Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

KATEDRA INŻYNIERII OPROGRAMOWANIA http://kio.wi.zut.edu.pl/

INŻYNIERIA OPROGRAMOWANIA

Zaawansowane Techniki Programowania Java

#05 : JNDI (*javax.naming*)

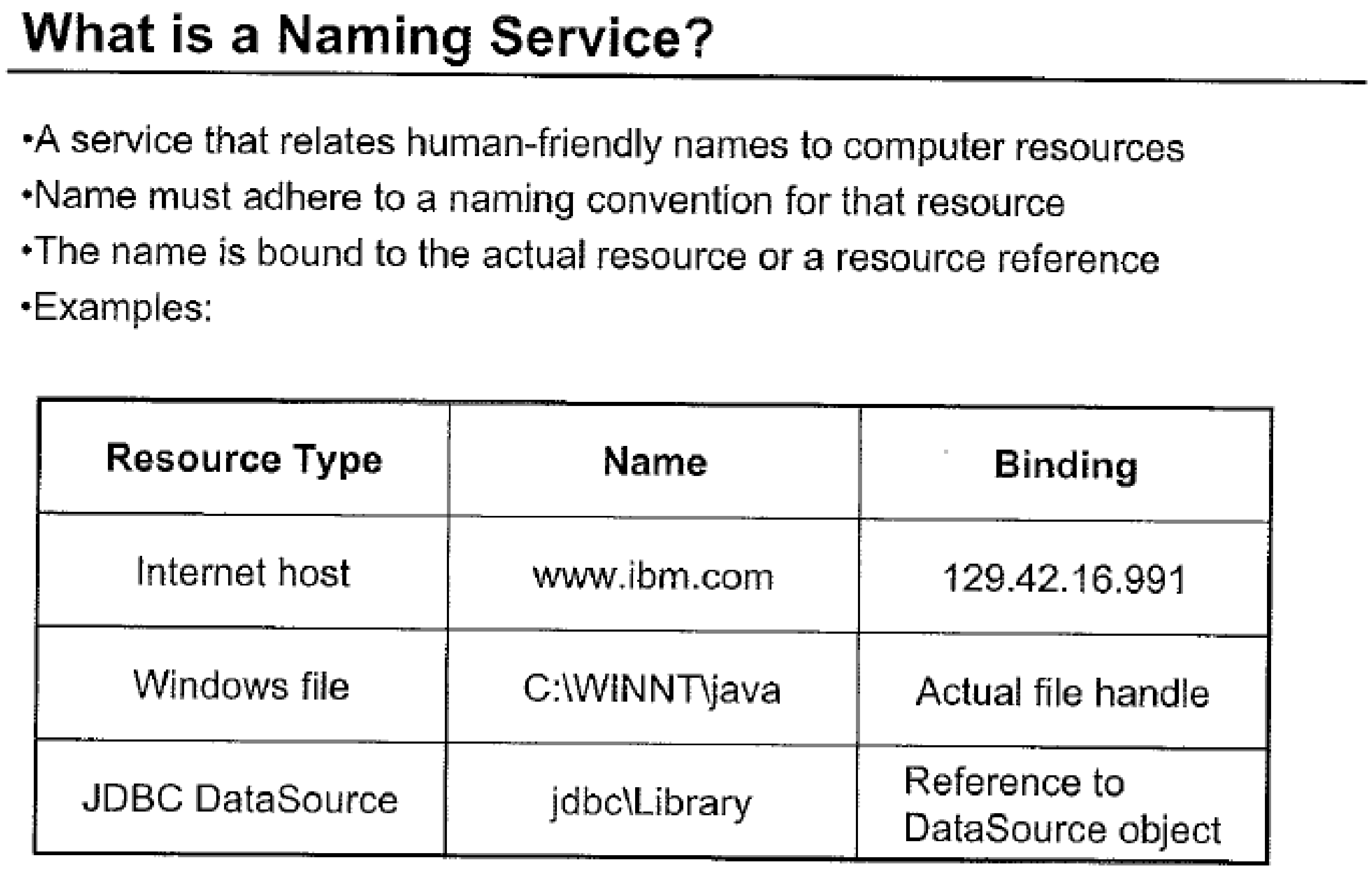
Prowadzący:

Krzysztof Kraska

email: kkraska@wi.zut.edu.pl

Szczecin, 12 maja 2018 r.

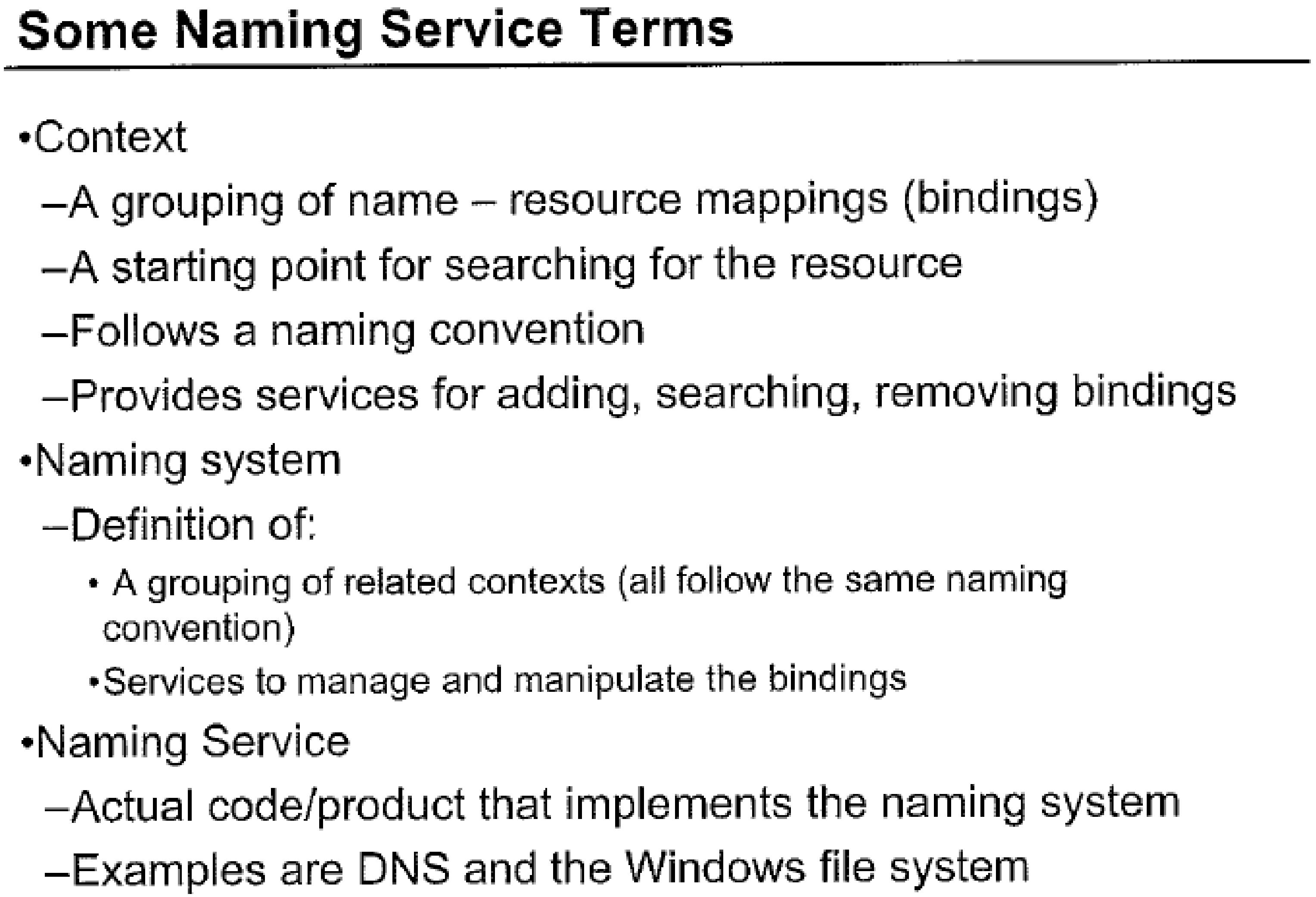
JAVA NAMING AND DIRECTORY INTERFACE



What is a Naming Service?

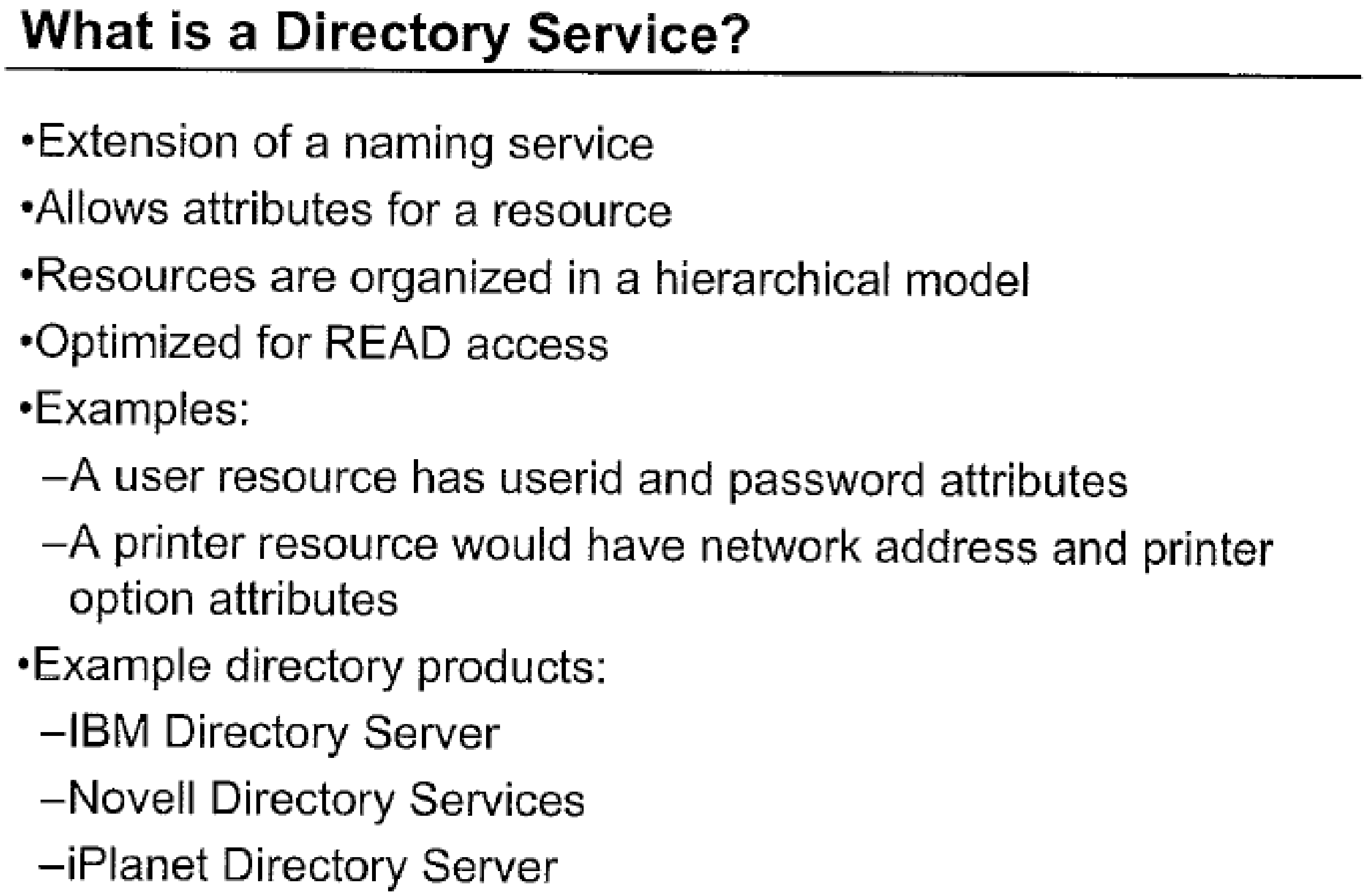
- A service that relates human-friendly names to computer resources
- Name must adhere to a naming convention for that resource
- The name is bound to the actual resource or a resource reference
- Examples:

Resource Type	Name	Binding
Internet host	www.ibm.com	129.42.16.991
Windows file	C:\WINNT\java	Actual file handle
JDBC DataSource	jdbc\Library	Reference to DataSource object



Some Naming Service Terms

- Context
  - A grouping of name – resource mappings (bindings)
  - A starting point for searching for the resource
  - Follows a naming convention
  - Provides services for adding, searching, removing bindings
- Naming system
  - Definition of:
    - A grouping of related contexts (all follow the same naming convention)
    - Services to manage and manipulate the bindings
- Naming Service
  - Actual code/product that implements the naming system
  - Examples are DNS and the Windows file system

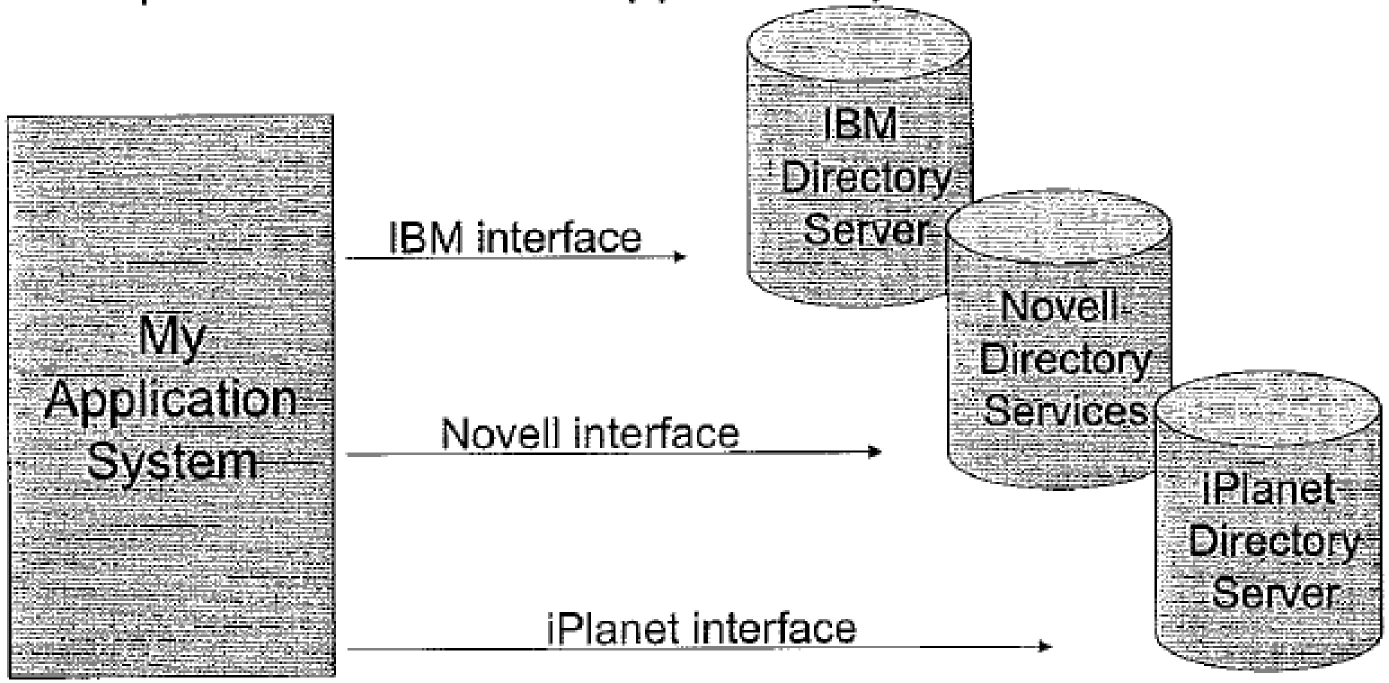


What is a Directory Service?

- Extension of a naming service
- Allows attributes for a resource
- Resources are organized in a hierarchical model
- Optimized for READ access
- Examples:
  - A user resource has userid and password attributes
  - A printer resource would have network address and printer option attributes
- Example directory products:
  - IBM Directory Server
  - Novell Directory Services
  - iPlanet Directory Server

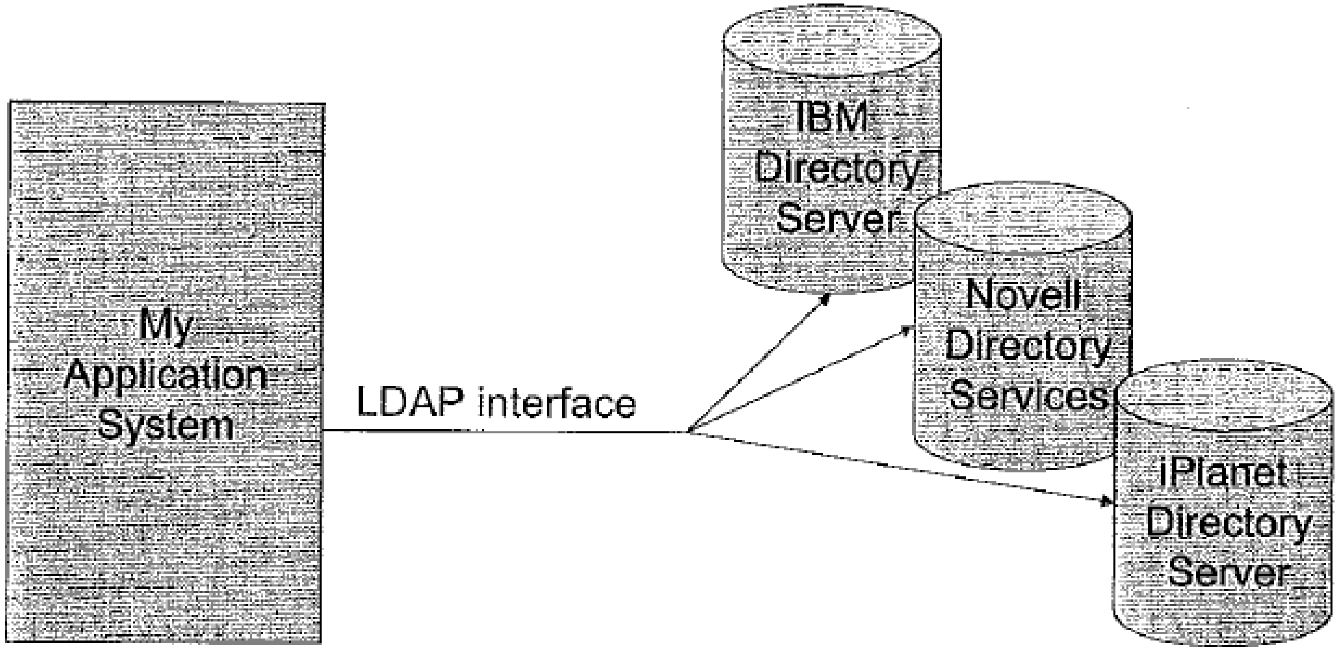
A Critical Issue with Directory Services

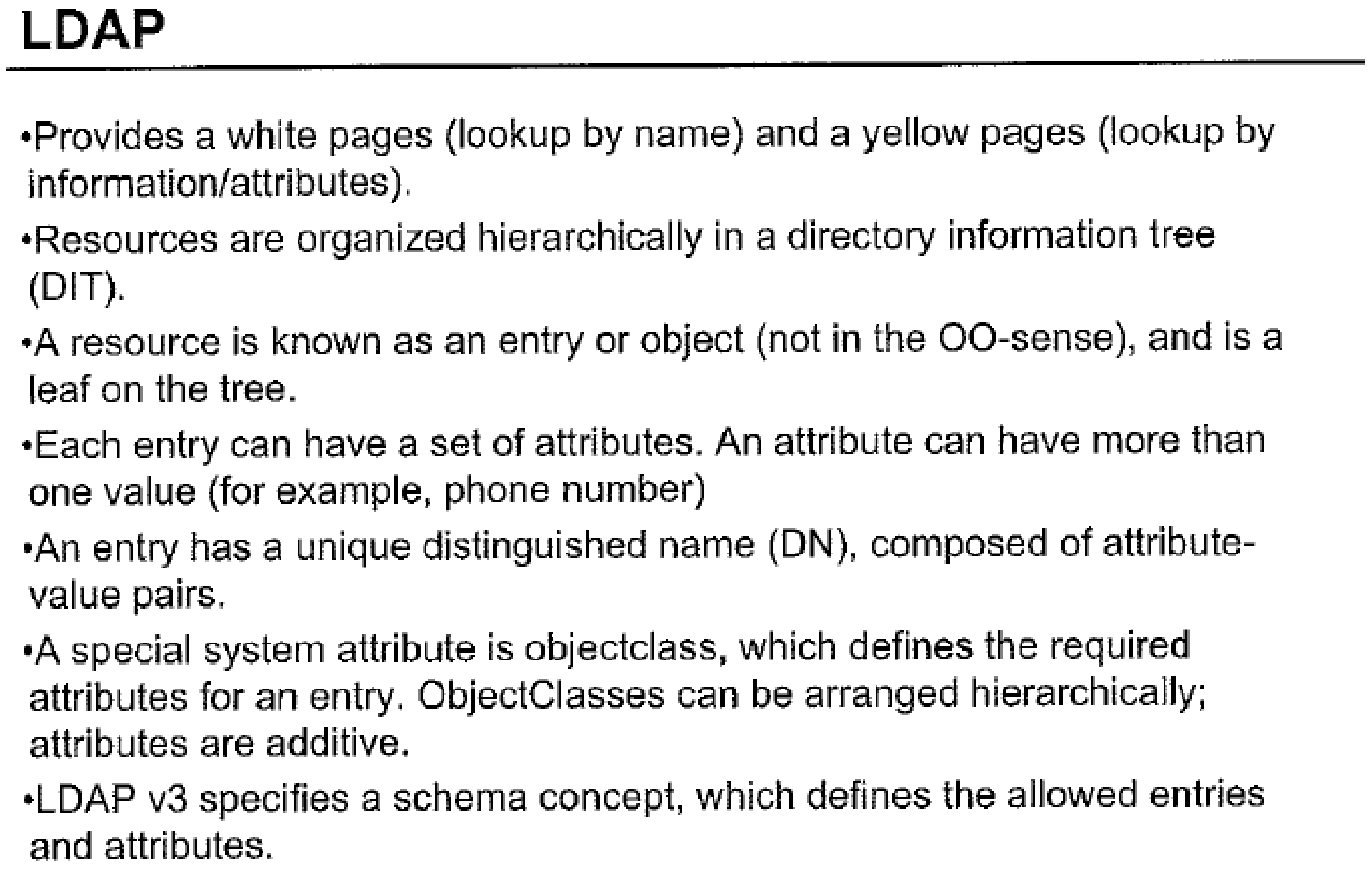
- Each directory has its own interface
- A company may use different directories for different resources
- Developers have to learn/support multiple interfaces



LDAP to the Rescue

- Lightweight Directory Access Protocol (LDAP)
  - IETF RFC 2251 (LDAP v3)
- Lightweight implementation of Directory Access Protocol (DAP) to access X.500-based directories
- Many directory services also provide an LDAP interface





LDAP

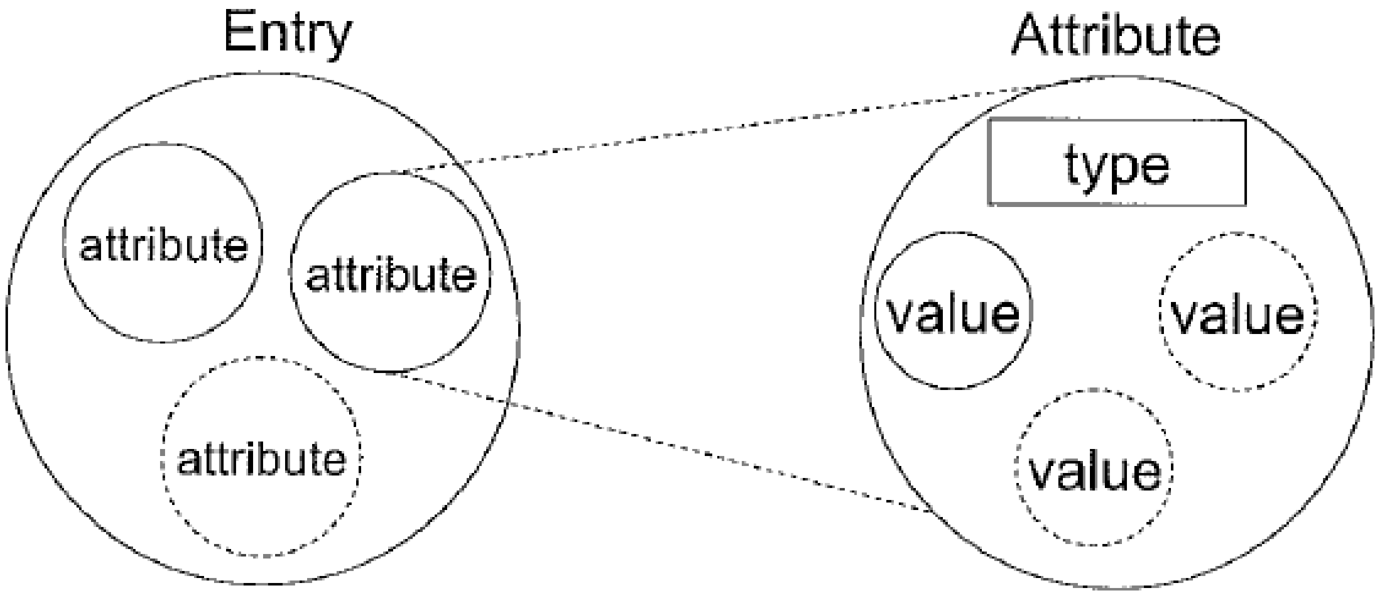
- Provides a white pages (lookup by name) and a yellow pages (lookup by information/attributes).
- Resources are organized hierarchically in a directory information tree (DIT).
- A resource is known as an entry or object (not in the OO-sense), and is a leaf on the tree.
- Each entry can have a set of attributes. An attribute can have more than one value (for example, phone number)
- An entry has a unique distinguished name (DN), composed of attribute-value pairs.
- A special system attribute is objectclass, which defines the required attributes for an entry. ObjectClasses can be arranged hierarchically; attributes are additive.
- LDAP v3 specifies a schema concept, which defines the allowed entries and attributes.

JAVA NAMING AND DIRECTORY INTERFACE

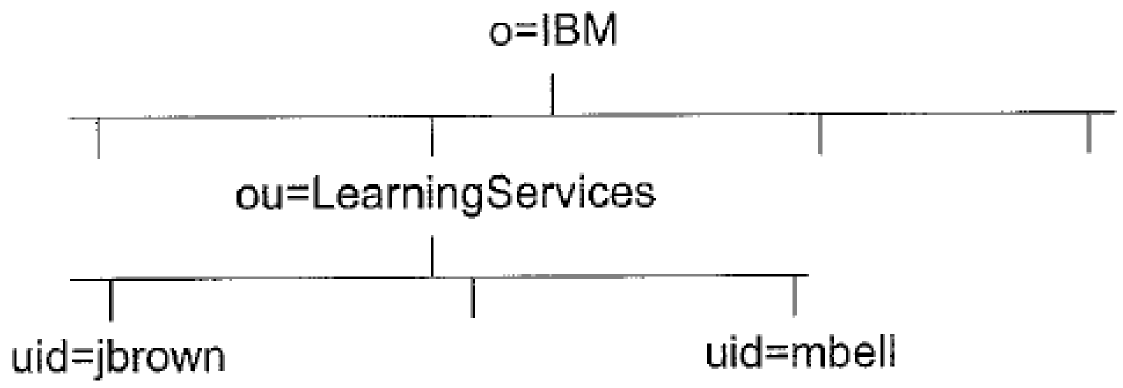
Entry – Attribute Relationship

•An entry has required and possibly optional attributes

•An attribute has a type (syntax) and one or more values



LDAP Directory Entry Example



- Distinguished Name:
  - uid=jbrown, ou=LearningServices, o=IBM
- Besides the DN, the entry also has required (and optional) attributes:
  - cn=Jim Brown, sn=Brown, givenname=Jim, empid=12345, telephoneNumber=310-555-1212, telephoneNumber=310-555-9999
- The Relative Distinguished Name (RDN) is the leftmost part of the DN (uid=jbrown in this example)

Java Naming and Directory Interface

The DN is made up of attribute-value pairs

The schema specifies the allowed objectClasses and attributes

Another entry might be:

DN of "uid=mbell, ou=LearningServices, o=IBM"

cn=Mark Bell, sn=Bell, givenname=Mark, empid=54321, telephoneNumber=417-555-1213, telephoneNumber=417-555-8888

objectClass entries indicate a form of inheritance: specifications of required and optional attributes.

LDAP Directory Data - LDIF

- LDAP does not dictate how entries are to be stored
- LDAP does have a human-readable format
  - LDAP Data Interchange Format (LDIF)
- Example:

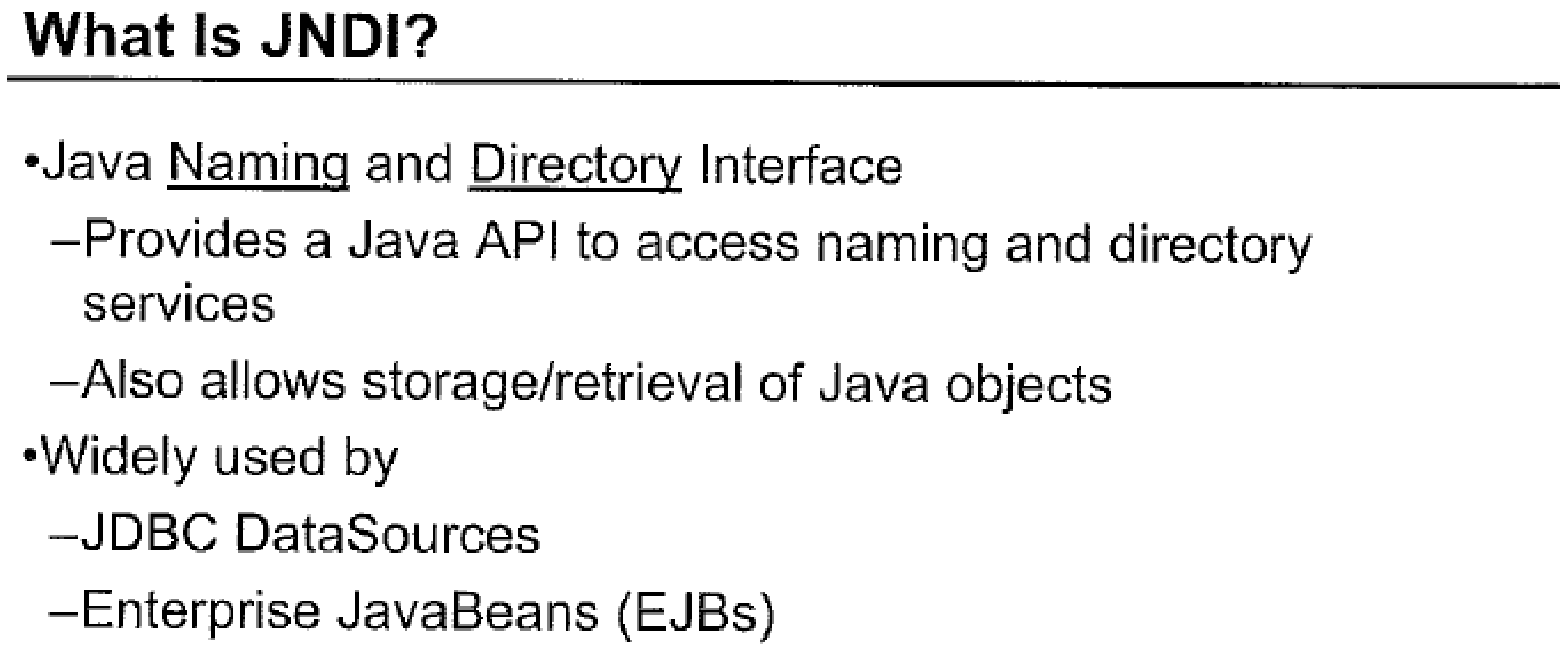
```
dn: uid=jbrown, ou=LearningServices, o=IBM
objectclass: organizationalPerson
objectclass: person
objectclass: top
cn: Jim Brown
sn: Brown
givenname: Jim
empid: 12345
telephoneNumber: 310-555-1212
telephoneNumber: 310-555-9999
```

- The schema must contain:
  - **objectClasses** of person and organizationalPerson
  - **Attributes** of dn, uid, ou, o, cn, sn, givenname, empid, telephoneNumber

LDAP Operations on the Directory

- Query
  - Search the directory
  - Compare an entry for a specific attribute value
- Update
  - Add an entry to the directory
  - Delete a leaf-node entry from the directory
  - Modify the attributes and values in an entry
  - Move an entry or subtree of entries elsewhere in the DIT
- Authentication
  - Bind (authenticate) between a client and directory
  - Terminate a client-directory connection
  - Abandon an operation outstanding on the directory

JAVA NAMING AND DIRECTORY INTERFACE



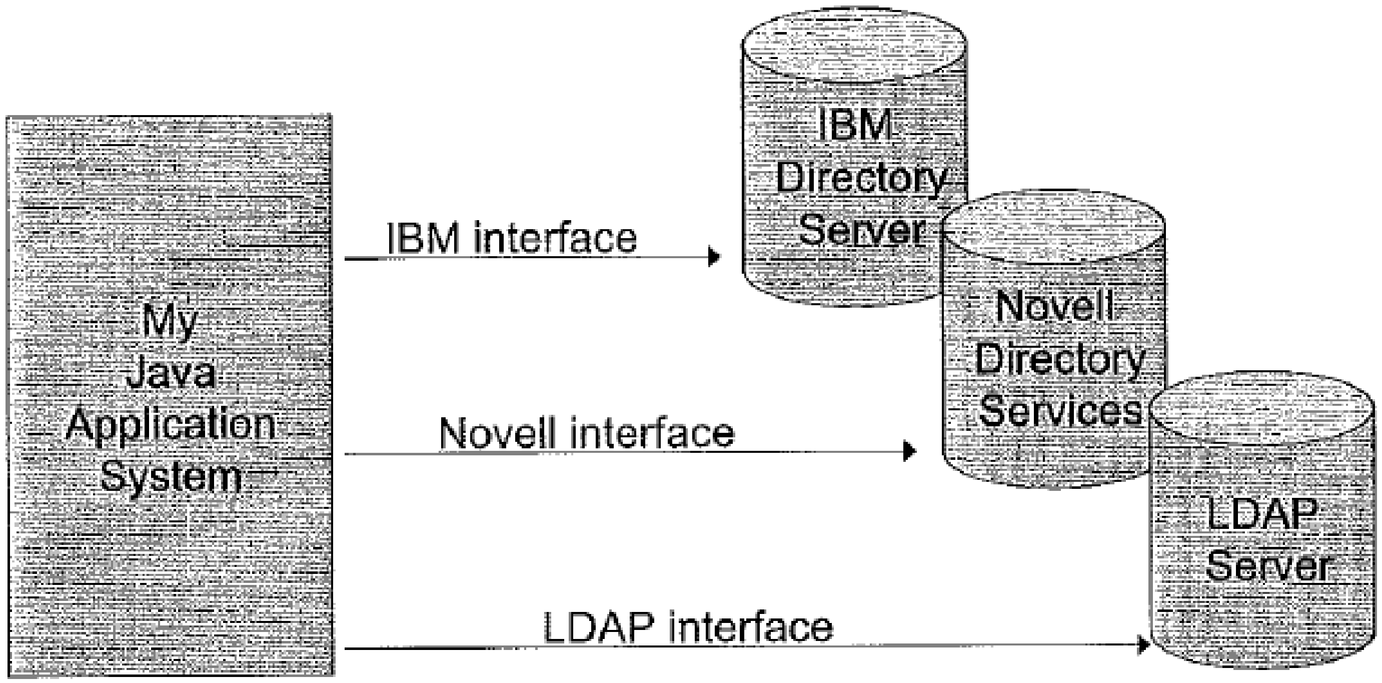
What Is JNDI?

- Java <u>Naming</u> and <u>Directory</u> Interface
  - Provides a Java API to access naming and directory services
  - Also allows storage/retrieval of Java objects
- Widely used by
  - JDBC DataSources
  - Enterprise JavaBeans (EJBs)

JAVA NAMING AND DIRECTORY INTERFACE

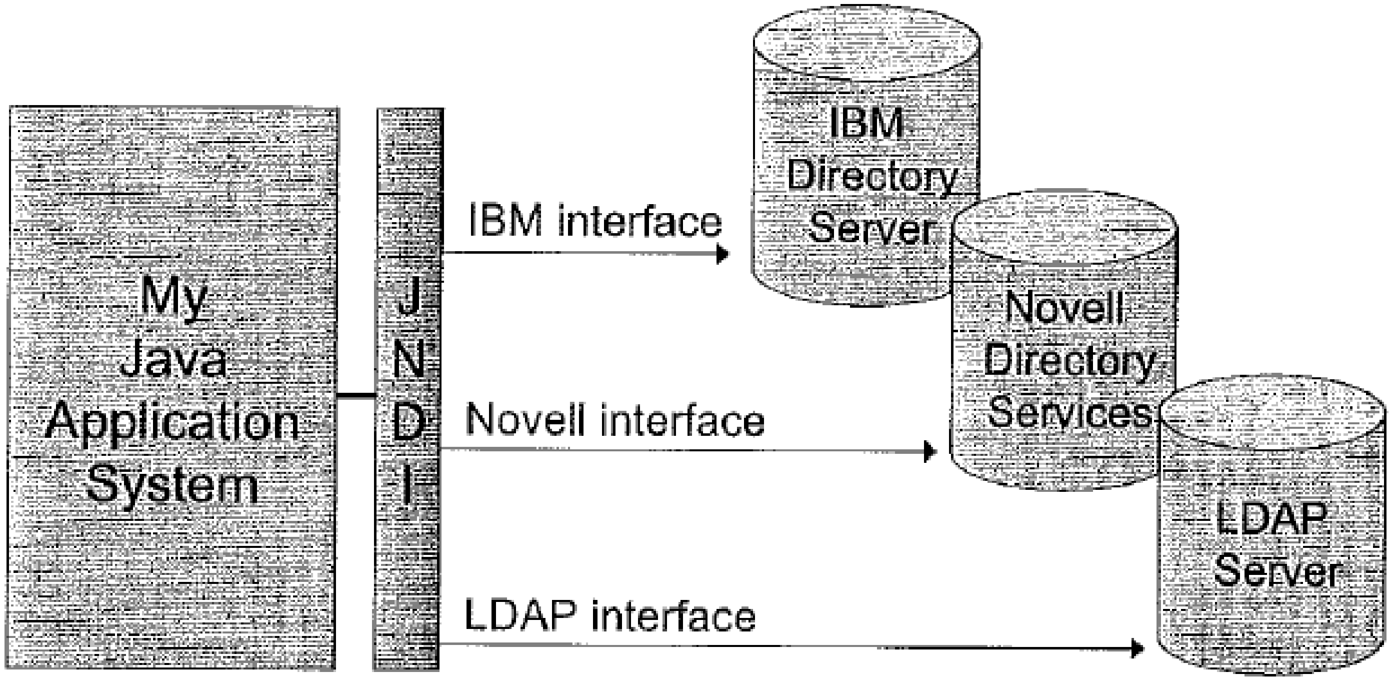
Why JNDI? (1 of 2)

•Java applications accessing the directory services would need to code to each unique interface



Why JNDI? (2 of 2)

•Java applications using JNDI to access the directory services need to code only to the JNDI interface

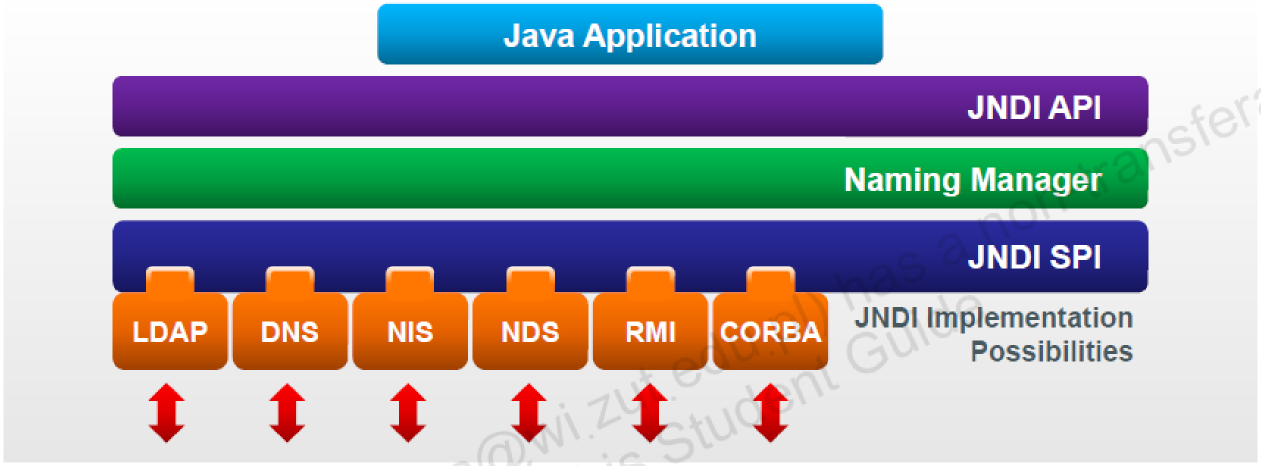


Java Naming and Directory Interface

JNDI

The Java Naming and Directory Interface (JNDI) is an API that provides naming and directory functionality to applications.

- Independent of any implementation
- Built on a service provider interface (SPI)
  - SPIs can be plugged in to an API.

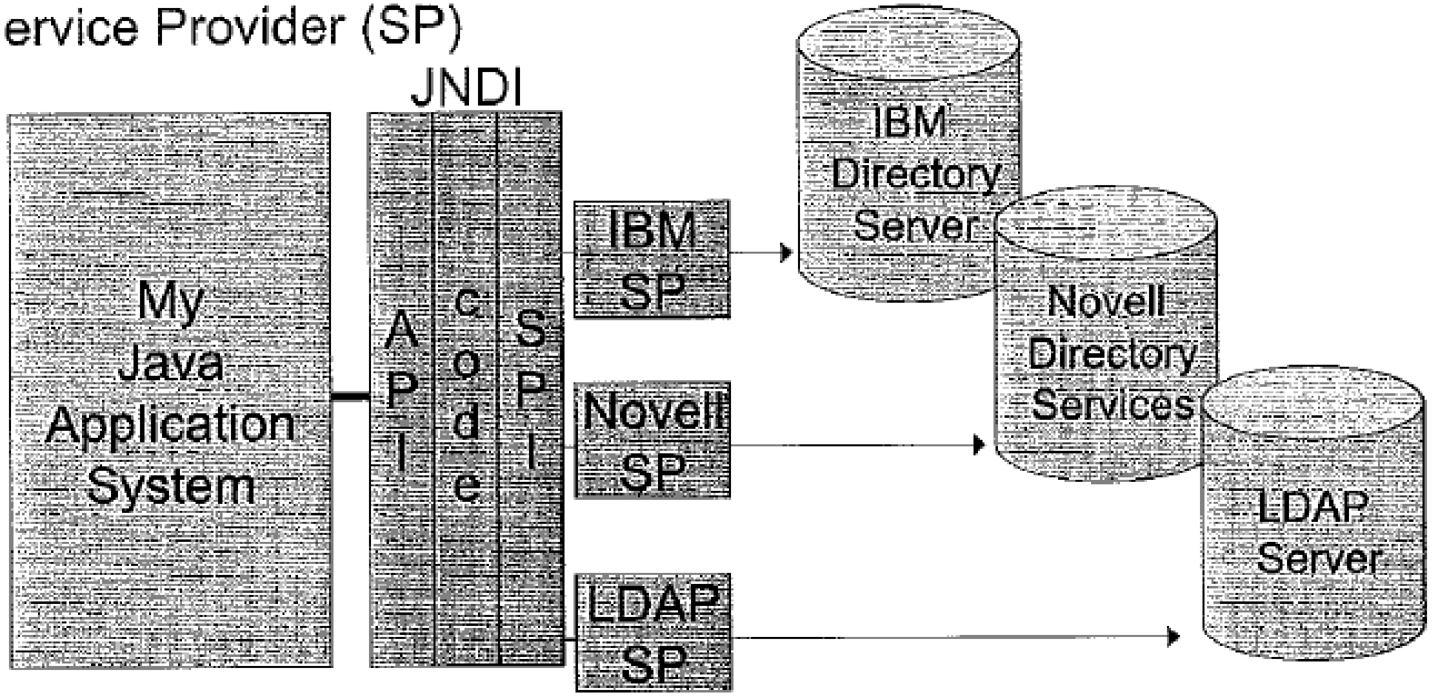


ORACLE

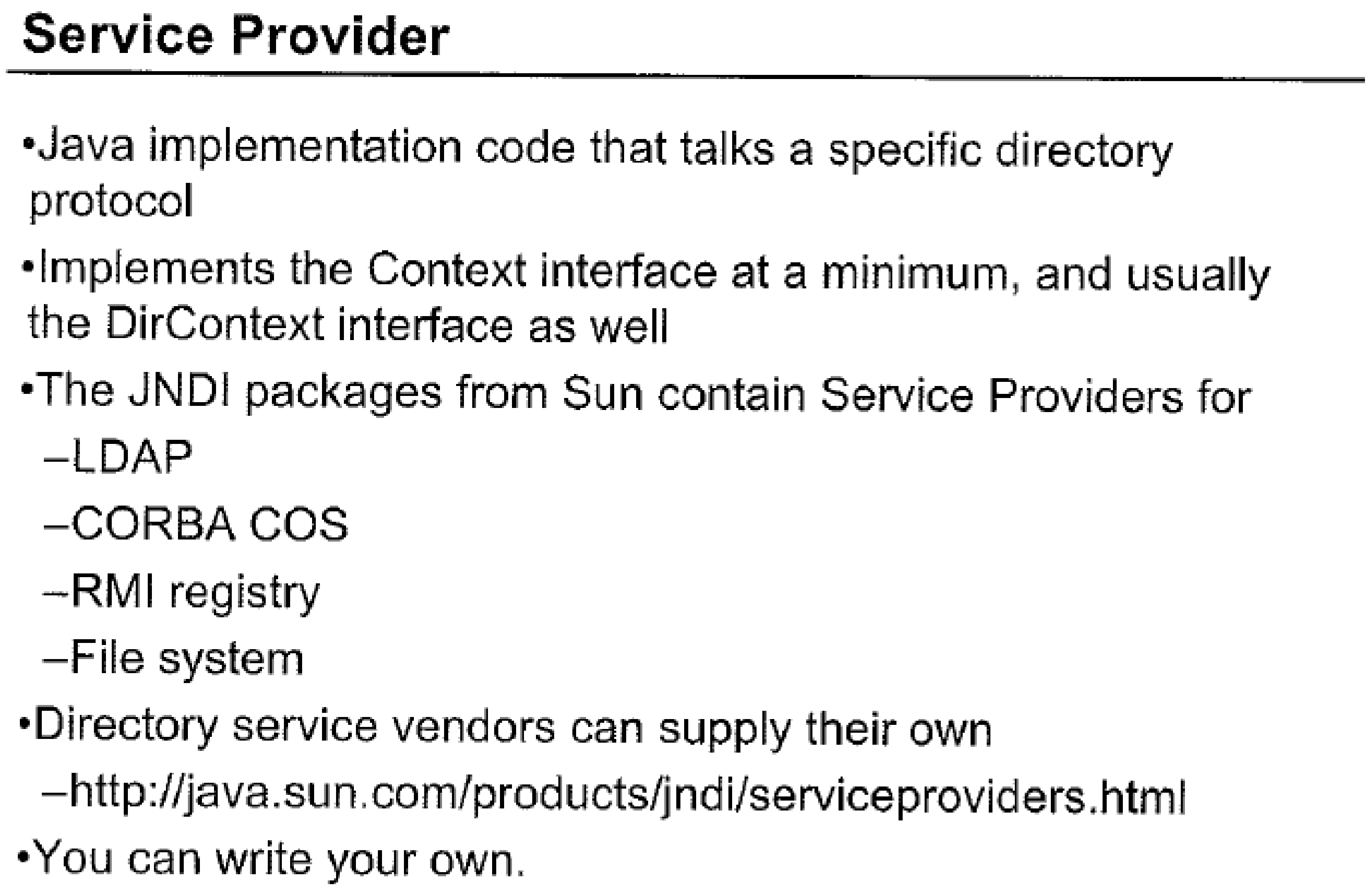
Szczecin, 12 maja 2018 r.

JNDI Structure

- Java applications using JNDI to access the directory services need to code only to the JNDI interface (API)
- The JNDI implementation code provides a Service Provider Interface (SPI), which allows each directory to plug in its own Service Provider (SP)



JAVA NAMING AND DIRECTORY INTERFACE



Service Provider

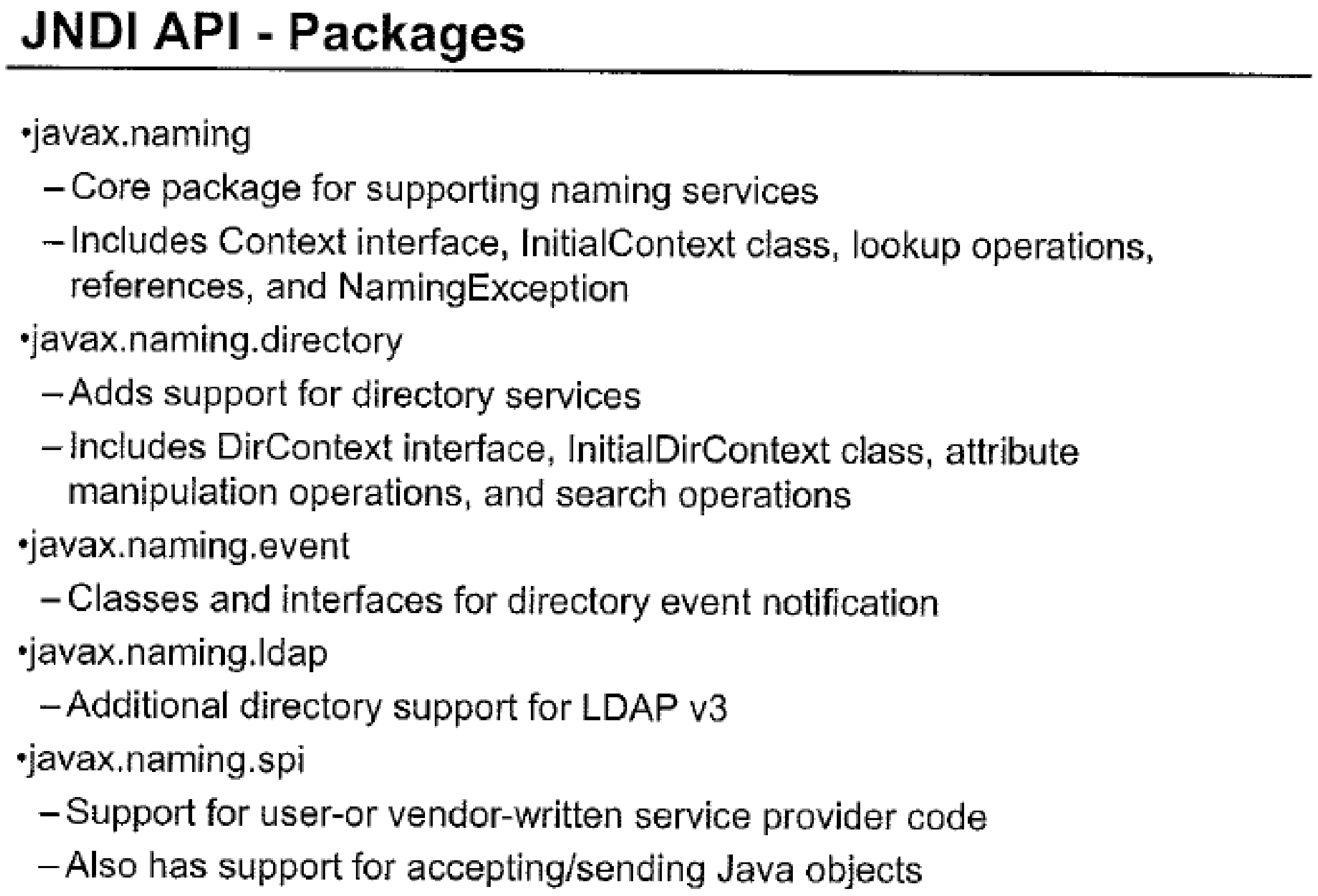
- Java implementation code that talks a specific directory protocol
- Implements the Context interface at a minimum, and usually the DirContext interface as well
- The JNDI packages from Sun contain Service Providers for
  - LDAP
  - CORBA COS
  - RMI registry
  - File system
- Directory service vendors can supply their own
  - http://java.sun.com/products/jndi/serviceproviders.html
- You can write your own.

JAVA NAMING AND DIRECTORY INTERFACE

JNDI Setup

- Have The JNDI packages available
  - JDK 1.4 already includes the JNDI packages
  - For pre-1.3 JDKs, you can download the packages from Sun
- Install a JNDI-accessible Directory Server
  - The JDK or the vendor supplies a Service Provider
- For Directory Services, define a schema that includes the objectClasses and attributes you need (LDAP)
- Write your application code to use the services!

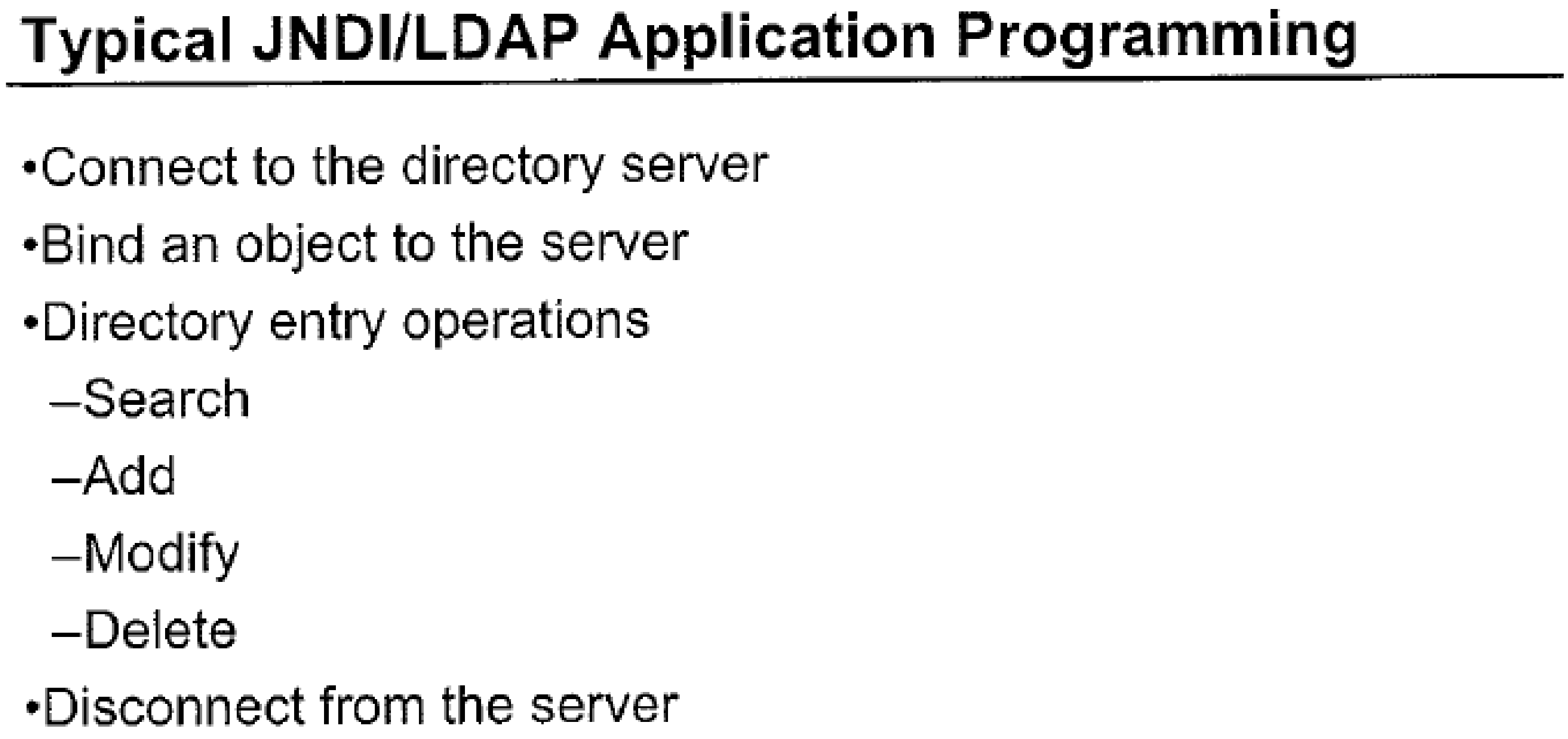
JAVA NAMING AND DIRECTORY INTERFACE



JNDI API - Packages

- javax.naming
  - Core package for supporting naming services
  - Includes Context interface, InitialContext class, lookup operations, references, and NamingException
- javax.naming.directory
  - Adds support for directory services
  - Includes DirContext interface, InitialDirContext class, attribute manipulation operations, and search operations
- javax.naming.event
  - Classes and interfaces for directory event notification
- javax.naming.ldap
  - Additional directory support for LDAP v3
- javax.naming.spi
  - Support for user-or vendor-written service provider code
  - Also has support for accepting/sending Java objects

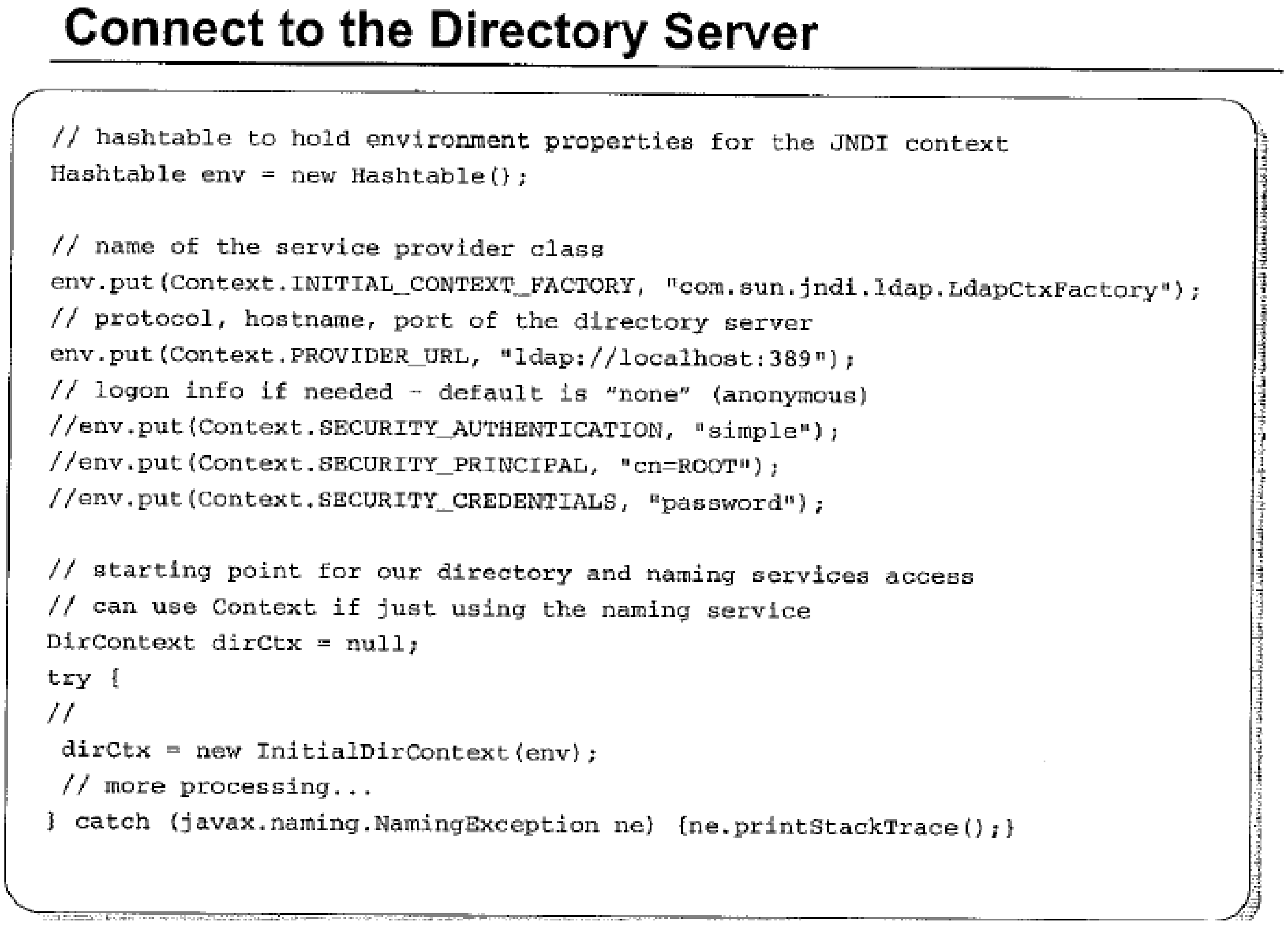
Java Naming and Directory Interface



Typical JNDI/LDAP Application Programming

- Connect to the directory server
- Bind an object to the server
- Directory entry operations
  - Search
  - Add
  - Modify
  - Delete
- Disconnect from the server

JAVA NAMING AND DIRECTORY INTERFACE



Connect to the Directory Server

```
// hashtable to hold environment properties for the JNDI context
Hashtable env = new Hashtable();

// name of the service provider class
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
// protocol, hostname, port of the directory server
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
// logon info if needed - default is "none" (anonymous)
//env.put(Context.SECURITY_AUTHENTICATION, "simple");
//env.put(Context.SECURITY_PRINCIPAL, "cn=ROOT");
//env.put(Context.SECURITY_CREDENTIALS, "password");

// starting point for our directory and naming services access
// can use Context if just using the naming service
DirContext dirCtx = null;
try {
//
 dirCtx = new InitialDirContext(env);
 // more processing...
} catch (javax.naming.NamingException ne) {ne.printStackTrace();}
```

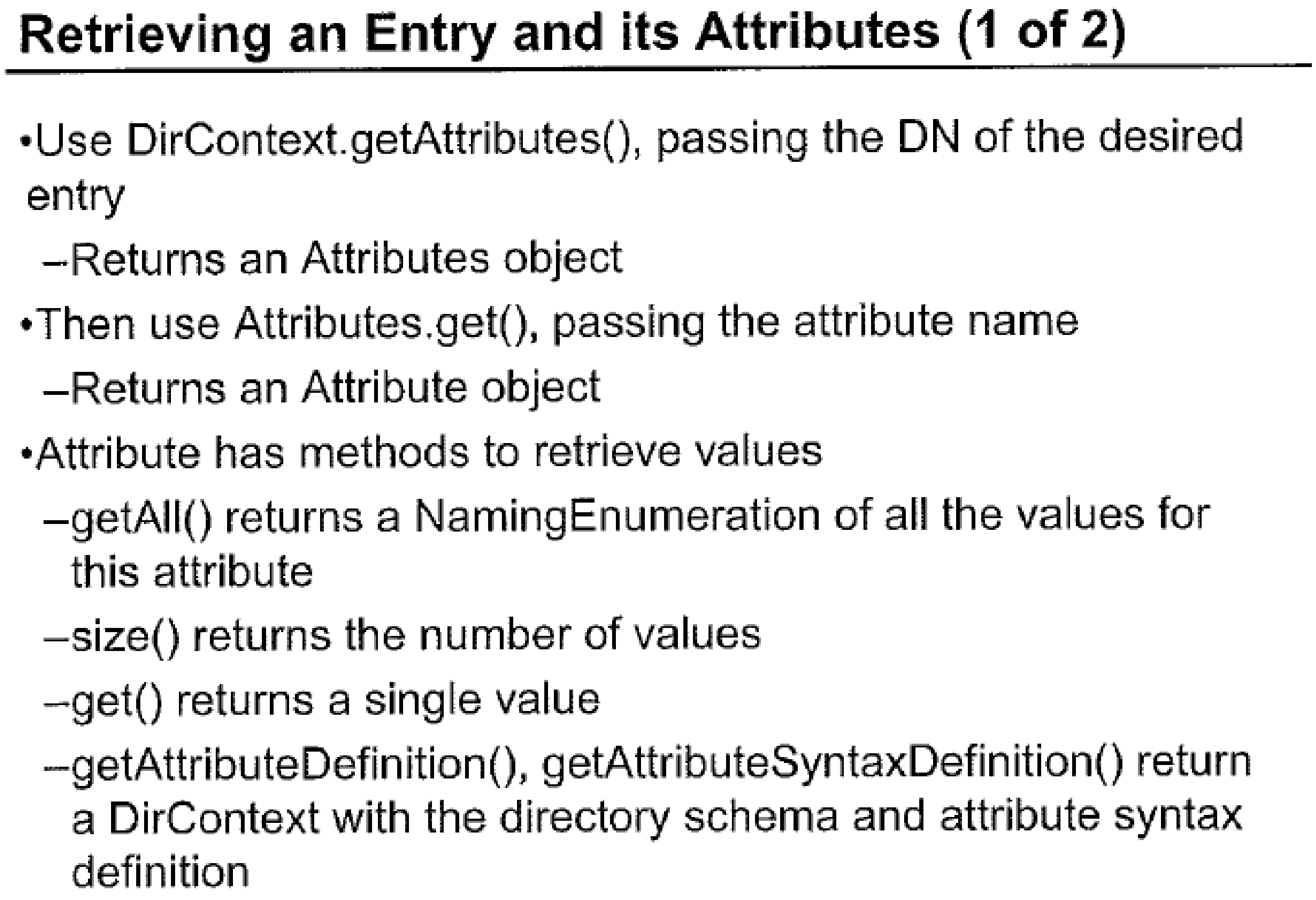
Java Naming and Directory Interface

Context and NamingException are in *javax.naming*, DirContext is in *javax.naming.directory.*

This example is accessing the directory as anonymous, so the sign-on properties are commented out. simple authentication passes a userid/password in clear text. strong allows use of encrypted userids/passwords or certificates, dependent on the directory capabilities.

A good approach (pattern) is to use a singleton for retrieval of the DirContext. Then once the reference is obtained, it can be easily retrieved for subsequent processing.

JAVA NAMING AND DIRECTORY INTERFACE



Retrieving an Entry and its Attributes (1 of 2)

- Use DirContext.getAttributes(), passing the DN of the desired entry
  - Returns an Attributes object
- Then use Attributes.get(), passing the attribute name
  - Returns an Attribute object
- Attribute has methods to retrieve values
  - getAll() returns a NamingEnumeration of all the values for this attribute
  - size() returns the number of values
  - get() returns a single value
  - getAttributeDefinition(), getAttributeSyntaxDefinition() return a DirContext with the directory schema and attribute syntax definition

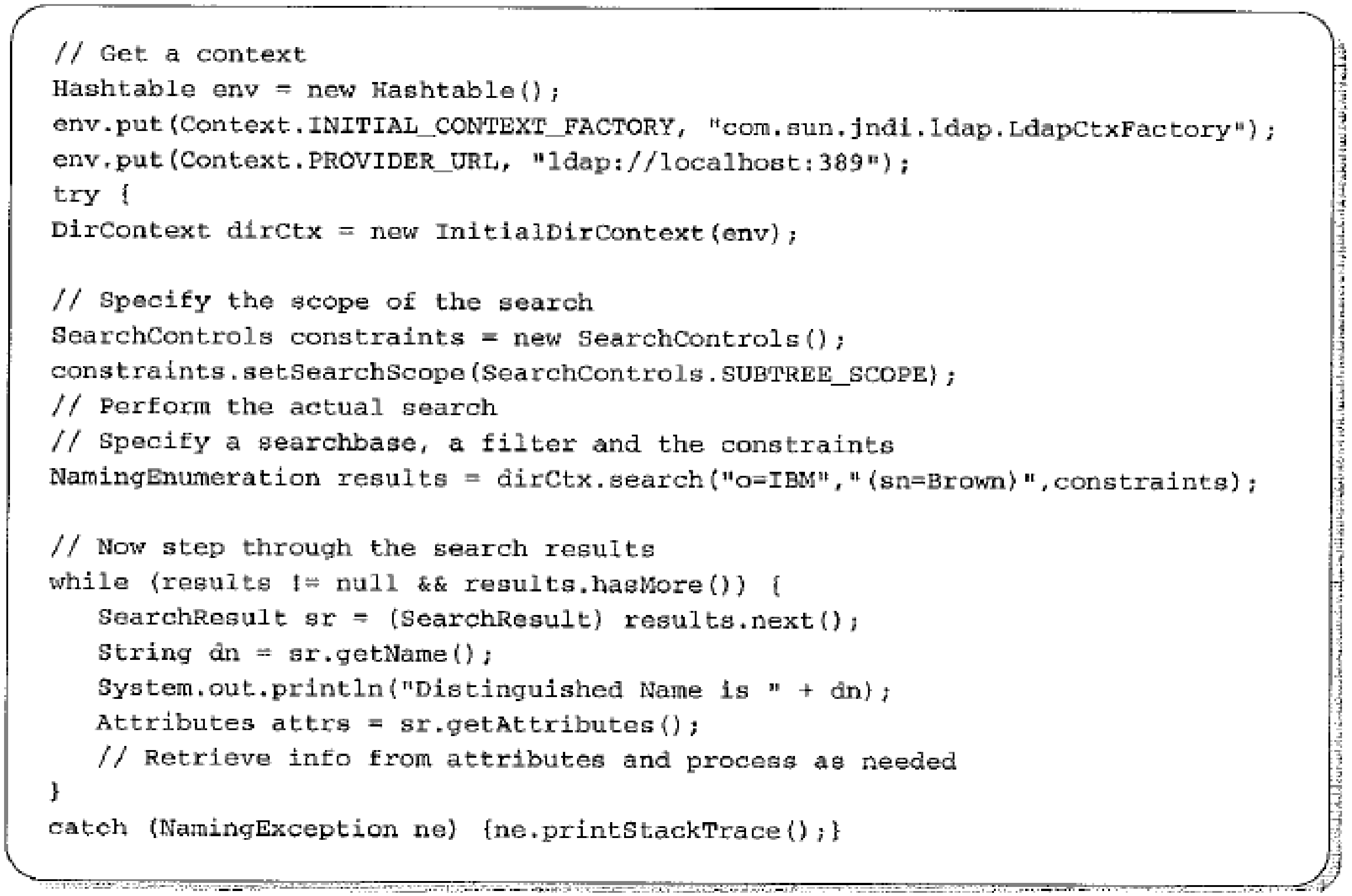
Retrieving an Entry and its Attributes (2 of 2)

```
// Get a context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
try {
DirContext dirCtx = new InitialDirContext(env);

Attributes attrs =
   dirCtx.getAttributes(" uid=jbrown, ou=LearningServices, o=IBM ");

// Get the value of the commonName attribute for this entry
Attribute cnAttr = attrs.get("cn");
String fullName = (String) cnAttr.get();

// Get the value of empid
String employeeID = (String) attrs.get("empid").get();

} catch (NamingException ne) {ne.printStackTrace();}
```

Searching the Directory (1 of 2)

- DirContext.search() takes parameters to control the actual search
  - A search base to indicate the starting point in the tree
  - A filter to indicate the search criteria (attribute-value info)
  - A SearchControl object to control the scope of the tree to search
- The search() returns a NamingEnumeration of SearchResult objects
- SearchResult has
  - getName() to return this entry's DN
  - getAttributes() to return the Attributes object for this entry
- The Attributes object can be processed to return attributes and their values

Searching the Directory (2 of 2)

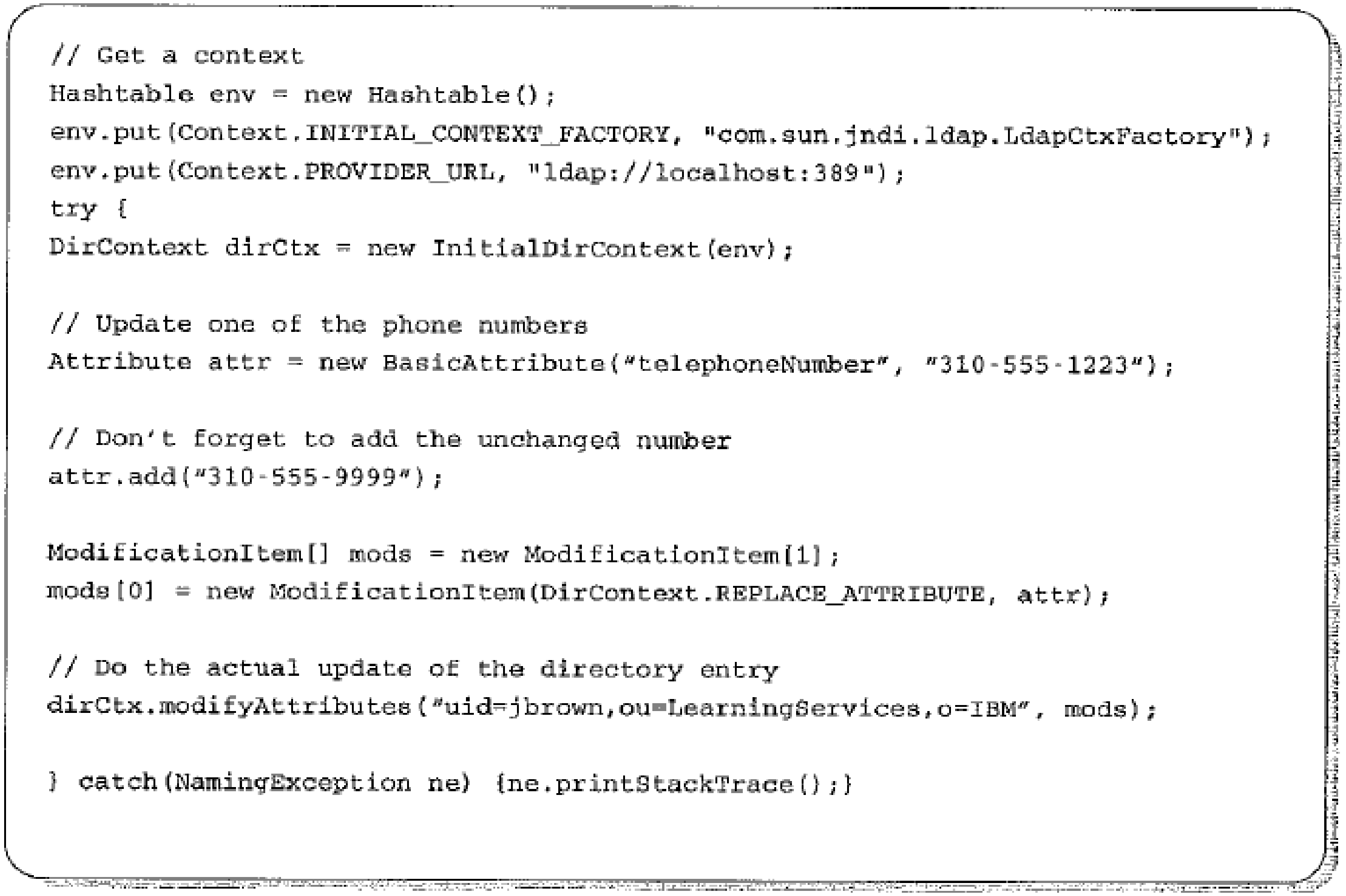


```
// Get a context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
try {
DirContext dirCtx = new InitialDirContext(env);

// Specify the scope of the search
SearchControls constraints = new SearchControls();
constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
// Perform the actual search
// Specify a searchbase, a filter and the constraints
NamingEnumeration results = dirCtx.search("o=IBM","(sn=Brown)",constraints);

// Now step through the search results
while (results != null && results.hasMore()) {
   SearchResult sr = (SearchResult) results.next();
   String dn = sr.getName();
   System.out.println("Distinguished Name is " + dn);
   Attributes attrs = sr.getAttributes();
   // Retrieve info from attributes and process as needed
}
catch (NamingException ne) {ne.printStackTrace();}
```

Modifying an Entry (1 of 2)

- You can modify an existing entry's attributes:
  - DirContext.REPLACE_ATTRIBUTE, also ADD and DELETE
- Use ModificationItem and BasicAttribute to specify changes
  - BasicAttribute is used to specify the new attribute-value
    - Additional values can be added
  - ModificationItem is used to indicate the type of modification, and the new attribute-value
- Actual change occurs via DirContext.modifyAttributes()
  - Operation type
  - Array of ModificationItem objects
- Be sure to include all values for a multivalued attribute, or they will be lost

Modifying an Entry (2 of 2)



```
// Get a context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
try {
DirContext dirCtx = new InitialDirContext(env);

// Update one of the phone numbers
Attribute attr = new BasicAttribute(“telephoneNumber”, “310-555-1223”);

// Don’t forget to add the unchanged number
attr.add(“310-555-9999”);

ModificationItem[] mods = new ModificationItem[1];
mods[0] = new ModificationItem(DirContext.REPLACE_ATTRIBUTE, attr);

// Do the actual update of the directory entry
dirCtx.modifyAttributes(“uid=jbrown,ou=LearningServices,o=IBM”, mods);

} catch(NamingException ne) {ne.printStackTrace();}
```

Java Naming and Directory Interface

BasicAttribute and ModificationItem are in *javax.naming.directory.*

This example shows updating one of the telephone numbers in the preexisting entry. A new attribute is constructed containing the new number, plus the other original number.

The DirContext defines REPLACE_ATTRIBUTE, ADD_ATTRIBUTE, and REMOVE_ATTRIBUTE.

The original entry was:

DN: uid=jbrown, ou=Learning Services, o=IBM

cn=Jim Brown, sn=Brown, givenname=Jim, empid=12345,

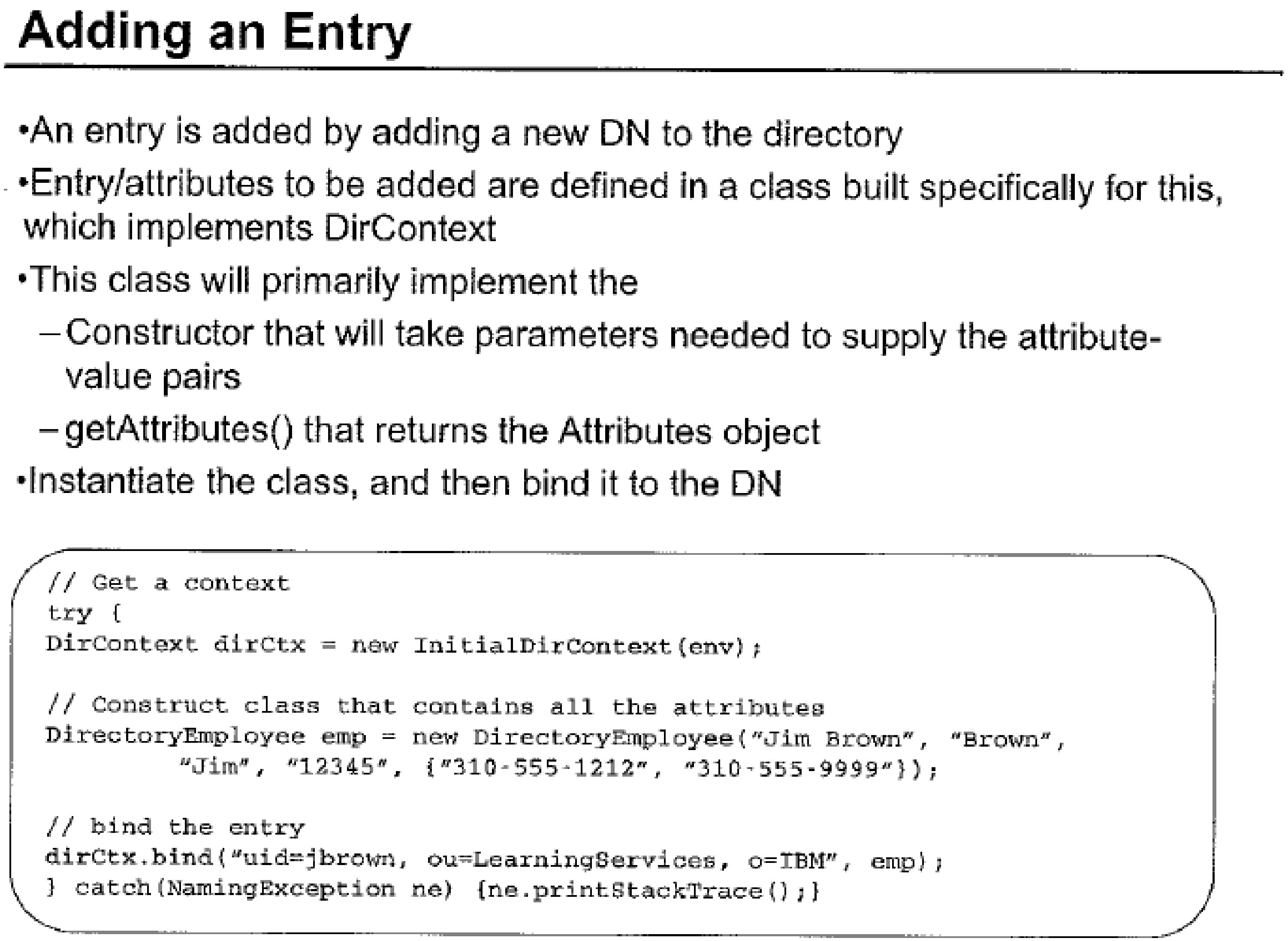
telephonenumber=310-555-1212, telephonenumber=310-555-9999

Removing an Entry

- Usually a restricted operation, done infrequently
- Removes the entry and its attributes

```
// Get a context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
try {
DirContext dirCtx = new InitialDirContext(env);

// Remove the entry
dirCtx.destroySubContext("uid=jbrown, ou=LearningServices, o=IBM");

} catch(NamingException ne) {ne.printStackTrace();}
```

JAVA NAMING AND DIRECTORY INTERFACE

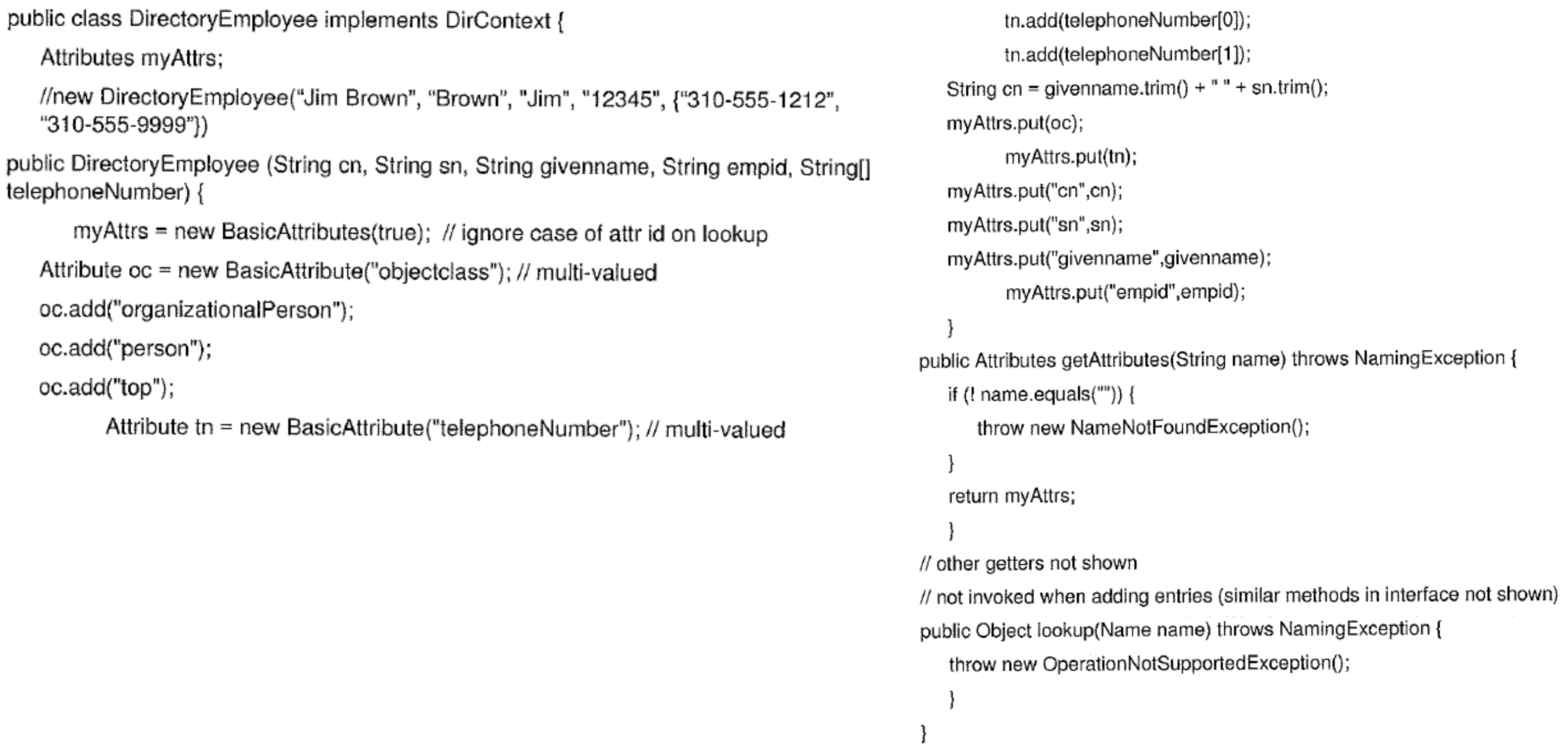


Adding an Entry

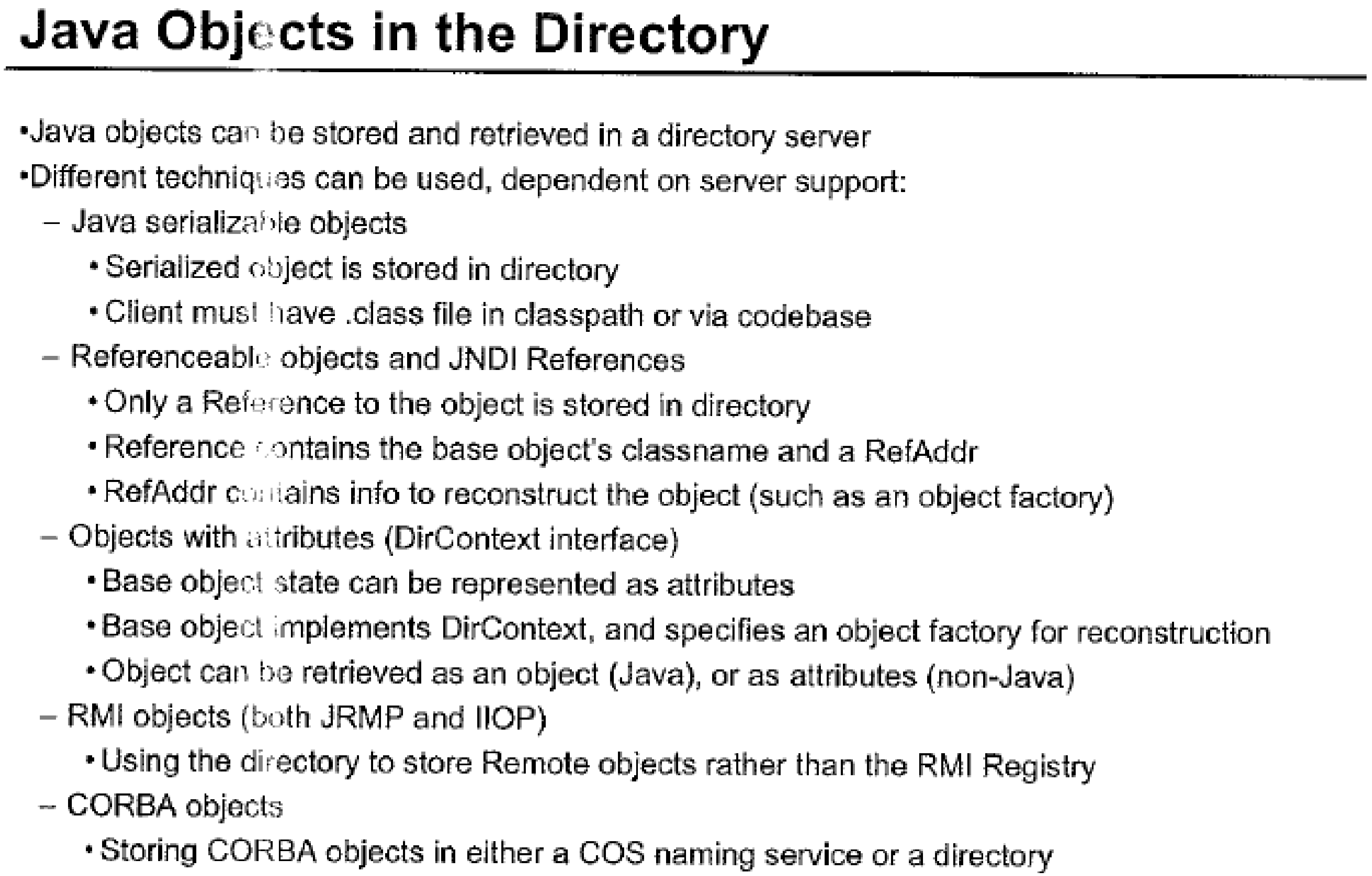
- An entry is added by adding a new DN to the directory
- Entry/attributes to be added are defined in a class built specifically for this, which implements DirContext
- This class will primarily implement the
  - Constructor that will take parameters needed to supply the attribute-value pairs
  - getAttributes() that returns the Attributes object
- Instantiate the class, and then bind it to the DN

```
// Get a context
try {
DirContext dirCtx = new InitialDirContext(env);

// Construct class that contains all the attributes
DirectoryEmployee emp = new DirectoryEmployee("Jim Brown", "Brown",
         "Jim", "12345", {"310-555-1212", "310-555-9999"});

// bind the entry
dirCtx.bind("uid=jbrown, ou=LearningServices, o=IBM", emp);
} catch(NamingException ne) {ne.printStackTrace();}
```

JAVA NAMING AND DIRECTORY INTERFACE



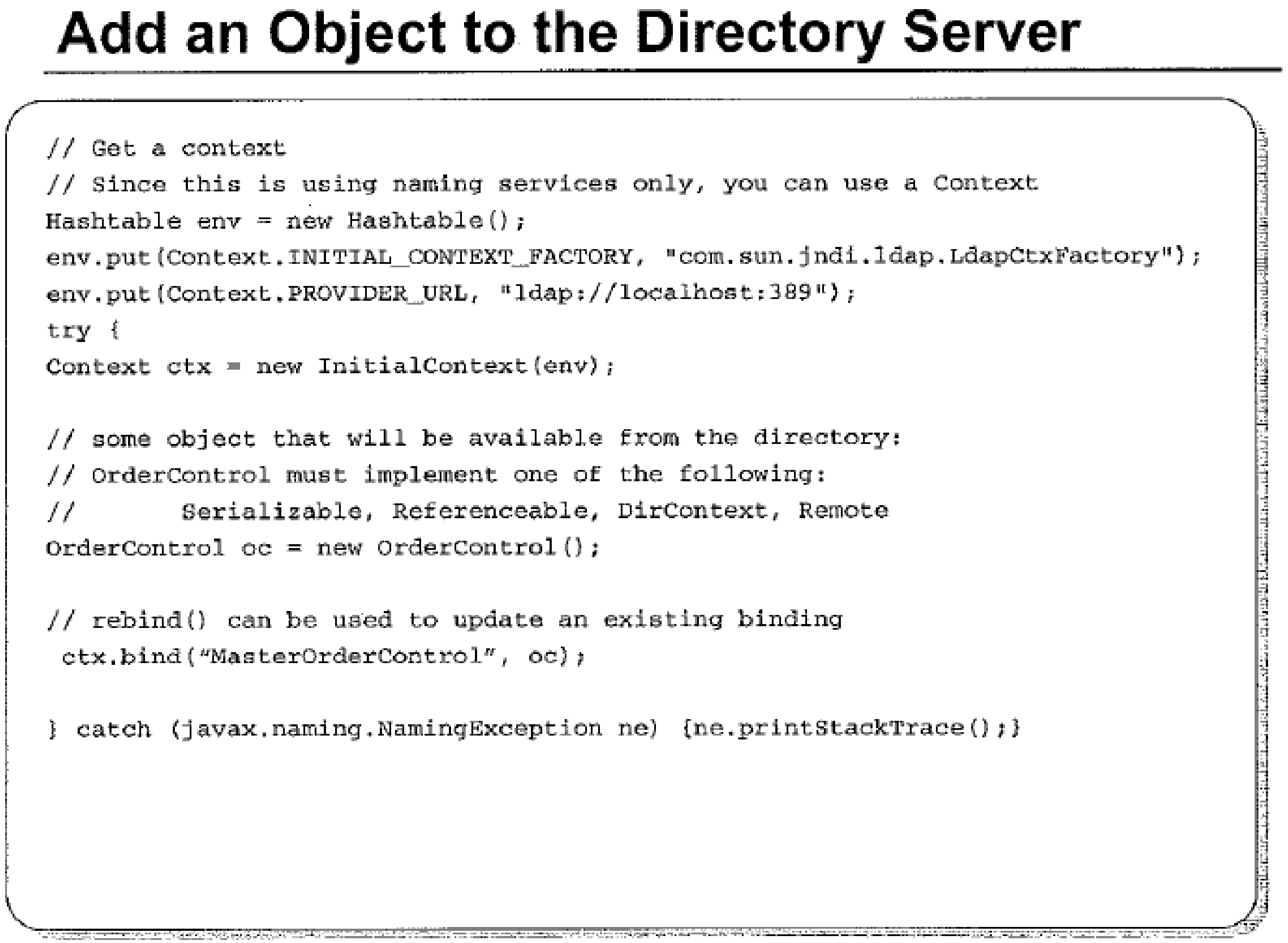
```
public class DirectoryEmployee implements DirContext {
    Attributes myAttrs;
    //new DirectoryEmployee("Jim Brown", "Brown", "Jim", "12345", {"310-555-1212",
    "310-555-9999"})
public DirectoryEmployee (String cn, String sn, String givenname, String empid, String[]
telephoneNumber) {
        myAttrs = new BasicAttributes(true);  // ignore case of attr id on lookup
    Attribute oc = new BasicAttribute("objectclass"); // multi-valued
    oc.add("organizationalPerson");
    oc.add("person");
    oc.add("top");
            Attribute tn = new BasicAttribute("telephoneNumber"); // multi-valued
            tn.add(telephoneNumber[0]);
            tn.add(telephoneNumber[1]);
    String cn = givenname.trim() + " " + sn.trim();
    myAttrs.put(oc);
            myAttrs.put(tn);
    myAttrs.put("cn",cn);
    myAttrs.put("sn",sn);
    myAttrs.put("givenname",givenname);
            myAttrs.put("empid",empid);
    }
public Attributes getAttributes(String name) throws NamingException {
    if (! name.equals("")) {
        throw new NameNotFoundException();
    }
    return myAttrs;
    }
// other getters not shown
// not invoked when adding entries (similar methods in interface not shown)
public Object lookup(Name name) throws NamingException {
    throw new OperationNotSupportedException();
    }
}
```



Java Objects in the Directory

- Java objects can be stored and retrieved in a directory server
- Different techniques can be used, dependent on server support:
  - Java serializable objects
    - Serialized object is stored in directory
    - Client must have .class file in classpath or via codebase
  - Referenceable objects and JNDI References
    - Only a Reference to the object is stored in directory
    - Reference contains the base object's classname and a RefAddr
    - RefAddr contains info to reconstruct the object (such as an object factory)
  - Objects with attributes (DirContext interface)
    - Base object state can be represented as attributes
    - Base object implements DirContext, and specifies an object factory for reconstruction
    - Object can be retrieved as an object (Java), or as attributes (non-Java)
  - RMI objects (both JRMP and IIOP)
    - Using the directory to store Remote objects rather than the RMI Registry
  - CORBA objects
    - Storing CORBA objects in either a COS naming service or a directory

JAVA NAMING AND DIRECTORY INTERFACE

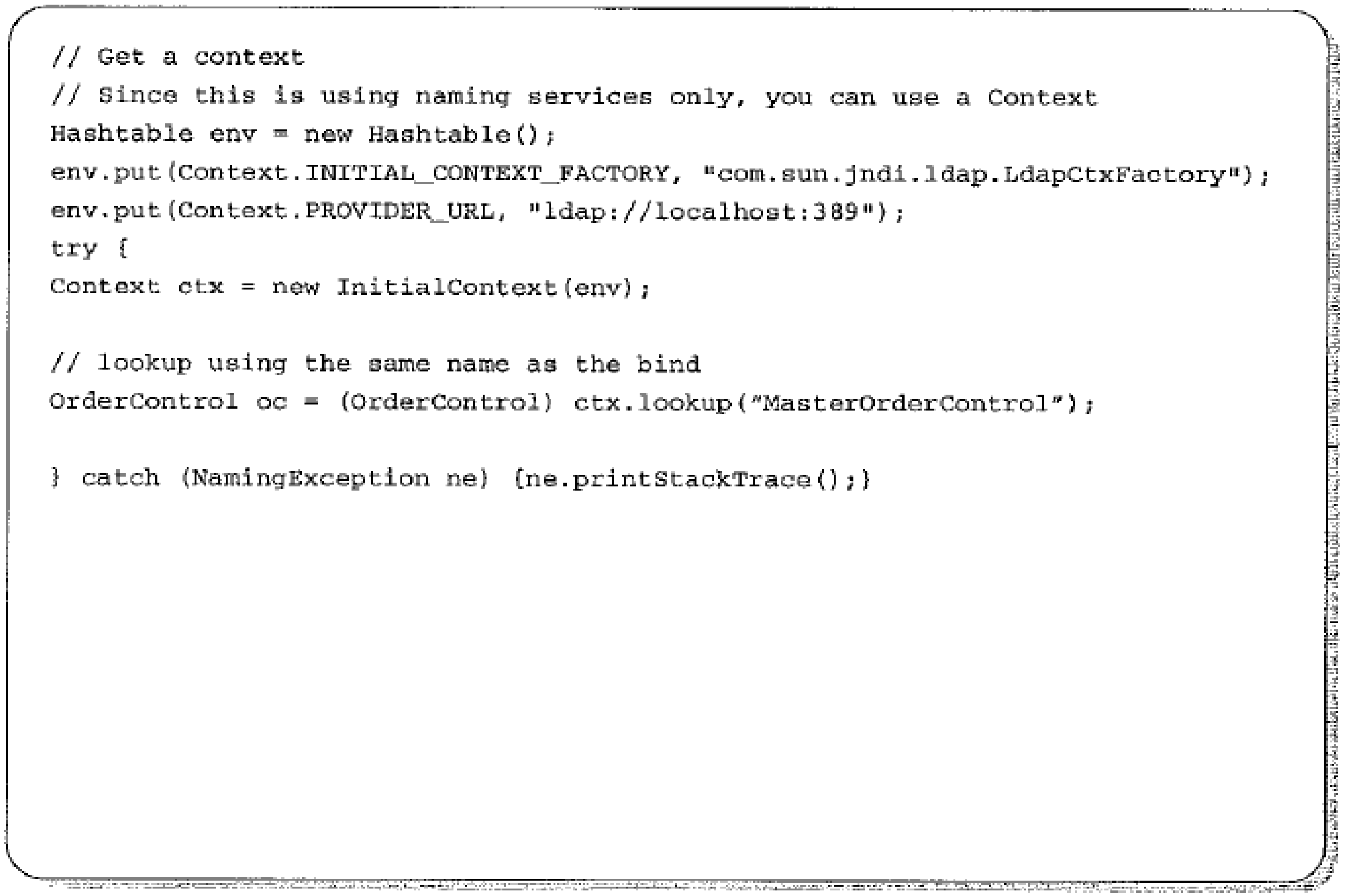


Add an Object to the Directory Server

```
// Get a context
// Since this is using naming services only, you can use a Context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
try {
Context ctx = new InitialContext(env);

// some object that will be available from the directory:
// OrderControl must implement one of the following:
//       Serializable, Referenceable, DirContext, Remote
OrderControl oc = new OrderControl();

// rebind() can be used to update an existing binding
 ctx.bind("MasterOrderControl", oc);

} catch (javax.naming.NamingException ne) {ne.printStackTrace();}
```

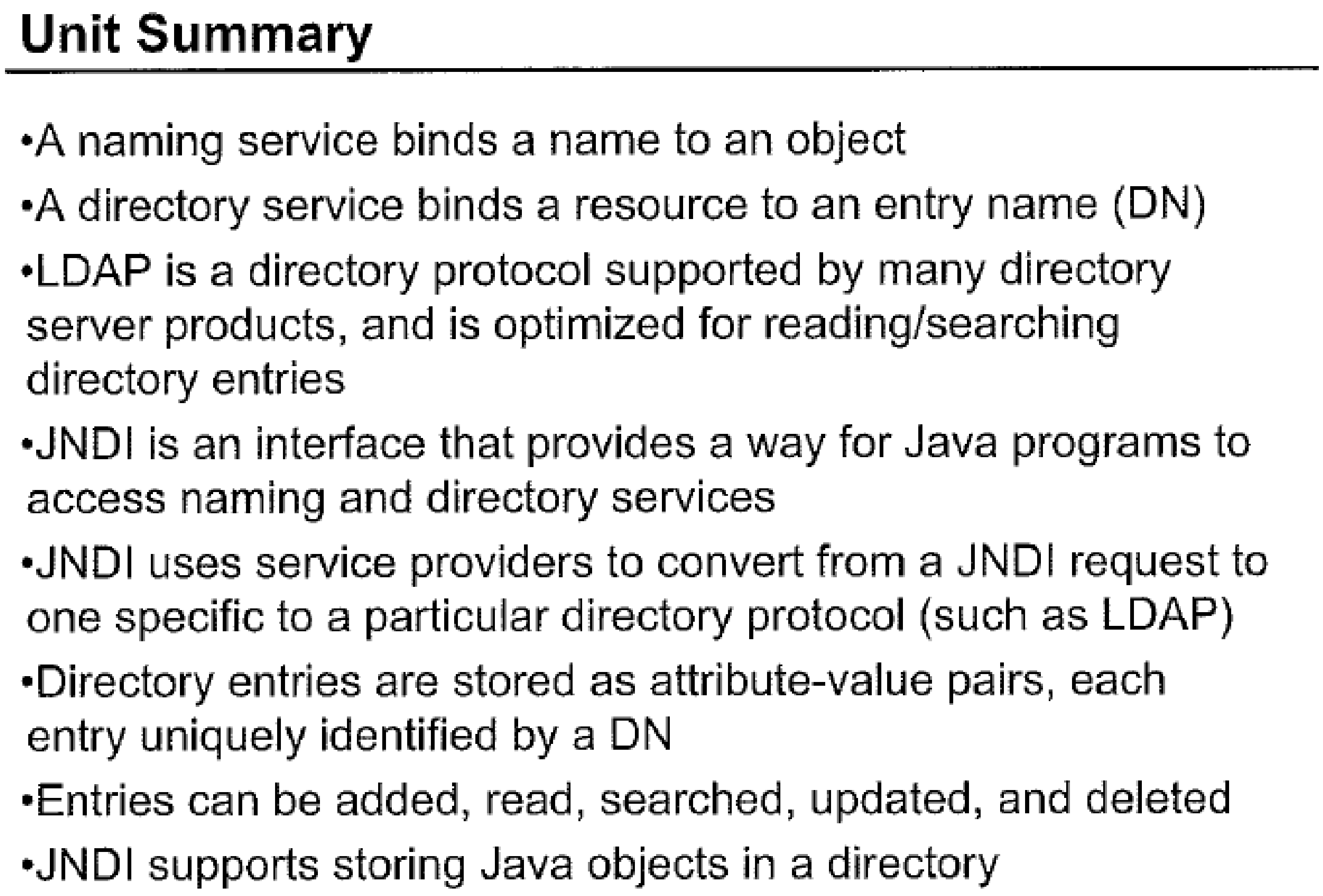
JAVA NAMING AND DIRECTORY INTERFACE

Retrieve an Object from the Directory Server



```
// Get a context
// Since this is using naming services only, you can use a Context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
try {
Context ctx = new InitialContext(env);

// lookup using the same name as the bind
OrderControl oc = (OrderControl) ctx.lookup("MasterOrderControl");

} catch (NamingException ne) {ne.printStackTrace();}
```

Java Naming and Directory Interface



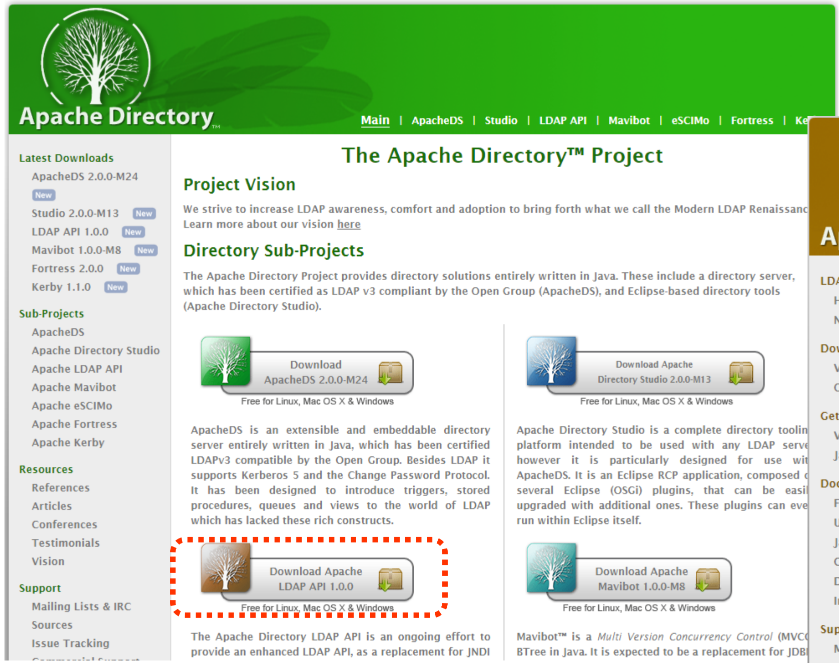
Unit Summary

- A naming service binds a name to an object
- A directory service binds a resource to an entry name (DN)
- LDAP is a directory protocol supported by many directory server products, and is optimized for reading/searching directory entries
- JNDI is an interface that provides a way for Java programs to access naming and directory services
- JNDI uses service providers to convert from a JNDI request to one specific to a particular directory protocol (such as LDAP)
- Directory entries are stored as attribute-value pairs, each entry uniquely identified by a DN
- Entries can be added, read, searched, updated, and deleted
- JNDI supports storing Java objects in a directory

JAVA NAMING AND DIRECTORY INTERFACE

The Apache Directory Project

http://directory.apache.org


Apache Directory
Main | ApacheDS | Studio | LDAP API | Mavibot | eSCIMo | Fortress | Ke
Download ApacheDS 2.0.0-M24
Download Apache Directory Studio 2.0.0-M13
Download Apache LDAP API 1.0.0
Download Apache Mavibot 1.0.0-M8
Free for Linux, Mac OS X & Windows

Latest Downloads
ApacheDS 2.0.0-M24 New
Studio 2.0.0-M13 New
LDAP API 1.0.0 New
Mavibot 1.0.0-M8 New
Fortress 2.0.0 New
Kerby 1.1.0 New

Sub-Projects
ApacheDS
Apache Directory Studio
Apache LDAP API
Apache Mavibot
Apache eSCIMo
Apache Fortress
Apache Kerby

Resources
References
Articles
Conferences
Testimonials
Vision

Support
Mailing Lists & IRC
Sources
Issue Tracking

The Apache Directory™ Project

Project Vision

We strive to increase LDAP awareness, comfort and adoption to bring forth what we call the Modern LDAP Renaissanc
Learn more about our vision here

Directory Sub-Projects

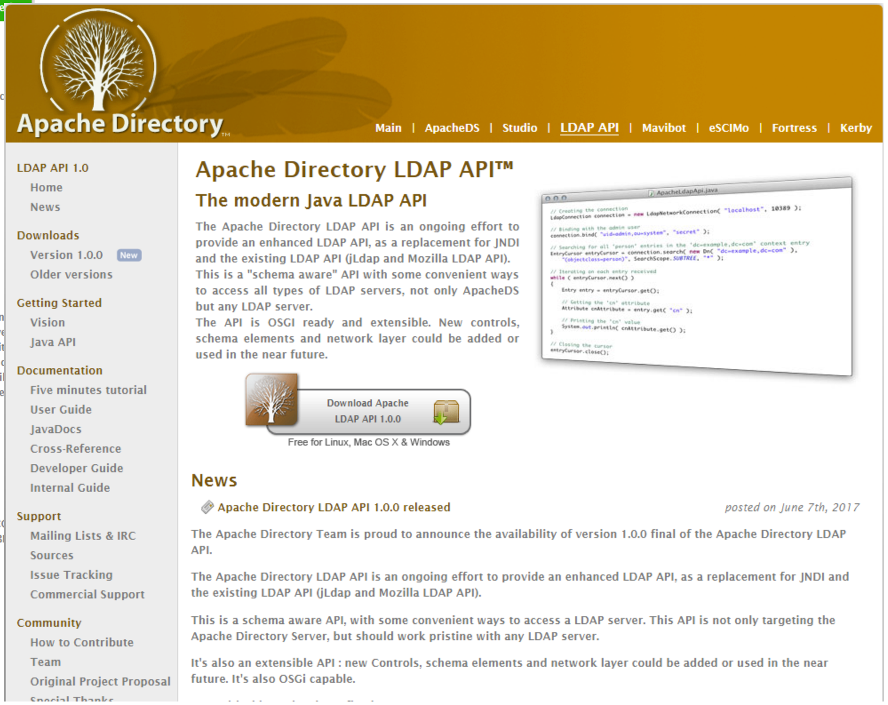
The Apache Directory Project provides directory solutions entirely written in Java. These include a directory server, which has been certified as LDAP v3 compliant by the Open Group (ApacheDS), and Eclipse-based directory tools (Apache Directory Studio).

ApacheDS is an extensible and embeddable directory server entirely written in Java, which has been certified LDAPv3 compatible by the Open Group. Besides LDAP it supports Kerberos 5 and the Change Password Protocol. It has been designed to introduce triggers, stored procedures, queues and views to the world of LDAP which has lacked these rich constructs.

Apache Directory Studio is a complete directory toolin platform intended to be used with any LDAP serve however it is particularly designed for use wi ApacheDS. It is an Eclipse RCP application, composed o several Eclipse (OSGi) plugins, that can be easi upgraded with additional ones. These plugins can eve run within Eclipse itself.

The Apache Directory LDAP API is an ongoing effort to provide an enhanced LDAP API, as a replacement for JNDI

Mavibot™ is a *Multi Version Concurrency Control* (MVC BTree in Java. It is expected to be a replacement for JDB


Apache Directory
Main | ApacheDS | Studio | LDAP API | Mavibot | eSCIMo | Fortress | Kerby
ApacheLdapApi.java
Download Apache LDAP API 1.0.0
Free for Linux, Mac OS X & Windows

LDAP API 1.0
Home
News

Downloads
Version 1.0.0 New
Older versions

Getting Started
Vision
Java API

Documentation
Five minutes tutorial
User Guide
JavaDocs
Cross-Reference
Developer Guide
Internal Guide

Support
Mailing Lists & IRC
Sources
Issue Tracking
Commercial Support

Community
How to Contribute
Team
Original Project Proposal

Apache Directory LDAP API™

The modern Java LDAP API

The Apache Directory LDAP API is an ongoing effort to provide an enhanced LDAP API, as a replacement for JNDI and the existing LDAP API (jLdap and Mozilla LDAP API).
This is a "schema aware" API with some convenient ways to access all types of LDAP servers, not only ApacheDS but any LDAP server.
The API is OSGI ready and extensible. New controls, schema elements and network layer could be added or used in the near future.

```
// Creating the connection
LdapConnection connection = new LdapNetworkConnection( "localhost", 10389 );

// Binding with the admin user
connection.bind( "uid=admin,ou=system", "secret" );

// Searching for all 'person' entries in the 'dc=example,dc=com' context entry
EntryCursor entryCursor = connection.search( new Dn( "dc=example,dc=com" ),
    "(objectclass=person)", SearchScope.SUBTREE, "*" );

// Iterating on each entry received
while ( entryCursor.next() )
{
    Entry entry = entryCursor.get();

    // Getting the 'cn' attribute
    Attribute cnAttribute = entry.get( "cn" );

    // Printing the 'cn' value
    System.out.println( cnAttribute.get() );
}

// Closing the cursor
entryCursor.close();
```

News

Apache Directory LDAP API 1.0.0 released — *posted on June 7th, 2017*

The Apache Directory Team is proud to announce the availability of version 1.0.0 final of the Apache Directory LDAP API.

The Apache Directory LDAP API is an ongoing effort to provide an enhanced LDAP API, as a replacement for JNDI and the existing LDAP API (jLdap and Mozilla LDAP API).

This is a schema aware API, with some convenient ways to access a LDAP server. This API is not only targeting the Apache Directory Server, but should work pristine with any LDAP server.

It's also an extensible API : new Controls, schema elements and network layer could be added or used in the near future. It's also OSGi capable.

- **Apache Directory LDAP API**

  http://directory.apache.org/api/

KONIEC

DZIĘKUJĘ ZA UWAGĘ!!!

Szczecin, 12 maja 2018 r.