



1

Organizzazione della Lezione

- Introduzione agli EJB
- Esempio: HelloWorld EJB
- Esempio: Book EJB
- Conclusioni



UNIVERSITÀ DEGLI STUDI DI SALERNO

2

2

Un HelloWorld EJB

- Esempio HelloWorld che include
 1. Il bean (stateless)
 2. Interfaccia remota
 3. Client



UNIVERSITÀ DEGLI STUDI DI SALERNO

3

3

HelloWorld Bean

- Il package del bean
- Import necessaria per lo Stateless
- Annotazione per il tipo di EJB
- Classe
- Interfaccia remota
- Metodo di business che
- restituisce un saluto al nome passato

```
package hello;
import javax.ejb.Stateless;

@Stateless
public class HelloBean
    implements HelloWorldBeanRemote {

    @Override
    public String sayHello(String name) {
        return "Hello"+name+"!";
    }
}
```



UNIVERSITÀ DEGLI STUDI DI SALERNO

4

4

L'Interfaccia Remota

- Package del EJB
- Import necessaria per la annotazione
- Interfaccia remota
- Dichiarazione interfaccia
- Metodo di business per dire ciao

```
package hello;
import javax.ejb.Remote;

@Remote
public interface HelloWorldBeanRemote {

    String sayHello(String name);
}
```

Il Client

- Package
- Import vari
- Nome client
- Lancia eccezione per la lookup
- Contesto di esecuzione
- Preleva il contesto (niente parametri: libreria client GlassFish)
- Restituendo un bean remoto
 - si fa la lookup a JNDI
 - con il nome globale
- Invocazione remota

```
package helloworldclient;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class HelloWorldClient {

    public static void main(String[] args)
        throws NamingException {
        Context ctx;
        ctx = new InitialContext();

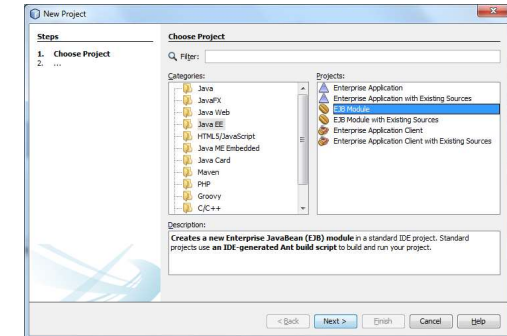
        HelloWorldBeanRemote helloBean = (HelloWorldBeanRemote)
            ctx.lookup("java:global/HelloWorldBean/HelloBean!hello.
                HelloWorldBeanRemote");
        System.out.println("Ora invoco...");
        System.out.println(helloBean.sayHello("Ciao!"));
    }
}
```

Pratica su NetBeans con un HelloWorld EJB

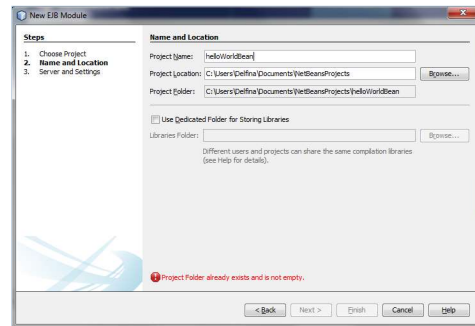
- I passi da seguire
 - Creiamo un progetto per l'EJB
 - Creiamo un progetto per il client
 - incluso la configurazione di librerie, etc.
 - Deployment ed esecuzione su server dell'EJB
 - Invocazione del servizio da parte del client
- IMPORTANTE:** Il nome del progetto deve essere diverso dal nome del session bean!

HelloWorld EJB: Tipo del Progetto

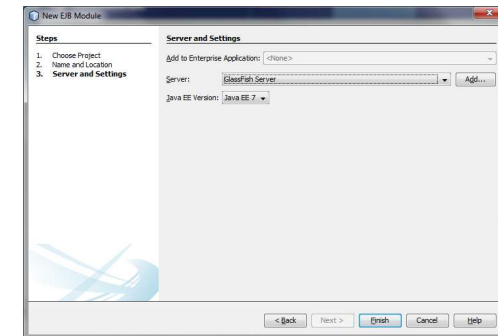
- Creazione del progetto
 - New Project
 - Java with Ant
 - Java Enterprise
 - EJB Module



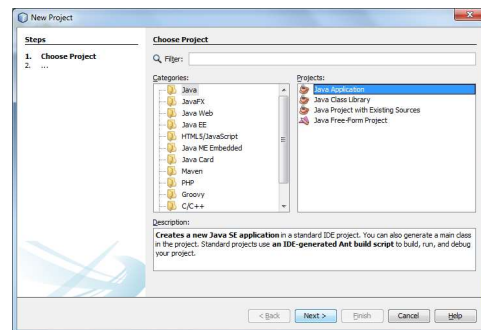
HelloWorld EJB: Nome del Progetto



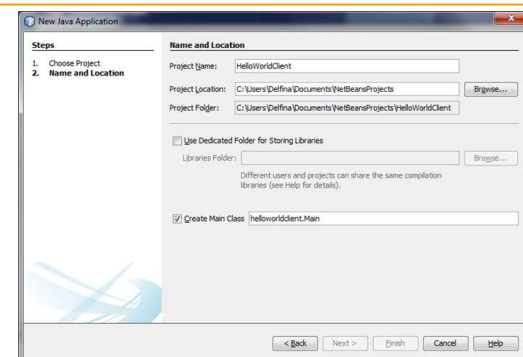
HelloWorld EJB: Setting del Progetto



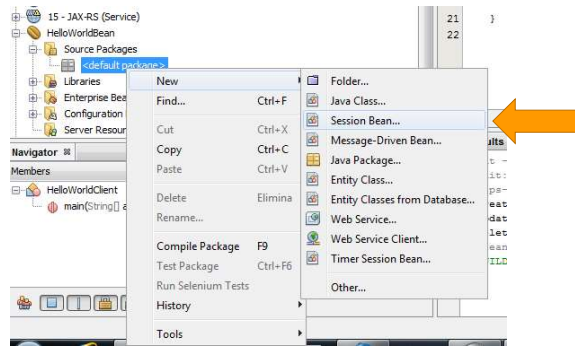
HelloWorld EJB: Client: Tipo del Progetto



HelloWorld EJB: Client: Nome del Progetto

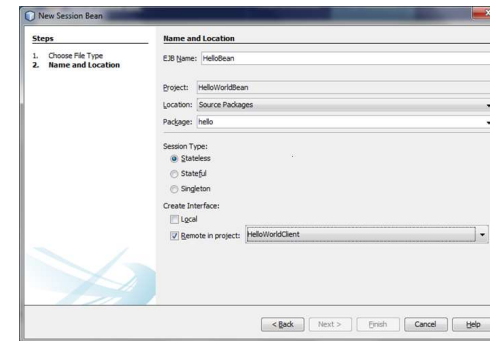


HelloWorld EJB: Creiamo un Session Bean



13

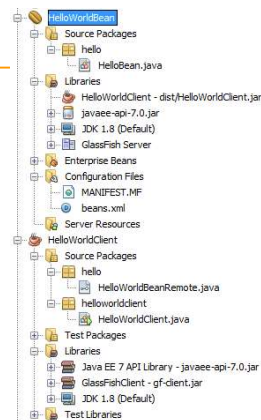
HelloWorld EJB: Parametri del Session Bean



14

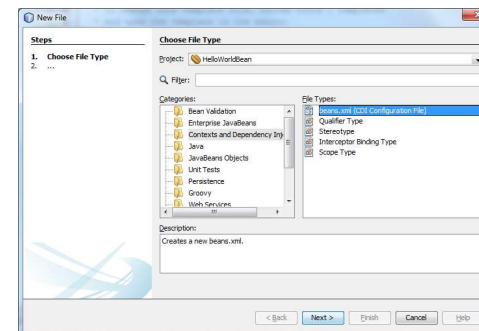
Status dei Progetti

- Due progetti
 1. Uno di tipo EJB (notare l'icona)
 2. Uno di tipo standard (client)
- Notare la presenza del package hello anche dentro il Client
- La presenza dell'interfaccia remota è spostata automaticamente solo sul client
 - al deployment il server riesce a generare automaticamente l'interfaccia remota



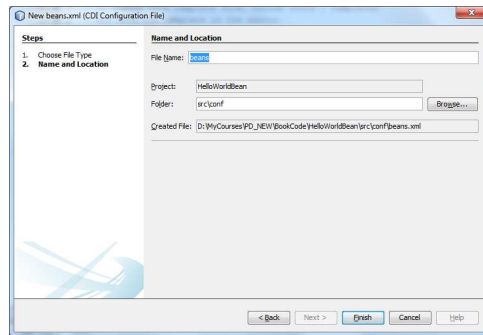
15

HelloWorld EJB: Creazione del file beans.xml



16

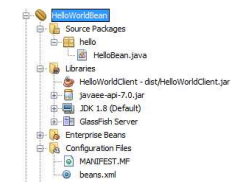
HelloWorld EJB: Nome del file beans.xml



17

HelloWorld EJB: Bean discovery mode

- Il file beans.xml è «vuoto»
 - Senza di esso, il server non lo considera «iniettabile»
 - Sintomo: al deploy tutto bene, ma poi non riesce a fornirlo
 - bean-discovery-mode: da annotated deve diventare all



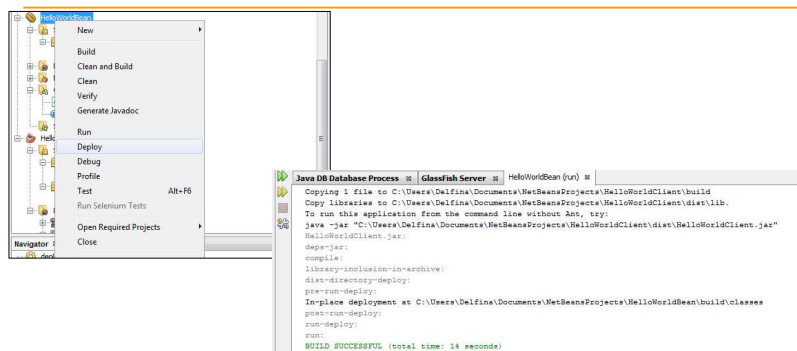
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
5       bean-discovery-mode="annotated">
6 </beans>

```

18

HelloWorld EJB: Deployment



19

HelloWorld EJB: Log del server al deploy

```

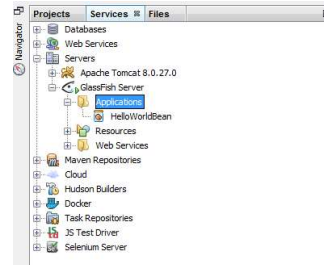
Output | Test Results
Java DB Database Process | GlassFish Server | HelloWorldBean (run)
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] public org.glassfish.jms.injection.
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] private org.glassfish.jersey.gf.cdi
Informazioni: Loading application _JMS_MDB_Lab5 done in 7.419 ms
Informazioni: GlassFish Server Open Source Edition 4.1 (13) startup time : Felix (19.737ms),
Informazioni: Registered com.sun.enterprise.glassfish.bootstrap.osgi.EmbeddedOSGiGlassFishImpl
Informazioni: Grizzly Framework 2.3.15 started in: 15ms - bound to [/0.0.0.0:8181]
Informazioni: Grizzly Framework 2.3.15 started in: 15ms - bound to [/0.0.0.0:8080]
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: Portable JNDI names for EJB HelloWorldBean: [java:global/HelloWorldBean/HelloBean, ja
Informazioni: GlassFish-specific (Non-portable) JNDI names for EJB HelloWorldBean: (hello.HelloWorl
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] org.glassfish.sse.impl.ServerSentEv
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] private org.glassfish.jersey.gf.cdi
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] public org.glassfish.jms.injection.
Informazioni: HelloWorldBean was successfully deployed in 6.639 milliseconds.

```

20

HelloWorld EJB: Situazione sul Server

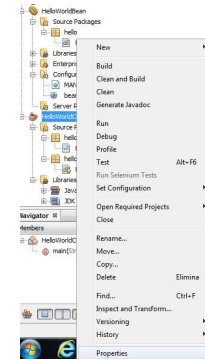
- Tab *Services*, icona *Servers*
- Sotto il *GlassFish Server* si scelgono le applicazioni
- Per ciascuna applicazioni si può abilitare, disabilitare, e fare undeploy



21

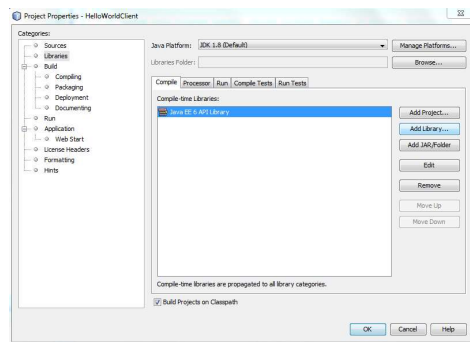
HelloWorld EJB: Il Client: La libreria

- Necessario caricare una libreria per usare con facilità GlassFish
 - non strettamente necessaria: possibile collegarsi con i parametri a qualsiasi server
 - serve a facilitare il testing
- La libreria si trova sotto la directory di GlassFish
 - sotto NetBeans
- GLASSFISH HOME/glassfish/lib/gf-client.jar



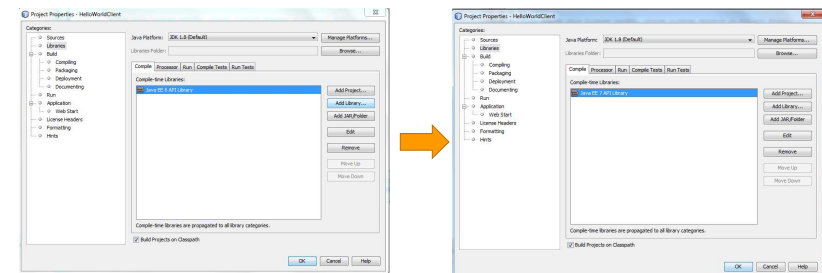
22

HelloWorld EJB: Setting della libreria lato client



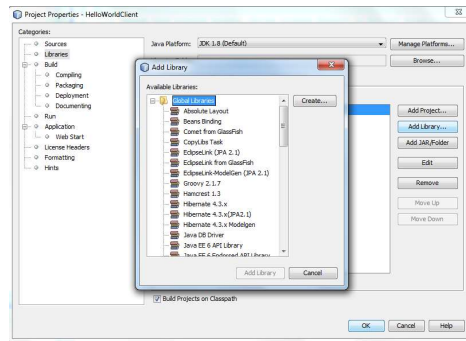
23

HelloWorld EJB: Se necessario cancelliamo la libreria EE6 e aggiungiamo EE7

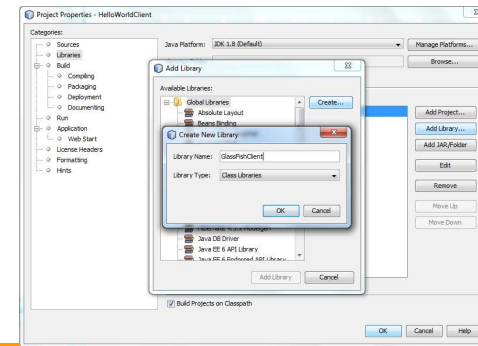


24

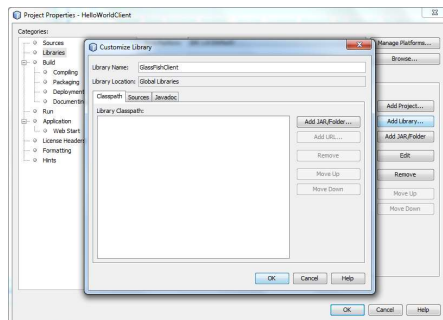
HelloWorld EJB: Creiamo una nuova libreria



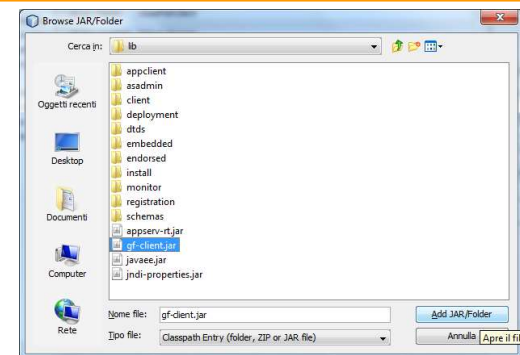
Nome della nuova libreria: GlassFishClient



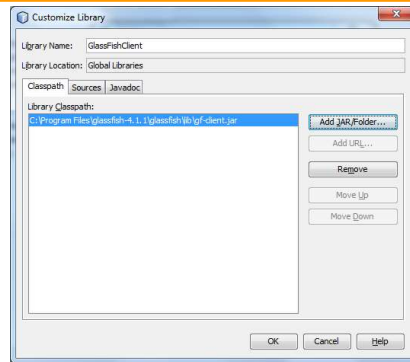
Sceita del file jar



Nella directory del Server: Glassfish/LIB

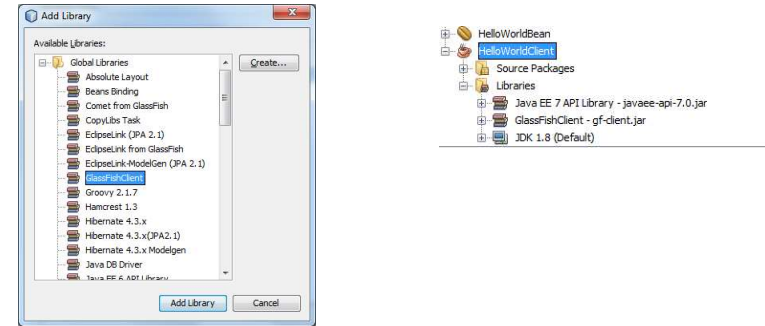


Ora possiamo aggiungerla



29

Situazione finale



30

Se non carica la libreria

```
run:
Exception in thread "main" javax.naming.NoInitialContextException: Need to specify class name in environment or system property, or as an applet parameter, or in an application resource file: javax.naming.factory.initial
    at javax.naming.spi.NamingManager.getInitialContext(NamingManager.java:662)
    at javax.naming.InitialContext.getDefaultInitCtx(InitialContext.java:313)
    at javax.naming.InitialContext.getURLorDefaultInitCtx(InitialContext.java:350)
    at javax.naming.InitialContext.lookup(InitialContext.java:417)
    at helloworldclient.HelloWorldClient.main(HelloWorldClient.java:30)

Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

31

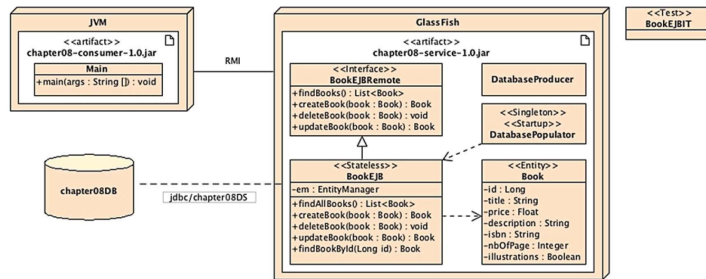
Esecuzione del client



32

Book EJB

- Esempio di servizio Book EJB (Capitolo 8)



33

Book EJB: Struttura del Progetto

- Struttura:

- java:
 - classi Book entity, BookEJB
 - Interface BookEJBRemote
 - classi DatabasePopulator e DatabaseProducer
- resources:
 - persistence.xml: file contenente le informazioni sulla *persistence unit* usata per il database Derby
 - beans.xml: fa trigger di CDI

34

Book EJB: l'entità Book

- Annotazione entità
- Query named per trovare tutti i libri
- ID generata
 - chiave primaria
- Titolo non vuoto

```
@Entity
@NamedQuery(name = FIND_ALL, query = "SELECT b FROM Book b")
public class Book implements Serializable {
    public static final String FIND_ALL = "Book.findAllBooks";
    @Id @GeneratedValue
    private Long id;
    @NotNull
    @Column(nullable = false)
    private String title;
    private Float price;
    @Size(max = 2000)
    @Column(length = 2000)
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    // Constructors, getters, setters
}
```

35

Book EJB: Struttura della logica di business

- EJB che gestisce le operazioni CRUD per la entità Book
- Metodi per:
 - trovare un libro
 - creare un libro
 - aggiornare un libro
 - cancellare un libro

36

Book EJB: l'EJB

- Bean stateless
- EM iniettato
- Un metodo (con validazione) per creare un libro persistente
- Un metodo (con validazione) per re-inserire un libro
 - precedentemente detached
- Interfaccia remota

```
@Stateless
@LocalBean
public class BookEJB implements BookEJBRemote
{
    @Inject
    private EntityManager em;

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

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

    public @NotNull Book updateBook(@NotNull Book book) {
        return em.merge(book);
    }

    public void deleteBook(@NotNull Book book) {
        em.remove(em.merge(book));
    }
}

@Remote
public interface BookEJBRemote {
    List<Book> findBooks();
    Book createBook(Book book);
    void deleteBook(Book book);
    Book updateBook(Book book);
}
```

37

Book EJB: un Producer per l'Entity Manager

- Per poter iniettare un EM con un parametro (la PU) necessario scrivere un **producer**
 - che produce un EM
 - scegliendo la PU (e il contesto) da utilizzare
- Ecco la entità generata

```
public class DatabaseProducer {
    @Produces
    @PersistenceContext(unitName = "EJB_Lab4PU")
    private EntityManager em;
}
```

38

Book EJB: il file persistence.xml

- **Transazioni** a carico del container
 - nessuna gestione da parte del bean
- **Data source**
- **Tipologia DB**
- **Ricrea** le tabelle
 - attenzione, cancella quelle esistenti

```
<?xmlversion="1.0"encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">

  <persistence-unit name="EJB_Lab4PU" transaction-type="JTA">
    <jta-data-source>
      <jta-data-source>
        <property name="eclipselink.target-database"
          value="DB999"/>
        <property name="eclipselink.ddl-generation"
          value="create-tables"/>
        <property name="eclipselink.logging.level" value="INFO"/>
      </properties>
    </persistence-unit>
  </persistence>
```

39

Come creare i dati

- Normalmente i dati vengono creati, una volta, all'inizio
- Gli EJB vengono magari aggiornati ma non vanno a ricreare i dati
 - in un esempio didattico, questo invece va fatto
- Creiamo in un DB un paio di libri
 - creiamo un EJB singleton il cui compito è quello di popolare il DB
 - quando termina (all'undeploy) viene effettuata la cancellazione del DB

40

Book EJB: Creazione del DB

- Singola istanza EJB
- Definizione dei dati
 - Il driver JDBC
 - Il nome globale (JNDI)
 - Il nome del DB
- La classe
 - logica di business EJB iniettata dal container
- Appena costruito, popola il DB
 - inserendo dei libri
- Alla fine li cancella

```
@Singleton
@Startup
@DataSourceDefinition(
    className = "org.apache.derby.jdbc.EmbeddedDataSource",
    name = "java:global/jdbc/EJB_Lab4DS",
    user = "app",
    password = "app",
    databaseName = "EJB_Lab4DB",
    properties = {"connectionAttributes=create=true"}
)
public class DatabasePopulator {
    @Inject
    private BookEJB bookEJB;
    private Book h2g2, lord;
    @PostConstruct
    private void populateDB() {
        h2g2 = new Book("Beginning Java EE", 35F, "Great book",
            "1-8763-9125-7", 605, true);
        lord = new Book("The Lord of the Rings", 50.4f, "SciFi",
            "1-84023-742-2", 1216, true);
        bookEJB.createBook(h2g2);
        bookEJB.createBook(lord);
    }

    @PreDestroy
    private void clearDB() {
        bookEJB.deleteBook(h2g2);
        bookEJB.deleteBook(lord);
    }
}
```

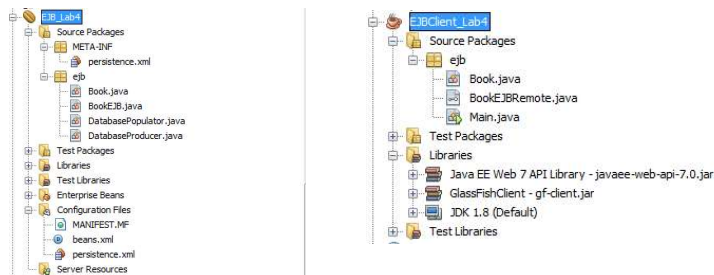
Book EJB: il client

- Lancia eccezione per lookup
- Contesto per la lookup
Lookup via JNDI
- Lista dei libri
- Creazione di un nuovo libro
 - passato all'EJB
- Cambiamento del titolo
 - e cancellazione

```
import java.util.*;
import javax.naming.*;
public class Main {
    public static void main(String[] args) throws NamingException {
        Context ctx;
        ctx = new InitialContext();
        BookEJBRemote bookEJB = (BookEJBRemote)
            ctx.lookup("java:global/EJB_Lab4/BookEJB!ejb.BookEJBRemote");
        List<Book> books = bookEJB.findBooks();
        for (Book aBook : books) {
            System.out.println("----" + aBook);
        }
        Book book = new Book("The Hitchhiker's Guide..", 12.5F,
            "Douglas Adams.", "1-24561-799-0", 354, false);
        book = bookEJB.createBook(book);
        System.out.println("###Book created:" + book);
        book.setTitle("H2G2");
        book = bookEJB.updateBook(book);
        System.out.println("###Book updated:" + book);
        bookEJB.deleteBook(book);
        System.out.println("###Book deleted");
    }
}
```

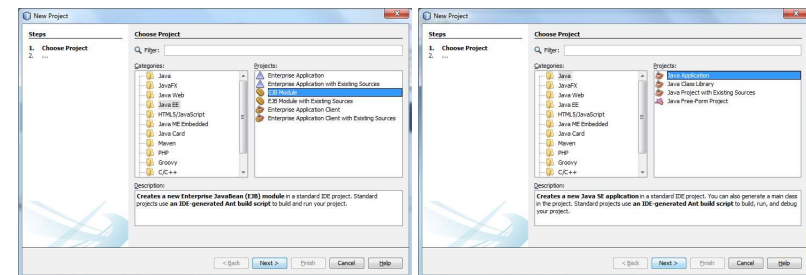
Book EJB: Pratica su NetBeans

- Due progetti



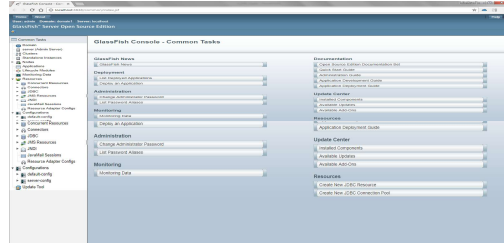
Book EJB: Passo 1: Creazione dei Progetti su NetBeans

- Seguendo i passi che abbiamo visto per HelloWorld EJB



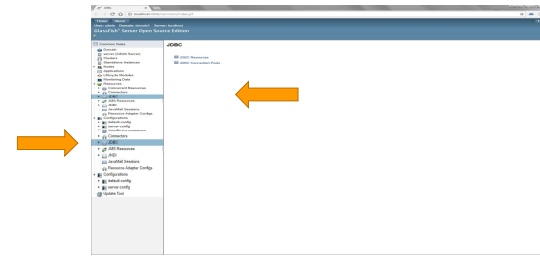
Book EJB: Passo 1: Creazione del database

- Da admin console (porta 4848)
 - creiamo un Connection Pool
 - creiamo un Data Source



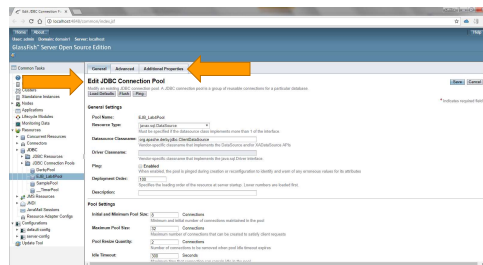
Book EJB: Passo 1: Creazione del database

- 1. Creiamo un Connection Pool
 - JBDC → JBD Connection Pool



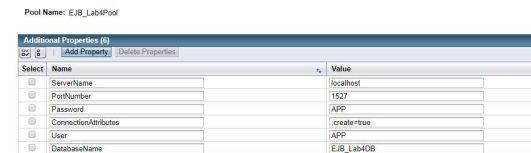
Book EJB: Passo 1: Creazione del database

- 1. Creiamo un Connection Pool
 - JBDC → JBD Connection Pool → Edit JDBC Connection Pool → Additional Properties



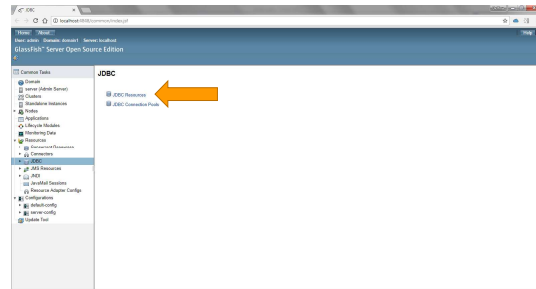
Book EJB: Passo 1: Creazione del database

- 1. Creiamo un Connection Pool
 - JBDC → JBD Connection Pool → Edit JDBC Connection Pool → Additional Properties



Book EJB: Passo 2: Creazione del database

2. Creiamo un Data Source



Book EJB: Passo 2: Creazione del database

2. Creiamo un Data Source

Edit JDBC Resource

Edit an existing JDBC data source.

[Load Defaults](#)

JNDI Name: jdbcEJB_Lab4DS

Pool Name: EJB_Lab4Pool

Use the JDBC Connection Pools page to create new pools.

Deployment Order: 100

Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

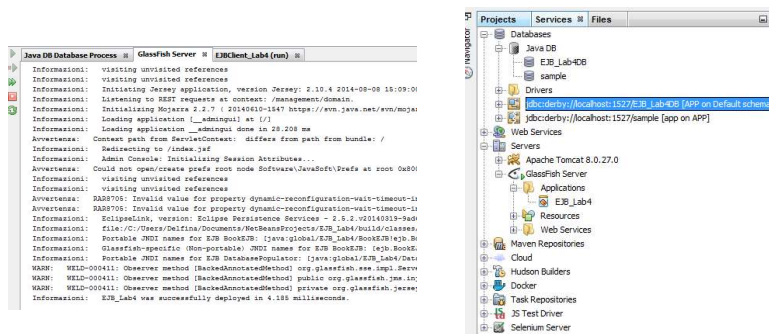
Status: ☒ Enabled

Additional Properties (0)

[Add Property](#) [Delete Properties](#)

Select	Name	Value	Description
No items found.			

Book EJB: Passo 3: Compilazione e deploy



Book EJB: Passo 4: Esecuzione

Codice client da eseguire

```

20 public static void main(String[] args) throws NamingException {
21     Context ctx;
22     BookEJBRemote bookEJB = (BookEJBRemote)ctx.lookup("java:global/EJB_Lab4/bookEJB!ejb.BookEJBRemote");
23
24     Book book1 = new Book("Qualcuno", 12.5F, "Book on Database and Math", "1-24561-799-0", 200, false);
25     book1 = bookEJB.createBook(book1);
26     System.out.println("Book 1 created: " + book1);
27
28     Book book2 = new Book("Computer Networks", 12.5F, "Networks", "1-24561-799-0", 400, false);
29     book2 = bookEJB.createBook(book2);
30     System.out.println("Book 2 created: " + book2);
31
32     System.out.println("Lista di tutti i libri");
33
34     List<Book> books = bookEJB.findBooks();
35     for (Book book : books) {
36         System.out.println("-----" + book);
37     }
38
39     System.out.println("Opzioni libro Book1 (bookEJB)");
40     book2.setAuthor("Computer Networks");
41     book2 = bookEJB.updateBook(book2);
42     System.out.println("Book 2 updated: " + book2);
43
44     System.out.println("Cancelliamo Book1 (bookEJB)");
45     bookEJB.deleteBook(book1);
46     System.out.println("Book 1 deleted: " + book1);
47
48     System.out.println("Lista di tutti i libri dopo la cancellazione di Book1");
49     List<Book> booksAfter = bookEJB.findBooks();
50     for (Book book : booksAfter) {
51         System.out.println("-----" + book);
52     }
53 }

```

Book EJB: Passo 4: Esecuzione

Output client

```

Output | Test Results
Java DB Database Process | ClassFish Server | EJBClientLab4 (run) |
---
##Book 1 created:Book[id=1, title="Statistics", price=12.5, description="Book on Statistics and Maths", isbn="1-24561-799-0", nbOfPage=200, illustration=
##Book 2 created:Book[id=2, title="Computer Networks", price=12.5, description="Networks", isbn="1-24561-799-0", nbOfPage=400, illustration=false]
Lista di tutti i libri:
---Book[id=1, title="Beginning Java ee", price=55.0, description="GreatBook", isbn="1-4324-43", nbOfPage=600, illustration=true]
---Book[id=2, title="Signore degli anelli", price=50.4, description="Fantasy", isbn="1-4342-221", nbOfPage=1216, illustration=true]
---Book[id=3, title="Statistic", price=12.5, description="Book on Statistics and Maths", isbn="1-24561-799-0", nbOfPage=200, illustration=false]
---Book[id=4, title="Computer Networks", price=12.5, description="Networks", isbn="1-24561-799-0", nbOfPage=400, illustration=false]
Aggiorniamo Book1 (Networks)
##Book1 updated:Book[id=4, title="Computer Networks II", price=12.5, description="Networks", isbn="1-24561-799-0", nbOfPage=400, illustration=false]
Cancelliamo Book2 (Networks)
##Book2 deleted.
Lista di tutti i libri dopo la cancellazione di Book2
---Book[id=1, title="Beginning Java ee", price=55.0, description="GreatBook", isbn="1-4324-43", nbOfPage=600, illustration=true]
---Book[id=3, title="Signore degli anelli", price=50.4, description="Fantasy", isbn="1-4342-221", nbOfPage=1216, illustration=true]
---Book[id=4, title="Statistics", price=12.5, description="Book on Statistics and Maths", isbn="1-24561-799-0", nbOfPage=200, illustration=false]
TOTAL SUCCESSFUL: total time: 20 seconds
1

```

Possibili errori

- Nome del progetto diverso dal nome del bean
- Ricordarsi del beans.xml

Conclusioni

- Introduzione agli EJB
- Esempio: HelloWorld EJB
- Esempio: Book EJB
- Conclusioni