Lecture 13

Message-oriented middleware
MOM

Remote execution issues

- *Synchronous communication*: the caller code must block and wait (suspend processing) until the called code completes execution and returns control to it; the caller code can now continue processing.
- *Tightly coupling of the client and the server*: the server interface explicitly indicates what can be invoked for execution.
- *Reliability*: any failure outside of the application – code, network, hardware, service, other, etc – can affect the reliable transport of data between the client and the server.
- *Scalability*: the whole system slows down to the maximum speed of its slowest component (client or server).
- *Availability*: it requires the simultaneous availability of all components; a failure in a component could cause the entire system to fail.

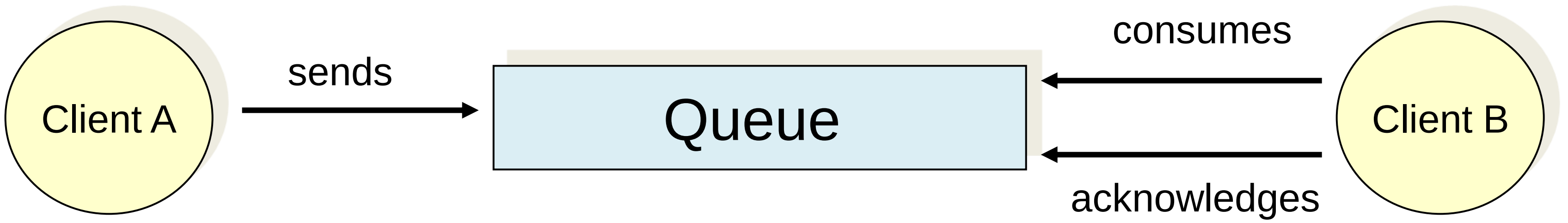
A. The case for MOM

- Distributed components of an application can asynchronously exchange messages in a *P2P fashion.* The MOM system does not require simultaneous or "same-time" availability of all components.
- The sender and the receiver need to know only the message format and what destination to use – *loosely coupled*.
- *Message loss* through network or system failure is prevented by using a store and forward mechanism for message persistence. The specific level-of reliability is typically configurable, but MOM messaging systems are able to guarantee that a message will be delivered, and that it will be delivered to each intended recipient exactly once.
- MOM also decouples the performance characteristics of the components from each other. Subsystems can be *scaled independently*, with little or no disruption to other components.

The concepts

- Each component connects to a *messaging agent* that provides means for creating, sending, receiving and reading messages.
- The MOM provider can implement one of the following models:
  – *point-to-point* communication;
  – *publish-subscribe* communication
- When using these models, a consuming client has two methods of receiving messages from the MOM provider:
  – *Pull* - the consumer can poll the provider to check for any messages, effectively pulling them from the provider.
  – *Push* - a consumer can request the provider to send on relevant messages as soon as the provider receives them.
- It is different from email – communication involving people; it is similar to the postal service.

Point-to-point

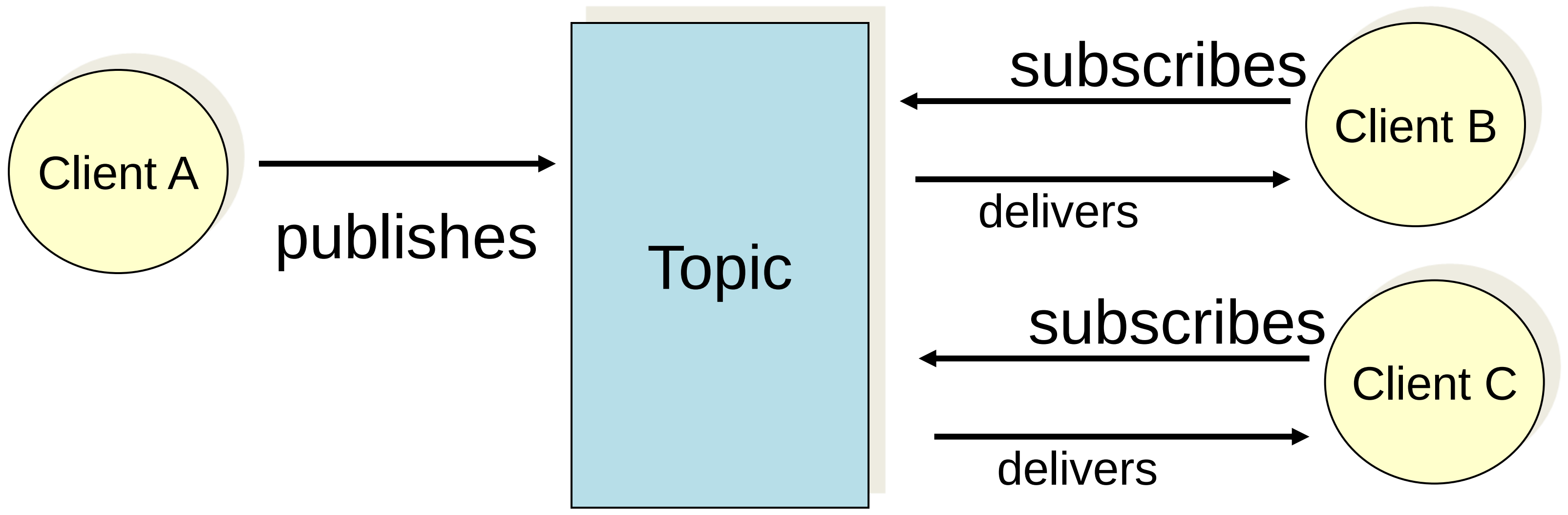


Each message is addressed to a specific queue.
Queues retain all messages until they are consumed or expire.

Features:
- each message has only one consumer;
- there is no timing dependencies among peers;
- the receiver acks the successful processing of a message.

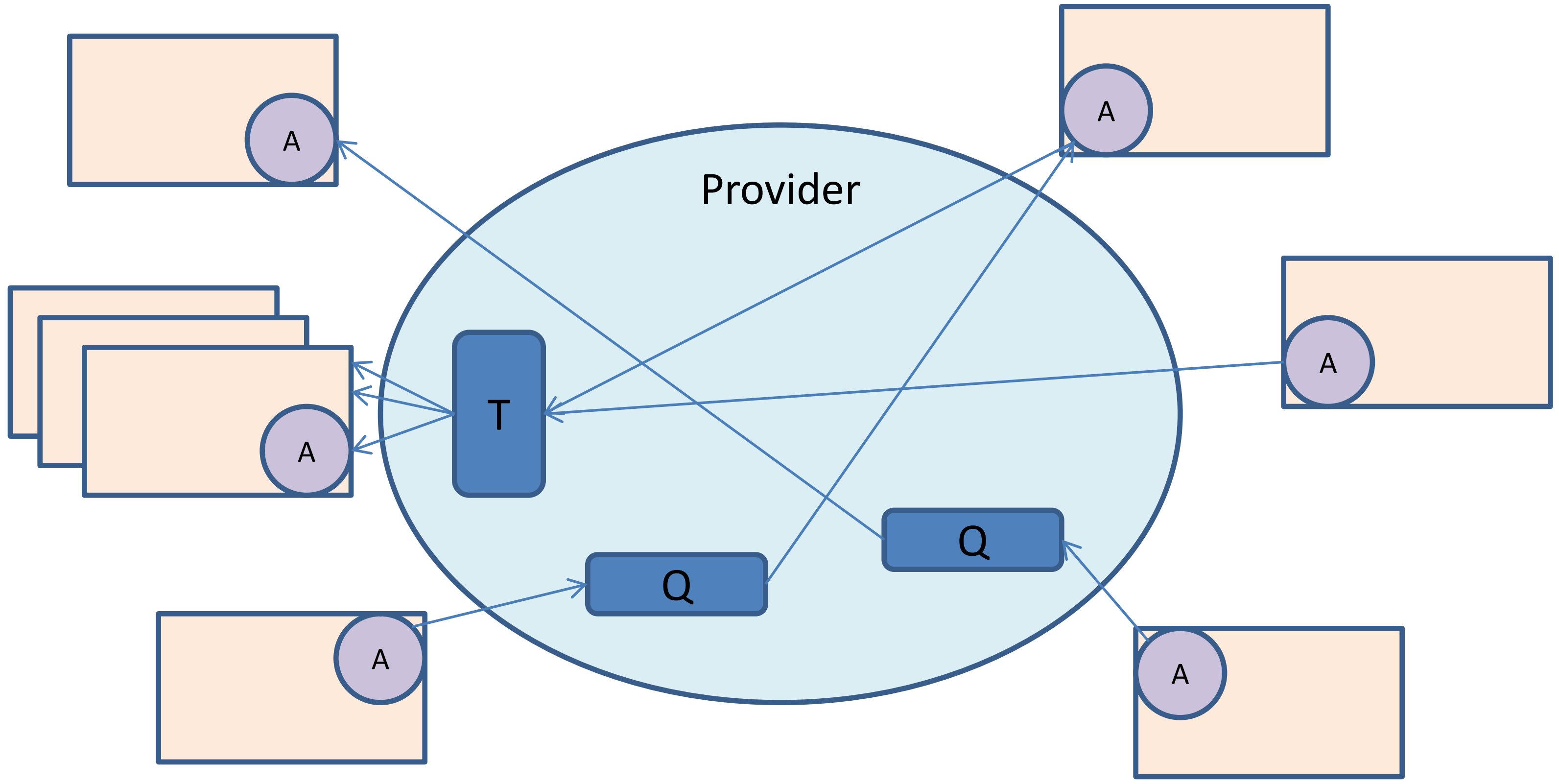
Queue attributes: queue’s name, queue’s size, the save threshold of the queue, message-sorting algorithm,...

Publish/subscribe



Messages are addressed to a topic.
Topics retain messages as long as it takes to distribute them to current subscribers.
There is a timing dependency: a subscriber will consume messages after it created a subscription, and it must continue to be active in order for it to consume messages.

MOM model



A – agent
T – Topic
Q - queue

Comparing the two models

- The two models are very similar.
- However, within the P/S model, every consumer to a topic will receive a message published to it, whereas in point-to-point model only one consumer will receive it.
- P/S is normally used in a broadcast scenario. The publisher has no control over the number of clients who receive the message, nor have they a guarantee any will receive it. Even in a one-to-one messaging scenario, topics can be useful to categorize different types of messages. The P/S model is the more powerful messaging model for flexibility; the disadvantage is its complexity.
- A common application of the point-to-point model is load balancing. With multiple consumers receiving from a queue, the workload for processing the messages is distributed among them.

MOM services I

- *Message filtering*
  - *on the attributes/values of the message;*
  - *on the message payload.*

- *Transactions*: when transactional messaging is used, the sending or receiving application has the opportunity to commit the transaction (all the operations have succeeded), or to abort the transaction (one of the operations failed) so all changes are rolled back. Messages delivered to the provider in a transaction are not forwarded on to the receiving client until the sending client commits the transaction. Transactions may contain multiple messages.

MOM services II

- *Load balancing*: by using the poll model, the load is balanced by placing incoming messages into a point-to-point queue, and the consuming servers can then pull messages from this queue at their own pace. This allows for true load balancing, as a server will only pull a message from the queue once they are capable of processing it.

B. Oracle (SUN) JMS

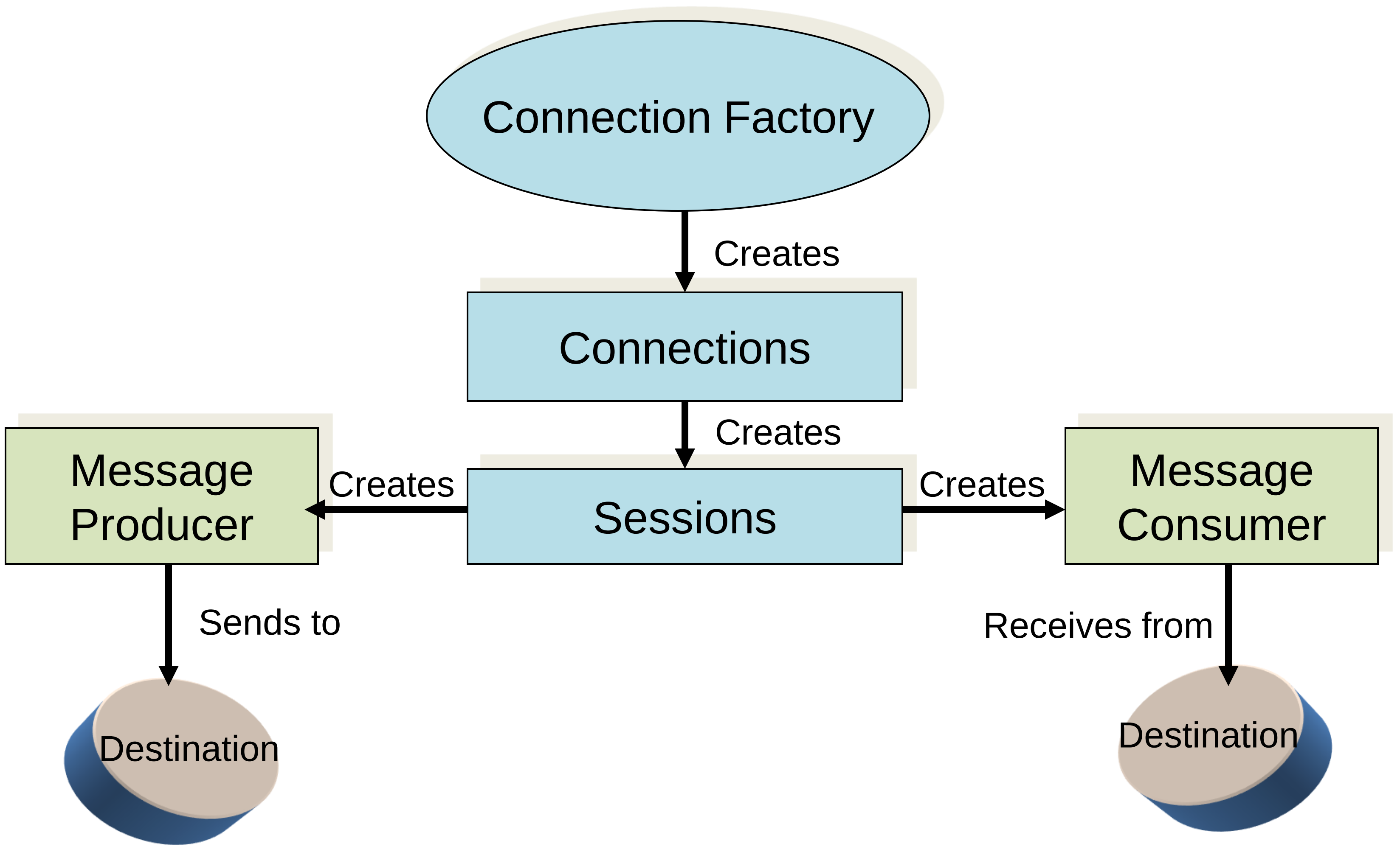
- Java Message Service is a specification for a general Java API that allows distributed applications to use messages. It is a tool for message-oriented middleware (MOM).
- It also allows Java applications to access other messaging systems, such as IBM MQSeries.

- JMS specifications: messages can be consumed in either of two ways,
  - synchronously: the receiver explicitly fetches the messages from the destination by calling the receive method; the method can block until a message arrives, or can time out.
  - asynchronously: the receiver registers a message listener. Whenever a message arrives, the JMS provider delivers the message by calling the listener's onMessage method.

Basic JMS Concepts

- A JMS provider is a messaging system that implements the JMS interfaces and provides admin and control features.
- JMS clients produce and consume messages.
- Messages are the objects that communicate information among peers.
- Administered objects are pre-configured JMS objects created by an administrator for the use of clients. Two types: destination and connection factories.
- Native clients are entities that use a different messaging API (not JMS).

The JMS Programming Model



The JMS Programming Model

- A connection factory: the object a client uses to create a connection with a provider.

- It encapsulates a set of connection config parameters that has been defined by an administrator.

- A pair of connection factories come preconfigured with J2EE SDK and are available as soon as the service is started.

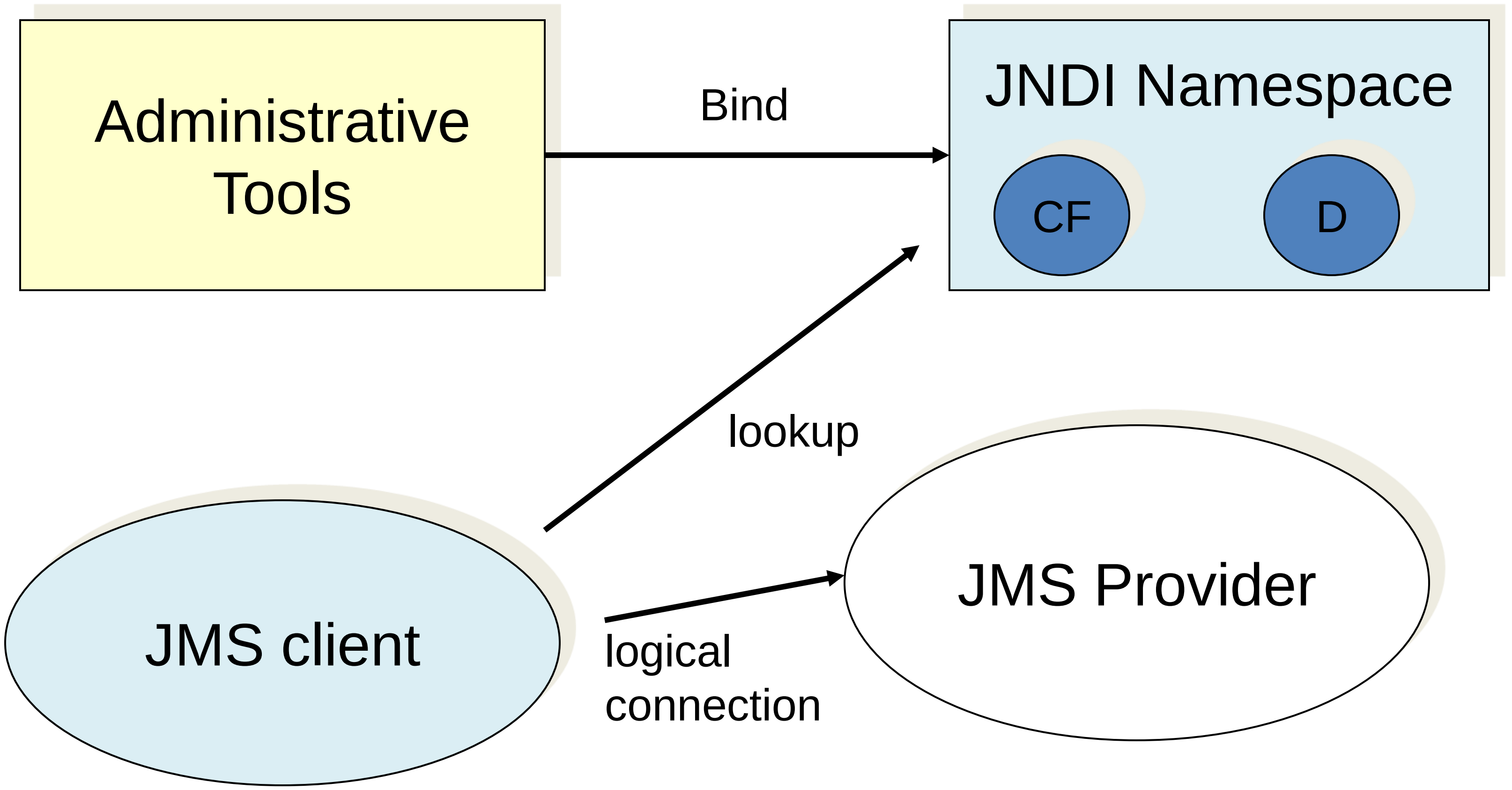
- Each connection factory is an instance of either the QueueConnectionFactory, or the TopicConnectionFactory.

- The JMS client performs at the beginning a lookup of a connection factory.

- By calling the InitialContext method with no parameters, the current classpath is searched for a vendor-specific file named jndi.properties. This file indicates which JNDI API implementation and namespace to use.

```
Context ctx = new InitialContext();
QueueConnectionFactory queueConnectionFactory =
(QueueConnectionFactory)
    ctx.lookup(“QueueConnectionFactory”)
………………………
```

JMS Architecture



Destinations

- A destination is an object a client uses to specify the target of the messages it produces and the source of the messages it consumes.
- These can be created by simple commands:

j2eeadmin –addJmsDestination *queue_name* queue

j2eeadmin –addJmsDestination *topic_name* topic

- A destination is looked up:

Topic myTopic = (Topic) ctx.lookup("MyTopic");

…………

similar for queues

Connections

- A connection encapsulates a virtual connection with a JMS provider. It could represent an open TCP/IP socket between a client and a provider service daemon.
- The connection is used to create one or more sessions.
- Once a factory exists, the connection can be created :

```
QueueConnection queueConnection =
    queue.ConnectionFactory.createQueueConnection();
```

…………..

similar for topic

- When the application completes, the connection must be closed.

```
queueConnection.close();
```

……………….

Sessions

- A session is a single-threaded context for producing and consuming messages. Sessions are used to create message producers and consumers and messages.

- A session provides a transactional context with which to group a set of sends and receives into an atomic unit of work.

- A session implements either a QueueSession or a TopicSession interface.

Message Producers

- A message producer is an object created by a session and used for sending messages to a destination.

```
QueueSender queue Sender =
    queueSession.createSender(myqueue);
TopicPublisher topicPublisher =
    topicSession.createPublisher(myTopic);
```

If the argument is null, the producer is unidentified. The destination specification is postponed until the messages is sent or published.

- Once created, a producer can send messages:

```
queueSender.send(message);
topicPublisher.publish(message);
```

Message Consumers

- A message consumer is an object created by a session and used for receiving messages sent to a destination.
- It allows a client to register its interest in a destination with a JMS provider. The delivery of messages is managed by the JMS provider.

```
QueueReceiver queueReceiver =
    queueSession.createReceiver(myQueue);
queueConnection.start();
Message m = queueReceiver.receive();
```

Message Listener

- A message listener is an object that acts like an asynchronously event handler for messages. It implements the MessageListener interface, which contains one method, onMessage. This method includes the actions taken when a message arrives.

- The message listener is registered with a specific Queuereceiver or TopicSubscriber by using setMessageListener method.

- A message listener is not specific to a particular destination type; it can get messages from either a queue or a topic, depending where it is set.

Message Format

- Header: set of fields (values) used to identify and route messages – JMSMessageID, JMSDestination, JMSPriority etc.
- Properties (optional): for example, to provide compatibility with other messaging systems.
- Body (optional): five message body formats (message types), such as TextMessage, MapMessage, BytesMessage, StreamMessage, ObjectMessage, Message.

- Example:

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

JMS advantages

- The components do not depend on information about other components' interfaces, so that components can be easily replaced.
- The application runs whether or not all components are up and running simultaneously.
- A component can send information to another and then continue to operate without receiving an immediate response.