Distributed Systems

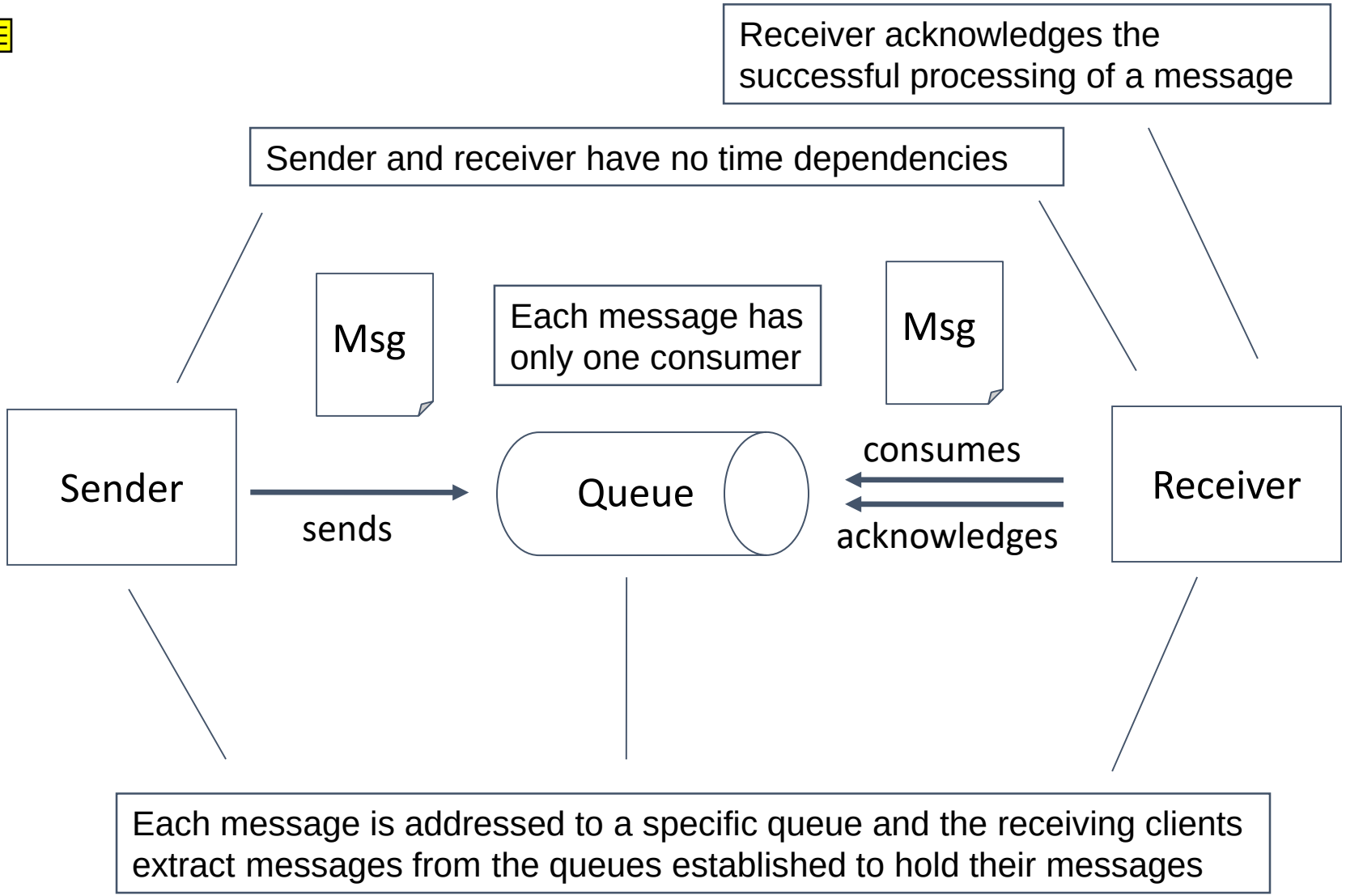
Java Messaging System

Prof. Agostino Poggi

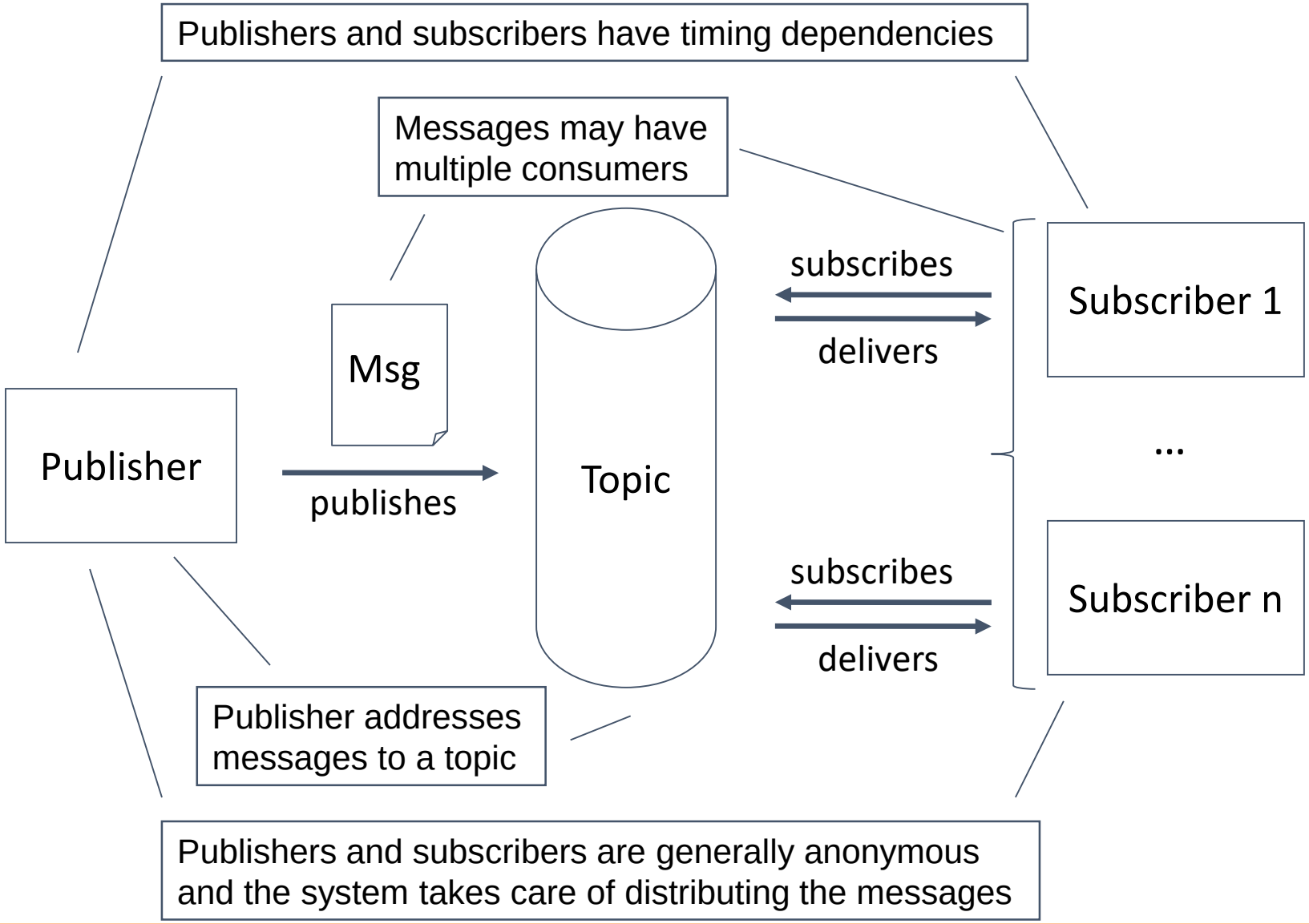
What is JMS?

- Acronym of Java Messaging System
- A **standardized** and **system independent** Java **API** for development of heterogeneous, distributed applications based on **message exchange** through both **point-point** and **publish-subscribe** protocols
- **Minimizes** the **set of concepts** a programmer must learn to **use messaging**, but provides enough features to **support sophisticated applications**
- **Maximizes** the **portability** of applications across JMS providers in the same messaging domain

Point-to-Point Messaging



Publish/Subscribe Messaging



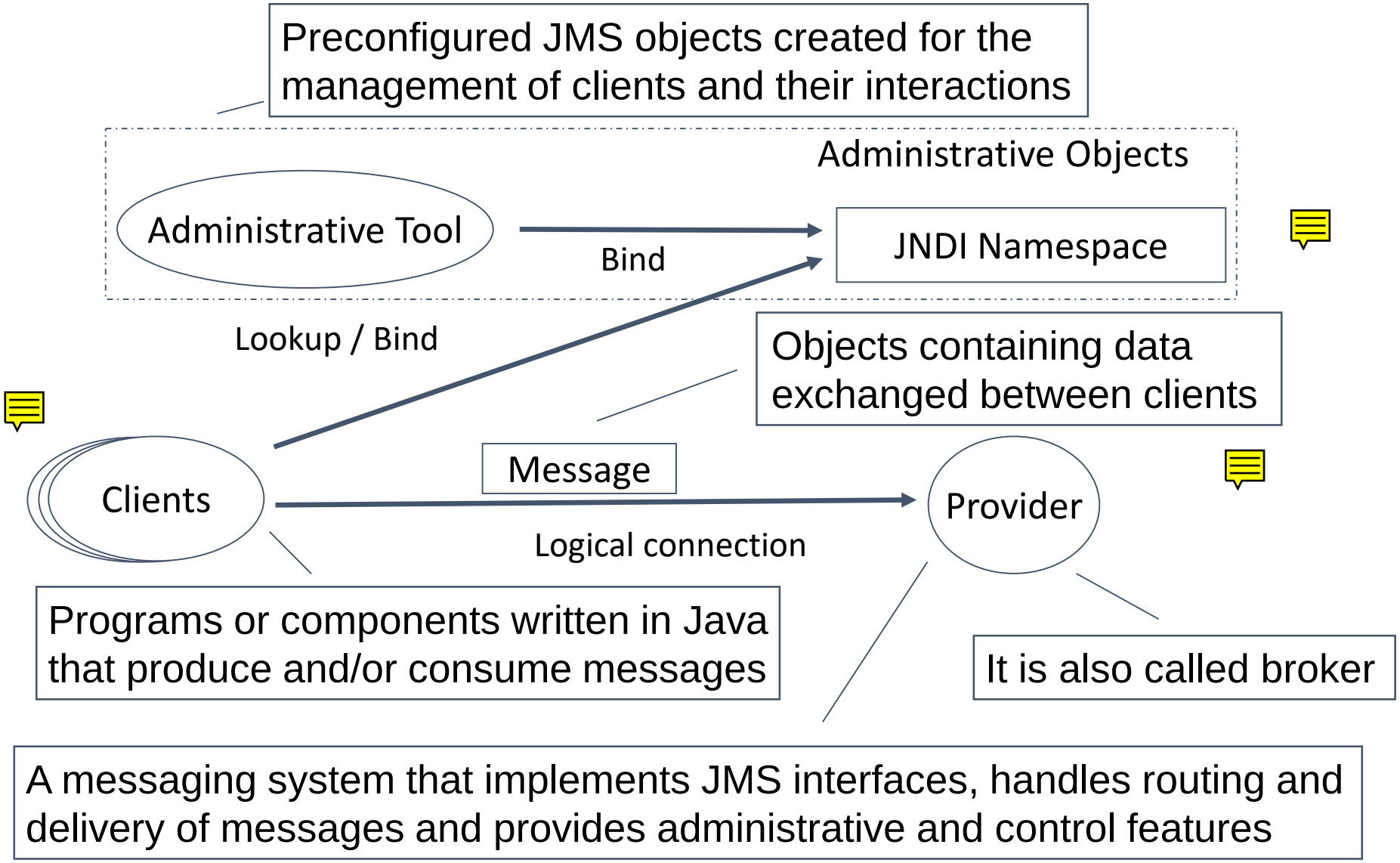
Client – Client Communication

- Communication between clients is not only loosely coupled but also
  - Asynchronous
    - JMS provider can deliver messages to a client as they arrive and the **client** does **not have** to **request messages** in order to receive them
  - Reliable
    - JMS API can ensure that a **message** is delivered once and **only once**
    - But **lower levels** of **reliability** are available for applications that can afford to **miss messages** or to receive **duplicate messages**

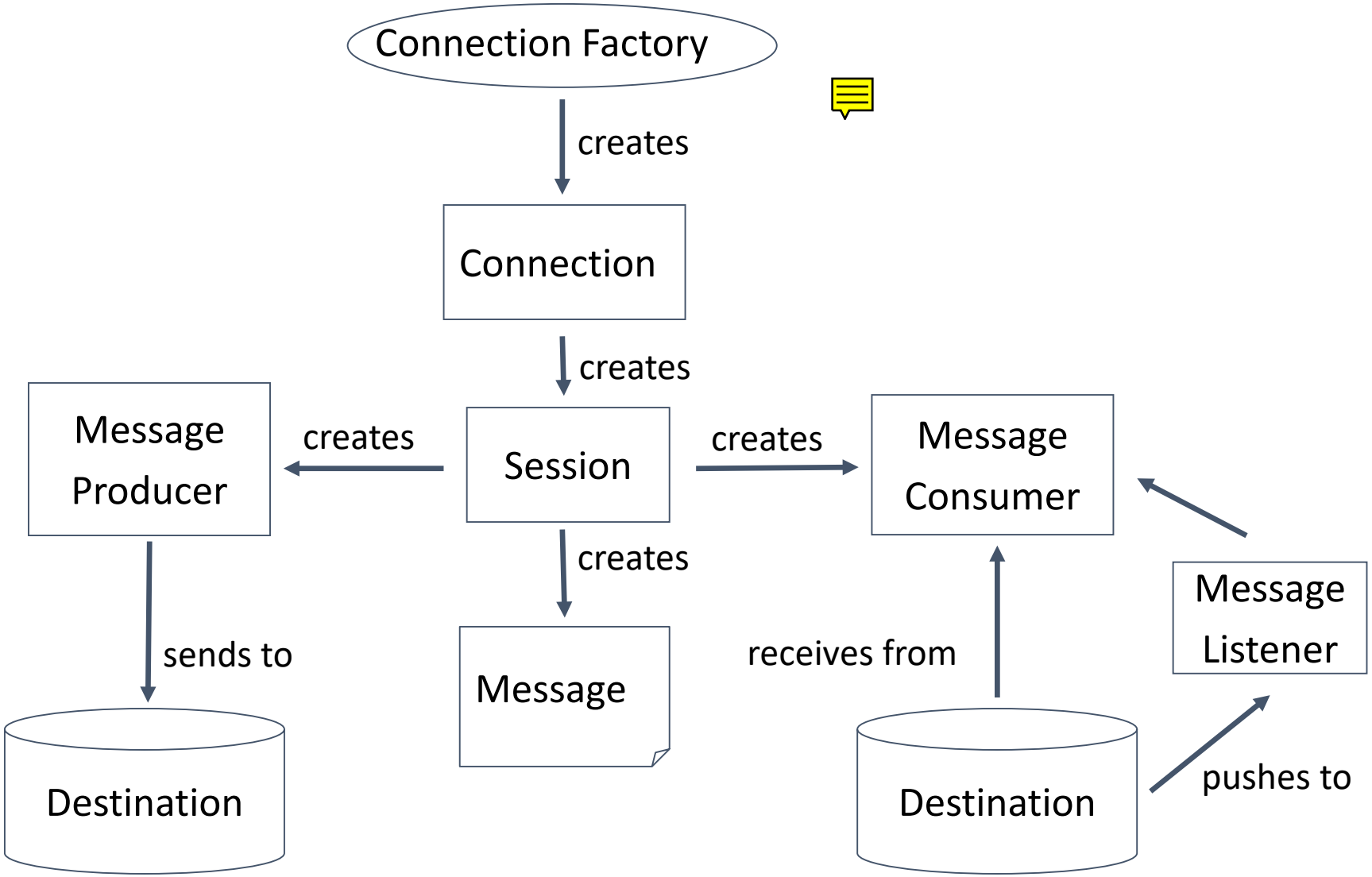
Message Consumptions

- Synchronously
  - A **client** (subscriber or receiver) explicitly fetches the message from the destination by calling the **receive method**
  - The receive method can **block** until a **message arrives** or can time out if a message does not arrive within a specified **time limit**
- Asynchronously
  - A **client** can **register** a **message listener** with a consumer
  - Whenever a **message arrives** at the destination, the **JMS provider** delivers the message by **calling** the **listener**

JMS Components Interaction



JMS Programming Model



Connection Factory

- Is used by a client to **create** a **connection** with a provider
- Encapsulates a set of connection **configuration parameters** that have been defined by an administrator
- Is an instance of one of the following interfaces
  - QueueConnectionFactory
  - TopicConnectionFactory

Connection

- Encapsulates a **virtual connection** with a **JMS provider**
- Could represent an open **TCP/IP socket** between a client and a provider service daemon
- Is used for **creating** one or **more sessions**
- Is an instance of one of the following interfaces
  - QueueConnection
  - TopicConnection

Session

- Is a **single-threaded context** for producing and consuming messages
- Is used for **creating** message **producers**, message **consumers** and **messages**
- Sessions **serialize** the **execution** of **message listeners**
- Is an instance of one of the following interfaces
  - QueueSession
  - TopicSession

Destination

- Is an object **created** by a **session** that a client uses to **specify**
  - The **target** of messages it **produces**
  - The **source** of messages it **consumes**
- Is an instance of one of the following interfaces
  - Queue
  - Topic

Message Producer

- Is an object **created** by a **session** and is used for **sending messages** to a destination

- Is an instance of one of the following interfaces
  - QueueSender
  - TopicPublisher

Message Consumer

- Is an object **created** by a **session** and is used for **receiving messages** sent to a destination
- Allows a JMS client to **register interest** in a **destination** with a JMS provider
- Manages the **retrieval** of **messages** from a destination
- Is an instance of one of the following interfaces
  - QueueReceiver
  - TopicSubscriber

Message Listener

- Is an object that acts as an asynchronous **event handler** for messages

- Is an instance of the following interface
  - MessageListener

- Contains a method, called ***onMessage***, that must define the actions to be taken when a message arrives

Message Components

- A header containing a **set of fields** describing the message and its scope

- An **optional body** whose **content depends** on the **message type**

- Some **optional properties** defined by the client

Message Header

Field	Set By	Meaning
JMSDestination	Send / publish method	Queue or Topic
JSMDeliveryMode	Send / publish method	(non-) persistent
JMSExpiration	Send / publish method	Expiration time
JMSPriority	Send / publish method	From 0 (low) to 9 (high)
JMSMessageID	Send / publish method	Unique identifier
JMSTimestamp	Send / publish method	Hand off time
JMSCorrelationID	Client	Messages correlation
JMSReplyTo	Client	Destination of the reply
JMSType	Client	Contents type
JMSRedelivered	Provider	Retransmitted message

Message Types

	Contains	Methods
TextMessage	String	getText, setText, …
MapMessage	Name and value pairs set	getString, setString, getLong, setLong, …
BytesMessage	Uninterpreted bytes stream	readBytes, writeBytes, …
StreamMessage	Primitive values stream	readString, writeString, readLong,writeLong, …
ObjectMessage	Serialized object	getObject, setObject, …

JMS Implementations

- ♦ ActiveMQ (http://activemq.apache.org)
  - Apache implementation (JMS 1.1)
- ♦ Open Message Queue (https://javaee.github.io/openmq/)
  - Oracle reference implementation (JMS 2.0)
- ♦ Several other open source and commercial implementations are available (BEA, IBM, Jboss, TIBCO, ….)

ActiveMQ

- Is the **most known** o**pen-source** message broker
- Is generally stable and **high-performance**
- Can be run standalone, or inside another process, application server, or Java EE application
- Supports **everything JMS** requires, plus various **extensions**
- Integrates well into other products
- Supports a **variety** of cross **language clients** and protocols: Java, C, C++, C#, Ruby, Perl, Python and PHP

ActiveMQ Extensions

- **Virtual destinations**
  - Load-balancing and failover/backup for topics
  - Messages are automatically sent to multiple queues
- **Retroactive subscriptions**
  - Subscribers can receive some previous messages
- **Exclusive consumers**
  - Broker will pick a consumer for a message queue
  - If that consumer fails, the broker choose another consumer
- **Message groups**
  - Provides load balancing in processing the messages of a queue
- **Mirrored queues**
  - Message sent to a queue are also sent to a topic

ActiveMQ Features

- Provides **two** main **protocols** for provider (broker) – client communication
  - **OpenWire** (binary) is the default and has the most history and best support (including SSL) for the main implementations (i.e., Java, C++ and C#)
  - **Stomp** (text) is the easiest to develop for and therefore has the most cross-language support
- Provides different **persistence strategies** that take advantage **of files** and **databases** and can be **customized** for specific **performance** requirements

ActiveMQ Initial Setup

- Make available **ActiveMQ** distribution **JAR file**
- Define the way of **managing connection factories** and **destinations**
  - Configure the **JNDI service** through the **jndi.properties** file (if the broker is started as a standalone application)
  - Use an **ActiveMQConnectionFactory** inside the clients code
- **Start the broker**
  - As a **standalone** application
  - As an **embedded broker** inside the code of a client

Object Messages and Security

- The use of object messages is generally not recommended, for the following reasons
  - Messaging systems enable loose coupling between producers and consumers, but objects introduces coupling of class paths between them
  - Using object messages presents a significant security risk, because the serialized objects can be used to transfer malicious code
- For this reason, ActiveMQ forces users to explicitly whitelist packages that can be exchanged using object messages
  - System.*setProperty(*
    "org.apache.activemq.SERIALIZABLE_PACKAGES",
    "java.util,org.apache.activemq,com.mycompany.myapp“);
  - System.setProperty(
    "org.apache.activemq.SERIALIZABLE_PACKAGES", "*");

JNDI Configuration File

```
java.naming.factory.initial =\
org.apache.activemq.jndi.ActiveMQInitialContextFactory

# use the following property to configure the default
connector

java.naming.provider.url = vm://localhost

# use the following property to specify the JNDI name of
the connection factory

#connectionFactoryNames =\

#connectionFactory, queueConnectionFactory,
topicConnectionFactory

# register some queues and topics in JNDI using the forms

# queue.[jndiName] = [physicalName]

queue.MyQueue = example.MyQueue

# topic.[jndiName] = [physicalName]

topic.MyTopic = example.MyTopic
```

Embedded Broker Code

```
BrokerService broker = BrokerFactory.createBroker(
 "broker:(tcp://localhost:61616)?persistent=false&useJmx=false");

broker.start();
```

Connection Factory Code

```
/*search the classpath for jndi.properties (vendor specific file) */
Context ctx = new InitialContext();
QueueConnectionFactory queueConnectionFactory =
  (QueueConnectionFactory) ctx.lookup("MyQueueConnectionFactory");
TopicConnectionFactory topicConnectionFactory =
  (TopicConnectionFactory) ctx.lookup("MyTopicConnectionFactory");
```

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

Connection Code (1/2)

```
QueueConnection queueConnection =
  queueConnectionFactory.createQueueConnection();
TopicConnection topicConnection =
  topicConnectionFactory.createTopicConnection( );

/* Before your application can consume messages, start them: */
queueConnection.start( );
topicConnection.start( );
...
/* To stop a connection temporarily use the stop method: */
queueConnection.stop( );
topicConnection.stop( );
...
/* When the application is complete, close the connections:  */
queueConnection.close( );
topicConnection.close( );
```

Connection Code (2/2)

```
ActiveMQConnection connection =
  (ActiveMQConnection) connectionFactory.createConnection();

/* Before your application can consume messages, start it: */
connection.start( );
...
/* To stop a connection temporarily use the stop method: */
connection.stop( );
...
/* When the application is complete, close the connection:  */
connection.close( );
```

Session Code

```
QueueSession queueSession = queueConnection.createQueueSession(
  false, Session.AUTO_ACKNOWLEDGE);
TopicSession topicSession = topicConnection.createTopicSession(
  true, 0);
```

AUTO_ACKNOWLEDGE
CLIENT_ACKNOWLEDGE
DUPS_OK_ACKNOWLEDGE

true = transacted
false = not transacted

Messages are acknowledged in batches rather than individually

ActiveMQ

```
QueueSession queueSession =  connection.createQueueSession(
  false, Session.CLIENT_ACKNOWLEDGE);
TopicSession topicSession = connection.createTopicSession(
  true, 0);
```

Destination Code

Destination is created if it does not exist

```
Queue myQueue = queueSession.createQueue("MyQueue");
Topic myTopic = topicSession.createTopic("MyTopic");
```

```
Context ctx = new InitialContext();
...
Queue myQueue = (Queue) ctx.lookup("MyQueue");
Topic myTopic = (Topic) ctx.lookup("MyTopic");
```

Message Producer Code

```
QueueSender queueSender =
  queueSession.createSender(myQueue);

... creation of a message ...

queueSender.send(message);


TopicPublisher topicPublisher =
  topicSession.createPublisher(myTopic);

... creation of a message ...

topicPublisher.publish(message);
```

Message Consumer Code

```
Message qm1, qm2;
QueueReceiver queueReceiver =
  queueSession.createReceiver(myQueue);
qm1 = queueReceiver.receive();
qm2 = queueReceiver.receive(1000);
```

Waits 1000 milliseconds

```
Message tm1, tm2;
TopicSubscriber topicSubscriber1 =
  topicSession.createSubscriber(myTopic);
TopicSubscriber topicSubscriber2 =
  topicSession.createDurableSubscriber(myTopic);
tm1 = topicSubscriber1.receive();
tm2 = topicSubscriber2.receive(1000);
```

A durable subscription **saves messages** for an **inactive subscriber** and **delivers** these saved messages when the **subscriber reconnects**

Message Listener Code

```
class MyMessageListener implements MessageListener {
  public void onMessage(Message m) {
    System.out.println("Received message= " + m);
}
```

```
MyMessageListener listener = new MyMessageListener();
queueReceiver.setMessageListener(listener);
topicSubscriber.setMessageListener(listener);
```

Message Code

```
ObjectMessage message =
  queueSession.createObjectMessage();

message.setObject(new Object[2] { "hello world!", 10 });
queueSender.send(message);
```

```
ObjectMessage message = queueReceiver.receive();
Object[] a = (Object[]) message.getObject();
```

A Point-to-Point Application (1/2)

1. Write a sender/client and a receiver/server or a message listener class
2. Compile the two classes
3. Start a JMS provider
4. Create the JMS administered objects (a queue)
5. Run the sender and the receiver (in any order)
6. Delete the queue
7. Stop the JMS provider

A Point-to-Point Application (2/2)

1. Write a receiver/server or a listener class with an embedded broker

2. Write a sender\client class

3. Compile the two (three) classes

4. Run the receiver and then the sender

Client Class (1/2)

```
public class Client
{
  private static final String BROKER_URL = "tcp://localhost:61616";
  private static final String QUEUE_NAME = "server";

  public void send(final int n)
  {
    ActiveMQConnection connection = null;

    try
    {

    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

  public static void main(final String[] args)
  {
    final int n = 3;

    new Client().send(n);
  }
}
```

Client Class (2/2)

```java
public class Client
{
  private static final String B
  private static final String Q

  public void send(final int n)
  {
    ActiveMQConnection connecti

    try
    {

    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

}
```

```java
ActiveMQConnectionFactory cf =
    new ActiveMQConnectionFactory(Client.BROKER_URL);

connection = (ActiveMQConnection) cf.createConnection();

connection.start();

QueueSession session =
  connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

Destination serverQueue  = session.createQueue(QUEUE_NAME);
MessageProducer producer = session.createProducer(serverQueue);

Destination tempDest = session.createTemporaryQueue();

MessageConsumer consumer = session.createConsumer(tempDest);

TextMessage request = session.createTextMessage();

request.setText("Client request message");
request.setJMSReplyTo(tempDest);
request.setJMSCorrelationID("123");

producer.send(request);

Message reply = consumer.receive();

System.out.println("Message: " + ((TextMessage) reply).getText());
```

```java
public static void main(final String[] args)
{
  final int n = 3;

  new Client().send(n);
}
```

Server Class (1/2)

```java
public class Server
{
  private static final String BROKER_URL   = "tcp://localhost:61616";
  private static final String BROKER_PROPS = "persistent=false&useJmx=false";
  private static final String QUEUE_NAME   = "server";

  public void receive()
  {
    ActiveMQConnection connection = null;
    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

  public static void main(final String[] args)
  {
    new Server().receive();
  }
}
```

Server Class (2/2)

```java
public class Server
{
  private static final String BR
  private static final String BR
  private static final String QU

  public void receive()
  {
    ActiveMQConnection connectio
    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

  public static void main(final String[] args)
  {
    new Server().receive();
  }
}
```

```java
BrokerService broker = BrokerFactory.createBroker(
    "broker:(" + BROKER_URL + ")?" + BROKER_PROPS);

broker.start();

ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(Server.BROKER_URL);
connection = (ActiveMQConnection) cf.createConnection();

connection.start();

QueueSession session =
  connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

Queue queue = session.createQueue(Server.QUEUE_NAME);

QueueReceiver receiver = session.createReceiver(queue);

Message request = receiver.receive();

System.out.println("Message: " + ((TextMessage) request).getText());

MessageProducer producer = session.createProducer(null);

TextMessage reply = session.createTextMessage();

reply.setText("Server reply message");
reply.setJMSCorrelationID(request.getJMSCorrelationID());

producer.send(request.getJMSReplyTo(), reply);
```

Sender Class (1/2)

```java
public class Sender
{
  private static final String BROKER_URL = "tcp://localhost:61616";
  private static final String QUEUE_NAME = "queue";

  public void send(final int n)
  {
    ActiveMQConnection connection = null;

    try
    {

    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }
}
```

```java
public static void main(final String[] args)
{
  final int n = 3;

  new Sender().send(n);
}
```

Sender Class (2/2)

```java
public class Sender
{
  private static final String BROKER_URL = "tcp://localhost:61616";
  private static final String QUEUE_NAME = "queue";

  public void send(final int n)
  {
    ActiveMQConnection connection = null;

    try
    {

    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

}
```

```java
ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(Sender.BROKER_URL);

connection = (ActiveMQConnection) cf.createConnection();

connection.start();

QueueSession session =
  connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

Queue queue         = session.createQueue(Sender.QUEUE_NAME);
QueueSender sender  = session.createSender(queue);
TextMessage message = session.createTextMessage();

for (int i = 0; i < n; i++)
{
  message.setText("This is message " + (i + 1));
  sender.send(message);
}

sender.send(session.createMessage());
```

```java
public static void main(final String[] args)
{
  final int n = 3;

  new Sender().send(n);
}
```

Receiver Class (1/2)

```java
public class Receiver
{
  private static final String BROKER_URL   = "tcp://localhost:61616";
  private static final String BROKER_PROPS = "persistent=false&useJmx=false";
  private static final String QUEUE_NAME   = "queue";

  public void receive()
  {
    ActiveMQConnection connection = null;
    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

}
```

```java
public static void main(final String[] args)
{
  new Receiver().receive();
}
```

Receiver Class (2/2)

```java
public class Receiver
{
  private static final St
  private static final St
  private static final St

  public void receive()
  {
    ActiveMQConnection co
    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace()
    }
    finally
    {
      if (connection != n
      {
        try
        {
          connection.clos
        }
        catch (JMSExcept
        {
          e.printStackTra
        }
      }
    }
  }

}
```

```java
BrokerService broker = BrokerFactory.createBroker(
    "broker:(" + BROKER_URL + ")?" + BROKER_PROPS);

broker.start();

ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(Receiver.BROKER_URL);

connection = (ActiveMQConnection) cf.createConnection();

connection.start();

QueueSession session =
  connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

Queue queue = session.createQueue(Receiver.QUEUE_NAME);

QueueReceiver receiver = session.createReceiver(queue);

while (true)
{
  Message message = receiver.receive();

  if (message instanceof TextMessage)
  {
    System.out.println("Message: " + ((TextMessage) message).getText());
  }
  else
  {
    break;
  }
}
```

```java
public static void main(final String[] args)
{
  new Receiver().receive();
}
```

QueueListener Class (1/3)

```java
public class QueueListener implements MessageListener
{
  private static final String BROKER_URL   = "tcp://localhost:61616";
  private static final String BROKER_PROPS = "persistent=false&useJmx=false";
  private static final String QUEUE_NAME   = "queue";

  private ActiveMQConnection connection = null;

  /**
   * Class constructor.
   *
  **/
  public QueueListener()
  {
    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] args)
  {
    new QueueListener();
  }
}
```

QueueListener Class (2/3)

```
public class QueueListener implements MessageListener
{
  private stati
  private stati
  private stati

  private Activ

  /**
   * Class cons
   *
  **/
  public QueueL
  {
    try
    {

    }
    catch (Exce
    {
      e.printSt
    }
  }

}
```

```
BrokerService broker = BrokerFactory.createBroker(
    "broker:(" + BROKER_URL + ")?" + BROKER_PROPS);

broker.start();

ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(QueueListener.BROKER_URL);

connection = (ActiveMQConnection) cf.createConnection();

connection.start();

QueueSession session =
  connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

Queue queue = session.createQueue(QueueListener.QUEUE_NAME);

MessageConsumer consumer = session.createConsumer(queue);

consumer.setMessageListener(this);
```

```
public static void main(final String[] args)
{
  new QueueListener();
}
```

QueueListener Class (3/3)

```java
public class QueueListener implements MessageListener
{
  private stati
  private stati
  private stati

  private Activ

  /**
   * Class cons
   *
  **/
  public QueueL
  {
    try
    {

    }
    catch (Exce
    {
      e.printSt
    }
  }

}
```

```java
BrokerService broker = BrokerFactory.createBroker(
    "broker:(" + BROKER_URL + ")?" + BROKER_PROPS);

broker.start();

Acti
  ne

conn

conn

Queu
  co

Queu

Mess

cons
```

```java
public void onMessage(final Message m)
{
  if (m instanceof TextMessage)
  {
    try
    {
      System.out.println("Message: " + ((TextMessage) m).getText());
    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
  }
  else if (connection != null)
  {
    try
    {
      connection.close();
    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
  }
}
```

```java
public st
{
  new Que
}
```

A Publish/Subscribe Application (1/2)

1. Write a publisher, a subscriber or a message listener class
2. Compile the two classes
3. Start a JMS provider
4. Create the JMS administered objects (a topic)
5. Run the subscriber and then the publisher
6. Delete the queue
7. Stop the JMS provider

A Publish/Subscribe Application(2/2)

1. Write a subscriber or a message listener class with an embedded broker

2. Write a publisher class

3. Compile the two classes

4. Run the subscriber and then the publisher

Publisher Class (1/2)

```java
public class Publisher
{
  private static final String BROKER_URL   = "tcp://localhost:61616";
  private static final String TOPIC_NAME   = "topic";

  public void publish(final int n)
  {
    ActiveMQConnection connection = null;

    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

}
```

```java
public static void main(final String[] args)
{
  final int n = 3;

  new Publisher().publish(n);
}
```

Publisher Class (2/2)

```java
public class Publisher
{
  private static final String BROKER_URL   = "tcp://localhost:61616";
  private static final String TOPIC_NAME   = "topic";

  public void publish(final int n)
  {
    ActiveMQConnection connec

    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e
        {
          e.printStackTrace();
        }
      }
    }
  }

}
```

```java
ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(Publisher.BROKER_URL);

connection = (ActiveMQConnection) cf.createConnection();

connection.start();

TopicSession session =
  connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);

Topic topic = session.createTopic(TOPIC_NAME);

TopicPublisher publisher = session.createPublisher(topic);

TextMessage message = session.createTextMessage();

for (int i = 0; i < n; i++)
{
  message.setText("This is message " + (i + 1));
  publisher.publish(message);
}

publisher.publish(session.createMessage());
```

```java
public static void main(final String[] args)
{
  final int n = 3;

  new Publisher().publish(n);
}
```

Subscriver Class (1/2)

```java
public class Subscriber
{
  private static final String BROKER_URL   = "tcp://localhost:61616";
  private static final String BROKER_PROPS = "persistent=false&useJmx=false";
  private static final String TOPIC_NAME   = "topic";

  public void receive()
  {
    ActiveMQConnection connection = null;

    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != null)
      {
        try
        {
          connection.close();
        }
        catch (JMSException e)
        {
          e.printStackTrace();
        }
      }
    }
  }

  public static void main(final String[] args)
  {
    new Subscriber().receive();
  }

}
```

Subscriver Class (2/2)

```java
public class Subscriber
{
  private static final Str
  private static final Str
  private static final Str

  public void receive()
  {
    ActiveMQConnection con

    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (connection != nu
      {
        try
        {
          connection.close
        }
        catch (JMSExcepti
        {
          e.printStackTrac
        }
      }
    }
  }

}
```

```java
BrokerService broker = BrokerFactory.createBroker(
    "broker:(" + BROKER_URL + ")?" + BROKER_PROPS);

broker.start();

ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(Subscriber.BROKER_URL);

connection = (ActiveMQConnection) cf.createConnection();

connection.start();

TopicSession session =
  connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);

Topic topic = session.createTopic(TOPIC_NAME);

TopicSubscriber subscriber = session.createSubscriber(topic);

while (true)
{
  Message message = subscriber.receive();

  if (message instanceof TextMessage)
  {
    System.out.println("Message: " + ((TextMessage) message).getText());
  }
  else
  {
    break;
  }
}
```

```java
public static void main(final String[] args)
{
  new Subscriber().receive();
}
```

TopicListener Class (1/3)

```java
public class TopicListener implements MessageListener
{
  private static final String BROKER_URL   = "tcp://localhost:61616";
  private static final String BROKER_PROPS = "persistent=false&useJmx=false";
  private static final String TOPIC_NAME   = "topic";

  private ActiveMQConnection connection = null;

  public TopicListener()
  {
    try
    {

    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

}
```

```java
public static void main(final String[] args)
{
  new TopicListener();
}
```

TopicListener Class (2/3)

```
public class TopicL
{
  private static fi
  private static fi
  private static fi

  private ActiveMQC

  public TopicListe
  {
    try
    {

    }
    catch (Exceptic
    {
      e.printStackTrace();
    }
  }

}
```

```
BrokerService broker = BrokerFactory.createBroker(
    "broker:(" + BROKER_URL + ")?" + BROKER_PROPS);

broker.start();

ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(TopicListener.BROKER_URL);

connection = (ActiveMQConnection) cf.createConnection();

connection.start();

TopicSession session =
  connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);

Topic topic = session.createTopic(TOPIC_NAME);

MessageConsumer consumer = session.createConsumer(topic);

consumer.setMessageListener(this);
```

```
public static void main(final String[] args)
{
  new TopicListener();
}
```

TopicListener Class (3/3)

```
public class TopicL
{
  private static fi
  private static fi
  private static fi

  private ActiveMQC

  public TopicListe
  {
    try
    {

    }
    catch (Exceptic
    {
      e.printStackTrace
    }
  }

}
```

```
public static
{
  new TopicLis
}
```

```
BrokerService broker = BrokerFactory.createBroker(
    "broker:(" + BROKER_URL + ")?" + BROKER_PROPS);

broker.start();

ActiveMQConnectionFactory cf =
  new ActiveMQConnectionFactory(TopicListener.BROKER_URL);

conn
conn
Topi
  co
Topi
Mess
cons
```

```
public void onMessage(final Message m)
{
  if (m instanceof TextMessage)
  {
    try
    {
      System.out.println("Message: " + ((TextMessage) m).getText());
    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
  }
  else if (connection != null)
  {
    try
    {
      connection.close();
    }
    catch (JMSException e)
    {
      e.printStackTrace();
    }
  }
}
```