# Accessing Resources with JNDI and Dependency Injection

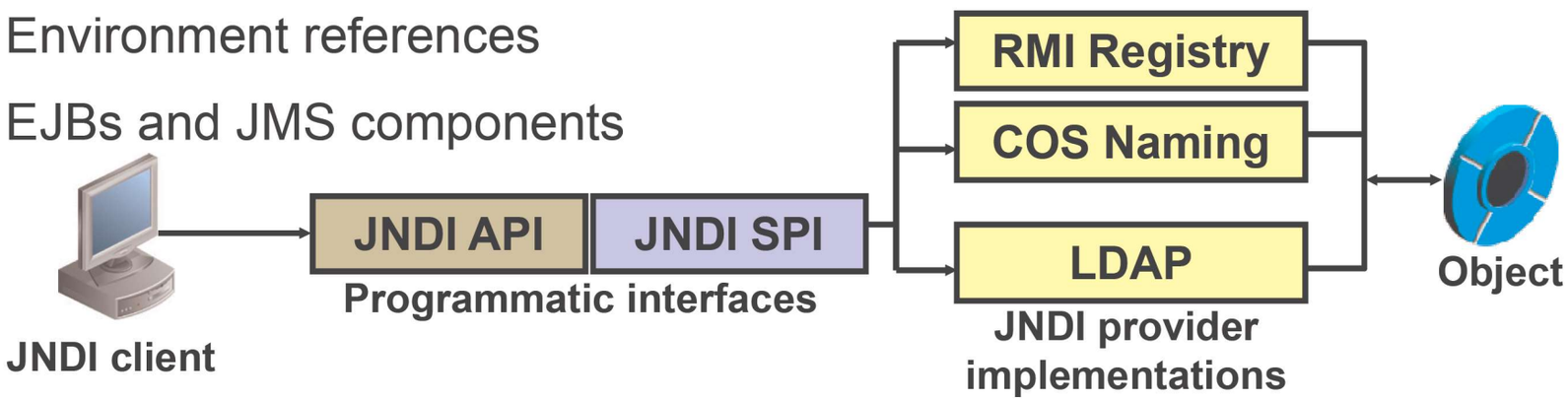
# Objectives

After completing this lesson, you should be able to do the following:

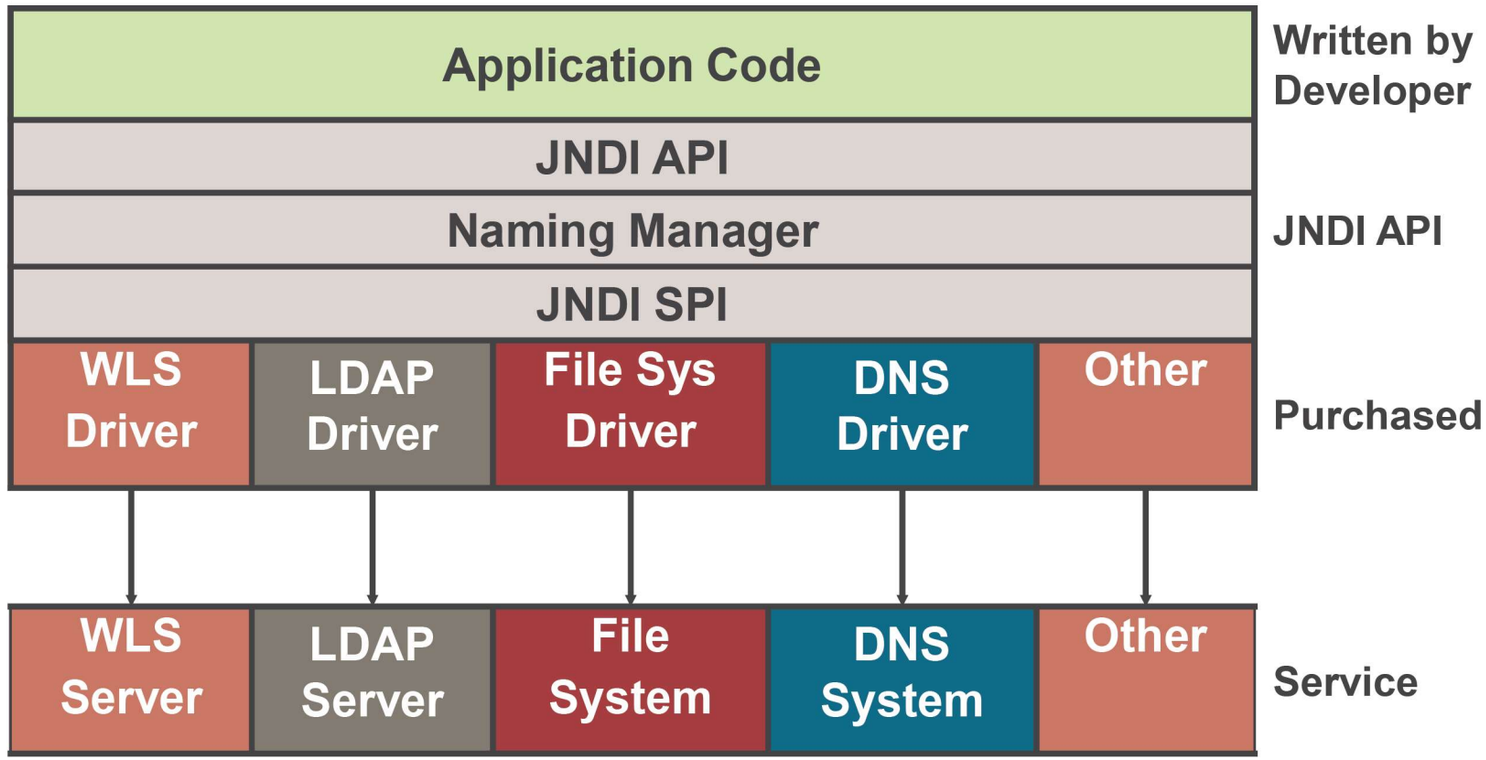
- Describe the Java Naming and Directory Interface (JNDI)
- Locate or look up resources and EJBs by using:
  - JNDI APIs
  - Dependency injection (DI)

The JNDI is:

- A standard set of interfaces that provide:
  - Naming services
  - Directory services
- A service provided by Java EE containers to locate Java EE resources or objects such as:
  - Data sources
  - Environment references
  - EJBs and JMS components

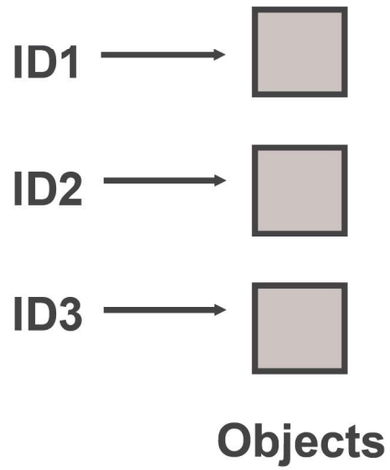


# JNDI Structure



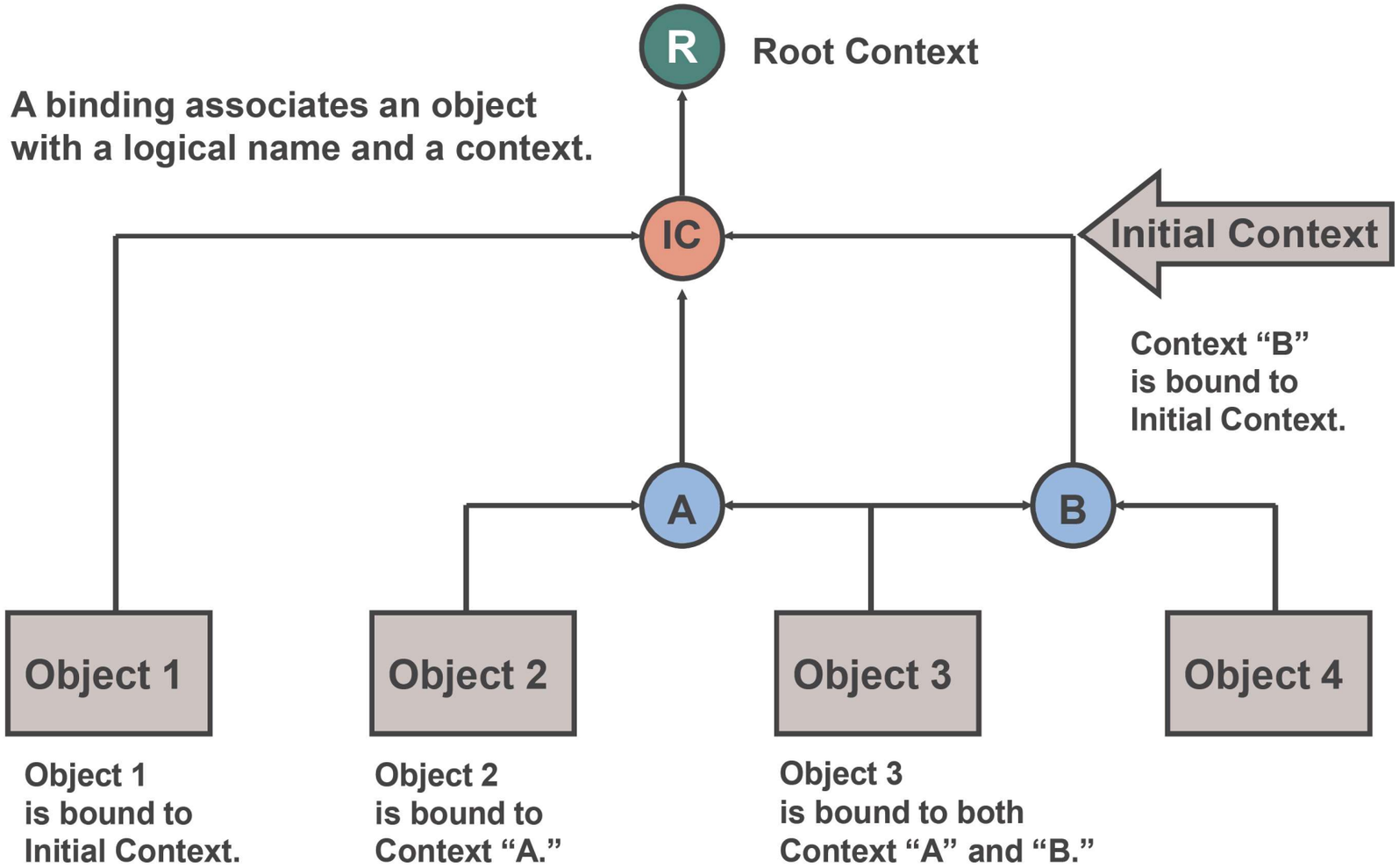
# Naming Service

- A naming service provides a method for mapping identifiers to entities or objects:




| Term | Definition | Example |
|---|---|---|
| Binding | Association of an atomic name and an object | `www.oracle.com` *`is bound to`* `209.10.217.38` |
| Namespace | A set of unique names in a naming system | `www.oracle.com/products` |

# JNDI Tree


R
Root Context
A binding associates an object with a logical name and a context.
IC
Initial Context
Context "B" is bound to Initial Context.
A
B
Object 1
Object 2
Object 3
Object 4
Object 1 is bound to Initial Context.
Object 2 is bound to Context "A."
Object 3 is bound to both Context "A" and "B."

# Contexts and Subcontexts

- Subcontexts are referenced through dot delimiters (.).
- Subcontexts must be created before objects are placed into them.
- Typically, when objects are bound to the JNDI tree, subcontexts are automatically created based on the JNDI name.
- For example, if the following context exists:

```
com.bea.examples
```

you cannot bind:

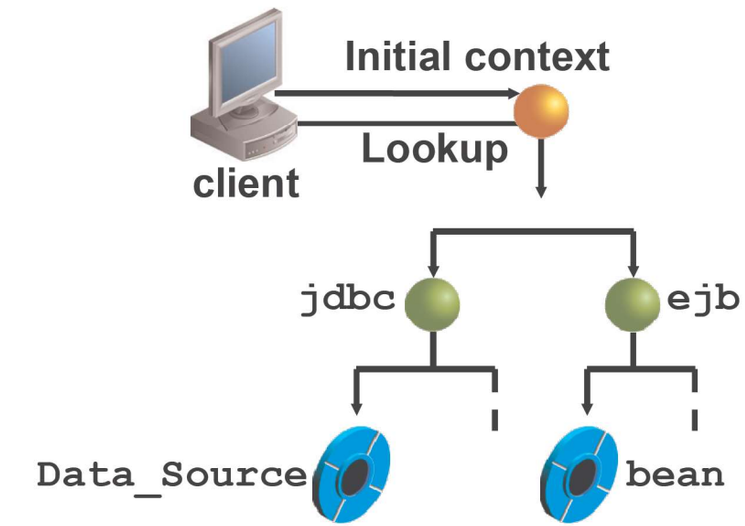
```
com.bea.examples.ejb.SomeObject
```

without first creating:

```
com.bea.examples.ejb
```

# Referencing Java EE Resources with JNDI

- Provide JNDI properties:
  - `java.naming.factory.initial`
  - `java.naming.security.principal`
  - `java.naming.security.credentials`
  - `java.naming.provider.url`




- Establish an initial context:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
                "weblogic.jndi.WLInitialContextFactory");
env.put(Context.PROVIDER_URL, t3://localhost:7001"); ...
Context ctx = new InitialContext(env);
```

- Perform the lookup operation:

```
import javax.sql.DataSource; ...
DataSource ds =
            (DataSource)ctx.lookup("jdbc/Data_Source");
```

# Providing JNDI Properties

JNDI properties may be provided as:

- A `jndi.properties` file in the `CLASSPATH` application

```
java.naming.factory.initial=weblogic.jndi.Environment.
                                        DEFAULT_INITIAL_CONTEXT_FACTORY
java.naming.provider.url=t3://localhost:7001
java.naming.security.principal=weblogic
java.naming.security.credentials=weblogic
```

- Entries added to a `java.util.Hashtable` object in the application code

```
import javax.naming.*;
...

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
                              "weblogic.jndi.WLInitialContextFactory");
env.put(Context.PROVIDER_URL, "t3://localhost:7001"); ...
Context ctx = new InitialContext(env);
```

# Referencing a Local Session EJB with JNDI

Within the same Java EE container (local reference):

- Define dependency in the deployment descriptor:
  - In `web.xml`, from a servlet or JSP
  - In `ejb-jar.xml`, from another session EJB

```
<ejb-local-ref>
  <ejb-ref-name>ejb/sessionBean</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local>ejb.EJBLocal</local>
</ejb-local-ref>
```

- Look up EJB using the reference name:

```
Context ctx = new InitialContext();
EJBLocal ref = (EJBLocal)
    ctx.lookup("java:comp/env/ejb/sessionBean");
...
```

# Referencing a Remote Session EJB with JNDI

From a different JVM or Java EE container (remote reference):

- Define dependency in the deployment descriptor:
  - In `web.xml`, from a servlet or JSP
  - In `ejb-jar.xml`, from another session EJB
  - In `application-client.xml`, for applications outside a container

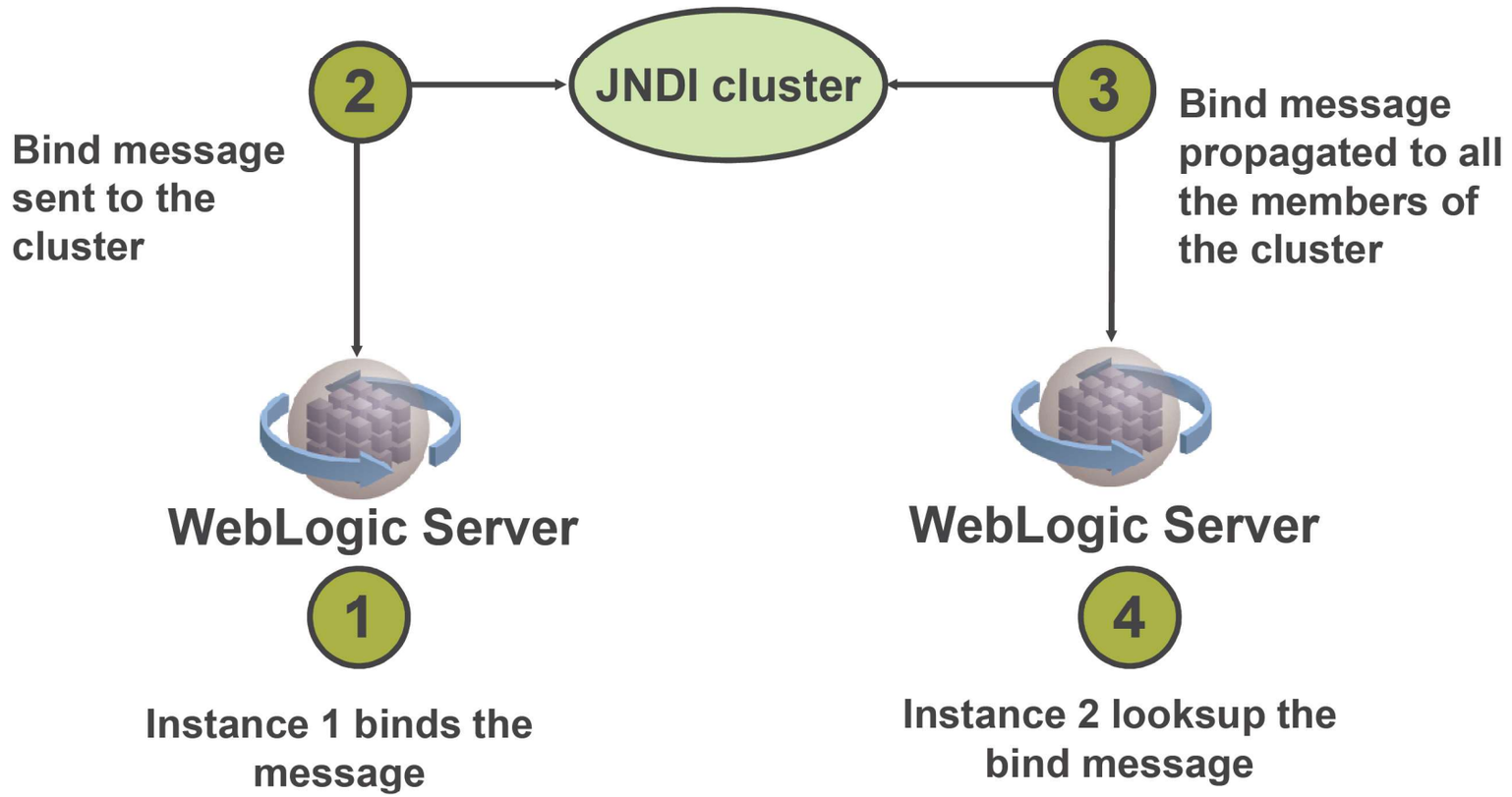
```
<ejb-ref>
  <ejb-ref-name>ejb/session</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <remote>ejb.EJBRemote</remote>
</ejb-ref>
```

- Look up EJB reference (provide `jndi.properties`):

```
Context ctx = new InitialContext();
EJBRemote ref = (EJBRemote) PortableRemoteObject.narrow(
   ctx.lookup("java:comp/env/ejb/session"),SessionBean.class);
...
```

# JNDI State Replication

- The JNDI state replication feature is being supported in a clustered WebLogic Servers environment.



When objects are bound to the JNDI tree, subcontexts are automatically created based on the JNDI name.

1. True
2. False

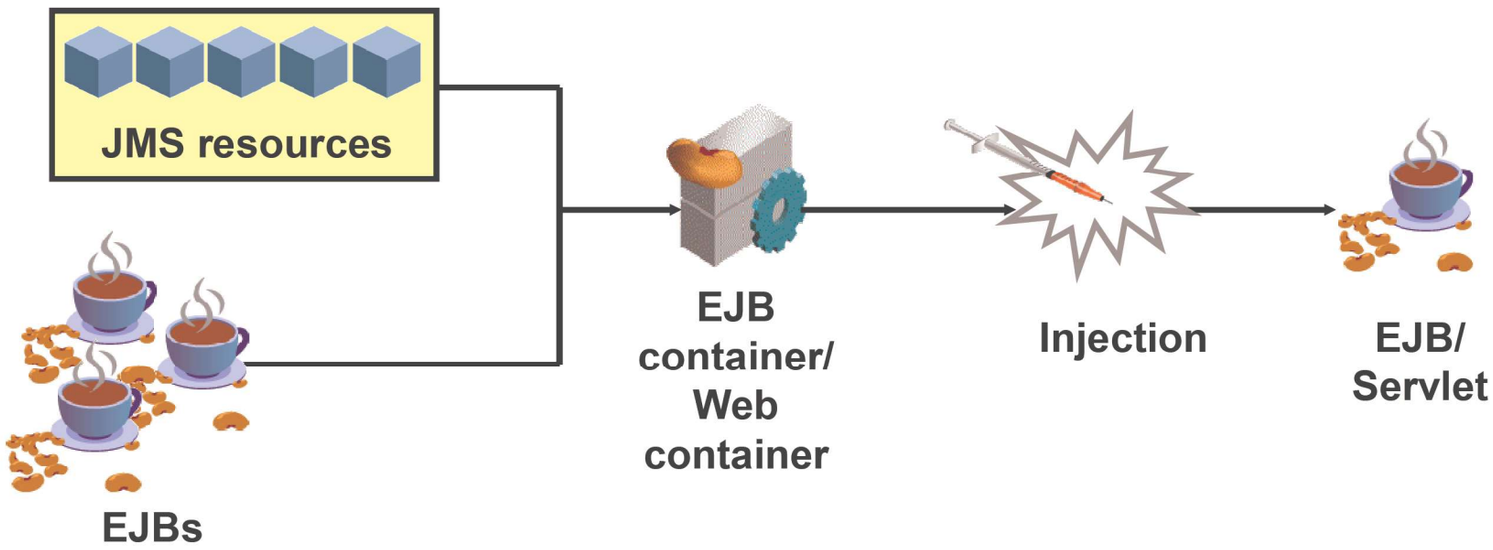
# What Are Annotations?

- They are additional information (metadata) in the operating code that instructs a development tool to process a Java class in a specific way.
- They are used to mark methods, classes, fields, parameters, variables, and constructors.

# Dependency Injection

The concept of dependency injection implies that it is:

- A programming design pattern that enables you to inject resources at run time
- Based on the principle of Inversion of Control (IOC)
- An alternative to the JNDI implementation



# Dependency Injection

Dependency injection is introduced in the Java EE 5.0 specification:

- Is based on declaring resources by using annotations (or deployment descriptors)
- Eliminates writing explicit JNDI API calls
- Is used by classes managed by a Java EE container
- Cannot be used by classes external to a Java EE container
- Is implemented by using the following key annotations:
  - `@Resource` (`javax.annotation.Resource`)
  - `@EJB` (`javax.ejb.EJB`)

# Types of Dependency Injection

The Java EE 5.0 specification supports the following types of dependency injection:

- Field injection, which injects a resource to a field

```
@Resource (name = "jdbc/fodDS")
private javax.sql.DataSource fodDS;
```

- Setter (property) injection, which injects a resource by invoking a setter method

```
private DataSource fodDS;
...
@Resource (name = "jdbc/fodDS")
private void setfodDS(javax.sql.DataSource ds) {
  fodDS = ds;
}
```

# Defining the @Resource Annotation

- The @Resource annotation definition is:

```
public @interface Resource {
 public enum AuthenticationType {CONTAINER, APPLICATION}
 String name() default "";
 Class type() default Object.class;
 AuthenticationType authenticationType()
   default  AuthenticationType.CONTAINER;
 boolean shareable() default true;
 String mappedName() default "";
 String description() default "";
}
```

- The @Resource annotation parameters can be:
  - Derived
  - Explicitly specified

# Using Java EE Resources with

Using Java EE resources with dependency injection is performed with the `@Resource` annotation. Examples are:

- Data sources:

```
@Resource(name="jdbc/fodDS")
private javax.sql.DataSource myDS;
Connection conn = myDS.getConnection();
```

- JMS resources (a queue or topic):

```
@Resource(name="jms/demoQueue")
private Queue myQueue;
```

- Environment entries:

```
@Resource int maxItemsPerCart = 20;
```

# Working with Dependency Injection

Dependency injection:

- Eliminates use of JNDI complexities
- Results in fewer lines of code
- Yields more concise code. However, annotations can:
  - Hard-code resources
  - Be overridden with XML elements in deployment descriptors
- Cannot be used with helper classes, which must use JNDI services

# Referencing EJBs with Dependency Injection

In EJB 3.0, dependency is expressed by using:

- The `@EJB` annotation:

```
@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface EJB {
  String name() default "";
  String beanName() default "";
  String mappedName() default "";
  String description() default "";
  Class beanInterface() default Object.class;
}
```

- The `<ejb-ref>` deployment descriptor element as an alternative method

# Practice: Overview

- These practices covers the following topics:
  - Using the `@Resource` annotation in a servlet application to access a database table
  - Using the `@EJB` annotation in a servlet application to inject an EJB

# Summary

In this lesson, you should have learned how to:

- Describe the Java Naming and Directory Interface (JNDI)
- Locate or look up resources and EJBs by using:
  - JNDI APIs
  - Dependency injection