

## JTable con Ejemplos-Parte I

- [Introduccion](#)
- [Un Ejemplo Sencillo](#)
  - [Ejemplo 1.SimpleTable1.java](#)
- [El Modelo de Tabla](#)
  - [DefaultTableModel](#)
    - [Ejemplo 2.SimpleTable2.java](#)
    - [Ejemplo 3.SimpleTable3.java](#)
  - [AbstractTableModel](#)
    - [Ejemplo 4.SimpleTable4.java](#)
- [API JTable](#)
  - [Campos](#)
  - [Constructores](#)
  - [Métodos](#)
- [API DefaultTableModel](#)
  - [Campos](#)
  - [Constructores](#)
  - [Métodos](#)
- [API AbstractTableModel](#)
  - [Campos](#)
  - [Constructores](#)
  - [Métodos](#)
- [Comentarios Finales](#)
- [Referencias](#)

### Introduccion

Con la llegada de Swing, como parte de la JFC(*Java Foundation Classes*), la construcción de Interfaces Gráficas de Usuario(GUI) recibió un excelente conjunto de componentes (aproximadamente 40) que la mejoraron: desde el siempre utilizado JButton, hasta el flexible JEditorPane o el JDesktopPane, pasando por los JTree y los JTable; sin dejar de mencionar a los JFileChooser y los JDialog, todos ellos, y los componentes restantes, permiten la creación de aplicaciones con interfaces gráficas más intuitivas y completas.

Swing es el resultado de la unión de esfuerzos entre [Netscape](#), con su *Internet Foundation Classes*, y [SUN](#). Swing es sólo una parte de la JFC, muchos cometemos el error de creer que Swing y JFC son lo mismo. La JFC contiene también otros elementos, estos son algunos de ellos:

- Cortar y Pegar
- Elementos de Accesibilidad
- Java 2D
- Impresión

De todos los componentes que forman Swing, quizá los JTree y las JTable, sean los componentes con APIs más extensas (la clase JTable tiene más de 100 [métodos](#)), y quizá también los más complejos.

<http://www.javahispano.org>

Afortunadamente esa complejidad les permite ser también de los componentes Swing más personalizables y potentes; al igual que en la mayoría de las otras clases Swing no es necesario conocer todos sus métodos para comenzar a utilizarlos y darles una utilidad práctica.

Como programadores, sabemos muy bien que la presentación de datos tabulados es una de las tareas más comunes que se presentan al momento de crear interfaces gráficas; desde la simple tabla que permite únicamente mostrar el resultado de una consulta, hasta las que permiten editar directamente el contenido de cada celda, ordenar las columnas, personalizar su apariencia, etc. Todas las tareas antes descritas, y muchas otras, son posibles de realizar utilizando la clase `JTable`; por supuesto, mientras más complejo sea el requerimiento a cubrir, se requerirá en igual medida utilizar más métodos o recursos de la clase.

Este primer artículo:

- Muestra como crear una `JTable` sencilla para la visualización de datos.
- Explica que es un *modelo de tabla*
- Muestra como crear una `JTable` con `DefaultTableModel` como *modelo de tabla*
- Muestra como crear una `JTable` con `AbstractTableModel` como *modelo de tabla*
- Describe la API `JTable`
- Describe la API `DefaultTableModel`
- Describe la API `AbstractTableModel`

## Un Ejemplo Sencillo

El título de este artículo es: `JTable` con ejemplos; así que comenzaremos creando una tabla sencilla. Esta tabla únicamente mostrará un conjunto de datos definidos desde su constructor, para ello primero daremos una vistazo rápido a los [constructores](#) que proporciona esta clase; puedes ver al final del artículo con detalle más información.

- `JTable()`
- `JTable(int numRows, int numColumns)`
- `JTable(Object[][] rowData, Object[] columnNames)`
- `JTable(TableModel dm)`
- `JTable(TableModel dm, TableColumnModel cm)`
- `JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)`
- `JTable(Vector rowData, Vector columnNames)`

Para este primer ejemplo utilizaremos el 3er. [constructor](#) de la lista anterior. El cual nos permite construir una tabla a partir de dos parámetros; el primero de ellos: `rowData` es un array bidimensional de objetos que representa el contenido de la tabla, y el segundo: `columnNames` representa los nombres de cada columna, contenidos también en un array que por lo general es un array de `String`'s.

Nuestro primer ejemplo tendrá las siguientes columnas:

```
String[] columnNames =
```

```
        {"Nombre", "Apellido", "Pasatiempo", "Años de Practica",
"Soltero(a)"};
```

Y utilizaremos el siguiente array para su contenido:

```
Object[][] data = {
    {"Mary", "Campione", "Esquiar", new Integer(5), new Boolean(false)},
    {"Lhucas", "Huml", "Patinar", new Integer(3), new Boolean(true)},
    {"Kathya", "Walrath", "Escalar", new Integer(2), new Boolean(false)},
    {"Marcus", "Andrews", "Correr", new Integer(7), new Boolean(true)},
    {"Angela", "Lalth", "Nadar", new Integer(4), new Boolean(false)}
};
```

El [constructor](#), entonces queda así:

```
JTable table = new JTable(data, columnNames);
```

El código completo de nuestro primer ejemplo es el siguiente:

### Ejemplo 1. SimpleTable1.java

```
import javax.swing.JTable;
import javax.swing.JScrollPane;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;

public class SimpleTable1 extends JFrame {
    public SimpleTable1() {
        super("Ejemplo 1");

        //array bidimensional de objetos con los datos de la tabla
        Object[][] data = {
            {"Mary", "Campione", "Esquiar", new Integer(5), new Boolean(false)},
            {"Lhucas", "Huml", "Patinar", new Integer(3), new Boolean(true)},
            {"Kathya", "Walrath", "Escalar", new Integer(2), new Boolean(false)},
            {"Marcus", "Andrews", "Correr", new Integer(7), new Boolean(true)},
            {"Angela", "Lalth", "Nadar", new Integer(4), new Boolean(false)}
        };

        //array de String's con los títulos de las columnas
        String[] columnNames = {"Nombre", "Apellido", "Pasatiempo",
            "Años de Practica", "Soltero(a)"};

        //se crea la Tabla
        final JTable table = new JTable(data, columnNames);
        table.setPreferredScrollableViewportSize(new Dimension(500, 70));

        //Creamos un JScrollPane y le agregamos la JTable
        JScrollPane scrollPane = new JScrollPane(table);

        //Agregamos el JScrollPane al contenedor
        getContentPane().add(scrollPane, BorderLayout.CENTER);


        //manejamos la salida
```

<http://www.javahispano.org>

```
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

public static void main(String[] args) {
    SimpleTable1 frame = new SimpleTable1();
    frame.pack();
    frame.setVisible(true);
}
}
```

Y la siguiente imagen muestra el resultado de la ejecución del código anterior:



Nombre	Apellido	Pasatiempo	Años de Practica	Soltero(a)
Mary	Campione	Esquiar	5	false
Lhucas	Huml	Patinar	3	true
Kathya	Walrath	Escalar	2	false
Marcus	Andrews	Correr	7	true

**Figura 1.** SimpleTable1 en ejecución

En lugar de arrays pudimos haber utilizado *vectores* y utilizado el ultimo de los [constructores](#) de la lista mostrada anteriormente.

Aunque la manerá más sencilla de construir tablas es utilizando cualquiera de los siguientes constructores:

- `JTable(Object[][] rowData, Object[] columnNames)`
- `JTable(Vector rowData, Vector columnNames)`

Su utilización presenta unas desventajas que debemos tener en cuenta:

- La primera, y más notoria de ellas, es que para la construcción de la tabla se tienen que tener de antemano los datos que queremos que contenga la tabla ya sea en un array o en un *vector*, lo que le resta flexibilidad al llenado de la tabla; ya que quizás en un momento dado, sería mucho más práctico y rápido, colocar directamente los datos en la tabla y no tener que colocarlos antes en un array o *vector*
- La segunda desventaja es que, como te darás cuenta al probar el ejemplo, estos constructores hacen automáticamente que todas las celdas sean editables.
- Y la tercera, y menos notoria a primera vista, es que todos los datos contenidos en la tabla, son tratados como de un mismo tipo de datos. A pesar de que hemos declarado columnas como `Boolean` o `Integer`, todas nuestras celdas muestran su contenido como `String's`

Estas tres desventajas pueden ser eliminadas si agregamos un *modelo de tabla* a nuestra aplicación.

## El Modelo de Tabla

Los *modelos de tabla* son objetos que implementan la *interface* `TableModel`; a través de ellos es posible personalizar mucho más y mejor el comportamiento de los componentes `JTable`, permitiendo utilizar al máximo sus potencialidades.

Todas las *tablas* cuentan con un *modelo de tabla*, aunque en el ejemplo 1 no se haya especificado, existe uno por omisión

El siguiente gráfico intenta mostrar como cada componente `JTable` obtiene siempre sus datos desde un *modelo de tabla*.



**Figura 2.** Relación Modelo -> Vista

La clase `AbstractTableModel` es la que implementa directamente a la *interface* `TableModel`, aunque es esta clase la que se recomienda *extender* para utilizarla como *modelo de tabla*, existe un *modelo de tabla* predeterminado que facilita mucho el trabajo con tablas. Este modelo predeterminado es la clase `DefaultTableModel`

### DefaultTableModel

Esta clase tiene el siguiente diagrama de herencia:

```
java.lang.Object
|
+- javax.swing.table.AbstractTableModel
|
+- javax.swing.table.DefaultTableModel
```

Nuevamente, antes de comenzar a utilizar esta clase, veremos cuales son los [constructores](#) con que cuenta:

- `DefaultTableModel()`
- `DefaultTableModel(int numRows, int numColumns)`
- `DefaultTableModel(Object[][] data, Object[] columnNames)`
- `DefaultTableModel(Object[] columnNames, int numRows)`
- `DefaultTableModel(Vector columnNames, int numRows)`
- `DefaultTableModel(Vector data, Vector columnNames)`

Utilizaremos el constructor que nos permite crear un `DefaultTableModel`, a partir de los datos

<http://www.javahispano.org>

con que ya contamos del ejemplo anterior:

```
Object[][] data = {
    {"Mary", "Campione",
     "Esquiar", new Integer(5), new Boolean(false)},
    {"Lhucas", "Huml",
     "Patinar", new Integer(3), new Boolean(true)},
    {"Kathya", "Walrath",
     "Escalar", new Integer(2), new Boolean(false)},
    {"Marcus", "Andrews",
     "Correr", new Integer(7), new Boolean(true)},
    {"Angela", "Lalth",
     "Nadar", new Integer(4), new Boolean(false)}
};

String[] columnNames = {"Nombre",
                        "Apellido",
                        "Pasatiempo",
                        "Años de Practica",
                        "Soltero(a)"};
```

Por lo tanto, el constructor queda así:

```
DefaultTableModel dtm= new DefaultTableModel(data, columnNames);
```

Después de haber creado el *modelo de tabla*, dtm en el ejemplo, se crea la tabla con el [constructor](#) correspondiente:

```
final JTable table = new JTable(dtm);
```

Una vez hecho esto, cualquier modificación que se realice sobre el *modelo de tabla* se reflejará directamente en la tabla. Así, podemos [agregar una columna](#):

```
String[] newColumn= {"Flan",
                    "Pastel",
                    "Helado",
                    "Barquillo",
                    "Manzana" };
dtm.addColumn("Postre",newColumn);
```

una [fila](#):

```
Object[] newRow={"Pepe", "Grillo",
                "Tenis", new Integer(5), new Boolean(false), "Pera"};
dtm.addRow(newRow);
```

o [modificar una celda](#) en especial, en este ejemplo la celda ubicada en la columna 1, fila 1:

```
dtm.setValueAt("Catherine", 1, 1);
```

Puedes revisar los [métodos](#) que proporciona la clase DefaultTableModel para conocer que otras cosas puedes realizar con ella.

A continuación se presenta el listado completo del código que muestra el uso del *modelo de tabla*

DefaultTableModel :

### Ejemplo 2.SimpleTable2.java

```
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import javax.swing.JScrollPane;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;

public class SimpleTable2 extends JFrame {
    public SimpleTable2() {
        super("Ejemplo 2");

        //array bidimensional de objetos con los datos de la tabla
        Object[][] data = {
            {"Mary", "Campione",
             "Esquiar", new Integer(5), new Boolean(false)},
            {"Lhucas", "Huml",
             "Patinar", new Integer(3), new Boolean(true)},
            {"Kathya", "Walrath",
             "Escalar", new Integer(2), new Boolean(false)},
            {"Marcus", "Andrews",
             "Correr", new Integer(7), new Boolean(true)},
            {"Angela", "Lalth",
             "Nadar", new Integer(4), new Boolean(false)}
        };

        //array de String's con los titulos de las columnas
        String[] columnNames = {"Nombre",
                                "Apellido",
                                "Pasatiempo",
                                "Años de Practica",
                                "Soltero(a)"};

        //creamos el Modelo de la tabla con los datos anteriores
        DefaultTableModel dtm= new DefaultTableModel(data, columnNames);
        //se crea la Tabla con el modelo DefaultTableModel
        final JTable table = new JTable(dtm);

        // una vez creada la tabla con su modelo
        // podemos agregar columnas
        String[] newColumn= {"Flan",
                             "Pastel",
                             "Helado",
                             "Barquillo",
                             "Manzana" };
        dtm.addColumn("Postre",newColumn);
        //filas
        Object[] newRow={"Pepe", "Grillo",
                         "Tenis", new Integer(5), new Boolean(false), "Pera"};

        dtm.addRow(newRow);

        //o modificar una celda en especifico
        dtm.setValueAt("Catherine", 1, 1);
        //se define el tamaño
        table.setPreferredScrollableViewportSize(new Dimension(500, 70));
    }
}
```

```
//Creamos un JScrollPane y le agregamos la JTable
JScrollPane scrollPane = new JScrollPane(table);

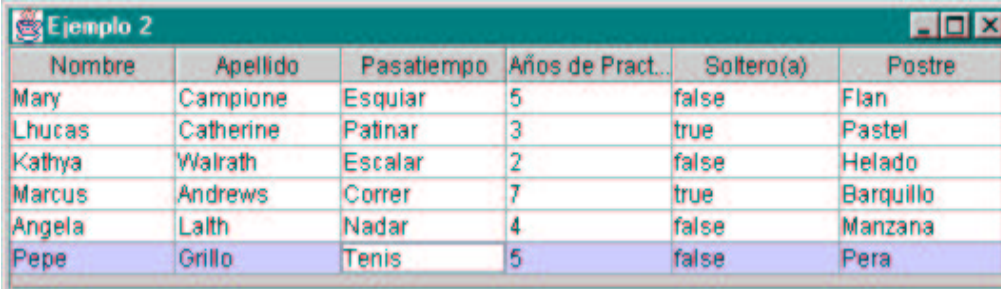
//Agregamos el JScrollPane al contenedor
getContentPane().add(scrollPane, BorderLayout.CENTER);

//manejamos la salida
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});

}

public static void main(String[] args) {
    SimpleTable2 frame = new SimpleTable2();
    frame.pack();
    frame.setVisible(true);
}
}
```

El resultado de ejecutar el ejemplo anterior será:



Nombre	Apellido	Pasatiempo	Años de Pract...	Soltero(a)	Postre
Mary	Campione	Esquiar	5	false	Flan
Lhucas	Catherine	Patinar	3	true	Pastel
Kathya	Walrath	Escalar	2	false	Helado
Marcus	Andrews	Correr	7	true	Barquillo
Angela	Lalth	Nadar	4	false	Manzana
Pepe	Grillo	Tenis	5	false	Pera

*Figura 3. SimpleTable2 en ejecución*

Aquí tenemos otro ejemplo con DefaultTableModel:

### **Ejemplo 3.SimpleTable3.java**

```
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import javax.swing.JScrollPane;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;

public class SimpleTable3 extends JFrame {
    public SimpleTable3() {
        super("Ejemplo 3");

        //creamos el arreglo de objetos que contendra el
        //contenido de las columnas
        Object[] data = new Object[5];

        // creamos el modelo de Tabla
```



```

DefaultTableModel dtm= new DefaultTableModel();

// se crea la Tabla con el modelo DefaultTableModel
final JTable table = new JTable(dtm);

// insertamos las columnas
for(int column = 0; column < 5; column++){
    dtm.addColumn("Columna " + column);
}

// insertamos el contenido de las columnas
for(int row = 0; row < 10; row++) {
    for(int column = 0; column < 5; column++) {
        data[column] = "Celda " + row + ", " + column;
    }
    dtm.addRow(data);
}

//se define el tamaño
table.setPreferredScrollableViewportSize(new Dimension(500, 70));

//Creamos un JScrollPane y le agregamos la JTable
JScrollPane scrollPane = new JScrollPane(table);

//Agregamos el JScrollPane al contenedor
getContentPane().add(scrollPane, BorderLayout.CENTER);

//manejamos la salida
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

public static void main(String[] args) {
    SimpleTable3 frame = new SimpleTable3();
    frame.pack();
    frame.setVisible(true);
}
}

```

En este segundo ejemplo del uso de la clase `DefaultTableModel`, creamos primeramente un array de objetos llamado `data` que podrá contener 5 elementos.

```
Object[] data = new Object[5];
```

Despues creamos un *modelo de tabla* llamado `dtm`:

```
DefaultTableModel dtm= new DefaultTableModel();
```

Antes de cualquier operacion sobre el *modelo de tabla*, debemos crear la `tabla` que lo utilizará:

```
final JTable table = new JTable(dtm);
```

Ahora, al *modelo de tabla* le agregamos 5 columnas:

```
for(int column = 0; column < 5; column++){
```

http://www.javahispano.org

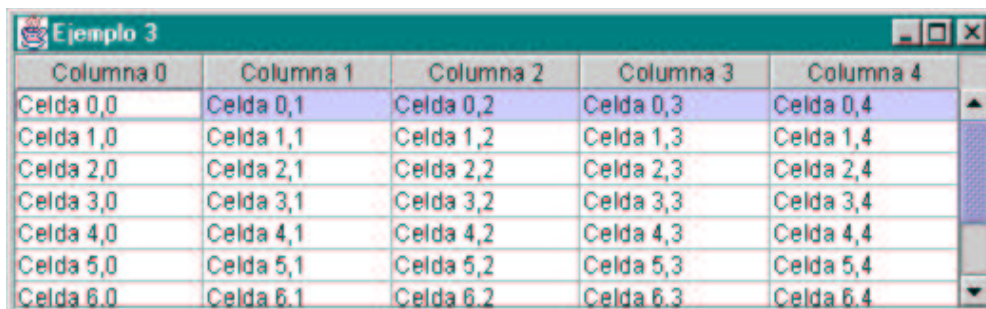
```
        dtm.addColumn("Columna " + column);  
    }
```

Y por ultimo, insertamos el contenido de las celdas, columna por columna:

```
for(int row = 0; row < 10; row++) {  
    for(int column = 0; column < 5; column++) {  
        data[column] = "Celda " + row + "," + column;  
    }  
    dtm.addRow(data);  
}
```

En el fragmento de código anterior, en lugar de hacer una sencilla asignación de valores consecutivos a cada celda dentro del ciclo `for`, podrías estar leyendo y asignando directamente el contenido de: un archivo de texto, una consulta a una base de datos, o alguna otra fuente; ya que, como hemos visto, también es posible agregar filas completas a la `tabla`.

La siguiente figura muestra el resultado de ejecutar el Ejemplo 3.



Columna 0	Columna 1	Columna 2	Columna 3	Columna 4
Celda 0,0	Celda 0,1	Celda 0,2	Celda 0,3	Celda 0,4
Celda 1,0	Celda 1,1	Celda 1,2	Celda 1,3	Celda 1,4
Celda 2,0	Celda 2,1	Celda 2,2	Celda 2,3	Celda 2,4
Celda 3,0	Celda 3,1	Celda 3,2	Celda 3,3	Celda 3,4
Celda 4,0	Celda 4,1	Celda 4,2	Celda 4,3	Celda 4,4
Celda 5,0	Celda 5,1	Celda 5,2	Celda 5,3	Celda 5,4
Celda 6,0	Celda 6,1	Celda 6,2	Celda 6,3	Celda 6,4

**Figura 4.** SimpleTable3 en ejecución

Como te podrás dar cuenta, al ejecutar los ejemplos 2 y 3, una de las desventajas de no manejar de manera directa un *modelo de tabla* a sido superada; ya es posible agregar directamente valores a las celdas de la `tabla`. Sin embargo, las celdas siguen siendo editables, y sus valores siguen siendo tratados aún como `String`'s.

Cuando utilices el *modelo de tabla* `DefaultTableModel` debes tener en cuenta que: éste utiliza un *vector de vectores* para almacenar los valores de las celdas de la `tabla`, tiene un desempeño inferior al de un *modelo de tabla* personalizado, debido a que sus métodos *están sincronizados* y que además en la [documentación oficial](#), se menciona que la *serialización* de objetos que realiza esta clase no será compatible con entregas futuras de `Swing`

Así, aunque la utilización del *modelo de tabla* `DefaultTableModel` es aún funcional y proporciona facilidades para la utilización de un *modelo de tabla*, es mucho más recomendable, por cuestiones de desempeño y personalización, utilizar la clase `AbstractTableModel`

#### **AbstractTableModel**

Con esta clase es posible implementar, de una manera más completa y eficiente, los métodos necesarios para crear un *modelo de tabla*.

Para crear un *modelo de tabla* personalizado, lo primero que necesitamos es *extender* la clase `AbstractTableModel`.

```
class MyTableModel extends AbstractTableModel {
    .....
}
```

Después, debemos de implementar los 3 métodos siguientes:

```
class MyTableModel extends AbstractTableModel {
    public int getRowCount() {
        ...
    }

    public int getColumnCount() {
        ...
    }

    public Object getValueAt(int row, int column) {
        ...
    }
}
```

Con la implementación de los métodos anteriores, las celdas de la tabla NO serán editables y NO se podrán modificar los valores de cada una de ellas.

Si deseamos tener un mecanismo para modificar los valores de las celdas de la tabla, tenemos que sobrescribir el [método](#) `setValueAt` de la clase `AbstractTableModel`:

```
class MyTableModel extends AbstractTableModel {
    public int getRowCount() {
        ...
    }

    public int getColumnCount() {
        ...
    }

    public Object getValueAt(int row, int column) {
        ...
    }

    public void setValueAt(Object value, int row, int col) {
        ...
    }
}
```

Y, si la modificación de los valores de las celdas, se hace directamente sobre ellas, necesitamos indicar a nuestro *modelo de tabla* que las celdas de la tabla serán editables, esto se hace sobrescribiendo el [método](#) `isCellEditable`:

```
class MyTableModel extends AbstractTableModel {
    public int getRowCount() {
        ...
    }
}
```

<http://www.javahispano.org>

```
public int getColumnCount() {
    ...
}

public Object getValueAt(int row, int column) {
    ...
}

public void setValueAt(Object value, int row, int col) {
    ...
}

public boolean isCellEditable(int row, int col) {
    ...
}
}
```

[Más adelante](#) puedes revisar los métodos restantes que proporciona la clase [AbstractTableModel](#)

Ya lo único que haría falta sería agregar los nombres de las columnas de nuestra tabla y definir su contenido inicial:

```
class MyTableModel extends AbstractTableModel {

    final String[] columnNames = {
        ...
    }

    final Object[][] data = {
        ...
    }

    public int getRowCount() {
        ...
    }

    public int getColumnCount() {
        ...
    }

    public Object getValueAt(int row, int column) {
        ...
    }

    public void setValueAt(Object value, int row, int col) {
        ...
    }

    public boolean isCellEditable(int row, int col) {
        ...
    }
}
```

JTable invoca un [método](#) del *modelo de tabla* para determinar el editor/*renderer* predeterminado que utilizará para mostrar el valor de cada celda. Por ejemplo para celdas con valores *booleanos* utilizará *checkbox's*; este método es: `getColumnClass`, y también es recomendable implementarlo:

```

class MyTableModel extends AbstractTableModel {

    final String[] columnNames = {
        ...
    }

    final Object[][] data = {
        ...
    }

    public int getRowCount(){
        ...
    }

    public int getColumnCount(){
        ...
    }

    public Object getValueAt(int row, int column){
        ...
    }

    public void setValueAt(Object value, int row, int col){
        ...
    }

    public boolean isCellEditable(int row, int col) {
        ...
    }

    public Class getColumnClass(int c) {
        ...
    }
}

```

Ahora, después de saber cuáles son los métodos que se necesitan implementar y sobrescribir de la clase `AbstractTableModel` para crear nuestro propio *modelo de tabla*; veremos un ejemplo que nos muestra ya completa la definición de la clase:

```

class MyTableModel extends AbstractTableModel {
    final String[] columnNames = {"Nombre",
        "Apellido",
        "Pasatiempo",
        "Años de Practica",
        "Soltero(a)"};

    final Object[][] data = {
        {"Mary", "Campione",
            "Esquiar", new Integer(5), new Boolean(false)},
        {"Lhucas", "Huml",
            "Patinar", new Integer(3), new Boolean(true)},
        {"Kathya", "Walrath",
            "Escalar", new Integer(2), new Boolean(false)},
        {"Marcus", "Andrews",
            "Correr", new Integer(7), new Boolean(true)},
        {"Angela", "Lalth",
            "Nadar", new Integer(4), new Boolean(false)}
    };

    //únicamente retornamos el numero de elementos del
    //array de los nombres de las columnas
    public int getColumnCount() {

```

```
        return columnNames.length;
    }

    //retornamos el numero de elementos
    //del array de datos
    public int getRowCount() {
        return data.length;
    }

    //retornamos el elemento indicado
    public String getColumnName(int col) {
        return columnNames[col];
    }

    //y lo mismo para las celdas
    public Object getValueAt(int row, int col) {
        return data[row][col];
    }

    /*
     * Este metodo sirve para determinar el editor predeterminado
     * para cada columna de celdas
     */
    public Class getColumnClass(int c) {
        return getValueAt(0, c).getClass();
    }

    /*
     * No tienes que implementar este método a menos que
     * las celdas de tu tabla sean Editables
     */
    public boolean isCellEditable(int row, int col) {
        return true;
    }

    /*
     * No tienes que implementar este método a menos que
     * los datos de tu tabla cambien
     */
    public void setValueAt(Object value, int row, int col) {
        data[row][col] = value;
        fireTableCellUpdated(row, col);
    }
}
```

Si estas utilizando una versión del JDK anterior a la 1.3, debes hacer las siguientes modificaciones al método `setValueAt`, ya que antes de esta version era necesario crear manualmente un `Integer` a partir del valor recibido, ya que de otra manera, el valor recibido se seguiría convirtiendo a un `String`, a partir de la Ver. 1.3, la conversion a `Integer` es automática.

```
//Version del metodo setValuAt para JDK's anteriores a la Ver.1.3
public void setValueAt(Object value, int row, int col) {

    if (data[0][col] instanceof Integer
        && !(value instanceof Integer)) {
        try {
            data[row][col] = new Integer(value.toString());
            fireTableCellUpdated(row, col);
        } catch (NumberFormatException e) {
```

```

        JOptionPane.showMessageDialog(SimpleTable4.this,
            "The \"" + getColumnName(col)
            + "\" column accepts only integer values.");
    }
} else {
    data[row][col] = value;
    fireTableCellUpdated(row, col);
}
}
}

```

**Nota:** Observa se hace una llamada al método `fireTableCellUptaded(row,col)` dentro del método `setValueAt` exacto asignado un nuevo valor a la celda especificada; esto es debido a que, como estamos implementando directamente los métodos `AbstractTableModel`, debemos notificar explícitamente que una celda determinada a sido modificada para que se actualice la tabla.

Puedes revisar cuales son los otros métodos [fireXXX](#) que proporciona la clase `AbstractTableModel` para realizar las notificaciones que indican modificación sobre la tabla.

Una vez creada nuestra clase podemos instanciarla, y utilizarla para [construir](#) una tabla, así:

```

MyTableModel myModel = new MyTableModel();
JTable table = new JTable(myModel);

```

El siguiente código, muestra la utilización de la clase `MyTableModel`; en este ejemplo, ésta clase aparece en la aplicación principal como una clase secundaria:

#### Ejemplo 4.SimpleTable4.java

```

import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.JOptionPane;
import java.awt.*;
import java.awt.event.*;

public class SimpleTable4 extends JFrame {

    public SimpleTable4() {
        super("SimpleTable4");

        MyTableModel myModel = new MyTableModel();
        JTable table = new JTable(myModel);
        table.setPreferredScrollableViewportSize(new Dimension(500, 70));

        //Creamos un contenedor para la Tabla
        JScrollPane scrollPane = new JScrollPane(table);

        //Agregamos nuestra tabla al contenedor
        getContentPane().add(scrollPane, BorderLayout.CENTER);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}

class MyTableModel extends AbstractTableModel {

```

```
final String[] columnNames = {"Nombre",
                              "Apellido",
                              "Pasatiempo",
                              "Años de Practica",
                              "Soltero(a)"};
final Object[][] data = {
    {"Mary", "Campione",
     "Esquiar", new Integer(5), new Boolean(false)},
    {"Lhucas", "Huml",
     "Patinar", new Integer(3), new Boolean(true)},
    {"Kathya", "Walrath",
     "Escalar", new Integer(2), new Boolean(false)},
    {"Marcus", "Andrews",
     "Correr", new Integer(7), new Boolean(true)},
    {"Angela", "Lalth",
     "Nadar", new Integer(4), new Boolean(false)}
};

//únicamente retornamos el numero de elementos del
//array de los nombres de las columnas
public int getColumnCount() {
    return columnNames.length;
}

//retormanos el numero de elementos
//del array de datos
public int getRowCount() {
    return data.length;
}

//retornamos el elemento indicado
public String getColumnName(int col) {
    return columnNames[col];
}

//y lo mismo para las celdas
public Object getValueAt(int row, int col) {
    return data[row][col];
}

/*
 * Este metodo sirve para determinar el editor predeterminado
 * para cada columna de celdas
 */
public Class getColumnClass(int c) {
    return getValueAt(0, c).getClass();
}

/*
 * No tienes que implementar este método a menos que
 * las celdas de tu tabla sean Editables
 */
public boolean isCellEditable(int row, int col) {
    return true;
}

/*
 * No tienes que implementar este método a menos que
 * los datos de tu tabla cambien
 */
public void setValueAt(Object value, int row, int col) {
    data[row][col] = value;
}
```



```

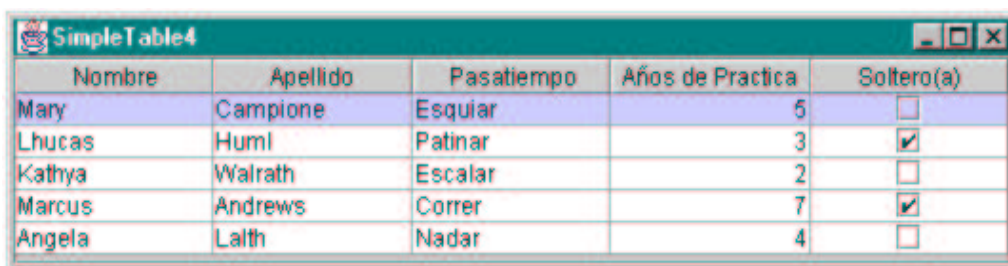
        fireTableCellUpdated(row, col);
    }

}

public static void main(String[] args) {
    SimpleTable4 frame = new SimpleTable4();
    frame.pack();
    frame.setVisible(true);
}
}

```

El resultado de ejecutar el código anterior es el siguiente:



Nombre	Apellido	Pasatiempo	Años de Practica	Soltero(a)
Mary	Campione	Esquiar	5	<input type="checkbox"/>
Lhucas	Huml	Patinar	3	<input checked="" type="checkbox"/>
Kathya	Walrath	Escalar	2	<input type="checkbox"/>
Marcus	Andrews	Correr	7	<input checked="" type="checkbox"/>
Angela	Lalth	Nadar	4	<input type="checkbox"/>

**Figura 5.** SimpleTable4 en ejecución

Podemos ver que, en efecto, en la ultima columna aparece una *checkbox* para mostrar los valores *booleanos*, ya que implementamos el método `getColumnClass`.

## La API JTable

### Campos

Resumen de los Campos		
Campo		Descripción
static int	AUTO_RESIZE_ALL_COLUMNS	Proporciona el Cambio de tamaño de Todas las columnas durante las operaciones de cambio de tamaño
static int	AUTO_RESIZE_LAST_COLUMN	Durante todas las operaciones de cambio de tamaño, aplica el ajuste únicamente a la última columna
static int	AUTO_RESIZE_NEXT_COLUMN	Cuando una columna se ajusta, este campo ajusta la siguiente, de forma opuesta
static int	AUTO_RESIZE_NEXT_OFFSET	Utiliza una barra de desplazamiento para ajustar el ancho de la columna
static int	AUTO_RESIZE_SUBSEQUENT_COLUMNS	Cambia las columnas siguientes para preservar el ancho total; este es el comportamiento por omisión
protected boolean	autoCreateColumnsFromModel	Si tiene el valor de <code>true</code> , la tabla consulta al <code>TableModel</code> para construir el conjunto de columnas
protected int	AutoResizeMode	Determina si la tabla cambia automáticamente el tamaño de la anchura de sus columnas para ocupar el ancho total de la tabla

protected TableCellEditor	cellEditor	Un objeto que sobrescribe la celda actual y permite al usuario cambiar sus contenidos
protected boolean	cellSelectionEnabled	Obsoleto desde la versión 1.3
protected TableColumnModel	columnModel	El <code>TableColumnModel</code> de la tabla
protected TableModel	dataModel	El <code>TableModel</code> de la tabla
protected Hashtable	defaultEditorsByColumnClass	Una tabla de objetos que muestra y edita el contenido de cada celda, indexado por clase como esta declarado en <code>getColumnClass</code> en la interface <code>TableModel</code>
protected Hashtable	defaultRenderersByColumnClass	Una tabla de objetos que muestra el contenido de cada celda, indexado por clase como esta declarado en <code>getColumnClass</code> en la interface <code>TableModel</code>
protected int	editingColumn	Identifica la columna de la celda que esta siendo editada
protected int	editingRow	Identifica la Fila de la celda que esta siendo editada
protected Component	editorComp	El componente que procesa la edición
protected Color	gridColor	El color de la rejilla( <i>grid</i> )
protected Dimension	preferredViewportSize	Utilizado por la interface <code>Scrollable</code> para determinar el area inicial visible
protected int	rowHeight	La altura -en pixeles- de las filas de la tabla
protected int	rowMargin	La altura -en pixeles- del margen entre las celdas en cada fila
protected boolean	rowSelectionAllowed	Devuelve <code>true</code> si se permite selección de fila en esta tabla
protected Color	selectionBackground	El color de fondo de las celdas seleccionadas
protected Color	selectionForeground	El color de primer plano de las celdas seleccionadas
protected ListSelectionModel	selectionModel	El <code>ListSelectionModel</code> de la tabla; se utiliza para controlar las filas seleccionadas
protected boolean	showHorizontalLines	Las líneas horizontales se dibujan entre las celdas cuando el campo esta en <code>true</code>
protected boolean	showVerticalLines	Las líneas verticales se dibujan entre las celdas cuando el campo esta en <code>true</code>
protected JTableHeader	tableHeader	El <code>JTableHeader</code> que funciona con la tabla

## Constructores

Constructores	
Constructor	Descripción
<code>JTable()</code>	Construye un <code>JTable()</code> predeterminado
<code>JTable(int numRows, int numColumns)</code>	Construye un <code>JTable()</code> con <code>numRows</code> y <code>numColumns</code> de celdas vacías, utilizando <code>DefaultTableModel</code>
<code>JTable(Object[][] rowData, Object[] columnNames)</code>	Construye un <code>JTable()</code> visualizando los valores de <code>rowData</code> en una matriz bidimensional, utilizando <code>columnNames</code> como nombres para las columnas

<code>JTable(TableModel dm)</code>	Construye un <code>JTable()</code> con <code>dm</code> como <i>modelo de tabla</i> , un modelo de columna predeterminado y un modelo de selección predeterminado
<code>JTable(TableModel dm, TableColumnModel cm)</code>	Construye un <code>JTable()</code> con <code>dm</code> como <i>modelo de tabla</i> , <code>cm</code> como modelo de columna y un modelo de selección predeterminado
<code>JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)</code>	Construye un <code>JTable()</code> con <code>dm</code> como <i>modelo de tabla</i> , <code>cm</code> como modelo de columna y <code>sm</code> como modelo de selección
<code>JTable(Vector rowData, Vector columnNames)</code>	Construye un <code>JTable()</code> para visualizar los valores del Vector de Vectores, <code>rowData</code> , con nombres de columnas dados en <code>columnNames</code>

## Métodos

Métodos		
	Método	Descripción
void	<code>addColumn(TableColumn aColumn)</code>	Añade la columna <code>aColumn</code> al final de la matriz de columnas
void	<code>addColumnSelectionInterval(int index0, int index1)</code>	Añade las columnas desde <code>index0</code> a <code>index1</code> , incluídam a la selección actual
void	<code>addNotify()</code>	llama al método <code>configureEnclosingScrollPane</code>
void	<code>addRowSelectinInterval(int index0, int index1)</code>	Añade las filas desde <code>index0</code> a <code>index1</code> ,incluida, a la selección actual
void	<code>clearSelection()</code>	Deselecciona todas las columnas y filas seleccionadas
void	<code>columnAdd(TableColumnModelEvent e)</code>	Invocado cuando una columna es agregada al modelo de columna
int	<code>columnAtPoint(Point point)</code>	Obtiene el índice de la columna en que <code>point</code> reside, o -1 si esta fuera del rango[0, <code>getColumn()-1</code> ]
void	<code>columnMarginChanged(ChangeEvent e)</code>	Invocado cuando una columna se mueve debido al cambio de márgenes
void	<code>columnMoved(TableColumnModelEvent e)</code>	Invocado cuando una columna cambia de posición
void	<code>columnRemoved(TableColumnModelEvent e)</code>	Invocado cuando una columna es removida del modelo de columnas actual
void	<code>columnSelectionChanged(ListSelectionChangeEvent e)</code>	Invocado cuando el modelo de selección del <code>TableColumnModel</code> a cambiado
protected void	<code>configureEnclosingScrollPane()</code>	Configura los <code>JScrollPane</code> constenidos instalando el <code>TableHeader</code> de la tabla, <code>columnHeaderView</code> del panel de desplazamiento y así sucesivamente
int	<code>convertColumnIndextoModel(int viewColumnIndex)</code>	Mapea el índice de la columna de la vista en <code>viewColumnIndex</code> al índice de la columna en el modelo de la tabla
int	<code>convertColumnIndexToView(int modelColumnIndex)</code>	Mapea el índice de la columna en el modelo de tabla en <code>modelColumnIndex</code> al índice de la columna en la vista
protected TableColumnModel	<code>createDefaultColumnModel()</code>	Obtiene el modelo de objeto de columna predeterminado, que es un <code>DefaultTableColumnModel</code>
void	<code>createDefaultColumnsFromModel()</code>	Crea columnas predeterminadas para la tabla a partir del modelo de datos utilizando los métodos <code>getColumnCount()</code> y <code>getColumnClass()</code> definidos en la interface <code>TableModel</code>

protected TableModel	createDefaultDataModel()	Obtiene el modelo de objetos de la tabla predeterminado, que es un <code>DefaultTableModel</code>
protected void	createDefaultEditors()	Crea editores de celdas predeterminados para objetos, números y booleanos
protected void	createDefaultRenderers()	Crea renderizadores predeterminados para objetos, números, doubles, dates, booleanos, e iconos
protected ListSelectionModel	createDefaultSelectionModel()	Devuelve el modelo de selección de objetos predeterminado, que es un <code>DefaultSelectionModel</code>
static JScrollPane	createScrollPaneForTable(JTable a table)	Obsoleto. Reemplazado por <code>JScrollPane(aTable)</code>
boolean	editCellAt(int row, int col)	Inicia la edición en la celda situada en <code>row</code> , <code>col</code> , si ésta es editable
boolean	editCellAt(int row, int col, EventObject e)	Inicia la edición en la celda situada en <code>row</code> , <code>col</code> , si ésta es editable
void	editingCanceled(ChangeEvent e)	llamado cuando la edición se cancela
void	editingStoped(ChangeEvent e)	llamado cuando termina la edición
AccessibleContext	getAccessibleContext()	Obtiene el <code>AccessibleContext</code> asociado con la <code>JTable</code>
boolean	getAutoCreateFromModel()	Determina si la tabla es capaz de crear columnas predeterminadas a partir del modelo actual
int	getAutoResizeMode()	Obtiene el modo de cambio de tamaño automático de la tabla
TableCellEditor	getCellEditor()	Devuelve el <code>cellEditor</code>
TableCellEditor	getCellEditor(int row, int column)	Obtiene el editor adecuado para la celda situada en <code>row</code> y <code>column</code>
Rectangle	getCellRect(int row, int column, boolean includeSpacing)	Devuelve un <code>Rectangle</code> que localiza la celda que reside en la intersección de <code>row</code> y <code>column</code>
TableCellRenderer	getCellRenderer(int row, int column)	Devuelve un rederizador adecuado para la celda ubicada en <code>row</code> y <code>column</code>
boolean	getCellSelectionEnabled()	Devuelve <code>true</code> si la seleccion de columnas y filas esta habilitada
TableColumn	getColumn(Object identifier)	Devuelve el objeto <code>TableColumn</code> para la <code>column</code> en la tabla cuyo indicador sea igual a <code>identifier</code> , cuando se compara utilizando <code>equals</code>
Class	getColumnClass(int column)	Devuelve el tipo de columna en una posición de la vista dada
int	getColumnCount()	Devuelve el número de columnas del modelo de columna(éste numero puede ser distinto del número de columnas en el modelo de la tabla)
TableColumnModel	getColumnModel()	Devuelve el <code>TableColumnModel</code> que contiene toda la información de columnas de la tabla
String	getColumnName(int column)	Obtiene el nombre de la columna en la posición <code>column</code> de la vista actual
boolean	getColumnSelectionAllowed()	Devuelve <code>true</code> si se pueden seleccionas las columnas
TableCellEditor	getDefaultEditor(Class columnClass)	Devuelve el editor que se debe utilizar cuando no se ha configurado ningún editor en un <code>TableColumn</code>

TableCellRenderer	getDefaultRenderer(Class columnClass)	Devuelve el renderizador que se debe utilizar cuando no se ha seleccionado un renderizador en un TableColumn
int	getEditingColumn()	Devuelve el índice de la columna de la celda que se encuentra actualmente en edición
int	getEditingRow()	Devuelve el índice de la fila de la celda que se encuentra actualmente en edición
Component	getEditorComponent()	Devuelve el componente que está manejando la sesión de edición
Color	getGridColor()	Devuelve Devuelve el color utilizado para dibujar las líneas de la rejilla
Dimension	getInterCellSpacing()	Devuelve el espaciado vertical y horizontal entre celdas
TableModel	getModel()	Retorna el TableModel que proporciona los datos mostrados por el receptor
Dimension	getPreferredScrollableViewPortSize()	Devuelve el tamaño predeterminado del ViewPort para la tabla
int	getRowCount()	Devuelve el número de filas en la tabla
int	getRowHeight()	Devuelve la altura, en píxeles, de una fila de la tabla
int	getRowMargin()	Devuelve la cantidad de espacio libre entre las filas
boolean	getRowSelectionAllowed()	Devuelve true si se pueden seleccionar las filas
int	getScrollableBlockIncrement(Rectangle visibleRect, int orientation, int direction)	Devuelve el visibleRect.height o visibleRect.width dependiendo de la orientación de la tabla
boolean	getScrollableTracksViewportHeight()	Devuelve true si la altura del ViewPort no determina la altura de la tabla
boolean	getScrollableTracksViewportWidth()	Devuelve true si la anchura del ViewPort no determina la anchura de la tabla
int	getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int direction)	Devuelve el incremento en desplazamiento, en píxeles, que expone completamente una nueva fila o columna
int	getSelectedColumn()	Devuelve el índice de la primera columna seleccionada ó -1 si no hay columna seleccionada
int	getSelectedColumnCount()	Devuelve el número de columnas seleccionadas
int []	getSelectedColumns()	Obtiene los índices de todas las columnas seleccionadas
int	getSelectedRow()	Devuelve el índice de la primera fila seleccionada ó -1 si no existen filas seleccionadas
int	getSelectedRow()	Devuelve el número de filas seleccionadas
int []	getSelectedRows()	Obtiene los índices de todas las filas seleccionadas
Color	getSelectionBackground()	Obtiene el color de fondo para las celdas seleccionadas
Color	getSelectionForeground()	Obtiene el color de primer plano para las celdas seleccionadas
ListSelectionModel	getSelectionModel()	Obtiene el ListSelectionModel que se utiliza para mantener el estado de selección de la fila
boolean	getShowHorizontalLines()	Devuelve true si el receptor dibuja líneas horizontales entre las celdas y false si no es así
boolean	getShowVerticalLines()	Devuelve true si el receptor dibuja líneas verticales entre las celdas y false si no es así
JTableHeader	getTableHeader()	Devuelve el tableHeader utilizado por la tabla

TableUI	getUI()	Devuelve el objeto L&F que renderiza este componente
Object	getValueAt(int row, int column)	Devuelve el valor de la celda ubicada en row, column
protected void	initializeLocalVars()	Inicia las propiedades de tabla con sus valores predeterminados
boolean	isCellEditable(int row, int column)	Devuelve true si la celda ubicada en row, column puede editarse
boolean	isCellSelected(int row, int column)	Devuelve true si la celda ubicada en row, column esta seleccionada
boolean	isColumnSelected(int column)	Devuelve true si la columna ubicada en column esta seleccionada
boolean	isEditing()	Devuelve true si la tabla esta editando una celda
boolean	isManagingFocus()	Sobrescrito para devolver true
boolean	isRowSelected(int row)	Devuelve true si la fila ubicada en row esta seleccionada
void	moveColumn(int column, int targetColumn)	Mueve la columna column a la posición ocupada actualmente por la columna targetColumn
protected String	paramString()	Devuelve una representacion en String de la Tabla
Component	prepareEditor(TableCellEditor editor, int row, int column)	Prepara el editor dado utilizando el valor de la celda dada
Component	prepareRenderer(TableCellRenderer renderer, int row, int column)	Prepara el renderizador dado con un valor adecuado del DataModel
protected boolean	processKeyBinding(KeyStroke ks, KeyEvent e, int condition, boolean pressed)	Invocado para procesar el conjunto de teclas para ks, como resultado de procesar KeyEvent e
void	removeColumn(TableColumn aColumn)	Elimina la columna aColumn de la matriz de columnas de la JTable
void	removeColumnSelectionInterval(int index0, int index1)	Deselecciona las columnas, desde index0 hasta index1, incluyendolo
void	removeEditor()	Descarta el objeto editor
void	removeRowSelectionInterval(int index0, int index1)	Deselecciona las filas, desde index0 hasta index1, incluyendolo
protected void	resizeAndRepaint()	Equivalente a llamar a revalidate() seguido de repaint()
int	rowAtPoint(Point point)	Devuelve el índice de la fila sobre la cual se sitúa el punto point, -1 si el resultado no esta en el rango [0, getColumnCount() - 1]
void	selectAll()	Selecciona todas las columnas, filas y celdas en la tabla
void	setAutoCreateColumnsFromModel()	Asigna el indicador autoCreateColumnsFromModel de la tabla
void	setAutoResize()	Activa el modo de cambio de tamaño automático de la tabla cuando ésta cambia de tamaño
void	setCellEditor(TableCellEditor anEditor)	Asigna la variable cellEditor
void	setCellSelectionEnabled(boolean flag)	Determina si la tabla permitirá tanto la selección de filas como de columnas al mismo tiempo
void	setColumnModel()	Asigna el modelo de columna para esta tabla a newModel y registra receptores de notificación para el nuevo modelo de columna

## Jtable (parte 1)

void	setColumnSelectionAllowed(boolean flag)	Asigna si las columnas de esta tabla pueden ser seleccionadas
void	setColumnSelectionInterval(int index0, int index1)	Selecciona las columnas desde index0, hasta index1, incluido
void	setDefaultEditor(Class columnClass, TableCellEditor editor)	Asigna el editor predeterminado que debe utilizarse si no se asigna un editor a un TableColumn
void	setDefaultRenderer(Class columnClass, TableCellRenderer renderer)	Asignan un renderizador predeterminado que se utilizará si no se asigna un renderizador a TableColumn
void	setEditingColumn(int aColumn)	Asigna la variable editingColumn
void	setEditingRow(int aRow)	Asigna la variable editingRow
void	setGridColor(Color newColor)	Asigna el color utilizado para dibujar las líneas de la rejilla con newColor y visualiza de nuevo el receptor
void	setInterCellSpacing(Dimension newSpacing)	Asigna la anchura y altura entre las celdas a newSpacing y dibuja de nuevo el receptor
void	setModel(TableModel newModel)	Asigna el modelo de datos para esta tabla a newModel y registra los receptores de modificaciones para el nuevo modelo de datos
void	setPreferredScrollableViewportSize(Dimension size)	Asigna el tamaño del viewport para esta tabla
void	setRowHeight(int newHeight)	Asigna la altura newHeight, en pixeles, de las filas
void	setRowHeight(int row, int newHeight)	Asigna la altura newHeight, en pixeles, de la fila row
void	setRowMargin(int newMargin)	Asigna la cantidad de espacio libre entre las filas
void	setRowSelectionAllowed(boolean flag)	Asigna si las filas de esta tabla pueden ser seleccionadas
void	setRowSelectionInterval(int index0, int index1)	Selecciona las filas desde index0, hasta index1, inclusive
void	setSelectionBackground(Color selectionBackground)	Asigna el color de fondo para las celdas seleccionadas
void	setSelectionForeground(Color selectionForeground)	Asigna el color de primer plano para las celdas seleccionadas
void	setSelectionMode(int selectionMode)	Asigna el modo de selección de tabla para permitir selección simple, un intervalo simple continuo ó intervalos múltiples
void	setSelectionModel(ListSelectionModel newModel)	Asigna el modelo de selección de filas newModel para esta tabla
void	setShowGrid(boolean b)	Asigna si se dibujan las líneas alrededor de las celdas
void	setShowHorizontalLines(boolean b)	Asigna si se dibujan líneas horizontales entre las celdas
void	setShowVerticalLines(boolean b)	Asigna si se dibujan líneas verticales entre las celdas
void	setTableHeader(JTableHeader newHeader)	Asigna el tableHeader que trabaja con esta tabla a newHeader
void	setUI(TableUI ui)	Asigna el L&F que renderiza este componente
void	setValueAt(Object aValue, int row, int column)	Asigna el objeto aValue a la celda ubicada en row, column
void	setSizeColumnsToFit(boolean lastColumnOnly)	Obsoleto. reemplazado por sizeColumnsToFit(int)

void	setSizeColumnsToFit(int resizingColumn)	Cambia el tamaño de una o más columnas en la tabla para que la anchura total de todas las columnas de la <code>JTable</code> sea igual a la anchura de la tabla
void	tableChanged( <code>TableModelEvent</code> )	Invocado cuando el <code>TableModel</code> de la tabla genera un <code>TableModelEvent</code>
void	unconfigureEnclosingScrollPane()	Anula los efectos de <code>configureEnclosingScrollPane</code> reemplazando <code>columnHeaderView</code> del panel de desplazamiento por <code>null</code>
void	updateUI()	Llamado por <code>UIManager</code> cuando se cambia el L&F
void	valueChanged( <code>ListSelectionEvent</code> )	Invocado cuando cambia la selección

## La API `DefaultTableModel`

### Campos

Resumen de los Campos		
Campo		Descripción
<code>protected Vector</code>	<code>columnIdentifiers</code>	Un vector de identificadores de columna
<code>protected Vector</code>	<code>dataVector</code>	Un vector de valores <code>Object</code>

### Constructores

Resumen de los Constructores	
Constructor	Descripción
<code>DefaultTableModel()</code>	Construye un <code>DefaultTableModel</code>
<code>DefaultTableModel(int numRows, int numColumns)</code>	Construye un <code>DefaultTableModel</code> con <code>numRows</code> y <code>numColumns</code>
<code>DefaultTableModel(Object[][] data, Object[] columnNames)</code>	Construye un <code>DefaultTableModel</code> e inicia la tabla pasando <code>data</code> y <code>columnNames</code> al método <code>setDataVector()</code>
<code>DefaultTableModel(Object[] columnNames, int numRows)</code>	Construye un <code>DefaultTableModel</code> con un número de columnas equivalentes al número de elementos o valores nulos en <code>columnNames</code> y <code>numRows</code>
<code>DefaultTableModel(Vector columnNames, int numRows)</code>	Construye un <code>DefaultTableModel</code> con un número de columnas equivalentes al número de elementos o valores nulos en <code>columnNames</code> y <code>numRows</code>
<code>DefaultTableModel(Vector data, Vector columnNames)</code>	Construye un <code>DefaultTableModel</code> e inicia la tabla pasando <code>data</code> y <code>columnNames</code> al método <code>setDataVector()</code>

### Métodos

Resumen de los Métodos		
Método		Descripción
void	<code>addColumnn(Object columnName)</code>	Añade una columna al modelo
void	<code>addColumnn(Object columnName, Object[] columnData)</code>	Añade una columna al modelo



## Jtable (parte 1)

void	addColumnn(Object columnName, Vector columnData)	Añade una columna al modelo
void	addRow(Object[] rowData)	Añade una fila al final de modelo
void	addRow(Vector rowData)	Añade una fila al final de modelo
public static Vector	convertToVector(Object[] anArray)	Obtiene un Vector que contiene los mismos objetos que el array
public static Vector	convertToVector(Object[][] anArray)	Obtiene un Vector de vectores que contiene los mismos valores que la matriz anArray
int	getColumnCount()	Devuelve el número de columnas en esta tabla de datos
String	getColumnName(int column)	Añade una fila al final de modelo
Vector	getDataVector	Devuelve el Vector de Vectores que contiene los valores de datos de la tabla
int	getRowCount()	Devuelve el número de filas en esta tabla de datos
Object	getValueAt(int row, int column)	Devuelve un valor de atributo para la celda en la posición row, column
void	insertRow(int row, Object[][] rowData)	Inserta una fila en row en el modelo
void	insertRow(int row, Vector rowData)	Inserta una fila en row en el modelo
boolean	isCellEditable(int row, int column)	Devuelve true si la celda ubicada en row, column se puede editar
void	moveRow(int startIndex, endIndex, int toIndex)	Mueve una o más filas comenzando desde startIndex hasta endIndex y las coloca en toIndex
void	newDataAvailable(TableModelEvent event)	Equivalente a fireTableChanged()
void	newRowsAdded(TableModelEvent event)	Este método permite asegurar que las nuevas filas tienen el número correcto de columnas
void	removeRow(int row)	Elimina del modelo la fila ubicada en row
void	rowsRemoved(TableModelEvent event)	Equivalente a fireTableChanged()
void	setColumnsIdentifiers(Object[] newIdentifiers)	Reemplaza los identificadores de columna del modelo
void	setColumnsIdentifiers(Vector newIdentifiers)	Reemplaza los identificadores de columna del modelo
void	setDataVector(Object[][] newData, Object[] columnNames)	Reemplaza el valor de la variable de instancia dataVector por los valores de la matriz newData
void	setDataVector(Vector newData, Vector columnNames)	Reemplaza el valor de la variable de instancia dataVector por los valores del Vector: newData
void	setNumRows(int newSize)	Asigna el número de filas del modelo. <i>Obsoleto desde la v1.3</i>
void	setRowCount(int rowCount)	Fija el número de filas del modelo
void	setValueAt(Object aValue, int row, int column)	Asigna el valor del objeto para row y column

## La API AbstractTableModel

### Campos

Resumen de los Campos		
Campo		Descripción
protected	EventListener	ListenerList
		La lista de todos los <i>Listener</i>

### Constructores

Constructores	
Constructor	Descripción
AbstractTableModel()	

### Métodos

Resumen de los Métodos		
Método		Descripción
void	addTableModelListener(TableModelListener l)	Agrega a un <i>listener</i> a la lista que se notifica cada vez que ocurre un cambio al modelo de los datos
int	findColumn(String columnName)	Retorna una columna dado su nombre
void	fireTableCellUpdated(int row, int column)	Notifica a todos los <i>listeners</i> que el valor de la celda [ <i>firstRow</i> , <i>lastRow</i> ], ha sido actualizado
void	fireTableChanged(TableModelEvent e)	Reenvia la notificación del evento dado a todos los <i>TableModelListeners</i> que se registraron como <i>listener</i> para este modelo de tabla
void	fireTableDataChanged()	Notifica a todos los <i>listeners</i> que el valor de todas las celdas en la tabla, pueden haber cambiado
void	fireTableRowsDeleted(int firstRow, int lastRow)	Notifica a todos los <i>listeners</i> que las filas dentro del rango [ <i>firstRow</i> , <i>lastRow</i> ], inclusive, han sido eliminadas
void	fireTableRowsInserted(int firstRow, int lastRow)	Notifica a todos los <i>listeners</i> que las filas dentro del rango [ <i>firstRow</i> , <i>lastRow</i> ], inclusive, han sido insertadas
void	fireTableRowsUpdated(int firstRow, int lastRow)	Notifica a todos los <i>listeners</i> que las filas dentro del rango [ <i>firstRow</i> , <i>lastRow</i> ], inclusive, han sido actualizadas
void	fireTableStructuredChanged()	Notifica a todos los <i>listeners</i> que la estructura de la tabla a sido actualizada
Class	getColumnClass (int columnIndex)	Retorna un <code>Object.class</code> sin tomar en cuenta a <code>columnIndex</code>
String	getColumnName(int column)	Retorna un nombre predefinido para la columna usando el estilo de la hoja de cálculo: A, B, C,...
EventListener []	getListeners(Class listenerType)	Devuelve un arreglo con todos los <i>listener</i> del tipo dado en este modelo.
boolean	isCellEditable(int rowIndex, int columnIndex)	Retorna <code>false</code> ; pero se puede sobrescribir y devolver <code>true</code> para que las celdas sean editables
void	removeTableModelListener(TableModelListener l)	Elimina un <i>listener</i> de la lista, esta se notifica cada vez que ocurre un cambio en el modelo de datos.

void	setValueAt(Object aValue, int rowIndex, int columnIndex)	Es un método vacío que no debe ser implementado si el modelo de datos no es editable
------	--	--

## Comentarios finales

Como hemos visto, la creación de tablas se convierte en una tarea sencilla cuando comenzamos a conocer las clases auxiliares que la acompañan para mejorar su rendimiento y eficiencia; la finalidad de este primer artículo es precisamente empezar a familiarizarte con estas clases, en futuros artículos comenzaremos a tratar el manejo de eventos relacionados con las tablas y la realización de tareas más complejas.

## Referencias

Swing Tutorial  
 Matthew Robinson & Pavel Vorobiev  
 Capitulo 18. JTables  
<http://www.manning.com/Robinson/chapter18.pdf>

Tutorial de Swing en SUN  
<http://java.sun.com/docs/books/tutorial/uiswing/components/table.html>

La Biblia de Java 2  
 Steven Holzner  
 Ed. Anaya Multimedia/Coriolis  
 ISBN: 84-415-1037-7

Puede descargar el código de los ejemplos: [jtable.zip](#)

Isaac Ruíz, **RuGI**, egresado del ITI (Istmo de Tehuantepec, Oaxaca, Mexico) en la Carrera de Ingeniería en Sistemas Computacionales, es actualmente desarrollador independiente Java con intenciones de realizar el examen de certificación  
 Cuando no esta programando o navegando (¿?) le gusta mucho leer todo aquello que le de el mas pequeño indicio de como llegar al Valhala o por lo menos a Avalon =:D  
 Para cualquier duda o comentario: [RuGI\\_ARROBA\\_javahispano.com](mailto:RuGI_ARROBA_javahispano.com)

