**JMS:**

**Connection:**Astrazione della connessione TCP/IP.  
Dopo il costoso handshake client e server collegano tra loro in modo simmetrico i loro socket (stream bufferizzati io) e tale collegamento rimane attivo fino a che non viene interrotto (close di uno dei due).  
 **Session: (thread io su una connection)**Su una connection e’ possibile creare N Session.   
Almeno una e’ sempre necessaria per creare un Producer o un Consumer.  
Per migliorare le performances dell io e’ possibile creare N Session su un’ unica Connection.  
(mentre una Session aspetta dati , un’altra Session puo’ spedire/ricevere)

**Producer:**

**Consumer:**

TRANSAZIONI:

ACK:

**Redelivery (Redelivery, messaggio marcato come non ricevuto da nessun Consumer)**Il Broker mette un messaggio in redelivery quando (in un Consumer):

1. A transacted session is used and **rollback()** is called.   
   ( Caso di ED)
2. A transacted session is closed before **commit()** is called.  
   (caso ED viene chiuso prima di completare ricezione + ack.)  
   onMessage e’ cominciato ma la connessione viene chiusa prima di dare commit.
3. A session is using **CLIENT\_ACKNOWLEDGE** and **Session.recover()** is called.
4. A client connection times out (perhaps the code being executed takes longer than the configured time-out period).  
   Simile a 2)

**Code review:**Consumer possibilita:  
ActiveMQMessageConsumer.dipatch

**PERFORMANCES:  
  
Riutilizzo oggetti costosi:**Connection, Session e Consumer richiedono una (o piu) chiamata di rete per essere create.  
Di conseguenza e’ necessario riutilizzarle il piu possibile.  
Gli oggetti Producer aggiungono un piccolo overhead per cui possono essere creati ogni volta (anche se sarebbe meglio riutilizzarli)

**In ambito JSE:**E’ necessario utilizzare un ConnectionPool ed eventualmente gestire da codice una cache di Session riutilizzabili.  
  
**in un Container JEE:**  
L’accesso alle risorse JMS passa da una CF JCA gestita dall AS che si occupa della cache di risorse, per cui le chiamte createSession(), createConnection() verranno in realta’ indirizzate ad una cache.

**Gestione comunicazioni a livello protocollo JMS: (transazionalita?)**

generalizzazione client (consumer o producer)

When the JMS client connects to the JMS broker over the network there are three phases to any API call.

1. The API call, including any message data, is transmitted over the wire to the broker.
2. The API call is executed by the broker.
3. The result code and any message data is transmitted back to the client.

**Consider the producer** for a minute.   
If the connection is broken in the first step then the broker never got the message and the app would need to send it again.   
If the connection is broken in the third step then the message has been successfully sent and sending it again would produce a duplicate message.   
The app cannot tell the difference between these and so the only safe choice is to resend the message on error. If the session is transacted the message can be safely resent in all cases because if the original had made it to the broker, it will be rolled back.

**Consider the consumer**.   
If the connection is lost in the third step then the message is deleted from the queue but never made it back to the client.   
But if the session is transacted the message will be redelivered when the application reconnects. (not acked)

Outside of transactions there is the possibility of lost or duplicate messages. Inside of a transaction the same window of ambiguity exists but it is on the COMMIT call rather then the PUT or GET.   
With transacted sessions it is possible to send or receive a message twice but not to lose one.

The JMS spec recognizes this window of ambiguity and provides the following guidance:

If a failure occurs between the time a client commits its work on a Session and the commit method returns, the client cannot determine if the transaction was committed or rolled back. The same ambiguity exists when a failure occurs between the non-transactional send of a PERSISTENT message and the return from the sending method.

It is up to a JMS application to deal with this ambiguity. In some cases, this may cause a client to produce functionally duplicate messages.

A message that is redelivered due to session recovery is not considered a duplicate message.

JMS sessions should always be transacted except for cases where it really is OK to lose messages.   
If the sessions are transacted then you'd need session and connection per-thread due to the JMS thread model.

**da verificare con lo specific MOM**  
Any advice about performance impacts would be vendor-specific but in general persistent messages outside of syncpoint are hardened to disk before the API call returns.  
But a transacted call can return before the persistent message is written to disk *so long as the message is persisted before the COMMIT returns*. If the vendor optimizes based on this, then it is much more performant to write several messages to disk and then commit them in batches. This allows the broker to optimize writes and disk flushes by disk block rather than per-message. The number of messages to put in the transaction decreases with the size of the message and beyond a certain message size dwindles back down to one.

If your 20k messages are relatively small (measured in k and not mb) then you probably want to use transacted sessions per thread and tune the commit interval.

**DEMO:**

**Note:**Le classi di test devono essere interrotte a mano perche’ la ConnectionFactory ha un TP di  
connections da gestire che previene la chiusura del metodo main.

**Note Avanzato:**  
Sembra che se ci sono N consumer in polling, anche se non consumano, alcuni messaggi  
vengano riservati per loro.   
E’ possibile riprodurlo con consumer in polling che non partono, ed uno in ED che consuma solo parte dei messaggi.  
Ad un Restart di entrambi sembra che i messaggi riservati (non essendo stati acked) vengano poi ripresi dall unico consumer successivamente.

**SJMS-100-ClientJSE:**

**TC: Eccezioni, session.commit e autoack**

* Code: Session.commit
  + autoack=true
  + autoack=false
* eccezione prima di commit
  + autoack=true
  + autoack=false
* eccezione RuntimeException
  + spedire messaggio vuoto (dopo N corretti)
    - JmsConsumer.validateMessage (va in eccezione)

**SJMS-200-ClientSpring:**

* Stessi TC precedenti. Enfasi su Config
* JndiTemplate
* JmsTemplate
  + panoramica api
  + MessageConverters (impl da iniettare in JmsTemplate
* SessionAwareMessageListener

**NOTE:  
Sia nel JmsTemplate che nel< jms:listener> e’ necessario specificare che si tratta di un Topic, altrimenti andra’ in default su Queue.  
Vedi files evdriven-test-activemq05.xml e   
common-topic-cfg-activemq.xml**

**Classi Spring JMS:**