Enterprise Beans

Part IV
Enterprise Beans

Part IV explores Enterprise JavaBeans components.

Enterprise Beans

This part contains the following chapters:

- Chapter 22, Enterprise Beans
- Chapter 23, Getting Started with Enterprise Beans
- Chapter 24, Running the Enterprise Bean Examples
- Chapter 25, A Message-Driven Bean Example
- Chapter 26, Using the Embedded Enterprise Bean Container
- Chapter 27, Using Asynchronous Method Invocation in Session Beans

Enterprise Beans

Chapter 22
Enterprise Beans

Enterprise beans are Java EE components that implement Enterprise JavaBeans (EJB) technology.

Enterprise beans run in the EJB container, a runtime environment within the GlassFish Server (see Container Types).

Enterprise Beans

Although transparent to the application developer, the EJB container provides system-level services, such as transactions and security, to its enterprise beans.

These services enable you to quickly build and deploy enterprise beans, which form the core of transactional Java EE applications.

Enterprise Beans

The following topics are addressed here:

- What Is an Enterprise Bean?
- What Is a Session Bean?
- What Is a Message-Driven Bean?
- Accessing Enterprise Beans
- The Contents of an Enterprise Bean
- Naming Conventions for Enterprise Beans
- The Lifecycles of Enterprise Beans
- Further Information about Enterprise Beans

Enterprise Beans

What Is an Enterprise Bean?

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application.

The business logic is the code that fulfills the purpose of the application.

Enterprise Beans

In an inventory control application, for example, the enterprise beans might implement the business logic in methods called `checkInventoryLevel` and `orderProduct`.

By invoking these 32–bit methods, clients can access the inventory services provided by the application.

Enterprise Beans

Benefits of Enterprise Beans

For several reasons, enterprise beans simplify the development of large, distributed applications.

First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems.

Enterprise Beans

The EJB container, rather than the bean developer, is responsible for system-level services, such as transaction management and security authorization.

Second, because the beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client.

Enterprise Beans

The client developer does not have to code the routines that implement business rules or access databases.

As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Enterprise Beans

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans.

Provided that they use the standard APIs, these applications can run on any compliant Java EE server.

Enterprise Beans

When to Use Enterprise Beans

You should consider using enterprise beans if your application has any of the following requirements.

- The application must be scalable.

Enterprise Beans

To accommodate a growing number of users, you may need to distribute an application's components across multiple machines.

Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.

Enterprise Beans

- Transactions must ensure data integrity.

  Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.

- The application will have a variety of clients.

  With only a few lines of code, remote clients can easily locate enterprise beans.

Enterprise Beans

These clients can be thin, various, and numerous.

Enterprise Beans

Types of Enterprise Beans

Table 22-1 summarizes the two types of enterprise beans.

The following sections discuss each type in more detail.

Enterprise Beans

Table 22-1 Enterprise Bean Types

Enterprise Bean Type	Purpose
Session	Performs a task for a client; optionally, may implement a web service
Message-driven	Acts as a listener for a particular messaging type, such as the Java Message Service API

Enterprise Beans

What Is a Session Bean?

A session bean encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views.

To access an application that is deployed on the server, the client invokes the session bean's methods.

Enterprise Beans

The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server.

A session bean is not persistent.

(That is, its data is not saved to a database.)

For code samples, see Chapter 24, Running the Enterprise Bean Examples.

Enterprise Beans

Types of Session Beans

Session beans are of three types: stateful, stateless, and singleton.

Enterprise Beans

Stateful Session Beans

The state of an object consists of the values of its instance variables.

In a stateful session bean, the instance variables represent the state of a unique client/bean session.

Enterprise Beans

Because the client interacts ("talks") with its bean, this state is often called the conversational state.

As its name suggests, a session bean is similar to an interactive session.

A session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user.

Enterprise Beans

When the client terminates, its session bean appears to terminate and is no longer associated with the client.

The state is retained for the duration of the client/bean session.

If the client removes the bean, the session ends and the state disappears.

Enterprise Beans

This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends, there is no need to retain the state.

Enterprise Beans

Stateless Session Beans

A stateless session bean does not maintain a conversational state with the client.

When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation.

Enterprise Beans

When the method is finished, the client-specific state should not be retained.

Clients may, however, change the state of instance variables in pooled stateless beans, and this state is held over to the next invocation of the pooled stateless bean.

Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client.

Enterprise Beans

That is, the state of a stateless session bean should apply across all clients.

Because they can support multiple clients, stateless session beans can offer better scalability for applications that require large numbers of clients.

Enterprise Beans

Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.

A stateless session bean can implement a web service, but a stateful session bean cannot.

Enterprise Beans

Singleton Session Beans

A singleton session bean is instantiated once per application and exists for the lifecycle of the application.

Singleton session beans are designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients.

Enterprise Beans

Singleton session beans offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean per application, as opposed to a pool of stateless session beans, any of which may respond to a client request.

Like stateless session beans, singleton session beans can implement web service endpoints.

Enterprise Beans

Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.

Applications that use a singleton session bean may specify that the singleton should be instantiated upon application startup, which allows the singleton to perform initialization tasks for the application.

Enterprise Beans

The singleton may perform cleanup tasks on application shutdown as well, because the singleton will operate throughout the lifecycle of the application.

Enterprise Beans

When to Use Session Beans

Stateful session beans are appropriate if any of the following conditions are true.

- The bean's state represents the interaction between the bean and a specific client.
- The bean needs to hold information about the client across method invocations.

Enterprise Beans

- The bean mediates between the client and the other components of the application, presenting a simplified view to the client.
- Behind the scenes, the bean manages the work flow of several enterprise beans.

To improve performance, you might choose a stateless session bean if it has any of these traits.

Enterprise Beans

- The bean's state has no data for a specific client.
- In a single method invocation, the bean performs a generic task for all clients.

  For example, you might use a stateless session bean to send an email that confirms an online order.
- The bean implements a web service.

Enterprise Beans

Singleton session beans are appropriate in the following circumstances.

- State needs to be shared across the application.
- A single enterprise bean needs to be accessed by multiple threads concurrently.
- The application needs an enterprise bean to perform tasks upon application startup and shutdown.
- The bean implements a web service.

Enterprise Beans

What Is a Message-Driven Bean?

A message-driven bean is an enterprise bean that allows Java EE applications to process messages asynchronously.

This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events.

Enterprise Beans

The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology.

Message-driven beans can process JMS messages or other kinds of messages.

Enterprise Beans

What Makes Message-Driven Beans Different from Session Beans?

The most visible difference between message-driven beans and session beans is that clients do not access message-driven beans through interfaces.

Interfaces are described in the section Accessing Enterprise Beans.

Enterprise Beans

Unlike a session bean, a message-driven bean has only a bean class.

In several respects, a message-driven bean resembles a stateless session bean.

- A message-driven bean's instances retain no data or conversational state for a specific client.

Enterprise Beans

- All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance.

  The container can pool these instances to allow streams of messages to be processed concurrently.

Enterprise Beans

- A single message-driven bean can process messages from multiple clients.

The instance variables of the message-driven bean instance can contain some state across the handling of client messages, such as a JMS API connection, an open database connection, or an object reference to an enterprise bean object.

Enterprise Beans

Client components do not locate message-driven beans and invoke methods directly on them.

Instead, a client accesses a message-driven bean through, for example, JMS by sending messages to the message destination for which the message-driven bean class is the `MessageListener`.

Enterprise Beans

You assign a message-driven bean's destination during deployment by using GlassFish Server resources.

Message-driven beans have the following characteristics.

- They execute upon receipt of a single client message.
- They are invoked asynchronously.
- They are relatively short-lived.

Enterprise Beans

- They do not represent directly shared data in the database, but they can access and update this data.
- They can be transaction-aware.
- They are stateless.

When a message arrives, the container calls the message-driven bean's `onMessage` method to process the message.

Enterprise Beans

The `onMessage` method normally casts the message to one of the five JMS message types and handles it in accordance with the application's business logic.

The `onMessage` method can call helper methods or can invoke a session bean to process the information in the message or to store it in a database.

Enterprise Beans

A message can be delivered to a message-driven bean within a transaction context, so all operations within the `onMessage` method are part of a single transaction.

If message processing is rolled back, the message will be redelivered.

For more information, see Chapter 43, Transactions.

Enterprise Beans

When to Use Message-Driven Beans

Session beans allow you to send JMS messages and to receive them synchronously but not asynchronously.

To avoid tying up server resources, do not to use blocking synchronous receives in a server-side component; in general, JMS messages should not be sent or received synchronously.

Enterprise Beans

To receive messages asynchronously, use a message-driven bean.

Enterprise Beans

Accessing Enterprise Beans

Note - The material in this section applies only to session beans and not to message-driven beans.

Because they have a different programming model, message-driven beans do not have interfaces or no-interface views that define client access.

Enterprise Beans

Clients access enterprise beans either through a no-interface view or through a business interface.

A no-interface view of an enterprise bean exposes the public methods of the enterprise bean implementation class to clients.

Enterprise Beans

Clients using the no-interface view of an enterprise bean may invoke any public methods in the enterprise bean implementation class or any superclasses of the implementation class.

A business interface is a standard Java programming language interface that contains the business methods of the enterprise bean.

Enterprise Beans

A client can access a session bean only through the methods defined in the bean's business interface or through the public methods of an enterprise bean that has a no-interface view.

The business interface or no-interface view defines the client's view of an enterprise bean.

All other aspects of the enterprise bean (method implementations and deployment settings) are hidden from the client.

Enterprise Beans

Well-designed interfaces and no-interface views simplify the development and maintenance of Java EE applications.

Not only do clean interfaces and no-interface views shield the clients from any complexities in the EJB tier, but they also allow the enterprise beans to change internally without affecting the clients.

Enterprise Beans

For example, if you change the implementation of a session bean business method, you won't have to alter the client code.

But if you were to change the method definitions in the interfaces, you might have to modify the client code as well.

Enterprise Beans

Therefore, it is important that you design the interfaces and no-interface views carefully to isolate your clients from possible changes in the enterprise beans.

Session beans can have more than one business interface.

Session beans should, but are not required to, implement their business interface or interfaces.

Enterprise Beans

Using Enterprise Beans in Clients

The client of an enterprise bean obtains a reference to an instance of an enterprise bean through either dependency injection, using Java programming language annotations, or JNDI lookup, using the Java Naming and Directory Interface syntax to find the enterprise bean instance.

Enterprise Beans

Dependency injection is the simplest way of obtaining an enterprise bean reference.

Clients that run within a Java EE server-managed environment, JavaServer Faces web applications, JAX-RS web services, other enterprise beans, or Java EE application clients, support dependency injection using the `javax.ejb.EJB` annotation.

Enterprise Beans

Applications that run outside a Java EE server-managed environment, such as Java SE applications, must perform an explicit lookup.

JNDI supports a global syntax for identifying Java EE components to simplify this explicit lookup.

Enterprise Beans

Portable JNDI Syntax

Three JNDI namespaces are used for portable JNDI lookups: `java:global`, `java:module`, and `java:app`.

- The `java:global` JNDI namespace is the portable way of finding remote enterprise beans using JNDI lookups.

Enterprise Beans

JNDI addresses are of the following form:

```
java:global
[/application name]
/module name/enterprise bean
name[/interface name]
```

Application name and module name default to the name of the application and module minus the file extension.

Enterprise Beans

Application names are required only if the application is packaged within an EAR.

The interface name is required only if the enterprise bean implements more than one business interface.

- The `java:module` namespace is used to look up local enterprise beans within the same module.

Enterprise Beans

JNDI addresses using the `java:module` namespace are of the following form:

`java:module/`*`enterprise bean name`*`/[`*`interface name`*`]`

The interface name is required only if the enterprise bean implements more than one business interface.

Enterprise Beans

- The `java:app` namespace is used to look up local enterprise beans packaged within the same application.

  That is, the enterprise bean is packaged within an EAR file containing multiple Java EE modules.

  JNDI addresses using the `java:app` namespace are of the following form:

Enterprise Beans

```
java:app[/module name]
/enterprise bean name
[/interface name]
```

The module name is optional.

The interface name is required only if the enterprise bean implements more than one business interface.

Enterprise Beans

For example, if an enterprise bean, `MyBean`, is packaged within the web application archive `myApp.war`, the module name is `myApp`.

The portable JNDI name is `java:module/MyBean` An equivalent JNDI name using the `java:global` namespace is `java:global/myApp/MyBean`.

Enterprise Beans

Deciding on Remote or Local Access

When you design a Java EE application, one of the first decisions you make is the type of client access allowed by the enterprise beans: remote, local, or web service.

Whether to allow local or remote access depends on the following factors.

Enterprise Beans

- **Tight or loose coupling of related beans:**
  Tightly coupled beans depend on one another.

  For example, if a session bean that processes sales orders calls a session bean that emails a confirmation message to the customer, these beans are tightly coupled.

  Tightly coupled beans are good candidates for local access.

Enterprise Beans

Because they fit together as a logical unit, they typically call each other often and would benefit from the increased performance that is possible with local access.

- Type of client: If an enterprise bean is accessed by application clients, it should allow remote access.

Enterprise Beans

In a production environment, these clients almost always run on machines other than those on which the GlassFish Server is running.

If an enterprise bean's clients are web components or other enterprise beans, the type of access depends on how you want to distribute your components.

Enterprise Beans

- **Component distribution: Java EE applications are scalable because their server-side components can be distributed across multiple machines.**

  In a distributed application, for example, the server that the web components run on may not be the one on which the enterprise beans they access are deployed.

Enterprise Beans

In this distributed scenario, the enterprise beans should allow remote access.

- Performance: Owing to such factors as network latency, remote calls may be slower than local calls.

On the other hand, if you distribute components among different servers, you may improve the application's overall performance.

Enterprise Beans

Both of these statements are generalizations; performance can vary in different operational environments.

Nevertheless, you should keep in mind how your application design might affect performance.

Enterprise Beans

If you aren't sure which type of access an enterprise bean should have, choose remote access.

This decision gives you more flexibility.

In the future, you can distribute your components to accommodate the growing demands on your application.

Enterprise Beans

Although it is uncommon, it is possible for an enterprise bean to allow both remote and local access.

If this is the case, either the business interface of the bean must be explicitly designated as a business interface by being decorated with the `@Remote` or `@Local` annotations, or the bean class must explicitly designate the business interfaces by using the `@Remote` and `@Local` annotations.

Enterprise Beans

The same business interface cannot be both a local and a remote business interface.

Enterprise Beans

Local Clients

A local client has these characteristics.

- It must run in the same application as the enterprise bean it accesses.
- It can be a web component or another enterprise bean.

Enterprise Beans

- To the local client, the location of the enterprise bean it accesses is not transparent.

The no-interface view of an enterprise bean is a local view.

The public methods of the enterprise bean implementation class are exposed to local clients that access the no-interface view of the enterprise bean.

Enterprise Beans

Enterprise beans that use the no-interface view do not implement a business interface.

The local business interface defines the bean's business and lifecycle methods.

Enterprise Beans

If the bean's business interface is not decorated with `@Local` or `@Remote`, and if the bean class does not specify the interface using `@Local` or `@Remote`, the business interface is by default a local interface.

To build an enterprise bean that allows only local access, you may, but are not required to, do one of the following:

Enterprise Beans

- Create an enterprise bean implementation class that does not implement a business interface, indicating that the bean exposes a no-interface view to clients.

For example:

```
@Session
public class MyBean { ... }
```

Enterprise Beans

- Annotate the business interface of the enterprise bean as a `@Local` interface.

For example:

```
@Local
public interface InterfaceName
{...}
```

Enterprise Beans

- Specify the interface by decorating the bean class with `@Local` and specify the interface name.

For example:

```
@Local(InterfaceName.class)
public class BeanName implements
InterfaceName { ... }
```

Enterprise Beans

Accessing Local Enterprise Beans Using the No-Interface View

Client access to an enterprise bean that exposes a local, no-interface view is accomplished through either dependency injection or JNDI lookup.

Enterprise Beans

- To obtain a reference to the no-interface view of an enterprise bean through dependency injection, use the `javax.ejb.EJB` annotation and specify the enterprise bean's implementation class:

```
@EJB
ExampleBean exampleBean;
```

Enterprise Beans

- To obtain a reference to the no-interface view of an enterprise bean through JNDI lookup, use the `javax.naming.InitialContext` interface's `lookup` method:

```
ExampleBean exampleBean =
(ExampleBean)
InitialContext.lookup
("java:module/ExampleBean");
```

Enterprise Beans

Clients do not use the `new` operator to obtain a new instance of an enterprise bean that uses a no-interface view.

Enterprise Beans

Accessing Local Enterprise Beans That Implement Business Interfaces

Client access to enterprise beans that implement local business interfaces is accomplished through either dependency injection or JNDI lookup.

Enterprise Beans

- To obtain a reference to the local business interface of an enterprise bean through dependency injection, use the `javax.ejb.EJB` annotation and specify the enterprise bean's local business interface name:

```
@EJB
Example example;
```

Enterprise Beans

- To obtain a reference to a local business interface of an enterprise bean through JNDI lookup, use the `javax.naming.InitialContext` interface's `lookup` method:

```
ExampleLocal example =
(ExampleLocal)
InitialContext.lookup
("java:module/ExampleLocal");
```

Enterprise Beans

Remote Clients

A remote client of an enterprise bean has the following traits.

- It can run on a different machine and a different JVM from the enterprise bean it accesses.
  (It is not required to run on a different JVM.)

Enterprise Beans

- It can be a web component, an application client, or another enterprise bean.
- To a remote client, the location of the enterprise bean is transparent.
- The enterprise bean must implement a business interface.

That is, remote clients may not access an enterprise bean through a no-interface view.

Enterprise Beans

To create an enterprise bean that allows remote access, you must either

- Decorate the business interface of the enterprise bean with the `@Remote` annotation:

```
@Remote
public interface InterfaceName
{ ... }
```

Enterprise Beans

- Decorate the bean class with `@Remote`, specifying the business interface or interfaces:

```
@Remote(InterfaceName.class)
public class BeanName implements
InterfaceName { ... }
```

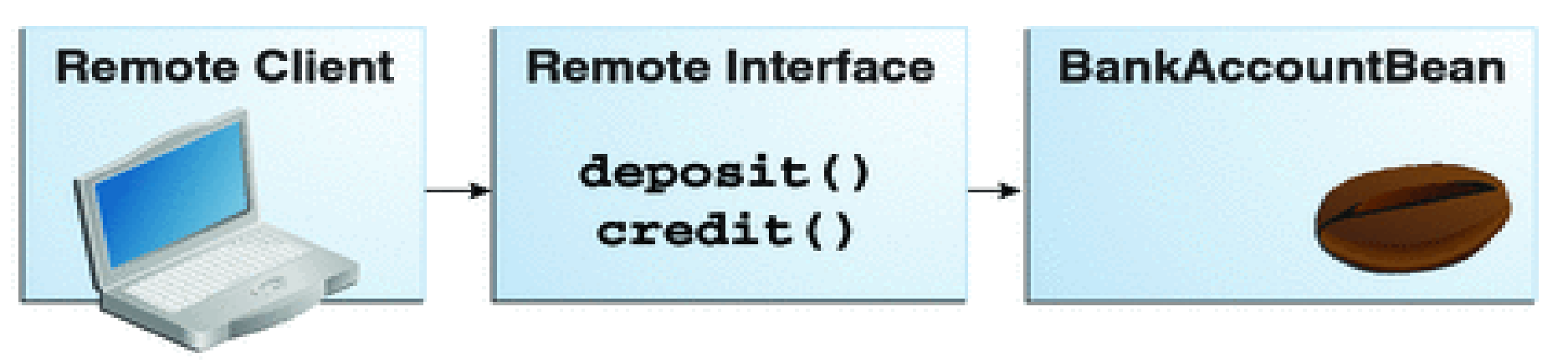
The remote interface defines the business and lifecycle methods that are specific to the bean.

Enterprise Beans

For example, the remote interface of a bean named `BankAccountBean` might have business methods named `deposit` and `credit`.

Figure 22-1 shows how the interface controls the client's view of an enterprise bean.

Figure 22-1 Interfaces for an Enterprise Bean with Remote Access



Enterprise Beans

Client access to an enterprise bean that implements a remote business interface is accomplished through either dependency injection or JNDI lookup.

- To obtain a reference to the remote business interface of an enterprise bean through dependency injection, use the `javax.ejb.EJB` annotation and specify the enterprise bean's remote business interface name:

Enterprise Beans

```
@EJB
Example example;
```

- To obtain a reference to a remote business interface of an enterprise bean through JNDI lookup, use the `javax.naming.InitialContext` interface's `lookup` method:

Enterprise Beans

```
ExampleRemote example =
(ExampleRemote)
InitialContext.lookup
("java:global/myApp/ExampleRemote");
```

Enterprise Beans

Web Service Clients

A web service client can access a Java EE application in two ways.

First, the client can access a web service created with JAX-WS.

Enterprise Beans

(For more information on JAX-WS, see Chapter 18, Building Web Services with JAX-WS.)

Second, a web service client can invoke the business methods of a stateless session bean.

Message beans cannot be accessed by web service clients.

Enterprise Beans

Provided that it uses the correct protocols (SOAP, HTTP, WSDL), any web service client can access a stateless session bean, whether or not the client is written in the Java programming language.

The client doesn't even "know" what technology implements the service: stateless session bean, JAX-WS, or some other technology.

Enterprise Beans

In addition, enterprise beans and web components can be clients of web services.

This flexibility enables you to integrate Java EE applications with web services.

A web service client accesses a stateless session bean through the bean's web service endpoint implementation class.

Enterprise Beans

By default, all public methods in the bean class are accessible to web service clients.

The `@WebMethod` annotation may be used to customize the behavior of web service methods.

If the `@WebMethod` annotation is used to decorate the bean class's methods, only those methods decorated with `@WebMethod` are exposed to web service clients.

Enterprise Beans

For a code sample, see A Web Service Example: `helloservice`.

Enterprise Beans

Method Parameters and Access

The type of access affects the parameters of the bean methods that are called by clients.

The following sections apply not only to method parameters but also to method return values.

Enterprise Beans

Isolation

The parameters of remote calls are more isolated than those of local calls.

With remote calls, the client and the bean operate on different copies of a parameter object.

If the client changes the value of the object, the value of the copy in the bean does not change.

Enterprise Beans

This layer of isolation can help protect the bean if the client accidentally modifies the data.

In a local call, both the client and the bean can modify the same parameter object.

In general, you should not rely on this side effect of local calls.

Enterprise Beans

Perhaps someday you will want to distribute your components, replacing the local calls with remote ones.

As with remote clients, web service clients operate on different copies of parameters than does the bean that implements the web service.

Enterprise Beans

Granularity of Accessed Data

Because remote calls are likely to be slower than local calls, the parameters in remote methods should be relatively coarse-grained.

A coarse-grained object contains more data than a fine-grained one, so fewer access calls are required.

Enterprise Beans

For the same reason, the parameters of the methods called by web service clients should also be coarse-grained.

Enterprise Beans

The Contents of an Enterprise Bean

To develop an enterprise bean, you must provide the following files:

- Enterprise bean class: Implements the business methods of the enterprise bean and any lifecycle callback methods.

Enterprise Beans

- Business interfaces: Define the business methods implemented by the enterprise bean class.

A business interface is not required if the enterprise bean exposes a local, no-interface view.

Enterprise Beans

- **Helper classes: Other classes needed by the enterprise bean class, such as exception and utility classes.**

Package the programming artifacts in the preceding list either into an EJB JAR file (a stand-alone module that stores the enterprise bean) or within a web application archive (WAR) module.

Enterprise Beans

Packaging Enterprise Beans in EJB JAR Modules

An EJB JAR file is portable and can be used for various applications.

Enterprise Beans

To assemble a Java EE application, package one or more modules, such as EJB JAR files, into an EAR file, the archive file that holds the application.

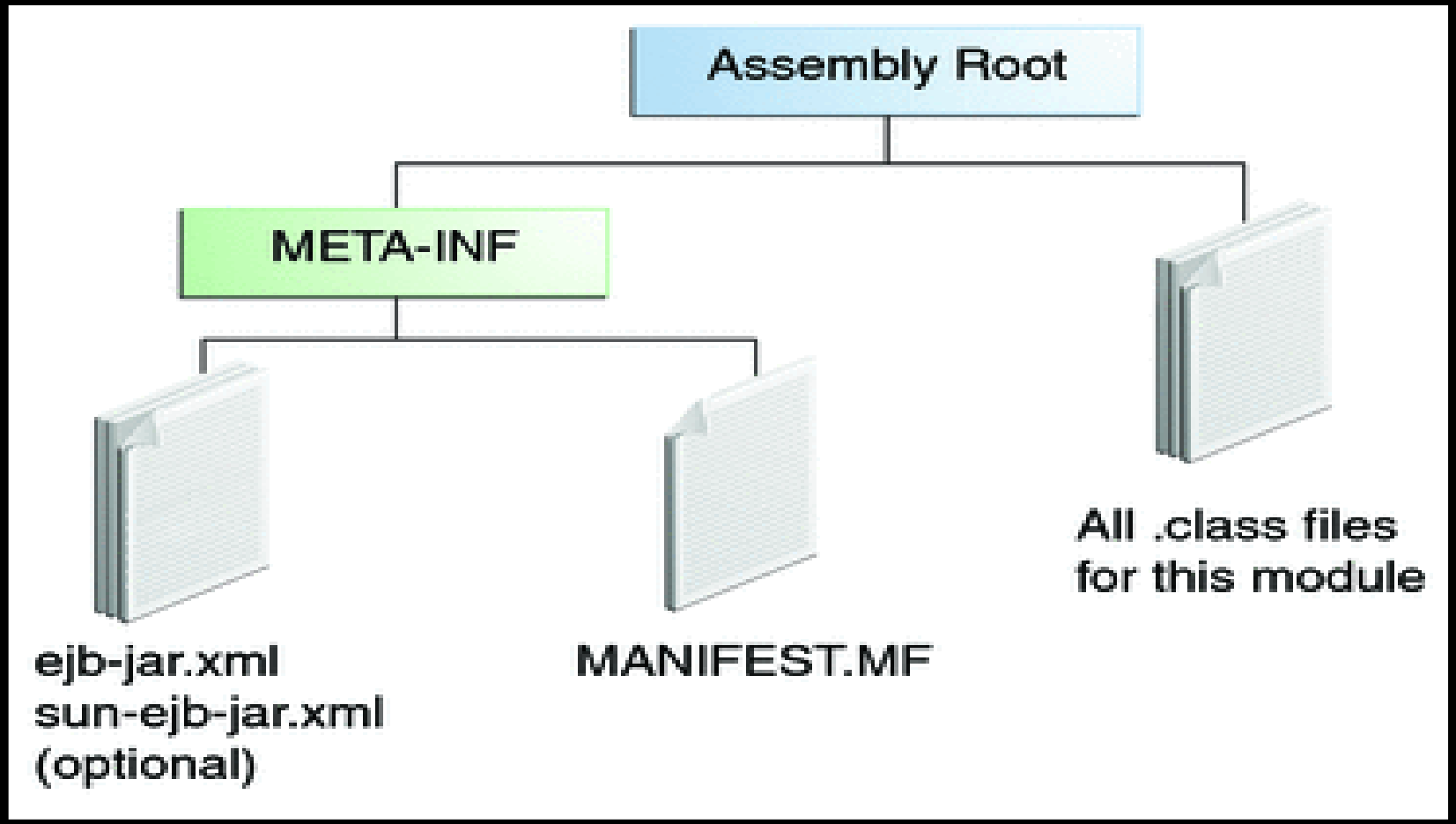
When deploying the EAR file that contains the enterprise bean's EJB JAR file, you also deploy the enterprise bean to the GlassFish Server.

You can also deploy an EJB JAR that is not contained in an EAR file.

Enterprise Beans

Figure 22-2 shows the contents of an EJB JAR file.

Figure 22-2 Structure of an Enterprise Bean JAR



Enterprise Beans

Packaging Enterprise Beans in WAR Modules

Enterprise beans often provide the business logic of a web application.

In these cases, packaging the enterprise bean within the web application's WAR module simplifies deployment and application organization.

Enterprise Beans

Enterprise beans may be packaged within a WAR module as Java programming language class files or within a JAR file that is bundled within the WAR module.

To include enterprise bean class files in a WAR module, the class files should be in the `WEB-INF/classes` directory.

Enterprise Beans

To include a JAR file that contains enterprise beans in a WAR module, add the JAR to the `WEB-INF/lib` directory of the WAR module.

WAR modules that contain enterprise beans do not require an `ejb-jar.xml` deployment descriptor.

Enterprise Beans

If the application uses `ejb-jar.xml`, it must be located in the WAR module's `WEB-INF` directory.

JAR files that contain enterprise bean classes packaged within a WAR module are not considered EJB JAR files, even if the bundled JAR file conforms to the format of an EJB JAR file.

Enterprise Beans

The enterprise beans contained within the JAR file are semantically equivalent to enterprise beans located in the WAR module's `WEB-INF/classes` directory, and the environment namespace of all the enterprise beans are scoped to the WAR module.

Enterprise Beans

For example, suppose that a web application consists of a shopping cart enterprise bean, a credit card processing enterprise bean, and a Java servlet front end.

The shopping cart bean exposes a local, no-interface view and is defined as follows:

Enterprise Beans

```
package com.example.cart;
@Stateless
public class CartBean { ... }
```

The credit card processing bean is packaged within its own JAR file, `cc.jar`, exposes a local, no-interface view, and is defined as follows:

```
package com.example.cc;
@Stateless
public class CreditCardBean{...}
```

Enterprise Beans

The servlet, `com.example.web.StoreServlet`, handles the web front end and uses both `CartBean` and `CreditCardBean`.

The WAR module layout for this application looks as follows:

```
WEB-INF/classes/com/example/cart/
CartBean.class
```

Enterprise Beans

```
WEB-INF/classes/com/example/web/
StoreServlet

WEB-INF/lib/cc.jar

WEB-INF/ejb-jar.xml

WEB-INF/web.xml
```

Enterprise Beans

Naming Conventions for Enterprise Beans

Because enterprise beans are composed of multiple parts, it's useful to follow a naming convention for your applications.

Table 22-2 summarizes the conventions for the example beans in this tutorial.

Enterprise Beans

Table 22-2 Naming Conventions for Enterprise Beans

Item	Syntax	Example
Enterprise bean name	*name*`Bean`	`AccountBean`
Enterprise bean class	*name*`Bean`	`AccountBean`
Business interface	*name*	`Account`

Enterprise Beans

The Lifecycles of Enterprise Beans

An enterprise bean goes through various stages during its lifetime, or lifecycle.

Each type of enterprise bean (stateful session, stateless session, singleton session, or message-driven) has a different lifecycle.

Enterprise Beans

The descriptions that follow refer to methods that are explained along with the code examples in the next two chapters.

If you are new to enterprise beans, you should skip this section and run the code examples first.

Enterprise Beans

The Lifecycle of a Stateful Session Bean

Figure 22-3 illustrates the stages that a session bean passes through during its lifetime.

The client initiates the lifecycle by obtaining a reference to a stateful session bean.

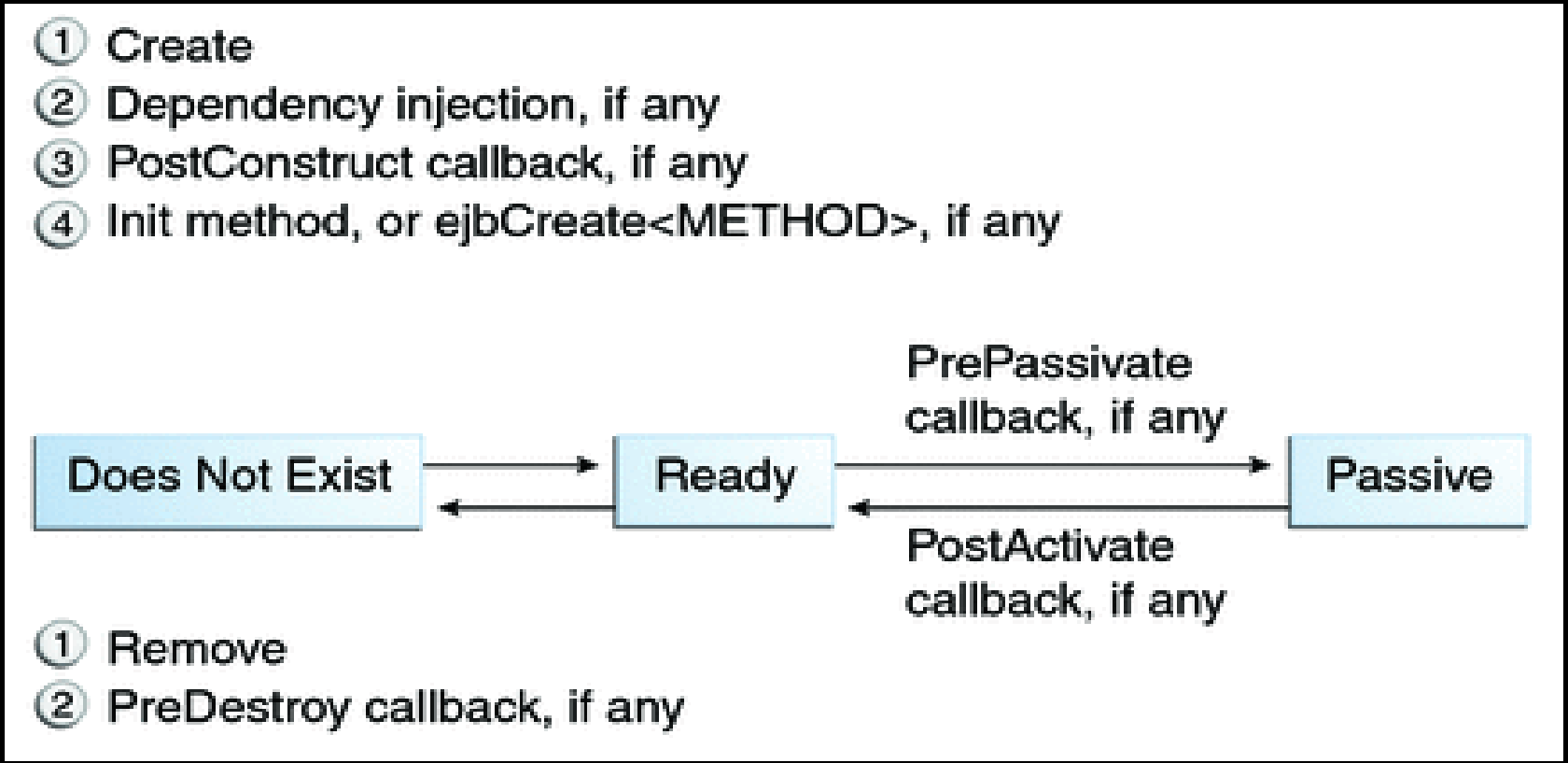
Enterprise Beans

The container performs any dependency injection and then invokes the method annotated with `@PostConstruct`, if any.

The bean is now ready to have its business methods invoked by the client.

Enterprise Beans

Figure 22-3 Lifecycle of a Stateful Session Bean



While in the ready stage, the EJB container may decide to deactivate, or passivate, the bean by moving it from memory to secondary storage.

Enterprise Beans

(Typically, the EJB container uses a least-recently-used algorithm to select a bean for passivation.) The EJB container invokes the method annotated `@PrePassivate`, if any, immediately before passivating it.

Enterprise Beans

If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean, calls the method annotated `@PostActivate`, if any, and then moves it to the ready stage.

At the end of the lifecycle, the client invokes a method annotated `@Remove`, and the EJB container calls the method annotated `@PreDestroy`, if any.

Enterprise Beans

The bean's instance is then ready for garbage collection.

Your code controls the invocation of only one lifecycle method: the method annotated `@Remove`.

All other methods in Figure 22-3 are invoked by the EJB container. See Chapter 44, Resource Connections for more information.

Enterprise Beans

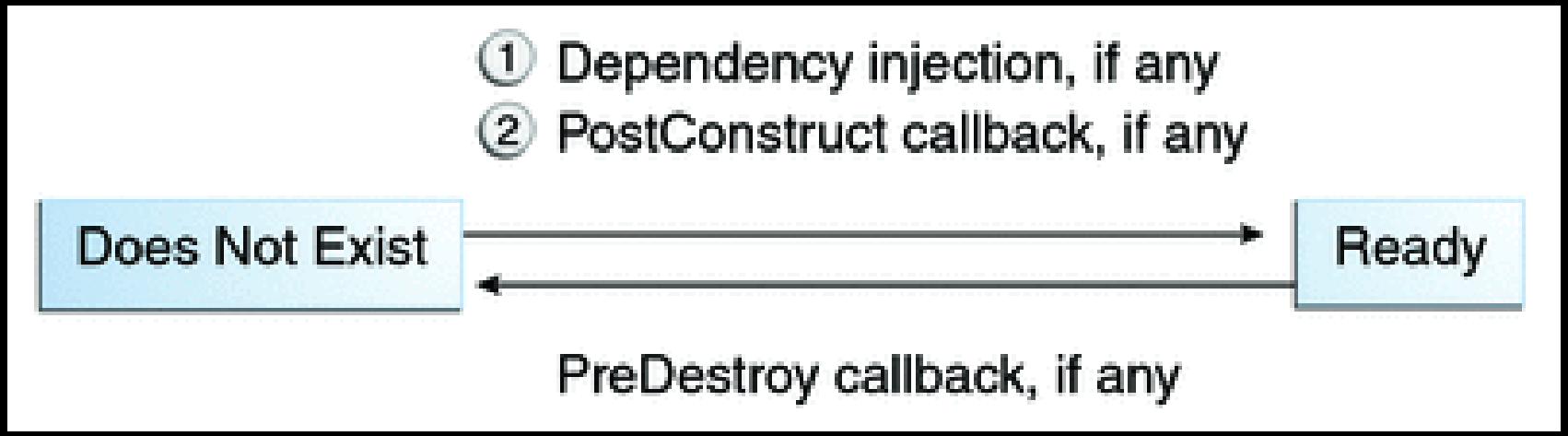
The Lifecycle of a Stateless Session Bean

Because a stateless session bean is never passivated, its lifecycle has only two stages: nonexistent and ready for the invocation of business methods.

Figure 22-4 illustrates the stages of a stateless session bean.

Enterprise Beans

Figure 22-4 Lifecycle of a Stateless Session Bean



The EJB container typically creates and maintains a pool of stateless session beans, beginning the stateless session bean's lifecycle.

Enterprise Beans

The container performs any dependency injection and then invokes the method annotated `@PostConstruct`, if it exists.

The bean is now ready to have its business methods invoked by a client.

At the end of the lifecycle, the EJB container calls the method annotated `@PreDestroy`, if it exists.

Enterprise Beans

The bean's instance is then ready for garbage collection.

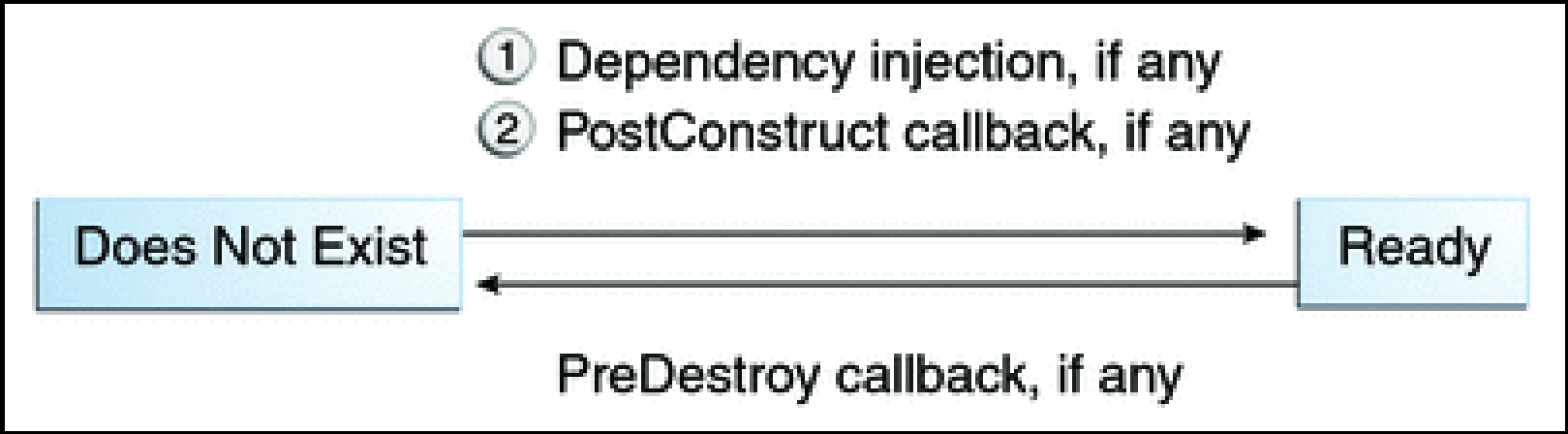
Enterprise Beans

The Lifecycle of a Singleton Session Bean

Like a stateless session bean, a singleton session bean is never passivated and has only two stages, nonexistent and ready for the invocation of business methods, as shown in Figure 22-5.

Enterprise Beans

Figure 22-5 Lifecycle of a Singleton Session Bean



The EJB container initiates the singleton session bean lifecycle by creating the singleton instance.

Enterprise Beans

This occurs upon application deployment if the singleton is annotated with the `@Startup` annotation The container performs any dependency injection and then invokes the method annotated `@PostConstruct`, if it exists.

The singleton session bean is now ready to have its business methods invoked by the client.

Enterprise Beans

At the end of the lifecycle, the EJB container calls the method annotated `@PreDestroy`, if it exists.

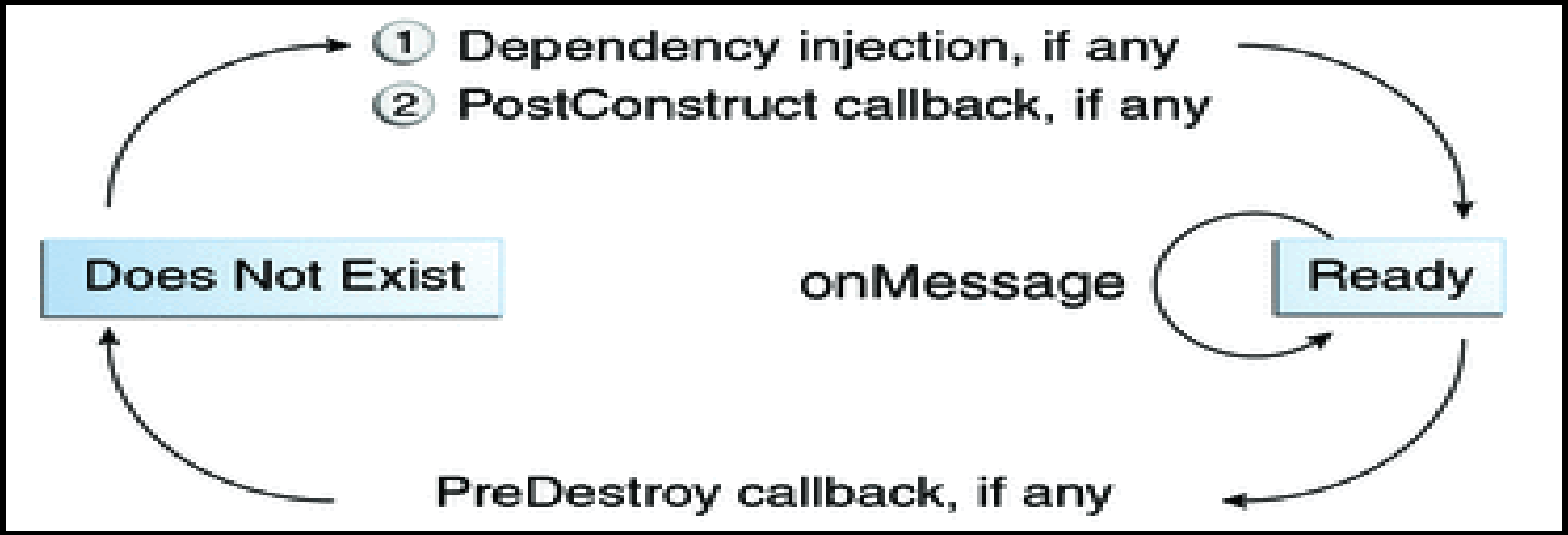
The singleton session bean is now ready for garbage collection.

Enterprise Beans

The Lifecycle of a Message-Driven Bean

Figure 22-6 illustrates the stages in the lifecycle of a message-driven bean.

Figure 22-6 Lifecycle of a Message-Driven Bean



Enterprise Beans

The EJB container usually creates a pool of message-driven bean instances.

For each instance, the EJB container performs these tasks.

1. If the message-driven bean uses dependency injection, the container injects these references before instantiating the instance.
2. The container calls the method annotated `@PostConstruct`, if any.

Enterprise Beans

Like a stateless session bean, a message-driven bean is never passivated and has only two states: nonexistent and ready to receive messages.

At the end of the lifecycle, the container calls the method annotated `@PreDestroy`, if any.

The bean's instance is then ready for garbage collection.

Enterprise Beans

Further Information about Enterprise Beans

For more information on Enterprise JavaBeans technology, see

- Enterprise JavaBeans 3.1 specification:

  http://jcp.org/en/jsr/summary?id=318

- Enterprise JavaBeans web site:

  http://www.oracle.com/technetwork/java/ejb-141389.html