SWING

1. INTRODUÇÃO

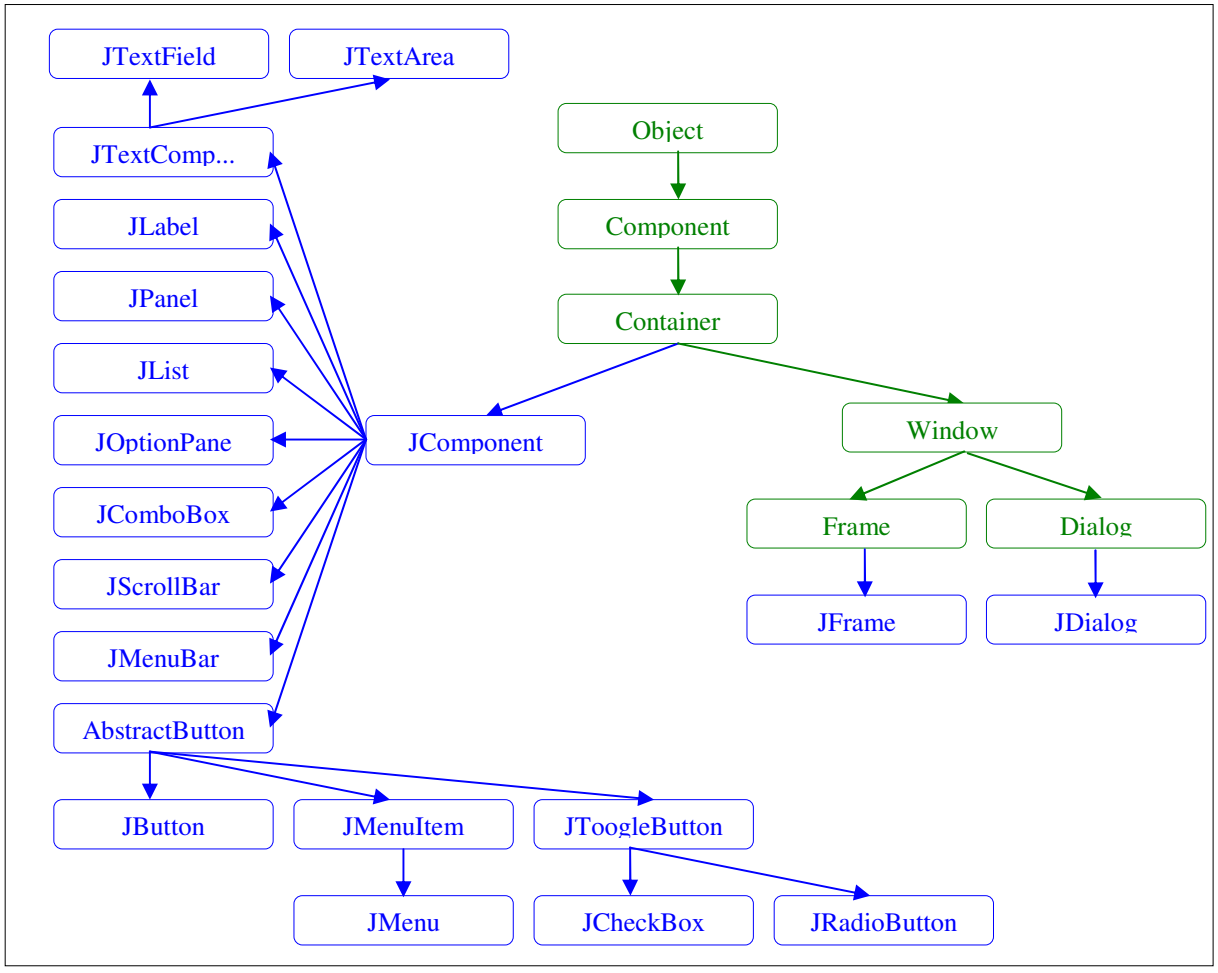
Swing

Swing é uma API Java usada na construção de interfaces gráficas, conhecidas como GUI – Graphical User Interface – estando presente no Java desde a versão 1.2.

Apesar de parecer uma evolução da API AWT (Abstract Window Toolkit), o Swing foi totalmente reescrito para poder desenhar os componentes sem usar APIs do sistema operacional. Cada componente Swing tem nome parecido com seu correspondente AWT, tendo apenas um "J" na frente para diferenciar, por exemplo, Button e JButton, AWT e Swing respectivamente.

Componentes

A figura abaixo traz uma visão geral sobre a arquitetura de componentes do Swing, com alguns dos principais componentes disponíveis. Em verde, classes do pacote AWT, e em azul do Swing:



A montagem das janelas pode ser feita diretamente no código fonte, ou através de IDEs (Integrated Development Environment) RAD (Rapid application development) que permitem "drag-and-drop" (arraste e solte) como o Eclipse com o VE e o NetBeans com o Matisse.

Eventos

Vários dos componentes que podem ser vistos na figura anterior geram eventos baseados na ação o usuário. Em nossos aplicativos, esses eventos são recuperados e devidamente tratados. O mais comum de todos é o clique no botão, mas existem muitos outros como apertar e soltar uma tecla, clicar com o botão direito do mouse, clicar e arrastar o mouse, escolher um item de menu, selecionar um checkBox, um item de uma lista, etc.

Para cada componente definimos uma classe que irá tratar o evento referente a este componente. Uma mesma classe poderá tratar mais de um evento. A classe de tratamento pode ser anônima, privada, a própria classe de exibição, ou uma classe externa.

Para trabalhar com eventos precisamos registrar um ouvinte para o evento (listener) e um manipulador (handler).

Por exemplo para uma janela temos o ouvinte "WindowListener" e podemos implementar qualquer um dos manipuladores: "windowClosing", "windowOpened", "windowActivated". Para o botão, tem o ouvinte "ActionListener" e o manipulador "actionPerformed".

2. COMPONENTES BÁSICOS

Novo projeto no Eclipse

Para criar um novo projeto Java para desktop clicamos no menu "File→New→Project", escolhemos "Java Project". Definimos um nome para projeto no campo "Project name", configuramos a pasta onde ficará o código fonte, e concluímos clicando em "Finish".

Depois criamos uma pasta para armazenar os pacotes e agrupa o código fonte do projeto. No menu "File→New→Source Folder", colocamos o nome da pasta no campo "Folder name", depois "Finish".

Em seguida criamos os pacotes necessários, conforme o padrão pré-definido. Para isso usamos o menu "File→New→Package", informamos o nome do pacote em "Name" e concluímos clicando em "Finish".

Ainda, precisamos de uma classe que implemente o método "main", que é por onde toda aplicação Java é iniciada. Selecionamos o pacote desejado e vamos em "File→New→Class", definimos um nome em "Name" e marcamos a opção "public static void main(String[] args)", depois "Finish".

Para rodar uma aplicação desktop de dentro do eclipse clicamos com o botão direito sobre a classe que contém o método "main", depois "Run As→Java Application". Logicamente nada acontecerá, pois não fizemos nenhuma implementação.

Para forçar a saída da aplicação usamos "`System.exit(0);`".

JLabel

O JLabel é conhecido como rótulo, podendo receber texto ou imagem, é um componente muito usado e que tem seu estado raramente alterado durante a execução do programa.

Exemplos das diferentes formas de utilização do JLabel:

```
public class ExemploJLabel extends JFrame {
        private JLabel rotulo;
        private JLabel imagem;
        private JLabel rotuloImagem;
        private JLabel rotuloPerson;
        public ExemploJLabel() {
                super("Exemplo de JLabel");
                setSize(300, 400);
                setLayout(null);

                rotulo = new JLabel("Apenas texto");
                imagem = new JLabel(new ImageIcon("images/teste.gif"));
                rotuloImagem = new JLabel("Texto e imagem", new
ImageIcon("images/teste.gif"), SwingConstants.LEFT);
                rotuloPerson = new JLabel("JLabel personalizado");
                rotuloPerson.setForeground(new Color(201,200,100));
                rotuloPerson.setFont(new Font("Arial",Font.BOLD,20));
                rotuloPerson.setToolTipText("Exemplo de toolTip de rótulo");

                rotulo.setBounds(30, 25, 100, 20);
                imagem.setBounds(30, 50, 100, 100);
                rotuloImagem.setBounds(30, 150, 200, 100);
                rotuloPerson.setBounds(30, 275, 270, 30);

                Container cp = getContentPane();
                cp.add(rotulo);
                cp.add(imagem);
                cp.add(rotuloImagem);
                cp.add(rotuloPerson);
                setVisible(true);
        }
//main …
}
```

Criamos três rótulos neste exemplo. O primeiro apenas texto, o segundo apenas imagem, e um terceiro juntando texto e imagem. Mais a frente, vamos detalhar os diferentes tipos de layout, por enquanto não estamos usando nenhum layout "`setLayout(null);`" e então podemos definir a posição e dimensões exatas de cada componente, como fizemos com "`rotulo.setBounds(30, 25, 100, 20);`". Para o rótulo personalizado aplicamos cor, tipo, estilo e tamanho de fonte, diferentes do padrão e adicionamos um texto explicativo "`rotuloPerson.setToolTipText("Exemplo de toolTip de rótulo");`" que irá aparecer quando o usuário deixar o mouse parado sobre o JLabel.

JTextField

O JTextField é o componente utilizado para entrada de informações pelo usuário, tais como textos, quantidades, valores monetários, datas, senhas, etc. Existem algumas variações como JTextArea, JPasswordField, JFormattedField.

Abaixo, exemplificamos o JTextField e mais a frente vamos detalhar os demais:

```
public class ExemploJTextField extends JFrame {
        private JLabel lbNome;
        private JLabel lbEmail;
        private JTextField fdNome;
        private JTextField fdEmail;
        public ExemploJTextField() {
                super("Exemplo de JTextField");
                setSize(400, 150);
                setLayout(null);

                lbNome = new JLabel("Nome:");
                lbEmail = new JLabel("E-mail:");
                fdNome = new JTextField(40);
                fdNome.requestFocus();
                fdEmail = new JTextField(60);
                fdEmail.setText("email@email.com.br");
                fdEmail.setEditable(false);

                lbNome.setBounds(20, 20, 40, 20);
                lbEmail.setBounds(20, 45, 40, 20);
                fdNome.setBounds(65, 20, 170, 20);
                fdEmail.setBounds(65, 45, 230, 20);

                Container cp = getContentPane();
                cp.add(lbNome);
                cp.add(fdNome);
                cp.add(lbEmail);
                cp.add(fdEmail);
                setVisible(true);
        }
//main …
}
```

Criamos dois rótulos com duas caixas de texto: nome e e-mail, com tamanho 40 e 60, respectivamente, definidos na criação. A caixa de texto "nome" recebe o foco ao abrir a janela "`fdNome.requestFocus();`". Já o "email" recebe um texto padrão e não pode ser alterado. Cores, fontes e alinhamentos funcionam igual ao que vimos nos tópicos anteriores.

JButton

O JButton é o componente usado para permitir que o usuário indique a ação desejada, pode estar representado de várias maneiras: botões normais, itens de menu, múltipla seleção. O evento gerado pelo botão é um ActionEvent que pode ser tratado por uma classe que implemente a interface ActionListener.

O exemplo abaixo é um continuação do exemplo do JTextField, os "..." representam o código já existente:

```
public class ExemploJButton extends JFrame {
//…
        private JButton btExibirNome;
        private JButton btEditarEmail;
        public ExemploJButton() {
                super("Exemplo de JButton");
                setSize(400, 150);
                setLayout(null);
//…
                btExibirNome = new JButton("Exibir nome");
                btExibirNome.addActionListener(new BtExibirNomeHandler());
                btEditarEmail = new JButton("Editar e-mail");
                btEditarEmail.addActionListener(new BtEditarEmailHandler());
                btEditarEmail.setMnemonic(KeyEvent.VK_E);
//…
                btExibirNome.setBounds(20, 70, 120, 20);
                btEditarEmail.setBounds(150, 70, 120, 20);
//…
                cp.add(btExibirNome);
                cp.add(btEditarEmail);
                getRootPane().setDefaultButton(btExibirNome);
                setVisible(true);
        }
        private class BtExibirNomeHandler implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        JOptionPane.showMessageDialog(null, fdNome.getText());
                }
        }
        private class BtEditarEmailHandler implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        fdEmail.setEditable(true);
                        fdEmail.requestFocus();
                }
        }
//main …
}
```

Os eventos podem ser tratados de várias formas. Uma das formas mais simples e claras é como está no exemplo "`private class BtExibirNomeHandler implements ActionListener`", onde temos uma classe que gerencia o evento de cada botão, e na criação do botão, associamos o objeto que fará o tratamento do evento "`btExibirNome.addActionListener(new BtExibirNomeHandler());`". A chamada "`btEditarEmail.setMnemonic(KeyEvent.VK_E);`" é usada para definir a tecla de atalho do botão. Definimos o botão principal da janela que responde ao "Enter" como "`getRootPane().setDefaultButton(btExibirNome);`". Assim como JLabel o JButton pode receber texto, imagem ou texto e imagem. E como os outros componentes que já vimos, podemos personalizar a fonte, cor, alinhamento, etc.

Eventos do mouse

Já vimos um exemplo de tratamento de evento do clique do botão. Outros muitos eventos podem ser tratados seguindo o padrão do exemplo anterior. A seguir mostramos um exemplo de tratamento dos eventos do mouse que são em maior número.

```
public class ExemploEventosMouse extends JFrame {
	public ExemploEventosMouse() {
		super("Exemplo de eventos do mouse");
		setSize(400, 300);
		addMouseListener(new MouseHandler());
		addMouseMotionListener(new MouseMotionHandler());
		setVisible(true);
	}
	private class MouseHandler implements MouseListener {
	public void mouseClicked(MouseEvent e) {
		if (e.getButton() == MouseEvent.BUTTON1) {
			JOptionPane.showMessageDialog(null, "Clique no botão esquerdo do
mouse");
		} else if (e.getButton() == MouseEvent.BUTTON3) {
			JOptionPane.showMessageDialog(null, "Clique no botão direito do
mouse");
		}
	}
// …
}
private class MouseMotionHandler implements MouseMotionListener {
	public void mouseMoved(MouseEvent e) {
		JFrame frame = (JFrame)e.getSource();
		frame.getGraphics().drawLine(e.getX(), e.getY(), e.getX()+4, e.getY());
	}
	public void mouseDragged(MouseEvent e) {
		JFrame frame = (JFrame)e.getSource();
		frame.getGraphics().drawLine(e.getX(), e.getY(), e.getX()+12,
e.getY()+12);
	}
}
//main …
}
```

Implementamos duas interfaces uma para manipular o evento de clique e a outra a de movimentação do mouse.

3. GERENCIADORES DE LAYOUT

Os gerenciadores de layout são usados para permitir que a mesma aplicação com o mesmo código, se comporte bem nos vários sistemas operações, nas várias resoluções de tela, e várias dimensões da janela.

O Swing disponibiliza vários gerenciadores de layout. Vamos conhecer cada um deles com um exemplo explicativo.

Um método muito utilizado em conjunto com os gerenciadores de layout é o "`pack();`", que auto dimensiona o tamanho da janela baseado no tamanho dos seus componentes.

Null Layout

Deixa livre para o programador indicar a posição e as dimensões de cada componente. O problema é que perdemos as vantagens de multi-plataforma e diferentes "peles" aplicáveis, assim como diferentes resoluções. No entanto, é mais simples e fácil de aprender, e o que mais se assemelha a API.

```
public class ExemploLayoutNull extends JFrame {
	private JButton button1;
	private JButton button2;
	private JButton button3;
	private JButton button4;
	private JButton button5;
	public ExemploLayoutNull() {
		super("Exemplo de Null Layout");
		setSize(400, 220);
		setLayout(null);
		button1 = new JButton("Botão 1");
		button2 = new JButton("Botão 2");
		button3 = new JButton("Botão 3");
		button4 = new JButton("Botão 4");
		button5 = new JButton("Botão 5");

		button1.setBounds(25, 25, 100, 25);
		button2.setBounds(200, 25, 100, 25);
		button3.setBounds(25, 75, 100, 25);
		button4.setBounds(200, 75, 100, 25);
		button5.setBounds(25, 125, 100, 25);

		add(button1);
		add(button2);
		add(button3);
		add(button4);
		add(button5);
		setVisible(true);
	}
//main …
}
```

O "`button1.setBounds(25, 25, 100, 25);`" é que define a posição na tela, sem essa chamada o objeto não irá aparecer, pois a API espera que o programador defina onde ele deve aparecer.

FlowLayout

Organiza os componentes conforme a ordem de adição, como se fosse um fluxo, um após o outro, definindo um tamanho para cada componente baseado no cálculo do seu tamanho mínimo. Muito útil para organizar uma barra de ferramentas ou status, mas ruim para o gerenciamento layout geral da grande maioria das janelas.

```
public class ExemploLayoutFlow extends JFrame {
//…
	public ExemploLayoutFlow() {
		super("Exemplo de FlowLayout");
		setSize(400, 150);
		setLayout(new FlowLayout());
//…
	}
//main …
}
```

Mesmo deixando as definições de posição e dimensão, elas não são respeitadas e podem ser descartadas. Se diminuirmos a largura da janela, os botões irão automaticamente para a linha de baixo. O alinhamento padrão é centralizado, mas podemos escolher passando nossa opção no construtor "`setLayout(new FlowLayout(FlowLayout.RIGHT));`".

BorderLayout

Organiza os componentes na ordem nos seguintes pontos da janela: norte, sul, leste, oeste, centro. Reservando o espaço maior para o objeto central. Só podemos inserir um componente por região, e este componente será dimensionado para preencher todo o

espaço disponível na região. Mais à frente veremos o JPanel, que comporta outros componentes, e com isso adicionamos o JPanel a região e vários outros componentes ao JPanel.

```
public class ExemploLayoutBorder extends JFrame {
//…
        public ExemploLayoutBorder() {
                super("Exemplo de BorderLayout");
                setSize(400, 150);
                setLayout(new BorderLayout());
//…
                add(button1, BorderLayout.NORTH);
                add(button2, BorderLayout.EAST);
                add(button3, BorderLayout.WEST);
                add(button4, BorderLayout.CENTER);
                add(button5, BorderLayout.SOUTH);
                setVisible(true);
        }
//main …
}
```

Ao adicionar o componente precisamos definir em qual região ele ficará "`add(button1, BorderLayout.NORTH);`". Também podemos definir um espaçamento entre os componentes na construtor "`setLayout(new BorderLayout(5, 5));`".

GridLayout

Organiza os componentes numa tabela com número de linhas e colunas pré-definido, tendo cada célula o mesmo tamanho, baseado no maior componente e no tamanho da janela. Cada novo componente adicionado é atribuído à próxima célula vazia, da esquerda para a direita, de cima para baixo.

```
public class ExemploLayoutGrid extends JFrame {
//…
        public ExemploLayoutGrid() {
                super("Exemplo de GridLayout");
                setSize(400, 150);
                setLayout(new GridLayout(3, 2));
//…
        }
//main …
}
```

Podemos também definir um espaçamento entre os componentes no construtor semelhante ao que é feito no BorderLayout "`setLayout(new GridLayout(3, 2, 5, 5));`".

BoxLayout

Posiciona os componentes um após o outro em linha ou em coluna com base na configuração realizada na criação. Os "..." que aparecerão na seqüência de exemplos e exercícios de layout representam o código já existente no primeiro exemplo e exercício deste tópico.

```
public class ExemploLayoutBox extends JFrame {
//…
        public ExemploLayoutBox() {
                super("Exemplo de BoxLayout");
                setSize(400, 220);
                setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS));
//…
        }
//main …
}
```

Definimos que queremos os componentes organizados em colunas com "`BoxLayout.Y_AXIS`". Poderia ser em linha, ou ainda linha ou coluna, baseado na orientação do painel onde estão sendo inseridos.

CardLayout

Organiza os objetos um sobre o outro e disponibiliza métodos para manipulá-los como next (próximo), last (último). Segue exemplo.

```
public class ExemploCardLayout extends JFrame {
//…
        public ExemploLayoutBorder() {
                super("Exemplo de Card Layout");
                setSize(400, 220);
                setLayout(new CardLayout());
//…
                ActionListener handler = new ButtonHandler();
                button1.addActionListener(handler);
                button2.addActionListener(handler);
                button3.addActionListener(handler);
                button4.addActionListener(handler);
                button5.addActionListener(handler);

                add(button1,"1");
                add(button2,"2");
                add(button3,"3");
                add(button4,"4");
                add(button5,"5");
                setVisible(true);
        }
//main …
}
```

GridBagLayout

É o mais flexível, porém o mais complexo de compreender. Ele permite que um componente utilize mais de uma célula, podendo expandir para ocupar o espaço ou ficar alinhado a algum dos lados da célula. Linhas e colunas se ajustam ao maior componente.

```
public class ExemploLayoutGridBag extends JFrame {
//…
        public ExemploLayoutGridBag() {
                super("Exemplo de GridBagLayout");
                setSize(400, 150);
                GridBagLayout layout = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                setLayout(layout);
//…
                c.gridy = 0;
                c.gridx = 0;
                layout.setConstraints(button1, c);
                c.gridy = 1;
                c.gridx = 1;
                layout.setConstraints(button2, c);
                c.gridy = 2;
                c.gridx = 2;
                layout.setConstraints(button3, c);
                c.gridy = 3;
                c.gridx = 0;
                c.gridwidth = 3;
                c.fill = GridBagConstraints.BOTH;
                layout.setConstraints(button4, c);
                c.gridy = 0;
                c.gridx = 2;
                layout.setConstraints(button5, c);
//…
        }
//main …
}
```

As linhas e colunas começam a contar de “0” (zero). O botão “4” ocupa o espaço de 3 células “`c.gridwidth = 3;`” e é dimensionado para ocupar todo o espaço “`c.fill = GridBagConstraints.BOTH;`”.

JPanel

O componente JPanel não é um gerenciador de layout, mas está intimamente relacionado.

Nem sempre conseguimos criar uma interface amigável usando apenas um único recurso de gerenciamento de layout. Para resolver esse problema o Swing possui o componente JPanel que permite agrupar componentes com um layout próprio.

```
public class ExemploJPanel extends JFrame {
        private JButton button1;
        private JButton button2;
        private JButton button3;
        private JButton button4;
        private JPanel panel1;
        private JPanel panel2;

        public ExemploJPanel() {
                super("Exemplo de JPanel");
                setLayout(new BorderLayout());
                setLocationRelativeTo(null);

                button1 = new JButton("Botão 1");
                button2 = new JButton("Botão 2");
                button3 = new JButton("Botão 3");
                button4 = new JButton("Botão 4");

                panel1 = new JPanel();
                panel1.setLayout(new GridLayout(2,1));

                panel1.add(button1);
                panel1.add(button2);

                panel2 = new JPanel();
                panel2.setLayout(new FlowLayout(FlowLayout.RIGHT));

                panel2.add(button3);
                panel2.add(button4);

                Container cp = getContentPane();
                cp.add(panel1, BorderLayout.CENTER);
                cp.add(panel2, BorderLayout.SOUTH);

                setVisible(true);
                pack();
        }
//main …
}
```

Adicionamos os JPanel à janela e aos JPanel os objetos que queremos, com o layout que necessitamos. Assim podemos dividir a nossa janela em milhares de pequenos pedaços, inclusive, adicionando JPanel a JPanel para formar layouts ainda mais elaborados.

4. COMPONENTES AVANÇADOS

JMenuBar

Para trabalhar com menu utilizamos várias classes, cada uma com uma função bem definida: JMenuBar representa a barra de menus; JMenu representa um menu que agrupa itens de menu; JMenuItem representa o item de menu; além destes temos especializações como JCheckBoxMenuItem e JRadioButtonMenuItem.

Abaixo, exemplo de utilização das classes de menu.

```
public class ExemploJMenu extends JFrame {
	private JMenuBar menuBar;
	private JMenu menuArquivo;
	private JMenu menuCodigo;
	private JMenu menuNovo;
	private JMenuItem menuItemNovoProjeto;
	private JMenuItem menuItemNovoClasse;
	private JMenuItem menuItemFechar;
	private JCheckBoxMenuItem menuItemAutoCompletar;
	private JRadioButtonMenuItem menuItemTexto;
	private JRadioButtonMenuItem menuItemHtml;
	private ButtonGroup group;
	public ExemploJMenu() {
		super("Exemplo de Menu");
		setSize(400, 150);
		menuBar = new JMenuBar();
		setJMenuBar(menuBar);
		menuArquivo = new JMenu("Arquivo");
		menuArquivo.setMnemonic(KeyEvent.VK_A);
		menuCodigo = new JMenu("Código");
		menuNovo = new JMenu("Novo");
		menuItemFechar = new JMenuItem("Fechar");
		menuItemFechar.setMnemonic(KeyEvent.VK_F);
		menuItemFechar.addActionListener(new MenuItemFecharHandler());
		menuItemNovoProjeto = new JMenuItem("Projeto");
		menuItemNovoProjeto.addActionListener(new MenuItemNovoProjetoHandler());
		menuItemNovoClasse = new JMenuItem("Classe");
		menuItemAutoCompletar = new JCheckBoxMenuItem("Auto completar", true);
		menuItemTexto = new JRadioButtonMenuItem("Somente Texto");
		menuItemHtml = new JRadioButtonMenuItem("HTML");
		group = new ButtonGroup();
		group.add(menuItemTexto);
		group.add(menuItemHtml);
		menuNovo.add(menuItemNovoProjeto);
		menuNovo.add(menuItemNovoClasse);
		menuArquivo.add(menuNovo);
		menuArquivo.add(menuItemFechar);
		menuCodigo.add(menuItemAutoCompletar);
		menuCodigo.addSeparator();
		menuCodigo.add(menuItemTexto);
		menuCodigo.add(menuItemHtml);
		menuBar.add(menuArquivo);
		menuBar.add(menuCodigo);
		setVisible(true);
	}
	private class MenuItemFecharHandler implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			System.exit(0);
		}
	}
	private class MenuItemNovoProjetoHandler implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			JOptionPane.showMessageDialog(null, "Novo projeto...");
		}
	}
//main ...
```

Criamos uma barra de menu com dois menus: Arquivo e Código. Dentro do menu Arquivo criamos um sub-menu “Novo”. Dentro de código criamos um item de menu do tipo checkbox, e outros dois agrupados. Usamos o “`setMenuBar(menuBar)`” para definir o menu da janela. O clica no item do menu gera uma ActionListener.

JToolBar

A barra de ferramentas é usada para agrupar botões e controles que dêem acesso rápido e prático as funcionalidades mais usadas do sistema. Para tal, usamos a classe JToolBar, que além de agrupar os componentes, permite por exemplo ser posicionado em outras partes de janela e até fora dela.

Abaixo, exemplo de barra de ferramentas agrupando botões:

```
public class ExemploJToolBar extends JFrame {
	private JToolBar toolBar;
	private JButton btExcluir;
	private JButton btPesquisar;
	private JButton btInformacoes;
	private JButton btFechar;
	public ExemploJToolBar() {
		super("Exemplo de ToolBar");
		setSize(400, 150);
		setLayout(new BorderLayout());
		toolBar = new JToolBar("Barra de Ferramentas");
		toolBar.setRollover(true);
		btExcluir = new JButton(new ImageIcon("images/excluir.png"));
		btExcluir.setToolTipText("Excluir");
		btPesquisar = new JButton(new ImageIcon("images/pesquisar.png"));
		btInformacoes = new JButton(new ImageIcon("images/informacoes.png"));
		btFechar = new JButton(new ImageIcon("images/fechar.png"));
		btFechar.addActionListener(new BtFecharHandler());

		toolBar.add(btExcluir);
		toolBar.add(btPesquisar);
		toolBar.add(btInformacoes);
		toolBar.addSeparator();
		toolBar.add(btFechar);
		add(toolBar, BorderLayout.NORTH);
		setVisible(true);
	}
	private class BtFecharHandler implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			System.exit(0);
		}
	}
//main...
}
```

São criados alguns botões com ícones que são adicionados à barra de ferramentas. Entre os dois últimos incluímos um espaço com "`toolBar.addSeparator();`". Para dar um efeito legal ao passar o mouse sobre os botões configuramos para "`toolBar.setRollover(true);`". Ao final adicionar um gerenciador de eventos para o botão Fechar e adicionamos a barra na parte norte da janela.

JList

Com o JList podemos criar caixas de listagem contendo vários itens, permitindo ao usuário selecionar um ou vários deles.

A seguir, um exemplo utilizando JList:

```
public class ExemploJList extends JFrame {
	private String[] regioes ={"Centro-Oeste", "Nordeste", "Norte", "Sudeste", "Sul"
};
	private JLabel lbLista;
	private JList lista;
	private JButton exibir;
	public ExemploJList() {
		super("Exemplo de JList");
		setSize(400, 200);
		setLayout(null);
		setLocationRelativeTo(null);
		lbLista = new JLabel("Lista de regiões:");
		lista = new JList(regioes);
		lista.setVisibleRowCount(3);
		lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
		exibir = new JButton("Exibir");
		exibir.addActionListener(new ExibirHandler());
		JScrollPane barraDeRolagem = new JScrollPane(lista);
		lbLista.setBounds(20, 20, 100, 20);
		barraDeRolagem.setBounds(20, 50, 200, 56);
		exibir.setBounds(20, 140, 100, 20);
		Container cp = getContentPane();
		cp.add(lbLista);
		cp.add(barraDeRolagem);
		cp.add(exibir);
		setVisible(true);
	}
	private class ExibirHandler implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			JOptionPane.showMessageDialog(null, lista.getSelectedValue());
		}

	}
//main ...
}
```

Ao criar a lista atribuímos à ela o vetor com os itens a serem exibidos "`lista = new JList(regioes);`". Definimos quantos itens estarão visíveis "`lista.setVisibleRowCount(3);`", e o tipo de seleção "`lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);`". Também adicionamos a lista a uma barra de rolagem para que possamos ter acesso aos itens não visíveis "`JScrollPane barraDeRolagem = new JScrollPane(lista);`", e é esta barra de rolagem que será adicionada a janela.

JComboBox

Semelhante a lista, porém o JComboBox mantém visível apenas o item selecionado, para acessar os demais basta clicar e a lista se abre.

O próximo código exemplifica a utilização do JComboBox:

```
public class ExemploJComboBox extends JFrame {
        private String[] regioes ={"", "Centro-Oeste", "Nordeste", "Norte", "Sudeste",
"Sul" };
        private JLabel lbLista;
        private JComboBox comboBox;
        private JButton exibir;
        public ExemploJComboBox() {
                super("Exemplo de JComboBox");
                setSize(400, 200);
                setLayout(new FlowLayout());
                setLocationRelativeTo(null);
                lbLista = new JLabel("Lista de regiões:");
                comboBox = new JComboBox(regioes);
                exibir = new JButton("Exibir");
                exibir.addActionListener(new ExibirHandler());
                Container cp = getContentPane();
                cp.add(lbLista);
                cp.add(comboBox);
                cp.add(exibir);
                setVisible(true);
        }
        private class ExibirHandler implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        JOptionPane.showMessageDialog(null, comboBox.getSelectedItem());
                }

        }
//main ...
}
```

Componente simples de interagir, definimos a lista na criação "`comboBox = new JComboBox(regioes);`", e para pegar a seleção usamos "`comboBox.getSelectedItem()`".

JPasswordField

O Swing possui uma classe específica para permitir a entrada de senhas pelo usuário, a JPasswordField. Vejamos abaixo um exemplo:

```
public class ExemploJPasswordField extends JFrame {
        private JLabel lbSenha;
        private JPasswordField senha;
        private JButton exibir;
        public ExemploJPasswordField() {
                super("Exemplo de JPasswordField");
                setSize(400, 200);
                setLayout(new FlowLayout());
                setLocationRelativeTo(null);
                lbSenha = new JLabel("Digite a senha:");
                senha = new JPasswordField(15);
                exibir = new JButton("Exibir");
                exibir.addActionListener(new ExibirHandler());
                add(lbSenha);
                add(senha);
                add(exibir);
                setVisible(true);
        }
        private class ExibirHandler implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        JOptionPane.showMessageDialog(null, new
String(senha.getPassword()));
                }
        }
//main…
}
```

O texto digitado na caixa fica oculto, e para recuperar o conteúdo realmente digitado precisamos usar "`senha.getPassword()`", se usarmos o comando normal do JTextField recuperaremos uma seqüência de "bolinhas". Para trocar o caractere que

aparece no lugar do texto digitado, por exemplo, pelo comum "*" (asterisco), usamos o método "`senha.setEchoChar('*');`".

JFormattedText

Com o JFormattedTextField podemos definir um padrão de entrada de texto a ser seguido, por exemplo, datas, CEP, CPF/CNPJ, telefone, etc. Abaixo a lista de caracteres que podem ser usados na formatação:

Símbolo	Valor válido
#	Número
?	Letra
A	Letra ou número
*	Qualquer conteúdo
U	Letra maiúscula
L	Letra minúscula

O código abaixo, traz alguns exemplos de implementação com o JFormattedTextField:

```
public class ExemploJFormattedTextField extends JFrame {
        private JLabel lbCep; private JLabel lbFone;
        private JLabel lbData;private JLabel lbCPF;
        private JLabel lbIdade;       private JLabel lbPeso;
        private JFormattedTextField cep;     private JFormattedTextField fone;
        private JFormattedTextField data;    private JFormattedTextField cpf;
        private JFormattedTextField idade;   private JFormattedTextField peso;
        private MaskFormatter maskCep;       private MaskFormatter maskFone;
        private MaskFormatter maskCpf;       private MaskFormatter masData;
        private NumberFormatter maskIdade;   private NumberFormatter maskPeso;
        public ExemploJFormattedTextField() {
                super("Exemplo de JFormattedTextField");
                setSize(400, 200);
                setLayout(new FlowLayout());
                setLocationRelativeTo(null);
                lbCep = new JLabel("CEP:");
                lbFone = new JLabel("Fone:");
                lbData = new JLabel("Data:");
                lbCPF = new JLabel("CPF:");
                lbIdade = new JLabel("Idade:");
                lbPeso = new JLabel("Peso:");
                try {
                        maskCep = new MaskFormatter("#####-###");
                        maskFone = new MaskFormatter("(##) ####-####");
                        masData = new MaskFormatter("##/##/####");
                        maskCpf = new MaskFormatter("###.###.###-##");
                        maskIdade = new
NumberFormatter(NumberFormat.getIntegerInstance());
                        maskPeso = new NumberFormatter(NumberFormat.getNumberInstance());
                } catch (ParseException exp) {}
                maskCep.setPlaceholderCharacter('_');
                maskFone.setPlaceholderCharacter('_');
                masData.setPlaceholderCharacter('_');
                maskCpf.setPlaceholderCharacter('_');
                maskIdade.setAllowsInvalid(false);
                maskIdade.setMaximum(150);
                maskPeso.setMaximum(150.0);
                cep = new JFormattedTextField(maskCep);
                fone = new JFormattedTextField(maskFone);
                data= new JFormattedTextField(masData);
                cpf = new JFormattedTextField(maskCpf);
                idade = new JFormattedTextField(maskIdade);
                idade.setColumns(3);
                peso = new JFormattedTextField(maskPeso);
                peso.setColumns(6);
                add(lbCep);
                add(cep);
                add(lbFone);
                add(fone);
                add(lbData);
                add(data);
                add(lbCPF);
                add(cpf);
                add(lbIdade);
                add(idade);
                add(lbPeso);
                add(peso);

                setVisible(true);
        }
//main …
}
```

Para definir a máscara da caixa de texto usamos uma instância da classe "`MaskFormatter`", com em "maskCep `= new MaskFormatter("#####-###");`" para definir a máscara do CEP. Definimos um caractere para ocupar o lugar dos valores enquanto eles não são inseridos "`maskCep.setPlaceholderCharacter('_');`". Ou então "`NumberFormatter`" para obrigar apenas números como em "`maskIdade = new NumberFormatter(NumberFormat.getIntegerInstance());`" para a idade.

JTextArea

O JTextArea é usado para as situações onde precisamos permitir ao usuário inserir grandes extensões de textos, como em campos de observação. O exemplo abaixo apresenta de forma sucinta como interar com o JTextArea:

```
public class ExemploJTextArea extends JFrame {
        private JLabel lbDestinatairos;
        private JTextArea destinatarios;
        public ExemploJTextArea() {
                super("Exemplo de JTextArea");
                setSize(400, 150);
                setLayout(new FlowLayout());
                lbDestinatairos = new JLabel("Destinatários:");
                destinatarios = new JTextArea(4, 30);
                destinatarios.setLineWrap(true);
                destinatarios.setWrapStyleWord(true);
                JScrollPane barraDeRolagem = new JScrollPane(destinatarios);

                add(lbDestinatairos);
                add(barraDeRolagem);
                setVisible(true);
        }
//main …
}
```

É criado um JScrollPane para gerenciar a rolagem o texto, como já visto anteriormente para o JList. Também configuramos duas propriedades interessantes do JTextArea, a primeira para ir automaticamente para a linha de baixo ao chegar no final da linha atual "`destinatarios.setLineWrap(true);`", e a outra para não dividir as palavras nessa quebra "`destinatarios.setWrapStyleWord(true);`".

BorderFatory

O BorderFatory nos permite criar bordas personalizadas para contornar nossos componentes formando áreas bem definidas em nossa interface.

O exemplo abaixo é uma evolução do exemplo do JTextArea:

```
public class ExemploBorderFactory extends JFrame {
        private JTextArea destinatarios;
        public ExemploBorderFactory() {
                super("Exemplo de BorderFactory");
                setSize(400, 150);
                destinatarios = new JTextArea(4, 30);
                destinatarios.setLineWrap(true);
                destinatarios.setWrapStyleWord(true);
                JScrollPane barraDeRolagem = new JScrollPane(destinatarios);
                barraDeRolagem.setBorder(BorderFactory.createCompoundBorder(
                                BorderFactory.createTitledBorder("Destinatarios:"),
                                BorderFactory.createEmptyBorder(4,4,4,4)));

                add(barraDeRolagem);
                setVisible(true);
        }
//main …
}
```

Sai fora o JLabel, e definimos ao JScrollPane uma borda com título e linha, deixando um espaçamento de 4 pixels.

JCheckBox

O JCheckBox é usado para situações em que necessitamos que uma informação esteja marcada ou não. Abaixo exemplo de sua utilização:

```
public class ExemploJCheckBox extends JFrame {
	private JLabel lbEsportes;
	private JLabel lbResultado;
	private JCheckBox cbFutebol;
	private JCheckBox cbVolei;
	private JCheckBox cbNatacao;
	private JCheckBox cbTenis;
	public ExemploJCheckBox() {
		super("Exemplo de JCheckBox");
		setSize(400, 200);
		setLayout(new GridLayout(6, 1));
		setLocationRelativeTo(null);
		lbEsportes = new JLabel("Quais esportes você curte:");
		lbResultado = new JLabel();
		cbFutebol = new JCheckBox("Fubebol");
		cbVolei = new JCheckBox("Volei");
		cbNatacao = new JCheckBox("Natação");
		cbTenis = new JCheckBox("Tênis");
		ActionListener actionListener = new EsportesCheckBoxHandler();
		cbFutebol.addActionListener(actionListener);
		cbVolei.addActionListener(actionListener);
		cbNatacao.addActionListener(actionListener);
		cbTenis.addActionListener(actionListener);
		add(lbEsportes);
		add(cbFutebol);
		add(cbVolei);
		add(cbNatacao);
		add(cbTenis);
		add(lbResultado);
		setVisible(true);
	}
	private class EsportesCheckBoxHandler implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			StringBuilder resultado = new StringBuilder();
			resultado.append("Resultado: ");
			if (cbFutebol.isSelected()) {
				resultado.append("   [futebol]");
			}
			if (cbVolei.isSelected()) {
				resultado.append("   [volei]");
			}
			if (cbNatacao.isSelected()) {
				resultado.append("   [natação]");
			}
			if (cbTenis.isSelected()) {
				resultado.append("   [tênis]");
			}
			lbResultado.setText(resultado.toString());
		}
	}
//main…
}
```

Podemos marcar um, vários ou todos. Eles são independentes entre si. Um gerenciador de eventos gera o resultado na parte inferior.

JRadioButton

Com o JRadioButton podemos disponibilizar um grupo de caixas de seleção semelhante ao JCheckBox, porém apenas uma das opções vai estar marcada por vez. Vejamos o código abaixo:

```
public class ExemploJRadioButton extends JFrame {
	private JLabel lbEsportes;
	private JLabel lbResultado;
	private JRadioButton rbFutebol;
	private JRadioButton rbVolei;
	private JRadioButton rbNatacao;
	private JRadioButton rbTenis;
	private ButtonGroup grupo;
	private JButton btResultado;
	public ExemploJRadioButton() {
		super("Exemplo de JRadioButton");
		setSize(400, 200);
		setLayout(new GridLayout(7, 1));
		setLocationRelativeTo(null);
		lbEsportes = new JLabel("Qual seu esporte favorito:");
		lbResultado = new JLabel();
		rbFutebol = new JRadioButton("Fubebol");
		rbVolei = new JRadioButton("Volei");
		rbNatacao = new JRadioButton("Natação");
		rbTenis = new JRadioButton("Tênis");
		btResultado = new JButton("Resultado");
		btResultado.addActionListener(new BtResultadoHandler());
		grupo = new ButtonGroup();
		grupo.add(rbFutebol);
		grupo.add(rbVolei);
		grupo.add(rbNatacao);
		grupo.add(rbTenis);
		add(lbEsportes);
		add(rbFutebol);
		add(rbVolei);
		add(rbNatacao);
		add(rbTenis);
		add(btResultado);
		add(lbResultado);
		setVisible(true);
	}
	private class BtResultadoHandler implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			StringBuilder resultado = new StringBuilder();
			resultado.append("Resultado: ");
			if (rbFutebol.isSelected()) {
				resultado.append("   [futebol]");
			}
			if (rbVolei.isSelected()) {
				resultado.append("   [volei]");
			}
			if (rbNatacao.isSelected()) {
				resultado.append("   [natação]");
			}
			if (rbTenis.isSelected()) {
				resultado.append("   [tênis]");
			}
			lbResultado.setText(resultado.toString());
		}
	}
//main …
}
```

O responsável por garantir que apenas um componente esteja selecionado por vez é o ButtonGroup, bastando adicionar os componentes ao mesmo grupo.

JToogleButton

São usados como botões que possuem dois estados: liga, desliga. Podem ser independentes ou, participarem de um grupo onde podemos ter apenas um ligado por vez. O exemplo abaixo apresenta as duas situações:

```
...
        private class FormatarTextoHandler implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        if (btAlinharEsquerda.isSelected()) {
                                label.setHorizontalAlignment(SwingConstants.LEFT);
                        } else if (btAlinharCentro.isSelected()) {
                                label.setHorizontalAlignment(SwingConstants.CENTER);
                        } else if (btAlinharDireita.isSelected()) {
                                label.setHorizontalAlignment(SwingConstants.RIGHT);
                        }
                        int estilo = Font.PLAIN;
                        if (btNegrito.isSelected()) {
                                estilo += Font.BOLD;
                        }
                        if (btItalico.isSelected()) {
                                estilo += Font.ITALIC;
                        }
                        fnt = new Font(FONT_TIPO, estilo, FONT_TAMANHO);
                        if (btSublinhado.isSelected()) {
                                Map map = fnt.getAttributes();
                                map.put(TextAttribute.UNDERLINE,
TextAttribute.UNDERLINE_ON);
                                fnt = new Font(map);
                        }
                        label.setFont(fnt);
                }
        }
//main …
}
```

Os botões de alinhamento formam um grupo e apenas um pode estar ligado por vez. Os botões de estilo são independentes e podem ter nenhum, um, vários ou todos marcados ao mesmo tempo.

JTable

Com o JTable podemos criar listagens com várias colunas nomeadas. As colunas podem ser trocadas de posição pelo usuário. O código abaixo exemplifica como interagir com esse componente:

```
public class ExemploJTable extends JFrame {
	private Object[][] populacao = {{"São Paulo", 10.8}, {"Rio de Janeiro", 6.0},
{"Salvador", 2.8}, {"Belo Horizonte", 2.4}, {"Fortaleza", 2.4}};
	private String[] colunas = {"Cidade", "População (Milhões)"};
	private JLabel label;
	private JTable table;
	private JButton btExibir;
	public ExemploJTable() {
		super("Exemplo de JTable");
		setSize(400, 150);
		setLayout(new BorderLayout());
		setLocationRelativeTo(null);
		label = new JLabel("Principais capitais:");
		table = new JTable(populacao, colunas);
		btExibir = new JButton("Exibir");
		btExibir.addActionListener(new BtExibirHandler());
		add(label, BorderLayout.NORTH);
		JScrollPane barraDeRolagem = new JScrollPane(table);
		add(barraDeRolagem, BorderLayout.CENTER);
		add(btExibir, BorderLayout.SOUTH);
		setVisible(true);
	}
	private class BtExibirHandler implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			JOptionPane.showMessageDialog(null,
table.getValueAt(table.getSelectedRow(), table.getSelectedColumn()));
		}
	}
//main …
}
```

Assim com ocorre com o JList, o JTable deve ser adicionar a um JScrollPane para que possa haver navegação no caso da tabela conter mais itens que o pode ser exibido.

5. JANELAS

JFrame

O JFrame é o componente responsável por gerar as janelas, como as conhecemos em qualquer sistema operacional, com opções de barra de título, borda, minimizar, maximizar, fechar, mover, etc. É nele que serão inseridos os demais componentes para a composição de nossas janelas.

Exemplo de janela simples com JFrame:

```
public class ExemploJFrame extends JFrame {
	public ExemploJFrame() {
		super("Primeira Janela com JFrame");
		setSize(300, 150);
		setVisible(true);
	}
	public static void main(String args[]){
		ExemploJFrame app = new ExemploJFrame();
		app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Definimos o título da janela "`super("Primeira Janela com JFrame")`", o tamanho "`setSize(300, 150)`", e dizemos que a janela deve ser exibida "`setVisible(true)`". Importante, sempre que desejarmos exibir a janela precisamos obrigatoriamente chamar esse método. Para que o programa encerre ao fechar a janela, usamos "`setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`".

O aspecto e o comportamento da janela podem ser personalizados através da manipulação dos atributos do JFrame. Vamos conhecer alguns deles, aplicando os incrementos abaixo ao nosso código:

Alterar a maneira como a janela inicia a apresentação, por exemplo, maximizada: `setExtendedState(MAXIMIZED_BOTH);`

Definir se a janela pode ou não ser redimensionada: `setResizable(false);`

Define a posição da janela, relativo a outro componente, quando "null" centraliza: `setLocationRelativeTo(null);`

Podemos definir as cores de fundo e frente, por exemplo, definindo a cor de fundo para azul: `getContentPane().setBackground(Color.blue);`

Definir um ícone para a janela: `ImageIcon icone = new ImageIcon("images/teste.gif"); setIconImage(icone.getImage());`

JDialog

Frame usado independente de frame "pai" que pode ser ou não modal. Se for modal, significa que bloqueará o acesso às janelas do fundo. O JDialog é a base para a construção das telas que visualizamos utilizando o JOptionPane.

Em suma, o JDialog tem várias das características do JFrame. Segue exemplo de utilização do JDialog.

```
public class ExemploJDialog extends JFrame implements ActionListener {
        private JButton btDialog;
        public ExemploJDialog() {
                super("Exemplo JDialog");
                setSize(300, 250);
                setLayout(new FlowLayout());
                setLocationRelativeTo(null);
                btDialog = new JButton("JDialog");
                btDialog.addActionListener(this);
                Container cp = getContentPane();
                cp.add(btDialog);
                setVisible(true);
        }
        public void actionPerformed(ActionEvent e) {
                JDialog dialog = new JDialog(this, "Título");
                dialog.setModal(true);
                dialog.setLocationRelativeTo(null);
                dialog.add(new JLabel("Teste de texto no JDialog"));
                dialog.pack();
                dialog.setVisible(true);
        }
//main ...
}
```

JInternalFrame

O JInternalFrame é conhecido por ser usado como "Janela Filha", ou seja, temos uma janela principal "pai" que pode abrigar várias outras janelas, e estas outras são justamente do tipo JInternalFrame.

O uso de janelas filhas está relacionado ao conceito de MDI – Multiple document interface – vários documentos abertos sobre uma mesma janela "pai". A outra forma de interface é o SDI – Single document interface – as janelas abrem como se fossem aplicativos diferentes.

Para utilizar várias janelas filha ao mesmo tempo precisamos do auxílio do JDesktopPane que irá gerenciar essa situação.

Segue exemplo de utilização de janelas filha.

```
public class ExemploJInternalFrame extends JFrame implements ActionListener {
//...
	private JDesktopPane desktopPane;
	public ExemploJInternalFrame() {
		super("Exemplo de JInternalFrame");
		setSize(400, 150);
		setLayout(new BorderLayout());
		setExtendedState(MAXIMIZED_BOTH);
//...
		desktopPane = new JDesktopPane();
		add(desktopPane, BorderLayout.CENTER);
		setVisible(true);
	}
	public void actionPerformed(ActionEvent e) {
		ProjetoInternalFrame projetoInternalFrame = new ProjetoInternalFrame();
		projetoInternalFrame.setVisible(true);
		desktopPane.add(projetoInternalFrame);
		projetoInternalFrame.moveToFront();
	}
	private class ProjetoInternalFrame extends JInternalFrame  {
		public ProjetoInternalFrame() {
			super("Janela Filha", true, true, true, true);
			setSize(400,300);
			setVisible(true);
		}
	}
//...
}
```

6. PROJETOS DE SOFTWARE

MVC – Model View Controler

O MVC é um padrão de criação de projetos que separa o modelo da visão usando um controlador. O modelo representa o negócio, compreendido pelo domínio, DAO (Data Acces Object). A visão é a interface com o usuário. E, o controlador é responsável por fazer o meio de campo entre o modelo e a visão. Dessa forma as regras de negócio não ficam espalhadas pelo código se misturando com os componentes das janelas.

Algumas implementações de MVC podem ser consideradas MVcC, onde a visão (View) é compreendida por uma visão e um controlador próprio da visão.

Uma implementação da MVC pode ir de uma simples separação de responsabilidades em classes diferentes, evoluindo para pacotes bem definidos, chegando ao ponto de ser altamente configurável, com baixo nível de acoplamento, e em outros casos ainda permitindo interferência a nível de produção. Algumas destas implementações usam xml e o padrão Factory.

Vamos nos ater a divisão de responsabilidades em pacotes, que é bem mais simples, e já permite a construção de softwares com ótimo nível de organização. Vejamos abaixo alguns exemplos e comentários.

View (visão): criamos o pacote "gui" para colocar todas as classes referentes às telas que queremos apresentar ao usuário.

Controler (controlador): criamos o pacote "service" para inserir as classes que farão a ligação entre ao modelo e a visão.

Model (modelo): a parte de modelo dividimos em dois pacotes "domain" e "dao". O "domain", é onde ficarão os domínios, ou seja, os nossos objetos de interesse (POJOS), que representam as nossas entidades (como cliente, produto, etc). No "dao" colocamos as classes que farão a persistência e a consulta dos dados da aplicação no banco de dados.

Framework

Dentre as atribuições do Framework temos a criação de componentes personalizados e a criação da estrutura base do sistema.

A sugestão é que não se utilize diretamente os componentes do Swing, mas sim, nossos componentes que estendem os componentes Swing. Dessa forma podemos acrescentar características aos componentes que serão refletidas em todo o sistemas alterando apenas o nosso componente personalizado. Abaixo ilustramos uma personalização simplificada para o JButton, mas isso serve e é sugerido para todos os componentes, janelas e eventos utilizados em nosso sistema.

```
public class MeuBotao extends JButton {

	public MeuBotao(String texto) {
		super(texto);
		setForeground(new Color(255, 140, 200));
		setBackground(Color.BLACK);
		Font f = getFont();
		f.deriveFont(20);
		f.deriveFont(Font.BOLD);
	}
}
```

Normalmente o Framework é compartilhado entre vários projetos, por isso, é conveniente que ele seja um projeto separado, e seja incorporados aos projetos como um arquivo ".jar". A geração de arquivos ".jar" será tratada a seguir.

JAR

Arquivos ".jar" são o padrão para distribuição de frameworks, api e atém aplicações Java SE. Os arquivos ".jar" nada mais é que uma forma organizada de guardar as classes.

Existem várias formas de gerar um arquivo ".jar". Vamos utilizar o Ant, que automatiza esse processo, bastando que a gente configure apenas um arquivo xml. Conforme segue exemplo abaixo.

```
<?xml version="1.0"?>
<project name="Framework" default="jar" basedir=".">
	<property name="name" value="Framework"/>
	<property name="src" value="./src"/>
	<property name="build" value="build"/>
	<property name="bin" value="bin"/>
	<target name="compile">
		<mkdir dir="${build}"/>
		<javac srcdir="${src}" includes="**/*.java" destdir="${build}" debug="on"
deprecation="on"/>
	</target>
	<target name="jar" depends="compile" description="cria um .jar file ">
		<delete dir="${bin}"/>
		<mkdir dir="${bin}"/>
		<jar jarfile="${bin}/${name}.jar" basedir="${build}">
		</jar>
	</target>
</project>
```

JavaDoc

O JavaDoc é a documentação padrão de código fonte utilizado com Java. Para documentar o código fonte de forma que ele apareça na geração do JavaDoc utilize a sintaxe:

/**

* <insira aqui os comentários necessários>

*/

Procurando na literatura encontraremos os padrões relacionados a sintaxe e formas de documentação.

Para gerar a documentação acesse no Eclipse o menu "Project→Generate Javadoc...". O resultado será uma coleção de arquivo html.

Projeto

Na criação de qualquer projeto devemos considerar a existência de um framework, que além dos componentes personalizados, conterá a estrutura base de interligação entre os vários estágios do MVC, implementando o que for possível para dar agilidade no desenvolvimento.

A seguir vamos ver um exemplo simples, com estrutura básica e fácil utilização. O projeto consiste em listar as pessoas armazenadas em um banco de dados.

Classe base para os domínios, candidata ao framework:

```
public class DomainBase {
}
```

Domínio da entidade pessoa:

```
public class Pessoa extends DomainBase {
	private Integer cdPessoa;
	private String nmPessoa;
	@Override
	public String toString() {
		return getNmPessoa();
	}
//getters and setters
}
```

Classe base para acesso ao banco de dados, candidata ao framework:

```
public abstract class ConexaoBase {
	private Connection con = null;
	protected abstract String getDriver();
	protected abstract String getUrl();
	protected abstract String getUser();
	protected abstract String getPassword();
	public Connection getConnection() throws Exception {
		if (con == null) {
			Class.forName(getDriver());
			con = DriverManager.getConnection(getUrl(), getUser(),
getPassword());
			System.out.println("Conexão aberta com sucesso");
		}
		return con;
	}
	public void closeConnection() throws Exception {
		if (con != null) {
			con.close();
			con = null;
		}
	}
}
```

Classe para conexão com o banco de dados da nossa aplicação:

```
public class Conexao extends ConexaoBase {
	private static Conexao instance = null;
	private Conexao() {
	}
	public static Conexao getInstance() {
		if (instance == null) {
			instance = new Conexao();
		}
		return instance;
	}
	@Override
	protected String getDriver() {
		return "com.mysql.jdbc.Driver";
	}
	@Override
	protected String getPassword() {
		return "mysql";
	}
	@Override
	protected String getUrl() {
		return "jdbc:mysql://localhost:3306/meubanco";
	}
	@Override
	protected String getUser() {
		return "root";
	}
}
```

Classe base para o acesso aos dados, candidata ao framework:

```
public abstract class DaoBase {
	private ConexaoBase conexaoBase;
	public abstract String getNomeDaTabela();
	public abstract Object populaDominio(ResultSet rs) throws Exception;
	public DaoBase(ConexaoBase conexaoBase) {
		setConexaoBase(conexaoBase);
	}
	@SuppressWarnings("unchecked")
	public List findAll() throws Exception {
		Connection con = conexaoBase.getConnection();
		try {
			Statement stmt = con.createStatement();
			ResultSet rs = stmt.executeQuery("select * from " +
getNomeDaTabela());
			List list = new ArrayList();
			while (rs.next()) {
				list.add(populaDominio(rs));
			}
			return list;
		} finally {
			con.close();
		}
	}
	protected ConexaoBase getConexaoBase() {
		return conexaoBase;
	}
	protected void setConexaoBase(ConexaoBase conexaoBase) {
		this.conexaoBase = conexaoBase;
	}
}
```

Classe de acesso aos dados da entidade pessoa:

```
public class PessoaDao extends DaoBase {
	public PessoaDao() {
		super(Conexao.getInstance());
	}
	@Override
	public String getNomeDaTabela() {
		return "pessoa";
	}
	@Override
	public Object populaDominio(ResultSet rs) throws Exception {
		Pessoa pessoa = new Pessoa();
		pessoa.setCdPessoa(rs.getInt("cdPessoa"));
		pessoa.setNmPessoa(rs.getString("nmPessoa"));
		return pessoa;
	}
}
```

Classe base para ligação entre o acesso aos dados e a visualização, candidata ao framework:

```
public class ServiceBase {
	private DaoBase daoBase;
	public ServiceBase(DaoBase daoBase) {
		setDaoBase(daoBase);
	}
	protected DaoBase getDaoBase() {
		return daoBase;
	}
	protected void setDaoBase(DaoBase daoBase) {
		this.daoBase = daoBase;
	}
	public List findAll() throws Exception {
		return getDaoBase().findAll();
	}
}
```

Classe que faz a ligação entre o acesso aos dados da entidade pessoa e a visualização dos dados da entidade pessoa:

```
public class PessoaService extends ServiceBase {
	public PessoaService() {
		super(new PessoaDao());
	}
}
```

Classe base para apresentar uma lista de dados na tela, candidata ao framework:

```
public abstract class ListaBase extends JFrame {
	private ServiceBase serviceBase;
	private JList lsDomainBase;
	protected abstract ServiceBase createServiceBase();
	public ListaBase() {
		super("");
		setSize(400, 300);
	}
	protected ServiceBase getServiceBase() {
		if (serviceBase == null) {
			serviceBase = createServiceBase();
		}
		return serviceBase;
	}
	protected void setServiceBase(ServiceBase serviceBase) {
		this.serviceBase = serviceBase;
	}
	public void lista() {
		try {
			List list = getServiceBase().findAll();
			lsDomainBase = new JList(list.toArray());
			mostraJanela();
		} catch (Exception ex) {
			JOptionPane.showMessageDialog(null, ex.getMessage(), "Erro ao
executar essa ação", JOptionPane.ERROR_MESSAGE, null);
		}
	}
	private void mostraJanela() {
		add(lsDomainBase, BorderLayout.CENTER);
		setVisible(true);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Classe que lista os registros da entidade pessoa:

```
public class ListaPessoaBase extends ListaBase {
	@Override
	protected ServiceBase createServiceBase() {
		return new PessoaService();
	}
}
```

Para acionar tudo isso, vamos implementar uma classe main:

```
        public static void main(String[] args) {
                ListaPessoaBase listaPessoaBase = new ListaPessoaBase();
                listaPessoaBase.lista();
                try {
                        Conexao.getInstance().closeConnection();
                } catch (Exception ex) {
                        System.out.println(ex.getMessage());
                }

        }
```

7. REFERÊNCIAS

Livros

Java Como Programar 6ª Edição

Java 2 – Fundamentos Swing e JDBC, 2º Ed

Sun

http://www.sun.com/

JavaDoc da API Java

http://java.sun.com/j2se/javadoc/

Conversões de código Java

http://java.sun.com/docs/codeconv/

Tutorial Swing Oficial da Sun

http://java.sun.com/docs/books/tutorial/uiswing/index.html

JCP – Java Community Process

http://jcp.org/

Eclipse

http://www.eclipse.org/

NetBeans

http://www.netbeans.org/