10 Introdução à Interface Gráfica com o Usuário

Onde são apresentados recursos para melhoria da aparência dos programas desenvolvidos em Java.

Pré-requisito(s):	Objetivos (ao final você deverá ser capaz de):
• Saber desenvolver programas modularizados em Java que façam uso de estruturas de dados compostas e que gravem e leiam dados em arquivos	• Utilizar janelas e botões em seus programas Java para solicitar entrada de dados do usuário • Exibir resultado de processamento em janelas gráficas • Desenhar figuras simples em telas • Gerenciar visualmente arquivos e pastas em seus programas • Conhecer o que são applets Java e como utilizá-los para escrever programas que executem em um *browser*

Os programas Java que desenvolvemos até o capítulo 9 realizam a leitura de dados a partir de uma janela de comando e exibem o resultado do processamento também na janela de comando. Sabemos, entretanto, que a maioria dos programas de que fazemos uso no dia a dia possui janelas e botões que nos permitem interagir através do uso do mouse. Editores de texto, planilhas eletrônicas, software para preparação de seminários, leitura e escrita de e-mails, *browsers* de navegação na Internet, jogos, e diversos outros aplicativos são exemplos de programas que possuem uma interface mais amigável com o usuário; uma interface mais rica, mais bonita e também mais fácil de utilizar. Este capítulo irá mostrar alguns mecanismos simples de Java que permitirão a criação de uma melhor interface para nossos programas.

10.1 Interface Gráfica com o Usuário

Quando um programa apresenta mecanismos visuais que melhoram a aparência e auxiliam sua manipulação por parte dos usuários, dizemos que ele possui uma **interface gráfica com o usuário** (*graphical user interface - GUI*).

Uma GUI é construída a partir de diversos tipos de componentes, conhecidos por ***widgets***. Um *widget* é um objeto visual que permite ao usuário interagir via um determinado dispositivo de entrada, como teclado, mouse ou microfone. Botões, janelas, menus, campos de texto, barras de rolagem, são exemplos de *widgets* que podem compor uma GUI.

10.1.1 Entrada e Saída com Caixas de Diálogo

Para exemplificação da criação de uma GUI, faremos uso de um *widget* em particular: as **caixas de diálogo**.

Uma caixa de diálogo é uma janela que os programas utilizam para permitir a entrada de dados pelo usuário e exibir mensagens de texto ou os resultados do processamento.

Java possui uma classe pré-definida chamada `JOptionPane` que possui métodos para exibir diferentes caixas de diálogo como interface de uma aplicação.

A figura 10.1 mostra um programa simples para cálculo da área de um retângulo, cujos valores dos lados são solicitados via caixas de diálogo e o cálculo da área é informado ao usuário também via uma caixa de diálogo (figura 10.2).

```
1.  import javax.swing.JOptionPane;
2.
3.  public class RetanguloGUI {
4.     public static void main(String[] args){
5.        String a1, b1;
6.        int a, b, area;
7.        a1 = JOptionPane.showInputDialog ("Digite o primeiro
8.                                          valor: ");
9.        b1 = JOptionPane.showInputDialog ("Digite o segundo
10.                                         valor: ");
11.       a = Integer.parseInt(a1);
12.       b = Integer.parseInt(b1);
13.       area = a * b;
14.       JOptionPane.showMessageDialog(null, "A área é igual a "
15.       + area, "Resultado", JOptionPane.INFORMATION_MESSAGE);
16.    }
17. }
```

Figura 10.1 – Uso de caixas de diálogo em Java.

Os comandos `JoptionPane.showInputDialog` das linhas 7 e 9 apresentam caixas de diálogo pré-definidas do tipo input. Caixas desse tipo possui uma interface padrão, composta por uma mensagem informativa da ação que o usuário deve tomar, que é definida pelo programador como parâmetro da função em questão (`"Digite o primeiro/segundo valor"`) e dois botões de ação chamados `OK` (para confirmação) e `Cancel` (para cancelamento). O resultado da aplicação dessa função é sempre um valor do tipo `String` (variáveis `a1` e `b1`) e, por isso, como o exemplo trata de valores inteiros, eles devem ser transformados adequadamente (linhas 11 e 12). A área é então calculada (linha 13) e o resultado é impresso em uma outra caixa de diálogo pré-definida para exibir mensagens. A função `JoptionPane.showMessageDialog` permite customizar a tanto a mensagem que será gerada (*A área é igual a 15*) quanto o título da janela em questão (*Resultado*).

Figura 10.2 – Duas caixas de diálogo para entrada de dados e uma para saída de resultados.

No exemplo anterior, uma mensagem de texto simples era impressa no conteúdo da caixa de diálogo com o resultado da aplicação. Uma caixa de diálogo, entretanto, permite que ao invés de exibir uma String, um outro *widget* possa ser adicionado em seu conteúdo, possibilitando assim uma interface um pouco mais elaborada.

O programa ilustrado na figura 10.3 calcula a raiz quadrada dos números inteiros de 1 a 15. Esses valores são concatenados sucessivamente para formar um texto-resposta contendo todas as raízes calculadas, que é então exibido para o usuário. O exemplo utiliza, entretanto, um *widget* chamado `JTextArea`, que

representa uma área de texto onde informações podem ser exibidas e/ou dados podem ser lidos.

```
1.  import javax.swing.*;
2.  public class RaizQuadradaGUI {
3.     public static void main(String[] args){
4.        double r;
5.        String texto = "";
6.        JTextArea a1 = new JTextArea(20, 20);
7.        for (int i = 1; i <= 15; i++){
8.          r = Math.sqrt((double)i);
9.          texto = texto + ("Raiz de " + i + " = " + r + "\n");
10.       }
11.       a1.setText(texto);
12.       JoptionPane.showMessageDialog(null,a1,
13.        "Cálculo de Raízes", JOptionPane.INFORMATION_MESSAGE);
14.    }
15. }
```

Figura 10.3 – Uso de caixas de diálogo com outros widgets.

Na linha 6 é definido e criado um componente do tipo `JTextArea` para comportar o relatório textual contendo o resultado do cálculo da raiz quadrada pra os inteiros de 1 a 15, conforme laço da linha 7. O relatório textual é "montado" através de concatenações sucessivas dentro do laço (linha 9) e então adicionado ao componente `JTextArea` (linha 11). Finalmente, na linha 12, uma caixa de diálogo é exibida incluindo o componente em seu layout. A figura 10.4 ilustra o efeito da execução do programa em questão.

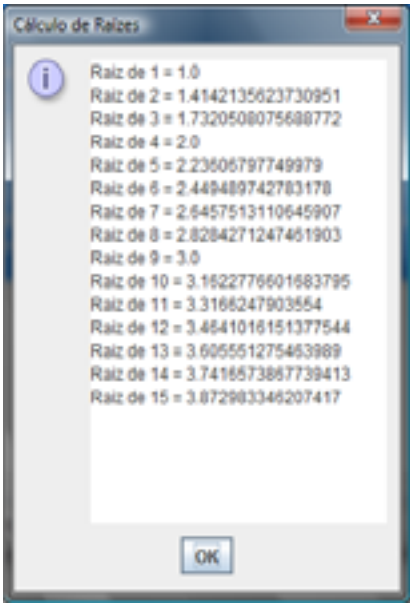


Figura 10.4 – Caixa de diálogo com widget `JTextArea` adicionado.

O programa ilustrado na figura 10.5 traz outro exemplo do uso JTextArea com JoptionPane. O programa determina a quantidade de moradores de um condomínio por faixa etária e por sexo. Faz uso de uma matriz de duas colunas, a primeira para o sexo masculino e a segunda para o feminino. Cada linha da matriz corresponde a uma faixa etária. As faixas são definidas como se segue:

linha 1: de 0 a 18 anos;

linha 2: de 19 a 29;

linha 3: de 30 a 59; linha 4: acima de 59 anos.

A figura 10.6 ilustra a caixa de diálogo que exibe a matriz com as estatísticas para um exemplo de dados fornecido.

```
1.  import javax.swing.*;
2.  public class MoradoresGUI {
3.     public static void main(String[] args){
4.        String[] faixa = {"até 18 anos:", "de 19 a 29:",
5.                          "de 30 a 59:", "acima de 60:"};
6.        String resultado = "";
7.        int[][] a = new int[4][2];
8.        int id, sx, lin, col;
9.        String a1 = JOptionPane.showInputDialog ("Digite o
10.                    número de moradores: ");
11.       int n = Integer.parseInt(a1);
12.
13.       // zerando a matriz
14.      for (int i = 0; i < 4; i++)
15.          for (int j = 0; j < 2; j++)
16.             a[i][j] = 0;
17.      for (int c = 0; c < n; c++){
18.          String a2 = JOptionPane.showInputDialog ("Digite o
19.                       sexo do morador (1-mas, 2-fem): ");
20.          sx = Integer.parseInt(a2);
21.          String a3 = JOptionPane.showInputDialog ("Digite a
22.                       idade do morador: ");
23.          id = Integer.parseInt(a3);
24.
25.          if (id < 19)
26.                   lin = 0;
27.          else
28.                   if (id < 30)
29.                            lin = 1;
30.                   else
31.                            if (id < 60)
32.                                     lin = 2;
33.                            else
34.                                     lin = 3;
35.          if (sx == 1)
36.                   col = 0;
37.          else
38.                   col = 1;
39.          a[lin][col] = a[lin][col] + 1;
40.      }
41.      resultado = resultado + "\nMatriz calculada:\tMASC\tFEM\
```

```
42. n";
43.
44.       for (int i = 0; i < 4; i++){
45.            resultado = resultado + "\t" + faixa[i];
46.            for (int j = 0; j < 2; j++)
47.                      resultado = resultado + "\t" + a[i][j];
48.            resultado = resultado + "\n";
49.       }
50.       JTextArea ta = new JTextArea(7, 22);
51.       ta.setText(resultado);
52.       JOptionPane.showMessageDialog(null,ta,"Estatística de
53.                                          moradores",
54.       JOptionPane.INFORMATION_MESSAGE);
55.     }
56. }
```

Figura 10.5 – Estatística de moradores com caixas de diálogo.

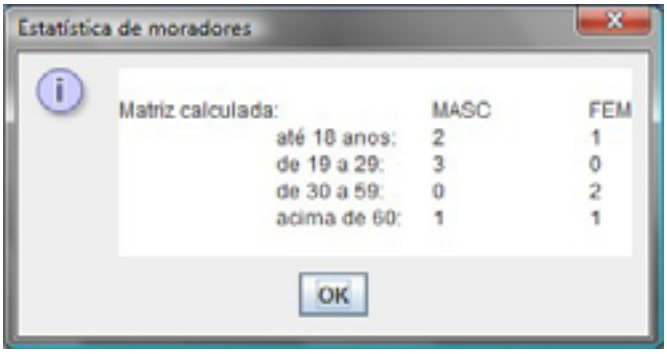


Figura 10.6 – Estatística de moradores exibidas com `JTextArea`.

10.1.2 Desenho

Um outro recurso poderoso para incrementar a interface de um programa Java é são as ferramentas de desenho gráfico em janelas, presentes na classe `java.awt.Graphics`. Essa classe possui métodos que permitem a criação de desenhos com linhas, formas e cores.

A figura 10.7 traz um programa Java que solicita ao usuário a largura e altura de retângulo através de caixas de diálogo e exibe como resultado uma janela gráfica, onde a área do retângulo é escrita, juntamente com o desenho do retângulo. As proporções visuais para os valores da largura e altura são respeitadas. A figura 10.8 ilustra um exemplo de execução para os valores de largura = 200 e altura = 300.

```
1.  import java.awt.Graphics;
2.  import javax.swing.*;
```

```
3.  public class RetanguloGraficoGUI extends JPanel{
4.     static int largura, altura;
5.     public static void main(String[] args){
6.     // Cria o panel para conter o desenho
7.           largura =
8.              Integer.parseInt(JOptionPane.showInputDialog
9.                       ("Digite a largura do retângulo: "));
10.          altura =
11.             Integer.parseInt(JOptionPane.showInputDialog
12.                      ("Digite a altura do retângulo: "));
13.          RetanguloGraficoGUI p = new RetanguloGraficoGUI();
14.          JFrame jan = new JFrame();
15.          jan.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16.                   jan.add(p);
17.                   jan.setSize(250, 250);
18.                   jan.setVisible(true);
19.          }
20.          // O paintComponent() é chamado automaticamente
21.             sempre que o painel precise ser exibido
22.          public void paintComponent(Graphics g){
23.                   super.paintComponent(g);
24.                   int area = largura*altura;
25.                   if (largura >= altura) {
26.                            altura = altura*200/largura;
27.                            largura = 100;
28.                   } else {
29.                            largura = largura*200/altura;
30.                            altura = 100;
31.                   }
32.
33.                   g.drawRect(10, 10, largura, altura);
34.                   g.drawString("Área =  " + area, 20, 50);
35.          }
36. }
```

Figura 10.7 – Cálculo da área de um retângulo com desenho.

Note que o tipo de janela utilizado no exemplo não é o mesmo utilizado nos exemplos anteriores. O programa utiliza um novo widget, chamado `JFrame` (linha 14), que permite uma maior liberdade de confecção de interfaces.

Figura 10.8 – Desenho da área de um retângulo em uma janela.

10.2 Gerenciador visual de Arquivos e Diretórios

No capítulo 9, vimos como criar, abrir, editar e fechar arquivos em Java. Entretanto, a interface que utilizamos para tal foi bem simples, sem recurso visual algum.

A linguagem Java fornece um widget apropriado para lidar com o gerenciamento de arquivos: a classe JFileChooser. Essa classe exibe uma caixa de diálogo que permite aos usuários do programa selecionar arquivos com que queiram trabalhar de forma mais fácil.

O programa ilustrado na figura 10.9 exibe informações sobre um arquivo ou diretório que o usuário selecione através da interface que lhe é apresentada.

```
1.  import java.awt.BorderLayout;
2.  import java.io.File;
3.  import javax.swing.JFileChooser;
4.  import javax.swing.JFrame;
5.  import javax.swing.JOptionPane;
6.  import javax.swing.JScrollPane;
7.  import javax.swing.JTextArea;
8.
9.  public class GerenciadorArquivos extends JFrame
10. {
11. private JTextArea outputArea; // utilizado para saída
12. private JScrollPane scrollPane; // utilizado para fornecer rolagem para saída
13.
14. // configura a GUI
15. public GerenciadorArquivos()
16. {
17.    super( "Testing class File" );
18.
19.    outputArea = new JTextArea();
20.
21.    // adiciona outputArea a scrollPane
22.    scrollPane = new JScrollPane( outputArea );
23.
24.    add( scrollPane, BorderLayout.CENTER ); // adiciona scrollPane a GUI
25.
26.    setSize( 400, 400 ); // configura o tamanho da GUI
27.    setVisible( true ); // exibe a GUI
28.
29.    analyzePath(); // cria e analisa o objeto File
30. } // fim do construtor de FileDemonstration
31.
32. // permite que o usuário especifique o nome de arquivo
33. private File getFile()
34. {
35.    // exibe o diálogo de arquivo para o usuário escolher o arquivo a abrir
36.    JFileChooser fileChooser = new JFileChooser();
37.    fileChooser.setFileSelectionMode(
38.       JFileChooser.FILES_AND_DIRECTORIES );
39.
40.    int result = fileChooser.showOpenDialog( this );
41.
42.    // se o usuário clicou no botão Cancel no diálogo, retorna
43.    if (result == JFileChooser.CANCEL_OPTION)
44.       System.exit( 1 );
45.
46.    File fileName = fileChooser.getSelectedFile(); // obtém o arquivo selecionado
47.
48.    // exibe erro se inválido
49.    if ( ( fileName == null ) || ( fileName.getName().equals( "" ) ) )
50.    {
51.       JOptionPane.showMessageDialog( this, "Invalid File Name",
52.          "Invalid File Name", JOptionPane.ERROR_MESSAGE );
53.       System.exit( 1 );
54.    } // fim do if
55.
56.    return fileName;
```

```
57.   } // fim do método getFile
58.
59.   // exibe informações sobre o arquivo que o usuário especifica
60.   public void analyzePath()
61.   {
62.      // cria o objeto File com base na entrada de usuário
63.      File name = getFile();
64.
65.      if ( name.exists() ) // se o nome existir, dá saída das informações sobre ele
66.      {
67.         // exibe informações sobre o arquivo (ou diretório)
68.         outputArea.setText( String.format(
69.            "%s%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s",
70.            name.getName(), " exists",
71.            ( name.isFile() ? "is a file" : "is not a file" ),
72.            ( name.isDirectory() ? "is a directory" :
73.               "is not a directory" ),
74.            ( name.isAbsolute() ? "is absolute path" :
75.               "is not absolute path" ), "Last modified: ",
76.            name.lastModified(), "Length: ", name.length(),
77.            "Path: ", name.getPath(), "Absolute path: ",
78.            name.getAbsolutePath(), "Parent: ", name.getParent() ) );
79.
80.         if ( name.isDirectory() ) // listagem de diretório de saída
81.         {
82.            String directory[] = name.list();
83.            outputArea.append( "\n\nDirectory contents:\n" );
84.
85.            for ( String directoryName : directory )
86.               outputArea.append( directoryName + "\n" );
87.         } // fim do else
88.      } // fim do if externo
89.      else // não for arquivo ou diretório, gera saída da mensagem de erro
90.      {
91.         JOptionPane.showMessageDialog( this, name +
92.            " does not exist.", "ERROR", JOptionPane.ERROR_MESSAGE );
93.      } // fim do else
94.   } // fim do método analyzePath
95.
96.   public static void main( String args[] )
97.   {
98.      GerenciadorArquivos application = new GerenciadorArquivos();
99.      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
100.  } // fim de main
101.
102.  } // fim da classe GerenciadorArquivos
```

Figura 10.9 – Uso de `JFileChooser` para gerenciamento visual de arquivos e diretórios.

Ao ser executado, o programa exibe uma janela para navegação por arquivos e diretórios (figura 10.10).

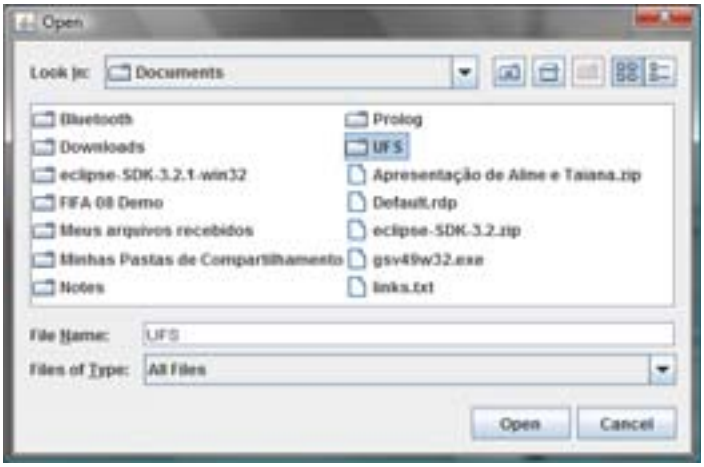


Figura 10.10 – Janela para navegação por arquivos e diretórios.

Uma vez que o usuário selecionou o arquivo ou diretório de que gostaria de obter informações, tais como, tamanho, data da última modificação, conteúdo (do diretório), caminho completo, etc., uma nova janela é apresentada contendo essas informações desejadas (figura 10.11).

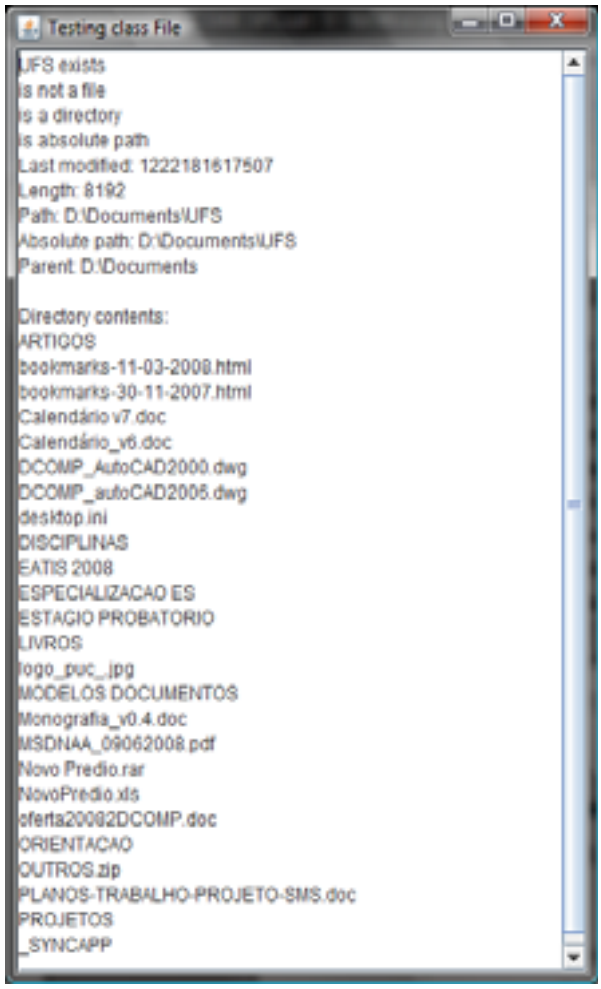


Figura 10.11 – Janela com informações sobre diretório selecionado.

10.3 Applets

Applet é o nome dado a um programa Java que pode ser adicionado a páginas Web (formato HTML). Quando um browser carrega uma página da Web que contém um applet, o applet é imediatamente executado no browser.

Um applet Java é um GUI em que podemos adicionar vários *widgets*, como os vistos anteriormente, entre outros.

A figura 10.12 mostra o código de um applet Java simples, que exibe uma mensagem de boas vindas ao curso de *Introdução à Ciência da Computação na*

UAB/UFS.

```
1.  import java.awt.Graphics;
2.  import javax.swing.JApplet;
3.
4.  public class BoasVindas extends JApplet
5.  {
6.     // desenha texto sobre o fundo do applet
7.        public void paint( Graphics g )
8.     {
9.              super.paint( g );
10.
11.       // desenha uma String nas coordenadas x 25 e y 25
12.       g.drawString( "Bem vindo ao curso de Introdução à
13.                   Ciência da Computação da UAB/UFS!", 25, 25 );
14.    }
15. }
16.
```

Figura 10.12 – Applet Java que exibe mensagem de boas vindas.

A linha 1 importa a classe `Graphics` que permite ao applet desenhar imagens gráficas como linhas, círculos, retângulos e strings de texto. A classe `JApplet` é usada para criar de fato o applet. O que vai ser desenhado em um applet deve obrigatoriamente estar codificado dentro de um método chamado `paint` (linha 7). Nesse exemplo, apenas uma string de boas vindas é desenhada (linha 13).

A próxima seção mostra como faremos para incorporar esse applet numa página HTML e podermos executá-lo a partir de um browser.

O applet da figura 10.13 realiza a soma de dois valores fornecidos pelo usuário através de uma caixa de diálogo `JOptionPane` e exibe o resultado desenhando uma string dentro de um retângulo no applet.

```
1.   import java.awt.Graphics;
2.   import javax.swing.JApplet;
3.   import javax.swing.JOptionPane;
4.
5.   public class SomaValores extends JApplet
6.   {
7.   private double sum; // soma dos valores inseridos pelo usuário
8.
9.   // inicializa um applet obtendo os valores inseridos pelo usuário
10.  public void init()
11.  {
12.     String firstNumber;  // primeira string inserida pelo usuário
13.     String secondNumber; // segunda string inserida pelo usuário
14.
15.     double number1; // primeiro número a adicionar
16.     double number2; // segundo número a adicionar
17.
18.     // obtém do usuário o primeiro número
19.     firstNumber = JoptionPane.showInputDialog(
20.           "Entre com o primeiro valor real:" );
21.
22.     // obtém do usuário o segundo número
```

```
23.     secondNumber = JOptionPane.showInputDialog(
24.        "Entre com o segundo valor real:" );
25.
26.     // converte os números de tipo String para tipo double
27.     number1 = Double.parseDouble( firstNumber );
28.     number2 = Double.parseDouble( secondNumber );
29.
30.     sum = number1 + number2; // soma os números
31.  } // fim do método init
32.
33.  // desenha os resultados em um retângulo sobre o fundo do applet
34.  public void paint( Graphics g )
35.  {
36.     super.paint( g ); // chama a versão da superclasse do método paint
37.
38.     // desenha um retângulo iniciando em (15, 10) que tem 270
39.     // pixels de largura e 20 pixels de altura
40.     g.drawRect( 15, 10, 270, 20 );
41.
42.     // desenha os resultados como uma String em (25, 25)
43.     g.drawString( "O valor da soma é " + sum, 25, 25 );
44.  } // fim do método paint
45.  } // fim da classe AdditionApplet
```

Figura 10.13 – Applet Java que manipula dados fornecidos pelo usuário.

O método `init` inicializa o applet e solicita que o usuário forneça dois valores de tipo `real` (linhas 19 e 23). A soma desses dois valores é realizada na linha 30. A o fim da execução do método `init`, o método `paint` é executado e desenha um retângulo na tela (linha 40). Em seguida o valor da soma é desenhado em forma de string dentro desse retângulo (linha 43).

10.3.1 Executando *applets* em um Browser

Para que seja possível executar os applets, faz-se necessário que estes sejam incorporados a páginas HTML, que são lidas por *browsers* de navegação na Web (e.g., *Internet Explorer, Mozilla Firefox, Google Chrome,* etc.).

Uma página em formato HTML pode ser criada e editada com editores simples de texto, como por exemplo, o *Notepad* do sistema operacional Windows. Uma página HTML simples que permite a execução de um applet como os vistos anteriormente, deve possuir o modelo ilustrado abaixo:

```
<html>
<applet code = "<nome_do_applet>.class"
        width = "<valor inteiro>" height = "<valor inteiro>" >
</applet>
</html>
```

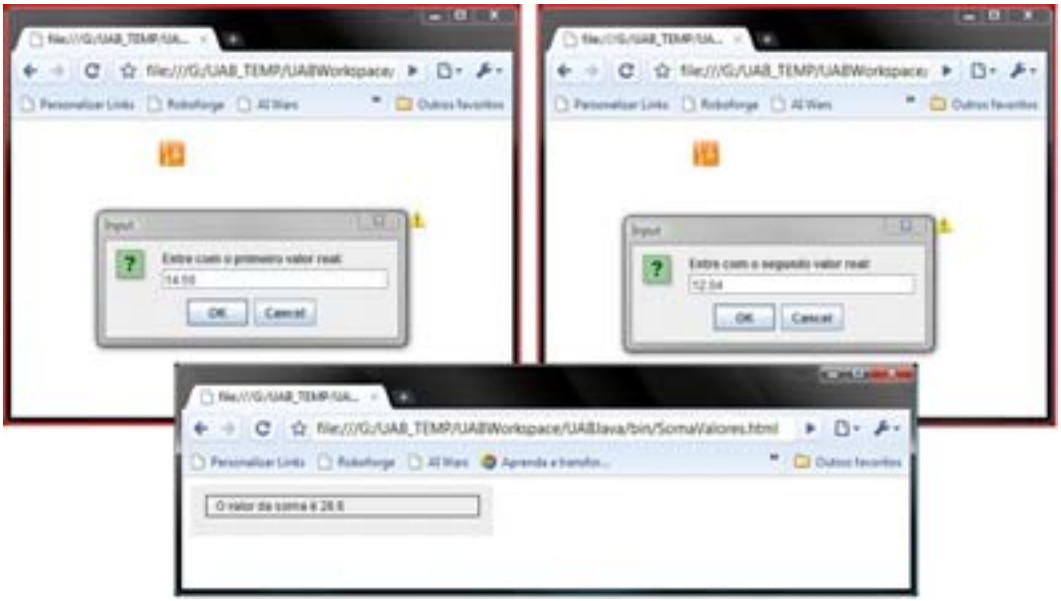
onde, as tags `<html>` e `</html>` delimitam o documento, o atributo `code` da tag `<applet ...>` indica o *bytecode* do applet a ser executado, `width` indica a largura do applet e `height` indica a altura do applet.

As figuras 10.14 e 10.15 mostram a execução dos applets de boas vindas e

soma de valores em páginas Web acessadas por um browser. Os códigos HTML que incorporam os applets também estão ilustrados na figura.


```
<html>
    <applet code = "BoasVindas.class"
            width = "500" height = "45">
    </applet>
</html>
```

Figura 10.14 – Execução do applet de boas vindas em um browser e código HTML que incorpora o applet.



```
<html>
    <applet code = "SomaValores.class"
            width = "300" height = "50">
    </applet>
</html>
```

Figura 10.15 – Execução do applet de soma de valores em um browser e código HTML.

Resumo

- Programas de computador podem possuir interface mais amigáveis ou menos amigáveis.
- Dizemos que um programa possui interface amigável quando apresenta recursos que possibilitam ao usuário o entendimento das funcionalidades do programa e facilidade em seu manuseio.
- Uma Interface Gráfica com o Usuário (GUI) é um componente gráfico que possibilita interfaces amigáveis em programas através da exibição de janelas, botões, caixas de texto, barras de rolagem, etc.
- A linguagem Java possui classes que facilitam a codificação de programas com essas características.
- Caixas de diálogos com `JOptionPane` permitem entrada de dados de forma visual pelo usuário e exibição de tela de resultados.
- A classe `JFileChooser` permite a exibição de janelas para gerenciamento de arquivos e diretórios pelo usuário.
- Uma das maiores razões do sucesso e popularidade da linguagem Java ocorreu em virtude da possibilidade de execução de programas diretamente em páginas Web, de forma bastante facilitada. Programas desse tipo são chamados de *Applets* Java.
- Para possibilitar a execução de um applet Java em uma página Web acessada por um *browser*, o código HTML da respectiva página deve incorporar o *bytecode* do applet em questão. A tag `<applet ...>` e seu conjunto de atributos possui esse fim.

Referências e Sugestões de Leitura

A página da Web **`java.sun.com/applets`** traz vários recursos de applets Java disponíveis, incluindo applets gratuitos para fazer o download e utilizar em seu próprio site Web.

A página **`http://www.walter-fendt.de/ph11br/`** traz vários applets Java ilustrativos de problemas de Física.

Para se aprofundar no desenvolvimento de interfaces gráficas em Java, recomendo a leitura dos capítulos 11, 12, 21 e 22 de

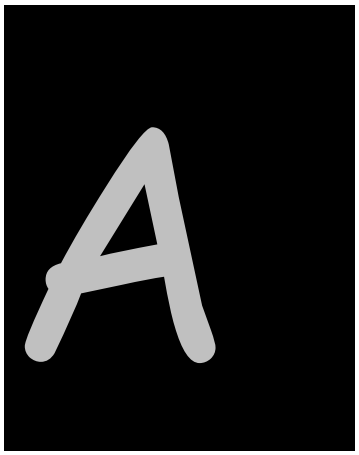
DEITEL, H., DEITEL, P. Java como Programar, Prentice-Hall, 2005.

Exercícios Propostos

10.1 – Modifique a forma de entrada de dados feita pelo usuário de alguns dos exercícios propostos em capítulos anteriores. Utilize JOptionPane para criar caixas de diálogos para este fim.

10.2 – Reimplemente os exercícios propostos no capítulo 9, utilizando `JFileChooser` quando for adequado.

10.3 – Exercite o uso de interface gráfica com Java, adaptando alguns dos exercícios propostos nos capítulos anteriores para que sejam executados como applets em uma página Web, ao invés de aplicações Java.



Código para Leitura de Dados do Teclado

OBS: Esta classe precisa ser importada em todos os programas Java que fazem uso de leitura de dados do usuário a partir do teclado.

```
import java.io.IOException;
import java.lang.StringBuffer;
import java.lang.NumberFormatException;
/** * Classe que fornece métodos para leitura de dados do teclado */
public class System_in {
    private static String keyboardReadInt () {
            int in = 0;
            char chr;
            boolean sinal = false;
            StringBuffer Valor = new StringBuffer("");
            do {
               try { in = System.in.read();
                  chr = (char) in;
                  if ((in != 10) & (in != 13)) {
                        if (in >= 48 && in <= 57 || (in == 45 && !sinal)){
                                    Valor.append(chr); sinal = true;
                        }
                  }
               }
               catch (IOException e) {}
            } while (in != 10);
            return Valor.toString();
    }

    private static String keyboardReadFloat () {
            int in = 0;
            char chr;
            boolean ponto = false;
            boolean sinal = false;
            StringBuffer Valor = new StringBuffer("");
            do {
              try { in = System.in.read();
                 chr = (char) in;
                 if ((in != 10) & (in != 13)) {
                   if (in >= 48 && in <= 57 || in == 46 || ( in == 45 && !sinal )) {
                       if ( in == 46 ) { if ( ! ponto ) {
                          Valor.append(chr); ponto = true;
                       }
                   } else {
                        Valor.append(chr); } sinal = true;
                   }
                 }
```

```
            }
            catch (IOException e) {}
         } while (in != 10);
         return Valor.toString();
    }

    /** * Lê um int do teclado * @result int lido */
    public static int readInt () {
          int retorno;
          try { retorno = Integer.parseInt(keyboardReadInt());
          }
          catch (NumberFormatException e) { retorno = 0; }
          return retorno;
    }

    /** * Lêr um float do teclado * @result float lido */
          public static float readFloat () {
             float retorno;
             try { retorno = Float.parseFloat(keyboardReadFloat()); }
             catch (NumberFormatException e) { retorno = 0; }
             return retorno;
          }

          /** * Lêr um char do teclado * @result char lido */
          public static char readChar () {
              int in = 0;
              char chr;
              int cont = 0;
              StringBuffer Valor = new StringBuffer("");
              do {
                       try { in = System.in.read();
                                  chr = (char) in;
                                  if ((in != 10) & (in != 13)) {
                                             if ( cont == 0 ) {
                                                        Valor.append(chr);
                                             }
                                             cont++;
                                   }
                        } catch (IOException e) {}
              } while (in != 10);
              return Valor.charAt(0);
           }

           /** * Lêr um String do teclado * @result String lido */
           public static String readString () {
                       int in = 0;
                       char chr;
                       StringBuffer Valor = new StringBuffer("");
                       do {
                                  try {
                                             in = System.in.read();
                                             chr = (char) in;
                                             if ((in != 10) & (in != 13)) {
                                                        Valor.append(chr);
                                                        }
                                             }
                                  catch (IOException e) {}
                        } while (in != 10);
                       return Valor.toString();
          }
}
```