EJERCICIO GUIADO. JAVA. ACCESO A BASE DE DATOS

Filtrados sobre una tabla

En la hoja guiada anterior se realizó 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 todo momento se mostraba en un JTable el listado de trabajadores de la empresa.

La eliminación y modificación de un trabajador se podía realizar simplemente haciendo clic sobre la fila del trabajador correspondiente en el JTable y luego activando el botón *Eliminar* o *Modificar*.

Sin embargo, en el momento en que tengamos una gran cantidad de trabajadores almacenados en la tabla, puede resultar un poco complicado encontrar al trabajador en concreto que se quiere eliminar o modificar.

Aparte, puede ser necesario a veces ver solo determinados registros de la tabla y no toda la tabla entera.

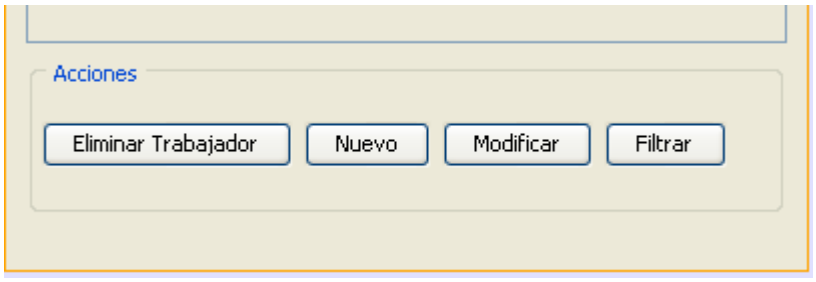
Se hace necesario pues añadir al programa ciertas opciones de filtrado que nos permitan visualizar en el JTable solo aquellos registros que más nos interesen.

En esta hoja guiada se mejorará el programa de la hoja anterior de forma que permita al usuario ciertas opciones de filtrado.

EJERCICIO GUIADO Nº 1

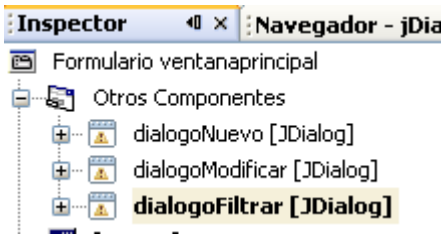
1. Abrir la aplicación de la hoja guiada anterior.

2. Primero debes añadir un botón *btnFiltar* a la ventana:

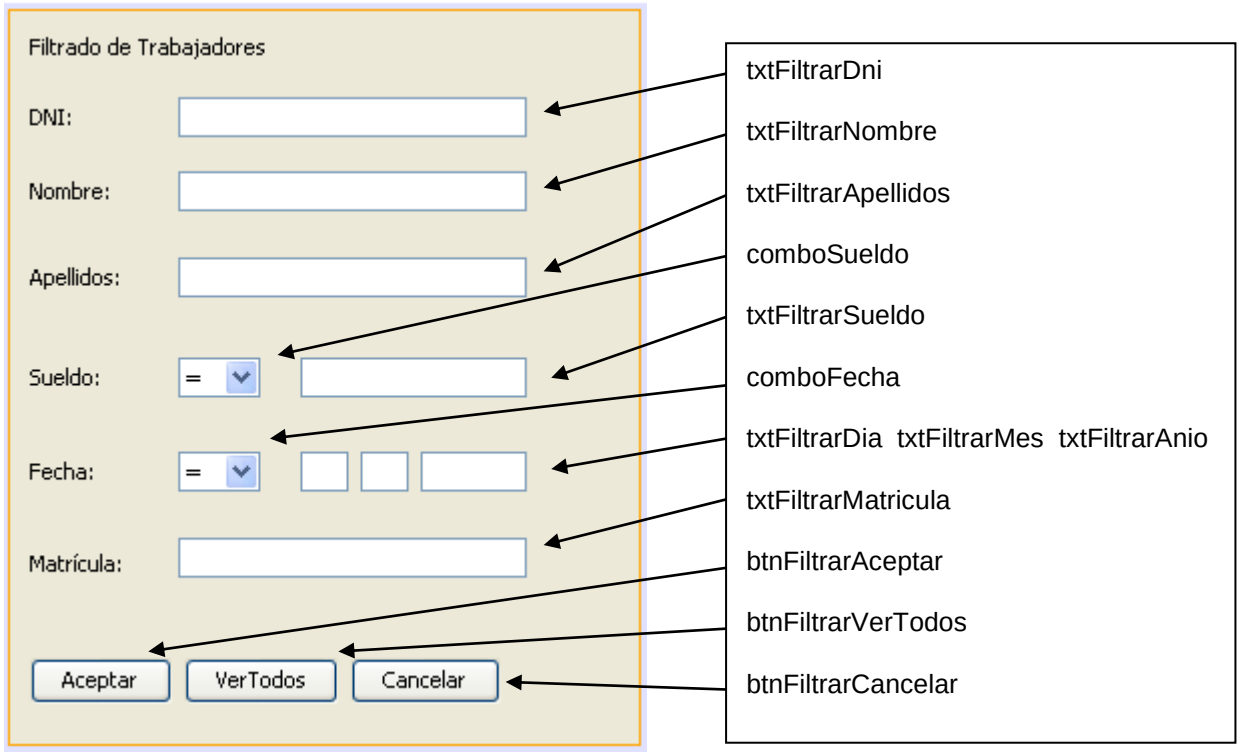


3. Al pulsar este botón aparecerá un cuadro de diálogo que contenga opciones de filtrado, de forma que el usuario pueda decidir qué trabajadores quiere ver.

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



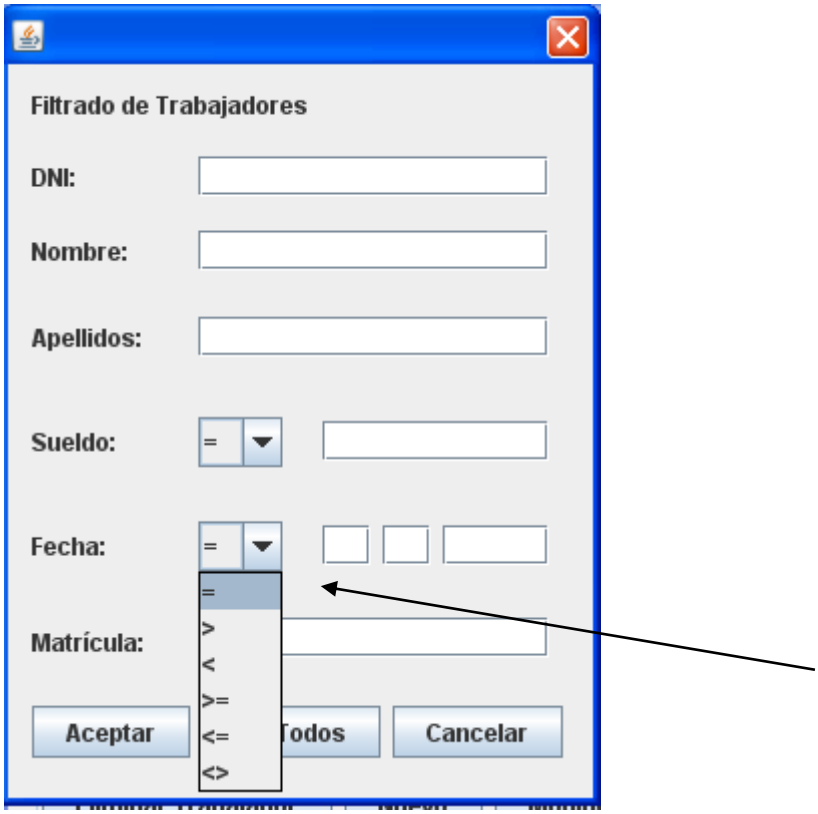
4. Haz doble clic sobre este diálogo y diséñelo de la siguiente forma:



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

= > < >= <= <>

Es decir, cuando se desplieguen aparecerá esto:



5. 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.

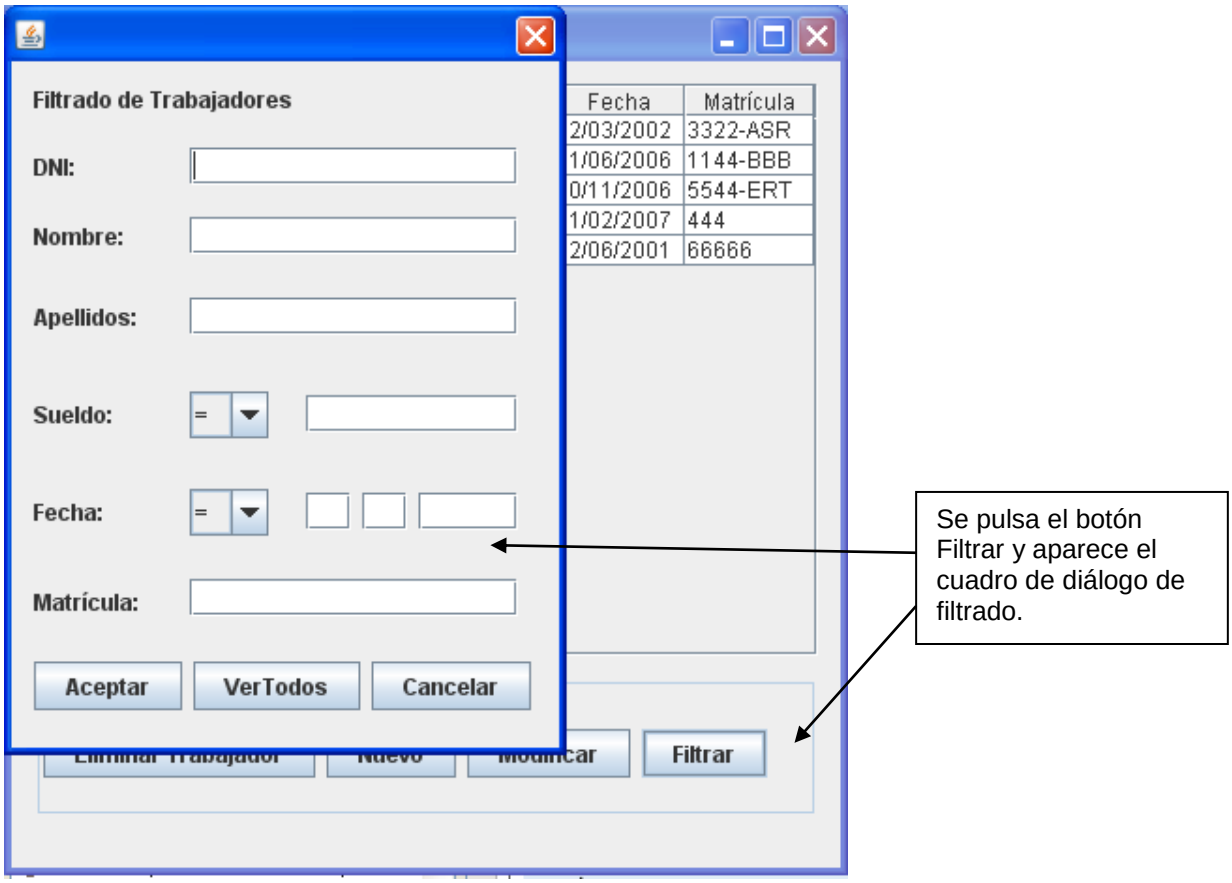
   Si el usuario pulsa el botón *Cancelar*, entonces el cuadro de diálogo de filtrado se cierra sin más.

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

```
    private void btnFiltrarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agrege su codigo aqui:
        dialogoFiltrar.setSize(300,400);
        dialogoFiltrar.setModal(true);
        dialogoFiltrar.setVisible(true);
    }
```

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

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



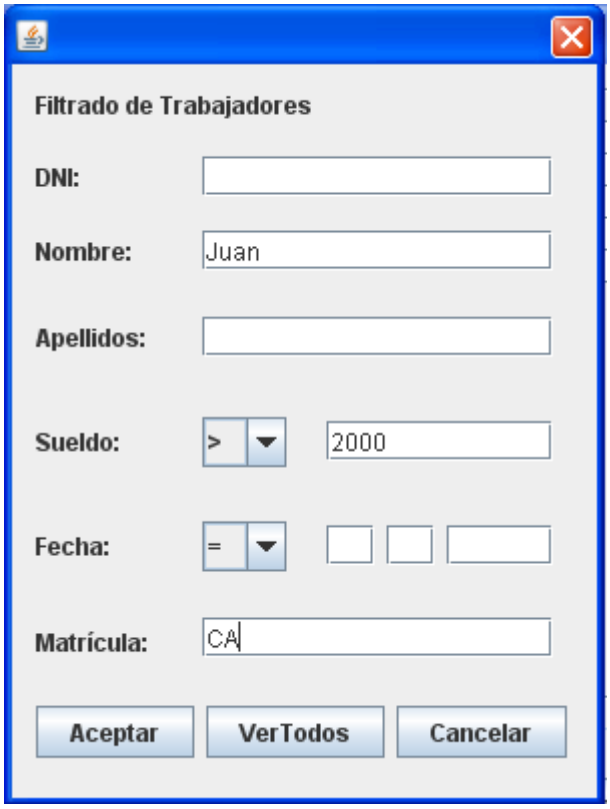
8. Programemos ahora el botón *Cancelar* del cuadro de diálogo de filtrado. La programación de este botón es muy sencilla:

```
    private void btnFiltrarCancelarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agrege su codigo aqui:
        dialogoFiltrar.dispose();
    }
```

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

9. 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 que cumplan todas las siguientes condiciones: **que tengan de nombre *Juan,* que ganen más de 2000 euros de sueldo y que conduzcan un coche cuya matricula 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.

10. 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*:

```
    private void btnFiltrarAceptarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agrege su codigo aqui:
        String dni,nombre,apellidos,sueldo,fecha,mat; //variables de datos
        String cbosueldo, cbofecha; //valores de los combos

        dni=txtFiltrarDni.getText();
        nombre=txtFiltrarNombre.getText();
        apellidos=txtFiltrarApellidos.getText();
        sueldo=txtFiltrarSueldo.getText().replace(",",".");
        fecha=txtFiltrarMes.getText()+"/"+
              txtFiltrarDia.getText()+"/"+
              txtFiltrarAnio.getText();
        mat=txtFiltrarMatricula.getText();
        cbosueldo=(String)comboSueldo.getSelectedItem();
        cbofecha=(String)comboFecha.getSelectedItem();

    }
```

    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.

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

```
    private void btnFiltrarAceptarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agrege su codigo aqui:
        String dni,nombre,apellidos,sueldo,fecha,mat; //variables de datos
        String cbosueldo, cbofecha; //valores de los combos
        String sql; //cadena de la consulta sql
        int ncond=0; //numero de condiciones
```

Agrega estas dos variables en la parte inicial del método.

..... *código anterior*.....

```
        //construcción de la instrucción SELECT de consulta
        sql="select * from trabajadores ";
        if (!dni.equals("")) {
            ncond++;
            sql+=" where dni like '%"+dni+"%' ";
        }
        if (!nombre.equals("")) {
            ncond++;
            if (ncond==1) {
                sql+=" where nombre like '%"+nombre+"%' ";
            } else {
                sql+=" and nombre like '%"+nombre+"%' ";
            }
        }
```

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.

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

```
if (!apellidos.equals("")) {
    ncond++;
    if (ncond==1) {
        sql+=" where apellidos like '%"+apellidos+"%' ";
    } else {
        sql+=" and apellidos like '%"+apellidos+"%' ";
    }
}
```

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”.

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

```
if (!sueldo.equals("")) {
    ncond++;
    if (ncond==1) {
        sql+=" where sueldo"+cbosueldo+sueldo;
    } else {
        sql+=" and sueldo"+cbosueldo+sueldo;
    }
}
```

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: `=, >, <, >=, <= , <>`

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

```
if (!fecha.equals("//")) {
    ncond++;
    if (ncond==1) {
        sql+=" where fecha"+cbofecha+"#"+fecha+"# ";
    } else {
        sql+=" and fecha"+cbofecha+"#"+fecha+"# ";
    }
}
```

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.

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

```
if (!mat.equals("")) {
    ncond++;
    if (ncond==1) {
        sql+=" where matricula like '%"+mat+"%' ";
    } else {
        sql+=" and matricula like '%"+mat+"%' ";
    }
}
```

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.

16. 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:

```
//Ejecutamos la consulta creada
ResultSet r = sentencia.executeQuery(sql);
```

    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)

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

```
String titulos[] = {"DNI","Nombre","Apellidos","Sueldo","Fecha","Matricula"};
m=new DefaultTableModel(null,titulos);

String fila[] = new String[6];
while(r.next()) {
        fila[0]=r.getString("DNI");
        fila[1]=r.getString("Nombre");
        fila[2]=r.getString("Apellidos");
        fila[3]=r.getString("Sueldo").replace(".",",");
        fecha=r.getString("Fecha");
        fecha=fecha.substring(8,10)+"/"+
              fecha.substring(5,7)+"/"+
              fecha.substring(0,4);
        fila[4]=fecha;
        fila[5]=r.getString("Matricula");
        m.addRow(fila);
}
tabla.setModel(m);
```

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.

18. 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:

```
dialogoFiltrar.dispose();
```

19. 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:

```
    private void btnFiltrarAceptarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agrege su codigo aqui:
        String dni,nombre,apellidos,sueldo,fecha,mat; //variables de datos
        String cbosueldo, cbofecha; //valores de los combos
        String sql; //cadena de la consulta sql
        int ncond=0; //numero de condiciones

        try {
```

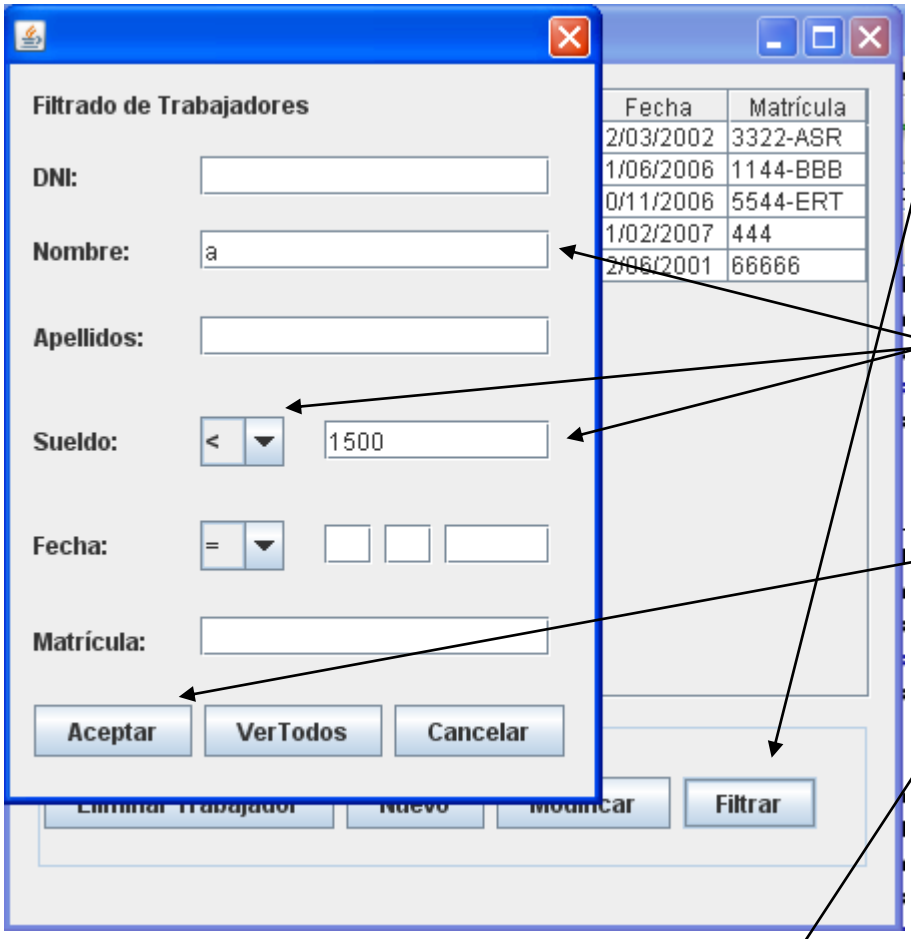
.... resto del código...

```
        } catch(Exception e) {
            JOptionPane.showMessageDialog(null,"Error al filtrar la tabla trabajadores");
        }

    }
```

20. 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.

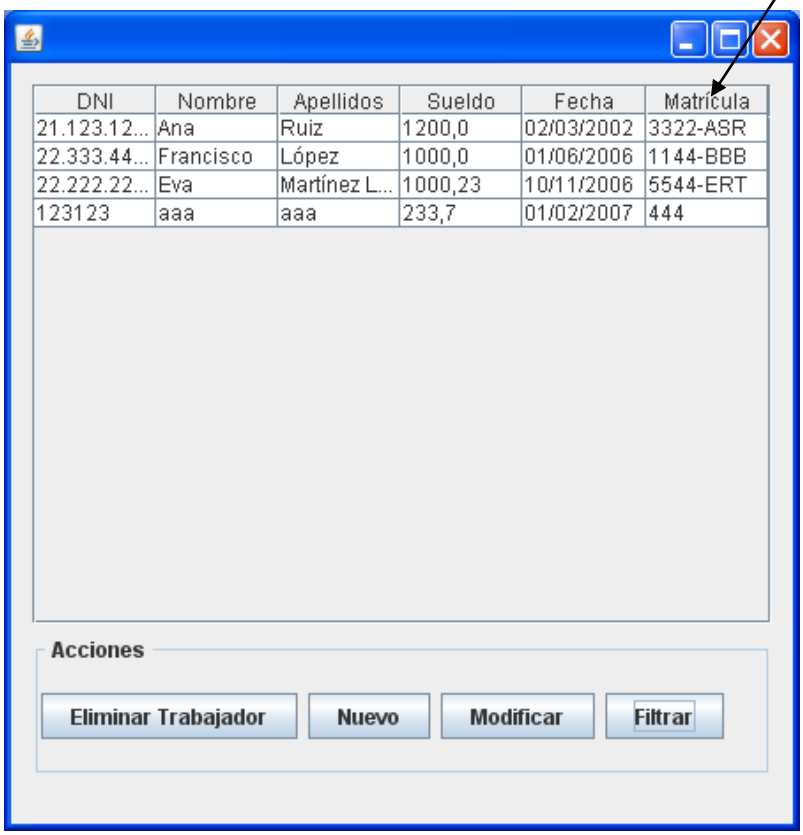
    Ejecuta el programa y haz lo siguiente:



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.



21. 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.

22. 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:

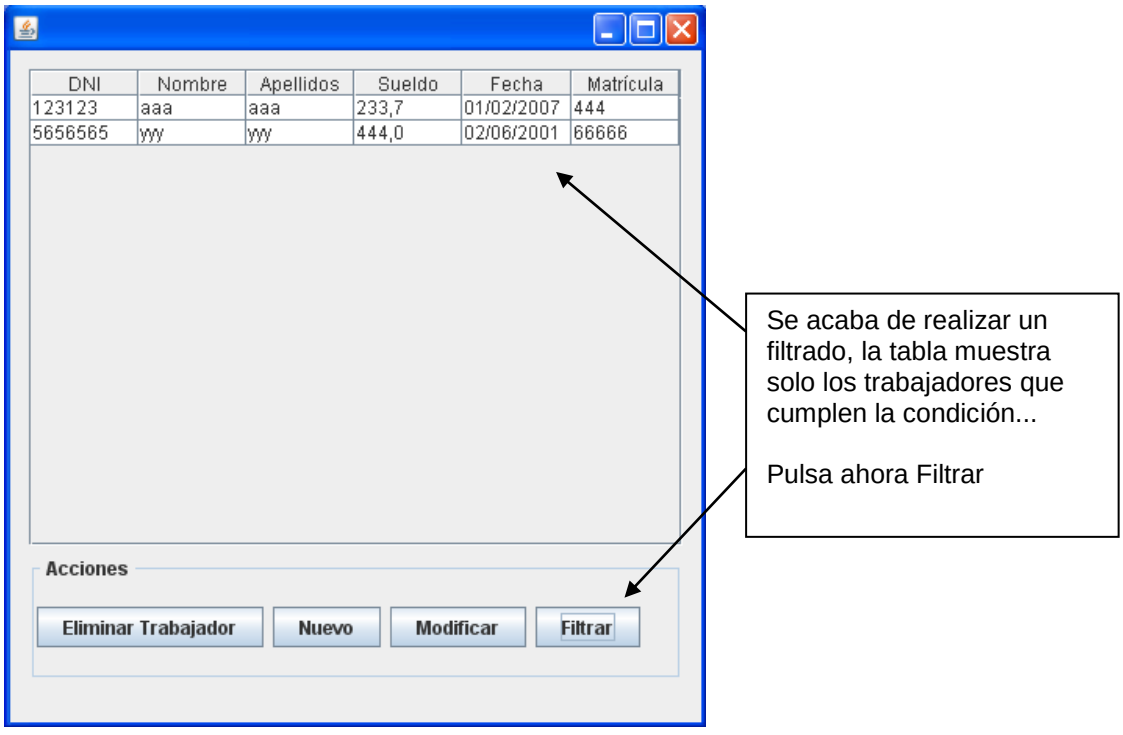
```
    private void btnFiltrarVerTodosActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agrege su codigo aqui:
        MostrarTrabajadores();
        dialogoFiltrar.dispose();
    }
```

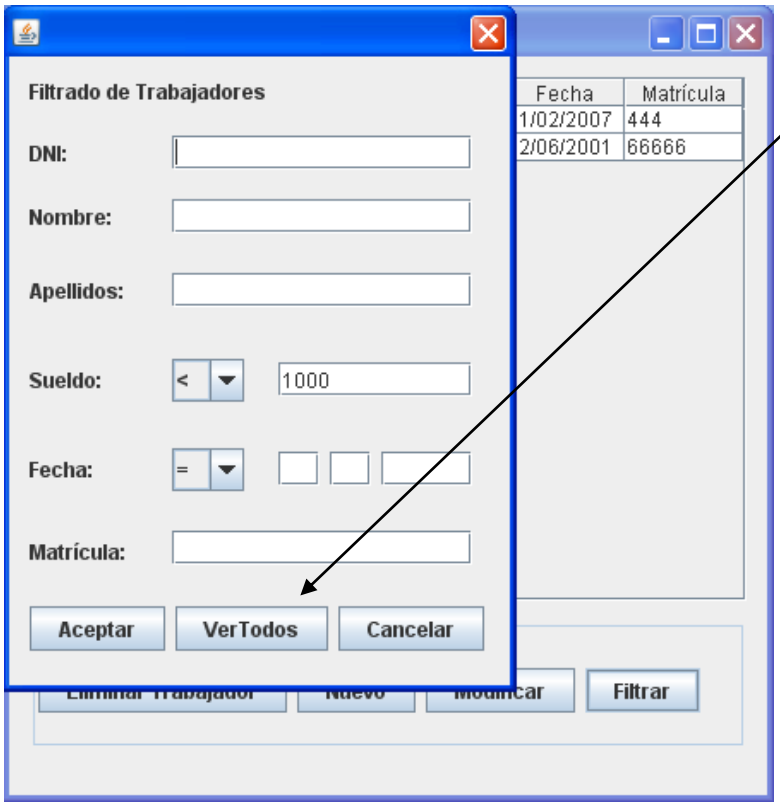
    Como ves aprovechamos el método ya programado *MostrarTrabajadores* que se encarga de mostrar todos los trabajadores en el JTable. Luego solo tenemos que descargar el cuadro de diálogo de filtrado.

23. Ejecuta el programa y prueba lo siguiente:

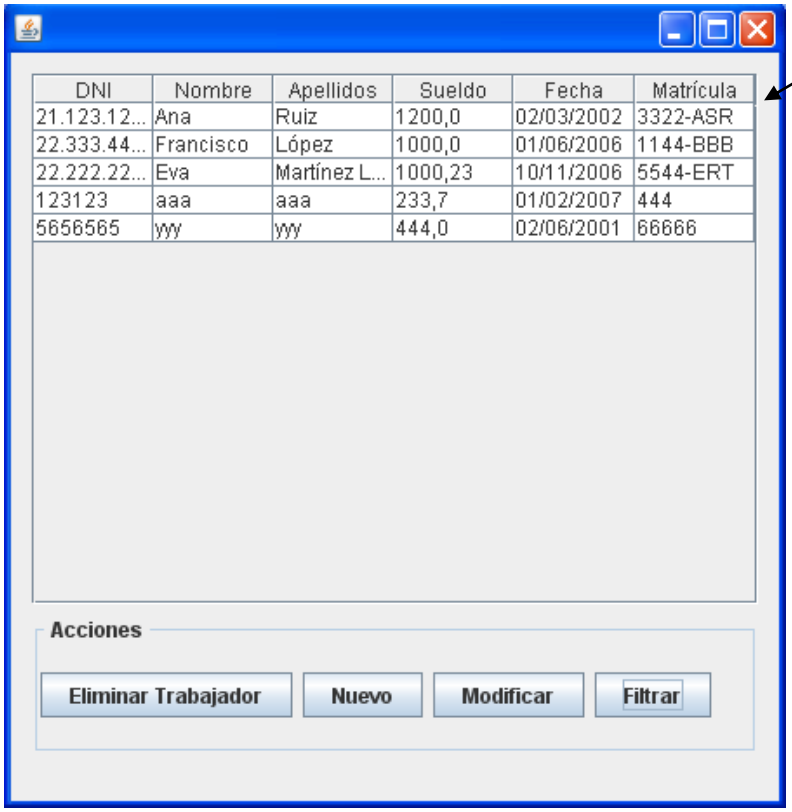
- Haz el filtrado que quieras sobre la tabla trabajadores. Acepta y observa el filtrado en la tabla.
- Vuelve a filtrar pero esta vez pulsa el botón *Ver Todos*. Observarás como vuelven a verse todos los trabajadores en la tabla.

    Ejemplo:





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.