Enterprise Java Beans
Parte 1

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
- Tipi di EJB
  - Stateless
  - Stateful
  - Singleton
- Come usare un EJB
  - Packaging e deploying
  - Invocazione
- Conclusioni

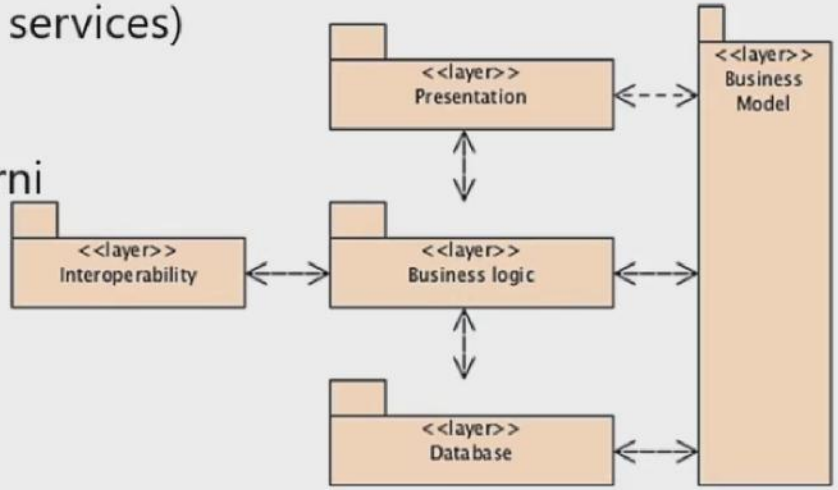
Introduzione: Ruolo degli EJB

- Ruolo degli EJB e la *business logic*
  - 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

Introduzione: Data e Business Layer

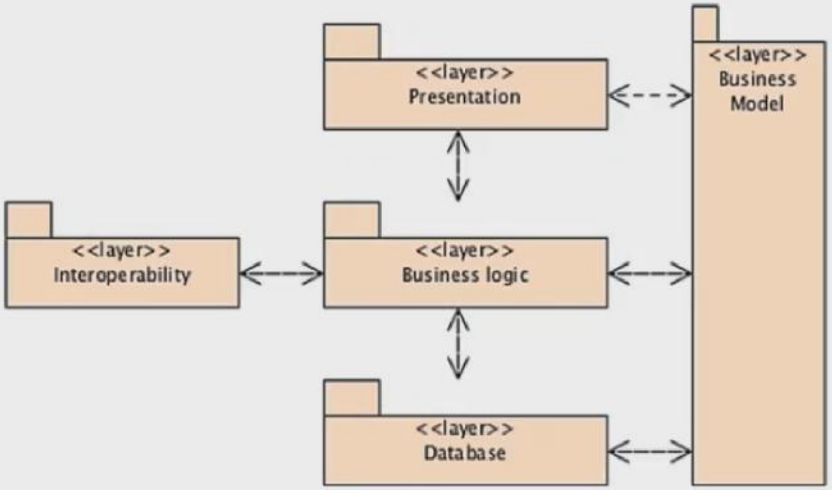
- JPA (Data layer) modella i "sostantivi" della nostra architettura
- EJB (Business layer) modella i "verbi"
- Il Business Layer ha anche il compito di:
  - interagire con servizi esterni (SOAP o RESTful web services)
  - inviare messaggi asincroni (usando JMS)
  - orchestrare componenti del DB verso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

Architecture Layering



Introduzione: Enterprise Java Beans

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



Servizi forniti dal Container

- 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

Interazione col Container

- 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

Caratteristiche di EJB Lite

- EJB Lite
  - Un sottoinsieme di funzionalita limitato a
    - stateless, stateful, and singleton session beans
    - local EJB interfaces or no interfaces
    - interceptors
    - container-managed and bean-managed transactions
    - declarative and programmatic security
    - embeddable API

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

Tipi di EJB

- Un session bean può avere diversi stati
- Stateless
  - il session bean non contiene *conversional state* tra i metodi
    - ogni istanza può essere usata da ogni client
  - utile per gestire task che possono essere conclusi con una singola chiamata a metodo
- Stateful
  - il session bean contiene un *conversional 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

Tipi di EJB: 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;
  }
```

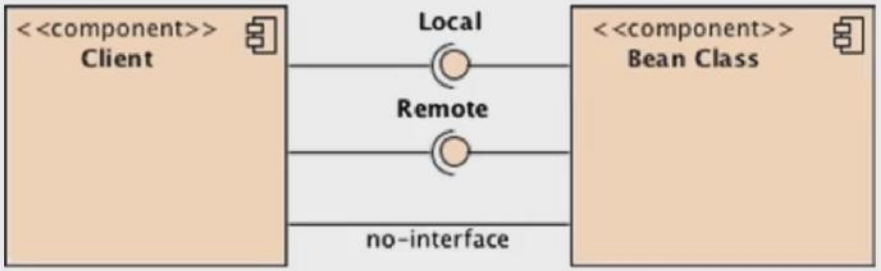
Tipi di EJB: un semplice EJB stateless

- Annotazione che definisce un bean senza stato
- Iniezione di dipendenza
  - un EM per la persistenza
- Variabile iniettata
- Un metodo del bean
- Un altro metodo del bean
  - che rende persistente un libro

```
@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;
  }
```

Anatomia di un EJB

- Un EJB si compone dei seguenti elementi:
  - Una classe bean: annotata con `@Stateless`, `@Stateful`, `@Singleton`
  - Interfacce di business che definisce quali metodi del bean sono visibili al client
    - Puo essere:
      - locale
      - remota
      - nessuna: significa solo accesso locale – invocando la classe bean stessa, client ed EJB nello stesso package



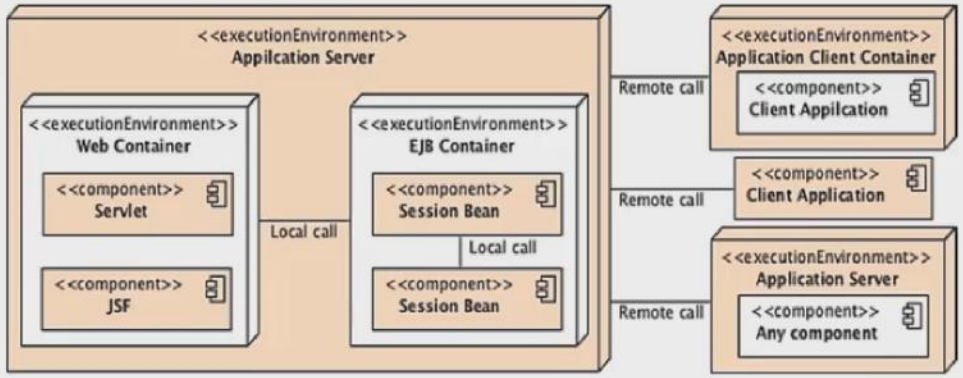
Tipi di business interface

Session Bean

- Una classe session bean è una classe Java standard che implementa la logica di business
- Requisiti per implementare un session bean:
  - annotata con `@Stateless`, `@Stateful`, `@Singleton` o con l'equivalente nel descrittore XML
  - deve implementare i metodi delle interfacce (se esistono)
  - deve essere `public`
    - non puo essere `final` o `abstract`
  - costruttore pubblico senza parametri
  - non deve definire il metodo `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

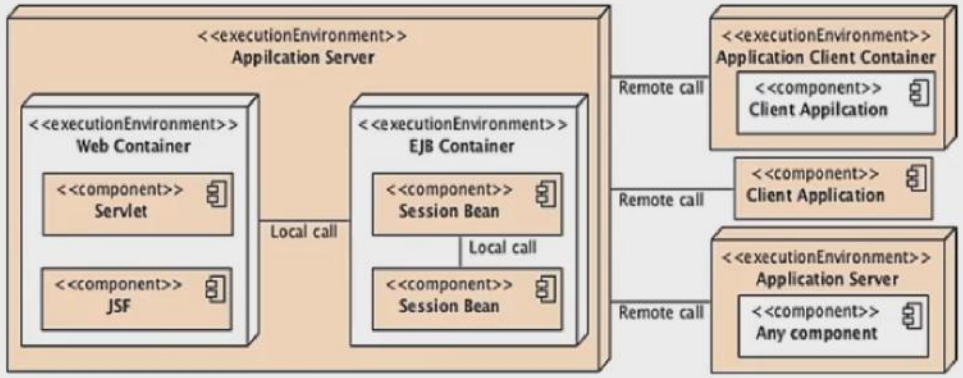
Remote, Local e No-interface Views (1)

- A seconda di dove un client invoca un session bean, il bean deve implementare una interfaccia: remota, locale o una no-interface view
- Remote View: Se l'architettura ha client che risiedono all'esterno dell'istanza JVM del container EJB, 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 RMI



Remote, Local e No-interface Views (2)

- Local view: 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



Remote, Local e No-interface Views (3)

- Un session bean può implementare diverse interfacce o nessuna
- Le annotazioni
  - `@Remote`: denota una remote business interface
    - Parametri di metodo passati per valore e serializzati essendo parte del protocollo RMI
  - `@Local`: denota una local business interface
    - Parametri di metodo passati per riferimento dal client al bean
- No-interface view
  - la vista senza interfaccia è una variante della vista locale (Local interface)
  - espone tutti i metodi pubblici di business della classe bean localmente senza l'utilizzo di un'interfaccia separata

Le Interfacce EJB

1. Annotazione per interfaccia locale

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

2. Annotazione per interfaccia remota

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

3. Utilizzo delle interfacce precedentemente annotate

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

- Nota: non si puo annotare la stessa interfaccia con piu di una annotazione

Le Interfacce EJB: Metodo alternativo per legacy interface

- Dichiarazione interfaccia normale

```
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);
```

- Specifica della dipendenza da interfacce
  - utilizzo delle interfacce precedentemente annotate
  - quando si marca con annotation `Remote` o `Local`, si perde la no-interface view automatica
  - `@LocalBean`: aggiunge la no-interface view

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

EJB con JNDI: Scope

- Alla creazione di un EJB viene creato automaticamente un nome Java Naming and Directory Interface (JNDI)
- Sintassi:

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

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

EJB con JNDI

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

- dove
  - 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

EJB con JNDI: un esempio

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

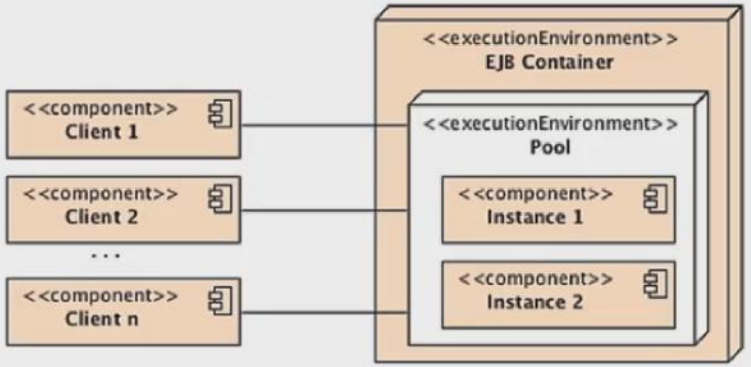
Tipi di EJB

- Stateless
- Stateful
- Singleton

Stateless Bean

- 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

Client che accedono a stateless bean in un pool



- 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
  - il container può gestire il loro ciclo di vita in maniera autonoma

Esempio di EJB Stateless

- Annotazione
- Iniezione del EM
- Metodi

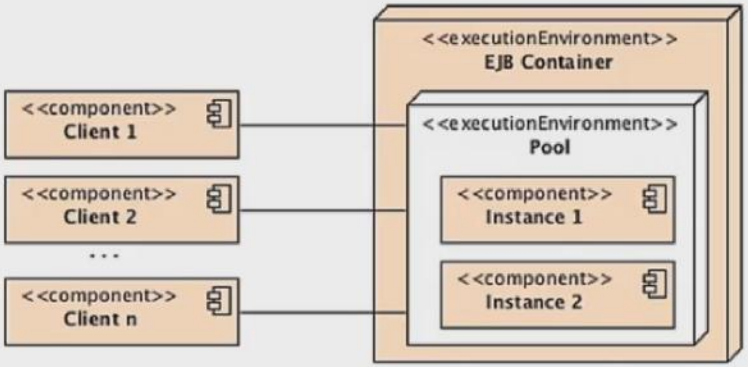
1. Metodo che esegue una query named sui libri (JPA)
2. Metodo che esegue una query named sui CD (JPA)
3. Crea un libro
4. Crea un CD

```
@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;
  }
}
```

Stateful Beans

- 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



Esempio di EJB Stateful (1)

- 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
- Struttura dati che mantiene lo stato
- Metodi per
  - aggiungere un elemento al carrello
  - rimuoverlo
  - per sapere quanti sono

```
@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();
  }

//...
```

Esempio di EJB Stateful (2)

- Metodi

1. Fa il totale degli elementi nel carrello
2. Svuota il carrello
3. Metodo invocato prima della rimozione
   - cancellazione del carrello, per il checkout

```
//...

 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();
 }
}
```

Esempio di EJB Stateful (2)

- Metodi

1. Fa il totale degli elementi nel carrello
2. Svuota il carrello
3. Metodo invocato prima della rimozione
   - cancellazione del carrello, per il checkout

- @Remove
  - applicato a un metodo di business di una classe session bean con stato,
  - indica al container che lo stateful session bean va rimosso dal container dopo aver eseguito il metodo

```
//...

 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();
 }
}
```

Esempio di EJB Stateful (1)

- 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
- Struttura dati che mantiene lo stato
- Metodi per
  - aggiungere un elemento al carrello
  - rimuoverlo
  - per sapere quanti sono

```
@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();
  }

//...
```

Esempio di EJB Stateful (2)

- Metodi

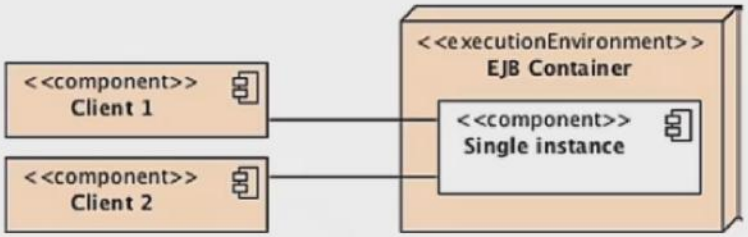
1. Fa il totale degli elementi nel carrello
2. Svuota il carrello
3. Metodo invocato prima della rimozione
   - cancellazione del carrello, per il checkout

```
//...

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();
}
}
```

Singleton Bean

- Dal design pattern Singleton
- 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



Design Pattern Singleton

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

Esempio di Pojo Singleton

- 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
- Metodi
  - aggiunge un oggetto
  - rimuove un oggetto
  - cerca un elemento

```
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;
  }
}
```

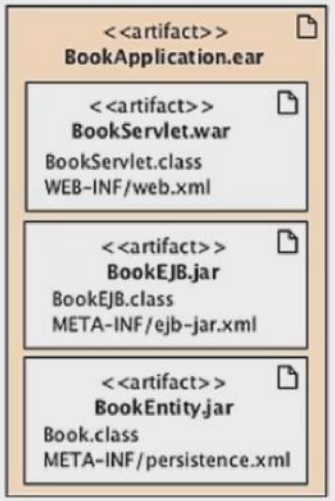
Rendere un Pojo un EJB Singleton

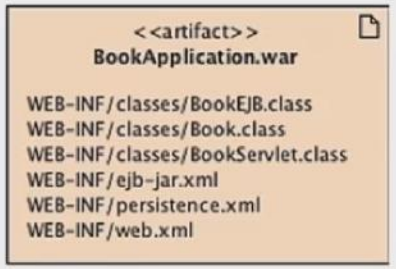
- Annotazione
- Creazione oggetto privato
- Metodi
  - aggiunge un oggetto in cache
  - rimuove un oggetto dalla cache
  - cerca un elemento in cache

```
@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;
  }
}
```

Come usare gli EJB: Packaging & deployment

- Packaging & deployment di EJB sul server
  - necessario il packaging per porre gli EJB in un container
  - necessario mettere insieme:
    - classi EJB, interfacce EJB
    - superclassi/superinterfacce
    - eccezioni, classi ausiliare
    - 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 deployed insieme





Come usare gli EJB: Invocazione di EJB da diverse componenti

- Il client di un EJB può essere di diversi tipi:
  - un POJO,
  - un client grafico
  - un CDI Managed Bean
  - una Servlet
  - un Bean JSF
  - un Web Service (SOAP o REST)
  - un altro EJB (in deployment nello stesso o un altro container)
- Il client NON può (ovviamente) fare `new()` su un EJB
  - ha bisogno di un riferimento, che può essere ottenuto via
    1. dependency injection: iniettato via `@EJB` o `@Inject`
    2. JNDI: acceduto via un lookup JNDI

Come usare gli EJB: Invocazione con iniezione del riferimento (1)

- 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;
```

Come usare gli EJB: Invocazione con iniezione del riferimento (2)

- 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;
```

Come usare gli EJB: Invocazione con iniezione del riferimento (3)

- Se l'EJB è remoto, si può specificare il nome JNDI
- Annotazione `@EJB` 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;
```

Come usare gli EJB: Invocazione con iniezione del riferimento (4)

- 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;
```

Come usare gli EJB: Invocazione diretta di JNDI

- 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");
```

Conclusioni

- Introduzione agli EJB
- Tipi di EJB
  - Stateless
  - Stateful
  - Singleton
- Come usare un EJB
  - Packaging e deploying
  - Invocazione
- Conclusioni