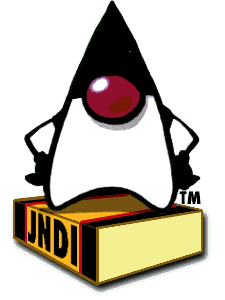
|  |  |
| --- | --- |
|  | Documentation |
|  | Class: 5AHITT  Last modification: 11/27/2014  Document Version: 1.0 |

Dezentrale Systeme

2014/2015

Christian Bobek, Osman Özsoy

Java Naming and Directory Interface



List of contents

[Statement of task 1](#_Toc404941584)

[Apportionment of work with effort estimation 2](#_Toc404941585)

[Final time apportionment 3](#_Toc404941586)

[Task execution 4](#_Toc404941587)

[Setting up a new naming service – OpenLDAP-Installation 4](#_Toc404941588)

[OpenLDAP-Installation towards Prof. Borko’s description 7](#_Toc404941589)

[Necessary LDAP-Setup/-Configuration for the tutorial after LDAP-Installation 9](#_Toc404941590)

[Directory Schema 9](#_Toc404941591)

[Providing directory content for this tutorial 9](#_Toc404941592)

[Implementing the Naming and Directory Operations 11](#_Toc404941593)

[Lookup an Object 12](#_Toc404941594)

[List the Context 13](#_Toc404941595)

[Add, Replace or Remove a Binding 15](#_Toc404941596)

[Rename 19](#_Toc404941597)

[Create and Destroy Subcontexts 20](#_Toc404941598)

[Attribute Names 22](#_Toc404941599)

[Read Attributes 22](#_Toc404941600)

[Modify Attributes 24](#_Toc404941601)

[Add, Replace Bindings with Attributes 25](#_Toc404941602)

[Basic Search 27](#_Toc404941603)

[Filters 29](#_Toc404941604)

[Scope 31](#_Toc404941605)

[Result Count 33](#_Toc404941606)

[Time Limit 34](#_Toc404941607)

[Bibliography 36](#_Toc404941608)

# 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

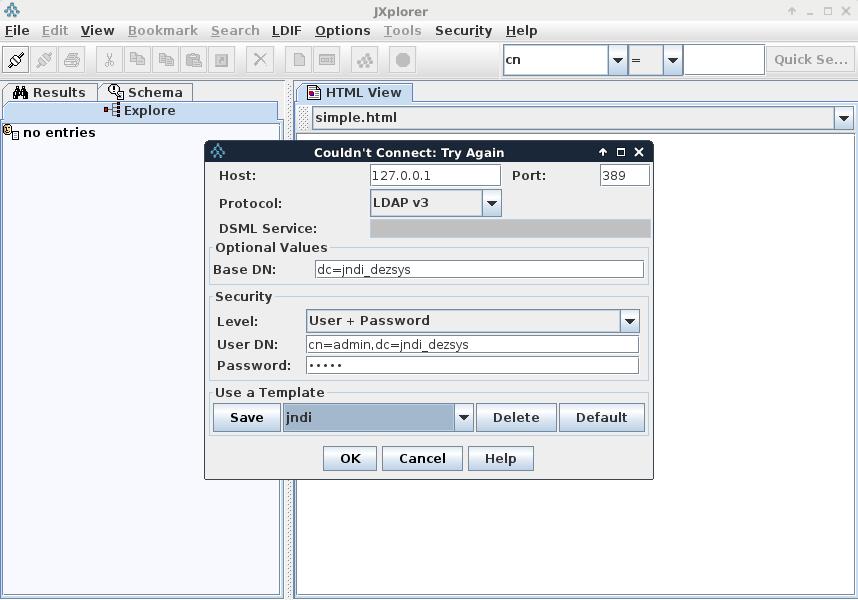
After finishing the modification start the server again with:

# 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.

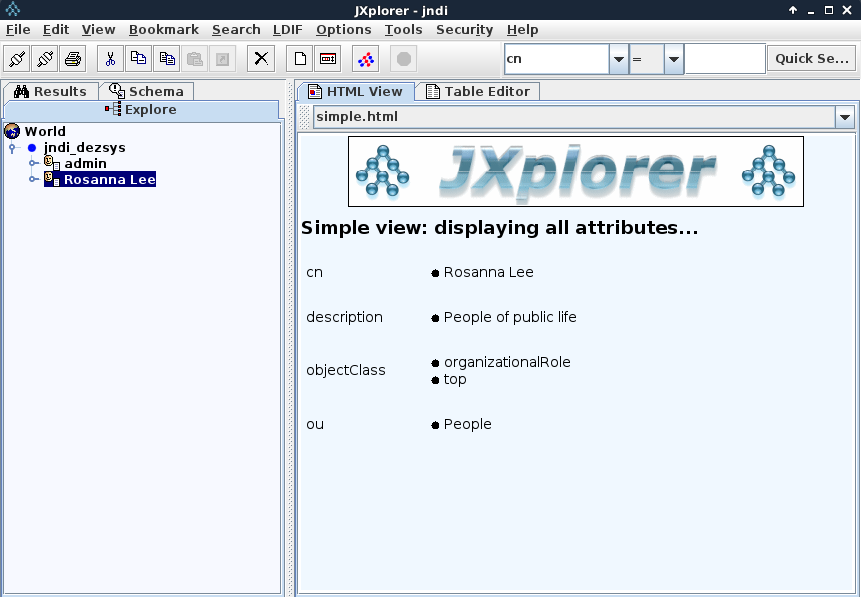
Once you have filled in the information, you can save it as a **template** and use this saved template when reconnecting to the same database.

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



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**:



## 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:

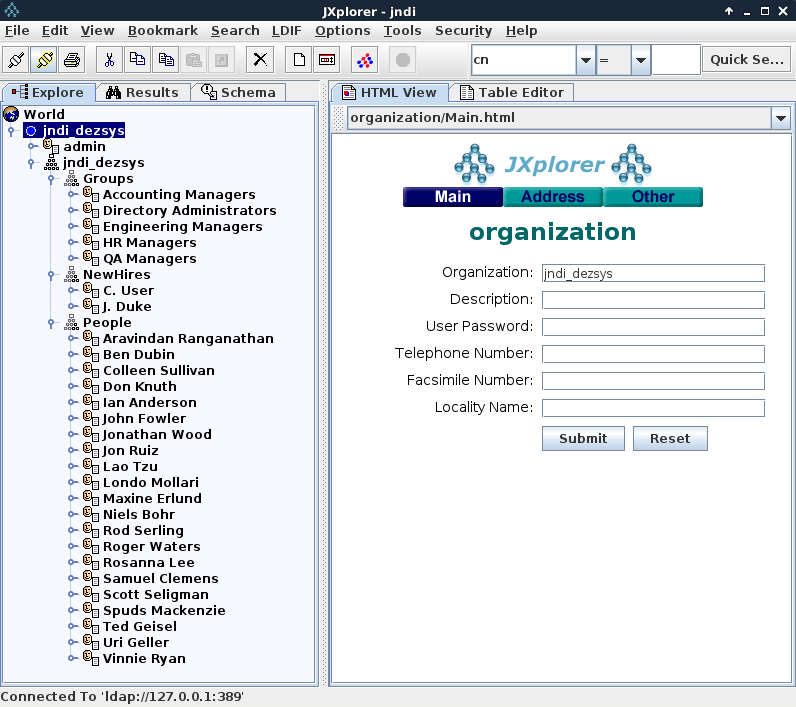
We replaced “o=JNDITutorial” to “o=jndi\_dezsys” in the whole file.

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:

C:\Users\Osman\Desktop\ldapmodify-adding_entries.JPG

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:



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

C:\Users\Osman\Desktop\ldapsearch-adding_entries.JPG

## Implementing the Naming and Directory Operations

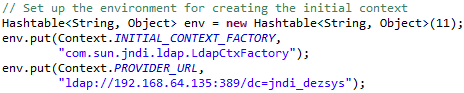
To implement the Naming and Directory Operations, we created a **Java-Project** in **Eclipse** and downloaded all the Sample-Files for the provided operations.

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

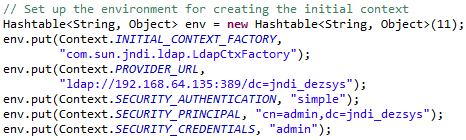
we implemented the following lines:



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

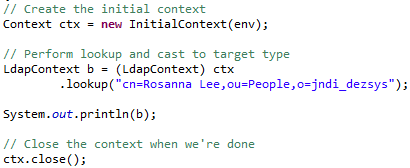
* Bind
* BindWithAttrs
* Create
* Destroy
* ModifyAttrs
* Rebind
* RebindWithAttrs
* Rename
* Unbind

we implemented the following lines:



### 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”:



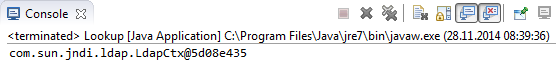
cn ……. common name

ou …… People

o ……... jndi\_dezsys

**Result:**

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

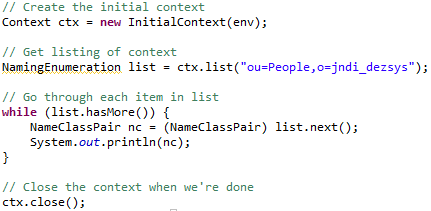


Otherwise we could get an error message in the console.

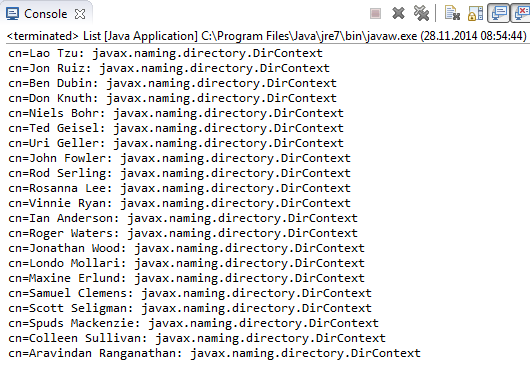
### List the Context

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.

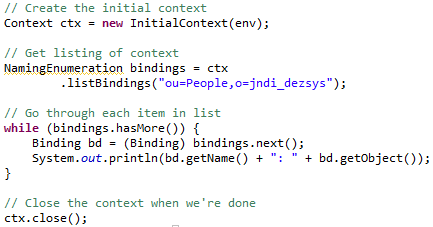
ctx.list() returns an enumeration of NameClassPair. Each NameClassPair consists of the object’s name and its class name. The following code fragment lists the contents of the “ou=People,o=jndi\_dezsys” directory (i.e., the files and directories found in “ou=People,o=jndi\_dezsys” directory):



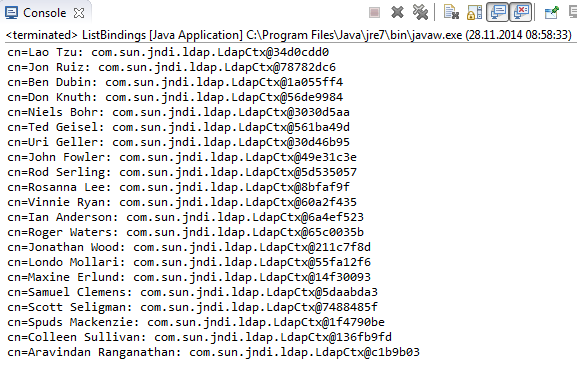
**Result:**



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:



**Result:**



### Add, Replace or Remove a Binding

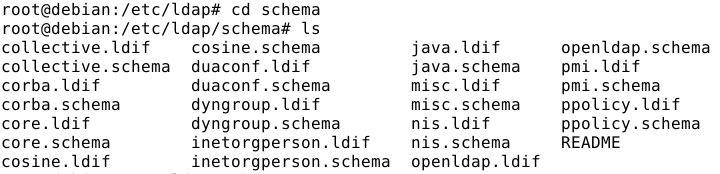
Before we were able to add, replace or remove a binding, we had to add the specified shemes by executing the following commands in the directory “/etc/ldap/schema/”:

# cd /etc/ldap/schema

# ldapadd –Y EXTERNAL –H ldapi:///-f java.ldif

# ldapadd –Y EXTERNAL –H ldapi:///-f corba.ldif

# ls

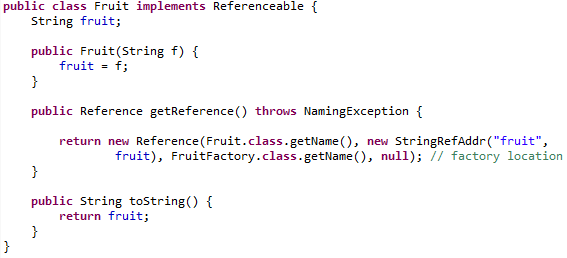


The following classes were necessary to create the objects to be bound:

Class “Fruit.java”:

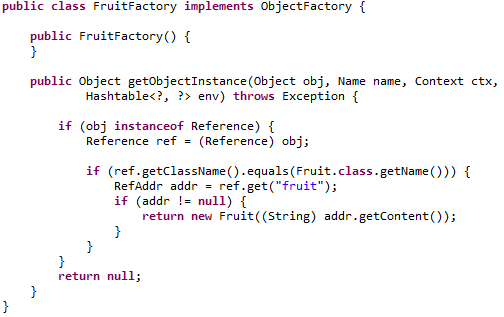
This class is used by the Bind example. It is a referenceable class that can

be stored by service providers like the LDAP and file system providers.



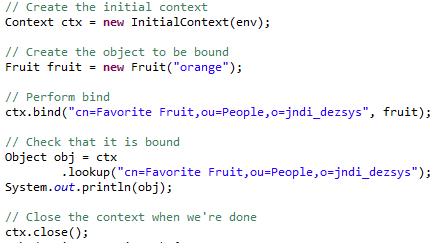
Class “FruitFactory.java”:

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



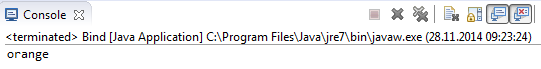
Now we are able to create the objects by using this classes.

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().

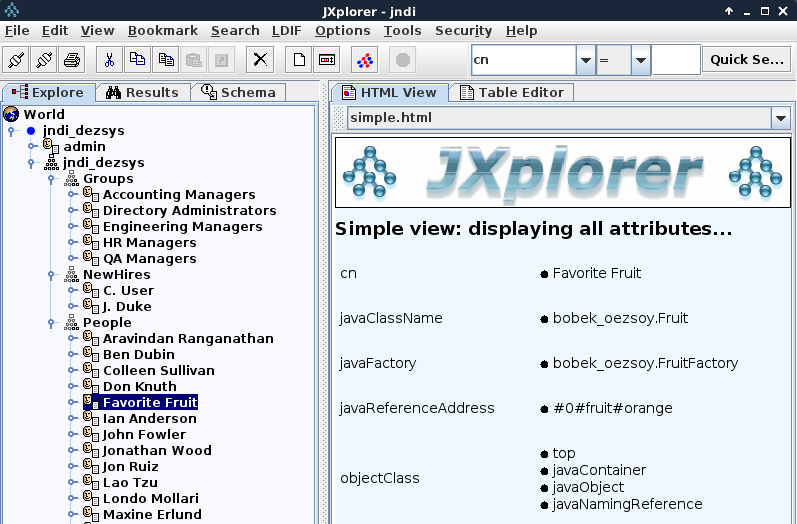


**Result:**

In the terminal we got the message:

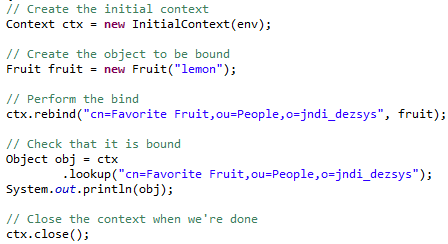


And if we take a look to the LDAP-Server, we can make sure that the object is bound successfully:



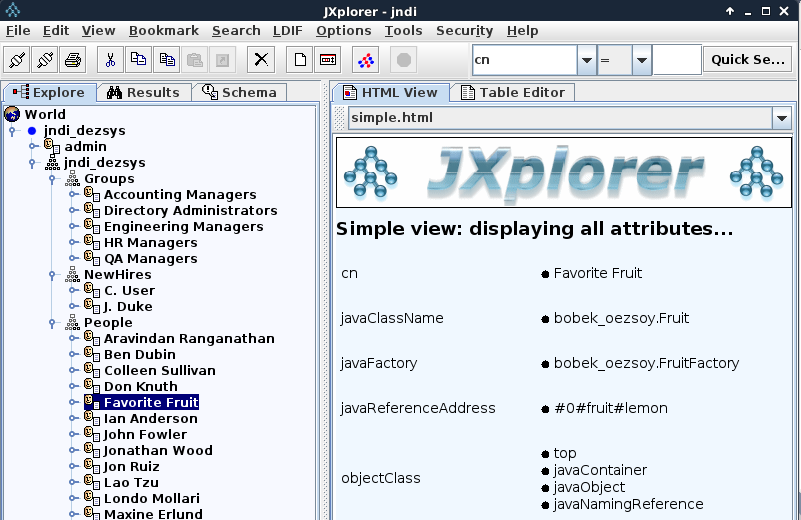
Now we are going to check the rebind(). It is used to add or replace a binding and accepts the same arguments as bind(), but the semantics are such that if the name is already bound, then it will be unbound and the newly given object will be bound.

The following code shows how to overwrite an existing binding:

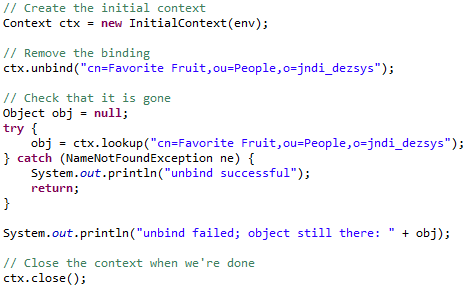


**Result:**

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”:

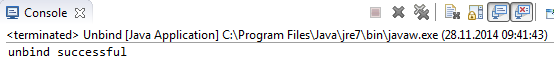


To remove a binding, we have to use unbind() as follows:



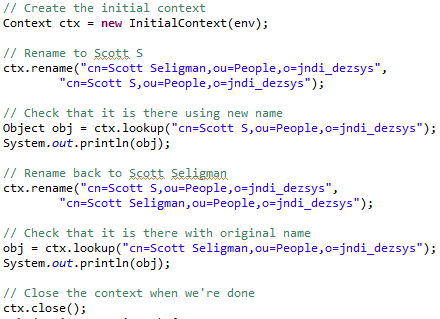
**Result:**

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



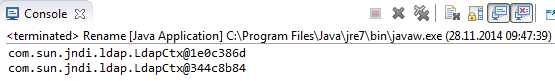
### Rename

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:



**Result:**

We got the following messages in the console:



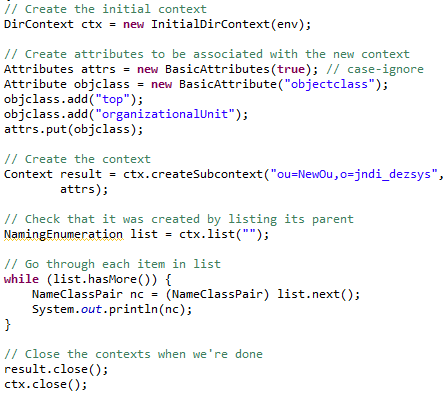
This messages mean that the rename-process is successfully finished, otherwise we could get an error-message.

### 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”).

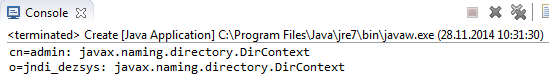
To create a naming context, we have to supply to createSubcontext() the name of the context that we want to create. To create a context that has attributes, we have to supply to DirContext.createSubcontext() the name of the context that we want to create and its attributes.

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:

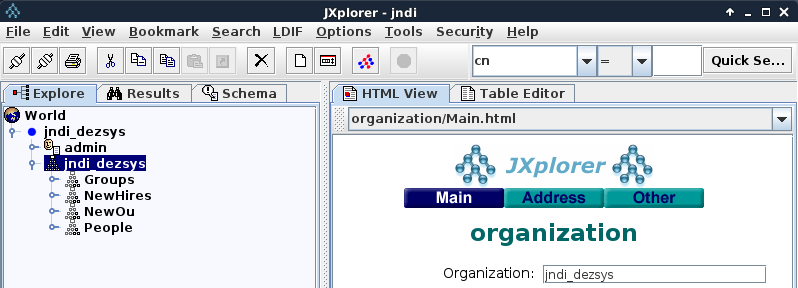


**Result:**

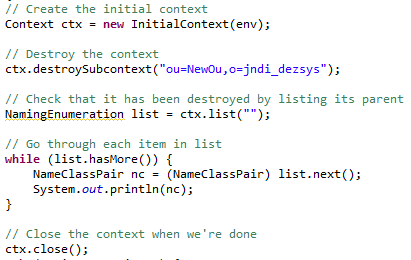
We got the following messages in the console:



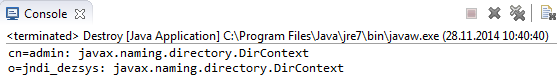
In the naming service, we can make sure that that specified context is created:



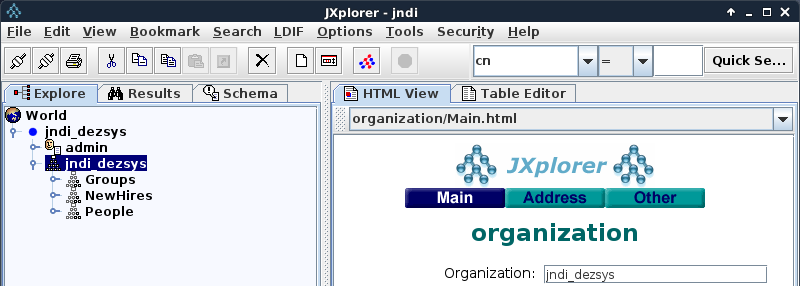
To destroy a context, we have to supply to destroySubcontext() the name of the context to destroy. The following code destroys the context “ou=NewOu,o=jndi\_dezsys” in the context ctx:



**Result:**



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

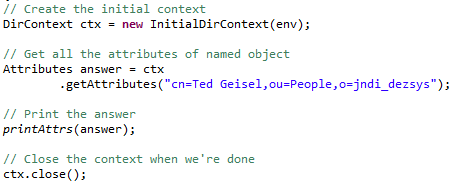


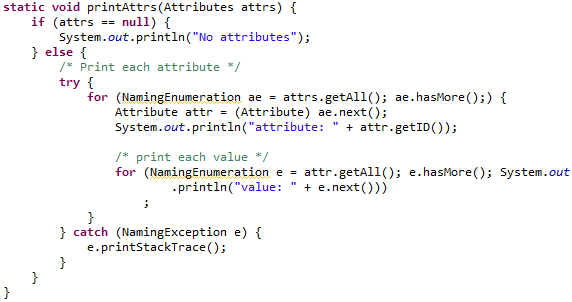
### Attribute Names

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

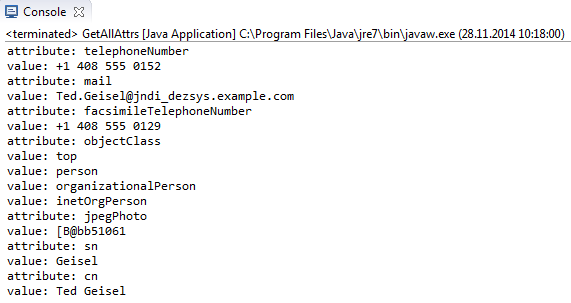
### Read Attributes

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:





**Result:**



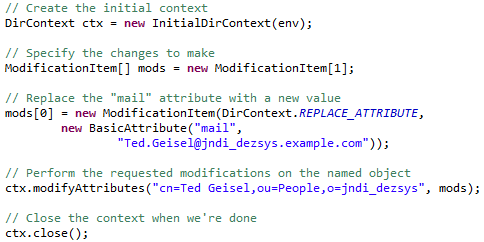
### Modify Attributes

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:

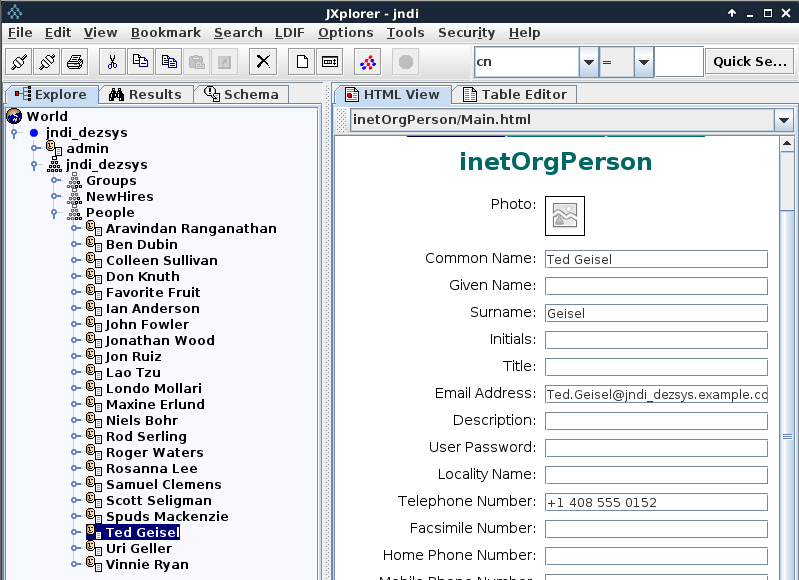
* ADD\_ATTRIBUTE
* REPLACE\_ATTRIBUTE
* REMOVE\_ATTRIBUTE

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”:



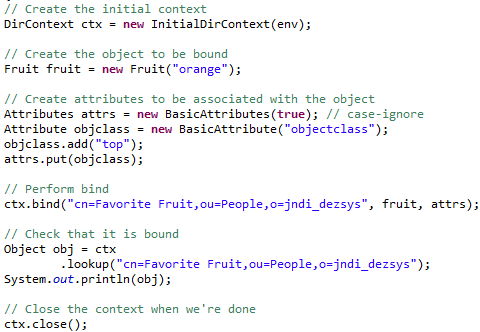
**Result:**



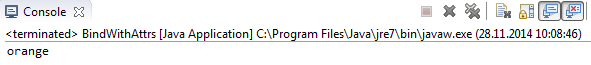
### Add, Replace Bindings with Attributes

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.

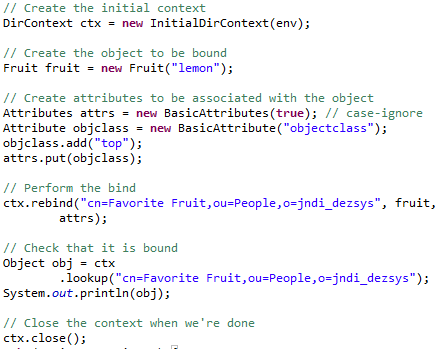
Adding a binding with attributes:



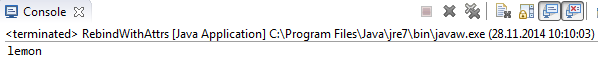
**Result:**



Rebinding an existing binding with attributes:



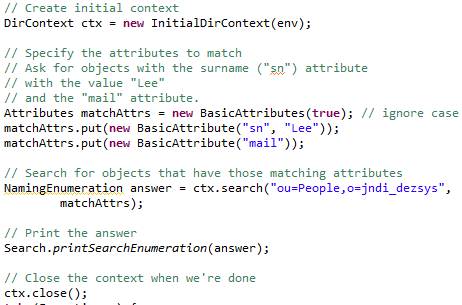
**Result:**



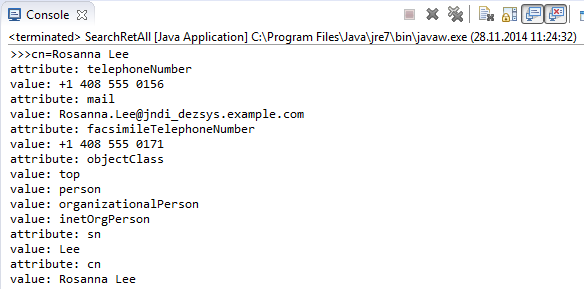
### Basic Search

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.

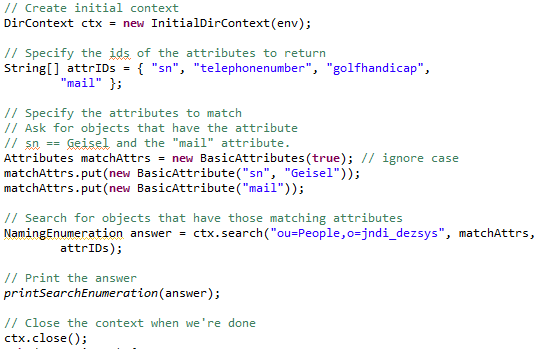
The following code creates an attribute set matchAttrs, which has two attributes "sn" and "mail". It specifies that the qualifying entries must have a surname ("sn") attribute with a value of "Lee" and a "mail" attribute with any value. It then invokes DirContext.search() to search the context "ou=People,o=jndi\_dezsys"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



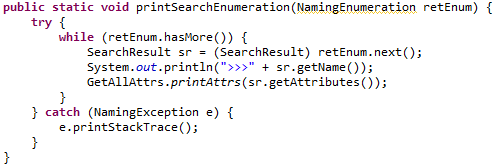
**Result:**



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":

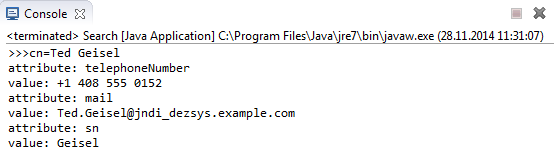


Method for printing out the answer:



**Result:**

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

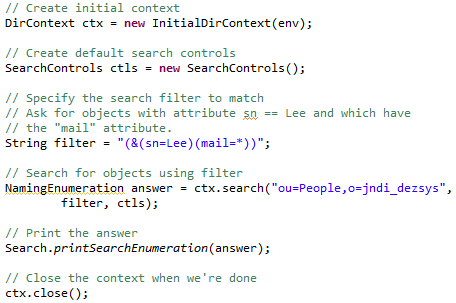


### Filters

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:

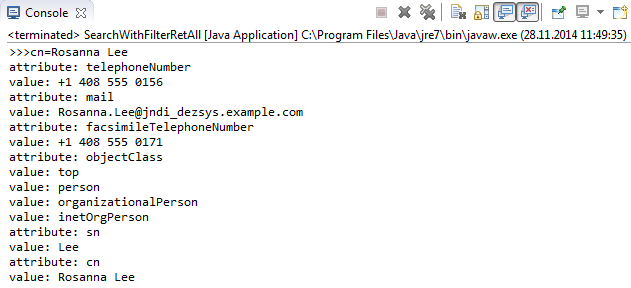


The following code creates a filter and default SearchControls, and uses them to perform a search:

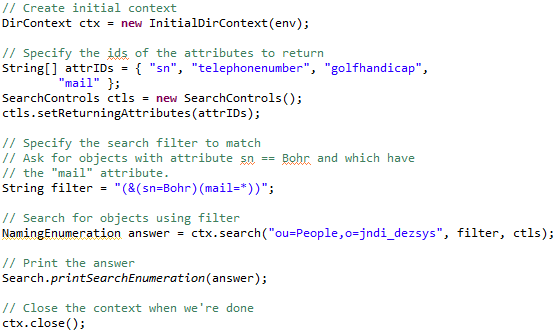


The search is equivalent to the one presented in the basic search example.

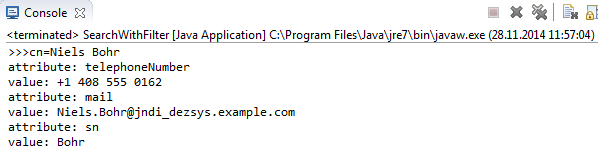
**Result:**



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():

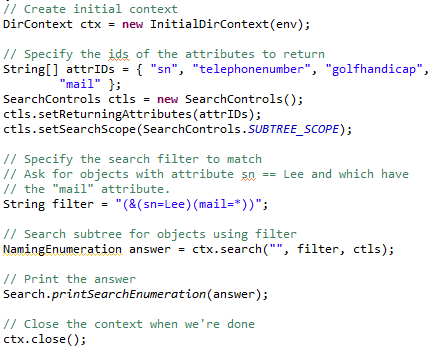


**Result:**



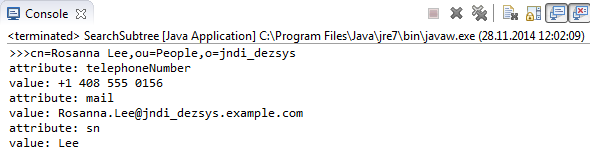
### Scope

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:

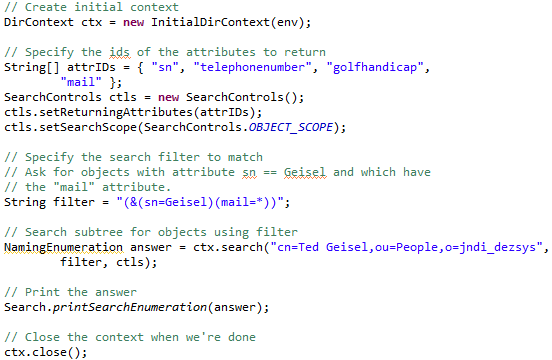


The code above searches the context ctx's subtree for entries that satisfy the specified filter. It finds the entry "cn= Rosanna Lee, ou=People,o=jndi\_dezsys" in this subtree that satisfies the filter.

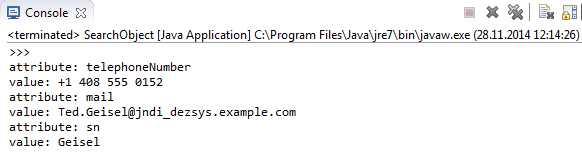
**Result:**



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

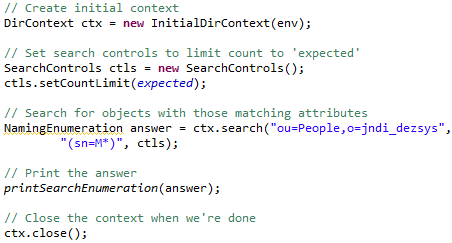


**Result:**

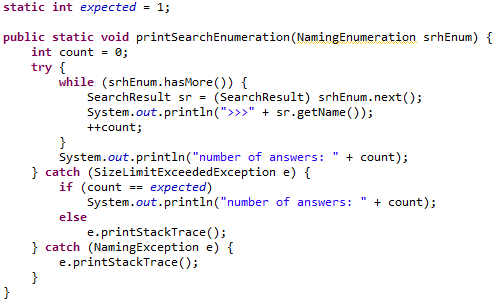


### Result Count

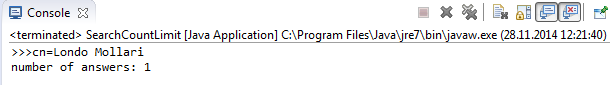
Sometimes, a query might produce too many answers and you want to limit the number of answers returned. You can do this by using the count limit search control. By default, a search does not have a count limit--it will return all answers that it finds. To set the count limit of a search, pass the number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



Method for printing out the answer:



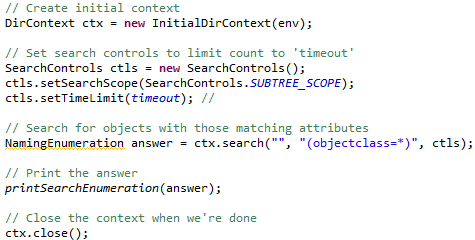
**Result:**



### 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:



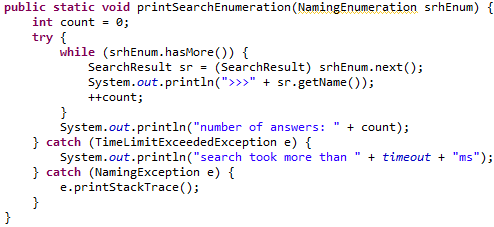
To get this particular example to exceed its time limit, you need to reconfigure it to use either a slow server, or a server that has lots of entries. Alternatively, you can use other tactics to make the search take longer than 1 second.

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.

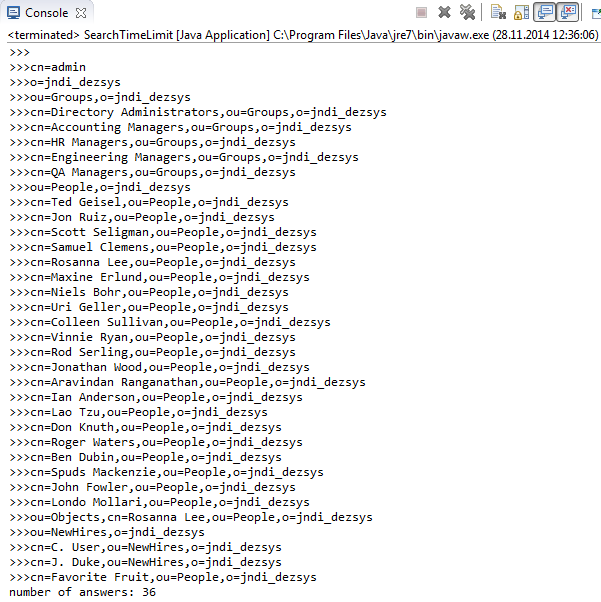
The value of the attribute “timeout” is set to:



Method for printing out the answer:



**Result:**



# 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 |