

# JMS Java Messaging Service

\* Ricorda di attivare activeMQ da terminale:

## Avvio applicazioni JMS



1. Avvio ActiveMQ
    - cd /<ACTIVEMQ\_PATH>/bin
    - ./activemq start
  2. Avvio subscriber (receiver) e publisher (sender)
    - cd /<PROJECT\_PATH>/bin
    - java -cp "<ACTIVEMQ\_PATH>/activemq-all-X.Y.Z.jar;" package\_name.class\_name
- Note:
  - <ACTIVEMQ\_PATH> è il path dove è stato estratto il contenuto del file compresso con ActiveMQ
  - In activemq-all-X.Y.Z.jar le lettere X.Y.Z vanno sostituite con la versione utilizzata.
    - Per ActiveMQ 5.16.6 (utilizzato al corso) il nome del file è activemq-all-5.16.6.jar
  - Attenzione ad includere il separatore ":" per i sistemi UNIX-based e ";" a valle del path al file menzionato al punto precedente
  - Gli step al punto 2, vanno ripetuti in diversi terminali, uno per ogni entità da avviare (cioè un terminale per ogni subscriber/receiver ed un terminale per ogni publisher/sender)
  - Il file activemq-all-5.16.6.jar è da includere anche nel classpath del progetto in Visual Studio Code, come *referenced library*

49

Questo è fondamentale anche per il riconoscimento delle librerie javax.jms

## Lookup dei Servizi (sia Client che Server):

Entrambi i file .java dovranno presentare una configurazione iniziale che è sempre "identica":

```
import java.util.Hashtable;
import javax.jms.*;
import javax.naming.*;

Hashtable<String, String> prop = new Hashtable<String, String> ();

prop.put( "java.naming.factory.initial",
"org.apache.activemq.jndi.ActiveMQInitialContextFactory" );
prop.put( "java.naming.provider.url", "tcp://127.0.0.1:61616" );

// jndi-queue-name, physical-queue-name
prop.put( "queue.test", "mytestqueue" );

try{

    Context jndiContext = new InitialContext(prop);
```

← **Attenzione:**  
Va nel main del "Server" e del "client"

## Caso Point-to-Point:

+ Andiamo ad utilizzare le queue  
• Server (Receiver):

- + File main con configurazione (e set del listener se presente)
- + File thread:
  - Metodo run()
  - Costruttore → a cui si possono passare messaggio, coda, queueConnection

```
QueueConnection queueConn = queueConnFactory.createQueueConnection();
queueConn.start(); //abilita il delivery dei messaggi!

QueueSession queueSession = queueConn.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
QueueReceiver receiver = queueSession.createReceiver(queue);

TextMessage message;

do{
    System.out.println ("In attesa di messaggi!");
    message = (TextMessage)receiver.receive();
    System.out.println (" + messaggio ricevuto: " +
    message.getText());
}while (message.getText().compareTo("fine") != 0);

receiver.close();
queueSession.close();
queueConn.close();
```

+ Listener (Vedere implementazione generale)

## • Client (Sender):

- + Listener

- + Main Client:

- Configurazione

- Creazione Struttura messaggi ed invio.

```
QueueSender sender = queueSession.createSender(queue);
TextMessage message = queueSession.createTextMessage();

for ( int i =0; i<5; i++){
    message.setText("hello_" + i);
    sender.send( message );
}

message.setText("fine");
sender.send( message );

System.out.println ("I messaggi sono stati inviati!");

// clean up risorse
sender.close();
queueSession.close();
queueConn.close();
```

## • Coda:

- + Interfaccia (ICoda e Wrapper)

- + Implementazioni: (fissa la classe CodaCircolare)

- CodaWrapperLock

- CodaWrapperSem

- CodaWrapperSynch

## Caso Publisher - Subscriber:

- + Utilizzeremo i Topic

- Publisher:

- Main con:

- Configurazione

- Creazione del publisher a cui passare il topic

- Creazione e "pubblicazione" dei messaggi

```
TopicConnectionFactory connFactory = (TopicConnectionFactory)
jndiContext.lookup("TopicConnectionFactory");

Topic topic = (Topic)jndiContext.lookup("test");

TopicConnection topicConn = connFactory.createTopicConnection();
TopicSession topicSession = topicConn.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);

TopicPublisher pub = topicSession.createPublisher(topic);
TextMessage text = topicSession.createTextMessage();

for ( int i =0; i<5; i++){
    text.setText("hello_" + i);
    pub.publish( text );
}

text.setText("fine");
pub.publish( text );

System.out.println ("I messaggi sono stati inviati!");

// clean up risorse
pub.close();
topicSession.close();
```

## • Subcriber:

- + Spetieci figli thread (pensiamo a TExecutor e TManager)
- + Listener
- + Main contenente:
  - o Configurazione
  - o Utilizzo Topic e TopicConnection
  - o Start threads

```
TopicConnectionFactory connFactory = (TopicConnectionFactory) jndiContext.lookup("TopicConnectionFactory");
Topic topic = (Topic) jndiContext.lookup("test");
TopicConnection topicConn = connFactory.createTopicConnection();
topicConn.start();
TopicSession topicSession = topicConn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
TopicSubscriber sub = topicSession.createSubscriber(topic);
TextMessage msg;
do{
    System.out.println ("In attesa di messaggi!");
    msg = (TextMessage)sub.receive();
    System.out.println (" + messaggio ricevuto: " + msg.getText());
} while (msg.getText().compareTo("fine") != 0);
sub.close();
topicSession.close();
topicConn.close();
```

## • Coda:

- + Interfaccia
- + Coda Circolare

## LISTENER:

Da utilizzare solo in caso di comunicazione asincrona ed ha un costruttore (caso PS in cui gli si passa l'interfaccia Coda) e solo il metodo onMessage (Message m):

```
public class TextMsgListener implements MessageListener {
    public void onMessage ( Message m ){
        try{
            System.out.println (" + messaggio ricevuto: " +
                ((TextMessage)m).getText());
        }catch ( JMSException e ){
            e.printStackTrace();
        }
    }
}
```

## Esempio di Receiver con Listener:

```
QueueReceiver receiver = queueSession.createReceiver(queue);
TextMsgListener msglistener = new TextMsgListener ();
receiver.setMessageListener( msglistener );
```

Ricorda sempre questo ordine:

Dopo la configurazione:

Point - To - Point: Aritoponre Queue

1. Context
2. CommFactory
3. Queue (risposta o richiesta)
4. QueueConnection = commFactory.create...
- 4.1 (Receiver) QueueConnection.start()

5. QueueSession

6. QueueSender o QueueReceiver

Sender:

7. TextMessage

8. Send

Receiver:

7. MyListener

8. Receiver.setMessageListener

Publisher - Subsriber: Aritoponre Topic

1. Context

2. CommFactory

3. Topic

4. Connection

4.1 per il Subsriber: comm.setClientID("DurableService");  
5 Session

6 Publisher o Subsriber:

Publisher: Subsriber

7. Publisher

8. MapMessage\*

9. (Close) varie.

7. DurableSubsriber

8. Listener

9. Start

\* Si usa SetString per  
assegnare una chiave al  
valore da pubblicare mentre  
SetStringProperties quando  
al posto della chiave  
abbiamo una "proprietà" con  
il nome!

## jMS e comunicazione indiretta

### + Comunicazione indiretta:

- Comunicazione tra entità distribuite di un sistema mediante intermediario.
- Dall'intermediario dipende l'approccio comunicativo.

### + MOM:

- Message Oriented Middleware.
- Basati su code o approcci Publish-Subscribe.
- Distributed event based Systems.
- Può monitorare i messaggi inviati.
- Comunicazione asincrona.
- Forniscono i meccanismi di disaccoppiamento spaziale e temporale (garanzie per sistemi distribuiti)

### + Pattern Observer:

- Comunicazione indiretta per disseminazione di eventi
  - Evento: Condizione rivelata da un'applicazione
  - Notifica: Atto di informare di un evento
- Observer si sottoscrive ad un soggetto che lo notifica degli eventi di suo interesse
- Permette l'accoppiamento uno (Subject) - a - molti (observer)
- → Più conveniente, ora si usa il **Notification Service**

### + jMS:

- Definisce uno standard per la comunicazione tramite MOM
- Provider dei servizi jMS: **Active MQ**
- Disaccoppiamento Client - Provider

#### + Domini dei Messaggi:

- Point-to-Point (Code)
- Publish-Subscriber (Topic)

### + Interfacce jMS:

- **ConnectionFactory**: Necessaria alla creazione di una **Connection**
- **Destination**: Destinazione messaggi o Coda / Topic da cui devono arrivare.

### + Abstract Factory:

- Design pattern Creazionale
- Interfaccia per la creazione di famiglie di oggetti
- Componenti:
  - **AbstractFactory**: interfaccia per i metodi di creazione degli oggetti
  - **ConcreteFactory**: implementazione specifica di ↗
  - **AbstractProduct**: interfaccia del tipo oggetto
  - **ConcreteProduct** implements ↗

administrated  
objects

## JMS nel "concreto"

### + Le entità:

- Client
- Provider
- Messaggi
- Administered Object: Posti nel **JNDI**, viene fatto il lookup dal client per recuperarli

### + JNDI:

- Naming Service che mantiene diversi insiemi di Binding Nome - Oggetto.
- Componenti:
  - Nome : nome oggetto
  - Binding: associazione nome - oggetto
  - Reference: puntatore oggetto
  - Context: insieme delle associazioni
- Service Providers Plug-In
- bisogna configurare il Factory Context e l'Url provider

### + Modello di programmazione JMS:

1. lookup degli administered objects.
2. Utilizzo ConnectionFactory per creazione Connection  
2.1 Utilizzo Connection per creazione Session.
3. Creazione MessageProducer, Consumer e Message tramite Session.
4. Cleanup risorse.

### + Receiver:

- Si occupa della ricezione dei messaggi, può essere:
  1. Sincrona
  2. Asincrona
- 2.1 In tal caso bisogna implementare l'istanza listener

### + Messaggi JMS:

- Componenti:
  1. header
  2. Properties
  3. Body

→ {  
• Text  
• Byte  
• Stream  
• Object

### • Acknowledgment:

1. Auto
2. Client
3. Dups

### + Transazioni:

- Nel contesto di una singola sessione permettono di raggruppare più operazioni JMS

### + Persistenza messaggi:

- Persistent (default)
- Non - Persistent

### + Durable Subscriber:

- Non duraturo di default

- Duraturo riconoscendolo con la coppia clientID + nome sottoscrizione.

↳ Bisognerà usare il metodo:

TopicSession sub = topicSession.createDurableSubscriber(topic, "nome.univoco.sottoscrizione")

## Annotazioni da Esercizi

### Topic con messaggi Asincroni e Servizio esterno tramite comunicazione UDP

#### + Server Side:

- Proxy:

1. Costruttore a cui passare la port
2. Override dei metodi dell'interfaccia remota (ne implementiamo la comunicazione)
  - 2.1 Crea DatagramSocket
  - 2.2 Stringify dato da inviare
  - 2.3 DatagramPacket con host locale e port del costruttore
  - 2.4 Send
  - 2.5 Crea buffer che conterrà la risposta
  - 2.6 DatagramPacket della risposta a cui passare buffer e buffer.length
  - 2.7 Receive
  - 2.8 String oms a cui passare parametri della risposta

- Thread:

1. Fa ciò che la traccia chiede
2. Utilizza i metodi remoti tramite proxy

- Listener:

1. Implementa metodo onMessage(Message m) ed avvia il thread a cui passare il MapMessage

- Main:

1. Guardare Configurazione del receiver sopra!

#### + Client - Side:

- Main:

1. Stessa implementazione, ma del sender