



# CONVERTIDOR APLICACIÓN DE ESCRITORIO

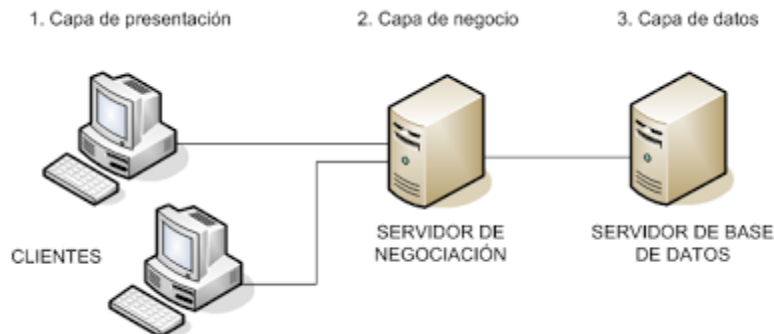
Aplicación de escritorio en java conectada con servlet  
diseñado como aplicación web.

Jenniffer Helena Alvarez Puello  
Juan Camilo Tejada Porto

Universidad de Cartagena  
Facultad de ingeniería  
Programa de Ingeniería de Sistemas  
Cartagena 2017

## Conexión a servlet desde aplicación de escritorio.

En el siguiente informe se explicara la estructura de una aplicación de escritorio desarrollada en Java, la cual estará dependiendo de una aplicación web que es su servlet,(Para desarrollar la aplicación web, leer guía servlet convertidor jsp), básicamente la estructura que se manejara será la siguiente:



*Ilustración 1*

En el cual los clientes están separados de las demás capas, en las cual la capa de negocio será la aplicación web desarrollada).

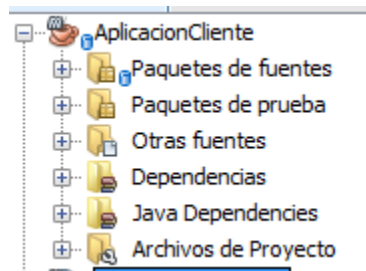
Para la construcción de esta aplicación se utilizó **MAVEN** la cual es una de las herramientas más útiles a la hora de utilizar librerías de terceros. Por qué lo vamos a utilizar, ya que este nos ofrece la posibilidad de descargar estas librerías directamente desde nuestro proyecto, sin necesidad de buscar archivos por internet para después importarlos al proyecto.

Si hacemos una búsqueda en Google para averiguar estos datos que necesitamos, simplemente poniendo “maven gson“, obtendremos, directamente del repositorio maven, esta información:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.2.4</version>
</dependency>
```

## Convertidor con conexión a servlet desde aplicación de escritorio

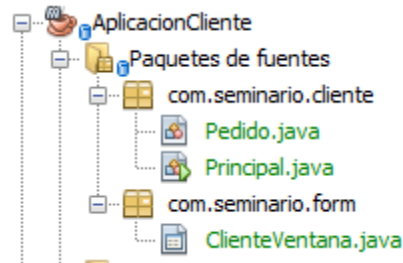
Lo primero es crear un proyecto **Java Application** con la siguiente estructura:



*Ilustración 2*

En el cual nos los paquetes que tenemos son: *paquetes de fuentes*, *dependencias*, *Java Dependencies* y *Archivos de proyecto*.

En el paquete de fuentes, se encuentra la parte del código encargada de la aplicación de escritorio, y de la misma forma la conexión con el servlet.



*Ilustración 3*

En la *Ilustración 3* nos encontramos con la estructura interna del paquete encargado de toda la lógica de la aplicación.

## 1. PRINCIPAL.

En la clase **Principal.java** tendremos la creación e instancia de la ventana creada que es la aplicación del de escritorio, como se muestra en el siguiente código:

```
package com.seminario.cliente;

import com.seminario.form.ClienteVentana;

public class Principal {
    public static void main(String[] args) {
        ClienteVentana ventana = new ClienteVentana();
        ventana.setTitle("Cliente Escritorio Convertidor");
        ventana.setLocationRelativeTo(null);
        ventana.setVisible(true);
    }
}
```

*Ilustración 4*

Primero creamos un objeto *ventana* de tipo **ClienteVentana**, y se añaden algunos atributos a la ventana como o son el título, la ubicación en la que se abrirá y finalmente se hace visible este formulario.

## 2. PEDIDO

Posteriormente crearemos una nueva clase con este **pedido.java**, y en ella añadiremos la información que tienen los documentos que se encuentran en esa colección. Con esta clase ya hemos recogido toda la estructura básica del JSON, ahora tendríamos que crear todos los getter y setter para todas las variables que hemos creado, En la clase **pedido.java** nos encontraremos con los métodos que hacen posible la obtención y envío de los objetos, esto va de la mano con la utilización de la librería Gson. quedando la clase así:

```
package com.seminario.cliente;
```

```
public class Pedido {  
    private String actual;  
    private String objetivo;  
    private String valor;  
    private String result;  
  
    public String getActual() {  
        return actual;  
    }  
  
    public void setActual(String actual) {  
        this.actual = actual;  
    }  
  
    public String getObjetivo() {  
        return objetivo;  
    }  
  
    public void setObjetivo(String objetivo) {  
        this.objetivo = objetivo;  
    }  
  
    public String getValor() {  
        return valor;  
    }  
  
    public void setValor(String valor) {  
        this.valor = valor;  
    }  
  
    public String getResult() {  
        return result;  
    }  
  
    public void setResult(String result) {  
        this.result = result;  
    }  
}
```

```
}
```

### 3. Cliente Ventana

En la clase **ClienteVentana.java** encontraremos el formulario diseñado que es la vista de la aplicación de escritorio, y tenemos la siguiente forma:

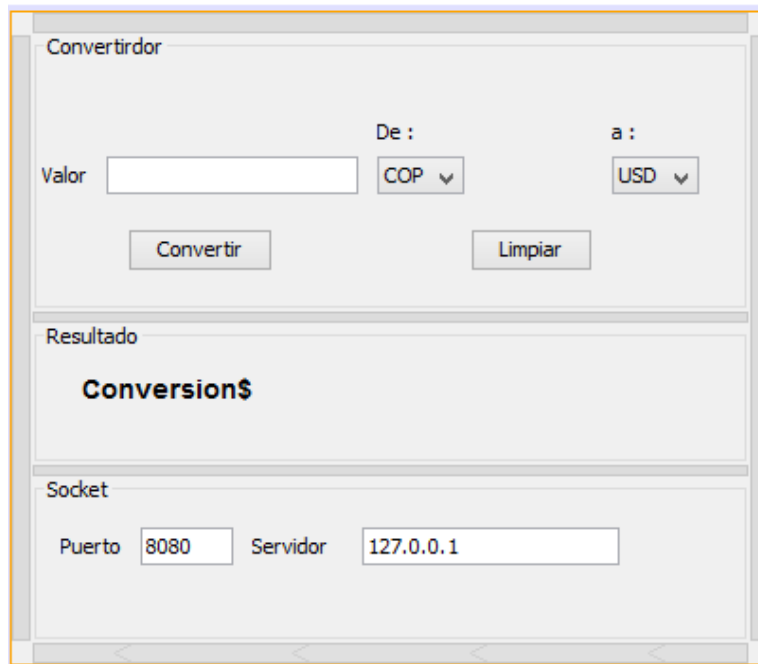


Ilustración 5

Cuando ingresamos al código de clienteVentana.java lo primero que observaremos la definición de variables globales:

Deserializar el documento JSON

```
DataOutputStream out ;  
DataInputStream in;  
Gson gson = new Gson();
```



En este caso hemos utilizado un objeto GsonBuilder porque necesitábamos establecer un patrón para la fecha y escapar los símbolos html, en el caso de que no necesitemos ningún tipo de configuración basta con hacer:

```
Gson gson = new Gson();
```

Para el desarrollo de la vista tenemos:

En el botón **Limpiar** encontraremos la siguiente lógica:

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {  
    labResultado.setText("Conversion$");  
}
```

El cual será el encargado de mostrar en el Label que muestra el resultado, dejar el campo "LIMPIO".

Con el botón **Convertir** encontramos la siguiente lógica:

1. Verificamos que los campos requeridos no se encuentren vacíos.

```
if(tPuerto.getText().equalsIgnoreCase("") && tServidor.getText().equalsIgnoreCase("")) {  
    JOptionPane.showMessageDialog(this, "Por Favor LLene los campos necesarios");
```

2. Hacemos la declaración de variables y la conexión con el servlet en la línea de código donde se define el **request** y se manda como parámetro en el objeto de tipo **URL**. Y luego con el objeto de tipo **URLConnection** manejamos la conexión con el servidor que en este caso es un servlet ejecutándose en la URL antes construida.

```
try {  
    String param = "pc=";  
    String request = "http://" + tServidor.getText() + ":" + tPuerto.getText() + "/ConvertidorMoneda/moneda";  
    URL url = new URL(request);  
    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
```

3. Este paso es para poder trabajar con la librería Gson:

En esta paso sería convertir este objeto que tenemos en Java en un documento JSON, para eso Gson nos permite deserializar nuestra clase para extraer el documento JSON a partir de sus datos, aunque esta es la parte más sencilla de todas. Se haría simplemente llamando al método `toJson` del objeto gson, y le pasaríamos como parámetro el objeto que queremos parsear a JSON:

```
Pedido obj = new Pedido();  
obj.setActual(comboActual.getSelectedItem().toString());  
obj.setObjetivo(comboFinal.getSelectedItem().toString());  
obj.setValor(tValor.getText());  
String json = gson.toJson(obj);  
param = param + json;
```

Y con esto ya tendríamos parseada toda la información que contuviera el objeto que parseamos. Tan sólo tendríamos que tener en cuenta que si alguno de los campos de nuestro objeto son null, en el documento JSON no aparecería. Si deseamos que aparezca este campo en el JSON como null, en la inicialización del objeto gson tendríamos que añadir el método `serializeNulls`.

Ahora procedemos a definir las respectivas instancias y forma de comunicación que se tendrán con la aplicación web, y de igual forma los métodos de envío de la información que se hace por el método **POST**. Estas instancias se definen llamando los métodos del objeto `connection` con esta definimos el encabezado para la comunicación con el protocolo HTTP, incluyendo la longitud del contenido a enviar.

```
connection.setDoOutput(true);
connection.setDoInput(true);
connection.setInstanceFollowRedirects(false);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
connection.setRequestProperty("charset", "utf-8");
connection.setRequestProperty("Content-Length", "" + Integer.toString(param.getBytes().length));
connection.setUseCaches(false);

System.out.println("Param : " + param);
```

Luego por medio de las variables globales definidas al inicio se hace el envío y recibido de los resultados, es la parte donde se manda al servidor y se recibe del mismo. Con el objeto `out` mediante el método `writeBytes` colocamos en cola de envío la petición al servidor luego con el método `flush` realizamos el envío de todo lo que está en cola y seguido cerramos la conexión, Luego con el objeto `in` obtenemos la respuesta del servidor y con ayuda de un `StringBuffer` armamos el `String` con el mensaje de respuesta del servidor cuyo valor es mostrado por la vista.

```
out = new DataOutputStream(connection.getOutputStream());
out.writeBytes(param);
out.flush();
out.close();
int response = connection.getResponseCode();
System.out.println("Response Code : " + response);

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

String result;
StringBuffer respon = new StringBuffer();

while ((result = in.readLine()) != null) {
    respon.append(result);
}
String conve = gson.fromJson(respon.toString(), String.class);
in.close();
connection.disconnect();

labResultado.setText("$"+conve);
```

