



The slide header features a collage of images: on the left, a row of computer monitors with blue light effects; in the center, a network diagram with a globe and several server icons; on the right, the Java EE logo. Below these images, the title 'Enterprise Java Beans (1)' is displayed in a large, bold, black serif font. To the right of the title is the official seal of the University of Salerno, which is circular and contains the text 'HIPPOCRATICA CIVITAS' and 'STUDIVM SALERNITANVM' around a central emblem. At the bottom left, the course information is provided in a smaller black font.

Enterprise Java Beans (1)

Corso di Laurea in Informatica, Programmazione Distribuita
Delfina Malandrino, dmandrino@unisa.it
<http://www.unisa.it/docenti/delfinamalandrino>

1



The slide header has a blue horizontal bar with the number '2' on the left. To the right of the bar, the title 'Organizzazione della lezione' is written in a bold, dark red serif font. Below the title, a list of topics is presented, each preceded by a small square icon. The first icon is red for the main topics, and the subsequent three are blue for sub-topics.

Organizzazione della lezione

- Introduzione agli EJB
- Come sono fatti gli EJB
- Tipi di EJB
 - ▣ Stateless
 - ▣ Stateful
 - ▣ Singleton
- Come usare un EJB
 - ▣ Packaging e deploying
 - ▣ Come invocare EJB
- Conclusioni

2

Organizzazione della lezione

3

- Introduzione agli EJB
- Come sono fatti gli EJB
- Tipi di EJB
 - Stateless
 - Stateful
 - Singleton
- Come usare un EJB
 - Packaging e deploying
 - Come invocare EJB
- Conclusioni

Il ruolo degli EJB

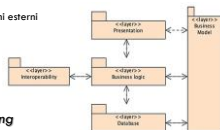
4

- Il layer di persistenza rende facilmente gestibile la memorizzazione, ma non è adatto per business processing
- User interfaces, allo stesso modo, non sono adatte per eseguire logica di business
- La logica di business ha bisogno di un layer dedicato per le caratteristiche proprie

Il ruolo degli EJB

5

- JPA (Data layer) ha modellato i "sostantivi" della nostra architettura, mentre EJB (Business layer) modella i "verbi"
- Il Business Layer ha anche il compito di:
 - ▣ interagire con servizi esterni (SOAP o RESTful web services)
 - ▣ inviare messaggi asincroni (usando JMS)
 - ▣ orchestrare componenti del DB verso sistemi esterni
 - ▣ servire il layer di presentazione

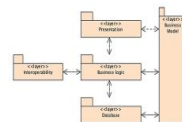


Architecture layering

Enterprise JavaBeans

6

- Componenti lato server che incorporano la logica di business . . .
- . . . gestiscono transazioni e sicurezza
- . . . gestiscono la comunicazione con componenti esterne all'architettura e interne all'architettura
- Orchestrano l'intera architettura
- Tipi di EJB:
 - ▣ Stateless
 - ▣ Stateful
 - ▣ Singleton



Servizi forniti dal Container

7

- Comunicazione remota: client EJB possono invocare metodi remoti per mezzo di protocolli standard
- Iniezione di dipendenze: JMS destination e factories, datasources, altri EJB, come pure altri POJO
- Gestione dello stato (per gli stateful)
- Pooling (efficienza, per gli stateless): creazione di un pool di istanze che possono essere condivise da client multipli
- Ciclo di vita
- Gestione dei messaggi JMS
- Transazioni
- Sicurezza
- Concorrenza
- Interceptor ai metodi
- Invocazione asincrona (senza messaggi)

7

Interazione con il Container

8

- Una volta fatto il deployment, il container offre i servizi, il programmatore si concentra solo sulla logica di business
- Gli EJB sono oggetti managed
- Quando un client invoca un metodo di un EJB, in effetti, invoca un proxy su di esso, frapposto dal container (che lo usa per fornire i servizi)
 - ▣ Chiamata intercettata dal container, in maniera totalmente trasparente al client
- Specifiche EJB Lite permettono di implementare solo una parte delle specifiche
 - ▣ per permettere implementazioni non impegnative per i container provider

8

Le caratteristiche offerte da EJB Lite

9

| Feature | EJB Lite | Full EJB 3.2 |
|--|----------|--------------|
| Session beans (stateless, stateful, singleton) | Yes | Yes |
| No-interface view | Yes | Yes |
| Local interface | Yes | Yes |
| Interceptors | Yes | Yes |
| Transaction support | Yes | Yes |
| Security | Yes | Yes |
| Embeddable API | Yes | Yes |
| Asynchronous calls | No | Yes |
| MDBs | No | Yes |
| Remote interface | No | Yes |
| JAX-WS web services | No | Yes |
| JAX-RS web services | No | Yes |
| Timer service | No | Yes |
| RMI/IIOP interoperability | No | Yes |

9

Organizzazione della lezione

10

- Introduzione agli EJB
- Come sono fatti gli EJB
- Tipi di EJB
 - Stateless
 - Stateful
 - Singleton
- Come usare un EJB
 - Packaging e deploying
 - Come invocare EJB
- Conclusioni

10

Tipi di EJBs

11

□ Un session bean può avere diversi stati

■ Stateless

- Il session bean non contiene conversational state tra i metodi ed ogni istanza può essere usata da ogni client
- Utile per gestire task che possono essere conclusi con una singola method call

■ Stateful

- Il session bean contiene un conversational state che deve essere mantenuto attraverso i metodi per un single user
- Utile per task che devono essere eseguiti in diversi step

■ Singleton

- Un session bean è condiviso da vari client e supporta accessi concorrenti
- Il container deve assicurare che esista una sola istanza per l'intera applicazione

Un semplice EJB Stateless

```
@Stateless
public class BookEJB
{
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

Annotazione che definisce un bean senza stato

Un semplice EJB Stateless

```
@Stateless
public class BookEJB
{
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

> Annotazione che definisce un bean senza stato

> Nome della classe

Un semplice EJB Stateless

```
@Stateless
public class BookEJB
{
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

> Annotazione che definisce un bean senza stato

> Nome della classe

> Iniezione di dipendenza: un EM (per la persistenza)

Un semplice EJB Stateless

```
@Stateless
public class BookEJB
{
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

- > Annotazione che definisce un bean senza stato
- > Nome della classe
- > Iniezione di dipendenza: un EM (per la persistenza)
- > Variabile iniettata

Un semplice EJB Stateless

```
@Stateless
public class BookEJB
{
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

- > Annotazione che definisce un bean senza stato
- > Nome della classe
- > Iniezione di dipendenza: un EM (per la persistenza)
- > Variabile iniettata
- > Un metodo del bean

Un semplice EJB Stateless

```
@Stateless
public class BookEJB
{
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

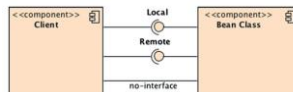
    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

- > Annotazione che definisce un bean senza stato
- > Nome della classe
- > Iniezione di dipendenza: un EM (per la persistenza)
- > Variabile iniettata
- > Un metodo del bean
- > Un altro metodo del bean (che rende persistente un libro)

Anatomia di un EJB

- UN EJB si compone dei seguenti elementi:
 - ▣ Una classe: annotata con `@Stateless`, `@Stateful`, `@Singleton`
 - ▣ Interfacce di business: locale, remota o nessuna (significa solo accesso locale – invocando la classe bean stessa, client ed EJB nello stesso package)



Tipi di business interfaces

Caratteristiche della classe Bean

19

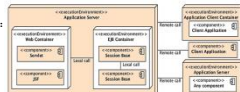
- Una classe session bean è una classe Java standard che implementa la logica di business
- I requirements per implementare un session bean sono i seguenti:
 - Annotata con `@Stateless`, `@Stateful`, `@Singleton` o con l'equivalente nel descrittore XML
 - Deve implementare i metodi delle interfacce (se esistono)
 - Deve essere public e non final o abstract
 - Costruttore pubblico senza parametri
 - Nessun metodo di `finalize()`
 - I metodi non possono iniziare per `ejb` e non possono essere final o static
 - Argomenti e valori di ritorno devono essere tipi legali per RMI

19

Remote, Local e No-interface Views

20

- A seconda di dove un client invoca un session bean, il bean deve implementare una interfaccia remota locale o una no-interface view
- Se l'architettura ha client che risiedono all'esterno dell' EJB container's JVM instance, allora devono usare una interfaccia remota
- Questo si verifica per client in esecuzione:
 - su una JVM separata (a rich client)
 - in un application client container (ACC)
 - in an external web o EJB container
- In questo caso i client invocheranno i metodi del bean attraverso Remote Method Invocation (RMI)

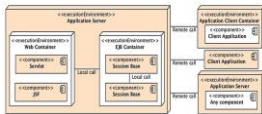


20

Remote, Local e No-interface Views

21

- Si possono usare invocazioni locali quando i bean sono in esecuzione nella stessa JVM
 - Un EJB che invoca un altro EJB o una web component (Servlet, JSF) in esecuzione in un web container nella stessa JVM
- E' possibile usare sia chiamate locali che remote sullo stesso session bean



21

Remote, Local e No-interface Views

22

- Un session bean può implementare diverse interfacce o nessuna
- Le annotazioni
 - @Remote: denota una remote business interface
 - Method parameters passati per valore e serializzati essendo parte del protocollo RMI
 - @Local: denota una local business interface
 - Method parameters passati per riferimento dal client al bean

22

Remote, Local e No-interface Views

23

□ No-interface view

- La vista senza interfaccia è una variante della vista locale (Local interface) che espone tutti i metodi pubblici di business della classe bean localmente senza l'utilizzo di un'interfaccia separata

Le interfacce

24

```

@Local
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCds();
}

@Remote
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCds();
    Book createBook(Book book);
    CD createCD(CD cd);
}

@Stateless
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}

```

Annotazione per interfaccia locale

Le interfacce

```
@Local
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

- > Annotazione per interfaccia locale
- > Dichiarazione interfaccia...

```
@Remote
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

```
@Stateless
public class ItemJ2B implements ItemLocal, ItemRemote {
    //...
}
```

Le interfacce

```
@Local
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

- > Annotazione per interfaccia locale
- > Dichiarazione interfaccia ...
- > ...e suoi metodi

```
@Remote
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

```
@Stateless
public class ItemJ2B implements ItemLocal, ItemRemote {
    //...
}
```

Le interfacce

```
@Local
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

- › Annotazione per interfaccia locale
- › Dichiarazione interfaccia...
- › ...e suoi metodi

```
@Remote
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

- › Annotazione per interfaccia remota

```
@Stateless
public class ItemJ2B implements ItemLocal, ItemRemote {
    //...
}
```

Le interfacce

```
@Local
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

- › Annotazione per interfaccia locale
- › Dichiarazione interfaccia...
- › ...e suoi metodi

```
@Remote
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

- › Annotazione per interfaccia remota
- › Dichiarazione interfaccia...

```
@Stateless
public class ItemJ2B implements ItemLocal, ItemRemote {
    //...
}
```

Le interfacce

```
@Local
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

- > Annotazione per interfaccia locale
- > Dichiarazione interfaccia...
- > ... e suoi metodi

```
@Remote
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

- > Annotazione per interfaccia remota
- > Dichiarazione interfaccia...
- > ... e suoi metodi

```
@Stateless
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

Le interfacce

```
@Local
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

- > Annotazione per interfaccia locale
- > Dichiarazione interfaccia...
- > ... e suoi metodi

```
@Remote
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

- > Annotazione per interfaccia remota
- > Dichiarazione interfaccia...
- > ... e suoi metodi

```
@Stateless
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

- > Utilizzo delle interfacce
(precedentemente annotate)

Metodo alternativo: utile per interfacce legacy

```
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

Dichiarazione interfaccia
(normale)...

```
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

Metodo alternativo: utile per interfacce legacy

```
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

Dichiarazione interfaccia
(normale)...

...e suoi metodi

```
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```


Metodo alternativo: utile per interfacce legacy

```
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

> Dichiarazione interfaccia
(normale)...

```
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

> ...e suoi metodi

> Dichiarazione interfaccia
(normale)...

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

Metodo alternativo: utile per interfacce legacy

```
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

> Dichiarazione interfaccia
(normale)...

> ...e suoi metodi

```
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

> Dichiarazione interfaccia
(normale)...

> ...e suoi metodi

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

Metodo alternativo: utile per interfacce legacy

```
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

> Dichiarazione interfaccia (normale)...

```
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

> ...e suoi metodi

> Dichiarazione interfaccia (normale)...

> ...e suoi metodi

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

> Specifica della dipendenza da interfaccia

35

Metodo alternativo: utile per interfacce legacy

```
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

> Dichiarazione interfaccia (normale)...

> ...e suoi metodi

> Dichiarazione interfaccia (normale)...

> ...e suoi metodi

> Specifica della dipendenza da interfaccia

```
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

> annotation -> bean has a no-interface view

36

Metodo alternativo: utile per interfacce legacy

```
public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

```
public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
    Book createBook(Book book);
    CD createCD(CD cd);
}
```

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {
    //...
}
```

- › Dichiarazione interfaccia (normale)...
 - › ...e suoi metodi
 - › Dichiarazione interfaccia (normale)...
 - › ...e suoi metodi
 - › Specifica della dipendenza da interfacce
 - › annotation -> bean has a no-interface view
- Utilizza delle interfacce (precedentemente annotate)

37

Come identificare un EJB con JNDI

45

- Automaticamente, alla creazione di un EJB viene creato un nome Java Naming and Directory Interface (JNDI)

```
java:<scope>[/<app-name>]/<module-name>/<bean-name>[!<EQ-interface-name>]
```

- **scope** può essere:
 - **global**: the java:global prefix allows a component executing outside a Java EE application to access a global namespace
 - Accesso a bean remoti attraverso JNDI lookups

45

Come identificare un EJB con JNDI

46

- Automaticamente, alla creazione di un EJB viene creato un nome JNDI

```
java:<scope>[/<app-name>]/<module-name>/<bean-name>[!<FQ-interface-name>]
```

- **scope** può essere:

- **app**: the java:app prefix allows a component executing within a Java EE application to access an application-specific namespace
- Look up di local enterprise beans packaged **all'interno della stessa applicazione**

46

Come identificare un EJB con JNDI

47

- Automaticamente, alla creazione di un EJB viene creato un nome JNDI

```
java:<scope>[/<app-name>]/<module-name>/<bean-name>[!<FQ-interface-name>]
```

- **scope** può essere:

- **module**: the java:module prefix allows a component executing within a Java EE application to access a module-specific namespace
- Look up di local enterprise beans **all'interno dello stesso modulo**

47

Come identificare un EJB con JNDI

49

```
java:<scope>[/<app-name>]/<module-name>/<bean-name>[!<FQ-interface-name>]
```

- **app-name**: richiesto solo se il bean viene packaged in un file .ear o .war
- **module-name**: nome del modulo in cui il session bean is packaged
- **bean-name**: nome del session bean
- **fully-qualified-interface-name**: Fully qualified name di ogni interfaccia definita

49

Un esempio

50

Nomi standard

```
java:global/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemRemote
java:global/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemLocal
java:global/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemEJB
```

Se fatto deployment in un .ear

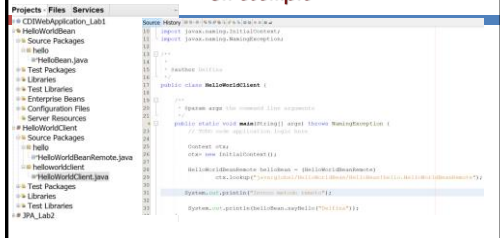
```
java:global/nyapplication/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemRemote
java:global/nyapplication/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemLocal
java:global/nyapplication/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemEJB
```

Accesso via moduli e app

```
java:app/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemRemote
java:app/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemLocal
java:app/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemEJB
java:module/ItemEJB!org.agoncal.book.javaee7.ItemRemote
java:module/ItemEJB!org.agoncal.book.javaee7.ItemLocal
java:module/ItemEJB!org.agoncal.book.javaee7.ItemEJB
```

50

Un esempio



51

Organizzazione della lezione

52

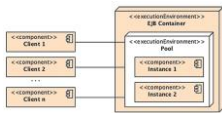
- Introduzione agli EJB
- Come sono fatti gli EJB
- **Tipi di EJB**
 - Stateless
 - Stateful
 - Singleton
- Come usare un EJB
 - Packaging e deploying
 - Come invocare EJB
- Conclusioni

52

Stateless Beans

53

- Il tipo più semplice e popolare di EJB
 - ▢ Quello dove la gestione del container è più efficiente (pooling)
- Stateless: un task viene completato in una singola invocazione di un metodo (nessuna memoria di precedenti interazioni)
- Il container mantiene un pool di EJB Stateless dello stesso tipo, che vengono assegnati a chi li richiede (per la durata della esecuzione del metodo) per poi tornare disponibili
 - ▢ Istanze che possono essere condivise da diversi client
- Un piccolo numero di EJB può servire molti client (e il container può gestire il loro ciclo di vita in maniera autonoma)



*Client che accedono a
stateless beans in un pool*

Esempio di EJB Stateless

54

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public List<Book> findBooks() {
        TypedQuery<Book> query =
            em.createNamedQuery(Book.FIND_ALL, Book.class);
        return query.getResultList();
    }

    public List<CD> findCDs() {
        TypedQuery<CD> query =
            em.createNamedQuery(CD.FIND_ALL, CD.class);
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }

    public CD createCD(CD cd) {
        em.persist(cd);
        return cd;
    }
}

```

Annotazione

Esempio di EJB Stateless

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public List<Book> findBooks() {
        TypedQuery<Book> query =
            em.createNamedQuery(Book.FIND_ALL, Book.class);
        return query.getResultList();
    }

    public List<CD> findCDs() {
        TypedQuery<CD> query =
            em.createNamedQuery(CD.FIND_ALL, CD.class);
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }

    public CD createCD(CD cd) {
        em.persist(cd);
        return cd;
    }
}

```

← Annotazione

← Nome del bean

Esempio di EJB Stateless

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public List<Book> findBooks() {
        TypedQuery<Book> query =
            em.createNamedQuery(Book.FIND_ALL, Book.class);
        return query.getResultList();
    }

    public List<CD> findCDs() {
        TypedQuery<CD> query =
            em.createNamedQuery(CD.FIND_ALL, CD.class);
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }

    public CD createCD(CD cd) {
        em.persist(cd);
        return cd;
    }
}

```

← Annotazione

← Nome del bean

← Iniezione del EM

Esempio di EJB Stateless

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public List<Book> findBooks() {
        TypedQuery<Book> query =
            em.createNamedQuery(Book.FIND_ALL, Book.class);
        return query.getResultList();
    }

    public List<CD> findCDs() {
        TypedQuery<CD> query =
            em.createNamedQuery(CD.FIND_ALL, CD.class);
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }

    public CD createCD(CD cd) {
        em.persist(cd);
        return cd;
    }
}

```

» Annotazione

» Nome del bean

» Iniezione del EM

» Metodo che esegue una query named sui libri (JPA)

Esempio di EJB Stateless

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public List<Book> findBooks() {
        TypedQuery<Book> query =
            em.createNamedQuery(Book.FIND_ALL, Book.class);
        return query.getResultList();
    }

    public List<CD> findCDs() {
        TypedQuery<CD> query =
            em.createNamedQuery(CD.FIND_ALL, CD.class);
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }

    public CD createCD(CD cd) {
        em.persist(cd);
        return cd;
    }
}

```

» Annotazione

» Nome del bean

» Iniezione del EM

» Metodo che esegue una query named sui libri (JPA)

» Metodo che esegue una query named sui CD (JPA)

Esempio di EJB Stateless

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public List<Book> findBooks()
    {
        TypedQuery<Book> query =
            em.createNamedQuery(Book.FIND_ALL, Book.class);
        return query.getResultList();
    }

    public List<CD> findCDs()
    {
        TypedQuery<CD> query =
            em.createNamedQuery(CD.FIND_ALL, CD.class);
        return query.getResultList();
    }

    public Book createBook(Book book)
    {
        em.persist(book);
        return book;
    }

    public CD createCD(CD cd)
    {
        em.persist(cd);
        return cd;
    }
}

```

↳ Annotazione

- › Nome del bean
- › Iniezione del EM
- › Metodo che esegue una query named sui libri (JPA)
- › Metodo che esegue una query named sui CD (JPA)

← Crea un libro

Esempio di EJB Stateless

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter07PU")
    private EntityManager em;

    public List<Book> findBooks()
    {
        TypedQuery<Book> query =
            em.createNamedQuery(Book.FIND_ALL, Book.class);
        return query.getResultList();
    }

    public List<CD> findCDs()
    {
        TypedQuery<CD> query =
            em.createNamedQuery(CD.FIND_ALL, CD.class);
        return query.getResultList();
    }

    public Book createBook(Book book)
    {
        em.persist(book);
        return book;
    }

    public CD createCD(CD cd)
    {
        em.persist(cd);
        return cd;
    }
}

```

↳ Annotazione

- › Nome del bean
- › Iniezione del EM
- › Metodo che esegue una query named sui libri (JPA)
- › Metodo che esegue una query named sui CD (JPA)
- › Crea un libro

← Crea un CD

Organizzazione della lezione

61

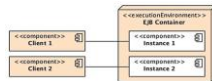
- Introduzione agli EJB
- Come sono fatti gli EJB
- **Tipi di EJB**
 - Stateless
 - **Stateful**
 - Singleton
- Come usare un EJB
 - Packaging e deploying
 - Come invocare EJB
- Conclusioni

61

Stateful Beans

62

- EJB stateless non mantengono stato con i client: ogni client è come se fosse "nuovo" per loro
- EJB stateful mantengono lo stato della conversazione (esempio: il carrello degli acquisti in un negozio di e-commerce)
- Relazione 1-1 con il numero di client: tanti client, tanti EJB (e tanto carico!)
- Per ridurre il carico, tecniche di attivazione e passivazione permettono di serializzare l'EJB su memoria di massa . . .
- . . . e riportarlo attivo quando serve
- Fatto automaticamente dal container, che così permette la scalabilità automaticamente



*Client che accedono a
stateful beans*

62

Esempio di EJB Stateful

62

```

@Stateful
@StatefulTimeout(value = 20, unit = TimeUnit.SECONDS)
public class ShoppingCartEJB {

    private List<Item> cartItems = new ArrayList<>();

    public void addItem(Item item) {
        if(!cartItems.contains(item))
            cartItems.add(item);
    }

    public void removeItem(Item item) {
        if(cartItems.contains(item))
            cartItems.remove(item);
    }

    public Integer getNumberOfItems() {
        if(cartItems == null || cartItems.isEmpty())
            return 0;
        return cartItems.size();
    }

    //...
}

```

› Annotazione EJB stateful

Esempio di EJB Stateful

63

```

@Stateful
@StatefulTimeout(value = 20, unit = TimeUnit.SECONDS)
public class ShoppingCartEJB {

    private List<Item> cartItems = new ArrayList<>();

    public void addItem(Item item) {
        if(!cartItems.contains(item))
            cartItems.add(item);
    }

    public void removeItem(Item item) {
        if(cartItems.contains(item))
            cartItems.remove(item);
    }

    public Integer getNumberOfItems() {
        if(cartItems == null || cartItems.isEmpty())
            return 0;
        return cartItems.size();
    }

    //...
}

```

› Annotazione EJB stateful
› Timeout*: tempo consentito per rimanere idle (con unità di tempo)

* Tempo in cui non si ricevono connessioni client, dopodiché bean rimosso dal container

Esempio di EJB Stateful

64

```

@Stateful
@StatefulTimeout(value = 20, unit = TimeUnit.SECONDS)
public class ShoppingCartEJB {

    private List<Item> cartItems = new ArrayList<>();

    public void addItem(Item item) {
        if(!cartItems.contains(item))
            cartItems.add(item);
    }

    public void removeItem(Item item) {
        if(cartItems.contains(item))
            cartItems.remove(item);
    }

    public Integer getNumberOfItems() {
        if(cartItems == null || cartItems.isEmpty())
            return 0;
        return cartItems.size();
    }

    //...

```

- › Annotazione EJB stateful
- › Timeout*: tempo consentito per rimanere idle (con unità di tempo)
- › Struttura dati che mantiene lo stato

Esempio di EJB Stateful

65

```

@Stateful
@StatefulTimeout(value = 20, unit = TimeUnit.SECONDS)
public class ShoppingCartEJB {

    private List<Item> cartItems = new ArrayList<>();

    public void addItem(Item item) {
        if(!cartItems.contains(item))
            cartItems.add(item);
    }

    public void removeItem(Item item) {
        if(cartItems.contains(item))
            cartItems.remove(item);
    }

    public Integer getNumberOfItems() {
        if(cartItems == null || cartItems.isEmpty())
            return 0;
        return cartItems.size();
    }

    //...

```

- › Annotazione EJB stateful
- › Timeout*: tempo consentito per rimanere idle (con unità di tempo)
- › Struttura dati che mantiene lo stato
- › Metodo per aggiungere un elemento al carrello ...

65

66

Esempio di EJB Stateful

66

```

@Stateful
@StatefulTimeout(value = 20, unit = TimeUnit.SECONDS)
public class ShoppingCartEJB {

    private List<Item> cartItems = new ArrayList<>();

    public void addItem(Item item) {
        if(!cartItems.contains(item))
            cartItems.add(item);
    }

    public void removeItem(Item item) {
        if(cartItems.contains(item))
            cartItems.remove(item);
    }

    public Integer getNumberOfItems() {
        if(cartItems == null || cartItems.isEmpty())
            return 0;
        return cartItems.size();
    }

    //...

```

- > Annotazione EJB stateful
- > Timeout*: tempo consentito per rimanere idle (con unità di tempo)
- > Struttura dati che mantiene lo stato
- > Metodo per aggiungere un elemento al carrello ...
- > ... e per rimuoverlo

Esempio di EJB Stateful

67

```

@Stateful
@StatefulTimeout(value = 20, unit = TimeUnit.SECONDS)
public class ShoppingCartEJB {

    private List<Item> cartItems = new ArrayList<>();

    public void addItem(Item item) {
        if(!cartItems.contains(item))
            cartItems.add(item);
    }

    public void removeItem(Item item) {
        if(cartItems.contains(item))
            cartItems.remove(item);
    }

    public Integer getNumberOfItems() {
        if(cartItems == null || cartItems.isEmpty())
            return 0;
        return cartItems.size();
    }

    //...

```

- > Annotazione EJB stateful
- > Timeout*: tempo consentito per rimanere idle (con unità di tempo)
- > Struttura dati che mantiene lo stato
- > Metodo per aggiungere un elemento al carrello ...
- > ... e per rimuoverlo
- > ... per sapere quanti sono

67

68

Esempio di EJB Stateful

68

```
//...
public Float getTotal() {
    if(cartItems == null || cartItems.isEmpty())
        return 0f;
    Float total = 0f;
    for(Item cartItem : cartItems) { total +=
        (cartItem.getPrice());
    }
    return total;
}

public void empty() {
    cartItems.clear();
}

@Remove
public void checkout() {
    //Do some business logic
    cartItems.clear();
}
}
```

Fa il totale degli elementi nel carrello

Esempio di EJB Stateful

69

```
//...
public Float getTotal() {
    if(cartItems == null || cartItems.isEmpty())
        return 0f;
    Float total = 0f;
    for(Item cartItem : cartItems) { total +=
        (cartItem.getPrice());
    }
    return total;
}

public void empty() {
    cartItems.clear();
}

@Remove
public void checkout() {
    //Do some business logic
    cartItems.clear();
}
}
```

Fa il totale degli elementi nel carrello

Svuota il carrello

69

70

Esempio di EJB Stateful

70

```
//...
public Float getTotal()
{
    if(cartItems == null || cartItems.isEmpty())
        return 0f;
    Float total = 0f;
    for(Item cartItem : cartItems) { total +=
        (cartItem.getPrice());
    }
    return total;
}

public void empty()
{
    cartItems.clear();
}

@Remove
public void checkout() {
    //Do some business logic
    cartItems.clear();
}
}
```

> Fa il totale degli elementi nel carrello

> Svuota il carrello

Metodo invocato prima della rimozione

Esempio di EJB Stateful

71

```
//...
public Float getTotal()
{
    if(cartItems == null || cartItems.isEmpty())
        return 0f;
    Float total = 0f;
    for(Item cartItem : cartItems) { total +=
        (cartItem.getPrice());
    }
    return total;
}

public void empty()
{
    cartItems.clear();
}

@Remove
public void checkout() {
    //Do some business logic
    cartItems.clear();
}
}
```

> Fa il totale degli elementi nel carrello

> Svuota il carrello

> Metodo invocato prima della rimozione

Cancellazione del carrello, per il checkout

71

72

Organizzazione della lezione

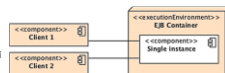
73

- Introduzione agli EJB
- Come sono fatti gli EJB
- **Tipi di EJB**
 - Stateless
 - Stateful
 - **Singleton**
- Come usare un EJB
 - Packaging e deploying
 - Come invocare EJB
- Conclusioni

Design pattern singleton

74

- Session bean istanziato una sola volta per applicazione
- Utile in diversi contesti
 - Ad esempio, se si vuole gestire una cache di oggetti (HashMap), per tutta l'applicazione



Client che accedono ad un singleton bean

73

74

Design pattern singleton

75

- In un'application managed environment bisogna fare diverse modifiche per trasformare un POJO in un bean singleton
 - Prevenire che qualcuno crei altre cache
 - usando un costruttore privato
 - Per ottenere una istanza, si deve avere un metodo (sincronizzato) che permette di ottenere la cache
 - Il metodo statico `getInstance()` restituisce una singola istanza della classe `CacheSingleton` class
 - se un client vuole aggiungere un oggetto alla cache deve invocare
 - `CacheSingleton.getInstance().addToCache(myobject)`

Metodo thread-safe garantendo sincronizzazione

75

Esempio di POJO che segue il Singleton Design pattern

Nome della classe

```

public class Cache {
    private static Cache instance = new Cache();
    private Map<Long, Object> cache = new HashMap<>();

    private Cache() {
    }

    public static synchronized Cache getInstance() {
        return instance;
    }

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}

```

76

Esempio di POJO che segue il Singleton Design pattern

> Nome della classe

```
public class Cache {
    private static Cache instance = new Cache();
    private Map<Long, Object> cache = new HashMap<>();

    private Cache() {
    }

    public static synchronized Cache getInstance() {
        return instance;
    }

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

Istanza privata e static (inizializzata, in maniera thread-safe a caricamento della classe nella JVM)

77

Esempio di POJO che segue il Singleton Design pattern

> Nome della classe

```
public class Cache {
    private static Cache instance = new Cache();
    private Map<Long, Object> cache = new HashMap<>();

    private Cache() {
    }

    public static synchronized Cache getInstance() {
        return instance;
    }

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

Istanza privata e static (inizializzata, in maniera thread-safe a caricamento della classe nella JVM)

Costruttore privato (nessuno può istanziare un' (altra) Cache)

78

Esempio di POJO che segue il Singleton Design pattern

```

public class Cache {
    private static Cache instance = new Cache();
    private Map<Long, Object> cache = new HashMap<>();

    private Cache() {}
    public static synchronized Cache getInstance() {
        return instance;
    }

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}

```

Nome della classe

Istanza privata e static (inizializzata, in maniera thread-safe a caricamento della classe nella JVM)

Costruttore privato (nessuno può istanziare un' (altra) Cache)

Metodo sincronizzato per restituire l'istanza

79

Esempio di POJO che segue il Singleton Design pattern

```

public class Cache {
    private static Cache instance = new Cache();
    private Map<Long, Object> cache = new HashMap<>();

    private Cache() {}
    public static synchronized Cache getInstance() {
        return instance;
    }

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}

```

Nome della classe

Istanza privata e static (inizializzata, in maniera thread-safe a caricamento della classe nella JVM)

Costruttore privato (nessuno può istanziare un' (altra) Cache)

Metodo sincronizzato per restituire l'istanza

Aggiunge un oggetto

80

Esempio di POJO che segue il Singleton Design pattern

```
public class Cache {
    private static Cache instance = new Cache();
    private Map<Long, Object> cache = new HashMap<>();

    private Cache() {}

    public static synchronized Cache getInstance() {
        return instance;
    }

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

Nome della classe

Istanza privata e static (inizializzata, in maniera thread-safe a caricamento della classe nella JVM)

Costruttore privato (nessuno può istanziare un' (altra) Cache)

Metodo sincronizzato per restituire l'istanza

Aggiunge un oggetto

Rimuove un oggetto

81

Esempio di POJO che segue il Singleton Design pattern

```
public class Cache {
    private static Cache instance = new Cache();
    private Map<Long, Object> cache = new HashMap<>();

    private Cache() {}

    public static synchronized Cache getInstance() {
        return instance;
    }

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

Nome della classe

Istanza privata e static (inizializzata, in maniera thread-safe a caricamento della classe nella JVM)

Costruttore privato (nessuno può istanziare un' (altra) Cache)

Metodo sincronizzato per restituire l'istanza

Aggiunge un oggetto

Rimuove un oggetto

Cerca un elemento

82

Come rendere un POJO un EJB Singleton

82

```
@Singleton
public class CacheEJB {
    private Map<Long, Object> cache = new HashMap<>();

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

↳ Annotazione

83

Come rendere un POJO un EJB Singleton

83

```
@Singleton
public class CacheEJB {
    private Map<Long, Object> cache = new HashMap<>();

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

↳ Annotazione
↳ Nome della classe

84

Come rendere un POJO un EJB Singleton

84

```
@Singleton
public class CacheEJB {
    private Map<Long, Object> cache = new HashMap<>();

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

- › Annotazione
- › Nome della classe
- › Creazione oggetto privato

85

Come rendere un POJO un EJB Singleton

85

```
@Singleton
public class CacheEJB {
    private Map<Long, Object> cache = new HashMap<>();

    public void addToCache(Long id, Object object) {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id) {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

- › Annotazione
- › Nome della classe
- › Creazione oggetto privato
- › Aggiunge un oggetto in cache

86

Come rendere un POJO un EJB Singleton

86

```
@Singleton
public class CacheEJB {
    private Map<Long, Object> cache = new HashMap<>();

    public void addToCache(Long id, Object object)
    {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id)
    {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

- › Annotazione
- › Nome della classe
- › Creazione oggetto privato
- › Aggiunge un oggetto in cache
- › Rimuove un oggetto dalla cache

87

Come rendere un POJO un EJB Singleton

87

```
@Singleton
public class CacheEJB {
    private Map<Long, Object> cache = new HashMap<>();

    public void addToCache(Long id, Object object)
    {
        if(!cache.containsKey(id))
            cache.put(id, object);
    }

    public void removeFromCache(Long id) {
        if(cache.containsKey(id))
            cache.remove(id);
    }

    public Object getFromCache(Long id)
    {
        if(cache.containsKey(id))
            return cache.get(id);
        else
            return null;
    }
}
```

- › Annotazione
- › Nome della classe
- › Creazione oggetto privato
- › Aggiunge un oggetto in cache
- › Rimuove un oggetto dalla cache
- › Cerca un elemento in cache

88

Organizzazione della lezione

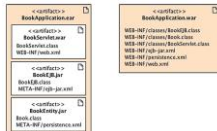
89

- Introduzione agli EJB
- Come sono fatti gli EJB
- Tipi di EJB
 - Stateless
 - Stateful
 - Singleton
- Come usare un EJB
 - Packaging e deploying
 - Come invocare EJB
- Conclusioni

EJB sul server

90

- Necessario il packaging per porre gli EJB in un container
- Necessario mettere insieme: classi EJB, interfacce EJB, superclassi/superinterfacce, eccezioni, classi ausiliare, e un deployment descriptor opzionale ejb-jar.xml
- Il file si chiama Enterprise Archive (EAR)
- Un file EAR raggruppa in maniera coerente EJB che hanno necessità di essere deployati insieme



Organizzazione della lezione

91

- Introduzione agli EJB
- Come sono fatti gli EJB
- Tipi di EJB
 - ▣ Stateless
 - ▣ Stateful
 - ▣ Singleton
- Come usare un EJB
 - ▣ Packaging e deploying
 - ▣ Come invocare EJB
- Conclusioni

91

L'invocazione di EJB da diverse componenti

91

- Il client di un EJB può essere di diversi tipi: un POJO, un client grafico, un CDI Managed Bean, un Servlet, un Bean JSF, un Web Service (SOAP o REST) o un altro EJB
- Il client NON può (ovviamente) fare `new ()` su un EJB: ha bisogno di un riferimento, che può essere:
 - ▣ iniettato via `@EJB` o `@Inject`
 - ▣ acceduto via un lookup JNDI

92

Invocazione con iniezione del riferimento

92

- Se il bean è del tipo "no-interface", allora il client deve solo ottenere un riferimento alla classe bean stessa
 - ▣ Attraverso l'annotazione @EJB

```
@Stateless
public class ItemEJB {...}

// Client code injecting a reference to the EJB
@EJB ItemEJB itemEJB;
```

93

Invocazione con iniezione del riferimento

93

- Se ci sono diverse interfacce, bisogna specificare quella alla quale ci si vuole riferire

```
@Stateless
@Remote(ItemRemote.class)
@Local(ItemLocal.class)
@LocalBean
public class ItemEJB implements ItemLocal, ItemRemote {...}

// Client code injecting several references to the EJB or interfaces
@EJB ItemEJB itemEJB;
@EJB ItemLocal itemEJBLocal;
@EJB ItemRemote itemEJBRemote;
```

94

Invocazione con iniezione del riferimento

94

- Se l'EJB è remoto, si può specificare il nome JNDI
- L'`@EJB` API ha diversi attributi, uno di questi è il nome JNDI dell' EJB che vogliamo iniettare
- Utile per remote EJBs in esecuzione su un server differente:

```
//...
@EJB(lookup = "java:global/classes/ItemEJB") ItemRemote itemEJBRemote;
```

95

Invocazione con iniezione del riferimento

95

- Possibile usare le `@Inject` generica di CDI al posto di `@EJB` ma in tal caso non si può passare la stringa di lookup (non consentita per CDI) e bisogna produrre il remote EJB da iniettare:

```
// Code producing a remote EJB
@Produces @EJB(lookup = "java:global/classes/ItemEJB") ItemRemote itemEJBRemote;

// Client code injecting the produced remote EJB
@Inject ItemRemote itemEJBRemote;
```

96

Invocazione diretta di JNDI

96

- JNDI è usato di solito per accesso remoto
- Ma anche per accesso locale: in questa maniera si evita il costoso resource injection
 - ▣ si chiedono dati quando servono, invece di farcene fare il push anche se non poi non dovessero servire
- Si usa il contesto iniziale di JNDI (settabile da parametri su linea di comando) per effettuare la query con il nome globale standard JNDI

```
//...
Context ctx = new InitialContext();
ItemRemote itemEJB = (ItemRemote)
    ctx.lookup("java:global/cdbookstore/ItemEJB!org.agoncal.book.javaee7.ItemRemote");
```

97

Organizzazione della lezione

98

- Introduzione agli EJB
- Come sono fatti gli EJB
- Tipi di EJB
 - ▣ Stateless
 - ▣ Stateful
 - ▣ Singleton
- Come usare un EJB
 - ▣ Packaging e deploying
 - ▣ Come invocare EJB
- Conclusioni

98