**JMS Interview Questions**

**Q1: What is JMS?**

A: JMS is an acronym used for Java Messaging Service. It is Java's answer to creating software using asynchronous messaging. It is one of the official specifications of the J2EE technologies and is a key technology.

**Q2: How JMS is different from RPC?**

A: In RPC the method invoker waits for the method to finish execution and return the control back to the invoker. Thus it is completely synchronous in nature. While in JMS the message sender just sends the message to the destination and continues it's own processing. The sender does not wait for the receiver to respond. This is asynchronous behavior.

**Q3: What are the advantages of JMS?**

A: JMS is asynchronous in nature. Thus not all the pieces need to be up all the time for the application to function as a whole. Even if the receiver is down the MOM will store the messages on it's behalf and will send them once it comes back up. Thus at least a part of application can still function as there is no blocking.

**Q4: Are you aware of any major JMS products available in the market?**

A: IBM's MQ Series is one of the most popular product used as Message Oriented Middleware. Some of the other products are SonicMQ, iBus etc.All the J2EE compliant application servers come built with their own implementation of JMS.

**Q5: What are the different types of messages available in the JMS API?**

A: Message, TextMessage, BytesMessage, StreamMessage, ObjectMessage, MapMessage are the different messages available in the JMS API.

**Q6: What are the different messaging paradigms JMS supports?**

A: Publish and Subscribe i.e. pub/suc and Point to Point i.e. p2p.

**Q7: What is the difference between topic and queue?**

A: A topic is typically used for one to many messaging i.e. it supports publish subscribe model of messaging. While queue is used for one-to-one messaging i.e. it supports Point to Point Messaging.

**Q8: What is the role of JMS in enterprise solution development?**

A: JMS is typically used in the following scenarios

1. Enterprise Application Integration: - Where a legacy application is integrated with a new application via messaging.

2. B2B or Business to Business: - Businesses can interact with each other via messaging because JMS allows organizations to cooperate without tightly coupling their business systems.

3. Geographically dispersed units: - JMS can ensure safe exchange of data amongst the geographically dispersed units of an organization.

4. One to many applications: - The applications that need to push data in packet to huge number of clients in a one-to-many fashion are good candidates for the use JMS. Typical such applications are Auction Sites, Stock Quote Services etc.

**Q9: What is the use of Message object?**

A: Message is a light weight message having only header and properties and no payload. Thus if theIf the receivers are to be notified abt an event, and no data needs to be exchanged then using Message can be very efficient.

**Q10: What is the basic difference between Publish Subscribe model and P2P model?**

A: Publish Subscribe model is typically used in one-to-many situation. It is unreliable but very fast. P2P model is used in one-to-one situation. It is highly reliable.

**Q11: What is the use of BytesMessage?**

A: BytesMessage contains an array of primitive bytes in it's payload. Thus it can be used for transfer of data between two applications in their native format which may not be compatible with other Message types. It is also useful where JMS is used purely as a transport between two systems and the message payload is opaque to the JMS client. Whenever you store any primitive type, it is converted into it's byte representation and then stored in the payload. There is no boundary line between the different data types stored. Thus you can even read a long as short. This would result in erroneous data and hence it is advisable that the payload be read in the same order and using the same type in which it was created by the sender.

**Q12: What is the use of StreamMessage?**

A: StreamMessage carries a stream of Java primitive types as it's payload. It contains some conveient methods for reading the data stored in the payload. However StreamMessage prevents reading a long value as short, something that is allwed in case of BytesMessage. This is so because the StreamMessage also writes the type information alonwgith the value of the primitive type and enforces a set of strict conversion rules which actually prevents reading of one primitive type as another.

**Q13: What is the use of TextMessage?**

A: TextMessage contains instance of java.lang.String as it's payload. Thus it is very useful for exchanging textual data. It can also be used for exchanging complex character data such as an XML document.

**Q14: What is the use of ObjectMessage?**

A: ObjectMessage contains a Serializable java object as it's payload. Thus it allows exchange of Java objects between applications. This in itself mandates that both the applications be Java applications. The consumer of the message must typecast the object received to it's appropriate type. Thus the consumer should before hand know the actual type of the object sent by the sender. Wrong type casting would result in ClassCastException. Moreover the class definition of the object set in the payload should be available on both the machine, the sender as well as the consumer. If the class definition is not available in the consumer machine, an attempt to type cast would result in ClassNotFoundException. Some of the MOMs might support dynamic loading of the desired class over the network, but the JMS specification does not mandate this behavior and would be a value added service if provided by your vendor. And relying on any such vendor specific functionality would hamper the portability of your application. Most of the time the class need to be put in the classpath of both, the sender and the consumer, manually by the developer.

**Q15: What is the use of MapMessage?**

A: A MapMessage carries name-value pair as it's payload. Thus it's payload is similar to the java.util.Properties object of Java. The values can be Java primitives or their wrappers.

**Q16: What is the difference between BytesMessage and StreamMessage?**

A: BytesMessage stores the primitive data types by converting them to their byte representation. Thus the message is one contiguous stream of bytes. While the StreamMessage maintains a boundary between the different data types stored because it also stores the type information along with the value of the primitive being stored. BytesMessage allows data to be read using any type. Thus even if your payload contains a long value, you can invoke a method to read a short and it will return you something. It will not give you a semantically correct data but the call will succeed in reading the first two bytes of data. This is strictly prohibited in the StreamMessage. It maintains the type information of the data being stored and enforces strict conversion rules on the data being read.

**Q17: What is point-to-point messaging?**

A: With point-to-point message passing the sending application/client establishes a named message queue in the JMS broker/server and sends messages to this queue. The receiving client registers with the broker to receive messages posted to this queue. There is a one-to-one relationship between the sending and receiving clients.

**Q18: Can two different JMS services talk to each other? For instance, if A and B are two different JMS providers, can Provider A send messages directly to Provider B? If not, then can a subscriber to Provider A act as a publisher to Provider B?**

A: The answers are no to the first question and yes to the second. The JMS specification does not require that one JMS provider be able to send messages directly to another provider. However, the specification does require that a JMS client must be able to accept a message created by a different JMS provider, so a message received by a subscriber to Provider A can then be published to Provider B. One caveat is that the publisher to Provider B is not required to handle a JMSReplyTo header that refers to a destination that is specific to Provider A.

**Q19: What is the advantage of persistent message delivery compared to nonpersistent delivery?**

A: If the JMS server experiences a failure, for example, a power outage, any message that it is holding in primary storage potentially could be lost. With persistent storage, the JMS server logs every message to secondary storage. (The logging occurs on the front end, that is, as part of handling the send operation from the message producing client.) The logged message is removed from secondary storage only after it has been successfully delivered to all consuming clients.

**Q20: Give an example of using the publish/subscribe model.**

A: JMS can be used to broadcast shutdown messages to clients connected to the Weblogic server on a module wise basis. If an application has six modules, each module behaves like a subscriber to a named topic on the server.

**Q21: Why doesn't the JMS API provide end-to-end synchronous message delivery and notification of delivery?**

A: Some messaging systems provide synchronous delivery to destinations as a mechanism for implementing reliable applications. Some systems provide clients with various forms of delivery notification so that the clients can detect dropped or ignored messages. This is not the model defined by the JMS API.

JMS API messaging provides guaranteed delivery via the once-and-only-once delivery semantics of PERSISTENT messages. In addition, message consumers can ensure reliable processing of messages by using either CLIENT\_ACKNOWLEDGE mode or transacted sessions. This achieves reliable delivery with minimum synchronization and is the enterprise messaging model most vendors and developers prefer.

The JMS API does not define a schema of systems messages (such as delivery notifications). If an application requires acknowledgment of message receipt, it can define an application-level acknowledgment message.

**Q22: What are the various message types supported by JMS?**

A: Stream Messages ? Group of Java Primitives

Map Messages ? Name Value Pairs. Name being a string& Value being a java primitive

Text Messages ? String messages (since being widely used a separate messaging Type has been supported)

Object Messages ? Group of serialize able java object

Bytes Message ? Stream of uninterrupted bytes

**Q23: How is a java object message delivered to a non-java Client?**

A: It is according to the specification that the message sent should be received in the same format. A non-java client cannot receive a message in the form of java object. The provider in between handles the conversion of the data type and the message is transferred to the other end.

**Q24: What is MDB and What is the special feature of that?**

A: MDB is Message driven bean, which very much resembles the Stateless session bean. The incoming and out going messages can be handled by the Message driven bean. The ability to communicate asynchronously is the special feature about the Message driven bean.

**Q25: What are the types of messaging?**

A: There are two kinds of Messaging.

Synchronous Messaging: Synchronous messaging involves a client that waits for the server to respond to a message.

Asynchronous Messaging: Asynchronous messaging involves a client that does not wait for a message from the server. An event is used to trigger a message from a server.

**Q26: What are the core JMS-related objects required for each JMS-enabled application?**

A: Each JMS-enabled client must establish the following:

• A connection object provided by the JMS server (the message broker)

• Within a connection, one or more sessions, which provide a context for message sending and receiving

• Within a session, either a queue or topic object representing the destination (the message staging area) within the message broker

• Within a session, the appropriate sender or publisher or receiver or subscriber object (depending on whether the client is a message producer or consumer and uses a point-to-point or publish/subscribe strategy, respectively) Within a session, a message object (to send or to receive).

**Q27: How may messaging models do JMS provide for and what are they?**

A: JMS provides for two messaging models, publish-and-subscribe and point-to-point queuing.

**Q28: What are the types of messaging?**

A: There are two kinds of Messaging:

Synchronous messaging involves a client that waits for the server to respond to a message.

Asynchronous messaging involves a client that does not wait for a message from the server. An event is used to trigger a message from a server.

**Q29: What is publish/subscribe messaging?**

A: With publish/subscribe message passing the sending application/client establishes a named topic in the JMS broker/server and publishes messages to this queue. The receiving clients register (specifically, subscribe) via the broker to messages by topic; every subscriber to a topic receives each message published to that topic. There is a one-to-many relationship between the publishing client and the subscribing clients.

**Q30: Why doesn’t the JMS API provide end-to-end synchronous message delivery and notification of delivery?**

A: Some messaging systems provide synchronous delivery to destinations as a mechanism for implementing reliable applications. Some systems provide clients with various forms of delivery notification so that the clients can detect dropped or ignored messages. This is not the model defined by the JMS API. JMS API messaging provides guaranteed delivery via the once-and-only-once delivery semantics of PERSISTENT messages. In addition, message consumers can insure reliable processing of messages by using either CLIENT\_ACKNOWLEDGE mode or transacted sessions. This achieves reliable delivery with minimum synchronization and is the enterprise messaging model most vendors and developers prefer. The JMS API does not define a schema of systems messages (such as delivery notifications). If an application requires acknowledgment of message receipt, it can define an application-level acknowledgment message.

**Q31: What are the core JMS-related objects required for each JMS-enabled application?**

A: Each JMS-enabled client must establish the following:

A connection object provided by the JMS server (the message broker)

Within a connection, one or more sessions, which provide a context for message sending and receiving Within a session, either a queue or topic object representing the destination (the message staging area) within the message broker Within a session, the appropriate sender or publisher or receiver or subscriber object (depending on whether the client is a message producer or consumer and uses a point-to-point or publish/subscribe strategy, respectively). Within a session, a message object (to send or to receive)

**Q32: What is the Role of the JMS Provider?**

A: The JMS provider handles security of the messages, data conversion and the client triggering. The JMS provider specifies the level of encryption and the security level of the message, the best data type for the non-JMS client.

**Q33: How does a typical client perform the communication?**

A: Use JNDI to locate administrative objects.

Locate a single ConnectionFactory object.

Locate one or more Destination objects.

Use the ConnectionFactory to create a JMS Connection.

Use the Connection to create one or more Session(s).

Use a Session and the Destinations to create the MessageProducers and MessageConsumers needed.

Perform your communication.

**Q34: Give an example of using the point-to-point model.**

A: The point-to-point model is used when the information is specific to a single client. For example, a client can send a message for a print out, and the server can send information back to this client after completion of the print job.

**Q35: Does Tomcat support JMS (Java Messaging Service)?**

A: Tomcat is just a servlet container, not an EJB container nor an application server, so it does not contains any JMS basic support. However, there’s nothing stopping you from using another JMS provider.

**Q36: Is it possible to send email messages using JMS?**

A: JMS has no inherent support for email operations.

**Q37: How do I communicate between two clients that are on different machines on a network using JMS? I want to use a standalone application for communicating between the machine and I want to pass the message using JMS.**

A: You can make two JMS client applications, say AppA and AppB. Make AppA listen to topic ‘forA’. Make AppB listen to topic ‘forB’.

If AppA sends a message to topic ‘forB’, AppB will receive it. If AppB sends a message to topic ‘forA’, AppA will receive it.

For sample code etc, try downloading SonicMQ (as a JMS server) and go through the samples.

**Q38: Is there any relationship between javax.jms.Message and javax.mail.Message?**

A: There is no direct relationship between javax.mail.Message and javax.jms.Message. If your requirement is to map (correlate) them, here is what you can do:

* From JMS domain to JavaMail domain (a javax.jms.Message is received):
* A JMS topic/queue can be associated with one or many e-mail id(s).
* The JMS Message Header can be mapped to ‘custom’ JavaMail Message Header.
* The JMS Message Body can be associated with the JavaMail Message body.
* A JavaMail client application should be able to process these ‘custom’ headers and the content of the message body.
* From JavaMail domain to JMS domain (a javax.mail.Message is received):
* An e-mail id can be associated with one or more JMS topics/queues.
* The JavaMail Message Header can be mapped to ‘custom’ JMS Message Header.
* The JavaMail Message Body can be associated with the JMS Message body.
* The JMS client application should be able to process these ‘custom’ headers and the content of the message body.
* In a simple application that I tried, I removed the ‘custom’ header scenario and just forwarded the contents of the message (text message), which worked without any problems.Try using SonicMQ bridges, which already has something like that.

**Q39: Is it possible to acknowledge individual messages on a queue without affecting previously received, but as yet unacknowledged, messages?**

A: If you acknowledge a message, all previously received messages will also be acknowledged. From the javax.jms.Message Javadoc, the acknowledge method will “Acknowledge this and all previous messages received.”

So the answer to your question is no, if what you meant by “affecting” is not-yet acknowledged.

I suggest an alternative. You should look at javax.jms.QueueBrowser to review queued messages. QueueBrowser has getEnumeration, which “Gets an enumeration for browsing the current queue messages in the order they would be received”.

**Q40: What encryption options are there for sending messages through JMS?**

A: Encryption is not handled by the JMS specification. It is left to the JMS provider to implement and provide encryption and decryption of messages. These days, Progress Software’s SonicMQ is a leading JMS provider and they have a robust encryption mechanism called Quality of Protection. They also provide an SSL-related feature, which also has build in encryption.

**Q41: How does the Application server handle the JMS Connection?**

A:

* Application server creates the server session and stores them in a pool.
* Connection consumer uses the server session to put messages in the session of the JMS.
* Server session is the one that spawns the JMS session.
* Applications written by Application programmers creates the message listener.