|  |  |  |
| --- | --- | --- |
| Documento de Estándares | 28 de septiembre  2009 | |
|  | | Construcción de software |

## Universidad de Talca, Facultad de ingeniería

**Índice**

**Diccionario…………………………………………………………………………………………………………………………3**

**Estándar de interfaz………………………………………………………………………………………………..…………..4**

**Estructura de la interfaz………………………..………………………..…………………………..……..…….5**

**Descripción de los componentes de la interfaz………………………..………………………………….6**

**Opciones de módulo………………………..………………………..………………………..………….………..7**

**Botones principales de acceso a los módulos………………………..………………………….………..8**

**Botones rápidos………………………..…………………………………………………….…………….………..9**

**Datos………………………..………………………..………………………..………………………..………………9**

**Componentes…………..………………………..………………………..………………………..………………12**

**Registro……………………..….……………………..………………………..………………………..………………13**

**Editar……………………..………………………..………………………..………………………..………………..15**

**Búsqueda……………………………………………………………………………………………………………..19**

**Eliminar y Purgar…………………………………………………………………………………………………..23**

**Advertencia del Sistema……………………………………………………………………………….……….25**

**Estándar conexión………………………..………………………..………………………..………………………………..28**

**Estándar SQL y consultas..…..………………………..………………………..………………………..………………35**

**Estándar de Consultas a la Base de Datos……………………………………………………………….35**

**Estándar Sql……………………………………………………..………………………………………………….39**

**Creación de tablas y Poblamiento…………………………………………………………………………..41**

**Modelo relacional………………………..………………………..………………………..………………………………..43**

**Modelo base de datos………………………..………………………..………………………..…………………………44**

**Estándar de documentación de código actionScript………………………..……………………….…………..45**

**Estándar de documentación de código java………….………………………..……………………….………….49**

**Estándar de Programación………………………………………………………………………………………………..59**

**Código de Java……………………………………………………………………………………………………….59**

**Código de Action Script……………………………………………………………………………………………71**

**Código de Flex…………………………………………………………………………………………………......77**

**Estándar del usuario basado en IEEEStd 1063-2001………………………………………………………….....79**

**Referencias………………………………………………………………………………………………………………………..88**

**Anexo………………………………………………………………………………………………………………………………….89**

Diccionario

**Veterinaria:** se refiere al establecimiento del doctor Bonati, en el que se ofrecen los servicios de: Peluquería, Hotelería, Pet-Shop y Clínico (Pabellón, Policlínico).

**Cliente:** Es una persona que usa un servicio de la clínica veterinaria y que no necesariamente tiene una o mas mascotas.

**Mascota:** Es un animal, que dispone de datos tales, como nombre, fecha de nacimiento y está asociado al registro de un cliente.

**Ficha:** Es un documento digital que posee cada mascota, en el que aparecen todos los registros de atenciones realizadas y servicios realizados. Ordenados por fecha y por servicio. Esta ficha está asociada a un cliente, de ahí que se distingue una ficha para cada mascota que tenga un cliente.

**Usuario de Sistema:** Es un trabajador de la clínica, que usa el sistema y que cuenta con una cuenta de usuario.

**Registrar:** Acción de  guardar un conjunto de datos en el sistema.

**Editar:** Acción que modifica uno o varios datos para posteriormente guardar los datos.  
  
**Borrar:** Registro que se borra temporalmente del sistema.

**Purgar:** Registro que se borra completamente de la base de datos o sistema.

**Visualizar:** Acción en la que se representan de alguna forma específica los datos en la pantalla o archivo.  
  
**Mantener:** Acción de borrar, registrar y editas uno o varios datos.

**ID Mascota**: Numero identificador único, generado por el sistema, para identificar de manera única a una mascota en el sistema.

**RUT Cliente**: Referencia única hacia un cliente.

**Cruza**: Se refiere al apareamiento de animales.

**Listar**: Acción de mostrar los datos al usuario y que no están necesariamente ordenados.  
  
**Eliminar:** Datos que no se muestran en el programa pero que si están en la base de datos.

**Policlínico:** Lugar donde la mascota se hospeda después de haber sido intervenida quirúrgicamente.

**Pabellón:** Lugar donde se hacen las intervenciones quirúrgicas a las mascotas

Estándar de Interfaz

1.- Luchar por la consistencia o coherencia:Este punto está enfocado a las diferentes terminologías y estructuras de pantallas para situaciones o escenarios semejantes, estos deben utilizarse en anuncios, menús y pantallas de ayuda, esto enfocado a los nombres de botones, utilizar los mismos nombres para la misma acción, distribución de las pantallas en cuanto a contenidos.

2.- Utilizar frecuentemente accesos directos:Esto consiste en presentar botones o link de acceso rápido para acceder a las diferentes prestaciones que tiene el sistema, permitir al usuario utilizar estos accesos se fundamenta en que en la medida que el tiempo en que usa el software aumenta, también lo hace la necesidad de este por reducir los pasos o acciones para realizar acceder a cierto servicio, disminuyendo el tiempo de respuesta y aumentando el ritmo de la interacción. En este punto se pueden utilizar acrónimos y abreviaturas, teclas de función, comandos ocultos, entre otros.

3.- Ofrece comentarios informativos:Por cada operador de acción, debe haber algún sistema de retroalimentación. Para acciones frecuentes y de menor uso, la respuesta puede ser modesta, mientras que para los poco frecuentes y las principales acciones, tales como errores graves del sistema, que puedan producir perdidas de información, la respuesta debería ser más sustancial.

4.- Diseño de diálogo para producir la clausura:Este punto está enfocado a la retroalimentación informativa, que se produzca en la conclusión de un grupo de acciones, esto da a los usuarios la satisfacción de logro, una sensación de alivio, esto en parte indica al operador que debe continuar con las acciones, o que se debe preparar para realizarlas.

5.- Ofrece una manipulación de errores simples:En la medida de lo posible, diseñar el sistema para que el usuario no ocasione un grave error. Si aparece un error, el sistema debería ser capaz de detectar el error y ofrecer de manera sencilla y comprensible una manera para identificar el dicho error.

6.- Permitir un fácil retroceso de las acciones**:** Esta característica alivia la ansiedad, ya que el usuario sabe que los errores se pueden deshacer, esto lleva a que el usuario tenga menos miedo al explorar el software y las diferentes opciones desconocidas para el usuario. Las unidades de reversibilidad pueden ser una sola acción, una entrada de datos, o un grupo de acciones.

7.- Apoyo interno a un enfoque de control total**:** Los usuarios experimentados desean el sentido de que están a cargo del sistema y que el sistema responde a sus acciones. Para esto es necesario diseñar el sistema para que los usuarios inicien las acciones en lugar de que inicien las respuestas. Y que este pueda ingresar y salir de diversas opciones de manera fácil, sin pasar por procesos intermedios.

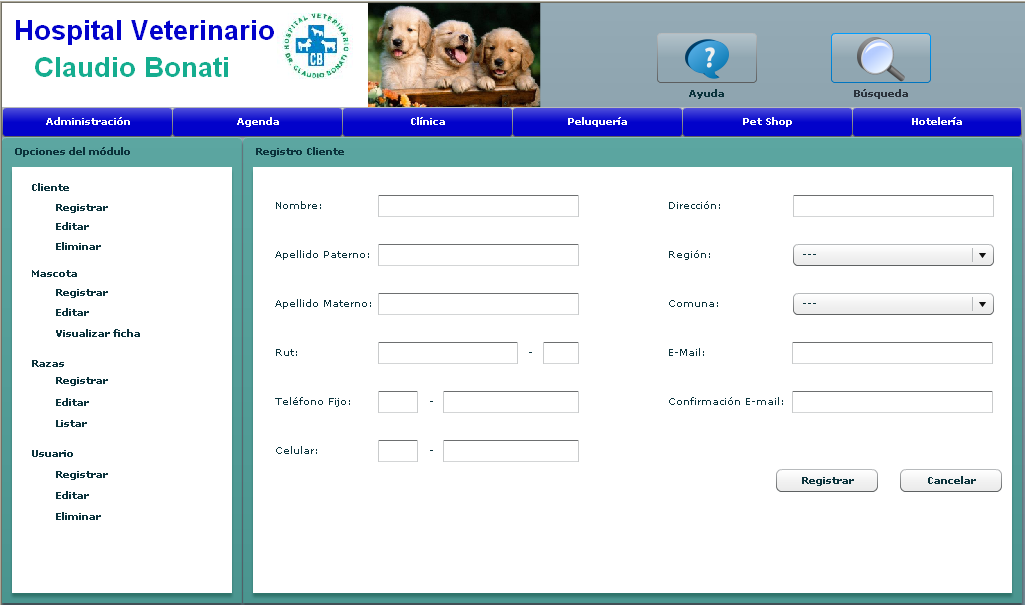
8.- Reducir la carga de la memoria a corto plazo:Este punto presenta vital importancia debido a las limitaciones de recursos humanos, en cuanto al procesamiento de la información en la memoria a corto plazo, esto exige que se muestren de manera sencilla, y que el software no ponga a prueba la memoria del usuario, si no que este le facilite al usuario el recordar la información, mediante un escenario de interacción.

# Estructura de la interfaz



**Imagen 1**

En la **imagen 1** se puede apreciar que la interfaz que se ha desarrollado para este sistema posee una división de los paneles de acuerdo a la información que estos posean, en los cuales se pueden observar claramente el panel de botones rápidos encontrados en la parte superior derecha de la interfaz, los botones de menú de los módulos que se encuentra en la parte superior del panel de datos y opciones del menú.



**Imagen 2**

En la imagen 2 se puede observar cómo está estructurado el software, y la forma en que se presentan los datos y los menús correspondientes a cada sub-modulo, esto al momento de seleccionar dicho modulo. Es por esto que se ha definido la mayor parte de la forma de estructurar la visualización de datos, el ingreso de datos, y en qué lugares deberán aparecer partes importantes de una interfaz como imágenes, botones, etc.

La forma de utilizar el software, explicado rápidamente, sería hacer login con la cuenta de usuario correspondiente. Luego el deberá seleccionar un modulo del sistema, que son los que se encuentran en la barra horizontal, después de esto aparecerá en el panel izquierdo Opciones del modulo, los diferentes servicios que este modulo presentara, se selecciona un servicio y en el panel de datos, aparecen los datos a ingresar o las diferentes visualizaciones. Y a continuación se seleccionara el botón correspondiente a la acción que se desea realizar del panel de Opciones de selección.

## Descripción de los componentes de la interfaz

En el software se definieron sectores que no deberían modificarse al momento de ingresar a cada sub-modulo, para lograr una consistencia en el software y evitar que al abrir cada sub-modulo aparezca una estructura diferente de los datos. Es por esto que se definió que solo deberán cambiar sectores internos de los paneles: Opciones del módulo, Datos, Opciones de selección.

## Opciones de módulo



**Imagen 3**

En este panel, se encontraran los servicios o procesos principales que presenta el modulo que fue seleccionado, los cuales si tienen sub-categorías, deberían aparecer indentados, como se muestra en la **imagen 3**, esto servirá de gran forma para retroceder rápidamente a secciones anteriores o servicios anteriores, ya que mientras se esté en el mismo modulo, estarán visibles las opciones anteriores y las de otros servicios relacionados con el modulo, por lo que si el usuario desea ingresar a otro servicio de este modulo, lo podrá hacer de manera rápida. Esto cubre en gran parte la regla número 2 de los anteriormente mencionados, el de presentar accesos directos. De igual forma se pueden definir teclas o combinaciones de teclas que permitan acceder rápidamente a cada servicio.

Para este panel el tamaño será el siguiente:

* Width: 241 pixeles.
* Heigth: 466 pixeles
* X: 0
* Y: 134

Es importante destacar que la forma en que se mostraran los datos es la siguiente: El titulo del panel debe ser el del modulo al cual pertenece, y los servicios que se prestan se deben dividir, agrupándolos en sustantivos si se puede decir, por ejemplo:



De esta forma las acciones deben ir indentados mas a la derecha de una “cosa” o sustantivo a la cual se le está aplicando la acción, de esta forma se entenderá fácilmente que se editara el cliente o que se registrara una mascota.

Ya que el profesor Narciso Cerpa menciono que debe ser coherente lo que va en este panel, en cuanto a que los paneles aparezcan en el mismo tipo de palabra, aquí en este ejemplo los paneles deberían aparecer al presionar los verbos que son editar o registrar, etc. Y a su vez los sub-servicios que aparecen indentados deberían ser todas acciones.

En este panel se cubre también el punto número 7 de las reglas de oro, ya que estas opciones aparezcan constantemente, permite al usuario ingresar y salir fácilmente de las opciones.

# Botones principales de acceso a los módulos



**Imagen 4**

Es donde se encuentra el acceso directo a los diferentes módulos, es desde aquí donde se accederá a los diversos módulos del sistema, además estos botones, o panel aparecen en todos los módulos en la misma posición, por lo que es una rápida forma de cambiar de un modulo a otro. Esto permite con una estructura más clara, permitiendo una consistencia en el sistema, al ingresar a cada modulo, punto 1 de las reglas de oro, y sirven a la vez de acceso directo, ya que al ingresar a un subsistema no deberá retroceder todo lo que ingreso, sino que bastara con presionar un botón, y podrá ingresar a otro subsistema, confirmando previamente la acción. Cubriendo el punto número 2 de las reglas de oro. También cumple el punto 7, permitiendo un apoyo interno al control total, ya que permite salir o ingresar a un sistema de forma fácil, sin procesos intermedios.

# Botones Rápidos

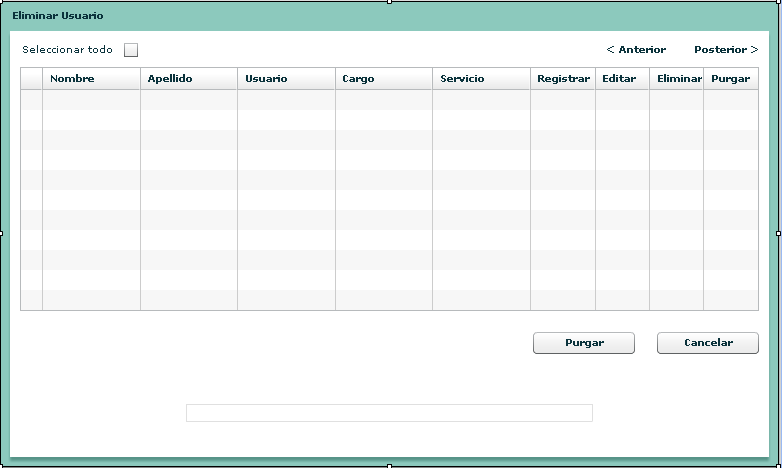


**Imagen 5**

En la **imagen 5** se muestran los diferentes botones que se encuentran en la parte superior derecha de las pantalla. Esto cubre los puntos 1, ya que siempre están en pantalla, permitiendo acceder fácilmente a ellos y siempre estarán en la misma posición aunque cambien de módulos. Se utiliza como acceso directo a algunos servicios, y permite salir fácilmente de otros, o acceder a esas opciones fácilmente cubriendo los puntos 2 y 7.

Son los que en cierta forma, presentan funcionalidades más generales del sistema, las cuales serán funciones de ayuda y búsqueda de datos importantes dentro del sistema.

# Datos



**Imagen 6.1**



**Imagen 6.2**

En este panel aparecerán los diversos campos a visualizar o en los cuales se ingresaran datos, aquí deberían cambiar los datos que se entregan o ingresan en los diferentes módulos, pero se debe seguir un patrón especifico, en cuanto a la forma de presentar la forma de multi-selección, las imágenes, los botones, etc. El titulo de este panel debe ser el nombre de la acción o servicio que está prestando, en este caso se está realizando una Eliminar Usuario.

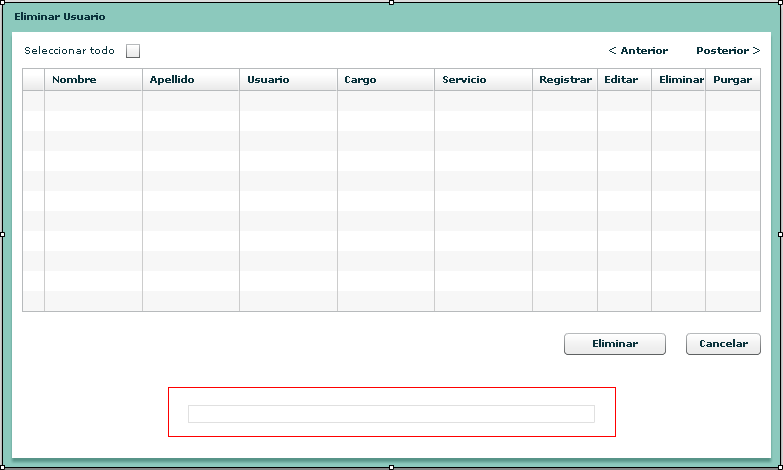
En el caso de presentar muchos datos a ingresar o mostrar, el panel se dividirá en dos partes iguales, en las presentando dos columnas de datos, para aprovechar de mejor forma el espacio.

En cuanto a los campos en los que se deba hacer una selección múltiple, esta deberá representarse con un combo box. A excepción de cuando se puedan seleccionar más de una opción, pues aquí se utilizaran otros componentes, que permitan esto, como CheckBox o RadioButton.

Si se presentan imágenes deberán aparecer en el lado derecho superior de éste panel, es decir deberá dividir el panel en dos, y utilizar la división derecha para visualizar la imagen.

Para los ingresos de datos comunes se utilizara el componente TexInput, que presenta Flex. Y para el ingreso de comentarios si es que se necesita, se utilizo TextArea. Mientras que para seleccionar una fecha se utilizo el DateField.

Como en este panel pueden existir diferentes botones, para visualizar o agregar datos, y que no son tan importantes para ir en el panel de selección, se creó un botón como componente, el cual será utilizado en el interior de este panel, para evitar que existan diferentes modelos y tamaños de botón, además se creó de un porte adecuado para una palabra bastante larga para evitar que después se pierda la palabra y aparezca “visuali…” por ejemplo. Y se definió que este botón no debe llevar imagen, solo llevaran imagen los del panel de opciones de selección. El botón creado se puede ver en la **imagen 6.2.** Y todas estas estructuras o patrones se pueden observar en la **imagen 6.1** Al definir las diferentes estructuras que deben ir dentro de este panel, y la forma en que debe aparecer por pantalla los datos, se cubre claramente el punto 1 de las reglas de oro, el cual es el de luchar por la consistencia.



**Imagen 8**

En la parte inferior de la componente de datos se alojaran los mensajes del sistema que serán los siguientes: los datos fueron guardado con éxito, modificado con éxito. Este sector del software cubre claramente las reglas numero 3 y 4 de las reglas de oro, ya que se presentan los resultados de operaciones al concluir, y además información relacionada con el proceso que se está llevando a cabo, como ayuda posiblemente.

Este mensaje tendrá el siguiente formato que se deberá seguir para que cada uno de los paneles posea una consistencia, los parámetros de este “Label” serán los siguientes:

* **Width: 407**
* **Height: 17**
* **X: 169**
* **Y: 372**

El mensaje deberá ir centrado dentro del “label” y los mensajes estándares dentro de este menú serán los siguientes:

**Ingreso El o La (nombre del registro) Con Éxito**

Por ejemplo:

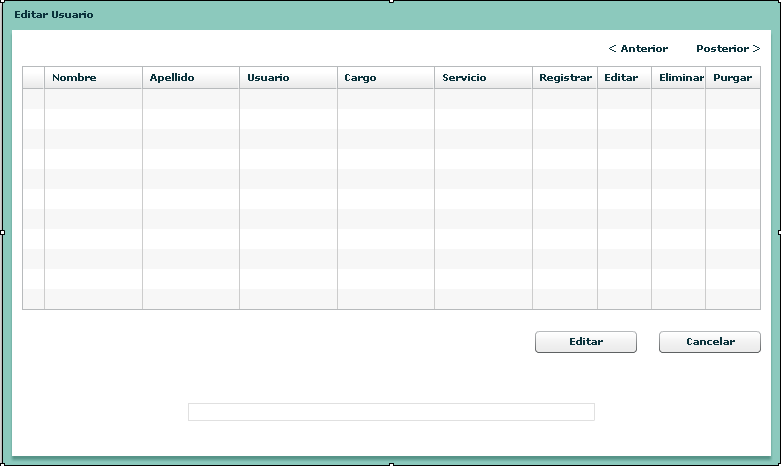
Mensaje [0]: Ingreso El Cliente Con Éxito.

Mensaje [1]: Ingreso La Cita Con Éxito.

Mensaje [2]: Ingreso La Mascota Con Éxito.

Mensaje [3]: Ingreso El Usuario Con Éxito.

Todos estos mensajes irán al interior de un actionScript para que sean utilizado por cada uno de los programadores, en los cuales se podrán agregar nuevos mensajes pero siguiendo el formato anteriormente mencionado, el nombre el archivo actionScript será llamado “Mensajes”.



**Imagen 9**

En la imagen 9 se muestra como se representaran las relaciones 1 a 1, 1 a n y n a n. La forma que tendrán los botones y el lugar en donde preferentemente irán.

# Componentes

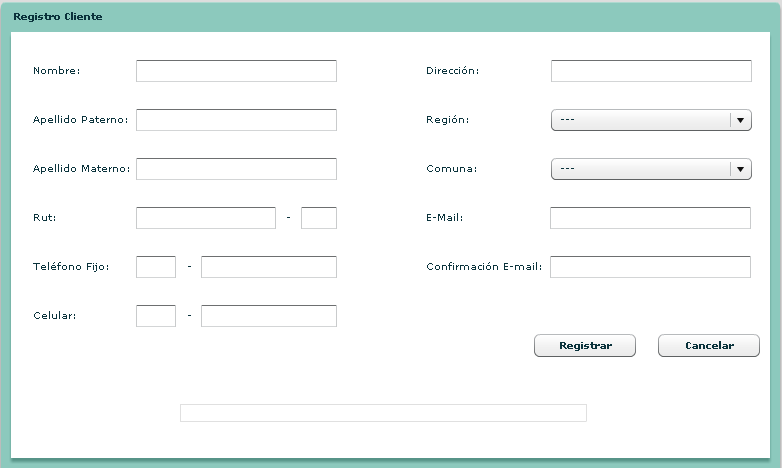
Cada una de las componentes tendrá los siguientes tamaños:

* Width: 779
* Height:383

El borde de cada una de las componentes deberá poseer los siguientes colores que son representativos para la clínica veterinaria:

Border color : #: 15ad8f

# Registro



Para la realización de los registros se deberán seguir el siguiente formato:

1. En la parte superior de la componente ira el nombre del registro expresado de la siguiente forma: “Registro (nombre del registro)” y el nombre deberá poseer tipo de letra “Verdana”, tamaño: “10” y el color a utilizar será: “# 0b333c”.
2. Todos los TextInput tendrán que tener un mismo tamaño el cual será de width: 201 e identados de igual forma y deberán tener una separación entre cada TextInput desde arriba hacia abajo de 49 pixeles.
3. En los TextInput cuando se requiera el ingreso solamente de un parámetro serán definidos a través de expresiones regulares que nos restringirá el acceso de valores no deseados, los que serán implementados de la siguiente forma:
   * Solamente String

<mx:TextInput restrict="A-Z a-z" x="125" y="28" width="201" id="nombre" maxChars="12"/>

**restrict="A-Z a-z":** Este será el código que deberá ser modificado para poder obtener cualquier expresión regular que se requiera por ingreso.

1. Cada uno de los TextInput deberán tener un máximo de carácter de ingreso, respetando el número de caracteres determinados en la base de datos, para tener una consistencia en los datos.

<mx:TextInput restrict="A-Z a-z" x="125" y="28" width="201" id="nombre" maxChars="12"/>

**maxChars=”12”:** Este será el código que se requerirá para poder darle un máximo de carácter a las entradas de teclado.

1. Cuando sea necesario registrar los apellidos de un usuario irán divididos de dos formas, primero será requerido el **apellido paterno** y luego el **apellido materno** del usuario.
2. Para el requerimiento del Rut se seguirá el siguiente formato, será dividido en dos TextInput donde en primer lugar se requerirá el Rut y en segundo lugar será pedido el código verificador del Rut, como serán mostrado en la siguiente imagen:



1. Para la petición de un teléfono fijo se deberá seguir el siguiente formato, será dividido en dos TextInput en donde en primer lugar deberá ir ingresar el código de la ciudad y en segundo lugar el numero que será presentado en la siguiente imagen:



1. Para la petición de un celular se deberá seguir el siguiente formato, será dividido en dos TextInput en donde en primer lugar ira el cogido del celular y en segundo lugar ira el numero de celular, que será presentado en la siguiente imagen:



1. Cada vez que se solicite el email de un usuario deberá existir una confirmación del email, que permita poder validar su ingreso, el email no podrá ser copiado del anteriormente ingresado por el usuario.
2. Los botones a utilizar en esta componente serán los siguientes, los cuales deberán ir alojados en la parte inferior de esta:

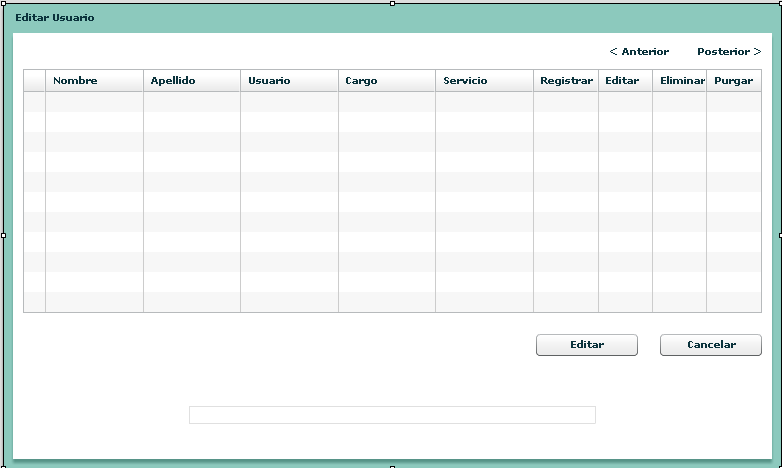
|  |  |  |
| --- | --- | --- |
| **Botones** | **Acciones** | **Formato** |
|  | Realizar el registro de los datos a la base de datos del sistema y luego volverá a la componente principal del servicio que se está utilizando. | Width: 102  Height:23  Corner Radius: 6  X: 523  Y: 302 |
|  | Vuelve a la componente principal del menú utilizado. | Width: 102  Height:23  Corner Radius: 6  X: 647  Y: 302 |

Al lado izquierdo ira alojado el “botón registrar” y el lado derecho ira el “botón cancelar”, los cuales llevaran una distancia de 22 pixeles.

## Editar

Poseerá dos pantallas para poder editar los datos del sistema, los cuales serán la siguiente:

1. En primer lugar se seleccionara el listado de datos que se desee editar, pero esta opción permitirá solamente seleccionar un solo listado no como el caso de la eliminación y purgación, a continuación será mostrada la componente a utilizar:



Para la realización del editar, los datos serán cargados automáticamente desde la base de datos una vez seleccionada la opción desde el menú del modulo, por lo cual se deberán seguir el siguiente formato:

1. En la parte superior de la componente ira el nombre del editar expresado de la siguiente forma: “Editar (nombre del editar)”. El nombre deberá poseer tipo de letra “Verdana”, tamaño: “10” y el color a utilizar será: “# 0b333c”.
2. Deberá llevar un “DataGrid” que le permitirá al usuario poder visualizar cada uno de los datos almacenados en la base de datos de un formulario seleccionado, en la primera columna de datagrid ira incorporado un checkbox que le permitirá al usuario poder seleccionar el dato que desea editar, como será mostrado a continuación:



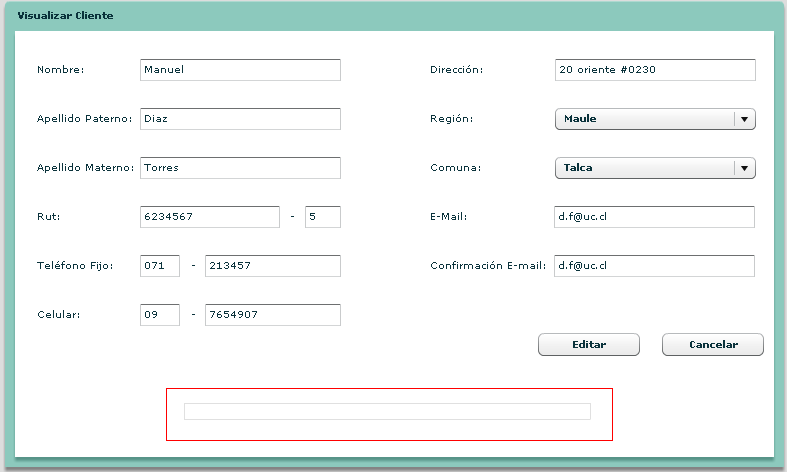
Al interior del dataGrid solamente serán mostrados 11 datos en una primera parte con su respectivo checkBox, porque esta componente tendrá opciones de avanzar o retroceder en búsqueda de un objetivo que se desee editar, cada una de esta acciones serán realizadas con los “LinkButton” llamados “Posterior” y “Anterior”, en todas las funcionalidades de “editar” podrán ser seleccionado solamente un checkBox a la vez.

1. Los botones a utilizar en esta componente serán los siguientes, que deberán ir alojados en la parte inferior de la componente:

|  |  |  |
| --- | --- | --- |
| **Botones** | **Acciones** | **Formato** |
|  | Al momento de presionar el botón el sistema arrojara un mensaje “pop up” que consultara por la acción de editar los datos y aparecerán las siguientes opciones: “Yes” cambiara de componente para editar los parámetros que se deseen y en caso contrario se volverá a la componente principal del menú. | Width: 102  Height:23  Corner Radius: 6  X: 523  Y: 302 |
|  | Vuelve a la componente principal del menú utilizado. | Width: 102  Height:23  Corner Radius: 6  X: 647  Y: 302 |

Al lado izquierdo ira alojado el “botón editar” y el lado derecho ira el “botón cancelar”, los cuales llevaran una distancia de 22 pixeles.

1. En segundo lugar se mostrara un formulario con el listado seleccionado en la etapa anterior en donde se podrán modificar cada uno de los parámetros que se deseen modificar y que puedan ser modificables por el usuario:



Para la realización del editar, los datos serán cargados automáticamente desde la base de datos en el formulario del modulo que se desea modificar, por lo cual se deberán seguir el siguiente formato:

1. En la parte superior de la componente ira el nombre del editar expresado de la siguiente forma: “Editar (nombre del editar)”. El nombre deberá poseer tipo de letra “Verdana”, tamaño: “10” y el color a utilizar será: “# 0b333c”.
2. Llevara el mismo formulario de la etapa de registro, que serán formados por TextInput que tendrán cargado cada uno de los datos obtenidos de la base de datos, en donde se podrán modificar algunos datos como por ejemplo:
   * + Teléfono Fijo (Cliente).
     + Fecha (Cita).
     + E-mail(Cliente).
   * No se pueden modificar las claves primarias de los objetos, porque aparecerán dentro de los textInput como no editables.
     + Rut(Cliente).
     + Raza(Mascota).
3. Los botones a utilizar en esta componente serán los siguientes, que deberán ir alojados en la parte inferior de la componente:

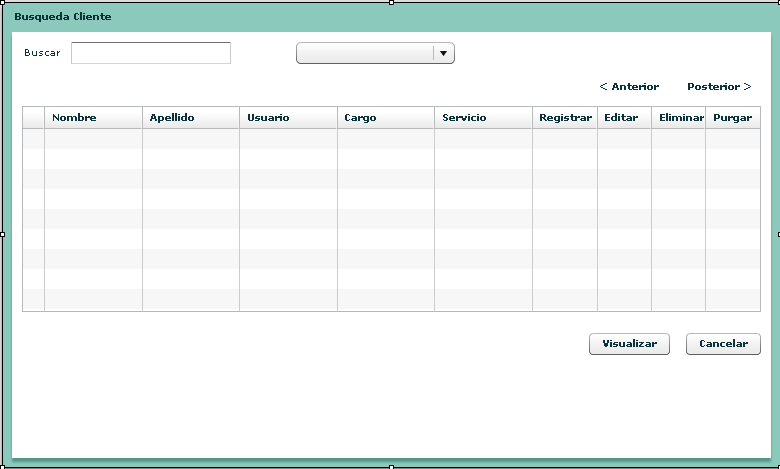
|  |  |  |
| --- | --- | --- |
| **Botones** | **Acciones** | **Formato** |
|  | Al momento de presionar el botón el sistema arrojara un mensaje “pop up” que consultara por la acción de editar los datos y aparecerán las siguientes opciones: “Yes” editara cada uno de los parámetros modificados pro el usuario al interior de un formulario y una vez efectuada esta operación volverá al menú principal del servicio, y en caso contrario se volverá a la componente principal del menú. | Width: 102  Height:23  Corner Radius: 6  X: 523  Y: 302 |
|  | Vuelve a la componente principal del menú utilizado. | Width: 102  Height:23  Corner Radius: 6  X: 647  Y: 302 |

Al lado izquierdo ira alojado el “botón editar” y el lado derecho ira el “botón cancelar”, los cuales llevaran una distancia de 22 pixeles.

## Búsqueda

Poseerá dos tipos de pantallas que le permitirá al usuario poder visualizar los datos buscados, que serán explicados a continuación:

1. En primer lugar nos encontramos con un panel que nos permitirá buscar un registro al interior de la base de datos como será mostrado a continuación:



Para la realización de una búsqueda, primero que nada el usuario deberá seleccionar el tipo de dato que desea buscar al interior del “ComboBox” y una vez seleccionada la opción, el usuario podrá ir escribiendo al interior de la TextInput donde serán cargados automáticamente desde la base de datos en el dataGrid a medida que el usuario va escribiendo en el textInput, por lo cual se deberán seguir el siguiente formato:

1. En la parte superior de la componente ira el nombre de la búsqueda expresado de la siguiente forma: “Búsqueda (nombre de la Búsqueda)”. El nombre deberá poseer tipo de letra “Verdana”, tamaño: “10” y el color a utilizar será: “# 0b333c”.
2. En la parte superior izquierda ira un textInput en donde se introducirán los datos que el usuario desee buscar, este tendrá el siguiente formato:

* Width: 169 pixeles.
* Height: 23 pixeles.

1. Al lado del textInput ira situado un comboBox que tendrán una separación de 56 pixeles, en este comboBox irán cada uno de los parámetros que se desea buscar en el modulo seleccionado y tendrá las siguientes medidas:
   * Width: 169 pixeles.
   * Height: 23 pixeles.
2. Deberá llevar un “DataGrid” que le permitirá al usuario poder visualizar cada uno de los datos encontrados en la base de datos con los parámetros introducidos por el usuario, en la primera columna de dataGrid ira incorporado un checkbox que le permitirá al usuario poder seleccionar el dato que desea visualizar posteriormente, como será mostrado a continuación:



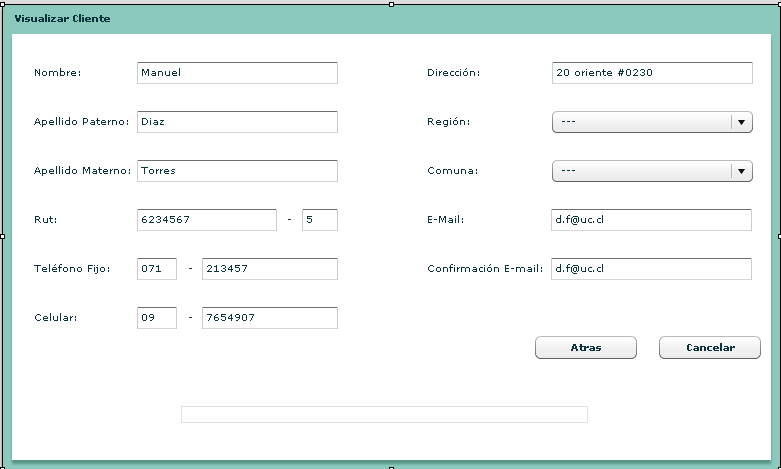
Al interior del CheckBox solamente serán mostrados 9 datos en una primera parte, porque esta componente tendrá opciones de avanzar o retroceder en búsqueda de un objetivo que se desee visualizar, cada una de esta acciones serán realizadas con los “LinkButton” llamados “Posterior” y “Anterior”.

1. Los botones a utilizar en esta componente serán los siguientes, que deberán ir alojados en la parte inferior de la componente:

|  |  |  |
| --- | --- | --- |
| **Botones** | **Acciones** | **Formato** |
|  | Al momento de presionar el botón el sistema cambiara automáticamente de pantalla para que el usuario pueda visualizar más detalladamente los datos buscados. | Width: 102  Height:23  Corner Radius: 6  X: 523  Y: 302 |
|  | Vuelve a la componente principal del menú utilizado. | Width: 102  Height:23  Corner Radius: 6  X: 647  Y: 302 |

Al lado izquierdo ira alojado el “botón visualizar” y el lado derecho ira el “botón cancelar”, los cuales llevaran una distancia de 22 pixeles.

1. Este componente nos permitirá poder visualizar todos los datos del listado anteriormente seleccionado, que será el siguiente:



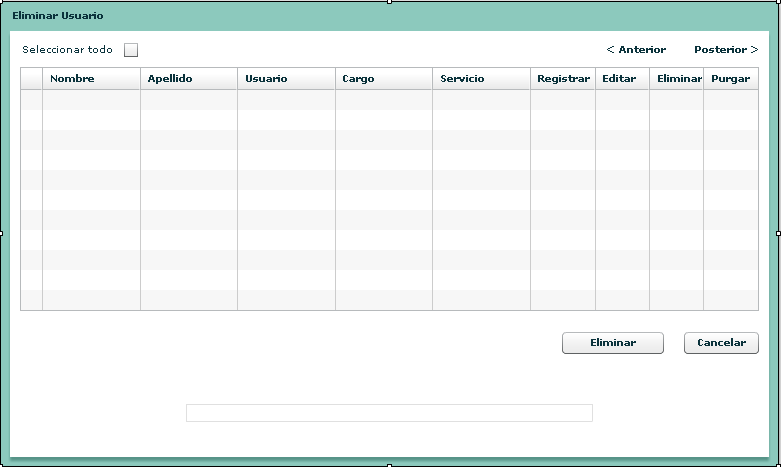
Aquí solamente nos permitirá poder visualizar los datos del listado seleccionado anteriormente sin poder realizar ningún tipo de modificación en los datos, el formato será descrito a continuación:

1. En la parte superior de la componente ira el nombre de la visualización expresado de la siguiente forma: “Visualizar (nombre del Visualizar)”. El nombre deberá poseer tipo de letra “Verdana”, tamaño: “10” y el color a utilizar será: “# 0b333c”.
2. Cada uno de los textInput serán cargado desde la base de datos, pero no podrán ser modificado por el usuario al momento de adquirir algún tipo de dato al interior de la componente.
3. Los botones a utilizar en esta componente serán los siguientes, que deberán ir alojados en la parte inferior de la componente:

|  |  |  |
| --- | --- | --- |
| **Botones** | **Acciones** | **Formato** |
|  | Al momento de presionar el botón el sistema cambiara automáticamente de a la pantalla anteriormente vista para tener la opción de visualizar un nuevo listado. | Width: 102  Height:23  Corner Radius: 6  X: 523  Y: 302 |
|  | Vuelve a la componente principal del menú utilizado. | Width: 102  Height:23  Corner Radius: 6  X: 647  Y: 302 |

Al lado izquierdo ira alojado el “botón atrás” y el lado derecho ira el “botón cancelar”, los cuales llevaran una distancia de 22 pixeles.

## Eliminar y Purgar



Para la realización de la eliminación o purgación, los datos serán cargados automáticamente una vez seleccionada la opción desde el menú del modulo, por lo cual se deberán seguir el siguiente formato:

1. En la parte superior de la componente ira el nombre de la eliminación o purgación expresado de la siguiente forma: “Eliminar (nombre de eliminación)” y “Pugar (nombre de purgación)”. El nombre deberá poseer tipo de letra “Verdana”, tamaño: “10” y el color a utilizar será: “# 0b333c”.
2. En la parte superior izquierda de la componente llevara un “CheckBox”, en la cual deberá ir el modo de seleccionar todos los datos, que serán posibles de eliminar o purgar de la base de datos del sistema, se aplicara de la siguiente forma:



1. Deberá llevar un “DataGrid” que le permitirá al usuario poder visualizar cada uno de los datos almacenados en la base de datos de un formulario seleccionado, en la primera columna de datagrid ira incorporado un checkbox que le permitirá al usuario poder seleccionar el dato que desea eliminar, como será mostrado a continuación:



Al interior del CheckBox solamente serán mostrados 11 datos en una primera parte, porque esta componente tendrá opciones de avanzar o retroceder en búsqueda de un objetivo que se desee eliminar o purgar, cada una de esta acciones serán realizadas con los “LinkButton” llamados “Posterior” y “Anterior”.

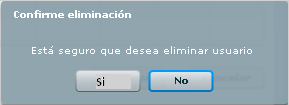
1. Los botones a utilizar en esta componente serán los siguientes, que deberán ir alojados en la parte inferior de la componente:

|  |  |  |
| --- | --- | --- |
| **Botones** | **Acciones** | **Formato** |
|  | Al momento de presionar el botón el sistema arrojara un mensaje “pop up” que consultara por la eliminación de los datos y aparecerán las siguientes opciones: “Yes” eliminara los datos seleccionados y volverá al menú del modulo utilizado y en caso contrario se volverá a la componente principal del menú. | Width: 102  Height:23  Corner Radius: 6  X: 523  Y: 302 |
|  | Al momento de presionar el botón el sistema arrojara un mensaje “pop up” que consultara por la purgación de los datos y aparecerán las siguientes opciones: “Yes” purgara los datos seleccionados y volverá al menú del modulo utilizado y en caso contrario se volverá a la componente principal del menú. | Width: 102  Height:23  Corner Radius: 6  X: 523  Y: 302 |
|  | Vuelve a la componente principal del menú utilizado. | Width: 102  Height:23  Corner Radius: 6  X: 647  Y: 302 |

Al lado izquierdo ira alojado el “botón eliminar” y el lado derecho ira el “botón cancelar”, los cuales llevaran una distancia de 22 pixeles.

## Advertencia Del Sistema

A continuación se mostrara un ejemplo de los mensajes de las advertencias del sistema producidos al interior del sistema, cabe destacar que los mensajes de pop up serán utilizados para las acciones de editar, purgar y eliminar:



Formato del mensaje de advertencia:

Esta seguro que desea eliminar (nombre registro)

O

Esta seguro que desea purgar (nombre registro)

O

Esta seguro que desea editar (nombre registro)

Ejemplos de tipo de advertencias:

Advertencia [0]: Esta seguro que desea eliminar la mascota.

Advertencia [1]: Esta seguro que desea editar el usuario.

Advertencia [2]: Esta seguro que desea purgar la ficha.

Advertencia [3]: Esta seguro que desea eliminar la cita

Advertencia [4]: Esta seguro que desea editar el usuario.

Advertencia [5]: Esta seguro que desea purgar el cliente.

Este tipo de pop up llevara dos tipos de opciones que deberá seleccionar el usuario, que serán las siguientes:

1. “Si”: Eliminara, Pugara o editara los datos que fueron seleccionados por el usuario y volverá al menú principal utilizado por el modulo.
2. “NO”: volverá al menú principal utilizado por el modulo seleccionado.

## Validación textInput

Para la validación de un textInput, flex nos entrega múltiples herramientas que nos permitirá poder avisarle al usuario de posibles errores que ocurran al interior del sistema, sin la necesidad de codificar mucho al interior del programa flex, la visualización de esta herramienta es observada de la siguiente manera:



Para poder realizar esta aplicación al interior de flex se deberá incorporar el siguiente código:

## Validación String

<mx:StringValidator id="email2Validator" requiredFieldError="Ingrese La Confirmación De E-Mail" source="{email2}" property="text" />

Llevara los siguientes parámetros que serán importantes configurar para el buen funcionamiento de la funcionalidad:

**Id:** Será el identificador del StringValidator.

**requiredFieldError:** Aquí ira el mensaje que será mostrado al usuario de acuerdo a la información faltante al interior del TextInput.

**Source:** Llevara el id del TextInput que desea que se le incorpore el mensaje anteriormente incorporado a este.

Al momento de utilizar este tipo de validor se nos permite poder marcar el marco del TextInput con color rojo para orientar al usuario donde se encuentra el error dentro de la componente.

## Validación E-Mail

**<mx:EmailValidator id="emailValidator" requiredFieldError="Complete su email" source="{email}" property="text" invalidCharError="Caracteres inválidos" invalidDomainError= "El dominio no es válido"**

**missingAtSignError="Necesita una @ en su email" missingPeriodInDomainError= "Necesita un nombre de dominio" />**

Esta validación llevara los siguientes parámetros, para la realización de una configuración correcta del validador:

**Id:** Será el identificador del StringValidator.

**requiredFieldError:** Aquí ira el mensaje que será mostrado al usuario de acuerdo a la información faltante al interior del TextInput.

**Source:** Llevara el id del TextInput que desea que se le incorpore el mensaje anteriormente incorporado a este.

**missingAtSignError:** Nos permite corroborar que al interior del TextInput exista la “@”, para seguir el formato de un email, a continuación será mostrado como se ve reflejado este mensaje:



**missingPeriodInDomainError:** No permitirá poder corroborar que al interior del TextInput exista el dominio de un email como por ejemplo “.cl o com”, que serán mostrado a continuación:



Estándar Conexión

En Flex, se crearan unas clases equivalentes, a las clases entidad de la capa 2, para enviarlos desde la capa de presentación los objetos, serializando con esto las entradas captadas de la interfaz.

En Java, se reciben los objetos serializados y mediante las clases de control, se comunica con una clase de la capa 3 que gestiona la conexión con la base de datos.

## Archivos necesarios y organización

La organización de los archivos es la siguiente:

Flex:

* Paquete *transferObjects*: Se crea todas las que son equivalentes a las clases entidad. Dichas clases solo deben ser declarados los atributos. Su instanciación es decir los objetos, serán enviados a Java.

Java:

* Paquete *dao*: Posee una clase que establece la conexión de Jboss con la BD. Sera utilizada por todas las clases de la capa de base de datos.
* Paquete *dao.postgres*: tiene las clases “mantener” las cuales disponen de los métodos para inserción / búsqueda / edición de datos sobre la BD. Cada método contiene sentencias SQL.
* Paquete *services:* Posee las clases de control, que instancian las clases entidad y que instancian las clases de los paquetes *dao.postgres y dao.*
* Paquete *transferObject*: Clases entidad, correspondiente a la capa 2.
* Directorio *WebContent/WEB-INF/flex*: Existe un archivo llamado *remoting-config.xml*  el cual deberá tener registrado todas las clases del paquete *services.*

Servidor:

* Directorio *jboss-4.0.5.GA/server/default/deploy:* Existe un archivo llamado *DBConnectionServer-ds.xml* se deberán configurar los siguientes tags
  + <connection-url>
  + <user-name>
  + <password>

## Normas y procedimientos

Flex: transferObject:

**Creación**: Nuevo -> ActionScript Class

**Convención de nombres**: Primera letra mayúscula

Ejemplo: Cliente.as

**Contenido**: No posee constructor ni métodos, solo atributos.

**Ejemplo:**

**package** transferObjects

{

[**Bindable**]

[RemoteClass(alias=**"transferObjects.Cliente"**)]

**public** **class** Cliente

{

**public** **var** rut:String;

}

}

Java: dao:

No se modifica. Contiene el archivo: DBConnectionManager.java

Java: dao.postgres:

**Creación**: Nuevo -> Class

**Convención de nombres**: Primera letra mayúscula, finalizado con “DB”,

Ejemplo: ClienteDB.java

**Contenido**: Constructor con parámetro <conection> y métodos que realizan acción mantener.

**Ejemplo:**

package dao.postgres;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import transferObjects.Cliente;

public class ClienteDB

{

PreparedStatement insert;

PreparedStatement selectAll;

public PersonDB(Connection connection)

{

try

{

insert = connection.prepareStatement(query);

}

catch (SQLException e)

{

e.printStackTrace();

}

}

public int addCliente(Cliente c){

}

}

Java: dao.service

**Creación**: Nuevo -> Class

**Convención de nombres**: Primera letra mayúscula, finalizado con “Service”,

Ejemplo: ClienteService.java

**Contenido**: Sin constructor, con métodos que realizan acción mantener.

**Ejemplo:**

package services;

import java.sql.Connection;

import java.sql.SQLException;

import dao.DBConnectionManager;

import dao.postgres.ClienteDB;

import transferObjects.Cliente;

public class PersonService

{

public int insertCliente(Cliente cliente)

{

int result=0;

try

{

Connection connection=DBConnectionManager.getConnection();

ClienteDB cdb= new ClienteDB (connection);

result= cdb.insertCliente(cliente);

connection.close();

} catch (SQLException e)

{

e.printStackTrace();

}

return result;

}

Java:transferObjects:

**Creación**: Nuevo -> Class

**Convención de nombres**: Primera letra mayúscula.

Ejemplo: Cliente.java

**Contenido**: Constructor con parámetro para la instanciación de una entidad. Métodos get y set de los atributos.

**Ejemplo:**

**package** transferObjects;

**public** **class** Cliente

{

**private** String nombre;

**public** String getNombre () {

**return** nombre;

}

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

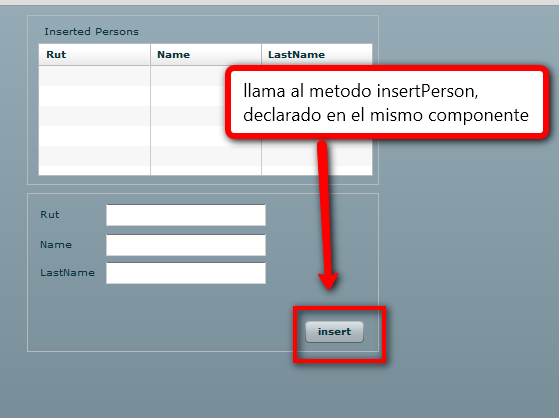
}

}

## Ejemplo

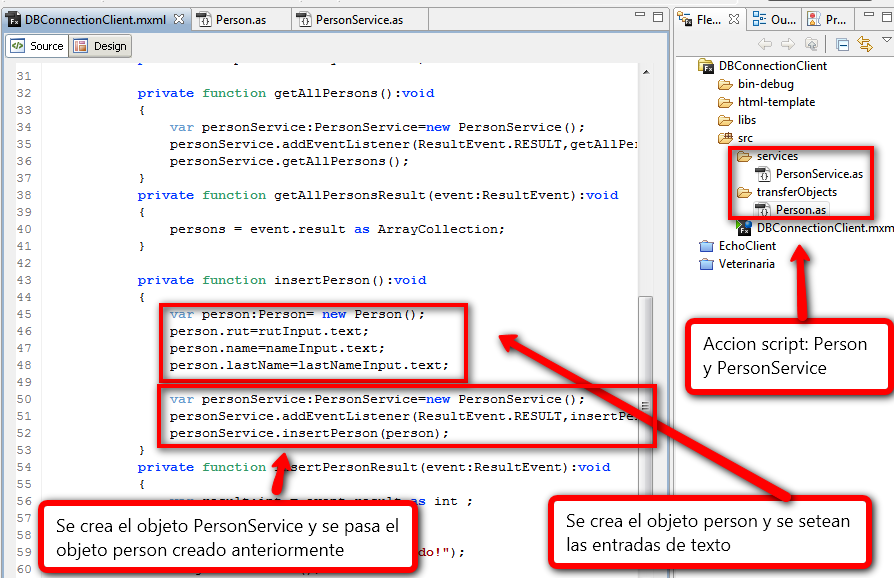
Considerando el ejemplo realizado en la ayudantía.

Clases que interactúan en Flex



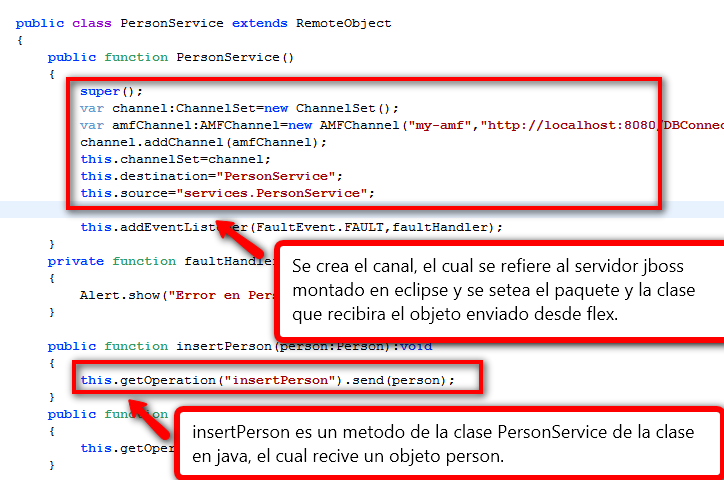
EL control insert llama a un método.

Observacion: En las propiedades de botton solo se hace el llamado al método inserPerson, en este caso no se pasa directamente los inputText de rut, name y lastName, debido a que pueden ser recogidos desde el mismo componente.



El botón insert, llama a la función insertPerson, en la cual se instancia una persona, y se asignan a los atributos, las entradas de los inputText.

Este objeto persona, es pasado al actionScript PersonService.

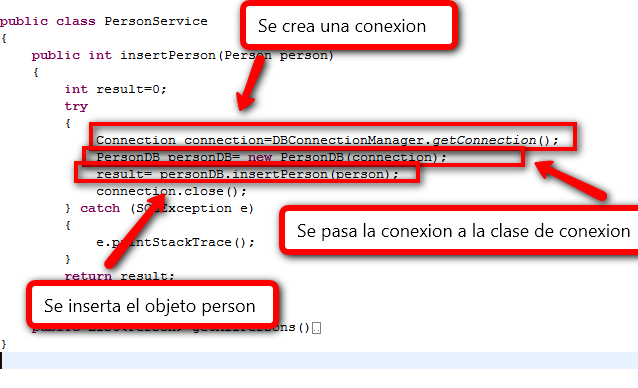


Este action script posee la configuración del servidor y la clase a la cual será enviado el objeto person.

En el método insertPerson, se pasa un objeto person.

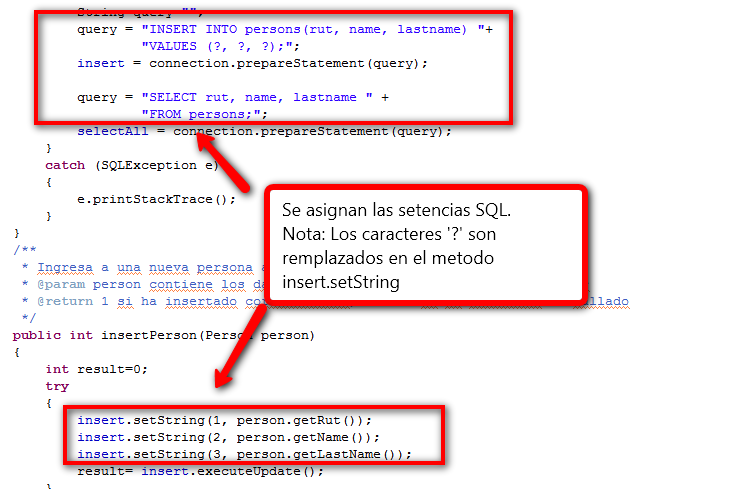
Clases que interactúan en Java

Clase que recibe el objeto que fue enviado desde el action script de flex.



Se declara una conexión, y un objeto que utilizara la conexión e insertara los datos.

Clase PersonDB, se recibe la conexión y el objeto person, y se crea la sentencia SQL



El objeto insert mediante el método setString, pasa en orden los atributos, para esto se debe indicar numéricamente la posición y el valor.

En el caso de insert.setString(1, person.getRut()) corresponde al primer símbolo ‘?’ del atributo “query”, en el cual se inserta el rut.

Estándar SQL

**Estándar de Consultas a la Base de Datos**

**Estándar Métodos**

Dentro del proyecto de Java contemplaremos los siguientes paquetes para la manipulación de datos:

**1. AdministracionBD:** En este paquete se almacenarán los distintos métodos de cada clase descritos en el archivo “Diagrama de clases”, como una clase. La implementación de estas clases nos permitirá implementar las distintas consultas SQL para el manejo de datos.

* **Nombre:** El nombre de cada clase corresponderá al nombre mismo del método existente (pero con mayúscula al inicio) con las iniciales “BD” al termino, es decir de la siguiente forma: <nombre\_método>BD.java.

**Ejemplo:** Dentro de la clase *MantenerUsuario* tenemos el método *addUsuario*, por lo tanto la nueva clase que crearemos dentro del paquete *Administración* será *AddUsuarioBD.java.*

* **Librerías:** Las librerías que importaremos para cada clase corresponderán a las clases utilizadas desde el paquete: TransferObjects.

**Ejemplo:** Para la clase *AddUsuarioBD.java* importaremos *TransferObjects.Usuario*.

* **Clase:** Lo primero que haremos dentro de nuestra clase, será crear la función que nos permita conectarnos para la obtención de datos, esta clase tendrá el mismo nombre de la clase.

**Ejemplo:**

public AddUsuarioBD(Connection connection)

{

this.connection = connection;

}

Luego, agregaremos la función encargada de realizar las acciones correspondientes al manejo de datos solicitado, esta llevara el nombre del método de la clase

Ejemplo: *addUsuario.*

* **Función:** Finalmente, dentro de la función correspondiente al método de la clase, declararemos la variable utilizada para la consulta llamada “query”, esta variable nos recibirá el resultado arrojado por la base de datos, para posteriormente ser utilizada en el manejo de los datos según sea la acción que se desee realizar.

**Ejemplo:**

query = "INSERT INTO Usuario (nombre,aPaterno,aMaterno,usuario,cargo,contrasena,servicio,pRegistrar,pEditar,pEliminar,pPurgar) "+

"VALUES (?, ?, ?, ?,?,? ,?,?,? ,?,?);";

insert = connection.prepareStatement(query);

**2. Administración:** Análogamente al paquete *AdministracionBD;* en este paquete se almacenarán los distintos métodos de cada clase descritos en el archivo “Diagrama de clases”, como una clase. La implementación de estas clases nos permitirá instanciarlas posteriormente al momento de realizar las acciones correspondientes al manejo de datos.

* **Nombre:** El nombre de cada clase corresponderá al nombre mismo del método existente (pero con mayúscula al inicio), es decir de la siguiente forma: <nombre\_método>.java.

**Ejemplo:** Dentro de la clase *MantenerUsuario* tenemos el método *addUsuario*, por lo tanto la nueva clase que crearemos dentro del paquete *Administración* será *AddUsuario.java.*

* **Librerías:** Las librerías que importaremos para cada clase corresponderán a las clases utilizadas desde los paquetes: TransferObjects y AdministracionBD.

**Ejemplo:** Para la clase *AddUsuario.java* importaremos *TransferObjects.Usuario* y *AdministracionBD.AddUsuarioBD*.

* **Clase:** Dentro de cada clase crearemos una función *public stactic* con su tipo correspondiente; el nombre será el mismo correspondiente al método de la clase.

**Ejemplo:**

public class AddUsuario {

public static String addUsuario(Usuario u) throws SQLException{

}

}

* **Función:** A su vez, dentro de la función insertaremos la conexión que realizaremos con la base de datos, correspondiente a la clase que estamos utilizando para poder acceder a los datos.

**Ejemplo:**

Connection connection=DBConnectionManager.getConnection();

AddUsuarioBD aubd = new AddUsuarioBD(connection);

return aubd.addUsuario(u);

Continuando con el manejo de datos al interior de una función, basándonos en el archivo “Diagrama de clases”, esta puede realizar las siguientes acciones: add, purg, set, del. Para donde cada uno de ellos se define de la siguiente forma.

* **Add:** Este tipo de método será representado por la variable “insert” al interior de la función. Una vez declarada por medio de *PreparedStatement,* hacemos uso de ella.

**Ejemplo:**

PreparedStatement insert;

Luego, al momento de insertar los datos, este será por medio de la variable declarada anteriormente mas el tipo que se desea agregar insert.<tipo>

**Ejemplo:**

insert.setString(1, u.nombre);

insert.setString(2, u.apellidoMaterno);

insert.setString(3, u.apellidoMaterno);

insert.setString(4, u.usuario);

insert.setString(5, u.cargo);

insert.setString(6, u.contrasena);

insert.setString(7, u.servicio);

insert.setBoolean(8, u.permisoRegistrar);

insert.setBoolean(9, u.permisoEditar);

insert.setBoolean(10, u.permisoEliminar);

insert.setBoolean(11, u.permisoPurgar);

* **Get:** Este tipo de métodos será representado por la variable “select” con el nombre correspondiente al dato al final, select<dato>.

**Ejemplo:**

PreparedStatement selectRut;

A su vez, para la obtención de estos datos crearemos una función que nos retorne una lista con la información, para luego ser utilizada. El nombre de esta función será *getAll<dato>.*

**Ejemplo:**

public List<Login> getAllPersons()

Para poder manipular los datos, debemos crear una instancia del objeto que deseamos, posteriormente lo agregamos a la lista que habíamos creado para utilizar.

**Ejemplo:**

public List<Login> getAllPersons()

{

List<Login> persons=new ArrayList<Login>();

Login person;

try

{

ResultSet result = selectAll.executeQuery();

while(result.next())

{

person= new Login();

person.setRut(result.getString(1));

person.setName(result.getString(2));

persons.add(person);

}

}

catch (SQLException e)

{

e.printStackTrace();

}

return persons;

}

* **Set:** Este tipo de métodos será representado por la variable “set” con el nombre correspondiente al dato al final, set<dato>.

**Ejemplo:**

PreparedStatement setRut;

Cabe mencionar que para “Set” debemos obtener los datos nuevamente para luego modificarlos, por lo tanto al igual que “Get” haremos el llamado a una función que nos retorne una lista con todos los datos solicitados.

* **Del:** Este tipo de métodos será representado por la variable “delet” con el nombre correspondiente al dato al final, delet<dato>.

**Ejemplo:**

PreparedStatmente deletRut;

* **Purg:** Este tipo de métodos será representado por la variable “purgar” con el nombre correspondiente al dato al final, purgar<dato>.

**Ejemplo variable:**

PreparedStatmente purgarRut;

**Ejemplo función:**

public int deleteMascota(int id){

int result=0;

try{

delete.setInt(1, id);

result = delete.executeUpdate();

}

catch (SQLException e){

e.printStackTrace();

}

return result;

}

**ESTANDAR SQL**

## 1. Select

La recuperación de los datos en el lenguaje SQL se realiza mediante la sentencia SELECT, seleccionar, la que consta de cuatro partes básicas:

* La cláusula SELECT seguida de la descripción de lo que se desea ver, los nombres de las columnas a seleccionar. Esta parte es obligatoria.
  + Las columnas a seleccionar se enumeran sin más en la cláusula SELECT. Si se desea seleccionar todas las columnas de una tabla se puede hacer enumerando a todas las columnas o colocando un asterisco, \*, en su lugar.
* La cláusula FROM seguida de la especificación de las tablas de las que se han de obtener los datos. Esta parte es obligatoria.
  + La cláusula FROM define las tablas de las que se van a seleccionar las columnas.
* La cláusula WHERE seguida por un criterio de selección, una condición. Esta parte es opcional.
  + Es aquí donde debemos proponer la condición que han de cumplir todas las filas para salir en el resultado de la consulta
* La cláusula ORDER BY seguida por el criterio de ordenación. Esta parte es opcional.
  + Se utiliza para especificar el criterio de ordenación de la respuesta a la consulta. Por defecto la ordenación es ascendente, aunque se puede especificar un orden descendente

Una primera aproximación a la sintaxis de la sentencia SELECT puede mostrarnos la siguiente expresión:

SELECT {\* | {columna,}+}

FROM {tabla,}+

[WHERE condición]

[ORDER BY {expresiónColumna [ASC | DESC],}+];

**Ejemplo**

query = "SELECT rut, name" +"FROM logins;";

selectAll = connection.prepareStatement(query);

## 2. Actualizar

Otra de las operaciones más comunes es la modificación de la información almacenada en las tablas. Para ello se utiliza el comando UPDATE cuya sintaxis se muestra a continuación.

UPDATE tabla SET {columna = expresión,}+ [WHERE condición];

Se especificará en la cláusula SET las columnas que se actualizarán y con qué valores. La cláusula WHERE indica las filas con las que se va a trabajar. Si se omite la actualización afectará a todas las filas de la tabla.

**Ejemplo**

query = "UPDATE Servicio SET Servicio.descripcion = '"+s.getDescripcion()+"', Servicio.Admistrador\_idAdmistrador='"+s.getAdministrador()+"';";

setServicio= connection.prepareStatement(query);

## 3. Borrar

Con insertar y modificar, la otra operación que completa el trio es la de borrado de filas. La sintaxis es la que sigue:

DELETE FROM tabla [WHERE condición];

Borrará todas las filas que cumplan la condición especificada en la cláusula WHERE. Si esta cláusula no se fija, se borrarán todas las filas de la tabla. Aquí cabe decir que aunque con DELETE borremos todas las filas de una tabla, no borramos la definición de la tabla del diccionario y podemos insertar datos posteriormente en la tabla. Esta es una diferencia con la sentencia DROP TABLE, que produce la eliminación tanto del contenido de la tabla como de la definición de la misma.

**Ejemplo**

query = "DELETE FROM Usuario” + “WHERE nombre=?”;

delete= connection.prepareStatement(query);

## 4. Insertar

El comando que permite insertar filas en las tablas es el siguiente.

INSERT INTO tabla [({columna,}\*)] VALUES ({expresión,}+);

Sólo especificaremos las columnas donde insertar y su orden cuando no insertemos datos en todas ellas o no lo hagamos en el mismo orden en que definimos la tabla. La asociación columna-valor es posicional. Los valores deben cumplir con los tipos de datos definidos. Los valores de tipo caracter y fecha deben ir encerrados entre comillas simples, ('').

**Ejemplo**

query = "INSERT INTO Usuario (nombre,aPaterno,aMaterno,usuario,cargo,contrasena,servicio,pRegistrar,pEditar,pEliminar,pPurgar) "+"VALUES (?, ?, ?, ?,?,? ,?,?,? ,?,?);";

insert = connection.prepareStatement(query);

# CREACIÓN DE TABLAS Y POBLAMIENTO

Se detallan las siguientes normas:

* Creación de tablas: Sintaxis SQL para la creación de tablas. Sera ingresada en la opción SQL de PgAdmin.
* Convención de nombres: Estándar de notación de atributos, y entidades.
* Poblamiento de la BD: Sintaxis SQL para la inserción de datos en la BD;
* Consideraciones de integridad referencial: Propiedades que deberán ser agregadas a la creación de tablas para permitir la integridad referencial.
* Ejemplo

## Creación de tablas

create table <nombre tabla> (

<nombre atributo> <tipo> <propiedad>,

<Otro nombre atributo> <tipo><propiedad>,

Constraint PK\_<nombre tabla> primary key (<nombre atributo>) < propiedad >

<Otro nombre atributo foráneo> <tipo> references <nombre tabla> (<nombre clave foránea >) <Propiedades de integridad referencial> );

* Nombre tabla: Nombre de la entidad, primera letra en mayúscula (Ver anexo)
* Nombre atributo: Minúscula, si es palabra compuesta separada por Primera letra de la segunda palabra en mayúscula.
* Tipo: Pueden ser:
  + bool
  + serial : Este tipo de datos permite que un atributo se incremente de forma automática en una unidad, Es utilizado para los identificadores.
  + char(n)
  + date
  + money
  + time
  + varchar(n)

Para ver otros tipos consultar <http://www.postgresql.org/docs/7.4/interactive/datatype.html> columna “aliases”

Para saber los valores

* Propiedad: (Ver anexo)
  + NOT NULL
  + NULL
  + Propiedad de integridad referencial: Esto es aplicado solo a las tablas que contienen clave foránea, y es aplicado para mantener la integridad de una tabla ante atributos apuntados desde otras tablas y ante actualizaciones de las claves foráneas.

La sentencia que se utilizara es la siguiente:

MATCH FULL ON DELETE RESTRICT ON UPDATE CASCADE

**Consideración integridad referencial:**

Deberán ser creadas primero las tablas que no contienen claves foráneas. Es decir se debe resolver antes la dependencia.

## Poblamiento BD

insert into <nombre tabla> (<nombre atributo>, <nombre atributo>)

values ('<valor>',’ <valor>’);

* Nombre tabla: Nombre de la entidad, primera letra en mayúscula (Ver anexo)
* Nombre atributo: Minúscula, si es palabra compuesta separada por Primera letra de la segunda palabra en mayúscula.
* Valor: Dato que será ingresado a la BD, este valor debe ser del mismo tipo del que fue declarado en la creación de la tabla.

**Consideración integridad referencial:**

Si el valor de la clave foránea no existe en la tabla, es decir no fue previamente insertado, el sistema de BD arrojara un problema de ejecución tal como el siguiente:

ERROR: insert or update on table "<nombre table>" violates foreign key constraint "<nombre atributo foraneo>"

## Ejemplo

Ejemplo de creación de tablas

create table ciudad (

nombre varchar(80) primary key,

ubicacion point

);

create table tiempo (

ciudadNombre varchar(80) references ciudad(nombre) MATCH FULL ON DELETE RESTRICT ON UPDATE CASCADE,

temperatura int

);

INSERT INTO ciudad (nombre, ubicacion)

VALUES ('Curico', '12,30');

Modelo Relacional

**post\_operatorio**(hora, fecha, costo, responsables, rut\_cliente, nombre\_mascota, medicamentos, alimentos, fechaNacimiento\_mascota, indicaciones, idPostOperatorio)

**pre\_operatorio**(hora, fecha, fechaNacimiento\_mascota, costo, responsables, rut\_cliente, nombre\_mascota, observaciones, síntomas, diagnostico, idPreOperatorio)

**cirugía**(hora, fecha, costo, fechaNacimiento\_mascota, responsables, rut\_cliente, nombre\_mascota, estado, diagnostico, tipo, idCirugia)

**consulta**(hora, fecha, costo, fechaNacimiento\_mascota, responsables, rut\_cliente, nombre\_mascota, anamnesis, idConsulta)

**control**(hora, fecha, costo, fechaNacimiento\_mascota, responsables, rut\_cliente, nombre\_mascota, descripcion, recomendaicon, próximaFecha, idControl)

**vacuna**(hora, fecha, costo, fechaNacimiento\_mascota, responsables, rut\_cliente, nombre\_mascota, precio, nombre, descripcion, fechaCaducidad, laboratorio)

**alojamiento**(hora, fecha, fechaNacimiento\_mascota, costo, responsables, rut\_cliente, nombre\_mascota, canil, fecha-salida, comentario, tiempoEstadia, solicitudes, idEstadia)

**baño**(hora, fecha, costo, fechaNacimiento\_mascota, responsables, rut\_cliente, nombre\_mascota, tipo, comentario, idBaño)

**corte**(hora, fecha, costo, fechaNacimiento\_mascota, responsables, rut\_cliente, nombre\_mascota, nombre, descripcion, raza, foto, idCorte)

**servicio**(nombre, nombresAdministradores, descripcion, rut\_cliente)

**hora\_bloqueada**(fecha, hora, nombre\_usuario, nombre\_servicio)

**usuario**(nombre, contraseña, cargo, privilegio)

**pertenece**(nombre\_usuario, nombre\_servicio)

**cliente**(rut, estado, teléfonos, nombre, correo)

**domicilio**(rut, numero, calle, comuna)

**mascota**(raza, estado, fotos, nombre, fechaNacimiento, sexo, edad, rut\_cliente)

**aviso**(hora, fecha, descripcion, correo\_cliente\_web, nombre\_mascota, fechaNacimiento\_mascota, rut\_cliente)

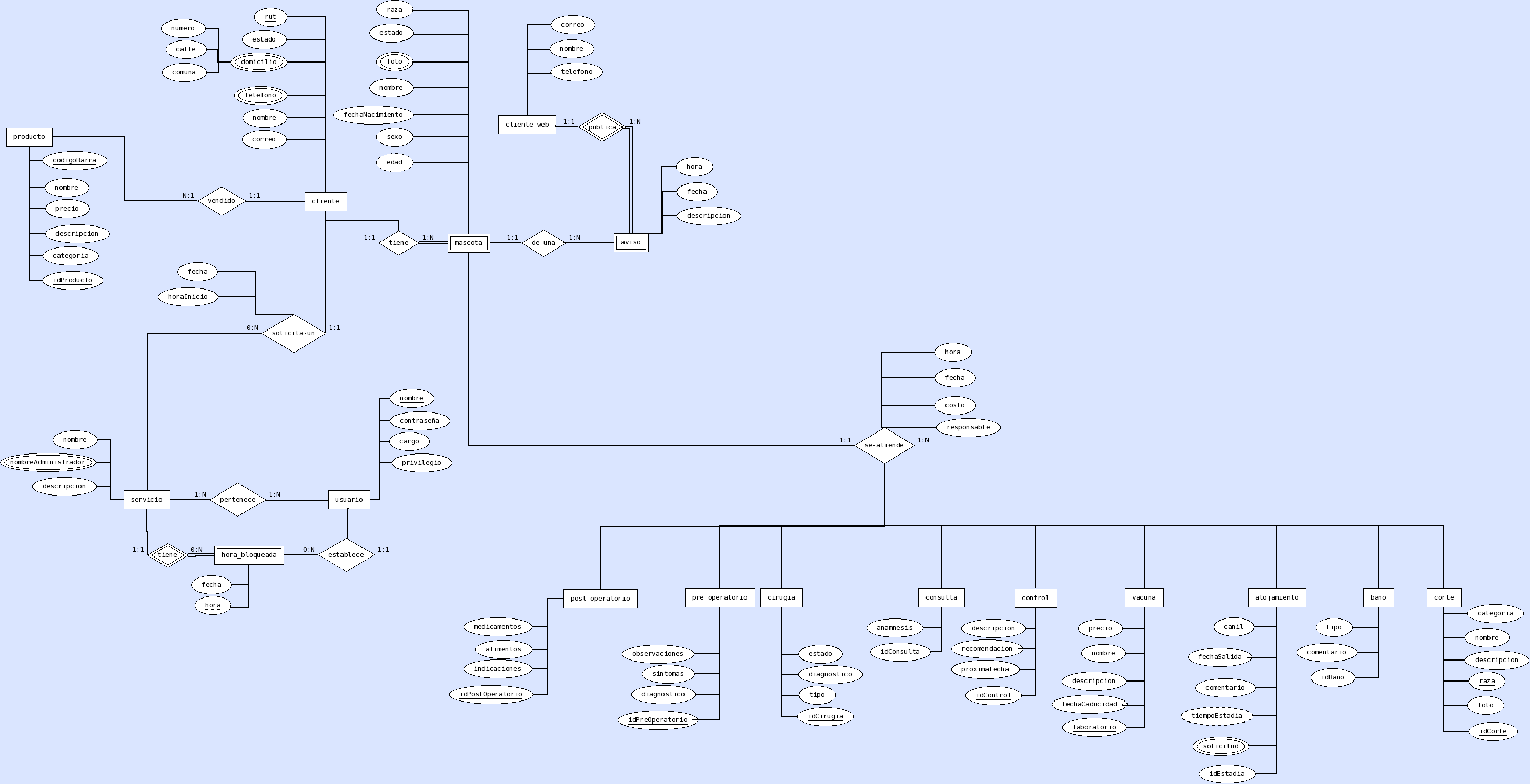
**cliente\_web**(correo, nombre, telefono)

**producto**(codigoBarra, nombre, precio, descripcion, categoría, idProducto)

**solicita-cita**(rut\_cliente, nombre\_servicio, fecha, horaInicio)

**producto-vendido**(rut\_cliente, codigoBarra, idProducto)

Modelo Base de datos



Estándar de documentación de código de actionScript

**Estructura general de un archivo fuente**

//=======================================================================

// FECHA CREACIÓN:

// AUTOR:

// …. *Comentarios generales*

//=======================================================================

package *xxxx*;

import *xxxxxx* ;

....

**Tags usados en ASDoc**

|  |  |
| --- | --- |
| **Tag** | **Descripción** |
| **@**private | Impide que una propiedad o método aparezca en la explicación html porque si no se especifica ningún comentario, ni este tag, aparecerá de todas formas. |
| **@**default | Explica el valor predeterminado de una propiedad. |
| @param | Definición de un parámetro de un método |
| @return | Describe el valor devuelto de un método, aunque la clase del valor devuelto aparecerá de todas formas. |
| @see | Referencia hacia otro elemento de la clase, otra clase (del mismo paquete o de otro distinto) o hacia un sitio externo. |
| @example | Enuncia un ejemplo de uso del elemento que comentemos. El código que vayamos a poner en el ejemplo va entre etiquetas **<listing version="3.0" ></listing>** |
| @langversion | Versión de actionScript |
| @author | Autor de la clase o archivo |

La forma de documentar el código es muy similar al utilizado en java para generar javadoc.

**Comentario de clases e interfaces**

/\*\*

\* La primera frase de un multi-parrafo de una clase.

\* <p>segundo parrafo.</p>

\*

\* @langversion ActionScript 3.0

\*

\* @author "Ray Koch"

\*/

class SomeThing() {

**Comentario de métodos**

/\*\*

\* Aquí van los comentarios del método

\*

\* @langversion ActionScript 3.0

\*

\* @param a Describe param a aquí.

\* @param b Describe param b aquí.

\*

\* @see otraFuncionAsociada

\*

\* @example myFunction("daddy-o", 3234);

\*/

public function myFunction(a:String, b:Number):void {}

**Comentario de atributos**

/\*\* String de propiedad privada \*/

private var privateProperty:String = "Keep Off: Private Property";

**Comentado MXML**

<!---

Este comentario soporta todos los tags de asDoc

Por ejemplo:

@see mx.container.VBox

-->

<mx:VBox xmlns="http://ns.adobe.com/mxml/2009" xmlns:mx="library:adobe/flex/halo" >

<!---

Comentario para el botón

-->

<mx:Button id="miBoton" label="Este botón ha sido comentado"/>

<!---

Este comentario no afecta a ningún componente por lo que es ignorado

-->

</mx:VBox>

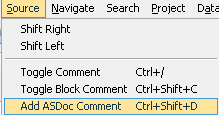
**Herramientas a utilizar para generar AsDoc**

Flex Builder 3 contiene una herramienta de ayuda para comentar el código, un ejemplo práctico es el siguiente:

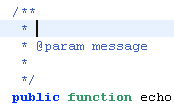
1. Posicionar el cursor de text imput al inicio de una clase



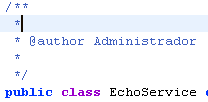
1. Seleccionar la pestaña “Source” y luego “Add ASDoc Comment”



1. Aparecerá lo siguiente:



Dependiendo si es función o clase varía el comentario que se genera, por ejemplo si es una clase se genera lo siguiente:



dentro de este tipo de comentarios se agregan los tags explicados anteriormente.

La precondición para generar el asDoc es tener bien comentados cada una de las clases y métodos, es decir que no existan errores de referencia, ni de sintaxis, por referencia se entiende a las importaciones de paquetes y llamadas a clases o métodos dentro de otra clase o método.

Para generar la documentación asdoc se utiliza el archivo **doc.bat** este se inserta en la carpeta del proyecto.

**Como crear el archivo doc.bat**

Se crea un nuevo docuemento dentro de un editor de textos sencillo, se recomienda usar notepad,

1. El archivo debe contener lo siguiente (solo se coloca el contenido dentro de la línea que se indique):

Linea |\_ Contenido\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. *path* ***G:\"Program Files"****\Adobe\"Flex Builder 3"\sdks\3.0.0\bin*
2. *asdoc -source-path . -doc-sources . -main-title "Nombre del proyecto" -window-title "Nombre asignado a la ventana" -output doc*
3. *pause*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

1. Lo que se debe modificar:
2. ***G:\"Program Files"***Se reemplaza con la dirección donde está la capeta "Adobe" o donde esté instalado "Flex Builder 3".
3. reemplazar "Nombre del proyecto" con el nombre del proyecto que se está trabajando.
4. reemplazar "Nombre asignado a la ventana" , lo ideal es reemplazarlo con el mismo nombre del proyecto.
5. En caso de ser necesario **excluir** algún archivo de asDoc agregar dentro de la línea 2 -exclude-classes nombreClase
6. Luego guardar como ***doc.bat*** dentro de la carpeta del proyecto.
7. Al ejecutarlo se crea automáticamente la carpeta *doc* dentro del proyecto.

Nota: Esta herramienta es más exigente que el generador javadoc por lo que el código no debe contener errores de referencia en los métodos.

ESTANDAR DE DOCUMENTACION DE CODIGO JAVA

**Estructura general de un archivo fuente**

//=======================================================================

// FECHA CREACIÓN:

// AUTOR:

// …. *Comentarios generales*

//=======================================================================

package *xxxx*;

import *xxxxxx* ;

....

**Tags usados en javadoc**

|  |  |
| --- | --- |
| **Tag** | **Descripción** |
| @author | Nombre del desarrollador. |
| @deprecated | Indica que el método o clase es antigua y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores. |
| @param | Definición de un parámetro de un método, es requerido para todos los parámetros del método. |
| @return | Informa de lo que devuelve el método, no se puede usar en constructores o métodos "void". |
| @see | Asocia con otro método o clase. |
| @throws | Excepción lanzada por el método |
| @version | Versión del método o clase. |
| @inv | Invariante de una clase |
| @since | Utilizada para indicar la versión del JDK |
| @pre | Precondición de un método o constructor. |
| @pos | Poscondición de un método o constructor. |
| @exception | Excepción lanzada por el método |

**Documentación del código**

El código debe comentarse utilizando la sintaxis apropiada para uso de *javadoc*, teniendo en cuenta que para la producción de la documentación deben incorporarse los tags particulares que no hagan parte del estándar.

**Comentario de clases e interfaces**

/\*\*

\* *Descripción de la clase*

\* @author *Nombre del desarrollador 1*

\* @author *Nombre del desarrollador 2*

\* @version *número versión y fecha*

\* @inv Invariante de clase

\*/

Ejemplo de documentación de una clase

/\*\*

\* Se utiliza para intentar convertir una cadena en un tipo

\* Numérico, pero la cadena no tiene el formato adecuado

\*

\* @author unascribed

\* @version 1.16, 02/02/00

\* @see java.lang.Integer#toString()

\* @since JDK1.0

\*/

public

class NumberFormatException extends IllegalArgumentException {

/\*\*

\* Construye una <code>NumberFormatException</code> con ningún mensaje de detalle.

\*/

public NumberFormatException () {

super();

}

/\*\*

\* Construye una <code>NumberFormatException</code> con la

\* especificación detallada del mensaje.

\*

\* @param s Se describe aquí la variable “s”

\*/

public NumberFormatException (String s) {

super (s);

}

}

**Comentario de atributos**

/\*\* Descripción del atributo \*/

Ejemplo de documentación de atributos

/\*\*

\* Número de alumnos

\*/

private int alumnos;

**Comentario de métodos constructores**

/\*\*

\* *Descripción de lo que hace el método*

\* @param p *Descripción del parámetro*.

\* @pre *precondición uno*

\* @pre *precondición dos*

\* @pos *poscondición uno*

\* @pos *poscondición dos*

\* @exception *TipoExcepción*, *Condiciones que causan la excepción*

\*/

Ejemplo de documentación de un constructor.

public class Medio {

protected String clave;

protected String titulo;

protected String genero;

protected int duracion;

protected Fecha fecha;

/\*\*

\* Constructor predeterminado

\*/

public Medio() {

}

/\*\*

\* Constructor que inicializa los atributos de la clase

\* @param clave Clave de la canción o película

\* @param titulo Título de la canción o película

\* @param genero Género de la canción o película

\* @param duracion Duración de la canción o película

\* @param fecha Fecha de liberación de la canción o película

\*/

public Medio(String clave, String titulo, String genero, int duracion,

Fecha fecha) {

this.clave = clave;

this.titulo = titulo;

this.genero = genero;

this.duracion = duracion;

this.fecha = fecha;

}

}

**Comentarios de métodos analizadores**

Un método analizador es el que retorna un valor, el nombre de la mayoría de estos métodos comienzan con “get”

/\*\*

\* *Descripción de lo que hace el método*

\* @param p *Descripción del parámetro*.

\* @pre *precondición uno*

\* @return *valor a retornar*

\* @exception *TipoExcepción*, *Condiciones que causan la excepción*

\*/

Ejemplo de documentación de un método analizador

/\*\*

\* Retorna las unidades de café del inventario

\*

\* @return int

\*/

public int getUnidCafe() {

return coffee;

}

**Comentario de métodos modificadores**

Un método analizador es el que retorna un valor, el nombre de la mayoría de estos métodos comienzan con “set”

/\*\*

\* *Descripción de lo que hace el método*

\* @param p *Descripción del parámetro*.

\* @pre precondición uno

\* @pos poscondición uno

\* @exception *TipoExcepción*, *Condiciones que causan la excepción*

\*/

Ejemplo de documentación de un método modificador

/\*\*

\* Inserta un título en la clase descripción.

\* Al ser el título obligatorio, si es nulo o vacío se lanzará

\* una excepción.

\*

\* @param titulo El nuevo título de la descripción.

\* @throws IllegalArgumentException Si titulo es null, está vacío o contiene solo espacios.

\*/

public void setTitulo (String titulo) throws IllegalArgumentException

{

if (titulo == null || titulo.trim().equals(""))

{

throw new Exception ("El título no puede ser nulo o vacío");

}

else

{

this.titulo = titulo;

}

}

**Otros comentarios internos**

// ................…

public class Inventario {

//código de Inventario aquí

}

**Herramientas a utilizar para generar JavaDoc**

Doclipse: plugin de eclipse que genera ayuda contextual para generar javadoc.

Ejemplo:

Se tiene la clase

public class Inventario {

//código de Inventario aquí

}

Antes de la clase se escribe /\*\* y se presiona Enter, se genera automáticamente el siguiente código

/\*\*

\*

\*

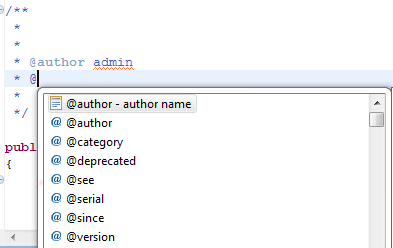
\* **@author** admin

\*

\*

\*/

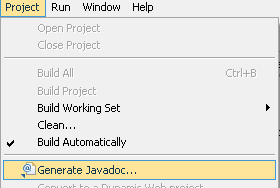
A continuación se escribe @ y luego Ctrl + Space y aparece lo siguiente



**Generando javadoc**

Una vez que se ha documentado el código se procede a generar el javadoc, para esto es necesario hacer lo siguiente:

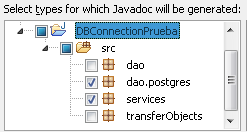
1. **Integrar JavaDoc a eclipse:** Dentro de Eclipse ir a la opción "Project" -> "Generate Javadoc".



1. E**specificar la dirección donde está instalado el generador javadoc (javadoc.exe) :**por ejemplo



1. **Seleccionar lo que se quiere incluir dentro del javadoc:**



1. **Carpeta de destino del javadoc generado**: se debe especificar una carpeta de destino donde se guardará el javadoc generado, por defecto queda dentro de la carpeta *doc* que está ubicada dentro de la carpeta del proyecto.



1. **Finalizando**: Para finalizar la creación de javadoc se presiona el botón “Finish”.



1. Ir al directorio donde se guardó el javadoc y ejecutar **index.html**

Estándar de programación

## CODIGO DE JAVA

## Indentación

Para la indentación de nuestra codificación se deben emplear cuatro espacios como unidad de indentación, ésta puede ser configurada en eclipse.

**Longitud de la línea**

Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.

**Salto de líneas**

Cuando una expresión no entre en una línea, se debe hacer el salto de línea de acuerdo a los siguientes principios:

* Después de una coma.
* Antes de un operador.
* Preferir saltos de línea de alto nivel (más a la derecha que el "padre") que de bajo nivel (más a la izquierda que el "padre").
* Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la anterior.

Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

Ejemplos de cómo hacer los saltos de línea en la llamada a un método:

unMetodo(expresionLarga1, expresionLarga2, expresionLarga3,

expresionLarga4, expresionLarga5);

var = unMetodo1(expresionLarga1,

unMetodo2(expresionLarga2,

expresionLarga3));

Ahora dos ejemplos de saltos de líneas en expresiones aritméticas. Se recomienda utilizar el primer ejemplo, ya que el salto de línea ocurre fuera de la expresión que encierra los paréntesis.

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo5) + 4 \* nombreLargo6; // PREFERIDA

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo) + 4 \* nombreLargo6; // EVITAR

Al momento de realizar un salto de línea en una sentencia, por ejemplo *if*, se recomienda indentar con 8 espacios en vez de 4, de manera que se vea más clara la estructura, por ejemplo:

//NO USAR ESTA INDENTACION

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) { //MALOS SALTOS

hacerAlgo(); //HACEN ESTA LINEA FACIL DE OLVIDAR

}

//USAR LA SIGUIENTE INDENTACION

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

//O USAR ESTA

if ((condicion1 && condicion2) || (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

**Declaraciones**

**Cantidad por línea**

Se recomienda una declaración por línea, ya que facilita los comentarios. Se acepta la declaración agrupada de variables si es que aquellas están relacionadas en algún procedimiento o estructura. Ejemplos:

int nivel; // nivel de indentación

int tam; // tamaño de la tabla

int ancho, alto; //dimensiones de la matriz

Nota: Los ejemplos anteriores usan un espacio entre el tipo y el comentario. Una alternativa aceptable es usar tabuladores, con lo que se obtiene más orden, por ejemplo:

int level; // nivel de indentación

int size; // tamaño de la tabla

Object currentEntry; // entrada de la tabla seleccionada actualmente

**Inicialización**

Intentar inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir. Con la inicialización evitamos posibles fallas en el uso de éstas

**Ubicación**

Orientar las declaraciones solo al principio de los bloques (un bloque es cualquier código encerrado por llaves "{" y "}".) No esperar al primer uso para declararlas; puede confundir y limitar la portabilidad del código dentro de su ámbito de visibilidad.

void myMethod() {

int int1 = 0; // comienzo del bloque del método

if (condition) {

int int2 = 0; // comienzo del bloque del "if"

...

}

}

La excepción de la regla son los indices de bucles *for*, que en Java se pueden declarar dentro de la sentencia:

for (int i = 0; i < N; i++) { ... }

Evitar las declaraciones locales que ocultan otras de niveles superiores, por ejemplo, no declarar la misma variable en un bloque interno:

int cuenta;

...

miMetodo() {

if (condicion) {

int cuenta = 0; // EVITAR!

...

}

...

}

**Declaraciones de class e interfaces**

Al codificar clases e interfaces de Java, se siguen las siguientes reglas de formato:

* Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
* La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración y no en la línea siguiente.
* La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

class Ejemplo extends Object {

int ivar1;

int ivar2;

Ejemplo(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int metodoVacio() {}

...

}

**Sentencias**

**Sentencias simples**

Cada línea debe contener como mucho una sentencia. Ejemplo:

argv++; // Correcto

argc--; // Correcto

argv++; argc--; // EVITAR!

**Sentencias compuestas**

Las sentencias compuestas son aquellas que contienen listas de procedimientos encerrados entre llaves " { sentencias }". Ellas siguen las siguientes reglas:

* Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
* La llave de apertura se debe poner al final de la línea que comienza la sentencia la llave de cierre debe empezar una nueva línea después de la última instrucción y ser indentada al mismo nivel que el pincipio.
* Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como en las sentencias *if-else* o *for*. Esto hace más sencillo añadir instrucciones sin producir errores accidentalmente por olvidar las llaves.

**Sentencias de retorno**

Una sentencia *return* con un valor no debe usar paréntesis. Ejemplos:

*return 0;*

*return arreglo.size();*

**Sentencias if, if-else, if else-if else**

La clase de sentencias if-else debe tener la siguiente forma:

if (condicion) {

sentencias;

}

if (condicion) {

sentencias;

}

else {

sentencias;

}

if (condicion) {

sentencia;

*}*

else {

if (condicion) {

sentencia;

}

else{

sentencia;

}

}

Nota: Las sentencias *if* como *else* usan siempre llaves {}. Evitar la siguiente forma, propensa a errores:

if (condicion) //EVITAR OMITIR LAS LLAVES COMO SE DIJO ANTERIORMENTE

sentencia;

**Sentencias for**

Una sentencia for debe tener la siguiente forma:

for (inicializacion; condicion; actualizacion) {

sentencias;

}

Una sentencia for vacía (una en la que todo el trabajo se hace en las clausulas de

inicialización, condicion, y actualizacion) debe tener la siguiente forma:

for (inicializacion; condicion; actualizacion);

Al usar el operador punto y coma en la clausula de inicialización o actualización de una sentencia *for*, evitar la complejidad de usar más de tres variables, éstas se deben manejar antes o después del ciclo según sea necesario.

**Sentencias while**

Una sentencia *while* debe tener la siguiente forma:

while (condicion) {

sentencias;

}

Una sentencia *while* vacía debe tener la siguiente forma:

while (condicion);

**Sentencias do-while**

Una sentencia *do-while* debe tener la siguiente forma:

do {

sentencias;

} while (condicion);

**Sentencias switch**

Una sentencia *switch* debe tener la siguiente forma:

switch (condicion) {

case ABC:

sentencias;

/\* este caso se propaga \*/

case DEF:

sentencias;

break;

case XYZ:

sentencias;

break;

default:

sentencias;

break;

}

Cada vez que un caso se propaga (no incluye la sentencia break), añadir un comentario donde la sentencia break se encontraría normalmente. Esto se muestra en el ejemplo anterior con el comentario */\* este caso se propaga \*/.*

Cada sentencia switch debe incluir un caso por defecto. El break en el caso por defecto es redundante, pero prevee que se propague por error si luego se añade otro caso. Además evitar dejar casos sin su correspondiente break.

**Sentencias try-catch**

Una sentencia try-catch debe tener la siguiente forma:

try {

sentencias;

} catch (ExceptionClass e) {

sentencias;

}

Una sentencia try-catch puede ir seguida de un finally, cuya ejecución se ejecutará independientemente de que el bloque try se haya completado con éxito o no.

try {

sentencias;

} catch (ExceptionClass e) {

sentencias;

} finally {

sentencias;

}

## Convenciones de nombres

**Clases**

Los nombres de las clases deben ser **sustantivos**, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo).

Ejemplos:

class **C**liente;

class **I**magen**A**nimal;

## Métodos

Los métodos deben ser **verbos**, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Ejemplos:

**e**jecutar();

**e**jecutar**R**apido();

**e**jecutar**M**uy**R**apido();

1. Nombres de métodos para Administración:
2. Registrar Objeto: **add**Objeto.
3. Editar Objeto: **edit**Objeto.
4. Eliminar Objeto: **del**Objeto.
5. Retornar Objeto: **get**Objeto.
6. Guardar Objeto: **set**Objeto.

Donde “Objeto” es el nombre de la clase como: Usuario, Cliente, Mascota, etc.

Para el caso de los objetos como Usuario, Mascota, Cliente, etc. sus métodos para interacción de atributos son:

1. Retornar Atributo: **get**Atributo
2. Guardar Atributo: **set**Atributo

Donde “Atributo” es el nombre completo del atributo pero con mayúscula por ejemplo, para retornar atributo “nombre” de la Clase Cliente el método se llamaría getNombre.

1. Parámetros de Métodos

El parámetro de entrada del método es el nombre completo en minúscula del atributo al que se refiere, Por ejemplo:

getUsuario(Usuario **usuario**)

setNombre(String **nombre**)

## Variables

Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas (como el caso de los métodos). Los nombres de las variables deben ser cortos pero con significado. Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales. Nombres comunes para variables temporales son i, j, k, m, y n para enteros (muy utilizados como contadores en ciclos for y while); c, d, y e para caracteres.

## Prefijos en Variables Globales

En el caso de que se utilicen variables que son únicamente de un objeto, y que son utilizadas en alguna clase de interfaz u otra en que intervienen otras clases u objetos, se recomienda utilizar un prefijo, que debe ser el nombre de la clase u objeto, seguido por un ‘\_’ y luego el nombre de la variable.

Ejemplos:

String mascota\_nombre;

String cliente\_nombre;

String mascota\_edad;

String cliente\_edad;

## Constantes

Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("\_").

Ejemplos:

static final int ANCHURA\_MINIMA = 4;

static final int ANCHURA\_MAXIMA = 99;

static final int NUMERO\_DE\_RAZAS = 100;

## Hábitos

## Proporcionando acceso a variables de instancia y de clase

Para que la parte lógica sea netamente orientada a objetos y el sistema sea seguro solo se permitirán atributos privados en una clase, por ende se restringe absolutamente la utilización de atributos públicos. Para poder acceder a estos se utilizaran métodos. Ya que a menudo las variables de instancia no necesitan ser asignadas/consultadas explícitamente, a menudo esto sucede como efecto lateral de llamadas a métodos.

## Referencias a variables y métodos de clase

Evitar usar un objeto para acceder a una variable o método de clase (static). Usar el nombre de la clase en su lugar.

Ejemplos:

CORRECTO

metodoDeClase();

UnaClase.metodoDeClase();

EVITAR

unObjeto.metodoDeClase();

## Asignaciones de variables

Evitar asignar el mismo valor a varias variables en la misma sentencia. Es difícil de leer.

Ejemplo:

EVITAR

fooBar.fChar = barFoo.lchar = 'c';

No usar asignaciones caladas como un intento de mejorar el rendimiento en tiempo de ejecución. Ese es el trabajo del compilador.

Ejemplos:

EVITAR

d = (a = b + c) + r;

CORRECTO

a = b + c;

d = a + r;

## Paréntesis

En general es una buena idea usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso si parece claro el orden de precedencia de los operadores, podría no ser así para otros, no se debe asumir que otros programadores conozcan el orden de precedencia.

Ejemplos:

EVITAR

if (a == b && c == d)

CORRECTO

if ((a == b) && (c == d))

## Variables temporales

Evitar variables temporales que no son posteriormente utilizadas o que son utilizadas una sola vez, esto para simplificar código, obviamente esto es aplicable si no complica su interpretación.

Ejemplos:

EVITAR

double basePrice = anOrder.basePrice();

return(basePrice > 100);

CORRECTO

return(anOrder.basePrice() > 1000);

En el ejemplo se puede ver que basePice es únicamente utilizada para retornar un valor en un función, pero esto se puede realizar de manera más directa y retornando anOrder.basePrice();

En caso de que una expresión de condición sea muy larga y compleja es mejor utilizar la forma anteriormente descrita como errónea.

Ejemplos:

EVITAR

if( ( platform.toUpperCase().indexOf(“MAC”) > -1 ) &&

( browser.toUpperCase().indexOf(“IE”) > -1) &&

wasInitialized() && resize > 0 ){

// do something

}

CORRECTO

final boolean isMacOs = plataform.toUpperCase().indexOf(“MAC”) > -1;

final boolean isIEBrowser = plataform.toUpperCase().indexOf(“IE”) > -1;

final boolean wasResized = resize > 0;

if (isMacOs && isIEBowser && wasInitialized() && wasResized){

// do something

}

## Sobre la administración de paquetes

En nuestro proyecto BonatiServer dentro del conjunto de códigos fuentes se encuentan distintos paquetes que organizan la estructura de nuestro sistema, los que podemos clasificar dentro de 3 grupos:

* **Bd**: paquete que contiene la clase DBConnectionManager.java, la que se encarga de conectar la capa 2 con la 3.
* **Administracion**: Contiene clases que gestionan los datos de la veterinaria. De manera que se trabaje más cómodamente con el SVN, este paquete tiene cada una de las clases por separadas, por ejemplo:
  + **addUsuario.java**
  + **getUsuarios.java**
  + **…**
* **AdministracionDB:** Contiene la equivalencia de las clases anteriores, pero éstas se conectan con la base de datos, deben seguir la siguiente forma:
  + **addUsuarioBD.java**
* **TransferObjects:** Clases de nuestro sistema que modelan las entidades, estas son equivalentes con las que existen en Flex, ejemplos:
  + **Usuario.java**
  + **Mascota.java**
  + **…**

Estándar código de action script

## Indentación

Para la indentación de nuestra codificación se deben emplear cuatro espacios como unidad de indentación, ésta puede ser configurada en Flex.

**Longitud de la línea**

Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.

**Salto de líneas**

Cuando una expresión no entre en una línea, se debe hacer el salto de línea de acuerdo a los siguientes principios:

* Después de una coma.
* Antes de un operador.
* Preferir saltos de línea de alto nivel (más a la derecha que el "padre") que de bajo nivel (más a la izquierda que el "padre").
* Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la anterior.

Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

Ejemplos de cómo hacer los saltos de línea en la llamada a un método:

unaFuncion(expresionLarga1, expresionLarga2, expresionLarga3,

expresionLarga4, expresionLarga5);

var variable:int = unMetodo1(expresionLarga1,

unMetodo2(expresionLarga2,

expresionLarga3));

Ahora dos ejemplos de saltos de líneas en expresiones aritméticas. Se recomienda utilizar el primer ejemplo, ya que el salto de línea ocurre fuera de la expresión que encierra los paréntesis.

CORRECTO:

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo5) + 4 \* nombreLargo6;

EVITAR:

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo) + 4 \* nombreLargo6;

Al momento de realizar un salto de línea en una sentencia, por ejemplo *if*, se recomienda indentar con 8 espacios en vez de 4, de manera que se vea más clara la estructura, por ejemplo:

EVITAR:

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) { //MALOS SALTOS

hacerAlgo(); //HACEN ESTA LINEA FACIL DE OLVIDAR

}

CORRECTO:

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

CORRECTO:

if ((condicion1 && condicion2) || (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

**Declaraciones**

**Cantidad por línea**

Se recomienda una declaración por línea, ya que facilita los comentarios. Se acepta la declaración agrupada de variables si es que aquellas están relacionadas en algún procedimiento o estructura. Ejemplos:

var nivel:int; // nivel de indentación

var tam:int; // tamaño de la tabla

Nota: Los ejemplos anteriores usan un espacio entre el tipo y el comentario. Una alternativa aceptable es usar tabuladores, con lo que se obtiene más orden, por ejemplo:

var nivel:int; // nivel de indentación

var size:int; // tamaño de la tabla

**Inicialización**

Intentar inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir. Con la inicialización evitamos posibles fallas en el uso de éstas.

**Sentencias**

**Sentencias simples**

Cada línea debe contener a lo más una sentencia, de igual forma que en el estándar de java.

**Sentencias compuestas**

Se deben cumplir las siguientes reglas:

* Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
* La llave de apertura se debe poner al final de la línea que comienza la sentencia la llave de cierre debe empezar una nueva línea después de la última instrucción y ser indentada al mismo nivel que el pincipio.
* Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como en las sentencias *if-else* o *for*. Esto hace más sencillo añadir instrucciones sin producir errores accidentalmente por olvidar las llaves.

**Sentencias de retorno**

Una sentencia *return* con un valor no debe usar paréntesis.

Ejemplos:

return 0;

return arreglo.size();

**Sentencias if, if-else, if else-if else**

La clase de sentencias if-else debe tener la siguiente forma:

if (condicion) {

sentencias;

}

if (condicion) {

sentencias;

}

else {

sentencias;

}

if (condicion) {

sentencia;

*}*

else {

if (condicion) {

sentencia;

}

else{

sentencia;

}

}

Se debe evitar no usar llaves

**Sentencias while**

Una sentencia *while* debe tener la siguiente forma:

while (condicion) {

sentencias;

}

Una sentencia *while* vacía debe tener la siguiente forma:

while (condicion);

**Sentencias do-while**

Una sentencia *do-while* debe tener la siguiente forma:

do {

sentencias;

} while (condicion);

## Convenciones de nombres

**Clases**

Al igual que en el caso de Java los nombres de las clases deben ser **sustantivos** y cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas, cabe señalar que los nombres de las clases en esta capa deben ser idénticos a la segunda capa (Java), mas adelante se explicara.

Ejemplos:

**C**liente.as;

**I**magen**A**nimal.as;

## Métodos

Los métodos en actionscript también deben ser **verbos** y cuando sean compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Ejemplos:

**public** **function** ejecutar():**void**

**public** **function** ejecutarRapido():**void**

**public** **function** ejecutarMuyRapido():**void**

Los nombres de las clases en actionscript deben ser los mismos con los que se comunican con java.

Por ejemplo:

Mascota.java -> Mascota.as

MascotaService.java -> MascotaService.as

Los Prefijos de los métodos de la clase ObjetoService son:

1. Registrar Objeto: **add**Objeto.
2. Editar Objeto: **edit**Objeto.
3. Eliminar Objeto: **del**Objeto.
4. Retornar Objeto: **get**Objeto.
5. Guardar Objeto: **set**Objeto.

En esta capa los objetos tienen atributos públicos, por ejemplo:

**public** **var** rutCliente:String;

**Parámetros de Métodos**

El parámetro de entrada de los métodos para las clases ObjetoService, es el nombre completo en minúscula del atributo al que se refiere, Por ejemplo:

**public** **function** addMascota(mascota:Mascota):**void**

**public** **function** delCliente(cliente:Cliente):**void**

## Variables

Este punto se comporta de la misma forma que el estándar en java, donde, excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas (como el caso de los métodos). Los nombres de las variables deben ser cortos pero con significado.

## Prefijos en Variables Globales o que interactúan en clases de interfaz que contengan a más de una clase

En el caso de que se utilicen variables que son únicamente de un objeto, y que son utilizadas en alguna clase de interfaz u otra en que intervienen otras clases u objetos, se utiliza un prefijo, que debe ser el nombre de la clase u objeto, seguido por un ‘\_’ y luego el nombre de la variable.

Ejemplos:

**public** **var** mascota\_nombre:String;

**public** **var** cliente\_nombre:String;

**public** **var** mascota\_edad:String;

**public** **var** cliente\_edad:String;

**ESTANDAR CODIGO DE FLEX**

Estructura de código en flex es de la siguiente forma:

1. Script
   1. Imports
   2. Variables Globales
   3. Funciones
2. Tags de Flex

Ejemplo:

<mx:Panel …">

<mx:Script>

<![CDATA[

**import** mx.controls.Alert;

[**Bindable**]

**public** **var** cliente:Person;

**private** **function** buscarCliente(event:Event):**void**{

getPerson(rutClienteInput.text);

}

]]>

</mx:Script>

<mx:Label x="28" y="28" text="Rut Cliente:" width="84"/>

</mx:Panel>

**Identificadores**

Los componentes son individualizados por sus identificadores y al ser estos importantes para referenciar o ser referenciados por sus funciones, se insertaran al inicio del tag, de la siguiente forma:

<mx:TextInput id="rutClienteInput" x="120" y="26" width="80"/>

**Convenciones de Nombres**

Los identificadores son ocupados principalmente en las entradas y salidas de textos que son utilizados para asignar los atributos del objeto o clase, por ende el identificador en las entradas el nombre del atributo seguido con Input, ejemplo:

Id=”rutInput”

Id=”nombreInput”

En los datagrid no se utiliza identificador para cada columna, sin embargo para poder desplegar un objeto en una fila de la tabla y hacer referencia a que cada atributo aparezca en una columna determinada, se debe cumplir lo siguiente:

La propiedad dataField de la columna de la tabla debe tener el mismo nombre del atributo del objeto que se debe desplegar por ejemplo:

Clase:

**public** **class** Cliente{

**public** **var** nombre:String;

**public** **var** edad:int;

**.**

**.**

}

Datagrid:

<mx:DataGrid …>

<mx:columns>

<mx:DataGridColumn dataField="nombre" …/>

<mx:DataGridColumn dataField="edad" …/>

.

.

</mx:columns>

</mx:DataGrid>

Estándar del usuario, basado en IEEEStd 1063-2001

**Acción**, 2

**Advertencia**, 3

**Advertencias y Precauciones**, 6

Ámbito de aplicación, 2

**Colocación de la información crítica**, 5

**Componentes iníciales**, 4

**conjunto de documentos**, 2

**Contenido de la información de la documentación del software del usuario**, 5

**Definiciones**, 2

**Descripción general**, 2

**documentación de usuario de software**, 3

**Estilo**, 3

**Estructura de la documentación del software del usuario**, 3

**Estructura general de la documentación**, 3

**Formato de la documentación del software del usuario**, 6

**Formatos de las advertencias, precauciones y notas**, 6

**Formatos para los dispositivos de documentación para acceder a la información**, 8

**ilustración**, 2

**información crítica**, 2

**La información sobre las fuentes de información relacionadas**, 6

**La información sobre los comandos de software:**, 5

**La información sobre mensajes de error y resolución de problemas**, 5

**Modo de instrucción**, 2

**Nota**, 3

**Notas**, 7

**paso**, 3

**Precauciones**, 2

**procedimiento**, 3

**Propósito**, 2

**Tutorial**, 3

**Usuario**, 3

1. **Descripción general**

El objetivo de este documento es establecer un consenso de la forma en que se entregaran las ayudas en el software de administración veterinaria para la clínica Claudio Bonati para que estas sean de real ayuda para los usuarios del sistema y que no se transformen en confusión e ineficiencia.

## Ámbito de aplicación

Este estándar de documentación para el usuario se aplica al producto (Software). El modelo utilizado “IEEE Std 1063-2001SW user documentation” se centra principalmente en las ayudas dinámicas al interior del software, ya sean mensajes, sugerencias o errores, este modelo establece los requisitos mínimos para la estructura, el contenido de la información y el formato para esta documentación, pero también establece como se deben estructurar los manuales de usuario, en donde se le explica a este cuales son las funcionalidades del software que está adquiriendo y como puede sacarle mayor provecho en pos de los objetivos estratégicos de la organización

* 1. **Propósito**

El propósito de este estándar es muy simple pero concreto, es dar una estructura clara a los documentos, la forma en que se representa la información tanto en el software mismo así también como en posibles documentos imprimibles y electrónica. Está dirigido tanto hacia la etapa de despliegue es decir a los usuarios finales así también como a los desarrolladores para que en la etapa de construcción permitiendo un software más integrado desde las etapas iníciales.

1. **Definiciones**

Es importante definir una terminología a utilizar de modo de facilitar el entendimiento en el grupo de desarrollo. La norma utilizada si bien no obliga una terminología especifica si cuenta con una, esta es la siguiente: IEEE 100,*The Authoritative Dictionary of IEEE Standards Terms Seventh Edition [B9]*

* 1. **Acción:** Pasos que el usuario debe realizar para completar un procedimiento
  2. **Precauciones:** Estas funcionan como un “asesor” en el software y se deben efectuar cuando el usuario ejecuta una acción que puede tener consecuencias no deseadas o que no se encuentran definidas, Ej.: Perdidas de datos o problemas en el equipo
  3. **Información crítica:** Se refiere a los mensajes sobre el uso seguro del software, y la privacidad de la información creada. Ej.: Mensajes de no tener premisos de escritura.
  4. **Conjunto de documentos:** Se trata de una recopilación de la documentación que se pueden generar de modo de facilitar la distribución entre los usuarios
  5. **Ilustración:** Es el término utilizado para definir cualquier elemento grafico que sea utilizado en la documentación. Ej.: Capturas de pantalla, iconos, diagramas de flujo.
  6. **Modo de instrucción:** Es la estructura que van a tener las instrucciones sobre el uso de las tareas el interior del software.
  7. **Nota:** Se trata de consejos prácticos con información que pueda ayudar al usuario Ej.: “Puede recargar la pagina presionando F5”
  8. **Procedimiento:** Es una serie ordenada de pasos que el usuario debe seguir para realizar una tarea.
  9. **Documentación de usuario de software:** Material que proporciona información al usuario sobre el uso del software, esta puede ser tanto impresa como digital.
  10. **Paso:** Es una parte de los procedimientos, un paso puede contener una o más acciones.
  11. **Estilo:** Son las convenciones que se hacen sobre la redacción, gramática, terminología y diseño(entre otros) de un documento de usuario de software.
  12. **Tutorial:** Procedimiento en el cual el usuario interactúa con el sistema guiado por una documentación.
  13. **Usuario:** Quien emplea el software para realizar una tarea.
  14. **Advertencia:** Tiene un funcionamiento como “asesor”, pero a diferencia de una sugerencia este advierte que una acción puede conducir a graves consecuencias.

**Siguientes puntos en construcción**

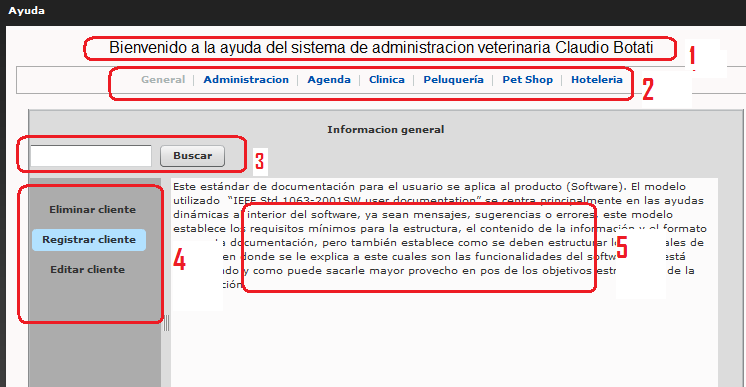
1. **Estructura de la documentación del software del usuario**

El principal documento generado será un documento electrónico que se mostrara al momento de clickear la opción de ayuda en la aplicación.



* 1. **Estructura general de la documentación:**

Este documento electrónico tiene la siguiente estructura:



Tamaño panel principal: 762x 516

Tamaño panel interno izquierdo (4) 130x 289

Tamaño panel de información (5) 547x 290

**Descripción por área:**

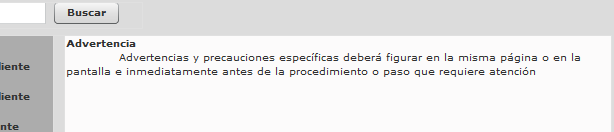
1. Corresponde al título bienvenida de la aplicación de ayuda: Ej. Bienvenido al sistema de administración veterinaria Claudio Bonati
2. Menú de los subsistemas de la aplicación
3. Búsqueda, permite buscar palabras dentro de la ayuda
4. Son las acciones que se pueden encontrar
5. Es la información de las acciones del área izquierda
   1. **Componentes iníciales:**

**Estructura de la información:**

Es importante especificar cómo va ir distribuida la información en el panel de texto (5),

Para esto se define lo siguiente:

El texto estará estandarizado como texto HTML, es decir se pueden ocupar todas las estructuras de este lenguaje.



<mx:Text width="100%" color="0x323232">

<mx:htmlText>

<![CDATA[<b>Registro Usuario</b>]]>

</mx:htmlText>

</mx:Text>

Títulos normales con negrita: <b>Registro Usuario</b>

Parrafo: <p>Parrafo</p>

Títulos errores graves con color rojo: <font color="#EE1122">Titulo con rojo</font>

De este modo el texto siguiente:

**Registro de usuario**

Es la forma de registrar un usuario

<![CDATA[<h1><b>Registro de usuario</h1></b> <div align="left">

<p>Es la forma de registrar un usuario </p>

</div> ]]>

* 1. **Colocación de la información crítica:**

La información critica va con un titulo en rojo para que se resalte en el texto

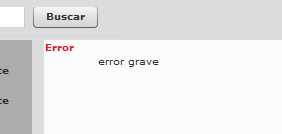
Error grave

Este es un error

<![CDATA[<h1><b><font color="#EE1122">Error grave </font></h1></b> <div align="left">

<p> Este es un error</p>

</div> ]]>

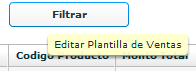


1. **Contenido de la información de la documentación del software del usuario:**
   1. **La información sobre los comandos de software:**

Es importante que el usuario se adapte lo más pronto posible al uso de la aplicación, para ello es importante que se le digan cuales son las formas más rápidas de realzar las acciones.

Esto puede realizarse de diversas formas pero principalmente mediante toolTips (1) o un área reservada para estos mensajes (2).

(1):



**Los tool tips deben ser muy representativos de lo que se espera que el usuario haga y deben estar en:**

* Botones de menú
* Botones submenú
* Botones de Ayuda y Búsqueda
* Botones de área derecha
* Text inputs

(2):



* 1. **La información sobre mensajes de error y resolución de problemas:**

El sistema tendrá un numero de errores que son conocidos y que en su momento no pudieron ser solucionados ya sea por el costo de este, o la falta de tiempo, por ello estos errores en algún momento pueden suceder, no es importante decirle al usuario en qué momento sucede el error sino que cual es la forma de que no le afecte en demasía. En este caso se debe mostrar el mensaje de error correspondiente, “Error grave”, y los caminos disponibles entre los cuales puede existir una vía de comunicación con los desarrolladores para reportar el error.



Este mensaje debe contener:

* Nombre del mensaje: Ej. Error grave
* Mensaje del error: Ej. El sistema ha sufrido un grave error
* Comunicar el error (si es necesario): Ej. Por favor reporte este error
* Pasos a seguir para minimizar las consecuencias: Ej. Debe reinicias la aplicación
  1. **La información sobre las fuentes de información relacionadas**

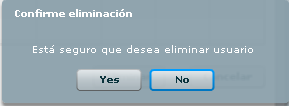
1. **Formato de la documentación del software del usuario**
   1. **Formatos de las advertencias, precauciones y notas:**
      1. **Advertencias y Precauciones:**

Todas las advertencias de este tipo deben ser claramente distinguibles, para esto se deben hacer popUp de la siguiente forma, se trata principalmente de los errores en que puede incurrir el usuario y no de errores del sistema:

*Descripción de los riesgos:* Cliente ya existe en el sistema, si continúa podría perder datos importantes

*Opciones disponibles:* Sobrescribir los datos, (cancelar, continuar)

A continuación se mostrara un ejemplo de los mensajes de las advertencias del sistema producidos al interior del sistema:



Formato del mensaje de advertencia:

Este seguro que desea eliminar (nombre registro)

O

Este seguro que desea editar (nombre registro)

Ejemplos de tipo de advertencias:

Advertencia [0]: Esta seguro que desea eliminar la mascota.

Advertencia [1]: Esta seguro que desea editar el usuario.

Advertencia [2]: Esta seguro que desea purgar la ficha.

Advertencia [3]: Esta seguro que desea eliminar la cita

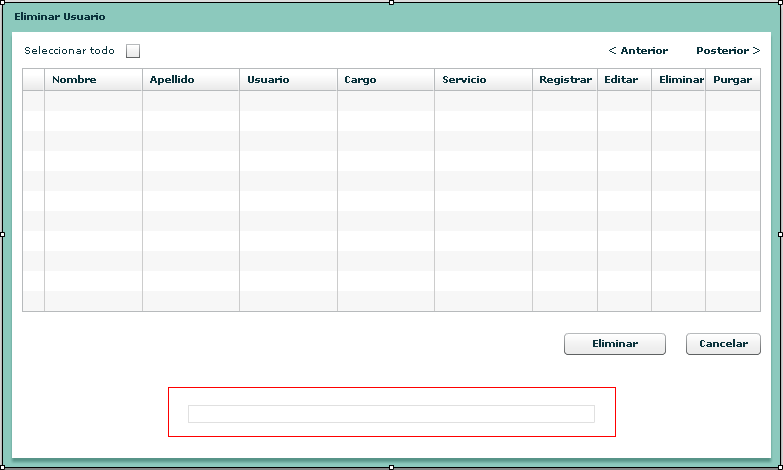
Advertencia [4]: Esta seguro que desea editar el usuario.

Advertencia [5]: Esta seguro que desea purgar el cliente.

(\*): Mayor información en Estándar de Flex/Advertencias del sistema

* + 1. **Notas:**

Las notas son mensajes menos importantes en el sistema, principalmente información de ayuda que no implica ningún riesgo para el usuario, por esto mismo va ubicado en sitio y en una forma que no interrumpa el normal funcionamiento de de las tareas. En la interfaz de flex se ha definido un espacio en la parte inferior, como se ve en la siguiente ilustración:



El formato que se espera para estas notas es el siguiente:

Nota de comando:

Presione “Comando” + acción que realiza

Ej. Presione Ctrl+G para guardar el documento

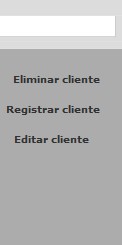
Nota de sugerencia:

Frase de sugerencia + acción que debe realizar

Ej. Puede eliminar todos los registros presionando “seleccionar todo + Eliminar”

* 1. **Formatos para los dispositivos de documentación para acceder a la información:**
     1. **Índice:**

Existirá un menú con todas las opciones, como se ve en la siguiente ilustración



## Referencias

**[1]** [**http://www.programacion.com/java/articulo/jap\_junit/**](http://www.programacion.com/java/articulo/jap_junit/)

**[2]** [**http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm**](http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm)

# **Anexo**

**Notacion**

* Los atributos en azul: Indican clave foránea
* Los atributos subrayados: Indican clave primaria.
* Los atributos en paréntesis: Indican el valor por defecto del atributo que está a la izquierda.
* En paréntesis cuadrados: Tipos de datos de cada atributo.

Usuario(nombre[char],

aMaterno[char],

aPaterno[char],

usuario[char],

idUsuario[serial],

contrasena[char],

cargo[char],

pRegistrar[bool],

pEditar[bool],

pEliminar[bool],

pPurgar[bool],

servicioNombre[char]

)

Servicio(nombre[char],

descripcion[char]

)

Cita(fecha[date],

horaInicio[time],

servicioNombre[char],

clienteRut [int]

)

ClienteWeb(rut[int],

telefono1[int],

telefono2[int],

nombre[char],

correo[char],

contraseña[char]

)

Aviso(descripcion[char],

fecha[date],

hora[time],

clienteWebRut[int]

)

ClientePresencial (rut[int],

telefono1[int],

telefono2[int],

nombre[char],

correo[char],

estado[bool],

domicilioNumero[int] ,

domicilioComuna[char],

domicilioCalle[char]

)

Producto(nombre[char],

Precio[Money],

descripcion[char],

categoria[char],

clientePresencialRut[int],

codigoBarra[int]

)

Mascota(clienteRut,

nombre[char],

fechaNacimiento[date],

sexo[char],

estado[bool],

raza[char],

foto

)

Atencion(clienteRut[int],

mascotaNombre[char],

mascotaFechaNacimiento[date],

servicio[char],

hora[time],

fecha[date]

)

AtencionPostOperatorio(medicamento[char],

alimento[char],

indicaciones[char],

servicio(pabellon) [char],

hora[time],

fecha[date]

)

AtencionPreOperatorio(observaciones[char],

sintomas[char],

diagnosticos[char],

servicio(pabellon) [char],

hora[time],

fecha[date]

)

AtencionCirugia(estado[bool],

diagnostico[char],

tipo[char],

servicio(pabellon) [char],

hora[time],

fecha[date]

)

AtencionConsulta(anamnesis,

servicio(policlinico) [char],

hora[time],

fecha[date]

)

AtencionControl(descripción[char],

recomendaciones[char],

proximaFecha[char],

servicio(policlinico)[char],

hora[time],

fecha[date]

);

AtencionVacuna(precio[money,

nombre[char],

descripcion[char],

fechaCaducidad[date],

servicio(policlinico) [char],

hora[time],

fecha[date]

);

AtencionAlojamiento(canil[char],

fechaSalida[date],

comentario[char],

solicitudes[char],

servicio(hoteleria) [char],

hora[time],

fecha[date]

);

AtencionBaño(tipo,

comentario[char],

servicio(peluqueria) [char],

hora[time],

fecha[date]

);

AtencionCorte(categoría[char],

nombre[char],

descripcion[char],

raza[char],

foto,

servicio(peluqueria) [char],

hora[time],

fecha[date]

);