Taller de Sistemas de Información 2

Web Services SOAP

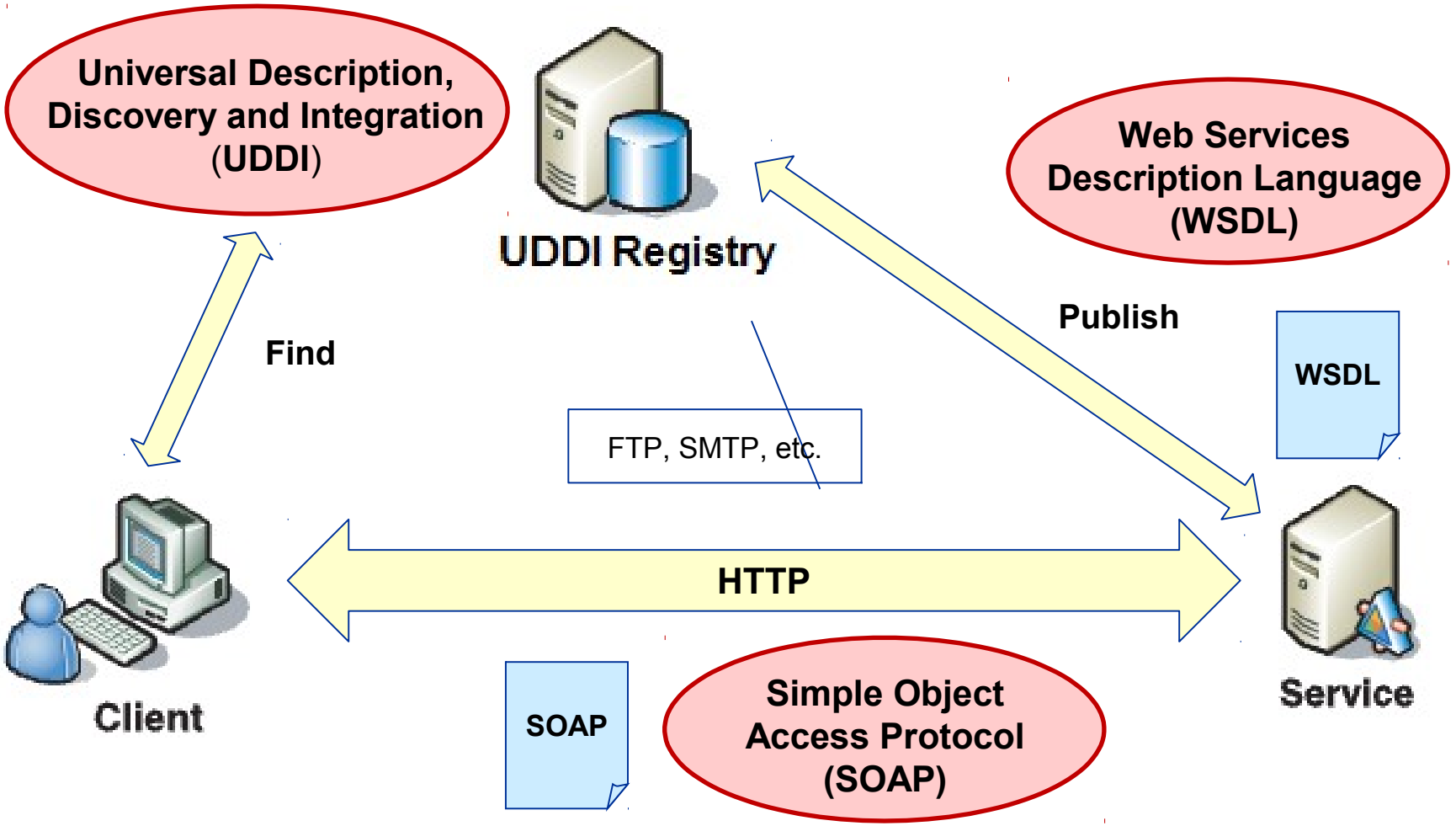
2 de Setiembre de 2014

Instituto de Computación
Facultad de Ingeniería
Universidad de la República de Uruguay

Que es un web service?

- Es un servicio o funcionalidad
- Se encuentra disponible a través de la web
- Utiliza una forma estandarizada de mensajería (XML)
- No se encuentra atado a ningún sistema operativo ni ningún lenguaje de programación
- Se puede auto describir (XML)
- Puede ser descubierto a través de un mecanismo de búsqueda

Web Services SOAP



Web Services SOAP

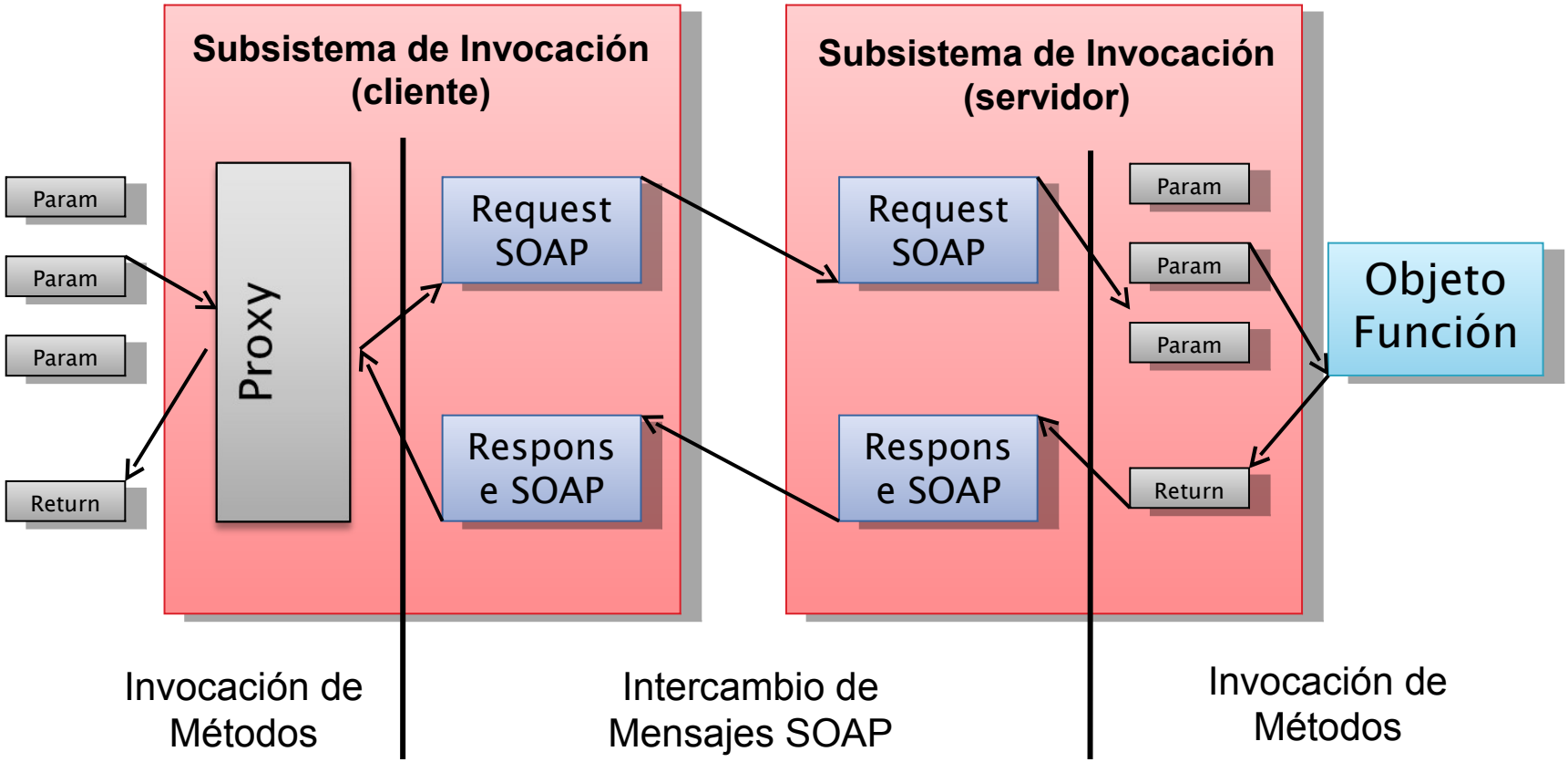
- Este tipo de servicios, implica el intercambio de mensajes XML, codificados según el protocolo SOAP

```
<SOAPenv:Envelope
        xmlns:SOAPenv="http://schemas.xmlSOAP.org/SOAP/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAPenv:Body>
  <req:getNumberOfArticles xmlns:req="http://daily-moon.com/CMS/">
     <req:category>classifieds</req:category>
  </req:getNumberOfArticles>
 </SOAPenv:Body>
</SOAPenv:Envelope>
```

Mensaje SOAP

```
<?xml version="1.0"?>
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-
  encoding">
      <soap:Header>
            ...
      </soap:Header>
      <soap:Body>
            ...
            <soap:Fault> ... </soap:Fault>
      </soap:Body>
</soap:Envelope>
```

Subsistema de invocación



Definiendo un web service

Laboratorio de Integracion de Sistemas

```
@WebService
public class CardValidator {
        public boolean validate(CreditCard creditCard) {
                String lastDigit = creditCard.getNumber().substring(
                        creditCard.getNumber().length() - 1,
                        creditCard.getNumber().length());
                if (Integer.parseInt(lastDigit) % 2 != 0) {
                        return true;
                } else {
                        return false;
                }
        }
}
```

Definiendo un web service

```
@XmlRootElement
public class CreditCard {
        private String number;
        private String expiryDate;

        private Integer controlNumber;
        private String type;

        // ...
}
```

Invocando un web service

```
public class Main {

        public static void main(String[] args) {
                CreditCard creditCard = new CreditCard();
                creditCard.setNumber("12341234");
                creditCard.setExpiryDate("10/10");
                creditCard.setType("VISA");
                creditCard.setControlNumber(1234);
                CardValidator cardValidator =
                        new CardValidatorService().getCardValidatorPort();
                cardValidator.validate(creditCard);
        }
}
```

La clase CreditCard

```
@XmlRootElement
public class CreditCard {
        private String number;
        private String expiryDate;

        private Integer controlNumber;
        private String type;

        // ...
}
```

El documento XML

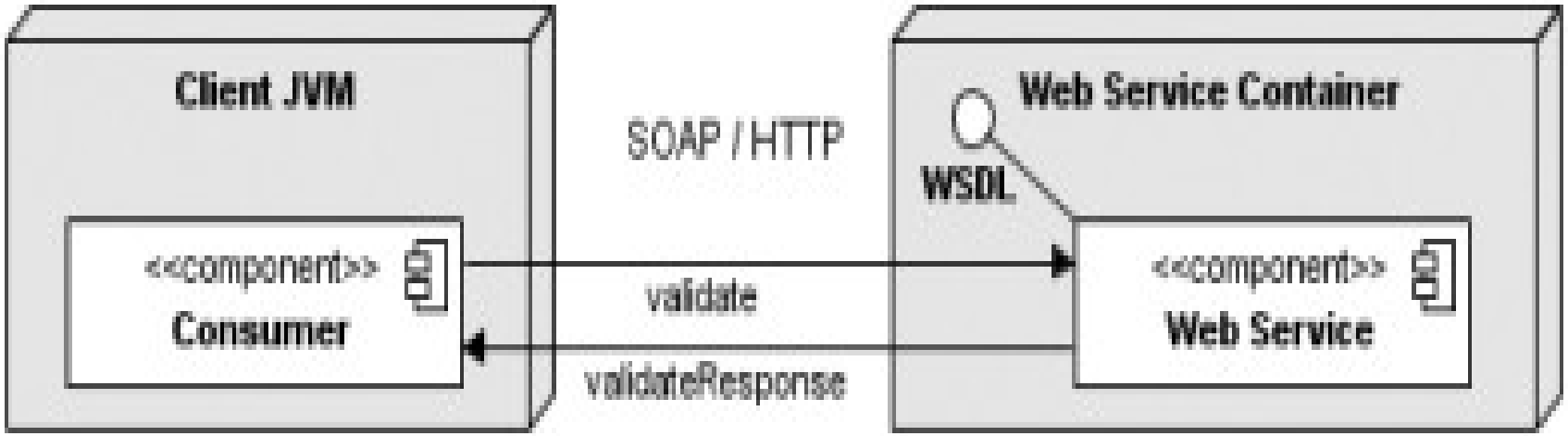
```
<?xml version="1.0" encoding="UTF-8"
      standalone="yes"?>

<creditCard>
        <controlNumber>6398</controlNumber>
        <expiryDate>12/09</expiryDate>
        <number>1234</number>
        <type>Visa</type>
</creditCard>
```

El schema XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="creditCard" type="creditCard"/>
   <xs:complexType name="creditCard">
      <xs:sequence>
         <xs:element name="controlNumber" type="xs:int" minOccurs="0"/>
         <xs:element name="expiryDate" type="xs:string" minOccurs="0"/>
         <xs:element name="number" type="xs:string" minOccurs="0"/>
         <xs:element name="type" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Web Services / Funcionamiento

- Si bien el programador Java no debe preocuparse del manejo de los protocolos de bajo nivel en web services, es conveniente conocer que esta sucediendo por detras...



JAX-WS

- La clase debe estar anotada con @javax.jws.WebService
- Si el web service es implementado con un EJB, entonces este solo puede ser stateless
- La clase debe ser publica, y no puede ser final ni abstract
- La clase debe tener un constructor por defecto
- La clase no debe definir el método finalize
- El servicio debe ser stateless

JAX-WS

- Basta con que un POJO cumpla con los requisitos anteriores, para que pueda ser desplegado en un servlet container
  - Esto se conoce como **servlet endpoint**
- Análogamente, un Stateless EJB puede ser utilizado como clase de implementación del servicio
  - Esto se conoce como **EJB endpoint**

@WebService

- Esta anotación marca una clase o interfaz como un web service
- Si se aplica directamente sobre la clase y no la interfaz, entonces el container generara la interfaz a partir de los métodos públicos

```
@WebService
public class CardValidator { ... }

@WebService
public interface CCValidator { ... }
public class CardValidator implements CCValidator { ...}
```

@WebMethod

```
@WebService
public class CardValidator {

        @WebMethod(operationName = "ValidateCreditCard")
        public boolean validate(CreditCard creditCard) {
                // logica de negocio
        }

        @WebMethod(exclude = true)
        public void validate(String ccNumber) {
                // logica de negocio
        }
}
```

@WebResult

```
@WebService
public class CardValidator {
        @WebMethod
        @WebResult(name = "IsValid")
        public boolean validate(CreditCard creditCard) {
                // ...
        }
}
```

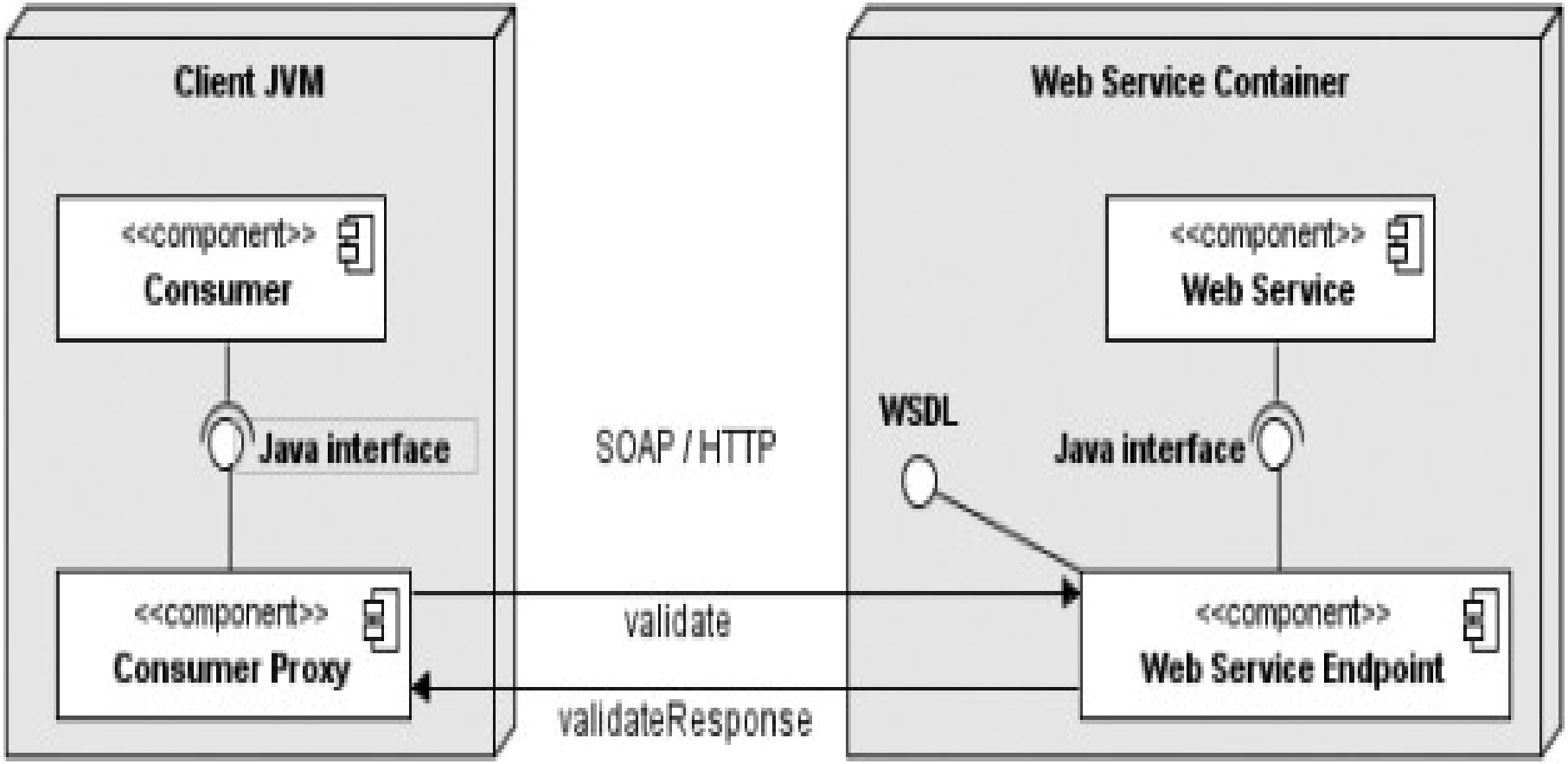
@WebParam

```
@WebService
public class CardValidator {
        @WebMethod
        public boolean validate(@WebParam(name = "Credit-Card")
                                CreditCard creditCard) {
                // logica del servicio
        }
}
```

@OneWay

- Es una anotación que permite indicar que un método, no retorna valores
- Por ejemplo, los métodos que retornan void
- Esto permite que el container realice optimizaciones con los métodos de este tipo, por ejemplo, usando Asynchronous (solo en EJB)

Invocando un web service



Invocando un web service

- Usando herramientas que generen stubs y clases Java a partir del WSDL, podemos invocar un servicio web
- El JDK provee una herramienta, denominada **wsimport** que genera las clases necesarias para poder realizar la invocación a un determinado web service del que conocemos su wsdl
- Las interfaces generadas se denominan SEI (Service Endpoint Interfaces)

Invocando un web service

- Las interfaces y artefactos generados, deben ser colocados en el proyecto que realizara el consumo del servicio web
- Una vez colocados los artefactos (.java) generados, podemos hacer la invocacion de esta forma:

```
CardValidatorService cardValidatorService = new CardValidatorService();
CardValidator cardValidator = cardValidatorService.getCardValidatorPort();
cardValidator.validate(creditCard);
```

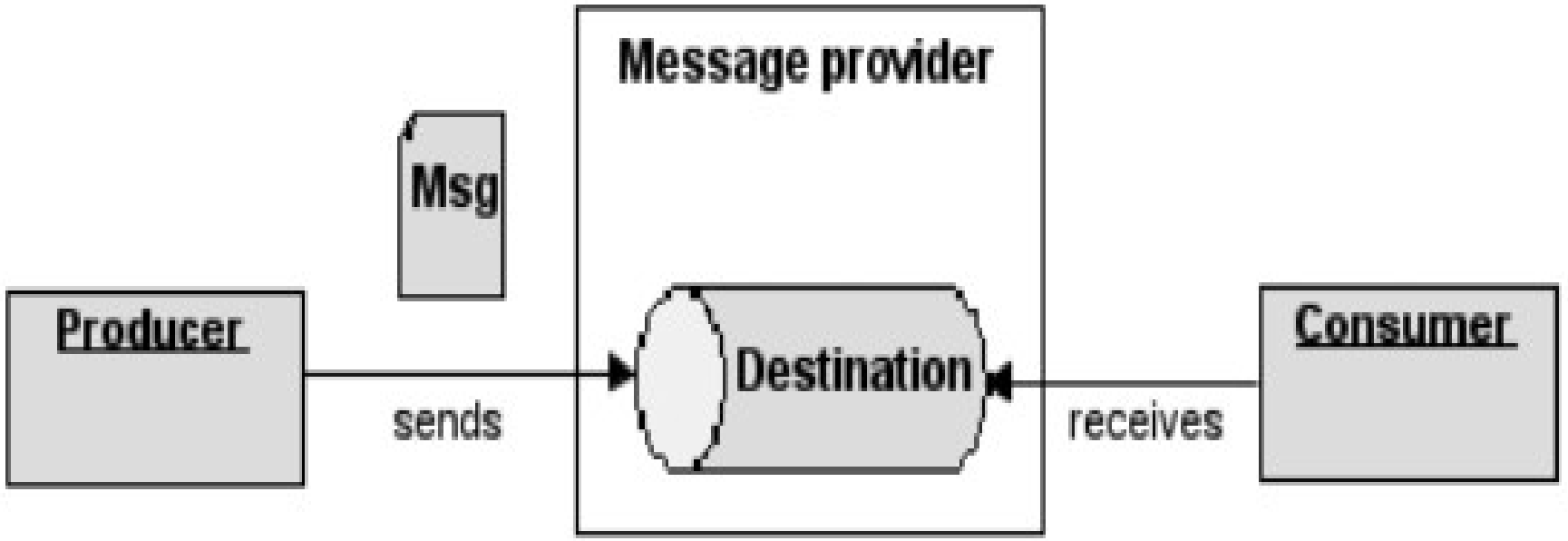
Generando los artefactos

- Usamos la herramienta wsimport
- Por ejemplo, en el practico tenemos este ejemplo:
  - **wsimport -keep -d ../src http://localhost:8080/WebService/CalculadoraServiceImpl/CalculadoraServiceImpl?wsdl**
  - **keep** permite mantener el codigo fuente generado
  - **-d ../src** es la ruta donde se almacena dicho codigo

Messaging

- La comunicación que hemos visto hasta ahora, es sincrónica
  - Un managed bean llama a un ejb
- El invocador y el invocado
  - Deben estar en funcionamiento para que la comunicación se de
  - Deben utilizar la misma tecnología
- Incluso en el caso de la comunicación asíncrona en EJBs

Message Oriented Middleware



Terminologia

- Cuando un mensaje es enviado, el software que efectúa el envío y almacenamiento del mensaje, se denomina **provider** o **broker**
- El emisor del mensaje se denomina **producer**
- El lugar donde el mensaje es enviado, se denomina **destination**
- El receptor del mensaje se denomina **consumer**

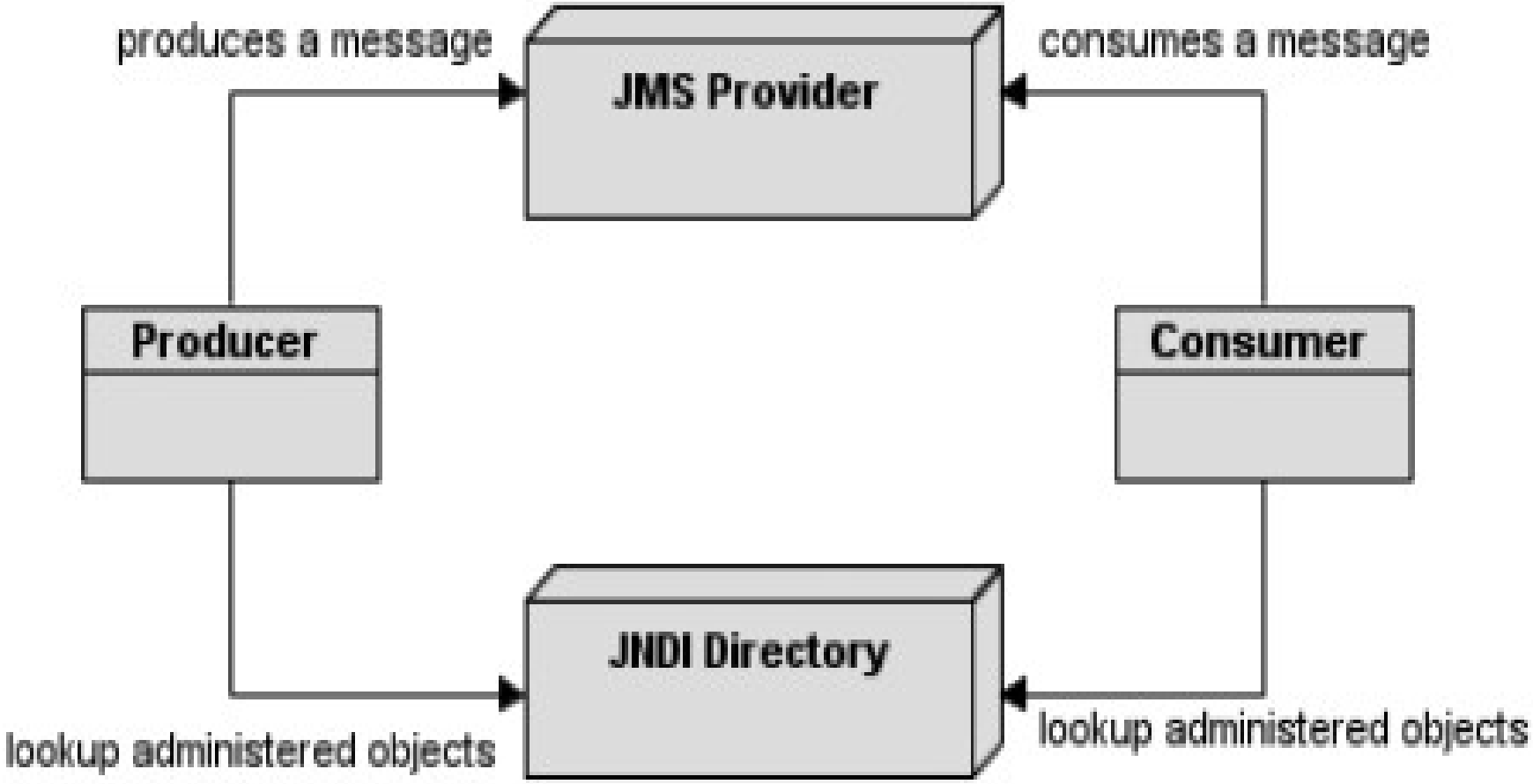
Java EE

- En Java EE, el API que permite manipular los conceptos antes mencionados, se denomina JMS (Java Message Service)
- JMS es un API, no se encarga de transportar el mensaje, sino que brinda acceso a un proveedor que es el que transporta el mensaje
- Es una analogia muy similar a JDBC (driver + manejador de base de datos)

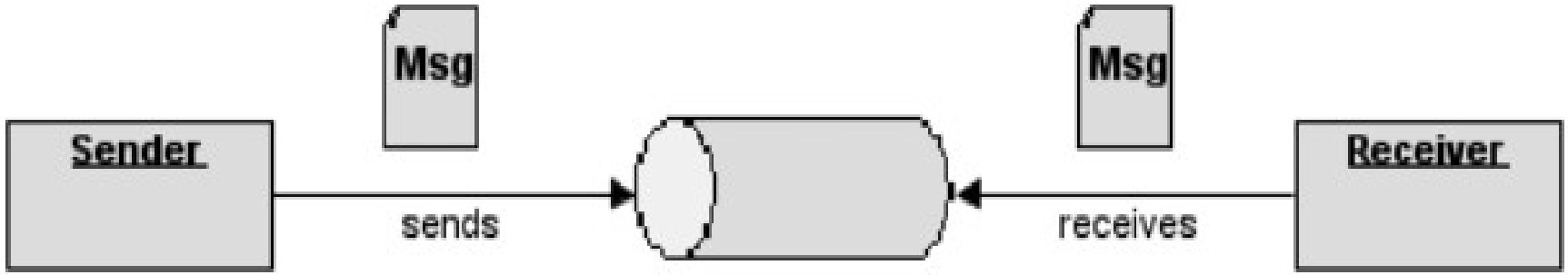
```
public class Sender {
   public static void main(String[] args) {
      // Gets the JNDI context
      Context jndiContext = new InitialContext();

      // Looks up the administered objects
      ConnectionFactory connectionFactory =
            (ConnectionFactory)jndiContext.lookup("jms/javaee6/ConnectionFactory");
      Queue queue = (Queue) jndiContext.lookup("jms/javaee6/Queue");

      // Creates the needed artifacts to connect to the queue
      Connection connection = connectionFactory.createConnection();
      Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      MessageProducer producer = session.createProducer(queue);

      // Sends a text message to the queue
      TextMessage message = session.createTextMessage();
      message.setText("This is a text message");
      producer.send(message);

      connection.close();
   }
}
```

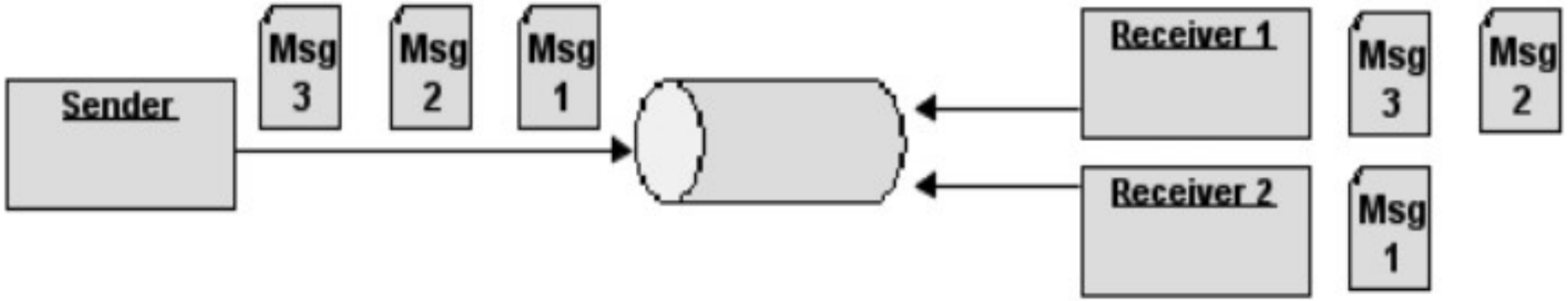
```
public class Receiver {
   public static void main(String[] args) {
      // Gets the JNDI context
      Context jndiContext = new InitialContext();

      // Looks up the administered objects
      ConnectionFactory connectionFactory =
         (ConnectionFactory)jndiContext.lookup("jms/javaee6/ConnectionFactory");
      Queue queue = (Queue) jndiContext.lookup("jms/javaee6/Queue");

      // Creates the needed artifacts to connect to the queue
      Connection connection = connectionFactory.createConnection();
      Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      MessageConsumer consumer = session.createConsumer(queue);
      connection.start();

      // Loops to receive the messages
      while (true) {
         TextMessage message = (TextMessage) consumer.receive();
         System.out.println("Message received: " + message.getText());
      }
   }
}
```

Arquitectura de JMS

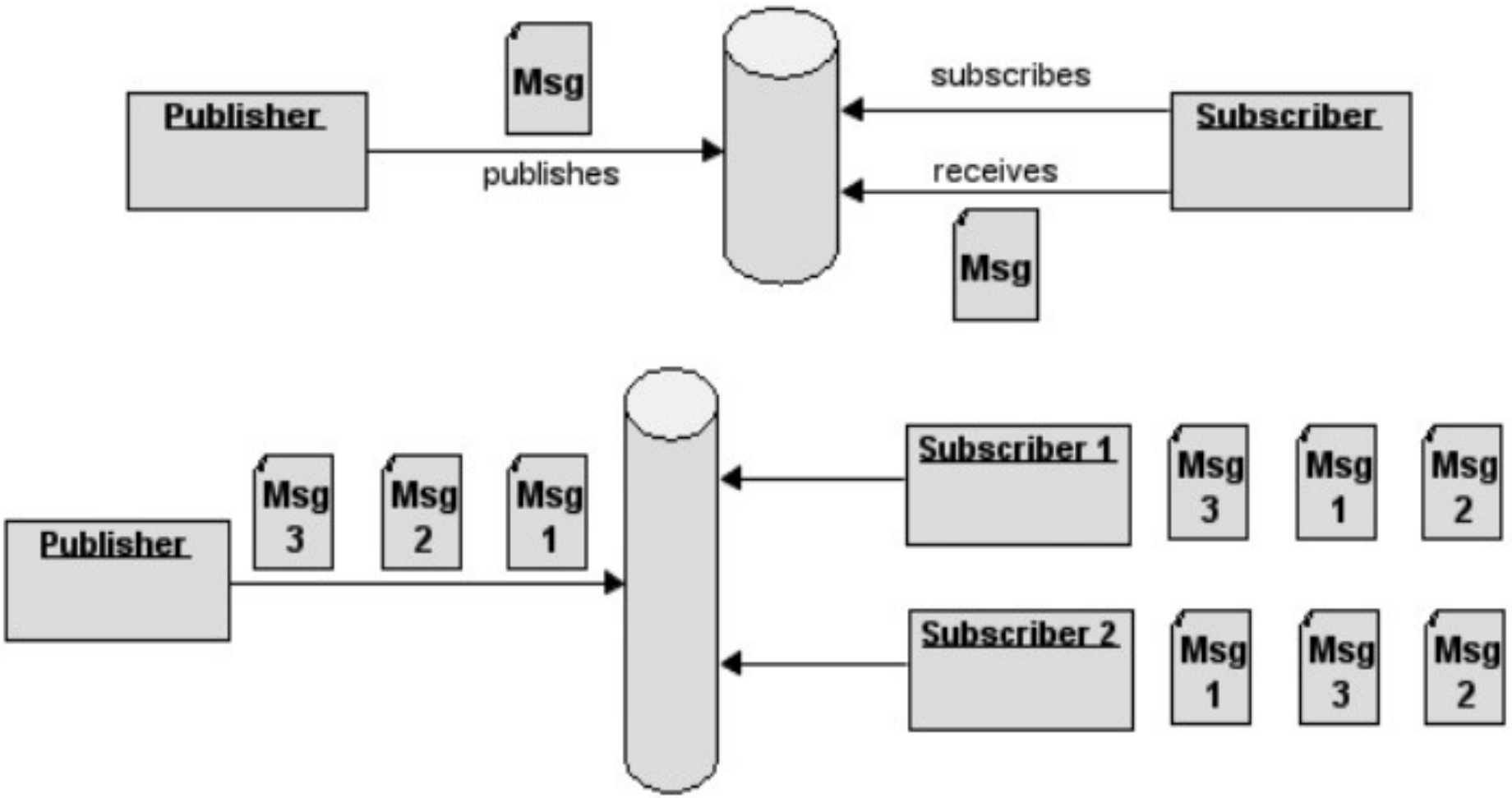


Point to Point

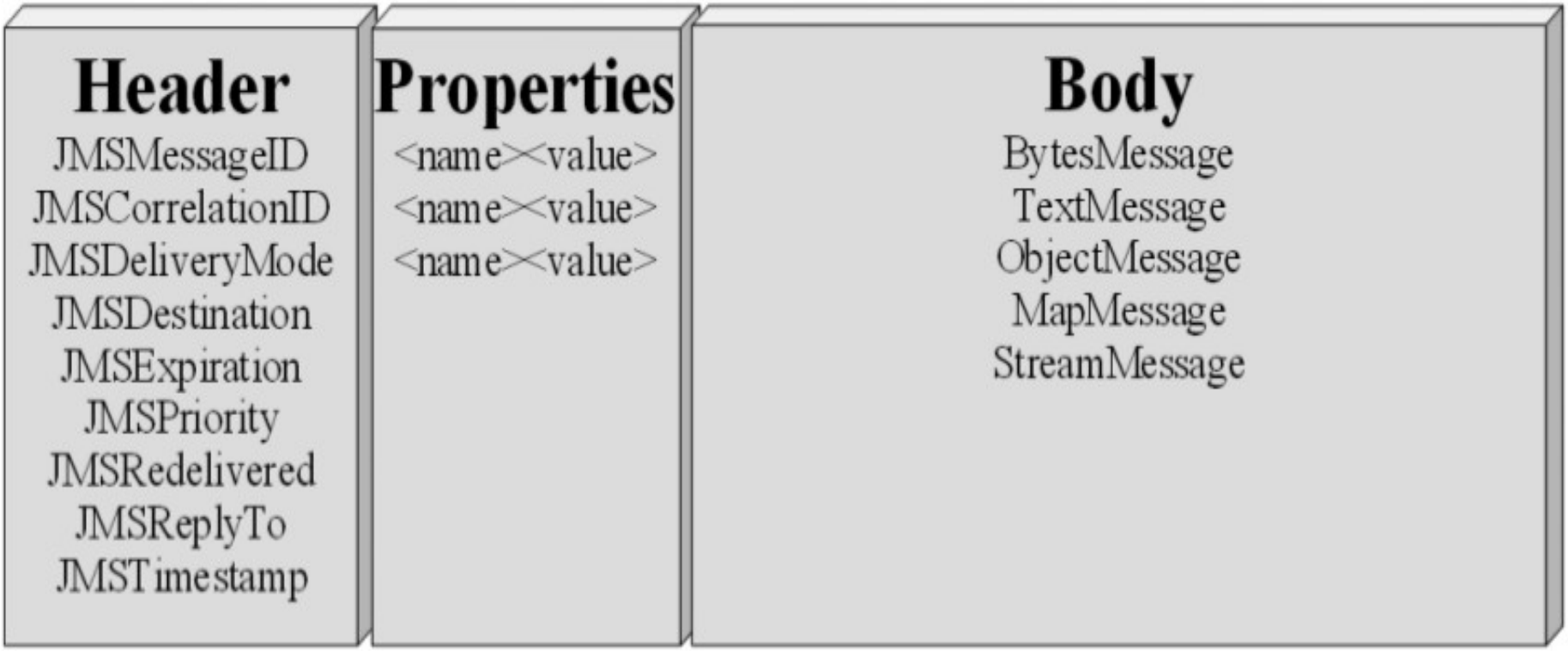




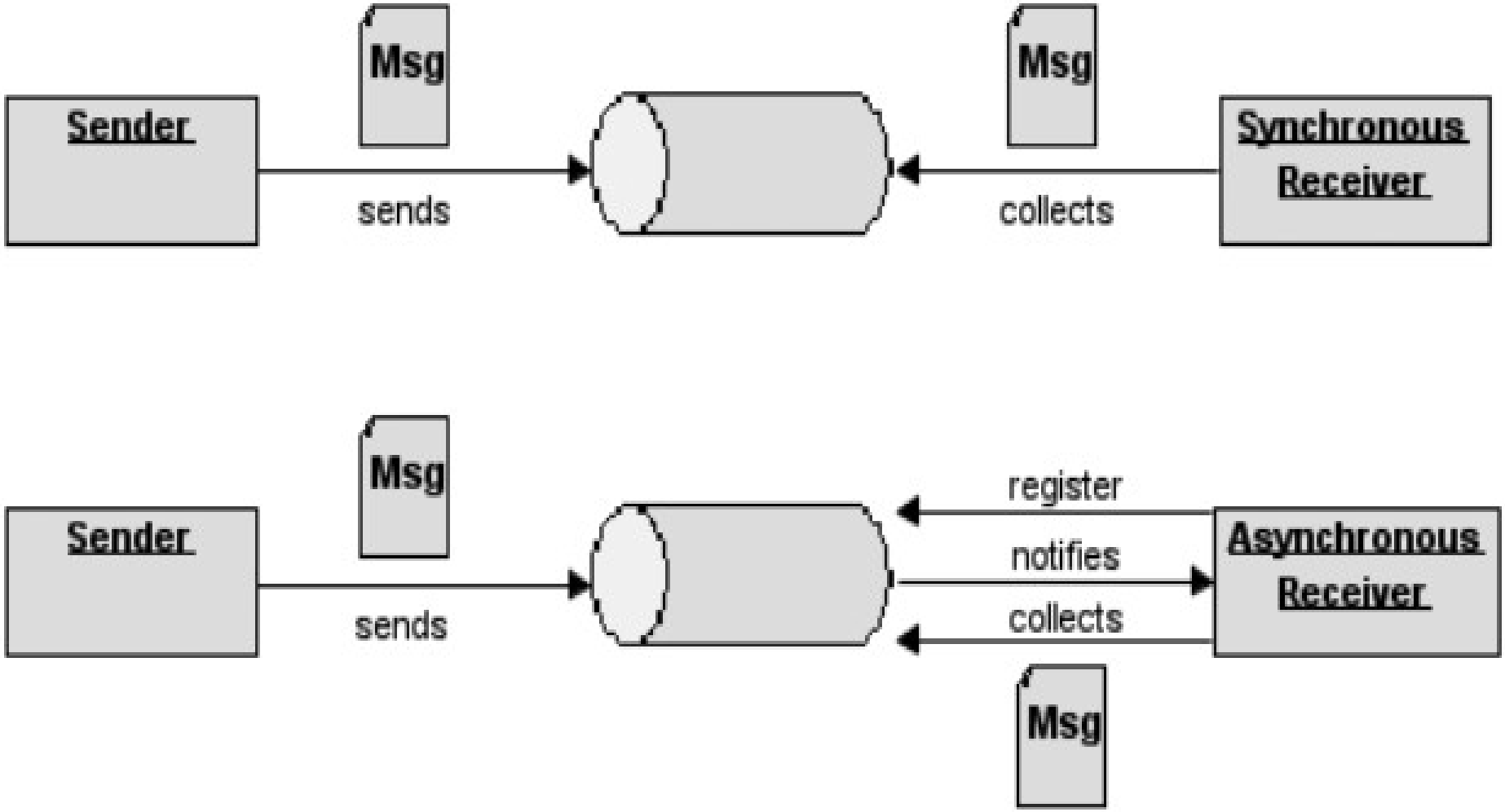
Publish - Subscribe



Mensaje JMS


Header
JMSMessageID
JMSCorrelationID
JMSDeliveryMode
JMSDestination
JMSExpiration
JMSPriority
JMSRedelivered
JMSReplyTo
JMSTimestamp
Properties
<name><value>
<name><value>
<name><value>
Body
BytesMessage
TextMessage
ObjectMessage
MapMessage
StreamMessage

Consumo de mensajes



Consumo asincrono

```
public class Listener implements MessageListener {
   @Resource(mappedName = "jms/javaee6/ConnectionFactory")
   private static ConnectionFactory connectionFactory;
   @Resource(mappedName = "jms/javaee6/Topic")
   private static Topic topic;

   public static void main(String[] args) {
      // Creates the needed artifacts to connect to the queue
      Connection connection = connectionFactory.createConnection();
      Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE)
      MessageConsumer consumer = session.createConsumer(topic);
      consumer.setMessageListener(new Listener());
      connection.start();
   }

   public void onMessage(Message message) {
      System.out.println("Message received: " +
                              ((TextMessage) message).getText());
   }
}
```

Message Driven Beans

- Un MDB es un consumidor asincrono, el cual es invocado por el container ante la llegada de un mensaje
- Son parte de la especificacion EJB, ya que su modelo es muy similar a los Stateless Session Beans
- **Por que usarlos?**
  - **Aprovecha los beneficios del modelo EJB de trabajo**

Message Driven Beans

```
@MessageDriven(mappedName = "jms/javaee6/Topic")
public class BillingMDB implements MessageListener {

    public void onMessage(Message message) {
        TextMessage msg = (TextMessage)message;
        System.out.println("Message received: " + msg.getText());
    }
}
```

Requisitos para un MDB

- La clase debe estar anotada con @javax.ejb.MessageDriven
- La clase debe implementar (directa o indirectamente) javax.jms.MessageListener
- La clase debe ser publica, no abstracta y no final
- La clase debe tener un constructor sin parámetros
- La clase no debe definir el método finalize()

@MessageDriven

Laboratorio de Integracion de Sistemas

```
@MessageDriven(
      mappedName = "jms/javaee6/Topic",
      activationConfig = {
            @ActivationConfigProperty(propertyName = "acknowledgeMode",
                                      propertyValue = "Auto-acknowledge"),
            @ActivationConfigProperty(propertyName = "messageSelector",
                                      propertyValue = "orderAmount < 3000")
      }
)
public class BillingMDB implements MessageListener {
    public void onMessage(Message message) {
        TextMessage msg = (TextMessage)message;
        System.out.println("Message received: " + msg.getText());
    }
}
```

Inyeccion de dependencias

- Como en el caso de otros EJBs, tambien se soportan inyecciones

```
@PersistenceContext
private EntityManager em;
@EJB
private InvoiceBean invoice;
@Resource(name = "jms/javaee6/ConnectionFactory")
private ConnectionFactory connectionFactory;
@Resource
private MessageDrivenContext context;
```

Excepciones

- Cualquier operación en los elementos del API de JMS, en caso de error, propagara una javax.jms.JMSException
- **Importante: JMSException es una checked exception**
- Por este motivo, el rollback de la transacción, si existiera, debe ser realizado manualmente usando el contexto

Excepciones

```
public void onMessage(Message message) {
    TextMessage msg = (TextMessage)message;
    try {
        System.out.println("Message received: " + msg.getText());
        sendPrintingMessage();
    } catch (JMSException e) {
        context.setRollBackOnly();
    }
}
```