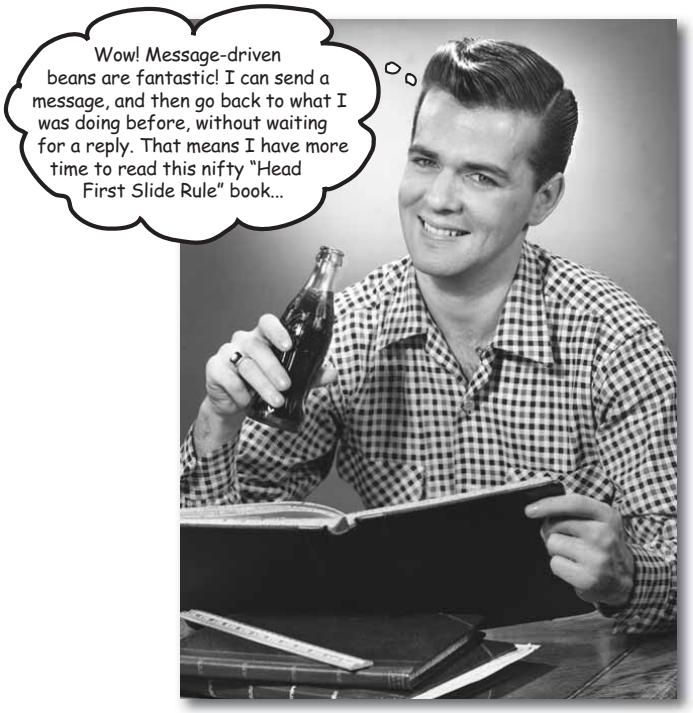
Table of Contents

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

8 message-driven beans

Getting the Message



It's fun to receive messages. Not as much fun as, say, getting that EBay package with the genuine Smurf™ lamp, but fun and *efficient* nonetheless. Imagine if you sent your order to EBay, and you couldn't leave your house until the package was delivered. That's what it's like with Session and Entity beans, because there's all calls to a bean's client interface (home, component, local, Remote, doesn't matter) are synchronous. But with message-driven beans, the client makes a message and sends it to a messaging service. Then the client walks away. Later, the messaging service sends the Container the message, and the Container give it to a message-driven bean.

this is a new chapter 437

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

***exam** objectives*

Message-Driven Beans

Official:	*What it really means:*
10.1 Identify correct and incorrect statements or examples about the client view, and lifecycle, of a message-driven bean.	You have to know that message-driven beans don't have clients! That means no home interface, no component interface, and no issues of local vs. Remote. Of course message-driven beans do have a caller, but that's the Container. And we don't consider the Container a *client*. The Container is the boss, not the customer. Message-driven beans have a very simply lifecycle—if you know how stateless session beans work, you know how message-driven beans work. The only different is that instead of bringing a bean out of the pool to service a client method call, the Container brings message-driven beans out of the pool to service *an incoming JMS message.*
10.2 Identify the interface(s) and methods a message-driven bean must implement.	You must know that message-driven beans implement two interfaces—javax.ejb.MessageDrivenBean, the interface with the Container callbacks (not many!) and javax.jms.MessageListener, the interface with a single onMessage(Message m) method. For a message-driven bean, onMessage() is the only business method.
10.3 Identify the use and behavior of the MessageDrivenContext interface methods.	You have to know that message-driven beans can't call most of the methods in MessageDrivenContext, ever. Think about it. If there's no client, how could you get client security info? If there's no client, which means no client view, how could you call getEJBObject() or getEJBHome()? You don't have a home! You don't have an EJBObject().
10.4 From a list, identify the responsibility of the Bean Provider, and the responsibility of the Container provider, for a message-driven bean.	You also have to recognize that if there's no client, you can't declare any checked exceptions. Who would catch them? Also, just as with stateless session beans, if you start a transaction in a method (which can only mean a BMT bean in onMessage()) you must finish it before the method ends.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Imagine this scenario...

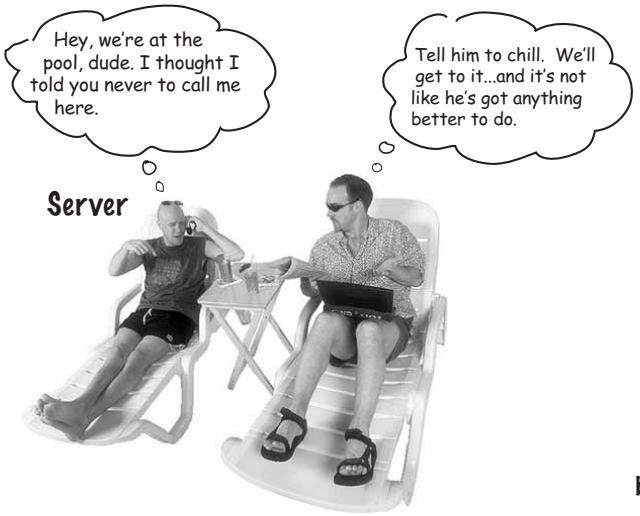
You have to ask someone to do a very important job.

You have no idea how long it's going to take them.

You have to wait right where you are until they finish.

You can't do anything else while you're waiting.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01



Too bad these guys aren't message-driven beans

Method calls, local or remote, are synchronous. The caller is stuck waiting until the server returns.

Message-driven beans (added in EJB 2.0) give you asynchronous communication between the client (message sender) and the server (message receiver).

In messaging terms, the sender is called the message Producer and the receiver is called the message Consumer.

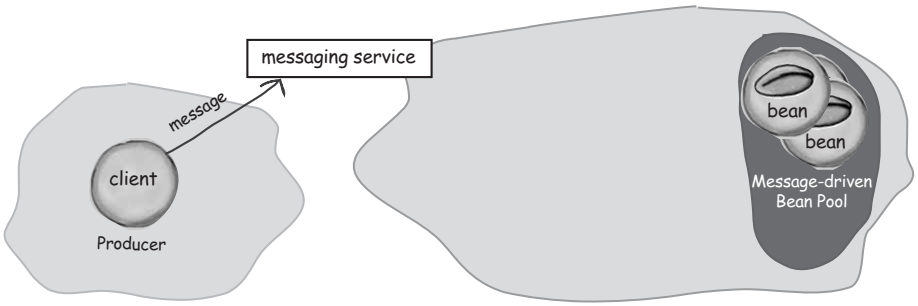
With messaging, the Producer sends a message and then goes about his business. He doesn't have to wait for the Consumer to even get the message, let alone process it.

When the Consumer gets a message, he processes it. In the meantime, the client can still have a life.

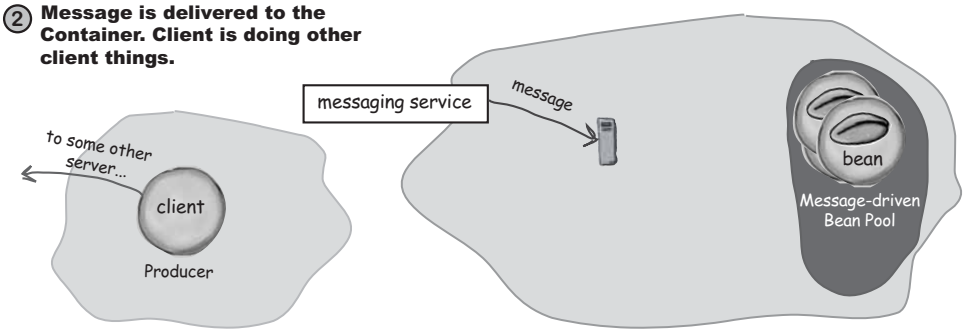
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Message-driven bean overview

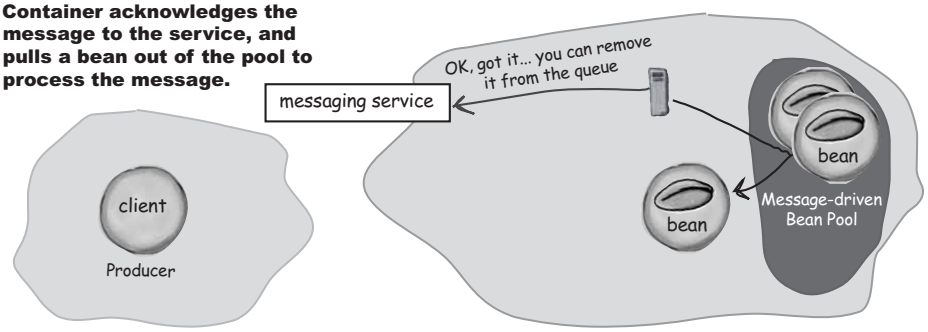
1 Client (Producer) send a message to the messaging service.



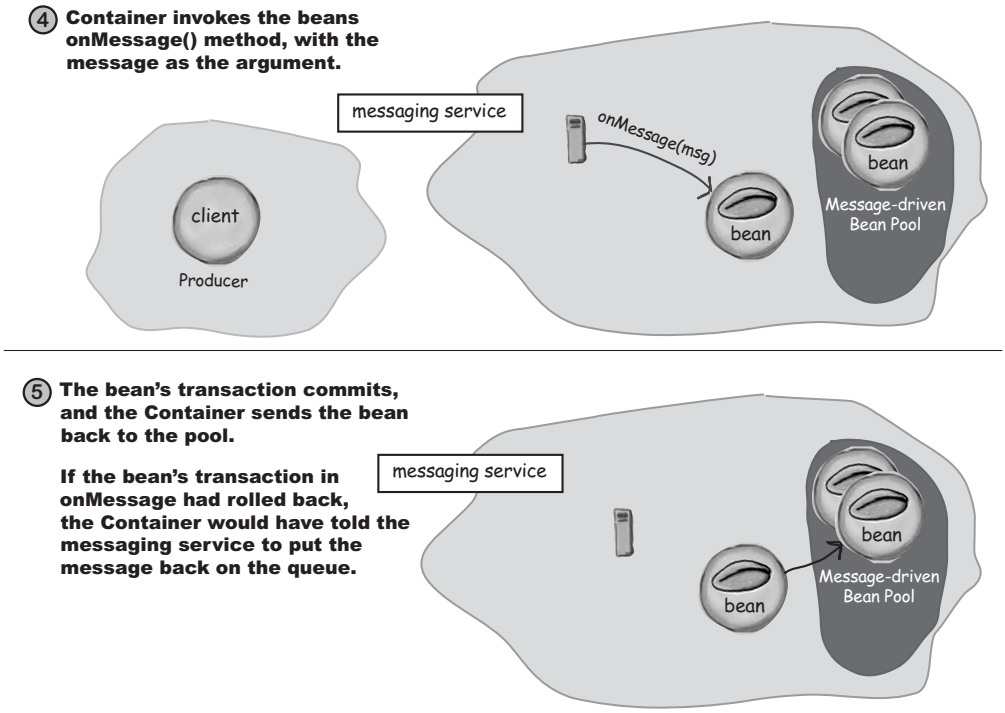
2 Message is delivered to the Container. Client is doing other client things.



3 Container acknowledges the message to the service, and pulls a bean out of the pool to process the message.



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

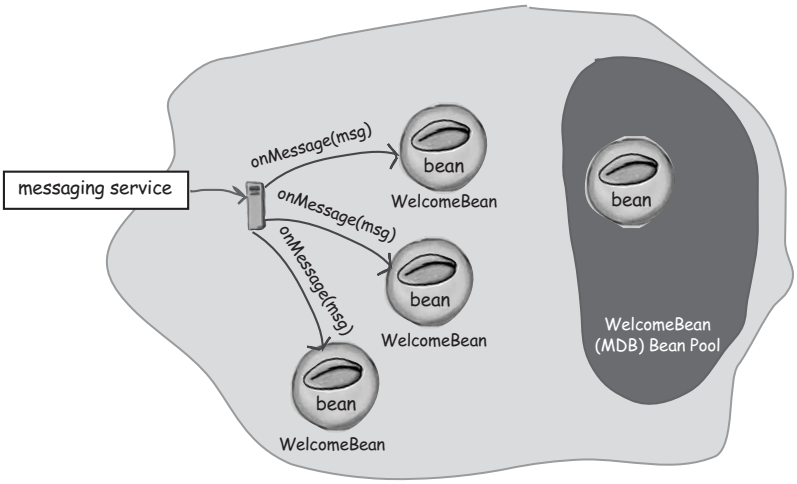


Don't they look a lot like stateless session beans?

Like stateless session beans, message-driven beans have no unique identity to clients (actually, they don't *have* clients, since the Container isn't really considered a *client* to the bean), they're pooled, and they have no individual state that affects their business logic. That means one message-driven bean (of a particular type) is the same as any other bean from the same home. Just like stateless beans.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

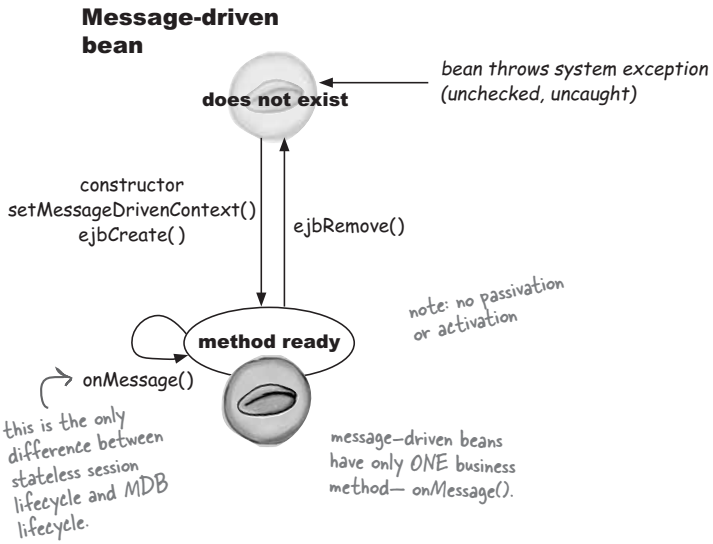
Multiple beans of the same type can process messages concurrently.



But the container will make sure that each bean is thread safe!

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The lifecycle of message-driven beans looks just like stateless session beans




```
public void onMessage(Message msg) {
   // open the message and do something with it

}
```

That's it! The only business logic thing a message-driven bean needs to do—or can do (other than the three container callbacks for creating, removing, and setting its context)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Message-driven bean class

```
package headfirst;

import javax.ejb.*;
import javax.jms.*;

public class WelcomeBean implements MessageDrivenBean, MessageListener {

   private MessageDrivenContext context;

   public void ejbCreate() { }

   public void ejbRemove() { }

   public void setMessageDrivenContext(MessageDrivenContext ctx) {
       context = ctx;
   }

   public void onMessage(Message msg) {
       // process the message
      try {

        if (msg instanceof TextMessage) {
         TextMessage message = (TextMessage) msg;

         System.out.println(message.getText());
        }
      } catch (JMSException ex) {
         ex.printStackTrace();
      }
   }
}
```

a new import: the javax.jms package

TWO interfaces to implement!

javax.ejb.MessageDrivenBean (with the container callbacks) javax.jms.MessageListener (with the onMessage() callback method)

extends EJBObject just like SessionContext and EntityContext

you MUST have a single, no-arg ejbCreate()

works just like stateless session beans... called when the Container wants to reduce the pool

just do the same thing you always do... save your context

FINALLY! A real business method. This is the only method defined in the MessageListener interface.

TextMessage is-a Message, but it might have been another kind of message like BytesMessage, MapMessage, ObjectMessage or StreamMessage. So we test it with instanceof before the cast.

Yikes. A checked exception! It's from the getText() method.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Writing a message-driven bean: your job as Bean Provider

You put THREE kinds of methods in the bean class:

1. **Bean Law: ejbCreate() method**

   Write a single, no-argument ejbCreate() method in the bean. It doesn't match anything or come from any interface. It's there because it MUST be.

2. **MessageListener implementation: onMessage()**

   This is your business method. Your *only* business method.

3. **MessageDrivenBean implementation: container callbacks**

   Implement both of the methods from the MessageDrivenBean interface, which your bean *must* implement in the official Java way (i.e. using the 'implements MessageDrivenBean' declaration either in your bean class or one of its superclasses)

Sharpen your pencil

Of the three types of methods you put in your bean, check off the ones the compiler cares about.

Compiler-checked?

- ❑ ejbCreate()
- ❑ onMessage()
- ❑ ejbRemove() and setMessageDrivenContext()

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Rules for the message-driven bean class

1. The class must implement *javax.ejb.MessageDrivenBean* and *javax.ejb.MessageListener*.

```
public class WelcomeBean implements MessageDrivenBean, MessageListener {
```

2. The class must be *public*, must not be *final*, and must not be *abstract*.

3. The class must have a public **constructor** that takes no arguments. (Just like the other beans... so the Container can make a new instance.)

```
public WelcomeBean() { }
```

If you can, just let the compiler put in the default constructor. But if you do have one, be SURE it's no-arg, and don't put any code in it. Wait for ejbCreate()

4. The class must have a no-arg **ejbCreate()** method. It must be *public*, not *final*, not *static*, with a void return type.

```
public void ejbCreate() { }
```

Put initialization code in here. By the time this method is called, you already have your context.

Message-driven beans can't declare checked exceptions!!

You'll learn more about exceptions in Chapter 10, but for now, just think about it. There's no client! So who are you expecting to wrap a try/catch around these calls? The Container will laugh if you try to do this. You MUST catch and handle any checked exceptions that you get!!

5. The class must define the **onMessage()** method from the MessageListener interface. It must be *public*, not *final*, not *static*, with a void return type, and it takes a single argument of type javax.jms.Message.

```
public void onMessage(Message msg) { ... }
```

The REAL business method. Your logic goes here...

6. You must have the **ejbRemove()** and **setMessageDrivenContext()** methods from the MessageDrivenBean interface, exactly as declared in the interface.

```
public void ejbRemove() { }
public void setMessageDrivenContext(MessageDrivenContext ctx) {
   context = ctx;
}
```

7. No methods in the class are allowed to throw application exceptions. (And they shouldn't be declaring runtime exceptions either, although it's technically legal.)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Notice something missing from the code?

We never said what kind of messages we're listening for, or where they might be coming from.

You don't put anything in your code that indicates the JMS destination. You can probably guess when that happens... *deploy-time.*

But as a Bean Provider, you tell the Deployer whether you're looking for messages from a topic or queue. And for that, we'll use a new tag in the deployment descriptor, just for message-driven beans. At deploy-time, the Deployer will bind your bean to a specific Topic or Queue configured as a resource in the EJB server.

```
<message-driven-destination>
    <destination-type>javax.jms.Topic</destination-type>
</message-driven-destination>
```

You tell the Deployer if you're expecting messages from a Topic or a Queue

Complete DD for a message-driven bean

```
<enterprise-beans>
    <message-driven>
      <ejb-name>WelcomeNewCustomer</ejb-name>
      <home>headfirst.CustomerHome</home>
      <remote>headfirst.Customer</remote>
      <ejb-class>headfirst.WelcomeBean</ejb-class>
      <transaction-type>Container</transaction-type>
      <message-driven-destination>
         <destination-type>javax.jms.Topic</destination-type>
      </message-driven-destination>
    </message-driven>
</enterprise-beans>
```

There's no client view! No local view, no Remote view. No view at all. So you don't put in the tags for home and component interface.

(We'll look at an optional tag in a few pages)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Topics and Queues

Messaging comes in two flavors: topics and queues, although topics come in two subflavors—durable subscriptions and non-durable subscriptions.

Queues are like FIFO lists (although First-In-First-Out order isn't guaranteed). The producer sends a message that's intended for just a single consumer. Once somebody processes the message, that's it. An example might be an employee reimbursement system, where the employee sends his reimbursement request to the messaging service, and somebody in accounting will process the request. It doesn't need to go to anybody else at that point. If it turns out that the next step is to send it for management processing, the accounting department might send a message to a different destination—the ManagerApproval queue.

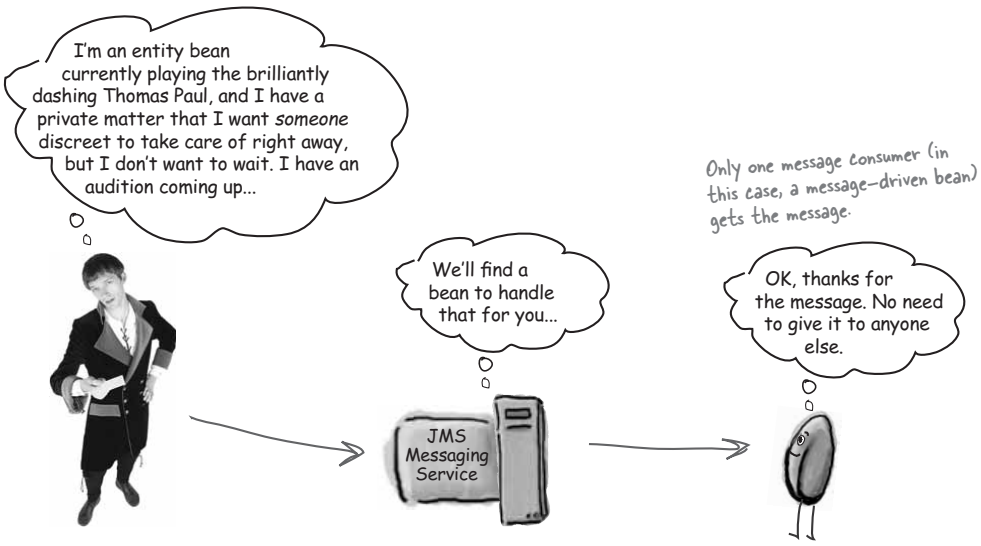
Topics use a publish and subscribe model, where a producer sends a message, and anyone who's listening as a consumer will get a copy of the message. Works just like a mailing list. If any one subscriber doesn't get the message, the producer doesn't care.

A topic subscriber can request a *durable subscription* if he wants to make sure that he sees all messages, including the ones that accumulated while he was offline, for example. That will almost always be your choice, because you'll want to get all the messages. Think about it...a non-durable topic subscription would be like you must be home at the time your magazine is delivered, os you simply won't see it.

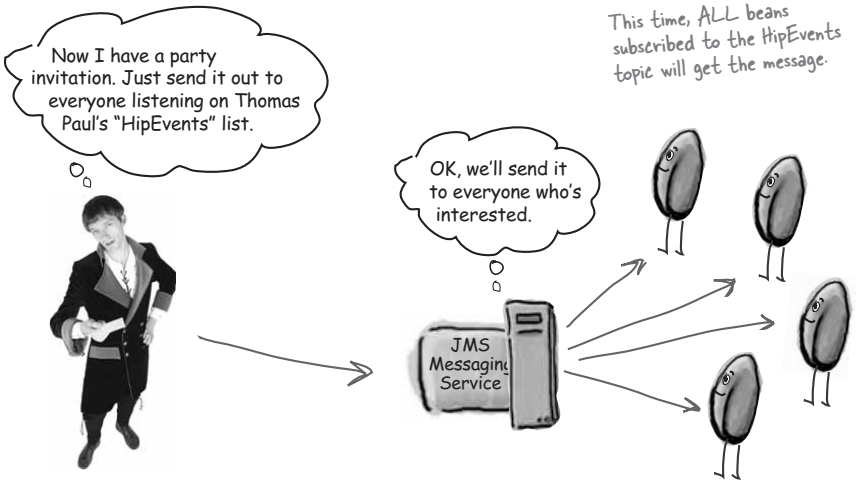
Enterprise systems tend to use either queues or durable topic subscriptions. A non-durable topic subscription might be appropriate sometimes, but if I'm a message consumer it's really like saying, "I don't care much about whether I get these messages."

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Queue: one to one (point-to-point)



Topic: one to many (publish/subscribe)



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

there are no Dumb Questions

Q: How does JMS fit into all this? And if I have a message-driven bean, where do I get the messaging service?

A: JMS 1.02 is the Java Messaging Service API that's required by EJB 2.0. That means your EJB container *must* have some kind of messaging service included, although many vendors can work with multiple messaging services as long as those services support the JMS API.

The JMS API works much like JDBC—you have a driver that knows how to take your standard JMS API calls, and translate them into something the underlying messaging system understands.

Q: Can I get messages from non-JMS messaging services?

A: Not right now, in EJB 2.0, but they (the infamous J2EE team) hope to add that capability... some day.

Q: Why doesn't the MessageDrivenBean just extend the MessageListener interface? That way you wouldn't have to implement both interfaces.

A: See your previous question. The J2EE team didn't want to lock all message-driven beans into being JMS listeners. One day, there might be many kinds of listener interfaces for different messaging types.

Q: Are messages guaranteed to come in order? Will I always get the first one first?

A: No!! You better not depend on it or Bad Things Might Happen. For example, you might get the "Cancel the Order" message before you get the "Place the Order" message. Design your system and code your bean on the assumption that message order is not strictly guaranteed.

Q: If it's a topic, does that mean *all* beans in a particular subscriber pool will get the message?

A: No! Remember, one bean can stand in for all the other beans. The Container will deliver the message to just ONE bean in the pool. *Unless*... no never mind.

Q: Unless *what*? Tell me!

A: OK, OK. But this isn't on the exam, so relax. Let's say your server is running as a clustered configuration, where you could have multiple instances of your server running. In that case, there is no guarantee about how the Container will represent itself to the messaging service.

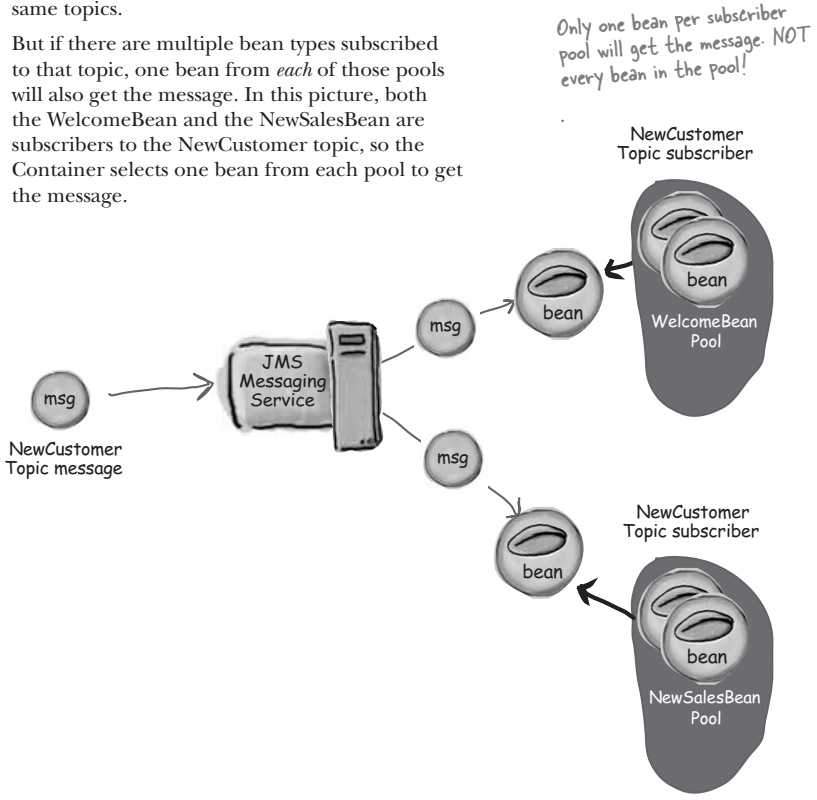
The question becomes, does the messaging service see the whole cluster as ONE version of your application, no matter how many JVMs its running on, or does the messaging service see each cluster (with its own duplicated pool for each bean type) as a separate listener?

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Only ONE bean per pool gets a copy of a topic message

If you have, say, the WelcomeBean subscribed to the NewCustomer topic, when a new message is published to that topic, the Container will choose *only one* bean from the WelcomeBean pool to get the message. Remember, the Container keeps a separate pool of beans for each home (and every deployed bean type gets its own home). The Container doesn't put all message-driven beans into one pool, even if they're subscribed to the same topics.

But if there are multiple bean types subscribed to that topic, one bean from *each* of those pools will also get the message. In this picture, both the WelcomeBean and the NewSalesBean are subscribers to the NewCustomer topic, so the Container selects one bean from each pool to get the message.



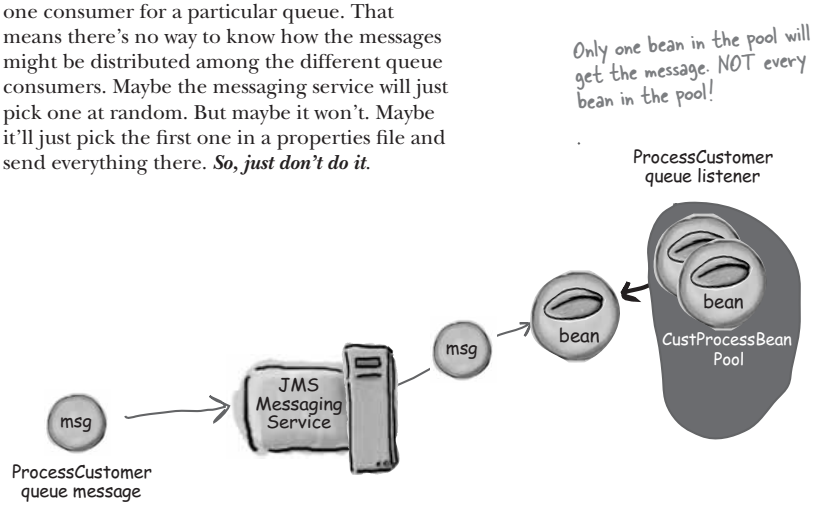
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

With a queue, only one bean gets the message. Period.

If you have a queue for messages requesting that a new customer be processed, only one bean from the pool associated with that queue will get the message. That's it.

Oh, and don't even think about asking what happens if you have more than one bean type (pool) associated with that queue.

But if you persist, we'll tell you that JMS doesn't define what happens when there is more than one consumer for a particular queue. That means there's no way to know how the messages might be distributed among the different queue consumers. Maybe the messaging service will just pick one at random. But maybe it won't. Maybe it'll just pick the first one in a properties file and send everything there. ***So, just don't do it.***



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

MessageDrivenContext

The server makes message-driven beans virtually the same way in which it makes stateless session beans.

1. Call the bean's constructor (must be no-arg).
2. Call the bean's context setter.
3. Call the beans ejbCreate() method.

In fact, the first two steps are the same for *all* bean types. The context setter always comes immediately after the bean's constructor, and at some point before ejbCreate(). (And of course an entity bean might never get an ejbCreate() call, if there aren't any clients trying to insert new entities into the database.)

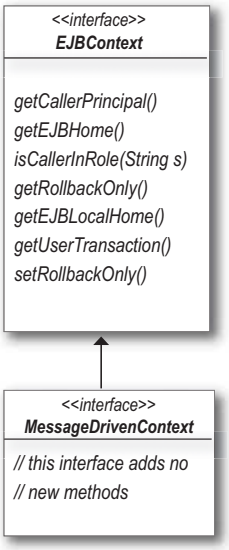
So, what can a message-driven bean do with its context? We think it's time for you to figure that out. We can guarantee there will be questions on the exam related to what a message-driven bean can and cannot do with its context. In fact, you'll find these questions scattered throughout the objectives, not just in the message-driven bean objectives (10.1 – 10.4). Questions from the transactions, exceptions, and security objectives might involve a message-driven bean and its context.

This is just our way of saying.. *do the damn exercise!*

You don't want to have to memorize this stuff, but if you just spend a few moments thinking about it, you'll figure it out.

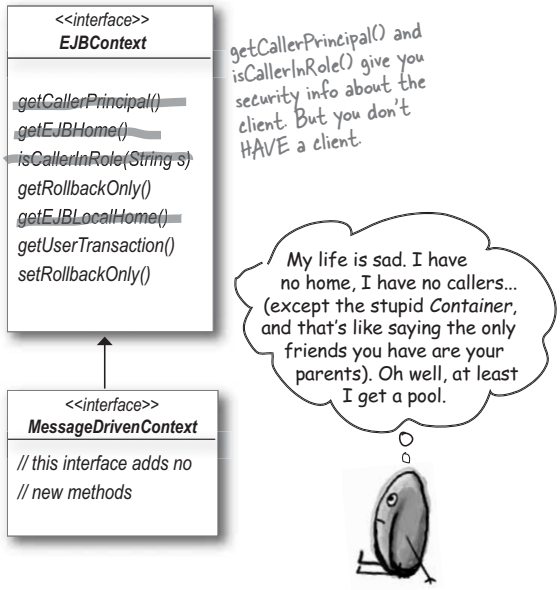
The answers are on the next page, though, so don't turn until you're done.

Think about which methods a message-driven bean could call on its context.
From the list of methods in EJBContext, cross out those methods that do not make sense for a message-driven bean (and by "do not make sense" we mean "will cause a horrible runtime exception with devastating and potentially career-ending consequences").
Oh, and be prepared to defend your answers. Imagine us sitting right behind you, scarily, demanding justification for your choices.



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

MessageDrivenContext



It's simple. Message-driven beans don't have clients. That means they don't have a client VIEW, so there's no home interface.

And since there's no client, there's no client security information.

So, you can't call the two methods for getting the home (since you HAVE no home), or the two methods for getting info about the caller's security.

there are no Dumb Questions

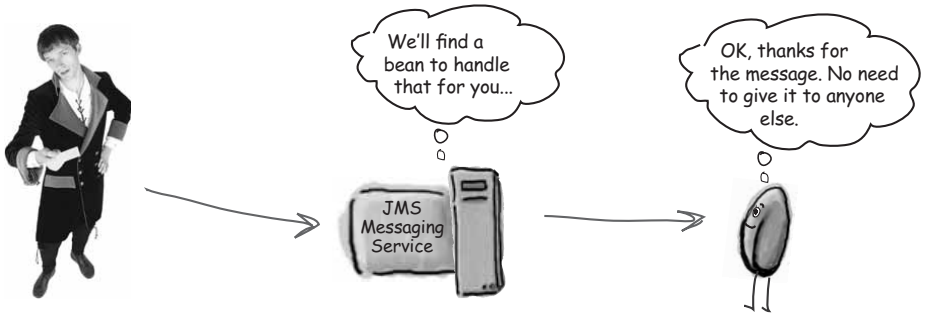
Q: You keep saying that message-driven beans don't have clients. But SOMEBODY has to call the bean's methods, right? They don't call themselves...

A: Technically, yes, the bean's methods are called. But we don't consider the Container to be a client. It's the boss, not the customer. So, in the Java sense, the Container is the bean's caller, but not a client. And as for getting security info, if the bean doesn't trust its own Container, you have waaaay bigger problems. We're talking one seriously paranoid bean.

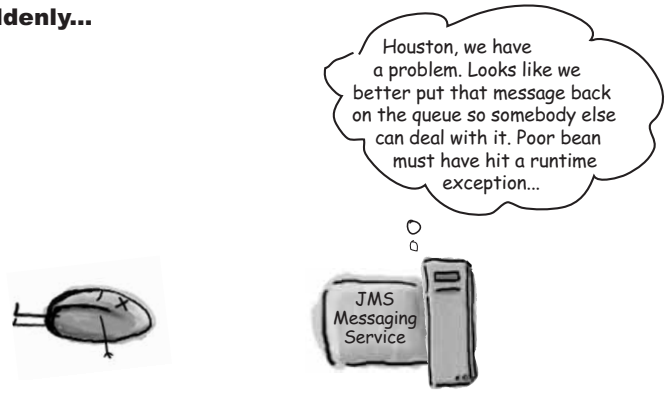
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

What if something goes wrong?

Everything was going so well...



When suddenly...



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Message acknowledgement

You don't want messages getting lost. If you have a crucial message on the queue, and the one bean who gets it can't commit—or worse, throws an exception and dies—what happens to your critical message?

That's the point of acknowledgement. The Container tells the messaging service (*not* the original producer) that the message was delivered and everything is fine.

But if later, while the consumer (the message-driven bean) is processing the message, a Bad Thing happens, the Container can tell the messaging service to put the message back in the queue.

How does the Container really know that something went wrong? Two ways, and it's your choice as a Bean Provider.

① The transaction status

Message acknowledgement is tied to whether the transaction commits or rolls back. If the transaction rolls back, the message goes back on the queue. You get this behavior for beans that use *container-managed transaction demarcation* (we'll cover transactions in the next chapter).

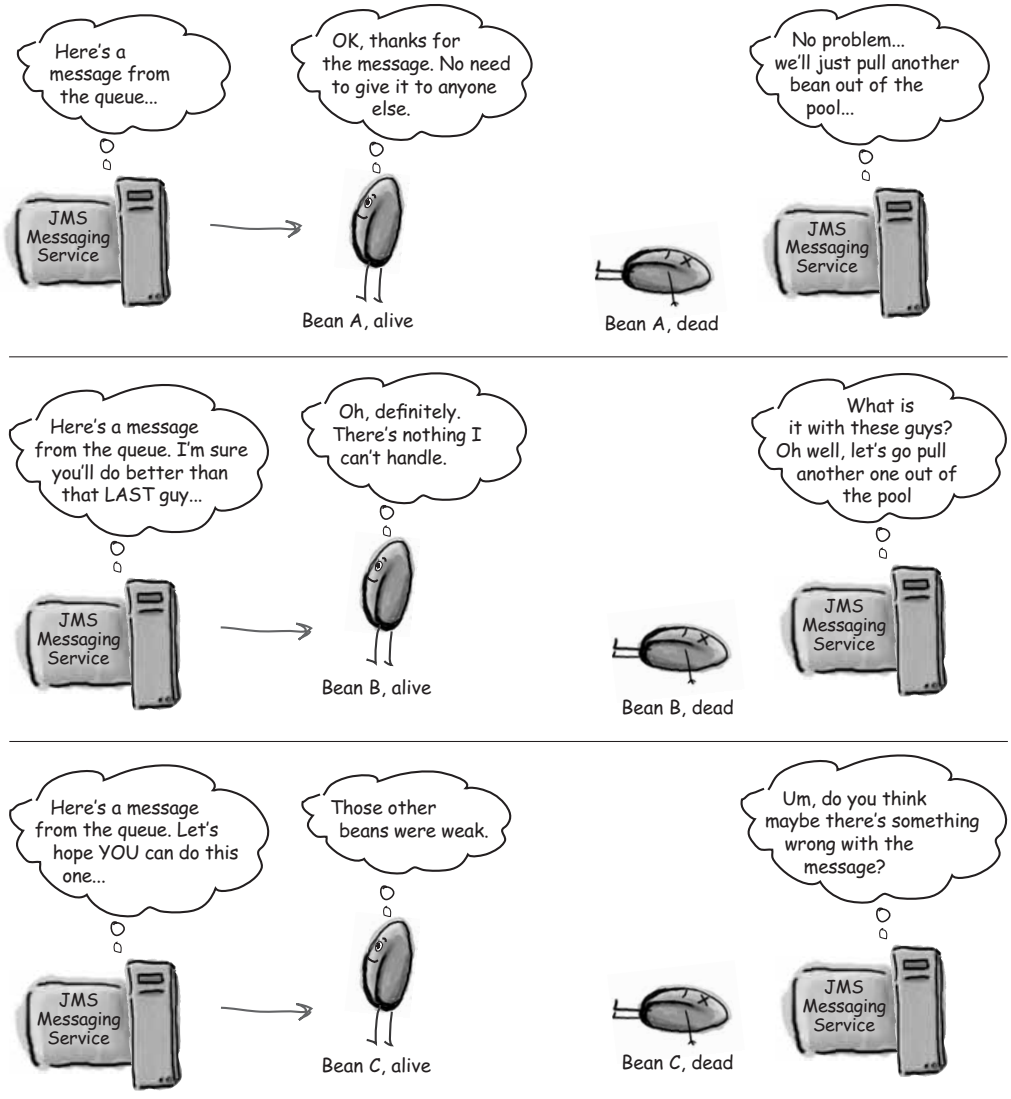
OR

② The method completion

Message acknowledgement is tied to whether the method completes successfully. If the method throws a runtime exception, the message goes back on the queue. You get this behavior for beans that use *bean-managed transaction demarcation.*

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

That's all well and good, but let's go back and see how our earlier scenario ended...



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Think about it.

If you want your message acknowledgement tied to the status of the transaction, use container-managed transactions (CMT). Most of the time, this will be your choice.

But if you want to decouple message-acknowledgement from the status of your transaction, your only choice is to use bean-managed transaction demarcation (BMT). With BMT, the Container looks only at whether your method completes. Method completes? Fine, the message acknowledgement stands. Runtime exception? Message goes back in the queue.

But what if you have a scenario where the bean can never commit, because of something inherently bad about the message? With CMT, that message will keep coming and coming and coming and... but not forever. Most messaging services let you configure how many times a particular message is resent before the service says, "This message looks like poison; let's send it to a Special Place (like a Bad Message queue) where it can't harm anyone". You really need to watch out for this, because there's nothing in the spec to help. You're reliant on your server vendor!

There is another possibility, though. With BMT, you could write your method in such a way that you rollback the method, but still finish the method. That way, the Container just goes whistling on its merry way thinking everything was fine. It has no idea that things went badly. In this scenario, you'd catch whatever exception comes up, rollback the transaction yourself (again, you'll learn how to do that very soon), and then end the method *looking* successful (at least to the Container).

Setting the acknowledgement mode

If you DO use BMT, you have two choices for how the Container sends an acknowledgement to the messaging service. The choices are:

```
<acknowledge-mode>Auto-acknowledge</acknowledge-mode>
```

OR

```
<acknowledge-mode>Dups-ok-acknowledge</acknowledge-mode>
```

This doesn't change the Container's behavior of whether it decides to do the acknowledgement—for a BMT bean it's always based on whether the method completes. The acknowledge mode is simply a way to tweak how the Container sends the acknowledgement.

With Dups-ok-acknowledge, you're telling the Container that you don't mind if it's kind of *slow* with the acknowledgement to the message service... that you don't *mind* if you get duplicate messages ("dups-ok"). This lets the Container take whatever time it wants to do the acknowledgement, and it might make the server more efficient, but at the risk of duplicate messages because the poor messaging service didn't get an acknowledgement quickly enough.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

1. Fill-in the blanks with the code that must be inserted for the bean to be a legal message-driven bean class.

```
package headfirst;

import javax.ejb.*;
import javax.jms.*;

public class FooListenerBean _____________________________________________ {

   private MessageDrivenContext context;



   public void _____________________________


   public void _____________________________


   public void setMessageDrivenContext(MessageDrivenContext ctx) {
       context = ctx;
   }

   public void _____________________________

}
```

2. List two things a stateful session bean can call on its SessionContext, that a message-driven bean can never call on its MessageDrivenContext.

_____

_____

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

1. Fill-in the blanks with the code that must be inserted for the bean to be a legal message-driven bean class.

```
package headfirst;

import javax.ejb.*;
import javax.jms.*;

public class FooListenerBean implements MessageDrivenBean, MessageListener {

   private MessageDrivenContext context;


   public void ejbCreate() { }

   public void ejbRemove() { }

   public void setMessageDrivenContext(MessageDrivenContext ctx) {
       context = ctx;
   }

   public void onMessage(Message msg) { }

}
```

2. List two things a stateful session bean can call on its SessionContext, that a message-driven bean can never call on its MessageDrivenContext.

call isCallerInRole() (there's no client!)

call getEJBHome() (there's no client view, so there's no home or component interface)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Mock Exam

---

1 What's true about message-driven beans? (Choose all that apply.)

- ❑ A. A message-driven bean has a home interface but no component interface.
- ❑ B. A client never knows a message-driven bean's identity.
- ❑ C. A client sees a message-driven bean as a JavaMail message consumer.
- ❑ D. The lifetime of a message-driven bean is controlled by the container.

---

2 Which interfaces must be implemented in a message-driven bean class or in one of its superclasses? (Choose all that apply.)

- ❑ A. `javax.jms.Message`
- ❑ B. `javax.jms.MessageListener`
- ❑ C. `javax.jms.MessageConsumer`
- ❑ D. `javax.ejb.MessageDrivenBean`
- ❑ E. `javax.ejb.MessageDrivenContext`

---

3 Which list properly sequences the methods called in the lifecycle of a message-driven bean? (Choose all that apply.)

- ❑ A. `ejbCreate()`, `newInstance()`, `setMessageDrivenContext()`, `onMessage()`
- ❑ B. `onMessage()`, `newInstance()`, `ejbCreate()`, `setMessageDrivenContext()`
- ❑ C. `newInstance()`, `setMessageDrivenContext()`, `ejbCreate()`, `onMessage()`
- ❑ D. `newInstance()`, `ejbCreate()`, `setMessageDrivenContext()`, `onMessage()`
- ❑ E. `ejbCreate()`, `setMessageDrivenContext()`, `newInstance()`, `onMessage()`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

4 Which are valid signature(s) for methods in a message-driven bean? (Choose all that apply.)

- ❑ A. `public void onMessage()`
- ❑ B. `public void ejbCreate()`
- ❑ C. `public static void onMessage()`
- ❑ D. `public void ejbCreate(javax.jms.Message m)`
- ❑ E. `public void onMessage(javax.jms.Message m)`
- ❑ F. `public void onMessage(javax.jms.Message m) throws java.rmi.RemoteException`

5 When is a message-driven bean able to access java:comp/env via JNDI?

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ❑ C. `setMessageDrivenContext()`
- ❑ D. None of the above

6 Which message-driven bean methods take an argument? (Choose all that apply.)

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ❑ C. `onMessage()`
- ❑ D. `setMessageDrivenContext()`

7 When is a message-driven bean able to access other enterprise beans?

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ❑ C. `onMessage()`
- ❑ D. `setMessageDrivenContext()`
- ❑ E. None of the above

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

8 What's true about Container support for message-driven beans? (Choose all that apply.)

- ❑ A. The Container must support the deployment of a message-driven bean as the consumer of a JavaMail queue.
- ❑ B. The Container is NOT required to support transaction scoping for message-driven beans.
- ❑ C. The Container guarantees first-in, first delivered message processing.
- ❑ D. The Container must ensure that the bean instances are non-reentrant.

9 When is a message-driven bean with BMT demarcation able to access resource managers?

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ❑ C. `onMessage()`
- ❑ D. `setMessageDrivenContext()`
- ❑ E. None of the above

10 What's true about message acknowledgment for message-driven beans? (Choose all that apply.)

- ❑ A. Message acknowledgement modes cannot be defined declaratively.
- ❑ B. The JMS API should be used for message acknowledgment.
- ❑ C. For BMT beans, the Container uses the acknowledge-mode deployment descriptor element.
- ❑ D. For CMT beans, the Container uses the acknowledge-mode deployment descriptor element.

11 What's true about the deployment descriptor for message-driven beans? (Choose all that apply.)

- ❑ A. The bean provider must guarantee that the bean is associated with a specific queue or topic.
- ❑ B. The deployment descriptor can indicate whether a bean is intended for a topic or a queue.
- ❑ C. It can indicate whether a Queue type bean should support durable subscription or not.
- ❑ D. It is appropriate to associate multiple beans with the same JMS queue.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

mock exam answers

Mock Exam Answers

(spec: 311–312)

1 What's true about message-driven beans? (Choose all that apply.)

- ❑ A. A message-driven bean has a home interface but no component interface. – MDBs have NO client view
- ☑ B. A client never knows a message-driven bean's identity.
- ❑ C. A client sees a message-driven bean as a JavaMail message consumer.
- ☑ D. The lifetime of a message-driven bean is controlled by the container.

(spec: 314–315)

2 Which interfaces must be implemented in a message-driven bean class or in one of its superclasses? (Choose all that apply.)

- ❑ A. `javax.jms.Message`
- ☑ B. `javax.jms.MessageListener` – this is where the onMessage() method is defined
- ❑ C. `javax.jms.MessageConsumer`
- ☑ D. `javax.ejb.MessageDrivenBean`
- ❑ E. `javax.ejb.MessageDrivenContext`

(spec: 319)

3 Which list properly sequences the methods called in the lifecycle of a message-driven bean? (Choose all that apply.)

- ❑ A. `ejbCreate()`, `newInstance()`, `setMessageDrivenContext()`, `onMessage()`
- ❑ B. `onMessage()`, `newInstance()`, `ejbCreate()`, `setMessageDrivenContext()`
- ☑ C. `newInstance()`, `setMessageDrivenContext()`, `ejbCreate()`, `onMessage()`
- ❑ D. `newInstance()`, `ejbCreate()`, `setMessageDrivenContext()`, `onMessage()`
- ❑ E. `ejbCreate()`, `setMessageDrivenContext()`, `newInstance()`, `onMessage()`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

4 Which are valid signature(s) for methods in a message-driven bean? (Choose all that apply.) (spec: 324)

- ❑ A. `public void onMessage()`
- ☑ B. `public void ejbCreate()`
- ❑ C. `public static void onMessage()`
- ❑ D. `public void ejbCreate(javax.jms.Message m)`
- ☑ E. `public void onMessage(javax.jms.Message m)`
- ❑ F. `public void onMessage(javax.jms.Message m) throws java.rmi.RemoteException` – you must never declare a RemoteException

5 When is a message-driven bean able to access java:comp/env via JNDI? (spec: 320)

- ☑ A. `ejbCreate()`
- ☑ B. `ejbRemove()`
- ☑ C. `setMessageDrivenContext()`
- ❑ D. None of the above

6 Which message-driven bean methods take an argument? (Choose all that apply.) (spec: 319, 324)

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ☑ C. `onMessage()` – takes a message
- ☑ D. `setMessageDrivenContext()` – takes a context

7 When is a message-driven bean able to access other enterprise beans? (spec: 320)

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ☑ C. `onMessage()` – only in a method with a 'meaningful transaction context'
- ❑ D. `setMessageDrivenContext()`
- ❑ E. None of the above

mock exam answers

8 What's true about Container support for message-driven beans? (Choose all that apply.) (spec: 325–326)

- ❑ A. The Container must support the deployment of a message-driven bean as the consumer of a JavaMail queue.
- ❑ B. The Container is NOT required to support transaction scoping for message-driven beans.
- ❑ C. The Container guarantees first-in, first delivered message processing.
- ☑ D. The Container must ensure that the bean instances are non-reentrant.

9 When is a message-driven bean with BMT demarcation able to access resource managers? (spec: 320)

- ❑ A. `ejbCreate()`
- ❑ B. `ejbRemove()`
- ☑ C. `onMessage()` – when there is a 'meaningful transaction context'
- ❑ D. `setMessageDrivenContext()`
- ❑ E. None of the above

10 What's true about message acknowledgment for message-driven beans? (Choose all that apply.) (spec: 317)

- ❑ A. Message acknowledgement modes cannot be defined declaratively.
- ❑ B. The JMS API should be used for message acknowledgment.
- ☑ C. For BMT beans, the Container uses the acknowledge-mode deployment descriptor element. – can be Auto-acknowledge or Dups-ok-acknowledge
- ❑ D. For CMT beans, the Container uses the acknowledge-mode deployment descriptor element.

11 What's true about the deployment descriptor for message-driven beans? (Choose all that apply.) (spec: 317)

- ❑ A. The bean provider must guarantee that the bean is associated with a specific queue or topic. – that's the deployer's job
- ☑ B. The deployment descriptor can indicate whether a bean is intended for a topic or a queue.
- ❑ C. It can indicate whether a Queue type bean should support durable subscription or not. – durable subscriptions are just for topics!
- ❑ D. It is appropriate to associate multiple beans with the same JMS queue.