

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

## Enterprise Java Beans (2)

Delfina Malandrino  
[dmandrino@unisa.it](mailto:dmandrino@unisa.it)  
<http://www.unisa.it/docenti/delfinamalandrino>

### Organizzazione della lezione

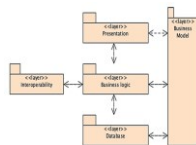
- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Organizzazione della lezione

- Introduzione agli EJB
  - Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Il ruolo degli EJB

- Il layer di persistenza rende facilmente gestibile la memorizzazione
- La logica di business ha bisogno di un layer dedicato per le caratteristiche proprie
- 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, inviare messaggi asincroni, orchestrare componenti del DB verso sistemi esterni, servire il layer di presentazione



## Enterprise JavaBeans

- 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 la intera architettura
- Tipi di EJB:
  - Stateless
  - Stateful
  - Singleton

## Servizi forniti dal Container a EJB

- Comunicazione verso EJB da remoto
- Iniezione di dipendenze
- Gestione dello stato (per gli stateful)
- Pooling (efficienza, per gli stateless)
- Ciclo di vita
- Gestione dei messaggi JMS
- Transazioni
- Sicurezza
- Concorrenza
- Interceptor ai metodi
- Invocazione asincrona (senza messaggi)

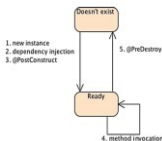
## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## La gestione degli EJB

- Il ciclo di vita degli EJB è gestito dal container
  - un client non può creare una istanza di un session bean con un `new`
- I riferimenti vengono ottenuti attraverso la iniezione di dipendenze o attraverso lookup JNDI
- I bean passano almeno attraverso: creazione e distruzione
- Quelli Stateful passano attraverso gli stati di attivazione e passivazione
- Possibile intercettare metodi annotati per essere eseguiti prima o dopo certi passaggi di stato

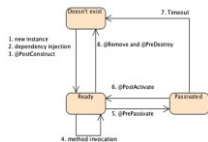
## Le transizioni per Stateless (e Singleton)



Stateless e Singleton bean life cycle

1. Il ciclo di vita inizia quando il client richiede un riferimento al bean. Il container crea una nuova istanza
2. Alla creazione, vengono anche iniettate le dipendenze richieste
3. Se l'istanza appena creata ha un metodo annotato con `@PostConstruct` il container lo invoca
4. Invocazioni dei metodi da parte dei client
5. Il container invoca il metodo annotato con `@PreDestroy`, se esiste, e termina il ciclo di vita della istanza del bean

## Le transizioni per Stateful



Stateful bean life cycle

1. Il ciclo di vita inizia quando il client richiede un riferimento al bean. Il container crea una nuova istanza e la memorizza in memoria
2. Alla creazione, vengono anche iniettate le dipendenze richieste
3. Se l'istanza appena creata ha un metodo annotato con `@PostConstruct` il container lo invoca
4. Invocazioni dei metodi da parte dei client
5. Se il client resta idle per un certo periodo di tempo, il container invoca il metodo annotato con `@PrePassivate` e rende passivo il bean in uno storage permanente
6. Se un client invoca un bean ipassivato, il container lo attiva riportandolo in memoria ed invoca il metodo annotato con `@PostActivate`, se esiste
7. Se un client non invoca il bean all'interno del session timeout period, il container lo elimina
8. Alternativamente al passo 7, se un client chiama un metodo annotato con `@Remove`, il container invocherà il metodo annotato con `@PreDestroy`, se esiste, e termina il ciclo di vita del bean

## Annotazioni di Callback

Annotation	Description
<code>@PostConstruct</code>	Marks a method to be invoked immediately after you create a bean instance and the container does dependency injection. This annotation is often used to perform any initialization.
<code>@PreDestroy</code>	Marks a method to be invoked immediately before the container destroys the bean instance. The method annotated with <code>@PreDestroy</code> is often used to release resources that had been previously initialized. With stateful beans, this happens after timeout or when a method annotated with <code>@Remove</code> has been completed.
<code>@PrePassivate</code>	Stateful beans only. Marks a method to be invoked before the container passivates the instance. It usually gives the bean the time to prepare for serialization and to release resources that cannot be serialized (e.g., it closes connections to a database, a message broker, a network socket, etc.).
<code>@PostActivate</code>	Stateful beans only. Marks a method to be invoked immediately after the container reactivates the instance. Gives the bean a chance to reinitialize resources that had been closed during passivation.

## Un esempio di callback per stateful EJB

```

@Stateful
public class ShoppingCartEJB {
    @Resource(lookup = "java:comp/defaultDataSource")
    private DataSource ds;
    private Connection connection;
    private List<Item> cartItems = new ArrayList<>();

    @PostConstruct
    @PostActivate
    private void init() {
        connection = ds.getConnection();
    }

    @PreDestroy
    @PrePassivate
    private void close() {
        connection.close();
    }

    //...

    @Remove
    public void checkout() {
        cartItems.clear();
    }
}

```

← Risorsa da condividere: accesso a DB

## Un esempio di callback per stateful EJB

```
@Stateful
public class ShoppingCartEJB {
    @Resource(lookup = "java:comp/defaultDataSource")
    private DataSource ds;
    private Connection connection;
    private List<Item> cartItems = new ArrayList<>();

    @PostConstruct
    @PostActivate
    private void init() {
        connection = ds.getConnection();
    }

    @PreDestroy
    @PrePassivate
    private void close() {
        connection.close();
    }

    //...

    @Remove
    public void checkout() {
        cartItems.clear();
    }
}
```

› Risorsa da condividere: accesso a DB

› Iniettata dal container e condivisa

## Un esempio di callback per stateful EJB

```
@Stateful
public class ShoppingCartEJB {
    @Resource(lookup = "java:comp/defaultDataSource")
    private DataSource ds;
    private Connection connection;
    private List<Item> cartItems = new ArrayList<>();

    @PostConstruct
    @PostActivate
    private void init() {
        connection = ds.getConnection();
    }

    @PreDestroy
    @PrePassivate
    private void close() {
        connection.close();
    }

    //...

    @Remove
    public void checkout() {
        cartItems.clear();
    }
}
```

› Risorsa da condividere: accesso a DB

› Iniettata dal container e condivisa

› Il container chiamerà il metodo  
@PostConstruct (init()), che crea una  
database connection

## Un esempio di callback per stateful EJB

```
@Stateful
public class ShoppingCartEJB {
    @Resource(lookup = "java:comp/defaultDataSource")
    private DataSource ds;
    private Connection connection;
    private List<Item> cartItems = new ArrayList<>();

    @PostConstruct
    @PostActivate
    private void init() {
        connection = ds.getConnection();
    }

    @PreDestroy
    @PrePassivate
    private void close() {
        connection.close();
    }

    //...

    @Remove
    public void checkout() {
        cartItems.clear();
    }
}
```

› Risorsa da condividere: accesso a DB

› Iniettata dal container e condivisa

› Il container chiamerà il metodo  
@PostConstruct (init()), che crea una  
database connection

Se il container rende passiva l'istanza, il  
metodo close() verrà invocato  
(@PrePassivate). Scopo: chiudere la  
connessione JDBC connection (non  
necessaria durante la passivation)

## Un esempio di callback per stateful EJB

```
@Stateful
public class ShoppingCartEJB {
    @Resource(lookup = "java:comp/defaultDataSource")
    private DataSource ds;
    private Connection connection;
    private List<Item> cartItems = new ArrayList<>();

    @PostConstruct
    @PostActivate
    private void init() {
        connection = ds.getConnection();
    }

    @PreDestroy
    @PrePassivate
    private void close() {
        connection.close();
    }

    //...

    @Remove
    public void checkout() {
        cartItems.clear();
    }
}
```

› Risorsa da condividere: accesso a DB

› Iniettata dal container e condivisa

› Il container chiamerà il metodo @PostConstruct  
(init()), che crea una database connection

› Se il container rende passiva l'istanza, il metodo  
close() verrà invocato (@PrePassivate). Scopo:  
chiudere la connessione JDBC connection (non  
necessaria durante la passivation)

› Quando un client invoca il metodo checkout()  
(annotato con @Remove), il container rimuove  
l'istanza, ma prima chiamerà di nuovo il metodo  
close() (@PreDestroy).



## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Il modello di sicurezza

- Obiettivo è controllare l'accesso al codice di business
- L'autenticazione è gestita dal layer di presentazione (web) o dall'applicazione client ...
- ... che poi passano l'utente autenticato al layer EJB, ...
- ... che deve verificare se ha accesso possibile al metodo, basato sul suo ruolo
- Autorizzazione può essere fatta in maniera dichiarativa (dall'EJB container) oppure da programma, attraverso Java Authentication and Authorization Service (JAAS API)

## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Autorizzazione Dichiarativa

- Può essere definita usando annotations oppure nel XML deployment descriptor
- Richiede di:
  - definire ruoli
  - assegnare permessi ai metodi (o all'intero bean)
  - o cambiare temporaneamente una security identity

Annotation	Bean	Method	Description
@PermitAll	X	X	Indicates that the given method (or the entire bean) is accessible by everyone (all roles are permitted).
@DenyAll	X	X	Indicates that no role is permitted to execute the specified method or all methods of the bean (all roles are denied). This can be useful if you want to deny access to a method in a certain environment (e.g., the method <code>launchMacLaunch(){} should only be allowed in production but not in a test environment).</code>
@RolesAllowed	X	X	Indicates that a list of roles is allowed to execute the given method (or the entire bean).
@DeclareRoles	X		Defines roles for security checking.
@RunAs	X		Temporarily assigns a new role to a principal.

## Un esempio di autorizzazione su un EJB

```

@Stateless
@RolesAllowed({"user", "employee", "admin"})
public class ItemDB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

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

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

    @RolesAllowed("admin")
    public void deleteBook(Book book) {
        em.remove(em.merge(book));
    }
}

```

EJB stateless

## Un esempio di autorizzazione su un EJB

```

@Stateless
@RolesAllowed({"user", "employee", "admin"})
public class ItemDB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

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

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

    @RolesAllowed("admin")
    public void deleteBook(Book book) {
        em.remove(em.merge(book));
    }
}

```

EJB stateless

Definizione dei ruoli permessi su tutti i metodi dell'EJB

## Un esempio di autorizzazione su un EJB

```
@Stateless
@RolesAllowed({ "user", "employee", "admin" })
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

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

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

    @RolesAllowed("admin")
    public void deleteBook(Book book) {
        em.remove(em.merge(book));
    }
}
```

- > EJB stateless
- > Definizione dei ruoli permessi su tutti i metodi dell'EJB
- > Metodo accessibile per default come definito per il bean

## Un esempio di autorizzazione su un EJB

```
@Stateless
@RolesAllowed({ "user", "employee", "admin" })
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

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

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

    @RolesAllowed("admin")
    public void deleteBook(Book book) {
        em.remove(em.merge(book));
    }
}
```

- > EJB stateless
- > Definizione dei ruoli permessi su tutti i metodi dell'EJB
- > Metodo accessibile per default come definito per il bean
- > ...idem

## Un esempio di autorizzazione su un EJB

```
@Stateless
@RolesAllowed({ "user", "employee", "admin" })
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

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

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

    @RolesAllowed("admin")
    public void deleteBook(Book book) {
        em.remove(em.merge(book));
    }
}
```

- > EJB stateless
- > Definizione dei ruoli permessi su tutti i metodi dell'EJB
- > Metodo accessibile per default come definito per il bean
- > ...idem
- > Questo invece è definito solo per gli admin

## Un esempio di autorizzazione su un EJB

```
@Stateless
@RolesAllowed({ "user", "employee", "admin" })
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

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

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

    @RolesAllowed("admin")
    public void deleteBook(Book book) {
        em.remove(em.merge(book));
    }
}
```

- > EJB stateless
- > Definizione dei ruoli permessi su tutti i metodi dell'EJB
- > Metodo accessibile per default come definito per il bean
- > ...idem
- > Questo invece è definito solo per gli admin
- > ... e permette la cancellazione

## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - **Autorizzazione da programma**
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Programmatic Authorization

- Permette una grana più fine nel controllo:
  - ad esempio per blocchi di programma, oppure per uno specifico utente
- Si usano direttamente le API di Java Authentication and Authorization Service (JAAS)
- Interfaccia `java.security.Principal` insieme al contesto del JavaBean
- L'interfaccia `SessionContext` offre metodi per verificare se chi chiama il metodo ha un certo ruolo, oppure verificare l'utente stesso

## Un esempio

```
@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

    @Resource
    private SessionContext ctx;

    public void deleteBook(Book book) {
        if(!ctx.isCallerInRole("admin"))
            throw new SecurityException("Only admins are allowed");
        em.remove(em.merge(book));
    }

    public Book createBook(Book book) {
        if(ctx.isCallerInRole("employee"))
            if(ctx.isCallerInRole("admin")) {
                book.setCreatedBy("employee only");
            } else if(ctx.getCallerPrincipal().
                getName().equals("paul")) {
                book.setCreatedBy("special user");
            }
        em.persist(book);
        return book;
    }
}
```

Contesto, iniettato dal container

## Un esempio

```
@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

    @Resource
    private SessionContext ctx;

    public void deleteBook(Book book) {
        if(!ctx.isCallerInRole("admin"))
            throw new SecurityException("Only admins are allowed");
        em.remove(em.merge(book));
    }

    public Book createBook(Book book) {
        if(ctx.isCallerInRole("employee"))
            if(ctx.isCallerInRole("admin")) {
                book.setCreatedBy("employee only");
            } else if(ctx.getCallerPrincipal().
                getName().equals("paul")) {
                book.setCreatedBy("special user");
            }
        em.persist(book);
        return book;
    }
}
```

Contesto, iniettato dal container

Nel metodo viene controllato se il metodo viene eseguito da qualcuno che non ha ruolo "admin" (ma si può fare per blocchi di codice)

## Un esempio

```
@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

    @Resource
    private SessionContext ctx;

    public void deleteBook(Book book) {
        if(!ctx.isCallerInRole("admin"))
            throw new SecurityException("Only admins are allowed");
        em.remove(em.merge(book));
    }

    public Book createBook(Book book) {
        if(ctx.isCallerInRole("employee"))
            // ctx.isCallerInRole("admin") {
            book.setCreatedBy("employee only");
        } else if(ctx.getCallerPrincipal().
            getName().equals("paul")) {
            book.setCreatedBy("special user");
        }
        em.persist(book);
        return book;
    }
}
```

Contesto, iniettato dal container

Nel metodo viene controllato se il metodo viene eseguito da qualcuno che non ha ruolo "admin" (ma si può fare per blocchi di codice)

Si setta un campo a seconda del ruolo dell'utente (employee ma non admin)

## Un esempio

```
@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter08PU")
    private EntityManager em;

    @Resource
    private SessionContext ctx;

    public void deleteBook(Book book) {
        if(!ctx.isCallerInRole("admin"))
            throw new SecurityException("Only admins are allowed");
        em.remove(em.merge(book));
    }

    public Book createBook(Book book) {
        if(ctx.isCallerInRole("employee"))
            // ctx.isCallerInRole("admin") {
            book.setCreatedBy("employee only");
        } else if(ctx.getCallerPrincipal().
            getName().equals("paul")) {
            book.setCreatedBy("special user");
        }
        em.persist(book);
        return book;
    }
}
```

Contesto, iniettato dal container

Nel metodo viene controllato se il metodo viene eseguito da qualcuno che non ha ruolo "admin" (ma si può fare per blocchi di codice)

Si setta un campo a seconda del ruolo dell'utente (employee ma non admin)

Oppure se è un utente specifico



## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Le transazioni alla base del Business

- I dati sono cruciali per il business e vanno mantenuti in maniera coerente
  - ad esempio: quando si fa un bonifico bancario, i soldi dal conto di uno vanno inseriti nel conto dell'altro, come una sola operazione
- Una transazione rappresenta un insieme di operazioni logiche che devono essere realizzate come una singola unità di lavoro
- Le operazioni vanno ovviamente realizzate separatamente, e se vanno tutte a buon fine, la transazione va a buon fine (commit) ...
- ... altrimenti si riporta la situazione a prima che la transazione iniziasse (rollback)

## Le proprietà ACID

Atomicity, Consistency, Isolation, Durability: ACID

Property	Description
Atomicity	A transaction is composed of one or more operations grouped in a unit of work. At the conclusion of the transaction, either these operations are all performed successfully (commit) or none of them is performed at all (rollback) if something unexpected or irrecoverable happens.
Consistency	At the conclusion of the transaction, the data are left in a consistent state.
Isolation	The intermediate state of a transaction is not visible to external applications.
Durability	Once the transaction is committed, the changes made to the data are visible to other applications.

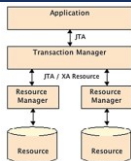
## Transazioni locali in JTA



- Esempio semplice di applicazione su una risorsa singola (DB)
  - Transaction Manager (TM): componente core che gestisce le operazioni, informa il RM chi partecipa nella transazione
  - Resource Manager (RM): responsabile della gestione delle risorse e della loro registrazione con il transaction manager
    - ad esempio un driver, risorsa JMS, etc.
  - Risorsa: persistent storage da cui leggere o in cui scrivere
    - DB (ma anche destinazione di un messaggio etc.)

*Una transazione che coinvolge una risorsa*

## Transazioni distribuite



- Quando ci sono più risorse coinvolte, ad esempio diversi DB
- Coordinamento usando Extended Architecture (XA) per Distributed Transaction Processing
- JTA supporta XA (eterogeneità vendor)
- Tecnica del two-phase commit: step di preparazione

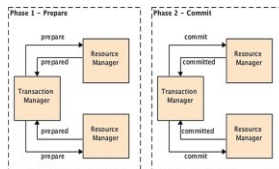
*Una transazione XA che coinvolge due risorse*

## Transazioni distribuite

- Supponiamo che si debba effettuare il trasferimento di denaro da un conto ad un altro
  - Prelievo di 1000 euro dal primo conto: transazione completata con successo
  - Accredito sul secondo conto: la transazione fallisce
  - E' necessario annullare la transazione sul conto che è stato addebitato
- Per evitare questo tipo di inconsistenza si usa il two-phase commit

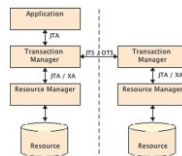
## Two phase commit

- Durante la fase 1, ogni Resource Manager viene notificato attraverso un comando **"prepare"** che un commit sta per essere eseguito
- Se tutti dichiarano che sono **prepared**, la transazione può procedere ed ai resource managers viene chiesto di fare commit nella seconda fase



Two-phase commit

## Transazioni distribuite attraverso la rete



- Applicazioni diverse su network che condividono la transazione
- Collaborazione attraverso le specifiche Java Transaction Service / Object Transaction Service (IIOP)
- I TM lavorano insieme per assicurare la two-phase commit

Una transazione XA distribuita

## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Ci pensa il Container

- Delega completa della gestione delle transazioni al container
- Il programmatore non sviluppa codice JTA
- I servizi vengono forniti ai session beans
- Gli EJB sono per loro natura transazionali: *configuration by exception*

## Un esempio

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Inject
    private InventoryEJB inventory;

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

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

```

Bean stateless

## Un esempio

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Inject
    private InventoryEJB inventory;

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

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

```

Bean stateless  
EM iniettato

## Un esempio

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Inject
    private InventoryEJB inventory;

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

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

```

- > Bean stateless
- > EM iniettato
- > Un altro EJB iniettato

## Un esempio

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Inject
    private InventoryEJB inventory;

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

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

```

- > Bean stateless
- > EM iniettato
- > Un altro EJB iniettato
- > Metodo per trovare libri

## Un esempio

```

@Stateless
public class ItemEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Inject
    private InventoryEJB inventory;

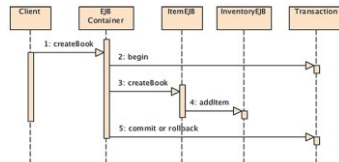
    public List<Book> findBooks() {
        TypedQuery<Book> query =
            em.createQuery(FIND_ALL, Book.class);
        return query.getResultList();
    }

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

```

- > Bean stateless
- > EM iniettato
- > Un altro EJB iniettato
- > Metodo per trovare libri
- > Metodo per creare un libro

Le transazioni: ecco cosa accade quando si invoca il metodo `createBook(Book book)`



*Il container gestisce la transazione*



## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni

## Un esempio

EJB stateless

```

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class InventoryEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public void oneItemSold(Item item) {
        try {
            ut.begin();
            item.decreaseAvailableStock();
            sendShippingMessage();
            if (inventoryLevel(item) == 0)
                ut.rollback();
            else
                ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
        sendInventoryAlert();
    }
}
  
```

## Un esempio

```

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class InventoryEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public void oneItemSold(Item item) {
        try {
            ut.begin();
            item.decreaseAvailableStock();
            sendShippingMessage();
            if (inventoryLevel(item) == 0)
                ut.rollback();
            else
                ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
        sendInventoryAlert();
    }
}

```

EJB stateless

Gestione transazioni a carico del bean

## Un esempio

```

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class InventoryEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public void oneItemSold(Item item) {
        try {
            ut.begin();
            item.decreaseAvailableStock();
            sendShippingMessage();
            if (inventoryLevel(item) == 0)
                ut.rollback();
            else
                ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
        sendInventoryAlert();
    }
}

```

EJB stateless

Gestione transazioni a carico del bean

Iniezione di una transazione utente

## Un esempio

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class InventoryEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public void oneItemSold(Item item) {
        try {
            ut.begin();
            item.decreaseAvailableStock();
            sendShippingMessage();
            if (inventoryLevel(item) == 0)
                ut.rollback();
            else
                ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
        sendInventoryAlert();
    }
}
```

- > EJB stateless
- > Gestione transazioni a carico del bean
- > Iniezione di una transazione utente
- > Inizio transazione

## Un esempio

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class InventoryEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public void oneItemSold(Item item) {
        try {
            ut.begin();
            item.decreaseAvailableStock();
            sendShippingMessage();
            if (inventoryLevel(item) == 0)
                ut.rollback();
            else
                ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
        sendInventoryAlert();
    }
}
```

- > EJB stateless
- > Gestione transazioni a carico del bean
- > Iniezione di una transazione utente
- > Inizio transazione
- > In una certa condizione, si chiede il rollback

## Un esempio

```

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class InventoryEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public void oneItemSold(Item item) {
        try {
            ut.begin();
            item.decreaseAvailableStock();
            sendShippingMessage();
            if (inventoryLevel(item) == 0)
                ut.rollback();
            else
                ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
        sendInventoryAlert();
    }
}

```

- > EJB stateless
- > Gestione transazioni a carico del bean
- > Iniezione di una transazione utente
- > Inizio transazione
- > In una certa condizione, si chiede il rollback
- > altrimenti, tutto ok

## Un esempio

```

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class InventoryEJB {
    @PersistenceContext(unitName = "chapter09PU")
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public void oneItemSold(Item item) {
        try {
            ut.begin();
            item.decreaseAvailableStock();
            sendShippingMessage();
            if (inventoryLevel(item) == 0)
                ut.rollback();
            else
                ut.commit();
        } catch (Exception e) {
            ut.rollback();
        }
        sendInventoryAlert();
    }
}

```

- > EJB stateless
- > Gestione transazioni a carico del bean
- > Iniezione di una transazione utente
- > Inizio transazione
- > In una certa condizione, si chiede il rollback
- > altrimenti, tutto ok
- > Se ci sono problemi, fai rollback

## Organizzazione della lezione

- Introduzione agli EJB
- Ciclo di Vita dei Session Beans
- Autorizzazioni
  - Autorizzazione dichiarativa
  - Autorizzazione da programma
- Transazioni
  - Cosa sono
  - Container-managed
  - Bean-Managed
- Conclusioni