SWING

1. INTRODUÇÃO

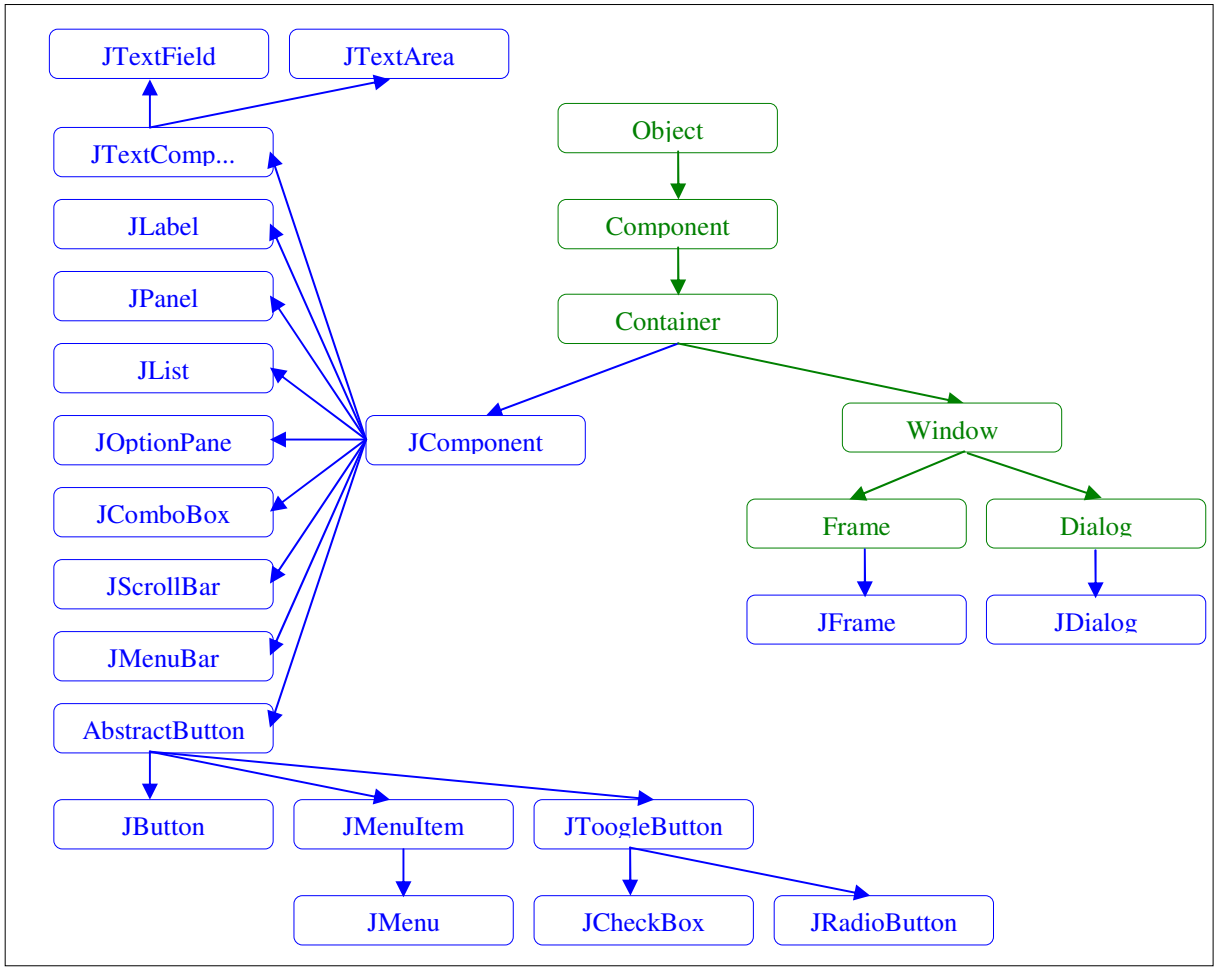
Swing

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.