Come creare un EJB:  
  
1) Cliccare su File > New > EJB Project  
2) Creare un EAR poiché a wildfly i progetti sparsi non piacciono (Crea l’EAR direttamente dalla tendina di creazione di EJB project.  
3) > Next > Fermiamoci alla tendina EJB module / Lui ci appenderà la parola Client (**modifichiamola** con la parola **Condiviso**), lui non è il client, ma la dipendenza da cui dipendono i client per poter parlare l’EJB server  
  
**Guarda il punto 3 e ricorda:**  
***"Lui non è il client"****:*

* *Il modulo che stai creando non è un client vero e proprio. Piuttosto, il JAR che viene generato è una libreria.*
* *Questa libreria contiene le interfacce remote o locali degli EJB. I client (che possono essere altre applicazioni Java, web app, ecc.) useranno questa libreria per comunicare con il server EJB.*

4) Clicchiamo su finish  
5) **ADESSO MI RACCOMANDO:** Verifica la versione Java della cartella ***\*\*\*Condiviso*** (la vecchia cartella Client) perché lui generalmente genera la versione di java “9”. [La versione si basa sulla versione che utilizzo di wildfly] *es, wildfly 12 arriva da Java 1.7 alla 9, la 27 + fino alla 23.*

**CREAZIONE DI UN SESSION BEAN**

All’interno della cartella (nel mio caso) libroEJB farò  
1) Tasto destro > New > Session Bean (3.x)  
2) Creiamo il package it.xxxx.ejb  
3) La classe la chiamiamo in modo inerente a ciò che vogliamo fare (Es: Saluto)  
4) All’interno di quella tendina mi verrà chiesto se avere; remote e local, io spunterò su entrambe le caselle. > vado su next, fine.

*In questo caso avrò l’annotazione* ***@Stateless****, identificando la natura dell’oggetto, osserviamo il LocalBean, ma non vuol dire che lui sia local poiché implementa l’interfaccia SalutoRemote, SalutoLocal.  
(In alcuni EJB che andremo a vedere ci sarà @remote, @local.)*

**ANDARE SU EJBCONDIVISO**

1. Apriamo la cartella ***EJBcondiviso (dentro it.exolab.ejb)*** in modo tale da vedere tutte le cartelle contenute al suo interno
2. Vediamo le due interfacce (dentro il package .ejb) le due interfacce generate in precedenza, nel mio caso la SalutoLocal.java e SalutoRemote.java.  
   Queste due interfacce verranno condivise con il client (Nel nostro caso il **client** che andremo a realizzare è una dynamic web project, quindi **servlet che vorrà parlare con esso.**).
3. Crediamo due metodi: Dentro SalutoRemote questo:

|  |
| --- |
| @Remote  **public** **interface** SalutoRemote {    String metodoPerClientRemoti();  ​  } |

1. Dentro SalutoLocal questo:

|  |
| --- |
| @Local  **public** **interface** SalutoLocal {    String metodoPerClientLocali();  ​  } |

Come ben sappiamo essendo interfacce, le classi che le implementano devono obbligatoriamente implementare tali metodi e quindi fare l’override, quindi cosa succede adesso? Che ***il client che cerca di parlare con il local chiamerà il metodo per local***, e quello che parla con il ***remote SOLO il metodo per remoti:***

***(Il local è più veloce rispetto all'accesso remoto poiché non comporta overhead di rete.)***

Le interfacce @Local e @Remote permettono una chiara separazione tra i client locali e remoti.

**La scelta dipende dai requisiti dell'applicazione:**

* Usa @Local per interazioni nella stessa JVM, come comunicazioni interne tra moduli di un'app Java EE.
* Usa @Remote quando il client e l'EJB risiedono in JVM diverse.

L'implementazione dei metodi permette di avere un comportamento personalizzato per ogni tipo di client. **Ma andiamo avanti.**

1. Dentro SalutaRemote implementiamo questo metodo Saluta (Che restituisce una stringa) e chiaramente dovrà essere overridato dentro Saluto.java

|  |
| --- |
| @Override  **public** String saluta() {  ​  **return** "ciao";  } |

La parte server finisce qui:  
  
**IMPLEMENTAZIONE DEL NOSTRO CLIENT (LA SERVLET)**

**Cosa fa questa servlet?**

Richiama il metodo presente nell’interfaccia e stamperà sulla console “ciao” ***(Guarda il metodo saluta sopra)***

1. Creiamo un Dynamic Web Project > chiamiamolo libroEJBClient (sarà il client vero e proprio)
2. Nella stessa tendina facciamoci generare anche l’EAR (libroEJBClientEAR) andiamo next [Attenzione, utilizzare sempre le versione di Java e wildfly inerente a quelle create con gli Ear ed i progetti precedenti]
   * + 1. *(Tecnicamente il gruppo di lavoro dovrebbe fare tasto destro su EJBCondiviso e mandarlo in allegato ai colleghi)*
3. Clicco tasto destro su EJBClientEAR > properties > Project references > referenzio il libroEJBCondiviso.
4. Adesso, sempre sul properties di **libroEJBClientEAR** vado su deployment assembly > project > xxxEJBCondiviso.

**Ma cosa vogliono dire questi due passaggi?**

I due passaggi servono a **collegare il progetto EJB condiviso (EJBCondiviso)** al progetto che utilizza i suoi componenti (EJBClientEAR).

1. **Project References**: Si assicura che il progetto EJBClientEAR riconosca EJBCondiviso come dipendenza, permettendo l'uso delle sue interfacce e classi.
   * È come dire: "Questo progetto dipende da quel codice, quindi rendilo disponibile."
2. **Deployment Assembly**: Include fisicamente il contenuto di EJBCondiviso nel processo di deploy di EJBClientEAR.
   * È come impacchettare EJBCondiviso insieme a EJBClientEAR per distribuirlo correttamente.

**LA SERVLET**

* 1. Su LibroEJBClient > New > Serlet: e la chiamiamo semplicemente “StampaSalutoServlet”
  2. Impostiamo come package “it.exolab.servlet”
  3. Cliccando next cambiamo il /StampaSalutoServlet in > /saluto.
  4. Adesso bisogna **LAVORARE SUL DOGET**
  5. Adesso istanzieremo un ***oggetto remoto***

|  |
| --- |
| **protected** void doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  ​  Properties props = **new** Properties();  ​  props.setProperty(Context.INITIAL\_CONTEXT\_FACTORY, "org.wildfly.naming.client.WildFlyInitialContextFactory");  ​  InitialContext ctx;  **try** {  ctx = **new** InitialContext(props);  String jndi = "java:global/libroEJBEAR/libroEJB/Saluto!it.exolab.ejb.SalutoLocal";  SalutoRemote ejb = (SalutoRemote) ctx.lookup(jndi);  ​  System.out.println(ejb.saluta());  ​  } **catch** (NamingException e) {  e.printStackTrace();  }  ​  } |

Ecco il nostro DoGet! Dobbiamo prendere il JNDI dalla console:  
  
**HO AVUTO UN PROBLEMA, TUTTE LE VERSIONI DEI PROGETTI DEVONO COMBACIARE EVITA DI USARE QUELLA DI DEFAULT DI ECLIPSE NON CI FIDIAMO**

**Cose da fare**

* 1. ProgettoClient > DWP FE > con dentro la servlet e la pagine JSP (anche la XML)
  2. Nella buildpath del client JSTL altri due per paura.
  3. Nei Facets controlla sempre la versione giusta sia di wildfly

Poi fare progetto Server è un EJB project dentro la ejbModul e ci metto DAO, MAPPER, SQLMAPFACTORY e L’EJB e dentro ci mettiamo session bean.  
  
dovro spostare local e remote nel condiviso.

Nel progetto server fuori da ejbModule in resources metto gli xml.  
  
il condiviso avrà interfacce model esso va mavenizzato.  
  
nel server va messo mybatis e mysql

Nella servlet istanzio le cose remote

utilizzerò l’ejb che mi richiamerà il dao