Módulo 2

Introdução à Programação II

Lição 7

Abstract Window Toolkit e *Swing*

Versão 1.0 - Mar/2007

Autor
Rebecca Ong

Equipe
Joyce Avestro
Florence Balagtas
Rommel Feria
Rebecca Ong
John Paul Petines
Sun Microsystems
Sun Philippines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori
Alexis da Rocha Silva
Aline Sabbatini da Silva Alves
Allan Wojcik da Silva
André Luiz Moreira
Andro Márcio Correa Louredo
Antoniele de Assis Lima
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alberto Vitorino de Almeida
Carlos Alexandre de Sene
Carlos André Noronha de Sousa
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Cleyton Artur Soares Urani
Cristiano Borges Ferreira
Cristiano de Siqueira Pires
Derlon Vandri Aliendres
Fabiano Eduardo de Oliveira
Fábio Bombonato
Fernando Antonio Mota Trinta
Flávio Alves Gomes
Francisco das Chagas
Francisco Marcio da Silva
Gilson Moreno Costa
Givailson de Souza Neves
Gustavo Henrique Castellano
Hebert Julio Gonçalves de Paula
Heraldo Conceição Domingues
Hugo Leonardo Malheiros Ferreira
Ivan Nascimento Fonseca
Jacqueline Susann Barbosa
Jader de Carvalho Belarmino
João Aurélio Telles da Rocha
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Leonardo Borges de Melo
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Lafaiete de Sá Guimarães
Leandro Silva de Morais
Leonardo Leopoldo do Nascimento
Leonardo Pereira dos Santos
Leonardo Rangel de Melo Filardi
Lucas Mauricio Castro e Martins
Luciana Rocha de Oliveira
Luís Carlos André
Luís Octávio Jorge V. Lima
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Manoel Cotts de Queiroz
Marcello Sandi Pinheiro
Marcelo Ortolan Pazzetto
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio Azevedo Gamarra
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Néres Chaves Rebouças
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo José Lemos Costa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Pedro Henrique Pereira de Andrade
Renato Alves Félix
Renato Barbosa da Silva
Reyderson Magela dos Reis
Ricardo Ferreira Rodrigues
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Pereira Machado
Rodrigo Rosa Miranda Corrêa
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Sergio Pomerancblum
Silvio Sznifer
Suzana da Costa Oliveira
Tásio Vasconcelos da Silveira
Thiago Magela Rodrigues Dias
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida
Vastí Mendes da Silva Rocha
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Regina Mariani -** Coordenadora responsável pela parte jurídica
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Mesmo sem conhecer interface gráfica com o usuário ou *GUI* (*Graphical User Interface*) é possível criar uma grande variedade de diferentes projetos. Entretanto, estas aplicações não teriam grandes chances de serem agradáveis aos usuários, tão acostumados aos ambientes gráficos como Windows, Solaris e Linux. Ter um projeto desenvolvido em *GUI* afeta o uso de sua aplicação pois resulta em facilidade de uso e melhor experiência para os usuários de seus aplicativos. Java fornece ferramentas como *Abstract Window Toolkit* (*AWT*) e *Swing* para desenvolver aplicações *GUI* interativas.

Ao final desta lição, o estudante será capaz de:

- Explicar similaridades e diferenças entre *AWT* e *Swing*
- Diferenciar entre os *containers* e componentes
- Criar aplicações *GUI* utilizando *AWT*
- Criar aplicações *GUI* utilizando *Swing*
- Descrever como os gerenciadores de *layout*, tais como *FlowLayout*, *BorderLayout* e *GridLayout*, posicionam os componentes *GUI*
- Criar *layouts* complexos ao elaborar aplicações *GUI*

2. *AWT (Abstract Window Toolkit)* vs. *Swing*

A *JFC* (*Java Foundation Classes*) é uma importante parte de Java SDK. Refere-se a uma coleção de *APIs* que simplificam o desenvolvimento de aplicações Java *GUI*. Consistem basicamente em cinco *APIs* incluindo *AWT* e *Swing*. As outras três *APIs* são *Java2D*, *Accessibility*, e *Drag and Drop*. Todas essas *APIs* dão suporte aos desenvolvedores na criação e implementação de aplicações visualmente destacadas.

Ambas *AWT* e *Swing* dispõem de componentes *GUI* que podem ser usadas na criação de aplicações Java e *applets*. Aprenderemos sobre applets mais tarde. Diferentemente de alguns componentes *AWT* que usam código nativo, *Swing* é escrito inteiramente usando a linguagem de programação Java. Em consequência, *Swing* fornece uma implementação independente de plataforma que assegura que aplicações desenvolvidas em diferentes plataformas tenham a mesma aparência. *AWT*, entretanto, assegura que o *look and feel* (a aparência) de uma aplicação executada em duas máquinas diferentes sejam compatíveis. A *API Swing* é construída sobre um número de *APIs* que implementa várias partes da *AWT*. Como resultado, componentes *AWT* ainda podem ser usados com componentes *Swing*.

3. Componentes *GUI AWT*

3.1. Fundamental Window Classes

No desenvolvimento de aplicações *GUI*, os componentes como os botões ou campos de texto são localizados em *containers*. Essa é uma lista de importantes classes *containers* fornecidas pela *AWT*.

Classe AWT	***Descrição***
Component	Uma classe abstrata para objetos que podem ser exibidos no console e interagir com o usuário. A raiz de todas as outras classes *AWT*.
Container	Uma subclasse abstrata da classe *Component*. Um componente que pode conter outros componentes.
Panel	Herda a classe *Container*. Uma área que pode ser colocada em um *Frame*, *Dialog* ou *Window*. Superclasse da classe *Applet*.
Window	Também herda a classe *Container*. Uma janela *top-level*, que significa que ela não pode ser contida em nenhum outro objeto. Não tem bordas ou barra de menu.
Dialog	Uma janela contendo a barra de título e o botão de fechar, utilizada para criar janelas para comunicação com o usuário.

Frame	Uma janela completa com um título, barra de menu, borda, e cantos redimensionáveis. Possui quatro construtores, dois deles possuem as seguintes assinaturas:
	`Frame()`
	`Frame(String title)`

Tabela 1: Classes Container AWT

Para configurar o tamanho da janela, podemos utilizar o método *setSize*, do seguinte modo:

```
void setSize(int width, int height)
```

Reconfigura o tamanho da janela para o *width* (largura) e *height* (altura) fornecidos como argumentos.

```
void setSize(Dimension d)
```

Reconfigura o tamanho da janela para os atributos *d.width* e *d.height* baseado em um objeto instanciado da classe *Dimension* especificado como argumento.

Por padrão uma janela não é visível a não ser que seja configurada a sua visibilidade para *true*. Esta é a sintaxe para o método *setVisible*:

```
void setVisible(boolean b)
```

Ao criar aplicações *GUI*, o objeto *Frame* é o mais comumente utilizado. Aqui está um exemplo de como criar uma aplicação dessas:

```
import java.awt.*;

public class FrameDemo extends Frame {
   public static void main(String args[]) {
      FrameDemo sf = new FrameDemo();
```

```
            sf.setSize(100, 100); //Tente removendo esta linha
            sf.setVisible(true);  //Tente removendo esta linha
        }
    }
```

Este é o resultado esperado pela execução da classe *SampleFrame*:

Figura 1: Executando SampleFrame

Note que o botão de fechar ainda não funciona porque nenhum mecanismo de suporte a eventos foi adicionado a classe até o momento. Conheceremos sobre suporte a eventos no próximo módulo.

3.2. Graphics

Vários métodos gráficos são encontrados na classe *Graphics*. Aqui está a lista de alguns desses métodos.

drawLine()	drawPolyline()	setColor()
fillRect()	drawPolygon()	getFont()
drawRect()	fillPolygon()	setFont()
clearRect()	getColor()	drawString()

Tabela 2: Alguns métodos da classe Graphics

Relacionada a essa classe está a classe *Color*, que tem três construtores.

Formato do Construtor	***Descrição***
Color(int r, int g, int b)	Valor inteiro de 0 a 255.
Color(float r, float g, float b)	Valor decimal de 0.0 a 1.0.
Color(int rgbValue)	Valor variável de 0 a 2^{24}-1 (preto a branco). Vermelho: bits 16-23 Verde: bits 8-15 Azul: bits 0-7

Tabela 3: Construtores Color

Aqui está uma classe demonstrando a utilização de alguns métodos da classe *Graphics*:

```
import java.awt.*;

public class PanelDemo extends Panel {
   PanelDemo() {
      setBackground(Color.black); //Constante na classe Color
   }
   public void paint(Graphics g) {
      g.setColor(new Color(0,255,0)); // verde
      g.setFont(new Font("Helvetica",Font.PLAIN,16));
      g.drawString("Hello GUI World!", 30, 100);
      g.setColor(new Color(1.0f,0,0)); // vermelho
      g.fillRect(30, 100, 150, 10);
```

```
    }
    public static void main(String args[]) {
       Frame f = new Frame("Testing Graphics Panel");
       PanelDemo painel = new PanelDemo();
       f.add(painel);
       f.setSize(600, 300);
       f.setVisible(true);
    }
}
```

Para um *panel* se tornar visível, ele deve ser colocado em uma janela visível como um *frame*.

Executando o código apresentado temos o seguinte resultado esperado:

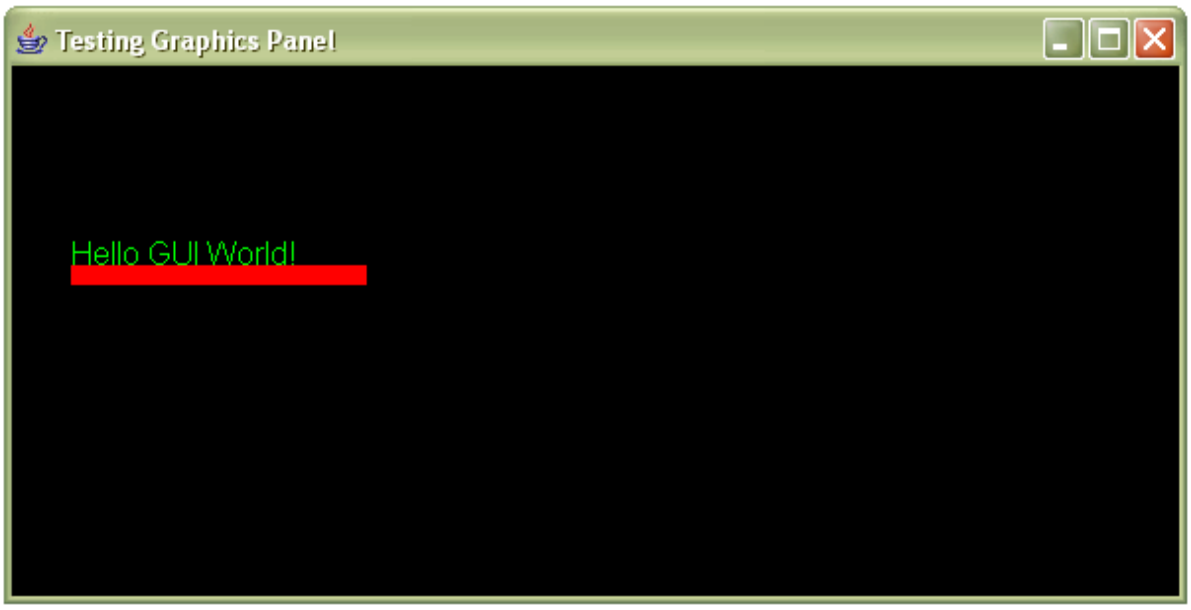


Figura 2: Executando GraphicsPanel

3.3. Mais Componentes AWT

Aqui está uma lista de controles *AWT*. Controles são componentes como botões ou campos textos que permitem ao usuário interagir com a aplicação *GUI*. Essas são todas as subclasses da classe *Component*.

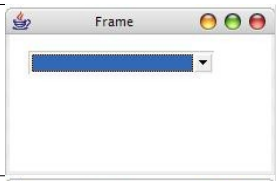
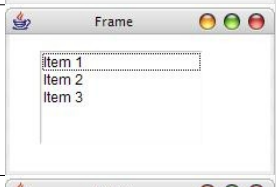
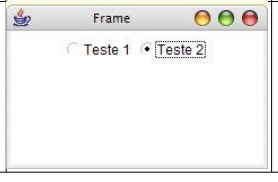
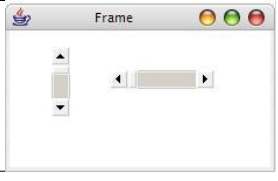
Label		Button		Choice	
TextField		Checkbox		List	
TextArea		CheckboxGroup		Scrollbar	

Tabela 4: Componentes AWT

A seguinte classe cria um *frame* com alguns componentes:

```
import java.awt.*;

class ControlsDemo extends Frame {
   public static void main(String args[]) {
      ControlsDemo fwc = new ControlsDemo();
      fwc.setLayout(new FlowLayout());
      fwc.setSize(600, 100);
      fwc.add(new Button("Test Me!"));
      fwc.add(new Label("Labe"));
      fwc.add(new TextField());
      CheckboxGroup cbg = new CheckboxGroup();
      fwc.add(new Checkbox("chk1", cbg, true));
      fwc.add(new Checkbox("chk2", cbg, false));
      fwc.add(new Checkbox("chk3", cbg, false));
      List list = new List(3, false);
      list.add("MTV");
      list.add("V");
      fwc.add(list);
      Choice chooser = new Choice();
      chooser.add("Avril");
      chooser.add("Monica");
      chooser.add("Britney");
      fwc.add(chooser);
      fwc.add(new Scrollbar());
      fwc.setVisible(true);
   }
}
```

Está e a janela que será mostrada na execução da classe *ControlsDemo*:

Figura 3: Executando ControlsDemo

4. Gerenciadores de *Layout*

A posição e o tamanho dos componentes em cada *container* é determinado pelo gerenciador de *layout*. O gerenciador de *layout* gerencia a disposição dos componentes no *container*. Esses são alguns dos gerenciadores de *layout* incluídos no Java.

- *FlowLayout*
- *BorderLayout*
- *GridLayout*
- *GridBagLayout*
- *CardLayout*
- *BoxLayout*

O gerenciador de *layout* pode ser configurado usando o método *setLayout* da classe *Container*. O método possui a seguinte assinatura:

```
void setLayout(LayoutManager mgr)
```

Caso seja não seja necessário utilizar nenhum gerenciador de *layout*, é possível passar o *layout* do tipo nulo (*NullLayout*) como argumento para esse método, ao se fazer isso será necessário posicionar todos os elementos manualmente com a utilização do método *setBounds* da classe *Component*. Este método possui a seguinte assinatura:

```
public void setBounds(int x, int y, int width, int height)
```

O método controla a posição baseada nos argumentos x (esquerda) e y (topo), e o tamanho *width* (largura) e *height* (altura) *especificados*. Isso seria bastante difícil e tedioso de programar, principalmente ao se possuir um *layout* com diversos objetos *Component* e *Container*. Teríamos de chamar esse método para cada componente, além de conhecer sua determinada posição em *pixels*.

4.1. O Gerenciador FlowLayout

O *FlowLayout* é o gerenciador padrão para a classe *Panel* e suas subclasses, incluindo a classe *Applet*. Ele posiciona os componentes da esquerda para a direita e de cima para baixo, começando no canto superior esquerdo. Imagine como se utilizasse um editor de textos. É assim que o gerenciador *FlowLayout* funciona.

Ele possui três construtores que são como os listados abaixo:

Construtores FlowLayout
`FlowLayout()`
Cria um novo objeto FlowLayout com o alinhamento centralizado e 5 unidades de intervalo horizontal e vertical aplicado aos componentes por padrão.
`FlowLayout(int align)`
Cria um novo objeto FlowLayout com o alinhamento especificado e o intervalo padrão de 5 unidades horizontal e vertical aplicado aos componentes.
`FlowLayout(int align, int hgap, int vgap)`
Cria um novo objeto FlowLayout com o primeiro argumento como o alinhamento aplicado, o intervalo horizontal *hgap* e o itervalo vertical v*gap* aplicado aos componentes.

Tabela 5: Construtores FlowLayout

O intervalo se refere ao espaçamento entre os componentes e é medido em *pixels*. O argumento alinhamento deve ser um dos seguintes:

- FlowLayout.LEFT
- FlowLayout.CENTER

- FlowLayout.RIGHT

Observe a seguinte classe:

```
import java.awt.*;

class FlowLayoutDemo extends Frame {
   public static void main(String args[]) {
      FlowLayoutDemo fld = new FlowLayoutDemo();
      fld.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 10));
      fld.add(new Button("ONE"));
      fld.add(new Button("TWO"));
      fld.add(new Button("THREE"));
      fld.setSize(100, 100);
      fld.setVisible(true);
   }
}
```

O resultado da execução sobre a plataforma Windows é apresentado abaixo.

Figura 4: Executando FlowLayoutDemo

4.2. O Gerenciador BorderLayout

O *BorderLayout* divide o *Container* em cinco partes – *north* (norte), *south* (sul), *east* (leste), *west* (oeste) e *center* (centro). Cada componente é adicionado a uma região específica. As regiões *north* e *south* espalham-se horizontalmente enquanto que as regiões east e west ajustam-se verticalmente. A região centro, por outro lado, ajusta-se em ambos horizontalmente e verticalmente. Esse *layout* é o padrão para objetos *Window*, incluindo as subclasses *Frame* e *Dialog*.

Construtores BorderLayout
`BorderLayout()`
Cria um novo objeto BorderLayout sem nenhum espaçamento aplicado sobre os diferentes componentes.
`BorderLayout(int hgap, int vgap)`
Cria um novo objeto BorderLayout com espaçamento horizontal *hgap* e vertical *vgap* aplicado sobre os diferentes componentes.

Tabela 6: Construtores BorderLayout

Como no gerenciador *FlowLayout*, os parâmetros *hgap* e *vgap* aqui também se referem ao espaçamento entre os componentes no *container*.

Para adicionar um componente a uma região específica, use o método *add* e passe dois argumentos: o componente a ser adicionado e a região onde o componente deve ser posicionado. Note que apenas um componente pode ser colocado em uma região. Adicionar mais de um componente a um *container* resulta em exibir apenas o último componente adicionado. A lista a seguir apresenta as regiões válidas que são campos predefinidos na classe *BorderLayout*.

- BorderLayout.NORTH

- BorderLayout.SOUTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.CENTER

Aqui está uma classe demonstrando como a *BorderLayout* pode ser utilizada:

```
import java.awt.*;

class BorderLayoutDemo extends Frame {
   public static void main(String args[]) {
      BorderLayoutDemo bld = new BorderLayoutDemo();
      bld.setLayout(new BorderLayout(10, 10)); //pode remover
      bld.add(new Button("NORTH"), BorderLayout.NORTH);
      bld.add(new Button("SOUTH"), BorderLayout.SOUTH);
      bld.add(new Button("EAST"), BorderLayout.EAST);
      bld.add(new Button("WEST"), BorderLayout.WEST);
      bld.add(new Button("CENTER"), BorderLayout.CENTER);
      bld.setSize(200, 200);
      bld.setVisible(true);
   }
}
```

Aqui está o resultado desta classe. A segunda figura mostra o efeito do redimensionamento do frame.

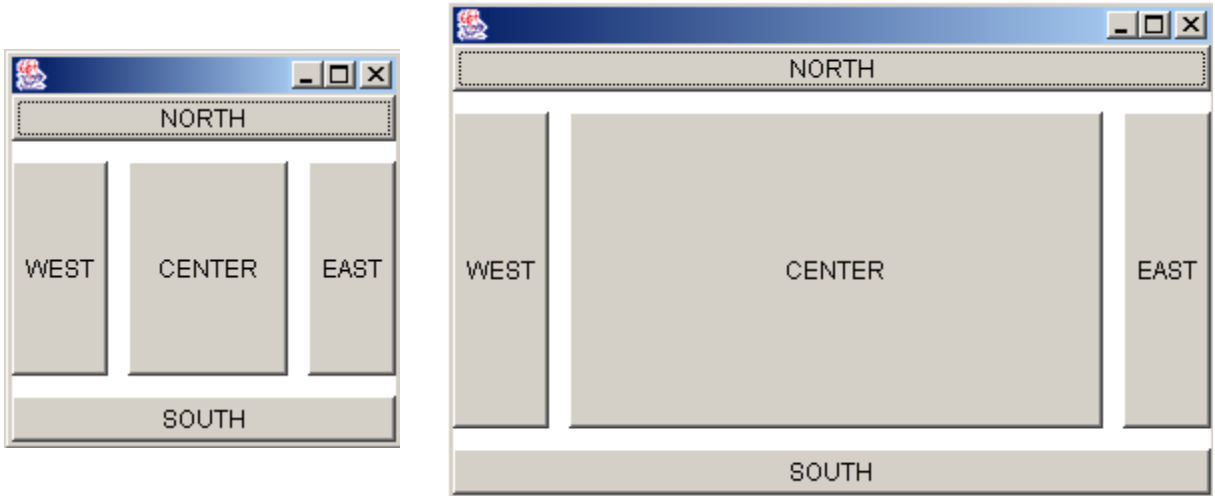


Figura 5: Executando BorderLayoutDemo

4.3. O gerenciador GridLayout

Com o gerenciador *GridLayout*, os componentes também são posicionados da esquerda para a direita e de cima para baixo como no gerenciador *FlowLayout*. Além disso, o gerenciador *GridLayout* divide o *container* em um número de linhas e colunas. Todas essas regiões são do mesmo tamanho. Ele sempre ignora o tamanho preferido do componente.

A seguir, são apresentados os construtores disponíveis para a classe *GridLayout*.

Construtores GridLayout
`GridLayout()`
Cria um novo objeto *GridLayout* com uma única linha e uma única coluna por padrão.
`GridLayout(int rows, int cols)`
Cria um novo objeto *GridLayout* com o número especificado de linhas e colunas.
`GridLayout(int rows, int cols, int hgap, int vgap)`

Cria um novo objeto *GridLayout* com o número especificado de linhas e colunas. Os espaçamentos horizontal *hgap* e vertical v*gap* são aplicados aos componentes.

Tabela 7: Construtores GridLayout

Vejamos a seguinte classe:

```
import java.awt.*;

class GridLayoutDemo extends Frame {
   public static void main(String args[]) {
      GridLayoutDemo gld = new GridLayoutDemo();
      gld.setLayout(new GridLayout(2, 3, 4, 4));
      gld.add(new Button("ONE"));
      gld.add(new Button("TWO"));
      gld.add(new Button("THREE"));
      gld.add(new Button("FOUR"));
      gld.add(new Button("FIVE"));
      gld.setSize(200, 200);
      gld.setVisible(true);
   }
}
```

Esse é o resultado da classe (observe o efeito do redimensionamento sobre o *frame* na segunda figura):

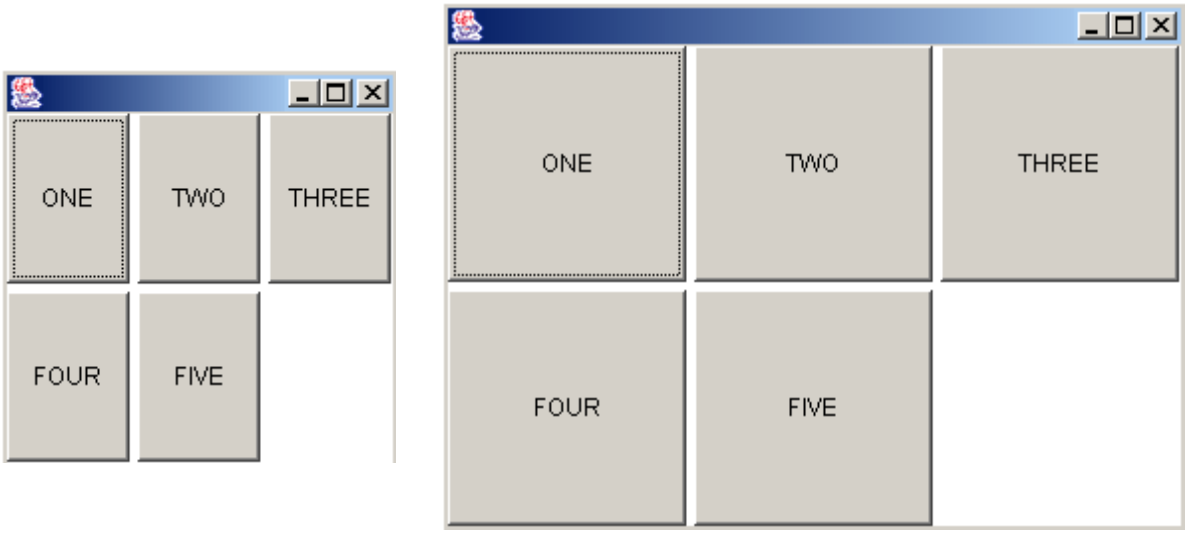


Figura 6: Executando GridLayoutDemo

4.4. Painéis e Layouts Complexos

Para criar *layouts* mais complexos, é possível combinar os diferentes gerenciadores de *layout* com o uso de objetos do tipo *Panel*. Lembre-se que *Panel* é um *Container* e um *Component* ao mesmo tempo. É possível inserir *Component*s em um *Panel* e adicionar este a uma região específica do *Container*.

Observe a técnica utilizada na classe a seguir:

```
import java.awt.*;

class ComplexDemo extends Frame {
   public static void main(String args[]) {
```

```
        ComplexDemo cl = new ComplexDemo();
        Panel panelNorth = new Panel();
        Panel panelCenter = new Panel();
        Panel panelSouth = new Panel();
        /* Painel North */
        //Painéis usam FlowLayout por padrão
        panelNorth.add(new Button("ONE"));
        panelNorth.add(new Button("TWO"));
        panelNorth.add(new Button("THREE"));
        /* Painel Center */
        panelCenter.setLayout(new GridLayout(4,4));
        panelCenter.add(new TextField("1st"));
        panelCenter.add(new TextField("2nd"));
        panelCenter.add(new TextField("3rd"));
        panelCenter.add(new TextField("4th"));
        /* Painel South */
        panelSouth.setLayout(new BorderLayout());
        panelSouth.add(new Checkbox("Choose me!"),
                                          BorderLayout.CENTER);
        panelSouth.add(new Checkbox("I'm here!"),
                                            BorderLayout.EAST);
        panelSouth.add(new Checkbox("Pick me!"),
                                            BorderLayout.WEST);
        /* Adicionando os Panels ao Frame container */
        //Frames usam BorderLayout por padrão
        cl.add(panelNorth, BorderLayout.NORTH);
        cl.add(panelCenter, BorderLayout.CENTER);
        cl.add(panelSouth, BorderLayout.SOUTH);
        cl.setSize(300,300);
        cl.setVisible(true);
    }
}
```

Este é o resultado da classe:

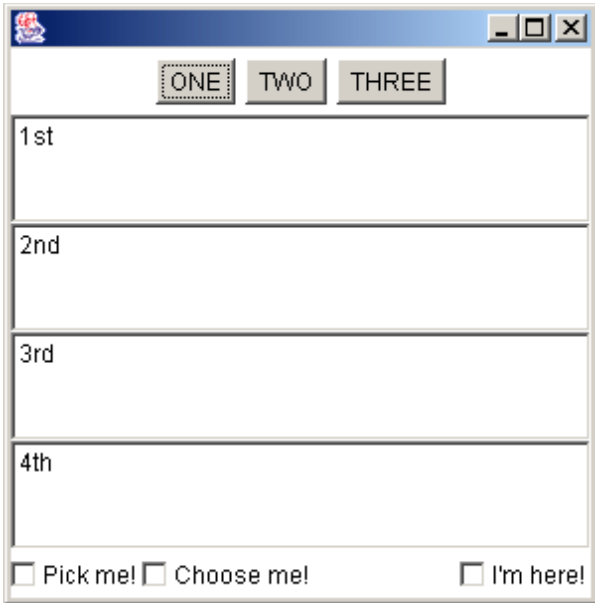


Figura 7: Executando Layout Complexo

5. Componentes *GUI Swing*

Como o pacote *AWT*, o pacote *Swing* fornece classes para criar aplicações *GUI*. O pacote é encontrado em *javax.swing*. A diferença principal entre esses dois é que o componente *Swing* é escrito inteiramente usando Java enquanto o outro não. Como resultado, Projetos *GUI* escritos utilizando classes do pacote *Swing* possuem a mesma aparência mesmo quando executado sobre plataformas completamente diferentes. Além disso, *Swing* fornece componentes mais interessantes como o que permite selecionar cores e a *OptionPane* (painel de opções).

Os nomes dos componentes de *Swing* são quase similares aos componentes *AWT*. Uma diferença óbvia é a convenção de nomes dos componentes. Basicamente, os nomes dos componentes *Swing* são os mesmos nomes dos componentes *AWT* mas com um prefixo em que é adicionado a letra "J". Por exemplo, um componente no *AWT* é a classe *Button*, o mesmo componente correspondente a este no pacote *Swing* é a classe *JButton.* Descrito abaixo, está uma lista de alguns dos componentes *Swing*:

Componente	***Descrição***
JComponent	A classe raiz para todos os componentes *Swing*, excluindo *containers* hierarquicamente superiores.
JButton	Um botão do tipo "pressionar". Corresponde a classe *Button* no pacote *AWT*.
JCheckBox	Um item que pode ser marcado ou desmarcado pelo usuário. Corresponde a classe *Checkbox* no pacote *AWT*.
JFileChooser	Permite ao usuário que selecione um arquivo. Corresponde a classe *FileChooser* no pacote *AWT*.
JTextField	Permite a edição de uma única linha de texto. Corresponde a classe *TextField* no pacote *AWT*.
JFrame	Herda e corresponde a classe *Frame* no pacote *AWT*, mas as duas são ligeiramente incompatíveis em termos de adição de componentes a esse *container*. É preciso pegar o conteúdo do *pane* atual antes de adicionar um componente.
JPanel	Herança de um *JComponent*. É uma classe *container* simples. Corresponde a classe *Panel* no pacote *AWT*.
JApplet	Herança da classe *Applet* no pacote *AWT*. Também ligeiramente incompatível com a classe *Applet* em termos de adição de componentes a esse *container*.
JOptionPane	Herda *JComponent*. Fornece uma maneira fácil de exibir caixas de diálogo pop-up.
JDialog	Herança da classe *Dialog* no pacote *AWT*. Normalmente utilizado para informar o usuário de alguma coisa ou alertá-lo para uma entrada.
JColorChooser	Herda *JComponent*. Permite ao usuário selecionar uma cor.

Tabela 8: Alguns componentes Swing

Para a lista completa de componentes *Swing*, por favor recorra à documentação API.

5.1. Configurando Containers JFrame e JApplet

Como mencionado, os *containers top-level* como o *JFrame* e o *JApplet* no pacote *Swing* são ligeiramente incompatíveis com seus correspondentes *AWT*. Isso em termos de adição de

componentes ao *container*. Ao invés de adicionar diretamente um componente ao *container* como nos *containers AWT*, é necessário primeiro pegar o conteúdo do *pane* do *container*. Para fazer isso utiliza-se o método *getContentPane* do *container*:

```
import javax.swing.*;
import java.awt.*;

class SwingDemo extends JFrame {
   public SwingDemo() {
      super("My First Swing Application");
      this.setSize(300,300);
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.setVisible(true);
   }
   public static void main(String args[]) {
      new SwingDemo();
   }
}
```

Note que o pacote *java.awt* ainda é importado porque os gerenciadores de *layout* em uso são definidos neste pacote. Além disso, dar um título ao *frame* e empacotar os componentes no *frame* também é aplicável para aos *frames AWT*.

Dicas de programação:

1. Compare o estilo de código aplicado neste exemplo para *AWT*.
2. Componentes são declarados como campos, um método *launchFrame* é definido, e a inicialização e adição de componentes são todas feitas no método *launchFrame*.
3. Não herdamos a classe *Frame*, mas a classe *JFrame*.
4. A vantagem de utilizar este estilo se tornará aparente quando chegarmos ao suporte a evento no tópico sobre tratamento de eventos.

Aqui está um resultado demonstrativo:

Figura 8: Executando SwingDemo

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.