Dipartimento di Informatica
e Sistemistica Antonio Ruberti

Sapienza
Università di Roma

Enterprise Java Beans (EJB)

SAPIENZA – Università di Roma

Corso di *Architetture Software Orientate ai Servizi*

Enterprise Java Beans

- Gli *Enterprise Java Beans* (***EJB***) sono componenti scritti in Java che realizzano la logica di business (**application layer**) di un'applicazione
  - La *business logic* è il codice che realizza l'obiettivo dell'applicazione.
    - Ad esempio, in un'applicazione di *e-Commerce*, il metodo `orderProduct` può essere usato da un client (*presentation layer*) per ordinare un prodotto.
- Vivono in un **EJB Container**, un *runtime environment* nell'*Application Server*.
  - Il *Container* gestisce l'intero ciclo di vita di un EJB e le sue invocazioni.
  - Fornisce meccanismi ausiliari, come la transazionalità e la sicurezza, in modo trasparente ai client

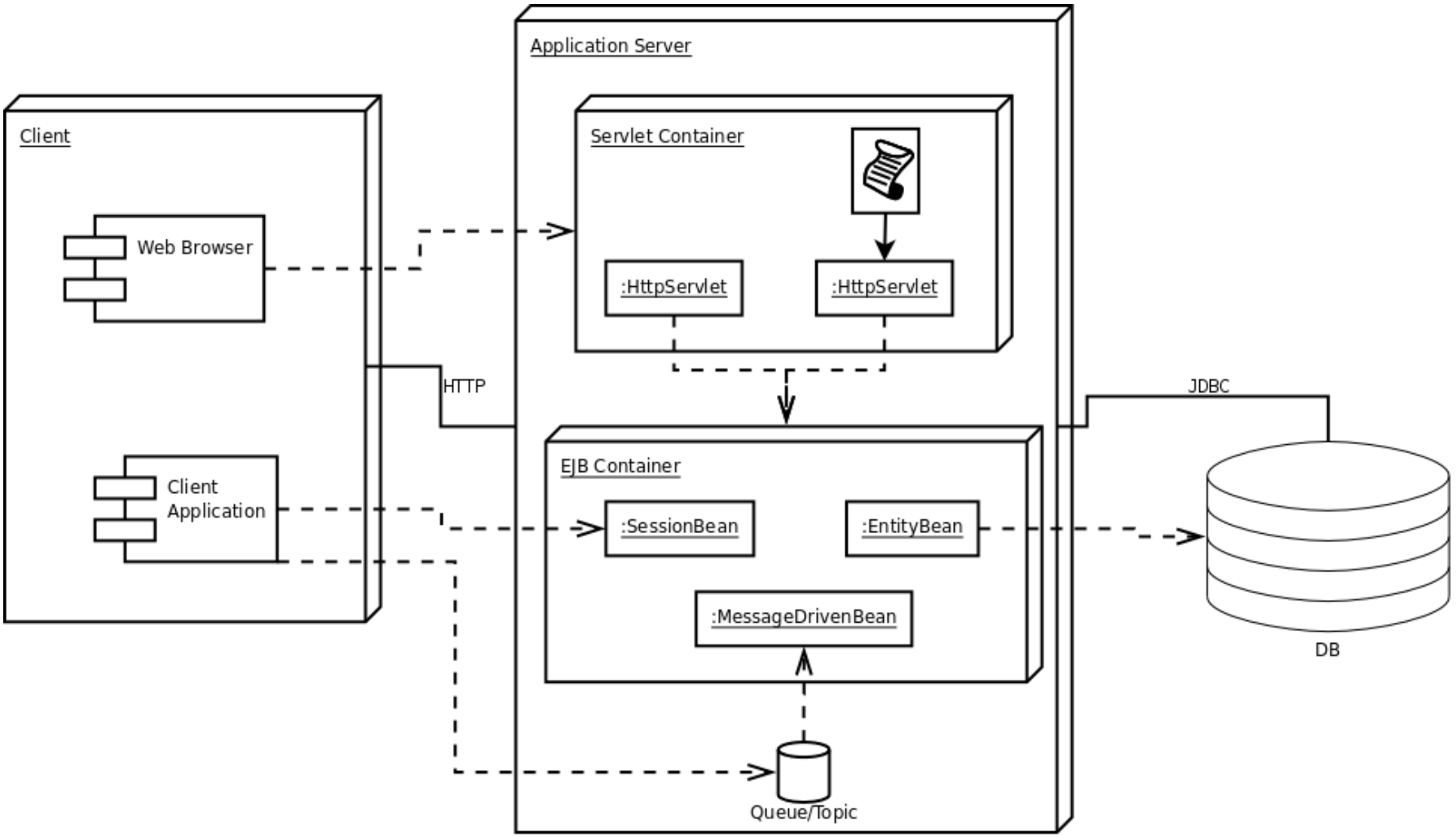
Benefici degli EJB

- Gli **EJB** realizzati coerentemente con le specifiche **J2EE** possono essere dispiegati (**deploy**) su differenti *Application Server* J2EE-Compliant
- Gli EJB gestiscono la *business logic* in modo trasparente ai client
  - I client possono essere più leggeri
    - non *fat client* ma *thin client*
      - aspetto importante per quei client eseguiti su *low-profile devices* (es. PDA, SmartPhone, PC poco potenti, ecc...).
- Gli EJB sono **portabili** e più EJB possono essere assemblati per formarne di nuovi

Tipi di Enterprise Java Beans

Tipo di Enterprise Bean	Uso
Session Beans	Rappresentano le sessioni di interazione con i client. Sono *transienti*: il loro stato non viene memorizzato in modo *persistente* in un database o simili.
Entity Beans	Rappresentano gli oggetti di business dell'applicazione. Sono memorizzati in modo *persistente*. Spesso, un'istanza di Entity Bean corrisponde ad una tupla di un DB.
Message-Driven Beans	Rimangono in "ascolto" su un topic/coda JMS e si attivano quando un messaggio è inviato a quel topic/coda .

Container


Application Server
Client
Servlet Container
Web Browser
:HttpServlet
:HttpServlet
HTTP
JDBC
EJB Container
Client
Application
:SessionBean
:EntityBean
:MessageDrivenBean
DB
Queue/Topic

Session Bean

- Un ***Session Bean*** rappresenta un singolo client presso l'Application Server. Il client accede al *Business Layer* di una applicazione invocando i metodi di Session Bean.
- Dal punto di vista del client, ogni Session Bean è associato ad uno ed uno solo Client
  - Lo stato dell'interazione con un client (lo stato *Session Bean) è transiente*: quando termina la sessione a cui è associato, un Session Bean viene distrutto, insieme con lo stato
  - Una sessione termina quando l'utente esplicitamente la rimuove oppure per un periodo di inattività superiore ad un tempo prefissato
- Esistono due tipi di Session Bean: ***stateless*** e ***stateful***

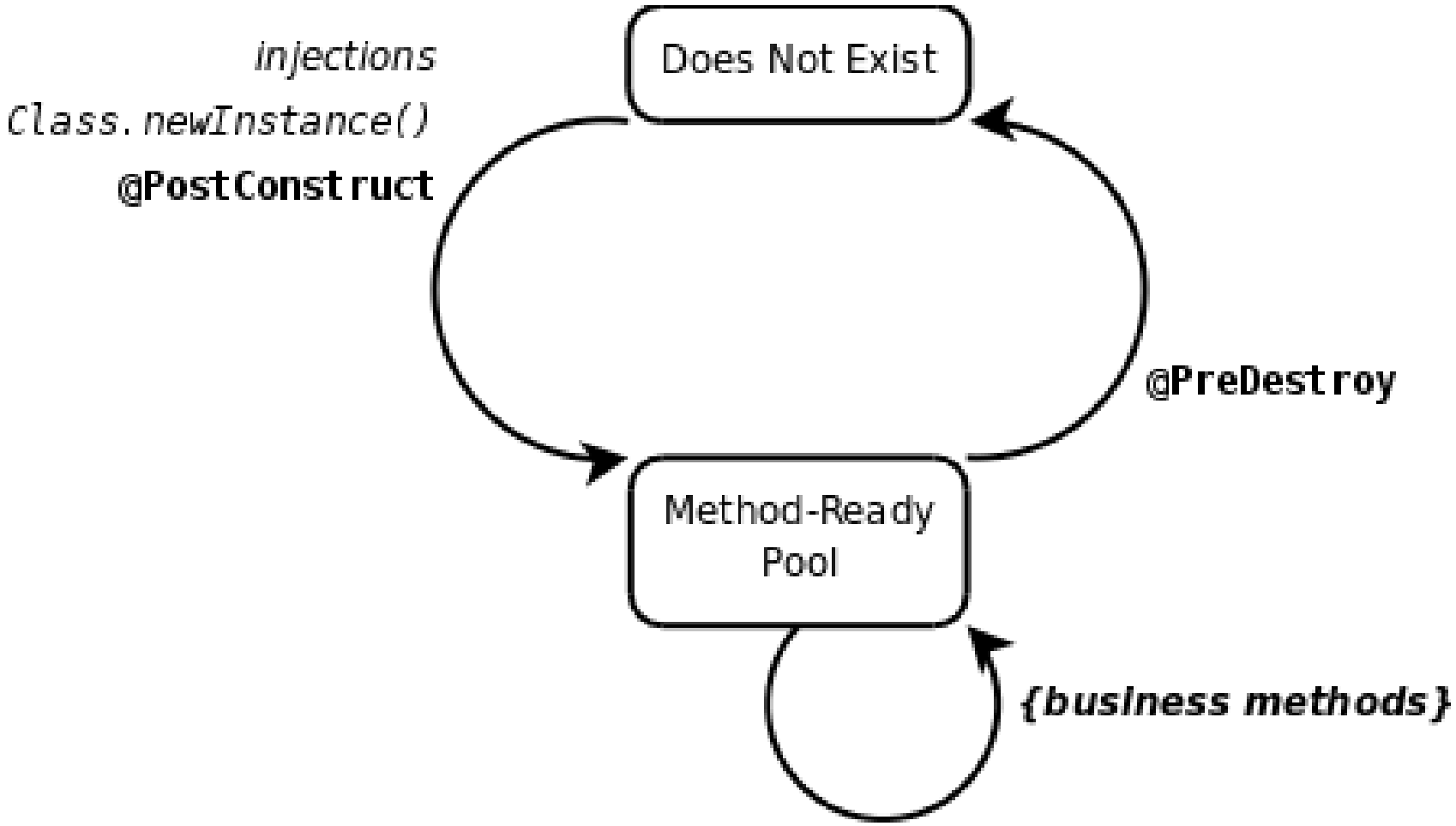
Stateless Session Bean

- Non mantengono lo stato conversazionale nella sessione con il client
- Poiché non è previsto uno stato conversazionale, il Container può assegnare un qualsiasi session EJB stateless già esistente a qualsiasi client
  - Il Container può associare lo stesso Stateless Bean a più client per risparmiare memoria
    - Questa politica di allocazione degli Stateless Bean dipende dall'implementazione dell'EJB Container ed è trasparente ai client che "credono" di avere uno *stateless session bean* dedicato.

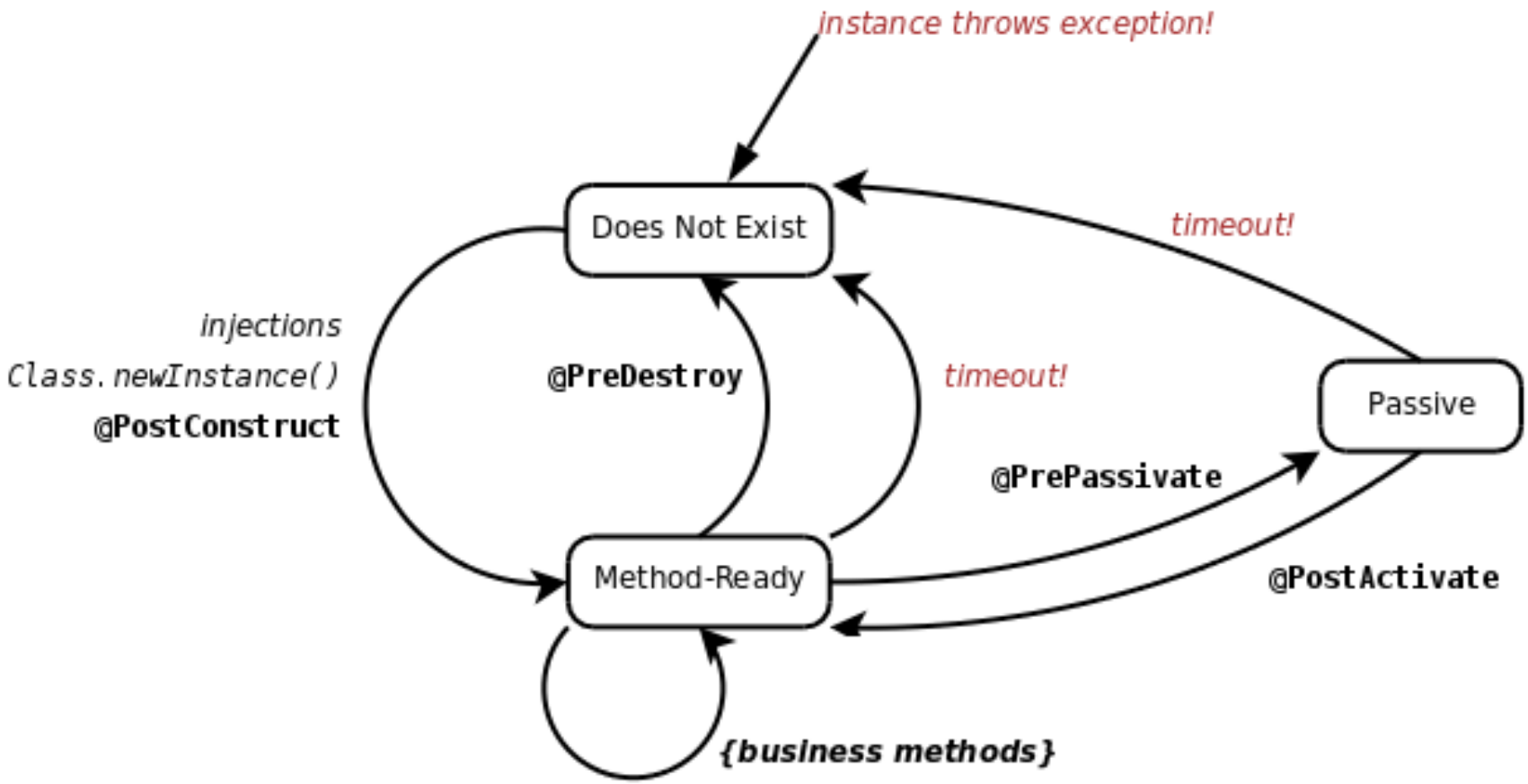
Stateful Session Bean

- Mantengono lo stato conversazionale nella sessione con il client
- Ogni Stateful Session Bean è associato sempre ad un solo client
  - Per esempio, un carrello della spesa è un candidato Session EJB di tipo Stateful
    - L'invocazione ad opportuni metodi *add* permettono di aggiungere oggetti al carrello e un metodo *completeOrder* chiude l'ordine, invocando un Entity Bean per la memorizzazione dell'ordine.

Ciclo di vita di Session Bean Stateless



Ciclo di vita di Session Bean Stateful



Creazione di Session Bean Stateless: l'interfaccia remota

1. Scrivere l'interfaccia remota che descrive i metodi implementati dal Bean per fornire il servizio.

```
import javax.ejb.Remote;

@Remote
public interface CalcolatriceRemota
{
   public int addiziona(int x, int y);
   public int sottrai(int x, int y);
}
```

Creazione di Session Bean Stateless: implementazione

2. Codificare la classe che implementa l'interfaccia

```
import javax.ejb.Stateless;

@Stateless(name="Calcolatrice")
public class CalcolatriceBean
          implements CalcolatriceRemota
{
   public int addiziona(int x, int y) {
          return x + y;
   }
   public int sottrai(int x, int y) {
          return x - y;
   }
}
```

Creazione di Session Bean Stateless: implementazione

2. Codificare la classe che implementa l'interfaccia

```
import javax.ejb.Stateless;

@Stateless(name="Calcolatrice")
@Remote(CalcolatriceRemota.class)
public class CalcolatriceBean
{
   public int addiziona(int x, int y) {
          return x + y;
   }
   public int sottrai(int x, int y) {
          return x - y;
   }
}
```

Creazione di Session Bean Stateless: implementazione

2. Codificare la classe che implementa l'interfaccia

```
import javax.ejb.Stateless;

@Stateless(name="Calcolatrice")
@Remote(CalcolatriceRemota.class)
public class CalcolatriceBean
          implements CalcolatriceRemota
{
   public int addiziona(int x, int y) {
          return x + y;
   }
   public int sottrai(int x, int y) {
          return x - y;
   }
}
```

Scrittura del client / 1

- Il codice del client tipicamente effettua i seguenti passi:
  1. Effettua il ***look-up*** per ottenere un riferimento ad un'istanza che implementi l'interfaccia remota del Session Bean, tramite **JNDI**
  2. Chiama i metodi di business sull'oggetto EJB ottenuto
- Per poter compilare ed eseguire il codice del client, occorre includere:
  1. le librerie *client* dell'Application Server
     - ad esempio, in JBoss **$JBOSS_HOME/client/jbossall-client.jar**
  2. file bytecode delle classi adoperate
     - interfacce remote
     - classi DTO
     - …

Scrittura del client / 2

```
public static void main(String args[])
{
  Properties prop = new Properties();
  prop.put(Context.INITIAL_CONTEXT_FACTORY,
            "org.jnp.interfaces.NamingContextFactory");
  prop.put(Context.URL_PKG_PREFIXES, "org.jnp.interfaces");
  prop.put(Context.PROVIDER_URL, "localhost");
  Context ctx = new InitialContext(prop);
  Object obj = ctx.lookup("Calcolatrice");
  CalcolatriceRemota calc = (CalcolatriceRemota)
          javax.rmi.PortableRemoteObject
          .narrow(obj, CalcolatriceRemota.class);
  System.out.println(
    "4 + 5 = " + calc.addiziona(4, 5)
  );
}
```

Informazioni necessarie per inizializzare il Contesto JNDI

Init del Context JNDI

Acquisizione del riferimento al Bean

Utilizzo del Bean

Le interfacce locali

- Le interfacce *remote* realizzano invocazioni, passaggio dei parametri e dei valori di ritorno attraverso la *serializzazione* ed il *marshalling*, in un ambiente client/server distribuito
- Le *interfacce locali* sono adoperate da client che devono risiedere nella stessa JVM del bean acceduto
  - e.g. componenti web, altri enterprise bean

```
import javax.ejb.Local;

@Local
public interface CalcolatriceLocale
{
   public int moltiplica(int x, int y);
   public int dividi(int x, int y);
}
```

Le interfacce locali

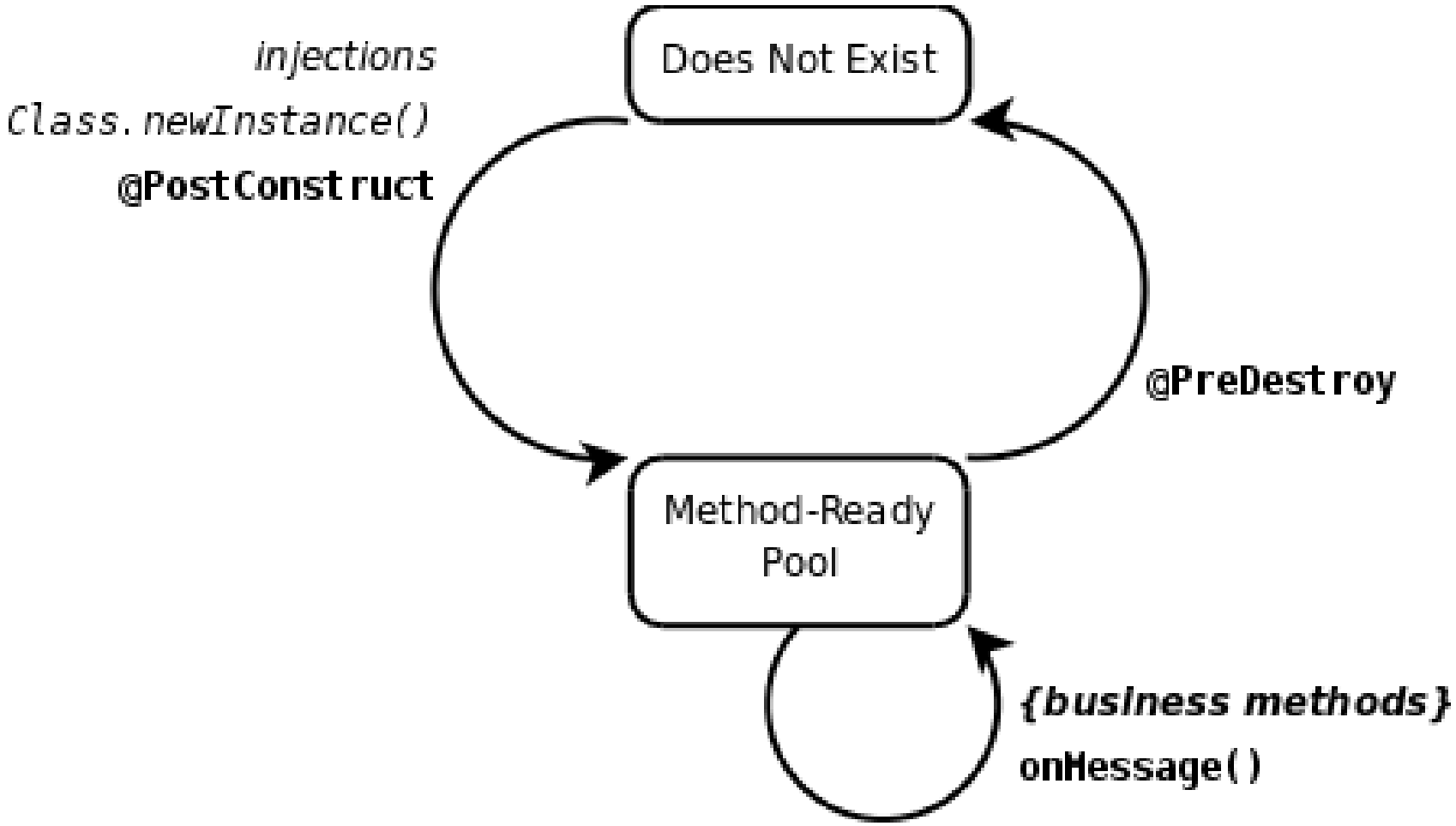
- L'ottenimento da parte del local client del riferimento ai bean acceduti è molto più semplice
  - gestito dal container

```
import javax.ejb.EJB;
//...
@Stateless
public class CalcolatriceGlobale
{
   @EJB
   private CalcolatriceLocale calc;
   //...
   public int moltiplica(int x, int y) {
       return this.calc.moltiplica(x, y);
   }
}
```

Message-Driven Bean

- Un *Message-Driven Bean* è un componente EJB che consuma messaggi da una queue/topic, inviati da un valido client JMS
- Un MDB è disaccoppiato da qualsiasi client che invia messaggi JMS
  - Un client non può accedere al Bean attraverso alcuna interfaccia. Il MDB non ha interfacce *locali* né *remote*.
  - Un MDB non ritorna valori né rilancia mai eccezioni ai client
  - I MDB sono *stateless* (non mantengono stato conversazionale)

Ciclo di vita di Message Driven Bean



Esempio

```
import javax.ejb.MessageDriven;
import javax.jms.MessageListener;
import javax.ejb.ActivationConfigProperty;
//...
@MessageDriven(
        activationConfig = {
                @ActivationConfigProperty(
                        propertyName = "destinationType",
                        propertyValue = "javax.jms.Queue"),
                @ActivationConfigProperty(
                        propertyName="destination",
                        propertyValue="queue/messaggi")
})
public class MessaggioBean implements MessageListener {
        public void onMessage(Message m) {
                TextMessage tm = (TextMessage)m;
                String text = tm.getText();
                System.out.print("Ricevuto messaggio:");
                System.out.println(text);
        }
}
```

Entity Bean / 1

- Rappresenta un “oggetto del business” conservato in un meccanismo di memoria persistente, i.e., database relazionale (e.g., utenti, prodotti nel carrello)
- tipicamente un entity bean corrisponde ad una tabella in un database relazionale, ed ogni istanza del bean corrisponde ad una riga nella tabella
  - prevedono l'esistenza di proprietà definite **chiavi primarie** per essere identificati univocamente
  - possono essere legati ad altri entity bean da relazioni
    - di molteplicità
      - one-to-one
      - one-to-many
      - many-to-one
      - many-to-many
    - con riferimenti
      - unidirezionali
      - bidirezionali

Entity Bean / 2

- Persistenza
  - Due tipi
    - Gestita dal bean
      - il codice del bean contiene le chiamate al database
    - Gestita dal contenitore
      - il container EJB genera automaticamente le chiamate al database
- Condivisi
  - Possono essere condivisi fra vari client
  - Poiché diversi clienti possono volere cambiare gli stessi dati, è importante che siano dotati di un meccanismo di transazioni
    - Di norma, il container fornisce la gestione delle *transazioni*

Entity Bean / 3

- Chiave primaria
  - Ogni entity bean ha un unico “identificatore”
    - un entity bean `Studente` può essere identificato dal campo `matricola`
  - La chiave primaria permette al gestore degli entity bean di individuare l'oggetto
- Relazioni
  - Come in un database relazionale, gli entity bean possono essere collegati da relazioni
  - Se la persistenza è
    - gestita dal bean
      - allora nel codice del bean devono essere esplicitamente gestite le relazioni
    - gestita dal contenitore
      - allora il container gestisce le relazioni in modo trasparente allo sviluppatore

Persistenza gestita dal container

- Il container EJB gestisce tutti gli accessi al database necessari; il codice del bean non contiene istruzioni SQL
  - il bean non è legato ad uno specifico database
    - se il bean è installato su un server diverso da quello di sviluppo iniziale, non occorre modificare o ricompilare il codice dal bean
      - **il bean è portabile**
- per generare gli accessi al database il container necessita di meta-dati
- **schema astratto** dell'entity bean
  - definisce i campi persistenti del bean e le sue relazioni con altri entity bean
  - astratto per distinguerlo dallo schema “concreto” del/dei database a cui il bean sarà collegato

Entity Bean / esempi (1)

```
package it.uniroma1.dis.sis.wster.db;
import java.util.List;
import javax.persistence.*;
@Entity
public class Author implements java.io.Serializable {
     @Id
     @GeneratedValue
     private Long id;
     @Column(nullable=false)
     private String name;
     @OneToMany(mappedBy="author")
     @OrderBy("title")
     private List<Song> songs; // seguono getter e setter...
}
```

Entity Bean / esempi (2)

```
package it.uniroma1.dis.sis.wster.db;
import java.util.List;
import javax.persistence.*;
@Entity
public class Album implements java.io.Serializable {
     @Id
     @GeneratedValue
     private Long id;
     @Column(nullable=false)
     private String title;
         // per default, anche year è una colonna, nullable!
     private Short year;
     @OneToMany(mappedBy="album")
     @OrderBy("title")
     private List<Song> songs; // seguono getter e setter...
}
```

Entity Bean / esempi (3)

```
package it.uniroma1.dis.sis.wster.db;
import javax.persistence.*;
@Entity
@NamedQueries(    {
      @NamedQuery(name="getSongByTitle",
        query="SELECT s FROM Song s" +
            " WHERE s.title LIKE" +
              " CONCAT('%', CONCAT(:title, '%'))" +
            " ORDER BY s.title, s.id"),
      @NamedQuery(name="getSongByAuthor",
          query="SELECT s FROM Song s" +
              " WHERE s.author.name LIKE" +
                " CONCAT('%', CONCAT(:authorName, '%'))" +
              " ORDER BY s.title, s.author.name, s.author.id")
}    )
public class Song implements java.io.Serializable {// continua...
```

Entity Bean / esempi (4)

```
public class Song implements java.io.Serializable {
     @Id
     @GeneratedValue
     private Long id;
     @Column(nullable=false)
     private String title;
     @ManyToOne(fetch=FetchType.EAGER, optional=false)
     private Author author;
     @ManyToOne(fetch=FetchType.EAGER)
     private Album album;
     @Column(nullable=false)
     private String url; // seguono getter e setter...
}
```

Packaging di applicazioni basate su EJB

- Per dispiegare l'applicazione su *Application Server* occorre creare un pacchetto JAR strutturato come in figura, e copiarlo all'interno della directory

  `$JBOSS_HOME/server/default/deploy`