Universidade Presbiteriana Mackenzie

JMS (Java Message Service) – Beans de Mensagens

Prof. Leandro Pupo Natale

Faculdade de Computação e Informática

Tópicos da Aula

- Motivação
- *MOM e Messaging*
- JMS – Java Message Service
- Domínios de Mensagens
- Usando a API JMS
- Interfaces
- Integração JMS-EJB
- Message-Driven Beans

Motivação

- Devido a algumas limitações de RMI-IIOP
  - **Performance.** Um cliente típico RMI-IIOP precisa esperar enquanto o servidor está processando. Apenas quando o servidor completa o trabalho e o cliente recebe o resultado, este pode continuar seu processamento
  - **Garantia.** Quando um cliente RMI-IIOP invoca o servidor, este tem que estar rodando. Se o servidor ou a rede cair o cliente não pode efetuar a operação desejada

Motivação

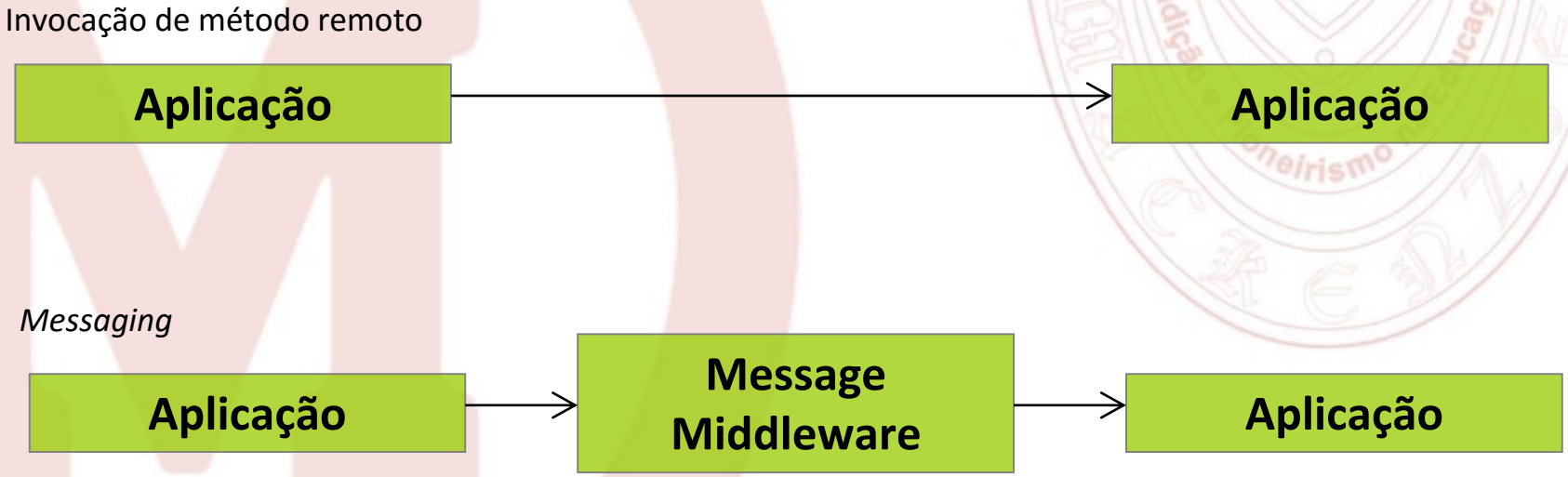
- Devido a algumas limitações de RMI-IIOP
  - Suporte para vários emissores e receptores.
  - RMI-IIOP se limita a comunicação de um único cliente a um único servidor
  - Integração com outros sistemas MOM (Middleware Orientado a Mensagem).

MOM – Middleware Orientado a Mensagens

- Um MOM permite automatizar a integração entre sistemas;
- Permite que um sistema receba ou envie mensagens para outros sistemas de forma **assíncrona**;
- O sistema que envia uma msg não precisa conhecer os sistemas que a receberão e vice-versa;
- Essas características permitem que os sistemas sejam integrados com **baixo acoplamento**.

Messaging

- Mensagens são uma alternativa a invocação de métodos remotos.
- A ideia é inserir uma camada entre o cliente e o servidor



Messaging

- Vantagens
  - Processos não bloqueáveis
  - Garantia de entrega
  - Suporte a múltiplos emissores e receptores

JMS – Java Message Service

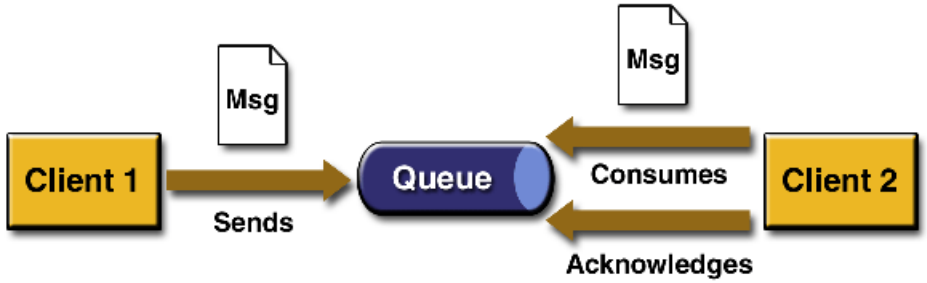
- **Java Message Service – JMS** é uma especificação do JEE que define o funcionamento de um Middleware Orientado a Mensagens - MOM.
- Todo servidor de aplicação que segue a especificação Java EE deve oferecer uma implementação do MOM definido pela JMS.
- Outras especificações do JEE que possuem relacionamento forte com JMS:
  - Enterprise Java Beans (EJB),
  - Java Transaction API (JTA) e
  - Java Transaction Service (JTS)

JMS – Java Message Service

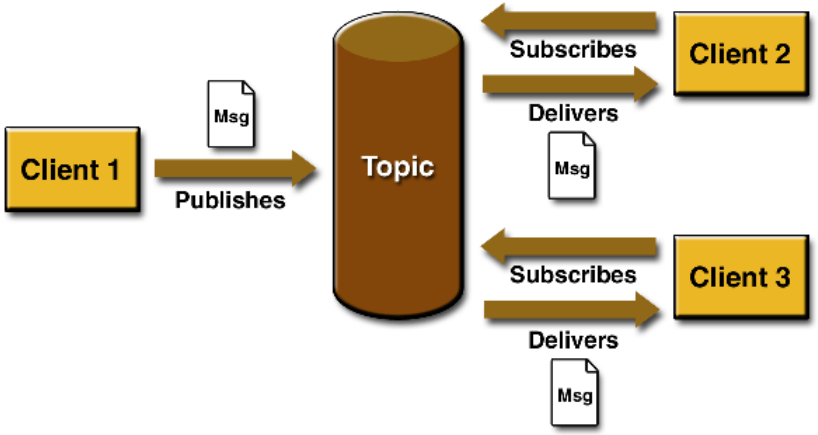
- JMS é um padrão para *Messaging*
- Tem como objetivo eliminar muitas das desvantagem que MOMs encontraram com o passar dos anos
- O Desenvolvedor aprende a usar a API JMS e reusa seu código com diferentes implementações plugáveis de MOM
  - ideia similar a APIs do JEE, como JNDI e JDBC

Domínios de Mensagens

- A arquitetura JMS suporte dois tipos modelos de troca de mensagens ou destinos (destinations):
- ***Point-to-point* (PTP) ou por filas (queues)**

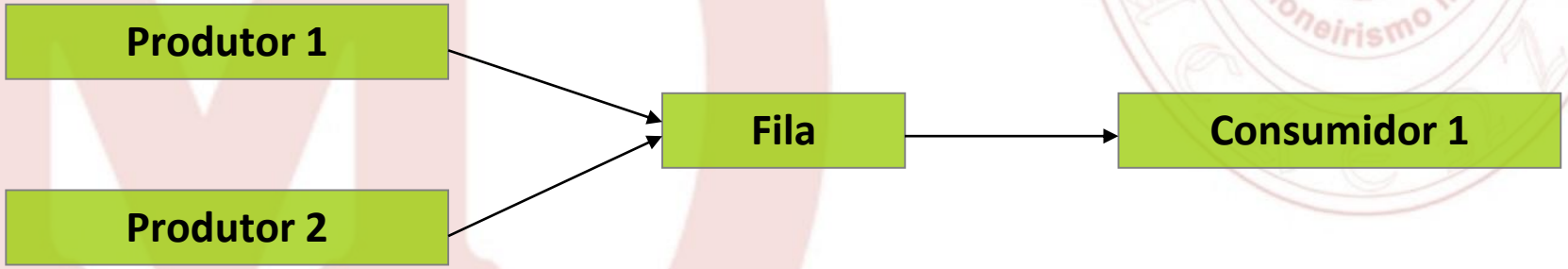


- ***Publish*-and-*subscribe* (pub/sub) ou por tópicos (topics)**



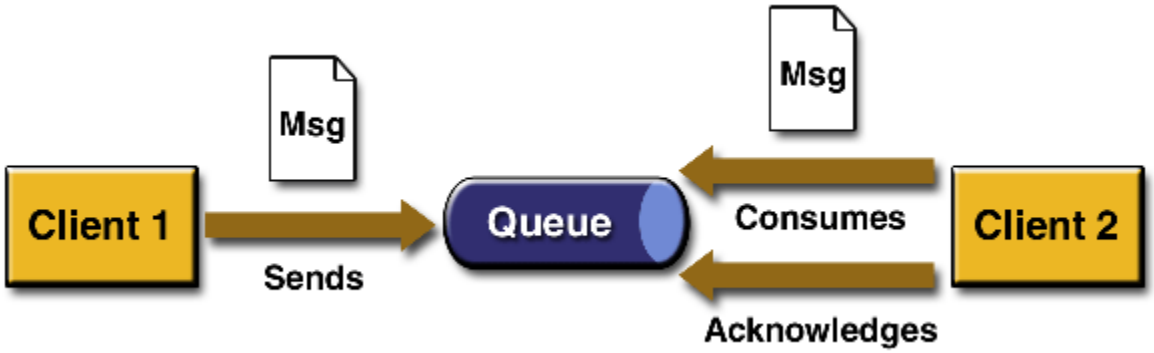
Point-to-point (PTP)

- Uma mensagem enviada para uma fila pode ser recebida por apenas **um** sistema
- um "produtor" (*producer*) envia mensagens para uma fila e um "consumidor" (*consumer*) as lê.
- Múltiplos produtores podem enviar mensagens para a fila mas será entregue a apenas um consumidor



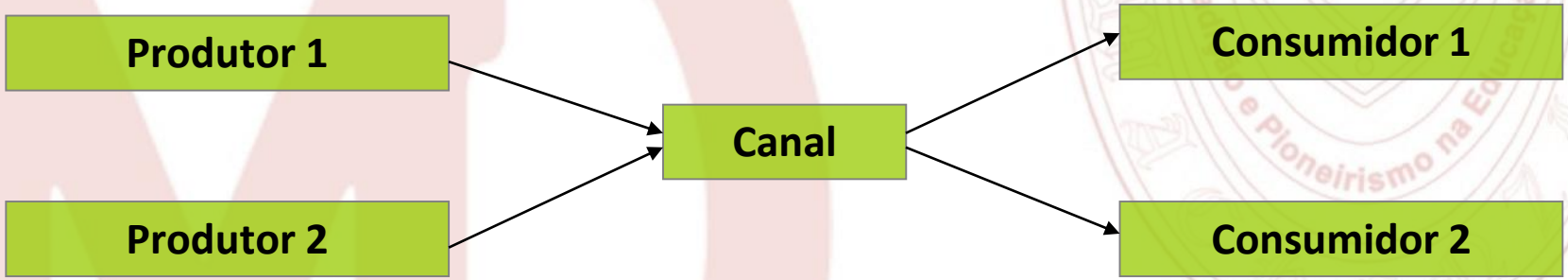
Point-to-point (PTP)

- Características do modelo:
  - Apenas um consumidor irá ler a mensagem;
  - Não é necessário que o produtor esteja em execução no momento em que o consumidor lê a mensagem;
  - Não é necessário que o consumidor esteja em execução no momento que o produtor envia a mensagem;
  - Quando lê uma mensagem com sucesso o consumidor envia um aviso (*acknowledged*) para o produtor.



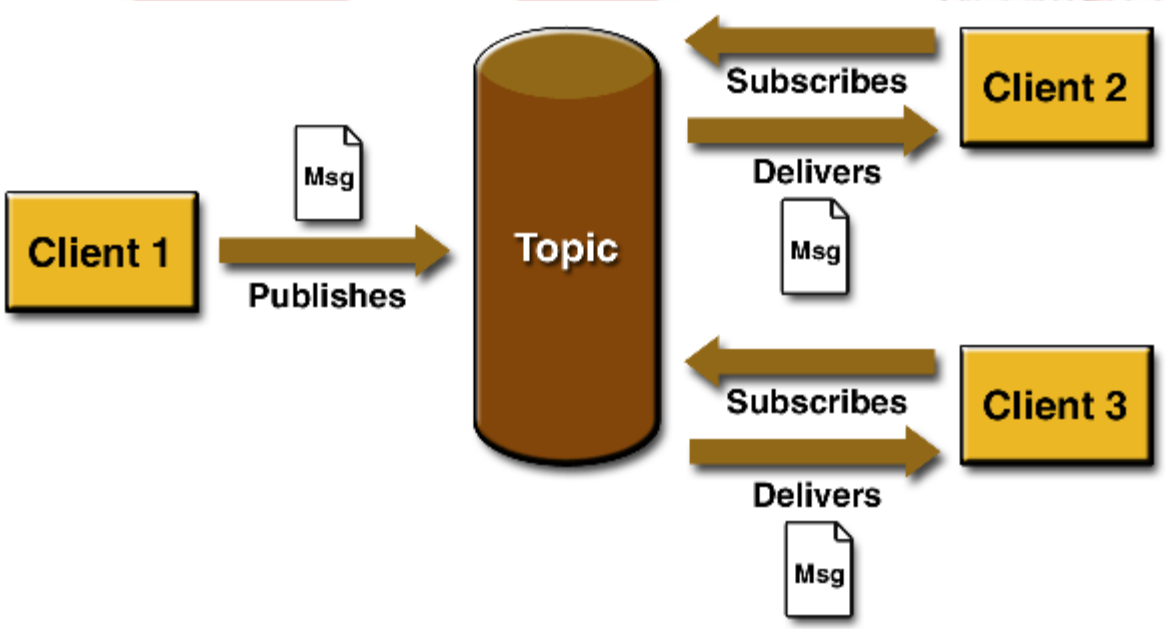
Publish/subscribe (pub/sub)

- Uma mensagem enviada para um tópico pode ser recebida por **diversos** sistemas
- Análogo a assistir televisão. Pode haver muitos produtores de mensagens e muitos consumidores.



Publish/subscribe (pub/sub)

- Características do modelo:
  - existe uma dependência temporal entre os publicadores e assinantes de um tópico.
  - Um editor deve criar uma "assinatura" (*subscription*) para que os assinantes possam receber mensagens.
  - O assinante do tópico deve estar em execução continuamente para receber as mensagens.



Administrador do MOM

- As filas e os tópicos são **objetos criados pelos administradores do MOM**.
- A especificação JMS não define uma forma padrão de criação desses objetos.
- Cada implementação JMS possui os seus próprios procedimentos para esse processo.
- No **Glassfish**, as filas e os tópicos podem ser criados através **da interface web de administração do servidor**.
- Toda fila ou tópico possui um **nome único** no MOM.

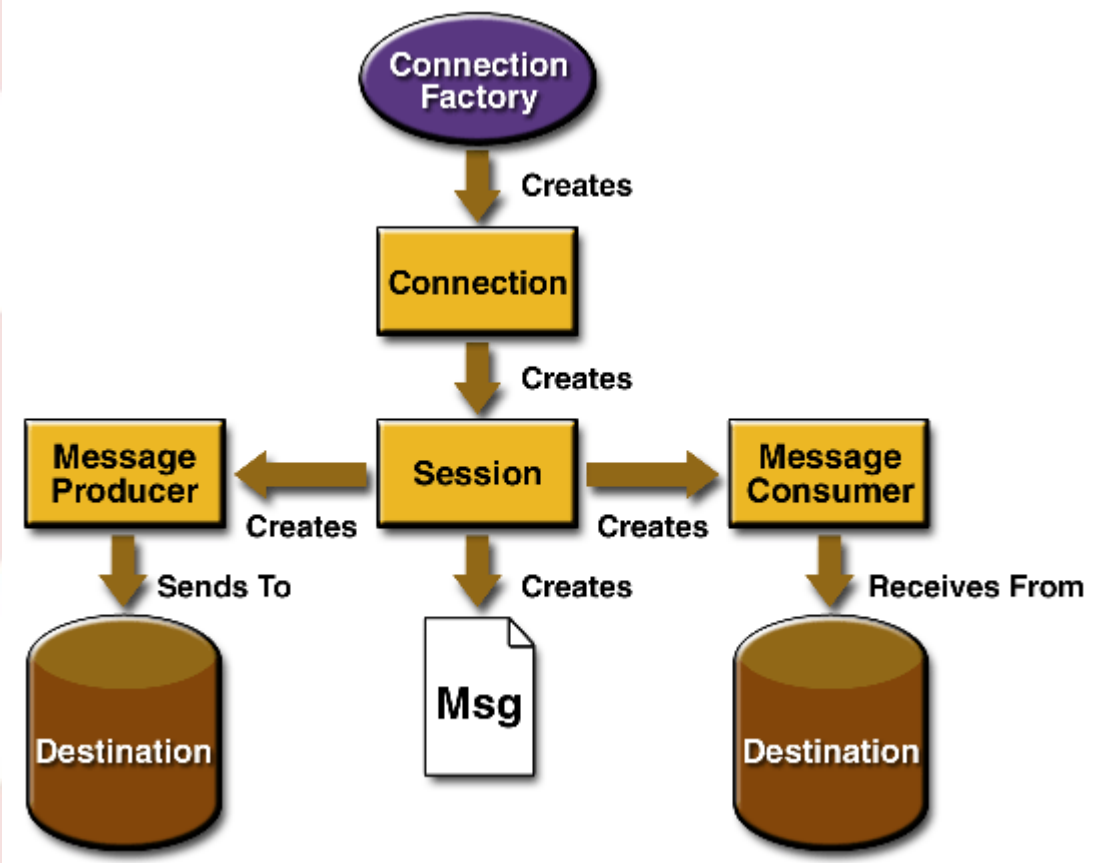
Fábrica de Conexões

- Os sistemas que desejam trocar mensagens através de filas ou tópicos devem **obter conexões** JMS através das **fábricas cadastradas no MOM**.
- As fábricas de conexões JMS também **são objetos criados pelos administradores do MOM.**
- Também não há uma forma padrão para criar essas fábricas, então cada implementação define a sua própria forma de criação.

Fábrica de Conexões

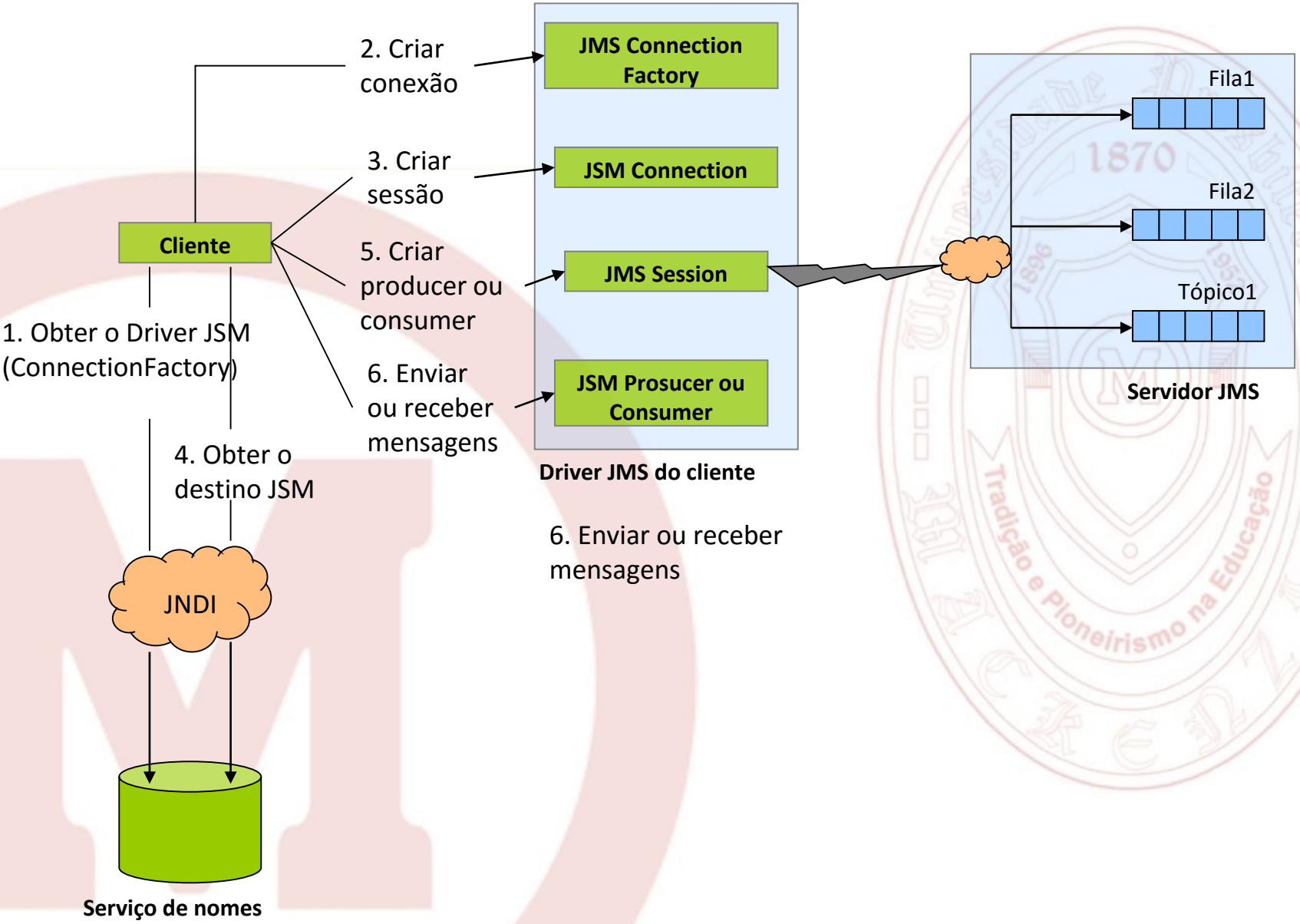
- No Glassfish as fábricas podem ser criadas através da **interface web de administração do servidor**.
- É possível criar fábricas especificadas para filas ou para tópicos ou genéricas que podem ser utilizadas para os dois tipos de destino.
- Toda fábrica possui um **nome único** no MOM.

Modelo de Programação na JMS



Usando a API JMS

- Passos
  1. Localizar o provedor JMS, instancia de **`ConnectionFactory`**
  2. Criar um conexão JMS
  3. Criar uma Sessão JMS
  4. Localizar o destino
  5. Criar um JMS Provider ou um JMS Consumer
  6. Enviar ou Receber suas mensagens


2. Criar conexão
JMS Connection Factory
3. Criar sessão
JSM Connection
Cliente
5. Criar producer ou consumer
JMS Session
1. Obter o Driver JSM (ConnectionFactory)
6. Enviar ou receber mensagens
JSM Prosucer ou Consumer
Driver JMS do cliente
Fila1
Fila2
Tópico1
Servidor JMS
4. Obter o destino JSM
6. Enviar ou receber mensagens
JNDI
Serviço de nomes

Obtenção filas, tópicos e fábricas

- As filas, tópicos e fábricas **são objetos criados** pelos administradores do MOM.
- Quando uma aplicação deseja **utilizar esses objetos**, ela deve obtê-los através de **pesquisas ao serviço de nomes do MOM**.
- O serviço de nomes é definido pela especificação JNDI.

Obtenção filas, tópicos e fábricas

```
InitialContext ctx = new InitialContext();

ConnectionFactory factory;
factory = (ConnectionFactory)ctx.lookup("ConnectionFactory");

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

TopicConnectionFactory factory;
factory = (TopicConnectionFactory)ctx.lookup("TopicConnectionFactory");

Queue queue;
queue = (Queue)ctx.lookup("Queue");

Topic topic;
topic = (Topic)ctx.lookup("Topic");
```

Obtenção filas, tópicos e fábricas

- Normalmente, os servidores de aplicação Java EE oferecem o recurso de **injeção de dependência** para que as aplicações obtenham as fábricas, filas ou tópicos.
- No Glassfish, é possível injetar esses objetos através da anotação **@Resource**.

Exemplo

```
public class Client {

    public static void main (String[] args) throws Exception {

        Context ctxt = new InitialContext();

        TopicConnectionFactory factory = (TopicConnectionFactory)
        ctxt.lookup("jms/TopicConnectionFactory");

        TopicConnection connection = factory.createTopicConnection();

        TopicSession session = connection.createTopicSession
        (false, Session.AUTO_ACKNOWLEDGE);

        Topic topic = (Topic) ctxt.lookup("jms/Topic");

        TopicPublisher publisher = session.createPublisher(topic);

        TextMessage msg = session.createTextMessage();

        msg.setText("This is a test message.");

        publisher.publish(msg);
    }
}
```

Interfaces

INTERFACE PAI	POINT-TO-POINT	PUB/SUB
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver, QueueBrowser	TopicSubscriber

Integração JMS-EJB

- Motivação
  - Possuir componentes EJBs com características como clientes “não-bloqueáveis” e comunicação n-ária

Integração JMS-EJB

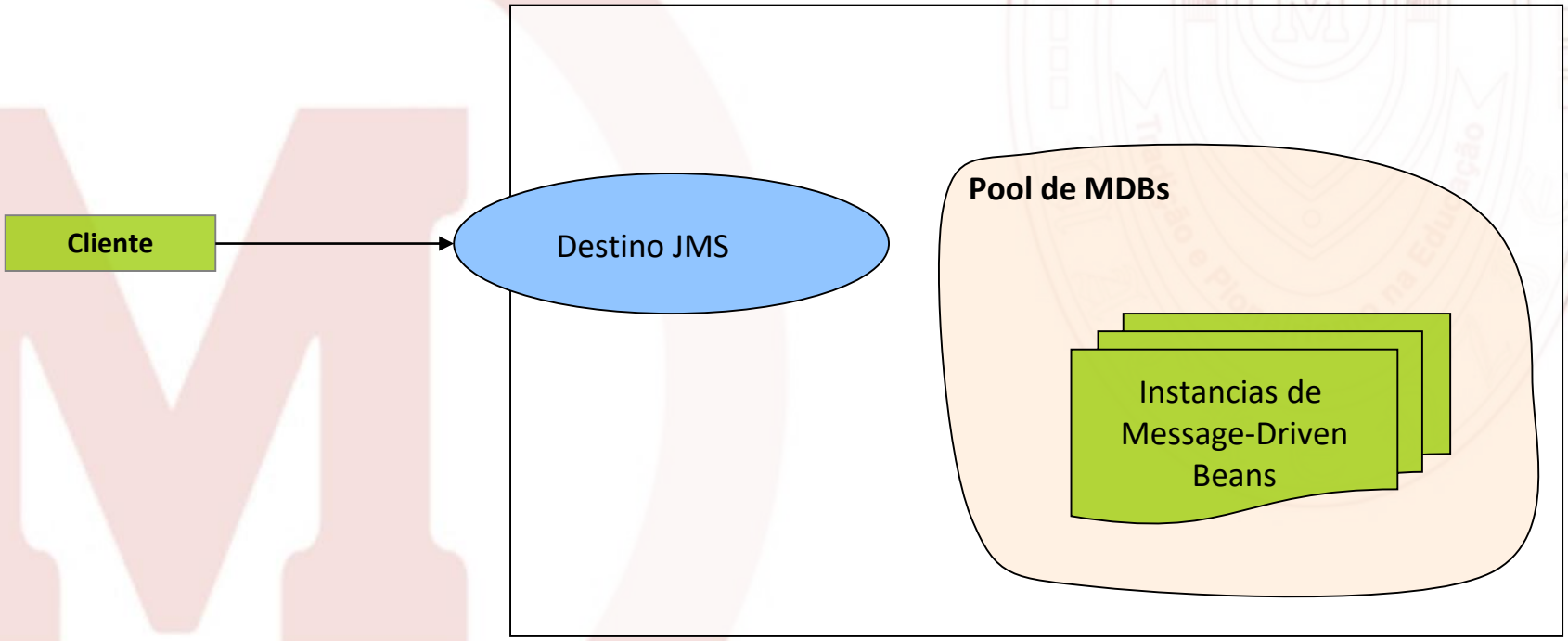
- Como implementar a integração?
  - Implementar objetos Java que recebem mensagens e realizam chamadas a componentes EJBs?
  - Reusar um tipo existente de componente EJB para receber Mensagens JMS?

Messsage-Driven Beans (MDB)

- O que são?
  - São componentes EJBs especiais capazes de receber mensagens enviadas a filas e tópicos JMS
  - Invocados pelo Contêiner dada a chegada de um mensagem ao destino que um MDB escuta
    - Não se envia uma mensagem direto a um MDB (envia-se ao tópico que o Bean escuta)
    - Proporcionando baixo acoplamento entre cliente e MDB (conhecimento do canal de comunicação)

Messsage-Driven Beans

– Para efetuar a comunicação é necessário o uso de uma API específica, como JMS



Messsage-Driven Beans

- Características:
  - Não possuem interface remota nem local
  - Possuem apenas um método que recebe qualquer tipo de mensagem
  - Não têm retorno, e também não lançam exceções ao cliente
  - São *Stateless*
  - Podem ser ouvintes de uma fila, ou assinantes durável ou não-durável de um tópico

Implementando MDBs

- Para implementar um MDB, deve-se:
  - Aplicar a anotação @MessageDriven
  - Implementar a interface
  - `javax.jms.MessageListener`
  - Método de MessageListener
  - `onMessage(Message m):` chamado cada vez que uma mensagem é enviada para o tópico do bean (se o bean estiver ativado).

Implementando MDBs

```
@MessageDriven(mappedName="jms/dest")
public class TratadorDeMensagensMDB implements MessageListener{
@Override
    public void onMessage(Message message) {

    try {
        TextMessage msg = (TextMessage) message;
        System.out.println(msg.getText());
    } catch (JMSException e) {
            System.out.println("erro");
        }
    }
}
```

Implementando cliente

- Cliente JMS que **envia** uma mensagem JMS para a fila pedidos

```
public class EnviaNovoPedido {
    public static void main(String[] args) throws Exception {
        // serviço de nomes - JNDI
        InitialContext ctx = new InitialContext();
        // fábrica de conexões JMS
        ConnectionFactory factory = (ConnectionFactory) ctx.lookup("jms/Factory");
        // fila
        Queue queue = (Queue) ic.lookup("jms/pedidos");
        // conexão JMS
        Connection connection = factory.createConnection();
        // sessão JMS
        Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
        // emissor de mensagens
        MessageProducer sender = session.createProducer(queue);
        // mensagem
        TextMessage message = session.createTextMessage();
        message.setText("Uma pizza de margherita" + System.currentTimeMillis());
        // enviando
        sender.send(message);
        // fechando
        sender.close();
        session.close();
        connection.close();

        System.out.println("Mensagem enviada");
        System.exit(0);
    }
}
```

Implementando cliente

- Cliente JMS comum que **recebe** uma mensagem JMS da fila pedidos

```
public class RecebePedido {
    public static void main(String[] args) throws Exception {
        // serviço de nomes - JNDI
        InitialContext ctx = new InitialContext();
        // fábrica de conexões JMS
        ConnectionFactory factory = (ConnectionFactory) ctx.lookup("jms/Factory");
        // fila
        Queue queue = (Queue) ic.lookup("jms/pedidos");
        // conexão JMS
        Connection connection = factory.createConnection();
        // sessão JMS
        Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
        // receptor de mensagens
        MessageConsumer receiver = session.createConsumer(queue);
        // inicializa conexão
        connection.start();
        // recebendo
        TextMessage message = (TextMessage) receiver.receive();
        System.out.println(message.getText());
        // fechando
        receiver.close();
        session.close();
        connection.close();

        System.out.println("FIM");
        System.exit(0);
    }
}
```

Bibliografia

- Integração de Sistemas com Webservices, JMS e EJB. Treinamento K19.
- Enterprise JavaBeans- Tutorial Java EE 6. http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html

Obrigado

Prof. Leandro Pupo Natale

Leandro.natale@mackenzie.br