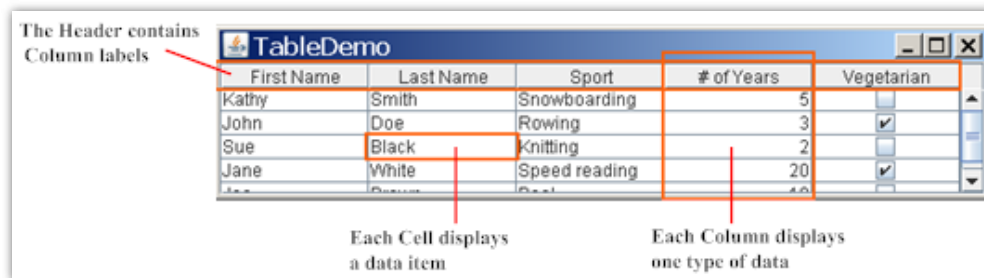


Java: Cách sử dụng bảng (Table)

JTable cung cấp cho ta cách thức hiển thị dữ liệu dưới dạng bảng. **JTable** không chứa hay đệm dữ liệu, nó đơn giản chỉ là cung cấp cách xem dữ liệu. Hình ảnh dưới đây cho thấy cách hiển thị bảng dữ liệu trong **scroll pane**:



Bài viết này sẽ trình bày các chủ đề sau về JTable:

- Tạo bảng đơn
- Đưa bảng vào Container
- Thiết lập và thay đổi độ rộng cột
- Lựa chọn của người dùng
- Tạo Table Model
- Lắng nghe sự thay đổi của dữ liệu
- Kích hoạt sự kiện thay đổi dữ liệu
- Các khái niệm: Editor và Renderer
- Sử dụng Renderer tùy chỉnh
- Chỉ định Tool Tip cho các ô
- Chỉ định Tool Tip cho các cột tiêu đề
- Sắp xếp và lọc
- Sử dụng Combo Box như một Editor
- Sử dụng các Editor khác
- Sử dụng Editor để validate dữ liệu
- In ấn
- Các ví dụ sử dụng Table



Tạo bảng đơn

Bảng trong ví dụ **SimpleTableDemo** bao gồm các cột được khai báo bằng cách dùng mảng String:

```
String[] columnNames = {"First Name",  
                        "Last Name",
```



Dữ liệu của bảng được khởi tạo và lưu trữ dưới dạng mảng đối tượng **hai chiều**:

```
Object[][] data = {
    {"Kathy", "Smith", "Snowboarding", new Integer(5), new Boolean(false)},
    {"John", "Doe", "Rowing", new Integer(3), new Boolean(true)},
    {"Sue", "Black", "Knitting", new Integer(2), new Boolean(false)},
    {"Jane", "White", "Speed reading", new Integer(20), new Boolean(true)},
    {"Joe", "Brown", "Pool", new Integer(10), new Boolean(false)}
};
```

Các cột và dữ liệu ở trên được dùng để tạo bảng bằng cách sử dụng câu lệnh sau:

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

Dưới đây là hai hàm tạo của **JTable** dùng để trực tiếp lấy dữ liệu (**SimpleTableDemo** dùng hàm tạo đầu tiên):

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

Ưu điểm của các hàm tạo này là chúng rất dễ sử dụng. Tuy nhiên chúng cũng có những nhược điểm sau:

- Chúng tự động tạo các ô có thể chỉnh sửa.
- Chúng coi mọi loại dữ liệu đều cùng kiểu (kiểu chuỗi). Ví dụ, nếu một cột của bảng có dữ liệu kiểu **Boolean** thì bảng có thể hiển thị dữ liệu theo dạng check box. Tuy nhiên, nếu bạn sử dụng hai hàm tạo này thì dữ liệu kiểu **Boolean** lại được hiển thị dưới dạng chuỗi. Ta có thể thấy sự khác biệt ở cột **Vegetarian** trong hình trên.
- Chúng yêu cầu rằng ta phải đặt tất cả các dữ liệu của bảng trong một mảng hay một vector, điều này không thích hợp với một số dữ liệu. Ví dụ, nếu ta tạo một tập các đối tượng từ cơ sở dữ liệu thì bạn có thể muốn truy vấn các đối tượng trực tiếp với dữ liệu của chúng thay vì sao chép tất cả các dữ liệu này vào một mảng hay vector.

Nếu bạn muốn khắc phục những hạn chế trên thì bạn cần thực thi model table của chính bạn được mô tả ở chủ đề [Tạo Table Model](#).

Đưa bảng vào Container



Để đưa một bảng nào đó vào bộ chứa thì ta tạo một scroll pane cho bảng đó:

```
JScrollPane scrollPane = new JScrollPane(table);

table.setFillViewportHeight(true);
```

Hai câu lệnh trên làm những việc sau:



thì này là **true** thì bảng sẽ sử dụng toàn bộ độ cao của container, ngay cả khi bảng không có đủ hàng để sử dụng theo chiều dọc. Điều này làm ta dễ sử dụng bảng hơn bởi có thể kéo và thả.

Scroll pane tự động đặt header của bảng ở phần top của viewport. Tên cột vẫn được hiển thị ở phía trên của vùng quan sát khi người dùng cuộn dữ liệu.

Nếu ta sử dụng bảng mà không có scroll pane thì ta phải lấy phần header của bảng và đặt nó vào nơi mong muốn. Ví dụ:

```
container.setLayout(new BorderLayout());
container.add(table.getTableHeader(), BorderLayout.PAGE_START);
container.add(table, BorderLayout.CENTER);
```

Thiết lập và thay đổi độ rộng cột

Mặc định thì tất cả các cột trong một bảng đều khởi đầu với độ rộng giống nhau và tổng độ rộng của chúng bằng độ rộng của bảng. Khi tăng hoặc giảm kích thước của bảng (điều này có thể xảy ra do người dùng thay đổi kích thước của cửa sổ chứa bảng), lúc đó độ rộng các cột cũng được thay đổi tương ứng.

Khi người dùng thay đổi độ rộng một cột bằng cách kéo đường viền của nó thì hoặc là các cột khác cũng phải thay đổi kích thước theo, hoặc là kích thước của bảng phải thay đổi. Mặc định thì kích thước của bảng vẫn như cũ, và tất cả các cột phía bên phải của cột kéo sẽ thay đổi kích thước cho phù hợp.

Để tùy chỉnh độ rộng của các cột thì ta có thể gọi phương thức `setPreferredWidth` trên mỗi cột của bảng. Ví dụ, ta thêm đoạn mã sau vào ví dụ `SimpleTableDemo` để tạo cột thứ ba lớn hơn so với các cột khác:

```
TableColumn column = null;
for (int i = 0; i < 5; i++) {
    column = table.getColumnModel().getColumn(i);
    if (i == 2) {
        column.setPreferredWidth(100); //cột thứ ba rộng hơn
    } else {
        column.setPreferredWidth(50);
    }
}
```

Đoạn mã trên cho thấy mỗi cột trong bảng được thể hiện bởi đối tượng `TableColumn`. `TableColumn` sẽ cung cấp các phương thức getter và setter để thiết lập các độ rộng nhỏ nhất, ưa thích, và lớn nhất của mỗi cột. Ví dụ, để lấy độ rộng ô dựa trên không gian tương đối cần để vẽ nội dung cho ô thì ta xem phương thức `initColumnSizes` trong ví dụ `TableRenderDemo`.

Khi người dùng tiến hành thay đổi kích thước cột thì độ rộng mặc định của cột sẽ được thiết lập lại. Tuy nhiên, khi bản thân bảng thay đổi kích thước — phổ biến là thay đổi kích thước cửa sổ — thì độ rộng mặc định của



Lựa chọn của người dùng

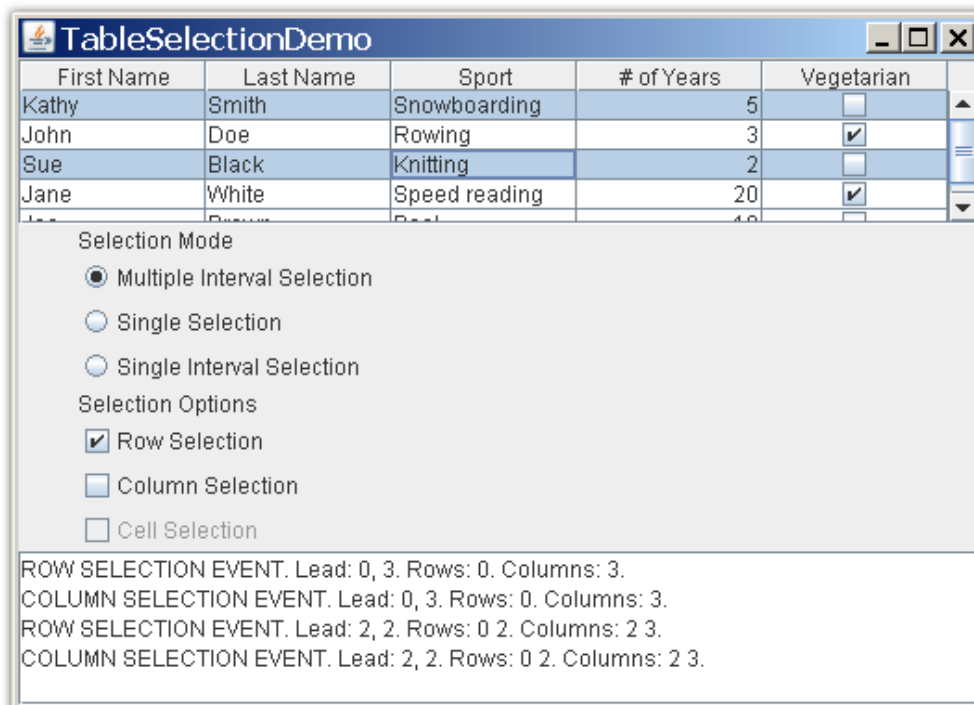
Với cấu hình mặc định thì bảng hỗ trợ sự lựa chọn trên một hoặc nhiều hàng. Người dùng có thể lựa chọn các hàng liên tiếp nhau hoặc lựa chọn các cột mong muốn. Ô cuối cùng mà người dùng chọn sẽ là ô làm dấu; trong Metal look and feel thì ô này sẽ được chỉ ra. Ô này còn được gọi là lựa chọn dẫn dắt, đôi khi nó cũng được gọi là "ô focus" hoặc "ô hiện thời".

Người dùng có thể sử dụng chuột hay bàn phím để chọn hàng như mô tả ở bảng dưới đây:

Hoạt động	Hành động chuột	Hành động bàn phím
Chọn một hàng.	Click.	Phím mũi tên lên và xuống.
Chọn nhiều hàng liên tiếp nhau.	Shift-Click hoặc kéo rê qua các hàng muốn chọn Drag over rows.	Shift-Mũi tên lên hoặc Shift-Mũi tên xuống.
Chọn những hàng mong muốn.	Control-Click	Chuyển lên đầu sự lựa chọn rồi nhấn tổ hợp phím Control-Mũi tên lên hoặc Control-Mũi tên xuống, sau đó sử dụng phím cách Space để thêm sự lựa chọn hoặc dùng tổ hợp phím Control-Space để bỏ đi hàng tương ứng.

Hình dưới đây thể hiện một ứng dụng có tên [TableSelectionDemo](#). Ví dụ này cho phép người dùng thao tác với các tùy chọn JTable. Ví dụ cũng có một text pane để đăng ký sự kiện đã lựa chọn.

Ở hình dưới thì khi người dùng click vào hàng đầu tiên rồi control-click vào hàng thứ ba thì người dùng sẽ chọn được hai hàng tương ứng.



Ở phần "Selection Mode" có ba radio button. Ta click vào lựa chọn "Single Selection", lúc này ta chỉ có thể chọn được một hàng tại một thời điểm. Nếu ta chọn radio "Single Interval Selection" thì ta có thể chọn được



javax.swing.ListSelectionModel: MULTIPLE_INTERVAL_SELECTION, SINGLE_INTERVAL_SELECTION, và SINGLE_SELECTION.

Quay trở lại ví dụ `TableSelectionDemo`, ta để ý tới các check box ở mục "Selection Options". Mỗi checkbox sẽ kiểm soát trạng thái của một biến `boolean` được định nghĩa trong `JTable`:

- "Row Selection" kiểm soát `rowSelectionAllowed` có phương thức setter `setRowSelectionAllowed` và phương thức getter `getRowSelectionAllowed`. Khi thuộc tính này chứa giá trị `true` (và thuộc tính `columnSelectionAllowed` chứa giá trị `false`) thì người dùng có thể chọn được các hàng.
- "Column Selection" kiểm soát `columnSelectionAllowed` có phương thức setter `setColumnSelectionAllowed` và phương thức getter `getColumnSelectionAllowed`. Khi thuộc tính này là `true` (và thuộc tính `rowSelectionAllowed` là `false`) thì ta có thể chọn các cột.
- "Cell Selection" kiểm soát phương thức `cellSelectionEnabled` có các phương thức setter là `setCellSelectionEnabled` và phương thức getter là `getCellSelectionEnabled`. Khi thuộc tính này là `true` thì người dùng có thể chọn một ô đơn hoặc chọn các ô theo khối chữ nhật.

Lưu ý: `JTable` sử dụng một khái niệm chọn rất đơn giản đó là giao điểm của hàng và cột. Nó không được thiết kế để xử lý đầy đủ những ô được chọn độc lập nhau.

Nếu ta không chọn check box nào thì chỉ có lựa chọn đầu tiên là được sử dụng.

Ta có thể lưu ý rằng checkbox "Cell Selection" bị vô hiệu trong chế độ lựa chọn đa thời gian. Nguyên nhân là bởi vì cell selection không được hỗ trợ trong chế độ này trong ví dụ demo. Ta có thể chỉ định sự lựa chọn bằng ô trong chế độ đa thời gian, nhưng kết quả là bảng sẽ không tạo ra những lựa chọn hữu dụng.

Ta cũng có thể lưu ý rằng bất kỳ sự thay đổi nào đối với ba tùy chọn lựa chọn đã nói ở trên đều có thể ảnh hưởng đến những lựa chọn khác. Nguyên nhân là bởi vì nó cho phép lựa chọn cả hàng và cột tương ứng với lựa chọn ô. `JTable` sẽ tự động cập nhật ba biến khi cần để chúng luôn giữ những giá trị phù hợp.

Lưu ý: Việc thiết lập `cellSelectionEnabled` thành một giá trị cũng ảnh hưởng đến việc thiết lập cả `rowSelectionEnabled` và `columnSelectionEnabled` cho giá trị đó. Việc thiết lập cả `rowSelectionEnabled` và `columnSelectionEnabled` cho một giá trị cũng ảnh hưởng đến việc thiết lập `cellSelectionEnabled` tới giá trị đó. Việc thiết lập `rowSelectionEnabled` và `columnSelectionEnabled` thành những giá trị khác nhau cũng ảnh hưởng đến việc thiết lập `cellSelectionEnabled` thành giá trị `false`.

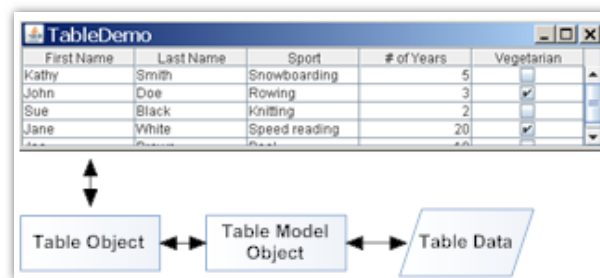
Để truy xuất tới lựa chọn hiện thời thì ta sử dụng phương thức `JTable.getSelectedRows`, phương thức này trả về một mảng các chỉ số hàng, và sử dụng phương thức `JTable.getSelectedColumns`, phương thức này trả về một mảng các chỉ số cột. Để truy xuất tọa độ của lựa chọn đầu tiên thì ta tham chiếu tới những model đã được chọn đối với bản thân bảng đó cũng như model cột của bảng đó. Đoạn mã sau đây định dạng một chuỗi có chứa hàng và cột của lựa chọn đầu tiên (Lưu ý là những lựa chọn của người dùng sẽ tạo ra một loạt các sự kiện):

```
String.format("Lead Selection: %d, %d. ",
    table.getSelectionModel().getLeadSelectionIndex(),
```

Lưu ý: Lựa chọn dữ liệu bản chất là sự mô tả những ô đã chọn bằng "view" (tức là bảng dữ liệu sẽ xuất hiện ra sao sau khi sắp xếp hoặc lọc) chứ không phải là table model. Sự phân biệt này không quan trọng trừ khi dữ liệu bạn đã xem đã được sắp xếp lại bằng cách phân loại, lọc, hoặc người dùng đã thao tác với các cột. Trong trường hợp đó thì ta phải chuyển tọa độ đã chọn bằng cách sử dụng các phương thức chuyển đổi được mô tả tại phần [Sắp xếp và Lọc](#).

Tạo Table Model

Mỗi đối tượng bảng sử dụng một đối tượng *table model* để quản lý dữ liệu bảng thực tế. Mỗi đối tượng table model phải thực thi giao diện `TableModel`. Nếu ta không cung cấp đối tượng table model thì `JTable` sẽ tự động tạo một thể hiện của `DefaultTableModel`. Mỗi quan hệ này được thể hiện như ở bảng dưới đây.



Hàm tạo `JTable` được sử dụng bởi `SimpleTableDemo` sẽ tạo table model cho nó bằng đoạn code như sau:

```

new AbstractTableModel() {
    public String getColumnName(int col) {
        return columnNames[col].toString();
    }
    public int getRowCount() { return rowData.length; }
    public int getColumnCount() { return columnNames.length; }
    public Object getValueAt(int row, int col) {
        return rowData[row][col];
    }
    public boolean isCellEditable(int row, int col)
        { return true; }
    public void setValueAt(Object value, int row, int col) {
        rowData[row][col] = value;
        fireTableCellUpdated(row, col);
    }
}
  
```



Ở đoạn mã trên, việc thực thi một table model có thể trở nên đơn giản hơn. Nói chung thì ta thực thi table model trong lớp con của lớp `AbstractTableModel`.

Model của ta có thể chứa dữ liệu dạng mảng hoặc ánh xạ băm, hoặc lấy dữ liệu từ bên ngoài nguồn như database chẳng hạn. Ta cũng có thể tạo dữ liệu tại thời điểm thực thi.



`SimpleTableDemo` thì sẽ không biết cột # of Years chứa số hay không (nói chung nên được canh phải và có một định dạng cụ thể). Ngoài ra, nó cũng không biết rằng cột `Vegetarian` có chứa các giá trị boolean, giá trị mà có thể được thể hiện bằng check box.

- Table model tùy chỉnh thực thi trong ví dụ `TableDemo` không cho phép bạn chỉnh sửa cột name. Còn trong ví dụ `SimpleTableDemo` thì ta có thể chỉnh sửa tất cả các cột.

Đoạn mã dưới đây trích từ ví dụ `TableDemo` cho thấy sự khác biệt với ví dụ `SimpleTableDemo`. Câu lệnh in đậm chỉ ra rằng việc tạo model của bảng khác so với table model được định nghĩa tự động như ở ví dụ `SimpleTableDemo`.

```
public TableDemo() {
    ...
    JTable table = new JTable(new MyTableModel());
    ...
}

class MyTableModel extends AbstractTableModel {
    private String[] columnNames = ...//same as before...
    private Object[][] data = ...//same as before...

    public int getColumnCount() {
        return columnNames.length;
    }

    public int getRowCount() {
        return data.length;
    }

    public String getColumnName(int col) {
        return columnNames[col];
    }

    public Object getValueAt(int row, int col) {
        return data[row][col];
    }

    public Class getColumnClass(int c) {
        return getValueAt(0, c).getClass();
    }

    /*
     * Don't need to implement this method unless your table's
     * editable.
     */
    public boolean isCellEditable(int row, int col) {
        //Note that the data/cell address is constant,
        //no matter where the cell appears onscreen.
        if (col < 2) {
            return false;
        } else {
            return true;
        }
    }
}
```





```

// Don't need to implement this method unless your table's
// data can change.
*/
public void setValueAt(Object value, int row, int col) {
    data[row][col] = value;
    fireTableCellUpdated(row, col);
}
...
}

```

Lắng nghe những thay đổi dữ liệu

Mỗi table model có thể có một tập các bộ lắng nghe được thông báo mỗi khi có sự thay đổi dữ liệu bảng. Bộ lắng nghe là các thể hiện của `TableModelListener`. Trong đoạn mã sau, `SimpleTableDemo` được mở rộng để tích hợp một bộ lắng nghe. Những dòng lệnh mới được thể hiện bằng chữ in đậm:

```

import javax.swing.event.*;
import javax.swing.table.TableModel;

public class SimpleTableDemo ... implements TableModelListener {
    ...
    public SimpleTableDemo() {
        ...
        table.getModel().addTableModelListener(this);
        ...
    }

    public void tableChanged(TableModelEvent e) {
        int row = e.getFirstRow();
        int column = e.getColumn();
        TableModel model = (TableModel)e.getSource();
        String columnName = model.getColumnName(column);
        Object data = model.getValueAt(row, column);

        ...// Làm gì đó với dữ liệu...
    }
    ...
}

```



Kích hoạt sự kiện thay đổi dữ liệu

Theo thứ tự thì để kích hoạt sự kiện thay đổi dữ liệu thì table model phải biết cách cấu trúc một đối tượng `TableModelEvent`. Điều này có thể là một thủ tục phức tạp, nhưng có thể thực hiện được bằng `DefaultTableModel`. Ta cũng có thể cho phép `JTable` sử dụng thể hiện mặc định của nó là `DefaultTableModel`, hoặc tạo lớp con tùy chỉnh của `DefaultTableModel`.



Sưu tập AbstractTableModel mới khi dữ liệu bảng thay đổi từ nguồn bên ngoài.

Phương thức	Thay đổi
<code>fireTableCellUpdated</code>	Cập nhật cột được chỉ định.
<code>fireTableRowsUpdated</code>	Cập nhật hàng được chỉ định.
<code>fireTableDataChanged</code>	Cập nhật toàn bộ bảng (chỉ dữ liệu của bảng).
<code>fireTableRowsInserted</code>	Những hàng mới được chèn vào.
<code>fireTableRowsDeleted</code>	Những hàng đã bị xóa
<code>fireTableStructureChanged</code>	Hủy bỏ tính hiệu lực trên toàn bảng, bao gồm cả dữ liệu và cấu trúc.

Các khái niệm: Editor và Renderer

Trước khi ta đi tới một số tác vụ tiếp theo thì ta cần phải hiểu được cách mà mỗi bảng vẽ các ô của nó. Có thể bạn mong muốn rằng mỗi ô của bảng là một thành phần để dễ thao tác. Tuy nhiên vì lý do hiệu năng mà bảng trong Swing lại không như vậy.

Thay vào đó, *cell renderer* được dùng đến với mục đích vẽ tất cả các ô có cùng kiểu dữ liệu. Ta có thể hiểu renderer như là con dấu mực có thể cấu hình mà bảng sử dụng để đánh dấu các định dạng dữ liệu khác nhau cho mỗi ô. Khi người dùng tiến hành sửa dữ liệu của ô thì *cell editor* sẽ được dùng đến trên ô đó để kiểm soát hành vi chỉnh sửa của ô.

Ví dụ như mỗi ô trong cột # of Years trong ví dụ `TableDemo` có chứa dữ liệu kiểu `Number` — để xác định đối tượng kiểu `Integer`. Mặc định thì cell renderer của cột kiểu `Number` được sử dụng như một thể hiện của `JLabel` để vẽ các số và canh phải chúng trên các ô của cột đó. Nếu người dùng tiến hành sửa một ô nào đó thì mặc định cell editor sẽ cảnh phải `JTextField` để kiểm soát việc chỉnh sửa ô.

Để chọn renderer để hiển thị ô trong cột thì trước tiên bảng sẽ xác định xem ta có chỉ định renderer cho cột tương ứng không, nếu bạn không làm điều này thì bảng sẽ gọi phương thức `getColumnClass` của table model để lấy kiểu dữ liệu của các ô của cột đó. Sau đó bảng sẽ so sánh kiểu dữ liệu của cột với danh sách kiểu dữ liệu cho cell renderer được đăng ký. Danh sách này được khởi tạo bởi bảng, tuy nhiên ta có thể thêm hoặc thay đổi nó. Bảng có thể đưa các kiểu dữ liệu sau vào danh sách:

- `Boolean` — được render với check box.
- `Number` — được render bởi nhãn canh phải.
- `Double`, `Float` — tương tự kiểu `Number` nhưng việc chuyển từ đối tượng sang text được thực hiện bởi một thể hiện kiểu `NumberFormat` (dùng định dạng số mặc định cho địa phương hiện thời).
- `Date` — được render bởi một nhãn, trong đó việc chuyển từ đối tượng sang text được thực hiện bởi thể hiện của `DateFormat` (sử dụng short style cho date and time).
- `ImageIcon`, `Icon` — được render bởi nhãn canh giữa.
- `Object` — được render bởi một nhãn hiển thị giá trị chuỗi của đối tượng.

Các cell editor được chọn bằng cách sử dụng cùng một giải thuật.

Cần nhớ rằng nếu ta tạo bảng trong model của chính nó thì kiểu dữ liệu cho mỗi cột của bảng đó sẽ là `Object`. Để chỉ định chính xác kiểu của cột thì table model phải định nghĩa phương thức `getColumnClass` một cách





bảng thì kỹ thuật mà ta sử dụng thay đổi theo sự sắp xếp của sự kiện mà ta quan tâm:

Tình huống	Cách lấy sự kiện
Để xác định các sự kiện từ ô đang được chỉnh sửa	Sử dụng cell editor (hoặc đăng ký một listener trên cell editor).
Để xác định hàng/cột/ô nào được chọn hoặc không được chọn	Sửa dụng selection listener được mô tả tại phần Lựa chọn của người dùng .
Để xác định các sự kiện chuột trên một tiêu đề cột	Đăng ký kiểu mouse listener thích hợp trên đối tượng <code>JTableHeader</code> của bảng (xem chi tiết tại ví dụ TableSorter).
Để xác định các sự kiện khác	Đăng ký bộ lắng nghe thích hợp trên đối tượng <code>JTable</code> .

Sử dụng Renderer tùy chỉnh

Phần này sẽ trình bày về cách tạo và chỉ định một cell renderer. Ta có thể thiết lập một cell renderer bằng cách sử dụng phương thức `JTable` là `setDefaultRenderer`. Để chỉ định các ô trong cột cụ thể thì ta nên sử dụng một renderer, khi này ta nên sử dụng phương thức của `TableColumn` là `setCellRenderer`. Ta cũng có thể chỉ định một cell renderer bằng cách tạo một lớp con của `JTable`.

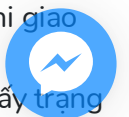
Ta có thể dễ dàng tùy chỉnh text hoặc ảnh được render bởi renderer mặc định là lớp `DefaultTableCellRenderer`. Cụ thể, ta chỉ cần tạo một lớp con của lớp này và thực thi phương thức `setValue` để nó gọi các phương thức `setText` hoặc `setIcon`. Dưới đây là ví dụ minh họa điều này:

```
static class DateRenderer extends DefaultTableCellRenderer {
    DateFormat formatter;
    public DateRenderer() { super(); }

    public void setValue(Object value) {
        if (formatter==null) {
            formatter = DateFormat.getDateInstance();
        }
        setText((value == null) ? "" : formatter.format(value));
    }
}
```

Nếu việc thừa kế lớp `DefaultTableCellRenderer` vẫn chưa đáp ứng đủ nhu cầu thì ta có thể xây dựng thêm một renderer sử dụng lớp con khác. Cách dễ nhất để tạo lớp con của một thành phần có sẵn là thực thi giao diện `TableCellRenderer`. `TableCellRenderer` chỉ yêu cầu một phương thức duy nhất là `getTableCellRendererComponent`. Việc thực hiện phương thức này sẽ cài đặt thành phần render để lấy trạng thái và trả về thành phần.

Ở ví dụ `TableDialogEditDemo` thì renderer được sử dụng cho các ô của cột Favorite Color là một lớp con của lớp `JLabel` gọi là `ColorRenderer`. Dưới đây là đoạn mã trích từ ví dụ [ColorRenderer](#) cho thấy cách thực hiện.





```
public ColorRenderer(boolean isBordered) {
    this.isBordered = isBordered;
    setOpaque(true); //dùng để hiển thị màu nền.
}

public Component getTableCellRendererComponent(
    JTable table, Object color,
    boolean isSelected, boolean hasFocus,
    int row, int column) {
    Color newColor = (Color)color;
    setBackground(newColor);
    if (isBordered) {
        if (isSelected) {
            ...
            //selectedBorder là một đường viền nét liền và có màu
            //table.getSelectionBackground().
            setBorder(selectedBorder);
        } else {
            ...
            //unselectedBorder cũng là một đường viền nét liền và có màu
            //table.getBackground().
            setBorder(unselectedBorder);
        }
    }

    setToolTipText(...); //được bàn luận ở phần tiếp theo
    return this;
}
}
```

Dưới đây là đoạn mã trích từ ví dụ [TableDialogEditDemo](#) dùng để đăng ký một thể hiện của `ColorRenderer` như là renderer mặc định cho tất cả các dữ liệu `Color`:

```
table.setDefaultRenderer(Color.class, new ColorRenderer(true));
```

Để chỉ định cell-specific renderer thì ta cần định nghĩa một lớp con của `JTable` để ghi đè phương thức `getCellRenderer`. Ví dụ, đoạn mã sau đây sẽ thiết lập ô đầu tiên của cột đầu tiên của bảng sử dụng renderer tùy chỉnh:

```
TableCellRenderer weirdRenderer = new WeirdRenderer();
table = new JTable(...) {
    public TableCellRenderer getCellRenderer(int row, int column) {
```





```
// nếu không thì  
return super.getCellRenderer(row, column);  
}  
};
```

Chỉ định Tool Tip cho ô

Mặc định thì tool tip text hiển thị cho ô của bảng được xác định bởi renderer của ô. Tuy nhiên thì đôi khi tool tip text có thể được chỉ định đơn giản hơn bằng cách ghi đè sự thực thi của `JTable` đối với phương thức `getToolTipText(MouseEvent)`. Phần này sẽ cung cấp cho ta cách sử dụng cả hai kỹ thuật này.

Để thêm tool tip cho ô sử dụng renderer của nó thì trước tiên ta cần lấy hoặc tạo renderer cho nó. Sau đó, sau khi đảm bảo rằng thành phần rendering là một `JComponent` thì ta gọi phương thức `setToolTipText` cho nó.

Dưới đây đoạn mã trích từ ví dụ `TableRenderDemo` về cách thiết lập tool tip cho các ô của cột **Sport**:

```
DefaultTableCellRenderer renderer =  
    new DefaultTableCellRenderer();  
renderer.setToolTipText("Click for combo box");  
sportColumn.setCellRenderer(renderer);
```

Mặc dù tool tip text trong ví dụ trên có dạng static nhưng ta có thể thực thi tool tip phụ thuộc vào trạng thái của ô hay chương trình. Dưới đây là hai cách thực hiện:

- Thêm một vài câu lệnh vào sự thực thi của renderer của phương thức `getTableCellRendererComponent`.
- Ghi đè phương thức của `JTable` là `getToolTipText(MouseEvent)`.

`TableDialogEditDemo` sử dụng một cho các màu và được thực hiện tại ví dụ `ColorRenderer`, nó sẽ thiết lập tool tip text bằng cách sử dụng các lệnh in đậm trong đoạn mã sau:

```
public class ColorRenderer extends JLabel implements TableCellRenderer {  
    ...  
    public Component getTableCellRendererComponent(  
        JTable table, Object color,  
        boolean isSelected, boolean hasFocus,  
        int row, int column) {  
        Color newColor = (Color)color;  
        ...  
        setToolTipText("RGB value: " + newColor.getRed() + ", "  
            + newColor.getGreen() + ", "  
            + newColor.getBlue());  
        return this;  
    }  
}
```





Hình dưới đây thể hiện sự hiện thị của tool tip.

First Name	Favorite Color	Sport	# of Years	Vegetarian
Mary		Snowboarding	5	<input type="checkbox"/>
Alison		Rowing	3	<input checked="" type="checkbox"/>
Kathy		Knitting	2	<input type="checkbox"/>
Sharon		Speedboarding	20	<input checked="" type="checkbox"/>

Ta có thể chỉ định tool tip text bằng cách ghi đè phương thức `JTable's getToolTipText\(MouseEvent\)`. Ví dụ `TableToolTipsDemo` sẽ cho bạn thấy điều này.

Các ô có tool tip nằm trong hai cột **Sport** và **Vegetarian**. Hình ảnh dưới đây thể hiện cho ví dụ `TableToolTipsDemo`:

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Rowing	3	<input checked="" type="checkbox"/>
Sue	Black	Knitting	2	<input type="checkbox"/>
Jane	White	Speedboarding	20	<input type="checkbox"/>

Còn dưới đây là đoạn mã (thuộc ví dụ `TableToolTipsDemo`) thực thi tool tip cho các ô của các cột **Sport** và **Vegetarian**:

```
JTable table = new JTable(new MyTableModel()) {
    //Implement table cell tool tips.
    public String getToolTipText(MouseEvent e) {
        String tip = null;
        java.awt.Point p = e.getPoint();
        int rowIndex = rowAtPoint(p);
        int colIndex = columnAtPoint(p);
        int realColumnIndex = convertColumnIndexToModel(colIndex);

        if (realColumnIndex == 2) { //Sport column
            tip = "This person's favorite sport to "
                + "participate in is: "
                + getValueAt(rowIndex, colIndex);
        } else if (realColumnIndex == 4) { //Veggie column
            TableModel model = getModel();
            String firstName = (String)model.getValueAt(rowIndex, 0);
            String lastName = (String)model.getValueAt(rowIndex, 1);
            Boolean veggie = (Boolean)model.getValueAt(rowIndex, 4);
            if (Boolean.TRUE.equals(veggie)) {
                tip = firstName + " " + lastName

```





```

        + " is not a vegetarian";
    }

    } else { //another column
        //You can omit this part if you know you don't
        //have any renderers that supply their own tool
        //tips.
        tip = super.getToolTipText(e);
    }
    return tip;
}
...
}

```

Đoạn mã trên khá đơn giản, ngoại trừ lời một điểm là lời gọi tới phương thức `convertColumnIndexToModel`. Lời gọi này là cần thiết bởi vì nếu người dùng di chuyển chuột quanh các cột thì chỉ số của view cho cột sẽ không tương thích với chỉ số của model cho cột đó. Ví dụ như người dùng có thể kéo quanh cột **Vegetarian** (có chỉ số view là 4) làm cho nó được hiển thị như là cột đầu tiên (có chỉ số view là 0). Vì `prepareRenderer` cung cấp chỉ số view nên ta cần dịch chỉ số view sang chỉ số model nên bạn có thể chắc chắn rằng cột tương ứng sẽ được chọn.

Chỉ định Tool Tip cho tiêu đề cột

Ta có thể thêm tool tip cho tiêu đề cột bằng cách thiết lập tool tip text cho `JTableHeader` của bảng. Thông thường thì các tiêu đề cột khác nhau sẽ yêu cầu những tool tip text khác nhau. Ta có thể thay đổi văn bản bằng cách ghi đè phương thức `getToolTipText` của tiêu đề của bảng. Ngoài ra, ta có thể gọi phương thức `TableColumn.setHeaderRenderer` để cung cấp một renderer tùy chỉnh cho tiêu đề.

Đoạn mã dưới đây trích từ ví dụ [TableSorterDemo](#) dùng để thiết lập tool tip text:

```

table.getTableHeader().setToolTipText(
    "Click to sort; Shift-Click to sort in reverse order");

```

[TableToolTipsDemo](#) có một ví dụ thực thi tool tip cho tiêu đề cột cho phép thay đổi theo cột. Kết quả của ví dụ này được thể hiện như hình dưới đây. Bạn sẽ nhìn thấy tool tip khi di chuyển chuột chạm vào tiêu đề cột bất kỳ ngoại trừ hai cột đầu tiên. Không có tool tip nào được áp dụng cho các cột First Name và Last Name là bởi vì tên của chúng đã đủ cung cấp thông tin cho ta biết cột chứa gì.

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Rowing		<input type="checkbox"/>
Sue	Black	Knitting	2	<input type="checkbox"/>
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>
...	<input type="checkbox"/>

Một số hàm của lớp `JTableHeader`:

```
protected String[] columnToolTips = {
    null, // "First Name" assumed obvious
    null, // "Last Name" assumed obvious
    "The person's favorite sport to participate in",
    "The number of years the person has played the sport",
    "If checked, the person eats no meat"};
...

JTable table = new JTable(new MyTableModel()) {
    ...

    //Implement table header tool tips.
    protected JTableHeader createDefaultTableHeader() {
        return new JTableHeader(columnModel) {
            public String getToolTipText(MouseEvent e) {
                String tip = null;
                java.awt.Point p = e.getPoint();
                int index = columnModel.getColumnIndexAtX(p.x);
                int realIndex =
                    columnModel.getColumn(index).getModelIndex();
                return columnToolTips[realIndex];
            }
        };
    }
};
```

Sắp xếp và lọc

Việc sắp xếp và lọc bảng được quản lý bởi đối tượng *sorter*. Cách đơn giản nhất để cung cấp một đối tượng sorter là thiết lập thuộc tính ràng buộc `autoCreateRowSorter` thành `true`:

```
JTable table = new JTable();
table.setAutoCreateRowSorter(true);
```



Các câu lệnh trên sẽ định nghĩa một row sorter là một thể hiện của lớp `javax.swing.table.TableRowSorter`. Nó cung cấp một bảng để thực hiện thao tác sắp xếp khi người dùng click vào tiêu đề của một cột. Hình dưới đây minh họa cho ví dụ [TableSortDemo](#) với khả năng sắp xếp:



Sue	Black	Knitting	2	<input type="checkbox"/>
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>
Jas	Brown	Fast	40	<input type="checkbox"/>

Để có nhiều quyền điều khiển hơn với việc sắp xếp thì ta có thể cấu trúc một thể hiện của `TableRowSorter` và chỉ định nó làm đối tượng sorter cho bảng của ta.

```
TableRowSorter<TableModel> sorter = new TableRowSorter<TableModel>
(table.getModel());
table.setRowSorter(sorter);
```

`TableRowSorter` sử dụng đối tượng `java.util.Comparator` để sắp xếp các hàng. Lớp thực thi giao diện này phải cung cấp một phương thức gọi là `compare` để so sánh hai đối tượng nhằm mục đích sắp xếp. Đoạn mã sau đây sẽ tạo một `Comparator` để sắp xếp một tập các chuỗi theo từ cuối cùng trong mỗi chuỗi:

```
Comparator<String> comparator = new Comparator<String>() {
    public int compare(String s1, String s2) {
        String[] strings1 = s1.split("\\s");
        String[] strings2 = s2.split("\\s");
        return strings1[strings1.length - 1]
            .compareTo(strings2[strings2.length - 1]);
    }
};
```

Ví dụ trên là khá đơn giản; thông thường thì việc thực thi `Comparator` được giao cho lớp con của lớp `java.text.Collator`. Ta có thể định nghĩa lớp con này, sử dụng các phương thức của `Collator` để lấy `Comparator` cho một vùng cụ thể, hoặc sử dụng `java.text.RuleBasedCollator`.

Để xác định xem `Comparator` được sử dụng cho cột nào thì `TableRowSorter` sẽ cố gắng áp dụng lần lượt một trong các quy tắc sau đây. Quy tắc nào mà việc cung cấp sorter với một `Comparator` được sử dụng thì những quy tắc còn lại sẽ bị bỏ qua.

1. Nếu một bộ so sánh được chỉ định bằng cách gọi `setComparator` thì bộ so sánh đó được sử dụng.
2. Nếu table model thông báo rằng dữ liệu cột bao gồm các chuỗi (`TableModel.getColumnClass` trả về `String.class` cho cột đó), thì việc sử dụng bộ so sánh để sắp xếp các chuỗi sẽ dựa trên vùng hiện thời.
3. Nếu column class được trả về bởi `TableModel.getColumnClass` thực thi `Comparable`, thì việc sử dụng bộ so sánh để sắp xếp chuỗi sẽ dựa trên giá trị trả về bởi `Comparable.compareTo`.
4. Nếu một bộ chuyển đổi chuỗi được chỉ định cho bảng bằng cách gọi `setStringConverter`, thì việc sử dụng bộ so sánh để sắp xếp chuỗi kết quả sẽ dựa trên vùng hiển thị.
5. Nếu không quy tắc nào ở trên được áp dụng thì sử dụng một bộ so sánh để gọi `toString` trên cột dữ liệu và sắp xếp các chuỗi kết quả dựa trên vùng hiển thị.

Để có thể sắp xếp phức tạp hơn thì ta sử dụng lớp con `TableRowSorter` hoặc lớp cha `javax.swing.DefaultRowSorter`.



```
List <RowSorter.SortKey> sortKeys
    = new ArrayList<RowSorter.SortKey>();
sortKeys.add(new RowSorter.SortKey(1, SortOrder.ASCENDING));
sortKeys.add(new RowSorter.SortKey(0, SortOrder.ASCENDING));
sorter.setSortKeys(sortKeys);
```

Ngoài việc sắp xếp lại các kết quả thì table sorter còn có thể chỉ định những hàng muốn hiển thị. Điều này được gọi là *lọc*. `TableRowSorter` sẽ thực hiện việc lọc bằng cách sử dụng đối tượng `javax.swing.RowFilter`. `RowFilter` thực hiện một số phương thức để tạo các bộ lọc phổ biến. Ví dụ, `regexFilter` trả về một `RowFilter` để lọc dựa trên *biểu thức chính quy*.

Trong đoạn mã dưới đây sẽ tạo một đối tượng sorter tường minh và ta có thể sử dụng nó để chỉ định một bộ lọc:

```
MyTableModel model = new MyTableModel();
sorter = new TableRowSorter<MyTableModel>(model);
table = new JTable(model);
table.setRowSorter(sorter);
```

Sau đó ta tiến hành lọc dựa trên giá trị muốn lọc nằm trong text field:

```
private void newFilter() {
    RowFilter<MyTableModel, Object> rf = null;
    //If current expression doesn't parse, don't update.
    try {
        rf = RowFilter.regexFilter(filterText.getText(), 0);
    } catch (java.util.regex.PatternSyntaxException e) {
        return;
    }
    sorter.setRowFilter(rf);
}
```

Ở ví dụ trên, `newFilter()` được gọi mỗi khi dữ liệu trong text field thay đổi. Khi người dùng nhập vào một *biểu thức chính quy* phức tạp thì khối `try...catch` sẽ có nhiệm vụ kiểm tra và ngăn lại cú pháp lỗi.

Khi bảng sử dụng một sorter thì dữ liệu mà người dùng nhìn thấy có thể có thứ tự khác với dữ liệu được chỉ định data model, và có thể không bao gồm những hàng được chỉ định bởi data model. Dữ liệu mà người dùng thực sự nhìn thấy được gọi là *view*, và được thiết lập tọa độ riêng. `JTable` cung cấp các phương thức để chuyển từ các tọa độ của model sang các tọa độ của view, những phương thức này gồm `convertColumnIndexToView` và `convertRowIndexToView`. Còn nếu muốn chuyển từ tọa độ view sang tọa độ model thì ta sử dụng các phương thức `convertColumnIndexToModel` và `convertRowIndexToModel`.



so thay doi nho so voi vi du **TableDemo**. Lưu ý rằng hàng số 3 trong model và tương tự như hàng số 3 trong view:

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Rowing	3	<input checked="" type="checkbox"/>
Sue	Black	Knitting	2	<input type="checkbox"/>
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>
Joe	Brown	Pool	10	<input type="checkbox"/>

Filter Text:

Status: Selected Row in view: 3. Selected Row in model: 3.

Nếu người dùng click hai lần vào hàng số 2 thì hàng số 4 trở thành hàng đầu tiên (áp dụng trong view):

First Name	Last Name	Sport	# of Years	Vegetarian
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Rowing	3	<input checked="" type="checkbox"/>
Joe	Brown	Pool	10	<input type="checkbox"/>
Sue	Black	Knitting	2	<input type="checkbox"/>

Filter Text:

Status: Selected Row in view: 0. Selected Row in model: 3.

Phần văn bản mà người dùng nhập vào phần "Filter Text" sẽ xác định một bộ lọc để xác định hàng nào được hiển thị. Cùng với việc sắp xếp, việc lọc có thể làm cho tọa độ view lệch so với tọa độ model:

First Name	Last Name	Sport	# of Years	Vegetarian
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>

Filter Text:

Status: Selected Row in view: 0. Selected Row in model: 3.

Dưới đây là đoạn mã cập nhật trường trạng thái để phản ánh sự lựa chọn hiện thời:

```
table.getSelectionModel().addListSelectionListener(
    new ListSelectionListener() {
        public void valueChanged(ListSelectionEvent event) {
            int viewRow = table.getSelectedRow();
            if (viewRow < 0) {
                //Selection got filtered away.
                statusText.setText("");
            } else {
```





```

        String.format("Selected Row in view: %d. " +
            "Selected Row in model: %d.",
            viewRow, modelRow));
    }
}
}
);

```

Sử dụng Combo Box như là một Editor

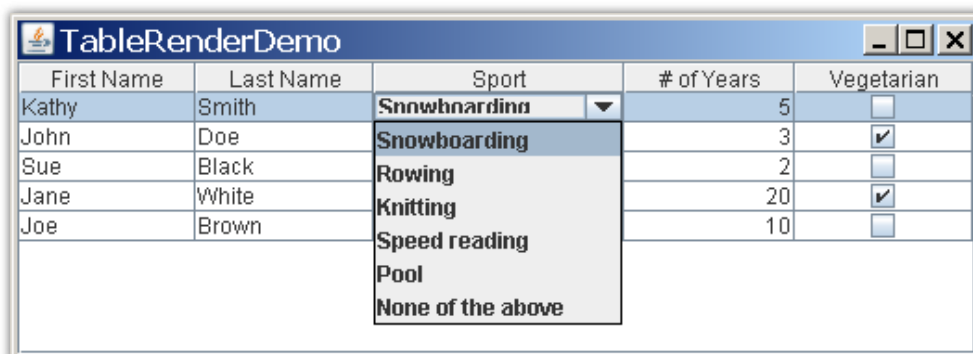
Việc thiết lập một combo box như là một editor khá đơn giản, cách thức của việc này được thể hiện như đoạn mã dưới đây được trích trong ví dụ `TableFilterDemo`. Câu lệnh in đậm của đoạn mã sẽ thiết lập combo box như là editor cho cột được chỉ định.

```

TableColumn sportColumn = table.getColumnModel().getColumn(2);
...
JComboBox comboBox = new JComboBox();
comboBox.addItem("Snowboarding");
comboBox.addItem("Rowing");
comboBox.addItem("Chasing toddlers");
comboBox.addItem("Speed reading");
comboBox.addItem("Teaching high school");
comboBox.addItem("None");
sportColumn.setCellEditor(new DefaultCellEditor(comboBox));

```

Hình dưới đây thể hiện kết quả:



First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Snowboarding	3	<input checked="" type="checkbox"/>
Sue	Black	Rowing	2	<input type="checkbox"/>
Jane	White	Knitting	20	<input checked="" type="checkbox"/>
Joe	Brown	Speed reading	10	<input type="checkbox"/>



Sử dụng Editor khác

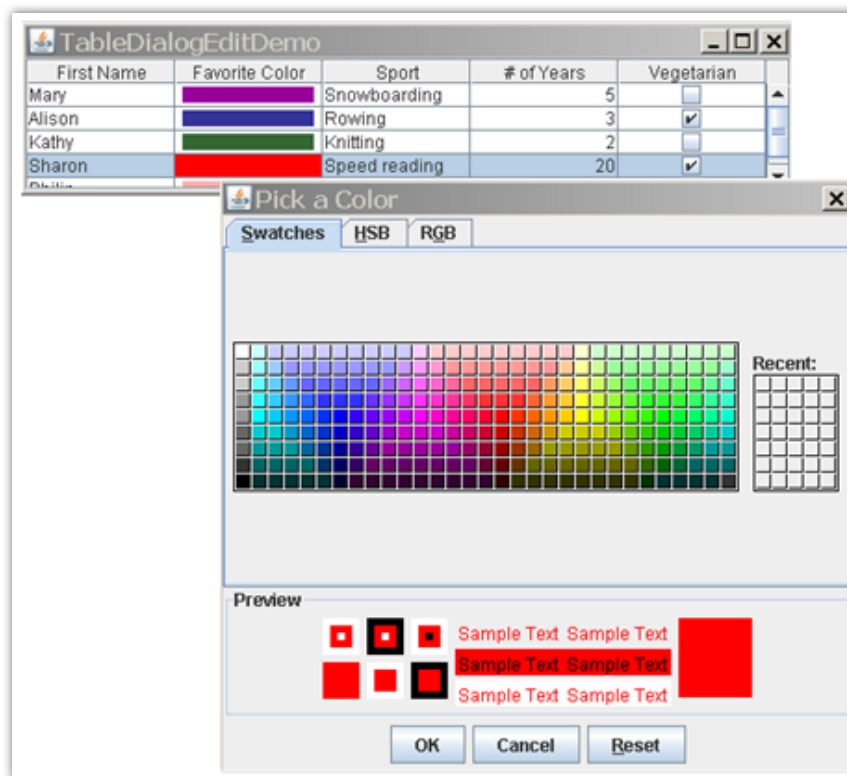
Khi ta thiết lập editor cho các ô của cột (sử dụng phương thức `setCellEditor` của `TableColumn`) hoặc để chỉ định kiểu dữ liệu (sử dụng phương thức `setDefaultEditor` của `JTable`), thì ta chỉ định editor bằng cách sử dụng một đối số tuân theo giao diện `TableCellEditor`. May mắn thay, lớp `DefaultCellEditor` thực thi giao diện và cung cấp các hàm tạo để cho phép ta chỉ định một thành phần soạn thảo là `JTextField`, `JCheckBox`,



bạn biết, `DefaultCellEditor` không hỗ trợ các kiểu thành phần khác, nên ta phải làm thêm một vài việc. Bạn cần tạo một lớp thực thi giao diện `TableCellEditor`. Lớp `AbstractCellEditor` là một lớp cha ta nên sử dụng, nó thực thi giao diện của `TableCellEditor` là `CellEditor`, điều này giúp ta giảm được những vấn đề liên quan đến thực thi sự kiện kích hoạt mã lệnh cần thiết cho các editor của các cell.

Lớp cell editor cần định nghĩa ít nhất hai phương thức là `getCellEditorValue` và `getTableCellEditorComponent`. Phương thức `getCellEditorValue` được yêu cầu bởi `CellEditor`, nó trả về giá trị của ô hiện thời. Phương thức `getTableCellEditorComponent` được yêu cầu bởi `TableCellEditor`, nó cần cấu hình và trả về thành phần mà bạn muốn sử dụng như là editor.

Dưới đây là một hình ảnh thể hiện một bảng với một hộp thoại được sử dụng như là một cell editor gián tiếp. Khi người dùng bắt đầu sửa một ô trong cột Favorite Color thì một nút lệnh (cell editor thực sự) sẽ xuất hiện và đưa tới hộp thoại, trong đó người dùng có thể chọn một màu mong muốn.



Dưới đây là đoạn mã trích từ ví dụ `ColorEditor` có nhiệm vụ thực thi cell editor.

```
public class ColorEditor extends AbstractCellEditor implements TableCellEditor,
    ActionListener {
    Color currentColor;
    JButton button;
    JColorChooser colorChooser;
    JDialog dialog;
    protected static final String EDIT = "edit";
```

