# Directory Servers

**A directory server is a type of network database that stores information represented as trees of entries.**

**This is different from a relational database, which uses tables comprised of rows and columns, so directory servers may be considered a type of NoSQL database.**

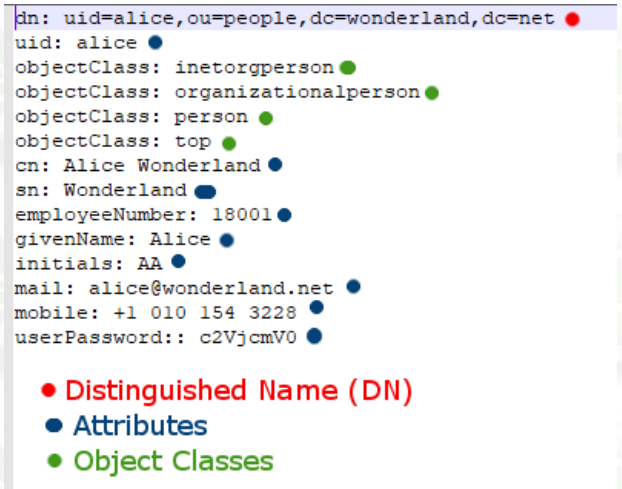
**It is typically employed to store user information and passwords, simplifying account administration and centralizing credentials authentication.**

**It is accessed using LDAP (Lightweight Directory Access Protocol)**

# Entries

An entry (stored data) is a collection of information about an entity (i.e. person), and has three primary components:

- A distinguished name (DN)
- A collection of attributes (key-value pairs)
- A collection of object classes



# Distinguished Name (DN)

**Uniquely identifies an entry and its position in the directory information tree (DIT) hierarchy.**

**It is comprised of zero or more elements called relative distinguished names (RDNs), which contain one or several key-value pairs (attributes).**

**For example, the DN**

**“uid=alice,ou=people,dc=wonderland,dc=net”**

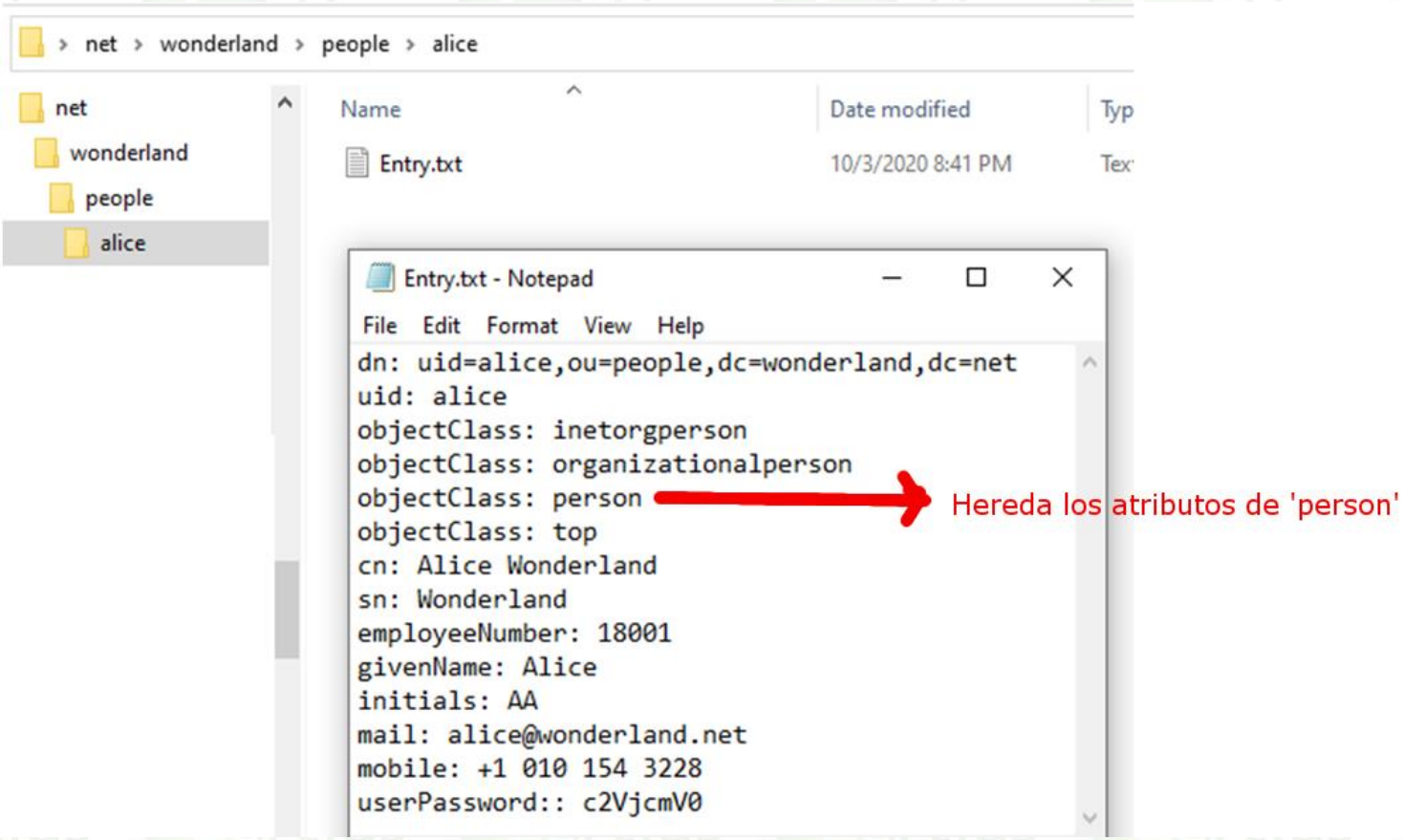
**has four RDNs:**

| | |
|---|---|
| **uid=alice** | ‘user id’ |
| **ou=people** | ‘organizational unit’ |
| **dc=wonderland** | ‘domain component’ |
| **dc=net** | ‘domain component’ |

# DIT vs Filesystem

*<------- LOWER -------- HIERARCHY --------- HIGHER -------->*

**“uid=alice,ou=people,dc=wonderland,dc=net”**



# Entries

```
dn: uid=alice,ou=people,dc=wonderland,dc=net
uid: alice
objectClass: inetorgperson
objectClass: organizationalperson
objectClass: person
objectClass: top
cn: Alice Wonderland
sn: Wonderland
employeeNumber: 18001
givenName: Alice
initials: AA
mail: alice@wonderland.net
mobile: +1 010 154 3228
userPassword:: c2VjcmV0
```

- Distinguished Name (DN)
- Attributes
- Object Classes

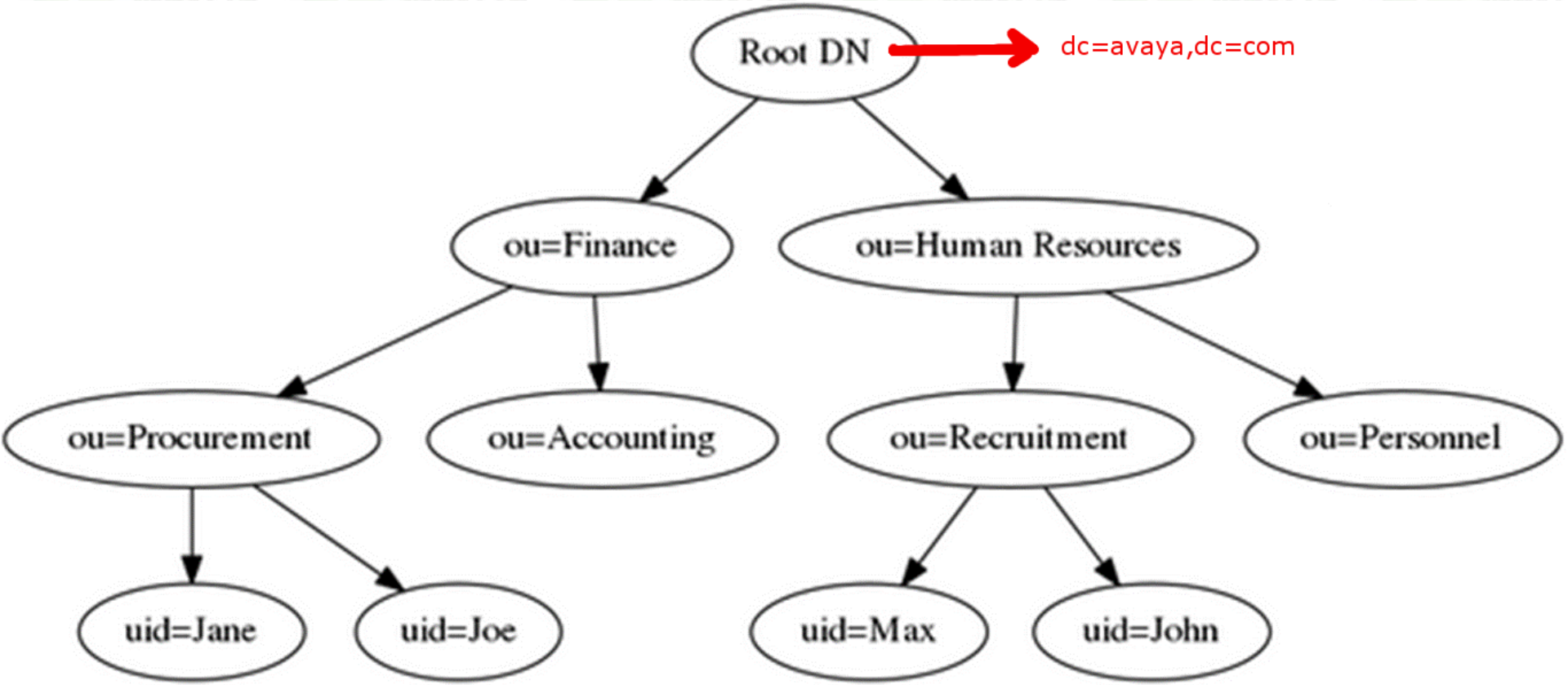
## An entry has

**A distinguished name (DN)**

**A collection of attributes**

**A collection of object classes**

# Example of DIT



# LDAP operations

| Operation | Description |
|---|---|
| Search | Allows the client to request the server to search for the certain part of DIT for information matching user-defined criteria and list the result(s). |
| Compare | Request the server to compare an entry for an attribute value. |
| Update (It can perform any of the following operations:) | Request to modify the content of the directory. |
| Delete | Delete existing entries from the directory |
| Add | Insert new entries in the directory |
| Modify | Change the attributes and values present in the existing entry |
| Bind | Initiates an LDAP session between a client and a server. |
| Unbind | Terminates a client/server session. |
| Abandon | Allows a client to request that the server abandon an outstanding operation. |

# LDAP security

## Access variants:

- ldap://domain.com -> basic LDAP protocol
- ldaps://domain.com -> LDAP over SSL/TLS
- ldapi://domain.com -> This is used to indicate LDAP over internal sockets instead of using an exposed network port.

# LDAP authentication

There are three options for binding:

- No Authentication
- Basic Authentication – The LDAP client is required to provide a DN and a password for authentication (the credentials are sent over cleartext, meaning they can be easily read by an unauthorized party if one were to infiltrate their session).
- SASL – Simple Authentication and Security Layer, or SASL, is a protocol that requires both the client and server to provide identifying information.

# Java connect - as simple as this:

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.directory.Attributes;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;

public class Test1 {

    public static DirContext connectToLDAP(String url, String user, String password) throws NamingException{
          Hashtable<String, String> env = new Hashtable<String, String>();
          env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
          env.put(Context.PROVIDER_URL, url);
          env.put(Context.SECURITY_AUTHENTICATION,"simple");
          env.put(Context.SECURITY_PRINCIPAL, user);
          env.put(Context.SECURITY_CREDENTIALS,password);
          //Conseguimos contexto de conexion
          DirContext ctx = new InitialDirContext(env);
          return ctx;
    }
```

**Supported directly by Java (javax)**

**No 3rd Party components**

# LDAP search & filters

## How do I match more than one attribute?

For example, if my users are distinguished by having two **objectClass** attributes (one equal to 'person' and another to 'user'), this is how I would match for it:

```
(&(objectClass=person)(objectClass=user))
```

Notice the `ampersand symbol '&'` symbol at the start. Translated this means: *search for objectClass=person AND object=user.*

Alternatively,

```
(|(objectClass=person)(objectClass=user))
```

Translated this means: *search for objectClass=person OR object=user.*

The `pipe symbol '|'` denotes 'OR'. As this is not a special XML character, it should not need escaping.

# LDAP search & filters

## Wildcards

```
(&(objectClass=user)(cn=*Marketing*))
```

## How do I match 3 attributes?

Just add an extra clause:

```
(&(objectClass=user)(objectClass=top)(objectClass=person))
```

## Using 'not'

To exclude entities which match an expression, use '!'.

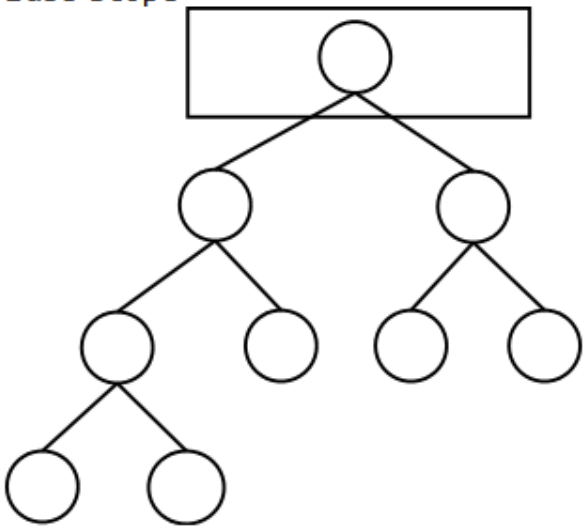
So

```
(&(objectClass=group)(&(ou:dn:=Chicago)(!(ou:dn:=Wrigleyville))))
```

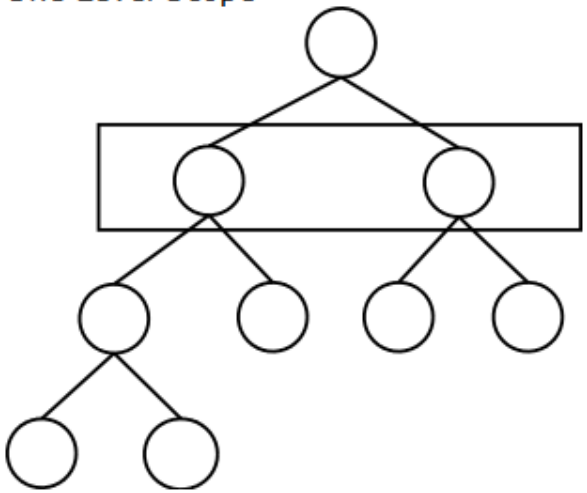
# LDAP search scope

- 3 types of scope:
  - base - limits to just the base object
  - onelevel - limits to just the immediate children
  - sub - search the entire subtree from the base object down

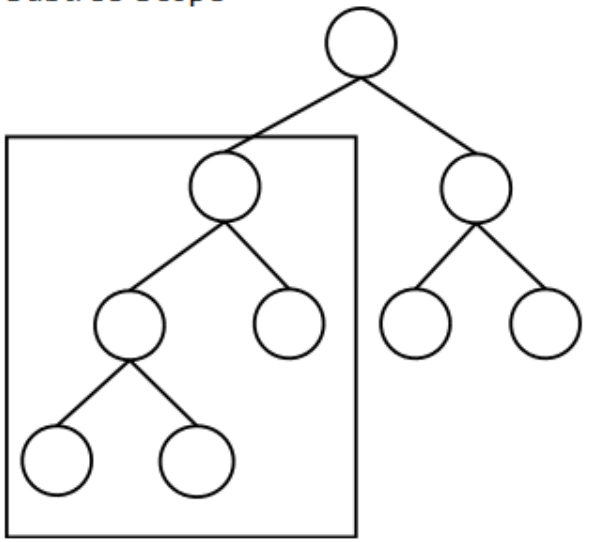
Base Scope



One Level Scope



Subtree Scope



# Java search - as simple as this:

```
try {
    ctx = connectToLDAP("ldap://ldap.forumsys.com:389", "cn=read-only-admin,dc=example,dc=com", "password");

    String filter = "(objectclass=*)";
    SearchControls ctrl = new SearchControls();
    ctrl.setSearchScope(SearchControls.SUBTREE_SCOPE);
    NamingEnumeration<SearchResult> answer = ctx.search("dc=example,dc=com", filter, ctrl);

    String dn;
    while (answer.hasMore()) {
        SearchResult result = (SearchResult) answer.next();
        dn = result.getNameInNamespace();
        Attributes att = result.getAttributes();
        System.out.println("");
        System.out.println("Entry     : " + dn);
        System.out.println("Attributes: " + att.toString());
    }
    answer.close();

} catch (NamingException e) {
    e.printStackTrace();
} finally {
    closeConnectionToLDAP(ctx);
}
```

# LDAP retrieve specific attributes

```
String filter = "(objectclass=*)";
String[] returninAttributes = { "uid", "cn" };

SearchControls ctrl = new SearchControls();
ctrl.setReturningAttributes(returninAttributes);
ctrl.setCountLimit(500);

ctrl.setSearchScope(SearchControls.SUBTREE_SCOPE);
NamingEnumeration<SearchResult> answer = ctx.search("dc=example,dc=com", filter, ctrl);

String dn;
while (answer.hasMore()) {
    SearchResult result = (SearchResult) answer.next();
    dn = result.getNameInNamespace();
    Attributes att = result.getAttributes();
    System.out.println("");
    System.out.println("Entry      : " + dn);
    System.out.println("Attributes: " + att.toString());
}
answer.close();
```

# Sample Java Code

Test1.java