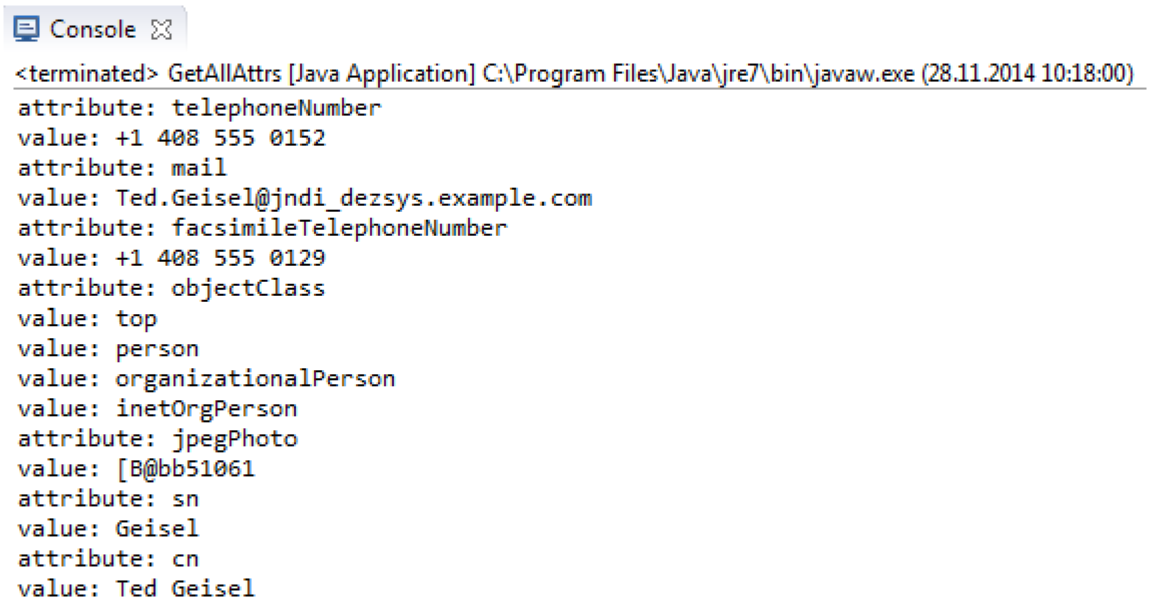
Read Attributes

To read the attributes of an object from the directory, we used `ctx.getAttributes()` and passed the name of the object for which we wanted the attributes. We made sure that an object in the naming service has the name "`cn=Ted Geisel,ou=People,o=jndi_dezsys`". To retrieve this object's attributes, we implemented the following code:

```
// Create the initial context
DirContext ctx = new InitialDirContext(env);

// Get all the attributes of named object
Attributes answer = ctx
        .getAttributes("cn=Ted Geisel,ou=People,o=jndi_dezsys");

// Print the answer
printAttrs(answer);

// Close the context when we're done
ctx.close();
```

```
static void printAttrs(Attributes attrs) {
    if (attrs == null) {
        System.out.println("No attributes");
    } else {
        /* Print each attribute */
        try {
            for (NamingEnumeration ae = attrs.getAll(); ae.hasMore();) {
                Attribute attr = (Attribute) ae.next();
                System.out.println("attribute: " + attr.getID());

                /* print each value */
                for (NamingEnumeration e = attr.getAll(); e.hasMore(); System.out
                        .println("value: " + e.next()))
                    ;
            }
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }
}
```

Result:



Console

```
<terminated> GetAllAttrs [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 10:18:00)
attribute: telephoneNumber
value: +1 408 555 0152
attribute: mail
value: Ted.Geisel@jndi_dezsys.example.com
attribute: facsimileTelephoneNumber
value: +1 408 555 0129
attribute: objectClass
value: top
value: person
value: organizationalPerson
value: inetOrgPerson
attribute: jpegPhoto
value: [B@bb51061
attribute: sn
value: Geisel
attribute: cn
value: Ted Geisel
```

Modify Attributes

One way to modify the attributes of an object is to supply a list of modification requests. Each `ModificationItem` consist of a numeric constant indicating the type of modification to make and an attribute describing the modification to make. Following are the three types of modifications:

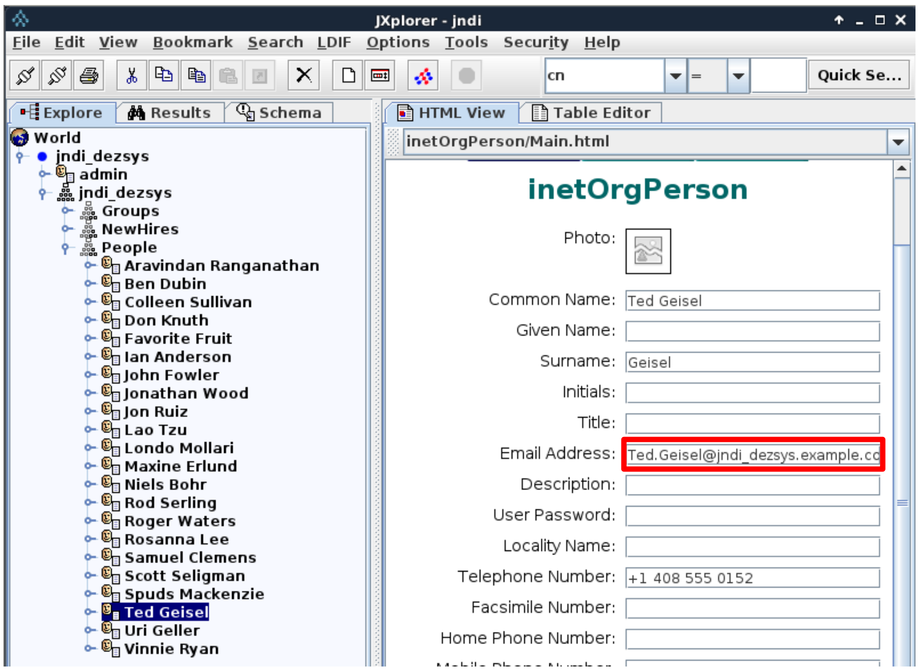
- ADD_ATTRIBUTE
- REPLACE_ATTRIBUTE
- REMOVE_ATTRIBUTE

Modifications are applied in the order in which they appear in the list. Either all the modifications are execited or none are.

The following code creates a new list of modifications. It replaces the "`mail`" attribute's value with a value of "`Ted.Geisel@jndi_dezsys.example.com`":

```
// Create the initial context
DirContext ctx = new InitialDirContext(env);

// Specify the changes to make
ModificationItem[] mods = new ModificationItem[1];

// Replace the "mail" attribute with a new value
mods[0] = new ModificationItem(DirContext.REPLACE_ATTRIBUTE,
        new BasicAttribute("mail",
                "Ted.Geisel@jndi_dezsys.example.com"));

// Perform the requested modifications on the named object
ctx.modifyAttributes("cn=Ted Geisel,ou=People,o=jndi_dezsys", mods);

// Close the context when we're done
ctx.close();
```

Result:



```
World
  jndi_dezsys
    admin
    jndi_dezsys
      Groups
      NewHires
      People
        Aravindan Ranganathan
        Ben Dubin
        Colleen Sullivan
        Don Knuth
        Favorite Fruit
        Ian Anderson
        John Fowler
        Jonathan Wood
        Jon Ruiz
        Lao Tzu
        Londo Mollari
        Maxine Erlund
        Niels Bohr
        Rod Serling
        Roger Waters
        Rosanna Lee
        Samuel Clemens
        Scott Seligman
        Spuds Mackenzie
        Ted Geisel
        Uri Geller
        Vinnie Ryan
```

inetOrgPerson/Main.html

inetOrgPerson

Photo:	
Common Name:	Ted Geisel
Given Name:	
Surname:	Geisel
Initials:	
Title:	
Email Address:	Ted.Geisel@jndi_dezsys.example.co
Description:	
User Password:	
Locality Name:	
Telephone Number:	+1 408 555 0152
Facsimile Number:	
Home Phone Number:	

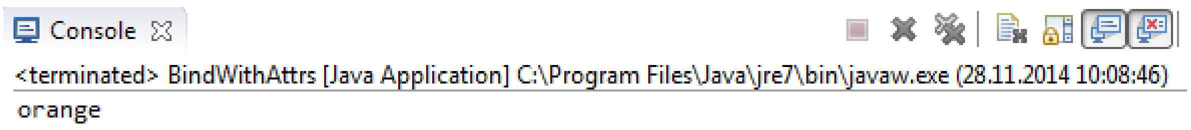
Add, Replace Bindings with Attributes

The adding and replacing of bindings with attributes requires the same things as above descripted, but we have to use `DirContext.bind()` or `DirContext.rebind()` to add or replace a binding that has attributes.

Adding a binding with attributes:

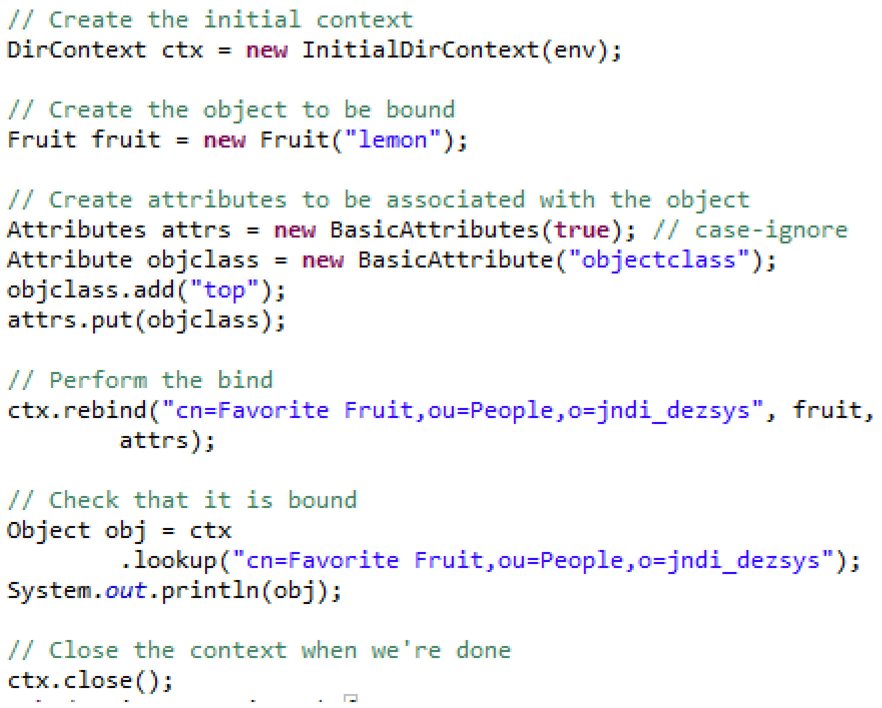
```
// Create the initial context
DirContext ctx = new InitialDirContext(env);

// Create the object to be bound
Fruit fruit = new Fruit("orange");

// Create attributes to be associated with the object
Attributes attrs = new BasicAttributes(true); // case-ignore
Attribute objclass = new BasicAttribute("objectclass");
objclass.add("top");
attrs.put(objclass);

// Perform bind
ctx.bind("cn=Favorite Fruit,ou=People,o=jndi_dezsys", fruit, attrs);

// Check that it is bound
Object obj = ctx
        .lookup("cn=Favorite Fruit,ou=People,o=jndi_dezsys");
System.out.println(obj);

// Close the context when we're done
ctx.close();
```

Result:



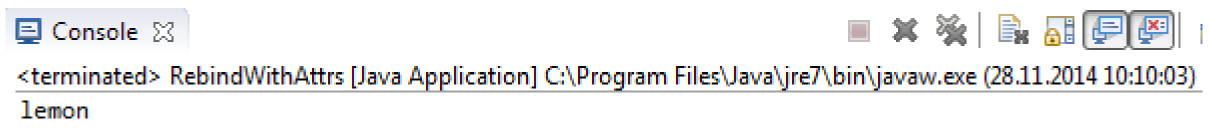
```
<terminated> BindWithAttrs [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 10:08:46)
orange
```

Rebinding an existing binding with attributes:



```
// Create the initial context
DirContext ctx = new InitialDirContext(env);

// Create the object to be bound
Fruit fruit = new Fruit("lemon");

// Create attributes to be associated with the object
Attributes attrs = new BasicAttributes(true); // case-ignore
Attribute objclass = new BasicAttribute("objectclass");
objclass.add("top");
attrs.put(objclass);

// Perform the bind
ctx.rebind("cn=Favorite Fruit,ou=People,o=jndi_dezsys", fruit,
        attrs);

// Check that it is bound
Object obj = ctx
        .lookup("cn=Favorite Fruit,ou=People,o=jndi_dezsys");
System.out.println(obj);

// Close the context when we're done
ctx.close();
```

Result:

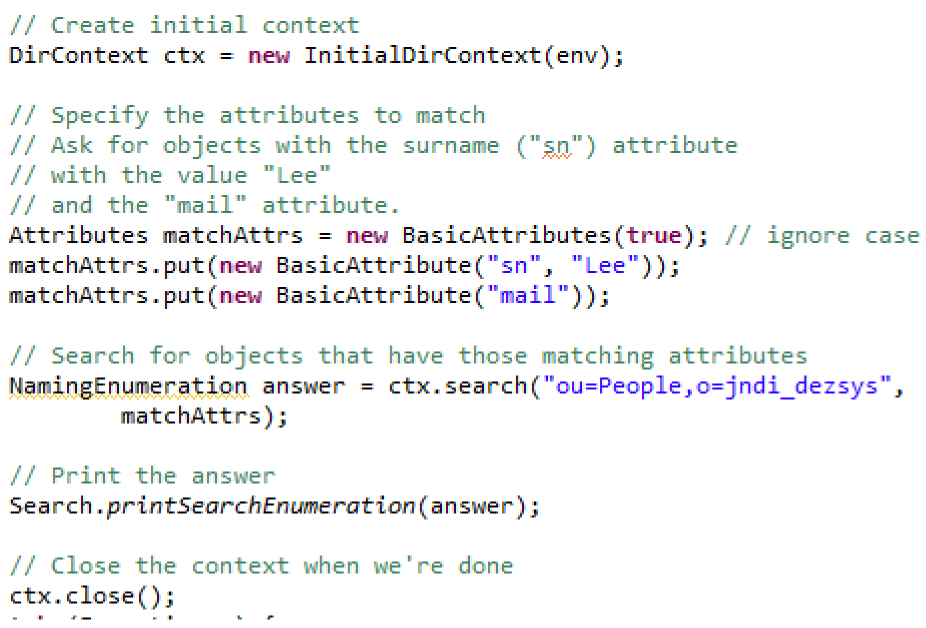


```
<terminated> RebindWithAttrs [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 10:10:03)
lemon
```

Basic Search

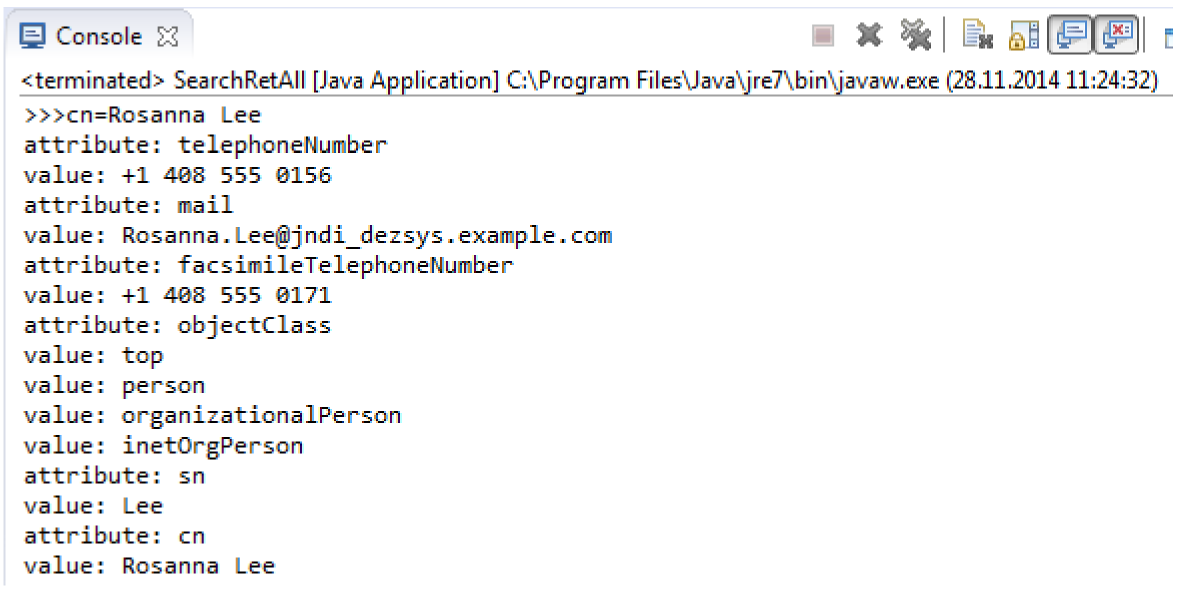
The simplest form of search requires that you specify the set of attributes that an entry must have and the name of the target context in which to perform the search.

The following code creates an attribute set `matchAttrs`, which has two attributes "`sn`" and "`mail`". It specifies that the qualifying entries must have a surname ("`sn`") attribute with a value of "`Lee`" and a "`mail`" attribute with any value. It then invokes `DirContext.search()` to search the context "`ou=People,o=jndi_dezsys`" for entries that have the attributes specified by `matchAttrs`:



```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Specify the attributes to match
// Ask for objects with the surname ("sn") attribute
// with the value "Lee"
// and the "mail" attribute.
Attributes matchAttrs = new BasicAttributes(true); // ignore case
matchAttrs.put(new BasicAttribute("sn", "Lee"));
matchAttrs.put(new BasicAttribute("mail"));

// Search for objects that have those matching attributes
NamingEnumeration answer = ctx.search("ou=People,o=jndi_dezsys",
        matchAttrs);

// Print the answer
Search.printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

Result:



```
<terminated> SearchRetAll [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 11:24:32)
>>>cn=Rosanna Lee
attribute: telephoneNumber
value: +1 408 555 0156
attribute: mail
value: Rosanna.Lee@jndi_dezsys.example.com
attribute: facsimileTelephoneNumber
value: +1 408 555 0171
attribute: objectClass
value: top
value: person
value: organizationalPerson
value: inetOrgPerson
attribute: sn
value: Lee
attribute: cn
value: Rosanna Lee
```

The following code returns the attributes "`sn`", "`telephonenumber`", "`golfhandicap`", and "`mail`" of entries that have an attribute "`mail`" and have a "`sn`" attribute with the value "`Geisel`":

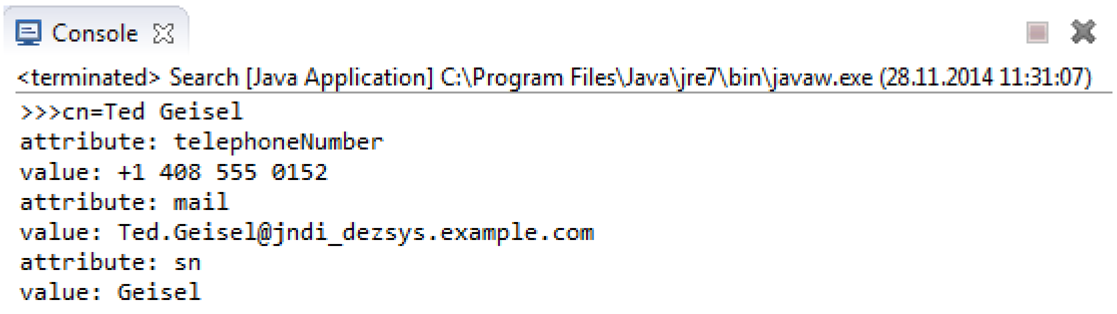
```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Specify the ids of the attributes to return
String[] attrIDs = { "sn", "telephonenumber", "golfhandicap",
        "mail" };

// Specify the attributes to match
// Ask for objects that have the attribute
// sn == Geisel and the "mail" attribute.
Attributes matchAttrs = new BasicAttributes(true); // ignore case
matchAttrs.put(new BasicAttribute("sn", "Geisel"));
matchAttrs.put(new BasicAttribute("mail"));

// Search for objects that have those matching attributes
NamingEnumeration answer = ctx.search("ou=People,o=jndi_dezsys", matchAttrs,
        attrIDs);

// Print the answer
printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

Method for printing out the answer:

```
public static void printSearchEnumeration(NamingEnumeration retEnum) {
    try {
        while (retEnum.hasMore()) {
            SearchResult sr = (SearchResult) retEnum.next();
            System.out.println(">>>" + sr.getName());
            GetAllAttrs.printAttrs(sr.getAttributes());
        }
    } catch (NamingException e) {
        e.printStackTrace();
    }
}
```

Result:

This example produces the following result (The entry does not have a "`golfhandicap`" attribute, so it is not returned):



```
<terminated> Search [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 11:31:07)
>>>cn=Ted Geisel
attribute: telephoneNumber
value: +1 408 555 0152
attribute: mail
value: Ted.Geisel@jndi_dezsys.example.com
attribute: sn
value: Geisel
```

Filters

A search filter is a search query expressed in the form of a logical expression. The following search filter specifies that the qualifying entries must have an "`sn`" attribute with a value of "`Lee`" and a "`mail`" attribute with any value:

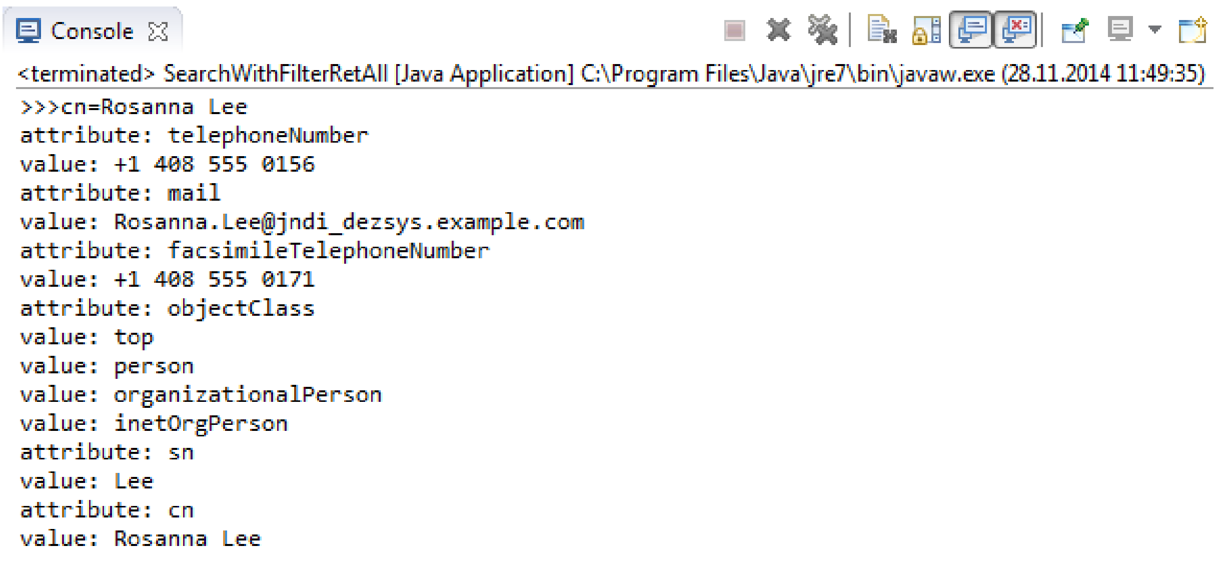
```
(&(sn=Lee)(mail=*))
```

The following code creates a filter and default `SearchControls`, and uses them to perform a search:

```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Create default search controls
SearchControls ctls = new SearchControls();

// Specify the search filter to match
// Ask for objects with attribute sn == Lee and which have
// the "mail" attribute.
String filter = "(&(sn=Lee)(mail=*))";

// Search for objects using filter
NamingEnumeration answer = ctx.search("ou=People,o=jndi_dezsys",
        filter, ctls);

// Print the answer
Search.printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

The search is equivalent to the one presented in the basic search example.

Result:

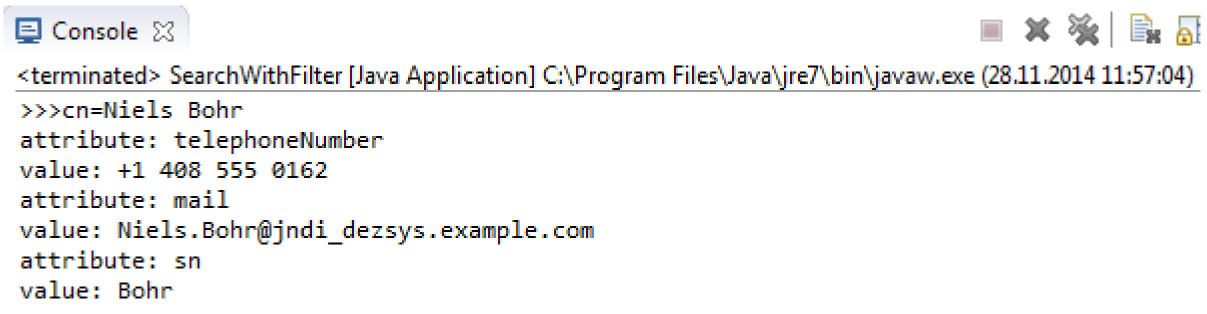


```
<terminated> SearchWithFilterRetAll [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 11:49:35)
>>>cn=Rosanna Lee
attribute: telephoneNumber
value: +1 408 555 0156
attribute: mail
value: Rosanna.Lee@jndi_dezsys.example.com
attribute: facsimileTelephoneNumber
value: +1 408 555 0171
attribute: objectClass
value: top
value: person
value: organizationalPerson
value: inetOrgPerson
attribute: sn
value: Lee
attribute: cn
value: Rosanna Lee
```

The previous example returned all attributes associated with the entries that satisfy the specified filter. You can select the attributes to return by setting the search controls argument. You create an array of attribute identifiers that you want to include in the result and pass it to `SearchControls.setReturningAttributes()`:

```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Specify the ids of the attributes to return
String[] attrIDs = { "sn", "telephonenumber", "golfhandicap",
        "mail" };
SearchControls ctls = new SearchControls();
ctls.setReturningAttributes(attrIDs);

// Specify the search filter to match
// Ask for objects with attribute sn == Bohr and which have
// the "mail" attribute.
String filter = "(&(sn=Bohr)(mail=*))";

// Search for objects using filter
NamingEnumeration answer = ctx.search("ou=People,o=jndi_dezsys", filter, ctls);

// Print the answer
Search.printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

Result:



```
<terminated> SearchWithFilter [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 11:57:04)
>>>cn=Niels Bohr
attribute: telephoneNumber
value: +1 408 555 0162
attribute: mail
value: Niels.Bohr@jndi_dezsys.example.com
attribute: sn
value: Bohr
```

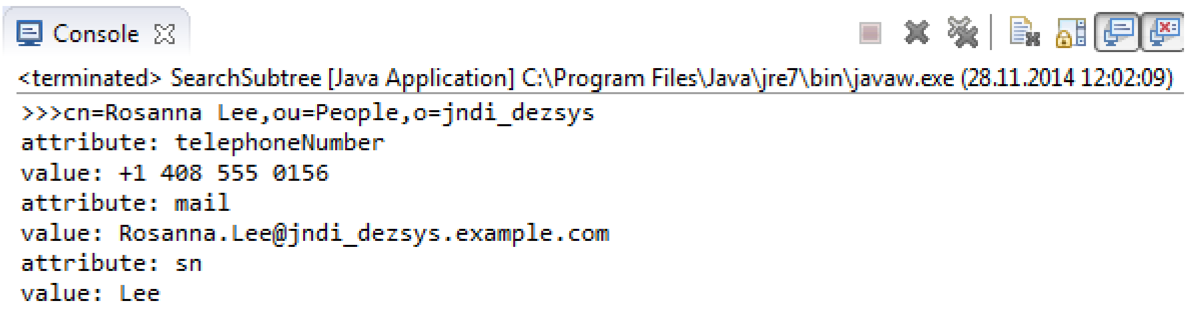
Scope

A search of the entire subtree searches the named object and all of its descendants. To make the search behave in this way, pass `SearchControls.SUBTREE_SCOPE` to `SearchControls.setSearchScope()` as follows:

```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Specify the ids of the attributes to return
String[] attrIDs = { "sn", "telephonenumber", "golfhandicap",
        "mail" };
SearchControls ctls = new SearchControls();
ctls.setReturningAttributes(attrIDs);
ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);

// Specify the search filter to match
// Ask for objects with attribute sn == Lee and which have
// the "mail" attribute.
String filter = "(&(sn=Lee)(mail=*))";

// Search subtree for objects using filter
NamingEnumeration answer = ctx.search("", filter, ctls);

// Print the answer
Search.printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

The code above searches the context `ctx`'s subtree for entries that satisfy the specified filter. It finds the entry "`cn= Rosanna Lee, ou=People,o=jndi_dezsys`" in this subtree that satisfies the filter.

Result:

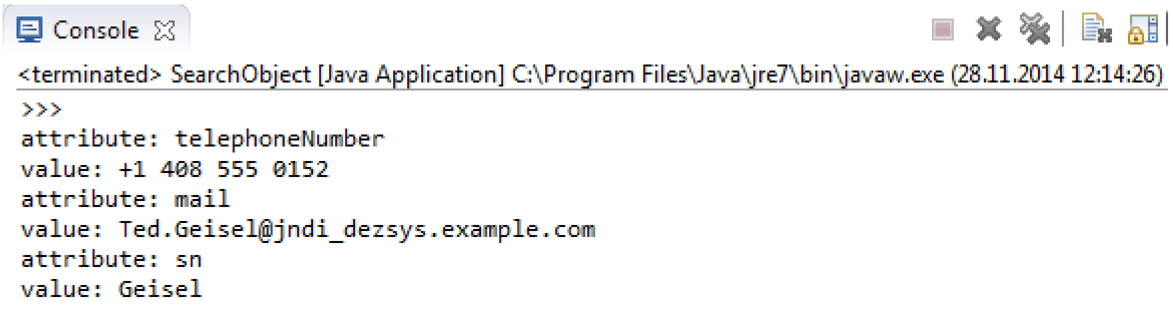


```
<terminated> SearchSubtree [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 12:02:09)
>>>cn=Rosanna Lee,ou=People,o=jndi_dezsys
attribute: telephoneNumber
value: +1 408 555 0156
attribute: mail
value: Rosanna.Lee@jndi_dezsys.example.com
attribute: sn
value: Lee
```

The following code tests whether the object “`cn=Ted Geisel,ou=People,o=jndi_dezsys`” satisfies the given filter:

```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Specify the ids of the attributes to return
String[] attrIDs = { "sn", "telephonenumber", "golfhandicap",
        "mail" };
SearchControls ctls = new SearchControls();
ctls.setReturningAttributes(attrIDs);
ctls.setSearchScope(SearchControls.OBJECT_SCOPE);

// Specify the search filter to match
// Ask for objects with attribute sn == Geisel and which have
// the "mail" attribute.
String filter = "(&(sn=Geisel)(mail=*))";

// Search subtree for objects using filter
NamingEnumeration answer = ctx.search("cn=Ted Geisel,ou=People,o=jndi_dezsys",
        filter, ctls);

// Print the answer
Search.printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

Result:



```
<terminated> SearchObject [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 12:14:26)
>>>
attribute: telephoneNumber
value: +1 408 555 0152
attribute: mail
value: Ted.Geisel@jndi_dezsys.example.com
attribute: sn
value: Geisel
```

Result Count

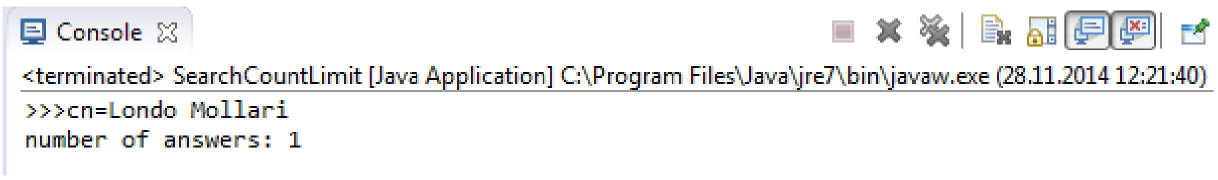
Sometimes, a query might produce too many answers and you want to limit the number of answers returned. You can do this by using the count limit search control. By default, a search does not have a count limit--it will return all answers that it finds. To set the count limit of a search, pass the number to `SearchControls.setCountLimit()`. The following code sets the count limit to 1 (the global attribute “`expected`” is set to 1):

```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Set search controls to limit count to 'expected'
SearchControls ctls = new SearchControls();
ctls.setCountLimit(expected);

// Search for objects with those matching attributes
NamingEnumeration answer = ctx.search("ou=People,o=jndi_dezsys",
        "(sn=M*)", ctls);

// Print the answer
printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

Method for printing out the answer:

```
static int expected = 1;

public static void printSearchEnumeration(NamingEnumeration srhEnum) {
    int count = 0;
    try {
        while (srhEnum.hasMore()) {
            SearchResult sr = (SearchResult) srhEnum.next();
            System.out.println(">>>" + sr.getName());
            ++count;
        }
        System.out.println("number of answers: " + count);
    } catch (SizeLimitExceededException e) {
        if (count == expected)
            System.out.println("number of answers: " + count);
        else
            e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    }
}
```

Result:



```
<terminated> SearchCountLimit [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 12:21:40)
>>>cn=Londo Mollari
number of answers: 1
```

Time Limit

A time limit on a search places an upper bound on the amount of time that the search operation will block waiting for the answers. This is useful when you don't want to wait too long for an answer. If the time limit specified is exceeded before the search operation can be completed, then a `TimeLimitExceededException` will be thrown.

To set the time limit of a search, the number of milliseconds to `SearchControls.setTimeLimit()` should be passed. The following code sets the time limit to 1 second:

```
// Create initial context
DirContext ctx = new InitialDirContext(env);

// Set search controls to limit count to 'timeout'
SearchControls ctls = new SearchControls();
ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);
ctls.setTimeLimit(timeout); //

// Search for objects with those matching attributes
NamingEnumeration answer = ctx.search("", "(objectclass=*)", ctls);

// Print the answer
printSearchEnumeration(answer);

// Close the context when we're done
ctx.close();
```

To get this particular example to exceed its time limit, you need to reconfigure it to use either a slow server, or a server that has lots of entries. Alternatively, you can use other tactics to make the search take longer than 1 second.

A time limit of zero means that no time limit has been set and that calls to the directory will wait indefinitely for an answer.

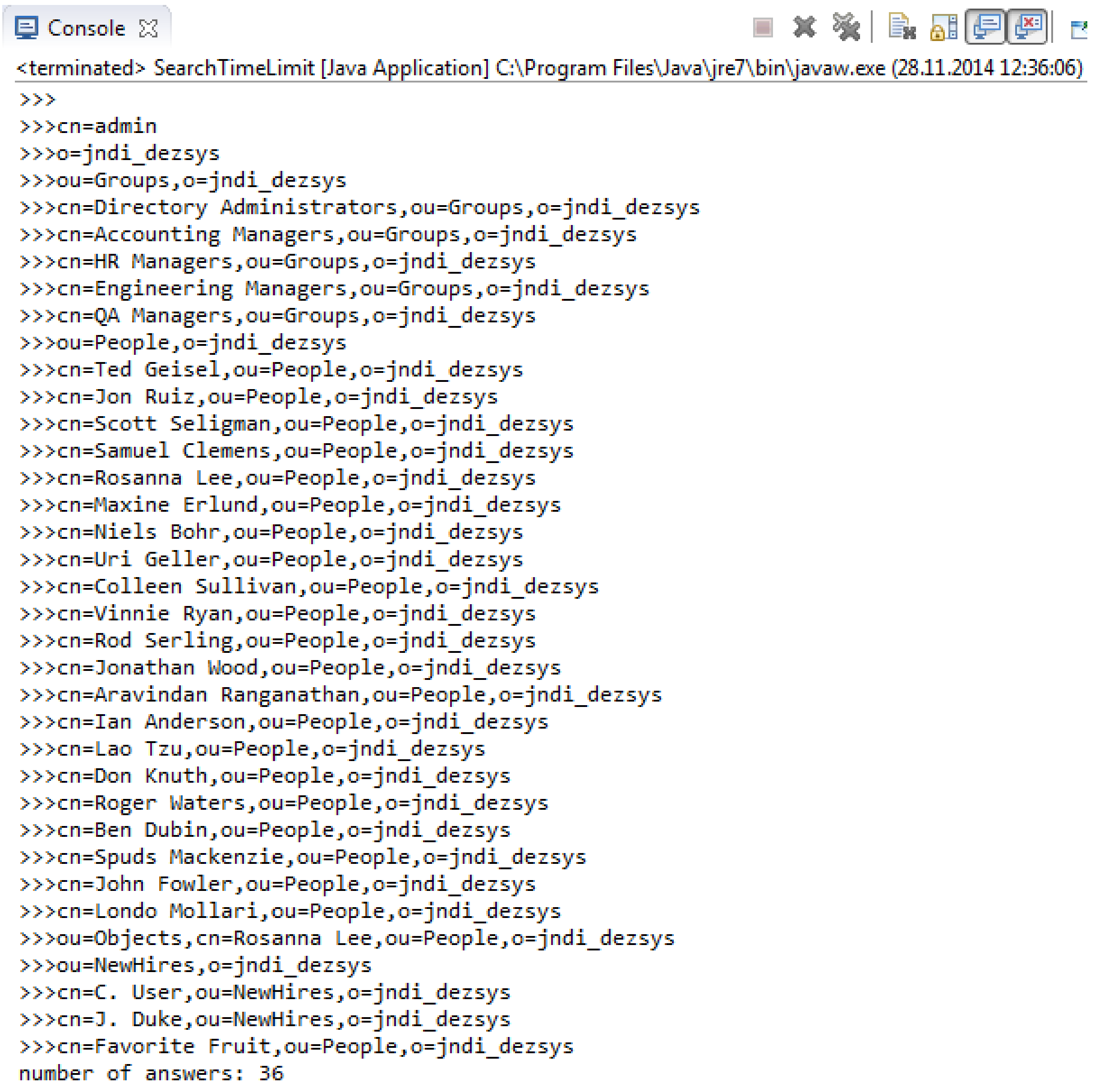
The value of the attribute “timeout” is set to:

```
static int timeout = 1000; // 1 second == 1000 ms
```

Method for printing out the answer:

```
public static void printSearchEnumeration(NamingEnumeration srhEnum) {
    int count = 0;
    try {
        while (srhEnum.hasMore()) {
            SearchResult sr = (SearchResult) srhEnum.next();
            System.out.println(">>>" + sr.getName());
            ++count;
        }
        System.out.println("number of answers: " + count);
    } catch (TimeLimitExceededException e) {
        System.out.println("search took more than " + timeout + "ms");
    } catch (NamingException e) {
        e.printStackTrace();
    }
}
```

Result:


Console

```
<terminated> SearchTimeLimit [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28.11.2014 12:36:06)
>>>
>>>cn=admin
>>>o=jndi_dezsys
>>>ou=Groups,o=jndi_dezsys
>>>cn=Directory Administrators,ou=Groups,o=jndi_dezsys
>>>cn=Accounting Managers,ou=Groups,o=jndi_dezsys
>>>cn=HR Managers,ou=Groups,o=jndi_dezsys
>>>cn=Engineering Managers,ou=Groups,o=jndi_dezsys
>>>cn=QA Managers,ou=Groups,o=jndi_dezsys
>>>ou=People,o=jndi_dezsys
>>>cn=Ted Geisel,ou=People,o=jndi_dezsys
>>>cn=Jon Ruiz,ou=People,o=jndi_dezsys
>>>cn=Scott Seligman,ou=People,o=jndi_dezsys
>>>cn=Samuel Clemens,ou=People,o=jndi_dezsys
>>>cn=Rosanna Lee,ou=People,o=jndi_dezsys
>>>cn=Maxine Erlund,ou=People,o=jndi_dezsys
>>>cn=Niels Bohr,ou=People,o=jndi_dezsys
>>>cn=Uri Geller,ou=People,o=jndi_dezsys
>>>cn=Colleen Sullivan,ou=People,o=jndi_dezsys
>>>cn=Vinnie Ryan,ou=People,o=jndi_dezsys
>>>cn=Rod Serling,ou=People,o=jndi_dezsys
>>>cn=Jonathan Wood,ou=People,o=jndi_dezsys
>>>cn=Aravindan Ranganathan,ou=People,o=jndi_dezsys
>>>cn=Ian Anderson,ou=People,o=jndi_dezsys
>>>cn=Lao Tzu,ou=People,o=jndi_dezsys
>>>cn=Don Knuth,ou=People,o=jndi_dezsys
>>>cn=Roger Waters,ou=People,o=jndi_dezsys
>>>cn=Ben Dubin,ou=People,o=jndi_dezsys
>>>cn=Spuds Mackenzie,ou=People,o=jndi_dezsys
>>>cn=John Fowler,ou=People,o=jndi_dezsys
>>>cn=Londo Mollari,ou=People,o=jndi_dezsys
>>>ou=Objects,cn=Rosanna Lee,ou=People,o=jndi_dezsys
>>>ou=NewHires,o=jndi_dezsys
>>>cn=C. User,ou=NewHires,o=jndi_dezsys
>>>cn=J. Duke,ou=NewHires,o=jndi_dezsys
>>>cn=Favorite Fruit,ou=People,o=jndi_dezsys
number of answers: 36
```

Bibliography

[1] Title: Trail: Java Naming and Directory Interface: Table of Contents
Author: Oracle
Online-/Resource: https://docs.oracle.com/javase/tutorial/jndi/TOC.html
last modified: /
abstracted: 11/25/2014

[2] Title: New York University Computer Science Department Courant Institute of Mathematical Sciences
Author: Jean-Claude Franchitti
Online-/Resource: http://www.nyu.edu/classes/jcf/g22.3033-007_sp01/handouts/g22_3033_h83.htm
last modified:
abstracted:

[3] Title: OpenLDAP installation on Debian
Author: docelic
Online-/Resource: https://www.debian-administration.org/article/585/OpenLDAP_installation_on_Debian
last modified: 03/19/2008
abstracted: 11/19/2014

[4] Title: Database Creation and Maintenance Tools
Author: OpenLDAP
Online-/Resource: http://www.openldap.org/doc/admin24/dbtools.html
last modified: /
abstracted: 11/19/2014