CmpE 473
Internet Programming

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Java Messaging

Messaging

- Between loosely-coupled systems
  - A component sends a message to a destination
  - The receiver picks it up whenever it wants
- Asynchronous
  - Messages delivered as recipients become online
- Contrast with RMI
  - No need to know the internal details of others
  - No need to be available at the same time
- JMS: Java API

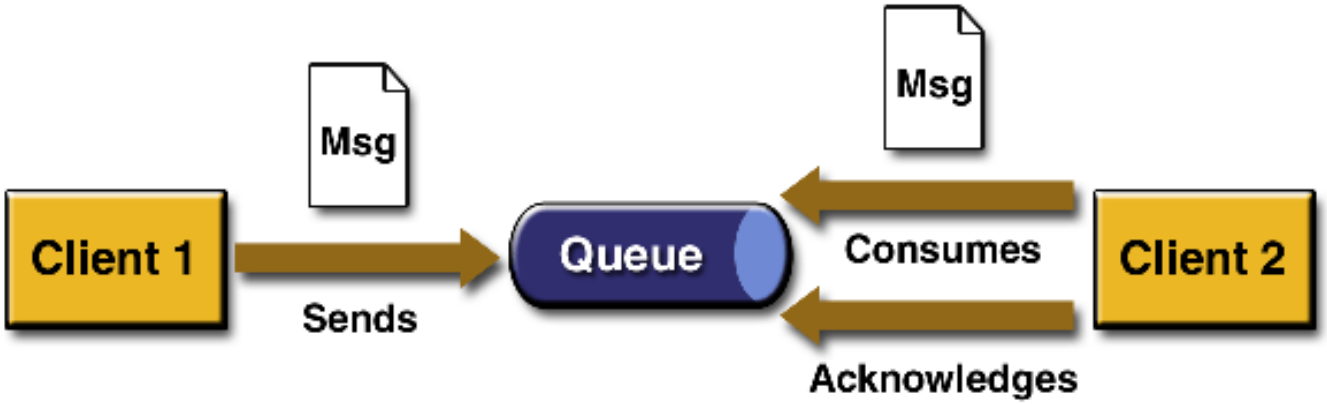
When to use it?

- Components should not depend on others' interfaces
- Application should run even when some components are not available
- Components can work without receiving immediate responses (non-blocking)

JMS API

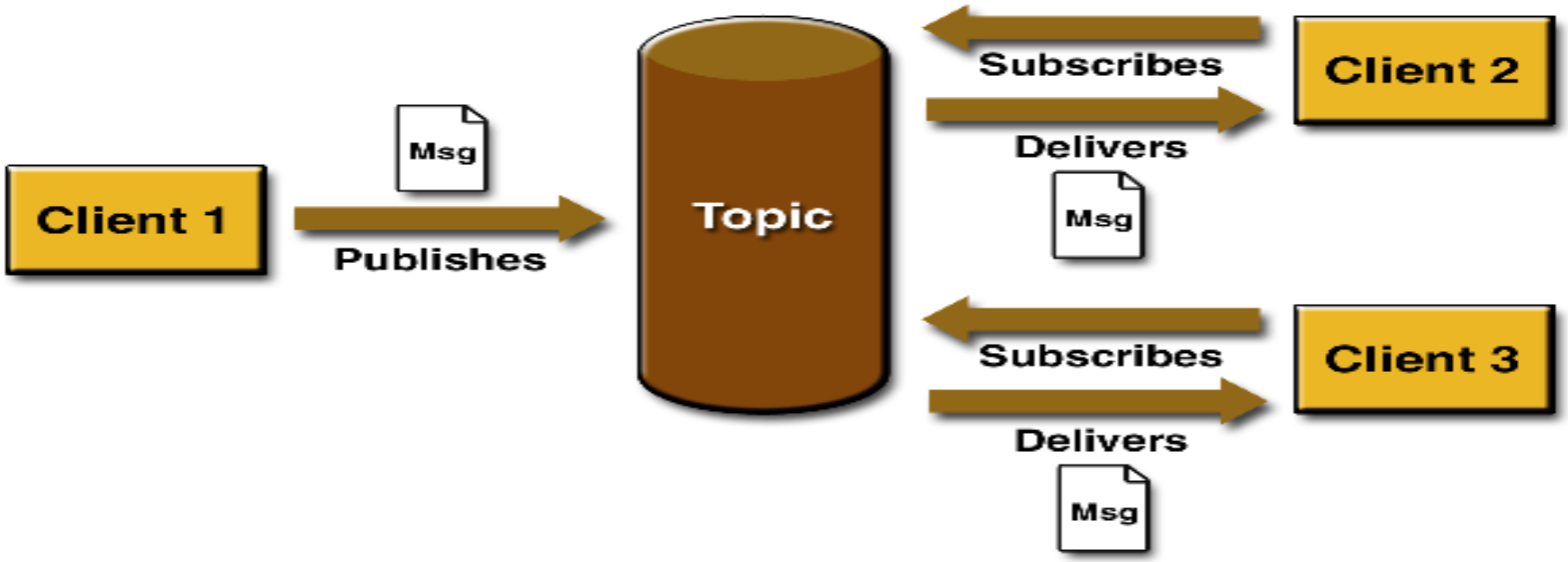
- JMS Provider implements JMS interface
  - Comes in J2EE
- JMS Clients produce and consume messages
- Messages are the communication objects
- Providers implement point-to-point and publish/subscribe

Point-to-Point



- Client1 produces messages for a particular queue
- Client2 consumes the messages and acknowledges receipt
- Queue keeps messages until they are consumed or until they expire

Publish/Subscribe

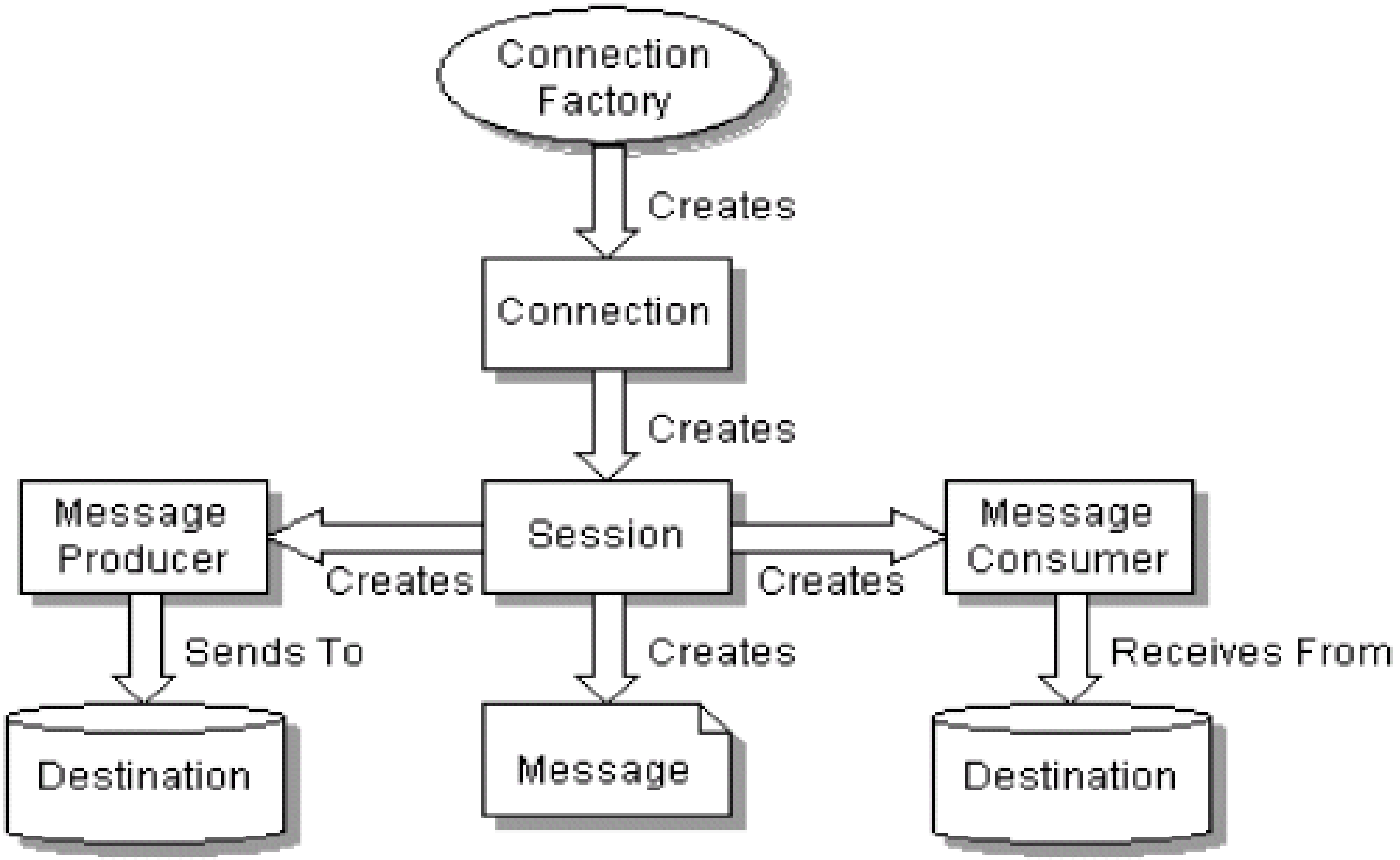


- Multiple producers; multiple consumers
- Topic retains messages until all subscribers are receive them
- Consumers start consuming after they subscribe

Message Consuming

- Conceptually asynchronous:
  - No need to be available at the same time
- Synchronous access:
  - Subscriber or receiver fetches messages by calling `receive`
  - Either blocks or wait for a time period
- Asynchronous access:
  - Consumer registers with a `message listener`
  - When a message arrives at the destination, JMS provider calls client's `onMessage` method
  - Similar to the JAVA delegation event model

JMS API


Connection Factory
Creates
Connection
Creates
Message Producer
Creates
Session
Creates
Message Consumer
Sends To
Creates
Receives From
Destination
Message
Destination

Administered Objects

- Connection Factories
  - Object used by the client to connect to the provider
  - Contains configuration information
  - Instance of the following interfaces
    - `QueueConnectionFactory`
    - `TopicConnectionFactory`
  - Default `QueueConnectionFactory` and `TopicConnectionFactory` objects

Connection Factories

Context ctx = new InitialContext();

- Looks for jndi.properties file
- JNDI implementation and namespaces to use

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

Destination

- Target for produced messages
- Source for consumed messages
- Create a queue or a topic (before usage)
  - j2eeadmin -addJmsDestination *queue_name* queue
  - j2eeadmin -addJmsDestination *topic_name* topic
- To look up a destination
  - Queue myQueue = (Queue) ctx.lookup("MyQueue");

Connection

- Denotes a virtual connection with the provider
- Use connection to create sessions
  - QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();
- Close the connection at the end
  - queueConnection.close();

Session

- Context for producing and consuming messages
- Used to create messages, message producers, and consumers
- Transaction option
  - Two values: Transaction set and acknowledgement
  - QueueSession queueSession = queueConnection.createQueueSession(true, 0);
  - TopicSession topicSession = topicConnection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
- Session.CLIENT_ACKNOWLEDGE:
  - Acknowledges all messages consumed so far

Message Producer

- Used for sending messages to a destination
- Create a message producer
  - QueueSender queueSender = queueSession.createSender(myQueue);
  - TopicPublisher topicPublisher = topicSession.createPublisher(myTopic);
- Send a message
  - queueSender.send(message);

Message Consumer

- Used for receiving messages from a destination
- Create a message consumer
  - QueueReceiver queueReceiver = queueSession.createReceiver(myQueue);
  - TopicSubscriber topicSubscriber = topicSession.createSubscriber(myTopic);
- Start the connection
  - queueConnection.start();
- Receive messages synchronously
  - Message m = queueReceiver.receive();

Message Listener

- Asynchronous message handler object
- Implements `MessageListener` interface
  - `onMessage(Message)` method
  - TopicListener implements MessageListener
  - TopicListener topicListener = new TopicListener();
- Register the message listener with a TopicSubscriber
  - topicSubscriber.setMessageListener(topicListener);
- Start the connection
  - topicConnection.start();
- JMS Provider calls messagelistener's onMessage

Message

- Contains header (required), properties, body
- Message header
  - Send or publish method sets the following
    - JMSDestination, JMSDeliveryMode (Default persistent)
    - JMSExpiration (Default: No expiration), `setTimeToLive`
    - JMSPriority (Range: 0—9; Default:4)
    - JMSMessageID, JMSTimestamp
  - Client sets the following
    - JMSCorrelationID, JMSReplyTo, JMSType
  - JMS Provider sets: JMSRedelivered

Message

- Message body
  - TextMessage: java.lang.String object (for XML)
  - MapMessage: Name (string)/ value (primitive) pairs
  - BytesMessage: A stream of bytes
  - StreamMessage: A stream of primitive values
  - ObjectMessage: A serializable Java object
  - Message: No body

Example

- To create and send a text message

```
TextMessage message = queueSession.createTextMessage();
message.setText(msg_text); // msg_text is a String
queueSender.send(message);
```

- On the receiving side, cast the message as needed

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

Exceptions

- JMS exceptions descend from `JMSException`
  - IllegalStateException
  - InvalidClientIDException
  - InvalidDestinationException
  - InvalidSelectorException
  - JMSSecurityException
  - MessageEOFException
  - MessageFormatException
  - MessageNotReadableException
  - MessageNotWriteableException
  - ResourceAllocationException
  - TransactionInProgressException
  - TransactionRolledBackException