Unidade 10: Introdução à Programação de GUIs com Swing

Prof. Daniel Caetano

Objetivo: Apresentar os conceitos inciais de Programação Orientada a Eventos.

Bibliografia: BEZERRA, 2007; JACOBSON, 1992; COAD, 1992.

INTRODUÇÃO

Uma interface gráfica com o usuário (**GUI**) é um meio pelo qual pode ocorrer **comunicação entre um ser humano e um computador**, utilizando-se para isso de uma metáfora de manipulação direta **através de imagens e texto**. Em outras palavras, as GUIs são instrumentos para que o usuário possa operar um computador através de metáforas gráficas.

Existem diversos tipos de GUIs, sendo o mais comum aquele que usa janelas, ícones, menus e um dispositivo apontador (**WIMP: Windows, Icons, Menus e Pointing Device**, tendo sido este um dos modelos pioneiros de interface, desenvolvido pela Xerox, e utilizado até hoje nos computadores.

Um dos grandes **benefícios** que as GUIs trouxeram para o ramo da computação foi uma grande **redução na dificuldade de uso dos computadores**. Isto é conseguido através do uso de metáforas de fácil reconhecimento pelo usuário para a execução de tarefas computacionais (por exemplo: apagar um documento pode ser feito arrastando o ícone do documento em um ícone de lixeira), o que permite que um usuário opere o computador com uma necessidade mínima de aprendizagem.

Além disso, as GUIs **permitem uma grande flexibilidade** na forma com que as tarefas são executadas, possibilitando um projeto de comunicação com o usuário que vise a um aumento de produtividade do mesmo, reduzindo seu cansaço ao máximo.

1. EVOLUÇÃO DAS UIs (OPCIONAL)

Os primeiros computadores desenvolvidos, há cerca de um século, eram equipamentos muito simples, mecânicos, que sequer eram programáveis. A interface de comunicação destes equipamentos, que tinham apenas algumas poucas funções, era feita através alavancas mecânicas e engrenagens.

Os equipamentos foram evoluindo lentamente e já em meados do século XX existiam computadores programáveis, mas sua interface de comunicação ainda era bastante precária, através de conexões elétricas físicas e cartões perfurados.

Durante a segunda metade do século XX os avanços foram incríveis. Surgiram teclados, impressoras e posteriormente monitores, e foram estes equipamentos que trouxeram as interfaces com o usuário para a era visual.

Inicialmente foram desenvolvidos os primeiros sistemas para impressora, que operavam basicamente com estrutura de terminal. O usuário digitava e o conteúdo aparecia na impressora. Com os monitores de vídeo, a interatividade aumentou, mas os primeiros software eram apenas adaptações do que já se utilizava nas impressoras para o vídeo. Estas interfaces estão disponíveis até hoje, em diversos sistemas, e são chamadas **Interfaces de Linha de Comando** (CLIs, Command Line Interfaces).

Durante algum tempo, os sistemas operavam na forma de texto puro, mas à medida em que os usuários se diversificavam (engenheiros, matemáticos, etc... e não apenas especialistas em computadores), começaram a surgir as primeiras **interfaces de menus**, embora ainda em modo texto. Seu uso ficou bastante popular no início da década de 80 e a grande maioria dos software desenvolvidos até 1985 usava este tipo de interface.

Entretanto, já no fim da década de 1970, uma idéia brilhante havia surgido em um dos laboratórios da **Xerox**, o PARC (Palo Alto Research Center), onde havia sido criado o computador **Star**, que era operado através de um ***mouse*** (o primeiro da história) e a comunicação do usuário era feita através de imagens **gráficas**. Este equipamento nunca tornou-se viável devido ao seu alto custo.

Posteriormente foi desenvolvido o padrão **X-Windows** para computadores que executassem o sistema operacional compatível com UNIX. Este padrão tornou-se muito importante e tem muitas características especiais, mas só veio a ter participação no mercado de usuários não especialistas a partir do meio da década de 1990.

Entretanto, já no início da década de 1980, a Apple Computers, naquela época uma pequena empresa, começava a desenvolver um computador pessoal que fosse capaz de executar uma interface daquele tipo, a um preço acessível. O resultado disso foi um computador chamado **Lisa**, que pode ser confundido com uma "versão beta" do **MacIntosh**. O Lisa não fez tanto sucesso, mas a Apple tinha feito o mais importante: tinha criado o *know-how*, ou seja, ela sabia como fazer.

Pouco tempo depois surgia a primeira versão do MacIntosh, a um preço acessível (apesar de alto), que tornou-se o primeiro computador pessoal comercial a oferecer uma interface gráfica com o usuário (GUI). A interface era bastante pobre, mas já era de uso bem mais simples que as tradicionais interfaces de "linha de comando".

A IBM, a grande empresa ocidental dos computadores pessoais da época (no oriente a história é bem mais complexa e pouco conhecida) não quis ficar atrás e contratou com a Microsoft que fizesse uma nova versão do MS/IBM-DOS, com uma interface gráfica e usando os recursos dos mais recentes processadores da Intel (na época, o i80286). Neste mesmo instante a IBM iniciou pesquisas na área de interfaces com usuário, para determinar qual seria a interface perfeita.

Deste convênio surgiram dois software distintos: o Microsoft **Windows** e o MS/IBM **OS/2**. O primeiro deles era apenas uma interface nova para o velho MS/IBM-DOS, já o segundo era um sistema operacional completamente novo, o qual já levava a interface gráfica em consideração em seu projeto. Na aparência, entretanto, os primeiros Windows e os primeiros OS/2 eram absolutamente similares, mas o OS/2 exigia bem mais recursos para executar, uma vez que fazia bem mais coisas que o Windows de então.

Por diversas razões mercadológicas, o Windows acabou se instituindo como padrão e a Microsoft resolveu romper seus contratos com a IBM, no início da década de 1990. A IBM não aceitou essa quebra de contrato pacificamente e, com os primeiros resultados de sua pesquisa sobre interfaces gráficas nas mãos (padrão **CUA-1991**, Common User Access), a IBM resolveu dar um passo ousado e contribuir significativamente para o avanço das interfaces gráficas.

O padrão CUA-1991 especificava diversas características de usabilidade para as interfaces, de forma a minimizar ao máximo a curva de aprendizagem dos usuários. Além disso, o padrão considerava uma implementação totalmente orientada a objetos, permitindo que os software desenvolvidos para a interface se integrassem a ela de forma tão sutil que o usuário nem precisaria ter consciência de quando utilizava uma ferramenta de sistema ou um software de terceiros.

Em 1992 a IBM lançava sua versão 2.0 do OS/2, contendo a WorkPlace Shell, implementando grande parte das características do CUA-1991. A Apple correu e implementou muitas das características também em seu MacOS 7, que na época sequer tinha o nome de MacOS.

Muitos dos recursos do CUA foram mimetizados pela Microsoft no Windows na versão 4.0 do mesmo (conhecido pelo nome de Windows 95). A partir de então os avanços foram muito disputados entre Apple, Microsoft, IBM e um novo jogador: Linus Torvalds que, com o seu Linux, trazia o mundo Unix para a realidade do usuário comum e, com ele, o X-Windows na versão gratuita chamada XFree.

Todas estas interfaces evoluíram e mudaram em muitos aspectos, culminando hoje com as disponíveis no MacOS X, Windows 7 e, no Linux, as diversas como KDE, Gnome, dentre outras. A WorkPlace Shell da IBM, apesar de ter sido a única a realmente implementar completamente orientação a objetos na interface, foi descontinuada em 2005, devido à mudanças na estratégia da empresa desde 1996, que fizeram com que o número de usuários se tornasse bastante reduzido.

2. A ORIENTAÇÃO A EVENTOS DE UMA GUI

Do ponto de vista do usuário, a principal característica das GUIs é, obviamente, que sua operação é gráfica. Entretanto, esta é apenas a diferença mais superficial, no que se chama

de "***look***" (aparência). Existe uma diferença também referente ao "***feel***" (sentimento) que é fundamental e tem uma importância fundamental inclusive na etapa de programação.

Esta diferença se manifesta em "**quem tem o controle**" da operação. Nas antigas **interfaces modo texto**, quem definia "o que acontecerá a seguir" **era sempre o computador**; de uma certa forma, o usuário era refém do software, algumas vezes sem saber como sair de uma dada tela ou mesmo como fechar um programa.

As interfaces modo texto via de regra se baseiam na interação do tipo diálogo: o computador pergunta e o usuário responde. Assim que o usuário responder todas as perguntas, o computador faz algum processamento e posteriormente permite que o usuário escolha alguma outra opção.

As **interfaces gráficas**, por sua vez, operam no modo de "manipulação direta", onde o usuário pode selecionar qualquer elemento visível da tela e comandar uma ação sobre ele, **sem uma ordem pré-estabelecida**. Isto significa que, nas GUIs, em geral é o usuário quem determina "o que vai acontecer em seguida".

Esta característica é fundamental do ponto de vista da programação, dado que ela implica numa mudança completa no conceito base para o projeto da interface. Enquanto nas **interfaces texto o código da interface é meramente seqüencial**, com uma ordem bastante rígida, **nas GUIs este código não tem uma ordem de execução prevista**: dependendo da ação do usuário, um trecho totalmente diferente do código será executado.

Chamamos de "**evento**" cada **ação do usuário** e, se são estes eventos que controlam a execução de um software, este software é dito ***orientado a eventos*** e, portanto, requer uma programação orientada a eventos.

2.1. Programação Orientada a Eventos

A **programação orientada a eventos**, no fundo, não difere muito da programação usual. A diferença reside na **estrutura de controle** do que será executado em cada instante.

Na **programação seqüencial** usual, o **software costuma ter um *loop* principal** que é executado até que o usuário selecione uma opção para sair. Todo o código do software fica dentro deste *loop* e é executado ininterruptamente.

Na **programação orientada a eventos**, o *loop* contém apenas uma chamada ao sistema, verificando se alguma *mensagem de evento* chegou para o programa. Caso nenhuma mensagem tenha chegado, o software fica bloqueado (sem executar) e o sistema (ou outros programas) continua sua operação. Caso alguma **mensagem** tenha chegado, o sistema retorna a mensagem que, através de uma **estrutura do tipo *switch*** escolhe qual trecho de código vai ser executado naquele instante.

Uma má prática de programação é colocar *código útil*, ou seja, o código que realiza a tarefa para a qual o software foi projetado, dentro deste *switch* ou mesmo em outros lugares. Isto não deve ser feito pois, fazendo isso, o programa exigirá a interface gráfica para poder realizar suas tarefas. O ideal é que dentro destas regiões sejam apenas chamadas operações de um outro componente do sistema, este sim executando as atividades do programa. Neste tipo de arquitetura, a MVC, a interface gráfica ocupa a camada *visão* (*view*) e as atividades do sistema ocupam a camada *controle* (*controller*).

3. CONCEITOS DE PROGRAMAÇÃO GUI

Como foi visto, a **programação GUI** é feita através do paradigma denominado "**programação orientada a eventos**". Isso significa que as tarefas do programa não ocorrem em uma ordem pré-definida: as tarefas são executadas quando determinados eventos ocorrem, na ordem em que tais eventos ocorrem.

Entretanto, **é necessária toda uma preparação** para que um software possa agir quando algum evento ocorre, além de ser preciso criar as janelas da aplicação e cuidar de uma série de detalhes que não são necessários em uma programação usual para modo texto (CLI).

Assim como existem diversas formas de resolver um problema usando programação, existem também diversas formas de programar uma GUI, não existindo uma forma explicitamente "correta" ou "incorreta". Em geral convenciona-se chamar de "correta" uma implementação extensível e de "incorretas" todas as outras formas de resolver o mesmo problema.

Nesta aula o foco está em aprender os conceitos básicos da programação orientada a eventos, então não serão focados os aspectos de extensibilidade. Serão, entretanto, apresentadas duas formas de executar uma mesma tarefa: a execução de uma soma com o uso de janelas GUI para receber os dados e apresentar a resposta. A primeira delas é a mais simples programacionalmente, mas a menos extensível e a menos prática. A segunda delas é um pouco mais flexível, mas também é de programação mais complexa.

3.1. Um Aplicativo de Soma com GUI Básica

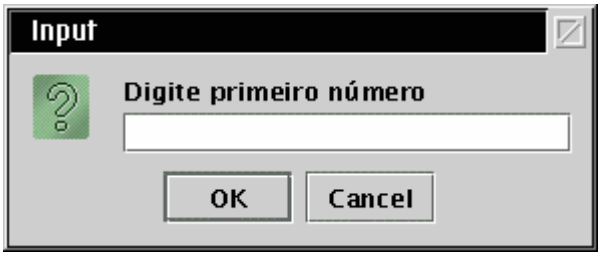
A primeira implementação segue conceitos já vistos em aulas anteriores, apenas para que fique bem clara a proposta da atividade. O projeto terá apenas uma classe, a classe Main.

Esta classe fará todas as funções da adição, criando janelas para pedir informações e janelas para apresentar os resultados. Serão usadas janelas prontas da biblioteca Swing do Java. Acompanhe os detalhes da implementação pelos comentários no código:

No NetBeans, crie um projeto chamado "calculadora" e defina a classe principal dele como "calculadora.Main"

```
/* Primeiramente definimos o pacote da aplicação */
package calculadora

/* Agora especificamos que vamos usar a biblioteca
   JOptionPane, do pacote javax.swing */
import javax.swing.JOptionPane;

/* Em seguida criamos a classe de nossa aplicação */
public class Main {
    /* E é definido o método main (principal) */
    public static void main ( String args[] ) {
        /* Declaramos algumas variáveis temporárias: alphaNum1 e
           alphaNum2 serão usadas para guardar os valores digitados
           pelo usuário no formato String (as janelas sempre retornam
           valores digitados pelo usuário como strings). num1 e num2
           serão usadas para colocar o valor digitado pelo usuário já
           convertidos para números e soma será usado para guardar o
           resultado da soma. */
        String alphaNum1, alphaNum2;
        int num1, num2, soma;

        /* Usamos o método estático showInputDialog (mostraJanelaDeEntrada)
           da classe JOptionPane (JPainelDeOpções) para colher entradas do
           usuário. A primeira chamada serve para pegar o primeiro valor da
           soma e a segunda para pegar o segundo valor da soma. Note que o
           valor é guardado na variável do tipo String */
        alphaNum1 = JOptionPane.showInputDialog ("Digite primeiro número");
        alphaNum2 = JOptionPane.showInputDialog ("Digite segundo número");

        /* Agora o método estático parseInt (traduzInteiro) da classe
           Integer (Inteiro) é chamado para traduzir o texto digitado pelo
           usuário em números, armazenados nas variáveis num1 e num2.
           Note que seria necessário realizar uma verificação, caso o
           valor digitado não fosse um número.  */
        num1 = Integer.parseInt( alphaNum1 );
        num2 = Integer.parseInt( alphaNum2 );

        /* A soma é realizada, e o resultado é armazenado em soma. */
        soma = num1 + num2;

        /* Usamos o método estático showMessageDialog (mostraJanelaDeMensagem)
           da classe JOptionPane (JPainelDeOpções) para mostrar o resultado para
           o usuário. */
        JOptionPane.showMessageDialog(null,"Soma: " + soma,
                "Soma de dois inteiros", JOptionPane.PLAIN_MESSAGE );
        }
     }
```

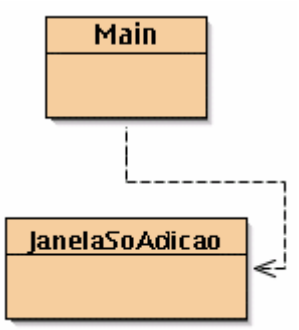
É importante notar que, neste exemplo, como usamos diretamente as janelas fornecidas pelo pacote Swing do Java, não precisamos nem mesmo entrar em contato com as mensagens (embora elas existam, dentro das janelas que abrimos)! Entretanto, este software é extremamente limitado e até mesmo pouco prático, pois usa várias janelas para fazer uma única coisa. Além disso, ele praticamente nem é orientado a objetos. Uma das janelas desta implementação está representada abaixo:



3.2. Um Aplicativo de Soma com GUI mais Avançada

A segunda implementação que será examinada já é um pouco mais complexa. Serão utilizadas três classes (duas delas explícitas e a outra implícita) para realizar a tarefa. Uma delas será a classe principal do aplicativo e a outra será a classe da janela. A execução da tarefa será toda executada dentro da classe de janela, que é única em todo o processo.

Conceitualmente, este é um "mal programa", entretanto é este tipo de programa que se costuma produzir usando os aplicativos de auxílio ao desenvolvimento de interfaces gráficas. Maneiras melhores de desenvolver um software de janela serão examinadas na disciplina Programação de Componentes. O diagrama de classe do novo programa proposto está representado abaixo:



A classe Main é a classe principal do aplicativo e a classe JanelaSoAdicao é a classe que apresenta a janela onde são realizadas as operações. Notem a seta indicando que "Main usa JanelaSoAdicao".

Acompanhe os detalhes da implementação pelos comentários do código, inciando pela classe Main.

No NetBeans, crie um projeto chamado "calculadora2" e defina como classe principal a classe "calculadora2.Main".

```
/* Primeiramente definimos o pacote da aplicação */
package calculadora2;

/* Agora criamos a classe da Aplicação */
public class Main {
    /* E seu método principal */
    public static void main ( String args[] ) {
        /* Declaramos uma referência para a janela que será criada */
        JanelaSoAdicao umaJanela;

        /* E é feita uma solicitacao que o Java nos crie a janela */
        umaJanela = new JanelaSoAdicao();
        }
    }
```

A classe principal é muito simples: ela apenas cria um objeto do tipo JanelaSoAdicao e o deixa realizar seu "trabalho". O código inicial da classe JanelaSoAdicao segue abaixo.

No NetBeans, clique com o botão direito no pacote "calculadora2" e selecione "Novo > Classe Java", dando o nome de "JanelaSoAdicao" à nova classe.

```
/* Primeiramente definimos o pacote da aplicação */
package calculadora2;

/* Agora especificamos que vamos usar várias bibliotecas
   dos pacotes java.awt, java.awt.event e javax.swing */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/* Em seguida é declarada a classe da janela da aplicação, que
   é uma especialização da classe JFrame */
public class JanelaSoAdicao extends JFrame {
    /* Aqui são declaradas as variáveis onde serão realizadas
       as operações, num1 e num2 para armazenar os valores numéricos
       digitados pelo usuário e soma para armazenar o valor a ser
       apresentado como saída */
    private int num1, num2, soma;

    /* O próximo passo é declarar o construtor da classe JaneaSoAdicao
       que nada faz além de construir a janela */
    public JanelaSoAdicao() {
        /* No caso de janelas derivadas de JFrame e JDialog, a primeira
           instrução deve ser - sempre - uma chamada ao construtor da
           superclasse. Esta chamada está especificando o título da
           janela a ser criada. */
        super("Programa de Adicao");

        /* Configurações Finais da Janela */
        /* Indica que, ao apertar o botão fechar, o aplicativo deve sair */
        setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );
        /* Ajusta o tamanho da janela */
        setSize ( 350, 80 );
        /* Indica que a janela não pode ser redimensionada */
        setResizable ( false );
        /* Indica para a janela aparecer */
        setVisible ( true );
        }
    }
```

Isso apresenta a janela, mas ela ainda está vazia, não tem elemento nenhum. Para que possamos adicionar elementos na janela, precisamos entender que a janela é composta por duas partes: **o frame** (ou moldura) e **a área cliente**. Normalmente não alteramos o frame, que é a moldura da janela, mas é comum desejarmos modificar a área cliente, ou seja, o interior da janela.

No Swing essa região é um objeto do tipo Conteiner, porque ele armazena todos os elementos da janela, ou seja, ele contém os elementos da janela. Assim, precisamos pegar uma referência, do tipo conteiner, para a região cliente, que vamos chamar de **painel**.

```
package calculadora2;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class JanelaSoAdicao extends JFrame {
    private int num1, num2, soma;

    public JanelaSoAdicao() {
        super("Programa de Adicao");

        /* Agora é declarada uma referência para o painel da janela
           criada, de forma que possamos adicionar coisas neste painel. */
        Container painel;
        /* E é pedido para a janela o valor da referência do painel */
        painel = getContentPane();

        setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );
        setSize ( 350, 80 );
        setResizable ( false );
        setVisible ( true );
        }
    }
```

Agora precisamos criar referências para os objetos que serão colocados na janela: dois campos de texto para entrada (tipo JTextField), um botão (tipo JButton) e um campo texto de saída (tipo JTextField)... e depois criar estes objetos:

```
package calculadora2;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JanelaSoAdicao extends JFrame {
    /* Aqui são declaradas as referências para os objetos que
       estarão presentes na janela: dois campos de texto para a
       entrada (entrada1 e entrada2), um botão para executar a
       soma (botaoSoma) e um outro campo de texto para a saída
       do resultado (saida). */
    private JTextField entrada1, entrada2;
    private JButton botaoSoma;
    private JTextField saida;

    private int num1, num2, soma;

    public JanelaSoAdicao() {
        super("Programa de Adicao");

        Container painel;
        painel = getContentPane();

        /* Cria-se o primeiro campo de entrada (com 5 posições)... */
        entrada1 = new JTextField(5);
        /* Cria-se o segundo campo de entrada (com 5 posições)... */
        entrada2 = new JTextField(5);
        /* Cria-se o botão com o texto "Soma!" */
        botaoSoma = new JButton("Soma!");
        /* Cria-se o campo de saída (com 10 posições) */
        saida = new JTextField(10);

        setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );
        setSize ( 350, 80 );
        setResizable ( false );
        setVisible ( true );
        }
    }
```

Agora que temos os elementos criados, precisamos adicioná-los ao painel. Entretanto, para que o painel aceite estes elementos, devemos indicar **como eles devem ser organizados**. Existem vários tipos de organização (procure na documentação da classe Conteiner, o método setLayout, para verificar as possibilidades). Neste caso usaremos um layout fluido (tipo **FlowLayout**), isto é, que não é fixo, posicionando um elemento ao lado do outro.

Quando o tipo de layout estiver definido para o painel, basta adicionar elementos nele:

```
package calculadora2;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JanelaSoAdicao extends JFrame {
    private JTextField entrada1, entrada2;
    private JButton botaoSoma;
    private JTextField saida;

    private int num1, num2, soma;

    public JanelaSoAdicao() {
        super("Programa de Adicao");

        Container painel;
        painel = getContentPane();

        /* Indica-se o layout como FlowLayout, ou seja, uma disposição
           em que os elementos serão colocados lado a lado. */
        painel.setLayout ( new FlowLayout() );

        entrada1 = new JTextField(5);
        /* Coloca-se o campo de entrada no painel. */
        painel.add(entrada1);

        entrada2 = new JTextField(5);
        /* Coloca-se o campo de entrada no painel. */
        painel.add(entrada2);

        botaoSoma = new JButton("Soma!");
        /* Coloca-se o botão no painel. */
        painel.add(botaoSoma);

        saida = new JTextField(10);
        /* Coloca-se o campo de saída no painel. */
        painel.add(saida);

        setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );
        setSize ( 350, 80 );
        setResizable ( false );
        setVisible ( true );
        }
    }
```

A classe já está ok... só falta agora fazer duas coisas: primeiro, marcar o campo de saída de texto como "Não Editável"; segundo, criar uma classe/método (tipo **ActionListener**) para executar a função do botão.

Observe atentamente o código a seguir e seus comentários:

```
package calculadora2;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JanelaSoAdicao extends JFrame {
    private JTextField entrada1, entrada2;
    private JButton botaoSoma;
    private JTextField saida;

    private int num1, num2, soma;

    public JanelaSoAdicao() {
        super("Programa de Adicao");
```

```
        Container painel;
        painel = getContentPane();
        painel.setLayout ( new FlowLayout() );

        entrada1 = new JTextField(5);
        painel.add(entrada1);

        entrada2 = new JTextField(5);
        painel.add(entrada2);

        botaoSoma = new JButton("Soma!");
        painel.add(botaoSoma);

        saida = new JTextField(10);
        /* Marca-se este campo como "não editável" */
        saida.setEditable(false);
        painel.add(saida);

        /* Agora será associada uma classe implícita, implementando
           a interface ActionListener, como a executora da ação do
           botão criado anteriormente. */
        botaoSoma.addActionListener( new ActionListener() {
            /* Quando o botão for clicado, o método actionPerformed
               desta classe será chamada. Deve-se, então, redefinir
               esse método para que ele realize as atividades desejadas */
           public void actionPerformed( ActionEvent event ) {
                /* entrada1.getText() retorna o texto que o usuário
                   digitou no campo entrada1. Este texto já é repassado
                   para o método Integer.parseInt, que o traduz em um
                   número e o armazena na variável num1. O procedimento
                   para obter o segundo valor é o mesmo, colhendo a
                   informação em entrada2 e armazenando o resultado em num2. */
                num1 = Integer.parseInt( entrada1.getText() );
                num2 = Integer.parseInt( entrada2.getText() );

                /* A soma é realizada normalmente */
                soma = num1 + num2;

                /* E o texto do objeto saida é alterado para a resposta calculada,
                   usando o método saida.setText. */
                saida.setText("Soma: "+soma);
                }
           });

        setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );
        setSize ( 350, 80 );
        setResizable ( false );
        setVisible ( true );
        }
    }
```

Assim, temos um aplicativo que realiza todas as operações em uma única janela, como a apresentada abaixo:

O código completo, com comentários, está a seguir:

```
/* Primeiramente definimos o pacote da aplicação */
package calculadora2;

/* Agora especificamos que vamos usar várias bibliotecas
```

```
   dos pacotes java.awt, java.awt.event e javax.swing */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/* Em seguida é declarada a classe da janela da aplicação, que
   é uma especialização da classe JFrame */
public class JanelaSoAdicao extends JFrame {
    /* Aqui são declaradas as referências para os objetos que
       estarão presentes na janela: dois campos de texto para a
       entrada (entrada1 e entrada2), um botão para executar a
       soma (botaoSoma) e um outro campo de texto para a saída
       do resultado (saida). */
    private JTextField entrada1, entrada2;
    private JButton botaoSoma;
    private JTextField saida;

    /* Também são declaradas as variáveis onde serão realizadas
       as operações, num1 e num2 para armazenar os valores numéricos
       digitados pelo usuário e soma para armazenar o valor a ser
       apresentado como saída */
    private int num1, num2, soma;

    /* O próximo passo é declarar o construtor da classe JaneaSoAdicao
       que nada faz além de construir a */
    public JanelaSoAdicao() {
        /* No caso de janelas derivadas de JFrame e JDialog, a primeira
           instrução deve ser - sempre - uma chamada ao construtor da
           superclasse. Esta chamada está especificando o título da
           janela a ser criada. */
        super("Programa de Adicao");

        /* Agora é declarada uma referência para o painel da janela
           criada, de forma que possamos adicionar coisas neste painel. */
        Container painel;
        /* E é pedido para a janela o valor da referência do painel */
        painel = getContentPane();
        /* Indica-se o layout como FlowLayout, ou seja, uma disposição
           em que os elementos serão colocados lado a lado. */
        painel.setLayout ( new FlowLayout() );

        /* Cria-se o primeiro campo de entrada (com 5 posições)... */
        entrada1 = new JTextField(5);
        /* E coloca-se o mesmo no painel. */
        painel.add(entrada1);

        /* Cria-se o segundo campo de entrada (com 5 posições)... */
        entrada2 = new JTextField(5);
        /* E coloca-se o mesmo no painel. */
        painel.add(entrada2);

        /* Cria-se o botão com o texto "Soma!" */
        botaoSoma = new JButton("Soma!");
        /* E coloca-se o mesmo no painel. */
        painel.add(botaoSoma);

        /* Cria-se o campo de saída (com 10 posições) */
        saida = new JTextField(10);
        /* Marca-se este campo como "não editável" */
        saida.setEditable(false);
        /* E coloca-se o mesmo no painel. */
        painel.add(saida);

        /* Agora será associada uma classe implícita, implementando
           a interface ActionListener, como a executora da ação do
           botão criado anteriormente. */
        botaoSoma.addActionListener( new ActionListener() {
            /* Quando o botão for clicado, o método actionPerformed
               desta classe será chamada. Deve-se, então, redefinir
               esse método para que ele realize as atividades desejadas */
           public void actionPerformed( ActionEvent event ) {
                /* entrada1.getText() retorna o texto que o usuário
                   digitou no campo entrada1. Este texto já é repassado
                   para o método Integer.parseInt, que o traduz em um
                   número e o armazena na variável num1. O procedimento
                   para obter o segundo valor é o mesmo, colhendo a
                   informação em entrada2 e armazenando o resultado em num2. */
               num1 = Integer.parseInt( entrada1.getText() );
               num2 = Integer.parseInt( entrada2.getText() );
```

```
                /* A soma é realizada normalmente */
                soma = num1 + num2;

                /* E o texto do objeto saida é alterado para a resposta calculada,
                   usando o método saida.setText. */
                saida.setText("Soma: "+soma);
                }
            });

        /* Terminada a definição da classe que interpreta o evento do botão,
           termina-se de definir as propriedades da janela */

        /* Indica que, ao apertar o botão fechar, o aplicativo deve sair */
        setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE );
        /* Ajusta o tamanho da janela */
        setSize ( 350, 80 );
        /* Indica que a janela não pode ser redimensionada */
        setResizable ( false );
        /* Indica para a janela aparecer */
        setVisible ( true );
        }
    }
```

BIBLIOGRAFIA

DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.

Bibliografia Complementar:

DIX, A; FINLAY, J; ABOWD, G; BEALE, R. **Human-Computer Interaction.** Edinburg, England: Prentice-Hall, 1997.

SHNEIDERMAN, B. **Designing the User Interface.** 3rd. Ed. Addison-Wesley. Reading, Ma. 1998.