

**Licence en génie logiciel et systèmes d'information**  
**Niveau: 3<sup>ème</sup> année**  
**Semestre 1**

**Chapitre 6**  
**Les middlewares orientés messages**

Dhikra KCHAOU  
dhikrafsegs@gmail.com

# Plan du chapitre

---

1. Introduction
2. Présentation de la communication distribué asynchrone
3. JMS: Java Message Service
4. Modèle de programmation JMS
5. Implémentation du modèle JMS

# Introduction

---

## ► Communication distribuée synchrone:

- Un objet A dans une machine M1 établit une connexion avec un objet B dans une machine M2.
- Si l'objet B n'est pas disponible, il n'y aura pas de communication.
- Si B est disponible, l'objet A envoie une requête pour faire appel à une opération à distance de l'objet B.
- L'objet A est bloqué jusqu'à qu'il reçoit la réponse de l'objet B.



# Problèmes d'une communication synchrone

---

## ☹ Communication distribuée synchrone:

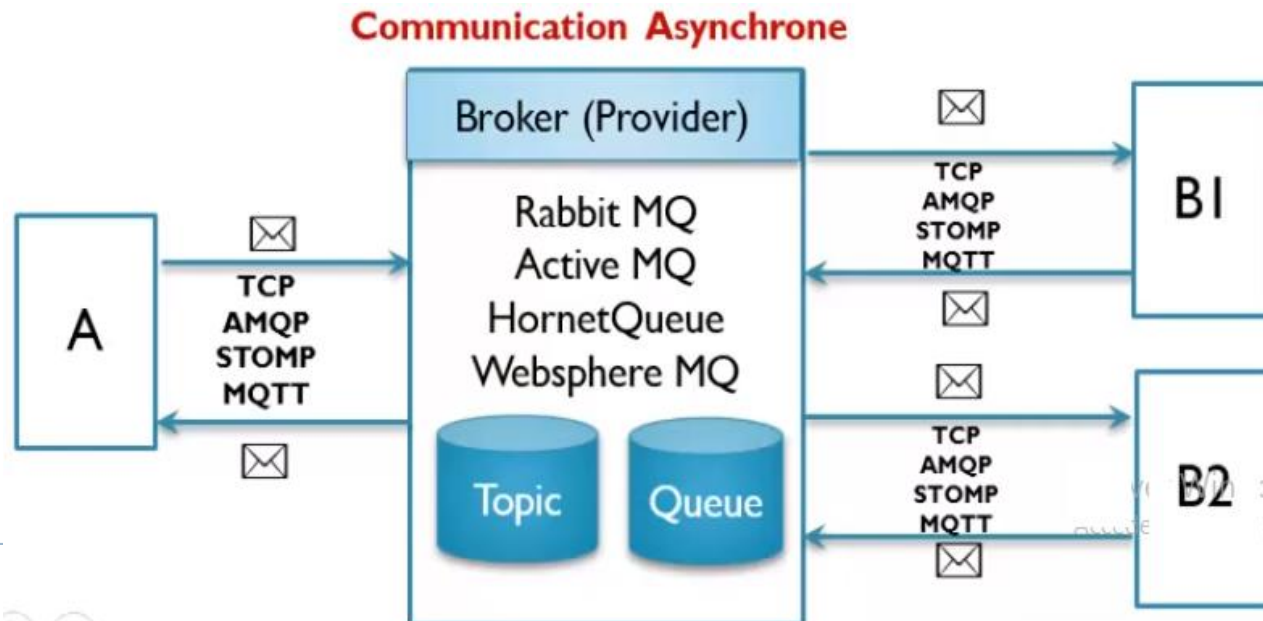
- ▶ Dépendance des applications
- ▶ Connexion permanente entre des 2 parties
- ▶ Blocage de communication

## 😊 **Communication distribuée asynchrone**

- ▶ Qualité de service
- ▶ Système distribué performant (calcul distribué de haute performance)

# Communication distribué asynchrone

- ▶ Dans une communication distribué asynchrone:
  - ▶ Un objet A établit une connexion vers un **provider (Broker)**.
  - ▶ L'objet A envoie le message à une file d'attente du broker pour le délivrer à l'objet B.
  - ▶ Si l'objet B n'est pas disponible, cela n'empêche pas l'objet A d'envoyer le message qui sera conservé dans la file d'attente jusqu'à B se connecte.
  - ▶ Si B est connecté à la file d'attente du Broker, ce dernier lui délivre le message.

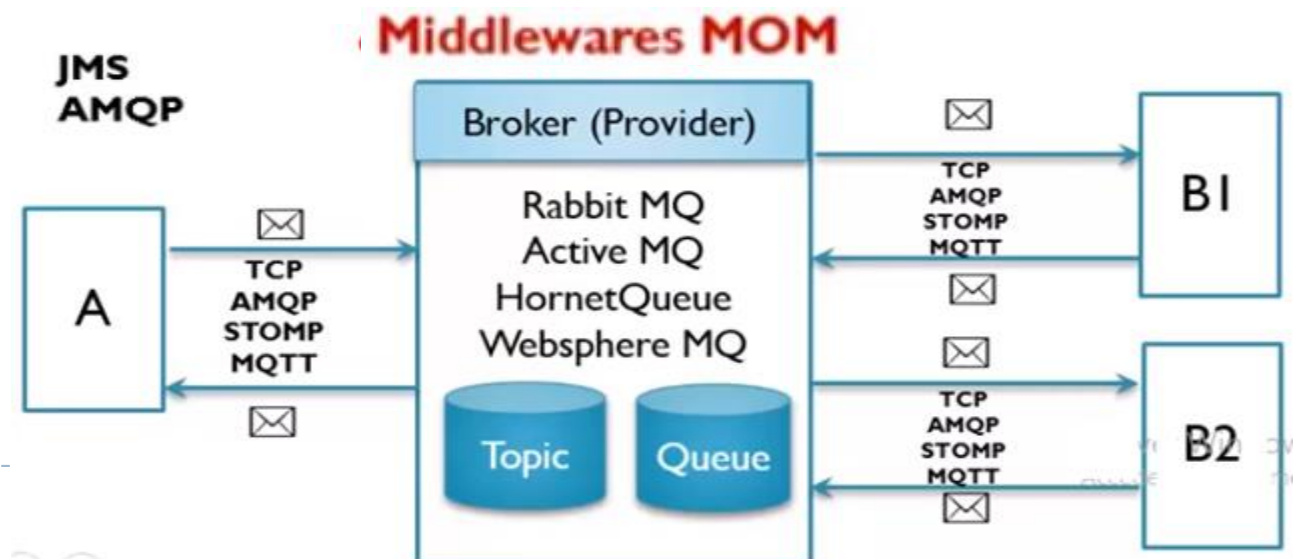


# Communication distribué asynchrone

- ▶ Il est possible que plusieurs objets B soient connectés à la même file d'attente.
  - ▶ Dans le cas d'une file d'attente de type **queue**, le message est délivré à un seul client en utilisant le principe de Round Robin.
  - ▶ Dans le cas d'une file d'attente de type **Topic**, le message peut être délivré à plusieurs clients B.
- ▶ Une fois le message est consommé, il est supprimé de la file d'attente.
- ▶ Il est possible de demander au broker de conserver les messages d'une manière persistante.

- ▶ Modèles de communication:

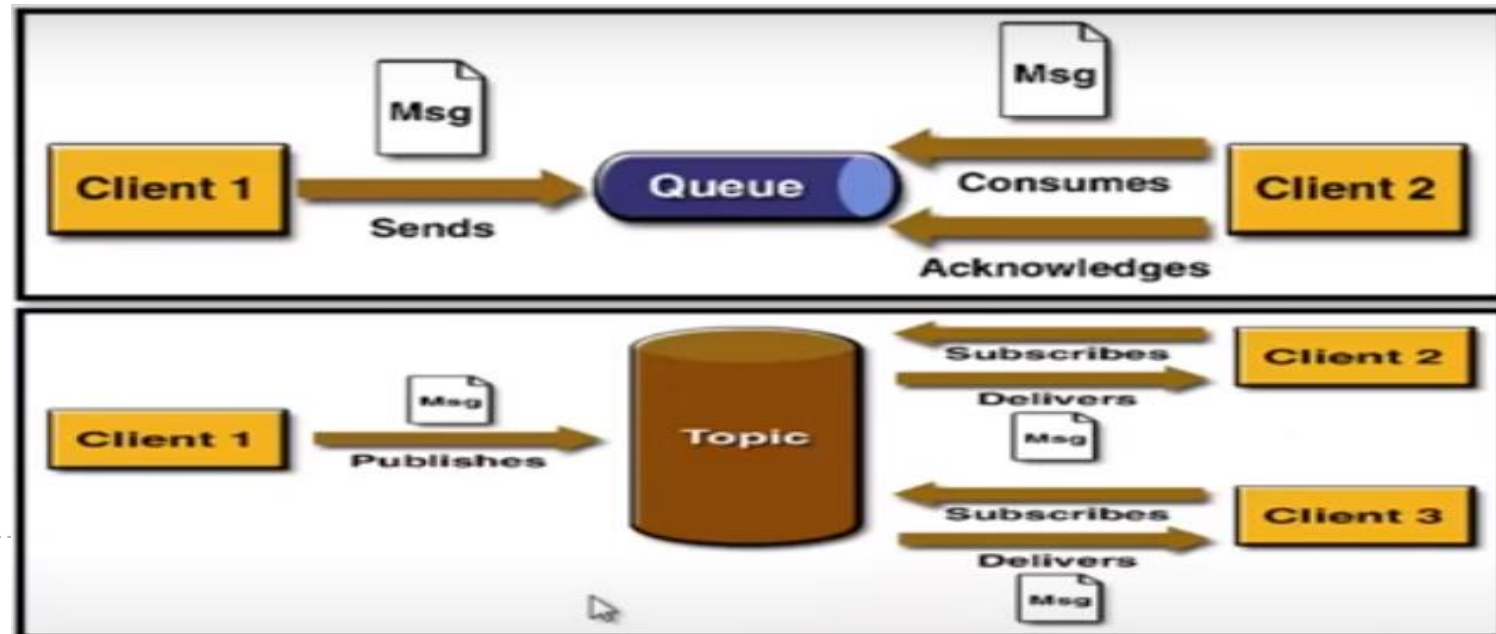
- ▶ **JMS**
- ▶ AMQP
- ▶ STOMP
- ▶ MQTT



# **JMS: Java Message Service**

# JMS: Java Message Service

- ▶ JMS est une **API** d'échange de messages pour permettre un dialogue entre applications via un fournisseur (Provider ou Broker) de messages.
- ▶ L'application cliente envoie un message dans une file d'attente sans se soucier de la disponibilité de cette application .
- ▶ Le client a, de la part du fournisseur de messages, une garantie de qualité de service (certitude de remise au destinataire, délai de remise, etc.)





# Définition des messages

---

- ▶ Les messages sont un moyen de communication entre les composants ou les applications.
- ▶ Les messages peuvent être envoyés par n'importe quel **composant**

## **J2EE:**

- ▶ Les applications clientes (java)
- ▶ Des composants EJB, Spring
- ▶ Des composants Web (Servlet, Web service)
- ▶ Etc.

# L'API Java Message Service

---

- ▶ JMS est une API, spécifiée en 1998, pour la création, l'envoi et la réception des messages de façon:
  - ▶ **Asynchrone**: un client reçoit les messages dès qu'il se connecte ou lorsqu'il est disponible (et sans avoir à demander si un message est disponible).
  - ▶ **Fiable**: le message est délivré une et une seule fois. Une fois le message est délivré, le broker va le supprimer de la file d'attente.

# Comment fonctionne JMS avec J2EE?

---

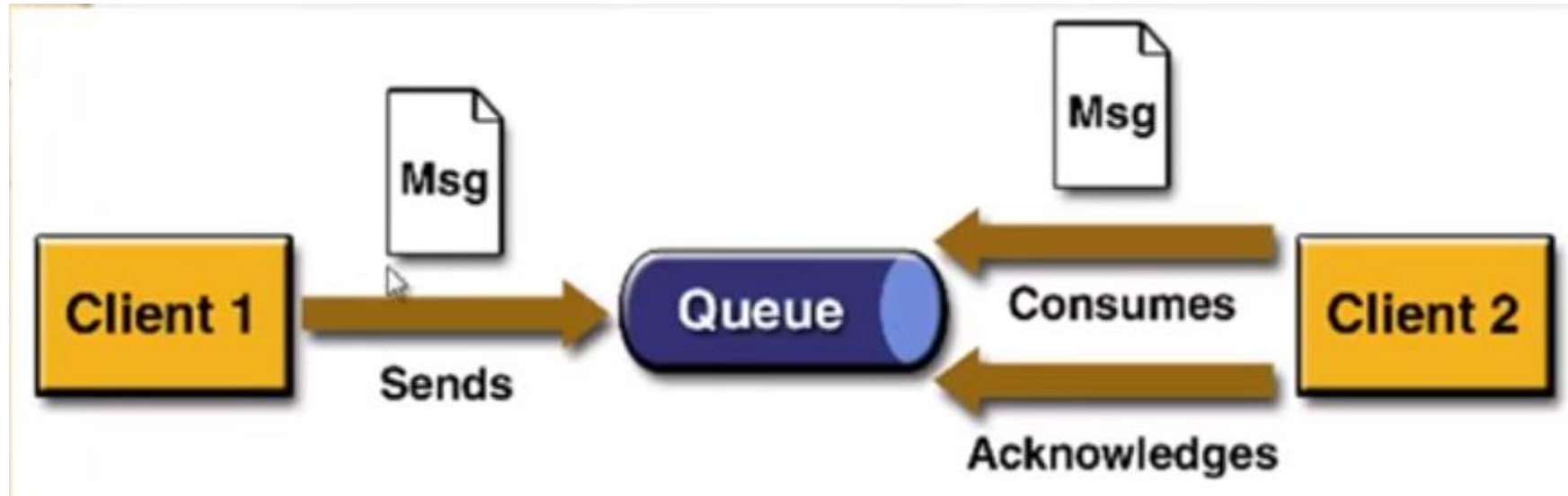
- ▶ JMS peut accéder aux **Message Oriented Middleware (MOM)** tels que:
  - ▶ MQSeries d'IBM,
  - ▶ HornetQueue de Jboss
  - ▶ **ActiveMQ de Apache**
  - ▶ One Message Queue de Sun Microsystem,...
- ▶ JMS fait partie intégrante de J2EE à partir de la version 1.3.

## Les protocoles de communication

---

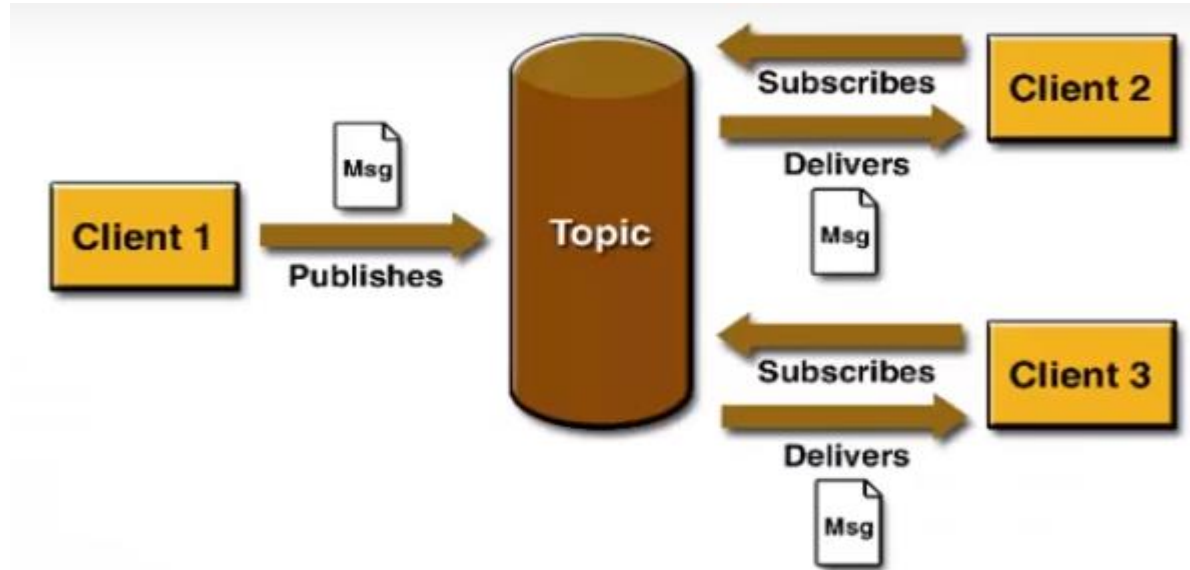
- ▶ Avant l'arrivée de JMS, les produits proposaient une gestion de message selon deux types de protocoles:
  - ▶ Un protocole **point à point** (Queue)
  - ▶ Un protocole **publier /souscrire** (Topic)
- ▶ Les JMS Broker implémentent obligatoirement ces deux protocoles.

# Le protocole Point-à-Point (Point to point)



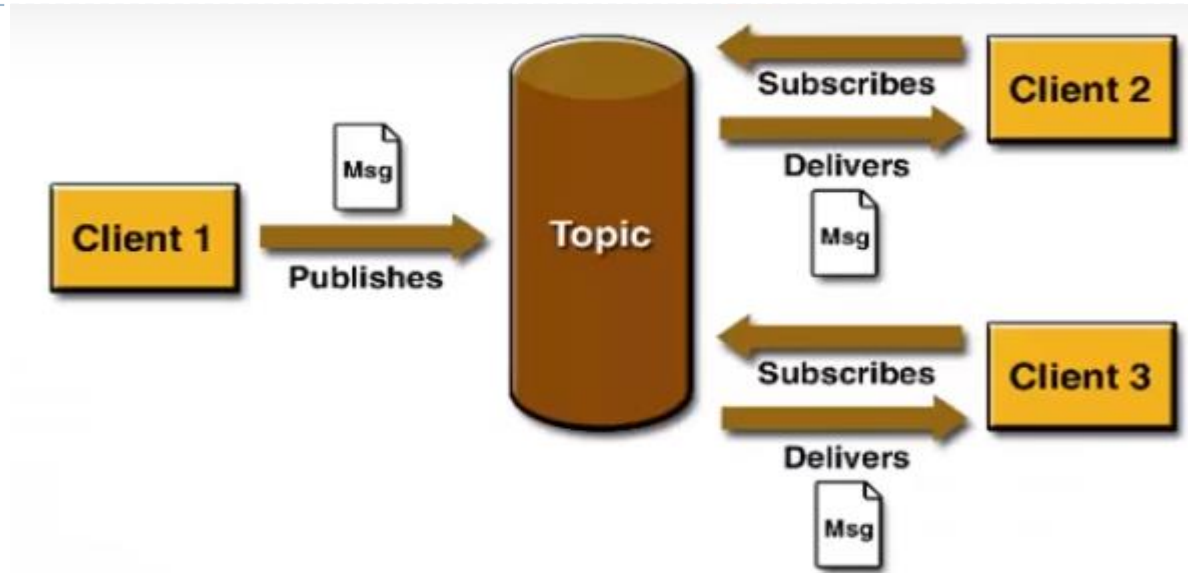
- ▶ Ce protocole est basé sur le concept de queue de messages, d'émetteurs et de receveurs.
- ▶ Chaque message est adressé à une queue spécifique et les clients consomment leurs messages.
- ▶ Ce protocole doit être utilisé pour s'assurer qu'un seul client consommera le message.

# Le protocole publier/souscrire (publish/subscribe) (Topic)



- ▶ Les clients sont anonymes et adressent les messages.
- ▶ Le système distribue le message à l'ensemble des souscripteurs puis l'efface.
- ▶ Ce protocole est efficace lorsque le message doit être envoyé à zéro, un ou plusieurs clients.

# Le protocole publier/souscrire (Topic)



- ▶ Il existe deux types de souscription: temporaire et durable:
  - ▶ Dans le cas d'une **souscription temporaire**: les consommateurs reçoivent les messages **tant qu'ils sont connectés au sujet**.
  - ▶ Dans le cas d'une **souscription durable**: on oblige le fournisseur (Broker) **à enregistrer les messages** lors d'une déconnexion, et à les envoyer lors de la nouvelle connexion du consommateur.

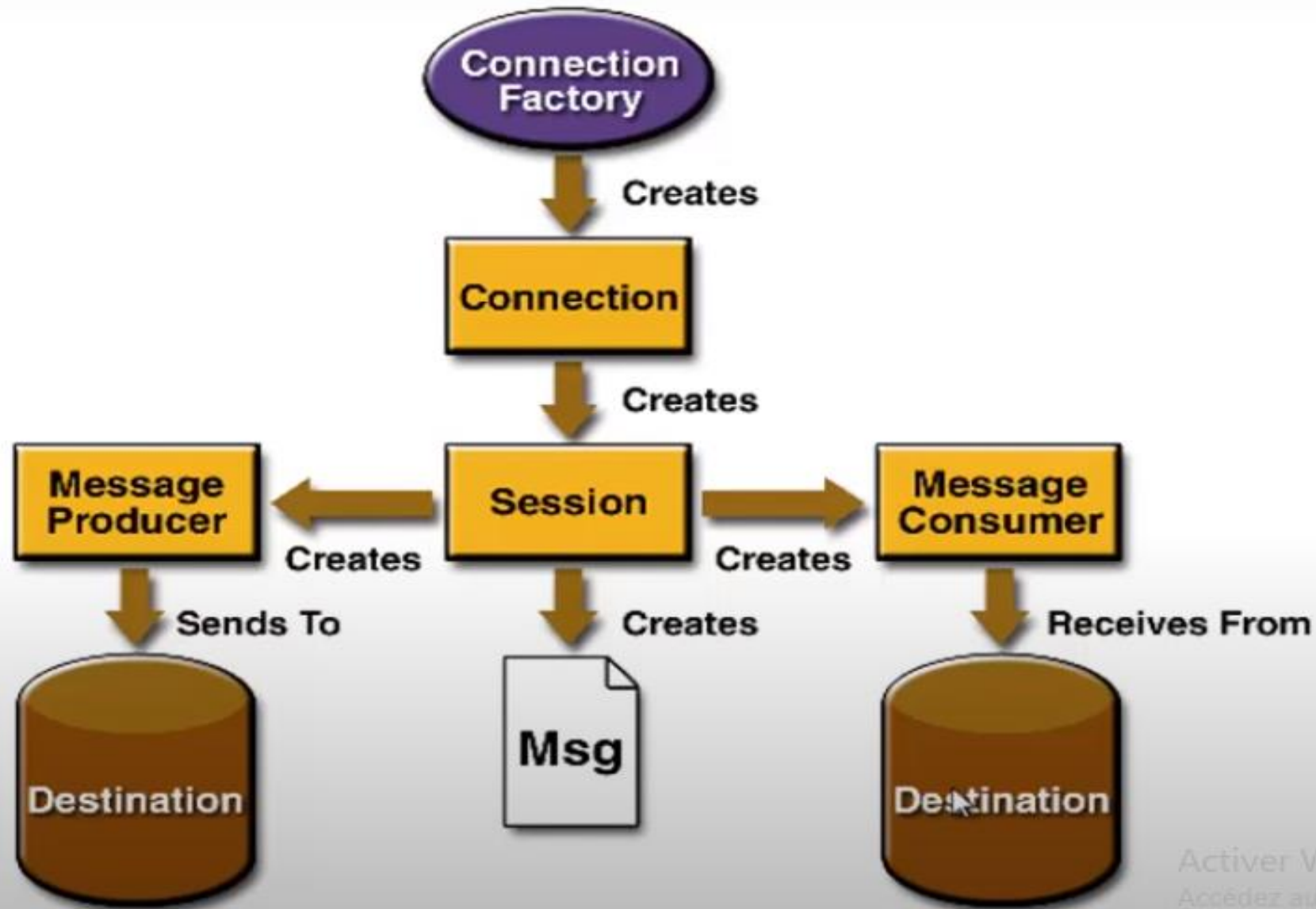
# La consommation de message

---

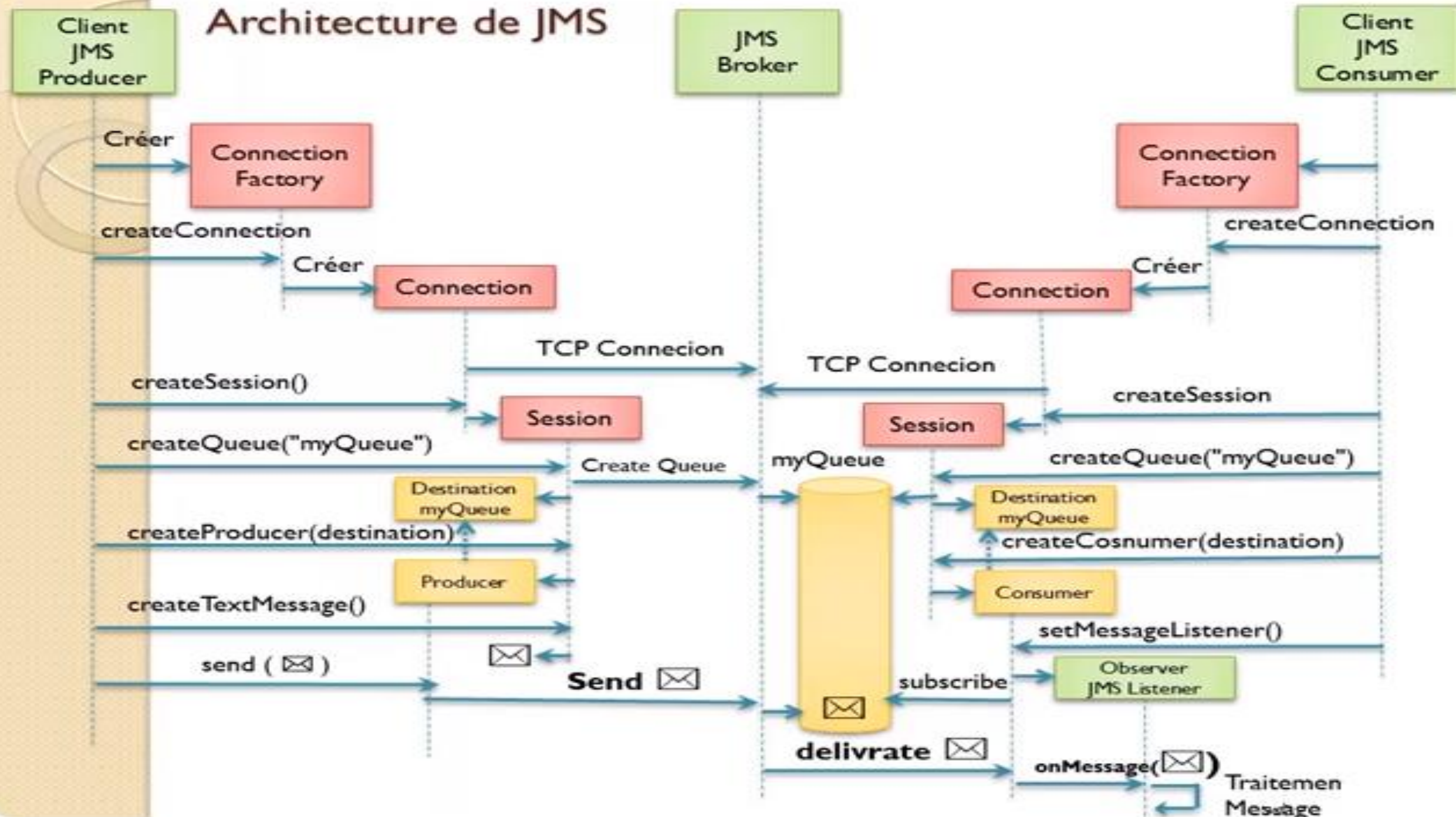
- ▶ Le client consomme le message de façon:
  - ▶ **Synchrone**: en appelant la méthode **receive** qui est bloquante jusqu'à ce qu'un message soit délivré, ou qu'un délai d'attente soit expiré.
  - ▶ **Asynchrone**: un client peut s'enregistrer auprès d'un écouteur de message (**listener**). Lorsqu'un message arrive, JMS le délivre en invoquant la méthode **onMessage** du listener, lequel agit sur le message.



# Le modèle de programmation JMS



# Architecture de JMS



## Exemple de code pour un producer JMS: Producer.java

```
try {  
    //Create a Connection Factory  
    ConnectionFactory connectionfactory = new ActiveMQConnectionFactory("tcp://localhost:61616");  
    //Create a Connection  
    Connection connection=connectionfactory.createConnection();  
    //Start the connection  
    connection.start();  
    //Create the session Object  
    Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);  
    //Create The destination (Topic or Queue)  
    Destination destination=session.createQueue("enset.queue");  
    //Create a Message Producer from the session to the Topic or Queue  
    MessageProducer producer = session.createProducer(destination);  
    //Delivrate the message without conserve  
    producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);  
    //Create the message and send it  
    TextMessage textmessage=session.createTextMessage();  
    textmessage.setText("Hello...");  
    producer.send(textmessage);  
    // Close Objects  
    session.close();  
    connection.close();  
} catch (JMSException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();}}
```

# Exemple de code pour un consumer JMS: Consumer.java

//Create a Connection Factory

```
ConnectionFactory connectionfactory= new ActiveMQConnectionFactory("tcp://localhost:61616");//
```

spécifier l'adresse ou se trouve le Broker

//Create a Connection

```
Connection connection=connectionfactory.createConnection();
```

//Start The connection

```
connection.start();
```

//Create the session

```
Session session= connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
```

//Create The destination (Topic or Queue)

```
Destination destination=session.createQueue("enset.queue");
```

//Create the Message Consumer from the session to the Topic or Queue

```
MessageConsumer consumer=session.createConsumer(destination);
```

//Create JMS Listener form messages

```
consumer.setMessageListener(new MessageListener() {
```

@Override

```
public void onMessage(Message message) {
```

```
if (message instanceof TextMessage) {
```

```
try {
```

```
TextMessage textmessage= (TextMessage) message;
```

```
System.out.println(textmessage.getText());
```

```
} catch (JMSException e) {
```

```
// TODO Auto-generated catch block
```


```
e.printStackTrace();
```

```
}}}});
```

# Broker ActiveMQ

## ► Télécharger ActiveMQ

[←](#) [→](#) [↻](#) <https://activemq.apache.org/components/classic/download/>

 [News](#)

### ActiveMQ Download

These are the current releases. For prior releases, please see the [past releases](#) page.

It is important to [verify the integrity](#) of the files you download.

#### ActiveMQ 6.0.0 (Nov 17th, 2023)

[Release Notes](#) | [Release Page](#) | [Documentation](#) | Java compatibility: **17+**

Windows	<a href="#">apache-activemq-6.0.0-bin.zip</a>	SHA512	GPG Signature
Unix/Linux/Cygwin	<a href="#">apache-activemq-6.0.0-bin.tar.gz</a>	SHA512	GPG Signature
Source Code Distribution:	<a href="#">activemq-parent-6.0.0-source-release.zip</a>	SHA512	GPG Signature

#### ActiveMQ 5.18.3 (Oct 25th, 2023)

[Release Notes](#) | [Release Page](#) | [Documentation](#) | Java compatibility: **11+**



# Démarrage du Broker ActiveMQ

- Activer le broker ActiveMQ en utilisant la commande **activemq start**

```
C:\Users\Administrateur>cd C:\Users\Administrateur\Desktop\FSG 23-24\S1\Dev des app Réparties\chap6-MOM\apache-activemq-6.0.0\bin  
C:\Users\Administrateur\Desktop\FSG 23-24\S1\Dev des app Réparties\chap6-MOM\apache-activemq-6.0.0\bin>activemq start
```

```
Administrateur: Invite de commandes - activemq start  
INFO | Apache ActiveMQ 6.0.0 (localhost, ID:DESKTOP-T2IUBA4-51011-1700817835296-0:1) is starting  
INFO | Listening for connections at: tcp://DESKTOP-T2IUBA4:61616?maximumConnections=1000&wireFormat.maxFrameSize=104857600  
INFO | Connector openwire started  
INFO | Listening for connections at: amqp://DESKTOP-T2IUBA4:5672?maximumConnections=1000&wireFormat.maxFrameSize=104857600  
INFO | Connector amqp started  
INFO | Listening for connections at: stomp://DESKTOP-T2IUBA4:61613?maximumConnections=1000&wireFormat.maxFrameSize=104857600  
INFO | Connector stomp started  
INFO | Listening for connections at: mqtt://DESKTOP-T2IUBA4:1883?maximumConnections=1000&wireFormat.maxFrameSize=104857600  
INFO | Connector mqtt started  
INFO | Starting Jetty server  
INFO | Creating Jetty connector  
WARN | ServletContext@o.e.j.s.ServletContextHandler@6a87026{/ ,null,STARTING} has uncovered HTTP methods for the following paths: [/]  
INFO | Listening for connections at ws://DESKTOP-T2IUBA4:61614?maximumConnections=1000&wireFormat.maxFrameSize=104857600  
INFO | Connector ws started  
INFO | Apache ActiveMQ 6.0.0 (localhost, ID:DESKTOP-T2IUBA4-51011-1700817835296-0:1) started  
INFO | For help or more information please see: http://activemq.apache.org  
WARN | Store limit is 102400 mb (current store usage is 0 mb). The data directory: C:\Users\Administrateur\Desktop\FSG 23-24\S1\Dev des app Réparties\chap6-MOM\apache-activemq-6.0.0\bin\..\data\kahadb only has 7490 mb of usable space. - resetting to maximum available disk space: 7490 mb  
WARN | Temporary Store limit is 51200 mb (current store usage is 0 mb). The data directory: C:\Users\Administrateur\Desktop\FSG 23-24\S1\Dev des app Réparties\chap6-MOM\apache-activemq-6.0.0\bin\..\data only has 7490 mb of usable space. - resetting to maximum available disk space: 7490 mb  
INFO | ActiveMQ WebConsole available at http://127.0.0.1:8161/  
INFO | ActiveMQ Jolokia REST API available at http://127.0.0.1:8161/api/jolokia/
```

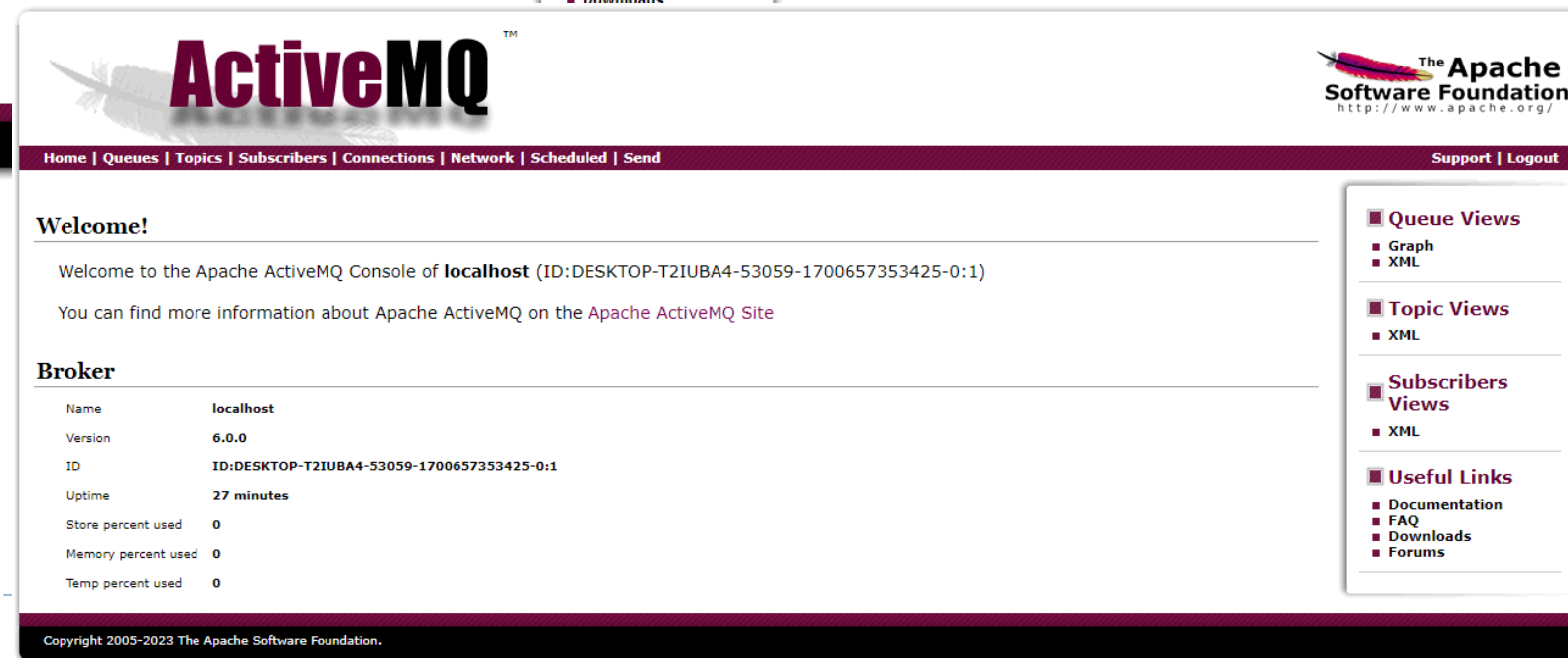
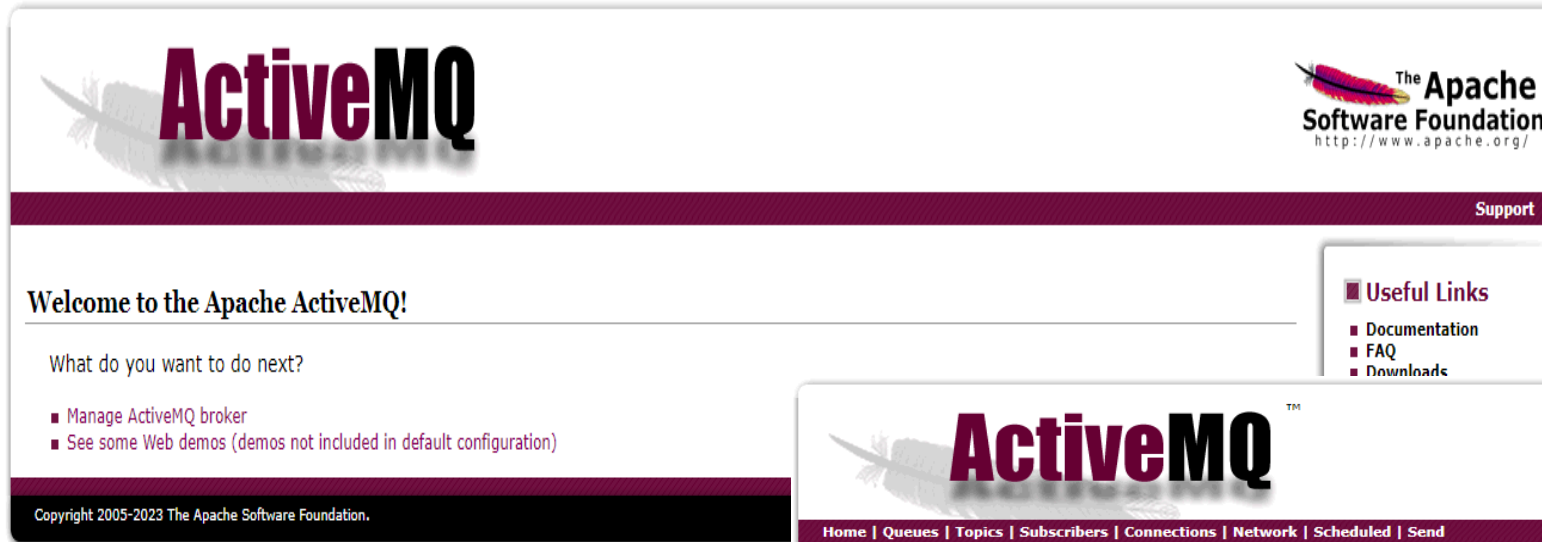
# Interface d'administration d'ActiveMQ

← → ↻ ⓘ http://localhost:8161/index.html



► Le nom utilisateur  
et le mot de  
passe:

**admin**



# Démarrer ActiveMQ à partir d'une application Java:

## Embedded ActiveMQ

```
ActiveMQBroker.java ×
1 package jmsenset;
2
3 import org.apache.activemq.broker.BrokerService;
4
5 public class ActiveMQBroker {
6
7     public static void main(String[] args) {
8
9         try {
10             BrokerService broker = new BrokerService();
11             broker.addConnector("tcp://0.0.0.0:61616");
12             broker.start();
13         } catch (Exception e) {
14             // TODO Auto-generated catch block
15             e.printStackTrace();
16         }
17     }
18 }
19 }
```

```
16 <!-- https://mvnrepository.com/artifact/org.apache.activemq/activemq-broker -->
17 <dependency>
18     <groupId>org.apache.activemq</groupId>
19     <artifactId>activemq-broker</artifactId>
20     <version>5.15.2</version>
21 </dependency>
22 <!-- https://mvnrepository.com/artifact/org.apache.activemq/activemq-kahadb-store -->
23 <dependency>
24     <groupId>org.apache.activemq</groupId>
25     <artifactId>activemq-kahadb-store</artifactId>
26     <version>5.9.0</version>
27 </dependency>
```



# Mise en place des composants JMS

---

- ▶ Il existe un certain nombre de composants qui s'occupe de la gestion globale du JMS et qui permettent ainsi une communication asynchrone entre applications clientes: **ConnectionFactory** et **Destination**
- ▶ Si vous travaillez dans le cadre d'un serveur d'application (Jboss, Glassfish, HornetQueue), la première étape consiste d'abord à se connecter au fournisseur JMS, pour cela, nous devons:
  - ▶ Récupérer un objet **ConnectionFactory** via JNDI qui rend la connexion possible avec le fournisseur (une ConnectionFactory fournit une connexion JMS au service de routage de message)
  - ▶ Récupérer l'élément Destination qui représente des messages qui véhiculent les messages, JMS comporte deux types de destinations:
    - ▶ Queue et Topic.

# Connection et Session

---

- ▶ L'objet `ConnectionFactory` permet de créer une connexion (`Connection`) avec le fournisseur JMS.
- ▶ Une fois la connexion est créée, elle est ensuite utilisée pour créer une session.

```
Connection connection=connectionfactory.createConnection();  
Session session= connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
```

- ▶ La méthode `createSession()` prend deux paramètres:
  - ▶ Booléen: indique que la session est transactionnel ou non.
  - ▶ `Session.AUTO_ACKNOWLEDGE`: indique si un accusé de réception doit être renvoyé pour préciser que message est bien arrivé à sa destination

# MessageProducer et MessageConsumer

---

- ▶ La dernière étape consiste à préciser le sens du transfert du message : envoi ou réception du message?
- ▶ Deux objets correspondent à ces deux situations:
  - ▶ **MessageProducer** pour l'envoi du message
  - ▶ **MessageConsumer** pour la consommation du message

```
MessageProducer producer = session.createProducer(destination);
```

```
MessageConsumer consumer=session.createConsumer(destination);
```

- ▶ Chacune des méthodes de l'objet session prend en paramètre la destination sur laquelle l'objet est connecté.

# Message

---

- ▶ Dans JMS, le message est un objet Java qui doit implémenter l'interface `javax.jms.Message` (Jakarta.jms.Message Version Activemq 6.0).
- ▶ Il est composé de trois parties:
  - ▶ L'entête (**header**): qui se compose des informations de destination, d'expiration, de priorité, date d'envoi, etc.
  - ▶ Les propriétés (**properties**): qui représentent les caractéristiques fonctionnelles du message.
  - ▶ Le corps de message (**body**): qui contient les données à transporter.

# Entête du message

## ► Liste des propriétés de l'entête d'un message JMS

Nom	Description
<b>JMSMessageID</b>	Identifiant unique du message
<b>JMSCorrelationID</b>	Utilisé pour synchroniser de façon applicative deux messages de la forme requête/réponse. Dans ce cas, dans le message réponse, ce champ contient le messageID du message requête
<b>JMSDestination</b>	File d'attente destinataire du message
<b>JMSExpiration</b>	Date d'expiration du message
<b>JMSPriority</b>	Priorité du message. Les message de niveau 9 sont plus prioritaire que les messages de niveau 0).
<b>JMSDeliveryMode</b>	Mode d'envoi du message: <b>persistent</b> (le message est délivré une et une seule fois au destinataire même en cas du panne du fournisseur le message sera délivré) et <b>non persistent</b> (le message peut ne pas être délivré en cas de panne).

## Les propriétés d'un message JMS

---

- ▶ Cette section du message est optionnelle et agit comme une extension des champs d'entête.
- ▶ Les propriétés d'un message JMS sont des couples (nom, valeur), où la valeur est un type de base du langage Java (entier, chaîne de caractères, Booléen, etc).
- ▶ L'interface `Javax.jms.Message` définit des accesseurs pour manipuler ces valeurs.
- ▶ Ces données sont généralement positionnées par le client avant l'envoi d'un message, et peuvent être utilisées pour filtrer les messages.

## Corps d'un message JMS

- ▶ Le corps du message, bien qu'optionnel, est la zone qui contient les données.
- ▶ Ces données sont formatées selon le type de messages qui est défini par les interfaces suivantes (qui héritent toutes de `javax.jms.Message`):

Interface	Description
<code>javax.jms.BytesMessage</code>	Les messages sous forme de flux d'octets.
<code>javax.jms.TextMessage</code>	Les messages de type Texte.
<code>javax.jms.ObjectMessage</code>	Les messages composés d'objets java sérialisés.
<code>javax.jms.MapMessage</code>	Les messages sous la forme de clé/Valeur. La clé doit être de type String et la valeur de type primitif.
<code>javax.jms.StreamMessage</code>	Messages en provenance d'un flux.

# javax.jms.BytesMessage

---

## ► Exemple

```
BytesMessage message=session.createBytesMessage();  
message.writeInt(15);  
message.writeDouble(-6.78);  
message.writeBoolean(true);  
envoi.send(message);
```



# javax.jms.TextMessage

---

```
TextMessage message2=session.createTextMessage();
```

```
message2.setText("Bienvenue");
```

```
envoi.send(message2);
```

# javax.jms.ObjectMessage

- ▶ Exemple: une classe serialisable **Personne** :

```
public class Personne implements Serializable {  
}
```

- Code JMS pour envoyer un ObjectMessage qui contient deux objet de type Personne :

```
Personne p1=new Personne();
```

```
Personne p2=new Personne();
```

```
ObjectMessage message3=session.createObjectMessage();
```

```
message3.setObject(p1);
```

```
message3.setObject(p2);
```

```
envoi.send(message3);
```

## javax.jms.MapMessage

- ▶ Ce type de message permet d'envoyer et de recevoir des informations suivant le système clé/valeur. Ainsi, nous retrouvons les mêmes méthodes que pour BytesMessage, mais à chaque fois, nous devons préciser la clé sous forme de chaîne de caractères.
- ▶ Les méthodes sont plutôt des accesseurs getXxx() et setXxx()
- ▶ Exemple:

```
MapMessage message4=session.createMapMessage();  
message4.setInt("code", 210);  
message4.setString("fonction", "ingenieur");  
message4.setDouble("salaire",22000.00);  
envoi.send(message4);
```

## javax.jms.StreamMessage

---

- ▶ Ce type de message est vu comme un flux.
- ▶ Il ressemble aux types DataOutputStream et DataInputStream.
- ▶ Exemple:

```
StreamMessage message5=session.createStreamMessage();  
message5.writeInt(15);  
message5.writeDouble(-6.78);  
message5.writeBoolean(true);  
envoi.send(message4);
```

# Comment envoyer un message

---

- ▶ En résumé, les éléments à mettre en œuvre pour envoyer un message sont:
  - ▶ Tout d'abord la fabrique de connexion (ConnectionFactory) et la destination (Destination) doivent être connus par le client JMS.
  - ▶ Une fois la référence de la ConnectionFactory obtenue, on se connecte au provider (fournisseur) JMS via l'objet Connection.
  - ▶ A partir de cette connexion, nous devons obtenir une session (Session).
  - ▶ A partir de cette session, nous devons créer un MessageProducer qui va permettre d'envoyer des messages auprès d'une destination.
  - ▶ La session permet également de créer le message suivant le type choisi.

## Comment recevoir un message

---

- ▶ Le consommateur du message est le client capable d'être à l'écoute d'une file d'attente (ou d'un sujet), et de traiter les messages à leur réception.
- ▶ En effet, le client doit être constamment à l'écoute (listener) et à l'arrivée d'un nouveau message, il doit pouvoir le traiter.
- ▶ Pour cela l'application doit appeler la méthode `onMessage()` de l'interface `javax.jms.MessageListener`.
- ▶ Cet interface permet la réception asynchrone des messages.
- ▶ Charge au développeur d'implémenter cette interface pour réaliser le traitement adéquat lors de la réception d'un message.

# Comment recevoir un message

---

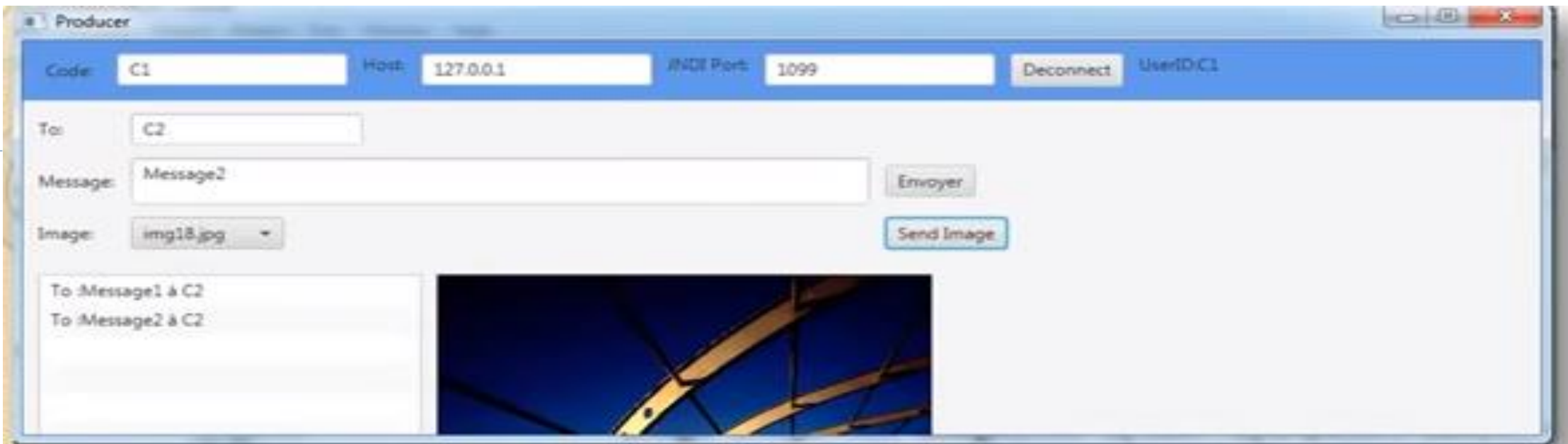
- ▶ La procédure à suivre:
  - ▶ Comme l'envoi de message, la fabrique de connexion (ConnectionFactory) et la destination (Destination) doivent être connues par le client JMS.
  - ▶ Une fois la référence de la ConnectionFactory obtenue, le consommateur doit se connecter au provider (fournisseur) JMS via l'objet Connection.
  - ▶ A partir de cette connexion, nous devons obtenir une session (Session).
  - ▶ A partir de cette session, nous devons créer un MessageConsumer qui va permettre de consommer les messages.
  - ▶ Pour ce faire, nous associons un listener MessageListener pour traiter les messages de façon asynchrone. Ainsi, à chaque réception d'un nouveau message, la méthode onMessage() est automatiquement invoquée et peut effectuer le traitement désiré.
  - ▶ Attention: à ce stade, il ne faut surtout pas oublier de démarrer la connexion avec la méthode start() sinon aucun message ne sera reçu.

# Application

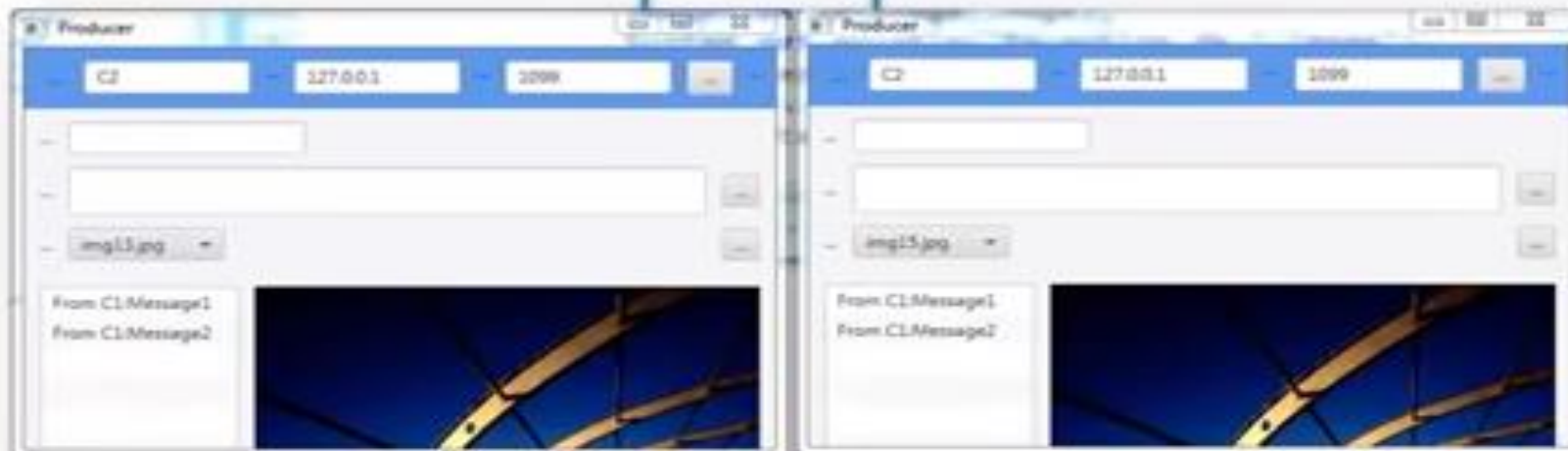
---

- ▶ On souhaite créer une application de chat basée sur JMS qui permet aux différents clients de :
  - ▶ Se connecter au Provider JMS en utilisant un code
  - ▶ Envoyer et recevoir des messages de type Texte
  - ▶ Envoyer et recevoir des messages contenant des images
- ▶ L'interface graphique de l'application est basée sur JavaFX



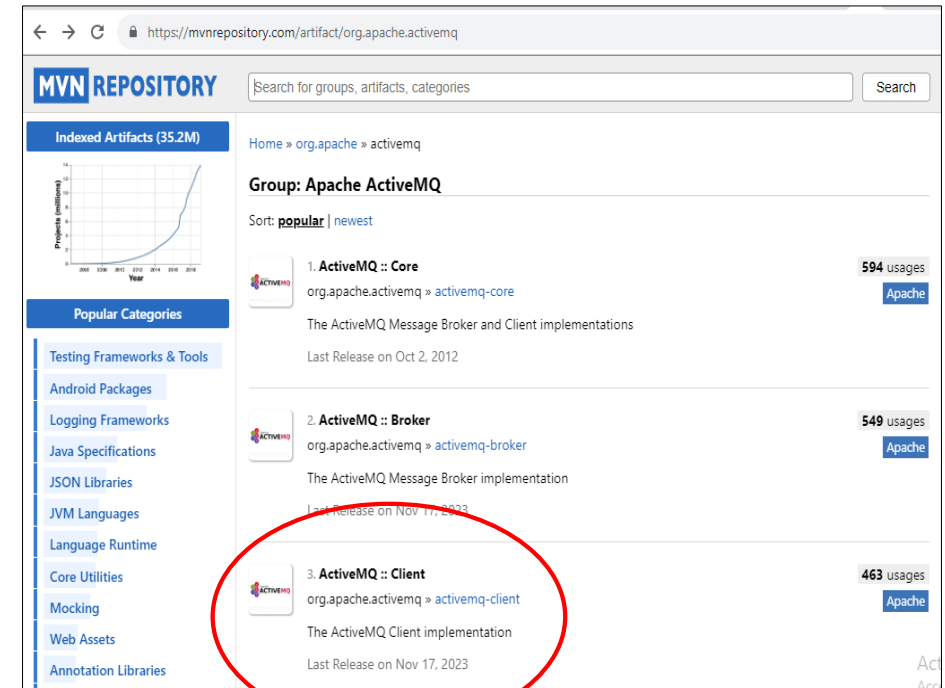


JMS Broker  
HornetQueue



# Création d'un projet Maven

- ▶ Maven est un outil de gestion de projet logiciel pour Java maintenu par l'Apache Software Foundation.
- ▶ Le choix du projet Maven est sa capacité de **gérer les dépendances** de votre projet et automatiser sa construction (compilation, test, production de livrable...).
- ▶ Pour créer une communication asynchrone JMS basée sur ActiveMQ on a besoin d'ajouter l'implémentation de l'API JMS (fourni par le Broker) dans le fichier pom.xml → **Ajout de dépendances**



# Création du producteur et du consommateur

```
Consumer.java X
1 import javax.jms.*;
5 public class Consumer {
6     public static void main(String[] args) {
7         try {
8             //Create a Connection Factory
9             ConnectionFactory connectionfactory= new ActiveMQConnectionFactory("tcp://localhost:61616");
10            //Create a Connection
11            Connection connection=connectionfactory.createConnection();
12            //Start The connection
13            connection.start();
14            //Create the session
15            Session session= connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
16            //Create The destination (Topic or Queue)
17            Destination destination=session.createTopic("DTopic");
18            //Create the Message Consumer from the session to the Topic or Queue
19            MessageConsumer consumer=session.createConsumer(destination);
20            //Create JMS Listener form messages
21            consumer.setMessageListener(new MessageListener() {
22                @Override
23                public void onMessage(Message message) {
24                    try {
25                        if (message instanceof TextMessage) {
26
27                            TextMessage textmessage= (TextMessage) message;
28                            String text= textmessage.getText();
29                            System.out.println("Received: "+ text);}
30                        else { System.out.println("Received "+ message);}
31                    } catch (JMSException e) {
32                        // TODO Auto-generated catch block
33                        e.printStackTrace();
34                    } } } );
35            } catch (JMSException e) {
36                // TODO Auto-generated catch block
37                e.printStackTrace();
38            }
```

# Création du producteur et du consommateur

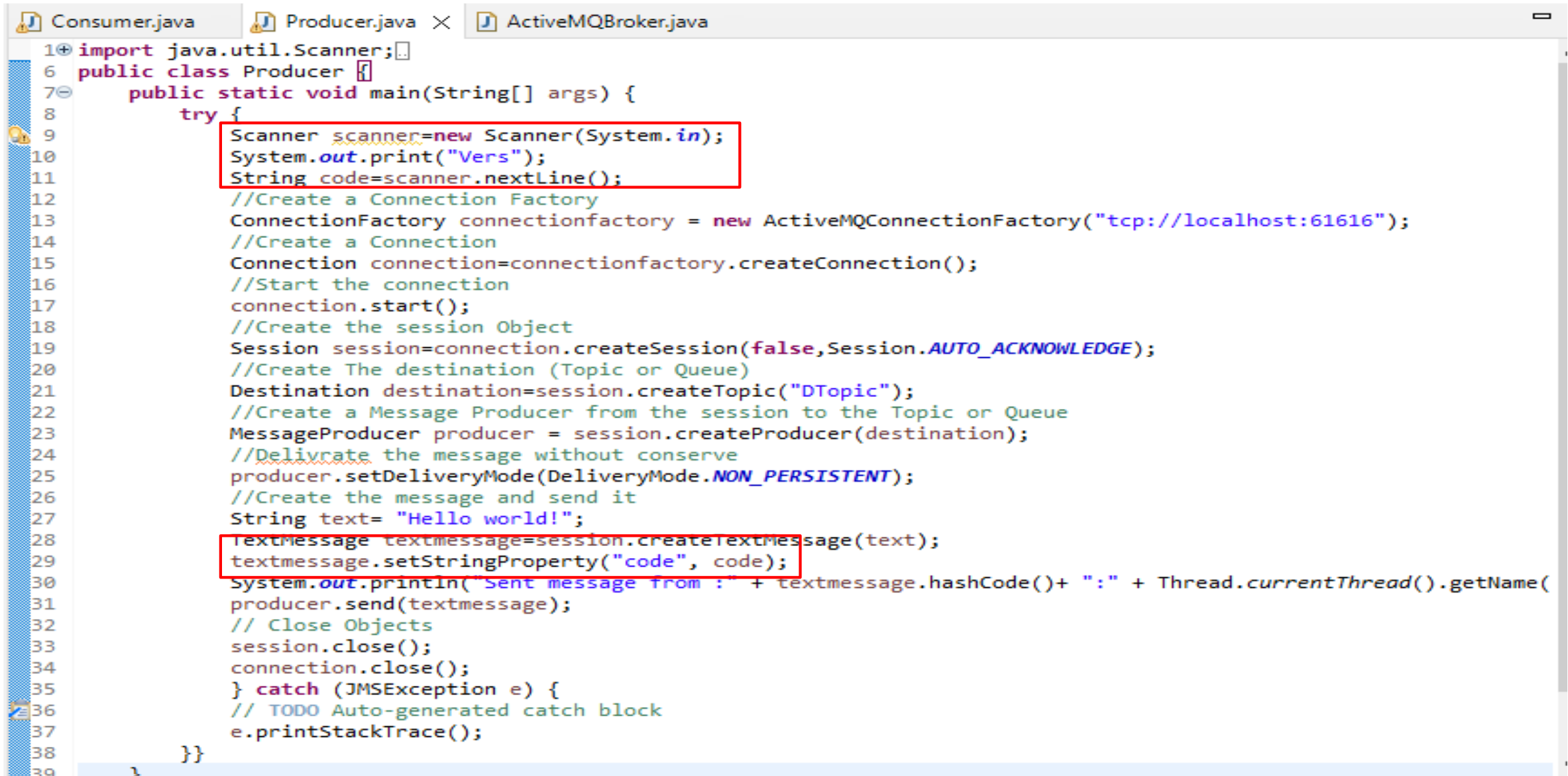
```
Consumer.java  *Producer.java X
1+ import javax.jms.*;
4 public class Producer {
5
6 public static void main(String[] args) {
7     try {
8         //Create a Connection Factory
9         ConnectionFactory connectionfactory = new ActiveMQConnectionFactory("tcp://localhost:61616");
10        //Create a Connection
11        Connection connection=connectionfactory.createConnection();
12        //Start the connection
13        connection.start();
14        //Create the session Object
15        Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
16        //Create The destination (Topic or Queue)
17        Destination destination=session.createTopic("DTopic");
18        //Create a Message Producer from the session to the Topic or Queue
19        MessageProducer producer = session.createProducer(destination);
20        //Delivrate the message without conserve
21        producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
22        //Create the message and send it
23        String text= "Hello world!";
24        TextMessage textmessage=session.createTextMessage(text);
25        System.out.println("Sent message from : " + textmessage.hashCode() + " : " + Thread.currentThread().getName());
26        producer.send(textmessage);
27        // Close Objects
28        session.close();
29        connection.close();
30
31    } catch (JMSException e) {
32        // TODO Auto-generated catch block
33        e.printStackTrace();
34    }}
35 }
```

# Filtrer les messages dans le cas de Topic

```
ActiveMQBroker.java *Consumer.java x Producer.java
1+ import java.util.Scanner;
7 public class Consumer {
8     public static void main(String[] args) {
9         Scanner scanner=new Scanner(System.in);
10        System.out.print("Code:");
11        String code=scanner.nextLine();
12        try {
13            //Create a Connection Factory
14            ConnectionFactory connectionfactory= new ActiveMQConnectionFactory("tcp://localhost:61616");//
15            //Create a Connection
16            Connection connection=connectionfactory.createConnection();
17            //Start The connection
18            connection.start();
19            //Create the session
20            Session session= connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
21            //Create The destination (Topic or Queue)
22            Destination destination=session.createTopic("DTopic");
23            //Create the Message Consumer from the session to the Topic or Queue
24            // MessageConsumer consumer=session.createConsumer(destination);
25            MessageConsumer consumer=session.createConsumer(destination, "code = '" + code + "'");
26            //Create JMS Listener form messages
27            consumer.setMessageListener(new MessageListener() {
28                @Override
29                public void onMessage(Message message) {
30                    try {
31                        if (message instanceof TextMessage) {
32
33                            TextMessage textmessage= (TextMessage) message;
34                            String text= textmessage.getText();
35                            System.out.println("Received: "+ text);}
36                        else { System.out.println("Received "+ message);}
37                    } catch (JMSEException e) {
38                        // TODO Auto-generated catch block
39                        e.printStackTrace();
40                    } } } );
41        } catch (JMSEException e) {
42            // TODO Auto-generated catch block
43            e.printStackTrace();
44        } }}
```



# Filtrer les messages dans le cas de Topic



```
Consumer.java  Producer.java  ActiveMQBroker.java
1 import java.util.Scanner;
2
3 public class Producer {
4     public static void main(String[] args) {
5         try {
6             Scanner scanner=new Scanner(System.in);
7             System.out.print("Vers");
8             String code=scanner.nextLine();
9             //Create a Connection Factory
10            ConnectionFactory connectionfactory = new ActiveMQConnectionFactory("tcp://localhost:61616");
11            //Create a Connection
12            Connection connection=connectionfactory.createConnection();
13            //Start the connection
14            connection.start();
15            //Create the session Object
16            Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
17            //Create The destination (Topic or Queue)
18            Destination destination=session.createTopic("DTopic");
19            //Create a Message Producer from the session to the Topic or Queue
20            MessageProducer producer = session.createProducer(destination);
21            //Delivrate the message without conserve
22            producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
23            //Create the message and send it
24            String text= "Hello world!";
25            TextMessage textmessage=session.createTextMessage(text);
26            textmessage.setStringProperty("code", code);
27            System.out.println("Sent message from : " + textmessage.hashCode()+ ":" + Thread.currentThread().getName());
28            producer.send(textmessage);
29            // Close Objects
30            session.close();
31            connection.close();
32        } catch (JMSException e) {
33            // TODO Auto-generated catch block
34            e.printStackTrace();
35        }
36    }
37 }
```

# Différence entre JMS et AMQP

---

	AMQP	JMS
<b>Abréviation</b>	Advanced Message Queuing Protocol.	Java Message Service.
<b>Abstraction</b>	AMQP est un protocole	JMS est un standard API
<b>Développé par</b>	JPMorgan Chase.	Sun Microsystems.
<b>Modèle d'envoi de message</b>	Utilise Direct, Fanout, Topic and Headers.	Utilise Publish/Subscribe et P2P (Point to Point).
<b>Types de données</b>	AMQP utilise uniquement des données binaires.	JMS supporte différents types de données : MapMessage, ObjectMessage, Text message, StreamMessage et BytesMessage.

## Application de ActiveMQ

---

- ▶ **Red Hat:** un des principaux fournisseurs de solutions open source, intègre ActiveMQ dans son produit Red Hat AMQ (ActiveMQ).
- ▶ **Alibaba:** le géant chinois du commerce électronique, est connu pour utiliser ActiveMQ dans sa pile technologique.
- ▶ **Cisco:** un conglomérat technologique multinational, a utilisé ActiveMQ dans certains de ses systèmes.
- ▶ **ING:** une société mondiale de services bancaires et financiers a utilisé ActiveMQ dans son infrastructure.
- ▶ **J.P. Morgan:** une institution financière majeure, a utilisé ActiveMQ pour répondre aux exigences de messagerie dans ses systèmes.



## Autres Brokers

---

- ▶ **RabbitMQ**: un broker Open source qui implémente les protocoles Advanced Message Queuing (AMQP), Streaming Text Oriented Messaging Protocol (STOMP) et Message Queuing Telemetry Transport (MQTT). Plus performant que ActiveMQ.
- ▶ **Kafka**: plus performant que RabbitMQ. Kafka est une plateforme de diffusion (Streaming) distribuée développé en Scala et Java. Son broker possède 3 fonctionnalités clés:
  - ▶ Permettre aux applications clientes de s'abonner à des files d'attente comme ActiveMQ et RabbitMQ pour échanger es messages entre les applications.
  - ▶ Permet de stocker les messages de manière durable et tolérante aux pannes (ne supprime les messages).
  - ▶ Permet de traiter les messages en temps réel (Big Data).