Ismael Calvo Villalvilla

51689141E

alaricoi@gmail.com

TRABAJO 3 DE EVALUACIÓN CONTINUA

Generación Automática de Código

Contenido

1 Introducción 2

1.1 Enunciado del trabajo Propuesto: 2

1.2 Consideraciones Generales de la Solución 2

2 Desarrollo de la práctica 3

2.1 Recursos de programación utilizados 3

2.2 Descripción (especificación) detallada de la solución 3

2.2.1 PHP 6

2.2.2 Java 6

2.2.3 Detalle de la implementación 7

2.3 Alcance y limitaciones de la solución propuesta 19

2.4 Descripción de los casos de Prueba 20

3 Ubicación de los programas fuente y explicación del resto de directorios/ficheros anexados 20

4 Manuales de uso/instalación 20

5 Conclusiones. 20

# Introducción

En el presente documento se va describir la solución aportada al trabajo 3 de la asignatura de Generación Automática de Código del máster universitario Investigación en Ingeniería de Software y Sistemas

## Enunciado del trabajo Propuesto:

*Se trata de generar un programa “CRUD” (altas baja consulta y modificación de un registro de cualquier tabla de una base de datos definida en un SGBD relacional).*

*El generador tendrá como entrada el nombre de una tabla.*

*El usuario indica el nombre de la tabla (que deberá existir, obviamente, en el SGBD correspondiente) y el programa generador implementará una clase con, al menos, los métodos correspondientes de alta, baja, consulta y modificación de dicha tabla.*

*Se valorará positivamente:*

*Que el generador se pueda utilizar contra SGBD’s diferentes*

*Que el generador obtenga la clase en más de un lenguaje de salida*

*El alumno deberá proponer algún mecanismo de “extensibilidad” que permita incorporar validaciones suplementarias (“custom code”) sin perder la posibilidad de regeneración; esto es, cuando se regenere el programa se incorporarán dichas validaciones de manera dinámica.*

## Consideraciones Generales de la Solución

**Se va a utilizar parte del trabajo 2** para la captura de los datos referentes a la conexión a los diferentes SGBD, es decir, **MySQL y SQLite**.

Al utiliza jdbc nativo, el acceso a los metadatos del esquema seleccionado y de sus tablas es genérico por lo que la ampliación a otros SGDB,s resultaría poco invasiva.

**La generación de código se realizará en base a plantillas,** los motivos de la elección son:

* **Flexibilidad** a la hora de generar distinto código en diferentes lenguajes. En nuestro caso he optado por generar PHP y JAVA. Aunque hay parte e código dedicado la mayoría se basa en tag de sustitución por lo que incorporar otro lenguaje no sería traumático, si bien es verdad que el mayor esfuerzo lo conllevaría crear las pantallas necesarias.
* **Personalización no intrusiva,** es decir, modificar una la salida de generación la mayoría de las veces conlleva solo modificar la plantilla, evitando el mantenimiento del código del generador.

Por el contrario, el desarrollo de este tipo de soluciones conlleva un gran esfuerzo inicial ya que hay que definir previamente las posibilidades de condigo generado y trasladarlo a una plantilla. Con otros tipos generadores basados en la especificación del lenguaje el desarrollo sería más rápido pero estaríamos constreñidos a los lenguajes y arquitecturas seleccionados en un principio, sin posibilidad de flexibilizar el proceso.

# Desarrollo de la práctica

## Recursos de programación utilizados

Al igual que en los trabajos anteriores, el lenguaje elegido para el desarrollo es java (jdk 1.8), por los mismos motivos de:

* Comodidad en la gestión de SGDB,
* facilidad a la hora de trabajar con cadenas de caracteres (búsqueda, reemplazo, concatenación).
* Facilidad en el tratamiento de ficheros.
* Lenguaje de programación ampliamente difundido.

Las herramientas utilizadas para la construcción del programa son:

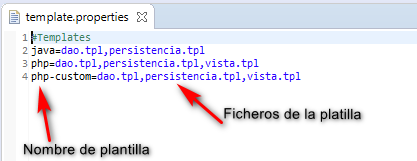
* Sistema operativo Windows 10.
* Como IDE (Integrated Development Environment) he utilizado Eclipse Oxigen en su instalación de Java.
* Librerías JAVA de terceros:
  + librería de SWT (Standard Widget Toolkit), de eclipse y openSource, nos permite crear de aplicaciones de ventanas y que crea la interfaz de usuario a partir de componentes operativos de cada Sistema operativo.
  + Librerías de conexión a SGDB. Para el trabajo hemos empleado SQLite y MySQL, sus conectores son los ficheros:
    - * mysql-connector-java-5.1.45-bin.jar
      * sqlite-jdbc-3.21.0.jar

## Descripción (especificación) detallada de la solución

Como se ha mencionado en la introducción, el generador se basa en plantillas, principalmente lo que se ha realizado es un trabajo previo de código bien formado en cada uno de los lenguajes y a partir de este código se han creado las diferentes plantillas.

Inicialmente se ha definido una arquitectura de ficheros de configuración donde podremos definir los lenguajes a generar y las plantillas de cada lenguaje, lo que nos permite ampliar el generador si modificar el su código fuente (o siendo lo menos invasivo posible).

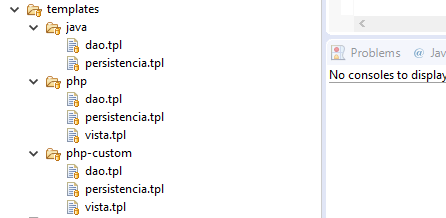
1. En la carpeta conf existe el fichero **template.properties** que nos servirá para definir el lenguaje y sus platillas.



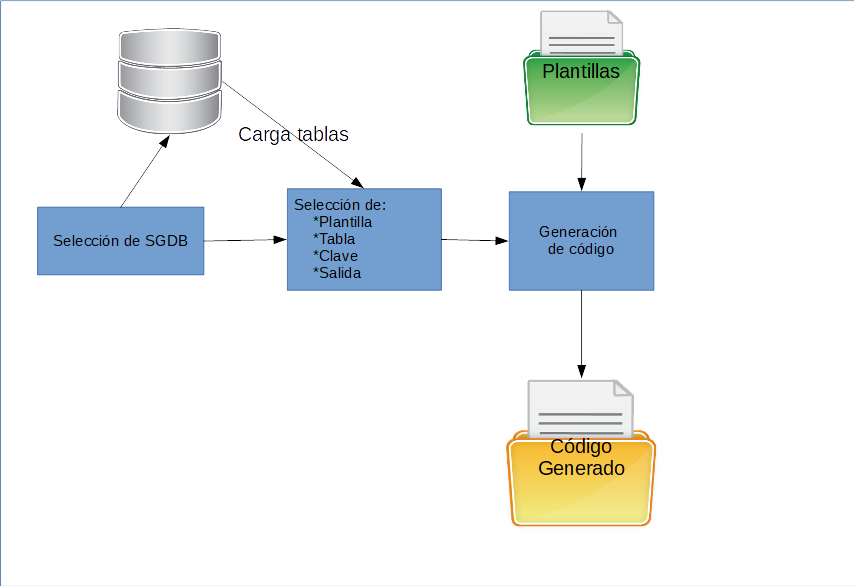
En el fichero de propiedades la clave es el nombre de plantilla y el valor cada uno de los ficheros de plantillas que luego generará un fichero de código fuente.

El nombre de plantilla está asociado al lenguaje, es decir, la extensión del fichero generado será el nombre la plantilla, como podemos tener varias configuraciones por lenguaje, he optado por que cada configuración debe empezar por su extensión de lenguaje hasta el carácter “–“, si no dispone de dicho carácter será todo el nombre la configuración

1. **Plantillas**. Las plantillas están situadas en la carpeta **templates**, cada configuración debe ser una carpeta y debe contener los ficheros definidos en **template.properties**



Una vez definidas las plantillas, el proceso de generación presenta las siguientes partes:



* Recogida de datos de conexión. El Generador recogerá los datos de conexión a base de datos y cargará los nombres de las tablas en un selector (comboBox).
* Recogida de los datos propios para la generación:
  + Plantilla: Se cargara un comboBox desde template.propeties con las platillas disponibles.
  + Tabla sobre la que queremos generar el CRUD. Desde el selector que se cargó en el punto anterior
  + Campo de la clave primaria de dicha tabla. Hay que tener en cuenta que muchos SGDBs son sensibles a mayúsculas y minúsculas.
  + Directorio de salida en filesystem.
* Generación propiamente dicha de los ficheros de código fuente a partir de las plantillas. Se han definido una serie de texto clave para su inclusión en la plantilla y que serán sustituidos por sus valores reales a la hora de generar el código, los tags definidos se presentan en la siguiente tabla.

| tag | | descripción |
| --- | --- | --- |
| <<#paquete\_modelos#>> | | En caso de java, el paquete donde reside la capa de persisitencia. En nuestro ejemplo será domain |
| <<#paquete\_daos#>> | | Java. Paquete de la capa de dao |
| <<#conexionBd#>> | | Cadena de conexón a los datos |
| <<#userDb#>> | | Usuario de conexión a base de datos |
| <<#nombre\_clave\_tabla#>> | | Nombre en base de datos de primaria de la tabla. Campo unico |
| <#nombre\_clave#>> | | Nombre de la vaiable del campo la clave en la generación de código |
| <<#nombreClase#>> | | Nombre que se asignara a la clase de persisitencia generada. |
| <<#nombreObjeto#>> | | La varible definida del tipo de la clase de persisitencia |
| <<#tablename#>> | | Nombre real de la tabla en SBDB |
| <<#todos\_campos\_tabla#>> | | Campos de la tabla en el SGDB separados por coma |
| <<#todos\_campos\_tabla\_parametro#>> | | Generación de campos de parámetro según el número de campos. |
| <<#todos\_campo\_valor#>> | | Cadena que genera el par clave valor a partir de los campos de la tabla. Se utiliza para el update de la tabla |
| <<#numero\_campos#>> | | Numero de campos de la tabla |
| "<<#seccion\_loop\_campos#>>  …..  <</#seccion\_loop\_campos#>> | | Etiqueta especial que nos permite realizar un proceso de bucle con tags relativos a campos. Se debe marcar con una etiqueta de cierre. Detro de este bloque van las siguientes entiquetas |
|  | <<#nombre\_campo#>> | Nombre de la variable campo con formato camelCase |
|  | <<#Nombre\_campo#>> | Igual que el anterior pero con la primera letra en mayúsculas. |
|  | <<#nombre\_campo\_tabla#>> | Nombre real del campo en la tabla del SGDB |
|  | <<#posicion\_campo#>> | Posición del campo en el sistema |
|  | <<#tipo\_campo#>>" | Tipo del campo trasformado al lenguaje, se emplea en las platillas Java. |

Por otra parte, para dar soporte a la funcionalidad de “custom mode” se han creado dos tipos de etiquetas diferentes que nos acotaran la parte del fichero donde podremos incorporar código sin que se machaque una nueva generación.

| tag | descripción |
| --- | --- |
| /\*<<#custom\_code\_head#>>\*/  /\*<</#custom\_code\_head#>>\*/ | Etiqueta especial que nos permite incorporar código de cabecera customizado en cabecera del fichero, por ejemplo en java sección de imports customizados. |
| /\*<<#custom\_code\_body#>>\*/  /\*<</#custom\_code\_body#>>\*/ | Etiqueta especial que nos permite incorporar código de customizado, solo puede exisitir uno por fichero. |

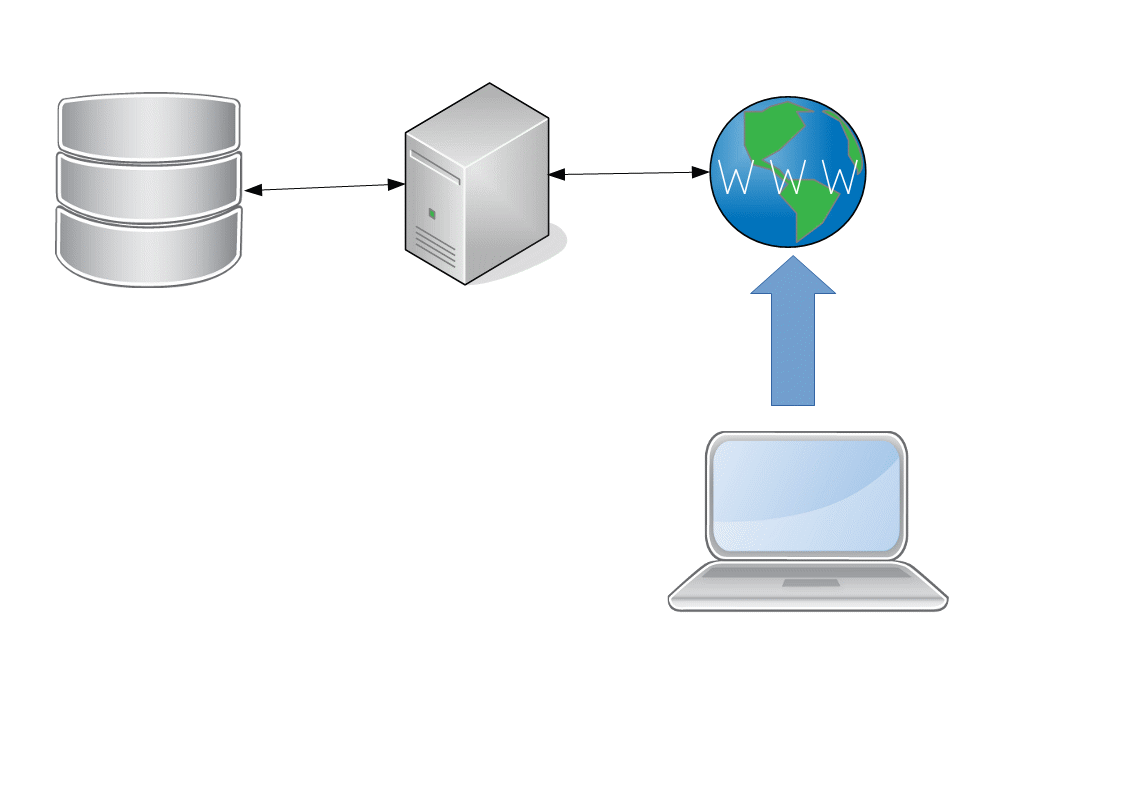
Para las pruebas de correcto funcionamiento del generador se ha partido de los siguientes casos:

### PHP

Se ha definido la salida del generador en tres ficheros dependiendo de su funcionalidad o capa funcional:

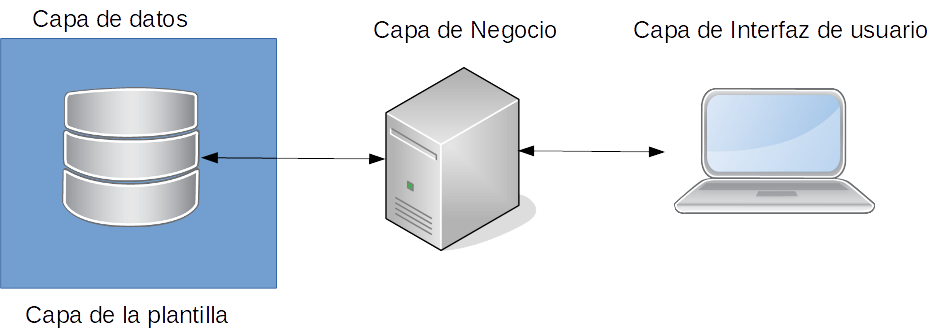
* Capa de Objeto de persistencia
* Capa de acceso a datos
* Capa de vista

Es un caso completo que nos generará una página funcional en el navegador, una clase dao que gestiona las acciones sobre los datos y una clase de persistencia que nos proporciona el objeto para trasladar las acciones desde la capa de dao a la vista.



### Java

Para este caso, me he centrado en la arquitectura de tres capas clásica, el generador solo creará la capa de acceso a datos y quedará pendiente del usuario del generador las llamadas de desde la capa de negocio. La elección de este tipo de generación es por la flexibilidad del código de generado ya que también se podrá utilizar en otras arquitecturas, como en desarrollo web la arquitectura Modelo-Vista-Controlador.



### Detalle de la implementación

Dado que la implementación se complica con respecto a los trabajos anteriores, se ha optado por dividir el código en clases dependiendo de la funcionalidad del mismo por tanto tendremos:

* t3App.java : Interfaz de usuario para la captura de datos.
* DatosCrud.java: Clase con los datos tratados desde el interfaz de usuario que se pasaran a la clase encargada de realizar el proceso.
* LecturaPlantilla.java: Clase que realizará la generación de código dependiendo de los datos pasados con un objeto de tipo DatosCrud. Esta clase es agnóstica del sistema operativo y junto con DatosCrud son exportables a cualquier sistema.

Inicialmente pasamos a describir que valores contiene los objetos de tipo **DatosCrud**, que nos va a permitir enlazar la parte de interfaz de usuario con la clase de negocio que se encarga de generar el código.

/\*\*

\* tabla de base de datos

\*/

**private** String tabla;

/\*\*

\* Nombre del campo clave

\*/

**private** String nombreClave;

/\*\*

\* Objeto que guarda los campos y el tipo de datos

\*/

**private** Map <String, String> campos;

/\*\*

\* cadena de conexiónn de la base de datos

\*/

**private** String conexionBd;

/\*\*

\*

\* Uuasrio de base de datos

\*/

**private** String usuarioBd;

/\*\*

\* clave de conexión a bd

\*/

**private** String claveBd;

/\*\*

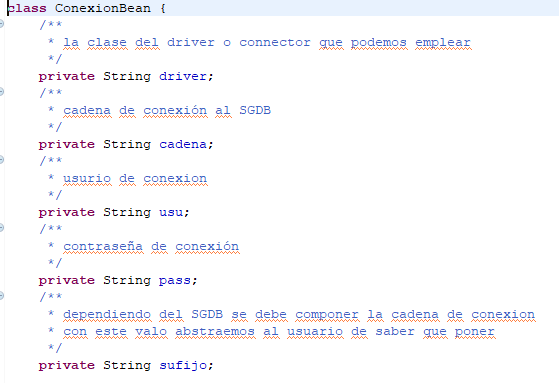
\* directorio raíz donde queremos exportar el código

\*/

**private** String pathSalida;

Parte de la clase de interfaz de usuario, **t3App.java** se basa en la solución del trabajo 2 para la captura de los datos de conexión y carga de las tablas en un selector.

Clase **ConexionBean** que nos permite guardar los datos introducidos por pantalla en un objeto y así tratar los parámetros más fácilmente.



A partir de los datos introducidos por pantalla se genera la conexión y se carga el comboBox con las tablas, los datos solicitados son:

* Tipo de conexión
* Cadena
* Usuario
* Contraseña

Desde el Botón de conectar llamamos al método **rellenaComboTablas**, que realiza:

/\*\*

\* Rellena el combo de tablas a partir de la base de datos seleccionada

\*/

**private** **void** rellenaComboTablas() {

// Es MySQL

String driver = **null**;

String sufijo = **null**;

**if** (comboTipo.getSelectionIndex() == 1) {

driver = "com.mysql.jdbc.Driver";

sufijo = "jdbc:mysql://";

}

// Es SQLite

**if** (comboTipo.getSelectionIndex() == 0) {

driver = "org.sqlite.JDBC";

sufijo = "jdbc:sqlite:";

}

// creamos el objeto conexión con los datos seleccionados

conexionBean = **new** ConexionBean(driver, tCadenaConex.getText(), tUsu.getText(), tPass.getText(), sufijo);

// conexion a bese de datos

Connection conn = dameConexion(conexionBean);

// Si no obtenemos conexión salimos del método

**if** (conn == **null**) {

**return**;

}

ResultSet rs = **null**;

**try** {

// extraccion del catalogo de datos del SGDB

DatabaseMetaData metaDatos = conn.getMetaData();

rs = metaDatos.getTables(**null**, **null**, "%", **null**);

comboTablas.removeAll();

**while** (rs.next()) {

// El contenido de cada columna del ResultSet se puede ver

// en la API, en el metodo getTables() de DataBaseMetaData.

// La columna 1 es TABLE\_CAT

// y la 3 es TABLE\_NAME

// String catalogo = rs.getString(1);

String tabla = rs.getString(3);

comboTablas.add(tabla);

}

muestraDialogoModal(SWT.***ICON\_INFORMATION*** | SWT.***OK***, "Información", "Creada la conexión");

comboTablas.setFocus();

} **catch** (SQLException e1) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error", "No se ha podido crear la conexión");

***logger***.error(e1);

// liberamos los objetos de de conexion

} **finally** {

**try** {

**if** (rs != **null**)

rs.close();

**if** (conn != **null**)

conn.close();

} **catch** (SQLException e1) {

***logger***.error(e1);

}

}

}

El método **dameConexion** nos crea la conexión a base de datos:

/\*\*

\* Crea una conexxión jdbc a partir de los parametros pasados por parametro

\*

\* **@param** tipo

\* valores: 0--> MySql 1--> SQLite

\* **@param** conexion:

\* cadena de conexión, en caso de SQLlite será la ruta del fichero db

\* en caso de que sea Mysql debemos indicar el host:puerto/esquema

\* **@param** usu:

\* usuario de la base de datos

\* **@param** pass:

\* contraseña de acceso

\* **@return**

\*/

**private** Connection dameConexion(ConexionBean con) {

**try** {

// carga del driver o conector

Class.*forName*(con.getDriver());

} **catch** (ClassNotFoundException e1) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha podido crear la conexión: " + e1.getMessage());

***logger***.error(e1);

}

Connection conexion = **null**;

**try** {

// creamos la conexion a base de datos

conexion = DriverManager.*getConnection*(con.getSufijo() + con.getCadena(),

con.getUsu(), con.getPass());

} **catch** (SQLException e1) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha podido crear la conexión: " + e1.getMessage());

***logger***.error(e1);

}

**return** conexion;

}

Una vez que tenemos la conexión solo necesitaremos seleccionar:

* La platilla desde donde tiraremos para generar nuestro código. Se muestra en un selector a partir de los datos configurados en el fichero template.properties.

// cargamos las plantillas

**try** {

String path = **new** File(".").getCanonicalPath();

***logger***.info(path);

FileInputStream input = **new** FileInputStream("./conf/template.properties");

Properties prop = **new** Properties();

prop.load(input);

Set<Object> keys = prop.keySet();

**for** (Object k : keys) {

comboLenguaje.add((String) k);

}

} **catch** (IOException e1) {

***logger***.error(e1);

}

* Selección de la tabla a partir de los datos cargados en la selección de conexión
* Nombre en base de datos de la clave primaria, tener en cuenta que algunos SGBD sin sensibles a mayúsculas y minúsculas, como por ejemplo SQLite.
* Path base de salida del fichero que se generara.

Una vez tengamos los datos de partida para la generación, se creará un objeto **DatosCrud**  con los datos a pasar a la clase de negocio de creación de la plantilla. La llama a la clase :

LecturaPlantilla plantilla = **new** LecturaPlantilla();

………

plantilla.cargaPlatilla(comboLenguaje.getText(), crud);

La llamada a LecturaPlantilla se realiza con los campos ya tratados para intenetr que sea lo más “agnóstica” posible.

/\*\*

\* Método que se encarga de capturar los datos introducidos en la interfaz

\* de usuario y llamar a la clase de generación de código

\*/

**private** **void** trataPlantilla() {

**if** (conexionBean == **null**) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"Para realizar este paso hace falta conectarse previamente");

**return**;

}

**if** (comboLenguaje.getText() == **null**) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha seleccionado lenguaje");

**return**;

}

**if** (comboTablas.getText() == **null**) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha seleccionado la tabla a exportar");

**return**;

}

**if** (tClave.getText() == **null**) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha seleccionado la clave de la tabla");

**return**;

}

**if** (tClave.getText() == **null**) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha seleccionado el destino de la generación");

**return**;

}

LecturaPlantilla plantilla = **new** LecturaPlantilla();

DatosCrud crud = **new** DatosCrud();

crud.setTabla(comboTablas.getText());

//quitamos la llamada a jdbc si no es java

String sufijo = conexionBean.getSufijo();

String conexion = conexionBean.getCadena();

//si es sqlite, como estamos en windows hat que escapar el caracter "\"

**if** (conexionBean.getDriver().equalsIgnoreCase("org.sqlite.JDBC")) {

conexion = conexion.replace("\\", "\\\\");

}

// si es php se debe tratart la conexión de otra manera a java

**if** (comboLenguaje.getText().toLowerCase().startsWith("php")) {

//es mysql y la conexón se realida de forma:

//mysql:dbname=employees;host=localhost;port=3306

**if** (!conexionBean.getDriver().equalsIgnoreCase("org.sqlite.JDBC")) {

//host

String host = conexionBean.getCadena().substring(0, conexionBean.getCadena().indexOf(":"));

//

String port = conexionBean.getCadena().substring(conexionBean.getCadena().indexOf(":") +1,

conexionBean.getCadena().indexOf("/"));

String dbname = conexionBean.getCadena().substring(conexionBean.getCadena().indexOf("/")+1);

conexion = "dbname=" +dbname + ";host=" + host + ";port=" + port;

}

}

crud.setConexionBd(sufijo + conexion);

crud.setUsuarioBd(conexionBean.getUsu());

crud.setClaveBd(conexionBean.getPass());

crud.setNombreClave(tClave.getText());

crud.setPathSalida(tSalida.getText());

ResultSet rs = **null**;

Connection conn = dameConexion(conexionBean);

**try** {

// Si no obtenemos conexión salimos del método

**if** (conn == **null**) {

**return**;

}

**try** {

DatabaseMetaData metaDatos = conn.getMetaData();

rs = metaDatos.getColumns(**null**, **null**, comboTablas.getText(), **null**);

Map<String, String> map = **new** HashMap<String, String>();

**while** (rs.next()) {

map.putIfAbsent(rs.getString("COLUMN\_NAME"), TRANSLADAS\_TIPOS.get(rs.getInt("DATA\_TYPE")));

}

crud.setCampos(map);

plantilla.cargaPlatilla(comboLenguaje.getText(), crud);

muestraDialogoModal(SWT.***ICON\_INFORMATION*** | SWT.***OK***, "Correcto", "Se ha creado la salida de la plantilla");

} **catch** (SQLException e1) {

***logger***.error(e1);

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error", "No se ha podido crear la conexión");

**return**;

} **catch** (Exception e) {

***logger***.error(e);

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error", "No se ha podido crear "

+ " la plantilla revise el correcto etiquetado custom");

**return**;

}

} **finally** {

**try** {

**if** (rs != **null**)

rs.close();

**if** (conn != **null**)

conn.close();

} **catch** (SQLException e1) {

***logger***.error(e1);

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error", "No se ha podido crear la conexión");

**return**;

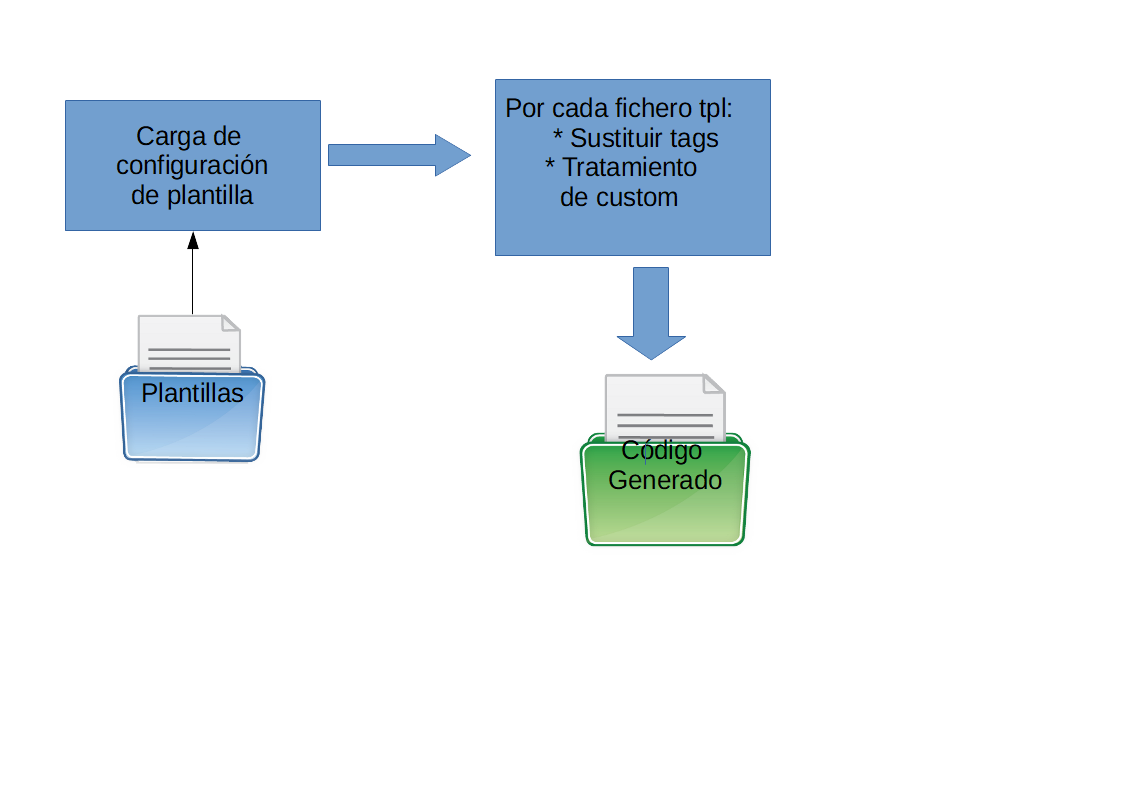
}

}

}

La Clase **LecturaPlantilla** va a realizar los siguientes pasos:

1. Lectura de la configuración de plantilla
2. Por cada fichero de plantilla configurado:
   1. Cargar su contenido
   2. Tratamiento de los tags de sustitución
   3. Tratamiento del contenido customizado
   4. Salva de código fuente generado en destino seleccionado.



La clase dispone de un método de llamada al que se le pasara el objeto DatosCrud y la plantilla seleccionada.

plantilla.cargaPlatilla(comboLenguaje.getText(), crud);

Este método se encarga de llamar al resto de acciones, carga, reemplazo de testo y salva de fichero.

/\*\*

\* Metodo publico que desencadena la generación de código

\* **@param** plantilla

\* **@param** crud

\* **@throws** Exception

\*/

**public** **void** cargaPlatilla(String plantilla, DatosCrud crud) **throws** Exception {

**try** {

// ClassLoader cLoader = this.getClass().getClassLoader();

FileInputStream input = **new** FileInputStream("./conf/template.properties");

Properties prop = **new** Properties();

// load a properties file

prop.load(input);

String[] plantillas = prop.getProperty(plantilla).split(",");

// Por cada fichero configurado

**for** (String pl : plantillas) {

String e = cargaFichero("./templates/" + plantilla + "/" + pl);

// Se trata la platilla con el reemplazo de textos clave

String salidaTratada = trataPlantilla(pl, e, crud);

salvaFuente(plantilla, crud, pl, salidaTratada);

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

***logger***.error(e);

}

}

El método **cargaFichero** se encarga de leer la plantilla y cargarla en un String.

**public** String cargaFichero(String plantilla) **throws** IOException {

// ClassLoader cLoader = this.getClass().getClassLoader();

**try** {

***logger***.info("buscando " + plantilla);

// File file = new File(cLoader.getResource(plantilla).getFile());

File file = **new** File(plantilla);

***logger***.info("File Found : " + file.exists());

// Read File Content

String content = **new** String(Files.*readAllBytes*(file.toPath()));

**return** content;

} **catch** (IOException e) {

***logger***.info("File no found " + plantilla);

**throw** e;

}

// File is found

}

El método **trataPlantilla** se encarga de reemplazar los diferentes tags configurados por los datos seleccionados de generación del CRUD.

/\*\*

\* reemplazar los diferentes tags configurados por los datos seleccionados de generación del CRUD

\* **@param** plantilla

\* **@param** crud

\* **@return**

\*/

**public** String trataPlantilla(String nombrePlantilla, String plantilla, DatosCrud crud) {

String plantillaTratada = plantilla;

// gestiÃ³n de paquetes

// será el nombre de la platilla sin extension

plantillaTratada = plantillaTratada.replaceAll("<<#paquete\_modelos#>>","persistencia");

plantillaTratada = plantillaTratada.replaceAll("<<#paquete\_daos#>>", "dao");

// datos de conexiÃ³n de base de datos

plantillaTratada = plantillaTratada.replaceAll("<<#conexionBd#>>", crud.getConexionBd());

plantillaTratada = plantillaTratada.replaceAll("<<#userDb#>>", crud.getUsuarioBd());

plantillaTratada = plantillaTratada.replaceAll("<<#passDb#>>", crud.getClaveBd());

// se asigan el campo que serÃ¡ clave

plantillaTratada = plantillaTratada.replaceAll("<<#nombre\_clave\_tabla#>>", crud.getNombreClave());

plantillaTratada = plantillaTratada.replaceAll("<<#nombre\_clave#>>", camelCase(crud.getNombreClave(), **false**));

plantillaTratada = plantillaTratada.replaceAll("<<#Nombre\_clave#>>", camelCase(crud.getNombreClave(), **true**));

// llamamos a la clase con el nombre de la tabla con la primera

// letra el mayusculas

String nombreClase = camelCase(crud.getTabla(), **true**);

plantillaTratada = plantillaTratada.replaceAll("<<#nombreClase#>>", nombreClase);

String nombreObjeto = camelCase(crud.getTabla(), **false**);

plantillaTratada = plantillaTratada.replaceAll("<<#nombreObjeto#>>", nombreObjeto);

// Nombre de la tabla

plantillaTratada = plantillaTratada.replaceAll("<<#tablename#>>", crud.getTabla());

// lista de todos los campos separados por coma

String campos = StringUtils.*join*(crud.getCampos().keySet(), ',');

plantillaTratada = plantillaTratada.replaceAll("<<#todos\_campos\_tabla#>>", campos);

String todos\_param = "";

String update\_param = "";

**int** index = 1;

**for** (Map.Entry<String, String> entry : crud.getCampos().entrySet()) {

todos\_param += "?";

update\_param += entry.getKey() + " = ?";

**if** (crud.getCampos().size() > index) {

todos\_param += ", ";

update\_param += ", ";

}

index++;

}

plantillaTratada = plantillaTratada.replaceAll("<<#todos\_campos\_tabla\_parametro#>>", todos\_param);

plantillaTratada = plantillaTratada.replaceAll("<<#todos\_campo\_valor#>>", update\_param);

plantillaTratada = plantillaTratada.replaceAll("<<#numero\_campos#>>",

Integer.*toString*(crud.getCampos().size()));

**if** (crud.getCampos() != **null**)

plantillaTratada = trataCampos(crud, plantillaTratada);

**return** plantillaTratada;

}

/\*\*

\* **@param** crud

\* **@param** plantillaTratada

\*/

**private** String trataCampos(DatosCrud crud, String plantillaTratada) {

// buscamos el reemplazo de campos

**int** ocurrenciaCampos = plantillaTratada.indexOf("<<#seccion\_loop\_campos#>>");

**int** finOcurrenciaCampos = plantillaTratada.indexOf("<</#seccion\_loop\_campos#>>");

**while** (ocurrenciaCampos > -1) {

String trataCampos = plantillaTratada.substring(ocurrenciaCampos, finOcurrenciaCampos);

// Quitamos las marcas de bucle de campos

trataCampos = trataCampos.replaceFirst("<<#seccion\_loop\_campos#>>", "");

trataCampos = trataCampos.replaceFirst("<</#seccion\_loop\_campos#>>", "");

String salidaCampos = "";

Integer index = 0;

**for** (Map.Entry<String, String> entry : crud.getCampos().entrySet()) {

String bucle = trataCampos;

bucle = bucle.replaceAll("<<#nombre\_campo#>>", camelCase(entry.getKey(), **false**));

bucle = bucle.replaceAll("<<#Nombre\_campo#>>", camelCase(entry.getKey(), **true**));

bucle = bucle.replaceAll("<<#nombre\_campo\_tabla#>>", entry.getKey());

bucle = bucle.replaceAll("<<#posicion\_campo#>>", (index++).toString());

bucle = bucle.replaceAll("<<#tipo\_campo#>>", entry.getValue());

salidaCampos += bucle;

}

String resultado = plantillaTratada.substring(0, ocurrenciaCampos);

resultado += salidaCampos;

resultado += plantillaTratada.substring(finOcurrenciaCampos + "<</#seccion\_loop\_campos#>>".length());

plantillaTratada = resultado;

// preparamos la siguiente iteraciÃ³n

ocurrenciaCampos = plantillaTratada.indexOf("<<#seccion\_loop\_campos#>>");

finOcurrenciaCampos = plantillaTratada.indexOf("<</#seccion\_loop\_campos#>>");

}

**return** plantillaTratada;

}

Por último, una vez tratada la plantilla se procede a llamar al metodo **salvaFuente**, que creará el fichero en la ruta de filesystem seleccioanda. En este punto se comprueba si la pantilla tienen configurada la opción de *custom mode* y se procede a comprobar si hay ficheros ya generados, en este caso, se sustituye el texto contenido entre las pablaras claves en el texto que estamos generados.

/\*\*

\*

\* **@param** plantilla

\* **@param** crud

\* **@param** pl

\* **@param** salidaTratada

\* **@throws** Exception

\*/

**private** **void** salvaFuente(String plantilla, DatosCrud crud, String pl, String salidaTratada) **throws** Exception {

// En la configuración de platillas el texto antes de "-" es la extensión

// del fichero a generar

**int** ext = plantilla.indexOf("-");

String extension;

**if** (ext > -1)

extension = plantilla.substring(0, ext);

**else**

extension = plantilla;

String nombreFichero= "";

String directoriodestino = "";

**if** (extension.equals("java")) {

directoriodestino = crud.getPathSalida() + '/' + pl.replace(".tpl", "").toLowerCase();

}

**else** {

directoriodestino = crud.getPathSalida();

}

compruebaCarpeta(directoriodestino);

nombreFichero = directoriodestino + '/' + camelCase(crud.getTabla(), **true**);

// La plantilla con nombre "persistencia" llevará

// el nomobre de la tabla, en caso contrario se

// añadira al nombre de la tabla el nombre del fichero.

**if** (!pl.equalsIgnoreCase("persistencia.tpl")) {

nombreFichero += camelCase(pl.replace(".tpl", ""), **true**);

}

File fichero = **new** File(nombreFichero + "." + extension);

// si el fichero que vamos a crear ya existe se buscan las etiquetas custom mode

// dentro del código. estas etiquetas deben estar en la plantilla origen

**if** (fichero.exists()) {

String content = **new** String(Files.*readAllBytes*(Paths.*get*(nombreFichero + "." + extension)));

// la plantilla tiene la posibilidad de customizar la cabecera

**int** inicioEtiqueta = salidaTratada.indexOf("/\*<<#custom\_code\_head#>>\*/");

**if** (inicioEtiqueta > -1) {

**int** finEtiqueta = salidaTratada.indexOf("/\*<</#custom\_code\_head#>>\*/");

// buscamos el texto en el fichero que se genero

**int** iC = content.indexOf("/\*<<#custom\_code\_head#>>\*/");

**if** (iC > -1) {

**int** fC = content.indexOf("/\*<</#custom\_code\_head#>>\*/");

String customHead = content.substring(iC, fC);

salidaTratada = salidaTratada.substring(1, inicioEtiqueta) + customHead

+ salidaTratada.substring(finEtiqueta);

}

}

// customización del cuerpo

inicioEtiqueta = salidaTratada.indexOf("/\*<<#custom\_code\_body#>>\*/");

**if** (inicioEtiqueta > -1) {

**int** finEtiqueta = salidaTratada.indexOf("/\*<</#custom\_code\_body#>>\*/");

// buscamos el texto en el fichero que se genero

**int** iC = content.indexOf("/\*<<#custom\_code\_body#>>\*/");

**if** (iC > -1) {

**int** fC = content.indexOf("/\*<</#custom\_code\_body#>>\*/");

String customHead = content.substring(iC, fC);

salidaTratada = salidaTratada.substring(1, inicioEtiqueta) + customHead

+ salidaTratada.substring(finEtiqueta);

}

}

}

FileOutputStream stream = **new** FileOutputStream(fichero, **false**); // false sobre escribe el fichero

**byte**[] myBytes = salidaTratada.getBytes();

stream.write(myBytes);

stream.close();

}

## Alcance y limitaciones de la solución propuesta

El generador solo contempla la generación de código sobre tablas con clave primaria de un solo campo, es decir, no contempla la posibilidad de tablas con claves primarias múltiples.

Las platillas para el caso de java se van a limitar a generar las clases de acceso a datos, es decir la capa de modelo en una arquitectura Model-Vista-Controaldor

## Descripción de los casos de Prueba

# Ubicación de los programas fuente y explicación del resto de directorios/ficheros anexados

# Manuales de uso/instalación

# Conclusiones.