Copyright Notice

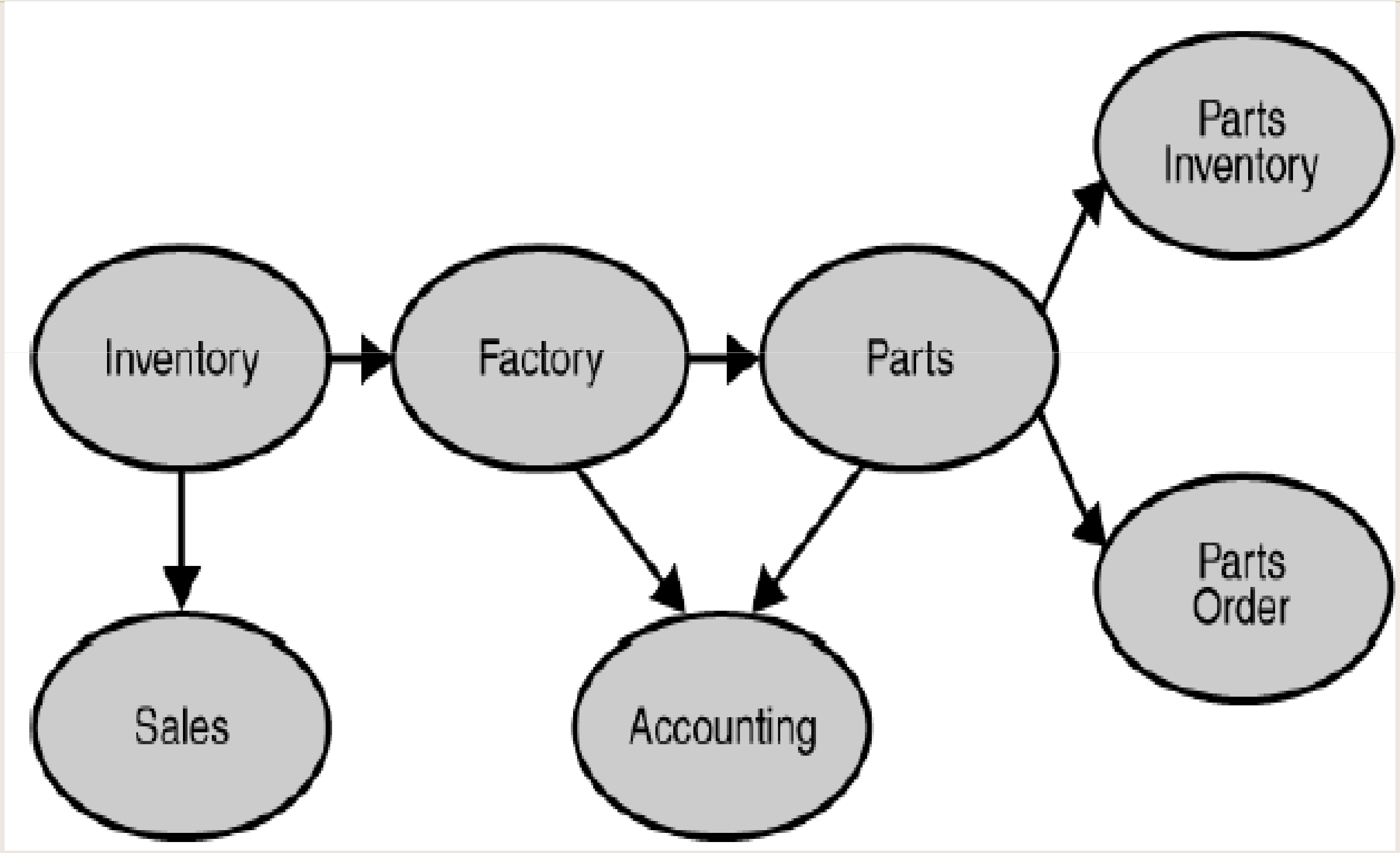
Java Messaging Service

Prakash Badhe

prakash.badhe@vishwasoft.in

www.vishwasoft.in

Application Messaging in Enterprise systems



Application Messaging

Messaging is a method of communication between software components or applications. A messaging system is a peer-to-peer facility: a messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging agent that provides facilities for creating, sending, receiving, and reading messages.Depending on the mode of communication there are two types of messaging systems

Publish-subscribe type system : Synchronous and asynchronous communication with multiple publishers and subscribers.

Point to point messaging : Synchronous communication between two clients at a time.

Messaging Applications

- *Credit card transactions*
- *Weather reporting*
- *Workflow*
- *Network management*
- *Supply chain management*
- *Customer care*
- *Communications (voice over ip,paging systems, etc.)*

Messaging System Types

- Depending on the mode of communication there are types of messaging systems are defined
- Publish-subscribe messaging : Synchronous and asynchronous communication with multiple publishers and subscribers. (one to many or many to many communication)
- Point to point messaging : Synchronous communication between two clients at a time. (one to one communication only)
- Request-Reply Messaging : web server and web browser communication.
-

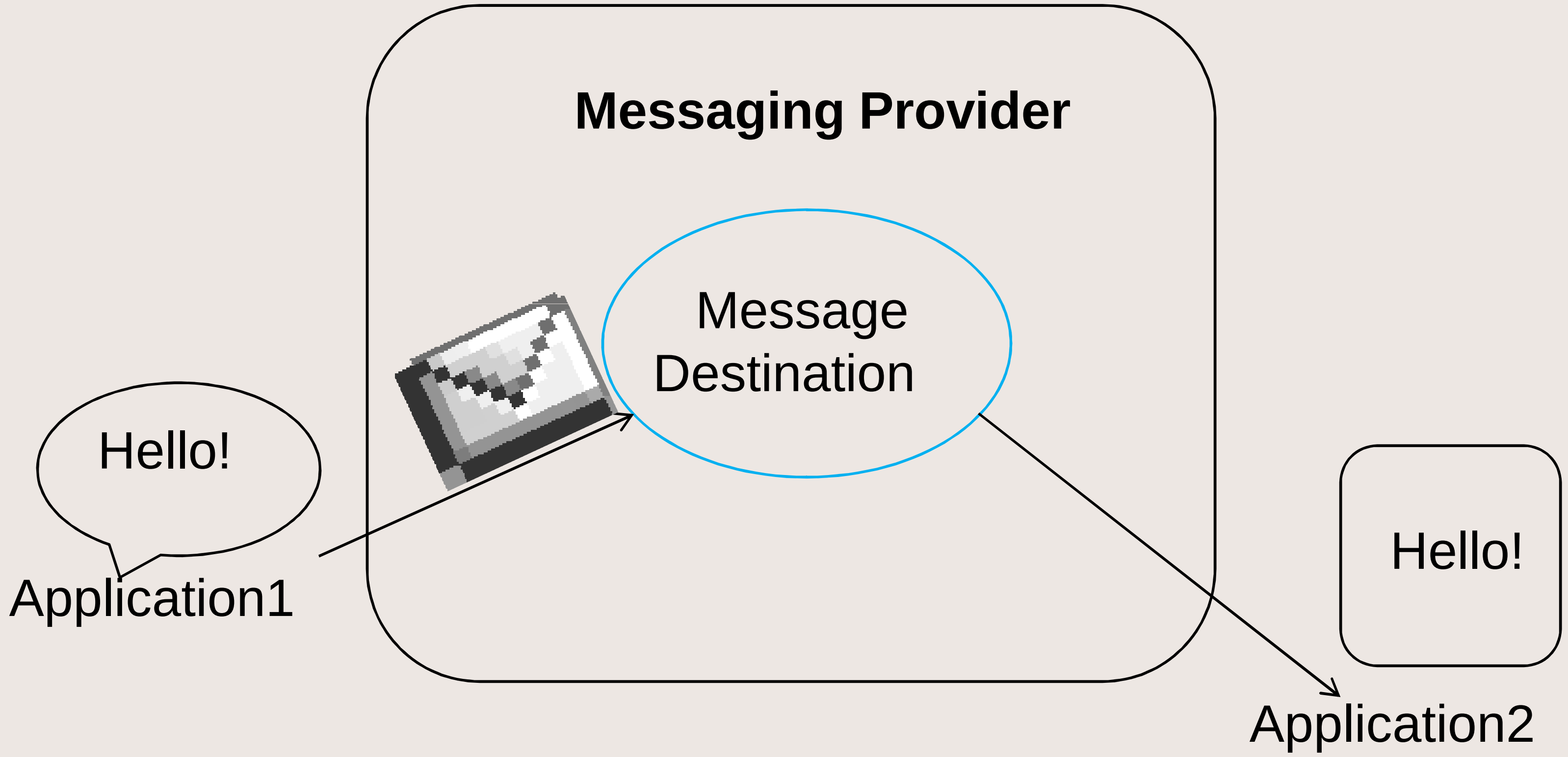
Messaging Provider

- Message-Oriented-Middleware provides a common reliable system for programs to create, send, receive and read messages in a distributed Enterprise System.
- MOM ensures fast, reliable asynchronous electronic communication, guaranteed message delivery, receipt notification and transaction control.

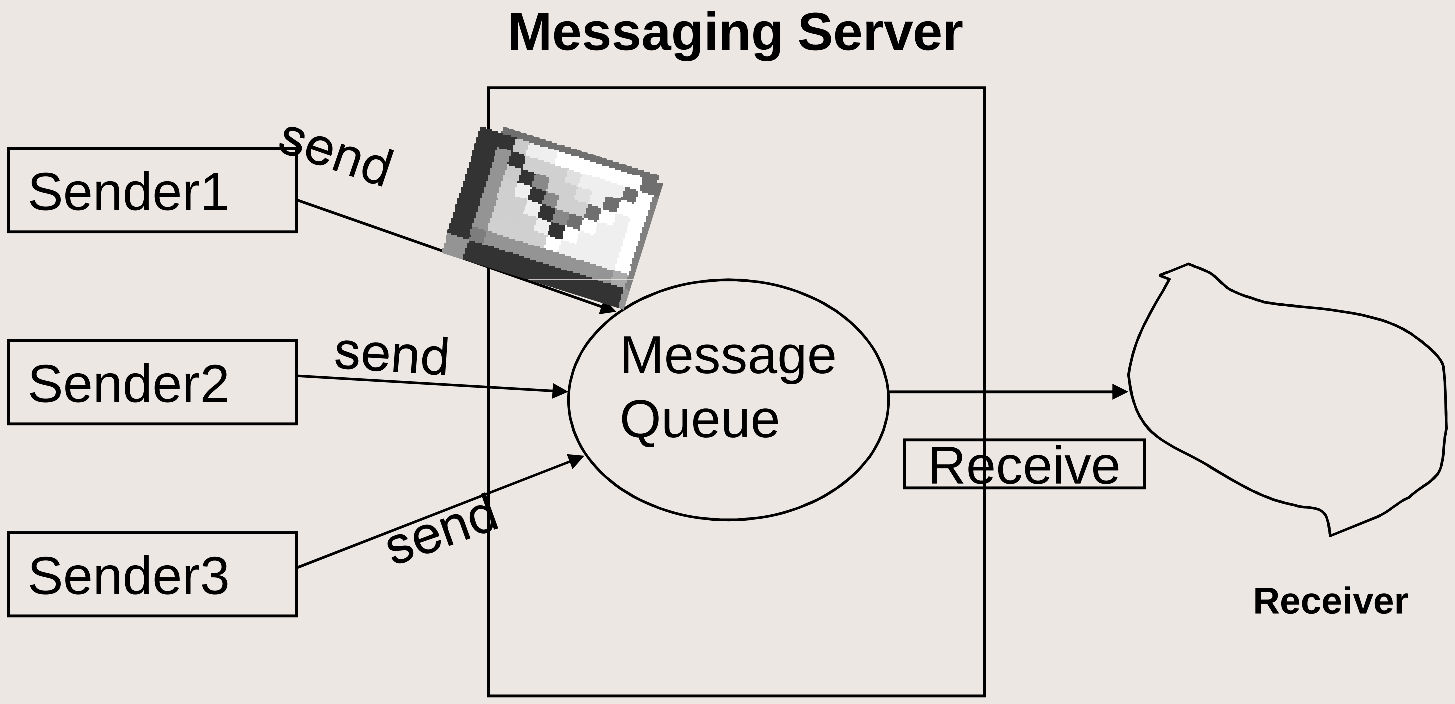
Messaging Systems

- In enterprise applications there are different types of messaging systems and they need to interoperate and communicate each other.
- Most of the systems are proprietary which makes it difficult for integration and portability.
- J2ee defines an open messaging standard system which works seamlessly with other systems and defines new messaging systems and services.

Messaging Architecture



Point to Point Messaging



Point to Point Messaging

- When one process needs to send a message to another running process, Point-To-Point Messaging is used.
- The central destination in this model is the **Queue**.
- This may or may not be a one-way relationship.
- The client to a Messaging system may only send messages, only receive messages, or send and receive messages. At the same time, another client can also send and/or receive messages.
- In the simplest case, one client is the Sender of the message and the other client is the Receiver of the message .

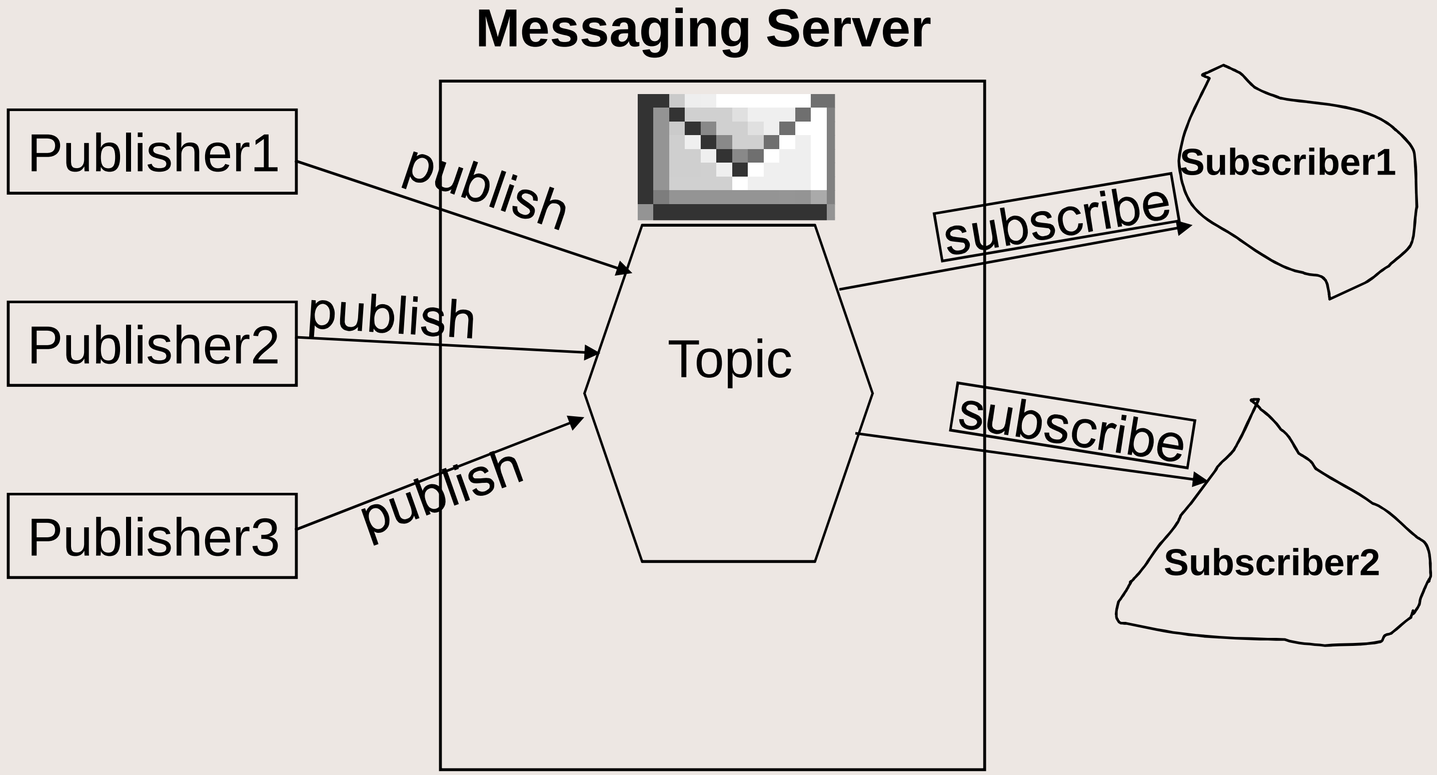
Point to point Modes

- First mode: the client directly sends a message to another client.
- The common implementation is based on the concept of a *Message Queue* where the clients connect to the message queue.
- The sender sends the message to the queue and it is delivered to the receiver, if it is connected.
- The message is preserved at Queue, until it is delivered to the receiver.
- The moment the message is delivered to the receiver, it is destroyed from the Queue.
- Even though there may be multiple Senders of messages in this mode there is only a single Receiver for the messages.
-

Message Delivery

- Reliable queuing
- Based on message queues
- Each message addressed to specific queue
- Client extracts messages from queue
- The message can be delivered synchronously or asynchronously

Publish-Subscribe Model



Publish-Subscribe Messaging

- When multiple applications need to receive the same messages, Publish-Subscribe Messaging is used.
- The central destination in a Publish-Subscribe messaging system is the **Topic**. Multiple Publishers may send messages to a Topic, and all Subscribers to that Topic receive all the messages sent to that Topic.
- This model is extremely useful when a group of applications want to notify each other of a particular occurrence of an event.
- In this messaging there may be multiple Senders and multiple Receivers.

Message Delivery

- The message can be delivered synchronously or asynchronously to the multiple subscribers
- A “topic” is associated with each message
- Publishers “publish” messages to topic
- Subscribers “subscribe” to messages from the topic
- The subscribers can get message notifications

Message Reliability

- It is some guarantee of delivery of messages.
- Different level of reliability is possible.
- Higher reliability means less throughput
- The persistent storage (database/Files)is used to store the messages temporarily on the server side.

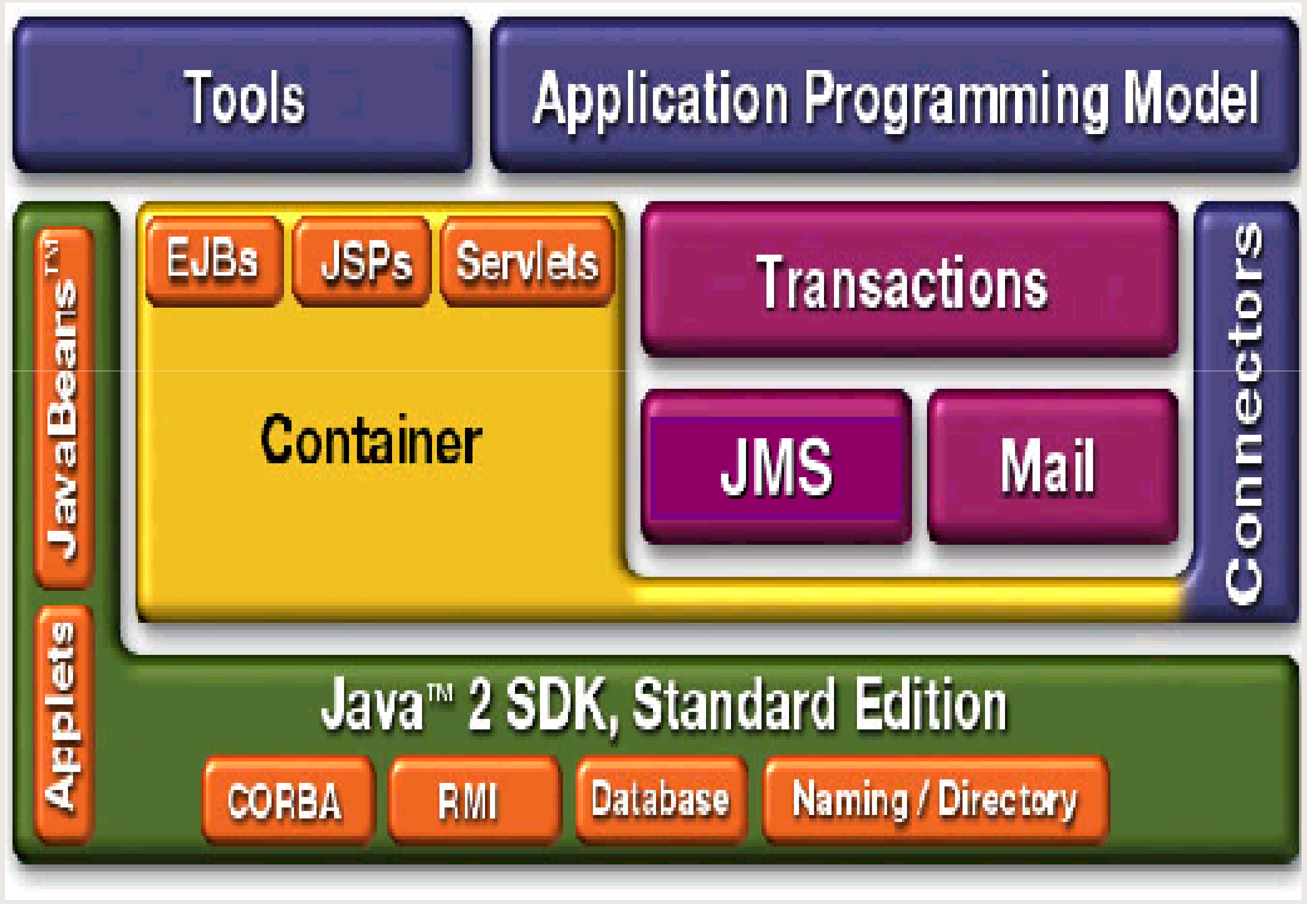
What is JMS ?

- JMS is j2ee specification and implementation for enterprise messaging services.
- Asynchronous messaging: A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them.
- Reliable Messaging : The JMS API can ensure that a message is delivered once and only once.
- Application clients, Enterprise JavaBeans (EJB^{TM}) components, and web components can interact with JMS service and send/receive synchronous/asynchronous messages.

JMS Goals

- Independent of Messaging system provider
- Minimal effort on part of Messaging system provider
- Consistency
- Provide most of the functionality of common messaging systems
- Leverage Java technology

JMS As a Service



JMS Features

- The Java Messaging Service is a standard API for messaging that supports reliable point-to-point messaging as well as the publish-subscribe model. This specification requires a JMS provider that implements both point-to- point messaging as well as publish-subscribe messaging.
- JMS supports following messaging features.

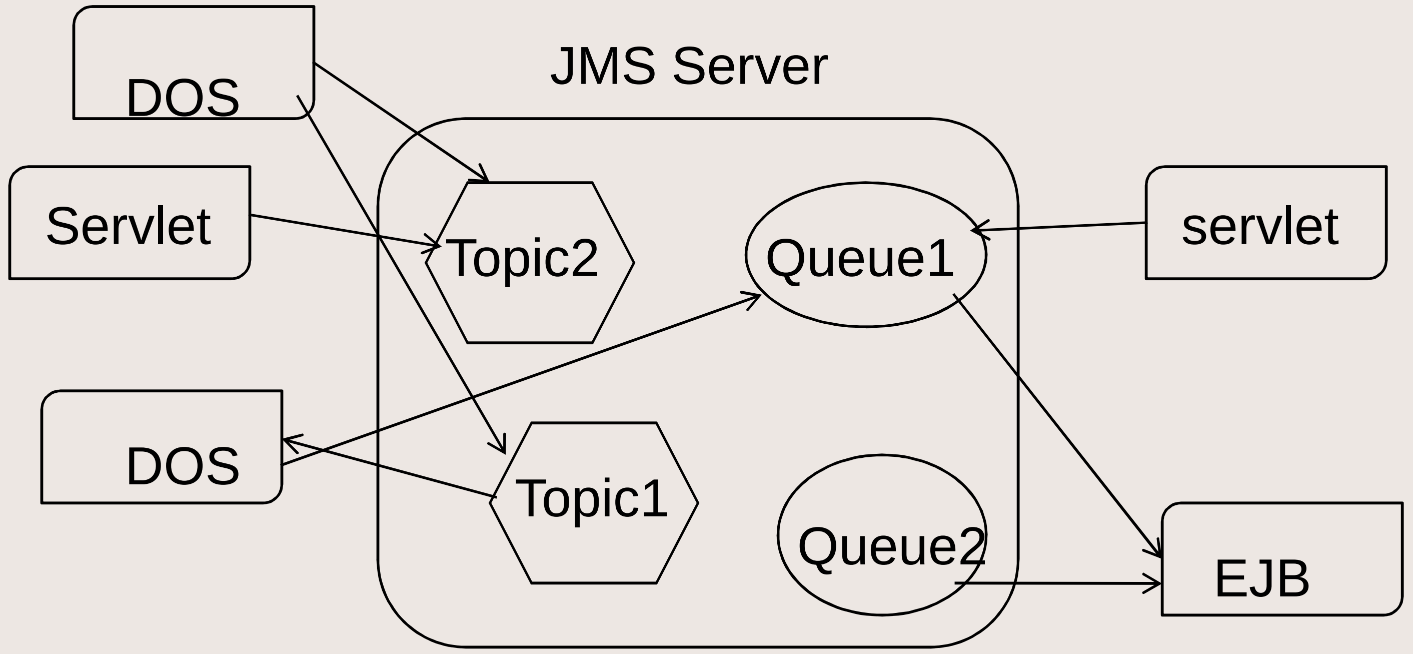
  secure transactions

  Guaranteed delivery

  Authentication and access control

  Load balancing

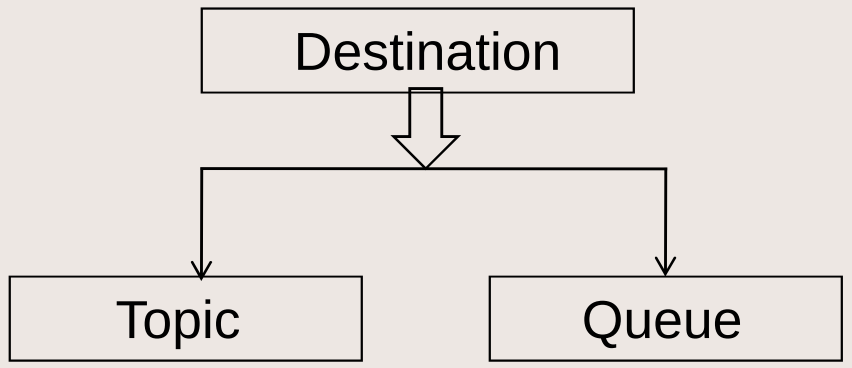
JMS Architecture



JMS Application components

- JMS provider : A JMS service provider
- Administered objects : Components interacting with the messages i.e. Topics and Queues.
- JMS clients :JMS Client applications
- Non-JMS clients : Other MOM middleware applications
- Messages :JMS provides a unified message model which maps to all message formats supported by providers.A message contains a message header,properties and body.

JMS Destinations



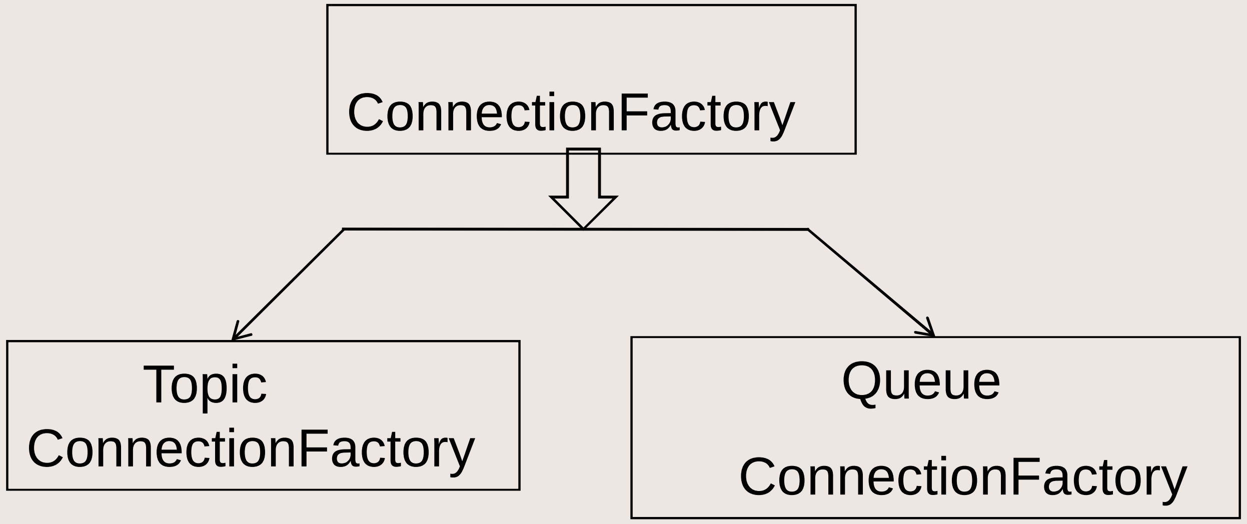
Topic and Queue are managed by the JMS Server

They are destinations for messages

JMS Message Destination

- A *destination* is the object a client uses to specify the target of messages it produces and the source of messages it consumes. In the PTP messaging ,destinations are called queues and in the pub/sub messaging, destinations are called topics.
- In JMS **Destination** is the base interface and **Topic** and **Queue** are sub interfaces extending from it.

JMS Drivers



The ConnectionFactory is used to create connection to the message destination a queue or topic

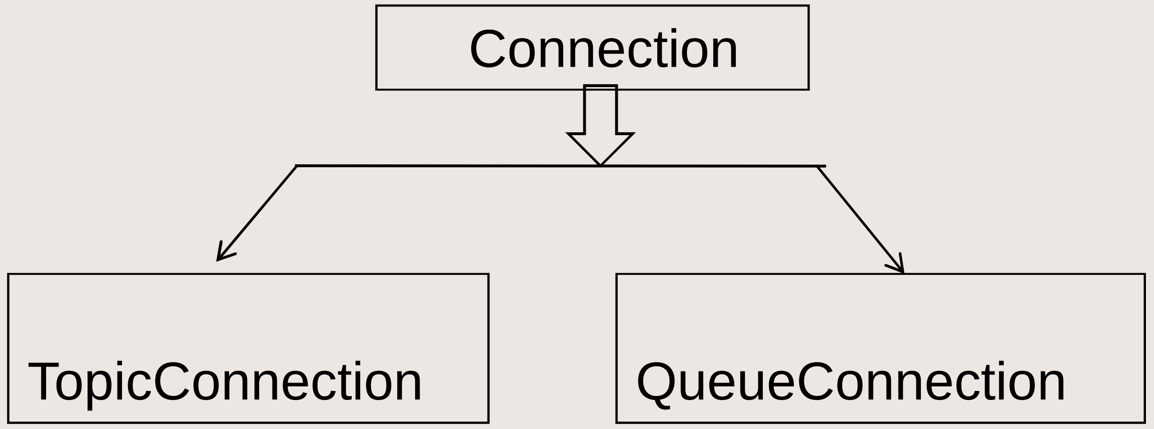
Understand the JMS

•JMS is protocol/standard.

•The ConnectionFactory is the specialized driver used to connect to JMS destination Topic/Queue.

•The Topic ConnectionFactory and QueueConnectionFactory are the drivers used to create connections to the Topic or Queue object

JMS Connections



Using the connection ,JMS clients can manage messages.

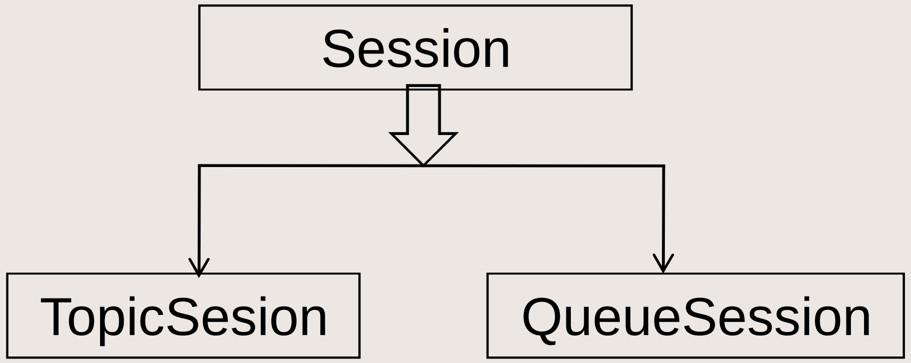
JMS Connection

- A *connection* represents a virtual connection with a JMS provider. A connection could represent an open TCP/IP socket between a client and a provider service.
- You use a connection to create one or more sessions.
- In JMS connection is specified as interface.
- ConnectionFactory objects are used to create a Connection to jms destination.
- Connection connection = connectionFactory.createConnection();

Creating a Connection

- Connection provides access to messaging system
- Performs resource allocation and management
- ConnectionFactory is located through JNDI lookup
- ConnectionFactory used to create Connection
- Connection to Queue/Topic is provided
- i.e.createQueueConnection() or createTopicConnection()
- This Connection is further used to create a jms session.

JMS Session



JMS Session is created from the JMS Connection

A JMS Session used to send and receive messages

JMS Sessions

- A *session* is a context for producing and consuming messages.
- Session is specified as interface in JMS
- Connections are used to create a one or more session objects.
- Sessions are used to create the following:
  – Message Producers
  – Message Consumers
  – Messages
  – Queue Browsers
  – Temporary queues and topics (see Creating Temporary Destinations)

Create JMS Sessions

- Create a jms session by using Connection.
- createQueueSession /createTopicSession
- Specify Transaction control and message acknowledgement for jms session.
- The session manages the message transactions on the Queue/Topic objects.
- Using the session create a MessageProducer or Message Consumer using the session object for current Topic/Queue.
- Using the session create the Message type object.
- Start the connection to start message processing on the connection object

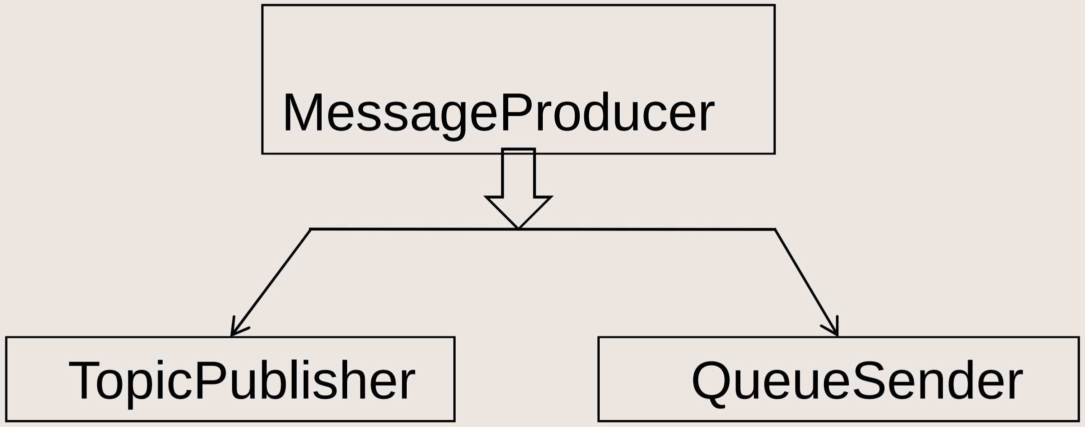
Message Acknowledgement

- How the destination objects know the message was delivered to the client ?
- Using acknowledgement mode.
- The acknowledgement can be sent automatically by the session object or application can send it when receiving process has received the message successfully.

Message Clients

- Use the Session to create
  - MessageProducers
  - MessageConsumers
  - Message.
- MessageProducers and MessageConsumers are further used to send and receive messages.

JMS Message Senders

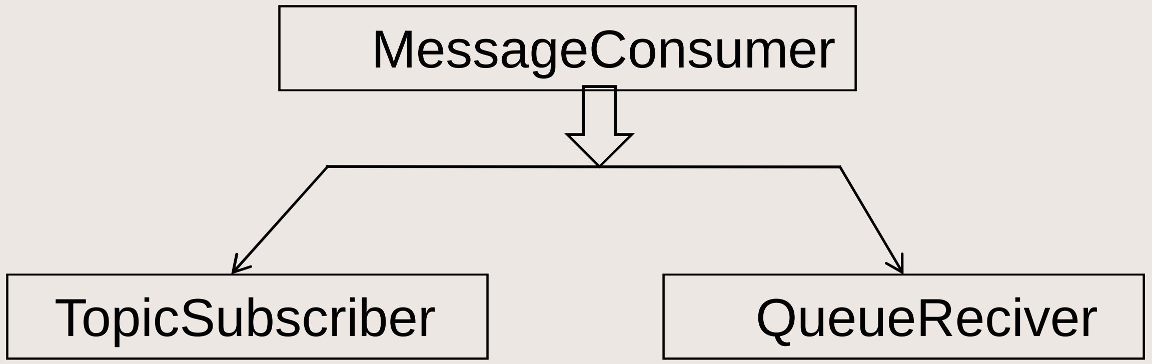


The TopicPublisher publishes message on the topic and the QueueSender sends the message to the Queue.

Message Producers

- A *message producer* is an object that is created by a session and used for sending messages to a destination.
- Is specified as MessageProducer interface.
- Use a Session to create a MessageProducer for a destination.
- MessageProducer producer = session.createProducer(dest);

JMS Message Receivers



TopicSubscriber subscribes to the messages from Topic and the QueueReciver receives messages from the Queue

Message Consumers

- A *message consumer* is an object that is created by a session and used for receiving messages sent to a destination.
- In jms it is specified as MessageConsumer interface.
- A message consumer allows a JMS client to register interest in a destination with a JMS provider.
- The JMS provider manages the delivery of messages from a destination to the registered consumers of the destination.

Message Listeners

- A *message listener* is an object that acts as an asynchronous event handler for messages.
- This is specified as MessageListener interface in JMS, which contains one method, onMessage().
- In the onMessage method, you define the actions to be taken when a message arrives.

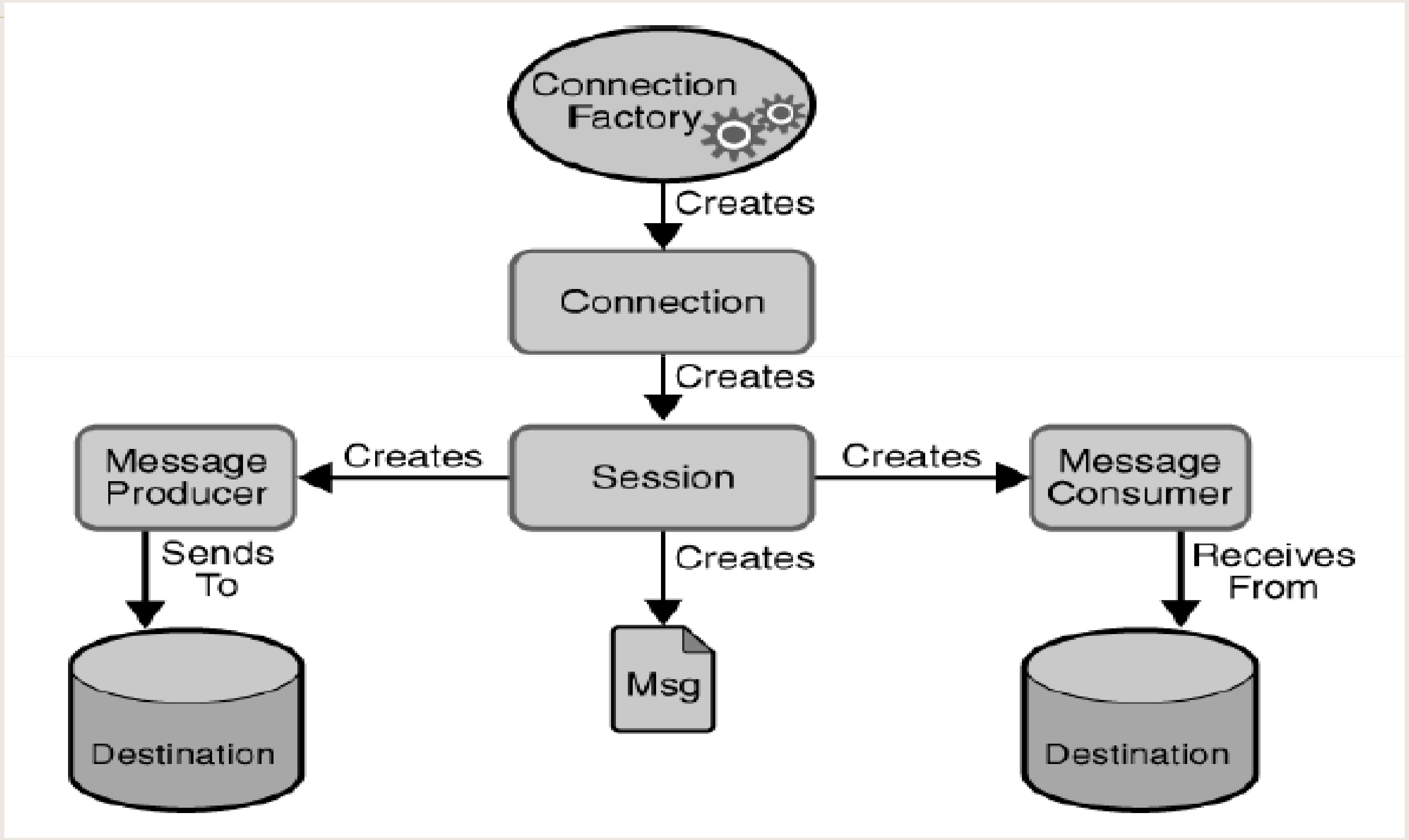
JMS Message

- JMS messages have a basic format that is simple but highly flexible, allowing to create messages that match formats used by non-JMS applications also.
  - A message has three parts
  - Message Headers
  - Message Properties (optional)
  - Message Bodies (optional)
- In JMS it is specified as Message interface.

JMS Message types

Message Type	Body Contains
`TextMessage`	A `java.lang.String` object (for example, the contents of an XML file).
`MapMessage`	A set of name-value pairs, with names as `String` objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
`BytesMessage`	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
`StreamMessage`	A stream of primitive values in the Java programming language, filled and read sequentially.
`ObjectMessage`	A `Serializable` object in the Java programming language.
`Message`	Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required.

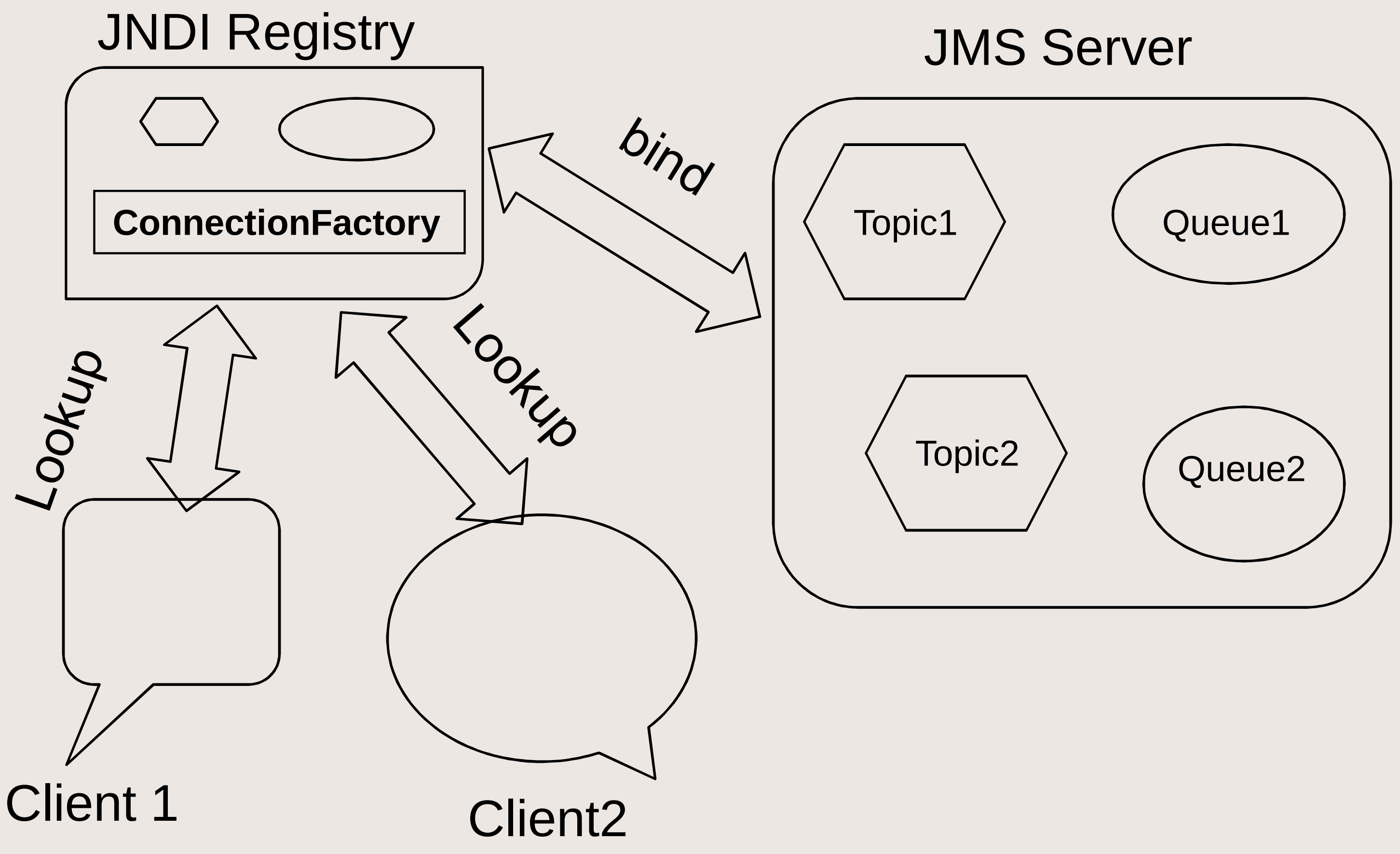
JMS Components



JMS Components Relations

JMS Parent	Publish-Subscribe Model	Point-To-Point Model
Destination	Topic	Queue
ConnectionFactory	TopicConnection Factory	QueueConnection Factory
Connection	TopicConnection	QueueConnection
Session	TopicSession	QueueSession
MessageProducer	TopicPublisher	QueueSender
MessageConsumer	TopicSubscriber	QueueReceiver QueueBrowser

JMS implementation



JMS Deployment Process

- Deploy the Topic/Queue
- Configure the destinations in JNDI Registry by names
- JMS server will manage messages received on the Topic/Queue automatically.

Creating a JMS Client

- Building a jms application client:
- Lookup for destination objects in JNDI Registry
- Look up for the ConnectionFactory objects
- Create a Connection to the provider
- Create Sessions to send/ receive messages
- Create MessageProducers to publish/send the message
- Create MessageConsumers to subscribe/receive messages
- Send and Receive messages

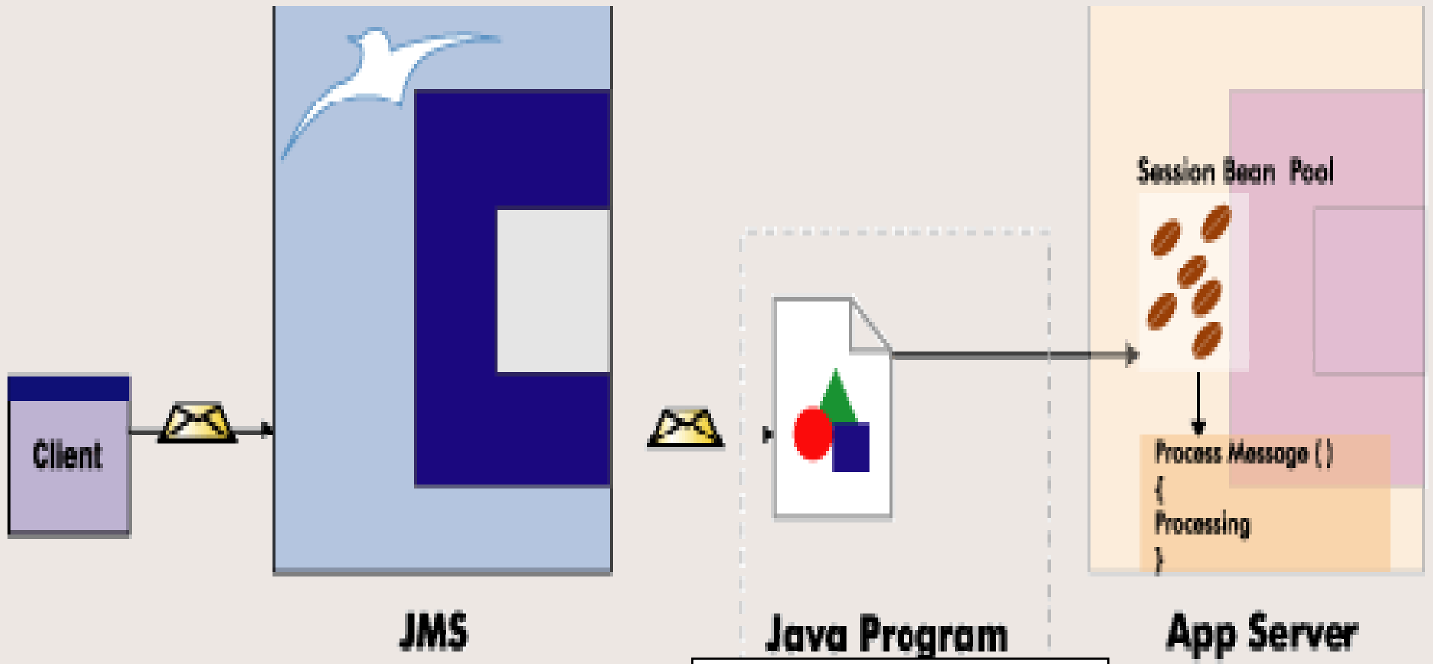
JMS Clients

- The MessageProducer is used to publish the message on topic or send the message on the queue.
- The MessageConsumer is used to receive the message from Queue or Subscribe to the Topic.
- Publisher will publish messages to specified Data topic.
- The MessageProducer is classified as TopicPublisher and QueueSender objects.
- Similiarly MessageConsumer is further classified as TopicSubscriber and QueueReceiver objects.
- TopicPublisher publishes a message on topic while QueueSender sends a message to queue
- TopicSubscriber and QueueReceiver receive the messages asynchronously or synchronously

Cleanup

- After message transfer is finished and use of objects is over close them just like jdbc objects.
- Close the producer/sender or receiver/subscriber.
- Close the session
- Close the connection.

JMS Integration

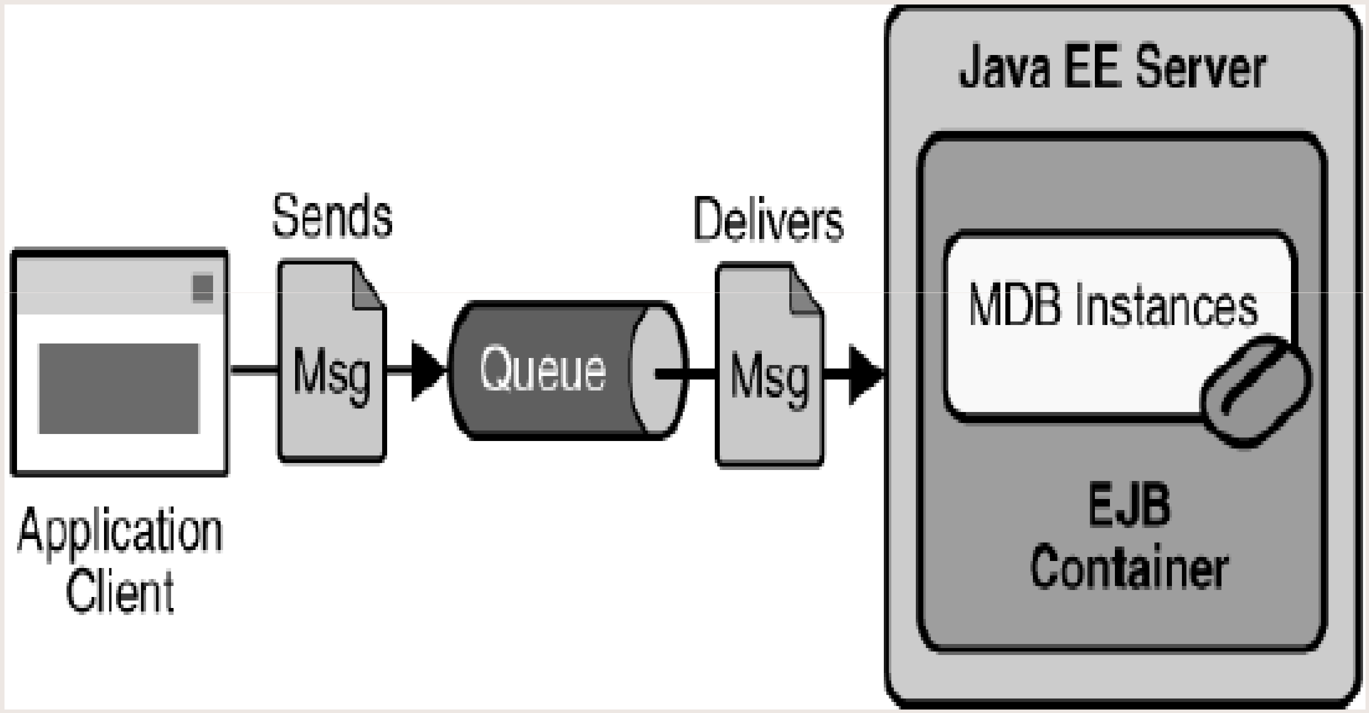


Listens to Topic
Message Pick up
SessionBean Looked Up
Business Method executed

JMS EJB

JMS supports a new type of ejb **MessageDriven** Bean for asynchronous message processing in Application Server.

Queue Receiver MDB



MDB and Other EJB

- The difference between message-driven beans and session and entity beans is that clients do not access message-driven beans through interfaces.
- Unlike a session or entity bean, a message-driven bean has only a single bean class.

MDB Instances Usage

- A message-driven bean's instances retain no data or conversational state for a specific client.
- All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance.
- The container can pool these instances to allow streams of messages to be processed concurrently.
- A single message-driven bean can process messages from multiple clients.

MDB Characteristics

- They execute upon receipt of a single client message.
- They are invoked asynchronously.
- They are relatively short-lived.
- They do not represent directly shared data in the database, but they can access and update this data.
- They can be transaction-aware.
- They are stateless.

When MDB ?

- Session beans allow you to send JMS messages and to receive them synchronously but not asynchronously.
- To avoid tying up server resources, you may prefer not to use blocking synchronous receives in a server-side component.
- To receive messages asynchronously, use a message-driven bean.

How to write MDB ?

The requirements of a message-driven bean class:

- It must implement the message listener interface for the message type it supports. A bean class should implement the javax.jms.MessageListener interface.
- The class must be defined as public.
- The class cannot be defined as abstract or final.
- It must contain a public constructor with no arguments.
- It must not define the finalize method.
- Unlike session beans and entities, message-driven beans do not have the remote or local interfaces that define client access.

MDB Clients

- Client applications do not locate message-driven beans references and invoke methods on them.
- The client will connect to Topic or Queue and publish or send message to it.
- In response to this message event at Topic or Queue, the container will invoke **OnMessage(Message msg)** method of MDB object.
- Although message-driven beans do not have business methods, they may contain helper methods that are invoked internally by the onMessage method.