# Enterprise Java Beans
# Parte 2

Programmazione Distribuita - A.A. 2020/2021


Biagio Cosenza
Dipartimento di Informatica
Università di Salerno
http://cosenza.eu/
bcosenza@unisa.it

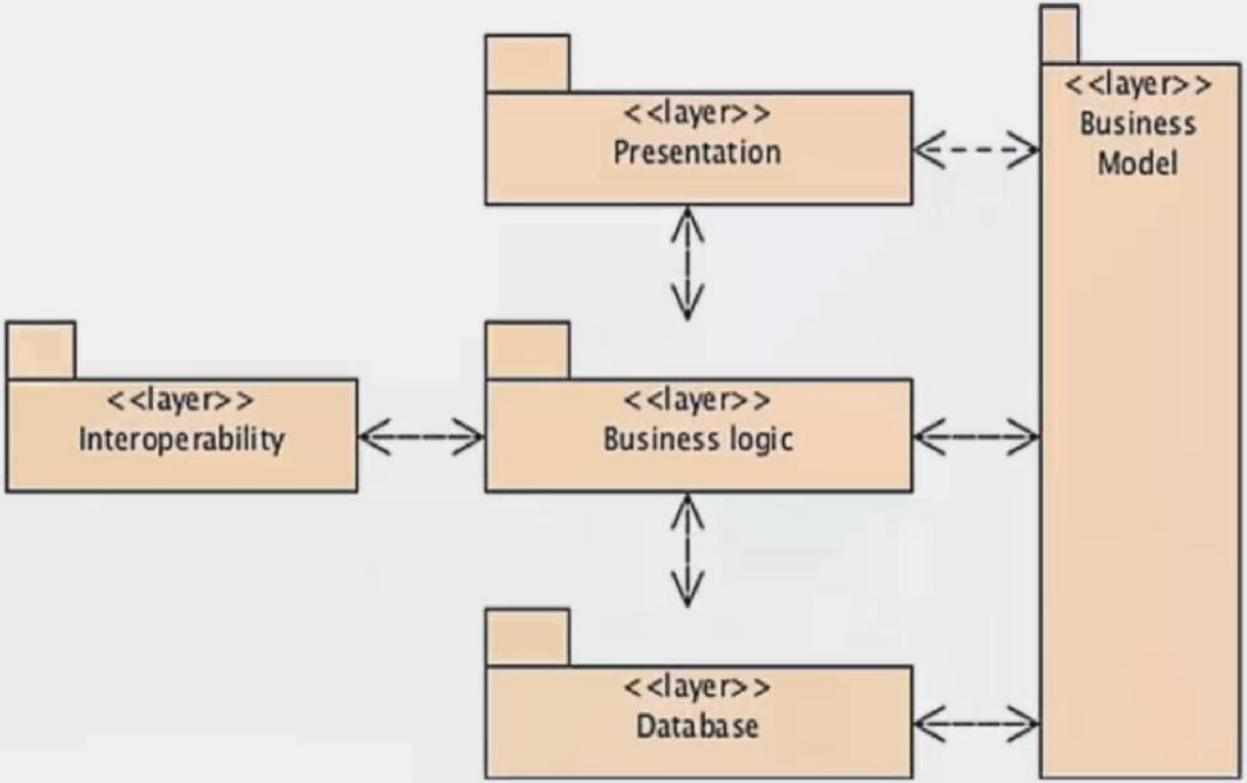
# Organizzazione della Lezione

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

# Introduzione: il ruolo deli 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) modella 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

# Introduzione: il ruolo centrale del Layer di Business



# Introduzione: Enterprise Java Beans

- 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

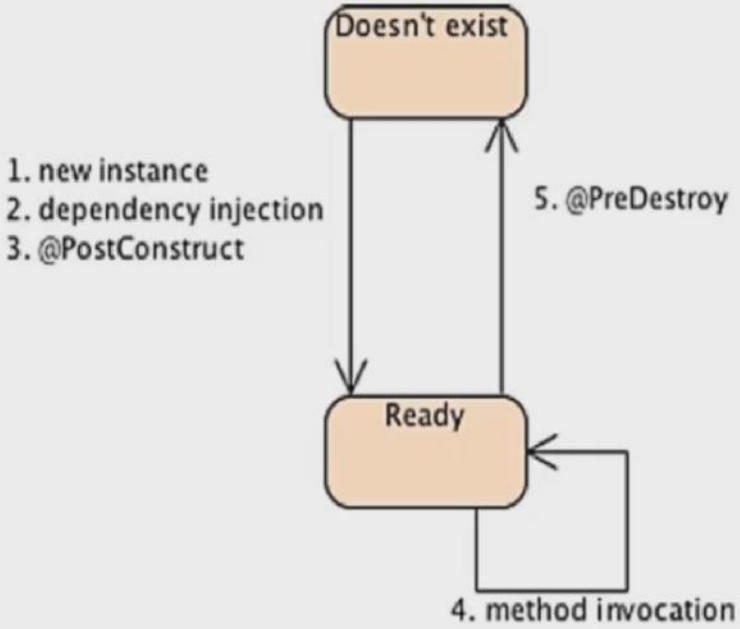
# Introduzione: servizi forniti dal Container a EJB

- Ciclo di vita
- Interceptor
- Transazioni
  - annotazioni per indicare transaction policy
  - il container e' responsabile di commit e rollback
- Sicurezza
  - user role authorization a livello di classe o metodo
- Concorrenza
  - Thread safety
- 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
- Messaging
  - gestione dei messaggi JMS
- Invocazione asincrona
  - senza messaggi

# Ciclo di vita dei Session Bean

- 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 l'iniezione di dipendenze o attraverso lookup JNDI
- Tutti 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

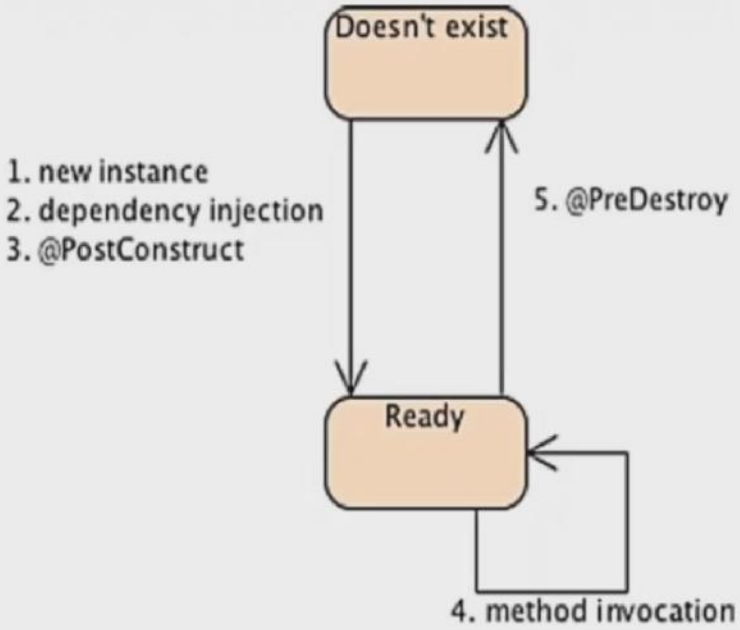
# Transizioni per Stateless e Singleton bean




Stateless e Singleton bean life cycle

# Transizioni per Stateless e Singleton bean

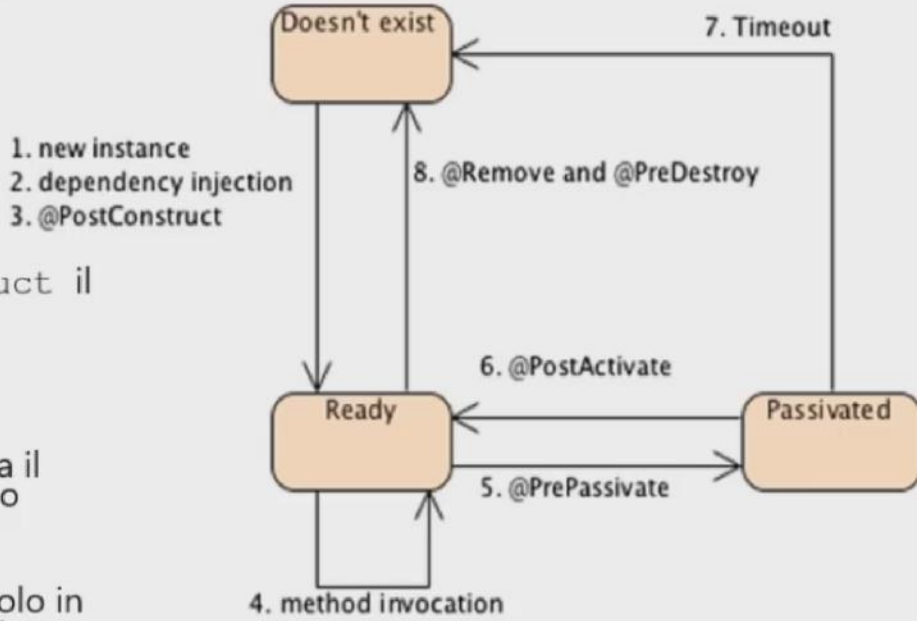
1. Il ciclo di vita inizia quando il client richiede un riferimento al bean
   - il container crea una nuova istanza
2. Alla creazione, vengono anche iniettate le dipendenze richieste
3. Se l'istanza appena creata ha un metodo annotato con `@PostContruct` il container lo invoca
4. Invocazioni dei metodi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




Stateless e Singleton bean life cycle

# Transizioni per Stateful bean

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 `@PostContruct` 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 *passivated*, 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




Stateful bean life cycle

# Annotazioni di Callback

| Annotation | Description |
|---|---|
| `@PostConstruct` | 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. |
| `@PreDestroy` | Marks a method to be invoked immediately before the container destroys the bean instance. The method annotated with `@PreDestroy` is often used to release resources that had been previously initialized. With stateful beans, this happens after timeout or when a method annotated with `@Remove` has been completed. |
| `@PrePassivate` | 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.). |
| `@PostActivate` | 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. |

# Esempio di Callback per Stateful EJB

- Risorsa da condividere: accesso a DB
  - iniettata dal container e condivisa
- Il container chiamerà il metodo `@PostConstruct` (`init()`)
  - crea una database connection
- Se il container rende passiva l'istanza, il metodo `close()` verrà invocato (`@PrePassivate`)
  - chiude 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`).

```
@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();
  }
}
```

# Autorizzazioni

- Modello di sicurezza: l'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
  - autorizzazione dichiarativa: dall'EJB container
  - autorizzazione da programma: attraverso Java Authentication and Authorization Service (JAAS API)

# Autorizzazione dichiarativa

- Può essere definita usando annotations oppure nel XML deployment descriptor

| 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 `launchNuclearWar()` should only be allowed in production but not in a test environment). |
| `@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. |

- Richiede di:
  - definire ruoli
  - assegnare permessi ai metodi (o all'intero bean)
  - o cambiare temporaneamente una security identity

# Esempio di autorizzazione su EJB

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

```
@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));
  }
}
```

# Autorizzazione da Programma

- 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

# Esempio di autorizzazione programmatica

- Contesto iniettato dal container
- Nel metodo viene controllato se 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 un utente specifico

```
@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;
 }
}
```

# Transazioni

- Le transazioni sono 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)

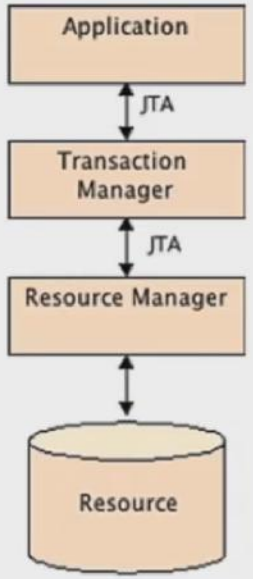
# Transazioni: Proprieta ACID

- 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

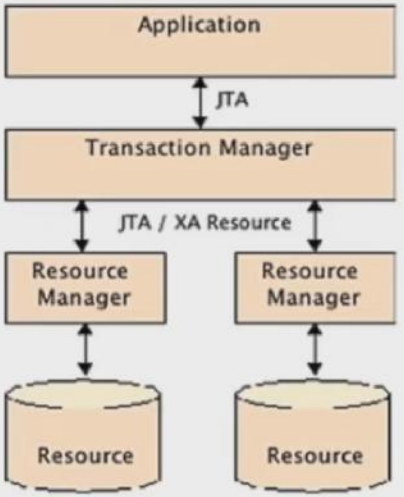
- Esempio semplice di applicazione su una risorsa singola (DB)
- Transaction Manager (TM)
  - componente core che gestisce le operazioni, informa il RM chi partecipa nella transazione
- Resource Manager (RM)
  - responsabile della gestione delle risorse e della loro registrazione con il transaction manager
  - 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.)




Transazione che coinvolge una risorsa

# Transazioni distribuite

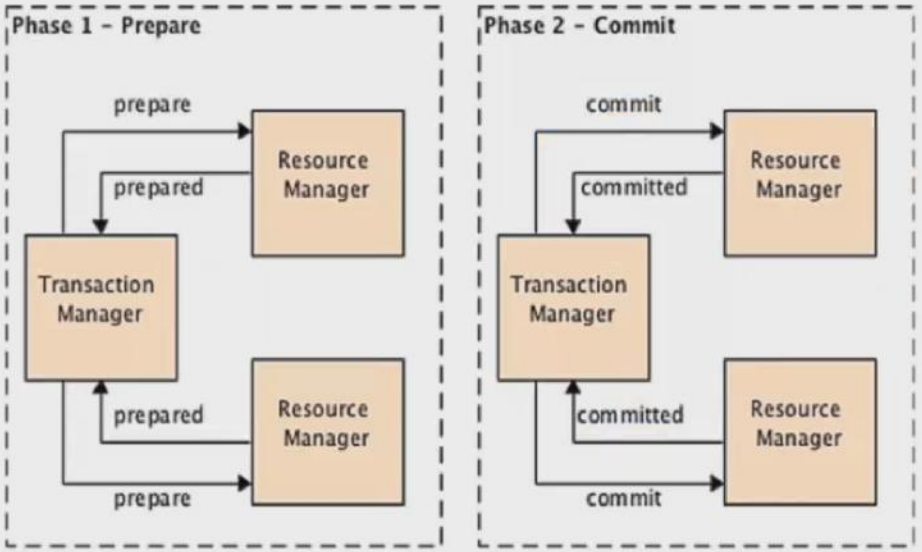
- Quando ci sono più risorse coinvolte, ad esempio diversi DB
- Coordinamento usando EXtended Architecture (XA) per Distributed Transaction Processing
- JTA supporta XA (eterogeneità vendor)
- Tecnica del two-phase commit: step di preparazione




Una transazione XA che coinvolge due risorse

# Two-phase commit

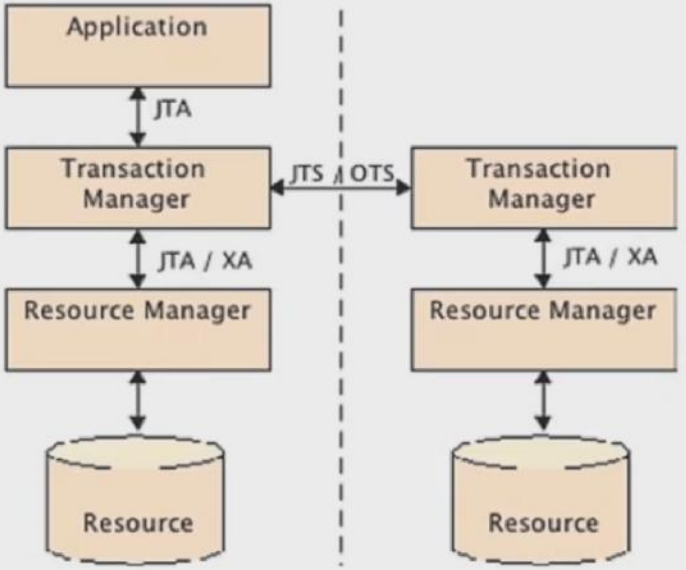
- Durante la fase 1, ogni Resource Manager viene notificato attraverso un comando "prepare" che un commit sta per essere eseguito
- Se tutti dichiarano che sono prepared, la transazione può procedere
  - ai resource managers viene chiesto di fare commit nella seconda fase




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

# Transazioni Container-Managed

- 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

# Transazioni Container-Managed: un esempio

- Bean stateless
- EM iniettato
- Un altro EJB iniettato

- Metodo per trovare libri
- Metodo per creare un libro

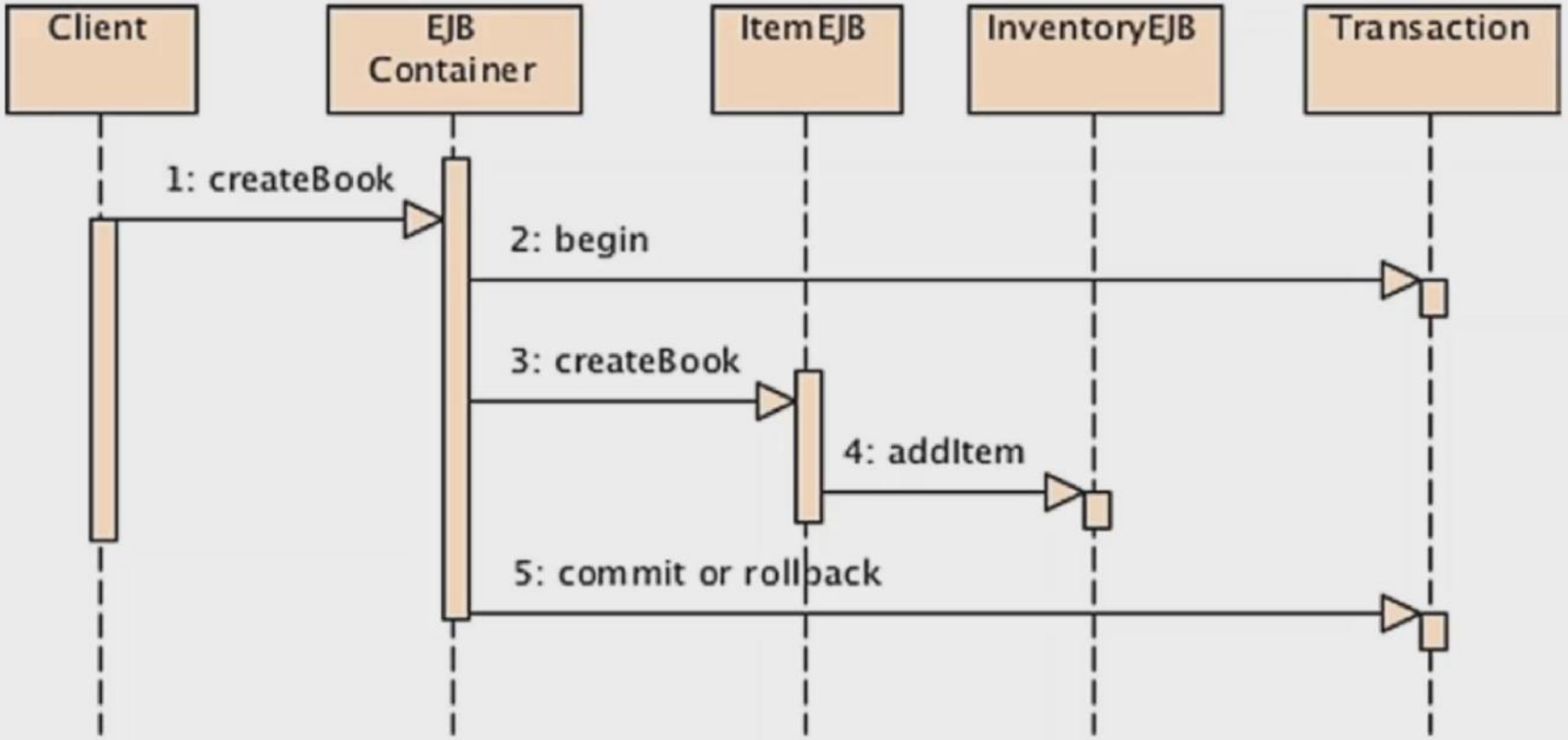
```
@Stateless
public class ItemEJB {
  @PersistenceContext(unitName ="chapter09PU")
  private EntityManager em;

  @Inject
  private InventoryEJB inventory;

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

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

# Transazioni Container-Managed quando si invoca createBook




Il container gestisce la transazione

# Transazioni Bean-managed

- EJB stateless
- Gestione transazioni a carico del bean
- Iniezione di una transazione utente
- Inizia transazione
  - in una certa condizione, si chiede il rollback
  - altrimenti, tutto ok e fa commit
- Se ci sono problemi, fai rollback

```
@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();
  }
}
```

# Conclusioni

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