CMSC838p
Topic – Enterprise Java Beans –
The Entity Beans
By
Niketu Parekh, Prashant Lamba
Dr. William Pugh
EJB Technology

EJB Technology – based on Java RMI-IIOP and JNDI
Java RMI - IIOP Technology

- Java Remote Method Invocation over the Internet Inter-ORB Protocol
- RMI – IIOP allows
  - Writing distributed objects in Java
  - Enabling objects to communicate in memory, across JVMs, and across physical devices
- RMI – IIOP is a special version of RMI which is compliant with CORBA
RPC ↔ RMI ↔ RMI-IIOP

- RPC enables traditional procedures to reside on multiple machines, yet still remain in communication.
- A simple way to perform cross-process OR cross-machine networking.
- RMI (a step further) allows distributed object communication
- RMI-IIOP allows invocation of methods on remote objects
RMI Vs RMI-IIOP

- RMI has some features such as Distributed Garbage Collection, Object Activation and Downloadable class files which are not available in RMI-IIOP.
RMI-IIOP

- RMI-IIOP exploits the following features of Object Oriented programming
  - Separation of interface of code from its implementation
  - Can change object’s proprietary logic w/o affecting client
  - Object serialization and parameter passing (pass by value & pass by reference)
Java Naming and Directory Interface (JNDI)

- A standard way of looking things up over the network
- JNDI is a system for Java based clients to interact with naming and directory systems.
- A bridge over naming and directory services that provide common interface to disparate directories.
- Users need to use the same API to access LDAP directory, NIS directory or Novell’s NDS directory.
JNDI - Benefits

- Single API to access all sorts of directory services information
- JNDI insulates the application from protocol and implementation details
- Link different types of directories
JNDI – J2EE Specific use

- Use JNDI to connect to resource factories such as JDBC drivers and JMS drivers
- Use JNDI for beans to lookup for other beans over the network.
- Use JNDI to acquire reference to Java Transactions API (JTA) User – Transaction interface
JNDI – Architecture

- **JNDI Client API** – Uniform API for all sorts of directories allowing Java code to perform directory operations
- **JNDI Service Provider Interface** – An interface to which naming and directory service vendors can plug in their proprietary services.
Client – EJB Component system interaction
Enterprise Bean Class

- Model business data
- Java Objects that cache Database Information
EJB Object

- Model business data
- Java Objects that cache Database Information
EJB Object

- Model business data
- Java Objects that cache Database Information
What is Entity Bean?

• Multi-tier Object Oriented Deployment Overview
• Two kinds of components deployed
• Application Logic Components
  • Method providers that perform common tasks.
  • Session beans model application logic components
• Persistent Data Components
  • Objects that know how to render themselves into persistent storage
  • They use some persistence mechanism such as serialization, O / R mapping to a DB or an Object DB.
• Entity beans model these persistent data components.
What is Entity Bean? (cont’d)

- Entity beans are enterprise beans that
  - An in-memory Java representation of persistent data.
  - Physical, storable parts of an enterprise
  - Store data as fields and have methods associated with them.
  - Smart enough to know how to read itself from a storage and populates its fields with the stored data
  - An object that can be modified in-memory to change the values of data
  - Persistable, so that it can be stored back into storage again, thus updating the database data.
What Makes An Entity Bean?

• Consists of
• Remote and / or local interface
• Home and / or local home interface
• Enterprise bean class
• Deployment descriptor
Noteworthy Differences

- Entity bean class maps to an entity definition in a database schema
- Entity bean class can expose simple methods to manipulate or access that data
- Primary Key Class makes every entity bean different.
- A primary key contains any number of attributes to uniquely identify an entity bean instance.
Features of Entity Beans

- Entity beans survive critical failures such as application server crashing, database crashing.
- Entity bean instances are a view into Database
- Several entity bean instances may represent the same underlying data
- Entity bean instances can be pooled
Persisting Entity Beans

- Since entity beans map to storage, it requires mechanism for database access.
- Two ways to persist entity bean
  1. Bean Managed Persistent (BMP) Entity Bean
  2. Container Managed Persistent (CMP) Entity Bean
Bean Managed Persistent (BMP) Entity Beans

- Bean provider writes the database access code, including looking up a DataSource, getting a Connection, and sending JDBC statements to the database.
- Container invokes a container callback on the bean when the bean needs to do something with the database and then the code does the actual JDBC work.
BMP Entity Beans Life Cycle

The lifecycle of a bean-managed persistent entity bean. Each method call shown is an invocation from the container to the bean instance.
Container Managed Persistent (CMP) Entity Beans

- Wimpy in EJB1.1, greatly enhanced in EJB2.0
- Bean provider designs the entity bean class, choosing which of the bean’s fields are part of bean’s persistent state. Persistent fields map to columns in one or more database tables.
- Container keeps track of changes to the bean’s state and updates the real entity as needed. The container makes the decision on how to keep the bean and the real entity synchronized based on the state of transactions.
- Bean provider writes EJB-QL to tell the container how to do selects.
- Using information in the deployment descriptor, the container writes the actual implementation of the CMP bean, including implementing the finder and select methods and all of the database access code.
CMP Entity Beans Life Cycle
Passivation and Activation
Session Vs. Entity Beans

- **Stateless** session beans go back to the pool w/o Passivation
- **Stateful** session beans are passivated when the container puts them to sleep to conserve resources between client method invocations.
- Unlike session beans, passivated entity beans are still live objects on the heap.
ejbPassivate(): Session Vs. Entity Beans

Entity Bean

- called when the bean has finished a business method and is about to go back to the pool. In the passivated state, the bean has no identity It’s not representing any entity from the database
- For entity beans, ejbPassivate() is used to release resources that you don’t want to waste while the bean is sitting in the pool, not running a business method
- The bean is not serialized or saved, so there is no requirement that you null out references to non-serializable objects.
ejbPassivate():
Session Vs. Entity Beans  (cont’d)

Stateful Session Bean
- Called when the container decides to conserve resources between business method invocations from the client
- Used to release resources and to prepare the bean’s state for what might be serialization (null our non-serializable references etc)
- Bean is put to sleep and is no longer taking up space on the heap

Stateless Session Bean
- Not Applicable
Writing A BMP Bean
Writing A CMP Bean