Corso di Laurea in Informatica
III Anno Triennale
Programmazione Distribuita – classe 1

Messaging (2)

Delfina Malandrino
dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino

Organizzazione della lezione

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Conclusioni

Organizzazione della lezione

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Conclusioni

Come assicurare un recapito affidabile

- JMS definisce diversi livelli di affidabilità per assicurare che i messaggi siano instradati correttamente
  - anche se il provider va in crash o è sotto carico elevato

Come assicurare un recapito affidabile

- I meccanismi sono i seguenti
  - Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - Setting message time-to-live: scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - Controlling acknowledgment: controllo degli ack a vari livelli
  - Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - Setting priorities: priorità di messaggi

Come assicurare un recapito affidabile

- I meccanismi sono i seguenti
  - **Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano**
  - Setting message time-to-live: scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - Controlling acknowledgment: controllo degli ack a vari livelli
  - Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - Setting priorities: priorità di messaggi

Filtering dei messaggi

- Si fa in modo che arrivino solo i messaggi a cui si è interessati
  - Messaggio mandato in broadcast a diversi client, si definisce un selector in modo che venga consumato solo da consumer interessati
- Nessuno spreco di tempo e banda per ricevere cose non di interesse
- Si può fare selezione su headers o metadati (JMSPriority < 6) o su proprietà custom (orderAmount < 200)
- Il message selector è una stringa che contiene una espressione:

```
context.createConsumer(queue, "JMSPriority < 6").receive();
context.createConsumer(queue, "JMSPriority < 6 AND orderAmount < 200").receive();
context.createConsumer(queue, "orderAmount BETWEEN 1000 AND 2000").receive();
```

Filtering dei messaggi

- Il messaggio viene creato dal Producer usando metodi per settare proprietà e priorità (nell'header)

```
context.createTextMessage().setIntProperty("orderAmount", 1530);
context.createTextMessage().setJMSPriority(5);
```

Filtering dei messaggi

Producer

```
context.createTextMessage().setIntProperty("orderAmount", 1530);
context.createTextMessage().setJMSPriority(5);
```

Consumer

```
context.createConsumer(queue, "JMSPriority < 6").receive();
context.createConsumer(queue, "JMSPriority < 6 AND orderAmount < 200").receive();
context.createConsumer(queue, "orderAmount BETWEEN 1000 AND 2000").receive();
```

Filtering dei messaggi

- Selector expression possono usare:
  - logical operators (NOT, AND, OR)
  - comparison operators (=, >, >=, <, <=, <>)
  - Arithmetic operators (+, -, *, /)
  - expressions ([NOT] BETWEEN, [NOT] IN, [NOT] LIKE, IS [NOT] NULL)
  - and so on.

Come assicurare un recapito affidabile

- I meccanismi sono i seguenti
  - Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - **Setting message time-to-live: scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti**
  - Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - Controlling acknowledgment: controllo degli ack a vari livelli
  - Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - Setting priorities: priorità di messaggi

Evitare messaggi obsoleti

- Un setting del time-to-live può essere di aiuto per evitare che messaggi obsoleti vengano recapitati ai destinatari
- Si setta il tempo in millisecondi, passato il quale il provider (il broker) rimuove il messaggio
- Si utilizza il metodo del producer:

```
context.createProducer().setTimeToLive(1000).send(queue, message);
```

Come assicurare un recapito affidabile

- I meccanismi sono i seguenti
  - Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - Setting message time-to-live: scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - **Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)**
  - Controlling acknowledgment: controllo degli ack a vari livelli
  - Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - Setting priorities: priorità di messaggi

Gestire la persistenza

- JMS supporta 2 modalità di message delivery: persistent e nonpersistent
  - Persistent delivery: messaggio salvato sul provider (disk/database)
    - Messaggi non persi in caso di restart del broker
  - Non-persistent delivery: messaggio non salvato
- Persistent delivery è il valore di default… che può essere "degradato" per migliorare le prestazioni

```
context.createProducer().setDeliveryMode(DeliveryMode.NON_PERSISTENT).send(queue, message);
```

Come assicurare un recapito affidabile

- I meccanismi sono i seguenti
  - Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - Setting message time-to-live: scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - **Controlling acknowledgment: controllo degli ack a vari livelli**
  - Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - Setting priorities: priorità di messaggi

Controllo degli Acknowledgment

- Si vuole ricevere una verifica del recapito del messaggio al destinatario
- Diverse modalità di acknowledgment:
  - AUTO_ACKNOWLEDGE: la sessione automaticamente fa ack di un messaggio
  - CLIENT_ACKNOWLEDGE: ack esplicito del client
    - chiamando il metodo Message.acknowledge()

Controllo degli Acknowledgment

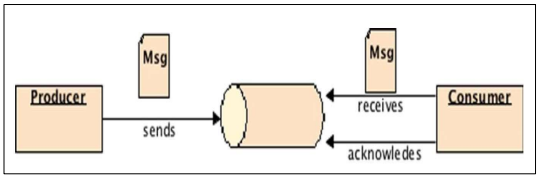
- Esempio di Producer che usa l'annotazione @JMSSessionMode per settare l'acknowledgment mode

```
// Producer
@Inject
@JMSConnectionFactory("jms/connectionFactory")
@JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
private JMSContext context;
...
context.createProducer().send(queue, message);
```

Controllo degli Acknowledgment

- Esempio di consumer che fa esplicito acknowledges del messaggio chiamando il metodo acknowledge()

```
// Consumer
message.acknowledge();
```



Come assicurare un recapito affidabile

- I meccanismi sono i seguenti
  - Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - Setting message time-to-live: scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - Controlling acknowledgment: controllo degli ack a vari livelli
  - **Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model**
  - Setting priorities: priorità di messaggi

Durable Consumer

- Nel modello publish-subscribe un consumer che non è in esecuzione perde i messaggi che vengono postati sul topic
- Con i **durable consumer** si può controllare che i messaggi vengano mantenuti dal provider fino a quando tutti i consumer li hanno ricevuti

Durable Consumer

- Con i durable suscribers, un consumer che si riconnette riceve i messaggi che sono arrivati durante la disconnessione
- Creazione attraverso JMSContext con una id specifica “unica”

```
context.createDurableConsumer(topic,"uniqueID").receive();
```

```
context.createDurableConsumer(topic, "javaee7DurableSubscription").receive();
```

- Quando si crea un durable consumer, si registra una "durable subscription" presso il broker JMS **con un nome univoco**. Questo nome viene usato dal broker per salvare i messaggi mentre si è offline

Durable Consumer: **RICAPITOLANDO**

```
context.createDurableConsumer(topic,"uniqueID").receive();
```

- A questo punto il client inizia la connessione e riceve messaggi
- Il nome "uniqueID" (nel nostro esempio javaee7DurableSubscription) è usato come identificatore della durable subscription (identificativo del subscriber)
- Ogni durable consumer deve Impostare un client ID per il durable subscriber

Durable Consumer: **RICAPITOLANDO**

```
context.createDurableConsumer(topic,"uniqueID").receive();
```

- Topic: È il canale sul quale i messaggi vengono pubblicati
- ”uniqueID”: questo è il nome univoco della durable subscription e serve al provider JMS per identificare la subscription e salvare i messaggi per quel consumer specific
- Una volta che il consumer è stato creato, è associato alla subscription con questo ID. Se viene eseguito nuovamente con lo stesso uniqueID, il provider JMS sa che appartiene allo stesso consumer

Durable Consumer

```
context.createDurableConsumer(topic,"uniqueID").receive();
```

- L’identificativo unico associato al durable subscriber serve al JMS server per memorizzare messaggi arrivati mentre il subscriber non è attivo
- Quando il subscriber si riconnette il JMS server provvede a inviare tutti i messaggi ancora validi accumulati fino a quel momento

Come assicurare un recapito affidabile

- I meccanismi sono i seguenti
  - Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - Setting message time-to-live: scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - Controlling acknowledgment: controllo degli ack a vari livelli
  - Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - **Setting priorities: priorità di messaggi**

Scegliere priorità del messaggio

- Si istruisce il JMS provider a fare il delivery di messaggi urgenti per primi
- Contenute nell'header del messaggio
- Valori da 0 (bassa priorità) a 9 (alta priorità)
- Esempio:

```
context.createProducer().setPriority(2).send(queue, message);
```

- Concatenazione di diversi meccanismi:

```
context.createProducer().setPriority(2)
                        .setTimeToLive(1000)
                        .setDeliveryMode(DeliveryMode.NON_PERSISTENT)
                        .send(queue, message);
```

Organizzazione della lezione

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Conclusioni

EJB che scambiano messaggi

- Un Message Driven Bean (MDB) è un **consumatore di messaggi**, asincrono, invocato dal container quando arriva un messaggio
- Parte delle specifiche di Enterprise JavaBeans: simili a stateless
- Tramite CDI può accedere a altri EJB, JDBC, risorse JMS, entity manager, etc.
- **Perché usare un MDB anziché un JMS Client?**
  - Il vantaggio di usare un MDB (rispetto ad un JMS Client) è che transazione, multithread, sicurezza, etc., sono gestiti dal container

Un esempio di un semplice MDB

```
@MessageDriven(mappedName =
    "jms/javaee7/Topic")
public class BillingMDB
        implements MessageListener

 public void onMessage(Message message)
     {  System.out.println("Received:"
       + message.getBody(String.class));
 }
```

› Definisce un MDB

Un esempio di un semplice MDB

```
@MessageDriven(mappedName =
    "jms/javaee7/Topic")
public class BillingMDB
        implements MessageListener

 public void onMessage(Message message)
     {  System.out.println("Received:"
       + message.getBody(String.class));
 }
```

› Definisce un MDB

› Implementa un `MessageListener` e quindi il metodo onMessage()

Un esempio di un semplice MDB

```
@MessageDriven(mappedName =
    "jms/javaee7/Topic")
public class BillingMDB
        implements MessageListener

 public void onMessage(Message message)
     {  System.out.println("Received:"
       + message.getBody(String.class));
 }
```

- Definisce un MDB
- Implementa un `MessageListener` e quindi il metodo onMessage()

Cosa fare quando si riceve un messaggio

Come è fatto un MDB

- MDB non è parte del modello EJB Lite: serve una implementazione full EE
- Annotazione con **`@javax.ejb.MessageDriven`** (o XML equivalente)
- Implementare l'interfaccia del listener
- Definita come public, non final o abstract
- Deve esserci un costruttore senza argomenti per permetterne l'istanziazione automatica da parte del container
- La classe non deve avere il metodo finalize()

Come configurare un MDB

- Possibile usare tutti i setting realizzabili con JMS

Property	Description
acknowledgeMode	The acknowledgment mode (default is AUTO_ACKNOWLEDGE)
messageSelector	The message selector string used by the MDB
destinationType	The destination type, which can be TOPIC or QUEUE
destinationLookup	The lookup name of an administratively-defined Queue or Topic
connectionFactoryLookup	The lookup name of an administratively defined ConnectionFactory
destination	The name of the destination.
subscriptionDurability	The subscription durability (default is NON_DURABLE)
subscriptionName	The subscription name of the consumer
shareSubscriptions	Used if the message-driven bean is deployed into a clustered
clientId	Client identifier that will be used when connecting to the JMS provider

Come usare le proprietà di un MDB

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount < 3000")
})
public class BillingMDB implements MessageListener {

    public void onMessage(Message message) {
        System.out.println("Message received:"+
                        message.getBody(String.class));
    }
```

Nella definizione del MDB

Come usare le proprietà di un MDB

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount < 3000")
})
public class BillingMDB implements MessageListener {

    public void onMessage(Message message) {
        System.out.println("Message received:"+
                        message.getBody(String.class));
    }
```

- Nella definizione del MDB
- Viene definita la configurazione …

Come usare le proprietà di un MDB

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount < 3000")
})
public class BillingMDB implements MessageListener {

    public void onMessage(Message message) {
        System.out.println("Message received:"+
                        message.getBody(String.class));
    }
```

- Nella definizione del MDB
- Viene definita la configurazione …
- … con le sue proprietà

Come usare le proprietà di un MDB

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount < 3000")
})
public class BillingMDB implements MessageListener {

    public void onMessage(Message message) {
        System.out.println("Message received:"+
                        message.getBody(String.class));
    }
```

- Nella definizione del MDB
- Viene definita la configurazione…
- …con le sue proprietà
- …filtro di messaggi

Come usare le proprietà di un MDB

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount < 3000")
})
public class BillingMDB implements MessageListener {

    public void onMessage(Message message) {
        System.out.println("Message received:"+
                        message.getBody(String.class));
    }
```

- Nella definizione del MDB
- Viene definita la configurazione…
- …con le sue proprietà
- …filtro di messaggi
- Metodo per gestire i messaggi

Dependency Injection

- Come per tutti gli EJB, gli MDBs possono usare Dependency Injection per ottenere riferimenti a risorse come JDBC datasources, EJBs, o altri oggetti
- Injection è il meccanismo attraverso il quale il container inserisce le dipendenze automaticamente dopo aver creato l'oggetto
- Queste risorse devono essere disponibili nel container oppure nell'environment context

```
@PersistenceContext
private EntityManager em;
@Inject
private InvoiceBean invoice;
@Resource(lookup = "jms/javaee7/ConnectionFactory")
private ConnectionFactory connectionFactory;
```

The MDB context can also be injected using the `@Resource` annotation:

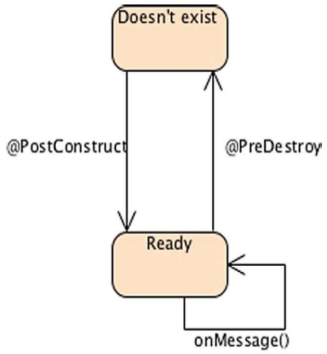
```
@Resource private MessageDrivenContext context;
```

MDB context: `MessageDrivenContext`

- Questa interfaccia fornisce accesso al runtime context, che il container fornisce per una istanza di un MDB
- Il container passa l'interfaccia MessageDrivenContext all'istanza, che rimane associata per il lifetime dell'MDB
- Permette all'MDB di fare roll back di una transazione, ottenere il caller (user principal), ecc.

Method	Description
`getCallerPrincipal`	Returns the `java.security.Principal` associated with the invocation
`getRollbackOnly`	Tests whether the current transaction has been marked for rollback
`getTimerService`	Returns the `javax.ejb.TimerService` interface
`getUserTransaction`	Returns the `javax.transaction.UserTransaction` interface to use to demarcate transactions. Only MDBs with bean-managed transaction (BMT) can use this method
`isCallerInRole`	Tests whether the caller has a given security role
`Lookup`	Enables the MDB to look up its environment entries in the JNDI naming context
`setRollbackOnly`	Allows the instance to mark the current transaction as rollback. Only MDBs with BMT can use this method

Il ciclo di vita di un MDB



- Simile a quello degli stateless beans
- Possibile inserire interceptors dei metodi

MDB as a consumer

- Per natura, gli MDBs sono progettati per funzionare come asynchronous message consumers
- Gli MDBs implementano una message listener interface, che viene "risvegliata" (triggered) dal container quando un messaggio arriva

MDB as a consumer

- **Può un MDB essere un synchronous consumer?**
- SI, ma non è raccomandato
  - Synchronous message consumers bloccano le risorse del server (gli EJBs si bloccheranno in un loop senza eseguire nessun lavoro ed il container non sarà in grado di liberarli)
  - Gli MDBs, come gli stateless session beans, vivono in un pool di una certa taglia
    - Quando il container ha bisogno di una istanza la prende dal pool e la usa
    - Se l'istanza va in un loop infinito, il pool si svuoterà e tutte le risorse saranno bloccate in un busy looping
  - L' EJB container può generare nuove istanze di MDB incrementando il pool ma aumentando così il consumo di memoria
  - Per questa ragione, **session beans e MDBs non dovrebbero essere usati come synchronous message consumers**

MDB as a producer

- Gli MDBs possono ANCHE diventare message producers
  - Workflow che prevede che essi ricevano messaggi da una destinazione, li processino, e li rinviino ad un'altra destinazione
- Per aggiungere questa capacità bisogna usare le API JMS
- Vediamo un esempio....

Esempio di MDB che riceve ed invia messaggi

- MDB con il topic

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

Esempio di MDB che riceve ed invia messaggi

- MDB con il topic
- Configurazione

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack
- Filtro messaggi

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack
- Filtro messaggi
- Implementa listener

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack
- Filtro messaggi
- Implementa listener
- Context iniettato dalla CF

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack
- Filtro messaggi
- Implementa listener
- Context iniettato dalla CF
- …nella variabile

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack
- Filtro messaggi
- Implementa listener
- Context iniettato dalla CF
- …nella variabile
- Destinazione iniettata dal container

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack
- Filtro messaggi
- Implementa listener
- Context iniettato dalla CF
- …nella variabile
- Destinazione iniettata dal container
- Metodo quando riceve messaggi

Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                propertyValue ="orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup ="jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
                + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage()throws JMSException
        {  context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
```

- MDB con il topic
- Configurazione
- Auto-ack
- Filtro messaggi
- Implementa listener
- Context iniettato dalla CF
- …nella variabile
- Destinazione iniettata dal container
- Metodo quando riceve messaggi
- Metodo per inviare

Transazioni… anche per i messaggi

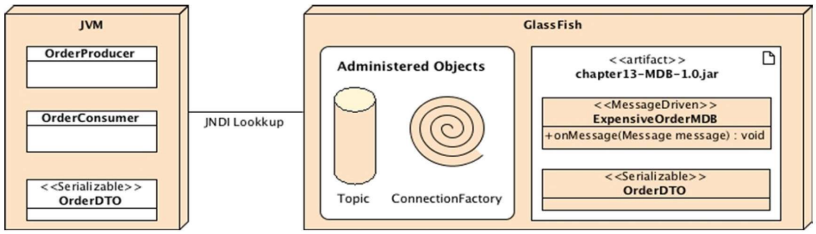
- Transazioni per scambio di messaggi: un certo numero di messaggi vanno recapitati tutti insieme o nessuno
- In quanto EJB, le transazioni possono essere Bean-managed oppure Container-managed

Organizzazione della lezione

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Conclusioni

Lo schema

- Producer stand-alone, `OrderProducer`, che invia messaggi a un topic, con un valore di amount (settato su linea di comando del producer)
- Il consumer `OrderConsumer` riceve tutti i messaggi
- . . . mentre l'MDB `ExpensiveOrderMDB` riceve solamente quelli sopra i 1000



L'ordine

```
public class OrderDTO implements Serializable {
    private Long orderId;
    private Date creationDate;
    private String customerName;
    private Float totalAmount;

    //Constructors,getters,setters
```

Deve essere trasmesso in un messaggio

L'ordine

```
public class OrderDTO implements Serializable {
    private Long orderId;
    private Date creationDate;
    private String customerName;
    private Float totalAmount;

    //Constructors,getters,setters
```

- Deve essere trasmesso in un messaggio
- Campi dell'ordine

L'ordine

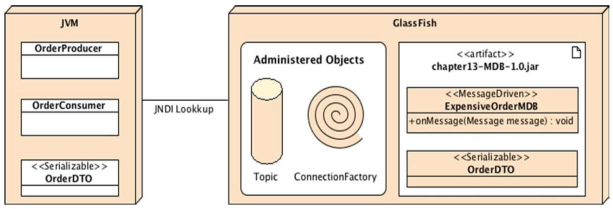
```
public class OrderDTO implements Serializable {
    private Long orderId;
    private Date creationDate;
    private String customerName;
    private Float totalAmount;

    //Constructors,getters,setters
```

- Deve essere trasmesso in un messaggio
- Campi dell'ordine
- Tra cui l'ammontare totale

Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                            "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                       jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAm
                       totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando



Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                            "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                       jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAmount",
                       totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando
- Si crea un nuovo ordine

Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                              "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAmount",
                      totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando
- Si crea un nuovo ordine
- Si ottiene il contesto JNDI

Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                              "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAmount",
                      totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando
- Si crea un nuovo ordine
- Si ottiene il contesto JNDI
- Lookup degli oggetti administered: CF

Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                              "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAmount",
                      totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando
- Si crea un nuovo ordine
- Si ottiene il contesto JNDI
- Lookup degli oggetti administered: CF
- …e topic

Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                              "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAmount",
                      totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando
- Si crea un nuovo ordine
- Si ottiene il contesto JNDI
- Lookup degli oggetti administered: CF
- …e topic
- Con il contesto creato…

Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                            "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAmount",
                      totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando
- Si crea un nuovo ordine
- Si ottiene il contesto JNDI
- Lookup degli oggetti administered: CF
- …e topic

Con il contesto creato…

- Si crea un produttore con una proprietà

Il Produttore

```
public class OrderProducer {
 public static void main(String[] args)
                                throws NamingException {

  Float totalAmount = Float.valueOf(args[0]);

  OrderDTO order = new OrderDTO(12341,new Date(),
                            "Betty Moreau", totalAmount);

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
       jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic");
  try(JMSContext jmsContext =
        connectionFactory.createContext()) {
      jmsContext.createProducer().setProperty("orderAmount",
                      totalAmount).send(topic, order);
  }
 }
}
```

- Parametro passato su linea di comando
- Si crea un nuovo ordine
- Si ottiene il contesto JNDI
- Lookup degli oggetti administered: CF
- …e topic

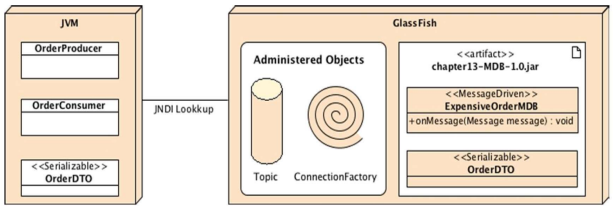
Con il contesto creato…

- Si crea un produttore con una proprietà
- e si invia il messaggio

Il consumatore standard JSE

```
public class OrderConsumer {
 public static void main(String[] args)
                                throws NamingException {

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic";

  try(JMSContext jmsContext=
                      connectionFactory.createContext()) {
   while(true) {
    OrderDTO order = jmsContext.createConsumer(topic)
                                .receiveBody(OrderDTO.class);
    System.out.println("Order received:"+ order);
   }
  }
 }
}
```

- Il contesto dall'ambiente di esecuzione



Il consumatore standard JSE

```
public class OrderConsumer {
 public static void main(String[] args)
                                throws NamingException {

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic";

  try(JMSContext jmsContext=
                      connectionFactory.createContext()) {
   while(true) {
    OrderDTO order = jmsContext.createConsumer(topic)
                                .receiveBody(OrderDTO.class);
    System.out.println("Order received:"+ order);
   }
  }
 }
}
```

- Il contesto dall'ambiente di esecuzione
- Lookup oggetti administered: CF e destinazione

Il consumatore standard JSE

```
public class OrderConsumer {
 public static void main(String[] args)
                                throws NamingException {

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic";

  try(JMSContext jmsContext=
                       connectionFactory.createContext()) {
   while(true) {
    OrderDTO order = jmsContext.createConsumer(topic)
                               .receiveBody(OrderDTO.class);
    System.out.println("Order received:"+ order);
   }
  }
 }
}
```

- Il contesto dall'ambiente di esecuzione
- Lookup oggetti administered: CF e destinazione
- Con il contesto ottenuto

Il consumatore standard JSE

```
public class OrderConsumer {
 public static void main(String[] args)
                                throws NamingException {

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic";

  try(JMSContext jmsContext=
                       connectionFactory.createContext()) {
   while(true) {
    OrderDTO order = jmsContext.createConsumer(topic)
                               .receiveBody(OrderDTO.class);
    System.out.println("Order received:"+ order);
   }
  }
 }
}
```

- Il contesto dall'ambiente di esecuzione
- Lookup oggetti administered: CF e destinazione
- Con il contesto ottenuto
- Va in loop infinito…

Il consumatore standard JSE

```
public class OrderConsumer {
 public static void main(String[] args)
                                throws NamingException {

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic";

  try(JMSContext jmsContext=
                      connectionFactory.createContext()) {
   while(true) {
    OrderDTO order = jmsContext.createConsumer(topic)
                                .receiveBody(OrderDTO.class);
    System.out.println("Order received:"+ order);}
  }
 }
}
```

- Il contesto dall'ambiente di esecuzione
- Lookup oggetti administered: CF e destinazione
- Con il contesto ottenuto
- Va in loop infinito…
- Crea un consumer

Il consumatore standard JSE

```
public class OrderConsumer {
 public static void main(String[] args)
                                throws NamingException {

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                      jndiContext.lookup("jms/javaee7/Topic";

  try(JMSContext jmsContext=
                      connectionFactory.createContext()) {
   while(true) {
    OrderDTO order = jmsContext.createConsumer(topic)
                                .receiveBody(OrderDTO.class);
    System.out.println("Order received:"+ order);
   }
  }
 }
}
```

- Il contesto dall'ambiente di esecuzione
- Lookup oggetti administered: CF e destinazione
- Con il contesto ottenuto
- Va in loop infinito…
- Crea un consumer
- Dal quale riceve messaggi

Il consumatore standard JSE

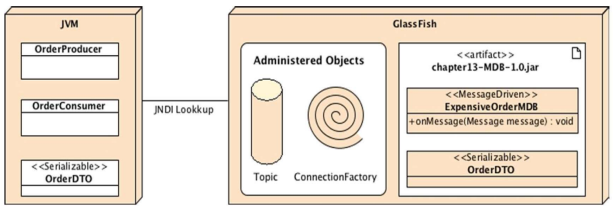
```
public class OrderConsumer {
 public static void main(String[] args)
                                throws NamingException {

  Context jndiContext = new InitialContext();

  ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
  Destination topic = (Destination)
                     jndiContext.lookup("jms/javaee7/Topic";

  try(JMSContext jmsContext=
                      connectionFactory.createContext()) {
   while(true) {
    OrderDTO order = jmsContext.createConsumer(topic)
                               .receiveBody(OrderDTO.class);
    System.out.println("Order received:"+ order);
   }
  }
 }
}
```

- Il contesto dall'ambiente di esecuzione
- Lookup oggetti administered: CF e destinazione
- Con il contesto ottenuto
- Va in loop infinito…
- Crea un consumer
- Dal quale riceve messaggi
- Che stampa a video

Message-driven Bean

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                              propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                              propertyValue ="orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
         {  try{
             OrderDTO order = message.getBody(OrderDTO.class);
             System.out.println("Expensive order received:"
                                              + orde
         }catch(JMSException e) {
             e.printStackTrace();
         }
    }
}
```

- MDB con il nome del topic da usare



Message-driven Bean

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
        {  try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                                        + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- MDB con il nome del topic da usare
- Configurazione: auto-ack

Message-driven Bean

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
        {  try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                                        + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- MDB con il nome del topic da usare
- Configurazione: auto-ack
- . . . e selettore messaggi con ammontare alto

Message-driven Bean

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
        {  try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                                        + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- MDB con il nome del topic da usare
- Configurazione: auto-ack
- . . . e selettore messaggi con ammontare alto
- Definizione dell'MDB

Message-driven Bean

```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
        {  try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                                        + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- MDB con il nome del topic da usare
- Configurazione: auto-ack
- . . . e selettore messaggi con ammontare alto
- Definizione dell'MDB
- Riceve il messaggio

Message-driven Bean

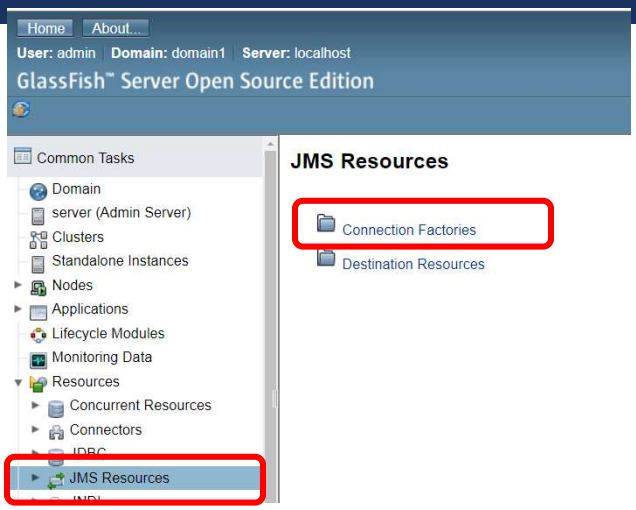
```
@MessageDriven(mappedName ="jms/javaee7/Topic",
 activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
                        propertyValue ="Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
                        propertyValue ="orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
        {  try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:
                                        + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- MDB con il nome del topic da usare
- Configurazione: auto-ack
- . . . e selettore messaggi con ammontare alto
- Definizione dell'MDB
- Riceve il messaggio
- Lo stampa a video

La configurazione degli oggetti administered

- Necessario che il server abbia:
  - Il Connection Factory
  - Il Topic
- Configurazione possibile da Console Web e da linea di comando
- Nella creazione del Topic necessario anche creare la destinazione fisica (creata di default con linea di comando)

Configurazione via Web Console



Configurazione via Web Console

Configurazione via Web Console

New JMS Connection Factory

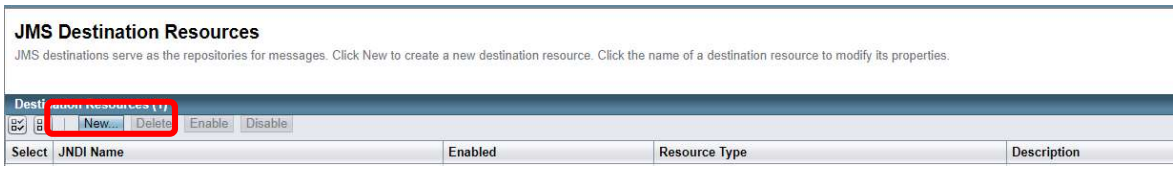
The creation of a new Java Message Service (JMS) connection factory also creates a connector connection pool for

General Settings

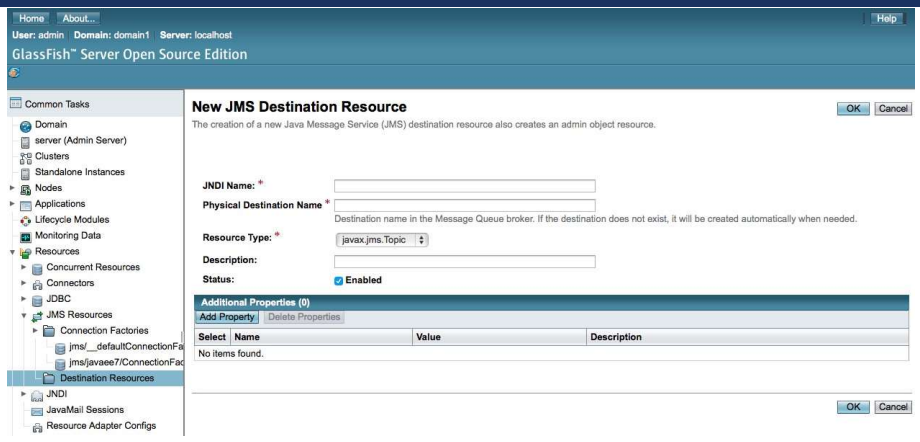
JNDI Name: *	jms/javaee7/ConnectionFactory
Resource Type:	javax.jms.ConnectionFactory
Description:	Connection factory esempio di prova
Status:	**Enabled**

Configurazione via Web Console

Configurazione via Web Console



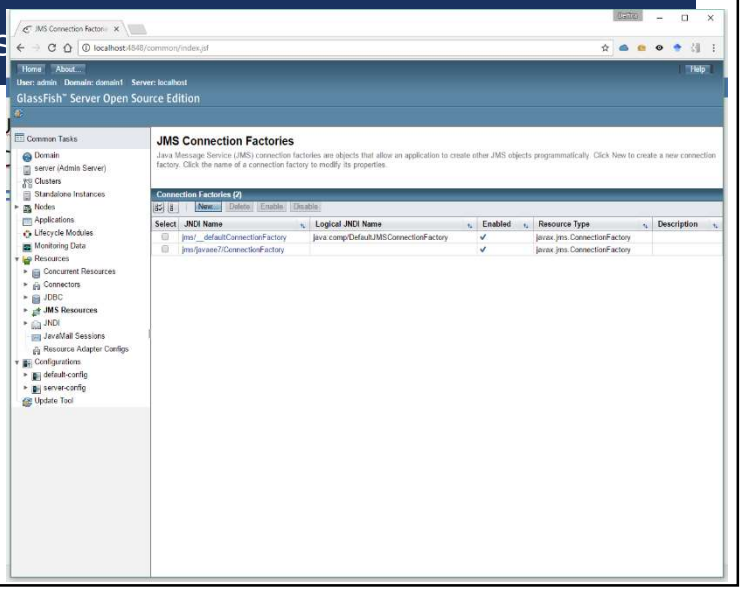
Configurazione via Web Console



Configurazione via Web Console

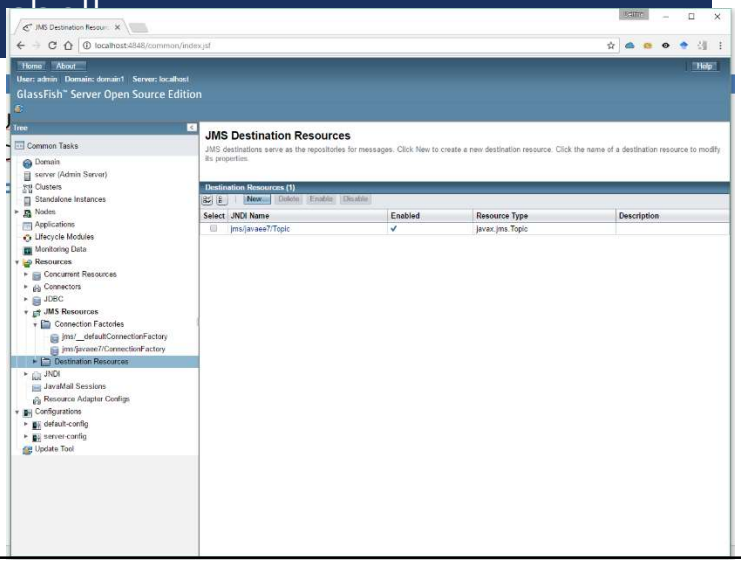
Configurazione via s

- Uso della shell `asadmin`
- Comandi utili:
  - `asadmin create-jms-resource --restype javax.jms.ConnectionFactory jms/javaee7/ConnectionFactory`



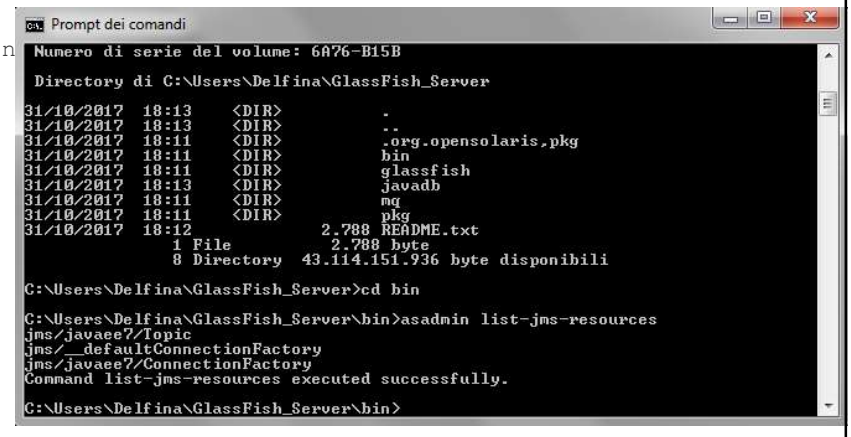
Configurazione via shell

- Uso della shell `asadmin`
- Comandi utili:
  - `asadmin create-jms-resource --restype javax.jms.Topic jms/javaee7/Topic`

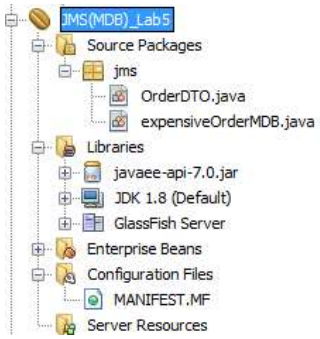


Configurazione via shell

- Uso della shell `asadmin`
- Comandi utili:
  - `asadmin list-jms-resources`

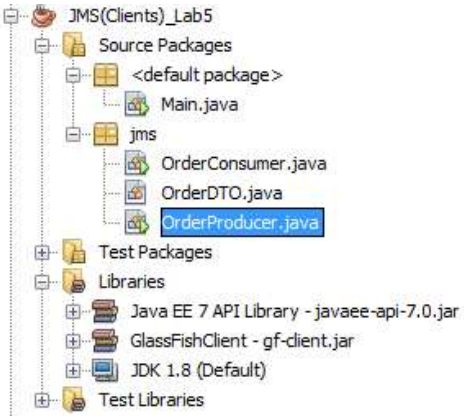


Il Message-Driven Bean



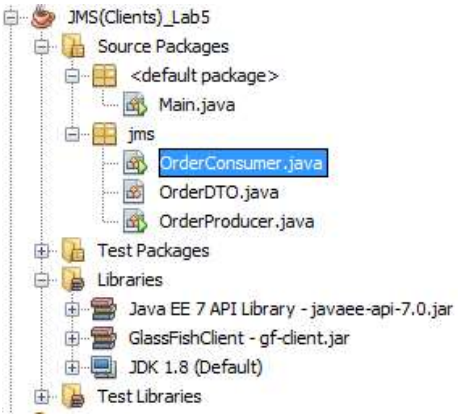
- Progetto per EJB
- Libreria di Java EE 7 aggiunta
- Build & deploy

Il Produttore



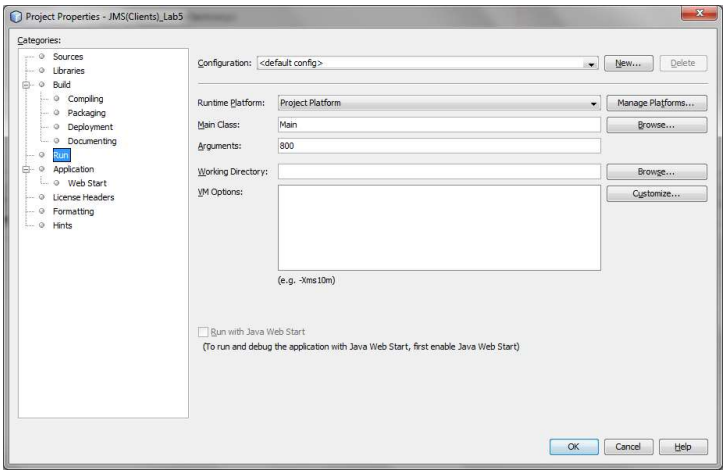
- Progetto standard JSE
- Libreria di Java EE 7 aggiunta
- Libreria di Glassfish client aggiunta
- Build e run
- Parametro su linea di comando da configurazione di lancio (right-click, Properties, Run)

Il Consumatore



- Progetto standard JSE (lo stesso di prima)
- Run

Esecuzione: set dei parametri



Esecuzione: Il C

Start Page | expensiveOrderMDB.java | OrderConsumer.java

Source | History

```
22
23  public static void main(String[] args) throws NamingException {
24
25    // Gets the JNDI context
26    Context jndiContext = new InitialContext();
27
28    // Looks up the administered objects
29    ConnectionFactory connectionFactory = (ConnectionFactory) jndiContext.lookup("jms/javaee7/Connectio
30    Destination topic = (Destination) jndiContext.lookup("jms/javaee7/Topic");
31
32    // Loops to receive the messages
33    System.out.println("\nInfinite loop. Waiting for a message...");
34    try (JMSContext jmsContext = connectionFactory.createContext()) {
35      while (true) {
36        OrderDTO order = jmsContext.createConsumer(topic).receiveBody(OrderDTO.class);
37        System.out.println("Order received: " + order);
38      }
39    }
```

Output | Test Results

Java DB Database Process | GlassFish Server | JMS(Clients)_Lab5 (run)

```
run:
nov 12, 2017 2:38:20 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
nov 12, 2017 2:38:22 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1  (Build 9-b) Compile:  July 29 2014 1229
nov 12, 2017 2:38:22 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
nov 12, 2017 2:38:22 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE

Infinite loop. Waiting for a message...
```

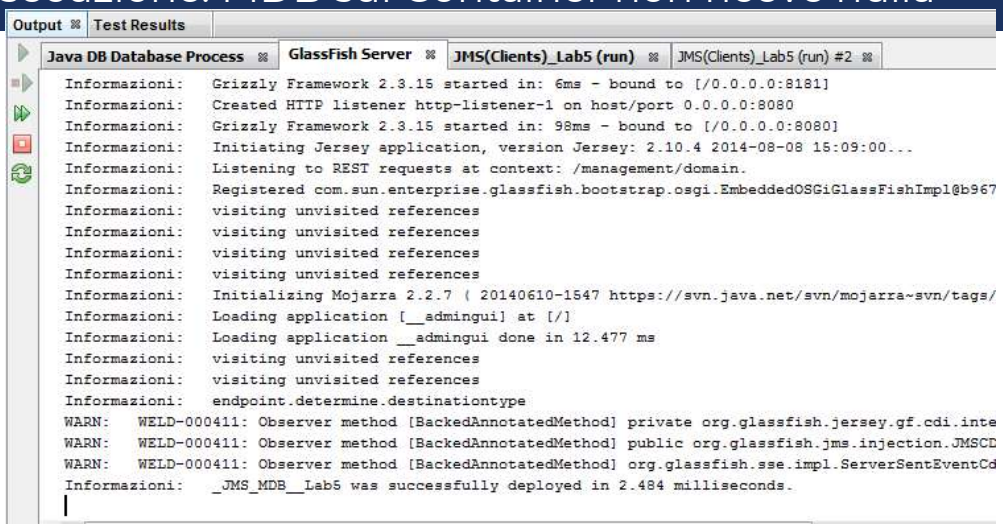
Esecuzione: Il Producer (800)

Output | Test Results

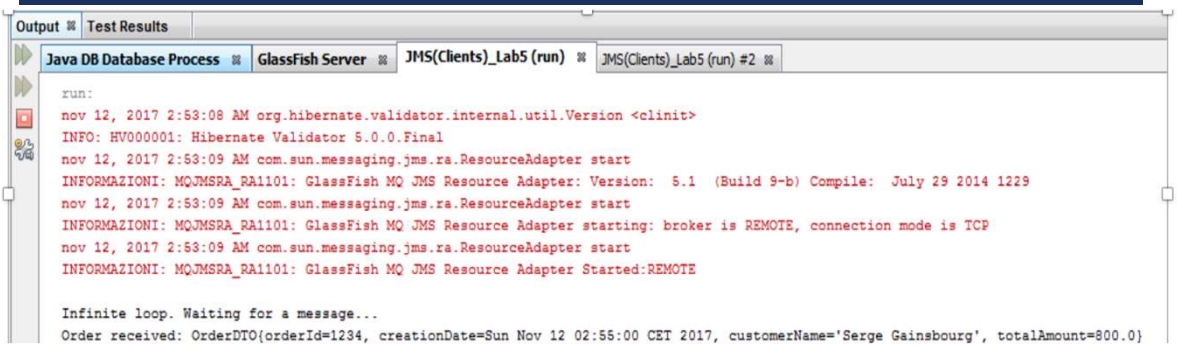
Java DB Database Process | GlassFish Server | JMS(Clients)_Lab5 (run) | JMS(Clients)_Lab5 (run) #2

```
run:
Sending message with amount = 800
nov 12, 2017 2:55:19 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
nov 12, 2017 2:55:20 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1  (Build 9-b) Compile:  July 29 2014 1229
nov 12, 2017 2:55:20 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
nov 12, 2017 2:55:20 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE

Order sent : OrderDTO{orderId=1234, creationDate=Sun Nov 12 02:55:00 CET 2017, customerName='Serge Gainsbourg', totalAmount=800.0}
BUILD SUCCESSFUL (total time: 28 seconds)
```

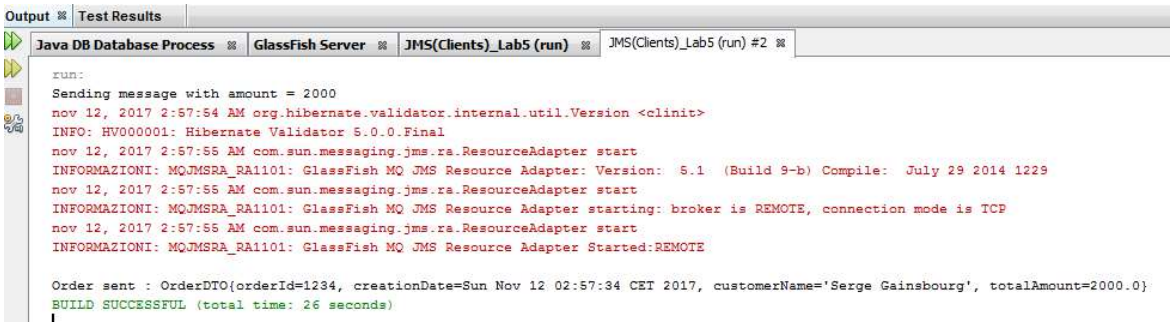
Esecuzione: MDB sul Container non riceve nulla



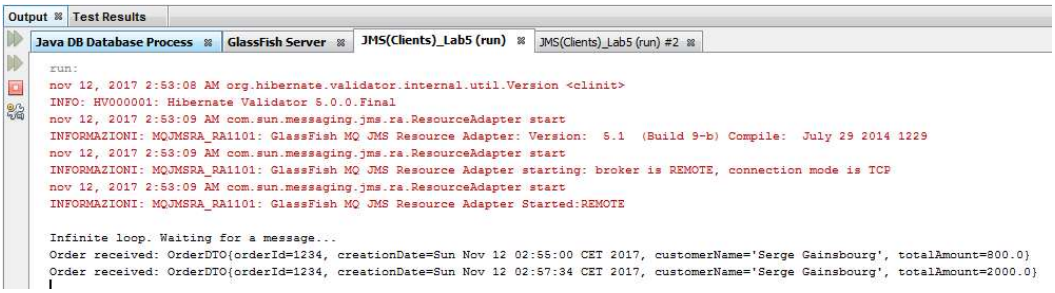
Esecuzione: Il Consumer riceve il messaggio (800)



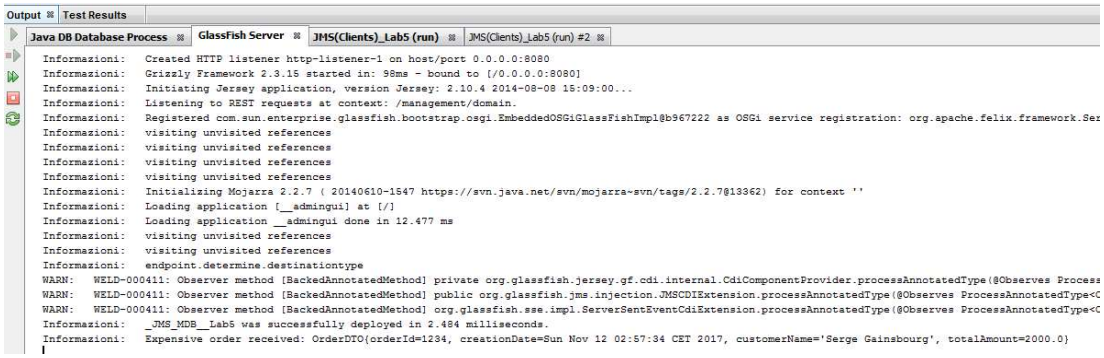
Esecuzione: Il Producer (2000)



Esecuzione: Il Consumer riceve il messaggio (2000)



Esecuzione: l'MDB? ANCHE



Organizzazione della lezione

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Conclusioni

Delfina Malandrino
dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino