Java2 Enterprise Edition

Java Naming and Directory Interface (JNDI)

Agenda

- Introduction
- Context and the JNDI Tree
- Working with JNDI
- Summary

Introduction

The Java Naming and Directory Interface (JNDI) provides an API for unified access to naming and directory services from Java classes

- –a naming and directory service provides resources stored in hierarchical order (in various“ contexts"), each identified by a logical name
- –common services are: LDAP, DNS, Files ystem, NIS, ...

Naming Service

- A naming service provides a method for mapping identifiers to entities or objects.
- There are many terms you need to know:
    - **Binding (*association of an atomic name with an object*)**
    - **Namespace (*names in a naming system in which the names remain unique*. *A file directory is a sample namespace*.)**

Directory Services

- **Directory services:**
  - provide structure to a set of directory objects:
  - are usually hierarchical in nature
  - have searchable attributes associated with each object
- **Directory objects represent objects in the computing environment**:
  - printers
  - servers
  - person

JNDI API and JNDI SPI

- **The JNDI architecture consists of the following parts:**
  - the API (Application Programming Interface) describes how an application developer can use JNDI
  - the SPI (Service Provider Interface) describes how to "JNDI-enable" any naming and directory service
    - e.g. the vendor of a JNDI Naming Manager must implement some factory interfaces defined in the SPI to provide implementing classes for interfaces of the API

JNDI Packages

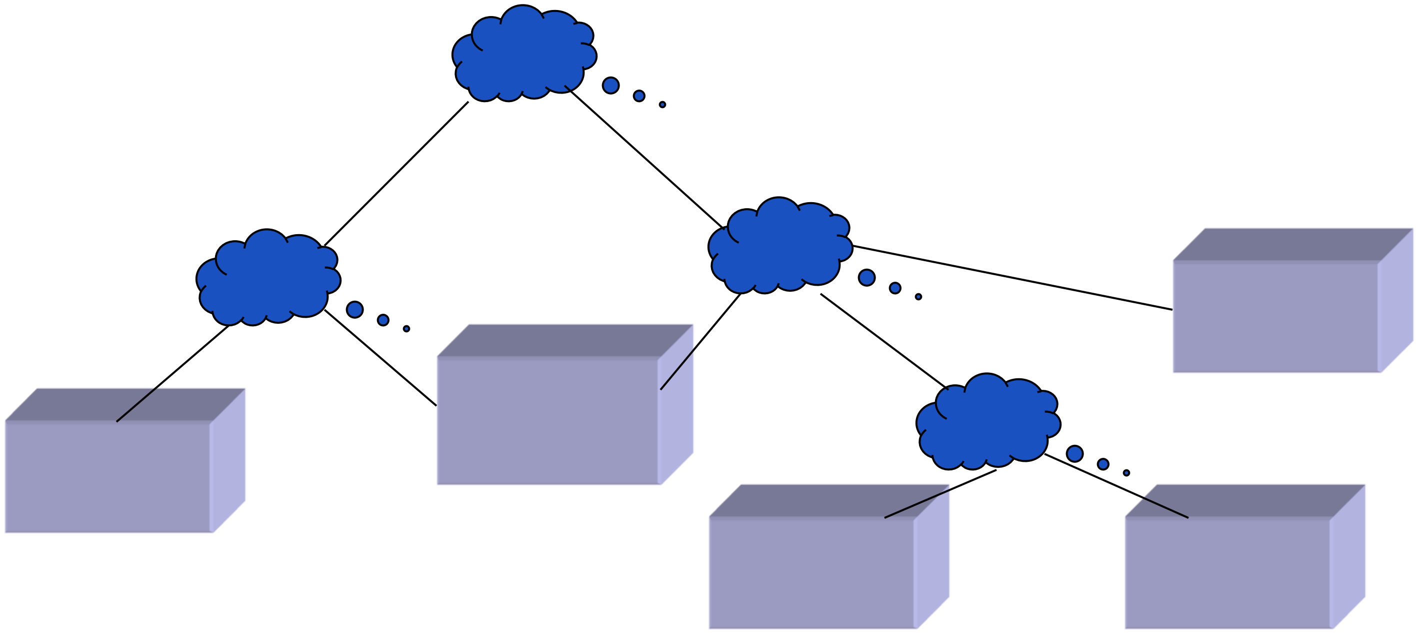
- **The JNDI packages are part of the J2SE SDK**
  - **javax.naming:** classes and interfaces for accessing naming services
  - **javax.naming.directory:** extension of javax.naming to provide access to directories
  - **javax.naming.events:** support for event notification in naming and directory services
  - **javax.naming.ldap:** components to support LDAP v3
  - **javax.naming.spi:** the Service Provider Interface

    mainly contains factory interfaces, e.g. InitialContextFactory etc.

Context and the JNDI Tree

The JNDI Tree

- **(…is not a tree, but an arbitrary directed graph)**
- **Each internal node is a C*ontext*(*javax.naming.Context*)**
- **Each leaf is an arbitrary resource(java.lang.Object)**
- **Instead of a resource, the tree may contain a place holder or stand-in for a resource (*javax.naming.Reference*)**
- **Or a link to some other node in the JNDI tree (*LinkRef*)**



The Gate to JNDI: javax.naming.Context

- ***javax.naming.Context*** **is the central interface of JNDI**
- ***Context*** **provides...**
  - – methods for binding names to resources and hence making them available through JNDI (**bind(…), rebind(…)**)
  - –methods for obtaining bound resources **lookup(...)**)
  - –methods for creating further contexts(**createSubcontext(...)**)
  - –methods for browsing the tree(**listBindings(...)**)
  - –various constants for context management
  - –and more
- **A special context is the class** ***javax.naming.InitialContext***
  - –it is the starting point for JNDI operations
  - –it is obtained from a server-specific implementation of *javax.naming.spi.InitialContextFactory*

Initial Context

- **An *InitialContext* is the starting point for all JNDI operations**
  - –it need not be the root context
  - –in fact, there is no universal root
- **All sub contexts and resources are named relative to the Initial Context:**

  –A

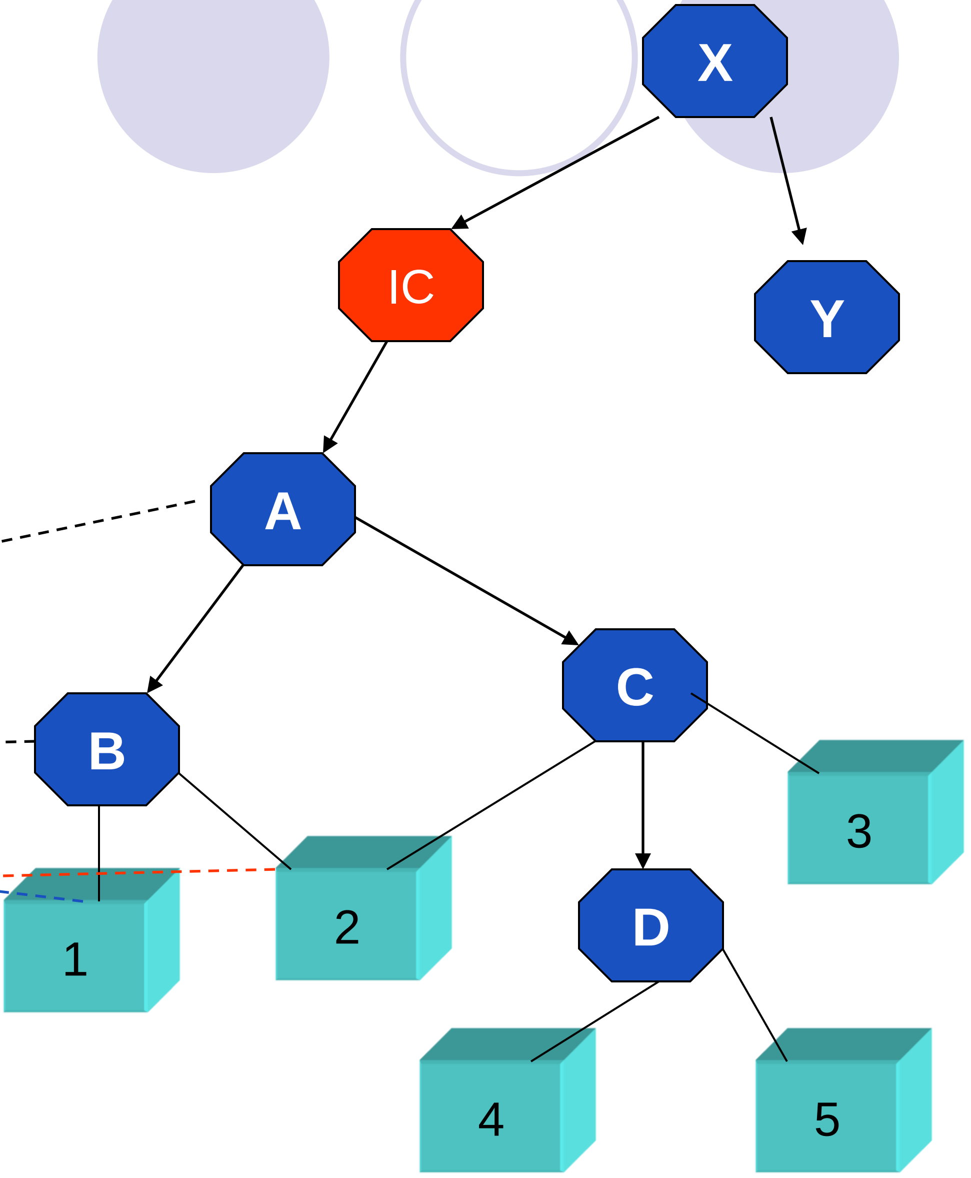
  –A/B

  –A/B/1

  –A/B/2 orA/C/2

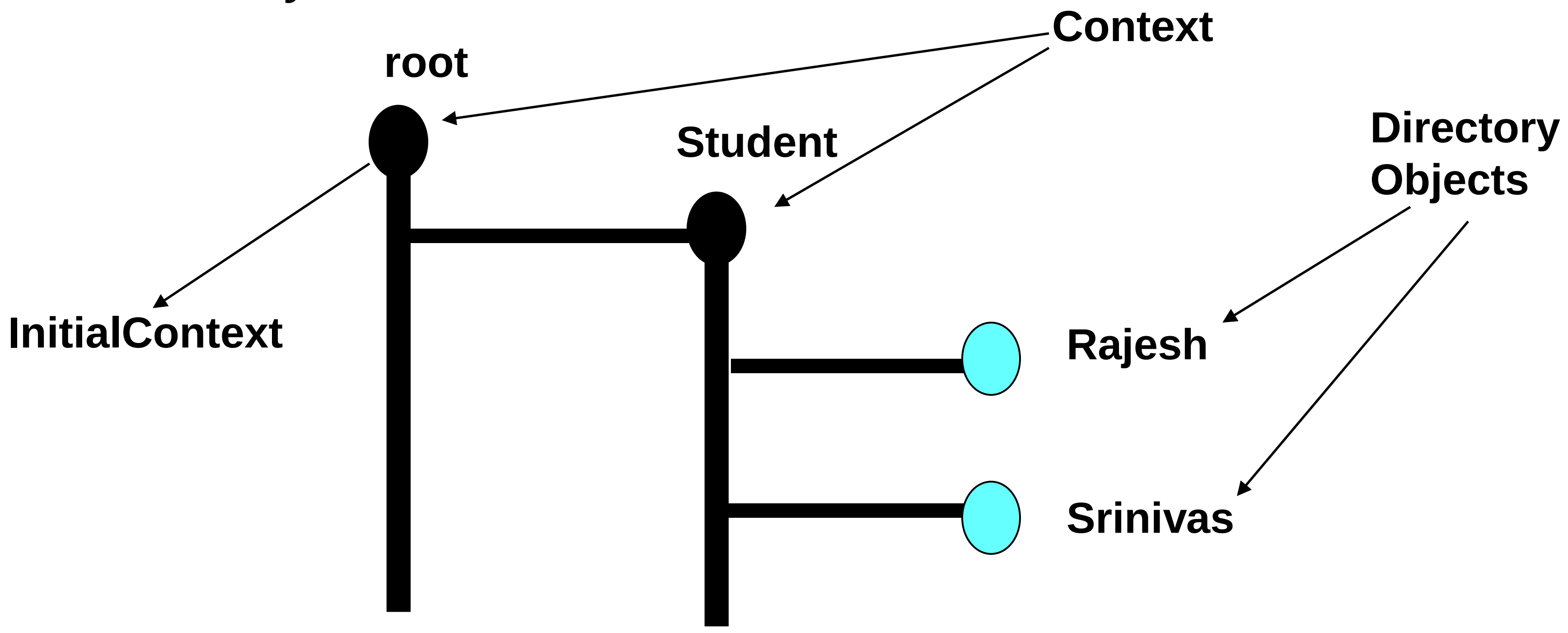
  –no access to X possible from this InitialContext

  –syntax maybe namespace-specific



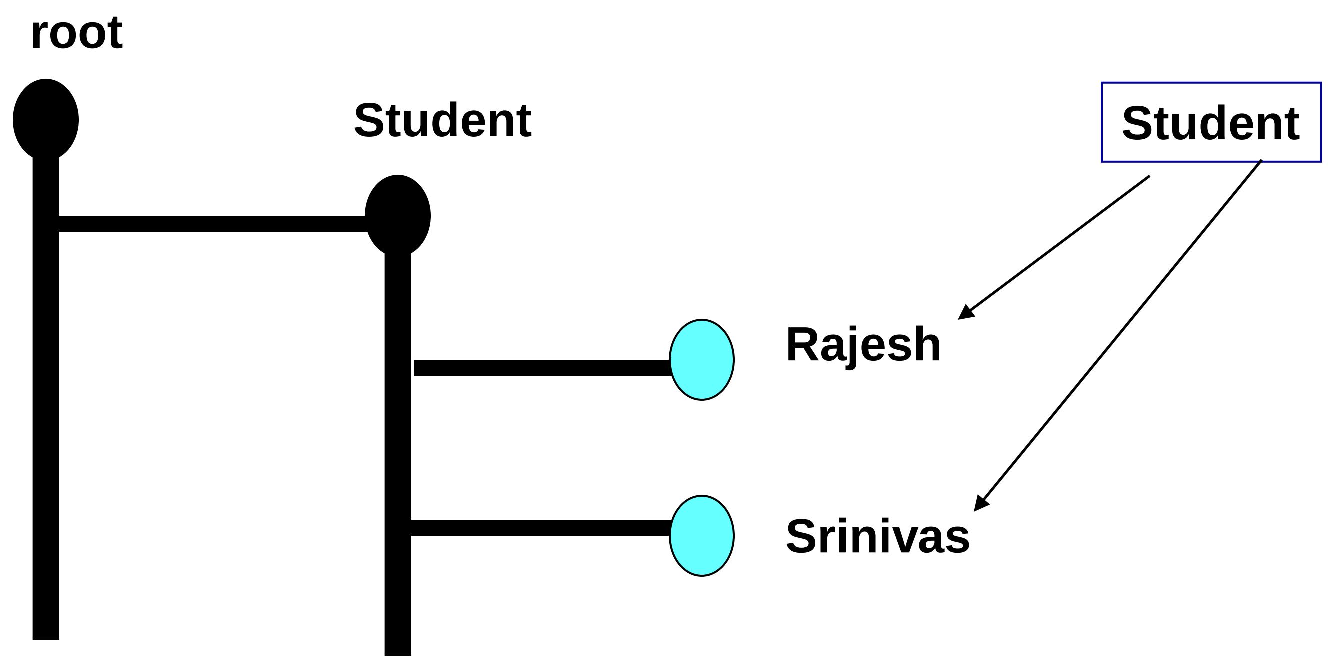
Contexts and InitialContexts

- Every node in a directory structure is called a context.
- The initial context is your starting point in traversing a directory structure.



Binding Objects in JNDI

- Objects bound in a naming service must be serializable objects
- Objects are copied into the naming service..



LookingUp Objects in JNDI

//Obtain the initial context

```
Context initialContext = new InitialContext();
```

// Lookup an existing Student object

```
Student rajesh = (Student)initialContext.lookup( student/Rajesh );
Student srinivas = (Student)initialContext.lookup( student/Srinivas );
```

Obtaining an InitialContext

```
Properties prop=new Properties();
prop.put(Context.PROVIDER_URL,"t3://localhost:7001");
prop.put(Context.INITIAL_CONTEXT_FACTORY,
                              "weblogic.jndi.WLInitialContextFactory");
/* may be put user name and password, too. */

Context ctx= new InitialContext(prop);
```

Exemplary values for Weblogic Server

More properties available, e.g. SECURITY_PRINCIPAL, SECURITY_CREDENTIALS,etc.

- JNDI server may run on the local machine or a remote one
- JNDI server may or may not require authentication
- Parameters are supplied by program or configuration file

Obtaining Objects from JNDI

- **An instance of a bound resource can be obtained by:**

  ```
  MyClass instance= (MyClass) ctx.lookup("URI");
  ```

  –the URI is the identifier to which the resource is bound, relative to the ***InitialContext***

- **If using IIOP (which is standard in J2EE applications) one has to narrow the obtained object:**

  ```
  Object object= ctx.lookup("URI");
  MyClass instance= (MyClass)
  javax.rmi.PortableRemoteObject.narrow(object, MyClass.class);
  ```

Binding Objects to JNDI

- **A resource can be bound to the JNDI tree by one of the following mechanisms:**
  - –programmatically with **Context.bind(...)**
  - –with utilities or interactive applications provided by theserver vendor
  - –automatically by the ApplicationServer, e.g. when deploying Enterprise Beans

Example

- **J2EE introduces the concept of DataSources which provide connection pools to databases**
  - –Application Servers must allow the creation and JNDI-binding of DataSources
  - –if bound, one can use a DataSource:

```
Contextctx= new InitialContext();
javax.sql.DataSource dataSource=
(DataSource) ctx.lookup(“jdbc/mydataSource");
java.sql.Connectioncon = dataSource.getConnection();
// do someJDBC operations
```

- **Other resources typically bound to JNDI contexts:**
  - –UserTransactions, Enterprise Beans, other remote objects or services, ...

Summary

- **JNDI provides a powerful mechanism for using various naming and directory services**
- **A JNDI tree consists of nodes and leafs:**
  - –a **node** is a **Context**
  - –a **leaf** can be any **object**
- **Arbitrary resources can be bound to the JNDI tree**
- **Instances of bound resources can be retrieved by Context.lookup**
- **for more information, please refer to the JNDI specification: http://java.sun.com/products/jndi/docs.html**