

Lezione 22: Java Messaging Services - 2

Corso di *Programmazione Distribuita*

Vittorio Scarano

Università di Salerno



Laurea in Informatica

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

IL PROGRAMMA DELLA LEZIONE

22: Java
Messaging
Services - 2

Meccanismi di affidabilità

Meccanismi di
affidabilità

Message-Driven Beans

Message-Driven
Beans

Un esempio conclusivo

Un esempio
conclusivo

Il codice

Il codice

Configurazione

Configurazione

I progetti

I progetti

Esercizi

Conclusioni

Esercizi

Conclusioni



Meccanismi di affidabilità

Meccanismi di affidabilità

Message-Driven Beans

Message-Driven Beans

Un esempio conclusivo

Un esempio conclusivo

Il codice

Il codice

Configurazione

Configurazione

I progetti

I progetti

Esercizi

Esercizi

Conclusioni

Conclusioni



- ▶ Meccanismo di filtering: ricevere solo i messaggi che si vuole
- ▶ Scegliere il *time-to-live* per evitare messaggi obsoleti
- ▶ Specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
- ▶ Controllo degli ack a vari livelli
- ▶ Durable subscription
- ▶ Priorità di messaggi

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



- ▶ Si fa in modo che i messaggi che arrivino siano solamente quelli a cui si è interessati
- ▶ Nessuno spreco di banda per ricevere cose non interessanti
- ▶ Si può fare selezione su headers (metadata fissati dallo standard) e sulle proprietà
- ▶ Il consumer può scegliere usando sintassi SQL come

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

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

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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



- ▶ 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);
```

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



- ▶ Diverse qualità del servizio possono essere richieste al broker
 - ▶ con conseguente aggravio di carico, per i requisiti più stringenti
- ▶ Persistent delivery: recapito *exactly-once* del messaggio viene garantito
- ▶ Non-persistent delivery: recapito *at most once*
- ▶ Persistent delivery è il valore di default
 - ▶ che può essere “degradato” per migliorare le prestazioni

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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

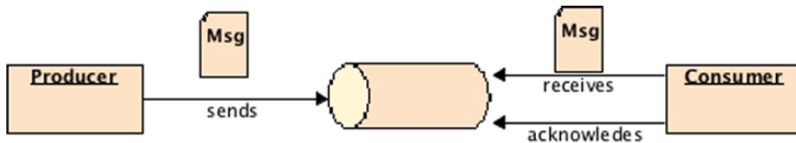
Esercizi

Conclusioni



- ▶ Si vuole ricevere una verifica del recapito del messaggio al destinatario
- ▶ Diverse modalità di acknowledgment
 - ▶ AUTO_ACKNOWLEDGE: automatico dalla sessione
 - ▶ CLIENT_ACKNOWLEDGE: automatico dalla sessione
 - ▶ AUTO_ACKNOWLEDGE: esplicito da parte del client `Message.acknowledge()`
 - ▶ DUPS_OK_ACKNOWLEDGE: possibili messaggi di ack duplicati
- ▶ Come si usa

```
@Inject  
@JMSConnectionFactory("jms/connectionFactory")  
@JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)  
private JMSContext context;
```



Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



- ▶ 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
- ▶ Con i durable subscribers, 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();
```



- ▶ 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);
```

**Meccanismi di
affidabilità**

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



Meccanismi di affidabilità

Meccanismi di
affidabilità

Message-Driven Beans

**Message-Driven
Beans**

Un esempio conclusivo

Un esempio
conclusivo

Il codice

Il codice

Configurazione

Configurazione

I progetti

I progetti

Esercizi

Esercizi

Conclusioni

Conclusioni



- ▶ 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.
- ▶ Il vantaggio di usare un MDB 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

Implementa un
MessageListener invece di
interfacce remote: non
serve specificare che
metodi vengono offerti

Cosa fare quando si riceve
un messaggio

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



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

Meccanismi di
affidabilità

**Message-Driven
Beans**

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



► 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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

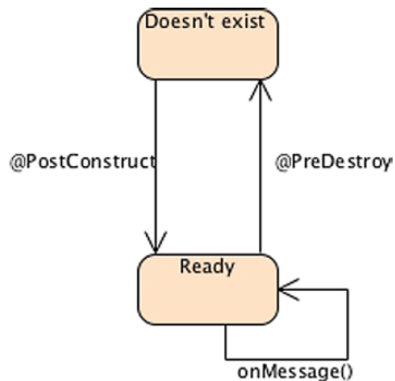
Esercizi

Conclusioni



Method	Description
getCallerPrincipal	Returns the <code>java.security.Principal</code> associated with the invocation
getRollbackOnly	Tests whether the current transaction has been marked for rollback
getTimerService	Returns the <code>javax.ejb.TimerService</code> interface
getUserTransaction	Returns the <code>javax.transaction.UserTransaction</code> 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





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

POSSIBILE RICEVERE E INVIARE 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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

- ▶ In quanto EJB, le transazioni possono essere Bean-managed oppure Container-managed
- ▶ Transazioni per scambio di messaggi: un certo numero di messaggi vanno recapitati tutti insieme o nessuno
- ▶ Eccezioni specifiche: `JMSEException` (checked) e `JMSRuntimeException` unchecked
- ▶ Eccezione `JMSEException` tipicamente usata per fare rollback di transazioni (esplicitamente da parte del programmatore)



PLAN

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

**Un esempio
conclusivo**

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

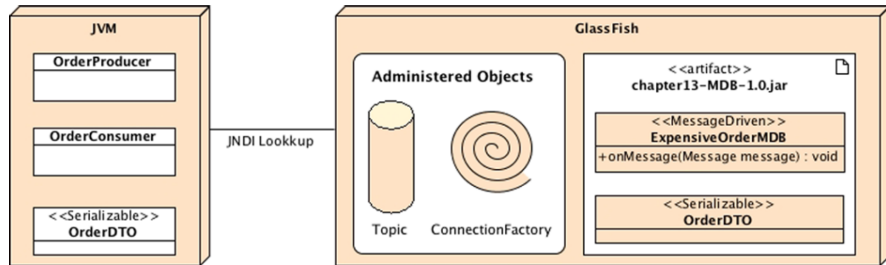
I progetti

Esercizi

Conclusioni



- ▶ Producer stand-alone 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 il MDB ExpensiveOrderMDB riceve solamente quelli sopra i 1000



Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

```
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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni




```
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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

```
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

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

```
@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 (JMSEException 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

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

PLAN

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



- ▶ 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)

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



CONFIGURAZIONE VIA WEB CONSOLE - 01

22: Java
Messaging
Services - 2

The screenshot shows the GlassFish Server Open Source Edition web console. The top navigation bar includes 'Home', 'About...', and 'Help' buttons. Below the navigation bar, the user information is displayed: 'User: admin', 'Domain: domain1', and 'Server: localhost'. The main title is 'GlassFish™ Server Open Source Edition'. On the left side, there is a 'Common Tasks' sidebar with a tree view showing the domain structure: Domain, server (Admin Server), Clusters, Standalone Instances, Nodes, Applications, Lifecycle Modules, Monitoring Data, Resources (with sub-items: Concurrent Resources, Connectors, JDBC, JMS Resources, and Connection Factories), JNDI, JavaMail Sessions, Resource Adapter Configs, Configurations (with sub-items: default-config, server-config), and Update Tool. The 'Connection Factories' item under 'JMS Resources' is selected. The main content area is titled 'New JMS Connection Factory' and includes a description: 'The creation of a new Java Message Service (JMS) connection factory also creates a connector connection pool for the factory and a connector resource.' Below the title, there are 'OK' and 'Cancel' buttons. The configuration is divided into two sections: 'General Settings' and 'Pool Settings'. In the 'General Settings' section, the 'JNDI Name' is a required text field, 'Resource Type' is set to 'javax.jms.TopicConnectionFactory', 'Description' is a text field, and 'Status' is checked as 'Enabled'. The 'Pool Settings' section includes: 'Initial and Minimum Pool Size' set to 1, 'Maximum Pool Size' set to 250, 'Pool Resize Quantity' set to 2, 'Idle Timeout' set to 300 seconds, 'Max Wait Time' set to 60000 milliseconds, 'On Any Failure' set to 'Close All Connections', 'Transaction Support' set to a dropdown menu, and 'Connection Validation' set to 'Required'. Each setting has a brief description of its function.

New JMS Connection Factory OK Cancel

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

General Settings

JNDI Name: *

Resource Type: javax.jms.TopicConnectionFactory

Description:

Status: ☒ Enabled

Pool Settings

Initial and Minimum Pool Size: 1 Connections
Minimum and initial number of connections maintained in the pool

Maximum Pool Size: 250 Connections
Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: 2 Connections
Number of connections to be removed when pool idle timeout expires

Idle Timeout: 300 Seconds
Maximum time that connection can remain idle in the pool

Max Wait Time: 60000 Milliseconds
Amount of time caller waits before connection timeout is sent

On Any Failure: ☐ Close All Connections
Close all connections and reconnect on failure, otherwise reconnect only when used

Transaction Support:
Level of transaction support. Overwrite the transaction support attribute in the Resource Adapter in a downward compatible way.

Connection Validation: ☐ Required
Validate connections, allow server to reconnect in case of failure

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

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: *	<input type="text" value="jms/javaee7/ConnectionFactory"/>
Resource Type:	<input type="text" value="javax.jms.ConnectionFactory"/>
Description:	<input type="text" value="ConnectionFactory esempio di prova"/>
Status:	<input checked="" type="checkbox"/> Enabled



Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

The screenshot shows the GlassFish Server Open Source Edition web console. The top navigation bar includes 'Home', 'About...', and 'Help' buttons. Below the navigation bar, the user information is displayed: 'User: admin', 'Domain: domain1', and 'Server: localhost'. The main title is 'GlassFish™ Server Open Source Edition'.

The left sidebar shows a tree view of the configuration hierarchy. The 'Resources' folder is expanded, showing 'Concurrent Resources', 'Connectors', 'JDBC', 'JMS Resources', and 'Destination Resources'. The 'Destination Resources' folder is selected.

The main content area is titled 'New JMS Destination Resource'. Below the title, a note states: 'The creation of a new Java Message Service (JMS) destination resource also creates an admin object resource.' There are 'OK' and 'Cancel' buttons at the top right of this section.

The configuration fields are as follows:

- JNDI Name:** A text input field.
- Physical Destination Name:** A text input field. Below it, a note states: 'Destination name in the Message Queue broker. If the destination does not exist, it will be created automatically when needed.'
- Resource Type:** A dropdown menu with 'javax.jms.Topic' selected.
- Description:** A text input field.
- Status:** A checkbox labeled 'Enabled' which is checked.

Below the configuration fields, there is a section titled 'Additional Properties (0)'. It includes 'Add Property' and 'Delete Properties' buttons. Below these buttons is a table with the following structure:

Select	Name	Value	Description
No items found.			

At the bottom right of the main content area, there are 'OK' and 'Cancel' buttons.

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

[Il codice](#)

[Configurazione](#)

[I progetti](#)

Esercizi

Conclusioni

New JMS Destination Resource

The creation of a new Java Message Service (JMS) destination resource also creates an admin object resource.

JNDI Name: *

Physical Destination Name *

Destination name in the Message Queue broker. If the destination does not exist, it will be created automatically.

Resource Type: *

Description:

Status:

☒ Enabled

Additional Properties (0)

[Add Property](#)[Delete Properties](#)

Select	Name	Value	Description
No items found.			



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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

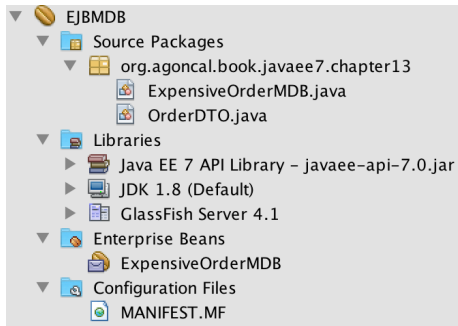
Configurazione

I progetti

Esercizi

Conclusioni





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

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

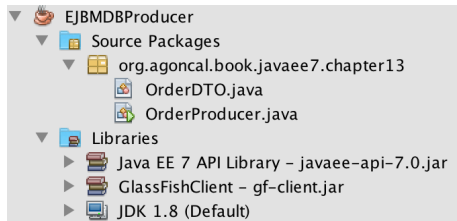
Configurazione

I progetti

Esercizi

Conclusioni





- ▶ 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)

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

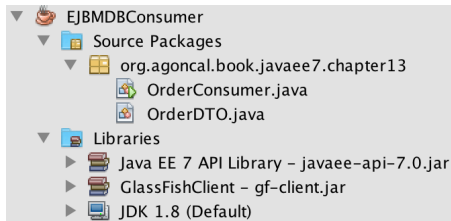
Configurazione

I progetti

Esercizi

Conclusioni





- ▶ Progetto standard JSE
- ▶ Libreria di Java EE 7 aggiunta
- ▶ Libreria di Glassfish client aggiunta
- ▶ Build e run

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

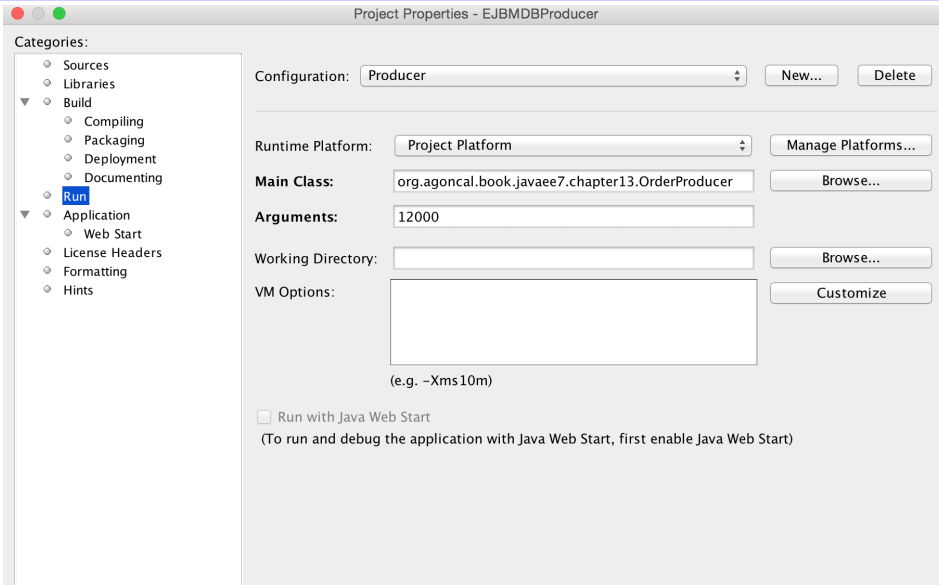
I progetti

Esercizi

Conclusioni



ESECUZIONE - 00: SET PARAMETRI



Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

ESECUZIONE - 01: IL PRODUTTORE (12000)

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

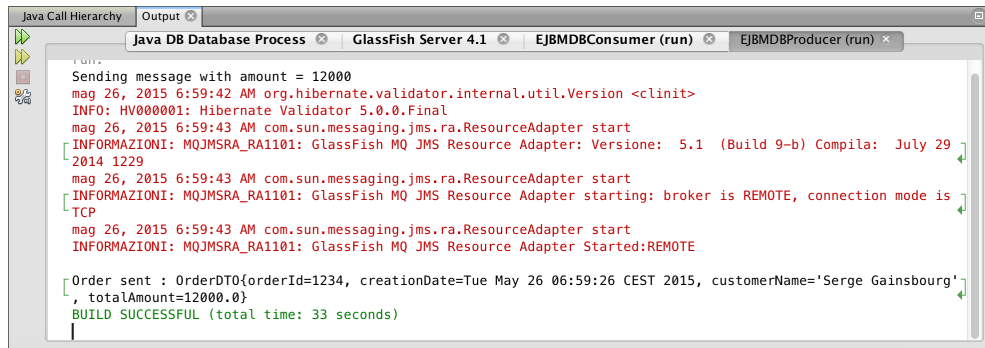
Il codice

Configurazione

I progetti

Esercizi

Conclusioni



```
run.  
Sending message with amount = 12000  
mag 26, 2015 6:59:42 AM org.hibernate.validator.internal.util.Version <clinit>  
INFO: HV000001: Hibernate Validator 5.0.0.Final  
mag 26, 2015 6:59:43 AM com.sun.messaging.jms.ra.ResourceAdapter start  
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Versione: 5.1 (Build 9-b) Compila: July 29  
2014 1229  
mag 26, 2015 6:59:43 AM com.sun.messaging.jms.ra.ResourceAdapter start  
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is  
TCP  
mag 26, 2015 6:59:43 AM com.sun.messaging.jms.ra.ResourceAdapter start  
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE  
  
[Order sent : OrderDTO{orderId=1234, creationDate=Tue May 26 06:59:26 CEST 2015, customerName='Serge Gainsbourg'  
, totalAmount=12000.0}  
BUILD SUCCESSFUL (total time: 33 seconds)
```



Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

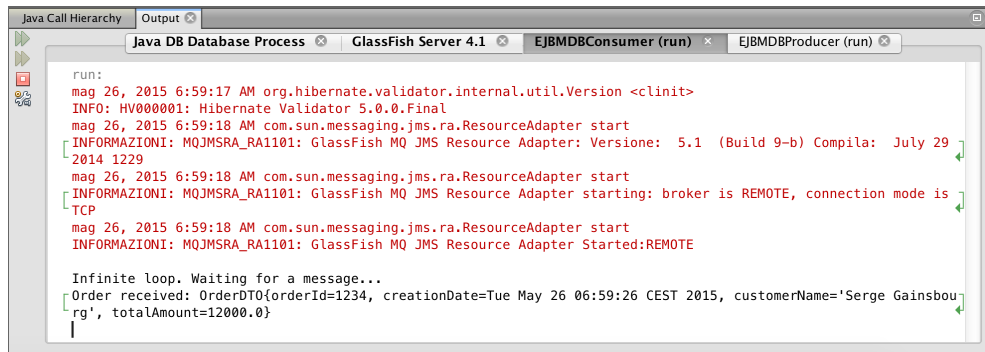
Il codice

Configurazione

I progetti

Esercizi

Conclusioni



```
run:
mag 26, 2015 6:59:17 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
mag 26, 2015 6:59:18 AM com.sun.messaging.jms.ra.ResourceAdapter start
[ INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Versione: 5.1 (Build 9-b) Compila: July 29 2014 1229
mag 26, 2015 6:59:18 AM com.sun.messaging.jms.ra.ResourceAdapter start
[ INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
mag 26, 2015 6:59:18 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE

Infinite loop. Waiting for a message...
[ Order received: OrderDTO{orderId=1234, creationDate=Tue May 26 06:59:26 CEST 2015, customerName='Serge Gainsbourg', totalAmount=12000.0}
|
```

ESECUZIONE - 03: IL MDB SU CONTAINER

22: Java
Messaging
Services - 2

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

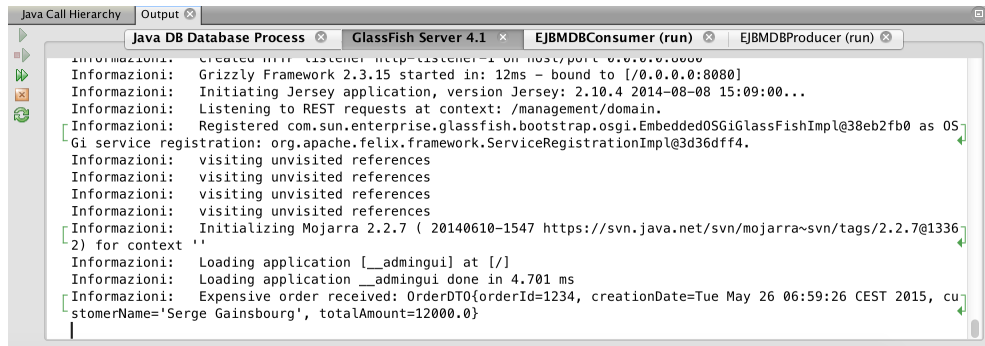
Il codice

Configurazione

I progetti

Esercizi

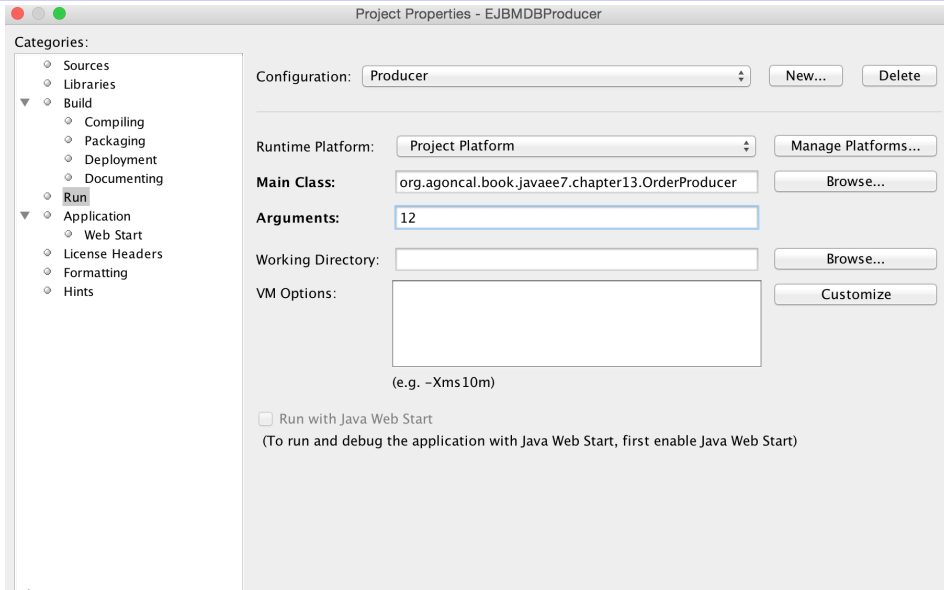
Conclusioni



```
Java DB Database Process x GlassFish Server 4.1 x EJBMDBConsumer (run) x EJBMDBProducer (run) x
Informazioni: Created http-listener http-listener-1 on host/port 0.0.0.0:8080
Informazioni: Grizzly Framework 2.3.15 started in: 12ms - bound to [/0.0.0.0:8080]
Informazioni: Initiating Jersey application, version Jersey: 2.10.4 2014-08-08 15:09:00...
Informazioni: Listening to REST requests at context: /management/domain.
Informazioni: Registered com.sun.enterprise.glassfish.bootstrap.osgi.EmbeddedOSGiGlassFishImpl@38eb2fb0 as OS
Gi service registration: org.apache.felix.framework.ServiceRegistrationImpl@3d36dff4.
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: Initializing Mojarra 2.2.7 ( 20140610-1547 https://svn.java.net/svn/mojarra~svn/tags/2.2.7@1336
2) for context ''
Informazioni: Loading application [__admingui] at [/]
Informazioni: Loading application __admingui done in 4.701 ms
Informazioni: Expensive order received: OrderDTO{orderId=1234, creationDate=Tue May 26 06:59:26 CEST 2015, cu
stomerName='Serge Gainsbourg', totalAmount=12000.0}
```



ESECUZIONE - 04: CAMBIO PARAMETRI A 12



Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice
Configurazione
I progetti

Esercizi

Conclusioni

ESECUZIONE - 06: INVIO CON AMOUNT=12

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

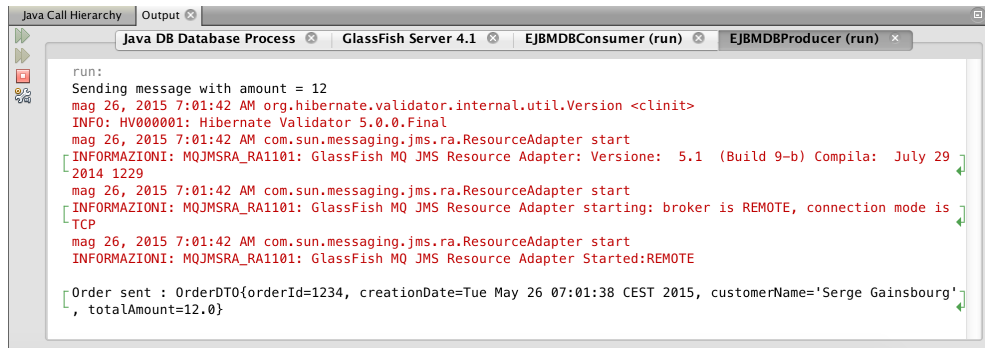
Il codice

Configurazione

I progetti

Esercizi

Conclusioni



```
run:
Sending message with amount = 12
mag 26, 2015 7:01:42 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
mag 26, 2015 7:01:42 AM com.sun.messaging.jms.ra.ResourceAdapter start
[ INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Versione: 5.1 (Build 9-b) Compila: July 29
2014 1229 ]
mag 26, 2015 7:01:42 AM com.sun.messaging.jms.ra.ResourceAdapter start
[ INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is
TCP ]
mag 26, 2015 7:01:42 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE

[ Order sent : OrderDTO{orderId=1234, creationDate=Tue May 26 07:01:38 CEST 2015, customerName='Serge Gainsbourg',
totalAmount=12.0} ]
```

ESECUZIONE - 07: IL CONSUMER INVECE SI

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

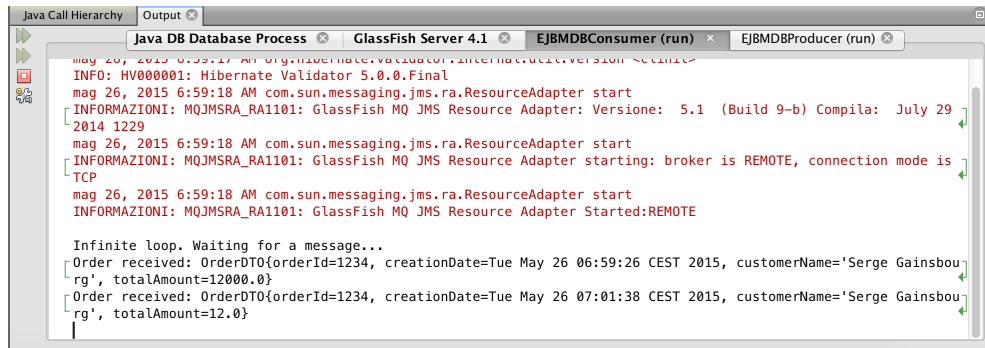
Il codice

Configurazione

I progetti

Esercizi

Conclusioni



```
Java Call Hierarchy Output
Java DB Database Process GlassFish Server 4.1 EJBMDBConsumer (run) EJBMDBProducer (run)

mag 26, 2015 6:59:17 AM org.hibernate.validator.internal.util.Version<...>
INFO: HV000001: Hibernate Validator 5.0.0.Final
mag 26, 2015 6:59:18 AM com.sun.messaging.jms.ra.ResourceAdapter start
[INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Versione: 5.1 (Build 9-b) Compila: July 29 2014 1229]
mag 26, 2015 6:59:18 AM com.sun.messaging.jms.ra.ResourceAdapter start
[INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP]
mag 26, 2015 6:59:18 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMAZIONI: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE

Infinite loop. Waiting for a message...
[Order received: OrderDTO{orderId=1234, creationDate=Tue May 26 06:59:26 CEST 2015, customerName='Serge Gainsbourg', totalAmount=12000.0}]
[Order received: OrderDTO{orderId=1234, creationDate=Tue May 26 07:01:38 CEST 2015, customerName='Serge Gainsbourg', totalAmount=12.0}]
```

ESECUZIONE - 08: IL MDB NON RICEVE NIENTE

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

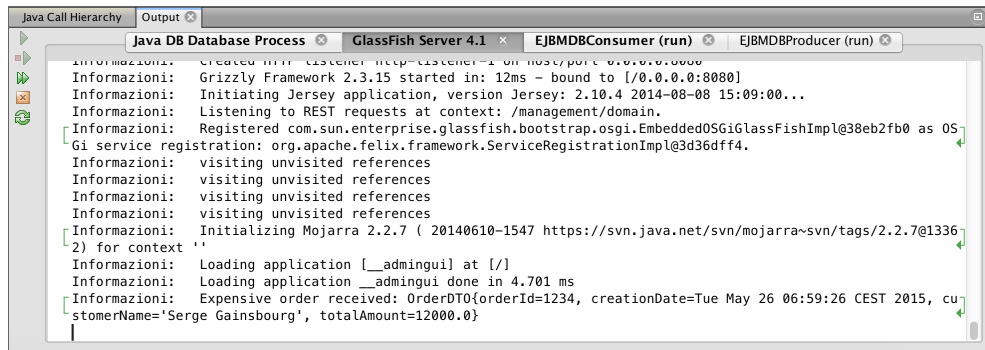
Il codice

Configurazione

I progetti

Esercizi

Conclusioni



```
Java DB Database Process x GlassFish Server 4.1 x EJBMDBConsumer (run) x EJBMDBProducer (run) x
Informazioni: created http-listener http-listener-1 on host/port 0.0.0.0:8080
Informazioni: Grizzly Framework 2.3.15 started in: 12ms - bound to [/0.0.0.0:8080]
Informazioni: Initiating Jersey application, version Jersey: 2.10.4 2014-08-08 15:09:00...
Informazioni: Listening to REST requests at context: /management/domain.
Informazioni: Registered com.sun.enterprise.glassfish.bootstrap.osgi.EmbeddedOSGiGlassFishImpl@38eb2fb0 as OS
Gi service registration: org.apache.felix.framework.ServiceRegistrationImpl@3d36dff4.
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: Initializing Mojarra 2.2.7 ( 20140610-1547 https://svn.java.net/svn/mojarra~svn/tags/2.2.7@1336
2) for context ''
Informazioni: Loading application [_admingui] at [/]
Informazioni: Loading application __admingui done in 4.701 ms
Informazioni: Expensive order received: OrderDTO{orderId=1234, creationDate=Tue May 26 06:59:26 CEST 2015, cu
stomerName='Serge Gainsbourg', totalAmount=12000.0}
```



PLAN

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



- ▶ Scrivere un insieme di EJB e client che rappresentino un archivio di libri (con titolo, id, autore, prezzo)
- ▶ La organizzazione deve essere fatta in distinti progetti NetBeans
- ▶ Devono essere previste delle query per id (chiave primaria) e una query che restituisce tutti i libri
- ▶ Deve esserci un client basato su messaggi, che invia un messaggio con la id e il prezzo nuovo, che prevede l'aggiornamento del prezzo
- ▶ Deve esserci un client basato su invocazione di un bean stateless, che prevede la stampa della lista dei libri
- ▶ Deve essere previsto un bean singleton che inizializzi l'archivio

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



- ▶ Poco codice da scrivere, ma si richiede precisione
- ▶ Importante la struttura dei file e dei progetti (sviluppati su carta)
- ▶ Non viene richiesta la scrittura di XML (si assume verrà fatto nel progetto consegnato successivamente)
- ▶ Si richiede l'uso congiunto di EJB Stateless/stateful, di JPA, di JMS, di Web Services (da fare)

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



PLAN

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni



QUELLO CHE ABBIAMO FATTO...

Meccanismi di affidabilità

Message-Driven Beans

Un esempio conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

Meccanismi di
affidabilità

Message-Driven
Beans

Un esempio
conclusivo

Il codice

Configurazione

I progetti

Esercizi

Conclusioni

