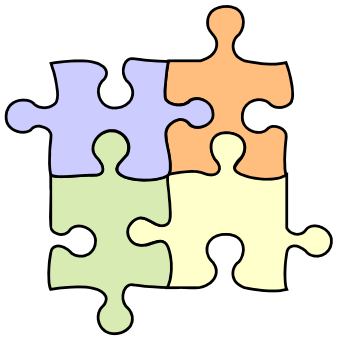
Luca Cabibbo



Architetture Software

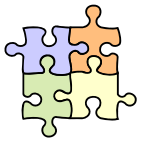
Componenti (middleware)

Dispensa ASW 850

ottobre 2014

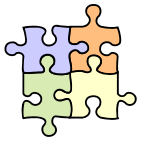
Le interfacce dei componenti devono essere progettate in modo che ciascun componente possa essere implementato in modo indipendente dalla specifica implementazione degli altri componenti con cui deve interagire.

Mark Maier



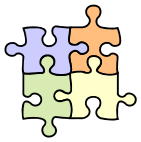
- Fonti

- Java Platform, Enterprise Edition
  http://www.oracle.com/technetwork/java/javaee/
- The Java EE 7 Tutorial
  - *http://docs.oracle.com/javaee/7/tutorial/doc/*
  - Chapter 32, Enterprise Beans
  - Chapter 33, Getting Started with Enterprise Beans
  - Chapter 34, Running the Enterprise Beans Examples
  - Chapter 36, Using Asynchronous Method Invocation in Session Beans
  - Chapter 46, Java Message Service Examples
- [POSA4] Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing

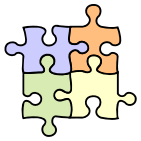


- Obiettivi e argomenti

- Obiettivi
  - introdurre le tecnologie a componenti
  - come studio di caso, introdurre la piattaforma Java EE – e descrivere la tecnologia a componenti EJB e il supporto all'esecuzione di componenti EJB
  - discutere alcuni aspetti metodologici sull'uso della piattaforma Java EE
- Argomenti
  - introduzione ai componenti
  - la piattaforma Java EE
  - enterprise bean
  - session bean
  - operazioni asincrone
  - message-driven bean
  - discussione

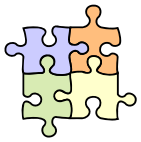


- Wordle



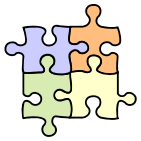
* Introduzione ai componenti

- Le tecnologie a oggetti distribuiti nascono come confluenza delle tecnologie RPC e OO
  - alcuni limiti della tecnologia a oggetti distribuiti
    - mancanza di meccanismi per separare le dipendenze tra gli oggetti dalle loro implementazioni – se un oggetto deve fruire di un servizio fornito da un altro oggetto, deve scoprire e connettersi a questo oggetto in modo esplicito
    - mancanza di strumenti di deployment e configurazione – il rilascio di un oggetto remoto su un certo nodo richiede la scrittura e l'esecuzione di uno script ad-hoc per creare, configurare e registrare l'oggetto
    - supporto limitato per proprietà di qualità
- Le tecnologie a componenti nascono, a metà degli anni '90, anche per superare i limiti delle tecnologie a oggetti distribuiti



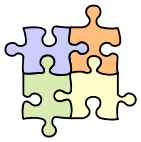
Componenti e contenitori

- Le tecnologie a componenti sostengono lo sviluppo e la gestione di componenti software (componenti) che possono essere rilasciati, composti ed eseguiti entro ambienti di esecuzione chiamati contenitori
  - un *componente* è un'entità software runtime
    - implementa un insieme di funzionalità
    - offre i suoi servizi mediante un insieme di interfacce con nome (interfacce fornite)
    - può richiedere servizi ad altri componenti, sempre sulla base di interfacce con nome (interfacce richieste)
  - in pratica, un'applicazione è formata da un insieme di componenti, che vengono composti (al momento del rilascio) sulla base di un'interconnessione delle loro interfacce



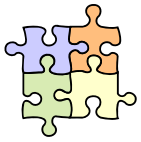
Componenti e contenitori

- Le tecnologie a componenti sostengono lo sviluppo e la gestione di componenti software (componenti) che possono essere rilasciati, composti ed eseguiti entro ambienti di esecuzione chiamati contenitori
  - un *contenitore* è un ambiente runtime, lato server, per l'esecuzione di componenti
    - i componenti, per essere eseguiti, devono essere configurati e rilasciati in un contenitore
  - il contenitore gestisce il ciclo di vita dei componenti in esso rilasciati
    - fornisce ai suoi componenti servizi come sicurezza, transazioni e persistenza – per sostenere qualità come sicurezza, affidabilità, prestazioni e scalabilità
    - grazie a questo, lo sviluppo dei componenti può essere focalizzato sull'implementazione di funzionalità

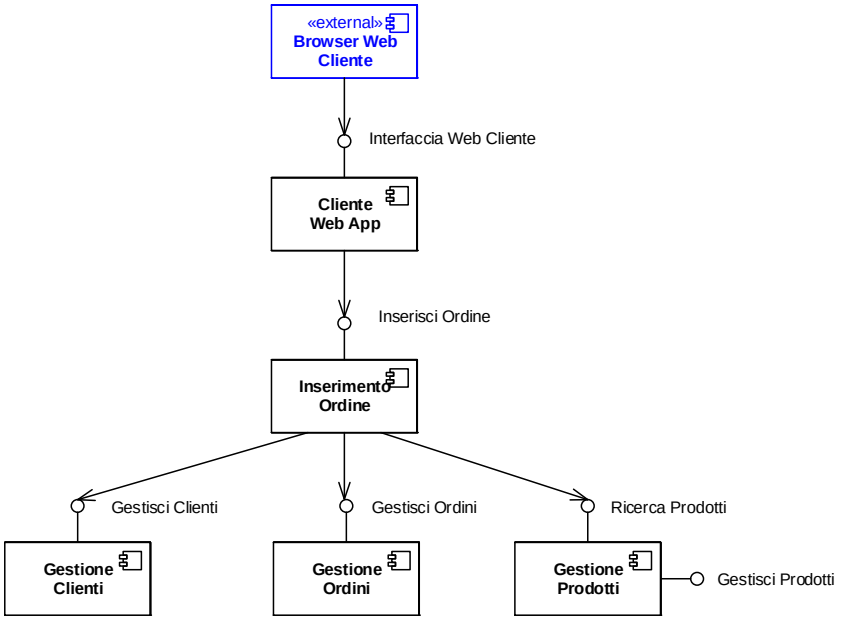


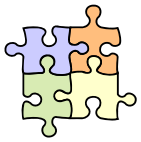
Componenti, interfacce e composizione

- I componenti sono dotati di due tipi di interfacce
  - *interfacce fornite*
    - specifica dei servizi forniti da un componente ad altri componenti
  - *interfacce richieste*
    - specifica dei servizi che devono essere resi disponibili a un componente, affinché possa funzionare come specificato
    - se non sono disponibili, il componente non funzionerà
- La composizione di un'applicazione può avvenire, in sede di deployment
  - sulla base di un certo numero di componenti
  - connettendo le interfacce richieste dei componenti con interfacce fornite da altri componenti



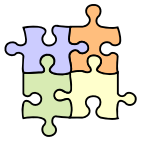
Componenti, interfacce e composizione





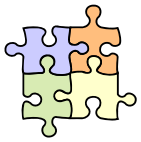
* La piattaforma Java EE

- La piattaforma *Java Enterprise Edition* (*Java EE*)
  - *il problema* – oggi c'è il bisogno di sviluppare applicazioni distribuite, transazionali e portabili, che sfruttino velocità, sicurezza e affidabilità di tecnologie lato server – le applicazioni di tipo enterprise forniscono la logica di business di un'organizzazione – queste applicazioni sono gestite centralmente, e spesso devono interagire con altre applicazioni di tipo enterprise – queste applicazioni devono essere progettate, costruite e prodotte con meno soldi, tempo e risorse
  - *la soluzione Java EE* – con Java EE lo sviluppo di applicazioni enterprise con Java non è mai stato così facile o veloce – intento di Java EE è fornire agli sviluppatori un insieme potente di API, per ridurre il tempo di sviluppo, ridurre la complessità delle applicazioni, e migliorare le prestazioni delle applicazioni
  - Java EE introduce un modello di programmazione semplificato ...



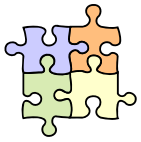
Java EE

- *Java Enterprise Edition* (*Java EE*)
  - modello/piattaforma per la progettazione, lo sviluppo, l'assemblaggio e il deployment di applicazioni distribuite/enterprise
  - supporto per
    - applicazioni distribuite multi-livello
    - componenti riusabili
    - gestione unificata della sicurezza
    - scalabilità e alta disponibilità
    - controllo flessibile delle transazioni
    - interazione sincrona/asincrona
    - web service
    - gestione di dati persistenti
    - ...

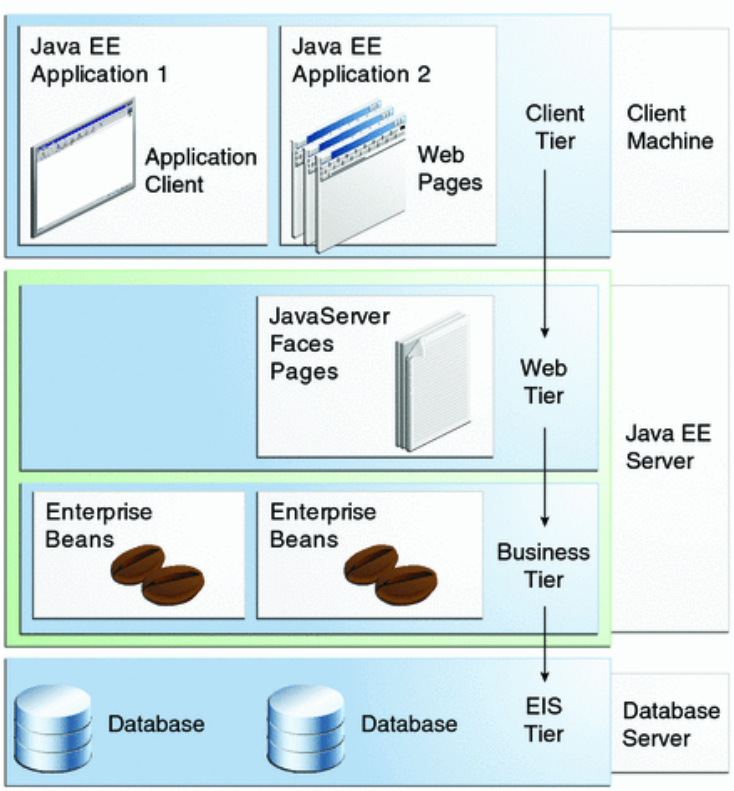


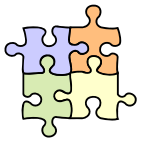
Java EE

- Alcuni obiettivi di Java EE
  - modello di programmazione semplificato
    - ma richiede una comprensione degli aspetti architetturali e di altri aspetti che la piattaforma rende trasparenti
    - è utile ad esempio ricordare che anche Java RMI è basato su un modello di programmazione semplificato, sulla base di una sintassi già familiare al programmatore poco esperto – tuttavia, la semantica dell'invocazione di metodi remoti non è affatto familiare al programmatore poco esperto
  - supporto per alcune qualità architetturalmente significative – come portabilità, disponibilità e scalabilità, sicurezza, ...
    - ma come sono ottenute?
  - supporto per l'interoperabilità e l'integrazione di sistemi esistenti



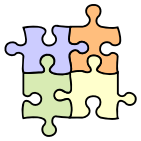
Modello applicativo di Java EE





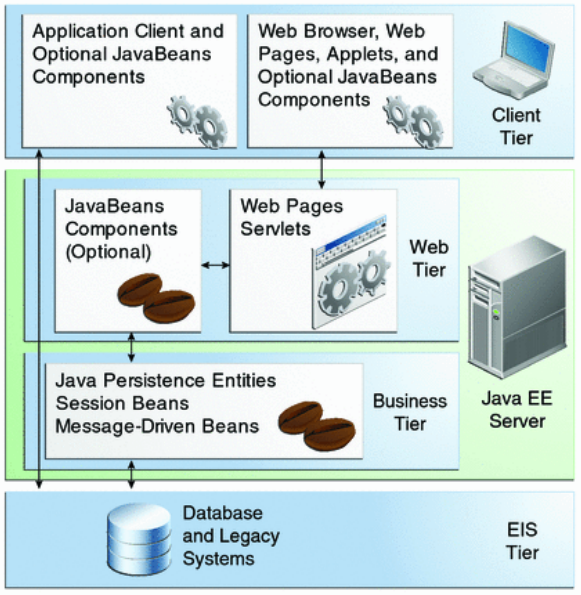
Modello applicativo di Java EE

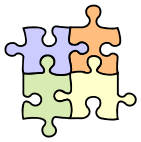
- Modello applicativo di Java EE
  - Java EE fornisce un modello per lo sviluppo di applicazioni distribuite multi-livello
  - client tier, middle tier e back-end tier
    - client tier – supporto per diversi tipi di client
    - web tier – componenti web in esecuzione sul server Java EE
    - business tier – componenti di logica applicativa in esecuzione sul server Java EE
    - back-end tier – EIS tier – accessibile tramite API standard



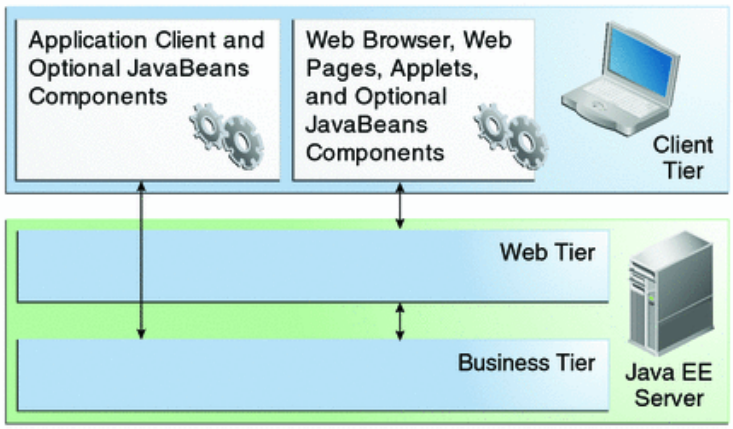
Java EE: scenari applicativi

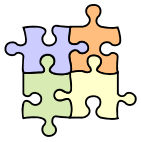
- In effetti, sono possibili numerosi scenari applicativi, a due o più livelli
  - i diversi livelli sono infatti opzionali – tranne il livello client nonché almeno un livello server



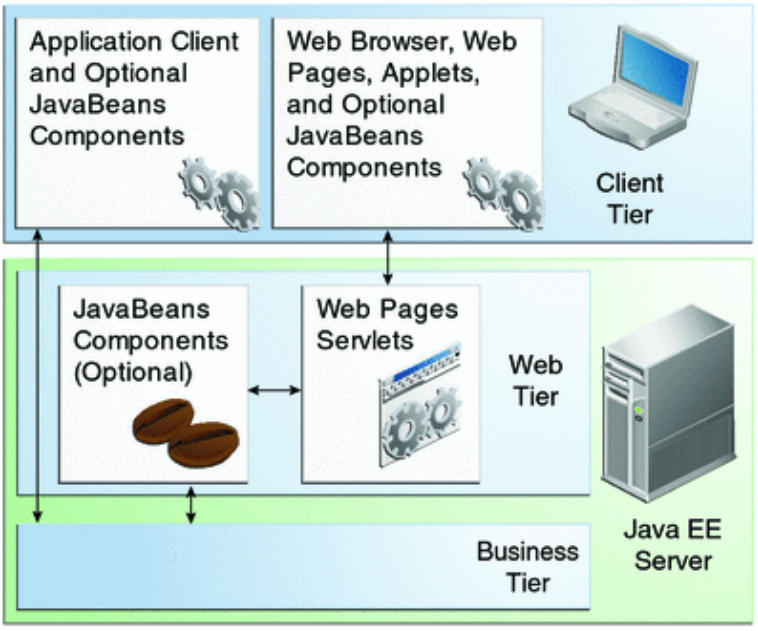


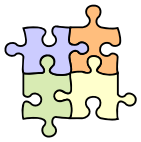
Livello client e comunicazione col server



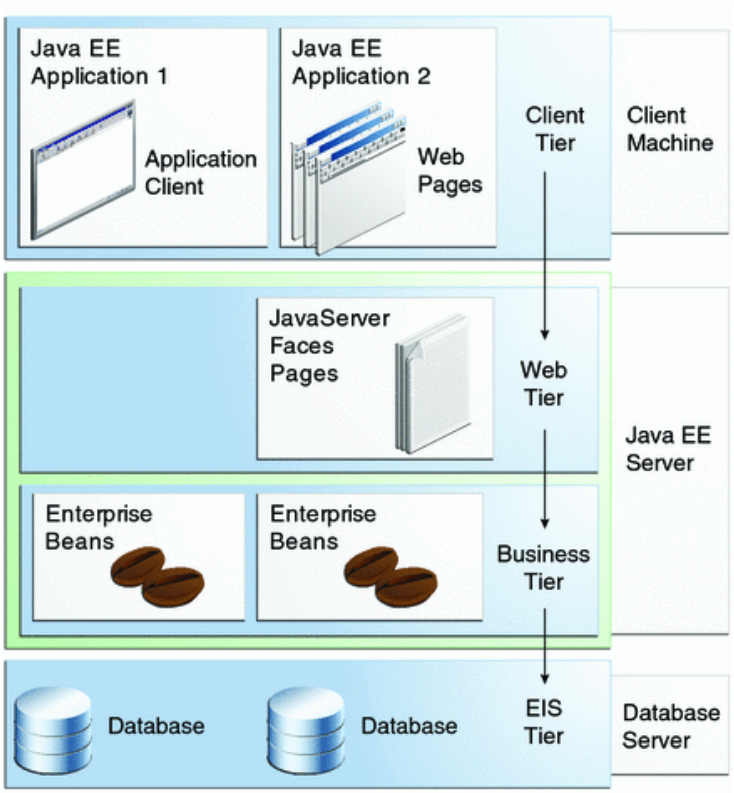


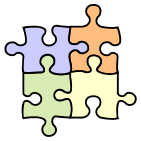
Livello web





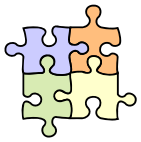
Livello business e EIS





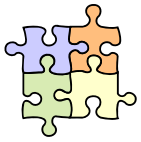
Componenti Java EE

- Le applicazioni Java EE sono fatte di *componenti* – Java EE definisce diverse tipologie di componenti
  - client applicativi e applet – componenti di tipo client
  - componenti web – Java Servlet, JSP, JSF – gestiti dal server Java EE (web container)
  - Enterprise Java Bean (EJB) o semplicemente Enterprise Bean – gestiti dal server Java EE (EJB container)
- In generale, i componenti Java EE sono essenzialmente dei "programmi" Java – il cui ciclo di vita (a runtime) è però diverso da quello dei programmi Java più semplici
  - è diversa anche la fase di costruzione/rilascio/esecuzione, basata sulle seguenti attività: (i) compilazione, (ii) assemblaggio, (iii) deployment e (iv) esecuzione e gestione a cura di un application server Java EE

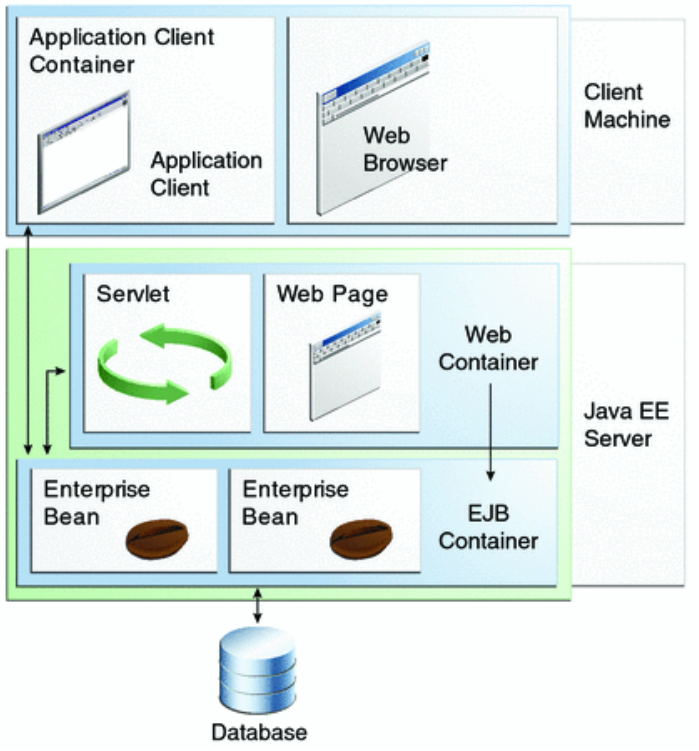


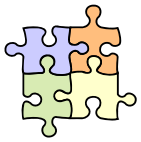
Contenitori Java EE

- I contenitori sono responsabili della gestione dei componenti
  - un *contenitore* (*container*) è un ambiente runtime standardizzato per l'esecuzione di componenti
  - i componenti, per poter erogare le loro funzionalità, devono essere assemblati e rilasciati (deployed) in un contenitore
  - il contenitore fornisce ai propri componenti anche degli ulteriori servizi specifici – ad es., sicurezza, gestione delle transazioni, persistenza, scalabilità, disponibilità, ...
    - i componenti Java EE si aspettano infatti che il contenitore Java EE gli offra questi servizi
    - l'assemblaggio di un'applicazione comprende la definizione di parametri per personalizzare il tipo di supporto che il contenitore deve fornire alla specifica applicazione
    - il contenitore poi eroga questi servizi mentre si occupa della gestione del ciclo di vita dei componenti, sulla base di meccanismi di intercettazione delle richieste e delegazione

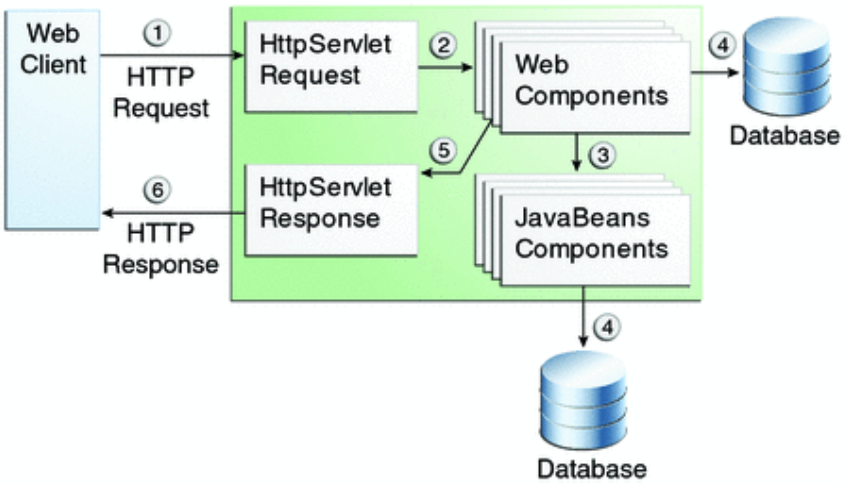


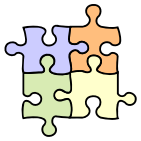
Contenitori





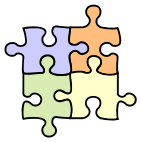
Un esempio noto: applicazioni web





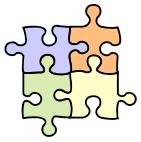
Un esempio noto: applicazioni web

- Una *servlet* (HTTP) è una classe usata per estendere le capacità del server web che ospita l'applicazione sulla base di un modello di programmazione richiesta-risposta
- Ciclo di vita (a runtime) di una servlet – controllato dal contenitore web
  - quando il server riceve una richiesta per una servlet
    - se un'istanza non è stata creata, la crea e la inizializza
    - invoca il metodo di servizio – ad es., doPost – passandogli gli oggetti richiesta e risposta
  - se il server ha bisogno di rimuovere la servlet, la finalizza e la rimuove
- In linea di principio, un oggetto servlet può essere usato per gestire zero, una o più richieste – in zero, una o più sessioni

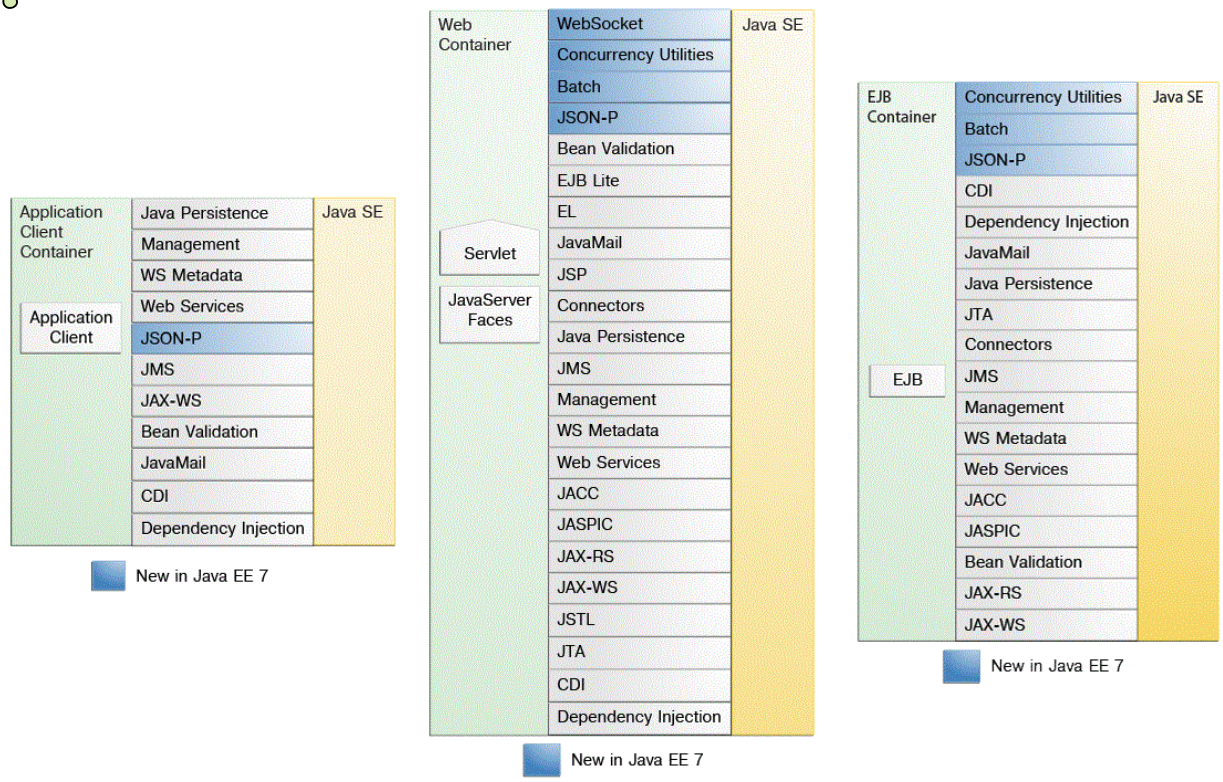


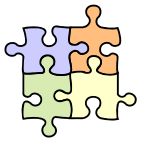
Un esempio noto: applicazioni web

- Condivisione di informazioni tra componenti web
  - non è opportuno che informazioni da condividere siano gestite dall'oggetto servlet
  - sono usati alcuni oggetti di supporto – gestiti dal contenitore
    - contesto web – per gestire lo stato dell'applicazione
    - sessione – per gestire lo stato della sessione
    - richiesta
    - pagina
- Come fa un componente web a conoscere questi oggetti?
  - la servlet può chiedere il suo contesto al contenitore
  - il contenitore passa la sessione alla servlet – nella richiesta – possibile perché è il contenitore che intercetta le richieste
  - contestualmente, il contenitore può offrire ulteriori servizi ai suoi componenti



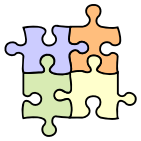
API di Java EE





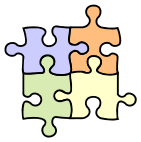
* Enterprise Bean

- Gli **Enterprise Bean** sono componenti Java EE che implementano lo standard **EJB** (*Enterprise Java Bean*)
  - componenti server-side
    - la specifica EJB definisce un “modello (standard) per componenti”
  - in applicazioni multi-livello
  - per implementare la logica di business di un’applicazione
  - sono eseguiti in un contenitore EJB
    - il contenitore fornisce agli Enterprise Bean, in modo trasparente, servizi come gestione delle transazioni, controllo della concorrenza, affidabilità, sicurezza, scalabilità
    - poiché questi servizi sono già disponibili, lo sviluppatore può concentrarsi sulla soluzione di problemi di business
  - componenti portabili, e che possono essere riusati in applicazioni diverse



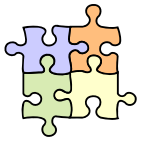
Quando usare gli Enterprise Bean

- L'utilizzo degli enterprise bean può essere suggerito dai seguenti requisiti
  - l'applicazione deve essere scalabile
    - la realizzazione distribuita di un application server sostiene la scalabilità orizzontale delle applicazioni composte da enterprise bean
  - l'applicazione deve avere una molteplicità di tipologie di client
    - è semplice scrivere dei componenti client per utilizzare servizi offerti dagli enterprise bean
  - necessità di usare transazioni per garantire l'integrità dei dati
    - gli enterprise bean possono essere usati in modo transazionale



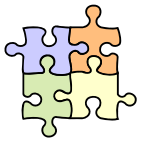
Tipi di Enterprise Bean

- Due tipi principali di enterprise bean – in prima approssimazione, sono basati su due stili di comunicazione differenti
  - *session bean*
    - componente che incapsula logica applicativa, offrendo servizi che possono essere richiesti dai suoi client
  - *message-driven bean*
    - componente in grado di ricevere e elaborare messaggi in modo asincrono



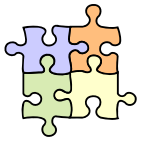
Session bean

- Session bean
  - un *session bean* è un componente che incapsula logica applicativa – i suoi servizi possono essere richiesti da un client locale o remoto
    - un'istanza di un session bean sarà utilizzata per eseguire compiti richiesti da uno specifico client – per questo, un session bean può rappresentare un singolo client all'interno di un'applicazione
  - lo stato di un session bean può riflettere (o meno) la sua interazione con un client particolare
    - in linea di principio, è possibile avere session bean le cui istanze possono essere usate in modo esclusivo da un solo client, oppure in modo condiviso da più client



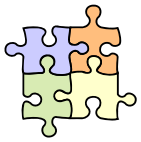
Session bean

- Ci sono tre tipi di session bean
  - *stateful session bean*
    - definisce un comportamento lato server – ciascuna istanza rappresenta una specifica conversazione (sessione) con un singolo client – e consente di gestire lo stato di quella sessione, per tutta la durata della conversazione
  - *stateless session bean*
    - definisce un comportamento lato server – il contenitore EJB gestisce un pool di istanze, che sono condivise da più client (le istanze non mantengono informazioni sulle sessioni con i loro client) – può implementare un web service
  - *singleton session bean*
    - simile a un session bean stateless – viene inoltre garantita la creazione di un singolo session bean per applicazione, che esiste per tutto il ciclo di vita dell'applicazione



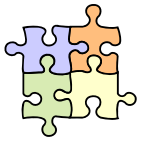
Session bean e stato conversazionale

- Si noti che la presenza o l'assenza di stato
  - non ha a che fare direttamente con il fatto che un session bean gestisca o meno un proprio stato
  - riguarda il fatto che il session bean conservi – o meno – lo stato della sessione/conversazione con il suo client



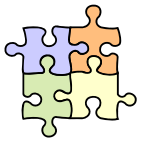
Session bean e interfacce

- I client di un session bean possono accedere ai servizi offerti dal bean tramite la sua interfaccia (fornita) – l'interfaccia (fornita) di un session bean può essere di diversi tipi
  - *business interface*
    - il bean implementa un'interfaccia Java che definisce *in modo esplicito* i metodi di business del bean
  - *no-interface view*
    - il bean non implementa nessuna interfaccia in modo esplicito – ma espone i metodi pubblici della classe che lo implementa
    - di solito poco preferibile, poiché il client del bean è accoppiato direttamente alla classe che implementa il bean



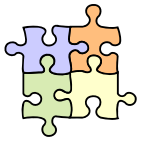
Session bean e interfacce

- Un client – un componente web, un altro enterprise bean o un application client – può accedere ai servizi di un session bean secondo diverse modalità
  - *remote access*
    - il bean può essere acceduto anche da client remoti – che vivono in un contenitore remoto
    - può essere chiaramente acceduto anche da client locali
  - *local access*
    - il bean può essere acceduto solo da client che vivono nello stesso contenitore del bean e che fanno parte della stessa applicazione
    - l'unica opzione possibile in caso di no-interface view
  - *web service access*
    - descritto in una successiva dispensa



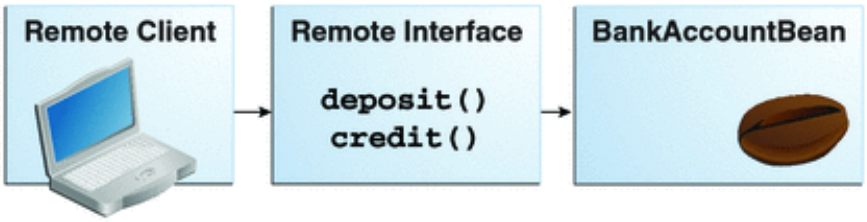
Message-driven bean

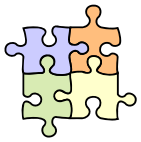
- Message-driven bean
  - un *message-driven bean* è un componente per ricevere e elaborare messaggi asincroni – in pratica, agisce come un listener di messaggi JMS (o di altri tipi di messaggi)
    - definisce un metodo **onMessage** – che viene invocato dal contenitore EJB quando è disponibile un messaggio in una certa destinazione
    - la destinazione da cui riceve messaggi è specificata dal suo descrittore di deployment
  - può anche inviare messaggi a destinazioni JMS
- Nota
  - anche i session bean possono essere usati per inviare e ricevere messaggi JMS – tuttavia, in un session bean la ricezione può avvenire solo secondo la modalità "sincrona" (**receive**) e non secondo quella "asincrona" (**onMessage**)



* Session bean

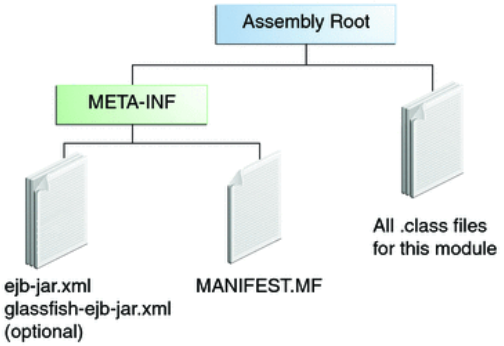
- Consideriamo il modello di programmazione degli enterprise bean di tipo session
  - consideriamo il caso di un session bean che implementa e espone un'*interfaccia di business remota*
    - per brevità, ignoriamo l'uso di interfacce locali
  - i client remoti del session bean possono accedere ai servizi offerti mediante l'interfaccia remota

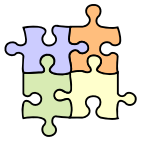




Modello di programmazione

- Implementazione di un session bean
  - un session bean è costituito da vari elementi
    - l'interfaccia di business
    - la classe che implementa il session bean
    - informazioni circa la sua configurazione
  - questi elementi vanno opportunamente "impacchettati" per poterne fare il deployment su un application server Java EE



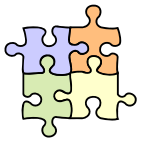


Modello di programmazione

- Implementazione di un session bean
  - un session bean è costituito da vari elementi
    - l'interfaccia di business
    - la classe che implementa il session bean
    - informazioni circa la sua configurazione
  - questi elementi vanno opportunamente "impacchettati" per poterne fare il deploy [illegible] va EE

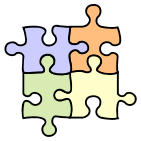
più precisamente, l'implementazione di un enterprise bean è di solito costituita da un gruppo di oggetti "tradizionali" – rappresentati complessivamente tramite un oggetto facade, designato come enterprise bean

(optional)



Parentesi: Eclipse e EJB

- Eclipse (versione per Java EE) definisce diversi tipi di "progetti" per Java EE e per gli Enterprise Bean – in particolare
  - *EJB project*
    - consente di definire uno o più EJB e di assemblarli in una unità di deployment (un file JAR) – tuttavia, un EJB può essere deployato e eseguito su un server solo se referenziato da un enterprise application project
  - *Enterprise Application project*
    - racchiude le risorse per definire un'applicazione di tipo enterprise – contiene (referenzia) un insieme di moduli, e li assembla in un file EAR
  - *Application Client project*
    - definisce un client (eseguibile) che può fruire di risorse lato server – ad esempio, l'iniezione di risorse e l'accesso a EJB oppure a destinazioni JMS



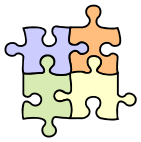
- Il session bean HelloBean (prima versione)

- Consideriamo un breve esempio
  - un enterprise bean di tipo session/stateless

- Interfaccia remota

```
package asw.asw850.hello;

import javax.ejb.Remote;

@Remote
public interface HelloRemote {

    /** Saluta nome. */
    public String hello(String name);

}
```

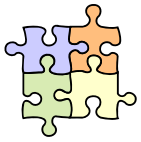
in Eclipse, session bean definito nell'ambito di un progetto EJB
da compilare e fare il deploy su un AS



Implementazione per HelloBean

```
package asw.asw850.hello;

import javax.ejb.Stateless;

@Stateless(mappedName = "ejb/asw850/HelloBean")
public class HelloBean implements HelloRemote {

    /** Saluta nome. */
    public String hello(String name) {
        return "Hello, " + name + "!";
    }

}
```

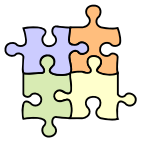
Nota: sono possibili diverse convenzioni con i nomi:
- HelloRemote o HelloRemoteBean o Hello per l'interfaccia
- HelloBean o Hello o HelloImpl per l'implementazione



Application Client per HelloBean (1)

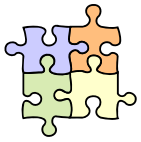
```
package asw.asw850.hello.client;

import asw.asw850.hello.HelloRemote;

import javax.ejb.EJB;

public class HelloClient {
   @EJB(lookup = "ejb/asw850/HelloBean")
   private static HelloRemote hello;

   public HelloClient() { }

   ...
}
```

da compilare ed eseguire come application client



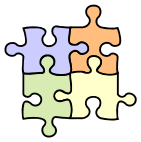
Application Client per HelloBean (2)

```
public static void main(String[] args) {
   HelloClient client = new HelloClient();
   client.doConversation();
}

public void doConversation() {
   String answer = hello.hello("Luca");
   System.out.println(answer);
}
```



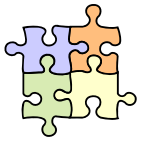
Enterprise bean e annotazioni

- Nel codice mostrato, le annotazioni rivestono un ruolo importante
  - le annotazioni @Remote, @Local, @Stateless, @Stateful e @Singleton dichiarano il ruolo svolto dalle diverse unità di programmazione
  - le annotazioni @Remote e @Local sono relative alla *tipologia* di un'*interfaccia* per enterprise bean
  - le annotazioni @Stateless, @Stateful e @Singleton hanno a che fare con le *interfacce fornite* da un enterprise bean
    - interfaccia fornita – specifica di servizi forniti da un componente ad altri componenti
    - inoltre, queste annotazioni consentono di specificare il nome simbolico JNDI con cui registrare l'enterprise bean – ad es., ejb/asw850/HelloBean – la registrazione su JNDI avviene al momento del deployment



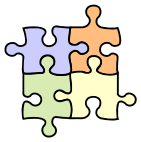
Enterprise bean e annotazioni

- Nel codice mostrato, le annotazioni rivestono un ruolo importante
  - viceversa, l'annotazione @EJB indica una dependency injection
    - prima di iniziare l'esecuzione dell'application client, verrà effettuata una ricerca JNDI di un enterprise bean di tipo HelloRemote dal nome specificato (ejb/asw850/HelloBean) – il riferimento remoto al bean verrà memorizzato nella variabile hello
  - l'annotazione @EJB ha a che fare con le *interfacce richieste* da un client – ad es., un application client oppure un altro enterprise bean o un altro tipo di client
    - interfaccia richiesta – specifica di servizi che devono essere resi disponibili a un componente affinché possa funzionare come specificato



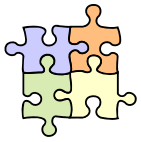
Enterprise bean e annotazioni

- Nel codice mostrato, le annotazioni rivestono un ruolo importante
  - anche se non è mostrato in questo esempio, l'implementazione di un enterprise bean potrebbe dichiarare sia interfacce fornite che interfacce richieste, utilizzando tutti i seguenti elementi
    - un'annotazione di tipo @Stateless oppure @Stateful oppure @Singleton – per indicare l'implementazione di un EJB
    - l'implementazione di una o più interfacce remote o locali – per indicare le interfacce fornite dal bean
    - una o più annotazioni di tipo @EJB – per indicare le interfacce richieste dall'enterprise bean e specificare anche la composizione con altri componenti EJB



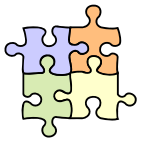
Enterprise bean e composizione

- La composizione (ovvero, il binding, collegamento) tra i diversi componenti runtime di un'applicazione Java EE – nell'esempio, tra l'EJB e l'application client, ma è possibile anche tra EJB e EJB – viene effettuato sulla base di informazioni di deployment
  - nell'esempio, queste informazioni sono immerse nel codice tramite annotazioni
  - in pratica, questi metadati possono essere separati dal codice in opportuni file di configurazione
- Si noti comunque il tipo di dipendenze tra componenti – usando sia annotazioni che altre informazioni di deployment
  - un componente servente non dipende dai suoi client
  - il client di un enterprise bean deve conoscere l'interfaccia remota e il nome simbolico del bean – ma non deve sapere nulla circa la sua particolare implementazione, ed in particolare non deve conoscere la sua classe



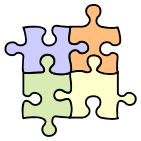
Modello di programmazione

- Ulteriori osservazioni
  - notare che le istanze di un enterprise bean non vengono mai create esplicitamente dal programmatore
    - come una servlet, verranno create dal contenitore – se e quando serve
  - le chiamate al bean sono in realtà chiamate remote – inoltre ciascuna chiamata viene gestita in un proprio thread
    - come in Java RMI
  - come per le servlet, ci aspettiamo che il bean possa definire dei metodi che possono essere chiamati dal contenitore – e non dai suoi client remoti – in momenti specifici della vita del bean
    - questo aspetto sarà discusso tra breve



Semantica dei session bean stateless

- Nel caso di un session bean di tipo stateless
  - il contenitore EJB gestisce un pool di istanze del session bean
  - un'istanza del session bean può essere condivisa da uno o più client – nell'ambito di una o più "sessioni di lavoro"
  - le richieste successive di un particolare client di un session bean – anche nell'ambito di una singola "sessione di lavoro" – possono essere gestite da istanze differenti del session bean
    - in realtà, potrebbero venire anche gestite tutte da una stessa istanza del session bean – ma questo sarebbe solo un caso (non c'è nessuna garanzia di questo)

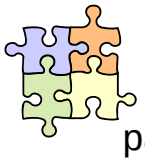


- Il session bean HelloBean (seconda vers.)

- Consideriamo un ulteriore esempio di enterprise bean di tipo session/stateless – un'estensione dell'esempio precedente

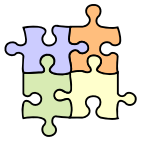
- Interfaccia remota

```
package asw.asw850.hello;

import javax.ejb.Remote;

@Remote
public interface HelloRemote {

    /** Saluta nome, usando un linguaggio specifico. */
    public String hello(String name, String language);

}
```

Implementazione per HelloBean

```
package asw.asw850.hello;

import javax.ejb.Stateless;
import javax.annotation.*;
import java.util.*;

@Stateless(mappedName = "ejb/asw850/HelloBean")
public class HelloBean implements HelloRemote {

    /* mappa dei saluti nelle diverse lingue */
    private Map<String, String> saluti;

    ... inizializza un'istanza del bean ...

    /** Saluta nome, usando un linguaggio specifico. */
    public String hello(String name) {
        String format = saluti.get(language);
        return String.format(format, name);
    }
}
```

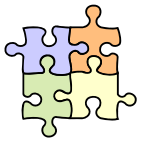


Implementazione per HelloBean

- L'inizializzazione dello stato di un enterprise bean va fatta in un metodo annotato @PostConstruct
  - e non nel costruttore – che normalmente non va definito

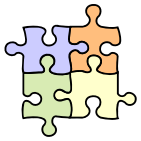
```
/* inizializza un'istanza del bean */
@PostConstruct
public void initialize() {
        /* inizializza la mappa dei saluti */
        saluti = new HashMap<String, String>();
        saluti.put("it", "Ciao, %s!");
        saluti.put("en", "Hello, %s!");
        saluti.put("fr", "Bonjour, %s!");
        ...
}
```

un metodo annotato @PostConstruct viene eseguito subito dopo la costruzione dell'istanza del bean



Stato di un bean stateless

- In generale, i session bean di tipo stateless (così come quelli di tipo singleton) non vanno usati per gestire uno stato relativo ad "informazioni di dominio"
  - né circa lo stato delle conversazioni con i client
    - in questo caso va probabilmente usato un bean stateful
  - ma anche nemmeno circa lo stato di un'applicazione
    - in questo caso, lo stato dell'applicazione va probabilmente rappresentato "fuori" dai bean – in particolare, nella base di dati
- Tuttavia, è possibile che bean di tipo stateless (così come quelli di tipo singleton) abbiano variabili d'istanza
  - usate però per gestire, ad esempio, l'accesso a risorse (come connessioni) oppure informazioni comuni (ad esempio, perché immutabili)



- Il session bean SessionCounterBean

- Consideriamo ora un altro esempio
  - un enterprise bean di tipo session/stateful

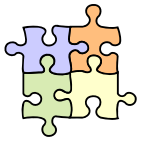
- Interfaccia remota

```
package asw.asw850.counter;

import javax.ejb.Remote;

@Remote
public interface SessionCounterRemote {

    /** Incrementa il contatore e restituisce il suo valore. */
    public int getCounter();

    /** Termina la sessione. */
    public void close();
}
```


Implementazione di SessionCounterBean

```
package asw.asw850.counter;

import javax.ejb.Stateful;
import javax.ejb.Remove;
import javax.annotation.*;

@Stateful(mappedName = "ejb/asw850/SessionCounterBean")
public class SessionCounterBean implements SessionCounterRemote {
    private int counter;    // valore corrente del contatore

    /* inizializza il session bean,
    * assegnando un valore iniziale al contatore */
    @PostConstruct
    public void initialize() {
        counter = 0;
    }

    … segue …
```

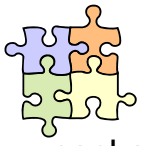
un metodo annotato @PostConstruct viene eseguito subito dopo la costruzione dell'istanza del bean



Implementazione di SessionCounterBean

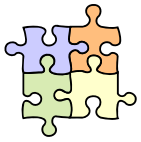
```
/** Incrementa il contatore e restituisce il suo valore. */
public int getCounter() {
        counter++; return counter;
}

/** Termina la sessione. */
@Remove
public void close() { }
}
```

l'esecuzione di un metodo annotato @Remove causa la terminazione della sessione sull'application server - che può così liberare le risorse impegnate nella gestione della conversazione



App. Client per SessionCounterBean (1)

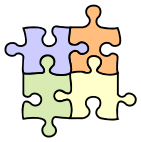
```
package asw.asw850.counter.client;

import asw.asw850.counter.CounterRemote;
import javax.ejb.EJB;

public class SessionCounterClient {
    @EJB(lookup = "ejb/asw850/SessionCounterBean")
    private static SessionCounterRemote counter;

    public SessionCounterClient() { }

    public static void main(String[] args) {
        SessionCounterClient client = new SessionCounterClient();
        client.doConversation();
    }

    ... segue ...
}
```



App. Client per SessionCounterBean (2)

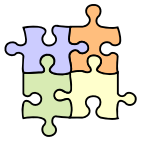
```
public void doConversation() {
    for (int i=1; i<=50; i++) {
        /* che cosa verrà stampato? */
        System.out.println( counter.getCounter() );
    }
    counter.close();
}
```

- Che cosa succede durante l'esecuzione di questo application client?
  - poiché counter è un session bean di tipo stateful, allora ogni client otterrà
    - valori del contatore in sequenza
    - partendo in ogni caso da 1
    - anche in presenza di più client concorrenti



Semantica dei session bean stateful

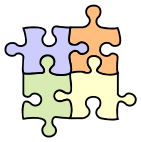
- Nel caso di un session bean di tipo stateful, *in linea di principio*
  - il contenitore EJB gestisce un'istanza distinta del session bean per ciascuna "sessione di lavoro" di ciascun client
    - le istanze del session bean non sono condivise da più client – nemmeno da sessioni diverse di uno stesso client
  - ciascuna richiesta successiva di un particolare client di un session bean – nell'ambito di una singola "sessione di lavoro" – viene gestita sempre da una stessa istanza del session bean
  - ma che vuol dire quel "in linea di principio"?
    - la semantica di riferimento è quella delineata qui sopra – quindi il programmatore può sempre pensare in termini di questa semantica
    - tuttavia, l'implementazione potrebbe essere diversa



- Per riflettere

- Si consideri ancora il session bean SessionCounter e il suo client SessionCounterClient
  - si supponga però di dichiarare il bean **stateless** anziché stateful – anche se di per sé questo non ha molto senso, se non per discutere la semantica dei bean stateless
  - che cosa si può aspettare il SessionCounterClient dall'esecuzione del metodo doConversation?
  - che cosa può succedere se c'è un solo client in esecuzione? e più client concorrenti? e più client in successione?

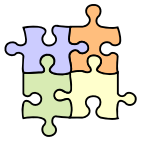
```
public void doConversation() {
    for (int i=1; i<=50; i++) {
        /* che cosa succederà in questo caso? */
        System.out.println( counter.getCounter() );
    }
}
```

- Per riflettere ancora

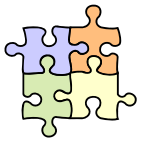
- Si consideri ancora il session bean SessionCounter e il suo client SessionCounterClient
  - si supponga però di dichiarare il bean **singleton** anziché stateful – anche se di per sé questo non ha molto senso, se non per discutere la semantica dei bean singleton
  - che cosa si può aspettare il SessionCounterClient dall'esecuzione del metodo doConversation?
  - che cosa può succedere se c'è un solo client in esecuzione? e più client concorrenti? e più client in successione?

```
public void doConversation() {
    for (int i=1; i<=50; i++) {
        /* che cosa succederà in questo caso? */
        System.out.println( counter.getCounter() );
    }
}
```



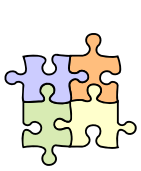
Semantica dei session bean singleton

- Nel caso di un session bean di tipo singleton, *in linea di principio*
  - il contenitore EJB gestisce una singola istanza del session bean
    - questa istanza del session bean è condivisa da tutti i client del sessione bean
  - dunque, tutte le richieste di tutti i client sono gestite sempre dalla stessa istanza (singleton) del session bean
- Attenzione
  - a seguito di uno shutdown o del crash del contenitore EJB, l'istanza del bean potrebbe venire distrutta e creata di nuovo
  - questo motiva il suggerimento di non usare nemmeno i bean di questo tipo per rappresentare lo stato di un'intera applicazione
  - meglio rappresentare lo stato di un'applicazione in una base di dati – gestita esternamente all'application server

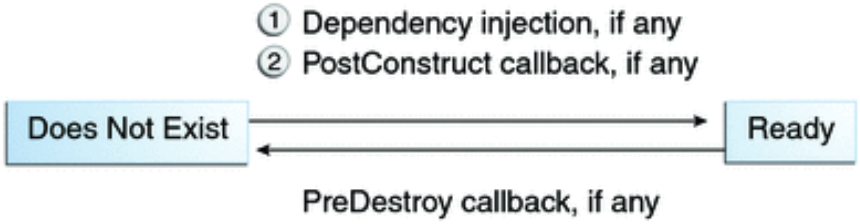


- Ciclo di vita dei session bean

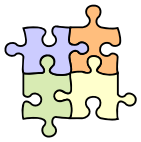
- La programmazione degli enterprise bean richiede la conoscenza del loro ciclo di vita
  - durante alcuni eventi significativi della vita di un enterprise bean, vengono infatti invocati eventuali metodi etichettati con delle opportune annotazioni – è ad esempio il caso di metodi annotati @PostConstruct
  - queste invocazioni sono fatte da parte del contenitore EJB, in cui gli enterprise bean vivono
- In tutti i casi, l'invocazione dei metodi di servizio da parte dei client dell'enterprise bean avviene solo quando il bean si trova nello stato *Ready*



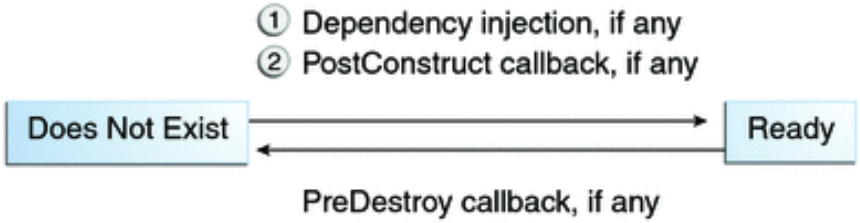
Ciclo di vita dei session bean stateless



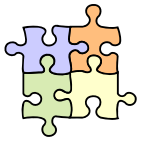
- Il contenitore EJB gestisce di solito un pool di istanze di session bean stateless
  - quando è necessario creare un'istanza (a discrezione del contenitore) – la creazione viene seguita dall'iniezione di dipendenze, e poi dall'esecuzione dei metodi etichettati @PostConstruct
  - quando è necessario distruggere un'istanza (a discrezione del contenitore) – la distruzione viene preceduta dall'esecuzione dei metodi etichettati @PreDestroy



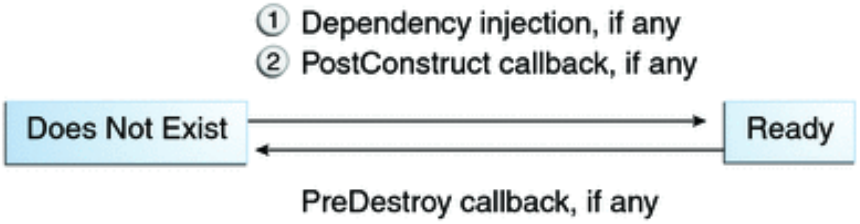
Ciclo di vita dei session bean stateless



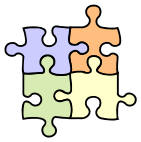
- Ad esempio, si consideri un session bean (stateless) che deve implementare un metodo **findProduct(int id)** per effettuare una ricerca in una base di dati
  - **findProduct** potrebbe aprire ogni volta una connessione alla base di dati, fare l'interrogazione, e poi chiudere la connessione
  - ancora meglio, il bean potrebbe usare una singola connessione jdbc, da usare in più esecuzioni di **findProduct** – la connessione può essere aperta in un metodo annotato @PostConstruct, e chiusa in un metodo annotato @PreDestroy



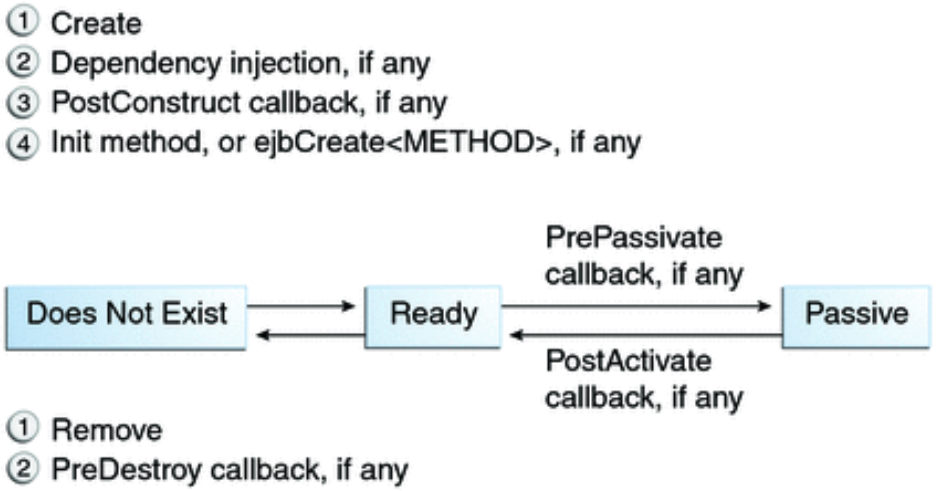
Ciclo di vita dei session bean singleton



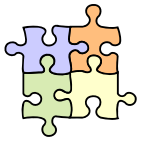
- Il contenitore EJB gestisce una singola istanza di un session bean singleton
  - il ciclo di vita dei session bean singleton è lo stesso dei session bean stateless



Ciclo di vita dei session bean stateful (1)

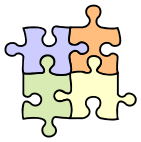


- Il ciclo di vita dei session bean stateful è più complesso
  - un session bean stateful può essere "passivato" – lo stato di un bean (una sessione con un client) può essere serializzato e salvato automaticamente in memoria secondaria – per essere poi "attivato" (ricaricato) in memoria in un secondo tempo



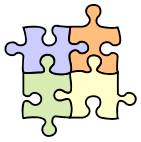
Ciclo di vita dei session bean stateful (2)

- I metodi di callback del ciclo di vita devono essere pubblici, senza parametri, void – annotati come segue
  - @PostConstruct
    - invocato dal contenitore su un'istanza del bean appena costruita – dopo che tutte le iniezioni di dipendenze sono state completate – prima della prima invocazione dei metodi di business
  - @Remove
    - un metodo annotato @Remove può essere invocato direttamente dal client per indicare la fine della sessione – ad es., @Remove public void endSession() { }
  - @PreDestroy
    - invocato dal contenitore dopo l'esecuzione di un metodo annotato @Remove – prima che l'istanza del bean sia rimossa dal contenitore



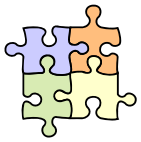
Ciclo di vita dei session bean stateful (3)

- @PrePassivate
  - invocato dal contenitore prima che il contenitore passivi il bean (ovvero, lo serializza e ne salva lo stato in memoria secondaria) – viene invocato subito prima che il container rimuova il bean (temporaneamente) dalla sua memoria
- @PostActivate
  - invocato dal contenitore dopo aver attivato (ovvero, caricato lo stato dalla memoria secondaria e deserializzato) il bean



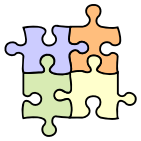
Passivazione a attivazione

- Il container gestisce lo swapping di istanze di bean, mediante due ulteriori operazioni
  - *passivazione* – lo stato del bean viene serializzato e salvato in memoria secondaria (a cura del contenitore, non del programmatore!) – il metodo @PrePassivate è utile per rilasciare risorse non serializzabili (ad es., connessioni)
  - *attivazione* – lo stato del bean viene ripristinato dalla memoria secondaria (a cura del contenitore, non del programmatore!) – il metodo @PostActivate è utile per recuperare risorse non serializzabili
  - va notato anche che lo stato di un bean serializzato può essere anche trasferito a un contenitore (nodo) diverso e poi lì ripristinato – utile per scalabilità e disponibilità



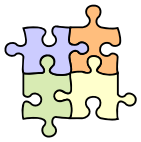
Un pool di istanze

- In generale, anche per ridurre il costo delle creazioni delle istanze degli enterprise bean, il contenitore EJB gestisce un pool di istanze di bean (dei diversi tipi)
  - assegna le richieste alle istanze in questo pool
  - ad esempio, può decidere di riassegnare un oggetto, inizialmente assegnato a un client C1, a un client diverso C2
    - questi due client potrebbero non aver ancora completato le loro “sessioni”
  - mentre questo non crea problemi per i bean stateless e singleton, sono necessari degli opportuni meccanismi per garantire la semantica dei session bean stateful – e al tempo stesso per sostenere prestazioni, scalabilità, affidabilità



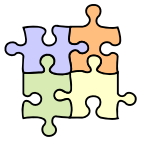
Semantica dei session bean stateful

- Nel caso di un session bean di tipo stateful
  - in linea di principio, un'istanza del session bean è associata a una singola sessione di un singolo client
    - in realtà, un'istanza utilizzata in precedenza nell'ambito di una sessione per un client potrebbe essere riusata in un'altra sessione per un altro client – ma prima ripetendo l'esecuzione dei metodi annotati @PostConstruct
    - il contenitore EJB può gestire un pool di istanze – per evitare la creazione di nuove istanze del session bean
  - in linea di principio, richieste successive di uno stesso client, nell'ambito di una singola sessione, sono tutte gestite da un'unica istanza del session bean
    - in realtà, l'istanza associata a quella sessione potrebbe essere stata passivata (e distrutta) – poi (ricreata e) riattivata – il che corrisponde a una nuova istanza, o a un'istanza "riciclata" – che potrebbe essere addirittura allocata su un nodo diverso



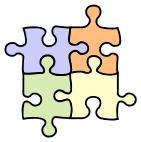
* Operazioni asincrone

- I session bean possono implementare anche delle *operazioni asincrone*
  - quando un client invoca un'operazione asincrona, il controllo torna immediatamente al client chiamante – prima che l'esecuzione dell'operazione sia terminata (in realtà, prima anche che sia cominciata)
  - il client può poi proseguire il suo lavoro, ignorando l'esecuzione dell'operazione – oppure, può usare le API per ottenere il risultato, oppure per cancellare l'esecuzione dell'operazione
  - le operazioni asincrone sono utili per implementare richieste di tipo *send-and-forget* – oppure in caso di operazioni di lunga durata



AsynchResult<V> e Future<V>

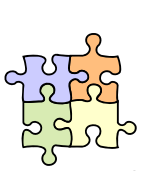
- Il modello di programmazione delle operazioni asincrone è basato su
  - la classe AsynchResult<V> – usata da un'operazione asincrona per restituire un valore di tipo V
  - l'interfaccia Future<V> – usata per indicare che un'operazione asincrona restituirà un valore di tipo V
  - al momento dell'invocazione di un'operazione asincrona, al client viene immediatamente restituito un oggetto Future<V> – il client può poi usare le seguenti operazioni
    - V get() (bloccante) – per ottenere il valore restituito
    - boolean isDone() – per sapere se l'esecuzione dell'operazione asincrona è terminata
    - cancel(true) – per chiedere la cancellazione dell'operazione asincrona



- Il bean AsynchronousBean

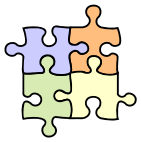
- Consideriamo un'operazione asincrona nell'ambito di un enterprise bean di tipo session/stateless
  - interfaccia remota

```
package asw.asw850.asynchronous;

import javax.ejb.Remote;
import javax.ejb.Asynchronous;
import java.util.concurrent.Future;

@Remote
public interface AsynchronousRemote {

    /** Un'operazione asincrona. */
    @Asynchronous
    public Future<String> asynchOperation(String param);

}
```



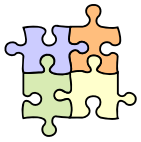
Implementazione per AsynchronousBean

```
package asw.asw850.asynchronous;

import java.util.concurrent.Future;
import javax.ejb.AsyncResult;
import javax.ejb.Asynchronous;
import javax.ejb.Stateless;

@Stateless(mappedName = "ejb/asw850/AsynchronousBean")
public class AsynchronousBean implements AsynchronousRemote {

    /** Un'operazione asincrona. */
    @Asynchronous
    public Future<String> asynchOperation(String param) {
        result = … il calcolo del risultato richiede un po' di tempo …
        return new AsyncResult<String>(result);
    }
}
```

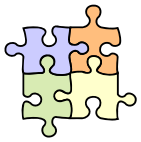
Application Client per AsynchronousBean (1)

```
package asw.asw850.asynchronous.client;

import asw.asw850.asynchronous.AsynchronousRemote;

import javax.ejb.EJB;
import java.util.concurrent.*;

public class AsynchronousClient {
   @EJB(lookup = "ejb/asw850/AsynchronousBean")
   private static AsynchronousRemote asynch;

   public static void main(String[] args) {
      new AsynchronousClient() . doConversation();
   }

   public void doConversation() { … }

}
```



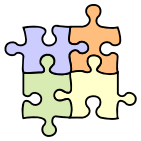
Application Client per AsynchronousBean (2)

```
public void doConversation() {
    try {
        Future<String> futureResult = asynch.asynchOperation("Alfa");
        … la chiamata termina subito,
            quindi il nostro client può mettersi a fare altre cose …
        … quando il client ha effettivamente bisogno del risultato …
        String result = futureResult.get();    // bloccante
        System.out.println(result);
    } catch(InterruptedException | ExecutionException e) {
        … gestisci l'eccezione …
    }
  }
}
```

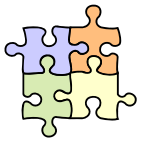
* Message-driven bean

- Message-driven bean
  - un message-driven bean è un enterprise bean che consente di elaborare messaggi in modo asincrono
    - funge normalmente da ascoltatore di messaggi JMS
    - questi messaggi JMS possono essere stati inviati da altri componenti, Java EE e non



Similitudini e differenze

- Per certi versi, i message-driven bean sono simili ai session bean stateless
  - le istanze sono equivalenti, poiché non gestiscono stato conversazionale
  - un message-driven bean può elaborare messaggi da più client
- Per altri versi, i message-driven bean sono diversi dai session bean
  - infatti non vengono acceduti mediante un'interfaccia (procedurale) remota – ma piuttosto tramite una destinazione JMS

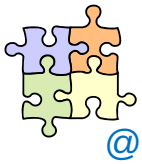


Un esempio

- Si supponga di voler realizzare un'applicazione di messaging con i seguenti componenti
  - un produttore di messaggi
    - ad esempio, basata sull'endpoint SimpleProducer mostrato nel contesto di JMS
    - invia messaggi a una coda jms/asw850/QueueOne
  - un filtro, che trasforma messaggi
    - riceve messaggi dalla coda jms/asw850/QueueOne
    - invia messaggi alla coda jms/asw850/QueueTwo
    - realizzato come message-driven bean (MDB)
  - un consumatore di messaggi
    - ad esempio, basato sull'endpoint SimpleAsynchConsumer mostrato nel contesto di JMS
    - riceve messaggi da una coda jms/asw850/QueueTwo

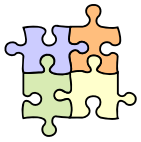
Il MDB FiltroMessaggiBean (1)

```
package asw.asw850.filtro;

import asw.jms.simpleproducer.*;

import javax.ejb.*;
import javax.jms.*;
import javax.annotation.Resource;

/* specifica la destinazione da cui il bean riceve messaggi */
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
            propertyName = "destinationType",
            propertyValue = "javax.jms.Queue"
        ) },
    mappedName = "jms/asw850/QueueOne")
public class FiltroMessaggiBean implements MessageListener {
    ...
}
```

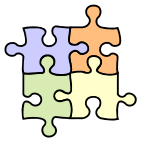
Il MDB FiltroMessaggiBean (2)

```
@MessageDriven(...)
public class FiltroMessaggiBean implements MessageListener {

    /* la destinazione su cui il bean invia messaggi */
    @Resource(lookup = "jms/asw850/QueueTwo")
    private Queue queue;
    @Resource(lookup = "jms/asw850/ConnectionFactory")
    private ConnectionFactory connectionFactory;

    /* produttore di messaggi associato a questo message-driven bean */
    private SimpleProducer simpleProducer;

    public FiltroMessaggiBean() {  }

    ...

}
```



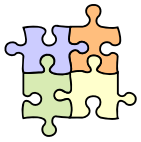
Il MDB FiltroMessaggiBean (3)

```
/* crea un produttori di messaggi per il MDB per la coda
 * QueueTwo ed apre la connessione JMS */
@PostConstruct
public void initialize() {
        this.simpleProducer =
                new SimpleProducer("FiltroMessaggiBean",
                                        queue, connectionFactory);
        simpleProducer.connect();
}

/* chiude la connessione JMS */
@PreDestroy
public void destroy() {
        simpleProducer.disconnect();
}
```

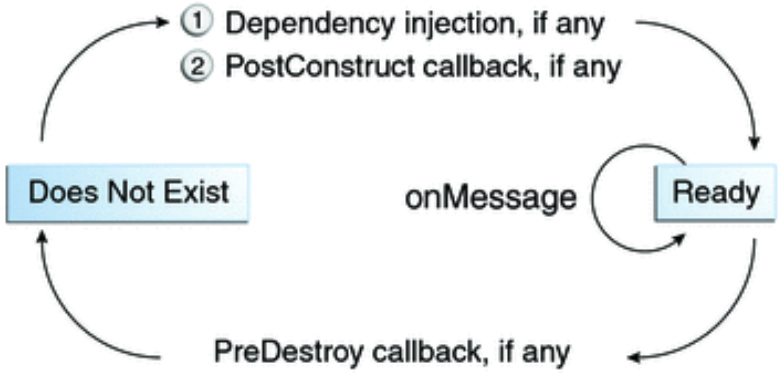



Il MDB FiltroMessaggiBean (4)

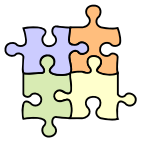
```
/* riceve un messaggio dalla coda QueueOne
 * (ne delega la gestione al metodo processMessage) */
public void onMessage(Message m) {
    if (m instanceof TextMessage) {
        TextMessage message = (TextMessage) m;
        try {
            this.processMessage( message.getText() );
        } catch (JMSException e) {
            System.out.println("Filtro.onMessage(): " + e.toString());
        }
    }
}

/* trasforma il messaggio ricevuto e lo invia alla coda QueueTwo */
private void processMessage(String ingoingMessage) {
    String outgoingMessage = ingoingMessage.toUpperCase();
    simpleProducer.sendMessage( outgoingMessage );
}
```



Ciclo di vita dei message-driven bean

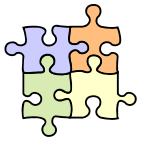


- Il contenitore EJB gestisce di solito un pool di istanze di message-driven bean
  - come i session bean stateless, i message-driven bean non vengono mai passivati – e possono trovarsi in due soli stati



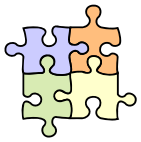
* Discussione

- In generale, un'applicazione Java EE sarà composta, nel livello EJB, da diversi Enterprise Bean
  - ma quanti e quali Enterprise Bean utilizzare in un'applicazione, e di che tipo?
  - questo aspetto sarà ripreso del contesto metodologico
    - ad esempio, si veda [UML Components]



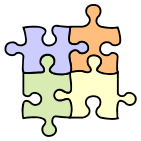
Un breve esempio

- Si consideri ad esempio un sistema di commercio elettronico, per la gestione di ordini, fatti da clienti relativamente a prodotti
  - come può essere realizzata con la piattaforma Java EE? (quello che viene descritta è solo una possibilità)
- Livello client
  - il sistema viene acceduto mediante un client (browser) web
- Livello web
  - è possibile definire un'applicazione web per ciascun attore primario del sistema – o per un gruppo di casi d'uso per un attore primario
  - ad esempio, un'applicazione web ClienteWebApp



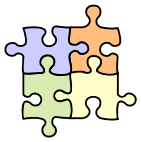
Un breve esempio

- Si consideri ad esempio un sistema di commercio elettronico, per la gestione di ordini, fatti da clienti relativamente a prodotti
  - come può essere realizzata con la piattaforma Java EE? (quello che viene descritta è solo una possibilità)
- Livello EJB – session bean stateful
  - è possibile definire un session bean stateful per ciascun caso d'uso del sistema – tra cui, ad esempio, un session bean InserimentoOrdineBean
  - ciascun bean di questo tipo è responsabile delle operazioni di sistema del caso d'uso, nonché di gestire lo stato della sessione con il suo attore – ad esempio, per la memorizzazione del "carrello della spesa"
  - le applicazioni web delegano la gestione delle operazioni di sistema a questi bean



Un breve esempio

- Si consideri ad esempio un sistema di commercio elettronico, per la gestione di ordini, fatti da clienti relativamente a prodotti
  - come può essere realizzata con la piattaforma Java EE? (quello che viene descritta è solo una possibilità)
- Livello EJB – session bean stateless
  - è possibile definire un session bean stateless per ciascun "tipo di dati principale" del sistema – tra cui, ad esempio, i session bean GestioneProdottiBean e GestioneOrdiniBean
  - ciascun bean in questo gruppo di bean è responsabile di definire le operazioni di accesso a dati di un certo tipo – nonché di implementare l'accesso ai dati persistenti corrispondenti
  - un bean stateful definito in corrispondenza a un caso d'uso delega (in parte) l'esecuzione delle operazioni di sistema a bean di questo tipo



Discussione

- In questo esempio è possibile vedere l'applicazione di alcuni stili architetturali fondamentali
  - *Client-Server a più livelli* / *Layers* – le applicazioni a componenti sono spesso organizzate in questo modo
  - *Domain Model* – sono stati applicati più tipi di modelli di dominio
    - una modellazione dei casi d'uso
      - ciascun attore è rappresentato da una web-app
      - ciascun caso d'uso, con le sue operazioni di sistema, è rappresentato da un session bean stateful
    - una modellazione delle informazioni
      - ciascun tipo di dati principali, con le operazioni per la sua gestione, è rappresentato da un session bean stateless