



## Enterprise Computing with Java

### MCA-305 UNIT IV

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason

U IV

---

---

---

---

---

---

---

---



## Learning Objectives

- Introducing session beans:
- Session beans life time
- Statefull and Stateless session beans
- Lifecycle of session beans.
- Introducing Entity beans
- Persistence concepts
- Features of entity beans
- Entity context,
- Introduction to JMS & Message driven beans.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason

U IV

---

---

---

---

---

---

---

---



## Session Bean

- A session bean represents work being performed for client code that is calling it.
- Session beans are business process objects that implement business logic, business rules, algorithms, and workflow.
- For example, a session bean can perform price quoting, order entry, video compression, banking transactions, stock trades, database operations, complex calculations, and more.
- They are reusable components that contain logic for business processes.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason

U IV

---

---

---


---

---

---

---

---



## Session Bean Lifetime

- A key difference between session beans and other bean types is the scope of their lives.
- A session bean instance is a relatively short-lived object. It has roughly the lifetime equivalent of a *session or of the client code that is calling the session bean*.
- Session bean instances are not shared between multiple clients.
- For example, if the client code contacted a session bean to perform order entry logic, the EJB container is responsible for creating an instance of that session bean component.
- When the client later disconnects, the application server may destroy the session bean instance.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Session Bean Lifetime

- A client's session duration could be as long as a browser window is open, perhaps connecting to an e-commerce site with deployed session beans.
- It could also be as long as your Java applet is running, as long as a standalone application is open, or as long as another bean is using your bean.
- The length of the client's *session generally determines how long a session bean is in use—that is where the term session bean originated*.
- *The EJB container* is empowered to destroy session beans if clients time out. If your client code is using your beans for 10 minutes, your session beans might live for minutes or hours, but probably not weeks, months, or years.
- Typically session beans do not survive application server crashes, nor do they survive machine crashes.
- They are in-memory objects that live and die with their surrounding environments.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Session Bean Lifetime

- In contrast, entity beans can live for months or even years because entity beans are *persistent objects*. *Entity beans are part of a durable, permanent storage*, such as a database. Entity beans can be constructed in memory from database data, and they can survive for long periods of time.
- Session beans are *non persistent*. *This means that session beans are not saved to permanent storage*, whereas entity beans are.
- Note that session beans *can* perform database operations, but the session bean *itself is not a persistent object*.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Session Bean Subtypes

- All enterprise beans hold *conversations with clients at some level*.
- A *conversation* is an interaction between a client and a bean, and it is composed of a number of method calls between the client and the bean.
- A conversation spans a business process for the client, such as configuring a frame-relay switch, purchasing goods over the Internet, or entering information about a new customer.
- A *session bean* represents work being performed for client code that is calling it. Session beans are business process objects that implement business logic, business rules, algorithms, and workflow. For example, a session bean can perform price quoting, order entry, video compression, banking transactions, stock trades, database operations, complex calculations, and more. They are reusable components that contain logic for business processes.
- The two subtypes of session beans are *stateful session beans* and *stateless session beans*. Each is used to model different types of these conversations.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Stateful Session Beans

- Some business processes are naturally drawn-out conversations over several requests. An example is an e-commerce Web store. As a user peruses an online e-commerce Web site, the user can add products to the online shopping cart.
- Each time the user adds a product, we perform another request. The consequence of such a business process is that the components must track the user's state (such as a shopping cart state) from request to request.
- Another example of a drawn-out business process is a banking application. You may have code representing a bank teller who deals with a particular client for a long period of time. That teller may perform a number of banking transactions on behalf of the client, such as checking the account balance, depositing funds, and making a withdrawal.
- A *stateful session bean* is a bean that is designed to service business processes that span multiple method requests or transactions. To accomplish this, stateful session beans retain state on behalf of an individual client. If a stateful session bean's state is changed during a method invocation, that same state will be available to that same client upon the following invocation.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Stateless Session Beans

- Some business processes naturally lend themselves to a single request conversation. A single request business process is one that does not require state to be maintained across method invocations.
- A *stateless session bean* is a bean that holds conversations that span a single method call. They are stateless because they do not hold multimethod conversations with their clients. After each method call, the container may choose to destroy a stateless session bean, or recreate it, clearing itself out of all information pertaining to past invocations. It also may choose to keep your instance around, perhaps reusing it for all clients who want to use the same session bean class. The exact algorithm is container specific.
- The takeaway point is this: Expect your bean to forget everything after each method call, and thus retain no conversational state from method to method. If your bean happens to hang around longer, then great—but that's your container's decision, and you shouldn't rely on it.
- For a stateless session bean to be useful to a client, the client must pass all client data that the bean needs as parameters to business logic methods. Alternatively, the bean can retrieve the data it needs from an external source, such as a database.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Stateless Session Bean

- *Stateless really means no conversational state. Stateless session beans can contain state that is not specific to any one client, such as a database connection factory that all clients would use. You can keep this around in a private variable.*
- An example of a stateless session bean is a high-performance engine that solves complex mathematical operations on a given input, such as compression of audio or video data. The client could pass in a buffer of uncompressed data, as well as a compression factor. The bean returns a compressed buffer and is then available to service a different client. The business process spanned one method request. The bean does not retain any state from previous requests.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---


---

---

---

---

---



## Required Methods for Session Bean

METHOD	DESCRIPTION	TYPICAL IMPLEMENTATION (STATELESS SESSION BEANS)	TYPICAL IMPLEMENTATION (STATEFULL SESSION BEANS)
<code>setSessionContext (SessionContext ctx)</code>	Associates your bean with a session context. Your bean can query the context about its current transactional state, its current security state, and more.	Store the context away in a member variable so the context can be queried later.	Store the context away in a member variable so the context can be queried later.
<code>ejbCreate()</code>	Initializes your session.	Perform any initialization your bean needs, such as setting member variables to the argument values passed in. Note: You can define several <code>ejbCreate()</code> methods, and each can take different arguments. You must provide at least one <code>ejbCreate()</code> method in your session bean.	Perform any initialization your bean needs, such as setting member variables to the argument values passed in. Note: You can define only a single empty <code>ejbCreate()</code> method with no parameters. If it had parameters, the bean would never remember what it initialized itself to upon subsequent calls, since it is stateless!
<code>ejbPassivate()</code>	Called immediately before your bean is passivated (swapped out to disk because there are too many instantiated beans).	Release any resources your bean may be holding.	Unused because there is no conversational state; leave empty.
<code>ejbActivate()</code>	Called immediately before your bean is activated (swapped in from disk because a client needs your bean).	Acquire any resources your bean needs, such as those released during <code>ejbPassivate()</code> .	Unused because there is no conversational state; leave empty.
<code>ejbRemove()</code>	Called by the container immediately before your bean is removed from memory.	Prepare your bean for destruction free all resources you may have allocated.	Prepare your bean for destruction. Free all resources you may have allocated.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---


---

---

---

---

---



## Session Bean Lifetime

- A key difference between session beans and other bean types is the scope of their lives. A session bean instance is a relatively short-lived object. It has roughly the lifetime equivalent of a *session or of the client code that is calling the session bean*. Session bean instances are not shared between multiple clients.
- For example, if the client code contacted a session bean to perform order entry logic, the EJB container is responsible for creating an instance of that session bean component. When the client later disconnects, the application server may destroy the session bean instance.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---

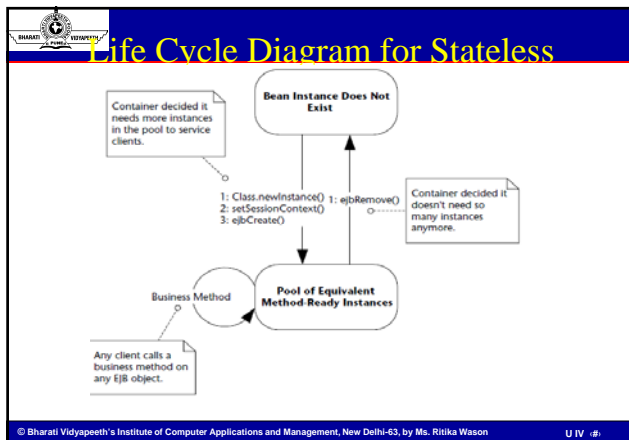
---

---

---

---

---




---

---

---

---

---

---

---

---

### Stateless Session Bean Life Cycle

- A stateless session bean has its life cycle inside the container.
- The client does not call the methods on the bean, since the client never accesses a bean directly. (The client always goes through the container.) The container (that is, the home object and EJB objects) calls methods on our bean.

- At first, the bean instance does not exist. Perhaps the application server has just started up.
- The container decides it wants to instantiate a new bean. When does the container decide it wants to instantiate a new bean? It depends on the container's policy for *pooling beans*. The container may decide to instantiate 10 beans all at once when the application server first starts because you told the container to do so using the vendor-specific files that you ship with your bean. Each of those beans are equivalent (because they are stateless) and they can be reused for many different clients.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

### Stateless Session Bean Life Cycle

- The container instantiates your bean. The container calls `Class.newInstance()` on your session bean class, which is the dynamic equivalent of calling `new HelloBean()`. The container does this so that the container is not hard-coded to any specific bean name; the container is generic and works with any bean. This action calls your bean's default constructor, which can do any necessary initialization.
- The container calls `setSessionContext()`. This associates you with a context object, which enables you to make callbacks to the container.
- The container calls `ejbCreate()`. This initializes your bean. Note that because stateless session beans' `ejbCreate()` methods take no parameters, clients never supply any critical information that bean instances need to start up. EJB containers can exploit this and pre-create instances of your stateless session beans. In general when a client creates or destroys a bean using the home object, that action might not necessarily correspond with literally creating or destroying in-memory bean objects, because the EJB container controls their life cycles to allow for pooling between heterogeneous clients.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

### Stateless Session Bean Life Cycle

6. **The container can call business methods on your bean.** The container can call as many business methods as it wants to call. Each business method could originate from a completely different client because all bean instances are treated exactly the same. All stateless session beans think they are in the same state after a method call; they are effectively unaware that previous method calls happened. Therefore the container can dynamically reassign beans to client requests at the *per-method level*. A different stateless session bean can service *each method call* from a client.

7. **Finally, the container calls *ejbRemove()*.** When the container is about to remove your session bean instance, it calls your bean's *ejbRemove()* callback method. *ejbRemove()* is a *clean-up method*, alerting your bean that it is about to be destroyed and allowing it to end its life gracefully. *ejbRemove()* is a *required method of all beans*, and it takes *no parameters*. Therefore there is only one *ejbRemove()* method per bean. This is in stark contrast to *ejbCreate()*, which has many forms. This makes perfect sense: Why should a destructive method be personalized for each client? Your implementation of *ejbRemove()* should prepare your bean for destruction. This means you need to free all resources you may have

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

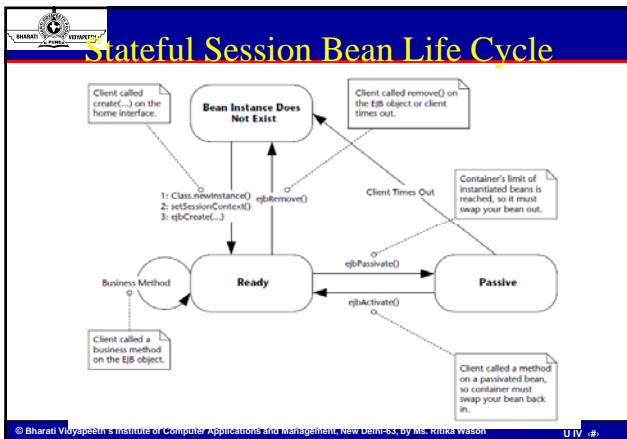
---

---

---

---

---




---

---

---

---

---

---

---

---

### Stateful Session Bean Life Cycle

- The life cycle for stateful session beans is similar to that of stateless session beans. The big differences are as follows:
- There is no pool of equivalent instances because each instance contains state.
- There are transitions for passivating and activating conversational state.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Entity Beans Introduction

- One of the key benefits of EJB is that it gives you the power to create *entity beans*. *Entity beans are persistent objects that you place in permanent storage.*
- This means you can model your business's fundamental, underlying data as entity beans.
- A popular way to store Java objects is to use a traditional relational database, such as Oracle, Microsoft SQL Server, or MySQL. Rather than serialize each object, we could decompose each object into its constituent parts and store each part separately.
- For example, for a bank account object, the bank account number could be stored in one relational database field and the bank account balance in another field. When you save your Java objects, you would use JDBC to *map the object data into a relational database*. When you want to
- load your objects from the database, you would instantiate an object from that class, read the data in from the database, and then populate that object instance's fields with the relational data read in.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Object-Relational Mapping

- This mapping of objects to relational databases is a technology called *object relational mapping*. *It is the act of converting and unconverted in-memory objects to relational data.*
- An object-relational (O/R) mapper may map your objects to any kind of relational database schema. For example, a simple object-relational mapping engine might map a Java class to a SQL table definition.
- An instance of that class would map to a row in that table, while fields in that instance would map to individual cells in that row.
- Object-relational mapping is a much more sophisticated mechanism of persisting objects than the simple object serialization offered by the Java language. By decomposing your Java objects as relational data, you can issue arbitrary queries for information.
- Mapping objects to relational data can be done in two ways. You can either handcraft this mapping in your code or use an object-relational mapping product, such as Oracle TopLink, or open source tools, such as Hibernate, to automate or facilitate this mapping. These tools have become increasingly popular.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## What is an Entity Bean?

- In any sophisticated, object-oriented multitier deployment, we can draw a clear distinction between two different kinds of components deployed.
- Application logic components. These components are method providers that perform common tasks.** Their tasks might include the following:
  - Computing the price of an order
  - Billing a customer's credit card
  - Computing the inverse of a matrix
- Note that these components represent actions (they're verbs). They are well suited to handling business processes. Session beans model these application logic components very well.
- They often contain interesting algorithms and logic to perform application tasks. Session beans represent work being performed for a user. They represent the user session, which includes any workflow logic.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## What is an Entity Bean?

- **Persistent data components. These are objects (perhaps written in Java)** that know how to render themselves into persistent storage. They use some persistence mechanism, such as serialization, O/R mapping to a relational database, or an object database. These kinds of objects represent *data—simple or complex information that you'd like saved*.  
*Examples here include:*
  - Bank account information, such as account number and balance
  - Human resources data, such as names, departments, and salaries of employees
  - Lead tracking information, such as names, addresses, and phone numbers of prospective customers that you want to keep track of over time

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## What is an Entity Bean?

- Note that these components represent people, places, and things (they're nouns). They are well suited to handling business data.
- The big difference between session beans and entity beans is that entity beans have an identity and client-visible state, and that their lifetime may be completely independent of the client application's lifetime. For entity beans, having an identity means that different entity beans can be distinguished by comparing their identities. It also means that clients can refer to individual entity bean instances by using that identity, pass handles to other applications, and actually share common entities with other clients. All this is not possible with session beans.
- It is handy to treat data as objects because they can be easily handled and managed and because they are represented in a compact manner. We can group related data in a unified object. We associate some simple methods with that data, such as compression or other data-related activities. We can also use implicit middleware services from an application server, such as relationships, transactions, network accessibility, and security. We can also cache that data for performance.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## What is an Entity Bean?

- Entity beans are these persistent data components. Entity beans are enterprise beans that know how to persist themselves permanently to a durable storage, such as a database or legacy system.
- They are physical, storable parts of an enterprise. Entity beans store data as fields, such as bank account numbers and bank account balances. They also have methods associated with them, such as `getBankAccountNumber()`.
- In some ways, entity beans are analogous to serializable Java objects. Serializable objects can be rendered into a bit-blob and then saved into a persistent store; entity beans can persist themselves in many ways, including Java serialization, O/R mapping, or even an object database persistence. Nothing in the EJB 2.x specification dictates any particular persistence mechanism, although O/R mappings are the most frequently used.
- Entity beans are different from session beans. Session beans model a process or workflow (actions that are started by the user and that go away when the user goes away). Entity beans, on the other hand, contain core business data—product information, bank accounts, orders, lead tracking information, customer information and more.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---


---

---

---

---





## What is an Entity Bean?

- An entity bean does not perform complex tasks or workflow logic, such as billing a customer. Rather, an entity bean *is the* customer itself. Entity beans represent persistent state objects (things that don't go away when the user goes away).
- For example, you might want to read a bank account data into an entity bean instance, thus loading the stored database information into the in-memory entity bean instance's fields. You can then play with the Java object and modify its representation in memory because you're working with convenient Java objects, rather than bunches of database records. You can increase the bank account balance in-memory, thus updating the entity bean's in-memory bank account balance field. Then you can save the Java object, pushing the data back into the underlying store. This would effectively deposit money into the bank account.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Conclusion

- In summary, you should think of an entity bean instance as the following:
- An in-memory Java representation of persistent data that knows how to read itself from storage and populate its fields with the stored data
- An object that can then be modified in-memory to change the values of data
- Persistable, so that it can be saved back into storage again, thus updating the database data

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Features of Entity Beans

- Entity Beans Survive Failures**
- Entity beans are long lasting. They survive critical failures, such as application servers crashing, or even databases crashing. This is because entity beans are just representations of data in a permanent, fault-tolerant, underlying storage.
- If a machine crashes, the entity bean can be reconstructed in memory. All we need to do is read the data back in from the permanent database and instantiate an entity bean Java object instance with fields that contain the data read in from the database.
- This is a huge difference between session and entity beans. Entity beans have a much longer life cycle than a client's session, perhaps years long, depending on how long the data sits in the database. In fact, the database records representing an object could have existed before the company even decided to go with a Java-based solution, because a database structure can be language independent.
- This makes sense—you definitely would want your bank account to last for a few years, regardless of technology changes in your bank.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



### Entity Bean Instances Are a View into a Database

- When you load entity bean data into an in-memory entity bean instance, you read in the data stored in a database so that you can manipulate the data within a Java Virtual Machine. However, *you should think of the in-memory object and the database itself as one and the same. This means if you update the in-memory entity bean instance, the database should automatically be updated as well. You should not think of the entity bean as a separate version of the data in the database.*
- The in-memory entity bean is simply a *view or lens into the database*.
- Of course, in reality there are multiple physical copies of the same data: the in-memory entity bean instance and the entity bean data itself stored in the database. Therefore, there must be a mechanism to transfer information back and forth between the Java object and the database.
- This data transfer is accomplished with two special methods that your entity bean class must implement, called *ejbLoad()* and *ejbStore()*.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



### General Entity Bean Instances May Represent the Same Underlying Data

- If many threads of execution want to access the same database data simultaneously. Ex: In e-commerce, many different client browsers may be simultaneously interacting with a catalog of products.
- To facilitate many clients accessing the same data, we need to design a high performance access system to our entity beans. One possibility is to allow many clients to share the same entity bean instance; that way, an entity bean could service many client requests simultaneously. While this is an interesting idea, it is not very appropriate for EJB, for two reasons.
- First, if we'd like an entity bean instance to service many concurrent clients, we'd need to make that instance thread-safe. Writing thread-safe code is difficult and error-prone. Remember that the EJB value proposition is rapid application development.
- Second, having multiple threads of execution makes transactions almost impossible to control by the underlying transaction system.
- For these reasons, EJB dictates that only a single thread can ever be running within a bean instance. With session beans and message-driven beans, as well as

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



### Entity Bean Instances Can Be Pooled

- As clients connect and disconnect, you could create and destroy beans as necessary to service those clients. Unfortunately this is not a scalable way to build an application server.
- Creation and destruction of objects is expensive, especially if client requests come frequently. How can we save on this overhead?
- One thing to remember is that an entity bean class describes the fields and rules for your entity bean, but it does not dictate any specific data. For example, an entity bean class may specify that all bank accounts have the following fields:
  - The name of the bank account owner
  - An account ID
  - An available balance
- That bean class can then represent any distinct instance of database data, such as a particular bank account record. The class itself, though, is not specific to any particular bank account.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## There Are Two Ways to Persist Entity Beans

- Since entity beans map to storage, someone needs to write the database access code.
- A *bean-managed persistent entity bean* is an entity bean that must be persisted by hand.
- You handle the persistent operations yourself—including saving, loading, and finding data—within the entity bean.
- Therefore, you must write to a persistence API, such as JDBC. For example, with a relational database, your entity bean could perform a SQL INSERT statement via JDBC to stick some data into a relational database.
- You could also perform an SQL DELETE statement via JDBC to remove data from the underlying store.
- EJB offers an alternative to bean-managed persistence: You can have your EJB container perform your persistence for you. This is called *container-managed persistence*.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Creation and Removal of Entity Beans

- As we mentioned earlier, entity beans are a view into a database, and you should think of an entity bean instance and the underlying database as one and the same (they are routinely synchronized).
- Because they are one and the same, the initialization of an entity bean instance should entail initialization of database data.
- Thus, when an entity bean is initialized in memory during *ejbCreate()*, it makes sense to create some data in an underlying database that correlates with the in-memory instance. That is exactly what happens with entity beans. When a bean-managed persistent entity bean's *ejbCreate()* method is called, the *ejbCreate()* method is responsible for creating database data.
- Similarly, when a bean-managed persistent entity bean's *ejbRemove()* method is called, the *ejbRemove()* method is responsible for removing database data.
- If container-managed persistence is used, the container will modify the database for you, and you can leave these methods empty of data access logic.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Entity Beans Can Be Found

- Because entity bean data is uniquely identified in an underlying storage, entity beans can also be *found rather than created*.
- *Finding an entity bean* is analogous to performing a SELECT statement in SQL. With a SELECT statement, you're searching for data from a relational database store.
- When you find an entity bean, you're searching a persistent store for some entity bean data. This differs from session beans because session beans cannot be found: They are not permanent objects, and they live and die with the client's session.
- You can define many ways to find an entity bean. You list these ways as methods in your entity bean home interface.
- These are called *finder methods*. Your home interface exposes finder methods in addition to methods for creating and destroying entity beans.
- This is the one big difference between an entity bean's home interface and other types of beans; the other bean types do not have finder methods.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

**You Can Modify Entity Bean Data without Using EJB**

- Usually you will create, destroy, and find entity bean data by using the entity bean's home object. But you can interact with entity beans another way, too: by directly modifying the underlying database where the bean data is stored.
- For example, if your entity bean instances are being mapped to a relational database, you can simply delete the rows of the database corresponding to an entity bean instance.
- You can also create new entity bean data and modify existing data by directly touching the database. This may be necessary if you have an investment in an existing system that touches a database directly

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

**Introduction**

- All enterprise beans have a *context object that identifies* the environment of the bean.
- These context objects contain environment information that the EJB container sets. Your beans can access the context to retrieve all sorts of information, such as transaction and security information.
- For entity beans, the interface is *javax.ejb.EntityContext*.
- The *javax.ejb.EJBContext* methods are:
- ```
public interface javax.ejb.EJBContext {
    public javax.ejb.EJBHome getEJBHome();
    public javax.ejb.EJBLocalHome getEJBLocalHome();
    public java.security.Principal getCallerPrincipal();
    public boolean isCallerInRole(java.lang.String);
    public void setRollbackOnly();    public boolean getRollbackOnly(); }
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

**Entity Context**

- Entity contexts add the following methods on top of the generic EJB context.
- ```
public interface javax.ejb.EntityContext extends javax.ejb.EJBContext {
    public javax.ejb.EJBLocalObject getEJBLocalObject();
    public javax.ejb.EJBObject getEJBObject();
    public java.lang.Object getPrimaryKey();
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



### getEJBLocalObject() / getEJBObject()

- Call the *getEJBObject()* method to retrieve the current, client-specific EJB object that is associated with the entity bean.
- Remember that clients invoke on EJB objects, not on entity beans directly.
- Therefore, you can use the returned EJB object as a way to pass a reference to yourself, simulating the *this* argument in Java.
- *getEJBLocalObject()* is the same, except it gets the more optimized EJB local object.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



### getPrimaryKey()

- *getPrimaryKey()* retrieves the primary key that is currently associated with this entity bean instance.
- Primary keys uniquely identify an entity bean. When an entity bean is persisted in storage, the primary key can be used to uniquely retrieve the entity bean because no two entity bean database data instances can ever have the same primary key.
- Why would you want to call *getPrimaryKey()*? You call it whenever you want to figure out with which database data your instance is associated.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



### Introduction to Message-Driven Beans

- While RMI-IIOP may be useful in many scenarios, several other areas are challenging for RMI-IIOP. For Ex:
- **Performance.** A typical RMI-IIOP client must wait (or block) while the server performs its processing. Only when the server completes its work does the client receive a return result, which enables it to continue processing.
- **Reliability.** When an RMI-IIOP client calls the server, the latter has to be running. If the server crashes or the network crashes, the client cannot perform its intended operation.
- **Support for multiple senders and receivers.** RMI-IIOP limits you to a single client talking to a single server at any given time. There is no built-in functionality for multiple clients to broadcast events to multiple servers.
- **Integration with other MOM systems.** When you have enterprise systems that communicate through messages, message-driven beans coupled along with J2EE connectors can integrate these different message-driven enterprise systems.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

**Message-Driven Beans**

- *Messaging is an alternative to remote method invocations.*
- The idea behind messaging is that a *middelman sits between the client and the server*. (A layer of indirection solves every problem in computer science.) This middleman receives messages from one or more *message producers and broadcasts* those messages to one or more *message consumers*.
- *Because of this middleman*, the producer can send a message and then continue processing. He can optionally be notified of the response later when the consumer finishes. This is called *asynchronous programming*.

Remote Method Invocations:

Messaging:

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

**Messaging (Contd.)**

- Messaging addresses the three previous concerns with RMI-IIOP as follows:
- **Non-blocking request processing.** A messaging client does not need to block when executing a request. As an example, when you purchase a book using the Amazon.com one-click order functionality, you can continue browsing the site without waiting to see if your credit card authorizes. Unless something goes wrong, Amazon.com sends you a confirmation e-mail afterwards. This type of fire-and-forget system could easily be coded using messaging. When the user clicks to buy the book, a message is sent that results in credit card processing later. The user can continue to browse.
- **Reliability.** If your message-oriented middleware supports *guaranteed delivery*, you can send a message and know for sure that it will reach its destination, even if the consumer is not available. You send the message to the MOM middleman, and that middleman routes the message to the consumer when he comes back alive again. With RMI-IIOP, this is not possible because there is no middleman. If the server is down, an exception is thrown.
- **Support for multiple senders and receivers.** Most message-oriented middleware products can accept messages from many senders and broadcast them to many receivers. This enables you to have multinary communications.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

**Messaging (Contd.)**

- Note that messaging also has many disadvantages. Performance, for one, can be slower in many circumstances due to the overhead of having the messaging middleman.
- *Message-oriented middleware (MOM)* is a term used to refer to any infrastructure that supports messaging. A variety of products are considered to have a MOM-based architecture. Examples include Tibco Rendezvous, IBM WebSphere MQ, BEA Tuxedo/Q, Sun Java System Messaging Server, Microsoft MSMQ, Sonic Software SonicMQ, and FioranoMQ.
- These products can give you a whole host of value-added services, such as guaranteed message delivery, fault tolerance, load balancing of destinations, subscriber throttling of message consumption, inactive subscribers, support for SOAP over JMS, and much, much more. By allowing the MOM server to address these infrastructure issues, you can focus on the business task at hand.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Java Message Service (JMS)

- Over the years, MOM systems have evolved in a proprietary way. Each product has its own API, which creates vendor lock-in because code is not portable to other messaging systems. It also hurts developers, because they need to relearn each messaging product's proprietary API.
- The Java Message Service (JMS) is a messaging standard, designed to eliminate many of the disadvantages that MOM-based products faced over past years. JMS has two parts: an API, for which you write code to send and receive messages, and a Service Provider Interface (SPI) where you plug in JMS providers. A JMS provider knows how to talk to a specific MOM implementation.*
- The JMS promise is that you can learn the JMS API once and reuse your messaging code with different plug-and-play MOM implementations (an idea similar to the other J2EE APIs, such as JNDI or JDBC).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---


---

---

---

---

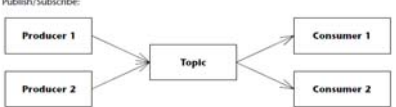
---



## Messaging Domains

- When you perform messaging, you need to choose a *domain*. A domain is a fancy word for style of messaging. The types of domains are:
- Publish/subscribe (pub/sub).** Publish/subscribe is analogous to watching television. Many TV stations broadcast their signals, and many people listen to those broadcasts. Thus, with publish/subscribe, you can have *many message producers talking to many message consumers*.
- In this sense, the pub/sub domain is an implementation of a distributed event-driven processing model. Subscribers (listeners) register their interest in a particular event *topic*. Publishers (event sources) create messages (events) that are distributed to all of the subscribers (listeners). Producers aren't hard-coded to know the specific consumers interested in receiving its messages; rather, the MOM system maintains the subscriber list.

Publish/Subscribe:



```

graph LR
    P1[Producer 1] --> T[Topic]
    P2[Producer 2] --> T
    T --> C1[Consumer 1]
    T --> C2[Consumer 2]

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---


---

---

---

---

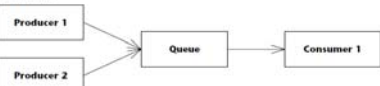
---



## Messaging Domains

- Point-to-point (P/P).** Point-to-point is analogous to calling a toll-free number and leaving a voice mail. Some person will listen to your voice mail and then delete it. Thus, with point-to-point, you can have only a single consumer for *each message*. Multiple consumers can grab messages off the queue, but any given message is consumed exactly once.
- In this sense, point-to-point is a degenerate case of publish/subscribe. Multiple producers can send messages to the queue, but each message is delivered only to a single consumer. The way this works is that publishers send messages directly to the consumer or to a centralized *queue*.
- Messages are typically distributed off the queue in a first-in, first-out (FIFO) order, but this isn't assured.

Point-to-Point:



```

graph LR
    P1[Producer 1] --> Q[Queue]
    P2[Producer 2] --> Q
    Q --> C1[Consumer 1]

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---


---

---

---

---

---



## The JMS API

- The JMS API is more involved than RMI-IIOP. You need to become familiar with many different interfaces to get going. Despite the complexities involved in working with each of these interfaces, low-level topology issues, such as networking protocol, message format and structure, and server location, are mostly abstracted from the developer.
- The JMS programming model is explained as follows:
  - 1. Locate the JMS Provider ConnectionFactory instance.** *You first need to get access to the JMS provider of the particular MOM product you're using. For this, you need to establish a connection using a **ConnectionFactory** instance. You can get hold of **ConnectionFactory** by looking it up in JNDI. An administrator will typically create and configure the **ConnectionFactory** for the JMS client's use.*
  - 2. Create a JMS connection.** *A **JMS Connection** is an active connection to the JMS provider, managing the low-level network communications (similar to a JDBC connection). You use the **ConnectionFactory** to get a **Connection**. If you're in a large deployment, this connection might be load-balanced across a group of machines.*

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## The JMS API

- 3. Create a JMS session.** *A **JMS Session** is a helper object that you use when sending and receiving messages. It serves as a factory for message consumers and producers, and also enables you to encapsulate your messages in transactions. You use the **Connection** to get a **Session**.*
- 4. Locate the JMS destination.** *A **JMS Destination** is the channel to which you're sending or from which you're receiving messages. Locating the right destination is analogous to tuning into the right channel when watching television or answering the correct phone, so that you get the messages you desire. Your deployer typically sets up the destination in advance by using your JMS provider's tools, so that the destination is permanently set up. Your code looks up that destination using JNDI. This enables your programs to use the destination over and over again at runtime.*

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## The JMS API

- 5. Create a JMS producer or a JMS consumer.** *If you want to send messages, you need to call a JMS object to pass it your messages. This object is called **producer**. To receive messages, you call a **JMS** object and ask it for a message. This object is called the **Consumer** object. You use the **Session** and **Destination** to get a hold of a **Producer** or a **Consumer** object.*
- 6. Send or receive your message.** *If you're producing, you first need to put your message together. There are many different types of messages, such as text, bytes, streams, objects, and maps. After you instantiate your message, you send it using the **Producer** object. If, on the other hand, you're receiving messages, you first receive a message using the **Consumer** object, and then crack it open (depending on the message type) and see what is in it.*

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

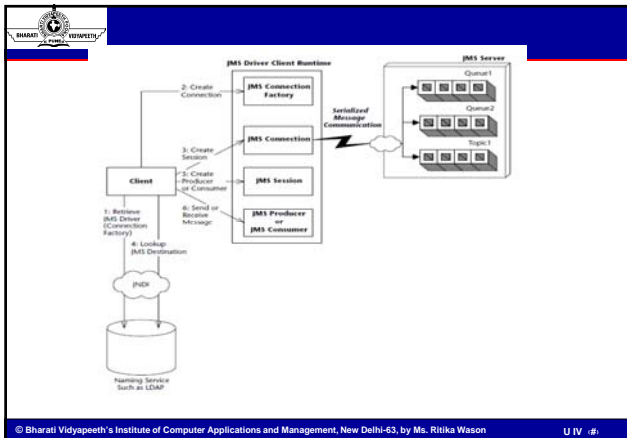
---

---

---

---






```
import javax.naming.*; import javax.jms.*; import java.util.*;
public class Client { public static void main (String[] args) throws Exception {
Context ctx = new InitialContext(System.getProperties()); // Initialize JNDI
// 1: Lookup ConnectionFactory via JNDI
TopicConnectionFactory factory = (TopicConnectionFactory)
ctx.lookup("TopicConnectionFactory");
// 2: Use ConnectionFactory to create JMS connection
TopicConnection connection = factory.createTopicConnection();
// 3: Use Connection to create session
TopicSession session = connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
// 4: Lookup Destination (topic) via JNDI
Topic topic = (Topic) ctx.lookup("testtopic");
TopicPublisher publisher = session.createPublisher(topic); // 5: Create a Message Producer
// 6: Create a text message, and publish it
TextMessage msg = session.createTextMessage();
msg.setText("This is a test message.");
publisher.publish(msg);}}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

## Integrating JMS with EJB

- JMS-EJB integration is a compelling idea. It would allow EJB components to benefit from the value proposition of messaging, such as non-blocking clients and multinary communications.
- To understand the motivations behind introducing a completely different type of bean to consume messages in an EJB application, let us contemplate for a moment what other approaches could we have taken and whether they would have worked:
  - i) **Using a Java object that receives JMS messages to call EJB components.**
- Rather than coming up with a whole new type of bean, the Java community could have promoted the idea of a Java object that can receive messages and in turn call the appropriate EJB components, such as session beans and entity beans. The problems with this approach are as follows:
  - You'd need to write special code to register yourself as a listener for JMS messages. This is a decent amount of code (as we demonstrated previously).
  - To increase the throughput of message consumption, you would have to write the multithreading logic such that you can listen to the messages on multiple threads. However, writing multithreaded applications is not a trivial task for a business application developer.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV



## Integrating JMS with EJB

- Your Java object that listens to messages would need some way of starting up, since it wrapped your other EJB components. If the class ran within the container, you would need to use an EJB server-specific *startup class to activate your Java object when the EJB server came up*. This is not portable because EJB specification does not define a standard way of activating a given logic.
- Your plain Java object wouldn't receive any services from an EJB container, such as automatic life cycle management, clustering, pooling, and transactions. You would need to hard-code this yourself, which is difficult and error-prone.
- You would need to hard-code the JMS destination name in your Java object. This hurts reusability, because you couldn't reuse that Java object with other destinations. If you get the destination information from a disk (such as with property files), this is a bit clunky.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Integrating JMS with EJB

ii) **Reuse an existing type of EJB component somehow to receive JMS messages. Another option could have been to shoehorn session beans or entity beans into receiving JMS messages.**  
Problems with this approach include:

- Threading. If a message arrives for a bean while it's processing other requests, how can it take that message, given that EJB does not allow components to be multithreaded?
- **Life cycle management. If a JMS message arrives and there are no beans, how does the container know to create a bean?**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## What Is a Message-Driven Bean?

- A *message-driven bean, introduced in EJB 2.0, is a special EJB component that can receive JMS messages as well as other types of messages.*
- A message-driven bean is invoked by the container upon arrival of a message at the destination or endpoint that is serviced by message-driven bean. A message-driven bean is decoupled from any clients that send messages to it.
- A *client cannot access a message-driven bean through a component interface. You will have to use message provider-specific API, such as JMS, to send messages from clients, which in turn would be received by the message-driven beans*

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

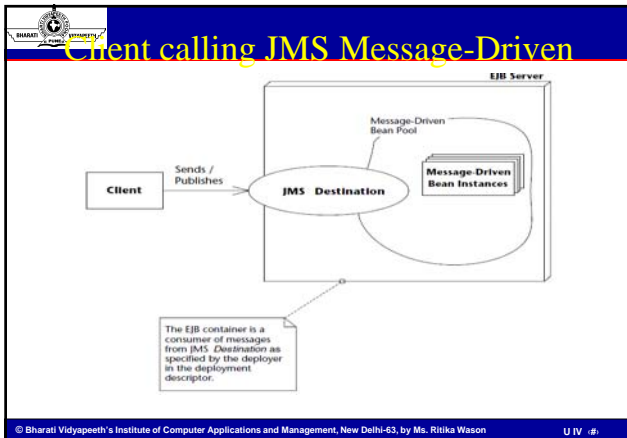
---

---

---

---

---




---

---

---

---

---

---

---

---

### Characteristics of Message-Driven Beans

- A message-driven bean does not have a home interface, local home interface, remote interface, or a local interface. You do not call message-driven beans using an object-oriented remote method invocation interface. The reason is that message-driven beans process messages, and those messages can come from any messaging client, such as an MQSeries client, an MSMQ client, or a J2EE client using the JMS API. Message-driven beans, along with appropriate message providers, can thus consume any valid message.
- Message-driven beans have weakly typed business method. Message driven beans are merely receiving messages from a destination or a resource adapter and they do not know anything about what's inside the messages. The listener interface implemented by message-driven beans typically has a method (or methods) called by an EJB container upon arrival of a message or by the resource adapter (via application server). JMS message listener interface, `javax.jms.MessageListener` has only one method called `onMessage()`. This method accepts a `JMS Message`, which could represent anything—a `BytesMessage`, `ObjectMessage`, `TextMessage`, `StreamMessage`, or `MapMessage`. In a typical implementation of `onMessage()`, the message is cracked open at runtime and its contents are examined, perhaps with the help of a bunch of `if` statements.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---

---

---

---

---

---

### Characteristics of Message-Driven Beans

- Message-driven bean listener method(s) might not have any return values. Although EJB 2.1 specification does not restrict a message driven bean listener method from returning a value to the client, certain messaging types might not be suitable for this. For example, consider the listener interface of a messaging type that supports asynchronous messaging, such as JMS. In this case, due to the asynchronous interaction between message producers and consumers, the message producers don't wait for your message-driven bean to respond. As a result, it doesn't make sense for the `onMessage()` listener method on the `javax.jms.MessageListener` interface to return value. The good news is that using several design patterns, it is possible to send a response to an asynchronous message producer. We discuss this later in this chapter.
- Message-driven beans might not send exceptions back to clients. Although, EJB 2.1 does not restrict message-driven bean listener interface methods from throwing application exceptions, certain messaging types might not be able to throw these exceptions to the clients. Again consider the example of a listener interface of a messaging type that supports asynchronous messaging, such as JMS. In this case, message producers won't wait for your message-driven bean to send a response because the interaction is asynchronous. Therefore clients can't receive any exceptions.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Characteristics of Message-Driven Beans

- **Message-driven beans are stateless. Message-driven beans hold no conversational state.** It would be impossible to spread messages across a cluster of message-driven beans if a message-driven bean held state. In this sense, they are similar to stateless session beans because the container can similarly treat each message-driven bean instance as equivalent to all other instances. All instances are anonymous and do not have an identity that is visible to a client. Thus, multiple instances of the bean can process multiple messages from a JMS destination or a resource adapter concurrently.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Developing Message-Driven Beans

- **The Semantics**
- JMS message-driven beans are classes that implement two interfaces:
- *javax.jms.MessageListener and javax.ejb.MessageDrivenBean.*  
Additionally, every JMS message-driven bean implementation class must provide an *ejbCreate()* method that returns *void* and accepts *no arguments*. Here is what the *javax.jms.MessageListener* interface looks like:
- ```
public interface javax.jms.MessageListener { public void
onMessage(Message message); }
```
- Here is what the *javax.ejb.MessageDrivenBean* interface looks like:
- ```
public interface javax.ejb.MessageDrivenBean extends
javax.ejb.EnterpriseBean { public void ejbRemove() throws EJBException;
public void setMessageDrivenContext(MessageDrivenContext ctx) throws
EJBException;}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Methods implemented in JMS Message Driven Beans

- **onMessage(Message)** This method is invoked for each message that is consumed by the bean. The input parameter of the method is the incoming JMS message that is being consumed. The container is responsible for serializing messages to a single message-driven bean. A single message-driven bean can process only one message at a time. It is the container's responsibility to provide concurrent message consumption by pooling multiple message-driven bean instances. A single instance cannot concurrently process messages, but a container can. This method does not have to be coded for reentrancy and should not have any thread synchronization code contained within.
- **ejbCreate()** This method is invoked when a message-driven bean is first created and added to a pool. Application server vendors can implement an arbitrary algorithm that decides when to add message-driven bean instances from the pool. Beans are typically added to the pool when the component is first deployed or when the load of messages to be delivered increases. Bean developers should initialize variables and references to resources needed by the bean, such as other EJBs or database connections. Bean developers should initialize only references to resources that are needed for every message that is consumed by the bean, as opposed to gaining access and releasing the resource every time a message is consumed.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U IV

---

---

---


---

---

---

---

---



## Methods implemented in Message Driven Beans

- ejbRemove()* This method is invoked when a message-driven bean is being removed from a pool. Application server vendors can implement an arbitrary algorithm that decides when to remove message driven bean instances from the pool. Beans are typically removed from the pool when the component is being undeployed or when a load of messages to be delivered by an application server decreases, thereby requiring the application server to remove idle instances to free up system resources. Bean developers should use this method to clean up any dangling resources that are used by the bean.
- setMessageDrivenContext* This method is called as part of the event (*MessageDrivenContext*) transition that a message-driven bean goes through when it is being added to a pool. This method is called before the *ejbCreate()* method is invoked. The input parameter for this method is an instance of the *MessageDrivenContext* interface. The input parameter gives the bean access to information about the environment that it executes within. The only methods on the *MessageDrivenContext* that are accessible by the message-driven bean are transaction related methods. Other methods, such as *getCallerPrincipal()*, cannot be invoked in this method because message-driven beans do not have home, local home, remote, or local interface, and do not have a visible client security context.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---


---

---

---

---

---



## Methods implemented in Message Driven Beans

```

package examples; import javax.ejb.*; import javax.jms.*;
/** Sample JMS Message-Driven Bean */
public class LogBean implements MessageDrivenBean, MessageListener {
    protected MessageDrivenContext ctx;
    /** Associates this Bean instance with a particular context. */
    public void setMessageDrivenContext(MessageDrivenContext ctx) {this.ctx = ctx;
    }
    /** Initializes the bean */
    public void ejbCreate() {System.err.println("ejbCreate()");}
    /** Our one business method*/
    public void onMessage(Message msg) {if (msg instanceof TextMessage) {
        TextMessage tm = (TextMessage) msg; try {
        String text = tm.getText(); System.err.println("Received new message : " + text);}
        catch(JMSEException e) {e.printStackTrace();}}
    public void ejbRemove() {System.err.println("ejbRemove()");} /** Destroys the bean*/
    
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason
U IV

---

---

---

---

---

---

---

---