**ACCESO A BASES DE DATOS DESDE JAVA (cont): Uso de JTable**

Un *ResultSet* básicamente es una tabla almacenada en memoria (y por tanto no visible). Veremos como trasladar el contenido de un *ResultSet* a un JTable para que el usuario pueda visualizar los datos de forma cómoda.

El proceso será básicamente el siguiente:

ResultSet

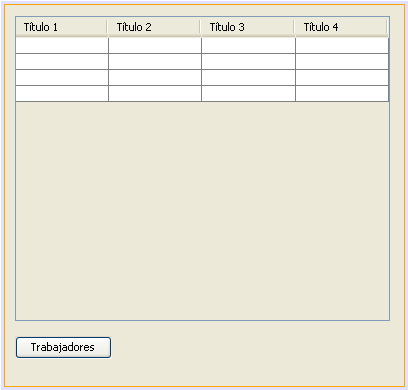
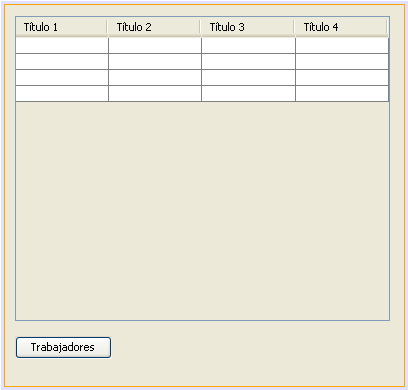
JTable

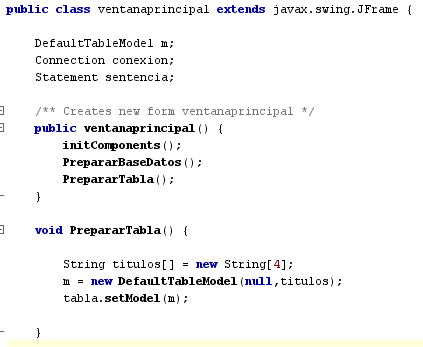
Base de Datos

**EJERCICIO 1.**

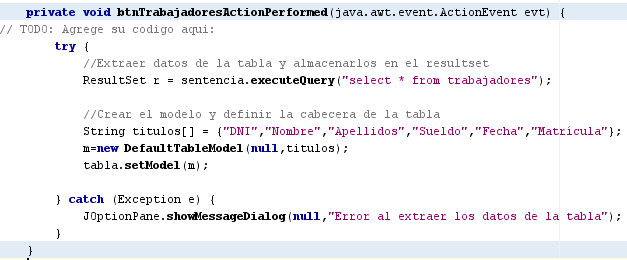
A continuación se pide una aplicación que muestre el contenido de una base de datos en un JTable.

1. Crear un proyecto java y prepararlo para que pueda acceder a la base de datos MANEMPSA.MDB:
   1. Declarar los dos objetos globales: *conexión* y *sentencia*.
   2. Programar el método *PrepararBaseDatos* de forma que el programa acceda a la base de datos y se construyan los objetos *conexión* y *sentencia*.
   3. Programar el evento *windowClosing* de la ventana principal de forma que se cierre la conexión.
2. Una vez hecho esto, añade a la ventana principal un objeto JTable y llámalo simplemente *tabla*.
3. Añade en la parte inferior un botón llamado *btnTrabajadores*. La ventana principal quedará así:



. 

1. Bien, ahora **programaremos el botón *btnTrabajadores*** de forma que al pulsarlo se muestre el contenido de la tabla Trabajadores. Programe lo siguiente:



Hasta el momento:

Lo que se hace es extraer el contenido de la tabla *Trabajadores* de la base de datos, ejecutando una consulta SQL usando el objeto *sentencia*. El resultado de dicha consulta se almacenará en un objeto *ResultSet* llamado r.

ResultSet r = sentencia.executeQuery("select \* from trabajadores");

A continuación, se extraerá información del *ResultSet* y se introducirá en la tabla. Lo primero que se hace es crear el modelo de la tabla definiendo la cabecera. Observa el código.

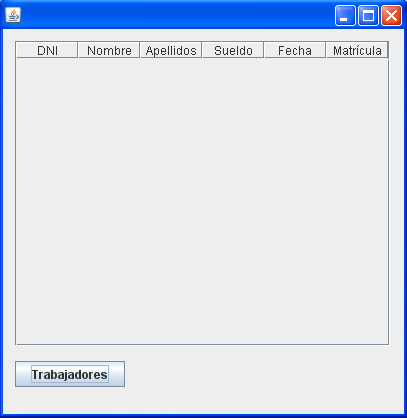
String titulos[] = {"DNI","Nombre","Apellidos","Sueldo","Fecha","Matrícula"};

m=new DefaultTableModel(null,titulos);

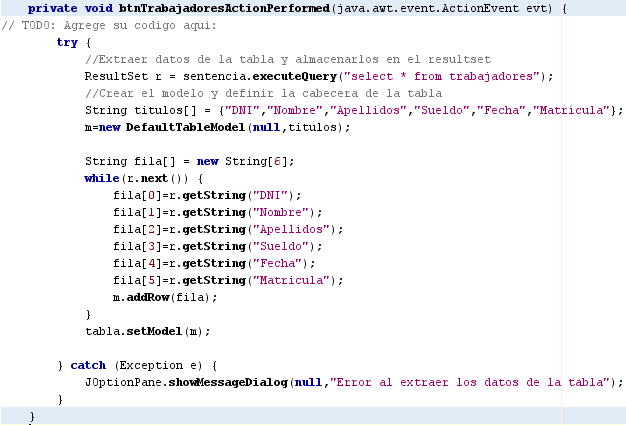
tabla.setModel(m);

Como ves, lo que hacemos es definir un vector de cadenas con los títulos correspondientes a los campos de la tabla, y luego usamos este vector para crear el modelo de la tabla.

Si ejecutamos tenemos:



1. Ahora ya solo nos queda recoger todo el contenido del *ResultSet* y mostrarlo en la tabla. Para ello, modifique el código del *actionPerformed* del botón *Trabajadores* para que quede como sigue:



Si estudiamos el código, veremos que se extrae la tabla trabajadores entera introduciéndola en un *ResultSet* llamado r, a través de una instrucción SELECT de SQL.

Luego construimos un modelo de tabla (*DefaultTableModel*) llamado m usando un vector de títulos, que se corresponde a los campos de la tabla trabajadores.

Una vez hecho esto, creamos un vector de seis elementos, correspondiente a los seis datos que contiene cada fila, y empezamos a recorrer el *ResultSet* usando el típico:

while (r.next()) {

En cada vuelta del bucle tomamos los valores de la fila del *ResultSet* y los almacenamos en el vector (en la posición correspondiente)

Luego se añade el vector que se ha construido al modelo de la tabla.

Así pues, en cada vuelta se añadirá un registro más al modelo m. Estos registros son extraídos del *ResultSet*.

Cuando el bucle ha terminado, el modelo de la tabla contiene exactamente lo mismo que el *ResultSet*. Es decir, contiene la tabla *trabajadores*, ya que este fue el resultado de la consulta SQL ejecutada.

Ya solo tenemos que asignar el modelo de la tabla al JTable correspondiente, es decir, a la tabla, con lo que esta visualizará los datos del modelo.

Todo este proceso se rodea con un try...catch para evitar errores inesperados.

**Resumiendo:**

###### Base de Datos 🡪 ResultSet 🡪 Vector 🡪 Modelo de Tabla 🡪 JTable

* + - De la base de datos extraemos datos a un *ResultSet*. (Esto se hace ejecutando una consulta SQL de tipo SELECT a través del objeto *sentencia*)
    - Del *ResultSet* extraemos los datos a un modelo de tabla. (Esto se hace recorriendo el *ResultSet* y almacenando cada registro en un vector)
    - Almacenamiento del vector en el modelo de tabla. (Cada vector extraido del *ResultSet* se inserta como fila en el modelo de tabla)
    - Se asigna el modelo de tabla al objeto JTable. (Al hacer esto el objeto JTable muestra el contenido del modelo)

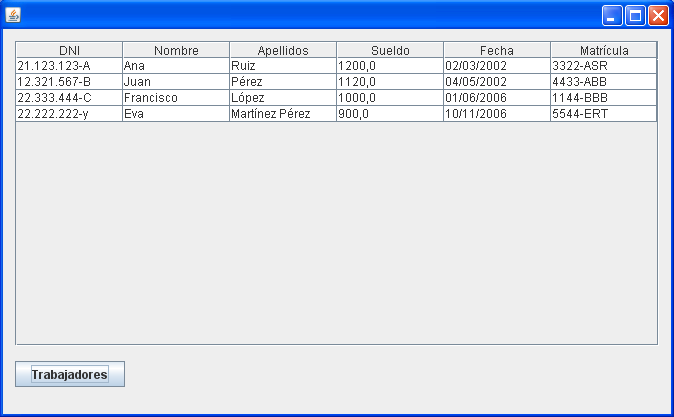
1. Si observamos la tabla, la fecha aparece en el formato estándar de almacenamiento:

Año-mes-dia hora:min:seg

Y que los sueldos aparecen con punto decimal.

Mejore estos detalles haciendo los siguientes cambios en el botón *Trabajadores*:





.

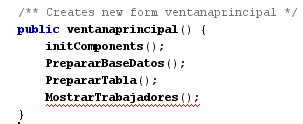
**EJERCICIO 2:**

Se realizará una pequeña aplicación de base de datos que permitía realizar *altas, bajas y modificaciones* sobre la tabla *trabajadores* de la base de datos MANEMPSA.

En este ejercicio vamos a reflejar todas las operaciones que realizamos sobre la base de datos en un JTable.

Por ejemplo, si eliminamos un registro automáticamente se verá la tabla actualizada en el JTable.

1. Crear un proyecto java y prepararlo para que pueda acceder a la base de datos MANEMPSA.MDB:
   1. Declarar los dos objetos globales: ***conexión*** y ***sentencia***.
   2. Programar el método ***PrepararBaseDatos*** de forma que el programa acceda a la base de datos y se construyan los objetos *conexión* y *sentencia*.
   3. Programar el evento ***windowClosing*** de la ventana principal de forma que se cierre la conexión.
   4. Programar el método ***PrepararTabla***
2. El primer objetivo será hacer que al ejecutar el programa aparezca automáticamente el contenido de la tabla *Trabajadores*  en el JTable. Para ello, realice los siguientes cambios en el código del programa:

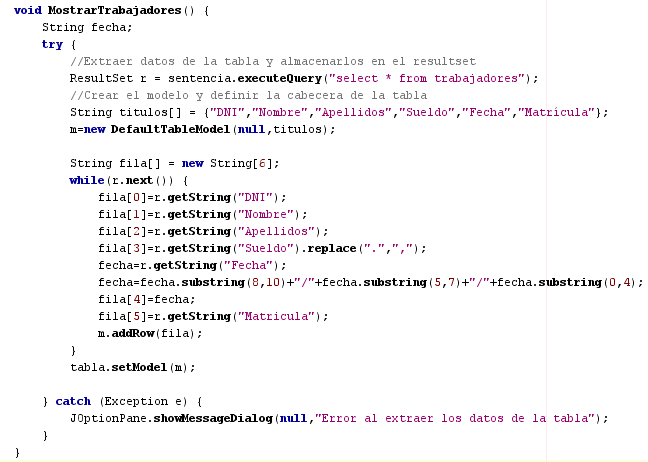


Añade una llamada a un método *MostrarTrabajadores*, en el constructor del programa.

Crea el método ***MostrarTrabajadore****s*:

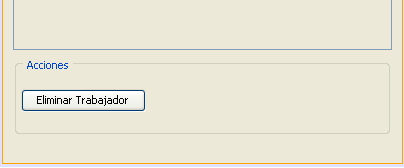


Copia en él el código del *actionPerformed* del botón *btnTrabajadores*del ejercicio anterior.



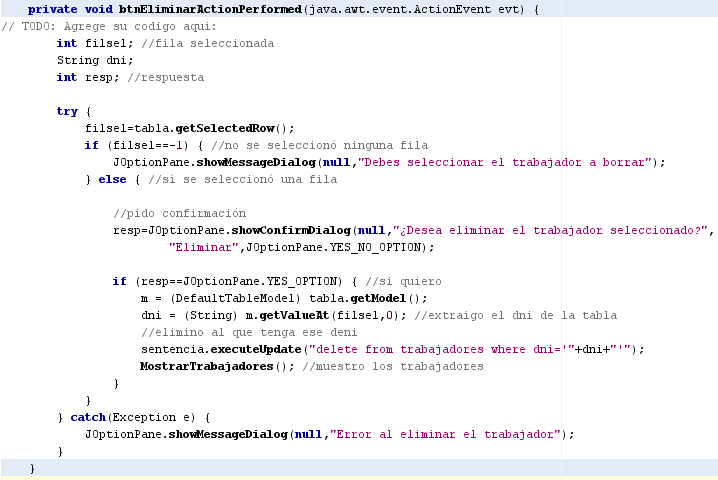
Lo que produce esta modificación del código es que al ejecutarse el programa se ejecute el método *MostrarTrabajadores*

1. **BAJAS:** Añade ahora en la parte inferior un **panel** llamado ***panelAcciones***y un **botón** *Eliminar Trabajador* llamado ***btnEliminar*:**



Se pretende que el usuario seleccione uno de los trabajadores de la tabla y al pulsar el botón *Eliminar* dicho trabajador se elimine de la base de datos. Esta eliminación por supuesto se verá reflejada en el JTable.

Para ello, programa en el botón ***Eliminar*** lo siguiente:



El código programado hace lo siguiente.

Lo primero que se hace es recoger la fila seleccionada de la tabla usando el método *getSelectedRow*. Si el valor devuelto es –1 entonces es que no hay ninguna fila seleccionada. El programa avisa de esta circunstancia.

Si se seleccionó a un trabajador, entonces podemos borrar. Pero antes, es interesante pedir confirmación. Esto es lo que se hace con el JOptionPane.showConfirmDialog.

Si el usuario acepta la eliminación del trabajador, entonces la llevamos a cabo. Observa el proceso:

* Extraemos del modelo del JTable el dni del trabajador seleccionado (este dni se encuentra en la fila seleccionada –filsel- columna 0 –la primera columna-):

m = (DefaultTableModel) tabla.getModel();

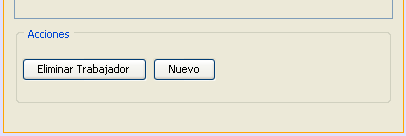
dni = (String) m.getValueAt(filsel,0);

* Ahora se construirá una instrucción de acción SQL del tipo DELETE para que se elimine el trabajador con el dni extraído. Esto se hace concatenando y ejecutando la instrucción SQL a través del objeto *sentencia*:

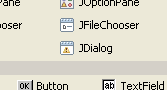
sentencia.executeUpdate("delete from trabajadores where dni='"+dni+"'");

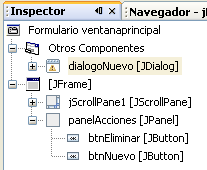
* Y finalmente se llama al procedimiento *MostrarTabla* para que se extraiga de la base de datos y muestre de nuevo la tabla *trabajadores*. Esto actualizará el JTable, y se podrá ver que el trabajador ha sido eliminado.
* Todo esto está dentro de un try...catch para capturar errores inesperados.

1. **ALTAS:** Ahora añade otro botón llamado  ***Nuevo***

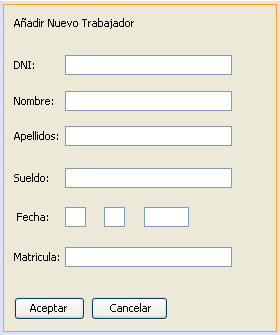


Cuando se pulse el botón ***Nuevo*** se pretende que aparezca un formulario donde se puedan introducir los datos de un nuevo trabajador. Esto se conseguirá añadiendo un cuadro de diálogo a nuestro proyecto. Para ello, **agrega un JDialog al proyecto**. Este diálogo se llamará ***dialogoNuevo*.**





Hacer doble clic en el *Inspector* sobre el ***dialogoNuevo*** para diseñarlo. Debe quedar como sigue, con los nombres que se indican a continuación:



txtNuevoDni

txtNuevoNombre

txtNuevoApellidos

txtNuevoSueldo

txtNuevoDia txtNuevoMes txtNuevoAnio

txtNuevoMatricula

btnNuevoAceptar

btnNuevoCancelar

La idea es la siguiente. Cuando el usuario pulse el botón *Nuevo*, aparecerá este cuadro de diálogo. El usuario introducirá los datos del nuevo trabajador y pulsará Aceptar, y entonces estos datos se introducirán en la tabla *trabajadores*.

Si el usuario pulsa Cancelar, entonces no se hará nada.

El JTable se actualizará para mostrar el resultado de la inserción del nuevo trabajador.

Entra en el *actionPerformed* del botón *btnNuevo* y programa lo siguiente:



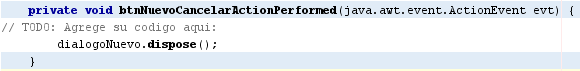
Este código empieza asignando un tamaño al cuadro de diálogo *dialogoNuevo*.

Luego, se define el *dialogoNuevo* como Modal. Esto significa que hasta que no se termine de trabajar con este cuadro de diálogo no se podrá continuar usando el programa principal.

Luego se muestra dicho cuadro de diálogo.

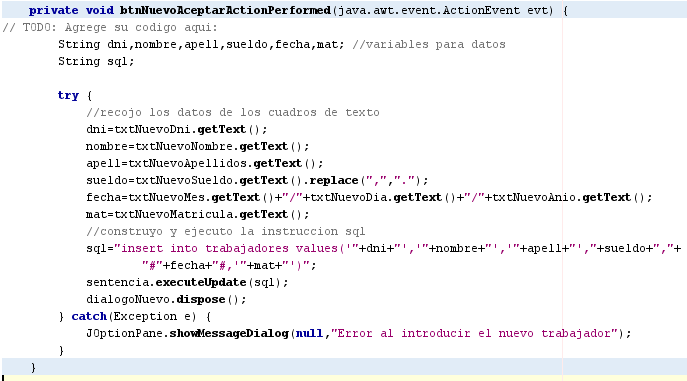
Y finalmente se actualiza el JTable por si se hubiera introducido un nuevo trabajador.

* 1. Ahora hay que programar los botones del cuadro de diálogo *dialogoNuevo*. Empecemos por el botón ***Cancelar*:**



El botón *Cancelar* debe limitarse a quitar de la pantalla el cuadro de diálogo *dialogoNuevo*.

* 1. Y finalmente se programará el botón ***Aceptar*** del cuadro de diálogo. Recuerda que este botón es el que introduce en la base de datos a un nuevo trabajador. Programa dentro de este botón lo siguiente:



Vamos a analizar detenidamente este código.

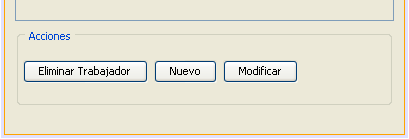
Lo primero que puedes observar es un conjunto de líneas que copian el contenido de los cuadros de texto en variables de cadena. Al hacer esto, cambiamos la coma decimal por punto decimal en el caso del sueldo y configuramos la fecha en el formato que entiende SQL: mes/dia/año.

Luego, con estas variables, se construye por concatenación una instrucción SQL de tipo INSERT que permita introducir los datos del nuevo trabajador en la tabla *trabajadores*.

Se ejecuta dicha instrucción SQL. Y se cierra el cuadro de diálogo.

Por supuesto, es necesario un try...catch para evitar problemas inesperados.

1. **MODIFICAR:** Nuestro programa ya puede hacer altas y bajas. Solo queda que pueda realizar modificaciones. Para ello añade un nuevo **botón** a la ventana, llamado ***btnModificar*:**

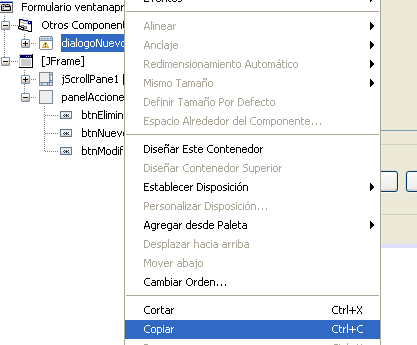


Se pretende que el usuario seleccione en la tabla el trabajador cuyos datos quiere modificar, y luego pulse este botón para efectuar la modificación.

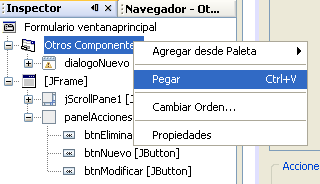
Al pulsar este botón debe aparecer un cuadro de diálogo donde el usuario pueda cambiar fácilmente los datos.

Ese cuadro de diálogo será muy parecido al que hemos hecho antes, así que básicamente solo tendrás que hacer una copia de dicho cuadro de diálogo y modificarlo un poco.

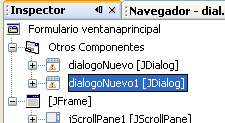
* 1. Haz clic con el derecho sobre el cuadro de diálogo *dialogoNuevo* y activa *Copiar* :



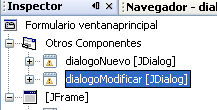
* 1. Luego activa *Pegar* sobre *Otros Componentes*:



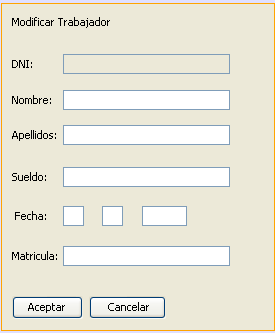
* 1. Aparecerá un nuevo cuadro de diálogo que es una copia del anterior. Este cuadro tendrá como nombre *dialogoNuevo1*.



* 1. Sin embargo, le cambiaremos el nombre para que sea más acorde con su función. Le llamaremos *dialogoModificar*. Cámbiale el nombre:



* 1. Vamos a modificar un poco el diseño del *dialogoModificar*. Haz doble clic sobre él y realiza las siguientes modificaciones en el diseño:



Cambia el título. Ahora es “Modificar Trabajador”

Los nombres para los cuadros de texto y botones serán:

txtModDni (Desactiva su propiedad *editable)*

txtModNombre

txtModApellidos

txtModSueldo

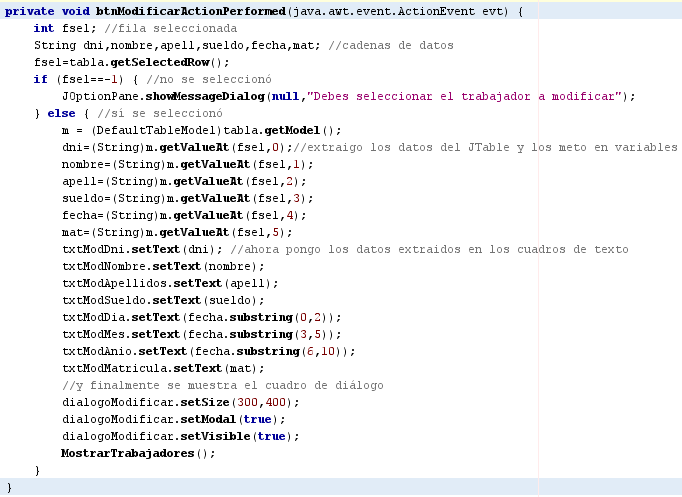
txtModDia txtModMes txtModAnio

txtModMatricula

btnModAceptar btnModCancelar

* 1. Empezaremos programando el botón ***Modificar***. Al pulsar este botón se debe mostrar el cuadro de diálogo anterior relleno con los datos del trabajador que se quiere modificar. Se supone que el usuario ha seleccionado a este trabajador en la tabla anteriormente.

Entra en el *actionPerformed* del botón ***Modificar***y programa lo siguiente:



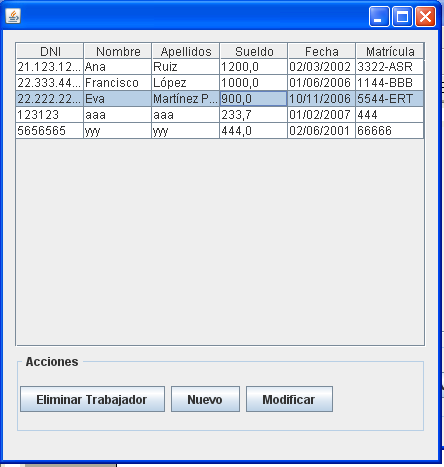
Qué hace este código:

Primero se comprueba el número de la fila seleccionada. Si no hubiera ninguna se muestra un mensaje de error, ya que es necesario modificar la fila del trabajador que se quiere modificar.

En el caso de que haya una fila seleccionada, se extraen los datos del modelo del JTable y se almacenan en varias variables de cadena.

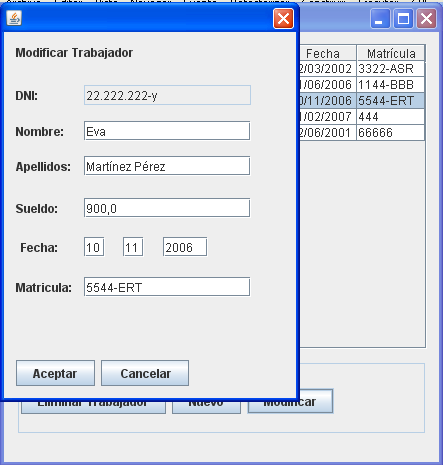
Una vez hecho esto, esas mismas variables se almacenan en los cuadros de texto del cuadro de diálogo *dialogoModificar*.

Y finalmente se prepara el cuadro de diálogo *dialogoModificar* y se muestra en la pantalla. Una vez realizada la modificación (no programada aún) se muestran los trabajadores en la tabla llamando al método *MostrarTrabajadores*.



Selecciona...

Y luego pulsa modificar.



El resultado es que aparece el cuadro de diálogo *dialogoModificar* ya relleno con los datos del trabajador que se seleccionó.

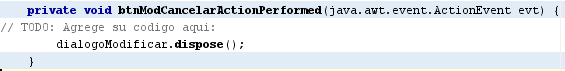
Lo bueno que tiene el rellenar el cuadro de diálogo *dialogoModificar* con los datos del trabajador que se quiere modificar es que no tenemos que escribir todos los datos, y solo modificar el campo que nos interese.

El usuario realizará los cambios en los cuadros de textos ya rellenos y luego pulsará el botón *Aceptar* para que se produzca la modificación.

Si se pulsa *Cancelar* no sucede nada. Simplemente se cerrará el cuadro de diálogo.

* 1. A continuación se programan los botones ***Cancelar*** y ***Aceptar*** del cuadro de diálogo.

Se empezará programando el botón *Cancelar*. Este botón debe limitarse a cerrar el cuadro de diálogo *dialogoModificar*:

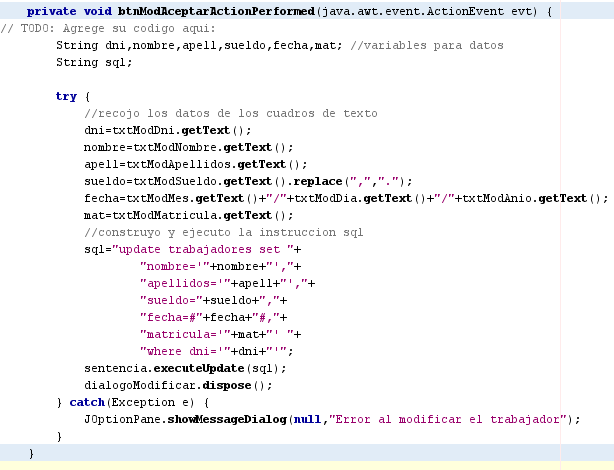


Ahora nos centraremos en el botón *Aceptar*. Este botón debe realizar la modificación, introduciendo todos los datos de los cuadros de texto en la tabla trabajadores.

El código siguiente tomará como referencia el DNI del trabajador que se está modificando. Es **importante que este DNI no cambie**, ya que entonces estaríamos modificando los datos de otro trabajador.

Esta es la razón por la que el **cuadro de texto *txtModDni*** se ha configurado como ***no editable***, para evitar que el usuario pueda cambiarlo accidentalente.

Programa el siguiente código en el botón *Aceptar*:



Estudiemos el código.

Lo primero que se hace es recoger en variables los datos introducidos en los cuadros de texto.

Luego, con estas variables, se construye una instrucción SQL del tipo UPDATE que permite modificar los datos del trabajador con el dni indicado en el cuadro de diálogo.

Finalmente se cierra el cuadro de diálogo.

Todo este código es susceptible de sufrir fallos por lo que está rodeado de un try...catch.

**CONCLUSIÓN**

**La más simple de las aplicaciones de base de datos debe ser capaz de realizar las siguientes operaciones sobre una tabla:**

* **Altas**
* **Bajas**
* **Modificaciones**

**Es muy interesante que la aplicación muestre en pantalla continuamente un JTable con el contenido de la tabla de la base de datos sobre la que se está trabajando.**

**El JTable nos permite visualizar los datos de la tabla y seleccionar rápidamente el registro que queramos manipular.**

**Para la inserción de nuevos registros en la tabla se recomienda la creación de un cuadro de diálogo que muestre un formulario donde el usuario pueda introducir los datos del nuevo registro cómodamente.**

**Para la modificación de un registro, se recomienda la creación de otro cuadro de diálogo que muestre los datos del registro que se quiere modificar.**

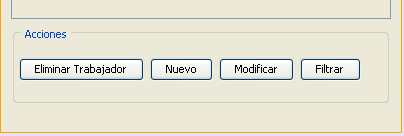
**EJERCICIO 3:**

Puede ser necesario a veces ver sólo determinados registros de la tabla y no toda la tabla entera.

Se hace necesario pues añadir al programa ciertas **opciones de filtrado** que nos permitan visualizar en el JTable solo aquellos registros que más nos interesen.

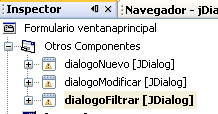
En este ejercicio se mejorará el programa anterior de forma que permita al usuario ciertas opciones de filtrado.

1. Abrir la aplicación del ejercicio anterior.
2. Primero añade un botón ***btnFiltrar*** a la ventana:



1. Al pulsar este botón aparecerá un cuadro de diálogo que contenga opciones de filtrado, de forma que el usuario pueda decidir qué trabajadores quiere ver.

Por lo tanto, tendrá que añadir un nuevo cuadro de diálogo (JDialog) y asígnele el nombre ***dialogoFiltrar*.**



1. Haz doble clic sobre este diálogo y diséñelo de la siguiente forma:



txtFiltrarDni

txtFiltrarNombre

txtFiltrarApellidos

comboSueldo

txtFiltrarSueldo

comboFecha

txtFiltrarDia txtFiltrarMes txtFiltrarAnio

txtFiltrarMatricula

btnFiltrarAceptar

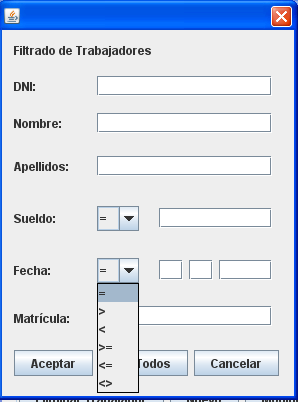
btnFiltrarVerTodos

btnFiltrarCancelar

NOTA: Los dos combos del cuadro de diálogo deben contener los siguientes elementos:

= > < >= <= <>

Es decir, cuando se desplieguen aparecerá esto:

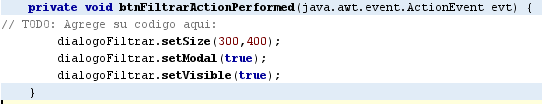


1. La idea es la siguiente. Cuando el usuario pulse el botón *Filtrar*, se mostrará este cuadro de diálogo. Entonces el usuario introducirá las condiciones de búsqueda y pulsará *Aceptar*, y entonces se visualizarán en el JTable aquellos registros que cumplan la condición.

Si el usuario pulsa el botón *Ver Todos*, entonces se mostrarán en el JTable todos los trabajadores.

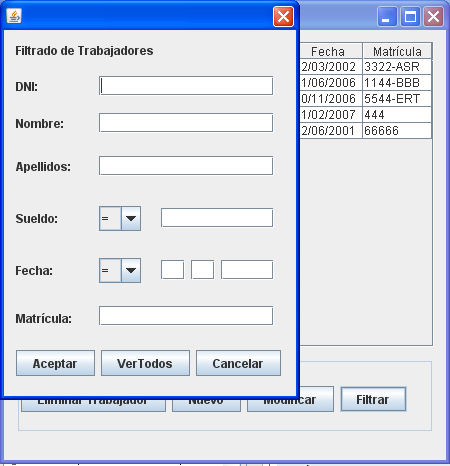
Si el usuario pulsa el botón *Cancelar*, entonces el cuadro de diálogo de filtrado se cierra sin más.

1. Empezaremos **programando el botón *Filtrar*** de la ventana principal. Introduzca en él el siguiente código:



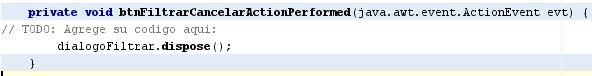
Lo único que hace este código es asignar un tamaño al cuadro de diálogo, configurarlo como modal, y finalmente presentarlo en pantalla.

1. Ejecute el programa si quiere ver el funcionamiento de este botón.



Se pulsa el botón Filtrar y aparece el cuadro de diálogo de filtrado.

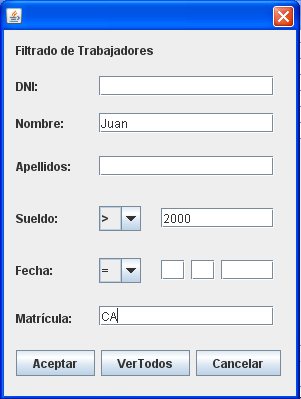
1. Programemos ahora el **botón *Cancelar*** del cuadro de diálogo de filtrado. La programación de este botón es muy sencilla:



Como puede ver consiste simplemente en cerrar el cuadro de diálogo de filtrado.

1. Se programará ahora el **botón *Aceptar***, pero antes, es necesario entender como funcionará el cuadro de diálogo de filtrado.

El usuario introducirá los datos que quiere buscar. Ten en cuenta que no tiene por qué rellenar todas las casillas. Además, puede hacer uso de los combos asignados a los campos sueldo y fecha. Por ejemplo, supongamos que el usuario introdujera los siguientes datos:



Si en este momento el usuario pulsara el botón *Aceptar*, se tendrían que mostrar aquellos trabajadores quí cumplan todas las siguientes condiciones: **que tengan de nombre *Juan,* que ganen más de 2000 euros de sueldo y que conduzcan un coche cuya matrícula contenga la cadena “*CA*”.**

Las casillas que estén vacías no se tendrán en cuenta a la hora de hacer el filtrado.

Si el usuario introduce una cadena en un campo de tipo texto, se buscará a todos aquellos trabajadores que contengan dicha cadena. Por ejemplo, si introduzco “Juan” en el nombre, el programa buscará a los trabajadores cuyo nombre contenga la palabra “Juan”. De esta manera la búsqueda será mucho más cómoda, al no tener que introducir el texto completo, y producirá más resultados (se encontrarán a los que se llamen “Juan”, “Juan Antonio”,”Juana”, etc.

1. El botón *Aceptar* tendrá que construir una consulta SELECT que incluya estas condiciones, y luego ejecutarla. Finalmente tendrá que mostrar el resultado de la consulta en el JTable.

El código de este botón será largo, así que veamos poco a poco como programarlo. Empiece programando lo siguiente dentro del botón *Aceptar*:

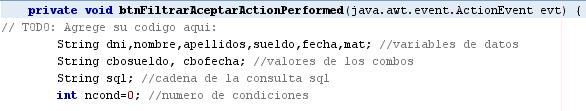


Como puede ver, empezamos recogiendo el contenido de los cuadros de texto en distintas variables. También recogemos los valores seleccionados en los combos del sueldo y fecha.

En el caso de la fecha, recogemos de los cuadros de texto el día, mes y año y los concatenamos para formar una fecha que Access pueda aceptar: mes/dia/año.

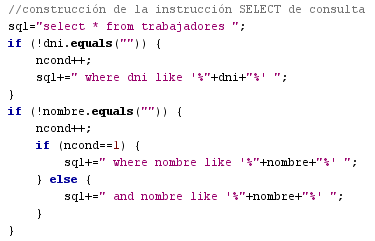
En el caso del sueldo cambiamos la coma decimal (ya que suponemos que el usuario introducirá el valor en formato español) por el punto, para que no haya problemas a la hora de ejecutar la consulta SQL.

1. Siga programando en el botón *Aceptar* de la siguiente forma:



Agrega estas dos variables en la parte inicial del método.

**...........** *código anterior***...........**



Y en la parte final del método añade el siguiente código...

Es necesario estudiar muy detenidamente el código que tenemos programado hasta ahora antes de continuar.

Se han añadido dos variables:

* La variable *sql* es una variable de cadena que contendrá la instrucción SELECT que efectuará el filtrado (la consulta)
* La variable *ncond* (número de condiciones) es una variable numérica que va contando cuantas condiciones hay. Ten en cuenta que habrá una condición cada vez que un cuadro de texto esté relleno con algún dato.

Esta variable es muy importante, ya que nos permite saber si tenemos que concatenar una condición con “where” o con “and”. Ten en cuenta que la primera condición que se añade vendrá precedida por “where”, pero las demás estarán precedidas por “and”.

La primera condición que se añade a la instrucción SELECT tendrá que tener esta forma (en rojo):

select \* from trabajadores where *condicion1*

La condición 2, condición 3, etc llevan el “and” delante. Por ejemplo, la segunda condición tendría esta forma (en azul):

select \* from trabajadores where *condicion1* *and condicion2*

Si añadiéramos una tercera condición también llevaría el “and” (en verde):

select \* from trabajadores where *condicion1* *and condicion2 and condicion3*

Etc.

Resumiendo, la primera condición vendrá antecedida de “where”, mientras que el resto de las condiciones vendrán antecedidas del “and”.

Si se observa el código añadido al final del método, en él se empieza construyendo el comienzo de la consulta SELECT: “*select \* from trabajadores*” y luego se añaden condiciones según si el cuadro de texto correspondiente está vacío o no.

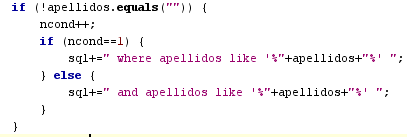
Cada vez que se encuentra un cuadro de texto no vacío, se añade una condición y se aumenta en 1 el contador de condiciones, es decir, la variable *ncond*.

Observa como cuando se añade una condición, se comprueba con un if el número de condiciones (*ncond*) para saber si hay que concatenar la condición con el “where” o con el “and”.

Debes observar también como nos decantamos por el uso de LIKE ya que este es más versátil. Y se recuerda una vez más que cuando usemos LIKE desde java en vez de asteriscos usaremos el símbolo tanto por ciento (%)

Estudia bien este código para entenderlo. Solo se ha indicado las posibles condiciones para el DNI y para el Nombre. Sin embargo será necesario añadir más if para el resto de los campos.

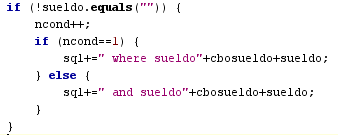
1. Añada al final del código anterior el siguiente código:



Este código comprueba si hay algún dato en el cuadro de texto apellidos, y, si es así, entonces crea una condición para filtrar por apellidos.

Se incrementa el contador de condiciones en uno y si estamos ante la primera condición esta se concatena usando “where”, y si no, esta se concatena usando “and”.

1. Añada al final del código anterior el siguiente código:

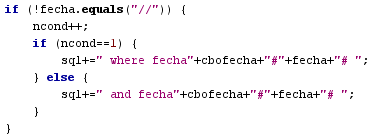


Este es el código que añade una condición para el sueldo (suponiendo que haya algún dato en el cuadro de texto del sueldo)

Como en los casos anteriores, se suma uno al contador de condiciones y, si estamos ante la primera condición se concatena con “where”, y si no, se concatena con “and”.

Hay que destacar aquí que, al ser el sueldo un campo numérico, no se usa LIKE, sino que se usa el operador que el usuario haya escogido dentro del combo del sueldo. Recuerda que ese operador puede ser uno de los siguientes: =, >, <, >=, <= , <>

1. Añada al final del código anterior el siguiente código:



Este es el código que añade una condición para la fecha (suponiendo que haya algún dato en los cuadros de texto correspondientes a las fechas)

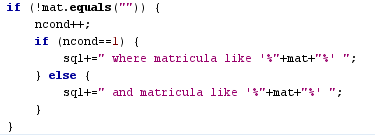
Aquí hay que tener en cuenta que los valores de los cuadros de texto del día, mes y año se concatenan dentro de una variable *fecha* usando la / como separador. Si los cuadros de texto dia, mes y año estuvieran vacíos, la variable *fecha* contendría la cadena “//”. Por eso preguntamos si la variable fecha es distinta de “//” para saber si es necesario añadir una condición para la fecha.

Como en los casos anteriores, se suma uno al contador de condiciones y se comprueba si estamos ante la primera condición. En este caso se concatena la condición usando “where” y en caso contrario se concatena la condición usando “and”.

Observa que para crear la condición se usa el operador que el usuario haya elegido en el combo de la fecha, el cual puede ser uno de los siguientes: =, >, <, >=, <= , <>

Ya que estamos manejando fechas recuerda que hay que añadir las almohadillas en la condición.

1. Añada el siguiente código al final del código anterior.



En este caso tenemos la construcción de la condición correspondiente a la matrícula del coche del trabajador. Por supuesto, esta condición solo se construye si se ha escrito algo en el cuadro de texto de la matrícula.

Como en los casos anteriores se suma uno al contador de condiciones y si estamos ante la primera condición entonces la condición se construirá usando “where”, y en caso contrario añadiremos esta condición usando “and”.

Como en los demás campos de tipo texto, para construir esta condición se ha preferido usar LIKE junto con los % (asteriscos en Access), que da más posibilidades de búsqueda.

1. El resultado final de todo este largo código es que la variable *sql* contendrá una instrucción SELECT que realizará un filtrado sobre la tabla *trabajadores* de la base de datos de forma que se extraigan a los trabajadores que cumplan las condiciones indicadas por el usuario en el cuadro de diálogo de filtrado.

Lo que hay que hacer ahora es ejecutar dicha instrucción SELECT. Y como ya sabe, esto se hace a través del objeto *sentencia*. Programe lo siguiente a continuación del código anterior:

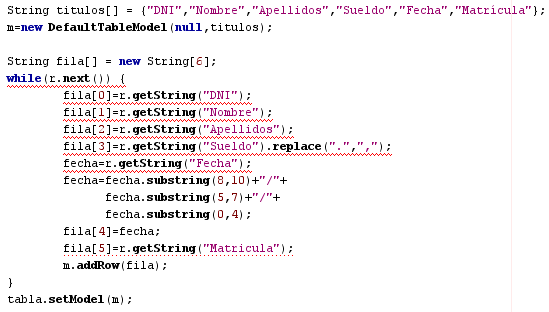


En este código se ejecuta la consulta creada y el resultado se introduce en un *ResultSet* r.

Ahora hay que mostrar el contenido del *ResultSet* r en el JTable de la ventana principal.

Como ve, el código da un error. Esto es debido a que es obligatorio rodear el código con un try...catch, ya que es susceptible de error. (Esto se hará más adelante)

1. Ahora mostraremos el *ResultSet* r en el JTable, así que añada el siguiente código a continuación:



Este código ya debe resultar conocido, ya que se usó también en el método *MostrarTrabajadores*, encargado de mostrar toda la tabla *trabajadores* en el JTable.

Si observa este código verá que lo que hace es definir un vector de títulos para el JTable. A partir de este vector de títulos se crea el modelo del JTable (*DefaultTableModel*)

Luego se recorre el *ResultSet* r, que contiene ahora mismo el resultado del filtrado realizado, y se extrae cada fila almacenándola en un vector al que se ha llamado *fila*.

Cada fila extraída se introduce en el modelo m de la tabla.

Finalmente cuando se han traspasado todas las filas desde el *ResultSet* r al modelo m, se asigna dicho modelo al JTable. Esto quiere decir que el JTable debería mostrar ya el resultado del filtrado.

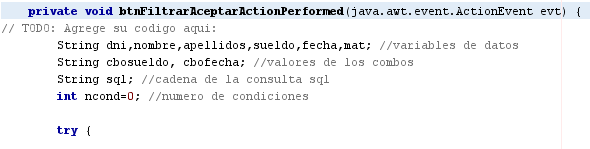
La razón de que aparezcan tantos errores es porque aún no hemos añadido el try...catch. Esto lo haremos al final.

1. El código programado hasta ahora analiza los datos introducidos en el cuadro de diálogo de filtrado y construye una consulta SELECT que extrae los trabajadores que cumplan las condiciones indicadas. Estos datos se muestran finalmente en un JTable. Lo único que queda por hacer es cerrar el cuadro de diálogo.

Añade por tanto el siguiente código:



1. Todo el código programado en el botón *Aceptar* es bastante delicado y pude dar gran cantidad de errores, por lo que será necesario rodearlo dentro de un try...catch:

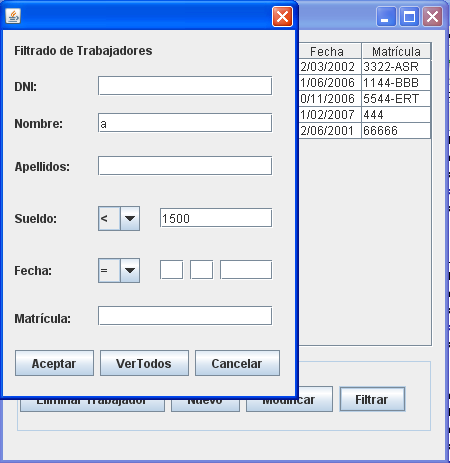


**.... resto del código...**



1. El código que se acaba de programar es un poco enrevesado y largo, pero le da al programa una potencia tremenda, ya que permite al usuario realizar fácilmente búsquedas y filtrados en la tabla trabajadores.

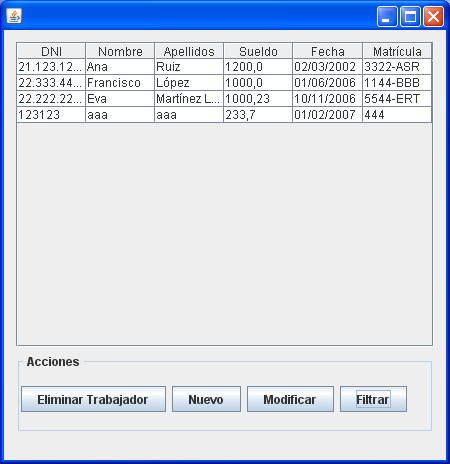
Ejecuta el programa y haz lo siguiente:



Pulsa *Filtrar* para ver el cuadro de diálogo de filtrado.

Queremos buscar a los trabajadores que tengan una “a” en el nombre y cobren menos de 1500 euros. (Recuerda activar el combo)

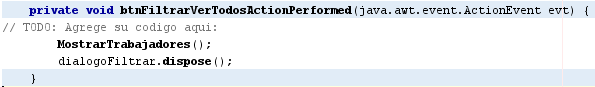
Pulsa Aceptar para ver el resultado.



1. Como habrá podido observar en este ejemplo, se pueden realizar búsquedas en la tabla trabajadores gracias al código programado. Esto es muy útil sobre todo en el momento en que la tabla contenga muchos registros.

Pruebe a ejecutar más veces el programa y realice varios filtrados.

1. Solo queda por programar el botón ***Ver Todos***del cuadro de diálogo de filtrado. Al pulsar este botón se pretende que se visualicen todos los registros de la tabla *trabajadores*. El código de este botón es bastante sencillo:

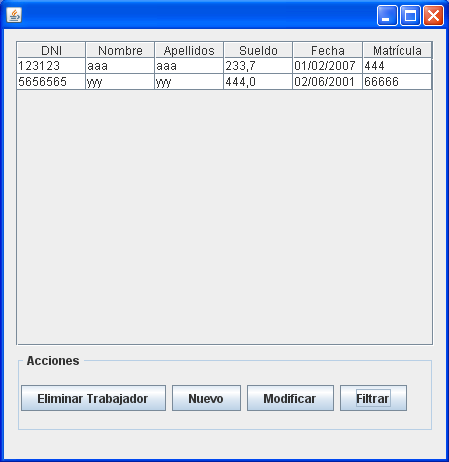


Como ves aprovechamos el método ya programado *MostrarTrabajadores* que se encarga de mostrar todos los trabajadores en el JTable. Luego solo tenemos que descargar el cuadro de diálogo de filtrado.

1. Ejecuta el programa y prueba lo siguiente:

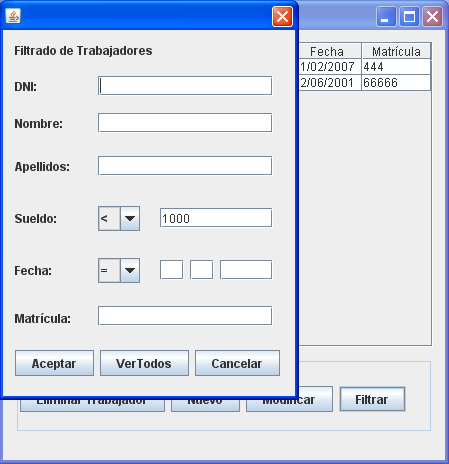
* Haz el filtrado que quieras sobre la tabla trabajadores. Acepta y observa el filtrado en la tabla.
* Vuelve a filtrar pero esta vez pulsa el botón *Ver Todos*. Observarás como vuelven a verse todos los trabajadores en la tabla.

Ejemplo:

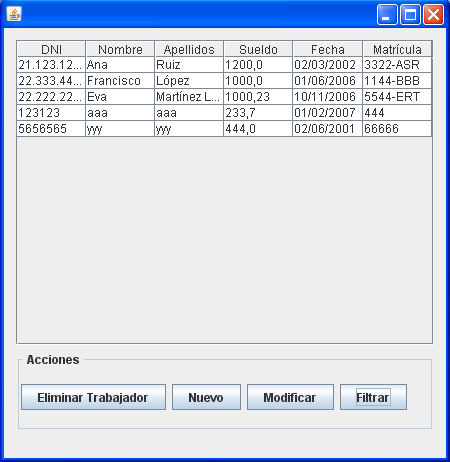


Se acaba de realizar un filtrado, la tabla muestra solo los trabajadores que cumplen la condición...

Pulsa ahora Filtrar



Pulsa Ver Todos para volver a ver todos los trabajadores.



Y el resultado será que se quita el filtrado y se vuelven a mostrar todos los trabajadores

**CONCLUSIÓN**

**Las tablas que tengan muchos datos son incómodas de manejar, por lo que resulta muy interesante añadir al programa ciertas opciones de filtrado. Estas opciones permitirán visualizar solo aquellos registros que cumplan determinadas condiciones.**

**Las opciones de filtrado consistirán básicamente en la construcción de una consulta SELECT a través de concatenaciones de cadenas.**

**Una vez construida la cadena SELECT según el filtrado que quiera hacer el usuario, se tendrá que ejecutar dicha consulta y se tendrá que mostrar el resultado. Normalmente en un JTable.**

**EJERCICIO 4: Ordenación y cálculos**

Se van a añadir ciertas mejoras al programa que se ha realizado en el ejercicio anterior. En este ejercicio se programarán ciertas opciones de ordenación y se realizarán determinados cálculos sobre la tabla trabajadores.

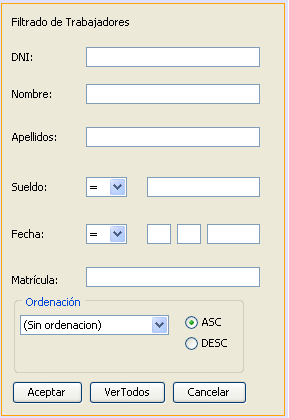
Todo programa de gestión de datos de una tabla deber tener opciones de ordenación para mostrar los datos de la tabla ordenados como el usuario quiera.

La opción de ordenación puede estar situada junto a las opciones de filtrado para permitir de esta manera que los filtrados aparezcan ordenados según le interese al usuario.

Además suele ser habitual que al lado del JTable aparezcan ciertos cálculos estadísticos relativos a la tabla.

1. Abrir la aplicación del ejercicio anterior.
2. Vamos a modificar el cuadro de diálogo de filtrado de forma que no sólo sea capaz de filtrar, sino que también permita que el listado que se ha filtrado aparezca ordenado según un campo.

Para ello, añadir lo siguiente al diálogo de filtrado:



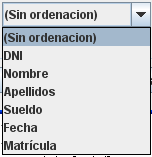
panelOrdenacion

cboCamposOrdenacion

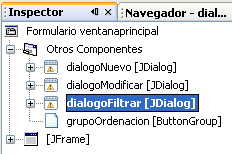
radioASC

radioDESC

NOTA Nº1: El combo *cboCamposOrdenacion* contendrá los siguientes elementos: *(Sin Ordenación),* *DNI, Nombre, Apellidos, Sueldo, Fecha, Matricula*:



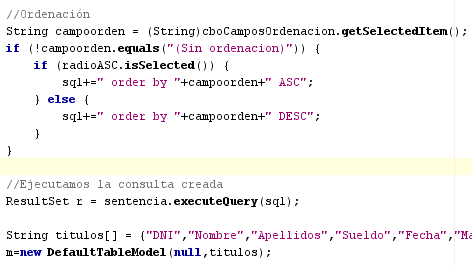
NOTA Nº2: Es necesario añadir un *ButtonGroup* al programa, al que llamaremos *grupoOrdenacion*. Los botones de opción *radioASC y radioDESC* deben pertenecer a dicho grupo.



1. La idea es la siguiente. Cuando el usuario quiera hacer un filtrado podrá indicar que el listado aparezca ordenado por algún campo, seleccionándolo en el combo de ordenación. También puede indicar la forma de ordenación (ascendente o descendente) a través de los botones de opción.

Al pulsar Aceptar el listado de trabajadores no solo saldrá filtrado, sino que también saldrá ordenado.

Para ello hay que realizar **modificaciones en el código del botón *Aceptar***. Añade el siguiente código en el *actionPerformed* del botón *Aceptar* (justo antes de la ejecución de la consulta):

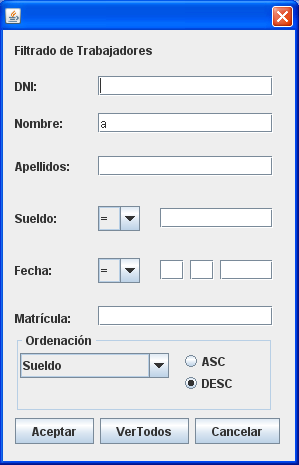


Estudiemos el código. Lo primero que se hace es extraer el valor del combo desplegable de los campos de ordenación.

Si este combo no contiene el valor “*(Sin ordenación)*” significará que se desea realizar una ordenación por el campo seleccionado. Así pues se concatena a la variable *sql* la cláusula *order by* con el campo que se ha seleccionado en el combo.

Se controla también la forma de ordenación teniendo en cuenta el botón de opción que esté activado, y según esto, se añade la cadena “ASC” o “DESC” para ordenar ascendentemente o descendentemente.

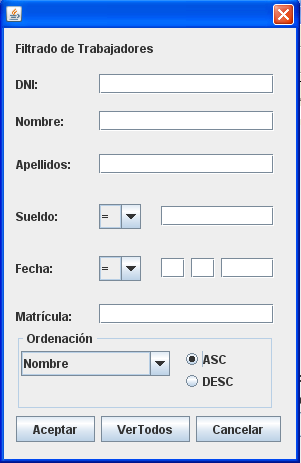
1. Ejecuta el programa y prueba a hacer un filtrado eligiendo un campo para ordenar y observa el resultado:



En este ejemplo se pretende visualizar a todos los trabajadores que contengan una “a” en su nombre...

... y el listado saldrá ordenado por el campo sueldo descendentemente.

1. Una ventaja de esto es que se puede mostrar el listado completo de trabajadores ordenado por el campo que se quiera siempre y cuando no se indique ninguna condición:

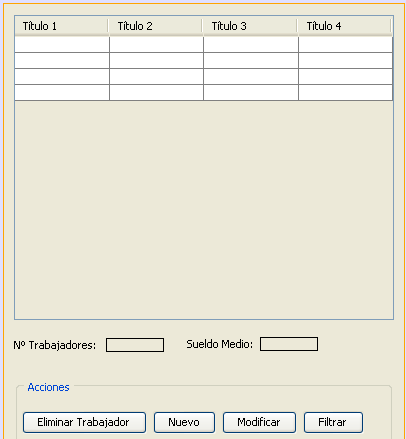


En este ejemplo aparecerían todos los trabajadores, ya que no se ha indicado ninguna condición.

... y el listado saldrá ordenado por el nombre ascendentemente.

1. Ahora vamos a añadir una nueva mejora al programa. La idea es que se visualice junto al JTable de trabajadores los siguientes datos: el número de trabajadores que se muestra y la media de los sueldos.

Para ello, añade las siguientes modificaciones en el diseño de la ventana principal:



Etiquetas con borde:

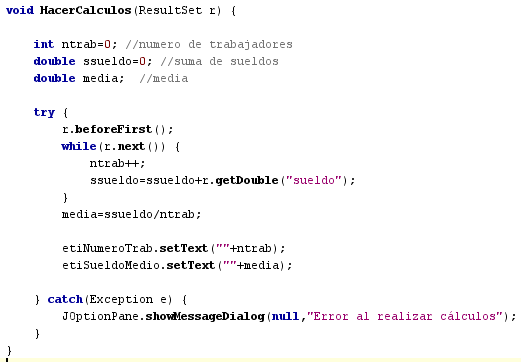
etiNumeroTrab

etiSueldoMedio

1. Ahora programaremos un método que sea capaz de calcular el número de trabajadores y el sueldo medio a partir del contenido de un *ResultSet*.

Este método recibirá un *ResultSet* como parámetro, y lo que hará será analizar el contenido de este *ResultSet* para calcular y luego mostrar el número de trabajadores y el sueldo medio.

Accede al código de tu programa y añade el siguiente procedimiento:

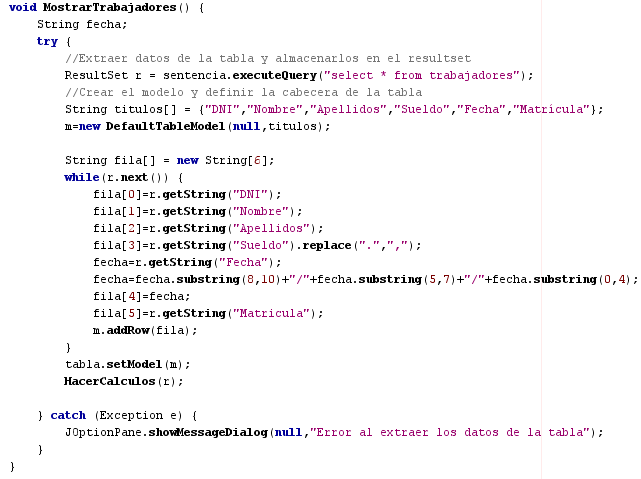


Este código básicamente toma el *ResultSet* pasado como parámetro y lo recorre aumentando en uno la variable *ntrab* cada vez que se pasa al siguiente trabajador.

Por otro lado se calcula la suma de los sueldos de los trabajadores del *ResultSet*.

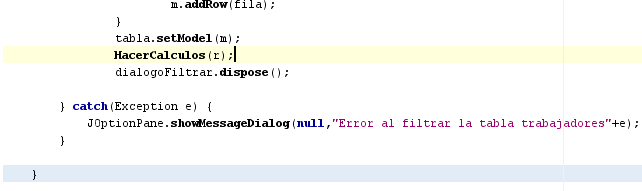
Una vez terminado el bucle solo hay que dividir la suma de los sueldos entre el número de trabajadores para obtener la media, y estos datos se colocan en las etiquetas correspondientes.

1. Interesa que cada vez que se muestren todos los trabajadores en el JTable se realicen estos cálculos, así pues debe añadir la siguiente línea en el método *MostrarTrabajadores*.



Esa línea es una llamada al método que acabamos de programar, y hará que después de que se presenten en el JTable todos los trabajadores se rellenen las etiquetas de los cálculos.

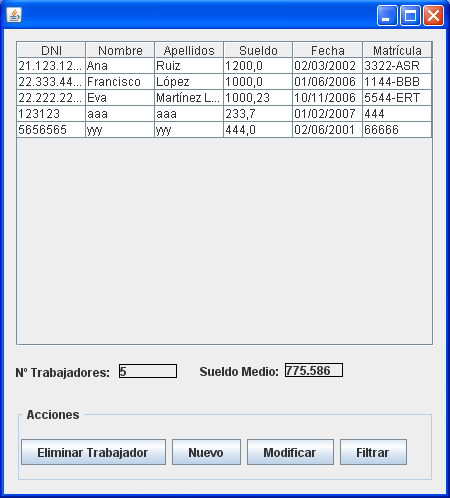
1. También interesa que cuando se realice un filtrado aparezca el número de trabajadores y el sueldo medio correspondiente al listado que se acaba de filtrar. Para ello debes añadir la siguiente línea al código del botón *Aceptar* del cuadro de diálogo de filtrado:



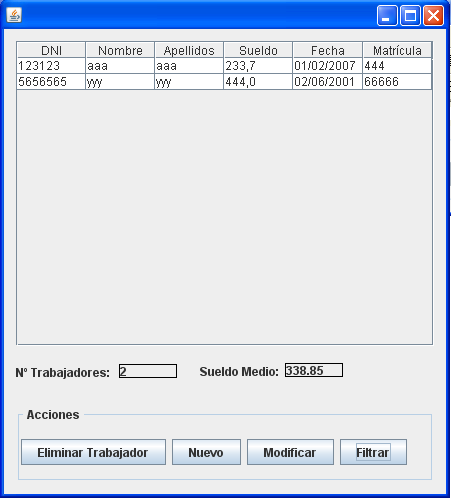
De nuevo se realiza la llamada al método de los cálculos justamente después de colocar en la tabla el listado de trabajadores, en este caso filtrado.

1. Ejecuta el programa y observa como al iniciar el programa ya aparecen en las etiquetas los cálculos (ya que el constructor llama al método *MostrarTrabajadores* que a su vez llama al método *HacerCalculos*)

También puedes comprobar como al hacer un filtrado se muestran los cálculos correspondientes al listado filtrado, ya que en el botón *Aceptar* del dialogo de filtrado se hace una llamada al método *HacerCalculos*.



Al iniciarse el programa aparece el listado completo de trabajadores y se calcula el número de trabajadores y el sueldo medio...



Si se hace un filtrado (en este ejemplo se visualizan los trabajadores con menos de 1000 de sueldo) los cálculos se corresponderán con el listado filtrado...