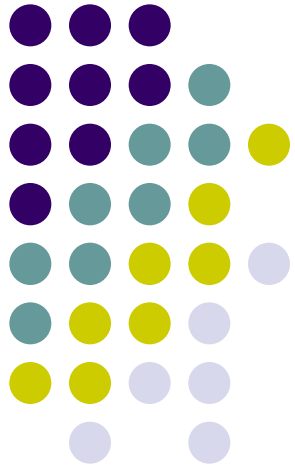
COM220

Aula 17: Interface Gráfica Containers Swing

Prof. Laércio Baldochi

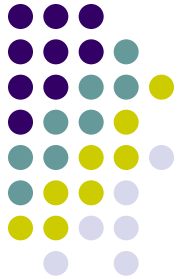


Containers Swing

- Janelas
  - JFrame
- Painéis
  - JPanel
- Applets
  - JApplet

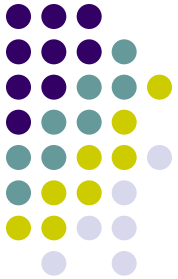
- Caixas de diálogo
  - JDialog
  - JOptionPane

JFrame



- Componente principal para a construção de interfaces gráficas em aplicações Swing
  - Componente "pesado"
  - Pode possuir
    - Bordas
    - Título
    - Menu
    - Botões de controle
      - Fechar, minimizar, etc.

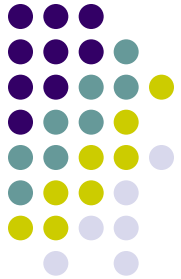
JFrame



- Vejamos como criar um JFrame e inserir elementos nele

```
// Cria a janela com o título
 JFrame frame = new JFrame("FrameDemo");
 // Define a ação a ser executada quando a janela for fechada
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 // Cria o componente e o adiciona ao container
 JLabel emptyLabel = new JLabel();
 frame.add(emptyLabel);
 // Define o tamanho da janela
 frame.setSize(185, 150);
 // Exibe a janela
 frame.setVisible(true);
```

JFrame

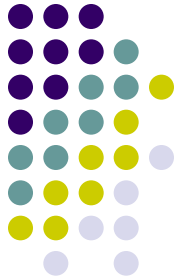


frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- Quando um JFrame é fechado, ele somente torna-se invisível
  - Continua executando (não desaloca recursos)

- Método setDefaultCloseOperation permite configurar o comportamento da janela
  - EXIT_ON_CLOSE
  - HIDE_ON_CLOSE
  - DISPOSE_ON_CLOSE
  - DO_NOTHING_ON_CLOSE

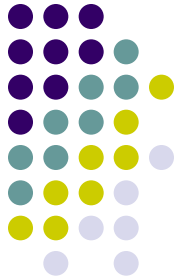
Painel



- Componente gráfico para uso geral
  - Agrupa de forma lógica outros componentes
  - Gerencia layout de uma área
  - Classe JPanel

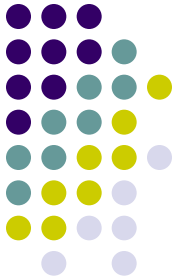
- Na criação de um painel
  - Deve-se especificar um gerenciador de layout
    - setLayout(layoutManager)
  - Pode-se especificar as características de sua borda
    - Classe BorderFactory
    - setBorder(border)

JPanel



- Pode-se adicionar objetos diretamente a um JPanel
  - Método -> add(component)
    - Pode ter um parâmetro opcional contendo a posição do componente dentro do layout escolhido

JPanel

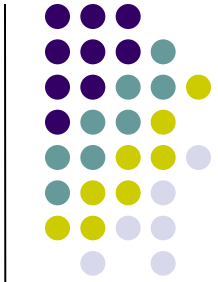


```
// Cria os painéis
JPanel p1 = new JPanel(), p2 = new JPanel(), p3 = new JPanel();
JPanel p4 = new JPanel(), p5 = new JPanel(), p6 = new JPanel();

// Seta diferentes tipos de bordas para os painéis
p1.setBorder(BorderFactory.createTitledBorder("Título"));
p2.setBorder(BorderFactory.createEtchedBorder());
p3.setBorder(BorderFactory.createLineBorder(Color.BLACK));
p4.setBorder(BorderFactory.createRaisedBevelBorder());
p5.setBorder(BorderFactory.createLoweredBevelBorder());
p6.setBorder(BorderFactory.createMatteBorder(5,5,5,5,Color.BLACK));

// Cria um painel principal, seta o layout e a borda
JPanel p = new JPanel();
p.setLayout(new GridLayout(2, 3, 5, 5));
p.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

// Adiciona os painéis ao painel principal
p.add(p1); p.add(p2); p.add(p3);
p.add(p4); p.add(p5); p.add(p6);
```

Caixas de Diálogo

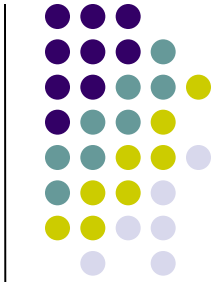
- Mais limitadas que os frames
  - Normalmente utilizadas para solicitar entradas do usuário e exibir mensagens
  - JDialog
    - O processo de criação de um diálogo envolve desde a construção dos componentes e gerenciamento do layout até a implementação dos listeners

JDialog



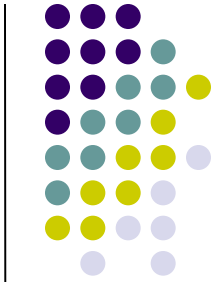
```
class SimpleDialog extends JDialog implements ActionListener {

   JFrame parent;
   JTextField field;
   JButton setButton;
   SimpleDialog(JFrame parent, String title) {
      // Chama o construtor da superclasse com os parâmetros:
      // janela-pai e título
      super(parent, title, true);
      // Cria e ajusta os componentes
      ...
      // Inicializa o diálogo com o tamanho preferido
      pack();
   }
   public void actionPerformed(ActionEvent event) {
      Object source = event.getSource();
      if (source == setButton)
         this.setTitle(field.getText());
   }
}

public static void main(String[] args) {
   // Código que cria uma caixa de diálogo e a torna visível
   JFrame parent = new JFrame();
   SimpleDialog dialog = new SimpleDialog(parent, "A Simple Dialog");
}
```

JDialog



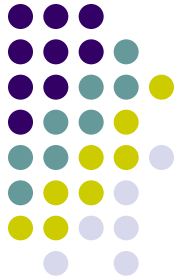
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SimpleDialog extends JDialog implements ActionListener {

    JFrame parent;
    JTextField textField;
    JButton setButton;

    SimpleDialog(JFrame parent, String title) {
        super(parent, title, true);
        JLabel label = new JLabel("Entre o texto aqui: ");
        textField = new JTextField(25);
        setButton = new JButton("Ajusta título");
        JPanel p = new JPanel();
        p.setLayout(new FlowLayout());
        p.add(label);
        p.add(textField);
        p.add(setButton);
        this.add(p);
        setButton.addActionListener(this);   //Adicionando listener ao botao
        this.pack();
        this.setVisible(true);
    }
```

JDialog



```
 public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        if (source == setButton) {
           this.setTitle(textField.getText());
        }
    }

    public static void main(String[] args) {
        // Código que cria uma caixa de diálogo e a torna visível
        JFrame parent = new JFrame();
        SimpleDialog dialog = new SimpleDialog(parent, "A Simple Dialog");
    }
}
```

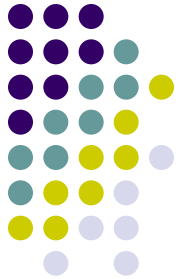
JOptionPane



- Fornece um conjunto de métodos para a criação automática de caixas de diálogo padronizadas
  - Solicitar entrada do usuário
  - Exibir mensagem

Nome do método	Descrição
showConfirmDialog	Faz uma pergunta e solicita a confirmação do usuário
showInputDialog	Solicita alguma entrada para o usuário
showMessageDialog	Informa o usuário sobre algo
showOptionDialog	Apresenta ao usuário um conjunto de opções e pede que ele escolha uma

JOptionPane

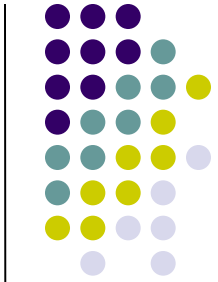


- É possível definir propriedades
  - Ícone a ser mostrado
  - Tipo da mensagem
  - Botões que irão aparecer na tela
- Possíveis valores
  - ERROR_MESSAGE
  - INFORMATION_MESSAGE
  - WARNING_MESSAGE
  - QUESTION_MESSAGE
  - PLAIN_MESSAGE

JOptionPane

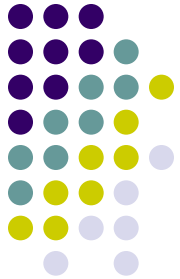
```
Object[] options = { "Option 1", "Option 2",   "Option 3" };
 JOptionPane.showInputDialog(
     this,      // Componente-pai (define a posição relativa)
     "Message", // A mensagem a ser apresentada
     "Title",   // O título da caixa de diálogo
     JOptionPane.QUESTION_MESSAGE, // O tipo da mensagem
     null,      // O ícone a ser mostrado
     options,   // A lista de opções
     options[1] // A opção inicial
 );
```

JOptionPane - Exemplo



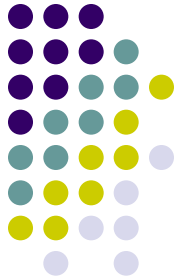
```
import javax.swing.JOptionPane;

public class Adicao
{
   public static void main( String args[] )
   {
      //  obtém a entrada de usuário a partir dos diálogos de entrada
      String firstNumber =
         JOptionPane.showInputDialog( "Digite o primeiro numero" );
      String secondNumber =
          JOptionPane.showInputDialog( "Digite o segundo numero" );

      // converte String em valores int para utilização em um cálculo
      int number1 = Integer.parseInt( firstNumber );
      int number2 = Integer.parseInt( secondNumber );

      int sum = number1 + number2; // soma os números

      // exibe o resultado em um diálogo de mensagem JOptionPane
      JOptionPane.showMessageDialog( null, "A soma dos numeros eh " + sum,
         "Soma de dois inteiros", JOptionPane.PLAIN_MESSAGE );
   } // fim do método main
} // fim da classe Adicao
```

Exercício 1



- Faça um programa para jogar par ou ímpar com o computador
  - Usar JOptionPane
  - Computador gera um nro aleatório primeiro
  - Jogador pede par ou ímpar na primeira janela
  - Jogador digita sua jogada na segunda janela
  - A terceira janela deve mostrar o valor do somatório e o ganhador
  - int r = (int) (Math.**random**() * 10);
    - Gera um nro randômico entre 0 e 99

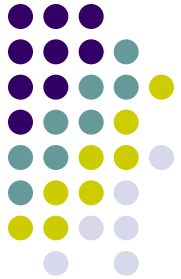
Exercício 1



- Dica – use

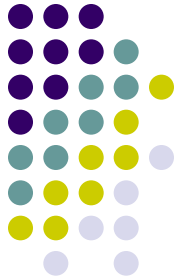
```
Object[] options = { "PAR", "IMPAR" };
opUser = JOptionPane.showOptionDialog(null,
                                      "Par ou Ímpar?",
                                      "Jogo do para ou ímpar",
                                      JOptionPane.DEFAULT_OPTION,
                                      JOptionPane.QUESTION_MESSAGE,
                                      null,
                                      options,
                                      null);
```

Eventos de janela



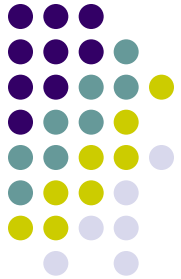
- Quando o usuário manipula uma janela, essa ação gera eventos de janela
  - Os ouvintes (listeners) desse tipo de evento são registrados com o método addWindowListener
- Interface associada ->WindowListener
  - windowActivated
  - windowClosed
  - windowClosing
  - windowDeactivated
  - windowIconified
  - windowDeiconified
  - windowOpened

Eventos de janela



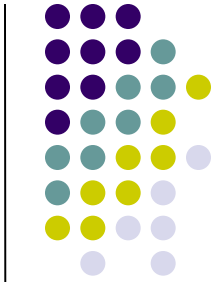
- windowActivated
  - Quando uma janela se torna ativa
- windowClosed
  - Depois que uma janela é fechada
- windowClosing
  - Quando o usuário tenta fechar uma janela
- windowDeactivated
  - Quando a janela ativa passa para o background ou é minimizada
- windowIconified
  - Quando uma janela é minimizada
- windowDeiconified
  - Quando uma janela minimizada é maximizada
- windowOpened
  - Quando a janela é exibida na tela pela primeira vez

Exercício 2



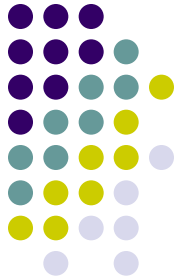
- Crie um JFrame que exibe uma caixa de diálogo quando é fechado, solicitando confirmação do usuário para sair da aplicação. A caixa deve ter duas opções: *sim* e *não*, sendo que se o usuário escolher *sim*, a aplicação é fechada.

Applets Java



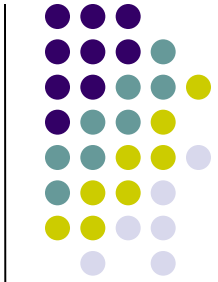
- Aplicação Java que pode ser embutida em páginas HTML
  - Executa no browser
- JApplet
  - Subclasse da classe Applet (AWT)
- Deve sobrescrever os métodos init(), start(), stop() e destroy()
- Não necessita do método main

Applets Java
Métodos



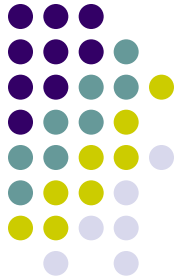
- init()
  - chamado quando o *applet* é carregado na página
    - utilizado para construir a interface gráfica, ler parâmetros e alocar demais recursos
- stop()
  - invocado quando o usuário sai da página do *applet* ou quando o *browser* é fechado
    - o *applet* pára de executar, mas permanece carregado;
- start()
  - invocado quando o *applet* é carregado no início (após o init) ou quando o usuário retorna à pagina com o *applet*
    - o *applet* (re)inicia a sua execução;
- destroy()
  - chamado quando o *browser* é fechado
    - o *applet* é destruído e seus recursos são desalocados.

Applets Java Exemplos



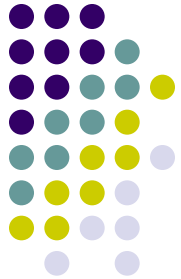
```
// Um applet deve estender a classe JApplet
public class MyApplet extends JApplet {
   private int i = 0;
   // Inicializa o applet
   public void init() {
      System.out.println(++i + ". Initializing");
   }
   // Começa a executar o applet
   public void start() {
      System.out.println(++i + ". Starting");
   }
   // Pára de executar o applet
   public void stop() {
      System.out.println(++i + ". Stopping");
   }
   // Destrói o applet
   public void destroy() {
      System.out.println(++i + ". Destroying");
   }
}
```

Inserindo Applets em páginas HTML



- Documento HTML deve utilizar a tag <APPLET> e configurar os seguintes atributos
  - CODE: indica o arquivo que contém a classe que estende a JApplet
  - CODEBASE: especifica a URL base do *applet*
    - o diretório que contém o arquivo especificado no atributo CODE
  - WIDTH e HEIGHT: definem a largura e a altura do *applet*

Passando parâmetros para applets



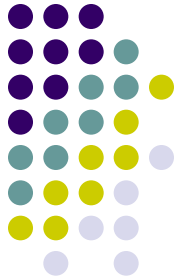
- É possível passar parâmetros da página HTML para o applet usando a tag <PARAM>
  - Atrituto NAME -> nome do parâmetro
  - Atributo VALUE -> valor do parâmetro

- Applet lê os parâmetros utilizando o método
  - getParameter(name)

Exemplo de utilização da tag APPLET

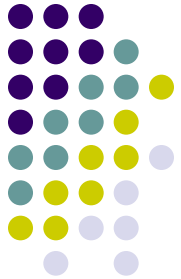
```
<APPLET CODE="MyApplet.class"
        CODEBASE="example/"
        WIDTH="300"
        HEIGHT="150">
    <PARAM NAME="userId" value="123">
 </APPLET>
```

Escrevendo em applets com a classe Graphics



- Classe Graphics é usada para enviar informação ao applet
- Provê métodos para "desenhar" palavras e figuras
  - drawImage()
  - drawPolygon()
  - drawString()
  - ...

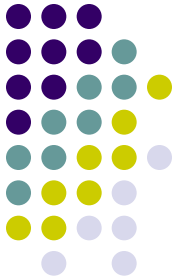
Escrevendo em applets com a classe Graphics



- Conteúdo é desenhado no applet utilizando o método paint()
- Exemplo

```
public void paint (Graphics g){
    g.drawString("Hello World", 50, 25);
}
```

Exercício 3



- Crie um *applet* que lê um nome de usuário a partir de um parâmetro do HTML e o escreve em um painel central.