Java Message Service (JMS)
What is JMS?

- A common way for Java programs to
  - create, send, receive
  - and read distributed enterprise messages
- loosely coupled communication
- Asynchronous messaging
- **Reliable** delivery
  - A message is guaranteed to be delivered once and only once.
A JMS Application

- JMS Clients
  - Java programs that send/receive messages
- Messages
- Administered Objects
  - preconfigured JMS objects created by an admin for the use of clients
  - ConnectionFactory, Destination (queue or topic)
- JMS Provider
  - messaging system that implements JMS and administrative functionality
JMS Administration

Administrative Tool → Bind → JNDI Namespace

JMS Client → Logical Connection → JMS Provider

Java Naming & Directory Interface (JNDI)
JMS Messaging Domains

• **Point-to-Point (PTP)**
  - built around the concept of message queues
  - each message has only one consumer

• **Publish-Subscribe systems**
  - uses a “topic” to send and receive messages
  - each message has multiple consumers
Point-to-Point (PTP) Messaging

Client1 → Queue

Msg: sends

Queue: consumes, acknowledges

Msg: consumes

Client2
Publish/Subscribe Messaging

Client1 publishes Topic

Client2 subscribes, delivers

Client3 subscribes, delivers
Message Consumptions

- **Synchronously**
  - A subscriber or a 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's `onMessage()` method.
JMS API Programming Model

- Message Producer creates Connection Factory
- Connection Factory creates Connection
- Connection creates Session
- Session creates Message
- Message Consumer receives from Destination
- Message Producer sends to Destination
- Message Consumer creates Destination
JMS Client Example

- Setting up a connection and creating a session

```java
InitialContext jndiContext = new InitialContext();
//look up for the connection factory
ConnectionFactory cf = jndiContext.lookup(connectionfactoryname);
//create a connection
Connection connection = cf.createConnection();
//create a session
Session session =
    connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
//create a destination object
Destination queue = (Queue) jndiContext.lookup("/jms/myQueue");
//for PointToPoint
Destination topic = (Topic)jndiContext.lookup("/jms/myTopic");
//for publish-subscribe
```
Producer Sample

• Setup connection and create a session..

• Creating producer
  
  ```java
  MessageProducer producer = 
  session.createProducer(dest1);
  ```

• Send a message
  
  ```java
  Message m = session.createTextMessage();
  m.setText("just another message");
  producer.send(m);
  ```

• Closing the connection
  
  ```java
  connection.close();
  ```
Consumer Sample
(Synchronous)

• Setup connection and create a session..

• Creating consumer

```java
MessageConsumer consumer = 
    session.createConsumer(dest1);
```

• Start receiving messages

```java
connection.start();
Message m = consumer.receive();
```
Consumer Sample (Asynchronous)

• Setup the connection, create a session...

• Create consumer

• Registering the listener
  
  ```java
  MessageListener listener=new MyListener();
  consumer.setMessageListener(listener);
  ```

• **MyListener** should have **onMessage**()
  
  ```java
  public void onMessage(Message msg){
    //read the massage and do computation
  }
  ```
public void onMessage(Message message) {
    TextMessage msg = null;
    try {
        if (message instanceof TextMessage) {
            msg = (TextMessage) message;
            System.out.println("Reading message:" + msg.getText());
        } else {
            System.out.println("Message of wrong type");
        }
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
    }
}
JMS Messages

• Message Header
  • used for **identifying** and **routining** messages
  • contains vendor-specified values, but could also contain application-specific data
  • typically **name/value** pairs
• Message Properties (optional)
• Message Body (optional)
  • contains the data
  • five different message body types in the JMS specification
# JMS Message Types

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Contains</th>
<th>Some Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextMessage</td>
<td>String</td>
<td>getText, setText</td>
</tr>
<tr>
<td>MapMessage</td>
<td>set of name/value pairs</td>
<td>setString, setDouble, setLong, getDouble, getString</td>
</tr>
<tr>
<td>BytesMessage</td>
<td>stream of uninterpreted bytes</td>
<td>writeBytes, readBytes</td>
</tr>
<tr>
<td>StreamMessage</td>
<td>stream of primitive values</td>
<td>writeString, writeDouble, writeLong, readString</td>
</tr>
<tr>
<td>ObjectMessage</td>
<td>serialize object</td>
<td>setObject, getObject</td>
</tr>
</tbody>
</table>
More JMS Features

• Durable subscription
  • by default a subscriber gets only messages published on a topic while a subscriber is alive
  • durable subscription retains messages until a they are received by a subscriber or expire
More JMS Features

- **Durable subscription**
  - by default a subscriber gets only messages published on a topic while a subscriber is alive
  - durable subscription retains messages until a they are received by a subscriber or expire

- **Request/Reply**
  - by creating temporary queues and topics
    - `Session.createTemporaryQueue()`
  - `producer=session.createProducer(msg.getJMSReplyTo());`
  - `reply= session.createTextMessage(“reply”);`
  - `reply.setJMSCorrelationID(msg.getJMSMessageID);`
  - `producer.send(reply);`
More JMS Features

- Transacted sessions
  - `session=connection.createSession(true,0)`
  - combination of queue and topic operation in one transaction is allowed
  - `void onMessage(Message m) {
      try {
        Message m2=processOrder(m);
        publisher.publish(m2); session.commit();
      } catch(Exception e) {
        session.rollback();
      }
    }`
More JMS Features

• Persistent/nonpersistent delivery
  • producer.setDeliveryMethod(DeliveryMode.NON_PERSISTENT);
  • producer.send(mesg, DeliveryMode.NON_PERSISTENT,3,1000);

• Message selectors
  • SQL-like syntax for accessing header:
    subscriber = session.createSubscriber(topic, "priority > 6 AND type = 'alert' ");
  • Point to point: selector determines single recipient
  • Pub-sub: acts as filter
JMS API in a Java EE Applications

- Java EE components can use the JMS API to send messages that can be consumed asynchronously by a specialized Enterprise Java Bean
  - message-driven bean
Enterprise Java Beans (EJB)

• EJB is a server-side component
  • encapsulates the business logic of an application

• EJB simplifies the development of large, distributed applications
  o EJB Container provides system-level services
    • e.g. transaction management, authorization
  o Control logic
Message-Driven Bean

- acts as a listener for the JMS,
  - processing messages asynchronously

- specialized adaptation of the JMS API used in the context of Java EE applications
JMS with EJB Example
public class MessageBean implements MessageDrivenBean, MessageListener{

    public void ejbCreate(){}
    public void ejbRemove(){}
    public void setMessageDrivenContext(MessageDrivenContext mdc){}

    public void onMessage(Message m){
        //do computation on the incoming message
        try {
            if (m instanceof TextMessage) {
                System.out.println("Mbean:message" + m.getText());
            }
        } catch (JMSException exp){ ...}
    }
}

MDB Example
JMS and JNDI

• JMS utilizes Java Naming & Directory Interface (JNDI).

• Advantages:
  
  o It hides provider-specific details from JMS clients.
  
  o It abstracts JMS administrative information into Java objects that are easily organized and administrated from a common management console.
SOAP and JMS

- Use JMS as a transportation layer for SOAP
- Example: Sun™ ONE Message Queue
  - enables to send JMS messages with SOAP payload
  - transportation of SOAP messages reliably and publishing SOAP messages to JMS subscribers
SOAP and JMS
(using Sun™ ONE MQ)

Send a SOAP message
• Create a JMS session
• Create a SOAP message
• Transfer the SOAP message into JMS message
  
  Message msg = MessageTransformer.SOAPMessageIntoJMSMessage(SOAPMessage, Session);

• Send the JMS message
SOAP and JMS (using Sun™ ONE MQ)

Receive a SOAP message

- Create a JMS session
- Receive the JMS message
- Transfer the JMS message into SOAP message

```java
SOAPMessage msg = MessageTransformer.SOAPMessageFromJMSMessage(Message, MessageFactory);
```
SOAP and JMS

Deferring SOAP Processing

MessageTransformer

reliable
Publishing a SOAP message
JMS Providers

- SunONE Message Queue (SUN)
- MQ JMS (IBM)
- WebLogic JMS (BEA)
- JMSCourier (Codemesh)
  - merging C++ applications into a JMS environment