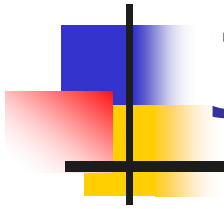
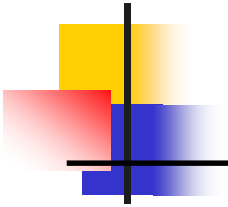


JNDI and LDAP – Part I

Noel J. Bergman

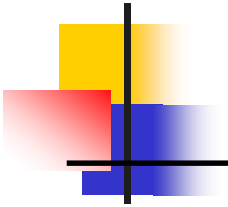
DevTech®



Session Overview

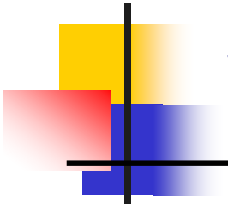
JNDI is commonly used as the general mechanism for exposing resources to J2EE applications. This session will go beyond this common but simple use of JNDI, and introduce you to LDAP – the Lightweight Directory Access Protocol. This powerful technology unleashes the full power of the JNDI API. You will learn how to expose, access, search for, and modify information, services and other resources in LDAP using JNDI.

PLEASE ASK QUESTIONS! ☺



Session Prerequisites

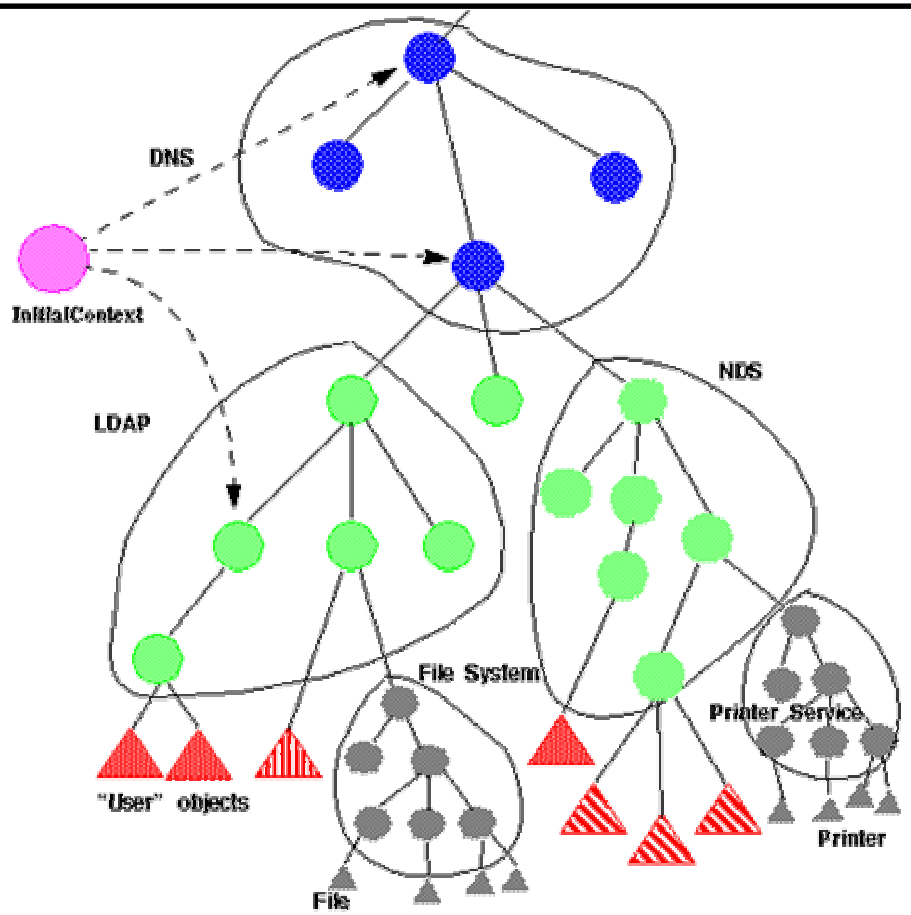
- You will want a good understanding of Java.
- To run most of the sample code, you'll need to install a suitable LDAP server.



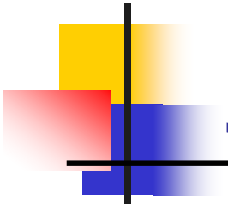
What Is JNDI

- "…a unified interface to multiple naming and directory services …"
- A simple, scalable, hierarchical, searchable, object-oriented data model combining aspects of both hierarchical and relational databases.
- An extensible service-provider based package capable of operating with arbitrary data types and access mechanisms.

The World According To JNDI

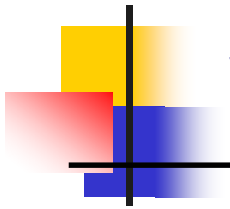


http://java.sun.com/j2se/1.4.1/docs/guide/jndi/spec/jndi/jndi-2.gif



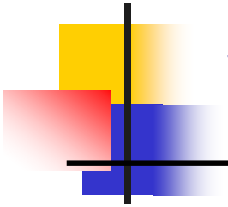
JNDI Can Interface With …

- LDAP
- OMG COS Naming
- RMI Registry
- NIS
- DSML
- DNS
- Novell Directory Services
- Windows Registry
- JSR-57
- In-Memory
  - typically: `java:comp/env`
- Smart Cards
- HTML
- File System
- The Domain Registry



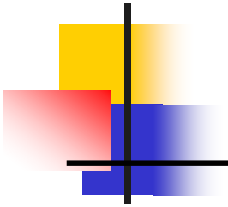
When to Use JNDI

- Anywhere you can make use of a scalable, searchable, hierarchical database interface.
- Application Configuration
- User Information
  - Access control
  - Contact info
  - *etc.*
- Public Network Information and Services



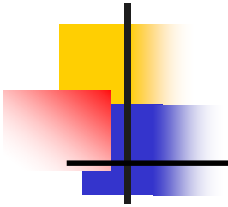
What Can You Store?

- `javax.naming.Reference`
- `javax.naming.Referenceable`
- `java.io.Serializable`
- `javax.naming.DirContext`
- Is that it?
  - No. We'll talk about how this all works, and the role of State Factories and Object Factories.



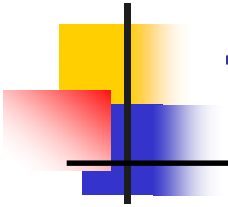
Major Interfaces

- `javax.naming.Context`
- `javax.naming.NamingEnumeration`
- `javax.naming.Directory.DirContext`
- `javax.naming.Directory.Attribute`
- `javax.naming.Directory.Attributes`
- `javax.naming.Directory.SearchControls`



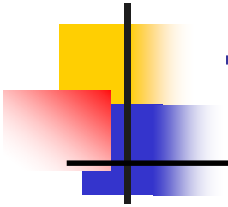
Major Operations

- Bind
- Rebind
- Unbind
- Lookup
- Rename
- List
- List Bindings
- Create Subcontext
- Destroy Subcontext
- Search
- Get Attributes
- Modify Attributes



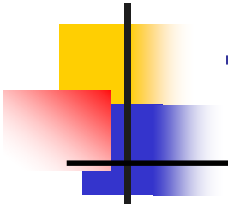
The Context

- `javax.naming.Context`
- Context, or the DirContext subclass, provides the primary interface for JNDI clients.
- Represents a node in the tree.
  - All interior nodes
  - Often leaf nodes, especially with DirContext
- Is hierarchically named *via* the path from the tree root.



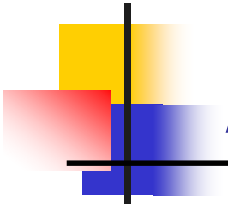
The DirContext

- `javax.naming.Directory.DirContext`
- Extends `javax.naming.Context`
- Adds
  - Attributes
  - Searching
  - Schema
- The most useful and used JNDI interface.



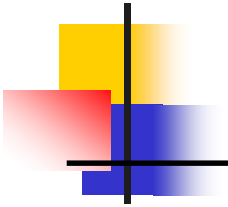
The Initial Context

- You gain access to the root of your tree (*aka* DIT) by instantiating an initial context. They are represented by `InitialContext` and `InitialDirContext`, respectively.
- Once you've instantiated the initial context, you just deal with `Context` or `DirContext` objects.
- Let's take a quick peak at three examples of setting up an initial context, including code for a very short but complete DNS Lookup.



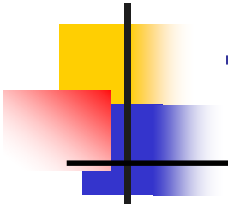
A Node by Any Other Name …

- JNDI operations generally come in two flavors:
  - A version that takes a `Name` to identify the target.
  - A version that takes a `String` to identify the target.
- Wherever you see one, assume the other as well.
- When writing JNDI client code, you will generally find the `String` flavor more appropriate.
- When writing a JNDI extension, you will generally find the `Name` flavor more helpful.
- These are hierarchical addresses, whose syntax is service-provider dependent.



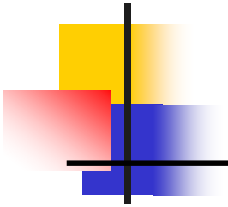
LDAP Names

- RFC 2253 defines the string representation of LDAP names.
- A Distinguished Name (DN) is composed of a sequence of Relatively Distinguished Names. A Relatively Distinguished Name (RDN) is a name that is unique within the context in which it is bound. RFC 2253 names are read from left to right. LDAP attributes are used for each RDN.
- X.500, on which LDAP is based, suggests but does not mandate how you compose your hierarchical namespace. RFC 2247 provides a useful approach for the Internet, using domain names, *e.g.*:
  - dc=apache,dc=org
  - dc=multitask,dc=com,dc=au



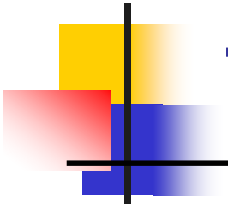
The Context

- `javax.naming.Context`
- Context, or the DirContext subclass, provides the primary interface for JNDI clients.
- Represents a node in the tree:
  - All interior nodes
  - Often leaf nodes, especially with DirContext
- Is hierarchically named *via* the path from the tree root.



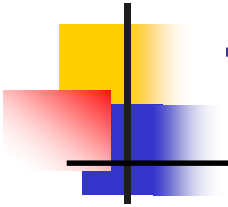
Context Operations

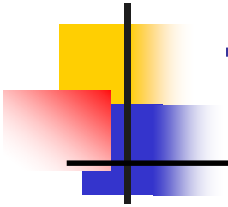
- Bind
- Rebind
- Unbind
- Lookup
- Rename
- List
- List Bindings
- Create Subcontext
- Destroy Subcontext
- Lookup Link
- Get Name In Namespace
- Get Name Parser
- Compose Name
- Add To Environment
- Remove From Environment
- Get Environment



The `bind` Operation

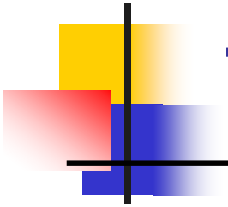
- ```
  void bind(String name,
            Object obj)
    throws NamingException;
  ```
- Associates an object as a named descendent node of this Context.
- Fundamental operation for building the tree.
- Throws an exception if that name is already bound to this Context.


  ```



The `rebind` Operation

- ```
  void rebind(String name,
              Object obj)
    throws NamingException;
  ```
- Associates a new object as a named descendent node of this Context.
- If re*name* changes a *name*, think of this as re*value*.


  ```



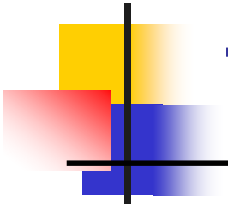
The `unbind` Operation

- `void unbind(String name)`
  `throws NamingException;`
- Removes a named descendent node of this Context.



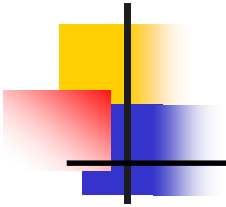
The `lookup` Operation

- `Object lookup(String name)`
  `throws NamingException;`
- Returns the named descendent node of this Context as an object.



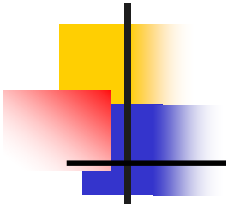
The `rename` Operation

- `void rename(String from,`
  `String to)`
  `throws NamingException;`
- Associates a new name with a named descendent node of this Context.



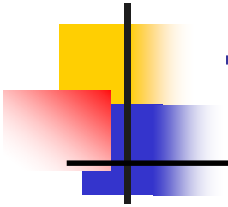
The `list` Operation

- `NamingEnumeration list(String name) throws NamingException;`
- Returns a `NamingEnumeration` providing all of the descendents of this Context.
  - The elements of the enumeration provide just the name of each descendent and its class, in the form of `NameClassPair` instances.
- This is lighter weight than `listBindings`.



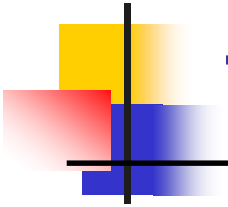
The `listBindings` Operation

- `NamingEnumeration listBindings(String name) throws NamingException;`
- Returns a `NamingEnumeration` providing all of the descendents of this Context.
  - The elements of the enumeration provide the name of each descendent, its class, and object, in the form of `Binding` instances.
- This is heavier weight than `list` due to the instantiation of the descendents.



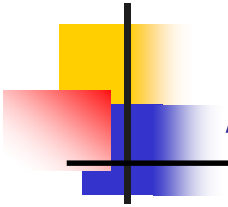
The `createSubcontext` Operation

- `Context createSubcontext(String name)`
  `throws NamingException;`
- Returns a Context bound to the name.
- This is used to build sub-trees.



The `destroySubcontext` Operation

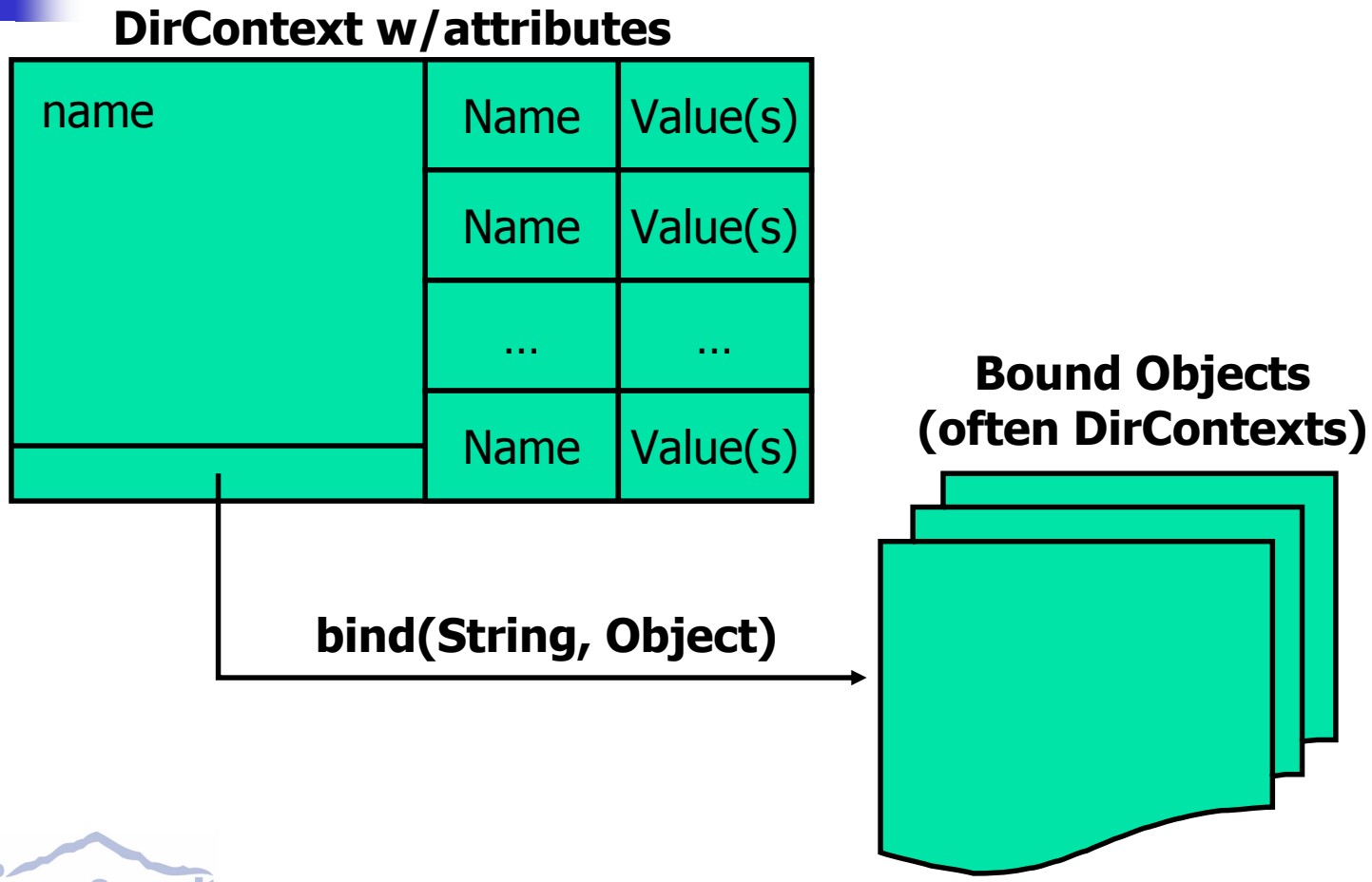
- `void destroySubcontext(String name) throws NamingException;`
- Removes the Context bound to the name.
- Throws an exception if the Context is not empty.

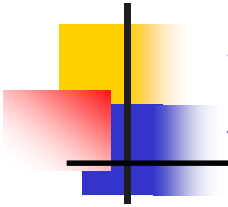


Attributes

- We would like to associate information with a node in the tree.
- The associated information is called an *attribute*.
- An attribute is a [name, object*]-tuple.
- For example, a file node might have creation, modification, and access-control information as attributes, none of which is part of the file's content.
- JNDI, as a generic data access mechanism, does not particularly care about what is stored as an attribute, but underlying service providers will.

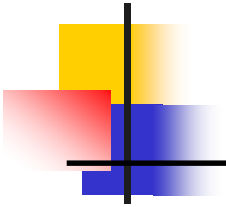
DirContext and Attributes





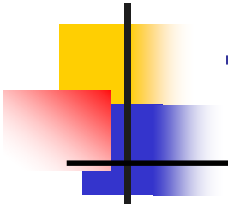
X.500, LDAP and Schema

- Although JNDI does not impose any schema requirement upon the DIT, the underlying service provider, *e.g.*, LDAP, may.
- JNDI draws from X.500 and simpler LDAP technologies.
- LDAP schema define attributes and "object classes", which are sets of related attributes.
- Our primary DirContext demo uses LDAP, and provides many examples of attributes and schema.



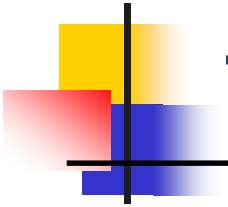
Important LDAP Schema

- LDAP Schema are standardized by RFC
  - RFC 2252 Core LDAP Schema
  - RFC 2256 X.500 User Schema for LDAP
  - RFC 2713 Java Objects
  - RFC 2714 CORBA Objects
  - RFC 2798 inetOrgPerson
- Each LDAP server has some way of telling it what schema are available. Most allow you to turn off schema enforcement, or allow the use of `extensibleObject`.



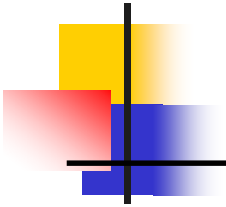
Typical LDAP Paradigm

- The standard LDAP technology is a common mechanism for storing information *via* JNDI.
- An object such as a person, place or thing will be represented as a `DirContext`.
- The `DirContext` will have a multi-valued attribute called `objectClass` that defines the sets of mandatory and optional attributes.
- The `DirContext` will have other attributes that define its semantic properties.



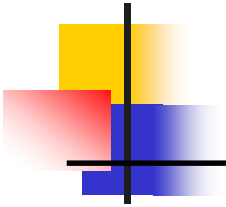
The DirContext

- `javax.naming.Directory.DirContext`
- Extends `javax.naming.Context`
- Adds
  - Attributes
  - Searching
  - Schema
- The most useful and used JNDI interface



DirContext Operations

- Bind
- Rebind
- Create Subcontext
- Destroy Subcontext
- Search
- Get Attributes
- Modify Attributes
- Get Schema
- Unbind
- Lookup
- Rename
- List
- List Bindings
- Lookup Link
- Get Name In Namespace
- Get Name Parser
- Compose Name
- Add To Environment
- Remove From Environment
- Get Environment

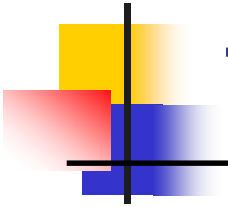


DirContext Operations

- Bind
- Rebind
- Create Subcontext
- Destroy Subcontext
- Search
- Get Attributes
- Modify Attributes
- Get Schema

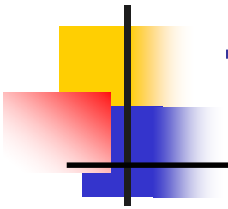
- Identical to the methods we are familiar with from Context, with the addition of a new parameter:
  - `Attributes attrs`
  - *e.g.*,

```
void bind(Name,
          Object,
          Attributes)
```



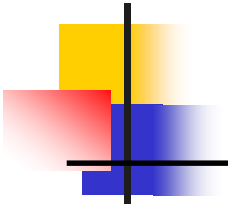
The `Attributes` Interface

- `javax.naming.directory.Attributes`
- A collection of Attribute objects.
- Basic Operations:
  - `Attribute get(String)`
  - `NamingEnumeration getAll()`
  - `NamingEnumeration getIDs()`
  - `Attribute put(Attribute)`
  - `Attribute put(String, Object)`
  - `Attribute remove(String)`



The `Attribute` Interface

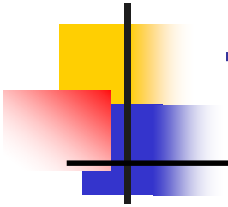
- `javax.naming.directory.Attribute`
- A single, possibly multi-valued, Attribute.
- Basic Operations:
  - `boolean add(Object)`
  - `Object get()`
  - `NamingEnumeration getAll()`
  - `String getID()`
  - `boolean remove(Object)`



DirContext Operations

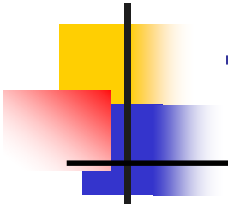
- Bind
- Rebind
- Create Subcontext
- Destroy Subcontext
- Search
- Get Attributes
- Modify Attributes
- Get Schema

- These are new methods for dealing with attributes and schema.



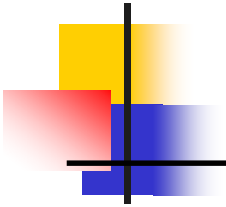
The `getAttributes` Operation

- `Attributes getAttributes(String name)`
  `throws NamingException;`
- Returns all attributes for the named node.
- A second form allows you to specify which attributes are to be fetched.
  - `Attributes getAttributes(String name,`
    `String[] ids)`
    `throws NamingException;`



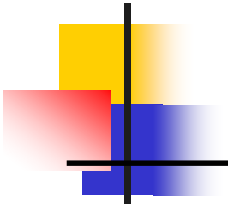
The `modifyAttributes` Operation

- Modifies the attributes of the named node.
- There are two forms of this method:
  - ```
    void modifyAttributes(String name,
                          int operation,
                          Attributes attrs)
    ```
  - ```
    void modifyAttributes(String name,
                          ModificationItem[] mods)
    ```
- The first applies a single operation (add, remove, replace) to all of the attributes passed.
- The second uses an array of `ModificationItem` instances. A `ModificationItem` is simply an object containing an `int` for the operation and an `Attribute`.



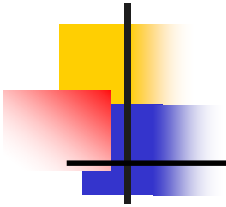
Searching

- JNDI provides two means for searching.
  - Searching based prototype set of attributes.
  - Searching using filter expressions, which also evaluate attributes.
- Searching is based upon attributes, not the nodes, themselves, *e.g.*, there is no search that will look inside a file object as a node.



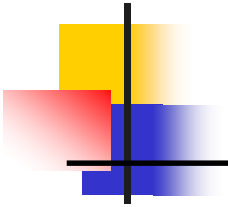
Searching by Filter Expression

- JNDI search expressions are based upon RFC 2254: LDAP Search Filters.
- Supports &, | and !
- Supports <, <=, =, =>, > and ~= (approx)
- Supports limited wildcard values ('*')
- Examples:
  - No phone number: `(!(telephoneNumber=*))`
  - Misheard name: `(cn~=Neal's Bore)`



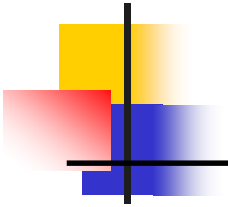
Search by Example

- `NamingEnumeration search(String name, Attributes attrs) throws NamingException;`
- Searches within a single named Context for bound objects whose attributes match the template.
- A second form of the search allows us to specify which attributes to return.
  - `Attributes search(String name, Attributes attrs, String[] ids) throws NamingException;`



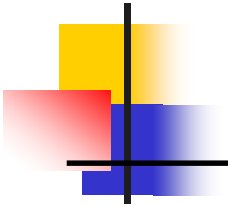
Search By Filter

- `NamingEnumeration search(String name, String filter SearchControls sc) throws NamingException;`
- The most powerful and flexible form of searching.
- A second form allows us to use attributes for which a string representation doesn't exist.
  - `NamingEnumeration search(String name, String filter, Object[] objects, SearchControls sc) throws NamingException;`
  - Example: `(jpegPhoto={0})`



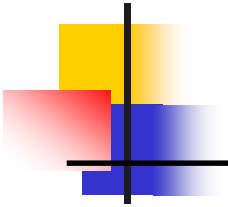
Search Controls

- `javax.naming.directory.SearchControls`
- The `SearchControls` class is used with the filter expression form of searching to control all of the search parameters, *e.g.*,
  - Scope: Object, Context, Tree
  - Which attributes to return
  - Maximum number of objects to return
- We'll make use of all of these in our demo.



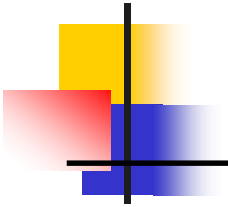
Namespace Federation

- We can bind what appears to be a context, but is actually a `Reference` to another naming context, *e.g.*, binding a DNS or File System context into an LDAP context.
- When JNDI hits that `Reference`, it will continue processing with a service provider for the referenced context.
- This is an extremely powerful capability that allows JNDI to become a core, linking many disparate naming spaces.



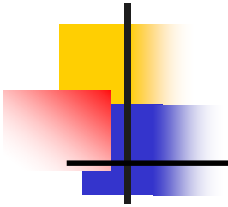
Storing Objects

- JNDI uses State and Object Factories to convert between object state and persistent state.
- Built-in support for several types of objects.
  - JNDI knows how to store and retrieve `Serializeable` and `Reference` objects.
  - When storing, JNDI will call …
    - `getReference` on `Referenceable` objects
    - `getAttributes` on `DirContext` objects



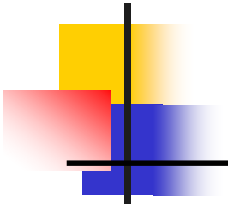
`Reference` Objects

- `javax.naming.Reference`
- A bindable object that contains the information necessary to locate and instantiate the real object.
- The JNDI documentation says that JNDI traverses references automatically, but the JDK provided service providers don't. See our `dereference` method for a workaround.
- Object Factories are used to instantiate the real object from the Reference.
- There are several specialized `Reference` sub-classes, as well.



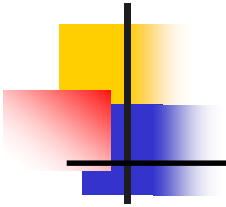
`Referenceable` Objects

- `javax.naming.Referenceable`
- When binding a `Referenceable` object, JNDI will bind a `Reference` instead.
- Implements the `getReference` method.
- See our `ReferenceableString` and `ReferenceableFileInputStream` sample code for examples.



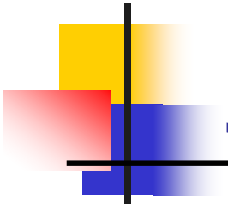
State Factories

- JNDI uses a `StateFactory` to convert from an object to persistent state.
- The `getReference` and `getAttributes` methods, and serialization can be viewed as convenient shortcuts to the otherwise greater task of providing a `StateFactory`.
- The `getStateToBind` method returns an Object suitable for binding.
- Factories intended for use when binding to a `DirContext` have a method that returns a `Result`, which often has a `null` object reference, and a set of attributes representing the persistent state.



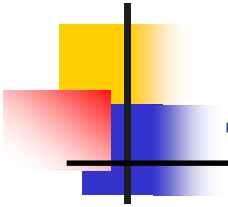
Object Factories

- JNDI uses an `ObjectFactory` to convert from persistent state to an object.
- The `getObjectInstance` method returns an `Object` based upon its persistent state.
- Examples from our sample programs:
  - `ReferenceableString`
  - `ReferenceableFileInputStream`
  - `User - uses a DirObjectFactory`



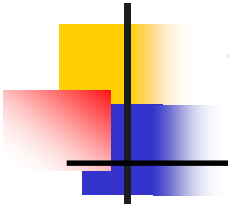
JNDI Events

- NamingEvent
  - Binding getOldBinding()
  - Binding getNewBinding
  - int getType() - *of event*
  - EventContext getEventContext()
- NamingListener
  - namingExceptionThrown(NamingExceptionEvent)
- NamespaceChangeListener
  - void objectAdded(NamingEvent evt)
  - void objectRemoved(NamingEvent evt)
  - void objectRenamed(NamingEvent evt)
- ObjectChangeListenter
  - void objectChanged(NamingEvent evt)



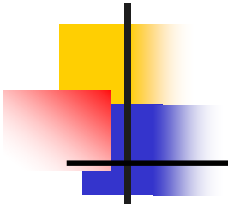
JNDI LDAP Demo

- Exercise all of the major Context and DirContext operations, including:
  - Attributes
  - Searching
  - Storing Objects
    - Referenceable
    - Serialization
      - ✓ Includes an example using the `javaCodebase` attribute
    - DirContext sub-class
  - Federation
  - Events



Wrap-up

- Are there any questions?
- Please remember to turn in your speaker evaluation forms.
- Thank you for coming. I hope that you've enjoyed the session.



Links and Related Sessions

Located at the end of Part II