****

**ENTERPRISE JAVABEANS EJB**

**Integrantes:**

Chávez Pacheco Víctor

Portocarrero Miranda Percy

**Agosto 2016**

# RESUMEN

El desarrollo de aplicaciones y soluciones para satisfacer los requerimientos de negocio ha conllevado a una evolución no solo de la tecnología, sino también de las cualidades que estas deben tener para poder soportar el mundo cambiante, las necesidades inmediatas y futuras, así como ser resistentes a problemas típicos de desarrollo como la entropía, falta de interoperabilidad, escalabilidad, etc.

Ante este panorama es que nacieron diversas soluciones, que permitieron crecer desde la programación lineal, hacía entornos de programación orientada a objetos para representar de manera más eficaz el entorno real. Es así que se implementan soluciones cada vez más robustas, y en este caso tomando como base la orientación a componentes que se implementa en el uso de los Enterprise javabeans.

Dicha tecnología surgió para estandarizar implementación y permitir la escalabilidad de aplicaciones bajo un entorno flexible, permitiendo soluciones orientadas a buscar que el desarrollador se despegue de realizar operaciones básicas y enfocarse en implementar las operaciones y procedimientos núcleo de negocio.

En este ambiente es que es materia de estudio de forma breve una infraestructura del entorno java, en este caso nos referimos a los Enterprise Javabeans EJB, para lo cual en el presente documento se revisará brevemente sus antecedentes, y una breve colección de sus versiones Release, para luego abordar su definición, categorías y estructura; para finalizar con un ejemplo demostrativo sencillo de su uso.

# ANTECEDENTES

En Marzo de 1998 Sun Microsystems propone la especificación 1.0 de la arquitectura Enterprise JavaBeans. Esta especificación comienza con la siguiente definición:

La arquitectura Enterprise JavaBeans es una arquitectura de componentes para el desarrollo y despliegue de aplicaciones de empresa distribuidas y orientadas a objetos. Las aplicaciones escritas usando la arquitectura Enterprise JavaBeans son escalables, transaccionales y seguras para multi usuarios. Estas aplicaciones pueden escribirse una vez, y luego desplegarse en cualquier servidor que soporte la especificación Enterprise JavaBeans.

Han pasado más de diez años desde su aparición y al día de hoy los EJB’s son clases puras de Java (POJO’s) los cuales al ser desplegados en un Servidor de Aplicaciones permiten reducir la complejidad de programación, agregando robustez, reusabilidad y escalabilidad a nuestras aplicaciones empresariales de misión crítica.

Aunque se han introducido nuevas versiones de la especificación, que incorporan muchas mejoras a la propuesta inicial, la definición de la arquitectura sigue siendo la misma. La siguiente tabla muestra las distintas revisiones que ha sufrido la especificación de la arquitectura EJB.

|  |  |  |
| --- | --- | --- |
| Especificación EJB | Fecha | Principales Novedades |
| EJB 1.0 | Marzo 1998 | Propuesta inicial de la arquitectura EJB. Se introducen los beans de sesión y los de entidad (de implementación opcional). Persistencia manejada por el contendedor en los beans de entidad. Manejo de transacciones. Manejo de seguridad. |
| EJB 1.1 | Diciembre 1999 | Implementación obligatoria de los beans de entidad. Acceso al entorno de los beans mediante JNDI. |
| EJB 2.0 | Agosto 2001 | Manejo de mensajes con los beans dirigidos por mensajes. Relaciones entre beans manejadas por el contenedor. Uso de interfaces locales entre beans que se encuentran en el mismo servidor. Consultas de beans declarativas, usando el EJB QL. |
| EJB 2.1 | Agosto 2002 | Soporte para servicios web. Temporizador manejado por el contenedor de beans. Mejora en el EJB QL. |
| EJB 3.0 | Mayo 2006 | Ahora con Clúster y está incluida en JEE 5.1. Simplifica el proceso de creación de EJB y facilita la implementación de la persistencia de una nueva manera por medio de la API JPA. |

Hoy más que nunca la versión EJB 3.1 pueden ser programados una vez y ejecutados en cualquier servidor de aplicaciones Java que soporte el estándar Java EE 6. Los EJB’s ya han cumplido más de una década desde su aparición, y al día de hoy son una tecnología muy probada y que brinda beneficios tales como seguridad, transaccionalidad, multi-threading, entre muchas características más, todo esto a través del servidor de aplicaciones Java.

A diferencia de un JavaBean, que es una clase pura de Java, un Enterprise JavaBean es una clase de Java con características que lo hacen mucho más potente y robusto.

# FUNDAMENTOS TEÓRICOS

* 1. **EJB**

Enterprise JavaBeans (EJB) es una arquitectura de componentes de servidor que simplifica el proceso de construcción de aplicaciones de componentes empresariales distribuidos en Java. Por lo tanto los EJB son componentes software que se ejecutan en el lado del servidor en aplicaciones multicapa. Los Enterprise Java Beans (EJB) es código Java del lado del Servidor. Normalmente tienen la lógica de negocio de nuestra aplicación, y por lo tanto cubren el rol de la capa de servicio de nuestras aplicaciones Java.

Mediante la construcción de estos componentes es posible escribir aplicaciones escalables, fiables y seguras sin escribir código de infraestructura, y ante la existencia de esta infraestructura permite un desarrollo más rápido de la parte servidor.

Y es que debido también a que son componentes, permite el desarrollo de aplicaciones portables entre distintas plataformas y servidores de aplicaciones.

A su vez, el lado cliente de los EJB debe acceder a él a través de una interfaz que esconde los detalles de la implementación del componente solicitado, el cual implementará los procesos de negocio, los accesos a datos, etc.

Un componente se puede ver como un objeto tradicional con un conjunto de servicios adicionales soportados en tiempo de ejecución por el contenedor de componentes. El contenedor de componentes se denomina contenedor EJB y es algo así como el sistema operativo en el que éstos residen. En Java existe un modelo de programación de objetos remotos denominado RMI. Con RMI es posible enviar peticiones a objetos que están ejecutándose en otra máquina virtual Java. Podemos ver un componente EJB como un objeto remoto RMI que reside en un contenedor EJB que le proporciona un conjunto de servicios adicionales.

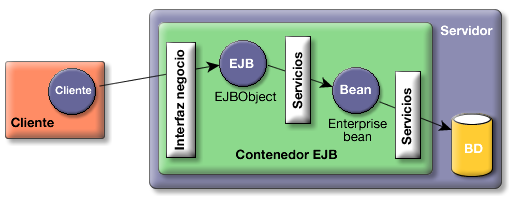
A diferencia de la programación orientada a objetos en la cual puedes reutilizar clases, es que con la infraestructura EJB y el desarrollo de componentes hace posible reutilizar un mayor nivel de funcionalidades e incluso es posible modificar estas funcionalidades y adaptarlas a cada entorno de trabajo particular sin tocar el código del componente desarrollado.

Ahora mientras se trabaja con componentes se debe dedicar tanta atención al despliegue (deployment) del componente como a su desarrollo. Entendemos por despliegue la incorporación del componente a nuestro contenedor EJB y a nuestro entorno de trabajo (bases de datos, arquitectura de la aplicación, etc.). El despliegue se define de forma declarativa, mediante un fichero XML (descriptor del despliegue, deployment descriptor) en el que se definen todas las características del bean.

* 1. **Contenedor EJB**

Como se indicó previamente, es el contenedor de componentes o Enterprise javabeans, todos los componentes desarrollados, el cual funciona a modo de un sistema operativo en el cual se coordina, comunican, invocan los beans servidores con las aplicaciones clientes.

El contenedor EJB es un programa Java que corre en el servidor y que contiene todas las clases y objetos necesarios para el correcto funcionamiento de los enterprise beans.



*Figura 1: Representación de alto nivel del funcionamiento de los enterprise beans.*

En la figura se puede ver una representación de muy alto nivel del funcionamiento básico de los enterprise beans. En primer lugar, el cliente realiza peticiones al bean y el servidor que contiene el bean está ejecutándose en máquinas virtuales Java distintas. Incluso pueden estar en distintos hosts. Otra cosa a resaltar es que el cliente nunca se comunica directamente con el enterprise bean, sino que el contenedor EJB proporciona un EJBObject que hace de interfaz. Cualquier petición del cliente (una llamada a un método de negocio del enterprise bean) se debe hacer a través del objeto EJB, el cual solicita al contenedor EJB una serie de servicios y se comunica con el enterprise bean. Por último, el bean realiza las peticiones correspondientes a la base de datos.

El contenedor EJB se preocupa de cuestiones como:

* El cliente tiene permiso para llamar al método
* Abrir la transacción al comienzo de la llamada y cerrarla al terminar.
* Refrescar el bean con los datos de la base de datos.
  1. **Estructura de un EJB**

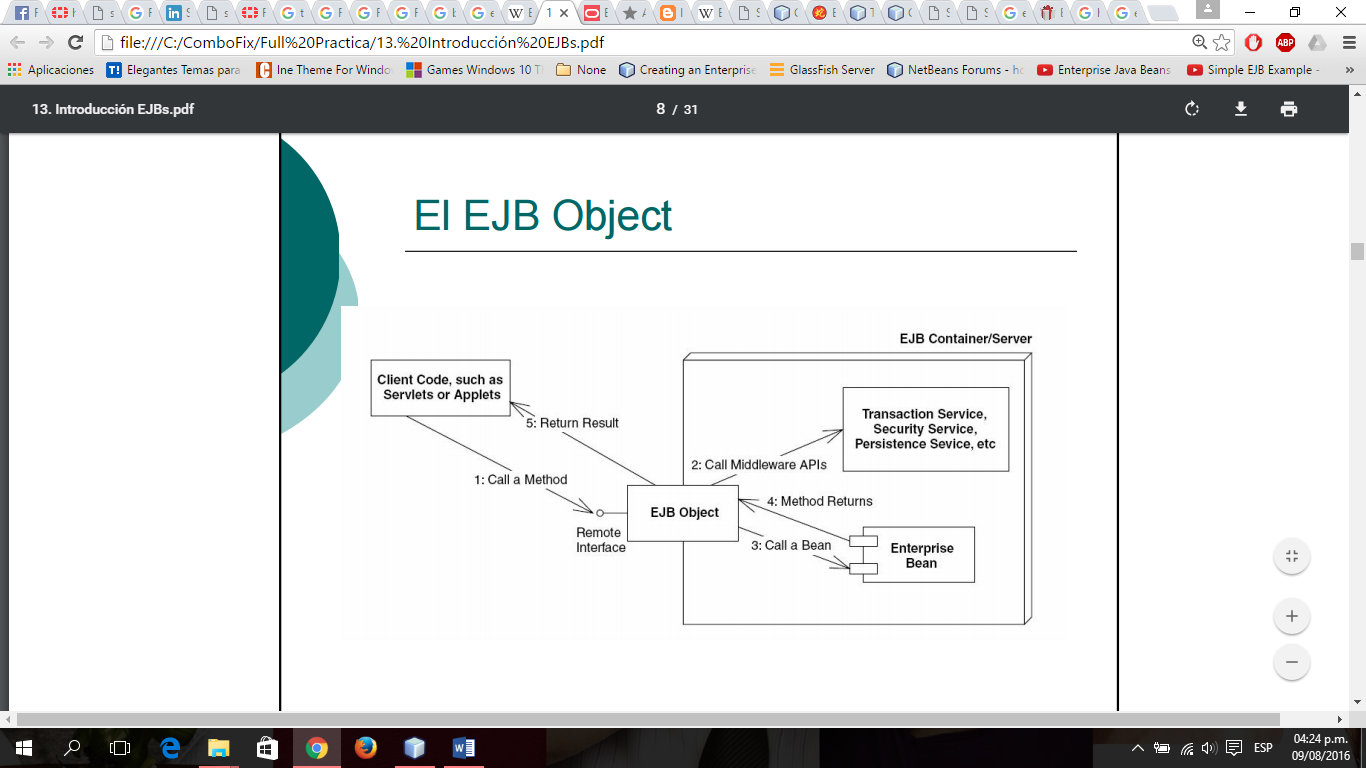
Un EJB no es una sola clase, sino un conjunto de clases, interfaces y recursos:

* + 1. **El bean de empresa o bean de negocio**

Se trata de implementación en sí del objeto distribuido, dependiendo del tipo de EJB podemos especificar el tipo de bean:

* Implementación de reglas de negocio en el caso de EJBs de sesión.
* Lógica de persistencia en el caso de los Entity EJBs.
* Lógica de negocio, bean orientado a mensajes.
  + 1. **El EJB Object**

Representa la capa de indirección que intercepta las invocaciones remotas al EJB y gestiona los servicios implícitos del contenedor de EJBs, funciona como un objeto inteligente que implementa la lógica intermedia que el contenedor de EJBs requiere antes de que una invocación a método sea servida por el bean de negocio.



*Figura 2: Funcionamiento del EJB Object*

* + 1. **Las interfaces Remotas**

Son los contratos entre el cliente del EJB y el bean de negocio. Están sujetas a la especificación y deben cumplir ciertas reglas. En la interfaz remota se publican todos los métodos del bean de negocio que se quieran invocar remotamente.

La interfaz remota es implementada por el EJB Object, el cual es generado automáticamente por el contenedor de EJBs.

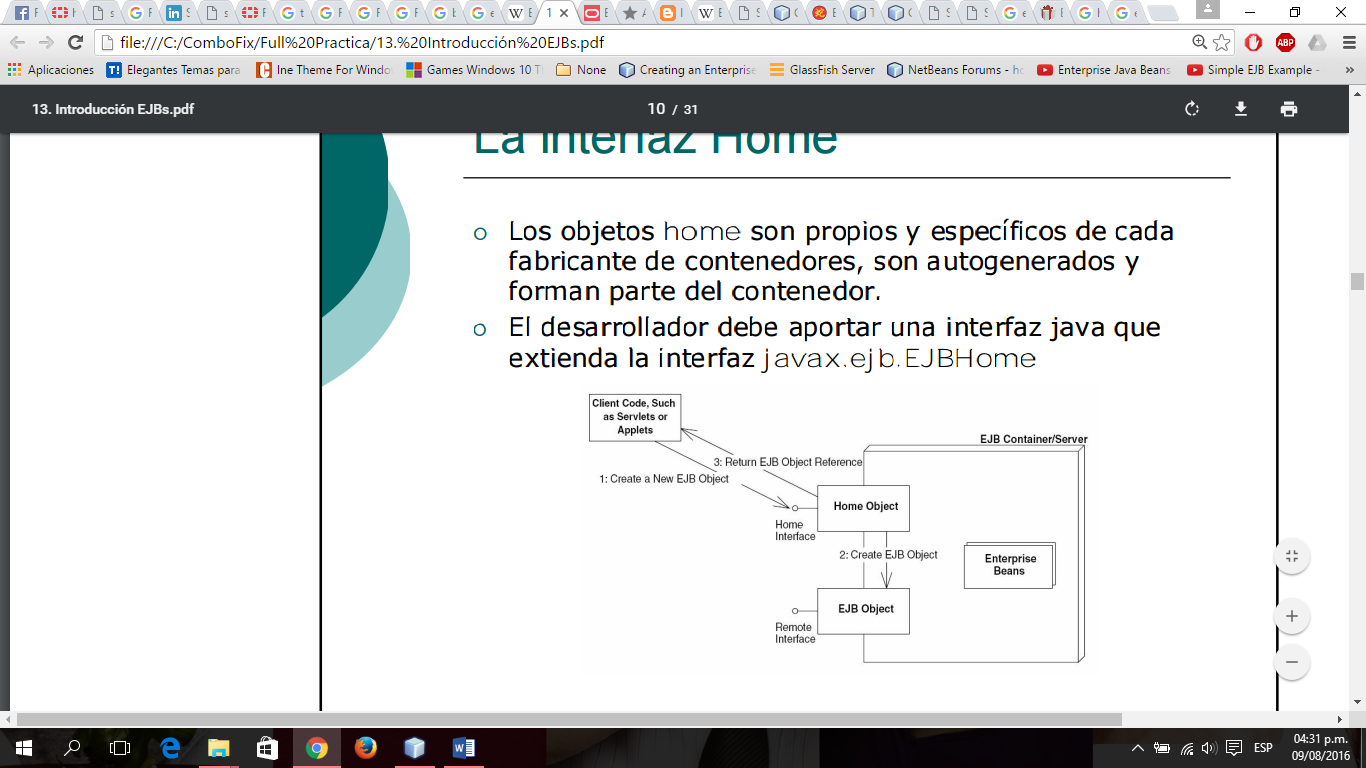
* + 1. **El Objeto Home**

El cliente no puede instanciar directamente un objeto remoto, dado que se encuentran en otra máquina distinta.

Debido al principio de transparencia, la ubicación del cliente es independiente de la localización del objeto remoto.

Las instancias de EJBs se obtienen por medio de factorías de objetos (Patrón de diseño Factory. Esta factoría es la responsable de instanciar, buscar y destruir los objetos. Él objeto home es la factoría para la obtención de referencias a EJBs.

Los objetos home son propios y específicos de cada fabricante de contenedores, son autogenerados y forman parte del contenedor.

*Figura 3: Object Home en el Contenedor EJB*

* + 1. **Las Interfaces Locales**

Solventan el problema de la sobrecarga cuando el EJB se ejecuta en la propia máquina del cliente. Además permiten invocar al EJB como si se tratara de un objeto local, tratando con un Objeto Local.

El Objeto Local realiza las tareas de middleware que le corresponderían al EJB Object, y luego le cede el control al bean de negocio, de esta forma, se evitan las tareas propias a la invocación remota (strubs, serialización, etc.).

* + 1. **Descriptores de Despliegue**

Especifica las propiedades y servicios del EJB de forma declarativa, describiendo como ha de ser desplegado el EJB en el contenedor, y como ha de ser manejado:

* Ciclo de vida
* Sistema de persistencia
* Control de transacciones
* Servicios de seguridad.
  + 1. **Recursos Específicos del Servidor de Aplicaciones**

Cada servidor describe y exige, por ejemplo, su propio descriptor de despliegue (además del ejbjar.xml), normalmente, los servidores de aplicaciones facilitan herramientas para realizar el despliegue y confeccionar los descriptores.

* 1. **Categorías de EJBs**

Se definen tres tipos de beans:

* Beans de Entidad (Entity Beans)
* Beans Dirigidos por Mensajes (Message Driven Beans)
* Beans de Sesión (Session Beans)
  + 1. **Beans de Entidad (Entity Beans)**

Los beans de entidad representan un objeto concreto que tiene existencia en alguna base de datos de la empresa. Una instancia de un bean de entidad representa una fila en una tabla de la base de datos. Su objetivo es encapsular los objetos del lado del servidor que almacena los datos. Los EJB de entidad presentan la característica fundamental de la persistencia y por ello presenta dos tipos:

* + - 1. Persistencia Manejada por el Bean (BMP)

El propio objeto entidad se encarga, mediante una base de datos u otro mecanismo, de almacenar y recuperar los datos a los que se refiere, por lo cual la responsabilidad de implementar los mecanismos de persistencia es del programador así como las tareas de persistir, buscar, y recuperar entidades.

* + - 1. Persistencia Manejada por el Contenedor(CMP)

El contenedor se encarga de almacenar y recuperar los datos del objeto de entidad mediante el mapeo o vinculación de las columnas de una tabla de la base de datos con los atributos del objeto, de forma que el desarrollador no se preocupa de las sentencias SQL de inserción, recuperación, etc.

* + 1. **Beans Dirigidor por Mensajes (Message Driven Beans)**

Los beans dirigidos por mensajes pueden escuchar mensajes de un servicio de mensajes JMS. Los clientes de estos beans nunca los llaman directamente, sino que es necesario enviar un mensaje JMS para comunicarse con ellos. Los beans dirigidos por mensajes no necesitan objetos EJBObject porque los clientes no se comunican nunca con ellos directamente. Un ejemplo de bean dirigido por mensajes podría ser un bean ListenerNuevoCliente que se activara cada vez que se envía un mensaje comunicando que se ha dado de alta a un nuevo cliente.

* + 1. **Beans de Sesión (Session Beans)**

Un bean de sesión representa un proceso o una acción de negocio, gestionan el flujo de la información en el servidor. Normalmente, cualquier llamada a un servicio del servidor debería comenzar con una llamada a un bean de sesión. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor.

A diferencia de los bean de entidad, los beans de sesión no se comparten entre más de un cliente, sino que existe una correspondencia uno-uno entre beans de sesión y clientes. Por esto, el contenedor EJB no necesita implementar mecanismos de manejo de concurrencia en el acceso a estos beans.

En la definición de EJB 3.1. se consideran tres tipos de beans de sesión:

* + - 1. Beans de Sesión Sin Estado (Stateless Session Beans)

Los beans de sesión sin estado no se modifican con las llamadas de los clientes. Los métodos que ponen a disposición de las aplicaciones clientes son llamadas que reciben datos y devuelven resultados, pero que no modifican internamente el estado del bean. Esta propiedad permite que el contenedor EJB pueda crear una reserva (pool) de instancias, todas ellas del mismo bean de sesión sin estado y asignar cualquier instancia a cualquier cliente. Incluso un único bean puede estar asignado a múltiples clientes, ya que la asignación sólo dura el tiempo de invocación del método solicitado por el cliente.

Una de las ventajas del uso de beans de sesión, frente al uso de clases Java u objetos RMI es que no es necesario escribir los métodos de los beans de sesión de una forma segura para threads (thread-safe), ya que el contenedor EJB se va a encargar de que nunca haya más de un thread accediendo al objeto. Para ello usa múltiples instancias del bean para responder a peticiones de los clientes.

Cuando un cliente invoca un método de un bean de sesión sin estado, el contenedor EJB obtiene una instancia de la reserva. Cualquier instancia servirá, ya que el bean no puede guardar ninguna información referida al cliente. Tan pronto como el método termina su ejecución, la instancia del bean está disponible para otros clientes. Esta propiedad hace que los beans de sesión sin estado sean muy escalables para un gran número de clientes. El contenedor EJB no tiene que mover sesiones de la memoria a un almacenamiento secundario para liberar recursos, simplemente puede obtener recursos y memoria destruyendo las instancias.

Los beans de sesión sin estado se usan en general para encapsular procesos de negocio, más que datos de negocio (tarea de los entity beans). Estos beans suelen recibir nombres como ServicioBroker o GestorContratos para dejar claro que proporcionan un conjunto de procesos relacionados con un dominio específico del negocio.

Es apropiado usar beans de sesión sin estado cuando una tarea no está ligada a un cliente específico. Por ejemplo, se podría usar un bean sin estado para enviar un e-mail que confirme un pedido on-line o calcular unas cuotas de un préstamo.

También puede usarse un bean de sesión sin estado como un puente de acceso a una base de datos o a un bean de entidad. En una arquitectura cliente-servidor, el bean de sesión podría proporcionar al interfaz de usuario del cliente los datos necesarios, así como modificar objetos de negocio (base de datos o bean de entidad) a petición de la interfaz. Este uso de los beans de sesión sin estado es muy frecuente y constituye el denominado patrón de diseño session facade.

Un ejemplo de bean de sesión sin estado podría ser un componente que comprueba si un símbolo de compañía está disponible en el mercado de valores y devuelve la última cotización registrada.

* + - 1. Beans de Sesión Con Estado (Stateful Session Beans)

En un bean de sesión con estado, las variables de instancia del bean almacenan datos específicos obtenidos durante la conexión con el cliente. Cada bean de sesión con estado, por tanto, almacena el estado conversacional de un cliente que interactúa con el bean. Este estado conversacional se modifica conforme el cliente va realizando llamadas a los métodos de negocio del bean. El estado conversacional no se guarda cuando el cliente termina la sesión.

La interacción del cliente con el bean se divide en un conjunto de pasos. En cada paso se añade nueva información al estado del bean. Cada paso de interacción suele denominarse con nombres como setNombre o setDireccion, siendo nombre y direccion dos variables de instancia del bean.

Un ejemplo típico es un carrito de la compra, en donde el cliente va guardando uno a uno los ítem que va comprando.

El estado del bean persiste mientras que existe el bean. A diferencia de los beans de entidad, no existe ningún recurso exterior al contenedor EJB en el que se almacene este estado.

Debido a que el bean guarda el estado conversacional con un cliente determinado, no le es posible al contenedor crear un almacén de beans y compartirlos entre muchos clientes. Por ello, el manejo de beans de sesión con estado es más pesado que el de beans de sesión sin estado.

En general, se debería usar un bean de sesión con estado si se cumplen las siguientes circunstancias:

* El estado del bean representa la interacción entre el bean y un cliente específico.
* El bean necesita mantener información del cliente a lo largo de un conjunto de invocaciones de métodos.
* El bean hace de intermediario entre el cliente y otros componentes de la aplicación, presentando una vista simplificada al cliente.
  + - 1. Singleton Session Beans

Son objetos de negocios compartiendo un estado global dentro una JVM, El acceso concurrente hacía una y única instancia del vean puede ser controlada por el contenedor (CMC) o por el mismo vean (BMC), CMC puede ser ajustado usando la notación @Lock\_annotation que designa una lectura o escritura fija que será usada por los llamados de los métodos. Un ejemplo de este tipo de Bean sería una lista de precios global que será igual para todos los usuarios, evitando que la aplicación tenga que ejecutar la misma consulta de forma repetitiva.

* 1. **Desarrollo de Beans**

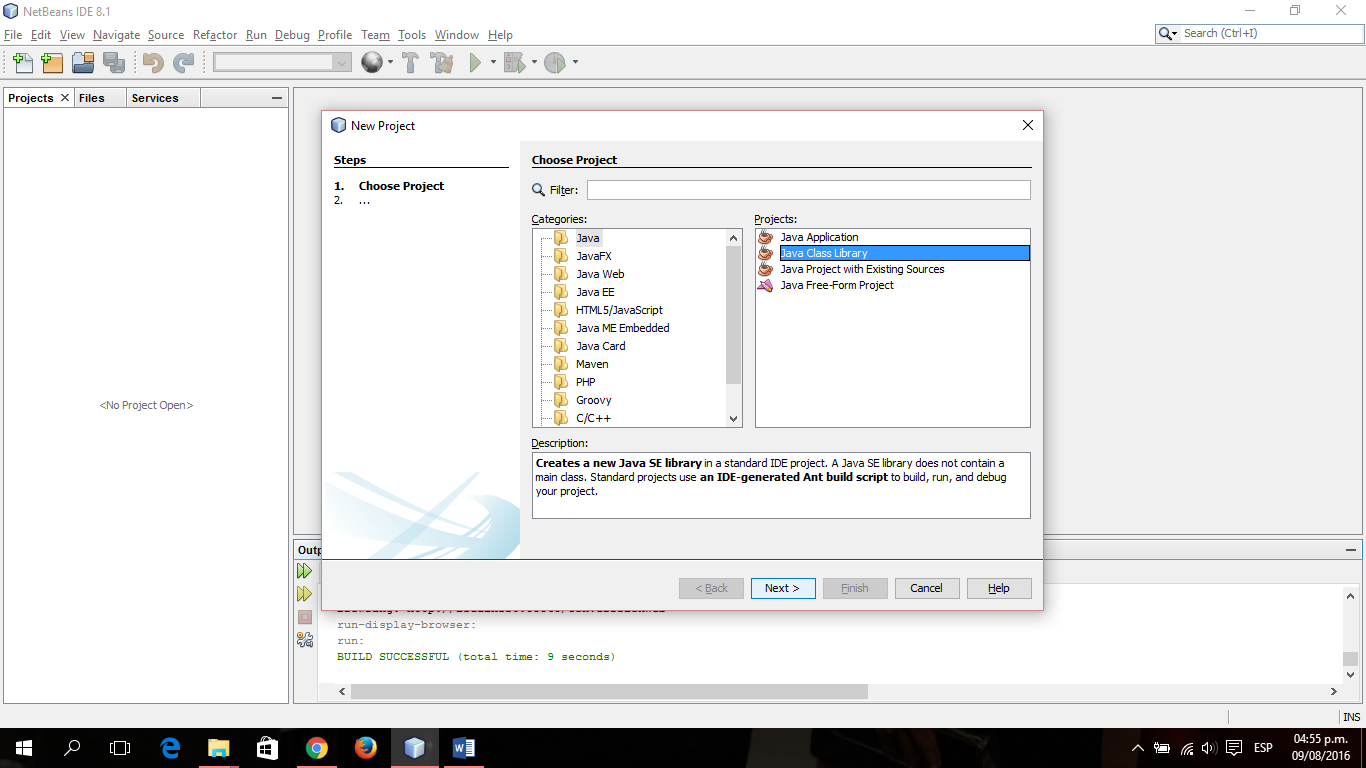
El desarrollo y programación de los beans suele ser un proceso bastante similar sea cual sea el tipo de bean. Consta de los siguientes 5 pasos:

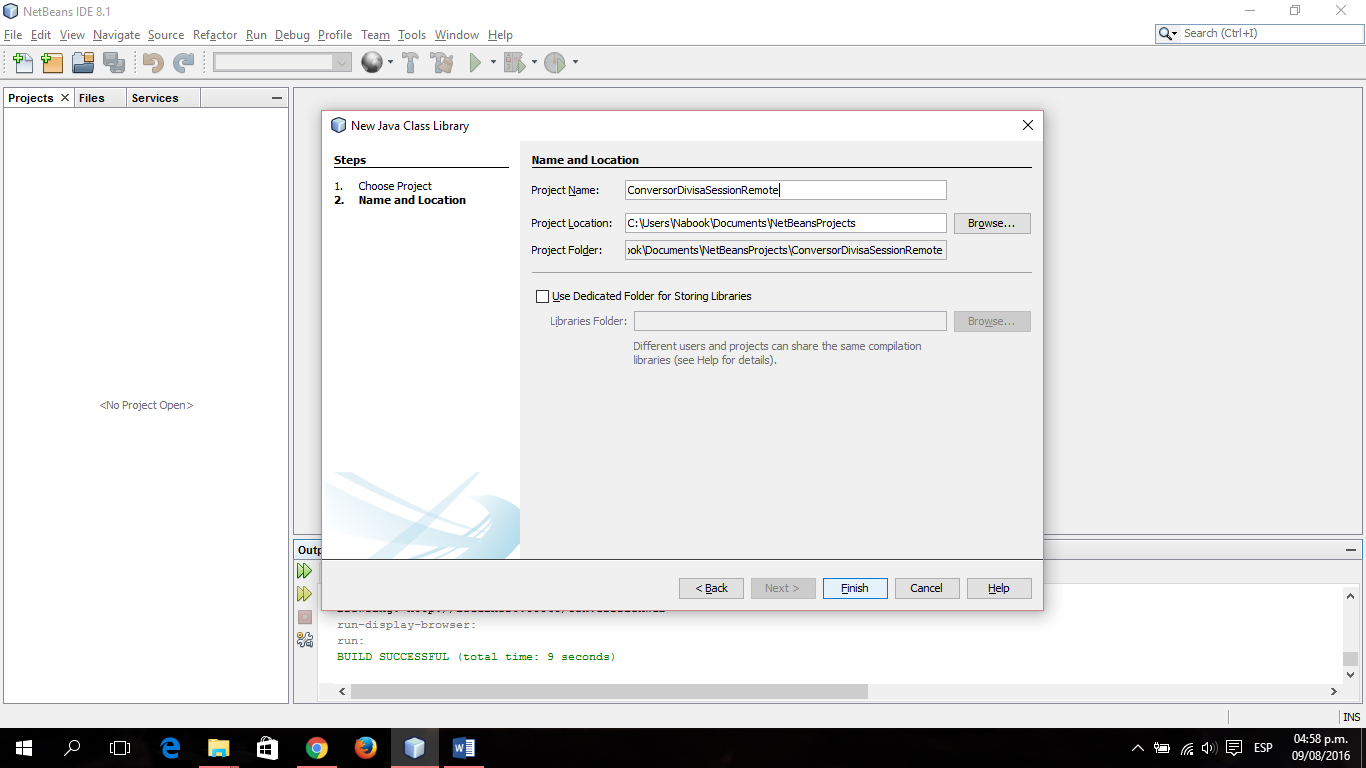
1. Escribe y compila la clase bean que contiene a todos los métodos de negocio.
2. Escribe y compila las dos interfaces del bean: home y componente.
3. Crea un descriptor XML del despliegue en el que se describa qué es el bean y cómo debe manejarse. Este fichero debe llamarse ejb-jar.xml.
4. Pon la clase bean, los interfaces y el descriptor XML del despliegue en un fichero EJB JAR . Podría haber más de un bean el mismo fichero EJB JAR, pero nunca habrá más de un descriptor de despliegue.
5. Despliega el bean en el servidor usando las herramientas proporcionadas por el servidor de aplicaciones.

# CASO DEMOSTRATIVO

En este caso se detalla la implementación de un Bean de Sesión que ejecute una operación de multiplicación en el Servidor, con un cliente Web en página JSP que se soporte en Servlets Cliente para enmascarar las solicitudes al contenedor EJB mediante el uso de la interfaz remota y local. Este pequeño ejemplo tendrá como parámetros el monto a convertir según el tipo de cambio ingresado, por lo cual lo consideraremos como un conversor de divisas. Esta implementación se realizara en el IDE de java Netbeans 8,1, utilizando el Glassfish como servidor web de aplicaciones.

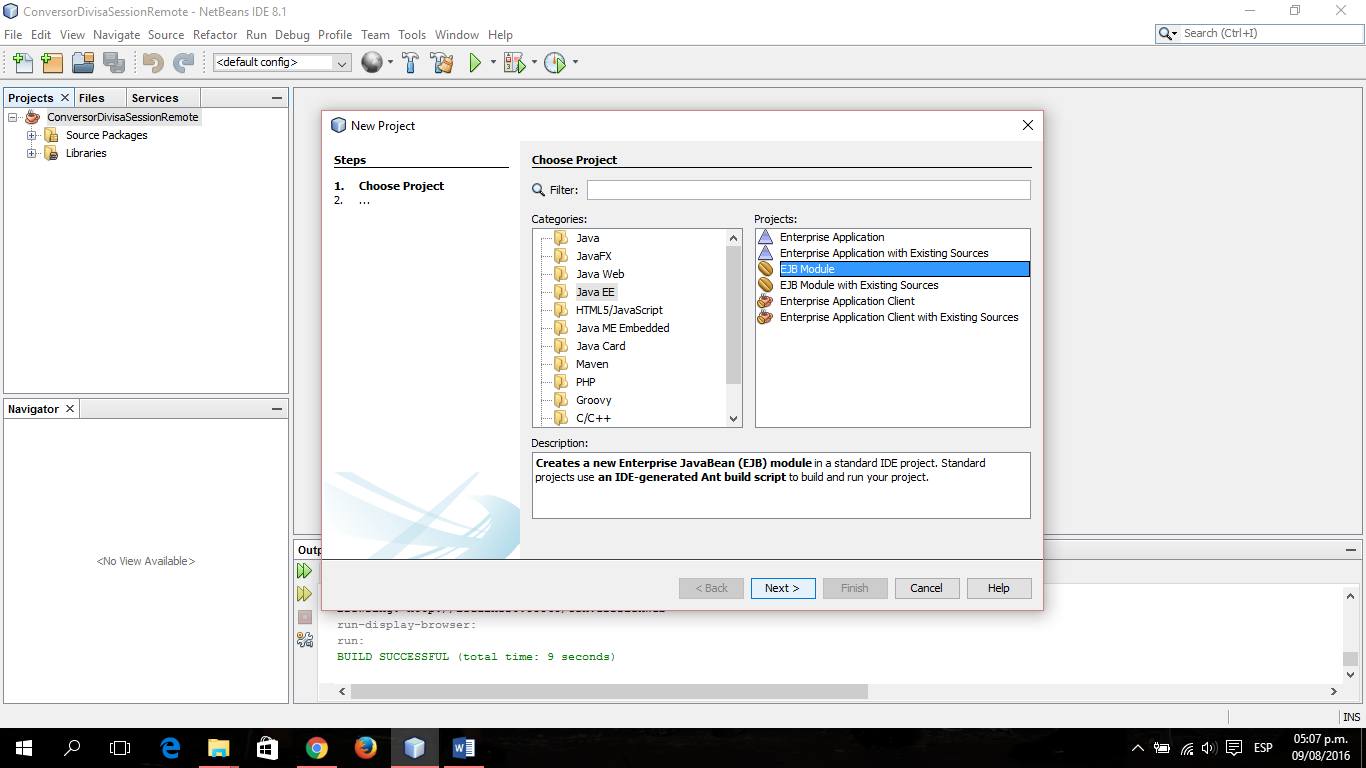
1. Primero definimos una librería de clases java para contener la interfaz remota asociada con el Bean de Sesión a crear. Lo llamaremos *ConversorDivisaSessionRemote*.

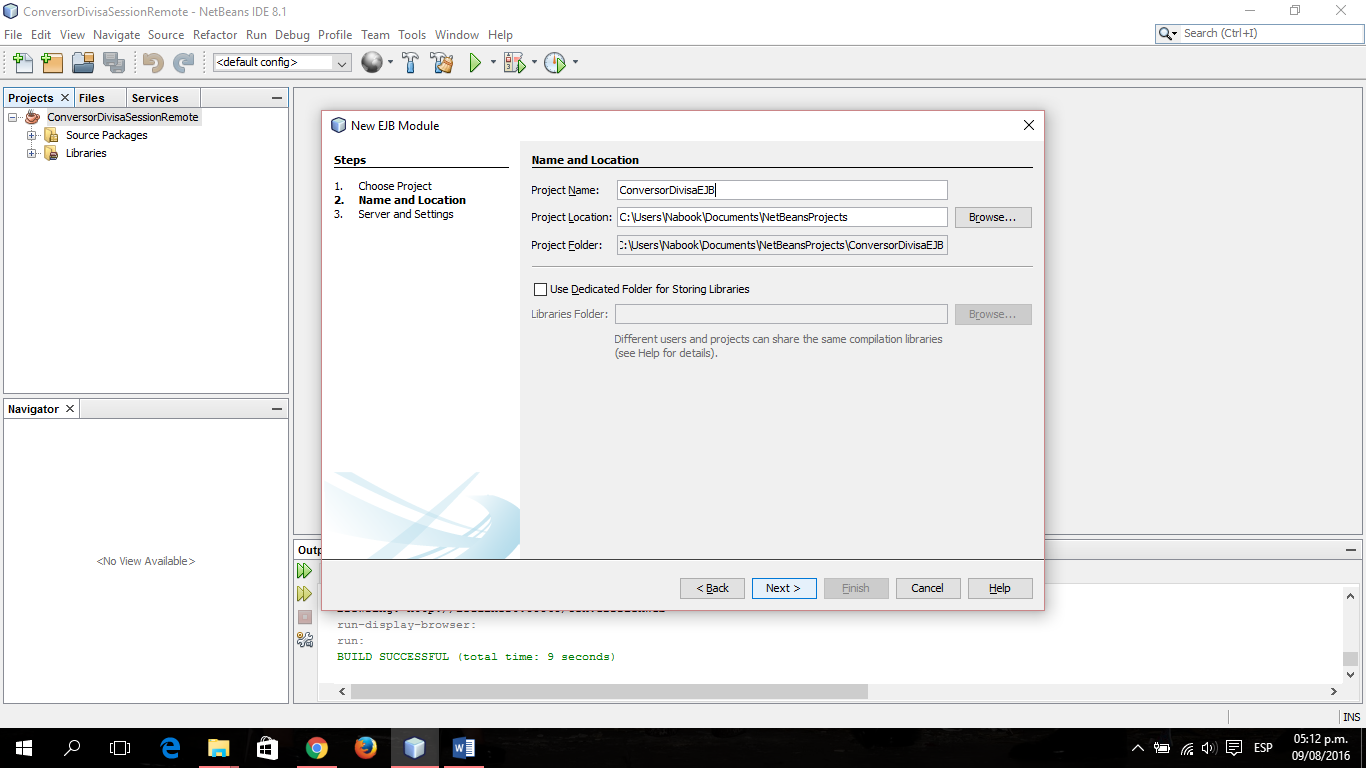


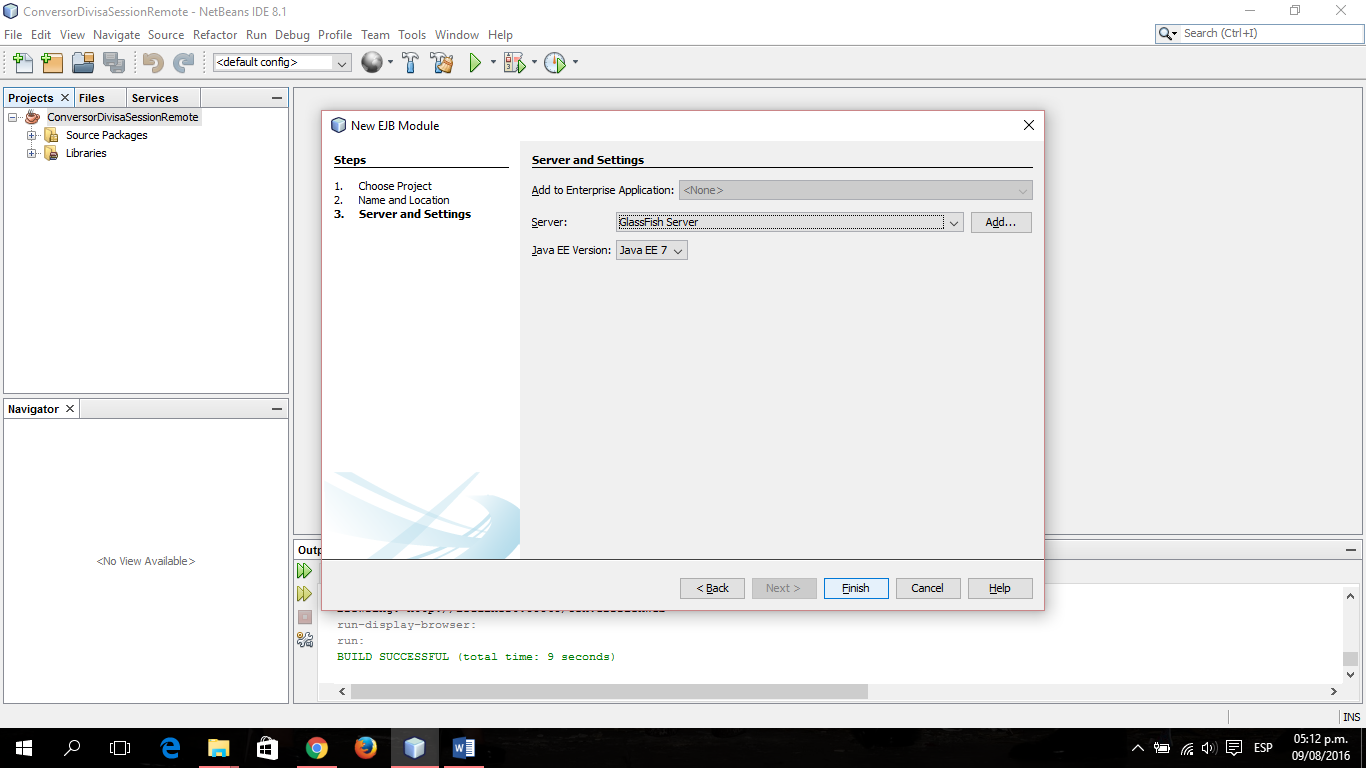


*Figura 4. Creación de Java Class Library para controlar la Interfaz Remota*

1. Una vez creada nuestra librería que albergará la interfaz remota, procedemos a implementar el módulo EJB, en el cual se implementarán los beans de negocio, para ello creamos un nuevo proyecto / java EE / EJB Module al cual llamaremos *ConversorDivisaEJB*. Se podría haber seleccionado una aplicación empresarial (Enterprise Application) el cual habría autogenerado los módulos EJB y Cliente Web automáticamente, pero se optó por la segregación para este ejemplo demostrativo.

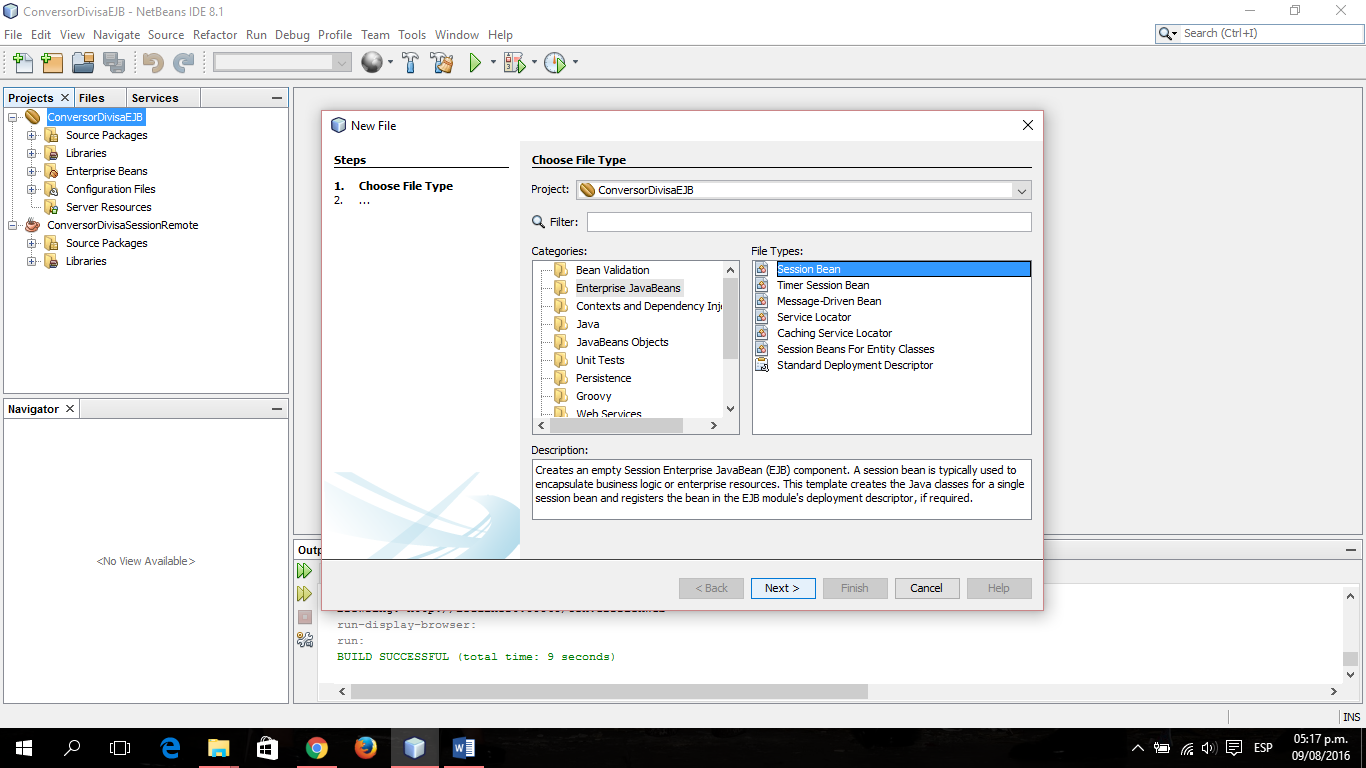


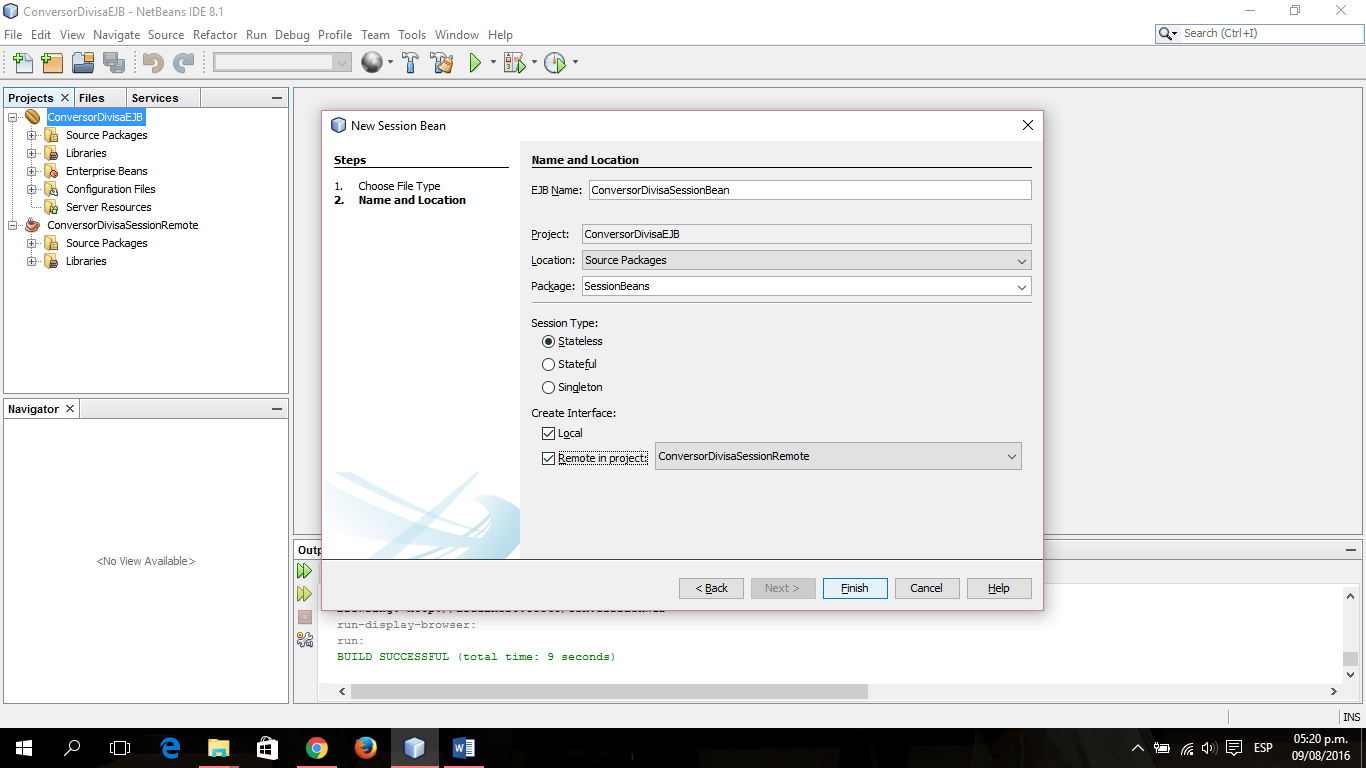




*Figura 5: Creando el Modulo EJB - ConversorDivisaEJB*

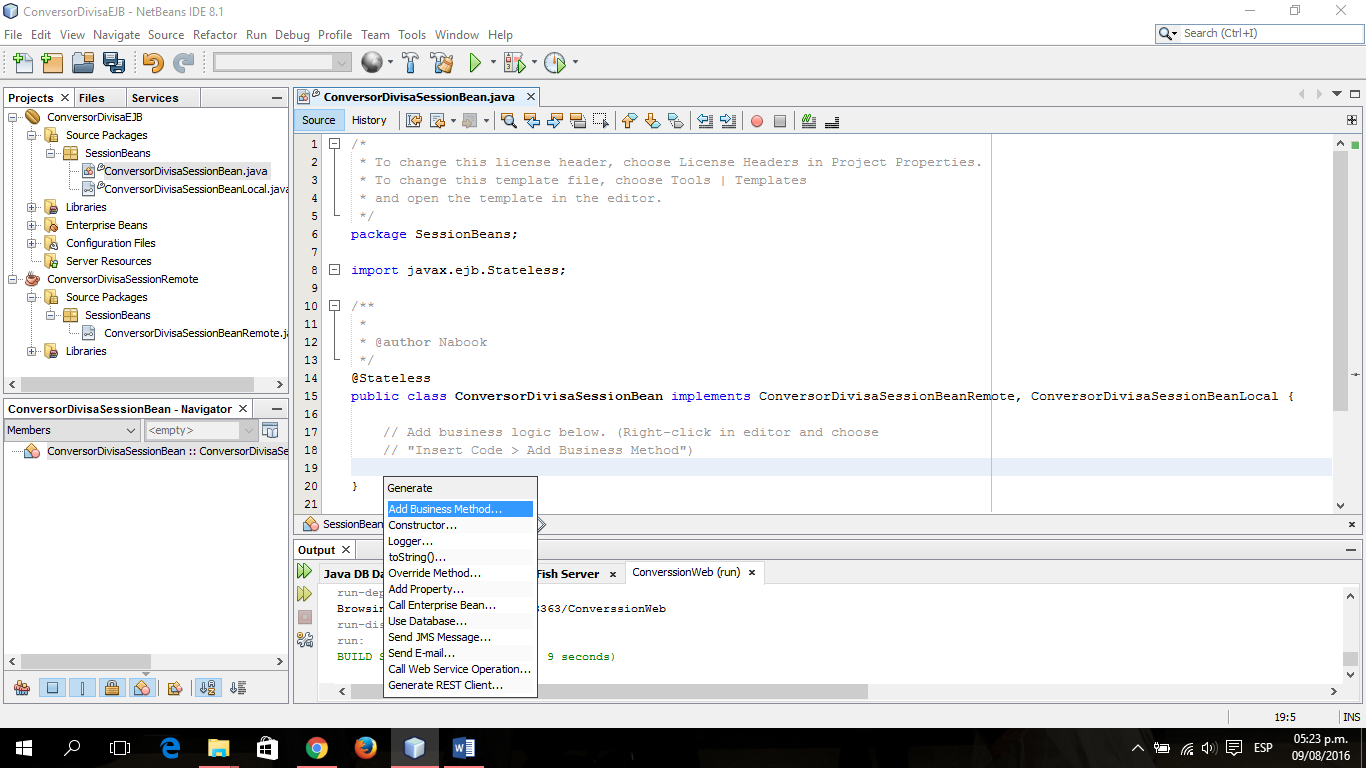
1. A continuación debemos ahora si crear el Bean de Sesión, para lo cual en el sub menú del proyecto ConversorDivisaEJB se agrega nuevo Session Bean, al cual llamaremos *ConversorDivisaSessionBean*, indicaremos que estará contenido en nuestro paquete o package SessionBeans, será un bean sin estado (stateless) porque es similar a una validación u operación que solo implica una llamada y no mantener datos constantes, ,implementaremos su interfaz local y remota (en la remota seleccionamos la librería de clases que creamos al inicio para su interfaz remota).

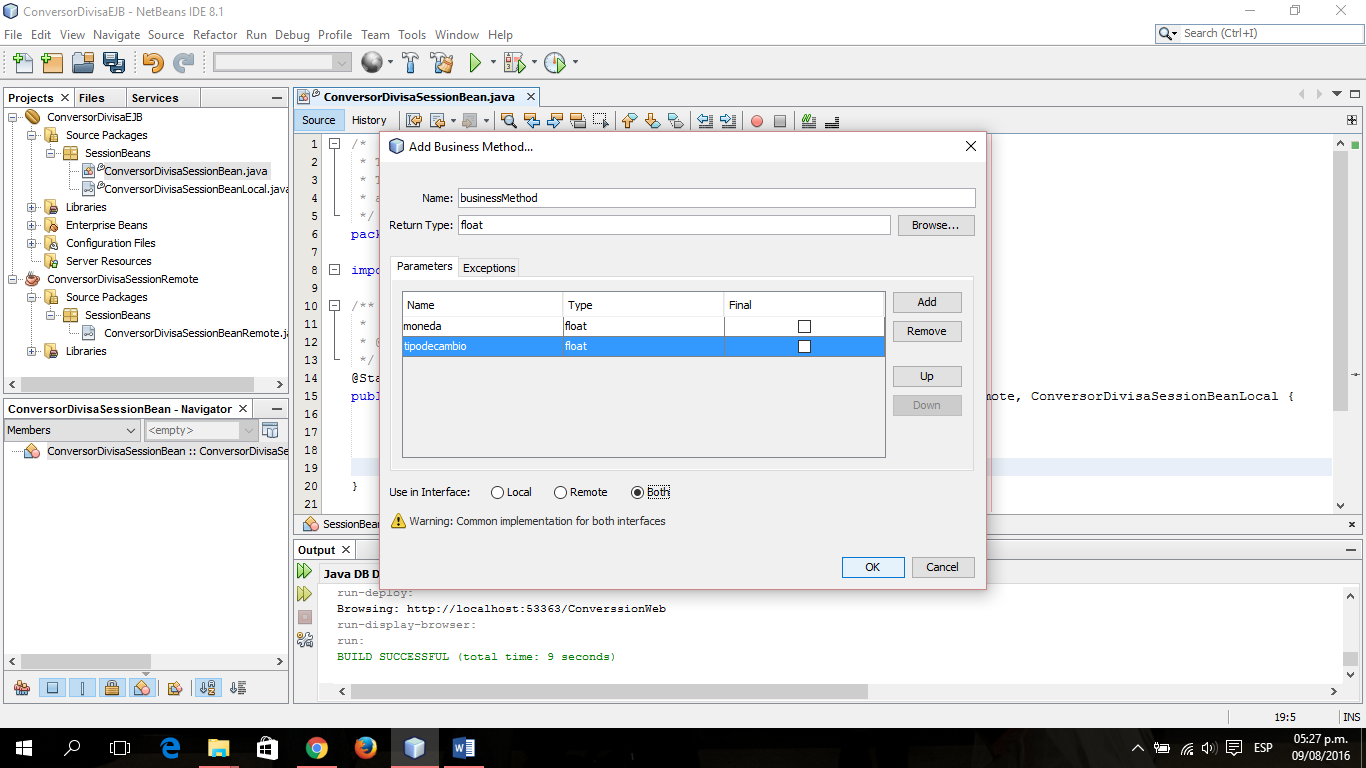




*Figura 6: Creación del Bean de Sesión - ConversorDivisaSessionBean*

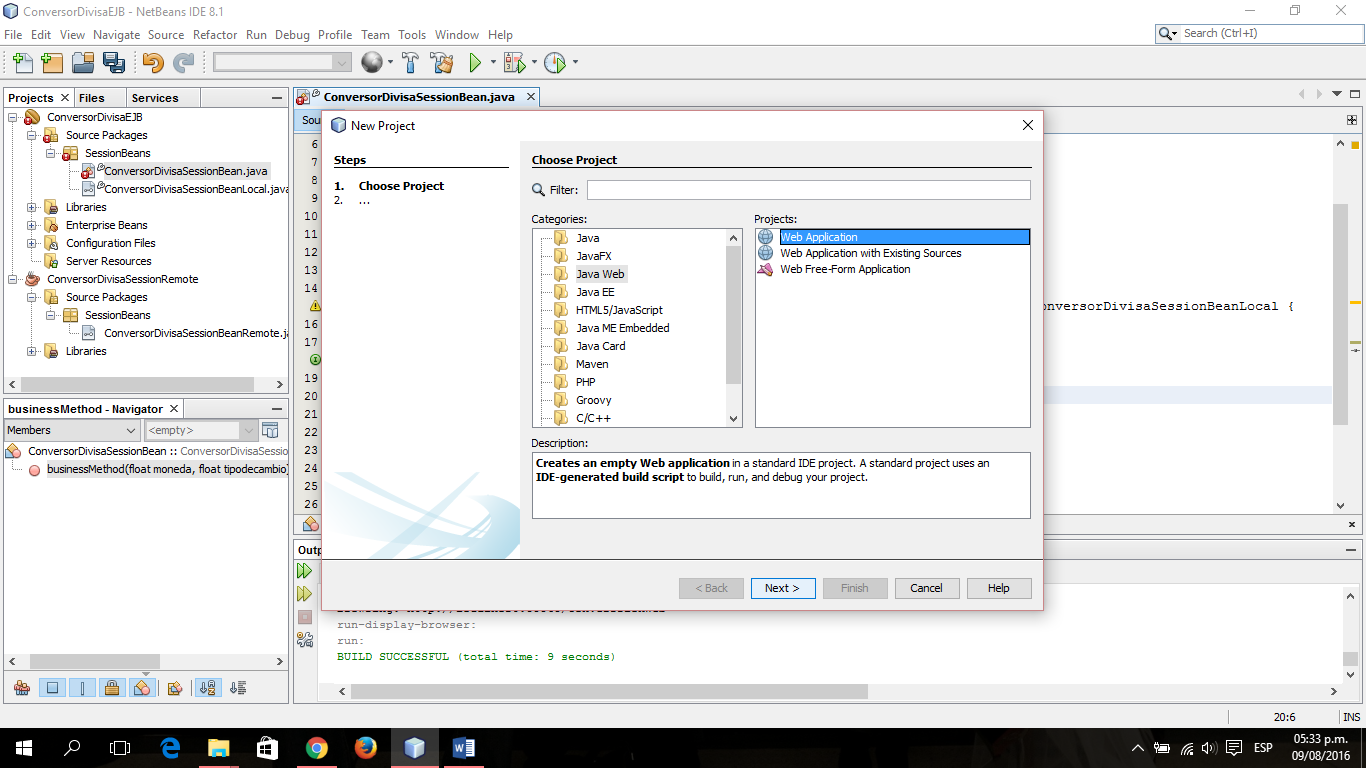
1. Ahora debemos implementar los métodos de negocio, para ello simplemente en nuestra clase ConversorDivisaSessionBean, que implementa la interfaz remota y local, debemos agregar dicho método con un clic derecho->insert code->add bussiness method. Agregaremos un método simple “Convertir” que recibe como parámetros dos números float (moneda y tipodecambio) y retorna la multiplicación de ambos como resultado convertir la moneda origen por el tipo de cambio.

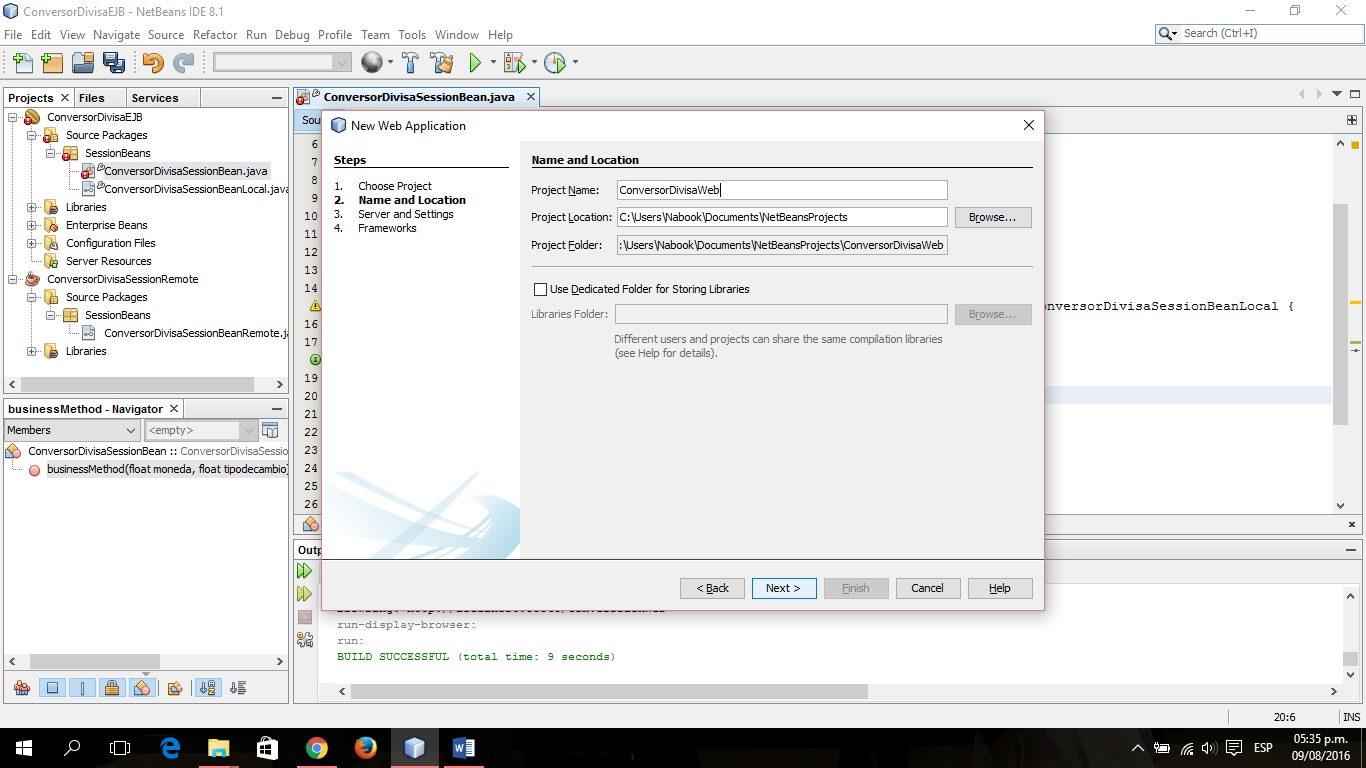


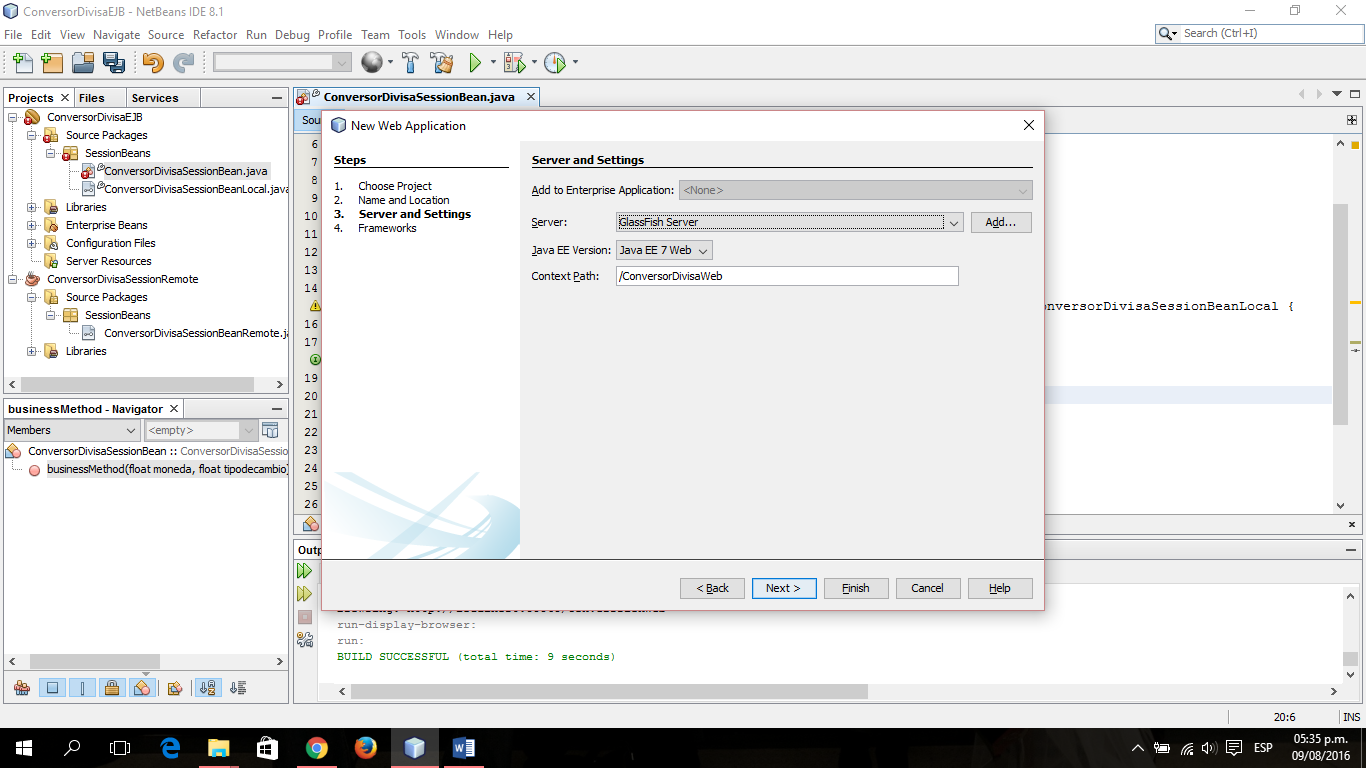


*Figura 7: Creación del Método Convertir en el Bean como Método de Negocio*

1. Ya se completó el Servidor, compuesto por el módulo EJB que implementa dos interfaces para atender los requerimientos clientes y tiene un bean que ejecuta proceso de negocio.
2. Ahora implementaremos el cliente para dar prueba al Servidor y más importante, al Bean de Sesión. Para ello creamos un nuevo proyecto, una aplicación web, que llamaremos *ConversorDivisaWeb*, .



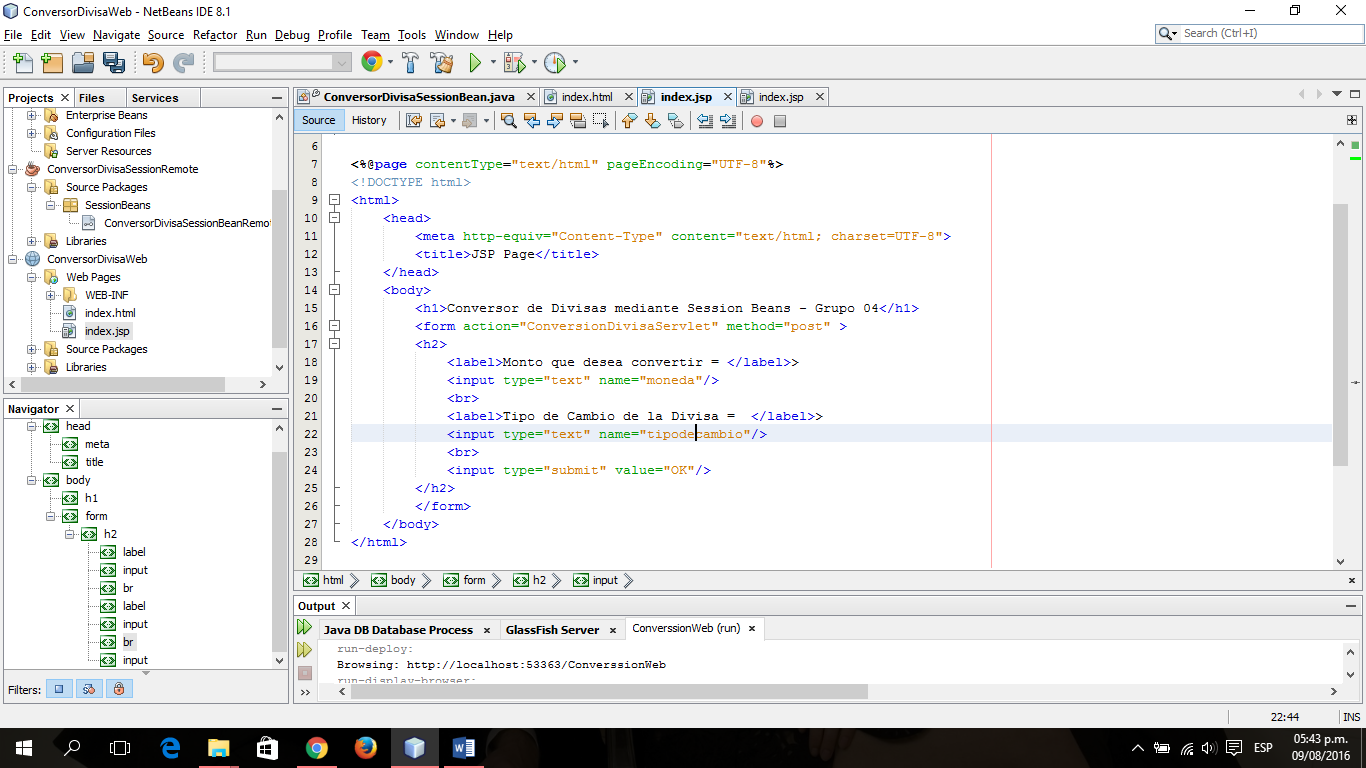




*Figura 8: Creación de la Aplicación Cliente en Web - ConversorDivisaWeb*

1. En la aplicación cliente debemos generar dos cosas: la página JSP que contiene el formulario que enviará los datos a procesar, y el Servlet que usaremos para invocar las operaciones que atenderá el módulo EJB en el Servidor y luego procesar los resultados en una vista al cliente.

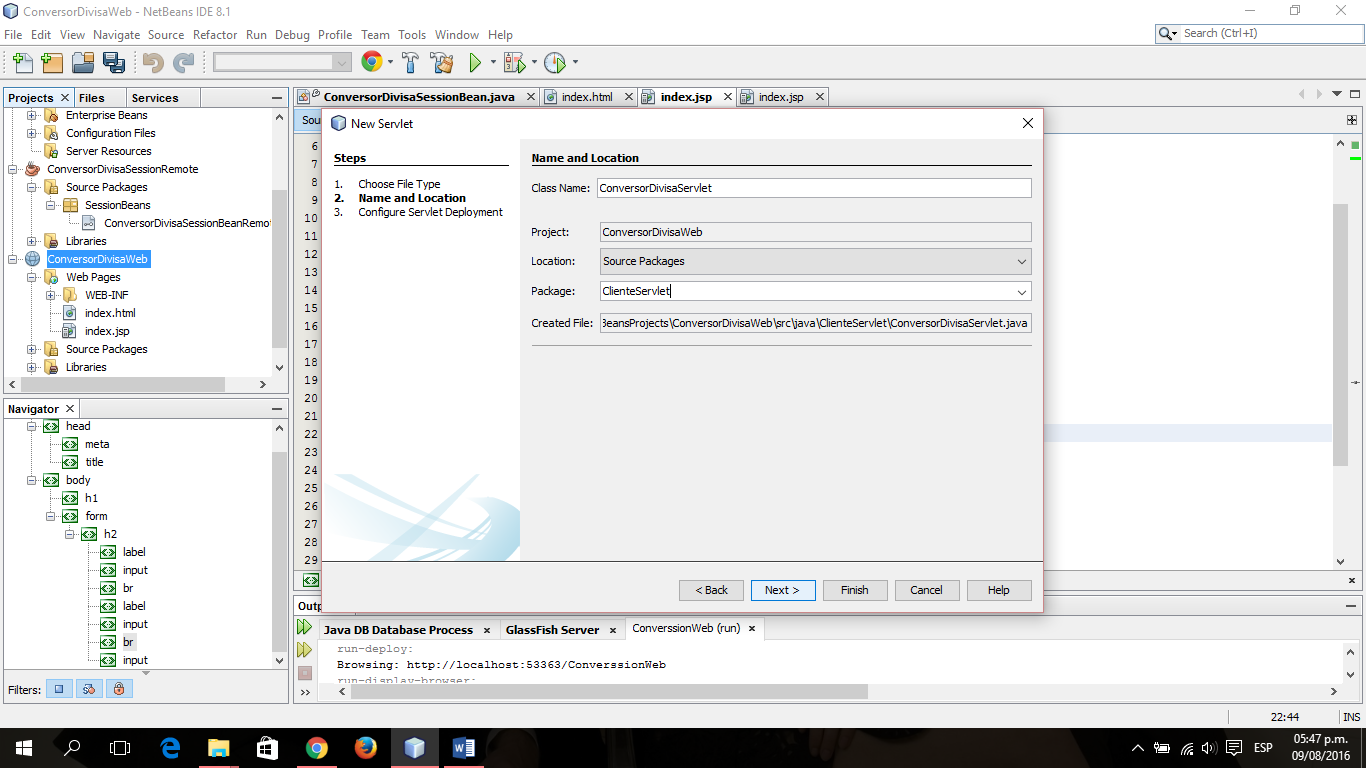
Para el primero solo ponemos un nuevo archivo->JSP, lo llamaremos index.jsp. Añadiremos el formulario con los campos “monto a convertir” y “tipo de divisa” quedando el código de la página JSP asi:

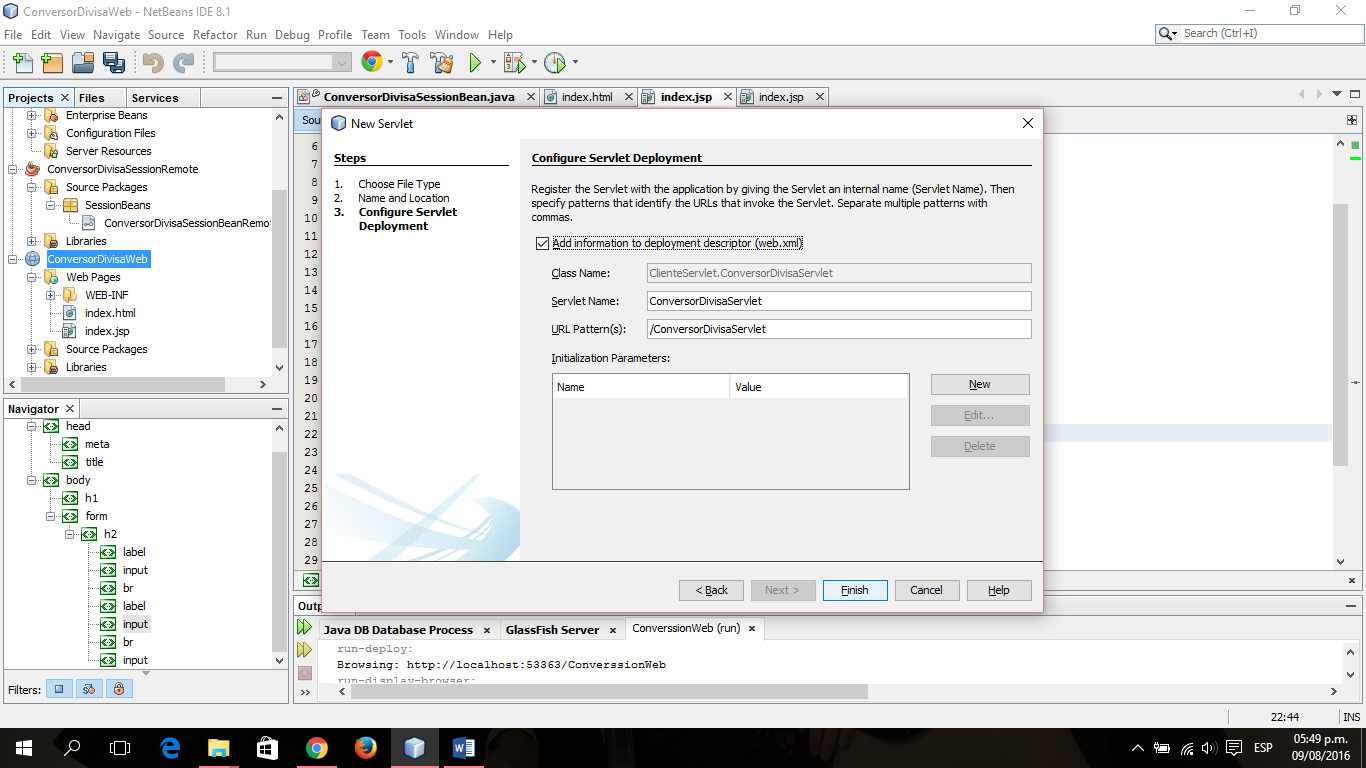


*Figura 9: Página JSP con formulario*

En la implementación debemos recalcar que este invocará en su método POST al servlet que interactúa con el contenedor EJB (*ConversorDivisaServlet*) comunicando ambos parámetros.

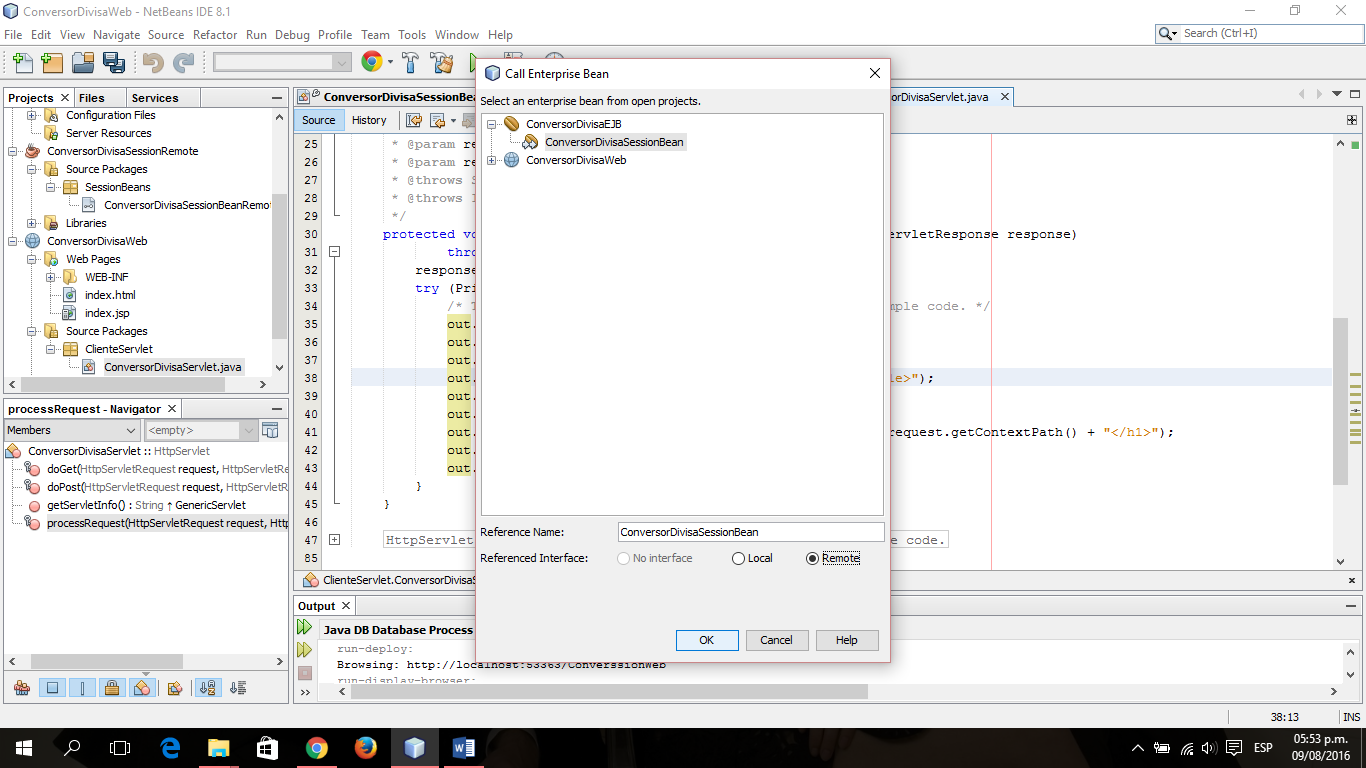
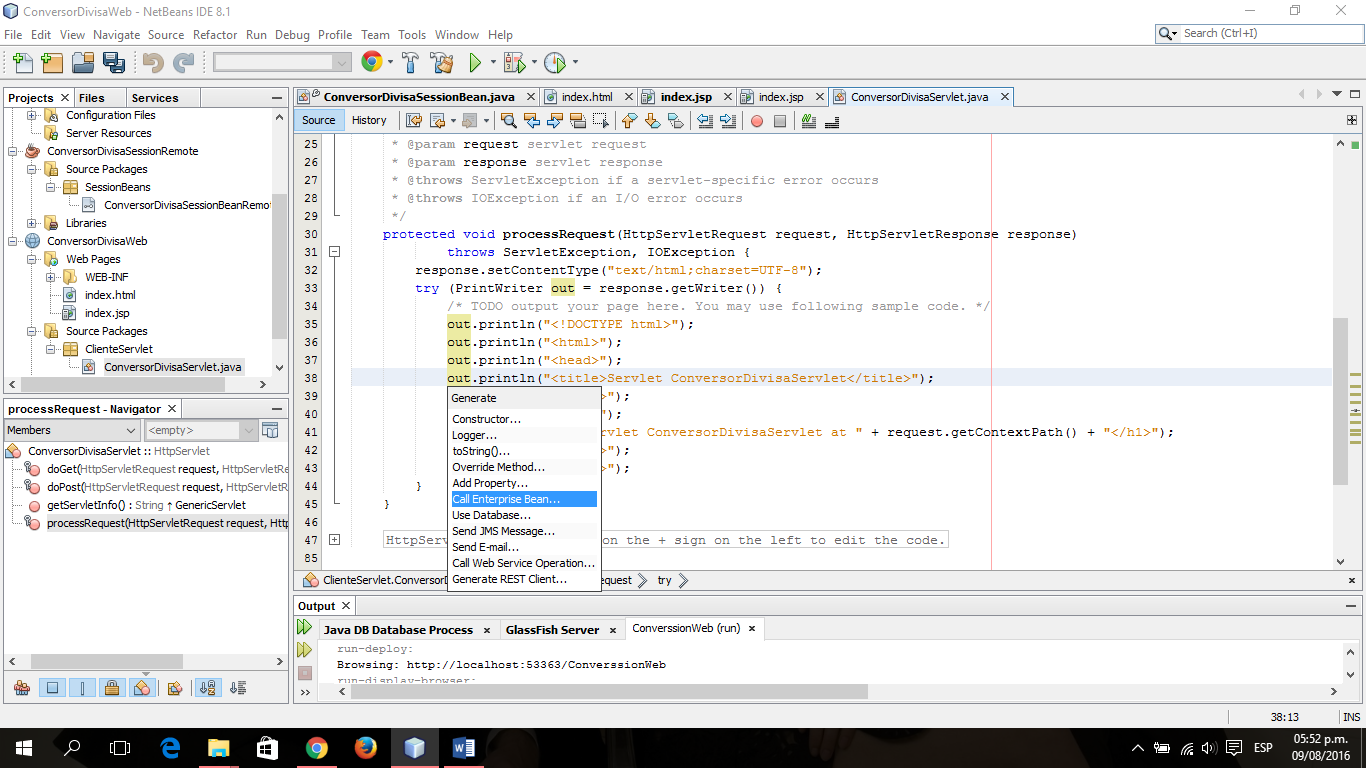
1. Ahora debemos crear el Servlet que realiza la comunicación con el contenedor EJB, este se llamará *ConversorDivisaServlet*. En la aplicación Web se añade nuevo->servlet. Hay que acotar que estamos indicando que automáticamente añada su información al archivo de despliegue XML.

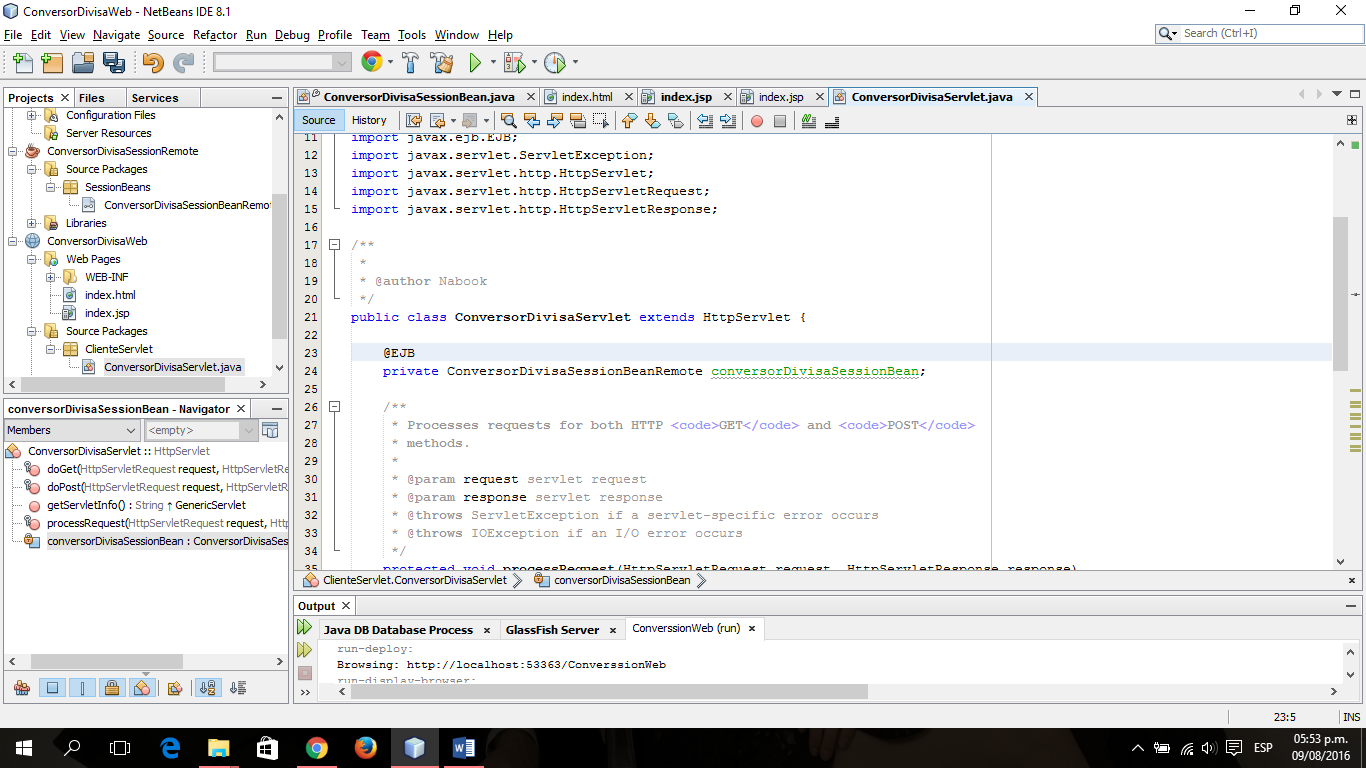




*Figura 10: Creación Servlet - ConversorDivisaServlet*

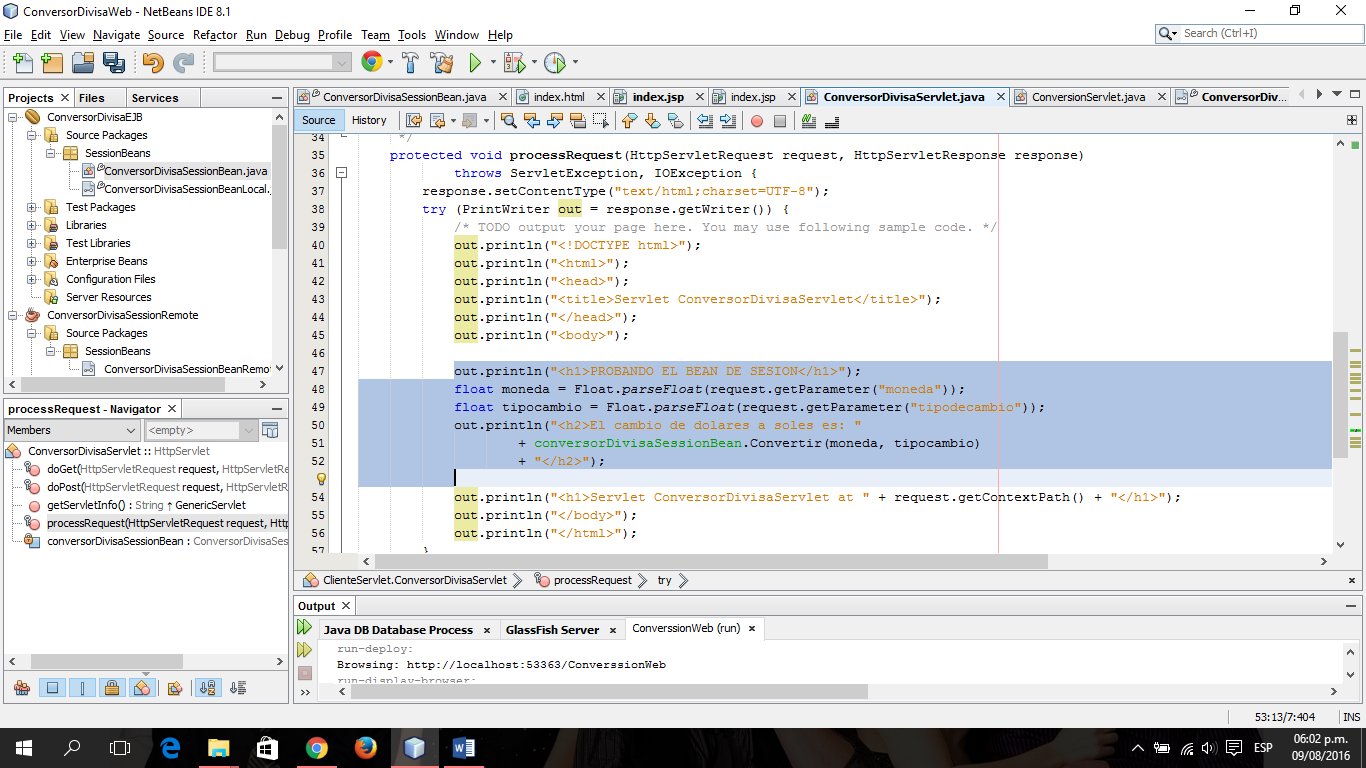
1. Ahora que se creó el servlet debemos realizar dos actividades: primero añadir la llamada a la interfaz remota del Bean de Sesión (menú contextual->insertar código->call Enterprise bean)





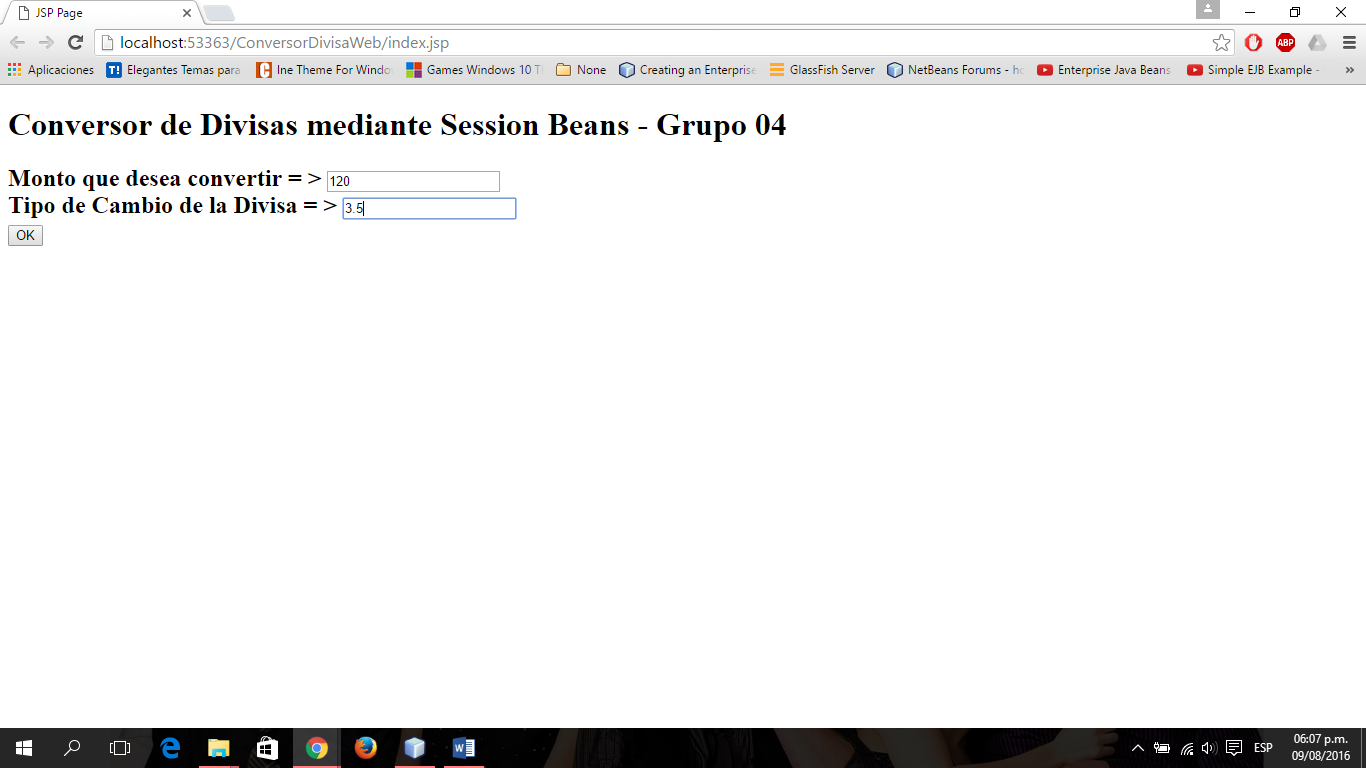
*Figura 11: Agregar la referencia a la interfaz remota del bean de sesión en el Servlet*

Y ahora procedemos a editar el código del servlet para que obtenga los datos del formulario de la página JSP, haga la llamada al bean de sesión, y presente los resultados obtenidos de la llamada y ejecución del bean de Sesión, específicamente de su método Convetir.



*Figura 12: Servlet – invocación del método del sesión bean*

1. Ahora solo desplegamos la solución y proyecto y vemos sus resultados:





*Figura 13: Despliegue y prueba del Bean de Sesión*

# DISCUSIÓN

# CONCLUSIONES

1. Los Enterprise javabeans facilitan la implementación de código, ya que permiten al desarrollador avocarse a los procesos específicamente de negocio, y evitando el desarrollo de código engorroso para acceder a las BD, o para estructurar el modelo vista controlador; que puede ser repetitivo, o incluso propenso a generar entropía, evitando que la solución sea adaptable de manera práctica a otros entornos.
2. La estandarización que llevan los Enterprise javabeans permite que los componentes creados sean reutilizables por cualquier otro desarrollador, agrega escalabilidad, flexibilidad al desarrollo de soluciones.
3. Los Enterprise javabeans por su estructura y desarrollo han facilitado la interoperabilidad entre aplicaciones e incluso la integración con sistemas no java, como sistemas ERP, u otros sistemas complejos o mainframes a través de sus APIs.
4. Es una infraestructura que separa de manera efectiva la lógica de presentación de la de negocio, facilitando desarrollos paralelos de forma más rápida y efectiva, ya que mientras los desarrolladores generan los beans que atiendan los procesos de negocio, encapsulando los mismos; delega la lógica de presentación al diseñador Web sin preocuparse de como formatear la salida.

# RECOMENDACIONES

1. Aun siendo una infraestructura sólida para el desarrollo de aplicaciones, siempre se debe evaluar si es la solución indicada con respecto al contexto del proyecto a implementar.
2. Se debe tener prudencia para la selección del tipo de beans a implementar de acuerdo a las necesidades y procedimientos de negocio a implementar, por lo cual se recomienda un análisis exhaustivo antes de la implementación.
3. En ocasiones el desarrollo de aplicaciones puede resultar muy complejo, por lo cual se recomienda llevar una biblioteca de datos y procedimientos durante el desarrollo.
4. En la implementación de interfaces locales debemos considerar que en pos del máximo rendimiento, manejan paso de parámetros por referencia en lugar de por valor. Esto implica un cambio importante en la semántica del código y es una fuente importante de errores.
5. Llevar un control de cambios, mayor aún con el uso de las IDE ante una modificación en las declaratorias, y por el código autogenerado, debe tener precaución en las modificaciones de declaratorias durante el proceso de desarrollo.

# BIBLIOGRAFÍA

* **Goncalves Antonio (Agosto, 2010). Java EE 6 Platfform with Glassfish 3**. Francia. Appress
* **Richard Monson Haefel y Bill Burke (Mayo, 2006). Enterprise JavaBeans 3.0.** Estados Unidos.O’Reilly Media
* **Kathy Sierra y Bert Bates (Octubre 2003). Head First EJB.** Estados Unidos. O’Reilly Media
* **Oracle Official Page. Enterprise JavaBeans Technology.** [en línea] Disponible en: www.oracle.com/technetwork/java/javaee/ejb/index.html
* **Jair Rillo Junior (November, 2009). Creating EJB3 Sessions Beans using Netbeans 6.1 and Glassfish** [en línea], Community NetBeans, Disponible en: wiki.netbeans.org/CreatingEJB3UsingNetbeansAndGlassfish
* **Diaz Gomez, Pedro (Enero, 2007). Ejemplo Sencillo con EJB** [en línea], Chuwiki.org. Disponible en: chuwiki.chuidiang.org/index.php?title=Ejemplo\_Sencillo\_con\_EJB
* **Netbeans Support Cast (Mayo, 2008). The NetBeans E-commerce Tutorial.** [en línea], Community NetBeans. Disponible en: netbeans.org/kb/docs/javaee/ecommerce/entity-session.html
* **Netbeans Support Cast (Junio, 2012). Creating an Enterprise Application with EJB 3.1.** [en línea], Community NetBeans. Disponible en: netbeans.org/kb/docs/javaee/javaee-entapp-ejb.html
* **Global Mentoring (Julio, 2012). Introducción a los Enterprise Javabeans.** [en línea] Disponible en: globalmetoring.com.mx/cursos-java/java-empresarial/introducción-a-los-enterprise-javabeans/
* **Wikipedia. (2016). Enterprise JavaBeans.** [en línea] Disponible en: en.wikipedia.org/wiki/Enterprise\_JavaBeans
* **Palermo Rodriguez, Rolando. (Marzo, 2012). Introducción a los EJB.** [en línea] Disponible en: blog.rolandopalermo.com/2012/03/ejb-30-netbeans-java.html
* **YNR Tutorials Channel (Enero 2014). Enterprise Java Beans** [Video][en línea] Disponible en: www.youtube.com/watch?v=giUULtliGs