```java
package pv_unidade_05_ex02_tablemodelpersonalizado;

import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.table.AbstractTableModel;

/**
 * @author Edwar Saliba Júnior
 */
public class NewTableModel extends AbstractTableModel {

    // As linhas serão definidas por um ArrayList;
    private ArrayList linhas = null;
    // O título das colunas será definido por um vetor de strings.
    private String[] colunas = null;

    public NewTableModel(ArrayList linhas, String[] colunas) {
        this.linhas = linhas;
        this.colunas = colunas;
    }

    public String[] getColunas() {
        return colunas;
    }

    public ArrayList getLinhas() {
        return linhas;
    }

    public void setColunas(String[] strings) {
        colunas = strings;
    }

    public void setLinhas(ArrayList list) {
        linhas = list;
    }

    @Override
    public int getRowCount() {
        return linhas.size();
    }

    @Override
    public int getColumnCount() {
        return colunas.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        // Obtém a linha, que é um vetor de Strings.
        Object[] linha = (Object[]) linhas.get(rowIndex);
        // Retorna o valor que está na coluna.
        return linha[columnIndex];
    }

    /**
     * Este método serve apenas para que as células do nosso JTable sejam
     * editáveis com o clique duplo do mouse. Quando se usa o DefaultTableModel
     * as células já vêm editáveis por padrão. No entanto quando se cria seu
     * prórpio TableModel, isto não acontece. Então sobrescrevemos o método
     * isCellEditable fazendo-o retornar sempre true, então, todas as células
     * da tabela passam a ser editáveis, como no DefaultTableModel.
     */
    @Override
    public boolean isCellEditable(int row, int column) {
        return true;
    }

    @Override
    public void setValueAt(Object value, int rowIndex, int colIndex) {
        // Obtém a linha, que é um vetor de strings.
        Object[] linha = (Object[]) linhas.get(rowIndex);
        // Altera o conteúdo no índice da coluna passado como parâmetro.
        linha[colIndex] = (String) value;
        // Dispara o evento de célula alterada.
        fireTableCellUpdated(rowIndex, colIndex);
    }
```

```
    public void addRow(String[] dadosDaLinha) {
        linhas.add(dadosDaLinha);
        // Informa ao JTable de que houve linhas incluídas no modelo.
        int linha = linhas.size() - 1;
        fireTableRowsInserted(linha, linha);
    }

    public void removeRow(int rowIndex) {
        linhas.remove(rowIndex);
        // Informa ao JTable que houve dados apagados e passa o índice da linha.
        fireTableRowsDeleted(rowIndex, rowIndex);
    }

    /**
     * Remove a linha pelo valor na célula da coluna informada.
     *
     * @param valor - valor a ser procurado.
     * @param colIndex - índice da coluna.
     * @return true caso tenha encontrado a linha e false caso contrário.
     */
    public boolean removeRow(String valor, int colIndex) {
        boolean operacaoBemSucedida = false;
        Iterator i = linhas.iterator();
        int linha = 0;

        // Percorre as linhas existentes.
        while (i.hasNext()) {
            // Obtém as colunas de uma linha.
            String[] linhaCorrente = (String[]) i.next();
            linha++;

            // Compara o conteúdo da linha atual na coluna desejada, com o valor informado em "valor".
            if (linhaCorrente[colIndex].trim().equals(valor.trim())) {
                linhas.remove(linha);
                // Informa ao JTable que uma linha foi apagada e passa o índice dela.
                fireTableRowsDeleted(linha, linha);
                operacaoBemSucedida = true;
            }
        }

        return operacaoBemSucedida;
    }

    /**
     * Retorna o nome da coluna.
     */
    @Override
    public String getColumnName(int colIndex) {
        return colunas[colIndex];
    }
}
```