



## GUÍA DIDÁCTICA PARA CONSTRUIR UN CLIENTE/SERVIDOR SERVICES EN JAVA USANDO EL IDE NETBEANS Y JPA COMO MECANISMO DE PERSISTENCIA RDO A LOS DATOS.

### INTRODUCCIÓN:

Durante el seguimiento de esta Guía Didáctica el lector podrá aprender a construir un sistema distribuido mediante el uso de la tecnología WebServices o Servicio Web, su integración con JPA como mecanismo de persistencia tipo RDO (Relational Data Object), es decir pasar de Tablas Relacionales a Objetos Java, luego la construcción de un servicio Web sobre el servidor Apache Tomcat con una operación simple que permita agregar Objetos como Registros en una base de datos de forma remota, a través de conexiones realizadas desde una aplicación Cliente.

### TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS EN ESTA GUÍA DIDÁCTICA.

**Java EE:** Es un conjunto de especificaciones estándares escritas en Java, las cuales conforman una plataforma de desarrollo de muy alto nivel conocida como Java Enterprise Edition

Sitio Web: <http://www.oracle.com/technetwork/java/javaee/overview/index.html>

**JPA:** Java Persistence API, forma parte de Java EE, su objetivo es facilitar el desarrollo de aplicaciones que realizan operaciones de persistencia de datos sobre Bases De Datos Relacionales.

Sitio Web: <http://www.oracle.com/technetwork/java/javaeetech/persistence-jsp-140049.html>

**WebServices:** Forma parte tanto de Java SE o Standard Edition como de Java EE. Su objetivo es facilitar la construcción e implementación de aplicaciones distribuidas basadas en Servicios Web HTTP.

Sitio Web: <http://www.oracle.com/technetwork/java/webservices-136604.html>

Sitio Web: <http://www.oracle.com/technetwork/java/javase/tech/webservices-jsp-136868.html>

**Java Swing:** Forma parte de Java Standard Edition, Es una biblioteca de clases Java que facilitan el desarrollo de aplicaciones de escritorio con Interface Gráfica de Usuario.

Sitio Web: <http://docs.oracle.com/javase/tutorial/uiswing/>

**MySQL:** Es un motor de bases de datos relacionales que soporta arquitectura cliente servidor, distribuidos como OpenSource auspiciado por Oracle. - Sitio Web: <http://www.mysql.com/>

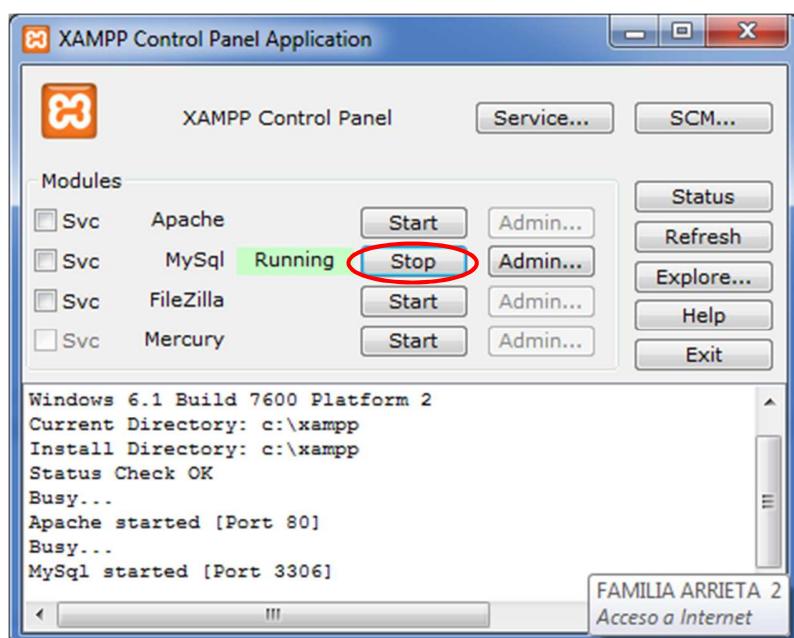
**Netbeans:** Es un IDE o Entorno Integrado de Desarrollo, inicialmente concebido para facilitar y agilizar el desarrollo de aplicaciones escritas en Java, pero con el tiempo ha evolucionado mejorando mucho más el desarrollo en Java y otras tecnologías como C/C++, JavaScript, HTML, JavaME, SQL, etc. Es distribuido bajo licencia de Software Libre GPL -Sitio Web: <https://netbeans.org/>

### Paso 1: INICIAR EL SERVIDOR DE BASES DE DATOS:

En esta guía se asume que el lector ya tiene conocimientos básicos sobre bases de datos, para este ejemplo se ha creado una base de datos llamada **jpa\_bd** sobre el motor de bases de datos MySQL, para simplificar el desarrollo de la guía, esta base de datos solo tendrá una tabla llamada **Usuario** con 3 campos, el primero llamado **id** de tipo

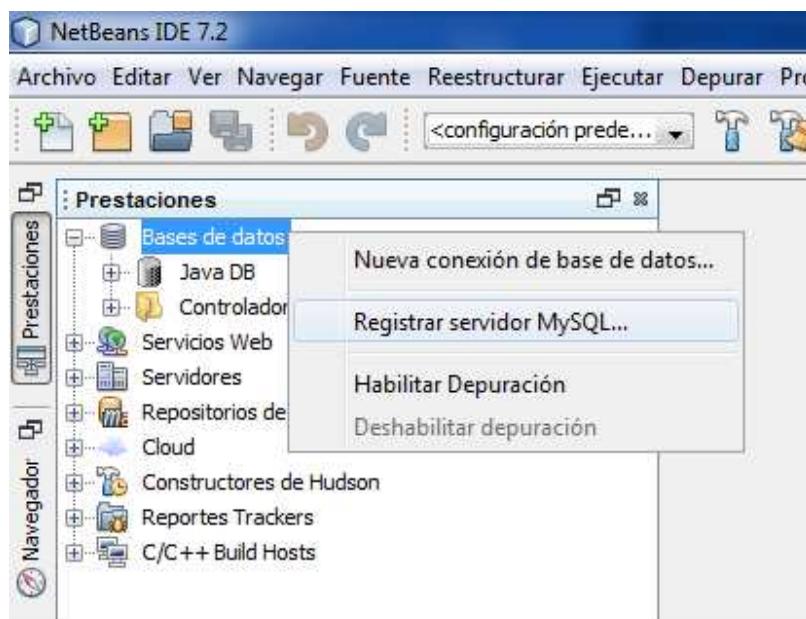


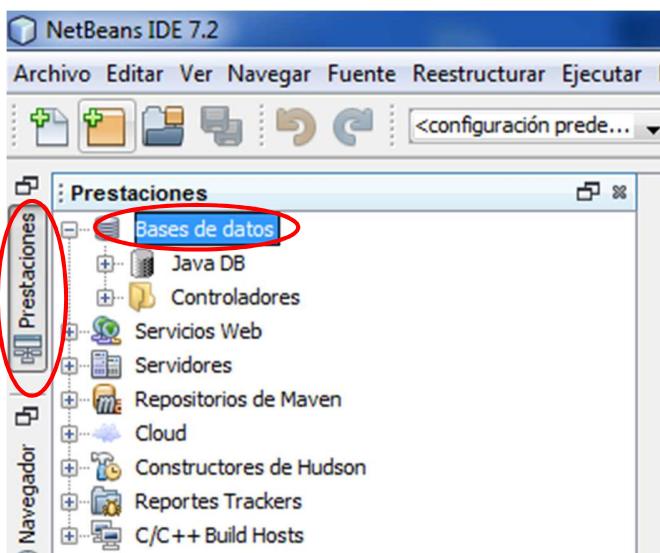
Entero, no acepta Nulos, es clave primaria y Auto Incremental, el segundo campo llama **password**, de tipo Texto y acepta valores vacíos o Nulos, el tercer y último campo llamado **nombre** es de tipo texto y no acepta Nulos. Para instalar y configurar automáticamente MySQL usaremos XAMPP.



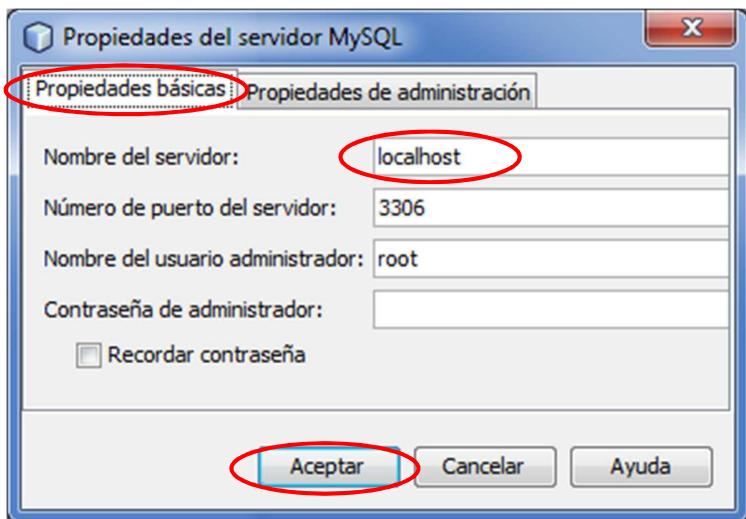
## Paso 2: INICIAR EL IDE NETBEANS Y CONFIGURARLO PARA SU CONEXIÓN A MYSQL

Para este paso se sugiere seguir la secuencia de las siguientes imágenes:





Al seleccionar el panel de Prestaciones del IDE Netbeans, seleccionar y hacer click derecho sobre el ítem Bases de Datos, procedemos a escoger la opción Registrar Servidor MySQL, proceso que nos muestra la siguiente ventana:

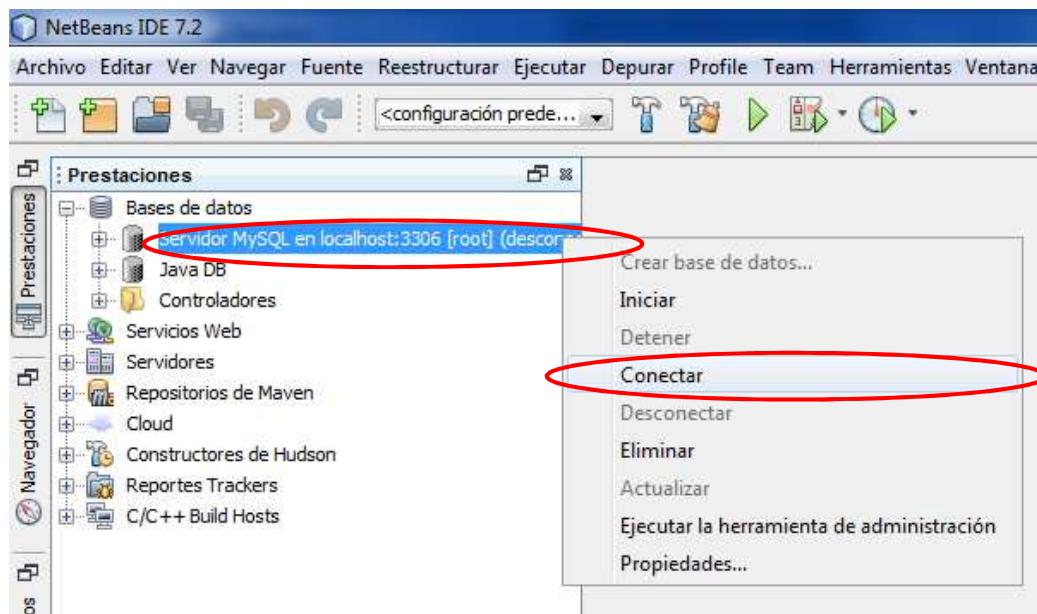


Sobre la cual debemos ingresar los parámetros de conexión hacia el servidor MySQL, estos parámetros aparecen por defecto en esta pantalla, por lo que no necesitamos configurar o cambiar nada adicional.

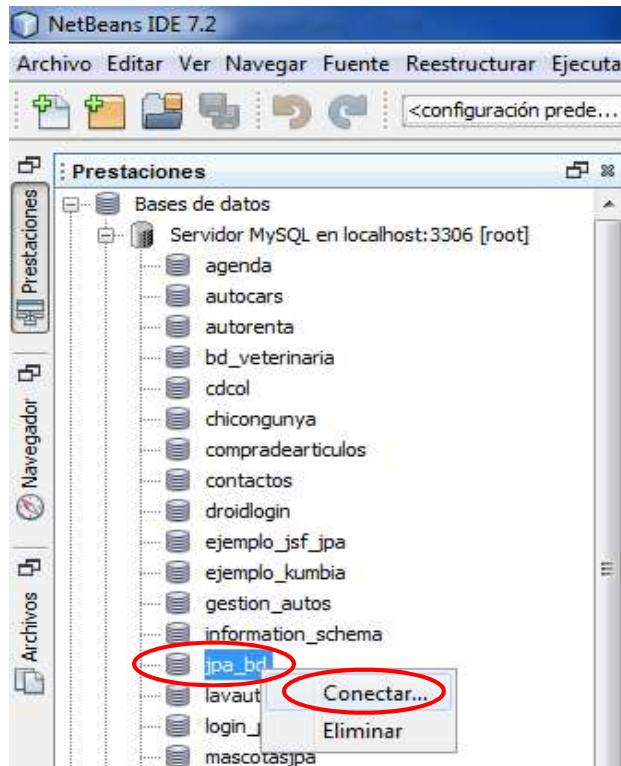
**localhost** hace referencia a la dirección ip 127.0.0.1 que posee todo computador con tarjeta de Red, usamos esta dirección porque el servidor de bases de datos MySQL se encuentra en ejecución localmente dentro del PC donde estamos desarrollado nuestra ampliación, si estuviese en otro PC conectado por Red a este PC.

**3306** es el puerto lógico de conexión por defecto del servidor MySQL.

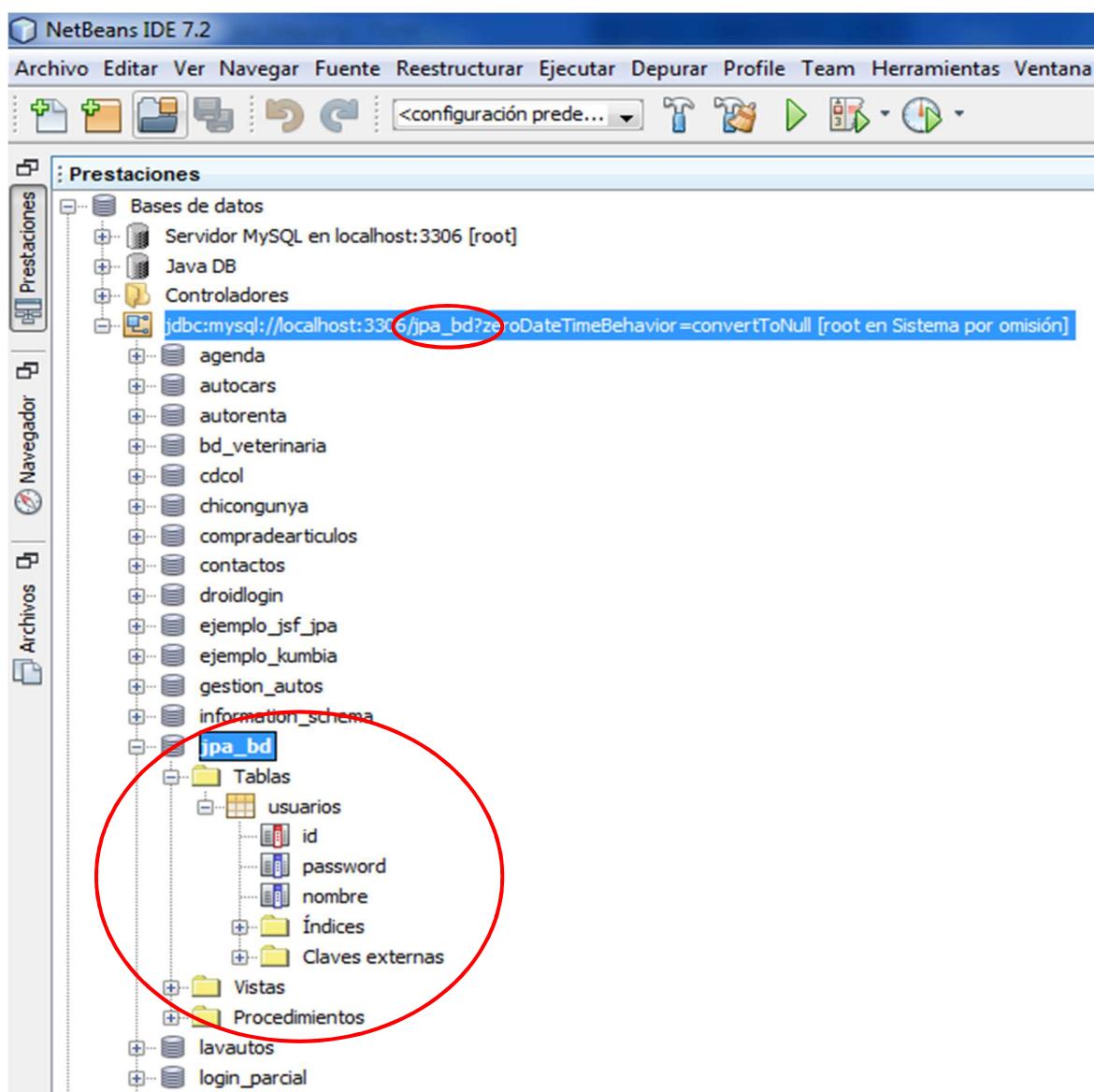
**root** es el identificador de usuario administrador por defecto sin password.



Una vez configurados los parámetros de conexión entre Netbeans y el Servidor BD MySQL, debemos establecer una conexión.



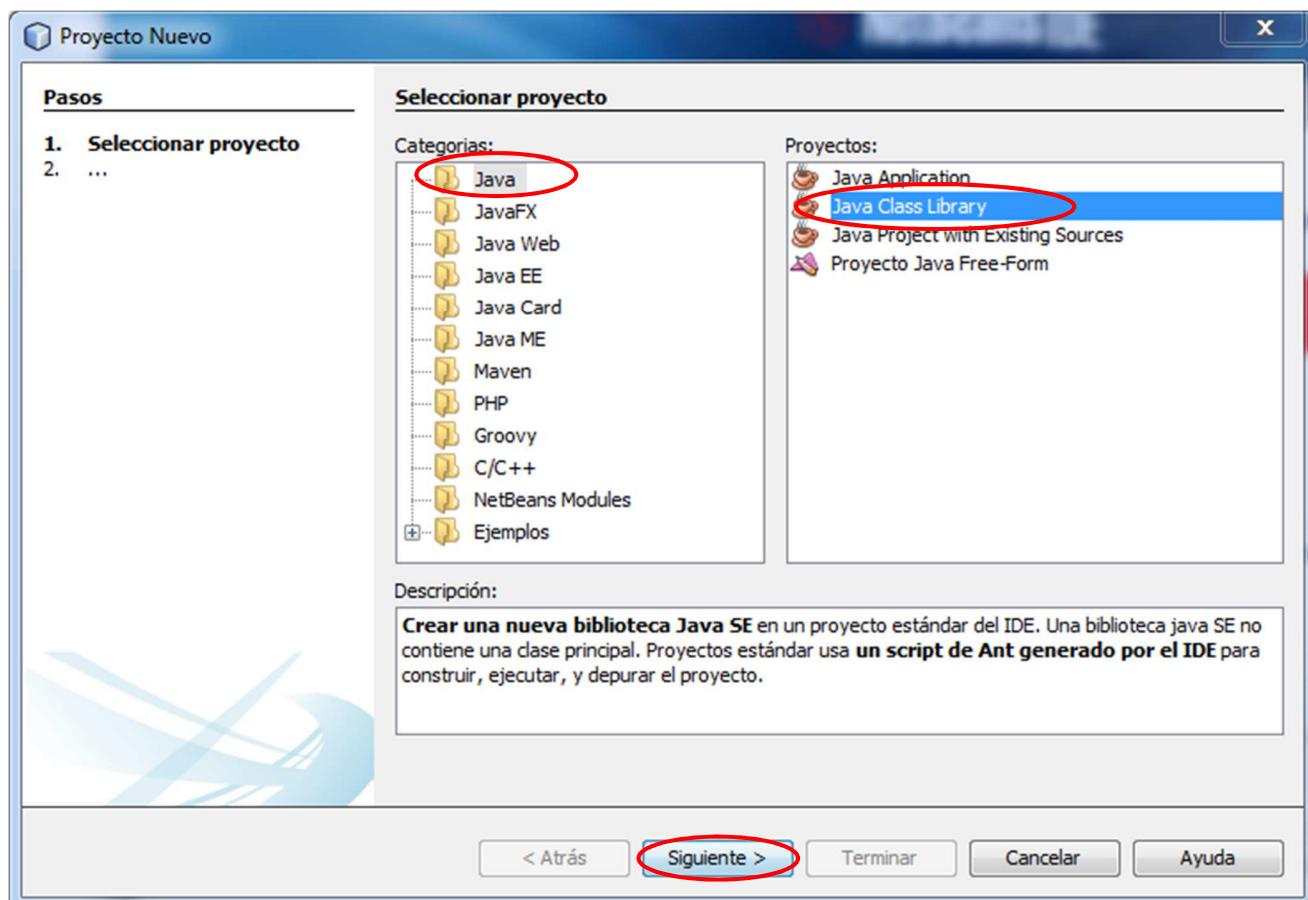
Realizada la conexión con el Servidor de BD MySQL procedemos a realizar la conexión con la base de datos **jpa\_bd**, los parámetros de conexión para este proceso por defecto son los mismos de conexión al servidor, es decir usuario es root sin password, luego podemos ubicar y explorar nuestra base de datos dentro del árbol conexión.





En las anteriores secuencias de imágenes aprendimos los pasos para preparar y establecer conexión entre el IDE Netbeans y el Servidor BD, observamos como desde el IDE podemos acceder a las distintas BD y su estructura.

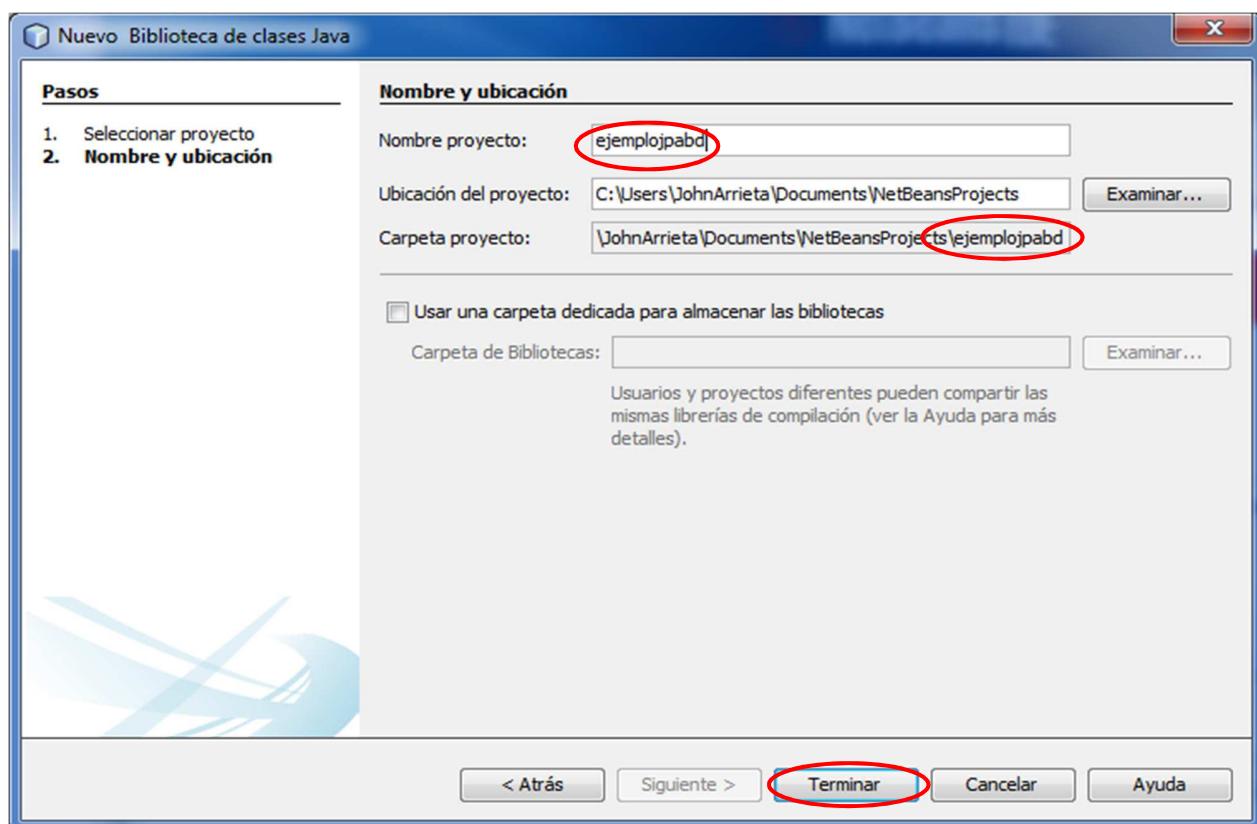
### Paso 3: CREAR UN PROYECTO TIPO LIBRERÍA JAVA (.jar)



Para crear un proyecto tipo librería Java desde este IDE, primero se debemos ir al menú **Archivo/Nuevo Proyecto**, nos aparece la ventana anterior en la cual debemos escoger la categoría Java y el Tipo Proyecto debe ser Java Class Library.

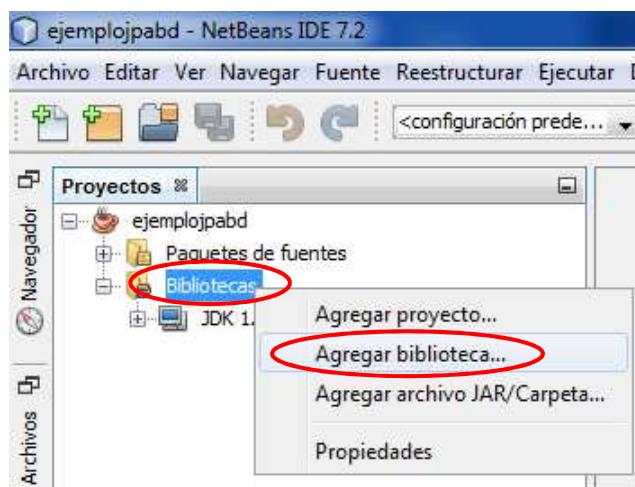
Un proyecto de librería normalmente no requiere de una clase que actué como punto de inicio o ejecución (método main), ya que el programa será utilizado como complemento requerido por una aplicación más grande, la ejecución de una librería generalmente es ordenada y controlada por la aplicación que la incorpora o utiliza.

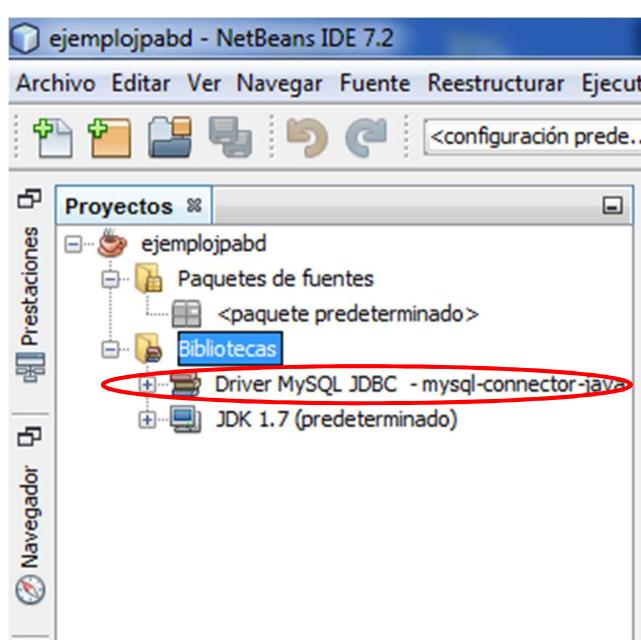
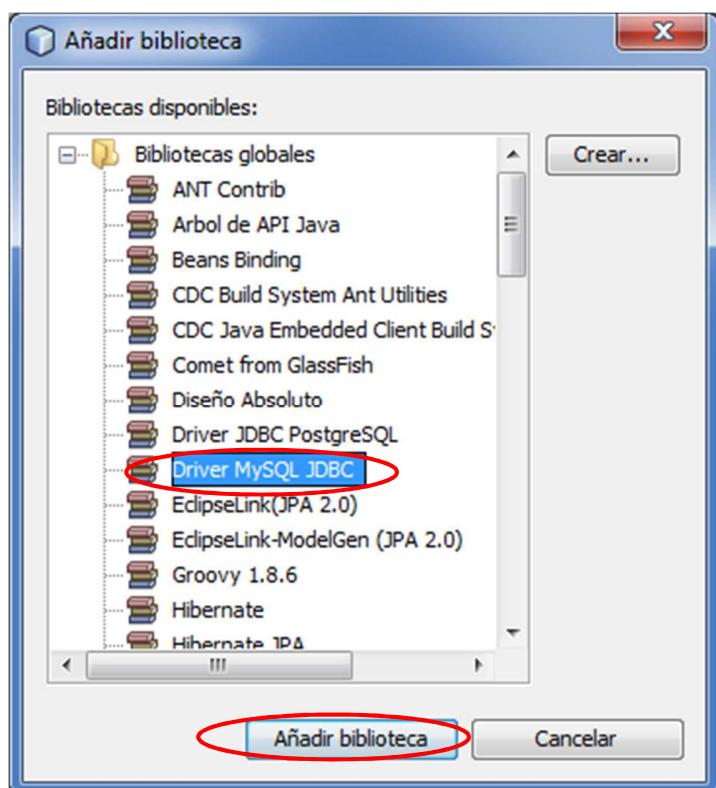
En nuestro ejemplo vamos a crear 3 proyectos, uno de tipo **librería** que contendrá las clases que permitan realizar operaciones sobre la tabla usuario de base de datos **jpa\_bd**, un segundo proyecto que será el **webservices** que permitirá publicar en la red (LAN o WAN) operaciones (por ejemplo **agregarUsuario**) para que sean ejecutadas o invocadas remotamente por otras aplicaciones, un tercer proyecto de tipo **ClienteWebServices** cuyo objetivo será consumir el servicio web publicado por el segundo proyecto. A continuación debemos ingresar los parámetros del proyecto, nombre, ubicación y carpeta raíz



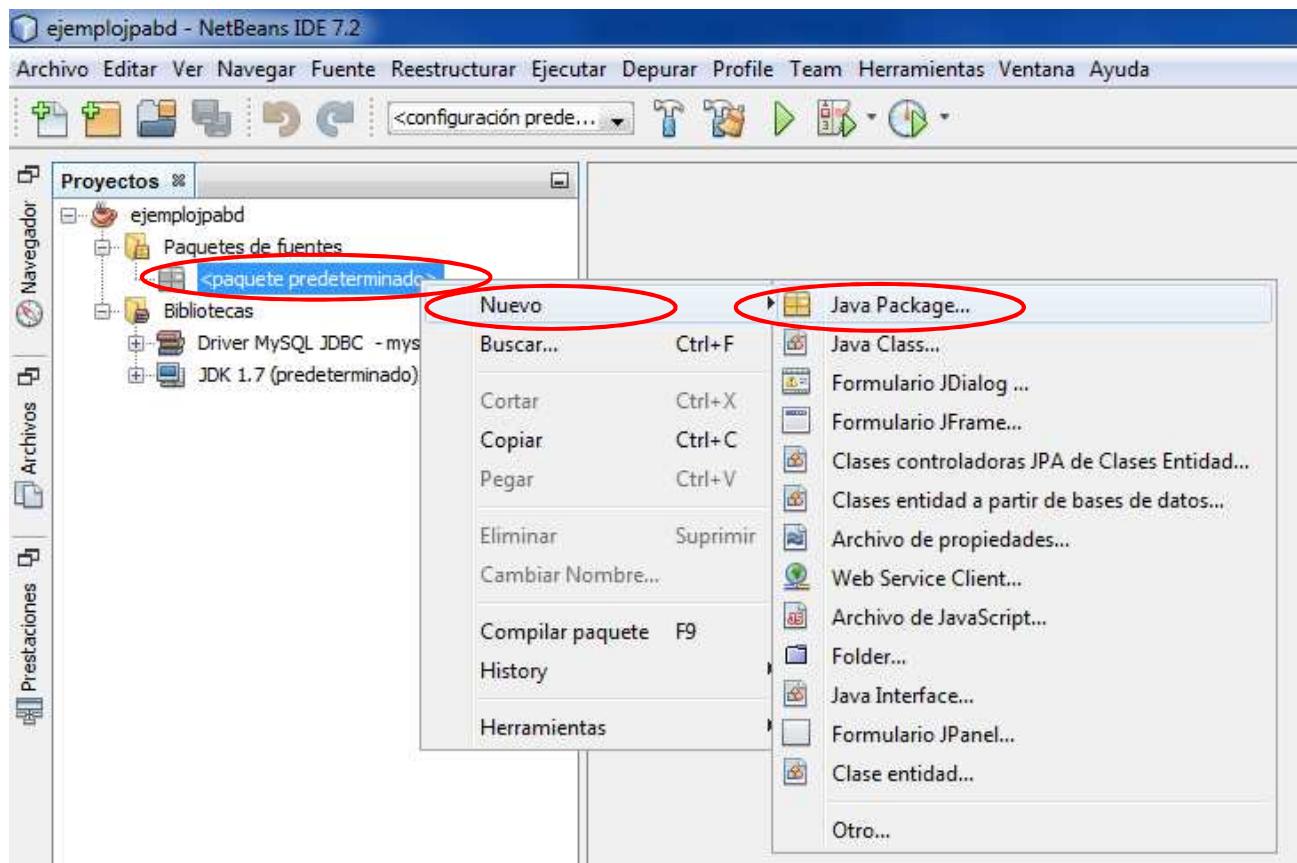
#### Paso 4: AGREGAR LA BIBLIOTECA DE CLASES DRIVER-MySQL JDBC

Para poder establecer conexión desde nuestro código Java hacia una base de datos implementada en un servidor de Bases de datos, necesitamos de un Driver (Manejador) que actua como puente entre las ordenes SQL enviadas desde Java y el motor de Base de Datos, en este caso como el motor de BD es MySQL requerimos de un conjunto de clases especialmente diseñadas para establecer conexión entre Java y MySQL, esta clases están empaquetadas en un archivo .jar llamado DIVER-MySQL-JDBC, el cual debemos incluir como librería de nuestro proyecto.

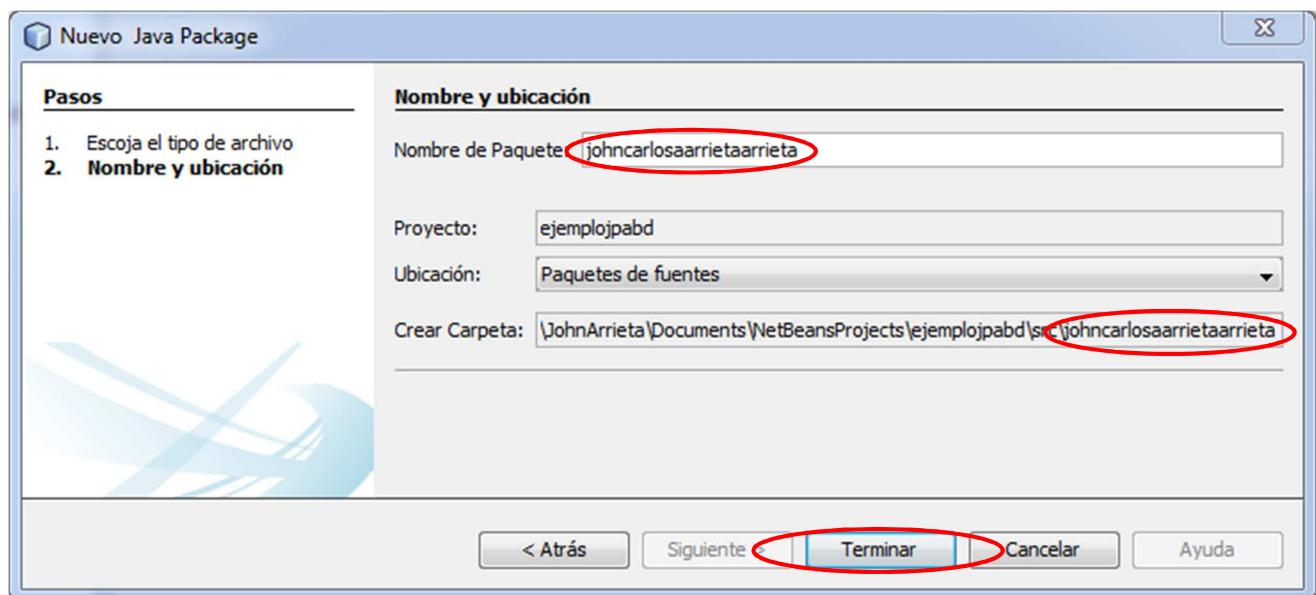




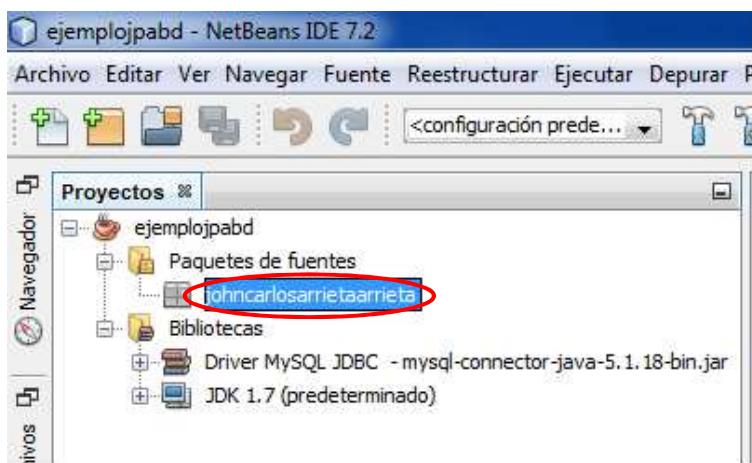
Como pudimos apreciar como es habitual el proceso consta de un simple conjunto de click por aquí y click por allá, esta es una de las ventajas que tenemos de utilizar un IDE robusto como Netbeans. Incluido el driver de conexión como librería o biblioteca de clases a nuestro proyecto, procedemos a crear los elementos y estructura de la aplicación, para ello iniciamos creando y definiendo los paquetes en los cuales se organizará el código.



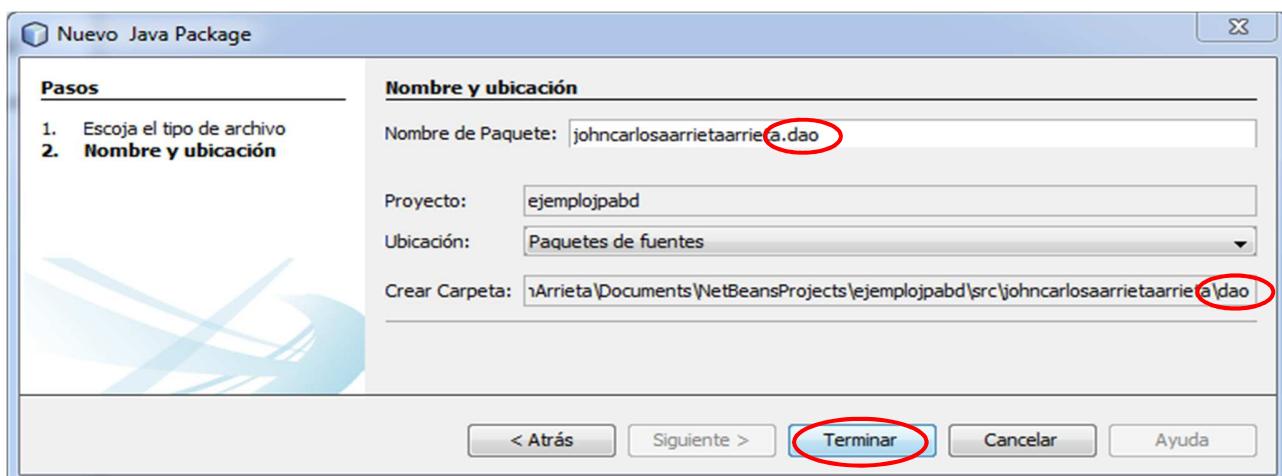
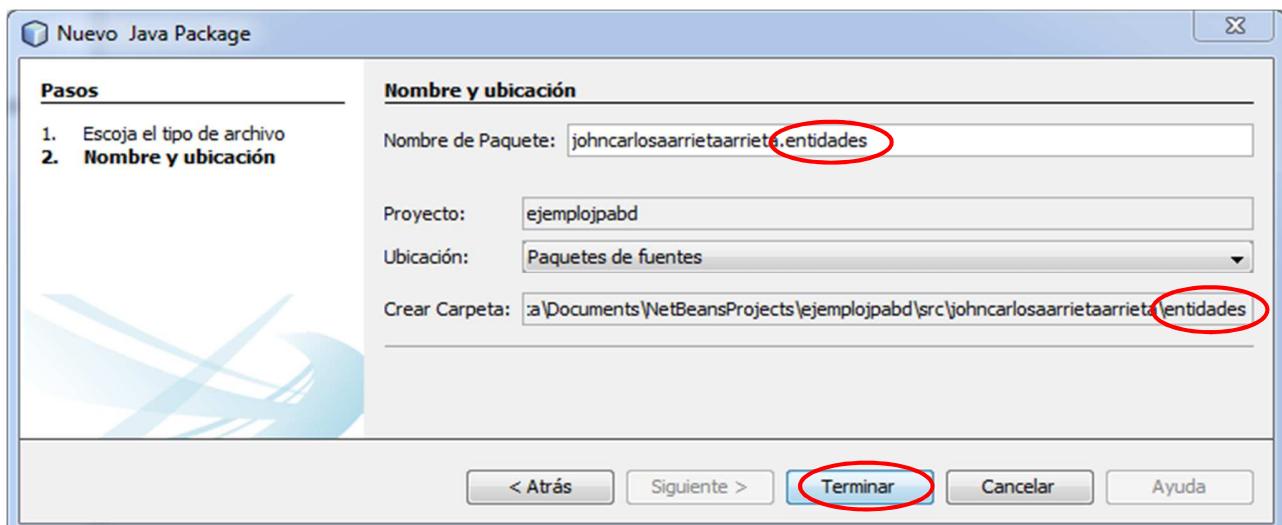
Ingresamos el nombre del primer paquete.



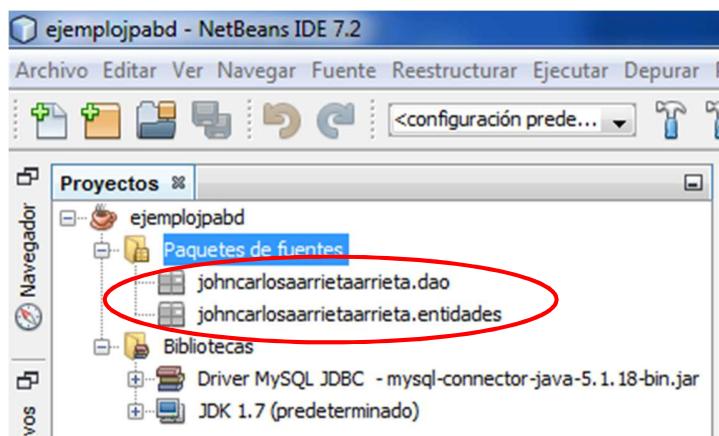
Luego realizamos el mismo procedimiento sobre el paquete inicial, pero esta vez agregamos otro paquete al final de este, notar que los paquetes se separan mediante . Punto.



Hacemos clic derecho sobre el paquete inicial y seleccionamos Nuevo/ Java Paquete e ingresamos el nombre

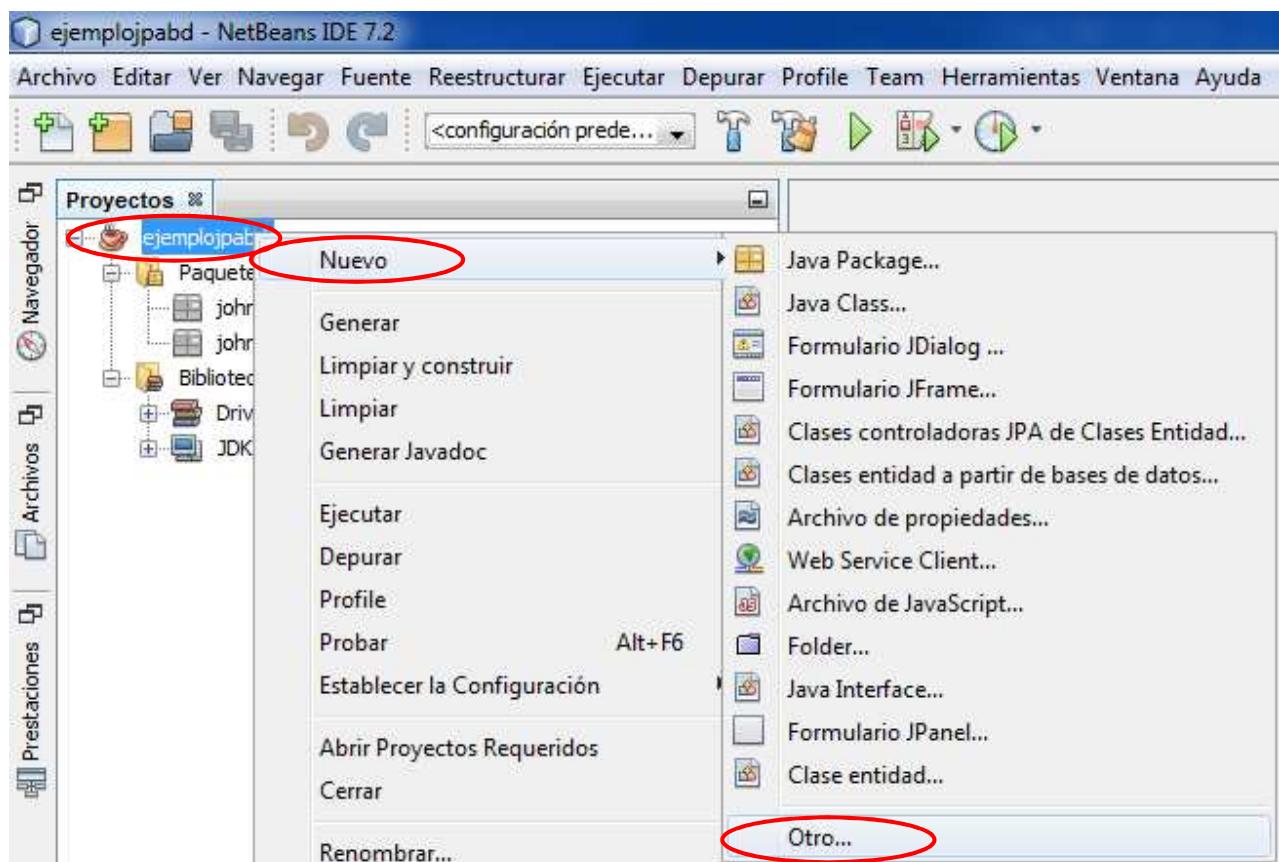


Igual hacemos para crear otro paquete, en total creamos un paquete raíz y dos paquetes dentro de este último.

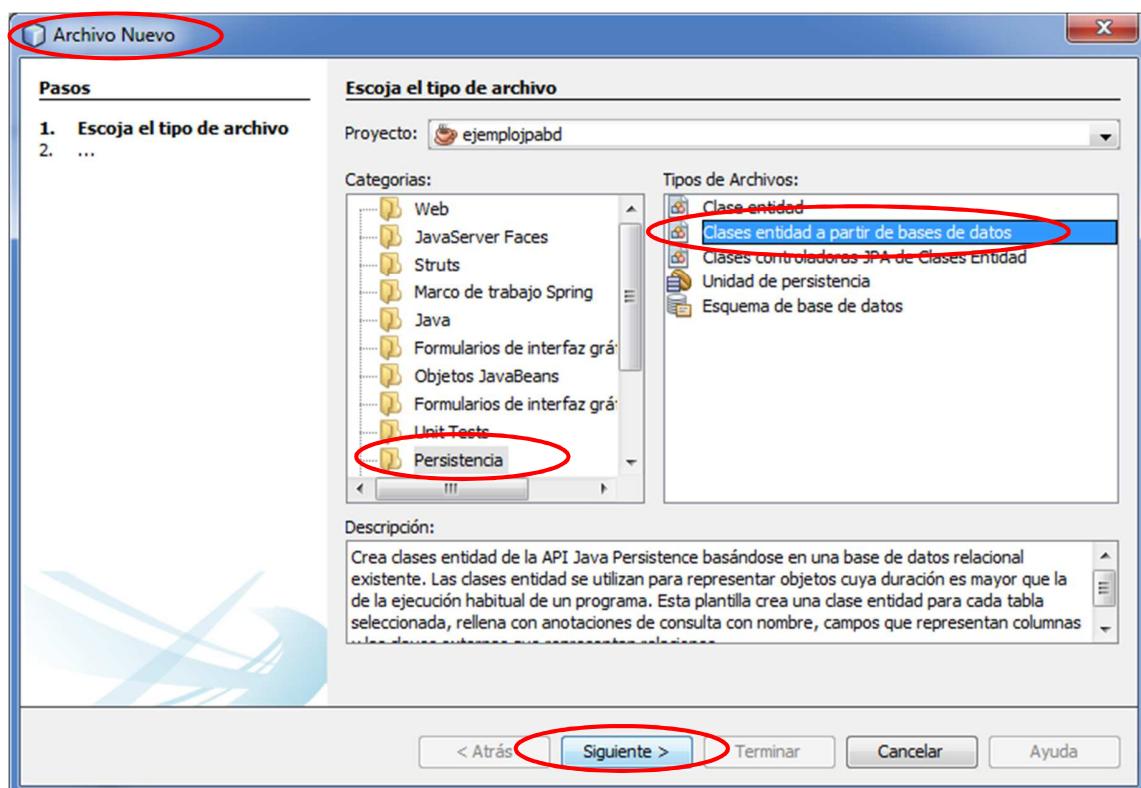


Como se puede apreciar, en el árbol de nuestra aplicación quedan 3 paquetes, johncarlosarrieta, dao y entidades

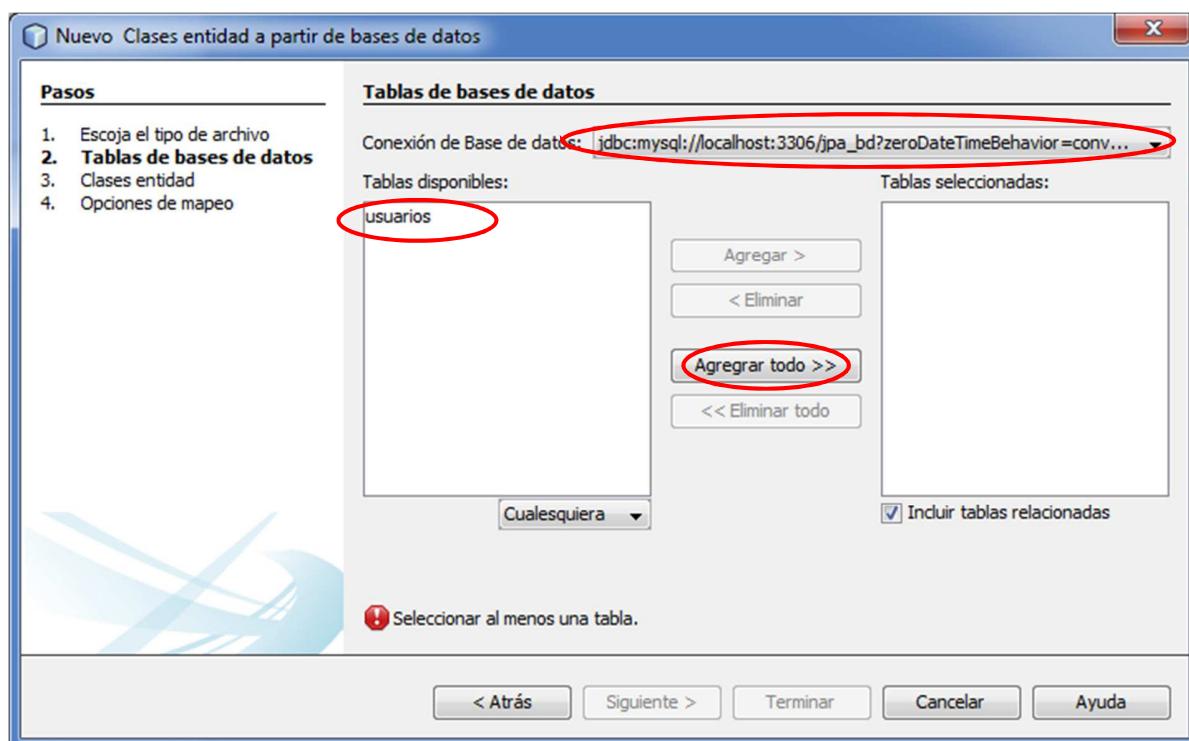
## Paso 5: CREAR LAS CLASES ENTIDADES A PARTIR DE LAS TABLAS DE NUESTRA BD.

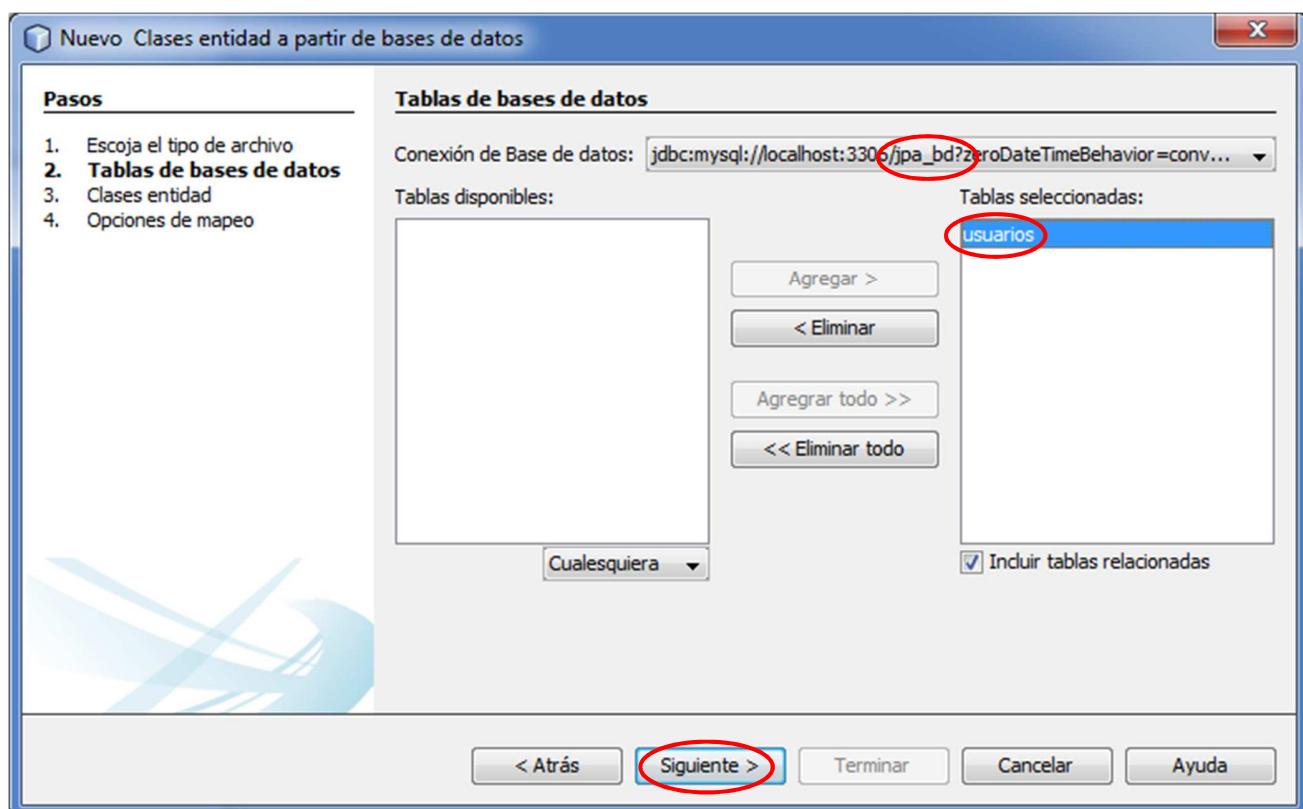


En este paso vamos a utilizar el generador de código de Netbeans para crear el código fuente de las clases Entidades, las cuales representan las Tablas de la BD como clases Java, estas clases se utilizan para tener un modelo Orientado a Objetos a partir del modelo Entidad Relación propio de la base de datos relacional que creamos dentro de un motor de BD como lo es MySQL .

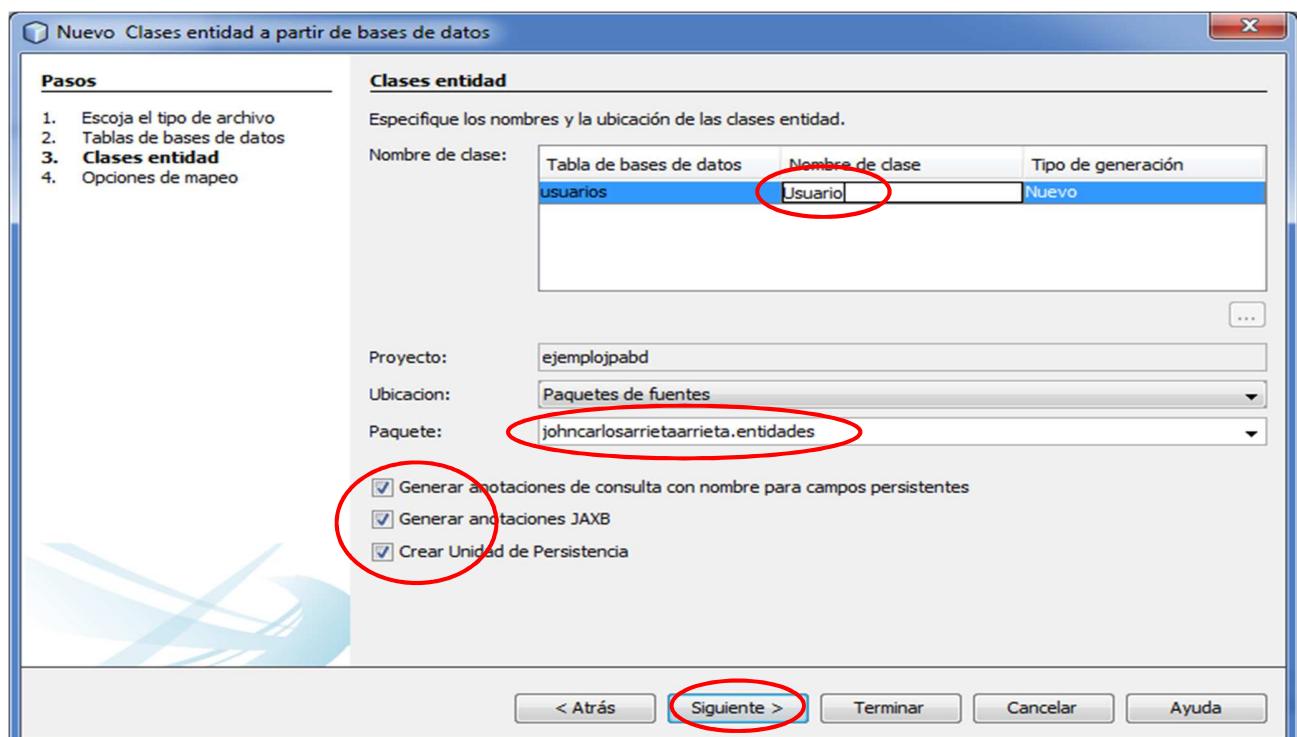


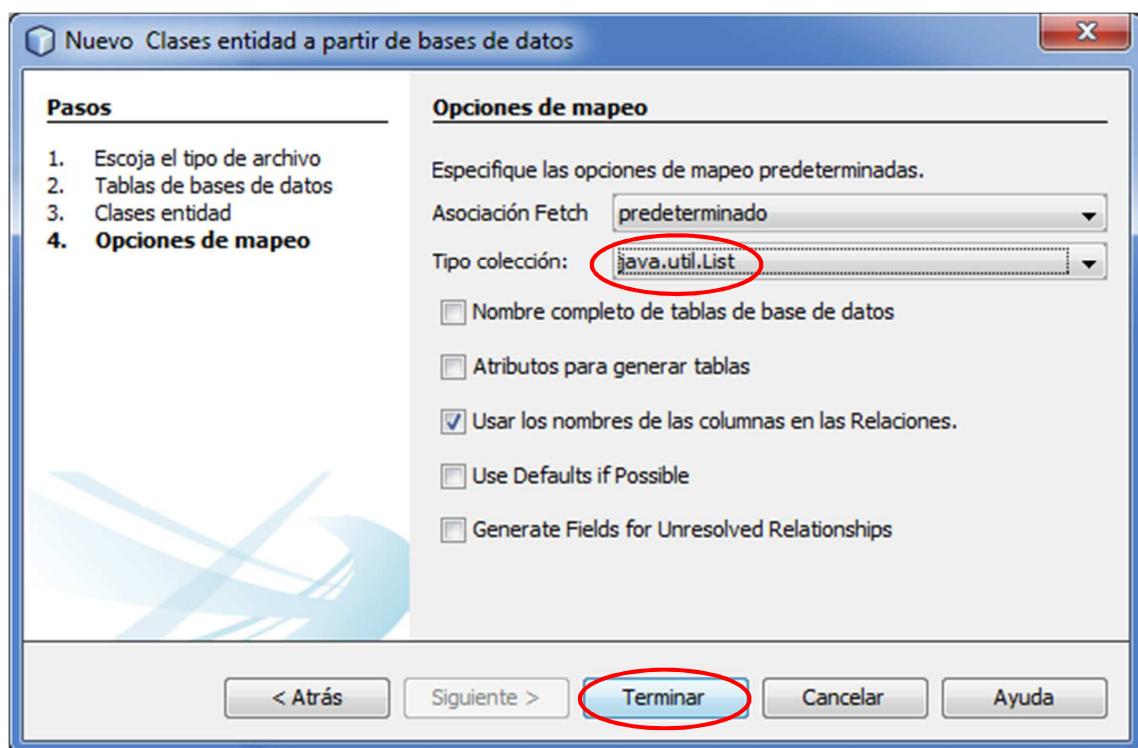
Seleccionamos click derecho sobre el proyecto/ Nuevo / Otros y nos aparece la ventana anterior.





Debemos escoger la conexión a la base de datos que configuramos anteriormente, así como sus las tablas





Estos pasos nos permiten generar de forma automática el código de las clases Entidades tomando como fuente la información la estructura de la base de datos (Nombre de tablas y de campos, tipo de campos, claves, índices, relaciones entre las tablas, etc.).

Por ejemplo, teniendo al siguiente tabla,

Este sería un ejemplo del código para la clase entidad de dicha tabla, observen el uso de las @anotaciones Java.

```
import java.io.Serializable;
import javax.persistence.*;
import javax.xml.bind.annotation.XmlRootElement;
@Entity
@Table(name = "usuarios")
public class Usuario implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Column(name = "password")
    private String password;
    @Column(name = "nombre")
    private String nombre;
    public Usuario() { }
    public Usuario(Integer id) { this.id = id; }
    //.... Metodos getXX y setXX
    public Integer getId() { return id; }
    public void setId(String id) { this.id = id; }
}
```



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (Left):** Shows the project "ejemplopabd" with its structure:
  - Paquetes de fuentes:
    - META-INF
    - johncarlosarrietaarrieta.dao
    - johncarlosarrietaarrieta.entidades
      - Usuario.java (highlighted with a red circle)
  - Bibliotecas:
    - Driver MySQL JDBC - mysql-connector-java-5.1.46-bin.jar
    - EclipseLink(JPA 2.0) - eclipselink-2.3.1.jar
    - EclipseLink(JPA 2.0) - javax.persistence-2.1.0.jar
    - EclipseLink(JPA 2.0) - org.eclipse.persistence-2.5.3.jar
    - JDK 1.7 (predeterminado)
- Editor Area (Right):** Displays the code for **Usuario.java**. The code defines an Entity class named **Usuario** with annotations for persistence and querying. It includes imports for Serializable, Basic, Column, Entity, GeneratedValue, GenerationType, Id, NamedQueries, NamedQuery, Table, and XmlRootElement. The class implements Serializable and has a constructor with an integer parameter.

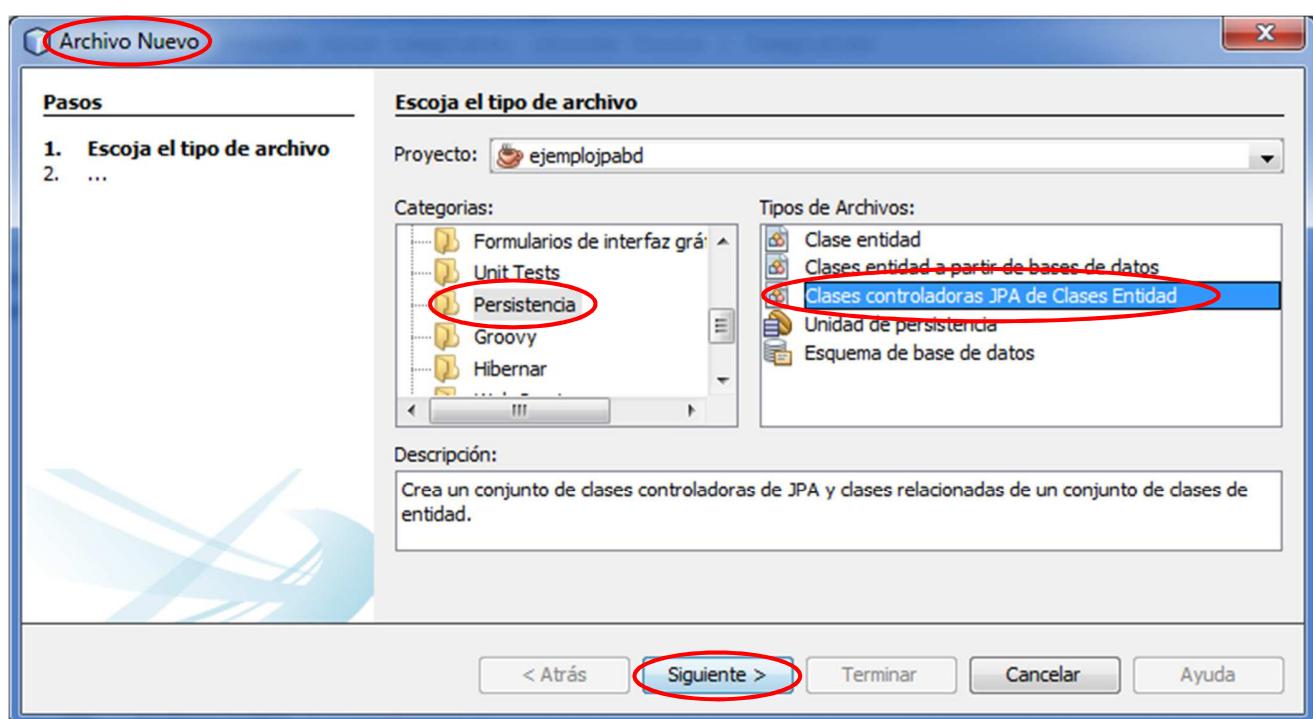
```
package johncarlosarrietaarrieta.entidades;
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;
/*
 *
 * @author JohnArrieta
 */
@Entity
@Table(name = "usuarios")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM Usuario u"),
    @NamedQuery(name = "Usuario.findById", query = "SELECT u FROM Usuario u WHERE u.id = :id"),
    @NamedQuery(name = "Usuario.findByPassword", query = "SELECT u FROM Usuario u WHERE u.password = :password"),
    @NamedQuery(name = "Usuario.findByNombre", query = "SELECT u FROM Usuario u WHERE u.nombre = :nombre")})
public class Usuario implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
```



## Paso 6: CREAR LAS CLASES DAO O CONTROLADORES PERSISTENCIA A PARTIR DE LAS CLASES ENTIDADES.

Este paso tiene como objetivo generar igualmente de forma automática las una clases DAO por cada clase Entidad generada anteriormente. Las Clases DAO en realidad son clases Java común y corrientes, yo las llamo DAO porque tienen métodos que nos permiten realizar operaciones de Acceso a Datos Orientado a Objetos, DAO es un patrón de diseño (algo así como un técnica estándar y optimizada de programación muy en ciertos casos) que separa la lógica de Persistencia (operaciones de Agregar, Buscar, Eliminar, Actualizar y Listar) sobre la base de datos de las clases entidades que modelan dicha base de datos , en este sentido, estas clases DAO deben poseer métodos para poder agregar , eliminar, buscar, actualizar y listar objetos de tipo Entidades sobre la correspondiente tabla de la base de datos. Veamos cómo hacemos esto usando la potencia del IDE Netbeans:

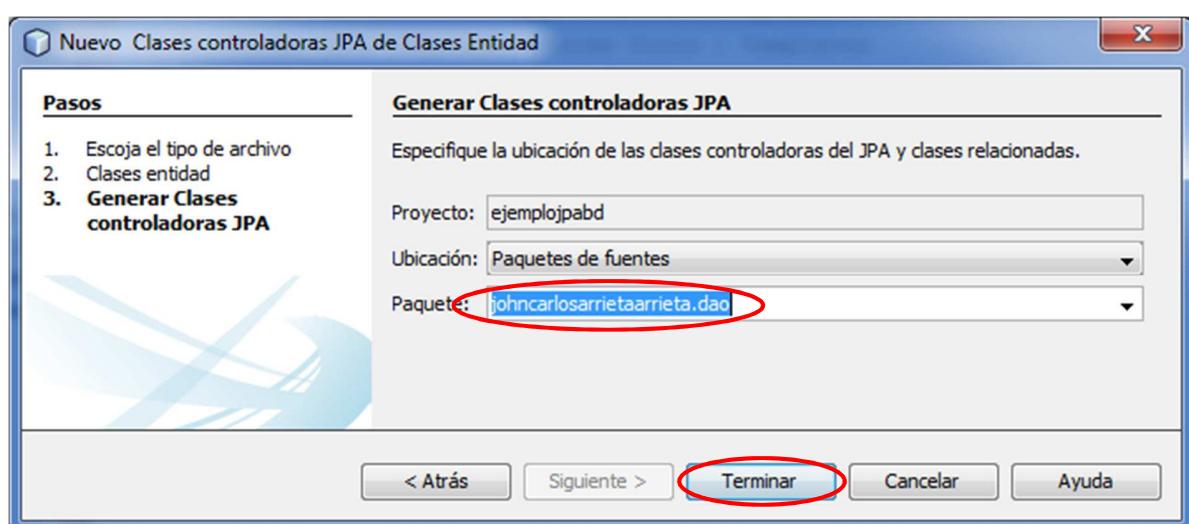
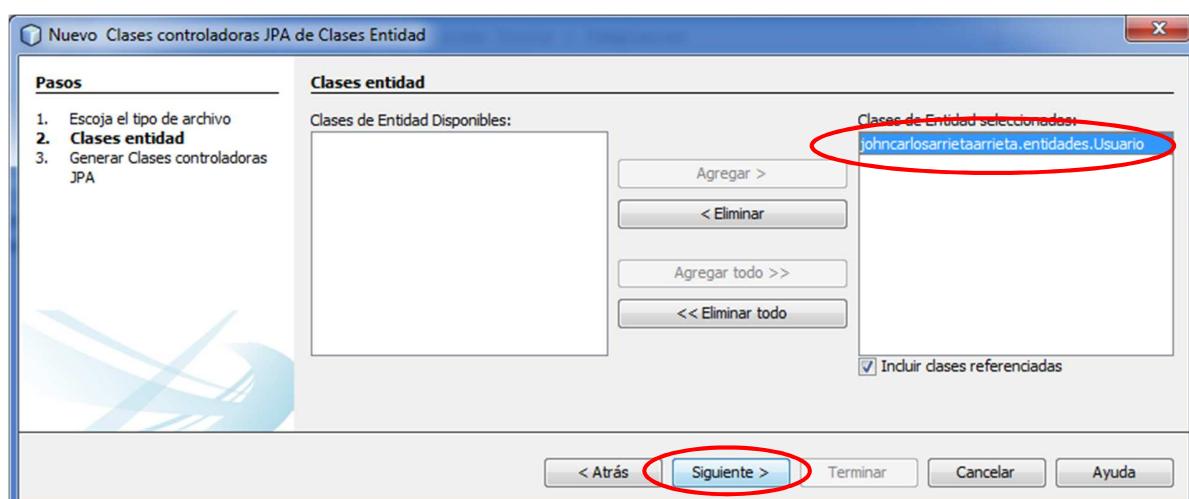
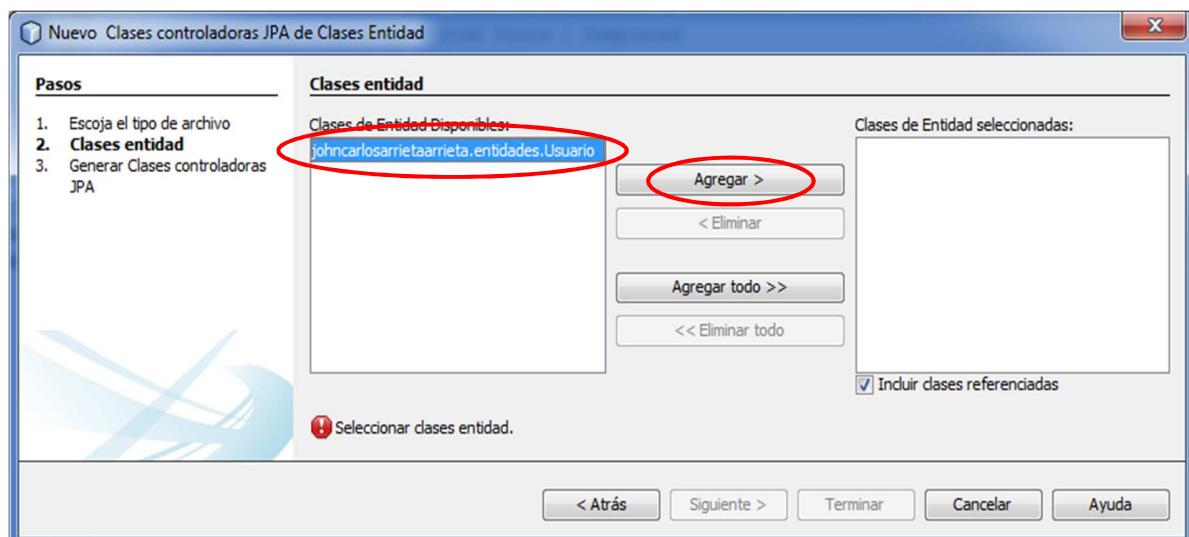
Click derecho sobre el paquete johnacarlosarrietaarrieta.dao / nuevo / otros y aparece la siguiente ventana.



Luego debemos escoger las clases entidades de las que queremos generar las clases DAO.

Seguidamente indicamos que se van a guardar en el paquete johnacarlosarrietaarrieta.dao

Por ultimo pulsamos click en el botón terminar y esperamos un par de segundos dependiendo del número de tablas que se encuentran en la BD y su estructura.





The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer view, displaying a project named 'ejemplojpabd'. Inside the project, there are several packages: 'Paquetes de fuentes' containing 'META-INF', 'johncarlosarrietaarrieta.dao' (which contains 'UsuarioJpaController.java' circled in red), 'johncarlosarrietaarrieta.dao.exceptions' (containing 'IllegalOrphanException.java', 'NonexistentEntityException.java', and 'PreexistingEntityException.java'), and 'johncarlosarrietaarrieta.entidades' (containing 'Usuario.java'); 'Bibliotecas' containing MySQL JDBC driver and EclipseLink JPA 2.0 libraries; and 'JDK 1.7 (predeterminado)'. The right side of the interface shows the code editor for 'UsuarioJpaController.java'. The tab bar at the top of the editor has 'Source' (circled in red) and 'History'. The code itself is a Java class definition:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package johncarlosarrietaarrieta.dao;

import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import johncarlosarrietaarrieta.dao.exceptions.NonexistentEntityException;
import johncarlosarrietaarrieta.entidades.Usuario;

/**
 *
 * @author JohnArrieta
 */
public class UsuarioJpaController implements Serializable {

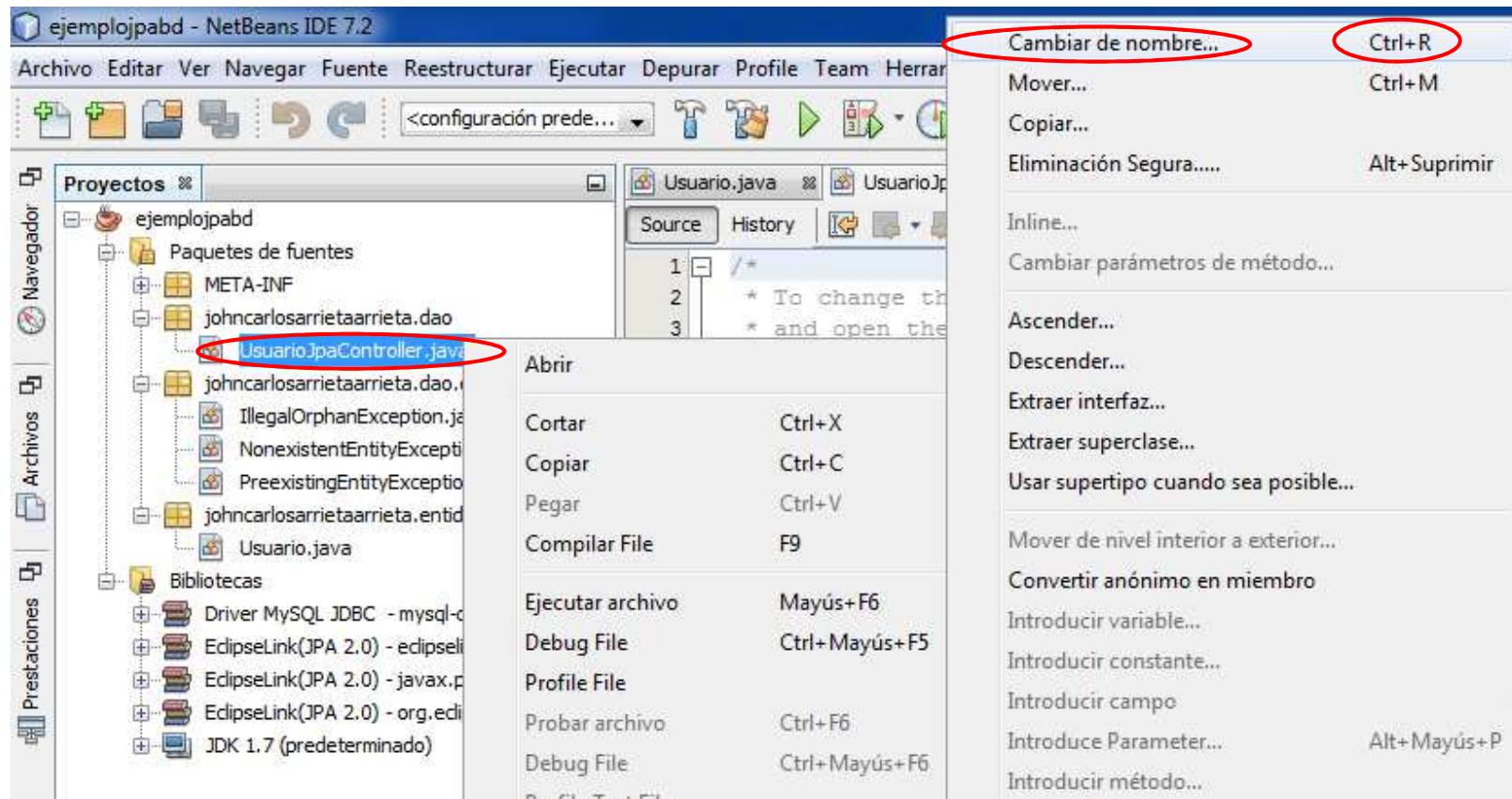
    public UsuarioJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

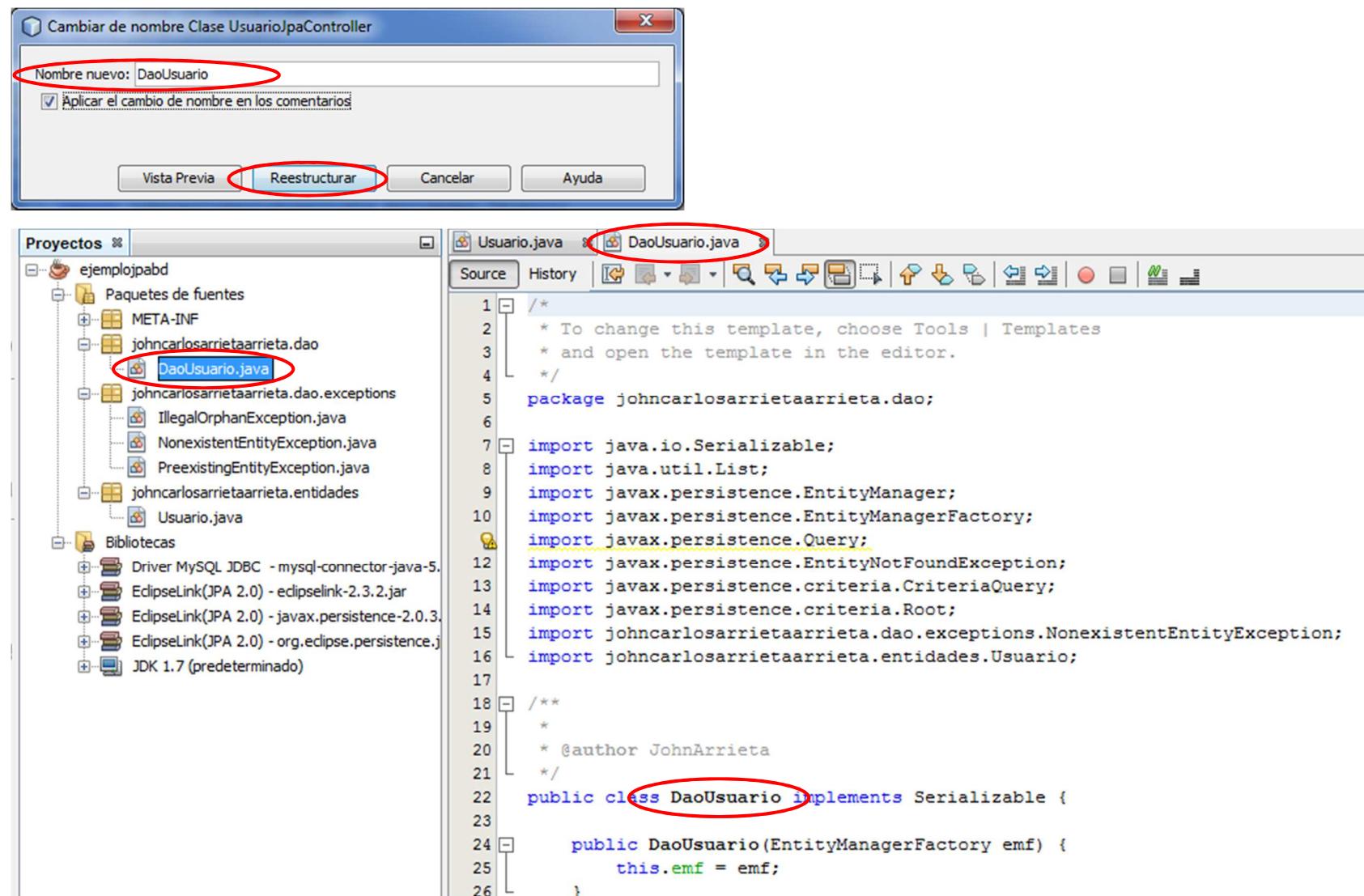
Observar el árbol del proyecto, el generador de código también creo otras clases utilitarias y las coloco en un subpaquete del paquete `johncarlosarrietaarrieta.dao.exception` igualmente generado de forma automática, estas clases se utilizan para controlar y obtener información sobre los errores en tiempo de ejecución que puedan ocurrir mientras se ejecuta y utiliza la aplicación, a estos errores se les conoce como Exceptions o Excepciones. Una de ellas es para saber cuando buscamos una entidad que no existe como registro en su respectiva tabla de la BD, la otra es para saber cuando insertamos una entidad que ya existe como registro dentro su respectiva tabla de la BD.



## Paso 7: (OPCIONAL) PASAR A ESPAÑOL LOS NOMBRES DE LAS CLASES DAO Y SUS METODOS.



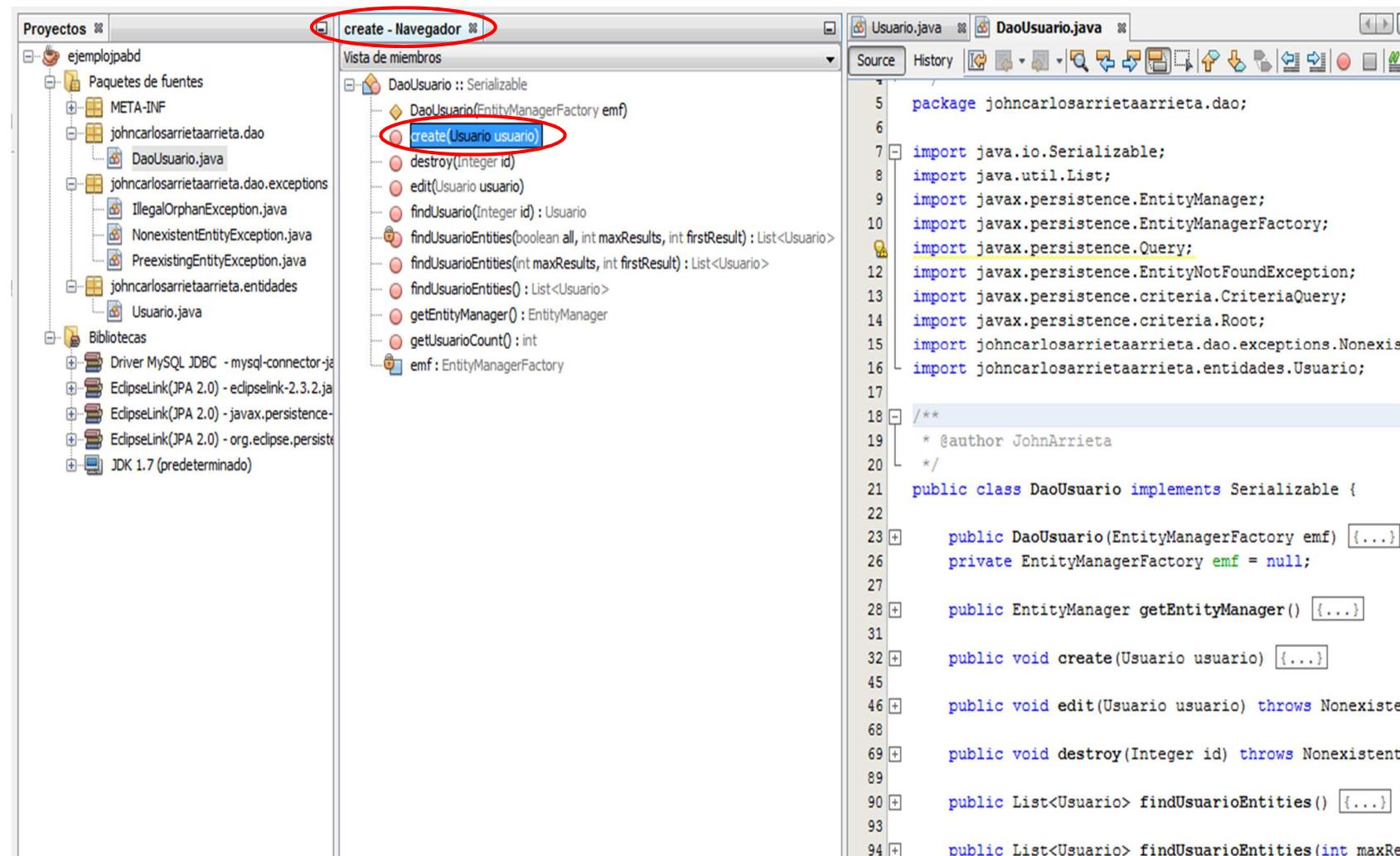
Iniciamos cambiando el nombre de la clase, el cual pasara de ser UsuarioJpaController a DAOUsuario, para ello hacemos click sobre el archivo.jar de dicha clase, luego escogemos la opciones reestructurar / Cambiar Nombre o simplemente al abreviación Ctrl+R . Esto nos permite cambiar el nombre de forma segura, ya que el IDE Netbeans recorre todos el código fuente del proyecto buscando aquellas líneas de código donde utiliza o invoca el antiguo nombre de la clase y lo cambia por el nuevo nombre de forma automática, solo imagínense tener que hacer esto manualmente línea por línea de código dentro de todos los archivos de nuestro proyecto. El proceso culmina cuando nos aparece una ventanita que nos solicita ingresar el nuevo nombre de la clase, lo ingresamos y damos clic en Reestructurar, tal y como se aprecia en la siguiente imagen.



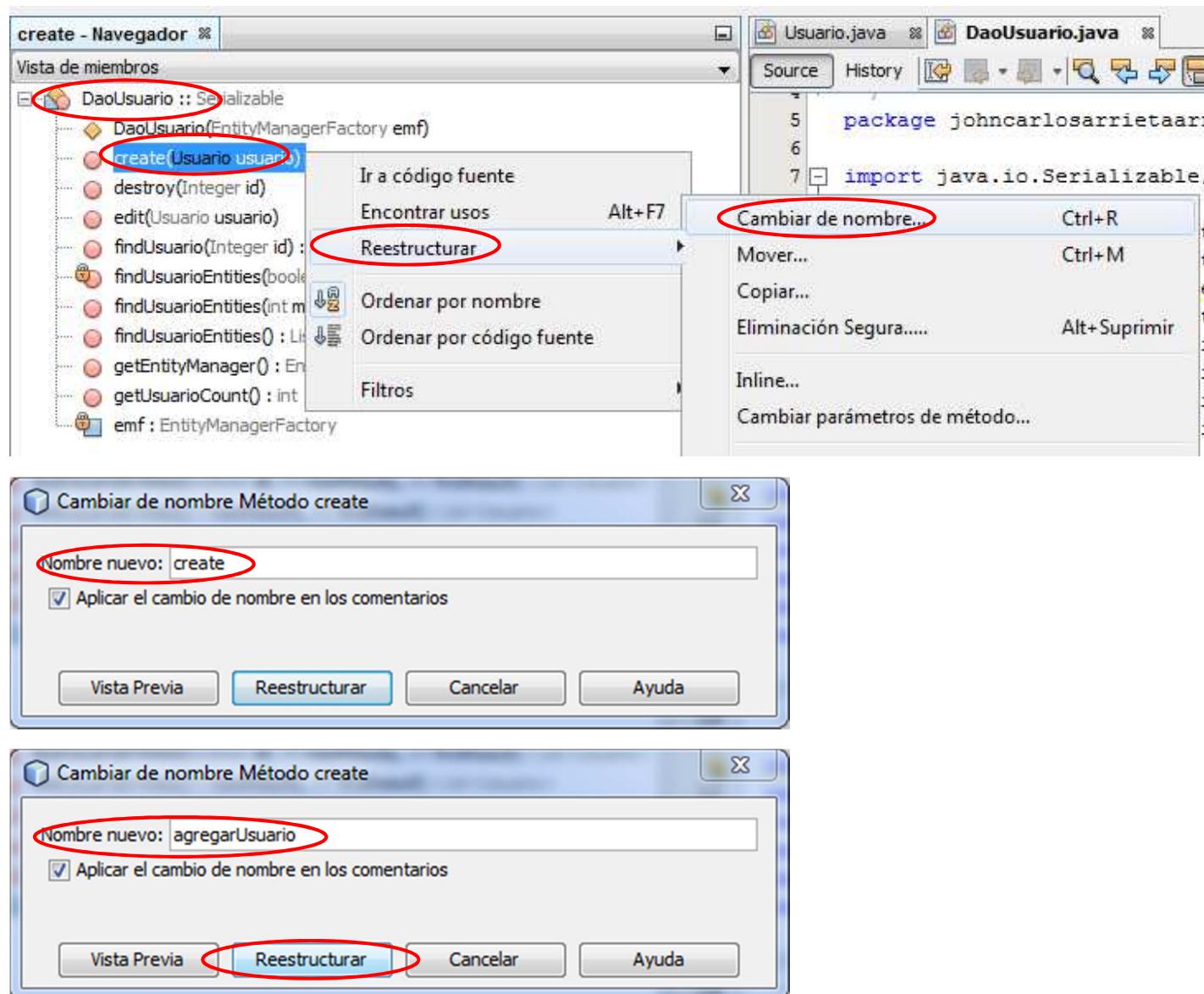
En la ventana anterior podemos verificar que el nombre de la clase fue cambiado en todos los lugares donde puede ser utilizado.



Ahora realizaremos el mismo procedimiento pero esta vez será para cambiar el nombre de los métodos de la clase DAOUsuarios, recuerden que este proceso es opcional, pues si lo desean pueden utilizar el nombre de la clase sus métodos en inglés como se generaron originalmente.

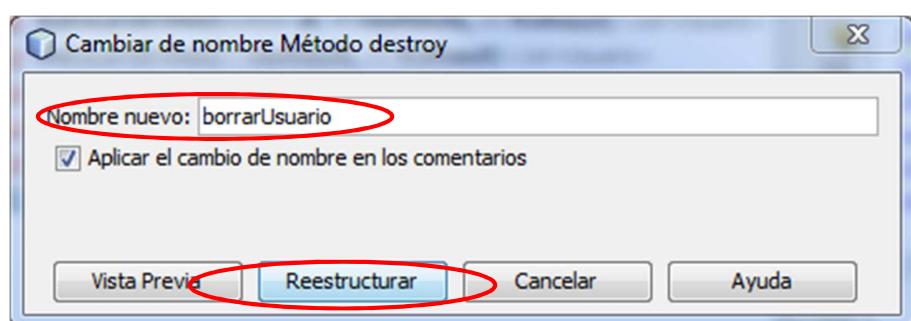
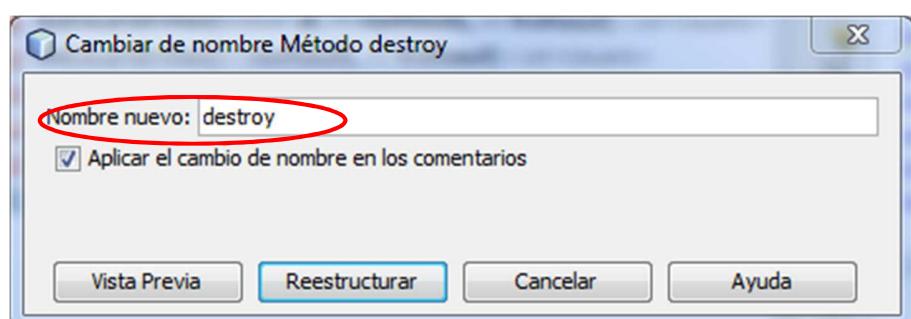
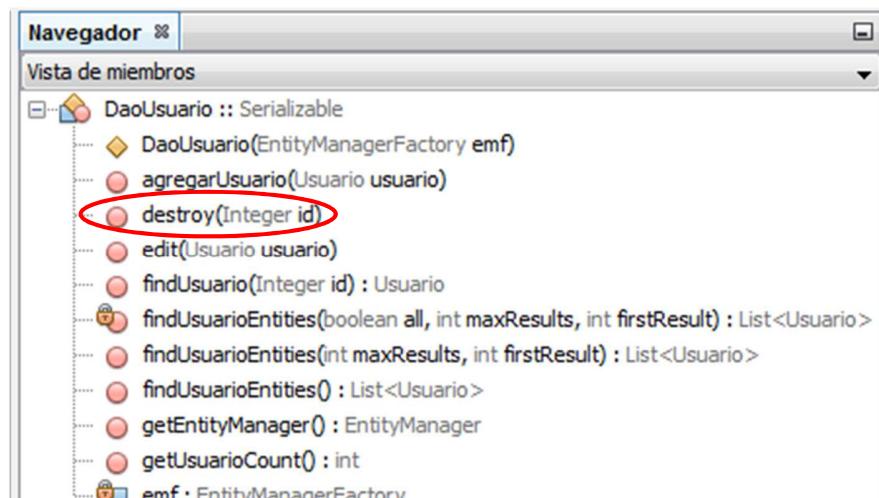


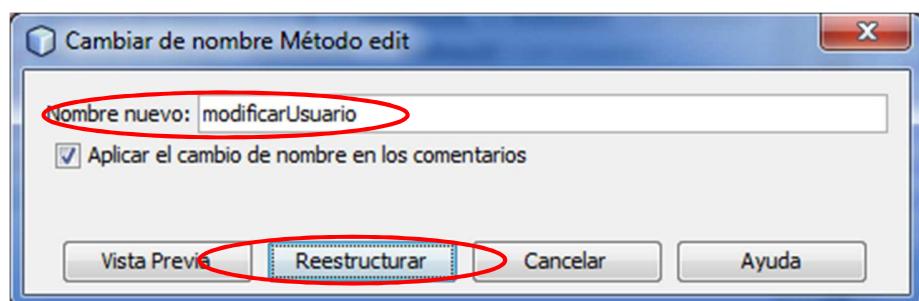
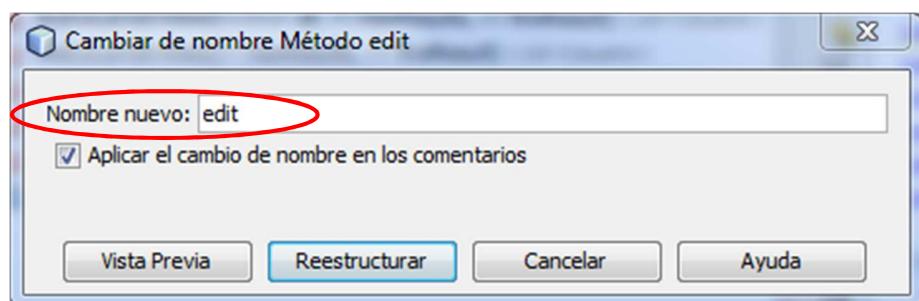
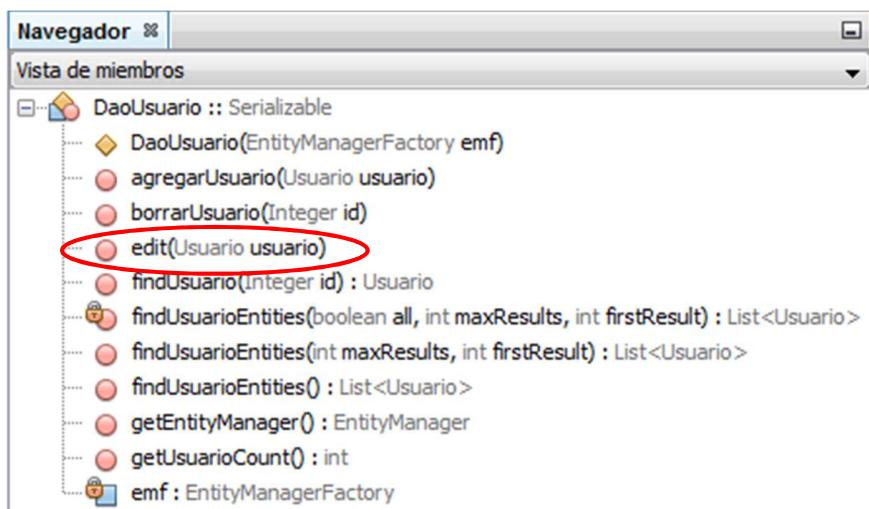
Observar que el proceso lo realizamos en el Panel Navegador y no en el Panel de Proyectos, ya que el Panel Navegar podemos observar o navegar por todas los elementos (Propiedades y metodos) que conforman la clase sobre la cual estamos trabajando desde el panel de Proyectos.

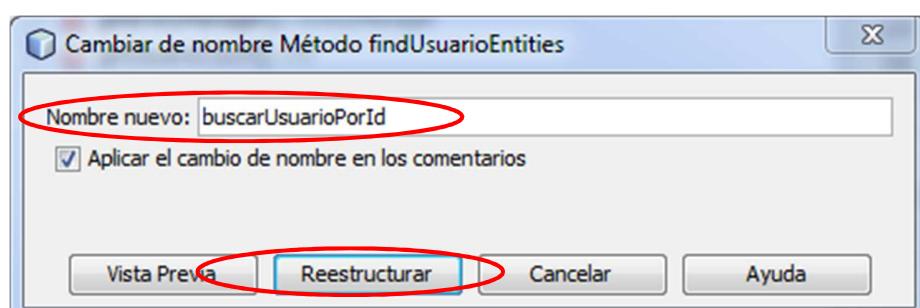
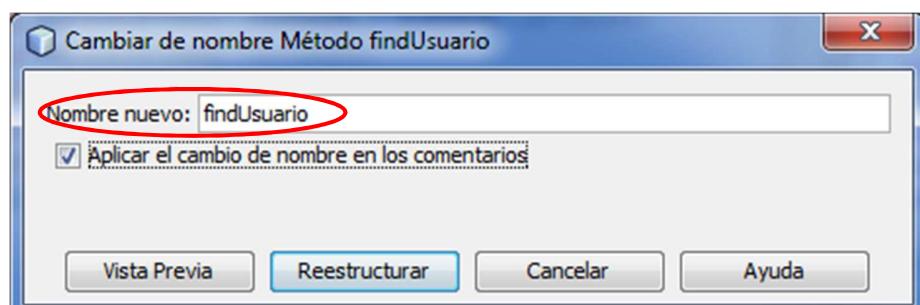
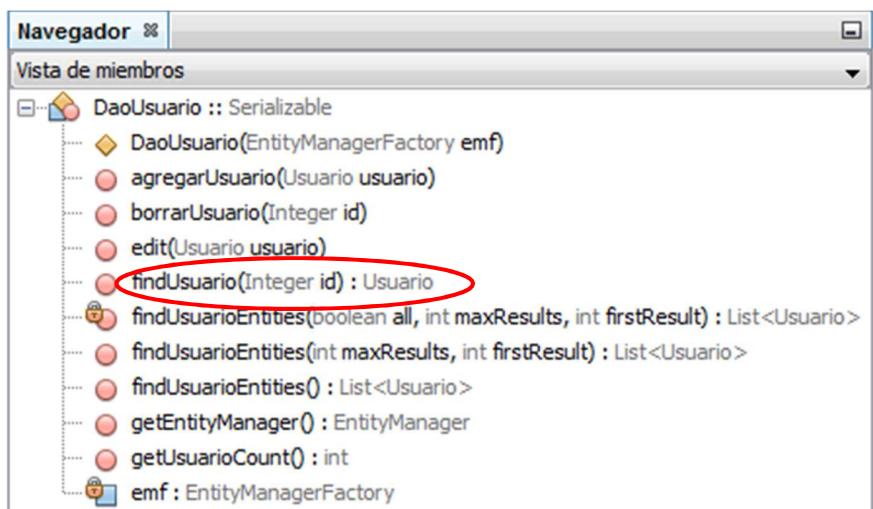


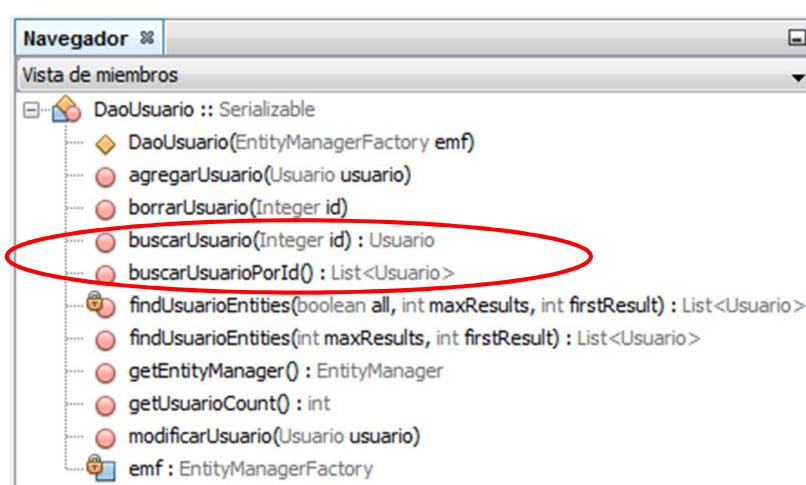
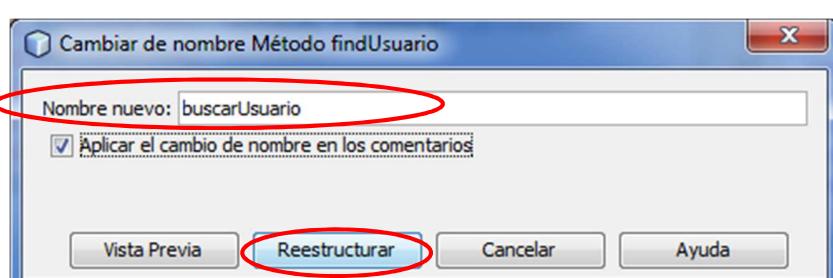
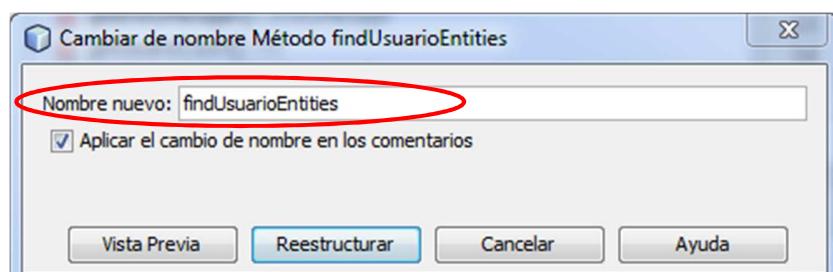
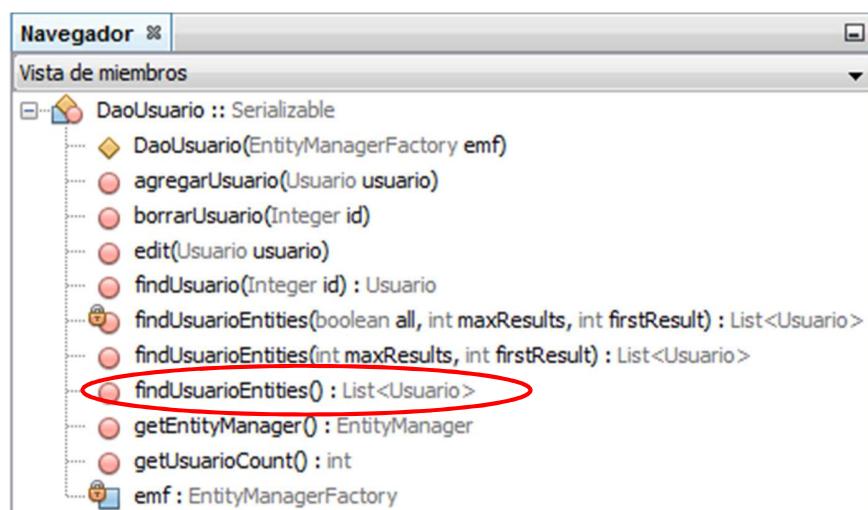


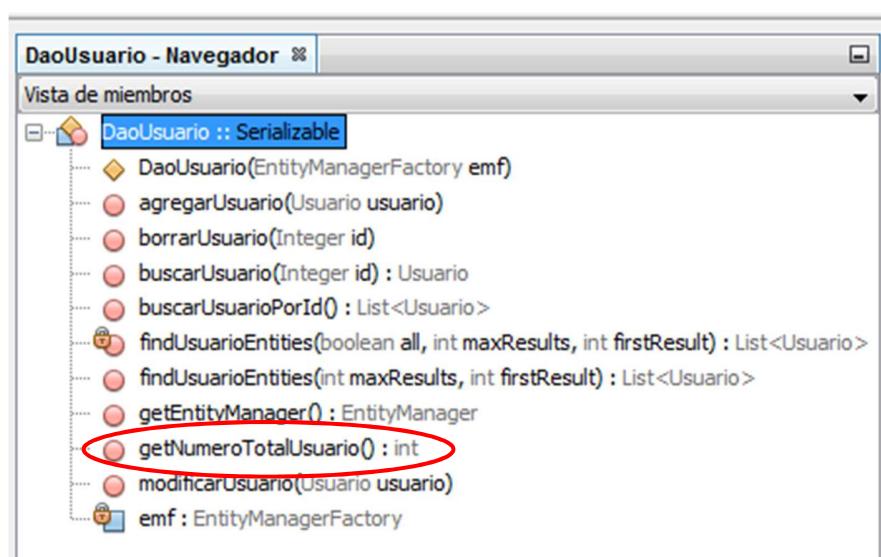
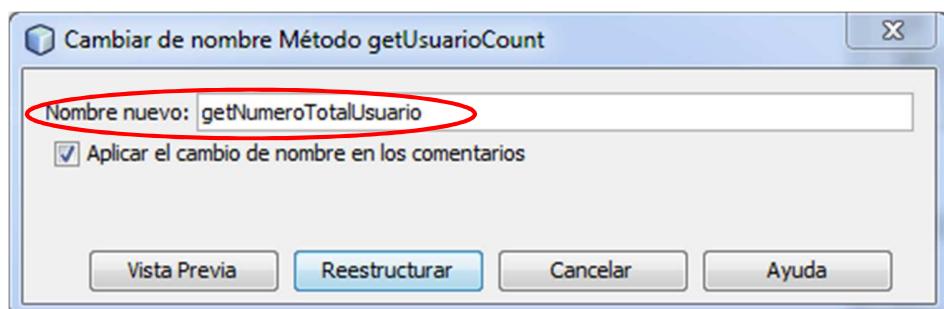
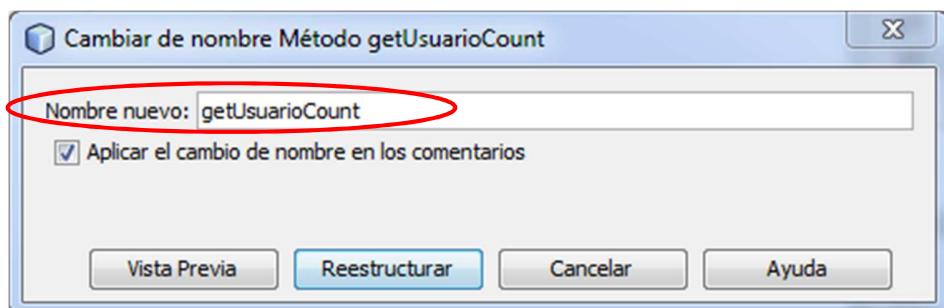
La siguiente secuencia de imágenes muestra el procedimiento repetido para los otros métodos











De esta manera deberíamos tener el nombre de la clase y sus métodos más legibles para los que no saben muchas inglés.

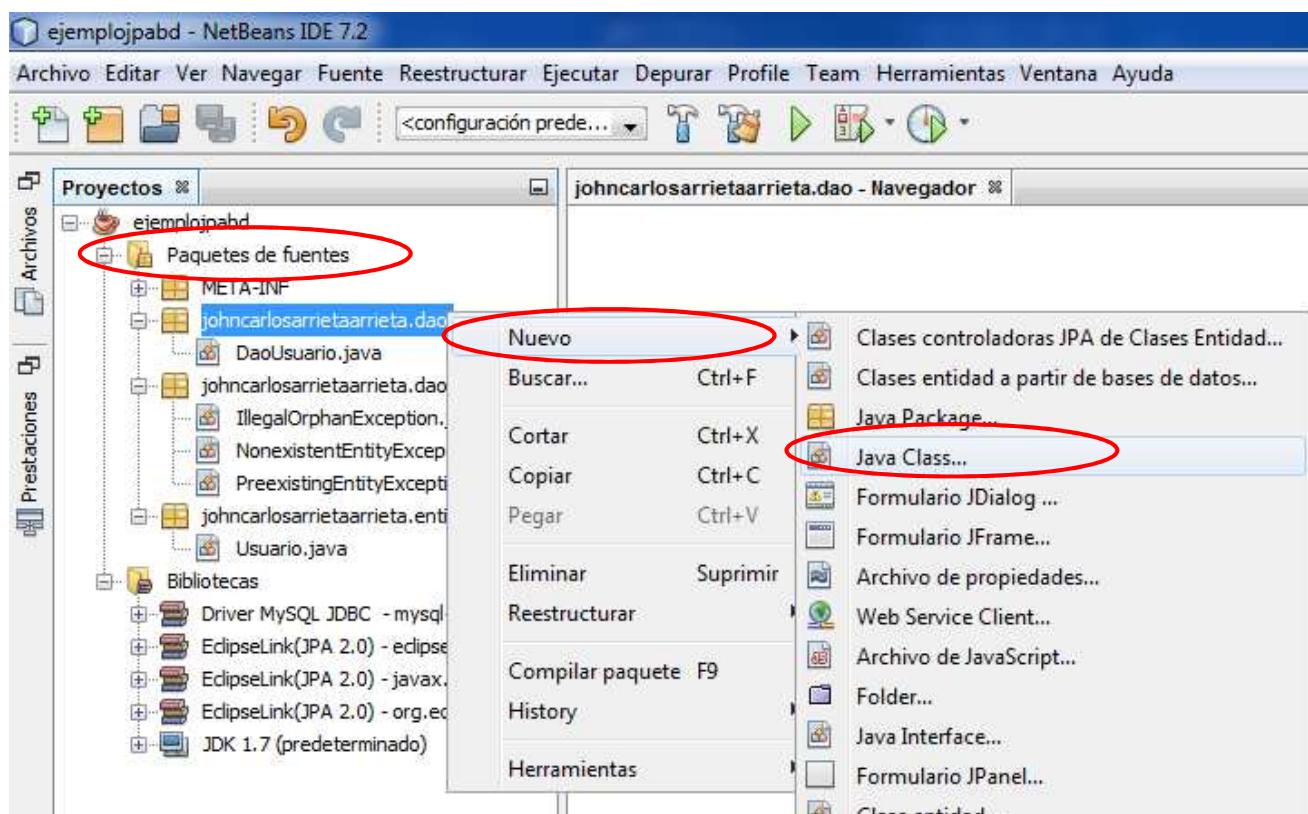


## Paso 8: CREAR LA CLASE QUE PERMITA FABRICAR LOS DIFERENTES OBJETOS O INSTANCIAS DE LAS CLASES DAO QUE TENGA NUESTRO PROYECTO.

En esta paso procedemos a escribir un poco de código Java para diseñar una clase que nos permita crear los objetos o instancias de las clases DAO que hayamos creado en nuestro proyecto. En este ejemplo solo tenemos una sola clase Dao (DAOUsuario) porque solo tenemos una sola clase Entidad (Usuario), algo que seguramente no va a ocurrir en otros proyectos no tan simple como este, en general la mayoría de los proyectos de software que almacenan sus datos en sistemas de bases de datos, utilizan muchas tablas relacionadas entre sí, por lo que deberíamos tener una equivalencia numérica entre la cantidad de clases Entidades vs la cantidad de tablas en la BD, en algunos casos cuando la relación entre dos tablas es de muchos a muchos, el generador de código de Netbeans crea una clase Entidad adicional cuyo nombre es la unión del nombre de las dos clases terminando con las letras PK, por ejemplo, de la siguiente figura se generaran mínimo 3 clases: Usuario, Eventos e InvitacionesPK

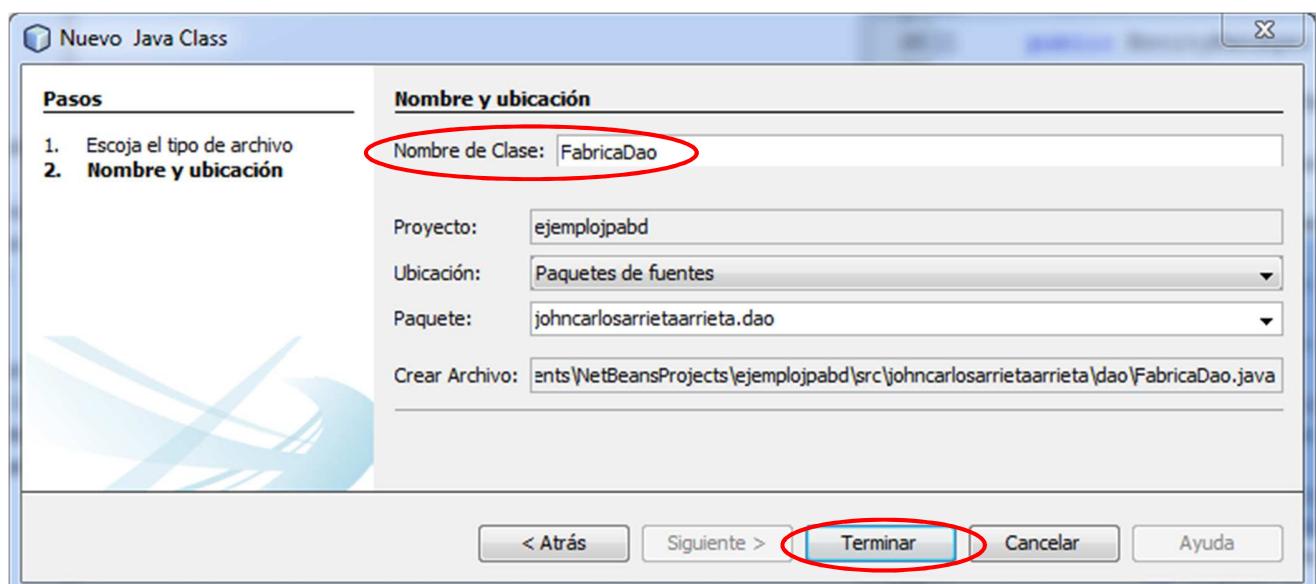
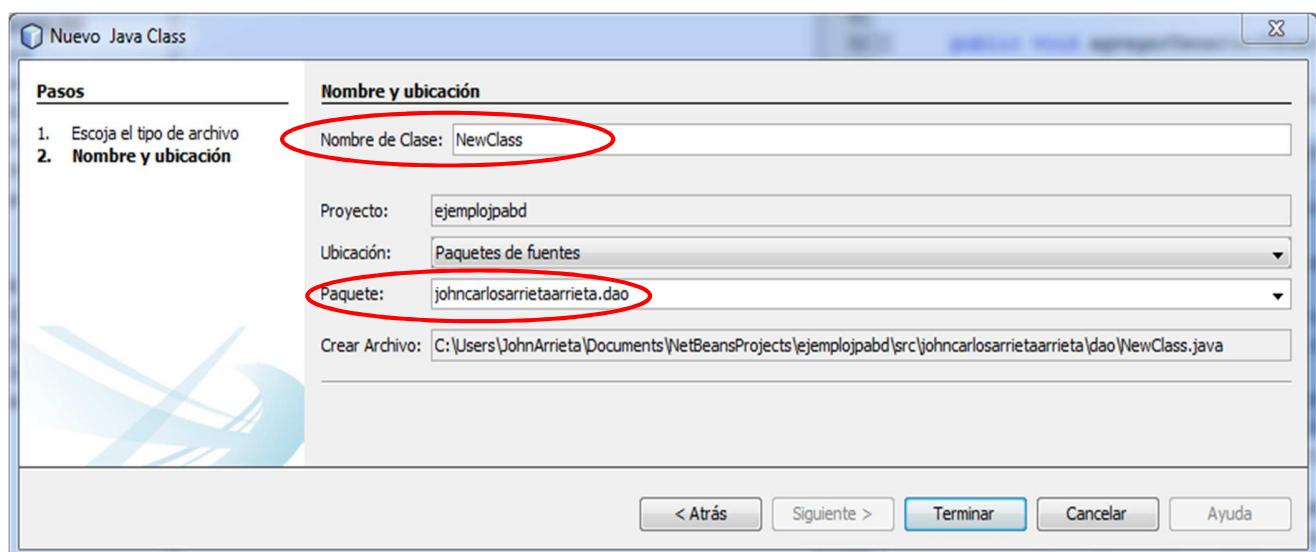


El procedimiento para crear una nueva clase Java desde Netbeans es simple, solo seguir los siguientes pasos:





Nos aparece una ventana en la cual debemos ingresar el nombre de la clase, recuerden que para el nombre de las clases se recomienda que cada una de las palabras que lo conforman estén pegadas sin \_ guion de piso y que cada palabra inicie con la primera letra en mayúscula y el resto de letras de cada palabra debe estar en minúscula, también es super importante recordar que existe una regla de sintaxis Java para los nombre de las clases, métodos y variables, la cual especifica que no deben iniciar con número, no deben tener los siguientes caracteres ¨!”. %&/()=?¿º@|#’¡+\*]`^[\ç]{‘-.:>,<;



Al pulsar click en el botón aceptar deberíamos tener un archivo llamado FabricaDao.java en el paquete dao.



En esta ventana podemos observar el código de la clase, al cual se ha agregado algunas líneas adicionales que son las que nos permitirán crear instancias u objetos de la clase DAOUsuario, líneas que explicare a continuación

```
5 package johncarlosarrietaarrieta.dao;
6
7 import javax.persistence.EntityManagerFactory;
8 import javax.persistence.Persistence;
9
10 /**
11 * @author JohnArrieta
12 */
13 public class FabricaDao {
14     private static FabricaDao instancia;
15     private static EntityManagerFactory emf;
16
17     // constructor privado
18
19     private FabricaDao() {
20
21     }
22
23     // metodo singleton
24     private static FabricaDao getInstancia() {
25         if (instancia == null) {
26             instancia = new FabricaDao();
27         }
28         emf = Persistence.createEntityManagerFactory("ejemplojpabdPU");
29         return instancia;
30     }
31
32     public static DaoUsuario getDaoUsuario() {
33         return new DaoUsuario(getInstancia().emf);
34     }
35
36     // aqui se colocan los metodos para los otros DAO de la BD
```

**Línea 5:** instrucción que indica que la clase se debe guardar en el paquete johncarlosarrietaarrieta.dao.

**Línea 7 y 8:** Instrucciones que indican que dentro del código de esta clase vamos a requerir y utilizar algunos elementos (propiedades y métodos) que hacen parte de las clases EntityManagerFactory y Persistence, ambas clases incluidas en el paquete javax.persistence. el cual viene incluido por defecto entre los archivos de instalación del JDK.

**Línea 13:** Instrucción que define la cabecera o firma de la clase y su inicio, esto indica su ámbito (público en este caso y la mayoría), en nombre de la clase y la { llave de inicio del cuerpo de la clase.

**Línea 14 y 15:** Declaramos dos propiedades de la clase, ambas con ámbito prohibido y estáticas (su valor permanece sin importar si se crean o instancias u objetos de esta clase, una de las propiedades se llama instancia



cuyo tipo es del mismo tipo de la clase que estamos escribiendo, la otra se llama emf de tipo EntityManagerFactory.

**Línea 19 a 21:** Indica la definición del método constructor por defecto. Un constructor es un método que se utiliza para crear instancias u objetos de su clase, tienen como particularidad que se llaman igual que su propia clase y no se les debe indicar que tipo de datos retorna o devuelven al momento de ser invocados.

**Línea 24 a la 30:** Define un método llamando getInstancia, el cual retorna como valor una instancia de la clase que lo contiene, es decir de la clase FabricaDao, este método es de ámbito privado (solo puede ser utilizado e invocado o llamado dentro del cuerpo de su propia clase), es estático (para invocarlos solo es necesario anteponerle el nombre de sus clase, es decir FabricaDao y el operador punto, por ejemplo FabricaDao.getInstalacion(); ), los () paréntesis vacíos indican que no recibe parámetros como argumentos de entrada, la { llave de apertura indica el inicio del cuerpo del método , mientras que la llave de } en la línea 30 indica que este es el fin del cuerpo del método.

**Línea 25 a 27:** representan una estructura de control de tipo condicional SI o IF en inglés, esta estructura valida la expresión que tiene dentro de sus () paréntesis, si el resultado de esta expresión es verdadero o TRUE en inglés, entonces se realizan las instrucciones que están dentro del cuerpo { llave de inicio y } llave de fin del método. La instrucción condicional dentro de los paréntesis básicamente compara si el valor de la propiedad instancia es igual null, es decir a nada, entonces devuelve TRUE en este caso, sino devuelve FALSE o falso en caso contrario, si es el primer caso entonces dentro del cuerpo del IF invocamos al método constructor de esta clase y la instancia u objeto devuelto lo asignamos como valor de la propiedad instancia, de esta forma ya no será igual a null.

**Línea 28:** Invocamos al método createEntityManagerFactory("ejemplojpabdPU") de la clase Persistence, este método recibe como parámetro o argumento un texto, cadena o String que contiene el nombre de la unidad de persistencia, la cual se generó automáticamente en los pasos anteriores, esta unidad de persistencia representa la base de datos a la cual nos vamos a conectar, este valor por defecto es igual al nombre de nuestro proyecto (ósea ejemplojpabd) terminando con las letras PU, es decir ejemplojpabdPU. El valor retornado por la llamada de este método lo asignamos a la propiedad emf, ya que este método retorna una instancia u objeto de la clase EntityManagerFactory.

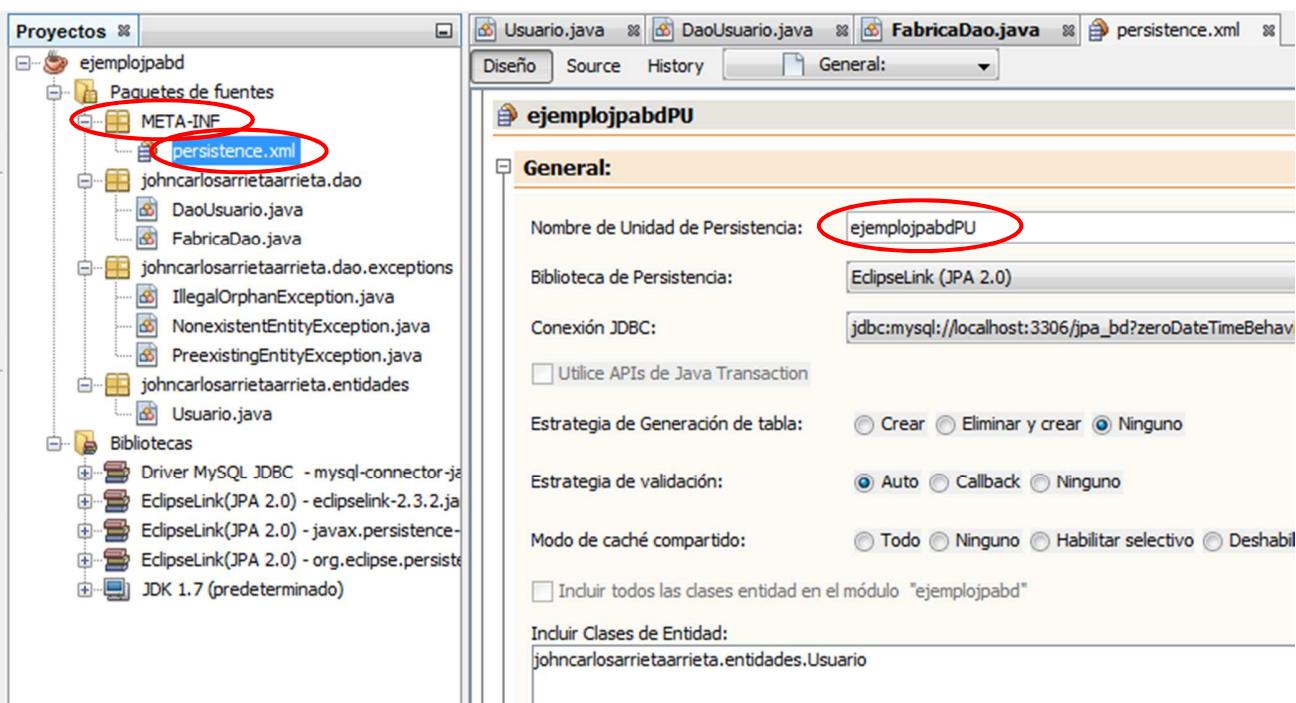
**Línea 29:** aquí retornamos la instancia de tipo EntityManagerFactory almacenada en la propiedad emf.

**Línea 32 a 34:** En este bloque de código definimos el método getDaoUsuario, el cual como ya podemos intuir es de ámbito público, es estático, no recibe parámetro o argumento alguno y retorna una instancia de tipo DAOUsuario.

**Línea 33:** En esta instrucción creamos una instancia de tipo DAOUsuario invocando el constructor de dicha clase utilizando la palabra de Java new y le pasamos como parámetro al constructor la instancia de tipo EntityManagerFactory que tenemos guardada en la propiedad emf de esta clase FabricaDao.

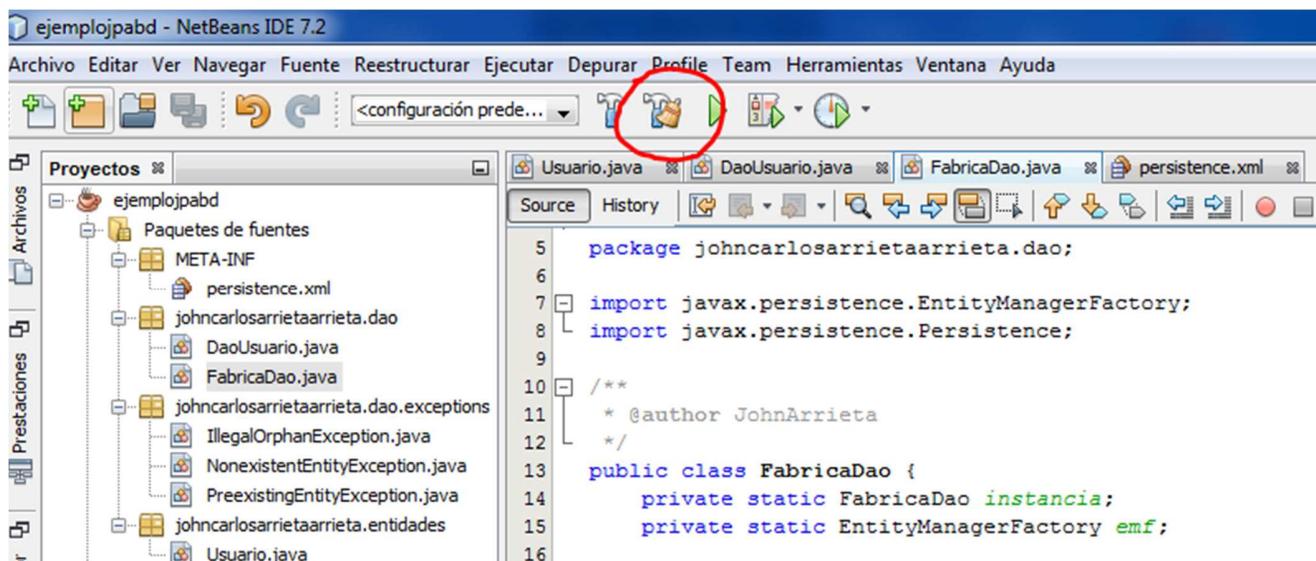
En resumen, esta clase contiene 2 propiedades una llamada instancia y otra emf, la primera es del mismo tipo de esta clase y la segunda de tipo EntityManagerFactory, tiene un constructor por defecto privado, un método privado que retorna crea una instancia de la clase en caso de que no esté creada y un método que retorna una instancia de la clase DAOUsuario.

En la siguiente ventana podemos consultar o cambiar el nombre de la unidad de persistencia de nuestro proyecto



## Paso 9: COMPILAR Y CONSTRUIR LA LIBRERÍA DEL PROYECTO EN UN ARCHIVO .JAR.

En este paso vamos a decirle al IDE que nos compile todos los archivos de código fuente que conforman nuestro proyecto (archivos de clase .java, archivos xml, etc) y el al finalizar tome todos los archivos compilados (.class y .jar) y los empaquete dentro un único archivo .jar, este archivo será nuestra biblioteca o libreraia de código reutilizable.





## Paso 10: CONSTRUIR EL SERVICIO WEB

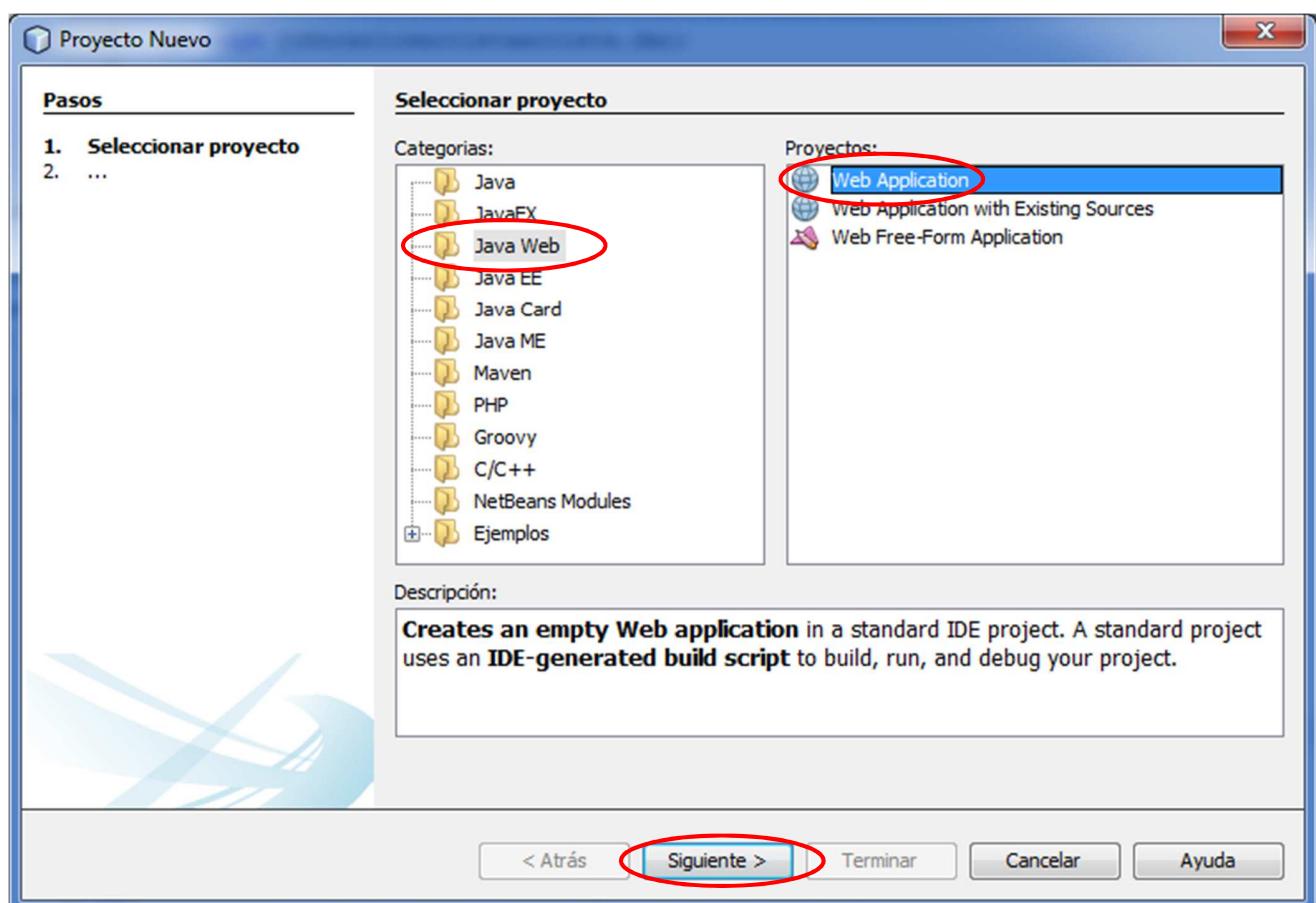
En este paso vamos utilizar el IDE Netbeans para construir un servicio web que al ser consumidos desde cualquiera PC o Dispositivo móvil este podrá agregar un Usuario a la base de datos. Las aplicaciones que pueden conectarse a un servicio web, enviando o recibiendo datos hacia o desde un servicio web se les conoce como Clientes Web, la particularidad de estos es que pueden ser escritos en cualquier lenguaje de programación que soporte la tecnología Webservices, y sin siquiera comprender o saber cuál es su estructura, simplemente se conectan a su dirección web y proceden a consumir sus operaciones remotamente.

Para construir el servicio web en escrito en Java es necesario utilizar las siguientes herramientas:

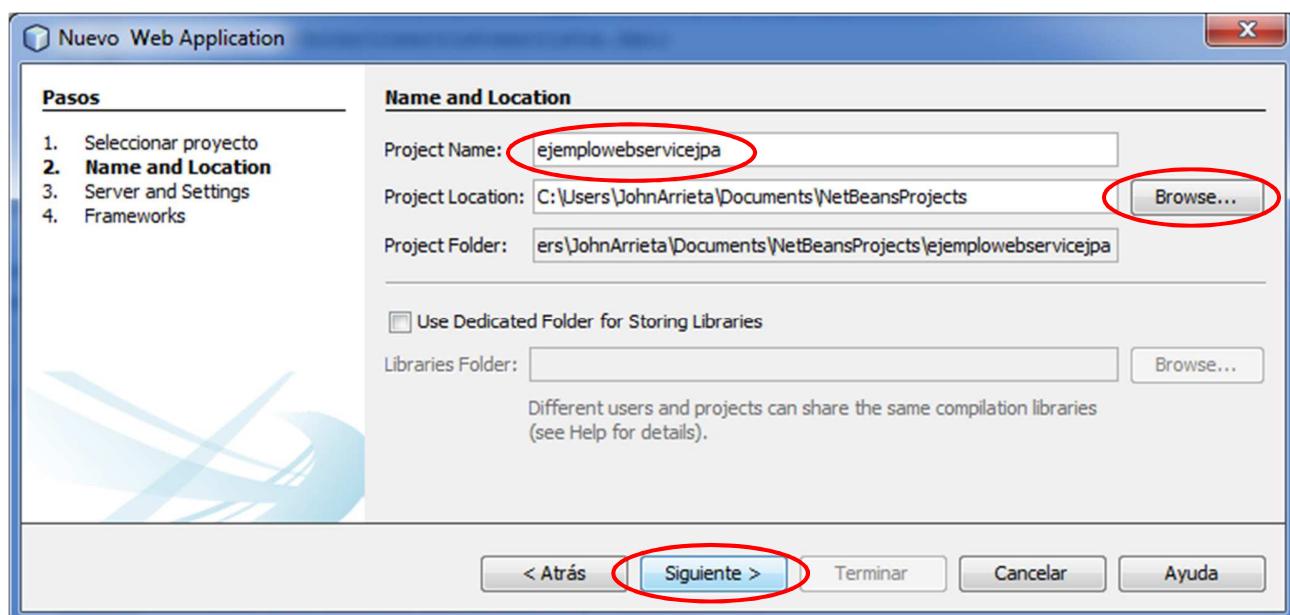
**Un servidor Web:** Apache http Web Server

**Un contenedor de JSP con soporte Webservices:** Apache Catalina Tomcat Aplicación Server.

Para iniciar debemos crear un nuevo proyecto en Nebeans, en la ventana de Nuevo proyecto seleccionamos la categoría **Java Web** y en tipos de Proyectos seleccionamos la opción **Web Application**

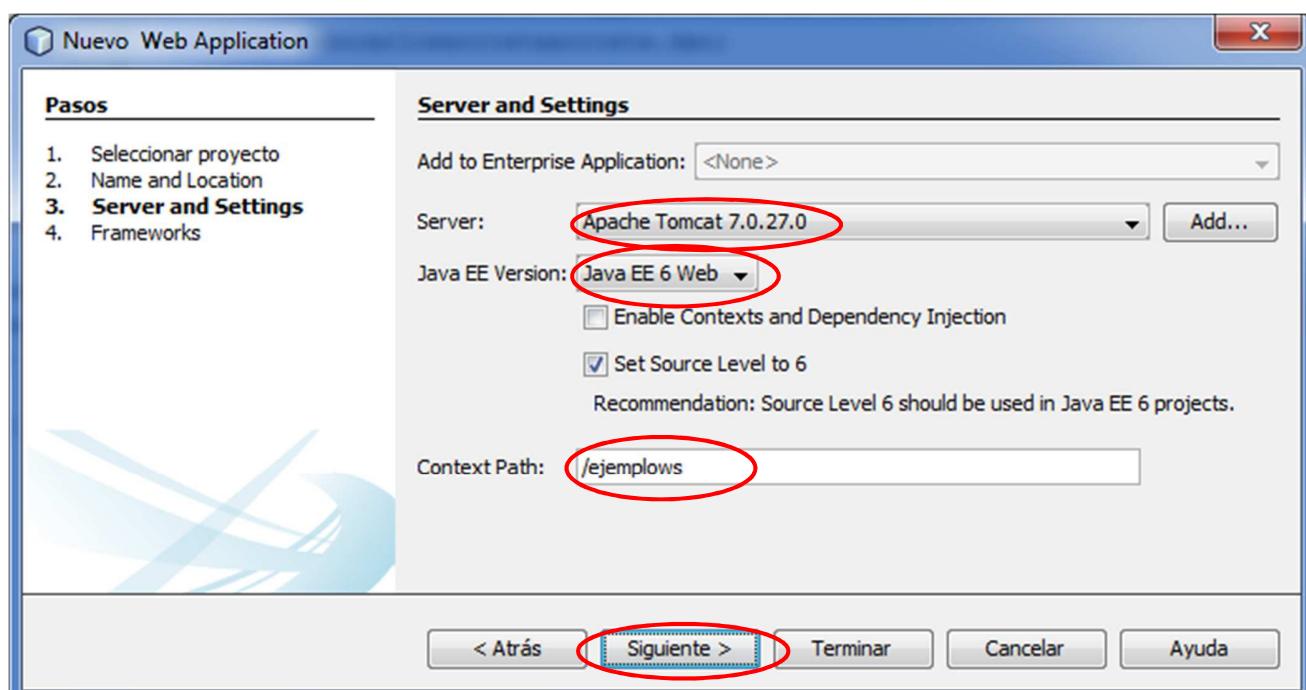


Luego en la siguiente pantalla debemos ingresar el nombre del proyecto We y la ubicación de la carpeta de donde se almacenara todos los elementos del proyecto.



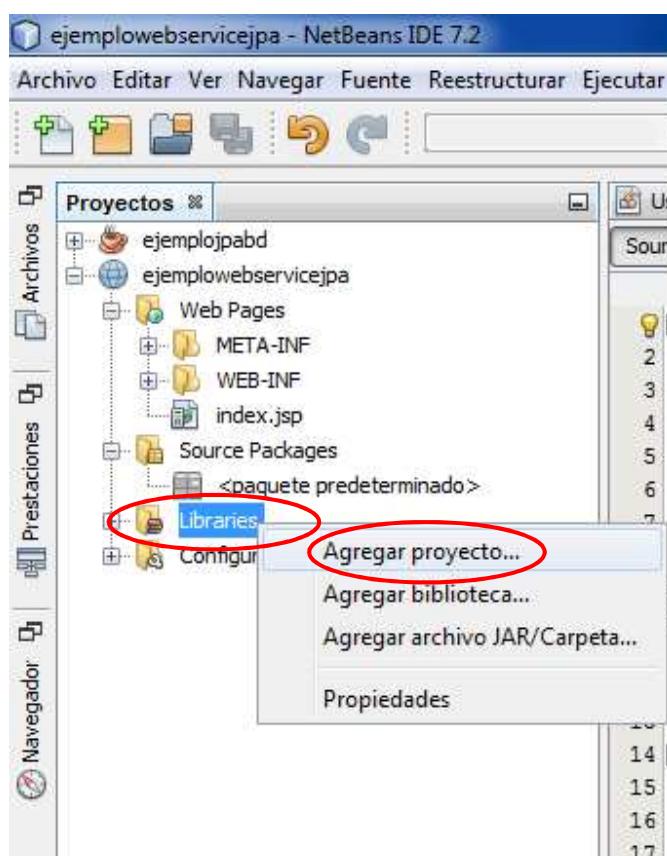
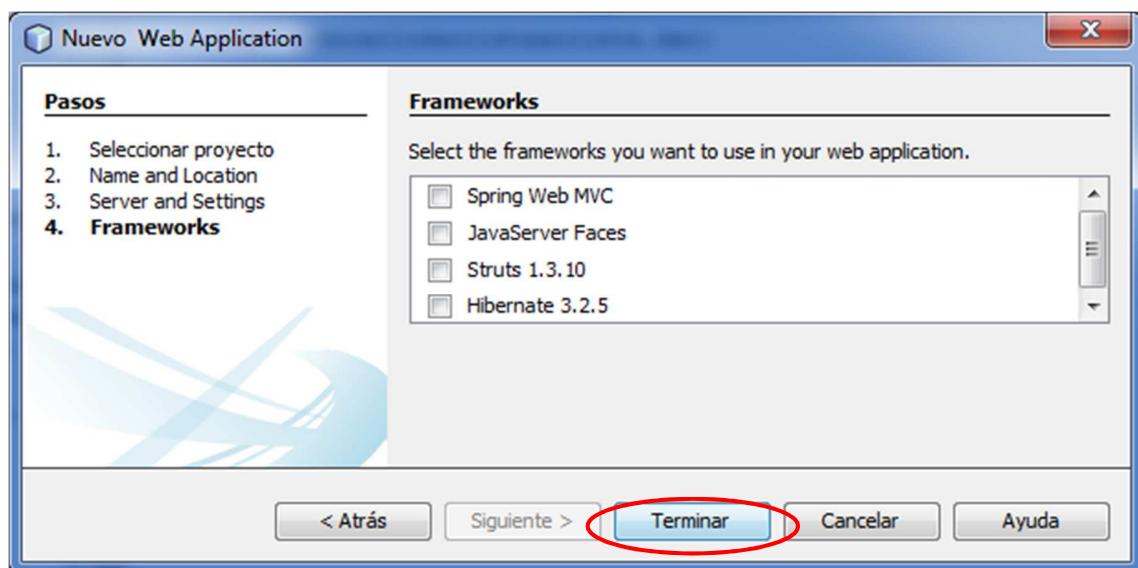
Luego se debe escoger el servidor de aplicaciones y contenedor de Servlet/JSP con soporte a WebServer, aunque existen varios servidores con estas características, en el mercado domina IBM WebSphere, JBoss, Adobe ColdFusion, Apache Tomcat, Oracle GlassFish y Jetty, entre otros menos populares.

Para nuestro proyecto vamos a utilizar Apache Tomcat el cual se puede instalar como complemento adicional al IDE Netbeans, ya que este IDE instala por defecto el servidor de Oracle.





En esta ventana se nos presentan un conjunto de Framework para el desarrollo de aplicaciones Web usando Java como lenguaje de programación del lado del servidor, en nuestro caso no vamos a escoger nada, debido a que nuestra aplicación no será una aplicación Web Propiamente dicha, sino una que ofrezca un servicio de procesamiento el cual será consumido o invocado desde Internet por cualquier aplicación que así lo desee.



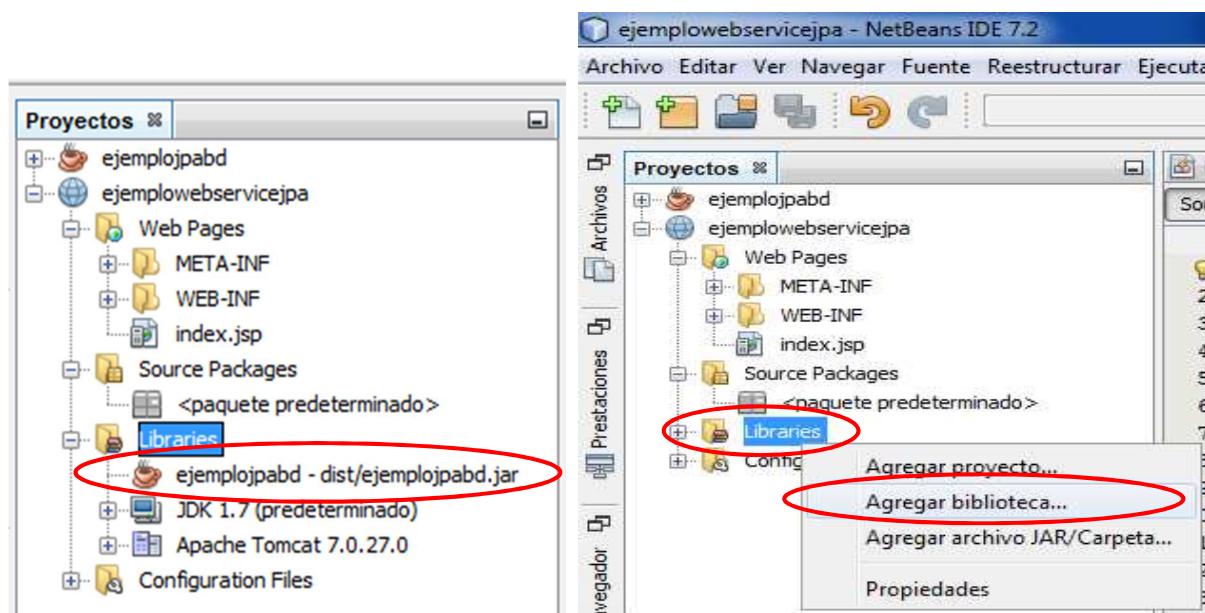
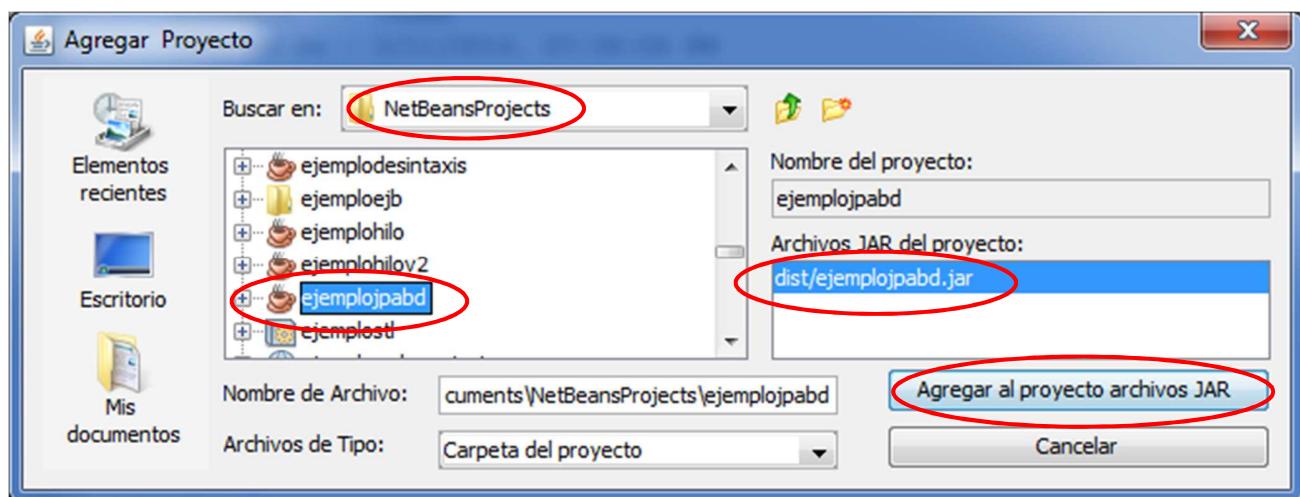
La estructura del proyecto web quedaría como se aprecia en la imagen de la izquierda: Una carpeta raíz con icono de globo terráqueo, una carpeta donde se almacenan las páginas Web HTML, JSP, JS, CSS, XML, Imágenes y multimedia. Una carpeta WEB-INF donde se almacenan archivos de información del proyecto, esta carpeta no puede ser accedida desde un cliente web cualquiera. Un archivo index.php el cual actúa como página de bienvenida al momento de realizar cualquier conexión. Una carpeta Source donde se guardan los paquetes y archivos de código Java, como los Java Beans, los Servlet y clases utilitarias. Una carpeta donde se almacenan las librerías que necesita el proyecto, este es justo el paso siguiente, seleccionar y agregar las librerías necesarias.

En este caso agregaremos la librería que desarrollamos en el proyecto anterior.



## Paso 11: AGREGAR EL PROYECTO LIBRERÍA JPA CONSTRUIDO ANTERIORMENTE

Unas vez escogida la opción Agregar proyecto según la ventana anterior, procedemos a buscar nuestro proyecto ejemplojpabd realizado anteriormente como proyecto de librería, lo seleccionamos y observamos que se agrega el archivo ejemplojpabd.jar

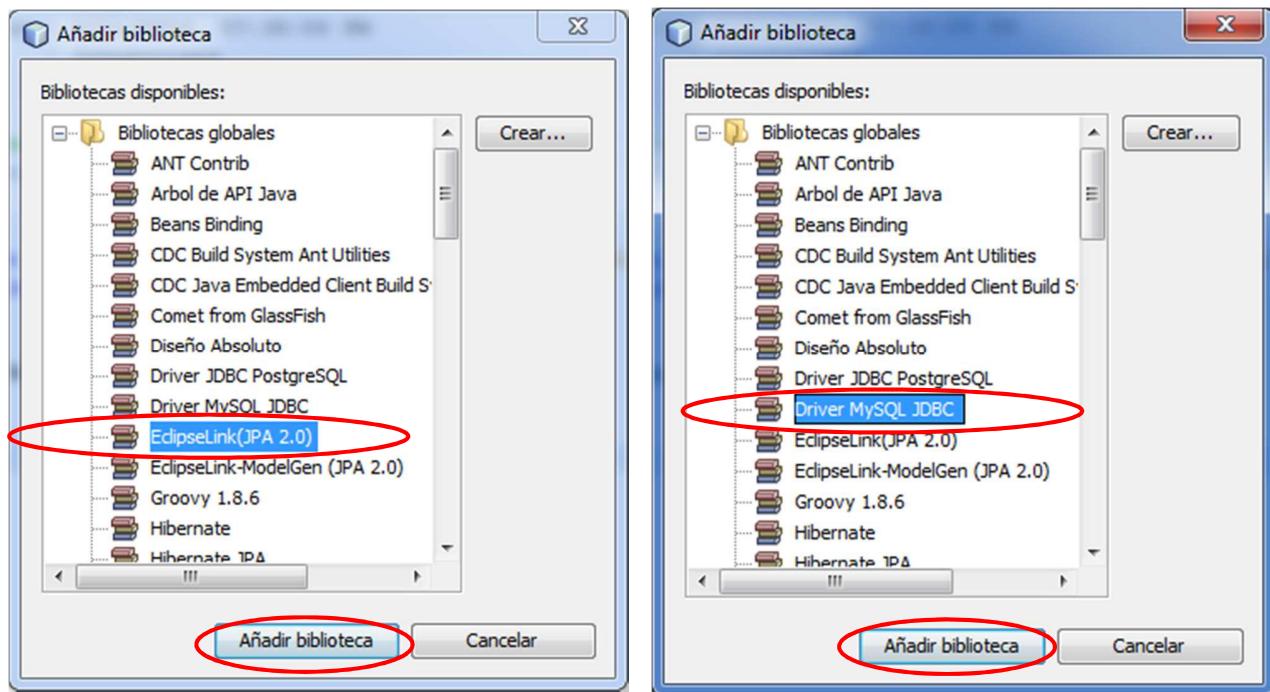


## Paso 12: AGREGAR LAS LIBRERÍAS DE JPA 2.0 PARA QUE FUNCIONE EL PROYECTO

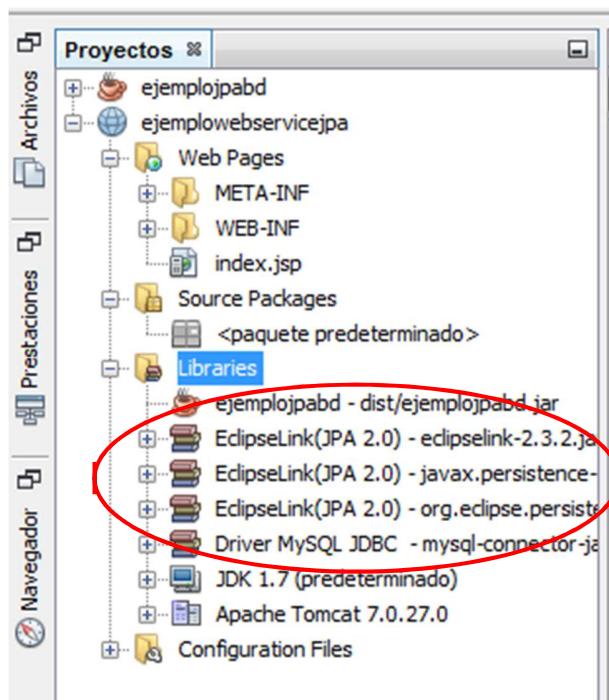
Como el proyecto desarrollado anteriormente utiliza una arquitectura estándar de JavaEE llamada JPA para poder facilitarnos la operaciones de persistencia a la BD, es necesario que en este nuevo proyecto tambien agregemos las librerías de JPA 2.0. Como se aprecia en la imagen anterior a la derecha, en el proceso para agregar las librerías JPA 2.0 se repiten los pasos realizados anteriormente para agregar el proyecto ejemplojpabd.jar



Buscamos y seleccionamos la librería EclipseLink (JPA 2.0), el mismo proceso hacemos para agregar el Driver de conexión al motor de bases de datos MySQL



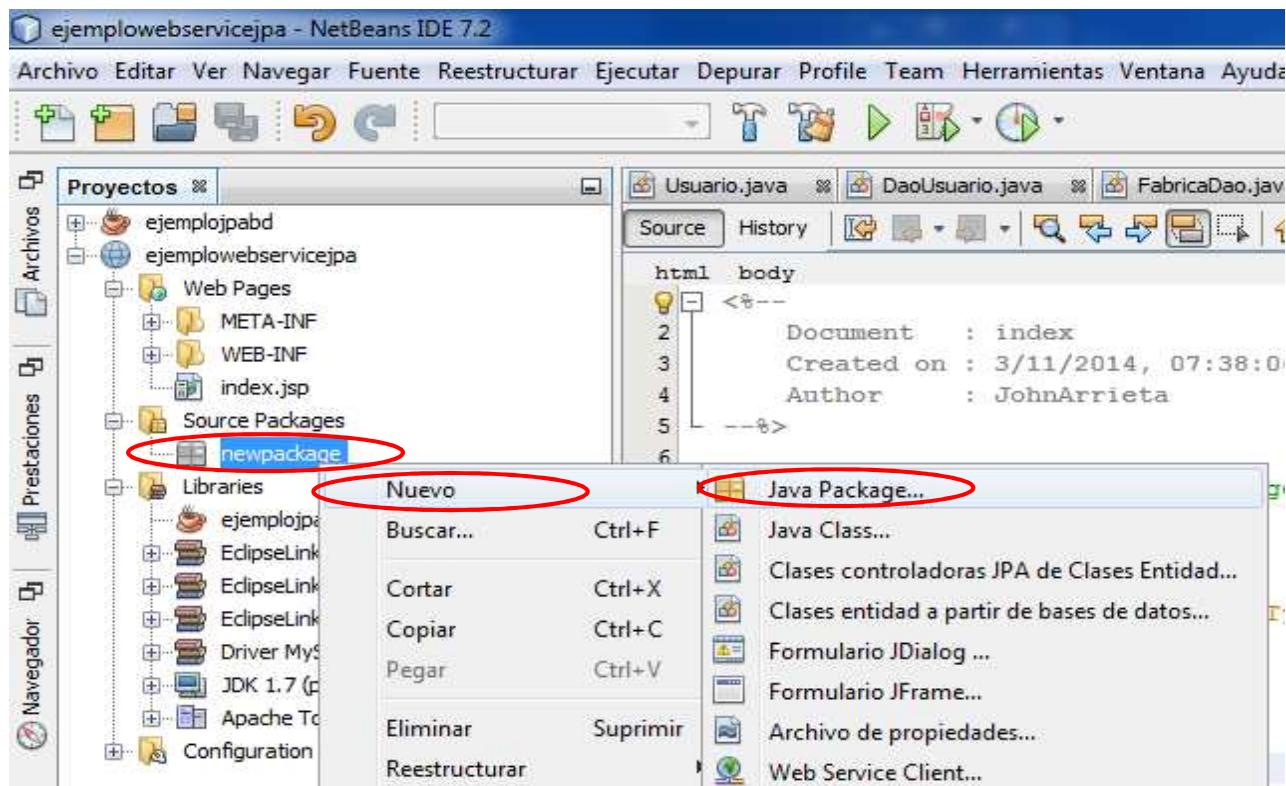
Una vez agregas las librerías necesarias, podemos apreciar cómo quedaría el árbol de nuestro proyecto y en efecto verificamos que las librerías fueron agregadas.



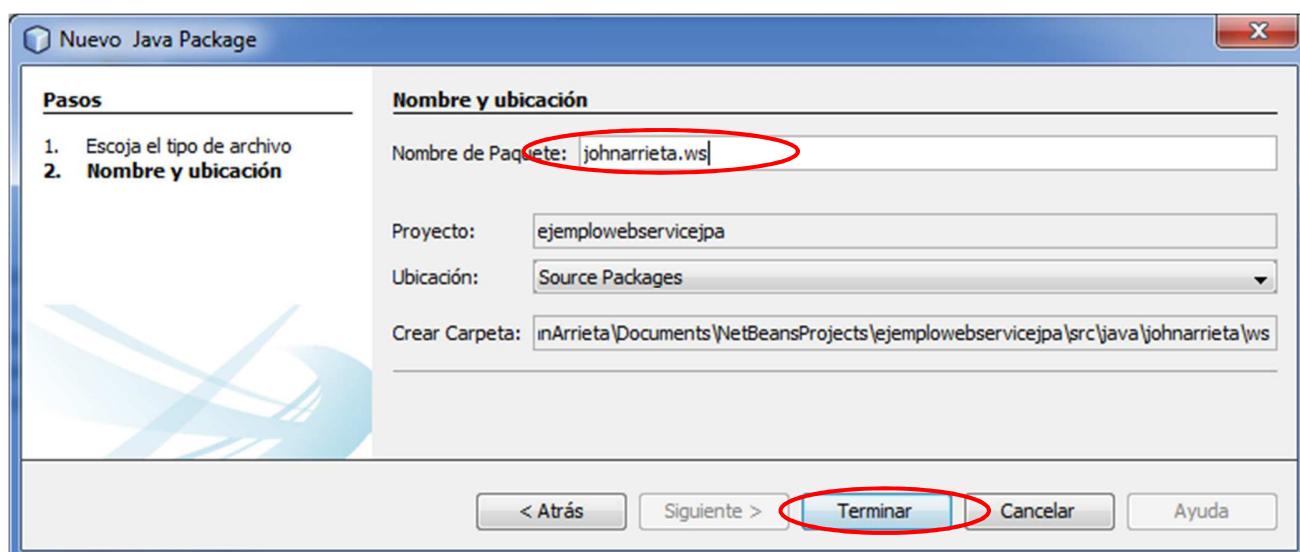


## Paso 13: AGREGAR LOS PAQUETES Y EL WEB SERVICES

Este paso consiste simplemente en agregar el paquete donde irá la clase del WebService y de paso crear el Webservices dotándolo de al menos una operación que pueda ser utilizada, consumida o invocada remotamente.

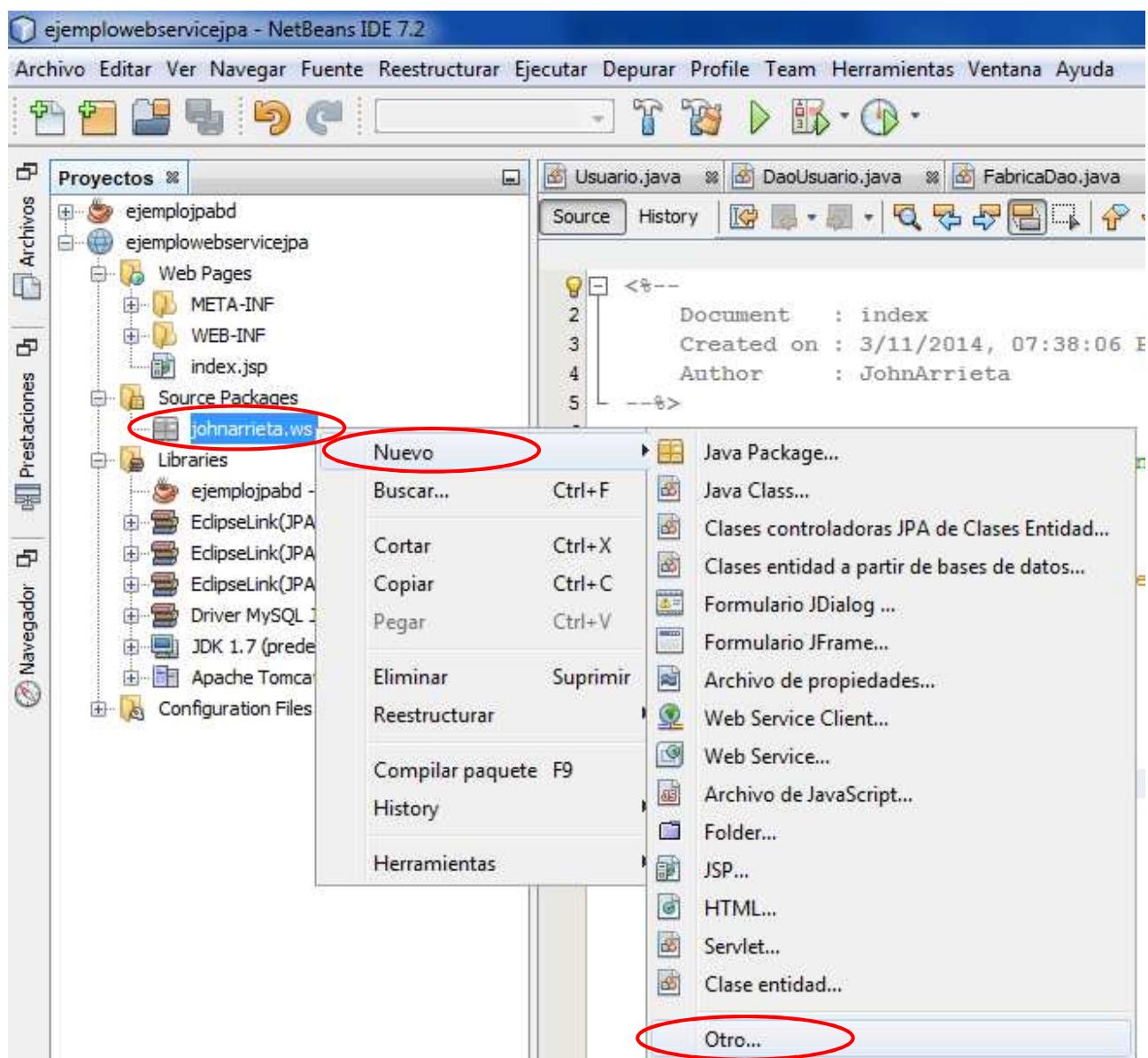


Indicamos el nombre del paquete

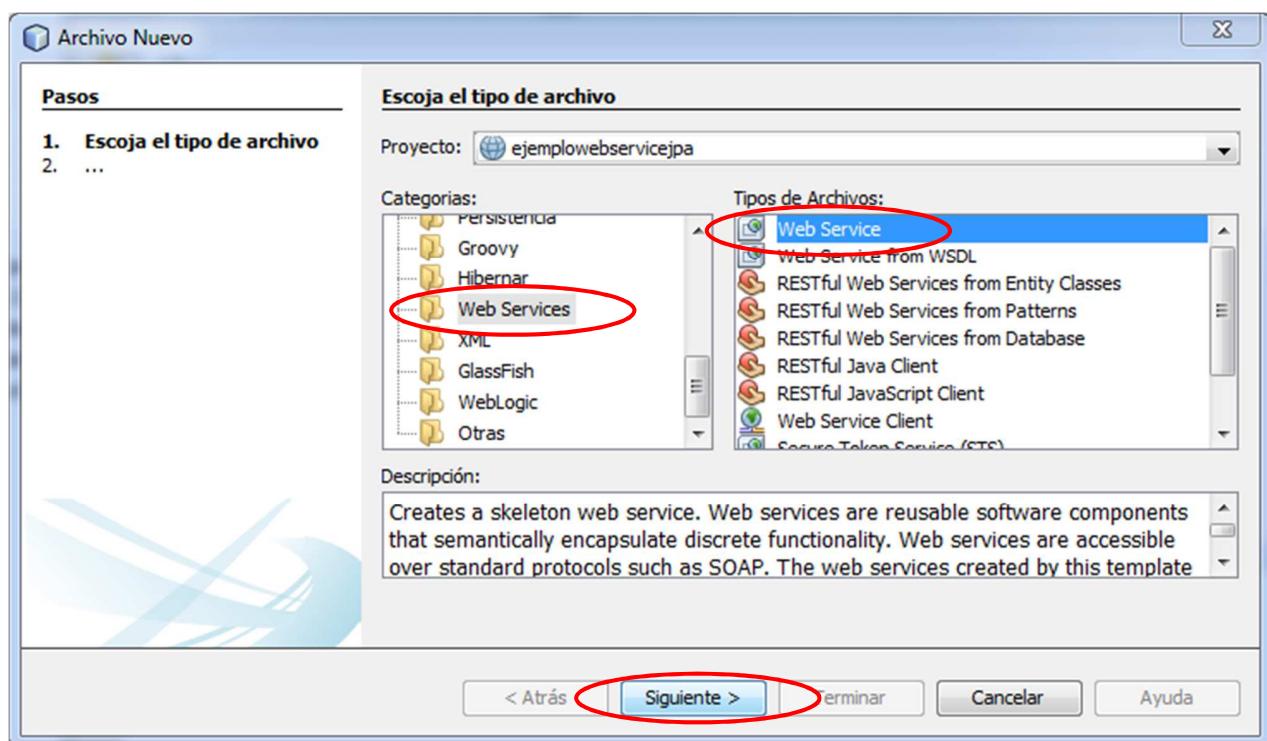




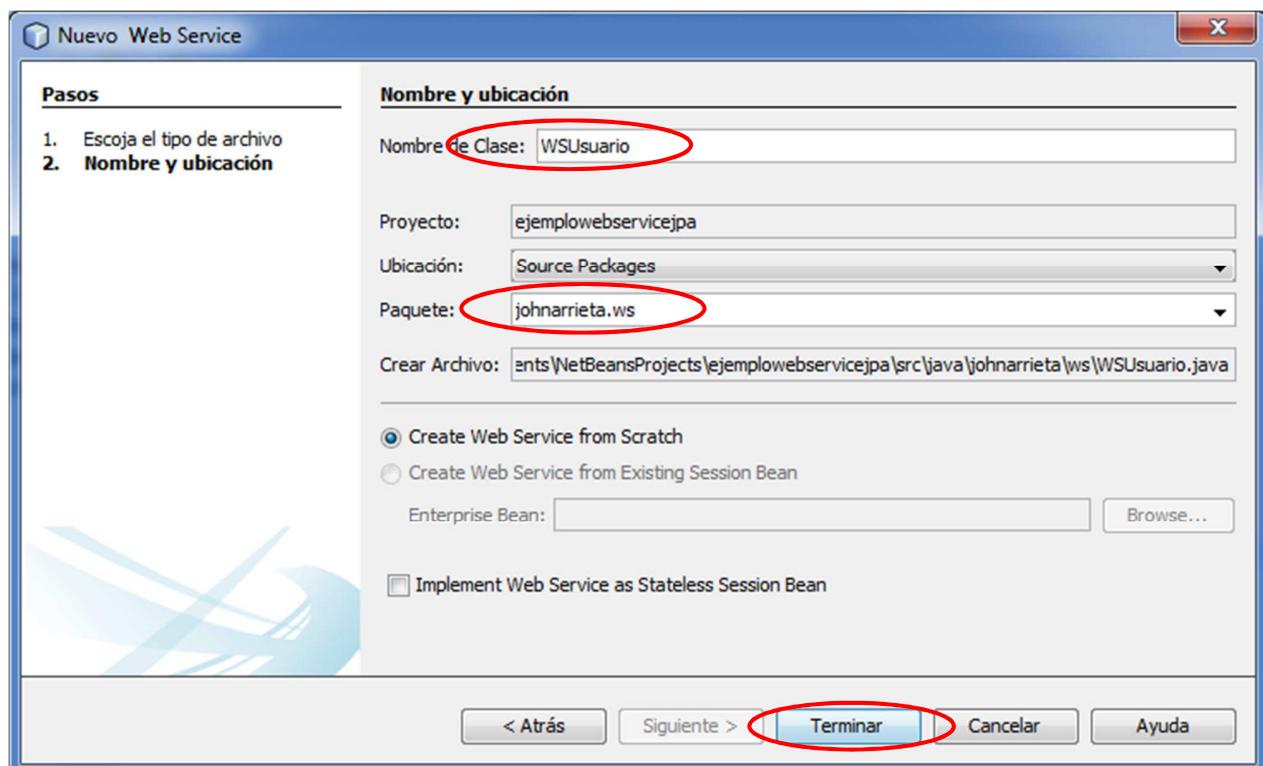
Ahora creamos la clase del WebService dentro del paquete anterior, pero ello seguimos los pasos de las imágenes que se presentan a continuación:



En la siguiente ventana debemos escoger la categoría WebServices y en tipo de Archivo seleccionamos WebServices, aunque como vemos existen más de una opción para WebServices, en este caso vamos a crear un WebServices estándar, puesto que actualmente podemos usar WebServices Genéricos, Web Services RestFul que son aún más simples pues emulan un formulario HTTP, como si se tratara de una aplicación Web normal.



Ahora ingresamos el nombre de la Clase y escógenos el paquete donde será guardada dicha clase

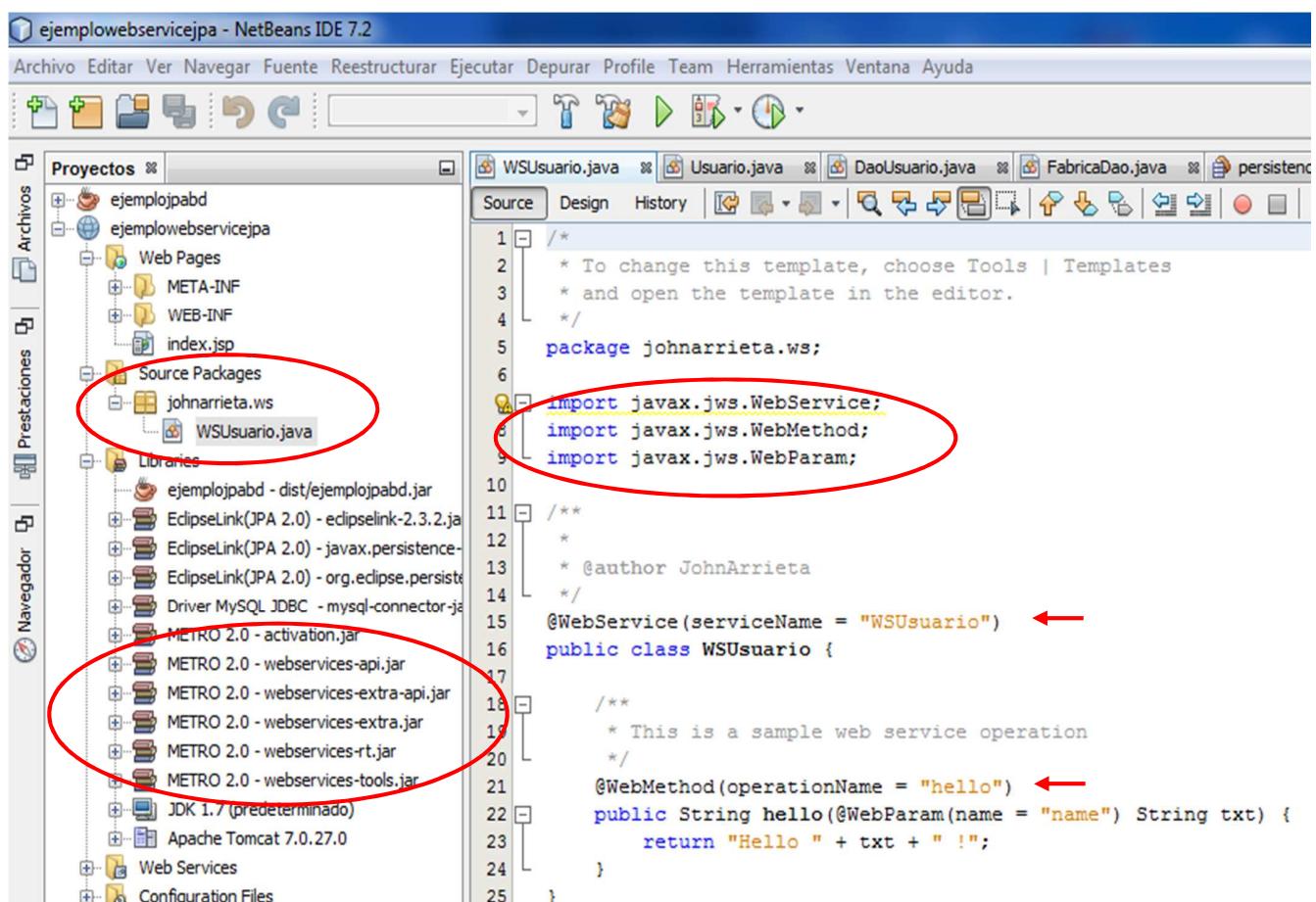




Nos aparece una ventana de notificación en la cual se nos informa que el WebServices será creado bajo el estándar J2EE JSR-109 conocida como METRO WebServices y su configuración se puede ver en el archivo sun-jaxws.xml.



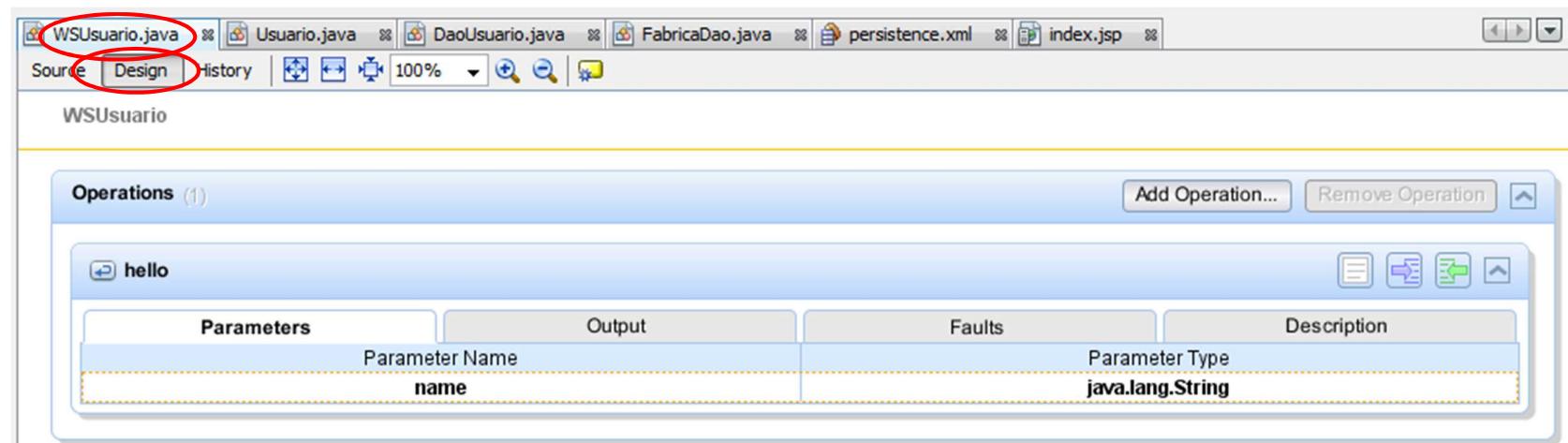
Seguidamente nos podemos ver que el IDE Netbeans no ha generado nuevamente de forma automática gran parte de los archivos y código fuente necesario para comenzar a trabajar con WebServices desde nuestra aplicación Web. Estos elementos incluyen las librerías.jar METRO 2.0 WebService, importación de clases y @anotaciones





## Paso 14: AGREGAR OPERACIONES REMOTAS AL WEB SERVICES

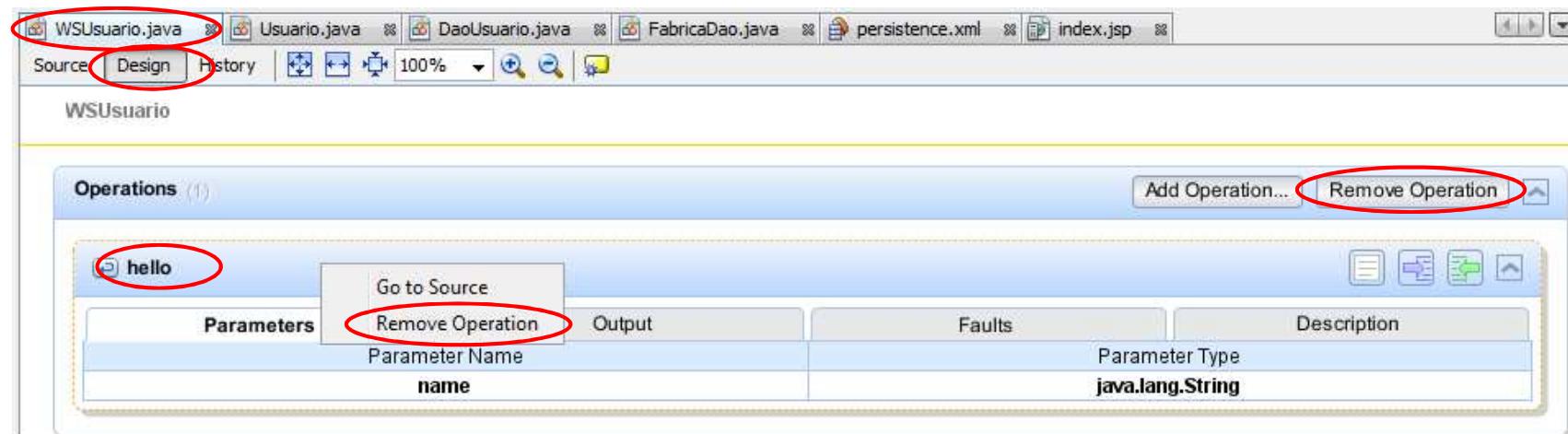
Observemos que dentro del código de la clase del WebServices llamada WSUsuario existe un método llamado hello, el cual esta marcado con la anotación Java **@WebMethod (operationName = "hello")** esto significa que este método será una de las operaciones del WebServices que podrán ser invocadas, consumidas o llamadas de forma remota a través de la internet por cualquier aplicación Cliente WebServices construida para tal fin. Nosotros podemos suprimir o eliminar este método puesto que es generado por el IDE Netbeans a manera de ejemplo, para ello debemos ir al panel de diseño de WebServices, siguiendo los pasos que se indican en las siguientes imágenes:



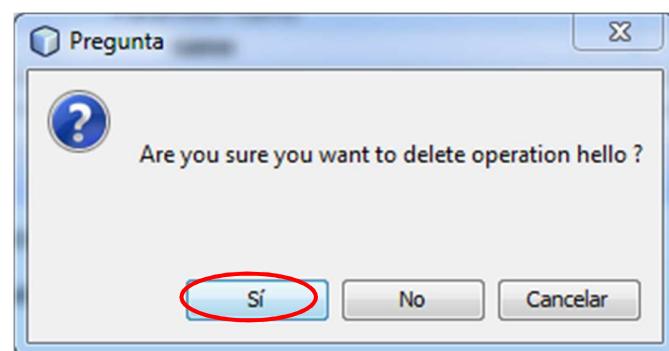
En este panel podemos apreciar los diferentes métodos u operaciones del WebServices que podrán ser invocados remotamente desde internet, mediante el panel de diseño podemos también configurar algunos parámetros como el nombre con el que será llamada la operación (el nombre del método no tiene que ser igual al nombre con el que será invocado desde internet), los parámetros que recibe como argumentos de entrada enviados desde internet y el tipo y valor que debe retornar cada operación, es decir el valor que debe devolver al Cliente WebServices que decida invocarla remotamente.



Para eliminar una operación o método remoto del WebServices procedemos a seleccionarlo desde el panel de diseño, luego damos click o click derecho sobre su nombre y escogemos la opción Remove Operation, o simplemente pulsamos el botón Remove Operation.



Seguidamente se nos pide confirmar o rechazar las opciones de eliminación de la operación seleccionada, para lo que respondemos con SI

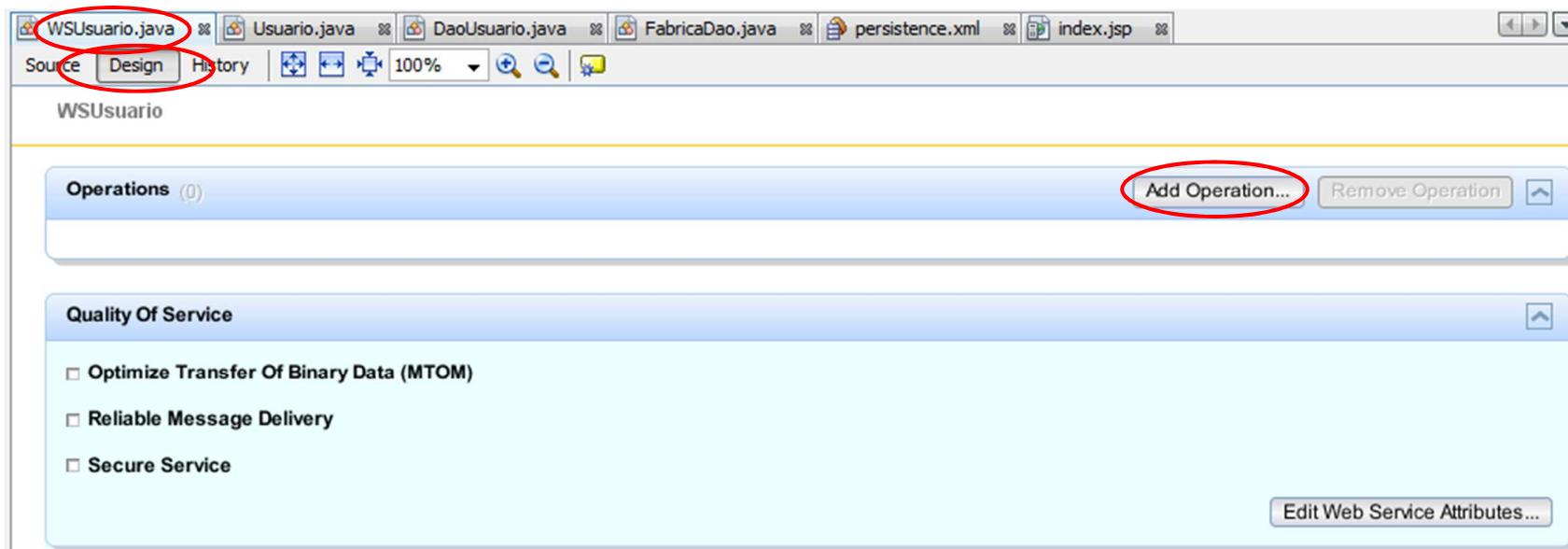




Una vez eliminada la operación del WebServices podemos verificar que esta no se encuentra en el panel de diseño, en este ejemplo el panel de diseño esta vacío pues la operación Hello que eliminamos era la única operación o método remoto que tenía el este WebService.

Ahora procedemos a agregar una nueva operación Remota para este WS, recordemos que por cada operación remota que agregemos ser generara un método Java dentro del cuerpo de la clase del WS sobre el cual estamos trabajado, estos métodos están marcados con las anotaciones Java @WebMethod y @WebParam, las cuales indican al compilador Java que debe generar el código Java y XML necesario para que dicho método pueda ser invocado, llamado o consumido por Internet desde un cliente WS, pero que además debe recibir X número y tipo de parámetros.

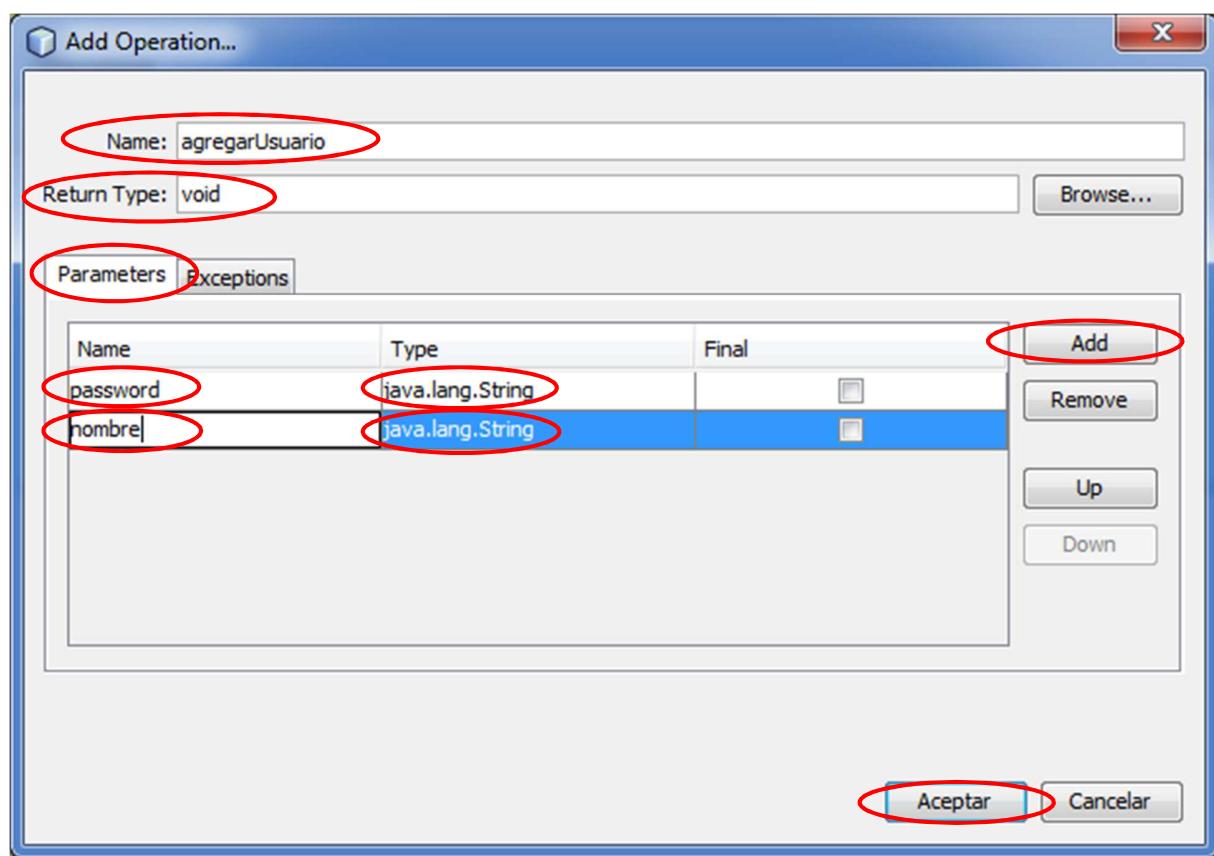
Seguir los siguientes pasos para agregar una nueva operación al WS:





Seleccionar el archivo del WB desde el panel de Proyecto, seleccionar el Panel de diseño y Pulsar el botón Add Operation.

A continuación nos aparece la siguiente ventana en la cual debemos ingresar el nombre de la operación remota (el cual será marcado con la anotación @WebMethod (name= ) ), los parámetros de entrada (los cuales serán marcados con la anotación @WebParam (name= )) y el tipo de datos que debe retornar dicha operación, tal como se aprecia en la siguiente imagen.



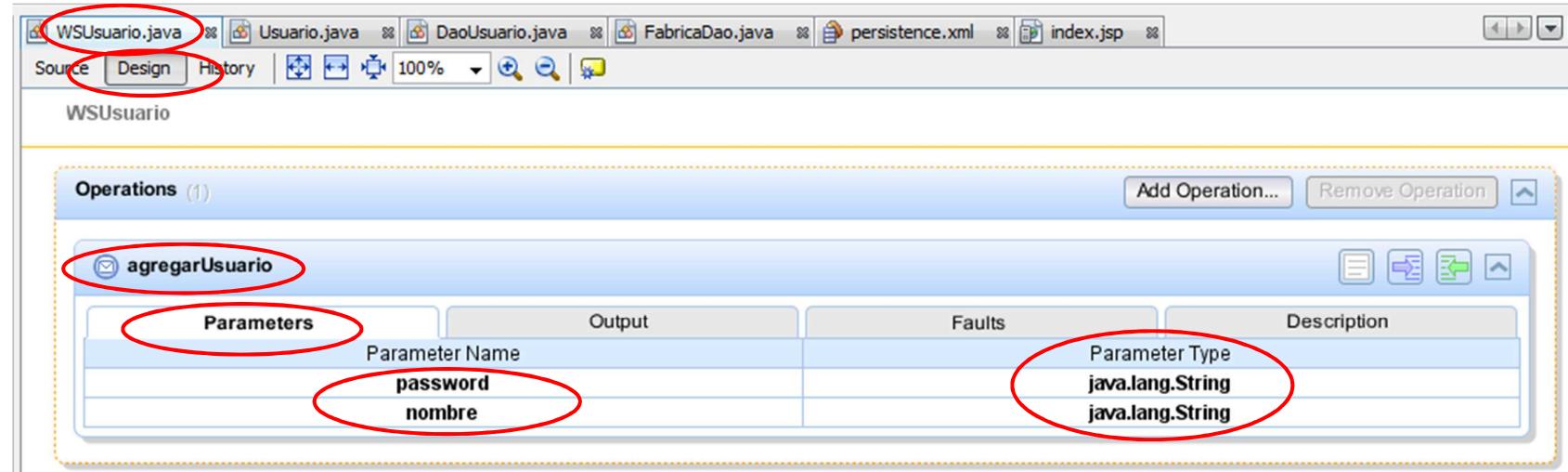
Opcionalmente también podemos especificar si la operación puede o no lanzar algún tipo de Excepcion Java.

**Nota:** Recordemos que las Excepciones Java son representan posibles errores que pueden o no ocurrir en tiempo de ejecución de nuestra aplicación, como por ejemplo, al intentar agregar un usuario a la BD pero el servidor de BD no está en ejecución, esto sin duda genera un error en tiempo de ejecución, cuya información la podemos obtener desde una instancia de la clase Exception.

En esta ventana también podemos editar los valores de la operación en cualquier momento, agregar, eliminar o modificar su nombre, parámetros o valor de retorno.



Al agregar la nueva operación podemos ver que esta ya aparece en el panel de diseño y dentro del código de la clase del WB:



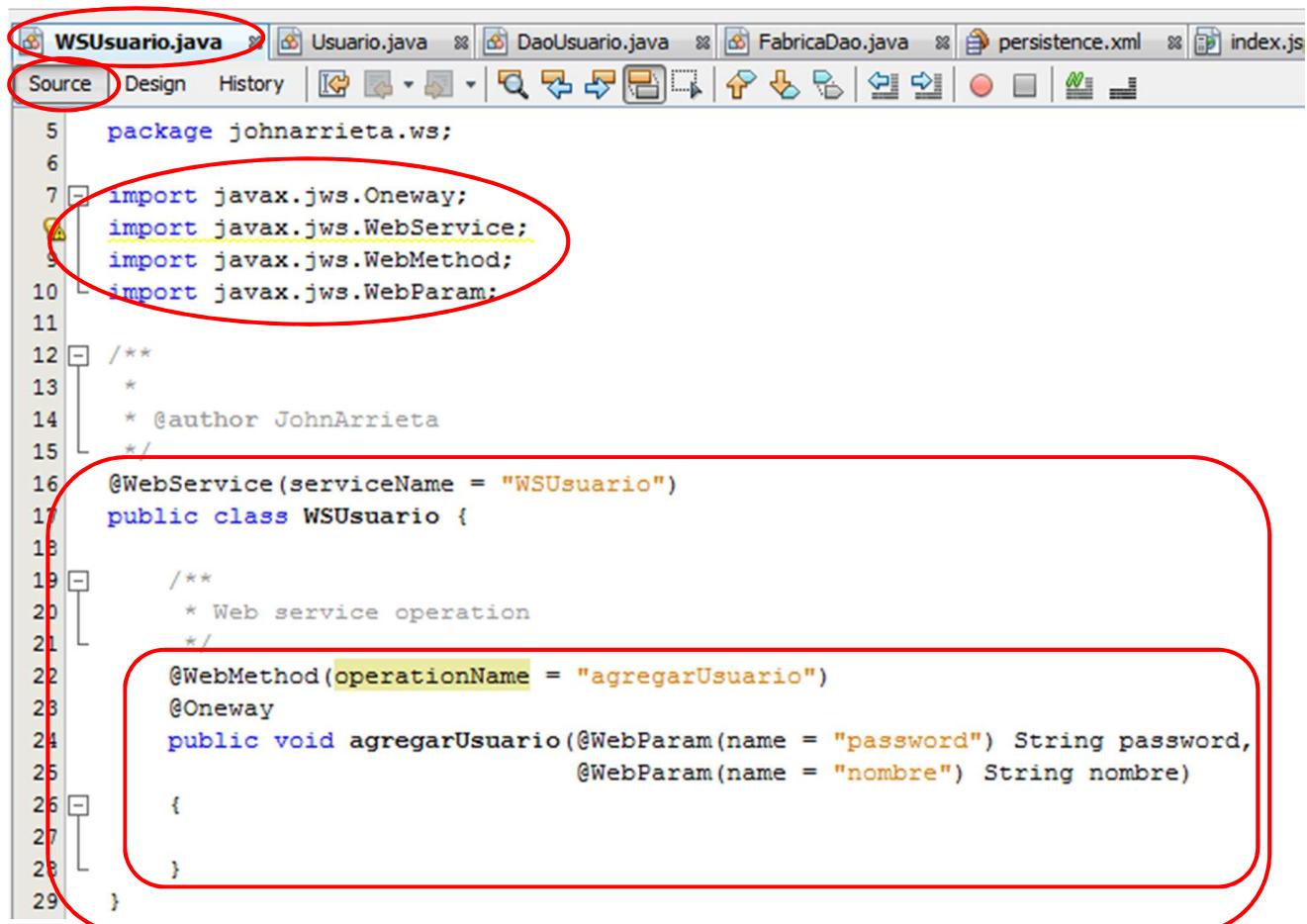
En este ejemplo agregamos al WSUsuario una operación Web llamada **agregarUsuario**, la cual recibe dos parámetros Web, uno llamado **password** de tipo **String** o Texto y otro llamado **nombre** igualmente de tipo **String**, esta operación Web no retornara ningún valor, por lo que hemos colocado como tipo de valor de retorno la palabra Java **void**.

En la siguiente ventana podremos observar y analizar el código fuente Java que se ha generado dentro de la clase WSUsuario, código que define claramente a la operación Web agregarUsuario. Recuerden que esta operación podrá ser invocada o ejecutada por internet de forma remota mediante otra aplicación (que vamos a construir esta misma guia) llamada Cliente WebService, en este caso por homogeneidad la vamos a construir en lenguaje de programación Java, pero en realidad una de las ventajas de los Web services es que han creado un estándar que hace posible construir aplicaciones Web que ponen a disposición un conjunto específico y especializado de operaciones Web, las cuales puede ser utilizadas, invocadas, llamadas, ejecutadas o consumidas a través de una red LAN, MAN o WAN como internet sin necesidad de conocer detalles específicos de cómo fue construida, es decir, no necesitamos saber en qué lenguajes fue escrita, cuál es el servidor sobre el cual se ejecuta, etc. Simplemente necesitamos conocer la dirección o URL donde está disponible para aceptar conexiones. En esta URL se nos entrega un archivo XML el cual contiene la información necesaria para poder utilizar el WebService, como el nombre de sus operaciones Web y los parámetros

## Paso 15: EDITAR EL CÓDIGO DE LAS OPERACIONES WEB AGREGADAS EL WEBSERCES.

En este paso es donde la mayoria de los IDE no nos pueden ayudar, ya que es aquí donde nuestra cabecita comienza a funcionar y escribir las instrucciones de código Java que permitan realizar el procedimiento o calculo que deseamos ofrecer en nuestra aplicación, (sería el colmo que tambien nos generen el código de la logica que deseamos implementar en nuestra aplicacion).

En la siguiente ventana podemos observar el código generado por el IDE para nuestro WS.



```
5 package johnarrieta.ws;
6
7 import javax.jws.Oneway;
8 import javax.jws.WebService;
9 import javax.jws.WebMethod;
10 import javax.jws.WebParam;
11
12 /**
13 *
14 * @author JohnArrieta
15 */
16 @WebService(serviceName = "WSUsuario")
17 public class WSUsuario {
18
19     /**
20      * Web service operation
21     */
22     @WebMethod(operationName = "agregarUsuario")
23     @Oneway
24     public void agregarUsuario(@WebParam(name = "password") String password,
25                               @WebParam(name = "nombre") String nombre)
26     {
27     }
28 }
29 }
```

**Línea 5:** indica que el archivo de la clase WSUsuario sera almacenado en el paquete johnarrieta.ws.

**Línea 7 a 10:** indica que esta clase necesita importar y utilizar las propiedades, métodos o anotaciones de algunas clases que se encuentran en el paquete **javax.jws**, tales como las clases Oneway, WebServices, WebMethod y WebParam.

**Línea 16 a 29:** corresponden al inicio y fin de la clase WSUsuario, como ya sabemos esto indica que la clase es pública (se puede utilizar en cualquier parte del programa), se llama WSUsuario, la { llave de inicio y la llave } de fin de su cuerpo.



**Línea 22 a 28:** en este bloque de código se define un método Java que corresponde a la operación Web que hemos agregado al WS, observemos que en la línea 22 se encuentra la anotación `@WebMethod(operationName="agregarUsuario")`, esta anotación es reconocida por el compilador Java y el indica que el siguiente método (lámese como se llame) debe ser tratado como una Operación Web que será llamada o invocada remotamente con el nombre de agregarUsuario, en este ejemplo tanto el nombre del método como el nombre con el que será invocado son iguales, pero eso no es una regla y pueden ser diferentes, pero el nombre con el que se desea invocar debe estar especificado como valor del atributo `operationName` de la anotación `@WebMethod`.

**Línea 27:** Define la firma o cabecera del método, como ya sabemos, este es público, no retorna nada, se llama agregarUsuario y recibe dos parámetros ambos de tipo String, los valores de estos parámetros serán suministrados como Parámetros Web en el momento que este método o operación Web sea invocada remotamente. Entre las {} no hay código Java, esto indica que el cuerpo del método está vacío, osea que al ser invocado no hace absolutamente nada.

En la siguiente imagen podemos observar que el código de la clase WSUsuario ha cambiado, esto es porque yo le he agregado las instrucciones Java necesarias para que al momento de ser invocada remotamente la operación agregarUsuario pueda agregar un nuevo Registro a la BD, cuyos datos serán pasados a los parámetros Web.

```

WSUsuario.java  ✘ Usuario.java  ✘ DaoUsuario.java  ✘ FabricaDao.java  ✘ persistence.xml  ✘ index.js
Source Design History |  ☰  ☱  ☲  ☳  ☴  ☵  ☶  ☷  ☸  ☹  ☺  ☻  ☻  ☻  ☻  ☻  ☻
11 import johncarlosarrietaarrieta.dao.DaoUsuario;
12 import johncarlosarrietaarrieta.dao.FabricaDao;
13 import johncarlosarrietaarrieta.entidades.Usuario;
14
15 /**
16 *
17 * @author JohnArrieta
18 */
19 @WebService(serviceName = "WSUsuario")
20 public class WSUsuario {
21
22     /**
23      * Web service operation
24      */
25     @WebMethod(operationName = "agregarUsuario")
26     @OneWay
27     public void agregarUsuario(@WebParam(name = "password") String password,
28                                @WebParam(name = "nombre") String nombre)
29     {
30         Usuario alguien = new Usuario();
31         alguien.setPassword(password);
32         alguien.setNombre(nombre);
33         DaoUsuario adoUsuario = FabricaDao.getDaoUsuario();
34         adoUsuario.agregarUsuario(alguien);
35     }
36 }
37

```



**Líneas 11 a 13:** como ya podemos intuir, estas líneas importan las clases DaoUsuario, FabricaDao y Usuario, las cuales se encuentran en la librería ejemplopabd.jar creada y agregada a este proyecto anteriamente.

**Líneas 30 a 34:** Conforman las instrucciones del cuerpo del método agregarUsuario.

**Línea 30:** Creamos una instancia u objeto de tipo **Usuario** y la guardamos en una variable llamada **alguien**.

**Línea 31:** utilizamos la variable **alguien** para invocar el método setPassword de la instancia de tipo Usuario, este método recibe como parámetro una variable de tipo String, cuyo valor sea copiado a la propiedad password definida en la clase Usuario, por ese motivo le pasamos como argumento o parámetro la variable password quien es uno de los parámetros que recibe el método agregarUsuario, este password será suministrado remotamente por la aplicación Cliente Web Services que invoque a este método como Operación Web.

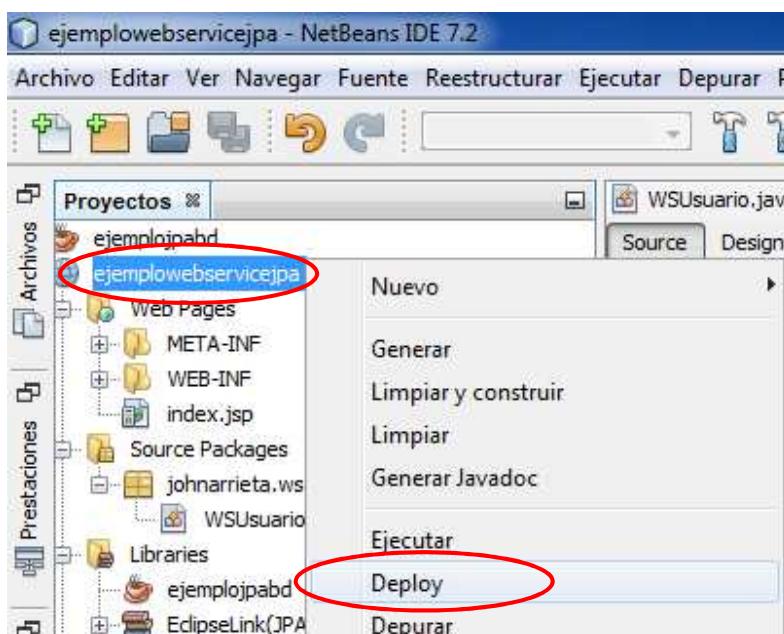
**Línea 32:** similar a la operación que hicimos en la línea 31, siendo que en esta ocasión se trata de la propiedad nombre de la instancia Usuario.

**Línea 33:** Obtenemos una instancia de tipo DaoUsuario apartir de la llamada al método getDaoUsuario de la clase FabricaDao, esto fue desarrollado y explicado anteriormente durante la construcción del proyecto ejemplopabd.

**Línea 34:** utilizamos la instancia u objeto de tipo DaoUsuario para invocar su método agregarUsuario al cual le pasamos como parámetro la instancia de tipo Usuario, este método agregarUsuario es quien realiza la operación de guardar en la BD los datos de la instancia de tipo Usuario, es decir id, password y nombre.

## Paso 16: DESPLEGAR EL WEBSERVICES Y EJECUTAR LA APLICACIÓN WEB.

Terminado el proceso de creación del WebServices y definidas las Operaciones Web que se van a poder invocar remotamente de este una Red como internet, ahora solo falta desplegarlo, es decir, colocar el Servicio Web listo y a disposición en una URL válida y disponible desde internet. Seguir los pasos indicados en la siguiente imagen.





Esto debe iniciar la ejecución del servidor de aplicaciones que utilizamos en este ejemplo, es decir Apache Tomcat, luego de esto el servicio web debe estar disponible en una URL que con la siguiente forma:



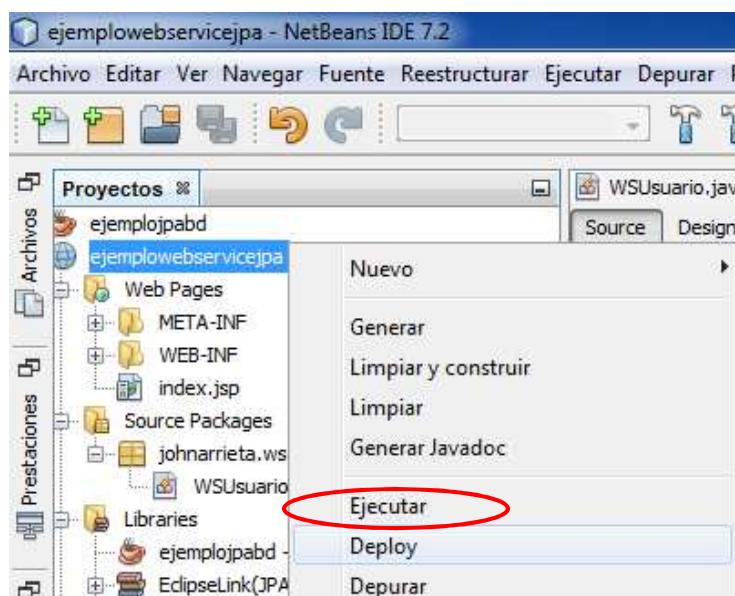
#### Descripción de cada parte:

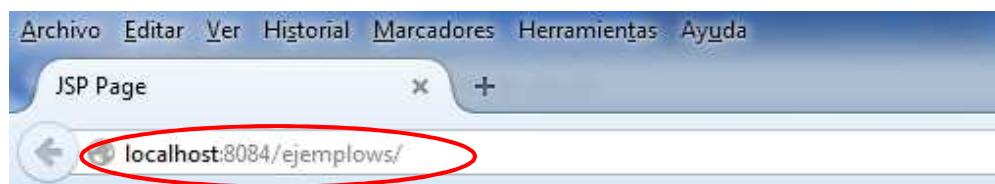
1. Es el protocolo de red, es este caso es http porque es una app web
2. Es la dirección IP (local, Privada o Pública) del servidor donde está corriendo nuestra aplicación WebServices, o el nombre de dominio (dominio de internet o nombre del PC servidor), comúnmente cuando estamos probando la App lo hacemos en el mismo PC donde la estamos desarrollando, por lo tanto la dirección IP será **127.0.0.1** o en su defecto el nombre de dominio será **localhost**.
3. El número de puerto lógico por donde escucha y atiende peticiones el servidor, en Tomcat por defecto es 8084.
4. Es la ruta de acceso que decidimos dar a nuestra Aplicación Web, ver página 34.
5. Corresponde al nombre que dimos a nuestro Servicio Web.

Teniendo en cuenta lo anterior, la URL de nuestro Servicio Web sea:

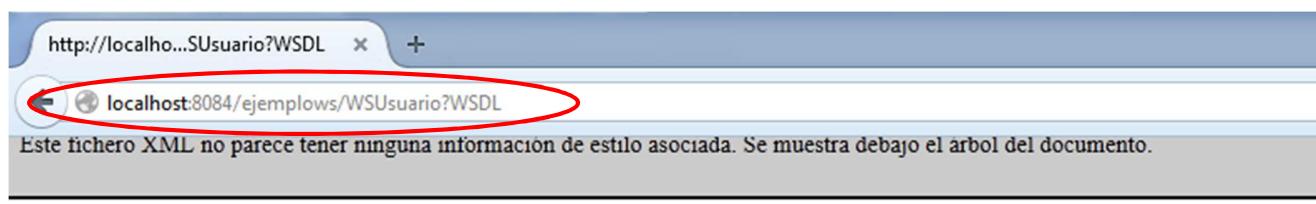
<http://localhost:8084/ejemplows/WSUsuario?WSDL>

Un Web Service puede funcionar en segundo plano al mismo tiempo que lo hace una aplicación Web dinámica Servlet, JSP o JSF, ya que ambos son parte integral de una misma aplicación Web, para probar la aplicación Web solo debemos dar click derecho sobre la carpeta raíz del proyecto y escoger la opción Ejecutar.





## Hello World!



```
--<!--
    Published by JAX-WS RI at http://jax-vs.dev.java.net. RI's version is JAX-WS RI 2.2-hudson-740-.
-->
--<!--
    Generated by JAX-WS RI at http://jax-vs.dev.java.net. RI's version is JAX-WS RI 2.2-hudson-740-.
-->
- <definitions targetNamespace="http://ws.johnarrieta/" name="WSUsuario">
-   <types>
-     <xsd:schema>
        <xsd:import namespace="http://ws.johnarrieta/" schemaLocation="http://localhost:8084/ejemplows/WSUsuario?xsd=1"/>
      </xsd:schema>
    </types>
-   <message name="agregarUsuario">
    <part name="parameters" element="tns:agregarUsuario"/>
  </message>
-   <portType name="WSUsuario">
    - <operation name="agregarUsuario">
      <input wsam:Action="http://ws.johnarrieta/WSUsuario/agregarUsuario" message="tns:agregarUsuario"/>
    </operation>
  </portType>
-   <binding name="WSUsuarioPortBinding" type="tns:WSUsuario">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="agregarUsuario">
      <soap:operation soapAction="" />
      - <input>
        <soap:body use="literal"/>
      </input>
    </operation>
  </binding>
-   <service name="WSUsuario">
    - <port name="WSUsuarioPort" binding="tns:WSUsuarioPortBinding">
      <soap:address location="http://localhost:8084/ejemplows/WSUsuario"/>
    </port>
  </service>
</definitions>
```



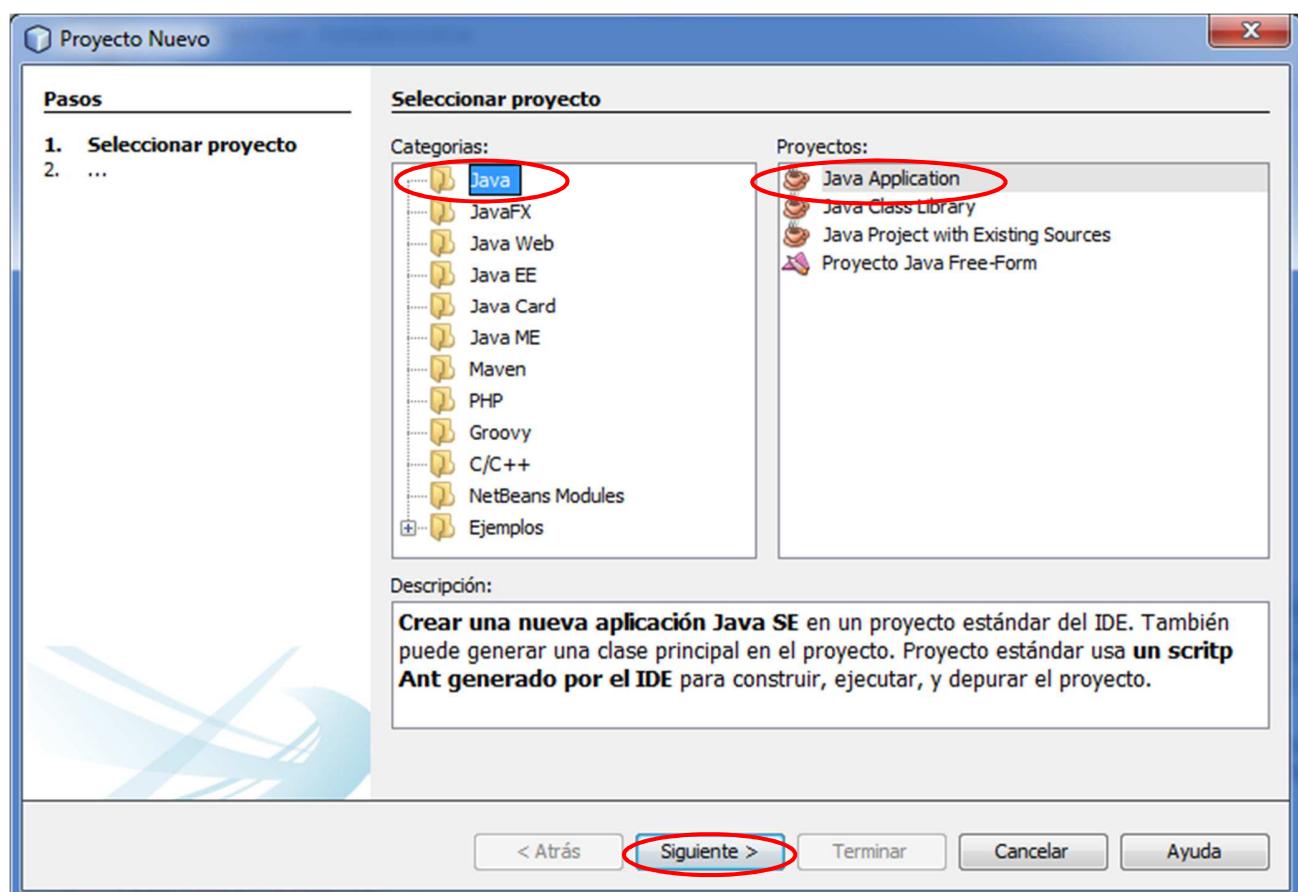
## Paso 17: CREAR LA APLICACIÓN CLIENTE WEBSERVICES.

A este punto ya tenemos dos aplicaciones totalmente funcionales:

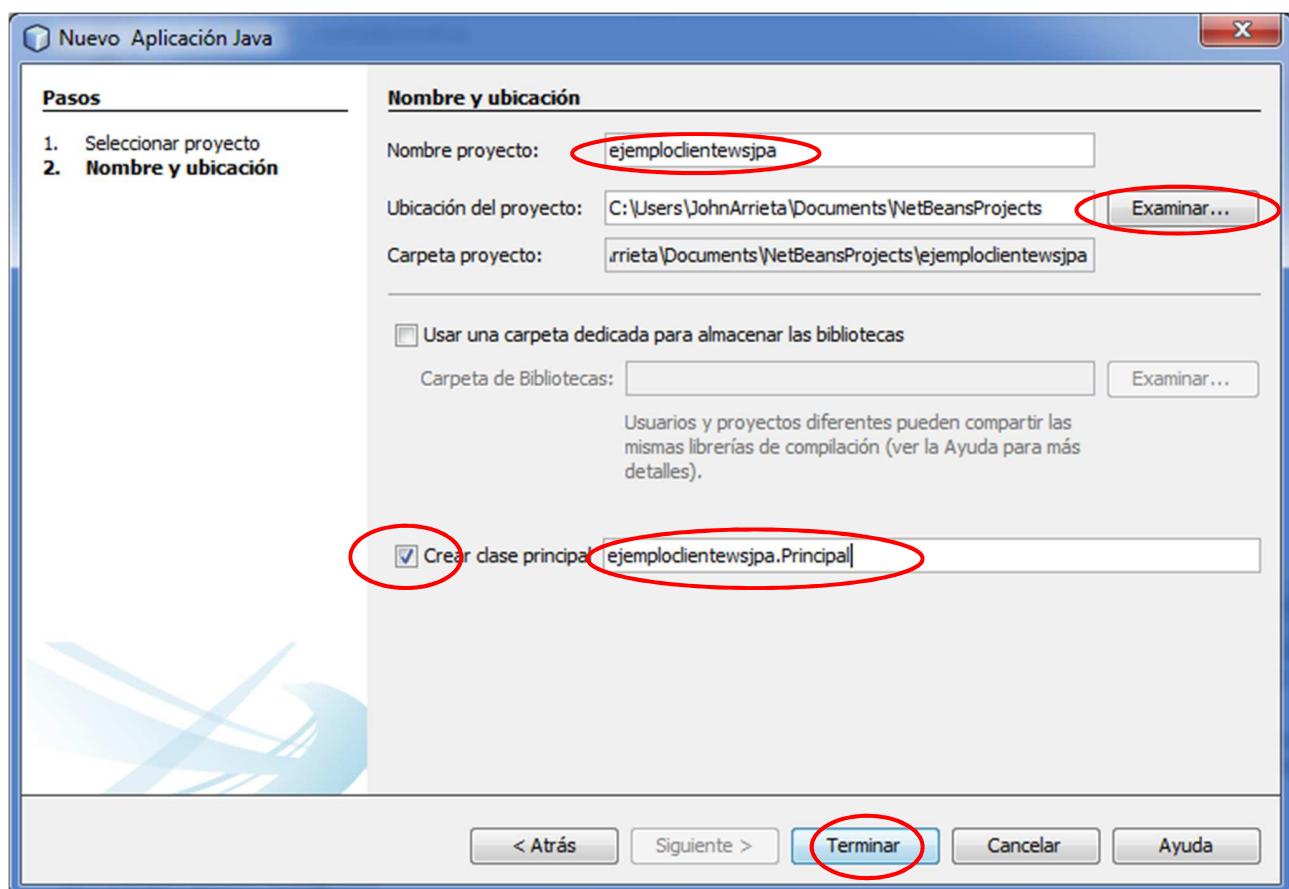
Una librería Java que se encarga la persistencia de los datos de los usuarios, realizando operaciones CRUD (Create, Read, Update Y Delete) sobre la tabla Usuarios en la base de datos.

Una Aplicación Web Java que contiene y despliega un Servicios Web llamado WSUsuario, quien dispone de una operación llamada agregarUsuario, la cual al ser invocada remotamente recibe como parámetros un nombre y password quienes serán utilizados para ser pasados como parámetros a las clases de la Librería de Persistencia y poder guardarlos en la Base de datos.

Solo nos falta construir la aplicación cliente que se conectara al Servicio Web, le pasara nombres y password para que sean agregados a la Base de datos. Esto es justo lo que vamos a aprender a realizar en estos pasos.



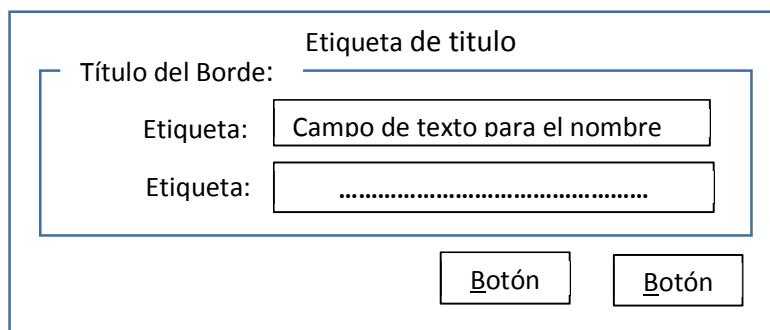
Vamos crear una aplicación de ventanas para escritorio Windows, MacOS o GNU/Linux, para ellos creamos un nuevo proyecto y seleccionamos la categoría **Java**, luego el tipo de proyecto seleccionamos **Java Application**.

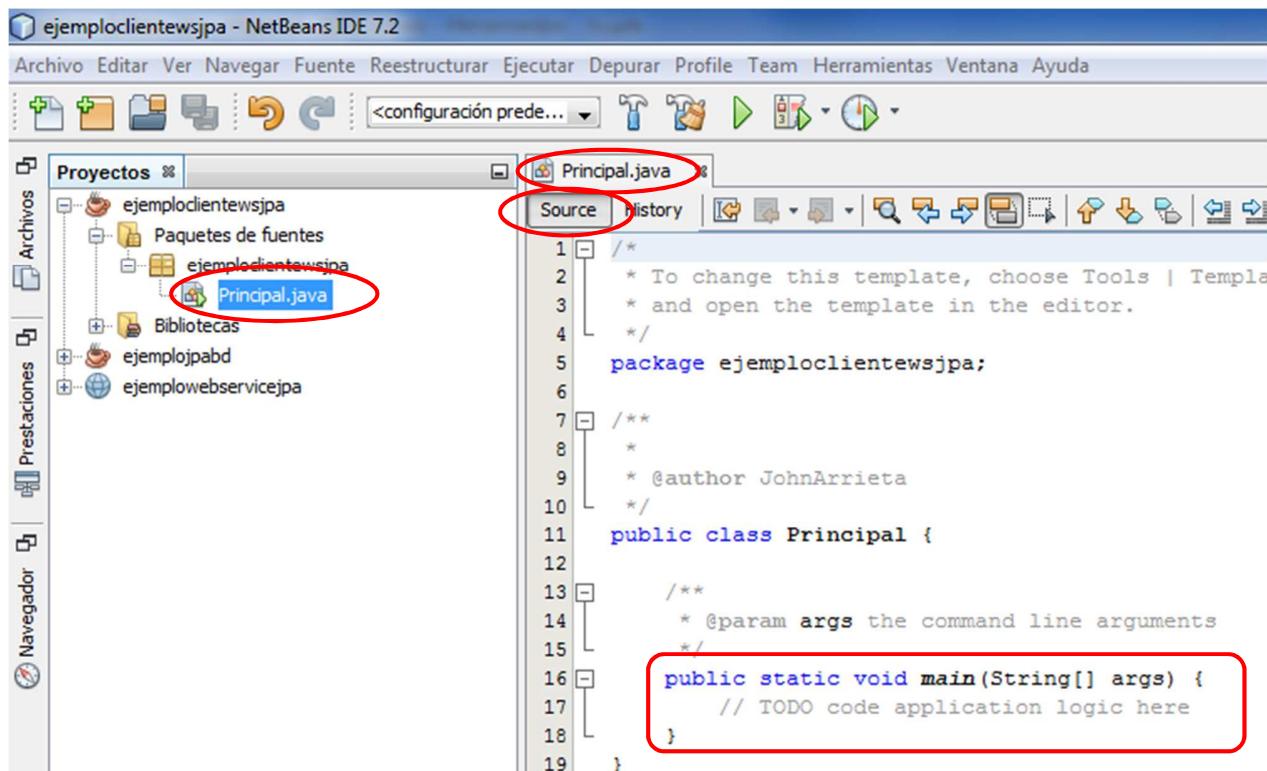


Toda aplicación de escritorio requiere de un punto de inicio o ejecución, el cual es una clase que contiene un método especial de Java llamado main, a esta clase se le llama clase Principal y el IDE genera una la clase principal y su método main, en la ventana anterior podemos decidir si creamos o automáticamente dicha clase.

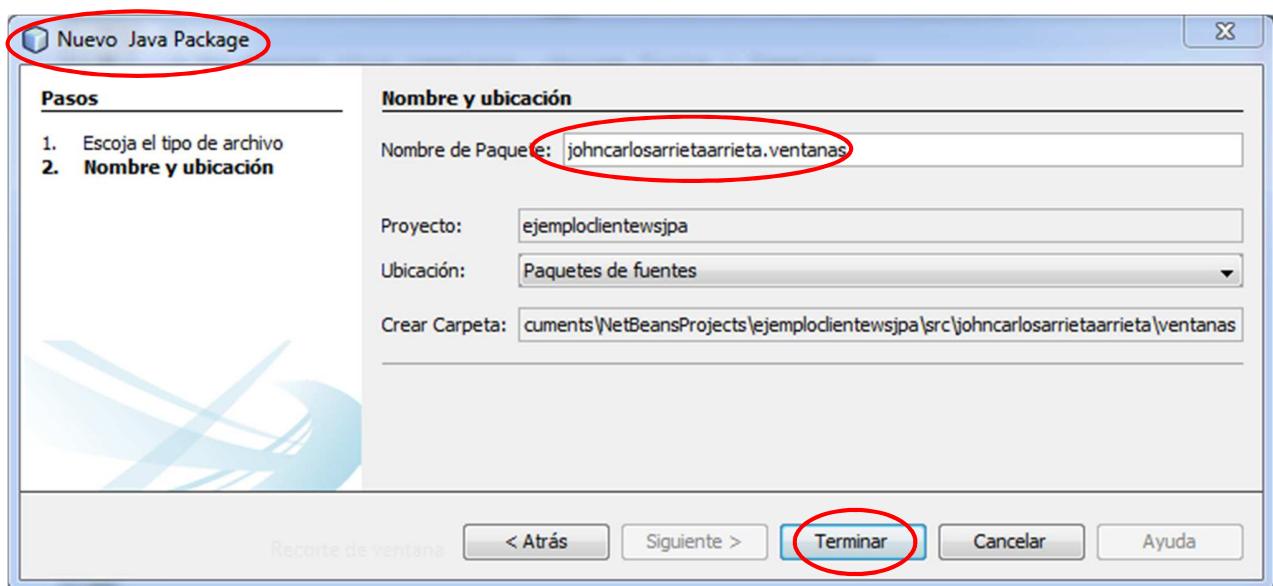
Esta tercera aplicación constara de una ventana con un formulario que tiene dos campos de texto, uno para ingresar el nombre y otro para ingresar el password del usuario, también tendrá dos botones, uno será utilizado para enviar al WebService los datos ingresados en los campos nombre y password, el otro botón se utilizara para limpiar y dejar vacíos los campos nombre y password del formulario, un ejemplo del diseño sería como el siguiente

#### Paso 18: DISEÑAR LA INTERFAZ GRAFICA DE USUARIO O GUI DE LA APP CLIENTE WS





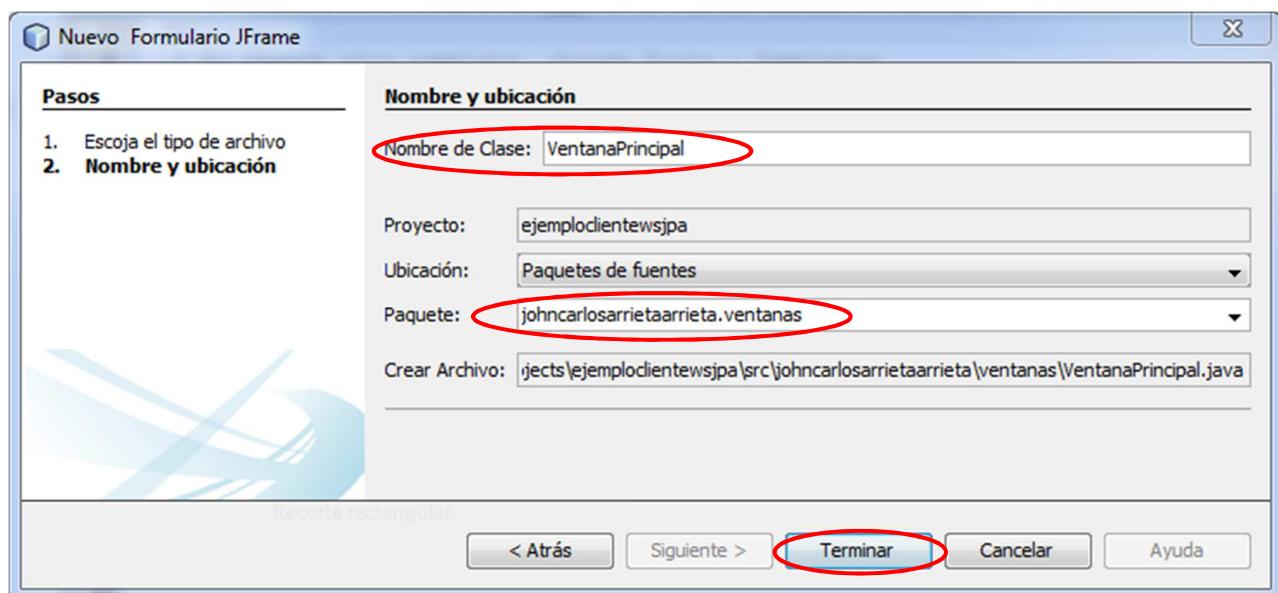
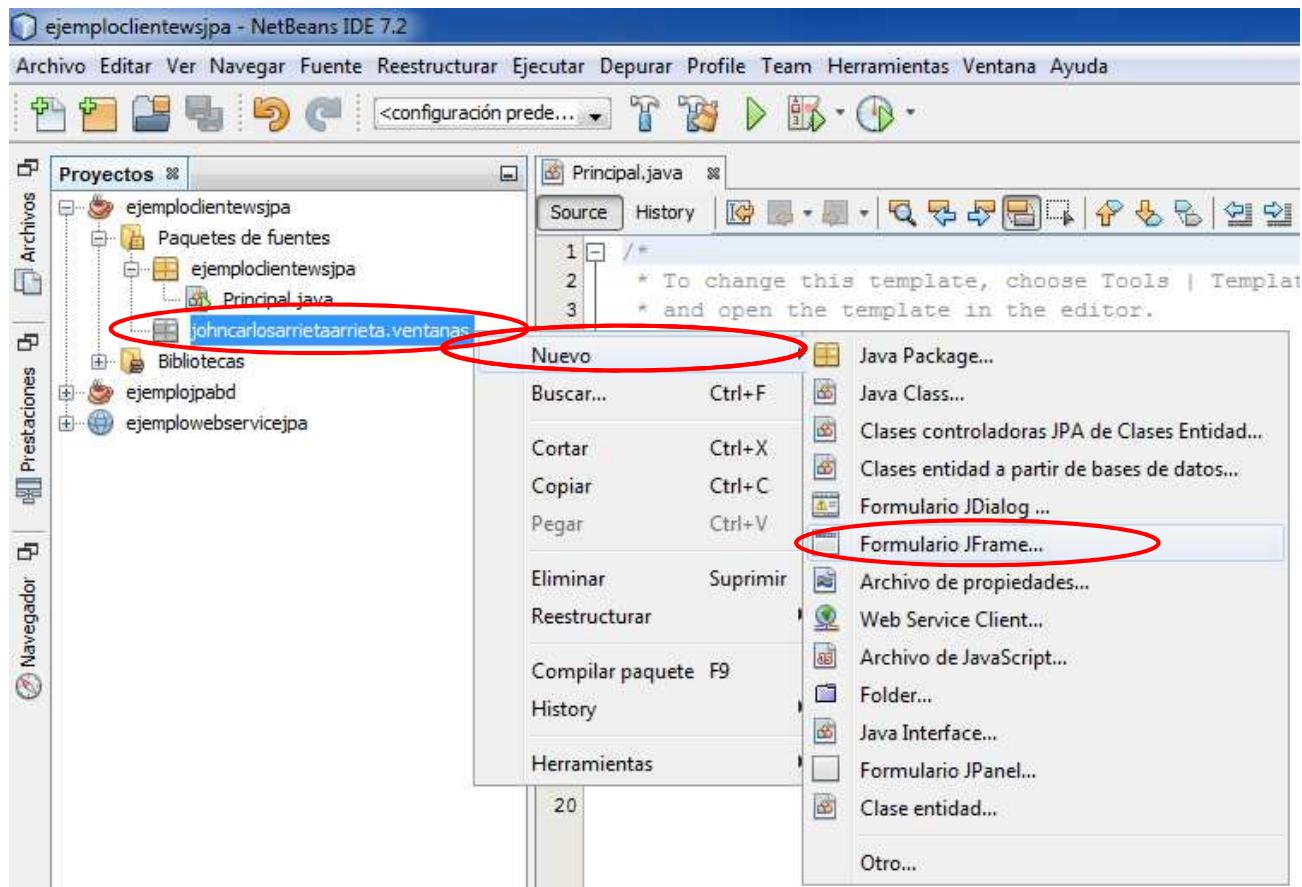
Ahora creamos un nuevo paquete para guardar la clase de la VentanaPrincipal de tipo JFrame.

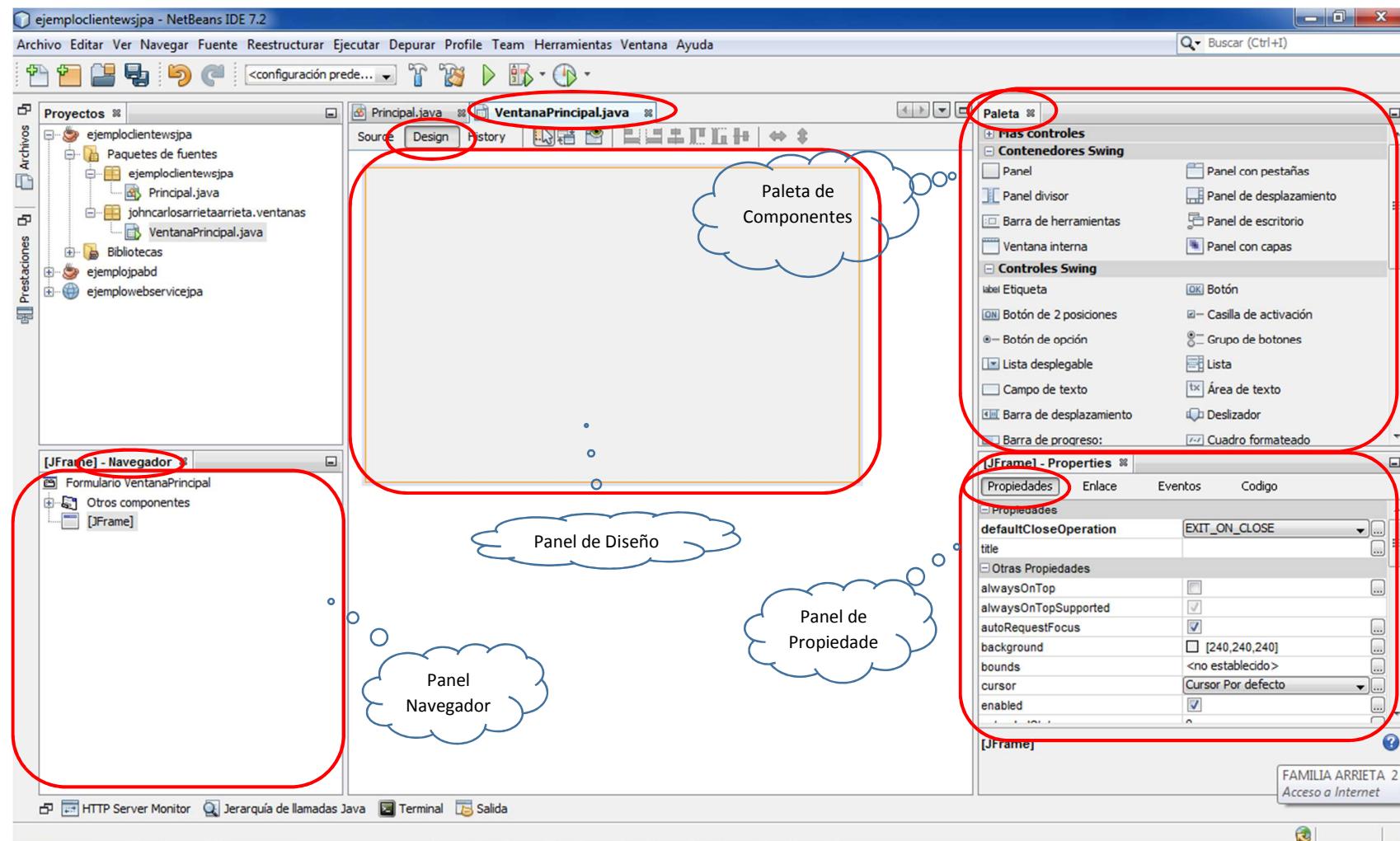


Ahora creamos la clase VentanaPrincipal, la cual debe heredar de la clase JFrame proporcionada por el JDK, esta clase se encuentra en el paquete javax.swing y contiene todo el código necesario para trabajar con componentes GUI de tipo Ventanas, al hacer que nuestra clase VentanaPrincipal herede de JFrame haremos que esta sea igualmente una Ventana sin tener que escribir preocuparnos como programarla, ya que esta fue programada en la clase JFrame.



Para crear una clase de tipo JFrame usando el IDE Netbeans debemos seguir los siguientes pasos:



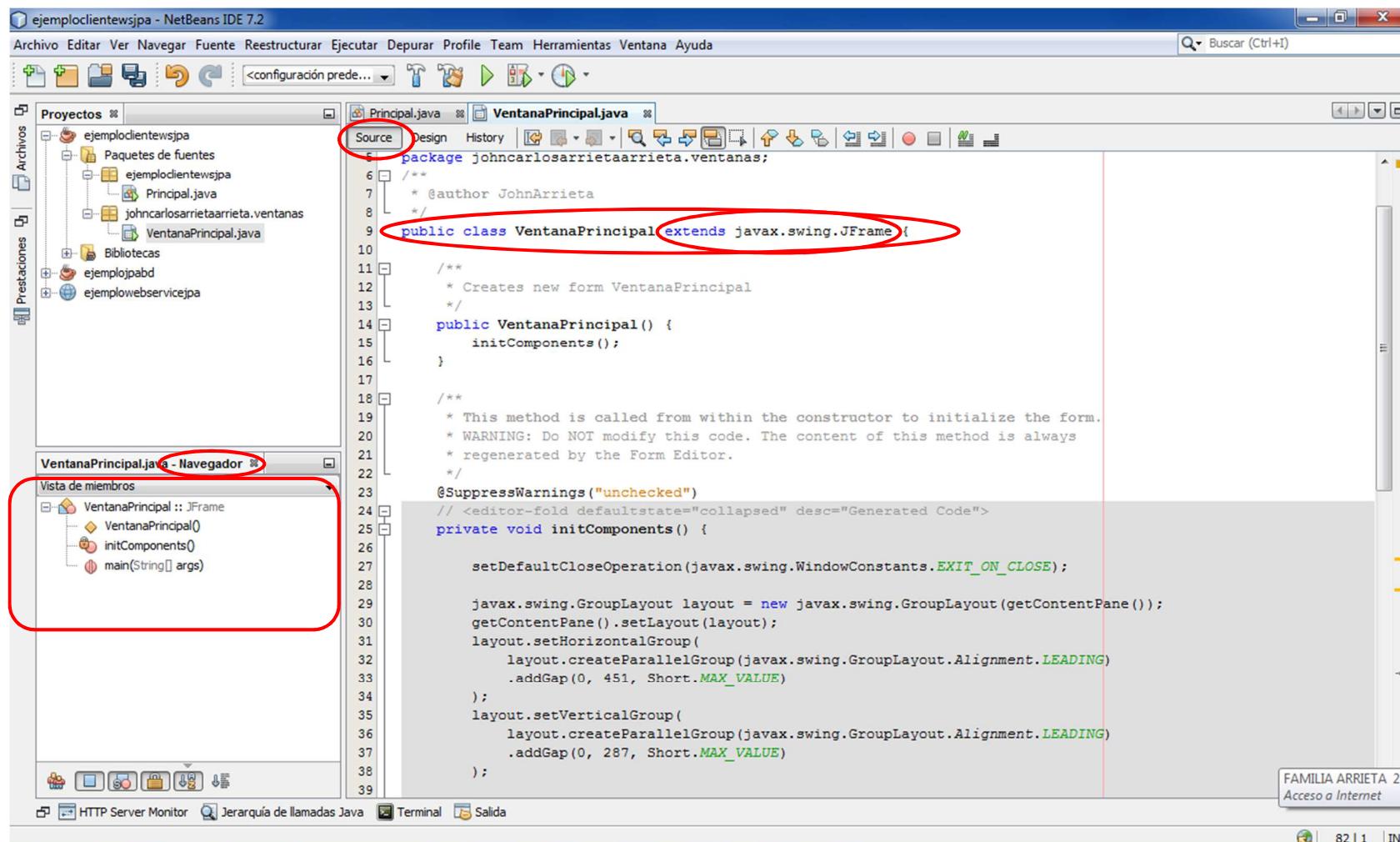


En la ventana de la página 55 debemos ingresar el nombre de la clase y elegir el paquete donde será guardado el archivo de la clase.

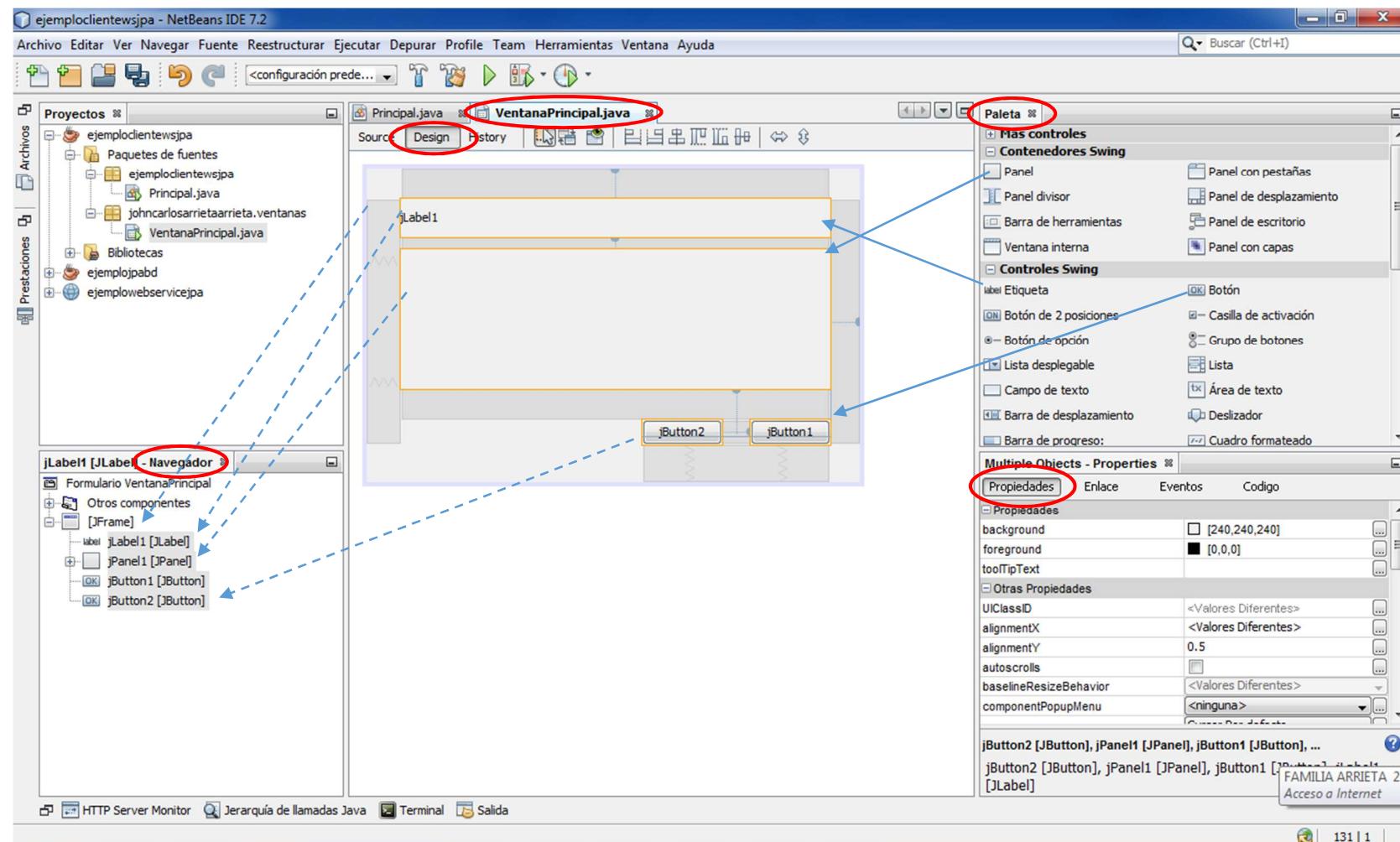
Luego de ingresar los datos de la clase, el IDE nos proporciona un diseñador para poder dar forma de manera visual al formulario de nuestra ventana, de una manera simple e intuitiva, solo necesitamos tener un bosquejo mental o físico del prototipo del diseño de nuestros formularios,



el resto de tareas se resumen en seleccionar un componente GUI (Interfaz Gráfica de Usuario), arrastrarlo hacia el panel de diseño y soltarlo, luego vamos al panel de propiedad de dicho componente GUI y configuraremos sus valores, el IDE va generando el código Java necesario.



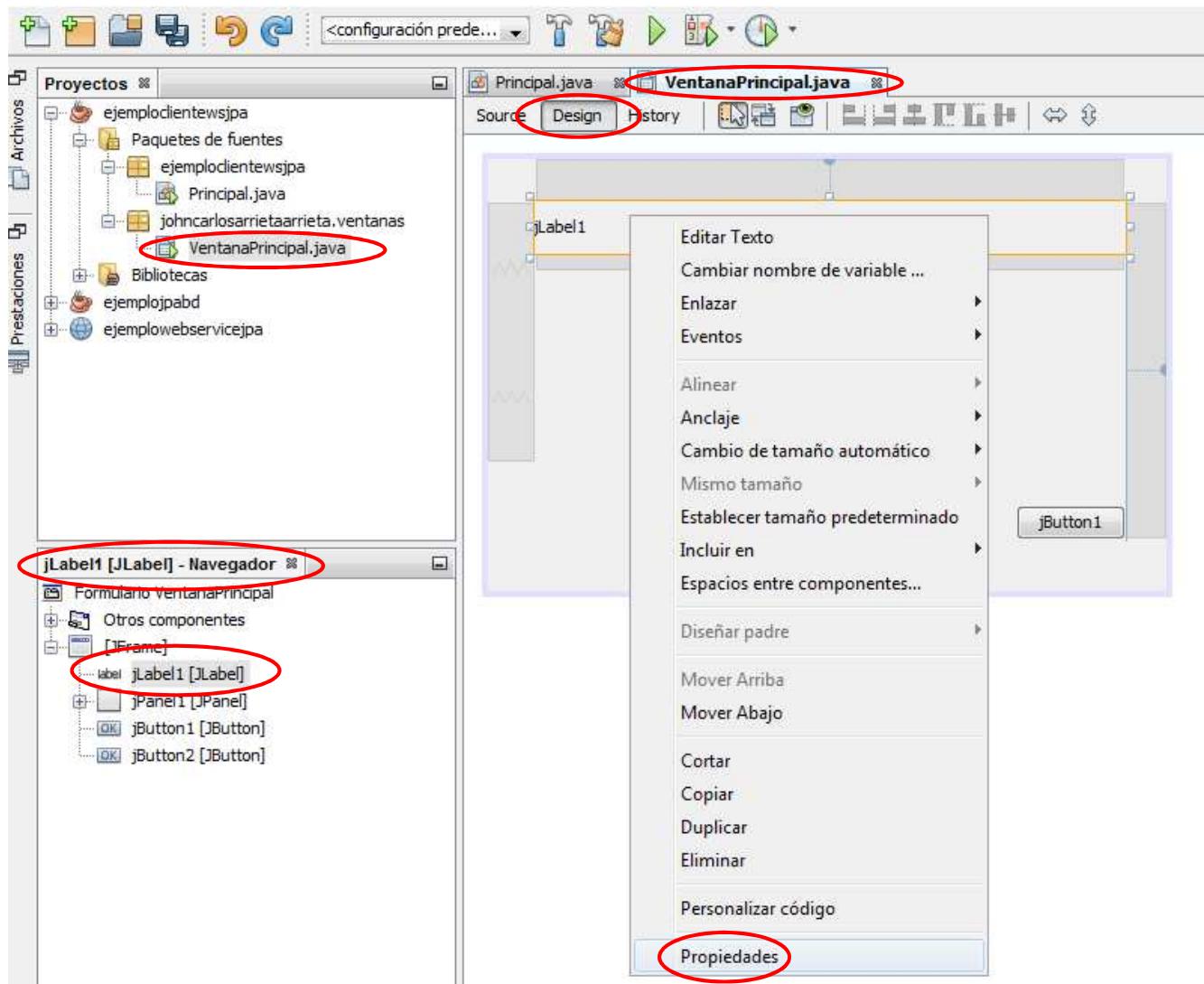
Desde el panel de Código o Source podemos observar y editar parte del código fuente Java generado desde el panel de diseño GUI, observemos que el Panel Navegador cambia de elementos cuando estamos en Diseño y cuando estamos en Edición de código



Lo primero que vamos a hacer es colocar en el Panel de diseño un **JLabel**, un **JPanel** y dos **JButton**, para ello vamos a la Paleta seleccionamos uno por uno y uno por uno los arrastramos y soltamos sobre la Ventana. Observar cómo se van agregando estos componentes al panel Navegador en forma Jerárquica, todos dentro del **JFrame** o ventana.

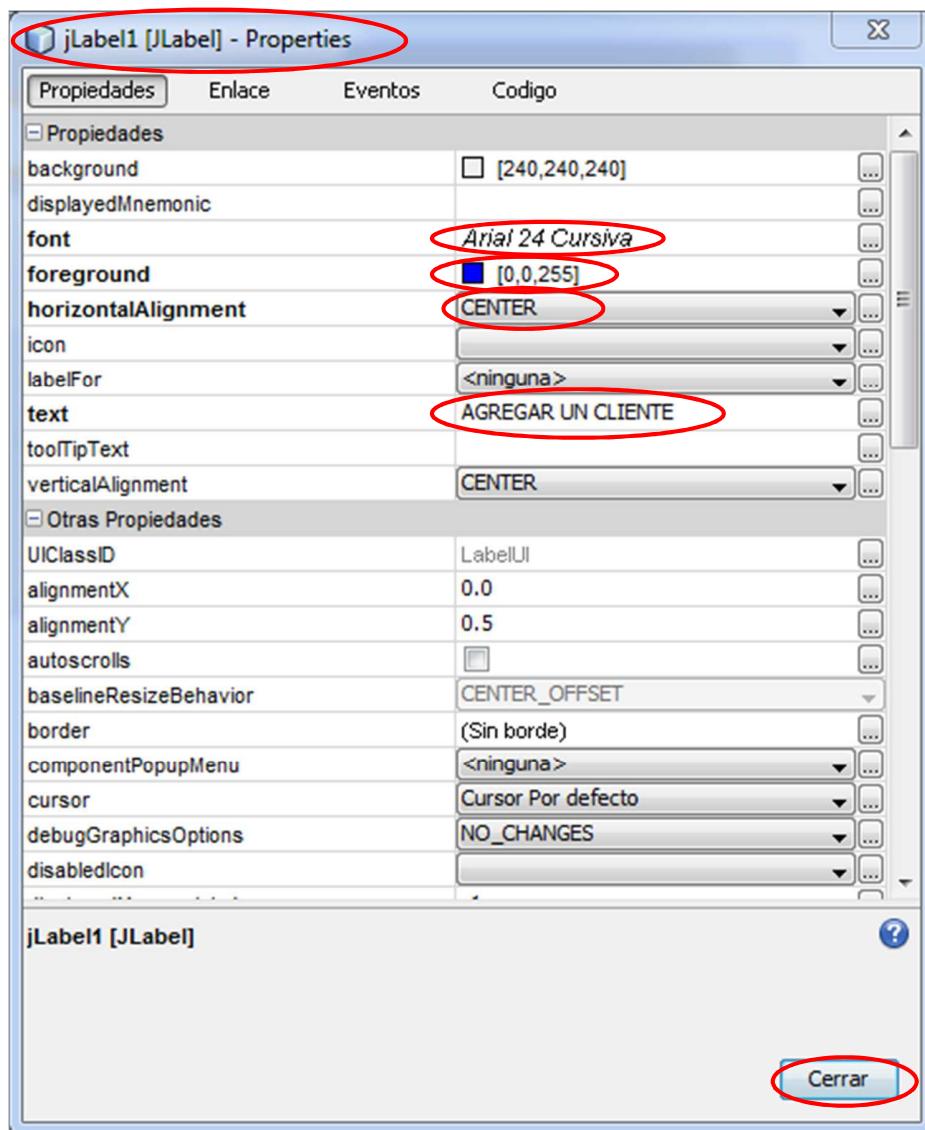


Procedemos a configurar o cambiar el valor pro defecto de sus propiedades, para ello iniciamos por ejemplo con la etiqueta de texto, la cual es una instancia de la clase **JLabel** del paquete **javax.swing**. Para cambiar las propiedades de un componente GUI primero se debe seleccionar desde el Panel de Diseño o desde el panel Navegador, luego se puede ir directamente al Panel de Propiedades al costado derecho del IDE, o simplemente se pulsa click derecho sobre el componente GUI y se selecciona la opción **Propiedades**.



Seguidamente el IDE nos presenta la ventana de propiedades del componente seleccionado, tal y como se puede apreciar en la siguiente imagen.

La ventana de Propiedades muestra dos columnas, la primera contiene el nombre de las propiedades y la segunda un campo para ingresar sus respectivos valores, observar que algunas propiedades ya tienen valores por defecto, el IDE se los coloca al momento de arrastrar y soltarlos sobre el panel de diseño.



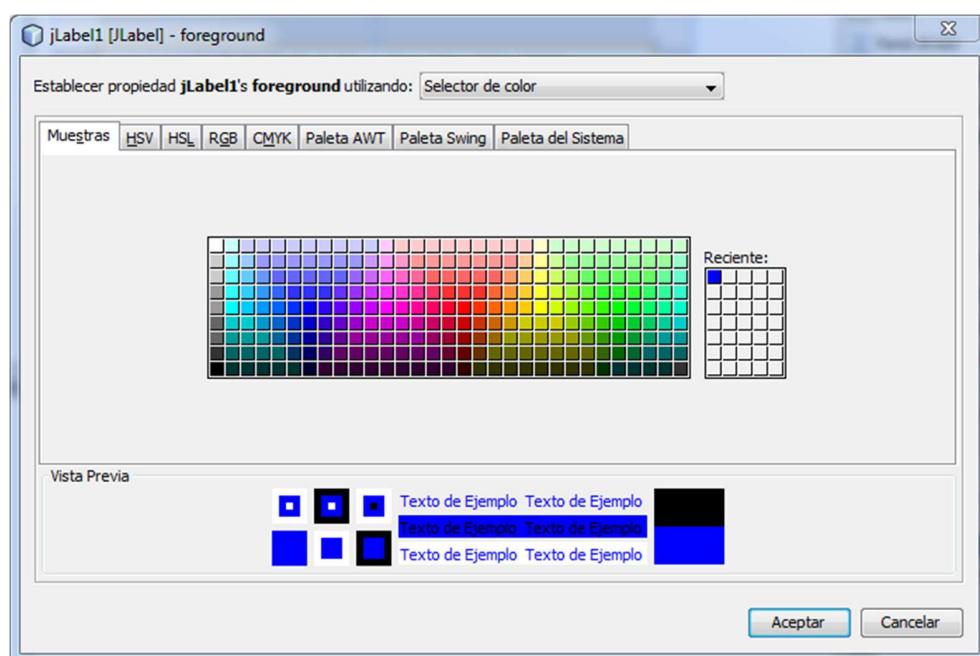
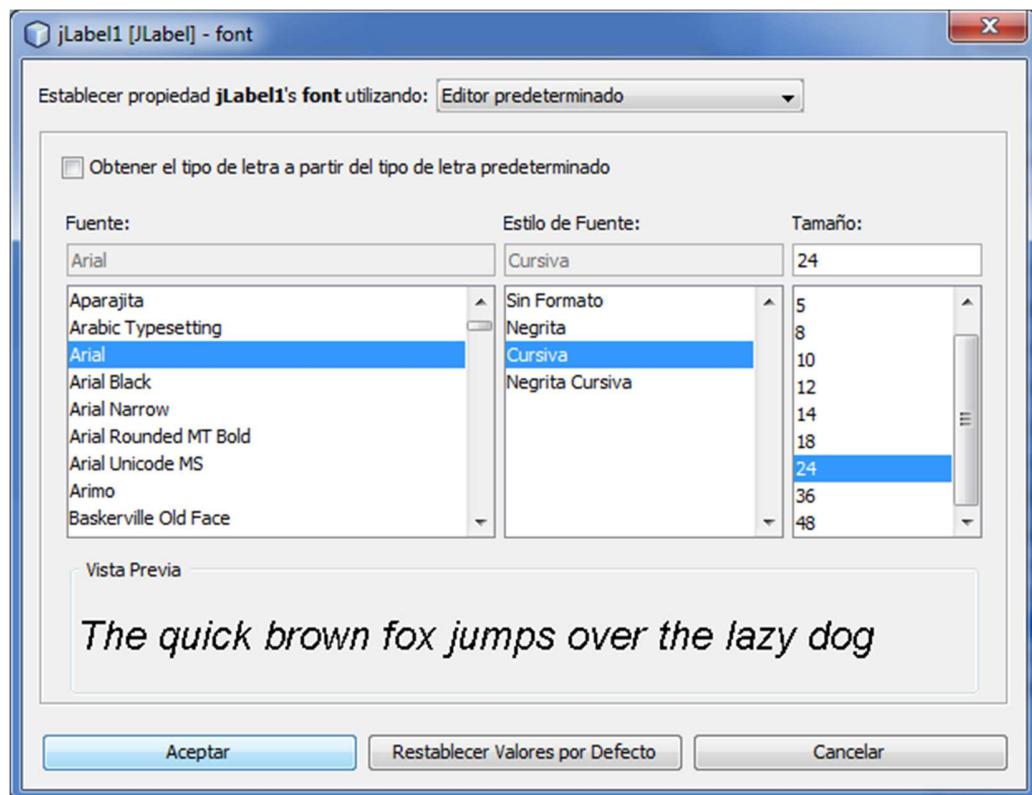
En este ejemplo podemos verificar que las propiedades corresponden a al componente **jLabel1** el cual es una instancia de la clase **JLabel**, al cual se cambiamos el valor de las siguientes propiedades:

- **Font:** Corresponde al Tipo de Letra que será presentada en el texto de la etiqueta (Arial 24 Cursiva)
- **Foreground:** Corresponde al color en el que se debe presentar el texto de la etiqueta (Azul claro)
- **HorizontalAlignment:** Corresponde a la alineación Horizontal del texto de la etiqueta (Centrado)
- **Text:** Corresponde al contenido del Texto que sea presentado dentro de la etiqueta (Agregar un cliente)

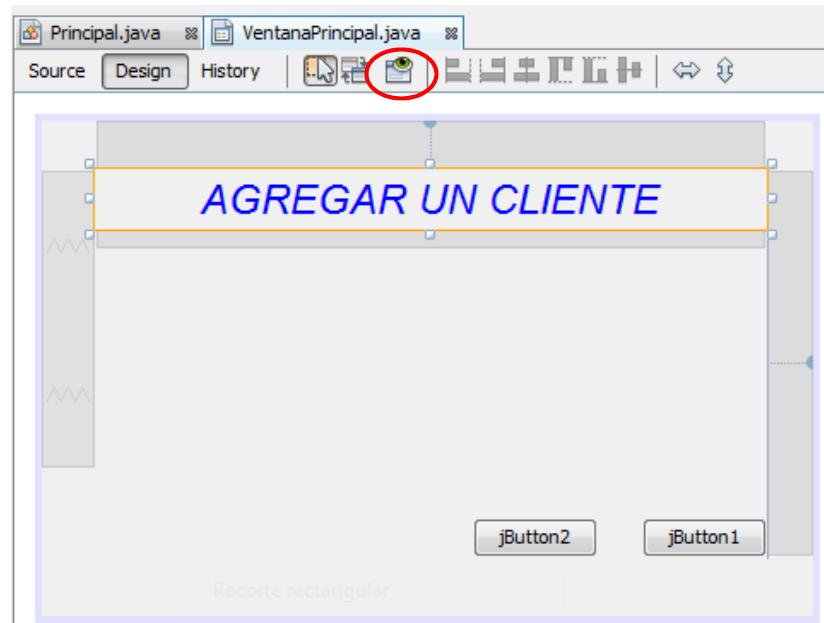
Como vemos existen muchas otras propiedades para el componente de tipo JLabel, las cuales podemos probar.



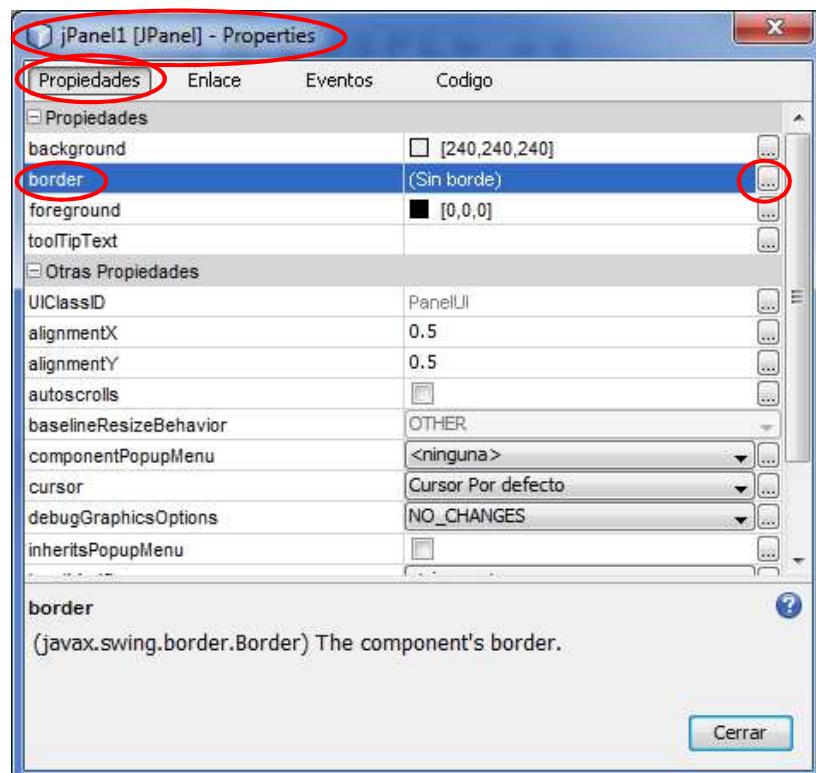
En las dos siguientes imágenes podemos observar las ventanas que aparecen cuando deseamos configurar las propiedades Font y Foreground respectivamente, en la primera podemos escoger características y estilos de letra, mientras que en la segunda podemos escoger el color de la letra entre una amplia gama de colores.

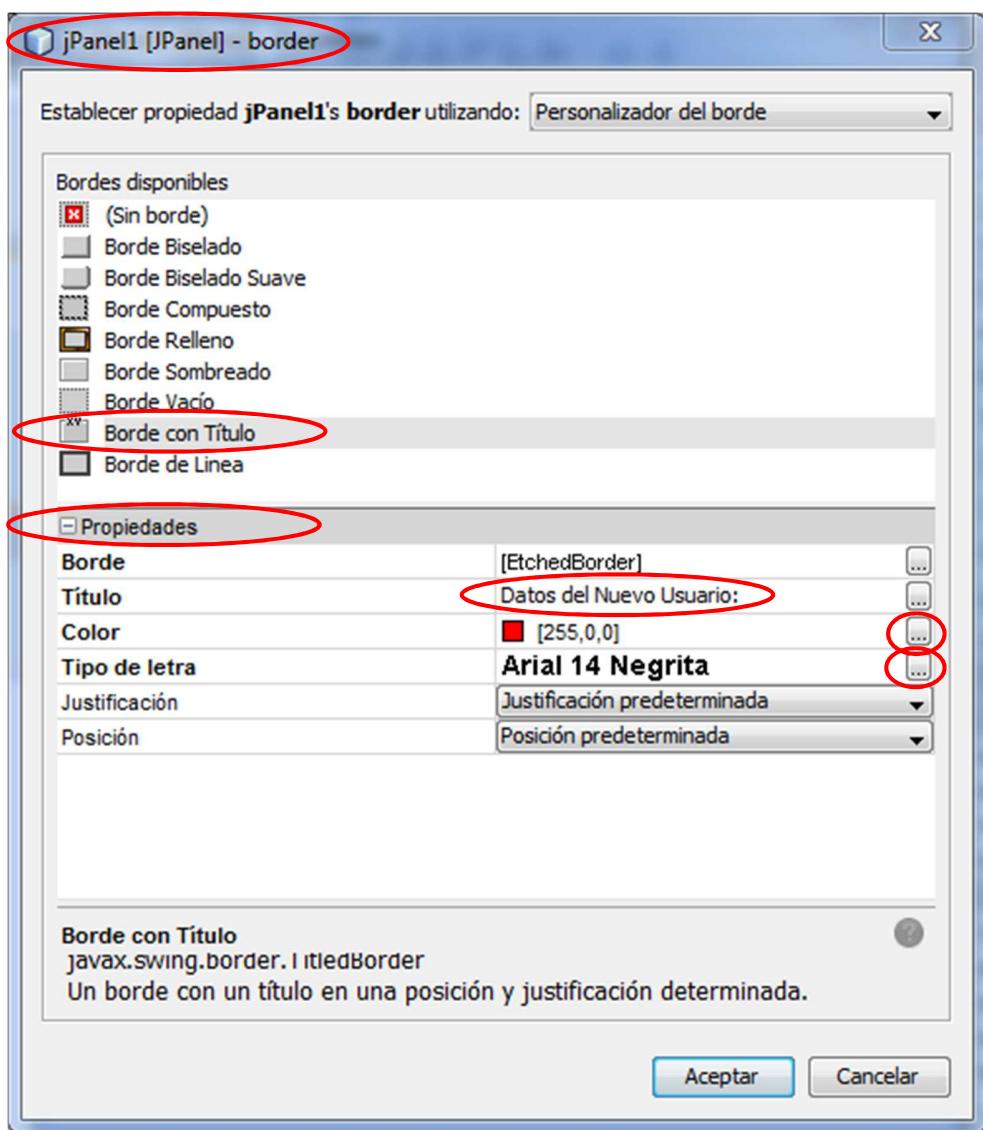


En la siguiente imagen podemos apreciar cómo fueron aplicados al diseño los cambios de las propiedades realizados anteriormente. Para tener una apreciación más real de cómo va quedando nuestro diseño, podemos dar clik en el icono de la ventana con un ojito.



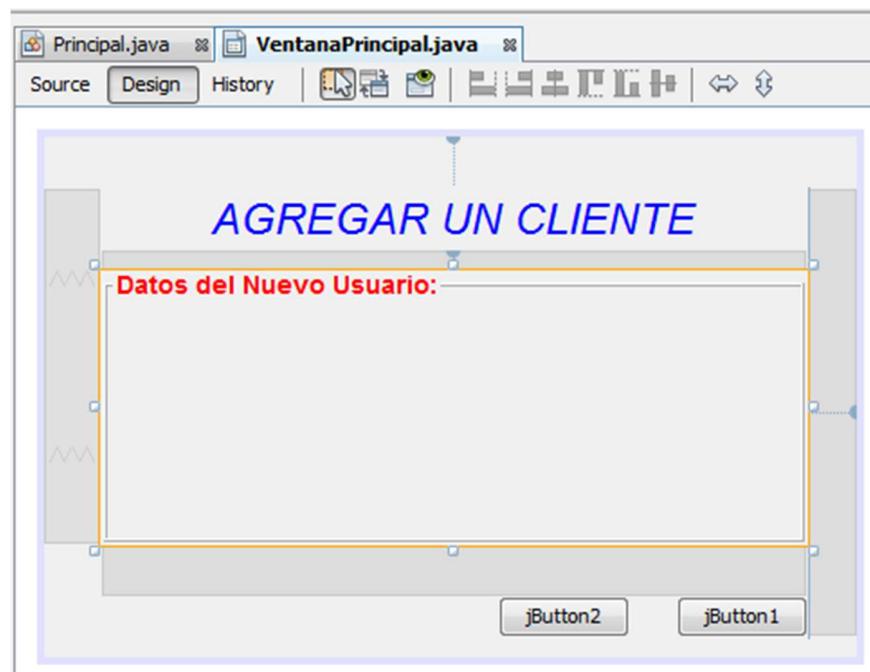
Configurar las propiedades del componente **jPanel1** de tipo **JPanel**:



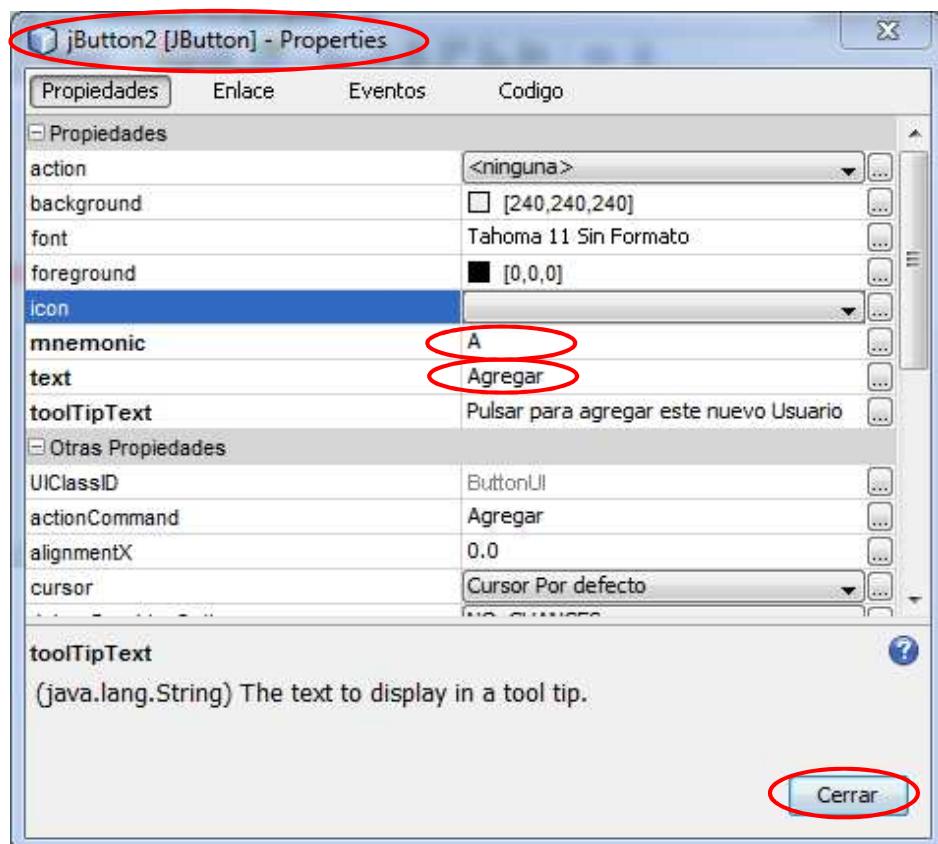


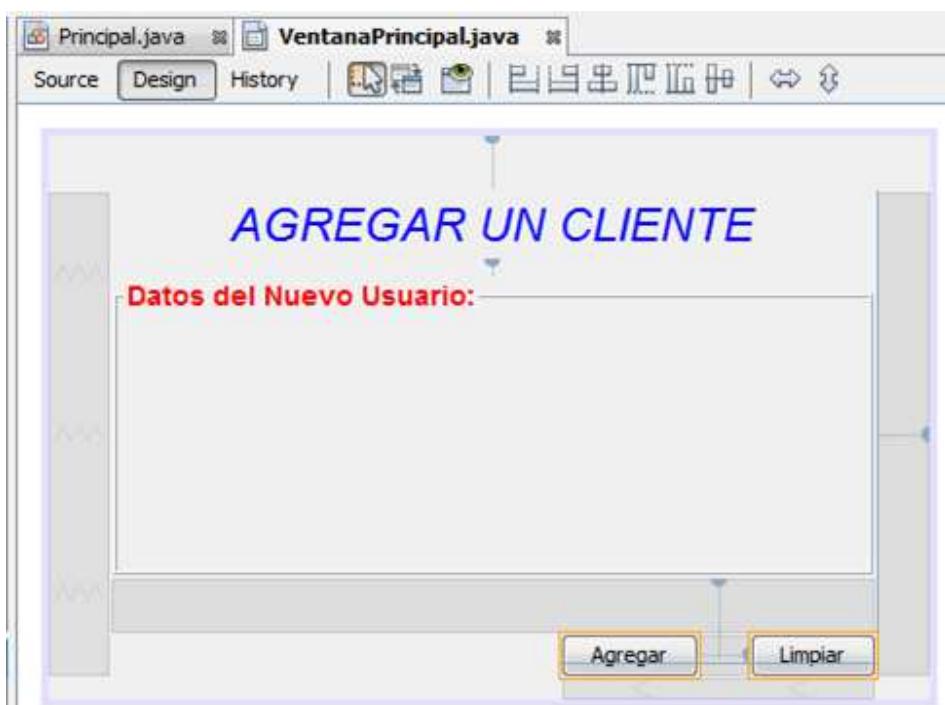
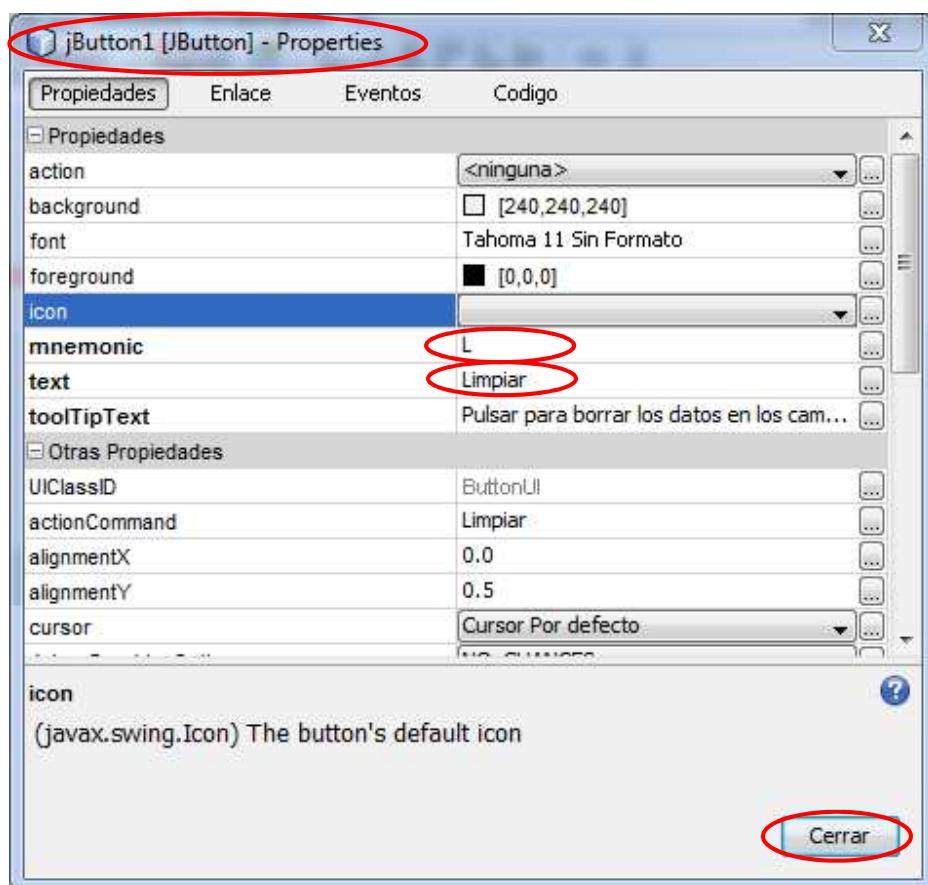
En este ejemplo solo vamos a cambiar la propiedad Border, la cual nos permite escoger uno de entre varios tipo de bordes para el JPanel, a mi me gusta mucho el Borde de Titulo, porque me da la posibilidad de colocar una legenda sobre el borde del JPanel, y a ese Borde de Titulo le puedo cambiar a su vez el borde por uno Sombreado. Observemos que contamos con un amplio conjunto de Bordes de los cuales podemos escoger uno, o formar uno personalizado mediante la combinación de varios. Las propiedades más importantes de un borde de título son:

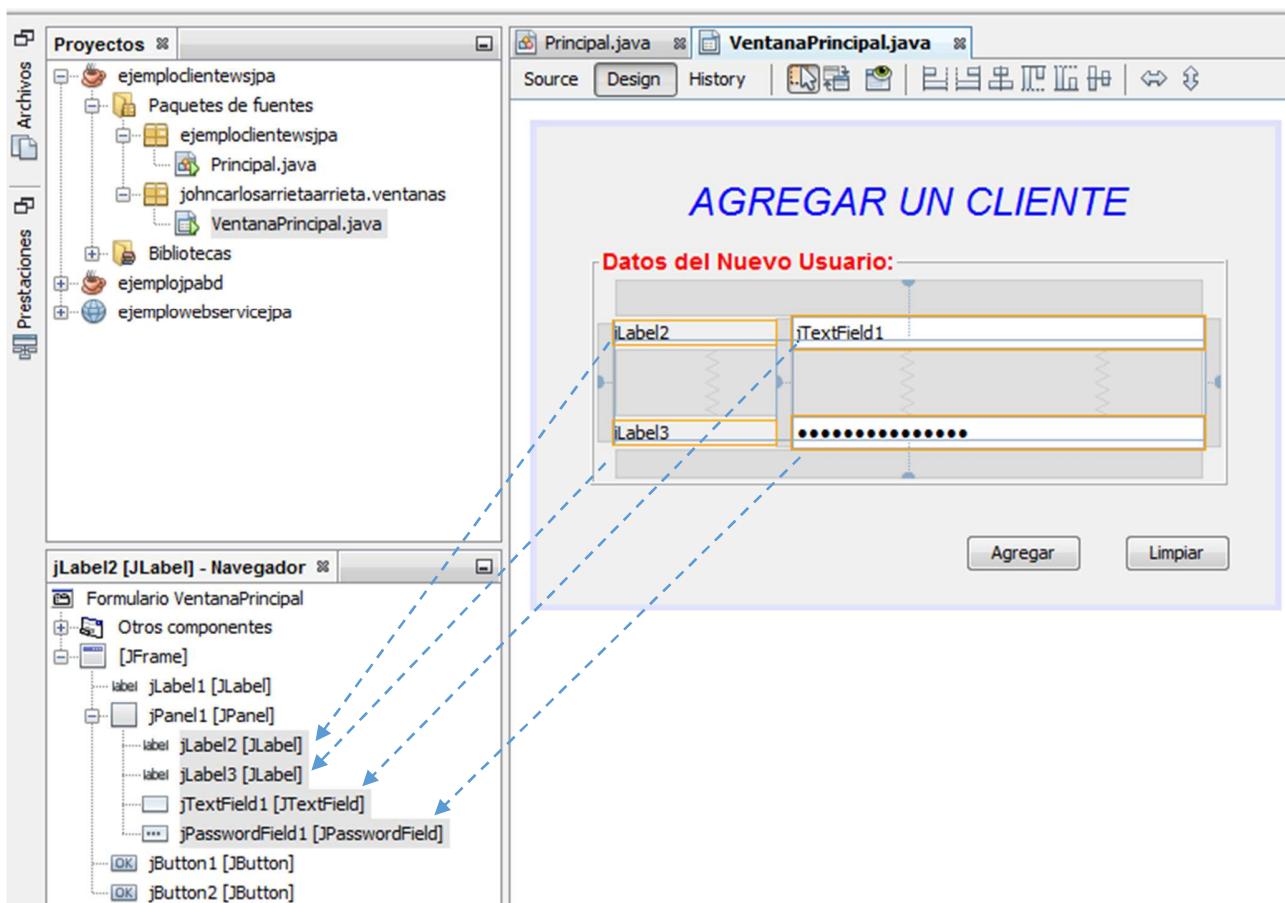
- El tipo de borde: sobre el cual se colocara el Titulo o legenda de texto.
- El Titulo: corresponde el texto que se presentara sobre el borde del JPanel (Datos del Nuevo Usuario:).
- El Color: Corresponde al color que tendrá el texto del título o legenda.
- El Tipo de letra: corresponde al estilo y tipo de letra del título o legenda (Arial 24 Negrita)
- La justificación: Corresponde a la posición horizontal del texto sobre el borde (Izquierda por defecto)



Volvemos al Panel de Diseño y nuevamente podemos apreciar aplicados sobre el **JPanel** los cambios realizados a las propiedades de este componente GUI. En los JButton la propiedad **Mnemonic** indica una alternativa del teclado

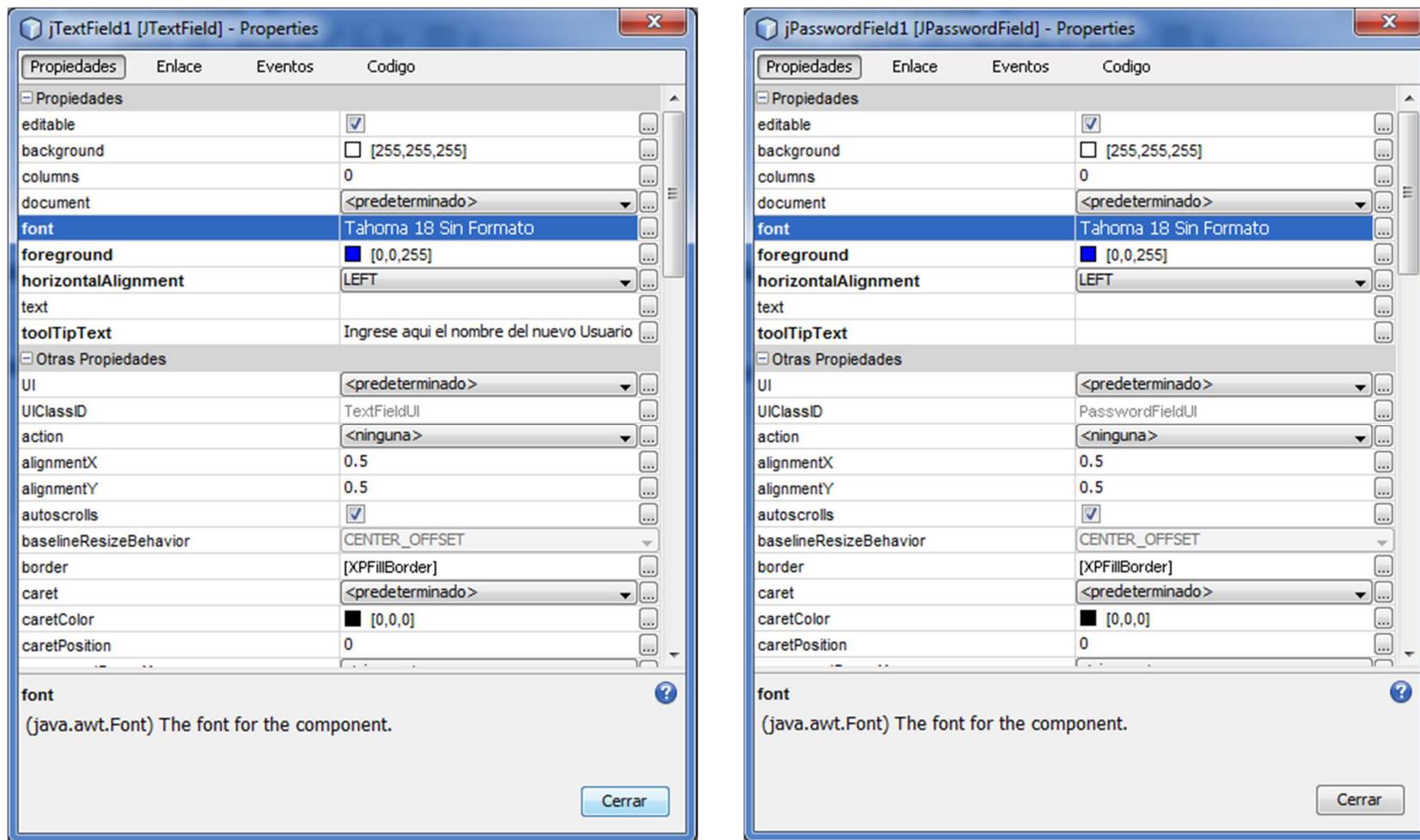




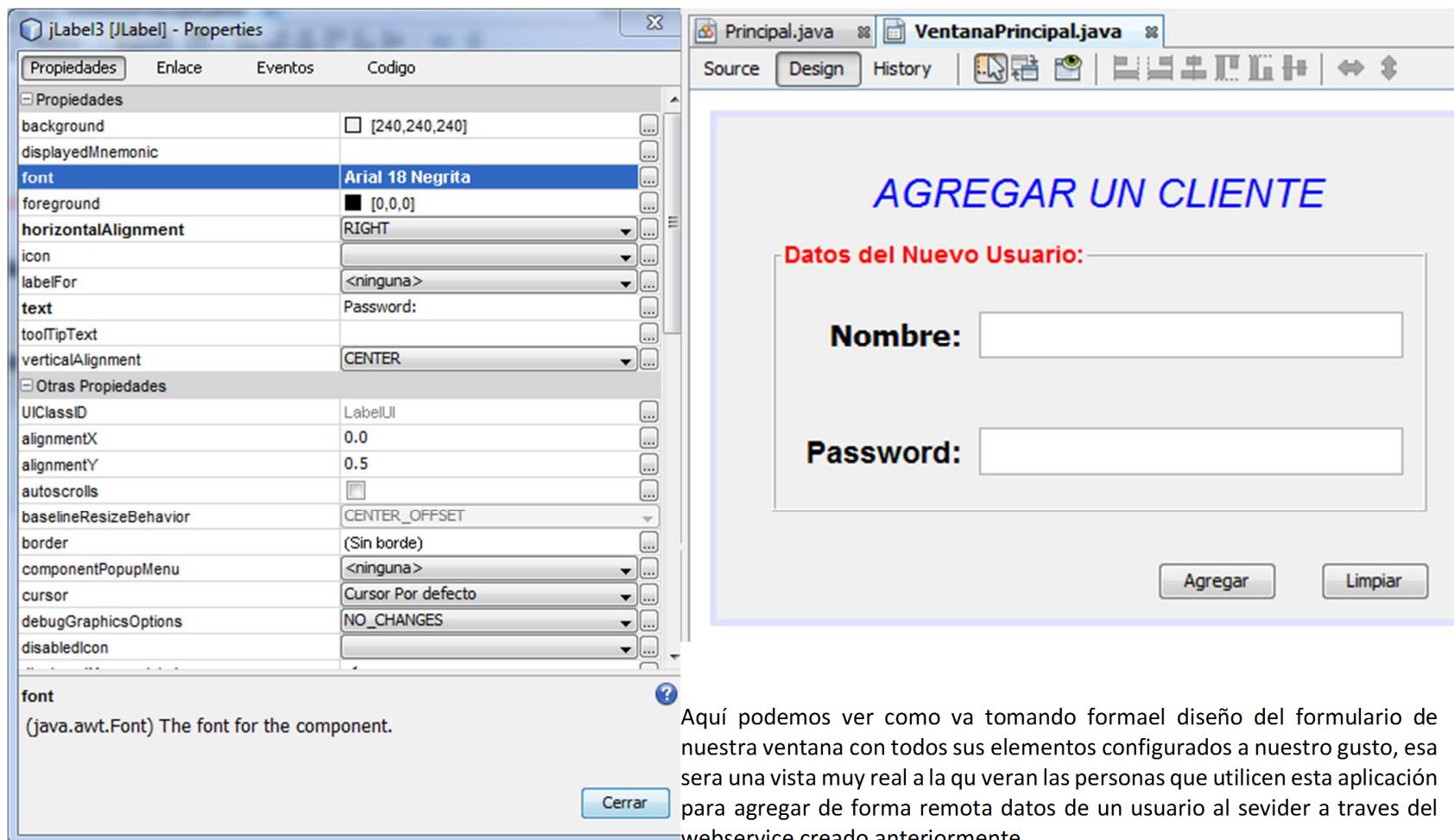


Ahora procedemos a agregar 4 nuevos componentes a la ventana pero dentro del componente **JPanel**, como es de suponerse estos 4 componentes fueron seleccionados de la Paleta de GUI, arrastrados y soltados sobre el componente **JPanel** que hemos colocado anteriormente dentro del Panel de Diseño de la **VentanaPrincipal**. Los componentes agregados son:

- Dos **JLabel**, los cuales como ya se deben imaginar se utilizan para mostrar pequeñas cantidades de texto, el cual será mostrado al usuario de la App con el objeto de dar una indicación o servir como etiqueta de texto para otros componentes.
- Un **JTextField** o Campo de Texto, el cual se utiliza como elemento de captura de texto (cualquier carácter que se pueda ingresar por el teclado) en forma lineal, es decir, en una sola y única línea de texto. En nuestro ejemplo este componente GUI campo de texto de tipo **JTextField** sera utilizado para que el usuario de la App pueda introducir el nombre del Usuario que desea agregar a la BD.
- Un **JPasswordField** o Campo de Contraseña, es similar al **JTextField** pero no permite visualizar el texto que se está escribiendo sobre dicho componente, en su lugar por cada carácter que sea tecleado sobre este campo, se mostrara por defecto el carácter \* Asterisco o el carácter . Punto, incluso podemos escoger cual carácter será el que se deba mostrar, para que no sea siempre . (punto) o \* (asterisco)



A ambos componentes les cambiamos las propiedades **Font** a Tahoma de tamaño 18 y sin negrita, subrayado ni cursiva, **Foreground** a color azul **HorizontalAlignment** a izquierdo y **Text** a vacío. La propiedad **ToolTipText** permite colocar una breve ayuda textual la cual aparece sobre el componente por 2 segundos cuando se coloca el puntero del ratón sobre dicho componente.



En este ejemplo se configuran las propiedades de las etiquetas **JLabel**, a **Font** se le coloco el valor Arial de tamaño 18 y negrita, **Foreground** a color negro **HorizontalAlignment** a derecho y **Text** a Pasword: y para el otro JLabel tex toma el valor de Nombre:.



## Paso 19: DARLE NOMBRE A LAS VARIABLES JAVA DE LOS COMPONENTES GUI.

Cuando utilizamos el diseñador de GUI del IDE Netbeans este nos genera de forma automática todo el código necesario para poder trabajar con dichos componentes, por lo tanto cada componente ya sea Ventana, Panel, Borde, Etiqueta, Campo de Texto, Botón, Menú, ícono, etc, le corresponde una variable dentro del código Java, Netbeans va colocando el nombre de cada variable de la siguiente forma:

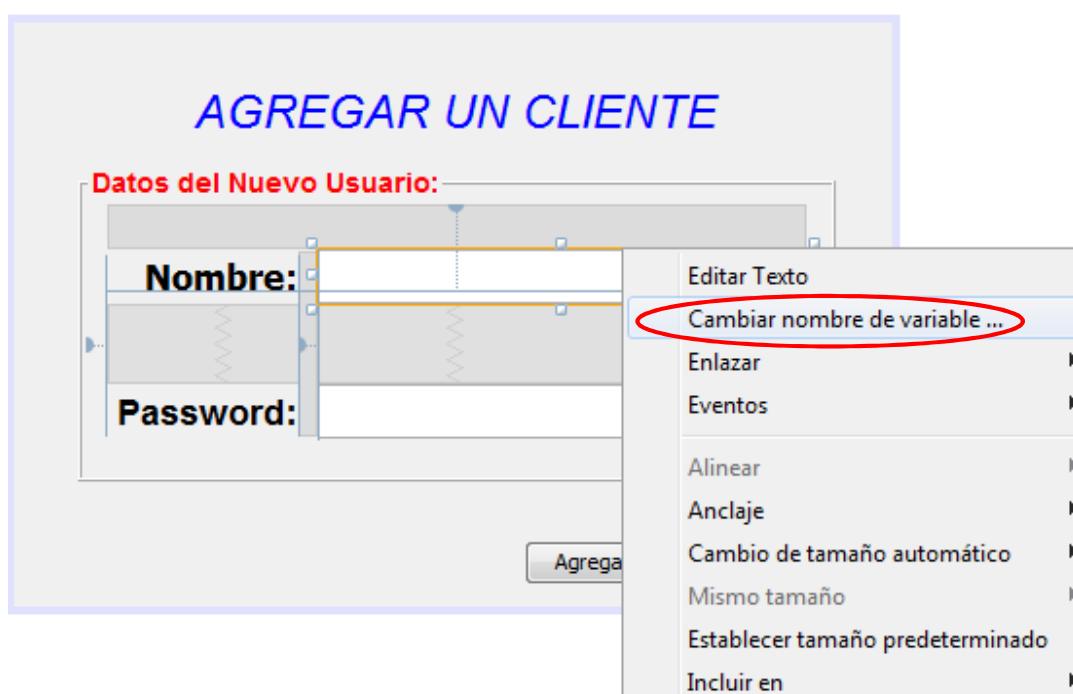
Si el componente es una etiqueta de texto y es la primera etiqueta que se arrastró y soltó sobre el panel de diseño, entonces de forma automática en el código fuente Java se declara una variable así:

`JLabel jLabel1;` donde `JLabel` es el tipo o nombre de la clase `JLabel` que se encuentra en el paquete `javax.swing` y `jLabel1` es el nombre de la variable.

Si por ejemplo, el diseño contiene otras etiquetas de texto, entonces sus variables serían:

```
JLabel jLabel2;  
JLabel jLabel3;  
JLabel jLabel4;
```

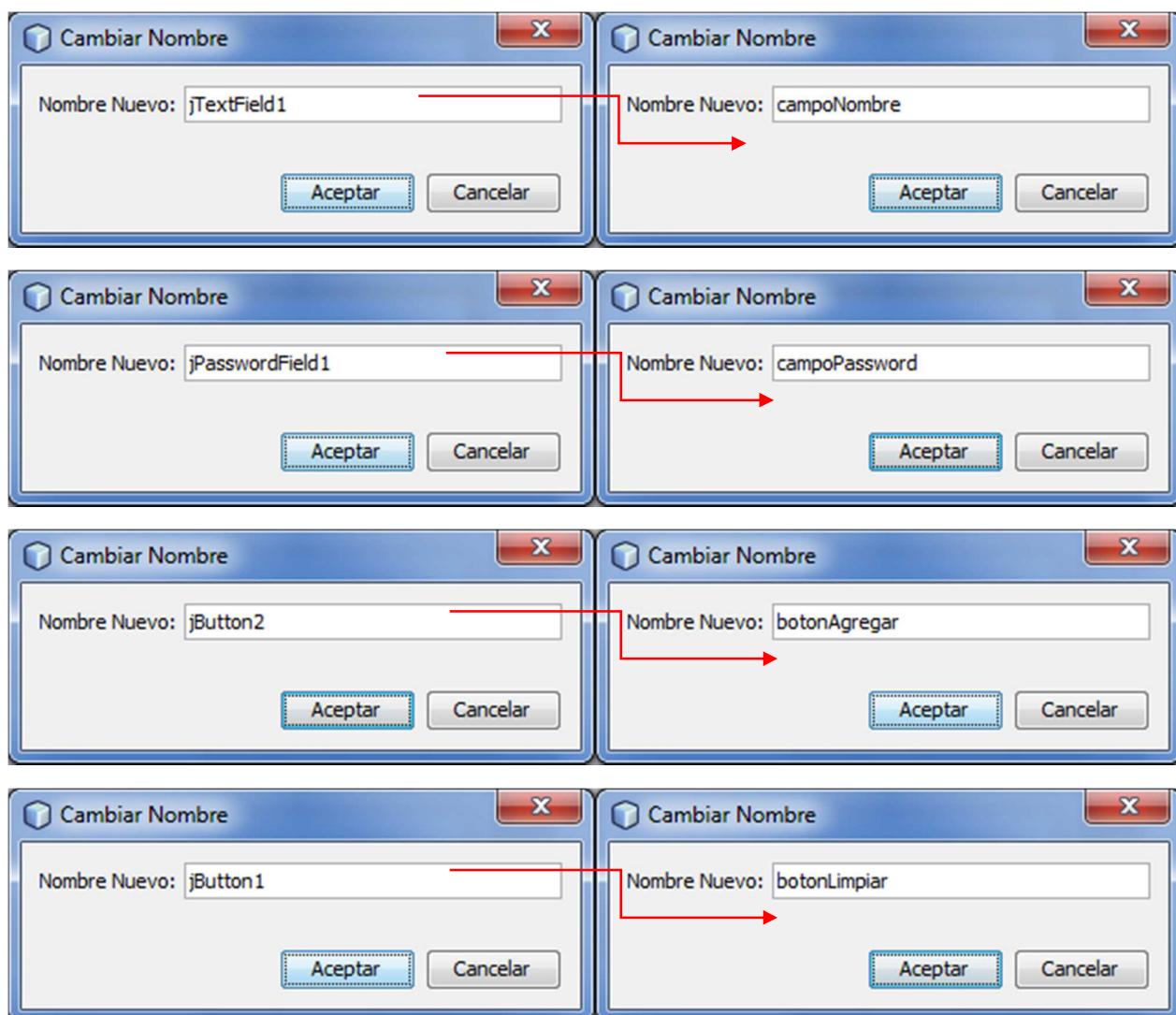
Y así sucesivamente, es igual para otros tipos de componentes, por ejemplo `JButton jButton1;` sería la declaración para la variable del primer botón que fue agregado al diseño y `JButton jButton2;` serial para el segundo, etc. No es buena idea dejar el nombre de algunas variables de esta manera, pues cuando tengamos que utilizarlas el código no podemos confundir al no saber cual variable es la que corresponde a x componente, para solucionarlo el IDE nos permite cambiar manualmente el nombre de las variables de cada uno de los componentes deseados, haciendo click derecho sobre el componente y seleccionado la opción Cambiar nombre de Variable:





Nos aparece una ventanita con el nombre actual del componente seleccionado, dato que podemos cambiar por un nombre más legible y fácil de recordar, por ejemplo, es costumbre mía colocar campo al inicio del nombre de todos los campos de texto, seguido de una palabra que represente el valor que se debe ingresar en dicho campo, por ejemplo, para el caso del campo donde se debe ingresar el nombre del usuario, su variable la llamaría campoNombre.

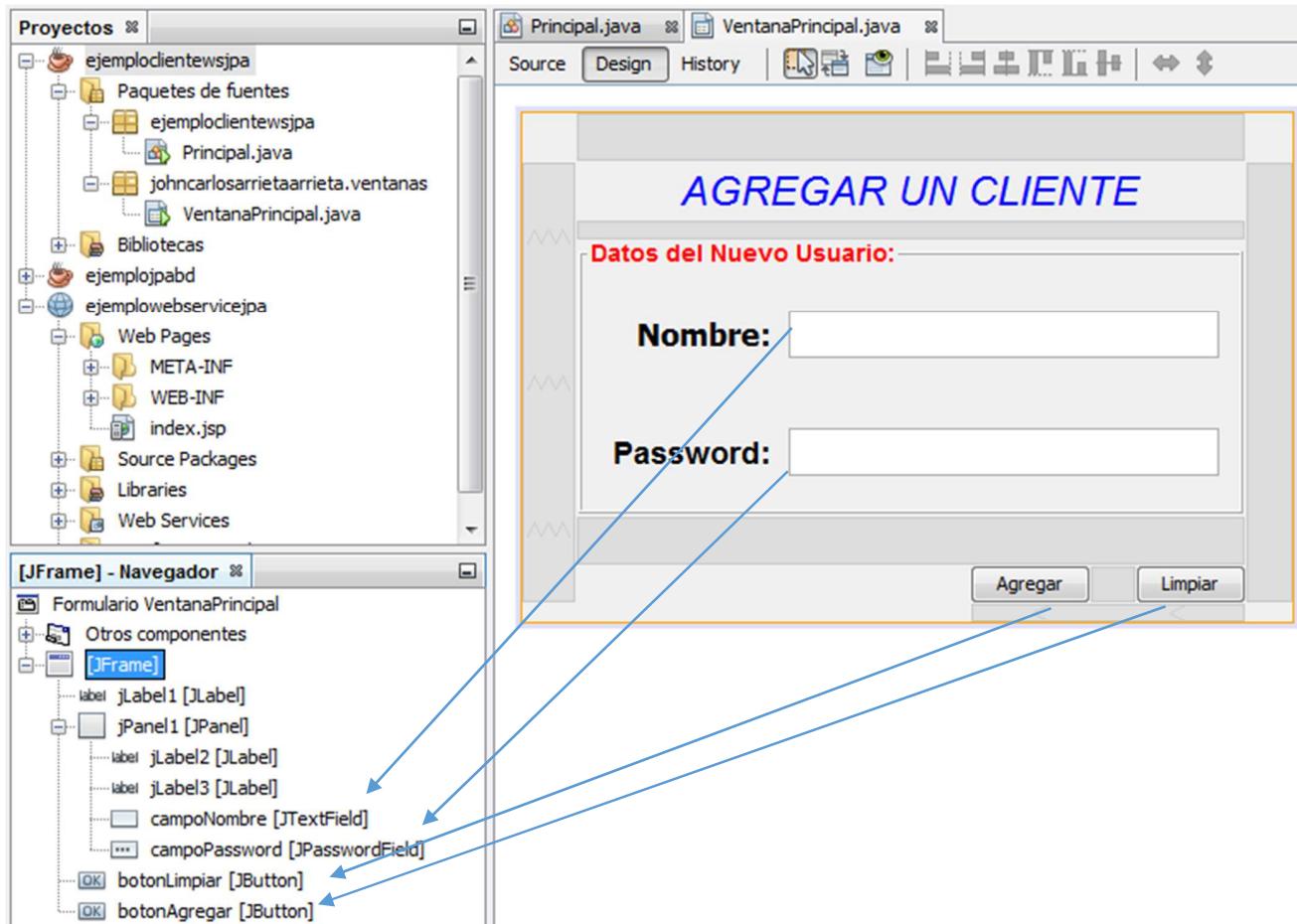
Nota: Recordar que los nombres de las Clases, Metodos y Variables Java no pueden iniciar con número, pero si pueden tener numero en cualquier otra parte del nombre, no puedes tener espacios en blanco, ni pueden tener alguno de los siguientes caracteres: `\|\"@·#%&/()=??.*+`^[]{}<,>;,` también se recomienda que si el nombre esta compuesto por mas de una palabra, la primera palabra esta en minúscula, la siguiente este pegada a anterior con la primera letra en Mayúscula y así sucesivamente, para el caso de nombres de las clases, todas las palabras incluyendo la primera deben tener la primera letra en Mayúscula.



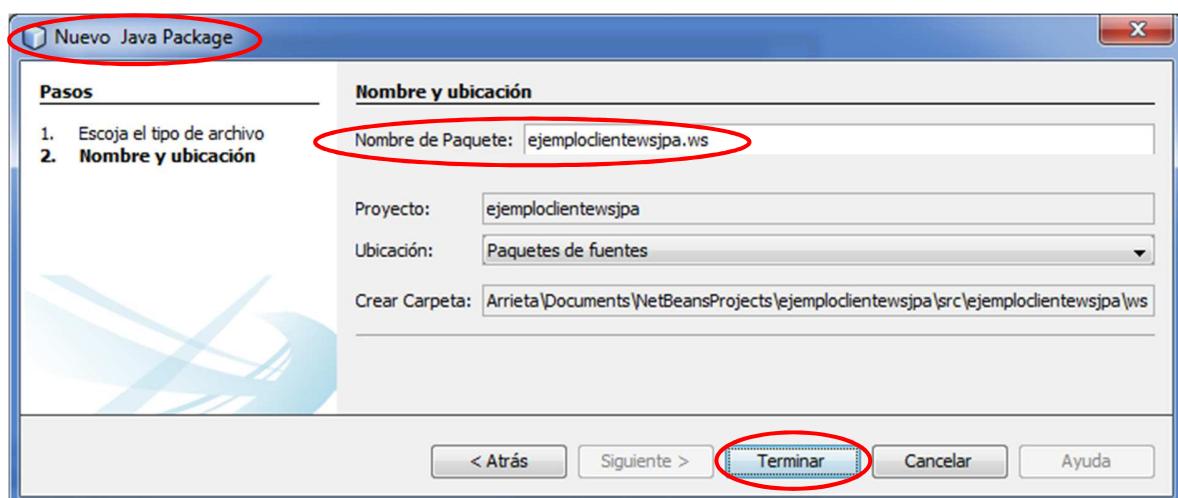
Para verificar el cambio de nombres podemos ir al panel de código (Source) y observar en la parte inferior



También se puede verificar dicho cambio desde el Panel Navegador



## Paso 20: INSERTAR EL CÓDIGO DE CONEXIÓN AL WEBSERVICE WSusuario

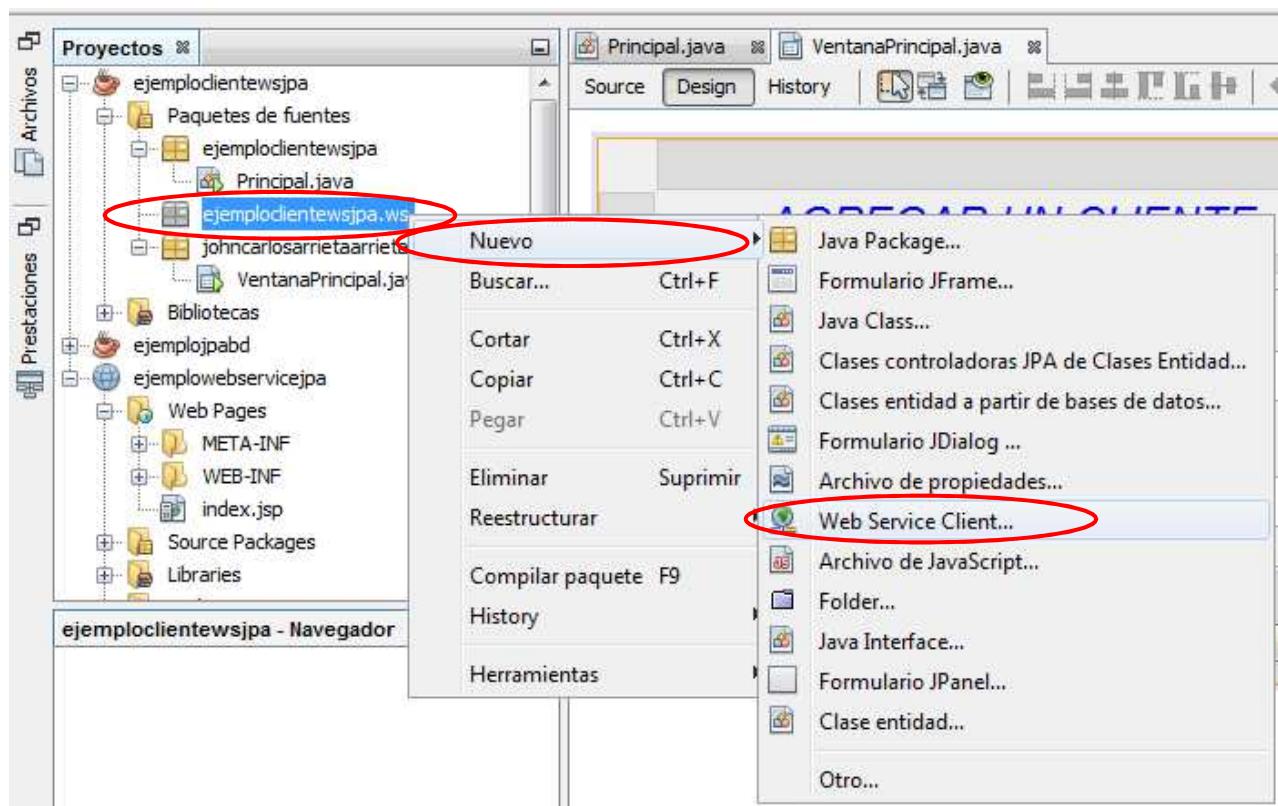




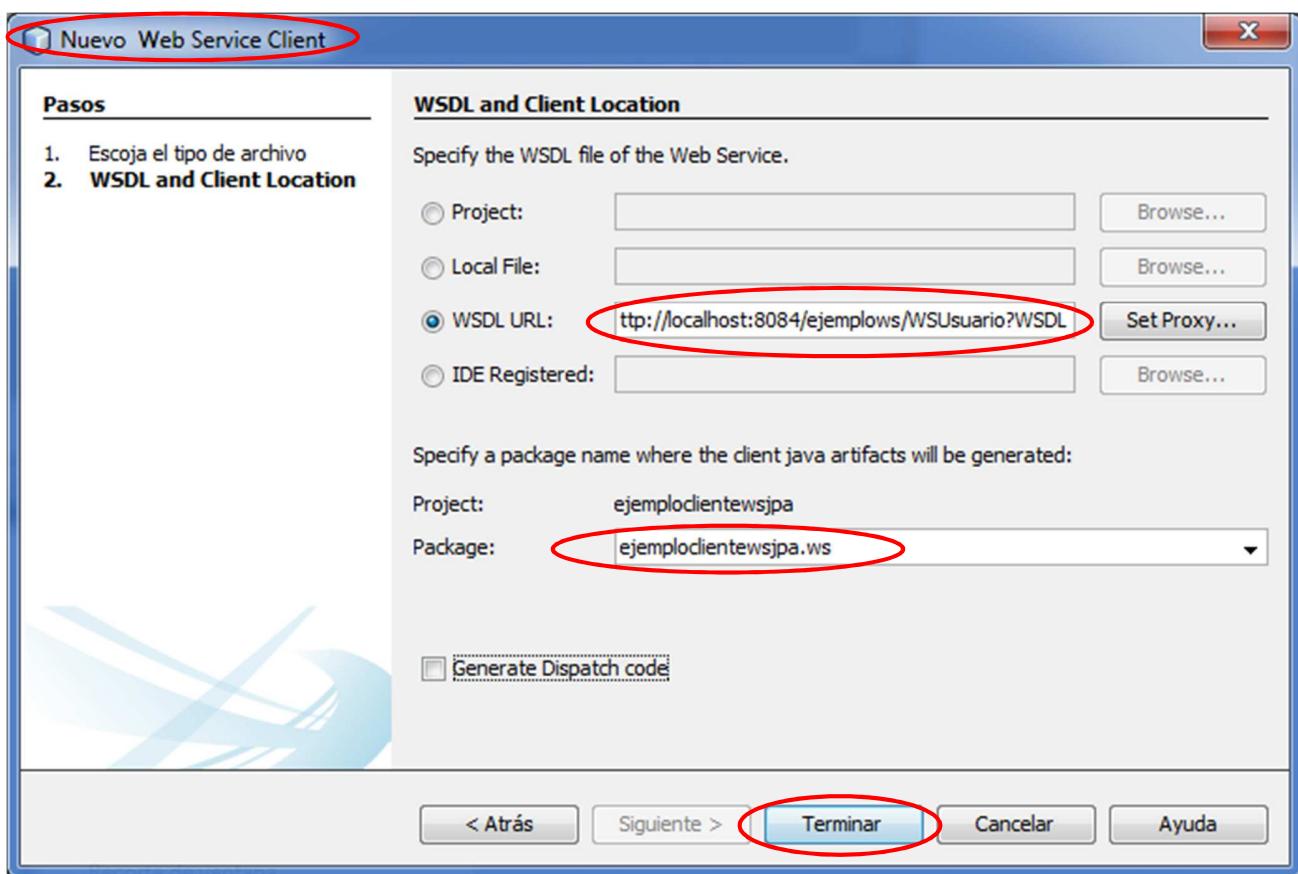
Terminado el diseño de la GUI, procedemos a generar el código necesario para poder interpretar los elementos que conforman el WebService WSusuario, así como la forma de poder conectarnos a él y consumir o invocar su operación agregarUsuario.

Con el ánimo de organizar bien el código de nuestra aplicación cliente, lo más correcto es crear un paquete donde se guardaran las clases generadas para el cliente servicio web, este paquete lo llamaremos ws, como se aprecia en la imagen anterior.

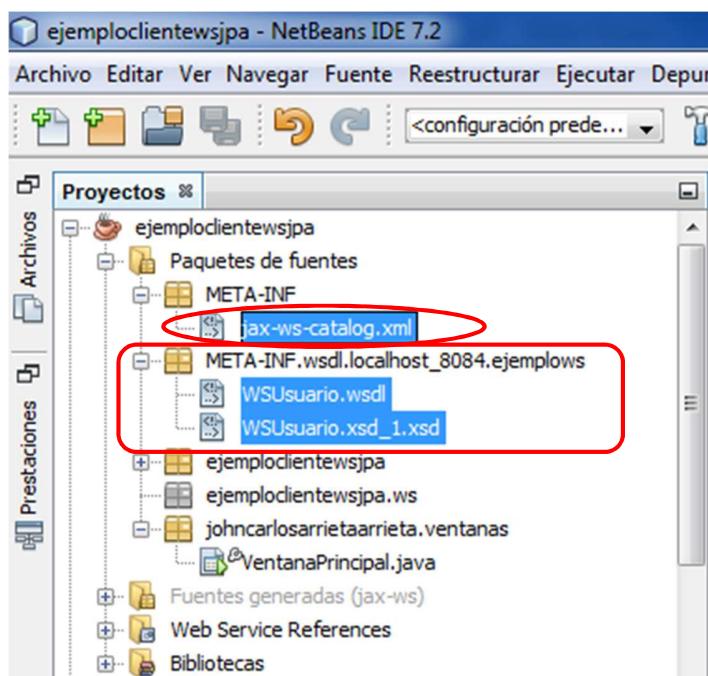
Ahora procedemos a utilizar el generador de código de Netbeans para solicitarle que genere el código de nuestro Cliente WebService, para ello debeos seguir los pasos que se muestran en la siguiente imagen.



Al igual que como ocurrió con los otros elementos construidos en nuestras aplicaciones anteriores, por ejemplo la conexión a la BD, las clases DAO, las clases Entidades, la clase WebServices, las Operaciones de Web, la clase Principal, la clase VentanaPrincipal, etc cuyos códigos fueron generados un 90% de forma automática por el propio IDE, para el caso de los Clientes de WebServices no podía ser diferente, el IDE nos genera todos los archivos con código fuente necesarios para poder conectarnos de forma simple a nuestro WebServices, solo tenemos que ingresar la dirección su URL correctamente en el campo WSDL URL de la siguiente ventana que nos presenta el IDE, la URL del WS que desarrollamos anteriormente es <http://localhost:8084/ejemplows/WSUsuario?WSDL>

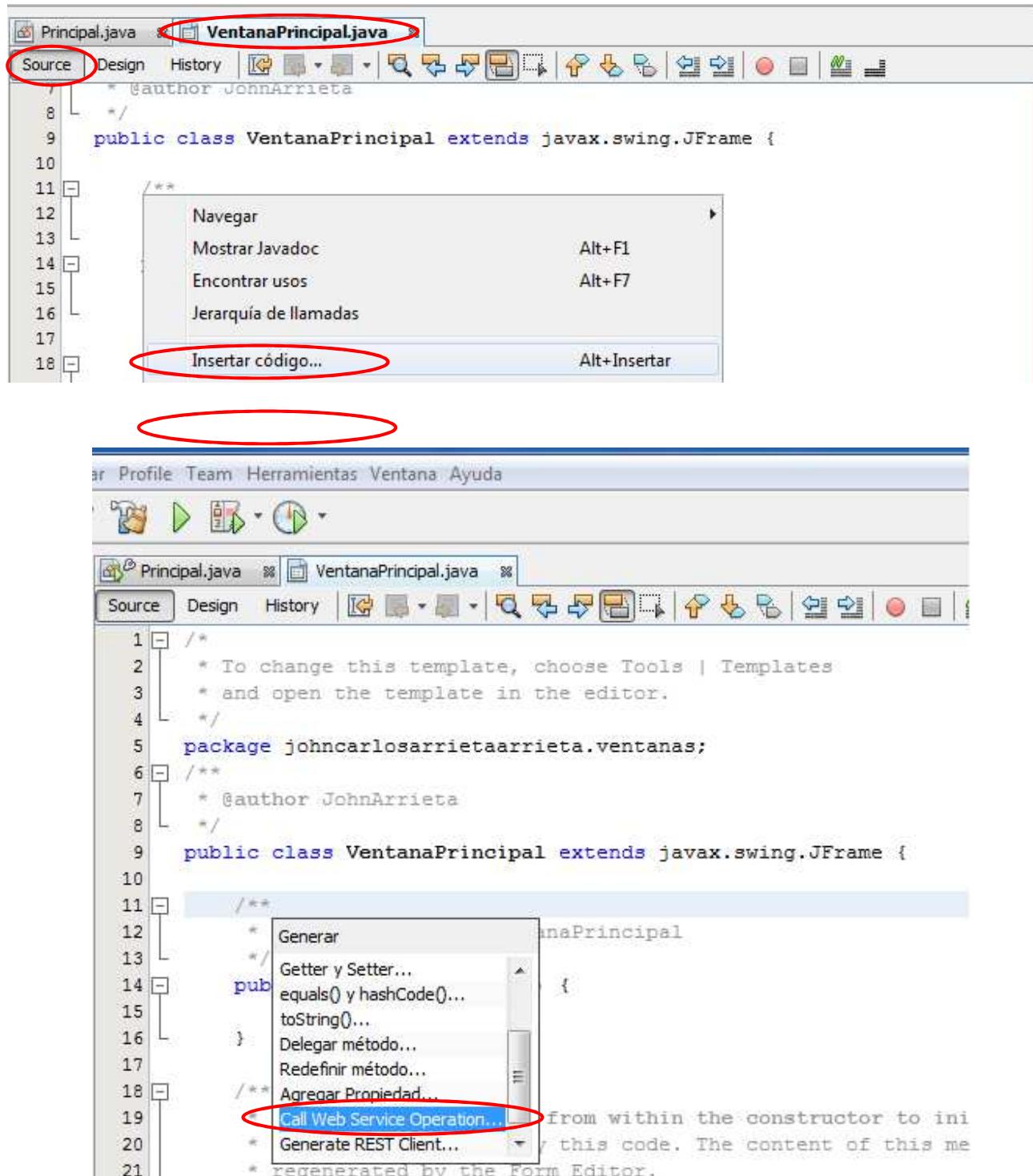


Podemos observar los archivos que ha generado el IDE los cuales contiene información de conexión al WS



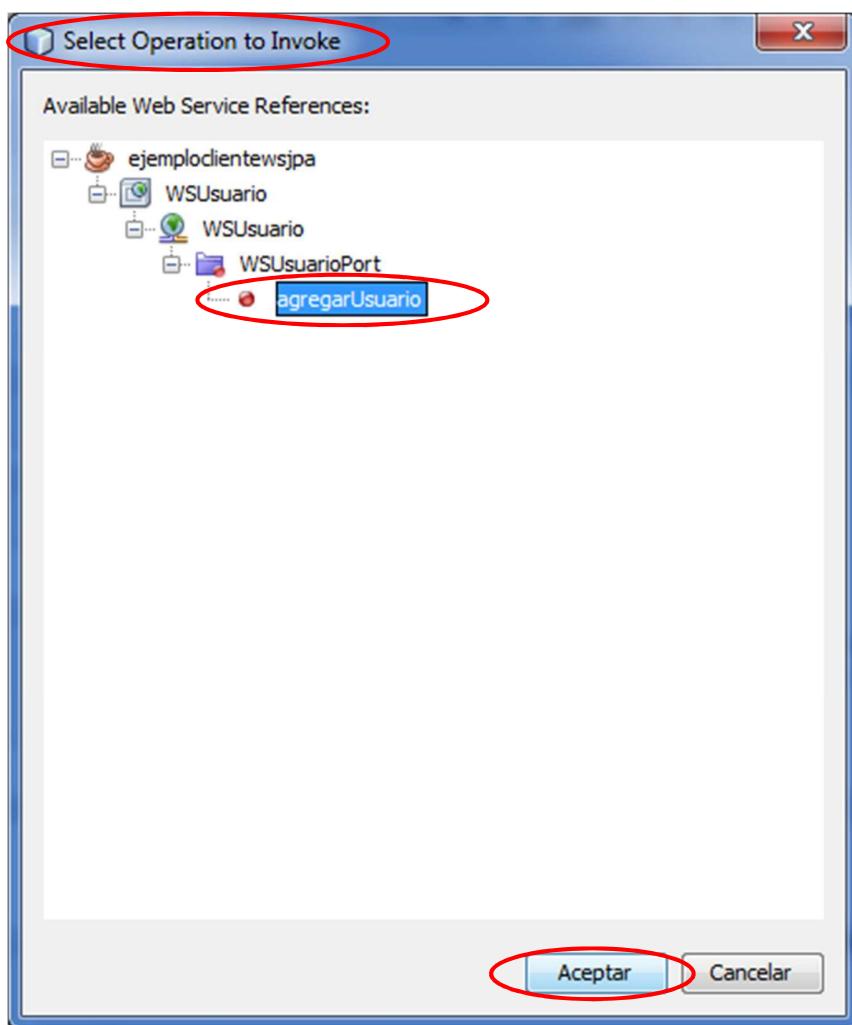
Ahora procedemos a seleccionar que operación del WebServices vamos a invocar, esto se hace igualmente utilizando el generador de código de NetBeans, tal y como se aprecia en las siguientes imágenes:

Nos ubicamos en cualquier lugar del código fuente de la Ventana principal y pulsamos click derecho





Nos aparece la siguiente ventana llamada Seleccione la operación a Invocar, en la cual claramente se aprecia un árbol que muestra todos los WS a los que nos hemos conectado desde esta aplicación Cliente, en este ejemplo solo se presenta el WS WSUsuario porque es el único que hemos hecho y al que no hemos conectado previamente.



En caso de que hubiesen mas WB, estos deberían aparecer dentro del árbol, cada uno con un ITEM con su nombre, dentro de este ítem, otro Item nuevamente con su nombre, luego dentro de este un Item con su nombre y la palabra Port, dentro de este aparecería la lista de operaciones que ofrecen dicho servicio Web, esta es la opción que vamos a seleccionar, de esta manera el IDE procesa el archivo XML que describe el WS y nuevamente genera el código necesario para poder invocar dichas operaciones desde nuestra aplicación Cliente WS, enviarle los parámetros que requieran y obtener su valor de retorno si es necesario.





## Paso 21: INVOCAR LAS OPERACIONES WEB DEL WEBSERVICES

A este punto solo nos queda escribir unas pocas líneas de código (gracias a dios, vamos a codificar) que nos permitan invocar el método anteriormente generado y pasarle los datos ingresados desde el formulario de la ventana que ya hemos diseñado. Para ellos debemos agregar un Oyente o escuchador de eventos de click sobre el componente botónAgregar.

The screenshot shows the NetBeans IDE interface with two tabs: 'Principal.java' and 'VentanaPrincipal.java'. The 'Design' tab is selected in the top bar, and the 'Source' tab is selected in the bottom bar. In the design view, a window titled 'AGREGAR UN CLIENTE' is displayed with fields for 'Nombre' and 'Password'. A context menu is open over the 'Agregar' button, with 'Eventos' selected. A submenu shows 'actionPerformed' under the 'Action' category, which is highlighted with a red circle. In the source code view, the 'Generated Code' section contains the following Java code:

```
private void botonAgregarActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```



```
Principal.java  VentanaPrincipal.java

Source Design History | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205

private void botonAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String nombre = campoNombre.getText();
    char password[] = campoPassword.getPassword();
    String password2 = new String(password);
    // INVOCAMOS AL METODO QUE CONSUME EL WEBSERVICE agregarUsuario
    agregarUsuario(password2, nombre);
    JOptionPane.showMessageDialog(this,"Usuario agregado al sistema");
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {...}
// Variables declaration - do not modify
private javax.swing.JButton botonAgregar;
private javax.swing.JButton botonLimpiar;
private javax.swing.JTextField campoNombre;
private javax.swing.JPasswordField campoPassword;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JPanel jPanel1;
// End of variables declaration

// ESTE METODO INVOC LA OPERACION agregarUsuario DEFINIDA EN EL WEBSERVICE
// WSUsuario cuya direccion es http://localhost:8084/ejemplows/WSUsuario?WSDL
private static void agregarUsuario(java.lang.String password, java.lang.String nombre) {
    ejemploclientewsjpa.ws.WSUsuario_Service service = new ejemploclientewsjpa.ws.WSUsuario_Service();
    ejemploclientewsjpa.ws.WSUsuario port = service.getWSUsuarioPort();
    port.agregarUsuario(password, nombre);
}
```

Solo nos queda agregarle un poquito de código manualmente al método del evento para que invoque al método agregarUsuario

**Línea 143:** es la firma o cabecera del método que se ejecuta automáticamente cuando alguien da click sobre el botónAgregar, ese método no retorna valor alguno y recibe como parámetro una instancia del evento que se ha realizado sobre el botónAgregar.

**Línea 145:** declaramos una variable de texto (tipo String) llamada nombre, a la cual se asignamos como valor el dato que se ha ingresado dentro del componente campoNombre, los campos de texto son instancias u objetos de la clase JTextField, la cual tiene entre otros métodos uno que permite obtener el texto que se ha ingresado dentro el mismo, este método se llama getText(), por esa razón es que lo invocamos en esta línea utilizando la instancia campoNombre;

**Línea 146:** es similar a la línea 147, pero la diferencia es que este no es un campo de texto sino un campo de contraseña, estos componentes son instancias de la clase JPasswordField, y no poseen un método llamado getText() para obtener la contraseña como texto, su equivalente es un método llamado getPassword(), el cual devuelve la clave como un arreglo de caracteres de tipo char.

**Línea 147:** Declaramos una variable de texto ósea de tipo String llamada password2, cuyo valor le asignamos una instancia de la clase String formada a partir del arreglo de caracteres que contiene el password ingresado en el campoPassword.

**Línea 149:** invocamos al método agregarUsuario generado automáticamente por el IDE Netbeans y es el que en realidad se conecta con el WS WSUsuario e invoca su operación agregarUsuario y la pasa el nombre y el password ingresados desde el formulario de la ventana.

**Línea 150:** Invocamos al método showMessageDialig de la clase JOptionPane del paquete javax.swing, el cual muestra una ventanita emergente tipo notificación con el mensaje que se le pasa como segundo parámetro, el primer parámetro this, indica que la ventanita de notificación será mostrada sobre esta (this es inglés) ventana en la que estamos escribiendo el código Java, es decir, la VentanaPrincipal.

```
ejemploclientewsjpa - NetBeans IDE 7.2
Archivo Editar Ver Navegar Fuente Reestructurar Ejecutar Depurar Profile Team Herramientas Ventana Ayuda
<configuración prede...
Principal.java - Navega... ☰ Principal.java ☰ VentanaPrincipal.java ☰
Vista de miembros
Principal.java
main(String[] args)
Source History ☰
1 /*
2  * To change this template, choose Tools | Templates
3  * and open the template in the editor.
4  */
5 package ejemploclientewsjpa;
6
7 /**
8  *
9  * @author JohnArrieta
10 */
11 public class Principal {
12
13 /**
14  * @param args the command line arguments
15 */
16 public static void main(String[] args) {
17     // TODO code application logic here
18
19 }
20
21 }
```



Solo nos queda hacer que cuando la aplicación inicie, es decir, se ejecute, sea la **VentanaPrincipal** lo primero aparezca en la pantalla del PC. Esto lo haremos agregando unas cuantas líneas de código dentro del método main de la clase Principal, que como ya sabemos esta clase será el punto de inicio o ejecución de nuestra aplicación.

The screenshot shows the Eclipse IDE interface. The title bar says ".2". The menu bar includes 'Estructurar', 'Ejecutar', 'Depurar', 'Profile', 'Team', 'Herramientas', 'Ventana', and 'Ayuda'. Below the menu is a toolbar with icons for configuration, run, stop, and others. The editor tab bar shows 'Principal.java' and 'VentanaPrincipal.java'. The code editor displays the following Java code:

```
3 * and open the template in the editor.
4 */
5 package ejemploclientewsjpa;
6
7 import johncarlosarrietaarrieta.ventanas.VentanaPrincipal;
8
9 /**
10 *
11 * @author JohnArrieta
12 */
13 public class Principal {
14
15 /**
16 * @param args the command line arguments
17 */
18 public static void main(String[] args) {
19     // TODO code application logic here
20     VentanaPrincipal aplicacion = new VentanaPrincipal();
21     aplicacion.setTitle("::: Ejemplo simple de WebServices con JPA :::");
22     aplicacion.setLocationRelativeTo(null);
23     aplicacion.setVisible(true);
24 }
25 }
```

**Línea 20:** declaramos una variable llamada aplicación de tipo **VentanaPrincipal** y le asignamos una instancia de su mismo tipo invocando la palabra Java new seguida del constructor de la clase **VentanaPrincipal**.

**Línea 21:** invocamos el método **setTitle** de la ventana para poder cambiarle la propiedad **Title**, la cual coloca un título en la barra de título de la ventana.

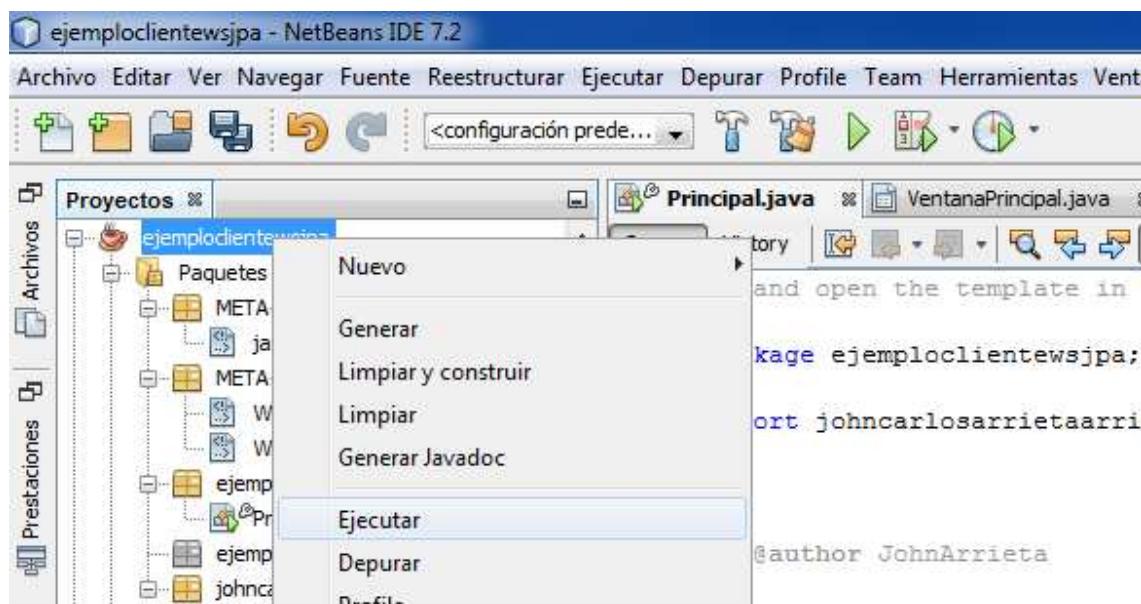
**Línea 22:** invocamos el método **setLocationRelativeTo** de la ventana, con el fin de poder colocarla relativa al centro de la pantalla del PC

**Línea 23:** invocamos el método **setVisible** de la ventana para poder hacerla visible.



## Paso 22: DEPURAR, EJECUTAR Y PROBAR EL SISTEMA DISTRIBUIDO

Si llegamos a este último paso, quiere decir que has seguido la pie de la letra cada paso anterior, ahora solo nos queda depurar, ejecutar y probar nuestra aplicación Cliente WebServices, para lo que solo tenemos que pulsar click derecho sobre la carpeta raíz de nuestro proyecto, luego escoger la opción Depurar o la opción Ejecutar, esperamos unos segundos y aparecerá nuestra ventana.

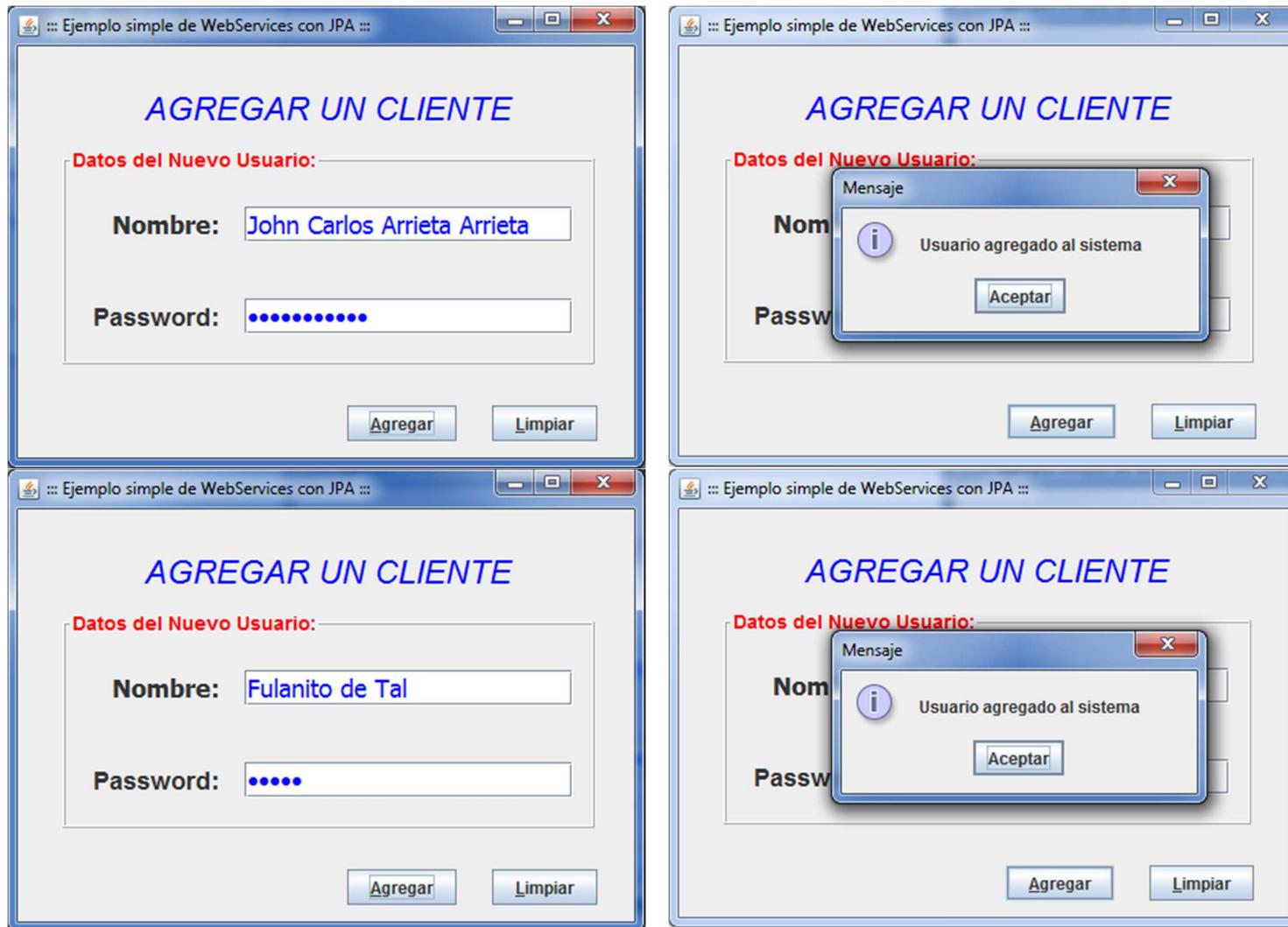


Es importante recordar que para que todo funcione bien, es necesario verificar los siguientes puntos:

- Tener iniciado o en ejecución el servidor de BD MySQL corriendo en su interior la base de datos utilizada como ejemplo en esta guía didáctica.
- Tener iniciado o en ejecución el servidor de aplicaciones Apache Tomcat desde el IDE Netbeans escuchando sobre el puerto 8084
- Tanto el servidor de base de datos, como el servidor Apache Tomcat y la Aplicación Cliente Web deben estar ejecutándose en el mismo PC
- Se deben ingresar todos los datos del formulario de usuarios en la ventana de la aplicación WebServices, ya que el código de validación de campos requeridos no lo hicimos en esta guía por cuestiones de tiempo y simplicidad.



Pantallazos de dos pruebas de la Aplicación cliente WS enviado datos al ServicioWeb, el cual los ha registrado en el servidor de BD.

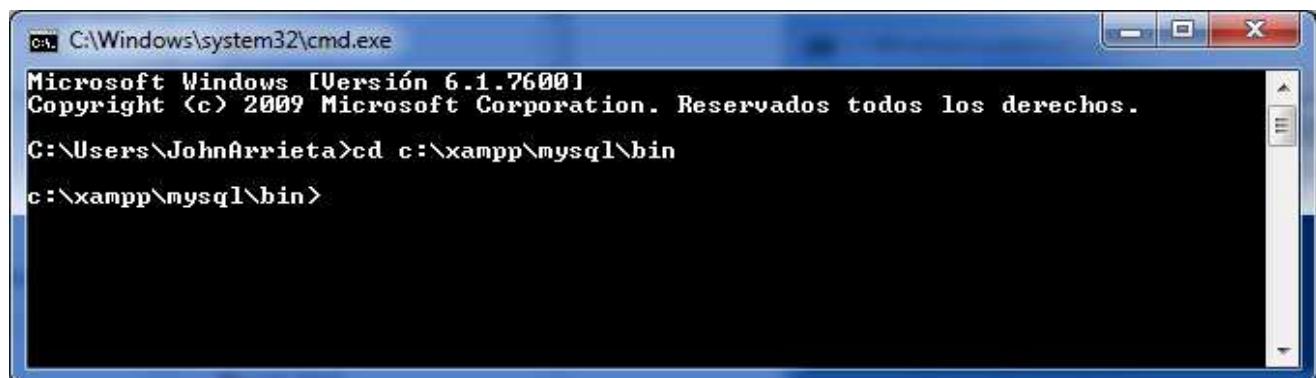


Para probar que los datos anteriores fueron ingresados en la BD, debemos primero utilizar alguna aplicación cliente de MySQL por ejemplo PhpMyAdmin, WorkBench, MySQL Administrator o el mismo IDE Netbeans, entre otros, pero en este ejemplo yo voy a utilizar el cliente por defecto que viene incluido con la instalación de MySQL.

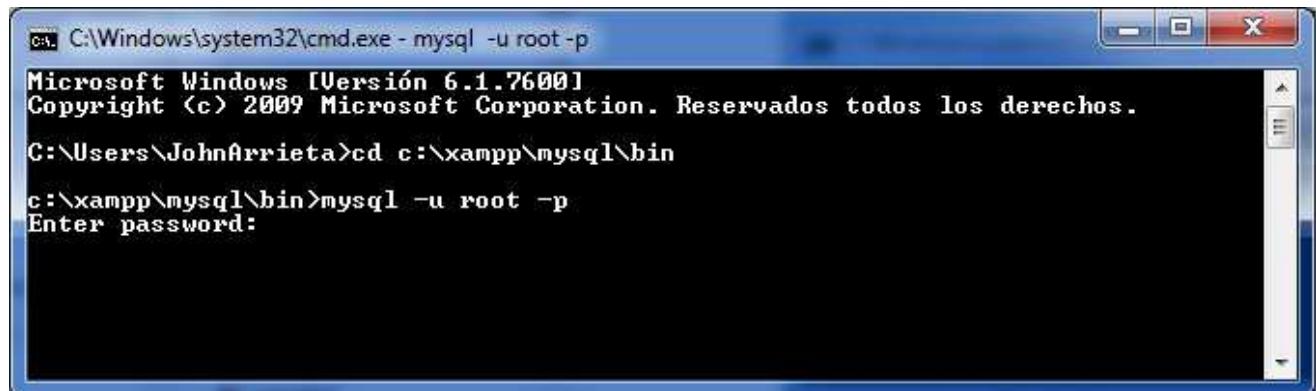
Primero abrimos una ventana de DOS, por ejemplo usando el comando cmd



Segundo, digitamos el comando CD y la ruta de la carpeta bin de MySQL, de tal manera que podamos entrar a ella



Tercero, dentro de la carpeta bin existe una aplicación llamada mysql.exe, el cual nos permite conectarnos al servidor utilizando los parámetros de seguridad requeridos, en este caso, como no se han modificado los datos por defecto de la instalación de mysql, entonces podemos entrar usando el usuario root sin password. Un ejemplo de conexión puede ser el que se muestra en la siguiente imagen, -u indica usuario, y -p indica password.



Cuarto, una vez entramos al servidor mysql, este nos presenta un mensaje de bienvenida acompañado de un poco de información sobre como solicitar ayuda de los comandos y sobre la versión utilizada del servidor.

Luego procedemos a entrar a la base de datos que creamos al inicio de esta guía didáctica, pero ellos ingresamos el comando use seguido del nombre de la BD terminando con ; punto y coma.

```
c:\Windows\system32\cmd.exe - mysql -u root -p
Microsoft Windows [Versión 6.1.7600]
Copyright © 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\JohnArrieta>cd c:\xampp\mysql\bin

c:\xampp\mysql\bin>mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.1.33-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use jpa_bd;
```

Quinto, ahora dentro de la base de datos jpa\_bd procedemos a ingresar una instrucción en lenguaje SQL de tipo consulta, en la cual solicitamos al servidor que nos muestre todos los campos o columnas y registros de la tabla Usuarios.

```
c:\Windows\system32\cmd.exe - mysql -u root -p
Microsoft Windows [Versión 6.1.7600]
Copyright © 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\JohnArrieta>cd c:\xampp\mysql\bin

c:\xampp\mysql\bin>mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.1.33-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use jpa_bd;
Database changed
mysql>
mysql> SELECT * FROM Usuarios;
```



Sexto, si la consulta está bien escrita y nuestras tres aplicaciones (libreriajpabd, ejemplowebseviceja y ejemploclientews) construidas durante esta guía didáctica funcionan perfectamente, el servidor mysql debería mostrarnos las siguiente información.

```
C:\Windows\system32\cmd.exe - mysql -u root -p
Microsoft Windows [Versión 6.1.7600]
Copyright <c> 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\JohnArrieta>cd c:\xampp\mysql\bin

c:\xampp\mysql\bin>mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.1.33-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use jpa_bd;
Database changed
mysql>
mysql> SELECT * FROM Usuarios;
+----+-----+-----+
| id | password | nombre |
+----+-----+-----+
| 1  | 127719   | John Carlos Arrieta Arrieta |
| 2  | 12345    | Fulanito de Tal               |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Para salir del cliente MySQL (no del servidor) se introduce el comando exit o \q.

## EJERCICIO COMO EVIDENCIA DE APRENDIZAJE Y AUTOAPRENDIZAJE

- 1) ¿Cuál de las tres aplicaciones del sistema distribuido que aquí desarrollamos y explicamos paso a paso debes modificar para poder desplegar las operaciones eliminarUsuario, modificarUsuario, buscarUsuario, listarUsuarios y contarUsuarios?. Explica y justifica tu respuesta detalladamente
- 2) ¿Sera que si haces las siguientes operaciones la aplicación ClienteWS que aquí desarrollamos pasó a paso, funcionara sin problemas o tendremos que realizar modificaciones y recompilar el proyecto?. Explica y justifica tu respuesta detalladamente.

Construir la funcionalidad del sistema distribuido para que pueda:

- 3) Eliminar usuarios remotamente
- 4) Modificar usuarios remotamente
- 5) Buscar usuarios remotamente
- 6) Listar usuarios remotamente
- 7) Contar usuarios remotamente
- 8) Hacer login y password remotamente. Para los puntos 3 al 8 se debe enviar a [arrietajohn@hotmail.com](mailto:arrietajohn@hotmail.com) un comprimido con los tres proyectos terminados.