Message-Driven Bean

# What is Message-Driven Bean?

Like session beans, MDBs process business logic. However, MDBs are different in one important way: clients never invoke MDB methods directly. Instead, MDBs are triggered by messages sent to a messaging server, which enables sending asynchronous messages between system components. MDBs are typically used for robust system integration or asynchronous processing.

MDB Programming rules

Like all EJBs, MDBs are plain Java objects that follow a simple set of rules and sometimes have annotations. Don’t take these rules too seriously yet; simply note them in preparation for going through the code-intensive sections that follow.

■ The MDB class must directly (by using the implements keyword in the class declaration) or indirectly (through annotations or descriptors) implement a message listener interface.

■ The MDB class must be concrete. It cannot be either a final or an abstract class.

■ The MDB must be a POJO class and not a subclass of another MDB.

■ The MDB class must be declared public.

■ The bean class must have a no-argument constructor. If you don’t have any constructors in your Java class, the compiler will create a default constructor. The container uses this constructor to create a bean instance.

■ You cannot define a finalize method in the bean class. If any cleanup code is necessary, it should be defined in a method designated as PreDestroy.

■ You must implement the methods defined in the message listener interface.These methods must be public and cannot be static or final.

■ You must not throw the javax.rmi.RemoteException or any runtime exceptions. If a RuntimeException is thrown, the MDB instance is terminated.

# MDB Code Example:

#### Simple JMS steps to send message:



#### PlaceOrderBean that produces the JMS message



#### Description :

Like stateless beans, MDBs are not guaranteed to maintain state. The

@MessageDriven annotation is the MDB counterpart of the @Stateless and

@Stateful annotations—it makes the container transparently provide messaging

and other EJB services into a POJO. The activation configuration properties

nested inside the @MessageDriven annotation tells the container what JMS destination

the MDB wants to receive messages from.

NOTE Behind the scenes, the container takes care of several mechanical

details to start listening for messages sent to the destination specified

by the activation configuration properties. As soon as a message arrives

at the destination, the container forwards it to an instance of the MDB.

Instead of implementing a remote or local business interface, MDBs implement the

javax.jms.MessageListener interface. The container uses this well-known JMS

interface to invoke an MDB. The onMessage method defined by the interface has a

single javax.jms.Message parameter that the container uses to pass a received

message to the MDB.

#### OrderBillingMDB : message consumer

package ejb3inaction.example.buslogic;

import javax.ejb.MessageDriven;

import javax.ejb.ActivationConfigProperty;

import javax.jms.Message;

import javax.jms.MessageListener;

import javax.jms.ObjectMessage;

import ejb3inaction.example.persistence.Order;

import ejb3inaction.example.persistence.OrderStatus;

@MessageDriven(

activationConfig = {

@ActivationConfigProperty(

propertyName="destinationName",

propertyValue="jms/OrderBillingQueue")

}

)

public class OrderBillingMDB implements MessageListener {

...

public void onMessage(Message message) {

try {

ObjectMessage objectMessage = (ObjectMessage) message;

Order order = (Order) objectMessage.getObject();

try {

bill(order);

notifyBillingSuccess(order);

order.setStatus(OrderStatus.COMPLETE);

} catch (BillingException be) {

notifyBillingFailure(be, order);

order.setStatus(OrderStatus.BILLING\_FAILED);

} finally {

update(order);

}

} catch (Exception e) {

e.printStackTrace();

}

}

...

}