

**Tema**

ENTERPRISE JAVABEANS (EJBS) ENTERPRISE EDITION

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

15/07/2023

ENTERPRISE JAVABEANS (EJBS)

[1. MARCO TEÓRICO 4](#_Toc125390578)

[1.1. INTRODUCCIÓN 4](#_Toc125390579)

[1.2. OBJETIVOS 4](#_Toc125390580)

[**1.2.1.** **OBJETIVO GENERAL** 4](#_Toc125390581)

[**1.2.2.** **OBJETIVOS ESPECÍFICOS** 4](#_Toc125390582)

[1.3. ENTERPRISE JAVABEANS 5](#_Toc125390583)

[1.4. SESSION BEANS 8](#_Toc125390584)

[**1.4.1 STATELESS SESSION BEAN** 8](#_Toc125390585)

[**1.4.2 STATEFULL SESSION BEAN** 9](#_Toc125390586)

[**1.4.3 SINGLETON SESSION BEAN** 10](#_Toc125390587)

[1.5. SOFTWARE MODULAR 11](#_Toc125390588)

[1.6. SERVIDORES 12](#_Toc125390589)

[1.7. CLIENTE – SERVIDOR 14](#_Toc125390590)

[1.8. .WAR 15](#_Toc125390591)

[1.9. JAVA WEB START 16](#_Toc125390592)

[1.10. INTERFACES LOCALES Y REMOTAS 16](#_Toc125390593)

[2. PARTE PRÁCTICA 18](#_Toc125390594)

[2.1. CREACIÓN DEL PROYECTO 18](#_Toc125390595)

[2.2. CODIFICACIÓN DEL PROYECTO 25](#_Toc125390596)

[2.3. EJECUCIÓN DEL PROYECTO 31](#_Toc125390597)

[3. CONCLUSIONES 32](#_Toc125390598)

[4. RECOMENDACIONES 33](#_Toc125390599)

[5. REFERENCIAS 33](#_Toc125390600)

**INDICE DE IMÁGENES**

[Figura 1. Funcionamiento de Enterprise Java Bean 5](#_Toc125390609)

[Figura 2. Servlet Container 5](#_Toc125390610)

[Figura 3. EJB Container 6](#_Toc125390611)

[Figura 4. EJB container con petición de un cliente 6](#_Toc125390612)

[Figura 5. Pool B - Pool A 7](#_Toc125390613)

[Figura 6. Thread Pool 7](#_Toc125390614)

[Figura 7. Ejemplo: @Stateless 8](#_Toc125390615)

[Figura 8. Ejemplo de Statefull Bean 9](#_Toc125390616)

[Figura 9. Ejemplo de Singleton Bean 10](#_Toc125390617)

[Figura 10. ERP 11](#_Toc125390618)

[Figura 11. Posible composición de un software modular 12](#_Toc125390619)

[Figura 12. Funcionamiento de un servidor 13](#_Toc125390620)

[Figura 13. EJB -> (Cliente - Servidor) 14](#_Toc125390621)

[Figura 14. .WAR 15](#_Toc125390622)

[Figura 15. Ejemplo de EJBLocalHome y EJBLocalObject 16](#_Toc125390623)

[Figura 16. Ejemplo de implementación de arquitectura EJB 17](#_Toc125390624)

[Figura 17. Logotipo identificativo de NetBeans 18](#_Toc125390625)

[Figura 18. Selección de crear nuevo proyecto 18](#_Toc125390626)

[Figura 19. Selección de Enterprise Application 19](#_Toc125390627)

[Figura 20. Nombramiento del Proyecto 19](#_Toc125390628)

[Figura 21. Selección de Servidor y versión de Java EE 20](#_Toc125390629)

[Figura 22. Visualización de proyectos creados 20](#_Toc125390630)

[Figura 23. Selección de crear nuevo proyecto 21](#_Toc125390631)

[Figura 24. Selección de Enterprise Application Client 21](#_Toc125390632)

[Figura 25. Nombramiento de Proyecto JavaBeansCliente 22](#_Toc125390633)

[Figura 26. Selección de Aplicación Enterprise, Servidor y versión de Java EE 22](#_Toc125390634)

[Figura 27. Visualización de proyectos 23](#_Toc125390635)

[Figura 28. Selección de crear nuevo proyecto 23](#_Toc125390636)

[Figura 29. Selección de Java Class Library 24](#_Toc125390637)

[Figura 30. Nombramiento de proyecto 24](#_Toc125390638)

[Figura 31. Visualización de proyectos 24](#_Toc125390639)

[Figura 32. Creación de Session Bean 25](#_Toc125390640)

[Figura 33. Configuración de clase SessionBean.java 25](#_Toc125390641)

[Figura 34. Visualización de clases creadas .java 26](#_Toc125390642)

[Figura 35. Generar método de negocio 26](#_Toc125390643)

[Figura 36. Creación de método repetir 27](#_Toc125390644)

[Figura 37. Método repetir 27](#_Toc125390645)

[Figura 38. Modificación de retorno en el método repetir 28](#_Toc125390646)

[Figura 39. Añadir proyecto a la librería 28](#_Toc125390647)

[Figura 40. Selección de proyecto librería para que pueda usar el cliente 28](#_Toc125390648)

[Figura 41. Visualización de librería añadida 29](#_Toc125390649)

[Figura 42. Opción de insertar código 29](#_Toc125390650)

[Figura 43. Opción de Call Enterprise Bean 30](#_Toc125390651)

[Figura 44. Selección de bean Repetir 30](#_Toc125390652)

[Figura 45. Código de clase main generado 31](#_Toc125390653)

[Figura 46. Código de clase Main.java 31](#_Toc125390654)

[Figura 47. Proceso de ejecución de la aplicación Beans 32](#_Toc125390655)

[Figura 48. Ejecución del proyecto 32](#_Toc125390656)

# MARCO TEÓRICO

## INTRODUCCIÓN

En el presente informe se abordarán temas referentes a JAVA siendo más específicos sobre Enterprise JavaBeans (EJBs) en donde se puede observar la importancia de estas para entender como una aplicación empresarial negocia con el servidor, y observando la principal ventaja que tiene Java Enterprise Edition que es la construcción sobre componente de software modulares, y tomando en cuenta que el manejo del proyecto es bajo una arquitectura MVC (modelo, vista, controlador).

En este contexto, [12] puntualiza que una de las capas más trascendentales dentro del desarrollo de aplicaciones es la capa de lógica de negocios que es en donde estará el comportamiento o funcionalidad en sí de la aplicación, a su vez uno de los principales componentes con el que cuenta JEE enfocado a este ámbito, es la tecnología de Enterprise JavaBeans, que proporciona un estándar para el desarrollo de las clases que encapsulan la funcionalidad y reglas del negocio.

Finalmente, en el trabajo de [12] se especifica que los EJB nacen para encapsular la lógica de negocio de una forma integrada, no quedando dispersos su representación en un grupo de sistemas empresariales, los EJB están especialmente pensados para integrar la lógica de la empresa que se encuentra en sistemas distribuidos, de tal forma que el desarrollador ya no se tendrá que preocupar por la programación a nivel del sistema (control de transacciones, seguridad, etc). Por lo contrario, este deberá centrarse en la representación de entidades y reglas de negocio.

## OBJETIVOS

### **OBJETIVO GENERAL**

Comprender la importancia y la funcionalidad de cada uno de los Enterprise JavaBeans (EJBs) para comprender como una aplicación empresarial negocia con el servidor y de esta manera alcanzar un conocimiento práctico y teórico aplicable a futuros proyectos.

### **OBJETIVOS ESPECÍFICOS**

* Realizar un estudio teórico con respecto a los conceptos abordados dentro de la presente práctica.
* Reconocer las diferentes funcionalidades que tiene los JavaBeans (EJB’s), para aprender a aprovechar el máximo de las capacidades de cada uno de estos.
* Mejorar y ampliar las habilidades y conocimientos referentes al lenguaje de programación JAVA EE.
* Trabajar de manera organizada y manteniendo estándares de programación, de esta manera obteniendo buenas prácticas de programación en el desarrollo de aplicaciones.

## ENTERPRISE JAVABEANS

Los Enterprise Java Beans son a día de hoy componentes fundamentales en el desarrollo de aplicaciones Java Enterprise Edition . Sin embargo, muchas veces surgen dudas sobre cómo funciona un Enterprise Java Bean a detalle. Se tiene bastante clara la idea básica y es que normalmente un cliente realiza una petición al servidor, el servidor la procesa a través de un Servlet o un Managed Bean (JSF ) delegando en un EJB . El EJB recibe la petición invoca la funcionalidad de negocio correspondiente devolviendo un resultado que el Servlet/ Managed Bean termina presentando en una vista JSP /JSF page [2].

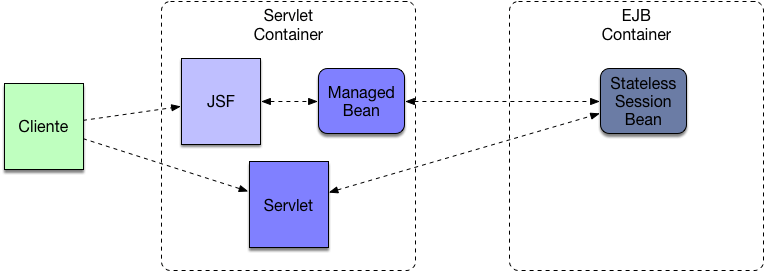


Figura 1. Funcionamiento de Enterprise Java Bean

Enterprise Java Beans y Pools

Los problemas comienzan con las preguntas más elementales. ¿Cuántas instancias de Servlet se tiene para gestionar varias peticiones de forma simultánea? . La realizad es que un servlet está diseñado fundamentalmente para no almacenar estado y poder ser accedido de forma concurrente existiendo una única instancia del Servlet a nivel de Servlet Container [2].

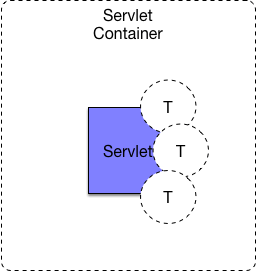


Figura 2. Servlet Container

¿Sucede lo mismo a nivel de EJBs? . ¿Existe un único EJB que se encarga de procesar todas las peticiones concurrentes? La realidad es que no . Los Enterprise Java Beans son configurados a través del concepto de Pool . Lo que implica que por cada EJB tipo de EJB existe un pool de EJBs que está disponible [2].

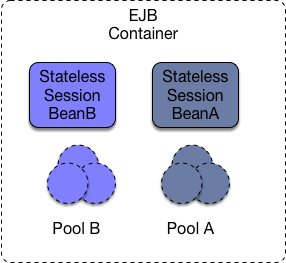


Figura 3. EJB Container

Cuando al EJB container le llega una petición desde un Cliente ( Servlet Container , o Swing) . Lo que hace el contenedor es sacar un EJB del pool y procesar la petición del cliente [2].

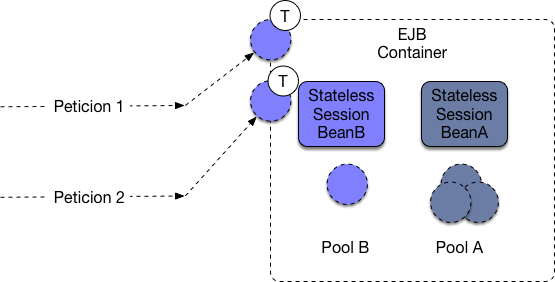


Figura 4. EJB container con petición de un cliente

De esta forma se optimiza el número de EJBs disponibles y se asegura que se reduce los problemas de concurrencia. Ya que cada EJB trabaja con un único thread luego estos regresan al pool [2].

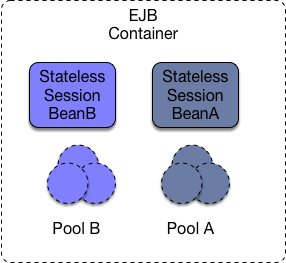


Figura 5. Pool B - Pool A

Para procesar estas peticiones el EJB container tendrá que apoyarse en su pool de threads ya que cada EJB obtendrá un hilo de ejecución distinta [2].

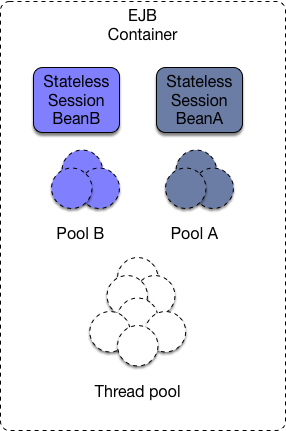


Figura 6. Thread Pool

Es así como funciona un Enterprise Java Beans tomando en cuenta cuando se trabaje con ellos. Esto es aplicable a Stateless Session Beans y Message Driven Beans que son los más utilizados . Los Statefull Session Beans y los EJB Singletons tienen comportamientos diferentes.

## SESSION BEANS

Un contenedor de aplicaciones JEE (JBoss, Glassfish, ..) puede controlar el ciclo de vida de una serie de clases java que lleven las anotaciones adecuadas. Entre estas clases, conocidas como Enterprise Java Beans o simplemente EJB, están los session beans, en concreto, Singleton, Statefull y Stateless. Se va a explicar aquí y ver un ejemplo sencillo de estos tres tipos de EJB [3].

Tienes el código de ejemplo de estos EJB en Github. Para ejecutarlo, tendrás que montar un proyecto EJB en tu IDE favorito y necesitarás algún contenedor de aplicaciones. En el proyecto, además de los tres EJB que se comentó aquí, hay uno de nombre MainBean, que no se explicara. Este MainBean sólo es un bean que hacer "cosas" con los otros tres EJB que sí explicara aquí. MainBean está solo por si se te ocurre descargar y desplegar el proyecto en un contenedor de aplicaciones, para que veas que hace algo [3].

### **1.4.1 STATELESS SESSION BEAN**

Un Stateless bean es una clase java sin estado, es decir, no debería tener atributos, sólo métodos a los que se pueda llamar de forma independiente. Para que el contenedor de aplicaciones lo maneje, únicamente hay que ponerle la anotación @Stateless, como en el siguiente ejemplo [3].

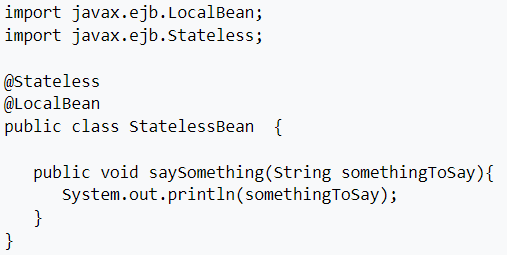


Figura 7. Ejemplo: @Stateless

Es una clase con un método al que se le pasas un texto y lo saca por pantalla. La anotación @Stateless es la que lo convierte en un EJB y le indica al contenedor de aplicaciones que debe encargarse de manejarlo. La anotación @LocalBean indica que a esta clase no se va a acceder de forma remota (desde fuera de la aplicación). La anotación es opcional, puesto que por defecto se entiende que el bean no va a ser accedido de forma remota [3].

¿Qué quiere decir que lo manejará el contenedor de aplicaciones? En el caso de Stateless beann, el contenedor de aplicaciones creará varias instancias de estas clase (se encargará de hacer los new) y le pasará estas instancias a quien las necesite. Como son clases sin estado, el contenedor dará cualquiera de las instancias que tiene al que la necesite y no se preocupará de entregar siempre la misma al mismo. Se vera más adelante, con la inyección de dependencias, como se entrega una instancia a otra clase que la necesite [3].

### **1.4.2 STATEFULL SESSION BEAN**

Un Statefull bean no es más que una clase en la que sí hay atributos cuyo valor es importante y debe conservarse entre llamadas a métodos. El contenedor de aplicaciones instanciará, al igual que con los Stateless bean, un número determinado de ellos y los reaprovechará. La diferencia es que el contenedor de aplicaciones se asegurará de dar siempre al mismo cliente/usuario la misma instancia. De esta forma, ese cliente/usuario puede guardar en él datos que sean particulares de él y que necesiten conservarse mientras está trabajando con la aplicación. Un ejemplo típico es el carrito de la compra de una web. Los productos que el usuario va añadiendo al carrito se guardarían en un Statefull bean [3]. En código, la siguiente clase:

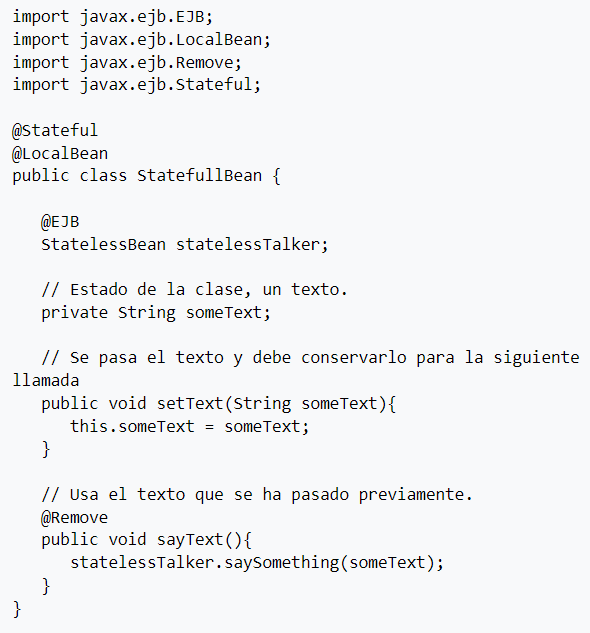


Figura 8. Ejemplo de Statefull Bean

A esta clase, para convertirla en un EJB, Se ha puesto la anotación @Stateful. La anotación @LocalBean, al igual que antes, es opcional e indica que este bean no se va a utilizar de forma remota.

El atributo someText hace las veces de estado de esta clase. Un método setText() le da un valor y este texto debe conservarse para la llamada a sayText(). Por eso este bean es Statefull, lo que hace sayText() depende de la llamada previa a setText().

La anotación @Remove en el método sayText() debe ponerse para indicar al contenedor que ya se a terminado de usar este Statefull bean. A partir de esta llamada, el contenedor sabe que este bean ya ha sido utilizado completamente y que puede dárselo a otro que lo necesite.

¿Y qué es el atributo @EJB StatelessBean statelessTalker; Se quiere que el Statefull bean utilice la clase StatelessBean para sacar por pantalla el texto. Como StatelessBean es un EJB controlado por el contenedor de aplicaciones, no se realiza el new de esta clase. Simplemente se anota con @EJB y el contenedor se encargará de pasar una de las instancias que tiene de StatelessBean que se podrá usar. Esto es lo que se comentaba de "inyección de dependencias". La clase EJB necesita otra clase EJB, así que basta con declarar el atributo y añadirle al anotación @EJB. El contenedor de aplicaciones se encargará de pasárnosla.

### **1.4.3 SINGLETON SESSION BEAN**

Un Singleton bean es una clase Java que el contenedor de aplicaciones instanciará una única vez y será compartida por todo el mundo. Está pensada para guardar los datos que son comunes a todos los usuarios o clientes de la aplicación [3]. El siguiente código de ejemplo tiene un Singleton bean:

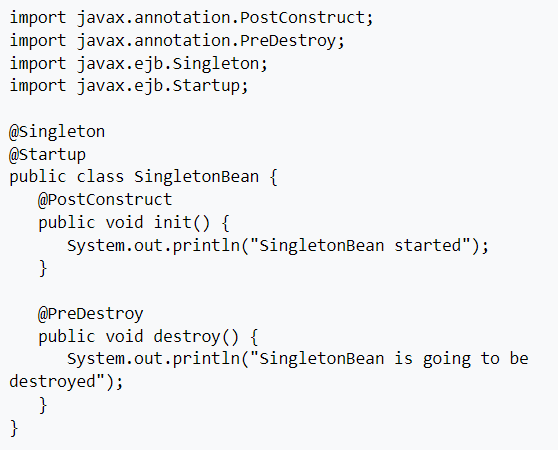


Figura 9. Ejemplo de Singleton Bean

La anotación @Singleton convierte a esta clase Java en un Singleton bean. No ha puesto la anotación @LocalBean puesto que por defecto el bean el local (no se puede acceder a él remotamente) si no se pone.

Se ha puesto la anotación @Startup para indicar al contenedor de aplicaciones que debe arrancar este bean al inicio. Si no se pone la anotación, el bean no será arrancado hasta que alguien lo necesite. Habitualmente no se pone @Startup, salvo que este bean realice algo que necesite ser hecho en el momento de arrancar el contenedor de la aplicación.

¿Y qué es lo que va a ejecutar este bean cuando lo arranquen? Pues lo que tenga el método que se anote con @PostConstruct. Si no hay ningún método con esta anotación, no se hará nada en el arranque, pero si hay algún método anotado de esta manera, se ejecutará después de cargar el bean, pero antes de que empiece a usarse. Viene a ser como un constructor o método de inicialización de la clase.

Si se quiere realizar alguna acción cuando el bean se vaya a destruir (al finalizar la aplicación), se debe anotar el método que se quiera que se ejecute con @PreDestroy.

## SOFTWARE MODULAR

Un software modular es un sistema que se compone de distintas funcionalidades que pueden agregarse según se requieran. De esta forma, solo se paga por lo que se necesita y se utilizará. Los módulos de los que se puede componer el software pueden ser funcionalidades comunes o, en algunos casos, incluir funcionalidades específicas para un tipo de industria [4].



Figura 10. ERP

Con este tipo de sistemas se entiende que las empresas tienen distintos departamentos que se van incorporando al sistema según sea necesario. Los módulos pueden incluirse al implementar el software o un tiempo después, lo que permite una gran escalabilidad. Es decir, proporciona una gran flexibilidad para aquellas empresas que crecen [4].

Esta escalabilidad es la que permite que las empresas puedan empezar rápidamente con la implementación y estar operativos antes con el nuevo sistema. Por eso, muchas empresas que se decantan por un software modular establecen un plan de implementación por fases. Así, consiguen trabajar rápidamente con los módulos más importantes para su actividad empresarial, y luego ir incorporando el resto paulatinamente [4].

Si una empresa lo desea, se pueden ampliar o disminuir las funcionalidades contratadas del software para ajustarse a requisitos, aunque la disminución de las funcionalidades contratadas puede estar sujeta a condiciones de contrato [4].

Por ejemplo: una empresa ha firmado por la contratación de los módulos de finanzas, recursos humanos, ventas, CRM y gestión documental durante un periodo de un año. Sin embargo, al pasar un par de meses, se da cuenta de que en verdad no necesitan el software de recursos humanos. Depende de lo que ponga en el contrato, puede ser más rentable mantenerlo o quitarlo. Ya que puede haber una cláusula que diga que, si quieres quitar un módulo, hay que pagar una multa de 2000€ y puede que la multa sea más que lo que se paga por el módulo en sí.

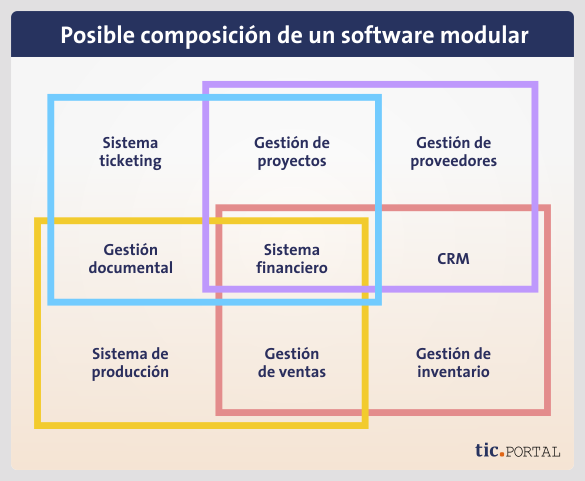


Figura 11. Posible composición de un software modular

## SERVIDORES

Es un aparato informático que almacena, distribuye y suministra información. Los servidores funcionan basándose en el modelo “cliente-servidor”. El cliente puede ser tanto un ordenador como una aplicación que requiere información del servidor para funcionar. Por tanto, un servidor ofrecerá la información demandada por el cliente siempre y cuando el cliente esté autorizado. Los servidores pueden ser físicos o virtuales [5].

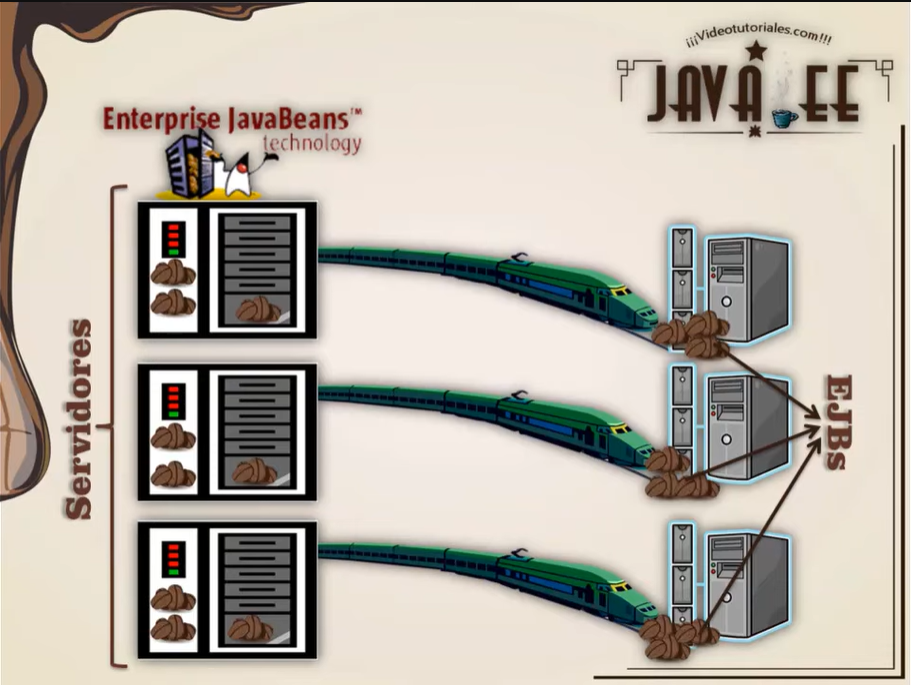


Figura 12. Funcionamiento de un servidor

Tipos de servidores

Dentro del mundo de los servidores hay una gran variedad. Los más usados y/o conocidos son los siguientes:

* **Servidor Web:** almacena y organiza el contenido de las páginas web y se lo proporciona al usuario a través del navegador web del usuario. La transmisión de los datos se suele realizar con http (HyperText Transfer Protocol). El http se encarga de transmitir la información de la WWW (World Wide Web), estableciendo unos criterios semánticos y sintácticos para que el ordenador solicitante y el que contiene la información tengan un idioma común en el que comunicarse. Los servidores web más conocidos son: Apache web server; Nginx; y Microsoft [5].
* **DNS:** son las siglas anglosajonas de Domain Name Server, aunque la “S” también puede referirse a service o space sin que cambie el significado ni la función del servidor. El dominio es el nombre que recibe una página web. El servidor DNS se encarga de relacionar una dirección de dominio (www.ejemplodominio.es) con su dirección IP correspondiente (00.000.000.00) [5].
* **Proxy Server:** es un servidor de puerta de entrada. Se encarga de conectar una red cliente (navegador web o aplicación) con un sistema externo para que se puede llevar a cabo la solicitud de conexión, mejor rendimiento y accesibilidad. Es decir, la conexión se beneficia del caché que almacena, haciendo que una página ya visitada por el usuario se cargue más rápido y reduciendo el ancho de banda de la red. Por ejemplo, cuando se suele mandar un tipo de factura a ciertos destinatarios, el sistema propone autocompletar uno o varios campos con los datos de aquellos destinatarios recurrentes [5].
* **Servidor de correo electrónico:** se encarga del flujo de correo electrónico de los usuarios, permitiendo que se almacene, envíe, reciba y reenvíe los e-mails [5].
* **Servidor FTP:** el nombre procede de File Transfer Protocol, también conocido como protocolo de transferencia de archivos en español. Sirve para transferir archivos entre un cliente y un servidor. De esta forma, el servidor puede recibir archivos del cliente y le posibilita la descarga de los archivos a los clientes. Para mejorar la velocidad, se usa SFTP el cual también ofrece un servicio de seguridad encriptada [5].

## CLIENTE – SERVIDOR

La expresión cliente servidor se utiliza en el ámbito de la informática. En dicho contexto, se llama cliente al dispositivo que requiere ciertos servicios a un servidor. La idea de servidor, por su parte, alude al equipo que brinda servicios a las computadoras (ordenadores) que se hallan conectadas con él mediante una red [6].

El concepto de cliente servidor, o cliente-servidor, refiere por lo tanto a un modelo de comunicación que vincula a varios dispositivos informáticos a través de una red. El cliente, en este marco, realiza peticiones de servicios al servidor, que se encarga de satisfacer dichos requerimientos.

Con esta arquitectura, las tareas se distribuyen entre los servidores (que proveen los servicios) y los clientes (que demandan dichos servicios). Dicho de otro modo: el cliente le pide un recurso al servidor, que brinda una respuesta.

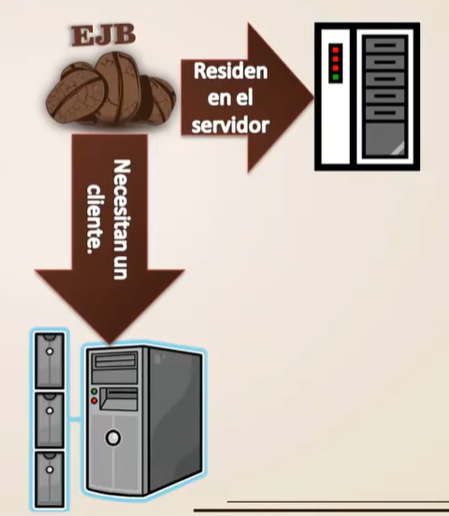


Figura 13. EJB -> (Cliente - Servidor)

Dirección Servidor

Una dirección de servidor es otro nombre para un protocolo de Internet o una dirección IP. Cada computadora tiene su propia dirección IP, y cuando una persona escribe el nombre de un sitio web en un navegador, realmente se le dirige a la dirección IP asociada con el nombre de ese sitio web.[7]

Si una persona usa Windows y quiere encontrar la dirección de su servidor, necesita abrir el panel de control de la computadora. En "red e Internet", debe seleccionar "ver el estado y las tareas de la red". Luego, haga clic en el nombre del enlace de su conexión a Internet. En este punto, aparece una ventana emergente con información sobre la conectividad y la intensidad de la señal, y la persona debe hacer clic en el botón "detalles". En la sección de detalles, el grupo de números después de las palabras "Dirección IPv4" es la dirección del servidor. [8]

## .WAR

Una aplicación Web es un grupo de páginas HTML, páginas JSP, servlets, recursos y archivo fuente, que se puede gestionar como una unidad. Un archivo WAR (Web Archive) es una aplicación Web empaquetada. Los archivos WAR se pueden utilizar para importar una aplicación Web a un servidor Web [9].

Además de los recursos del proyecto, el archivo WAR incluye un archivo de descriptor de despliegue Web. El descriptor de despliegue Web es un archivo XML que contiene información de despliegue, tipos MIME, detalles de configuración de sesión y otros valores de una aplicación Web. El archivo de descriptor de despliegue Web (web.xml) proporciona información sobre cómo se comparte el archivo WAR con los desarrolladores, ensambladores y desplegadores en un entorno Java EE [9].



Figura 14. .WAR

## JAVA WEB START

El software de Java Web Start permite descargar y ejecutar aplicaciones Java desde la Web. El software de Java Web Start [10]:

* Permite activar las aplicaciones con un simple clic
* Garantiza que se está ejecutando la última versión de la aplicación
* Elimina complejos procedimientos de instalación o actualización

## INTERFACES LOCALES Y REMOTAS

Hasta ahora se ha considerado siempre que el cliente y el bean se encuentran en distintas máquinas virtuales. Por eso es necesario RMI para conectar el cliente y el EJBObject. Pero ¿por qué introducir RMI cuando el cliente y el EJBObject se encuentran en la misma JVM? La introducción de RMI en la arquitectura la dota de flexibilidad, pero también le añade penalización en el rendimiento, debido sobre todo a la necesidad de serializar todos los argumentos y llamadas. Esta penalización no está justificada cuando el objeto cliente del bean es, por ejemplo, un servlet o una página JSP que reside en la misma JVM que el bean. Tampoco está justificado el uso de RMI cuando se están comunicando dos beans que residen en la misma JVM [11].

La especificación 2.0 de EJB propone el uso de las interfaces locales EJBLocalHome y EJBLocalObject como una solución a estas situaciones. A la hora de programar el bean lo único que cambia es que las interfaces home y componente deben heredar de EJBLocalHome y EJBLocalObject. Estas interfaces ya no son remotas, por lo que los métodos no van a tener que declarar la excepción RemoteException. En el nombre de las interfaces suelen añadirse la palabra local, para indicar que se tratan de objetos que van a llamarse sin usar RMI. En el caso del bean Saludo, se llamaría SaludoLocal a la interfaz componente y SaludoLocalHome a la interfaz home [11].

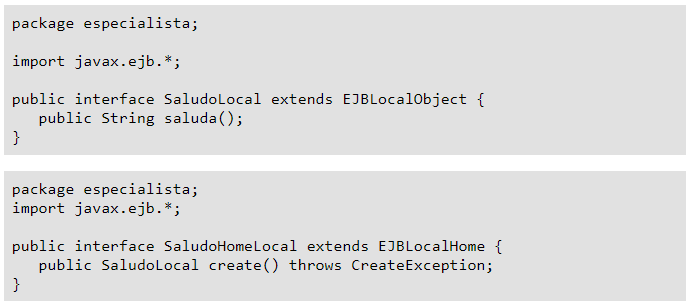


Figura 15. Ejemplo de EJBLocalHome y EJBLocalObject

En la arquitectura EJB, la única diferencia es que las llamadas entre el cliente y el EJBObject serían llamadas normales entre objetos Java y no tendrían que ser serializadas. Pero el objeto EJBObject seguiría haciendo las mismas funciones de proteger a los beans del exterior y de incorporar los servicios del contenedor EJB [11].

Por último, en el lado del cliente existen dos detalles que hay que modificar cuando se están utilizando interfaces locales. En primer lugar, las interfaces locales no declaran la excepción RemoteException, por lo que no es necesario que el cliente capture estas excepciones con un try/catch. El segundo aspecto tiene que ver con el casting del objeto home que se obtiene de JNDI. En el caso de que el objeto home sea remoto, se observa en la sesión anterior que antes de hacer el casting había que convertirlo en un objeto Java con la llamada al método PotableRemoteObject.narrow() [11]. En el caso de estar trabajando con interfaces locales no es necesario hacer esta conversión con lo que para hacer el casting bastaría con hacer:

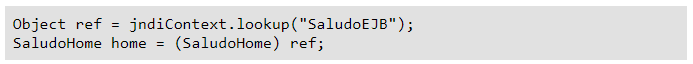


Figura 16. Ejemplo de implementación de arquitectura EJB

Se termina con unas breves consideraciones que pueden ayudar a determinar si conviene usar interfaces locales o remotas:

* El funcionamiento de los beans usando interfaces locales es más eficiente ya que todas las llamadas al interfaz componente se realiza en la misma JVM y los parámetros no tienen que ser copiados ni serializados [11].
* El uso de interfaces locales modifica la semántica de la aplicación, ya que hace que los objetos que se pasan en las llamadas a los beans se pasen por referencia en lugar de por valor. Una modificación de ese objeto en la implementación del método del bean tendría un efecto lateral en el cliente. Estos efectos laterales son los responsables de multitud de bugs muy difíciles de localizar [11].
* El funcionamiento de las interfaces locales sólo es posible cuando los clientes de los beans van a estar en la misma JVM que los beans. Por ello, sólo se podría aplicarlas a llamadas a los beans desde servlets o páginas JSP, pero no desde clientes independientes implementados con Swing o desde aplicaciones independientes para realizar tareas de integración [11].
* El uso de interfaces locales va a hacer mucho más rígida la arquitectura de la aplicación, ya que obliga a que los beans residan en la misma máquina. El uso de interfaces remotas permite una aplicación distribuida en distintas máquinas, lo que facilita la posible aplicación de técnicas de escalabilidad de la aplicación basadas en el clustering de contenedores EJB [11].
* El uso de interfaces locales permite en los beans de entidad usar relaciones entre beans gestionadas por el contenedor, cosa que no es posible cuando se usan interfaces remotas [11].

# PARTE PRÁCTICA

## CREACIÓN DEL PROYECTO

Se abre el IDE de desarrollo NetBeans



Figura 17. Logotipo identificativo de NetBeans

Una vez abierto el IDE de desarrollo se hace clic en New Project (Nuevo Proyecto)

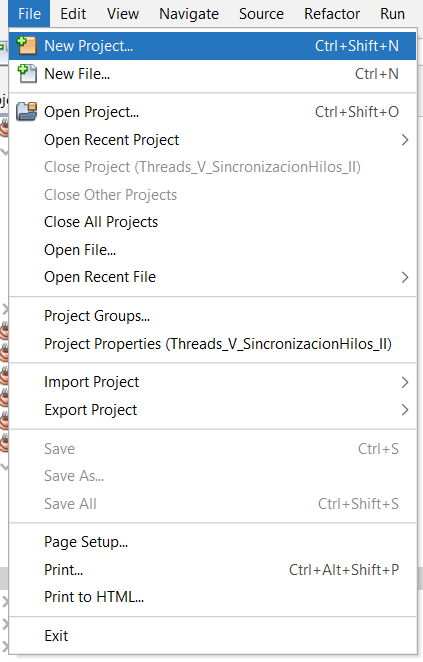


Figura 18. Selección de crear nuevo proyecto

Se selecciona el tipo de proyecto que se va a crear, que en este caso será un Enterprise Application

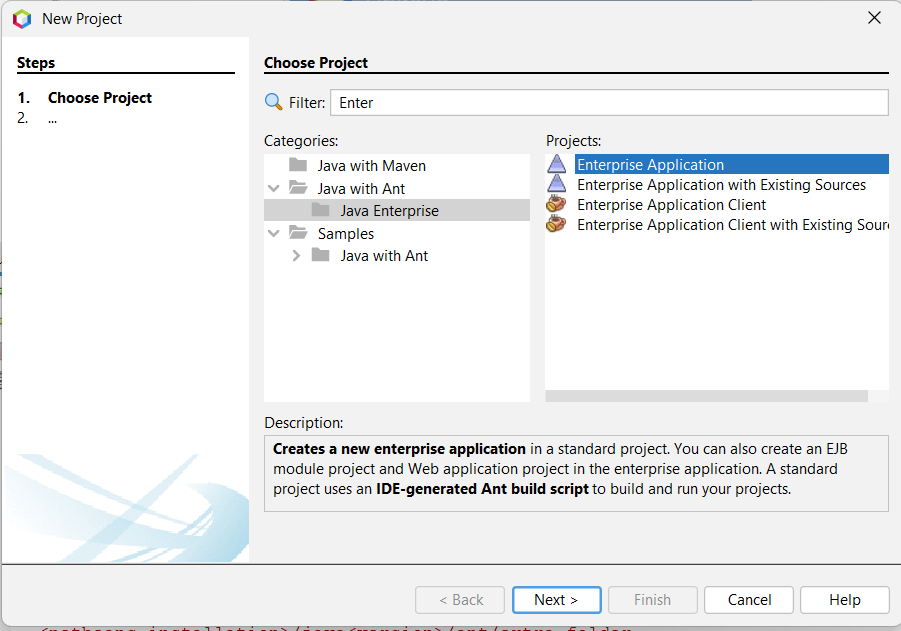


Figura 19. Selección de Enterprise Application

Se procede a nombrar el proyecto y a elegir la ruta en donde se desea guardar el proyecto y se da next

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 20. Nombramiento del Proyecto

Se procede a seleccionar el servidor que es Payara y la versión del JAVA EE que será la 6 , además el Create EJB Module debe estar marcado con un visto y en el Create Web Application se debe quitarle el visto y ahora si se hace clic en Finish.

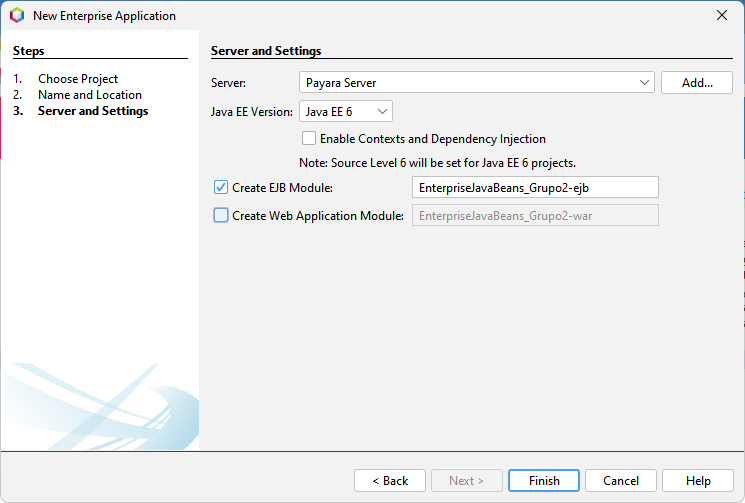


Figura 21. Selección de Servidor y versión de Java EE

Una vez dado clic en finish se crea dos proyectos en la ventana de proyectos de la figura 22.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 22. Visualización de proyectos creados

Ahora como se necesita un cliente, ahora se creara un nuevo proyecto para el cliente. Para realizarlo se hace clic de nuevo en New Project (Nuevo Proyecto)

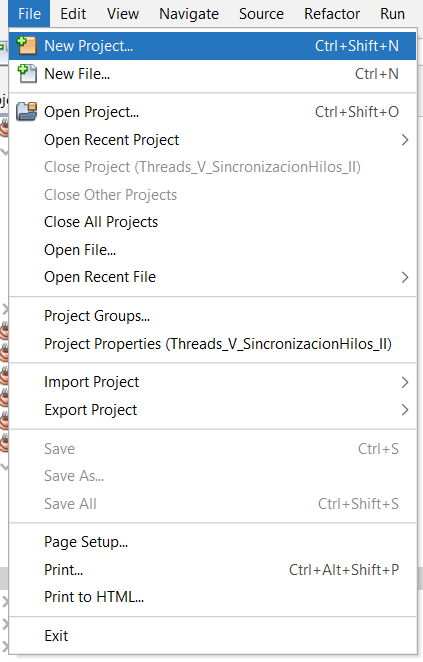


Figura 23. Selección de crear nuevo proyecto

Se selecciona el tipo de proyecto que se va a crear, que en este caso será un Enterprise Application Client.

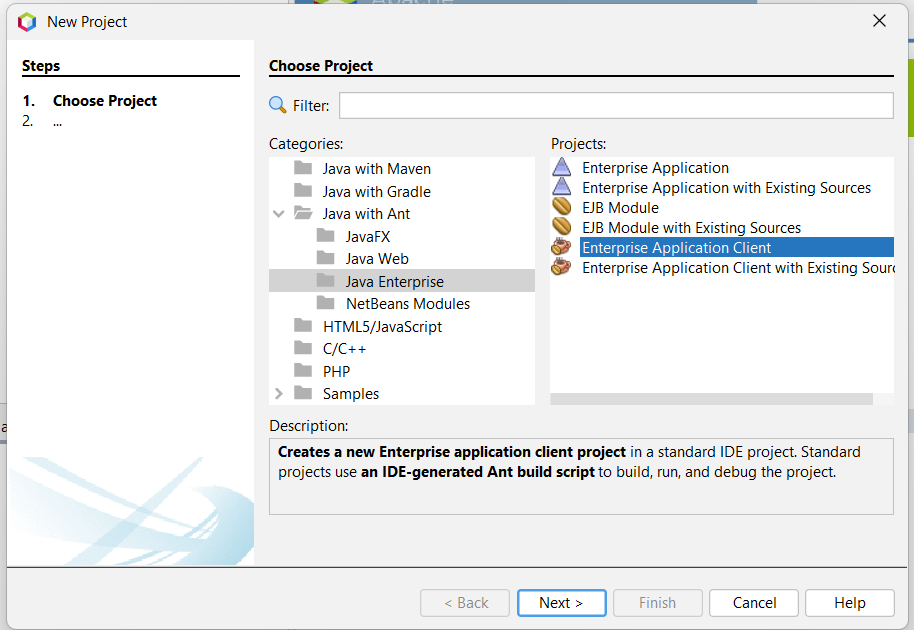


Figura 24. Selección de Enterprise Application Client

Se procede a nombrar el proyecto y a elegir la ruta en donde se desea guardar el proyecto y se da en next

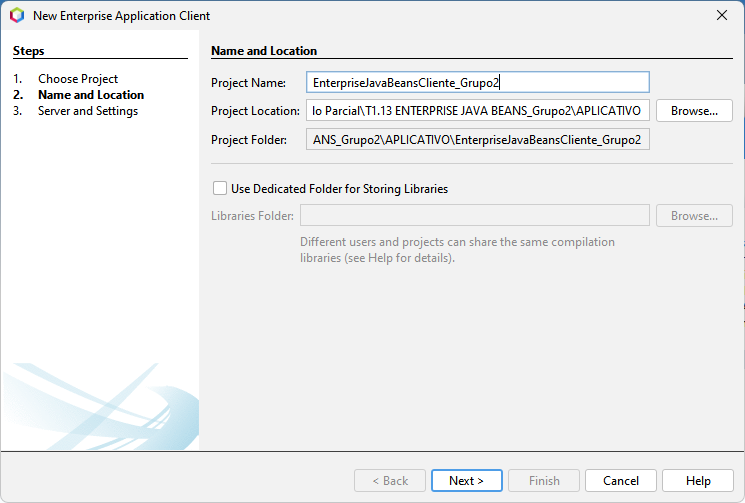


Figura 25. Nombramiento de Proyecto JavaBeansCliente

Ahora se procede a añadir la aplicación Enterprise a la que se creó anteriormente llamada EterpriseJavaBean\_Grupo1 en el caso presente, se selecciona el servidor de payara y la versión del Java EE que se va a trabajar con la versión 6.

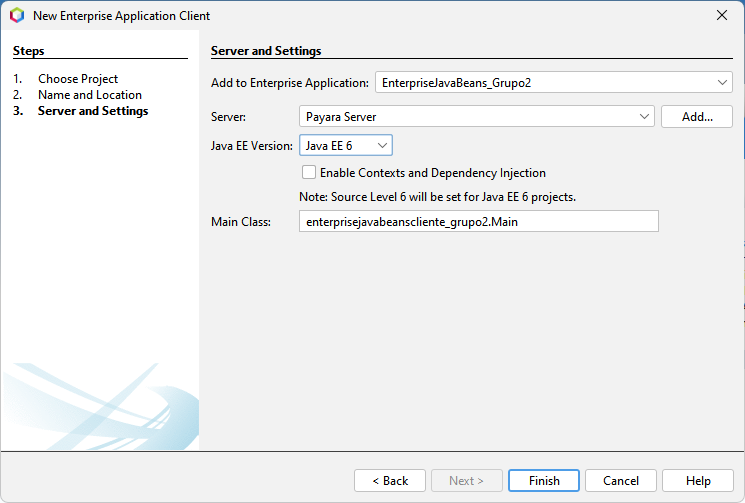


Figura 26. Selección de Aplicación Enterprise, Servidor y versión de Java EE

Ahora ya se tiene 3 proyectos creados como se ve en la figura 27

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 27. Visualización de proyectos

Ahora se necesita un proyecto que se va a necesitar mas adelante un proyecto de librerías de clases. Para crearlo se hace clic nuevamente en nuevo proyecto>Java>Java Class Library

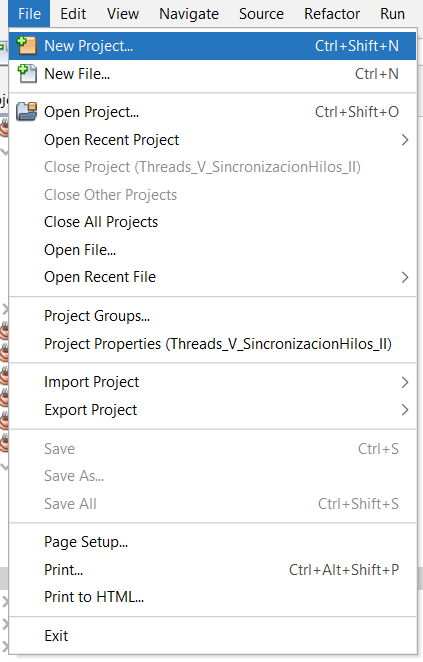


Figura 28. Selección de crear nuevo proyecto

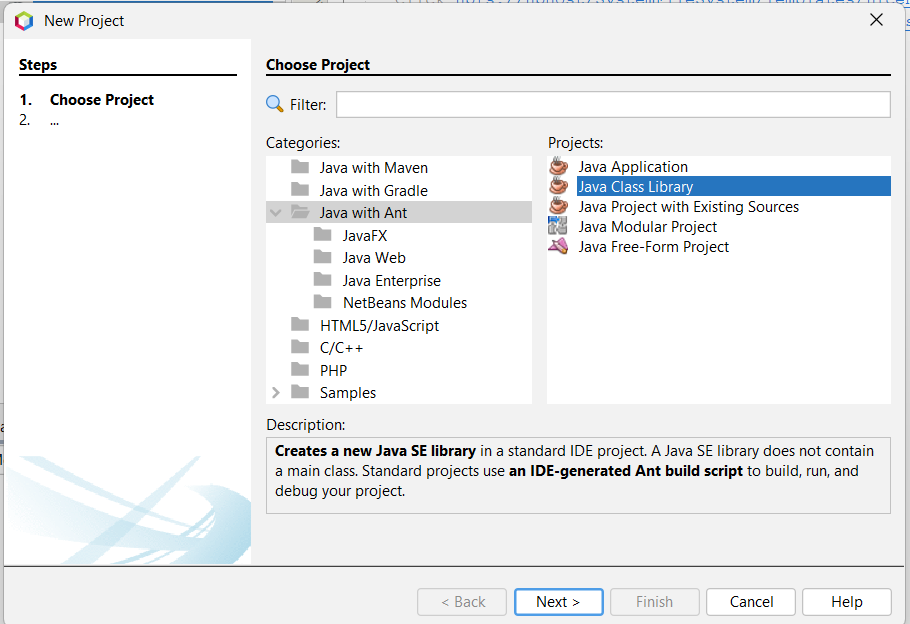


Figura 29. Selección de Java Class Library

Se procede a nombrar el proyecto de la librería y se hace clic en Finish

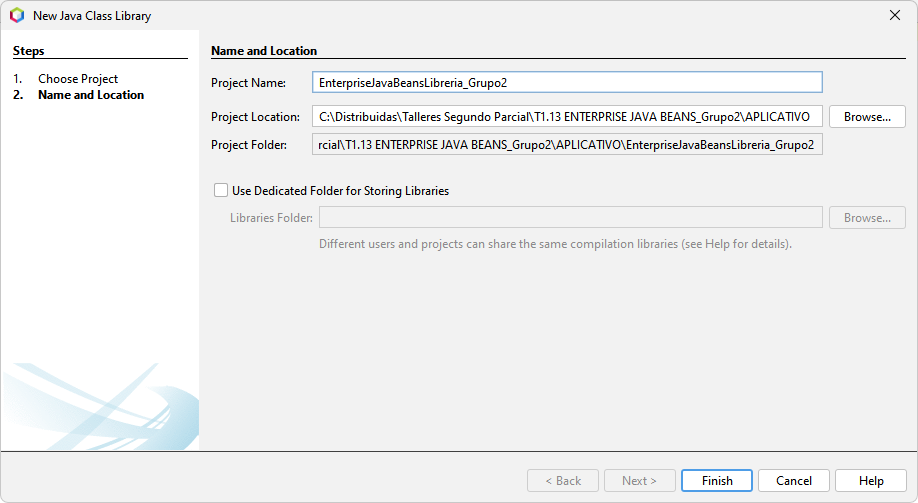


Figura 30. Nombramiento de proyecto

Y así ya se tiene el cuarto proyecto creado

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 31. Visualización de proyectos

## CODIFICACIÓN DEL PROYECTO

Ahora ya se está preparado para crear el primer SessionBean, se hace clic en el proyecto -ejb>new>Enterprise JavaBeans>Session Bean

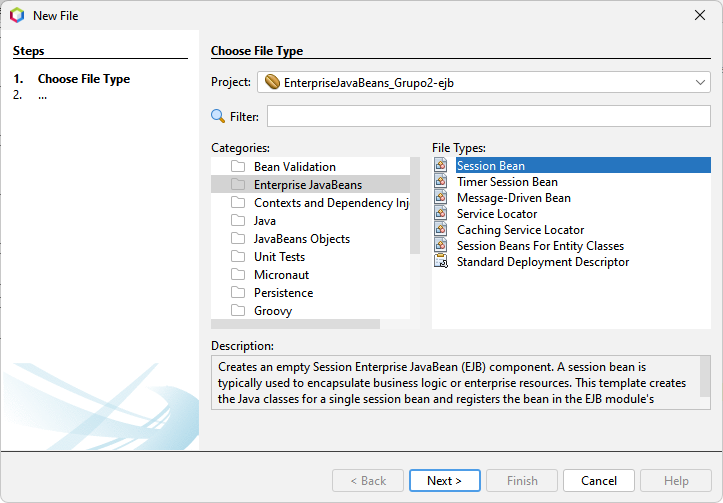


Figura 32. Creación de Session Bean

Una vez seleccionada la clase SessionBean.java se procede a nombrarla y a ubicarla en que paquete es que se desea crear, también se debe elegir cual va a ser el tipo de sesión que se va a utilizar en este caso Stateless y la interfaz va a ser remota y por defecto se selecciona el proyecto librería que se creó.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 33. Configuración de clase SessionBean.java

Y se observa que se crean los diferentes archivos en los diferentes proyectos

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Figura 34. Visualización de clases creadas .java

Ahora para generar el método de negocio en la clase Repetir se hace clic derecho>Insert Code>Add Business Method

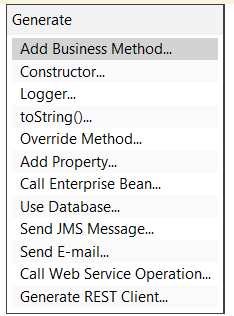


Figura 35. Generar método de negocio

Y para crear el método repetir va a llenar la pantalla que se abre de la siguiente manera

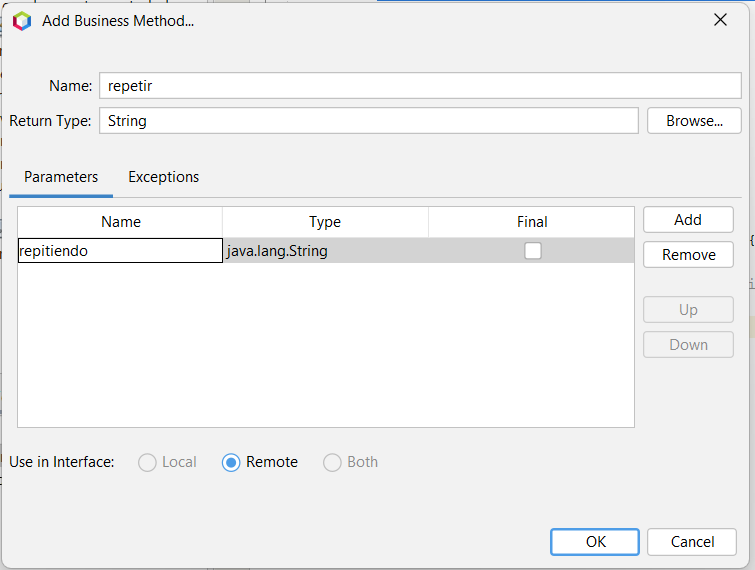


Figura 36. Creación de método repetir

Y se genera el siguiente código

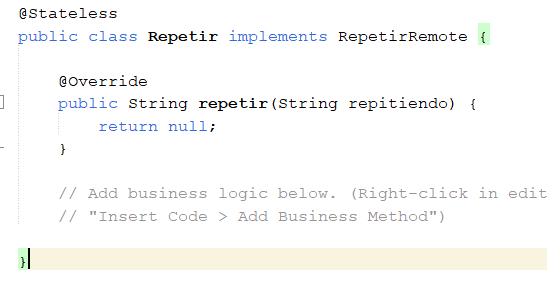


Figura 37. Método repetir

Ahora se va a modificar la el método la parte en que retorna un null y se va poner de la siguiente manera:

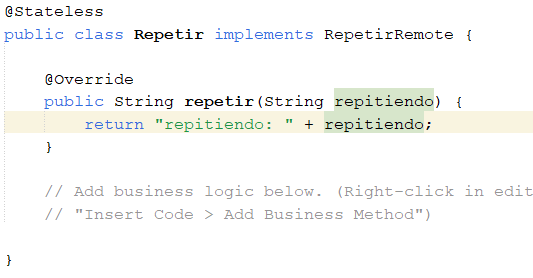


Figura 38. Modificación de retorno en el método repetir

Ahora lo que va hacer es que el cliente pueda usar la librearía, para realizarlo, dentro del proyecto del cliente creado anteriormente se hace clic en Libraries>Add Project > y selecciona el proyecto Librería que se había creado.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 39. Añadir proyecto a la librería

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 40. Selección de proyecto librería para que pueda usar el cliente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 41. Visualización de librería añadida

Ahora se va a invocar desde el cliente al método repetir que se tiene en SessionBean.  
Para hacerlo en el proyecto cliente se abre el archivo Main.java y se realiza lo siguiente: Se hace clic derecho en la clase>Insert code> y se selecciona Call Enterprise Bean

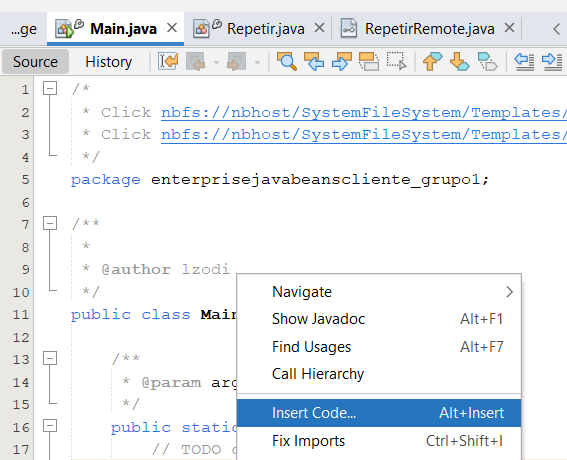


Figura 42. Opción de insertar código

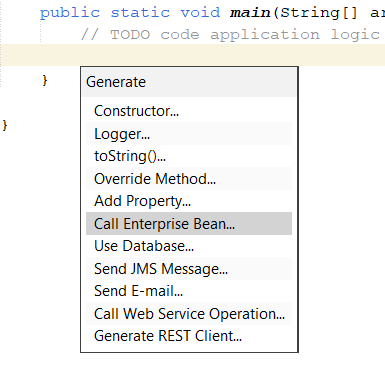


Figura 43. Opción de Call Enterprise Bean

Y se selecciona el bean de repetir

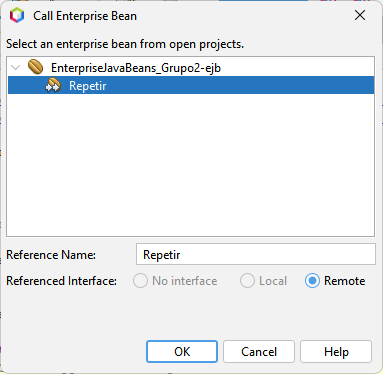


Figura 44. Selección de bean Repetir

Y se genera el código

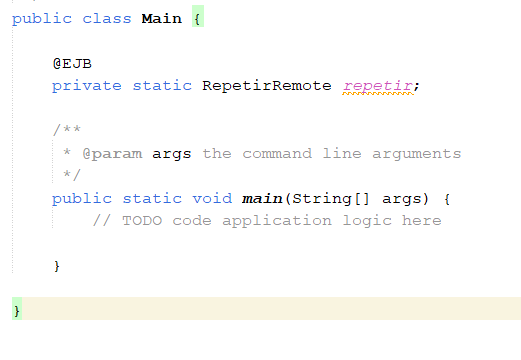


Figura 45. Código de clase main generado

Ahora se va añadir una llamada al método repetir en la clase main y quedaría de la siguiente manera:

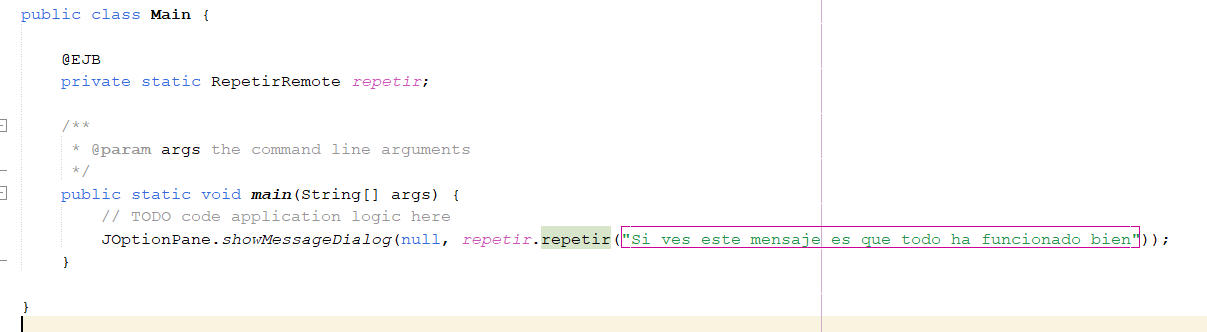


Figura 46. Código de clase Main.java

## EJECUCIÓN DEL PROYECTO

Y ya se ha terminado con la práctica, ahora ya se puede ejecutar el proyecto y el resultado sería el siguiente.

Se ejecuta la siguiente aplicación, ya que es la aplicación Beans, para ello únicamente se presiona en RUN y se observa el resultado

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 47. Proceso de ejecución de la aplicación Beans

Como se puede apreciar se muestra el mensaje dentro de la aplicación, por lo que se puede concluir que la aplicación está funcionando correctamente

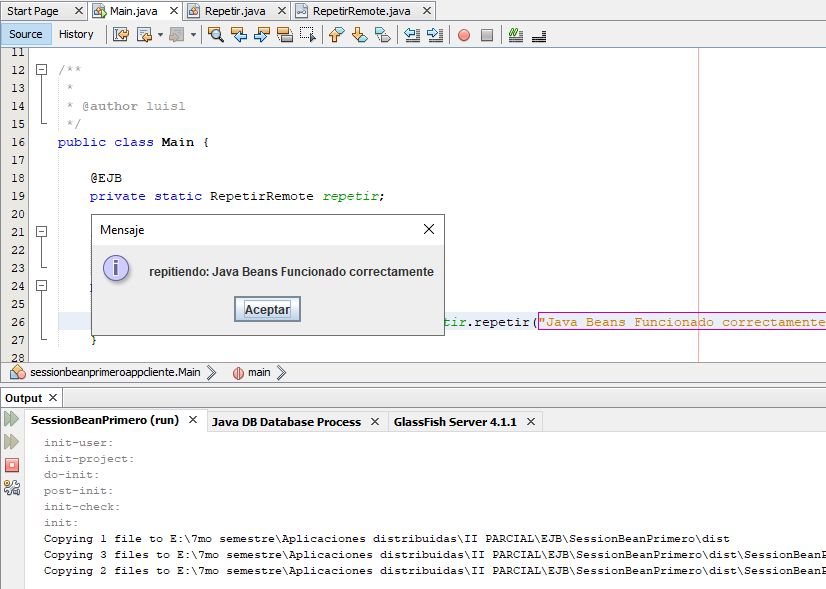


Figura 48. Ejecución del proyecto

# CONCLUSIONES

* En conclusión, fue posible conocer y aprender el funcionamiento de los Interceptor, siendo posible mediante este ejecutar ciertas acciones sobre un método siendo en el ejemplo identificar cuando se accede sobre un método y cuando sale del mismo
* Se obtuvo un aprendizaje a fondo respecto a la tecnología de Enterprise JavaBeans que es una propuesta del mundo Java para aplicaciones de empresas con distintas funcionalidades para distintas áreas de la empresa.

# RECOMENDACIONES

* Es recomendable el uso de Session Beans para ocultar y proteger la lógica empresarial, y además de garantizar una navegación fluida y continua para una buena experiencia del cliente.
* Es recomendable revisar la documentación necesaria antes de comenzar a trabajar en aplicaciones utilizando los Enterprise JavaBeans.
* Se recomienda que se sigan revisando más a profundidad los contenidos de la clase referentes a JAVA Enterprise Edition, de esta manera reforzando conceptos en los cuales hay que mejorar, ya que todos estos conceptos abordados serán de mucha utilidad para futuros proyectos.

# REFERENCIAS

[1]Oracle. "JSF 2.0 Page Decraration Language: Facelets Variant". Moved. https://docs.oracle.com/javaee/6/javaserverfaces/2.0/docs/pdldocs/facelets/index.html (accedido el 11 de enero de 2023).

[2] C. Álvarez. "Enterprise Java Beans y su funcionamiento". Arquitectura Java. https://www.arquitecturajava.com/enterprise-java-beans-y-su-funcionamiento/ (accedido el 23 de enero de 2023).

[3] ChuWiki. "Ejemplo de Session Bean con EJB 3 - ChuWiki". ChuWiki. https://chuwiki.chuidiang.org/index.php?title=Ejemplo\_de\_Session\_Bean\_con\_EJB\_3 (accedido el 23 de enero de 2023).

[4] European Knowledge Center for Information Technology. "Software modular: ¿cómo contratar sólo las funcionalidades necesarias?" TIC Portal. https://www.ticportal.es/glosario-tic/software-modular (accedido el 23 de enero de 2023).

[5] European Knowledge Center for Information Technology. "¿Qué es un servidor, cómo funciona y qué tipos hay?" TIC Portal. https://www.ticportal.es/glosario-tic/servidores (accedido el 23 de enero de 2023).

[6] J. D. M. González. "Sockets en Java: Un sistema cliente-servidor con sockets". Aprende a programar desde cero en múltiples lenguajes. <https://www.programarya.com/Cursos-Avanzados/Java/Sockets> (accedido el 7 de diciembre de 2022).

[7] "Definición de cliente servidor - Definicion.de". Definición.de. <https://definicion.de/cliente-servidor/> (accedido el 7 de diciembre de 2022).

[8] "¿Qué es una dirección de servidor?" Preguntas y respuestas interesantes - es.411answers.com. <https://es.411answers.com/a/que-es-una-direccion-de-servidor.html> (accedido el 7 de diciembre de 2022).

[9] IBM. "IBM Documentation". IBM - Deutschland | IBM. https://www.ibm.com/docs/es/rsas/7.5.0?topic=projects-web-archive-war-files (accedido el 23 de enero de 2023).

[10] JAVA. "¿Qué es Java Web Start y cómo se ejecuta?" JAVA. https://www.java.com/es/download/help/java\_webstart\_es.html (accedido el 23 de enero de 2023).

[11] "Sesión 2: La arquitectura EJB". JTech. http://www.jtech.ua.es/j2ee/2003-2004/modulos/ejb/sesion02-apuntes.htm (accedido el 23 de enero de 2023).

[12] O. Villacrés Cáceres, «Estudio de la Arquitectura de los Componentes EJB (Enterprise JavaBeans). Caso Aplicativo en el Sistema de Evaluación Docente de la ESPOCH.», feb. 2012, Disponible en: <http://dspace.espoch.edu.ec/handle/123456789/1465>