

Técnico em Informática

**Estruturas de Dados**

**Interface Gráfica**

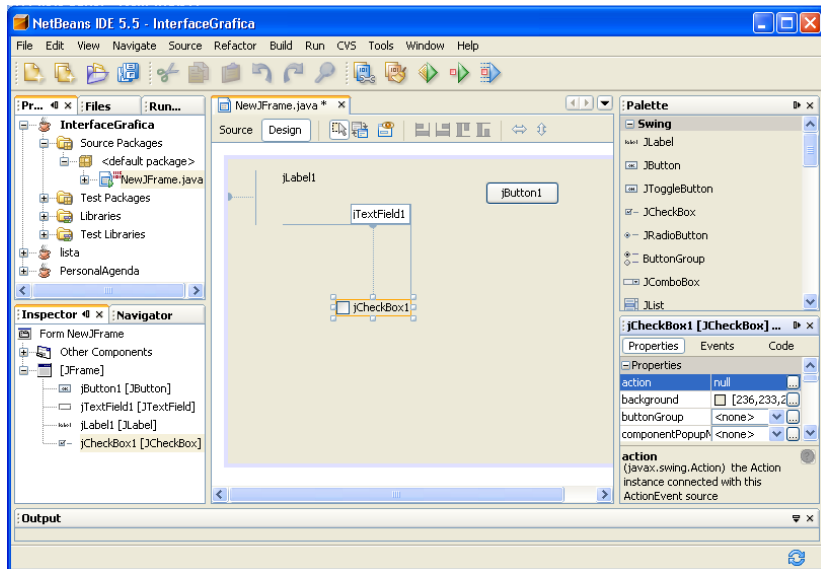
Introdução

Alex Helder Cordeiro do Rosário de Oliveira

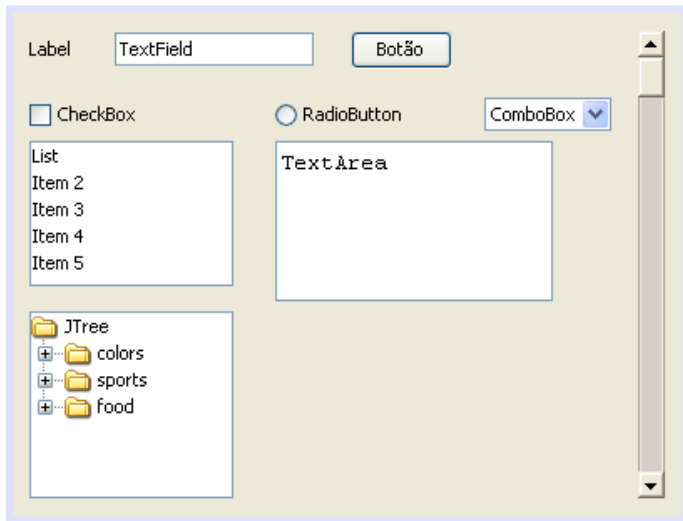
Instituto Federal de Brasília - *Campus* Brasília

1º semestre de 2017

# Desenvolvimento via IDE



# Componentes comuns de Swing e AWT\*

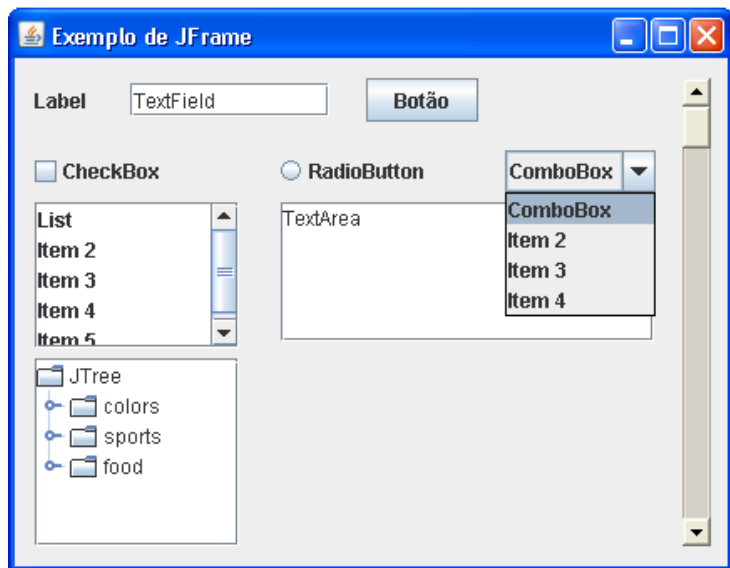


\* Abstract Window Toolkit.

# Componente JFrame

- É a janela onde se colocam os outros componentes.
- Normalmente criada através de herança do JFrame.

# Componente JFrame



# JFrame - Código Básico

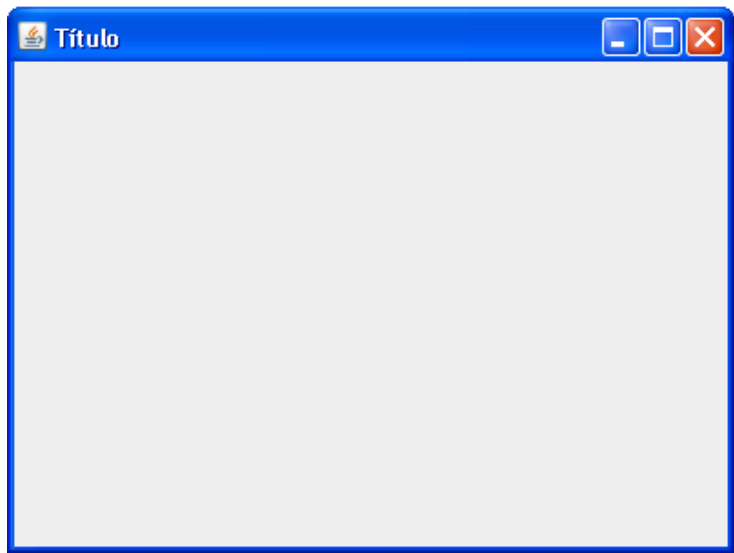
```
import javax.swing.*;

public class MainFrame extends JFrame {
    public MainFrame() {
        super("Título");
    }
    public static void main(String[] args) {
        MainFrame frame = new MainFrame();
        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

---

\*Exemplo encontrado no arquivo MainFrame.java.

# JFrame - Código Básico



# JFrame

- É interessante que não colocamos nenhum comando avançado de desenho para que fosse desenhada a janela na tela.
- Onde estão os códigos complexos para fazer o desenho da janela?



# JFrame

- É interessante que não colocamos nenhum comando avançado de desenho para que fosse desenhada a janela na tela.
- Onde estão os códigos complexos para fazer o desenho da janela?
- Essa “magia” se deve à **herança**;
- Uma vez que minha classe herdou do JFrame, ela tem todos os comandos que permitem o seu desenho na tela e eu não preciso reescrever este código (nem usar *Ctrl+C*, *Ctrl+V*) para que ele funcione na minha classe.
- Isto é o que chamamos de **Reuso de Código**.

# JFrame

- Olhando para o corpo do método `main()`, é interessante observarmos duas coisas:
  - Uma vez que o método `main()` é estático\*, ele existe independente do objeto da classe.
  - Enquanto não houver um comando explícito para construir o objeto da classe, ele não será construído.
  - Para botarmos um objeto da nossa janela na tela, temos primeiro de contruí-lo.
  - Por este motivo, temos o comando:

```
MainFrame frame = new MainFrame();
```

---

\*Declarado com o modificador `static`.

# JFrame

- A outra observação:
  - Usamos dois métodos que não foram declarados na nossa classe (`setSize()` e `setVisible()`).
  - Onde eles estão declarados e implementados?

---

\*Exceto membros privados.

# JFrame

- A outra observação:
  - Usamos dois métodos que não foram declarados na nossa classe (`setSize()` e `setVisible()`).
  - Onde eles estão declarados e implementados?
  - Eles foram herdados da classe `JFrame`.
  - Tudo que existe na superclasse também existem em suas subclasses.\*
  - Desta forma, podemos referenciar todos os métodos que existem na superclasse, que neste caso é o `JFrame`.

---

\*Exceto membros privados.

# Inserção de Componentes

- Pode-se criar objetos dos componentes desejados.

```
JLabel rotulo = new JLabel("Escreva o texto:");
```

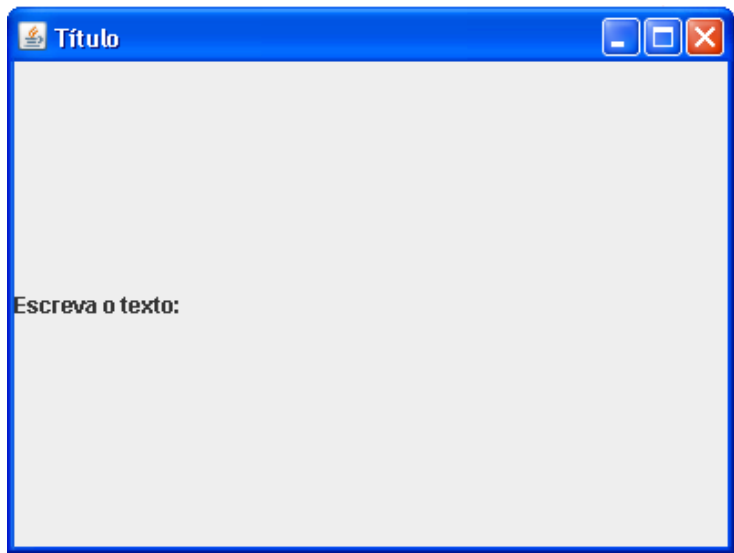
- A inserção na janela ocorre através do método add().

```
add(rotulo);
```

---

\*Exemplo encontrado no arquivo `MainFrameComRotulo.java`

# Inserção de Componentes



# Inserção de Componentes

- Onde, no código, devo colocar a inserção do componente?

---

\*Isto é uma composição: A janela é composta pelos componentes; A classe faz uma composição com os componentes que devem aparecer na janela.

# Inserção de Componentes

- Onde, no código, devo colocar a inserção do componente?
- É extremamente recomendável que os componentes que formam a janela sejam atributos da classe.\*

---

\*Isto é uma composição: A janela é composta pelos componentes; A classe faz uma composição com os componentes que devem aparecer na janela.



# Inserção de Componentes

- Onde, no código, devo colocar a inserção do componente?
- É extremamente recomendável que os componentes que formam a janela sejam atributos da classe.\*
- A inicialização é recomendável que ocorra antes da janela ser colocada na tela.
- Um mecanismo interessante é se tivermos como garantir que os componentes que compõe a janela sejam construídos durante a construção da janela.
- A construção da janela ocorre através da execução de seu método construtor.
- Então é interessante criar o objeto do componente e executar o `add()` dentro do construtor.
- Quem é o construtor?

---

\*Isto é uma composição: A janela é composta pelos componentes; A classe faz uma composição com os componentes que devem aparecer na janela.

# Inserção de Componentes

- A inicialização é recomendável que ocorra antes da janela ser colocada na tela.
- Um mecanismo interessante é se tivermos como garantir que os componentes que compõe a janela sejam construídos durante a construção da janela.
- A construção da janela ocorre através da execução de seu método construtor.
- Então é interessante criar o objeto do componente e executar o `add()` dentro do construtor.
- Quem é o construtor?

```
public MainFrame() {
```

- É um método com mesmo nome da classe e sem retorno.
- Veremos mais detalhes mais à frente, em um momento oportuno.

# Inserção de Componentes


- Faça um teste: Acrescente mais componentes ao JFrame.

# Inserção de Componentes - Layout

- Para se inserir diversos componentes sem problemas, deve ser selecionado um layout;
- Os Layouts são formas de se organizar os componentes dentro de um “contêiner”<sup>\*</sup>.
- Os Layouts serão estudados em breve.
- Temporariamente, utilizaremos a o `FlowLayout`.  
`setLayout(new FlowLayout());`

---

<sup>\*</sup>Contêiner são componentes de interface gráfica onde podem ser inseridos outros componentes de interface gráfica.

<sup>†</sup>Exemplo encontrado no arquivo `MainFrameComDiversosComponentes.java`. 

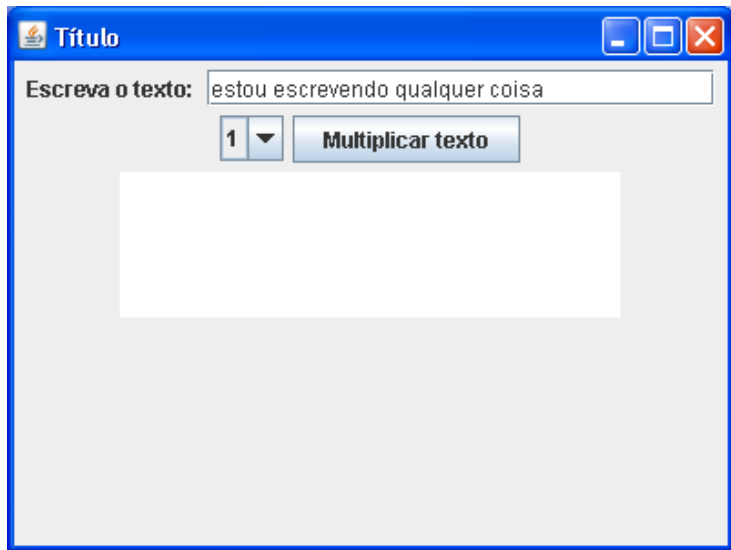
# Inserção de Componentes

- Alguns componentes podem ser instanciados sem argumento;
- Caso típico de JTextField e JTextArea;
- Se for o caso, utilize os métodos `.setColumns()` e `.setRows()` para que eles apareçam no programa.
- Componentes cuja informação principal é um conjunto de informações podem receber estas informações através do método `.addItem()`;
- É o caso do JComboBox.

# Inserção de Componentes

```
JLabel label = new JLabel("Escreva o texto: ");  
JTextField textField = new JTextField();  
JComboBox comboBox = new JComboBox();  
JButton button = new JButton("Multiplicar texto");  
JTextArea textArea = new JTextArea();  
setLayout(new FlowLayout());  
add(label);  
textField.setColumns(25);  
add(textField);  
comboBox.addItem("1");  
comboBox.addItem("2");  
add(comboBox);  
add(button);  
textArea.setColumns(25);  
textArea.setRows(5);  
add(textArea);
```

# Inserção de Componentes



# Fechei a janela?

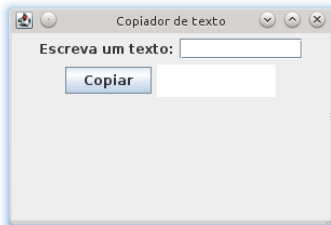
- Cliquei no botão fechar (canto superior direito da janela);
- A janela desapareceu;
- Processo continua rodando... O que aconteceu?



# Fechei a janela?

- Cliquei no botão fechar (canto superior direito da janela);
- A janela desapareceu;
- Processo continua rodando... O que aconteceu?
- Não foi associado nenhum comando ao botão de fechar. O comportamento da janela gráfica (pelo padrão do sistema operacional) será: retirar a janela da tela do computador.
- Para associar também o comando de encerramento do programa, use o comando:  
`setDefaultCloseOperation(EXIT_ON_CLOSE);`

# Eventos dos botões

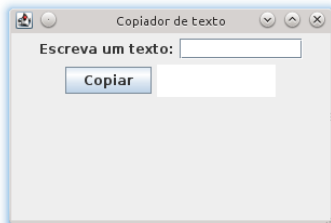


- “Ok. Coloquei os componentes e o programa ficou bonito.”

---

\*Exemplo encontrado no arquivo `EventosSemEvento.java`.

# Eventos dos botões



- “Ok. Coloquei os componentes e o programa ficou bonito.”
- “Mas não está exatamente útil.”
- “Eu precisaria que, ao clicar em um botão ele fizesse algo, mas ao clicar ele não está fazendo nada.”

---

\*Exemplo encontrado no arquivo `EventosSemEvento.java`.

# Eventos dos botões

- Um clique de um botão é um evento que só será percebido se mandamos “*alguém*” prestar atenção nele;
- Chamamos este “*alguém*” de Listener\*.

---

\*Ouvinte

## Eventos dos botões

- Um clique de um botão é um evento que só será percebido se mandamos “*alguém*” prestar atenção nele;
- Chamamos este “*alguém*” de Listener\*.
- Como quase tudo em Java, o nosso ouvinte também é um objeto.
- Este objeto tem que ter um método a ser executado quando o evento for identificado.

---

\*Ouvinte

# Eventos dos botões

- Um clique de um botão é um evento que só será percebido se mandamos “*alguém*” prestar atenção nele;
- Chamamos este “*alguém*” de Listener\*.
- Como quase tudo em Java, o nosso ouvinte também é um objeto.
- Este objeto tem que ter um método a ser executado quando o evento for identificado.
- Então basta escrevermos a classe deste objeto implementando o método específico.

---

\*Ouvinte

# Eventos dos botões

- Um clique de um botão é um evento que só será percebido se mandamos “*alguém*” prestar atenção nele;
- Chamamos este “*alguém*” de Listener\*.
- Como quase tudo em Java, o nosso ouvinte também é um objeto.
- Este objeto tem que ter um método a ser executado quando o evento for identificado.
- Então basta escrevermos a classe deste objeto implementando o método específico.
- Só temos as seguintes questões:
  - ❶ “Como o botão saberá qual o método que implementa a ação do evento?”
  - ❷ “Como garantir que minha classe ouvinte vai satisfazer a necessidade do evento?”

---

\*Ouvinte

# Eventos dos botões - Interfaces

- Interfaces são componentes de software que definem a visão que o mundo externo terá de um determinado grupo de classes;
- Elas contém as assinaturas dos métodos públicos que devem constar nas classes que as implementem;
- Interfaces nunca contém métodos implementados, apenas suas assinaturas.
- Elas funcionam como um contrato entre a classe e o sistema: Quando uma classe implementa uma interface, ela está se comprometendo a fornecer o comportamento publicado pela interface;



## Eventos dos botões - Interfaces

- Os desenvolvedores do Swing e do AWT não tinham como prever o que o programador vai querer que os ouvintes façam, então não podiam escrever as classes dos ouvintes;
- Entretanto eles precisavam que as classes ouvintes tivessem determinados métodos que seriam chamados.
- Portanto eles criaram as interfaces dos `Listeners` para especificar quais seriam estes métodos;
- E vincularam que os ouvintes teriam de implementar estas interfaces.

## Eventos dos botões

- Para que uma classe implemente uma interface, devemos usar a palavra-chave `implements` seguido do nome da interface que estamos implementando.
- A interface que é associada a eventos de botões é a `ActionListener`.
- Para a classe `Ouvinte` poder implementar a interface `ActionListener`, ela precisa implementar o método declarado na interface: `public void actionPerformed(ActionEvent ae)`

```
class Ouvinte implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        // Ação desejada.  
    }  
}
```


## Eventos dos botões

- Uma vez que temos uma classe ouvinte para monitorar o clique do botão, é necessário que criemos um objeto dela e o vinculemos a um botão.
- Para isso, usamos o método `addActionListener()` do botão.

```
Ouvinte ouvinte = new Ouvinte();  
botao.addActionListener(ouvinte);
```

- Agora nossas janelas já são um pouco mais úteis.

---

\*Exemplo encontrado no arquivo `EventosAcaoLimitada.java`. 

## Eventos dos botões

- Problema: *“Queria que a ação do botão usasse e alterasse informações de e em componentes da minha janela. :-)”*.
- Precisaríamos que minha classe ouvinte tivesse o mesmo tipo de relacionamento com os atributos da minha janela da mesma forma que os membros da janela.

# Eventos dos botões

- Problema: *“Queria que a ação do botão usasse e alterasse informações de e em componentes da minha janela. :-)”*.
- Precisaríamos que minha classe ouvinte tivesse o mesmo tipo de relacionamento com os atributos da minha janela da mesma forma que os membros da janela.
- Então porque não transformamos a classe ouvinte em um membro de minha janela.
- Como?

## Eventos dos botões

- Os atributos e métodos que são membros de uma classe são definidos dentro da classe.
- Então se quero que uma classe seja membro de outra é só declará-la e implementá-la dentro desta outra classe.

```
class Janela extends JFrame {  
    class Ouvinte implements ActionListener {  
    }  
}
```

- A este tipo de classe, damos o nome de **classe interna**.

---

\*Exemplo encontrado no arquivo `EventosClasseInterna.java`.

# Eventos dos botões

- É interessante verificar outro motivo para que os componentes da janela sejam atributos da janela:
- Se eles não forem atributos, as classes internas não terão acesso direto a eles;
- Assim as ações dos botões seriam muito complexas para poder executar ações com relação a valores dos componentes da janela.

# Eventos dos botões

- Para evitar o excesso de código\*, podemos usar um outro tipo de classe: a **classe anônima**.

```
botao.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent ae) {  
    }  
});
```

---

\*Especialmente quando se tratando de quando temos muitos botões, ou muitos eventos diferentes.


\*Exemplo encontrado no arquivo `EventosClasseAnonima.java`.



## Eventos dos botões

- Veja que a criação do objeto se faz direto no argumento do método `addActionListener()`;
- Veja também que estamos fazendo um `new` de uma interface e não de uma classe;
- Por isso temos de implementar os métodos da interface neste momento;
- Se instanciamos um objeto, temos uma classe; mas não demos nome para ela, por isso ela é **anônima**.

---

\*Exemplo encontrado no arquivo `EventosClasseAnonima.java`. 

# Sua Vez...

**(1.0 ponto) - Questão 19:** Escreva um programa composto por um frame que contenha, pelo menos um campo de texto, um botão e um rótulo. Quando o botão for clicado, o programa deve copiar o conteúdo presente no campo de texto para o rótulo.