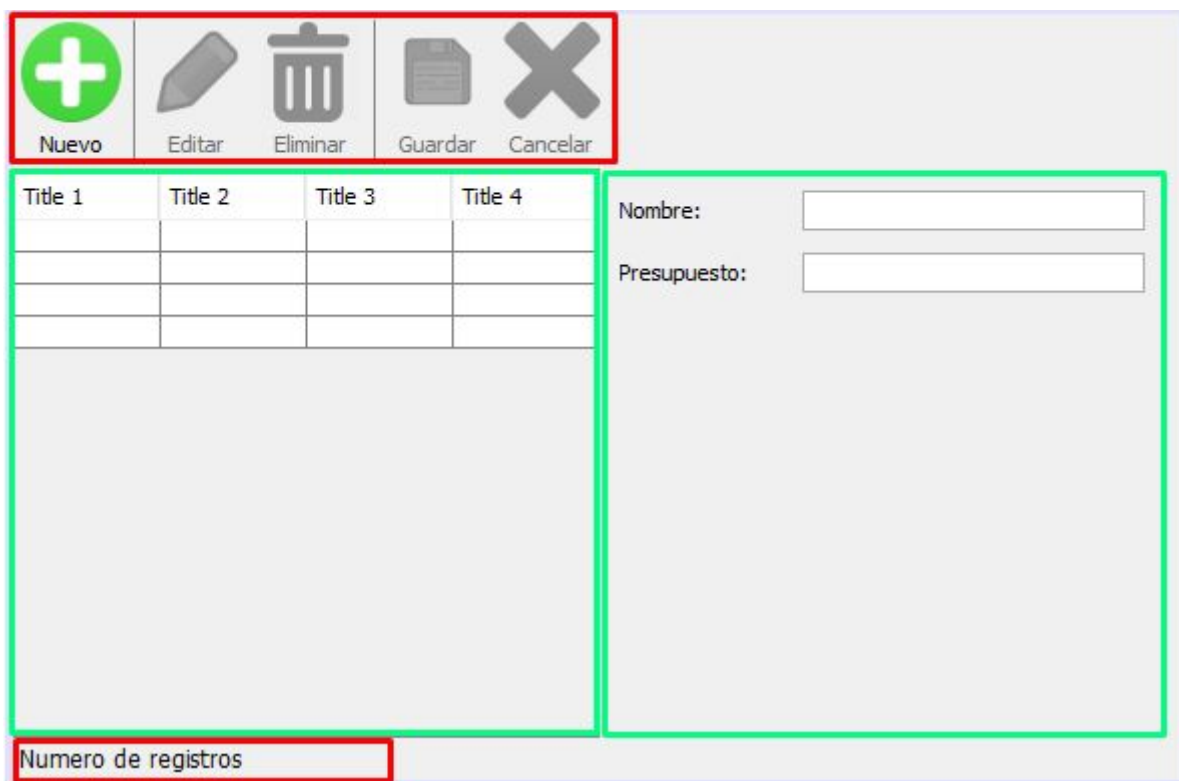


TUTORIAL JAVA SWING

Una vez tengamos terminada la parte del Modelo de nuestro proyecto, objetos, DAOs y código SQL, toca hacer la parte de la Vista. En este tutorial aprenderemos como crear un frame en JAVA Swing para nuestro CRUD.

1. Elementos del frame

En este tutorial vamos a crear un frame para cada entidad de la base de datos, por lo tanto tenemos que diferenciar entre elementos repetidos (rojos) y elementos únicos (verdes).



Title 1	Title 2	Title 3	Title 4

Nombre:

Presupuesto:

Numero de registros

- **Elementos repetidos**

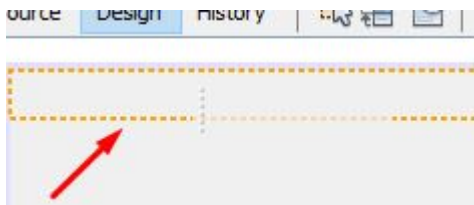
Son aquellos que una vez los tengamos hechos para el primer frame podremos hacer copia y pega con ellos para el siguiente frame. En este caso tenemos los botones y el número de registros.

- **Elementos únicos**

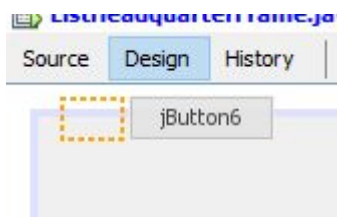
Son elementos que no puedes copiar de un frame a otro porque tienen diferente estructura. Cada frame tendrá su tabla y su panel de datos dependiendo de la entidad que contenga, el caso de la imagen anterior son Sedes. En este tutorial haremos una tabla y un panel de datos independientes del frame. En el caso del panel de datos sólo tendremos que introducirlo en el frame y el modelo de la tabla se actualizará al que le pasemos.

2. Crear el frame

- Hacemos click derecho en el paquete Vista.frames.headquarter > New > JFrame Form.
- En el JFrame hacemos click derecho > Set Layout > Border Layout.
- En Propiedades del JFrame cambiamos la propiedad **defaultCloseOperation** de EXIT_ON_CLOSE a DISPOSE y le ponemos un título.
- Desde el menú Palette (Ctrl+Mayús+8) añadimos un elemento Tool Bar y comprobando que se nos quede pegado al borde de arriba.

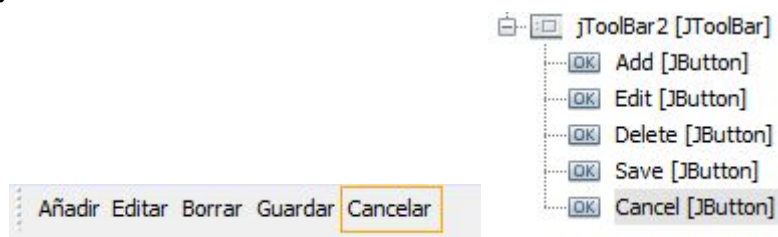


- A continuación añadimos el numero de botones que queramos al Tool Bar. Recordar que el Toolbar está en el borde, por lo tanto para añadirlo tenemos que colocar el botón en el borde superior.



Si este punto se vuelve muy complicado, añadimos el botón al JFrame y desde el menú **Navigador** (Ctrl+7) metemos el botón en el Tool Bar.

- Una vez tengamos los botones deseados, en propiedades editamos el texto del interior y su nombre.



- Para añadir imágenes a los botones, en propiedades > icon pulsamos el botón con los tres puntos a la derecha, seleccionamos el paquete que contiene las imágenes y seleccionamos la imagen que queramos.
- Otras medidas estéticas que le podemos añadir pueden ser el colocar separadores entre los botones, quitar el borderPainted, añadir un margen, etc.



- Cuando hayamos acabado con los botones añadimos:
 - Un panel en el borde derecho.
 - Un label en el borde de abajo.
 - Una tabla en el centro.
- Una vez colocados les ponemos un nombre a cada elemento.

3. Crear el modelo de la tabla

Para crear el modelo de la tabla, lo haremos en una clase java aparte. Hacemos click derecho en el paquete Vista.frames.headquarter > New > Java class y la llamaremos HeadquarterTableModel.

Copiamos el siguiente código:

```
import Modelo.DAO.HeadquarterDAO;
import Modelo.Headquarter;
import Modelo.auxiliary.DAOException;
import java.util.ArrayList;
import java.util.List;
import javax.swing.table.AbstractTableModel;

public class HeadquarterTableModel extends AbstractTableModel{

    private HeadquarterDAO headquarters;

    private List<Headquarter> datos = new ArrayList<>();

    public HeadquarterTableModel(HeadquarterDAO headquarters){
        this.headquarters = headquarters;
    }

    public void updateModel() throws DAOException {
        datos = headquarters.getAllHeadquarter();
    }

    //Titulo de las columnas
```

```

@Override
public String getColumnName(int column) {
    String n = "";
    switch (column){
        case 0: n = "ID"; break;
        case 1: n = "Nombre"; break;
        case 2: n = "Presupuesto"; break;
        case 3: n = "Número de Complejos"; break;
        default: n = "[no]";
    }
    return n;
}

//El numero de registros es el numero de filas
@Override
public int getRowCount() {
    return datos.size();
}

//El numero de columnas lo metemos a fuego
@Override
public int getColumnCount() {
    return 4;
}

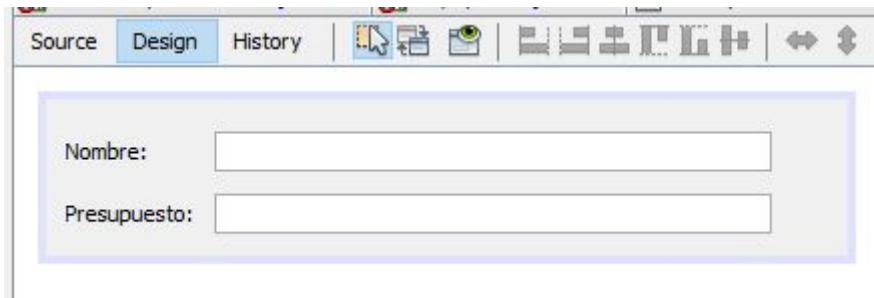
//El valor de cada casilla
@Override
public Object getValueAt(int rowIndex, int columnIndex) {
    Object o = null;
    Headquarter hq = datos.get(rowIndex);
    switch (columnIndex){
        case 0: o = hq.getId(); break;
        case 1: o = hq.getName(); break;
        case 2: o = hq.getBudget(); break;
        case 3: o = hq.getNumComplexes(); break;
        default: o = "";
    }
    return o;
}
}

```

4. Crear el panel de datos

El panel de datos también será una clase aparte. Hacemos click derecho en el paquete Vista.frames.headquarter > New > JPanel Form y seguimos los siguientes pasos:

- Añadir dos JLabels y dos JTextFields, editamos el texto del interior y los renombramos..
- Aumentar el ancho de los JTextFields como deseemos.
- Redimensionar el JPanel al tamaño justo de nuestro contenido.



- Una vez terminado el diseño vamos a la pestaña **Source** para editar el código. Copiamos el código en el lugar indicado por la siguiente imagen:

```
public class HeadquarterDetailsPanel extends javax.swing.JPanel {  
  
    COPIAR AQUÍ  
  
    /**  
     * This method is called from within the constructor to initialize the form.  
     * WARNING: Do NOT modify this code. The content of this method is always  
     * regenerated by the Form Editor.  
     */  
    @SuppressWarnings("unchecked")  
    Generated Code
```

```
private Headquarter headquarter;
```

```
private boolean edit;
```

```
public HeadquartersDetailsPanel() {  
    initComponents();  
}
```

```
public Headquarter getHeadquarter() {  
    return headquarter;  
}
```

```

public void setHeadquarter(Headquarter headquarter) {
    this.headquarter = headquarter;
}

public boolean isEdit() {
    return edit;
}

//El edit le afecta a todos los campos
public void setEdit(boolean edit) {
    this.edit = edit;
    TextName.setEditable(edit);
    TextBudget.setEditable(edit);
}

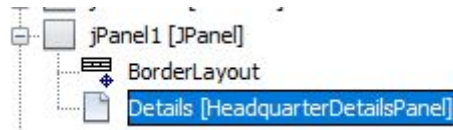
//Método para el botón editar
public void loadData(){
    if (headquarter != null){
        TextName.setText(headquarter.getName());
        TextBudget.setText(Float.toString(headquarter.getBudget()));
    }else{
        TextName.setText("");
        TextBudget.setText("");
    }
    TextName.requestFocus();
}

//Método para el botón guardar
public void saveData(){
    if (headquarter == null) {
        headquarter = new Headquarter();
    }
    headquarter.setName(TextName.getText());
    headquarter.setBudget(Float.parseFloat(TextBudget.getText()));
}

//Método que comprueba si los campos están vacíos
public boolean checkData(){
    boolean noEmpty = false;
    if (!TextBudget.getText().equals("") && !TextName.getText().equals("")) {
        noEmpty = true;
    }
    return noEmpty;
}

```

- El funcionamiento de este panel consiste en que el propio panel tiene el objeto que vamos a insertar, editar o borrar de la base de datos y los jTextField van a ser sus atributos.
- Para acabar volvemos al ListHeadquarterFrame y desde la ventana **Projects** arrastramos el HeadquarterDetailsPanel hasta el jPanel del ListHeadquarterFrame.
- Cuando lo tengamos dentro, le añadimos un BorderLayout al jPanel para que se ajuste y renombramos al HeadquarterDetailsPanel.



- Por último ajustamos el ListHeadquarterFrame al tamaño que deseemos.

5. Código del ListHeadquarterFrame

En ListHeadquarterFrame le damos a la pestaña Source y seguimos los siguientes pasos:

- Borrar el código del main y de los botones si tienen.
- Copiamos el siguiente código después del nombre de la clase:

```
private HeadquarterTableModel model;
```

```
public ListHeadquarterFrame() throws DAOException {
    initComponents();
    this.model = new HeadquarterTableModel(new HeadquarterDAO());
    getData();
    this.tabla.setModel(model); //Introducimos la estructura de la tabla
    this.Details.setEdit(false);
    this.Details.setHeadquarter(null);
    activateButtonsCRUD(false);
    activateButtonsSave(false);
    this.tabla.getSelectionModel().addListSelectionListener(e -> {
        activateButtonsCRUD(tabla.getSelectedRow() != -1);
    });
}
```

```
public final void getData() throws DAOException {
    Registros.setText("Actualizando tabla...");
    model.updateModel();
}
```

```

        model.fireTableDataChanged();
        Registros.setText(model.getRowCount() + " sedes visibles");
    }

```

```

private void activateButtonsCRUD(boolean active) {
    BEdit.setEnabled(active);
    BDelete.setEnabled(active);
}

```

```

private void activateButtonsSave(boolean active) {
    BSave.setEnabled(active);
    BCancel.setEnabled(active);
}

```

```

//Método para obtener el objeto de la fila seleccionada
private Headquarter getSelectedHeadquarter() throws DAOException {
    int id = (int) tabla.getValueAt(tabla.getSelectedRow(), 0);
    return new HeadquarterDAO().getHeadquarter(id);
}

```

- Código del botón añadir:

```

private void BAddActionPerformed(java.awt.event.ActionEvent evt) {
    tabla.clearSelection();
    Details.setHeadquarter(null);
    Details.loadData();
    Details.setEdit(true);
    activateButtonsSave(true);
}

```

- Código del botón editar:

```

private void BEditActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        Headquarter headquarter = getSelectedHeadquarter();
        Details.setHeadquarter(headquarter);
        Details.setEnabled(true);
        Details.setEdit(true);
        Details.loadData();
        activateButtonsSave(true);
    } catch (DAOException ex) {
        ex.getMessage();
    }
}

```

- Código del botón cancelar:


```

private void BCancelActionPerformed(java.awt.event.ActionEvent evt) {
    Details.setHeadquarter(null);
    Details.setEdit(false);
    Details.loadData();
    tabla.clearSelection();
    activateButtonsSave(false);
}

```

- Código del botón guardar:

```

private void BSaveActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if (Details.checkData()) {
            Headquarter headquarter = Details.getHeadquarter();
            //Diferenciar cuando está editando y creando uno nuevo
            if (headquarter.getId() == null) {
                new HeadquarterDAO(headquarter).insertHeadquarter();
            } else {
                new HeadquarterDAO(headquarter).modifyHeadquarter();
            }
            Details.setHeadquarter(null);
            Details.setEdit(false);
            Details.loadData();
            tabla.clearSelection();
            activateButtonsSave(false);
            getData();
        } else {
            JOptionPane.showMessageDialog(rootPane, "Por favor rellene "
                + "todos los campos", "Error datos incompletos",
                JOptionPane.OK_OPTION);
        }
    } catch (DAOException ex) {
        ex.getMessage();
    }
}

```

- Código del botón borrar:

```

private void BDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        Headquarter headquarter = getSelectedHeadquarter();
        //Comprueba si se puede borrar y pide confirmación
        if (headquarter.getNumComplexes() == 0) {
            if (JOptionPane.showConfirmDialog(rootPane, "¿Seguro que quieres borrar "

```

```

        + "esta sede?", "Borrar Sede", JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION) {
    new HeadquarterDAO(headquarter).deleteHeadquarter();
    getData();
    }
    } else {
        JOptionPane.showMessageDialog(rootPane, "No puedes borrar una sede "
        + "hasta borrar todos sus complejos", "Error borrar Sede",
        JOptionPane.OK_OPTION);
    }
    } catch (DAOException ex) {
        ex.getMessage();
    }
    }
}

```

6. Crear el MainMenu

El MainMenu es el va a controlar todas las ventanas, es decir, el CRUD de cada entidad. Su código es el siguiente:

```

private ListHeadquarterFrame headquartersFrame = null;
private ListSportComplexFrame complexesFrame = null;
private ListEquipmentFrame equipmentsFrame = null;
private ListCommissionerFrame commissionerFrame = null;

```

El código de los botones sería algo así (cambiar el nombre de la variable para cada botón):

```

try{
    if (headquartersFrame == null) {
        headquartersFrame = new ListHeadquarterFrame();
    }
    headquartersFrame.getData();
    headquartersFrame.setLocationRelativeTo(this);
    headquartersFrame.setVisible(true);
} catch (DAOException ex){
    ex.getMessage();
}
}

```

7. Modelo de ComboBox

Para este ejemplo nos iremos a SportComplexDetailsPanel y le añadiremos un JComboBox al lado del JLabel que pone "Sede:". Una vez tengamos el comboBox seguimos los siguientes puntos:

- En propiedades del JComboBox, en la propiedad **model** hacemos click en los tres puntos de la derecha y en la ventana que se nos muestra pulsamos la opción de abajo "Reset to Default" para vaciar el comboBox.
- En el menú de propiedades vamos a la pestaña **Code** y en la propiedad **Type Parameters** hacemos click en los tres puntos de la derecha y en la ventana que se nos muestra cambiamos "String" que está dentro de "<>" por <HeadquarterComboView>.
- Redimensionar el JComboBox como deseemos y cambiarle el nombre del elemento a ComboBoxHeadquarter.
- Ahora creamos una nueva clase java dentro del paquete y la llamamos HeadquarterComboView. El código es el siguiente:

```
public class HeadquarterComboView extends Headquarter{

    public HeadquarterComboView(Headquarter headquarter) {
        super(headquarter.getName(), headquarter.getBudget());
        this.id = headquarter.getId();
    }

    @Override
    public String toString() {
        return this.getName()+" (ID: "+this.getId()+")";
    }
}
```

- Después creamos una nueva clase java dentro del paquete y la llamamos HeadquarterComboModel. El código es el siguiente:

```
public class HeadquarterComboModel extends
DefaultComboBoxModel<HeadquarterComboView>{

    private HeadquarterDAO headquarters;
```

```

private List<Headquarter> list;

public HeadquarterComboModel(HeadquarterDAO headquarters) {
    this.headquarters = headquarters;
    this.list = new ArrayList<>();
}

public void update() throws DAOException{
    if (headquarters != null) {
        list = headquarters.getAllHeadquarter();
        removeAllElements();
        for (Headquarter hq : list) {
            addElement( new HeadquarterComboView(hq));
        }
    }
}
}

```

- Con estas 2 últimas clases que hemos creado, los elementos del JComboBox serán independientes del modelo, por lo tanto tenemos un método **toString** único para las Sedes del JComboBox.

- Para acabar volvemos a SportComplexDetailsPanel y en la pestaña de **Source** descomentamos la partes indicadas:

- Le pasamos nuestro modelo creado en nuestra clase java al JComboBox del panel de datos:
`ComboBoxHeadquarter.setModel(comboModel);`
- Habilitamos el JComboBox: `ComboBoxHeadquarter.setEnabled(edit);`

- Método para que al editar el comboBox seleccione el elemento correspondiente solo:

```

public void SelectItemComboBox() {
    boolean encontrado=false;
    for (int i = 0; i < ComboBoxHeadquarter.getItemCount() &&
!encontrado; i++) {
        if (ComboBoxHeadquarter.getItemAt(i).getId() ==
            sportComplex.getHeadquarter().getId()) {
            ComboBoxHeadquarter.setSelectedIndex(i);
            encontrado=true;
        }
    }
}

```

- Introducimos la Sede seleccionada en el objeto Complejo:

```
HeadquarterComboView hcv =  
    (HeadquarterComboView)  
ComboBoxHeadquarter.getSelectedltem();  
Headquarter headquarter = (Headquarter) hcv;  
sportComplex.setHeadquarter(headquarter);
```