Visión general

- Muestra una **lista** (JList), un **combobox** (JComboBox) y una **tabla** (JTable) con datos de **alumnos**.
- Permite **ordenar** por nombre, edad y nota con **botones** y también ordenando por **encabezados de la tabla**.
- Permite **filtrar** por nombre con una caja de texto.
- Cuida el idioma **español** (tildes/ñ) al comparar y ordenar usando **Collator**.
- Mantiene las tres vistas **sincronizadas** (si cambias el orden subyacente, todas se actualizan).

1) Modelo de dominio: Alumno

```
public static class Alumno {
  private final String nombre;
  private final int edad;
  private final double nota;
  @Override public String toString() { return nombre + " (" + edad + " años) - " + nota; }
}
```

Qué hace: define el tipo de dato que vas a mostrar.
Por qué así: separar "datos" (Alumno) de la "vista" (JList/JTable/JComboBox) es la base del patrón **MVC**.
Tip: el toString() ayuda a depurar y, si no pones renderer, es lo que verían JList/JComboBox.

2) AlumnoTableModel: el adaptador entre la lista y la JTable

```
public static class AlumnoTableModel extends AbstractTableModel {
  private final String[] cols = {"Nombre", "Edad", "Nota"};
  private List<Alumno> data;
    @Override public Object getValueAt(int row, int col) { … }
  @Override public Class<?> getColumnClass(int col) { … }
}
```

Qué hace: "traduce" List<Alumno> a filas/columnas que entiende la JTable.

- getRowCount() y getColumnCount() dicen cuántas filas/columnas hay.
- getValueAt(r, c) devuelve el valor específico a mostrar en cada celda.
- **Clave:** getColumnClass() indica el **tipo real** (String/Integer/Double).
  Esto permite que la tabla **ordene bien** los números (si no, ordena como texto y 10 < 2).

setData(…) + fireTableDataChanged() **avisa** a la tabla que los datos cambiaron (se reordenaron, por ejemplo).

3) Renderers para JList y JComboBox

public static class AlumnoListCellRenderer extends DefaultListCellRenderer { … }

public static class AlumnoComboRenderer extends DefaultListCellRenderer { … }

Qué hacen: deciden **cómo se dibuja cada elemento** en la lista/combobox.
Aquí construyes el texto visible:

- En la **JList**: Álvaro — 21 años • Nota: 7.5
- En el **JComboBox**: Álvaro (21)

Por qué así: sin renderer verías toString() o incluso Alumno@hashcode. Con renderer, la UI es más clara.
Regla: el renderer **pinta**; no debe cambiar datos ni crear componentes nuevos cada vez.

4) Utilidades de ordenamiento (comparadores) y Collator

static Collator collatorEs() { … }

static Comparator<Alumno> porNombreES() { … }

static Comparator<Alumno> porEdad() { … }

static Comparator<Alumno> porNota() { … }

Qué hacen: definen **cómo comparar** dos alumnos para saber cuál va "antes".

- Collator en **español** con PRIMARY ignora **tildes y mayúsculas** → "Álvaro" y "Alvaro" se tratan como iguales para ordenar.

- porEdad() y porNota() usan comparadores numéricos correctos.

5) main y creación de la UI en el hilo correcto

```
public static void main(String[] args) {
  SwingUtilities.invokeLater(Main::crearUI);
}
```

Qué hace: asegura que la UI se construya en el **Event Dispatch Thread** (EDT), como recomienda Swing.
Por qué así: evita errores gráficos y comportamientos raros.

6) Datos base

```
List<Alumno> alumnos = new ArrayList<>(List.of(
  new Alumno("Álvaro", 21, 7.5),
));
```

Qué hace: crea una lista inicial para poblar los controles.

7) JList (lista simple) + modelo + renderer

```
DefaultListModel<Alumno> listModel = new DefaultListModel<>();
alumnos.forEach(listModel::addElement);
JList<Alumno> jList = new JList<>(listModel);
jList.setCellRenderer(new AlumnoListCellRenderer());
```

Qué hace:

- DefaultListModel guarda los ítems.
- JList muestra esos ítems.
- El **renderer** decide el texto/estilo de cada fila.

Cuándo usar JList: cuando se quiere mostrar **varios elementos** y permitir selección (1 o varias).

8) JComboBox (lista desplegable) + modelo + renderer

DefaultComboBoxModel<Alumno> comboModel = new DefaultComboBoxModel<>();

alumnos.forEach(comboModel::addElement);

JComboBox<Alumno> combo = new JComboBox<>(comboModel);

combo.setRenderer(new AlumnoComboRenderer());

Qué hace: igual que JList, pero para **selección única** en un desplegable.
Cuándo usar JComboBox: cuando el usuario debe **elegir un elemento** de un conjunto.

9) JTable (tabla) + TableRowSorter (orden por columna) + comparadores

AlumnoTableModel tableModel = new AlumnoTableModel(alumnos);

JTable table = new JTable(tableModel);

table.setAutoCreateRowSorter(true);

TableRowSorter<TableModel> sorter = new TableRowSorter<>(tableModel);

sorter.setComparator(0, (o1, o2) -> collatorEs().compare((String)o1, (String)o2));

sorter.setComparator(1, Comparator.comparingInt(a -> ((Integer)a)));

sorter.setComparator(2, Comparator.comparingDouble(a -> ((Double)a)));

table.setRowSorter(sorter);

Qué hace:

- Crea la tabla a partir del TableModel.
- Activa **orden por encabezados**.
- Crea un **TableRowSorter** y define **comparadores por columna**:
    - Columna 0 (Nombre): usa **Collator** → orden correcto en español.
    - Columna 1 (Edad) y 2 (Nota): orden **numérico**, no alfabético.

Nota: setAutoCreateRowSorter(true) ya crea un sorter; aquí además pones uno propio para controlar comparadores (está bien, el tuyo manda).

10) Filtro de texto (por nombre) con RowFilter

```
JTextField filtroTxt = new JTextField();
filtroTxt.getDocument().addDocumentListener(new DocumentListener() {
  void apply() {
    String q = filtroTxt.getText().trim();
    if (q.isEmpty()) {
      sorter.setRowFilter(null);
    } else {
      Collator es = collatorEs();
      sorter.setRowFilter(new RowFilter<TableModel, Integer>() {
        public boolean include(Entry<? extends TableModel, ? extends Integer> entry) {
          String nombre = entry.getStringValue(0);
          // Aprox. "empieza con" usando Collator + fallback contains
          return es.compare(
                nombre.substring(0, Math.min(nombre.length(), q.length())),
                q
              ) == 0
              || nombre.toLowerCase().contains(q.toLowerCase());
        }
      });
    }
  }
  public void insertUpdate(DocumentEvent e) { apply(); }
  public void removeUpdate(DocumentEvent e) { apply(); }
  public void changedUpdate(DocumentEvent e) { apply(); }
});
```

Qué hace: cada vez que escribes/borra en el textbox, se **recalcula el filtro**:

- Si el campo está vacío → **sin filtro** (muestra todo).
- Si hay texto → crea un RowFilter que **incluye** solo filas cuyo **nombre** coincide (prefijo con Collator) o **contiene** el texto (insensible a mayúsculas).

Cómo reutilizar: cambia la columna y la condición. Por ejemplo, filtrar por **nota ≥ 8** o por **rango de edades**.

11) Botones de ordenamiento "manual" sobre la lista base

```
sortNombre.addActionListener(e -> {
  alumnos.sort(porNombreES());
  refrescar(...);
});
sortEdad.addActionListener(e -> {
  alumnos.sort(porEdad().reversed());
  refrescar(...);
});
sortNota.addActionListener(e -> {
  alumnos.sort(porNota().reversed().thenComparing(porNombreES()));
  refrescar(...);
});
```

Qué hacen: cambian el **orden de la lista original** con tus Comparator.
Por qué refrescar: porque JList/JComboBox no se enteran solos de que cambiaste la lista **fuera** de sus modelos.

12) refrescar(...): sincronizar las tres vistas

```
private static void refrescar(DefaultListModel<Alumno> listModel,
                              DefaultComboBoxModel<Alumno> comboModel,
```

```
                    AlumnoTableModel tableModel,
                    List<Alumno> alumnos) {
    listModel.clear();
    alumnos.forEach(listModel::addElement);

    comboModel.removeAllElements();
    alumnos.forEach(comboModel::addElement);

    tableModel.setData(alumnos);
}
```

Qué hace: repuebla **JList**, **JComboBox** y notifica a **JTable** que los datos cambiaron.
Por qué así: cada control tiene su **modelo**; al cambiar la lista subyacente, hay que **actualizar** esos modelos para que lo reflejen.

13) Layout y ventanas

- JFrame con tamaño, título y cierre.
- JSplitPane para dividir la vista: izquierda (JList+JComboBox) y derecha (JTable).
- Bordes y paneles para una UI más clara.

Ejercicio Completo:

```
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.*;
import java.text.Collator;
import java.util.*;
import java.util.List;
import java.util.stream.Collectors;

public class Main {

  // ====== Modelo de dominio ======
  public static class Alumno {
    private final String nombre;
    private final int edad;
    private final double nota;

    public Alumno(String nombre, int edad, double nota) {
      this.nombre = nombre;
      this.edad = edad;
      this.nota = nota;
    }
    public String getNombre() { return nombre; }
    public int getEdad() { return edad; }
    public double getNota() { return nota; }
```

```
    @Override public String toString() {
        // Útil para depuración (ComboBox/JList pueden usar toString si no pones renderer)
        return nombre + " (" + edad + " años) - " + nota;
    }
}

// ====== TableModel para JTable ======
public static class AlumnoTableModel extends AbstractTableModel {
    private final String[] cols = {"Nombre", "Edad", "Nota"};
    private List<Alumno> data;

    public AlumnoTableModel(List<Alumno> data) { this.data = data; }

    public void setData(List<Alumno> nueva) {
        this.data = nueva;
        fireTableDataChanged();
    }

    @Override public int getRowCount() { return data.size(); }
    @Override public int getColumnCount() { return cols.length; }
    @Override public String getColumnName(int c) { return cols[c]; }

    @Override public Object getValueAt(int row, int col) {
        Alumno a = data.get(row);
        return switch (col) {
```

```
            case 0 -> a.getNombre();
            case 1 -> a.getEdad();
            case 2 -> a.getNota();
            default -> null;
        };
    }

    @Override public Class<?> getColumnClass(int col) {
        return switch (col) {
            case 0 -> String.class;
            case 1 -> Integer.class;
            case 2 -> Double.class;
            default -> Object.class;
        };
    }
  }

  // ====== Renderers para JList y JComboBox ======
  public static class AlumnoListCellRenderer extends DefaultListCellRenderer {
    @Override
    public Component getListCellRendererComponent(JList<?> list, Object value, int
index,
                                                  boolean isSelected, boolean cellHasFocus) {
        super.getListCellRendererComponent(list, value, index, isSelected, cellHasFocus);
        if (value instanceof Alumno a) {
            setText(a.getNombre() + " — " + a.getEdad() + " años  •  Nota: " + a.getNota());
```

```
        }
        return this;
      }
    }
    public static class AlumnoComboRenderer extends DefaultListCellRenderer {
      @Override
      public Component getListCellRendererComponent(JList<?> list, Object value, int
index,
                                                    boolean isSelected, boolean cellHasFocus) {
        super.getListCellRendererComponent(list, value, index, isSelected, cellHasFocus);
        if (value instanceof Alumno a) {
          setText(a.getNombre() + " (" + a.getEdad() + ")");
        }
        return this;
      }
    }

    // ====== Utilidades de ordenamiento ======
    static Collator collatorEs() {
      Collator es = Collator.getInstance(new Locale("es", "ES"));
      es.setStrength(Collator.PRIMARY); // ignora tildes/case
      return es;
    }

    static Comparator<Alumno> porNombreES() {
      Collator es = collatorEs();
```

```
        return (a, b) -> es.compare(a.getNombre(), b.getNombre());
    }
    static Comparator<Alumno> porEdad() { return
Comparator.comparingInt(Alumno::getEdad); }
    static Comparator<Alumno> porNota() { return
Comparator.comparingDouble(Alumno::getNota); }

    // ====== App ======
    public static void main(String[] args) {
        SwingUtilities.invokeLater(Main::crearUI);
    }

    private static void crearUI() {
        JFrame f = new JFrame("Listas de objetos + ordenamiento (Swing)");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(900, 600);
        f.setLocationRelativeTo(null);

        // Datos base
        List<Alumno> alumnos = new ArrayList<>(List.of(
            new Alumno("Álvaro", 21, 7.5),
            new Alumno("Beatriz", 19, 8.7),
            new Alumno("Camila", 22, 6.9),
            new Alumno("Diego", 20, 9.1),
            new Alumno("Esteban", 23, 5.8),
            new Alumno("Íñigo", 21, 7.9),
            new Alumno("José", 20, 8.1),
```

```
    new Alumno("María", 19, 9.4)
));

// ====== JList ======
DefaultListModel<Alumno> listModel = new DefaultListModel<>();
alumnos.forEach(listModel::addElement);
JList<Alumno> jList = new JList<>(listModel);
jList.setCellRenderer(new AlumnoListCellRenderer());
JScrollPane listScroll = new JScrollPane(jList);
listScroll.setBorder(BorderFactory.createTitledBorder("JList"));

// ====== JComboBox ======
DefaultComboBoxModel<Alumno> comboModel = new DefaultComboBoxModel<>();
alumnos.forEach(comboModel::addElement);
JComboBox<Alumno> combo = new JComboBox<>(comboModel);
combo.setRenderer(new AlumnoComboRenderer());
JPanel comboPanel = new JPanel(new BorderLayout());
comboPanel.setBorder(BorderFactory.createTitledBorder("JComboBox"));
comboPanel.add(combo, BorderLayout.NORTH);

// ====== JTable + sorter/filters ======
AlumnoTableModel tableModel = new AlumnoTableModel(alumnos);
JTable table = new JTable(tableModel);
table.setAutoCreateRowSorter(true);
TableRowSorter<TableModel> sorter = new TableRowSorter<>(tableModel);
// Comparadores por columna (0 nombre, 1 edad, 2 nota)
```

```
    sorter.setComparator(0, (o1, o2) -> collatorEs().compare((String)o1, (String)o2));
    sorter.setComparator(1, Comparator.comparingInt(a -> ((Integer)a)));
    sorter.setComparator(2, Comparator.comparingDouble(a -> ((Double)a)));
    table.setRowSorter(sorter);

    // Filtrado por texto
    JTextField filtroTxt = new JTextField();
    filtroTxt.setToolTipText("Filtrar por nombre (ignora tildes/mayúsculas)");
    filtroTxt.getDocument().addDocumentListener(new
javax.swing.event.DocumentListener() {
      void apply() {
        String q = filtroTxt.getText().trim();
        if (q.isEmpty()) {
          sorter.setRowFilter(null);
        } else {
          Collator es = collatorEs();
          sorter.setRowFilter(new RowFilter<TableModel, Integer>() {
            @Override
            public boolean include(Entry<? extends TableModel, ? extends Integer> entry) {
              String nombre = entry.getStringValue(0);
              // "Contiene" aproximado usando Collator: simplificamos comparando por
prefijo
              // Alternativa: normalizar (NFD) y quitar diacríticos manualmente.
              return es.compare(
                  nombre.substring(0, Math.min(nombre.length(), q.length())),
                  q
              ) == 0 || nombre.toLowerCase().contains(q.toLowerCase());
```

```
            }
        });
      }
    }
    public void insertUpdate(javax.swing.event.DocumentEvent e) { apply(); }
    public void removeUpdate(javax.swing.event.DocumentEvent e) { apply(); }
    public void changedUpdate(javax.swing.event.DocumentEvent e) { apply(); }
  });

  JPanel filterPanel = new JPanel(new BorderLayout());
  filterPanel.add(new JLabel("Filtro: "), BorderLayout.WEST);
  filterPanel.add(filtroTxt, BorderLayout.CENTER);

  JScrollPane tableScroll = new JScrollPane(table);
  JPanel tablePanel = new JPanel(new BorderLayout());
  tablePanel.setBorder(BorderFactory.createTitledBorder("JTable (ordenar clickeando encabezados)"));
  tablePanel.add(filterPanel, BorderLayout.NORTH);
  tablePanel.add(tableScroll, BorderLayout.CENTER);

  // ====== Botones de ordenamiento (sobre la lista subyacente) ======
  JButton sortNombre = new JButton("Ordenar por Nombre (ES)");
  JButton sortEdad = new JButton("Ordenar por Edad desc");
  JButton sortNota = new JButton("Ordenar por Nota desc, luego Nombre");

  sortNombre.addActionListener(e -> {
```

```
    alumnos.sort(porNombreES());
    refrescar(listModel, comboModel, tableModel, alumnos);
});
sortEdad.addActionListener(e -> {
    alumnos.sort(porEdad().reversed());
    refrescar(listModel, comboModel, tableModel, alumnos);
});
sortNota.addActionListener(e -> {
    alumnos.sort(porNota().reversed().thenComparing(porNombreES()));
    refrescar(listModel, comboModel, tableModel, alumnos);
});

JPanel sortPanel = new JPanel(new GridLayout(1, 0, 8, 8));
sortPanel.add(sortNombre);
sortPanel.add(sortEdad);
sortPanel.add(sortNota);

// ====== Layout general ======
JSplitPane left = new JSplitPane(JSplitPane.VERTICAL_SPLIT, listScroll, comboPanel);
left.setResizeWeight(0.7);

JSplitPane mainSplit = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, left, tablePanel);
mainSplit.setResizeWeight(0.35);

JPanel root = new JPanel(new BorderLayout(10, 10));
root.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
```

```
        root.add(mainSplit, BorderLayout.CENTER);
        root.add(sortPanel, BorderLayout.SOUTH);

        f.setContentPane(root);
        f.setVisible(true);
    }

    private static void refrescar(DefaultListModel<Alumno> listModel,
                                  DefaultComboBoxModel<Alumno> comboModel,
                                  AlumnoTableModel tableModel,
                                  List<Alumno> alumnos) {
        // Refrescar JList
        listModel.clear();
        alumnos.forEach(listModel::addElement);
        // Refrescar JComboBox
        comboModel.removeAllElements();
        alumnos.forEach(comboModel::addElement);
        // Refrescar JTable
        tableModel.setData(alumnos);
    }
}
```