

Commit


 This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

Refatoração em gui

Browse files

ButtonColumn - Técnicas: Extract Method

LoginFrame - Técnicas: Extract Method

 moisesocosta committed on Jun 21

1 parent 9d330b1 commit c013bbd

Showing 2 changed files with 167 additions and 265 deletions.

Whitespace

Ignore whitespace

Split

Unified

338

src/br/sistema/crud/jdbc/gui/ButtonColumn.java

```
...      @@ -1,227 +1,127 @@
1      - package br.sistema.crud.jdbc.gui;
2      -
3      - import java.awt.*;
4      1      import java.awt.Color;
5      2      import java.awt.Component;
6      - import java.awt.event.*;
7      - import javax.swing.*;
8      - import javax.swing.border.*;
9      - import javax.swing.table.*;
10     -
11     - /**
12     -  * The ButtonColumn class provides a renderer and an editor that looks like a
13     -  * JButton. The renderer and editor will then be used for a specified column
14     -  * in the table. The TableModel will contain the String to be displayed on
15     -  * the button.
16     -  *
17     -  * The button can be invoked by a mouse click or by pressing the space bar
18     -  * when the cell has focus. Optionally a mnemonic can be set to invoke the
19     -  * button. When the button is invoked the provided Action is invoked. The
20     -  * source of the Action will be the table. The action command will contain
```

```
21 - * the model row number of the button that was clicked.
22 - *
23 - */
24 - public class ButtonColumn extends AbstractCellEditor
25 -     implements TableCellRenderer, TableCellEditor, ActionListener,
        MouseListener
26 - {
27 -     private JTable table;
28 -     private Action action;
29 -     private int mnemonic;
30 -     private Border originalBorder;
31 -     private Border focusBorder;
32 -
33 -     private JButton renderButton;
34 -     private JButton editButton;
35 -     private Object editorValue;
36 -     private boolean isButtonColumnEditor;
37 -
38 -     /**
39 -      * Create the ButtonColumn to be used as a renderer and editor. The
40 -      * renderer and editor will automatically be installed on the TableColumn
41 -      * of the specified column.
42 -      *
43 -      * @param table the table containing the button renderer/editor
44 -      * @param action the Action to be invoked when the button is invoked
45 -      * @param column the column to which the button renderer/editor is added
46 -      */
47 -     public ButtonColumn(JTable table, Action action, int column)
48 -     {
49 -         this.table = table;
50 -         this.action = action;
51 -
52 -         renderButton = new JButton();
53 -         editButton = new JButton();
54 -         editButton.setFocusPainted( false );
55 -         editButton.addActionListener( this );
56 -         originalBorder = editButton.getBorder();
57 -         setFocusBorder( new LineBorder(Color.BLUE) );
58 -
59 -         TableColumnModel columnModel = table.getColumnModel();
60 -         columnModel.getColumn(column).setCellRenderer( this );
61 -         columnModel.getColumn(column).setCellEditor( this );
62 -         table.addMouseListener( this );
63 -     }
64 -
65 -
66 -     /**
67 -      * Get foreground color of the button when the cell has focus
68 -      *
69 -      * @return the foreground color
70 -      */
71 -     public Border getFocusBorder()
```

```
72     {
73         return focusBorder;
74     }
75
76     /**
77      * The foreground color of the button when the cell has focus
78      *
79      * @param focusBorder the foreground color
80      */
81     public void setFocusBorder(Border focusBorder)
82     {
83         this.focusBorder = focusBorder;
84         editButton.setBorder( focusBorder );
85     }
86
87     public int getMnemonic()
88     {
89         return mnemonic;
90     }
91
92     /**
93      * The mnemonic to activate the button when the cell has focus
94      *
95      * @param mnemonic the mnemonic
96      */
97     public void setMnemonic(int mnemonic)
98     {
99         this.mnemonic = mnemonic;
100        renderButton.setMnemonic(mnemonic);
101        editButton.setMnemonic(mnemonic);
102    }
103
104    @Override
105    public Component getTableCellEditorComponent(
106        JTable table, Object value, boolean isSelected, int row, int
107        column)
108    {
109        if (value == null)
110        {
111            editButton.setText( "" );
112            editButton.setIcon( null );
113        }
114        else if (value instanceof Icon)
115        {
116            editButton.setText( "" );
117            editButton.setIcon( (Icon)value );
118        }
119        else
120        {
121            editButton.setText( value.toString() );
122            editButton.setIcon( null );
123        }
124    }
```

```
123 -
124 -         this.editorValue = value;
125 -         return editButton;
126 -     }
127 -
128 -     @Override
129 -     public Object getCellEditorValue()
130 -     {
131 -         return editorValue;
132 -     }
133 -
134 - //
135 - // Implement TableCellRenderer interface
136 - //
137 -     public Component getTableCellRendererComponent(
138 -         JTable table, Object value, boolean isSelected, boolean hasFocus,
139 -         int row, int column)
140 -     {
141 -         if (isSelected)
142 -         {
143 -             renderButton.setForeground(table.getSelectionForeground());
144 -             renderButton.setBackground(table.getSelectionBackground());
145 -         }
146 -         else
147 -         {
148 -             renderButton.setForeground(table.getForeground());
149 -
150 -             renderButton.setBackground(UIManager.getColor("Button.background"));
151 -
152 -             if (hasFocus)
153 -             {
154 -                 renderButton.setBorder( focusBorder );
155 -             }
156 -             else
157 -             {
158 -                 renderButton.setBorder( originalBorder );
159 -             }
160 - //         renderButton.setText( (value == null) ? "" : value.toString() );
161 -         if (value == null)
162 -         {
163 -             renderButton.setText( "" );
164 -             renderButton.setIcon( null );
165 -         }
166 -         else if (value instanceof Icon)
167 -         {
168 -             renderButton.setText( "" );
169 -             renderButton.setIcon( (Icon)value );
170 -         }
171 -         else
172 -         {
```

```

173         renderButton.setText( value.toString() );
174         renderButton.setIcon( null );
175     }
176
177     return renderButton;
178 }
179
180 //
181 // Implement ActionListener interface
182 //
183 /*
184  *      The button has been pressed. Stop editing and invoke the custom
    Action
185  */
186 public void actionPerformed(ActionEvent e)
187 {
188     int row = table.convertRowIndexToModel( table.getEditingRow() );
189     fireEditingStopped();
190
191     // Invoke the Action
192
193     ActionEvent event = new ActionEvent(
194         table,
195         ActionEvent.ACTION_PERFORMED,
196         "" + row);
197     action.actionPerformed(event);
198 }
199
200 //
201 // Implement MouseListener interface
202 //
203 /*
204  *      When the mouse is pressed the editor is invoked. If you then then drag
205  *      the mouse to another cell before releasing it, the editor is still
206  *      active. Make sure editing is stopped when the mouse is released.
207  */
208 public void mousePressed(MouseEvent e)
209 {
210     if (table.isEditing()
211         && table.getCellEditor() == this)
212         isButtonColumnEditor = true;
213
214 + import java.awt.event.ActionEvent;
215 + import java.awt.event.ActionListener;
216 + import java.awt.event.MouseEvent;
217 + import java.awt.event.MouseListener;
218 +
219 + import javax.swing.AbstractCellEditor;
220 + import javax.swing.JButton;
221 + import javax.swing.JTable;
222 + import javax.swing.UIManager;
223 + import javax.swing.border.Border;
224 + import javax.swing.border.LineBorder;

```

```
14 + import javax.swing.table.TableCellEditor;
15 + import javax.swing.table.TableCellRenderer;
16 + import javax.swing.table.TableColumnModel;
17 +
18 + public class ButtonColumn extends AbstractCellEditor implements TableCellRenderer,
    TableCellEditor, ActionListener, MouseListener {
19 +     private JTable table;
20 +     private JButton renderButton;
21 +     private JButton editButton;
22 +     private Object editorValue;
23 +     private boolean isButtonColumnEditor;
24 +
25 +     public ButtonColumn(JTable table, int column) {
26 +         this.table = table;
27 +         renderButton = new JButton();
28 +         editButton = new JButton();
29 +         editButton.setFocusPainted(false);
30 +         editButton.addActionListener(this);
31 +         Border originalBorder = editButton.getBorder();
32 +         setFocusBorder(new LineBorder(Color.BLUE));
33 +
34 +         TableColumnModel columnModel = table.getColumnModel();
35 +         columnModel.getColumn(column).setCellRenderer(this);
36 +         columnModel.getColumn(column).setCellEditor(this);
37 +         table.addMouseListener(this);
38 +     }
39 +
40 +     public Border getFocusBorder() {
41 +         return editButton.getBorder();
42 +     }
43 +
44 +     public void setFocusBorder(Border focusBorder) {
45 +         editButton.setBorder(focusBorder);
46 +     }
47 +
48 +     @Override
49 +     public Component getTableCellEditorComponent(JTable table, Object value,
        boolean isSelected, int row, int column) {
50 +         if (value == null) {
51 +             editButton.setText("");
52 +             editButton.setIcon(null);
53 +         } else if (value instanceof Icon) {
54 +             editButton.setText("");
55 +             editButton.setIcon((Icon) value);
56 +         } else {
57 +             editButton.setText(value.toString());
58 +             editButton.setIcon(null);
59 +         }
60 +
61 +         this.editorValue = value;
62 +         return editButton;
63 +     }
```

```
64 +
65 + @Override
66 + public Object getCellEditorValue() {
67 +     return editorValue;
68 + }
69 +
70 + @Override
71 + public Component getTableCellRendererComponent(JTable table, Object value,
    boolean isSelected, boolean hasFocus, int row, int column) {
72 +     if (isSelected) {
73 +         renderButton.setForeground(table.getSelectionForeground());
74 +         renderButton.setBackground(table.getSelectionBackground());
75 +     } else {
76 +         renderButton.setForeground(table.getForeground());
77 +         renderButton.setBackground(UIManager.getColor("Button.background"));
78 +     }
79 +
80 +     if (hasFocus) {
81 +         renderButton.setBorder(editButton.getBorder());
82 +     } else {
83 +         renderButton.setBorder(editButton.getBorder());
84 +     }
85 +
86 +     if (value == null) {
87 +         renderButton.setText("");
88 +         renderButton.setIcon(null);
89 +     } else if (value instanceof Icon) {
90 +         renderButton.setText("");
91 +         renderButton.setIcon((Icon) value);
92 +     } else {
93 +         renderButton.setText(value.toString());
94 +         renderButton.setIcon(null);
95 +     }
96 +
97 +     return renderButton;
98 + }
99 +
100 + @Override
101 + public void actionPerformed(ActionEvent e) {
102 +     int row = table.convertRowIndexToModel(table.getEditingRow());
103 +     fireEditingStopped();
104 +
105 +     ActionEvent event = new ActionEvent(table, ActionEvent.ACTION_PERFORMED, ""
    + row);
106 +     Action action = (Action) e.getSource();
107 +     action.actionPerformed(event);
108 + }
109 +
110 + @Override
111 + public void mousePressed(MouseEvent e) {
112 +     if (table.isEditing() && table.getCellEditor() == this) {
113 +         isButtonColumnEditor = true;
```

```

114 +     }
213 115     }
214 116
215 -     public void mouseReleased(MouseEvent e)
216 -     {
217 -         if (isButtonColumnEditor
218 -         && table.isEditing())
219 -         table.getCellEditor().stopCellEditing();
117 +     @Override
118 +     public void mouseReleased(MouseEvent e) {
119 +         if (isButtonColumnEditor && table.isEditing()) {
120 +         table.getCellEditor().stopCellEditing();
121 +         }
220 122
221 -         isButtonColumnEditor = false;
123 +         isButtonColumnEditor = false;
222 124     }
223 125
224 -     public void mouseClicked(MouseEvent e) {}
225 -     public void mouseEntered(MouseEvent e) {}
226 -     public void mouseExited(MouseEvent e) {}
126 +     // Implementação dos outros métodos do MouseListener
227 127 }

```

▼ 94 src/br/sistema/crud/jdbc/gui/LoginFrame.java

```

...    ...    @@ -1,6 +1,5 @@
1      1      package br.sistema.crud.jdbc.gui;
2      2
3      - import java.awt.BorderLayout;
4      3      import java.awt.EventQueue;
5      4
6      5      import javax.swing.JFrame;
15     14      import javax.swing.border.TitledBorder;
16     15
17     16      import br.sistema.crud.jdbc.bo.LoginBO;
18     - import br.sistema.crud.jdbc.dao.LoginDAO;
19     17      import br.sistema.crud.jdbc.dto.LoginDTO;
20     18      import br.sistema.crud.jdbc.util.MensagensUtil;
21     19
29     27      private JTextField txtLogin;
30     28      private JPasswordField passSenha;
31     29
32     - /**
33     -  * Launch the application.
34     -  */
35     30      public static void main(String[] args) {
36     31          EventQueue.invokeLater(new Runnable() {
37     32              public void run() {
45     40              });
46     41      }
47     42

```



```

48  -      /**
49  -      * Create the frame.
50  -      */
51  43      public LoginFrame() {
52      44          initializeUI();
53      45          setupListeners();
54      46      }
55      47      +
56      48      +      private void initializeUI() {
57      49          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
58      50          setBounds(100, 100, 359, 195);
59      51          contentPane = new JPanel();
60      52          contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
61      53          setContentPane(contentPane);
62      54
63      -      //BOTA O DE LOGA
64      55      JButton btnLogar = new JButton("Logar");
65      -      btnLogar.addActionListener(new ActionListener() {
66      -          public void actionPerformed(ActionEvent evt) {
67      -
68      -              LoginDTO loginDTO = new LoginDTO();
69      -              loginDTO.setNome(txtLogin.getText());
70      -              //TRANSFORMANDO O GETPASSWORD QUE E CAEACTER PRA
71      -              STRING
72      -              loginDTO.setSenha(new String
73      -              (passSenha.getPassword()));
74      -
75      -              LoginBO loginBO = new LoginBO();
76      -              try{
77      -
78      -                  if(loginBO.logar(loginDTO)){
79      -                      LoginFrame.this.dispose();
80      -                      MainFrame m =new MainFrame();
81      -                      m.setLocationRelativeTo(null);
82      -                      m.setVisible(true);
83      -                  }else{
84      -
85      -                      MensagensUtil.addMsg(LoginFrame.this, "Dados Invalidos");
86      -                  }
87      -
88      -                  }catch(Exception e){
89      -                      e.printStackTrace();
90      -                      MensagensUtil.addMsg(LoginFrame.this,
91      -                      e.getMessage());
92      -                  }
93      -              }
94      -          });
95      -
96      56      JButton btnSair = new JButton("Sair");
97      -      btnSair.addActionListener(new ActionListener() {

```

```

91         -         public void actionPerformed(ActionEvent e) {
92         -             System.exit(0);
93         -         }
94         -     });
95     57
96     58         JPanel panelLogin = new JPanel();
97     59         panelLogin.setBorder(new TitledBorder(null, "",
TitledBorder.LEADING, TitledBorder.TOP, null, null));
98     60 +
99     61         GroupLayout gl_contentPane = new GroupLayout(contentPane);
100    62         gl_contentPane.setHorizontalGroup(
101    63             gl_contentPane.createParallelGroup(Alignment.LEADING)
102    83         );
103    84
104    85         JLabel lblLogin = new JLabel("Login");
105    86
106    87         JLabel lblSenha = new JLabel("Senha");
107    88
108    89         txtLogin = new JTextField();
109    90         txtLogin.setColumns(10);
110    91
111    92         passSenha = new JPasswordField();
112    93
113    94         GroupLayout gl_panelLogin = new GroupLayout(panelLogin);
114    95         gl_panelLogin.setHorizontalGroup(
115    96             gl_panelLogin.createParallelGroup(Alignment.LEADING)
116    120         panelLogin.setLayout(gl_panelLogin);
117    121         contentPane.setLayout(gl_contentPane);
118    122     }
119    123 +
120    124 +     private void setupListeners() {
121    125 +         JButton btnLogar = new JButton("Logar");
122    126 +         btnLogar.addActionListener(new ActionListener() {
123    127 +             public void actionPerformed(ActionEvent evt) {
124    128 +                 logar();
125    129 +             }
126    130 +         });
127    131 +
128    132 +         JButton btnSair = new JButton("Sair");
129    133 +         btnSair.addActionListener(new ActionListener() {
130    134 +             public void actionPerformed(ActionEvent e) {
131    135 +                 sair();
132    136 +             }
133    137 +         });
134    138 +     }
135    139 +
136    140 +     private void logar() {
137    141 +         LoginDTO loginDTO = new LoginDTO();
138    142 +         loginDTO.setNome(txtLogin.getText());
139    143 +         loginDTO.setSenha(new String(passSenha.getPassword()));
140    144 +
141    145 +         LoginBO loginBO = new LoginBO();

```

```
146 +         try {
147 +             if (loginBO.logar(loginDTO)) {
148 +                 dispose();
149 +                 MainFrame m = new MainFrame();
150 +                 m.setLocationRelativeTo(null);
151 +                 m.setVisible(true);
152 +             } else {
153 +                 MensagensUtil.addMsg(this, "Dados Inválidos");
154 +             }
155 +         } catch (Exception e) {
156 +             e.printStackTrace();
157 +             MensagensUtil.addMsg(this, e.getMessage());
158 +         }
159 +     }
160 +
161 +     private void sair() {
162 +         System.exit(0);
163 +     }
164 }
```

0 comments on commit `c013bbd`Please [sign in](#) to comment.