


GESTIÓN DE CLÍNICAS

COLEGIO OFICIAL DE


VETERINARIOS DE BIZKAIA

Manual de Desarrollador

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	


ÍNDICE

1 -	INTRODUCCIÓN	3
2 -	ARQUITECTURA DE LA APLICACIÓN	4
2.1	TECNOLOGÍA	4
2.1.1	SWING	4
2.1.2	HSQLDB	5
2.2	NOMENCLATURA	6
2.2.1	Paquetes org.attest.....	6
2.2.2	Paquetes org.cv	7
3 -	CLASES DE LA APLICACIÓN	9
3.1	EXTENSIÓN DE CLASES SWING	9
3.1.1	AttestDateJTextField	9
3.1.2	AttestJButton.....	9
3.1.3	AttestJCheckBox	10
3.1.4	AttestJComboBox	11
3.1.5	AttestJTable	12
3.1.6	AttestJTextArea	12
3.1.7	AttestJTabbedPane	13
3.1.8	AttestTimeJTextField	14
3.2	CLASES DE LA APLICACIÓN	15
3.2.1	org.cv.app	15
3.2.2	org.cv.app.model	16
3.2.3	org.cv.core	16
3.2.4	org.cv.ui	19
3.2.5	org.cv.ui.agenda	20
3.2.6	org.cv.ui.almacen.....	20
3.2.7	org.cv.ui.clientes	21
3.2.8	org.cv.ui.config.....	24
3.2.9	org.cv.ui.informes	25
3.2.10	org.cv.ui.tienda.....	26
3.2.11	org.cv.util.....	27
4 -	MODELO E/R	29

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

1 - INTRODUCCIÓN

El presente documento pretende ser una guía para los desarrolladores que se adentren en la modificación del proyecto de Gestión de Clínicas del Colegio de Veterinarios de Bizkaia. En él se comentan en detalle todas aquellas cuestiones que se han considerado interesantes para un programador a la hora de manejar el código fuente generado para la aplicación, haciendo hincapié en todos aquellos aspectos no funcionales pero sí estructurales y de organización de código.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

2 - ARQUITECTURA DE LA APLICACIÓN

2.1 TECNOLOGÍA

Dentro del software libre utilizado, se encuentran, además de los paquetes básicos de java, las librerías de los proyectos SWING-AWT y HSQLDB.

Mientras el primero de ellos ha sido indispensable a la hora de plantear una interfaz gráfica para una aplicación cliente en java, el segundo se ha utilizado como base de datos embebida en el propio programa.

Se ha tratado de que todos los componentes utilizados en el GUI formasen parte del paquete SWING, teniendo la obligación de utilizar AWT cuando la situación así lo ha requerido; en casos en los que no existe un componente SWING análogo, por ejemplo.


A su vez, las librerías HSQLDB han permitido disponer de un servidor de base de datos completamente embebido en la aplicación. Además, permite gestionar las conexiones a bases de datos tanto locales como en servidores ajenos, lo cual agrupa en un solo paquete toda la gestión de acceso a base de datos, sea local o externa.

2.1.1 SWING

Las librerías de SWING se encuentran en el paquete javax.swing y proporcionan todas las clases necesarias (junto con AWT) para la creación y gestión de interfaces gráficas en java.

En el caso particular de este proyecto se han utilizado ventanas, frames, paneles de distinto tipo, campos de texto, etc..., así como gestores de posición (LayoutManagers) para controlar que la disposición de los elementos dentro de los paneles se mantenga dentro de un orden al redimensionar la ventana de ejecución. Utilizar un LayoutManager permite a la aplicación aprovechar todo el espacio en sistemas configurados con distintas resoluciones. Si se hubiera desarrollado basándose en posicionamiento absoluto, se habría perdido esa capacidad.

También se controlan con elementos de SWING la acción sobre los elementos de la aplicación a través de Listeners, de modo que se asocia una acción determinada a los distintos componentes. Gracias a esto es posible interaccionar a través de los diversos controles (combinaciones de teclas, ratón, etc...).

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

2.1.2 HSQLDB


Se trata de un proyecto 100% Java de gestión de bases de datos. Básicamente se trata de un servidor de bases de datos como cualquier otro, con la salvedad de que es posible incluirlo en otras aplicaciones formando parte de ellas y controlando conexiones e instancias de servidor. Más información puede encontrarse en: <http://www.hsqldb.org>

Gracias a esta librería, es posible elegir entre realizar conexiones contra un servidor instanciado en el propio programa o contra un servidor dentro de la red local de ordenadores. Esta opción es realmente interesante, ya que nos permite la posibilidad de ejecutar la aplicación con o sin servidor de base de datos mediante, simplemente, un parámetro de configuración.

De este modo, solamente uno de los ordenadores de la red debe ejecutarse en modo servidor, siendo el resto clientes de éste.

Para funcionar como servidor local se instancia un objeto Server. Después, las conexiones van a través de un objeto Connection que lanza las peticiones al servidor local instanciado o a un servidor remoto que se ha iniciado en la ejecución del programa en otra aplicación.

HSQLDB permite también otros modos de funcionamiento. Una de ellas es especialmente interesante; se trata del modo "fichero". Esto permite realizar una conexión a los datos de una HSQLDB sin necesidad de instanciar un servidor. Sin embargo, al no existir éste, no es posible realizar dos conexiones simultáneas sobre la misma base de datos, requisito de la aplicación desarrollada. Por esta razón, se descartó este modo de funcionamiento desde el primer momento.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

2.2 NOMENCLATURA


El código de la aplicación está organizado en paquetes de manera que resulte sencillo e intuitivo encontrar las clases que se busquen para su consulta o modificación.

Inicialmente se han dividido las clases en dos grandes paquetes:

- org.attest: Se trata de clases genéricas desarrolladas por ATTEST y utilizadas aquí para gestionar las conexiones a la base de datos, java beans, clases extendidas de SWING con alguna funcionalidad extra propicias para la aplicación, etc.
- org.cv: Aquí se engloba todo el código propio de la aplicación de Gestión de Clínicas propiamente dicho, tanto la parte de interfaz gráfica como de funcionalidad.

2.2.1 Paquetes org.attest

- org.attest.db: Contiene las clases de gestión de conexiones y consultas a base de datos.
- org.attest.maestros: se trata del paquete que contiene todas las clases para la realización del módulo genérico de configuración de tablas maestras. Se compone de:
 - org.attest.maestros.bean: Contiene beans genéricos utilizados como representación de tablas maestras de la base de datos.
 - org.attest.maestros.core: Contiene clases para la gestión de las tablas maestras. Estas clases son las que gestionan el alta, edición y eliminado de registros de tablas maestras de la base de datos.
 - org.attest.maestros.model: Contiene modelos utilizables en objetos JTable, JComboBox y JTree. Estas clases de SWING permiten gestionar por separado el contenido de dichos componentes y su visualización. Estas clases gestionan esos contenidos a través de distintos orígenes de datos (base de datos, listado de directorios, etc).
 - org.attest.maestros.ui: Contiene clases que componen la interfaz gráfica para la edición de tablas maestras; listado, alta, edición y eliminado.
- org.attest.ui: Contiene extensiones de clases SWING con funcionalidades extra que ayudan en la implementación de las funcionalidades de la aplicación, con la intención de reutilizar el código en otras aplicaciones. Un ejemplos de este paquete: JAttestTabbedPane; panel con


	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

pestañas similar al JTabbedPane, cuyo funcionamiento se ha depurado con el objetivo de ahorrar memoria en el ordenador en el que se esté ejecutando la aplicación.


- org.attest.ui.imgs: Contiene imágenes genéricas utilizadas en las clases del paquete anterior.
- org.attest.util: Clases de utilidades genéricas.

2.2.2 Paquetes org.cv

- org.cv.app: Contiene las clases de inicio de la aplicación (JApplication y JFrame).
- org.cv.app.imgs: Contiene las imágenes adicionales utilizadas en la aplicación.
- org.cv.app.model: Contiene modelos adicionales para objetos SWING utilizados en la aplicación.
- org.cv.core: Contiene las clases que implementan toda la funcionalidad de la aplicación a nivel de núcleo o funcionalidad. Aquí se encuentran todos los Gestores de la aplicación, tanto de base de datos como de ficheros RTF. Se han separado las clases de interfaz gráfica (org.cv.ui) de las funcionalidades (org.cv.core) para facilitar la localización de los métodos que las implementan.
- org.cv.ui: Contiene las clases con los paneles (pantallas) de introducción.
- org.cv.ui.agenda: Contiene las clases con los paneles (pantallas) utilizados en el módulo de agenda.
- org.cv.ui.almacen: Contiene las clases con los paneles (pantallas) utilizados en el módulo de almacen.
- org.cv.ui.clientes: Contiene las clases con los paneles (pantallas) utilizados en el módulo de clientes.
- org.cv.ui.config: Contiene las clases con los paneles (pantallas) utilizados en el módulo de configuración de la aplicación.
- org.cv.ui.informes: Contiene las clases con los paneles (pantallas) utilizados en el módulo de informes.
- org.cv.ui.tienda: Contiene las clases con los paneles (pantallas) utilizados en el módulo de tienda.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

- org.cv.util: Contiene clases de utilidades propias de la aplicación (no-genéricas) que incluyen clases de constantes, de gestión de paso de un módulo a otro, etc.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

3 - CLASES DE LA APLICACIÓN

Como ya se ha visto, las clases de la aplicación se encuentran divididas en paquetes que aúnan clases con contenidos o finalidades comunes. Pasamos ahora a explicar en detalle las clases de todos los paquetes relacionados con la programación de las funcionalidades del programa.

3.1 EXTENSIÓN DE CLASES SWING

Dentro de los paquetes org.attest, se han extendido algunas clases de SWING para añadirles funcionalidades extra o para gestionar de manera centralizada código que de otro modo quedaría repetido en muchas otras clases (reutilización de código).

Estas son las clases extendidas (paquete org.attest.ui):

3.1.1 AttestDateJTextField

Esta clase extiende el campo de texto de SWING para convertirlo en un campo que admita, exclusivamente, fechas.

Al introducir un valor en el campo de texto, éste debe cumplir con alguno de los formatos de fechas disponibles. En caso de no cumplirse esta condición, una alerta avisa al usuario del error y le remite al campo de texto para su corrección. Los formatos disponibles son: dd/MM/yy, dd/MM/yyyy, ddMMyy, ddMMyyyy, dd-MM-yy, dd-MM-yyyy.

Además, se dispone de un método que devuelve la fecha introducida como un java.sql.Date para ser introducido, directamente, en base de datos.


3.1.2 AttestJButton

Se extiende el botón de SWING para conseguir botones genéricos para abrir, buscar, cancelar, guardar, eliminar, nuevo, modificar y restar.

Con esto se consigue no repetir código cada vez que se quiera un botón de, por ejemplo, guardar, sino que, con los parámetros adecuados en el constructor, se obtiene, directamente, un botón para guardar igual (con el mismo icono) en todas las partes de la aplicación (análogamente para el resto de casos).

Se dispone de tres constructores para esta clase:

- AttestJButton()

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Constructor genérico.

- AttestJButton(Object panelContenedor, Icon icono, String label, String method)

Constructor con el que se define en qué panel está incluido el botón, el icono que queremos que tenga (puede ser null), el texto que queremos en el botón (puede ser null) y el método que queremos que se ejecute al accionar el mismo (puede ser vacío o null).

- AttestJButton(Object panelContenedor, int tipo, String method)


Constructor con el que se define en qué panel está incluido el botón, el tipo de botón (dentro de los enumerados a continuación) y el método a ejecutar al accionarlo (puede ser vacío o null). Los tipos disponibles se definen como enteros públicos estáticos finales en la propia clase:

```
AttestJButton.ABRIR,
AttestJButton.BUSCAR,
AttestJButton.CANCELAR,
AttestJButton.GUARDAR,
AttestJButton.ELIMINAR,
AttestJButton.NUEVO,
AttestJButton.MODIFICAR,
AttestJButton.RESTAR.
```

En caso de utilizar un constructor en el que se pase un método a ejecutar, éste debe ser el nombre de un método público definido en el panel que contiene el botón. Esto es así, ya que se llama a este método de este objeto a través de reflexión, centralizando en esta clase la gestión de acciones sobre el botón.

3.1.3 AttestJCheckBox

Se extiende el JCheckBox de SWING para añadir un estado más. La clase de SWING solamente admite dos estados (seleccionado o no); en esta extensión, se tienen tres: seleccionado, no seleccionado, indiferente (null). Es una clase especialmente indicada para formularios de búsqueda, donde se puede pretender buscar por los que tengan un determinado dato, no lo tengan o que no se busque por ese dato (un ejemplo claro de esto se tiene en el booleano que define al animal como potencialmente peligroso –PPP-, donde se puede buscar aquellos que lo sean, aquellos que no lo sean, o todos independientemente de este dato).

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Se dispone de cuatro constructores para esta clase:

- `AttestJCheckBox()`

Constructor genérico.

- `AttestJCheckBox(String texto)`

Constructor con el que se define el texto asociado al objeto. El estado inicial es el de indiferente.

- `AttestJCheckBox(String text, State initial)`

Constructor con el que se define el texto asociado al objeto y su estado inicial.

- `AttestJCheckBox(String text, Icon icon, State initial)`

Constructor con el que se define el texto asociado al objeto y su estado inicial.

Los estados disponibles se definen como estados públicos estáticos y finales:

`AttestJCheckBox.NOT_SELECTED`

`AttestJCheckBox.SELECTED`

`AttestJCheckBox.DONT_CARE`

3.1.4 **AttestJComboBox**

Se extiende esta clase de SWING para ponerle el mismo tipo de fuente que los campos de texto. Se modifican los constructores para que realicen este cambio respecto al original.


Se dispone de dos constructores para esta clase:

- `AttestJComboBox()`

Constructor genérico.

- `AttestJComboBox(MaestroComboBoxModel model)`

Constructor con el que se define el modelo de datos que va a utilizar el JComboBox.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

3.1.5 AttestJTable

Se extiende esta clase de SWING para centralizar el estilo de todas las tablas utilizadas en una misma clase, amén de añadir diversas funcionalidades que facilitan la implementación de las funcionalidades de la aplicación.

Se dispone de tres constructores para esta clase:

- AttestJTable()

Constructor genérico.

- AttestJTable(TableModel modelo)

Constructor en el que se define el modelo de datos que se utilizará en esta tabla. En particular pueden utilizar los contenidos en org.attest.maestros.model referentes a tablas.

- AttestJTable(TableModel modelo, Object panelContenedor, String metodo)

Constructor en el que se define el modelo de datos a utilizar por la tabla. Además, se define el panel (pantalla) en el que está contenida la tabla y un método a ejecutar cuando se haga doble clic sobre una fila de la misma, o pulse ENTER sobre una fila seleccionada.

Estos dos eventos redundan en la ejecución del método indicado en el panel especificado.


El método debe ser el nombre de un método público del panel que contiene a la tabla.

Además, se han definido dos métodos de exportación de los datos visualizados en la tabla. Por un lado se dispone de una conversión a formato RTF para su inserción en documentos de este tipo, y por otro de una conversión a formato CSV (texto plano con separadores) para la generación de documentos exportables a Excel.

En algunos casos ha sido necesario sumar el contenido total de una columna (precios, número de animales, etc), para lo cual se han creado dos métodos que, dados el nombre de la columna o el número de columna dentro de la tabla (atención a posibles columnas que se mantengan ocultas al usuario), se obtiene la suma de todas las filas de la tabla de esa columna.

3.1.6 AttestJTextArea

Se extiende esta clase para igualar el estilo (márgenes, tipo de letra, etc) con los campos de texto (JTextField) por defecto que cree JSWING. Por defecto los campos de texto de varias

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

líneas tienen un estilo que no es acorde con el resto de la aplicación. Así, con esta extensión, todos los componentes utilizados tienen el mismo estilo; unificado, además, en esta clase.

3.1.7 AttestJTabbedPane

Se extiende esta clase para disponer de un panel (pantalla) de pestañas gestionado de manera dinámica y así ahorrar en memoria.

La idea es que un único panel sea utilizado para todas las pestañas (tabs) de modo que cuando el usuario cambia de pestaña, en realidad esté viendo el mismo panel pero con diferente contenido. De este modo no se tienen tantos paneles en memoria como pestañas haya (aunque sólo se vea uno), sino que se vacía y se rellena un único panel para todas.

Se dispone de dos constructores para esta clase:

- AttestJTabbedPane()

Constructor genérico.

- AttestTabbedPane(String[] titulos, char[] mneponicos, Class clase, int indiceMetodoOnWindowListener, Imagen[] iconos, Color colorTabs)


Constructor con el que se definen las pestañas a mostrar en el JTabbedPane de SWING.

Se definen los títulos y los mnemónicos de cada pestaña, así como los iconos asociados a ellas. Es posible, además, definir el color de las pestañas para que coincida con el del panel (pantalla).

La clase indicada es la clase que, extendiendo a su vez de ésta, es la que define los métodos para el cambio de pestañas. Cada uno de estos métodos vacía la pestaña y después la completa con los componentes adecuados a cada una de ellas, de modo que un cambio de pestaña es, en realidad, una llamada a uno de estos métodos, a los que se accede por reflexión.

Los métodos definidos en los paneles (pantallas) con pestañas, deben llamarse "tabbedXXXX" y ordenarse alfabéticamente de modo que cada uno se corresponda con la pestaña asociada. Así, por ejemplo, con el abecedario como guía, "tabbedA" mostraría la primera pestaña, "tabbedB" la segunda, etc...

En algunos casos en los que es necesario refrescar la misma pestaña que se está mostrando (al editar datos maestros, por ejemplo), se permite pasar al constructor cuál de


	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

estos métodos es el que hay que ejecutar al recuperar el foco (los maestros se editan en ventanas nuevas que, al cerrarse, devuelven el foco al panel de pestañas). Este valor se puede modificar a través de su “setter”.

3.1.8 AttestTimeJTextField

Se extiende el campo de texto de SWING para poder disponer, análogamente al caso de las fechas, de un campo de horas del día.

Al introducir un valor en el campo de texto, éste debe cumplir con alguno de los formatos de horas disponibles. En caso de no cumplirse esta condición, una alerta avisa al usuario del error y le remite al campo de texto para su corrección. Los formatos disponibles son: hmm, hhmm, hh:mm.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

3.2 CLASES DE LA APLICACIÓN

Dentro del paquete org.cv se encuentran todas las clases que implementan la interfaz gráfica del programa, así como el conjunto de clases que conforman su funcionalidad.

A continuación se detallan estos paquetes y clases.

3.2.1 org.cv.app

- AppPrincipal

Clase principal de la aplicación. Es la clase que lanza todo el resto del programa a través de su método main.

Controla la apertura de la ventana nueva, su posición, el estilo general de la aplicación, y el cierre de la misma; propia de toda aplicación java cliente.

- frmPrincipal

Clase que, extendiendo de JFrame (SWING), es la clase inicial de la aplicación.

Forma parte de la interfaz gráfica al ser el frame donde se van colocando el resto de paneles (pantallas). En este aspecto, instancia también la barra de tareas (en el método jblnit típico de toda clase SWING) con los botones para cambiar de módulo en la aplicación, así como las combinaciones de teclas (Alt+Nº) para estas acciones (a través de un método independiente llamado teclado()).

A nivel funcional, contiene un booleano estático por cada módulo para controlar si el usuario accedió al mismo previamente. En ese caso no se le vuelve a pedir la contraseña dentro de la misma sesión de ejecución del programa (al cerrar la aplicación, la siguiente vez que se abra se empieza de nuevo) Estos booleanos se llaman en forma de "conectado"+Nombre del módulo (conectadoAgenda, conectadoalmacen, etc...) y pueden modificarse a través de sus respectivos accesorios (setter y getter), estáticos también (para no arrastrar la instancia del JFrame por toda la aplicación).

También gestiona la lectura de los ficheros de configuración disponibles (config.properties, por ejemplo) para el usuario, cargándolos en objetos estáticos de tipo Configuration, en principio privados, pero accesibles a través del método público getConfiguracion(); estático también.

Estos ficheros pueden modificarse a través del módulo de configuración o manualmente. En caso de hacerlo manualmente, los cambios serán activos en la siguiente ejecución del programa (requiere reiniciar, si la aplicación está activa) y si no se modifican después a través del módulo de configuración, que lo rescribirá perdiendo los datos manuales.

En base a los datos configurables, esta clase se encarga, también, de crear la instancia de servidor HSQLDB correspondiente. En caso de configurarse el programa para su ejecución en forma de servidor, un objeto privado `org.hsqldb.Server` es instanciado y utilizado como servidor de base de datos. En caso contrario, las conexiones, que también se controlan desde aquí, se realizan contra la máquina remota indicada en la configuración. Las conexiones son accesibles a través del método público `getConexion()`.

El objeto servidor es un thread independiente de modo que, su ejecución o no, no afecta al desarrollo de la ejecución del programa. Tanto el servidor como la conexión a base de datos son eliminados al cerrar la aplicación a través del método público `finalizar()`.

3.2.2 **org.cv.app.model**

- `jVacioModel`

Clase que implementa un modelo de datos vacío utilizable en tablas que aún no han sido rellenadas con datos reales.

3.2.3 **org.cv.core**


- `GeneradorQueries`

Clase que implementa la generación de consultas a base de datos a partir de unos datos dados. Contiene métodos estáticos que retornan las consultas en forma de String en base a datos en Maps, índices de la base de datos en forma de Integer, etc para cada una de las funcionalidades que requieren consultar la base de datos. Las funcionalidades van desde un buscador en las tablas Cliente y/o Paciente hasta obtener las razas de una especie.

De esta manera, se centralizan todas las consultas de la aplicación en una sola clase para su más cómoda modificación y acceso.

- `GestorAgenda`

Clase intermedia entre la interfaz gráfica y el acceso de a base de datos.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Contiene métodos estáticos para obtener, insertar o editar, y eliminar datos de una cita del módulo Agenda.

Básicamente, recoge los parámetros de la interfaz gráfica, genera la consulta mediante el GeneradorQueries visto anteriormente y la lanza contra la base de datos a través del controlador de base de datos (DBManager) y sus métodos disponibles.

- GestorClientes

Clase intermedia entre la interfaz gráfica y el acceso de a base de datos.

Contiene métodos estáticos para obtener, insertar o editar, eliminar datos de un cliente del módulo Clientes.

Básicamente, recoge los parámetros de la interfaz gráfica, genera la consulta mediante el GeneradorQueries visto anteriormente y la lanza contra la base de datos a través del controlador de base de datos (DBManager) y sus métodos disponibles.

- GestorPacientes

Clase intermedia entre la interfaz gráfica y el acceso de a base de datos.

Contiene métodos estáticos para obtener, insertar o editar, eliminar datos de un paciente del módulo Paciente. También contiene métodos para gestionar los tratamientos aplicados a un paciente cuando éstos se realizan indirectamente a través de un servicio de una consulta de ese paciente. En este caso el usuario no introduce los datos del tratamiento, sino que se toman los asociados al servicio prestado.


Básicamente, recoge los parámetros de la interfaz gráfica, genera la consulta mediante el GeneradorQueries visto anteriormente y la lanza contra la base de datos a través del controlador de base de datos (DBManager) y sus métodos disponibles.

Todos los métodos relacionados con el apartado Consultas se encuentran en la siguiente clase.

- GestorConsultas

Clase intermedia entre la interfaz gráfica y el acceso de a base de datos.

Contiene métodos estáticos para obtener, insertar o editar, eliminar datos de una consulta en el submódulo Consultas.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Básicamente, recoge los parámetros de la interfaz gráfica, genera la consulta mediante el GeneradorQueries visto anteriormente y la lanza contra la base de datos a través del controlador de base de datos (DBManager) y sus métodos disponibles.

Excepcionalmente, se han añadido aquí métodos para la generación de documentos en formato RTF asociados a las funcionalidades del submódulo Consultas.

- GestorInformes

Clase intermedia entre la interfaz gráfica y el acceso de a base de datos.

Contiene un método estático para la exportación en formato CSV (texto plano) de los datos asociados a las funcionalidades del módulo Informes.

- GestorProductos

Clase intermedia entre la interfaz gráfica y el acceso de a base de datos.

Contiene métodos estáticos para obtener y eliminar datos de un producto del módulo Tienda. El alta y edición de productos se realiza como una tabla de datos maestros. También se tienen dos métodos para actualizar el número de unidades disponibles de ese producto en la tienda.

Básicamente, recoge los parámetros de la interfaz gráfica, genera la consulta mediante el GeneradorQueries visto anteriormente y la lanza contra la base de datos a través del controlador de base de datos (DBManager) y sus métodos disponibles.


- GestorRtf

Clase dedicada a la generación de documentos en formato RTF.

Contiene métodos estáticos para generar distintos documentos basados en una serie de diferentes plantillas y con diverso destino final en el sistema de archivos manejados por la aplicación.

Básicamente, recibe la información necesaria para recoger los datos de un determinado paciente y su dueño e insertarla en una plantilla existente. El reemplazo se realiza mediante la búsqueda de determinados tags (&CLI_NOMBRE&, por ejemplo) en el contenido de la plantilla y su sustitución.

Eventualmente puede recibir parte de código RTF generado en una tabla (AttestJTable comentada anteriormente) para exportar sus datos, así como el total de un importe, etc.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

- GestorServicios

Clase intermedia entre la interfaz gráfica y el acceso de a base de datos.

Contiene métodos estáticos para insertar y eliminar datos de un servicio de una consulta. Debido a que un servicio puede tener asociado un tratamiento, aquí nos encontraremos también la inserción de tratamientos de un paciente.

Básicamente, recoge los parámetros de la interfaz gráfica, genera la consulta mediante el GeneradorQueries visto anteriormente y la lanza contra la base de datos a través del controlador de base de datos (DBManager) y sus métodos disponibles

3.2.4 org.cv.ui

En este paquete se contienen todas las interfaces gráficas (paneles / pantallas) que se muestran al usuario para utilizar el programa.

Todos los paneles pertenecientes a este paquete contienen un método llamado **jblnit()** que es llamado en cada constructor y es quien se encarga de poner y disponer todos los elementos (botones, textos, combos, etc) en el panel.


Además, en la mayoría de estas clases se tiene un método llamado **teclado()** que gestiona el uso del teclado para interactuar con el programa. Existen combinaciones de teclas que permiten editar maestros, accionar botones, llevar el foco a un determinado elemento, etc., que son definidos para cada panel en este método.

Cada módulo (sus paneles asociados) está insertado en un paquete con su nombre, quedando a este nivel de paquetería el panel de introducción (panel inicial de la aplicación).

- pnlIntro

Clase que, extendiendo de JPanel (SWING), implementa el panel inicial con los botones de acceso a los módulos de la aplicación y una presentación a modo de inicio del programa.

Se dispone de un método público estático para cargar el módulo de configuración que es independiente del resto de los módulos, debido a que debe poder cargarse aunque aún no se haya configurado la aplicación (la primera vez que se ejecute la aplicación aún no existe configuración definida, que debe poder definirse en éste módulo de configuración).

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

3.2.5 org.cv.ui.agenda

- pnlAgenda

Clase que, extendiendo de JPanel (SWING), implementa el panel inicial asociado al módulo de Agenda.

Básicamente es un calendario en el que seleccionar un día para pasar a ver el listado de citas de ese día (acceder al panel pnlCitasDia).

- pnlCitasDia

Clase que, extendiendo de JPanel (SWING), implementa el panel asociado al listado de citas de un día en el módulo Agenda.

Contiene métodos para ver el detalle de una cita (acceder al panel pnlFichaCita) seleccionada para verla y/o modificarla, eliminar una cita, o añadir una nueva a través del GestorAgenda comentado anteriormente.

- pnlFichaCita

Clase que, extendiendo de JPanel (SWING), implementa el panel asociado al listado de citas de un día en el módulo Agenda.

Contiene métodos para guardar los datos de una cita seleccionada a través del GestorAgenda comentado anteriormente.

3.2.6 org.cv.ui.almacen


- pnlAlmacen

Clase que, extendiendo de JPanel (SWING), implementa el panel inicial asociado al módulo de Almacén.

Básicamente, se trata de un buscador donde se listan los productos del almacén y puede crearse uno nuevo y editar y eliminar uno existente a través del GestorAlmacen comentado anteriormente.

Además, para facilitar la funcionalidad, se ha creado un método que permite restar una determinada cantidad de un producto seleccionado (acceder al panel pnlRestarProducto).

- pnlRestarProducto

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Clase que, extendiendo de JDialog (SWING), implementa el panel asociado a la resta de un determinado número de elementos de la cantidad habida de un producto seleccionado.

Básicamente, permite restar la cantidad indicada a través del GestorAlmacen comentado anteriormente.

3.2.7 org.cv.ui.clientes

- pnlClientes

Clase que, extendiendo de JPanel (SWING), implementa el panel inicial asociado al módulo de Clientes.

Básicamente, consiste en un buscador de clientes / pacientes que muestra un listado de los resultados obtenidos.

Desde el listado es posible acceder a los datos del cliente o del paciente (acceder a pnlFichaCliente y pnlFichaPaciente respectivamente), así como insertar nuevos clientes a través del GestorClientes comentado anteriormente.

Se dispone de un método público llamado **paciente()**, dejado como público ya que al hacer doble clic o pulsar ENTER sobre un paciente de la lista, por defecto, se accede a la ficha del paciente (el listado es un AttestJTable, comentado anteriormente).


A través de la función teclado() se definen atajos por teclado para llevar el foco al componente indicado (campo de texto, combo, etc...), así como accionar los botones existentes en el panel.

También es posible acceder a los paneles de edición de los datos maestros (principalmente, los contenidos de los JComboBox) seleccionables en el buscador. Al modificar estos datos maestros se actualiza el panel completo, borrando los valores existentes.

- pnlFichaCliente

Clase que, extendiendo de JPanel (SWING), implementa la visualización y edición de los datos propios de un cliente, al igual que se muestra un listado de los pacientes que tiene el cliente seleccionado.

Es posible editar y guardar los datos, eliminar el cliente (requiere tres confirmaciones y eliminará toda la información de sus pacientes también, así como la información que de estos depende gracias a las restricciones ON CASCADE DELETE de la base de datos HSQLDB), y añadir o eliminar pacientes al / del cliente.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

También se permite crear un presupuesto de una consulta. Al generarla, se puede imprimir un documento en formato RTF que será guardado en el lugar indicado de la estructura de directorios del programa; pero no será guardada en base de datos.

Todo ello a través del GestorClientes comentado anteriormente.

Al cancelar (método cerrar()), se vuelve al panel de clientes inicial (buscador).

- pnlFichaConsulta

Clase que, extendiendo de JDialog (SWING), implementa la visualización y edición de los datos propios de una Consulta, al igual que se muestran dos listados; uno de los servicios que tiene la consulta seleccionada, y otro de los documentos asociados a la consulta.

Es posible editar y guardar los datos, eliminar la consulta (requiere confirmación), y añadir o eliminar servicios a la consulta (acceder a pnlNuevoServicio).

También se permite crear un historial de una consulta. Al generarlo, se puede imprimir un documento en formato RTF que será guardado en el lugar indicado de la estructura de directorios del programa. Además del historial, es posible generar (e imprimir) facturas y albaranes que serán guardados en el mismo directorio. Todos estos documentos se pueden eliminar mediante la acción correspondiente.

Es posible añadir nuevos documentos (fotos, videos, imágenes, documentos y cualquier tipo de documento soportado por el sistema operativo en general) que se abrirán con el programa asociado a cada extensión. Dicha ejecución se hace con el siguiente comando:


```
Runtime.getRuntime().exec("cmd /C start " + path);
```

Como se puede comprobar, la ejecución solamente funcionará en sistemas operativos Windows NT/2000/XP. Para poder hacerlos funcionar en otros, esta línea de código deberá ser modificada.

Todo ello gestionado a través del GestorConsultas comentado anteriormente.

- pnlFichaPaciente

Clase que, extendiendo de AttestTabbedPane (anteriormente comentado), implementa la visualización y edición de los datos propios de un paciente, separados en un panel con varias pestañas.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Cada pestaña es generada por un método asociado a ella (tabbedA(), tabbedB()...), cuya gestión se realiza a través de los métodos implementados en AttestTabbedPane. Cada pestaña tiene, igualmente, su función teclado() correspondiente.

Contiene todos los métodos que se utilizan en cualquiera de las pestañas, ya que no existe un panel en cada pestaña, sino que un mismo panel es borrado y completado a cada cambio de pestaña (el panel y sus métodos son siempre los mismos, cambia el contenido del mismo).

Es posible, en la primera pestaña, editar y guardar los datos del paciente, y eliminarlo (requiere dos confirmaciones y eliminará toda la información de sus consultas, etc) de la base de datos (sus documentos quedan en la estructura de directorios de la aplicación).

En la segunda se tienen los documentos asociados al paciente, como un historial generable a partir de una plantilla. Además, es posible añadir todos los documentos que se quieran y aparecerán listados como parte de los documentos del paciente. Cada uno de estos documentos es abierto por la aplicación predefinida en el sistema operativo para cada extensión (.doc con Word, .htm con Explorer, etc...). En particular, si es un archivo llamado FOTO.*, se tiene un botón especial para verlo.

En la tercera se tiene un listado con los tratamientos periódicos aplicados al paciente, y un segundo listado con los tratamientos caducados. Se permite introducir nuevos tratamientos y eliminar los existentes, actualizándose cada vez ambos listados.

La cuarta pestaña presenta un listado con las consultas realizadas. Es posible eliminar y añadir consultas, pero la acción por defecto es ver el detalle de una consulta (acceder a pnlFichaConsulta) para su edición.


Todo ello a través de los GestorPaciente, GestorRTF y GestorConsultas comentados anteriormente.

Al cancelar (método cerrar()), se vuelve al panel de detalle del cliente asociado al paciente (pnlFichaCliente).

- pnlNuevoPresupuesto

Clase que, extendiendo de JDialog (SWING), implementa la creación de presupuestos de consultas.

Básicamente, consta de un listado con los servicios presupuestados, que pueden eliminarse y añadirse (accediendo a pnlNuevoServicio) e imprimirse en un documento con formato RTF.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Los presupuestos creados no se guardan en base de datos.

- pnlNuevoServicio

Clase que, extendiendo de JDialog (SWING), implementa la creación de un nuevo servicio para una consulta seleccionada o para un nuevo presupuesto (cada uno tiene su constructor).

Básicamente, consta de un listado en forma de árbol de todos los servicios existentes en base de datos. Este árbol es navegable y al seleccionar un servicio (doble clic o pulsar ENTER), se completan una serie de campos asociados al nuevo servicio. El usuario puede modificar estos datos antes de proceder a guardarlos. Uno de ellos es el precio, que aparece ya con el descuento asociado al tipo de cliente que sea el cliente seleccionado (el cliente del paciente).

En caso de que el servicio consuma un producto del almacén, se resta la cantidad utilizada, y en caso de consumir un tratamiento, se inserta en los tratamientos del paciente.

Al cerrar se actualiza el listado de servicios del que se llamó (nuevo presupuesto o ficha de consulta).

3.2.8 org.cv.ui.config

- pnlConf


Clase que, extendiendo de JPanel (SWING), implementa la edición de los parámetros configurables de la aplicación, tanto a nivel local como global. Parámetros como la habilitación de módulos disponibles son locales (se guardan en el config.properties de la aplicación), mientras que otros, como las password de acceso a cada módulo son fijas para todas las instancias que se conecten a un mismo servidor (se guardan en el ordenador que hace de servidor HSQLDB).

Básicamente, se muestran todos los valores disponibles, con la posibilidad de editarlos y guardarlos.

En los cambios que impliquen un cambio en el acceso a base de datos debe reiniciarse la aplicación para que los cambios tengan efecto (se exige reiniciar, ya que HSQLDB no refresca las conexiones como lo hacen otros servidores tipo MySQL).

El resto (módulos disponibles, contraseñas de módulos, etc...) están disponibles sin necesidad de reinicio y en la misma ejecución del programa.

Al cancelar (método cerrar()) se vuelve al panel inicial de la aplicación (pnlIntro).

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

3.2.9 org.cv.ui.informes

- pnlFiltroInformes

Clase que, extendiendo de JPanel (SWING), implementa un panel común para la definición de los criterios de búsqueda del buscador de pnlInformes.

Básicamente, consta de una serie de campos de texto y combos con los que definir criterios de búsquedas sobre campos de cliente y de paciente.

Cada uno de sus componentes accesibles mediante teclado tiene accesorios (setters y getters), y se dispone de un método que recupera los datos introducidos en forma de Map para su uso en el GeneradorQueries anteriormente comentado.

Forma parte del pnlInformes y no aparece nunca por sí solo.

- pnlInformes


Clase que, extendiendo de AttestTabbedPane, implementa el panel del módulo Informes, en el que se dispone de tres pestañas para la generación de diversos tipos de documentos asociados a diversos tipos de búsquedas.

Al igual que en pnlFichaPaciente, el cambio de pestañas se gestiona a través de los métodos heredados de AttestTabbedPane.

La primera pestaña consta de un buscador de clientes o pacientes en base a los tratamientos de los pacientes y la fecha en que se hicieron. Pueden imprimirse varios documentos asociados a la búsqueda (cartas y etiquetas en formato RTF a partir de una plantilla para cada cliente y el listado completo en formato CSV).

La segunda pestaña consta de un buscador de pacientes en base a las consultas de los pacientes en el que se encuentra incluido un pnlFiltroInformes y de un listado con los resultados obtenidos al aplicar los criterios indicados. Cuando se encuentran resultados que implican un importe, se muestra un total (suma de los importes que hay en la AttestJTable) de los resultados encontrados.

Puede listarse por diferentes criterios (incluyendo búsquedas agrupadas por un determinado campo de cliente o paciente), pudiendo exportarse el listado obtenido en formato CSV (texto plano).

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

La tercera pestaña muestra un nuevo buscador sobre los datos de las consultas y con un `pnlFiltroInformes` incluido, pero con la diferencia de que se listan solamente pacientes y agrupaciones por campos del cliente o del paciente.

En caso de buscarse por paciente, es posible exportar el listado completo a formato CSV. En las búsquedas agrupadas por un campo, es posible, además, generar una carta en formato RTF a partir de una plantilla.

Todas las funcionalidades se realizan a través del `GestorInformes` comentado anteriormente.

3.2.10 `org.cv.ui.tienda`

- `pnlFichaConsultasTienda`

Clase que, extendiendo de `JPanel` (SWING), implementa la visualización y posible edición de los datos de consultas realizadas en el módulo de Tienda (cliente y paciente de ID igual a uno).

Es posible modificar y guardar los datos de la consulta, así como modificar el listado de servicios asociados añadiendo o eliminado servicios. En este caso particular no se consumen tratamientos (no se añaden tratamientos al paciente "VENTAS MOSTRADOR"), pero sí se consumen productos. Por lo demás es idéntico al `pnlFichaConsulta`.

Toda la funcionalidad se gestiona a través del `GestorConsultas` comentado anteriormente.

- `pnlTienda`


Clase que, extendiendo de `JPanel` (SWING), implementa el panel inicial del módulo de Tienda.

Básicamente, se trata de un buscador sobre las consultas realizadas sobre el paciente "VENTAS MOSTRADOR" del cliente "VENTAS MOSTRADOR", que, por defecto, se asocian al cliente de id y al paciente de id iguales a uno (los primeros).

Se trata del caso especial en el que se venda algo en la tienda sin ser una consulta propiamente dicha, sino una "consulta en tienda". Así pues, se tiene un código casi idéntico a las consultas de un paciente.

Estas consultas tienen la particularidad de que gastan productos, pero no tratamientos (en la tienda se venden productos, pero no tratamientos).

Es posible añadir una nueva consulta y editar y eliminar las existentes en el listado mostrado.

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

Todo ello a través del GestorConsultas comentado anteriormente.

3.2.11 org.cv.util

- Constantes

Clase de constantes utilizadas en la aplicación.

Básicamente, contiene constantes públicas estáticas y finales para su uso como nombres de tablas de la base de datos, colores utilizados, etc. Se centralizan todas estas constantes en una clase para facilitar la propagación del cambio de sus valores por toda la aplicación.

- ControlAccesoModulo

Clase que controla el paso de un módulo a otro en base a la inserción o no de una contraseña.

Básicamente, contiene un método estático para controlar el acceso a un módulo mediante la inserción de una clave de acceso una única vez. Una vez introducida la clave correcta no se vuelve a pedir en el resto de la ejecución del programa (si se cierra y se vuelve a abrir sí que se pide, de nuevo, una vez).

Este método se llama **control()** y tiene como parámetro un identificador de módulo (su nombre) y el panel a mostrar. En caso de poder accederse, el panel del módulo solicitado es insertado en el frame principal de la aplicación para su visionado.


Para casos particulares de vuelta atrás, se dispone de otro método estático que no pide claves nunca y es utilizable a nivel de programación para cambios controlados por código sin necesidad de control de contraseñas.

- RTFParser

Clase utilizada en el parseo de las plantillas en formato RTF al generar documentos en este formato.

Básicamente, consta de un método estático que, dado un String que empieza y acaba con el carácter "&" (esto define un tag utilizable en las plantillas para que se introduzca en su lugar un dato de un cliente o paciente) y devuelve el tag limpio de posible código RTF intermedio que pudiera contener.

Se ha comprobado que en la edición habitual de una plantilla, Word (y probablemente otros editores de formato RTF) inserta código RTF adicional en medio de los tags, especialmente si se

	PROYECTO: Gestión de Clínicas CLIENTE: Colegio Oficial de Veterinarios de Bizkaia
MANUAL DE DESARROLLADOR	

edita el formato (negrita, colores, etc) o se borra y vuelve a poner, etc. Para limpiar este código sobrante, se tiene el método cleanPossibleTag().

4 - MODELO E/R

El modelo entidad/relación de la base de datos creada para la aplicación es especificado por el siguiente diagrama:

