EJB Container

Provides a Runtime Environment for an Enterprise Bean

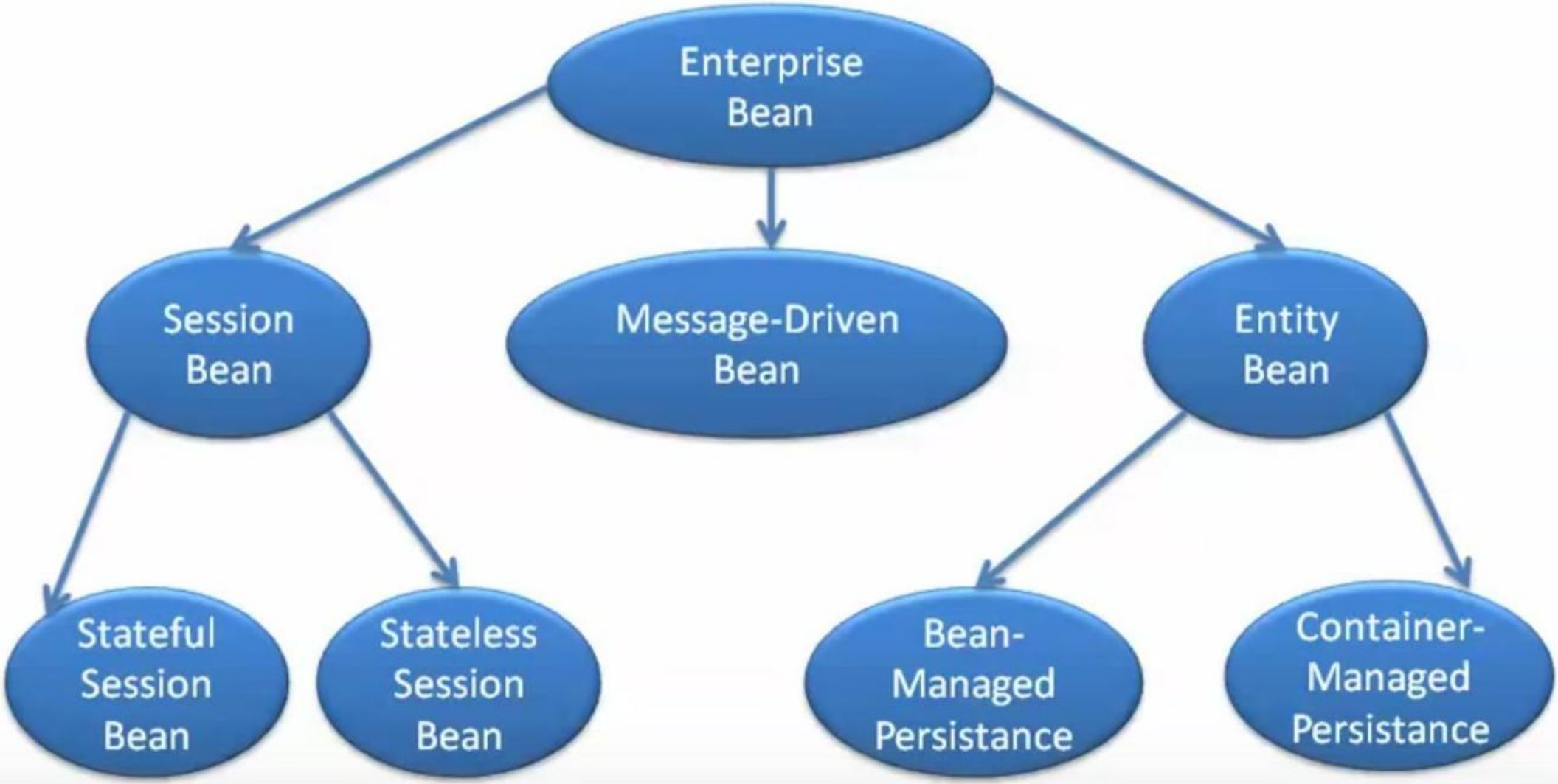
- Hosts the Enterprise JavaBeans
- Provides services to Enterprise JavaBeans:
  - Naming
  - Life Cycle Management
  - Persistance (state management)
  - Transaction Management
  - Security

Enterprise JavaBean

A specialized Java class where the real business logic lives

- Distributed over a network
- Transactional
- Secure

Enterprise JavaBean (2)



EJB Client

EJB Client is a local program which can call and operate Remote Bean

- Clients locates an Enterprise JavaBean through Java Naming and Directory Interface (JNDI)
- RMI is the standard method for accessing a bean over network

Components of EJB Architecture

EJB Container

Provides a Runtime Environment for an Enterprise Bean

- Hosts the Enterprise JavaBeans
- Provides services to Enterprise JavaBeans:
  - Naming
  - Life Cycle Management
  - Persistance (state management)
  - Transaction Management
  - Security

EJB Server

Provides a runtime environment

- The EJB Server provides system services and manages resources:
  - Process and thread management
  - System resources management
  - Database connection pooling and caching

Lifecycle Callbacks

- As an EJB developer, you can receive notification as a bean transitions through each phases
- Notification is done through use of a callback method
- Callback methods are annotated with the appropriate lifecycle callback annotation

Lifecycle Example

```
1   package com.developintelligence.tutorials.ejb3;
2
3   import ...
7
8   /**...*/
11  @LocalBean
12  @Singleton
13  public class SingletonCalculatorBean {
14
15      private long initialization;
16
17      public SingletonCalculatorBean() {}
18
19      @PostConstruct
20      private void startCounter() {
21        initialization = System.currentTimeMillis();
22      }
23
24      @PreDestroy
25      private void recordLifespan() {
26        long destruction = System.currentTimeMillis();
27        System.out.printf("SingletonCalculatorBean lived: %s ms\n", (destruction - initialization));
28      }
29
30      public int add(int a, int b) {...}
33      public int subtract(int a, int b) {...}
36      public int multiply(int a, int b) {...}
39      public double divide(int a, int b) {...}
```

5 Key Lifecycle Phases

- @PostConstruct - after object is created
- @PreDestroy – before object is removed from container
- @PrePassivate – before object's states are preserved
- @PostActivate – after object's states are resurrected
- @Remove – after client signals object removal

: : NOTE : :

NOT ALL SESSION BEANS GO THROUGH ALL PHASES

Message Driven Beans [MDB]

- Reusable workflow logic components
  - Rely on Java Messaging System
  - Support transactions
- Similar to Stateless Session Beans
  - MDBs have no client state data
  - No distinction across client or bean
- Different from Stateless Session Beans
  - No direct client access
  - Invoked through message notification
  - Asynchronous interactions

Access Mode Annotations

Bean Type	Annotation

Session Bean	`@Local`
	`@Remote`
	`@LocalBean`*
	`@WebService`

Message Driven	`@WebService`
JPA Entities	N/A

EJB Management Modes

- Two management modes:
  - Container-managed
  - Bean-managed
- Management-modes specified by Annotations

EJB Classifications

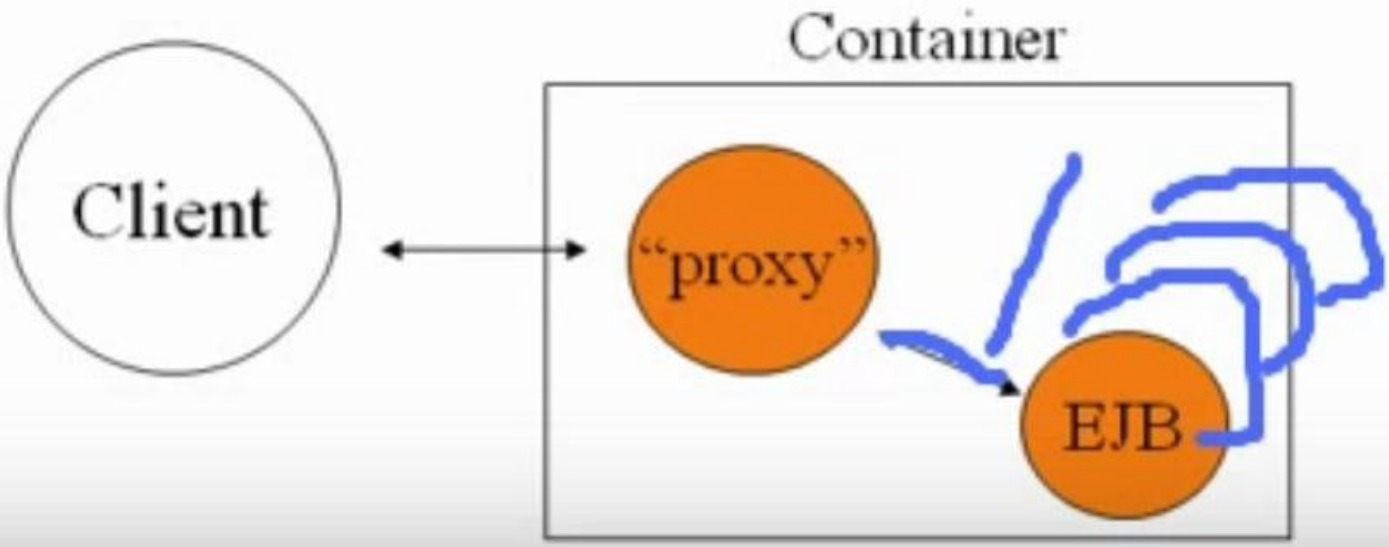
- In container-managed, container manages:
  - Transactions
  - Roles
  - Security
  - Persistence and Entities (PersistenceContext)
- In bean-managed, bean manages:
  - Transactions
  - Persistence

Management Mode Annotations

Bean Type	Annotation
Transactions	`@TransactionManagement(CONTAINER)` `@TransactionManagement(BEAN)`
Security	`@RunAs()` `@RolesAllowed`

EJB Access

- EJB Instance access is controlled by container
- There is no direct client access



EJB Instantiation

- Instantiation of EJB is handled by container
- Container determines when and “how many”
- Uses standard instantiation mechanism (public no argument constructor)
  - Creates an EJBObect (proxy)
  - Creates an EJB instance (actual bean instance)

EJB Lifecycle

- Lifecycle of an instance is managed by container

- Lifecycle phases:
  - Does Not exist – no bean instance in memory
  - Post-construct / Not-ready – bean instance exists but isn't ready for client interaction
  - Ready – exists, initialized, and ready for interactions
  - Pre-removal – bean instance is about to be removed

- Lifecycle transition notifications are handled through call-backs

Choosing a Session Bean

- Use a Session bean if:
  - Only one client has access to bean at any given time
  - State of bean is not persistent
  - Bean represents a web service
- Use a Stateful Session bean if:
  - Bean state represents client interaction
  - Bean needs to hold client data across interactions
  - Bean acts as a client mediator to other beans
  - Need thread-safe interactions

Session Bean Cardinality

	@Stateless	@Stateful	@Singleton
Remote Client Access	✓	✓	✓
Local Client Access	✓	✓	✓
Concurrent client access	✗	✗	✓
Unique per client	✗	✓	✗
Client-bean instances	Pooled	1:1	Many:1

Developing Business Interface

- Similar to standard Java interface
  - public interface SomeInterface
  - public int getSomeProperty();
- Annotated with client-access mode
  - @Local – local (in context of application only)
  - @Remote – remote (inside and outside of context)

Remote Interface

```
1   package com.developintelligence.tutorials.ejb3;
2
3   import javax.ejb.Remote;
4
5   /...*/
9   @Remote
10  public interface RemoteCalculator {
11      public int add(int a, int b);
12      public int subtract(int a, int b);
13      public int multiply(int a, int b);
14      public double divide(int a, int b);
15
16  }
```

Session Bean Creation

- Session bean instances are created and managed by container
  - No way for client to directly instantiate a bean
  - Physical EJB object creation is “hidden” from client

- Session beans are created as a result of some other action
  - Stateless – first client lookup
  - Stateful – on every client lookup
  - Singleton – on application load

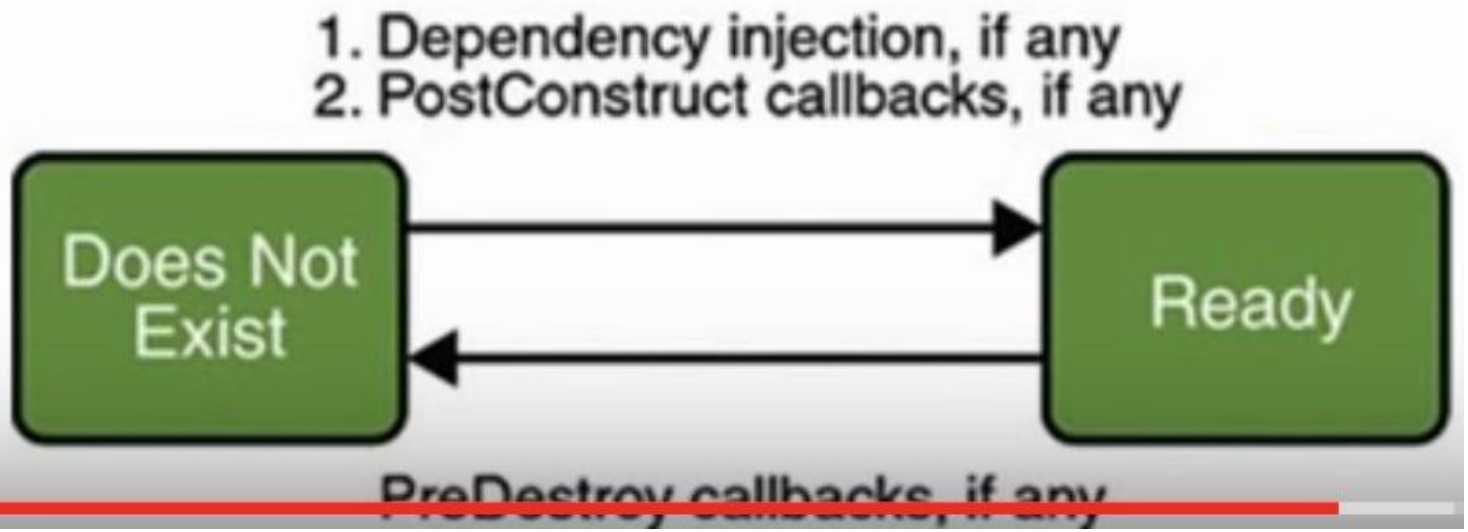
Session Bean Destruction

- Session bean instance is destroyed by container
- Session bean "destruction" is result of some other action
  - Stateless – pool clean up or inactivity
  - Stateful – client inactivity or remove
  - Singleton – application shutdown
- Certain exceptions can cause beans to be destroyed (on initialization)

Singleton Session Beans

Singleton Session Bean Lifecycle

- Create instance
- @PostConstruct – perform initialization
- Process business methods
- @PreDestroy – undo initialization



Stateful Session Beans

Stateful Session Bean Lifecycle

- Create instance
- @PostConstruct – perform initialization
- Process business methods
  - @PrePassivate – prepare for serialization
  - @PostActivate – recover from serialization
- @Remove – remove ejb from container
- @PreDestroy – undo initialization

What is EJB?

- An Enterprise Java Bean is:
  - A reusable component
  - A Java object
  - An encapsulation of enterprise business logic and data
  - Executed in a Containers
- EJB Containers provide:
  - Pooling
  - Transaction Management
  - Security
  - Naming and Directory
  - Configuration
  - Etc.

Types of Enterprise Java Beans

- Three main categories
  - Business logic – `SessionBean`
  - Workflow logic – `MessageDrivenBean`
  - Persistence logic - JPA Entity

- Classifications specified with through Annotations

Session Beans

- Reusable business logic components
- Can be used to manage state across client interactions
- Three types
  - Stateless (SLSB)
  - Stateful (SFSB)
  - Singleton (SSB) [3.1 or higher]

Message Driven Beans [MDB]

- Reusable workflow logic components
  - Rely on Java Messaging System
  - Support transactions
- Similar to Stateless Session Beans
  - MDBs have no client-oriented state
  - No distinction across client or bean
- Different from Stateless Session Beans
  - No direct client access
  - Asynchronous interactions

Types of EJB Access Modes

- Three client access modes:
  - Local client-access
  - Remote client-access
  - Web service client-access
- Client-access modes specified by Annotations

EJB Classifications [cont.]

- Local beans are accessible
  - Only by other EJBs in the *same* context
  - All categories of beans can be local
- Remote beans are accessible
  - By EJBs in the same context and outside the context
  - By other "objects" outside of the container
  - Only Session Beans can be remote
- Web Service end-points
  - Translate SB and MDBs into web-services
  - Container manages WSDL/SOAP mappings

Developing a Session Bean

Class requirements similar to JavaBeans:

- Must be a top level class
- Must be defined as public
- Can not be final or abstract
- Must have a public no-argument constructor that takes no parameters.
- Must not define the finalize method
- Must implement the methods of the business interface

Local Stateless Bean

```
1   package com.developintelligence.tutorials.ejb3;
2
3   import javax.ejb.Stateless;
4
5   /*...*/
8   @Stateless
9   public class LocalCalculatorBean implements LocalCalculator {
10
11      public LocalCalculatorBean() {...}
13
14      public int add(int a, int b) {...}
17
18      public int subtract(int a, int b) {...}
21
22      public int multiply(int a, int b) {...}
25
26      public double divide(int a, int b) {...}
29  }
```

Remote Stateful Bean Implementa

```
1   package com.developintelligence.tutorials.ejb3;
2
3   import javax.ejb.Stateful;
4
5   /**...*/
8   @Stateful
9   public class RemoteCalculatorBean implements RemoteCalculator {
10
11      public RemoteCalculatorBean() {...}
13
14      public int add(int a, int b) {...}
17
18      public int subtract(int a, int b) {...}
21
22      public int multiply(int a, int b) {...}
25
26      public double divide(int a, int b) {...}
29  }
```

LocalBean Singleton Bean Implementati

```
1   package com.developintelligence.tutorials.ejb3;
2
3   import javax.ejb.LocalBean;
4   import javax.ejb.Singleton;
5
6   /**...*/
9   @LocalBean
10  @Singleton
11  public class SingletonCalculatorBean implements RemoteCalculator {
12
13      public SingletonCalculatorBean() {...}
15
16      public int add(int a, int b) {...}
19
20      public int subtract(int a, int b) {...}
23
24      public int multiply(int a, int b) {...}
27
28      public double divide(int a, int b) {...}
31  }
```

Summary

- There are three types of session beans:
  - Stateless - @Stateless
  - Stateful - @Stateful
  - Singleton - @Singleton
- Every session bean has:
  - Business interface
  - Implementation class
  - Deployment descriptor information

:: QUESTION ::

What is the business interface of a LocalBean?

Steps for Interacting with an EJB

3 Step Plan:

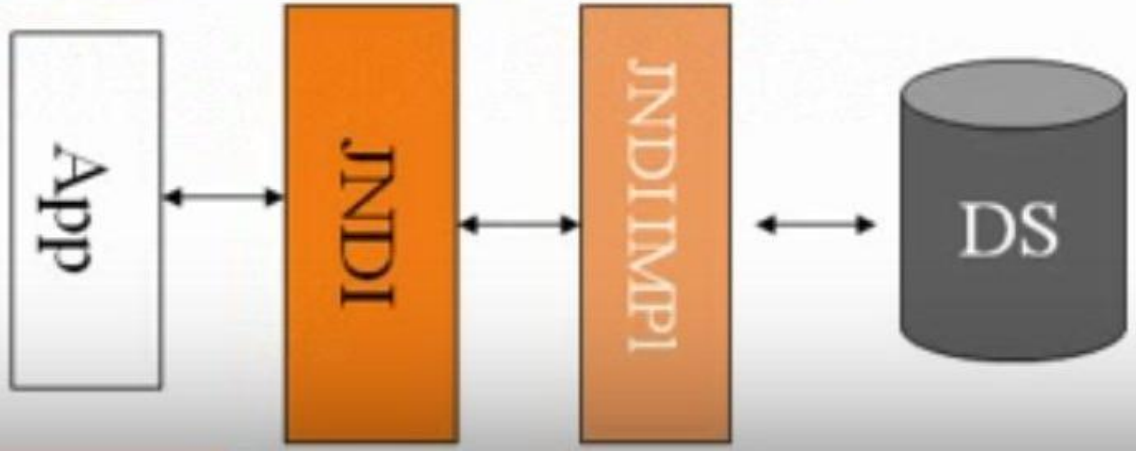
1. Declare a reference for the EJB's interface

2. Get the EJB "instance"
   - Using JNDI for remote clients or legacy EJBs
   - Using DI for local clients or application clients

3. Invoke methods on the interface

JNDI

- Java Naming and Directory Interface is part of Java SE
  - Standard API to interact with naming and directory services
  - Provide the ability to look things up in a "registry"
  - Used in enterprise solutions to locate resources such as EJBs, JMS queues or Web Services
- JNDI resources are organized in a tree structure
  - Analogous to the folder structure of a computer's file system
  - Supports events, lookups, and complex searching against structure



Key JNDI Concepts

Finding an EJB with JNDI uses four key concepts

- Context
- Initial Context
- Path & Name
- Search

JNDI Contexts

JNDI supports multiple contexts . . .
. . . each containing different resources

- ***Local context*** - an application can obtain access to *its* resources—EJBs, DataSources, etc.
- ***Remote context*** - an application can also obtain access to a *remote* application server's resources – remote EJBs, etc.

Finding EJB References

- Traditionally, EJB containers defined their own "naming" scheme
  - Varied by app server vendor
  - Path and entry name could be overridden in xml deployment descriptor
- In EJB3.1, there are global JNDI Naming conventions

```
qualified class name
java:global[/<app-name>]/<module-name>/<bean-name>
java:app[/<module-name>]/<bean-name>
java:module/<bean-name>
```

Simple JNDI Lookup

```
1   package com.developintelligence.tutorials.ejb3.clients;
2
3   import ...
8
9   /**...*/
12  public class Main {
13
14    static RemoteCalculator calculator;
15
16    public static void main(String[] args) {
17      Context c;
18      String jndiPath = "com.developintelligence.tutorials.ejb3.RemoteCalculator";
19      try {
20        c = new InitialContext();
21        calculator = (RemoteCalculator) c.lookup(jndiPath);
22        int sum = calculator.add(5,5);
23        System.out.println("The sum is: " + sum);
24      } catch (NamingException e) {
25        e.printStackTrace();
26      }
27
28    }
29
30  }
```

Dependency Injection

- Java EE 5 introduced support for dependency injection (DI)
  - Container can automatically “inject” references
  - Used commonly in context of JNDI
  - Based on annotations
- Dependency injection
  - Simplifies programming
  - Makes access to JNDI largely transparent

Summary

- EJB's are located using JNDI or DI
- JNDI relies on Context, InitialContext, and lookups
- DI relies on @EJB annotation