EBU6501 - Middleware
Week 1, Day 1: Middleware and Message-Oriented Middleware

Dr. Gokop Goteng

Queen Mary
University of London

Lecture Aim and Outcome

- Aim
  - This lecture builds upon the lecture on messaging services to introduce message-oriented middleware (MOM)
- Outcome
  - At the end of this class students should be able to:
    - Design the architecture of MOM and know the functions of the parts of the architecture
    - Know the types of MOMs and their implementations
    - Understand the security function of MOMs in cloud computing

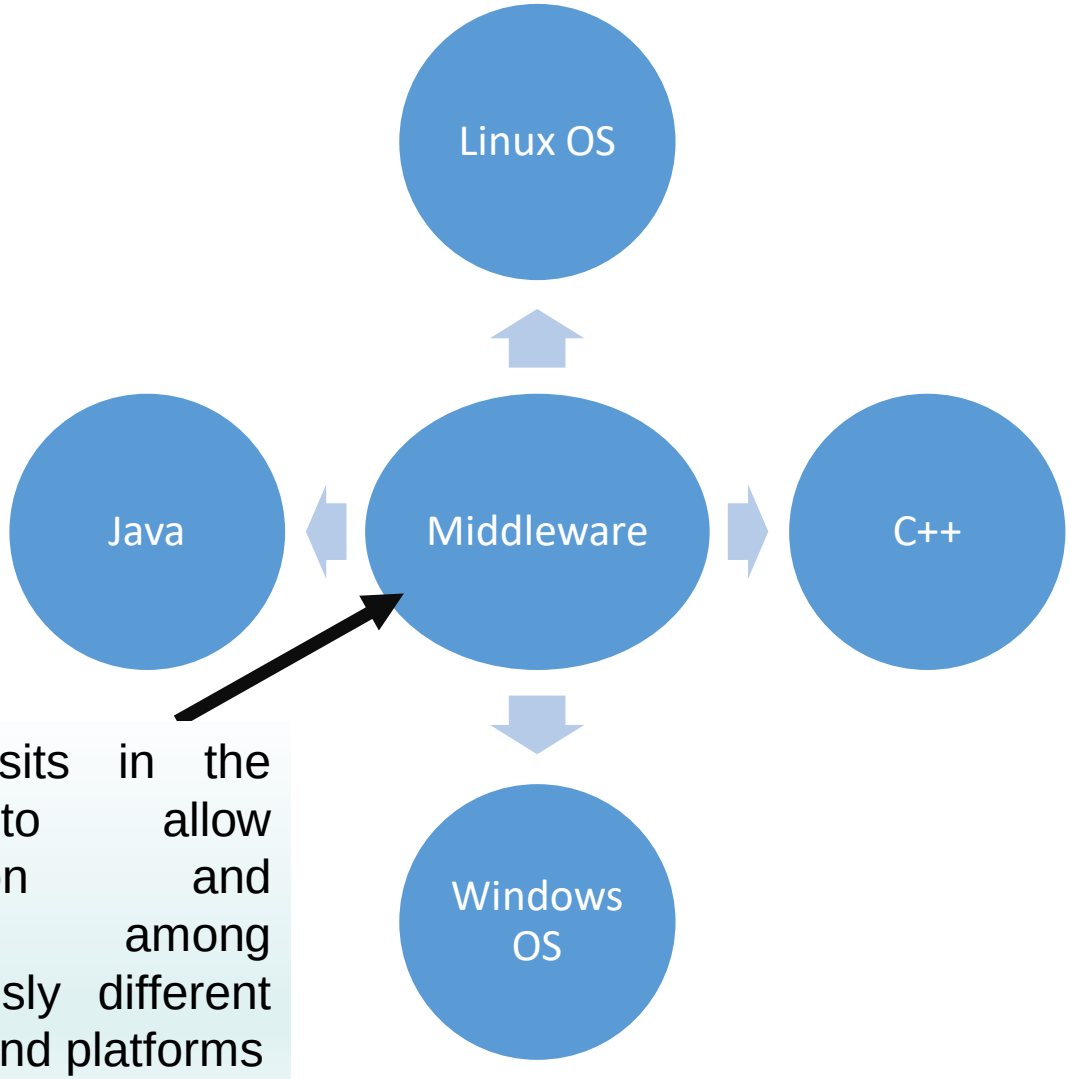
Lecture Outline

- Middleware
- Message-Oriented Middleware (MOM)
- MOM Architecture
- Types of MOM
- MOM Implementation Models
- Java Messaging Service (JMS)
- MOM's Messaging Models
- Advantages of MOM
- Types of Publish-Subscribe Message
- Security in MOMs
- Types of Queuing Systems in MOM
- Parts of a MOM Message
- Class Task

Middleware

- Middle is a software or hardware that allows two or more heterogeneous computing components to communicate and interact.
- Types of Middleware
  - Object-Oriented Middleware (OOM)
    - Remote Method Invocation (RMI)
    - Remote Procedure Call (RPC)
    - Object Request Broker (ORB)
      - Common Object Request Broker Architecture (CORBA)
  - Transaction Processing Monitor (TPM)
  - Reflective Middleware (RM)
  - Message-Oriented Middleware (MOM)

Middleware Concept



Middleware sits in the "middle" to allow communication and interaction among heterogeneously different applications and platforms

Why Do we Need a Middleware?

- Enterprise systems communicate and run on different platforms
  - Linux, Windows, Android, DOS, Mac, etc
- Software written in different languages run and communicate
  - Java, C/C++, C#, Basic, .Net, Eclips, NetBeans, etc
- Users may want to access or transfer data from different systems to different platforms
  - From Windows to Linux
    - WinSCP, Putty
  - From Linux to Windows
    - Remote Desktop Protocol (RDP)

Messaging & Cloud-Based Messaging Services

- What is messaging?
  - Messaging is a method of communication between software components or applications.
  - A messaging system is a peer-to-peer facility which has messaging client that 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
  - Messaging enables distributed communication that is loosely coupled
  - A component sends a message to a destination and the recipient can retrieve the message from the destination
  - The sender and the receiver do not have to be available at the same time to communicate
  - The sender and the receiver do not need to know anything about each other apart from what message format and destination to use

Messaging & Cloud-Based Messaging Services

- What is messaging?...
  - This is different from tightly coupled communication technologies such as the Remote Method Invocation (RMI) which requires an application to know a remote application's methods.
  - Messaging also differs from electronic mail (e-mail), which is a method of communication between people or between software applications and people.
    - Messaging is used for communication between software applications or software components
- Messaging Service
  - A message service is a software component functionality based on service-oriented architecture which allows users to create, send, receive, read, delete and save messages in form of texts, audio, pictures and video from one software component to another and vice versa.
  - It is a form of peer-to-peer (P2P) communication
    - Each component behaves both as a client as well as a server
  - The sender only needs to know the data format to send and destination of the receiver
  - It is a different form of communication than email
  - It is a loosely coupled communication so it is different than Remote Method Invocation (RMI) which is tightly coupled where the sending application must know the receiving application's method.

Advantages of Messaging

- Heterogeneous Integration
  - The most classic use of messaging is the communication and integration of heterogeneous platforms.
  - This means that using messaging, you can invoke services from applications and systems that are implemented in completely different platforms
  - Many open source and commercial messaging systems provide seamless connectivity between Java and other languages and plaforms (eg Java on Linux and C++ on Windows can communicate using messaging)
  - This means that messaging is a form of ***middleware***
- Reduce System Bottlenecks
  - System and application bottlenecks occur whenever you have a process that cannot keep up with the rate of requests made to that process
- Increase Scalability
  - Scalability in messaging systems is achieved by introducing multiple message receivers that can process different messages concurrently.

Cloud-Based Messaging Services

- Cloud Messaging Service
  - This is a large-scale messaging service running on a cloud solution that allows users to create, send, receive and read data/information from distributed heterogeneous systems.
  - It usually uses middleware to communicate to heterogeneous systems
    - Message-Oriented Middleware (MOM) is a common term in cloud-based messaging services.

Types of Messaging Services

- Short Message Service (SMS)
  - This is a text messaging service for phones, web and mobile systems.
  - It uses standard communication protocols
  - It allows fixed line or mobile phones to send and receive text messages
- Instant Message Service (IMS)
  - This is an online chart messaging system which allows real-time communication of text conversations over the internet
- Multimedia Message Service (MMS)
  - This is a messaging service that extends SMS to send messages that include multimedia (audio, video, pictures, text) to and from mobile phones

Desktop and Mobile Apps with Messaging Services

- WhatsApp messaging service application
- Viber messaging service application
- Skype messaging service application
- Facebook Messenger application
- Hipchat messaging service application

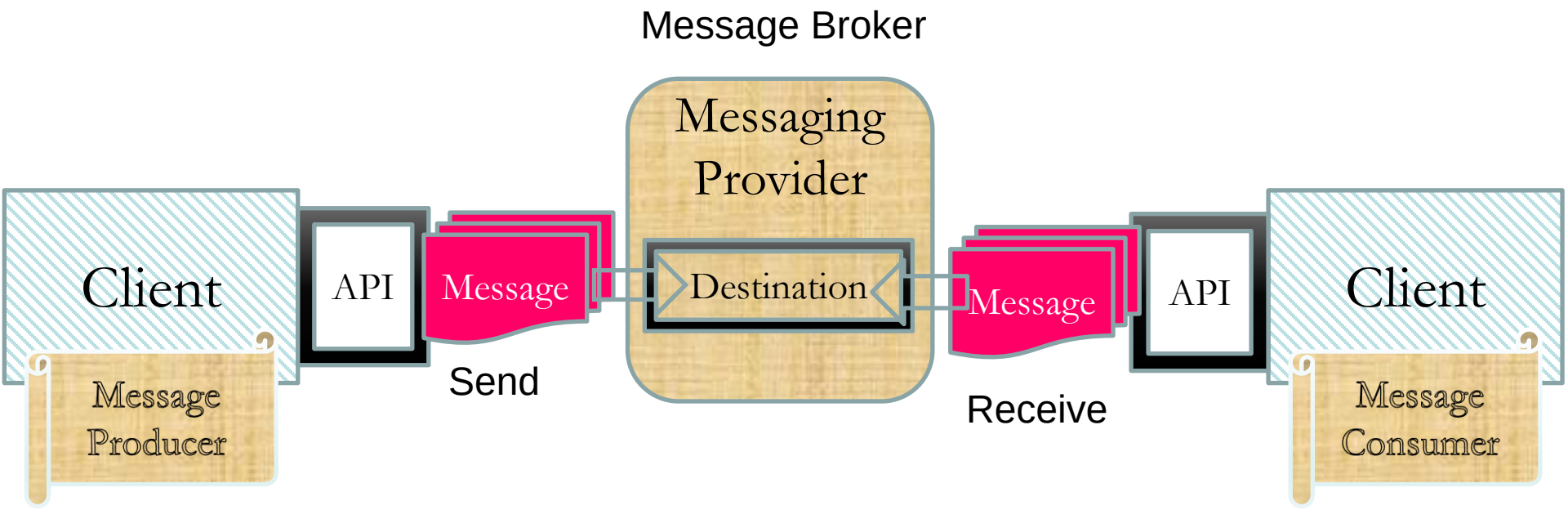
Examples of Cloud-Based Messaging Services

- Google Cloud Messaging (GCM) Service
  - It is a cloud messaging service for devices that use Android and Chrome to send and receive messages from distributed servers
- Oracle Messaging Cloud (OMC) Services
  - It is a distributed infrastructure that allows software components to send and receive messages through a single API
  - It establishes a dynamic and automated business workflow environment
- Java Messaging Service (JMS)
  - It is a Java API based on Java message-oriented middleware (MOM) for sending messages between two or more distributed clients.
- Apache Qpid
  - This is an open-source messaging system that implements the Advanced Message Queueing Protocol (AMQP) and runs on many computer languages and platforms.
  - It is based on a point-to-point and brokered messaging system
- Xiaomi MIUI Cloud Messaging Service
  - This is the messaging service for the Chinese Xiaomi smartphone company.
  - It is offered as a free service within the MIUI operating system.

Message-Oriented Middleware (MOM)

- Message-oriented Middleware (MOM) is a software or hardware that provides a communication channel between heterogeneous applications, platforms and systems by passing messages across between the communicating systems.
  - MOM provides capabilities for sending and receiving messages among distributed systems
  - MOM uses the eXtensible Messaging and Presence Protocol (XMPP) for communication
    - This is based on the XML publish/subscribe standards
    - The XMPP Working group recommends four standard specifications for implementation which can be found at http://tools.ietf.org/html/rfc6120, http://tools.ietf.org/html/rfc6121, http://tools.ietf.org/html/rfc3922 and http://tools.ietf.org/html/rfc3923 respectively.
  - MOM provides asynchronous message-based communication between modules and processes

MOM Architecture



MOM Client - Producer

- The “MessageProducer” object is used by the client to produce a message
- The message is then sent by the message producer
- The message is usually sent to a physical destination which is written in the API using the destination object

Message Broker

- The broker is responsible for receiving and sending the message
- The broker facilitates the connections between producers, administration services and consumers
- The broker maintains a persistent data store to reliably send data
- The broker provides secure connection using encrypted SSL authenticated connection.
- The broker monitors message and connection activities to provide logs of diagnostics information for administrators to access during troubleshooting

MOM Client - Consumer

- Because MOM uses asynchronous consumption method, the message is automatically given to “MessageListener” object that is defined by the consumer
- Alternatively, messages can be consumed by only the messages that match the properties selected by a consumer

Types of MOM

- Message Queuing
- Publish/Subscribe

MOM Implementation Models

- Microsoft's Message Queuing (MSMQ)
- Oracle's Java Messaging Service (JMS)
- IBM's WebSphere Message Queuing (WebSphere MQ)
- Sonic Message Queuing (Sonic MQ)

MOM's Messaging Models

- Point-to-Point (P2P)
  - This is a model in which one sender sends a message to only one receiver at a time
  - It is a one-to-one relationship
  - Client sends a message to a particular queue
  - Messages are sent to the particular address specified by the client
  - The receiver needs to be aware that the message is being sent
- Publish/Subscribe (Pub/Sub)
  - This is a model which allows either one sender sends one message that is received by many receivers or many messages are sent and are received by many receives
  - It is either a one-to-many or many-to-many relationships
  - The sender and receiver of messages are decoupled
    - Receivers are anonymous
    - There is no need for the producer and consumers to be running at the same time
  - This offers the best solution for distributed and dynamic cloud systems

Advantages of MOM

- Hides the complexity of applications running on different operating systems
- The mode of sending and receiving messages is reliable
- MOM embeds security features so that messages cannot be intercepted and changed
- It sends messages asynchronously and so the receiver does not have to know that messages are being sent prior to sending them
- Allows heterogeneous systems to communicate
- Allows different programming languages to be ported (portability) across different platforms

Types of Publish-Subscribe Message

- Topics-Based
- Content-Based
- Type-Based

Publish/Subscribe Model

Publish/Subscribe: Topics-Based

- It uses a hierarchical naming scheme
  - There is a logical arrangement in the way the topics appear using some form of classification method
  - It uses specific terms to describe the topics
    - Political, mathematics, scientific, etc
  - It uses dedicated queues
- Disadvantage
  - It is too restrictive as it uses specific terms

Publish/Subscribe: Content-Based

- It does not use specific terms to describe the content
- It uses commonly used query-like syntax for subscription of contents
- Advantage
  - It is less restrictive as it does not use specific terms for subscriptions

Publish/Subscribe: Type-Based

- Subscribers need to register for the type of message the want
- The messages are represented as objects
- Disadvantage
  - It is also restrictive as subscribers need to register for the type of messages they want to subscribe

Security in MOMs

- There are implementation of access controls between the client and the API for sending and receiving messages
  - IBM uses System Authorisation Facility (SAF) to control access to MOM
- Encrypted Messages
  - The Qpid Advanced Message Queuing Protocol (AMQP) uses SASL/TLS (Simple Authentication and Security Layer/Transport Layer Security) for authentication and encryption of messages in Java Messaging Service (JMS)

Types of Queuing Systems in MOM

- First-In-First-Out (FIFO)
  - The first message sent will be the first to be received and the last will be the last to be received
- FIFO with Priorities
  - It is the same as FIFO but the message sent/received depends on the priority set for the publisher and consumer
- Journal Queue
  - The system keeps track and copy of all messages
- Public Queue
  - It is open for access to all users
- Private Queue
  - It requires authentication and authorisation to use the queue
- Connector/Bridge
  - This is a proxy to connect to commercial MOM's queues

Parts of a MOM Message

- A MOM Message Consists of 3 Parts
  - Header
    - A header consists of information about the message destination, message expiration time, message unique identification, message delivery mode (persistent or non-persistent) and message time stamp.
  - Properties
    - Consists of some information in the header
    - Each property item is a tuple consisting of property name and property value pair
    - The properties are used to extend the functionality of the header
    - Programmers can create properties to describe the message they want to send or receive
  - Body
    - The body contents the message and how it will be sent and received

Java Message Service (JMS)

- What is JMS?
  - The Java Message Service (JMS) is a Java Application Programming Interface (API) that allows applications to create, send, receive and read messages
  - Designed originally by Sun and several partner companies, the JMS API defines a common set of interfaces and associated semantics that allow programs written in Java programming language to communicate with other messaging implementations
  - The JMS API minimises the set of concepts programmers need to learn to use messaging products
  - JMS API enables loosely coupled communication as well as Asynchronous communication
    - Asynchronous communication means JMS provider can deliver messages to a client as they arrive, meaning a client does not have to request messages in order to receive them
  - The JMS API is reliable as it ensures that a message is delivered once and only once

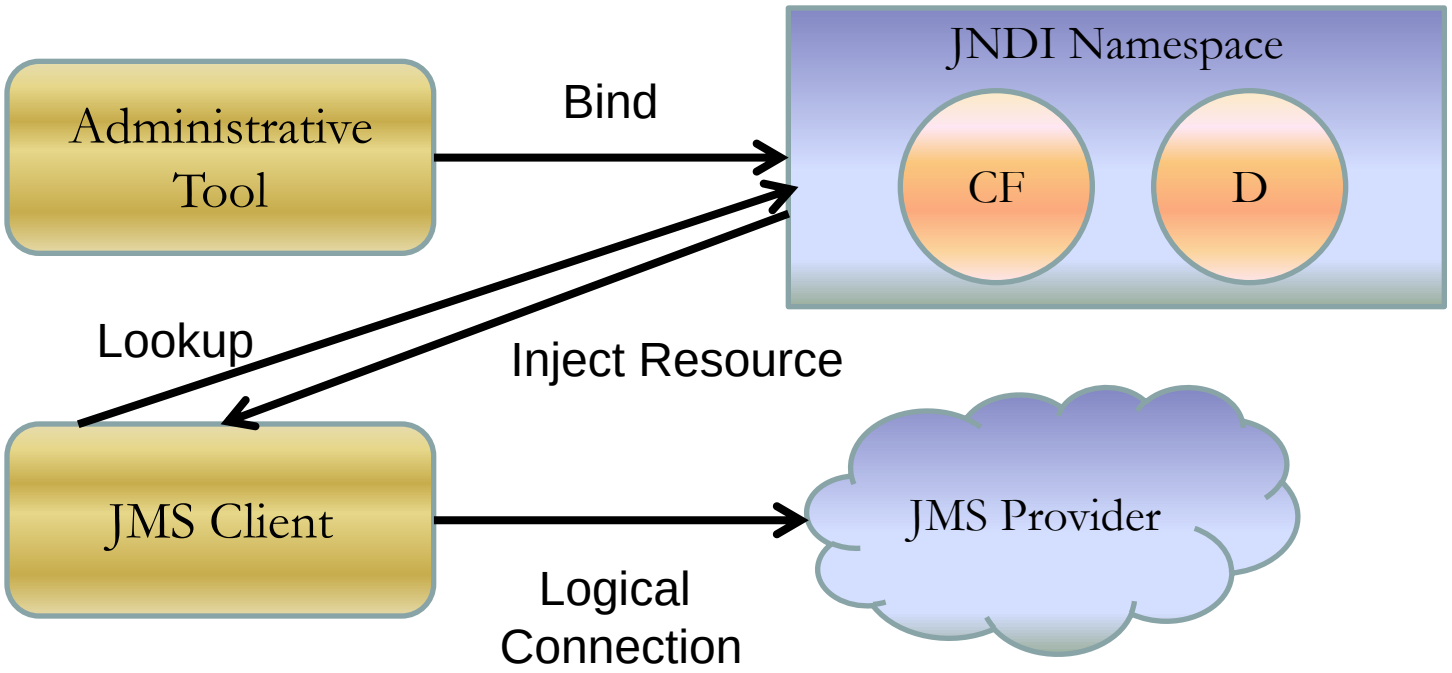
When To Use JMS API?

- The enterprise application provider wants the components not to depend on information about components' interfaces, so that components can be easily replaced (loose coupling)
- The provider wants the application to run whether or not all components are up and running simultaneously (asynchronous communication)
- The application business model allows a component to send information to another and to continue to operate without receiving an immediate response
  - Applications in manufacturing where you need to send messages on inventory of goods in stock to order more goods when the stock is low can use JMS API

How Does the JMS API Work with J2EE Platform?

- JMS API in Java 2 Enterprise Edition 1.3 (J2EE 1.3) has these features
  - Application clients, Enterprise JavaBeans(EJB) components and Web components can send or synchronously receive a JMS message
    - In addition, application clients can receive JMS messages asynchronously
  - A kind of enterprise bean known as the message-driven bean, enables the asynchronous consumption of messages
    - A JMS provider may optionally implement concurrent processing of messages by message-driven beans
  - Message sends and receives can participate in distributed transactions
- The addition of JMS API enhances the J2EE platform by simplifying enterprise development, allowing loosely coupled, reliable and asynchronous interactions among J2EE components and legacy systems capable of messaging

Java Messaging Service (JMS) API Architecture



Keys:

CF = Connection Factories
D = Destinations
JNDI = Java Naming and Directory Interface

Java Messaging Service (JMS) API Architecture

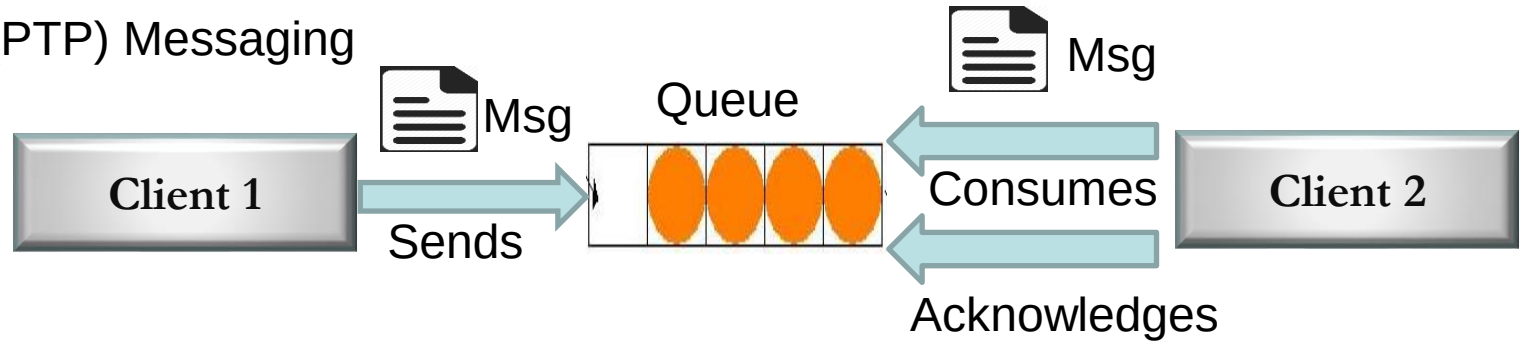
- JMS Provider
  - This is the messaging system which implements the JMS interfaces and provides administrative and control functionalities
  - Java EE platforms consist of JMS Provider
- JMS Clients
  - These are java programs or components that produce and consume messages
  - Any Java EE component can act as a JMS Client
- Messages
  - These are the objects that communicate information between JMS Clients
- Administered Objects
  - Preconfigured JMS objects created by an administrator for the use of clients
  - The two types of JMS administered objects are Destinations (D) and Connection Factories (CN)
- JNDI Namespace
  - This is a Java directory service API that allows Java program clients to discover and look up data and objects through a name.
  - It connects a java application to external directory services such as LDAP (Lightweight Directory Access Protocol)
  - It also allows Java Servlets to lookup configuration information provided by the hosting web container such as Apache Tomcat configuration file.

Java Messaging Domains

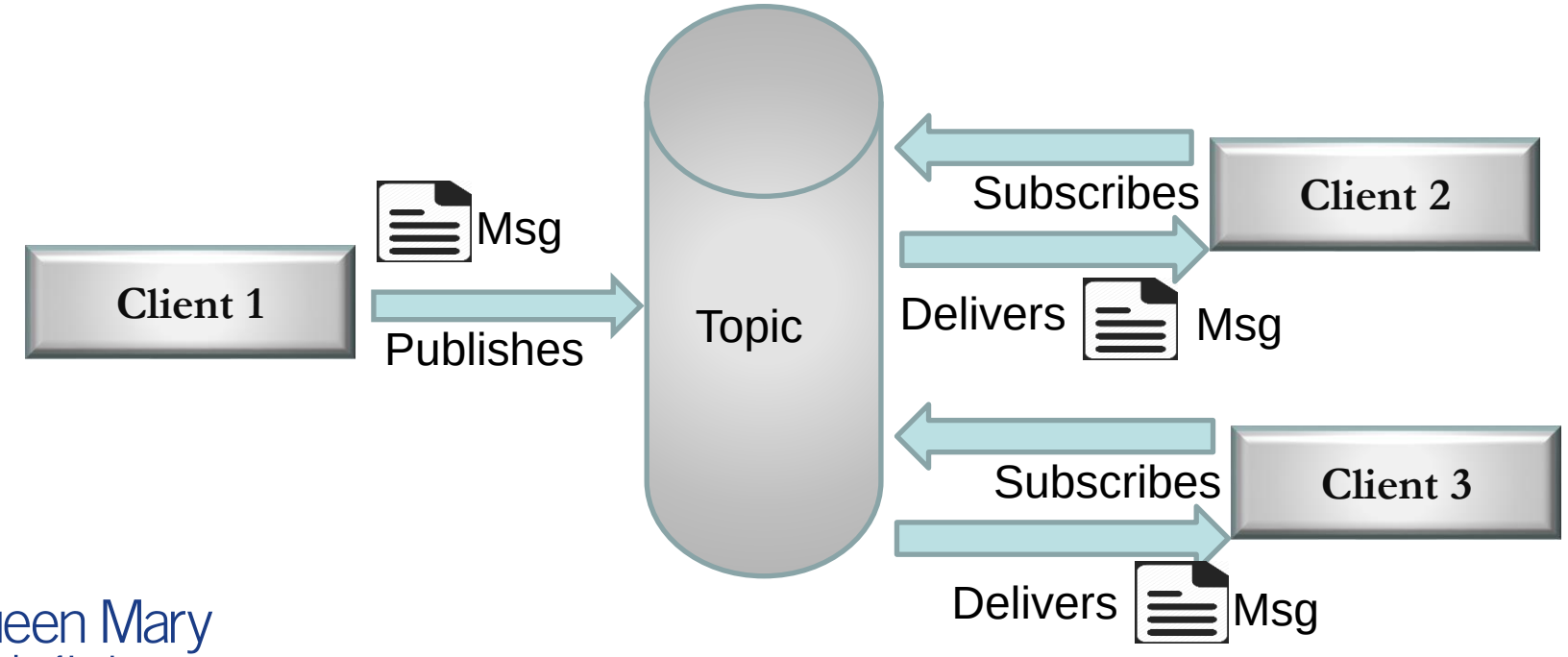
- Messaging Domain
  - Messaging domain is the concept of how message is published and consumed with regards to the systems used by the provider and clients.
- Types of Java Messaging Domains
  - Point-to-point (PTP) messaging domain
    - This is based on the message queues, senders and receivers concept.
    - A message is sent to a specific queue so that receiving clients can extract the message from the queue that they know have their messages
    - Queues will hold messages until they are consumed or they reach their expiration threshold
  - Publish/Subscribe messaging domain
    - This domain allows clients to address messages to a topic that functions similar to a bulletin board
    - Publishers and subscribers are anonymous and can publish/subscribe dynamically
    - Each message can have multiple clients
    - Publishers and subscribers have a timing dependency
      - A client that subscribes for a topic can only consume messages that are published after the client has created a subscription.

PTP and Pub/Sub Domains

1. Point-to-point (PTP) Messaging



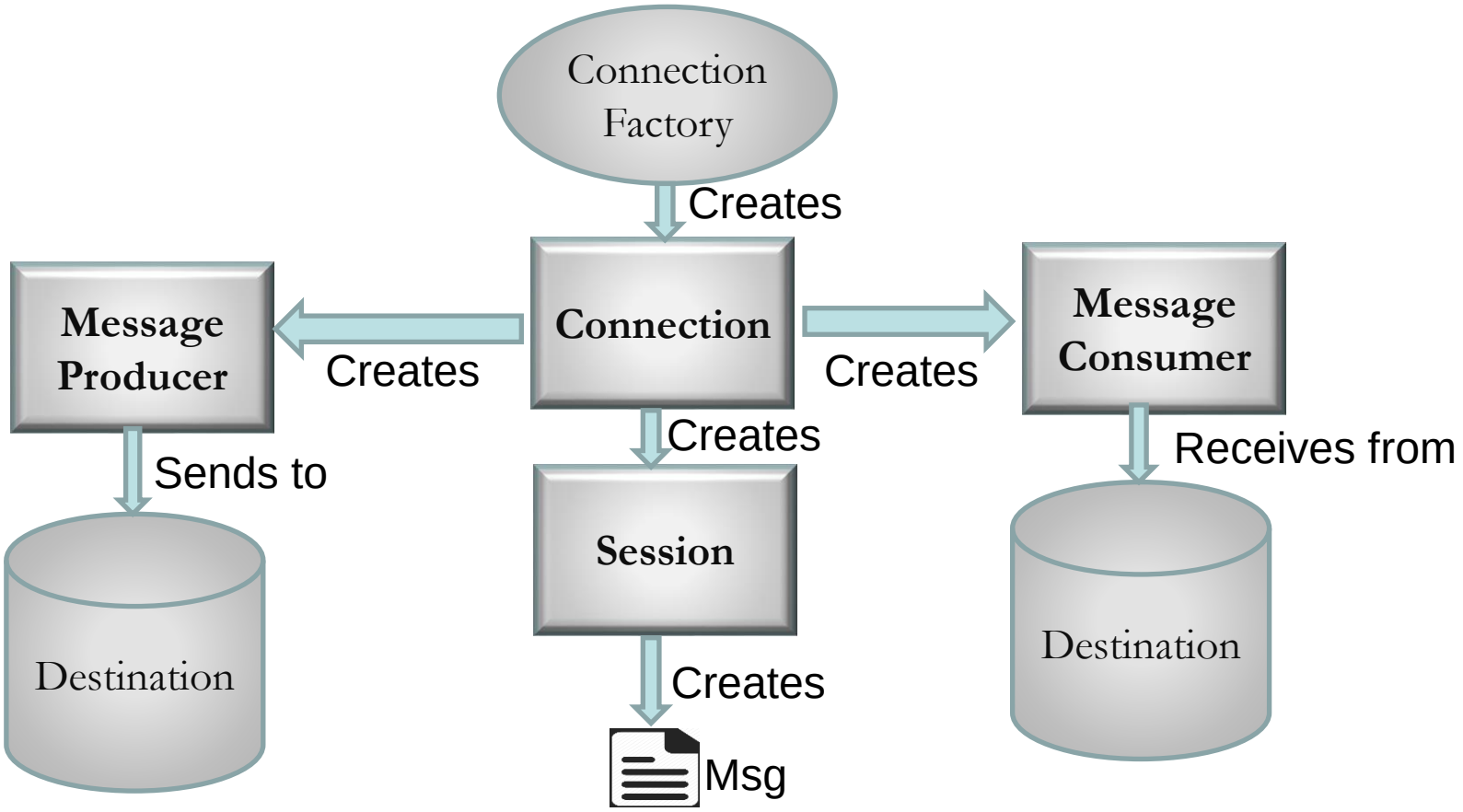
2. Publish/Subscribe (Pub/Sub) Messaging



Types of JMS Message Consumption

- Message products are typically asynchronous in that no fundamental timing dependency exists between the production and consumption of a message
- However, the JMS Specification can implement message consumption the following two ways:
  - Synchronously:
    - A 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 message does not arrive within a specified time limit
  - Asynchronously:
    - A client can register a message listener with a consumer
      - A message listener is similar to an event listener
    - Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's onMessage method, which acts on the contents of the message

The JMS API Programming Model



The JMS API Programming Model

- Administered objects
  - Two parts of the JMS application namely destinations (D) and connection factories (CF) are best maintained administratively rather than programmatically
  - The technology underlying these objects is likely to be very different from one implementation of the JMS API to another
  - Therefore the management of these objects belongs with other administrative tasks that vary from provider to provider
  - With the J2EE 1.3 SDK version 1.3, you use a tool called ***j2eeadmin*** to perform administrative tasks
  - For help to use this tool, type ***j2eeadmin*** and press enter (with no arguments)
- Connection Factories (CF)
  - A connection factory is the object a client uses to create a connection with a provider
  - As said above, it is an administrative task which encapsulates connection configuration parameters
  - A pair of connection factories come preconfigured with the J2EE SDK and are accessible as soon as you start the service
  - Each connection factory is an instance of either the ***QueueConnectionFactory*** or the ***TopicConnectionFactory*** interface
  - In J2EE SDK, you can use the default CF objects ***QueueConnectionFactory*** and ***TopicConnectionFactory*** to create connections
  - You can also use the following commands to create new connection factories
    - J2eeadmin –addJmsFactory jndi_name queue
    - j2eeadmin –addJmsFactory jndi_name topic

The JMS API Programming Model

- Connection Factories (CF)
  - A connection factory is the object a client uses to create a connection with a provider
  - A connection factory (CF) encapsulates a set of connection configuration paramerers has been defined an administrator
  - A pair of connection factories come preconfigured with the J2EE SDK and are accessible as soon as you start the service
  - Each CF is an instance of either the QueueConnectionFactory or the TopicConnectionFactory interface
  - In J2EE SDK, you can use the default CF objects named QueueConnectionFactory and TopicConnectionFactory to create connections
  - You can also create new connection factories using the following commands:
    - ***j2eeadmin –addJmsFactory jndi_name queue***
    - ***j2eeadmin –addJmsFactory jndi_name topic***

The JMS API Programming Model

- Connection Factories (CF)…
  - At the beginning of a JMS client program, you perform a JNDI API lookup of the connection factory. The following code does that:

```
Context ctx = new InitialContext();

QueueConnectionFactory queueConnectionFactory =
(QueueConnectionFactory)  ctx.lookup("QueueConnectionFactory");

TopicConnectionFactory topicConnectionFactory = (TopicConnectionFactory)
ctx.lookup("TopicConnectionFactory");
```

  - Calling the InitialContext method with no arguments results in a search of the current classpath for vendor-specific file named ***jndi.properties***

The JMS API Programming Model

- Destinations
  - A destination is the object a client uses to specify the target of messages it produces and the source of message it consumes
  - The PTP and Pub/Sub in J2EE SDK, you can create destination objects using the following commands respectively for a queue and topic:
    - ***j2eeadmin –addJmsDestination queue_name queue***
    - ***j2eeadmin –addJmsDestination topic_name topic***
  - A JMS application may use multiple queues and/or topics
  - In addition to looking up connection factory, you also usually look up for destination
  - For example the code below performs a JNDI API lookup of a previously created topic "MyTopic" and assigns to a Topic object:
    - ***Topic myTopic = (Topic) ctx.lookup("MyTopic");***
  - The following code looks up a queue named "MyQueue" and assigns it to a Queue object:
    - ***Queue myQueue = (Queue) ctx.lookup("MyQueue");***

The JMS API Programming Model

- Connections
  - A connection encapsulates a virtual connection with a JMS provider
  - A connection could represention an open TCP/IP socket between a client and a provider service daemon
  - You use connections to create one or more sessions
  - Comes come in two forms like CFs, namely QueueConnection and TopicConnection interfaces
  - Once you have a QueueConnectionFactory or a TopicConnectionFactory object can use it to create a connection as:
    - ***QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();***
    - ***TopicConnection topicConnection = topicConnectionFactory.createTopicConnection();***
  - Before your application can consume messages, you must call the connection's "***start***" method

The JMS API Programming Model

- Sessions
  - A session is a single threaded context for producing and consuming messages
  - You use sessions to create message producers, message consumers and messages.
  - There are two forms by implementing the QueueSession or the TopicSession interfaces.
  - For example if you create a TopicConnection object, you use it to create a TopicSession as:
    - TopicSession topicSession = topicConnection.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
      - The first argument is a Boolean and in this case it is false, meaning the session is not transacted and the second argument means session automatically acknowledges messages when they have been received successfully
    - Class work: write the command to create a session for a queue and find out different types of arguments to be used

The JMS API Programming Model

- Message Producers
  - A message producer is an object created by a session and is used for sending messages to a destination
  - The PTP message producer implements the ***QueueSender*** interface while the Pub/Sub message producer implements the ***TopicPublisher*** interface
  - For example, you use ***QueueSession*** to create a sender for the queue myQueue and you use ***TopicSession*** to create a publisher for the topic myTopic as:
    - ***QueueSender queueSender = queueSession.createSender(myQueue);***
    - ***TopicPublisher topicPublisher = queueSession.createPublisher(myTopic);***
  - Once you have created a message producer, you can use it to send messages
    - Note that you must first create the message to do this
  - With a QueueSender you use the ***send*** method and with a TopicPublisher, you use the ***publish*** method as follows:
    - ***queueSender.send(message);***
    - ***topicPublisher.publish(message);***

The JMS API Programming Model

- Message Consumers
  - A message consumer is an object created by a session and is used for receiving messages sent to a destination
  - 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
  - The PTP message consumer implements the QueueReceiver interface and the Pub/Sub message consumer implements the TopicSubscriber interface to consume messages
  - For example, you use ***QueueSession*** to create a receiver for the queue myQueue and you use ***TopicSession*** to create a subscriber for the topic myTopic as:
    - QueueReceiver queueReceiver = queueSession.createReceiver(myQueue);
    - TopicSubscriber TopicSubscriber = topicSession.createSubscriber(myTopic);
  - Use the ***receive*** method for both TPT and Pubb/Sub to consume messages synchronously. You can use the receive method anytime you have called the ***start*** method as:
    - ***queueConnection.start();***
    - ***Message m = queueReceiver.receive();***
    - ***topicConnection.start(0;***
    - Message m = topicSubscriber.receive(1000); // timeout after a second
  - To consume a message synchronously, you use a message listener

The JMS API Programming Model

- Message Listeners
  - A message listener is an object that acts as an asynchronous event handler for messages
  - It implements the MessageListener interface, which contains one method called ***onMessage***
  - In the ***onMessage*** method, you define the actions to be taken when the message arrives
  - You register the message listener with a specific QueueReceiver or TopicSubscriber by using the ***setMessageListener*** method
  - For example:
    - TopicListener topicListener = new TopicListener();
    - topicSubscriber.setMessageListener(topicListener);

The JMS API Programming Model

- Message Selectors
  - If your messaging application needs to filter the message it receives, you can use a JMS API message selector, which allows a message consumer to specify the messages it is interested in
- Messages
  - The ultimate purpose and goal of a JMS application is to produce and to consume messages that can then be used by other software applications
  - JMS messages have a format with the following parts:
    - Header (needed)
    - Properties (optional)
    - A body (optional)

The JMS API Programming Model

- Messages…
  - Message Headers
    - A JMS message header contains a number of predefined fields that contain values that both clients and producers use to identify and route messages
    - Every message has a unique identidier represented by the header field JMSMessageID
    - The value of the header field JMSDestination represents the queue or the topic to which the message is sent

The JMS API Programming Model

- JMS Message Header Field Values are Set:

Header Field	Set by
JMSDestination	send or publish method
JMSDeliveryMode	send or publish method
JMSPriority	send or publish method
JMSMessageID	send or publish method
JMSTimestamp	send or publish method
JMSExpiration	send or publish method
JMSCorrelationID	Client
JMSReplyTo	Client
JMSType	Client
JMSRedelivered	JMS provider

The JMS API Programming Model

- Messages…
  - Message Properties
    - You can create and set properties for a message if you need values in addition to those provided by the header fields
    - You can use properties to provide compatibility with other messaging systems or you can use them to create message selectors
  - Message Bodies
    - The JMS API defines five message body formats also called message types.
    - Message bodies allow you to send and receive data in many different forms and provides compatibility with existing messaging formats
    - The JMS API provides methods for creating messages of each type and for filling in their contents. For exam you can create and send TextMessage to a queue with the command:

      TextMessage message = queueSession.createTextMessage();
      Message.setText(msg.text);
      queueSender.send(message);

The JMS API Programming Model

- Messages…
  - Message Bodies…
    - At the consuming end, a message arrives as a generic Message object and must be cast to the appropriate message type.
    - You can use one or more getter methods to extract the message contents
    - The following example uses the ***getText*** method:

```
Message m = queueReceiver.receive();
if (m instanceof TextMessage) {
        TextMessage message = (TextMexxafe)m;
        Systems.out.println("Reading message: " + message.getText();
} else {
        // Handle error
}
```

The JMS API Programming Model

- JMS Message Types:

Message Type	Body Contents
TextMessage	A java.lang.String object (eg the contents of an Extensible Markup Language file)
MapMessage	A set of name/value pairs, with names as String objects and values as primitive types in Java programming language
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

The JMS API Programming Model

- Exception Handling
  - The root class for exceptions thrown by JMS API methods is ***JMSException***.
  - Catching ***JMSException*** provides a generic way of handling all exceptions related to the JMS API
  - The ***JMSException*** class includes the following subclasses:
    - IllegalStateException
    - InvalidClientException
    - InvalidDestinationException
    - InvalidSelectorException
    - JMSSecurityException
    - MessageEDFException
    - MessageFormatException
    - MessageNotReadableException
    - MessageNotWritableException
    - ResourceAllocationException
    - TransactionInProgressException
    - TransactionRolledBackException
  - It is a normal programming practice to catch and handle JMSException when it is appropriate in your program

Writing JMS Programs: Tutorial Classes and Labs

- Writing Simple JMS Client Applications
- Writing Simple JMS Provider Applications
- Creating connections and sessions
- Creating message producers
- Creating message consumers
- Sending and receiving messages

Class Task

- Why do you think that middleware keeps appearing in both cloud computing and middleware modules?
- List 3 types of middleware
- Draw and label the architecture of MOM
- State 3 advantages of MOM