

## SwingのJTableコンポーネントでセルを描く

### Professional Java Programming 第6章からの抜粋

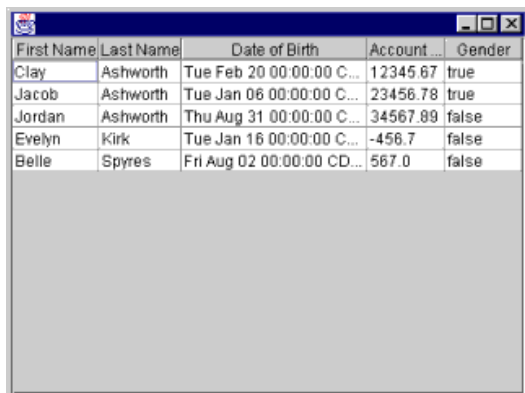
Brett Spell

2000年 11月 01日

Javaに関する上級者向けトピックを扱ったProfessional Java Programming には、Java 2セキュリティ・モデル、国際化対応、パフォーマンス微調整とメモリー管理、印刷、ヘルプ、ドラッグ・アンド・ドロップ操作、切り貼り操作などが説明されています。また、この本には、Swingのより複雑なコンポーネントのうちJTreeおよびJTableに関する章も含まれています。この記事は第6章からの抜粋で、JTableがどのようにセルを描画 (レンダリング) し、セルのさまざまな値をサポートするかを説明しています。さらに、Swingに付属のレンダラー (描画機能) およびエディター (編集機能) の動作を説明し、読者自身のインプリメンテーション時の設定についても検討します。原書の第6章自体には、しばしば必要になる機能 (たとえば行の並べ替え、複数行にわたる列ヘッダーの作成など) を追加するためにJTable機能を拡張する方法も紹介されています。

### セルのレンダリング (描画)

以下の画面ショットでは、一部の列のデータ表示方法が最適ではありません。



First Name	Last Name	Date of Birth	Account ...	Gender
Clay	Ashworth	Tue Feb 20 00:00:00 C...	12345.67	true
Jacob	Ashworth	Tue Jan 06 00:00:00 C...	23456.78	true
Jordan	Ashworth	Thu Aug 31 00:00:00 C...	34567.89	false
Evelyn	Kirk	Tue Jan 16 00:00:00 C...	-456.7	false
Belle	Spyres	Fri Aug 02 00:00:00 CD...	567.0	false

具体的には、3つの点で改善の余地があります。

- ・「誕生日 (Date of Birth)」列には日付と時刻の両方が表示されていますが、日付だけで十分であり、曜日を省略した表示形式にすべきです。
- ・「口座残高 (Account Balance)」には単に数値が表示されているだけですが、通貨単位を使った形式にすべきです。

- 「性別 (Gender)」列は、「男 (Male)」 「女 (Female)」ではなく、何やらわかりにくい "true" とか "false" という値になっています。

JTable のセルはセル・レンダラーによって描画されます。これは、`TableCellRenderer` インターフェースをインプリメントするクラスです。このインターフェースには `getTableCellRendererComponent()` という1つのメソッドが定義され、このメソッドは、描画操作を行う `Component` への参照を戻します。ただし、通常は `TableCellRenderer` をインプリメントしてレンダリング (描画) を行うクラスを1つだけ定義するのが便利ですから、`TableCellRenderer` が自分自身への参照を戻すようにする場合が多いです。 `getTableCellRendererComponent()` に渡されるパラメーターは次のとおりです。

- 描画対象のセルが含まれる JTable への参照
- セルの値の参照
- セルが選択されているかどうかを示す `boolean` フラグ
- セルに入力フォーカスが設定されているかどうかを示す `boolean` フラグ
- 描画対象のセルの行インデックス
- 描画対象のセルの列インデックス

レンダリングを行う `component` (コンポーネント) への参照を戻すのに加えて、`getTableCellRendererComponent()` はそのコンポーネントの状態を初期化します。上記のパラメーターのうち1つは、これからレンダリングされるセルに保管された値を参照していることに注意してください。通常、レンダリングを行うコンポーネントへの参照が戻される前に、そのセルの値を何らかの形式で表したものがコンポーネントに保管されるようにします。

JTable の事前定義レンダラーを使ってデータを正しく表示できることについてはこのあとすぐ説明するとして、ここではまず、カスタム・レンダラー・クラスを定義するのがいかに簡単かを見てください。

## カスタム・レンダラーの作成

以下のクラスはカスタム・レンダラーの例です。このクラスを使って、サンプル・アプリケーションのテーブルの「性別 (Gender)」フィールドの値を表示します。これまで、セルの値に応じて「true」または「false」というテキスト・ストリングが表示されていましたが、このレンダラーを使用すれば、以下のように値が `JComboBox` によって描画されます。

```
import java.awt.Component;
import javax.swing.JComboBox;
import javax.swing.JTable;
import javax.swing.table.TableCellRenderer;

public class GenderRenderer extends JComboBox
    implements TableCellRenderer {

    public GenderRenderer() {
        super();
        addItem("Male");
        addItem("Female");
    }

    public Component getTableCellRendererComponent(JTable table,
        Object value, boolean isSelected, boolean hasFocus, int
row,
        int column) {
```

```

        if (isSelected) {
            setForeground(table.getSelectionForeground());
            super.setBackground(table.getSelectionBackground());
        } else {
            setForeground(table.getForeground());
            setBackground(table.getBackground());
        }

        boolean isMale = ((Boolean) value).booleanValue();
        setSelectedIndex(isMale ? 0 : 1);
        return this;
    }
}

```

このクラスのインスタンスが作成されると、2つの項目、つまり「男性 (Male)」および「女性 (Female)」という選択肢が追加されます。getTableCellRendererComponent() メソッドは、前景および背景の色を簡単な方法で選択したあと、セルの値に基づいて適切な性別を選びます (true は男性、false は女性)。いったんこのレンダラー・クラスを作成したら、SimpleTableTest を次のように変更して、このクラスが「性別 (Gender)」列に使用されるよう指定できます。

```

import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;

public class SimpleTableTest extends JFrame {

    protected JTable table;

    public static void main(String[] args) {
        SimpleTableTest stt = new SimpleTableTest();
        stt.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        stt.setSize(400, 300);
        stt.setVisible(true);
    }

    public SimpleTableTest() {
        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());
        TableValues tv = new TableValues();
        table = new JTable(tv);
        TableColumnModel tcm = table.getColumnModel();
        TableColumn tc = tcm.getColumn(TableValues.GENDER);
        tc.setCellRenderer(new GenderRenderer());
        JScrollPane jsp = new JScrollPane(table);
        pane.add(jsp, BorderLayout.CENTER);
    }
}

```

テーブルのデータは、以下のようなTableValues クラスから取得します (ただし、実際のアプリケーションではデータベースから得る場合が多いでしょう)。

```

import java.util.Calendar;
import java.util.GregorianCalendar;
import javax.swing.table.AbstractTableModel;

public class TableValues extends AbstractTableModel {

    public final static int FIRST_NAME = 0;
    public final static int LAST_NAME = 1;
}

```

```
public final static int DATE_OF_BIRTH = 2;
public final static int ACCOUNT_BALANCE = 3;
public final static int GENDER = 4;

public final static boolean GENDER_MALE = true;
public final static boolean GENDER_FEMALE = false;

public final static String[] columnNames = {
    "First Name", "Last Name", "Date of Birth", "Account Balance", "Gender"
};

public Object[][] values = {
    {
        "Clay", "Ashworth",
        new GregorianCalendar(1962, Calendar.FEBRUARY, 20).getTime(),
        new Float(12345.67), new Boolean(GENDER_MALE)
    }, {
        "Jacob", "Ashworth",
        new GregorianCalendar(1987, Calendar.JANUARY, 6).getTime(),
        new Float(23456.78), new Boolean(GENDER_MALE)
    }, {
        "Jordan", "Ashworth",
        new GregorianCalendar(1989, Calendar.AUGUST, 31).getTime(),
        new Float(34567.89), new Boolean(GENDER_FEMALE)
    }, {
        "Evelyn", "Kirk",
        new GregorianCalendar(1945, Calendar.JANUARY, 16).getTime(),
        new Float(-456.70), new Boolean(GENDER_FEMALE)
    }, {
        "Belle", "Spyres",
        new GregorianCalendar(1907, Calendar.AUGUST, 2).getTime(),
        new Float(567.00), new Boolean(GENDER_FEMALE)
    }
};

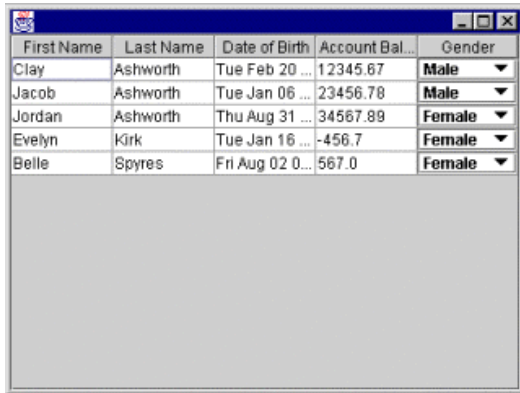
public int getRowCount() {
    return values.length;
}

public int getColumnCount() {
    return values[0].length;
}

public Object getValueAt(int row, int column) {
    return values[row][column];
}

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

アプリケーションの修正バージョンをコンパイルして実行すると、以下のような画面が生成されます。以前Gender (性別) 列に表示されていた「true」および「false」というストリングが、JCheckBox のインスタンスによって置換されたことに注意してください。



First Name	Last Name	Date of Birth	Account Bal...	Gender
Clay	Ashworth	Tue Feb 20 ...	12345.67	Male ▼
Jacob	Ashworth	Tue Jan 06 ...	23456.78	Male ▼
Jordan	Ashworth	Thu Aug 31 ...	34567.89	Female ▼
Evelyn	Kirk	Tue Jan 16 ...	-456.7	Female ▼
Belle	Spyres	Fri Aug 02 0...	567.0	Female ▼

ビジュアルなコンポーネントがContainer に追加されるのと同じようにして、レンダラーが実際にJTable インスタンスに追加されるわけではないことに注意してください。つまり、テーブルにはJCheckBox のインスタンスがいっさい含まれていません。その代わり、テーブルが描かれるとき、それぞれのセルは内容の描画をレンダラーに代行させるわけですが（代行させるために、Graphics オブジェクトがレンダラー・コンポーネントのpaint() メソッドに渡されます）。そして、セルの占める領域と描画領域が一致するように設定されます。言いかえると、この例ではJCheckBox のインスタンスがJTable に追加されたのではなく、Gender 列のそれぞれのセルの領域において、簡単なJCheckBox インスタンスが自らを描画したわけです。この方法は一見して複雑に見えるかもしれませんが、こうすれば、ただ1つのコンポーネントにほとんど（またはすべての）テーブル・セルを描画させることができます。一方、テーブルのそれぞれのセルにコンポーネントを割り当てておくと、ずっと多くのメモリーを消費してしまうでしょう。

通常、カスタム・セル・レンダラーを定義する最も簡単な方法は、Swing のDefaultTableCellRenderer を拡張することです。このクラスは、（その名前が示すように）JTable 内のデフォルト・セル・レンダラーです。DefaultTableCellRenderer はJLabel を拡張したもので、String 表記を使ってセル値を表示します。オブジェクトのString 表記を取得するにはtoString() メソッドを呼び出します。するとDefaultTableCellRenderer は、JLabel から継承したsetText() メソッドにその表記を渡します。この動作は次のようなsetValue() メソッドとしてインプリメントされます。このメソッドには、レンダリング対象セルの値の参照が渡されます。

```
protected void setValue(Object value) {
    setText((value == null) ? "" : value.toString());
}
```

事実上、DefaultTableCellRenderer はレンダリング対象セルの値に基づいて自分自身のテキストを設定するJLabel に過ぎません。

ほとんどの場合、セルの値を得るためにtoString() を呼び出すのは適切な方法ではありません。その1つの例として、今回のサンプル・アプリケーションの「口座残高 (Account Balance)」列を見てください。この列に表示される値は技術的には正しくても、通貨を表した値であることが明白にわかるような形式ではありません。しかし、カスタムTableCellRenderer を作成し、この列のセルの描画をこのレンダラーに実行させれば、問題は簡単に解決できます。

```
import java.text.NumberFormat;
import javax.swing.table.DefaultTableCellRenderer;

public class CurrencyRenderer extends DefaultTableCellRenderer {

    public CurrencyRenderer() {
        super();
        setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    }

    public void setValue(Object value) {
        if ((value != null) && (value instanceof Number)) {
            Number numberValue = (Number) value;
            NumberFormat formatter = NumberFormat.getCurrencyInstance();
            value = formatter.format(numberValue.doubleValue());
        }
        super.setValue(value);
    }
}
```

この簡単なクラスが行う操作は2つだけです。描画中にラベルの水平方向の位置を合わせ、`DefaultTableCellRenderer` に定義されている `setValue()` メソッドをオーバーライドします。このレンダラー・クラスの唯一の用途は数値を含むセルのレンダリングですから、セルの値を `Number` にキャストして、Java の `NumberFormat` クラスを使って値を通貨形式にすることができます。

こうして口座残高列用のカスタム・レンダラーができれば、次は、テーブルが口座残高列のセルを描画するときこのレンダラーを使用するよう設定する必要があります。そうするには、前の例と同じように、レンダラーを明示的に `TableColumn` に割り当てます。ところで、もう1つ方法があります。その方法の方が適切であることが多いですから、それについても紹介しましょう。つまり、特定の列にレンダラーを関連付ける以外に、特定のデータ型にレンダラーを関連付けるという方法もあります。こうすれば、列の中でそのデータ型を含むすべてのセルが、そのレンダラーを使って描画されます。

`JTable` が初期化されるとき、クラスとレンダラーの関連を定義するマップが作成されます。特定のレンダラーが明示的に指定されていない列のセルを描画するときには、そのマップを使ってセル・レンダラーが選択されます。言いかえると、(これまでの例のように) レンダラーを特定の列に明示的に割り当てない場合、`JTable` は、その列に保管されているデータ型に応じてレンダラーを選択します。列のデータ型を判別するには、`TableModel` で `getColumnClass()` メソッドを呼び出します。このメソッドは `Class` のインスタンスを戻します。ただし、`AbstractTableModel` における `getColumnClass()` のインプリメンテーションは、以下のように、`Object` インスタンスがすべての列に含まれることを示すに過ぎません。

```
public Class getColumnClass(int columnIndex) {
    return Object.class;
}
```

サブクラスにどんな種類のデータが入るか `AbstractTableModel` にはわかりませんから、安全な唯一の想定として、各セルに `Object` のインスタンスが入っていると見なします。しかし実際には、セルの中に `Object` のサブクラス (たとえば `Float`、`Date` など) が含まれます。したがって、列に入っている具体的なデータ型をテーブルが判別できるようにするためには、`TableModel` クラスの

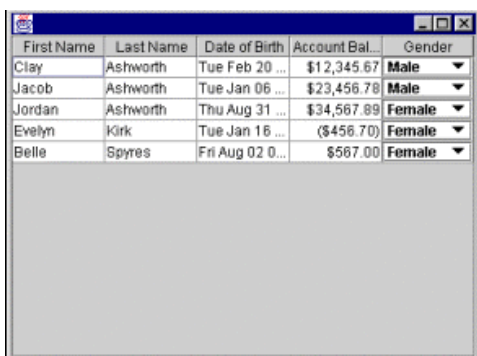
中の `getColumnClass()` をオーバーライドしなければなりません。たとえば、「口座残高 (Account Balance)」列のすべての列は `Float` のインスタンスですから、次のような `getColumnClass()` インプリメンテーションを `TableValues` クラスに追加することができます。

```
public Class getColumnClass(int column) {
    Class dataType = super.getColumnClass(column);
    if (column == ACCOUNT_BALANCE) {
        dataType = Float.class;
    }
    return dataType;
}
```

これで、`JTable` は口座残高列に `Float` データが入っていることを判別できるようになりましたから、このデータ型を `CurrencyRenderer` クラスに関連付ける必要があります。以下のように `setDefaultRenderer()` を呼び出せば簡単に関連付けることができます。

```
public SimpleTableTest() {
    Container pane = getContentPane();
    pane.setLayout(new BorderLayout());
    TableValues tv = new TableValues();
    table = new JTable(tv);
    TableColumnModel tcm = table.getColumnModel();
    TableColumn tc = tcm.getColumn(TableValues.GENDER);
    tc.setCellRenderer(new GenderRenderer());
    table.setDefaultRenderer(Float.class, new CurrencyRenderer());
    JScrollPane jsp = new JScrollPane(table);
    pane.add(jsp, BorderLayout.CENTER);
}
```

こうして `SimpleTableTest` に追加した後は、`CurrencyRenderer` が `Float` データを含むすべての列のデフォルト・レンダラーになります。この結果、口座残高列に特定のレンダラーが割り当てられていないため、また、この列に `Float` データが入っていることを `getColumnClass()` が示すようになったため、口座残高列の描画には `CurrencyRenderer` が使用されます。これらの修正を加えた後、プログラムを実行すると、たとえば次のようなインターフェースが表示されます。



First Name	Last Name	Date of Birth	Account Bal.	Gender
Clay	Ashworth	Tue Feb 20 ...	\$12,345.67	Male
Jacob	Ashworth	Tue Jan 06 ...	\$23,456.78	Male
Jordan	Ashworth	Thu Aug 31 ...	\$34,567.89	Female
Evelyn	Kirk	Tue Jan 16 ...	(\$456.70)	Female
Belle	Spyres	Fri Aug 02 0...	\$567.00	Female

さて、列に特定のレンダラーが明示的に割り当てられず、しかもテーブルのクラス・レンダラー関連マップの中にその列のデータ型を示す項目が存在しない場合には、いったいどうなるのでしょうか。そのとおりです、レンダリングは `DefaultTableCellRenderer` によって処理されます。しかし、どのように処理されるかを正確に理解してください。

列に特定のレンダラーが明示的に割り当てられず、しかもテーブルのクラス・レンダラー関連マップの中にその列の `Class` を示す項目が存在しない場合、`JTable` は列の `Class` の継承階層を走

査して、クラス・レンダラー関連マップの中でそれぞれのスーパークラスに対応する項目を見つけようとしています。たとえば、列にFloat データが入っていることをgetColumnClass() が示すものの、クラス・レンダラー関連マップにFloat の項目が見つからない場合、JTable は、Float の直接のスーパークラス (つまりNumber) のマップ項目を見つけようとしています。Number の項目も見つからない場合、(Number の直接のスーパークラスである)Object の項目を検索しようとしています。マップにはObject 列をDefaultTableCellRenderer に関連付ける項目が自動的に含まれるので、この検索は必ず成功します。

JTable の動作をまとめると、次のような手順でレンダラーを検出します。

- セルのTableColumn にレンダラーが設定されていれば、そのレンダラー。
- TableModel のgetColumnClass() メソッドを呼び出すことによって、Class インスタンスの参照を取得する。
- そのClass にマップされるレンダラーがあれば、そのレンダラーを使用する。
- そのデータ型のスーパークラスのClass インスタンスの参照を取得する。マッチするレンダラーが見つかるまで、前の手順を繰り返す。

テーブル・セルにレンダラーを割り当てるこの方法は、柔軟性の点で非常に優れています。開発者は独自にレンダラーを作成して、そのレンダラーに特定のデータ型 (およびその型のサブクラス) の列を描画させることができるからです。

## JTableのデフォルト・レンダラー

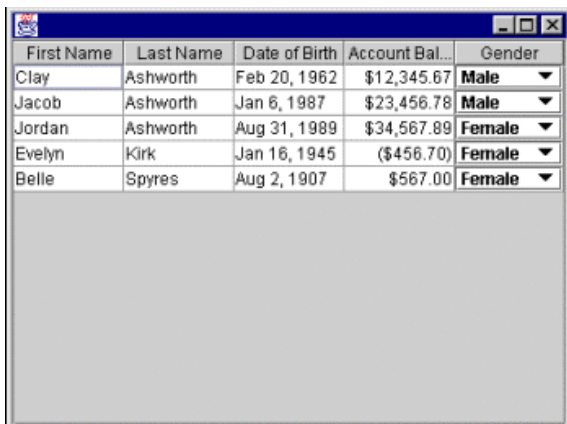
ここまでで、カスタム・レンダラーの作成方法、および特定のデータ型にレンダラーを関連付ける方法について見てきました。しかし、このような操作は実際には必要ないかもしれません。JTableには、よく使われるデータ型用の事前定義レンダラーが多数提供されており、クラス・レンダラー関連マップにはこれらのレンダラー項目が自動的に含まれるのです。

たとえば、Object 列をDefaultTableCellRenderer と関連付ける項目がマップの中に存在することについてはすでに述べましたが、その他にも、洗練されたレンダラーがいくつもあるのです。つまり、いずれかの適切な事前定義レンダラーをアプリケーションで使用するのがふさわしいなら、必要なコーディング作業は、JTable が適切なレンダラーを使用できるように、その列のデータ型をgetColumnClass() のインプリメンテーションで明示することだけです。この点を例示しましょう。誕生日 (Date of Birth) 列にDate インスタンスが含まれることを示すようにTableValues を修正するだけで、JTable のjava.util.Date インスタンス用の事前定義レンダラーを使用できるようになります。

```
public Class getColumnClass(int column) {
    Class dataType = super.getColumnClass(column);
    if (column == ACCOUNT_BALANCE) {
        dataType = Float.class;
    }
    else if (column == DATE_OF_BIRTH) {
        dataType = java.util.Date.class;
    }
    return dataType;
}
```



さきほど見たように、DefaultTableCellRenderer によって描かれる日付値は冗長で、時刻も含んでいました (JavaのDate クラスは日付と時刻の両方を表すからです)。しかしJTable の事前定義の日付レンダラーは、以下のように、より短く適切な形式でそれぞれの日付値を表示します。

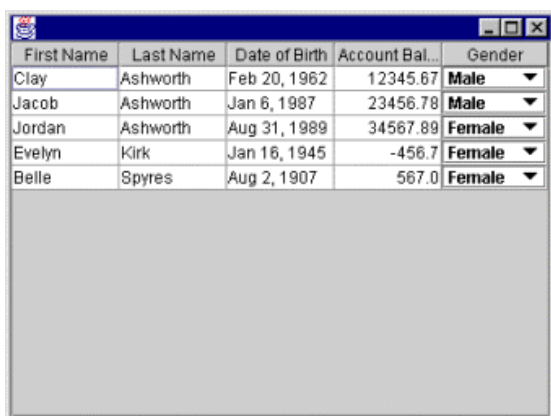


First Name	Last Name	Date of Birth	Account Bal...	Gender
Clay	Ashworth	Feb 20, 1962	\$12,345.67	Male
Jacob	Ashworth	Jan 6, 1987	\$23,456.78	Male
Jordan	Ashworth	Aug 31, 1989	\$34,567.89	Female
Evelyn	Kirk	Jan 16, 1945	(\$456.70)	Female
Belle	Spyres	Aug 2, 1907	\$567.00	Female

java.util.Date の他にも、JTable には、さまざまなクラス用の事前定義レンダラーが含まれています。たとえば次のようなものがあります。

## java.lang.Number

これは数値ラッパー (たとえばInteger、Float、Long など) のスーパークラスです。Number 用に定義されているレンダラーは、(さきほどのCurrencyRenderer の場合のように、位置合わせの値を単にRIGHT に設定するだけの)DefaultTableCellRenderer のサブクラスです。言いかえると、Number レンダラーはセル値のtoString() 表記を表示しますが、テキストをセルの左端に寄せて (デフォルト) ではなく、右端に寄せて表示します。たとえばSampleTableTest クラスの口座残高 (Account Balance) 列でこれを使用した場合、次のようになります。



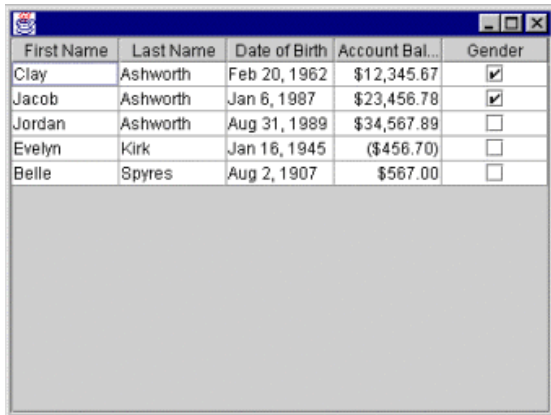
First Name	Last Name	Date of Birth	Account Bal...	Gender
Clay	Ashworth	Feb 20, 1962	12345.67	Male
Jacob	Ashworth	Jan 6, 1987	23456.78	Male
Jordan	Ashworth	Aug 31, 1989	34567.89	Female
Evelyn	Kirk	Jan 16, 1945	-456.7	Female
Belle	Spyres	Aug 2, 1907	567.0	Female

## javax.swing.ImageIcon

このクラスに関連付けられているレンダラーを使用すれば、ImageIcon のインスタンスをテーブルに表示できます。このレンダラーは、JLabel にはテキストとアイコンの両方を入れることができる点を利用したDefaultTableCellRenderer のインスタンスに過ぎません。テキスト値を設定してセルをレンダリングする代わりに、このレンダラーはアイコンを設定します。

## java.lang.Boolean

このレンダラーを使用すると、(セル値がtrueのとき) チェックの付いた、または (値がfalseのとき) チェックがはずされたJCheckBox としてセル値が表示されます。たとえばSimpleTableTest の「性別 (Gender)」列でこれを使用した場合、次のように表示されます。



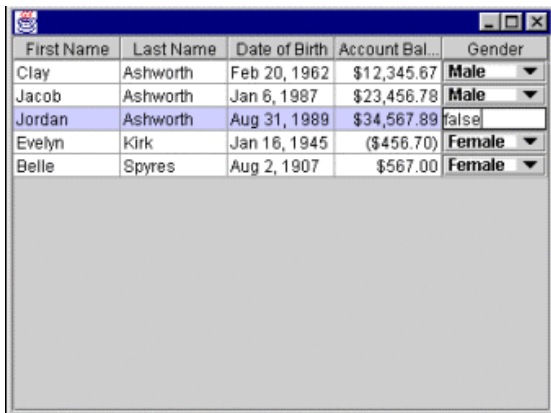
First Name	Last Name	Date of Birth	Account Bal...	Gender
Clay	Ashworth	Feb 20, 1962	\$12,345.67	<input checked="" type="checkbox"/>
Jacob	Ashworth	Jan 6, 1987	\$23,456.78	<input checked="" type="checkbox"/>
Jordan	Ashworth	Aug 31, 1989	\$34,567.89	<input checked="" type="checkbox"/>
Evelyn	Kirk	Jan 16, 1945	(\$456.70)	<input type="checkbox"/>
Belle	Spyres	Aug 2, 1907	\$567.00	<input type="checkbox"/>

## テーブル・セルの編集

「性別 (Gender)」列のそれぞれのセルはJComboBox のように見えますが、選択済みの性別を変更できるわけではありません。実際、テーブルのセルはどれも編集不能であり、セルをクリックしても単に行が選択されるだけです。この動作を変更するには、DefaultTableModel でのインプリメンテーションが常にfalse を戻すため、isCellEditable() メソッドをオーバーライドする必要があります。以下のコードをTableValues に追加するだけで、簡単にこれを変更できるのです。

```
public boolean isCellEditable(int row, int column) {
    if (column == GENDER) {
        return true;
    }
    return false;
}
```

上記は、性別 (Gender) 列のセルが編集可能になったことを示しています。しかし、JComboBox から性別を選択するつもりでこの列のセルをクリックすると、意外にも、クリックした行が選択された状態になるだけです。セルをダブルクリックすると、以下のように、そのセルのブール値 (「true」または「false」) を表すストリングを使って初期化されたJTextField が現れ、テキスト・フィールドが編集可能になります。



First Name	Last Name	Date of Birth	Account Bal.	Gender
Clay	Ashworth	Feb 20, 1962	\$12,345.67	Male
Jacob	Ashworth	Jan 6, 1987	\$23,456.78	Male
Jordan	Ashworth	Aug 31, 1989	\$34,567.89	false
Evelyn	Kirk	Jan 16, 1945	(\$456.70)	Female
Belle	Spyres	Aug 2, 1907	\$567.00	Female

セルにはJComboBox が含まれるにもかかわらず、セル編集時にはテキスト・フィールドが現れることを不思議に思われるかもしれません。しかし、テーブル・セルには実際にどんなコンポーネントも含まれないことに注意してください。セルは単にコンポーネント (レンダラー) によって描画されるだけであり、この場合、そのコンポーネントがたまたまJComboBox であるわけです。しかし編集となるとまったく別の処理であり、レンダリングを行ったのと同じ種類のコンポーネントが処理する場合もあれば、そうでない場合もあります。たとえば、JTable の使用するデフォルト・レンダリング・コンポーネントはJLabel であり、デフォルト編集コンポーネントはJTextField です (上の例でテキスト・フィールドが現れたのはこのためです)。

どんな種類のコンポーネントを使用するかにかかわらず、セルは結局は編集可能になると思われるかもしれません。これは部分的に正しいです。ところが、いずれかのセルに値を入力して編集を終了すると、入力した値が廃棄されるのです。なぜこんなことが起きるのでしょうか。また、どう対処すればいいのでしょうか。それを理解するには、セル・エディターについて、およびJTable がどのようにセルを編集するかについて見ていく必要があります。

## セル・エディター

セル・レンダラーがセル値の描画方法を制御するのと同じように、セル・エディター はセル値の編集を処理します。エディターはレンダラーよりも少しばかり複雑ですが、たとえば以下のように、レンダラーと似ている点もたくさんあります。

- 1つのエディターは、1つまたは複数のTableColumn インスタンスに関連付けることができます。
- 1つのエディターは1つまたは複数のデータ型 (クラス) に関連付けることができ、セルが属する列に他のエディターが関連付けられていない場合、そのデータ型を表示するためにこのエディターを使用できる。
- レンダラーがセル値を描画するために既存のビジュアル・コンポーネントを使うのと同じように、既存のビジュアル・コンポーネントを使って編集機能を提供することができる。実際、セル・レンダラーとして使われるビジュアル・コンポーネントと同じ種類のものが、エディターとしても使われる場合が多いです。たとえば、あるセルにJComboBox を使用するレンダラーを割り当てると同時に、同じコンポーネントを使用するエディターを割り当てることもできます。

1つのエディターを1つまたは複数のTableColumn インスタンスまたはオブジェクト・タイプに割り当てることができます (TableColumn の中でsetCellEditor() メソッドを、またはJTable の中

で `setDefaultEditor()` をそれぞれ使用する)。ただし `TableCellEditor` インターフェースのインプリメンテーションは `TableCellRenderer` よりも複雑であり、`TableCellEditor` に定義されたメソッドを理解するために、エディターが `JTable` インスタンスと対話する方法を調べる必要があるでしょう。

`JTable` はいずれかのセルがマウス・クリックされたことを検出すると、`TableModel` 内の `isCellEditable()` メソッドを呼び出します。このメソッドは、セルが編集不能であれば `false` の値を返します。その場合、処理が終了して、それ以上のアクションは行われません。しかし、このメソッドが `true` を返した場合、テーブルはそのセル用のセル・エディターを識別するとともに、`CellEditor` の `isCellEditable()` メソッドを呼び出します。

`TableModel` と `CellEditor` はどちらも `isCellEditable()` という名前のメソッドを定義していますが、両者には重要な違いがあります。`TableModel` メソッドには行と列のインデックス値が渡されるだけですが、`CellEditor` メソッドには、マウス・クリックを表す `EventObject` も渡されます。これを使って、たとえば、イベントの中に保管されたクリック・カウントをチェックできます。「性別 (Gender)」列値を編集する例ですで見たとおり、セルを編集する前には、そのセルをダブルクリックする必要があります。言いかえると、`isCellEditable()` メソッドはクリック・カウントが1であれば値 `false` を返し、カウントが1より大であれば `true` を返します。この動作によって、セル・エディターはセルの選択要求 (シングルクリック) とセルの編集要求 (ダブルクリック) を区別することができます。

編集操作が許可されるのは、`TableModel` の `isCellEditable()` メソッドと `CellEditor` の `isCellEditable()` メソッドがどちらも `true` の値を返す場合に限られます。両者とも `true` の場合、編集操作は `getTableCellEditorComponent()` メソッドを呼び出すことによって始まります。このメソッドには以下のようなパラメーターを渡します。

- 編集対象のセルが含まれる `JTable` への参照
- セルの現在の値の参照
- セルを選択された状態にするかどうかを示す `boolean` フラグ
- 編集対象のセルの行インデックス
- 編集対象のセルの列インデックス

これらのパラメーターに見覚えがありますか。そうです、`TableCellRenderer` の `getTableCellRendererComponent()` メソッドに渡されたパラメーターとほぼ同じです。唯一の違いは、セルに入力フォーカスを設定するかどうかを示す `boolean` 値がこのメソッドには渡されない点です。セルが編集されるわけですから、暗黙的に入力フォーカスを示唆するのです。

編集処理を担当するコンポーネントへの参照を返す前に、`getTableCellEditorComponent()` は、エディターの値がセルの現在の値と一致するように、値を適切に初期化する必要があります。たとえば、`TableValues` の「性別 (Gender)」列を表す `JComboBox` から、ユーザーが「男性 (Male)」または「女性 (Female)」のどちらかを選択できるようなエディターを作成していきましょう。この場合、編集を行う `JComboBox` では、セルの性別値に対応する項目をあらかじめ選択しておく必要があります (値が `true` であれば「男性 (Male)」、`false` であれば「女性 (Female)」)。

編集を行うコンポーネントの準備が整って `getTableCellEditorComponent()` メソッドから戻されたら、`JTable` はそのコンポーネントのサイズと場所を設定して、コンポーネントが編集対象セル

とぴったり重なって表示されるようにします。こうして、セルがその場所で編集されているように見えます。実際には、編集をサポートするコンポーネント (たとえば `JTextField` や、この例の場合は `JComboBox`) がセルの上に重なっているだけです。

編集を行うコンポーネントを編集対象セルの上に置くことによって、最初に編集処理をトリガーしたイベントは、編集を行うコンポーネントに配置されます。たとえば、`JComboBox` に基づくエディターの場合、編集を起動したのと同じマウス・イベントが今度はコンボ・ボックスに渡されるようになり、それによって、編集開始時にたとえばドロップダウン・メニューが表示されます。最後に `CellEditor` の `shouldSelectCell()` メソッドに同じマウス・イベント・オブジェクトが渡され、`true` を戻した場合には、セル (および、テーブルの選択設定に応じて他のもの) が選択されます。

それぞれの `CellEditor` には `addCellEditorListener()` メソッドと `removeCellEditorListener()` メソッドをインプリメントする必要があります。 `CellEditorListener` インターフェースは `editingStopped()` および `editingCanceled()` という2つのメソッドを定義します。事実上、唯一のリスナーは `JTable` そのものであることが多く、編集が中止またはキャンセルされたとき、これに対して通知されます。さらに、`CellEditor` は `cancelCellEditing()` メソッドと `stopCellEditing()` メソッドをインプリメントする必要があります。これらのメソッドは、登録されたリスナーの `editingStopped()` メソッドおよび `editingCanceled()` メソッドを呼び出します。

編集の終了要求は、セルを含んでいる `JTable` から、またはエディター・コンポーネントそのものから出すことができます。たとえば、ユーザーがあるセルをクリックして、その値を編集し始めたとします。このとき別のセルをクリックすると、`JTable` はこの2番目のセルの編集を開始する前に、最初のセルの `stopCellEditing()` メソッドを呼び出します。別の方法として、編集が完了したことを示唆する何らかのイベントが発生したときに、エディター・コンポーネントに編集を停止させることもできます。たとえば、エディターとして `JComboBox` を使う場合、何かが選択されたことを示す `ActionEvent` メッセージを受け取ったら、編集を終了するのが適切です。同様に `JTextField` では、Return キーが押されたことを検出すると、編集終了を通知することができます。

登録された `CellEditorListener` は `JTable` の `editingStopped()` メソッドであるため、編集終了要求がどこから出るかにかかわらず、このメソッドが呼び出されます。このメソッドの内部では、セルの新しい値を検索するためにテーブルがエディターの `getCellEditorValue()` メソッドを呼び出して、その値を `JTable` の `TableModel` 内の `setValueAt()` メソッドに渡します。つまり、セルの新しい値をエディターから取得してデータ・モデルに送ることによって、その値が「永久的に」保管されるようにするわけです。

以下のクラスは、`TableValues` の「性別 (Gender)」列の行を編集可能にするコンポーネントを定義します。このクラスが定義する `JComboBox` のサブクラスは、男性 (Male) 項目または女性 (Female) 項目を使って自身を初期化し、状態が変更されるのを `listen` します (選択が行われるのを待ちます)。

「性別 (Gender)」列のいずれかのセルの編集が始まると、`getTableCellEditorComponent()` メソッドが呼び出され、セルが表示される前にエディターはセルの状態を初期化できます。ここで

は、エディターは単に、編集対象セルに保管された値に基づいて「男性 (Male)」または「女性 (Female)」のいずれかの項目を選択するだけです。ユーザーがJComboBox の中で1つの項目を選択すると、編集セッションの終了をテーブルに通知する`fireEditingStopped()` が呼び出されます。その後、テーブルは、セルに保管すべき新しい値を取得するために`getCellEditorValue()` を呼び出して、その値をTableModel の`setValueAt()` メソッドに渡します。

```
import java.awt.Component;
import java.util.EventObject;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

public class GenderEditor extends JComboBox implements TableCellEditor {

    protected EventListenerList listenerList = new EventListenerList();
    protected ChangeEvent changeEvent = new ChangeEvent(this);

    public GenderEditor() {
        super();
        addItem("Male");
        addItem("Female");
        addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                fireEditingStopped();
            }
        });
    }

    public void addCellEditorListener(CellEditorListener listener) {
        listenerList.add(CellEditorListener.class, listener);
    }

    public void removeCellEditorListener(CellEditorListener listener) {
        listenerList.remove(CellEditorListener.class, listener);
    }

    protected void fireEditingStopped() {
        CellEditorListener listener;
        Object[] listeners = listenerList.getListenerList();
        for (int i = 0; i < listeners.length; i++) {
            if (listeners[i] == CellEditorListener.class) {
                listener = (CellEditorListener) listeners[i + 1];
                listener.editingStopped(changeEvent);
            }
        }
    }

    protected void fireEditingCanceled() {
        CellEditorListener listener;
        Object[] listeners = listenerList.getListenerList();
        for (int i = 0; i < listeners.length; i++) {
            if (listeners[i] == CellEditorListener.class) {
                listener = (CellEditorListener) listeners[i + 1];
                listener.editingCanceled(changeEvent);
            }
        }
    }

    public void cancelCellEditing() {
        fireEditingCanceled();
    }

    public boolean stopCellEditing() {
```

```

        fireEditingStopped();
        return true;
    }

    public boolean isCellEditable(EventObject event) {
        return true;
    }

    public boolean shouldSelectCell(EventObject event) {
        return true;
    }

    public Object getCellEditorValue() {
        return new Boolean(getSelectedIndex() == 0 ? true : false);
    }

    public Component getTableCellEditorComponent(JTable table, Object
value,
        boolean isSelected, int row, int column) {
        boolean isMale = ((Boolean) value).booleanValue();
        setSelectedIndex(isMale ? 0 : 1);
        return this;
    }
}

```

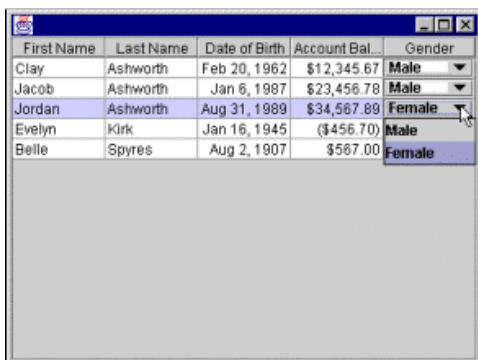
こうしてエディター・コンポーネントの定義が終わったら、以下のコードに示されているように、性別 (Gender) 列にこれを関連付ける必要があります。

```

public SimpleTableTest() {
    Container pane = getContentPane();
    pane.setLayout(new BorderLayout());
    TableValues tv = new TableValues();
    table = new JTable(tv);
    TableColumnModel tcm = table.getColumnModel();
    TableColumn tc = tcm.getColumn(TableValues.GENDER);
    tc.setCellRenderer(new GenderRenderer());
    tc.setCellEditor(new GenderEditor());
    table.setDefaultRenderer(Float.class, new CurrencyRenderer());
    JScrollPane jsp = new JScrollPane(table);
    pane.add(jsp, BorderLayout.CENTER);
}

```

このコードをコンパイルして実行すると、JComboBox が正しく表示され、適切な性別値を使って初期化されて、「男性 (Male)」または「女性 (Female)」のいずれかを選択できるようになります。



First Name	Last Name	Date of Birth	Account Bal.	Gender
Clay	Ashworth	Feb 20, 1962	\$12,345.67	Male
Jacob	Ashworth	Jan 6, 1987	\$23,456.78	Male
Jordan	Ashworth	Aug 31, 1989	\$34,567.89	Female
Evelyn	Kirk	Jan 16, 1945	(\$456.70)	Male
Belle	Spyres	Aug 2, 1907	\$567.00	Female

ただし、セルに保管された値とは違う値を選択しても、セルの値が変更されるわけではありません。なぜなら、`TableModel` の値が変わらないからです。この値を変更するには、以下のように、`TableValues` クラスで `setValueAt()` メソッドをインプリメントします。

```
public void setValueAt(Object value, int row, int column) {  
    values[row][column] = value;  
}
```

## DefaultCellEditor

まったく新しいエディターをいつも作成しなければならないとは限りません。実際、`DefaultCellEditor` クラスを利用すれば、`JCheckBox`、`JComboBox`、または `JTextField` を使ったエディター・コンポーネントを簡単に作成できます。ただ `DefaultCellEditor` のインスタンスを作って、それに3つのコンポーネントのいずれかのインスタンスを渡すだけでよいのです。ただし `DefaultCellEditor` はあまり柔軟ではないので、今回の例のように、独自のエディターを作成しなければならないことも多いでしょう。



## 関連トピック

- この本に関する情報を調べて注文するには、[Amazon.com](https://www.amazon.com) をご覧ください。
- すべてのWroxタイトルを[www.wrox.com](https://www.wrox.com) でご覧ください。

© Copyright IBM Corporation 2000

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

商標

([www.ibm.com/developerworks/jp/ibm/trademarks/](http://www.ibm.com/developerworks/jp/ibm/trademarks/))