

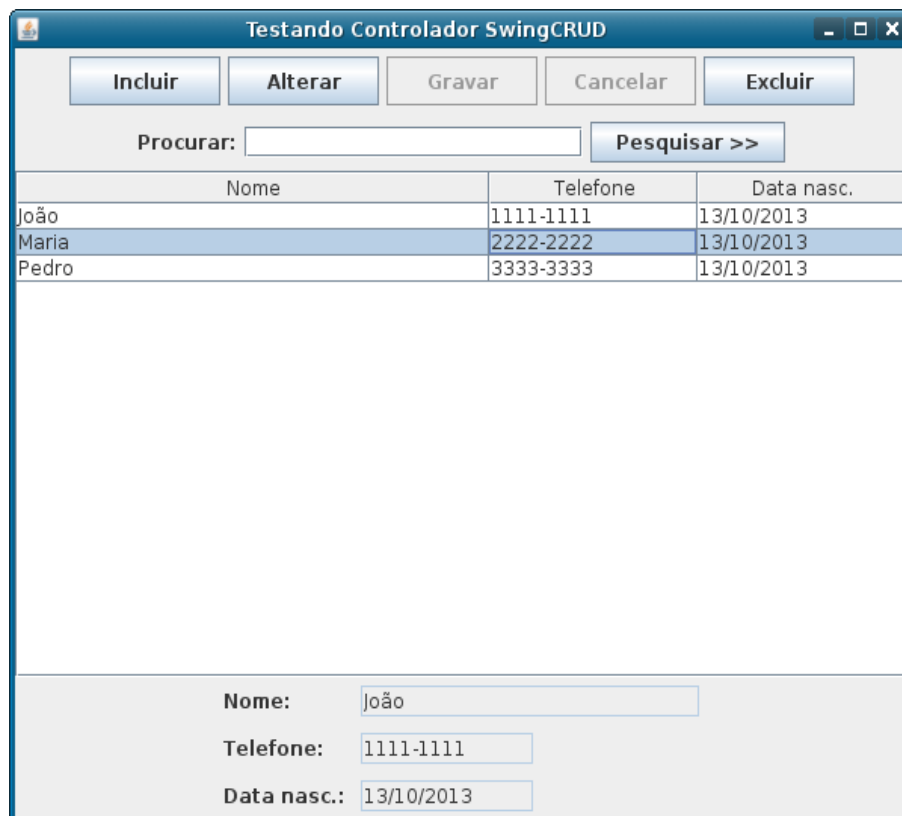
Framework SwingCRUD

Criado por:

Éderson Cássio Lacerda Ferreira
ederson_cassio@yahoo.com.br

Licença: LGPL 3.0 (100% livre, por sua conta e risco!)

Função: fornecer uma mecânica de CRUD fácil de ser acoplada a interfaces Swing, sem se prender a um layout específico.



Nome	Telefone	Data nasc.
João	1111-1111	13/10/2013
Maria	2222-2222	13/10/2013
Pedro	3333-3333	13/10/2013

Nome: João
Telefone: 1111-1111
Data nasc.: 13/10/2013

Como funciona

Tirando proveito dos tipos genéricos do Java, o framework controla o funcionamento de formulários de cadastro sem se preocupar com o tipo de objeto que está manipulando. Ele simplifica a interação entre um componente `JTable` e seu `TableModel`, um painel com campos onde o usuário realiza a entrada de dados e os botões que recebem os comandos do usuário.

É composto pelo controlador e um conjunto de classes abstratas, que devem ser derivadas para se obter o comportamento específico para cada tipo de objeto a ser manipulado. Ex.:

```
public class TableModelContato extends TableModelEntidade<Contato> { ... }  
public class PainelContato extends PainelCampos<Contato> { ... }  
public class CrudContatoListener extends CRUDListener<Contato> { ... }
```

Componentes

- **TableModelEntidade<T>:** o TableModel requerido pelo componente.JTable, já adaptado para interagir com os objetos armazenados e o controlador. Suas classes derivadas devem apenas informar os nomes das colunas e qual atributo da entidade corresponde a cada coluna.
- **PainelCampos<T>:** o painel onde deverão ser criados os campos para a entrada de dados pelo usuário, e métodos para exibir os dados de um objeto ou criar um objeto a partir dos dados digitados.
- **CRUDListener<T>:** monitora as ações do usuário e responde invocando o banco de dados, solicitando confirmações, efetuando validações, etc.
- **ControladorCRUD<T>:** o componente central, que captura as ações do usuário e coordena o trabalho dos outros três. Esta classe é a única que não é abstrata e não precisa ser derivada, bastando instanciá-la.

É interessante saber que há um outro framework, o **SwingBean** (swingbean.sourceforge.net), que faz exatamente as tarefas deixadas a cargo do PainelCampos, visto que criar layouts de formulários em Swing e trocar seus dados com objetos são tarefas bastante trabalhosas. O SwingCRUD abstrai exatamente o que o outro framework já faz, e é ao implementar seu PainelCampos customizado que o SwingBean seria invocado. No exemplo implementado neste guia, no entanto, não será usado o SwingBean.

Finalmente - criando uma aplicação!

O layout padrão

O SwingCRUD não prende o desenvolvedor a nenhum layout específico. Embora não traga um layout padrão já pronto, pretendo mostrar-lhes como criar seu próprio layout padrão, e a partir dele gerar novos CRUDs em poucos minutos!

```
public abstract class FormPadrao<T> extends JFrame {  
  
    private TableModelEntidade<T> tableModel;  
    private PainelCampos<T> painelCampos;  
    private CRUDListener<T> crudListener;  
  
    private ControladorCRUD<T> controlador;  
  
    // Será customizável pelos filhos, o form pai apenas reserva espaço.  
    // Repare que podemos criar os componentes que desejarmos no formulário padrão.  
    private JPanel painelPesquisa;  
  
    // continua...  
}
```

Veja que o formulário padrão é uma classe genérica e abstrata. Estendemos JFrame, mas poderíamos estender qualquer componente de tela do Swing.

Vamos ao construtor:

```

public FormPadrao() {
    // Criando as dependências do controlador
    tableModel = criaTableModel();
    painelCampos = criaPainelCampos();
    crudListener = criaCrudListener();

    // Criando o controlador
    controlador = new ControladorCRUD<T>(tableModel, painelCampos,
        crudListener);

    // Obtendo os componentes criados pelo controlador
    JTable tabela = controlador.getTabela();
    JButton incluir = controlador.getIncluir();
    JButton alterar = controlador.getAlterar();
    JButton gravar = controlador.getGravar();
    JButton cancelar = controlador.getCancelar();
    JButton excluir = controlador.getExcluir();

    // Personalizando os componentes
    Dimension d = new Dimension(100, 30);
    incluir.setPreferredSize(d);
    alterar.setPreferredSize(d);
    gravar.setPreferredSize(d);
    cancelar.setPreferredSize(d);
    excluir.setPreferredSize(d);

    // Layout do formulário a gosto - é VOCÊ que está criando um layout padrão.
    painelPesquisa = new JPanel();
    JPanel painelBotoes = new JPanel();
    painelBotoes.add(incluir);
    painelBotoes.add(alterar);
    painelBotoes.add(gravar);
    painelBotoes.add(cancelar);
    painelBotoes.add(excluir);

    JPanel painelNorte = new JPanel();
    painelNorte.setLayout(new BorderLayout(painelNorte, BorderLayout.Y_AXIS));
    painelNorte.add(painelBotoes);
    painelNorte.add(painelPesquisa);

    JScrollPane scroll = new JScrollPane(tabela);
    scroll.getViewport().setBackground(Color.WHITE);

    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    c.add(painelNorte, BorderLayout.NORTH);
    c.add(scroll, BorderLayout.CENTER);
    c.add(painelCampos, BorderLayout.SOUTH);

    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
}

```

Sei que este construtor não é um exemplo de bom design pelo seu tamanho, mas o objetivo disso é ser mais direto e fazer o guia caber em menos páginas. Refatorar o código fica por conta do desenvolvedor. Layouts em Swing são complexos mesmo, e o objetivo do formulário padrão é encapsular esse layout.

Perceba que, no início, devem ser criadas as dependências do controlador. Mas elas são classes abstratas, e ainda não estamos criando nenhum formulário de Clientes, Produtos... Portanto, os métodos que criam esses objetos serão abstratos também:

```

public abstract TableModelEntidade<T> criaTableModel();
public abstract PainelCampos<T> criaPainelCampos();
public abstract CRUDListener<T> criaCrudListener();

```

Saindo fora do escopo do framework, vamos dar às classes filhas acesso aos componentes do layout (no nosso caso, um painel de pesquisa):

```

protected JPanel getPainelPesquisa() {
    // Permitir aos filhos customizar o painel
    return painelPesquisa;
}

```

Criando um cadastro de contatos

Agora que temos um layout padrão para distribuir os componentes visuais do framework (o painel de campos, o JTable, os botões e outros componentes mais que você criar), iremos criar um formulário para cadastrar contatos derivado desse layout padrão.

Inicialmente teremos nossa entidade Contato:

```
public class Contato {  
  
    private String nome;  
    private String telefone;  
    private Calendar nascimento;  
  
    public Contato() {  
    }  
  
    public Contato(String nome, String telefone, Calendar nascimento) {  
        this.nome = nome;  
        this.telefone = telefone;  
        this.nascimento = nascimento;  
    }  
  
    // Getters e setters omitidos  
}
```

Em seguida criaremos uma classe derivada do formulário padrão e declararemos nela os nossos componentes concretos necessários para o controlador:

```
public class FormContato extends FormPadrao<Contato> {  
  
    private TableModelContato tableModel;  
    private PainelContato painelContato;  
    private CrudContatoListener crudListener;  
  
    // Componentes customizados a gosto!  
    private JTextField txtPesquisa;  
  
    @Override  
    public TableModelEntidade<Contato> criaTableModel() {  
        // Criamos e guardamos referência ao TableModel  
        tableModel = new TableModelContato();  
        return tableModel;  
    }  
  
    @Override  
    public PainelCampos<Contato> criaPainelCampos() {  
        // Criamos e guardamos referência ao painel de campos  
        painelContato = new PainelContato();  
        return painelContato;  
    }  
  
    @Override  
    public CRUDListener<Contato> criaCrudListener() {  
        // Criamos e guardamos referência ao monitor dos eventos  
        crudListener = new CrudContatoListener();  
        return crudListener;  
    }  
}
```

Implementando o TableModel

O TableModel é o mais simples de todos. Criar um TableModel do zero em Java não é fácil, mas o SwingCRUD já faz “quase” tudo o que é preciso. Você só precisa informar quais são suas colunas e que atributo do objeto aparece em cada coluna:

```

public class TableModelContato extends TableModelEntidade<Contato> {

    // TableModel responsável por exibir os contatos no JTable

    private DateFormat formato = DateFormat.getDateInstance();

    @Override
    public String[] getColunas() {
        // Aqui retorno os nomes das colunas na grid
        return new String[] {"Nome", "Telefone", "Data nasc."};
    }

    @Override
    public Object getDadoColuna(int coluna, Contato contato) {
        String nascimento = formato.format(contato.getNascimento().getTime());

        // Aqui retorno um dado do contato de acordo com os índices das colunas
        switch (coluna) {
            case 0: return contato.getNome();
            case 1: return contato.getTelefone();
            case 2: return nascimento;
            default: return null;
        }
    }
}

```

O painel de campos

Este painel exige algum trabalho braçal... Aqui você pode usar o SwingBean para auxiliar, recomendo que você pesquise sobre ele.

```

public class PainelContato extends PainelCampos<Contato> {

    // Painel contendo a exibição dos campos
    // Aqui eu poderia chamar o framework SwingBean para facilitar a criação
    // do layout e a troca de dados entre o formulário e os objetos.

    private JTextField nome = new JTextField(20);
    private JTextField telefone = new JTextField(10);
    private JTextField nascimento = new JTextField(10);

    private Contato contatoAtual;

    public PainelContato() {

        // Layout a gosto...

        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.anchor = gbc.WEST;
        gbc.insets = new Insets(5, 5, 5, 5);

        gbc.gridx = 0;
        gbc.gridy = 0;

        add(new JLabel("Nome:"), gbc);
        gbc.gridy++;
        add(new JLabel("Telefone:"), gbc);
        gbc.gridy++;
        add(new JLabel("Data nasc."), gbc);
        gbc.gridx = 1;
        gbc.gridy = 0;

        add(nome, gbc);
        gbc.gridy++;
        add(telefone, gbc);
        gbc.gridy++;
        add(nascimento, gbc);
    }
}

```

```

@Override
public void exibir(Contato c) {
    // Aqui exibimos o contato no painel e guardamos uma referência a ele
    contatoAtual = c;
    nome.setText(c.getNome());
    telefone.setText(c.getTelefone());
    nascimento.setText(DateFormat.getDateInstance()
        .format(c.getNascimento().getTime()));
}

@Override
public Contato novoObjeto() throws CRUDEXception {
    // Aqui é chamado quando vai gravar um novo contato
    // Clicou Incluir, depois Gravar
    // Posso lançar uma exceção para indicar que os dados são inválidos
    // e impedir o controlador de efetuar a gravação do contato.
    Contato novo = new Contato();
    popula(novo);
    contatoAtual = novo;
    return novo;
}

@Override
public Contato objetoSendoAlterado() throws CRUDEXception {
    // Aqui é chamado quando vai gravar um contato existente
    // Clicou Alterar, depois Gravar
    // É interessante retornar o mesmo objeto recebido no Exibir, pois
    // ele pode conter, por exemplo, o id do banco de dados
    popula(contatoAtual);
    return contatoAtual;
}

private void popula(Contato c) throws CRUDEXception {
    try {
        c.setNome(nome.getText().trim());
        c.setTelefone(telefone.getText().trim());
        Calendar calNascimento = Calendar.getInstance();

        // Aqui pode haver uma exceção se a data estiver em formato inválido
        calNascimento.setTime(DateFormat.getDateInstance()
            .parse(nascimento.getText().trim()));

        c.setNascimento(calNascimento);
    }
    catch (ParseException pex) {
        throw new CRUDEXception("Data inválida: " + nascimento.getText());
    }
}

@Override
public void limpar() {
    // Aqui eu limpo meus campos
    nome.setText("");
    telefone.setText("");
    nascimento.setText("");
}

@Override
public void habilitarCampos() {
    // Aqui eu habilito os campos
    nome.setEditable(true);
    telefone.setEditable(true);
    nascimento.setEditable(true);
}

@Override
public void desabilitarCampos() {
    // Aqui eu desabilito os campos.
    // O controlador começa em modo não editável.
    nome.setEditable(false);
    telefone.setEditable(false);
    nascimento.setEditable(false);
}
}

```

O monitor de eventos

Esta classe optei por criar como interna do formulário de contatos (como são em geral os Listeners), para ter mais facilidade de acesso aos componentes do formulário.

A interface CRUDListener possui muitos métodos, mas o SwingCRUD fornece o CRUDAdapter para que você sobrescreva somente os que interessar ao seu projeto.

```
private class CrudContatoListener extends CRUDAdapter<Contato> {

    // Aqui ficam os eventos do CRUD:
    // - as ações de incluir, alterar e excluir
    // - eventos antes e após estas ações
    // Os eventos antes de ações podem retornar false para impedir a ação
    // (Sim, eu copieei o modelo do Dataset do Delphi aqui :D)

    @Override
    public void aposBotaoIncluir() {
        painelContato.focoEmNome();
    }

    @Override
    public void aposBotaoAlterar() {
        painelContato.focoEmNome();
    }

    @Override
    public void acaoGravarInclusao(Contato contato) throws Exception {
        // Aqui eu mando incluir o contato no banco de dados (ou lanço uma exceção
        // caso falhe)
    }

    @Override
    public void acaoGravarAlteracao(Contato contato) throws Exception {
        // Aqui eu mando atualizar o contato no banco de dados (ou lanço uma exceção
        // caso falhe)
    }

    @Override
    public void aposBotaoGravar() {
        getControlador().getAlterar().requestFocus();
    }

    @Override
    public boolean antesBotaoExcluir() {
        if (JOptionPane.showConfirmDialog(null, "Tem certeza?", "Excluir",
            JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
            return true;
        else
            return false;
    }

    @Override
    public void acaoExcluir(Contato contato) throws Exception {
        // Aqui eu mando excluir o contato no banco de dados (ou lanço uma exceção
        // caso falhe)
    }

    @Override
    public void aposBotaoExcluir() {
        // É bom direcionar o foco para algum lugar, de modo que o usuário
        // não fique perdido...
        getControlador().getIncluir().requestFocus();
    }
}
```

Esta classe chama alguns métodos que ainda não temos. Vamos criá-los:

No formulário padrão:

```
protected ControladorCRUD<T> getControlador() {
    // Permitir aos filhos interagir com os componentes criados
    return controlador;
}
```

No painel de campos:

```
public void focoEmNome() {
    nome.requestFocus();
}
```

Também temos os métodos que chamariam o banco de dados, mas no momento não estão fazendo nada. Para testar o formulário, eles podem ficar vazios - se não lançarem exceções, o controlador insere/atualiza/exclui os contatos do TableModel automaticamente, mesmo sem interagir com um banco de dados real.

Finalizando a aplicação

Para finalizar, vamos criar um construtor no FormContato que faz uma série de configurações iniciais:

```
public FormContato() {
    setTitle("Testando Controlador SwingCRUD");
    setSize(600, 500);

    // Tamanho de uma coluna do JTable
    tamanhoColuna(0, 250);

    // Painel de pesquisa customizado
    txtPesquisa = new JTextField(20);
    JButton btnPesquisar = new JButton("Pesquisar >>");
    txtPesquisa.addActionListener(new AcaoPesquisar());
    btnPesquisar.addActionListener(new AcaoPesquisar());

    getPainelPesquisa().add(new JLabel("Procurar:"));
    getPainelPesquisa().add(txtPesquisa);
    getPainelPesquisa().add(btnPesquisar);

    // Simulando o banco de dados
    List<Contato> lista = new ArrayList<Contato>();
    lista.add(new Contato("João", "1111-1111", Calendar.getInstance()));
    lista.add(new Contato("Maria", "2222-2222", Calendar.getInstance()));
    lista.add(new Contato("Pedro", "3333-3333", Calendar.getInstance()));
    tableModel.setListaObjetos(lista);
}
```

Para dimensionar uma coluna do JTable, fazemos (optei no FormPadrao):

```
protected void tamanhoColuna(int indice, int largura) {
    getControlador().getTabela().getColumnModel().getColumn(indice)
        .setPreferredWidth(largura);
}
```

Para simular uma pesquisa, fazemos uma classe interna do FormContato:

```
private class AcaoPesquisar implements ActionListener {

    // Simulando uma pesquisa no banco de dados...

    @Override
    public void actionPerformed(ActionEvent e) {
        List<Contato> lista = new ArrayList<Contato>();
        lista.add(new Contato("Carla", "4444-4444", Calendar.getInstance()));
        lista.add(new Contato("Priscila", "5555-5555", Calendar.getInstance()));
        lista.add(new Contato("Antônio", "6666-6666", Calendar.getInstance()));
        tableModel.setListaObjetos(lista);
    }
}
```


O primeiro CRUD está pronto! Basta criar um método main em algum lugar para chamá-lo:

```
public static void main(String[] args) {  
    new FormContato().setVisible(true);  
}
```